BINF 6110 – Project 4

Name: Abelhard Jauwena

Student ID: 1040413

Topic: Performing Discriminant Analysis of Principal Components

Introduction

The Purpose and Goal of My Analysis

This tutorial describes how to perform "Discriminant Analysis of Principal Components" (DAPC) in R to identify genetic groups in a data set of Ethiopian barley (*Hordeum vulgare* L.) landrace genotypes provided by Dido et al. (2022). Dido et al. (2021) originally performed this analysis as part of their aim, which was to investigate spatial and temporal genetic variation in Ethiopian barley landraces based on simple sequence repeat (SSR) markers. Unfortunately, they did not specify the steps they took in their analysis.

At the end of this tutorial, I want to see whether I can arrive at the same conclusions Dido et al. (2021) did. Before we proceed, you can download the data set that we will analyze by visiting this link: **https://datadryad.org/stash/dataset/doi:10.5061%2Fdryad.4tmpg4f8v** (Dido et al., 2022). The data set should be in ".xlsx" format.

Applications of DAPC

In recent years, DAPC has become the method of choice for inferring the genetic structure of a population and identifying the alleles that drive phenotypic differentiation in that population (Dido et al., 2021; Miller et al., 2020). The main reason for its popularity lies in the fact that it confers the same benefits as both principal component analysis (PCA) and discriminant analysis (DA) but possesses very few of their drawbacks (Jombart et al., 2010). As such, DAPC has been extensively applied in various genetic analyses. For example, it has been used to assess the genetic diversity and structure of a potato (*Solanum tuberosum*) population using SNP markers (Deperi et al., 2018). It was also used to analyze genetic diversity, infer the population structure, and detect patterns of linkage disequilibrium in sweet cherry (*Prunus avium*) to create new cultivars that are more resistant to various environmental challenges

(Campoy et al., 2016). Lastly, it was used in tandem with the software STRUCTURE to evaluate genetic diversity and population structure in watermelons (*Citrullus lanatus*) using SNP data obtained via genotyping by sequencing (GBS) (Lee et al., 2019).

Tutorial

We will conduct all our analyses entirely in RStudio. If you have not downloaded RStudio, you can do so by visiting this link: **https://posit.co/download/rstudio-desktop/** (*RStudio Desktop*, n.d.).

Preparing R

We begin by first loading the appropriate packages.

```r
library(adegenet) # (Jombart, 2008; Jombart & Ahmed, 2011).
library(dplyr) # (Wickham et al., 2023).
library(janitor) # (Firke, 2023).
library(openxlsx) # (Schauberger & Walker, 2023).
library(tidyverse) # (Wickham et al., 2019).
```

If you do not have these packages installed, you can do so by running the following code.

```r
install.packages(‘adegenet’)
install.packages(‘dplyr’)
install.packages(‘janitor’)
install.packages(‘openxlsx’)
install.packages(‘tidyverse’)
```

Next, make sure that you set your working directory in the directory that contains the data set, as doing so will make reading in the data set easier. To do this, click “Session” in the top navigation bar, hover over “Set Working Directory,” click “Choose Directory,” and select the appropriate directory.

Processing Data

Read in the data set as a data frame using the function `read.xlsx`.

```r
```

```
df_data <- read.xlsx(xlsxFile = 'ethiopian_barley_data_set.xlsx', fillMergedCells = T
RUE, colNames = TRUE)
```

In my case, I have named my data set “ethiopian_barley_data_set.xlsx,” but you can name yours any way you want. Just make sure to pass in the appropriate name in the argument `xlsxFile`.

Set the data in the first row as column names and delete the last two columns (i.e., the columns “Loci” and “Code”), which contain loci annotations.

```R
df_data <- row_to_names(df_data, row_number = 1) %>% # (zek19, 2019).
  subset(., select = -c(Loci, Code))
```

Subset the data frame such that it only contains data for account numbers, regions of origin, altitude classes, collection years, and SSR markers (Dido et al., 2021).

```R
df_data <- df_data[, c('Acc.No', 'Region', 'Altitude', 'Year collected', 'M1', 'M2',
'M3', 'M4', 'M5', 'M6', 'M7', 'M8', 'M9', 'M10', 'M11', 'M12', 'M13', 'M14', 'M15', '
M16', 'M17', 'M18', 'M19', 'M20', 'M21', 'M22', 'M23', 'M24', 'M25', 'M26', 'M27', 'M
28', 'M29', 'M30', 'M31', 'M32', 'M33', 'M34', 'M35', 'M36', 'M37', 'M38', 'M39', 'M4
0', 'M41', 'M42', 'M43', 'M44', 'M45', 'M46', 'M47', 'M48', 'M49')]
```

Convert the data in the column “Year collected” into factors to allow for sorting downstream.

```R
df_data$`Year collected` <- as.factor(df_data$`Year collected`)
```

Create a new data frame sorted by the collection years in the column “Year collected” in ascending order. Then, omit any NAs from the data frame to make downstream analyses easier.

```R
df_data_sorted <- df_data[order(df_data$`Year collected`), ] %>%
  na.omit()
```

Creating Data Frames for Each Collection Year

To find the most probable number of clusters in the data set, Dido et al. (2021) used the function `find.clusters` for each collection year. Since `find.clusters` only accepts `data.frame`, `matrix`, `genind`, or `genlight` objects as inputs, we need to create separate data frames for each collection year. Moreover, each data frame must only contain data for the SSR markers in `numeric` format, as `find.clusters` only accepts numeric data as input.

To do this, we first create a list containing data frame subsets as elements, where each data frame subset corresponds to all entries for a collection year.

R

```
list_df_subsets <- split(df_data_sorted, df_data_sorted$`Year collected`)
```

Then, create an empty vector that will contain the names of the data frames for each collection year.

R

```
vec_df_names <- c()
```

Populate the empty vector by looping through all the names in the list of data frame subsets and adding each name to the empty vector.

R

```
for (i in names(list_df_subsets)) {
  vec_df_names <- append(vec_df_names, i)
{
```

Add the prefix “df_” for each name in the vector for organizational purposes. Because we will assign each name to its corresponding data frame subset, the prefix “df_” tells us that the object the name is assigned to is a `data.frame` object.

R

```
vec_df_names <- paste0('df_', vec_df_names, sep = '')
```

Now, we want to filter each data frame subset in our list so that they can be used as inputs to the function `find.clusters`. Specifically, we want to:

1. Delete the columns that we will not use in our analyses (i.e., the columns containing data for account numbers, regions of origin, altitude classes, and collection years).
2. Convert the data for the SSR markers into `numeric` format.

We will do all of this by defining our own function.

