# THE UNIVERSITY OF AUCKLAND

---

**SEMESTER 2, 2020**
**Campus: City**

---

**STATISTICS**

**Statistical Computing**

**(Time allowed: TWO Hours)**

**INSTRUCTIONS**

- Attempt ALL questions.
- Total marks are 100.
- Calculators are permitted.
- R Quick Reference is available in Attachment.

# Part I: Programming

**For questions in Part I, avoid using explicit loops or anything equivalent as much as possible, unless the question asks to use them.**

1. Use the colon operator `(:)`, `seq()`, `rep()` and some other commonly used arithmetic operators/functions to create the sequences given below. *Note that you must not use `c()` or any explicit loop to create the sequences.*

   (a) `1 -4 3 -8 5 -12 7 -16`

   [3 marks]

   (b) `1 1 1 2 2 3 3 3 4 4`

   [3 marks]

   (c) `1.4 2.1 2.8 3.5 4.2 4.9`

   [3 marks]

   (d) `0 0 1 1 1 2 2 2 3 3 3 4`

   [3 marks]

   (e) `0 0 0 0 0 5 0 0 0 0 4 0 0 0 3 0 0 2 0 1`

   [3 marks]

   **[15 marks]**

2. Let $X = (x_1, x_2, \ldots, x_n)$ be a non-zero real vector. For each part write suitable `R` code. Note that your code should handle `NA` values in $X$.

   (a) Find the sum of elements of $X$.

   [2 marks]

   (b) Find the mean of positive elements of $X$.

   [3 marks]

   (c) Find the first element of $X$ which is less than the preceding value (e.g. if $X = c(4, 2, 3)$ the answer is 2), or `NA` if no such element exists.

   [4 marks]

   (d) Write a function called `index.n` which determine the index of the element of $X$ which is the $n$-th to satisfy a given condition. *Note that n should be an argument and your function should handle exceptional cases such as follows.*

   [5 marks]

```
> X= 10:20
> index.n (X > 14 , 2)

[1] 7

> index.n (X > 14 , 7)

[1] NA
```

**[14 marks]**

3. Assume that `X` is a matrix of 1s and 0s. Write an `R` function called `maj.mat` that creates a vector as follows: For each row of the matrix `X`, the corresponding element of the vector will be either 1 or 0, depending on whether the majority of the first `d` elements in that row is 1 or 0. Here, `d` will be a parameter that we may wish to vary.

```
> X

      [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    1    1    0
[2,]    1    1    0    0    0
[3,]    1    0    0    0    1
[4,]    0    1    1    1    0

> apply(x, 1 , maj.mat , d = 3)

  [1] 1 1 0 1

> apply(x, 1 , maj.mat , d = 2)

 [1] 0 1 0 0
```

**[6 marks]**

4. Complete the following R code (replace the dashes with your code) to create the panels shown in Figure 1.

```
>  mat = matrix(c(---------------),4 ,3 , byrow = -------)
>  layout(mat, widths = -------, heights = c(1,2,2,4))
>   ----(-------)
>  box("-----")
```

