

Introduction to R

Load the R package by double-clicking the **R** icon.

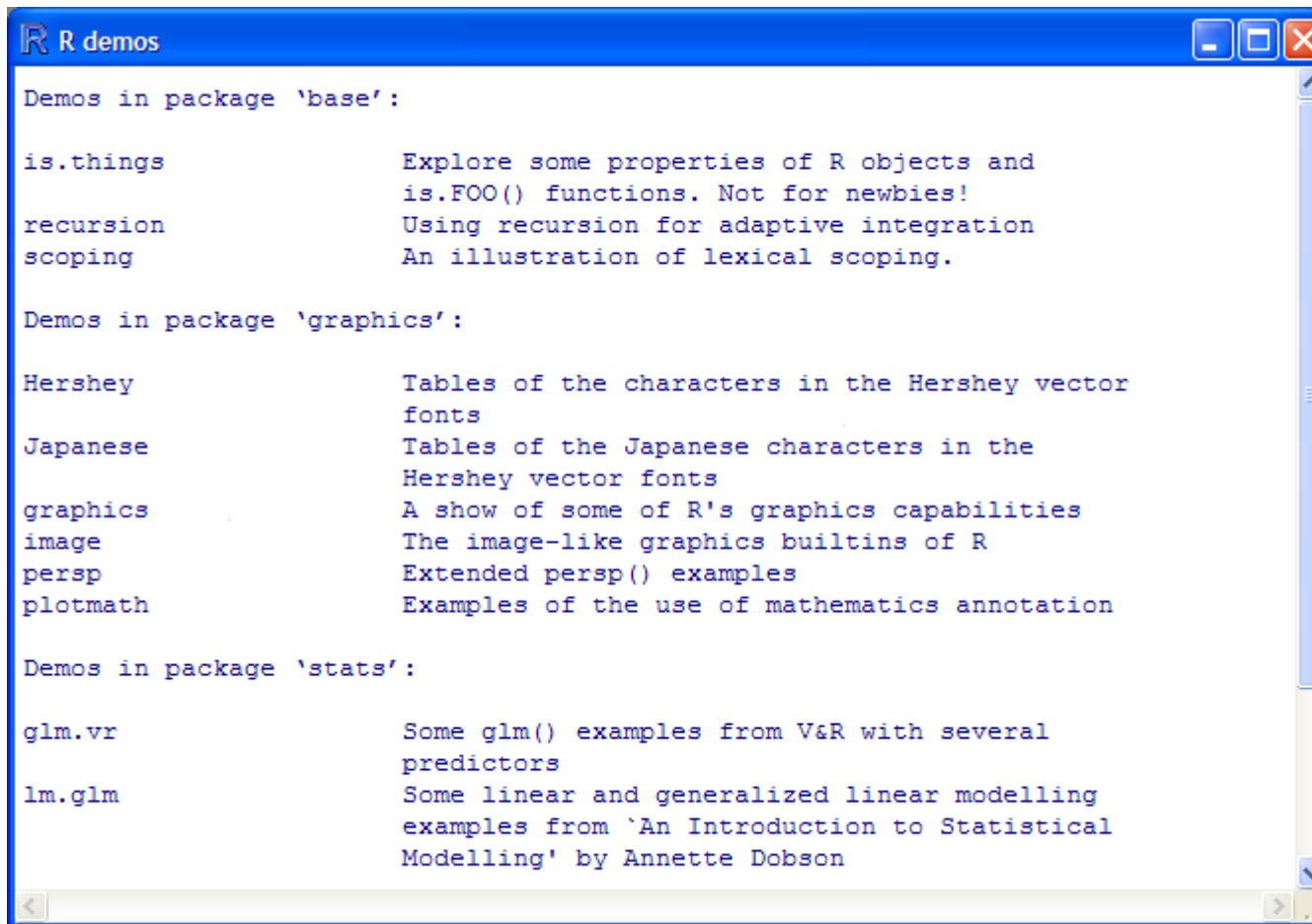
After some preliminary messages, you'll see the prompt.

```
>
```

