**R Quick Reference,** by E. Slate and E. Hill, adapted from the "R Reference Card" by Jonathan Baron. Parentheses are for functions, brackets are for indicating the position of items in a vector or matrix.

### Miscellaneous

| | |
|---|---|
| `q()` | quit |
| `options()` | view/set global options, e.g. number of digits |
| `history()` | view past commands you've issued |
| `<-` | assignment |
| `INSTALL package1` | install `package1` |
| `m1[,2]` | column 2 of matrix `m1` |
| `m1[,2:5]` or `m1[,c(2,3,4,5)]` | columns 2–5 |
| `m1$a1` | variable `a1` in data frame or list `m1` |
| `NA` | missing data |
| `is.na()` | true if data missing |
| `library(mva)` | load the package (e.g.) `mva` |
| `require(mva)` | load the package (e.g.) `mva`, if not already loaded |
| `NaN` | not a number |
| `Inf` | infinity |
| `data()` | available data sets |
| `demo()` | run demos |

### Help

| | |
|---|---|
| `help(command1)` | get help with `command1` (*use this command for more detail than is provided here*) |
| `help.start()` | start browser help |
| `help(package=mva)` | help with (e.g.) package `mva` |
| `apropos("topic1")` | commands relevant to `topic1` |
| `example(command1)` | examples of `command1` |
| `args(fn1)` | show arguments for function `fn1` |

### Input and output

| | |
|---|---|
| `source("file1")` | run the commands in `file1` |
| `read.table("file1"), read.csv("file1"), read.delim("file1")` | read in data from `file1` |
| `data.entry()` | spreadsheet |
| `scan("file1")` | read from `file1` (primitive) |
| `download.file(url1)` | from internet |
| `url.show(url1), read.table.url(url1)` | remote input |
| `sink("file1")` | output to `file1`, until `sink()` |
| `write(object, "file1")` | writes an object to `file1` |
| `write.table(dataframe1,"file1")` | writes a table to `file1` |

### Arithmetic

| | |
|---|---|
| `%*%` | matrix multiplication |
| `%/%, ^, %%, sqrt()` | integer division, power, modulus, square root |
| `outer` | outer "product" function |

### Managing variables and objects

| | |
|---|---|
| `attach(x1)` | put variables in `x1` in search path |
| `detach(x1)` | remove from search path |
| `search()` | view the search path |
| `ls()` | lists all the active objects |
| `rm(object1)` | removes `object1` |
| `save(obj), save.image(fname)` | save the workspace |
| `load(fname)` | load the workspace in `fname` |
| `dim(matrix1)` | dimensions of `matrix1` |
| `dimnames(x1)` | names of dimensions of `x1` |
| `names(df1)` | variable names in data frame df1 |
| `length(vector1)` | length of `vector1` |
| `1:3` | the vector `1,2,3` |
| `c(1,2,3)` | creates the same vector |
| `seq(from,to,by), seq(from,to,length)` | create a sequence |
| `rep(x1,n1)` | repeats the vector `x1 n1` times |
| `cbind(a1,b1,c1), rbind(a1,b1,c1)` | binds columns or rows into a matrix |
| `merge(df1,df2)` | merge data frames |
| `matrix(vector1,r1,c1)` | make `vector1` into a matrix with `r1` rows and `c1` columns |
| `data.frame(var1=v1,var2=v2)` | make a data frame from vectors `v1` and `v2` |
| `as.factor(), as.matrix(), as.vector()` | conversion |
| `is.factor(), is.matrix(), is.vector()` | what it is—returns `TRUE/FALSE` |
| `t()` | transpose |
| `which(x1==a1)` | returns indices of `x1` where `x1==a1` |
| `unique(x1)` | returns the unique elements in `x1` |
| `is.element(e, x1), union(x1,x2), setdiff(x1,x2), intersect(x1,x2)` | set operations |

### Control flow

| | |
|---|---|
| `for (i1 in vector1){...}` | repeat `length(vector1)` times |
| `if (condition1){...} else {...}` | conditional |
| `while (condition) {...}` | do while condition is `TRUE` |

### Logic

| | |
|---|---|
| `! x` | NOT `x`, elementwise |
| `x & y` | elementwise AND, all elements evaluated |
| `x && y` | sequential AND, only first elements of `x`, `y` used |
| `x \| y, x \|\| y` | elementwise and sequential OR |
| `xor(x, y)` | elementwise exclusive OR |

**Numerical summaries**

| | |
|---|---|
| `max(), min(), mean(), quantile(), median(), sum(), var(), cor()` | as named |
| `summary(data.frame)` | prints statistics |
| `rank(), sort(), order()` | ranking and sorting |
| `ave(x1,y1)` | averages of `x1` grouped by factor `y1` |
| `by()` | apply function to data frame by factor |
| `apply(x1,n1,function1)` | apply `function1` (e.g. `mean`) to x by rows (`n1=1`) or columns (`n2=2`) |
| `tapply(x1,list1,function1)` | apply function to `x1` by `list1` |
| `lapply(list1, function1)`<br>`sapply(list1, function1)` | apply `function1` to the elements of `list1; sapply` *simplifies* |
| `table()` | make a table |
| `tabulate()` | tabulate a vector |

**Basic statistical analysis**

| | |
|---|---|
| `aov(), anova(), lm(), glm(), nls(), nlm()` | linear and nonlinear models, anova |
| more for regression: `confint(), deviance(), df.residual(), rstandard(), rstudent(), dffits(), dfbetas(), cooks.distance(), hatvalues(), vcov(), predict()` | |
| `t.test()` | t test |
| `prop.test(), binom.test()` | proportions tests |
| `chisq.test(matrix1)` | chi-square test on `matrix1` columns |
| `fisher.test()` | Fisher exact test |
| `cor(a)` | show correlations |
| `cor.test(a,b)` | test correlation |
| `friedman.test()` | Friedman test |
| `runif(), rnorm(), rgamma(), rbeta(), rchisq(), rbinom(), rt(), rpois(), etc.` | probability distributions; prefix is r = random, p = prob, d = density, q = quantile |
| `optim` | general purpose optimization |
| `contrasts()` | set dummy coding for factor variables |
| `library(help="stats")` | additional standard stat methods |

**Some statistics in** mva **package**

| | |
|---|---|
| `prcomp()` | principal components |
| `kmeans()` | kmeans cluster analysis |
| `factanal()` | factor analysis |
| `cancor()` | canonical correlation |

