R Programming Week 4

Rudy Rupp

13/10/2020

R Programming Week 4: Simulation and Profiling

Simulating data in R can be useful for simulation studies. The profiler tool lets you collect detailed information about how your functions are running, to identify bottlenecks that need to be addressed. This week also covers the use of the STR function.

Learning Objectives

- Call the 'str' function on an object.
- describe the difference beween the 'by.self' and 'by.total' output produced by the R profiler.
- Simulate a random normal variable with an arbitrary mean and standard deviation.
- Simulate data form a normal linear model.

The 'str' Function

It compactly displays the internal structure of a object. It's a simple diagnostic function, similar to 'summary'. It's particularly well suited to displaying the contents of nested lists, and tries to give only one line per basic object.

```
str(mtcars)  ## summary of a data frame or nested list
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

```
str(lm) ## gives arguments of a function
```

```
## function (formula, data, subset, weights, na.action, method = "qr", model = TRUE,
##     x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE, contrasts = NULL,
##     offset, ...)
```

```r
x <- rnorm(100, 2, 4)
summary(x)  ## quantiles of numeric data
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -11.037  -1.347   1.524   1.533   4.508  13.795
```

```r
str(x)  ## class, size and head
```

```
##  num [1:100] 1.11 -1.93 4.8 2.54 -1.18 ...
```

```r
f <- gl(40, 10)
str(f)  ## identifies this numeric set as a list of factors
```

```
##  Factor w/ 40 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```r
summary(f)  ## summary is not as useful in this case.
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
## 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
## 27 28 29 30 31 32 33 34 35 36 37 38 39 40
## 10 10 10 10 10 10 10 10 10 10 10 10 10 10
```

```r
library(datasets)
str(airquality)  ## output for a data frame
```

```
## 'data.frame':    153 obs. of  6 variables:
##  $ Ozone  : int  41 36 12 18 NA 28 23 19 8 NA ...
##  $ Solar.R: int  190 118 149 313 NA NA 299 99 19 194 ...
##  $ Wind   : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
##  $ Temp   : int  67 72 74 62 56 66 65 59 61 69 ...
##  $ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
##  $ Day    : int  1 2 3 4 5 6 7 8 9 10 ...
```

```r
m <- matrix(rnorm(100), 10, 10)
str(m)  ## results for a matrix, dimensions and sample of first column
```

```
##  num [1:10, 1:10] -1.65 1.2 -1.17 1.76 -2.2 ...
```

```r
s <- split(airquality, airquality$Month)
 ## this is a list with 5 data frames, each one a piece of the
 ## original, separated by month.
str(s)  ## gives some detail for each element
```

```
## List of 5
##  $ 5:'data.frame':   31 obs. of  6 variables:
##   ..$ Ozone  : int [1:31] 41 36 12 18 NA 28 23 19 8 NA ...
##   ..$ Solar.R: int [1:31] 190 118 149 313 NA NA 299 99 19 194 ...
##   ..$ Wind   : num [1:31] 7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
##   ..$ Temp   : int [1:31] 67 72 74 62 56 66 65 59 61 69 ...
##   ..$ Month  : int [1:31] 5 5 5 5 5 5 5 5 5 5 ...
##   ..$ Day    : int [1:31] 1 2 3 4 5 6 7 8 9 10 ...
##  $ 6:'data.frame':   30 obs. of  6 variables:
##   ..$ Ozone  : int [1:30] NA NA NA NA NA NA 29 NA 71 39 ...
##   ..$ Solar.R: int [1:30] 286 287 242 186 220 264 127 273 291 323 ...
##   ..$ Wind   : num [1:30] 8.6 9.7 16.1 9.2 8.6 14.3 9.7 6.9 13.8 11.5 ...
##   ..$ Temp   : int [1:30] 78 74 67 84 85 79 82 87 90 87 ...
##   ..$ Month  : int [1:30] 6 6 6 6 6 6 6 6 6 6 ...
##   ..$ Day    : int [1:30] 1 2 3 4 5 6 7 8 9 10 ...
##  $ 7:'data.frame':   31 obs. of  6 variables:
##   ..$ Ozone  : int [1:31] 135 49 32 NA 64 40 77 97 97 85 ...
##   ..$ Solar.R: int [1:31] 269 248 236 101 175 314 276 267 272 175 ...
##   ..$ Wind   : num [1:31] 4.1 9.2 9.2 10.9 4.6 10.9 5.1 6.3 5.7 7.4 ...
##   ..$ Temp   : int [1:31] 84 85 81 84 83 83 88 92 92 89 ...
##   ..$ Month  : int [1:31] 7 7 7 7 7 7 7 7 7 7 ...
##   ..$ Day    : int [1:31] 1 2 3 4 5 6 7 8 9 10 ...
##  $ 8:'data.frame':   31 obs. of  6 variables:
##   ..$ Ozone  : int [1:31] 39 9 16 78 35 66 122 89 110 NA ...
##   ..$ Solar.R: int [1:31] 83 24 77 NA NA NA 255 229 207 222 ...
##   ..$ Wind   : num [1:31] 6.9 13.8 7.4 6.9 7.4 4.6 4 10.3 8 8.6 ...
##   ..$ Temp   : int [1:31] 81 81 82 86 85 87 89 90 90 92 ...
##   ..$ Month  : int [1:31] 8 8 8 8 8 8 8 8 8 8 ...
##   ..$ Day    : int [1:31] 1 2 3 4 5 6 7 8 9 10 ...
##  $ 9:'data.frame':   30 obs. of  6 variables:
##   ..$ Ozone  : int [1:30] 96 78 73 91 47 32 20 23 21 24 ...
##   ..$ Solar.R: int [1:30] 167 197 183 189 95 92 252 220 230 259 ...
##   ..$ Wind   : num [1:30] 6.9 5.1 2.8 4.6 7.4 15.5 10.9 10.3 10.9 9.7 ...
##   ..$ Temp   : int [1:30] 91 92 93 93 87 84 80 78 75 73 ...
##   ..$ Month  : int [1:30] 9 9 9 9 9 9 9 9 9 9 ...
##   ..$ Day    : int [1:30] 1 2 3 4 5 6 7 8 9 10 ...
```

Simulation - Generating Random Numbers

