iCheck: A package checking data quality of Illumina expression data

Weiliang Qiu[‡,*], Brandon Guo[‡,†], Christopher Anderson[‡,‡], Barbara Klanderman[‡,§], Vincent Carey[‡,¶], Benjamin Raby[‡,‖]

May 19, 2021

[‡]Channing Division of Network Medicine
Brigham and Women's Hospital / Harvard Medical School
181 Longwood Avenue, Boston, MA, 02115, USA

Contents

1 Overview of iCheck **2**

2 Exclude failed arrays **4**

3 Check QC probes **5**

4 Check squared correlations among genetic control (GC) arrays **6**

5 Exclude GC arrays **8**

6 Check squared correlations among replicated arrays **9**

7 Obtain plot of estimated density for each array **10**

8 Obtain plot of quantiles across arrays **11**

9 Exclude gene probes with outlying expression levels **12**

10 Obtain plot of the ratio (p_{95}/p_{05}) of 95-th percentile to 5-th percentile across arrays **13**

11 Exclude arrays with $p_{95}/p_{05} \leq 6$ **14**

12 Obtain Plot of principal components **15**

[*]stwxq (at) channing.harvard.edu
[†]brandowonder (at) gmail.com
[‡]christopheranderson84 (a) gmail.com
[§]BKLANDERMAN (at) partners.org
[¶]stvjc (at) channing.harvard.edu
[‖]rebar (at) channing.harvard.edu

13 Perform background correction, data transformation and normalization 16

14 Obtain Plot of principal components for pre-processed data 16

15 Incorporate phenotype data 18

16 Data analysis 18
16.1 lmFitWrapper and lmFitPaired 18
16.2 glmWrapper 20
16.3 lkhrWrapper 21

17 Result Visualization 22

18 Session Info 24

1 Overview of iCheck

The `iCheck` package provides QC pipeline and data analysis tools for high-dimensional Illumina mRNA expression data. It provides several visualization tools to help identify gene probes with outlying expression levels, arrays with low quality, batches caused technical factors, batches caused by biological factors, and gender mis-match checking, etc.

We first generate a simulated data set to illustrate the usage of iCheck functions.

```
>   library(iCheck)
>   if (!interactive())
+   {
+     options(rgl.useNULL = TRUE)
+   }
>   # generate sample probe data
>   set.seed(1234567)
>   es.sim = genSimData.BayesNormal(nCpGs = 110,
+     nCases = 20, nControls = 20,
+     mu.n = -2, mu.c = 2,
+     d0 = 20, s02 = 0.64, s02.c = 1.5, testPara = "var",
+     outlierFlag = FALSE,
+     eps = 1.0e-3, applier = lapply)
>   print(es.sim)
```

```
ExpressionSet (storageMode: lockedEnvironment)
assayData: 110 features, 40 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: subj1 subj2 ... subj40 (40 total)
  varLabels: arrayID memSubj
  varMetadata: labelDescription
featureData
  featureNames: probe1 probe2 ... probe110 (110 total)
  fvarLabels: probe gene chr memGenes
```

```
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation:
```

```
>   # create replicates
>   dat=exprs(es.sim)
>   dat[,1]=dat[,2]
>   dat[,3]=dat[,4]
>   exprs(es.sim)=dat
>   es.sim$arrayID=as.character(es.sim$arrayID)
>   es.sim$arrayID[1]=es.sim$arrayID[2]
>   es.sim$arrayID[3]=es.sim$arrayID[4]
>   es.sim$arrayID[5:8]="Hela"
>   # since simulated data set does not have 'Pass_Fail',
>   #  'Tissue_Descr', 'Batch_Run_Date', 'Chip_Barcode',
>   #  'Chip_Address', 'Hybridization_Name', 'Subject_ID', 'gender'
>   # we generate them now to illustrate the R functions in the package
>
>   es.sim$Hybridization_Name = paste(es.sim$arrayID, 1:ncol(es.sim), sep="_")
>   # assume the first 4 arrays are genetic control samples
>   es.sim$Subject_ID = es.sim$arrayID
>   es.sim$Pass_Fail = rep("pass", ncol(es.sim))
>   # produce genetic control GC samples
>   es.sim$Tissue_Descr=rep("CD4", ncol(es.sim))
>   # assume the first 4 arrays are genetic control samples
>   es.sim$Tissue_Descr[5:8]="Human Hela Cell"
>   es.sim$Batch_Run_Date = 1:ncol(es.sim)
>   es.sim$Chip_Barcode = 1:ncol(es.sim)
>   es.sim$Chip_Address = 1:ncol(es.sim)
>   es.sim$gender=rep(1, ncol(es.sim))
>   set.seed(12345)
>   pos=sample(x=1:ncol(es.sim), size=ceiling(ncol(es.sim)/2), replace=FALSE)
>   es.sim$gender[pos]=0
>   # generate sample probe data
>   es.raw = es.sim[-c(1:10),]
>   print(es.raw)
```

```
ExpressionSet (storageMode: lockedEnvironment)
assayData: 100 features, 40 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: subj1 subj2 ... subj40 (40 total)
  varLabels: arrayID memSubj ... gender (10 total)
  varMetadata: labelDescription
featureData
  featureNames: probe11 probe12 ... probe110 (100 total)
  fvarLabels: probe gene chr memGenes
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation:
```

```
>   # generate QC probe data
>   es.QC = es.sim[c(1:10),]
>   # since simulated data set does not have 'Reporter_Group_Name'
>   #  we created it now to illustrate the usage of 'plotQCCurves'.
>   fDat=fData(es.QC)
>   fDat$Reporter_Group_Name=rep("biotin", 10)
>   fDat$Reporter_Group_Name[3:4]="cy3_hyb"
>   fDat$Reporter_Group_Name[5:6]="housekeeping"
>   fDat$Reporter_Group_Name[7:8]="low_stringency_hyb"
>   fData(es.QC)=fDat
>   print(es.QC)

ExpressionSet (storageMode: lockedEnvironment)
assayData: 10 features, 40 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: subj1 subj2 ... subj40 (40 total)
  varLabels: arrayID memSubj ... gender (10 total)
  varMetadata: labelDescription
featureData
  featureNames: probe1 probe2 ... probe10 (10 total)
  fvarLabels: probe gene ... Reporter_Group_Name (5 total)
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation:

>
```

2 Exclude failed arrays

The meta data variable `Pass_Fail` indicates if an array is technically failed. We first should exclude these arrays.