**Graphics**

| | |
|---|---|
| `windows(), postscript(), pdf()` | new graphics device |
| `plot(x,y), barplot(), boxplot(), stem(), hist()` | basic plots |
| `matplot(xmat, ymat)` | matrix plot `ymat[,i]` on `xmat[,i]` |
| `pairs(matrix)` | scatterplots of all pairs of columns |
| `scatter.smooth()` | scatterplot with smooth trend |
| `coplot()` | conditional plot |
| `stripplot()` | strip plot (lattice) |
| `qqplot()` | quantile-quantile plot |
| `qqnorm(), qqline()` | fit normal distribution |
| `contour(), persp()` | plots for 3D data |
| `heatmap()` | |
| `points(), lines(), segments(), arrows(), text(), polygon(), symbols(), abline()` | add to current plot |
| `mfrow(), layout()` | multiple plotting regions per page |
| `par()` | View/set graphics parameters |
| `lattice package: xyplot(), bwplot(), densityplot(), cloud(), wireframe(), splom(), parallel()` | trellis graphics |

**Programmimg**

| | |
|---|---|
| `print(), cat(), traceback(), options(error = dump.frames), debugger()` | debugging |
| `substitute()` | read the help carefully! |
| `missing()` | check for missing function arguments |
| `...` | additional `name = value` type arguments |
| `match.call()` | expand function arguments |
| `do.call()` | evaluate a constructed function call |
| `R CMD` | DOS/unix command line interface |

**Useful packages**

| | |
|---|---|
| `lattice` | trellis graphics |
| `survival` | survival analyses |
| `mvtnorm` | multivariate normal and t distributions |
| `maps` | displaying geographical data |
| `boot` | bootstrapping |
| `nnet` | neural nets |
| `nlme, lme4` | linear and nonlinear mixed models |
| `hmisc, xtable` | exporting tables to LateX, HTML, plus others |
| `coda` | handling output from BUGS (see also the BOA code) |
| `ellipse` | for elliptical confidence regions in plots |
| `rggobi` | graphical data exploration |
| `gvlma` | a new diagnostic for linear models |
| `cluster` | various clustering methods |
| `mclust` | model-based (i.e. Gaussian) clustering |
| `gam` | generalized additive models |
| `BRugs, rbugs, R2WinBUGS` | linking R and Win/OpenBUGS |

## Übersicht: Operatoren, Funktionen, Werte

| Operator | Erklärung |
|---|---|
| `+, -, *, /` | Grundrechenarten |
| `^` | Potenz |
| `%%` | Modulo Division |
| `%/%` | Ganzzahlige Division |
| `&, \|, ==, !, !=, <, >, <=, >=` | Logische Operatoren |
| | `("und", "oder", "gleich", "nicht", "ungleich",...)` |
| Funktion | |
| `abs()` | Betrag |
| `sqrt()` | Wurzel |
| `round(), floor(), ceiling()` | Rundungsfunktionen |
| `log(), log10(), log2()` | Logarithmus |
| `exp()` | Exponentialfunktion |
| `sin(), cos(), tan(),` `asin(), acos(), atan()` | Trigonometrische Funktionen |
| Werte | |
| `pi` | $\pi$ |
| `Inf, -Inf` | Unendlich |
| `NaN` | nicht definiert |
| `NA` | fehlender Wert |
| `NULL` | leere Menge |

## Datentypen

- Jedes Objekt besitzt einen Datentyp. Folgend die wichtigsten Datentypen.

| Datentyp | Erklärung | Beispiel |
|---|---|---|
| *logical* | logische Werte | `TRUE, FALSE` |
| *numeric* | ganze oder reelle Zahlen | `2.54` |
| *complex* | komplexe Zahlen | `1.75 + 3i` |
| *character* | Zeichenketten | `"IBE"` |
| *factor* | Nominale und Ordinale Werte | `as.factor("CSU")` |
| *list* | Listen | |
| In jedem Typ steht `NA` für *Missing Value*. | | |

## Datenstrukturen

- Die Daten können auf verschiedene Weise angeordnet werden. Folgende Anordnungen (Strukturen) werden in diesem Kurs vorgestellt:

| Datenstruktur | Beispiel |
|---|---|
| *Vektoren* | `c(1,2,3,4)` |
| *Matrizen* | `matrix(1:6, nrow=2)` |
| *Listen* | `list(matrix(1:6, nrow=2), c(1,2,3), "Hallo IBE")` |
| *Datensätze* | `data.frame(PatId=c(3525,1242), Alter=c(25, 29))` |

## Übersicht: Datenstrukturen

| | |
|---|---|
| `data.class()` | allgemeine Abfrage der Datenklasse |
| `mode()` | Abfrage der Speicherart |
| `typeof()` | Abfrage der internen Speicherart (z.B. *integer* vs. *double*) |
| Datentyp | |
| `is.numeric(), is.integer(), is.double(),` `is.character(), is.logical(), is.na(),` `is.complex(), is.factor(), is.null()` | Überprüfen des Datentyps |
| `as.numeric(), as.integer(), as.double(),` `as.character(), as.logical()` `as.factor, as.complex()` | Modifikation des Datentyps |
| Datenstruktur | |
| `is.vector(), is.matrix(), is.array(),` `is.factor(), is.list(),` `is.data.frame(), is.function()` | Überprüfen der Datenstruktur |
| `as.vector(), as.matrix(), as.array(),` `as.list(), as.data.frame(), as.function()` | Modifikation der Datenstruktur |

## Übersicht: Vektorfunktionen

| Funktion | Erklärung |
|---|---|
| `c()` | Zusammenfügen von Elementen/Vektoren |
| `seq()` | Sequenzen |
| `rep()` | Wiederholungen |
| `max(), min()` | Extremwerte |
| `sum(), prod()` | Summe/ Produkt |
| `cumsum(), cumprod()` | kumulative Summe/ Produkt |
| `length()` | Länge |
| `sort()` | Sortiert den Vektor |
| `order()` | Indizes nach denen der Vektor sortiert wäre... |
| `x[order(x)]==sort(x)` | ...ist also überall *TRUE* |
| `y[order(x)]` | Sortiert y nach x |
| | |
| `rank()` | Rangzahlen |
| `which(` *expression* `)` | Indizes der Elemente auf die *expression* zutrifft. |
| *a* `%in%` *b* | *TRUE* für alle Elemente in *a* die auch in *b* sind |

