Introduction to dbgapr

Luning Hao, NIH/NCBI/dbGaP

2017-06-08

Overview

The Database of Genotypes and Phenotypes (dbGaP) is a National Institutes of Health (NIH) sponsored repository charged to archive, curate and distribute information produced by studies investigating the interaction of genotype and phenotype. The individual level genotype and phenotype data are distributed in different consent groups through the dbGaP Authorized Access System.

Due to the large number and complicated structures of the dbGaP data files, it often requires significant amount of effort to process, view, and understand the data files before anything else. **The dbgapr is a collection of functions that make it easier to organize, view, and use the phenotype data downloaded from the dbGaP**. The following are a few highlights of the package.

- Organize downloaded phenotype data files in a central location by studies.
- Make data meta-info readily available for search and view.
- Query data meta-info and retrieve data by accessions.
- Basic plotting for data visualization.

This document is a step-by-step introduction of how to setup and use the package. **The examples are based on the data of study phs000001.v3.p1 (AREDS). The input accessions should be changed accordingly based on your specific need and the studies to which your data belong.**

Installation

It is recommended to run dbgapr through RStudio. The table-viewer available in recent versions of RStudio displays dataframe in a human friendly table view, which is perfect for browsing and filtering the dbGaP dataset and variable meta-info resulting from the dbgapr functions. The RStudio desktop version can be downloaded from here.

```
# The dbgapr development version can be installed from GitHub:
install.packages("devtools")
devtools::install_github("jameslhao/dbgapr_dev")
```

Import the package

After the dbgapr package is installed, import it to the R console.

```
library('dbgapr')
```

List of public functions

All public functions belong to either class `Commons()` or class `Study()`. Run the following to view the list of these functions.

```
?dbgapr
```

Project Setup

Create user project

Before anything else, the very first step is to create an user project directory using `prjConfig()`. The user project directory can be any directory in the file system.

```
c <- Commons()
prjConfig(c, prjDir="/home/username/myprj")
```

By the way, various forms of the directory or file paths are acceptable by the functions of this package. Assuming '/home/username' or 'C:/Users/foo' is a **system home directory**, all forms of the directory (or file) path shown below can be used.

```
prjConfig(c, prjDir="/home/username/myprj")        # all platforms
prjConfig(c, prjDir="~/myprj")                  # all platforms
prjConfig(c, prjDir="C:/Users/username/myprj")     # Windows
prjConfig(c, prjDir="C:\\Users\\username\\myprj")  # Windows
```

You may confirm the project directory is created by the following.

```
dir("~/myprj")
```

```
output:
-------
  [1] "gapwork"
```

The user project directory is where all the data and metadata files are located. Multiple project directories can be created, but only one is defined as the current. An old project directory can be reinstated to become current again.

To get the information of created project directories, use `getPrjDir()`.

```
c <- Commons()
getPrjDir(c, showErr=TRUE)
```

Initiate class Commons again

After the project directory is setup, always remember to **initiate the class `Commons()` again** before anything else.

```
c <- Commons()
```

Data preparation

Now it is time to call `prepareData()` and get all the data and metadata ready for search, view, and query. This is a wrapper function that calls other functions descripted next. **When this function is called, the next four functions all the way to `mergeDatasetConsent()` can be skipped.**. However, you may still want to read the instruction of the skipped functions to understand what has been done and how to check the results of each of these steps.

The step may take quite a while to finish.

```
c <- Commons()
prepareData(c, userDataDir='/home/username/decrypted_dbgap_data_dir')
```

Note:

> The userDataDir is the top level directory that contains all the dbGaP phenotype data files intended to be copied (or moved) to the project directory. Before calling this function.
>
> Please make sure all the dbGaP files under the directory are decrypted. To decrypt the dbGaP files, use the vdb-decrypt utility included in the NCBI sra_toolkit. See the related instructions in the dbGaP FAQ for more details.

If you have many studies, running through all of them may take too long to complete. In this case, it is suggested to prepare the data one study at a time.

```r
c <- Commons()
prepareData(c, userDataDir='/home/username/decrypted_dbgap_data_dir', phsAcc='phs000001.v3.p1')
```

Note:

> **Skip next four steps if above function is called.**. This is because `parepareData()` is simply a wrapper function that calls the next four functions described below. It copies user data files to the project directory through `searchCopyPhenoFiles()`, downloads supplemental metadata files from the dbGaP ftp site through `ftpDownload()`, extracts and saves the file meta-info through `recordPrjFileInfo()`, and finally merges the data files of different consents through `mergeDatasetConsent()`. After `prepareData()` is called for the first time, each of these functions can be called individually later if needed.

Copy data files

Ignore this step if `prepareData()` is called.

Once the project directory is setup, the decrypted data files and meta-info (data dictionary) files need to be copied (or moved) to the directory. This is done by `searchCopyPhenoFiles()`.

```r
c <- Commons()
searchCopyPhenoFiles(c, userDataDir='~/decrypted_dbgap_data_dir')
# or to only copy the files of a specific study
searchCopyPhenoFiles(c, userDataDir='~/decrypted_dbgap_data_dir', phsAcc="phs000001.v3.p1")
```

To move instead of copy the files to the project directory, run the followng.

```r
c <- Commons()
searchCopyPhenoFiles(c, userDataDir='~/decrypted_dbgap_data_dir', copyOrMove = 'move')
```

You may confirm the copying is successful by listing the phenotype dataset or data dictionary files such as below.

```r
dir("~/myprj/gapwork/data/phs000001/phs000001.v3/original")
```

```
output:
-------
  [1] "phs000001.v3.pht000001.v2.p1.c1.genspecphenotype.EDO.txt.gz"
  [2] "phs000001.v3.pht000001.v2.p1.c2.genspecphenotype.GRU.txt.gz"
  [3] "phs000001.v3.pht000370.v2.p1.c1.adverse.EDO.txt.gz"
  [4] "phs000001.v3.pht000370.v2.p1.c2.adverse.GRU.txt.gz"
  ...
```

Note:

> The data of different studies are located under the respective study directories. For cohort studies, the files of parent and child studies are located in separate directories, such as the parent study phs000007 and child studies phs000363, phs000401 . . . , as shown below.

```
dir("~/myprj/gapwork/data")
```

```
output:
-------
  [1] "phs000001" "phs000007" "phs000307" "phs000342"
  [5] "phs000363" "phs000401" "phs000429" "phs000572"
  [9] "phs000651" "phs000724"
```

Note:

> There is no direct way to tell their relationship just by the directory structure. The parent child study relationship however can be queried by calling `getAllStudyInfo()` after the supplemental files are copied and the data dictionary files and parsed in next a few steps.

Get supplemental files through ftp

Ignore this step if `prepareData()` is called.

