**Creating Vector**

> y <- vector(length=2)

> y[1] <- 5

> y[2] <- 12

The following will also work:

> y <- c(5,12)

**Recycling**

> c(1,2,4) + c(6,0,9,20,22)

[1] 7 2 13 21 24

Warning message:

longer object length is not a multiple of shorter object length in: c(1, 2, 4) + c(6,0, 9, 20, 22)

Beware of operator precedence issues.

> i <- 2

> 1:i-1 # this means (1:i) - 1, not 1:(i-1)

[1] 0 1

> 1:(i-1)

[1] 1

In the expression 1:i-1, the colon operator takes precedence over the subtraction.

Problem:

Suppose we observe 0- and 1-valued data, one per time period. To make things concrete, say it’s daily weather data: 1 for rain and 0 for no rain. Suppose we wish to predict whether it will rain tomorrow, knowing whether it rained or not in recent days. Specifically, for some number k, we will predict tomorrow’s weather based on the weather record of the last k days. We’ll use majority rule: If the number of 1s in the previous k time periods is at least k/2, we’ll predict the next value to be 1; otherwise, our prediction is 0. For instance, if k = 3 and the data for the last three periods is 1,0,1, we’ll predict the next period to be a 1.

***Repeating Vector Constants with rep()***

The rep() (or *repeat*) function allows us to conveniently put the same constant into long vectors. The call form is rep(x,times), which creates a vector of *times\*length(x)* elements—that is, times copies of x. Here is an example:

> x <- rep(8,4)

> x

[1] 8 8 8 8

> rep(c(5,12,13),3)

[1] 5 12 13 5 12 13 5 12 13

> rep(1:3,2)

[1] 1 2 3 1 2 3

There is also a named argument each, with very different behavior, which interleaves the copies of x.

> rep(c(5,12,13),each=2)

[1] 5 5 12 12 13 13

The any() and all() functions are handy shortcuts. They report whether any or all of their arguments are TRUE.

> x <- 1:10

> any(x > 8)

[1] TRUE

> any(x > 88)

[1] FALSE

> all(x > 88)

[1] FALSE

> all(x > 0)

[1] TRUE

> x <- 1:3

> y <- c(1,3,4)

> x == y

[1] TRUE FALSE FALSE

> all(x == y)

[1] FALSE

**NA and NULL Values**

R actually has two such values: NA and NULL.

In statistical data sets, we often encounter missing data, which we represent in R with the value NA. NULL, on the other hand, represents that the value in question simply doesn’t exist, rather than being existent but unknown. Let’s see how this comes into play in concrete terms.

Using NA

In many of R’s statistical functions, we can instruct the function to skip over

any missing values, or NAs. Here is an example:

> x <- c(88,NA,12,168,13)

> x

[1] 88 NA 12 168 13

> mean(x)

[1] NA

> mean(x,na.rm=T)

[1] 70.25

> x <- c(88,NULL,12,168,13)

> mean(x)

[1] 70.25

Remove na

vy

# 1 2 3 NA 5

vy[ !is.na(vy) ]

# 1 2 3 5

vz

# 1 2 3 NaN 5

vz[ !is.nan(vz) ]

# 1 2 3 5

**Filtering with the subset() Function**

Filtering can also be done with the subset() function. When applied to vectors, the difference between using this function and ordinary filtering lies in the manner in which NA values are handled.

> x <- c(6,1:3,NA,12)

> x

[1] 6 1 2 3 NA 12

> x[x > 5]

[1] 6 NA 12

> subset(x, x > 5)

[1] 6 12

When we did ordinary filtering in the previous section, R basically said,“Well, x[5] is unknown, so it’s also unknown whether its square is greater than 5.” But you may not want NAs in your results. When you wish to exclude NA values, using subset() saves you the trouble of removing the NA values yourself.