## Übersicht: Matrixbefehle

Standardoperationen funktionieren, wie bei Vektoren *elementweise*! Das gilt auch für Operationen zwischen Matrizen (**Achtung**: Die beiden Matrizen müssen die gleiche Dimension besitzen).

| Funktion | Erklärung |
|---|---|
| `%*%` | Matrixmultiplikation |
| `t()` | Transponieren |
| `crossprod(x,y)` | Kreuzprodukt (entspricht: `t(x) %*% y`) |
| `diag()` | Hauptdiagonale |
| `dim(), nrow(), ncol()` | Zeilen-, Spaltenanzahl |
| `dimnames()` | Zeilen-, Spaltennamen |
| `eigen()` | Eigenwerte und -vektoren |
| `solve()` | Invertierung |

## Übersicht: Befehle für Data Frames

| Erzeugen | |
|---|---|
| `data.frame()` | Erzeugen |
| `as.data.frame()` | Umwandeln von Objekten in einen Data Frame |
| Arbeiten mit Data Frames | |
| `str()` | Anzeigen der Struktur (geht für alle Objekte) |
| `subset()` | Erzeugen eines Teildatensatzes |
| `split()` | Aufteilen in Teildatensätze gemäß einer Gruppierungsvariablen |
| `merge()` | Zusammenfügen von Datensätzen gemäß einer gemeinsamen Variablen |
| R Suchpfad | (nicht nur für Data Frames) |
| `attach()` | Hinzufügen |
| `detach()` | Herausnehmen |
| `search()` | Inhalt des Suchpfades anzeigen |

## Übersicht: Verteilungen

| Bsp. Normalverteilung | |
|---|---|
| `dnorm()` | Dichte |
| `pnorm()` | Verteilungsfunktion |
| `qnorm()` | Quantilfunktion |
| `rnorm()` | Zufallszahlen |
| entsprechend | |
| `..binom()` | Binomialverteilung |
| `..chisq()` | $\chi^2$–Verteilung |
| `..exp()` | Exponentialverteilung |
| `..f()` | F–Verteilung |
| `..hyper()` | Hypergeometrische Verteilung |
| `..logis()` | Logistische Verteilung |
| `..multinom()` | Multinomialverteilung |
| `..pois()` | Poissonverteilung |
| `..signrank()` | Verteilung der Wilcoxon Vorzeichenstatistik |
| `..t()` | Student t–Verteilung |
| `..unif()` | Gleichverteilung |
| `..wilcox()` | Verteilung der Wilcoxon Rangsummenstatistik |
| ⋮ | weitere siehe `help.search("distribution")` |

## Übersicht: Deskriptive Statistik

| Wichtige Kenngrößen | |
|---|---|
| `mean()` | Mittelwert |
| `median()` | Median |
| `quantile()` | Quantile |
| `min(), max()` | Minimum, Maximum |
| `var(), sd()` | Varianz, Standardabweichung |
| `cov(), cor()` | Kovarianz, Korrelation |
| Zusammenfassung | |
| `summary()` | Mittelwert, Quantile etc. bei stetigen<br>bzw. Häufigkeiten bei kategorialen Variablen |
| `table()` | Häufigkeiten der einzelnen Werte |

## Übersicht: Grafiken

| Grafikbefehle | Erklärung |
|---|---|
| `plot()` | allgemeine plot–Funktion, meist Scatterplot |
| `hist(), boxplot(),` | Histogramm, Boxplot |
| `barplot(), pie()` | Säulendiagramm, Kuchendiagramm |
| `plot(density()), qqnorm()` | Kerndichteschätzer, Q–Q–Plot |
| `ts.plot(), curve()` | Zeichnen von Zeitreihen, Funktionen |
| `persp(), contour(), ...` | dreidimensionaler Plot, Contour Plot |
| Häufige Optionen | |
| `main, sub, xlab, ylab,` | Titel, Untertitel, Achsenbeschriftung |
| `xlim, ylim` | Skalierung der Achsen |
| `col, pch, lty` | Farbe, Symbol, Linientyp |
| `axes=TRUE` | sollen Achsen gezeichnet werden? |
| ⋮ | |
| Graphische Parameter mit `par()` | |
| `mfrow=c(`$a$`,`$b$`)` | Darstellung von $a \times b$ Grafiken gleichzeitig |
| `new=TRUE` | neue Grafik wird zu bestehender hinzugefügt |
| ⋮ | fast alles kann eingestellt werden ... |
| Hinzufügen von | |
| `lines(), abline(), polygon()` | Kurve, Gerade, Polygonzug |
| `points(), legend(), text()` | Punkt, Legende, Text |
| `axis(), title()` | Achse, Titel |

## Übersicht: Einige Tests

| stetige Zielgröße | |
|---|---|
| `t.test()` | one / two sample t–Test (auch für verb. Stichpr.) |
| `wilcox.test()` | one / two sample Wilcoxon–Test (Mann–Whitney)<br>(auch für verb. Stichpr.) |
| `ks.test()` | one / two sample Kolmogorov–Smirnov–Test |
| `aov()` | Varianzanalyse |
| kategoriale Zielgröße | |
| `binom.test()` | Binomialtest |
| `chisq.test()` | $\chi^2$–Test |
| `fisher.test()` | Fishers exakter Test |
| `prop.test()` | Test auf Anteile |

# R - COMMANDS

by Hubert Hackl - Institute for Genomics and Bioinformatics - Graz University of Technology

---

MISCELLANOUS

---

```
<-                  assignment operator
q()                 quit
data()              include predefined dataset
attach()            include dataset in path
detach()            exclude dataset of path
search()            search in path
help(plot)          help for command
?plot               search help for command
demo()              demonstration
library(mwa)        include library mva
INSTALL package     install package
getwd()             get working directory
setwd()             set working directory
dir()               list content of working directory
ls()                list current objects (functions, variables..)
source()            run R script
rm()                remove current objects
sink()              redirect output from console to file
```

