ShinyDataSHIELD User's Guide

Escribà Montagut, Xavier; González, Juan R.

2020-10-26

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

Setup

In this section, a list of all the R packages required to launch ShinyDataSHIELD is given. Following it, two methodologies to install them is provided. The recommended methodology is to install all the packages using RStudio and launch the Shiny app from there. However, depending on the version of R and other variables that can't be contemplated on a case to case basis, the user might have problems installing the R packages, if that is the case, an alternative to use ShinyDataSHIELD has been created using Docker.

2.1 R packages

The packages that contain the main functionalities of ShinyDataSHIELD are the following.

- DSI: The DataSHIELD Interface (DSI) handles the connections to the databases.
- DSOpal: DSOpal is an extension of DSI to connecto to Opal servers.
- dsBaseClient: Implementation of the base package R functions to obtain non-disclosive returns from the Opal servers (Example: Base package function `as.factor` is implemented as `ds.asFactor`).
- dsOmicsClient: Functions to perform non-disclosive omic analysis using BioConductor packages on the Opal servers.

On the following table, all the packages required by ShinyDataSHIELD are shown along the versions used during development.

Package	Version
DSI	1.1.0
DSOpal	1.1.0
dsBaseClient	6.0.1

Package	Version
dsOmicsClient	1.0.0
shinydashboard	0.7.1
shiny	1.4.0.2
shinyalert	1.1
DT	0.13
data.table	1.12.8
shinyjs	1.1
shinyBS	0.61
shinycssloaders	0.3
shinyWidgets	0.5.4
stringr	1.4.0

2.2 Install

2.2.1 Install with RStudio

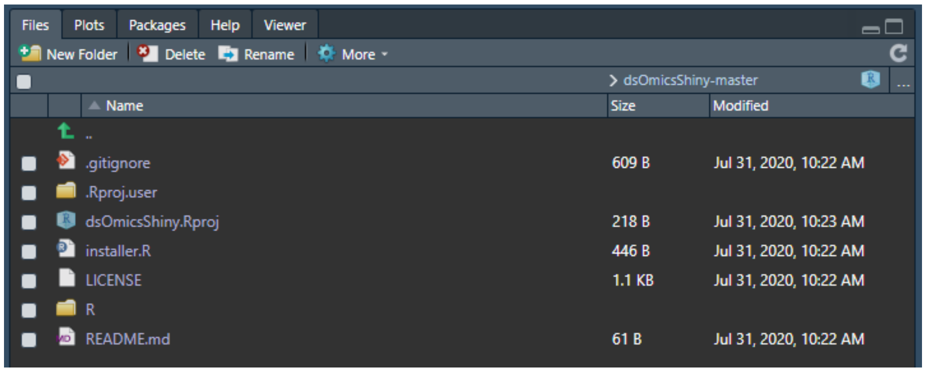
The first step to use ShinyDataSHIELD via RStudio is to install all the required R packages. To do so, run the following lines on the RStudio console.

```
# Install the devtools package
install.packages("devtools")
# Install the R packages
devtools::source_url("https://raw.githubusercontent.com/isglobal-brge/dsOmicsShiny/mas
```

Once all the packages are installed, download and setup the environment to launch ShinyDataSHIELD, the latest version of the software has to be downloaded from GitHub. To do so, open a new RStudio session and run the following code on the console.

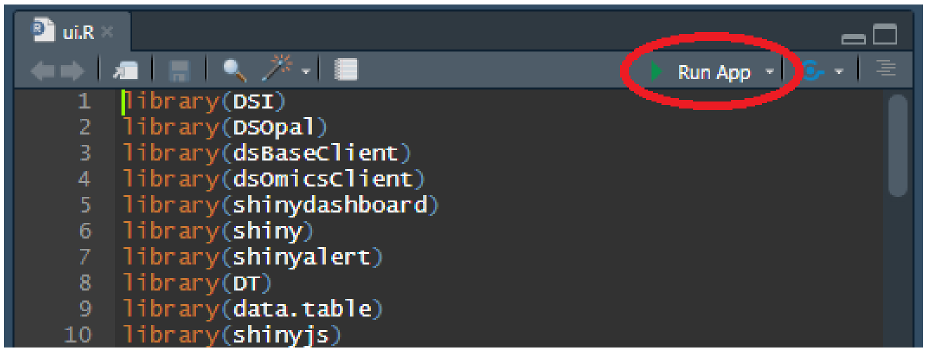
```
  # Set working directory
setwd(dir = "/some/path/")

  # Download zip
download.file(url = "https://github.com/isglobal-brge/ShinyDataSHIELD/archive/master.z

  # Unzip the .zip to the working directory
unzip(zipfile = "master.zip")

  # Set the working directory inside the downloaded folder
setwd(dir = "/some/path/ShinyDataSHIELD-master")
```

Now all the source files are downloaded to the location chosen and the working directory moved to the correct folder, to start the project, open the `Rproj` file by clicking it on the Files explorer of RStudio.



Name	Size	Modified
..		
.gitignore	609 B	Jul 31, 2020, 10:22 AM
.Rproj.user		
dsOmicsShiny.Rproj	218 B	Jul 31, 2020, 10:23 AM
installer.R	446 B	Jul 31, 2020, 10:22 AM
LICENSE	1.1 KB	Jul 31, 2020, 10:22 AM
R		
README.md	61 B	Jul 31, 2020, 10:22 AM

Now everything is ready to launch the Shiny application. To do so there a two approaches, one is to open the `ui.R` or the `server.R` files that are inside the `R` folder and press `Run App`.



```
library(DSI)
library(DSOpal)
library(dsBaseClient)
library(dsOmicsClient)
library(shinydashboard)
library(shiny)
library(shinyalert)
library(DT)
library(data.table)
library(shinyjs)
```

The other option is to input the following command on the RStudio console.

```
shiny::runApp('R')
```

2.2.2 Install with Docker

Another option to use ShinyDataSHIELD is to install it using Docker. Docker can be installed on a Linux / Mac OS X machine without any complications as any other application, on Windows systems however it can be a little bit more troubling, there are many online resources to help. Please refer to the following links to install Docker on Windows Home, to setup the Linux Windows Subsystem and terminal and to execute Docker on Windows.

Once Docker is up and running, execute the following command on a bash terminal (make sure Docker is running, if not search for the `Docker Desktop` app and launch it) to download and launch ShinyDataSHIELD. Be aware that the Docker images weights ~ 1.5 GB, so if your internet connection is slow it may take a while.

```
docker run --rm -p 80:80 brgelab/shiny-data-shield
```

The container will be exposed on the local port 80 and it will render on that port the application itself, so to start using ShinyDataSHIELD open your web browser of choice and go to the site

```
localhost:80
```

At the beginning it may take some time for the application to render, this is because all the needed R libraries are being loaded, to be sure the container is actually working, take a look at the terminal where you inputed the Docker command, there you will see all the R verbose stating the libraries are being loaded.

Once the user has finished using ShinyDataSHIELD, the container needs to be stopped to avoid wasting CPU resources, to do so, input the following command on a bash terminal (the command needs to be inputed on a new bash window):

```
docker container ls
```

This will prompt all the running containers, find the one with the NAMES `brgelab/shiny-data-shield` and copy it's CONTAINER ID, then input the following bash command:

```
docker stop xxxxxxxxxxxx
```

Where xxxxxxxxxxxx is the CONTAINER ID.

To run the application again, just enter the first bash command (`docker run --rm -p 80:80 brgelab/shiny-data-shield`), since it has already been downloaded, the application is cached on the computer and it will launch straight away. If the user wants to remove the Docker image from the computer, input the following bash command:

```
docker image rm brgelab/exposome-shiny
```

If the user wants to download the actual source code of the Shiny, install all the required packages and launch it locally on it's machine, feel free to download it from Github. There's a script called `installer.R` at the root of the repository with a short installer of all the required packages. Please note that the installer script may fail depending on the R version and others, for that reason is advised to always run the Docker version of ShinyDataSHIELD, as it only requires a single terminal command and will work no matter what.

Chapter 3

Functionalities

Along this section, an overview of the functionalities implemented on Shiny-DataSHIELD is given. There's information about how to use the funcionalities as well as some limitations or constraints to take into account when using Shiny-DataSHIELD.

3.1 Data entry

DISCLAIMER
Along this section the terms **table** and **resource** are widely used, it is important noting that when the autor talks about a table, it refers to what is shown as a table on the Opal server. A resource that holds a table is called (and treated) as a resource.

The first step to any analysis is to load the required tables or resources to the study server(s). To do so, the user has to provide the server URL and the login credentials. This will allow the application to connect to the OPAL server and retrieve all the projects and resources / tables from them. Afterwards, the user can select the desired resources / tables and load them to the study servers.

There are some limitations to how the data entry is coded in ShinyDataSHIELD:

- When loading multiple tables / resources on a study server, it is assumed only tables are selected or only resources, not combined.
- When dealing with tables, there can only be one table per study.
- A pooled study can only contain one table (or resource) per study server.
- When performing a pooled study, consistency among the tables is assumed, meaning all tables share the same columns.

- When loading multiple resources per study server, only one study is allowed.

The limitations can be overcome by changing the part of the code in charge of the data entry. Look at the developers guide flowcharts to get some references on which parts of the code could be altered to change the current limitations.

When loged into the Opal server, the interface shows two selectors, one is a dropdown with all the projects found on the server, the other is a dropdown (with multiple selection) with the resources / tables inside the selected project. The user has to select the project and resource(s) / table(s) that wants to load on a study server and press the “Add study” button.

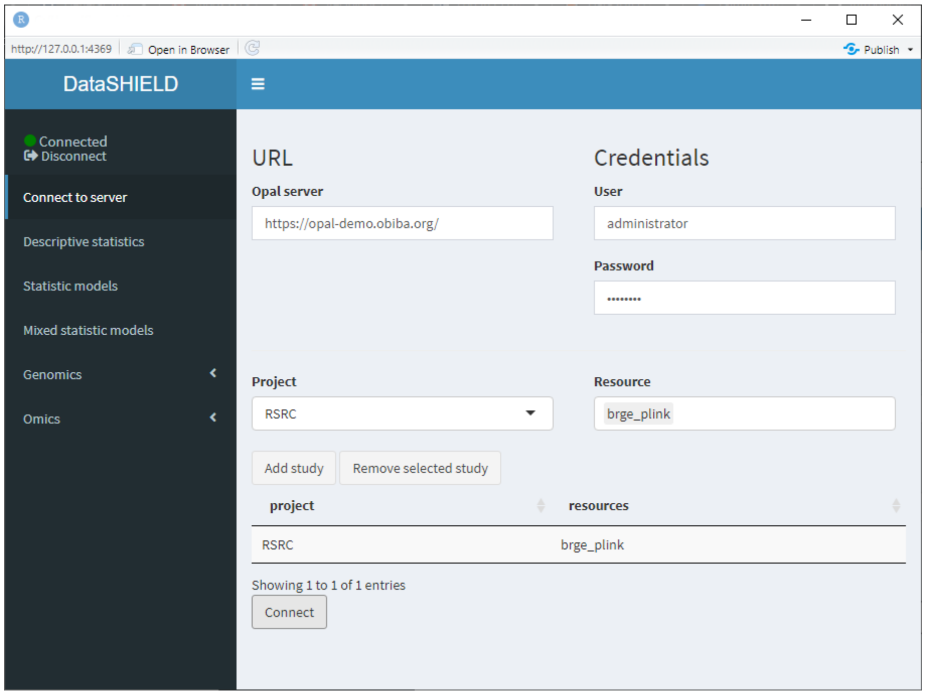
Below the two selectors a table will be shown, on this table each line corresponds to a study server. If the user made a mistake when adding a study server, the rows on this table can be selected, and by pressing the “Remove selected study” button, the selected study servers will be removed.

