Lab 5: Plotting Tools

Statistical Computing, 36-350

Week of Monday September 24, 2018

Name:
Andrew ID:
Collaborated with:

This lab is to be done in class (completed outside of class if need be). You can collaborate with your classmates, but you must identify their names above, and you must submit **your own** lab as an knitted HTML file on Canvas, by Sunday 11:59pm, this week.

This week's agenda: getting familiar with basic plotting tools; understanding the way layers work; recalling basic text manipulations; producing histograms and overlaid histograms; heatmaps.

Fastest 100m sprint times

Below, we read in a data set of the fastest times ever recorded for the 100m sprint, in men's track. (Usain Bolt may have slowed down now ... but he was truly one of a kind!) We also read in a data set of the fastest times ever recorded for the 100m, in women's track. Both of these data sets were scraped from http://www.alltime-athletics.com/m_100ok.htm (we scraped it in spring 2018; this website may have been updated since).

```
sprint.m.dat = read.table(
  file="http://www.stat.cmu.edu/~ryantibs/statcomp/data/sprint.m.dat",
  sep="\t", quote="", header=TRUE)
sprint.w.dat = read.table(
  file="http://www.stat.cmu.edu/~ryantibs/statcomp/data/sprint.w.dat",
  sep="\t", quote="", header=TRUE)
```

Data frame and apply practice

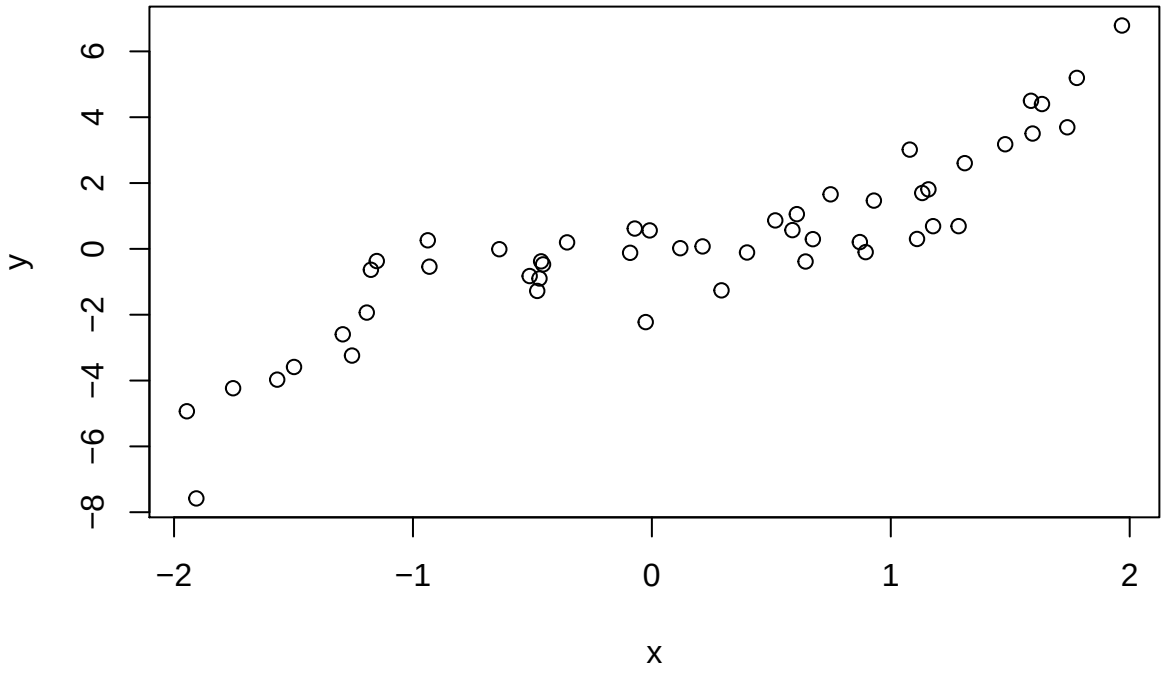
- **1a.** Confirm that both `sprint.m.dat` and `sprint.w.dat` are data frames. Delete the `Rank` and `City` columns from each data frame. Then display the first and last 5 rows of each. **Challenge**: compute the ranks for the men's data set from the `Time` column and add them back as a `Rank` column to `sprint.m.dat`. Do the same for the women's data set.
- **1b.** Using `table()`, compute for each unique country in the `Country` column of `sprint.m.dat`, the number of sprint times from this country that appear in the data set. Call the result `sprint.m.counts`. Do the same for the women, calling the result `sprint.w.counts`. What are the 5 most represented countries, for the men, and for the women? (Interesting side note: go look up the population of Jamaica, compared to that of the US. Pretty impressive, eh?)
- **1c.** Are there any countries that are represented by women but not by men, and if so, what are they? Vice versa, represented by men and not women? Hint: you will want to use the `%in%` operator. If you're sure what it does you can read the documentation.
- **1d.** Using some method for data frame subsetting, and then `table()`, recompute the counts of countries in `sprint.m.dat`, but now only counting sprint times that are faster than or equal to 10 seconds. Call the result `sprint.m.10.counts`. Recompute counts for women too, now only counting sprint times