---

ASSIGNMENT AND CALCULATIONS

---

```
> a<-c(1,2,3,4,5)
> a
  [1] 1 2 3 4 5
> a<-c(1:5)
> a
  [1] 1 2 3 4 5

> q<-rep(a,2)
> q
  [1] 1 2 3 4 5 1 2 3 4 5
> e<-c(rep("n",5),rep("y",6))
> e
  [1] "n" "n" "n" "n" "n" "y" "y" "y" "y" "y" "y"

> x<-a*2+3
> x
  [1]  5  7  9 11 13
> x*1/x
  [1] 1 1 1 1 1
> x*x
  [1]  25  49  81 121 169

> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
> y <- c(x, x)
> y
  [1] 10.4  5.6  3.1  6.4 21.7 10.4  5.6  3.1  6.4 21.7
> x
  [1] 10.4  5.6  3.1  6.4 21.7
> v <- 2 * x + y + 1
> v
 [1] 32.2 17.8 10.3 20.2 66.1 32.2 17.8 10.3 20.2 66.1

> k<-seq(2,20,by=2)
> k
  [1]  2  4  6  8 10 12 14 16 18 20
```

```
> z<-seq(-pi,pi,len=11)
> z
  [1] -3.1415927 -2.5132741 -1.8849556 -1.2566371 -0.6283185  0.0000000
  [7]  0.6283185  1.2566371  1.8849556  2.5132741  3.1415927

> y<-seq(1,13,by=2)
> y
  [1]  1  3  5  7  9 11 13
> y[1]<-7
> y
  [1]  7  3  5  7  9 11 13

> data.entry(y)     # can do the same
```

---

ORDER OF ELEMENTS

---

```
> rev(1:10)
  [1] 10  9  8  7  6  5  4  3  2  1

> g<-c(8,4,7,9)
> rank(g)
  [1] 3 1 2 4
> order(g)
  [1] 2 3 1 4
> sort(g)
  [1] 4 7 8 9
> sort(g, decreasing=T)
  [1] 9 8 7 4
> h<-c(2,3,1,4)
> g[order(h)]
  [1] 7 8 4 9
```

---

SUBSET OF VECTOR

---

```
> z<-seq(-pi,pi,len=11)
> z[9:11]
  [1] 1.884956 2.513274 3.141593
> z[-(9:11)]
  [1] -3.1415927 -2.5132741 -1.8849556 -1.2566371 -0.6283185  0.0000000  0.6283185  1.2566371
> z[-1]
  [1] -2.5132741 -1.8849556 -1.2566371 -0.6283185  0.0000000  0.6283185  1.2566371  1.8849556
2.5132741  3.1415927
> z[-2]
  [1] -3.1415927 -1.8849556 -1.2566371 -0.6283185  0.0000000  0.6283185  1.2566371  1.8849556
2.5132741  3.1415927
> z[z>0]
  [1] 0.6283185 1.2566371 1.8849556 2.5132741 3.1415927
> which(z>0)
  [1]  7  8  9 10 11
```

---

DATA STRUCTURE

---

```
> x
[1]  5  7  9 11 13
> mode(x)
[1] "numeric"
> typeof(x)
[1] "double"
> data.class(x)
[1] "numeric"
```

```
> is.vector(x)
[1] TRUE
> is.matrix(x)
[1] FALSE
> str(x)
 num [1:5] 5 7 9 11 13
```

---

VECTOR FUNCTIONS AND DESCRIPTIVE STATISTICS

---

```
> sum(x)
  [1] 45
> prod(x)
  [1] 45045
> cumsum(x)
  [1]  5 12 21 32 45
> cumprod(x)
  [1]     5    35   315  3465 45045
> max(x)
  [1] 13
> min(x)
  [1] 5
> length(x)
  [1] 5
> mean(x)
  [1] 9
> median(x)
  [1] 9
> mad(x)    #median absolute deviation
  [1] 2.9652
> var(x)
  [1] 10
> sd(x)
  [1] 3.162278
> table(x)
  x
   5  7  9 11 13
   1  1  1  1  1
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      5       7       9       9      11      13
> fivenum(x)
  [1]  5  7  9 11 13
> quantile(x,0.95)
   95%
  12.6
```

---

MISSING VALUES

---

```
> x<-c(1,2,3,NA,6,5)
> x[!is.na(x)]
  [1] 1 2 3 6 5
> mean(x)
  [1] NA
> mean(x,na.rm=T)
  [1] 3.4
> is.na(x)
  [1] FALSE FALSE FALSE  TRUE FALSE FALSE
```

---

RANDOM NUMBERS

---

```
> rn<-rnorm(12)
> rn
   [1] -0.8007859  0.2431818 -0.1990018  1.6554131  0.3707471 -1.3375725  1.0086379 -0.5049361 -
3.2712872 -0.8691483
  [11] -0.1805588 -0.7649395
> rn<-rnorm(12)
> rn
   [1] -0.10310011 -0.59889805 -0.86924723  0.06923738 -0.53107761  0.42140447 -1.02597246 -
0.19028913 -0.06164463
  [10] -0.25752183  1.17905023  1.45788310
> tabulate(c(2,3,3,5), nbins = 10)
 [1] 0 1 2 0 1 0 0 0 0 0
```

---

MATRIX

---

```
> x <- matrix(1:6, ncol=3)
> x
      [,1] [,2] [,3]
  [1,]    1    3    5
  [2,]    2    4    6
> u
  [1]  5  7  9 11 13
> v<-seq(2,15,len=5)
> v
  [1]  2.00  5.25  8.50 11.75 15.00
> M<-cbind(u,v)
> M
       u     v
  [1,]  5  2.00
  [2,]  7  5.25
  [3,]  9  8.50
  [4,] 11 11.75
  [5,] 13 15.00
> N<-rbind(u,v)
> N
    [,1] [,2] [,3]  [,4] [,5]
  u    5 7.00  9.0 11.00   13
  v    2 5.25  8.5 11.75   15
> P<-t(N)

> P
       u     v
  [1,]  5  2.00
  [2,]  7  5.25
  [3,]  9  8.50
  [4,] 11 11.75
  [5,] 13 15.00
> Q<-P[2:4,]
> Q
       u     v
  [1,]  7  5.25
  [2,]  9  8.50
  [3,] 11 11.75
> Q<-P[1,2]
> Q
  [1] 2
> is.matrix(Q)
  [1] FALSE
```

---

FUNCTION

---

```
> fsqu<-function(x){return(x*x+2.3)}
> fsqu(4)
  [1] 16
> x<-c(2,3,4,NA,7,6)
> x
  [1]  2  3  4 NA  7  6
> fsqu(x)
  [1]  6.3 11.3 18.3   NA 51.3 38.3
```

---

ROUND

---

```
> round(0.032456,digit=3)
  [1] 0.032
> signif(0.032456,digit=3)
  [1] 0.0325
> trunc(1.03245)
  [1] 1
> ceiling(1.03245)
  [1] 2
> floor(1.03245)
  [1] 1
```