When the desired study serveres are selected, press the “Connect” button to connect to them.

Following the limitations explained before, there are three different use cases.

3.1.1 Single resource / table approach

To perform analysis on data that is contained on a single resource / table of the server, the user has to select it from the dropdown inputs, add it to the study and connect to it.



URL

Opal server: https://opal-demo.obiba.org/

Credentials

User

administrator

Password: ••••••••

Project: RSRC

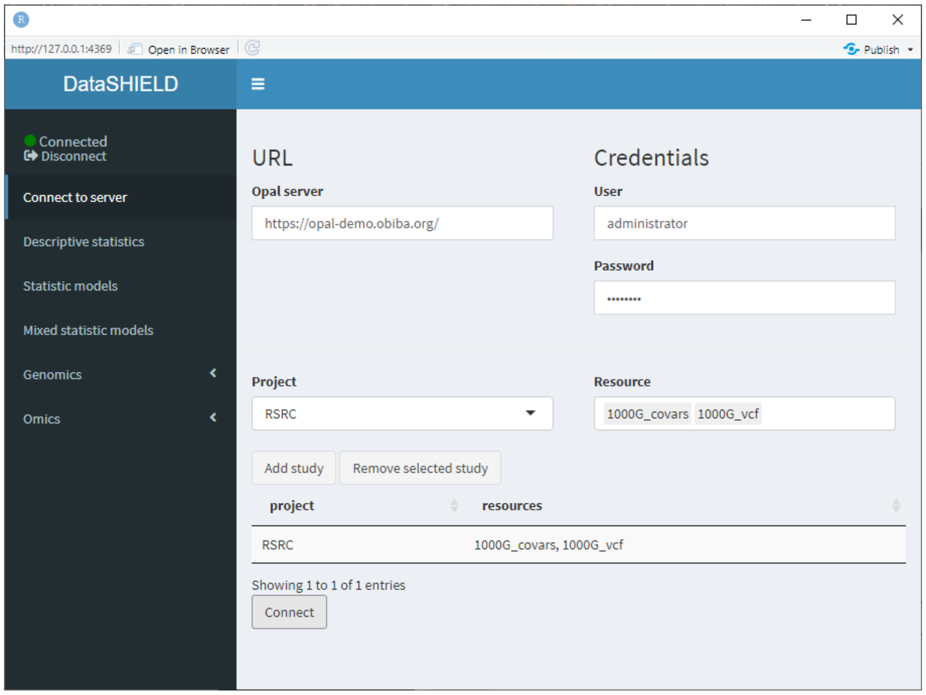
Resource: brge_plink

project	resources
RSRC	brge_plink

Showing 1 to 1 of 1 entries

3.1.2 Multiple resource study

Some studies require to put more than one resource on a study server, this is the case for example of using VCF files to perform a GWAS; they require two resources, the VCF resource and the covariates resource (which is a resource that holds a plain table). For this use case, the user has to select the multiple resources from the dropdown inputs, add them to a single study and connect to it.



URL

Opal server: https://opal-demo.obiba.org/

Credentials

User: administrator

Password: ••••••••

Project: RSRC

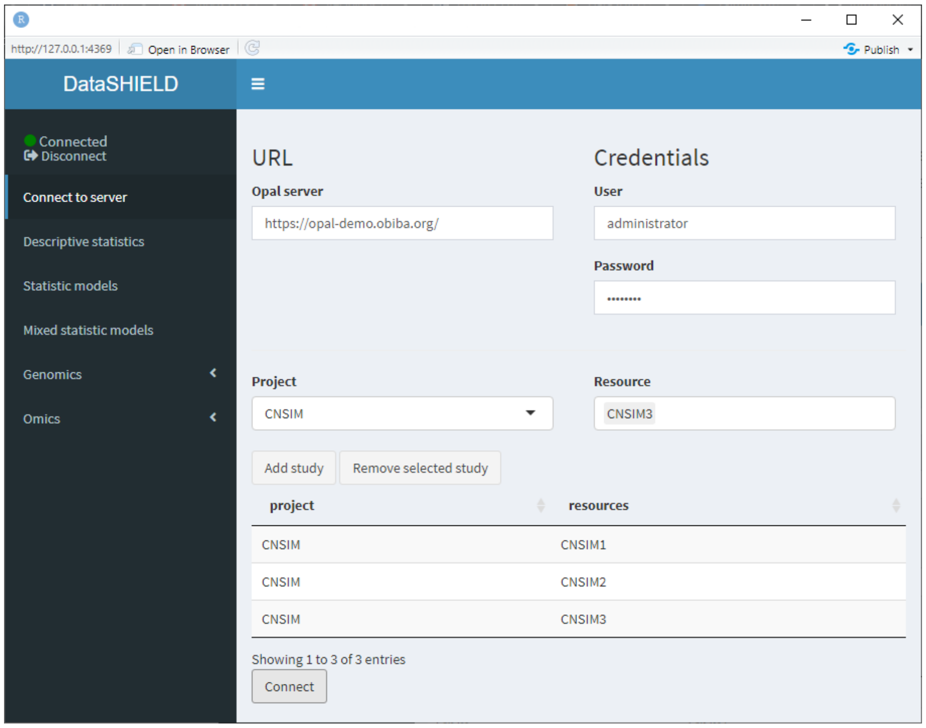
Resource: 1000G_covars 1000G_vcf

project	resources
RSRC	1000G_covars, 1000G_vcf

Showing 1 to 1 of 1 entries

3.1.3 Pooled data approach

Some studies have the data distributed across multiple tables or resources that need to be pooled to perform a study. To load this kind of studies, the user has to select each different table / resource, add it to a study separately and connect to it.



Opal server: https://opal-demo.obiba.org/
User: administrator
Password: ••••••••
Project: CNSIM
Resource: CNSIM3

project	resources
CNSIM	CNSIM1
CNSIM	CNSIM2
CNSIM	CNSIM3

Showing 1 to 3 of 3 entries

3.2 Descriptive statistics

The descriptive statistics functionality is available for tables as well as the following resource types:

- SQL tables
- Tidy data files (tables): `*.csv`, `*.tsv`, etc
- ExpressionSets
- RangedSummarizedExperiments

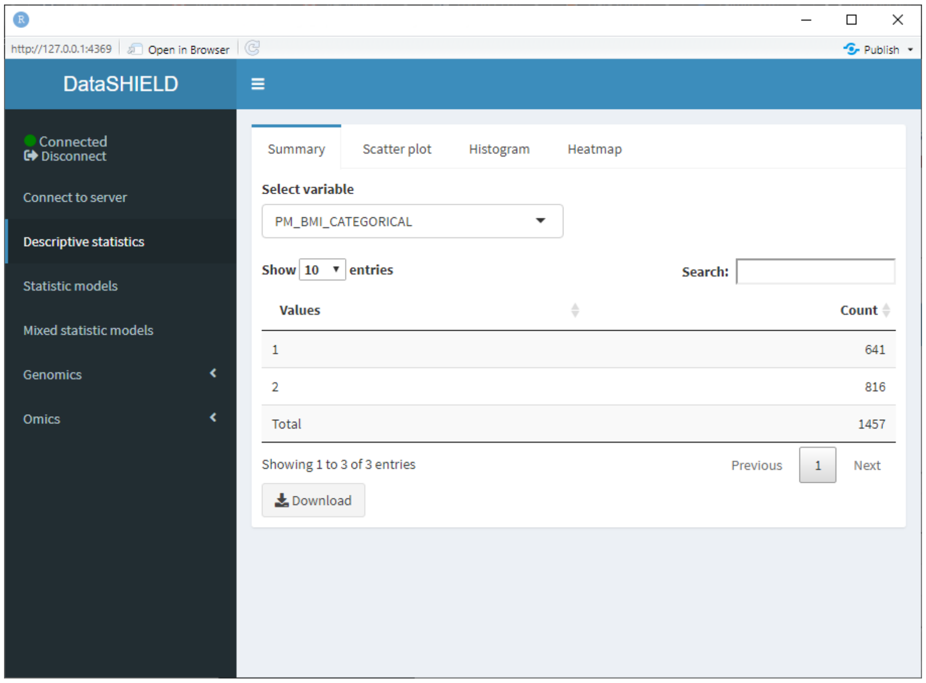
When using pooled data the descriptive statistics is by default of the pooled data, however, the graphical visualizations included on descriptive statistics provide the option of showing separated plots for the different studies.

The download button will prompt a system window to select where to store the shown table, it will save it as a `*.csv`.

3.2.1 Summary

The summary provides non-disclosive insights on the different variables of the loaded data. This functionality is only available for factors and numeric variables, otherwise no table will be rendered. When the desired summary is disclosive no table will be shown (as the function call returns an Error stating that the the return is disclosive).

When the selected variable is a factor, the output shown is a count of all the different factors.



Select variable

PM_BMI_CATEGORICAL

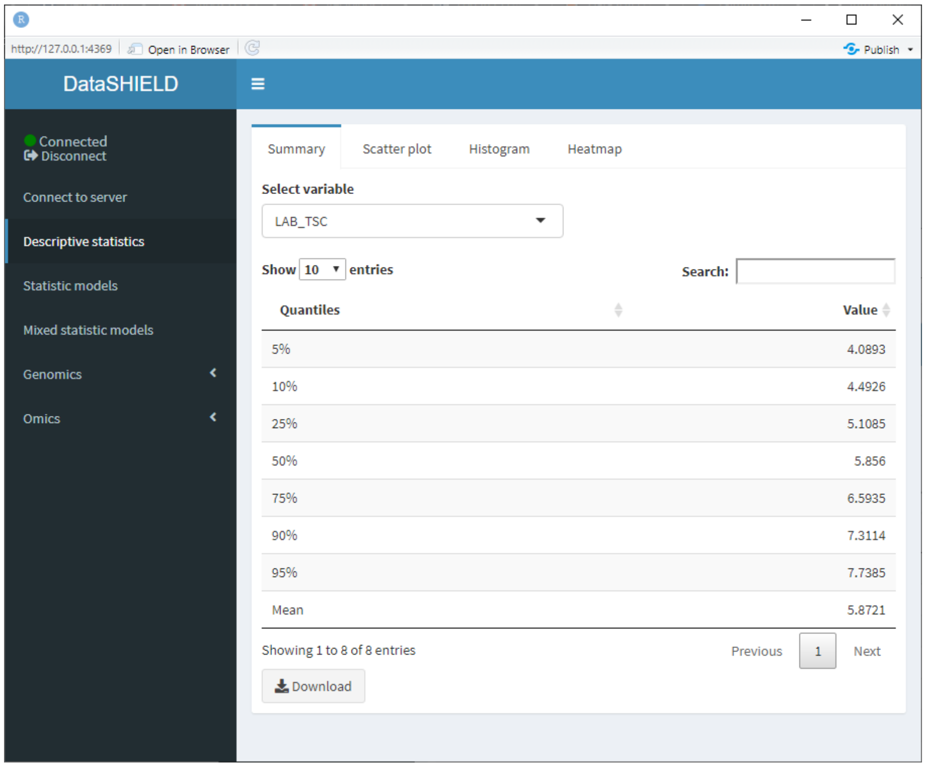
Show 10 entries

Search:

Values	Count
1	641
2	816
Total	1457

Showing 1 to 3 of 3 entries

When the selected variable is numerical, the output shown is a quantiles and mean table.



