JG|U UNIVERSITÄTSmedizin. MAINZ

Microarray-Auswertung im Zusammenspiel mit der Bium$^{(MZ)}$

Johanna Mazur & Timo Itzel

31. August 2012

Inhaltsverzeichnis

1 R und Bioconductor **4**

2 RStudio **5**

3 Erste Versuche mit R und RStudio **6**
3.1 Die RStudio-Oberfläche . . . 6
3.2 Rechnen mit R . . . 6
3.3 Installieren von Paketen . . . 7
3.4 Funktionen . . . 8
3.5 Hilfe in R und RStudio . . . 8
3.5.1 Hilfe zu Funktionen . . . 8
3.5.2 Hilfe zu Paketen . . . 9
3.6 Die Golub-Daten . . . 10
3.7 Histogramm . . . 11
3.8 Grafiken speichern . . . 12
3.9 Dateien exportieren . . . 13
3.10 Boxplot . . . 14
3.11 Der Editor in der RStudio-Oberfläche . . . 16

4 Datenauswertung von Affymetrix-Microarrays **17**
4.1 Konvertieren Excel-Datei nach CSV . . . 17
4.2 In RStudio einladen . . . 18
4.3 Verarbeiten der Rohdaten . . . 20
4.3.1 Einlesen der CEL-Dateien . . . 20
4.3.2 Qualitätskontrolle der Rohdaten . . . 21
4.3.3 Einlesen der Phenodaten . . . 22
4.4 Hintergrundkorrektur und Normalisierung . . . 24
4.5 Die ExpressionSet-Klasse . . . 25
4.6 Filtern nicht-informativer Gene . . . 27
4.7 Testen für differentiell exprimierte Gene . . . 28
4.8 Korrektur für multiples Testen . . . 28
4.9 Der Volcano-Plot . . . 29
4.10 Das limma-Paket . . . 30
4.11 GO-Terme und hypergeometrisches Testen . . . 32
4.12 Die Heatmap . . . 33

5 Die Core Facility Bioinformatik **35**
5.1 Kontakt . . . 35

1 R und Bioconductor

R ist eine für die Statistik und stochastische Simulation entwickelte Programmiersprache, die in diesem Bereich zum kostenfreien Standard geworden ist. Informationen zu R im Allgemeinen und seiner Installation befinden sich auf der Seite

`www.r-project.org`

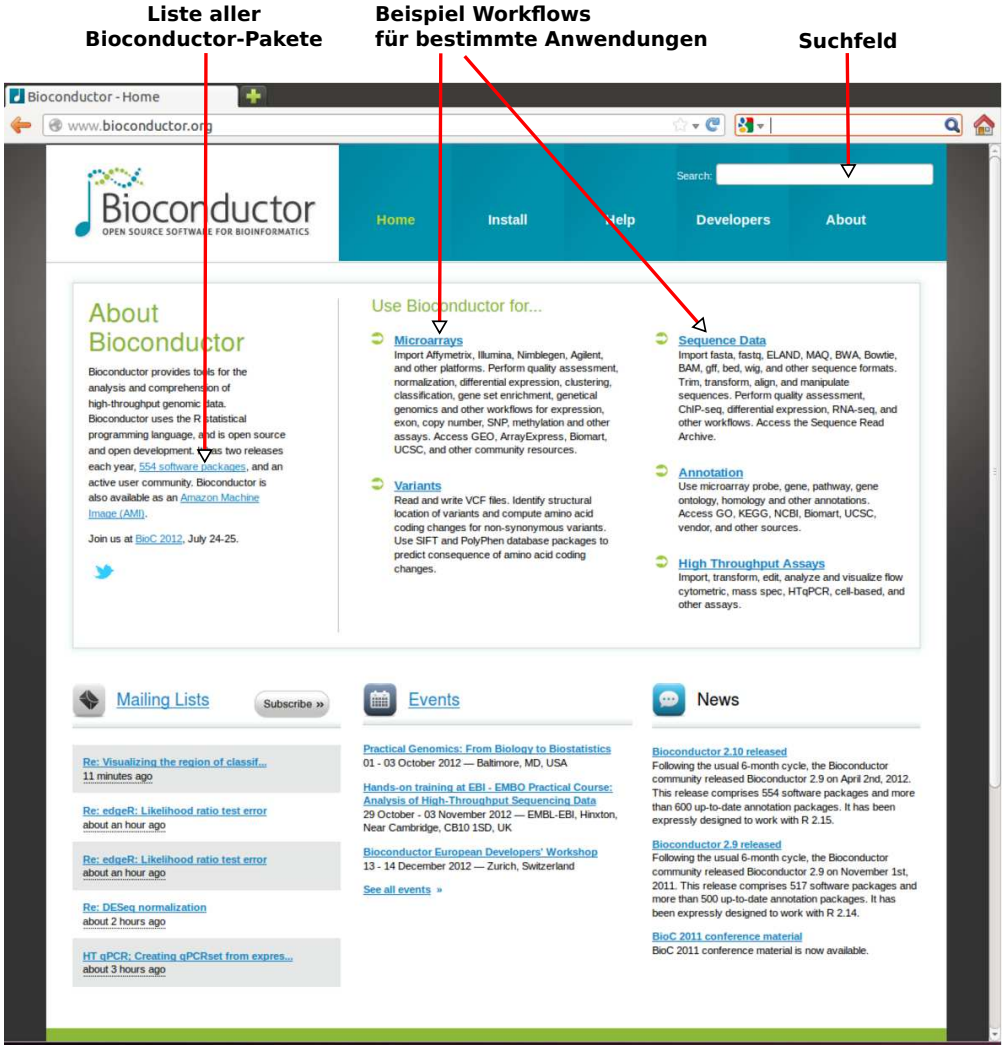
Dort befinden sich auch zahlreiche einführende Dokumentationen.

Bioconductor ist ein kostenfreies Softwareprojekt für bioinformatische Fragestellungen, welches Hilfsmittel für das Verständnis und die Analyse von molekularen Hochdurchsatzdaten bietet. Es basiert auf der Programmiersprache R. Auf der Seite

`www.bioconductor.org`

befinden sich zur Zeit 554 Softwarepakete für die Verarbeitung, Analyse und Visualisierung hochdimensionaler molekularer Daten. Hinzu kommen noch zahlreiche Annotationspakete und Pakete, die beispielhafte Daten enthalten.

Weiterhin befinden sich dort Beispiel-"Workflows", wie man zum Beispiel Microarraydaten oder Sequenzierungsdaten auswertet und wie hierfür verschiedene Bioconductor-Pakete benutzt werden können.



2 RStudio

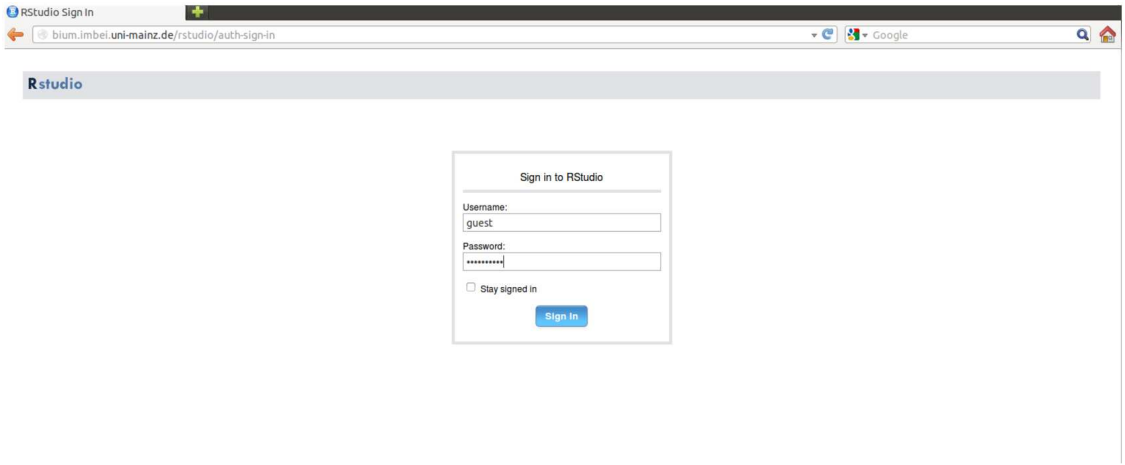
RStudio ist eine kostenfreie grafische Oberfläche für das Programmieren mit R. Details zu Download und Installation finden sich auf

`www.rstudio.org`

Im Bereich der Universität und Universitätsmedizin Mainz bietet die BIUM$^{(MZ)}$ die Möglichkeit, sich auf der Webseite

`bium.imbei.uni-mainz.de/rstudio`

eine RStudio-Sitzung zu öffnen und auf dem Rechencluster der Core Facility Bioinformatik selbstständig Datenanalysen durchzuführen. Für Hilfestellungen sowohl technischer als auch statistischer Art stehen die Mitarbeiter der BIUM$^{(MZ)}$ jederzeit zur Verfügung (siehe Abschnitt 5 für das Angebot der BIUM$^{(MZ)}$ und den Kontakt). Nach der Eingabe von `bium.imbei.uni-mainz.de/rstudio` in die Adresszeile von einem Webbrowser Ihrer Wahl erscheint folgende Seite

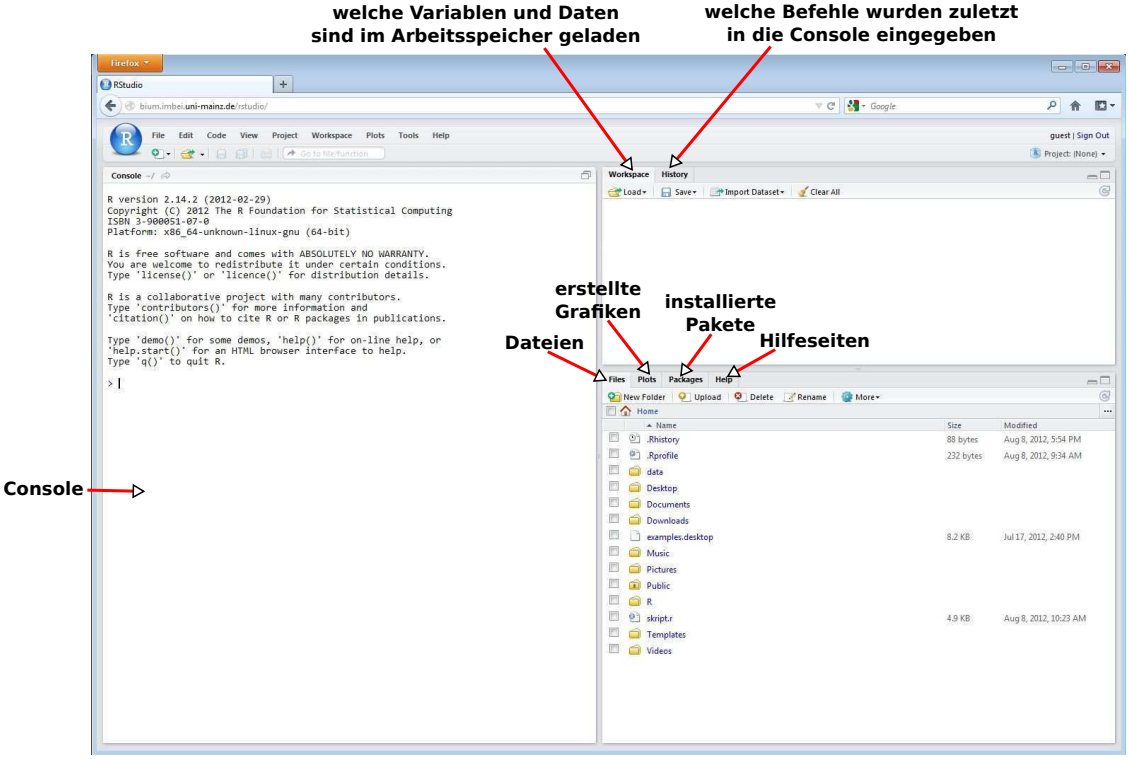


In das Feld `username` geben Sie Ihren Benutzernamen ein, dem Sie dem Zettel, den Sie am Anfang erhalten haben, entnehmen können. Und in das Feld `Password` geben Sie Ihr Passwort, welches auch auf dem Zettel zu finden ist, ein. Dann klicken Sie auf `Sign In`.

3 Erste Versuche mit R und RStudio

3.1 Die RStudio-Oberfläche

Nach Einloggen auf dem RStudio-Server, wie im Abschnitt 2 beschrieben, erscheint folgende Seite



Die RStudio-Oberfläche ist grob unterteilt in drei Bereiche: Auf der linken Seite befindet sich die *Console*. Dort werden die Kommandos eingetippt. Auf der rechten Seite oben kann man sehen welche Variablen und Daten im Arbeitsspeicher (`Workspace`) geladen sind. In unserem Fall natürlich noch nichts. Der zweite Tab im rechten oberen Feld ist das `History`-Feld. Dort kann man die zuletzt in der Console eingegebenen Befehle sehen.

Auf der rechten unteren Seite gibt es 4 Tabs. Das erste Tab ist das `Files`-Tab, in dem man die Dateien in Ihrem Benutzerverzeichnis sehen kann. In Ihrem Fall ist das ein Ordner mit dem Namen `data` und eine Datei mit dem Namen `skript.r`. Das zweite Tab ist das `Plots`-Tab, in dem man die Grafiken sieht, die man in der Console programmiert hat. Das dritte Tab ist das `Packages`-Tab, in dem man die R- und Bioconductor-Pakete, die installiert sind, aufgelistet bekommt. Das vierte Tab ist das `Help`-Tab. Dort befinden sich die Hilfeseiten.

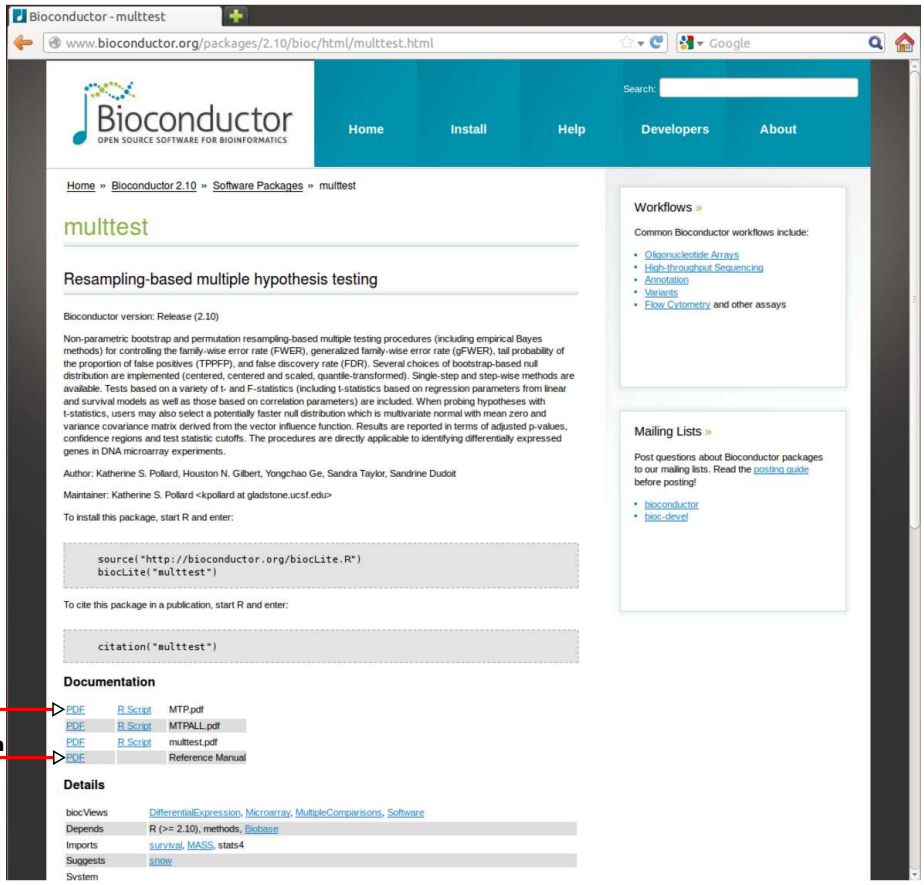
3.2 Rechnen mit R

Die Console in RStudio kann benutzt werden wie ein Taschenrechner. Für die vier Grundrechenarten benutzt man die Symbole + für die Addition, − für die Subtraktion, ∗ für die Multiplikation und / für die Division. Zum Potenzieren verwendet man das Symbol ˆ. Als Übung geben wir mal in die Console ein

```
> 2+3
> 2-3
> 2*3
> 2/3
> 2^3
```

3.3 Installieren von Paketen

Exemplarisch schauen wir uns mal erstmal die Webseite in Bioconductor für das Paket `multtest` an. Dieses Paket enthält Korrekturmethoden für multiples Testen und auch den *Golub*-Datensatz, den wir später benutzen möchten.