```
## List of 4
##  $ dnorm:function (x, mean = 0, sd = 1, log = FALSE)
##  $ pnorm:function (q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
##  $ qnorm:function (p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
##  $ rnorm:function (n, mean = 0, sd = 1)
```

- `rnorm`: random normal variates, with given mean and standard deviation.

- `dnorm` calculates the density. I.e.: for a value of `x`, the function gives the frequency (vertical axis) of the distribution. Equivalent of applying the normal distribution function

$$\frac{1}{\sigma\sqrt{2\pi}} * e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- The Log can be calculated instead, but is FALSE by default.
- It's useful to think of `x` in `dnorm` as the z-score, for the standard normal distribution.

- `pnorm` returns the area under the curve, from $-\infty$ to `q`. `q` is a z-score. (I.e. the probability that any given value of the population falls between $-\infty$ and `q`.)

- `qnorm` is the opposite of `pnorm`. Given a probability `p`, it returns the z-score for that probability.

  - `pnorm` and `qnorm` also can evaluate the Log, and also have an option for the lower tail. This is the left side of the distribution. Setting it to FALSE will determine the upper tail instead.

- The same 4 functions exist for different distributions, too. E.g. pois for poisson and unif for uniform, gamma for gama, etc.

When using random number generators, you must set a seed first, for the values to be reproducible

```
set.seed(1)
rnorm(5)
```

```
## [1] -0.6264538  0.1836433 -0.8356286  1.5952808  0.3295078
```

```
rnorm(5)  ## different from original, seed not reset.
```

```
## [1] -0.8204684  0.4874291  0.7383247  0.5757814 -0.3053884
```

```
set.seed(1)
rnorm(5)  ## equal to original, seed has been reset.
```

```
## [1] -0.6264538  0.1836433 -0.8356286  1.5952808  0.3295078
```

Simulation - Simulating a Linear Model

Suppose we want to simulate the model

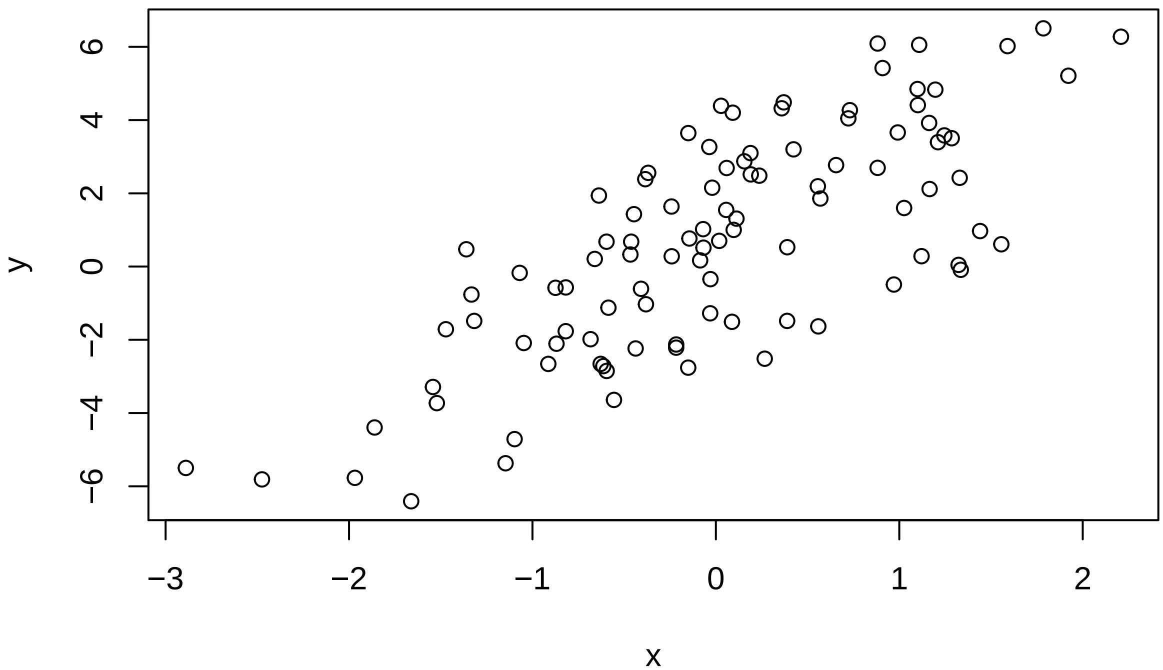
$$y = \beta_0 + \beta_l x + \varepsilon$$

where $\varepsilon \sim N(0, 2^2)$. Assume $x \sim N(0, 1^2)$, $\beta_0 = 0.5$ and $\beta_l = 2$.

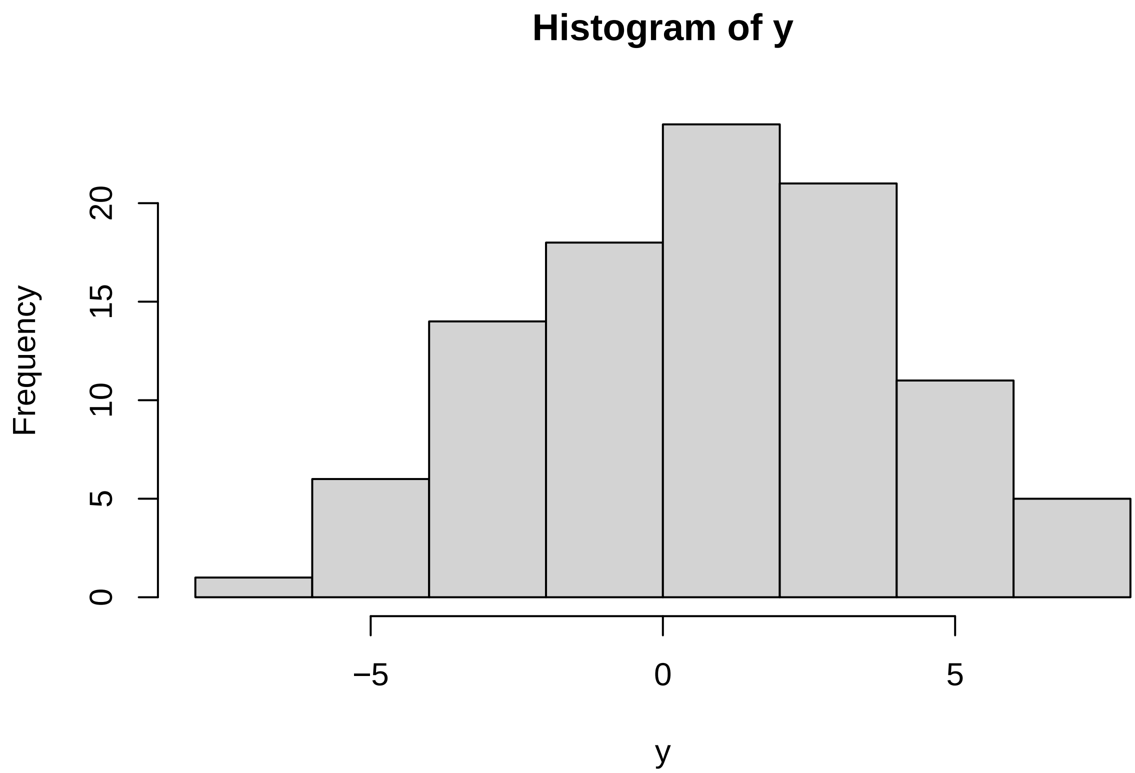
```
set.seed(20)
x <- rnorm(100)
e <- rnorm(100, 0, 2)
y <- 0.5 + 2 * x + e
summary(y)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -6.4084 -1.5402  0.6789  0.6893  2.9303  6.5052
```

