EMBO Population Genomics Practical 1

Andrea Manica

Introduction

For this practical, we will use `admixtools` v2, an R package that reimplements and expands the methods originally found in the classic ADMIXTOOLS program. The advantage of using `admixtools` v2 is that we can use R to issue commands and directly inspect and plot the outputs (a workflow in classic ADMIXTOOLS consisted a combination of sed/awk/shell scripting and manual editing of input files).

Combining and inspecting data to be passed to admixtools is also a challenge. These data often come in different formats, and might have different properties (e.g. modern vs ancient data). Data wrangling is often done in PLINK, but PLINK was not really designed for this type of work (it focuses on GWAS, not generic pop gen). In this practical, we will use `tidypopgen`, a new package developed to provide a clear grammar of population genetics, making data wrangling user friendly. `tidypopgen` has specific functions to prepare data for `admixtools`, and so the two blend together into a single R-centric workflow.

Using `tidypopgen` to wrangle data

`tidypopgen` follows the "tidyverse" logic and syntax. It is designed to make the processing and analysis of genetic data easy, reproducible and within a single programming language. The basic idea behind tidy data is that each observation should have its own row and each variable its own column, such that each value has its own cell. Applying this logic to population genetic data means that each individual should have its own row, with individual metadata (such as its population, sex, phenotype, etc) as the variables. Genotypes for each locus can also be thought of as variables, however, due to the large number of loci and the restricted values that each genotype can take, it would be very inefficient to store them as individual standard columns. They are stored in a file on disk, called a File Backed Matrix (FBM).

Hence, in `tidypopgen`, we represent data as a `gen_tbl`, a subclass of `tibble` which has two compulsory columns: `id` of the individual (as a `character`, which must be unique for each individual), and `genotypes` (stored in a compressed format as a File-Backed Matrix, with the vector in the tibble providing the appropriate link to those data).

```
library(tidypopgen)
```

```
## Loading required package: dplyr
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
## Loading required package: tibble
```

```
library(admixtools)
library(ggplot2)
```

Data types

This is additional reference information. It is not needed to run the practical, so you can skip it now, but it can be useful if you want to poke into the data files.

In this practical, we will use two types of data: PLINK binary BED files, and VCF (Variant Call Format) files. If you work with humans, you might also encounter PackedAncestry files. All these three formats can be read into a `gen_tibble` in `tidypopgen`.

PLINK binary files are used to store genotype data in a compact and efficient binary format. They typically consist of three main files:

`.bed` file: Contains the binary genotype data. `.bim` file: Contains variant information. `.fam` file: Contains individual sample information.

The `.bed` file stores the genotype data in a compact binary format. It does not have a human-readable structure but is designed for efficient storage and access by PLINK and other compatible tools.

Genotype Encoding:

00: Homozygous for the reference allele (AA) 01: Missing genotype 10: Heterozygous (AB) 11: Homozygous for the alternate allele (BB)

The `.bim` file is a text file that contains variant information. Each row corresponds to a variant, and it has the following columns:

`chrom`: Chromosome number or ID `variant ID`: Unique identifier for the variant (e.g., rsID) `genetic distance`: Genetic distance (can be set to 0 if not available) `base-pair position`: Physical position of the variant on the chromosome `allele 1`: Reference allele (usually coded as the minor allele) `allele 2`: Alternate allele

The `.fam` file is a text file that contains information about each individual sample in the dataset. Each row corresponds to an individual and has the following columns:

`Family ID`: Identifier for the family (can be set to 0 if not applicable) `Individual ID`: Unique identifier for the individual `Paternal ID`: Identifier for the father (0 if not available) `Maternal ID`: Identifier for the mother (0 if not available) `Sex`: Sex of the individual (1 = male, 2 = female, 0 = unknown) `Phenotype`: Phenotype information (1 = unaffected, 2 = affected, -9 = missing)

VCF files are a standardized text file format widely used in bioinformatics for storing gene sequence variations. They consist of two main sections: the header and the data section. The header begins with ## and provides meta-information about the dataset, such as the version of the VCF file format, reference genome, and other key details. The header ends with a line starting with a single #, which specifies the column names for the data section.

The data section contains rows of variant calls, each representing a genetic variant. The columns, as defined in the header, typically include:

`CHROM`: Chromosome number or ID `POS`: Position of the variant on the chromosome `ID`: Identifier for the variant (e.g., dbSNP ID) `REF`: Reference allele `ALT`: Alternate allele(s) `QUAL`: Quality score of the variant call `FILTER`: Filter status indicating if the variant passed certain quality thresholds `INFO`: Additional information about the variant in a semi-colon-separated list of key-value pairs `FORMAT`: Format of the genotype fields in the subsequent columns `sample1`, `sample2`, ...: Genotype information for each sample, structured as per the FORMAT field

---

Loading the data

In this practical, we will use a panel of modern populations from the Human Origins genetics dataset, which is a collection of genetic data aimed at understanding human evolution, population structure, and migration patterns. The dataset contains genotypes from a wide array of modern human populations.

We will assume that you use a directory structure for your project where code sits in the `code` directory, and data sit in a `data` directory. So, from the `code` directory, we can look at the files in `data` with:

```
dir("./data")
```

```
## [1] "ancient_samples.vcf" "LBK_modern_pops.csv" "modern_samples.bed"
## [4] "modern_samples.bim"  "modern_samples.fam"
```

We can see that there are three files with the prefix 'modern_samples', a .bed, a .bim and a .fam file. To convert them into a `gen_tibble`, we can simply use:

```
modern_gt <- tidypopgen::gen_tibble("./data/modern_samples.bed",
                                    valid_alleles = c("A","T","C","G"),
                                    missing_alleles = c("X"))
```

```
##
## gen_tibble saved to /home/andrea/git/f_stats_practical/data/modern_samples.gt

## using bigSNP file: /home/andrea/git/f_stats_practical/data/modern_samples.rds

## with backing file: /home/andrea/git/f_stats_practical/data/modern_samples.bk

## make sure that you do NOT delete those files!

## to reload the gen_tibble in another session, use:

## gt_load('/home/andrea/git/f_stats_practical/data/modern_samples.gt')
```

The message shows us where in our directories the gen_tibble(.gt) and its backing file (.bk) and R object file (.rds) are saved (these are used to store the genetic data for our `gen_tibble`). You only need to do this once. In the future, if you save your `gen_tibble`, you will be simply able to load it with `gt_load()`.