that are faster than or equal to 11 seconds, and call the result `sprint.w.11.counts`. What are the 5 most represented countries now, for men, and for women?

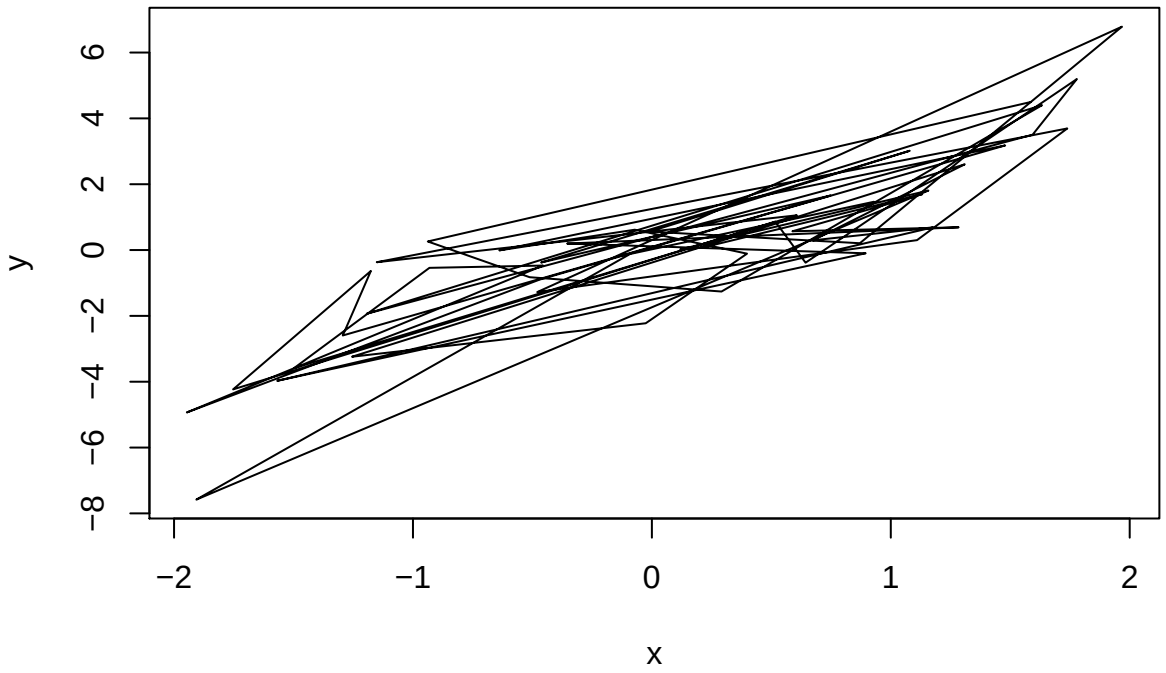
Plot basics

- **2a.** Below is some code that is very similar to that from the lecture, but with one key difference. Explain: why does the `plot()` result with with `type="p"` look normal, but the `plot()` result with `type="l"` look abnormal, having crossing lines? Then modify the code below (hint: modify the definition of `x`), so that the lines on the second plot do not cross.

```
n = 50
set.seed(0)
x = runif(n, min=-2, max=2)
y = x^3 + rnorm(n)
plot(x, y, type="p")
```

