01 Exploratory Data Analysis and Introduction to R

David Gerard

2018-12-07

Learning Objectives

- Gain a basic grasp of R
- Understand common graphical and numerical summaries.
- Section 1.5 in *Statistical Sleuth*.

Motivation

Motivation

In this class, you will be introduced to the R statistical language, a programming language designed to analyze data. There are lots of point-and-click statistical programs out there, so why learn R?

Motivation

1. It's free.
    - You will always have access to R.
    - Not true for other statistical softwares (SPSS, STATA, SAS).

Motivation

1. It's free.
    - You will always have access to R.
    - Not true for other statistical softwares (SPSS, STATA, SAS).
2. It's widely used.
    - If you need to do some special analysis, someone has probably already made an R package for it.
    - Just use Google to check.

Motivation

1. It's free.
    - You will always have access to R.
    - Not true for other statistical softwares (SPSS, STATA, SAS).
2. It's widely used.
    - If you need to do some special analysis, someone has probably already made an R package for it.
    - Just use Google to check.
3. It's easy (especially graphics).

Motivation

1. It's free.
   - You will always have access to R.
   - Not true for other statistical softwares (SPSS, STATA, SAS).
2. It's widely used.
   - If you need to do some special analysis, someone has probably already made an R package for it.
   - Just use Google to check.
3. It's easy (especially graphics).
4. It makes reproducible research easy.
   - When part of the pipeline is copying and pasting excel spreadsheets, people make mistakes.
   - E.g. an excel mistake led countries to adopt austerity measures to increase economic growth.
   - In R, you can automate your analysis, reducing the chance for mistakes and making your analysis transparent to the wider research community.

Basic R

The most important function i

I cannot teach you everything there is to know in R. When you know the name of a function, but don't know the commands, use the `help` function. For example, to learn more about `sum` type

```
help(sum)
```

Alternatively, if you do not know the name of the function, you can Google the functionality you want. Googling coding solutions is a lot of what real programmers do.

Example Basic R commands

- When you first invoke R, it opens the R workspace with an open R Console.
- RStudio is similar but in addition to the Console, RStudio provides two other views, the Environment window, which gives a listing of any variables created in your R workspace, and the Files windows, which shows your working directory.

Example Basic R commands

- When you first invoke R, it opens the R workspace with an open R Console.
- RStudio is similar but in addition to the Console, RStudio provides two other views, the Environment window, which gives a listing of any variables created in your R workspace, and the Files windows, which shows your working directory.
- Commands are entered at the command prompt, $>$, and results are displayed in the same window.
- Try the following commands by typing them directly into the Console window at the command prompt.
- Note that anything following # is a comment, which is ignored by R.

Basic R Commands Continued i

```
2 + 4
```

```
## [1] 6
```

<- is the "assignment" operator.

```
x <- 20
```

For the most part, you can also just use the = sign. But R experts tend to prefer <-.

```
y = 82
x + y
```

```
## [1] 102
```

Basic R Commands Continued ii

```
x - y ## This is a comment
```

```
## [1] -62
```

```
x * y
```

```
## [1] 1640
```

```
x / y
```

```
## [1] 0.2439
```

```
exp(x) # exp is the built in exponential function
```

```
## [1] 485165195
```

Basic R Commands Continued iii

```r
sin(x) # similarly for sine
```

```
## [1] 0.9129
```

```r
log(y) # and the logarithm functions
```

```
## [1] 4.407
```

```r
z <- "a string variable"
z
```

```
## [1] "a string variable"
```

Basic R Commands Continued iv

Notice that the second, third, and 2nd to last lines do not produce any output. These lines are storing the values of 20, 82, and "a string variable" in the variables `x`, `y`, and `z`, respectively.

R Tip: Type your commands in a "script" file so that you can save them for later reference. In RStudio, go to the File menu and select New File and then R Script. This will open an empty editor window within RStudio. You can type the commands in the editor and run them by hitting CTRL+ENTER in RStudio (CTRL+R in plain R).

Running Code i

1. Place your cursor on the line you want to run and either press CTRL+ENTER (using the Windows version of RStudio, it is Command+Enter on a Mac) or click on the Run command in the script source code window:
2. Highlight all of the lines you want to run (by left clicking on them with the mouse) and either press CTRL+ENTER or click on the Run command in the script source code window.

Note that the code stops running when there is an error. Errors (and other information) are given in the Console window in red.

Entering Data Manually i

```r
# The c() function concatenates the list of
# values into a vector
x <- c(1,2,8,10,18,23,36)

x # displays the contents of x
```

```
## [1]  1  2  8 10 18 23 36
```

```r
x[3] # displays the 3rd element of x
```

```
## [1] 8
```

```r
x[1:3] # displays the first 3 elements of x
```

```
## [1] 1 2 8
```

Entering Data Manually ii

```r
x[c(2,4)] # displays the 2nd and 4th elements of x
```

```
## [1]  2 10
```

```r
# create a new variable which
# is a function of x
y <- 36 * x - x ^ 2

# puts x and y together for easier
# viewing in a matrix
cbind(x, y)
```

Entering Data Manually iii

```
##       x   y
## [1,]  1  35
## [2,]  2  68
## [3,]  8 224
## [4,] 10 260
## [5,] 18 324
## [6,] 23 299
## [7,] 36   0
```

```r
# puts them together for easier
# viewing in a data frame
data.frame(x,y)
```

Entering Data Manually iv

```
##    x   y
## 1  1  35
## 2  2  68
## 3  8 224
## 4 10 260
## 5 18 324
## 6 23 299
## 7 36   0
```

Echoing

Notice that each line you run is “echoed” (i.e. shows up) in the Console window as text along with any resulting output the command generates. If there is no output, then it will only show the command.

Set Working Directory i

1. Using menu navigation:

Session > Set Working Directory > To Source File Location

2. Using code:

```
setwd("~/Dropbox/teaching/stat_514/514_notes/01_introductio
```

Note that you will need to edit the location ("~/Dropbox/") to match where your data are located on your specific computer. In the Anderson Labs, data is typically saved to the Desktop or in a local user file directory.

Tip: I tend to use code so that I and others can completely reproduce an analysis.

Entering Data from Files

- Most datasets in this class will either be `.rdata` or `.csv` or `.txt` files.
- Open `.rdata` files with `load()`
- Open `.csv` files with `read.csv()`
- Open `.txt` files with `read.table()`

Blood Alcohol Data

```
bloodalc <- read.csv("../data/BLOODALC.csv")
```

Entering Data from Packages

- Some R packages come with datasets.
- *The Statistical Sleuth* has its own R package with all of the case studies.
- You first load the package using `library()`, then access the data using `data()`.
- You can see a list of all the datasets (and functions) available in a package with `library(help = "packagename")`, where "packagename" is the name of the package you want to explore.

Sleuth

```
install.packages("Sleuth3")
```

```
library(Sleuth3)
```

```
library(help = "Sleuth3")
```

```
data("case0101")
head(case0101)
```

```
##   Score Treatment
## 1   5.0 Extrinsic
## 2   5.4 Extrinsic
## 3   6.1 Extrinsic
## 4  10.9 Extrinsic
## 5  11.8 Extrinsic
## 6  12.0 Extrinsic
```

Graphical Summaries

We will be using the `qplot()` function from the ggplot2 R package for plotting in this course. If you do not have `ggplot2` installed on your computer already, you can do so now with the following code:

```
install.packages("ggplot2")
```

You should only need to install ggplot2 on your local computer once.

After installing ggplot2, you can load it in R using the `library()` function. *You'll need to reload ggplot2 every time you start up R.*

