Basic management and analysis of genome-wide data for Genetic Anthropology

Dang Liu

29-30 November 2023

Introduction

In this workshop, we will learn the first few steps one can do when they have generated genome-wide SNP (single nucleotide polymorphism) data from genotyping arrays. We will use a small subset of data being analyzed in our previous study Kutanan, Liu et al 2021. I divided this subset into two more subsets, reference populations (published data that we wanna include to have reference genetic sources) and studying populations (new data that we generated in the study). So, the steps are as following: 1) merge the two dataset 2) perform quality control 3) analyze population structure by principal component analysis (PCA) and model-based clustering ADMIXTURE

If you have experience in Linux cluster systems, you can try to follow all steps. For the other beginners, you can just focus on how to use the generated outputs to analyze the data and visualize the results. But, it is important for all of you to understand the purpose of each step.

Software

- **R and Rstudio** - R programing language and a super friendly R working space
  - R package **tidyverse** - a super useful R package allows you to read, manage, and visualize your data
  - R package **ggmap** - another useful R package to plot maps
- **plink1.9** - a powerful tool for genome-wide data management (optional)
- **plink2** - an under-development new version of plink, useful for some quality control step, such as kinship quality control (optional)
- **run_hardy_withinPop.sh** - a shell script that I wrote for performing Hardy-Weinberg equilibrium test within each population (optional)
- **ADMIXTURE** - a widely used model-based clustering software for population structure analysis

---

Data

Metadata and genotype files

- **pop_info.txt** - metadata for all the individuals in our data

- **Ref_pop.bed/bim/fam** - plink format data for reference populations
- **Study_pop.bed/bim/fam** - plink format data for studying populations

QC files

- **Merged_Study_Ref.missing.lmiss** - the statistics of missing data for all the SNPs
- **Merged_Study_Ref.missing.imiss** - the statistics of missing data for all the individuals
- **Merged_Study_Ref.kin.kin0** - the statistics of kinship coefficients for all individual pairs

PCA files

- **Pruned_QC_Study_Ref.pca.eigenval** - the final pca eigenvalues
- **Pruned_QC_Study_Ref.pca.eigenvec** - the final pca eigenvectors

ADMIXTURE files

- **ind.pop.list** - a list of individuals and their population labels, the same order as the plink fam file for running ADMIXTURE
- **Pruned_QC_Study_Ref.cv.error** - the cross-validation error for ADMIXTURE runs of K=2 to K=5
- **Pruned_QC_Study_Ref.[K].Q** - ADMIXTURE output Q file, the estimated propotions for each K for each individual from K=2 to K=5

Sample Information

Let's start with learning the sample information from our metadata (pop_info.txt). A well-documented metadata is important and useful for data management and data visualization. It will also help the reproducibility of our results. So, we will take a look of this file in R.

```
# In R, setup environment first
# load in tidyverse
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.4.4     v purrr   1.0.2
## v tibble  3.2.1     v dplyr   1.1.3
## v tidyr   1.3.0     v stringr 1.5.0
## v readr   2.1.1     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(ggmap)
```

```
## i Google's Terms of Service: <https://mapsplatform.google.com>
##   Stadia Maps' Terms of Service: <https://stadiamaps.com/terms-of-service/>
##   OpenStreetMap's Tile Usage Policy: <https://operations.osmfoundation.org/policies/tiles/>
## i Please cite ggmap if you use it! Use `citation("ggmap")` for details.
```

```
# Now, we can read the info file
info <- read_delim("pop_info.txt", delim = "\t", col_names = TRUE)
```

```
## Rows: 103 Columns: 14
## -- Column specification --------------------------------------------------------
## Delimiter: "\t"
## chr (11): FID, IID, Pop, Region, Country, Type, Period, Language, Group, Ref...
```

```
## dbl  (3): PC_code, Latitude, Longitude
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
head(info)
```

```
## # A tibble: 6 x 14
##   FID   IID   Pop      PC_code Region Country  Type   Period  Language  Latitude
##   <chr> <chr> <chr>      <dbl> <chr>  <chr>    <chr>  <chr>   <chr>        <dbl>
## 1 TN201 TN201 HtinPray      15 SEA    Thailand modern Present Austroas~     19.2
## 2 TN207 TN207 HtinPray      15 SEA    Thailand modern Present Austroas~     19.2
## 3 TN209 TN209 HtinPray      15 SEA    Thailand modern Present Austroas~     19.2
## 4 TN213 TN213 HtinPray      15 SEA    Thailand modern Present Austroas~     19.2
## 5 TN219 TN219 HtinPray      15 SEA    Thailand modern Present Austroas~     19.2
## 6 TN223 TN223 HtinPray      15 SEA    Thailand modern Present Austroas~     19.2
## # i 4 more variables: Longitude <dbl>, Group <chr>, Ref <chr>, QC <chr>
```

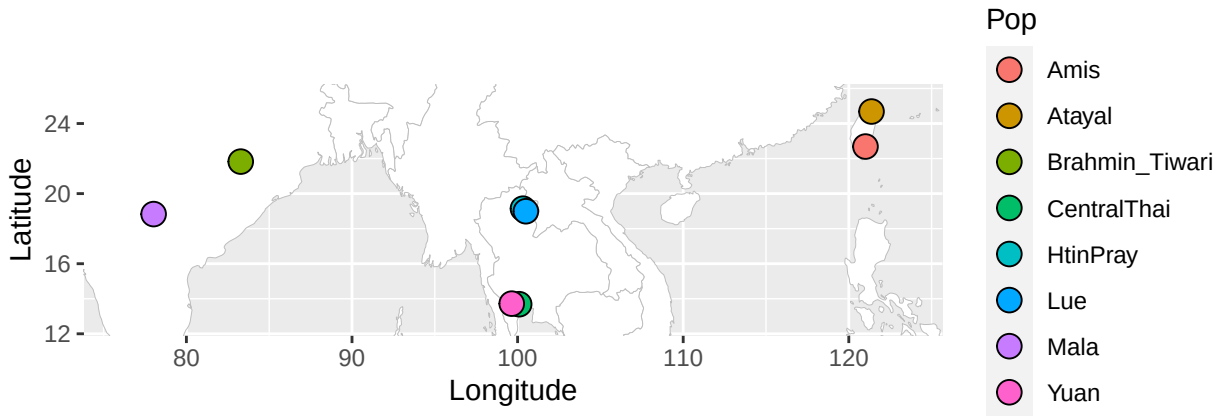
```
# Many information we can get from this info data
# For example, what's the sample size of each population?
table(info$Pop)
```

```
##
##           Amis         Atayal Brahmin_Tiwari    CentralThai       HtinPray
##             10             10             15             20             15
##            Lue           Mala           Yuan
##             10             15              8
```

