TUTORIUM BIOINFORMATIK SS17

Was ist R?

- Programmiersprache für statistische Analysen
- Funktionelle Programmiersprache
- „Einfach“ und effektiv
- Bietet Funktionen zum Einlesen und Analysieren von Daten
- Erzeugen von Grafiken

Syntax – Variablen

- var <- 0 # Wert Zuweisung
- var <- c(0,2,4) # Vektor
- var <- “population“ # String zuweisen
- var <- list(1,2,3) # Liste
- mtrx <- matrix (c(1,2,3,4,5,6),ncol=2,nrow=3) # Matrix
- Anzeigen des Types:
  - class(var)
- Umwandeln mit:
  - as.vector(mtrx)

Daten

- Einlesen:
  - `read.table(file = “path/to/my/file.tsv“)`
    - Wird als data.frame gespeichert!
      -> Umwandeln in matrix mit: `as.matrix(read.table (file))`
    - Zusatzoptionen mit Komma trennen:
      - `quote = ““` # Unterdrücken von “ in Werten
- Schreiben:
  - `write.table(mtrx, file =“path/to/my/outFile.out“)`
  - Zusatzoptionen:
    - `sep = “\t“` # Tab als Trennsymbol verwenden
    - `row.names = F` # Keine Zeilennamen übernehmen
    - `col.names = F` # Keine Spaltennamen
    - `append = T` # Anhängen an existierende Datei

Vektoren

- Auf gleiche Länge prüfen und addieren
  - ```
    vec1 <- c(1,2,3)
    vec2 <- c(2,2,2)
    length(vec1) == length(vec2) #TRUE
    vec1 + vec2  #result: (3,4,5)
    ```
  - Andere Rechenoperationen ebenfalls möglich
- `vec1 %in% vec2`
  - Prüfen welche Indices von Vector 1 in Vector 2 sind
  - Länge egal
  - Ergebnis: `FALSE TRUE FALSE`
- Einzelne Werte Zuweisen/ Anzeigen
  - `vec1[1] = 0` # Zuweisung von 0 zu **erstem** Element in vec1
  - `vec1[length(vec1)]` # Letztes Element anzeigen
    ```

Matrix

- `matrix(x,nrow,ncol)`
  - x Vektor oder data.frame
  - nrow Anzal Zeilen
  - ncol Anzahl Spalten
- `m[,1] # erste Spalte`
- `m[1,] # erste Zeile`
- `is.matrix(m)`
  - Prüfen ob m vom Typ matrix ist
- `as.matrix(df)`
  - df (data.frame) in Matrix umwandeln
- `dim(m)`
  - Dimensionen
- `nrow(m) bzw. ncol(m)`
  - Zeilen- bzw. Spaltenanzahl
- `which(<Logischer Ausdruck>)`
  - Gibt Indices zurück die TRUE ergeben

```
> m <- matrix(c(1,1,3,4,5,6),ncol=3,nrow=2)
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    1    4    6
> dim(m)
[1] 2 3
> which(m == 1)
[1] 1 2
> which(m[,1] == 1)
[1] 1 2
> which(m[,2] == 1)
integer(0)
> which(m[1,] == 1)
[1] 1
> which(m == 1,arr.ind = T)
     row col
[1,]   1   1
[2,]   2   1
```

Listen

- Erzeugen: list()
  - Unterschiedliche Werttypen erlaubt
- In Vektor verwandeln:
  - vec <- unlist(liste)
  - In Kombination mit strsplt() hilfreich
    - content <- unlist(strsplt(str))

```
> n <- c(2,3,5)
> s <- c("aa","bb","cc")
> b <- c(TRUE,FALSE,T,F)
> b
[1]  TRUE FALSE  TRUE FALSE
> x <- list(n,s,b,3)
> x
[[1]]
[1] 2 3 5

[[2]]
[1] "aa" "bb" "cc"

[[3]]
[1]  TRUE FALSE  TRUE FALSE

[[4]]
[1] 3

> x[[2]][3]
[1] "cc"
> x[[1]]
[1] 2 3 5
```

If – else

```
If (<Bedingung>){
        <Anweisung>
}else if (<Bedingung>){
        <Anweisung>
}else{
        <Anweisung>
}
```

- Wenn <Anweisung> einzeilig ist dann können {} weggelassen werde

```
> x <- 1
> y <- 42
> if(x > y){
+     print(x)
+ }else if (x == y){
+     print(0)
+ }else{
+     print(y)
+ }
[1] 42
```

for - Schleife

- Syntax:

```
for( <Variable>  in <Vektor>){
      <Anweisung>
}
```

- Bsp.:

```
for( i in 1:10){
      print(i) # Ausgabe: 1 2 3 4 5 6 7 8 10
}

for( letter  in c(“a“, “b“, “c“)){
      print(letter)
}
```

Funktionen

- Syntax:

function_name <- function (par1,par2=<default>){
<Anweisungen>
return(<result>)
}

```
> printVector <- function(x,multiply=1){
+     print(x*multiply)
+ }
> vec <- c(1,2,3,4,5)
> printVector(vec,2)
[1]  2  4  6  8 10
```

Wo schreiben wir?

- IDE: **Rstudio**
  - Standardmäßig nicht am Cip Rechner installiert
  - Alternative: emacs (gewöhnungsbedürftig)
- In der **Konsole**
  - Den Buchstaben R in die Konsole tippen und Enter drücken
  - Alternative: R-3.3.2 (Versions Nummer mit angeben)
  - Interaktiver Modus wird gestartet:
    - **>** für Einzeiler (primäre Eingabeaufforderung)
    - **+** für Mehrzeiler (sekundäre Eingabeaufforderung)
    - q() zum Beenden
- Mit **Editor** ein Skript (<name>.R) schreiben und dann mit
  - `Rscript <name>.R` aufrufen

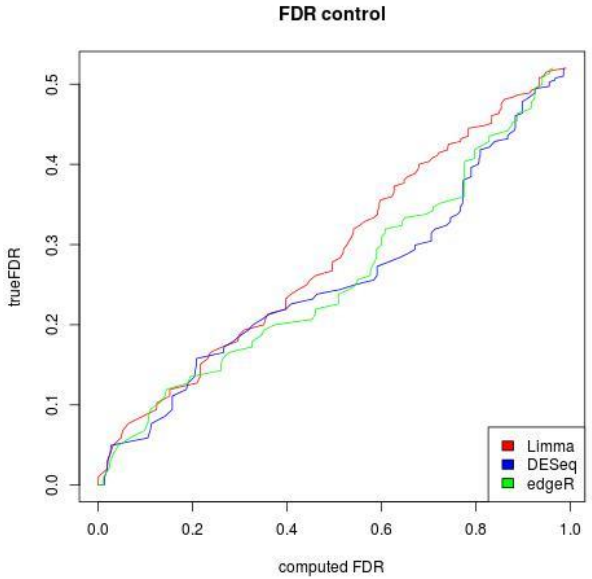
Speichern und Laden von Sessions

- `save.image("/path/to/dir/save.RData")`
  - Speichern aller erzeugten Variablen mit Werten in der Session
- `load("/path/to/dir/save.RData")`
  - Laden geschlossener Sessions

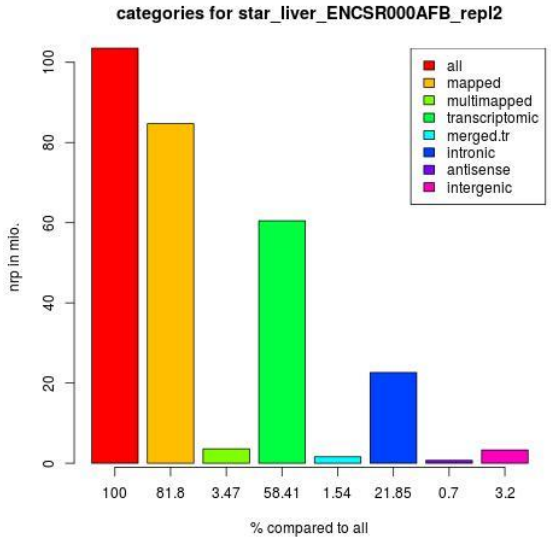
Grafiken

- Standard Grafiken: plot(…)
- Verbesserte Grafiken: ggplot2(…)
- Typen:
  - Line plots — plot(x,y) — # x: Vektor, y: Vektor, gleiche Länge
