ShinyDataSHIELD User's Guide

Escribà Montagut, Xavier; González, Juan R.

2022-01-17

Contents

Chapter 1

Overview

ShinyDataSHIELD is a non-disclosive data analysis toolbox powered by DataSHIELD with the following features:

- Descriptive statistics: Summary, scatter plots, histograms and heatmaps of table variables.
- Statistic models: GLM and GLMer model fittings
- Omic analysis: GWAS, LIMMA, ... using different types of resources (VCF files, PLINK, RSE, eSets)

The features available on ShinyDataSHIELD are powered by different packages of the DataSHIELD project (dsBaseClient and dsOmicsClient), it uses them in a seamless way so the final user of ShinyDataSHIELD can perform all the included studies without writing a single line of code and get all the resulting figures and tables by the click of a button.

a

Chapter 2

Omic data analysis: types of implemented analyses

The Figure 2.1 describes the different types of 'omic association analyses that can be performed using DataSHIELD client functions implemented in the *dsOmicsClient* package. Basically, data ('omic and phenotypes/covariates) can be stored in different sites (http, ssh, AWS S3, local, …) and are managed with Opal through the *resourcer* package and their extensions implemented in *dsOmics*.

Then, `dsOmicsClient` package allows different types of analyses: pooled and meta-analysis. Both methods are based on fitting different Generalized Linear Models (GLMs) for each feature when assesing association between 'omic data and the phenotype/trait/condition of interest. Of course, non-disclosive 'omic data analysis from a single study can also be performed.

The **pooled approach** (Figure 2.2) is recommended when the user wants to analyze 'omic data from different sources and obtain results as if the data were located in a single computer. It should be noted that this can be very time consuming when analyzing multiple features since it calls a base function in DataSHIELD (`ds.glm`) repeatedly. It also cannot be recommended when data are not properly harmonized (e.g. gene expression normalized using different methods, GWAS data having different platforms, …). Furthermore when it is necesary to remove unwanted variability (for transcriptomic and epigenomica analysis) or control for population stratification (for GWAS analysis), this approach cannot be used since we need to develop methods to compute surrogate variables (to remove unwanted variability) or PCAs (to to address population stratification) in a non-disclosive way.

The **meta-analysis approach** Figure 2.3 overcomes the limitations raised when performing pooled analyses. First, the computation issue is addressed

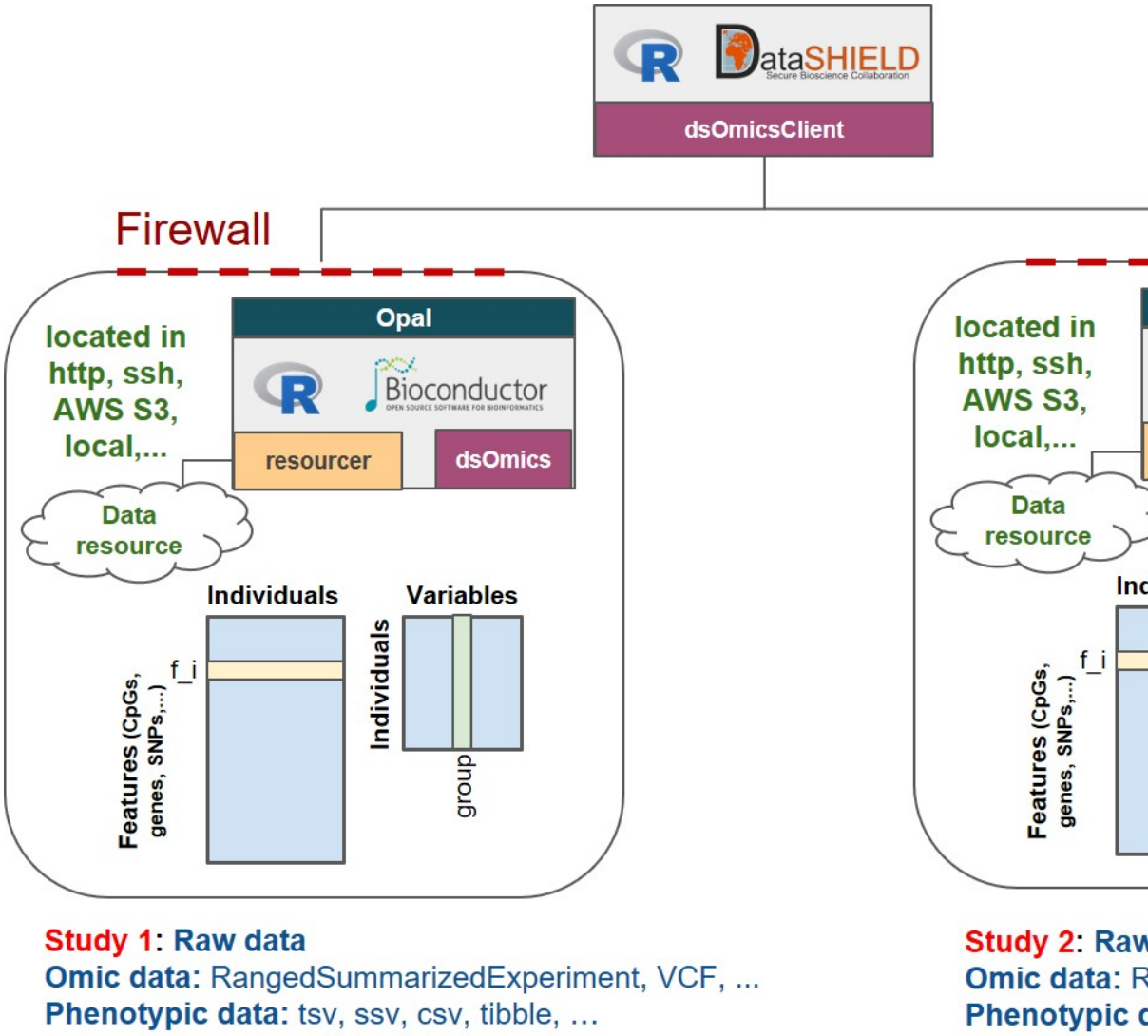


Figure 2.1: Non-disclosive omic data analysis with DataSHIELD and Bioconductor. The figure illustrates how the 'resourcer' package is used to get access to omic data through the Opal servers. Then DataSHIELD is used in the client side to perform non-disclosive data analyses.

by using scalable and fast methods to perform data analysis at whole-genome level at each location The transcriptomic and epigenomic data analyses make use of the widely used *limma* package that uses `ExpressionSet` or `RangedSummarizedExperiment` Bioc infrastructures to deal with 'omic and phenotypic (e.g covariates). The genomic data are analyzed using *GWASTools* and *GENESIS* that are designed to perform quality control (QC) and GWAS using GDS infrastructure.

Next, we describe how both approaches are implemented:

- **Pooled approach:** Figure 2.2 illustrate how this analysis is performed. This corresponds to generalized linear models (glm) on data from single or multiple sources. It makes use of `ds.glm()` function which is a DataSHIELD function that uses an approach that is mathematically equivalent to placing all individual-level data froma all sources in one central warehouse and analysing those data using the conventional `glm()` function in R. The user can select one (or multiple) features (i.e., genes, transcripts, CpGs, SNPs, …)
- **Meta-analysis:** Figure 2.3 illustrate how this analysis is performed. This corresponds to performing a genome-wide analysis at each location using functions that are specifically design for that purpose and that are scalable. Then the results from each location can be meta-analyzed using methods that meta-analyze either effect sizes or p-values.