```
plot(x, y)
```




```r
hist(y)
```



Suppose we want to simulate from a Poisson model where

$$Y \sim Poisson(\mu)$$

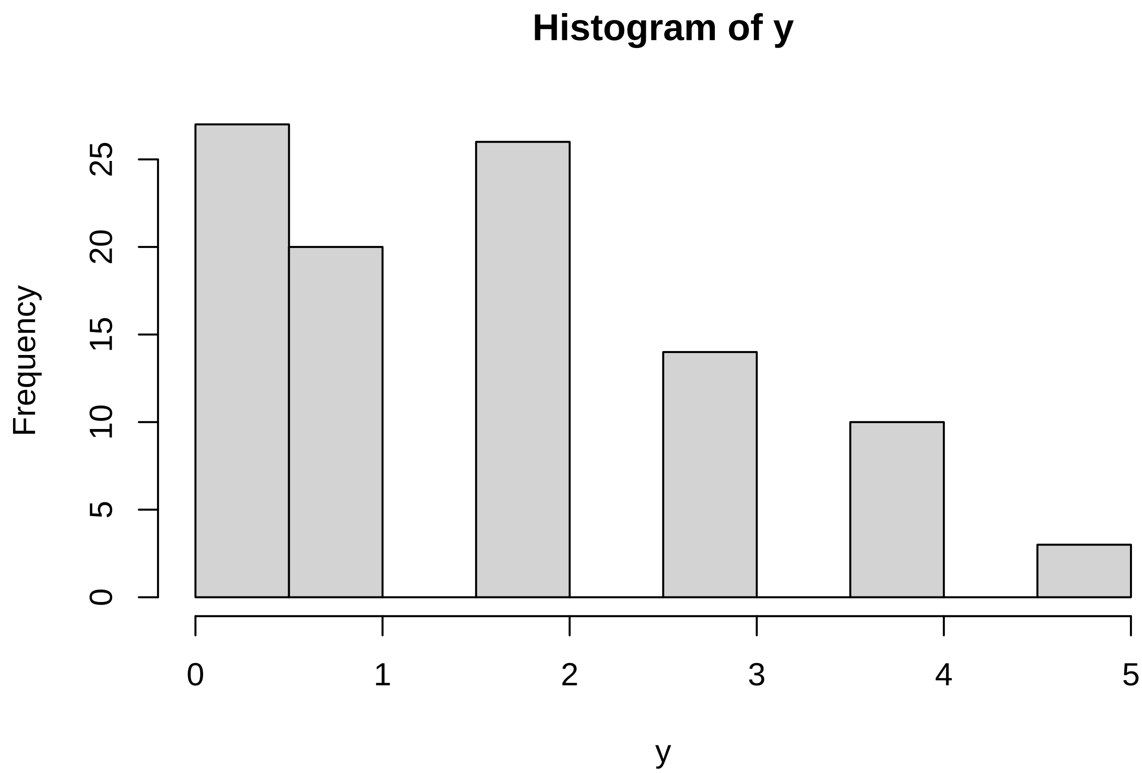
$$log(\mu) = \beta_0 + \beta_l x$$

and $\beta_0 = 0.5$ and $\beta_l = 0.3$. We need to use `rpois` function for this:

```
set.seed(20)
x <- rnorm(100)  ## same x as previous simulation
log.mu <- 0.5 + 0.3 * x
y <- rpois(100, exp(log.mu))
y
```

```
##   [1] 3 3 2 0 3 2 1 0 2 2 2 4 1 4 0 3 3 4 0 4 3 0 5 2 0 1 0 1 0 1 0 0 0 1 4 0 1
##  [38] 2 1 1 2 2 2 2 2 2 5 2 3 0 3 0 3 4 0 0 1 1 0 1 0 0 3 0 0 0 5 2 0 4 1 0 1 1
##  [75] 3 2 0 2 2 3 1 0 1 1 2 2 4 2 2 0 3 1 2 2 4 2 2 4 1 3
```

```
hist(y)
```



Simulation - Random Sampling

The `sample` function allows us to get random samples from a vector of numbers or characters.