The supplemental metadata of the data required for the package to work needs to be downloaded from the dbGaP ftp site through `ftpDownload()`.

```
c <- Commons()
ftpDownload(c)
# or to limit the download to a specific study
ftpDownload(c, phsAcc="phs000001.v3.p1")
```

Note:

> The downloaded supplemental file 'all_released_study_info.txt.gz' is located under the ~/ncbi/dbgapr_conf directory as shown below.

```
dir("~/.dbgapr/supplemental_data")
```

```
output:
-------
  [1] "all_released_study_info.txt.gz"
```

> The downloaded study specific supplemental files are located under the study directory as shown below.

```
dir("~/myprj/gapwork/data/phs000001/phs000001.v3/supplemental_data")
```

```
output:
-------
  [1] "phs000001.v3_study_dataset_info.txt.gz"
  [2] "phs000001.v3_study_file_manifest.txt.gz"
  [3] "phs000001.v3_study_id_variable_name.txt.gz"
  [4] "phs000001.v3_study_info.txt.gz"
  [5] "phs000001.v3_study_variable_code_value.txt.gz"
  [6] "phs000001.v3_study_variable_info.txt.gz"
```

Record file info

Ignore this step if `prepareData()` is called.

After all the data (dataset) and metadata (data dictionary) files are in place under the project directory. Run `recordPrjFileInfo()` to extract the data and metadata file info and write it into a json file.

```
c <- Commons()
recordPrjFileInfo(c)
# or to process only specific study
recordPrjFileInfo(c, phsAcc="phs000001.v3.p1")
```

The json file can be viewd as below.

```
file.show("~/.dbgapr/pht_file_info_indiv.json")
```

Merge dataset consents

Ignore this step if `prepareData()` is called.

The dbGaP phenotype data is distributed in consent groups. The data files of the same dataset but different consent should be merged before use. Run the following to merge the files.

```
c <- Commons()
mergeDatasetConsent(c)
# or to only merged the files of a specific study
mergeDatasetConsent(c, phsAcc='phs000001.v3.p1')
```

The original phenotype dataset files that have split consents are located in the directory such as the following.

```
dir("~/myprj/gapwork/data/phs000001/phs000001.v3/original")
```

```
output:
-------
  [1] "phs000001.v3.pht000001.v2.p1.c1.genspecphenotype.EDO.txt.gz"
  [2] "phs000001.v3.pht000001.v2.p1.c2.genspecphenotype.GRU.txt.gz"
  [3] "phs000001.v3.pht000370.v2.p1.c1.adverse.EDO.txt.gz"
  [4] "phs000001.v3.pht000370.v2.p1.c2.adverse.GRU.txt.gz"
  [5] "phs000001.v3.pht000371.v2.p1.c1.amdlensphenotype.EDO.txt.gz"
  [6] "phs000001.v3.pht000371.v2.p1.c2.amdlensphenotype.GRU.txt.gz"
  ...
```

After the merge, the dataset files with combined consents are located in the new directory such as below.

```
dir("~/myprj/gapwork/data/phs000001/phs000001.v3/combined")
```

```
output:
-------
  [2] "phs000001.v3_pht000001.v2_union_c1-c2.txt"
  [3] "phs000001.v3_pht000370.v2_union_c1-c2.txt"
  [4] "phs000001.v3_pht000371.v2_union_c1-c2.txt"
  ...
```

The combined data file info of the study is stored in a json file under the log directory such as below.

```
dir("~/myprj/gapwork/data/phs000001/phs000001.v3/combined/log")
```

```
output:
-------
  [1] "phs000001.v3_study_pht_combo_info.json"
```

Data file locations

After going through above steps, all the meta-data and data files should have been in place. From this point on, you can continue to use the functions available in this package to further view, search and query the data. Alternatively, you can also directly work on the files available under the project directory with any of your favorite tools without depending on this package. The locations of various meta-data and data files are described below.

File info file

The meta-info of all processed phenotype data files is saved in the file 'pht_file_info_indiv.json' file as shown below.

```
dir('~/ncbi/dbgapr_conf')
```

```
output:
-------
    [1] "archived_log"              "pht_file_info_indiv.json"
    [3] "project_config.json"       "project_setup.log"
    [5] "supplemental_data"
```

All study info file

The file 'all_released_study_info.txt.gz' contains the meta-info of all the dbGaP released studies. The file is in tab delimited csv format.

```
dir('~/ncbi/dbgapr_conf/supplemental_data')
```

```
output:
-------
    [1] "all_released_study_info.txt.gz"
```

The meta-info all available studies and their datasets is saved in different file formats under the project data directory as shown below.

```
dir("~/myprj/gapwork/data/all_study_info")
```

```
output:
-------
[1] "all_study_dataset_info.csv"
[2] "all_study_dataset_info.json"
[3] "all_study_dataset_info.rds"
[4] "all_study_dataset_info_brief_ljustify.txt"
```

Study specific files

All data files are organized by studies

```
dir("~/myprj/gapwork/data/")
```

```
output:
-------
 [1] "all_study_info" "phs000001"      "phs000007"
 [4] "phs000307"      "phs000342"      "phs000363"
 [7] "phs000401"      "phs000429"      "phs000572"
 [10] "phs000651"      "phs000724"
```

Under each study version directory, there are three sub-directories that contain the original data files, merged dataset files, as well as meta-info files, respectively. The files are in tab delimited csv format.

```r
dir("~/myprj/gapwork/data/phs000001/phs000001.v3")
```

```
output:
-------
    [1] "combined"          "original"          "supplemental_data"
```

The unprocessed original data files are under the 'original' sub-directory.

```r
dir("~/myprj/gapwork/data/phs000001/phs000001.v3/original")
```

```
output:
-------
 [1] "phs000001.v3.pht000001.v2.p1.c1.genspecphenotype.EDO.txt.gz"
 [2] "phs000001.v3.pht000001.v2.p1.c2.genspecphenotype.GRU.txt.gz"
 [3] "phs000001.v3.pht000370.v2.p1.c1.adverse.EDO.txt.gz"
 [4] "phs000001.v3.pht000370.v2.p1.c2.adverse.GRU.txt.gz"
 ...
```

The merged dataset files of different consents are under the 'combined' sub-directory. The files are in tab delimited csv format.

```r
dir("~/myprj/gapwork/data/phs000001/phs000001.v3/combined")
```

```
output:
-------
 [1] "log"
 [2] "phs000001.v3_pht000001.v2_union_c1-c2.txt"
 [3] "phs000001.v3_pht000370.v2_union_c1-c2.txt"
 [4] "phs000001.v3_pht000371.v2_union_c1-c2.txt"
 [5] "phs000001.v3_pht000372.v2_union_c1-c2.txt"
 ...
```

The study, dataset, variable meta-data and release manifest files are under the 'supplemental_data' sub-directory. The files are in tab delimited csv format.

```r
dir("~/myprj/gapwork/data/phs000001/phs000001.v3/supplemental_data")
```

```
output:
-------
 [1] "phs000001.v3_study_dataset_info.txt.gz"
 [2] "phs000001.v3_study_file_manifest.txt.gz"
 [3] "phs000001.v3_study_id_variable_name.txt.gz"
 [4] "phs000001.v3_study_info.txt.gz"
 [5] "phs000001.v3_study_variable_code_value.txt.gz"
 [6] "phs000001.v3_study_variable_info.txt.gz"
 ...
```

View, Search, and Query

Note:

> The dbGaP object accessions **can be used as input arguments with or without the p (participant) or c (consent) number**. This is true for study, dataset, and variable accessions.