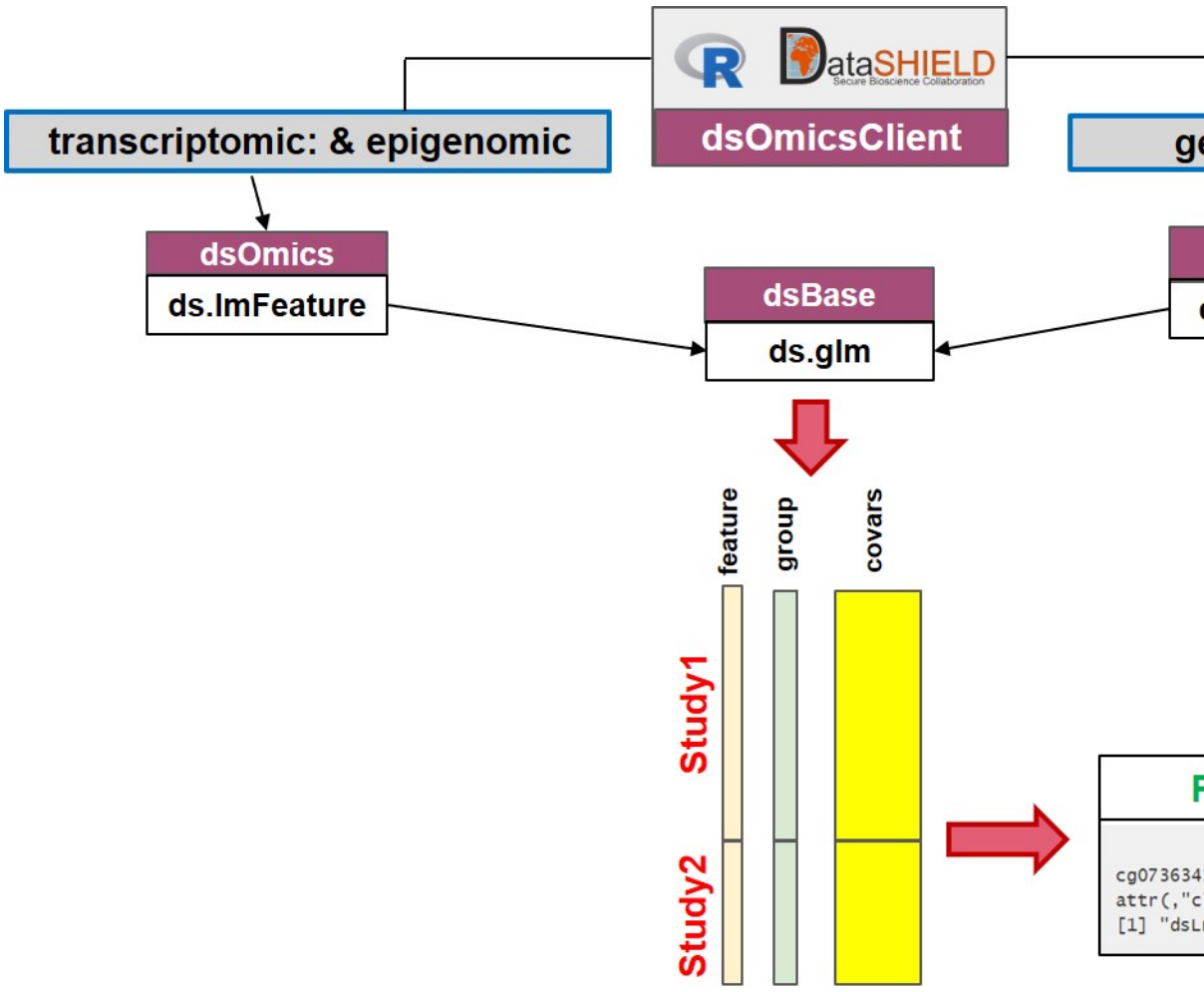


Figure 2.2: Non-disclosive omic data analysis with DataSHIELD and Bioconductor. The figure illustrates how to perform single pooled omic data analysis. The analyses are performed by using a generalized linear model (glm) on data from one or multiple sources. It makes use of 'ds.glm()', a DataSHIELD function, that uses an approach that is mathematically equivalent to placing all individual-level data from all sources in one central warehouse and analysing those data using the conventional 'glm()' function in R.