  - Barplot — barplot(x) — # x: Vektor oder Matrix oder data.frame
  - Boxplots — boxplot(x) — # x: Vektor oder Matrix
  - Scatterplots — plot(x,y,type="p") — # x: Vektor, y: Vektor, gleiche Länge
  - Heatmaps — heatmap(x) — # x: Matrix
  - ….

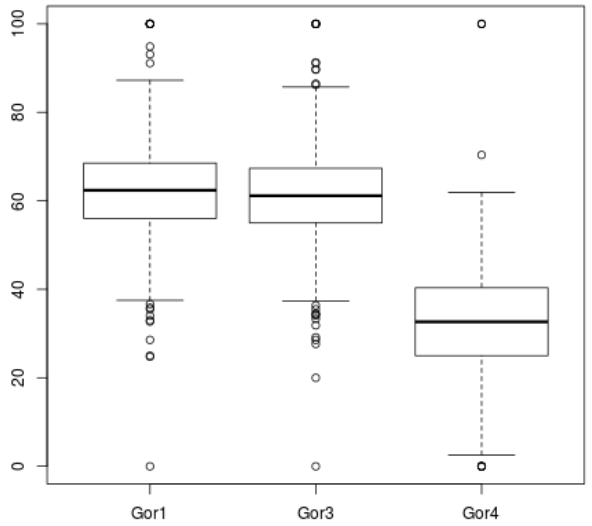
Beispiele



Line Plot

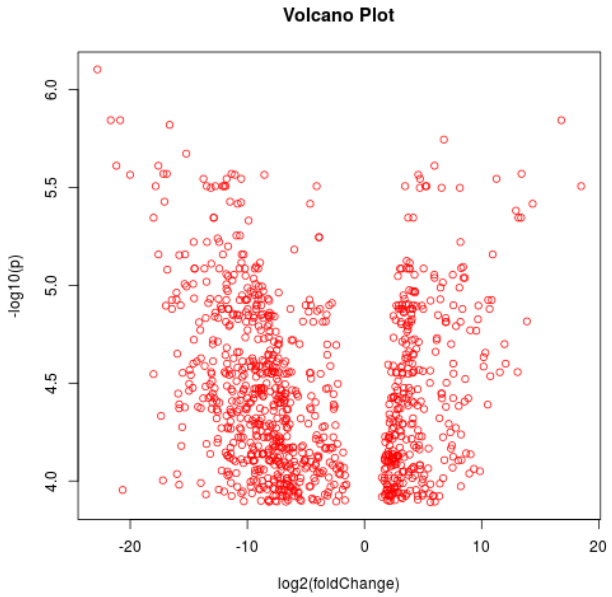


Barplot

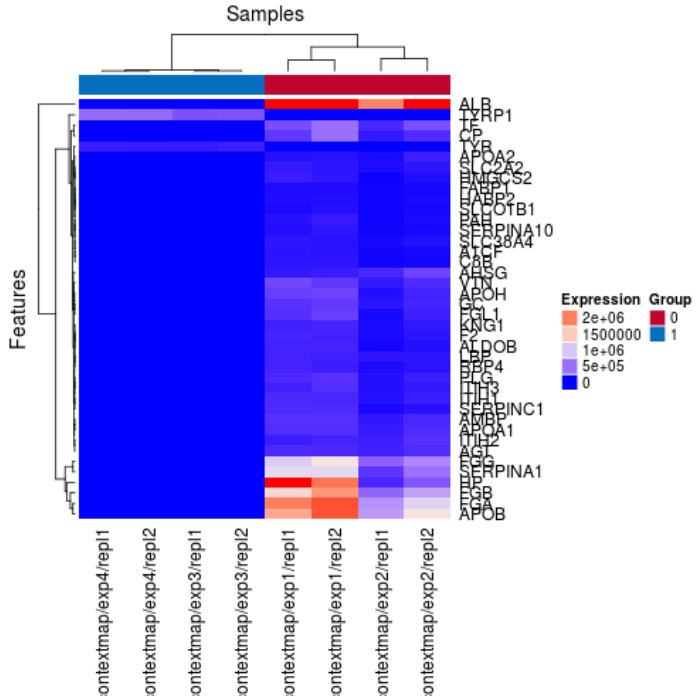


Boxplot

Beispiele



Scatterplot (hier: Vulcano Plot)



Heatmap

Nützliche Befehle

- `?<funktionsName>` # Anzeige von Hilfsseite
  - `?length()`
- `as.numeric(x)` # Umwandeln von String x der aus Zahlen besteht
- `as.vector(x)` # Umwandeln von x zu einem Vektor
- `paste0("hallo_","du")` # Kein Seperator dazwischen, Ausgabe: hallo_du
- `paste("hallo","du",sep="_")` # Seperator zwischen Strings, Ausgabe: hallo_du
- `strsplt(string,split="\t")` # Trennt string nach Split-Zeichen, Rückgabetyp: Liste
- `basename("/weg/zu/meinem/Ordner/text.txt")` # gibt text.txt zurück

Nützliche Befehle

- `install.packages("ggplot2")` # Installiert ggplot2
- `library(ggplot2)` # Laden von ggplot2
- `require(ggplot2)` # Lade, falls noch nicht geladen
- `png("/pfad/zu/grafik/testGrafik.png")` # "Öffnen" eines Bildes
  `<plot Befehl>` # Bild erzeugen
  `dev.off()` # Bild schreiben (speichern)
- `head(x)` # Anzeigen der ersten 5 Zeilen von x
- `tail(x)` # Anzeigen der letzten 5 Zeilen