Exercise

Introduction to PLINK

Running PLINK

PLINK is run at the command line. Additional arguments ('options', 'flags') specify what exactly PLINK should do. All arguments are documented at the PLINK web site (http://pngu. mgh.harvard.edu/~purcell/plink/). Under Linux, running PLINK requires to open a shell (or terminal) window. Under Windows, PLINK requires a command prompt ('DOS shell'). Use the shell commands `ls`/`dir` and `cd` to change the working directory as requested.

When working with PLINK, it is highly recommendable to save all commands in a text file. This way, your work is documented and you will easily (and with certainty) recapitulate what you have done, say, six or twelve month ago. Therefore, also start the text editor and type all commands in some text file, say `PLINK_exercise.q`, and then copy & paste the command lines from the text editor into the shell.

I. The data set

You are provided with a data set on diastolic blood pressure and the genotypes of 20 SNP markers. The data set is already in PLINK format. There are three files:

- **`dbp.qt.ped`**: Pedigree file with information on family, sex, the quantitative trait (diastolic blood pressure), and genotypes
- **`dbp.cc.ped`**: Pedigree file with information on family, sex, the dichotomized trait (affected yes/no based on blood pressure), and genotypes
- **`dbp.map`**: Map file for the SNP markers (*three* columns format)
- **`dbp.age.pheno`**: Covariate file containing the age of each individual

Use a text editor (`notepad`/`Wordpad` under Windows, `pico`/`vi`/`nano`/`emacs` under Linux) to a have a look at the contents of these files. Make sure you understand the meaning of each column in the files.

`dbp.qt.ped`

```
4928   1      0      0      1      85.51  2      2      1      1      1    ...
1838   1      0      0      1      84.51  1      1      1      1      2    ...
2450   1      0      0      1      84.3   1      1      1      1      2    ...
647    1      0      0      2      89.14  2      2      2      2      1    ...
2772   1      0      0      1      90.39  1      2      1      1      1    ...
...
```

`dbp.cc.ped`

```
4928 1    0    0    1    2    2    2    1    1    1    1    0    0    1   ...
1838 1    0    0    1    2    1    1    1    1    2    2    2    2    2   ...
2450 1    0    0    1    2    1    1    1    1    2    2    2    2    2   ...
647  1    0    0    2    2    2    2    2    2    1    2    1    2    2   ...
2772 1    0    0    1    2    1    2    1    1    1    2    0    0    1   ...
...
```

`dbp.map`

```
11  rs1101   1021
11  rs1102   3886
11  rs1103   15023
11  rs1104   15788
11  rs1105   21702
...
```

dbp.age.pheno

```
4928 1    66
1838 1    67
2450 1    89
647  1    36
2772 1    54
...
```

II. Missing data and filtering

Variables with too many missing values may bias a statistical analysis and lead to spurious results. We will use PLINK to assess the extent of missing values in the data set and to filter variables and samples with too many missing observations.

PLINK requires as the first argument the data set to be processed. This is specified by using the options `--ped` and `--map`. Since the map file contains only three columns instead of the default of four, we additionally have to specify the `--map3` flag. In a first step, we are going to assess the proportion of missing values for each marker and for each sample:

```
plink  --ped dbp.cc.ped  --map dbp.map  --missing
```

Note I: All arguments of a PLINK call have to go on a single line!! Arguments after a line-feed (after pressing the 'Enter' key) will be ignored.

Note II: Using a backslash ('\') at the end of a line suppressed the line-feed and emulates a continuing line. Using backslashes, a single PLINK call can therefore be distributed over numerous lines. The following PLINK call is identical to the one above:

```
plink --ped dbp.cc.ped \
      --map dbp.map\
      --missing
```

PLINK has created three files. The file `plink.log` contains all output from the screen. The files `plink.imiss` and `plink.lmiss` contain the proportion of missing values for each sample and marker, respectively. Use a text editor to have a look at all three files.

plink.log

```
...
PLINK v1.90b6.9 64-bit (4 Mar 2019)
Options in effect:
  --map dbp.map
  --missing
  --ped dbp.cc.ped
...
Start time: ...
...
Scanning .ped file... done.
Performing single-pass .bed write (20 variants, 600 people).
--file: plink-temporary.bed + plink-temporary.bim + plink-temporary.fam
written.
20 variants loaded from .bim file.
600 people (329 males, 271 females) loaded from .fam.
600 phenotype values loaded from .fam.
Using 1 thread (no multithreaded calculations invoked).
Before main variant filters, 600 founders and 0 nonfounders present.
Calculating allele frequencies... done.
Total genotyping rate is 0.988333.
--missing: Sample missing data report written to plink.imiss, and variant-based
missing data report written to plink.lmiss.
End time: ...
```

plink.imiss

```
FID  IID MISS_PHENO   N_MISS   N_GENO   F_MISS
4928   1          N        1       20     0.05
1838   1          N        0       20        0
2450   1          N        1       20     0.05
 647   1          N        0       20        0
...
1284   1          N        2       20      0.1
 172   1          N        1       20     0.05
...
```

plink.lmiss

```
 CHR      SNP   N_MISS   N_GENO   F_MISS
  11   rs1101        0      600        0
  11   rs1102        0      600        0
  11   rs1103        0      600        0
  11   rs1104       92      600   0.1533
  11   rs1105        0      600        0
  11   rs1106       48      600     0.08
  11   rs1107        0      600        0
...
```