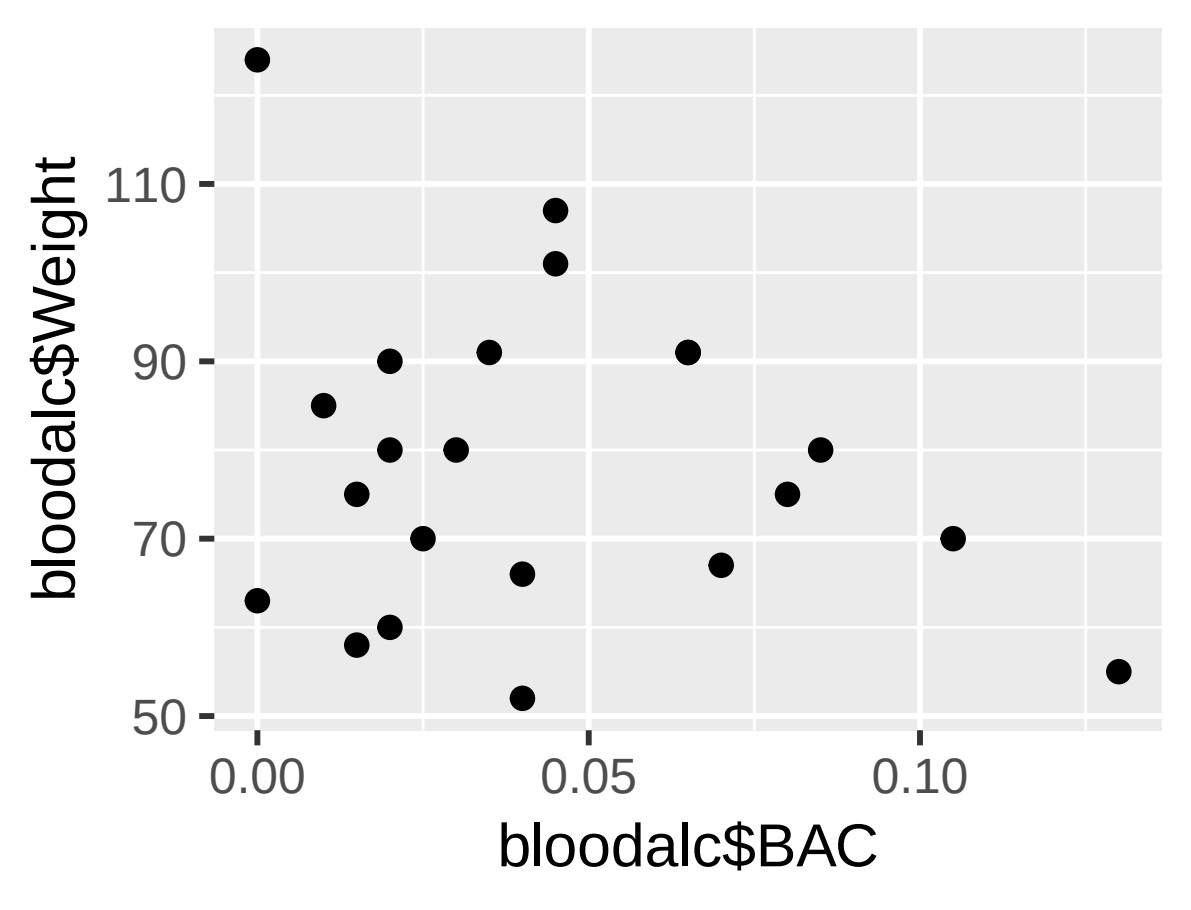
```
library(ggplot2)
```

Scatterplot

- A scatterplot has an **explanatory variable** on the x-axis and a **response variable** on the y-axis.
- We think an explanatory variable either explains or causes the response variable. This is either because of scientific knowledge, because the explanatory variable occurred before the response, or we are investigating the effect of the explanatory on the response.
- Each point represents one **observational unit**

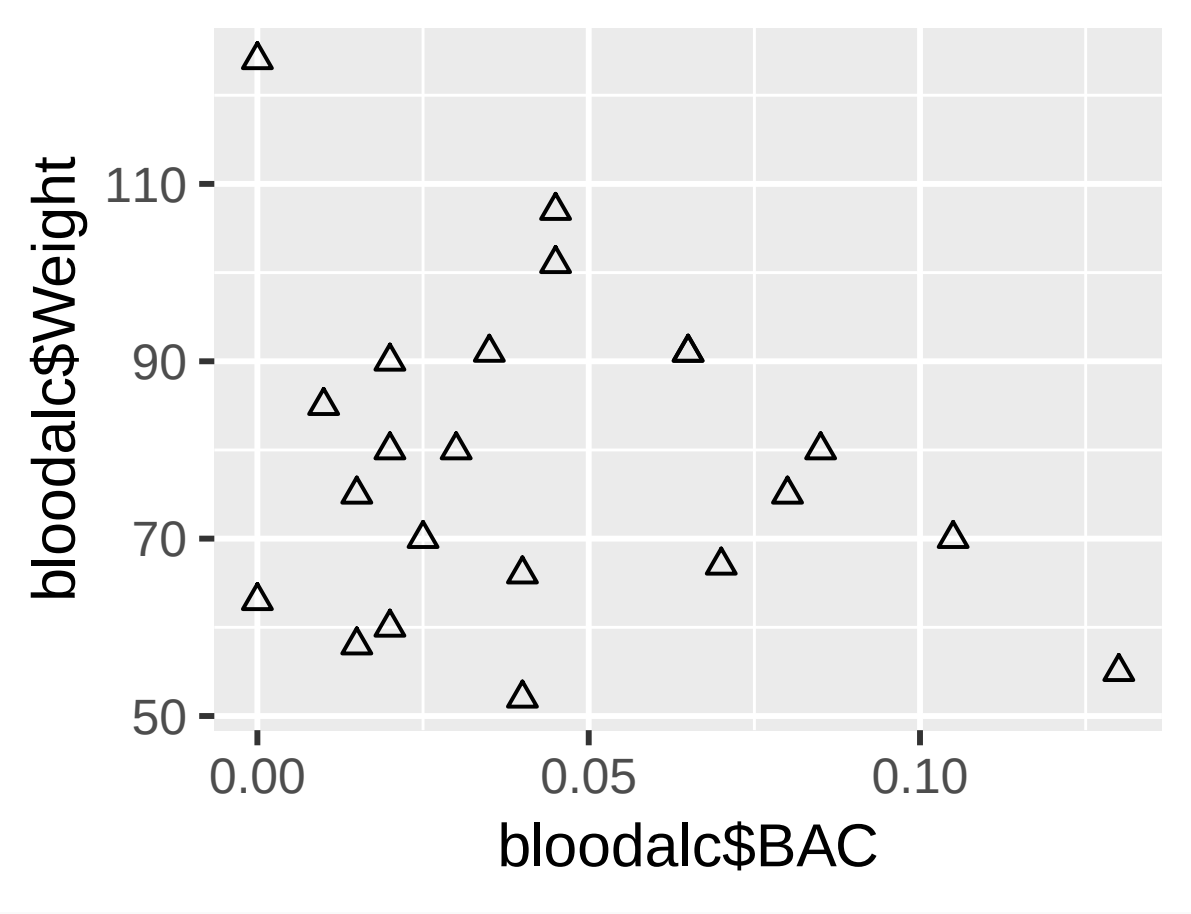
Example Scatterplot

```
qplot(x = bloodalc$BAC,
      y = bloodalc$Weight,
      geom = "point")
```



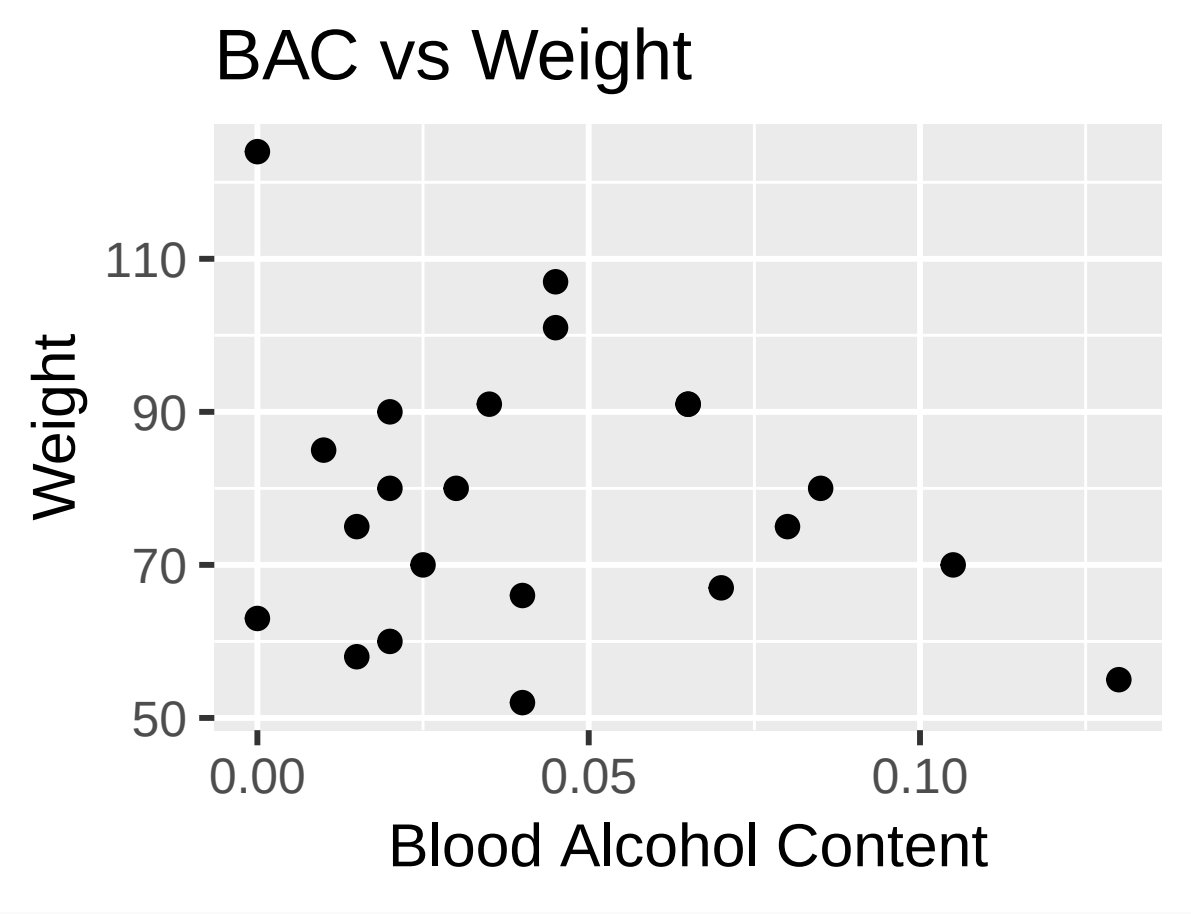
Change the point character

```
qplot(x = bloodalc$BAC,
      y = bloodalc$Weight,
      geom = "point",
      shape = I(2))
```