We first check the values of the variable `Pass_Fail`:

```
> print(table(es.raw$Pass_Fail, useNA="ifany"))

pass
  40
```

If there exist failed arrays, then we exclude them:

```
> pos<-which(es.raw$Pass_Fail != "pass")
> if(length(pos))
+ {
+   es.raw<-es.raw[, -pos]
+   es.QC<-es.QC[, -pos]
+ }
```

3 Check QC probes

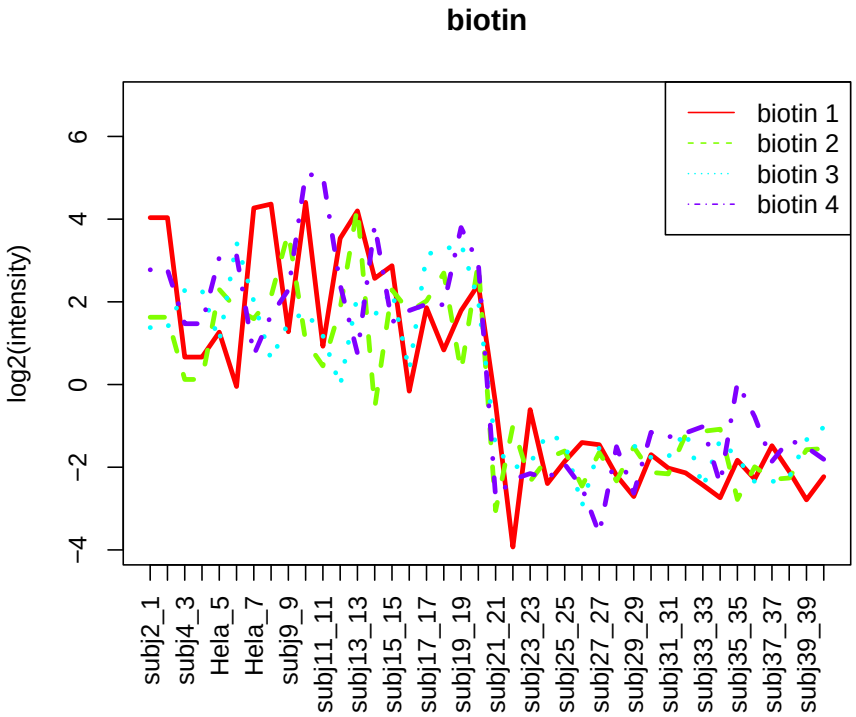
The function `plotQCCurves` shows plot of quantiles across arrays for each type of QC probes. We expect the trajectories of quantiles across arrays are horizontal lines.

To get a better view, the arrays will be sorted based on variables specified in the function argument `varSort`.

```
>      plotQCCurves(
+          esQC=es.QC,
+          probes = c("biotin"), #"cy3_hyb", "housekeeping"),
+            #"low_stringency_hyb"),
+          labelVariable = "subjID",
+          hybName = "Hybridization_Name",
+          reporterGroupName = "Reporter_Group_Name",
+          requireLog2 = FALSE,
+          projectName = "test",
+          plotOutPutFlag = FALSE,
+          cex = 1,
+          ylim = NULL,
+          xlab = "",
+          ylab = "log2(intensity)",
+          lwd = 3,
+          mar = c(10, 4, 4, 2) + 0.1,
+          las = 2,
+          cex.axis = 1,
+          sortFlag = TRUE,
+          varSort = c("Batch_Run_Date", "Chip_Barcode", "Chip_Address"),
+          timeFormat = c("%m/%d/%Y", NA, NA)
+        )

probes>>
[1] "biotin"


********** k= 1  *******
QC probe= biotin
```



4 Check squared correlations among genetic control (GC) arrays

Next, we draw heatmap of the squared correlations among GC arrays. We expect the squared correlations among GC arrays are high (> 0.90).

The function argument `labelVariable` indicates which meta variable will be used to label the arrays in the heatmap.

If we draw heatmap for replicated arrays, we can set the function arguments `sortFlag=TRUE`,

```
varSort=c("Subject_ID", "Hybridization_Name",
  "Batch_Run_Date", "Chip_Barcode", "Chip_Address")

and

timeFormat=c(NA, NA, "%m/%d/%Y", NA, NA)
```

so that arrays from the same subjects will be grouped together in the heatmap.

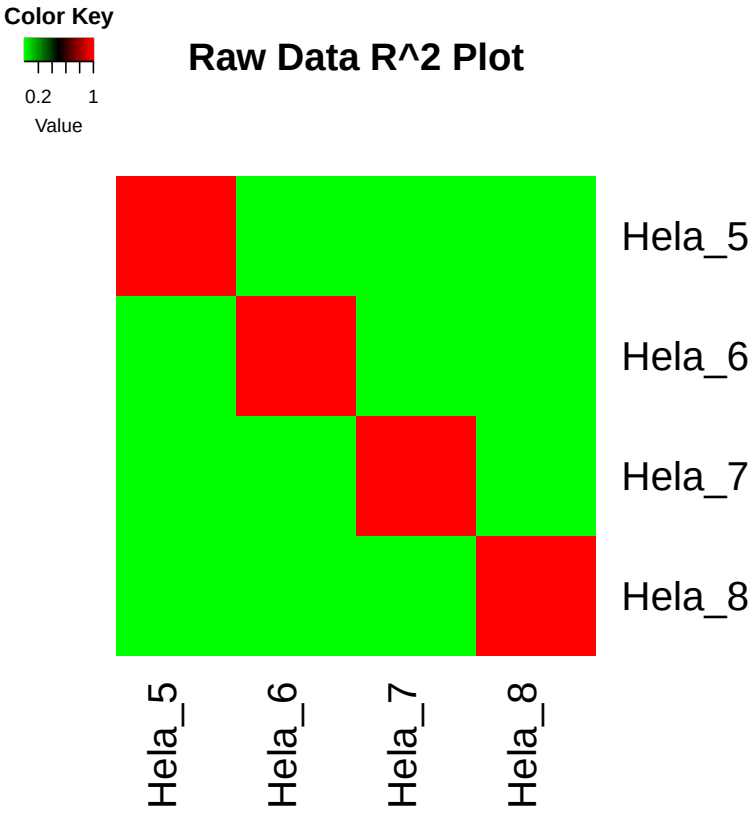
Note that although the meta variable `Batch_Run_Date` records time, it is vector of string character in R. The function `R2PlotFunc` will automatically

convert it to time variable if we set the value of the argument `timeFormat` corresponding to the variable `Batch_Run_Date` as a time format like `"%m/%d/%Y"`. Details about the time format, please see the R function `strptime`.

The followings show example R code to draw heatmap of GC arrays.

```
>      R2PlotFunc(
+          es=es.raw,
+          hybName = "Hybridization_Name",
+          arrayType = "GC",
+          GCid = c("128115", "Hela", "Brain"),
+          probs = seq(0, 1, 0.25),
+          col = gplots::greenred(75),
+          labelVariable = "subjID",
+          outFileName = "test_R2_raw.pdf",
+          title = "Raw Data R^2 Plot",
+          requireLog2 = FALSE,
+          plotOutPutFlag = FALSE,
+          las = 2,
+          keysize = 1,
+          margins = c(10, 10),
+          sortFlag = TRUE,
+          varSort=c("Batch_Run_Date", "Chip_Barcode", "Chip_Address"),
+          timeFormat=c("%m/%d/%Y", NA, NA)
+        )

quantile of R^2>>
          0%          25%          50%          75%         100%
4.807015e-06 1.015917e-03 1.675295e-03 4.155686e-03 6.798879e-03
```



