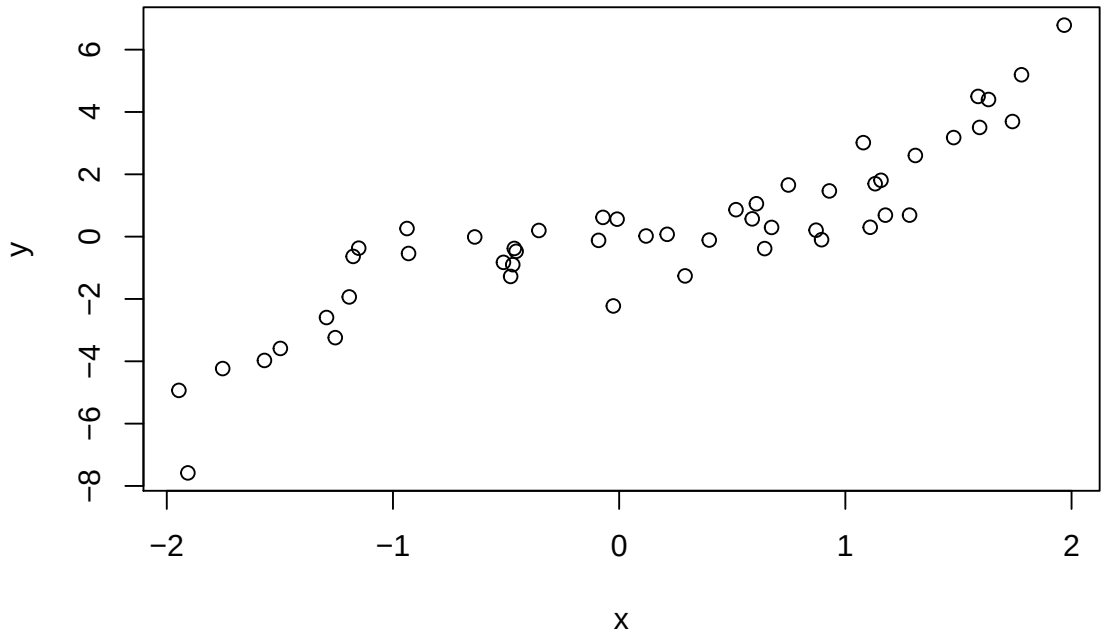
Homework 3

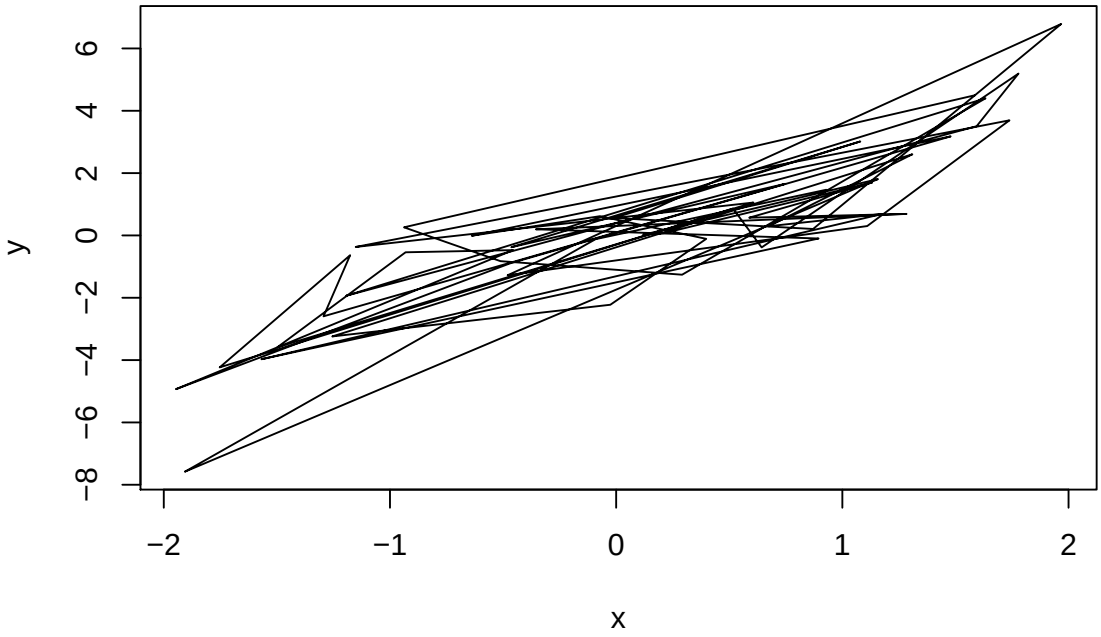
Plot basics

- **1a.** 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")
```

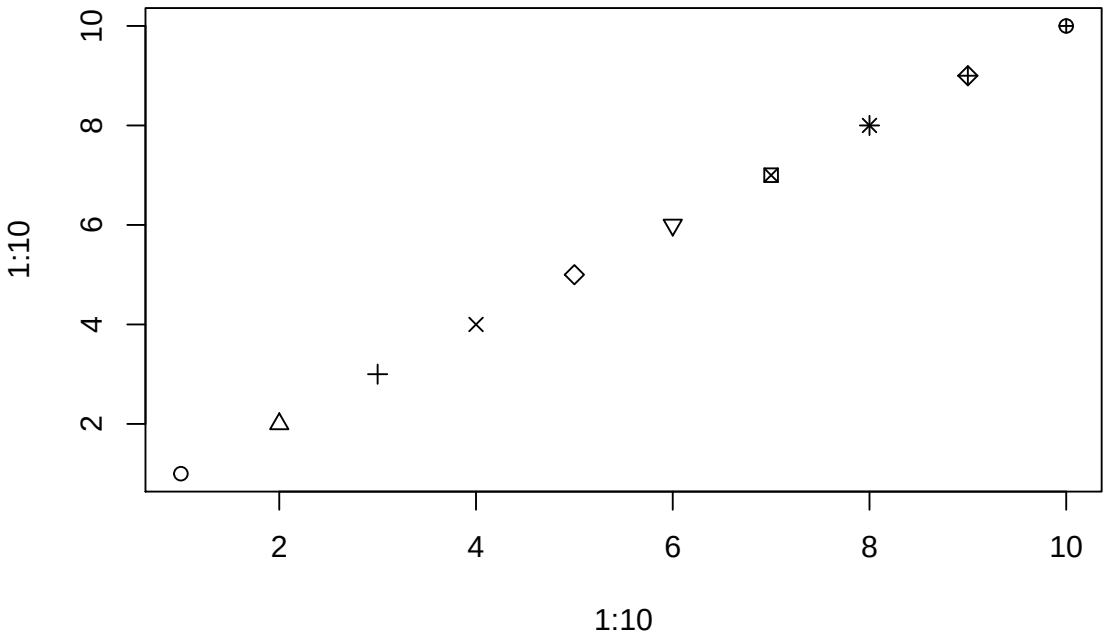



```
plot(x, y, type="l")
```



- **1b.** 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.
- **1c.** 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.
- **1d.** 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`.
- **1e.** 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.

- **1f.** 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

- **2a.** 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)
```

- **2b.** 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?