Add labels

```
qplot(x = bloodalc$BAC, y = bloodalc$Weight,
      geom = "point",
      xlab = "Blood Alcohol Content", ylab = "Weight",
      main = "BAC vs Weight")
```

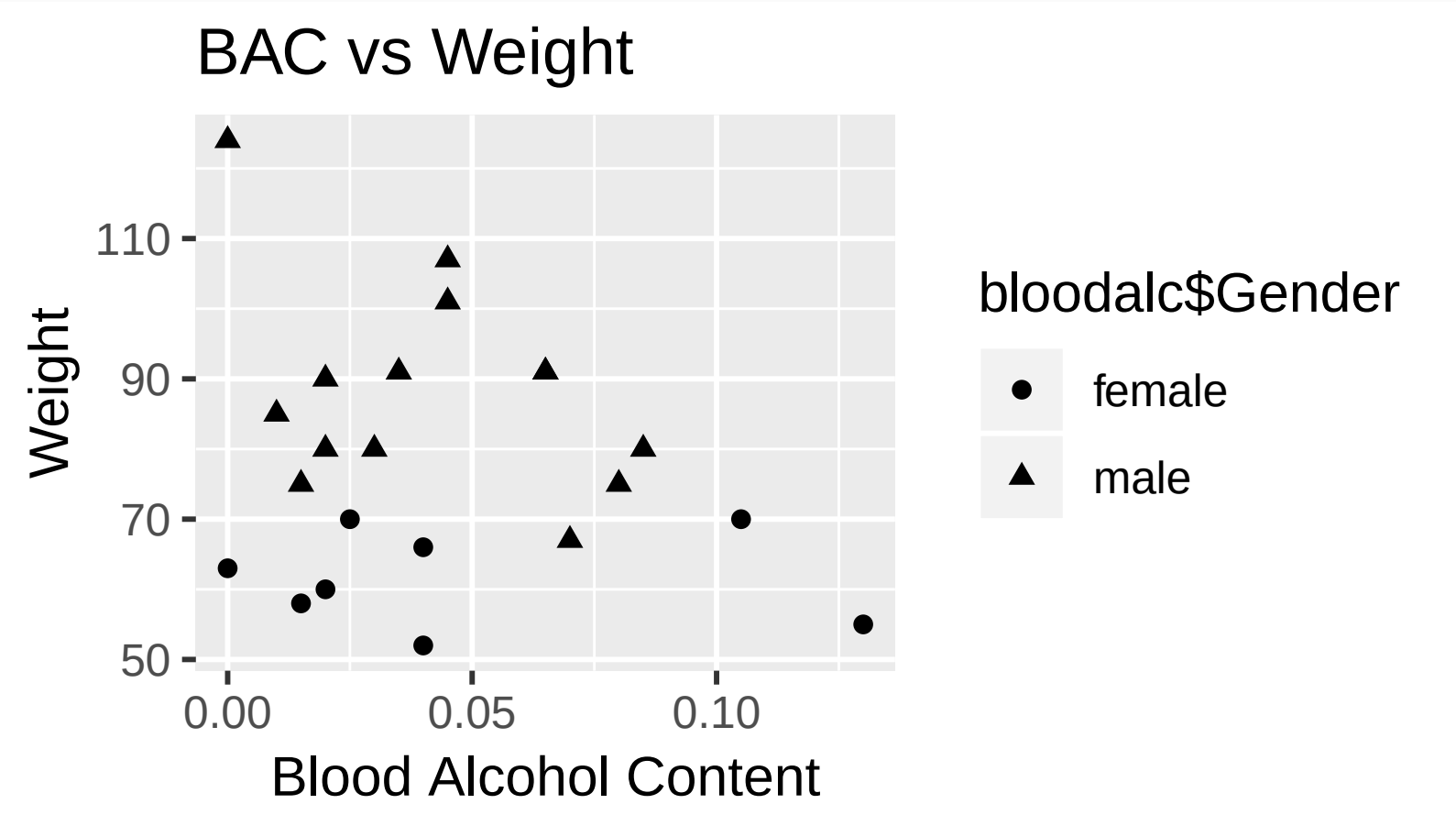


The `I()` Functions i

The `I()` function used within `qplot()` tells `qplot()` to force all points to be the same character. If you don't use `I()`, `qplot()` is expecting another variable the same length as `x` and `y` that will determine what each character will be.

```
qplot(x = bloodalc$BAC,
      y = bloodalc$Weight,
      shape = bloodalc$Gender,
      geom = "point",
      xlab = "Blood Alcohol Content",
      ylab = "Weight",
      main = "BAC vs Weight")
```

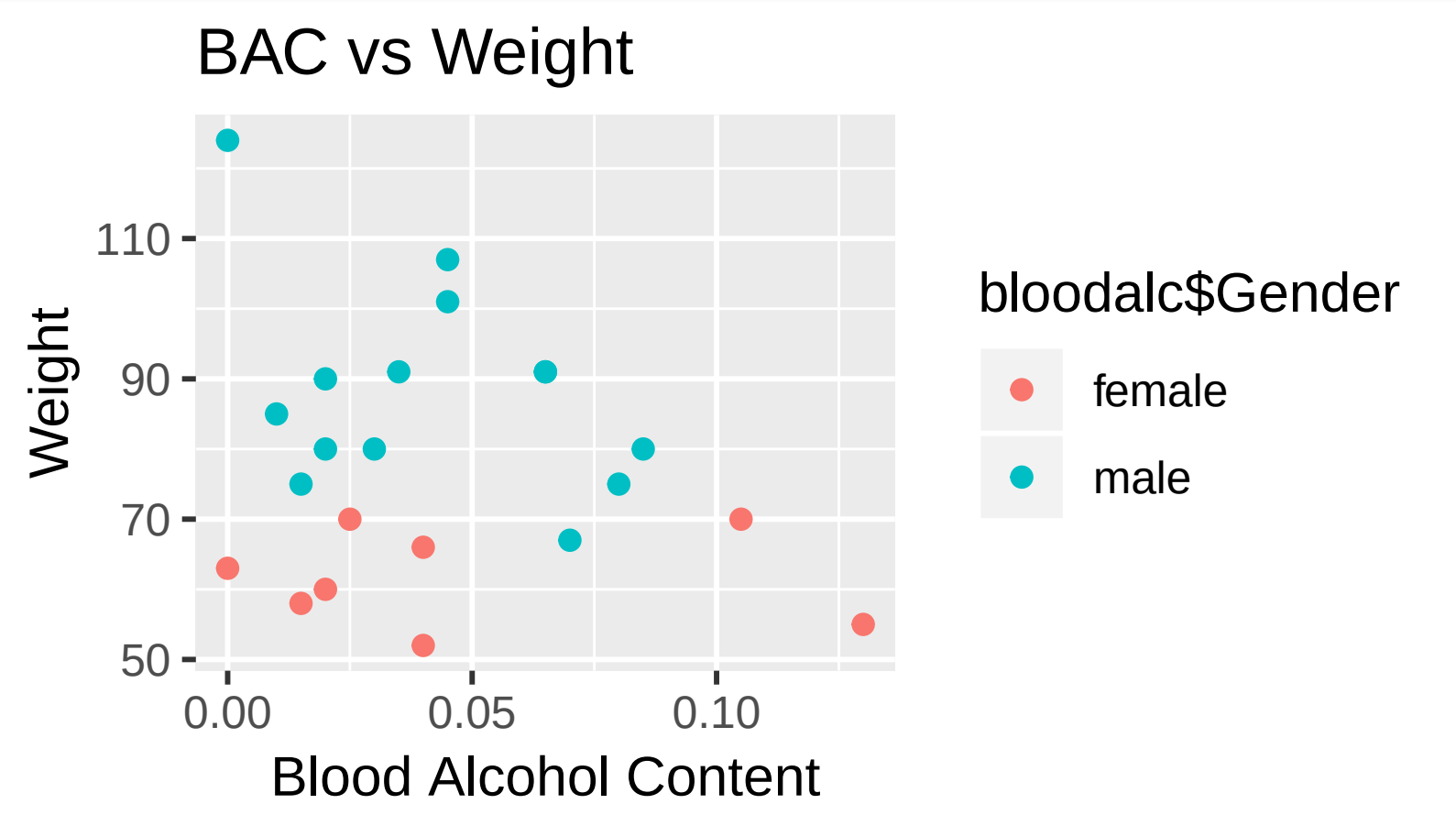
The `I()` Functions ii



Colors i

```
qplot(x = bloodalc$BAC,
      y = bloodalc$Weight,
      color = bloodalc$Gender,
      geom = "point",
      xlab = "Blood Alcohol Content",
      ylab = "Weight",
      main = "BAC vs Weight")
```

Colors ii


BAC vs Weight
Weight
110
90
70
50
0.00
0.05
0.10
Blood Alcohol Content
bloodalc$Gender
female
male

Cycling

The plots created by the `qplot()` functions used in the script can be found in the Plot window. Use the arrow to cycle through all four plots we just created. Use the Export menu to save or copy the current plot.

