R -Assignment

Lucky Kofi Gbeda

2025-03-19

Load necessary libraries

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.4.3
```

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.4.3
```

```
##
## 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
```

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.4.3
```

```
## Warning: package 'readr' was built under R version 4.4.3
```

```
## Warning: package 'forcats' was built under R version 4.4.3
```

```
## Warning: package 'lubridate' was built under R version 4.4.3
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v forcats   1.0.0     v stringr   1.5.1
## v lubridate 1.9.4     v tibble    3.2.1
## v purrr     1.0.2     v tidyr     1.3.1
## v readr     2.1.5
```

```
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(tidyr)
library(purrr)
```

Load SNP genotype data and SNP position data from input files using read_tsv

read_tsv() reads a tab-separated values (TSV) file into a data frame (or tibble). fang_data will store the contents of fang_et_al_genotypes.txt.

```
fang_data <- read_tsv("~/ISU/SPRING 2025/BCB4560/UNIX_Assignment/fang_et_al_genotypes.txt")
```

```
## Rows: 2782 Columns: 986
## -- Column specification --------------------------------------------------------
## Delimiter: "\t"
## chr (986): Sample_ID, JG_OTU, Group, abph1.20, abph1.22, ae1.3, ae1.4, ae1.5...
##
## 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.
```

```
snp_data <- read_tsv("~/ISU/SPRING 2025/BCB4560/UNIX_Assignment/snp_position.txt")
```

```
## Rows: 983 Columns: 15
## -- Column specification --------------------------------------------------------
## Delimiter: "\t"
## chr (9): SNP_ID, Chromosome, Position, alt_pos, mult_positions, amplicon, cd...
## dbl (6): cdv_marker_id, Genaissance_daa_id, Sequenom_daa_id, count_amplicons...
##
## 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.
```

Data Inspection for Genotype data (fang_et_al_genotypes)

head (fang_data) will display the first few rows of each dataset.
tail (fang_data) will display the last few rows of each dataset.
object.size(fang_data):This function returns the memory size of the fang_data object in bytes.
nrow(fang_data) returns the number of rows in the fang_data data frame.
ncol(fang_data) returns the number of columns (features or variables) in the data frame.
cat() prints the messages in a clean format without quotes.
The slash "n" adds a new line for readability.
dim(fang_data) returns the dataset's dimensions as a vector
dim(fang_data)[1] is the number of rows and dim(fang_data)[2] is the number of columns.
`Count the number of missing values represented as "?/?"`
fang_data == "?/?" creates a logical matrix where each element is TRUE if it equals "?/?", and FALSE

otherwise.
sum(. . . , na.rm = TRUE) counts the number of TRUE values, effectively giving the total number of missing values.
na.rm = TRUE ensures that any existing NA values in fang_data do not interfere with the sum.

```
# Display the first few rows of the fang_data dataset to get an overview of the data.
head(fang_data)
```

```
## # A tibble: 6 x 986
##   Sample_ID JG_OTU   Group abph1.20 abph1.22 ae1.3 ae1.4 ae1.5 an1.4 ba1.6 ba1.9
##   <chr>     <chr>    <chr> <chr>    <chr>    <chr> <chr> <chr> <chr> <chr> <chr>
## 1 SL-15     T-aust-1 TRIPS ?/?      ?/?      T/T   G/G   T/T   C/C   ?/?   G/G
## 2 SL-16     T-aust-2 TRIPS ?/?      ?/?      T/T   ?/?   T/T   C/C   A/G   G/G
## 3 SL-11     T-brav-1 TRIPS ?/?      ?/?      T/T   G/G   T/T   ?/?   G/G   G/G
## 4 SL-12     T-brav-2 TRIPS ?/?      ?/?      T/T   G/G   T/T   C/C   G/G   G/G
## 5 SL-18     T-cund   TRIPS ?/?      ?/?      T/T   G/G   T/T   C/C   ?/?   G/G
## 6 SL-2      T-dact-1 TRIPS ?/?      ?/?      T/T   G/G   T/T   C/C   A/G   G/G
## # i 975 more variables: bt2.5 <chr>, bt2.7 <chr>, bt2.8 <chr>, Fea2.1 <chr>,
## #   Fea2.5 <chr>, id1.3 <chr>, lg2.11 <chr>, lg2.2 <chr>, pbf1.1 <chr>,
## #   pbf1.2 <chr>, pbf1.3 <chr>, pbf1.5 <chr>, pbf1.6 <chr>, pbf1.7 <chr>,
## #   pbf1.8 <chr>, PZA00003.11 <chr>, PZA00004.2 <chr>, PZA00005.8 <chr>,
## #   PZA00005.9 <chr>, PZA00006.13 <chr>, PZA00006.14 <chr>, PZA00008.1 <chr>,
## #   PZA00010.5 <chr>, PZA00013.10 <chr>, PZA00013.11 <chr>, PZA00013.9 <chr>,
## #   PZA00015.4 <chr>, PZA00017.1 <chr>, PZA00018.5 <chr>, ...
```

```
# Display the last few rows of the fang_data dataset to inspect how the data ends.
tail(fang_data)
```

```
## # A tibble: 6 x 986
##   Sample_ID JG_OTU   Group abph1.20 abph1.22 ae1.3 ae1.4 ae1.5 an1.4 ba1.6 ba1.9
##   <chr>     <chr>    <chr> <chr>    <chr>    <chr> <chr> <chr> <chr> <chr> <chr>
## 1 SYN262    Zmm-IL-~ ZMMIL C/C      A/A      T/T   G/G   C/C   C/C   G/G   G/G
## 2 S0398     Zmm-IL-~ ZMMIL G/G      A/A      T/T   G/G   C/C   C/C   G/G   G/G
## 3 S1636     Zmm-IL-~ ZMMIL G/G      A/A      T/T   G/G   C/C   C/C   G/G   G/G
## 4 CU0201    Zmm-IL-~ ZMMIL C/C      A/A      T/T   G/G   C/C   C/C   G/G   G/G
## 5 S0215     Zmm-IL-~ ZMMIL G/G      A/A      T/T   ?/?   C/C   C/C   G/G   G/G
## 6 CU0202    Zmm-IL-~ ZMMIL C/C      A/A      T/T   G/G   C/C   C/C   ?/?   G/G
## # i 975 more variables: bt2.5 <chr>, bt2.7 <chr>, bt2.8 <chr>, Fea2.1 <chr>,
## #   Fea2.5 <chr>, id1.3 <chr>, lg2.11 <chr>, lg2.2 <chr>, pbf1.1 <chr>,
## #   pbf1.2 <chr>, pbf1.3 <chr>, pbf1.5 <chr>, pbf1.6 <chr>, pbf1.7 <chr>,
## #   pbf1.8 <chr>, PZA00003.11 <chr>, PZA00004.2 <chr>, PZA00005.8 <chr>,
## #   PZA00005.9 <chr>, PZA00006.13 <chr>, PZA00006.14 <chr>, PZA00008.1 <chr>,
## #   PZA00010.5 <chr>, PZA00013.10 <chr>, PZA00013.11 <chr>, PZA00013.9 <chr>,
## #   PZA00015.4 <chr>, PZA00017.1 <chr>, PZA00018.5 <chr>, ...
```

```
# Check the size of the dataset
cat("The fang_data object occupies approximately",
    format(object.size(fang_data), units = "MB"), "of memory.\n")
```