```
## function (x, size, replace = FALSE, prob = NULL)
```

```r
set.seed(1)
sample(1:10, 4) ## gives a random sample of 4 elements from the vector
```

```
## [1] 9 4 7 1
```

```r
sample(letters, 5) ## also works with character vectors
```

```
## [1] "b" "w" "k" "n" "r"
```

```r
sample(1:10)  ## if no size is specified, it gives a random permutation.
```

```
##  [1]  3  1  5  8  2  6 10  9  4  7
```

```r
sample(1:10, replace = TRUE) ## allows an item to be sampled more than once
```

```
##  [1]  5  9  9  5  5  2 10  9  1  4
```

R Profiler

Useful tool for very large Data sets, or other work where processing might be taking a very long time. It helps you find out why things are taking so long and suggest strategies for faster solutions.

Why is My Code So Slow?

- Profiling is a systematic way to examine how much time is spent on different parts of the program.
- Useful when trying to optimize your code.
- Often code runs fine once, but what if you have to put it in a loop for 1000 iterations? Is it still fast enough?
- Profiling is better than guessing.

On Optimizing your code
* Optimization is not the first priority when writing code. It is more important to focus on readability and making sure it works. – It's often difficult to understand where exactly your program is spending most of its time, and this cannot be done without performance analysis or profiling.
> We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. > –Donald Knuth

General Principles
* Design first, then optimize * Remember premature optimization is the root of all evil * Measure (collect data), don't guess. * If you're going to be a scientist, you need to apply the same principles here.

Using `system.time()`

- Takes an arbitrary R expression as input (can be wrapped in curly braces) and returns the amount of time taken to evaluate the expression.
- Computes the time (in seconds) needed to execute an expression. – If there's an error, it gives the time until the error occured.
- Returns an object of class `proc_time` – **user time**: time charged to the CPU for this expression – **elapsed time**: "wall clock" time. – (usually both times are similar, but may differ if the computer has multiple CPUs and the program can use them, if the CPU spends time waiting for other things to occur, or in other more complex scenarios)

```
## Example Elapsed time > user time
system.time(readLines("http://www.jhsph.edu"))
```

```
##    user  system elapsed
##    0.00    0.01    0.87
```

```
## Elapsed time < user time
hilbert <- function(n) {
  i <- 1:n
  1 / outer(i - 1, i, "+")
}
x <- hilbert(1000)
system.time(svd(x))
```

```
##    user  system elapsed
##    2.43    0.02    2.45
```

```r
## the svd function takes advantage of multiple cores in some computers.

system.time( {
  n <- 1000
  r <- numeric(n)
  for (i in 1:n) {
    x <- rnorm(n)
    r[i] <- mean(x)
  }
  print(summary(r))
})  ## example of system.time using curly braces.
```

```
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## -0.1279199 -0.0226552 -0.0007275 -0.0013621  0.0194086  0.1170306
```

```
##    user  system elapsed
##    0.08    0.00    0.07
```

- `system.time()` allows you to test cartain functions or code blocks to see if they are taking excessive amounts of time.
- this assumes you already know where the problem is and can call `system.time()` on it.

Using `Rprof()`

- The `Rprof()` functions starts the profiler in R. – R must be compiled with profiler support, but this is usually the case.
- the `summaryRprof()` function summarizes the output from `Rprof()`. The raw output is unreadable.
- **DO NOT** use `system.time()` and `Rprof()` together.
- `Rprof()` keeps track of the function call stack at regularly sampled intervals and tabulates how much time is spent in each function.
- default sampling interval is 0.02 seconds.
- Note: if your code runs very quickly, the profiler is not helpful, but then you probably don't need it anyway.

Using `summaryRprof()`

- This function tabulates the R profiler and calculates how much time is spent in which function.
- There are two methods for normalizing the data.
- `by.total` divides the time spent in each function by the total run time.
- `by.self` does the same but first it subtracts out time spent in functions above the call stack.

Swirl Exercises Week 4

Swirl 12: Looking at Data

```r
dataframe <- read.csv(file = "hw1_data.csv")
class(dataframe)
```

```
## [1] "data.frame"
```

```r
dim(dataframe)
```

```
## [1] 153   6
```

```r
nrow(dataframe)
```

```
## [1] 153
```

```r
ncol(dataframe)
```

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

```r
object.size(dataframe)  ## memory space used
```

```
## 5632 bytes
```

```r
names(dataframe)  ## names of columns
```

```
## [1] "Ozone"   "Solar.R" "Wind"    "Temp"    "Month"   "Day"
```

```r
head(dataframe, 10)  ## show first 10 lines
```

```
##    Ozone Solar.R Wind Temp Month Day
## 1     41     190  7.4   67     5   1
## 2     36     118  8.0   72     5   2
## 3     12     149 12.6   74     5   3
## 4     18     313 11.5   62     5   4
## 5     NA      NA 14.3   56     5   5
## 6     28      NA 14.9   66     5   6
## 7     23     299  8.6   65     5   7
## 8     19      99 13.8   59     5   8
## 9      8      19 20.1   61     5   9
## 10    NA     194  8.6   69     5  10
```

```r
tail(dataframe, 10)  ## show last 10 lines
```

```
##     Ozone Solar.R Wind Temp Month Day
## 144    13     238 12.6   64     9  21
## 145    23      14  9.2   71     9  22
## 146    36     139 10.3   81     9  23
## 147     7      49 10.3   69     9  24
## 148    14      20 16.6   63     9  25
## 149    30     193  6.9   70     9  26
## 150    NA     145 13.2   77     9  27
## 151    14     191 14.3   75     9  28
## 152    18     131  8.0   76     9  29
## 153    20     223 11.5   68     9  30
```

```r
summary(dataframe)  ## shows some summarised data
```

```
##      Ozone           Solar.R           Wind             Temp      
##  Min.   :  1.00   Min.   :  7.0   Min.   : 1.700   Min.   :56.00  
##  1st Qu.: 18.00   1st Qu.:115.8   1st Qu.: 7.400   1st Qu.:72.00  
##  Median : 31.50   Median :205.0   Median : 9.700   Median :79.00  
##  Mean   : 42.13   Mean   :185.9   Mean   : 9.958   Mean   :77.88  
##  3rd Qu.: 63.25   3rd Qu.:258.8   3rd Qu.:11.500   3rd Qu.:85.00  
##  Max.   :168.00   Max.   :334.0   Max.   :20.700   Max.   :97.00  
##  NA's   :37       NA's   :7                                       
##      Month            Day      
##  Min.   :5.000   Min.   : 1.0  
##  1st Qu.:6.000   1st Qu.: 8.0  
##  Median :7.000   Median :16.0  
##  Mean   :6.993   Mean   :15.8  
##  3rd Qu.:8.000   3rd Qu.:23.0  
##  Max.   :9.000   Max.   :31.0  
## 
```