Histogram i

- The **distribution** of a variable tells us what values it takes and how often it takes these values.
- **Histograms** plot the frequencies (counts), percents, or proportions of equal-width classes of values. They describe the distribution of a **continuous** variable. E.g.

```
x <- c(1, 1.2, 2, 3, 3.5, 3.9)
```

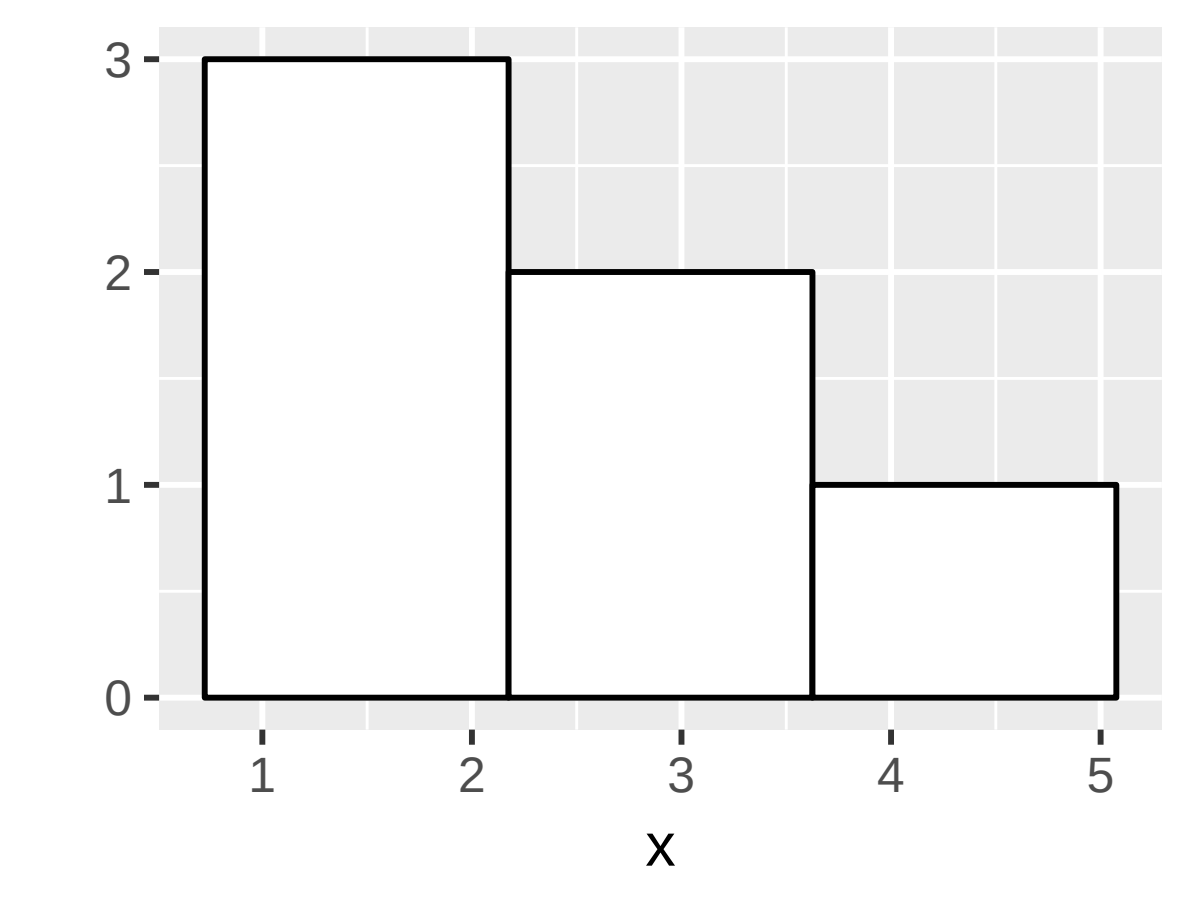
Bin the observations into one of three groups:

- group1 $= x : x \leq 2$
- group2 $= x : 2 < x \leq 3$
- group3 $= x : 3 < x \leq 4$

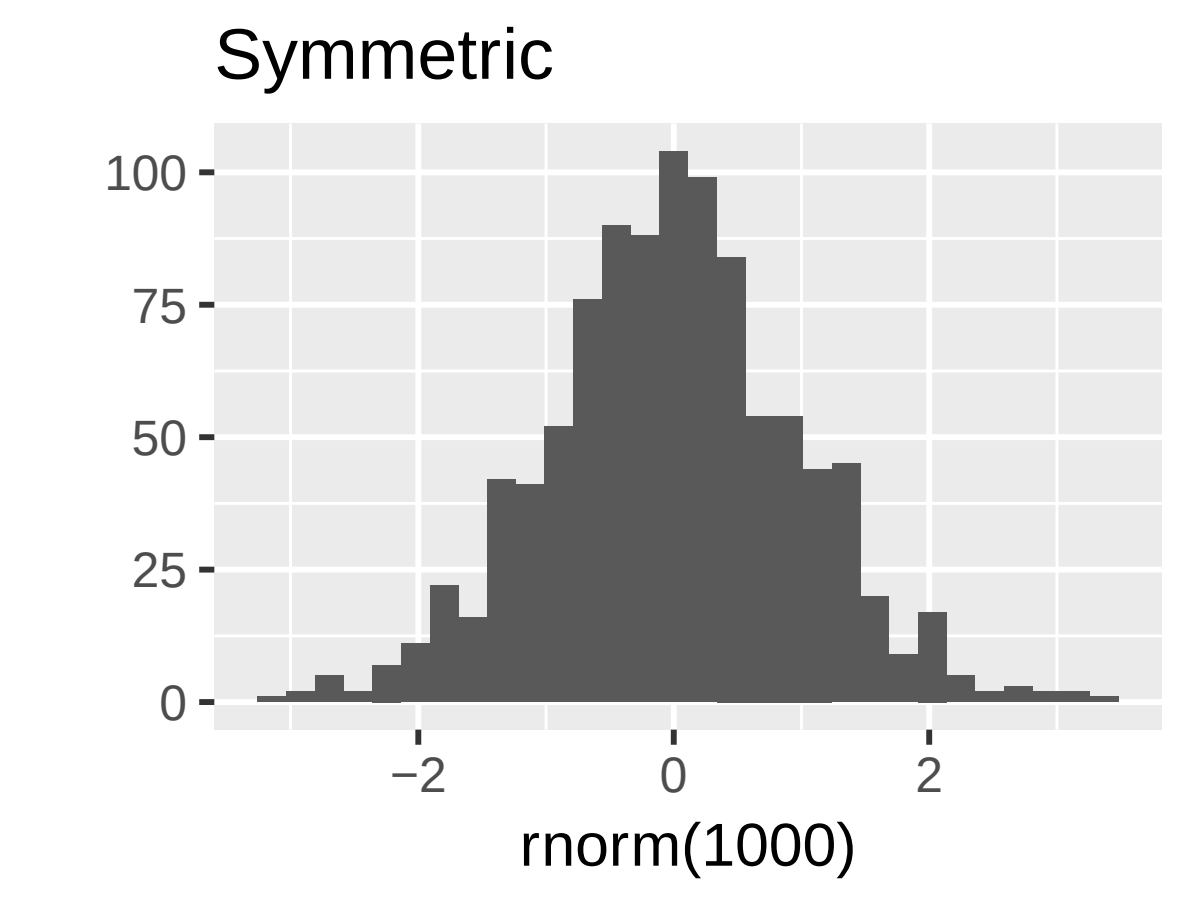
Histogram ii

```
qplot(x,
      geom = "histogram",
      bins = 3,
      fill = I("white"),
      color = I("black"))
```