**The Selection Function which()**

As you’ve seen, filtering consists of extracting elements of a vector z that satisfy a certain condition. In some cases, though, we may just want to find the positions within z at which the condition occurs. We can do this using which(), as follows:

> z <- c(5,2,-3,8)

> which(z\*z > 8)

[1] 1 3 4

The result says that elements 1, 3, and 4 of z have squares greater than 8. As with filtering, it is important to understand exactly what occurred in the preceding code. The expression

z\*z > 8

> x <- 1:10

> y <- ifelse(x %% 2 == 0,5,12) # %% is the mod operator

> y

[1] 12 5 12 5 12 5 12 5 12 5

Problem:

Consider vectors x and y, which are time series, say for measurements of air temperature and pressure collected once each hour. We’ll define our measure of association between them to be the fraction of the time x and y increase or decrease together—that is, the proportion of i for which y[i+1]-y[i] has the same sign as x[i+1]-x[i].

Due to the vector nature of the arguments, you can nest ifelse() operations.

In the following example, which involves an abalone data set, gender is coded as M, F, or I (for infant). We wish to recode those characters as 1, 2, or 3. The real data set consists of more than 4,000 observations, but for our example, we’ll say we have just a few, stored in g:

> g

[1] "M" "F" "F" "I" "M" "M" "F"

> ifelse(g == "M",1,ifelse(g == "F",2,3))

[1] 1 2 2 3 1 1 2

**Sorting**

# sorting examples using the mtcars dataset  
attach(mtcars)  
  
# sort by mpg  
newdata <- mtcars[order(mpg),]   
  
# sort by mpg and cyl  
newdata <- mtcars[order(mpg, cyl),]  
  
#sort by mpg (ascending) and cyl (descending)  
newdata <- mtcars[order(mpg, -cyl),]   
  
detach(mtcars)

**Factors**

Factors form the basis for many of R’s powerful operations, including many of those performed on tabular data. The motivation for factors comes from the notion of nominal, or categorical, variables in statistics. These values are nonnumerical in nature, corresponding to categories such as Democrat, Republican, and naffiliated, although they may be coded using numbers.

An R *factor* might be viewed simply as a vector with a bit more information added (though, as seen below, it’s different from this internally). That extra information consists of a record of the distinct values in that vector, called *levels*. Here’s an example:

> x <- c(5,12,13,12)

> xf <- factor(x)

> xf

[1] 5 12 13 12

Levels: 5 12 13

The distinct values in xf—5, 12, and 13—are the levels here.

Let’s take a look inside:

> str(xf)

Factor w/ 3 levels "5","12","13": 1 2 3 2

> unclass(xf)

[1] 1 2 3 2

attr(,"levels")

[1] "5" "12" "13"

This is revealing. The core of xf here is not (5,12,13,12) but rather (1,2,3,2). The latter means that our data consists first of a level-1 value, then level-2 and level-3 values, and finally another level-2 value. So the data has been recoded by level. The levels themselves are recorded too, of course, though as characters such as "5" rather than 5. The length of a factor is still defined in terms of the length of the data rather than, say, being a count of the number of levels:

> length(xf)

[1] 4

We can anticipate future new levels, as seen here:

> x <- c(5,12,13,12)

> xff <- factor(x,levels=c(5,12,13,88))

> xff

[1] 5 12 13 12

Levels: 5 12 13 88

> xff[2] <- 88

> xff

[1] 5 88 13 12

Levels: 5 12 13 88

Originally, xff did not contain the value 88, but in defining it, we allowed for that future possibility. Later, we did indeed add the value. By the same token, you cannot sneak in an “illegal” level. Here’s what happens when you try:

> xff[2] <- 28

Warning message:

In `[<-.factor`(`\*tmp\*`, 2, value = 28) :

invalid factor level, NAs generated

**tapply( )**

> ages <- c(25,26,55,37,21,42)

> affils <- c("R","D","D","R","U","D")

> tapply(ages,affils,mean)

D R U

41 31 21

d <- data.frame(list(gender=c("M","M","F","M","F","F"),

+ age=c(47,59,21,32,33,24),income=c(55000,88000,32450,76500,123000,45650)))

> d

gender age income

1 M 47 55000

2 M 59 88000

3 F 21 32450

4 M 32 76500

5 F 33 123000

6 F 24 45650

> d$over25 <- ifelse(d$age > 25,1,0)

> d

gender age income over25

1 M 47 55000 1

2 M 59 88000 1

3 F 21 32450 0

4 M 32 76500 1

5 F 33 123000 1

6 F 24 45650 0

> tapply(d$income,list(d$gender,d$over25),mean)