For example, the following various forms of the accessions are all acceptable:
phs000001.v3, phs000001.v3.p1, phs000001.v3.p1.c1,
pht000370.v2, pht000370.v2.p1, pht000370.v2.p1.c1,
phv00054119.v1, phv00054119.v1.p1, phv00054119.v1.p1.c2

Get dbGaP object info

After data preparation is done, all the data and metadata are ready for search, view, and query by dbGap accessions. There are three types of the dbGaP objects related to phenotype data. They are study, dataset, and variable. To view the meta-info of an individual object, call `accInfo()` as below.

```
c <- Commons()
df <- accInfo(c, acc='phs000001.v3.p1')    # study
```

```
output:
-------
  Object Type   :   Study
  Accession     :   phs000001.v3
  Name          :   NEI Age-Related Eye Disease Study (AREDS)
  Disease       :   Cataract
  Child Study   :   phs000429.v1
```

```
c <- Commons()
df <- accInfo(c, acc='pht000370.v2')       # dataset
```

```
output:
-------
  Object Type   :   Dataset
  Accession     :   pht000370.v2 ( phs000001.v3 )
  Name          :   adverse
  Description   :   The data table (n=12 variables) contains information about the occurrence of advers
```

```
c <- Commons()
df <- accInfo(c, acc='phv00000084.v2')    # variable of integer type
```

```
output:
-------
  Object Type   :   Variable
  Accession     :   phv00000084.v2 ( phs000001.v3, pht000001.v2 )
  Name          :   SYST00 ( SITTING SYSTOLIC BLOOD PRESSURE AT BASELINE  (PARTICIPANTS WITH A GENETIC
  Value Type    :   integer
  Units         :   mmHg
  Dataset       :   pht000001.v2 ( genspecphenotype )
  Study         :   phs000001.v3
```

```
c <- Commons()
df <- accInfo(c, acc='phv00000031.v2')    # varible of enum_integer type
```

```
output:
-------
  Object Type   :   Variable
  Accession     :   phv00000031.v2 ( phs000001.v3, pht000001.v2 )
```

```
Name          :   TRTCAT ( AREDS TREATMENT ASSIGNMENT (PARTICIPANTS WITH A GENETIC SPECIMEN) )
Value Type    :   enum_integer
Code Value    :   1:Placebo|2:Antioxidants|3:Zinc|4:Antioxidants+Zinc
Dataset       :   pht000001.v2 ( genspecphenotype )
Study         :   phs000001.v3
```

Get info of all studies

The meta-info of all studies available in the project directory can be queried and viewed by `getAllStudyInfo()`. The study-info is returned as a data frame.

```
c <- Commons()
df <- getAllStudyInfo(c)
# or
# display result as a table
df <- getAllStudyInfo(c, showAs='table')
# or
# display result in json with notepad
df <- getAllStudyInfo(c, showAs='json', editor='notepad')
```

Get info of all datasets

The meta-info of all datasets available in the project directory can be queried and viewed by `getAllDatasetInfo()`. The study-info is returned as a data frame.

```
c <- Commons()
df <- getAllDatasetInfo(c)
# or
# display result as a table
df <-  getAllDatasetInfo(c, showAs='table')
# or
# display result in json format with notepad
df <- getAllDatasetInfo(c, showAs='json', editor='notepad')
# or
# display result in a text format
df <-  getAllDatasetInfo(c, showAs='text')
```

Get study variable info

The meta-info of variables under a specified study can be queried and viewed by `getStudyVariableInfo()`. The variable-info is returned as a data frame.

```
s <- Study(phsAcc='phs000001.v3.p1')
# all study variables
df <- getStudyVariableInfo(s)
# or
# display result as a table
df <- getStudyVariableInfo(s, showAs='table')
# or
# all dataset variables
df <- getStudyVariableInfo(s, phtAcc='pht000370.v2.p1')
```

```r
# or
# all dataset variables of numeric type
df <- getStudyVariableInfo(s, phtAcc='pht000370.v2.p1', dataType = 'num')
# or
# a list of variables
acc_list = c('phv00054119.v1.p1.c2', 'phv00054118.v1.p1')
df <- getStudyVariableInfo(s, phvAccList=acc_list)
```

Search study variables by terms

After viewing the meta-info of multiple variables through above function, specific variables can be selected based on the terms present in either the variable descriptions or the variable names. To conduct the search, use `getPhvAccListByTerms()` as below.

```r
s <- Study(phsAcc='phs000001.v3.p1')

##### Search for matches in variable descriptions #####
# define search terms
t1 = c('Diabetes Treatment', 'Smoking Status')
t2 = c('Year 10')
t3 = c('6 Months')
# term match with 'Diabetes Treatment' OR 'Smoking Status'
df <- getPhvAccListByTerms(s, terms_1=t1)
# or
#  display matching variable in a table
df <- getPhvAccListByTerms(s, terms_1=t1, showTable=T)
# or
# term match with ('Diabetes Treatment' OR 'Smoking Status') AND 'Year 10'
df <- getPhvAccListByTerms(s, terms_1=t1, terms_2=t2, showTable=T)
# or
# term match with ('Diabetes Treatment' OR 'Smoking Status') AND 'Year 10' AND '6 Months'
df <- getPhvAccListByTerms(s, terms_1=t1, terms_2=t2, terms_3=t3, showTable=T)

##### Search for matches in variable names #####
t4 = 'SMK'
t5 = 'AGE'
# term match with 'SMK'
df <- getPhvAccListByTerms(s, terms_1=t4, searchIn='name', showTable=T)
# or
# term match with 'SMK' AND 'AGE'
df <- getPhvAccListByTerms(s, terms_1=t4, terms_2=t5, searchIn='name', showTable=T)
```

Get study variable data

The variable data of the study can be obtained by either a dataset accession or a list of variable accessions. Thi is done by calling `getStudyVariableData()` as below.

```r
s <- Study(phsAcc = 'phs000001.v3.p1')
# get data of variables in a dataset
df <- getStudyVariableData(s, phtAcc='pht000370.v2.p1')
# or
# get data of a list of variables
# acc_list = c('phv00054119.v1.p1.c2', 'phv00053733.v2')
df <- getStudyVariableData(s, phvAccList=acc_list)
```

Note:

> It may take a while if retrieving by dataset accession (phtAcc) and the dataset is large.