5 Exclude GC arrays

We next exclude GC arrays and will focus on sample arrays to check data quality.

```
> print(table(es.raw$Tissue_Descr, useNA="ifany"))

            CD4 Human Hela Cell
             36               4

> # for different data sets, the label for GC arrays might
> # be different.
> pos.del<-which(es.raw$Tissue_Descr == "Human Hela Cell")
> cat("No. of GC arrays=", length(pos.del), "\n")

No. of GC arrays= 4

> if(length(pos.del))
+ {
+   es.raw<-es.raw[,-pos.del]
+   es.QC<-es.QC[,-pos.del]
```

```
+   print(dims(es.raw))
+   print(dims(es.QC))
+ }

         exprs
Features   100
Samples     36
         exprs
Features    10
Samples     36
```

6 Check squared correlations among replicated arrays

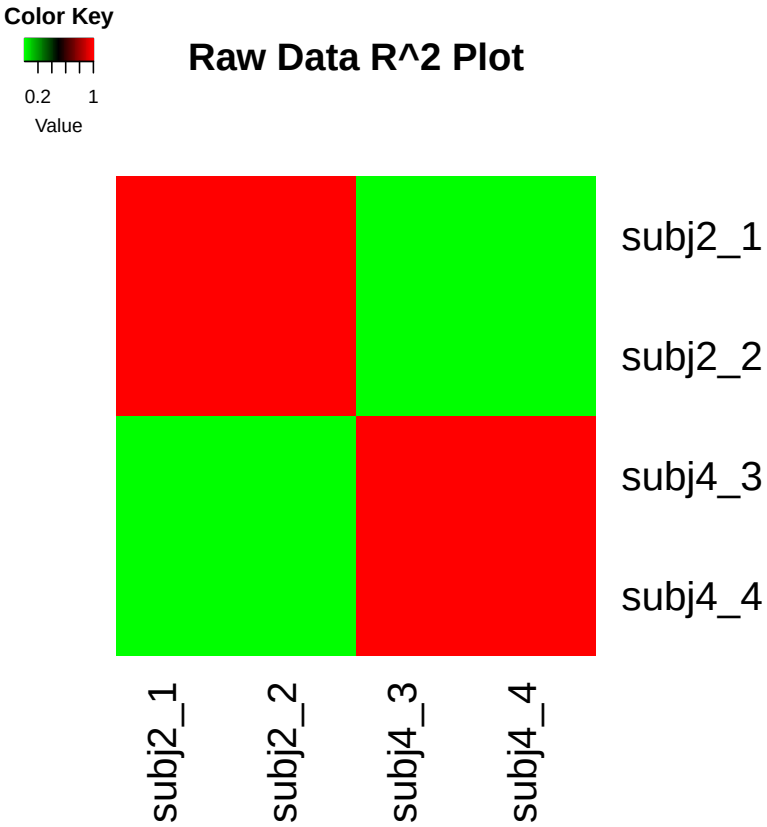
Check squared correlations among replicated arrays (excluding GC arrays). We expect within subject correlations will be high.

```
>      R2PlotFunc(
+          es=es.raw,
+          arrayType = c("replicates"),
+          GCid = c("128115", "Hela", "Brain"),
+          probs = seq(0, 1, 0.25),
+          col = gplots::greenred(75),
+          labelVariable = "subjID",
+          outFileName = "test_R2_raw.pdf",
+          title = "Raw Data R^2 Plot",
+          requireLog2 = FALSE,
+          plotOutPutFlag = FALSE,
+          las = 2,
+          keysize = 1,
+          margins = c(10, 10),
+          sortFlag = TRUE,
+          varSort=c("Subject_ID", "Hybridization_Name", "Batch_Run_Date", "Chip_Barcode",
+          timeFormat=c(NA, NA, "%m/%d/%Y", NA, NA)
+        )

quantile of R^2>>
         0%         25%         50%         75%        100%
0.007151179 0.007151179 0.007151179 0.751787795 1.000000000

quantile of within-replicate R^2>>
  0%  25%  50%  75% 100%
   1    1    1    1    1

>
```

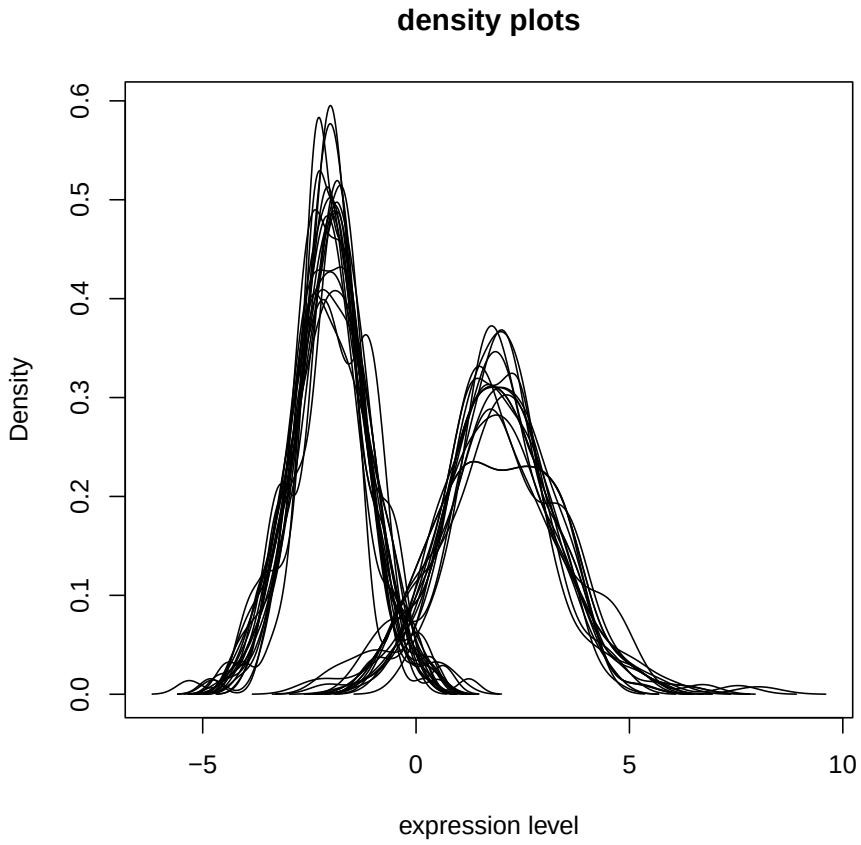


7 Obtain plot of estimated density for each array

We next draw plot of estimated density for each array. We expect the estimated densities of all arrays to be similar. However, for real data, some patterns of the estimated densities might appear indicating the existence of some batch effects.

Note that by default, the function argument `requireLog2 = TRUE`. Since the distributions of simulated data are from normal distribution, we don't need to do log2 transformation here.

```
>     densityPlots(
+       es = es.raw,
+       requireLog2 = FALSE,
+       myxlab = "expression level",
+       datExtrFunc = exprs,
+       fileFlag = FALSE,
+       fileFormat = "ps",
+       fileName = "densityPlots_sim.ps")
>
```



8 Obtain plot of quantiles across arrays

We next draw plot of quantiles across sample arrays. We expect the trajectories of quantiles be horizontal. However, for real data, some patterns of the trajectories might appear indicating the existence of some batch effects.

