

Vectors in S+

Some Remarks

Everything in S+ is an object.

The assignment operator `<-` is used for storing objects into variables. When making up names for your objects, be careful not to use names reserved by S+. In particular, never use **c**, **t**, **cat**, **F**, **T**, **D**.

Once you create an object in S+, it is “permanent” in S+ in the sense that you do not explicitly have to save it in order to have it available in the next S+ session. In other words, if you create an object called **fish** and then S+ crashes, **fish** will still exist in the next session. This is different from applications such as Word or Excel which require an explicit **Save** action.

Vectors

The basic data object in S+ is the vector. Even scalars are vectors of length 1.

There are several ways to create vectors.

The `:` operator creates sequences incrementing/decrementing by 1.

```
> 1:10
> 5:-6
```

The `seq()` function creates sequences also.

```
> seq(0,3,by=.2)
> seq(0,3,length=15)
```

To create vectors with no particular pattern, use the `c()` function (`c` for `combine`).

```
> c(1,4,8,2,9)
> x <- c(2,0,-4)
> x
> c(x, 0:5, x)
```

For longer vectors, use `scan()`.

```
> y <- scan() <ENTER>
1: 3 5 8 0 -2 4 <ENTER>
7: 9 11 8 0 <ENTER>
11: 0 3 <ENTER>
13: <ENTER>
> y
```

For vectors of characters,

```
> c("a","b","c","d")
```

or logical values (note that there are no double quotes):

```
> c(T,F,F,T,T,F)
```

The `rep()` command for repeating values:

```
> rep("a",5)
> rep(c("a","b"),5)
> rep(c("a","b"),c(5,2))
```

Basic Arithmetic

```
> x <- 1:5
> x-3
> x*10
> x/10
> x^2
> 2^x
> y <- 6:10
> y
> x*y
```

Note that multiplication is being performed coordinate-wise.

```
> x < 3
> x == 4
```

Subscripting

The basic syntax is `vector[index]`.

```
> z <- c(8,3,0,9,9,2,1)
```

The first element of `z`:

```
> z[1]
```

The first, third and fourth element,

```
> z[c(1,3,4)]
```

The elements of `z` in reverse:

```
> z[7:1]
```

All elements *except* the first and third:

```
> z[-c(1,3)]
```

Those elements of `z` less than 4:

```
> z[z < 4]
```

To understand this, note:

```
> z < 4
```

```
[1] FALSE TRUE TRUE FALSE FALSE TRUE TRUE
```

The TRUE's appear in positions 2, 3, 6, 7.

So `z[z < 4]` is really specifying `z[c(F, T, T, F, F, T, T)]`. In other words, return the values of `z` corresponding to the Trues (positions 2, 3, 6, 7).

```
> w <- c(-2,1,0,1,3,4,8)
```

Those values of `w` when `z` is 9:

```
> w[z==9]
```

The july.precip data set

The data set `july.precip` has precipitation amounts for the 48 contiguous states from 1895-1999.

Attach the data set to access the columns:

```
> attach(july.precip)
```

To find the mean of `Oregon`

```
> mean(Oregon)
```

To find the mean of `Oregon` over all years except 1977,

```
> mean(Oregon[year != 1977])
```

To find the mean of `Oregon` over the years 1970-1999:

```
> mean(Oregon[1970 <= year & year <=1999])
```

The ampersand (&) stands for “and.”

Note: To get a sense of what is going on, let's take apart one of the above commands `mean(Oregon[1970 <= year & year <=1999])`.

```
> year
```

```
> 1970 <= year & year <= 1999
```

```
> Oregon[1970 <= year & year <= 1999]
```

So we only get those values of `Oregon` that correspond to the “TRUEs”.

We'll now find the mean precipitation in Oregon for those years when Oregon's precipitation was greater than the median Idaho precipitation. First, find the rows where Oregon precipitation values are greater than the median of Idaho's precipitation values using the `which` command (this command locates the position of the TRUE's.)

```
> which(Oregon > median(Idaho))
```

We'll store this information into a vector called `index`.

```
> index <- which(Oregon > median(Idaho))
```

Let's see what years corresponding to the values in `index`.

```
> year[index]
```

Now find the average precipitation for those years.

```
> mean(Oregon[index])
```

```
> detach()
```

Vectorized operators

<code>==</code>	equal to
<code>!=</code>	not equal to
<code>></code>	greater than
<code>>=</code>	greater than or equal to
<code><</code>	less than
<code><=</code>	less than or equal to
<code>&</code>	vectorized AND
<code> </code>	vectorized OR
<code>!</code>	not

Programming Note: The vectorized AND and OR are for use with vectors (when you are extracting subsets of vectors). For control in programming (ex. when writing `for` or `if` statements), the operators are `&&` and `||`.