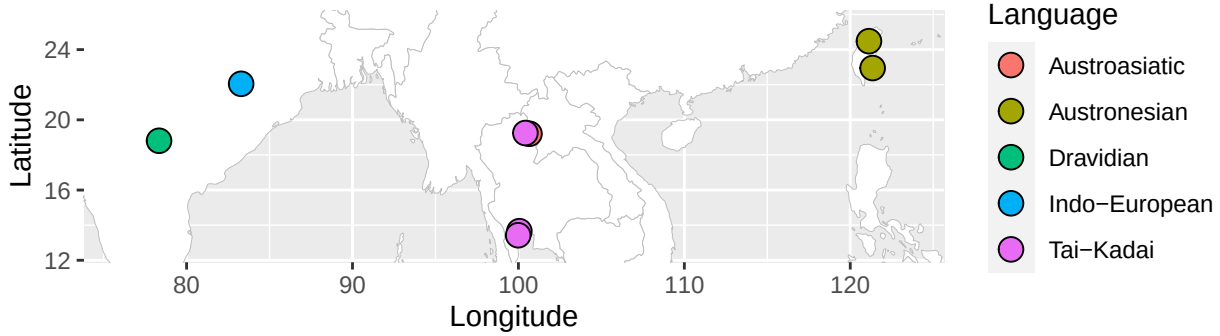
Questions: How many different language groups in our data? How many individuals from different regions or countries?

```
# We can also plot the samples on a map to have an idea where they come from
# get the population median of sampling geo-coordinates of our samples
map_info <- info %>% group_by(Pop) %>%
  summarise_at(vars(Latitude,Longitude), funs(median(.))) %>%
  left_join(select(info,-(FID:IID),-(Latitude:Longitude))) %>%
  distinct(Pop, .keep_all = TRUE)

# Get a world map
map.world <- map_data(map="world")

# zoom-in on the world map according to the sample geo-coordinates
p <- ggplot() +
  geom_map(data = map.world,
           map = map.world,
           aes(map_id = region),
           fill = "white", colour = "grey", size = 0.15) +
  coord_quickmap(
    xlim = c(min(map_info$Longitude - 2, na.rm = TRUE),
             max(map_info$Longitude + 2, na.rm = TRUE)),
    ylim = c(min(map_info$Latitude - 1, na.rm = TRUE),
             max(map_info$Latitude + 1, na.rm = TRUE)))
# Now we plot our sample points on, colored by Pops
# when two points are too close, we slightly jitter them
jitter <- position_jitter(width = 0.2, height = 0.2)
p <- p + geom_point(data = map_info,
                    aes(x = Longitude, y= Latitude, fill = Pop),
```

```
                    size = 4, shape = 21, position = jitter)

# See the plot
p
```




```
# We can do another one colored by their languages
# zoom-in on the world map according to the sample geo-coordinates
p <- ggplot() +
  geom_map(data = map.world,
           map = map.world,
           aes(map_id = region),
           fill = "white", colour = "grey", size = 0.15) +
  coord_quickmap(
    xlim = c(min(map_info$Longitude - 2, na.rm = TRUE),
             max(map_info$Longitude + 2, na.rm = TRUE)),
    ylim = c(min(map_info$Latitude - 1, na.rm = TRUE),
             max(map_info$Latitude + 1, na.rm = TRUE)))
# Now we plot our sample points on, colored by Languages
# when two points are too close, we slightly jitter them
jitter <- position_jitter(width = 0.2, height = 0.2)
p <- p + geom_point(data = map_info,
                    aes(x = Longitude, y= Latitude, fill = Language),
                    size = 4, shape = 21, position = jitter)

# See the plot
p
```



Questions: Do you think the genetic population structure will align with the geographic pattern or the linguistic pattern? Are our studying Thai groups closer to the Taiwanese groups or the Indian groups?

Data Formats

plink format files are commonly used for human genome-wide data analysis, you can find the descriptions of .bed/.bim/.fam from their webpage. Here, we can just have a quick look of our data.

```
## CODE FOR LINUX BASH ##
# In linux terminals
less Study_pop.bim
less Study_pop.fam
```

```
# Read in the fam file
study_fam <- read_delim("Study_pop.fam", delim = " ", col_names = FALSE)
```

```
## Rows: 53 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: " "
## chr (2): X1, X2
## dbl (4): X3, X4, X5, X6
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
colnames(study_fam) <- c("FID", "IID", "FID_father", "FID_mather", "Sex", "Pheno")
head(study_fam)
```

```
## # A tibble: 6 x 6
##   FID   IID   FID_father FID_mather   Sex Pheno
##   <chr> <chr>      <dbl>      <dbl> <dbl> <dbl>
## 1 TN207 TN207          0          0     0     1
## 2 TL152 TL152          0          0     0     1
## 3 CT211 CT211          0          0     0     1
## 4 CT307 CT307          0          0     0     1
## 5 TN209 TN209          0          0     0     1
## 6 TL160 TL160          0          0     0     1
```

```
study_bim <- read_delim("Study_pop.bim", delim = "\t", col_names = FALSE)
```

```
## Rows: 616974 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: "\t"
## chr (3): X2, X5, X6
## dbl (3): X1, X3, X4
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
colnames(study_bim) <- c("CHR", "SNP", "PHY", "POS", "A1", "A2")
head(study_bim)
```

```
## # A tibble: 6 x 6
##     CHR SNP         PHY    POS A1    A2
##   <dbl> <chr>     <dbl>  <dbl> <chr> <chr>
## 1     1 1_565596 0.0062 565596 A     G
## 2     1 1_567137 0.0062 567137 0     C
## 3     1 1_752566 0.0083 752566 G     A
```

```
## 4     1 1_842013 0.0093 842013 G     T
## 5     1 1_891021 0.0099 891021 G     A
## 6     1 1_903426 0.01   903426 T     C
```

Questions: How many individuals in the studying population data? How many variants/SNPs? Why are there alleles coded as 0 instead of ACGT?

Data Merge

We can use plink –bmerge to merge plink files of Ref pops and Study pops. Before we doing that, we will need to find the common variants between the two datasets, so that there won't be certain variants totally missing in indviduals from one dataset or the other.

```
## CODE FOR LINUX BASH ##
# Find common SNPs
grep -wFf <(awk '{print $2}' Study_pop.bim) Ref_pop.bim | \
  awk '{print $2}' > common_snps.txt
# Merge the two dataset with the phenotype column coded as 1
# the individuals are ordered according to our metadata
plink --bfile Study_pop \
  --bmerge Ref_pop \
  --extract common_snps.txt \
  --indiv-sort f pop_info.txt \
  --output-missing-phenotype 1 \
  --make-bed \
  --allow-no-sex \
  --out Merged_Study_Ref
```

Data QC

We need to perform QC for both SNPs and individuals. Control for missing data is a common practice for both SNPs and individuals using plink –missing; at the SNP level, the within-group missing rate can be further checked. A strict Hardy-Weinberg equilibrium (HWE) test threshold is often used to filter out genotyping error SNPs using plink –hwe. Finally, high kinship between individuals can bias population structure results, so we will also deal with that using king implemented in plink2.