R

```
# Define a function that takes a data frame subset as input and filters it.
filter_df_subsets <- function(df_subset) { # (L, 2015).
  # Set the data in the column “Acc.No” as the row names for identification purposes.
  row.names(df_subset) <- df_subset[, 1]
  # Delete the columns “Acc.No,” “Region,” “Altitude,” and “Year collected.”
  df_subset <- df_subset[, -c(1, 2, 3, 4)]
  # Loop through the remaining columns, which contain data for the SSR markers.
  for (i in 1:ncol(df_subset)) {
    # Convert the data in each column into “numeric” format.
    df_subset[, i] <- as.numeric(df_subset[, i])
  }
  # Return the filtered data frame subset.
  return(df_subset)
}
```

Create a new list containing filtered data frame subsets as elements. We do this by applying our function that we just defined on each data frame subset in our original list.

```
R
list_df_subsets_filtered <- lapply(list_df_subsets, function(df_subset) filter_df_sub
sets(df_subset)) # (mpalanco, 2015).
```

Loop through each filtered data frame in our new list and assign an appropriate name to it according to its collection year.

```
R
for (i in 1:length(list_df_subsets_filtered)) {
  assign(vec_df_names[i], list_df_subsets_filtered[[i]])
}
```

Finding the Most Probable Number of Clusters for Each Collection Year

Find the most probable number of clusters in the data set by running the function `find.clusters` for each data frame subset. Unfortunately, `find.clusters` only works on data frames with a substantial number of entries (i.e., rows), so I can only run it on the data frame subsets for 1979 and 1986.

```
R
grp_1979 <- find.clusters(df_1979, n.pca = 200, method = 'kmeans', stat = 'BIC',  max
.n.clust = 40)
grp_1986 <- find.clusters(df_1986, n.pca = 200, method = 'kmeans', stat = 'BIC',  max
.n.clust = 40)
```

In the lines of code above, I explicitly told the function to retain up to 200 principal components (PCs). As a result, I retain all PCs in each data frame, which are around 50 in total. The reason for doing so is that there is no point in keeping a small number of PCs for `find.clusters` aside from time constraints (Jombart & Collins, 2015). Additionally, I also specified the function to find clusters using the "*k*-means" method, compute the Bayesian Information Criterion (BIC) for each number of clusters, and find up to 40 clusters.

When run, `find.clusters` will interactively ask the user to specify the number of retained PCs and the desired number of clusters (*k*). However, the user can bypass the interactive session entirely by explicitly providing the arguments `n.pca` and `n.clust`, respectively (Jombart & Collins, 2015). Since I provided the argument `n.pca` but not `n.clust`, I have to choose the number of clusters through the interactive session.

Running the two lines of code above outputs the following graphs, which correspond to the years 1979 and 1968, respectively:

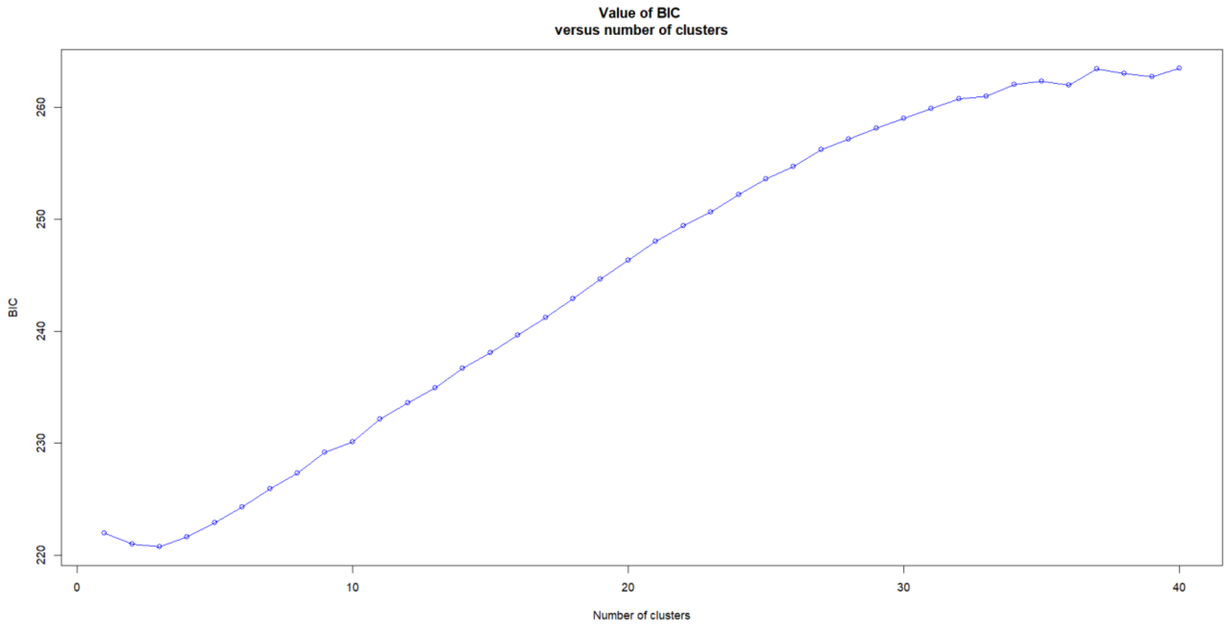


Fig. 1. The BIC values for each number of clusters in the Ethiopian barley landrace data from 1979.

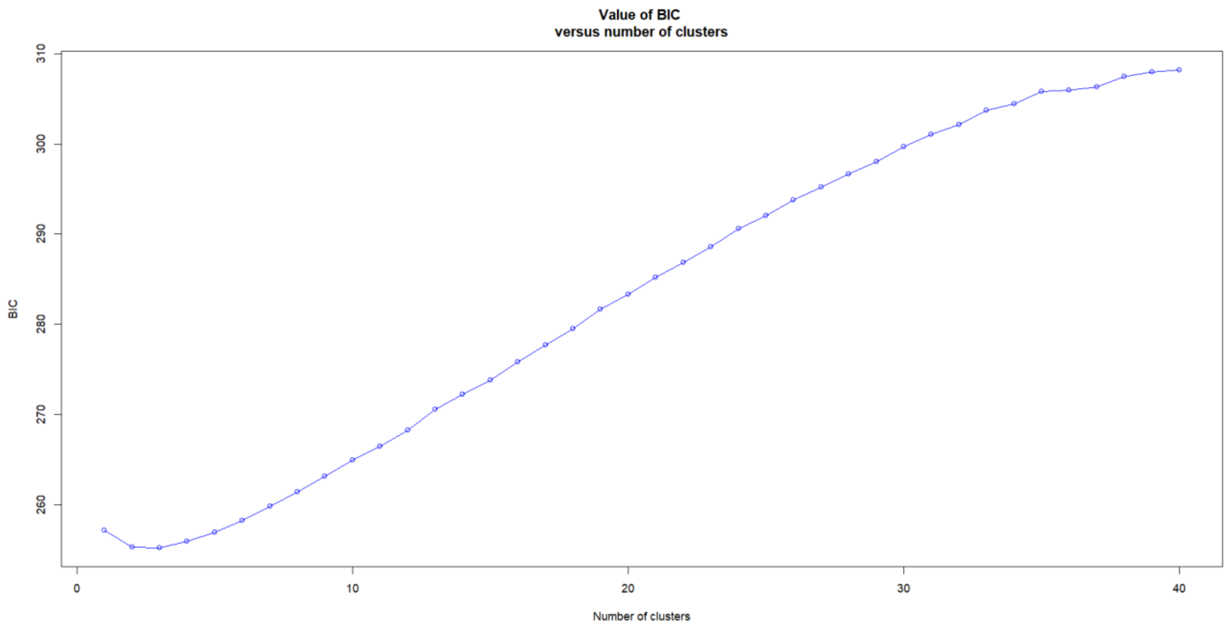


Fig. 2. The BIC values for each number of clusters in the Ethiopian barley landrace data from 1986.

During the interaction session, we should pick the value of *k* that corresponds to the lowest BIC value (Jombart & Collins, 2015). As such, I chose *k* = 3 for both years, as it corresponds to the lowest BIC values.