Auf der Seite findet sich eine kurze Information über die Funktionalität des Paketes und weiter unten unter "Documentation" findet sich üblicherweise mindestens eine sogenannte *Vignette* als pdf-Datei mit zugehörigem R-Code, die detailliert an einem Beispiel die Benutzung des Paketes erklärt und wie die darin enthaltenen Funktionen benutzt werden können. Das dort auch befindliche *Reference Manual* erklärt die enthaltenen Funktionen im Detail, so wie sie auch auf den Hilfeseiten von RStudio zu finden sind (siehe Abschnitt 3.5).

Zum Installieren eines beliebigen Bioconductor-Paketes geben Sie zunächst in die Console von RStudio ein

```
> source("http://www.bioconductor.org/biocLite.R")
```

Um nun das Paket `multtest` zu installieren, geben Sie in die Console von RStudio ein

```
> biocLite("multtest")
```

Wenn das Paket fertig installiert ist, muss das Paket noch geladen werden, damit seine Funktionalitäten zur Verfügung stehen. Dies geschieht mit der Eingabe in die Console von

```
> library(multtest)
```

Zusammenfassung: Installieren und Laden von Bioconductor-Paketen

```
> source("http://www.bioconductor.org/biocLite.R")
> biocLite("PAKETNAME")
> library(PAKETNAME)
```

3.4 Funktionen

Wir möchten nun einer *Variablen* einen Wert zuweisen, damit wir mit dieser Variablen arbeiten können anstatt mit den Werten selbst. Dies geschieht mit dem Symbol <– und wir weisen mal der Variablen `a` den Vector $(1, 2, 3, 4, 5)$ zu

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

Auf der Variablen a können wir nun verschiedene Funktionen wie zum Beispiel die Summe oder das Produkt der Elemente ausführen. Dies geschieht durch die Eingabe in die Console von

```
> sum(a)
> prod(a)
```

Der Befehl `sum(a)` bezeichnet dabei die *Funktion* `sum()`, die die Summe der Elemente des Vectors `a` berechnet. Im Allgemeinen sind Funktionen in R dadurch gekennzeichnet, dass sie keine, eine oder mehrere Parameter (durch Kommata getrennt) mit runden Klammern einschließen. Die Befehle, die in Abschnitt 3.3 benutzt wurden, sind somit auch R-Funktionen.

Nützliche Funktionen

```
> c(1,2,3)          # erzeugt einen Vektor (hier mit den Werten 1, 2 und 3)
> ls()              # listet alle Variablen, die sich im Moment im Arbeitsspeicher
                      befinden
> factor()          # erzeugt einen Faktor aus einem Vektor
```

Als Übung erzeugen wir einen Faktor `bunt` mit den Leveln *rot*, *grün* und *blau*. Dazu geben wir in die Console ein

```
> bunt <- factor("rot","grün","blau","blau")
> bunt
```

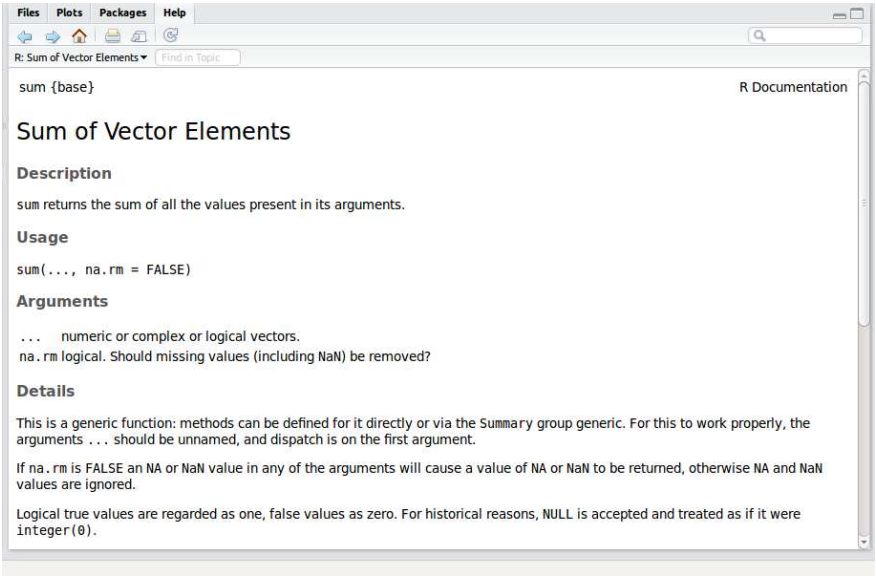
3.5 Hilfe in R und RStudio

3.5.1 Hilfe zu Funktionen

Zu jeder Funktion in R gibt es eine Hilfeseite, die die Benutzung und Funktionsweise der Funktion erläutert. In der Console wird diese Hilfeseite aufgerufen, wenn man dem Namen der Funktion das Symbol ? voranstellt. Schauen wir uns nun die Hilfeseite der Funktion `sum()` an. Dazu geben wir in die Console ein

```
> ?sum
```

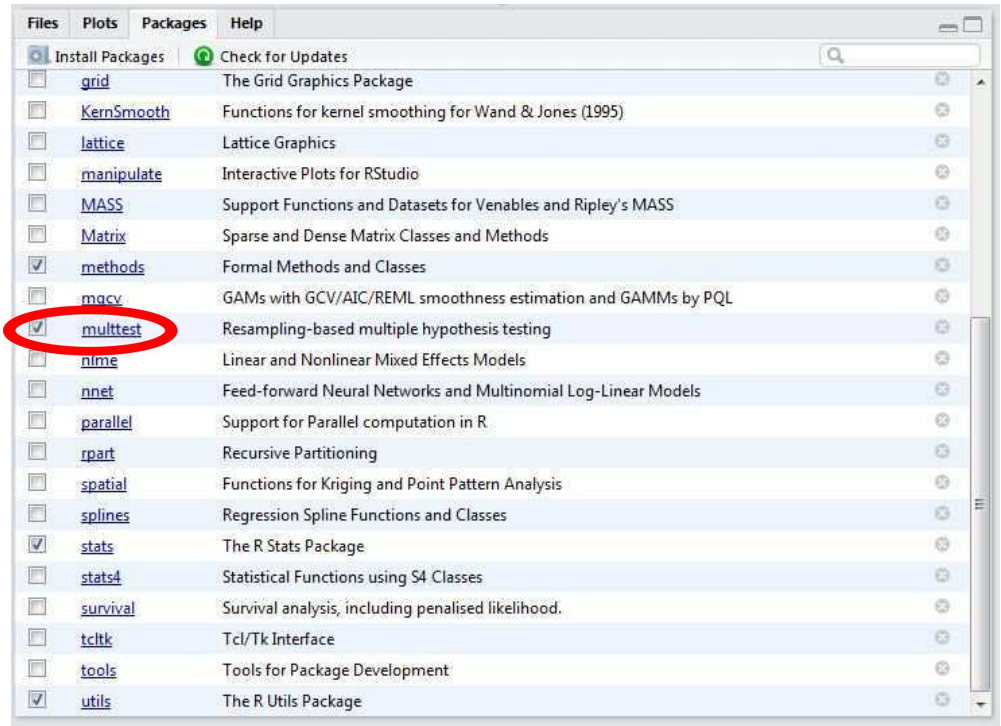
Dann öffnet sich im unteren rechten Fenster in der RStudio-Oberfläche die Hilfeseite der Funktion `sum()`. Diese Seite sieht so aus:



Dasselbe Ergebnis erziehlt man, wenn man in dem `Help`-Tab des unteren rechten Fensters der RStudio-Oberfläche in das Feld mit der Lupe `sum()` einträgt. Es gibt somit zwei Möglichkeiten in RStudio auf Hilfeseiten von Funktionen zu kommen.

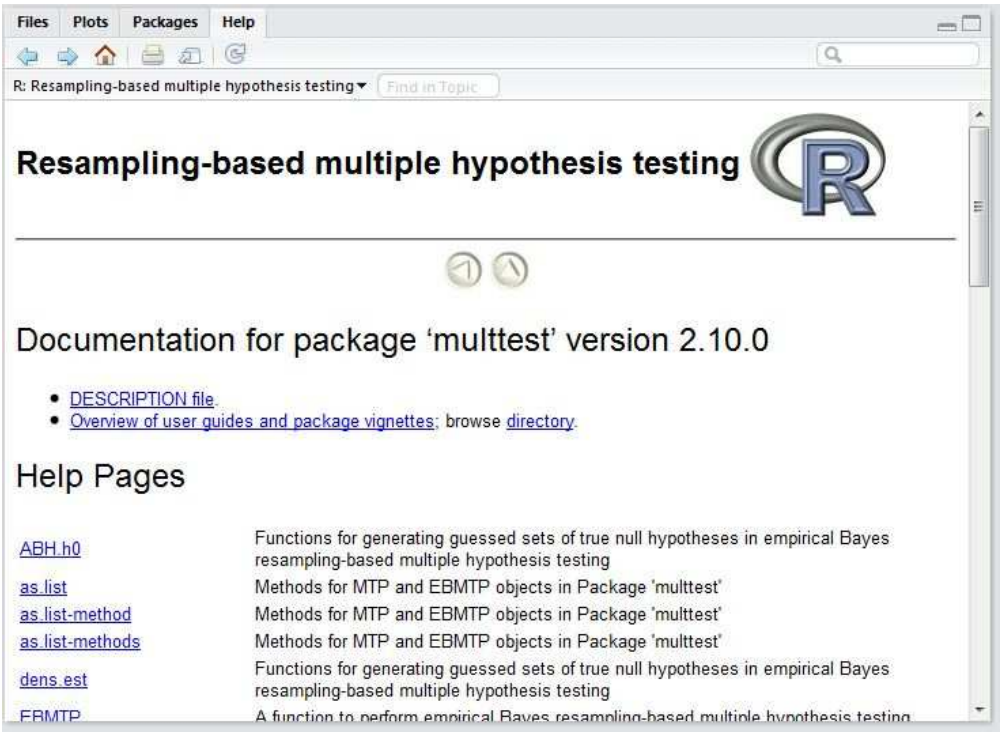
3.5.2 Hilfe zu Paketen

Wenden wir uns nun dazu, Hilfe zu einzelnen Paketen zu bekommen. Dazu klicken wir zunächst auf den `Packages`-Tab und erhalten im unteren rechten Fenster in der RStudio-Oberfläche das folgende Ergebnis



Wir sehen links neben dem Paket `multtest` ein Kästchen mit einem Häkchen. Dieses Häkchen bedeutet, dass das Paket im Arbeitsspeicher geladen ist. Dies haben wir vorhin in der Console gemacht. Es besteht somit hier auch die Möglichkeit, bereits installierte Pakete durch Klicken links neben den Paketnamen in den Arbeitsspeicher zu laden.

Durch Klicken auf `multtest` kommen wir nun auf die Hilfeseiten des Paketes. Diese Seite sieht nun so aus



Dort findet man Links zu den Hilfeseiten aller in dem Paket enthaltenen Funktionen und Daten. Auch können dort die Vignetten des Paketes angeschaut und heruntergeladen werden.

3.6 Die Golub-Daten

Nach einer kurzen technischen Einführung in R und RStudio wenden wir uns nun ersten echten Daten, den Golub-Daten zu. Diese Daten sind Genexpressiondaten mit 3051 Genen und 38 Tumorstichproben aus dem Jahr 1999 von Golub *et al.* [1]. Die Golub-Daten sind in dem Paket `multtest` vorhanden, welches wir vorhin schon installiert und geladen haben. Zum Laden und weiteren Betrachten der Daten geben wir in die Console ein

```
> data(golub)
> ls()
> ?golub
> golub.cl
> golub.gnames
> dim(golub)
```

Um mit den Daten weiterzuarbeiten erzeugen wir erstmal einen Faktor für die 38 Stichproben mit den Labels ALL und AML

```
> gol.fac <- factor(golub.cl,labels=c("ALL","AML"))
```

Weiterhin möchten wir uns die Expressionswerte für das Gen CCND3 genauer anschauen. Zunächst müssen wir hierfür in der Variable `golub.gnames` den Index herausfinden, welche Zeile der Genexpressionsdaten `golub` die Expression des Gens CCND3 beschreibt. Dies geschieht mit der Funktion `grep()`. Wir geben in die Console ein

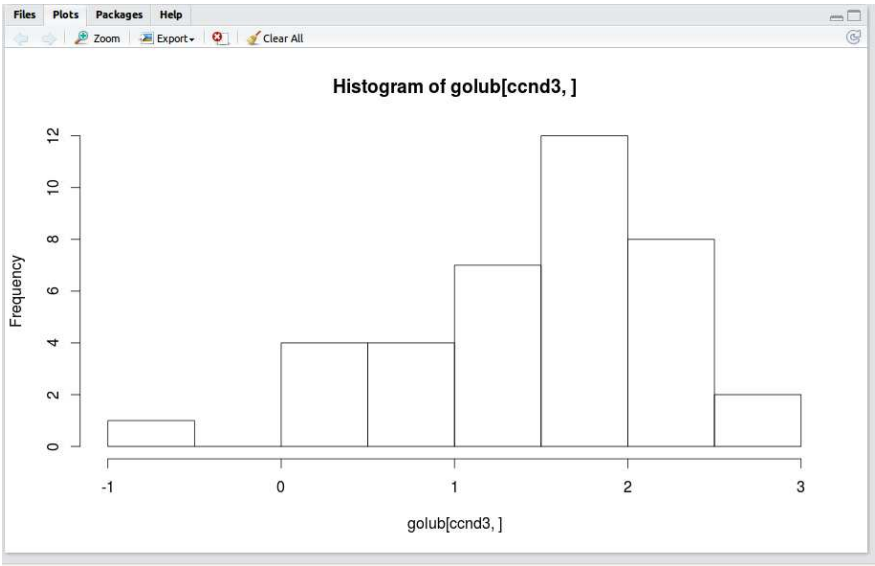
```
> ccnd3 <- grep("CCND3",golub.gnames[,2])
```

3.7 Histogramm

Wir möchten uns nun ein Histogramm der Expressionswerte des Gens CCND3 anschauen. Dies funktioniert mit der Funktion `hist`. Zunächst schauen wir uns das Histogramm für alle 38 Stichproben an. Dazu geben wir in die Console ein

```
> hist(golub[ccnd3,])
```