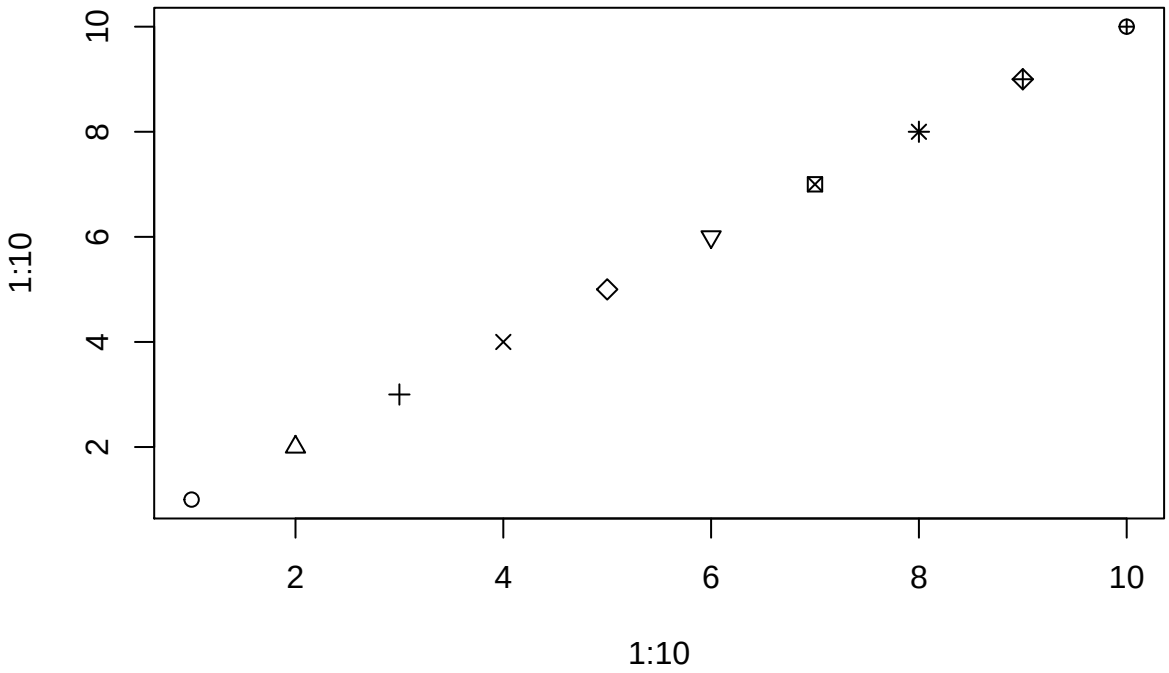



```
plot(x, y, type="l")
```



- **2b.** The `cex` argument can used to shrink or expand the size of the points that are drawn. Its default value is 1 (no shrinking or expansion). Values between 0 and 1 will shrink points, and values larger than 1 will expand points. Plot `y` versus `x`, first with `cex` equal to 0.5 and then 2 (so, two separate plots). Give titles "Shrunken points", and "Expanded points", to the plots, respectively.
- **2c.** The `xlim` and `ylim` arugments can be used to change the limits on the x-axis and y-axis, repsectively. Each argument takes a vector of length 2, as in `xlim = c(-1, 0)`, to set the x limit to be from -1 to 0. Plot `y` versus `x`, with the x limit set to be from -1 to 1, and the y limit set to be from -5 to 5. Assign x and y labels "Trimmed x" and "Trimmed y", respectively.
- **2d.** Again plot `y` versus `x`, only showing points whose x values are between -1 and 1. But this time, define `x.trimmed` to be the subset of `x` between -1 and 1, and define `y.trimmed` to be the corresponding subset of `y`. Then plot `y.trimmed` versus `x.trimmed` without setting `xlim` and `ylim`: now you should see that the y limit is (automatically) set as "tight" as possible. Hint: use logical indexing to define `x.trimmed`, `y.trimmed`.
- **2e.** The `pch` argument, recall, controls the point type in the display. In the lecture examples, we set it to a single number. But it can also be a vector of numbers, with one entry per point in the plot. So, e.g.,

```
plot(1:10, 1:10, pch=1:10)
```



displays the first 10 point types. If `pch` is a vector whose length is shorter than the total number of points to be plotted, then its entries are recycled, as appropriate. Plot `y` versus `x`, with the point type alternating in between an empty circle and a filled circle.

