ShinyDataSHIELD User's Guide

Escribà Montagut, Xavier; González, Juan R.

2022-01-18

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.

Chapter 2

Intro

a

Chapter 3

Omic data analysis: types of implemented analyses

The Figure **??** 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 **??**) 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 **??** overcomes the limitations raised when performing pooled analyses. First, the computation issue is addressed by using scalable and fast methods to perform data analysis at whole-genome

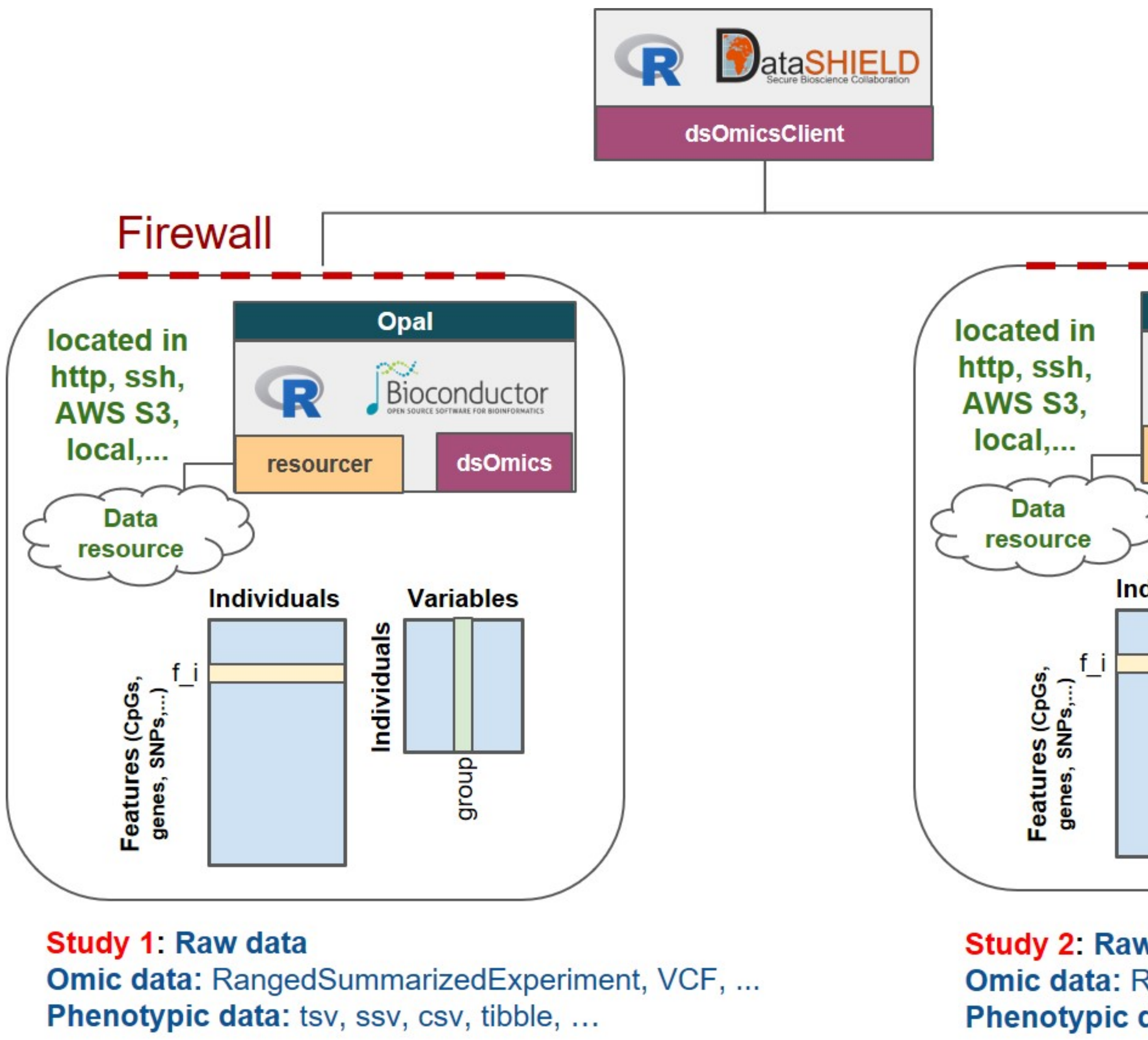
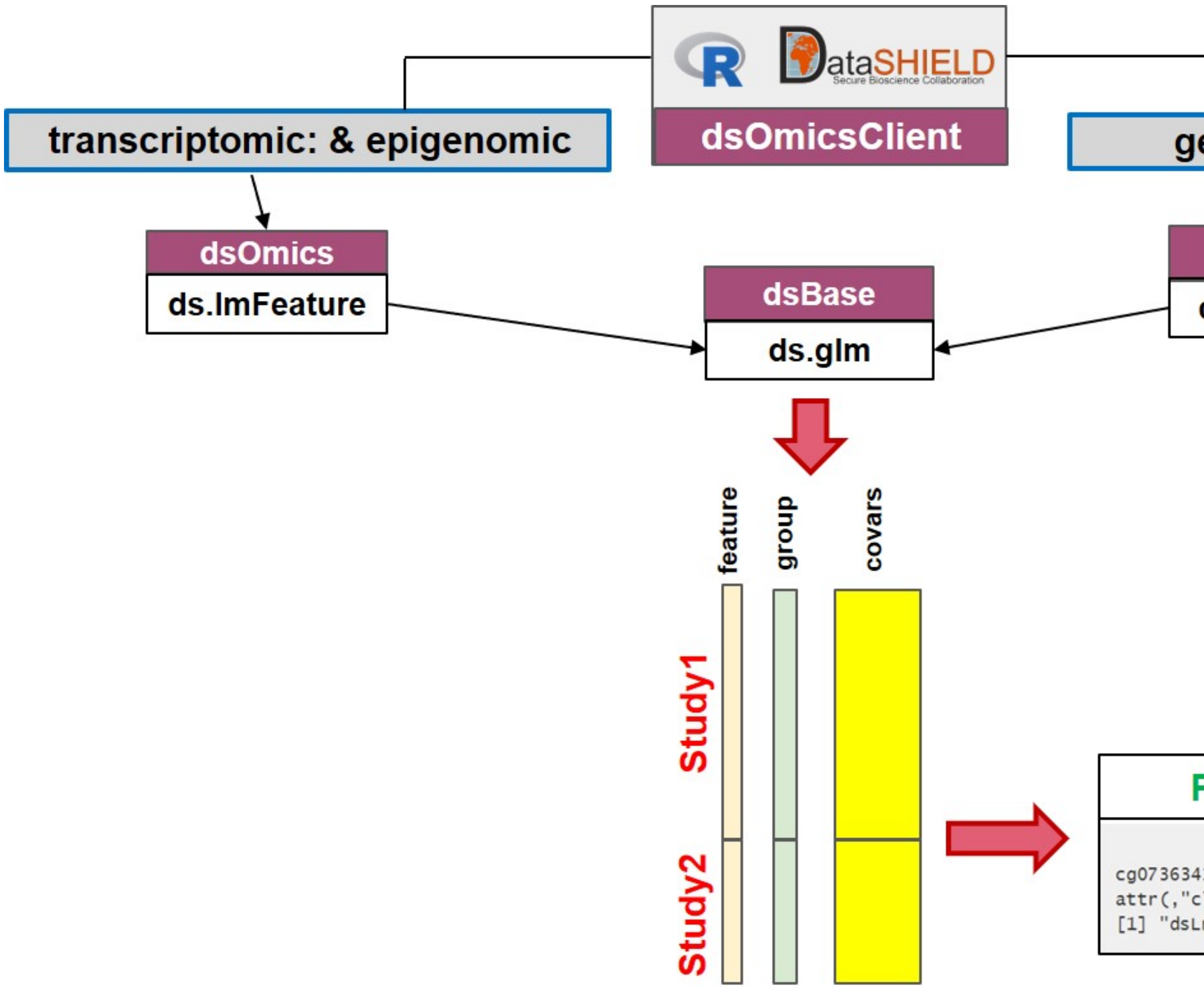

