Lab 5: Prediction from genetic data

Biomedical Data Science

Marco Colombo, University of Edinburgh

GWAS analyses

Given the size of the datasets and the large number of computations, usually genome-wide association studies are run with specialized software that is optimized to read data in binary format and can represent genetic data in memory efficiently (plink, SNPTEST, RegScan, etc). R is very inefficient memory-wise, and performs poorly in loops (that's a reason why using vectorised code is always to be preferred). However we can still perform the same analyses in R, provided that datasets are not too large.

The simplest types of association studies rely on testing one SNP at a time for association with the outcome: for a continuous trait, we use linear regression, for a binary trait we use logistic regression.

In real analyses, it is common to include other relevant covariates in the models, to ensure that the SNP association is independent of those covariates. In particular, models are adjusted by an indicator variable corresponding to the cohort of origin (as they may have been genotyped with different technologies, or may have differences in the way their phenotypes are measured) and by the first few principal components (often 5-10 PCs) to remove possible confounding effects due to population stratification. It's also increasingly common to use linear mixed models to account for population structure and relatedness between individuals.

In what follows, we will perform *crude associations*, that is not adjusted for other covariates, bearing in mind that it's a simplification. We start by reading file `chrom21.csv` (available on Learn), which contains 1500 SNPs from chromosome 21 as well as a column with a quantitative outcome (the first column).

```
> chrom21 <- read.csv("data/chrom21.csv")
```

Also genetic data can contain missing values, due to genotyping errors or poor quality reads that are discarded at the quality control step.

```
> table(is.na(chrom21))

 FALSE   TRUE
301447    254
```

A common approach to impute missing values in a SNP is by replacing them with the mean of the SNP: this corresponds to assign the expected allele dosages to the missing entries.

```
> column.means <- colMeans(chrom21, na.rm=TRUE)
>
> # loop only over the columns with missing values
> for (col in which(is.na(colSums(chrom21))))
+   chrom21[is.na(chrom21[, col]), col] <- column.means[col]
```

Given that we have a continuous outcome, to test for a crude association of a SNP it is sufficient to run a simple linear regression model, then check the summary statistics.

```
> snp1 <- lm(outcome ~ snp1, data=chrom21)
> coef(summary(snp1))
             Estimate Std. Error    t value   Pr(>|t|)
(Intercept) -0.2169840 0.08694013 -2.4957863 0.01338047
snp1        -0.1410125 0.26897298 -0.5242626 0.60067937
```

This specific SNP is not associated with the outcome.

It's clear that we do not want to explicitly formulate a model for each SNP in the dataset, but want to create a small function that could be used within a `for` loop to automate the process.

```
# run univariate tests of associations for all SNPs (columns of x)
univ.lm.test <- function(x, y) {
  stopifnot(length(x) == length(y))
  regr <- lm(y ~ x)

  # remove the row corresponding to the intercept and the column containing
  # the t-value, then convert to a dataframe
  output <- data.frame(coef(summary(regr)))[-1, -3]

  # assign better column names
  colnames(output) <- c("beta", "std.error", "p.value")
  return(output)
}
```

With this function we can run the same model as before by running the following command.