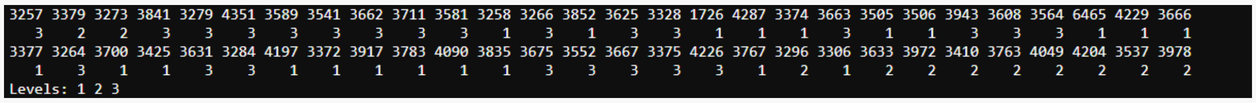
Performing DAPC for Each Collection Year

The function `find.clusters` returns a list object with four components. We are only interested in the third component, `grp`, which corresponds to the group memberships assigned to each data point.

We can view the group memberships for each year to get a sense of their distributions.

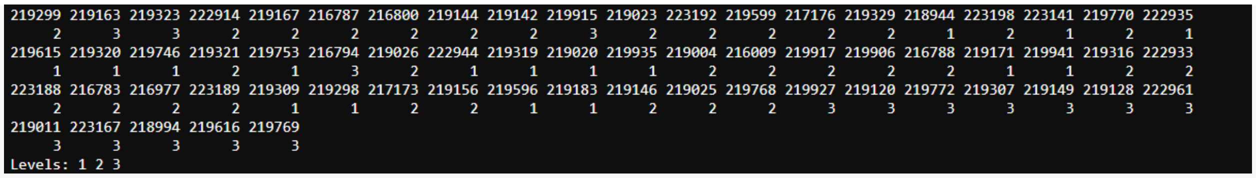
R

```
grp_1979$grp
```



```
3257 3379 3273 3841 3279 4351 3589 3541 3662 3711 3581 3258 3266 3852 3625 3328 1726 4287 3374 3663 3505 3506 3943 3608 3564 6465 4229 3666
   3    2    2    3    3    3    3    3    3    3    3    1    3    1    3    3    1    1    1    3    1    1    3    3    3    1    1    1
3377 3264 3700 3425 3631 3284 4197 3372 3917 3783 4090 3835 3675 3552 3667 3375 4226 3767 3296 3306 3633 3972 3410 3763 4049 4204 3537 3978
   1    3    1    1    3    3    1    1    1    1    1    1    3    3    3    3    3    1    2    1    2    2    2    2    2    2    2    2
Levels: 1 2 3
```

```
grp_1986$grp
```



```
219299 219163 219323 222914 219167 216787 216800 219144 219142 219915 219023 223192 219599 217176 219329 218944 223198 223141 219770 222935
     2      3      3      2      2      2      2      2      2      3      2      2      2      2      2      1      2      1      2      1
219615 219320 219746 219321 219753 216794 219026 222944 219319 219020 219935 219004 216009 219917 219906 216788 219171 219941 219316 222933
     1      1      1      2      1      3      2      1      1      1      1      2      2      2      2      2      1      1      2      2
223188 216783 216977 223189 219309 219298 217173 219156 219596 219183 219146 219025 219768 219927 219120 219772 219307 219149 219128 222961
     2      2      2      2      1      1      2      2      1      1      2      2      2      3      3      3      3      3      3      3
219011 223167 218994 219616 219769
     3      3      3      3      3
Levels: 1 2 3
```

Using the corresponding group memberships identified above, perform DAPC on the data frame subsets for 1979 and 1986.

```R
dapc_1979 <- dapc(df_1979, grp_1979$grp)
dapc_1986 <- dapc(df_1986, grp_1986$grp)
```

Like `find.clusters`, the function `dapc` also presents the user with an interactive session. The first phase of the interactive session asks the user to choose the number of PCs to retain while outputting the following graphs (one for 1979 and 1986):

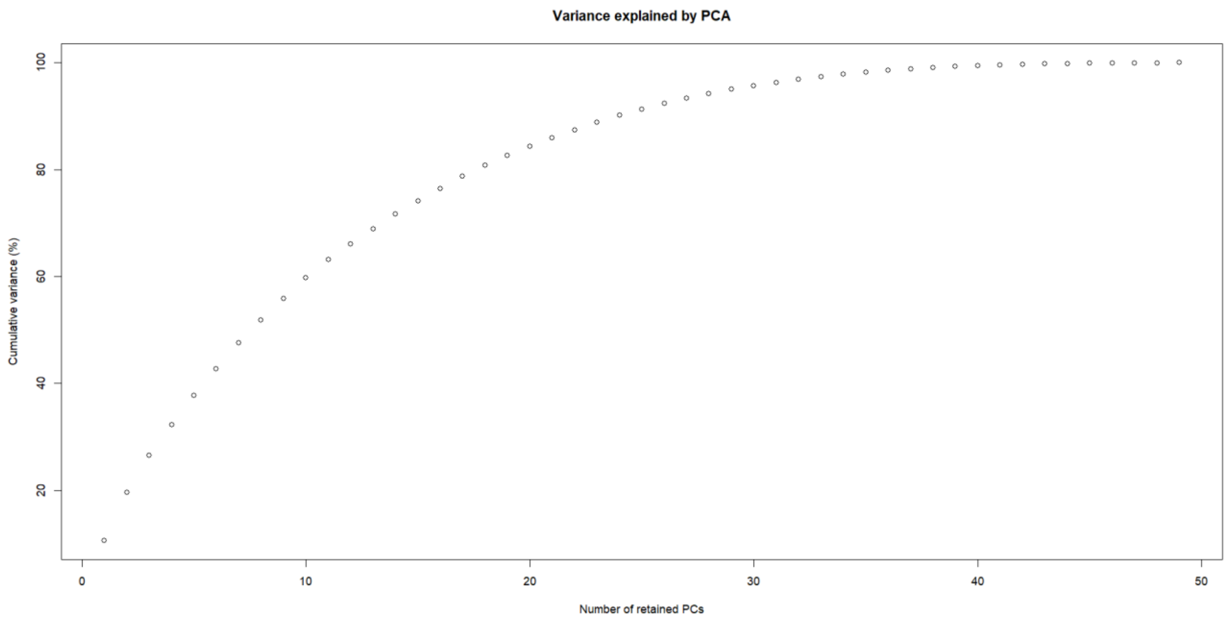


Fig. 3. The percentage cumulative variance explained by PCA for the Ethiopian barley landrace data from 1979.

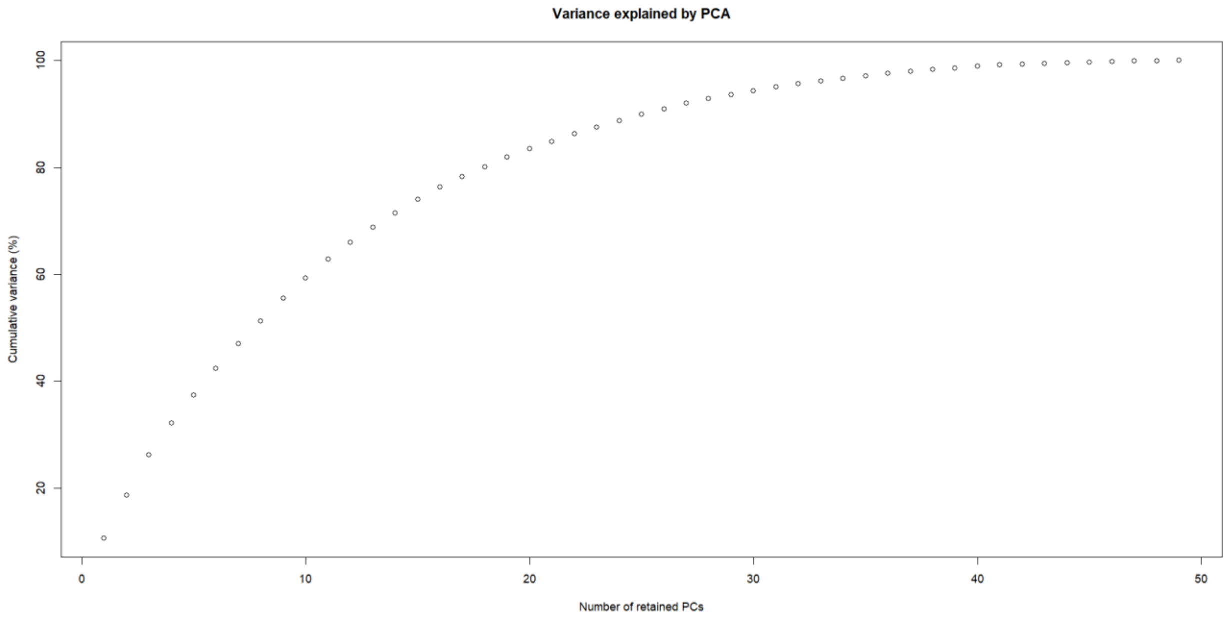


Fig. 4. The percentage cumulative variance explained by PCA for the Ethiopian barley landrace data from 1986.