DataSHIELD
Secure Bioscience Collaboration
dsOmicsClient
Firewall
Opal
Bioconductor
OPEN SOURCE SOFTWARE FOR BIOINFORMATICS
resourcer
dsOmics
located in http, ssh, AWS S3, local,...
Data resource
Individuals
Variables
f_i
Features (CpGs, genes, SNPs,...)
Individuals
group
Study 1: Raw data
Omic data: RangedSummarizedExperiment, VCF, ...
Phenotypic data: tsv, ssv, csv, tibble, …
located in http, ssh, AWS S3, local,...
Data resource
f_i
Features (CpGs, genes, SNPs,...)
Study 2: Raw
Omic data: R
Phenotypic

Figure 3.1: 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.

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.

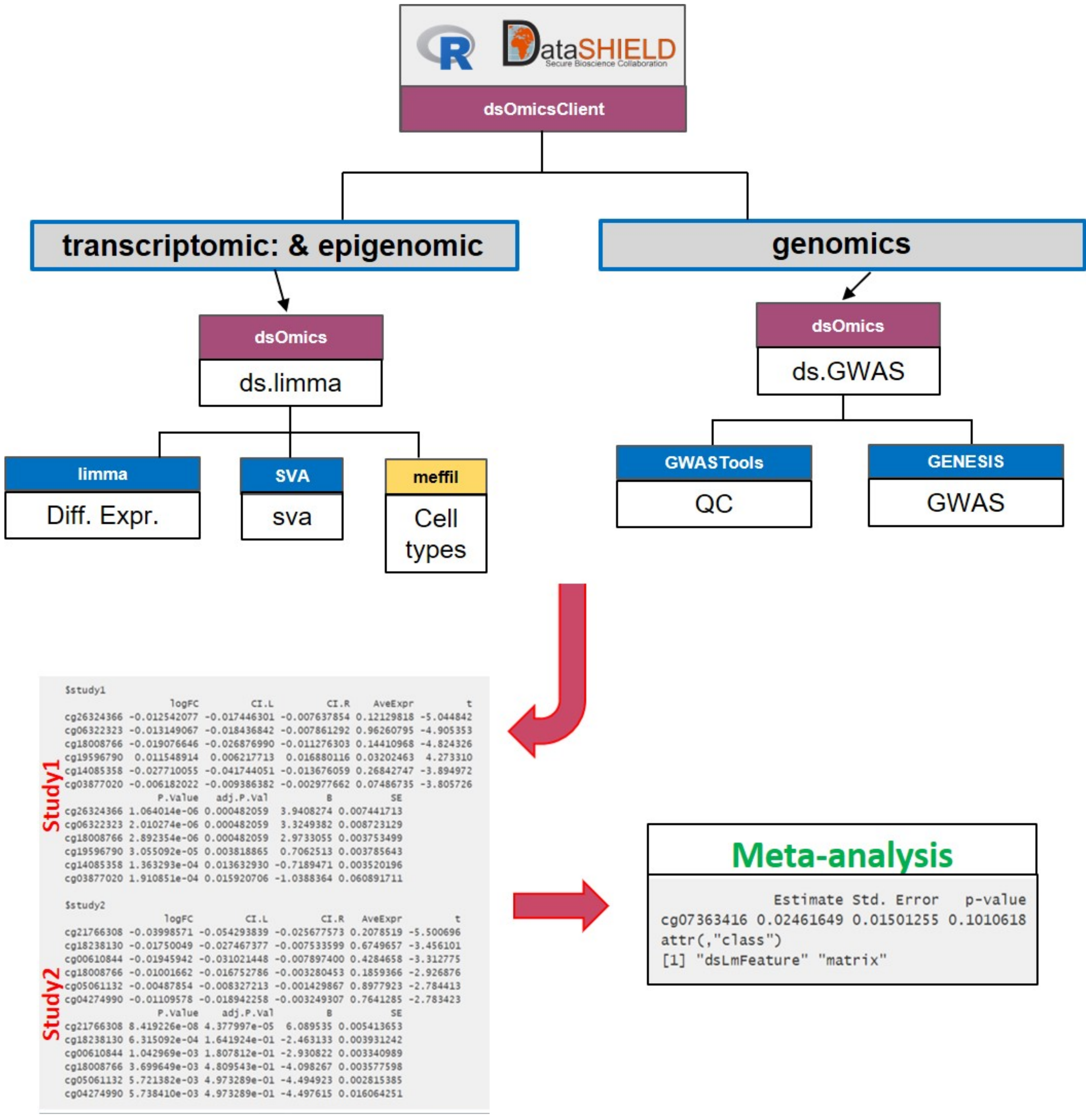
Next, we describe how both approaches are implemented:

- **Pooled approach:** Figure **??** 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 **??** 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.



```
cg073634
attr(,"c
[1] "dsL
```

Figure 3.2: 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.



```
$study1
                logFC        CI.L         CI.R    AveExpr         t
cg26324366 -0.012542077 -0.017446301 -0.007637854 0.12129818 -5.044842
cg06322323 -0.013149067 -0.018436842 -0.007861292 0.96260795 -4.905353
cg18008766 -0.019076646 -0.026876990 -0.011276303 0.14410968 -4.824326
cg19596790  0.011548914  0.006217713  0.016880116 0.03202463  4.273310
cg14085358 -0.027710055 -0.041744051 -0.013676059 0.26842747 -3.894972
cg03877020 -0.006182022 -0.009386382 -0.002977662 0.07486735 -3.805726
               P.Value    adj.P.Val          B          SE
cg26324366 1.064014e-06 0.000482059  3.9408274 0.007441713
cg06322323 2.010274e-06 0.000482059  3.3249382 0.008723129
cg18008766 2.892354e-06 0.000482059  2.9733055 0.003753499
cg19596790 3.055092e-05 0.003818865  0.7062513 0.003785643
cg14085358 1.363293e-04 0.013632930 -0.7189471 0.003520196
cg03877020 1.910851e-04 0.015920706 -1.0388364 0.060891711

$study2
               logFC        CI.L         CI.R   AveExpr         t
cg21766308 -0.03998571 -0.054293839 -0.025677573 0.2078519 -5.500696
cg18238130 -0.01750049 -0.027467377 -0.007533599 0.6749657 -3.456101
cg00610844 -0.01945942 -0.031021448 -0.007897400 0.4284658 -3.312775
cg18008766 -0.01001662 -0.016752786 -0.003280453 0.1859366 -2.926876
cg05061132 -0.00487854 -0.008327213 -0.001429867 0.8977923 -2.784413
cg04274990 -0.01109578 -0.018942258 -0.003249307 0.7641285 -2.783423
               P.Value    adj.P.Val         B          SE
cg21766308 8.419226e-08 4.377997e-05  6.089535 0.005413653
cg18238130 6.315092e-04 1.641924e-01 -2.463133 0.003931242
cg00610844 1.042969e-03 1.807812e-01 -2.930822 0.003340989
cg18008766 3.699649e-03 4.809543e-01 -4.098267 0.003577598
cg05061132 5.721382e-03 4.973289e-01 -4.494923 0.002815385
cg04274990 5.738410e-03 4.973289e-01 -4.497615 0.016064251
```

```
             Estimate Std. Error   p-value
cg07363416 0.02461649 0.01501255 0.1010618
attr(,"class")
[1] "dsLmFeature" "matrix"
```

Figure 3.3: 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 uninng standar R functions.

Chapter 4

Genome wide association study (GWAS) analysis

Three different use cases will be illustrated:

- Single cohort GWAS. Using data from all the individuals.
- Multi-cohort GWAS. Using the same individuals as the single cohort separated into three synthetic cohorts.
- Polygenic risk score. Using data from all the individuals.

For all the illustrated cases we will use data divided by chromosome (VCF files). The procedure is the same if all the variants are on a single VCF.

The data used along this section is a synthetic data set generated by the CINECA project made freely available under the Creative Commons Licence (CC-BY) (funding: EC H2020 grant 825775). It has been pruned and the individuals have been separated into three synthetic cohorts. For the "Single cohort" use case, all the individuals are used. Information on individuals and SNP count of each cohort can be found on the Table **??**.