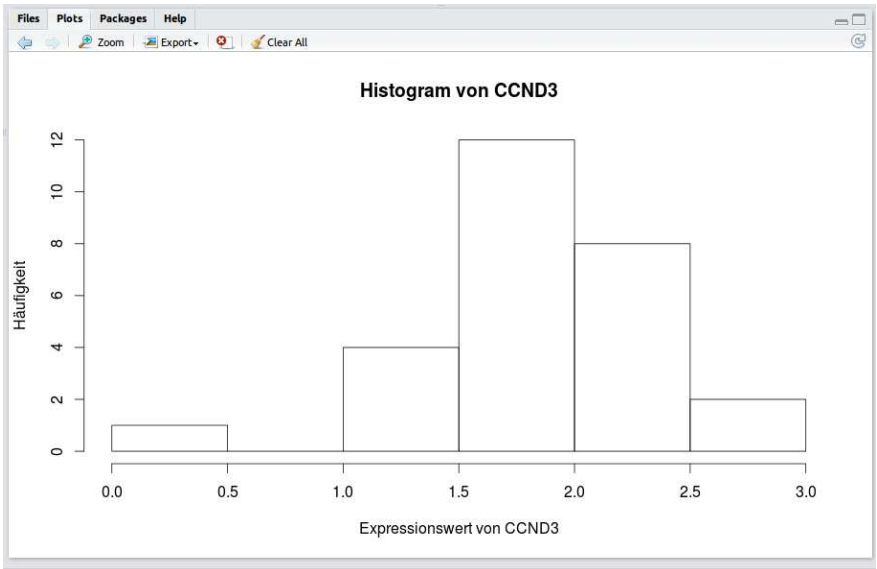
Dann wird im unteren rechten Fenster in der RStudio-Oberfläche die zugehörige Grafik angezeigt in dem `Plots`-Tab. Dieses Fenster sieht dann so aus:



Als nächstes möchten wir uns nur noch das Histogramm für die Stichproben mit den Leveln ALL anschauen und noch zusätzlich die Achsenbeschriftungen ändern. Dazu geben wir in die Console ein

```
> hist(golub[ccnd3,gol.fac=="ALL"])
> hist(golub[ccnd3,gol.fac=="ALL"],main="Histogramm von CCND3",
       xlab="Expressionswert von CCND3",ylab="Häufigkeit")
```

Nach Eingabe des letzten Befehls sieht man im unteren rechten Fenster in der RStudio-Oberfläche die Grafik

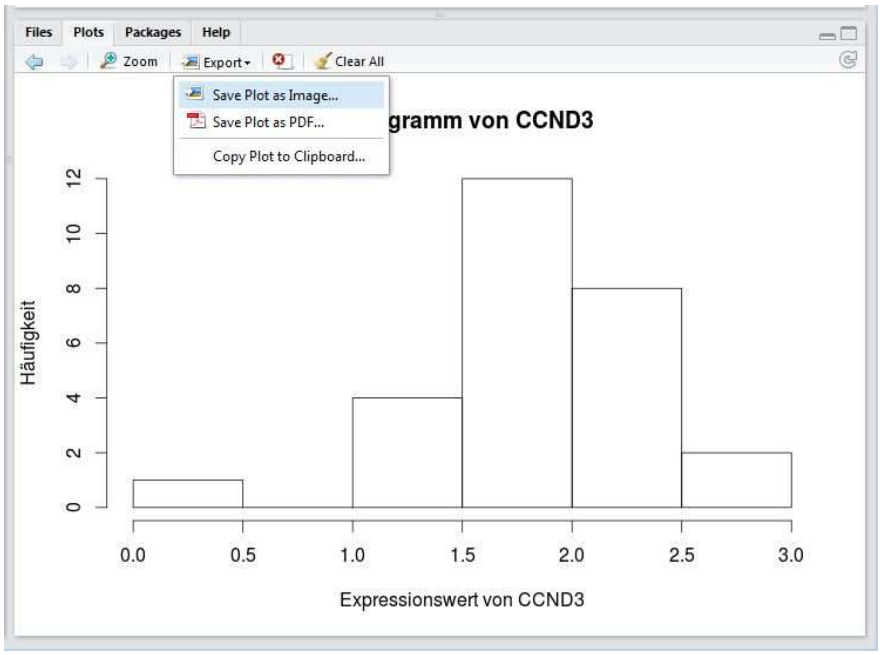


Parameter zum Ändern der Beschriftung innerhalb einer Grafik

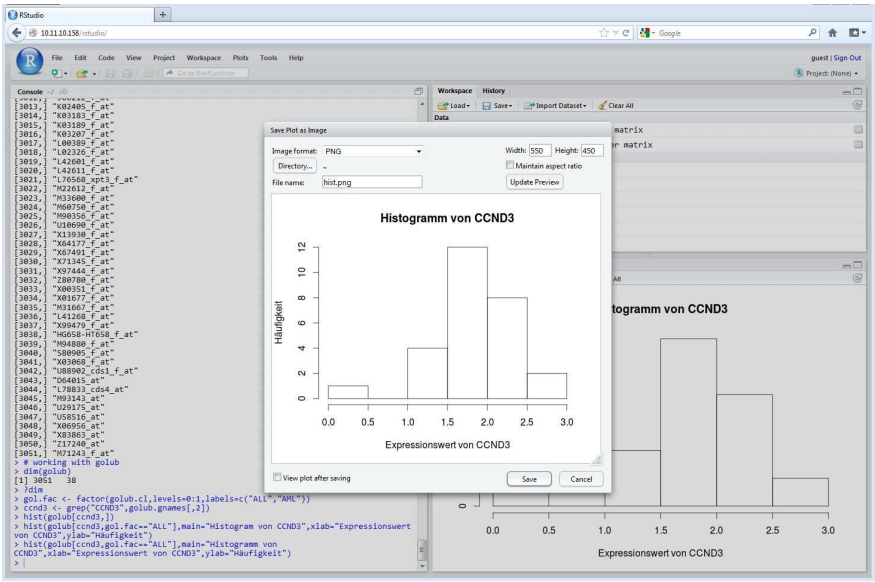
- `main` – Überschrift der Grafik
- `xlab` – x-Achsenbeschriftung
- `ylab` – y-Achsenbeschriftung

3.8 Grafiken speichern

Diese Grafik möchten wir nun zunächst speichern als `hist.png`. Hierzu klicken wir auf die `Export`-Schaltfläche. Dann erscheint dieses Bild im unteren rechten Fenster in der RStudio-Oberfläche



Durch klicken auf `Save Plot as Image...` öffnet sich ein Fenster. Die RStudio Oberfläche sieht nun folgendermaßen aus



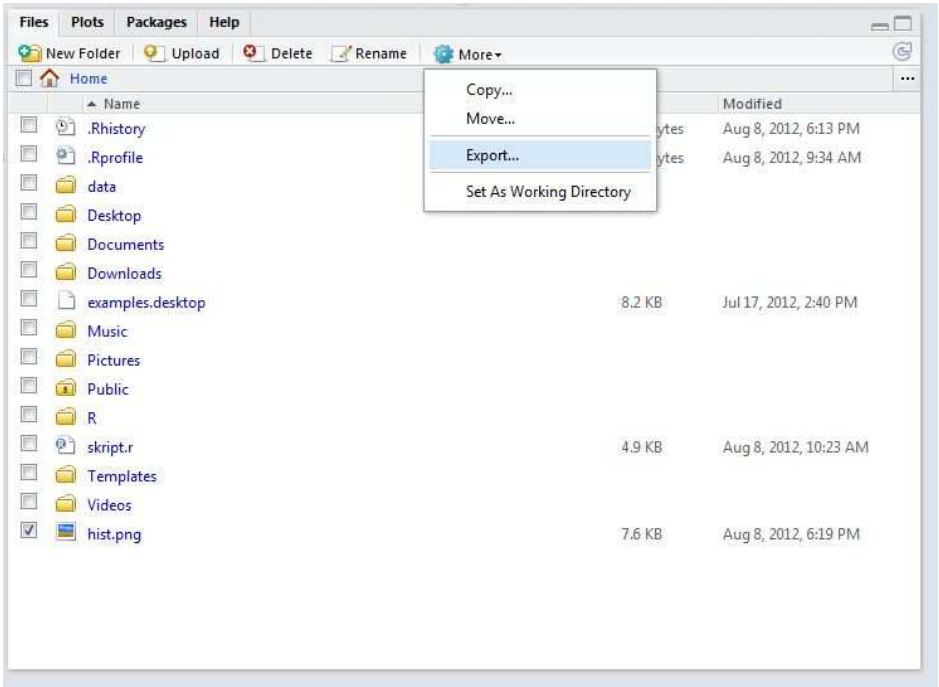
Die Grafik kann dabei als `PNG`-, `JPEG`-, `TIFF`-, `BMP`-, `SVG`- oder `Postscript`-Datei gespeichert werden. Wir entscheiden uns für `PNG` im Feld `Image format:` und benennen die Datei noch `hist.png`, was wir in das Feld `File name:` schreiben. Nach Klicken auf `Save` finden wir die gespeicherte Datei in unserem Arbeitsordner, welches wir uns anschauen mit Klicken auf den `Files`-Tab.

Zusammenfassung: Speichern einer Grafik

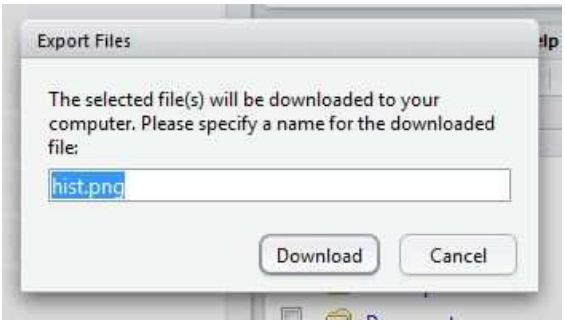
1. Klick auf `Export` im "Plots"-Tab
2. Klick auf `Save Plot as Image...`
3. `Image Format` angeben (Wahl zwischen `PNG`, `JPEG`, `TIFF`, `BMP`, `SVG`, `Postscript`)
4. `File Name` angeben
5. Klick auf `Save`

3.9 Dateien exportieren

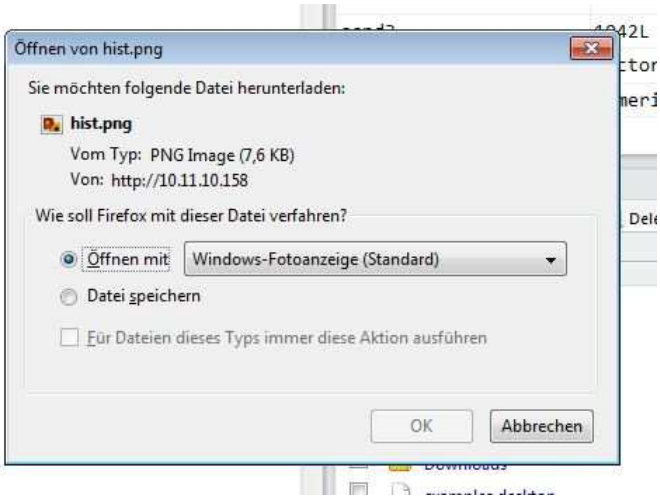
Als nächstes möchten wir diese Grafik herunterladen auf unseren lokalen Rechner. Hier klicken wir zuerst das Kästchen an, welches sich links neben der Datei `hist.png` befindet. Durch weiteres Klicken auf `More` in dem unteren rechten Fenster der RStudio-Oberfläche sehen wir dies



Durch Klicken auf `Export` öffnet sich das folgende Fenster



Durch weiteres Klicken auf `Download` öffnet sich das folgende übliche Fenster unter Windows



Zusammenfassung: Exportieren einer Datei

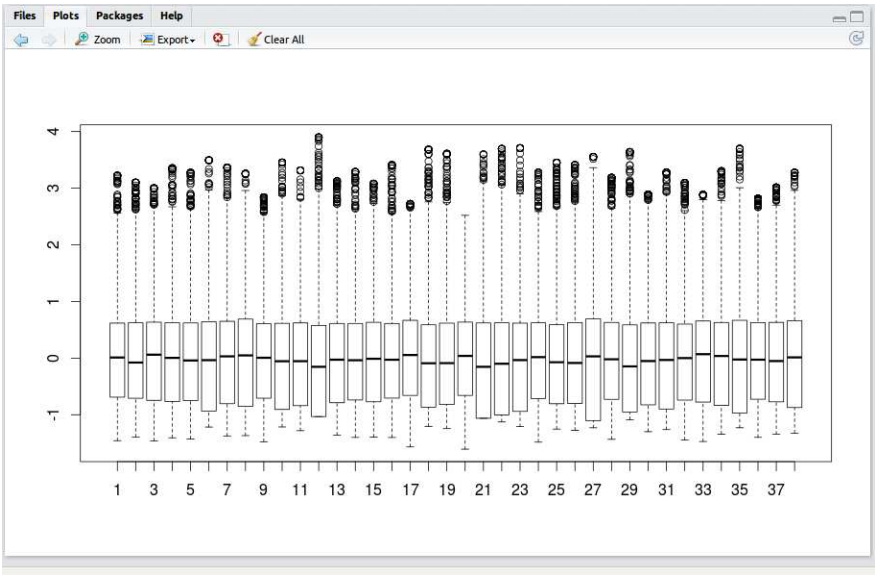
1. Im "Files"-Tab links in das Kästchen der zu exportierenden Datei klicken
2. Klick auf `More` im "Files"-Tab
3. Klick auf `Export...`
4. Namen spezifizieren, unter dem man die Datei speichern möchte
5. Klick auf `Download`

3.10 Boxplot

Nun wollen wir als weitere grafische Darstellungsform einen Boxplot der *Golub*-Daten erstellen. Zuerst wollen wir uns die Boxplots aller Stichproben über alle Gene anschauen. Dazu geben wir in die Console ein

```
> boxplot(golub)
```

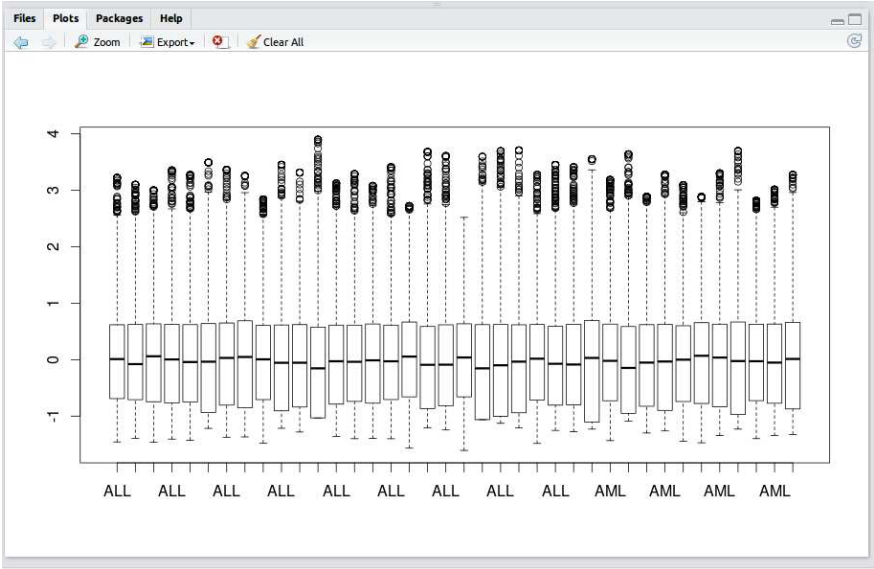
Im unteren rechten Fenster in der RStudio-Oberfläche erscheint dann die Grafik



Anstatt der Nummer der Stichproben als Beschriftung der Boxplots möchten wir nun die Label ALL und AML stehen haben. Dazu geben wir in die Console ein

```
> boxplot(golub,names=as.character(gol.fac))
```

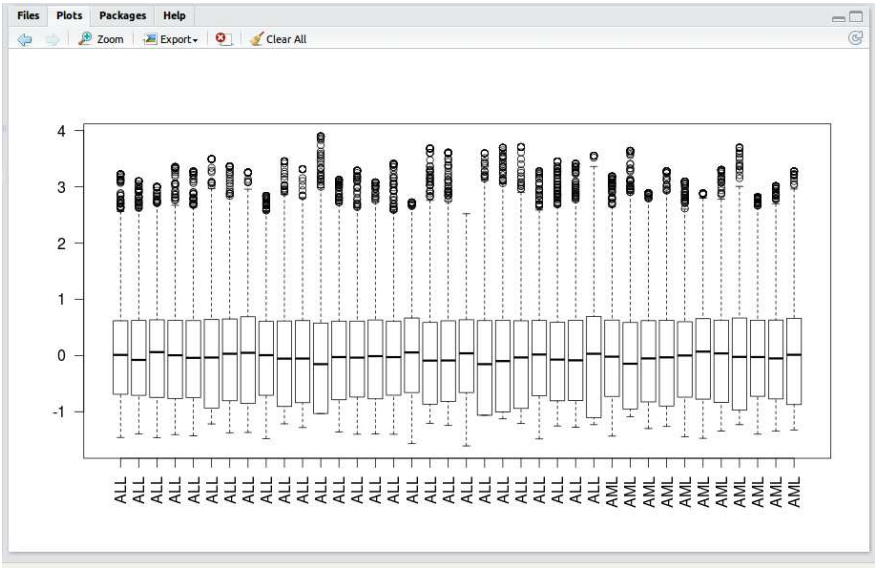
Im unteren rechten Fenster in der RStudio-Oberfläche erscheint dann die Grafik