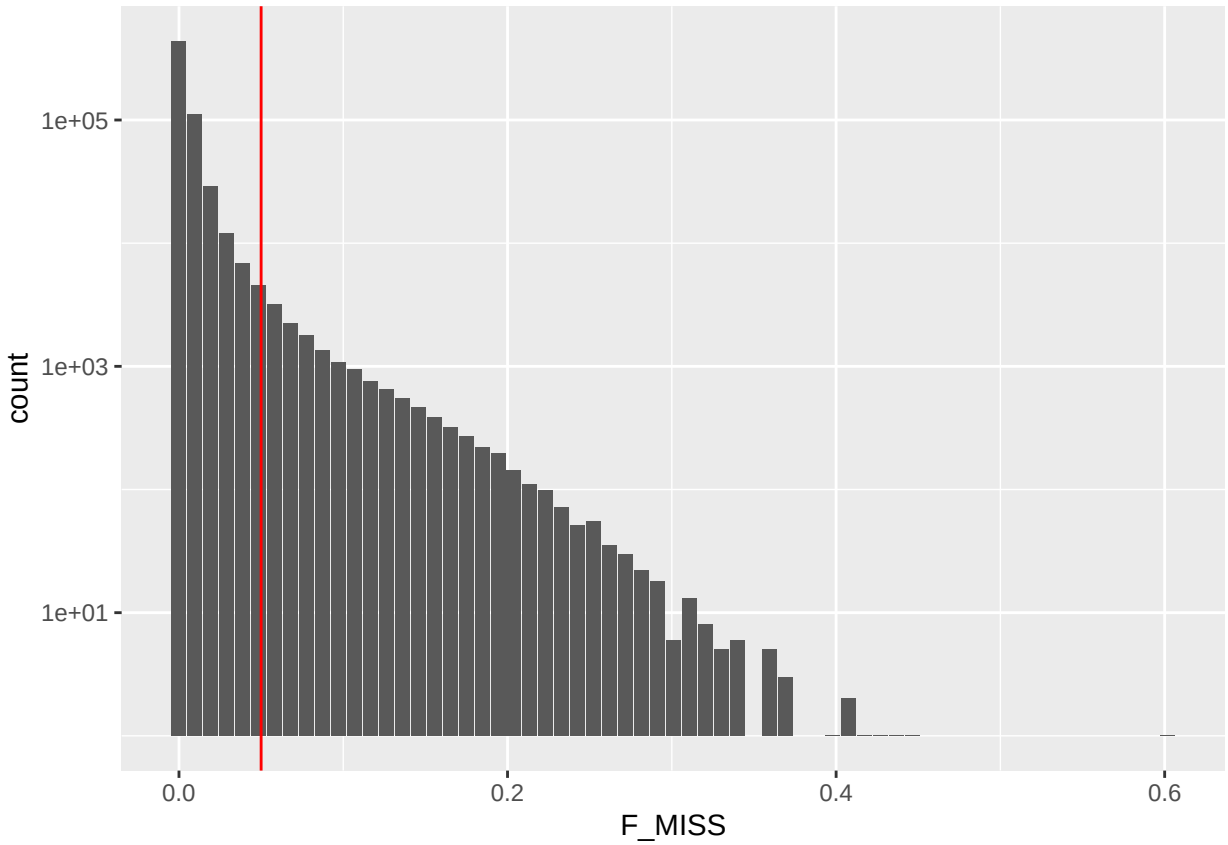
```
## CODE FOR LINUX BASH ##
# the global missing rate
plink --bfile Merged_Study_Ref --missing --out Merged_Study_Ref.missing
# it will output .imiss file for individual and .lmiss for SNP
less Merged_Study_Ref.missing.imiss
less Merged_Study_Ref.missing.lmiss
```

```
# We can have a look of them in R
# Starting with the lmiss file
lmiss <- read_table("Merged_Study_Ref.missing.lmiss", col_names = TRUE)
```

Missing data of all SNPs and individuals

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   CHR = col_double(),
##   SNP = col_character(),
##   N_MISS = col_double(),
##   N_GENO = col_double(),
##   F_MISS = col_double()
## )
```

```
# We can plot the distribution of fraction of missing data (F_MISS)
lmiss %>% ggplot() +
  geom_bar(aes(x = F_MISS)) +
  scale_y_continuous(trans = 'log10') +
  geom_vline(xintercept = 0.05, color = "red") # common threshold of 5% missing rate
```




```
# So, how many variants show more than 5% missing rate?
lmiss %>% filter(F_MISS > 0.05) %>% count()
```

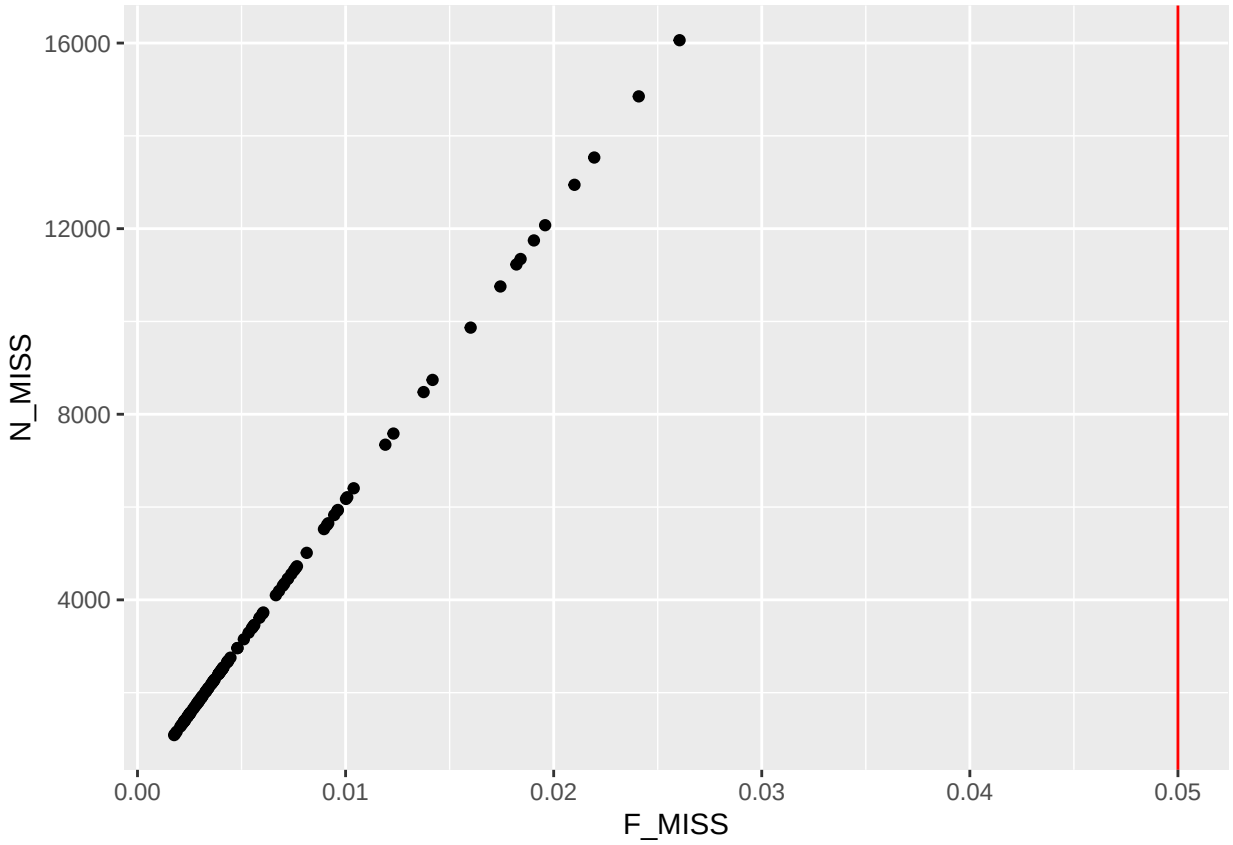
```
## # A tibble: 1 x 1
##       n
##   <int>
## 1 15046
```