Unlike when performing *k*-means, we should retain as few PCs as possible without sacrificing information when performing DAPC (Jombart & Collins, 2015). When observing *Fig. 3* and *Fig. 4*, we can see that little additional information is gained after retaining more than around 30 PCs (Jombart & Collins, 2015). For this reason, I chose to retain 30 PCs during both interactive sessions.

The second phase of the interactive session asks the user to choose the number of eigenvalues to retain while outputting the following graphs (one for 1979 and 1986):

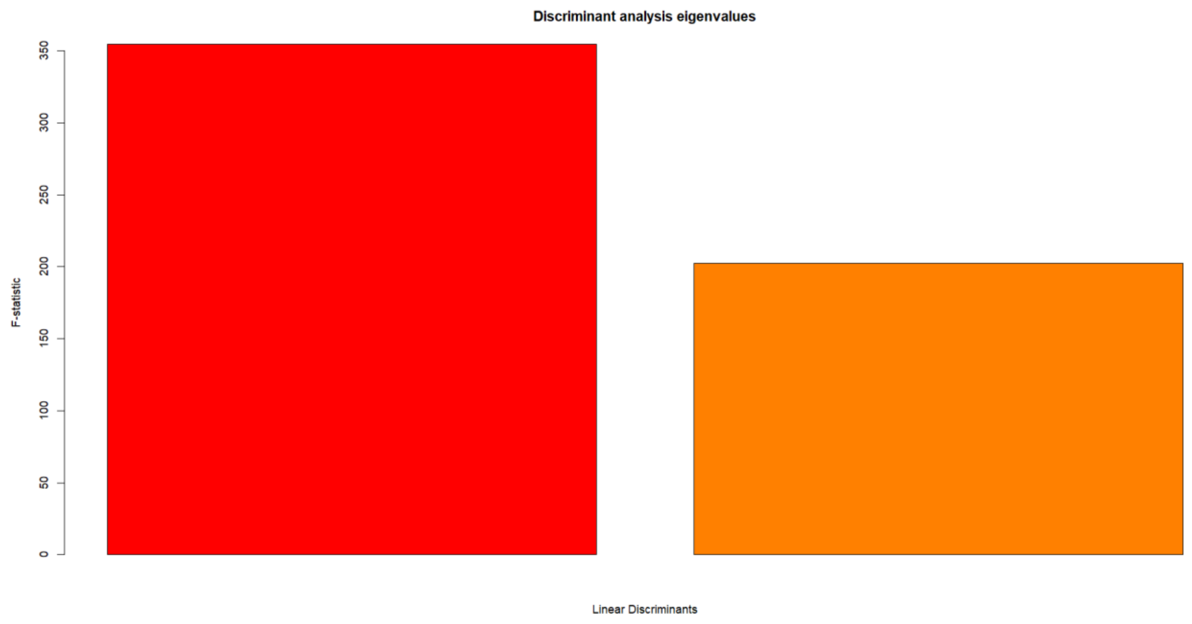


Fig. 5. The F-statistic for each eigenvalue obtained using DA for the Ethiopian barley landrace data from 1979.

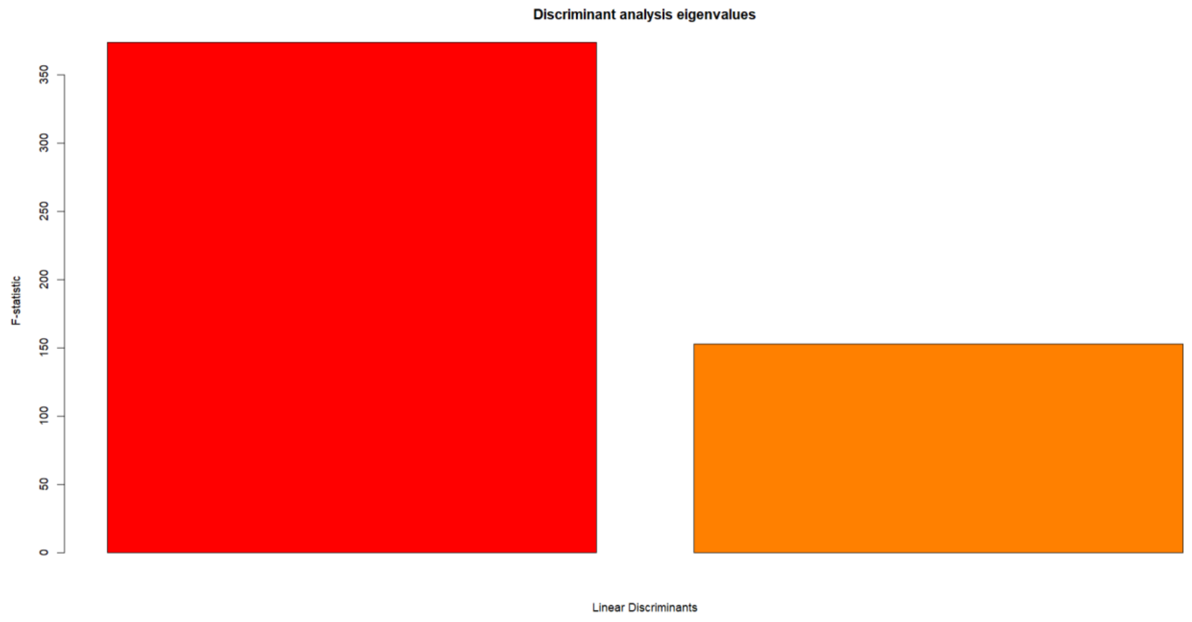


Fig. 6. The F-statistic for each eigenvalue obtained using DA for the Ethiopian barley landrace data from 1986.

I chose to retain all two eigenvalues during both interactive sessions; since we only need to analyze a small number of clusters (i.e., three), R can examine all of them without taking much time and computational effort (Jombart & Collins, 2015). However, when we need to analyze a larger number of clusters, we can retain only the first few eigenvalues, as they carry more information than the rest (Jombart & Collins, 2015).

Plot scatter plots to visualize the clusters obtained by performing DAPC on the data frame subsets for 1979 and 1986.