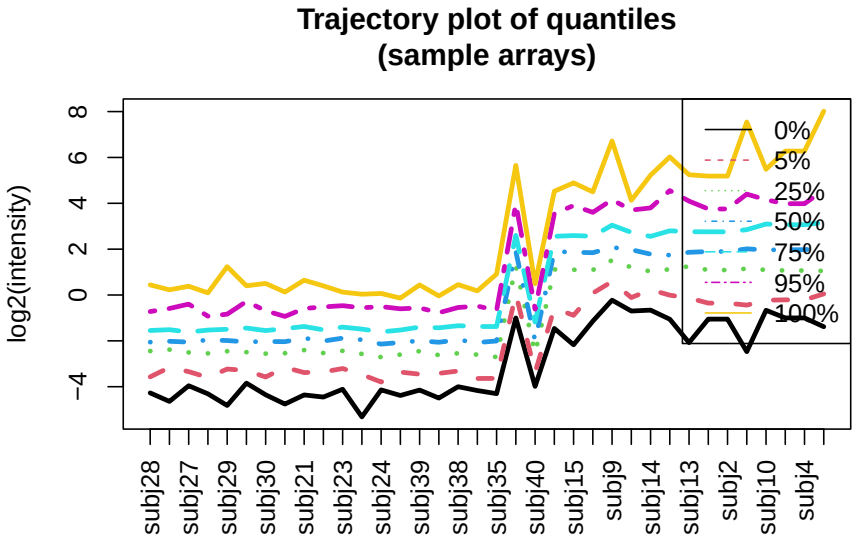
Some times, the quantile plots can show that some probes have some outlying expression levels. In this case, we can delete those gene probes.

Note that by default, the function argument `requireLog2 = TRUE`. Hence, we need to take log2 transformation to identify which gene probes containing outlying expression levels.

By default, we will sort the arrays by the ascending order of the median absolute deviation (MAD) to have a better view of the trajectories of quantiles.

```
>      quantilePlot(
+          dat=exprs(es.raw),
+          fileName,
+          probs = c(0, 0.05, 0.25, 0.5, 0.75, 0.95, 1),
+          plotOutPutFlag = FALSE,
+          requireLog2 = FALSE,
```

```
+           sortFlag = TRUE,
+           cex = 1,
+           ylim = NULL,
+           xlab = "",
+           ylab = "log2(intensity)",
+           lwd = 3,
+           main = "Trajectory plot of quantiles\n(sample arrays)",
+           mar = c(15, 4, 4, 2) + 0.1,
+           las = 2,
+           cex.axis = 1
+           )
***** Arrays were sorted by MAD (median absolute deviation)!
```



9 Exclude gene probes with outlying expression levels

if quantile plots show some outlying expression levels, we can use the following R code to identify the gene probes with outlying expression levels.

```
> # note we need to take log2 transformation
> # if requireLog2 = TRUE.
```

```
> requireLog2 = FALSE
> if(requireLog2)
+ {
+   minVec<-apply(log2(exprs(es.raw)), 1, min, na.rm=TRUE)
+   # suppose the cutoff is 0.5
+   print(sum(minVec< 0.5))
+   pos.del<-which(minVec<0.5)
+
+   cat("Number of gene probes with outlying expression levels>>",
+   length(pos.del), "\n")
+   if(length(pos.del))
+   {
+     es.raw<-es.raw[-pos.del,]
+   }
+ }
>
```

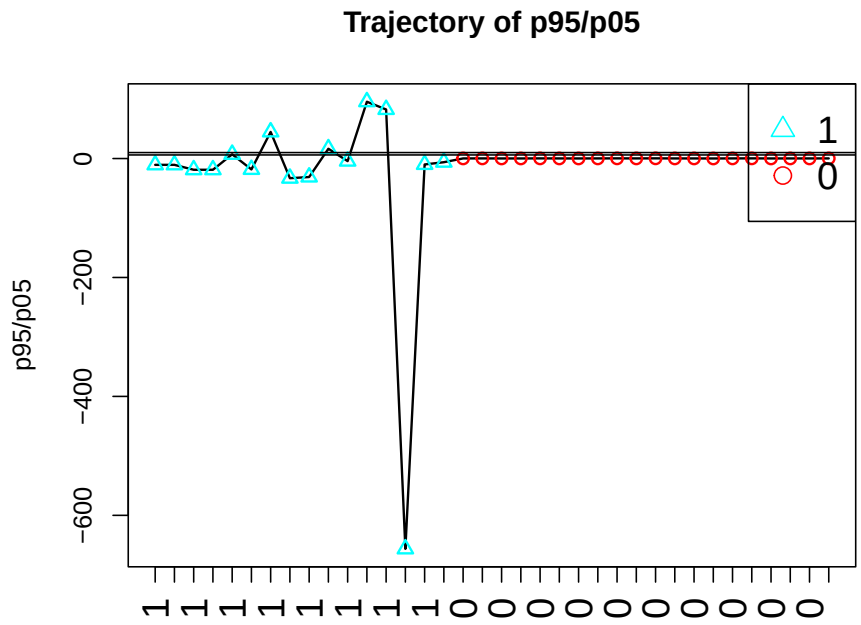
10 Obtain plot of the ratio (p_{95}/p_{05}) of 95-th percentile to 5-th percentile across arrays

We next draw the plot of the ratio of p95 over p05 across arrays, where p95 (p05) is the 95-th (5-th) percentile of a array. If an array with the ratio $p95/p05$ is less than 6, then we regard this array as a bad array and should delete it before further analysis.

Note that we should set `requireLog2 = FALSE`.

```
>      plotSamplep95p05(
+          es=es.raw,
+          labelVariable = "memSubj",
+          requireLog2 = FALSE,
+          projectName = "test",
+          plotOutPutFlag = FALSE,
+          cex = 1,
+          ylim = NULL,
+          xlab = "",
+          ylab = "",
+          lwd = 1.5,
+          mar = c(10, 4, 4, 2) + 0.1,
+          las = 2,
+          cex.axis=1.5,
+          title = "Trajectory of p95/p05",
+          cex.legend = 1.5,
+          cex.lab = 1.5,
+          legendPosition = "topright",
+          cut1 = 10,
+          cut2 = 6,
+          sortFlag = TRUE,
+          varSort = c("Batch_Run_Date", "Chip_Barcode", "Chip_Address"),
```

```
+           timeFormat = c("%m/%d/%Y", NA, NA),
+           verbose = FALSE)
```



11 Exclude arrays with $p_{95}/p_{05} \leq 6$

If there exist arrays with $p95/p05 < 6$, we then need to exclude these arrays from further data analysis. The followings are example R code:

```
> p95<-quantile(exprs(es.raw), prob=0.95)
> p05<-quantile(exprs(es.raw), prob=0.05)
> r<-p95/p05
> pos.del<-which(r<6)
> print(pos.del)

95%
  1

> if(length(pos.del))
+ {
+   es.raw<-es.raw[,-pos.del]
```

```
+   es.QC<-es.QC[,-pos.del]
+ }
>
```