```r
table(dataframe$Month)  ## tabulates categorical data
```

```
## 
##  5  6  7  8  9 
## 31 30 31 31 30
```

```r
str(dataframe)  ## shows some other summarised data
```

```
## 'data.frame':    153 obs. of  6 variables:
##  $ Ozone  : int  41 36 12 18 NA 28 23 19 8 NA ...
##  $ Solar.R: int  190 118 149 313 NA NA 299 99 19 194 ...
##  $ Wind   : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
##  $ Temp   : int  67 72 74 62 56 66 65 59 61 69 ...
##  $ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
##  $ Day    : int  1 2 3 4 5 6 7 8 9 10 ...
```

Swirl 13: Simulation

```r
sample(1:6, 4, replace = TRUE) ## simulate rolling 4 dice
```

```
## [1] 4 6 1 5
```

A **binomial random variable** represents the number of successes in a given number of independent trials. Therefore, we can generate a single random variable that represents the number of heads in 100 flips of a fair coin using rbinom(1, size = 100, prob = 0.5). Note that you only specify the probability of 'success' (heads) and NOT the probability of 'failure' (tails).

```r
rbinom(1, size = 100, prob = 0.5)
```

```
## [1] 56
```

```r
rbinom(10, size = 100, prob = 0.5) ## repeats simulation 10 times
```

```
##  [1] 52 41 49 48 47 53 52 47 44 46
```

```r
rbinom(100, size = 1, prob = 0.5) ## shows each of the 100 coinflips
```

```
##   [1] 0 1 1 1 1 1 0 1 1 1 0 1 1 0 1 0 1 0 1 1 1 1 0 0 1 1 1 1 1 1 0 1 0 0 0 0 0
##  [38] 1 1 1 0 0 0 1 1 1 1 0 1 1 0 0 1 1 0 0 0 1 0 1 0 0 0 0 1 1 0 1 0 1 1 1 0 0
##  [75] 1 0 1 1 0 0 1 0 0 0 1 1 0 0 1 1 0 0 0 1 1 1 1 1 1 0
```

```r
sample(0:1, 100, replace = TRUE) ## identical to last command
```

```
##   [1] 1 0 1 0 0 1 1 0 1 0 0 1 1 1 1 0 1 0 0 1 1 0 1 1 1 0 0 1 1 1 1 0 0 1 1 1 1
##  [38] 1 0 0 1 0 0 1 0 1 1 1 1 0 1 1 0 0 1 0 1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 1 0 1 0
##  [75] 0 1 0 1 0 0 0 1 0 1 1 0 0 1 1 1 0 1 1 0 1 0 1 1 0 0
```

The standard **normal distribution** has mean 0 and standard deviation 1. The default values for the 'mean' and 'sd' arguments to rnorm() are 0 and 1, respectively. Thus, `rnorm(10)` will generate 10 random numbers from a standard normal distribution.

```r
rnorm(10)
```

```
##  [1] -0.44410204  1.22368429 -0.72602911  1.94510988  0.94878656  0.72589370
##  [7]  0.08198657 -0.41133967 -0.03299439 -0.86353703
```

The **Poisson distribution** expresses the probability of a given number of events occurring in a fixed interval, if these events occur with a known constant mean rate and independently of the time since the last event. For instance, the amount of mail someone receive each day may have an average number of 4 letters per day. If receiving any particular piece of mail does not affect the arrival times of future pieces of mail, then a reasonable assumption is that it obeys a Poisson distribution. Other examples include the number of phone calls received by a call center per hour and the number of decay events per second from a radioactive source.

```r
rpois(5, 4) ## average events per interval is 4, run 5 simulations
```

```
## [1] 7 4 4 2 4
```

Replicating random simlations to create a matrix

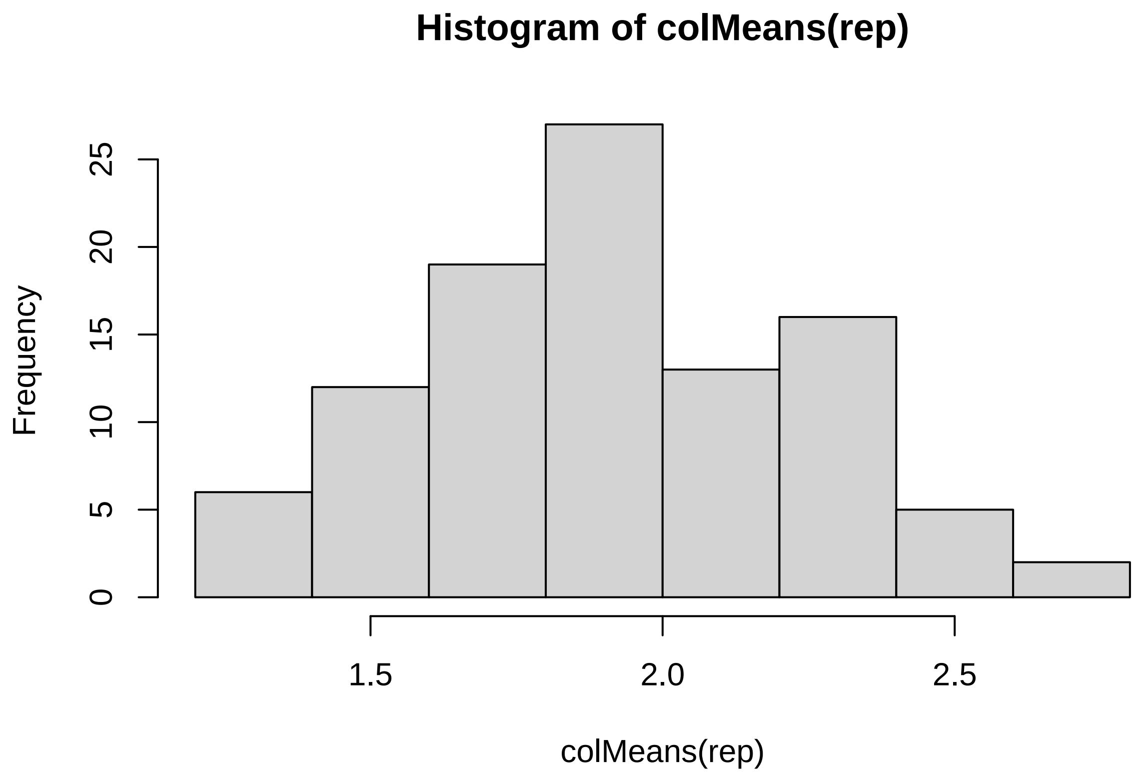
```r
rep <- replicate(100, rbinom(10, 5, prob = 0.4))
rep
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
##  [1,]    3    2    3    2    1    3    3    3    1     1     1     1     1
##  [2,]    0    1    4    2    1    1    3    3    5     4     2     3     2
##  [3,]    3    1    1    3    1    3    3    1    1     3     2     3     3
##  [4,]    4    2    2    3    0    2    0    2    4     1     1     3     1
##  [5,]    3    1    1    0    2    1    1    4    1     1     3     2     2
##  [6,]    4    3    2    3    1    1    2    3    2     4     2     1     1
##  [7,]    3    2    0    1    2    0    2    1    1     3     3     2     3
```