Da nicht unter allen Boxplots eine Beschriftung steht, benutzen wir den grafischen Parameter `las` innerhalb der Boxplot-Funktion, um die Beschriftungen zu drehen, was nach der Eingabe in die Console von

```
> boxplot(golub,names=as.character(gol.fac),las=2)
```

zur folgenden Grafik führt



Als letztes abschließendes Beispiel geben wir noch eine Demonstration der Nützlichkeit von Faktoren. Mit der Eingabe in die Console von

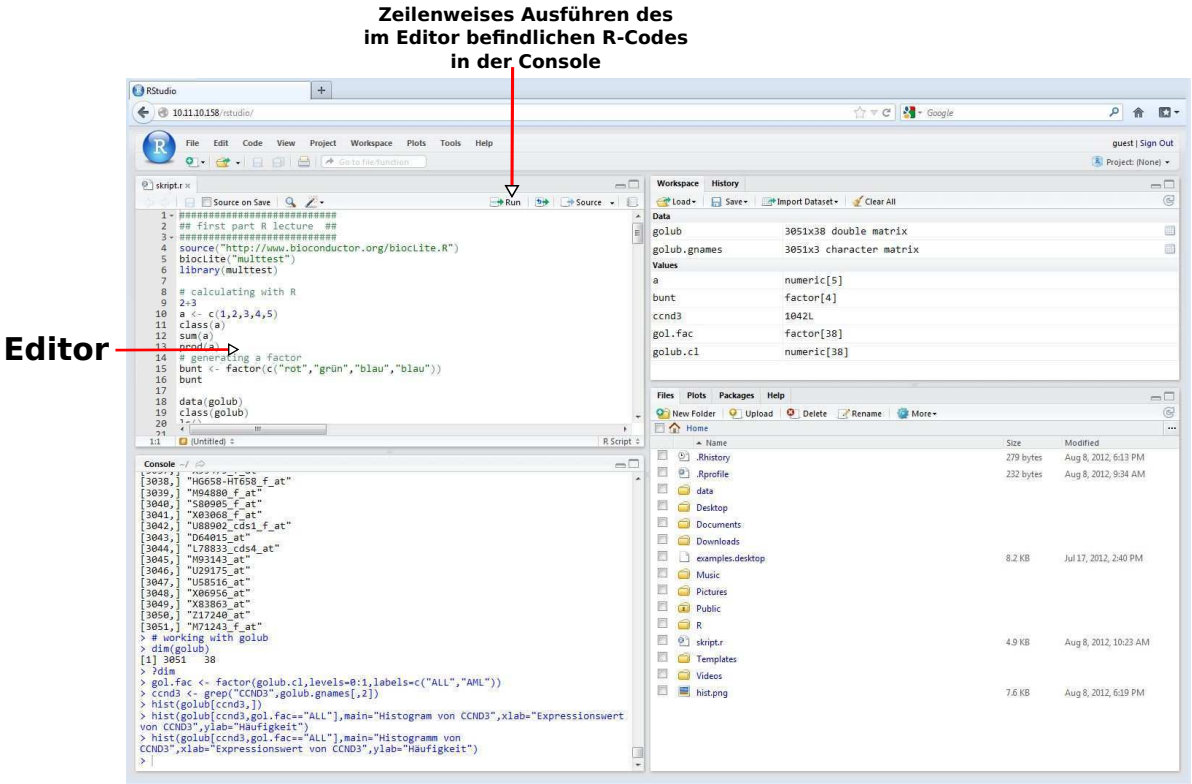
```
> boxplot(golub[ccnd3,] ~ gol.fac)
```

erhalten wir die Grafik

welche uns die beiden Boxplots der Expressionswerte des Gens CCND3 für die beiden Tumortypen ALL und AML zeigt.

3.11 Der Editor in der RStudio-Oberfläche

Wir klicken nun im `Files`-Tab auf die Datei `skript.r`. Dann öffnet sich ein viertes Fenster in der oberen linken Ecke der RStudio-Oberfläche, in dem die Datei `skript.r` dann zu sehen ist



Dieses Fenster ist der *Editor*. Die R-Kommandos, die nun im Editor in der Datei `skript.r` zu sehen sind, können zeilenweise in der Console ausgeführt werden, indem man auf `Run` klickt in der Taskleiste des Editorfensters. Es wird dabei die Zeile der Datei ausgeführt, in der sich der Cursor befindet.

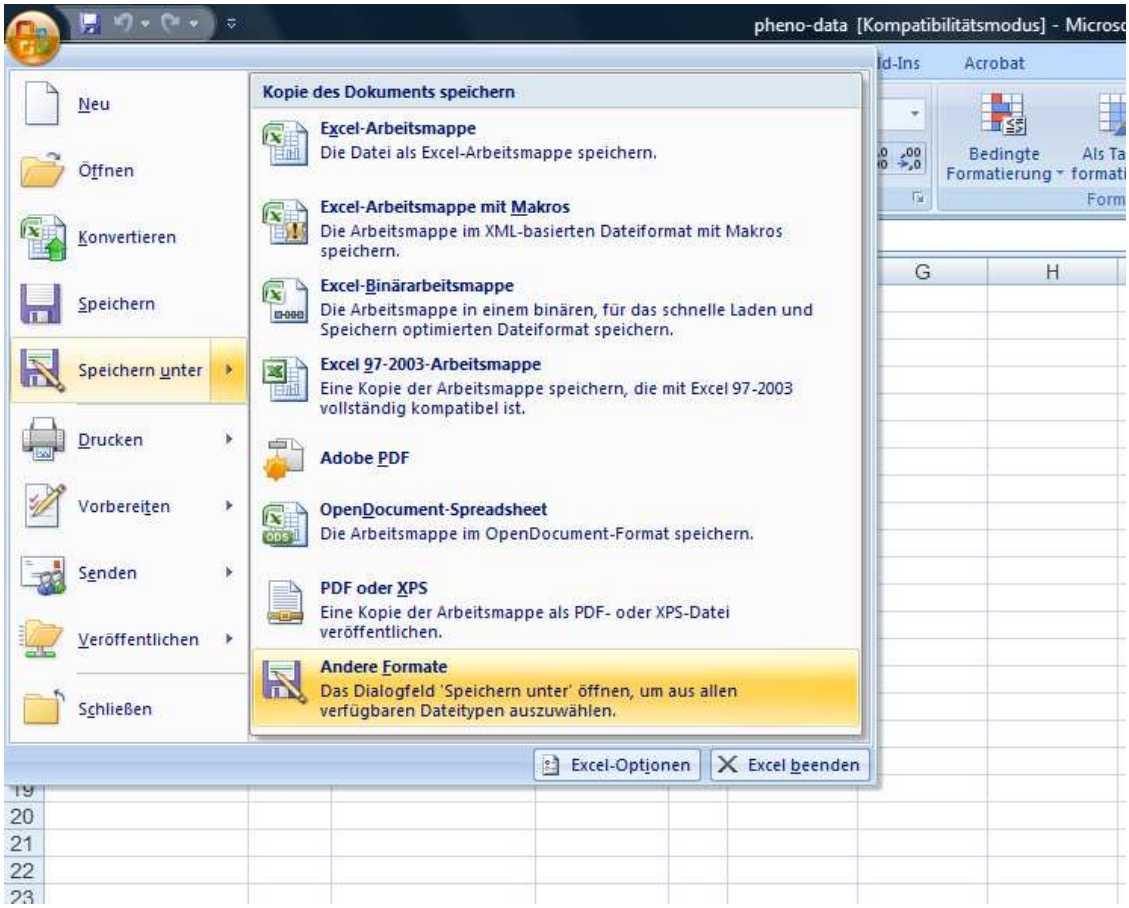
4 Datenauswertung von Affymetrix-Microarrays

Zur exemplarischen Analyse von Affymetrix-Microarrays benutzen wir einen Datensatz, der unter dem Namen 'Full-term placenta, smokers and non-smokers' bei Gene Expression Omnibus unter dem Label GSE7434[2] frei verfügbar ist. Dieser Datensatz besteht aus 10 Microarrays und enthält biologisches Material aus der Plazenta von rauchenden und nichtrauchenden Frauen. Weiterhin ist für den Datensatz das Geschlecht des Kindes bekannt.

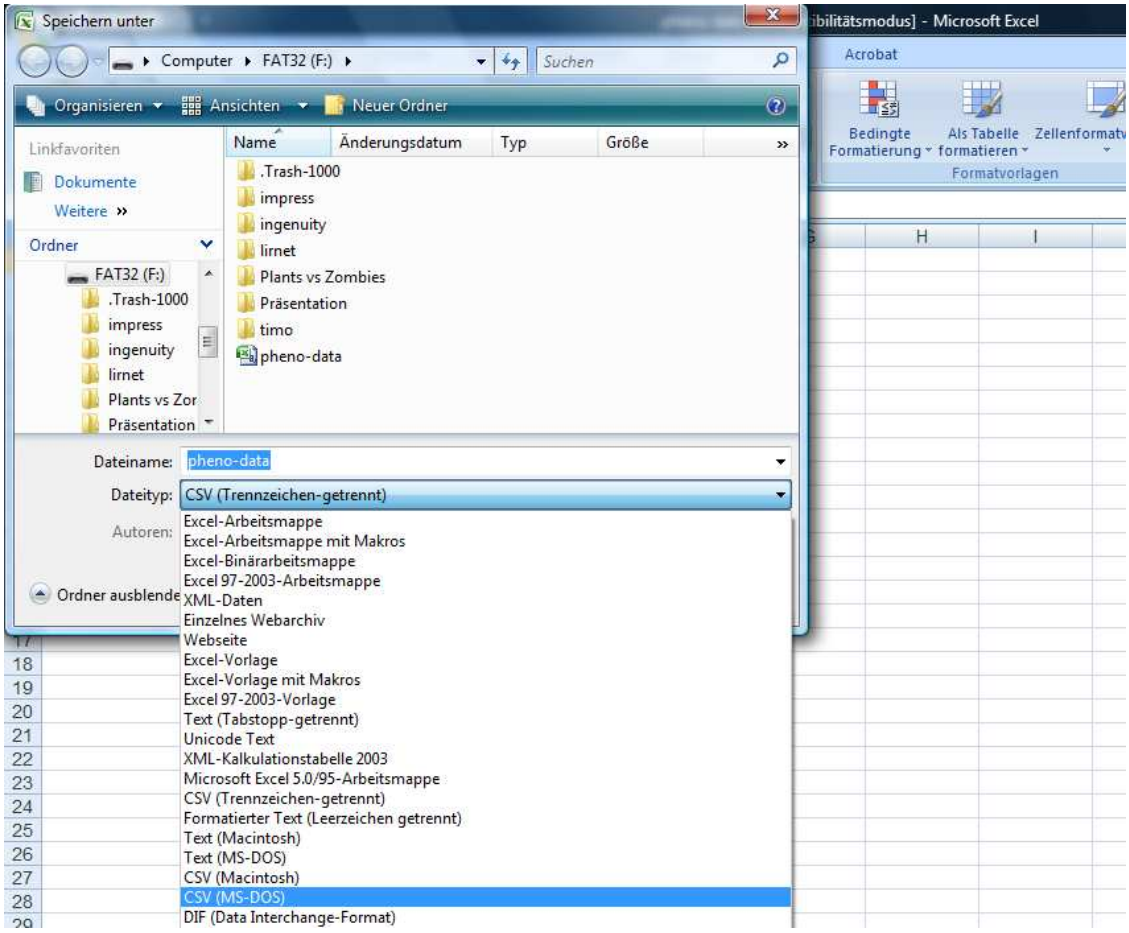
Die dazu gehörigen CEL-Dateien haben wir vorab herunter geladen und in Ihrer Arbeitsumgebung unter *path/do/directory* für Sie verfügbar gemacht. Auch haben wir die experiment-spezifischen Daten vorab in einer Excel-Tabelle für Sie zusammengestellt.

4.1 Konvertieren Excel-Datei nach CSV

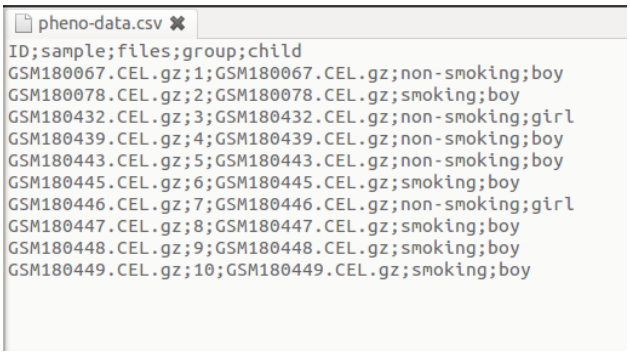
Zum Konvertieren der Excel-Datei öffnen wir diese mit Excel. Als nächstes öffnen wir den Dialog `Speichern untern` oder `Save As`.



Als nächstes wählen wir die Dateiart: CSV. CSV-Dateien sind ursprünglich "Komma separierte Dateien" zur Darstellung von Tabellen, die zeilenweise die einzelnen Zellen durch Kommatas abtrennt. Diese Trennzeichen kann man beliebig setzen, allerdings dürfen diese nicht in Zellentexten vorkommen.

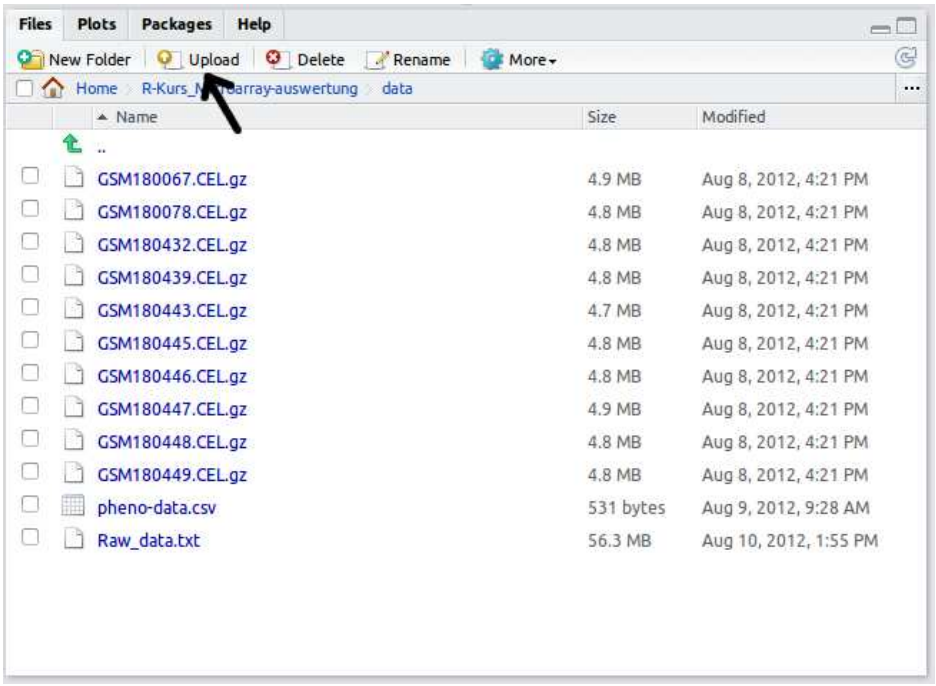


Nun erhalten wir eine Semikolon separierte Datei.