To see some demos, type

```
> demo ( )
```

A demo window will open up



```
R R demos
Demos in package 'base':
is.things          Explore some properties of R objects and
                   is.FOO() functions. Not for newbies!
recursion          Using recursion for adaptive integration
scoping            An illustration of lexical scoping.

Demos in package 'graphics':
Hershey            Tables of the characters in the Hershey vector
                   fonts
Japanese           Tables of the Japanese characters in the
                   Hershey vector fonts
graphics           A show of some of R's graphics capabilities
image              The image-like graphics builtins of R
persp              Extended persp() examples
plotmath           Examples of the use of mathematics annotation

Demos in package 'stats':
glm.vr             Some glm() examples from V&R with several
                   predictors
lm.glm             Some linear and generalized linear modelling
                   examples from 'An Introduction to Statistical
                   Modelling' by Annette Dobson
```

Click back on the console window, and type

>demo(graphics)

You may need to hit the <Enter> key repeatedly to see the graphics. You will recognize one of them as a histogram.

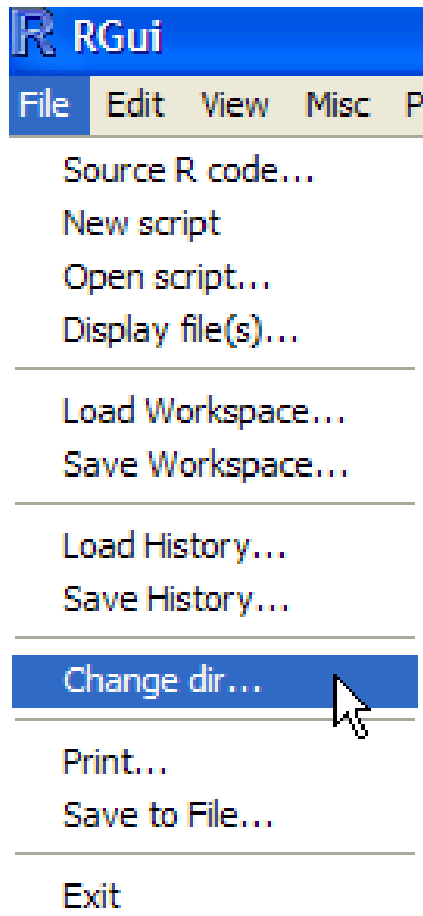
These graphs demonstrate just a small part of R's graphics capabilities.

R keeps a running record of your activity during a session. It is a good idea to create a special directory for each major R session and keep a record of what happens.

That way you can replicate the analysis.

So choose a directory, as follows.

Make sure the “focus” is on the R console. Then, from the *File* menu, select *Change dir*.



Choose a directory you won't forget.

R can be used in many ways.

To begin with, you can use it as a simple calculator. The key things to remember are that

- Multiplication is always indicated with a “*”
- Division is always indicated with a “/”
- Exponentiation is always indicated with a “^”

DO NOT FORGET!!

Examples.

$$2^3 = 8$$

```
> 2^3
```

```
[1] 8
```

$$\sqrt{\frac{4+5}{7}} = 1.133893$$

```
> sqrt( (4+5)/7 )
```

```
[1] 1.133893
```

Note how using spacing can make parentheses easier to read!!

Try these:

$$2\left(\frac{14+5}{36}\right)^3 = 0.2940243$$

Did you forget something?

Remember, if you make a mistake in R, you can backtrack to the previous lines using the up-arrow key. This can save a lot of time.

Now, let's enter some data.

There are several ways to do this. Let's begin by just entering some data and assigning it to the list *X*.

```
> x <- c(1,2,3,4,5)
```

The `c()` function concatenates a list of numbers into a vector. Once we have assigned this list of numbers to `x`, we can perform all kinds of operations on the list.

For example, if we want the sum of the numbers, we just enter `sum(x)`. Try it!

Suppose we want the mean of the x 's.

We can just enter

```
> mean(x)
```

```
[1] 3
```

We just finished getting the sum of the squares of the x values. It is a lot easier in R.

```
> sum(x^2)
```

```
[1] 55
```

Getting the variance and standard deviation of the **x** values is a snap.

```
> var(x)  
[1] 2.5
```

```
> sd(x)  
[1] 1.581139
```

Many advanced statistical functions are built into R. But what makes R special is that you can extend it to do many tasks very easily, by defining your own *functions*.