Next we are going to exclude samples with more than 10% missing genotypes (`--mind 0.10`) and markers with more than 5% (`--geno 0.05`). We will write this filtered data set to a set of new files, called `cleaned.ped` and `cleaned.map`, using `--recode` and `--out`. Further and quality measures and analyses can then be based on this cleaned data set:

```
plink  --ped dbp.cc.ped  --map dbp.map  --mind 0.10  --geno 0.05 \
       --recode  --out cleaned
```

PLINK has created three different files. A log file called `cleaned.log` (because we used the `--out` flag) and the two data files `cleaned.ped` and `cleaned.map`. Two markers with too many missing values (rs1104 and rs1106) have been excluded. Use the text editor to have a look at these files. Note that the map file has now the default four columns:

cleaned.map

```
11   rs1101   0   1021
11   rs1102   0   3886
11   rs1103   0   15023
11   rs1105   0   21702
11   rs1107   0   23508
11   rs1108   0   28769
11   rs1109   0   31385
11   rs1110   0   33198
11   rs1111   0   1245388
11   rs1112   0   1245604
11   rs1113   0   1246723
11   rs1114   0   1246765
11   rs1115   0   1247100
11   rs1116   0   1257557
11   rs1117   0   1258119
11   rs1118   0   1258732
11   rs1119   0   1259178
11   rs1120   0   1259848
```

We now use this filtered data set to estimate the minor allele frequencies (MAF) of the markers using the `--freq` flag:

```
plink --ped cleaned.ped  --map cleaned.map  --freq  --out cleaned
```

This steps estimates the MAFs, but does *not* filter for a minimum frequency (use the `--maf` flag to this end). The resulting file `cleaned.frq` contains the frequency estimates (check with the text editor). Note that the MAF is always ≤0.50, with the reference allele being *automatically* changed by PLINK! **A2** represents the **major** (more

frequent, 'baseline') allele, while **A1** represents the **minor** (less frequent, 'risk') allele. *This automatic allele flipping is applied throughout many analyses by PLINK, including association testing!!* You have to carefully check which allele is actually the baseline and which is the minor, or risk, allele in your analysis results! For example, the allele '2' is major (more frequent) allele A2 and allele '1' is the minor (less frequent allele A1 of marker rs1101. In order to avoid automatic allele flipping, use the **`--keep-allele-order`** flag throughout!

`cleaned.frq`

```
 CHR      SNP   A1   A2          MAF  NCHROBS
  11   rs1101    1    2       0.4508     1200
  11   rs1102    2    1       0.2642     1200
  11   rs1103    2    1       0.4675     1200
  11   rs1105    1    2       0.4558     1200
  11   rs1107    2    1       0.1525     1200
  11   rs1108    2    1         0.48     1200
...
```

Now let's check for deviations from Hardy-Weinberg equilibrium (HWE):

```
plink  --ped cleaned.ped  --map cleaned.map  --hardy  --out cleaned
```

This steps tests for deviations, but does *not* filter for *P*-values below some threshold (use the `--hwe` flag to this end). The resulting file `cleaned.hwe` looks as follows:

`cleaned.hwe`

```
CHR    SNP  TEST   A1   A2            GENO   O(HET)   E(HET)            P
 11 rs1101   ALL    1    2     115/311/174   0.5183   0.4952       0.2838
 11 rs1101   AFF    1    2       54/159/87     0.53   0.4939       0.2426
 11 rs1101 UNAFF    1    2       61/152/87   0.5067   0.4962       0.8159
 11 rs1102   ALL    2    1      34/249/317    0.415   0.3888        0.115
 11 rs1102   AFF    2    1      15/127/158   0.4233   0.3864       0.1339
 11 rs1102 UNAFF    2    1      19/122/159   0.4067   0.3911       0.5565
...
```

There are three result lines for each marker: one for cases and controls each and one for complete sample. Each line contains the baseline allele ('A2'), the observed genotype counts ('GENO'), the observed and expected frequencies of heterozygotes ('O(HET)' and 'E(HET)'), and the corresponding *P*-values ('P').

Note: Importing data in PLINK format into R

Importing data in PLINK (LINKAGE) format into R can be sometimes troublesome. A helpful format is the creation of tab-separated text files, where columns are separated by a single, 'well-defined' tabular sign ("`\t`"). However, genotypes are distributed over *two* columns in PLINK format, one for each allele. Since these alleles belong to a single genotype, or variable, a different column separation, e.g. by a space, would be desirable. This can be achieved by additionally using the `--tab` flag:

```
plink  --ped cleaned.ped  --map cleaned.map  --out cleaned.R  --recode  \
       --tab
```

The resulting text file can be easily read into R using the `read.table` function, used with the argument `sep="\t"`.

A note on larger projects

PLINK can create a large number of files, overwriting existing files without warning. This can be at times confusing. It is also a potent source of errors when it is not clear, say, which filtering criterions actually applied to some particular data set before an analysis. **"Strategic" planning of filtering and exporting steps as well as well-planned naming of files and distributing of output files in different subdirectories is highly recommended with PLINK.**

III. Binary PLINK format

Genome-wide marker genotype data can be massive, resulting in very large file sizes. To reduce file size and speed up calculations, genotype information is usually compressed. Let's convert the text files into binary PLINK format:

```
plink  --ped dbp.cc.ped  --map dbp.map  --make-bed  --out dbp
```

PLINK has created four files. The file `dbp.log` contains all output from the screen. File `dbp.fam` contains the family information, `dbp.bim` the marker information and `dbp.bed` the marker genotypes in binary (compressed form). Use a text editor to have a look at the fam and the bim files.