```
# Same for the imiss file
imiss <- read_table("Merged_Study_Ref.missing.imiss", col_names = TRUE)
```

```
##
## -- Column specification --------------------------------------------------------
```

```
## cols(
##   FID = col_character(),
##   IID = col_character(),
##   MISS_PHENO = col_character(),
##   N_MISS = col_double(),
##   N_GENO = col_double(),
##   F_MISS = col_double()
## )
```

```
# We can plot the distribution of fraction of missing data (F_MISS)
imiss %>% ggplot() +
  geom_point(aes(x = F_MISS, y = N_MISS)) +
  geom_vline(xintercept = 0.05, color = "red") # common threshold of 5% missing rate
```




```
# So, how many variants show more than 5% missing rate?
imiss %>% filter(F_MISS > 0.05) %>% count()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1     0
```

```
## CODE FOR LINUX BASH ##
# Here, we get the SNP IDs and individual IDs that fail the 5% missing data threshold
cat Merged_Study_Ref.missing.lmiss | \
  awk '$5 > 0.05' | \
  awk '{print $2}' | \
  sort -u | \
  grep -v SNP > Merged_Study_Ref.missing.lmiss.0.05
```

```
cat Merged_Study_Ref.missing.imiss | \
  awk '$6 > 0.05' | \
  awk '{print $1"\t"$2}' | \
  grep -v IID > Merged_Study_Ref.missing.imiss.0.05
```

Question: How many SNPs and individuals fail the 5% missing data threshold?

```
## CODE FOR LINUX BASH ##
# We usually further filter the within-poulation missing data
# for SNPs that might be totally missing in some populations
# This can be done by including the --within option with our metadata
# the metadata 3rd column should be population label
plink --bfile Merged_Study_Ref \
  --missing \
  --within pop_info.txt \
  --out Merged_Study_Ref.within.missing
# We will filter out SNPs >50% missing in one populations
# And the population has at least more than one individual in our data
cat Merged_Study_Ref.within.missing.lmiss | \
  awk '$7 > 0.5 && $6 > 1' | \
  awk '{print $2}' | \
  sort -u | \
  grep -v SNP > Merged_Study_Ref.within.missing.lmiss.0.5
# Similarly, we do the group-based filtering for HWE with a wrapper script I wrote
./run_hardy_withinPop.sh pop_info.txt Merged_Study_Ref
# We will filter out SNPs show HWE p value smaller than 1e-05
cat Merged_Study_Ref.within.hwe | \
  awk '$10 < 1e-05' | \
  awk '{print $3}' | \
  sort -u | \
  grep -v SNP > Merged_Study_Ref.within.hwe.1e-05
```

Missing data and HWE test of SNPs per group

```
## CODE FOR LINUX BASH ##
# Now, at the individual level, we will check for kinship using plink2
~/bin/plink2 --bfile Merged_Study_Ref \
  --make-king triangle bin \
  --make-king-table \
  --out Merged_Study_Ref.kin
# This will output a binary kinship matrix and a kinship table
```

```
# We can read the kinship table in to have a look
kin_table <- read_table("Merged_Study_Ref.kin.kin0", col_names = TRUE) %>%
  rename(FID1 = "#FID1")
```

Kinship of individuals

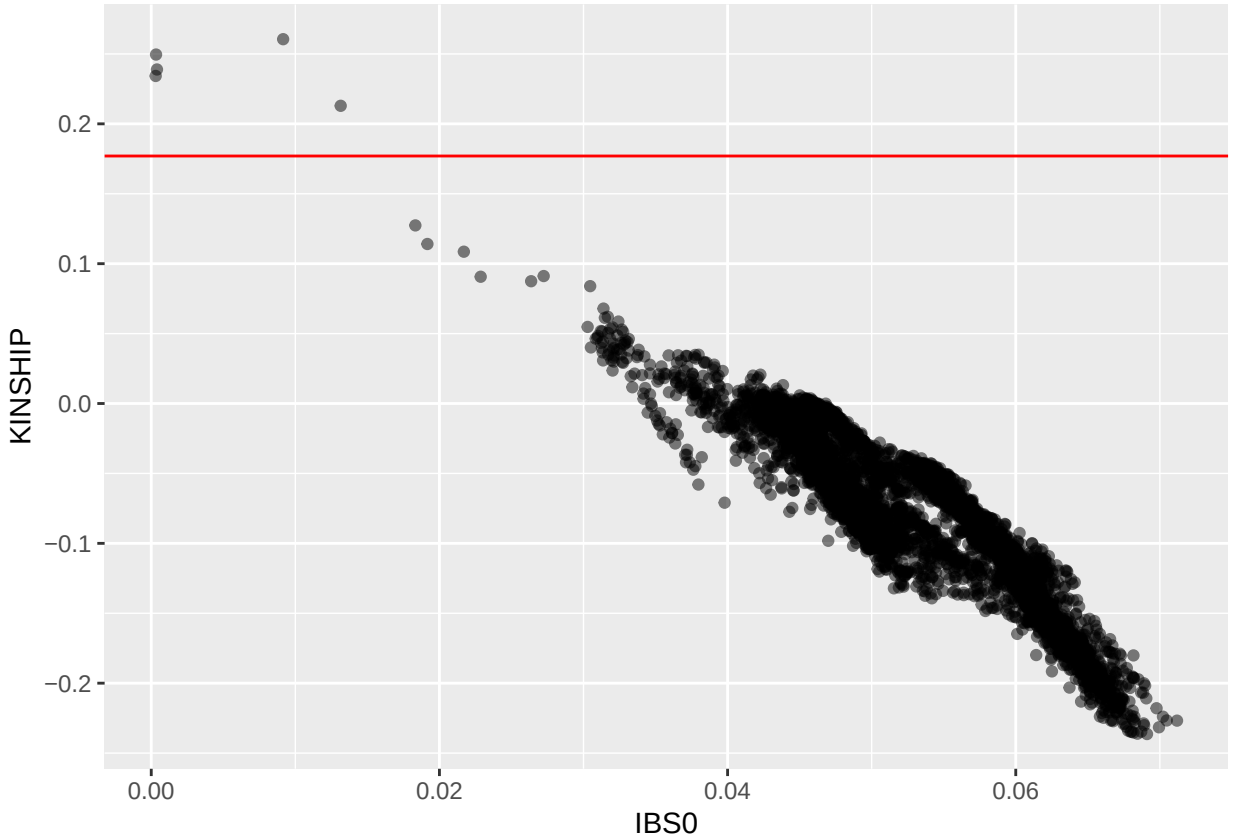
```
##
## -- Column specification --------------------------------------------------------
## cols(
```