You can quickly cascade a list of functions into an amazing amount of capability.

One reason this is easy is that R automatically does *listwise* operations on a list of numbers. A listwise transformation is a transformation performed the same way on all the numbers in a list.

For example, try these

```
> x
```

```
[1] 1 2 3 4 5
```

```
> 2*x
```

```
[1] 2 4 6 8 10
```

```
> 2*x + 5  
[1]  7  9 11 13 15
```

```
> x^2  
[1]  1  4  9 16 25
```

```
> x - mean(x)  
[1] -2 -1  0  1  2
```

Notice how we were able to calculate deviation scores in one line.

Let's define our own *deviation score function*. This is a function that takes a vector of data and returns the *deviation scores*.

Here is how you define the function:

```
> dev <- function(x)
{ return( x - mean(x) ) }
```

Notice how easily we could have defined the variance ourselves.

```
> myvar <- function(x)
{
N <- length(x)
return( sum ( dev(x)^2 ) / (N-1) )
}
```

Random Number Generation in R

One of the neat things about R is its ability to generate random numbers. This capability allows us to perform “simulation experiments” in which we play God and create a simulated universe according to our own specifications. Then we can “see what happens.”

Sometimes what happens is surprising.

How a Random Number Generator Works

Mathematicians have developed formulas that allow a completely deterministic system to mimic, almost perfectly, the mathematical properties of a random system.

This allows us to study random systems in a way that is replicable, if we know the “seed” or starting value of the random number generation system.

In order to make our work replicable, we will all start by “seeding” the random number generator with the value “12345” as follows.

```
> set.seed(12345)
```

Next, we will create simulated results for two large university classes. The following commands will create two samples of size 200, sampled from a normal distribution (a bell-shaped population we will learn more about later).

```
> x <- rnorm(200, 70, 12)
> y <- rnorm(200, 80, 8)
```

The x group is sampled from a population with a mean of 70 and a standard deviation of 12. The y group is sampled from a population with a mean of 80 and a standard deviation of 8. Note that these samples *will not* have means that are exactly 70 and 80, nor will they have standard deviations of 12 and 8, because they represent *random samples* from a population, and hence show random variation.

Since we all seeded our random number generators with the same value, we should all have the same data.

Getting a Quick Summary

```
> summary(x)
```

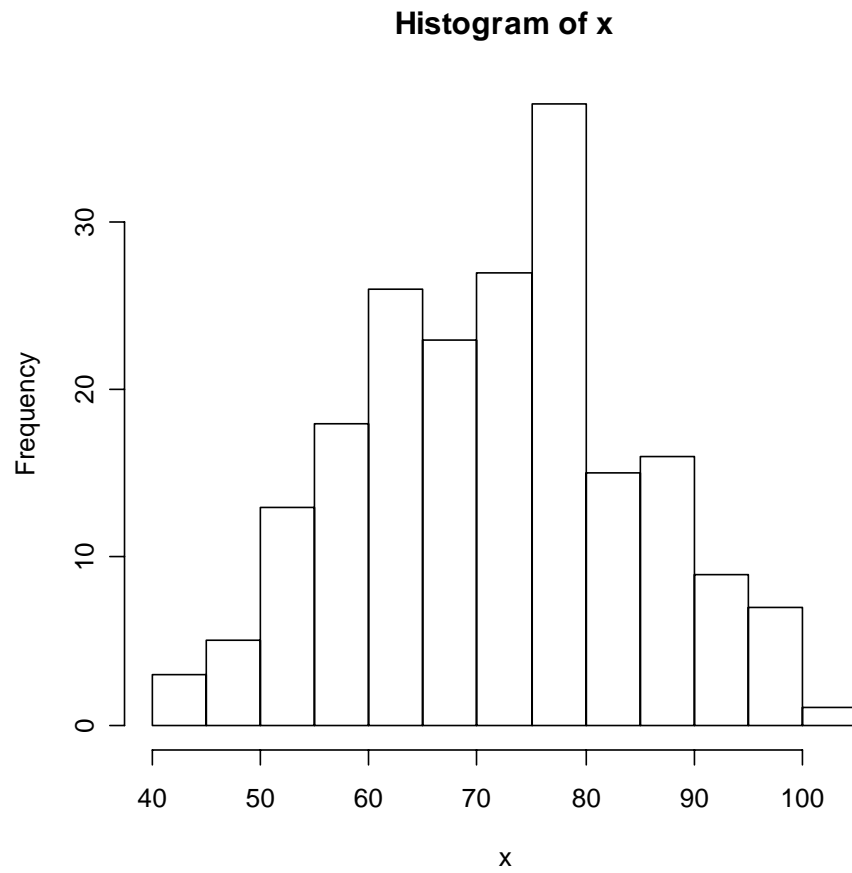
| Min. | 1st Qu. | Median | Mean | 3rd Qu. | Max. |
|-------|---------|--------|-------|---------|--------|
| 41.44 | 63.38 | 72.07 | 71.74 | 79.90 | 101.90 |

```
> summary(y)
```