To subset the data by subject ids, call the function such as the following.

```r
s <- Study(phsAcc = 'phs000001.v3.p1')
# variable data of a set of specified subjects in a dataset
subj_ids = c(219, 220, 221)
df <- getDatasetDataByPhtAcc(s, phtAcc='pht000371.v2', subjIdsOrFile=subj_ids)
# or
# variable data of a set of specified subjects in a dataset
# The subject id file is a plain text file with one dbGaP_Subject_ID per line.
subj_file = "~/temp/subj_ids.txt"
df <- getDatasetDataByPhtAcc(s, phtAcc='pht000371.v2', subjIdsOrFile=subj_file)
```

Note:

> In the returned data frame of variable data, the first column "dbGaP_Subject_ID" is added by dbGaP. **Each of the dbGaP subject id value is unique within the study**. The second column "Submitted_Subject_ID" is also added by dbGaP but **it is a duplicate of submitter provided subject id column**.

The reason to have two columns with duplicated values is that the name of submitter provided id column often varies from study to study. In the following example, the submitter provided id column name is "ID2". The added "Submitted_Subject_ID" is a duplicate of "ID2" with a normalized name.

```r
s <- Study(phsAcc = 'phs000001.v3.p1')
df <- getStudyVariableData(s, phtAcc='pht000370.v2.p1')
# return column names
colnames(df)
```

```
output:
-------
 [1] "dbGaP_Subject_ID"     "Submitted_Subject_ID"
 [3] "ID2"                  "ADVDESC"
 [5] "ADVSTAT"              "SEVERITY"
 ...
```

```r
s <- Study(phsAcc = 'phs000001.v3.p1')
df <- getStudyVariableData(s, phtAcc='pht000370.v2.p1')
# return column values
head(subset(df, select=c('Submitted_Subject_ID', 'ID2')))
```

```
output:
-------
    Submitted_Subject_ID  ID2
  1                 5400 5400
  2                 5537 5537
  3                 4999 4999
  4                 1015 1015
  5                 3554 3554
  6                 2522 2522
```

In the returned data frame, the names of variable columns can be shown as a concatenated strings of the variable name and the respective accession. The following is an example.

```r
s <- Study(phsAcc='phs000001.v3.p1')
df <- getStudyVariableData(s, phtAcc='pht000370.v2.p1', colNameWithAcc=TRUE)
# return column names
colnames(df)
```

```
output:
-------
 [1] "dbGaP_Subject_ID"         "Submitted_Subject_ID"
 [3] "ID2_phv00053730.v2"       "ADVSTAT_phv00053733.v2"
 [5] "OBSTIME_phv00053740.v2"   "ADVDESC_phv00053732.v2"
 ...
```

Get ID mapping

The mapping table between subject, sample, and pedigree IDs can be obtained by calling `getIdInfo()` as below.

```r
s <- Study(phsAcc = 'phs000001.v3.p1')
# get subject id info
df <- getIdInfo(s, infoType='subject')
# or get sample id info
df <- getIdInfo(s, infoType='sample')
# or get pedigree id info
df <- getIdInfo(s, infoType='pedigree')
```

Show variable data summary

The metadata and statistical summary of a given variable can be viewed by calling `variableSummary()` as below.

```r
s <- Study(phsAcc='phs000001.v3.p1')
df <- variableSummary(s, phvAcc='phv00054119.v1.p1.c2')   # numeric
```

```
output:
-------
  Variable Info:
    MATCHSPEC (phv00054119.v1) [integer]  :
     NUMBER OF MATCHED GENETIC SPECIMENS FOR THE PARTICIPANT (PARTICIPANTS WITH A GENETIC SPECIMEN)

  Summary table:
    variable_accession      name calculated_type units max min median     mean na empty    n
  1     phv00054119.v1 MATCHSPEC         integer         4   0      2 1.578682  0     0 3762
```

```r
s <- Study(phsAcc='phs000001.v3.p1')
df <- variableSummary(s, phvAcc='phv00000035.v2')         # categorical
```

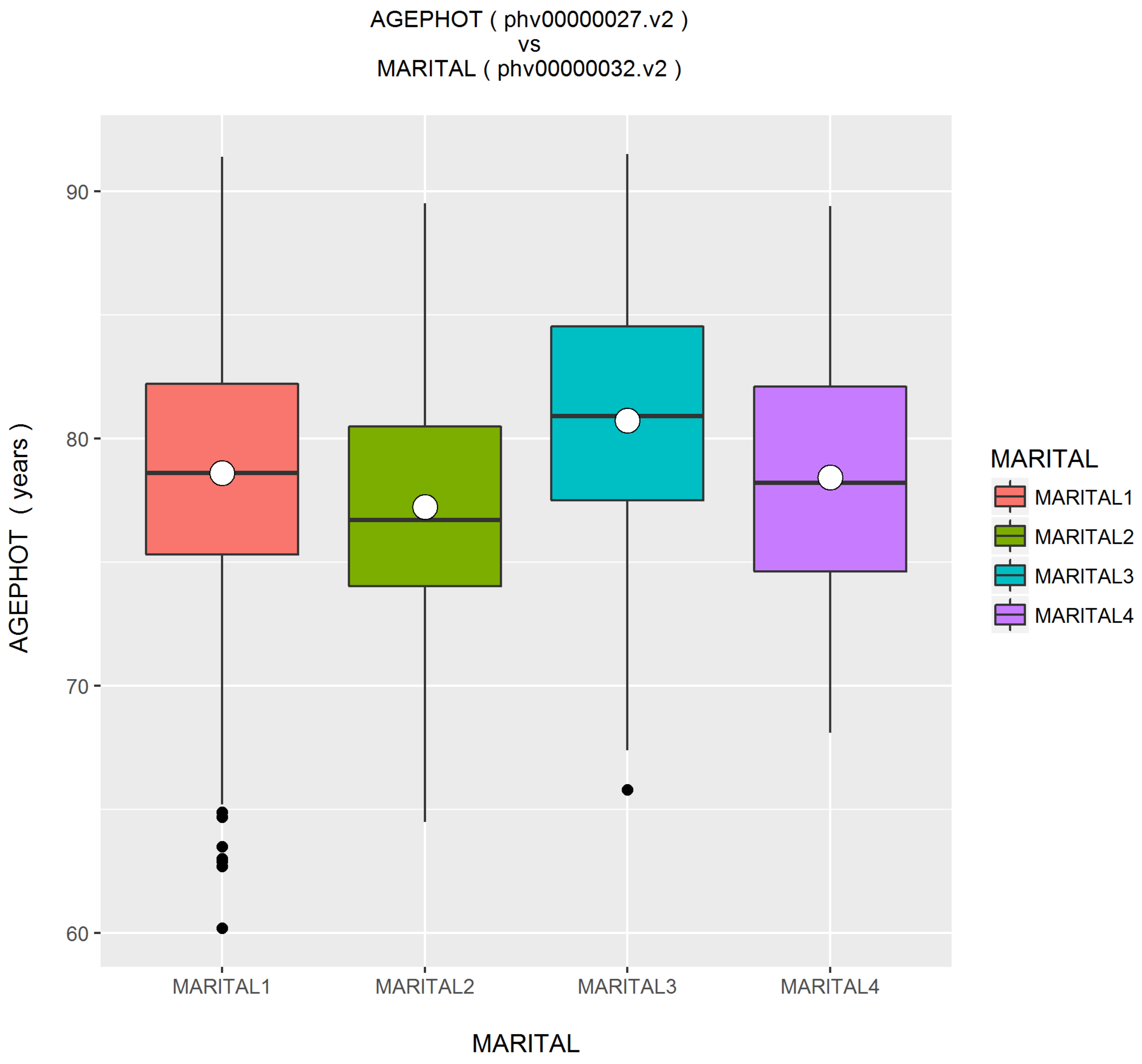
```
output:
-------
  Variable Info:
    SCHOOL (phv00000035.v2) [enum_integer]  :
     HIGHEST LEVEL OF SCHOOL ATTENDED (PARTICIPANTS WITH A GENETIC SPECIMEN)
    code value  :
     1:Grade 11 or less | 2:High school graduate | 3:Some college or associate degree | 4:Bachelor& | 5

  Summary table:
    variable_accession   name calculated_type units na empty    n
  1     phv00000035.v2 SCHOOL    enum_integer        0     0 3762

  Frequency table:
    SCHOOL freq
  1      1  313
  2      2  956
  3      3 1139
  4      4  601
  5      5  751
  6      6    2
```