```
##   `#FID1` = col_character(),
##   ID1 = col_character(),
##   FID2 = col_character(),
##   ID2 = col_character(),
##   NSNP = col_double(),
##   HETHET = col_double(),
##   IBS0 = col_double(),
##   KINSHIP = col_double()
## )
```

```
head(kin_table)
```

```
## # A tibble: 6 x 8
##   FID1  ID1   FID2  ID2     NSNP HETHET   IBS0  KINSHIP
##   <chr> <chr> <chr> <chr>  <dbl>  <dbl>  <dbl>    <dbl>
## 1 TN207 TN207 TN201 TN201 608738 0.0776 0.0400 -0.0145
## 2 TN209 TN209 TN201 TN201 609883 0.0777 0.0409 -0.0150
## 3 TN209 TN209 TN207 TN207 610859 0.0792 0.0402 -0.00670
## 4 TN213 TN213 TN201 TN201 601520 0.0912 0.0229  0.0907
## 5 TN213 TN213 TN207 TN207 602248 0.0852 0.0352  0.0208
## 6 TN213 TN213 TN209 TN209 603263 0.0835 0.0395 -0.00835
```

```
# We can try to plot it
kin_table %>% ggplot() +
  geom_point(aes(x = IBS0, y = KINSHIP), alpha = 0.5) +
  geom_hline(yintercept = 0.177, color = "red") # cutoff for up to 1st degree kinship
```



Question: How many more individuals might need to be removed if we set our kinship cutoff to up to 2nd degree kinship? (hint: common cutoff for 2nd degree kinship is ~0.0884)

```
## CODE FOR LINUX BASH ##
# we can now filter all the SNPs and individuals failing the QC steps
# For SNPs
cat Merged_Study_Ref.missing.lmiss.0.05 \
  Merged_Study_Ref.within.missing.lmiss.0.5 \
  Merged_Study_Ref.within.hwe.1e-05 | \
  sort -u > Merged_Study_Ref.exclude.snp
# For individuals
cat Merged_Study_Ref.missing.imiss.0.05 \
  Merged_Study_Ref.kin.king.cutoff.out.id | \
  grep -v IID | \
  sort -u > Merged_Study_Ref.remove.ind
# Use plink to do the filtering job
plink --bfile Merged_Study_Ref \
  --remove Merged_Study_Ref.remove.ind \
  --exclude Merged_Study_Ref.exclude.snp \
  --make-bed \
  --out QC_Study_Ref
# This will be the pass-QC data you will you for all the rest analysis of your project!
# Although I have done that,
# but this will usually be the time for one to update the QC column in the metadata
```

Population Structure

To explore population structure, one usually starts with unsupervised approaches (i.e., the method does not have any a prori knowledge of your samples) to see how the samples relate to each other. PCA and ADMIXTURE are commonly used nowadays in probably *every* human genetics study. It is important to know the assumptions of the tool you use and what exactly its result tell us. PCA and ADMIXTURE, especially the latter, assume each variant in the model is independent from the others; however, in our genome, many variants are in linkage-disequilibrium (LD) with others, which means they are non-independently inherited. So, before doing PCA and ADMIXTURE, we will do LD-pruning for our data using plink. Another important thing to keep in mind is that the population structure revealed PCA and ADMIXTURE can be explained through tons of evolutionary scenario. Unsupervised approaches to explore population structure is useful for understanding the data variation and generating hypotheses, but one should always avoid over-interpretation of such results.

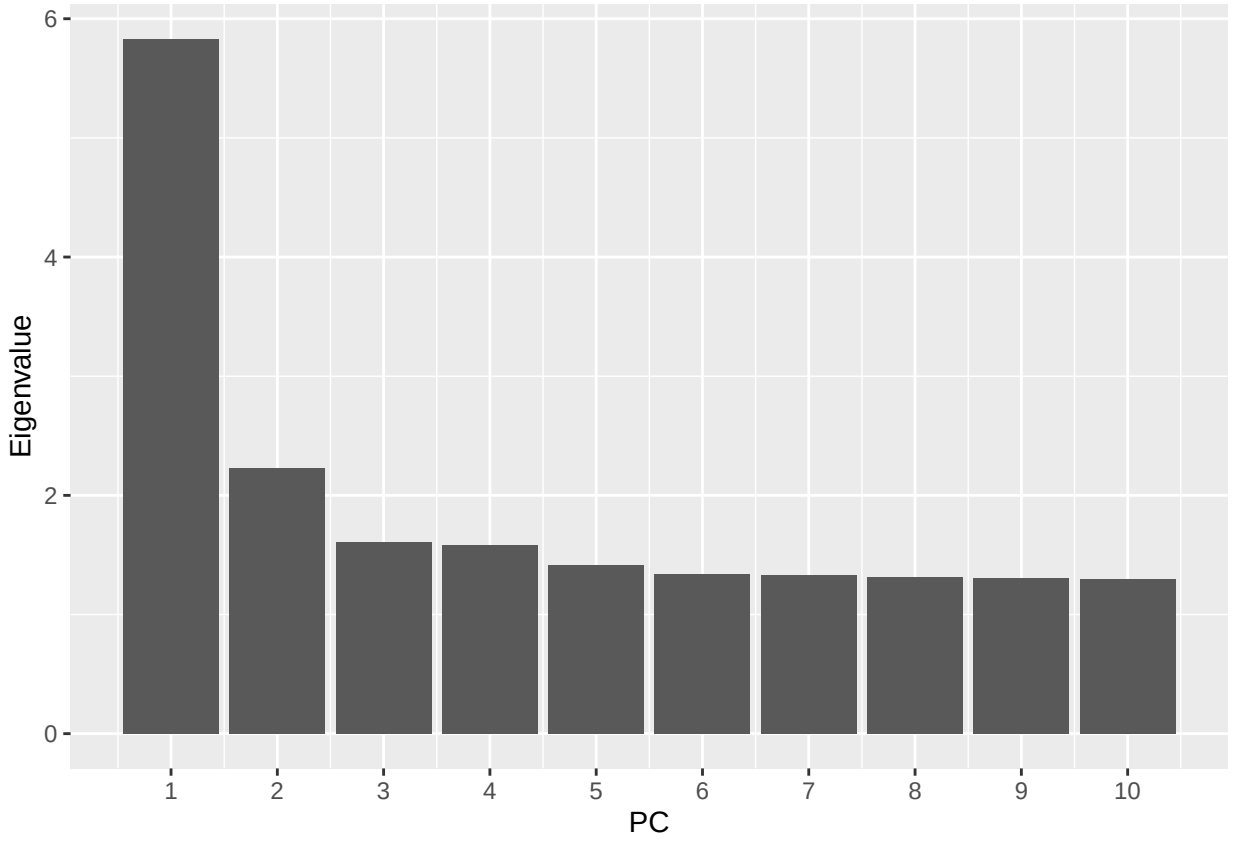
```
## CODE FOR LINUX BASH ##
# We prune our data in
# window size of 200kb, step size of 25 variants, and r^2 threshold is 0.4
# This are the parameters have been shown working well with our genotyping array data
plink --bfile QC_Study_Ref --indep-pairwise 200 25 0.4 --out QC_Study_Ref.LD
plink --bfile QC_Study_Ref \
  --extract QC_Study_Ref.LD.prune.in \
  --allow-no-sex \
  --make-bed \
  --out Pruned_QC_Study_Ref
```

LD pruning

PCA PCA is a method to decompose the variation of data into few linear dimensions. There are many tools to run PCA. In principle, all of them should give similar overall patterns. Depending on the data and questions, some might have certian advantages over the others. Here, we will just use the –pca implemented in plink, which is fast in computation time and easy to use.