Let's quickly inspect our data:

```
modern_gt
```

```
## # A gen_tibble: 588768 loci
## # A tibble:     413 x 3
##    id     population  genotypes
##    <chr>  <chr>      <vctr_SNP>
##  1 AD_006 AA                  1
##  2 AD_015 AA                  2
##  3 AD_061 AA                  3
##  4 AD_064 AA                  4
##  5 AD_066 AA                  5
##  6 AD_076 AA                  6
##  7 AD_500 AA                  7
##  8 AD_505 AA                  8
##  9 AD_510 AA                  9
## 10 AD_511 AA                 10
## # i 403 more rows
```

We can see that we have >400 individuals and >500k markers. We can get a tally of how individuals we have per population with:

```
modern_gt %>% group_by(population) %>% tally()
```

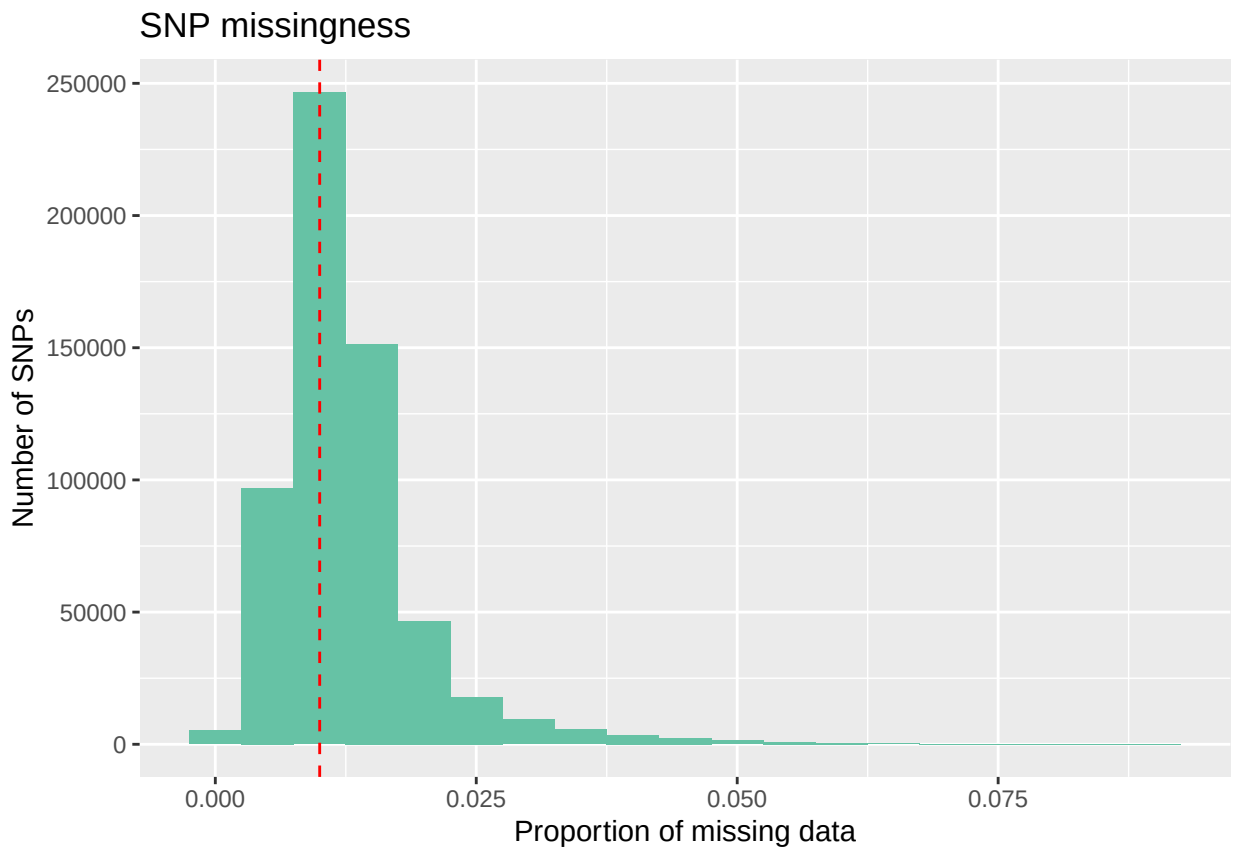
```
## # A tibble: 21 x 2
##    population     n
##    <chr>      <int>
##  1 AA            12
##  2 Basque        29
##  3 Bedouin2      19
##  4 Cypriot        8
##  5 Dinka          7
##  6 Druze         39
##  7 French        25
##  8 Han           33
##  9 Mayan         18
## 10 Mbuti         10
## # i 11 more rows
```

Note that we also have a chimp (pan_troglodytes), which we will use later on as an outgroup for certain analyses.

It is important to perform quality control (QC) on your data. Generally, we will be more stringent on the modern data, and allow more missingness on the ancient data. So, QC is often performed separately on different dataset depending on their nature. `tidypopgen` has two key functions to examine the quality of data, either across loci or across individuals. These functions are `qc_report_loci` and `qc_report_indiv`. We don't have space here to explore quality control in detail, but to exemplify, lets say we want to assess missingness across loci in our datasets. We can create a loci report and visualise our data. We will focus on missingness (the proportion of calls missing for each locus):

```
loci_report <- qc_report_loci(modern_gt)

autoplot(loci_report, type = "missing")
```



There is a tail of a few loci with high missingness. Let's remove all sites that have missingness >= 4%. We can simply do that selecting the loci with missingness less than 0.04:

```
modern_gt <- modern_gt %>%
  select_loci_if(loci_missingness(genotypes)<0.04)
```

The `select_loci_if()` function can be used to filter data based on summary statistics such as missingness or minor allele frequency. In practice, this means that quality control of data can all be completed within R, without switching to command line software and rewriting files. QC is a very important process that we will not cover here, but make sure that you check the quality of your data BEFORE analysing them.