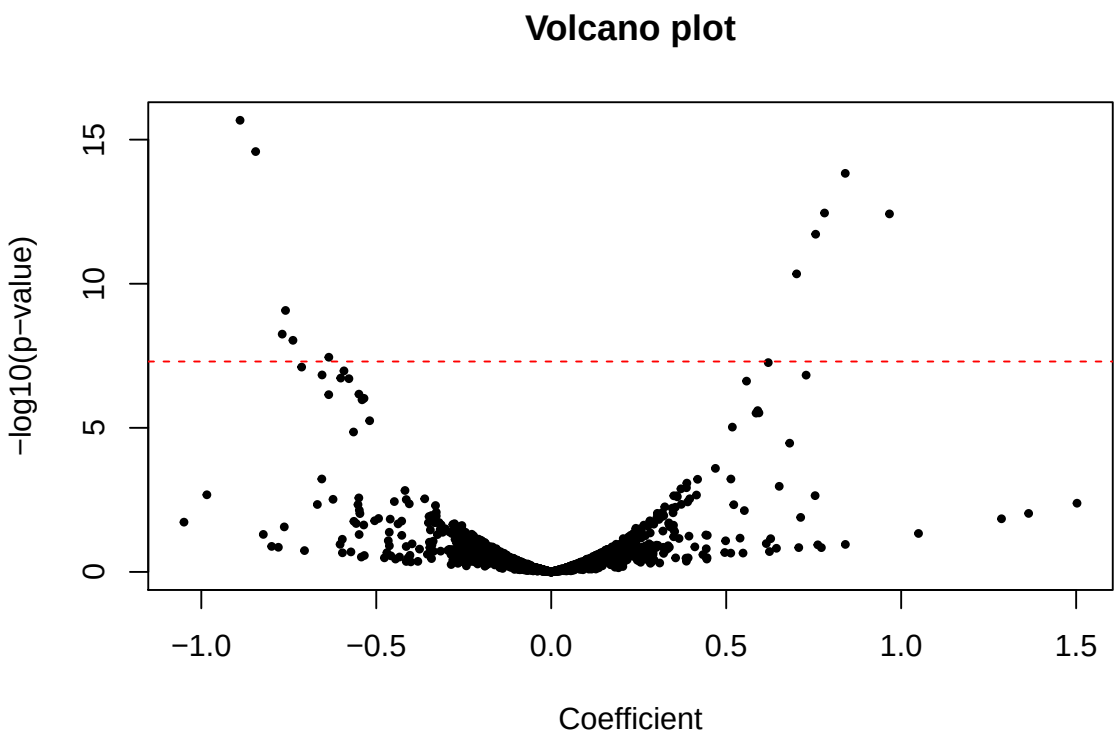
```
> univ.lm.test(chrom21$snp1, chrom21$outcome)
       beta std.error   p.value
x -0.1410125  0.268973 0.6006794
```

Now this can be automated in a loop to run the association test over all SNPs: this may take several seconds, depending on the processor speed (as all models are independent, the process could be trivially parallelized).

```
> crude <- NULL
> for (snp in 2:ncol(chrom21)) { # skip column 1 as it contains the outcome
+   crude <- rbind(crude,
+                  univ.lm.test(chrom21[, snp], chrom21$outcome))
+ }
>
> # add column of snp identifiers
> crude <- cbind(snp=colnames(chrom21)[-1], crude)
```

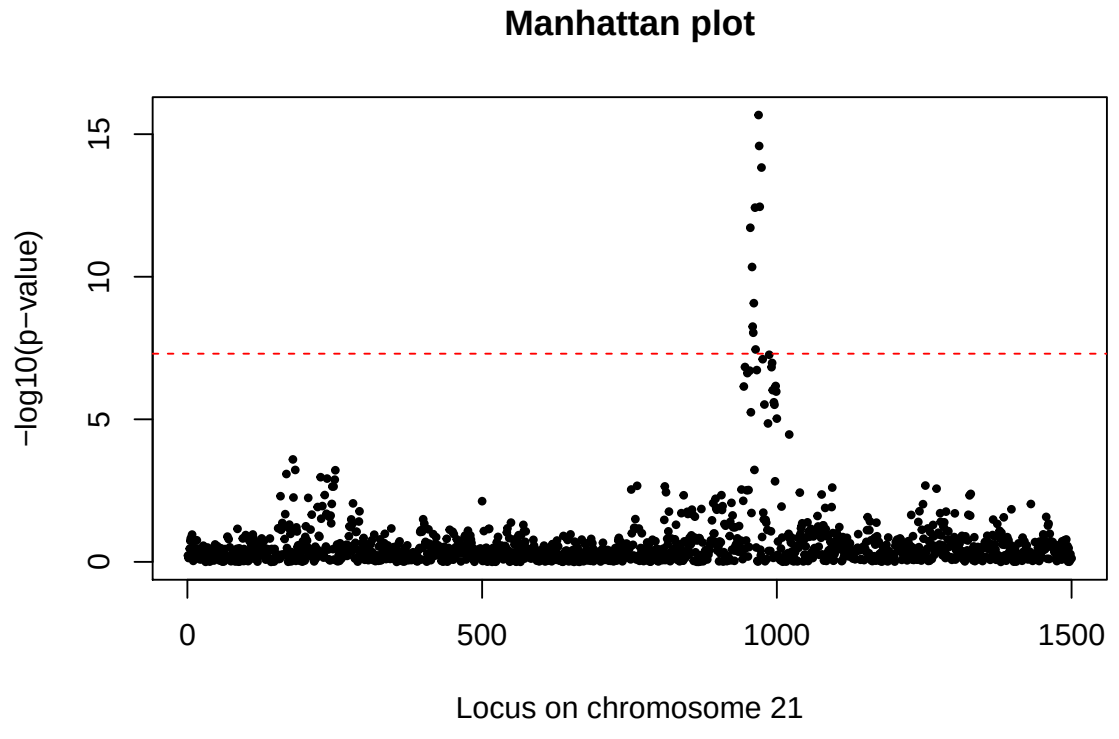
These associations can be visualised through a volcano plot, where strongest associations are those that are towards the top (small p-values) and further to the sides (large effect sizes) of the plot. For reference we also draw a horizontal line corresponding to the standard genome-wide significance level of 5×10^{-8}. Note that in case of a logistic regression model, a volcano plot should use the log-odds ratio (that is, the regression coefficient) on the x axis. Using odds ratios would produce an asymmetric plot, as odds ratios are bounded below by zero, but are unbounded above.

```
> plot(crude$beta, -log10(crude$p.value), main="Volcano plot",
+      xlab="Coefficient", ylab="-log10(p-value)", pch=19, cex=0.5)
> abline(h=-log10(5e-8), lty=2, col="red")   # genome-wide significance threshold
```