R

```
scatter(dapc_1979)
scatter(dapc_1986)
```

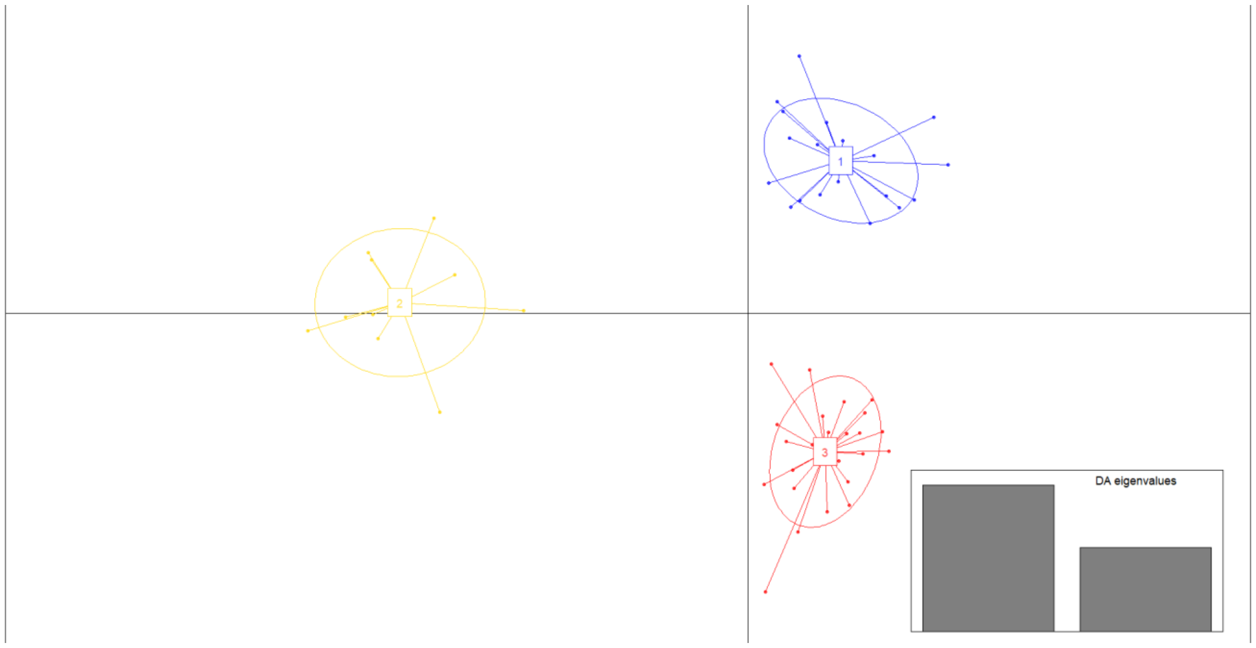


Fig. 7. A scatter plot of the population clusters obtained by performing DAPC on the Ethiopian barley landrace data from 1979.

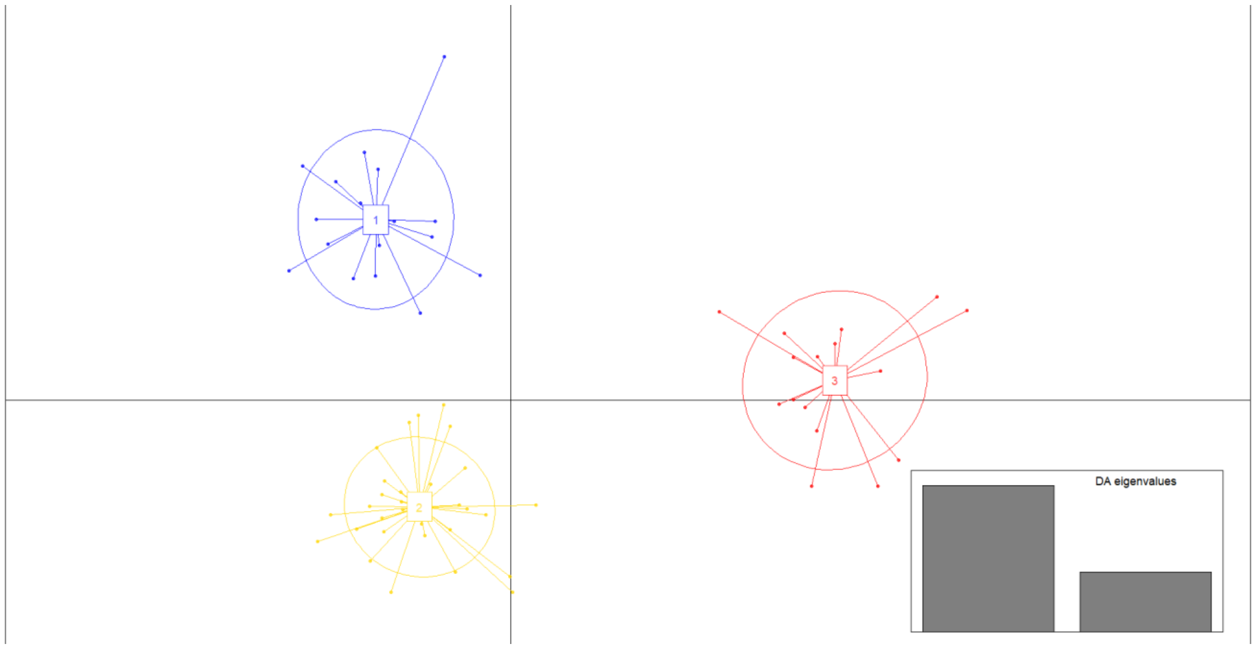


Fig. 8. A scatter plot of the population clusters obtained by performing DAPC on the Ethiopian barley landrace data from 1986.

From observing *Fig. 7* and *Fig. 8*, we know that the most probable number of clusters in the entire data set is indeed three.

Performing DAPC on the Entire Data Set

Now that we know the most probable number of clusters, we can group the data points in our original, sorted data frame (i.e., the one sorted by the collection years in the column "Year collected" in ascending order).

First, convert all data for SSR markers into `numeric` format.

```
R
df_data_sorted[, -c(1, 2, 3, 4)] <- as.numeric(unlist(df_data_sorted[, -c(1, 2, 3, 4)
])) # (Zach, 2021).
```

Then, run `find.clusters` on our original, sorted data frame. This time, specify `n.clust = 3` to tell the function that we want it to find three clusters.

```
R
grp_main <- find.clusters(df_data_sorted[, -c(1, 2, 3, 4)], n.pca = 200, n.clust = 3,
 method = 'kmeans', stat = 'BIC',  max.n.clust = 40)
```

Perform DAPC on our original, sorted data frame. Use the group memberships obtained from running `find.clusters`.

```R
dapc_main <- dapc(df_data_sorted[, -c(1, 2, 3, 4)], grp_main$grp)
```

Again, running `dapc` opens an interactive session that asks the user to choose the number of PCs and eigenvalues to retain. The function also outputs the following graphs:

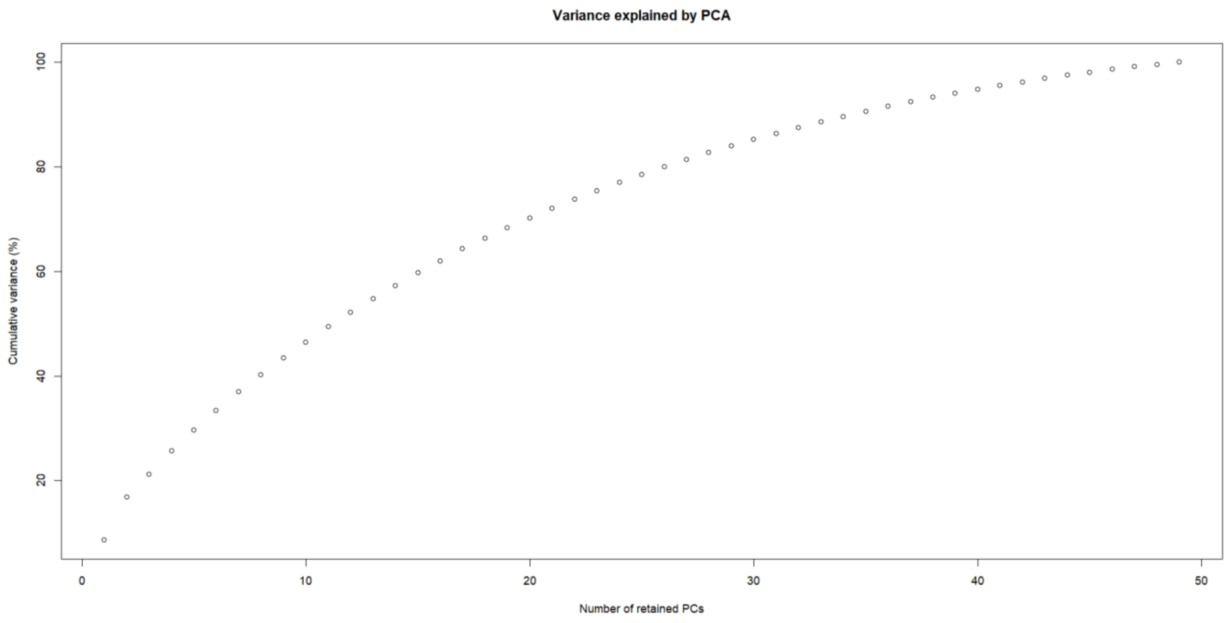


Fig. 9. The percentage cumulative variance explained by PCA for Ethiopian barley landrace data from 1976 to 2017.

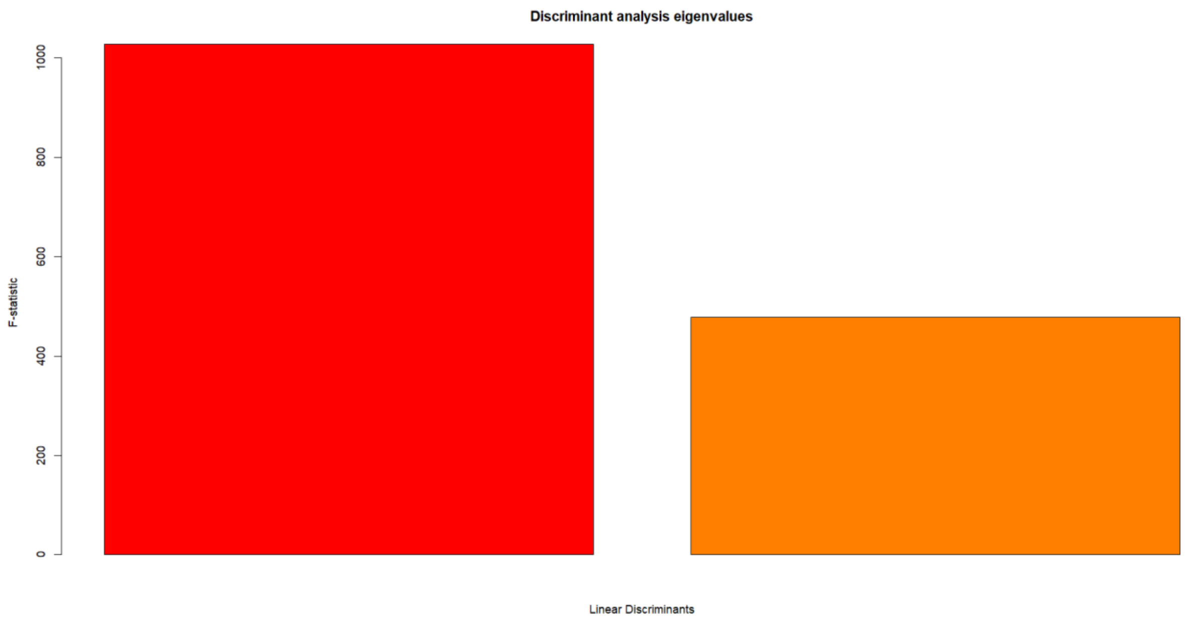


Fig. 10. The F-statistic for each eigenvalue obtained using DA for Ethiopian barley landrace data from 1976 to 2017.

During the interactive session, I chose to retain 40 PCs, as we gain less information after retaining more than 40 PCs. I also chose to retain both eigenvalues.

Plot a scatter plot to visualize the clusters obtained by performing DAPC on our original, sorted data frame. This time, we will customize our scatter plot so that it displays information in an easily understandable and aesthetically pleasing way.

```R
colors <- c('#07099F', '#A40707', '#098D4B') # (Jombart & Collins, 2015).
scatter(dapc_main, col = colors, scree.da = TRUE, scree.pca = TRUE, posi.da = 'topright', posi.pca = 'topleft', bg = 'white', pch = 20, cex = 3) # (Jombart & Collins, 2015).
```

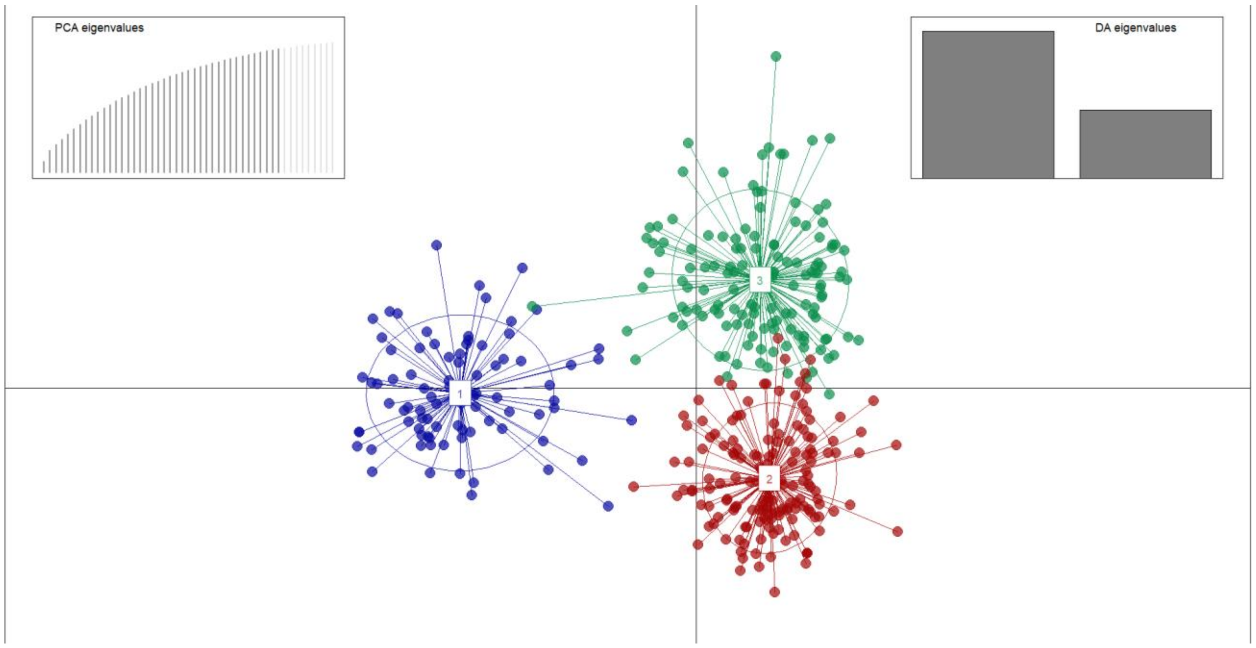


Fig. 11. A scatter plot of the population clusters obtained by performing DAPC on Ethiopian barley landrace data from 1976 to 2017.