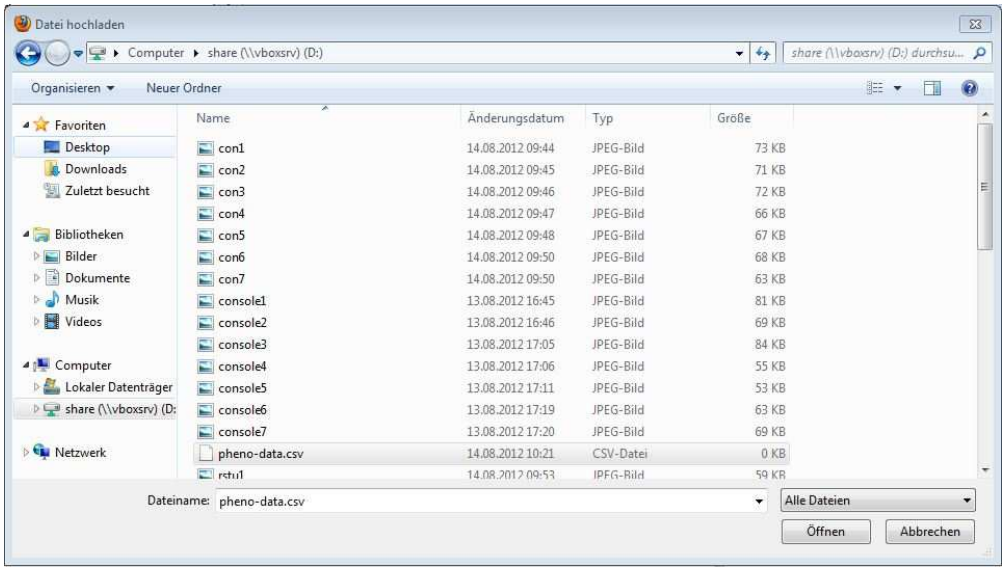
4.2 In RStudio einladen

In RStudio laden wir die Datei `pheno-data.csv` über den `Upload`-Knopf auf den Server.

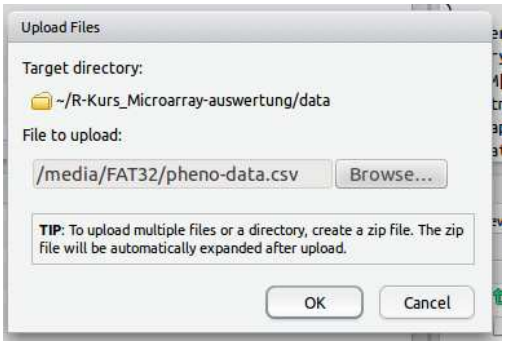


Es erscheint ein Fenster zur Auswahl der Datei. Hier drücken wir auf den `Browse`-Knopf.

Über den Browse-Dialog können wir nun die Datei auswählen.



Mit der OK-Taste bestätigen wir den upload.



Die Datei befindet sich nun auf dem Server in dem aktuellen Verzeichnis.

4.3 Verarbeiten der Rohdaten

Das Einlesen der Daten erstreckt sich in unserem Fall auf 2 Schritte. Als erstes werden wir die CEL-Dateien einladen. Der 2. Schritt beinhaltet das Einladen der Phenodaten über die Import-Funktion von RStudio.

Mit den Funktionen aus dem Paket `affy` [3] werden wir die Daten in der Arbeitsumgebung zur Verfügung stellen, diese Verarbeiten und die Qualitätskontrollen erstellen. Hierfür installieren wir das Paket `affy` und laden es.

```
> source("http://bioconductor.org/biocLite.R")
> biocLite("affy")
> library(affy)
Loading required package: Biobase

Welcome to Bioconductor

  Vignettes contain introductory material. To view, type
  'browseVignettes()'. To cite Bioconductor, see
  'citation("Biobase")' and for packages 'citation("pkgname")'.
```

4.3.1 Einlesen der CEL-Dateien

Zum Einlesen der CEL-Dateien benötigen wir aus dem Paket `affy` den Befehl `ReadAffy()`. Da wir uns aktuell in dem Verzeichnis befinden, in dem die CEL-Dateien vorliegen, brauchen wir keine weiteren Parameter anzugeben.

```
> batch <- ReadAffy()
```

Eine kurze Übersicht des Objektes bekommt man durch das Aufrufen des Objektnamens.

```
> batch
AffyBatch object
size of arrays=1164x1164 features (20 kb)
cdf=HG-U133_Plus_2 (54675 affyids)
number of samples=10
number of genes=54675
annotation=hgu133plus2
notes=
```

Wie wir der Ausgabe entnehmen können, haben wir ein Objekt vom Typ `AffyBatch`. Darin enthalten sind 10 Experimente mit je 54675 Expressionswerten. Die Annotation, bzw. die Plattform der Arrays ist vom Typ `hgu133plus2`.

Nützlich kann es sein, die Rohdaten zu exportieren. Auf die Expressionswerte der AffyBatch kann man durch den Befehl `exprs()` zugreifen. Gespeichert werden die Daten durch das Kommando `write.table()` in die Datei `Raw_data.csv`. Zur Strukturierung der Datei geben wir an, dass die Daten TAB-separiert sein sollen und das die darin enthaltenen Strings nicht in Anführungzeichen zu setzen sind.

```
> data.raw <- exprs(batch)
> write.table(data.raw,"Raw_data.csv", sep="\t", quote=FALSE)
```

Datenströme: Funktionen und deren Parameter

```
> ReadAffy()     : Einlesen der aller CEL-Dateien innerhalb des Workspace
  filenames      : Vektor von Dateinamen (selektives einlesen)
  celfile.path   : Pfad zu den CEL-Dateien
  phenoData      : Uebergabe der Experiment-Datei als data.frame
> read.table()   : einlesen von CSV-Dateien
  sep            : Angabe des Trennzeichen. (Tabulator := "\t")
  header         : Ob Datei eine Titlezeile beinhaltet.
  row.names      : Name der Spalte die als Referenz genutzt werden soll
> write.table()  : Schreiben von Dateien.
  sep            : Angabe des Trennzeichen. (Tabulator := "\t")
  quote          : Soll der Text in Anfuehrungszeichen stehen.
```

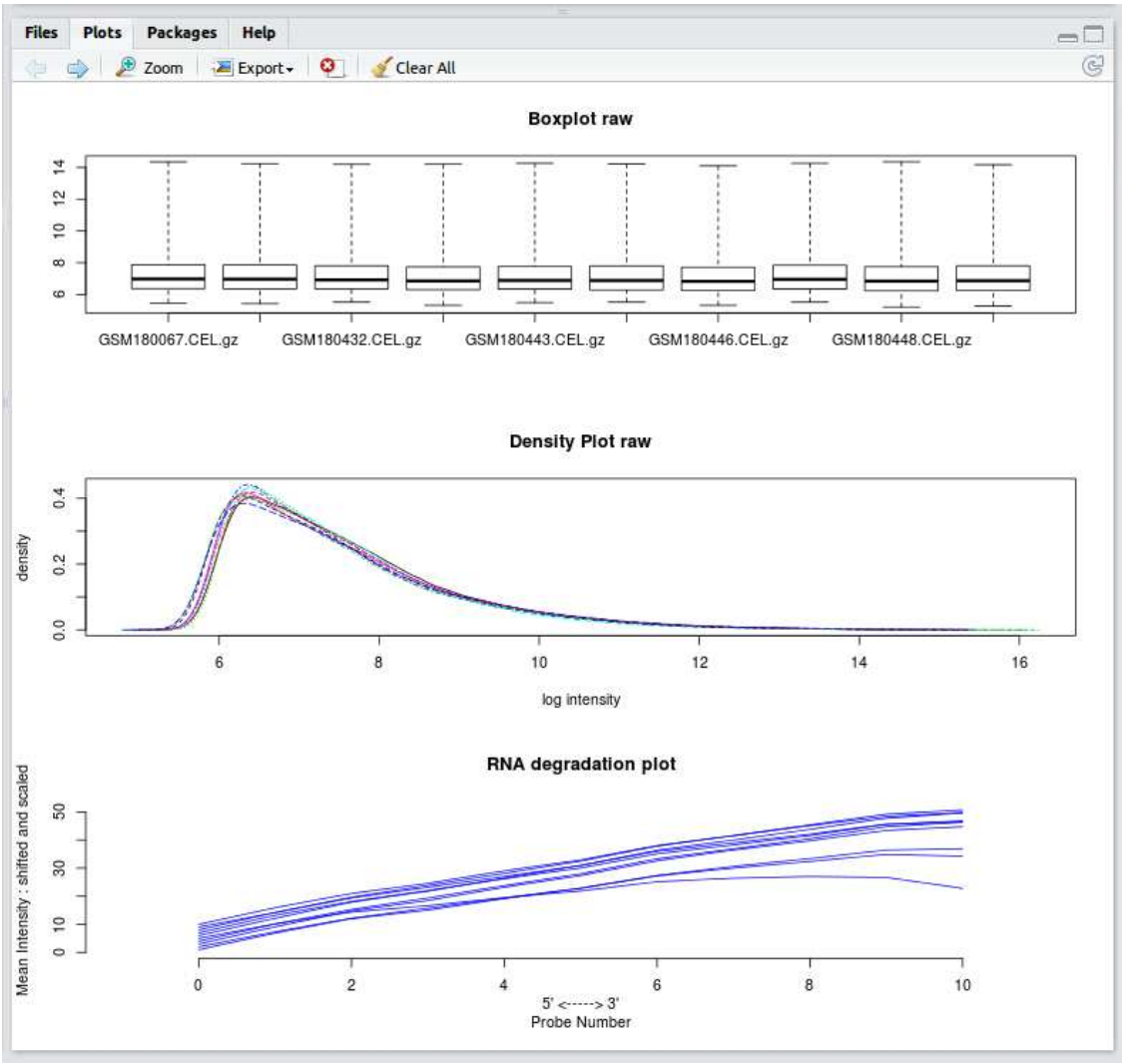
4.3.2 Qualitätskontrolle der Rohdaten

Zur Qualitätskontrolle der Rohdaten erstellen wir einen Boxplot für jedes Microarray. Des weiteren ist ein Dichte-Verteilungs-Plot hilfreich. Weiterhin erstellen wir einen Plot, der uns den 'Zerfall/ Zersetzung' der verwendeten RNA innerhalb der einzelnen Arrays aufzeigt.

Um die Qualitätskontrolle übersichtlich zu halten erstellen wir diese zusammen in einer Grafik.

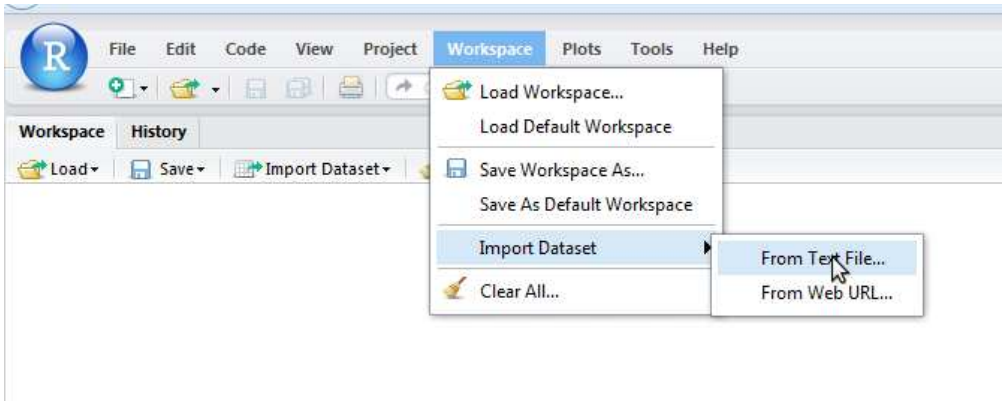
```
> old.par <- par(mfrow=c(3,1))
> boxplot(batch, main="Boxplot raw")
> plotDensity.AffyBatch(batch, main="Density Plot raw", log=TRUE)
> deg <- AffyRNAdeg(batch, log.it=TRUE)
> plotAffyRNAdeg(deg)
```

Dieser Code generiert uns die folgende Grafik

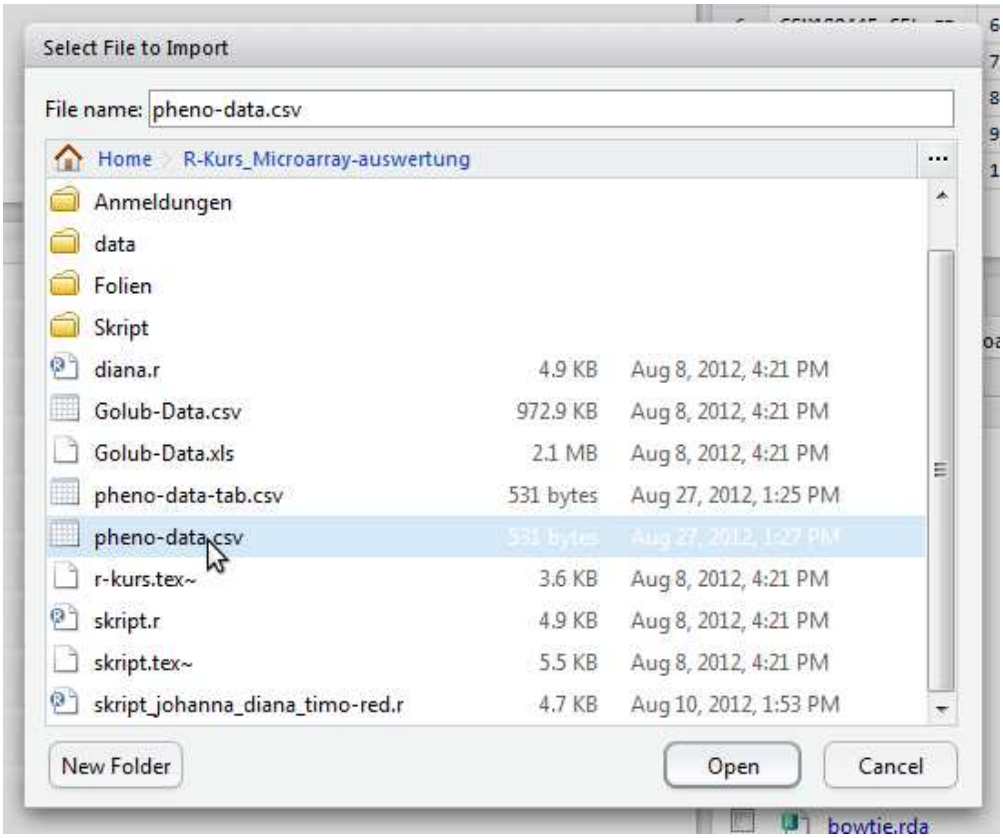


4.3.3 Einlesen der Phenodaten

Um die Phenodaten in RStudio zur Verfügung zu stellen, benutzen wir die Möglichkeiten, die uns der Server bietet. Hierzu öffnen wir in der Menüleiste den Eintrag `Workspace`. Unter `Import Dataset` wählen wir die Option `From Text File...` aus.