12 Obtain Plot of principal components

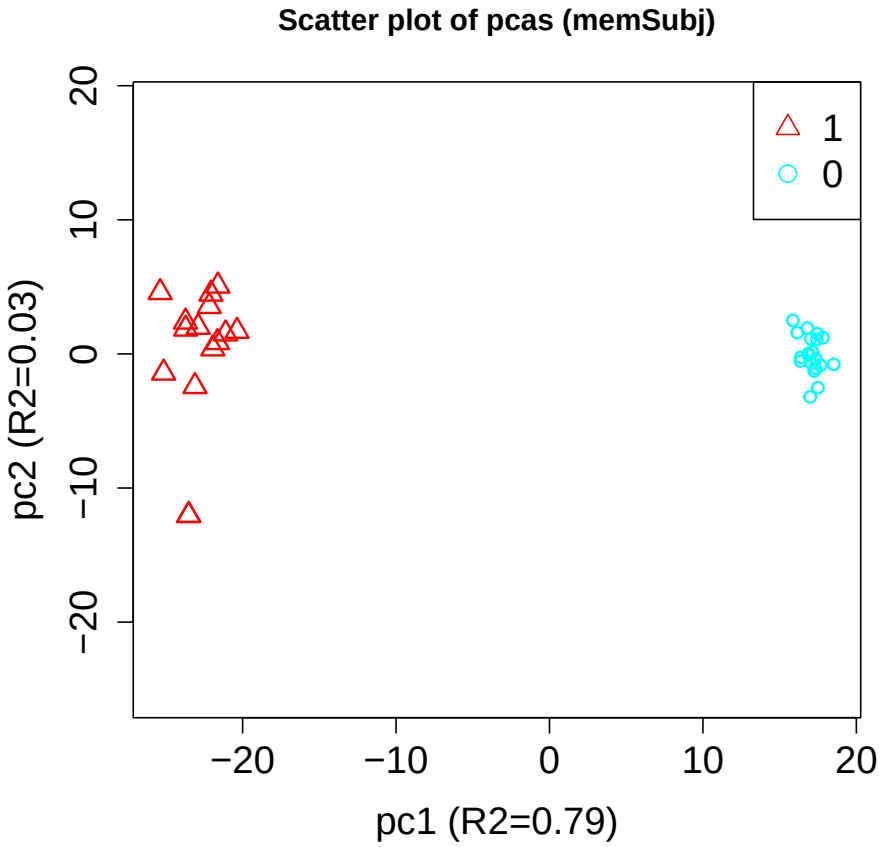
We next draw pca plots to double check batch effects or treatment effects indicated by dendrogram.

The first step is to obtain principal components using the function `getPCAFunc`. For large data set, this function might be very slow.

```
>      pcaObj<-getPCAFunc(es=es.raw,
+                 labelVariable = "subjID",
+                 requireLog2 = FALSE,
+                 corFlag = FALSE
+
+
+      )
>
```

We then plot the first 2 or 3 principal components and label the data points by meta variables of interests, such as tissue type, study center, batch id, etc..

```
>      pca2DPlot(pcaObj=pcaObj,
+                plot.dim = c(1,2),
+                labelVariable = "memSubj",
+                outFileName = "test_pca_raw.pdf",
+                title = "Scatter plot of pcas (memSubj)",
+                plotOutPutFlag = FALSE,
+                mar = c(5, 4, 4, 2) + 0.1,
+                lwd = 1.5,
+                equalRange = TRUE,
+                xlab = NULL,
+                ylab = NULL,
+                xlim = NULL,
+                ylim = NULL,
+                cex.legend = 1.5,
+                cex = 1.5,
+                cex.lab = 1.5,
+                cex.axis = 1.5,
+                legendPosition = "topright"
+
+                )
```



13 Perform background correction, data transformation and normalization

```
> tt <- es.raw
> es.q<-lumiN(tt, method="quantile")

Perform quantile normalization ...
```

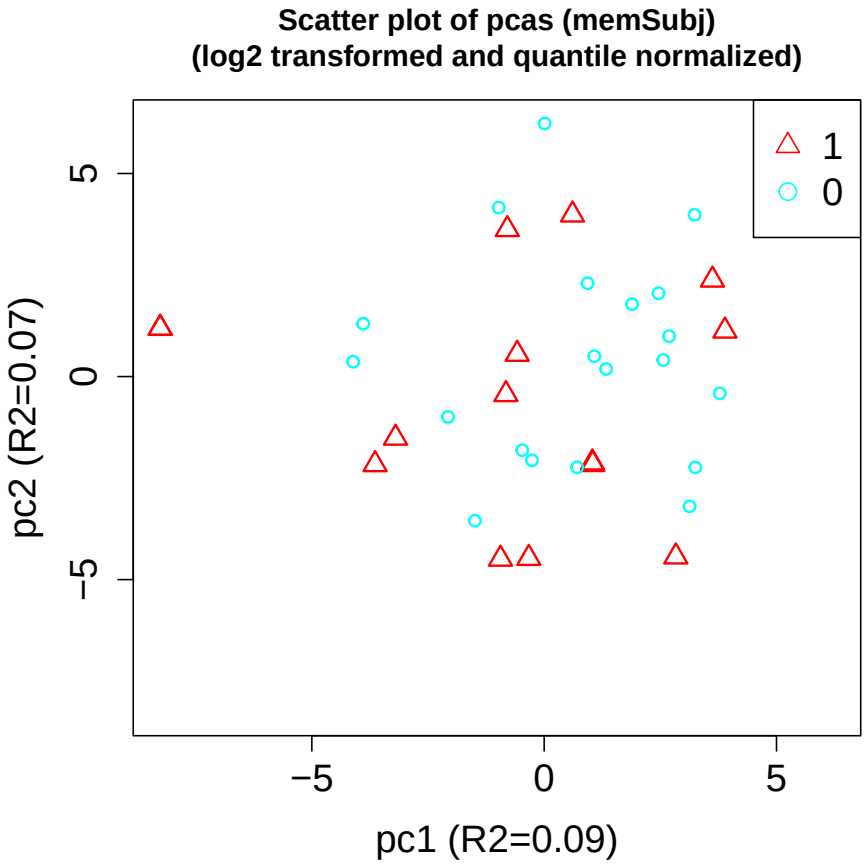
14 Obtain Plot of principal components for pre-processed data

After pre-processing data, we do principal component analysis again.