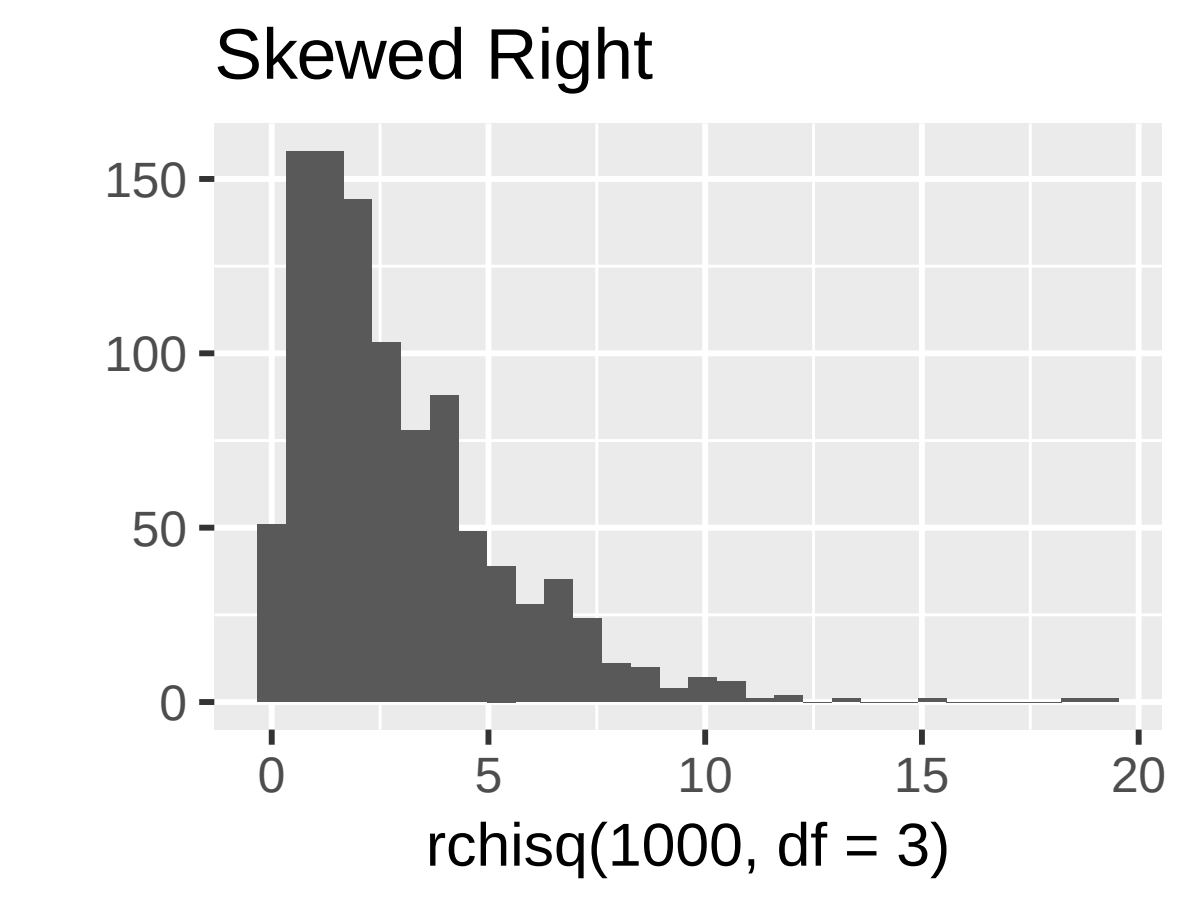
Histogram iii



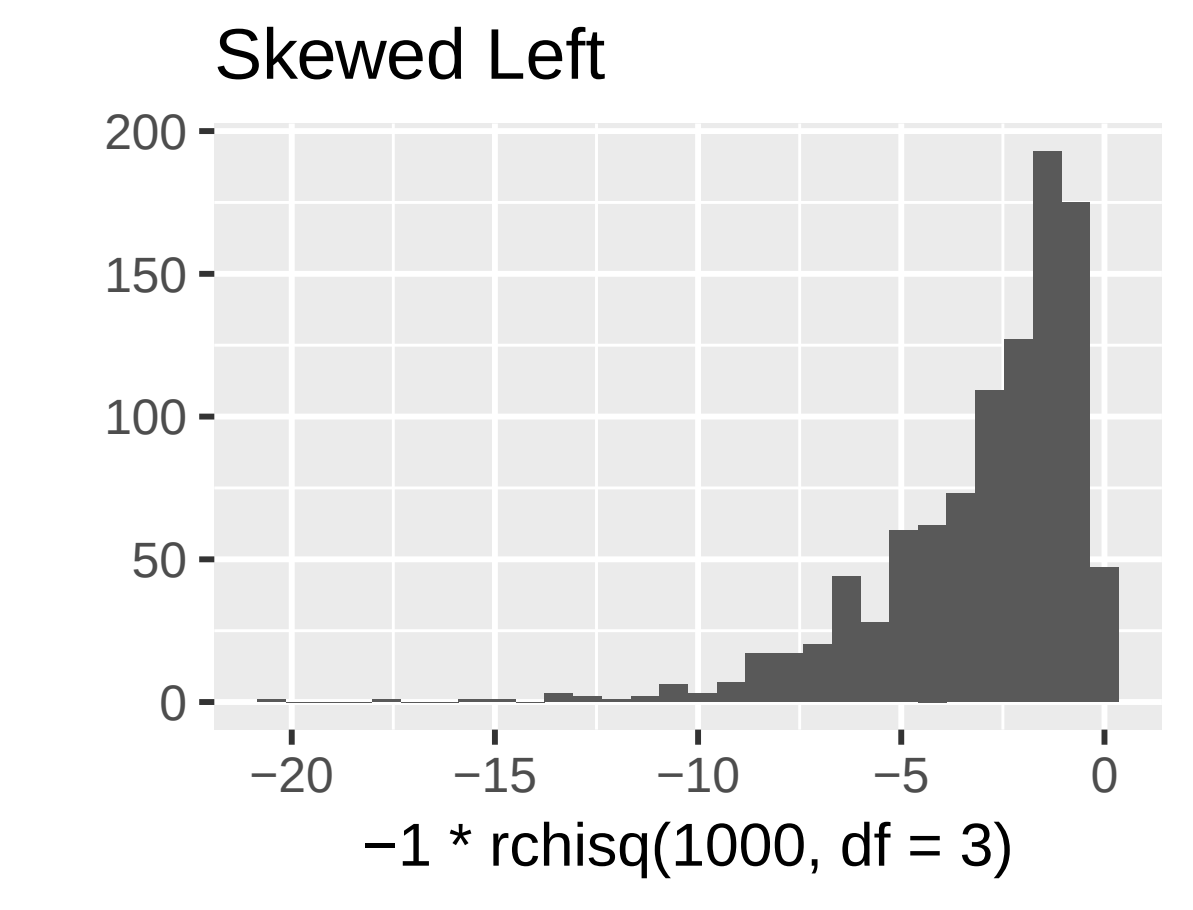
Skew i



Skew ii



Skew iii



Numerical Summaries

The Mean

The **mean** is the average value. You sum of the values, then divide by the number of observations. The mean of $x_1, x_2, \ldots, x_n$ is

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

In R, you use the `mean()` function.

```
mean(bloodalc$BAC)
```

```
## [1] 0.04364
```

The Mean is Sensitive to outliers

```
mean(c(2,3,5,7))
```

```
## [1] 4.25
```

```
mean(c(2,3,5,70))
```

```
## [1] 20
```

```
mean(c(2,3,5,700))
```

```
## [1] 177.5
```

```
mean(c(2,3,5,7000))
```

```
## [1] 1752
```

The Median

The **median** is the "middle point". It is defined as

- The $\left(\frac{n+1}{2}\right)$th largest observation if n is odd.
- The average of the $\left(\frac{n}{2}\right)$th and $\left(\frac{n}{2}+1\right)$th largest observations if n is even.

Use the `median()` function in R.

```
median(bloodalc$BAC)
```

```
## [1] 0.0375
```

Median "by hand" i

```
x <- c(1,2,8,10,18,23,36)
```

Sort the observations

```
sorted_x <- sort(x)
sorted_x
```

```
## [1]  1  2  8 10 18 23 36
```

`x` is odd with length `length(x)` $= 7$, so we take the 4th observation of `sorted_x`.

```
middle_num <- (length(x) + 1) / 2
sorted_x[middle_num]
```

Median “by hand” ii

```
## [1] 10
```

Same as `median(x)`

```
median(x)
```

```
## [1] 10
```

The Median is Robust to outliers

```
median(c(2,3,5,7))
```

```
## [1] 4
```

```
median(c(2,3,5,70))
```

```
## [1] 4
```

```
median(c(2,3,5,700))
```

```
## [1] 4
```

```
median(c(2,3,5,7000))
```

```
## [1] 4
```

When to use each measure of center

- Generally you use the mean when you have (i) symmetric data with no and few outliers or (ii) when a "total" is important.
- You use the median with you have (i) skewed data or (ii) many outliers or (iii) the "typical value" is important.