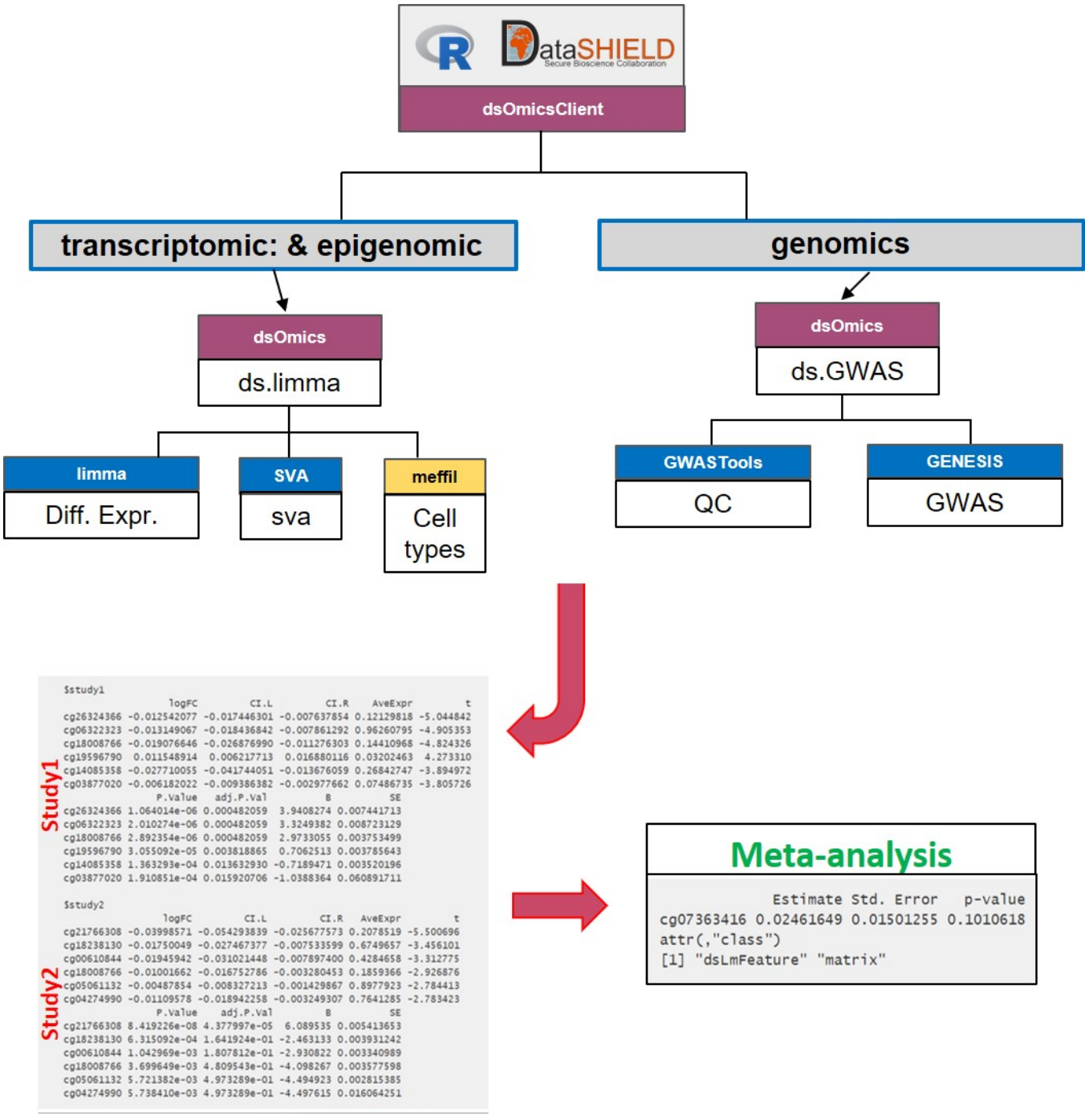


Figure 2.3: Non-disclosive omic data analysis with DataSHIELD and Bioconductor. The figure illustrates how to perform anlyses at genome-wide level from one or multiple sources. It runs standard Bioconductor functions at each server independently to speed up the analyses and in the case of having multiple sources, results can be meta-analyzed uning standar R functions.

Chapter 3

Differential gene expression (DGE) analysis

Let us illustrate how to perform transcriptomic data analysis using data from TCGA project. We have uploaded to the opal server a resource called `tcga_liver` whose URL is http://duffel.rail.bio/recount/TCGA/rse_gene_liver.Rdata which is available through the recount project. This resource contains the `RangeSummarizedExperiment` with the RNAseq profiling of liver cancer data from TCGA. Next, we illustrate how a differential expression analysis to compare RNAseq profiling of women vs men (variable `gdc_cases.demographic.gender`). The DGE analysis is normally performed using *limma* package. In that case, as we are analyzing RNA-seq data, `limma + voom` method will be required.

Let us start by creating the connection to the opal server:

```
builder <- newDSLoginBuilder()
builder$append(server = "study1", url = "https://opal-demo.obiba.org",
               user = "dsuser", password = "P@ssw0rd",
               resource = "RSRC.tcga_liver", profile = "omics")

logindata <- builder$build()

conns <- datashield.login(logins = logindata, assign = TRUE,
                          symbol = "res")
```

```
##
## Logging into the collaborating servers

##
## Assigning resource data...
```

Then, let us coerce the resource to a `RangedSummarizedExperiment` which is the type of object that is available in the recount project.

```
datashield.assign.expr(conns, symbol = "rse",
                       expr = quote(as.resource.object(res)))
ds.class("rse")
```

```
## $study1
## [1] "RangedSummarizedExperiment"
## attr(,"package")
## [1] "SummarizedExperiment"
```

The number of features and samples can be inspected by

```
ds.dim("rse")
```

```
## $`dimensions of rse in study1`
## [1] 58037   424
##
## $`dimensions of rse in combined studies`
## [1] 58037   424
```

And the names of the features using the same function used in the case of analyzing an `ExpressionSet`

```
name.features <- ds.featureNames("rse")
lapply(name.features, head)
```

```
## $study1
## [1] "ENSG00000000003.14" "ENSG00000000005.5"  "ENSG00000000419.12"
## [4] "ENSG00000000457.13" "ENSG00000000460.16" "ENSG00000000938.12"
```

Also the covariate names can be inspected by

```
name.vars <- ds.featureData("rse")
lapply(name.vars, head, n=15)
```

```
## $study1
##  [1] "project"
##  [2] "sample"
##  [3] "experiment"
##  [4] "run"
##  [5] "read_count_as_reported_by_sra"
##  [6] "reads_downloaded"
##  [7] "proportion_of_reads_reported_by_sra_downloaded"
##  [8] "paired_end"
##  [9] "sra_misreported_paired_end"
## [10] "mapped_read_count"
## [11] "auc"
## [12] "sharq_beta_tissue"
```

```
## [13] "sharq_beta_cell_type"
## [14] "biosample_submission_date"
## [15] "biosample_publication_date"
```

We can visualize the levels of the variable having gender information that will be our condition (i.e., we are interested in obtaining genes that are differentially expressed between males and females)

```
ds.table1D("rse$gdc_cases.demographic.gender")
```

```
## Warning: 'ds.table1D' is deprecated.
## Use 'ds.table' instead.
## See help("Deprecated")
```

```
## $counts
##        rse$gdc_cases.demographic.gender
## female                              143
## male                                281
## Total                               424
##
## $percentages
##        rse$gdc_cases.demographic.gender
## female                            33.73
## male                              66.27
## Total                            100.00
##
## $validity
## [1] "All tables are valid!"
```

We have implemented a function called `ds.RNAseqPreproc()` to perform RNAseq data pre-processing that includes:

- transforming data into log2 CPM units
- filtering lowly-expressed genes
- data normalization

```
ds.RNAseqPreproc('rse', group= 'gdc_cases.demographic.gender',
                 newobj.name = 'rse.pre')
```

Note that it is recommended to indicate the grouping variable (i.e., condition). Once data have been pre-processed, we can perform differential expression analysis. Notice how dimensions have changed given the fact that we have removed genes with low expression which are expected to do not be differentially expressed.

```
ds.dim('rse')
```

```
## $`dimensions of rse in study1`
## [1] 58037   424
##
```

```
## $`dimensions of rse in combined studies`
## [1] 58037   424
```

```
ds.dim('rse.pre')
```

```
## $`dimensions of rse.pre in study1`
## [1] 40363   424
##
## $`dimensions of rse.pre in combined studies`
## [1] 40363   424
```

The differential expression analysis is ´dsOmicsClient/dsOmics´ is implemented in the funcion `ds.limma()`. This functions runs a limma-pipeline for microarray data and for RNAseq data allows:

- voom + limma
- DESeq2
- edgeR

We recommend to use the `voom + limma` pipeline proposed here given its versatility and that `limma` is much faster than `DESeq2` and `edgeR`. By default, the function consider that data are obtained from a microarray experiment (`type.data = "microarray"`). Therefore, as we are analyzing RNAseq data, we much indicate that `type.data = "RNAse"`

```
ans.gender <- ds.limma(model =  ~ gdc_cases.demographic.gender,
                       Set = "rse.pre", type.data = "RNAseq")
```

The top differentially expressed genes can be visualized by:

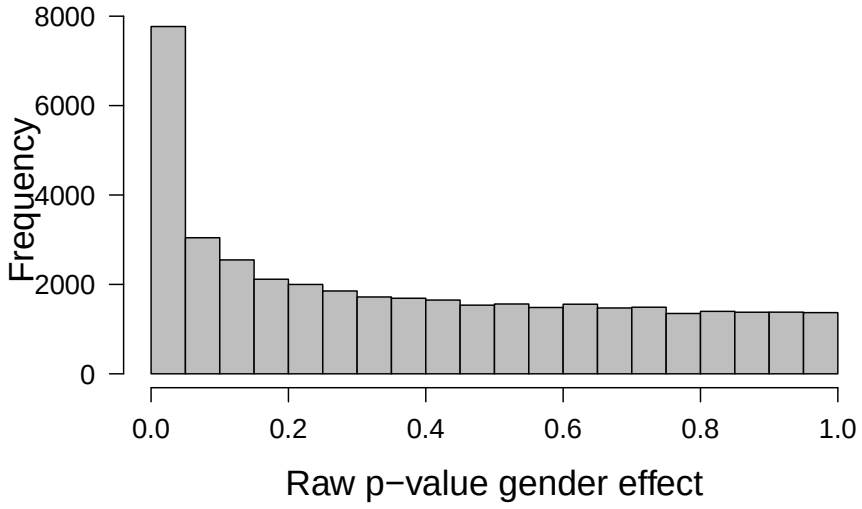
```
ans.gender
```

```
## $study1
## # A tibble: 40,363 x 7
##    id                    n   beta     SE     t   P.Value adj.P.Val
##    <chr>             <int>  <dbl>  <dbl> <dbl>     <dbl>     <dbl>
##  1 ENSG00000274655.1   424 -12.4  0.0761 -52.1 2.74e-187 1.11e-182
##  2 ENSG00000270641.1   424 -10.2  0.461  -43.8 5.21e-160 1.05e-155
##  3 ENSG00000229807.10  424 -11.0  0.0603 -39.5 1.08e-144 1.45e-140
##  4 ENSG00000277577.1   424 -11.3  0.0651 -36.0 2.27e-131 2.29e-127
##  5 ENSG00000233070.1   424  10.9  0.0885  35.5 1.85e-129 1.49e-125
##  6 ENSG00000260197.1   424  10.2  0.118   32.9 3.72e-119 2.50e-115
##  7 ENSG00000213318.4   424  11.4  0.128   31.9 5.57e-115 3.21e-111
##  8 ENSG00000278039.1   424  -7.78 0.0812 -28.8 3.85e-102 1.94e- 98
##  9 ENSG00000067048.16  424   9.62 0.0894  27.4 4.72e- 96 2.12e- 92
## 10 ENSG00000131002.11  424  11.4  0.0924  27.3 9.63e- 96 3.89e- 92
## # ... with 40,353 more rows
##
## attr(,"class")
```

```
## [1] "dsLimma" "list"
```

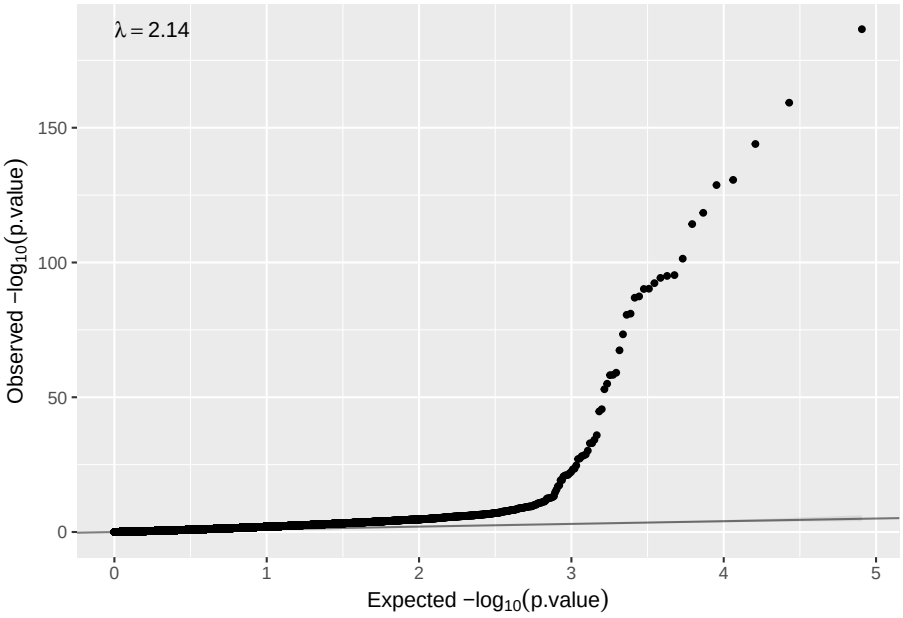
We can verify whether the distribution of the observed p-values are the ones we expect in this type of analyses

```
hist(ans.gender$study1$P.Value, xlab="Raw p-value gender effect",
     main="", las=1, cex.lab=1.5, cex.axis=1.2, col="gray")
```



We can also check whether there is inflation just executing

```
qqplot(ans.gender$study1$P.Value)
```

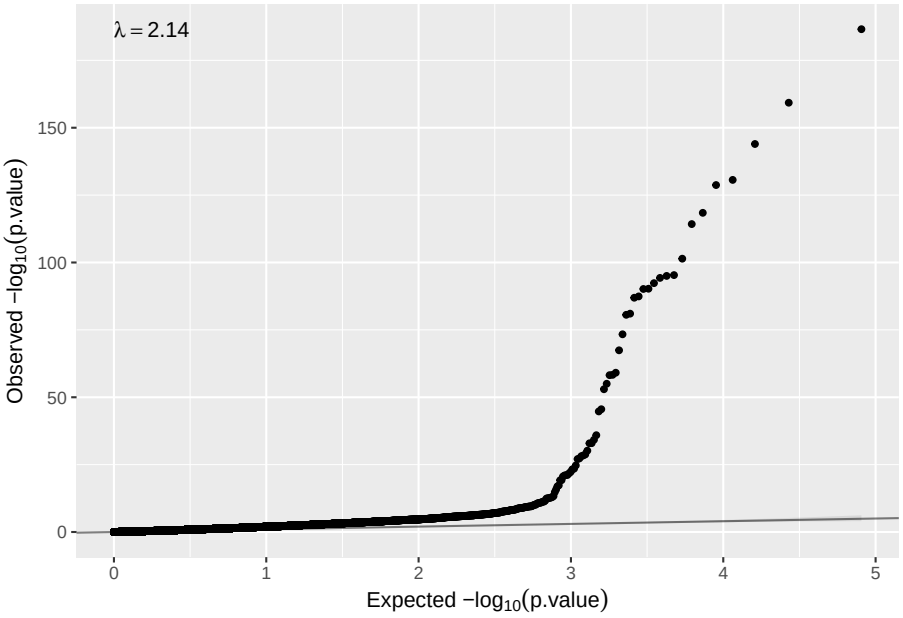


So, in that case, the model needs to remove unwanted variability ($\lambda > 2$). If so, we can use surrogate variable analysis just changing the argument `sva=TRUE`

```
ans.gender.sva <- ds.limma(model =  ~ gdc_cases.demographic.gender,
                        Set = "rse.pre", type.data = "RNAseq",
                        sva = TRUE)
```

Now the inflation has dramatically been reduced ($\lambda > 1.12$)

```
qqplot(ans.gender.sva$study1$P.Value)
```



We can add annotation to the output that is available in our RSE object. We can have access to this information by

```
ds.fvarLabels('rse.pre')
```

```
## $study1
## [1] "chromosome" "start"      "end"        "width"      "strand"
## [6] "gene_id"    "bp_length"  "symbol"
##
## attr(,"class")
## [1] "dsfvarLabels" "list"
```