**[5 marks]**

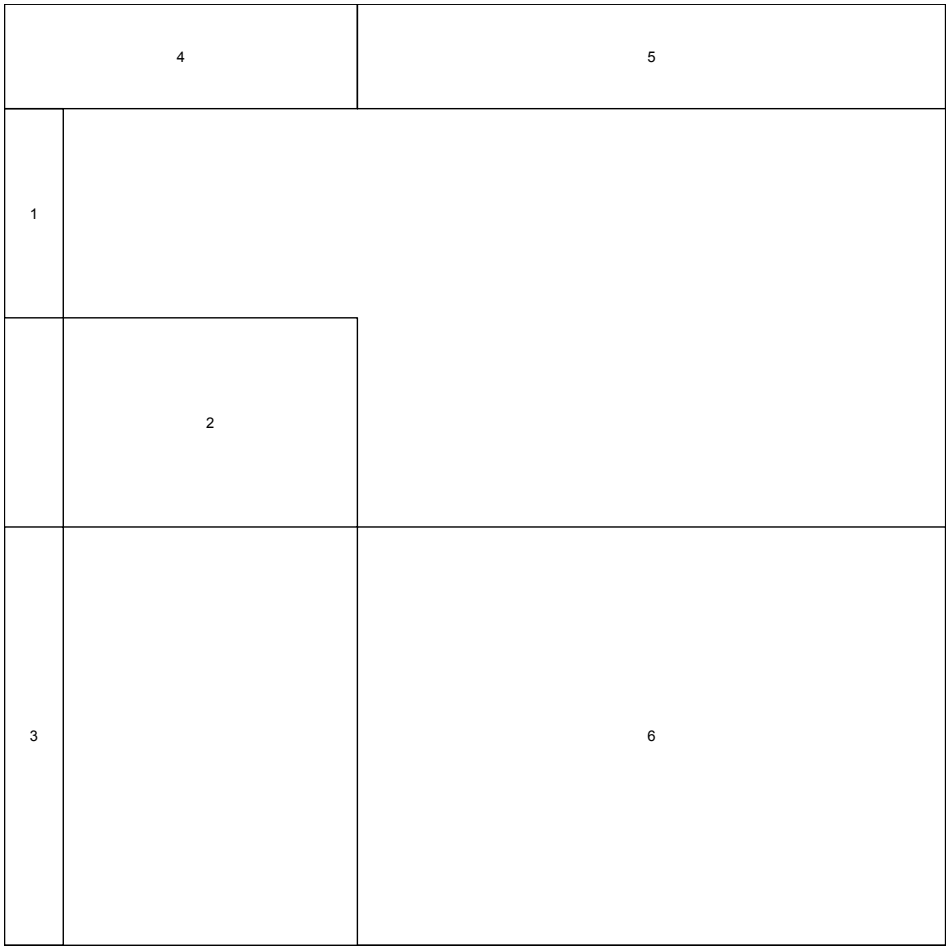


Figure 1: A layout to display 6 plots.

5. Figure 2 shows the number of individual visits to a medical center during the last year. The data is collected from a group of 50 men and 50 women living in an Aged Care. This figure is produce with the following code:

```
> plot(x, y , pch=2)     # Male
> points(x, z , pch=16)  # Female
```

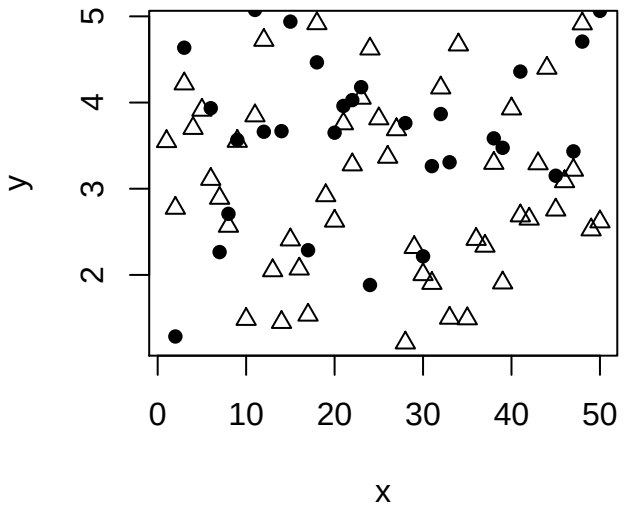


Figure 2: Simple Plot.

Complete the following R code to improve it to Figure 3 (the improvments are a) the colour of character, b) the axes limits and labels , c) legend of the graph, and d) the blue dashed line in the middle of the graph).

```
>   plot(x, y , pch = 2, xlim = c(-2, 52), ylim = ------,
                          ------, -----)
>  points(x, z , pch = 16, col = 'red')
>  legend( -----  , c('Male', 'Female'), pch = -----, col = -----,
                              cex = 0.75)
>  ---(-----)
>    lines( x= -----, y = c(3, 3), ------ , col = "blue")
```

**[10 marks]**

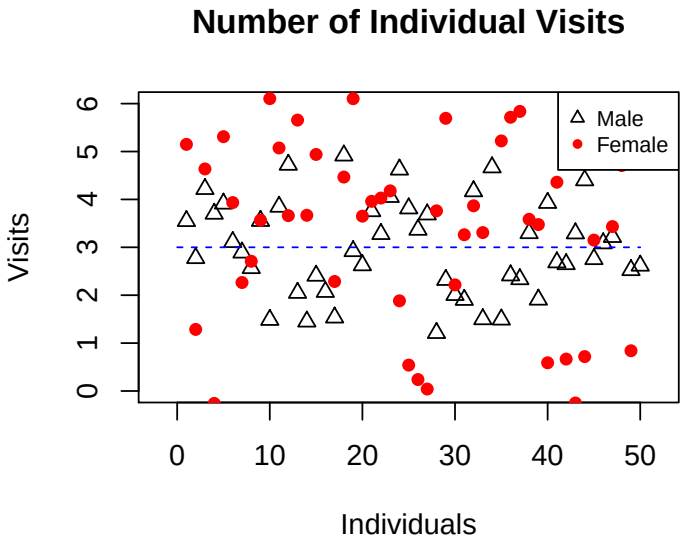


Figure 3: Revised Simple Plot.

# Part II: Data Technology

6. Suppose `text` is a character vector in the current R session. **Explain in thorough detail** the purpose of the following R expressions. Your answer should include an explanation of the purpose of each expression, the meaning of each argument in the function calls, and what sort of data structure is produced by each expression and function call.