Then we can load our ancient data into a `gen_tibble`, this may take a moment, as we are reading from `vcf` format which is a large text file:

```
ancient_gt <- tidypopgen::gen_tibble("./data/ancient_samples.vcf",
                                     valid_alleles = c("A","T","C","G","X"),
                                     quiet = TRUE)
```

The ancient data contains ancient modern humans as well as Neanderthal and Denisovan samples:

```
ancient_gt$id
```

```
##  [1] "AltaiNea"  "Clovis"    "Denisova"  "GB20"      "Otzi"      "Kostenki"
##  [7] "LBK"       "Loschbour" "MA1"       "UstIshim"
```

Having data in a tibble means that we can easily edit the metadata. Sample GB20 represents Mota, a ~4,000 year old individual from Ethiopia. Let's rename the ID of this sample to be more intuitive:

```
ancient_gt$id[ancient_gt$id == "GB20"] <- "Mota"
```

To use the data with admixtools, we need to assign a 'population' to all of our samples. For now, we can simply duplicate their individual id (so, each sample will be a population):

```
ancient_gt$population <- ancient_gt$id
```

Merging data

Once we are happy with the quality of our independent datasets, we can merge them to perform analyses.

Merging data from different sources is a common problem, especially in human population genetics where there is a wealth of SNP chips available. In `tidypopgen`, merging is enacted with an `rbind` operation between `gen_tibbles`.

If the datasets have the same loci, then the merge is trivial. If not, then it is necessary to subset to the same loci, and ensure that the data are coded with the same reference and alternate alleles (or swap them if needed).

Additionally, if data come from SNP chips, there is the added complication that the strand is not always consistent, so it might also be necessary to flip strand (in that case, ambiguous SNPs have to be filtered out). The `rbind` method for `gen_tibbles` has a number of parameters that allow us to control the behaviour of the merge.

To check our data compatibility before merging, we can run `rbind_dry_run`:

```
merged_dry <- rbind_dry_run(modern_gt, ancient_gt,
                            flip_strand = TRUE)
```

```
## harmonising loci between two datasets
## flip_strand =  TRUE  ; remove_ambiguous =  TRUE
## ------------------------------
## dataset: reference
## number of SNPs: 581472 reduced to 581472
## ( 0 are ambiguous, of which 0  were removed)
## ------------------------------
## dataset: target
## number of SNPs: 588768 reduced to 581472
## ( 0 were flipped to match the reference set)
## ( 0 are ambiguous, of which 0 were removed)
```

The results show that merging these two datasets will cause a loss of around 7,000 SNPs from our 'target' dataset (in this case, the ancient samples as these are given as the second argument to `rbind_dry_run`, and are therefore the 'target').

Given the large overlap between our datasets, we can now merge using `rbind` (we need to give the name of the new backing file):

```
merged_gt <- rbind(modern_gt, ancient_gt,
                   flip_strand = TRUE,
                   backingfile = "./data/merged_samples")
```

```
## harmonising loci between two datasets
## flip_strand =  TRUE  ; remove_ambiguous =  TRUE
## ------------------------------
## dataset: reference
## number of SNPs: 581472 reduced to 581472
## ( 0 are ambiguous, of which 0  were removed)
## ------------------------------
## dataset: target
```

```
## number of SNPs: 588768 reduced to 581472
## ( 0 were flipped to match the reference set)
## ( 0 are ambiguous, of which 0 were removed)
##
## gen_tibble saved to /home/andrea/git/f_stats_practical/data/merged_samples.gt
## using bigSNP file: /home/andrea/git/f_stats_practical/data/merged_samples.rds
## with backing file: /home/andrea/git/f_stats_practical/data/merged_samples.bk
## make sure that you do NOT delete those files!
## to reload the gen_tibble in another session, use:
## gt_load('/home/andrea/git/f_stats_practical/data/merged_samples.gt')
```

f_2 statistics

Now that data are cleaned and merged, we can start our analysis. As all f statistics are combinations of pairs of f_2 statistics, we can begin by pre-computing f_2's for all pairs of populations in our sample. First, we group our data by population, and then we can use `gt_extract_f2` to save our f_2 pairs to disk into the *outdir*. This might take a minute or so.

```
# group test_gt by population
merged_gt <- merged_gt %>% group_by(population)

f2_dir <- "./data/f2_tidypopgen"

# compute f2
f2_tidypopgen <- gt_extract_f2(merged_gt,
                               outdir = "./data/f2_tidypopgen",
                               overwrite = TRUE)
```

```
## 202754 SNPs remain after filtering. 191339 are polymorphic.
## i Allele frequency matrix for 202754 SNPs and 30 populations is 63 MB
## i Computing pairwise f2 for all SNPs and population pairs requires 3796 MB RAM without splitting
## i Computing without splitting since 3796 < 8000 (maxmem)...
## Data written to ./data/f2_tidypopgen/
```

Now we can read read them in using `admixtools`:

```
f2_blocks = f2_from_precomp("./data/f2_tidypopgen")
```

```
## i Reading precomputed data for 30 populations...
## i Reading f2 data for pair 1 out of 465...i Reading f2 data for pair 2 out of 465...i Reading f2 data
```

Ignore the warning about `afprod = TRUE`; it is a consequence of doing all possible f_2 in one go for multiple exercises. Usually, we would compute the f_2 for each project, but it would just take too long for this exercise.

