DArTSeq SNPs: Visualisation and Filtering

Dasyurus hallucatus

Robyn Shaw

2022-08-03

Contents

Required packages:

- knitr (Xie, 2014, 2015, 2022)
- flextable (Gohel, 2022a)
- officer (Gohel, 2022b)
- captioner (Alathea, 2015)
- dartR (Gruber, Unmack, Berry, & Georges, 2018)
- ggplot2 (Wickham, 2016)
- dplyr (Wickham, François, Henry, & Müller, 2022)
- tibble (Müller & Wickham, 2022)
- tidyr (Wickham & Girlich, 2022)
- tidyverse (Wickham et al., 2019)
- viridis (Garnier et al., 2021)
- gridExtra (Auguie, 2017)
- ggpubr (Kassambara, 2020)
- grid (R Core Team, 2021)
- corrplot (Wei & Simko, 2021)
- SNPRelate (Zheng et al., 2012)
- rgdal (Bivand, Keitt, & Rowlingson, 2021)
- ggsn (Santos Baquero, 2019)
- sessioninfo (Wickham, Chang, Flight, Müller, & Hester, 2021)
- details (Sidi, 2022)

1 Input variables and check population sample sizes

1.1 Define path to data folder, file names and variables

```
# Path to folder containing data
datpath <- "Data/"

# Define prefix for naming output files
Outfile_Desc <- "Dhal"

# Create a new directory/sub folder for saving outputs (if it doesn't already exist)
outpath <- "Filtering_outputs/"
ifelse(!dir.exists(file.path(outpath)), dir.create(file.path(file.path(outpath))), FALSE)


# Name of DArT SNP data (1-row)
DartGenos_1Row <- "Report_DDasy19-4717_1Row_NameEdit.csv"

# Name of last metadata column in SNP files before samples start
LastGeno.MD.Col <- "RepAvg"

# Name of DArT read count data and 2-row genotype file Make sure that sample names are in
# the same row as the rest of the headers before loading in
DartGenos_2Row <- "Report_DDasy19-4717_2Row_NameEdit.csv"
DartReadCounts <- "ReadCounts_DDasy19-4717_NameEdit.csv"

# Name of last metadata column in read count file before samples start
LastRC.MD.Col <- "TotalPicRepSnpTest"

# Name of individual meta data file Must include column for: sample IDs (that match IDs
# in DArT SNP file) and populations
Ind_Metadat <- "Dasyurus_hallucatus_ind.metadata.csv"
pop_col <- "pop"

# Have DArT SNPs been blasted to a reference? (Y/N)
BLAST <- "Y"

# Add contig info from NCBI if available This will allow us to see which chromosome the
# SNPs are on (rather than just the contig name) Also add DArT column names for the
# chromosome/contig ID, position, alignment count and e value
if (BLAST == "Y") {
    # If sex chromosomes are known, list here (so they can be filtered out)
    SexChrom <- "x"
    ChromInfo.file <- "TasDevil7.1_Scaffold_Info.csv"
    ContigCol_dart <- "Chrom_Tasmanian_devil_v7"
    ContigPos_dart <- "ChromPos_Tasmanian_devil_v7"
    ContigCol_ncbi <- "Devil7.0.SCAFFOLD"
    ChromPos_ncbi <- "CHROMOSOME.POSITION"
    EVal.col <- "AlnEvalue_Tasmanian_devil_v7"
```

```
    Cnt.col <- "AlnCnt_Tasmanian_devil_v7"
}


# Path to shapefile for sample map
Map.shp <- "PilbaraIBRA.shp"

# Define mapping variables

# Map extent This will be used to place the north arrow and the scale bar So make it
# slightly smaller than the actual map extent
xmin <- 115
xmax <- 122
ymin <- -24
ymax <- -19.5

# Define fill variable for map (e.g. here it is IBRA sub-regions)
MapFill <- "SUB_NAME_7"

# Define variables for map scale bar (distance unit, crs, distance increments)
distU <- "km"
CoordRef <- "WGS84"
dist <- 100
```

1.2 Determine population sample size cutoff

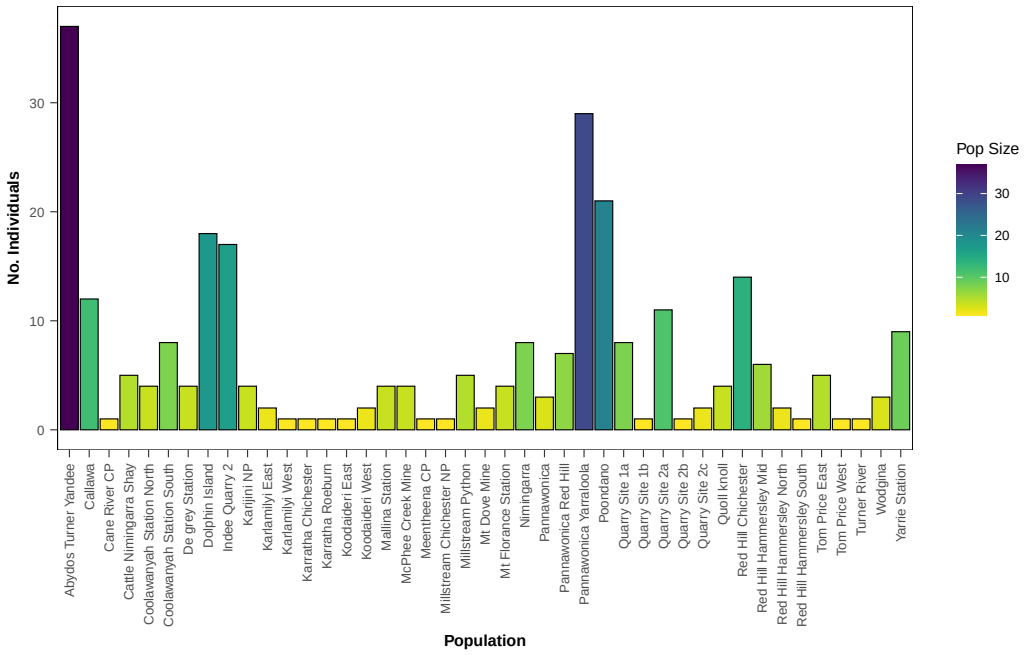

Figure 1: Population sample sizes.

Table 1: Sample number by population.

Population	N
Abydos Turner Yandee	37
Callawa	12
Cane River CP	1
Cattle Nimingarra Shay	5
Coolawanyah Station North	4
Coolawanyah Station South	8
De grey Station	4
Dolphin Island	18
Indee Quarry 2	17
Karijini NP	4
Karlamilyi East	2
Karlamilyi West	1
Karratha Chichester	1
Karratha Roeburn	1
Koodaideri East	1
Koodaideri West	2
Mallina Station	4
McPhee Creek Mine	4
Meentheena CP	1
Millstream Chichester NP	1
Millstream Python	5
Mt Dove Mine	2
Mt Florance Station	4
Nimingarra	8
Pannawonica	3
Pannawonica Red Hill	7
Pannawonica Yarraloola	29
Poondano	21
Quarry Site 1a	8
Quarry Site 1b	1
Quarry Site 2a	11
Quarry Site 2b	1
Quarry Site 2c	2
Quoll knoll	4

Population	N
Red Hill Chichester	14
Red Hill Hammersley Mid	6
Red Hill Hammersley North	2
Red Hill Hammersley South	1
Tom Price East	5
Tom Price West	1
Turner River	1
Wodgina	3
Yarrie Station	9

Use Figure 1 and Table 1 to decide which populations to include in calculating population level metrics. If populations have a small sample size, the population genetic metrics are less robust and will throw off mean values. Choose a threshold that makes sense for the data set, at least 10 if possible (ideally at least three populations with sample sizes at or above 30). It's worth noting that the population summary statistics calculated in this script are only used to visualise the data (in other words, filtering is not based on these population groupings and their associated summary statistics). However, it's still a good idea to decide on reasonable populations, to see if any strange population genetic patterns have real biological meaning, or if they are instead correlated with other measures such as quality.

In this case, there are quite a few populations with very good sample sizes. Therefore, the sampling site will be used as the 'population.'

```
# If the sample sizes in defined pops are too low, choose a different way to group
# samples (i.e. merge some pops, use regional info etc.)

# Do you want to choose a different population grouping? (Y/N)
Change_pop <- "N"

if (Change_pop == "Y") {
    # Choose a new column to group samples by (in individual metadata file)
    pop_col <- "IBRA_SubRegion"
    PopNo <- data.frame(table(ind.metrics[, pop_col]))

    popNo.plot <- ggplot(data = PopNo, aes(x = Var1, y = Freq, fill = Freq)) +
↪   geom_bar(stat = "identity",
        colour = "black", size = 0.2) + scale_fill_viridis(discrete = F, option = "D",
        ↪  direction = -1) +
        labs(x = "Population", y = "No. Individuals", fill = "Pop Size") +
        ↪  theme(panel.background = element_rect(fill = "white"),
        panel.border = element_rect(linetype = "solid", colour = "black", fill =
        ↪  "transparent",
            size = 0.2), panel.grid.minor = element_blank(), panel.grid.major =
            ↪  element_blank(),
        axis.title.x = element_text(face = "bold", size = 6), axis.title.y =
        ↪  element_text(face = "bold",
            size = 6), axis.ticks = element_line(size = 0.2), axis.text.x =
            ↪  element_text(size = 5,
```

```
            angle = 90, hjust = 1, vjust = 0.5), axis.text.y = element_text(size = 5),
            ↪  legend.text = element_text(size = 5),
        legend.key.size = unit(0.4, "cm"), legend.title = element_text(size = 6))
}
```

```
# Define population number threshold
PopCutOff <- 10

# List pop names above this threshold
PopKeep <- names(table(ind.metrics[, pop_col]) >= PopCutOff)[table(ind.metrics[,
↪  pop_col]) >=
    PopCutOff]
```

This leaves 8 populations to use in the calculation of population genetic summary statistics.

2 Data preparation and filtering on individual read count

Is it necessary to filter on individual read count, or is it okay to filter on the mean locus values routinely provided by DArT in their meta data instead?