(a) `lapply(text, nchar)`

[2 marks]

(b) `paste(1:length(text), text, sep = "\\)")`

[2 marks]

(c) `regmatches(text, gregexpr("WORD", text)) <- "word"`

[2 marks]

(d) `lapply(strsplit(text, " |,|[.]"), length)`

[2 marks]

(e) `sub("[(](.+?)[)]\\{(.+?)\\}", "(\\2){\\1}", text)`

[2 marks]

**[10 marks]**

7. Suppose `allData` is a data frame in R. Each column of `allData` contains the information of the sales for different retail stores of an imaginary company for a specific month in a specific year.

```
> head(colnames(allData))

[1] "jan2009" "feb2009" "jun2009" "aug2009" "dec2009" "jan2010"
```

Complete the following R expression in such a way that it prints out the last few rows of the sales information during January.

```
> tail(allData[, __________________ ])
```

**[5 marks]**

8. The symbol `population` is a data frame in R that contains the population information of different cities in a country. The first few rows and columns of this data frame are shown for your reference.

```
> population[1:3, 1:4]

    name  pop2010 pop2011 pop2012
1 city 1      100     100     105
2 city 2      110     111     110
3 city 3      120     125     110

> ncol(population)

[1] 11
```

Complete the following R expression in such a way that it calculates the average population of each city over time.

```
> apply(population, 1, function(x){ ____________________ })
```

**[5 marks]**

9. For each day of the year, the hourly strength of the wind in a place is recorded in a text file called "`wind.txt`". The strength of wind is recorded as a number between 0–9. Each line of "`wind.txt`" corresponds to a specific day of the year, and it contains 24 numbers for each hour of that day. We use the `readLines()` function to read this file into R as a character vector and assign it to the symbol `wind`.

```
> head(wind)

[1] "435353365363344466336554" "57436336575655376635343 4"
[3] "673655763567754773765635" "35636675777434476764656 5"
[5] "637377343346735543676445" "006523701725174167555206"
```

Write R code to process `wind` and create a matrix of the values of the strength throughout the year. Your code should assign the result to the symbol `strength`.

```
> strength[1:3, 1:4]

      [,1] [,2] [,3] [,4]
[1,]    4    3    5    3
[2,]    5    7    4    3
[3,]    6    7    3    6

> ncol(strength)

[1] 24
```

**[10 marks]**

10. The information about a medical measurement for some individuals are recorded in a data frame in R called `df`.

```
> head(df)

     id age gender measure
1 pe001  39      M    0.19
2 pe002  40      M    0.95
3 pe003  41      M    0.36
4 pe004  45      F    0.11
5 pe005  50      M    0.91
6 pe006  47      M    0.04
```

Complete the following code in such a way that it selects a random sample of size 5 from each level of `gender`.

```
> selectedID = tapply(df$id, ________ , function(x){ __________ })
> result = lapply(selectedID, function(x){ _____________ })
```

```
> result

$F
       id age gender measure
26 pe0026  45      F    0.34
33 pe0033  30      F    0.91
35 pe0035  29      F    0.03
52 pe0052  26      F    0.85
80 pe0080  42      F    0.98

$M
       id age gender measure
20 pe0020  41      M    0.07
22 pe0022  50      M    0.83
56 pe0056  48      M    0.32
63 pe0063  22      M    0.03
83 pe0083  32      M    0.11
```

**[10 marks]**

11. The summary statistics for variables $x_1$ to $x_{100}$ are recorded in a data frame in R called `sum.out`. The first few rows of the first few columns of this data frame are shown for your reference.

```
>  sum.out[1:4, 1:4]

  name           x1           x2           x3
1 mean   0.49090990 1.248966e-02   0.49860616
2    n 100.00000000 1.010000e+03 200.00000000
3  var   0.08825365 9.970411e-01   0.08952302
4  std   0.29707516 9.985195e-01   0.29920397
```

Use the functions in the `reshape2` library to transpose this data frame and assign it to the symbol `t.sum.out`. The first few observations of the desired output are shown for your reference.

```
>  t.sum.out[1:4, 1:4]


  variable       max        mean          min
1       x1 0.9932668  0.49090990  0.005427403
2       x2 4.1965327  0.01248966 -3.218583296
3       x3 0.9966696  0.49860616  0.002607813
4       x4 0.4664198 -0.45272577 -1.470112336
```

**[10 marks]**

**ATTACHMENT FOLLOWS**

# R QUICK REFERENCE

## Basic Data Representation

| | |
|---|---|
| `TRUE`, `FALSE` | logical true and false |
| `1`, `2.5`, `117.333` | simple numbers |
| `1.23e20` | scientific notation, $1.23 \times 10^{20}$. |
| `3+4i` | complex numbers |
| `"hello, world"` | a character string |
| `NA` | missing value (in any type of vector) |
| `NULL` | missing value indicator in lists |
| `NaN` | not a number |
| `Inf` | positive infinity |
| `-Inf` | negative infinity |
| "*var*" | quotation for special variable name (e.g. `+`, `%*%`, etc.) |