```
## CODE FOR LINUX BASH ##
# run pca to PC10
plink --bfile Pruned_QC_Study_Ref --pca 10 header tabs --out Pruned_QC_Study_Ref.pca
```

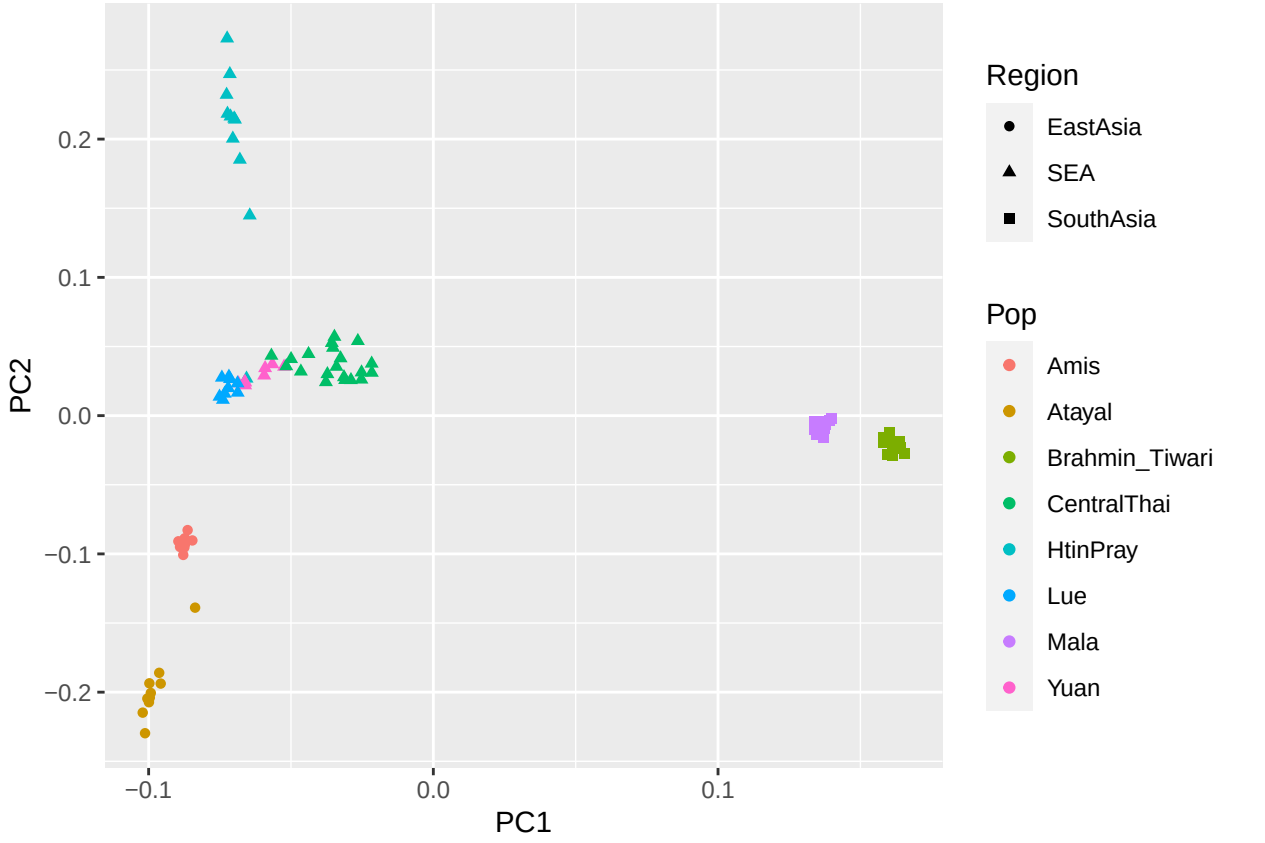
```
# Let's now look at the results in R
# start with the eigenvalues
# which gives us the idea of how much variation is explained by this PC
pc_eigenvalues <- read_delim("Pruned_QC_Study_Ref.pca.eigenval",
                             delim = "\t", col_names = FALSE) %>%
  mutate(PC=row_number()) %>% rename(Eigenvalue="X1")

# plot it
pc_eigenvalues %>% ggplot(aes(x = PC, y = Eigenvalue)) +
  geom_histogram(stat = "identity") +
  scale_x_continuous(breaks = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10))
```



Question: Which PC explains the most genetic variation of the data?

```
# let's move to eigenvectors
# which are the linear decomposed coordinates that separate individuals on different PCs
# we will need the information from metadata here to help with visualization
# let's just read it in again and filter out individuals failing QC
info <- read_delim("pop_info.txt", col_names = TRUE, delim = "\t") %>%
  filter(QC == "PASS")
```