Outgroup f_3

Imagine that we have a new mystery sample that we just sequenced, and we want to know what is the closest population in a panel of already characterised samples. For this exercise, we will use a Neolithic sample from the Linear Band Keramik culture, called LBK. We will use a set of contemporary populations from western Eurasia in our dataset:

```
lbk_modern_panel <- c("Basque", "Bedouin2", "Druze", "Cypriot", "Tuscan",
  "Sardinian", "French", "Spanish", "Onge", "Han", "Mayan", "Mixe", "Surui")
```

Now we compute the outgroup f_3, using Mbuti (a divergent African population) as an outgroup:

```
lbk_f3out <- f3(data = f2_blocks,
                pop1 = "Mbuti",
                pop2 = "LBK",
                pop3 = lbk_modern_panel)
lbk_f3out
```

```
## # A tibble: 13 x 7
##    pop1  pop2  pop3         est       se     z     p
##    <chr> <chr> <chr>      <dbl>    <dbl> <dbl> <dbl>
##  1 Mbuti LBK   Basque    0.0629 0.000546 115.      0
##  2 Mbuti LBK   Bedouin2  0.0583 0.000521 112.      0
##  3 Mbuti LBK   Cypriot   0.0614 0.000537 114.      0
##  4 Mbuti LBK   Druze     0.0605 0.000527 115.      0
##  5 Mbuti LBK   French    0.0624 0.000541 115.      0
##  6 Mbuti LBK   Han       0.0512 0.000579  88.4     0
##  7 Mbuti LBK   Mayan     0.0531 0.000595  89.3     0
##  8 Mbuti LBK   Mixe      0.0531 0.000626  84.8     0
##  9 Mbuti LBK   Onge      0.0505 0.000616  82.0     0
## 10 Mbuti LBK   Sardinian 0.0638 0.000552 116.      0
## 11 Mbuti LBK   Spanish   0.0621 0.000533 117.      0
## 12 Mbuti LBK   Surui     0.0533 0.000668  79.8     0
## 13 Mbuti LBK   Tuscan    0.0626 0.000544 115.      0
```

Now, let's check which populations are closest to LBK (have the highest f_3)

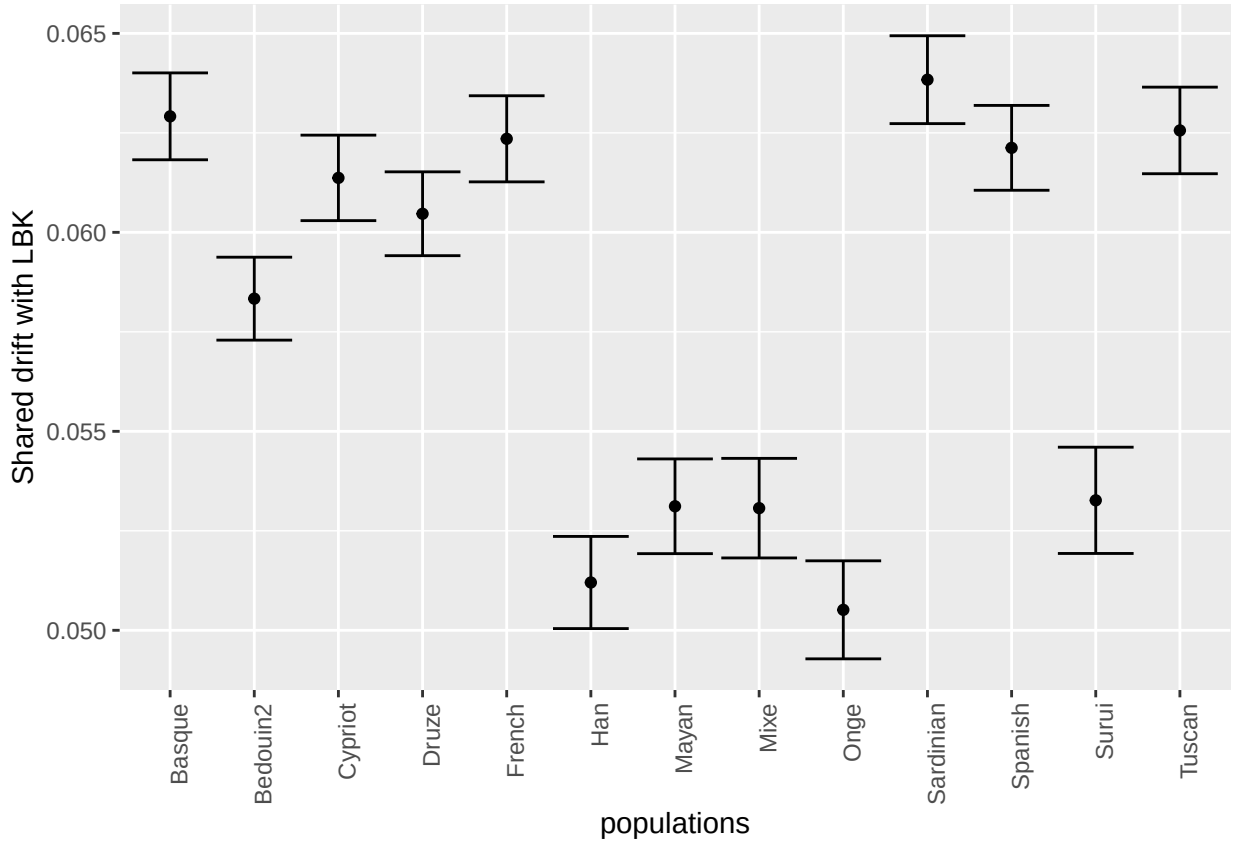
```
lbk_f3out %>% arrange(desc(est))
```

```
## # A tibble: 13 x 7
##    pop1  pop2  pop3         est       se     z     p
##    <chr> <chr> <chr>      <dbl>    <dbl> <dbl> <dbl>
##  1 Mbuti LBK   Sardinian 0.0638 0.000552 116.      0
##  2 Mbuti LBK   Basque    0.0629 0.000546 115.      0
##  3 Mbuti LBK   Tuscan    0.0626 0.000544 115.      0
##  4 Mbuti LBK   French    0.0624 0.000541 115.      0
##  5 Mbuti LBK   Spanish   0.0621 0.000533 117.      0
##  6 Mbuti LBK   Cypriot   0.0614 0.000537 114.      0
##  7 Mbuti LBK   Druze     0.0605 0.000527 115.      0
##  8 Mbuti LBK   Bedouin2  0.0583 0.000521 112.      0
##  9 Mbuti LBK   Surui     0.0533 0.000668  79.8     0
## 10 Mbuti LBK   Mayan     0.0531 0.000595  89.3     0
## 11 Mbuti LBK   Mixe      0.0531 0.000626  84.8     0
## 12 Mbuti LBK   Han       0.0512 0.000579  88.4     0
## 13 Mbuti LBK   Onge      0.0505 0.000616  82.0     0
```

QUESTION: Which populations share the most drift with LBK?