A Manhattan plot is better suited to visualize the ordering of SNPs along the chromosome, and it can show that hits generally do not appear alone, but they form spikes because of *linkage disequilibrium* (SNPs close together have more chances of being transmitted together).

```
> plot(-log10(crude$p.value), pch=19, cex=0.5, main="Manhattan plot",
+      xlab="Locus on chromosome 21", ylab="-log10(p-value)")
> abline(h=-log10(5e-8), lty=2, col="red")   # genome-wide significance threshold
```



Note that in this particular example, we did not have genetic positions of each SNP: if we did, we should use them to better represent the spread of SNPs on the chromosome. Also, this dataset contains only one chromosome: if we had more than one, it would be better to differentiate chromosomes by colour, to make it easier to pinpoint where the hits are.

Genetic risk scores

The SNPs that are most strongly associated with a specific trait or disease outcome can be used to build a genetic risk score (or polygenic risk score). A threshold on the p-value for such scores doesn't need to be as strict as the genome-wide significance threshold: often 10^{-5} or 10^{-4} are used, and in some cases those can be relaxed even further.

```
> # subset of snps to enter the genetic risk score
> snps.grs <- subset(crude, p.value < 1e-5)
> chrom21.grs <- chrom21[, colnames(chrom21) %in% snps.grs$snp]
>
> # ensure that the ordering of snps is respected
> stopifnot(colnames(chrom21.grs) == snps.grs$snp)
```

We can build a simple genetic risk score for each patient simply by summing the allele counts across all the associated SNPs (*unweighted score*). To take the direction of the effect into consideration, we switch the sign for the SNPs that have an inverse association: we use function `sign()` for this purpose, which returns 1 for positive elements, and -1 for negative elements.

```
> unweighted.score <- as.matrix(chrom21.grs) %*% sign(snps.grs$beta)
```

Once a score is built, it can be considered as an additional predictor and tested for association with the outcome.

```
> mod.unweighted <- lm(chrom21$outcome ~ unweighted.score)
> coef(summary(mod.unweighted))
                   Estimate  Std. Error    t value     Pr(>|t|)
(Intercept)      -0.04036358 0.071647896 -0.5633603 5.738238e-01
unweighted.score  0.04819855 0.005162174  9.3368699 2.022963e-17
```

More commonly, however, the SNPs that enter a score are weighted by their regression coefficient, so that not only the direction of the effect is taken into consideration, but also so that loci with larger effect contribute more to the final score. Whenever people refer to genetic (or polygenic) risk scores, it can be assumed that they refer to the weighted version.

```
> weighted.score <- as.matrix(chrom21.grs) %*% snps.grs$beta
> mod.weighted <- lm(chrom21$outcome ~ weighted.score)
> coef(summary(mod.weighted))
                 Estimate  Std. Error    t value     Pr(>|t|)
(Intercept)    -0.06098417 0.070183136 -0.8689291 3.859327e-01
weighted.score  0.07359710 0.007616342  9.6630505 2.345462e-18
```

The weighted score seems to perform marginally better, not only as measured by the Wald test p-value, but also by looking at the adjusted R^2: 0.316 for the weighted score versus 0.301 for the unweighted score. This tells us that the weighted score explains a bit more of the variance in the outcome.