Fig. 11 shows that there are three moderately distinct population clusters in our data set. Dido et al. (2021) also arrived at the same conclusion, and you can see the population clusters they identified in *Fig. 12* and *Fig. 13* below.

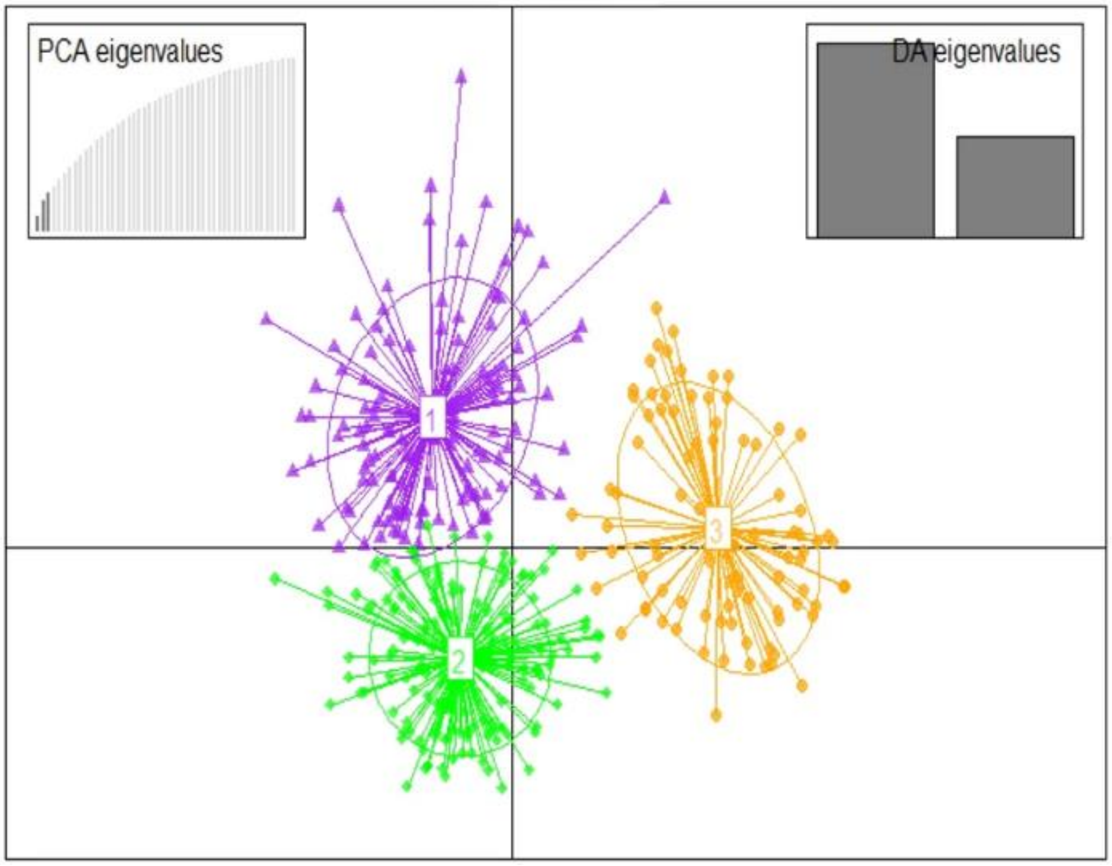


Fig. 12. The first scatter plot of the population clusters obtained by Dido et al. (2021) by performing DAPC on Ethiopian barley landrace data from 1976 to 2017.

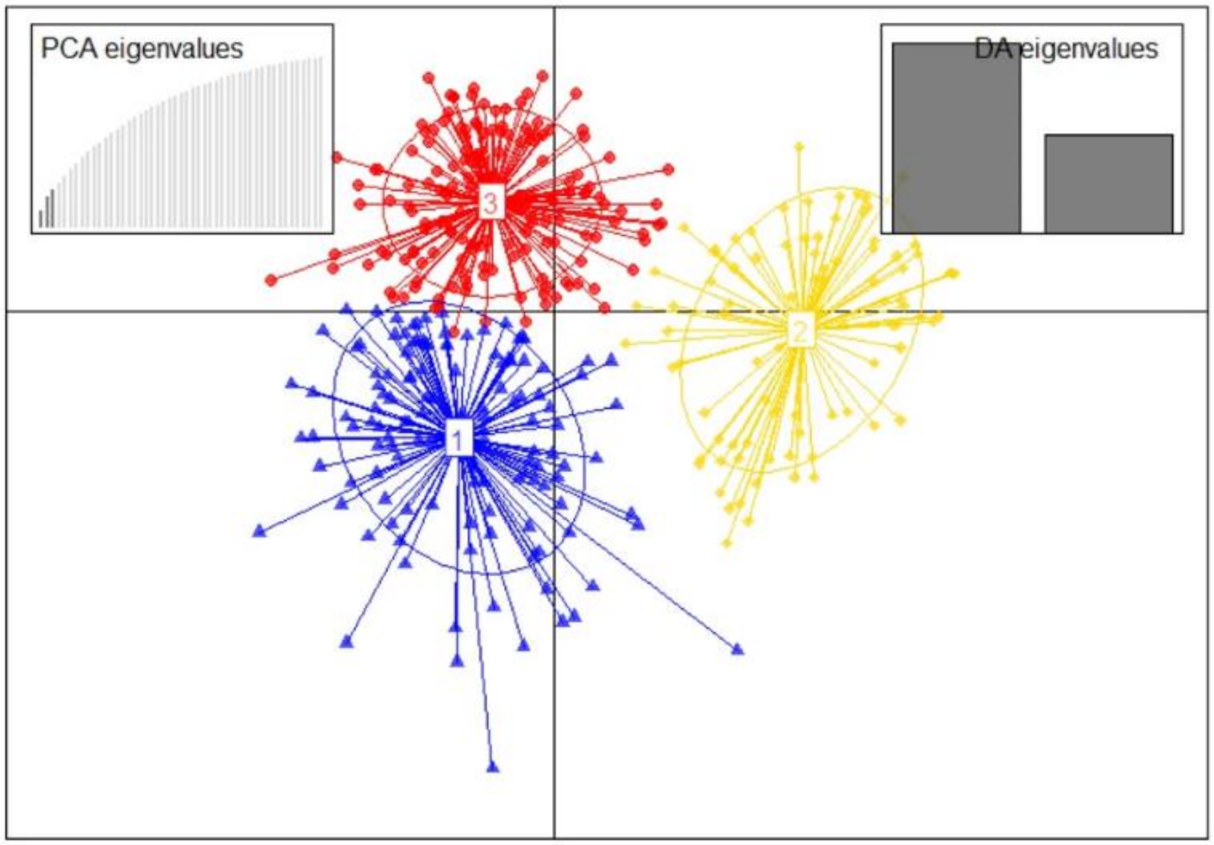


Fig. 13. The second scatter plot of the population clusters obtained by Dido et al. (2021) by performing DAPC on Ethiopian barley landrace data from 1976 to 2017.