You could produce a nice plot by using:

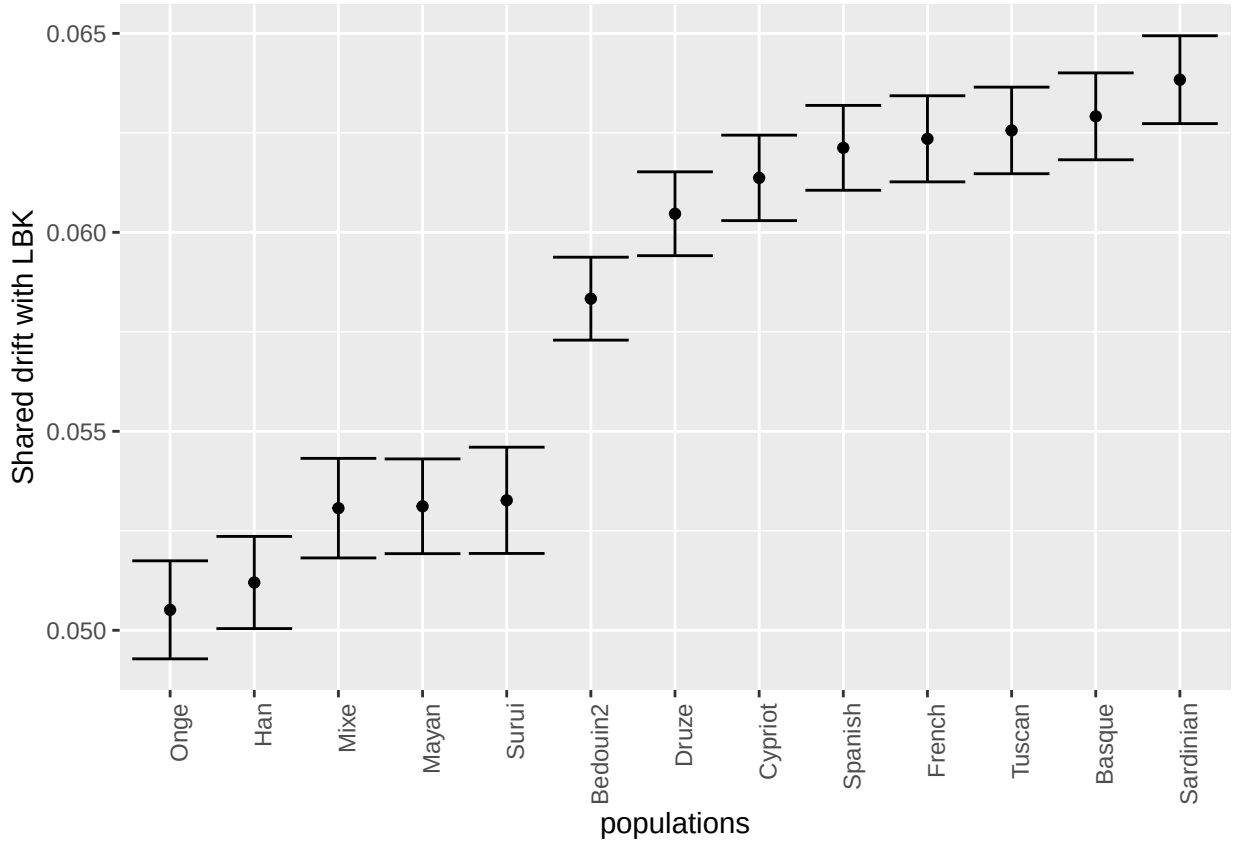
```
ggplot(lbk_f3out, aes(pop3, est)) +
  geom_point() +
  geom_errorbar(aes(ymin = est - 2 * se, ymax = est + 2 * se)) +
  labs(y = "Shared drift with LBK", x = "populations") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



But it would be nicer if our populations were ordered by the level of shared drift. To do so, we need to reorder the levels of the population factor:

```
lbk_f3out$pop3<-factor(lbk_f3out$pop3, levels = lbk_f3out$pop3[order(lbk_f3out$est)])

ggplot(lbk_f3out, aes(pop3, est)) +
  geom_point() +
  geom_errorbar(aes(ymin = est - 2 * se, ymax = est + 2 * se)) +
  labs(y = "Shared drift with LBK", x = "populations") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



QUESTION: Now we have another interesting sample from Central Asia that came into our lab, called 'MA1'. Investigate its relationship with contemporary populations in your sample.

Admixture f_3

We now want to test whether a dataset of African Americans has detectable European ancestry (in other words, can they be modelled as a mixture of an African and a European population):

```
aa_f3admix <- f3(data = f2_blocks,
                 pop1 = "AA",
                 pop2 = "Yoruba",
                 pop3 = "French")
aa_f3admix
```

```
## # A tibble: 1 x 7
##   pop1  pop2   pop3        est       se     z     p
##   <chr> <chr>  <chr>     <dbl>    <dbl> <dbl> <dbl>
## 1 AA    Yoruba French -0.00472 0.000120 -39.4     0
```

QUESTION: Do we have evidence for admixture?

