Polygenic Risk Score Analyses Workshop 2022

Day 1: GWAS & relevant Statistics

Day 1 Timetable

Time	Title	Presenter
9:00 - 9:15	Welcome Address	Dr Daneshwar and Dr Baichoo
9:15 - 9:30	Opening Speech from Organisers	Dr Segun Fatumo and Dr Nicki Tiffin
9:30 - 10:30	Lecture: Background to PRS: GWAS & relevant Statistics	Dr Paul O'Reilly
10:30 - 11:00	Coffee Break and Q&A	-
11:00 - 12:00	Practical: Introduction to Bash and R	Dr Paul O'Reilly & Tutors
12:00 - 13:30	Lunch	-
13:30 - 15:00	Practical: Introduction to PLINK I - Basics	Dr Conrad Iyegbe & Tutors
15:00 - 15:30	Coffee Break and Q&A	-
15:30 - 16:30	Practical: Introduction to PLINK II - QC & GWAS	Dr Conrad Iyegbe & Tutors

Contents

Day 1 Timetable **1**
Day 1 Timetable 1

1 Introduction to Bash **3**
1.1 Moving around the File System 3
1.2 Looking at the Current Directory 4
1.3 Counting Number of Lines in File 4
1.4 Search File Content 5
1.5 Filtering and Reshuffling Files 6

2 Introduction to R **7**
2.1 Basics 7
2.1.1 Libraries 7
2.1.2 Variables in R 8
2.1.3 Functions 8
2.2 Plotting 8
2.3 Regression Models 9

1 Introduction to Bash

Most software in Bioinformatics and Statistical Genetics need to be run in a Unix environment (e.g. Linux or Mac OS) and most high-performance computer clusters run Unix systems. Therefore, although there are alternatives available on Windows (command line, Linux subsystems or Virtual Machines), it will be highly beneficial to become familiar with performing research in a Unix-only environment.

In this practical, we will only go through some basic operations.

Please refer to online tutorials after this workshop to learn more about the details (e.g. here or here) or more advanced commands.

1.1 Moving around the File System

To begin our practical, please open up a "terminal" on your computer (on a Mac this is stored in Applications/Utilities/).

We can change our directory using the following command:

```
cd <Path>
```

where *<Path>* is the path to the target directory.

cd stands for change directory

Some common usage of cd includes

```
cd ~ # will bring you to your home directory
cd ../  # will bring you to the parent directory (up one level)
cd XXX # will bring you to the XXX directory, so long as it is in the current directory
```

As an example, we can move to the **PRS_Workshop** directory by typing:

```
cd ~/Desktop/PRS_Workshop/
```

1.2 Looking at the Current Directory

Once we have moved into the **PRS_Workshop** folder, we can list out the folder content by typing:

```
ls
```

ls stands for list

For ls, there are a number of additional Unix command options that you can append to it to get additional information, for example:

```
ls -l # shows files as list
ls -lh # shows files as a list with human readable format
ls -lt # shows the files as a list sorted by time-last-edited
ls -lS # shows the files as a list sorted by size
```

ls provides information about *files* in a directory and does not provide information about the contents of *sub-directories*. Therefore, a sub-directory containing a large amount of data may not appear at the top of an ls -lS command because the size of files within that directory are not considered.

Using the terminal, can you tell what the **PRS Workshop** folder contains?

Use cd to navigate into the Day_1a/ directory and then to navigate into the Data/ folder inside Day_1a/, and then use one of the ls commands above to find out what the largest file is inside Data/. What is the name of the largest file and how big is it?

1.3 Counting Number of Lines in File

We can also count the number of lines in a file with the following command (where <*file*> is the file of interest):

```
wc -l <file>
```

How many lines are there in the **Data/GIANT_Height.txt** file?

Often we would like to store the output of a command, which we can do by *redirecting* the output of the command to a file. For example, we can redirect the count of the **GIANT_Height.txt** to **giant_count** using the following command:

```
1 wc -l GIANT_Height.txt > giant_count.txt
```

1.4 Search File Content

Another common task is to search for specific words or characters in a file (e.g. does this file contain our gene of interest?). This can be performed using the "grep" command as follows:

```
1 grep <string> <file>
```

For example, to check if the Single Nucleotide Polymorphism (SNP) *rs10786427* is present in **GIANT_Height.txt**, we can do:

```
1  grep rs10786427 GIANT_Height.txt
```

In addition, grep allows us to check if patterns contained in one file can be found in another file. For example, if we want to extract a subset of samples from the phenotype file (e.g. extract the list of samples in **Data/Select.sample**), we can do:

```
1 grep -f  Select.sample TAR.height
```

An extremely useful feature of the terminal is chaining multiple commands into one command, which we call ***piping***.

For example, we can use piping to count the number of samples in **Select.sample** that were found in **TAR.height** in a single command, as follows:

```
1 grep -f  Select.sample TAR.height | wc -l
```

How many samples from **Select.sample** were found in **TAR.height**?

1.5 Filtering and Reshuffling Files

A very powerful feature of the terminal is the **awk** programming language, which allows us to extract subsets of a data file, filter data according to some criteria or perform arithmetic operations on the data. awk manipulates a data file by performing operations on its **columns** - this is extremely useful for scientific data sets because typically the columns features or variables of interest.