```
## The fang_data object occupies approximately 22.1 Mb of memory.
```

```r
# Calculate the number of rows in the fang_data dataset
num_lines_fang_data <- nrow(fang_data)
# Calculate the number of columns  in the fang_data dataset.
num_columns_fang_data <- ncol(fang_data)
cat("The fang_et_al_genotypes dataset contains", num_lines_fang_data, "rows (samples) and",
    num_columns_fang_data, "columns .\n")
```

```
## The fang_et_al_genotypes dataset contains 2782 rows (samples) and 986 columns .
```

```r
# Display the dimensions of the dataset
#dim(fang_data)

cat("The dimensions of the dataset are:", dim(fang_data)[1], "rows and", dim(fang_data)[2], "columns.\n
```

```
## The dimensions of the dataset are: 2782 rows and 986 columns.
```

```r
num_missing_fang <- sum(fang_data == "?/?", na.rm = TRUE)
cat("The sum of the missing values is:", num_missing_fang)
```

```
## The sum of the missing values is: 135452
```

```r
# Count the number of missing values represented as "?/?" in the dataset
num_missing_fang <- sum(fang_data == "?/?", na.rm = TRUE)

# Print the total number of missing values
cat("The sum of the missing values in the fang_data:", num_missing_fang)
```

```
## The sum of the missing values in the fang_data: 135452
```

Data Inspection for Genotype data (fang_et_al_genotypes)

view(fang_data) opens the data in a spreadsheet-like viewer within RStudio.

```r
# View the dataset (This will open it in RStudio's Data Viewer)
cat("Opening the fang_data dataset in the Viewer...\n")
```

```
## Opening the fang_data dataset in the Viewer...
```

```r
View(fang_data)
```

Data Inspection for Genotype data (fang_et_al_genotypes)

str(fang_data) displays the structure of the fang_data object including data type of each column

```r
# Display the structure of the dataset
cat("Here is the structure of the fang_data dataset:\n")
```

```
## Here is the structure of the fang_data dataset:
```

```r
#str(fang_data)
```

The same file inspections were don for SNP Postiion data

Data Inspection for SNP Position data (snp_position)

```r
# Display the first few rows of the snp_data dataset to get an overview of the data.
head(snp_data)
```

```
## # A tibble: 6 x 15
##   SNP_ID   cdv_marker_id Chromosome Position  alt_pos mult_positions amplicon
##   <chr>            <dbl> <chr>      <chr>     <chr>   <chr>          <chr>
## 1 abph1.20          5976 2          27403404  <NA>    <NA>           abph1
## 2 abph1.22          5978 2          27403892  <NA>    <NA>           abph1
## 3 ae1.3             6605 5          167889790 <NA>    <NA>           ae1
## 4 ae1.4             6606 5          167889682 <NA>    <NA>           ae1
## 5 ae1.5             6607 5          167889821 <NA>    <NA>           ae1
## 6 an1.4             5982 1          240498509 <NA>    <NA>           an1
## # i 8 more variables: cdv_map_feature.name <chr>, gene <chr>,
## #   `candidate/random` <chr>, Genaissance_daa_id <dbl>, Sequenom_daa_id <dbl>,
## #   count_amplicons <dbl>, count_cmf <dbl>, count_gene <dbl>
```

```r
# Display the last few rows of the snp_data dataset to inspect how the data ends.
tail(snp_data)
```

```
## # A tibble: 6 x 15
##   SNP_ID cdv_marker_id Chromosome Position  alt_pos mult_positions amplicon
##   <chr>          <dbl> <chr>      <chr>     <chr>   <chr>          <chr>
## 1 zap1.2          3514 2          233128584 <NA>    <NA>           zap1
## 2 zen1.1          3519 unknown    unknown   <NA>    <NA>           zen1
## 3 zen1.2          3520 unknown    unknown   <NA>    <NA>           zen1
## 4 zen1.4          3521 unknown    unknown   <NA>    <NA>           zen1
## 5 zfl2.6          6463 2          12543294  <NA>    <NA>           zfl2
## 6 zmm3.4          3527 9          16966348  <NA>    <NA>           zmm3
## # i 8 more variables: cdv_map_feature.name <chr>, gene <chr>,
## #   `candidate/random` <chr>, Genaissance_daa_id <dbl>, Sequenom_daa_id <dbl>,
## #   count_amplicons <dbl>, count_cmf <dbl>, count_gene <dbl>
```

```r
# Check the size of the dataset
cat("The snp_data object occupies approximately",
    format(object.size(snp_data), units = "MB"), "of memory.\n")
```

```
## The snp_data object occupies approximately 0.3 Mb of memory.
```

```r
# Calculate the number of rows in the snp_data dataset
num_lines_snp_data <- nrow(snp_data)

# Calculate the number of columns in the snp_data dataset.
num_columns_snp_data <- ncol(snp_data)

cat("The snp_position dataset contains", num_lines_snp_data, "rows (SNP positions) and",
    num_columns_snp_data, "columns.\n")
```

```
## The snp_position dataset contains 983 rows (SNP positions) and 15 columns.
```

```r
# Display the dimensions of the dataset
cat("The dimensions of the dataset are:", dim(snp_data)[1], "rows and", dim(snp_data)[2], "columns.\n")
```

```
## The dimensions of the dataset are: 983 rows and 15 columns.
```

```r
# Count the number of missing values represented as "?/?" in the dataset
num_missing_snp <- sum(snp_data == "?/?", na.rm = TRUE)

# Print the total number of missing values
cat("The sum of the missing values in snp_data is:", num_missing_snp)
```

```
## The sum of the missing values in snp_data is: 0
```

Data Inspection for SNP Position data (snp_position)

```r
# View the snp_position dataset (This will open it in RStudio's Data Viewer)
cat("Opening the snp_position in the Viewer...\n")
```

```
## Opening the snp_position in the Viewer...
```

```r
View(snp_data)
```

Data Inspection for SNP Position data (snp_position)

```r
# Display the structure of the dataset
cat("Here is the structure of the snp_position:\n")
```

```
## Here is the structure of the snp_position:
```

```r
str(snp_data)
```

```
## spc_tbl_ [983 x 15] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ SNP_ID              : chr [1:983] "abph1.20" "abph1.22" "ae1.3" "ae1.4" ...
##  $ cdv_marker_id       : num [1:983] 5976 5978 6605 6606 6607 ...
##  $ Chromosome          : chr [1:983] "2" "2" "5" "5" ...
##  $ Position            : chr [1:983] "27403404" "27403892" "167889790" "167889682" ...
##  $ alt_pos             : chr [1:983] NA NA NA NA ...
##  $ mult_positions      : chr [1:983] NA NA NA NA ...
##  $ amplicon            : chr [1:983] "abph1" "abph1" "ae1" "ae1" ...
##  $ cdv_map_feature.name: chr [1:983] "AB042260" "AB042260" "ae1" "ae1" ...
##  $ gene                : chr [1:983] "abph1" "abph1" "ae1" "ae1" ...
##  $ candidate/random    : chr [1:983] "candidate" "candidate" "candidate" "candidate" ...
##  $ Genaissance_daa_id  : num [1:983] 8393 8394 8395 8396 8397 ...
##  $ Sequenom_daa_id     : num [1:983] 10474 10475 10477 10478 10479 ...
##  $ count_amplicons     : num [1:983] 1 0 1 0 0 1 1 0 1 0 ...
##  $ count_cmf           : num [1:983] 1 0 1 0 0 1 0 0 1 0 ...
##  $ count_gene          : num [1:983] 1 0 1 0 0 1 1 0 1 0 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   SNP_ID = col_character(),
##   ..   cdv_marker_id = col_double(),
##   ..   Chromosome = col_character(),
##   ..   Position = col_character(),
##   ..   alt_pos = col_character(),
##   ..   mult_positions = col_character(),
##   ..   amplicon = col_character(),
##   ..   cdv_map_feature.name = col_character(),
##   ..   gene = col_character(),
##   ..   `candidate/random` = col_character(),
##   ..   Genaissance_daa_id = col_double(),
##   ..   Sequenom_daa_id = col_double(),
##   ..   count_amplicons = col_double(),
##   ..   count_cmf = col_double(),
##   ..   count_gene = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