The standard deviation

- The **standard deviation** is, roughly, how far away the points are on average from the *mean*.
- Exact definition:

$$\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

- The **variance** is the square of the standard deviation.
- Generally use in same situations where mean is appropriate.

The standard deviation is not robust

```
sd(c(2,3,5,7))
```

```
## [1] 2.217
```

```
sd(c(2,3,5,70))
```

```
## [1] 33.36
```

```
sd(c(2,3,5,700))
```

```
## [1] 348.3
```

```
sd(c(2,3,5,7000))
```

```
## [1] 3498
```

The range

- The **range** is the largest value minus the smallest value.
- A measure of spread.
- Variables that have large ranges are more spread out.
- The range is not robust to outliers.

Quantiles

- The ***p*th quantile** (or ***p*th percentile**) is the value V_p such that p percent of the sample points are at or below V_p.
- People generally return the 25th, 50th, and 75th quantiles to give you an idea of how spread out the data are.

```
quantile(bloodalc$BAC, c(0.25, 0.5, 0.75))
```

```
##     25%     50%     75%
## 0.0200 0.0375 0.0650
```

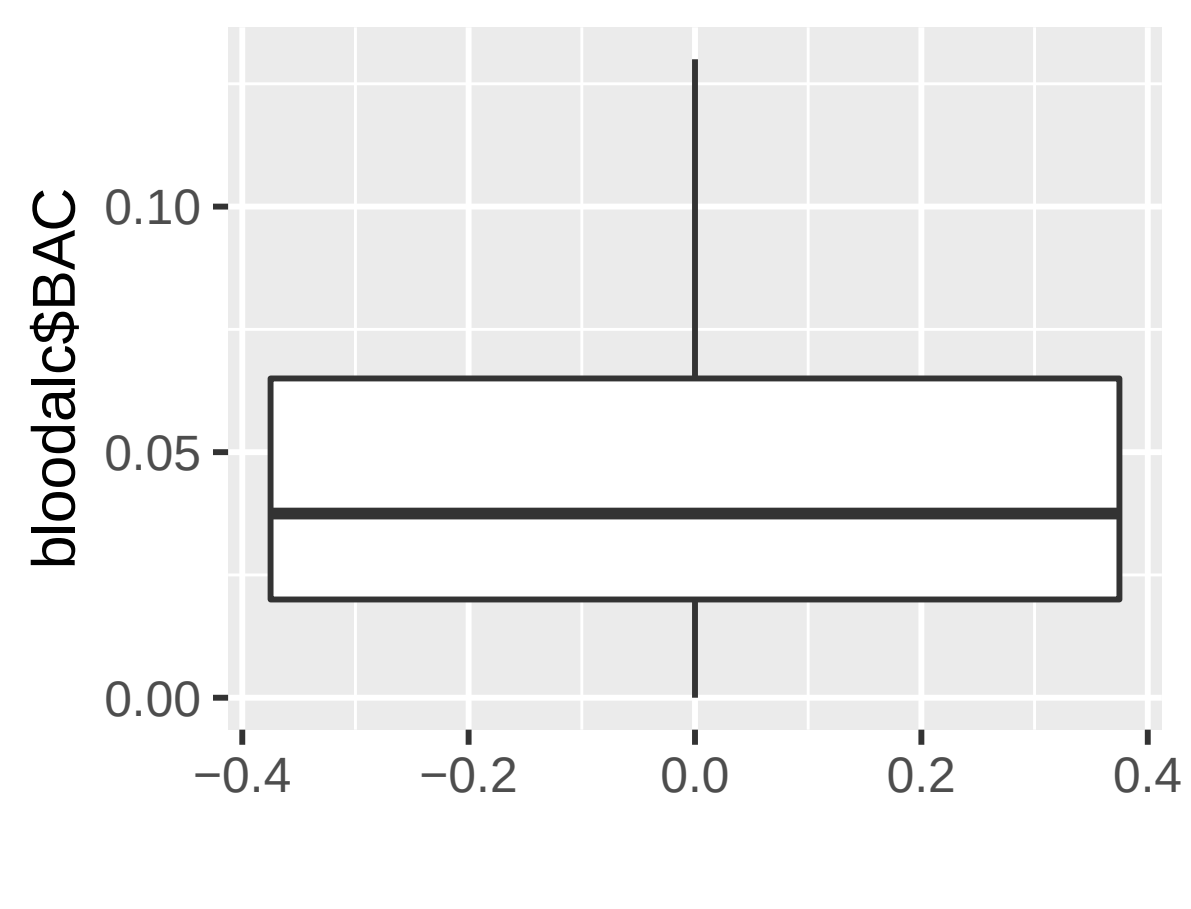
Boxplot

- Plots the 0th, 25th, 50th, 75th, and 100th quantiles
- Useful for comparing continuous distributions.

Boxplots with qplot

- To plot one variable, just use `geom = "boxplot"`

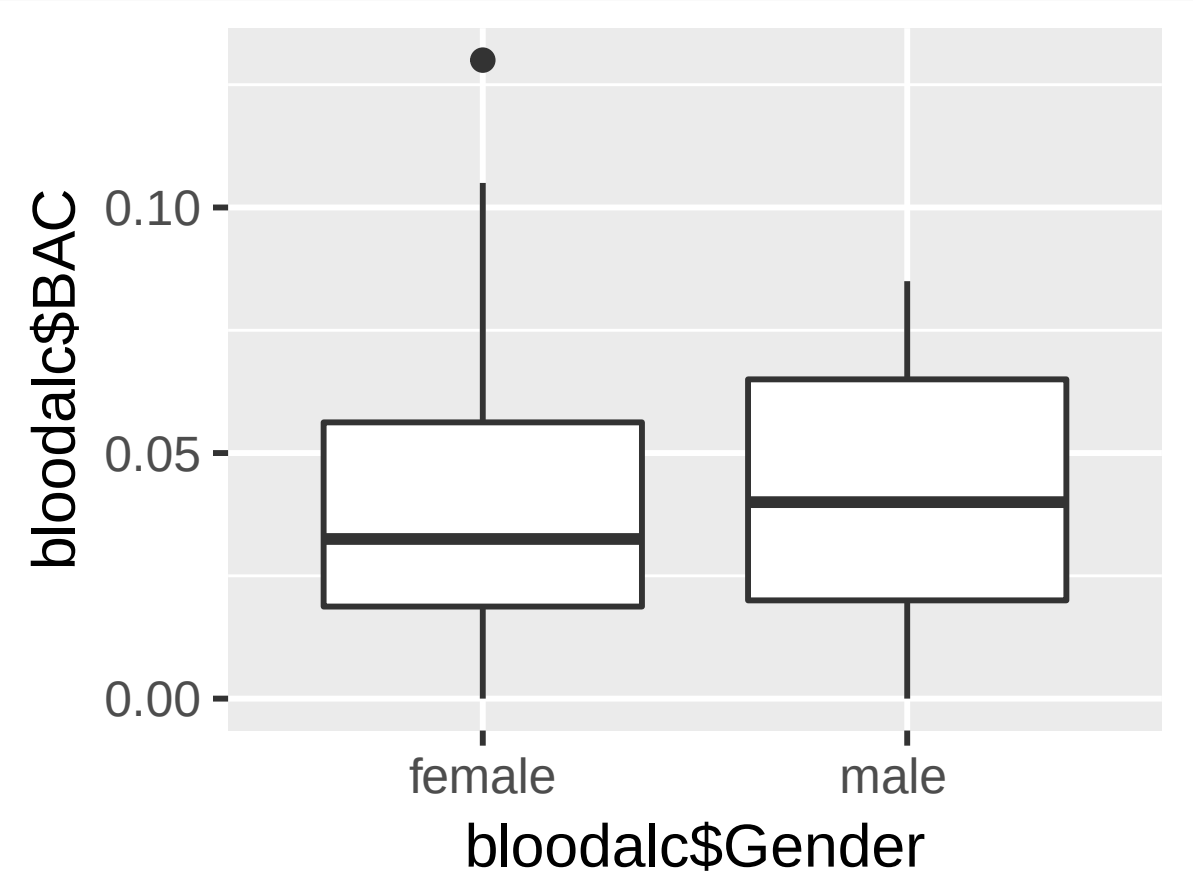
```
qplot(y = bloodalc$BAC, geom = "boxplot")
```



Boxplots with qplot

- To plot one two variables, need an x-axis variable distinguishing between the different variables.

```
qplot(x = bloodalc$Gender,
      y = bloodalc$BAC, geom = "boxplot")
```



Why use `qplot()`?