| Min. | 1st Qu. | Median | Mean | 3rd Qu. | Max. |
|-------|---------|--------|-------|---------|--------|
| 59.35 | 76.11 | 80.90 | 80.68 | 85.18 | 102.00 |

Getting a Histogram

```
> hist(x)
```



Getting Help

```
> ?hist
```

Getting Frequencies

```
> hist(x, plot=FALSE)
```

Getting a Stem-Leaf Diagram

```
> stem(x)
```

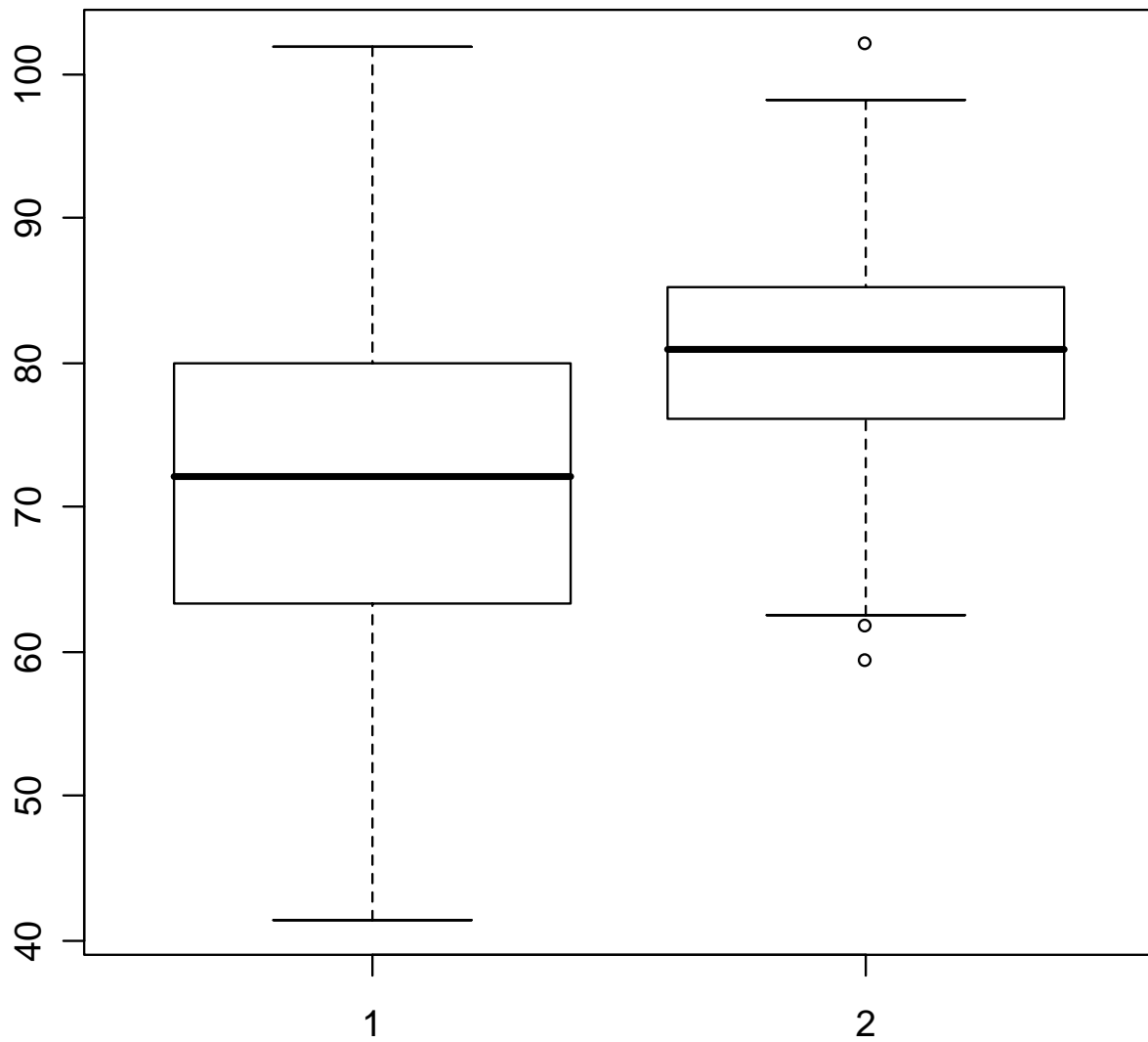
The decimal point is 1 digit(s) to the right of
the |

```
 4 | 12  
 4 | 58888  
 5 | 00111123444444  
 5 | 55666667778888999  
 6 | 000112222333444444444  
 6 | 5555555566666677777889999999  
 7 | 000001111222222333344444444  
 7 | 556666666677777777788888888999  
 8 | 00000000000111112233444  
 8 | 56666777889999  
 9 | 0001122233  
 9 | 566668  
10 | 002
```