How do these files differ from the previous `dbp.ped` and `dbp.map` files?

Introduction to R

Starting R

For starting R under Windows, simply double-click on the R icon: . This will start the R console where all the commands for R can be entered. Under Linux, R is started by simply typing `R` at the terminal shell prompt.

When working with R, it is highly recommendable to save all commands in a text file, usually with a `.q`, `.r` or `.R` suffix. This way, your work is documented and you can easily (and with certainty) recapitulate what you have done, say, six or twelve month ago. Therefore, also start a text editor (`notepad`/`Wordpad` under Windows, `pico`/`vi`/`nano`/`emacs` under Linux) and type all commands in some text file, say `R_exercise.q`, and then copy & paste command lines from the text editor into the R console.

In many cases, you may also want to change the working directory, i.e. the file folder on your computer where R saves files with exported data and from where it expects to read data files into working memory. Under Windows, this can be done via the menu of the R console. Under Linux, the working directory should be changed at the shell prompt *before* starting R.

If you are unsure how to use a function in R or if you want to specify more arguments of the function, use the help function in R. Simply type `?` and the name of the function at the console, e.g. `?summary`.

I: Data Types

Data can be of different types, for example numeric, strings, or logical values. Suppose we want to compile a (very short) list of European cities with a few features for every city. Enter the following commands (*please* remember to first type these commands into the text editor and only then copy & paste them into the R window):

```
city       = c("Oslo", "Bergen", "Munich", "Berlin", "Rome", "Milan")
population = c(0.58, 0.25, 1.3, 3.4, 2.7, 1.3)
country    = factor( c("Norway" , "Norway", "Germany",
                       "Germany", "Italy", "Italy"   ))
capital    = c(TRUE, FALSE, FALSE, TRUE, TRUE, FALSE)
updated    = 2009
```

You have now created various data objects (city names, population sizes, countries of location, capital status of each cities, year of last update) in the working memory of R by using the assignment operator '='. To print the contents of an object, simply type its name:

```
city
population
country
capital
```

Each of these objects is a *vector*, i.e. all elements are of the *same data type*. For example, `city` contains only strings (characters), while `capital` contains only logical values of the cities being the capital of their country or not. Vectors can be concatenated using the `c` function:

```
c(city, city)
c(population, updated)
```

It is often useful to get a short summary of an object. The `summary` function is helpful here:

```
summary (city)
summary (population)
summary (country)
summary (capital)
```

Depending on the data type of an object (or `class` in R), the `summary` function does different things. For example, the mean value can be calculated for numerical variables, but not for nominal ones (represented as `factor` type in R). The type of an object can be assessed by various functions:

```
is.numeric(city)
is.character(city)
is.factor(city)

class (city)
class (population)
class (country)
class (capital)
```

The data type is an attribute of an object. But objects can have more than one attribute. One example is the length, which is the number of elements of an object (i.e. number of entries in the vector):

```
length(city)
```

Note: R can also handle objects with elements of *different type* and *length*. The data type `list` is used to represent such data.

II: Names & Indexes

For better data organization, access and presentation, elements in a vector can have names:

```
names(population) = city
population
```

In many data analyses, one would like to access only parts of the complete data set or even only single elements. For example, markers should be tested for deviations from Hardy-Weinberg equilibrium (HWE) separately in affected and unaffected samples. Access to elements of data objects is achieved by means of *indexes*. There are three different kinds of indexes. The simplest one is addressing by *position*:

```
city [3]
city [2:4]
city[c(1,5:6)]
population[3]
```

If the elements of an object have *names*, these names can also be used to access the elements:

```
population["Oslo"]
population[c("Berlin","Rome")]
```

A third option with vectors is a *logical* index, where only those elements that are marked `TRUE` are accessed. For example, one could select only capitals from the set of all cities:

```
population
capital
population[capital]
```

Logical indexes are quite powerful. One can formulate conditions and store the results in logical vectors. These vectors can then repeatedly be used to access only those elements of the vector that meet the condition. For example, the following commands will select only those cities from our list which have a population of at least one million:

```
population>=1.0
population[population>=1.0]
```

III: Data frames

Data sets are often presented in a tabular form. Columns usually represent features or measurements and are of the *same* data type. Rows represent observations or samples and may contains possibly *different* data types. For example, work sheets from SPSS or Microsoft Excel as well as tables extracted from SQL databases usually adhere to this format. The corresponding R representation is a *data frame*:

```
cities = data.frame (city=city, pop=population,
                     country=country, capital=capital, stringsAsFactors=F)
cities
length(cities)
dim(cities)
is.data.frame(cities)
is.list(cities)
```

Data frames are special lists where all vectors (features) have identical length. They also have some added functionality for printing, summarizing etc. Data frames have two dimensions: rows and columns. Rows (samples) and columns (features) of data frames can also have names:

```
colnames(cities)
rownames(cities)
```

Indexing is similar to that of vectors. Since there are two dimensions (rows & columns), we need two indexes. We can access single elements as well as complete rows or columns. Logical indexes can also be used:

```
cities$city
cities[,1]
cities[2,]
cities[2,3]
cities$pop[3]
cities[capital,]
cities[cities$pop>=1.0,]
```

IV: Export & Import

All objects in R are held in the *working memory*. After quitting R, all objects are lost unless they have been saved in external files on the computer disk!! There are several possibilities to save objects to the disk.