---

LIST

---

```
> list(matrix(1:6, ncol=3),"rot", 1:4)
  [[1]]
       [,1] [,2] [,3]
  [1,]    1    3    5
  [2,]    2    4    6

  [[2]]
  [1] "rot"

  [[3]]
  [1] 1 2 3 4

> coliste<-list(werte=matrix(1:6, ncol=3),farbe="blau", reihe=1:4)
> coliste
  $werte
       [,1] [,2] [,3]
  [1,]    1    3    5
  [2,]    2    4    6
  $farbe
  [1] "blau"
  $reihe
  [1] 1 2 3 4
> names(coliste)
  [1] "werte" "farbe" "reihe"
> coliste[[1]]
       [,1] [,2] [,3]
  [1,]    1    3    5
  [2,]    2    4    6
> coliste[["farbe"]]
  [1] "blau"
> coliste$farbe
  [1] "blau"
> coliste[[c(3,2)]]
  [1] 2
```

---

FACTOR

---

```
> geschlecht <- factor(c("m", "m", "w", "m", "w", "w", "w", "m", "m", "w", "m", "w"))
> geschlecht
   [1] m m w m w w w m m w m w
  Levels: m w

> x<-c(1,2,3,4,9,8)
> cut(x,breaks=seq(1,9,length=9),right=FALSE)
  [1] [1,2) [2,3) [3,4) [4,5) <NA>  [8,9)
  Levels: [1,2) [2,3) [3,4) [4,5) [5,6) [6,7) [7,8) [8,9)
> table(cut(x,breaks=seq(1,9,length=9),right=FALSE))

  [1,2) [2,3) [3,4) [4,5) [5,6) [6,7) [7,8) [8,9)
      1     1     1     1     0     0     0     1
```

---

DATAFRAME

---

```
> a <- c(10,20,15,43,76,41,25,46)                          # numerisch
> b <- factor(c("m", "w", "m", "w", "m", "w", "m", "w")) # Faktor Geschlecht: m=männlich,
w=weiblich
> c <- c(2,5,8,3,6,1,5,6)                                  # numerisch
> myframe <- data.frame(a,b,c)
> myframe
    a b c
  1 10 m 2
  2 20 w 5
  3 15 m 8
  4 43 w 3
  5 76 m 6
  6 41 w 1
  7 25 m 5
  8 46 w 6


> data(iris)                            # Load predefined dataset of package 'base' in R

> is.data.frame(iris)
  [1] TRUE
> is.matrix(iris)
  [1] FALSE
> iris
      Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
  1            5.1         3.5          1.4         0.2     setosa
  2            4.9         3.0          1.4         0.2     setosa
  3            4.7         3.2          1.3         0.2     setosa
  4            4.6         3.1          1.5         0.2     setosa
  5            5.0         3.6          1.4         0.2     setosa
  6            5.4         3.9          1.7         0.4     setosa
  ...
> nrow(iris)
  [1] 150
> ncol(iris)
  [1] 5
> dim(iris)
  [1] 150   5
> colnames(iris)
  [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
> rownames(iris)
  [1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"  "11"  "12"
 [13] "13"  "14"  "15"  "16"  "17"  "18"  "19"  "20"  "21"  "22"  "23"  "24"
 [25] "25"  "26"  "27"  "28"  "29"  "30"  "31"  "32"  "33"  "34"  "35"  "36"
 ...
```

```
> subset(iris,Species=="virginica")
     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
  101          6.3         3.3          6.0         2.5 virginica
  102          5.8         2.7          5.1         1.9 virginica
  103          7.1         3.0          5.9         2.1 virginica
  104          6.3         2.9          5.6         1.8 virginica
  105          6.5         3.0          5.8         2.2 virginica
  ...
> subset(iris,Species %in% c("setosa","virginica"))
     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
  1            5.1         3.5          1.4         0.2     setosa
  2            4.9         3.0          1.4         0.2     setosa
  3            4.7         3.2          1.3         0.2     setosa
  4            4.6         3.1          1.5         0.2     setosa
  ...
  101          6.3         3.3          6.0         2.5 virginica
  102          5.8         2.7          5.1         1.9 virginica
  103          7.1         3.0          5.9         2.1 virginica
  ...
> irisa<-data.frame(iris[,c(1,3,4)])
> irisa
     Sepal.Length Petal.Length Petal.Width
  1            5.1          1.4         0.2
  2            4.9          1.4         0.2
  3            4.7          1.3         0.2
  4            4.6          1.5         0.2
```

---

APPLY, TAPPLY, LAPPLY, SAPPLY, AGGREGATE, BY

---

```
> apply(irisa,1,mean)             # apply function mean to rows (rows=1, columns=2) of irisa
         1         2         3         4         5         6         7         8         9
10        11        12        13        14
   11.166667 10.833333 10.333333 10.500000 11.000000 12.500000 10.500000 11.166667 10.000000
10.833333 11.833333 11.000000 10.500000  9.166667
   ....
> apply(irisa,2,mean)             # apply function mean to columns of irisa
  Sepal.Length Petal.Length  Petal.Width
     29.216667    18.790000     5.996667
> mean(irisa)                                 #same as apply(irisa,2,mean)
  Sepal.Length Petal.Length  Petal.Width
     29.216667    18.790000     5.996667
> sapply(iris[1:3],mean)
  Sepal.Length  Sepal.Width Petal.Length
     5.843333     3.057333     3.758000
> lapply(iris[1:3],mean)
  $Sepal.Length
  [1] 5.843333

  $Sepal.Width
  [1] 3.057333

  $Petal.Length
  [1] 3.758
> is.vector(sapply(iris[1:3],mean))
  [1] TRUE
> is.list(lapply(iris[1:3],mean))
  [1] TRUE
> aggregate(iris$Sepal.Width,by=list(iris$Species),mean)
       Group.1     x
  1     setosa 3.428
  2 versicolor 2.770
  3  virginica 2.974
> tapply(iris$Sepal.Width,iris$Species,mean)
     setosa versicolor  virginica
      3.428      2.770      2.974
```

---

STATISTICS ON DATAFRAME

---

```
> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
> fivenum(iris$Sepal.Width)
  [1] 2.0 2.8 3.0 3.3 4.4
> quantile(iris$Sepal.Width)
    0%  25%  50%  75% 100%
   2.0  2.8  3.0  3.3  4.4
> quantile (iris$Sepal.Width,0.95)
  95%
  3.8
> fivenum(iris$Sepal.Width[15:50])
  [1] 2.30 3.20 3.45 3.75 4.40
```