Why use `qplot()`? i

Most R intro classes will teach graphics with the default `plot()` function. So why are we using `qplot()`? Reasons:

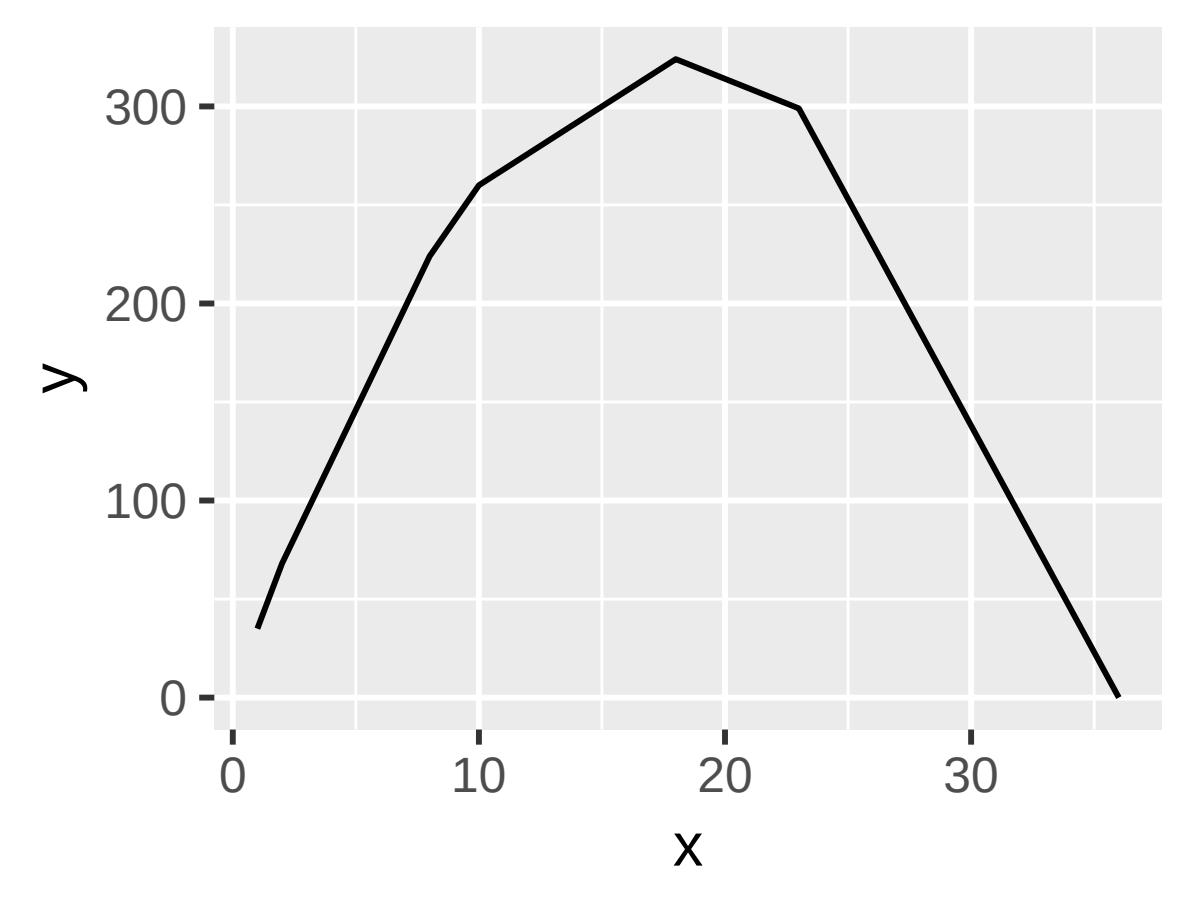
1. Unless you are doing something exotic, you can plot more with fewer lines of code.
2. The defaults look a little better.

Insert Data

```
x <- c(1, 2, 8, 10, 18, 23, 36)
y <- 36 * x - x ^ 2
```

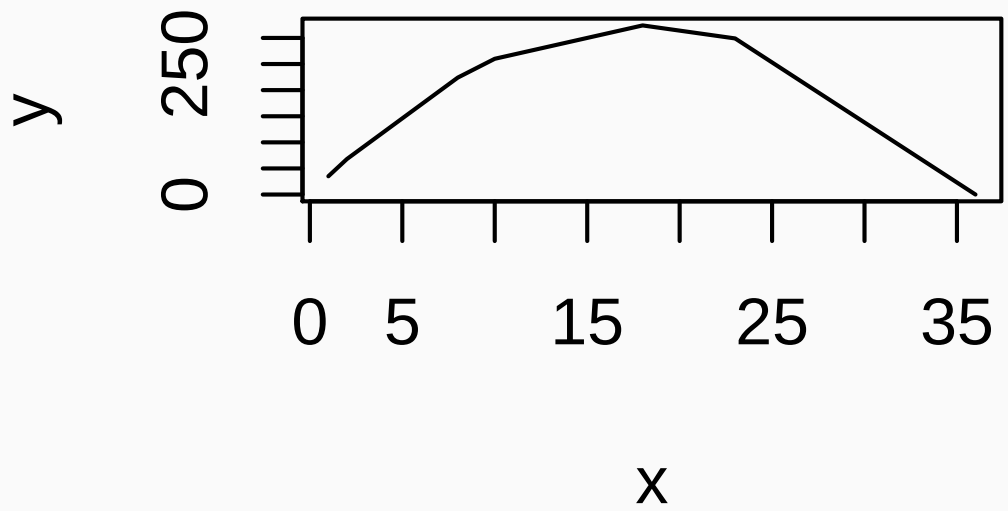
Using ggplot

```
qplot(x, y, geom = "line")
```



Using base graphics

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



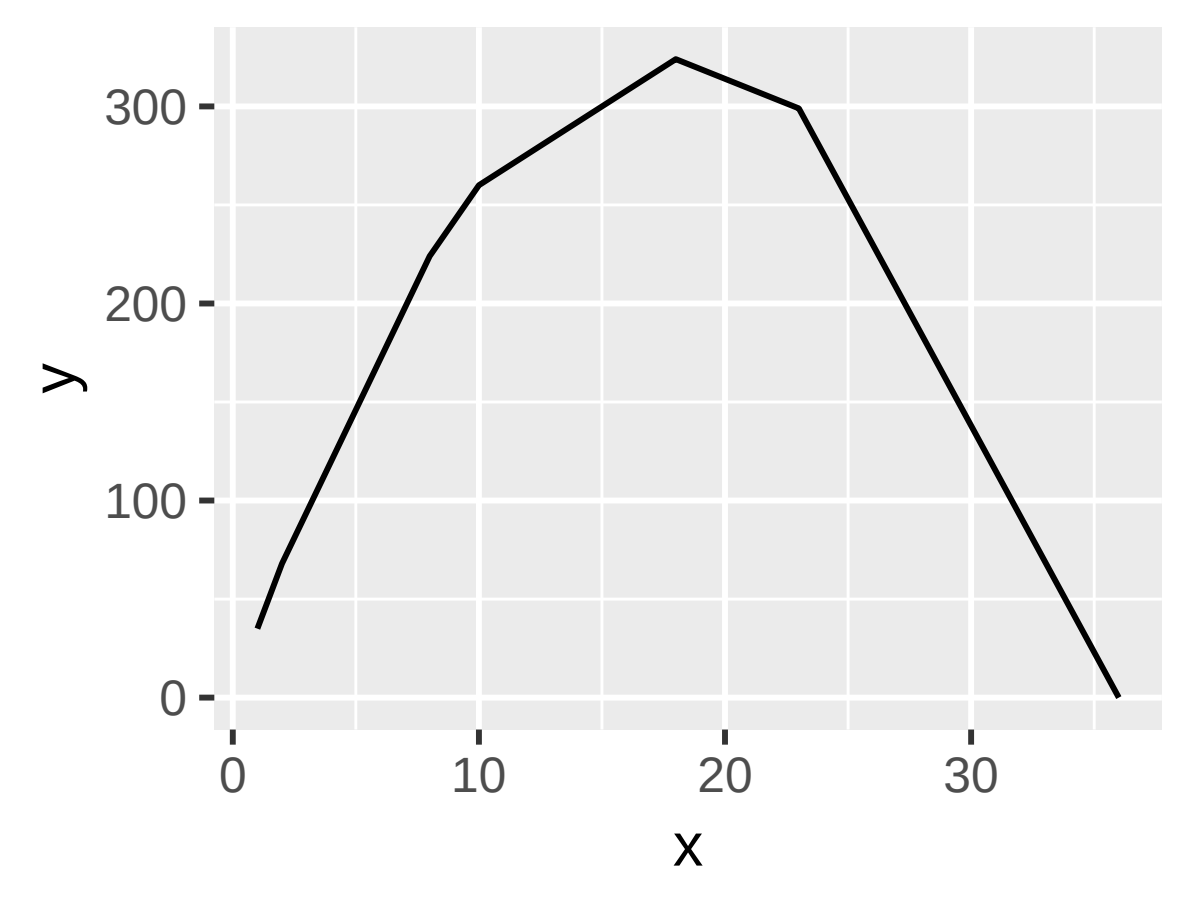
Unsorted Data

`plot()` does not work well with unsorted data, but `qplot()` handles it automatically

```
x <- c(36, 18, 8, 2, 10, 23, 1)
y <- 36 * x - x ^ 2
```