Basic Plotting

Variable box-and-whisker plot

The box-and-whisker plot of a numeric variable vs each category of a categorical variable can be viewed by calling `variableBoxplot()` as below.

```
s <- Study(phsAcc='phs000001.v3.p1')
df <- variableBoxplot(s, numPhvAcc='phv00000027.v2', catPhvAcc='phv00000032.v2')
```



For all plotting functions, the information of the respective variables and the location of saved images is console displayed as shown below.

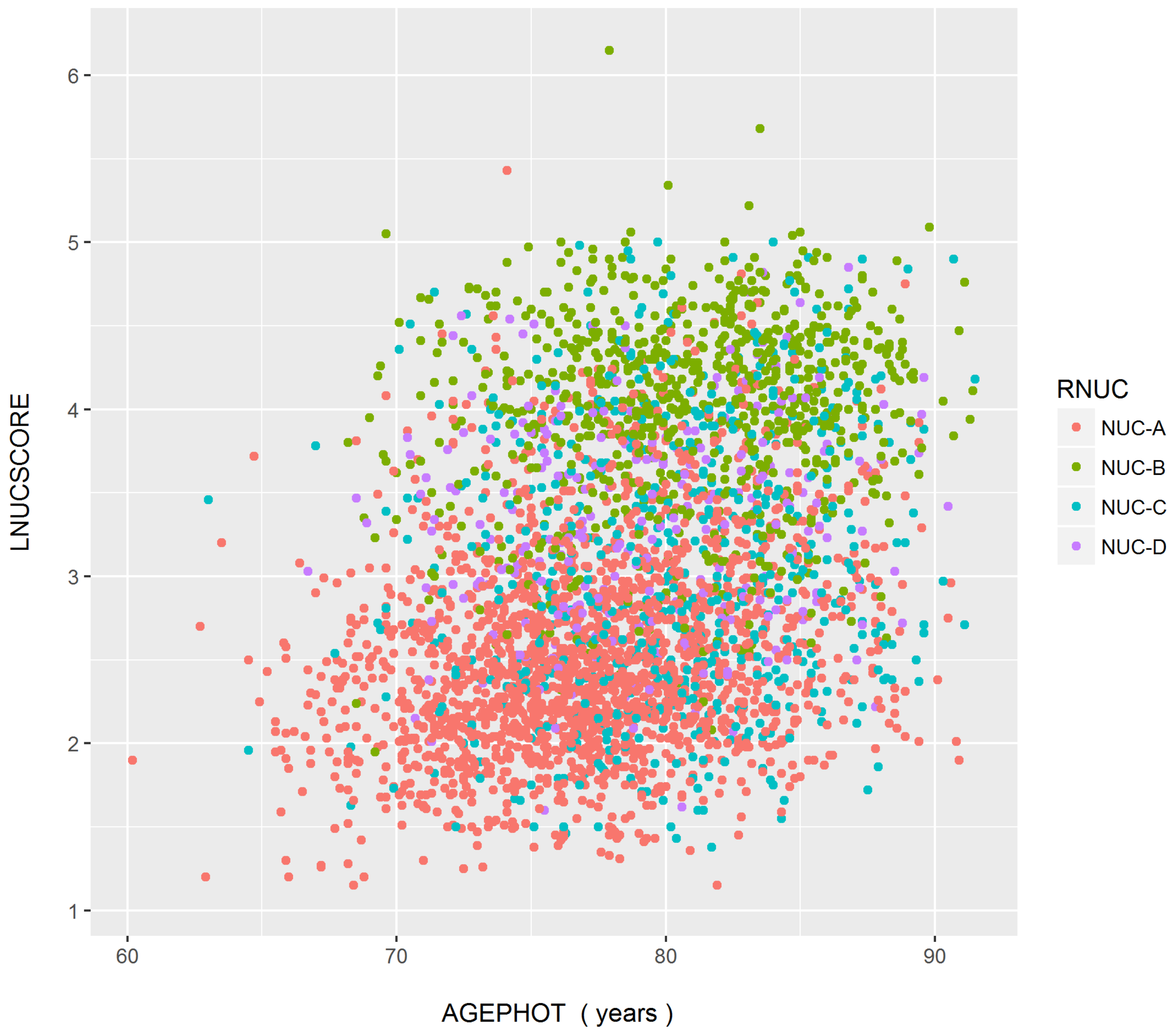
```
output:
-------
  Variable Info:
    MARITAL (phv00000032.v2) [enum_integer]  :
     MARITAL STATUS (PARTICIPANTS WITH A GENETIC SPECIMEN)
    code value  :
     1:Married | 2:Divorced/separated | 3:Widowed | 4:Never married

    AGEPHOT (phv00000027.v2) [decimal]  :
     AGE AT LAST FUNDUS PHOTO (PARTICIPANTS WITH A GENETIC SPECIMEN)

  Saving 6.42 x 7.52 in image
  Saving 6.42 x 7.52 in image

  Saved Files:
   C:/Users/foo/myprj/gapwork/temp/ ...
   C:/Users/foo/myprj/gapwork/temp/ ...
```

Variable scatter plot

The scatter-plot of a numeric variable vs each category of a categorical variable can be viewed by calling `variableScatterplot()` as below.

```r
s <- Study(phsAcc='phs000001.v3.p1')
num_1 = 'phv00000027.v2'
num_2 = 'phv00053747.v2'
cat_1 = 'phv00053757.v2'
df <- variableScatterplot(s, numPhvAcc_1=num_1, numPhvAcc_2=num_2, catPhvAcc=cat_1)
```