Note that the steps involving individual read count reuire

2.1 Prepare data

Note that data were obtained from DArT in 2019-2020, and formats may have changed DArT provides read counts for each individual at each locus. This data is in two-row format, with read counts provided for the reference and SNP allele. The data is similar to the two-row SNP format file, however the cloneIDs do not match between files. Therefore, a bit of data wrangling is needed to compare individual read counts to called genotypes. This step (i.e. exploring individual read counts) requires a fair bit of tailoring to each specific data set, to check the two data sets match properly.

```
## Add a unique column (sequence and SNP poistion) to match loci across data sets Read in
## data as dataframe (instead of genlight) DArT read count data csv (skip top 'no data'
## rows)
ReadCounts <- read.csv(file = paste0(datpath, DartReadCounts), skip =
↪  sum(read.csv(paste0(datpath,
    DartReadCounts), header = FALSE)[, 1] == "*", na.rm = TRUE), header = TRUE,
↪  stringsAsFactors = FALSE)

# DArT 1-row Genotype csv (skips top 'no data' rows)- easier to manipulate as data frame
# versus genlight in this case
Genos_1row <- read.csv(file = paste0(datpath, DartGenos_1Row), na.strings = "-", skip =
↪  sum(read.csv(paste0(datpath,
```

```
    DartGenos_1Row), header = FALSE)[, 1] == "*", na.rm = TRUE), header = TRUE,
↪   stringsAsFactors = FALSE)

# DArT 2-row Genotype csv (skips top 'no data' rows)- easier to manipulate as data frame
# versus genlight in this case
Genos_2row <- read.csv(file = paste0(datpath, DartGenos_2Row), na.strings = "-", skip =
↪   sum(read.csv(paste0(datpath,
    DartGenos_2Row), header = FALSE)[, 1] == "*", na.rm = TRUE), header = TRUE,
↪   stringsAsFactors = FALSE)

# The clone/allele ids do not match between the geno and read count data sets.  Add a
# unique sequence column that combines the sequence with the SNP position.  This will
# ensure that the correct ref sequence is paired with the corresponding SNP sequence when
# there are multiple SNPs per sequence.
ReadCounts <- add_column(ReadCounts, UniqueSeq = paste(ReadCounts$AlleleSequence,
↪   ReadCounts$SnpPosition,
    sep = "_"), .before = 1)
Genos_1row <- add_column(Genos_1row, UniqueSeq = paste(Genos_1row$AlleleSequence,
↪   Genos_1row$SnpPosition,
    sep = "_"), .before = 1)
Genos_2row <- add_column(Genos_2row, UniqueSeq = paste(Genos_2row$AlleleSequence,
↪   Genos_2row$SnpPosition,
    sep = "_"), .before = 1)


## Check that the number of loci matches across data sets Check if these unique sequences
## match between csv files
length(setdiff(ReadCounts$UniqueSeq, Genos_2row$UniqueSeq))
length(setdiff(Genos_2row$UniqueSeq, ReadCounts$UniqueSeq))


# There are some unique allele sequences that are in one data set, but not the other. I
# contacted DArT to ask about this mismatch and got this reply from Jason Carling (Lead,
# Analytical Services):

# 'There will sometimes be relatively small differences in the marker sets present in the
# files which have 1 data column per assay vs those which have 1 data column per sample,
# due to the application of automated marker quality filtering. In these two file types
# the underlying genotype frequencies measured across the markers can be very slightly
# different due to the difference in sample number (reps present vs consensus only). In
# these cases the markers which are not in common across the files will be those which
# are close to the filtering threshold value in one or more metadata parameters used in
# filtering.'

# Therefore, just filter these out.

## Filter out mismatched loci, create a new cloneID that matches across all data sets (so
## don't have to use the entire sequence) and create a column specifying if reference/SNP
## allele (for matching with one-row format)

# Read count data frame: Only keep Unique Sequences that are in both files
ReadCounts.SeqFilt <- ReadCounts[ReadCounts$UniqueSeq %in%
↪   intersect(ReadCounts$UniqueSeq,
```

```
    Genos_2row$UniqueSeq), ]

# 1-row Genotype data frame: Only keep Unique Sequences that are in both files
Genos_1row.SeqFilt <- Genos_1row[Genos_1row$UniqueSeq %in%
↪  intersect(Genos_1row$UniqueSeq,
    ReadCounts$UniqueSeq), ]

# 2-row Genotype data frame: Only keep Unique Sequences that are in both files
Genos_2row.SeqFilt <- Genos_2row[Genos_2row$UniqueSeq %in%
↪  intersect(Genos_2row$UniqueSeq,
    ReadCounts$UniqueSeq), ]

# Create unique cloneID per SNP by appending SNP position Identify Snps versus reference
# alleles Copy this identifier over to read count DF, matching by sequence (for ease of
# comparing DFs)

Genos_2row.SeqFilt <- add_column(Genos_2row.SeqFilt, Locus =
↪  paste(Genos_2row.SeqFilt$CloneID,
    Genos_2row.SeqFilt$SnpPosition, sep = "_"), .before = 1)
Genos_2row.SeqFilt <- add_column(Genos_2row.SeqFilt, SNP.Ref = Genos_2row.SeqFilt$SNP,
↪  .after = "Locus")
Genos_2row.SeqFilt$SNP.Ref[Genos_2row.SeqFilt$SNP.Ref == ""] <- "Ref"
# The SNP column provided by Dart is empty if it's a ref allele, but contains a
# position/base if it's a SNP allele.  Therefore, rename everything that's empty as
# 'Ref', everything else as 'SNP'

Genos_2row.SeqFilt$SNP.Ref[!Genos_2row.SeqFilt$SNP.Ref == "Ref"] <- "SNP"

table(Genos_2row.SeqFilt$SNP.Ref)  # Make sure even numbers of each

# Add unique CloneIDs, and Ref/SNP info (from the 2-row geno DF) to 1-row geno DF
Geno2Geno_Match <- match(Genos_1row.SeqFilt$UniqueSeq, Genos_2row.SeqFilt$UniqueSeq)
Genos_1row.SeqFilt <- add_column(Genos_1row.SeqFilt, Locus =
↪  Genos_2row.SeqFilt$Locus[Geno2Geno_Match],
    .before = 1)
sum(is.na(Genos_1row.SeqFilt$Locus))

# Add unique CloneIDs, and Ref/SNP info (from the 2-row geno DF) to the read count DF
Geno2Count_Match <- match(ReadCounts.SeqFilt$UniqueSeq, Genos_2row.SeqFilt$UniqueSeq)
ReadCounts.SeqFilt <- add_column(ReadCounts.SeqFilt, Locus =
↪  Genos_2row.SeqFilt$Locus[Geno2Count_Match],
    .before = 1)
ReadCounts.SeqFilt <- add_column(ReadCounts.SeqFilt, SNP.Ref =
↪  Genos_2row.SeqFilt$SNP.Ref[Geno2Count_Match],
    .after = "Locus")

table(ReadCounts.SeqFilt$SNP.Ref)  # Make sure even numbers of each (ref vs snp)


## Remove replicate samples by aggregating duplicates and keeping the max read count

# DArT runs 30 technical replicates, which means that there are 30 duplicate samples (per
# plate) in the read count data set. However, there is only one genotype called for each
```

```
# duplicated sample in the SNP data sets. Unsure how DArT comes up with a consensus
# across repeated samples, so I'm going to take the maximum value (i.e. for each row,
# take the maximum read count between the original and the repeat).


# READ COUNT DF

# Remove extra metadata info from read count data frame, order rows by CloneID and
# columns by sample name
LastRC.MD.No <- which(colnames(ReadCounts.SeqFilt) == LastRC.MD.Col)
RC_Rm_MD <- ReadCounts.SeqFilt[, c("Locus", "SNP.Ref",
↪  colnames(ReadCounts.SeqFilt)[(LastRC.MD.No +
    1):ncol(ReadCounts.SeqFilt)])]
RC_order <- RC_Rm_MD[order(RC_Rm_MD$Locus), c(1:2,
↪  order(colnames(RC_Rm_MD)[3:ncol(RC_Rm_MD)]) +
    2)]

# Find column number where sample names start in the 2-row data set (for comparing sample
# names)
LastGeno.MD.No.2 <- which(colnames(Genos_2row.SeqFilt) == LastGeno.MD.Col)

# Find technical replicates (i.e. duplicated samples) Duplicate sample IDs are
# automatically appended with '. + dup number' in R (so that column names aren't
# repeated).  Remove these characters to find duplicated sample IDs
col.rename <- colnames(RC_order)

# Find samples names that don't match between data sets (for example, because they've
# been appended with '.1')
reps <- which(col.rename %in% setdiff(col.rename[3:length(col.rename)],
↪  colnames(Genos_2row.SeqFilt[,
    LastGeno.MD.No.2:ncol(Genos_2row.SeqFilt)])))

# Remove the last two characters and replace these sample names
col.rename[reps] <- substr(col.rename[reps], 1, nchar(col.rename[reps]) - 2)

# Create loop to find the column numbers for identical samples (technical replicates)
# Take the maximum read count value (across columns) for each row and add this aggregate
# column to the end of the data frame

ColNo <- 0
col_ID <- colnames(RC_order)
DupNo <- as.data.frame(table(col.rename), stringsAsFactors = FALSE)
rmCol_Vect <- vector()

for (ColNo in (1:nrow(DupNo) - 1)) {
    Col <- ColNo + 1
    if (DupNo[Col, 2] == 1) {
    } else {
        rmCol_Vect <- c(rmCol_Vect, which(col.rename %in% DupNo[Col, 1]))
        RC_order <- cbind(RC_order, apply(RC_order[, which(col.rename %in% DupNo[Col,
↪  1])],
            1, max, na.rm = TRUE))
        colnames(RC_order)[ncol(RC_order)] <- DupNo[Col, 1]
```

```
    }
}

# Keep only unique column numbers
rmCol_Vect <- unique(rmCol_Vect)

# Remove replicate and original sample from data frame (just keep new max read count
# column), then order sample columns
RC_MaxRep <- RC_order[, -(rmCol_Vect)]
RC_MaxRep <- RC_MaxRep[, c(1:2, order(colnames(RC_MaxRep)[3:ncol(RC_MaxRep)]) + 2)]
colnames(RC_MaxRep)  # Check

# 2-ROW GENO DF

# Order rows by CloneID and columns by sample name
G2_Final <- Genos_2row.SeqFilt[order(Genos_2row.SeqFilt$Locus), c(1:2,
↪  order(colnames(Genos_2row.SeqFilt)[(LastGeno.MD.No.2 +
    1):ncol(Genos_2row.SeqFilt)]) + LastGeno.MD.No.2)]

# Check that order of both data frames match exactly
sum(colnames(G2_Final) == colnames(RC_MaxRep)) == dim(G2_Final)[2]
sum(G2_Final$Locus == RC_MaxRep$Locus) == dim(G2_Final)[1]

# 1-ROW GENO DF
LastGeno1.MD.No <- which(colnames(Genos_1row.SeqFilt) == LastGeno.MD.Col)
G1_Final <- Genos_1row.SeqFilt[order(Genos_1row.SeqFilt$Locus), c(1,
↪  order(colnames(Genos_1row.SeqFilt)[(LastGeno1.MD.No +
    1):ncol(Genos_1row.SeqFilt)]) + LastGeno1.MD.No)]

# Check that SNP numbers add up/sample IDs match
2 * dim(G1_Final)[1] == dim(G2_Final)[1]  # Should be 2x as many SNPs (ref+snp) in 2-row
↪  Geno DF
sum(!colnames(G1_Final) == colnames(G2_Final)[-2])

## Screen out read counts for NA genotypes

# Create a new data frame where read count data for genotypes that haven't been called
# are screened out (i.e. replace count value with NA if same sample/loci is missing in
# SNP data set). Don't include these in the read count summary statistics, instead
# account for these through missing data filters.

RC_Final.Raw <- map2_df(G2_Final, RC_MaxRep, ~ifelse(is.na(.x), .x, .y))
RC_Final.Raw <- as.data.frame(RC_Final.Raw)

## Create sample/loci/metric data frames

# Reshape data, calculate summary statistics and visualise individual read count.

# Make new data frame with total read counts
Tot.Counts <- RC_Final.Raw[, -2]
Tot.Counts <- group_by(Tot.Counts, Locus)
Tot.Counts <- summarise_each(Tot.Counts, sum)
Tot.Counts <- Tot.Counts[order(Tot.Counts$Locus), ]
```

```
# Make new data frame with ref read counts
Ref.Counts <- RC_Final.Raw[RC_Final.Raw$SNP.Ref == "Ref", -2]
Ref.Counts <- Ref.Counts[order(Ref.Counts$Locus), ]

# Make new data frame with SNP read counts
SNP.Counts <- RC_Final.Raw[RC_Final.Raw$SNP.Ref == "SNP", -2]
SNP.Counts <- SNP.Counts[order(SNP.Counts$Locus), ]

# Make a new data frame with locus metrics to append to genlight
loc.metrics <- Genos_1row.SeqFilt[order(Genos_1row.SeqFilt$Locus),
↪  1:which(colnames(Genos_1row.SeqFilt) ==
    LastGeno.MD.Col)]
sum(!loc.metrics$Locus == Tot.Counts$Locus) + sum(!loc.metrics$Locus == Ref.Counts$Locus)
↪  +
    sum(!loc.metrics$Locus == SNP.Counts$Locus)

# Calculate summary metrics across individuals for each locus (total read counts)
loc.metrics$RC_MeanTot <- apply(Tot.Counts[, 2:ncol(Tot.Counts)], 1, mean, na.rm = TRUE)
loc.metrics$RC_MeanRef <- apply(Ref.Counts[, 2:ncol(Ref.Counts)], 1, mean, na.rm = TRUE)
loc.metrics$RC_MeanSNP <- apply(SNP.Counts[, 2:ncol(SNP.Counts)], 1, mean, na.rm = TRUE)

# INDIVIDUAL METRICS

# Reshape each data frame so that 1 row per locus per sample (i.e. from 'wide' to 'long'
# format)
Locus.Samp.TotRC <- gather(data = Tot.Counts, Sample_ID, Total_ReadCount, -Locus)
Locus.Samp.RefRC <- gather(data = Ref.Counts, Sample_ID, Ref_ReadCount, -Locus)
Locus.Samp.SNPRC <- gather(data = SNP.Counts, Sample_ID, SNP_ReadCount, -Locus)
Locus.Samp.Geno <- gather(data = G1_Final, Sample_ID, Genotype, -Locus)

# Create individual level read count data frame
IndRC_DF <- left_join(Locus.Samp.Geno, Locus.Samp.TotRC, by = c("Locus", "Sample_ID"))
IndRC_DF <- left_join(IndRC_DF, Locus.Samp.RefRC, by = c("Locus", "Sample_ID"))
IndRC_DF <- left_join(IndRC_DF, Locus.Samp.SNPRC, by = c("Locus", "Sample_ID"))

# Reshape data for comparing read count by proportion of each genotype
Reads.by.geno <- table(IndRC_DF$Total_ReadCount, IndRC_DF$Genotype, useNA = "no")
colnames(Reads.by.geno) <- c("Ref.Hom", "SNP.Hom", "Het")
Reads.by.geno <- cbind(Reads.by.geno, Total.Called.Genos = rowSums(Reads.by.geno))
Reads.by.geno <- as.data.frame(Reads.by.geno)
Reads.by.geno <- add_column(Reads.by.geno, Total_ReadCount =
↪  as.integer(row.names(Reads.by.geno)),
    .before = 1)
Reads.by.geno <- cbind(Reads.by.geno, Prop.Ref.Hom =
↪  Reads.by.geno$Ref.Hom/Reads.by.geno$Total.Called.Genos,
    Prop.SNP.Hom = Reads.by.geno$SNP.Hom/Reads.by.geno$Total.Called.Genos, Prop.Het =
    ↪  Reads.by.geno$Het/Reads.by.geno$Total.Called.Genos)

# Calculate allele balance (compare read count by looking at proportion of ref reads
# compared to total across hets)

# INDIVIDUAL METRICS
```

```
HetRC <- filter(IndRC_DF, Genotype == 2)
HetRC.T <- HetRC %>%
    group_by(Locus) %>%
    summarise(AverageT = mean(Total_ReadCount))
HetRC.R <- HetRC %>%
    group_by(Locus) %>%
    summarise(AverageR = mean(Ref_ReadCount))
HetRC.Final <- left_join(HetRC.T, HetRC.R, by = "Locus")
HetRC.Final$AB <- HetRC.Final$AverageR/HetRC.Final$AverageT
loc.metrics <- left_join(loc.metrics, HetRC.Final[, c(1, 4)], by = "Locus")

# Reformat data for facet_wrap in ggplot
Reads.by.geno.reshape <- pivot_longer(data = Reads.by.geno[, c("Total_ReadCount",
↪  "Prop.Ref.Hom",
    "Prop.SNP.Hom", "Prop.Het")], cols = c("Prop.Ref.Hom", "Prop.SNP.Hom", "Prop.Het"),
    ↪  names_to = "Prop_Geno")

# Rename variables in plotting order
Reads.by.geno.reshape$Prop_Geno[Reads.by.geno.reshape$Prop_Geno == "Prop.Ref.Hom"] <-
↪  "a_Prop.Ref.Hom"
Reads.by.geno.reshape$Prop_Geno[Reads.by.geno.reshape$Prop_Geno == "Prop.SNP.Hom"] <-
↪  "b_Prop.Ref.Hom"
Reads.by.geno.reshape$Prop_Geno[Reads.by.geno.reshape$Prop_Geno == "Prop.Het"] <-
↪  "c_Prop.Ref.Het"

PropGeno.labs <- c("Ref Homozygotes", "SNP Homozygotes", "Heterozygotes")
names(PropGeno.labs) <- unique(Reads.by.geno.reshape$Prop_Geno)

RC.geno.Raw <- ggplot(Reads.by.geno.reshape, aes(x = Total_ReadCount, y = value, colour =
↪  as.factor(Prop_Geno))) +
    geom_point(size = 0.5, na.rm = T) + facet_wrap(~Prop_Geno, nrow = 3, dir = "h",
    ↪  strip.position = "right",
    labeller = labeller(Prop_Geno = PropGeno.labs)) + labs(x = "\nTotal read count", y =
    ↪  "Proportion of called genotypes (raw)\n") +
    scale_x_continuous(breaks = seq(0, max(Reads.by.geno.reshape$Total_ReadCount), 50)) +
    ↪  scale_colour_brewer(palette = "Dark2") +
    theme(panel.background = element_rect(fill = "white"), panel.border =
    ↪  element_rect(linetype = "solid",
        colour = "black", fill = "transparent", size = 0.2), panel.grid.minor =
        ↪  element_blank(),
        panel.grid.major = element_blank(), axis.title.x = element_text(face = "bold",
        ↪  size = 6),
        axis.title.y = element_text(face = "bold", size = 6), axis.ticks =
        ↪  element_line(size = 0.2),
        axis.text.x = element_text(size = 5, angle = 90, hjust = 1, vjust = 0.5),
        ↪  axis.text.y = element_text(size = 5),
        strip.text = element_text(face = "bold", size = 6), strip.background =
        ↪  element_rect(linetype = "solid",
            colour = "black", fill = "transparent", size = 0.2), legend.position =
            ↪  "none")

loc.metrics.raw <- loc.metrics
```

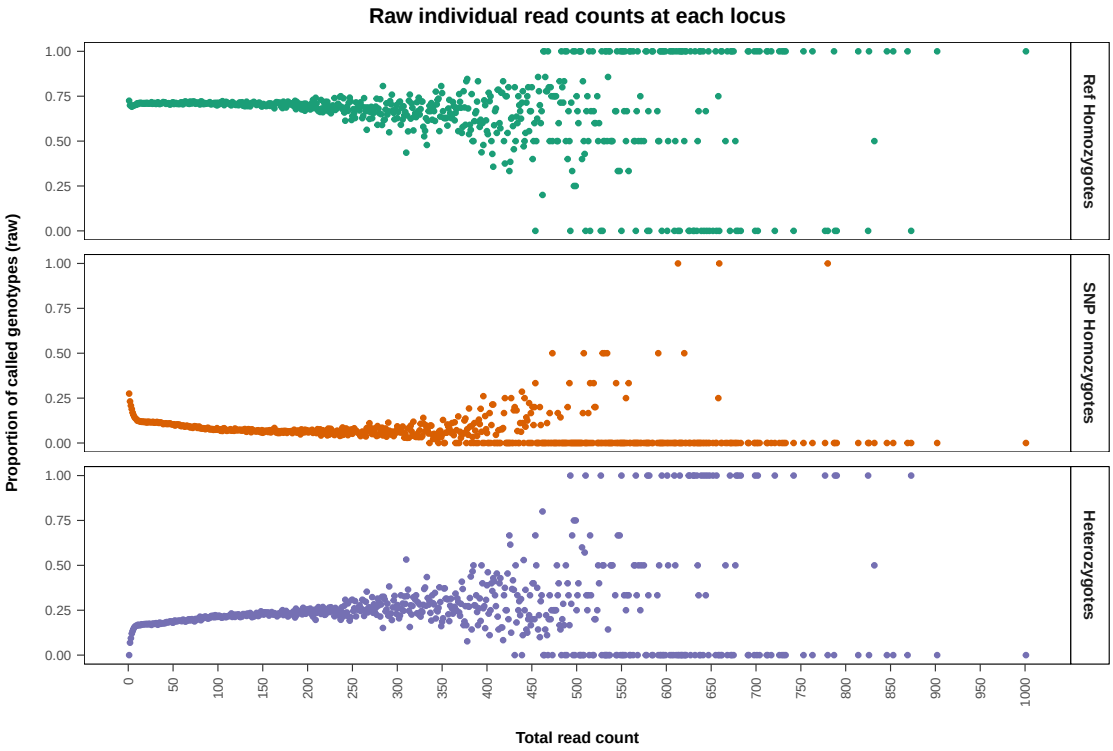

Figure 2: Individual read count for each locus, by genotype.

2.2 Decide on individual read count threshold

```
# DEFINE MIN MAX VALUES
MinRC <- 4
MaxRC <- 300
```

Inspect Figure 2 to decide at which min/max values the genotypes are no longer strongly impacted by read count, then filter individual genotypes on these values. It might be expected that the proportion of each genotype should be stable, regardless of read count. However, at the extremes, low read counts increase the likelihood that one of the alleles hasn't been sequenced (likely the minor allele, which is usually the SNP), and high read counts might indicate that there are multiple copies of this sequence spread throughout the genome (i.e. paralogues).

In this data set, there is a strong relationship between individual read count and the proportion of each called genotype at the lower and upper extremes. At the lower end of the range, heterozygotes are under called, while SNP homozygotes are over called. After about 350 reads, the genotypes become fixed for the reference or the SNP allele (indicating that these may represent paralogous regions).