Note that we should set `requireLog2 = FALSE`.

```
>      pcaObj<-getPCAFunc(es=es.q,
+                 labelVariable = "subjID",
+                 requireLog2 = FALSE,
+                 corFlag = FALSE
```

```
+
+     )
>     pca2DPlot(pcaObj=pcaObj,
+               plot.dim = c(1,2),
+               labelVariable = "memSubj",
+               outFileName = "test_pca_raw.pdf",
+               title = "Scatter plot of pcas (memSubj)\n(log2 transformed and quantile n
+               plotOutPutFlag = FALSE,
+               mar = c(5, 4, 4, 2) + 0.1,
+               lwd = 1.5,
+               equalRange = TRUE,
+               xlab = NULL,
+               ylab = NULL,
+               xlim = NULL,
+               ylim = NULL,
+               cex.legend = 1.5,
+               cex = 1.5,
+               cex.lab = 1.5,
+               cex.axis = 1.5,
+               legendPosition = "topright"
+               )
>
```



15 Incorporate phenotype data

In addition meta data, we usually have phenotype data to describe subjects. We can now add them in.

16 Data analysis

16.1 lmFitWrapper and lmFitPaired

`iCheck` provide 2 limma wrapper functions `lmFitPaired` (for paired data) and `lmFitWrapper` (for unpaired data).

Note that the function argument `pos.var.interest = 1` requests the results (test statistic and p-value) for the first covariate will be print out.

If `pos.var.interest = 0`, then the results (test statistic and p-value) for the intercept will be print out.

The outcome variable must be gene probes. Can not be phenotype variables.

```
> res.limma=lmFitWrapper(
+   es=es.q,
```

```
+   formula=~as.factor(memSubj),
+   pos.var.interest = 1,
+   pvalAdjMethod="fdr",
+   alpha=0.05,
+   probeID.var="probe",
+   gene.var="gene",
+   chr.var="chr",
+   verbose=TRUE)

dim(dat)>>
[1] 100  35

Running lmFit...
Running eBayes...
Preparing output...
   probeIDs geneSymbols chr     stats        pval     p.adj pos
1   probe29      gene29   1 -3.376879 0.001401733 0.1005857  19
2   probe16      gene16   1  3.254099 0.002011714 0.1005857   6
3   probe92      gene92   1  2.688895 0.009634725 0.3211575  82
4   probe59      gene59   1 -2.210193 0.031558408 0.4968483  49
5   probe32      gene32   1  2.181275 0.033750422 0.4968483  22
6   probe17      gene17   1 -2.143605 0.036806038 0.4968483   7
7  probe103     gene103   1 -2.117636 0.039051322 0.4968483  93
8   probe35      gene35   1 -2.109845 0.039747866 0.4968483  25
9   probe54      gene54   1  1.712229 0.092867773 0.8375408  44
10  probe74      gene74   1 -1.703176 0.094560774 0.8375408  64
11  probe40      gene40   1  1.638282 0.107456092 0.8375408  30
12  probe61      gene61   1 -1.604613 0.114691882 0.8375408  51
13  probe12      gene12   1  1.527377 0.132783363 0.8375408   2
14  probe56      gene56   1 -1.521438 0.134263881 0.8375408  46
15  probe90      gene90   1 -1.445861 0.154271925 0.8375408  80
16  probe79      gene79   1 -1.406303 0.165637004 0.8375408  69
17  probe89      gene89   1 -1.367624 0.177366059 0.8375408  79
18  probe38      gene38   1  1.360555 0.179576911 0.8375408  28
19  probe96      gene96   1 -1.351534 0.182428796 0.8375408  86
20  probe23      gene23   1  1.347172 0.183820114 0.8375408  13

pvalue quantiles for intercept and covariates>>
        (Intercept) as.factor(memSubj)1
min    0.0002637651         0.001401733
25%    0.0356690890         0.242102862
median 0.1922335675         0.455084256
75%    0.4874162700         0.780677827
max    0.9978539142         0.998200667

formula>>
~as.factor(memSubj)

covariate of interest is  as.factor(memSubj)
Number of tests= 100
```

```
Number of arrays= 35
Number of significant tests (raw p-value <  0.05 )= 8
Number of significant tests after p-value adjustments= 0


**********************************************
No genes are differentially expressed!

>
```

16.2 glmWrapper

outcome variable can be phenotype variables. The function argument `family` indicates if logistic regression (`family=binomial`) used or general linear regression (`family=gaussian`) used.

```
> res.glm=glmWrapper(
+   es=es.q,
+   formula = xi~as.factor(memSubj),
+   pos.var.interest = 1,
+   family=gaussian,
+   logit=FALSE,
+   pvalAdjMethod="fdr",
+   alpha = 0.05,
+   probeID.var = "probe",
+   gene.var = "gene",
+   chr.var = "chr",
+   applier=lapply,
+   verbose=TRUE
+   )
```

	probeIDs	geneSymbols	chr	stats	coef	pval	p.adj	pos
1	probe16	gene16	1	3.305320	1.0593407	0.002293003	0.1763387	6
2	probe29	gene29	1	-3.142649	-1.3263843	0.003526774	0.1763387	19
3	probe92	gene92	1	2.508446	1.0489929	0.017219384	0.5739795	82
4	probe59	gene59	1	-2.238212	-0.7231024	0.032072229	0.6917895	49
5	probe35	gene35	1	-2.173225	-0.6718451	0.037042472	0.6917895	25
6	probe103	gene103	1	-2.121164	-0.7059218	0.041507370	0.6917895	93
7	probe17	gene17	1	-2.029732	-0.8040525	0.050510402	0.7215772	7
8	probe32	gene32	1	1.915186	1.0631079	0.064169227	0.8021153	22
9	probe74	gene74	1	-1.786253	-0.5282879	0.083249917	0.8072836	64
10	probe61	gene61	1	-1.725751	-0.4815208	0.093746106	0.8072836	51
11	probe54	gene54	1	1.633937	0.6301915	0.111777893	0.8072836	44
12	probe40	gene40	1	1.609611	0.5661679	0.117008899	0.8072836	30
13	probe90	gene90	1	-1.605115	-0.4182303	0.117997257	0.8072836	80
14	probe38	gene38	1	1.542007	0.3851935	0.132607854	0.8072836	28
15	probe56	gene56	1	-1.464652	-0.5487800	0.152478453	0.8072836	46
16	probe99	gene99	1	1.405098	0.3951053	0.169336626	0.8072836	89
17	probe108	gene108	1	-1.380558	-0.3907145	0.176695014	0.8072836	98
18	probe79	gene79	1	-1.339224	-0.5201586	0.189649429	0.8072836	69
19	probe12	gene12	1	1.333659	0.7656150	0.191448222	0.8072836	2

```
20  probe44      gene44   1  1.304127  0.3796869 0.201213900 0.8072836  34

pvalue quantiles for intercept and covariates>>
       pval.(Intercept) pval.as.factor(memSubj)1
min        0.0006238292              0.002293003
25%        0.0245832240              0.237742030
median     0.1952130722              0.444338213
75%        0.5048805444              0.784209263
max        0.9980082033              0.998167350

formula>>
xi ~ as.factor(memSubj)

covariate of interest is  as.factor(memSubj)
Number of tests= 100
Number of arrays= 35
Number of significant tests (raw p-value <  0.05 )= 6
Number of significant tests after p-value adjustments= 0


**********************************************
No genes are differentially expressed!

>
```