Select variable: LAB_TSC

Show 10 entries

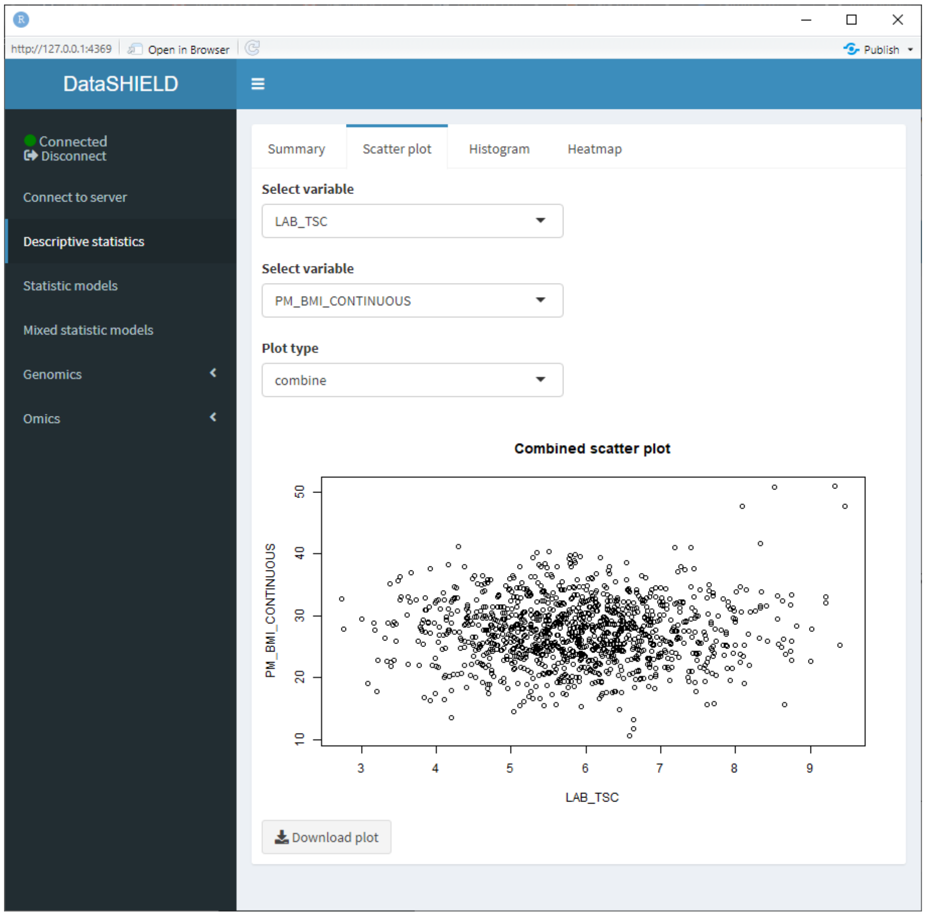
Quantiles	Value
5%	4.0893
10%	4.4926
25%	5.1085
50%	5.856
75%	6.5935
90%	7.3114
95%	7.7385
Mean	5.8721

Showing 1 to 8 of 8 entries

3.2.2 Scatter plot

Create a non-disclosive scatter plot by selecting two numerical variables (one for each axis). There's the option of obtaining a pooled data plot or a different plot for each study server. The selected variables have to be numerical, otherwise the plot is not displayed and a warning message appears.

The dowload plot button saves the shown figure as a `*.png`.



Select variable
LAB_TSC

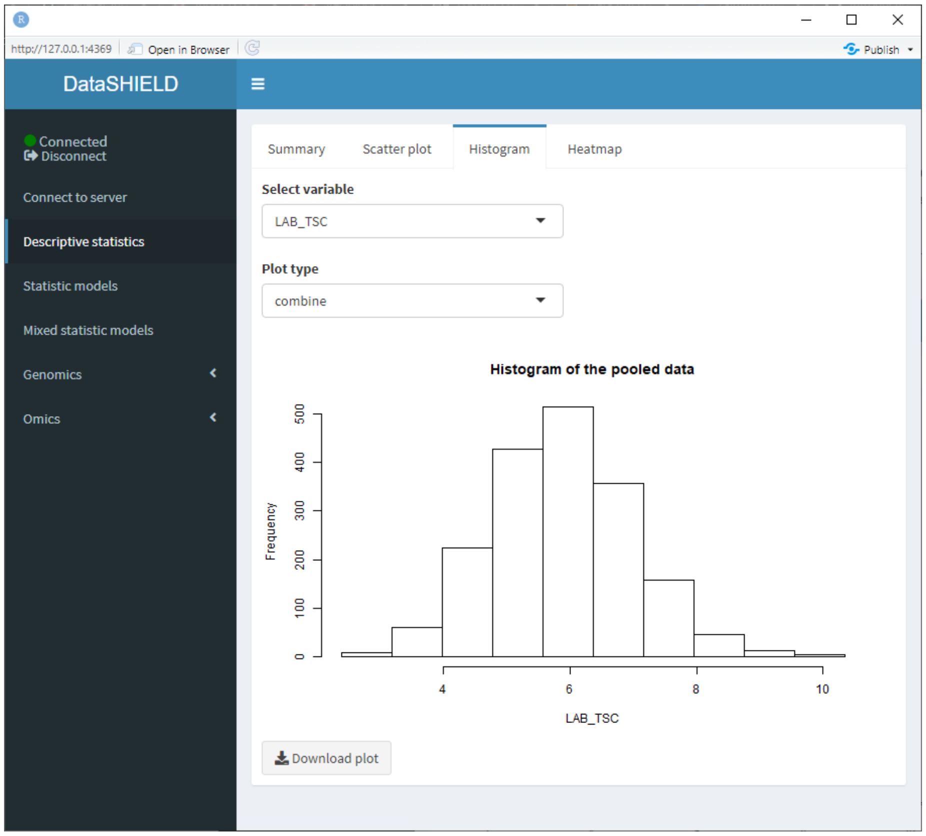
Select variable
PM_BMI_CONTINUOUS

Plot type
combine

3.2.3 Histogram

Create a non-disclosive histogram of a selected variable. There's the option of obtaining a pooled data plot or a different plot for each study server. The selected variable has to be numerical, otherwise the plot is not displayed and a warning message appears.

The dowload plot button saves the shown figure as a `*.png`.



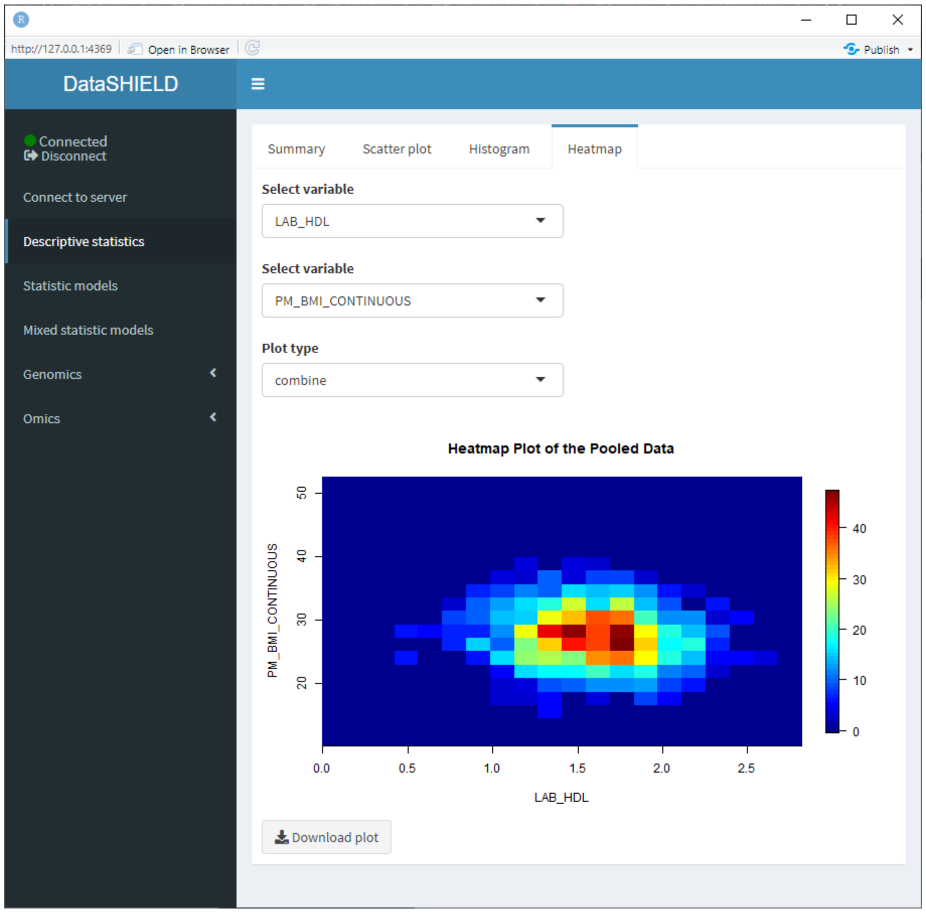
Select variable
LAB_TSC

Plot type
combine

3.2.4 Heatmap

Create a non-disclosive heatmap plot by selecting two numerical variables (one for each axis). There's the option of obtaining a pooled data plot or a different plot for each study server. The selected variables have to be numerial, otherwise the plot is not displayed and a warning message appears.

The dowload plot button saves the shown figure as a `*.png`.



Select variable: LAB_HDL

Select variable: PM_BMI_CONTINUOUS

Plot type: combine

3.3 Statistic models

Statistic models are available for tables as well as the following resource types:

- SQL tables
- Tidy data files (tables): `*.csv`, `*.tsv`, etc

There are two different statisticals models available to fit, GLM models (Statistics models tab) and GLMer models (Mixed statistical models tab).