Here, a minimum of 4 and a maximum of 300 have been chosen.

```
# Create a new data frame where count data for genotypes that haven't been called are
# screened out (i.e. replace count value with NA if same sample/loci is missing in geno
# file)

Tot.Counts <- as.data.frame(RC_Final.Raw[, -2] %>%
    group_by(Locus) %>%
    summarise_each(sum))
rownames(Tot.Counts) <- Tot.Counts[, 1]
Tot.Counts <- Tot.Counts[, -1]

G1_Final_screen <- as.matrix(G1_Final[, -1])
rownames(G1_Final_screen) <- G1_Final$Locus

# Check that data frames match exactly
sum(!rownames(Tot.Counts) == rownames(G1_Final_screen))
sum(!colnames(Tot.Counts) == colnames(G1_Final_screen))

# Screen out genotypes above/below threshold
Geno_RC.Filt <- ifelse(Tot.Counts < MinRC | Tot.Counts > MaxRC, NA, G1_Final_screen)

# Screen out read counts using genotype file above (screen out counts for genotypes = NA)
# Cant just do this by removing counts below/above thresholds because that will replace
# read counts of 0 with NA Read counts of 0 are important, because there might be 0 reads
# of the SNP, but 10 reads of the reference Therefore, have to use a slightly convoluted
# workaround

# Save screened/filtered genotype file as data frame
Geno_RC.Filt <- as.data.frame(Geno_RC.Filt)

# Separate read count data frame into ref and SNPs
RC_MaxRep.SNP <- RC_MaxRep[RC_MaxRep$SNP.Ref == "SNP", -c(1, 2)]
RC_MaxRep.Ref <- RC_MaxRep[RC_MaxRep$SNP.Ref == "Ref", -c(1, 2)]

row.names(RC_MaxRep.SNP) <- RC_MaxRep[RC_MaxRep$SNP.Ref == "SNP", 1]
row.names(RC_MaxRep.Ref) <- RC_MaxRep[RC_MaxRep$SNP.Ref == "Ref", 1]

# Make sure loci match/data frames are identical
sum(!row.names(RC_MaxRep.SNP) == row.names(RC_MaxRep.Ref))
sum(!row.names(RC_MaxRep.SNP) == row.names(Geno_RC.Filt))
sum(!row.names(RC_MaxRep.Ref) == row.names(Geno_RC.Filt))

sum(!colnames(RC_MaxRep.SNP) == colnames(RC_MaxRep.Ref))
sum(!colnames(RC_MaxRep.SNP) == colnames(Geno_RC.Filt))
sum(!colnames(RC_MaxRep.Ref) == colnames(Geno_RC.Filt))

# Create a new data frame where count data for genotypes that were screened out is
# replaced with NA
RC_Final_RefFilt <- map2_df(Geno_RC.Filt, RC_MaxRep.Ref, ~ifelse(is.na(.x), .x, .y))
RC_Final_SNPFilt <- map2_df(Geno_RC.Filt, RC_MaxRep.SNP, ~ifelse(is.na(.x), .x, .y))

RC_Final_RefFilt <- cbind(RC_MaxRep[RC_MaxRep$SNP.Ref == "Ref", 1:2], RC_Final_RefFilt)
RC_Final_SNPFilt <- cbind(RC_MaxRep[RC_MaxRep$SNP.Ref == "SNP", 1:2], RC_Final_SNPFilt)
```

```
# Merge data frames
RC_Final_RCFilt <- rbind(RC_Final_RefFilt, RC_Final_SNPFilt)
RC_Final_RCFilt <- RC_Final_RCFilt[order(RC_Final_RCFilt$Locus), ]

# Get Geno DF in correct format for function
Geno_RC.Filt.loc <- as.data.frame(Geno_RC.Filt)
Geno_RC.Filt.loc$Locus <- rownames(Geno_RC.Filt.loc)


# Reshape data, calculate summary statistics and visualise individual read count.

# Make new data frame with total read counts
Tot.Counts <- RC_Final_RCFilt[, -2]
Tot.Counts <- group_by(Tot.Counts, Locus)
Tot.Counts <- summarise_each(Tot.Counts, sum)
Tot.Counts <- Tot.Counts[order(Tot.Counts$Locus), ]

# Make new data frame with ref read counts
Ref.Counts <- RC_Final_RCFilt[RC_Final_RCFilt$SNP.Ref == "Ref", -2]
Ref.Counts <- Ref.Counts[order(Ref.Counts$Locus), ]

# Make new data frame with SNP read counts
SNP.Counts <- RC_Final_RCFilt[RC_Final_RCFilt$SNP.Ref == "SNP", -2]
SNP.Counts <- SNP.Counts[order(SNP.Counts$Locus), ]

# Make a new data frame with locus metrics to append to genlight
loc.metrics <- Genos_1row.SeqFilt[order(Genos_1row.SeqFilt$Locus),
↪  1:which(colnames(Genos_1row.SeqFilt) ==
    LastGeno.MD.Col)]

# Check that order of loci matches among data frames
sum(!loc.metrics$Locus == Tot.Counts$Locus) + sum(!loc.metrics$Locus == Ref.Counts$Locus)
↪  +
    sum(!loc.metrics$Locus == SNP.Counts$Locus)

# Calculate summary metrics across individuals for each locus (total read counts)
loc.metrics$RC_MeanTot <- apply(Tot.Counts[, 2:ncol(Tot.Counts)], 1, mean, na.rm = TRUE)
loc.metrics$RC_MeanRef <- apply(Ref.Counts[, 2:ncol(Ref.Counts)], 1, mean, na.rm = TRUE)
loc.metrics$RC_MeanSNP <- apply(SNP.Counts[, 2:ncol(SNP.Counts)], 1, mean, na.rm = TRUE)

# INDIVIDUAL METRICS

# Reshape each data frame so that 1 row per locus per sample (i.e. from 'wide' to 'long'
# format)
Locus.Samp.TotRC <- gather(data = Tot.Counts, Sample_ID, Total_ReadCount, -Locus)
Locus.Samp.RefRC <- gather(data = Ref.Counts, Sample_ID, Ref_ReadCount, -Locus)
Locus.Samp.SNPRC <- gather(data = SNP.Counts, Sample_ID, SNP_ReadCount, -Locus)
Locus.Samp.Geno <- gather(data = Geno_RC.Filt.loc, Sample_ID, Genotype, -Locus)

# Create individual level read count data frame
IndRC_DF <- left_join(Locus.Samp.Geno, Locus.Samp.TotRC, by = c("Locus", "Sample_ID"))
IndRC_DF <- left_join(IndRC_DF, Locus.Samp.RefRC, by = c("Locus", "Sample_ID"))
IndRC_DF <- left_join(IndRC_DF, Locus.Samp.SNPRC, by = c("Locus", "Sample_ID"))
```

```
# Reshape data for comparing read count by proportion of each genotype
Reads.by.geno <- table(IndRC_DF$Total_ReadCount, IndRC_DF$Genotype, useNA = "no")
colnames(Reads.by.geno) <- c("Ref.Hom", "SNP.Hom", "Het")
Reads.by.geno <- cbind(Reads.by.geno, Total.Called.Genos = rowSums(Reads.by.geno))
Reads.by.geno <- as.data.frame(Reads.by.geno)
Reads.by.geno <- add_column(Reads.by.geno, Total_ReadCount =
↪  as.integer(row.names(Reads.by.geno)),
    .before = 1)
Reads.by.geno <- cbind(Reads.by.geno, Prop.Ref.Hom =
↪  Reads.by.geno$Ref.Hom/Reads.by.geno$Total.Called.Genos,
    Prop.SNP.Hom = Reads.by.geno$SNP.Hom/Reads.by.geno$Total.Called.Genos, Prop.Het =
    ↪  Reads.by.geno$Het/Reads.by.geno$Total.Called.Genos)

# Calculate allele balance (compare read count by looking at proportion of ref reads
# compared to total across hets)

# INDIVIDUAL METRICS

HetRC <- filter(IndRC_DF, Genotype == 2)
HetRC.T <- HetRC %>%
    group_by(Locus) %>%
    summarise(AverageT = mean(Total_ReadCount))
HetRC.R <- HetRC %>%
    group_by(Locus) %>%
    summarise(AverageR = mean(Ref_ReadCount))
HetRC.Final <- left_join(HetRC.T, HetRC.R, by = "Locus")
HetRC.Final$AB <- HetRC.Final$AverageR/HetRC.Final$AverageT
loc.metrics <- left_join(loc.metrics, HetRC.Final[, c(1, 4)], by = "Locus")

# Reformat data for facet_wrap in ggplot
Reads.by.geno.reshape <- pivot_longer(data = Reads.by.geno[, c("Total_ReadCount",
↪  "Prop.Ref.Hom",
    "Prop.SNP.Hom", "Prop.Het")], cols = c("Prop.Ref.Hom", "Prop.SNP.Hom", "Prop.Het"),
    ↪  names_to = "Prop_Geno")

# Rename variables in plotting order
Reads.by.geno.reshape$Prop_Geno[Reads.by.geno.reshape$Prop_Geno == "Prop.Ref.Hom"] <-
↪  "a_Prop.Ref.Hom"
Reads.by.geno.reshape$Prop_Geno[Reads.by.geno.reshape$Prop_Geno == "Prop.SNP.Hom"] <-
↪  "b_Prop.Ref.Hom"
Reads.by.geno.reshape$Prop_Geno[Reads.by.geno.reshape$Prop_Geno == "Prop.Het"] <-
↪  "c_Prop.Ref.Het"

PropGeno.labs <- c("Ref Homozygotes", "SNP Homozygotes", "Heterozygotes")
names(PropGeno.labs) <- unique(Reads.by.geno.reshape$Prop_Geno)

RC.geno.Filt <- ggplot(Reads.by.geno.reshape, aes(x = Total_ReadCount, y = value, colour
↪  = as.factor(Prop_Geno))) +
    geom_point(size = 0.5, na.rm = T) + facet_wrap(~Prop_Geno, nrow = 3, dir = "h",
    ↪  strip.position = "right",
    labeller = labeller(Prop_Geno = PropGeno.labs)) + labs(x = "\nTotal read count", y =
    ↪  "Proportion of called genotypes (filtered)\n") +
```

```
    scale_x_continuous(breaks = seq(0, max(Reads.by.geno.reshape$Total_ReadCount), 50)) +
    ↪  scale_colour_brewer(palette = "Dark2") +
    theme(panel.background = element_rect(fill = "white"), panel.border =
    ↪  element_rect(linetype = "solid",
        colour = "black", fill = "transparent", size = 0.2), panel.grid.minor =
        ↪  element_blank(),
        panel.grid.major = element_blank(), axis.title.x = element_text(face = "bold",
        ↪  size = 6),
        axis.title.y = element_text(face = "bold", size = 6), axis.ticks =
        ↪  element_line(size = 0.2),
        axis.text.x = element_text(size = 5, angle = 90, hjust = 1, vjust = 0.5),
        ↪  axis.text.y = element_text(size = 5),
        strip.text = element_text(face = "bold", size = 6), strip.background =
        ↪  element_rect(linetype = "solid",
            colour = "black", fill = "transparent", size = 0.2), legend.position =
            ↪  "none")

loc.metrics.filt <- loc.metrics
```

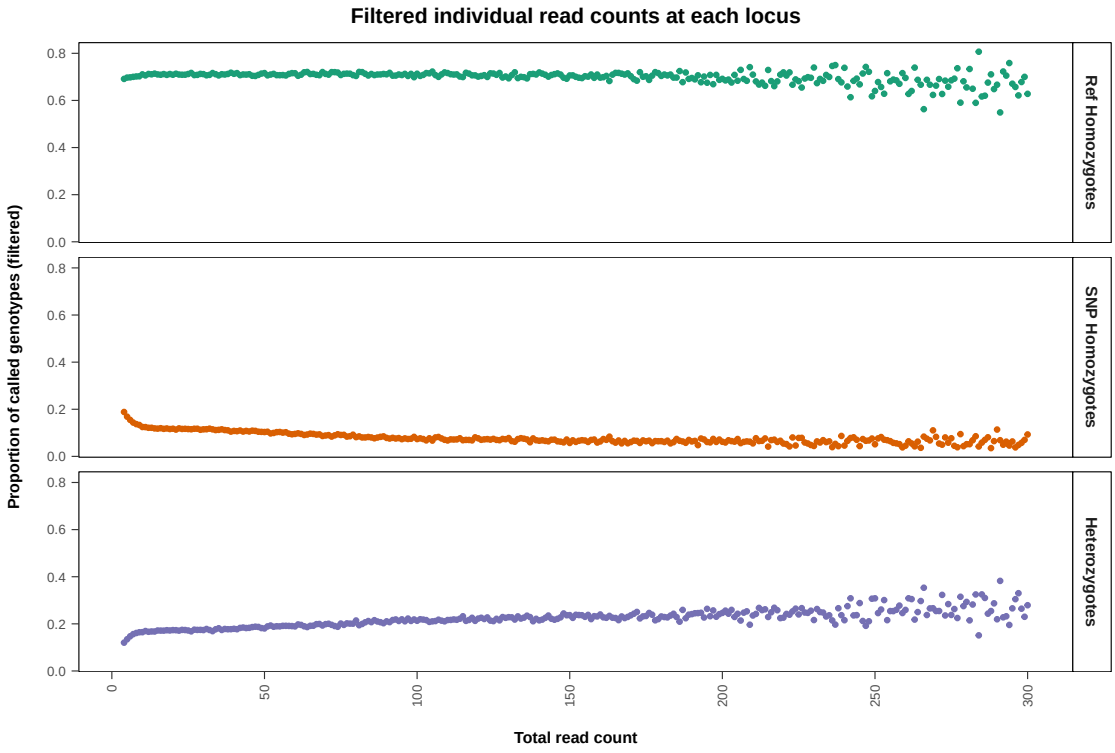

Figure 3: Individual read count for each locus, by genotype after filtering

Figure 3 shows that individual calls that were most strongly impacted by read count have been removed. The stability of the called genotypes does start to dissolve at around 200-300 reads. However, there is a trade off between removing the effect of read count and introducing too much missing data; i.e. be pragmatic.

2.3 Create genlight objects for raw and filtered data sets

```
# Raw data set

# Recode Genotypes in standard format (i.e. not Dart format)
G1_Final_screen[G1_Final_screen == 1] <- 9
G1_Final_screen[G1_Final_screen == 2] <- 1
G1_Final_screen[G1_Final_screen == 9] <- 2

# Transpose so in correct format to load as genlight
G1_Final.t <- t(G1_Final_screen)

# Create new genlight with genotypes
gl.RC.raw <- new("genlight", G1_Final.t)

# Check that locus order matches between loc.metrics and new genlight so can append
sum(!gl.RC.raw@loc.names == loc.metrics.filt$Locus)
gl.RC.raw@other$loc.metrics <- loc.metrics.filt

# Match individual metrics to genlight
ind.metrics <- ind.metrics[match(gl.RC.raw@ind.names, rownames(ind.metrics)), ]
sum(!rownames(ind.metrics) == gl.RC.raw@ind.names)

# Check that individual order matches between meta data and new genlight so can append
sum(!rownames(ind.metrics) == gl.RC.raw@ind.names)

# Append to ind.metrics
gl.RC.raw@other$ind.metrics <- ind.metrics
gl.RC.raw@other$latlong <- ind.metrics[, 2:3]

# Fill in variables to match dartR's genlight
pop(gl.RC.raw) <- gl.RC.raw@other$ind.metrics[, pop_col]
gl.RC.raw@loc.all <- gsub(".*:", "", gsub(">", "/", gl.RC.raw@other$loc.metrics$SNP))
if (BLAST == "Y") {
    gl.RC.raw@chromosome <- as.factor(gl.RC.raw@other$loc.metrics[, ContigCol_dart])
    gl.RC.raw@position <- gl.RC.raw@other$loc.metrics[, ContigPos_dart]
}
gl.RC.raw@ploidy <- as.integer(rep(2, nInd(gl.RC.raw)))

# Recalculate other metrics (like call rate)
recalc.flags <- c("AvgPIC", "OneRatioRef", "OneRatioSnp", "PICRef", "PICSnp", "CallRate",
↪  "maf",
    "FreqHets", "FreqHomRef", "FreqHomSnp", "monomorphs", "OneRatio", "PIC")
gl.RC.raw@other$loc.metrics.flags <- data.frame(matrix(TRUE, nrow = 1, ncol =
↪  length(recalc.flags)))
names(gl.RC.raw@other$loc.metrics.flags) <- recalc.flags
gl.RC.raw <- gl.recalc.metrics(gl.RC.raw, mono.rm = FALSE)


# Filtered data set

# Recode Genotypes in standard format (i.e. not Dart format)
```

```
Geno_RC.Filt[Geno_RC.Filt == 1] <- 9
Geno_RC.Filt[Geno_RC.Filt == 2] <- 1
Geno_RC.Filt[Geno_RC.Filt == 9] <- 2

# Transpose so in correct format to load as genlight
Geno_RC.Filt.t <- t(Geno_RC.Filt)

# Create new genlight with filtered genotypes
gl.RC.filt <- new("genlight", Geno_RC.Filt.t)

# Check that locus order matches between loc.metrics and new genlight so can append
sum(!gl.RC.filt@loc.names == loc.metrics.filt$Locus)
gl.RC.filt@other$loc.metrics <- loc.metrics.filt

# Read in individual metadata
ind.metrics <- read.csv(paste0(datpath, Ind_Metadat), row.names = 1)
ind.metrics <- ind.metrics[match(gl.RC.filt@ind.names, rownames(ind.metrics)), ]
sum(!rownames(ind.metrics) == gl.RC.filt@ind.names)

# Check that individual order matches between meta data and new genlight so can append
sum(!rownames(ind.metrics) == gl.RC.filt@ind.names)

# Append to ind.metrics
gl.RC.filt@other$ind.metrics <- ind.metrics
gl.RC.filt@other$latlong <- ind.metrics[, 2:3]

# Fill in variables to match dartR's genlight
pop(gl.RC.filt) <- gl.RC.filt@other$ind.metrics[, pop_col]
gl.RC.filt@loc.all <- gsub(".*:", "", gsub(">", "/", gl.RC.filt@other$loc.metrics$SNP))
if (BLAST == "Y") {
    gl.RC.raw@chromosome <- as.factor(gl.RC.raw@other$loc.metrics[, ContigCol_dart])
    gl.RC.raw@position <- gl.RC.raw@other$loc.metrics[, ContigPos_dart]
}
gl.RC.filt@ploidy <- as.integer(rep(2, nInd(gl.RC.filt)))

# Recalculate other metrics (like call rate)
gl.RC.filt@other$loc.metrics.flags <- data.frame(matrix(TRUE, nrow = 1, ncol =
↪  length(recalc.flags)))
names(gl.RC.filt@other$loc.metrics.flags) <- recalc.flags
gl.RC.filt <- gl.recalc.metrics(gl.RC.filt)
```