Table 4.1: Numer of SNPs and individuals by cohort

	Cohort 1	Cohort 2	Cohort 3	Total
Number of SNPs	865,240	865,240	865,240	865,240
Number of individuals	817	1,073	614	2,504

4.1 Single cohort

The single cohort analysis is a way of performing a GWAS study guaranteeing GDPR data confidentiality. The structure followed is illustrated on the following figure.

The data analyst corresponds to the "RStudio" session, which through DataSHIELD Interface (DSI) connects with the Opal server located at the cohort network. This Opal server contains an array of resources that correspond to the different VCF files (sliced by chromosome) [1] and a resource that corresponds to the phenotypes table of the studied individuals.

4.1.1 Connection to the Opal server

We have to create an Opal connection object to the cohort server. We do that using the following functions.

```
require('DSI')
require('DSOpal')
require('dsBaseClient')
require('dsOmicsClient')

builder <- DSI::newDSLoginBuilder()
builder$append(server = "cohort1", url = "https://opal-demo.obiba.org/",
               user =  "dsuser", password = "P@ssw0rd",
               driver = "OpalDriver", profile = "omics")
logindata <- builder$build()
conns <- DSI::datashield.login(logins = logindata)
```

4.1.2 Assign the VCF resources

Now that we have created a connection object to the Opal, we have started a new R session on the server, our analysis will take place in this remote session, so we have to load the data into it. This is done via a two step process:

1. Load the resources to the R session. The resources can be imagined as a data structure that contains the information about were to find a data set and the access credentials to it; we as users are not able to look at this information (it is privately stored on the Opal server), but we can load it into our remote R session to make use of it. Following that, the second step comes naturally.
2. Resolve the resources. Once we have the information on how to access the data, we have to actually retrieve it on the remote R session to analyze it.

In this use case we will use 21 different resources from the `GWAS` project hosted on the demo Opal server. This resources correspond to VCF files with information on individual chromosomes. The names of the resources are `chrXXA` (where XX is the chromosome number). Following the Opal syntax, we will refer to them using the string `GWAS.chrXXA`.

[1]The same methodology and code can be used with unitary VCF resources that contain the variant information of all chromosomes.