0 1

F 39050 123000.00

M NA 73166.67

# aggregate data frame mtcars by cyl and vs, returning means  
# for numeric variables  
attach(mtcars)  
aggdata <-aggregate(mtcars, by=list(cyl,vs),   
  FUN=mean, na.rm=TRUE)  
print(aggdata)  
detach(mtcars)

## Compute the averages according to region and the occurrence of more

## than 130 days of frost.

aggregate(state.x77,

list(Region = state.region,

Cold = state.x77[,"Frost"] > 130),

mean)

## (Note that no state in 'South' is THAT cold.)

## example with character variables and NAs

testDF <- data.frame(v1 = c(1,3,5,7,8,3,5,NA,4,5,7,9),

v2 = c(11,33,55,77,88,33,55,NA,44,55,77,99) )

by1 <- c("red", "blue", 1, 2, NA, "big", 1, 2, "red", 1, NA, 12)

by2 <- c("wet", "dry", 99, 95, NA, "damp", 95, 99, "red", 99, NA, NA)

aggregate(x = testDF, by = list(by1, by2), FUN = "mean")

# and if you want to treat NAs as a group

fby1 <- factor(by1, exclude = "")

fby2 <- factor(by2, exclude = "")

aggregate(x = testDF, by = list(fby1, fby2), FUN = "mean")

## Formulas, one ~ one, one ~ many, many ~ one, and many ~ many:

aggregate(weight ~ feed, data = chickwts, mean)

aggregate(breaks ~ wool + tension, data = warpbreaks, mean)

aggregate(cbind(Ozone, Temp) ~ Month, data = airquality, mean)

aggregate(cbind(ncases, ncontrols) ~ alcgp + tobgp, data = esoph, sum)

## Dot notation:

aggregate(. ~ Species, data = iris, mean)

aggregate(len ~ ., data = ToothGrowth, mean)

# merge two data frames by ID  
total <- merge(data frameA,data frameB,by="ID")

# merge two data frames by ID and Country  
total <- merge(data frameA,data frameB,by=c("ID","Country"))

x <- sample(0:20, 100, replace=TRUE)

x

Now cut x at 0, 10 and 20:

cut(x, breaks=c(0, 10, 20), include.lowest=TRUE)

[1] (10,20] [0,10] [0,10] (10,20] (10,20] (10,20] [0,10] (10,20] (10,20]

[10] (10,20] [0,10] (10,20] (10,20] (10,20] [0,10] (10,20] [0,10] [0,10]

[19] [0,10] (10,20] [0,10] [0,10] [0,10] (10,20] [0,10] (10,20] (10,20]

[28] (10,20] (10,20] [0,10] [0,10] [0,10] [0,10] (10,20] [0,10] [0,10]

[37] [0,10] [0,10] (10,20] (10,20] (10,20] (10,20] [0,10] (10,20] [0,10]

[46] (10,20] [0,10] (10,20] (10,20] [0,10] [0,10] (10,20] (10,20] (10,20]

[55] [0,10] [0,10] (10,20] [0,10] [0,10] [0,10] [0,10] (10,20] (10,20]

[64] (10,20] [0,10] [0,10] (10,20] (10,20] (10,20] (10,20] (10,20] (10,20]

[73] (10,20] [0,10] [0,10] [0,10] (10,20] [0,10] (10,20] [0,10] (10,20]

[82] [0,10] [0,10] (10,20] [0,10] [0,10] [0,10] (10,20] (10,20] [0,10]

[91] [0,10] [0,10] (10,20] (10,20] [0,10] [0,10] [0,10] [0,10] (10,20]

[100] (10,20]

Levels: [0,10] (10,20]

> a <- runif(100)

> cut(a, seq(from = 0, to = 1, by = 0.2))

**Bar Graph**

A bar graph of a qualitative data sample consists of vertical parallel bars that shows the frequency distribution graphically.

**Example**

In the data set [painters](http://www.r-tutor.com/node/19), the bar graph of the School variable is a collection of vertical bars showing the number of painters in each school.

**Problem**

Find the bar graph of the painter schools in the data set painters.

**Solution**

We first apply the table function to compute the frequency distribution of the School variable.