3.3.1 GLM models

The tab to fit a non-disclosive generalized linear models (GLM) contains a box to manually input the formula, a selector for the output family and a table displaying the variables of the data and the type of each variable. The possible output families are:

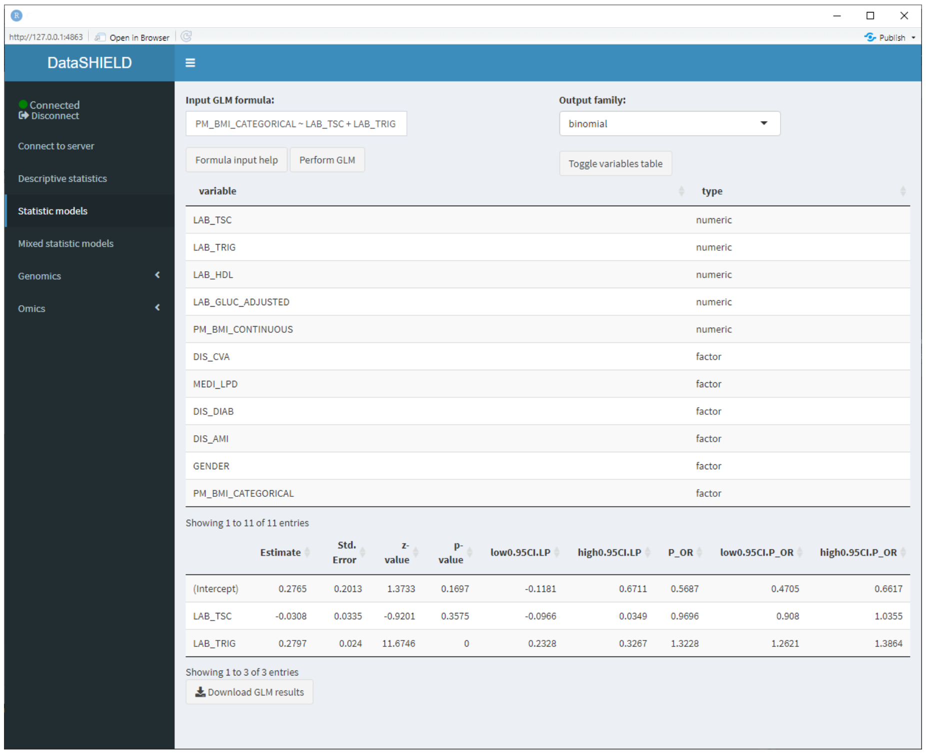
- Gaussian
- Poisson
- Binomial

There's some help built into ShinyDataSHIELD regarding how to write the GLM formula, which is prompted to the user when clicking on the "Formula input help" button. The display of the variables can be toggled on and off for the convenience of use.

Once the GLM model is fitted a table below the variables display will be rendered with the model results. The download button will prompt a system window to select where to store the shown table, it will save it as a `*.csv`.

When using pooled data, the results of the GLM model will be of the combined data.

(To do: Display more information of why a model fitment fails)



Input GLM formula: PM_BMI_CATEGORICAL ~ LAB_TSC + LAB_TRIG

Output family: binomial

variable	type
LAB_TSC	numeric
LAB_TRIG	numeric
LAB_HDL	numeric
LAB_GLUC_ADJUSTED	numeric
PM_BMI_CONTINUOUS	numeric
DIS_CVA	factor
MEDI_LPD	factor
DIS_DIAB	factor
DIS_AMI	factor
GENDER	factor
PM_BMI_CATEGORICAL	factor

Showing 1 to 11 of 11 entries

	Estimate	Std. Error	z-value	p-value	low0.95CI.LP	high0.95CI.LP	P_OR	low0.95CI.P_OR	high0.95CI.P_OR
(Intercept)	0.2765	0.2013	1.3733	0.1697	-0.1181	0.6711	0.5687	0.4705	0.6617
LAB_TSC	-0.0308	0.0335	-0.9201	0.3575	-0.0966	0.0349	0.9696	0.908	1.0355
LAB_TRIG	0.2797	0.024	11.6746	0	0.2328	0.3267	1.3228	1.2621	1.3864

Showing 1 to 3 of 3 entries

3.3.2 Mixed models

The tab to fit non-disclosive generalized mixed effects models (GLMer) contains a box to manually input the formula, a selector for the output family and a table displaying the variables of the data and the type of each variable. The possible output families are:

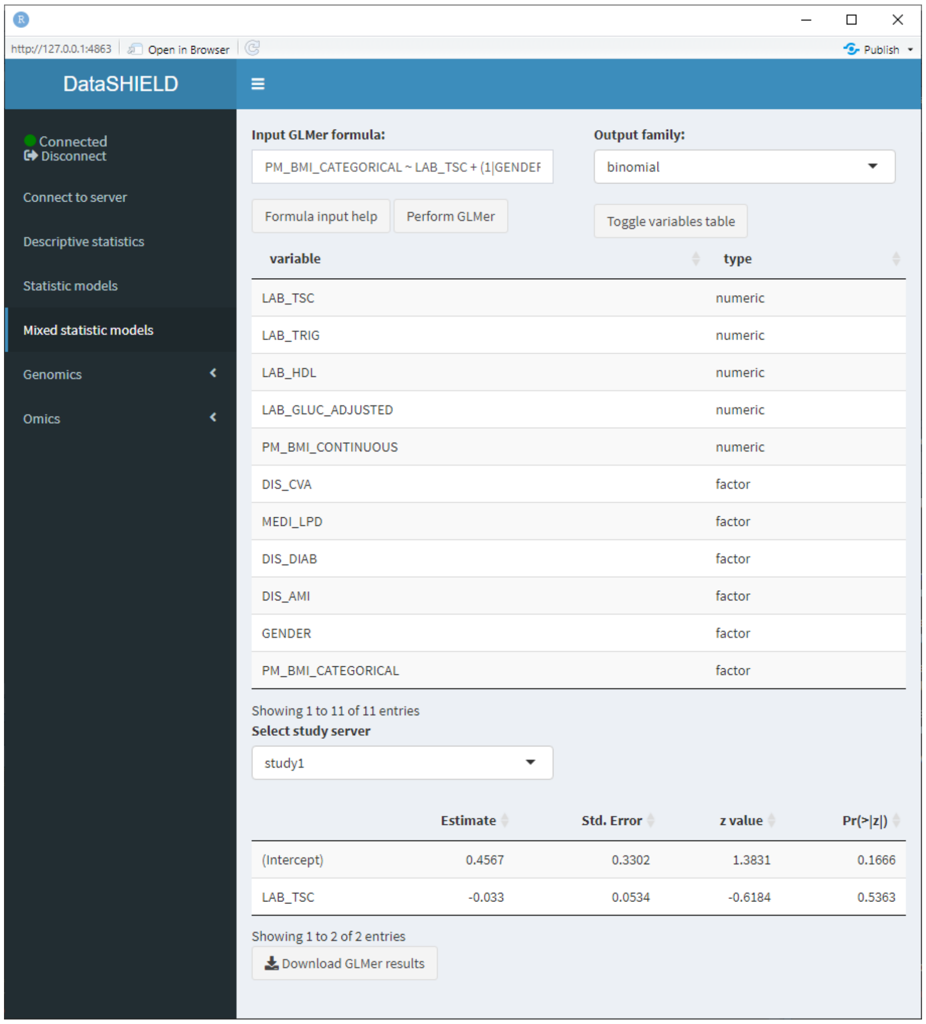
- Poisson
- Binomial

There's some help built into ShinyDataSHIELD regarding how to write the GLMer formula, which is prompted to the user when clicking on the "Formula input help" button. The display of the variables can be toggled on and off for the convenience of use.

Once the GLMer model is fitted a table below the variables display will be rendered displaying the results. The download button will prompt a system window to select where to store the shown table, it will save them as a `*.csv`.

The mixed model results are independent for each study server. There's a selector to toggle between the results of the different study servers.

(To do: Display more information of why a model fitment fails)



Input GLMer formula: PM_BMI_CATEGORICAL ~ LAB_TSC + (1|GENDEF

Output family: binomial

variable	type
LAB_TSC	numeric
LAB_TRIG	numeric
LAB_HDL	numeric
LAB_GLUC_ADJUSTED	numeric
PM_BMI_CONTINUOUS	numeric
DIS_CVA	factor
MEDI_LPD	factor
DIS_DIAB	factor
DIS_AMI	factor
GENDER	factor
PM_BMI_CATEGORICAL	factor

Showing 1 to 11 of 11 entries

Select study server: study1

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	0.4567	0.3302	1.3831	0.1666
LAB_TSC	-0.033	0.0534	-0.6184	0.5363

Showing 1 to 2 of 2 entries

3.4 Genomics

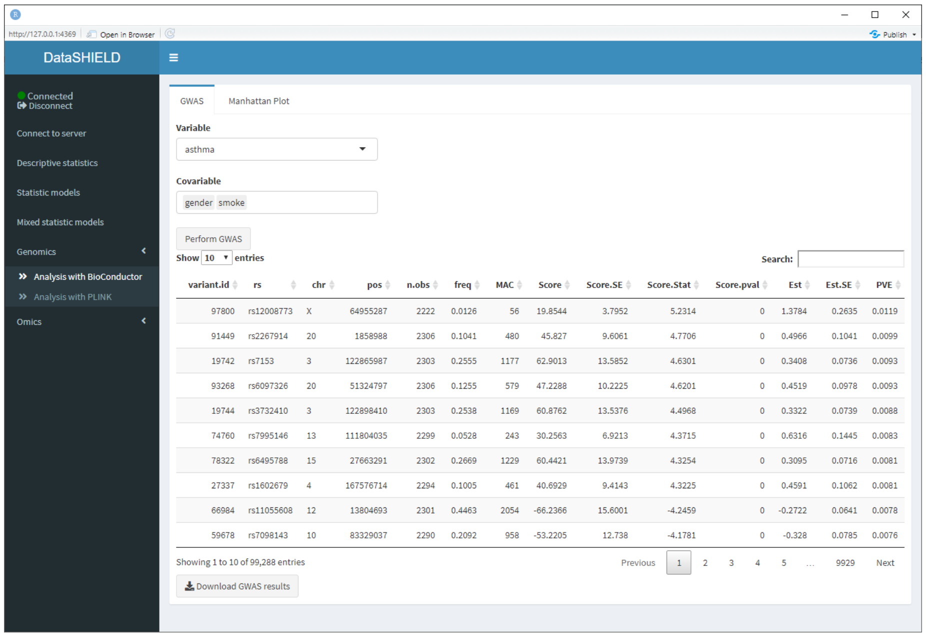
Inside the genomics tab of dsOmicshiny there are two subtabs, one to perform analysis using BioConductor methods and another to perform analysis using PLINK methods.

3.4.1 Analysis with BioConductor

To perform non-disclosive genomic analysis using BioConductor methodologies, the user has to input a VCF resource with a covariates resource (table) on the same study.

When performing this kind of analysis, as explained on the Data Entry section, only one study server can be used.