2.4 Create function to calculate population genetics summary statistics

For each locus, the following function calculates:

- Allele frequencies (total and within populations)
- Mean observed heterozygosity (within populations)
- Mean expected heterozygosity (HS - over populations)

- HT (He over total using mean allele frequencies over pops)
- FIS
- FIT
- FST
- The proportion of populations significantly out of HWE (using a Bonferroni correction)

Note that calculations were based on GenAlEx formulae (Peakall & Smouse, 2006, 2012)

A genlight object is returned, with these metrics appended to the loc.metrics slot. The population-level metrics are only calculated for the populations specified above.

```
PopGenMetrics <- function(gl, pops) {

    # Calculate SNP/ref allele frequency over total
    gl@other$loc.metrics$AlleleFreq1 <- gl.alf(gl)[, 1]
    gl@other$loc.metrics$AlleleFreq2 <- gl.alf(gl)[, 2]

    # Calculate sample number (excluding missing data) for each locus, for pops above or
    # equal to pop cutoff threshold
    SampleNo <- aggregate(gl[gl@pop %in% pops], list(gl[gl@pop %in% pops]@pop),
↪   function(x) {
        sum(!is.na(x))
    })

    # count number of heterozygotes (excluding missing data) for each locus, for pops
    # above or equal to pop cutoff threshold
    HetCount <- aggregate(gl[gl@pop %in% pops], list(gl[gl@pop %in% pops]@pop),
↪   function(x) {
        sum(x[x == "1"], na.rm = TRUE)
    })

    # Calculate allele frequencies (excluding missing data) for each locus, for pops
    # above or equal to pop cutoff threshold and append to genlight loc.metrics
    PopAlleleFreq1 <- aggregate(gl[gl@pop %in% pops], list(gl[gl@pop %in% pops]@pop),
↪   function(x) {
        gl.alf(x)[, 1]
    })

    PopAlleleFreq2 <- aggregate(gl[gl@pop %in% pops], list(gl[gl@pop %in% pops]@pop),
↪   function(x) {
        gl.alf(x)[, 2]
    })

    # Calculate observed and expected heterozygosity (excluding missing data) for each
    # locus, for pops above or equal to pop cutoff threshold
    PopHo <- HetCount[, -1]/SampleNo[, -1]
    PopHe <- 1 - ((PopAlleleFreq1[, -1]^2) + (PopAlleleFreq2[, -1]^2))

    # Calculate HS (Mean He over pops) and append to locus metrics in genlight object
    gl$other$loc.metrics$HS <- colMeans(PopHe, na.rm = TRUE)

    # Calculate mean Ho over pops and append to locus metrics in genlight object
    gl$other$loc.metrics$meanHo <- colMeans(PopHo, na.rm = TRUE)
```

```
    # Calculate HT (He over total using mean allele freq over pops) and append to locus
    # metrics in genlight object First append mean allele frequencies over pops to
    # genlight loc.metrics
    gl$other$loc.metrics$PopAlleleFreq1 <- colMeans(PopAlleleFreq1[, -1])^2
    gl$other$loc.metrics$PopAlleleFreq2 <- colMeans(PopAlleleFreq2[, -1])^2

    # Then replace NaN with 0 in pop allele frequency df, so matches the way that Genalex
    # calculates HT
    is.nan.df <- function(x) {
        do.call(cbind, lapply(x, is.nan))
    }

    PopAlleleFreq1[is.nan.df(PopAlleleFreq1)] <- 0
    PopAlleleFreq2[is.nan.df(PopAlleleFreq2)] <- 0

    gl$other$loc.metrics$HT <- 1 - ((colMeans(PopAlleleFreq1[, -1])^2) +
↪   (colMeans(PopAlleleFreq2[,
        -1])^2))

    # Calculate FIS and append to locus metrics in genlight object
    gl$other$loc.metrics$FIS <- 1 - (gl$other$loc.metrics$meanHo/gl$other$loc.metrics$HS)

    # Calculate FIT and append to locus metrics in genlight object
    gl$other$loc.metrics$FIT <- (gl$other$loc.metrics$HT -
↪   gl$other$loc.metrics$meanHo)/gl$other$loc.metrics$HT

    # Calculate FST and append to locus metrics in genlight object
    gl$other$loc.metrics$FST <- (gl$other$loc.metrics$HT -
↪   gl$other$loc.metrics$HS)/gl$other$loc.metrics$HT

    # Calculate HWE (with Bonferroni Corrected significance) and append proportion of
    # pops sig out of HWE to locus metrics in genlight object Calculate HWE with
    # Bonferroni correction for multiple testing usng dartR function only over pops with
    # sample size over threshold
    HWE_Raw <- gl.report.hwe(x = gl[gl@pop %in% pops, ], method_sig = "ChiSquare",
↪   plot.out = FALSE,
        multi_comp = TRUE, multi_comp_method = "bonferroni")
    HWE_Raw$Locus <- as.character(HWE_Raw$Locus)

    # Replace 'ns' (non-significant) with a zero, and everything else (i.e. significant
    # loci '*') with a one. Note that I am using the bonferroni significance, rather than
    # just the 0.05 significance which is why there are ns results.
    HWE_Raw$BonSig_1.0 <- ifelse(HWE_Raw$Sig.adj == "ns", 0, 1)

    # Calculate the proportion of populations where that locus is out of HWE
    HWE.sig.pop.count <- HWE_Raw %>%
        group_by(Locus) %>%
        summarise(PropPopsSig.HWE = sum(BonSig_1.0)/length(pops))

    # Create a new data frame with all loci (rather than just the subset out of HWE)
    Locus_HWE <- data.frame(Locus = locNames(gl))
    Locus_HWE$Locus <- as.character(Locus_HWE$Locus)
```

```
    Locus_HWE <- left_join(Locus_HWE, HWE.sig.pop.count, by = "Locus")

    # Replace NA (i.e. those that were not out of HWE) with a 0
    Locus_HWE$PropPopsSig.HWE[is.na(Locus_HWE$PropPopsSig.HWE)] <- 0

    # Check locus order matches so can append to genlight
    sum(!Locus_HWE$Locus == locNames(gl))

    # Append to genlight
    gl@other$loc.metrics$PropPopsSig.HWE <- Locus_HWE$PropPopsSig.HWE

    return(gl)

}
```

2.5 Calculate summary stats for raw versus filtered SNP data sets

```
gl.RC.raw.PG <- PopGenMetrics(gl = gl.RC.raw, pops = PopKeep)
gl.RC.filt.PG <- PopGenMetrics(gl = gl.RC.filt, pops = PopKeep)
```

How does individual read count impact population genetics statistics (across loci)?

Note that mean read count and call rate thresholds were based on default values purely for exploring the impact of different filtering strategies. Real thresholds will be assigned later during the filtering steps.

Visualise the different data sets:
1) Raw SNPs vs. filtered on individual read count (min= 4, max= 300).
2) Filtered on mean total read count (min= 20, max= 200) vs. filtered on individual read count (min= 4, max= 300).
3) Filtered on mean total read count and call rate (min= 20, max= 200, CR = 90%) vs. filtered on individual read count and call rate (min= 4, max= 300, CR = 90%).

Visualise how the distribution of population genetics metrics change with different read count filters.

Also visualise allele balance, the proportion of reference reads compared to total reads across heterozygotes (expected = 0.5), as large deviations may indicate false heterozygotes due to coverage effects, multilocus contigs or other artifacts (O'Leary, Puritz, Willis, Hollenbeck, & Portnoy, 2018).

Figure 4 demonstrates that filtering on mean read count is roughly equivalent to filtering on individual read count, when combined with a call rate filter (in terms of population genetic metrics).

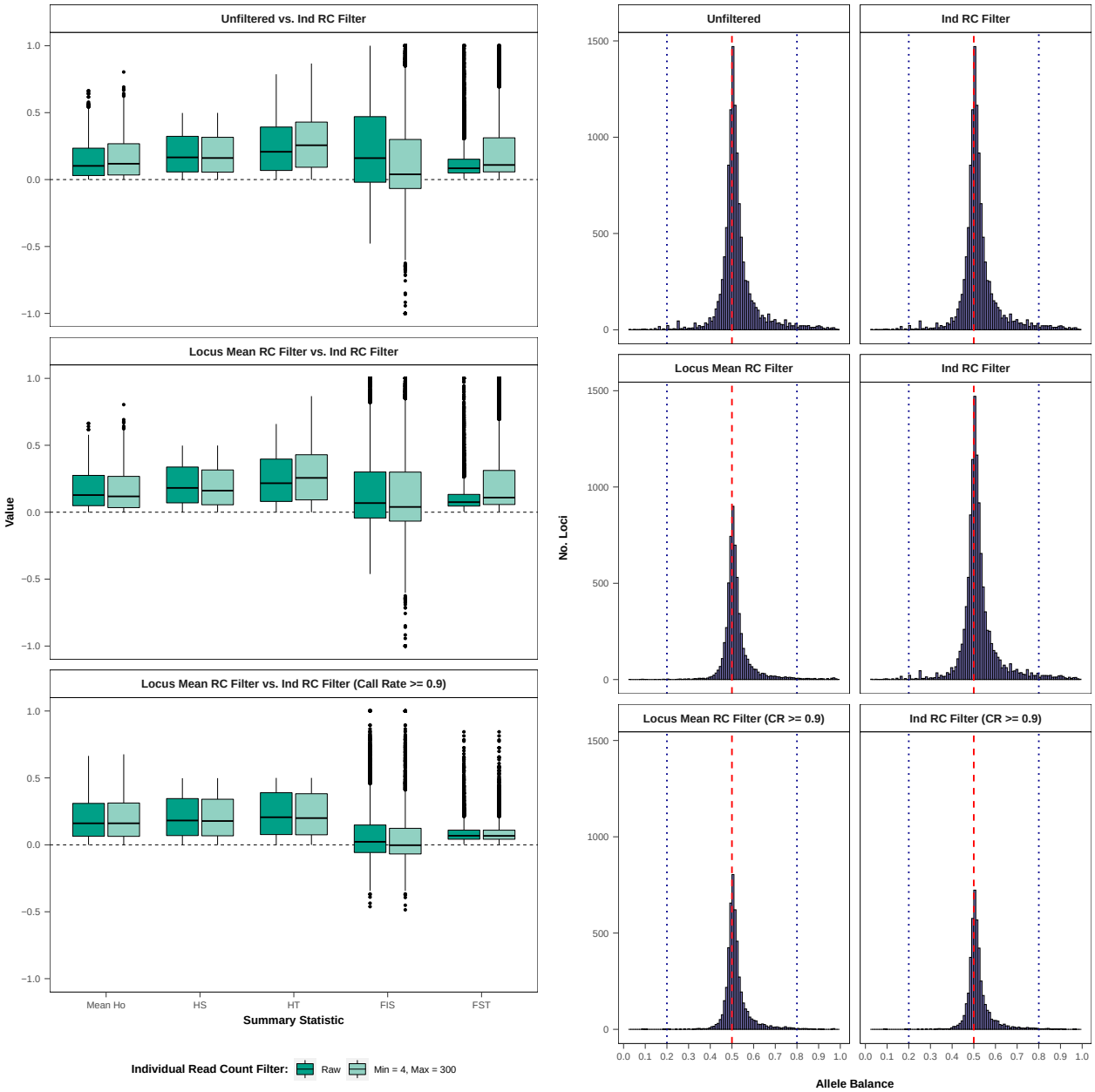

Figure 4: Read count summary statistics (individual and mean over loci).

3 Remove failed individuals

After exploring read count, go through the visualisation/filtering process. The first step involves removing individuals that have failed, i.e. individuals that have such high levels of missing data that they will lower the quality of the entire data set if kept in. Retain as many individuals as possible, as they may be usable after downstream locus filtering. However, it's important to remove any individuals that will have too much missing data at the start, because metrics will need to be recalculated every time samples are removed.

3.1 Visualise missing data for individuals

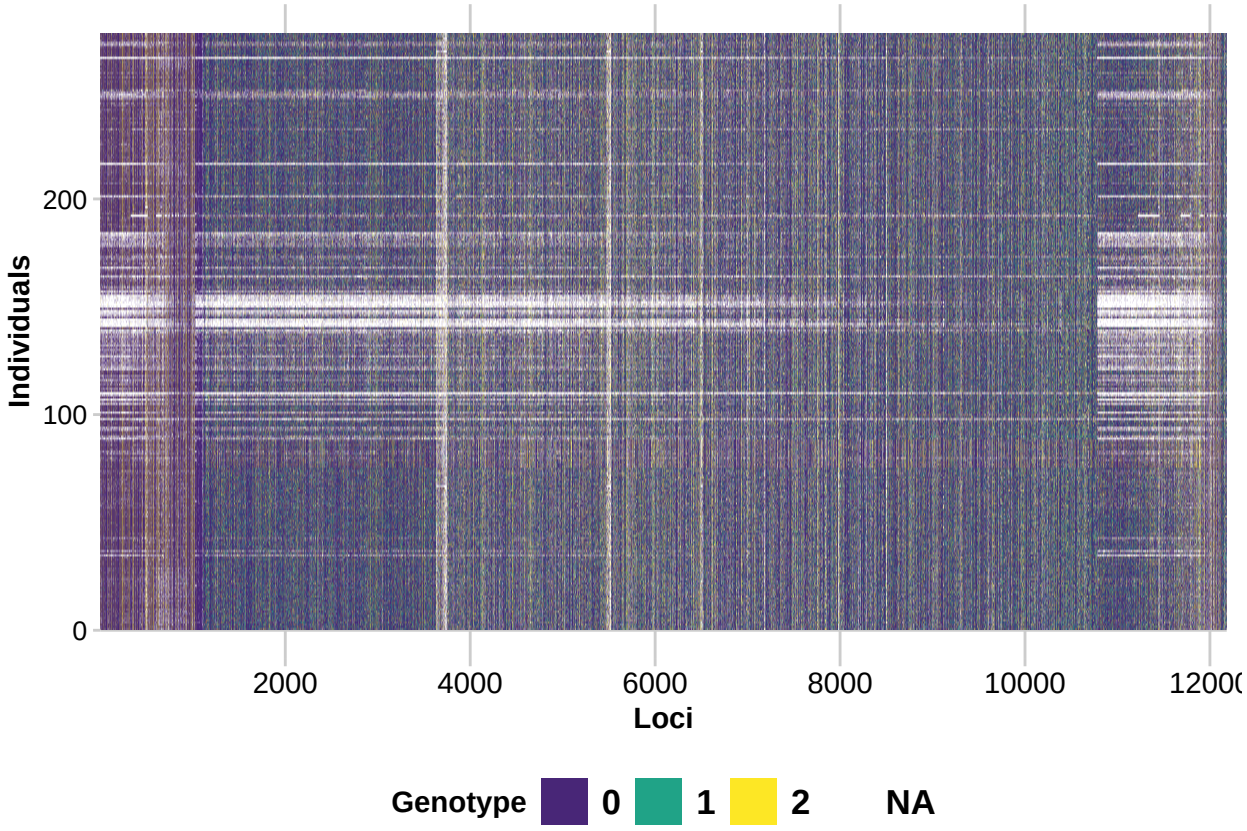

Figure 5: Raw smear plot of individual genotypes across all loci.