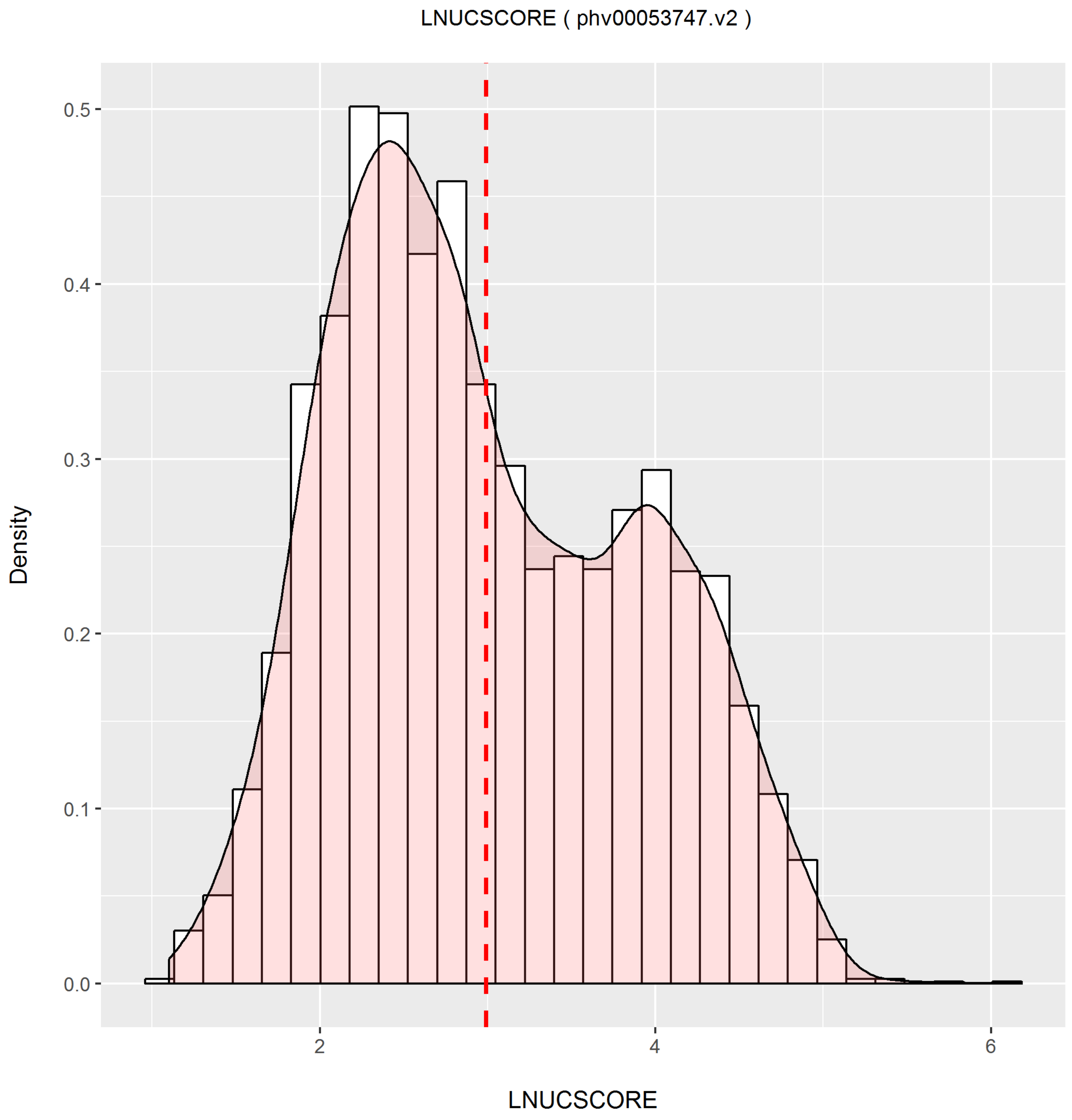
AGEPHOT (phv00000027.v2) vs
LNUCSCORE (phv00053747.v2) vs
RNUC (phv00053757.v2)



Variable histogram

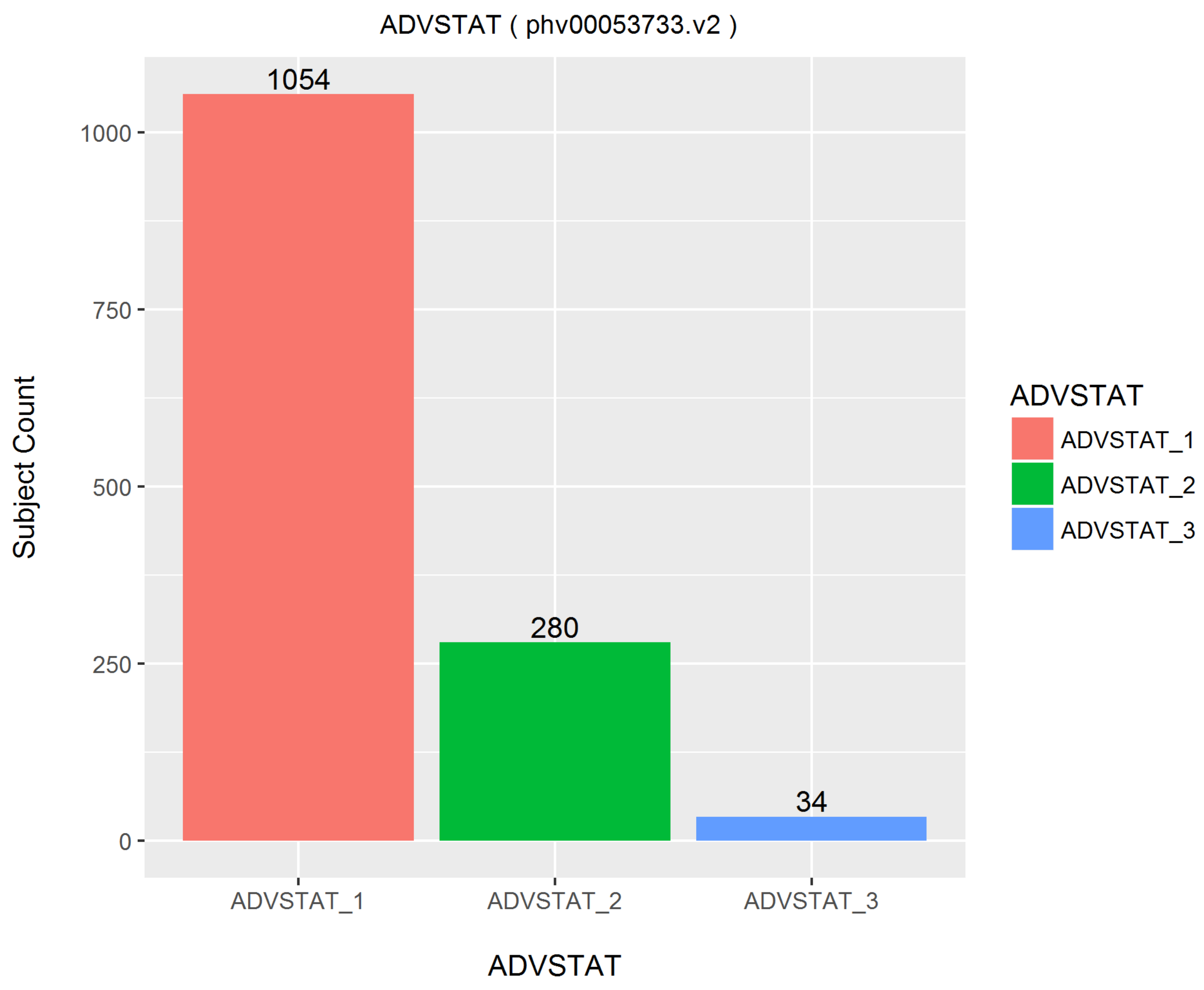
The histogram of a numeric variable can be drawn by calling `variableHistogram()` as below.

```
s <- Study(phsAcc='phs000001.v3.p1')
# with density plot
df <- variableHistogram(s, phvAcc='phv00053747.v2')
# or
# without Density plotâ<...
df <- variableHistogram(s, phvAcc='phv00053747.v2', withDensity=F)
```