The Analysis with BioConductor has two sub-tabs, the first one corresponds to the GWAS, and as the name implies is used to perform a GWAS (Genome wide association study) non-disclosive analysis on the loaded data. There is a selector for the condition and the covariates to adjusted for. The fitted model is: snp ~ condition + covar1 + ... + covarN. The results of the model appear on a table below the selectors. The download button will prompt a system window to select where to store the shown table, it will save it as a `*.csv`. The second subtab is to display a Manhattan plot of the GWAS results. The dowload plot button saves the shown figure as a `*.png`.



Variable: asthma

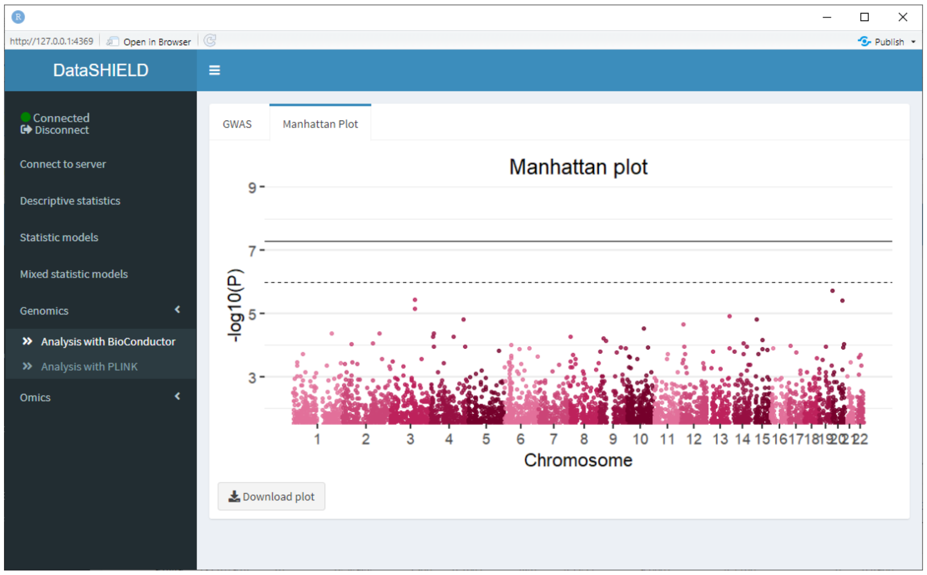
Covariable: gender smoke

Show 10 entries

Search:

variant.id	rs	chr	pos	n.obs	freq	MAC	Score	Score.SE	Score.Stat	Score.pval	Est	Est.SE	PVE
97800	rs12008773	X	64955287	2222	0.0126	56	19.8544	3.7952	5.2314	0	1.3784	0.2635	0.0119
91449	rs2267914	20	1858988	2306	0.1041	480	45.827	9.6061	4.7706	0	0.4966	0.1041	0.0099
19742	rs7153	3	122865987	2303	0.2555	1177	62.9013	13.5852	4.6301	0	0.3408	0.0736	0.0093
93268	rs6097326	20	51324797	2306	0.1255	579	47.2288	10.2225	4.6201	0	0.4519	0.0978	0.0093
19744	rs3732410	3	122898410	2303	0.2538	1169	60.8762	13.5376	4.4968	0	0.3322	0.0739	0.0088
74760	rs7995146	13	111804035	2299	0.0528	243	30.2563	6.9213	4.3715	0	0.6316	0.1445	0.0083
78322	rs6495788	15	27663291	2302	0.2669	1229	60.4421	13.9739	4.3254	0	0.3095	0.0716	0.0081
27337	rs1602679	4	167576714	2294	0.1005	461	40.6929	9.4143	4.3225	0	0.4591	0.1062	0.0081
66984	rs11055608	12	13804693	2301	0.4463	2054	-66.2366	15.6001	-4.2459	0	-0.2722	0.0641	0.0078
59678	rs7098143	10	83329037	2290	0.2092	958	-53.2205	12.738	-4.1781	0	-0.328	0.0785	0.0076

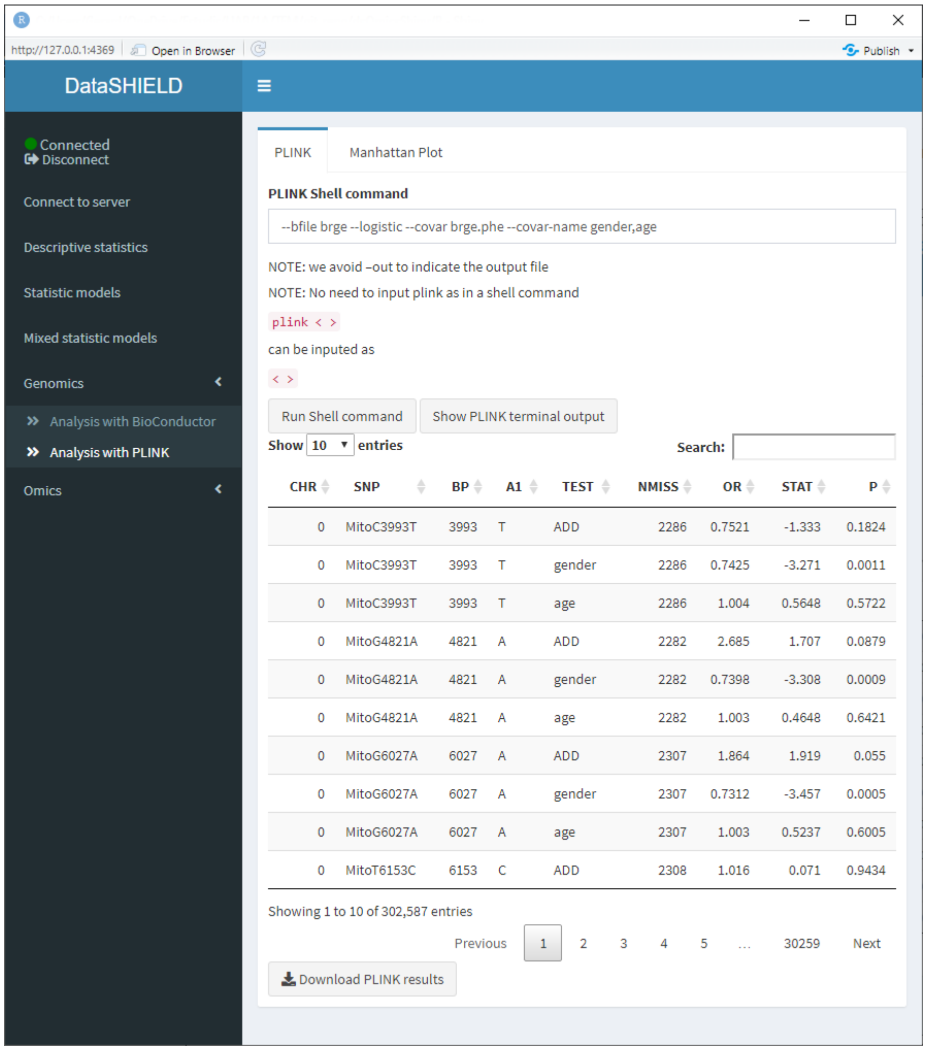
Showing 1 to 10 of 99,288 entries



3.4.2 Analysis with PLINK

To perform non-disclosive analysis using PLINK commands, the user has to load a SSH resource. The tab contains a field to input the PLINK command and a brief memo stating that when inputing the PLINK command to run there is no need of inputting it as `plink ...` as would be done on a terminal interface, the user has to input just the `...`; also, there is no need to put `-out` to indicate the output file.

Once the command is run, a table with the results is displayed under the command input, the download button will prompt a system window to select where to store the shown table, it will save them as a `*.csv`. A button to display the raw terminal output appears to display the user on a popup the plain text.



PLINK Shell command

```
--bfile brge --logistic --covar brge.phe --covar-name gender,age
```

NOTE: we avoid –out to indicate the output file

NOTE: No need to input plink as in a shell command

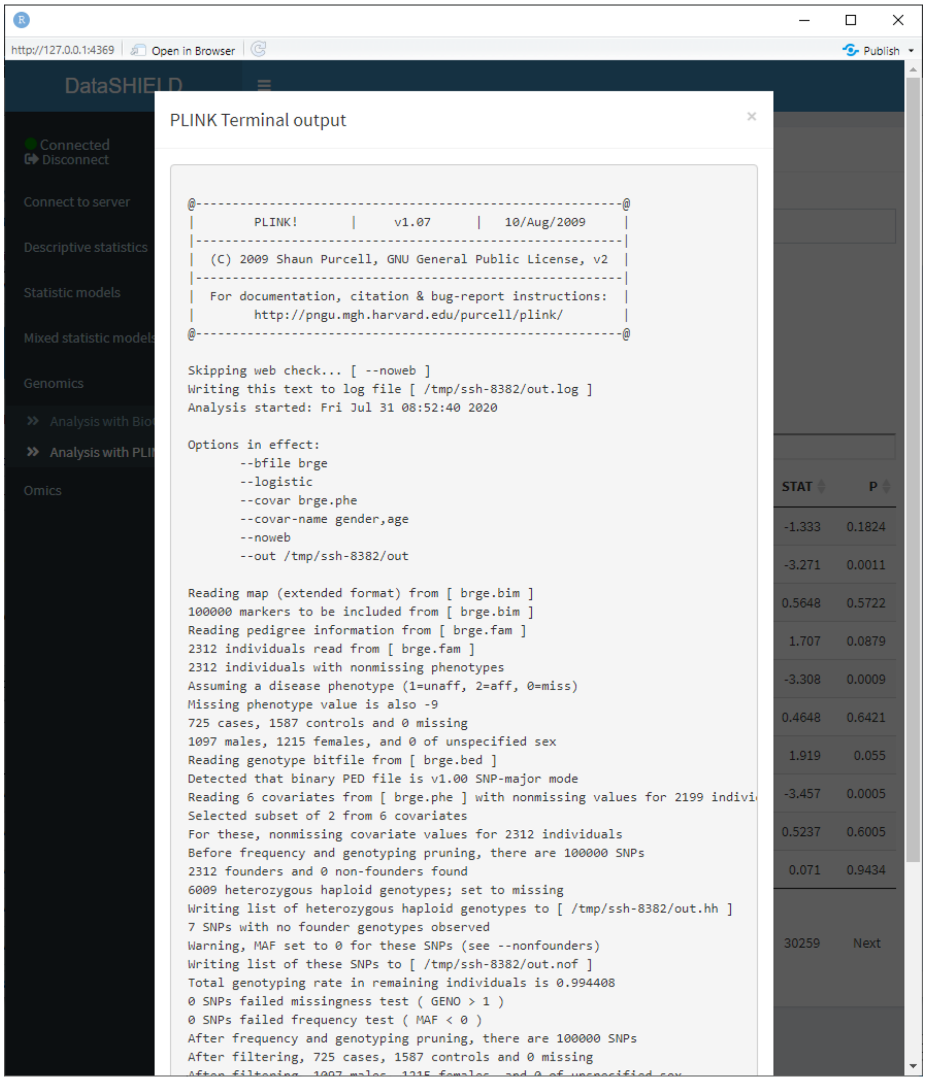
`plink < >`

can be inputed as

`< >`

CHR	SNP	BP	A1	TEST	NMISS	OR	STAT	P
0	MitoC3993T	3993	T	ADD	2286	0.7521	-1.333	0.1824
0	MitoC3993T	3993	T	gender	2286	0.7425	-3.271	0.0011
0	MitoC3993T	3993	T	age	2286	1.004	0.5648	0.5722
0	MitoG4821A	4821	A	ADD	2282	2.685	1.707	0.0879
0	MitoG4821A	4821	A	gender	2282	0.7398	-3.308	0.0009
0	MitoG4821A	4821	A	age	2282	1.003	0.4648	0.6421
0	MitoG6027A	6027	A	ADD	2307	1.864	1.919	0.055
0	MitoG6027A	6027	A	gender	2307	0.7312	-3.457	0.0005
0	MitoG6027A	6027	A	age	2307	1.003	0.5237	0.6005
0	MitoT6153C	6153	C	ADD	2308	1.016	0.071	0.9434