```
# read the eigenvectors
pc_eigenvectors <- read_delim("Pruned_QC_Study_Ref.pca.eigenvec",
                              col_names = TRUE, delim = "\t")
# join the info and pc for plotting
info %>% left_join(pc_eigenvectors) %>% ggplot() +
  geom_point(aes(x = PC1, y = PC2, color = Pop, shape = Region))
```



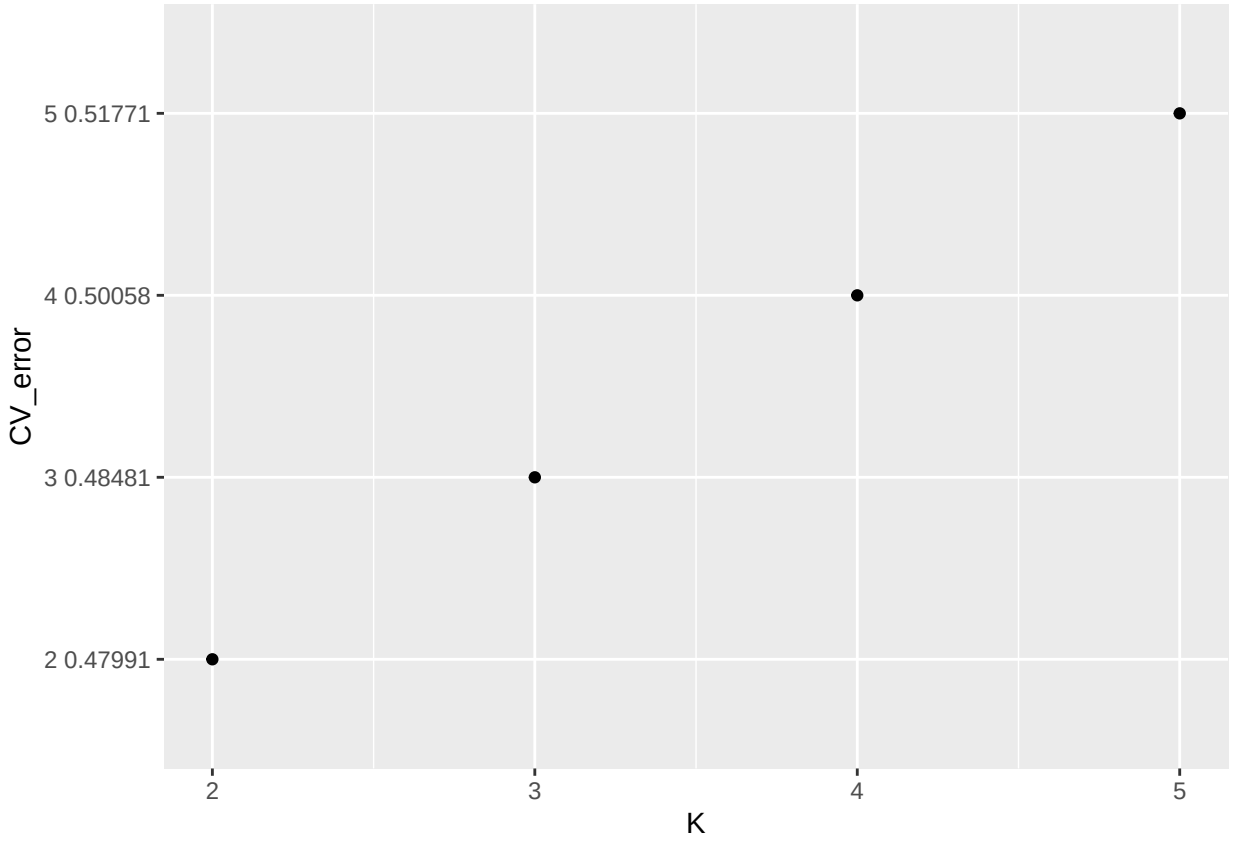
Questions: How does it look if you color by languages? So, can we now answer our previous question that whether geography or language aligns with the genetic structure? Are Thai groups closer to Taiwanese groups or Indian groups? Why along PC1, some Thai groups seem closer to South Asian/Indian groups? What does the separation along PC2 indicate?

ADMIXTURE ADMIXTURE is a clustering algorithm to estimate the proportions of different K components in each indvidual given a specified number of K. One can run different number of K and record the cross-validation errors for each run to see which number of K best fits the data variation. This usually requires multiple independent runs of each K to get a sense of the distributions and consensus as the convergent results for each run might be different. For the sake of time, we will just run it once for eack K from K=2 to K=5, which will already take 10-20 minutes.

```
## CODE FOR LINUX BASH ##
# we make another folder for this
mkdir admixture
cd admixture
# use a for loop to run from K=2 to K=5
for i in {2..5}; do admixture --cv ../Pruned_QC_Study_Ref.bed $i > log${i}.out; done
# extract the cross-validation error of each K from the log files
awk '/CV/ {print $3,$4}' *out | cut -c 4,7-20 > Pruned_QC_Study_Ref.cv.error
```

```
# to plot the results, we will also want to know which individual from which populations
# the order of ind-pop list file should be the same order as
# the plink fam file of our ADMIXTURE input
# you will find this file in the folder named ind.pop.list
# here is a command to make this file
cat pop_info.txt | grep PASS | grep -v IID | awk '{print $2"\t"$3}' > ind.pop.list
```

```
# let's now try to visualize the results in R
# first, we have a look of cross-validation errors
cv <- read_delim("admixture/Pruned_QC_Study_Ref.cv.error",
                 delim = "\t", col_names = FALSE) %>%
  mutate(K=2:5) %>%
  rename(CV_error="X1")
cv %>% ggplot() + geom_point(aes(x = K, y = CV_error))
```



Question: Based on the current result, which number of K best fits our data variation?

Now, we will use a script that I adopted from here to plot the results

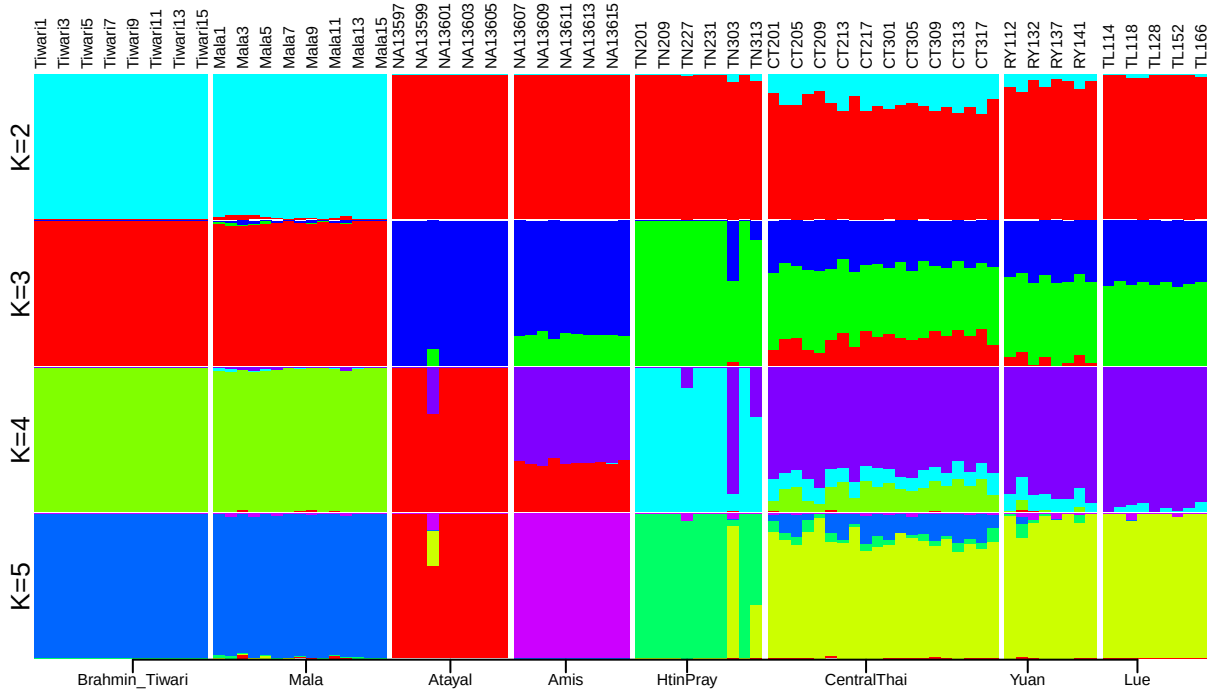
```
# This script will plot the results of K=2 to K=5 based on the ADMIXTURE output Q files
# The individual and population labels come from the ind.pop.list
# You will also need to specify the order of populations for the plotting
# Assign the first argument to prefix
prefix = "admixture/Pruned_QC_Study_Ref"