Inspect Figure 5 and Figure 6 to decide if any individuals should be removed initially, by deciding on an individual missing data threshold. The aim here is just to remove any samples that are obvious mis-ids (for example, a different species where only a small number of loci have amplified that are obviously fixed for a different allele) or samples that have mainly failed (where there will never be enough loci for them to be usable). Use the map to decide if any individuals are particularly important (i.e. they are the only sample

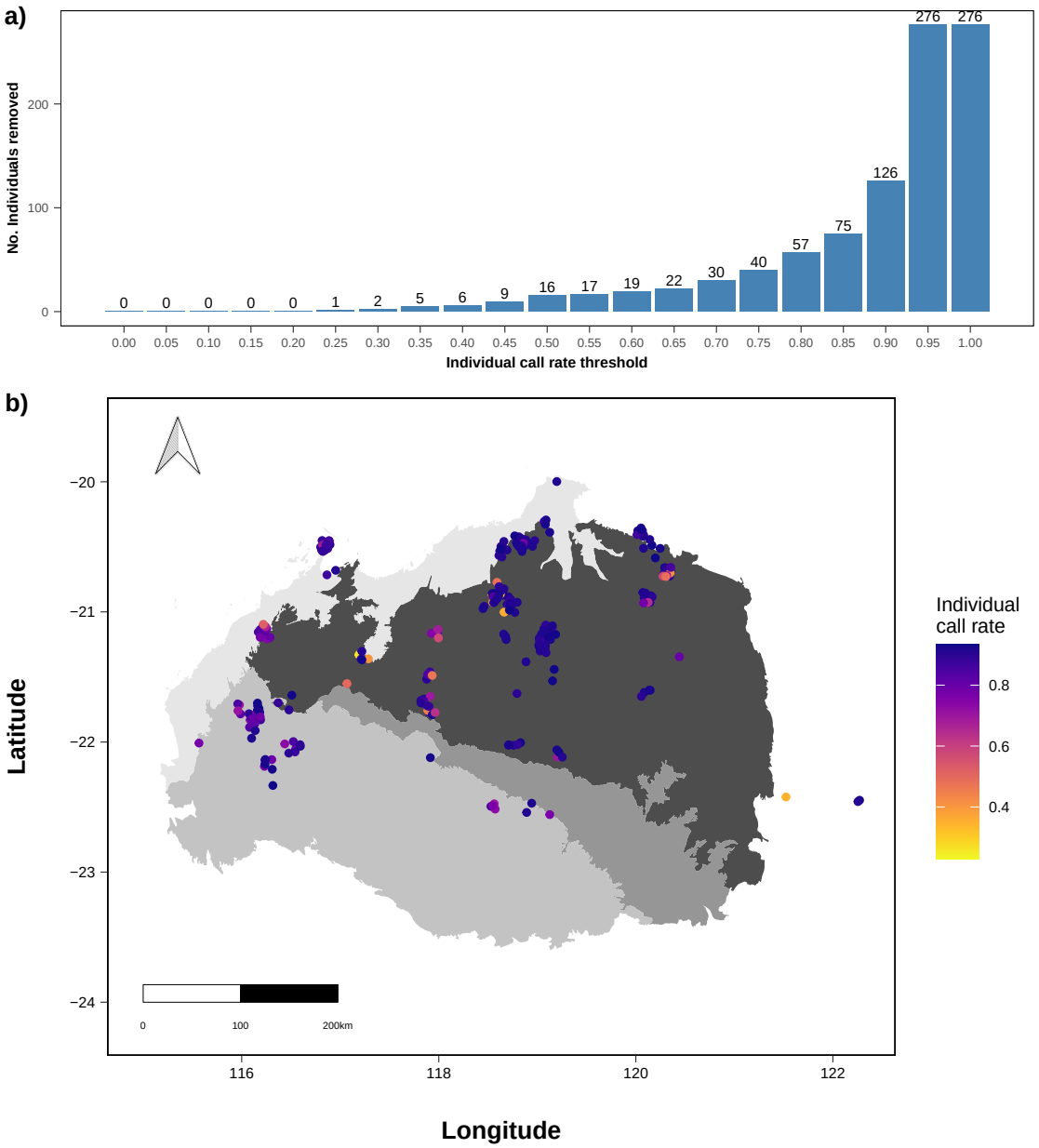

Figure 6: Individual call rate. a) The number of individuals removed at different thresholds, b) Map of sample locations with individuals coloured by call rate.

in a priority location) so that the call rate threshold can be tailored to retain these samples if possible.

3.2 Filter failed individuals from data set

```
IndCallRate_Cutoff <- 0.64

## Remove failed samples from SNP data frame and recalculate metadata
gl.RC.DropInds <- gl.filter.callrate(x = gl.RC.raw, method = "ind", recalc = TRUE,
↪  mono.rm = FALSE,
    threshold = IndCallRate_Cutoff, plot.out = FALSE)

# Use a default locus filtering value to check if any samples drop below an acceptable
# level of missing data Carry out the locus filtering properly later on. However, be sure
# to get the individual filtering right here so that the population genetic summary
# statistics only need to be calculated once (because they take a long time and have to
# be recalculated every time individuals are removed)
gl.checkCR <- gl.filter.callrate(gl.RC.DropInds, method = "loc", threshold = 0.85,
↪  plot.out = FALSE)
IndCR.2 <- data.frame(ID = rownames(as.matrix(gl.checkCR)), IndCR = 1 -
↪  rowSums(is.na(as.matrix(gl.checkCR)))/nLoc(gl.checkCR))
```

Define individual missing data threshold. This depends on the data set! Data sets with lots of SNPs can afford a lower threshold (e.g. in a data set of 100k snps, 10% still leaves 10k SNPs to play with).

Unfortunately, many individuals did not sequence well in this data set (likely due to DNA quality/quantity). This species also yielded low numbers of loci (with only ~12k SNPs called in the unfiltered data set). This means that the individual call rate threshold will have to be quite stringent so that the data set has enough power to detect the (presumed) low genetic structure present across the Pilbara. I've decided on a threshold of 0.64, which has removed a massive 21 individuals. If data were (hypothetically) filtered on a locus call rate of 85%, 14 individual would drop below an 85% individual call rate threshold. However, 85% of 12181 still leaves 1.0354×10^4 loci to play with, so I'm hopeful that we'll end up with enough overlapping loci after further downstream filtering.

3.3 Check population cutoff

Check if the number of individuals in each population have dropped below the sample size cutoff now that failed individuals have been filtered out of the data set. If they have changed, decide whether or not to include these populations when calculating the population genetic metrics for each locus.

```
# Check if any of the pops have dropped below threshold sample size number
PopKeep.IndFilt <- names(table(gl.RC.DropInds@pop) >=
↪  PopCutOff)[table(gl.RC.DropInds@pop) >=
    PopCutOff]
```

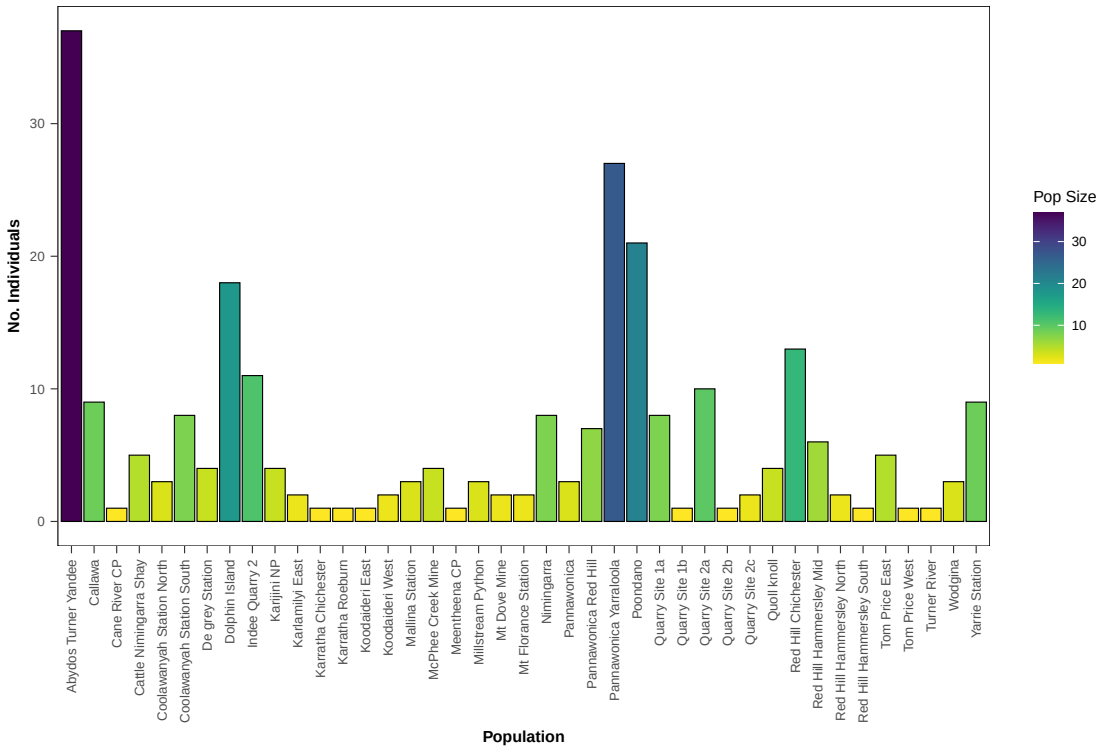

Figure 7: Population sample sizes after individual filtering.

```
if (!length(PopKeep.IndFilt) == length(PopKeep)) {
    PopKeep <- PopKeep.IndFilt
}
```

Here, one population has dropped below the sample size cutoff of 10 (so will be removed from population genetic summary statistic calculations).

4 Add extra chromosome information

Some DArTseq data sets are blasted against a reference genome (if one has been provided). The information that comes back is often in the form of contig IDs, rather than the actual chromosome. Searching the genome on NCBI often provides extra information relating these contig IDs to the chromosome (depending on how good the reference is). If available, add this information to explore how many SNPs there are per Chromosome.

```
if (BLAST == "Y") {

    ChromInfo.ncbi <- read.csv(file = paste0(datpath, ChromInfo.file), stringsAsFactors =
↪   FALSE)
```

```
    ## Rename column so I can join the two data frames by this variable
    colnames(ChromInfo.ncbi)[which(colnames(ChromInfo.ncbi) == ContigCol_ncbi)] <-
    ↪  ContigCol_dart

    ## Create a new data frame with all loci
    gl.RC.DropInds@other$loc.metrics[, c("Locus", ContigCol_dart, ContigPos_dart)] %>%
        mutate_if(is.factor, as.character) -> Locus_Chrom

    ## Join data frames for the chromosome number that corresponds to the contig ID
    Locus_Chrom <- left_join(Locus_Chrom, ChromInfo.ncbi, by = ContigCol_dart)

    ## Add the contig position to the chromosome position to get the actual position
    ## along the chromosome
    Locus_Chrom$TrueChromPos <- Locus_Chrom[, ContigPos_dart] + Locus_Chrom[,
↪   ChromPos_ncbi]

    ## Rename NA as 'UnMapped
    Locus_Chrom$Chromosome[is.na(Locus_Chrom$Chromosome)] <- "UnMapped"

    ## Check locus order matches so can append to genlight
    sum(!Locus_Chrom$Locus == locNames(gl.RC.DropInds))

    ## Append to genlight
    gl.RC.DropInds@other$loc.metrics$Chromosome <- Locus_Chrom$Chromosome
    gl.RC.DropInds@other$loc.metrics$TrueChromPos <- Locus_Chrom$TrueChromPos
}
```

5 Calculate population genetic metrics

Run the function to calculate population genetic metrics across the data set filtered on failed individuals.

```
# Run popgen function
gl.RC.DI.PG <- PopGenMetrics(gl = gl.RC.DropInds, pops = PopKeep)
```

6 Calculate number of private alleles

How does the minor allele frequency affect the number of private alleles. This is a useful way of visualising the interplay between real differences among populations (which may be important for detecting migrants, for example), and the prevalence of error in the data set (i.e. a low frequency SNP may indicate sequencing error). For now, just calculate private alleles across the raw data set, and dig into this after filtering. Note that this script calculates the mean number of private alleles between pairwise population comparisons.

```
# Use dartR script to calculate number of private alleles between each population pair
PrivAllele_raw <- gl.report.pa(gl.RC.DI.PG[gl.RC.DI.PG@pop %in% PopKeep, ], plot.out =
↪  FALSE)

# Drop pop factors that have been filtered out using drop levels and only keep relevant
# columns (rename so can combine)
priv1_raw <- PrivAllele_raw[, c("pop1", "priv1")]
priv2_raw <- PrivAllele_raw[, c("pop2", "priv2")]
colnames(priv1_raw) <- c("pop", "priv")
colnames(priv2_raw) <- c("pop", "priv")

## Combine into one data set so all pop comparisons are in one column
priv_raw <- rbind(priv1_raw, priv2_raw)
```

7 Output summary statistics for pre-filtered data set

```
write.csv(gl.RC.DI.PG@other$loc.metrics, paste0(outpath, Outfile_Desc,
↪  ".RawSumStats.csv"),
    row.names = F)
```

8 Visualise and filter raw data

Thoroughly interrogate the raw data set to tailor filtering thresholds to the study. The first step is to look at the patterns in the unfiltered, raw data set using a PCoA.

```
# PCoA

# Specify column for colouring/grouping PCoA
GroupCol_PCoA <- "IBRA_SubRegion"
gl.RC.DI.PG.pcoa <- gl.RC.DI.PG
pop(gl.RC.DI.PG.pcoa) <- gl.RC.DI.PG.pcoa$other$ind.metrics[, GroupCol_PCoA]

# Run a PCoA
pc <- gl.pcoa(gl.RC.DI.PG.pcoa, plot.out = FALSE)

# Calculate the percentage of variation represented by the axes
Eig <- round(pc$eig * 100/sum(pc$eig), 1)
```

```
## pdf
##   2
```

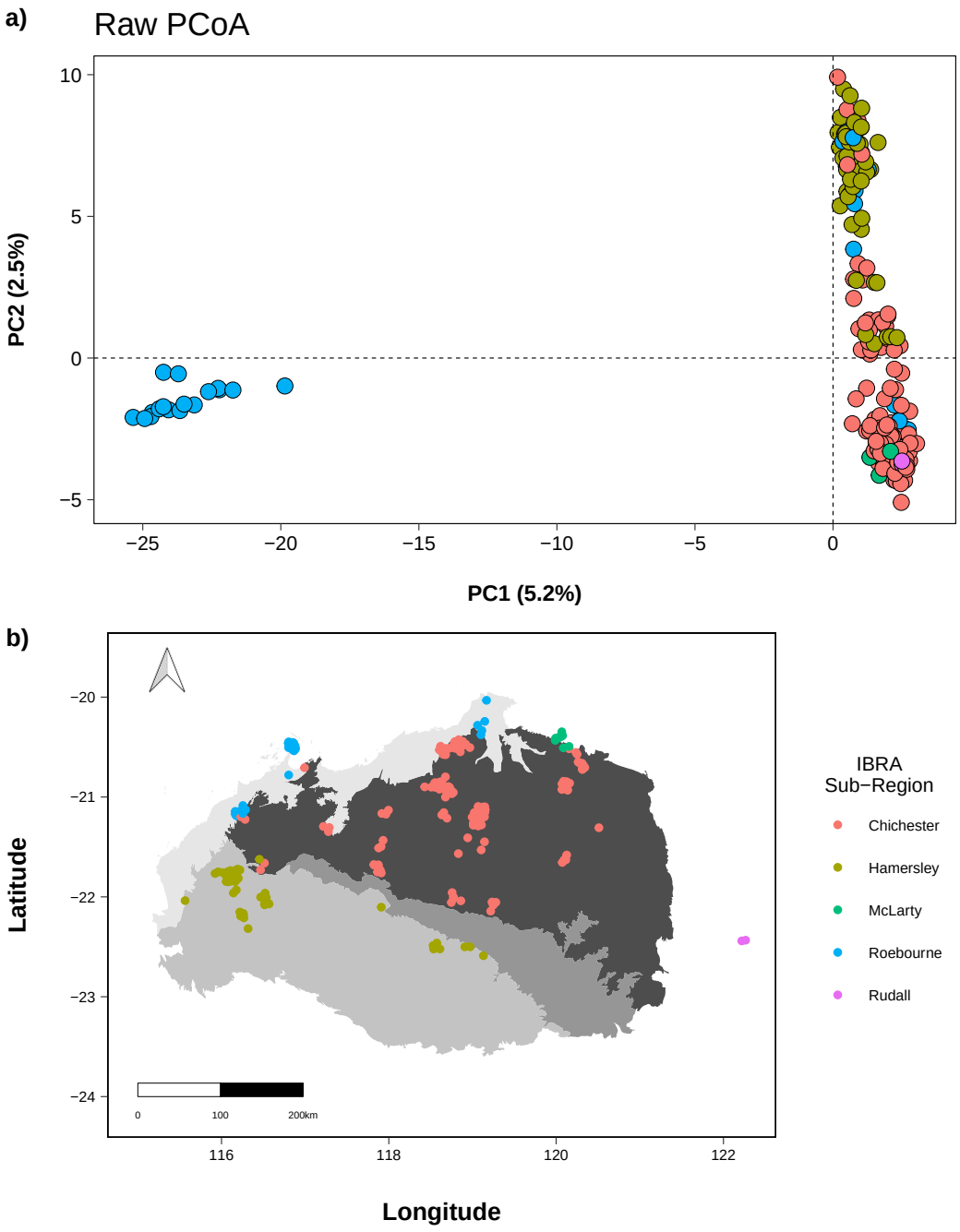

Figure 8: a) PCoA of raw SNP data (after removing failed individuals), b) Sample map coloured by population group.