Showing 1 to 10 of 302,587 entries

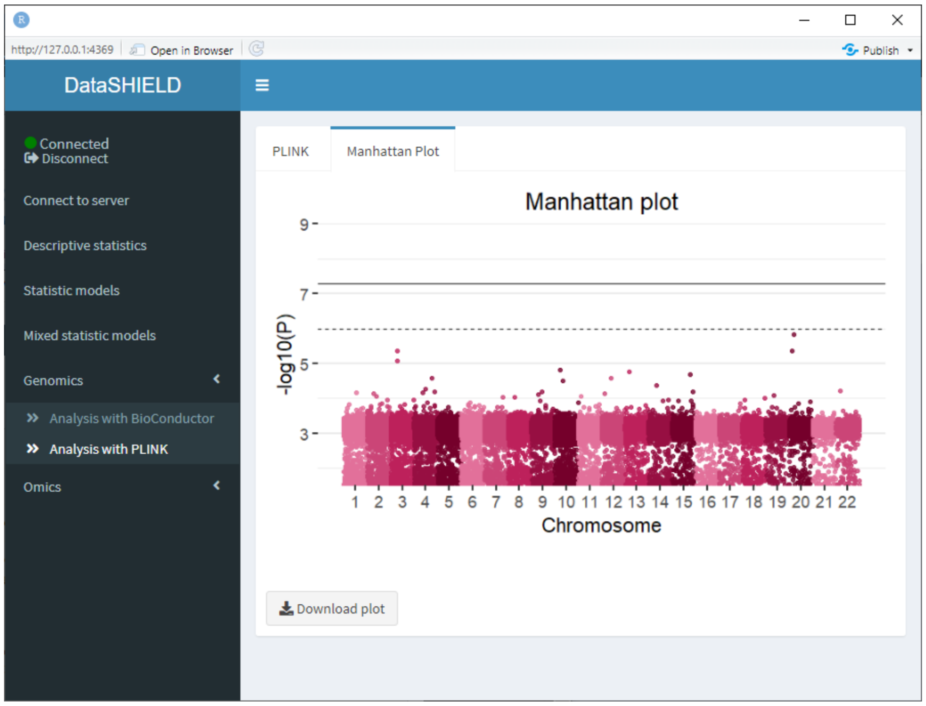


```
@----------------------------------------------------------@
|        PLINK!       |     v1.07      |   10/Aug/2009     |
|----------------------------------------------------------|
|  (C) 2009 Shaun Purcell, GNU General Public License, v2  |
|----------------------------------------------------------|
|  For documentation, citation & bug-report instructions:  |
|        http://pngu.mgh.harvard.edu/purcell/plink/        |
@----------------------------------------------------------@

Skipping web check... [ --noweb ]
Writing this text to log file [ /tmp/ssh-8382/out.log ]
Analysis started: Fri Jul 31 08:52:40 2020

Options in effect:
       --bfile brge
       --logistic
       --covar brge.phe
       --covar-name gender,age
       --noweb
       --out /tmp/ssh-8382/out

Reading map (extended format) from [ brge.bim ]
100000 markers to be included from [ brge.bim ]
Reading pedigree information from [ brge.fam ]
2312 individuals read from [ brge.fam ]
2312 individuals with nonmissing phenotypes
Assuming a disease phenotype (1=unaff, 2=aff, 0=miss)
Missing phenotype value is also -9
725 cases, 1587 controls and 0 missing
1097 males, 1215 females, and 0 of unspecified sex
Reading genotype bitfile from [ brge.bed ]
Detected that binary PED file is v1.00 SNP-major mode
Reading 6 covariates from [ brge.phe ] with nonmissing values for 2199 indivi
Selected subset of 2 from 6 covariates
For these, nonmissing covariate values for 2312 individuals
Before frequency and genotyping pruning, there are 100000 SNPs
2312 founders and 0 non-founders found
6009 heterozygous haploid genotypes; set to missing
Writing list of heterozygous haploid genotypes to [ /tmp/ssh-8382/out.hh ]
7 SNPs with no founder genotypes observed
Warning, MAF set to 0 for these SNPs (see --nonfounders)
Writing list of these SNPs to [ /tmp/ssh-8382/out.nof ]
Total genotyping rate in remaining individuals is 0.994408
0 SNPs failed missingness test ( GENO > 1 )
0 SNPs failed frequency test ( MAF < 0 )
After frequency and genotyping pruning, there are 100000 SNPs
After filtering, 725 cases, 1587 controls and 0 missing
After filtering, 1097 males, 1215 females, and 0 of unspecified sex
```

STAT	P
-1.333	0.1824
-3.271	0.0011
0.5648	0.5722
1.707	0.0879
-3.308	0.0009
0.4648	0.6421
1.919	0.055
-3.457	0.0005
0.5237	0.6005
0.071	0.9434

30259

There's also a sub-tab to show a Manhattan plot with the results obtained. The dowload plot button saves the shown figure as a `*.png`.



3.5 Omics

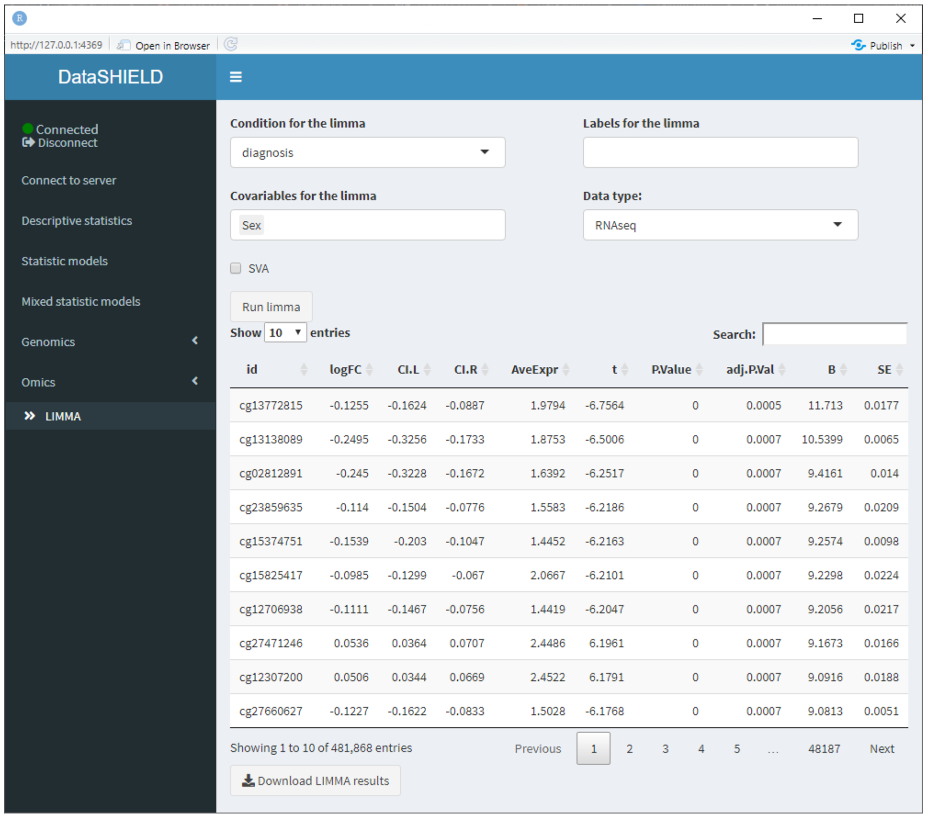
On the Omics tab there are three different subtabs for different methodologies to perform non-disclosive analysis: limma, DESeq and edgeR. The resources that can be used are ExpressionSets and RangegSummarizedExperiments. If the resources are pooled the user has to input each one in a different study on the data entry.

3.5.1 LIMMA

The limma non-disclosive analysis tab contains two selectors to select the condition and covariables of the analysis (resulting formula is: feature ~ condition + covar1 + ... + covarN), there's also a selector to input the annotations columns desired on the output of the analysis. Finally, there's a selector to indicate the type of data that is being studied, whether is microarray or RNAseq. There's a selector to choose to do a surrogate variable analysis.

Once the analysis is performed a table with the results is displayed below the parameter selectors. The download button will prompt a system window to select where to store the shown table, it will save them as a `*.csv`.

If the analysis is being performed usging a pooled dataset, the shown table corresponds to all the pooled data.



Condition for the limma: diagnosis

Labels for the limma:

Covariables for the limma: Sex

Data type: RNAseq

SVA

Show 10 entries

Search:

id	logFC	CI.L	CI.R	AveExpr	t	P.Value	adj.P.Val	B	SE
cg13772815	-0.1255	-0.1624	-0.0887	1.9794	-6.7564	0	0.0005	11.713	0.0177
cg13138089	-0.2495	-0.3256	-0.1733	1.8753	-6.5006	0	0.0007	10.5399	0.0065
cg02812891	-0.245	-0.3228	-0.1672	1.6392	-6.2517	0	0.0007	9.4161	0.014
cg23859635	-0.114	-0.1504	-0.0776	1.5583	-6.2186	0	0.0007	9.2679	0.0209
cg15374751	-0.1539	-0.203	-0.1047	1.4452	-6.2163	0	0.0007	9.2574	0.0098
cg15825417	-0.0985	-0.1299	-0.067	2.0667	-6.2101	0	0.0007	9.2298	0.0224
cg12706938	-0.1111	-0.1467	-0.0756	1.4419	-6.2047	0	0.0007	9.2056	0.0217
cg27471246	0.0536	0.0364	0.0707	2.4486	6.1961	0	0.0007	9.1673	0.0166
cg12307200	0.0506	0.0344	0.0669	2.4522	6.1791	0	0.0007	9.0916	0.0188
cg27660627	-0.1227	-0.1622	-0.0833	1.5028	-6.1768	0	0.0007	9.0813	0.0051

Showing 1 to 10 of 481,868 entries

Previous 1 2 3 4 5 … 48187 Next

3.5.2 DESeq

3.5.3 edgeR

Chapter 4

Developers Guide

Along this section, documentation for future developers and maintainers of ShinyDataSHIELD is provided. It contains information about how the whole Shiny application is structured, all the different scripts that contains, flowcharts of the different files and information on how to extend the capabilities of Shiny-DataSHIELD to new types of resources as well as new methodologies.