---

INPUT-OUTPUT

---

```
> chol<-scan("cholesterol.txt")
  Read 86 items
> is.vector(chol)
  [1] TRUE
> chol
   [1]  3.7  3.8  3.8  4.4  4.5  4.5  4.5  4.7  4.7  4.8  4.8  4.9  4.9  4.9  5.0  5.1  5.1  5.2
5.3  5.3  5.4
  [22]  5.4  5.5  5.5  5.5  5.6  5.6  5.6  5.6  5.6  5.6  5.6  5.7  5.7  5.7  5.8  5.8  5.9  6.0
6.1  6.1  6.1
  ...

> chol<-read.table("age.txt")
  Warning message:
  Anzahl der gelesenen Daten ist kein Vielfaches der Anzahl der Spalten
> chol
    V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11
  1 48 49 51 56 52 64 34 54 36  29  53
  2 51 39 72 78 59 68 54 51 47  39  48
  3 32 68 55 60 54 55 51 71 64  66  67
  4 48 67 61 40 41 33 63 56 54  61  55
  5 60 48 37 56 47 43 73 67 74  62  53
  6 52 51 53 58 44 37 42 49 42  55  47
  7 62 51 51 44 57 58 36 57 52  53  NA
  8 48 NA NA NA NA NA NA NA NA  NA  NA
> is.data.frame(chol)
  [1] TRUE
```

```
>write.table(age,"age2.txt",row.names=F,col.names=F)

   age2.txt:  48
              49
              51
              56
              52
              64
```

```
>write(x, file="output.txt"), append=T,sep="\t")
```

---

TEXT MANIPULATION

---

```
> Vornamen <- c("Hugo","Walter","Felix")
> nchar(Vornamen)
  [1] 4 6 5
> paste ("Lieber",Vornamen)
  [1] "Lieber Hugo"   "Lieber Walter" "Lieber Felix"
> paste(1:4,Vornamen)
  [1] "1 Hugo"   "2 Walter" "3 Felix"  "4 Hugo"
> substring(Vornamen,1,1)
  [1] "H" "W" "F"
> abbreviate(Vornamen,3)
  Hugo Walter  Felix
 "Hug"  "Wlt"  "Flx"
> paste(Vornamen,collapse="|")
  [1] "Hugo|Walter|Felix"
```

---

GRAPHICS

---

```
par()               graphics parameter
par(mfrow=c(1,2))   2 plots in 1 graph 1 row and 2 columns

plot()              scatter plot
boxplot()           box-and-whiskers plot
hist()              histogram
barplot()           bar chart
pie()               pie chart
curve()             plot a function
qqplot()            q-q-plot
qqnorm()            q-q-plot against normal distribution
stem()              stem-and-leaf diagramt
dotchart()          dot plot
stripchart()        strip chart

lines()             add line
abline()            add a straight line
qqline()            add line to qqplot
polygon(x,y)        add a polygon (x,y: vectors containing the coordinates of the vertices)
points()            add points
legend()            add legend
grid()              add grid

identify()   identify points in graph with left mouse button
win.graph() opens new graphics window
graphics.off() closes all garaphics windows


> curve(x^2,-2,2)
> curve(2*abs(x),-2,2,add=T)      # 2nd plot will be added to the 1st graph.


> par(mfrow=c(1,2),lwd=2,ps = 20,lty='dashed')   # 2 diagrams in 1 graph: 1 row 2 columns
> hist(x)
> hist(x, xlim = c(2,12), ylim = c(0, 30), col=gray(0.8), right=FALSE , main="Histogramm",
  xlab="Levels", ylab="Frequency",breaks=seq(2,12,length=10))
> axis(side=1,lwd=2)
> pie (x, labels= BG, main="Pie chart", col= col, init.angle=0)


> par(mfrow=c(1,1),lwd=2,ps = 20)
> a <- c(90,120,150,180,210,240,270,300)
> b <- c(1.2,1.26,1.31,1.34,1.36,1.39,1.40,1.44)
> plot (a,b, type="o",xlab="time",ylab="log2ratio",ylim = c(1.19, 1.45),xlim = c(90, 300),
  main="line plot")
> grid (7,10)
```

```
> boxplot(Sepal.Length ~Species)
> identify(Sepal.Length ~Species)

> B<-c(1.1,1.9,2.7,4.1,5.0)
> A<-c(1.3,2.0,3.0,4.6,5.9)
> Dif<-(A-B)
> Ave<-0.5*(A+B)
> plot(Ave,Dif, ylim = c(-2, 2), xlim = c(0, 7), xlab="Average of A and B",ylab="Difference
  (A- B)",font=2)
> abline(h=mean(Dif))
> text(143,0.8,labels="mean")

> h<-hist(x)
> h
  $breaks
  [1]  0  2  4  6  8 10
  $counts
  [1] 2 2 0 1 1
  $intensities
  [1] 0.16666663 0.16666667 0.00000000 0.08333333 0.08333333
  $density
  [1] 0.16666663 0.16666667 0.00000000 0.08333333 0.08333333
  $mids
  [1] 1 3 5 7 9
  $xname
  [1] "x"
  $equidist
  [1] TRUE
  attr(,"class")
  [1] "histogram"
> h$counts
  [1] 2 2 0 1 1

> h<-hist(x)
> h$counts <- cumsum(h$counts)
> h$density <- cumsum(h$density)
> h$itensities <- cumsum(h$itensities)
> plot(h, col=gray(0.8))

> plot (ecdf(x)
> plot (ecdf(x), xlab="levels", ylab="Empirical cumulative distribution function", verticals=TRUE,
  do.points=FALSE)

> qqnorm(x)
> qqline(x)

> stem(x,1)

> plot(x,y)    # is the same as plot (y~x)

> set.seed(1)
> x <- rnorm(25)
> par(mar = c(4, 4, 3, 1))
> hist(x, freq = FALSE, breaks = (-6:6)/2, main = "Histogramm")
> lines(density(x))
> curve(dnorm, -3, 3, add = TRUE, col = "red")

> win.graph(j, width=7, height=7)
> par(mfrow=c(1,1))
> hist(x,col=gray(0.8),xlim=c(min(x)-2,max(x)+2),freq=F,breaks=seq(min(x),max(x),length=k+1))
> lines(density(x),col="red")
> nam1<-paste("histogram_rel_",j,sep="")
> savePlot(filename=nam1, type="png")
```