## Creating Vectors

| | |
|---|---|
| `c`$(a_1, \ldots, a_n)$ | combine into a vector |
| `logical(n)` | logical vector of length `n` (containing falses) |
| `numeric(n)` | numeric vector of length `n` (containing zeros) |
| `complex(n)` | complex vector of length `n` (containing zeros) |
| `character(n)` | character vector of length `n` (containing empty strings) |

## Creating Lists

| | |
|---|---|
| `list`$(e_1, \ldots, e_k)$ | combine as a list |
| `vector(k, "list")` | create a list of length `k` (the elements are all `NULL`) |

## Basic Vector and List Properties

| | |
|---|---|
| `length(x)` | the number of elements in `x` |
| `mode(x)` | the mode or type of `x` |

## Tests for Types

| | |
|---|---|
| `is.logical(x)` | true for logical vectors |
| `is.numeric(x)` | true for numeric vectors |
| `is.complex(x)` | true for complex vectors |
| `is.character(x)` | true for character vectors |
| `is.list(x)` | true for lists |
| `is.vector(x)` | true for both lists and vectors |

## Tests for Special Values

| | |
|---|---|
| `is.na(x)` | true for elements which are `NA` or `NaN` |
| `is.nan(x)` | true for elements which are `NaN` |
| `is.null(x)` | tests whether `x` is `NULL` |
| `is.finite(x)` | true for finite elements (i.e. not `NA`, `NaN`, `Inf` or `-Inf`) |
| `is.infinite(x)` | true for elements equal to `Inf` or `-Inf` |

## Explicit Type Coercion

| | |
|---|---|
| `as.logical(x)` | coerces to a logical vector |
| `as.numeric(x)` | coerces to a numeric vector |
| `as.complex(x)` | coerces to a complex vector |
| `as.character(x)` | coerces to a character vector |
| `as.list(x)` | coerces to a list |
| `as.vector(x)` | coerces to a vector (lists remain lists) |
| `unlist(x)` | converts a list to a vector |

## Vector and List Names

| | |
|---|---|
| `c(`$n_1$`=`$e_1$`,...,`$n_k$`=`$e_k$`)` | combine as a named vector |
| `list(`$n_1$`=`$e_1$`,...,`$n_k$`=`$e_k$`)` | combine as a named list |
| `names(x)` | extract the names of `x` |
| `names(x) = v` | (re)set the names of `x` to `v` |
| `names(x) = NULL` | remove the names from `x` |

## Vector Subsetting

| | |
|---|---|
| `x[1:5]` | select elements by index |
| `x[-(1:5)]` | exclude elements by index |
| `x[c(TRUE, FALSE)]` | select elements corresponding to `TRUE` |
| `x[c("a", "b")]` | select elements by name |

## List Subsetting

| | |
|---|---|
| `x[1:5]` | extract a *sublist* of the list `x` |
| `x[-(1:5)]` | extract a *sublist* by excluding elements |
| `x[c(TRUE, FALSE)]` | extract a *sublist* with logical subscripts |
| `x[c("a", "b")]` | extract a *sublist* by name |

## Extracting Elements from Lists

| | |
|---|---|
| `x[[2]]` | extract an *element* of the list `x` |
| `x[["a"]]` | extract the *element* with name `"a"` from `x` |
| `x$a` | extract the *element* with name name `"a"` from `x` |

## Logical Selection

| | |
|---|---|
| `ifelse(cond, yes, no)` | conditionally select elements from `yes` and `no` |
| `which(v)` | returns the indices of `TRUE` values in `v` |

## List Manipulation

| | |
|---|---|
| `lapply(X, FUN, ...)` | apply `FUN` to the elements of `X` |
| `split(x, f)` | split `x` using the factor `f` |

## Sequences and Repetition

| | |
|---|---|
| `a:b` | sequence from `a` to `b` in steps of size 1 |
| `seq(n)` | same as `1:n` |
| `seq(a,b)` | same as `a:b` |
| `seq(a,b,by=s)` | `a` to `b` in steps of size `s` |
| `seq(a,b,length=n)` | sequence of length `n` from `a` to `b` |
| `seq(along=x)` | like `1:length(n)`, but works when `x` has zero length |
| `rep(x,n)` | `x`, repeated `n` times |
| `rep(x,v)` | elements of `x` with `x[i]` repeated `v[i]` times |
| `rep(x,each=n)` | elements of `x`, each repreated `n` times |

## Sorting and Ordering

| | |
|---|---|
| `sort(x)` | sort into ascending order |
| `sort(x, decreasing=TRUE)` | sort into descending order |
| `rev(x)` | reverse the elements in x |
| `order(x)` | get the ordering permutation for `x` |