First, let's have an overview on which objects are currently held in the working memory:

```
ls()
```

Now save the objects `cities`, `city`, and `country` in an external archive file called `myobjects.R`. Note that this file is in a format that is only readable with R!

```
save(cities, city, country, file="myobjects.R")
```

It is often useful to export your data set into text format, so that the data can be read with a text editor, such as Word, or be imported into other software programs. For data frames, this can be done with the `write.table` function:

```
write.table(cities, file="cities.txt")
```

Output can also be re-directed from the R console to some text file:

```
sink ("cities.output.txt")
print (cities)
```

```
sink ()
```

Now check if these files have properly been created in the working directory of your computer:

```
dir()
```

This command lists the contents of the current working directory on your computer hard disk. If the files `cities.txt` and `cities.output.txt` have been not been created by R, check for possible errors in your script or ask for help. If these files have been created, delete all objects from the R working memory:

```
rm(list=ls())
ls()
```

No objects are currently held in the working memory. Import the data frame from the external text file `cities.txt` using the `read.table` function and assign it to some object called `new.table`:

```
new.table = read.table ("cities.txt")
ls()
new.table
```

Next, import the objects from the R archive file `myobjects.R` using the `load` function:

```
load ("myobjects.R")
ls()
cities
new.table
```

Quitting R

Quit the R session by entering the following command and answer no to the upcoming question:

```
q()
```

<u>Answers</u>
Introduction to PLINK

I: The data set

dbp.qt.ped

```
4928   1      0      0      1      85.51  2      2      1      1      1    ...
1838   1      0      0      1      84.51  1      1      1      1      2    ...
2450   1      0      0      1      84.3   1      1      1      1      2    ...
647    1      0      0      2      89.14  2      2      2      2      1    ...
2772   1      0      0      1      90.39  1      2      1      1      1    ...
...
```

dbp.cc.ped

```
4928 1    0    0    1    2    2    2    1    1    1    1    0    0    1   ...
1838 1    0    0    1    2    1    1    1    1    2    2    2    2    2   ...
2450 1    0    0    1    2    1    1    1    1    2    2    2    2    2   ...
647  1    0    0    2    2    2    2    2    2    1    2    1    2    2   ...
2772 1    0    0    1    2    1    2    1    1    1    2    0    0    1   ...
...
```

dbp.map

```
11  rs1101   1021
11  rs1102   3886
11  rs1103   15023
11  rs1104   15788
11  rs1105   21702
...
```

dbp.age.pheno

```
4928 1    66
1838 1    67
2450 1    89
647  1    36
2772 1    54
...
```

II. Missing data and filtering

plink --ped dbp.cc.ped --map dbp.map --missing

```
PLINK v1.90b6.9 64-bit (4 Mar 2019)            www.cog-genomics.org/plink/1.9/
(C) 2005-2019 Shaun Purcell, Christopher Chang   GNU General Public License v3
Logging to plink.log.
Options in effect:
  --map dbp.map
  --missing
  --ped dbp.cc.ped

16384 MB RAM detected; reserving 8192 MB for main workspace.
.ped scan complete (for binary autoconversion).
Performing single-pass .bed write (20 variants, 600 people).
--file: plink-temporary.bed + plink-temporary.bim + plink-temporary.fam
written.
20 variants loaded from .bim file.
600 people (329 males, 271 females) loaded from .fam.
600 phenotype values loaded from .fam.
Using 1 thread (no multithreaded calculations invoked).
Before main variant filters, 600 founders and 0 nonfounders present.
Calculating allele frequencies... done.
Total genotyping rate is 0.988333.
--missing: Sample missing data report written to plink.imiss, and variant-based missing
data report written to plink.lmiss.
```

The screen printout documented above is also contained in the file `plink.log`. PLINK has also generated two other files. The files `plink.imiss` and `plink.lmiss` contain the proportion of missing values for each sample and marker, respectively.

plink.imiss

```
FID  IID MISS_PHENO   N_MISS   N_GENO   F_MISS
4928    1          N        1       20     0.05
1838    1          N        0       20        0
2450    1          N        1       20     0.05
 647    1          N        0       20        0
...
1284    1          N        2       20      0.1
 172    1          N        1       20     0.05
...
```

plink.lmiss

```
 CHR      SNP   N_MISS   N_GENO   F_MISS
  11   rs1101        0      600        0
  11   rs1102        0      600        0
  11   rs1103        0      600        0
  11   rs1104       92      600   0.1533
  11   rs1105        0      600        0
  11   rs1106       48      600     0.08
  11   rs1107        0      600        0
  11   rs1108        0      600        0
  11   rs1109        0      600        0
  11   rs1110        0      600        0
```

```
11   rs1111        0      600        0
11   rs1112        0      600        0
11   rs1113        0      600        0
11   rs1114        0      600        0
11   rs1115        0      600        0
11   rs1116        0      600        0
11   rs1117        0      600        0
11   rs1118        0      600        0
11   rs1119        0      600        0
11   rs1120        0      600        0
```

```
plink  --ped dbp.cc.ped  --map dbp.map  --mind 0.10  --geno 0.05 \
       --recode  --out cleaned
```

```
PLINK v1.90b6.9 64-bit (4 Mar 2019)            www.cog-genomics.org/plink/1.9/
(C) 2005-2019 Shaun Purcell, Christopher Chang   GNU General Public License v3
Logging to cleaned.log.
Options in effect:
  --geno 0.05
  --map dbp.map
  --mind 0.10
  --out cleaned
  --ped dbp.cc.ped
  --recode

16384 MB RAM detected; reserving 8192 MB for main workspace.
.ped scan complete (for binary autoconversion).
Performing single-pass .bed write (20 variants, 600 people).
--file: cleaned-temporary.bed + cleaned-temporary.bim + cleaned-temporary.fam
written.
20 variants loaded from .bim file.
600 people (329 males, 271 females) loaded from .fam.
600 phenotype values loaded from .fam.
0 people removed due to missing genotype data (--mind).
Using 1 thread (no multithreaded calculations invoked).
Before main variant filters, 600 founders and 0 nonfounders present.
Calculating allele frequencies... done.
Total genotyping rate is 0.988333.
2 variants removed due to missing genotype data (--geno).
18 variants and 600 people pass filters and QC.
Among remaining phenotypes, 300 are cases and 300 are controls.
--recode ped to cleaned.ped + cleaned.map ... done.
```