Getting a Boxplot

```
>boxplot(x)
```

```
>boxplot(x,y)
```



Percentiles

```
>quantile(x, .75)
```

Creating a Sequence

```
w <- 1:100
```

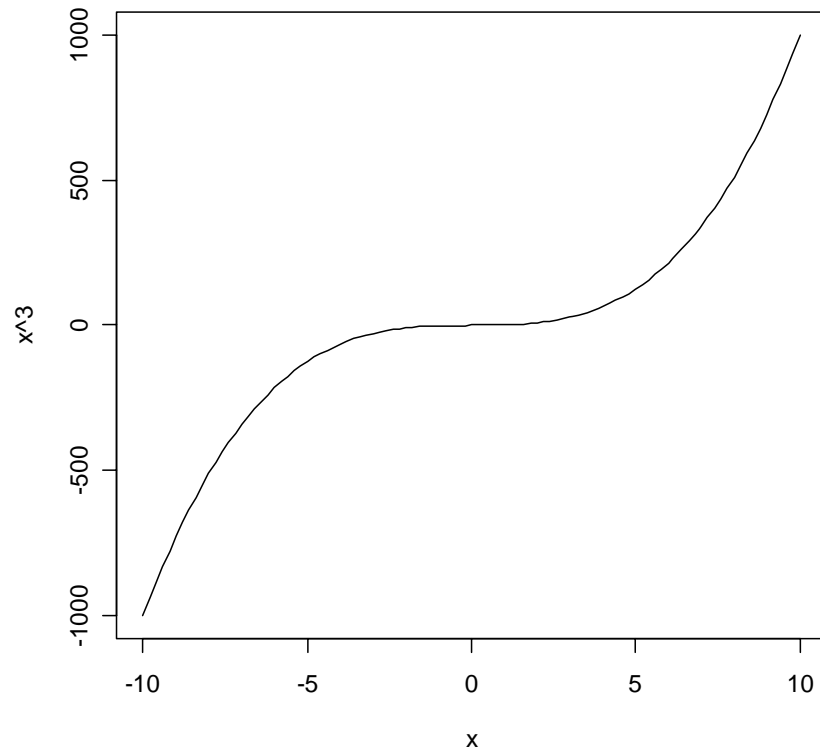
What is the sum of the **w** values?

What is the sum of the squared **w** values?

What is the standard deviation of the **w** values?

Plotting Curves

```
> curve(x^3, -10, 10)
```



Plotting the Normal Curve

```
> curve(dnorm(x, 100, 15), 50, 150)
```