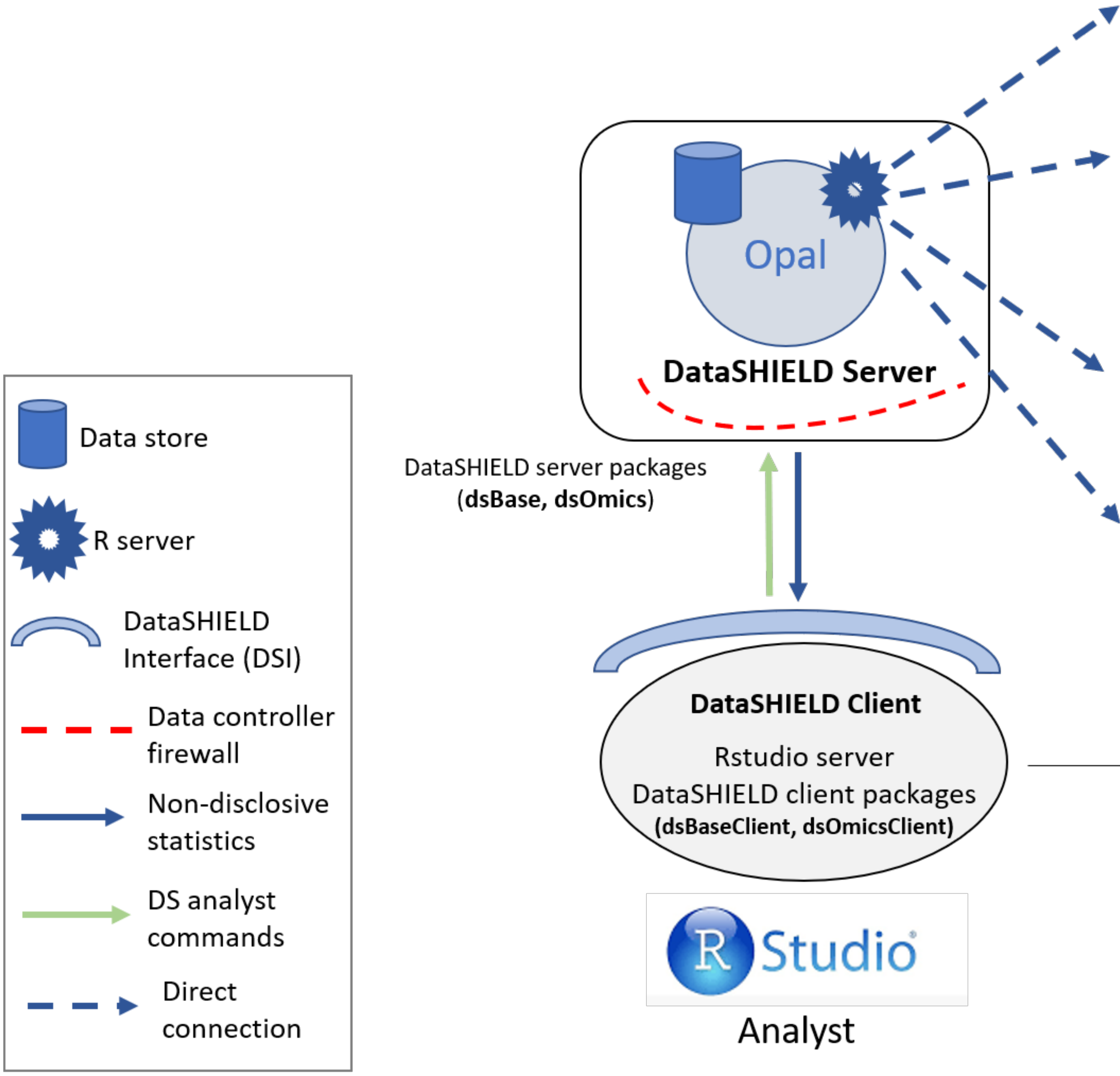

Data store
R server
DataSHIELD Interface (DSI)
Data controller firewall
Non-disclosive statistics
DS analyst commands
Direct connection
Opal
DataSHIELD Server
DataSHIELD server packages (dsBase, dsOmics)
DataSHIELD Client
Rstudio server
DataSHIELD client packages
(dsBaseClient, dsOmicsClient)
R Studio
Analyst

Figure 4.1: Proposed infrastructure to perform single-cohort GWAS studies.

To load the resources we will use the `DSI::datashield.assign.resource()` function. Note that along the use case we use the `lapply` function to simplify our code since we have to perform repetitive operations. `lapply` is a way of creating loops in R, for more information visit the documentation.

```
lapply(1:21, function(x){
  DSI::datashield.assign.resource(conns, paste0("chr", x), paste0("GWAS.chr", x))
  })
```

Now we have assigned all the resources named `GWAS.chrXXA` into our remote R session. We have assigned them to the variables called `chrXX`. To verify this step has been performed correctly, we could use the `ds.class` function to check for their class and that they exist on the remote session.

```
ds.class("chr1")
```

```
## $cohort1
## [1] "GDSFileResourceClient" "FileResourceClient"    "ResourceClient"
## [4] "R6"
```

To resolve the resources and retrieve the data in the remote session we will use the `DSI::datashield.assign.expr()` function. This function runs a line of code on the remote session [2], and in particular we want to run the function `as.resource.object()`, which is the DataSHIELD function in charge of resolving the resources.

```
lapply(1:21, function(x){
  DSI::datashield.assign.expr(conns = conns, symbol = paste0("gds", x, "_object"),
                              expr = as.symbol(paste0("as.resource.object(chr", x, ")")))
})
```

Now we have resolved the resources named `chrXX` into our remote R session. The objects retrieved have been assigned into variables named `gdsXX_object`. We can check the process was successful as we did before.

```
ds.class("gds1_object")
```

```
## $cohort1
## [1] "GdsGenotypeReader"
## attr(,"package")
## [1] "GWASTools"
```

4.1.3 Assign the phenotypes

The objects we have loaded into our remote session are VCF files that contain genomic information of the individuals. To perform a GWAS this information has to be related to some phenotypes to extract relationships. Therefore, we

[2]The same methodology and code can be used with unitary VCF resources that contain the variant information of all chromosomes.

need to load the phenotypes into the remote session. The phenotypes information is a table that contains the individuals as rows and phenotypes as columns. In this use case, we will use a resource (as with the VCF files) to load the phenotypes table into the remote session.