PLINK has created three different files. A log file called `cleaned.log` (because we used the `--out` flag) and the two data files `cleaned.map` and `cleaned.ped`. Two markers with too many missing values (rs1104 and rs1106) have been excluded. Use the text editor to have a look at these files. Note that the map file has now the default four columns:

cleaned.map

```
11   rs1101   0   1021
11   rs1102   0   3886
11   rs1103   0   15023
11   rs1105   0   21702
11   rs1107   0   23508
11   rs1108   0   28769
11   rs1109   0   31385
11   rs1110   0   33198
11   rs1111   0   1245388
11   rs1112   0   1245604
11   rs1113   0   1246723
```

```
11   rs1114   0   1246765
11   rs1115   0   1247100
11   rs1116   0   1257557
11   rs1117   0   1258119
11   rs1118   0   1258732
11   rs1119   0   1259178
11   rs1120   0   1259848
```

cleaned.ped

```
4928 1 0 0 1 2 2 2 1 1 1 1 1 1 1 1 2 2 1 1 1 1 2 1 2 1 2 1 2 1 1 2 2 2 1 1 ...
1838 1 0 0 1 2 1 1 1 1 2 2 2 2 1 1 1 1 2 2 2 2 2 2 1 1 1 1 2 2 2 2 2 2 1 1 ...
2450 1 0 0 1 2 1 1 1 1 2 2 2 2 1 1 1 1 2 2 2 2 2 2 2 1 2 1 2 2 1 2 2 2 1 1 ...
 647 1 0 0 2 2 2 2 2 2 2 1 2 2 2 1 2 1 2 2 1 2 1 1 2 1 2 1 1 1 1 2 2 2 2 1 ...
2772 1 0 0 1 2 1 2 1 1 2 1 1 2 1 1 2 1 1 2 1 2 1 1 1 1 1 1 1 1 2 2 2 2 1 1 ...
 148 1 0 0 2 2 2 2 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 2 1 2 1 1 1 1 2 2 2 2 1 ...
   1 1 0 0 1 2 1 2 2 1 2 2 2 2 2 1 1 1 2 2 1 2 1 1 1 1 2 1 2 1 2 2 1 2 1 1 ...
...
```

plink --ped cleaned.ped --map cleaned.map --freq --out cleaned

```
PLINK v1.90b6.9 64-bit (4 Mar 2019)            www.cog-genomics.org/plink/1.9/
(C) 2005-2019 Shaun Purcell, Christopher Chang   GNU General Public License v3
Logging to cleaned.log.
Options in effect:
  --freq
  --map cleaned.map
  --out cleaned
  --ped cleaned.ped

16384 MB RAM detected; reserving 8192 MB for main workspace.
.ped scan complete (for binary autoconversion).
Performing single-pass .bed write (18 variants, 600 people).
--file: cleaned-temporary.bed + cleaned-temporary.bim + cleaned-temporary.fam
written.
18 variants loaded from .bim file.
600 people (329 males, 271 females) loaded from .fam.
600 phenotype values loaded from .fam.
Using 1 thread (no multithreaded calculations invoked).
Before main variant filters, 600 founders and 0 nonfounders present.
Calculating allele frequencies... done.
--freq: Allele frequencies (founders only) written to cleaned.frq
```

PLINK has created the file `cleaned.frq`, containing the frequency estimates. The log file `cleaned.log` has been overwritten:

cleaned.frq

```
 CHR      SNP   A1   A2          MAF  NCHROBS
  11   rs1101    1    2       0.4508     1200
  11   rs1102    2    1       0.2642     1200
  11   rs1103    2    1       0.4675     1200
  11   rs1105    1    2       0.4558     1200
  11   rs1107    2    1       0.1525     1200
  11   rs1108    2    1         0.48     1200
  11   rs1109    1    2       0.4425     1200
  11   rs1110    1    2       0.4558     1200
  11   rs1111    2    1        0.435     1200
  11   rs1112    2    1       0.2958     1200
  11   rs1113    2    1       0.2683     1200
  11   rs1114    2    1       0.4175     1200
  11   rs1115    1    2       0.2642     1200
```