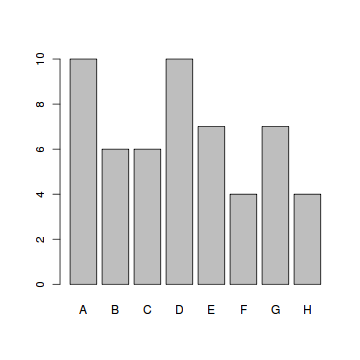
> library(MASS)                 # load the MASS package   
> school = painters$School      # the painter schools   
> school.freq = table(school)   # apply the table function

Then we apply the barplot function to produce its bar graph.

> barplot(school.freq)         # apply the barplot function

**Answer**

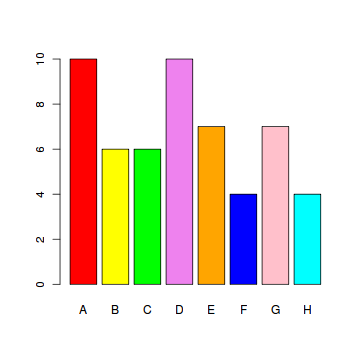
The bar graph of the school variable is:



**Enhanced Solution**

To colorize the bar graph, we select a color palette and set it in the col argument of barplot.

> colors = c("red", "yellow", "green", "violet",   
+   "orange", "blue", "pink", "cyan")   
> barplot(school.freq,         # apply the barplot function   
+   col=colors)                # set the color palette



**Exercise**

Find the bar graph of the composition scores in painters.

**Pie Chart**

A pie chart of a qualitative data sample consists of pizza wedges that shows the frequency distribution graphically.

**Example**

In the data set [painters](http://www.r-tutor.com/node/19), the pie chart of the School variable is a collection of pizza wedges showing the proportion of painters in each school.

**Problem**

Find the pie chart of the painter schools in the data set painters.

**Solution**

We first apply the table function to produce the frequency distribution of School.

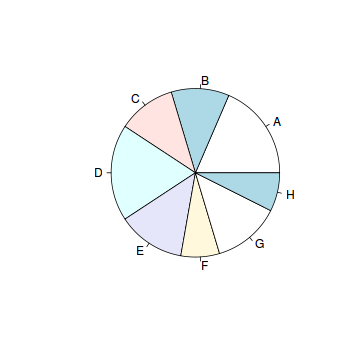
> library(MASS)                 # load the MASS package   
> school = painters$School      # the painter schools   
> school.freq = table(school)   # apply the table function

Then we apply the pie function to produce its pie chart.

> pie(school.freq)              # apply the pie function

**Answer**

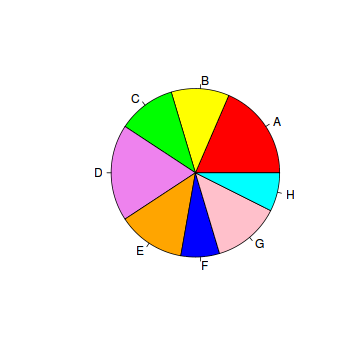
The pie chart of the school variable is:



**Enhanced Solution**

To colorize the pie chart, we select a color palette and set it in the col argument of pie.

> colors = c("red", "yellow", "green", "violet",   
+   "orange", "blue", "pink", "cyan")   
> pie(school.freq,             # apply the pie function   
+   col=colors)                # set the color palette



**Exercise**

Find the pie chart of the composition scores in painters.

#### Problem

Find out the mean composition score of school C in the data set painters.

> tapply(painters$Composition, painters$School, mean)   
     A      B      C      D      E      F      G      H   
10.400 12.167 13.167  9.100 13.571  7.250 13.857 14.000

#### Exercise

1. Find programmatically the school with the highest composition scores.
2. Find the percentage of painters whose color score is equal to or above 14.

# Frequency Distribution of Quantitative Data

The frequency distribution of a data variable is a summary of the data occurrence in a collection of non-overlapping categories.

#### Example

In the data set [faithful](http://www.r-tutor.com/node/25), the frequency distribution of the eruptions variable is the summary of eruptions according to some classification of the eruption durations.

#### Problem

Find the frequency distribution of the eruption durations in faithful.

