```
> hospitals[1:3,]
  Hosp Cond Surv
1    A Good Died
2    A Good Died
3    A Good Died
> hospitals[nrow(hospitals),]
     Hosp Cond     Surv
2900    B Poor Survived
>


> .Table <- xtabs(~Surv,data=hospitals)
> 100*.Table/sum(.Table)
Surv
     Died  Survived
 2.724138 97.275862
> .Table
Surv
    Died Survived
      79     2821
> 100*.Table/sum(.Table)
Surv
     Died  Survived
 2.724138 97.275862
```

```
> .Table <- xtabs(~Surv+Hosp,data=hospitals)
> .Table
          Hosp
Surv          A    B
  Died       63   16
  Survived 2037  784
> rowPercents(.Table)
          Hosp
Surv          A    B Total Count
  Died     79.7 20.3   100    79
  Survived 72.2 27.8   100  2821
> colPercents(.Table)
          Hosp
Surv          A   B
  Died        3   2
  Survived   97  98
  Total     100 100
  Count    2100 800
> totPercents(.Table)
            A    B Total
Died      2.2  0.6   2.7
Survived 70.2 27.0  97.3
Total    72.4 27.6 100.0
> .Table <- xtabs(~Surv+Cond,data=hospitals)
> .Table
          Cond
Surv       Good Poor
  Died       14   65
  Survived 1186 1635
> rowPercents(.Table)
          Cond
Surv       Good Poor Total Count
  Died     17.7 82.3   100    79
  Survived 42.0 58.0   100  2821
> colPercents(.Table)
          Cond
Surv         Good   Poor
  Died        1.2    3.8
  Survived   98.8   96.2
  Total     100.0  100.0
  Count    1200.0 1700.0
> totPercents(.Table)
         Good Poor Total
Died      0.5  2.2   2.7
Survived 40.9 56.4  97.3
Total    41.4 58.6 100.0
```

```
> .Table <- xtabs(~Surv+Hosp+Cond,data=hospitals)
> .Table
, , Cond = Good

          Hosp
Surv          A    B
  Died        6    8
  Survived  594  592

, , Cond = Poor

          Hosp
Surv          A    B
  Died       57    8
  Survived 1443  192

> rowPercents(.Table)
, , Cond = Good

          Hosp
Surv          A    B Total Count
  Died     42.9 57.1   100    14
  Survived 50.1 49.9   100  1186

, , Cond = Poor

          Hosp
Surv          A    B Total Count
  Died     87.7 12.3   100    65
  Survived 88.3 11.7   100  1635
```

```
> colPercents(.Table)
, , Cond = Good

          Hosp
Surv         A     B
  Died       1   1.3
  Survived  99  98.7
  Total    100 100.0
  Count    600 600.0

, , Cond = Poor

          Hosp
Surv            A   B
  Died        3.8   4
  Survived   96.2  96
  Total     100.0 100
  Count    1500.0 200

> totPercents(.Table)
        tab Total
<NA>    0.2   2.7
<NA>   20.5  97.3
<NA>    0.3   2.7
<NA>   20.4  97.3
<NA>    2.0   2.7
<NA>   49.8  97.3
<NA>    0.3   2.7
<NA>    6.6  97.3
Total 100.0 400.0
>
```

```
### Using the Weight data from the NCSS Sample dataset.
> sum(htwt$Weight)
[1] NA
### Why the NA?
> htwt$Weight
 [1] 159 155 157 125 103 122 101  82 228 199 195 110 191 151 119 119 112  87 190
[20]  87  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA
[39]  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA
[58]  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA
### The sum function allows us to remove the NA's
> args(sum)
function (..., na.rm = FALSE)
NULL
> sum(htwt$Weight,na.rm=TRUE)
[1] 2792
### However, the number of observations/rows reported includes the NA's
> length(htwt$Weight)
[1] 75
### Create a new variable with only those observations with non-NA's
> weight <- htwt$Weight[!is.na(htwt$Weight)]
### Now the number of observations/rows is correct
> length(weight)
[1] 20
### Compute the mean by summing and dividing by the number of obs
> sum(weight)
[1] 2792
> sum(weight)/length(weight)
[1] 139.6
### OR, use the mean function
> mean(weight)
[1] 139.6
### Now compute the variance by summing the squared deviations from the
###   mean and dividing by n-1. Computing the mean once and assigning
###   it to xbar and then calling xbar is more efficient than using
###   mean(weight) in the sum.
> xbar <- mean(weight)
> sum((weight-xbar)^2)
[1] 35330.8
> sum((weight-xbar)^2)/(length(weight)-1)
[1] 1859.516
### OR, use the var fucntion
> var(weight)
[1] 1859.516
### The standard deviation is the square root of the variance.
> sqrt(var(weight))
[1] 43.1221
```