## Basic Arithmetic Operations

| | |
|---|---|
| `x + y` | addition, "x plus y" |
| `x - y` | subtraction, "x minus y" |
| `x * y` | multiplication, "x times y" |
| `x / y` | division, "x divided by y" |
| `x ^ y` | exponentiation, "x raised to power y" |
| `x %% y` | remainder, "x modulo y" |
| `x %/% y` | integer division, "x divided by y, discard fractional part" |

## Rounding

| | |
|---|---|
| `round(x)` | round to nearest integer |
| `round(x,d)` | round `x` to `d` decimal places |
| `signif(x,d)` | round `x` to `d` significant digits |
| `floor(x)` | round down to next lowest integer |
| `ceiling(x)` | round up to next highest integer |

## Common Mathematical Functions

| | |
|---|---|
| `abs(x)` | absolute values |
| `sqrt(x)` | square root |
| `exp(x)` | exponential functiopn |
| `log(x)` | natural logarithms (base $e$) |
| `log10(x)` | common logarithms (base 10) |
| `log2(x)` | base 2 logarithms |
| `log(x,base=b)` | base `b` logarithms |

## Trigonometric and Hyperbolic Functions

| | |
|---|---|
| `sin(x)`, `cos(x)`, `tan(x)` | trigonometric functions |
| `asin(x)`, `acos(x)`, `atan(x)` | inverse trigonometric functions |
| `atan2(x,y)` | arc tangent with two arguments |
| `sinh(x)`, `cosh(x)`, `tanh(x)` | hyperbolic functions |
| `asinh(x)`, `acosh(x)`, `atanh(x)` | inverse hyperbolic functions |

## Combinatorics

| | |
|---|---|
| `choose(n, k)` | binomial coefficients |
| `lchoose(n, k)` | log binomial coefficients |
| `factorial(x)` | factorials |
| `lfactorial(x)` | log factorials |

## Special Mathematical Functions

| | |
|---|---|
| `beta(x,y)` | the beta function |
| `lbeta(x,y)` | the log beta function |
| `gamma(x)` | the gamma function |
| `lgamma(x)` | the log gamma function |
| `psigamma(x,deriv=0)` | the psigamma function |
| `digamma(x)` | the digamma function |
| `trigamma(x)` | the trigamma function |

## Bessel Functions

| | |
|---|---|
| `besselI(x,nu)` | Bessel Functions of the first kind |
| `besselK(x,nu)` | Bessel Functions of the second kind |
| `besselJ(x,nu)` | modified Bessel Functions of the first kind |
| `besselY(x,nu)` | modified Bessel Functions of the third kind |

## Special Floating-Point Values

| | |
|---|---|
| `.Machine$double.xmax` | largest floating point value ($1.797693 \times 10^{308}$) |
| `.Machine$double.xmin` | smallest floating point value ($2.225074 \times 10^{-308}$) |
| `.Machine$double.eps` | machine epsilon ($2.220446 \times 10^{-16}$) |

## Basic Summaries

| | |
|---|---|
| `sum(`$x_1$`,`$x_2$`,...)` | sum of values in arguments |
| `prod(`$x_1$`,`$x_2$`,...)` | product of values in arguments |
| `min(`$x_1$`,`$x_2$`,...)` | minimum of values in arguments |
| `max(`$x_1$`,`$x_2$`,...)` | maximum of values in arguments |
| `range(`$x_1$`,`$x_2$`,...)` | range (minimum and maximum) |

## Cumulative Summaries

| | |
|---|---|
| `cumsum(x)` | cumulative sum |
| `cumprod(x)` | cumulative product |
| `cummin(x)` | cumulative minimum |
| `cummax(x)` | cumulative maximum |

## Parallel Summaries

| | |
|---|---|
| `pmin(`$x_1$`,`$x_2$`,...)` | parallel minimum |
| `pmax(`$x_1$`,`$x_2$`,...)` | parallel maximum |

## Statistical Summaries

| | |
|---|---|
| `mean(x)` | mean of elements |
| `sd(x)` | standard deviation of elements |
| `var(x)` | variance of elements |
| `median(x)` | median of elements |
| `quantile(x)` | median, quartiles and extremes |
| `quantile(x, p)` | specified quantiles |

## Uniform Distribution

| | |
|---|---|
| `runif(n)` | vector of `n` Uniform[0,1] random numbers |
| `runif(n,a,b)` | vector of `n` Uniform[`a`,`b`] random numbers |
| `punif(x,a,b)` | distribution function of Uniform[`a`,`b`] |
| `qunif(x,a,b)` | inverse distribution function of Uniform[`a`,`b`] |
| `dunif(x,a,b)` | density function of Uniform[`a`,`b`] |