When calling the function with a categorical variable, a barchart of each of the categories is drawn.

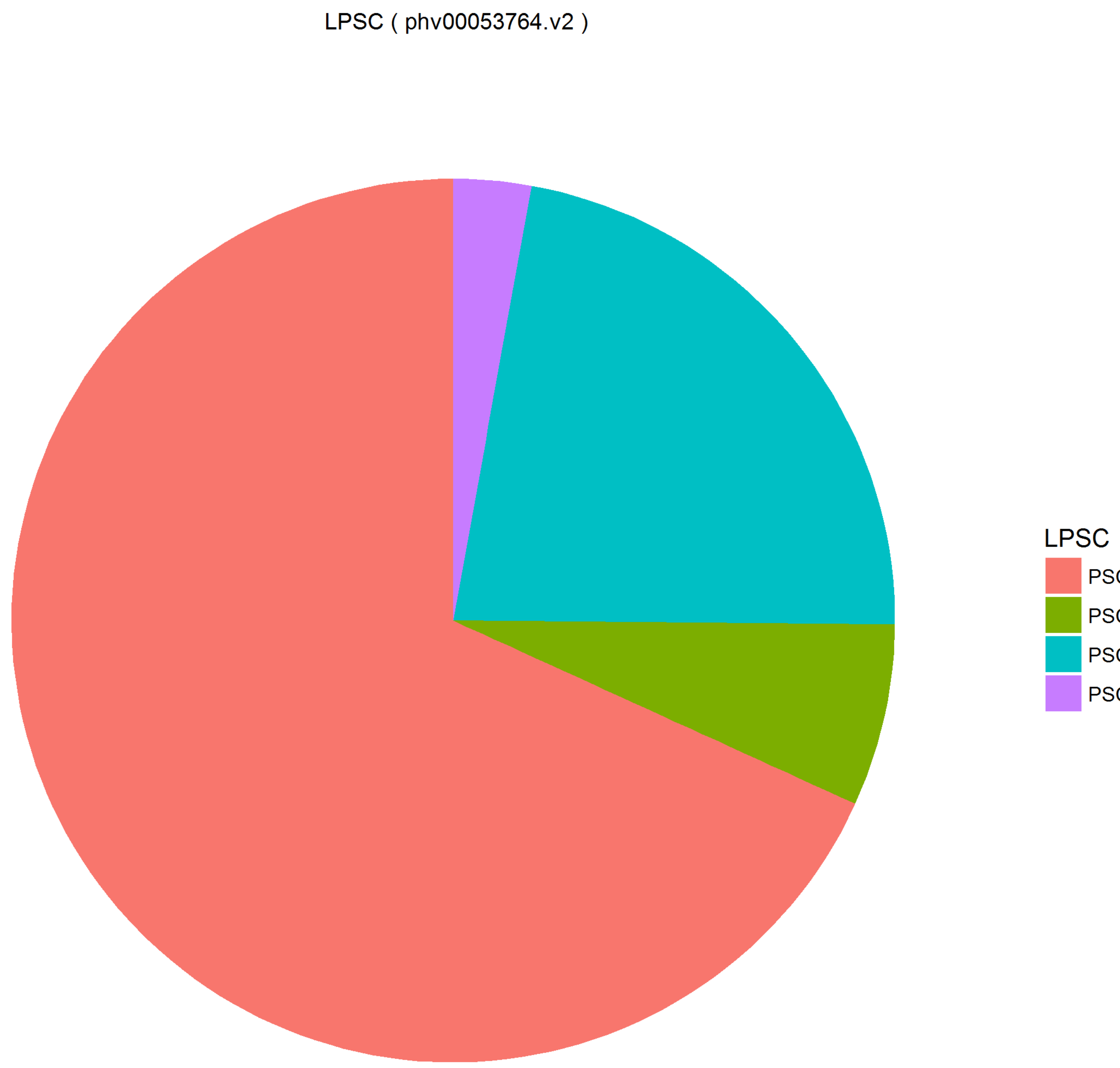
```
s <- Study(phsAcc='phs000001.v3.p1')
# with density plot
df <- variableHistogram(s, phvAcc='phv00053733.v2')
```



Variable piechart

The pie-chart of subject count of the different categories of a categorical variable can be viewed by calling `variablePiechart()` as below.

```
s <- Study(phsAcc='phs000001.v3.p1')
df <- variablePiechart(s, catPhvAcc='phv00053764.v2')
```