There are some strong regional groupings in Figure 8, with 5% of the variation described by PC1 (suggesting that there is a high level of genetic differentiation between dolphin island and the rest of the Pilbara). Note that the patterns shown in the unfiltered PCoA often do not change substantially after filtering (which is a good indication that the filtering hasn't biased results, but has just cleaned the data set so that inferences are based on high quality, true loci).

8.1 Summary statistics

How do quality metrics impact population genetic summary statistics? The goal is to avoid biasing 'real' results by artifacts in the data (e.g. quality). Therefore, choose filters based on a threshold where data quality scores are no longer driving/correlated with summary statistics.

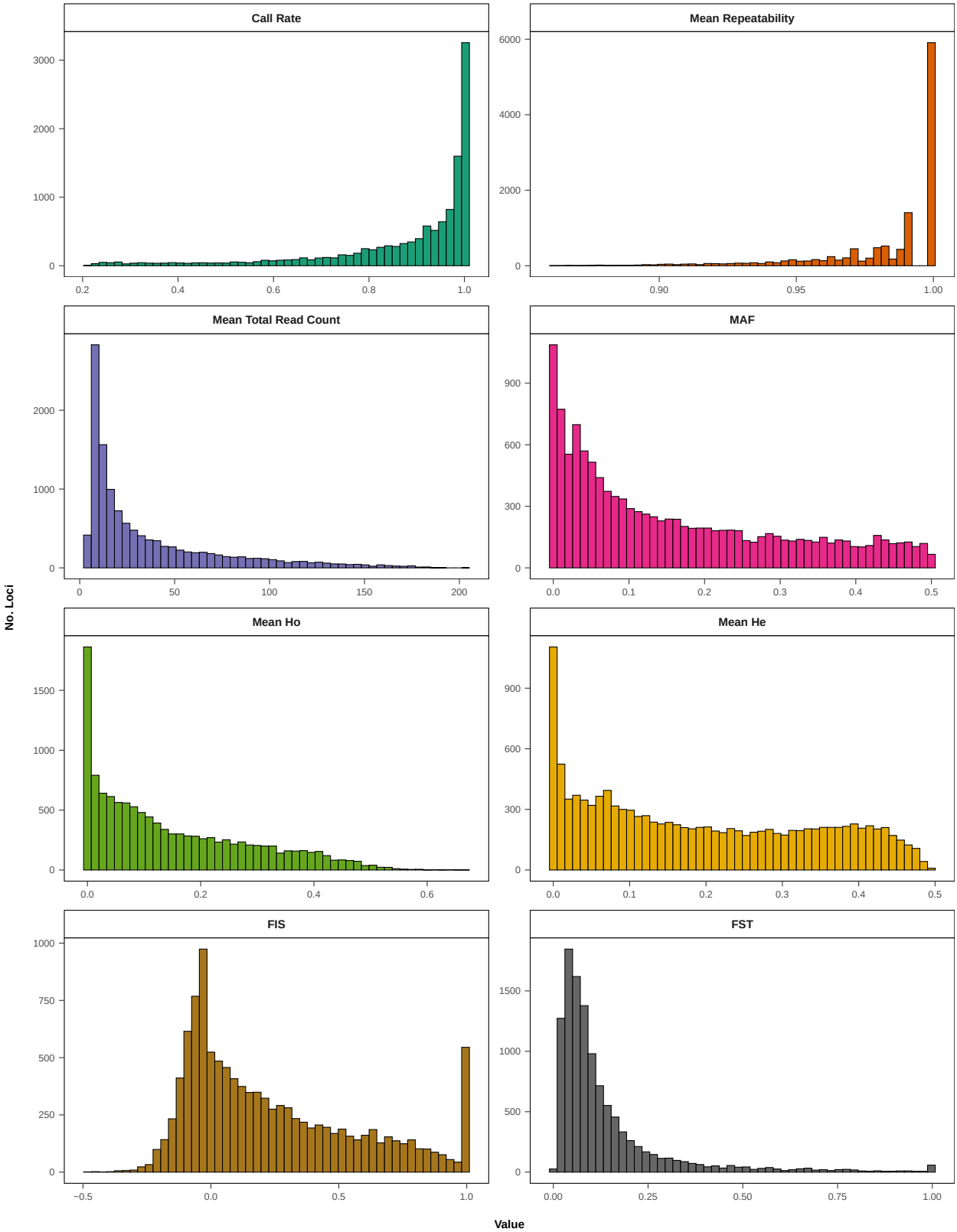

Figure 9: Pre-filtering summary statistics: histograms.

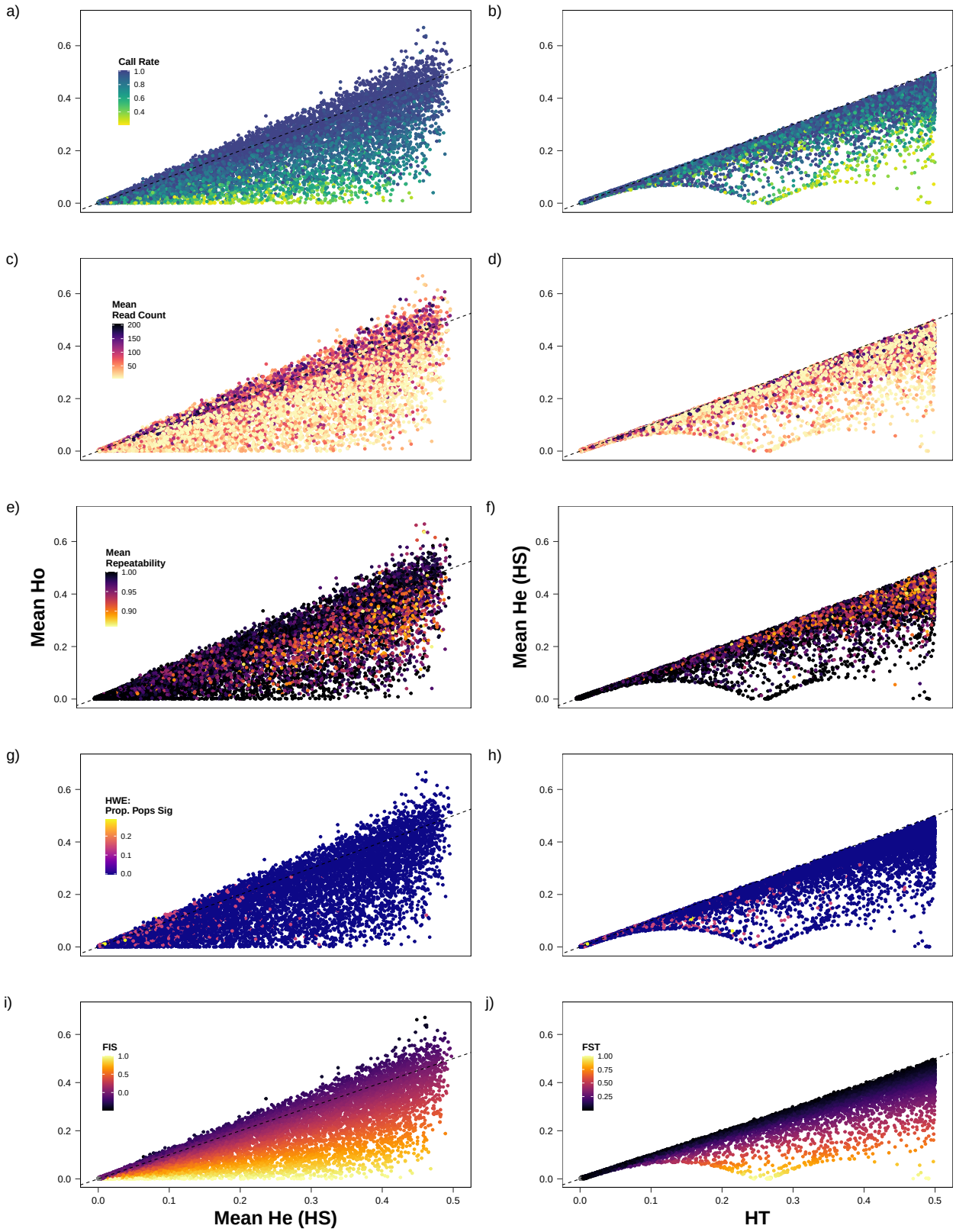

Figure 10: Pre-filtering scatter plots: mean Ho, HS, HT.

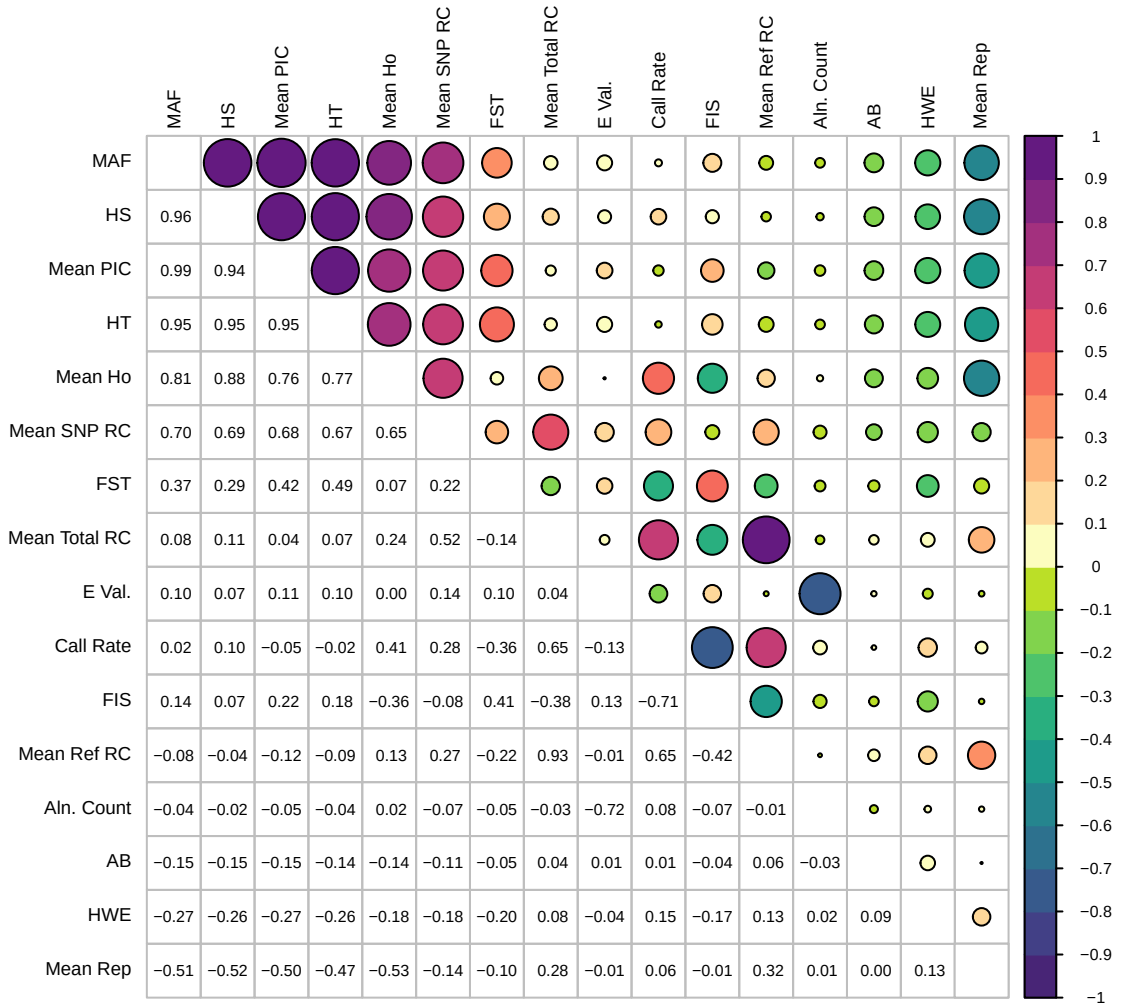

Figure 11: Pre-filtering correlation plot.

Figure 11 shows that call rate is strongly correlated with read count. This is not surprising, as a low read count results in missing data. Call rate is also strongly correlated with FIS, FST, HWE and heterozygosity (which can also be seen in Figure 10).

It is not particularly interesting to look into correlations between the different population genetic metrics, because it makes sense for the different heterozygosity measures and read count metrics to be correlated. Instead, look for correlations between quality metrics (read count, call rate, repeatability) and biological inferences (heterozygosity, FIS, FST, HWE). In this way, the data set can be filtered based on justifiable thresholds that correspond to data quality rather than biological hypotheses/expectations. After filtering on these metrics, see if any strange patterns in the data set disappear by monitoring the shape of the distributions in Figure 9 and by checking if the quality and population genetic metrics are no longer correlated. As an interesting side note, here, mean repeatability appears to be correlated with heterozygosity and MAF.

8.2 How are SNPs distributed among/along chromosomes?

If a reference genome was provided, explore how SNPs are mapped to the reference by visualising how SNPs are distributed among the different chromosomes.

```
if (BLAST == "Y") {
    # Plot distribution of SNPs along chromosomes (if you have this info)
    ChromDensity <- ggplot(RawRCSummStats, aes(x = Chromosome, fill = Chromosome)) +
↪   geom_bar(show.legend = FALSE,
        col = "black", size = 0.2) + scale_fill_viridis(discrete = TRUE, option = "D") +
        ↪  xlab("Chromosome") +
        ylab("Number of SNPs") + theme(panel.background = element_rect(fill = "white"),
        ↪  panel.border = element_rect(linetype = "solid",
        colour = "black", fill = "transparent", size = 0.2), panel.grid.minor =
        ↪  element_blank(),
        panel.grid.major = element_blank(), axis.title.x = element_text(face = "bold",
        ↪  size = 6),
        axis.title.y = element_text(face = "bold", size = 6), axis.ticks =
        ↪  element_line(size = 0.2),
        axis.text.x = element_text(size = 5), axis.text.y = element_text(size = 5),
        ↪  legend.position = "None")

    SNPDensity <- ggplot(Chrom_Mapped, aes(fill = Chromosome)) + geom_histogram(aes(x =
↪   TrueChromPos),
        show.legend = FALSE, col = "black", size = 0.2) + facet_wrap(~Chromosome, ncol =
        ↪  2,
        scales = "free_x") + scale_fill_viridis(discrete = TRUE, option = "D") +
        ↪  xlab("Position on Chromosome") +
        ylab("SNP density") + theme(panel.background = element_rect(fill = "white"),
        ↪  panel.border = element_rect(linetype = "solid",
        colour = "black", fill = "transparent", size = 0.2), panel.grid.minor =
        ↪  element_blank(),
        panel.grid.major = element_blank(), strip.text = element_text(face = "bold", size
        ↪  = 6),
        strip.background = element_rect(linetype = "solid", colour = "black", fill =
        ↪  "transparent",
            size = 0.2), axis.title.x = element_text(face = "bold", size = 6),
            ↪  axis.title.y = element_text(face = "bold",
            size = 6), axis.ticks = element_line(size = 0.2), axis.text.x =
            ↪  element_text(size = 5),
        axis.text.y = element_text(size = 5), legend.position = "None")

    # Output as jpeg
    jpeg(filename = paste0(outpath, Outfile_Desc, "Chrom.jpg"), units = "cm", width = 20,
    ↪  height = 28,
        res = 300)
    grid.arrange(ChromDensity, SNPDensity, nrow = 2, top = textGrob("Pre-Filtering SNP
    ↪  distributions on Chromosomes",
        gp = gpar(fontsize = 10, fontface = "bold")), heights = c(1/4, 3/4))
    dev.off()
}
```

In this case, 22% of SNPs mapped to the reference genome. This suggests that the Tasmanian devil genome is fairly divergent from *D. hallucatus*, although still seems to make an adequate reference. In addition, some loci BLASTED to sex chromosomes and unknown contigs.