11	rs1116	1	2	0.08	1200
11	rs1117	2	1	0.1817	1200
11	rs1118	2	1	0.2842	1200
11	rs1119	1	2	0.185	1200
11	rs1120	1	2	0.3025	1200

plink --ped cleaned.ped --map cleaned.map --hardy --out cleaned

```
PLINK v1.90b6.9 64-bit (4 Mar 2019)            www.cog-genomics.org/plink/1.9/
(C) 2005-2019 Shaun Purcell, Christopher Chang   GNU General Public License v3
Logging to cleaned.log.
Options in effect:
  --hardy
  --map cleaned.map
  --out cleaned
  --ped cleaned.ped

16384 MB RAM detected; reserving 8192 MB for main workspace.
.ped scan complete (for binary autoconversion).
Performing single-pass .bed write (18 variants, 600 people).
--file: cleaned-temporary.bed + cleaned-temporary.bim + cleaned-temporary.fam
written.
18 variants loaded from .bim file.
600 people (329 males, 271 females) loaded from .fam.
600 phenotype values loaded from .fam.
Using 1 thread (no multithreaded calculations invoked).
Before main variant filters, 600 founders and 0 nonfounders present.
Calculating allele frequencies... done.
--hardy: Writing Hardy-Weinberg report (founders only) to cleaned.hwe ... done.
```

PLINK has created the file `cleaned.hwe`, containing the *P* values for the test of deviation from Hardy-Weinberg equilibrium (HWE). The log file `cleaned.log` has again been overwritten:

cleaned.hwe

CHR	SNP	TEST	A1	A2	GENO	O(HET)	E(HET)	P
11	rs1101	ALL	1	2	115/311/174	0.5183	0.4952	0.2838
11	rs1101	AFF	1	2	54/159/87	0.53	0.494	0.2426
11	rs1101	UNAFF	1	2	61/152/87	0.5067	0.4962	0.8159
11	rs1102	ALL	2	1	34/249/317	0.415	0.3888	0.115
11	rs1102	AFF	2	1	15/127/158	0.4233	0.3864	0.1339
11	rs1102	UNAFF	2	1	19/122/159	0.4067	0.3911	0.5565
11	rs1103	ALL	2	1	126/309/165	0.515	0.4979	0.4136
11	rs1103	AFF	2	1	57/159/84	0.53	0.496	0.2943
11	rs1103	UNAFF	2	1	69/150/81	0.5	0.4992	1
11	rs1105	ALL	1	2	118/311/171	0.5183	0.4961	0.2859
11	rs1105	AFF	1	2	56/165/79	0.55	0.4971	0.08129
11	rs1105	UNAFF	1	2	62/146/92	0.4867	0.495	0.8155
11	rs1107	ALL	2	1	13/157/430	0.2617	0.2585	0.8749
11	rs1107	AFF	2	1	6/85/209	0.2833	0.2711	0.5274
11	rs1107	UNAFF	2	1	7/72/221	0.24	0.2456	0.6406
11	rs1108	ALL	2	1	139/298/163	0.4967	0.4992	0.9348
11	rs1108	AFF	2	1	74/152/74	0.5067	0.5	0.9081
11	rs1108	UNAFF	2	1	65/146/89	0.4867	0.4968	0.7281
11	rs1109	ALL	1	2	113/305/182	0.5083	0.4934	0.508
11	rs1109	AFF	1	2	56/154/90	0.5133	0.4936	0.5587
11	rs1109	UNAFF	1	2	57/151/92	0.5033	0.4932	0.8148
11	rs1110	ALL	1	2	112/323/165	0.5383	0.4961	0.04003
11	rs1110	AFF	1	2	50/169/81	0.5633	0.4947	0.01965
11	rs1110	UNAFF	1	2	62/154/84	0.5133	0.4973	0.6426
11	rs1111	ALL	2	1	116/290/194	0.4833	0.4915	0.6785
11	rs1111	AFF	2	1	62/140/98	0.4667	0.4928	0.3509
11	rs1111	UNAFF	2	1	54/150/96	0.5	0.4902	0.8138

```
  11   rs1112      ALL    2    1    52/251/297   0.4183   0.4166            1
  11   rs1112      AFF    2    1    39/145/116   0.4833   0.4671       0.6212
  11   rs1112    UNAFF    2    1    13/106/181   0.3533   0.3432       0.7367
  11   rs1113      ALL    2    1    43/236/321   0.3933   0.3927            1
  11   rs1113      AFF    2    1    27/136/137   0.4533   0.4328       0.5044
  11   rs1113    UNAFF    2    1    16/100/184   0.3333   0.3432       0.6146
  11   rs1114      ALL    2    1   111/279/210    0.465   0.4864       0.2764
  11   rs1114      AFF    2    1    52/137/111   0.4567   0.4807        0.401
  11   rs1114    UNAFF    2    1     59/142/99   0.4733   0.4911       0.5568
  11   rs1115      ALL    1    2    45/227/328   0.3783   0.3888       0.5286
  11   rs1115      AFF    1    2    35/127/138   0.4233   0.4411       0.5128
  11   rs1115    UNAFF    1    2    10/100/190   0.3333     0.32       0.5887
  11   rs1116      ALL    1    2      4/88/508   0.1467   0.1472        0.785
  11   rs1116      AFF    1    2      3/43/254   0.1433     0.15       0.4294
  11   rs1116    UNAFF    1    2      1/45/254     0.15   0.1444            1
  11   rs1117      ALL    2    1    14/190/396   0.3167   0.2973       0.1309
  11   rs1117      AFF    2    1    12/117/171     0.39   0.3595       0.1974
  11   rs1117    UNAFF    2    1      2/73/225   0.2433   0.2237       0.1935
...
```