```
##  [8,]    3    3    2    4    3    2    0    2    0     1     2     3     2
##  [9,]    2    2    1    3    2    1    2    3    3     3     4     2     2
## [10,]    1    2    0    4    1    1    2    3    2     3     2     3     0
##       [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
##  [1,]     2     2     0     2     1     1     1     3     4     3     2     2
##  [2,]     2     1     0     2     2     3     1     2     1     4     2     2
##  [3,]     1     2     2     3     2     3     0     3     0     2     2     3
##  [4,]     1     4     3     3     1     2     3     1     2     3     2     1
##  [5,]     1     3     2     2     1     2     1     2     2     0     3     2
##  [6,]     3     2     4     0     2     3     1     3     3     3     2     2
##  [7,]     2     1     2     2     1     1     0     3     3     1     2     1
##  [8,]     0     2     3     3     1     4     4     0     3     2     3     1
##  [9,]     2     1     4     3     2     0     2     1     0     1     1     1
## [10,]     2     2     4     1     2     1     2     2     1     1     0     1
##       [,26] [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37]
##  [1,]     2     2     4     4     1     3     1     3     2     1     1     5
##  [2,]     1     0     4     1     2     0     4     1     1     1     1     4
##  [3,]     2     2     1     1     0     2     1     2     4     1     2     3
##  [4,]     0     0     2     4     2     2     3     2     0     4     0     1
##  [5,]     4     0     2     2     1     2     2     3     2     2     2     2
##  [6,]     1     0     3     2     0     1     3     2     1     1     2     2
##  [7,]     2     3     2     2     2     2     3     3     3     2     3     1
##  [8,]     0     1     0     2     3     2     2     1     2     3     3     1
##  [9,]     4     2     2     1     2     0     3     1     3     1     2     1
## [10,]     3     3     2     1     3     1     0     3     2     4     3     3
##       [,38] [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49]
##  [1,]     2     2     1     3     4     2     1     3     1     2     1     1
##  [2,]     1     2     2     1     1     3     1     2     3     3     3     0
##  [3,]     0     4     3     0     0     1     2     0     2     1     2     2
##  [4,]     2     1     2     3     3     0     2     1     2     2     2     2
##  [5,]     1     1     3     2     1     3     2     3     1     1     2     1
##  [6,]     1     4     4     1     3     2     1     2     3     2     1     1
##  [7,]     3     1     1     0     2     1     3     1     1     2     1     4
##  [8,]     1     2     4     2     2     0     3     3     2     1     2     3
##  [9,]     3     2     2     0     2     2     1     1     2     2     2     3
## [10,]     2     1     2     2     0     4     2     2     3     2     3     2
##       [,50] [,51] [,52] [,53] [,54] [,55] [,56] [,57] [,58] [,59] [,60] [,61]
##  [1,]     1     1     1     1     2     1     3     3     2     0     2     1
##  [2,]     3     3     2     3     2     4     2     2     2     1     3     3
##  [3,]     2     2     1     0     1     1     1     2     0     3     1     3
##  [4,]     3     1     2     2     1     3     3     0     2     2     4     4
##  [5,]     1     2     2     1     3     1     3     2     3     2     1     2
##  [6,]     0     2     1     3     0     1     0     4     2     3     2     1
##  [7,]     0     4     3     2     3     2     1     1     1     1     1     3
##  [8,]     0     3     2     0     0     1     0     2     2     2     1     1
##  [9,]     2     1     0     4     3     2     4     2     2     1     1     0
## [10,]     1     1     4     1     2     3     1     2     2     2     4     2
##       [,62] [,63] [,64] [,65] [,66] [,67] [,68] [,69] [,70] [,71] [,72] [,73]
##  [1,]     1     4     3     3     4     2     1     2     0     2     3     1
##  [2,]     2     3     4     1     3     0     3     2     4     2     1     3
##  [3,]     2     3     0     2     4     3     3     1     4     1     2     3
##  [4,]     3     3     2     2     0     1     1     4     3     1     4     3
##  [5,]     4     1     2     3     4     2     1     1     4     3     4     2
##  [6,]     2     1     3     3     1     3     1     1     1     1     1     2
```

```
##  [7,]     3     4     4     1     4     3     2     2     2     2     3     1
##  [8,]     2     2     1     3     3     2     0     3     3     1     1     1
##  [9,]     0     1     2     1     2     3     1     1     2     2     1     1
## [10,]     2     2     2     2     0     3     1     4     3     3     4     1
##       [,74] [,75] [,76] [,77] [,78] [,79] [,80] [,81] [,82] [,83] [,84] [,85]
##  [1,]     1     4     2     2     2     2     2     3     3     2     2     3
##  [2,]     1     3     4     3     2     2     3     3     2     2     2     1
##  [3,]     3     0     3     3     2     1     1     1     4     1     1     2
##  [4,]     1     2     1     2     1     4     1     3     1     3     4     1
##  [5,]     1     0     0     2     3     3     3     2     1     2     0     4
##  [6,]     1     4     1     2     2     2     4     2     1     1     0     3
##  [7,]     2     3     2     3     4     2     1     1     2     3     4     3
##  [8,]     1     2     3     2     3     2     3     2     0     4     1     2
##  [9,]     1     1     3     2     1     2     1     1     1     2     4     2
## [10,]     2     1     2     2     1     3     3     1     2     4     2     3
##       [,86] [,87] [,88] [,89] [,90] [,91] [,92] [,93] [,94] [,95] [,96] [,97]
##  [1,]     4     3     1     2     3     2     3     0     3     1     1     1
##  [2,]     3     3     2     2     1     3     3     3     3     4     1     0
##  [3,]     2     0     1     1     2     1     1     3     3     4     2     2
##  [4,]     2     2     3     1     4     3     3     3     2     2     3     2
##  [5,]     1     3     3     2     3     2     2     2     4     3     3     3
##  [6,]     2     1     1     3     3     2     3     1     1     5     2     1
##  [7,]     3     1     2     1     3     3     1     3     3     2     1     4
##  [8,]     2     1     1     2     2     1     2     1     2     3     1     1
##  [9,]     1     2     2     2     2     2     2     4     1     2     0     0
## [10,]     0     1     0     2     1     0     4     3     2     2     1     1
##       [,98] [,99] [,100]
##  [1,]     2     3      4
##  [2,]     2     3      0
##  [3,]     3     3      1
##  [4,]     0     2      2
##  [5,]     4     3      2
##  [6,]     2     2      3
##  [7,]     1     3      2
##  [8,]     1     3      2
##  [9,]     2     3      3
## [10,]     1     2      3
```