8.3 Mapping quality

Visualise the distributions of the different population genetic summary statistics and quality/frequency metrics to see if there's a difference between SNPs that were mapped to the reference genome, versus those that were not. There might be a difference if the reference genome is closely related (in which case, unmapped reads may represent poorer quality SNPs), or if the reference is very distant (in which case, mapped reads may be less variable). In either situation, it may be useful to filter based on whether a SNP was mapped to the reference or not (although the former may be more justifiable than the latter).

```
if (BLAST == "Y") {
    # Plot histograms- mapped data set
    prefiltDF.hist.mapped <- prefiltDF.hist[prefiltDF.hist$Chrom.Mapped == "Mapped", ]

    Prefilt.hist.M.plot <- ggplot(data = prefiltDF.hist.mapped, aes(x = value, fill =
↪   MetaDat_Var)) +
        geom_histogram(bins = 50, colour = "black", size = 0.2) +
        ↪  facet_wrap(~MetaDat_Var,
        ncol = 2, scales = "free", labeller = labeller(MetaDat_Var = hist.labs)) +
        ↪  scale_fill_brewer(palette = "Dark2") +
        labs(x = "\nValue", y = "No. Loci\n", title = "Mapped SNPs") +
        ↪  theme(panel.background = element_rect(fill = "white"),
        panel.border = element_rect(linetype = "solid", colour = "black", fill =
        ↪  "transparent",
            size = 0.2), panel.grid.minor = element_blank(), panel.grid.major =
            ↪  element_blank(),
        strip.text = element_text(face = "bold", size = 6), strip.background =
        ↪  element_rect(linetype = "solid",
            colour = "black", fill = "white", size = 0.2), plot.title = element_text(face
            ↪  = "bold",
            size = 8, hjust = 0.5, colour = "DarkBlue"), axis.title.x = element_text(face
            ↪  = "bold",
            size = 6), axis.title.y = element_text(face = "bold", size = 6), axis.ticks =
            ↪  element_line(size = 0.2),
        axis.text.x = element_text(size = 5), axis.text.y = element_text(size = 5),
        ↪  legend.position = "None")

    # Plot histograms- unmapped data set

    prefiltDF.hist.unmapped <- prefiltDF.hist[prefiltDF.hist$Chrom.Mapped == "UnMapped",
↪   ]

    Prefilt.hist.UM.plot <- ggplot(data = prefiltDF.hist.unmapped, aes(x = value, fill =
↪   MetaDat_Var)) +
        geom_histogram(bins = 50, colour = "black", size = 0.2) +
        ↪  facet_wrap(~MetaDat_Var,
```

Figure 12: Distribution of SNPs across chromosomes before filtering.

```
        ncol = 2, scales = "free", labeller = labeller(MetaDat_Var = hist.labs)) +
        ↪  scale_fill_brewer(palette = "Dark2") +
        labs(x = "\nValue", y = "No. Loci\n", title = "Un-Mapped SNPs") +
        ↪  theme(panel.background = element_rect(fill = "white"),
        panel.border = element_rect(linetype = "solid", colour = "black", fill =
        ↪  "transparent",
            size = 0.2), panel.grid.minor = element_blank(), panel.grid.major =
            ↪  element_blank(),
        strip.text = element_text(face = "bold", size = 6), strip.background =
        ↪  element_rect(linetype = "solid",
            colour = "black", fill = "white", size = 0.2), plot.title = element_text(face
            ↪  = "bold",
            size = 8, hjust = 0.5, colour = "DarkBlue"), axis.title.x = element_text(face
            ↪  = "bold",
            size = 6), axis.title.y = element_text(face = "bold", size = 6), axis.ticks =
            ↪  element_line(size = 0.2),
        axis.text.x = element_text(size = 5), axis.text.y = element_text(size = 5),
        ↪  legend.position = "None")

    jpeg(filename = paste0(outpath, Outfile_Desc, ".RawHistograms.Mapped-UnMapped.jpg"),
    ↪  units = "cm",
        width = 28, height = 20, res = 300)

    grid.arrange(Prefilt.hist.M.plot + theme(plot.background = element_rect(colour =
    ↪  "black",
        size = 0.5)), Prefilt.hist.UM.plot + theme(plot.background = element_rect(colour
        ↪  = "black",
        size = 0.5)), ncol = 2)

    dev.off()
}
```

Looks like there's no real difference between mapped versus unmapped reads in terms of quality and population genetic metrics. Therefore, it isn't worth filtering on mapped/unmapped loci.

8.4 Quality filtering

Now that the data has been inspected, it's time to start filtering.

```
CR.report <- data.frame(CallRate = seq(0, 1, 0.05), Filtered =
↪  cumsum(table(cut(gl.RC.DI.PG@other$loc.metrics$CallRate,
    seq(-0.05, 1, 0.05)))))

tab.3_cap <- table_nums(name = "tab_3", caption = "Locus call rate report.")

CR.tab <- flextable(data = CR.report)
CR.tab <- autofit(CR.tab)
```

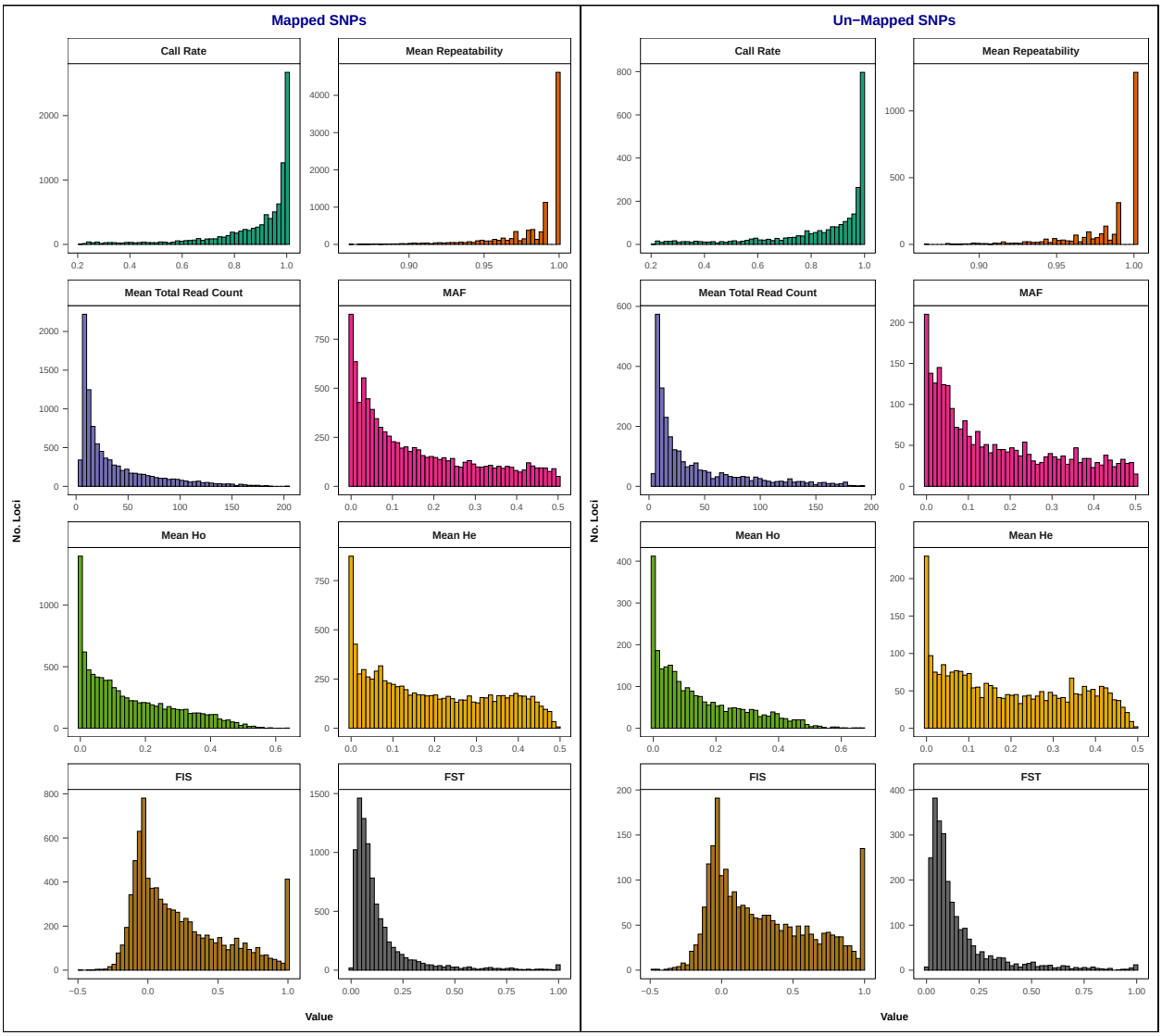

Figure 13: Pre-filtering histograms of summary statistics across mapped versus unmapped SNPs.

```
CR.tab <- bg(CR.tab, bg = "lightgrey", part = "header")
CR.tab <- bold(CR.tab, part = "header")
CR.tab <- align(CR.tab, align = "left")
CR.tab <- border_outer(CR.tab, part = "header", border = fp_border(color = "black", style
↪  = "solid",
    width = 0.5))
CR.tab <- border_outer(CR.tab, part = "body", border = fp_border(color = "black", style =
↪  "solid",
    width = 0.5))
CR.tab <- fontsize(CR.tab, size = 10)
```

Table 2: Locus call rate report.

CallRate	Filtered
0.00	0
0.05	0
0.10	0
0.15	0
0.20	0
0.25	83
0.30	208
0.35	338
0.40	456
0.45	573
0.50	708
0.55	852
0.60	1,067
0.65	1,318
0.70	1,622
0.75	2,047
0.80	2,629
0.85	3,422
0.90	4,474
0.95	6,037
1.00	12,181

Choose a call rate filtering threshold

```
CallRate <- 0.95

gl.CR.Filt <- gl.filter.callrate(gl.RC.DI.PG, method = "loc", threshold = CallRate,
↪  mono.rm = TRUE,
    recalc = TRUE, plot.out = FALSE)
```

After inspecting Table 2 and Figures 9, 10 and 11, choose a call rate filter that makes sense for the data. This will take some playing around and is always a balance between retaining the highest quality SNPs and maintaining a large enough data set for adequate statistical power. It will likely take a few runs to get the balance right.

In this case, I chose a call rate threshold of 0.95, which has reduced the data set down to 5997 SNPs.

Choose read count filtering thresholds:

```
RClower <- 20
RCupper <- 200

# Subset genlight rather than use dart function, because I've recalculated mean call rate
gl.CR.RC.Filt <- gl.CR.Filt[, !is.na(gl.CR.Filt@other$loc.metrics$RC_MeanTot) &
↪  gl.CR.Filt@other$loc.metrics$RC_MeanTot >
    RClower & gl.CR.Filt@other$loc.metrics$RC_MeanTot < RCupper]

# Have to subset loc.metrics too (annoyingly it doesn't do it automatically)
gl.CR.RC.Filt@other$loc.metrics <-
↪  gl.CR.Filt@other$loc.metrics[!is.na(gl.CR.Filt@other$loc.metrics$RC_MeanTot) &
    gl.CR.Filt@other$loc.metrics$RC_MeanTot > RClower &
↪  gl.CR.Filt@other$loc.metrics$RC_MeanTot <
    RCupper, ]
```

Mean read count appears to represent individual read counts well (see Figure 4). Use Figure 9 to make decisions about minimum and maximum read count (while keeping in mind that genotypes were strongly impacted by read count below/above individual read count thresholds in Figure 2). Low read counts could represent erroneous SNP calls (or loci that will have high levels of allelic dropout and therefore poor estimates of heterozygosity, etc.). Very high read counts often represent paralogues (two different regions of the genome, that have been collapsed as a single locus due to having similar sequences).

In this case, I chose a minimum mean read count of 20 and a maximum of 200, which has reduced the data set down to 4623 SNPs.

Choose repeatability filter:

```
RepFilt <- 0.95

gl.CR.RC.Rp.Filt <- gl.filter.reproducibility(gl.CR.RC.Filt, threshold = RepFilt,
↪  plot.out = FALSE)
```

It's important to be stringent with repeatability filters to avoid including poor quality loci. However, being too stringent may result in filtering out SNPs with high levels of heterozygosity (as heterozygote genotype calls may vary between replicates due to slight variations in ref/SNP reads, as suggested by the correlation between mean repeatability and heterozygosity in Figure 11).

For this reason, I've chosen a mean repeatability threshold of 0.95, which has reduced the data set down to 4558 SNPs.

Look at Post quality filtering plots:

Now that 'quality' filters are complete, visualise the data again.

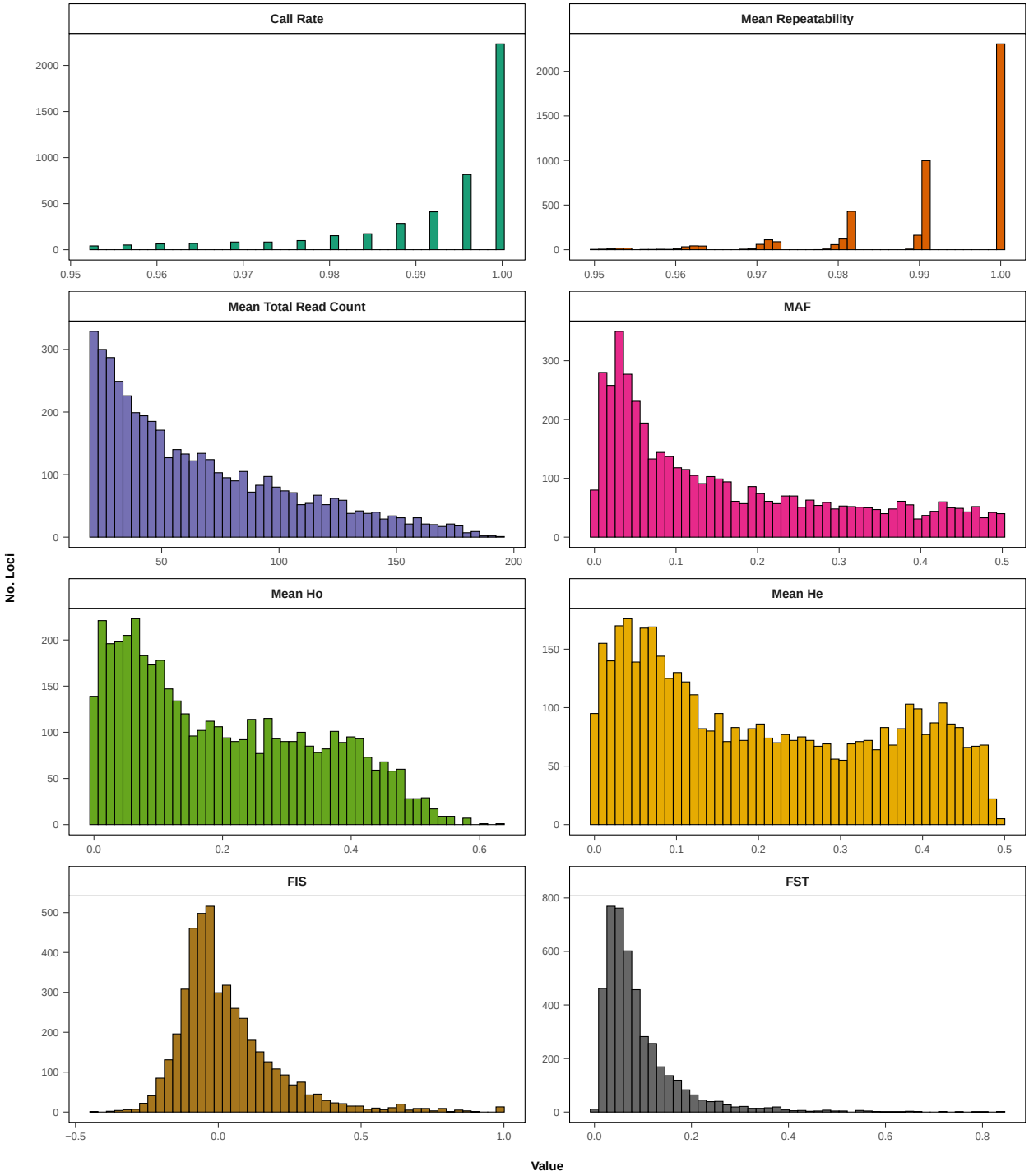

Figure 14: Post-filtering summary statistics: histograms.