For example, we can use awk to produce a new results file that only contains SNP rsIDs (column 1), allele frequencies (column 4) and *P*-values (column 7) as follows:

```
awk '{ print $1,$4,$7}' GIANT_Height.txt > GIANT_Height_3cols.txt
```

See if you can work out how to create a new file from GIANT_Height.txt that contains the SNP rsIDs, Allele1, Allele2, the allele frequency **as a percentage**, and the sample size (N).

We can also use a "conditional statement" in awk to extract all *significant SNPs* from the results file, using the following command:

```
awk '{if($7 < 5e-8) { print } }' GIANT_Height.txt > Significant_SNPs.txt
# Or the short form:
awk '$7 < 5e-8{ print}' GIANT_Height.txt > Significant_SNPs.txt
```

"if($7<5e-8)" and "$7 < 5e-8" tell awk to extract any rows with column 7 (the column containing *P*-value) with a value of smaller than 5e-8 and {print} means that we would like to print the entire row when this criterion is met.

How many SNPs from the Height GWAS passed the significance threshold?

2 Introduction to R

R is a useful programming language that allows us to perform a variety of statistical tests and data manipulation. It can also be used to generate fantastic data visualisations. Here we will go through some of the basics of **R** so that you can better understand the practicals throughout the workshop.

R Studio is a very useful graphical user interface (GUI) for **R**. You may prefer to use R Studio rather than running R from the terminal.

2.1 Basics

If you are not using R Studio then you can type **R** in your terminal to run **R** in the terminal.

Working Directory

When we start **R**, we will be working in a specific folder called the **working directory**. We can check the current/working directory we are in by typing:

```
1 getwd()
```

And we can change our working directory to the **Practical** folder by

```
1 setwd("~/Desktop/PRS\_Workshop/Day_1a/")
```

2.1.1 Libraries

Most functionality of **R** is organised in "packages" or "libraries". To access these functions, we will have to install and "load" these packages. Most commonly used packages are installed together with the standard installation process. You can install a new library using the install.packages function.

For example, to install *ggplot2*, run the command:

```
1 install.packages("ggplot2")
```

After installation, you can load the library by typing

```
library(ggplot2)
```

Alternatively, we can import functions (e.g. that we have written) from an R script file on our computer. For example, you can load the Nagelkerke R^2 function by typing

```
source("Software/nagelkerke.R")
```

And you are now able to use the NagelkerkeR2 function (we will use this function at the end of this worksheet).

2.1.2 Variables in R

You can assign a value or values to any variable you want using <-. e.g

```
# Assign a number to a
a <- 1
# Assign a vector containing a,b,c to b
v1 <- c("a", "b","c")
```

2.1.3 Functions

You can perform lots of operations in **R** using different built-in R functions. Some examples are below:

```
# Assign number of samples
nsample <- 10000
# Generate nsample random normal variable with mean = 0 and sd = 1
normal <- rnorm(nsample, mean=0,sd=1)
normal.2 <- rnorm(nsample, mean=0,sd=1)
# We can examine the first few entries of the result using head
head(normal)
# And we can obtain the mean and sd using
mean(normal)
sd(normal)
# We can also calculate the correlation between two variables using cor
cor(normal, normal.2)
```

2.2 Plotting

While **R** contains many powerful plotting functions in its base packages, customisation can be difficult (e.g. changing the colour scales, arranging the axes). **ggplot2**

is a powerful visualization package that provides extensive flexibility and customisation of plots. As an example, we can do the following

```
# Load the package
library(ggplot2)
# Specify sample size
nsample<-1000
# Generate random grouping using sample with replacement
groups <- sample(c("a","b"), nsample, replace=T)
# Now generate the data
dat <- data.frame(x=rnorm(nsample), y=rnorm(nsample), groups)
# Generate a scatter plot with different coloring based on group
ggplot(dat, aes(x=x,y=y,color=groups))+geom_point()
```

2.3 Regression Models

In statistical modelling, regression analyses are a set of statistical techniques for estimating the relationships among variables or features. We can perform regression analysis in **R**.

Use the following code to perform linear regression on simulated variables "x" and "y":

```
# Simulate data
nsample <- 10000
x <- rnorm(nsample)
y <- rnorm(nsample)
# Run linear regression
lm(y~x)
# We can store the result into a variable
reg <- lm(y~x)
# And get a detailed output using summary
summary(lm(y~x))
# We can also extract the coefficient of regression using
reg$coefficient
# And we can obtain the residuals by
residual <- resid(reg)
# Examine the first few entries of residuals
head(residual)
# We can also include covariates into the model
covar <- rnorm(nsample)
lm(y~x+covar)
# And can even perform interaction analysis
lm(y~x+covar+x*covar)
```

Alternatively, we can use the glm function to perform the regression:

```
glm(y~x)
```

For binary traits (case controls studies), logistic regression can be performed using

```
# Simulate samples
```

```
nsample<- 10000
x <- rnorm(nsample)
# Simulate binary traits (must be coded with 0 and 1)
y <- sample(c(0,1), size=nsample, replace=T)
# Perform logistic regression
glm(y~x, family=binomial)
# Obtain the detail output
summary(glm(y~x, family=binomial))
# We will need the NagelkerkeR2 function
# to calculate the pseudo R2 for logistic model
source("Software/nagelkerke.R")
reg <- glm(y~x, family=binomial)
# Calculate the Nagelkerke R2 using the NagelkerkeR2 function
NagelkerkeR2(reg)
```