Now, we want to test admixture in two East African target populations (Somali from Somalia, Dinka from Sudan). Source populations are Mota (~4,000 year old individual from Ethiopia) and different modern and ancient (LBK) Eurasian populations:

```
eurasian_sources <- c("French","Spanish","Sardinian","LBK")
```

We first test the Somali against all possible combinations of Mota and these Eurasian sources.

```
somali_f3admix <- f3(data = f2_blocks,
                     pop1 = "Somali",
                     pop2 = eurasian_sources,
                     pop3 = "Mota")
somali_f3admix
```

```
## # A tibble: 4 x 7
##   pop1   pop2      pop3       est       se     z         p
##   <chr>  <chr>     <chr>    <dbl>    <dbl> <dbl>     <dbl>
## 1 Somali French    Mota  -0.00456 0.000219 -20.8 2.27e- 96
## 2 Somali LBK       Mota  -0.00511 0.000353 -14.5 1.64e- 47
## 3 Somali Sardinian Mota  -0.00495 0.000216 -22.9 4.24e-116
## 4 Somali Spanish   Mota  -0.00450 0.000207 -21.7 7.86e-105
```

QUESTION: What can you conclude?

QUESTION: Now do the same for the Dinka. Do you get a similar result? If not, what can we conclude?

f_4 or D statistics

Another way to look at admixture is to use the f_4 or D statistics. They do the same thing, and thus tend to give very similar results. D stats are easier to interpret in terms of magnitude (as they are standardised), but they can not be computed from f_2. You can compute D stats if you start again from a file of genotypes (you would need to export a .bed file from the `gen_tibble` to be read directly in admixtools), but for this exercise we will use f_4, as we already have all the f_2.

For example, we might be interested in gene flow from a certain source, and we want to ask whether a population received it or not. The classical example is Neanderthal admixture into Eurasians. So, we want to ask whether Eurasian populations have more Neanderthal ancestry than Africans. We set up a test as:

```
neand_french_f4 <- f4(data = f2_blocks,
                     pop1 = "pan_troglodytes",
                     pop2 = "AltaiNea",
                     pop3 = "Mbuti",
                     pop4 = "French")
neand_french_f4
```

```
## # A tibble: 1 x 8
##   pop1            pop2     pop3  pop4       est       se     z          p
##   <chr>           <chr>    <chr> <chr>    <dbl>    <dbl> <dbl>      <dbl>
## 1 pan_troglodytes AltaiNea Mbuti French 0.00148 0.000328  4.50 0.00000696
```

So, we can see that there is excess Neanderthal ancestry in the French. Now do the same for Han Chinese (population 'Han')

QUESTION: Now we want to investigate the extend of Yamnaya ancestry that came into Europe during the Bronze age. If we assume that the LBK was the predominant genetic ancestry in Europe before the Yamnaya arrived, test: 1. that "French", "Basque" and "Spanish" all harbour Yamnaya ancestry 2. whether these populations differ in how much ancestry they share

Can you avoid writing down every comparison individually, and use vectors of populations to create some simple code to ask each question (look at what we did with the outgroup f_3).

```
# Ignoring the comparisons of one population against itself, we see that
# French and Basque do not differ in their amount of Yamnaya ancestry, but they
# do differ from Spanish (in accordance to the test above).
```

f_4 ratio estimation

Let's go back to the question of European admixture into African Americans. We will use an f_4 ratio to estimate the proportion of European admixture. The order of populations is important. We have the sister taxon and the outgroup on the left, the target population in the centre, then the sister population, and the source of admixture to the right.

```
pops <- c("Han", "pan_troglodytes", "AA","Yoruba","French")
qpf4ratio(data = f2_blocks,
          pops = pops)
```

```
## # A tibble: 1 x 8
##   pop1  pop2            pop3  pop4   pop5   alpha      se     z
##   <chr> <chr>           <chr> <chr>  <chr>  <dbl>   <dbl> <dbl>
## 1 Han   pan_troglodytes AA    Yoruba French 0.158 0.00522  30.3
```

QUESTION: What is the proportion of European contribution into African Americans in the dataset?

The order of our 'pops' is important. The population of interest, or target, is African American ('AA') and the population hypothesised to admix into the target are French. Here, we also use Yoruba as a proxy source population for African ancestry in African Americans (i.e. its ancestral source). You might want to draw a tree to make sure you are happy with who is going where.

- SISTER: refers to a sister population, one is related more closely to your hypothesised source of admixture than to the ancestral population of our target (in this case Han)
- OUTGROUP: refers to an outgroup to all other populations (in this case Pan troglodytes)
- TARGET: the admixed population under investigation (in this case, African American)
- ANCESTRAL: refers to the ancestral population that we know to be related to our admixed target (in this case, Yoruba)
- HYPOTHESISED SOURCE OF ADMIXTURE: refers to the population who is suggested to have admixed into the target (in this case, French)

QUESTION: Can you write some code to compute the f_4 ratio by hand (i.e. estimate the numerator and denominator f_4, and compute the ratio by hand)?

QUESTION: Repeat the analysis using other European populations (Sardinian, Spanish and Basque). Do the results change when using different European populations?

QUESTION: A final little challenge. `qpf4ratio` can take a matrix of populations, where each line gives the 5 pops for an f_4 ratio. Can you write the code to run the 4 comparisons we just run as a single command?

Combining outgroup f_3 with PCA

In case you have extra time, or simply do it at home.