16.3 lkhrWrapper

Likelihood ratio test wrapper. Compare 2 glm models. One is reduced model. The other is full model.

```
> res.lkh=lkhrWrapper(
+   es=es.q,
+   formulaReduced = xi~as.factor(memSubj),
+   formulaFull = xi~as.factor(memSubj)+gender,
+   family=gaussian,
+   logit=FALSE,
+   pvalAdjMethod="fdr",
+   alpha = 0.05,
+   probeID.var = "probe",
+   gene.var = "gene",
+   chr.var = "chr",
+   applier=lapply,
+   verbose=TRUE
+   )

********************************

Top  20  tests>>>
   probeIDs geneSymbols chr    Chisq Df        pval     p.adj pos
20  probe30      gene30   1 7.665514  1 0.005628622 0.4115445  20
34  probe44      gene44   1 6.674654  1 0.009779351 0.4115445  34
```

```
70  probe80      gene80   1 5.787806  1 0.016137715 0.4115445  70
36  probe46      gene46   1 5.697967  1 0.016984571 0.4115445  36
5   probe15      gene15   1 5.362249  1 0.020577223 0.4115445   5
78  probe88      gene88   1 4.471478  1 0.034465173 0.5722795  78
42  probe52      gene52   1 4.087700  1 0.043196384 0.5722795  42
73  probe83      gene83   1 3.553840  1 0.059407808 0.5722795  73
33  probe43      gene43   1 3.402508  1 0.065097366 0.5722795  33
86  probe96      gene96   1 3.352279  1 0.067112049 0.5722795  86
38  probe48      gene48   1 3.179674  1 0.074559584 0.5722795  38
22  probe32      gene32   1 3.061740  1 0.080155786 0.5722795  22
29  probe39      gene39   1 2.909647  1 0.088051166 0.5722795  29
94 probe104     gene104   1 2.901516  1 0.088496307 0.5722795  94
4   probe14      gene14   1 2.851523  1 0.091287395 0.5722795   4
26  probe36      gene36   1 2.846647  1 0.091564723 0.5722795  26
98 probe108     gene108   1 2.738579  1 0.097951851 0.5761874  98
52  probe62      gene62   1 2.504063  1 0.113553018 0.6308501  52
8   probe18      gene18   1 2.171881  1 0.140554180 0.6973549   8
75  probe85      gene85   1 2.156022  1 0.142011926 0.6973549  75


formulaReduced>>
xi ~ as.factor(memSubj)

formulaFull>>
xi ~ as.factor(memSubj) + gender


Number of tests>>> 100

Number of arrays>>> 35

Number of tests with pvalue<0.05>>> 7

Number of tests with FDR adjusted pvalue<0.05>>> 0

>
```

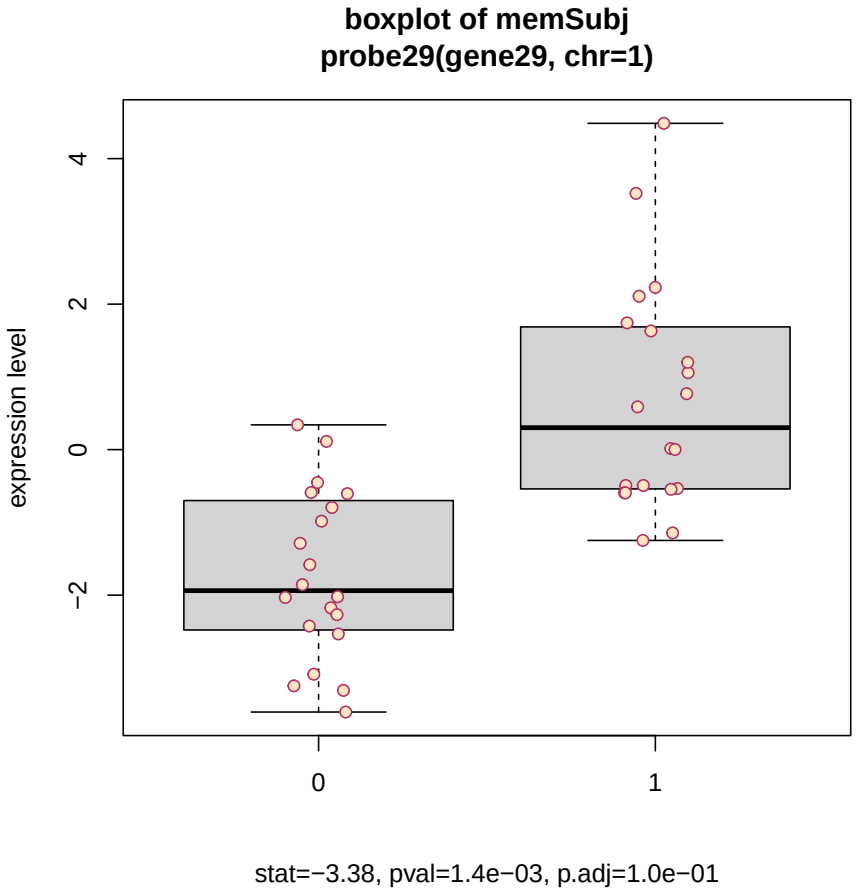
17 Result Visualization

Once we get analysis results, we need to check if the results are reasonable or not (e.g,were results affected by outliers?).

If the phenotype variable of interest is a binary type variable, then we can draw parallel boxplots of expression level versus the phenotype for each of top results. `iCheck` provides function `boxPlots` to do such a task.

```
>     boxPlots(
+       resFrame = res.limma$frame,
+       es = es.sim,
+       col.resFrame = c("probeIDs", "stats", "pval", "p.adj"),
+       var.pheno = "memSubj",
```

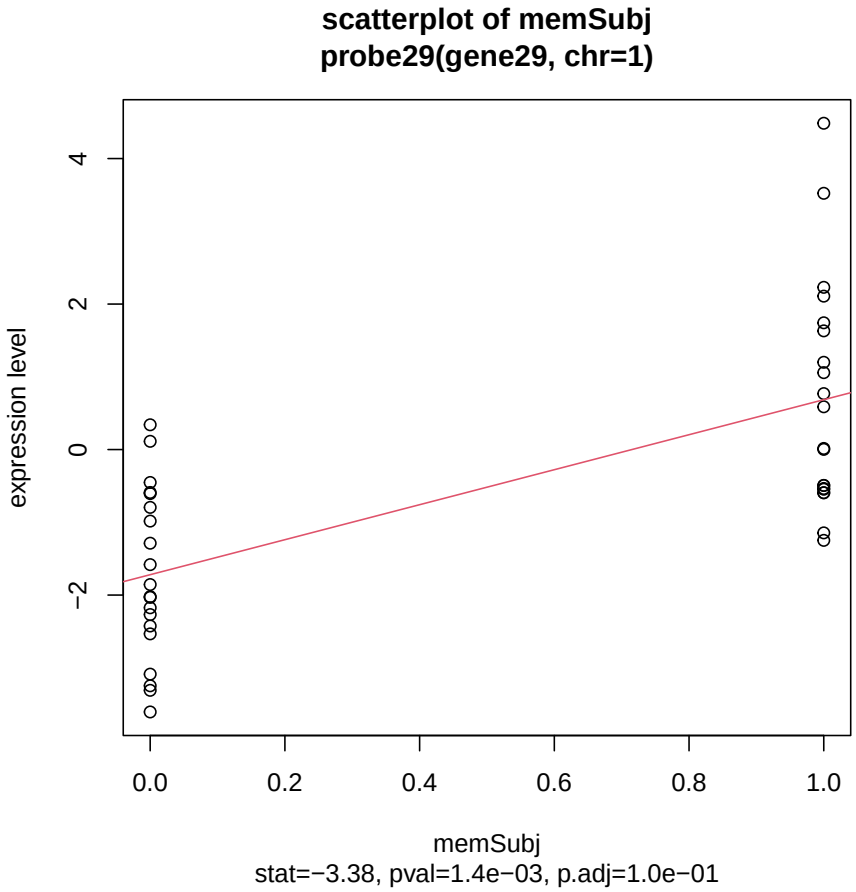
```
+        var.probe = "probe",
+        var.gene = "gene",
+        var.chr = "chr",
+        nTop = 20,
+        myylab = "expression level",
+        datExtrFunc = exprs,
+        fileFlag = FALSE,
+        fileFormat = "ps",
+        fileName = "boxPlots_sim.ps")
>
```