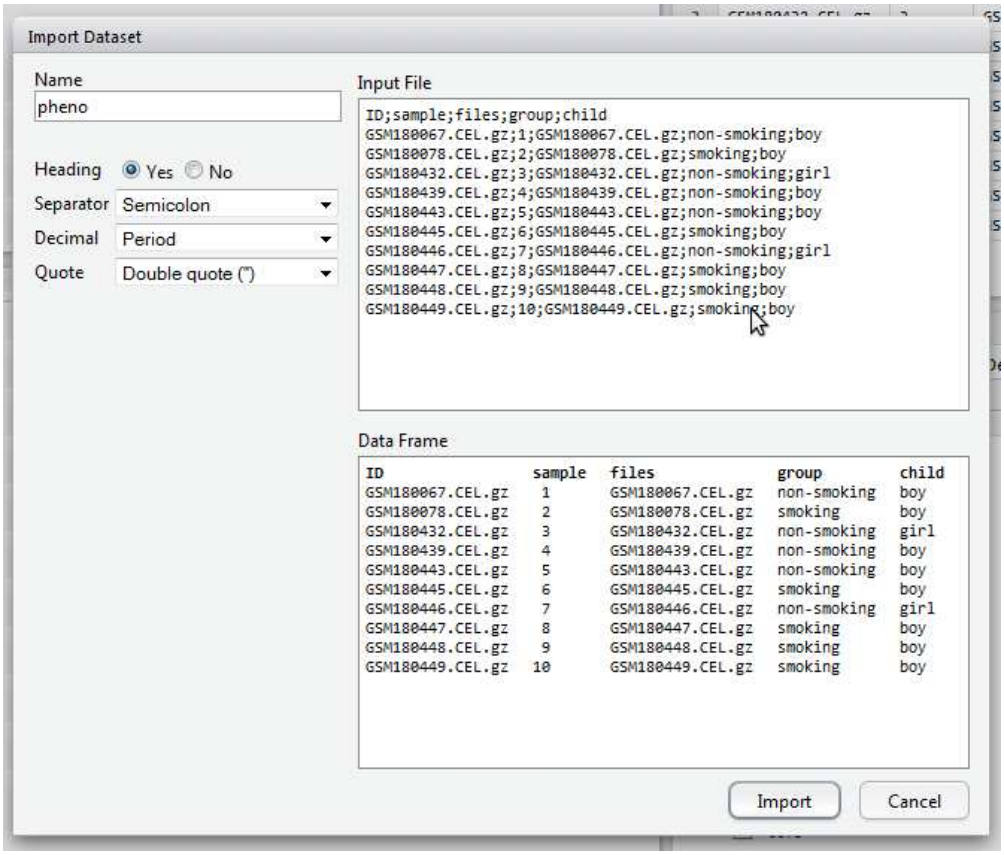


Es öffnet sich ein Dialog in dem wir die zuvor unter Abschnitt 4.1 konvertierte Datei `pheno-data.csv` auswählen.



Ein weiterer Dialog öffnet sich nun, indem wir diverse Einstellungen zum Importieren vornehmen können. Hierbei bekommen wir im oberen Feld `Input File` die Datei in ihrer ursprünglichen Version angezeigt. Im unteren Feld `Data Frame` bekommen wir eine Vorschau des Objektes zu sehen, wie es durch die Enstellungen modifiziert wird.

In unserem Fall geben wir dem Objekt den Namen `pheno`. Da wir Spaltennamen haben stellen wir `Heading` auf `Yes`. Die Datei ist Semikolon separiert und somit stellen wir als `Separator Semicolon` ein. Die beiden anderen Parameter können wir ignorieren, da wir keine reellen Zahlen innerhalb unseren Phenodaten haben, auch sind die Text-Inhalte nicht durch Anführungszeichen markiert.



Zum Bestätigen drücken wir auf `Import`. Im `Workspace`-Tab in dem oberen rechten Fenster der RStudio-Oberfläche sehen wir, dass die Daten unter dem Namen pheno verfügbar sind. Da diese Daten momentan noch getrennt von den Expressionsdaten vorliegen, speichern wir diese in das Objekt `batch`.

```
> rownames(pheno) <- pheno[["ID"]]
> pData(batch) <- pheno
```

4.4 Hintergrundkorrektur und Normalisierung

Die Normalisierung der Daten gestalten sich in dem Fall einfach. Mit der Funktion `rma()` [4, 5, 6] werden die 3 Bereiche

- Hintergrund-Normalisierung
- Normalisierung zwischen den Arrays
- Zusammenfassen der Probesets

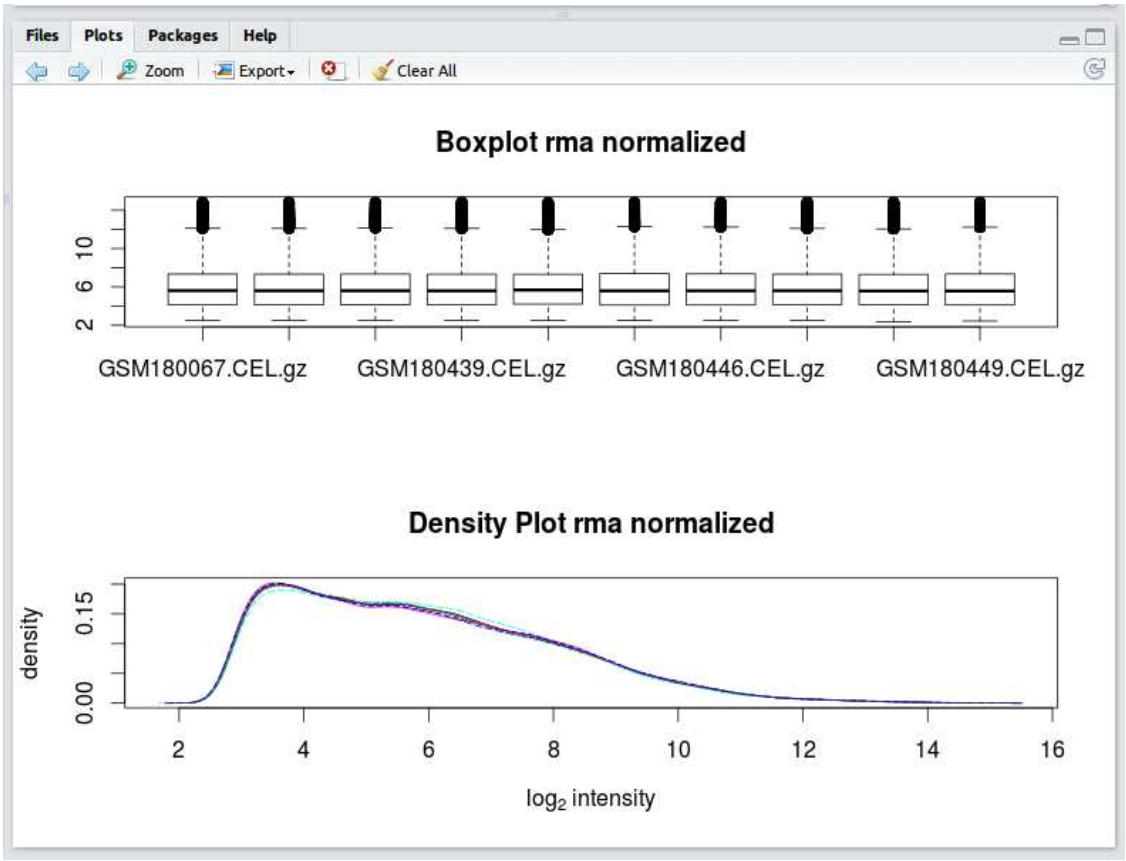
abgedeckt. Generell ist die Normalisierung komplex, daher gibt es verschiedene Lösungen und somit auch viele verschieden Funktionen.

```
> eSet <- rma(batch)
Background correcting
Normalizing
Calculating Expression
```

Um die Qualität der Normalisierung zu überprüfen, erstellen wir gleich noch mal wie unter Abschnitt 4.3.2 einen Boxplot der jeweiligen normalisierten Microarrays sowie ebenfalls einen Dichte-Verteilungs-Plot.

```
> par(mfrow=c(2,1))
> boxplot(exprs(eSet), main="Boxplot rma normalized")
> plotDensity(exprs(eSet), main="Density Plot rma normalized")
```

Der daraus enstehende Grafik sieht wie folgt aus



4.5 Die ExpressionSet-Klasse

Microarray-Experimente enthalten normalerweise verschiedene Daten. Die Expressionen stehen dabei im Mittelpunkt, aber die Informationen über die Experimente wie Gruppierung, klinische Parameter, Geschlecht, etc. oder aber auch deren Zuordnung bezüglich der Gene und Sequenzen vervollständigen die Experimentdaten. Um das Verarbeiten der Daten zu erleichtern, werden die Informationen bezüglich des Experimentes in einem Objekt vom Typ `ExpressionSet` zusammengefasst.

ExpressionSets und dessen Funktionen werden in dem Paket `Biobase` [7] zur Verfügung gestellt. Eine schnelle Übersicht der enthaltenen Daten erhält man durch aufrufen des Objektes `eSet`

```
> eSet
ExpressionSet (storageMode: lockedEnvironment)
assayData: 54675 features, 10 samples
  element names: exprs
protocolData
  sampleNames: GSM180067.CEL.gz GSM180078.CEL.gz ... GSM180449.CEL.gz (10 total)
  varLabels: ScanDate
  varMetadata: labelDescription
phenoData
  sampleNames: GSM180067.CEL.gz GSM180078.CEL.gz ... GSM180449.CEL.gz (10 total)
  varLabels: sample files group child
```

```
  varMetadata: labelDescription
featureData: none
experimentData: use 'experimentData(object)'
Annotation: hgu133plus2
```

In unserem Fall besteht das Experiment aus 10 Microarrays mit jeweils 54675 Spots. Die Metainformationen zu den Experimenten sind `files`, `group` und `child`. Informationen zu den Spots sind aktuell keine gelistet. Die Microarrays sind vom Typ `hgu133plus2`.

Mit der Funktion `pData()` bekommt man die experiment-spezifischen Daten gelistet

```
> pData(eSet)
                sample            files       group child
GSM180067.CEL.gz      1 GSM180067.CEL.gz non-smoking   boy
GSM180078.CEL.gz      2 GSM180078.CEL.gz     smoking   boy
GSM180432.CEL.gz      3 GSM180432.CEL.gz non-smoking  girl
GSM180439.CEL.gz      4 GSM180439.CEL.gz non-smoking   boy
GSM180443.CEL.gz      5 GSM180443.CEL.gz non-smoking   boy
GSM180445.CEL.gz      6 GSM180445.CEL.gz     smoking   boy
GSM180446.CEL.gz      7 GSM180446.CEL.gz non-smoking  girl
GSM180447.CEL.gz      8 GSM180447.CEL.gz     smoking   boy
GSM180448.CEL.gz      9 GSM180448.CEL.gz     smoking   boy
GSM180449.CEL.gz     10 GSM180449.CEL.gz     smoking   boy
```

Mit der Funktion `exprs()` wird die Expressionsmatrix zurück gegeben

```
> head(exprs(eSet))
          GSM180067.CEL.gz GSM180078.CEL.gz GSM180432.CEL.gz GSM180439.CEL.gz
1007_s_at        11.209749        11.093431        11.004637        10.994296
1053_at           6.993569         6.990220         6.949070         7.233663
117_at            8.925019         7.673605         7.813071         7.613706
121_at            7.878124         7.968303         7.830270         7.707467
1255_g_at         3.172054         3.119008         3.103605         3.360302
1294_at           8.062381         7.707101         7.728834         7.809180
 ...
```

Informationen zur Zuordnung der Spots können über `fData()` bezogen werden. In unserem Fall sind noch keine Daten vorhanden.

```
> fData(eSet)
data frame with 0 columns and 54675 rows
```

Das Auslesen der Microarray-Plattform erfolgt durch den Befehl `annotation()`.

```
> anotation(eSet)
[1] "hgu133plus2"
```

Methoden von ExpressionSets

```
> exprs()       : Auslesen der Expressionswerte als Matrix
> pData()       : Auslesen der Metainformationen des Experiments
> fData()       : Auslesen der Metainformationen der Spots
> annotation()  : Auslesen der Chip-Plattform und der benutzten Annotation
> show()        : Ausgabe einer Uebersicht. Wenn man ausschließlich den Objektnamen
                  aufruft wird ebenfalls die Methode zur Visualisierung genutzt.
```

4.6 Filtern nicht-informativer Gene

Unser Interesse gilt den differentiell exprimierten Genen. Daher filtern wir das Rauschen, aber auch die Gene, die keine größere Regulation erfahren, heraus. Zur Detektion dieser Gene kann man die Streuung der jeweiligen Expressionswerte erfassen, da in diesem Falle eine kleine Streuung auf eine nicht-, bzw. nicht nennenswerte Regulation zurück schließen läßt.

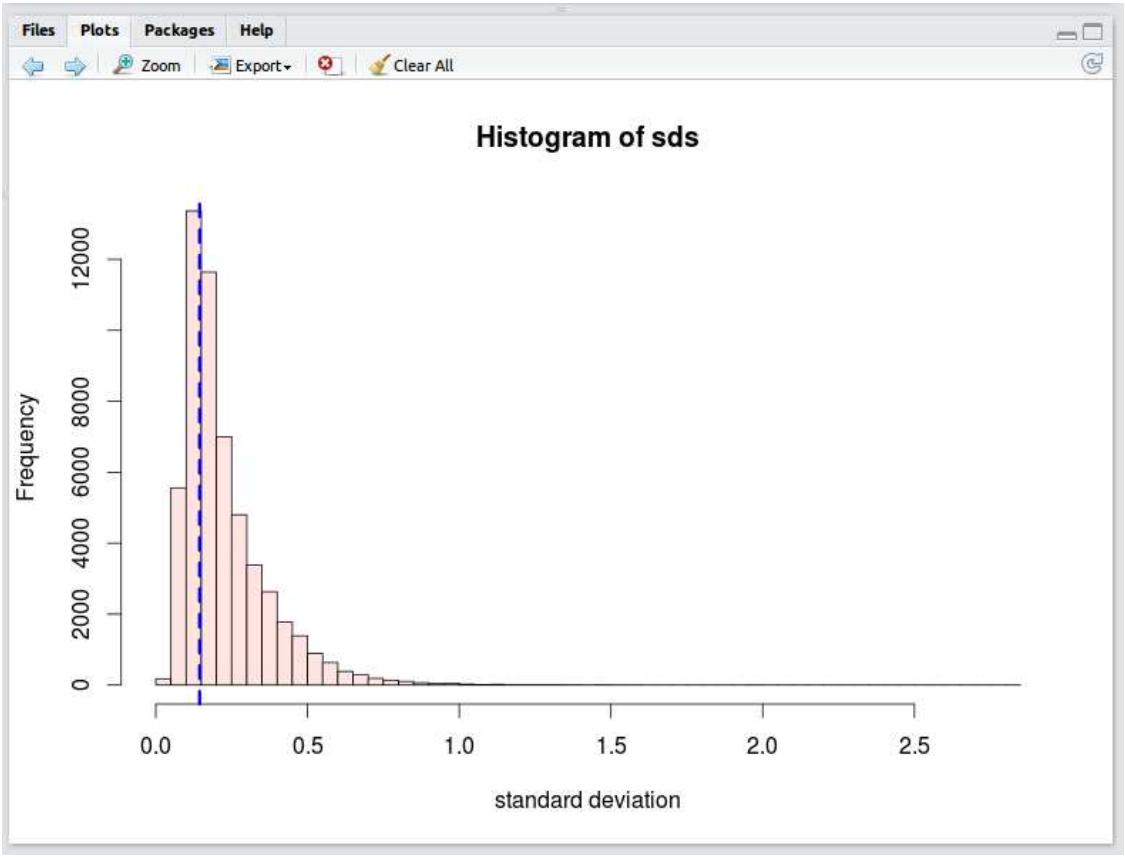
Hierzu benutzen wir Funktionen aus dem Paket `genefilter` [8], welches vorab noch in unsere Bibliothek zu installieren ist.

```
> biocLite(c("genefilter", "multtest"))
> library(multtest)
> library(genefilter)
> par(old.par)
> sds <- rowSds(exprs(eSet))
> sh <- shorth(sds)
> sh
[1] 0.1443116
```

Die Variable `sh` beinhaltet nun den Wert, der uns als Schwelle zum Filtern der interessanten Gene dient. Visualisieren läßt sich das Ergebnis als Histogramm in das der 'Cutoff' als blaue vertikale Linie eingetragen wird.

```
> hist(sds,breaks=50,col="mistyrose",xlab="standard deviation")
> abline(v=sh,col="blue",lwd=3,lty=2)
```

Das Ergebnis der letzten beiden Befehle führt zur folgenden Grafik



Das Filtern erfolgt durchs Bilden einer Teilmenge der interessanten Spots aus dem ExpressionSet

```
> eSetfilt <- eSet[sds>=sh,]
> dim(eSet)
Features  Samples
   54675       10
> dim(eSetfilt)
Features  Samples
   37158       10
```

Durch das unspezifische Filtern haben wir vorab den Datensatz um ca. 17500 Spots reduziert.