---

CONTROL FLOW AND SCRIPTS

---

test.R:

```
ff<-function(x) {
   y<-x^2+3
}
par(mfrow=c(1,1))
my<-read.table("text.txt", header=FALSE)
for (i1 in 1:4) {
  win.graph(i1,)
  if (i1>2) {
     hist (my$V1[3:16])
  }
  else {
     var<-ff(my$V1)
     hist (var)
  }
}
```

Run script with source("test.R")

### Random samples

```
sample(1:6,size=10, replac=True)   ## roll a die 10 times
```

### Normal distribution

```
dnorm(x, mean=0, sd=1, log = FALSE)                          ## pdf
pnorm(q, mean=0, sd=1, lower.tail = TRUE, log.p = FALSE)   ## cdf
qnorm(p, mean=0, sd=1, lower.tail = TRUE, log.p = FALSE)   ## quantiles
rnorm(n, mean=0, sd=1)                                       ## random samples
```

### Uniform distribution

```
dunif(x, min=0, max=1, log = FALSE)                          ## pdf
punif(q, min=0, max=1, lower.tail = TRUE, log.p = FALSE)   ## cdf
qunif(p, min=0, max=1, lower.tail = TRUE, log.p = FALSE)   ## quantiles
runif(n, min=0, max=1)                                       ## random samples
```

### Binominal distribution

```
dbinom(x, size, prob, log = FALSE)
pbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE)
qbinom(p, size, prob, lower.tail = TRUE, log.p = FALSE)
rbinom(n, size, prob)
```

### Poisson distribution

```
dpois(x, lambda, log = FALSE)
ppois(q, lambda, lower.tail = TRUE, log.p = FALSE)
qpois(p, lambda, lower.tail = TRUE, log.p = FALSE)
rpois(n, lambda)
```

### Student t Distribution

```
dt(x, df, ncp, log = FALSE)
pt(q, df, ncp, lower.tail = TRUE, log.p = FALSE)
qt(p, df, ncp, lower.tail = TRUE, log.p = FALSE)
rt(n, df, ncp)
```

### Chi-Squared Distribution

```
dchisq(x, df, ncp=0, log = FALSE)
pchisq(q, df, ncp=0, lower.tail = TRUE, log.p = FALSE)
qchisq(p, df, ncp=0, lower.tail = TRUE, log.p = FALSE)
rchisq(n, df, ncp=0)
```

### F Distribution

```
df(x, df1, df2, ncp, log = FALSE)
pf(q, df1, df2, ncp, lower.tail = TRUE, log.p = FALSE)
qf(p, df1, df2, ncp, lower.tail = TRUE, log.p = FALSE)
rf(n, df1, df2, ncp)
```

### Quantiles for significance level alpha

```
zstar<-qnorm(1-alpha/2)
tstar<-qt(1-alpha/2, df=n-1)
```

### Confidence interval

```
t.test(x,conf.level=0.95)
prop.test(x=c(560,570),n=c(1000,1200),conf.level=0.95)  ## for proportions
t.test(x,y, var.equal=FALSE, conf.level=0.95)  ## differences of means for two independent samples
t.test(x,y, paired=TRUE) ## comparison of means from matched samples
```

### Hypothesis tests

```
prop.test(x, n, p = NULL, alternative = c("two.sided", "less", "greater"), conf.level = 0.95, correct =
TRUE) # for proportions
binom.test(x, n, p = 0.5, alternative = c("two.sided", "less", "greater"), conf.level = 0.95) #sign test
t.test(x, y = NULL, alternative = c("two.sided", "less", "greater"), mu = 0, paired = FALSE, var.equal =
FALSE, conf.level = 0.95, ...) # for normal distributed data
```

### Wilcoxon signed rank test

```
wilcox.test(x, y = NULL, alternative = c("two.sided", "less", "greater"), mu = 0, paired = TRUE, exact =
NULL, correct = TRUE,conf.int = FALSE, conf.level = 0.95, ...)
```

### Mann-Whitney U test

```
wilcox.test(x, y = NULL, alternative = c("two.sided", "less", "greater"), mu = 0, paired = FALSE, exact =
NULL, correct = TRUE,conf.int = FALSE, conf.level = 0.95, ...)
```

### F test

```
var.test(x, y, ratio = 1,alternative = c("two.sided", "less", "greater"),conf.level = 0.95, ...)
var.test(formula, data, subset, na.action, ...)
```

### One way ANOVA

```
data:
          V C
1 -0.4150107 A
2 -0.2889898 A
3 -0.3727820 A
4  1.7770802 B
5  1.0688233 B
6  1.3608474 B
7 -0.3260012 C
8 -0.3511097 C
9 -0.1009256 C

anova(lm(V ~ C, data))
oneway.test(V ~ C, data, subset, na.action, var.equal = FALSE)
pairwise.t.test(x, g, p.adjust.method = p.adjust.methods,pool.sd = TRUE, ...) # to find which group differs
```

### Krukal-Walis test

```
kruskal.test(x, g, ...)
kruskal.test(x~f,data, subset, na.action, ...)
```

### Chi-squared test

```
chisq.test(x, y = NULL, correct = TRUE,p = rep(1/length(x), length(x)), rescale.p = FALSE,
          simulate.p.value = FALSE, B = 2000)
```

### Fisher's exact test

```
fisher.test(x, y = NULL, workspace = 200000, hybrid = FALSE,control = list(), or = 1, alternative =
"two.sided",conf.int = TRUE, conf.level = 0.95,simulate.p.value = FALSE, B = 2000)
```