# Get individual names in the correct order
labels <- read.table("admixture/ind.pop.list")
```

```
# Population order
populations_order="Brahmin_Tiwari,Mala,Atayal,Amis,HtinPray,CentralThai,Yuan,Lue"

# Name the columns
names(labels) <- c("ind","pop")

# Add a column with population indices to order the barplots
# Use the order of populations specified at the beginning (list separated by commas)
labels$n <- factor(labels$pop, levels = unlist(strsplit(populations_order, ",")))
levels(labels$n) <- c(1:length(levels(labels$n)))
labels$n <- as.integer(as.character(labels$n))

# read in the different admixture output files
minK = 2
maxK = 5
tbl <- lapply(minK:maxK, function(x) read.table(paste0(prefix, ".", x, ".Q")))

# Prepare spaces to separate the populations/species
rep <- as.vector(table(labels$n))
spaces <- 0
for(i in 1:length(rep)){spaces = c(spaces, rep(0, rep[i]-1), 0.5)}
spaces <- spaces[-length(spaces)]

# Plot the cluster assignments as a single bar for each individual
# and for each K as a separate row
par(mfrow = c(maxK-1,1),
    mar = c(0,1,0,0),
    oma = c(2,1,9,1),
    mgp = c(0,0.2,0),
    xaxs = "i",
    cex.lab = 1.2,
    cex.axis = 0.8)
# Plot minK
bp <- barplot(t(as.matrix(tbl[[1]][order(labels$n),])),
              col = rainbow(n=minK),
              xaxt = "n",
              border = NA,
              ylab = paste0("K=", minK),
              yaxt = "n",
              space = spaces)
axis(3,
     at = bp,
     labels = labels$ind[order(labels$n)],
     las = 2,
     tick = F,
     cex = 0.6)
# Plot higher K values
if(maxK > minK)lapply(2:(maxK - 1), function(x)
  barplot(t(as.matrix(tbl[[x]][order(labels$n),])),
          col = rainbow(n=x+1),
          xaxt = "n",
          border = NA,
          ylab = paste0("K=",x+1),
```

```
          yaxt = "n",
          space = spaces))
```

```
## [[1]]
##  [1]   0.5   1.5   2.5   3.5   4.5   5.5   6.5   7.5   8.5   9.5  10.5  11.5
## [13]  12.5  13.5  14.5  16.0  17.0  18.0  19.0  20.0  21.0  22.0  23.0  24.0
## [25]  25.0  26.0  27.0  28.0  29.0  30.0  31.5  32.5  33.5  34.5  35.5  36.5
## [37]  37.5  38.5  39.5  40.5  42.0  43.0  44.0  45.0  46.0  47.0  48.0  49.0
## [49]  50.0  51.0  52.5  53.5  54.5  55.5  56.5  57.5  58.5  59.5  60.5  61.5
## [61]  62.5  64.0  65.0  66.0  67.0  68.0  69.0  70.0  71.0  72.0  73.0  74.0
## [73]  75.0  76.0  77.0  78.0  79.0  80.0  81.0  82.0  83.0  84.5  85.5  86.5
## [85]  87.5  88.5  89.5  90.5  91.5  93.0  94.0  95.0  96.0  97.0  98.0  99.0
## [97] 100.0 101.0
##
## [[2]]
##  [1]   0.5   1.5   2.5   3.5   4.5   5.5   6.5   7.5   8.5   9.5  10.5  11.5
## [13]  12.5  13.5  14.5  16.0  17.0  18.0  19.0  20.0  21.0  22.0  23.0  24.0
## [25]  25.0  26.0  27.0  28.0  29.0  30.0  31.5  32.5  33.5  34.5  35.5  36.5
## [37]  37.5  38.5  39.5  40.5  42.0  43.0  44.0  45.0  46.0  47.0  48.0  49.0
## [49]  50.0  51.0  52.5  53.5  54.5  55.5  56.5  57.5  58.5  59.5  60.5  61.5
## [61]  62.5  64.0  65.0  66.0  67.0  68.0  69.0  70.0  71.0  72.0  73.0  74.0
## [73]  75.0  76.0  77.0  78.0  79.0  80.0  81.0  82.0  83.0  84.5  85.5  86.5
## [85]  87.5  88.5  89.5  90.5  91.5  93.0  94.0  95.0  96.0  97.0  98.0  99.0
## [97] 100.0 101.0
##
## [[3]]
##  [1]   0.5   1.5   2.5   3.5   4.5   5.5   6.5   7.5   8.5   9.5  10.5  11.5
## [13]  12.5  13.5  14.5  16.0  17.0  18.0  19.0  20.0  21.0  22.0  23.0  24.0
## [25]  25.0  26.0  27.0  28.0  29.0  30.0  31.5  32.5  33.5  34.5  35.5  36.5
## [37]  37.5  38.5  39.5  40.5  42.0  43.0  44.0  45.0  46.0  47.0  48.0  49.0
## [49]  50.0  51.0  52.5  53.5  54.5  55.5  56.5  57.5  58.5  59.5  60.5  61.5
## [61]  62.5  64.0  65.0  66.0  67.0  68.0  69.0  70.0  71.0  72.0  73.0  74.0
## [73]  75.0  76.0  77.0  78.0  79.0  80.0  81.0  82.0  83.0  84.5  85.5  86.5
## [85]  87.5  88.5  89.5  90.5  91.5  93.0  94.0  95.0  96.0  97.0  98.0  99.0
## [97] 100.0 101.0
```

```
  axis(1,
       at = c(which(spaces == 0.5), bp[length(bp)]) -
         diff(c(1,which(spaces == 0.5), bp[length(bp)]))/2,
       labels = unlist(strsplit(populations_order,",")))
```



Questions: At K=2, the South Asians/Indians are sparated from other East Asians by standing out in the light blue component while some Thai populations also share this component, does this agree with our PCA result? Which population further got their own color at K=3, and what does it imply? What about K=4 and K=5?

Conclusion

Congratulations on making it to the end! I hope you have learned the basic steps that you could do from receiving a human genome-wide data to visualize the population structure. From the revealed structure, one can make good observations and formulate some interesting hypotheses. For example, this pattern of CetralThai share more affinity to South Asian/Indian populations might suggest admixture, which is indeed one of the main findings of our previous paper! But as mentioned before, one will need to further investigate and test the hypotheses before over-interpreting the results. You can have a look in Kutanan, Liu et al 2021 how we further tested the hypotheses of admixture. For example, F-statistics from AdmixTools2 is commonly used for testing admixture. I hope you enjoy this exercise and feel free to contact me (dang.liu@pasteur.fr) if you have any questions!