Note that we cannot compare the regression coefficients directly, as the two scores are scaled differently: either we should have standardized the two scores to have the same standard deviation before fitting the models (as we did in Lab 2), or we correct the regression coefficients by multiplying each of them by the standard deviation of their corresponding predictor. Here we follow the latter approach, so the standardized regression coefficients become:

```
> # standardized regression coefficients
> beta.unweighted.std <- coef(mod.unweighted)[2] * sd(unweighted.score)
> beta.weighted.std <- coef(mod.weighted)[2] * sd(weighted.score)
> round(c(beta.unweighted.std, beta.weighted.std), 3)
unweighted.score   weighted.score
           0.643            0.658
```

Also according to this criterion, it appears that the weighted score is a marginally better estimate of the genetic risk for the quantitative trait under study.

What would happen if we didn't filter the SNPs that contribute to the genetic risk score? This would consider all the SNPs in the dataset as being related to the outcome under study: in some cases, it's not reasonable to expect that a score computed in this way would perform particularly well. However, complex traits (such as height) may be better explained by a large number of small effects spread throughout the genome than by a reduced set of most strongly associated SNPs.

```
> # ensure that the ordering of snps is respected
> stopifnot(colnames(chrom21)[-1] == crude$snp)
> all.score <- as.matrix(chrom21[, -1]) %*% crude$beta
>
> # standardize the score to allow for direct comparison of effect sizes
> all.score <- all.score / sd(all.score)
> mod.all <- lm(chrom21$outcome ~ all.score)
> coef(summary(mod.all))
             Estimate Std. Error  t value     Pr(>|t|)
(Intercept) 0.0919377 0.05006955  1.83620 6.782005e-02
all.score   0.9520540 0.04751121 20.03851 1.285663e-49
```

In this specific instance, using all SNPs produces a significant improvement in the fit of the model (R^2 is 0.667), as well as in effect size (0.952). However, remember that the true assessment of performance of a genetic risk score, as for any other predictor, can be done only on withdrawn data (on a new set of observations or within a cross-validation setting).

Predicting with scores

Let's consider the case of a single training-test split, which can be created by using function `createDataPartition()` from the `caret` package.

```
> library(caret)
> set.seed(1)
> train.idx <- createDataPartition(chrom21$outcome, p=0.7)$Resample1
```

We will use the training set for two purposes:

1. Run association tests for all the SNPs considered individually, and use the summary statistics to develop a genetic risk score

2. Learn the regression coefficients for a model that uses the score

After this, we will be able to build genetic risk scores for the observations in the test set and make a prediction of the outcome.

We start by testing each SNP for association with the outcome: this uses the same approach developed earlier, but this time we restrict the data to the training set.

```
> chrom21.train <- chrom21[train.idx, ]
> crude <- NULL
> for (snp in 2:ncol(chrom21.train)) {
+   crude <- rbind(crude,
+                  univ.lm.test(chrom21.train[, snp], chrom21.train[, 1]))
+ }
>
> # add column of snp identifiers
> crude <- cbind(snp=colnames(chrom21.train)[-1], crude)
```

Now we can create a weighted score by using the top SNPs. Note that we do it for all samples, also those in the test set: this is fine, we are not using the outcome variable, but just summing up the columns of the SNP matrix weighted by the coefficients learnt on the training set. Having a score for all samples will make it easier further on when testing the performance of the score.

```
> snps.grs <- subset(crude, p.value < 1e-5)
> chrom21.grs <- chrom21[, colnames(chrom21) %in% snps.grs$snp]
> weighted.score <- as.matrix(chrom21.grs) %*% snps.grs$beta
```

It's time to fit a model with the score variable: here it's important to use only the training set (defined by the `subset` option), as we are using the outcome to learn the regression coefficients.

```
> model.score <- lm(outcome ~ weighted.score, data=chrom21, subset=train.idx)
```

We are finally ready to predict the observations in the test set. Given that the `weighted.score` is outside of the `chrom21` dataset, the easiest way to achieve this, is to predict all observations and then discard those that are in the training set.

```
> pred <- predict(model.score, newdata=chrom21)[-train.idx]
```

As a measure of performance we can compute the correlation coefficient between predicted and observed outcome (restricted to samples in the test set), or the square of that (corresponding to the R^2).

```
> round(cor(pred, chrom21$outcome[-train.idx]), 3)    # correlation
[1] 0.626
> round(cor(pred, chrom21$outcome[-train.idx])^2, 3)  # r-squared
[1] 0.392
```

GWAS meta-analysis

When genome-wide association studies of the same phenotype are performed in multiple independent studies, it is common to merge the results by performing a *meta-analysis*. This has the benefit of increasing the statistical power and reducing false-positive findings (type I error rate). Again, this type of analysis is most often done through specialized software (such as plink, METAL, GWAMA, etc) which can perform several checks on the input files and produce helpful diagnostics.