Reflection

There are two main reasons as to why I chose to analyze population genetic structures using DAPC for this project. Firstly, discerning the phenotypic differentiation of crop populations via investigating their genetic structures has become an increasingly important task. Because the global climate has become more variable over the past few years, agricultural production systems worldwide struggle to produce adequate yields, leading to food shortages and increased food prices (Dido et al., 2021). A potential solution to make these systems more stable is to grow locally adapted crops, which exhibit greater fitness to their environment compared to crops originating from other regions (Dido et al., 2021). Specifically, locally adapted crops possess phenotypic traits that make them more resistant to the stresses present in their environment, such as being of a certain size, having more efficient photosynthetic processes, or being able to take up more resources from the soil (Dido et al., 2021; Milla, 2023). Therefore, analyzing the genetic structure of a crop population allows us to identify the genes that control these favorable traits and subsequently breed crop varieties for these genes (Dido et al., 2021). Secondly, this project serves to teach me useful skills that I can use in my BINF 6999 summer project, which involves investigating genomic prediction in over 70 Canadian barley lines. Exactly, I can use DAPC to achieve one of the project's aims, which is to identify and analyze allelic differences among the barley lines.

I was able to learn how to conduct DAPC quite quickly, as the documentation on this topic is quite widely available on the internet. One of the most useful tutorials available was written by Jombart & Collins (2015) and provides a thorough, step-by-step guide on how to identify clusters in a data set using DAPC in R using the *adegenet* package. Specifically, Jombart & Collins (2015) outlines all the functions used and their respective purposes, as well as ways to visualize the results of DAPC analyses. In addition, Jombart et al. (2010) also provides a detailed rationale as to why DAPC is superior to both PCA and DA for identifying clusters in a data set.

Although learning how to perform DAPC has been an enjoyable experience for me, I encountered several points throughout this project that may pose significant challenges to others. Firstly, it was rather difficult to process the data set – which was originally in ".xlsx" format – such that it can be used as input to the function `find.clusters`. There were two reasons for this, namely that the data set contains merged columns and contains a gap (i.e., a column with empty rows). Fortunately, I was able to solve these issues by using the function `read.xlsx` from the *openxlsx* package and specifying the argument `fillMerged` as `TRUE`. Secondly, I was unable to find clusters in and perform DAPC on the Ethiopian barley landrace data for each year like Dido et al. (2021) did. Again, the reason for this is that the function `find.clusters` only works on `data.frame`, `matrix`, `genind`, or `genlight` objects containing a substantial number of entries. In the future, I should try finding workarounds for identifying clusters in smaller data frames or conduct my analyses on a different data set entirely.

References

Campoy, J. A., Lerigoleur-Balsemin, E., Christmann, H., Beauvieux, R., Girollet, N., Quero-Garcia, J., Dirlewanger, E., & Barreneche, T. (2016). Genetic diversity, linkage disequilibrium, population structure and construction of a core collection of Prunus avium L. landraces and bred cultivars. *BMC Plant Biology, 16*(1). **https://doi.org/10.1186/s12870-016-0712-9**

Deperi, S. I., Tagliotti, M. E., Bedogni, M. C., Manrique-Carpintero, N. C., Coombs, J. E., Zhang, R., Douches, D. S., & Huarte, M. (2018). Discriminant analysis of principal components and pedigree assessment of genetic diversity and population structure in a tetraploid potato panel using SNPs. *PLOS ONE, 13*(3), e0194398. **https://doi.org/10.1371/journal.pone.0194398**

Dido, A. A., Degefu, D. T., Assefa, E., Krishna, M. S. R., Singh, B. P., & Tesfaye, K. (2021). Spatial and temporal genetic variation in Ethiopian barley (Hordeum vulgare L.) landraces as revealed by simple sequence repeat (SSR) markers. *Agriculture & Food Security, 10*(1). **https://doi.org/10.1186/s40066-021-00336-3**

Dido, A. A., Singh, B. J. K., Assefa, E., Krishna, M. S. R., Degefu, D., & Tesfaye, K. (2022). Spatial and temporal genetic variation in Ethiopian barley (Hordeum vulgare L.) landraces as revealed by simple sequence repeat (SSR) markers. *Dryad*. **https://doi.org/10.5061/dryad.4tmpg4f8v**

Firke, S. (2023, February 2). *janitor: Simple Tools for Examining and Cleaning Dirty Data*. The Comprehensive R Archive Network. **https://CRAN.R-project.org/package=janitor**

Jombart, T. (2008) adegenet: a R package for the multivariate analysis of genetic markers. *Bioinformatics, 24*(11), 1403-1405. **https://doi.org/10.1093/bioinformatics/btn129**

Jombart, T., & Ahmed, I. (2011) adegenet 1.3-1: new tools for the analysis of genome-wide SNP data. *Bioinformatics, 27*(21), 3070-3071. **https://doi.org/10.1093/bioinformatics/btr521**

Jombart, T., & Collins, C. (2015, June 23). *A tutorial for Discriminant Analysis of Principal Components (DAPC) using adegenet 2.0.0*. adegenet on the web. **https://adegenet.r-forge.r-project.org/files/tutorial-dapc.pdf**

Jombart, T., Devillard, S., & Balloux, F. (2010). Discriminant analysis of principal components: a new method for the analysis of genetically structured populations. *BMC Genetics, 11*(1), 94. **https://doi.org/10.1186/1471-2156-11-94**

L, P. [Pierre L]. (2015, August 17). *use first row data as column names in r* [Online forum post]. Stack Overflow. **https://stackoverflow.com/questions/32054368/use-first-row-data-as-column-names-in-r**

Lee, K. J., Lee, J., Sebastin, R., Shin, M., Kim, S., Cho, G., & Hyun, D. Y. (2019). Genetic Diversity *Assessed* by Genotyping by Sequencing (GBS) in Watermelon Germplasm. *Genes*, *10*(10), 822. **https://doi.org/10.3390/genes10100822**

Milla, R. (2023). Phenotypic evolution of agricultural crops. *Functional Ecology*. **https://doi.org/10.1111/1365-2435.14278**

Miller, J. D., Cullingham, C. I., & Peery, R. M. (2020). The influence of a priori grouping on inference of genetic clusters: simulation study and literature review of the DAPC method. *Heredity*, *125*(5), 269–280. **https://doi.org/10.1038/s41437-020-0348-2**

mpalanco [mpalanco]. (2015, August 17). *use first row data as column names in r* [Online forum post]. Stack Overflow. **https://stackoverflow.com/questions/32054368/use-first-row-data-as-column-names-in-r**

RStudio Desktop. (n.d.). Posit. Retrieved April 14, 2023, from **https://posit.co/download/rstudio-desktop/**

Schauberger, P., & Walker, A. (2023). *openxlsx: Read, Write and Edit xlsx Files*. The Comprehensive R Archive Network. **https://CRAN.R-project.org/package=openxlsx**

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., Takahashi, K., Vaughan, D., Wilke, C., Woo, K., Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, *4*(43), 1686. **https://doi.org/10.21105/joss.01686**

Wickham, H., François, R., Henry, L., Müller, K., & Vaughan, D. (2023, March 22). *dplyr: A Grammar of Data Manipulation*. The Comprehensive R Archive Network. **https://CRAN.R-project.org/package=dplyr**

Zach. (2021, May 27). *How to Fix: (list) object cannot be coerced to type 'double'*. Statology. **https://www.statology.org/r-list-object-cannot-be-coerced-to-type-double/**

zek19 [zek19]. (2019, December 12). *use first row data as column names in r* [Online forum post]. Stack Overflow. **https://stackoverflow.com/questions/32054368/use-first-row-data-as-column-names-in-r**