Data Processing

t(fang_data) wiil transposes the dataset, swapping rows and columns.
as.data.frame(..., stringsAsFactors = FALSE) will converts the transposed matrix back into a data frame while preventing automatic conversion of text to factors. fang_transposed[3,]:is used to extracts the third row to use as column names, and colnames(fang_transposed) <- ...:is used to assign the third row values as the new column names for the transposed data.
rownames(fang_transposed): Retrieves the original row names from the transposed data and cbind(...):is used to combines the original row names as a new first column (Original_Colnames).
fang_transposed[-c(1:3),]: removes the first 3 rows, which are no longer needed after setting column names.
colnames(fang_transposed)[1] <- "SNP_ID": Renames the first column to "SNP_ID".
rownames(fang_transposed) <- NULL: resets row names to sequential integers starting from 1.

```
# Transpose the fang_data dataset to switch rows and columns
fang_transposed <- as.data.frame(t(fang_data), stringsAsFactors = FALSE)
```

```r
# View the transposed data (optional, for checking)
view(fang_transposed)

# Convert the third row into column names (assuming row 3 contains SNP names)
colnames(fang_transposed) <- fang_transposed[3, ]

# Add the original row names as a new first column to preserve them
fang_transposed <- cbind(Original_Colnames = rownames(fang_transposed), fang_transposed)

# Remove the first 3 rows as they are not needed after setting column names
fang_transposed <- fang_transposed[-c(1:3), ]

# Rename the first column to "SNP_ID" for clarity
colnames(fang_transposed)[1] <- "SNP_ID"

# View the cleaned transposed data (optional, for checking)
view(fang_transposed)

# Reset row names to ensure they are sequential and start from 1
rownames(fang_transposed) <- NULL
```

Joining the transposed genotype dat and the SNP position data by SNP

select(snp_data, SNP_ID, Chromosome, Position): Extracts SNP_ID, Chromosome, and Position columns from snp_data.
make.unique(colnames(fang_transposed)): Ensures unique column names in fang_transposed.
left_join(snp_info, fang_transposed, by = "SNP_ID"): Merges snp_info with fang_transposed based on SNP_ID.
write.table(df_joined, "joined_data.txt", sep = "\t", row.names = FALSE, quote = FALSE): Saves the joined dataset as a tab-separated text file without row names or quotes.

```r
# Select relevant columns (SNP_ID, Chromosome, Position) from snp_data
snp_info <- select(snp_data, SNP_ID, Chromosome, Position)

# Ensure unique column names in fang_transposed to avoid duplication issues
colnames(fang_transposed) <- make.unique(colnames(fang_transposed))

# Perform a left join to merge snp_info with fang_transposed based on SNP_ID
df_joined <- left_join(snp_info, fang_transposed, by = "SNP_ID")

# View the resulting dataframe in RStudio
view(df_joined)

# Save the joined dataset as a tab-separated text file
write.table(df_joined, "joined_fang_snp_data.txt", sep = "\t", row.names = FALSE, quote = FALSE)
```

Maize

Extract relevant columns for maize

The code below uses select() to extract relevant columns from df_joined.
Keeps SNP_ID, Chromosome, and Position columns.
Selects all columns whose names start with "ZMMIL", "ZMMLR", or "ZMMMR" (presumably maize-related data).
The write.table(...) saves df_joined as a tab-separated text file named maize_data.txt inside the "Maize" directory
row.names = FALSE prevents row numbers from being written.
quote = FALSE ensures that values are not enclosed in quotes.

```
  # Extract relevant columns for maize, including SNP_ID, Chromosome, Position, and columns starting wi
df_maize <- df_joined %>% select(SNP_ID, Chromosome = Chromosome, Position = Position,
                                 starts_with("ZMMIL"), starts_with("ZMMLR"), starts_with("ZMMMR"))

# View the maize-specific dataset in RStudio
view(df_maize)

write.table(df_joined, "Maize/maize_data.txt", sep = "\t", row.names = FALSE, quote = FALSE)
```

Sort by ascending order of SNP position

The code below uses arrange () to sort the df_maize dataset in ascending order based on the Position column. The result is saved as df_maize_sorted_incre. the sorted dataset (df_maize_sorted_incre) as a tab-separated.txt file called maize_sorted_incre_data.txt in the "Maize" directory

```
# Sort by SNP position in ascending order
df_maize_sorted_incre <- df_maize %>% arrange(Position)

# Save the sorted maize dataset as a tab-separated text file
write.table(df_maize_sorted_incre, "Maize/maize_sorted_incre_data.txt", sep = "\t", row.names = FALSE, 
```

Extract rows with 'multiple' or 'unknown' in Chromosome column for Maize

The uses filter () to filter df_maize_sorted_incre to keep only rows where the Chromosome column has the value "multiple" and saves the result in df_multiple.
Similarly silters df_maize_sorted_incre to keep only rows where the Chromosome column has the value "unknown". The filtered df_multiple and df_unknown datasets as tab-separated .txt files, named maize_chromo_multiple.txt and maize_chromo_unknown.txt are respectively saved in the "Maize" directory

```
# Extract rows with 'multiple' in the Chromosome column
df_multiple <- df_maize_sorted_incre %>% filter(Chromosome == "multiple")

# Extract rows with 'unknown' in the Chromosome column
df_unknown <- df_maize_sorted_incre %>% filter(Chromosome == "unknown")
```

```r
# Save the 'multiple' chromosome data to a tab-separated text file
write.table(df_multiple, "Maize/maize_chromo_multiple.txt",sep = "\t", row.names = FALSE, quote = FALSE)

# Save the 'unknown' chromosome data to a tab-separated text file
write.table (df_unknown, "Maize/maize_chromo_unknown.txt",sep = "\t", row.names = FALSE, quote = FALSE)
```

Create separate chromosome files for Increasing order of SNP Position for maize

The uses for loop to over chromosome numbers from 1 to 10.
the code filters df_maize_sorted_incre to include only rows where the chromosome column matches the current chromosome number (chr_num) and saves the filtered result as df_chr.
The filtered chromosome dataset (df_chr) are saved to a tab-separated text file. The file name includes the chromosome number (chr_num), such as Maize_incre_chromo1.txt for chromosome 1.

```r
#Increasing SNP Position (Maize)
# Create separate chromosome files for chromosomes 1 to 10
for (chr_num in 1:10) {
  # Filter rows for each chromosome number
  df_chr <- df_maize_sorted_incre %>% filter(Chromosome == chr_num)

  # Save the filtered chromosome data to a tab-separated text file without row names or quotes
  write.table(df_chr, paste0("Maize/Maize_Increasing_Chromo/Maize_incre_chromo", chr_num, ".txt"), sep =
}
```