PLINK has performed HWE tests for each marker in each of three sample sets: controls (‘`UNAFF`’), cases (‘`AFF`’) and controls and cases combined (‘`ALL`’). The ‘`GENO`’ column gives the counts of A1/A1, A1/A2 and A2/A2 genotypes in the sample set, respectively. The columns ‘`O(HET)`’ and ‘`E(HET)`’ give the observed and the expected frequency of heterozygous genotypes A1/A2 according to the Hardy-Weinberg proportions (i.e. $2pq$ if p denotes the frequency of the A1 allele and q that of the A2 allele). The ‘`P`’ column contains the *P*-value from the test.

```
plink  --ped cleaned.ped  --map cleaned.map  --out cleaned.R  --recode  \
       --tab

PLINK v1.90b6.9 64-bit (4 Mar 2019)            www.cog-genomics.org/plink/1.9/
(C) 2005-2019 Shaun Purcell, Christopher Chang   GNU General Public License v3
Logging to cleaned.R.log.
Options in effect:
  --map cleaned.map
  --out cleaned.R
  --ped cleaned.ped
  --recode
  --tab

Note: --tab flag deprecated.  Use '--recode tab ...'.
16384 MB RAM detected; reserving 8192 MB for main workspace.
.ped scan complete (for binary autoconversion).
Performing single-pass .bed write (18 variants, 600 people).
--file: cleaned.R-temporary.bed + cleaned.R-temporary.bim +
cleaned.R-temporary.fam written.
18 variants loaded from .bim file.
600 people (329 males, 271 females) loaded from .fam.
600 phenotype values loaded from .fam.
Using 1 thread (no multithreaded calculations invoked).
Before main variant filters, 600 founders and 0 nonfounders present.
Calculating allele frequencies... done.
18 variants and 600 people pass filters and QC.
Among remaining phenotypes, 300 are cases and 300 are controls.
--recode ped to cleaned.R.ped + cleaned.R.map ... done.
```

cleaned.R.ped

```
4928  1  0  0  1  2   2 2  1 1  1 1  1 1  ..
1838  1  0  0  1  2   1 1  1 1  2 2  2 2  ..
2450  1  0  0  1  2   1 1  1 1  2 2  2 2  ..
647   1  0  0  2  2   2 2  2 2  2 1  2 2  ..
2772  1  0  0  1  2   1 2  1 1  2 1  1 2  ..
148   1  0  0  2  2   2 2  1 1  1 1  1 1  ..
1     1  0  0  1  2   1 2  2 1  2 2  2 2  ..
1696  1  0  0  2  2   1 2  2 1  2 1  1 2  ..
890   1  0  0  1  2   1 2  1 1  2 1  1 2  ..
1832  1  0  0  1  2   1 2  1 1  2 1  1 2  ..
...
```

cleaned.R.map

```
11 rs1101 0      1021
11 rs1102 0      3886
11 rs1103 0      15023
11 rs1105 0      21702
11 rs1107 0      23508
11 rs1108 0      28769
11 rs1109 0      31385
11 rs1110 0      33198
...
```

II. Missing data and filtering

plink --ped dbp.cc.ped --map dbp.map --missing

```
PLINK v1.90b4.4 64-bit (21 May 2017)           www.cog-genomics.org/plink/1.9/
(C) 2005-2017 Shaun Purcell, Christopher Chang   GNU General Public License v3
Logging to dbp.log.
Options in effect:
  --make-bed
  --map dbp.map
  --out dbp
  --ped dbp.cc.ped

16384 MB RAM detected; reserving 8192 MB for main workspace.
.ped scan complete (for binary autoconversion).
Performing single-pass .bed write (20 variants, 600 people).
--file: dbp-temporary.bed + dbp-temporary.bim + dbp-temporary.fam written.
20 variants loaded from .bim file.
600 people (329 males, 271 females) loaded from .fam.
600 phenotype values loaded from .fam.
Using 1 thread (no multithreaded calculations invoked).
Before main variant filters, 600 founders and 0 nonfounders present.
Calculating allele frequencies... done.
Total genotyping rate is 0.988333.
20 variants and 600 people pass filters and QC.
Among remaining phenotypes, 300 are cases and 300 are controls.
--make-bed to dbp.bed + dbp.bim + dbp.fam ... done.
```

dbp.fam

```
4928 1 0 0 1 2
1838 1 0 0 1 2
2450 1 0 0 1 2
647  1 0 0 2 2
2772 1 0 0 1 2
148  1 0 0 2 2
1    1 0 0 1 2
1696 1 0 0 2 2
890  1 0 0 1 2
1832 1 0 0 1 2
...
```

dbp.bim

```
11 rs1101 0      1021   1      2
11 rs1102 0      3886   2      1
11 rs1103 0      15023  2      1
11 rs1104 0      15788  1      2
11 rs1105 0      21702  1      2
11 rs1106 0      23505  2      1
11 rs1107 0      23508  2      1
11 rs1108 0      28769  2      1
11 rs1109 0      31385  1      2
11 rs1110 0      33198  1      2
...
```

File `dbp.fam` contains the first six columns of `dbp.ped`, whereas `dbp.bim` contains all four columns from `dbp.map` and two additional columns from the `dbp.ped` file listing the A2 and A1 alleles.