```r
hist(colMeans(rep)) ## generate a histogram of means
```



Notice the distribution is approximately normal, as it is a distribution of means.

Other distributions with random samplers:

```
str(runif)
```

```
## function (n, min = 0, max = 1)
```

```
 ## generates n random numbers, with minimum and max set, based on a
 ## uniform distribution. d, p and q functions also exist for this
 ## distribution
str(rpois)
```

```
## function (n, lambda)
```

```
 ## generates n random numbers, with mean number of events per
 ## interval equal to lambda, based on a poisson distribution. d, p
 ## and q functions also exist for this distribution.
str(rbinom)
```

```
## function (n, size, prob)
```

```
 ## generates `size` coin tosses with probability `prob` of success
 ## and sums up the successes, the repeats the process `n` times and
 ## outputs a vector of the results. d, p and q functions also exist
str(rexp)  ## exponential
```

```
## function (n, rate = 1)
```

```r
str(rchisq)  ## chi-squared
```

```
## function (n, df, ncp = 0)
```

```r
str(rgamma)  ## gamma
```

```
## function (n, shape, rate = 1, scale = 1/rate)
```

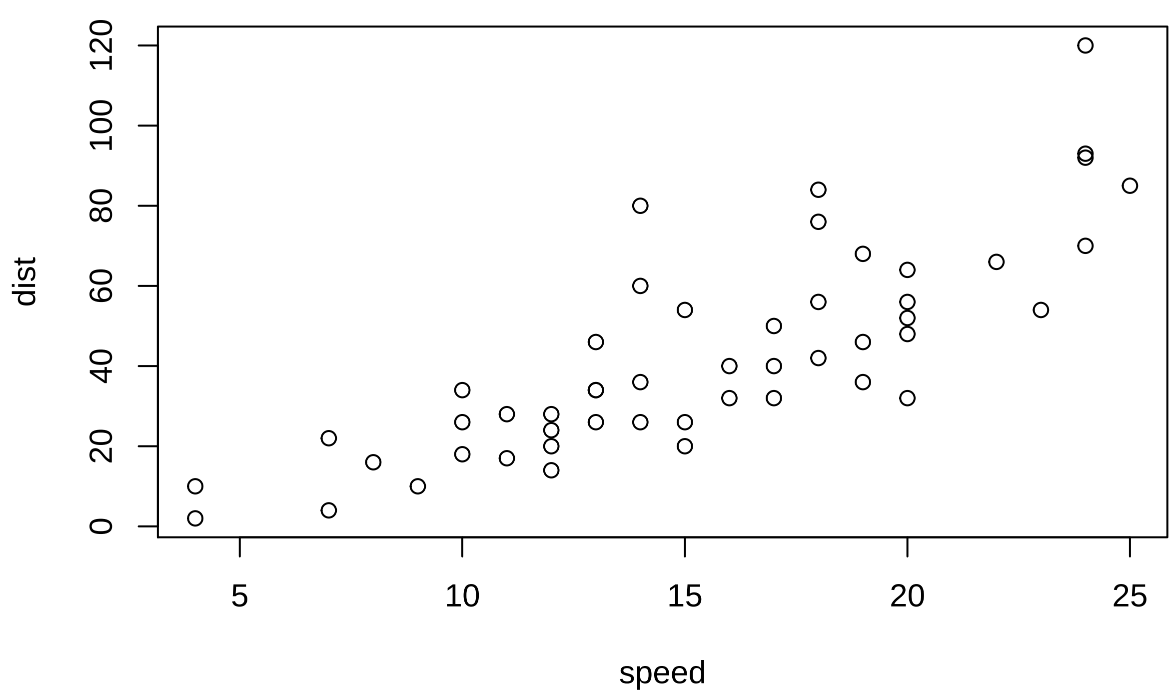
Swirl Exercise 15: Base Graphics

Note: read http://varianceexplained.org/r/teach_ggplot2_to_beginners/ for learning to plot with ggplot2. For learning more elements of base graphics try http://www.ling.upenn.edu/~joseff/rstudy/week4.html.

Before plotting, it's good to get a sense of the data with summary commands such as `dim()`, `names()`, `head()`, `tail()`, `summary()` and `str()`.

plot is short for *scatterplot*