Create separate chromosome files for decreasing order of SNP Position for maize

Repalce “?/?” by “-/-” and Sort by decreasing order of SNP position

This code performs the following steps:
df_maize_dash[df_maize_dash == “?/?”] <- “-/-”. This step directly replaces the “?/?” placeholder with “-/-” in the dataset.
The modified dataset is saved as a tab-separated .txt file without row names and quotes.
The arrange(desc(Position))is used to sort the dataset by the Position column in descending order. The sorted dataset is then saved as a tab-separated .txt file.

```r
# maize decreasing SNP Positions
# Create a copy of the maize dataset
df_maize_dash <- df_maize

# Replace all occurrences of "?/?" with "-/-" in the dataset
df_maize_dash[df_maize_dash == "?/?"] <- "-/-"

# View the modified dataset in RStudio
view(df_maize_dash)
```

```r
# Save the modified dataset to a tab-separated text file
write.table(df_maize_dash, "Maize/data_maize_dash.txt", sep = "\t", row.names = FALSE, quote = FALSE)

# Sort the modified dataset by Position in descending order
df_maize_sorted_decre_dash <- df_maize_dash %>% arrange(desc(Position))

# View the sorted dataset in RStudio
view(df_maize_sorted_decre_dash)

# Save the sorted dataset to a tab-separated text file
write.table(df_maize_sorted_decre_dash, "Maize/maize_sorted_decre_dash.txt", sep = "\t", row.names = FA
```

Generate separate chromosome files for decreasing order of SNP Position for maize

The uses for loop to over chromosome numbers from 1 to 10.
the code filters df_maize_sorted_incre to include only rows where the chromosome column matches the current chromosome number (chr_num) and saves the filtered result as df_chr.
The filtered chromosome dataset (df_chr) are saved to a tab-separated text file. The file name includes the chromosome number (chr_num), such as Maize_decre_chromo1.txt for chromosome 1.

```r
# Create separate chromosome files for chromosomes 1 to 10
for (chr_num in 1:10) {
  # Filter rows for each chromosome number
  df_chr <- df_maize_sorted_decre_dash %>% filter(Chromosome == chr_num)

  # Save the filtered chromosome data to a tab-separated text file
  write.table(df_chr, paste0("Maize/Maize_Decreasing_Chromo/Maize_decre_chromo", chr_num, ".txt"), sep =
}
```

Teosinte

Similar to the codes used in maize, the codes below do same in teosinte. The summary of what the codes is below with detailed explanation seen in maize

Extract relevant columns for Teosinte

Extract Relevant Columns for Teosinte
The script extracts SNP data for Teosinte from a combined dataset (df_joined).
It selects SNP_ID, Chromosome, Position, and columns starting with “ZMPBA”, “ZMPIL”, or “ZMPJA”.
Saves the extracted data as “data_teosinte.txt”

```r
# Define the directory path for saving the output file
file_dir = "~/ISU/SPRING 2025/BCB4560/R_Assignment_Final/Teosinte/"

# Extract relevant columns for Teosinte from the joined dataframe
# Selecting SNP_ID, Chromosome, Position, and columns that start with "ZMPBA", "ZMPIL", or "ZMPJA"
```

```
df_teosinte <- df_joined %>% select(SNP_ID, Chromosome = Chromosome, Position = Position,
                                    starts_with("ZMPBA"), starts_with("ZMPIL"), starts_with("ZMPJA"))

# View the extracted dataframe in RStudio
view(df_teosinte)

# Save the extracted Teosinte data to a text file with tab-separated values
write.table(df_teosinte, paste0(file_dir, "data_teosinte.txt"), sep = "\t", row.names = FALSE, quote = F
```

Sort by ascending order of SNP position (Teosinte)

Sorts SNPs by Position in ascending order.
Saves the sorted data as "data_teosinte_sorted_incre.txt".

```
# Sort the Teosinte dataframe by SNP position in increasing order
df_teosinte_sorted_incre <- df_teosinte %>% arrange(Position)

# View the sorted dataframe in RStudio
view(df_teosinte_sorted_incre)

# Save the sorted Teosinte data to a text file with tab-separated values
write.table(df_teosinte_sorted_incre, paste0(file_dir, "data_teosinte_sorted_incre.txt"),
            sep = "\t", row.names = FALSE, quote = FALSE)
```

Extract rows with 'multiple' or 'unknown' in Chromosome column (Teosinte)

dentifies SNPs where Chromosome is labeled as "multiple" or "unknown". Saves them separately as:"Teosinte_chromo_multiple.txt" and "Teosinte_chromo_unknown.txt"

```
# Extract rows where the Chromosome column has the value "multiple"
df_multiple <- df_teosinte_sorted_incre %>% filter(Chromosome == "multiple")

# Extract rows where the Chromosome column has the value "unknown"
df_unknown <- df_teosinte_sorted_incre %>% filter(Chromosome == "unknown")

# Save the extracted data for "multiple" chromosomes to a text file with tab-separated values
write.table(df_multiple, paste0(file_dir, "Teosinte_chromo_multiple.txt"),
            sep = "\t", row.names = FALSE, quote = FALSE)

# Save the extracted data for "unknown" chromosomes to a text file with tab-separated values
write.table(df_unknown, paste0(file_dir, "Teosinte_chromo_unknown.txt"),
            sep = "\t", row.names = FALSE, quote = FALSE)
```

Increasing SNP Position (Teosinte)

terates over chromosomes 1 to 10.
Filters SNPs for each chromosome.

Saves them in the "Teosinte_Increasing_Chromo" directory as:
"Teosinte_incre_chromo1.txt"
"Teosinte_incre_chromo2.txt", etc.

```
# Create separate chromosome files for Teosinte (Increasing SNP Positions)
# Create separate files for each chromosome (1 to 10) in Teosinte dataset, sorted by increasing SNP pos
for (chr_num in 1:10) {

  # Filter the dataframe to include only rows where Chromosome matches the current chromosome number
  df_chr <- df_teosinte_sorted_incre %>% filter(Chromosome == chr_num)

  # Save the filtered data to a text file, named according to the chromosome number
  write.table(df_chr, paste0(file_dir, "/Teosinte_Increasing_Chromo/Teosinte_incre_chromo", chr_num, ".
              sep = "\t", row.names = FALSE, quote = FALSE)
}
```

Decreasing SNP Position (Teosinte)

Repalce "?/?" by "-/-" and Sort by decreasing order of SNP position

Creates a copy of the dataset.
Replaces all "?/?" genotypes with "-/-".
Sorts SNPs by Position in descending order.
Saves the cleaned dataset as "data_teosinte_sorted_decre_dash.txt".

```
# Create a copy of the Teosinte dataframe to modify the genotype format
df_teosinte_dash <- df_teosinte

# Replace all occurrences of "?/?" with "-/-" in the dataframe
df_teosinte_dash[df_teosinte_dash == "?/?"] <- "-/-"

# View the modified dataframe in RStudio
view(df_teosinte_dash)

# Save the modified Teosinte data to a text file with tab-separated values
write.table(df_teosinte_dash, paste0(file_dir, "data_teosinte_dash.txt"),
            sep = "\t", row.names = FALSE, quote = FALSE)

# Sort the modified dataframe by SNP position in decreasing order
df_teosinte_sorted_decre_dash <- df_teosinte_dash %>% arrange(desc(Position))

# View the sorted dataframe in RStudio
view(df_teosinte_sorted_decre_dash)

# Save the sorted Teosinte data to a text file with tab-separated values
write.table(df_teosinte_sorted_decre_dash, paste0(file_dir, "data_teosinte_sorted_decre_dash.txt"),
            sep = "\t", row.names = FALSE, quote = FALSE)
```

Create Separate Files for Each Chromosome (Decreasing SNP Positions)

Similar to Step 4 but with SNPs sorted in descending order.
Saves files in "Teosinte_Decreasing_Chromo" directory.

```
# Create separate chromosome files for Teosinte (Decreasing SNP Positions)
# Create separate files for each chromosome (1 to 10) in the Teosinte dataset, sorted by decreasing SNP
for (chr_num in 1:10) {

  # Filter the dataframe to include only rows where Chromosome matches the current chromosome number
  df_chr <- df_teosinte_sorted_decre_dash %>% filter(Chromosome == chr_num)

  # Save the filtered data to a text file, named according to the chromosome number, using tab-separate
  write_tsv(df_chr, paste0(file_dir, "/Teosinte_Decreasing_Chromo/Teosinte_decre_chromo", chr_num, ".tx
}
```

#Part II Visualization

The group_by(Chromosome) function organizes the dataset (df_maize) into groups based on unique values in the Chromosome column.
The summarise(SNP_Count = n()) function calculates the number of rows (SNPs) in each chromosome group and stores the count in a new column called SNP_Count.
The mutate(Group = "Maize") function adds a new column called Group to the resulting dataframe and assigns the value "Maize" to all rows. This helps in labeling the data when combining results from multiple datasets.