Introduction to R

I: Data Types

```
city       = c("Oslo", "Bergen", "Munich", "Berlin", "Rome", "Milan")
population = c(0.58, 0.25, 1.3, 3.4, 2.7, 1.3)
country    = factor ( c("Norway", "Norway", "Germany",
                        "Germany", "Italy", "Italy") )
capital    = c(TRUE, FALSE, FALSE, TRUE, TRUE, FALSE)
updated    = 2009
city
[1] "Oslo"   "Bergen" "Munich" "Berlin" "Rome"   "Milan"
population
[1] 0.58 0.25 1.30 3.40 2.70 1.30
country
[1] Norway  Norway  Germany Germany Italy   Italy
Levels: Germany Italy Norway
capital
[1]  TRUE FALSE FALSE  TRUE  TRUE FALSE

c(city, city)
[1] "Oslo"   "Bergen" "Munich" "Berlin" "Rome"   "Milan"  "Oslo"   "Bergen" "Munich"
"Berlin" "Rome"   "Milan"
c(population, updated)
[1]    0.58    0.25    1.30    3.40    2.70    1.30 2009.00
```

```
summary (city)
Length     Class      Mode
        6 character character
summary (population)
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.250   0.760   1.300   1.588   2.350   3.400
summary (country)
Germany   Italy  Norway
      2       2       2
summary (capital)
   Mode   FALSE    TRUE    NA's
logical       3       3       0

is.numeric(city)
[1] FALSE
is.character(city)
[1] TRUE
is.factor(city)
[1] FALSE
class (city)
[1] "character"
class (population)
[1] "numeric"
class (country)
[1] "factor"
class (capital)
[1] "logical"
length(city)
[1] 6
```

II: Names & Indexes

```
names(population) = city
  Oslo Bergen Munich Berlin   Rome  Milan
  0.58   0.25   1.30   3.40   2.70   1.30
population
city [3]
[1] "Munich"
city [2:4]
[1] "Bergen" "Munich" "Berlin"
city[c(1,5:6)]
[1] "Oslo"  "Rome"  "Milan"
population[3]
Munich
   1.3
population["Oslo"]
Oslo
0.58
population[c("Berlin","Rome")]
Berlin   Rome
   3.4    2.7
population[capital]
  Oslo Berlin   Rome
  0.58   3.40   2.70
population>=1.0
  Oslo Bergen Munich Berlin   Rome  Milan
 FALSE  FALSE   TRUE   TRUE   TRUE   TRUE
population[population>=1.0]
Munich Berlin   Rome  Milan
   1.3    3.4    2.7    1.3
```

III: Data frames

```
cities = data.frame (city=city, pop=population,
                     country=country, capital=capital,
                     stringsAsFactors = F)

cities
         city  pop country capital
Oslo     Oslo 0.58  Norway    TRUE
Bergen Bergen 0.25  Norway   FALSE
Munich Munich 1.30 Germany   FALSE
Berlin Berlin 3.40 Germany    TRUE
Rome     Rome 2.70   Italy    TRUE
Milan   Milan 1.30   Italy   FALSE
length(cities)
[1] 4
dim(cities)
[1] 6 4

is.data.frame(cities)
[1] TRUE
is.list(cities)
[1] TRUE

colnames(cities)
[1] "city"    "pop"     "country" "capital"
rownames(cities)
[1] "Oslo"   "Bergen" "Munich" "Berlin" "Rome"   "Milan"

cities$city
[1] "Oslo"   "Bergen" "Munich" "Berlin" "Rome"   "Milan"
cities[,1]
[1] "Oslo"   "Bergen" "Munich" "Berlin" "Rome"   "Milan"
cities[2,]
         city  pop country capital
Bergen Bergen 0.25  Norway   FALSE
cities[2,3]
[1] Norway
Levels: Germany Italy Norway
cities$pop[3]
[1] 1.3

cities[capital,]
         city  pop country capital
Oslo     Oslo 0.58  Norway    TRUE
Berlin Berlin 3.40 Germany    TRUE
Rome     Rome 2.70   Italy    TRUE

cities[cities$pop>=1.0,]
         city pop country capital
Munich Munich 1.3 Germany   FALSE
Berlin Berlin 3.4 Germany    TRUE
Rome     Rome 2.7   Italy    TRUE
Milan   Milan 1.3   Italy   FALSE
```

IV: Export & Import

```
ls()
[1] "capital"       "cities"        "city"          "country"
[5] "population"    "updated"

save(cities, city, country, file="myobjects.R")
```

```
write.table(cities, file="cities.txt")

sink (“cities.output.txt”)
print (cities)
sink ()

dir()
[1] "R_exercise.txt"  "cities.output.txt"  "myobjects.R"

rm(list=ls())
ls()
character(0)

new.table = read.table ("cities.txt")
ls()
[1] "new.table"
new.table
         city  pop country capital
Oslo     Oslo 0.58  Norway    TRUE
Bergen Bergen 0.25  Norway   FALSE
Munich Munich 1.30 Germany   FALSE
Berlin Berlin 3.40 Germany    TRUE
Rome     Rome 2.70   Italy    TRUE
Milan   Milan 1.30   Italy   FALSE

load ("myobjects.R")
ls()
[1] "cities"    "city"      "country"   "new.table"

cities
         city  pop country capital
Oslo     Oslo 0.58  Norway    TRUE
Bergen Bergen 0.25  Norway   FALSE
Munich Munich 1.30 Germany   FALSE
Berlin Berlin 3.40 Germany    TRUE
Rome     Rome 2.70   Italy    TRUE
Milan   Milan 1.30   Italy   FALSE

new.table
         city  pop country capital
Oslo     Oslo 0.58  Norway    TRUE
Bergen Bergen 0.25  Norway   FALSE
Munich Munich 1.30 Germany   FALSE
Berlin Berlin 3.40 Germany    TRUE
Rome     Rome 2.70   Italy    TRUE
Milan   Milan 1.30   Italy   FALSE
```