The procedure is practically the same as before with some minor tweaks. To begin, we have the phenotypes information in a single table (hence, a single resource). Then, instead of using the function `as.resource.object`, we will use `as.resource.data.frame`, this is because before we were loading a special object (VCF file) and now we are loading a plain table, so the internal treatment on the remote session has to be different.

```
DSI::datashield.assign.resource(conns, "pheno", "GWAS.ega_phenotypes")
DSI::datashield.assign.expr(conns = conns, symbol = "pheno_object",
                            expr = quote(as.resource.data.frame(pheno)))
```

We can follow the same analogy as before to know that we have assigned the phenotypes table to a variable called `pheno_object` on the remote R session.

```
ds.class("pheno_object")
```

```
## $cohort1
## [1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

We can also check the column names to see which information is present on the table.

```
ds.colnames("pheno_object")[[1]]
```

```
##  [1] "age_cancer_diagnosis"                "age_recruitment"
##  [3] "alcohol_day"                         "basophill_count"
##  [5] "behaviour_tumour"                    "bmi"
##  [7] "c_reactive_protein_reportability"    "record_origin"
##  [9] "report_format"                       "cholesterol"
## [11] "birth_country"                       "actual_tobacco_smoker"
## [13] "date_diagnosis"                      "diabetes_diagnosed_doctor"
## [15] "diastolic_blood_pressure"            "duration_moderate_activity"
## [17] "duration_vigorous_activity"          "energy"
## [19] "eosinophill_count"                   "ethnicity"
## [21] "ever_smoked"                         "fasting_time"
## [23] "alcohol_frequency"                   "hdl_cholesterol"
## [25] "hip_circumference"                   "histology_tumour"
## [27] "home_e_coordinate"                   "home_n_coordinate"
## [29] "e_house_score"                       "s_house_score"
## [31] "w_house_score"                       "e_income_score"
## [33] "w_income_score"                      "ldl_direct"
## [35] "lymphocyte_count"                    "monocyte_count"
## [37] "neutrophill_count"                   "n_accidental_death_close_gen_family"
## [39] "household_count"                     "children_count"
## [41] "operations_count"                     "operative_procedures"
## [43] "past_tobacco"                         "platelet_count"
## [45] "pulse_rate"                           "qualification"
## [47] "cancer_occurrences"                   "reticulocuyte_count"
## [49] "sleep_duration"                       "height"
## [51] "subject_id"                           "triglycerides"
## [53] "cancer_type_idc10"                    "cancer_type_idc9"
## [55] "weight"                               "leukocyte_count"
## [57] "sex"                                  "phenotype"
## [59] "age_death"                            "age_diagnosis_diabetes"
## [61] "date_k85_report"                      "date_k86_report"
## [63] "date_death"                           "enm_diseases"
## [65] "source_k85_report"                    "source_k86_report"
## [67] "age_hbp_diagnosis"                    "mental_disorders"
## [69] "respiratory_disorders"                "cancer_year"
## [71] "circulatory_disorders"                "digestive_disorders"
## [73] "nervous_disorders"                    "immune_disorders"
```

4.1.4 Merge VCF (genotype) and phenotype information

Arrived at this point, we have 21 VCF objects (each one corresponds to a chromosome) and a phenotypes table on our remote session. The next step is merging each of the VCF objects with the phenotypes table. Before doing that however, we have to gather some information from the phenotypes table. The information to gather is summarized on the Table **??**.

Table 4.2: Information to gather from the phenotypes table.

Information	Details
Which column has the samples identifier?	Column name that contains the IDs of the samples. Those are the IDs that will be matched to the ones present on the VCF objects.
Which is the sex column on the covariates file?	Column name that contains the sex information.
How are males and how are females encoded into this column?	There is not a standard way to encode the sex information, some people use 0/1; male/female; etc. Our approach uses a library that requires a specific encoding, for that reason we need to know the original encoding to perform the translation.
Which is the case-control column?	Column name that contains the case/control to be studied.

Information	Details
What is the encoding for case and for control on this column?	Similar as the sex column, the case/control will be translated to the particular standard of the software we have used to develop our functionalities [3] .

If we are not sure about the exact column names, we can use the function `ds.colnames` as we did before. Also, we can use the function `ds.table1D` to check the level names of the categorical variables.