Count SNPs per chromosome for maize

```
# Count the number of SNPs per chromosome for the maize dataset
snp_count_maize <- df_maize %>%

  # Group the data by the Chromosome column
  group_by(Chromosome) %>%

  # Count the number of SNPs in each chromosome group
  summarise(SNP_Count = n()) %>%

  # Add a new column to indicate the dataset source as "Maize"
  mutate(Group = "Maize")
```

Count SNPs per chromosome for teosinte

```
# Count the number of SNPs per chromosome for the Teosinte dataset
snp_count_teosinte <- df_teosinte %>%

  # Group the data by the Chromosome column
  group_by(Chromosome) %>%
```

```
  # Count the number of SNPs in each chromosome group
  summarise(SNP_Count = n()) %>%

  # Add a new column to indicate the dataset source as "Teosinte"
  mutate(Group = "Teosinte")
```

#Combining Data from Two Dataframes:

The bind_rows(snp_count_maize, snp_count_teosinte) function merges the snp_count_maize and snp_count_teosinte dataframes by stacking them on top of each other.
This works because both dataframes have the same column structure (Chromosome, SNP_Count, and Group).

```
# Combine SNP count data from both Maize and Teosinte datasets
snp_counts <- bind_rows(snp_count_maize, snp_count_teosinte)
```

#Creating the Plot This code generates a bar plot to visualize the SNP distribution across chromosomes for both Maize and Teosinte

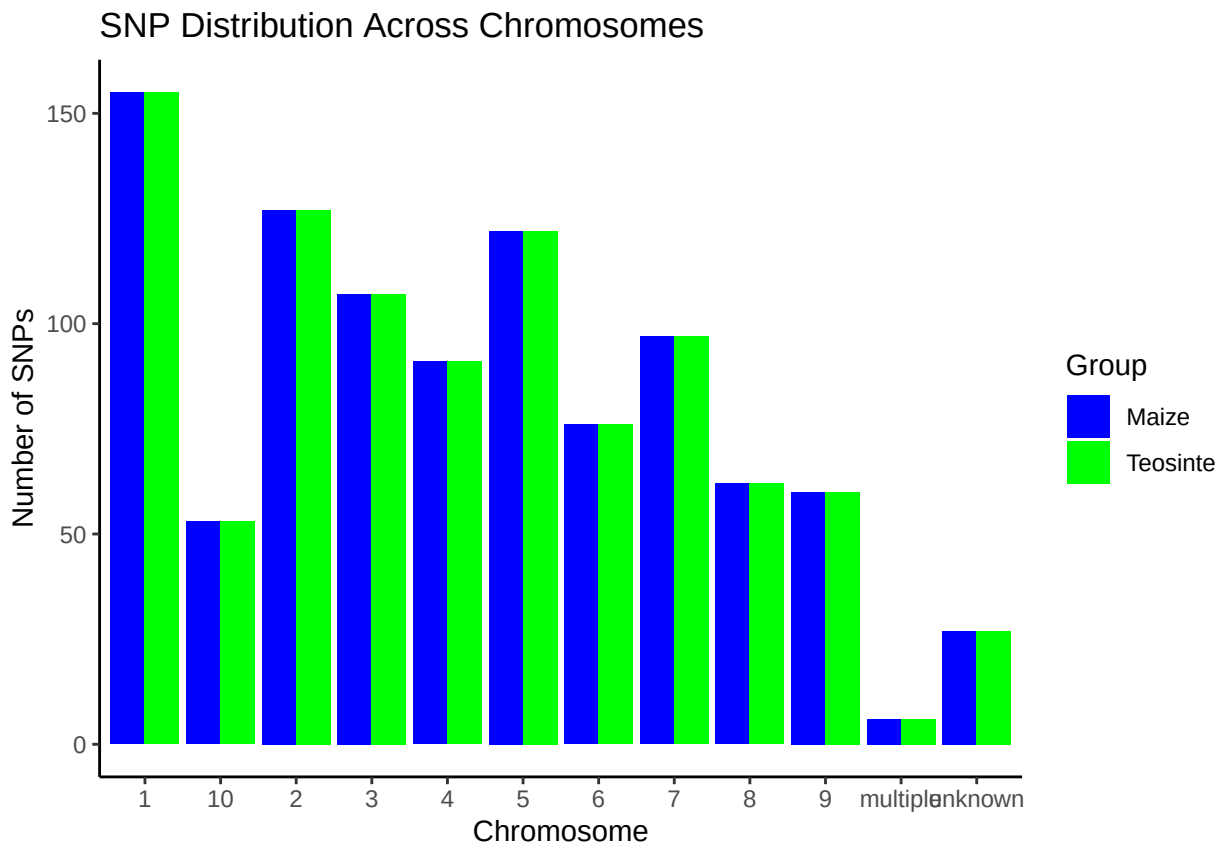
```
# Plot SNP count per chromosome for Maize and Teosinte
ggplot(snp_counts, aes(x = Chromosome, y = SNP_Count, fill = Group)) +

  # Create a bar plot with "Chromosome" on the x-axis and "SNP_Count" on the y-axis
  # The bars are grouped by "Group" (Maize vs. Teosinte)
  geom_bar(stat = "identity", position = "dodge") +

  # Add labels to the plot
  labs(title = "SNP Distribution Across Chromosomes",
       x = "Chromosome", y = "Number of SNPs") +

  # Set custom colors for the "Maize" and "Teosinte" groups
  scale_fill_manual(values = c("Maize" = "blue", "Teosinte" = "green")) +

  # Use a clean and simple theme for the plot
  theme_classic()
```



For optimized mapping

This code reshapes the genotype data from both Maize and Teosinte, classifies each SNP as "Homozygous", "Heterozygous", or "Missing", and then combines both datasets into a single dataframe (df_combined) that includes both Maize and Teosinte genotypes.

pivot_longer: Converts the maize data from wide format (where each sample is a column) to long format (where each row represents a genotype for a specific SNP and sample).The SNP_ID, Chromosome, and Position columns are kept intact, while the rest of the columns (representing samples) are "pivoted" into two new columns: "Sample" and "Genotype".

mutate(SNP_Type = map_chr(Genotype, classify_snp)): Uses the classify_snp function to classify the Genotype for each row. The result is stored in a new column called SNP_Type.
bind_rows: Combines the df_maize_long and df_teosinte_long datasets into one dataframe. The mutate(Group = "Maize") and mutate(Group = "Teosinte") commands add a new column called "Group" to each dataset to label whether the data comes from "Maize" or "Teosinte".