Observation
Please read this documentation with the actual source code on the side for easier understanding.

4.1 File structure of ShinyDataSHIELD

Typically Shiny applications are contained in a single file or two files, since the typical structure of a Shiny application is to have a `server` function and a `ui` function that can be on the same file or split for larger applications. On dsOmicShiny the `server` function has been split into different scripts where all of them contains the code of a certain block of the application. It has been done this way to not have a really long `server` file that is difficult to navigate and debug. There is no need to split the `ui` file into different scripts since it only contains the graphical declarations of the applications and is really easy to update and navigate.

The different scripts that compose the whole ShinyDataSHIELD are the following:

- `ui.R`
- `server.R`, composed of the folowing scripts:
  - `connection.R`
  - `descriptive_stats.R`

  - download_handlers.R
  - genomics.R
  - omics.R
  - plot_renders.R
  - statistic_models.R
  - table_renders.R

The file `server.R` exists to source the different files and it also includes some small funcionalities.

Now a file per file explanation will be given with flowcharts (when needed), remarkable bits of code explanations and general remarks. Also, details on how to implement new functionalities will be given when needed.

4.1.1 ui.R

Inside this file there are all the declarations of how the graphical user interface (GUI) will look like.

First, it contains a declaration of all the libraries that have to be loaded for the application to run. The libraries are the following: `DSI`, `DSOpal`, `dsBaseClient`, `dsOmicsClient`, `shinydashboard`, `shiny`, `shinyalert`, `DT`, `data.table`, `shinyjs`, `shinyBS`, `shinycssloaders`).

The next piece of code found

```
jscode <- '
$(document).keyup(function(event) {
    if ($("#password").is(":focus") && (event.keyCode == 13)) {
        $("#connect_server").click();
    }
});
'
```

Is a JavaScript declaration that reads as: When the #password item (corresponds to the text input of the password on the data entry tab) is active (the user is writting in it) and the "Intro" key is pressed, trigger the #connect_server item (corresponds to the "Connect" button on the GUI). That provides the user the typical experience of inputting the login credentials and pressing "Intro" to log in.

It's important noting that this is only the declaration of a string with the code inside, to actually make use of it, there is the line 58 of this same file that actually implements it.

```
tags$head(tags$script(HTML(jscode)))
```

There are a some functions used in this file that are worth mentioning:

- `hidden()`: From the `shinyjs` library. The elements wrapped inside of this function will not be rendered by default, they have to be toggled from the server side. Example: A GUI element that needs to be displayed only when a certain condition is met.

- `withSpinner()`: From the `shinycssloaders` library. The elements wrapped inside of this function will be displayed as a "loading spinner" when they are being processed. This is used to wrap figure displays. Example: A plot that is being rendered, it's better for the user experience to see a "loading spinner" so that it knows something is being processed rather than just staring at a blank screen waiting for something to happen.
- `bsModal()`: From the `shinyBS` library. It's used to prompt pop-ups to the user. Example: By the click of a button you want to render a pop-up to the application with a figure of an histogram of a selected column of a table.

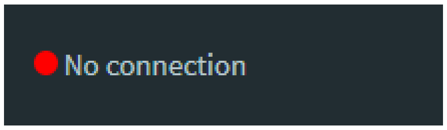
The rest of this file is your average Shiny functions and declarations, read the official documentation for any doubts. Please note that ShinyDataSHIELD uses `shinydashboard` to improve the looks and user experience, for any doubts regarding that please read it's documentation.

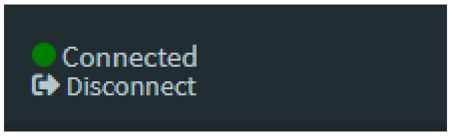
4.1.2 `server.R`

The server file is divided into the following blocks.

- Declaration of reactiveValues: As a code practice measure, all the variables that have to be used in different parts of the code (Example: Table that contains the information about the loaded resources, has to be written when loading the data and afterwards to check whether a resource has been loaded or not) are reactive values. The only occassions where there are "regular" variables are inside functions that use variables as placeholders to be used only inside of that function (Example: Storing the results of a middle ground operation to be later used inside the same function to perform the final analysis, whose results will be saved on a reactive value variable). Developers used to lower level languages can see this as `public` and `private` variables.
- Sourcing of scripts: Sourcing all the different scripts that actually make up `server.R`. As said before this is done this way to have a more structured application where each script takes care of a certain block of the application.
- Function declaration: Declaration of a function that given a column of a data table will truncate the decimal places to 4, it's used when rendering tables to not have tables with 9 decimals that look hideous.
- Functions to manage the "Connected" / "No connection" display. It's a bunch of logic and CSS to just control a small element of the GUI. Basically if the variable `connection$active` is `TRUE` the GUI will show "Connected" next to a green dot with a "Disconnect" button, otherwise it will display "No connection" next to a red dot. When the button "Discon-

nect" is pressed, the function to log out of the server is triggered and the `connection$active` variable is set to false.





The scripts sourced for by the `server.R` are the following:

4.1.2.1 `connection.R`

This is probably the most important script of the whole application, as it's the one that puts some constraints on the capabilities of the application. It also is responsible for loading the data in a proper way in order to ensure that the application capabilities can be extended in the future painlessly (modular).

Inside this script there are five different sections that are triggered by different actions:

- Connection to the server to obtain the projects and resources. Triggered by the button with label "connect_server".
- Get tables / resources from the selected project. Triggered everytime the selector with label "project_selected" is changed.
- Add a study. Triggered by the button with label "add_server".
- Remove a study. Triggered by the button with label "remove_server".
- Load the selected studies to the study servers. Triggered by the button with label "connect_selected".

Since it's very important to understand how the code is working in this script, some flowcharts have been created to be read along the code itself and easily understand what's going on.

Observation
Click on the flowcharts to open them on a separate tab in full-resolution.

There are various red boxes stating some assumptions and some red boxes that show how to implement new features. Let's start by reviewing the assumptions:

- Assumption: Only tables or resources are selected for a study, never both combined
- Assumption: Always only one table per study
- Assumption: Consistency of columns between all the tables of the different

studies (pooled data)

- Assumption: A single study can contain multiple resources, multiple studies can only contain one resource per study

All the assumptions boxes are located on the parts of the flowchart that would have to be modified in order to change them.

When loading the selected resources or tables into the study servers, the table `available_tables` is created. The name is a little bit confusing since it actually contains the information about tables and resources, the developer apologizes as this variable was set at the beginning of the development and has not been updated. Nevertheless, it's an important variable of the application, the structure of this table is the following.

Observation
On the following table there are code examples, it is marked as `THIS` wherever the column value would be placed.

Column	Description
server	Contains the names of the study servers.`connection$builder$append(server = THIS, url = input$url, user = input$user, password = input$password, driver = "OpalDriver")`
table	Name of the table in the Opal server. Structure: `project_name.table_name`
table_internal	Name assigned to the table on the study server. Assigned by:`datashield.assign.table(THIS, table)`
type_resource	Type of the resource
resource	Name of the resource in the Opal server. Structure: `project_name.resource_name`
resource_internal	Name assigned to the resource on the study server. Assigned by:`datashield.assign.resource(THIS, resource)`

The Opal server can host different types of resources, to name a few there are `ExpressionSet`, `RangedSummarizedExperiment` and `SQLResourceClient`. Each type of resource needs a special treatment to be used, for example `SQLResourceClient` resources are plain tables, so they need to be converted to tables on the study server to use them. Currently the following resource types are supported by ShinyDataSHIELD.

Resource type	Treatment	Name of the resource type on `available_tables`
TidyFileResourceClient, SQLResourceClient	- `as.resource.data.frame(resource)`- Change name to `table1` instead of `resource1`- Get column names (`table_columns`) and types (`table_columns_types`) and save them	`table`
SshResourceClient	No treatment	`ssh`
GdsGenotypeReader	- `as.resource.object(resource)`	`r_obj_vcf`
ExpressionSet	- `as.resource.object(resource)`- Copy as `table1` to use the descriptive statistics tab with this resource	`r_obj_eset`
RangedSummarizedExperiment	- `as.resource.object(resource)`- Copy as `table1` to use the descriptive statistics tab with this resource	`r_obj_rse`
Any other resource type	- `as.resource.object(resource)`	`r_obj`

Examples of `available_tables` depending on the selected resources and study servers.

- A pooled study of three tables, each one on a different study server:

server	table	table_internal	type_resource
server1	CNSIM.CNSIM1	table1	table
server2	CNSIM.CNSIM2	table1	table
server3	CNSIM.CNSIM3	table1	table

What can be seen from this table is that there are three study servers and each one contains a different table. Since they are located on different servers they have the same internal names.

- A pooled study using resources that contain ExpressionSets

server	resource	resource_internal	type_resource
server1	RSRC.GSE66351_1	resource1	r_obj_eset
server2	RSRC.GSE66351_2	resource1	r_obj_eset

We can observe the same things as the previous case but here we have a different `type_resource`

- Single study server with two resources (one of which is actually a table)

server	resource	resource_internal	type_resource	table	table_internal
server1	RS.G_covars	resource1	table	RS.G_covars	table1
server1	RS.G_vcf	resource2	r_obj_vcf	NA	NA

On this case we can see that there are two resources on the same server, and the resource of type table is actually copied into a new variable (internal) with the name `table1`.

Now, let's look at some examples to add new resource types on the `connection.R` file. There are different cases for the treatment that the new resource requires.

- Resources that contain tables. They have to be treated as so in order to be able to use them on the "Descriprive statistics", if a new type of resource that contains tables is being implemented, change line 88 `if (any(c("TidyFileResourceClient", "SQLResourceClient") %in% resource_type)){`, in this line we would add the name of the new resource type that is going to be treated as a table. Example: A new resource called `NewTables` is being introduced, the line 88 has to be changed to `if (any(c("TidyFileResourceClient", "SQLResourceClient", "NewTables") %in% resource_type)){`.
- Resources that just need to be loaded with no further action performed to them. Add another `else if` statement after line 107. Example: New resource called `Simple_resource`

```
else if ("Simple_resource" %in% resource_type){
            # Update available_tables list with the new resource type name
            lists$available_tables <- lists$available_tables[resource_internal == paste0("resourc
          }
```

- Resources that need to be converted into R objects (`datashield.assign.expr(conns, symbol = "methy", expr = quote(as.resource.object(res))))` and nothing else. Will work out of the box (the `type_resource` column of the `lists$available_tables` table will read `r_obj`).
- Resources that need to be converted into R objects (`datashield.assign.expr(conns, symbol = "methy", expr = quote(as.resource.object(res))))` and be further processed. Add another `else if` statement after line 135. Example: A new type of resource called `special_resource` that contains some variable names that are desired to be saved on a variable to feed a list on the GUI.

```
else if("special_resource" %in% resource_type) {
              # Update available_tables list with the new resource type name
```

```
      lists$available_tables <- lists$available_tables[resource_internal == pas
      # Perform the needed actions for this resource, on this example: using th
      # and save the output on a variable
      lists$table_columns <- ds.varLabels(paste0("resource", i), datasources =
    }
```

There's already a variable called `table_columns` for those types of cases. If the new resource requires novel things just update the variables on the top of the `server.R` and apply the required methodologies.