## Binomial Distribution

| | |
|---|---|
| `rbinom(n,size,prob)` | a vector of `n` Bin(`size`,`prob`) random numbers |
| `pbinom(x,size,prob)` | Bin(`size`,`prob`) distribution function |
| `qbinom(x,size,prob)` | Bin(`size`,`prob`) inverse distribution function |
| `dbinom(x,size,prob)` | Bin(`size`,`prob`) density function |

## Normal Distribution

| | |
|---|---|
| `rnorm(n)` | a vector of `n` $N(0,1)$ random numbers |
| `pnorm(x)` | $N(0,1)$ distribution function |
| `qnorm(x)` | $N(0,1)$ inverse distribution function |
| `dnorm(x)` | $N(0,1)$ density function |
| `rnorm(n,mean,sd)` | a vector of `n` normal random numbers with given mean and s.d. |
| `pnorm(x,mean,sd)` | normal distribution function with given mean and s.d. |
| `qnorm(x,mean,sd)` | normal inverse distribution function with given mean and s.d. |
| `dnorm(x,mean,sd)` | normal density function with given mean and s.d. |

## Chi-Squared Distribution

| | |
|---|---|
| `rchisq(n,df)` | a vector of `n` $\chi^2$ random numbers with degrees of freedom `df` |
| `pchisq(x,df)` | $\chi^2$ distribution function with degrees of freedom `df` |
| `qchisq(x,df)` | $\chi^2$ inverse distribution function with degrees of freedom `df` |
| `dchisq(x,df)` | $\chi^2$ density function with degrees of freedom `df` |

## *t* Distribution

| | |
|---|---|
| `rt(n,df)` | a vector of `n` $t$ random numbers with degrees of freedom `df` |
| `pt(x,df)` | $t$ distribution function with degrees of freedom `df` |
| `qt(x,df)` | $t$ inverse distribution function with degrees of freedom `df` |
| `dt(x,df)` | $t$ density function with degrees of freedom `df` |

## *F* Distribution

| | |
|---|---|
| `rf(n,df1,df2)` | a vector of `n` $F$ random numbers with degrees of freedom `df1` & `df2` |
| `pf(x,df1,df2)` | $F$ distribution function with degrees of freedom `df1` & `df2` |
| `qf(x,df1,df2)` | $F$ inverse distribution function with degrees of freedom `df1` & `df2` |
| `df(x,df1,df2)` | $F$ density function with degrees of freedom `df1` & `df2` |

## Matrices

| | |
|---|---|
| `matrix(x, nr=`$r$`, nc=`$c$`)` | create a matrix from `x` (column major order) |
| `matrix(x, nr=`$r$`, nc=`$c$`, byrow=TRUE)` | create a matrix from `x` (row major order) |

## Matrix Dimensions

| | |
|---|---|
| `nrow(x)` | number of rows in `x` |
| `ncol(x)` | number of columns in `x` |
| `dim(x)` | vector coltaining `nrow(x)` and `ncol(x)` |

## Row and Column Indices

| | |
|---|---|
| `row(x)` | matrix of row indices for matrix `x` |
| `col(x)` | matrix of column indices for matrix `x` |

## Naming Rows and Columns

| | |
|---|---|
| `rownames(x)` | get the row names of `x` |
| `rownames(x) = v` | set the row names of `x` to `v` |
| `colnames(x)` | get the column names of `x` |
| `colnames(x) = v` | set the column names of `x` to `v` |
| `dimnames(x)` | get both row and column names (in a list) |
| `dimnames(x) = list(`$rn$`,`$cn$`)` | set both row and column names |

## Binding Rows and Columns

| | |
|---|---|
| `rbind(`$v_1$`,`$v_2$`,...)` | assemble a matrix from rows |
| `cbind(`$v_1$`,`$v_2$`,...)` | assemble a matrix from columns |
| `rbind(`$n_1$`=`$v_1$`,`$n_2$`=`$v_2$`,...)` | assemble by rows, specifying row names |
| `cbind(`$n_2$`=`$v_1$`,`$n_2$`=`$v_2$`,...)` | assemble by columns, specifying column names |

## Matrix Subsets

| | |
|---|---|
| `x[i,j]` | submatrix, rows and columns specified by `i` and `j` |
| `x[i,j] = v` | reset a submatrix, rows and columns specified by `i` and `j` |
| `x[i,]` | submatrix, contains just the rows a specified by `i` |
| `x[i,] = v` | reset specified rows of a matrix |
| `x[,j]` | submatrix, contains just the columns specified by `j` |
| `x[,j] = v` | reset specified columns of a matrix |
| `x[i]` | subset as a vector |
| `x[i] = v` | reset elements (treated as a vector operation) |

## Matrix Diagonals

| | |
|---|---|
| `diag(A)` | extract the diagonal of the matrix `A` |
| `diag(v)` | diagonal matrix with elements in the vector `v` |
| `diag(n)` | the `n`×`n` identity matrix |