df_combined: This combined dataframe contains data for both Maize and Teosinte, with columns: SNP_ID, Chromosome, Position, Sample, Genotype, SNP_Type, and Group.

```
# Function to classify SNPs
classify_snp <- function(genotype) {
  switch(genotype,
         "?/?" = "Missing",
         ifelse(grepl("A/A|C/C|G/G|T/T", genotype), "Homozygous", "Heterozygous"))
}
```

```r
# Apply classification to maize data
df_maize_long <- df_maize %>%
  pivot_longer(cols = -c(SNP_ID, Chromosome, Position),  # Select all except metadata columns
               names_to = "Sample", values_to = "Genotype") %>%
  mutate(SNP_Type = map_chr(Genotype, classify_snp))
# Faster than sapply
head(df_maize_long)
```

```
## # A tibble: 6 x 6
##   SNP_ID   Chromosome Position Sample  Genotype SNP_Type
##   <chr>    <chr>      <chr>    <chr>   <chr>    <chr>
## 1 abph1.20 2          27403404 ZMMIL   G/G      Homozygous
## 2 abph1.20 2          27403404 ZMMIL.1 C/C      Homozygous
## 3 abph1.20 2          27403404 ZMMIL.2 C/C      Homozygous
## 4 abph1.20 2          27403404 ZMMIL.3 C/C      Homozygous
## 5 abph1.20 2          27403404 ZMMIL.4 C/C      Homozygous
## 6 abph1.20 2          27403404 ZMMIL.5 G/G      Homozygous
```

```r
# Apply classification to teosinte data
df_teosinte_long <- df_teosinte %>%
  pivot_longer(cols = -c(SNP_ID, Chromosome, Position),  # Select all except metadata columns
               names_to = "Sample", values_to = "Genotype") %>%
  mutate(SNP_Type = map_chr(Genotype, classify_snp))

head(df_teosinte_long)
```

```
## # A tibble: 6 x 6
##   SNP_ID   Chromosome Position Sample  Genotype SNP_Type
##   <chr>    <chr>      <chr>    <chr>   <chr>    <chr>
## 1 abph1.20 2          27403404 ZMPBA   C/G      Heterozygous
## 2 abph1.20 2          27403404 ZMPBA.1 C/G      Heterozygous
## 3 abph1.20 2          27403404 ZMPBA.2 G/G      Homozygous
## 4 abph1.20 2          27403404 ZMPBA.3 G/G      Homozygous
## 5 abph1.20 2          27403404 ZMPBA.4 C/G      Heterozygous
## 6 abph1.20 2          27403404 ZMPBA.5 C/G      Heterozygous
```

```r
# Combine both datasets
df_combined <- bind_rows(
  df_maize_long %>% mutate(Group = "Maize"),
  df_teosinte_long %>% mutate(Group = "Teosinte")
)

# Combine the long format data for both maize and teosinte
df_long <- bind_rows(df_maize_long, df_teosinte_long)

# Summarize SNP types by group (Maize or Teosinte)
df_snp_summary <- df_combined %>%

  # Group the data by "Group" (Maize or Teosinte) and "SNP_Type" (Homozygous, Heterozygous, Missing)
  group_by(Group, SNP_Type) %>%

  # Count the number of occurrences for each SNP type in each group
```

```
  summarise(Count = n(), .groups = "drop") %>%

  # Calculate the proportion of each SNP type within each group
  mutate(Proportion = Count / sum(Count))

# View the summarized results (SNP counts and proportions)
print(df_snp_summary)
```

```
## # A tibble: 6 x 4
##   Group    SNP_Type       Count Proportion
##   <chr>    <chr>          <int>      <dbl>
## 1 Maize    Heterozygous  207813     0.0830
## 2 Maize    Homozygous   1263501     0.504
## 3 Maize    Missing        74945     0.0299
## 4 Teosinte Heterozygous  202086     0.0807
## 5 Teosinte Homozygous    713892     0.285
## 6 Teosinte Missing        42447     0.0169
```

Count proportions of SNP types per group

df_long <- bind_rows(maize_long, teosinte_long): Combines the long-format maize and teosinte datasets into a single dataframe (df_long). group_by(Group, SNP_Type): Groups the data by two columns: Group (which differentiates between Maize and Teosinte) and SNP_Type (Homozygous, Heterozygous, or Missing).

summarise(Count = n(), .groups = "drop"): For each group and SNP type combination, it counts the number of occurrences (i.e., how many times each SNP type appears for a given group). The .groups = "drop" argument ensures that the grouping structure is removed after summarizing.

mutate(Proportion = Count / sum(Count)): Adds a new column (Proportion) that calculates the proportion of each SNP type within each group by dividing the count of each SNP type by the total count within the group.

print(df_snp_summary): Displays the summarized results, showing the count and proportion of each SNP type (Homozygous, Heterozygous, Missing) for both Maize and Teosinte.

```
# Summarize SNP types by group (Maize or Teosinte)
df_snp_summary <- df_combined %>%

  # Group the data by "Group" (Maize or Teosinte) and "SNP_Type" (Homozygous, Heterozygous, Missing)
  group_by(Group, SNP_Type) %>%

  # Count the number of occurrences for each SNP type in each group
  summarise(Count = n(), .groups = "drop") %>%

  # Calculate the proportion of each SNP type within each group
  mutate(Proportion = Count / sum(Count))

# View the summarized results (SNP counts and proportions)
print(df_snp_summary)
```

```
## # A tibble: 6 x 4
##   Group    SNP_Type       Count Proportion
```

```
##   <chr>    <chr>          <int>      <dbl>
## 1 Maize    Heterozygous  207813     0.0830
## 2 Maize    Homozygous   1263501     0.504
## 3 Maize    Missing        74945     0.0299
## 4 Teosinte Heterozygous  202086     0.0807
## 5 Teosinte Homozygous    713892     0.285
## 6 Teosinte Missing        42447     0.0169
```

Plot SNP Type distribution per Chromosome

ggplot(df_combined, aes(x = Chromosome, fill = SNP_Type)): Initializes the plot using df_combined, with Chromosome on the x-axis and different colors for each SNP_Type (e.g., Homozygous, Heterozygous, Missing).