Before being able to perform a meta-analysis, the summary statistics from the studies need to be harmonized, to ensure that the effect sizes reported refer to the same allele. Thus, if two studies used a different effect allele, the sign for the effect size of one of them should be switched.

Files `OPN_sub_study1.txt` and `OPN_sub_study2.txt` contain a small subset of results from two different GWASs run on Osteopontin (OPN) a protein that can be measured in blood serum.

```
> # set the stringsAsFactors options to avoid generating factors for alleles
> gwas1 <- read.delim("data/OPN_sub_study1.txt", stringsAsFactors=FALSE)
> gwas2 <- read.delim("data/OPN_sub_study2.txt", stringsAsFactors=FALSE)
```

We need to harmonize the two datasets: in this specific case it's enough to order them by chromosome and position, but we need to ensure that the SNP identifiers correspond.

```
> snps.in.both <- intersect(gwas1$rsid, gwas2$rsid)
> gwas1 <- subset(gwas1, rsid %in% snps.in.both)
> gwas2 <- subset(gwas2, rsid %in% snps.in.both)
>
> # order by chromosome and position
> gwas1 <- gwas1[with(gwas1, order(chromosome, position)), ]
> gwas2 <- gwas2[with(gwas2, order(chromosome, position)), ]
> stopifnot(all.equal(gwas1$rsid, gwas1$rsid))
```

Ignore the differences in position between the two studies: they may have been carried out using different genome builds (the relative order of SNPs is unaffected). What matters is that SNP names and both alleles match, and that both studies report the effect size on the same allele. In this example, both studies considered A1 to be the effect allele, but different studies may have different conventions, and they need to be harmonized as well in case of discrepancies.

We start by finding which SNPs have both alleles matching, and which would match if the alleles were flipped.

```
> both.ok <- gwas1$A1 == gwas2$A1 & gwas1$A2 == gwas2$A2
> flipped <- gwas1$A1 == gwas2$A2 & gwas1$A2 == gwas2$A1
> table(both.ok, flipped)
       flipped
both.ok FALSE TRUE
  FALSE     0   25
  TRUE     27    0
```

In this case, we can see that the alleles either match already, or they match after flipping them. If both alleles do not match even after swapping them, that SNP cannot be meta-analysed. This could be caused by a genotyping or imputation error, or because of some other discrepancy in the GWAS pipelines adopted by the different studies.

The effect sizes of the SNPs which we identified to have alleles flipped need to have their direction of effect swapped in one of the studies before entering the meta-analysis. Here we swap the sign for the second study, but this is an arbitrary choice.

```
> beta1 <- gwas1$Effect_A1
> beta2 <- gwas2$Effect_A1
> beta2[flipped] <- -beta2[flipped]
```

Finally we are ready to perform a fixed effect meta-analysis by using inverse variance weighting.

```
> weight.gwas1 <- 1 / gwas1$StdErr_A1^2
> weight.gwas2 <- 1 / gwas2$StdErr_A1^2
```

By looking at the weights assigned to the two studies, it appears that in general the second study is more powered.