- **2f.** The `col` argument, recall, controls the color the points in the display. It operates similar to `pch`, in the sense that it can be a vector, and if the length of this vector is shorter than the total number of points, then it is recycled appropriately. Plot `y` versus `x`, and repeat the following pattern for the displayed points: a black empty circle, a blue filled circle, a black empty circle, a red filled circle.

Adding to plots

- **3a.** Produce a scatter plot of `y` versus `x`, and set the title and axes labels as you see fit. Then overlay on top a scatter plot of `y2` versus `x2`, using the `points()` function, where `x2` and `y2` are as defined below. In the call to `points()`, set the `pch` and `col` arguments appropriately so that the overlaid points are drawn as filled blue circles.

```
x2 = sort(runif(n, min=-2, max=2))
y2 = x^2 + rnorm(n)
```

- **3b.** Starting with your solution code from the last question, overlay a line plot of `y2` versus `x2` on top of the plot (which contains empty black circles of `y` versus `x`, and filled blue circles of `y2` versus `x2`), using the `lines()` function. In the call to `lines()`, set the `col` and `lwd` arguments so that the line is drawn in red, with twice the normal thickness. Look carefully at your resulting plot. Does the red line pass overtop of or underneath the blue filled circles? What do you conclude about the way R *layers* these additions to your plot?

- **3c.** Starting with your solution code from the last question, add a legend to the bottom right corner of the the plot using `legend()`. The legend should display the text: "Cubic" and "Quadratic", with corresponding symbols: an empty black circle and a filled blue circle, respectively. Hint: it will help to look at the documentation for `legend()`.

- **3d.** Produce a plot of `y` versus `x`, but with a gray rectangle displayed underneath the points, which runs has a lower left corner at `c(-2, qnorm(0.1))`, and an upper right corner at `c(2, qnorm(0.9))`.

Hint: use `rect()` and consult its documentation. Also, remember how layers work; call `plot()`, with `type="n"` or `col="white"` in order to refrain from drawing any points in the first place, then call `rect()`, then call `points()`.

- **Challenge.** Produce a plot of `y` versus `x`, but with a gray tube displayed underneath the points. Specifically, this tube should fill in the space between the two curves defined by $y = x^3 \pm q$, where q is the 90th percentile of the standard normal distribution (i.e., equal to `qnorm(0.90)`). Hint: use `polygon()` and consult its documentation; this function requires that the x coordinates of the polygon be passed in an appropriate order; you might find it useful to use `c(x, rev(x))` for the x coordinates. Lastly, add a legend to the bottom right corner of the plot, with the text: "Data", "Confidence band", and corresponding symbols: an empty circle, a very thick gray line, respectively.

Text manipulations, and layered plots

- **4a.** Back to the sprinters data set: define `sprint.m.times` to be the `Time` column of `sprint.m.dat`. Define `sprint.m.dates` to be the `Date` column of `sprint.m.dat`, converted into a character vector. Define a character vector `sprint.m.years` to contain the last 4 characters of an entry of `sprint.m.dates`. Hint: use `substr()`. Finally, convert `sprint.m.years` into a numeric vector. Display its first 10 entries.
- **4b.** Plot `sprint.m.times` versus `sprint.m.years`. For the point type, use small, filled black circles. Label the x-axis "Year" and the y-axis "Time (seconds)". Title the plot "Fastest men's 100m sprint times". Using `abline()`, draw a dashed blue horizontal line at 10 seconds. Using `text()`, draw below this line, in text on the plot, the string "N men", replacing "N" here by the number of men who have run under 10 seconds. Your code should programmatically determine the correct number here, and use `paste()` to form the string. Comment on what you see visually, as per the sprint times across the years. What does the trend look like for the fastest time in any given year?
- **4c.** Reproduce the previous plot, but this time, draw a light blue rectangle underneath all of the points below the 10 second mark. The rectangle should span the entire region of the plot below the horizontal line at $y = 10$. And not only the points of sprint times, but the blue dashed line, and the text "N men" (with "N" replaced by the appropriate number) should appear *on top* of the rectangle. Hint: use `rect()` and layering as appropriate.
- **4d.** Repeat Q4a but for the women's sprint data, arriving at vectors `sprint.w.times` and `sprint.w.years`. Then repeat Q4c for this data, but with the 10 second cutoff being replaced by 11 seconds, the rectangle colored pink, and the dashed line colored red. Comment on the differences between this plot for the women and your plot for the men, from Q4c. In particular, is there any apparent difference in the trend for the fastest sprint time in any given year?