```r
data(cars)
plot(cars)
```



R notes that the data frame you have given it has just two columns, so it assumes that you want to plot one column versus the other. Since we do not provide labels for either axis, R uses the names of the columns. It creates axis tick marks at nice round numbers and labels them accordingly. It also uses other defaults supplied in `plot()`.

The `plot()` command has many more parameters that aren't listed under `?plot`. Find more options in `?par` and `?points`.

- `main` adds a main title at the top of the graph
- `sub`adds a subtitle at the bottom
- `xlab` and `ylab` chages the axis labels
- `col` changes the colour of the dots
- `pch` changes the shape of the dots

Because these arguments aren't always intuitive, many people use packages such as ggplot2 for creating graphics in R.

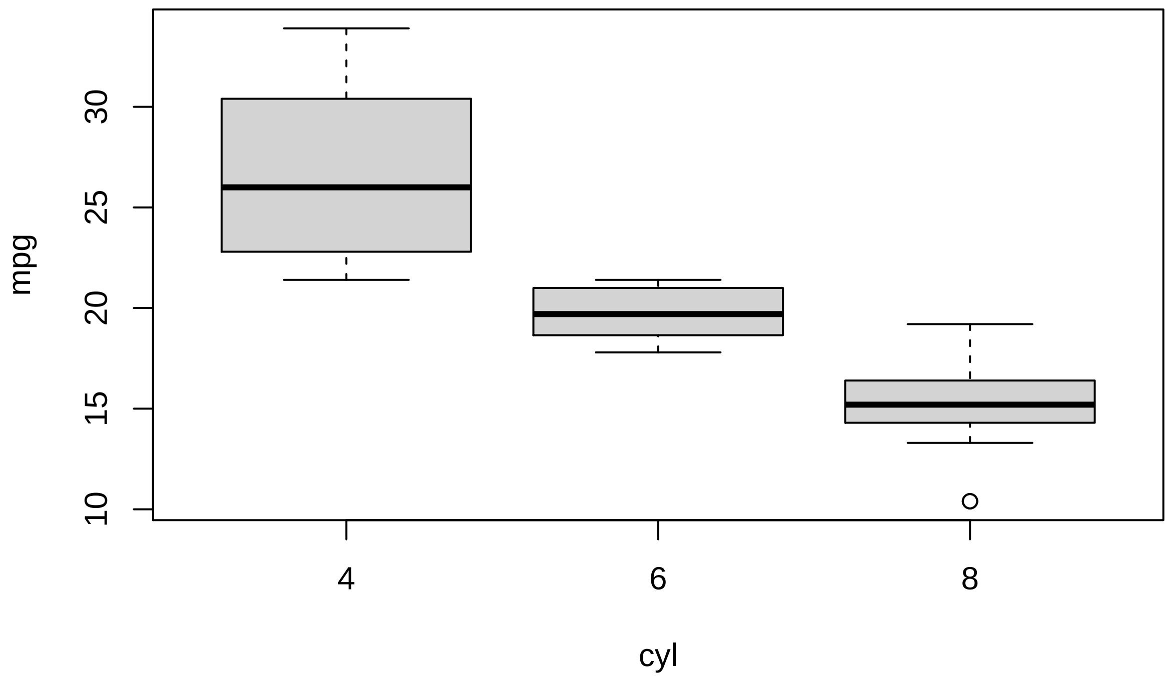
formula argument

The `formula =` argument is generally an expression with a tilde ("~") which indicates the relationship between the input variables. This allows you to enter something like x ~ y to plot the relationship between x on the x-axis and y on the y-axis. May need to include a `data =` argument, too.

Box and Whiskers: `boxplot()`

Instead of adding data columns directly as input arguments, as we did with `plot()`, it is often handy to pass in the entire data frame. This is what the `data` argument in `boxplot()` allows.

```
data(mtcars)
boxplot(formula = mpg ~ cyl, data = mtcars)
```

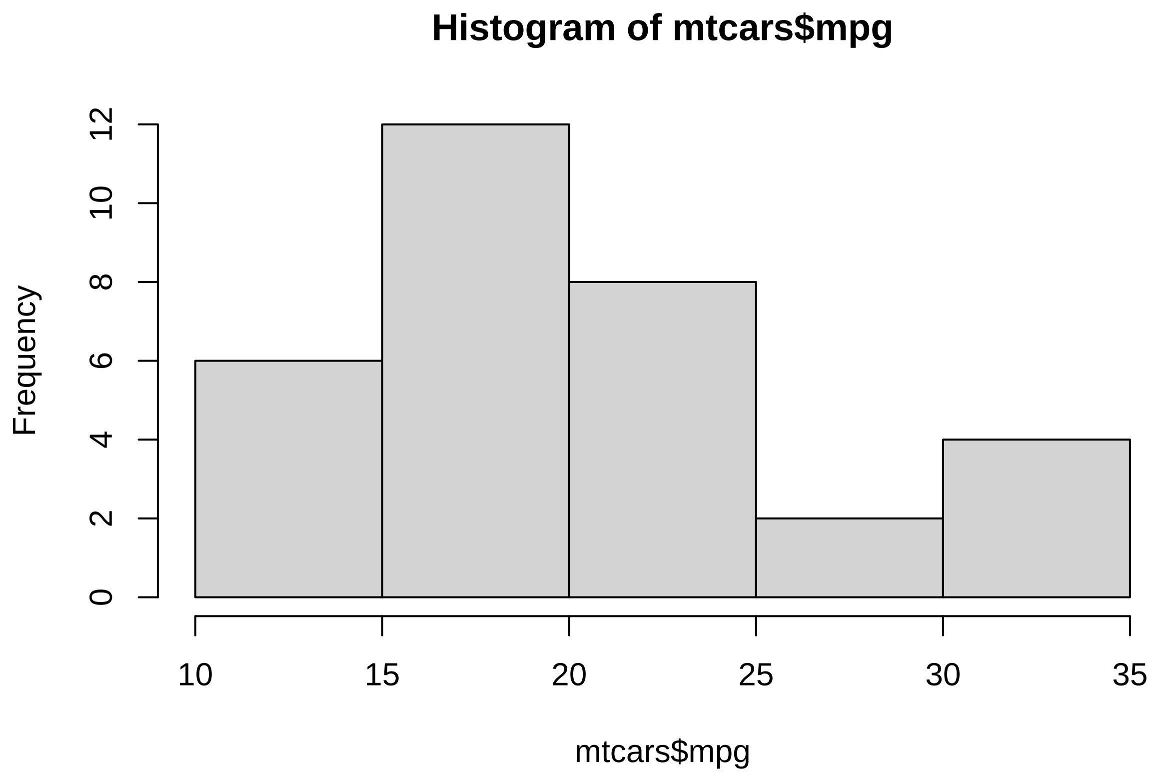


Axis labels, titles, etc can be added with the same arguments as for `plot()`.

Histograms: `hist()`

When looking at a single variable, histograms are a useful tool. Like `plot()`, `hist()` is best used by just passing in a single vector.

```
hist(mtcars$mpg)
```



Histogram of mtcars$mpg
Frequency
0
2
4
6
8
10
12
10
15
20
25
30
35
mtcars$mpg