- **2c.** 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()`.

- **2d.** 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()`.
- **3e** 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.

Maungawhau volcano and heatmaps

- **3a.** 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.
- **3b.** 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.
- **3c.** 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".
- **3d.** Reproduce the previous plot, and now draw contour lines on top of the heatmap.
- **3e.** 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()`.

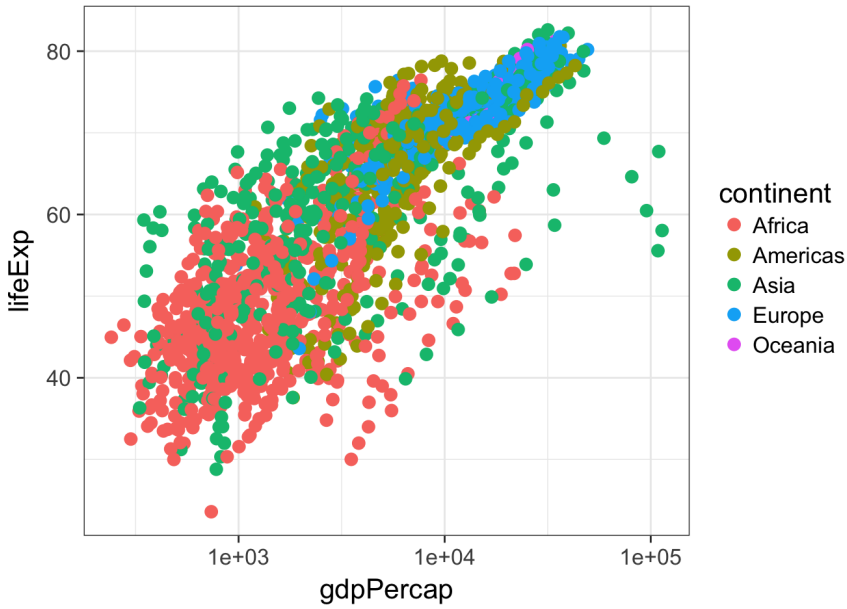
ggplot

Read the `gapminder-FiveYearData.csv` via the following code:

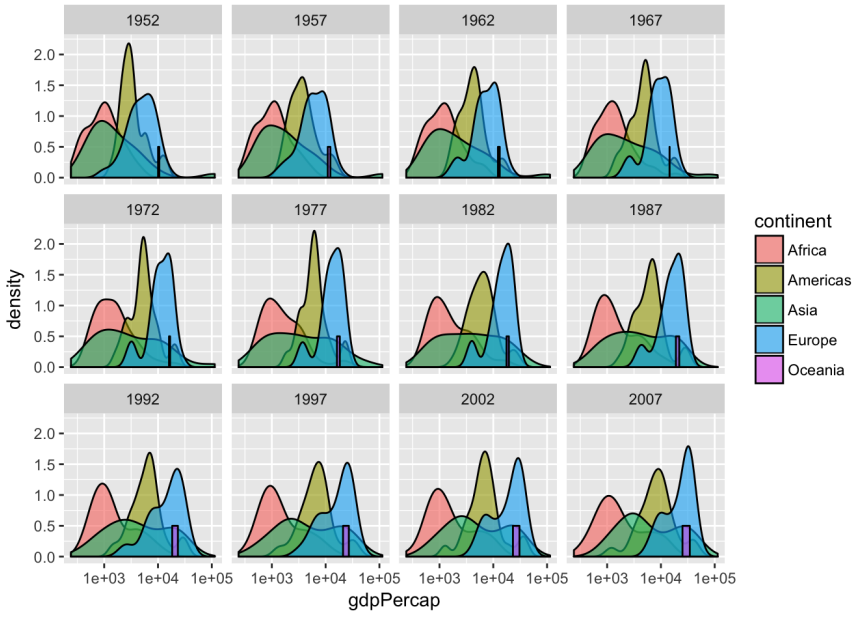
```
gm <- read.csv("gapminder-FiveYearData.csv", header = T)
```

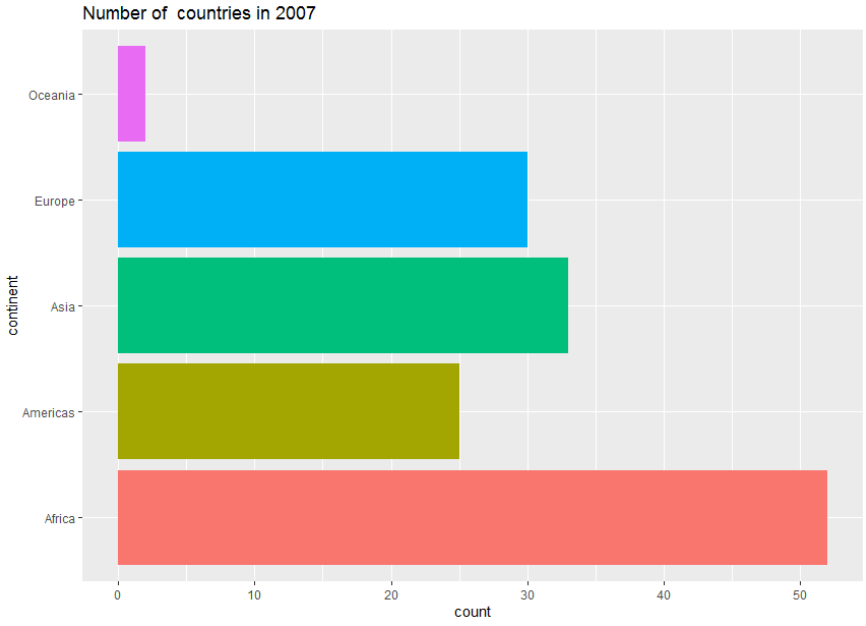
- **4a.** Re-create the following plot

That is, start from the `ggplot` call, use the `gm` data. Map `gdpPercap` to the x-axis and `lifeExp` to the y-axis and `continent` onto the aesthetics. Add points to the plot with points size 3. Use a log10 scale for the x-axis.



- **4b.** Make a scatter plot of `lifeExp` on the y-axis against `year` on the x faceting on `continent`. Add a fitted curve, `smooth` or `lm`on each facet. Set the title to "Figure 2", X label to "Year", Y Label to "Life expectancy", and Legend title to "Continent".
- **4c.** Using `geom_line()` and and aesthetic mapping `country` to `group=`, make a "spaghetti plot", showing semitransparent lines connected for each country, faceted by continent. Add a smoothed loess curve with a thick (lwd=3) line with no standard error stripe. Reduce the opacity (`alpha=`) of the individual black lines. Don't show Oceania countries (that is, filter the data where continent!="Oceania" before you plot it).
- **4d.** Make a jittered strip plot of `gdpPercap` against `continent`.
- **4e.** Make a box plot of `gdpPercap` against `continent`.
- **4f.** Using a log10 y-axis scale, overlay semitransparent jittered points on top of box plots, where outlying points are colored.
- **4g.** Create a density plot of GDP per capita, filled by continent that looks like the following: NOTE: Transform the x axis using a log10 to better visualise the data spread. - Add a facet layer to panel the density plots by year.
- **4h.** Consider the data with `gm$year==2007`, create a horizontal bar chart of `continent` like the following plot


1952
1957
1962
1967
1972
1977
1982
1987
1992
1997
2002
2007
density
gdpPercap
1e+03
1e+04
1e+05
continent
Africa
Americas
Asia
Europe
Oceania


Number of countries in 2007
Oceania
Europe
Asia
Americas
Africa
continent
0
10
20
30
40
50
count