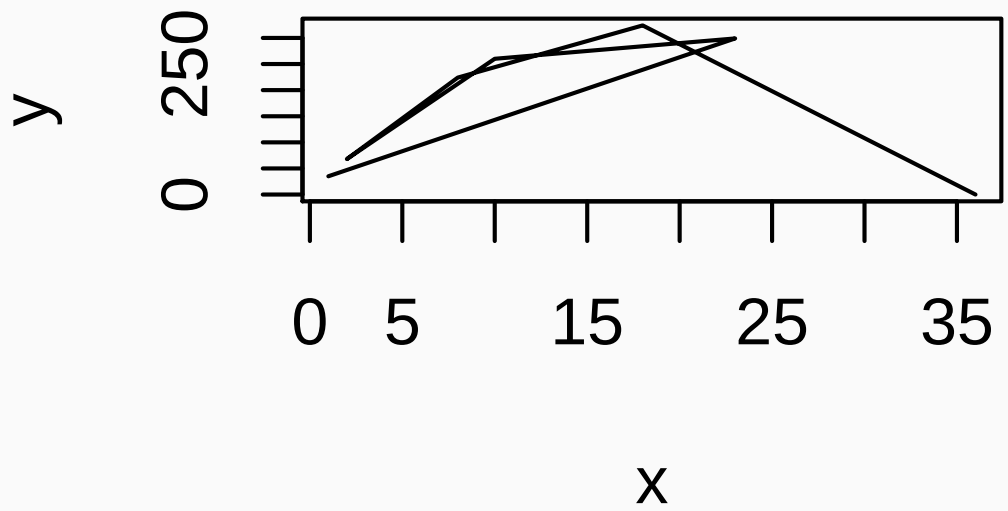
Using ggplot

```
qplot(x, y, geom = "line")
```



Using base graphics the wrong way

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



Using base graphics the right way i

To use `plot()`, you need to first sort your `x` values and correspondlingly permute the matching `y` values.

```
orderx <- order(x)
orderx
```

```
## [1] 7 4 3 5 2 6 1
```

```
x[orderx]
```

```
## [1]  1  2  8 10 18 23 36
```

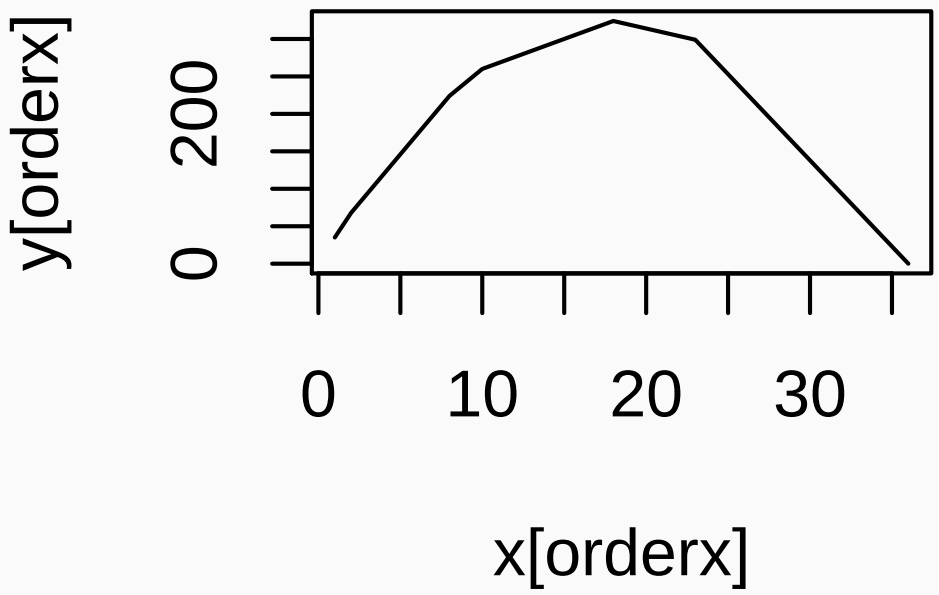
```
y[orderx]
```

```
## [1]  35  68 224 260 324 299   0
```

Using base graphics the right way ii

```
plot(x[orderx], y[orderx], type = "l")
```

Using base graphics the right way iii



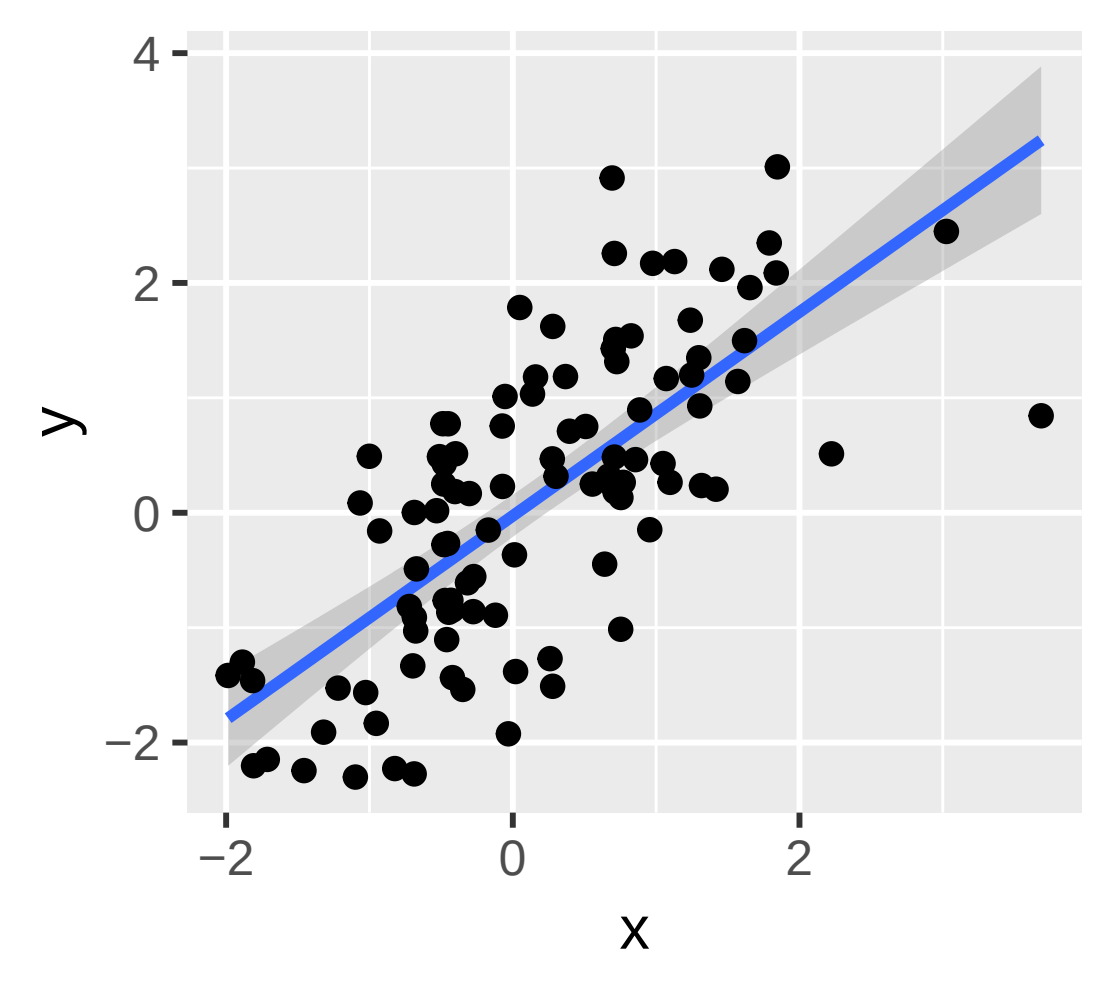
Really Complicated Plot i

For even more complicated tasks, ggplot is a lot better.

```
## Generate data
x <- rnorm(100)
y <- x + rnorm(100)

## A complicated plot using qplot()
qplot(x, y, geom = "smooth", method = "lm") + geom_point()
```

Really Complicated Plot ii



Same type of plot with `plot()` i

```
lm_out <- lm(y ~ x)
par(mar = c(3, 3, 3, 0.5))
pred_out <- predict(lm_out, se.fit = TRUE)
upper <- pred_out$fit + 1.96 * pred_out$se.fit
lower <- pred_out$fit - 1.96 * pred_out$se.fit
orderx <- order(x)
plot(x[orderx], pred_out$fit[orderx], type = "l",
     ylim = c(min(lower), max(upper)))
lines(x[orderx], upper[orderx])
lines(x[orderx], lower[orderx])
points(x[orderx], y[orderx])
```

Same type of plot with `plot()` ii