If the phenotype variable of interest is a continous type variable, then we can draw scatter plot of expression level versus the phenotype for each of top results. `iCheck` provides function `scatterPlots` to do such a task.

```
>     # regard memSubj as continuos for illustration purpose
>     scatterPlots(
+       resFrame = res.limma$frame,
+       es = es.sim,
+       col.resFrame = c("probeIDs", "stats", "pval", "p.adj"),
+       var.pheno = "memSubj",
+       var.probe = "probe",
```

```
+         var.gene = "gene",
+         var.chr = "chr",
+         nTop = 20,
+         myylab = "expression level",
+         datExtrFunc = exprs,
+         fileFlag = FALSE,
+         fileFormat = "ps",
+         fileName = "scatterPlots_sim.ps")
>
```



18 Session Info

Finally, we need to print out the session info so that later we can know which versions the packages are from.

```
> toLatex(sessionInfo())
```

- R version 4.1.0 (2021-05-18), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_GB`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`,

LC_PAPER=en_US.UTF-8, LC_NAME=C, LC_ADDRESS=C, LC_TELEPHONE=C, LC_MEASUREMENT=en_US.UTF-8, LC_IDENTIFICATION=C

- Running under: Ubuntu 20.04.2 LTS
- Matrix products: default
- BLAS: /home/biocbuild/bbs-3.13-bioc/R/lib/libRblas.so
- LAPACK: /home/biocbuild/bbs-3.13-bioc/R/lib/libRlapack.so
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, utils
- Other packages: Biobase 2.52.0, BiocGenerics 0.38.0, gplots 3.1.1, iCheck 1.22.0, lumi 2.44.0
- Loaded via a namespace (and not attached): AnnotationDbi 1.54.0, BiocFileCache 2.0.0, BiocIO 1.2.0, BiocManager 1.30.15, BiocParallel 1.26.0, Biostrings 2.60.0, DBI 1.1.1, DelayedArray 0.18.0, DelayedMatrixStats 1.14.0, GEOquery 2.60.0, GeneSelectMMD 2.36.0, GenomeInfoDb 1.28.0, GenomeInfoDbData 1.2.6, GenomicAlignments 1.28.0, GenomicFeatures 1.44.0, GenomicRanges 1.44.0, HDF5Array 1.20.0, IRanges 2.26.0, KEGGREST 1.32.0, KernSmooth 2.23-20, MASS 7.3-54, Matrix 1.3-3, MatrixGenerics 1.4.0, R6 2.5.0, RColorBrewer 1.1-2, RCurl 1.98-1.3, RSQLite 2.2.7, Rcpp 1.0.6, Rhdf5lib 1.14.0, Rsamtools 2.8.0, Rttf2pt1 1.3.8, S4Vectors 0.30.0, SummarizedExperiment 1.22.0, XML 3.99-0.6, XVector 0.32.0, affy 1.70.0, affyio 1.62.0, annotate 1.70.0, askpass 1.1, assertthat 0.2.1, base64 2.0, beanplot 1.2, biomaRt 2.48.0, bit 4.0.4, bit64 4.0.5, bitops 1.0-7, blob 1.2.1, bumphunter 1.34.0, caTools 1.18.2, cachem 1.0.5, codetools 0.2-18, colorspace 2.0-1, compiler 4.1.0, crayon 1.4.1, crosstalk 1.1.1, curl 4.3.1, data.table 1.14.0, dbplyr 2.1.1, digest 0.6.27, doRNG 1.8.2, dplyr 1.0.6, ellipsis 0.3.2, extrafont 0.17, extrafontdb 1.0, fansi 0.4.2, fastmap 1.1.0, filelock 1.0.2, foreach 1.5.1, fs 1.5.0, genefilter 1.74.0, generics 0.1.0, glue 1.4.2, grid 4.1.0, gtools 3.8.2, hms 1.1.0, htmltools 0.5.1.1, htmlwidgets 1.5.3, httpuv 1.6.1, httr 1.4.2, illuminaio 0.34.0, iterators 1.0.13, jsonlite 1.7.2, knitr 1.33, later 1.2.0, lattice 0.20-44, lifecycle 1.0.0, limma 3.48.0, lmtest 0.9-38, locfit 1.5-9.4, magrittr 2.0.1, manipulateWidget 0.10.1, matrixStats 0.58.0, mclust 5.4.7, memoise 2.0.0, methylumi 2.38.0, mgcv 1.8-35, mime 0.10, minfi 1.38.0, miniUI 0.1.1.1, multtest 2.48.0, nleqslv 3.3.2, nlme 3.1-152, nor1mix 1.3-0, openssl 1.4.4, pillar 1.6.1, pkgconfig 2.0.3, pkgdown 1.6.1, plyr 1.8.6, png 0.1-7, preprocessCore 1.54.0, prettyunits 1.1.1, progress 1.2.2, promises 1.2.0.1, purrr 0.3.4, quadprog 1.5-8, randomForest 4.6-14, rappdirs 0.3.3, readr 1.4.0, reshape 0.8.8, restfulr 0.0.13, rgl 0.106.8, rhdf5 2.36.0, rhdf5filters 1.4.0, rjson 0.2.20, rlang 0.4.11, rngtools 1.5, rstudioapi 0.13, rtracklayer 1.52.0, scatterplot3d 0.3-41, scrime 1.3.5, shiny 1.6.0, siggenes 1.66.0, sparseMatrixStats 1.4.0, splines 4.1.0, stats4 4.1.0, stringi 1.6.2, stringr 1.4.0, survival 3.2-11, tibble 3.1.2, tidyr 1.1.3, tidyselect 1.1.1, tools 4.1.0, utf8 1.2.1, vctrs 0.3.8, webshot 0.5.2, xfun 0.23, xml2 1.3.2, xtable 1.8-4, yaml 2.2.1, zlibbioc 1.38.0, zoo 1.8-9