4.7 Testen für differentiell exprimierte Gene

Die differentiell exprimierten Gene werden in Abhängigkeit des Experimentaufbaus erstellt. Hierzu benutzen wir die Gruppendefinition aus der Spalte `group` der Phenodaten. Eine Übersicht der Gruppengröße kann man mit dem Befehl `table()` erhalten.

```
> table(eSetfilt$group)

non-smoking     smoking
          5           5
```

Signifikante Unterschiede der Spots berechnen wir mit der Funktion `rowttests()` aus dem Paket `multtest`[9]. Hierbei wird zeilenweise die Expressionswerte der einzelnen Gruppen durch einen t-Test auf differentielle Exprimierung getestet.

```
> tt <- rowttests(eSetfilt,"group")
> names(tt)
[1] "statistic" "dm"        "p.value"
```

4.8 Korrektur für multiples Testen

Der t-Test berechnet eine Warscheinlichkeit inwieweit die Null-Hypothese zutrifft. Ein p-Wert von 0.01 besagt, dass eine Warscheinlichkeit von 1% besteht, dass obwohl die These abgelehnt wird, die Null-Hypothese doch zu trifft. Da die einzelnen Spots unabhängig getestet wurden, besteht die Möglichkeit in diesem Fall, dass im Mittel jedes 100. Gen mit einem p-Wert von 0.01 doch nicht differentiell reguliert ist. Daher ist ein adjustieren der p-Werte sinvoll um die Anzahl der möglichen Falsch-Positiven zu kontrollieren. Hierfür benutzen wir die Funktion `mt.rawp2adjp()` aus dem Paket `multest`.

```
> mt <- mt.rawp2adjp(tt$p.value,proc="BH")
```

Nun lassen wir uns die 10 Gene mit den kleinsten p-Werten ausgeben. Hierzu extrahieren wir aus den adjustierten p-Werten die ersten 10 Indeces und speichern uns die zugehörigen Namen der Spots in `g`. Zur Annotation des Microarrays installieren wir uns das Paket `hgu133plus2.db` und laden es in die Arbeitsumgebung. Mit dem Befehl `links` verknüpfen wir die Spot-Namen mit den zugehörigen Genesymbolen aus der Umgebung `hgu133plus2SYMBOL`.

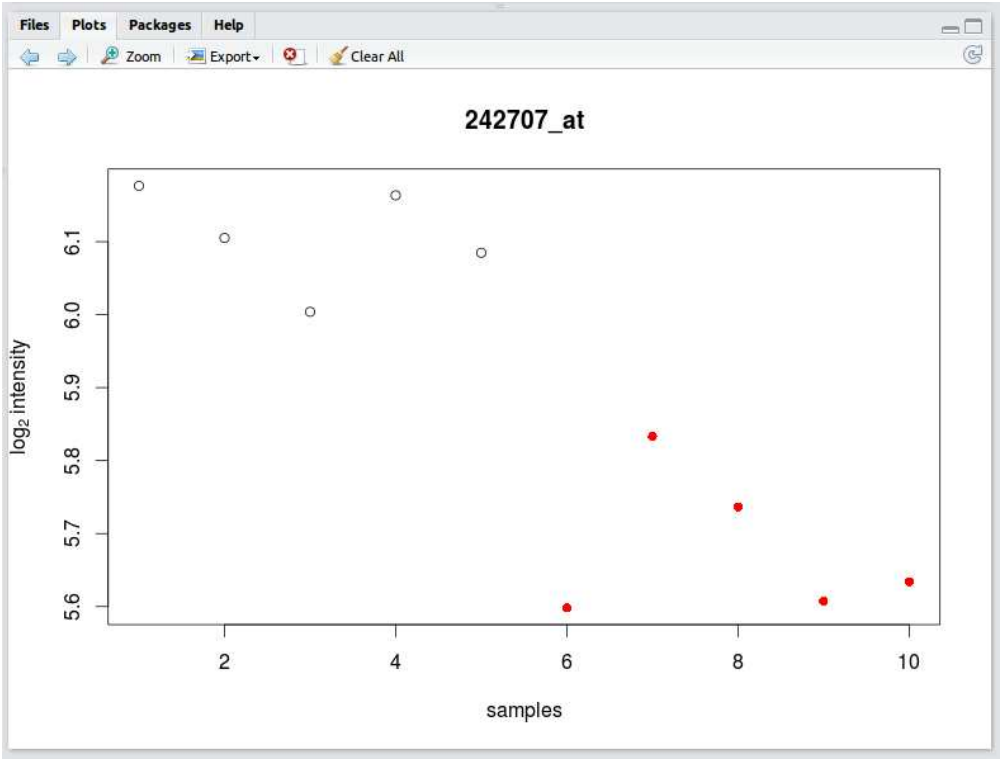
```
> biocLite("hgu133plus2.db"}
> g <- featureNames(eSetfilt)[mt$index[1:10]]
> library("hgu133plus2.db")
> links(hgu133plus2SYMBOL[g])
     probe_id       symbol
1 201123_s_at        EIF5A
2 203770_s_at          STS
3 205161_s_at       PEX11A
```

```
4   213748_at       TRIM66
5   219794_at        VPS53
6   220572_at DKFZp547G183
7 224045_x_at    LINC00470
8   237750_at      XPNPEP3
9   242707_at        MED23
```

Als nächstes lassen wir uns die Expressionswerte des Spots mit dem kleinsten p-Wert anzeigen. Hierzu extrahieren wir uns den Vektor der Expressionen des Spots. Diesen plotten wir sortiert nach den Gruppen und färben die Werte gruppenabhängig ein.

```
> mb <- eSetfilt$group
> y <- exprs(eSetfilt)[g[1],]
> ord <- order(mb)
> par(old.par)
> plot(y[ord],pch=c(1,16)[mb[ord]],col=c("black","red")[mb[ord]],
    main=g[1],ylab=expression(log[2]~intensity),xlab="samples")
```

Das Ergebnis der Befehle erhalten wir folgende Grafik

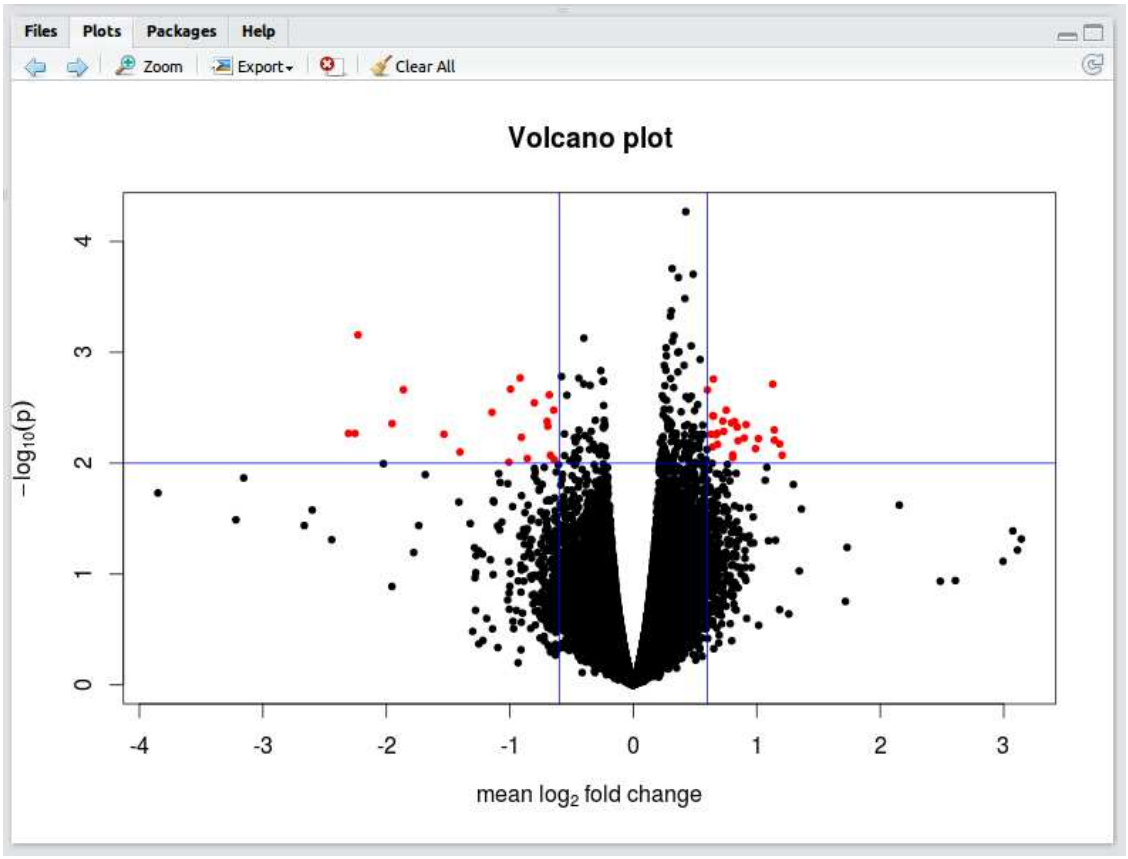


4.9 Der Volcano-Plot

Eine sehr anschauliche Variante, die differentielle Expression mit den zugehörigen p-Werten aller Spots darzu stellen, ist der Volcano-Plot. Hierzu wird auf der Y-Achse der negative, logarithmierte p-Wert, auf der X-Achse der log2-Foldchange der jeweilgen Spots aufgetragen.

```
> col <- ifelse((abs(tt$dm)>0.6) & (-log10(tt$p.value) > 2),"red","black")
> plot(tt$dm,-log10(tt$p.value),col=col,pch=20,
    main="Volcano plot",xlab=expression(mean~log[2]~fold~change),ylab=expression(-log[10](p)))
> abline(h=2, v=c(-0.6,0.6), col="blue")
```

Als Feature haben wir die Spots, mit den p-Werten kleiner 0.01 sowie einer log 2-Regulation von größer 0.6, bzw. kleiner −0.6 besitzen, rot eingefärbt. Die jeweiligen Grenzen sind durch blaue Linien gekennzeichnet.



4.10 Das limma-Paket

Mit dem Paket `limma` besteht ebenfalls die Möglichkeit die Experimente zu analysieren. Die p-Werte werden hierbei durch eine andere Methode berechnet, die bei größeren Experimenten mit mehr unabhängigen Replikaten sehr ähnliche Ergebnisse wie der t-Test bietet, hingegen bei kleineren Experimenten diese deutlich verbessert. Daher wollen wir die letzten Schritte zur Bildung der differentiell exprimierten Gene mit dem limma-Paket wiederholen.

Hierzu müssen wir der Expressionsmatrix die Informationen zum Design des Experiments mitgeben. Diese Information erstellen wir wie folgt.

```
> biocLite("limma")
> library(limma)
> cl <- as.numeric(eSetfilt$group=="smoking")
> design <- cbind(mean=1, diff=cl)
```

Im nächsten Schritt werden die Expressionsmatrix mit der Designmatrix gefittet und durch den Befehl `eBayes()` berechnet.

```
> fit <- lmFit(exprs(eSetfilt),design)
> fit <- eBayes(fit)
```

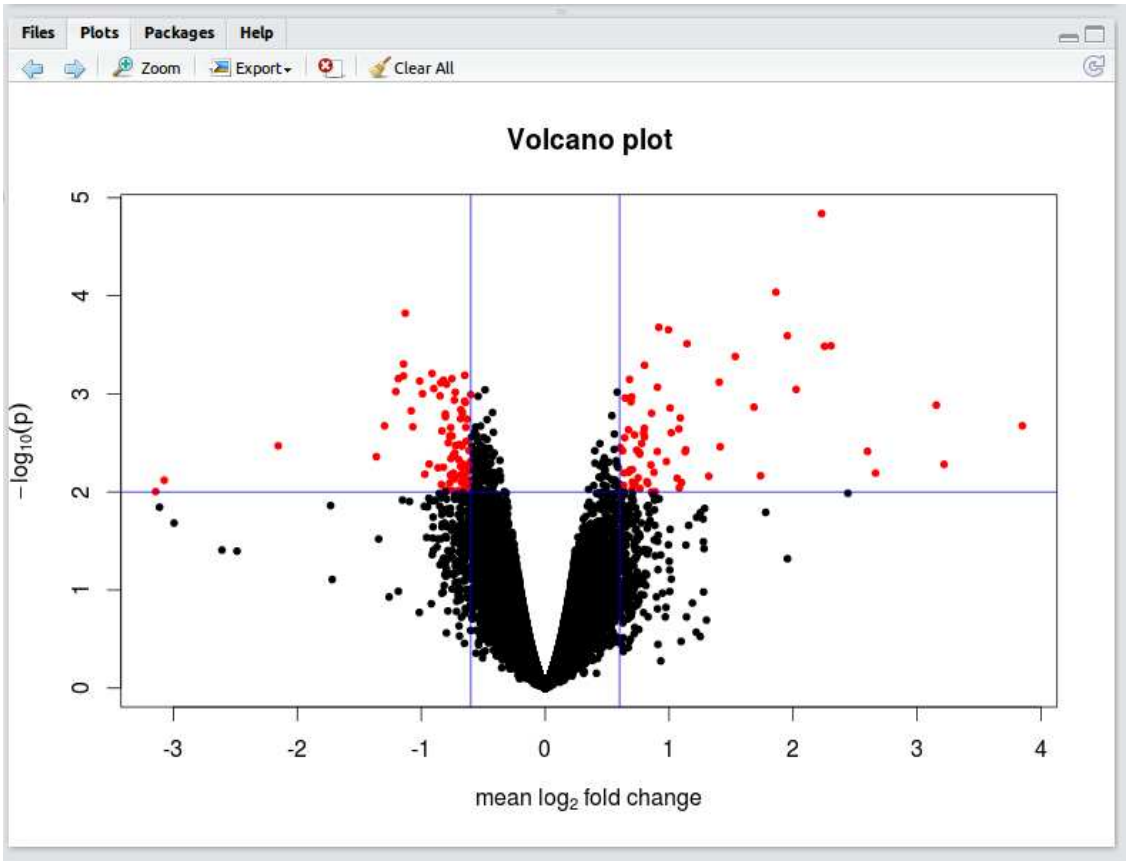
Mit dem Befehl `topTable()` bekommt man eine aufsteigend nach den p-Werten aufsortierte Übersicht der Spots. Diesem Befehl kann man Informationen zur Annotation der Spots übergeben; in unserem Fall die Gen-Symbole.

```
> glim <- cbind(ID=rownames(exprs(eSetfilt)),
                Symbol=unlist(mget(rownames(exprs(eSetfilt)), hgu133plus2SYMBOL)))
> show(g <- topTable(fit,coef="diff",adjust.method="fdr",sort.by="p",genelist=glim))
               ID   Symbol      logFC  AveExpr         t      P.Value adj.P.Val         B
5429  201123_s_at    EIF5A  2.2302886 7.019775  6.608740 1.451280e-05 0.5392667 -2.940430
28131   230760_at      ZFY  1.8613775 5.873791  5.497874 9.166003e-05 0.8364893 -3.134695
28995   232141_at    U2AF1 -1.1292945 7.551850 -5.216171 1.502964e-04 0.8364893 -3.194896
5428  201122_x_at    EIF5A  0.9165121 8.077492  5.030867 2.092964e-04 0.8364893 -3.237276
30669   234994_at TMEM200A  0.9944483 4.009409  4.999113 2.216175e-04 0.8364893 -3.244769
6566  202436_s_at   CYP1B1  1.9540727 7.318886  4.920625 2.554190e-04 0.8364893 -3.263586
24531   226322_at    TMTC1  1.1443785 6.475873  4.818688 3.074927e-04 0.8364893 -3.288664
6567  202437_s_at   CYP1B1  2.3064885 6.753689  4.791216 3.233322e-04 0.8364893 -3.295548
8297    204410_at   EIF1AY  2.2542094 6.235208  4.785208 3.269078e-04 0.8364893 -3.297061
6565  202435_s_at   CYP1B1  1.5338856 5.512225  4.655115 4.152621e-04 0.8364893 -3.330451
```