```
ds.table1D("pheno_object$sex")$counts
```

```
##        pheno_object$sex
## female             1271
## male               1233
## Total              2504
```

```
ds.table1D("pheno_object$diabetes_diagnosed_doctor")$counts
```

```
##                      pheno_object$diabetes_diagnosed_doctor
## Do_not_know                                             612
## No                                                      632
## Prefer_not_to_answer                                    590
## Yes                                                     668
## Total                                                  2502
```

From the different functions we used, we can extract our answers (Summarized on the Table **??**)

Table 4.3: Summary of options from the example phenotypes table.

Question	Answer
Which column has the samples identifier?	“subject_id”
Which is the sex column on the covariates file?	“sex”
How are males and how are females encoded into this column?	“male”/“female”
Which is the case-control column of interest of the covariates?	“diabetes_diagnosed_doctor”
What is the encoding for case and for control on this column?	“Yes”/“No”

[3]It’s important to note that all the other encodings present on the case-control column that are not the case or control, will be turned into missings. As an example, if we have `"cancer"/"no cancer"/"no information"/"prefer not to answer"` we can specifcy `case="cancer"`, `control="no cancer"` and all the individuals with `"no information"/"prefer not to answer"` will be interpreted as missings.

With all this information we can now merge the phenotypes and VCF objects into a type of object named GenotypeData. We will use the `ds.GenotypeData` function.

```
lapply(1:21, function(x){
  ds.GenotypeData(x=paste0('gds', x,'_object'), covars = 'pheno_object', columnId = "su
                  sexId = "sex", male_encoding = "male", female_encoding = "female",
                  case_control_column = "diabetes_diagnosed_doctor", case = "Yes", con
                  newobj.name = paste0('gds.Data', x), datasources = conns)
})
```

The objects we just created are named `gds.DataXX` on the remote session. Now we are ready to perform a GWAS analysis. To perform a GWAS we have to supply this "Genotype Data" objects and some sort of formula in which we will specify the type of association we are interested on studying. The R language has it's own way of writing formulas, a simple example would be the following (relate the `condition` to the `smoke` variable adjusted by `sex`):

```
condition ~ smoke + sex
```

For this use case we will use the following formula.

```
diabetes_diagnosed_doctor ~ sex + hdl_cholesterol
```

4.1.5 Performing the GWAS

We are finally ready to achieve our ultimate goal, performing a GWAS analysis. We have our data (genotype + phenotypes) organized into the correspondent objects (GenotypeData) and our association formula ready. The function `ds.metaGWAS` is in charge of doing the analysis, we have to supply it with the names of the GenotypeData objects of the remote R session and the formula. Since we have 21 different objects, the `paste0` function is used to simplify the call.

```
results_single <- ds.metaGWAS(genoData = paste0("gds.Data", 1:21), model = diabetes_dia

results_single
```

```
## # A tibble: 865,240 x 10
##    rs        chr        pos n.obs  freq   p.value    Est Est.SE reference_allele
##  * <chr>     <chr>    <int> <dbl> <dbl>     <dbl>  <dbl>  <dbl> <chr>
##  1 rs4648446 1       2.89e6  1292 0.512   7.42e-6 -0.323 0.0721 C
##  2 rs4659014 1       1.20e8  1296 0.587   8.74e-6  0.321 0.0723 T
##  3 rs127465~ 1       2.90e6  1294 0.548   9.69e-6 -0.321 0.0725 C
##  4 rs313728  1       8.62e7  1293 0.277   1.08e-5 -0.375 0.0852 C
##  5 rs113272~ 1       1.17e8  1292 0.859   1.08e-5 -0.459 0.104  T
##  6 rs173625~ 1       2.82e7  1293 0.864   2.26e-5 -0.466 0.110  C
##  7 rs292001  1       2.30e7  1293 0.343   2.64e-5  0.327 0.0779 G
```