So, we can run

```
ans.gender.sva <- ds.limma(model =  ~ gdc_cases.demographic.gender,
                      Set = "rse.pre", type.data = "RNAseq",
                      sva = TRUE, annotCols = c("chromosome"))
```

The results are:

```
ans.gender.sva
```

```
## $study1
## # A tibble: 40,363 x 8
##    id                    n   beta     SE     t   P.Value adj.P.Val chromosome
##    <chr>             <int>  <dbl>  <dbl> <dbl>     <dbl>     <dbl> <chr>
##  1 ENSG00000274655.1   424 -12.4  0.0761 -52.1 2.74e-187 1.11e-182 chrX
##  2 ENSG00000270641.1   424 -10.2  0.461  -43.8 5.21e-160 1.05e-155 chrX
```

```
##  3 ENSG00000229807.10   424 -11.0  0.0603 -39.5 1.08e-144 1.45e-140 chrX
##  4 ENSG00000277577.1    424 -11.3  0.0651 -36.0 2.27e-131 2.29e-127 chrX
##  5 ENSG00000233070.1    424  10.9  0.0885  35.5 1.85e-129 1.49e-125 chrY
##  6 ENSG00000260197.1    424  10.2  0.118   32.9 3.72e-119 2.50e-115 chrY
##  7 ENSG00000213318.4    424  11.4  0.128   31.9 5.57e-115 3.21e-111 chr16
##  8 ENSG00000278039.1    424  -7.78 0.0812 -28.8 3.85e-102 1.94e- 98 chrX
##  9 ENSG00000067048.16   424   9.62 0.0894  27.4 4.72e- 96 2.12e- 92 chrY
## 10 ENSG00000131002.11   424  11.4  0.0924  27.3 9.63e- 96 3.89e- 92 chrY
## # ... with 40,353 more rows
##
## attr(,"class")
## [1] "dsLimma" "list"
```

The function has another arguments that can be used to fit other type of models:

- sva: estimate surrogate variables
- annotCols: to add annotation available in the
- method: Linear regression ("ls") or robust regression ("robust") used in limma (`lmFit`)
- robust: robust method used for outlier sample variances used in limma (`eBayes`)
- normalization: normalization method used in the `voom` transformation (default "none")
- voomQualityWeights: should `voomQualityWeights` function be used instead of `voom`? (default FALSE)
- big: should SmartSVA be used instead of SVA (useful for big sample size or when analyzing epigenome data. Default FALSE)

We have also implemented two other functions `ds.DESeq2` and `ds.edgeR` that perform DGE analysis using DESeq2 and edgeR methods. This is the R code used to that purpose:

To be supplied

We close the DataSHIELD session by:

```
datashield.logout(conns)
```

Chapter 4

Epigenome-wide association analysis (EWAS)

EWAS requires basically the same statistical methods as those used in DGE. It should be notice that the **pooled analysis** we are going to illustrate here can also be performed with transcriptomic data since each study must have different range values. If so, gene expression harmonization should be performed, for instance, by standardizing the data at each study. For EWAS where methylation is measured using beta values (e.g CpG data are in the range 0-1) this is not a problem. In any case, adopting the **meta-analysis** approach could be a safe option.

We have downloaded data from GEO corresponding to the accesion number GSE66351 which includes DNA methylation profiling (Illumina 450K array) of 190 individuals. Data corresponds to CpGs beta values measured in the superior temporal gyrus and prefrontal cortex brain regions of patients with Alzheimer's. Data have been downloaded using *GEOquery* package that gets GEO data as `ExpressionSet` objects. Researchers who are not familiar with `ExpressionSet`s can read this Section. Notice that data are encoded as beta-values that ensure data harmonization across studies.

In order to illustrate how to perform data analyses using federated data, we have split the data into two `ExpressionSet`s having 100 and 90 samples as if they were two different studies. Figure **??** shows the two resources defined for both studies (GSE66351_1 and GSE66351_2)

In order to perform omic data analyses, we need first to login and assign resources to DataSHIELD. This can be performed using the `as.resource.object()` function

```
builder <- DSI::newDSLoginBuilder()
builder$append(server = "study1", url = "https://opal-demo.obiba.org",
               user = "dsuser", password = "P@ssw0rd",
               resource = "RSRC.GSE66351_1", profile = "omics")
builder$append(server = "study2", url = "https://opal-demo.obiba.org",
               user = "dsuser", password = "P@ssw0rd",
               resource = "RSRC.GSE66351_2", profile = "omics")

logindata <- builder$build()

conns <- DSI::datashield.login(logins = logindata, assign = TRUE,
                               symbol = "res")
```

```
##
## Logging into the collaborating servers
```

```
##
## Assigning resource data...
```

```
# Assign to the original R class (e.g ExpressionSet)
datashield.assign.expr(conns, symbol = "methy",
                       expr = quote(as.resource.object(res)))
```

Now, we can see that the resources are actually loaded into the R servers as their original class

```
ds.class("methy")
```

```
## $study1
## [1] "ExpressionSet"
## attr(,"package")
## [1] "Biobase"
##
## $study2
## [1] "ExpressionSet"
## attr(,"package")
## [1] "Biobase"
```

Then, some Bioconductor-type functions can be use to return non-disclosive information of `ExpressionSet`s from each server to the client, using similar functions as those defined in the `dsBaseClient` package. For example, feature names can be returned by

```
fn <- ds.featureNames("methy")
lapply(fn, head)
```

```
## $study1
## [1] "cg00000029" "cg00000108" "cg00000109" "cg00000165" "cg00000236"
```

```
## [6] "cg00000289"
##
## $study2
## [1] "cg00000029" "cg00000108" "cg00000109" "cg00000165" "cg00000236"
## [6] "cg00000289"
```

Experimental phenotypes variables can be obtained by

```
ds.varLabels("methy")
```

```
## $study1
##  [1] "title"                    "geo_accession"
##  [3] "status"                   "submission_date"
##  [5] "last_update_date"         "type"
##  [7] "channel_count"            "source_name_ch1"
##  [9] "organism_ch1"             "characteristics_ch1"
## [11] "characteristics_ch1.1"    "characteristics_ch1.2"
## [13] "characteristics_ch1.3"    "characteristics_ch1.4"
## [15] "characteristics_ch1.5"    "characteristics_ch1.6"
## [17] "characteristics_ch1.7"    "characteristics_ch1.8"
## [19] "molecule_ch1"             "extract_protocol_ch1"
## [21] "label_ch1"                "label_protocol_ch1"
## [23] "taxid_ch1"                "hyb_protocol"
## [25] "scan_protocol"            "description"
## [27] "data_processing"          "platform_id"
## [29] "contact_name"             "contact_email"
## [31] "contact_phone"            "contact_laboratory"
## [33] "contact_institute"        "contact_address"
## [35] "contact_city"             "contact_zip/postal_code"
## [37] "contact_country"          "supplementary_file"
## [39] "supplementary_file.1"     "data_row_count"
## [41] "age"                      "braak_stage"
## [43] "brain_region"             "cell type"
## [45] "diagnosis"                "donor_id"
## [47] "sentrix_id"               "sentrix_position"
## [49] "Sex"
##
## $study2
##  [1] "title"                    "geo_accession"
##  [3] "status"                   "submission_date"
##  [5] "last_update_date"         "type"
##  [7] "channel_count"            "source_name_ch1"
##  [9] "organism_ch1"             "characteristics_ch1"
## [11] "characteristics_ch1.1"    "characteristics_ch1.2"
## [13] "characteristics_ch1.3"    "characteristics_ch1.4"
## [15] "characteristics_ch1.5"    "characteristics_ch1.6"
## [17] "characteristics_ch1.7"    "characteristics_ch1.8"
```

```
## [19] "molecule_ch1"            "extract_protocol_ch1"
## [21] "label_ch1"               "label_protocol_ch1"
## [23] "taxid_ch1"               "hyb_protocol"
## [25] "scan_protocol"           "description"
## [27] "data_processing"         "platform_id"
## [29] "contact_name"            "contact_email"
## [31] "contact_phone"           "contact_laboratory"
## [33] "contact_institute"       "contact_address"
## [35] "contact_city"            "contact_zip/postal_code"
## [37] "contact_country"         "supplementary_file"
## [39] "supplementary_file.1"    "data_row_count"
## [41] "age"                     "braak_stage"
## [43] "brain_region"            "cell type"
## [45] "diagnosis"               "donor_id"
## [47] "sentrix_id"              "sentrix_position"
## [49] "Sex"
##
## attr(,"class")
## [1] "dsvarLabels" "list"
```