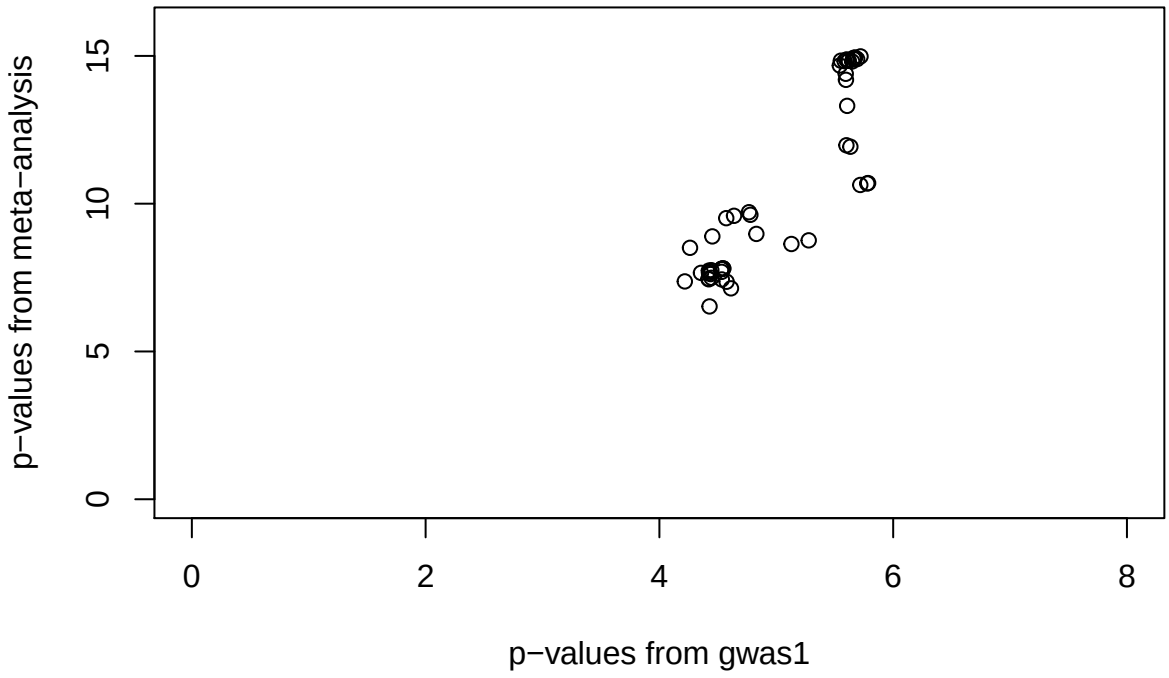
```
> head(weight.gwas1)
[1] 929.5062 734.4247 730.4602 935.2000 935.2000 929.5062
> head(weight.gwas2)
[1] 1339.100 1104.129 1118.638 1401.177 1379.630 1348.865
```

The compuation of the meta-analysis effect size is a weighted sum of the effect sizes from each study, weighted according to the weight just derived.

```
> beta.ma <- (weight.gwas1 * beta1 + weight.gwas2 * beta2) / (weight.gwas1 + weight.gwas2)
> se.ma <- sqrt(1 / (weight.gwas1 + weight.gwas2))
```

Note that by combining the two studies through a meta-analysis we have increased the power. This can be seen by comparing the p-values of the first study (which has them available already) to the meta-analysis p-values.

```
> pval.ma <- 2 * pnorm(abs(beta.ma / se.ma), lower.tail=FALSE)
> plot(-log10(gwas1$P.value), -log10(pval.ma), xlim=c(0, 8), ylim=c(0, 16),
+      xlab="p-values from gwas1", ylab="p-values from meta-analysis")
```



Practical exercises

1. Using file `chrom21.csv` (available on Learn):

   - Impute the missing values in the SNPs.
   - Set the random seed to 1 and create 5 cross-validation folds.
   - Within each training fold, perform an association study for all SNPs in the dataset (hint: store the dataframe of results as an element in a list) and report the most associated SNP (answer: snp971, snp969, snp963, snp969, snp969).
   - Within each training fold, build a weighted genetic risk score that uses SNPs filtered at p-value $< 10^{-4}$, as well as a weighted genetic risk score that uses all available SNPs (hint: build the scores for all observations, also those in the test set).
   - Within each training fold, fit a linear regression models for each score (hint: store each regression object as an element in a list).
   - Use the two genetic risk scores to predict the outcome in each of the test folds and compute the correlation coefficient between the observed outcome and the predicted outcome. Report the average correlation over all folds (answer: 0.558, 0.799).

2. Using file `dpt2.txt` (available on Learn):

   - Sort the dataframe in increasing order of chromosome number and base pairs (column "pos").
   - Using function `as.numeric()`, convert the "pos" column from integer to numeric (to avoid overflow). Add column "cum.pos" to the dataframe to contain the cumulative sum of the positions (use function `cumsum()`), and assert that no NAs were produced.
   - Create a Manhattan plot that orders the SNPs according to their cumulative position along the genome.
   - Redo the plot by using alternating colours for adjacent chromosomes, and add a horizontal line corresponding to the standard genome-wide significance threshold.
   - Produce a volcano plot for this set of results, using a different colour for SNPs that have minor allele frequency (MAF) $< 5\%$.