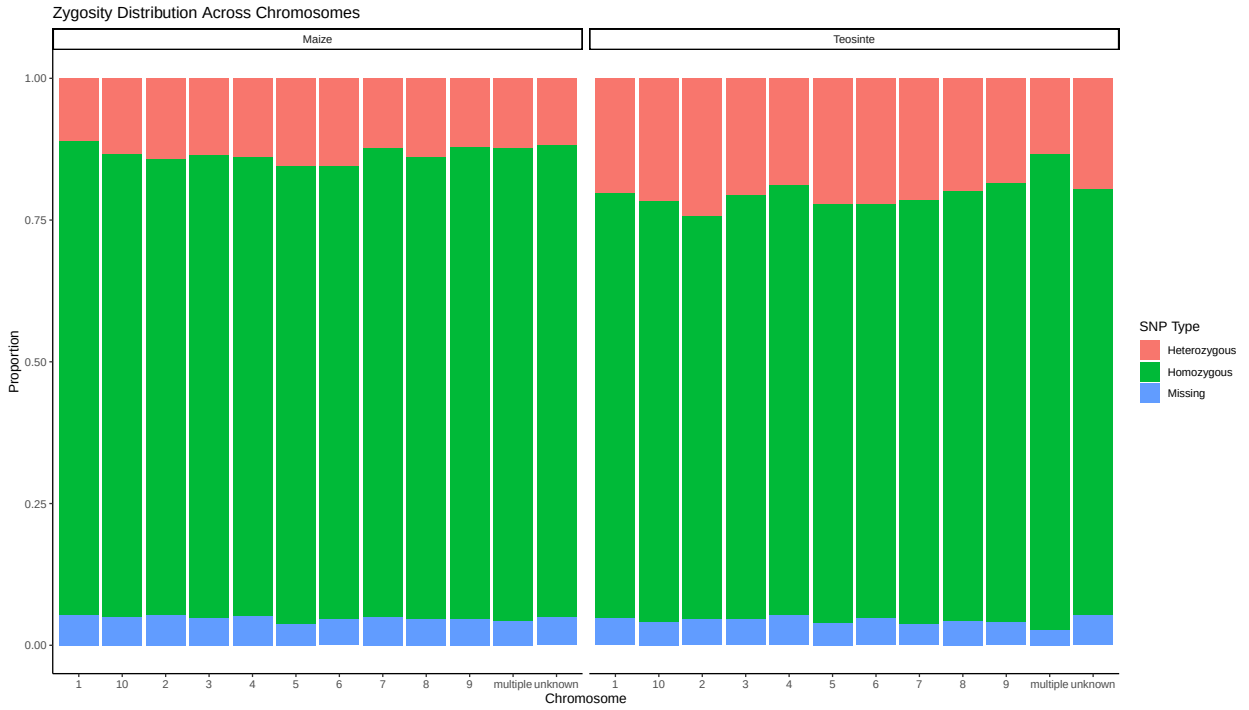
geom_bar(position = "fill"): Creates a bar chart where the height of each bar represents the proportion of each SNP type, not the count, scaling the bars to show percentages.

facet_wrap(~ Group): Creates separate subplots for each group (Maize and Teosinte), showing the SNP distribution across chromosomes for each group.

labs(...): Adds titles and axis labels: Title: "Zygosity Distribution Across Chromosomes", X-axis: "Chromosome" Y-axis: "Proportion", Legend: "SNP Type" theme_classic(): Applies a clean, minimalist theme to the plot, removing gridlines for a simpler appearance.

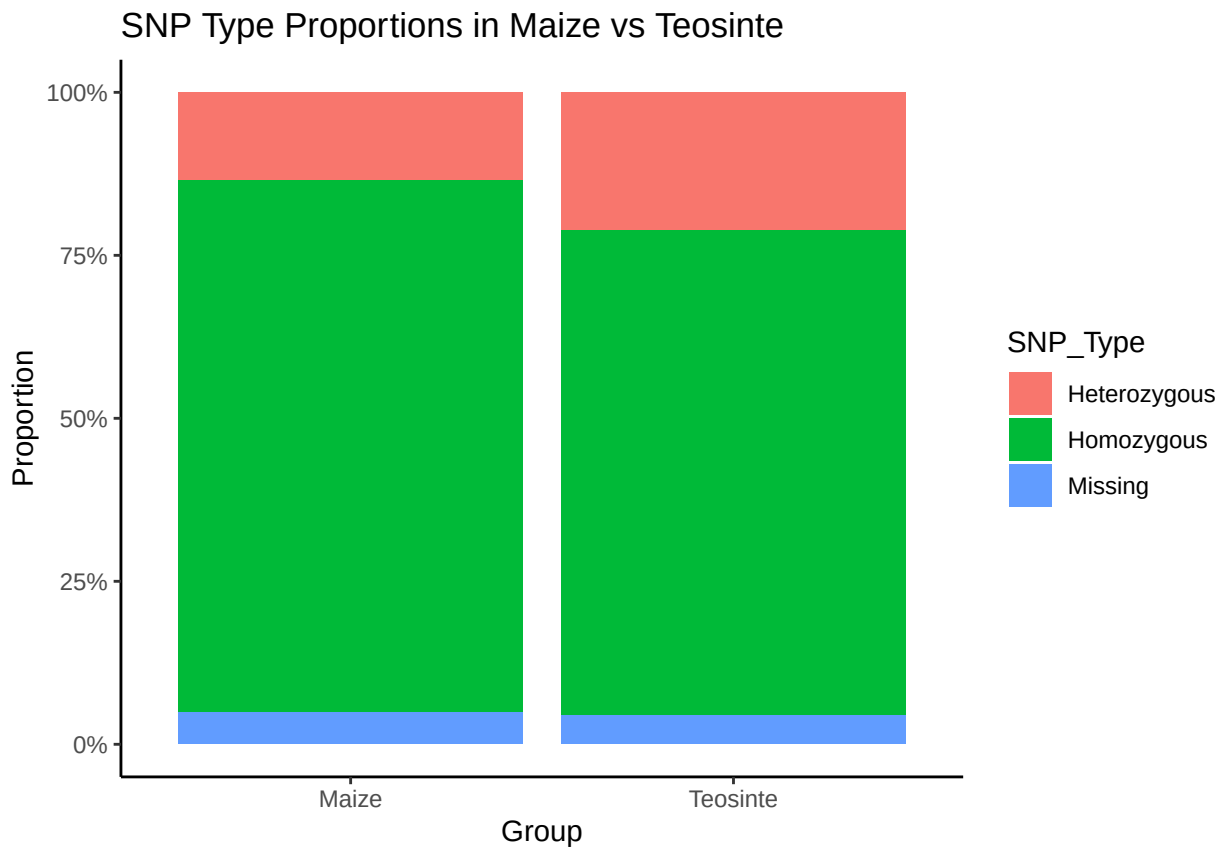
This creates a clear, proportion-based bar chart of SNP distribution across chromosomes for both Maize and Teosinte.

```
ggplot(df_combined, aes(x = Chromosome, fill = SNP_Type)) +
  geom_bar(position = "fill") +  # "fill" makes proportions instead of counts
  facet_wrap(~ Group) +  # Separate plots for Maize & Teosinte
  labs(title = "Zygosity Distribution Across Chromosomes",
       x = "Chromosome", y = "Proportion",
       fill = "SNP Type") +
  theme_classic()
```



#Proportional Bar Plot (SNP Distribution by Group) Shows the proportions of SNP types (Homozygous, Heterozygous) in Maize vs Teosinte.
Uses a stacked bar chart where the height represents the proportion of each SNP type.
Y-axis labels are formatted as percentages.