4.1 Single CpG analysis

Once the methylation data have been loaded into the opal server, we can perform different type of analyses using functions from the `dsOmicsClient` package. Let us start by illustrating how to analyze a single CpG from two studies by using an approach that is mathematically equivalent to placing all individual-level.

```
ans <- ds.lmFeature(feature = "cg07363416",
                    model = ~ diagnosis + Sex,
                    Set = "methy",
                    datasources = conns)
```

```
## Iteration 1...

## CURRENT DEVIANCE:      7.834782376469

## Iteration 2...

## CURRENT DEVIANCE:      5.21465032637778

## Iteration 3...

## CURRENT DEVIANCE:      5.21465032637778

## SUMMARY OF MODEL STATE after iteration 3

## Current deviance 5.21465032637778 on 187 degrees of freedom

## Convergence criterion TRUE (0)
```

```
##
## beta: 0.143108602924252 0.0345988560191347 -0.12176433102489

##
## Information matrix overall:

##               (Intercept) diagnosisCTRL SexM
## (Intercept)           190            84   89
## diagnosisCTRL          84            84   50
## SexM                   89            50   89

##
## Score vector overall:

##                        [,1]
## (Intercept)   1.187939e-13
## diagnosisCTRL 4.898859e-14
## SexM          4.524159e-14

##
## Current deviance: 5.21465032637778
ans

##              Estimate Std. Error   p-value
## cg07363416 0.03459886 0.02504291 0.1670998
## attr(,"class")
## [1] "dsLmFeature" "matrix"      "array"
```

4.2 Multiple CpG analysis

The same analysis can be performed for all features (e.g. CpGs) just avoiding the `feature` argument. This process can be parallelized using `mclapply` function from the `multicore` package.

```
ans <- ds.lmFeature(model = ~ diagnosis + Sex,
                    Set = "methy",
                    datasources = conns,
                    mc.cores = 20)
```

This method corresponds to the **pooled analysis** approach and can be very time consiming since the function repeatedly calls the DataSHIELD function `ds.glm()`. We can adopt another strategy that is to run a glm of each feature independently at each study using `limma` package (which is really fast) and then combine the results (i.e. **meta-analysis** approach).

```
ans.limma <- ds.limma(model = ~ diagnosis + Sex,
                      Set = "methy",
                      datasources = conns)
```

Then, we can visualize the top genes at each study (i.e server) by

```
lapply(ans.limma, head)
```

```
## $study1
## # A tibble: 6 x 7
##   id             n    beta      SE     t       P.Value adj.P.Val
##   <chr>      <int>   <dbl>   <dbl> <dbl>         <dbl>     <dbl>
## 1 cg13138089   100 -0.147  0.0122  -6.62 0.00000000190  0.000466
## 2 cg23859635   100 -0.0569 0.00520 -6.58 0.00000000232  0.000466
## 3 cg13772815   100 -0.0820 0.0135  -6.50 0.00000000327  0.000466
## 4 cg12706938   100 -0.0519 0.00872 -6.45 0.00000000425  0.000466
## 5 cg24724506   100 -0.0452 0.00775 -6.39 0.00000000547  0.000466
## 6 cg02812891   100 -0.125  0.0163  -6.33 0.00000000731  0.000466
##
## $study2
## # A tibble: 6 x 7
##   id             n    beta      SE     t      P.Value adj.P.Val
##   <chr>      <int>   <dbl>   <dbl> <dbl>        <dbl>     <dbl>
## 1 cg04046629    90 -0.101  0.0128  -5.91 0.0000000621    0.0172
## 2 cg07664323    90 -0.0431 0.00390 -5.85 0.0000000822    0.0172
## 3 cg27098804    90 -0.0688 0.0147  -5.79 0.000000107     0.0172
## 4 cg08933615    90 -0.0461 0.00791 -5.55 0.000000298     0.0360
## 5 cg18349298    90 -0.0491 0.00848 -5.42 0.000000507     0.0489
## 6 cg02182795    90 -0.0199 0.0155  -5.36 0.000000670     0.0538
```

The annotation can be added by using the argument `annotCols`. It should be a vector with the columns of the annotation available in the `ExpressionSet` or `RangedSummarizedExperiment` that want to be showed. The columns of the annotation can be obtained by

```
ds.fvarLabels("methy")
```

```
## $study1
##  [1] "ID"                          "Name"
##  [3] "AddressA_ID"                 "AlleleA_ProbeSeq"
##  [5] "AddressB_ID"                 "AlleleB_ProbeSeq"
##  [7] "Infinium_Design_Type"        "Next_Base"
##  [9] "Color_Channel"               "Forward_Sequence"
## [11] "Genome_Build"                "CHR"
## [13] "MAPINFO"                     "SourceSeq"
## [15] "Chromosome_36"               "Coordinate_36"
## [17] "Strand"                      "Probe_SNPs"
## [19] "Probe_SNPs_10"               "Random_Loci"
## [21] "Methyl27_Loci"               "UCSC_RefGene_Name"
## [23] "UCSC_RefGene_Accession"      "UCSC_RefGene_Group"
## [25] "UCSC_CpG_Islands_Name"       "Relation_to_UCSC_CpG_Island"
```

```
## [27] "Phantom"                      "DMR"
## [29] "Enhancer"                     "HMM_Island"
## [31] "Regulatory_Feature_Name"      "Regulatory_Feature_Group"
## [33] "DHS"                          "RANGE_START"
## [35] "RANGE_END"                    "RANGE_GB"
## [37] "SPOT_ID"
##
## $study2
##  [1] "ID"                           "Name"
##  [3] "AddressA_ID"                  "AlleleA_ProbeSeq"
##  [5] "AddressB_ID"                  "AlleleB_ProbeSeq"
##  [7] "Infinium_Design_Type"         "Next_Base"
##  [9] "Color_Channel"                "Forward_Sequence"
## [11] "Genome_Build"                 "CHR"
## [13] "MAPINFO"                      "SourceSeq"
## [15] "Chromosome_36"                "Coordinate_36"
## [17] "Strand"                       "Probe_SNPs"
## [19] "Probe_SNPs_10"                "Random_Loci"
## [21] "Methyl27_Loci"                "UCSC_RefGene_Name"
## [23] "UCSC_RefGene_Accession"       "UCSC_RefGene_Group"
## [25] "UCSC_CpG_Islands_Name"        "Relation_to_UCSC_CpG_Island"
## [27] "Phantom"                      "DMR"
## [29] "Enhancer"                     "HMM_Island"
## [31] "Regulatory_Feature_Name"      "Regulatory_Feature_Group"
## [33] "DHS"                          "RANGE_START"
## [35] "RANGE_END"                    "RANGE_GB"
## [37] "SPOT_ID"
##
## attr(,"class")
## [1] "dsfvarLabels" "list"
```