More text manipulations, and histograms

- **5a.** Extract the birth years of the sprinters from the data frame `sprint.m.dat`. To do so, define `sprint.m.bdates` to be the `Birthdate` column of `sprint.m.dat`, converted into a character vector. Then define a character vector `sprint.m.byears` to contain the last 2 characters of each entry of `sprint.m.bdates`. Convert `sprint.m.byears` into a numeric vector, add 1900 to each entry, and redefine `sprint.m.byears` to be the result. Finally, compute a vector `sprint.m.ages` containing the age (in years) of each sprinter when their sprint time was recorded. Hint: use `sprint.m.byears` and `sprint.m.years`.
- **5b.** Repeat the last question, but now for the data `sprint.w.dat`, arriving at a vector of ages called `sprint.w.ages`.
- **5c.** Using one of the apply functions, compute the average sprint time for each age in `sprint.m.ages`, calling the result `time.m.avg.by.age`. Similarly, compute the analogous quantity for the women,

calling the result `time.w.avg.by.age`. Are there any ages for which the men's average time is faster than 10 seconds, and if so, which ones? Are there any ages for which the women's average time is faster than 10.98 seconds, and if so, which ones?

- **5d.** Plot a histogram of `sprint.m.ages`, with break locations occuring at every age in between 17 and 40. Color the histogram to your liking; label the x-axis, and title the histogram appropriately. What is the mode, i.e., the most common age? Also, describe what you see around the mode: do we see more sprinters who are younger, or older?
- **Challenge.** Plot a histogram of `sprint.m.ages`, now with `probability=TRUE` (so it is on the probability scale, rather than raw frequency scale). Overlay a histogram of `sprint.w.ages`, also with `probability=TRUE`. Set the break locations so that the plot captures the full range of the very youngest to the very oldest sprinter present among both men and women. Your code should determine these limits programmatically. Choose colors of your liking, but use transparency as appropriate so that the shapes of both histograms are visible; label the x-axis, and title the histogram appropriately. Add a legend to the histogram, identifying the histogram bars from the men and women. Compare, roughly, the shapes of the two histograms: is there a difference between the age distributions of the world's fastest men and fastest women?

Maungawhau volcano and heatmaps

- **6a.** The `volcano` object in R is a matrix of dimension 87 x 61. It is a digitized version of a topographic map of the Maungawhau volcano in Auckland, New Zealand. Plot a heatmap of the volcano using `image()`, with 25 colors from the terrain color palette.
- **6b.** Each row of `volcano` corresponds to a grid line running east to west. Each column of `volcano` corresponds to a grid line running south to north. Define a matrix `volcano.rev` by reversing the order of the rows, as well as the order of the columns, of `volcano`. Therefore, each row `volcano.rev` should now correspond to a grid line running west to east, and each column of `volcano.rev` a grid line running north to south.
- **6c.** If we printed out the matrix `volcano.rev` to the console, then the elements would follow proper geographic order: left to right means west to east, and top to bottom means north to south. Now, produce a heatmap of the volcano that follows the same geographic order. Hint: recall that the `image()` function rotates a matrix 90 degrees counterclockwise before displaying it; and recall the function `clockwise90()` from the lecture, which you can copy and paste into your code here. Label the x-axis "West –> East", and the y-axis "South –> North". Title the plot "Heatmap of Maungawhau volcano".
- **6d.** Reproduce the previous plot, and now draw contour lines on top of the heatmap.
- **Challenge.** The function `filled.contour()` provides an alternative way to create a heatmap with contour lines on top. It uses the same orientation as `image()` when plotting a matrix. Use `filled.contour()` to plot a heatmap of the volcano, with (light) contour lines automatically included. Make sure the orientation of the plot matches proper geographic orientation, as in the previous question. Use a color scale of your choosing, and label the axes and title the plot appropriately. It will help to consult the documentation for `filled.contour()`.