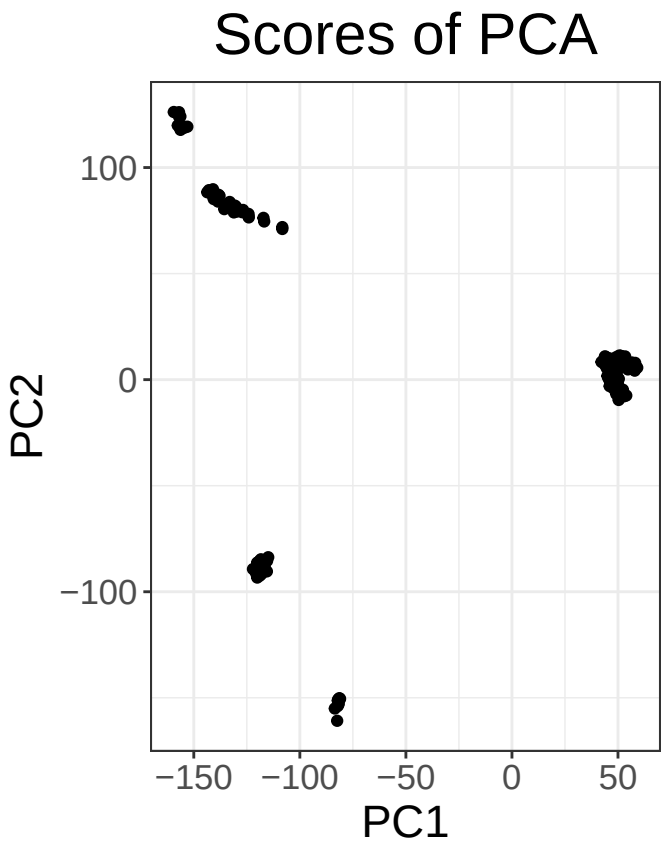
We should always explore our data before running f statistics. In the practical above we skipped that step as it would have taken too long. Let's have a look at the outgroup f_3 analysis we did to place LBK, and complement it with a PCA

```
lbk_modern_panel <- c("Basque", "Bedouin2", "Druze", "Cypriot", "Tuscan",
  "Sardinian", "French", "Spanish", "Onge", "Han", "Mayan", "Mixe", "Surui")
modern_panel_gt <- merged_gt %>% filter (population %in% lbk_modern_panel)
# remove monomorphic sites (ungrouping the tibble, as we want to use global frequencies
# rather than population frequencies)
modern_panel_gt <- modern_panel_gt %>% ungroup() %>% select_loci_if(loci_maf(genotypes)>0)
# pca can not cope with missing data
# tidypopgen wants you to explicitly impute, so that you know what you are doing
modern_panel_gt <- modern_panel_gt %>% gt_impute_simple(method="mode")
# filter for LD
```

```
modern_panel_gt <- modern_panel_gt %>% select_loci_if(loci_ld_clump(genotypes))
# fit the PCA
modern_pca <- modern_panel_gt %>% gt_pca_randomSVD()
```

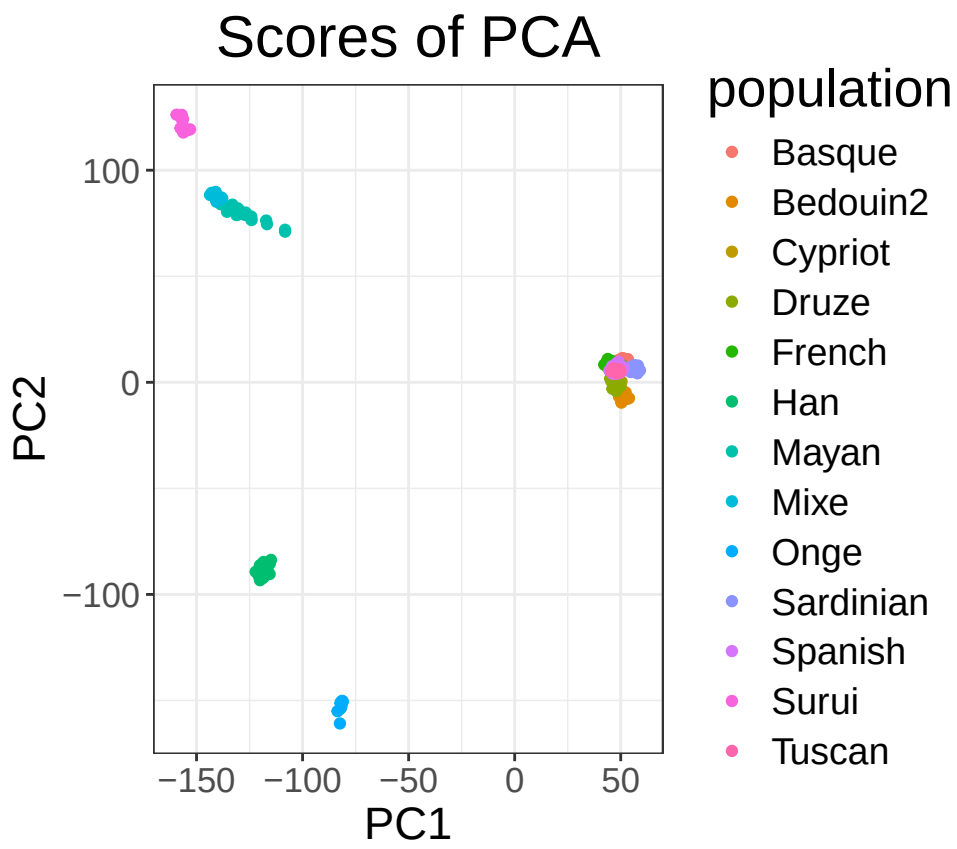
We can now quickly plot the PCA:

```
autoplot(modern_pca, type = "scores")
```



To colour our points, we can simply decorate the basic `autoplot()` output with `ggplot2`

```
library(ggplot2)
autoplot(modern_pca, type = "scores") +
  aes(color = modern_panel_gt$population) +
  labs(color = "population")
```