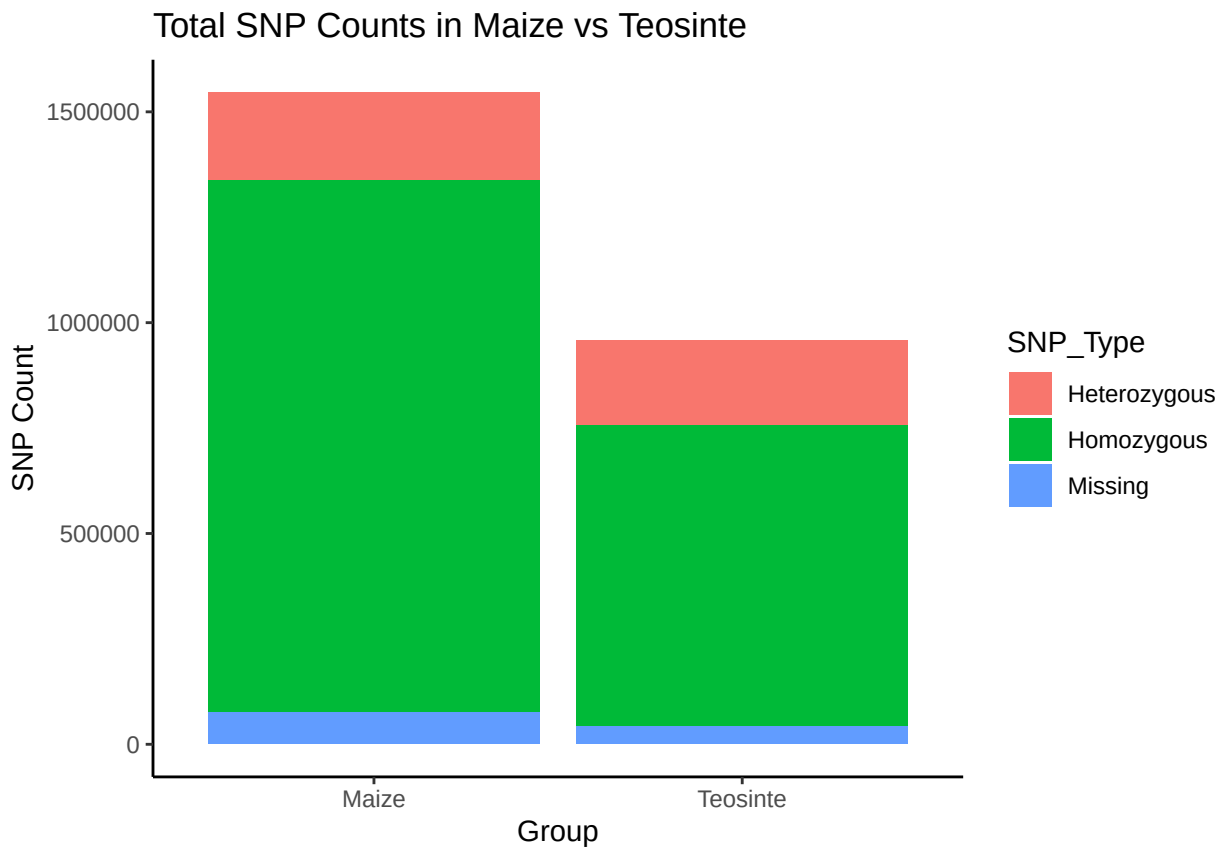
```
#Bar Plot - SNP Distribution by Group
ggplot(df_snp_summary, aes(x = Group, y = Proportion, fill = SNP_Type)) +
  geom_bar(stat = "identity", position = "fill") +  # Stacked proportional bar chart
  labs(title = "SNP Type Proportions in Maize vs Teosinte",
       x = "Group", y = "Proportion") +
  scale_y_continuous(labels = scales::percent) +
  theme_classic()
```



Stacked Bar Plot (Absolute SNP Counts):

Displays the total SNP counts for each SNP type in Maize vs Teosinte.
Uses a stacked bar chart where the total height represents the sum of SNP counts, segmented by SNP type.

```
#Stacked Bar Plot - Absolute SNP Counts
ggplot(df_snp_summary, aes(x = Group, y = Count, fill = SNP_Type)) +
  geom_bar(stat = "identity", position = "stack") +  # Stacked bar chart
  labs(title = "Total SNP Counts in Maize vs Teosinte",
       x = "Group", y = "SNP Count") +
  theme_classic()
```

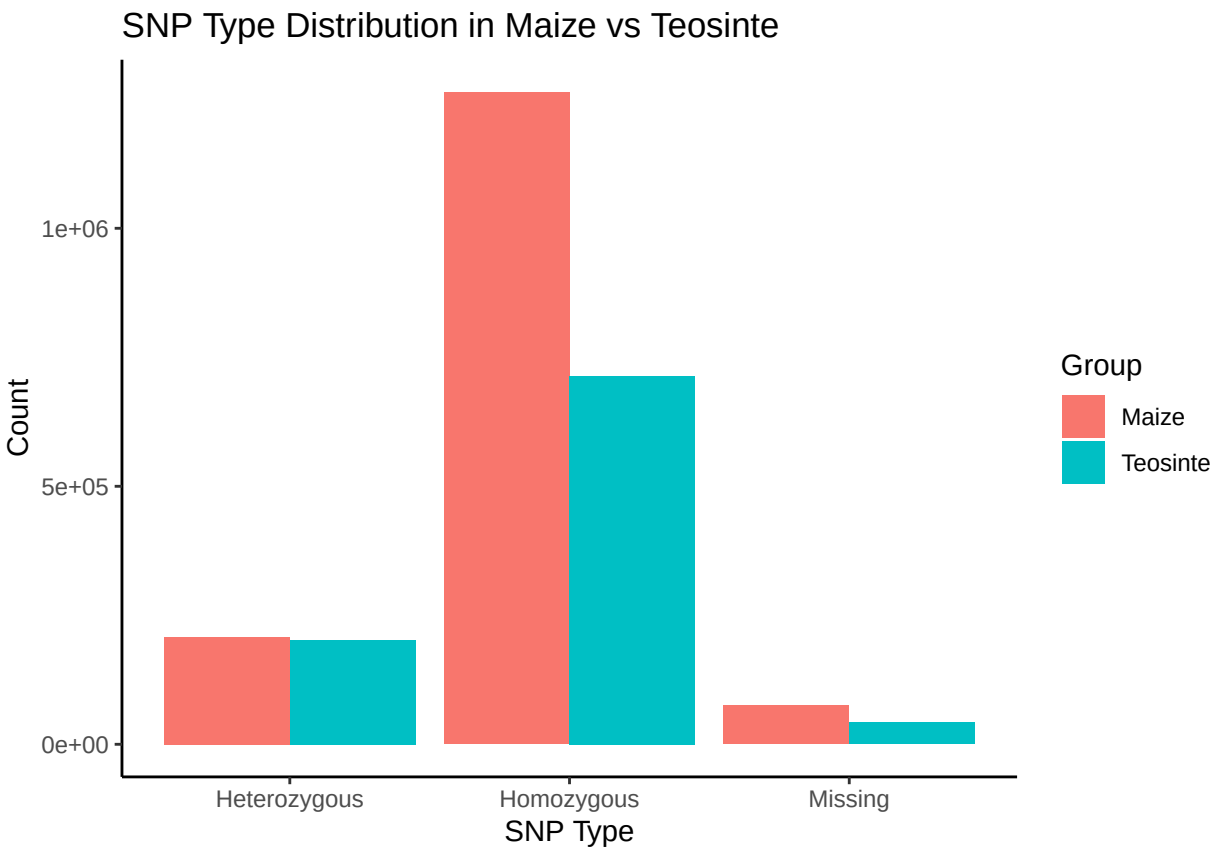


#SNP Distribution Bar Plot per Sample:

Displays the count of each SNP Type (Homozygous, Heterozygous, etc.) in Maize vs Teosinte.
Uses a dodge position for a side-by-side comparison of SNP types across the two groups (Maize and Teosinte).
Labels include the title: "SNP Type Distribution in Maize vs Teosinte", X-axis: "SNP Type", and Y-axis: "Count".
The plot uses a clean theme (theme_classic()).

```
# Bar Plot - SNP Distribution per Sample
ggplot(df_combined, aes(x = SNP_Type, y = ..count.., fill = Group)) +
  geom_bar(position = "dodge") +
  labs(title = "SNP Type Distribution in Maize vs Teosinte",
       x = "SNP Type", y = "Count") +
  theme_classic()
```

```
## Warning: The dot-dot notation (`..count..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(count)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```


SNP Type Distribution in Maize vs Teosinte
Count
1e+06
5e+05
0e+00
Heterozygous
Homozygous
Missing
SNP Type
Group
Maize
Teosinte