#### Solution

The solution consists of the following steps:

1. We first find the range of eruption durations with the range function. It shows that the observed eruptions are between 1.6 and 5.1 minutes in duration.

> duration = faithful$eruptions   
> range(duration)   
[1] 1.6 5.1

1. Break the range into non-overlapping sub-intervals by defining a sequence of equal distance break points. If we round the endpoints of the interval [1.6, 5.1] to the closest half-integers, we come up with the interval [1.5, 5.5]. Hence we set the break points to be the half-integer sequence { 1.5, 2.0, 2.5, ... }.

> breaks = seq(1.5, 5.5, by=0.5)    # half-integer sequence   
> breaks   
[1] 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5

1. Classify the eruption durations according to the half-unit-length sub-intervals with cut. As the intervals are to be closed on the left, and open on the right, we set the right argument as FALSE.

> duration.cut = cut(duration, breaks, right=FALSE)

1. Compute the frequency of eruptions in each sub-interval with the table function.

> duration.freq = table(duration.cut)

#### Answer

The frequency distribution of the eruption duration is:

> duration.freq   
duration.cut   
[1.5,2) [2,2.5) [2.5,3) [3,3.5) [3.5,4) [4,4.5) [4.5,5)   
     51      41       5       7      30      73      61   
[5,5.5)   
      4

#### Enhanced Solution

We apply the cbind function to print the result in column format.

> cbind(duration.freq)   
        duration.freq   
[1.5,2)            51   
[2,2.5)            41   
[2.5,3)             5   
[3,3.5)             7   
[3.5,4)            30   
[4,4.5)            73   
[4.5,5)            61   
[5,5.5)             4

#### Note

Per R documentation, you are advised to use the hist function to find the frequency distribution for performance reasons.

#### Exercise

1. Find the frequency distribution of the eruption waiting periods in faithful.
2. Find programmatically the duration sub-interval that has the most eruptions.

**Histogram**

A histogram consists of parallel vertical bars that graphically shows the frequency distribution of a quantitative variable. The area of each bar is equal to the frequency of items found in each class.

**Example**

In the data set [faithful](http://www.r-tutor.com/node/25), the histogram of the eruptions variable is a collection of parallel vertical bars showing the number of eruptions classified according to their durations.

**Problem**

Find the histogram of the eruption durations in faithful.

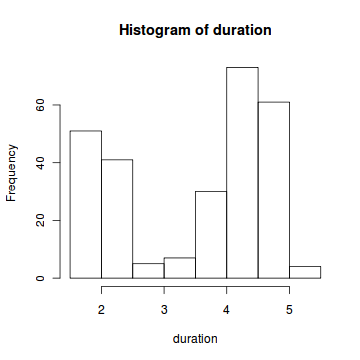
**Solution**

We apply the hist function to produce the histogram of the eruptions variable.

> duration = faithful$eruptions   
> hist(duration,    # apply the hist function   
+   right=FALSE)    # intervals closed on the left

**Answer**

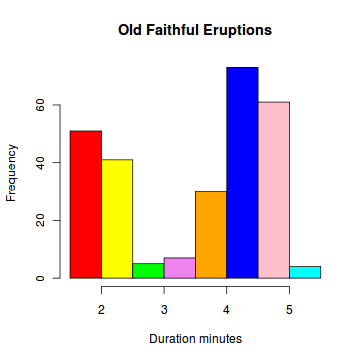
The histogram of the eruption durations is:



**Enhanced Solution**

To colorize the histogram, we select a color palette and set it in the col argument of hist. In addition, we update the titles for readability.

> colors = c("red", "yellow", "green", "violet", "orange",   
+   "blue", "pink", "cyan")   
> hist(duration,    # apply the hist function   
+   right=FALSE,    # intervals closed on the left   
+   col=colors,     # set the color palette   
+   main="Old Faithful Eruptions", # the main title   
+   xlab="Duration minutes")       # x-axis label



Bin width:

h = 2 \frac{\operatorname{IQR}(x)}{n^{1/3}},

The number of bins *k* can be assigned directly or can be calculated from a suggested bin width *h* as:

k = \left \lceil \frac{\max x - \min x}{h} \right \rceil.