## Applying Summaries over Rows and Columns

| | |
|---|---|
| `apply(X,1,fun)` | apply `fun` to the rows of `X` |
| `apply(X,2,fun)` | apply `fun` to the columns of `X` |

## Basic Matrix Manipulation

| | |
|---|---|
| `t(A)` | matrix transpose |
| `A %*% B` | matrix product |
| `outer(u, v)` | outer product of vectors |
| `outer(u, v, f)` | generalised outer product |

## Linear Equations

| | |
|---|---|
| `solve(A, b)` | solve a system of linear equations |
| `solve(A, B)` | same, with multiple right-hand sides |
| `solve(A)` | invert the square matrix `A` |

## Matrix Decompositions

| | |
|---|---|
| `chol(A)` | the Choleski decomposition |
| `qr(A)` | the QR decomposition |
| `svd(A)` | the singular-value decomposition |
| `eigen(A)` | eigenvalues and eigenvectors |

## Least-Squares Fitting

| | |
|---|---|
| `lsfit(X,y)` | least-squares fit with carriers `X` and response `y` |

## Factors and Ordered Factors

| | |
|---|---|
| `factor(x)` | create a factor from the values in `x` |
| `factor(x,levels=l)` | create a factor with the given level set |
| `ordered(x)` | create an ordered factor with the given level set |
| `is.factor(x)` | true for factors and ordered factors |
| `is.ordered(x)` | true for ordered factors |
| `levels(x)` | the levels of a factor or ordered factor |
| `levels(x) = v` | reset the levels of a factor or ordered factor |

## Tabulation and Cross-Tabulation

| | |
|---|---|
| `table(x)` | tabulate the values in `x` |
| `table(`$f_1$`,`$f_2$`,...)` | cross tabulation of factors |

## Summary over Factor Levels

| | |
|---|---|
| `tapply(x,f,fun)` | apply summary `fun` to `x` broken down by `f` |
| `tapply(x,list(`$f_1$`,`$f_2$`,...),fun)` | apply summary `fun` to `x` broken down by several factors |

## Data Frames

| | |
|---|---|
| `data.frame(`$n_1$`=`$x_1$`,`$n_2$`=`$x_2$`,...)` | create a data frame |
| `row.names(df)` | extract the observation names from a data frame |
| `row.names(df) = v` | (re)set the observation names of a data frame |
| `names(df)` | extract the variable names from a data frame |
| `names(df)` = v | (re)set the variable names of a data frame |

## Subsetting and Transforming Data Frames

| | |
|---|---|
| `df[i,j]` | matrix subsetting of a data frame |
| `df[i,j] = dfv` | reset a subset of a data frame |
| `subset(df,subset=i)` | subset of the cases of a data frame |
| `subset(df,select=i)` | subset of the variables of a data frame |
| `subset(df,subset=i,select=j)` | subset of the cases and variables of a data frame |
| `transform(df,`$n_1$`=`$e_1$`,`$n_2$`=`$e_2$`,...)` | transform variables in a data frame |
| `merge(df1,df2,...)` | merge data frames based on common variables |

## Reading Lines

| | |
|---|---|
| `readline(prompt="")` | read a line of input |
| `readLines(file, n)` | read `n` lines from the specified file |
| `readLines(file)` | read all lines from the specified file |

## Reading Vectors and Lists

| | |
|---|---|
| `scan(file, what = numeric())` | read a vector or list from a file |

## Formatting and Printing

| | |
|---|---|
| `format(x)` | format a vector in a common format |
| `sprintf(fmt, ...)` | formatted printing of R objects |
| `cat(...)` | concatenate and print vectors |
| `print(x)` | print an R object |

## Reading Data Frames

| | |
|---|---|
| `read.table(file, header=FALSE)` | read a data frame from a file |
| `read.csv(file, header=FALSE)` | read a data frame from a csv file |

## Options for `read.table` and `read.csv`

| | |
|---|---|
| `header=`*true/false* | does first line contain variable names? |
| `row.names=`$\cdots$ | row names specification |
| `col.names=`$\cdots$ | variable names specification |
| `na.strings="NA"` | entries indicating `NA` values |
| `colClasses=NA` | the types associated with columns |
| `nrows=`$\cdots$ | the number of rows to be read |

## Writing Data Frames

| | |
|---|---|
| `write.table(x, file)` | write a data frame to a file |
| `write.csv(x, file)` | write a data frame to a csv file |

## String Handling

| | |
|---|---|
| `paste(..., sep = " ", collapse = NULL)` | paste strings together |
| `strsplit(x, split)` | split `x` on pattern `split` (returns a list) |
| `grep(pattern, x)` | return subscripts of matching elements |
| `grep(pattern, x, value = TRUE)` | return matching elements |
| `sub(pattern, replacement, x)` | replace pattern with given replacement |
| `gsub(pattern, replacement, x)` | globally replace |