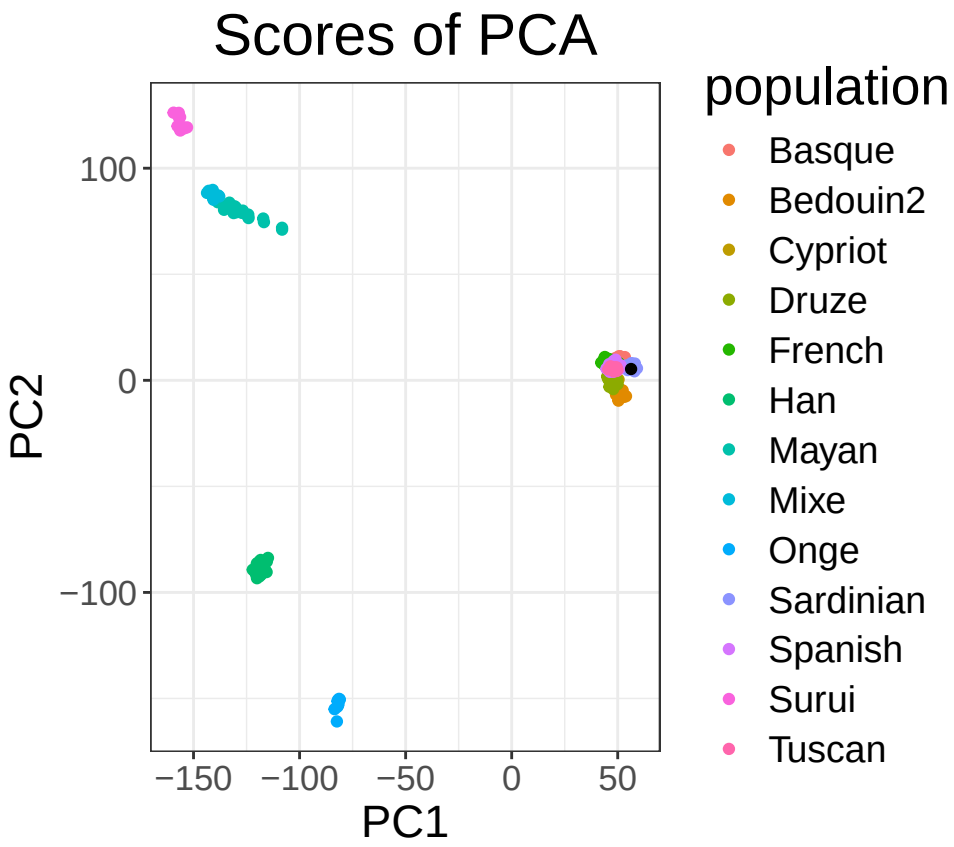
We can now project the LBK onto the modern PCA:

```
lbk_gt <- merged_gt %>% filter(id == "LBK")
lbk_pca_scores <- predict(modern_pca, new_data = lbk_gt, project_method = "least_square")
lbk_pca_scores
```

```
##          .PC1     .PC2
## LBK 56.30226 5.324804
```

And now plot it:

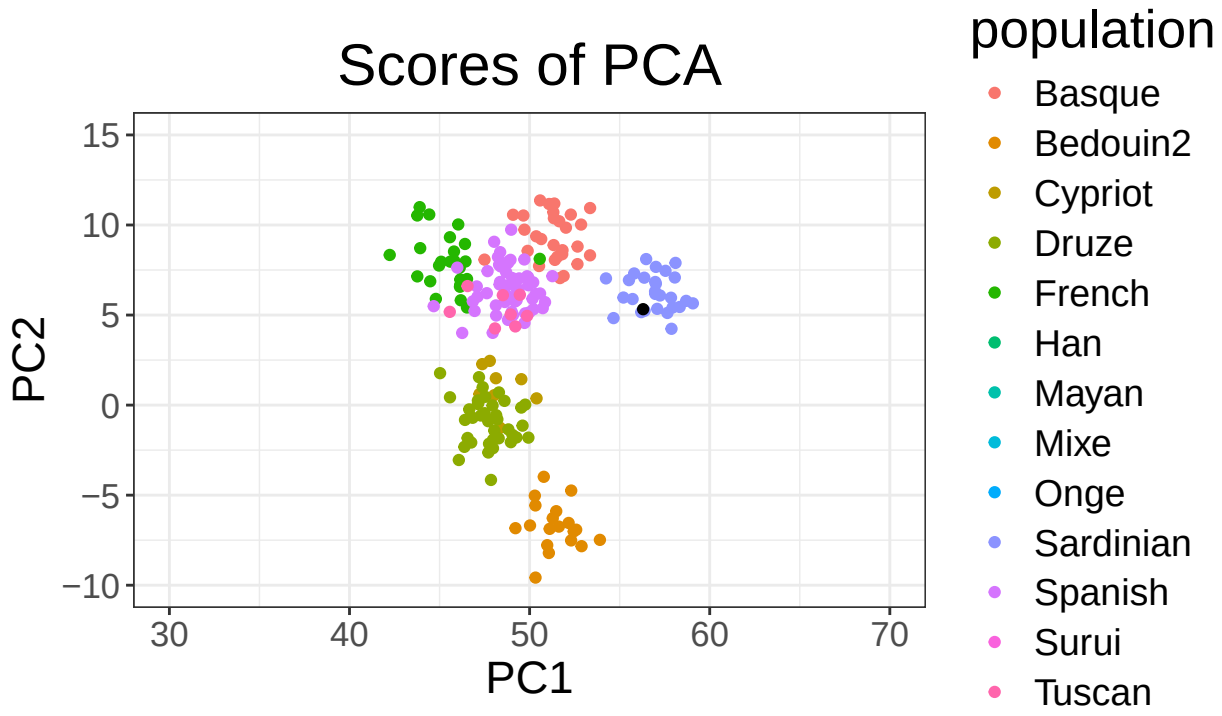
```
autoplot(modern_pca, type = "scores") +
  aes(color = modern_panel_gt$population) +
  labs(color = "population")+
  geom_point(data=lbk_pca_scores, mapping=aes(x=.data$.PC1, y=.data$.PC2), col = "black")
```



Let us zoom in onto Europe

```
autoplot(modern_pca, type = "scores") +
  aes(color = modern_panel_gt$population) +
  labs(color = "population") +
  geom_point(data=lbk_pca_scores, mapping=aes(x=.data$.PC1, y=.data$.PC2), col = "black") +
  lims(x=c(30, 70), y = c(-10, 15))
```

```
## Warning: Removed 84 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



We can see that LBK falls squarely onto the Sardinians, confirming our f_3 analysis.