As a summary, `connection.R` loads the selected tables / resources into the study servers and updates the table `available_tables` with the information about the loaded resources / tables and their types. The idea is that when X resource is loaded, it's already processed on the `connection.R` to be used on a certain block, so all the funcional blocks only contain the code of their core functionality, not code to pre-process the resource before analyzing it.

4.1.2.2 `descriptive_stats.R`

The script contains all the render functions to render the selectors of the block. This has to be done this way since the "Variable" selector for example is fed a list created when inputing the data, so it's a dynamic value. Putting the render functions on the `ui.R` will result in the application crashing since the list of variables is non existant at the time of launching the Shiny application.

Besides that, there's also an `observe` function that takes care of warning the user if no connection is active and this tab is selected, as well as warning the user if the resource of the study server is not appropiate for this tab.

This block basically contains figures and tables, and they are actually created on the `table_renders.R` and `plot_renders.R`.

4.1.2.3 `statistics_models.R`

This script contains the functions to run two tabs, the "Statistic models" and the "Mixed statistic models". In this tab there are two sets of very similar triggers (`observeEvent()`):

- A toggle to show and hide the table that displays the variables of the study table and their types. (`toggleElement()`)
- Perform the model with the inputed data. Encapsulated on a `tryCatch` so that if there's an error when calculating the model, there's a `shinyalert()` to prompt a popup warning in the GUI.
- Trigger that displays a `shinyalert()` popup with some help regarding the input of the formula.

Those tree elements are doubled, one for the GLM model and the other for the GLMerSLMA. Note that for the GLMerSLMA there's an extra element to select the study to show the results of, since the GLM provides a model for the pooled data and GLMer outputs the coefficients for each of the study servers.

Besides that, there's also an `observe` function that takes care of warning the user if no connection is active and this tab is selected, as well as warning the user if the resource of the study server is not appropiate for this tab.

4.1.2.4 `genomics.R`

On this scripts there are all the genomics methodologies implemented into Shiny-DataSHIELD

- PLINK files: The functions corresponding to the PLINK analysis is just a trigger that reads the inputed command and performs the querry and a `renderText()` function to render the raw text output from the PLINK.
- VCF files: For this resources, there are the following items
  - Render functions for the lists of variables.
  - Trigger to perform the GWAS, encapsulated on a `tryCatch()` so that if there's an error when calculating the GWAS, there's a `shinyalert()` to prompt a popup warning in the GUI.

It's important noting that when using VCF files the user has to load two files into the study, the VCF and a covariates files (table). The `connection.R` script handles that and loads the VCF as a resource and the covariates as a table, so on the `available_tables` they have their own different `type_resource` value. Therefore, this can be done:

```
x=lists$available_tables[type_resource == "r_obj_vcf", resource_internal]
covars = lists$available_tables[type_resource == "table", resource_internal]
```

This assumes that when the user wants to perform this analysis, there's only a single study with two resources, one of type `table` for the covars and one of type `r_obj_vcf`. It's important noting (assuming that the part of the `connection.R` has been understood) that a VCF resource is a type of resource that does not need further treatment than being converted to an R object (`datashield.assign.expr(conns, symbol = "methy", expr = quote(as.resource.object(res)))`) however it was of interest giving it a different `type_resource` name than the standard `r_obj` that is given to this types of resources, so it has it's own `else if` statement on the `connection.R`.

Besides that, there's also an `observe` function that takes care of warning the user if no connection is active and this tab is selected, as well as warning the user if the resource of the study server is not appropiate for either tab (a SSH resource will trigger a warning if the "Analysis with BioConductor" tab is selected).

4.1.2.5 `omics.R`

This script is virtually the same as the VCF part of the `genomics.R`, it contains GUI renders, a trigger to perform the analysis itself encapsulated on a `tryCatch()` and the `observe` function for good measure. It's important noting that so far it only has implemented LIMMA, on the future it should include DESeq and edgeR methodologies.

4.1.2.6 `table_renders.R`

This script creates the displays of all the tables of ShinyDataSHIELD, it uses the `DT` package to do so. Besides the `descriptive_summary` table, all the other tables just render results from other functions.

There are some things to point of this script:

- As can be seen in `descriptive_summary` table, you can actually perform operations inside of a `renderDT` function and display the result of them.
- The most used options for the tables aesthetics are the following

```
options=list(columnDefs = list(list(visible=FALSE, targets=c(0))),
                                          paging = FALSE, searching = FALSE)
```

This prevents the `rownames` column to be displayed (usually it just contains the numeration of rows 1…N, be aware sometimes it's of interest to see this column) and eliminates the paging and searching functionalities of the table. For small tables it makes sense to not show that but on big tables those options are set to `TRUE`, as it's very useful to have a search box on them.

- The tables that display numerical columns (mixed or not with non-numerical columns) are actually passed through the `format_num` function (defined on `server.R`) so the displayed table has only four decimals but the actual table (the one that can be saved) has all the decimals. This is done using the following code

```
as.data.table(lapply(as.data.table(vcf_results$result_table_gwas$server1), format_num)
```

This will pass each column to the function and if it's numerical the decimals will be cut to 4.

- The table output structure of the LIMMA results look different than the others, this is because when performing a LIMMA with pooled resources it returns one table for each study, what is being done is just binding them to display to the user all the obtained results.

4.1.2.7 `plot_renders.R`

There are two types of plots on ShinyDataSHIELD, the ones created with the base function plot and the ones created with the ggplot library. In order to later recover the plots to download them, they actually have a different structure.

- Base plot structure:

```
output$random_plot <- renderPlot({
  plots$random_plot <- function(){
    function_that_generates_the_plot_using_base_package(arguments)
  }
  plots$random_plot()
})
```

For the base plots, a function is declared that returns the plot and is called to generate the plot to the GUI.

- Ggplot structure:

```
output$manhattan <- renderPlot({
  plots$ggplot <- function_that_generates_the_plot_using_ggplot2_package(arguments)
  plots$ggplot
})
```

In this case the plot is saved, ggplot will generate a plot variable that can be called to render the plot.

On this script there are two plots that are inside a `renderCachedPlot` function instead of a `renderPlot` because they take really long to calculate and it's better to cache them.

Inside of the `renderPlot` function some other code can be put, such as toggles to GUI elements or `tryCatch()` functions.

4.1.2.8 `download_handlers.R`

In this script everything related to downloading plots and tables is found. There are basically three types of structures

- Table downloader: To download a `*csv`. Structure:

```
output$table_download <- downloadHandler(
  filename = "table.csv",
  content = function(file) {
    write.csv(
      variable_that_contains_table
      , file, row.names = FALSE)
  }
)
```

The `row.names = FALSE` argument may not be needed in tables where the row names are important.

- Base plot downloader: To download a `*.png`. Structure:

```
output$base_plot_download <- downloadHandler(
  filename = "base_plot.png",
  content = function(file) {
    png(file = file)
    plots$base_plot()
    dev.off()
  }
)
```

Basically this calls the previously declared function and captures the plot into a `*.png`.

- GGplot downloader: To download a `*.png`. Structure:

```
output$ggplot_download <- downloadHandler(
  filename = "ggplot.png",
  content = function(file) {
    ggsave(file, plot = last_plot())
  }
)
```

When using ggplot, the function `last_plot()` renders the last plot rendered by ggplot. This only has one inconvenient, that is when you are downloading a plot that takes a while to render, the application doesn't show the save window dialog until it has rendered again. This should be addressed in the future as it really halters the user experience.

4.1.3 How to add a new block

To add a new block to ShinyDataSHIELD, the developer has to create a new `*.R` script inside the `R` folder of the project and give it a descriprive name of the function that it will perform.

So the Shiny application actually sees it, the `server.R` needs to be updated after line 17 and source the new file. Example: New block called `new_analysis.R`, the update to the `server.R` will be

```
source("new_analysis.R", local = TRUE)
```

Afterwards, the `ui.R` can be updated by defining how the new block will be presented to the user. The `sidebarMenu` function needs to be updated so that the new tab appears on the sidebar of the application, follow the structure of the other tabs. Afterwards update the `dashboardBody` function by defining all the different elements of the new tab, follow the structure of the other available tabs to follow the general design lines, all the functions that need to be used here are standard Shiny functions mostly and there's plenty of documentation and examples available online, when in doubt just try to copy an already implemented structure.

Now the user can focus on the types of files that will feed this new block, if it's a table there's no need to worry, if it's a resource that is not implemented the `connection.R` needs to be updated. Read the above documentation for guidance on the changes that need to be done for new resources types.

Once the GUI is setup and the table / resource that this block will use is setup, the backend for this block can be built on the `new_analysis.R` file. Include on that file all the required `renderUI()` functions and steps to process the file and analyze it. Probably a new variable will be required to hold the results, update the `server.R` header and include a new `reactiveValues()` declaration for the new block.

If the new block requires to display tables or figures, update the `table_renders.R` and `plot_renders.R` following the given examples on their sections of the documentation. Make sure to include the download buttons for them on the `download_handlers.R`.

If there is some part of the code that takes some time to process, there's the option of wrapping it inside the `withProgress()` function in order to display a loading annimation to the GUI to alert the user that something is being processed.

To conclude the new block script, the developer has to make sure to include the `observe()` function to alert the user if there's no connection active or if the resources loaded are not the ones that this block requires. The structure for this is the following

```
observe({
  if(input$tabs == "new_block") {
    if (!connection$active) {shinyalert("Oops!", "Not connected", type = "error")}
    else if (!(any("required_resource_type" %in% unlist(lapply(unique(lists$available_tables$reso
      shinyalert("Oops!", "Selected resource(s) is not the one required by this block.", type = '
    }
    else {
      # Whatever needs to be done
    }
  }
})
```

Note that the `if(input$tabs == "new_block")` refers to the tabname assigned on the `sidebarMenu()` function of the `ui.R`.

When developing a new block there will probably be many problems occuring, in order to debug a Shiny application there is the `browser()` function, if the developer is getting some sort of error at X line of the script, just write `browser()` on the line avobe of the error, the execution will be stopped at that point and the developer can interact with all the available variables of the environment through the RStudio console, usually running the line that is giving an error on the console will provide enough information to kill the bug. If the line breaking is a function call it is advisable to type the variables that are being passed into the function on the console, that way the developer can see what exactly is being passed and can see that some argument is `NULL` when it shouldn't or it's a character when it should be a number, those are quite common problems.

When a new block is developed and integrated into ShinyDataSHIELD, please conclude it by updating this documentation and the user guide with a brief explanation of the new block and some remarks of the most interesting bits of it.