## High-Level Graphics

| | |
|---|---|
| `plot(x, y)` | scatter plot |
| `plot(x, y, type = "l")` | line plot |
| `plot(x, y, type = "n")` | empty plot |

## Adding to Plots

| | |
|---|---|
| `abline(a, b)` | line in intercept/slope form |
| `abline(h = yvals)` | horizontal lines |
| `abline(v = xvals)` | vertical lines |
| `points(x, y)` | add points |
| `lines(x, y)` | add connected polyline |
| `segments(x0, y0, x1, y1)` | add disconnected line segments |
| `arrows(x0, y0, x1, y1, code)` | add arrows |
| `rect(x0, y0, x1, y1, col)` | add rectangles filled with colours |
| `polygon(x, y)` | a polygon(s) |

## Low-Level Graphics

| | |
|---|---|
| `plot.new()` | start a new plot/figure/panel |
| `plot.window(xlim, ylim, ...)` | set up plot coordinates |

## Options to `plot.window`

| | |
|---|---|
| `xaxs="i"` | don't expand $x$ range by 8% |
| `yaxs="i"` | don't expand $y$ range by 8% |
| `asp=1` | equal-scale $x$ and $y$ axes |

## Graphical Parameters

| | |
|---|---|
| `par(...)` | set/get graphical parameters |

## Useful Graphical Parameters

| | |
|---|---|
| `mfrow = c(`$m$`,`$n$`)` | set up an $m$ by $n$ array of figures, filled by row |
| `mfcol = c(`$m$`,`$n$`)` | set up an $m$ by $n$ array of figures, filled by column |
| `mar=c(`$m_1$`,`$m_2$`,`$m_3$`,`$m_4$`)` | set the plot margins (in lines) |
| `mai=c(`$m_1$`,`$m_2$`,`$m_3$`,`$m_4$`)` | set the plot margins (in inches) |
| `cex=`$m$ | set the basic font magnification to $m$ |
| `bg=`*col* | set the device background to *col* |

## Measuring Text Size

| | |
|---|---|
| `strwidth(x, "inches", font, cex)` | widths of text strings in inches |
| `strheight(x, "inches", font, cex)` | heights of text strings in inches |

## Layouts

| | |
|---|---|
| `layout(mat,heights,widths)` | set up a layout |
| `layout.show(n)` | show layout elements (up to `n`) |
| `lcm(x)` | size specification in cm |

## Compound Expressions

| | |
|---|---|
| `{` $expr_1$`, ...,` $expr_n$`}` | compound expressions |

## Alternation

| | |
|---|---|
| `if (`*cond*`)` $expr_1$ `else` $expr_1$ | conditional execution |
| `if (`*cond*`)` *expr* | conditional execution, no alternative |

## Iteration

| | |
|---|---|
| `for (`*var* `in` *vector*`)` *expr* | `for` loops |
| `while (`*cond*`)` *expr* | `while` loops |
| `repeat` *expr* | infinite repetition |
| `continue` | jump to end of enclosing loop |
| `break` | break out of enclosing loop |

## Function Definition

| | |
|---|---|
| `function(`*args*`)` *expr* | function definition |
| *var* | function argument with no default |
| *var*`=`*expr* | function argument with default value |
| `return(`*expr*`)` | return the given value from a function |
| `missing(a)` | true if argument `a` was not supplied |

## Error Handling

| | |
|---|---|
| `stop(`*message*`)` | terminate a computation with an error message |
| `warning(`*message*`)` | issue a warning message |
| `on.exit(`*expr*`)` | save an expression for execution on function return |

## Language Computation

| | |
|---|---|
| `quote(`*expr*`)` | returns the expression *expr* unevaluated |
| `substitute(`*arg*`)` | returns the expression passed as argument *arg* |
| `substitute(`*expr*`,`*subs*`)` | make the specified substitutions in the given expression |

## Interpolation

| | |
|---|---|
| `approx(x, y, xout)` | linear interpolation at `xout` using `x` and `y` |
| `spline(x, y, xout)` | spline interpolation at `xout` using `x` and `y` |
| `approxfun(x, y, xout)` | interpolating linear function for `x` and `y` |
| `splinefun(x, y, xout)` | interpolating spline for `x` and `y` |

## Root-Finding and Optimisation

| | |
|---|---|
| `polyroot(coef)` | roots of polynomial with coefficients in `coef` |
| `uniroot(f,interval)` | find a root of the function `f` in the given interval |
| `optimize(f,interval)` | find an extreme of the function `f` in the given interval |
| `optim(x,f)` | find an extreme of the function `f` starting at the point `x` |
| `nlm(f,x)` | an alternative to `optim` |
| `nlminb(x,f)` | optimization subject to constraints |

## Integration

| | |
|---|---|
| `integrate(x,lower,upper)` | integrate the function `f` from `lower` to `upper` |