Then we can run the analysis and obtain the output with the chromosome and gene symbol by:

```
ans.limma.annot <- ds.limma(model = ~ diagnosis + Sex,
                            Set = "methy",
                            annotCols = c("CHR", "UCSC_RefGene_Name"),
                            datasources = conns)
```

```
lapply(ans.limma.annot, head)
```

```
## $study1
## # A tibble: 6 x 9
##   id          n    beta      SE     t   P.Value adj.P.Val CHR   UCSC_RefGene_Na~
##   <chr>   <int>   <dbl>   <dbl> <dbl>     <dbl>     <dbl> <chr> <chr>
## 1 cg1313~   100 -0.147  0.0122  -6.62   1.90e-9  0.000466 2     "ECEL1P2"
## 2 cg2385~   100 -0.0569 0.00520 -6.58   2.32e-9  0.000466 2     "MTA3"
```

```
## 3 cg1377~   100 -0.0820 0.0135  -6.50   3.27e-9  0.000466 17    ""
## 4 cg1270~   100 -0.0519 0.00872 -6.45   4.25e-9  0.000466 19    "MEX3D"
## 5 cg2472~   100 -0.0452 0.00775 -6.39   5.47e-9  0.000466 19    "ISOC2;ISOC2;IS~
## 6 cg0281~   100 -0.125  0.0163  -6.33   7.31e-9  0.000466 2     "ECEL1P2"
##
## $study2
## # A tibble: 6 x 9
##   id          n    beta      SE     t   P.Value adj.P.Val CHR   UCSC_RefGene_Na~
##   <chr>   <int>   <dbl>   <dbl> <dbl>     <dbl>     <dbl> <chr> <chr>
## 1 cg0404~    90 -0.101  0.0128  -5.91   6.21e-8    0.0172 11    "CD6"
## 2 cg0766~    90 -0.0431 0.00390 -5.85   8.22e-8    0.0172 6     "MUC21"
## 3 cg2709~    90 -0.0688 0.0147  -5.79   1.07e-7    0.0172 11    "CD6"
## 4 cg0893~    90 -0.0461 0.00791 -5.55   2.98e-7    0.0360 1     ""
## 5 cg1834~    90 -0.0491 0.00848 -5.42   5.07e-7    0.0489 3     "RARRES1;RARRES~
## 6 cg0218~    90 -0.0199 0.0155  -5.36   6.70e-7    0.0538 8     ""
```

Then, the last step is to meta-analyze the results. Different methods can be used to this end. We have implemented a method that meta-analyze the p-pvalues of each study as follows:

```
ans.meta <- metaPvalues(ans.limma)
ans.meta
```

```
## # A tibble: 481,868 x 4
##    id                study1       study2   p.meta
##    <chr>              <dbl>        <dbl>    <dbl>
##  1 cg13138089 0.00000000190 0.00000763  4.78e-13
##  2 cg25317941 0.0000000179  0.00000196  1.12e-12
##  3 cg02812891 0.00000000731 0.00000707  1.63e-12
##  4 cg12706938 0.00000000425 0.0000161   2.14e-12
##  5 cg16026647 0.000000101   0.000000797 2.51e-12
##  6 cg12695465 0.00000000985 0.0000144   4.33e-12
##  7 cg21171625 0.000000146   0.00000225  9.78e-12
##  8 cg13772815 0.00000000327 0.000122    1.18e-11
##  9 cg00228891 0.000000166   0.00000283  1.38e-11
## 10 cg21488617 0.0000000186  0.0000299   1.62e-11
## # ... with 481,858 more rows
```

We can verify that the results are pretty similar to those obtained using pooled analyses. Here we compute the association for two of the top-CpGs:

```
res1 <- ds.lmFeature(feature = "cg13138089",
                     model = ~ diagnosis + Sex,
                     Set = "methy",
                     datasources = conns)
```

```
## Iteration 1...
```

```
## CURRENT DEVIANCE:      29.765420228408
```

```
## Iteration 2...

## CURRENT DEVIANCE:      2.43819505985575

## Iteration 3...

## CURRENT DEVIANCE:      2.43819505985575

## SUMMARY OF MODEL STATE after iteration 3

## Current deviance 2.43819505985575 on 187 degrees of freedom

## Convergence criterion TRUE (0)

##
## beta: 0.443404581787976 -0.137334787076049 -0.0218537351160394

##
## Information matrix overall:

##               (Intercept) diagnosisCTRL SexM
## (Intercept)           190            84   89
## diagnosisCTRL          84            84   50
## SexM                   89            50   89

##
## Score vector overall:

##                          [,1]
## (Intercept)   -9.681145e-14
## diagnosisCTRL -6.980527e-14
## SexM          -5.542788e-14

##
## Current deviance: 2.43819505985575
```

```
res1
```

```
##              Estimate Std. Error      p-value
## cg13138089 -0.1373348 0.01712405 1.057482e-15
## attr(,"class")
## [1] "dsLmFeature" "matrix"      "array"
```

```
res2 <- ds.lmFeature(feature = "cg13772815",
                     model = ~ diagnosis + Sex,
                     Set = "methy",
                     datasources = conns)
```

```
## Iteration 1...

## CURRENT DEVIANCE:      37.4410001835

## Iteration 2...

## CURRENT DEVIANCE:      0.692937396766391
```