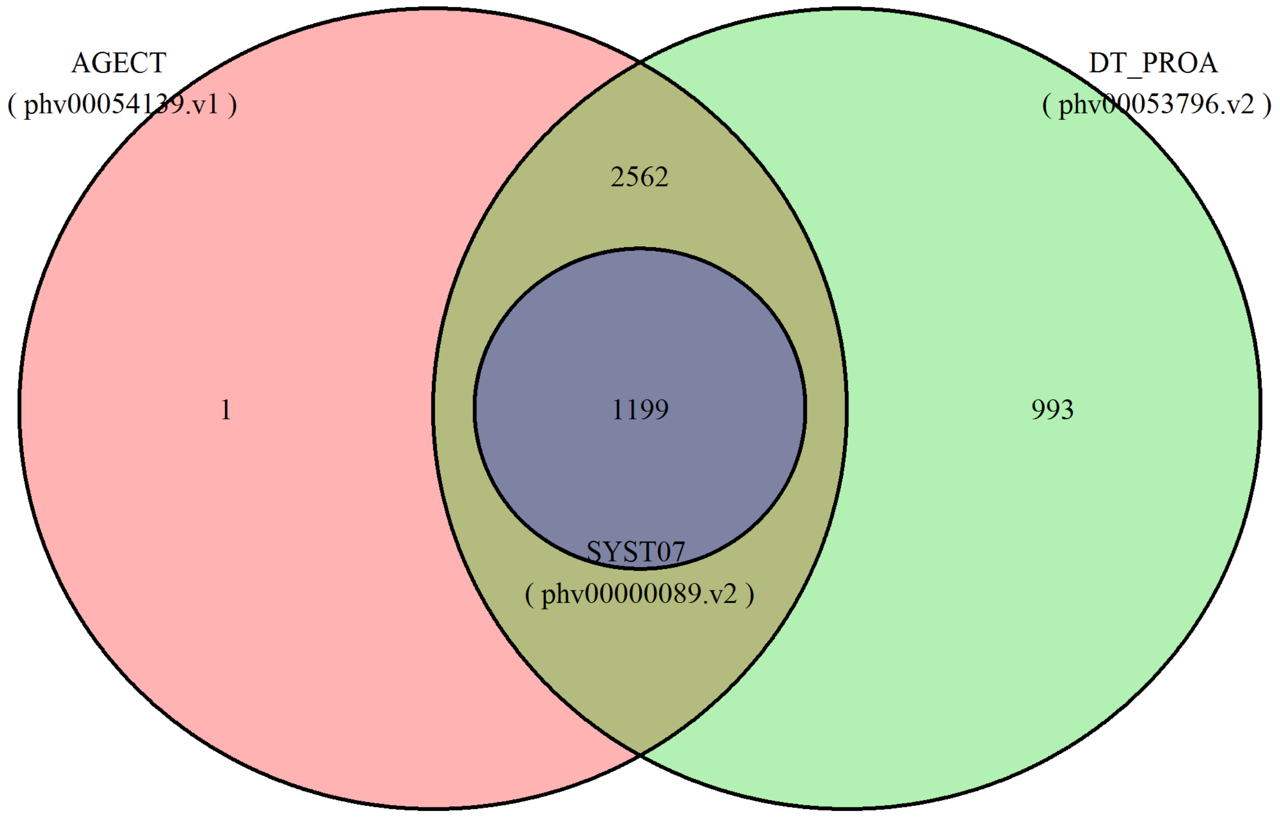


Variable Venndiagram

The Venndiagram that shows the overlapping subjects between a specified list of variables can be viewed by calling `variableVenndiagram()` as below.

```
s <- Study(phsAcc='phs000001.v3.p1')
acc_list = c('phv00054139.v1', 'phv00053796.v2', 'phv00000089.v2')
```

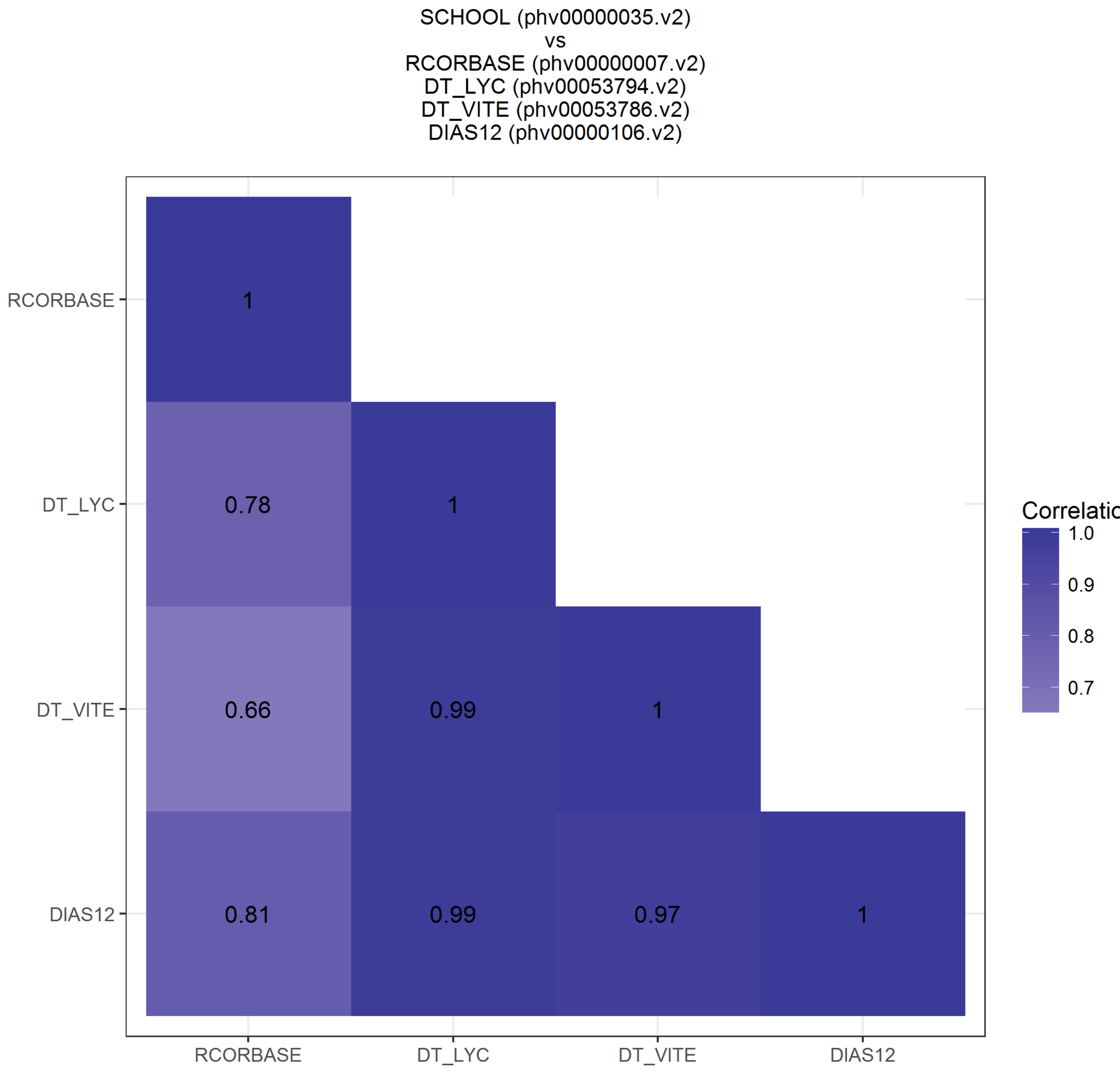
```
df <- variableVenndiagram(s, phvAccList=acc_list)
```



Variable correlation heatmap

The correlation graph between different categories of a given categorical variable can be viewed by calling `variableCorrelHeatmap()` as below.

```
s <- Study(phsAcc='phs000001.v3.p1')
cat_acc = 'phv00000035.v2'
num_acc_list = c('phv00000007.v2', 'phv00053794.v2', 'phv00053786.v2', 'phv00000106.v2')
df <- variableCorrelHeatmap(s, catPhvAcc=cat_acc, numPhvAccList=num_acc_list)
```



Write to dbgap-help@ncbi.nlm.nih.gov for any questions.