```
### Compute the quartiles by sorting and counting observations.
> sort(weight)
 [1]  82  87  87 101 103 110 112 119 119 122 125 151 155 157 159 190 191 195 199
[20] 228
### OR, use the quantile function
> quantile(weight)
    0%    25%    50%    75%   100%
 82.00 108.25 123.50 166.75 228.00
> median(weight)
[1] 123.5

### Rcmdr has the function numSummary which can be called from the
###   menu.  It computes all of the above with a single call.
> numSummary(htwt[,"Weight"], statistics=c("mean", "sd", "quantiles"))
  mean      sd 0%    25%   50%    75% 100%  n NA
 139.6 43.1221 82 108.25 123.5 166.75  228 20 55

### While it is possible to use mean, var, etc. and get results by
###  group, using numSummary with groups= is easier.
> numSummary(htwt[,"Weight"], groups=htwt$Group, statistics=c("mean",
"sd", "quantiles"))
       mean       sd 0%   25% 50% 75% 100%  n
Male    155 48.99235 87 119.0 159 191  228  9
Female  127 34.99714 82 106.5 119 153  199 11
```

```
### Enter only the last twelve observations into temp
> temp <- c(228,199,195,110,191,151,119,119,112,87,190,87)
> temp
 [1] 228 199 195 110 191 151 119 119 112  87 190  87
### Or, select only the last 12 observations from temp
> temp <- weight[9:20]
> temp
 [1] 228 199 195 110 191 151 119 119 112  87 190  87
### Compute the mean of all twelve observations
> mean(temp)
[1] 149
### Ignore the 228 and recompute
> temp[-1]
 [1] 199 195 110 191 151 119 119 112  87 190  87
> mean(temp[-1])
[1] 141.8182
### Compute the variance of the twelve observations
> (sum(temp^2)-length(temp)*(mean(temp))^2)/(length(temp)-1)
[1] 2425.818
### Or, use the var function
> var(temp)
[1] 2425.818
> sqrt(var(temp))
[1] 49.2526
### Compute the standard deviation ignoring the 228
> sqrt(var(temp[-1]))
[1] 44.5821
### Note the effect of ignoring the 228 on the quantiles, range,
###   and IQR
> quantile(temp)
   0%   25%   50%   75%  100%
 87.0 111.5 135.0 192.0 228.0
> quantile(temp[-1])
   0%   25%   50%   75%  100%
 87.0 111.0 119.0 190.5 199.0
> range(temp)
[1]  87 228
> range(temp)[2]-range(temp)[1]
[1] 141
> range(temp[-1])[2]-range(temp[-1])[1]
[1] 112
> quantile(temp)[4]-quantile(temp)[2]
 80.5
> quantile(temp[-1])[4]-quantile(temp[-1])[2]
 79.5
>
```

```
### Vector and matrix manipulations in R (S-Plus)
### Assign the integers 1 through 10 to a vector temp
> temp <- 1:10
> temp
 [1]  1  2  3  4  5  6  7  8  9 10
### Scalar multiplication
> temp <- temp*10
> temp
 [1]  10  20  30  40  50  60  70  80  90 100
### Select the odd elements
> temp[c(1,3,5,7,9)]
[1] 10 30 50 70 90
### Or, select the odd elements
> temp[(0:4)*2+1]
[1] 10 30 50 70 90
### Select the even elements
> temp[(1:5)*2]
[1]  20  40  60  80 100
### Select the even elements by excluding (-) the odd
> temp[-((0:4)*2+1)]
[1]  20  40  60  80 100
### Select the third through first elements
> temp[3:1]
[1] 30 20 10
### Sum by vector multiplication (%*%) with a vector of ones
> temp%*%rep(1,length(temp))
     [,1]
[1,]  550
### Or, use the faster sum function
> sum(temp)
[1] 550
### Compute the sum of squares quickly
> temp%*%temp
      [,1]
[1,] 38500
### Compute the sum of squares slowly
> sum(temp^2)
[1] 38500
### Define a 3x3 matrix using the first nine elements of temp
> matrix(temp[1:9],nrow=3)
     [,1] [,2] [,3]
[1,]   10   40   70
[2,]   20   50   80
[3,]   30   60   90
```

```
### Attempt to invert the matrix and determine that it is singular
> solve(matrix(temp[1:9],nrow=3))
Error in solve.default(matrix(temp[1:9], nrow = 3)) :
        system is computationally singular: reciprocal condition number
= 9.25186e-19
### Matrix mult (%*%)
> matrix(temp[1:9],nrow=3)%*%matrix(temp[1:9],nrow=3)
     [,1] [,2]  [,3]
[1,] 3000 6600 10200
[2,] 3600 8100 12600
[3,] 4200 9600 15000
### Element mult (*)
> matrix(temp[1:9],nrow=3)*matrix(temp[1:9],nrow=3)
     [,1] [,2] [,3]
[1,]  100 1600 4900
[2,]  400 2500 6400
[3,]  900 3600 8100
### Matrix transpose (t)
> t(matrix(temp[1:9],nrow=3))
     [,1] [,2] [,3]
[1,]   10   20   30
[2,]   40   50   60
[3,]   70   80   90
>
### Other matrix/vector functions exist. E.g. qr, chol, chol2inv,
###   eigen, det, svd, %o% (outer product), etc.
```