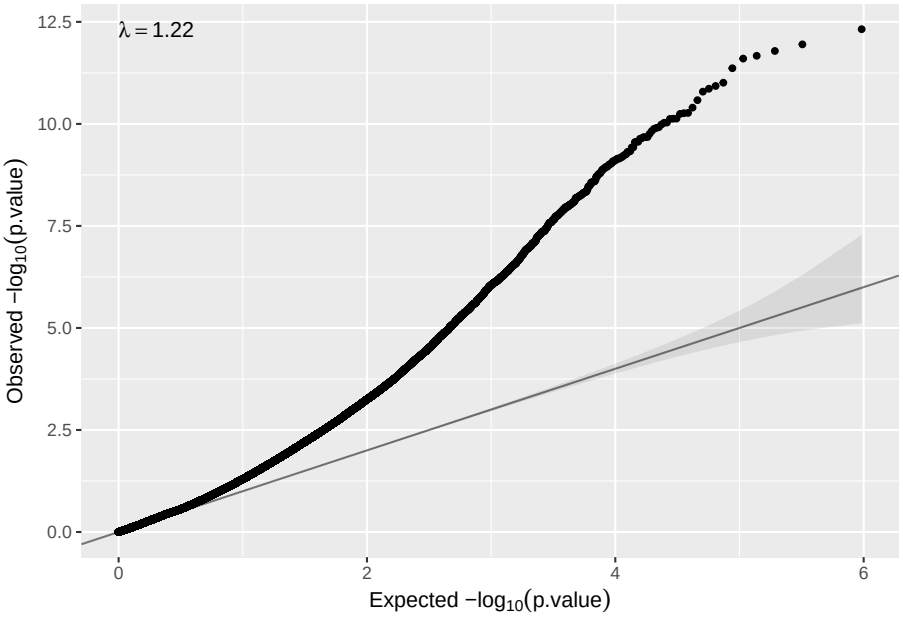
```
## Iteration 3...

## CURRENT DEVIANCE:      0.692937396766391

## SUMMARY OF MODEL STATE after iteration 3

## Current deviance 0.692937396766391 on 187 degrees of freedom

## Convergence criterion TRUE (0)

##
## beta: 0.471427788697297 -0.0678613723066762 -0.00640757953624292

##
## Information matrix overall:

##               (Intercept) diagnosisCTRL SexM
## (Intercept)           190            84   89
## diagnosisCTRL          84            84   50
## SexM                   89            50   89

##
## Score vector overall:

##                          [,1]
## (Intercept)   -2.370326e-14
## diagnosisCTRL -1.237899e-14
## SexM          -1.260103e-14

##
## Current deviance: 0.692937396766391
```

```
res2
```

```
##               Estimate  Std. Error      p-value
## cg13772815 -0.06786137 0.009128915 1.056225e-13
## attr(,"class")
## [1] "dsLmFeature" "matrix"      "array"
```

We can create a QQ-plot by using the function `qqplot` available in our package.

```
qqplot(ans.meta$p.meta)
```



Here In some cases inflation can be observed, so that, correction for cell-type or surrogate variables must be performed. We describe how we can do that in the next two sections.

4.3 Adjusting for Surrogate Variables

The vast majority of omic studies require to control for unwanted variability. The surrogate variable analysis (SVA) can address this issue by estimating some hidden covariates that capture differences across individuals due to some artifacts such as batch effects or sample quality sam among others. The method is implemented in *SVA* package.

Performing this type of analysis using the `ds.lmFeature` function is not allowed since estimating SVA would require to implement a non-disclosive method that computes SVA from the different servers. This will be a future topic of the `dsOmicsClient`. NOTE that, estimating SVA separately at each server would not be a good idea since the aim of SVA is to capture differences mainly due to experimental issues among ALL individuals. What we can do instead is to use the `ds.limma` function to perform the analyses adjusted for SVA at each study.

```
##
## Logging into the collaborating servers

##
## Assigning resource data...
```

```
ans.sva <- ds.limma(model = ~ diagnosis + Sex,
                    Set = "methy",
                    sva = TRUE, annotCols = c("CHR", "UCSC_RefGene_Name"))
ans.sva
```

```
## $study1
## # A tibble: 481,868 x 9
##    id          n    beta      SE     t  P.Value adj.P.Val CHR   UCSC_RefGene_Na~
##    <chr>   <int>   <dbl>   <dbl> <dbl>    <dbl>     <dbl> <chr> <chr>
##  1 cg1313~   100 -0.147  0.0122  -6.62  1.90e-9  0.000466 2     "ECEL1P2"
##  2 cg2385~   100 -0.0569 0.00520 -6.58  2.32e-9  0.000466 2     "MTA3"
##  3 cg1377~   100 -0.0820 0.0135  -6.50  3.27e-9  0.000466 17    ""
##  4 cg1270~   100 -0.0519 0.00872 -6.45  4.25e-9  0.000466 19    "MEX3D"
##  5 cg2472~   100 -0.0452 0.00775 -6.39  5.47e-9  0.000466 19    "ISOC2;ISOC2;IS~
##  6 cg0281~   100 -0.125  0.0163  -6.33  7.31e-9  0.000466 2     "ECEL1P2"
##  7 cg2766~   100 -0.0588 0.0198  -6.33  7.48e-9  0.000466 16    "ANKRD11"
##  8 cg1537~   100 -0.0709 0.0115  -6.32  7.83e-9  0.000466 2     "LPIN1"
##  9 cg1552~   100 -0.0446 0.00750 -6.29  8.69e-9  0.000466 10    ""
## 10 cg1269~   100 -0.0497 0.00155 -6.27  9.85e-9  0.000475 6     "GCNT2;GCNT2;GC~
## # ... with 481,858 more rows
##
## $study2
## # A tibble: 481,868 x 9
##    id          n    beta      SE     t P.Value adj.P.Val CHR   UCSC_RefGene_Name
##    <chr>   <int>   <dbl>   <dbl> <dbl>   <dbl>     <dbl> <chr> <chr>
##  1 cg040~    90 -0.101  0.0128  -5.91 6.21e-8    0.0172 11    "CD6"
##  2 cg076~    90 -0.0431 0.00390 -5.85 8.22e-8    0.0172 6     "MUC21"
##  3 cg270~    90 -0.0688 0.0147  -5.79 1.07e-7    0.0172 11    "CD6"
##  4 cg089~    90 -0.0461 0.00791 -5.55 2.98e-7    0.0360 1     ""
##  5 cg183~    90 -0.0491 0.00848 -5.42 5.07e-7    0.0489 3     "RARRES1;RARRES1"
##  6 cg021~    90 -0.0199 0.0155  -5.36 6.70e-7    0.0538 8     ""
##  7 cg160~    90 -0.0531 0.0196  -5.31 7.97e-7    0.0548 17    "MEIS3P1"
##  8 cg012~    90 -0.0537 0.00971 -5.18 1.39e-6    0.0582 7     "HOXA2"
##  9 cg251~    90 -0.0224 0.00736 -5.15 1.57e-6    0.0582 3     "ZBTB20;ZBTB20;ZB~
## 10 cg074~    90 -0.0475 0.00166 -5.13 1.67e-6    0.0582 22    "C22orf27"
## # ... with 481,858 more rows
##
## attr(,"class")
## [1] "dsLimma" "list"
```

Then, data can be combined meta-anlyzed as follows:

```
ans.meta.sv <- metaPvalues(ans.sva)
ans.meta.sv
```

```
## # A tibble: 481,868 x 4
##    id                study1      study2   p.meta
```

```
##    <chr>              <dbl>       <dbl>    <dbl>
##  1 cg13138089 0.00000000190 0.00000763  4.78e-13
##  2 cg25317941 0.0000000179  0.00000196  1.12e-12
##  3 cg02812891 0.00000000731 0.00000707  1.63e-12
##  4 cg12706938 0.00000000425 0.0000161   2.14e-12
##  5 cg16026647 0.000000101   0.000000797 2.51e-12
##  6 cg12695465 0.00000000985 0.0000144   4.33e-12
##  7 cg21171625 0.000000146   0.00000225  9.78e-12
##  8 cg13772815 0.00000000327 0.000122    1.18e-11
##  9 cg00228891 0.000000166   0.00000283  1.38e-11
## 10 cg21488617 0.0000000186  0.0000299   1.62e-11
## # ... with 481,858 more rows
```

The DataSHIELD session must by closed by:

```
datashield.logout(conns)
```