### McNemar test

```
mcnemar.test(x, y = NULL, correct = TRUE)
```

### Power of a test

```
power.prop.test(n = NULL, p1 = NULL, p2 = NULL, sig.level = 0.05,power = NULL,alternative = c("two.sided",
"one.sided"),strict = FALSE)
power.t.test(n = NULL, delta = NULL, sd = 1, sig.level = 0.05, power = NULL, type = c("two.sample",
"one.sample", "paired"),alternative = c("two.sided", "one.sided"),strict = FALSE)
```

```
================================================================================

Useful R commands to work with biological sequences

================================================================================


paste(..., sep = " ", collapse = NULL)

substr(x, 2, 5)

strsplit(x, split, extended = TRUE, fixed = FALSE, perl = FALSE)

grep(pattern, x, ignore.case = FALSE, extended = TRUE,
     perl = FALSE, value = FALSE, fixed = FALSE, useBytes = FALSE)

sub(pattern, replacement, x,
    ignore.case = FALSE, extended = TRUE, perl = FALSE,
    fixed = FALSE, useBytes = FALSE)

gsub(pattern, replacement, x,
     ignore.case = FALSE, extended = TRUE, perl = FALSE,
     fixed = FALSE, useBytes = FALSE)

regexpr(pattern, text, ignore.case = FALSE, extended = TRUE,
        perl = FALSE, fixed = FALSE, useBytes = FALSE)

gregexpr(pattern, text, ignore.case = FALSE, extended = TRUE,
         perl = FALSE, fixed = FALSE, useBytes = FALSE)

chartr(old, new, x)

tolower(x)

toupper(x)

casefold(x, upper = FALSE)

capwords <- function(s, strict = FALSE) {
    cap <- function(s) paste(toupper(substring(s,1,1)),
                  {s <- substring(s,2); if(strict) tolower(s) else s},
                             sep = "", collapse = " " )
    sapply(strsplit(s, split = " "), cap, USE.NAMES = !is.null(names(s)))
}

strReverse <- function(x) {
    sapply(lapply(strsplit(x, NULL), rev), paste, collapse="")
}

charmatch(x, table, nomatch = NA_integer_)

pmatch(x, table, nomatch = NA_integer_, duplicates.ok = FALSE)

match(x, table, nomatch = NA_integer_, incomparables = FALSE)
x %in% table

library(seqinr)

read.fasta(file = system.file("sequences/ct.fasta", package = "seqinr"),
  seqtype = "DNA", File = NULL, as.string = FALSE, forceDNAtolower = TRUE,
  set.attributes = TRUE, legacy.mode = TRUE, seqonly = FALSE, strip.desc = FALSE)


write.fasta(sequences, names, nbchar = 60, file.out, open = "w")
```

```
================================================================================

Example for working with biological sequences

================================================================================


fasta file (fasta1.txt)

>Acc12345|sequence 1
ATGCTGAAGTTCCAAACAGTTCGAGGGGGCCTGAGGCTCCTGGGTGTCCGCCGATCCTCC
TCGGCCCCTGTTGCCTCCCC


*** read in fasta file
> seq1<-read.fasta("fasta1.txt",as.string = TRUE,seqonly=TRUE)


*** how seq 1 looks like (=list)
> seq1
[[1]]
[1] "ATGCTGAAGTTCCAAACAGTTCGAGGGGGCCTGAGGCTCCTGGGTGTCCGCCGATCCTCCTCGGCCCCTGTTGCCTCCCC"

*** how access sequence in list
> seq1[[1]][1]
[1] "ATGCTGAAGTTCCAAACAGTTCGAGGGGGCCTGAGGCTCCTGGGTGTCCGCCGATCCTCCTCGGCCCCTGTTGCCTCCCC"


*** assign sequence to a new variable dna (dna is now vector not list)
> dna<-seq1[[1]][1]
> dna
[1] "ATGCTGAAGTTCCAAACAGTTCGAGGGGGCCTGAGGCTCCTGGGTGTCCGCCGATCCTCCTCGGCCCCTGTTGCCTCCCC"
> is.list(dna)
[1] FALSE
> is.vector(dna)
[1] TRUE


*** find "AGGCT" in sequence dna (start positions of matches and length of matches in list
> matches<-gregexpr("AG",dna)
> matches
[[1]]
[1]  8 18 24 34
attr(,"match.length")
[1] 2 2 2 2


*** find match length (e.g for third match)
> attr(matches[[1]],"match.length")[3]
[1] 2


*** function for string reverse
strReverse <- function(x) { sapply(lapply(strsplit(x, NULL), rev), paste, collapse="") }
> strReverse(dna)
[1] "CCCCTCCGTTGTCCCCGGCTCCTCCTAGCCGCCTGTGGGTCCTCGGAGTCCGGGGGAGCTTGACAAACCTTGAAGTCGTA"


*** make lower sequence
> tolower(dna)
[1] "atgctgaagttccaaacagttcgaggggcctgaggctcctgggtgtccgccgatcctcctcggcccctgttgcctcccc"
```

***** extract subsequence**

```
> dna_small<-substr(dna, 2, 13)
> dna_small
[1] "TGCTGAAGTTCC"
```

***** find complementary sequence**

```
> dna_small_compl<-chartr("AGCT", "TCGA", dna_small)
> dna_small_compl
[1] "ACGACTTCAAGG"
```

***** substitute all GAG, GCG, GTG, GGG with XXX**

```
> dna_subst<-gsub("G.G","XXX",dna)
> dna_subst
[1] "ATGCTGAAGTTCCAAACAGTTCXXXXXXGCCTXXXGCTCCTXXXTGTCCGCCGATCCTCCTCGGCCCCTGTTGCCTCCCC"
```

***** write fasta file with changed sequence**

```
write.fasta(sequences=dna_subst,names="seq", nbchar="30", file.out="fasta3.txt")
```

***** read in file linewise**

```
> dna2<-readLines("fasta1.txt")
> dna2
[1] ">Acc12345|sequence 1"
[2] "ATGCTGAAGTTCCAAACAGTTCGAGGGGGCCTGAGGCTCCTGGGTGTCCGCCGATCCTCC"
[3] "TCGGCCCCTGTTGCCTCCCC"
```

***** use apply instead of the very slow for-loop**

```
> add_something<-function(x) {paste(x,"_this_is_cool",sep="")}

> dna2<-as.matrix(dna2)
> dna2_neu<-apply(dna2,1,add_something)
> dna2_neu
[1] ">Acc12345|sequence 1_this_is_cool"
[2] "ATGCTGAAGTTCCAAACAGTTCGAGGGGGCCTGAGGCTCCTGGGTGTCCGCCGATCCTCC_this_is_cool"
[3] "TCGGCCCCTGTTGCCTCCCC_this_is_cool"
```

or

```
> dna3<-lapply(dna2,add_something)
>dna3
[[1]]
[1] ">Acc12345|sequence 1_this_is_cool"

[[2]]
[1] "ATGCTGAAGTTCCAAACAGTTCGAGGGGGCCTGAGGCTCCTGGGTGTCCGCCGATCCTCC_this_is_cool"

[[3]]
[1] "TCGGCCCCTGTTGCCTCCCC_this_is_cool"
```