Die Werte für den Volcano-Plot extrahieren wir diesmals aus dem `topTable`. Um die gesamte Liste zu bekommen geben wir über den Parameter `number` die Anzahl der Spots mit. Wir benötigen hierzu nur die Spalten `logFC` sowie den `P.Value`.

```
> x <- topTable(fit, coef="diff", adjust.method="fdr", sort.by="p",
         genelist=glim, number=nrow(fit$p.value))[,c("logFC", "P.Value")]
> col <- ifelse((abs(x[,"logFC"])>0.6) & (-log10(x[,"P.Value"]) > 2),"red","black")
> plot(x[,1],-log10(x[,2]),col=col,pch=20,main="Volcano plot",
         xlab=expression(mean~log[2]~fold~change),
         ylab=expression(-log[10](p)))
> abline(h=2, v=c(-0.6,0.6), col="blue")
```

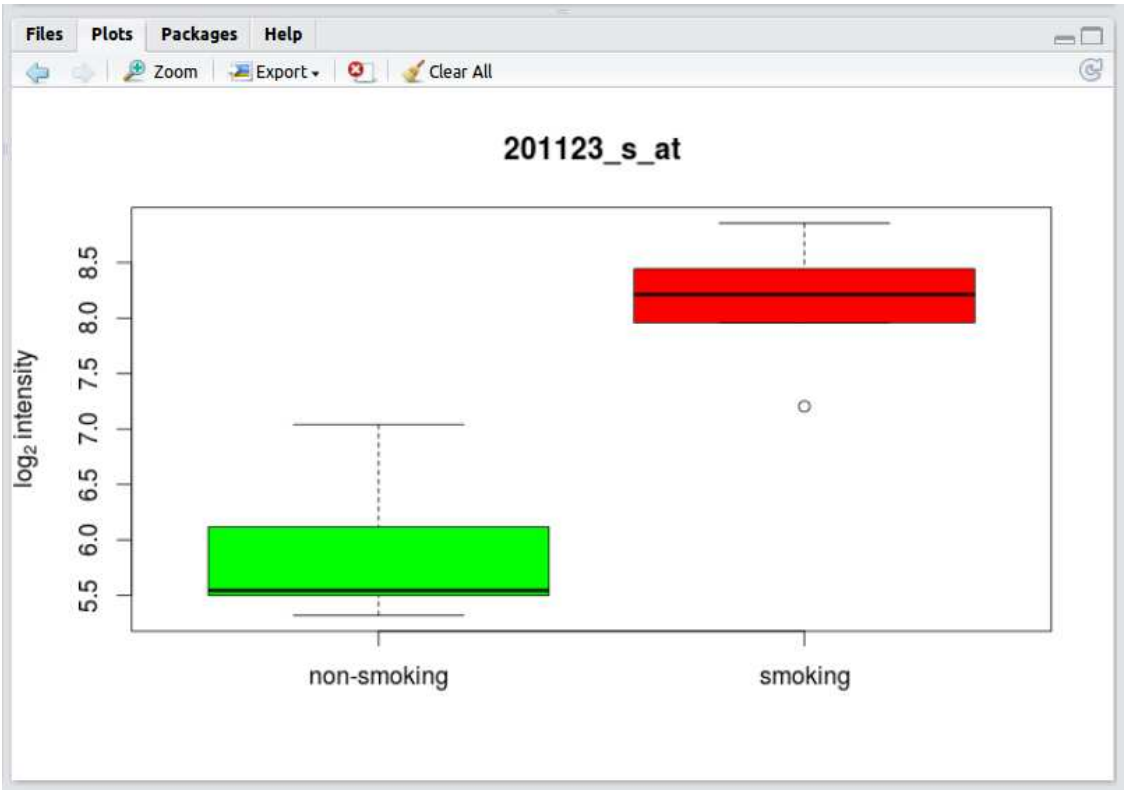
Die zugehörige Grafik sieht wie folgt aus



Die Expressionswerte des besten Spots sehen wir uns als Boxplot der jeweiligen Gruppe an. Mit dem folgenden Befehl

```
> boxplot(exprs(eSetfilt)[g[1],] ~ eSetfilt$group, main=g[1],
          ylab=expression(log[2]~intensity),
          col=c("green", "red")[unique(eSetfilt$group)])
```

erhalten wir folgende Grafik



Die Darstellung zeigt eine klare Abgrenzung der Gruppen durch ihre jeweiligen Expressionswerte.

4.11 GO-Terme und hypergeometrisches Testen

Eine weitere Information über die differentiell exprimierten Gene bietet uns die Anreicherung der GO-Terme. Dazu verwenden wir das Paket `GOstats`. Hierbei werden die differentiell exprimierten Gene im Kontext aller Gene auf dem Microarray hinsichtlich der Terms analysiert. Die GO-Terms werden durch die EntrezID gelistet, daher müssen wir die Spots in diese ID's überführen. Dies ist für die Referenz, als auch für die differentiell exprimierten Gene notwendig.

```
> biocLite("GOstats")
> library(GOstats)
> affyUniverse <- featureNames(eSetfilt)
> uniId <- hgu133plus2ENTREZID[affyUniverse]
> entrezUniverse <- unique(as.character(uniId))
> hitGenes <- fit$genes$ID[fit$p.value[,"diff"]<0.01]
> EGsub <- unique(as.character(hgu133plus2ENTREZID[hitGenes]))
```

Zur eigentlichen Analyse werden wir zuvor eine Klasse `GOHyperGParams` erstellen und mit den notwendigen Parametern beladen. Mit dem Befehl `new(''Objektname'')` wird die Klasse erzeugt. Dem Parameter

`geneIds` geben wir die EntrezIDs der differentiell exprimierten Gene mit, die Referenz wird dem Parameter `universeGeneIds` übergeben. Die Annotation wird dem Parameter `annotation` zugewiesen. Wir sind an der Kategorie 'Biologische Prozesse' interessiert und setzen den Parameter `ontology` auf `BP`. Wir wollen 'Biologische Prozesse' identifizieren, die einen p-Wert kleiner 0.001 haben. Weiterhin setzen wird den Parameter `testDirection` auf `over`. Mit dem Befehl `hyperGTest()` wird die Analyse des vorab erstellten Objektes gestartet.

```
> params <- new("GOHyperGParams",geneIds=EGsub,
        universeGeneIds=entrezUniverse,
        annotation="hgu133plus2",ontology="BP",
pvalueCutoff=0.001,conditional=FALSE,testDirection="over")
> mfhyper <- hyperGTest(params)
```

Mit dem Befehl `summary` können wir uns das Ergebnis der Analyse in Tabellenform ansehen. Durch den Parameter `p=0.001` filtern wir die Ergebnistabelle.

```
> summary(mfhyper,p=0.001)
      GOBPID       Pvalue  OddsRatio   ExpCount Count Size
1 GO:2000602 0.0002638029   6.137129 1.26006300     7  120
2 GO:0045750 0.0003257966 191.644068 0.03150158     2    3
3 GO:0070672 0.0006471107  95.813559 0.04200210     2    4
4 GO:0007090 0.0006895749  20.684982 0.17850893     3   17
5 GO:0045087 0.0007653863   3.230496 4.07420371    12  388
                                                 Term
1       regulation of interphase of mitotic cell cycle
2 positive regulation of S phase of mitotic cell cycle
3                           response to interleukin-15
4          regulation of S phase of mitotic cell cycle
5                               innate immune response
```

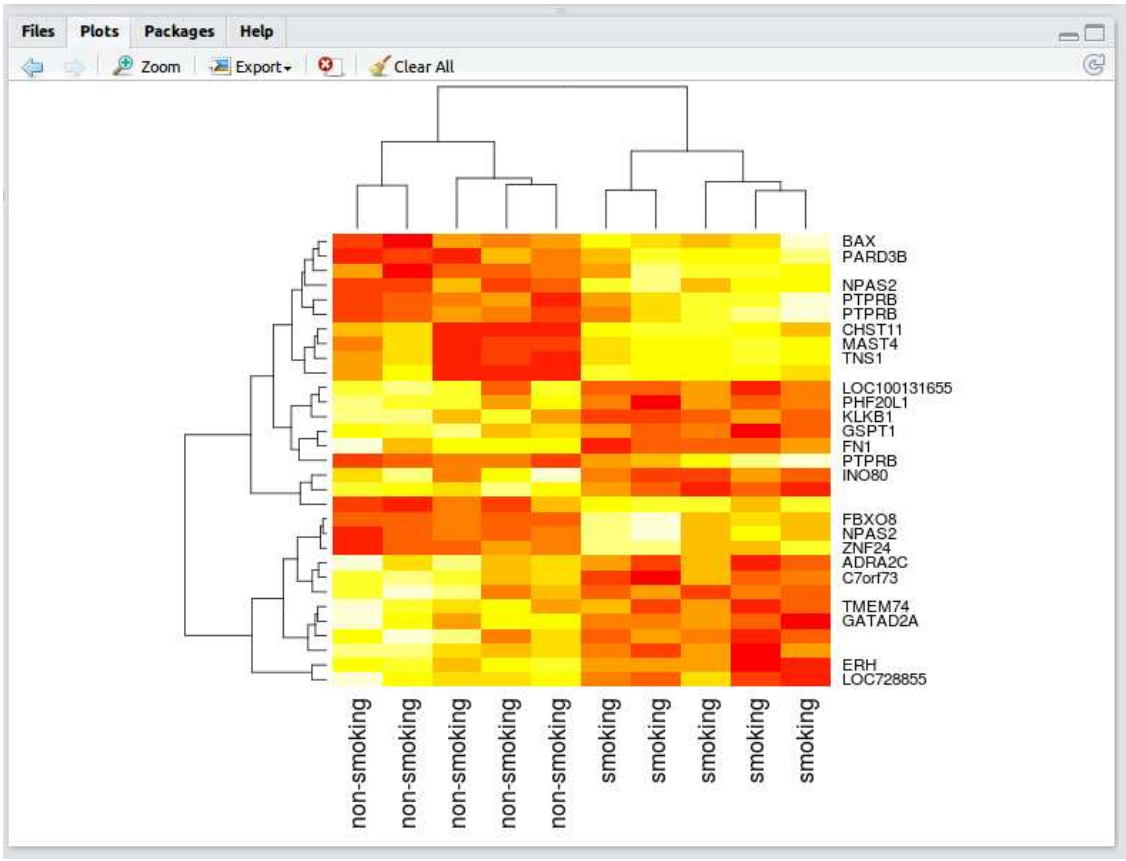
Weiterhin besteht die Möglichkeit die Definition der einzelnen GO-Terms auszulesen. Das Objekt `GOTERM` wird dabei wärend der Analyse angelegt.

```
> GOTERM[["GO:2000602"]]
GOID: GO:2000602
Term: regulation of interphase of mitotic cell cycle
Ontology: BP
Definition: Any process that modulates the frequency, rate or extent of interphase of
    mitotic cell cycle.
```

4.12 Die Heatmap

Gerne wird auch die Analyse der Gene sowie der Gruppen kombiniert angezeigt. Diese Analyse wird durch eine Heatmap visualisiert, wodurch man eine Übersicht der Gruppenzugehörigkeit in Abhängigkeit der Gene angezeigt bekommt. Die folgenden Befehle erzeugen eine "Heatmap" und gruppieren die Einträge der 2 Achsen jeweils durch ein hierachisches Clustering der Daten.

```
> heatmap(exprs(eSetfilt)[fit$p.value[,"diff"]<0.001,],
         labCol=eSetfilt$group,
         labRow=glim[fit$p.value[,"diff"]<0.001,2],
         margins=c(7,5))
```


Files
Plots
Packages
Help
Zoom
Export
Clear All
BAX
PARD3B
NPAS2
PTPRB
PTPRB
CHST11
MAST4
TNS1
LOC100131655
PHF20L1
KLKB1
GSPT1
FN1
PTPRB
INO80
FBXO8
NPAS2
ZNF24
ADRA2C
C7orf73
TMEM74
GATAD2A
ERH
LOC728855
non-smoking
non-smoking
non-smoking
non-smoking
non-smoking
smoking
smoking
smoking
smoking
smoking

5 Die Core Facility Bioinformatik

Die Core Facility Bioinformatik bietet Expertise und Software für die

- statistische Modellierung und Visualisierung hochdimensionaler molekularer Daten
- Analyse von Microarray-Daten (SNPs, mRNA, miRNA oder Methylierung) und von "next-generation" Sequenzierungsdaten (z.B. RNA-Seq)
- Signalwegs-, Netzwerk- und biologische Funktionsanalyse differentiell regulierter Gene
- Proteomanalyse
- Integration von Daten aus verschiedenen biologischen Ebenen (Transkriptom, Proteom, etc.)
- problemspezifische Anpassung und Neuentwicklung bioinformatischer Techniken

Die benutzte Software für die obigen Aufgaben sind, unter anderem, für die Datenanalyse hochdimensionaler molekularer Daten

- R/Bioconductor
- Chipster
- BRB-ArrayTools

für die Analyse der Signalwege und Netzwerke differentiell regulierter Gene

- DAVID
- Ingenuity
- WebGestalt
- GOMiner
- STEM

für die Visualisierung der Daten und Ergebnisse aus statistischer Modellierung und Netzwerkanalyse

- VennDiagram
- Cytoscape
- BiologicalNetworks2.0

5.1 Kontakt

Die kompetenten Mitarbeiter der BIUM$^{(MZ)}$ sind Johanna Mazur (`mazur@uni-mainz.de`) und Timo Itzel (`itzel@uni-mainz.de`). Die Core Facility Bioinformatik ist telefonisch zu erreichen unter der

17 **8207**

und befindet sich im Erdgeschoss des IMBEI (Gebäude 902) im Raum 007.

Wir freuen uns auf spannende gemeinsame Projekte!

Referenzen

[1] Golub *et al.* (1999). Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science*, Vol. 286:531–537.

[2] Huuskonen P, Storvik M, Reinisalo M, Honkakoski P et al. Microarray analysis of the global alterations in the gene expression in the placentas from cigarette-smoking mothers. *Clin Pharmacol* Ther 2008 Apr;83(4):542-50. *PMID: 17928820*

[3] Gautier, L., Cope, L., Bolstad, B. M., and Irizarry, R. A. 2004. affy—analysis of Affymetrix GeneChip data at the probe level. *Bioinformatics* 20, 3 (Feb. 2004), 307-315.

[4] Rafael. A. Irizarry, Benjamin M. Bolstad, Francois Collin, Leslie M. Cope, Bridget Hobbs and Terence P. Speed (2003), Summaries of Affymetrix GeneChip probe level data. *Nucleic Acids Research* 31(4):e15

[5] Bolstad, B.M., Irizarry R. A., Astrand M., and Speed, T.P. (2003), A Comparison of Normalization Methods for High Density Oligonucleotide Array Data Based on Bias and Variance. *Bioinformatics* 19(2):185-193

[6] Irizarry, RA, Hobbs, B, Collin, F, Beazer-Barclay, YD, Antonellis, KJ, Scherf, U, Speed, TP (2003) Exploration, Normalization, and Summaries of High Density Oligonucleotide Array Probe Level Data. *Biostatistics* Vol. 4, Number 2: 249-264

[7] Bioconductor: Open software development for computational biology and bioinformatics R. Gentleman, V. J. Carey, D. M. Bates, B.Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, and others 2004, *Genome Biology*, Vol. 5, R80

[8] R. Gentleman, V. Carey, W. Huber and F. Hahne (). genefilter: genefilter: methods for filtering genes from microarray experiments. R package version 1.36.0.

[9] Katherine S. Pollard, Houston N. Gilbert, Yongchao Ge, Sandra Taylor and Sandrine Dudoit (). multtest: Resampling-based multiple hypothesis testing. R package version 2.10.0.