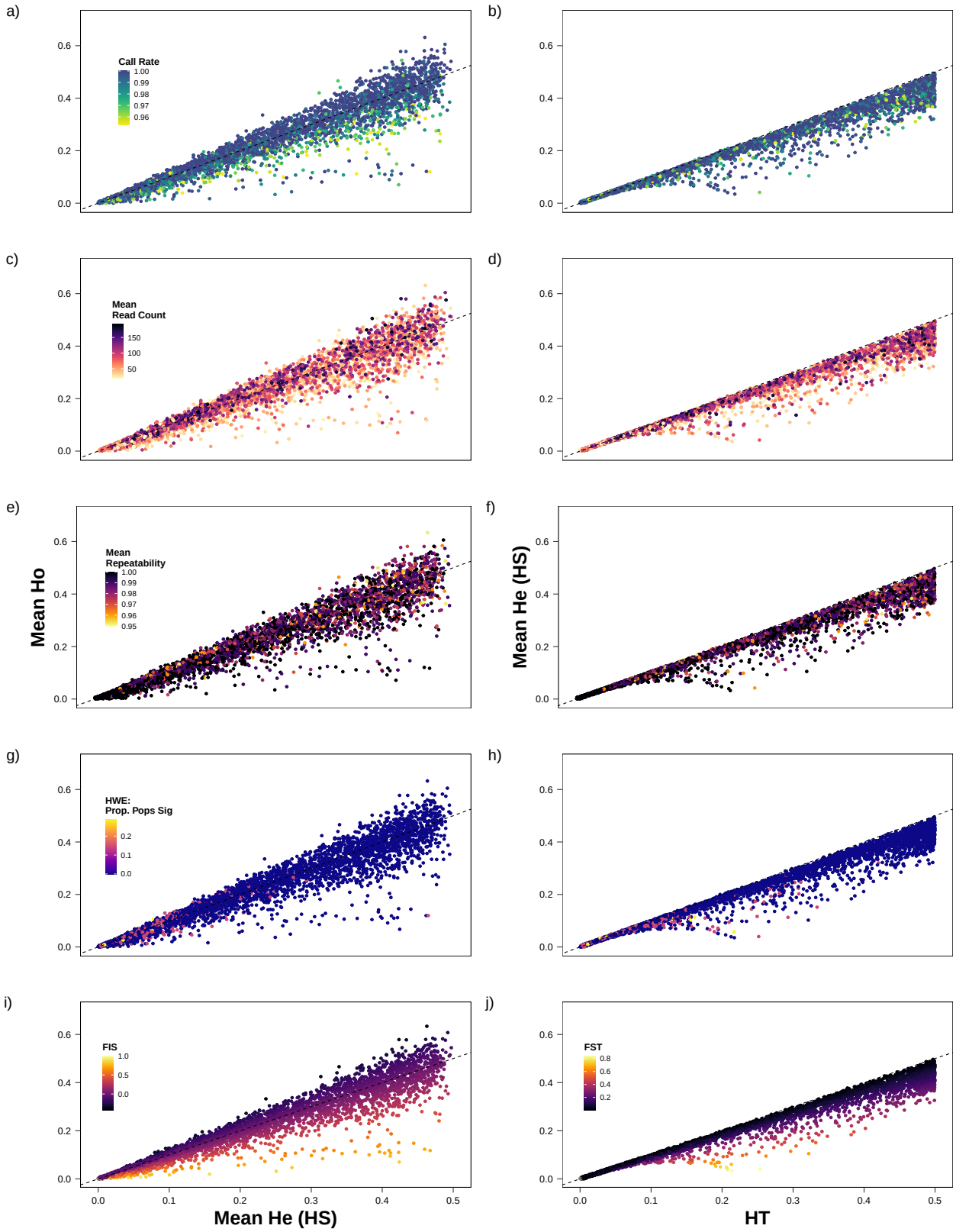

Figure 15: Post-filtering scatter plots: mean Ho, HS, HT.

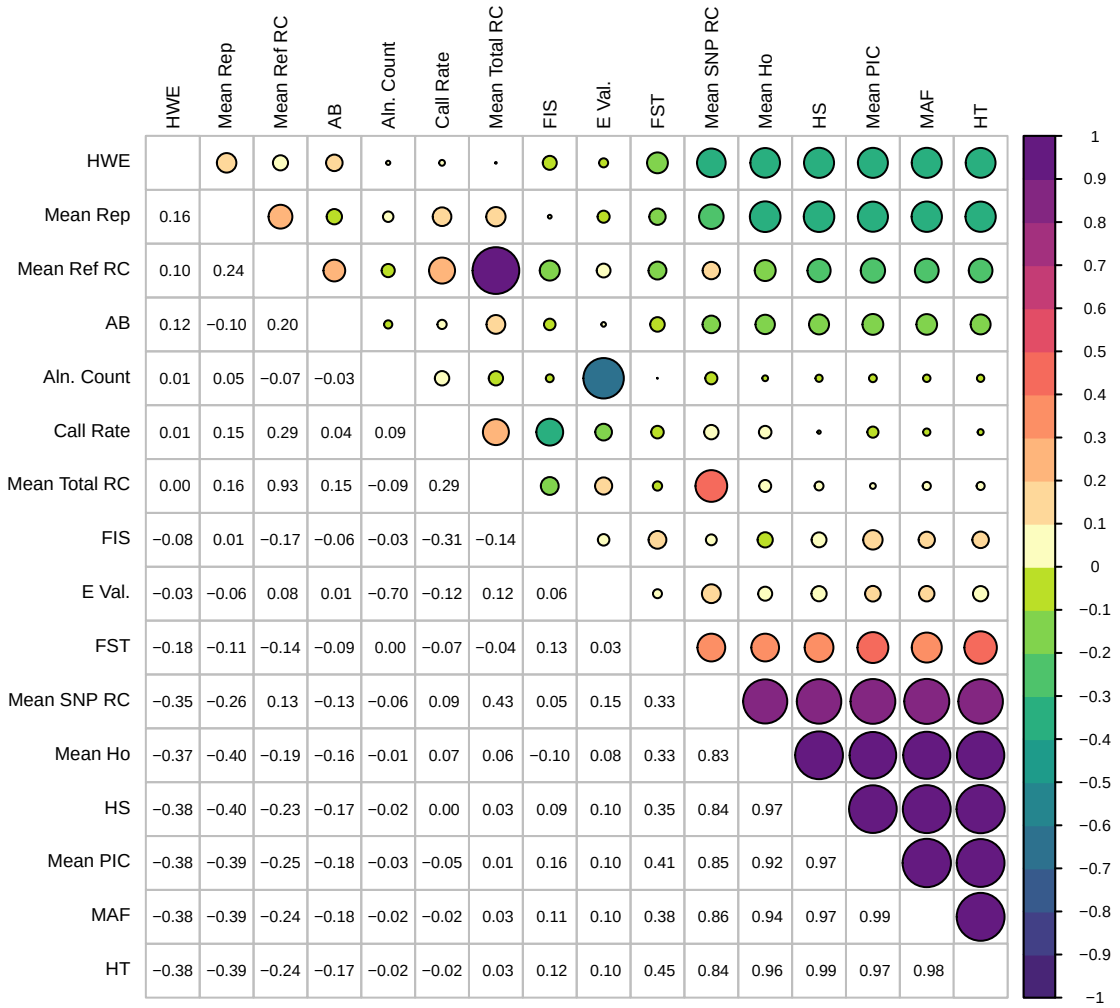

Figure 16: Post-filtering correlation plot

Call rate is no longer making such a large impact on population summary statistics (although FIS still seems to be somewhat influenced; Figure 16). Many of the loci with high FST and FIS values have been filtered out (Figure 14) and the remaining loci follow the diagonal Ho/He line much more closely (Figure 15).

8.5 Frequency filtering

Minor Allele Frequency (MAF)
It is important not to explicitly filter on allele frequencies, in order not to bias results/shape the data set based on the expected biological patterns. One exception to this is MAF. While choosing your MAF threshold, carefully think through what this filter actually means. For example, a MAF of 0.05 will mean very different things in a data set of 10 individuals compared to a data set of 1000 (i.e. 1 copy of this allele, versus 100). In the former case, this allele could be a sequencing error (which would be unlikely in the latter given how many copies of the allele are present). However, low frequency alleles may be missed at low sample sizes. All of this needs to be weighed up when coming up with the MAF threshold (what is the acceptable

error? How does this impact private alleles? How will this impact heterozygosity? What are the specific requirements/assumptions of flow on analyses?).

In this data set, the lowest MAF possible (1 copy of the allele) is 0.0078431. Therefore, if I want to filter so that an allele appears at least twice in the data set, my threshold must be greater than 0.0156863.

Hardy Weinberg Equilibrium (HWE)

Does it make sense to filter on HWE? For example, are there discrete populations? Will real results be removed that represent a biological question being tested? Do analyses assume HWE? If it's clear that the locus is out of HWE across populations, it might make sense to remove these loci for certain analyses, but in many cases, the SNPs out of HWE will be removed by quality filters anyway (i.e. if it's driven by missing data/low read count). Correct for multiple tests (Bonferroni), otherwise 5% of SNPs will be significant due to random chance, and in a large data set this means that many perfectly good SNPs will be removed for no reason.

Choose MAF filtering threshold:

```
MAFFilt <- 0.02

gl.CR.RC.Rp.MAF.Filt <- gl.filter.maf(gl.CR.RC.Rp.Filt, threshold = MAFFilt, plot.out =
↪  FALSE)
```

I chose a MAF threshold of 0.02, which has reduced the data set down to 4035 SNPs.

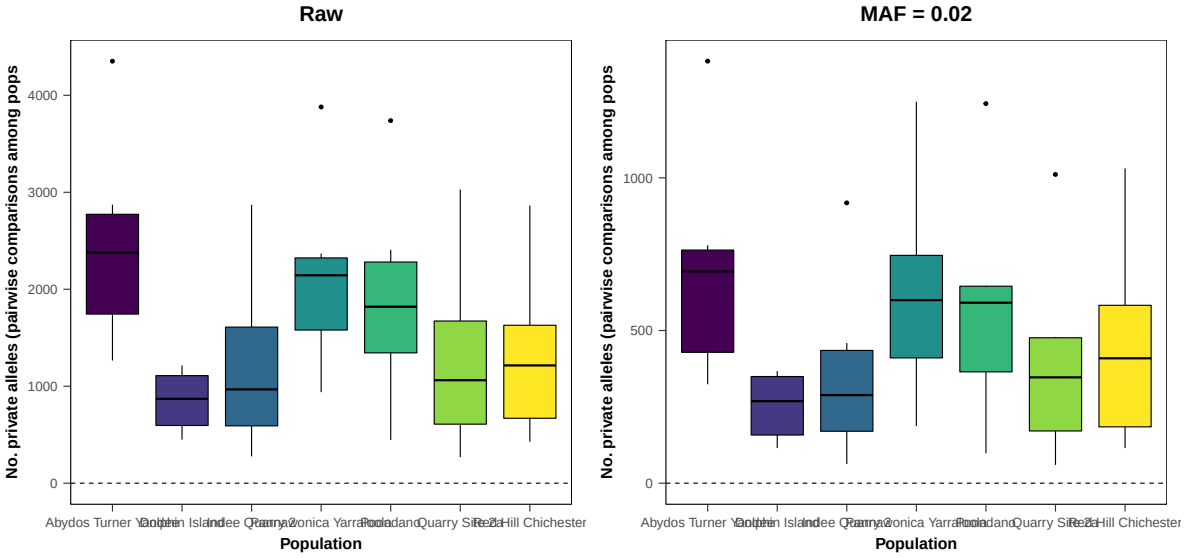

Figure 17: Pre versus post filtering box plots of mean number of pairwise private alleles among populations.

Figure 17 shows that the pattern of private alleles among populations doesn't really change, but the scale is drastically reduced after filtering (so filtering didn't change patterns of uniqueness/differences across populations - if it had dramatically changed, explore further/find out why).

8.6 Linkage filtering

Linkage means that SNPs are close enough to each other that they are not acting independently. It's important to remove non-independent SNPs from the data set, so that they don't bias the results by strengthening false patterns. However, it's a good idea to run this filter last, as many of these non-independent SNPs will be removed by earlier filters. This means that performing the LD filter last maximises the number of quality loci that are retained.

The first way to filter on LD is to remove multiple SNPs per sequence/locus (these are the SNPs that are known to be close enough to each other that they are not acting independently). After this, test for LD (possible even without reference genome) by determining if they are acting in a similar way (correlated). It is also important to remove sex-linked loci from the data set, as they have different patterns of inheritance to autosomal loci.

```
# Filter so 1 SNP per fragment Have to rename locus names as cloneID for this function to
# work
locNames(gl.CR.RC.Rp.MAF.Filt) <- gl.CR.RC.Rp.MAF.Filt@other$loc.metrics$CloneID
gl.CR.RC.Rp.MAF.lD.Filt <- gl.filter.secondaries(gl.CR.RC.Rp.MAF.Filt, method = "random")

# Filter out sex chromosomes
if (BLAST == "Y") {

↪  gl.CR.RC.Rp.MAF.lD.Filt@other$loc.metrics$Chromosome[gl.CR.RC.Rp.MAF.lD.Filt@other$loc.metrics$Chro
↪  ==
        "UnMapped"] <- NA
    gl.CR.RC.Rp.MAF.lD.Filt <- gl.CR.RC.Rp.MAF.lD.Filt[,
↪  !gl.CR.RC.Rp.MAF.lD.Filt@other$loc.metrics$Chromosome %in%
        SexChrom]

    # Have to subset loc.metrics too (annoyingly it doesn't do it automatically)
    gl.CR.RC.Rp.MAF.lD.Filt@other$loc.metrics <-
↪  gl.CR.RC.Rp.MAF.lD.Filt@other$loc.metrics[!gl.CR.RC.Rp.MAF.lD.Filt@other$loc.metrics$Chromosome
↪  %in%
        SexChrom, ]
}
```

After removing multiple SNPs per locus, there are 3794 SNPs.

```
# Now use SNP relate to remove linked loci
if (BLAST == "Y") {
    Chrom <- as.integer(gl.CR.RC.Rp.MAF.lD.Filt@other$loc.metrics$Chromosome)
    genmat.m <- t(as.matrix(gl.CR.RC.Rp.MAF.lD.Filt)[, !is.na(Chrom)])
    SampID.m <- colnames(genmat.m)
    SNP_ID.m <- rownames(genmat.m)
    ChromPos <- gl.CR.RC.Rp.MAF.lD.Filt@other$loc.metrics$TrueChromPos[!is.na(Chrom)]

    snpgdsCreateGeno(paste0(outpath, "Mapped_gl.CR.RC.Rp.MAF.lD.gds"), genmat = genmat.m,
    ↪  sample.id = SampID.m,
        snp.id = SNP_ID.m, snp.chromosome = Chrom[!is.na(Chrom)], snp.position =
        ↪  ChromPos,
        snpfirstdim = TRUE)
```

```
    genmat.um <- t(as.matrix(gl.CR.RC.Rp.MAF.lD.Filt)[, is.na(Chrom)])
    SampID.um <- colnames(genmat.um)
    SNP_ID.um <- rownames(genmat.um)

    snpgdsCreateGeno(paste0(outpath, "UnMapped_gl.CR.RC.Rp.MAF.lD.gds"), genmat =
    ↪  genmat.um,
        sample.id = SampID.um, snp.id = SNP_ID.um, snpfirstdim = TRUE)

    gds.m <- snpgdsOpen(paste0(outpath, "Mapped_gl.CR.RC.Rp.MAF.lD.gds"))
    gds.um <- snpgdsOpen(paste0(outpath, "UnMapped_gl.CR.RC.Rp.MAF.lD.gds"))

    # LD pruning - Try different LD thresholds for sensitivity analysis
    gds_LD.m <- snpgdsLDpruning(gdsobj = gds.m, sample.id = SampID.m, snp.id = SNP_ID.m,
↪   ld.threshold = 0.5,
        method = "corr", autosome.only = FALSE)

    gds_LD.um <- snpgdsLDpruning(gdsobj = gds.um, sample.id = SampID.um, snp.id =
↪   SNP_ID.um,
        ld.threshold = 0.5, method = "corr", autosome.only = FALSE)

    # Subset last Dart file with these snps
    SNP.List <- c(unlist(gds_LD.m, use.names = FALSE), unlist(gds_LD.um, use.names =
↪   FALSE))

    # Filter
    gl.CR.RC.Rp.MAF.lD.SR.Filt <- gl.CR.RC.Rp.MAF.lD.Filt[,
↪   gl.CR.RC.Rp.MAF.lD.Filt$loc.names %in%
        SNP.List]

    gl.CR.RC.Rp.MAF.lD.SR.Filt@other$loc.metrics <-
↪   gl.CR.RC.Rp.MAF.lD.SR.Filt@other$loc.metrics[gl.CR.RC.Rp.MAF.lD.SR.Filt@other$loc.metrics$CloneID
↪   %in%
        SNP.List, ]

    snpgdsClose(gds.um)
    snpgdsClose(gds.m)
    rm(gds.m, gds.um)


} else {
    genmat <- t(as.matrix(gl.CR.RC.Rp.MAF.lD.Filt))
    SampID <- colnames(genmat)
    SNP_ID <- rownames(genmat)

    snpgdsCreateGeno(paste0(outpath, "gl.CR.RC.Rp.MAF.lD.gds"), genmat = genmat,
    ↪  sample.id = SampID,
        snp.id = SNP_ID, snpfirstdim = TRUE)
    gds <- snpgdsOpen(paste0(outpath, "gl.CR.RC.Rp.MAF.lD.gds"))

    # LD pruning - Try different LD thresholds for sensitivity analysis
    gds_LD <- snpgdsLDpruning(gdsobj = gds, sample.id = SampID, snp.id = SNP_ID,
↪   ld.threshold = 0.5,
        method = "corr", autosome.only = FALSE)
```

```
    # Subset last Dart file with these snps
    SNP.List <- unlist(gds_LD, use.names = FALSE)

    # Filter
    gl.CR.RC.Rp.MAF.lD.SR.Filt <- gl.CR.RC.Rp.MAF.lD.Filt[,
↪   gl.CR.RC.Rp.MAF.lD.Filt$loc.names %in%
        SNP.List]

    gl.CR.RC.Rp.MAF.lD.SR.Filt@other$loc.metrics <-
↪   gl.CR.RC.Rp.MAF.lD.SR.Filt@other$loc.metrics[gl.CR.RC.Rp.MAF.lD.SR.Filt@other$loc.metrics$CloneID
↪   %in%
        SNP.List, ]

    snpgdsClose(gds)
    rm(gds)
}
```

Filtering on linkage disequilibrium has reduced the data set down to 3764 SNPs.

9 Inspect final plots

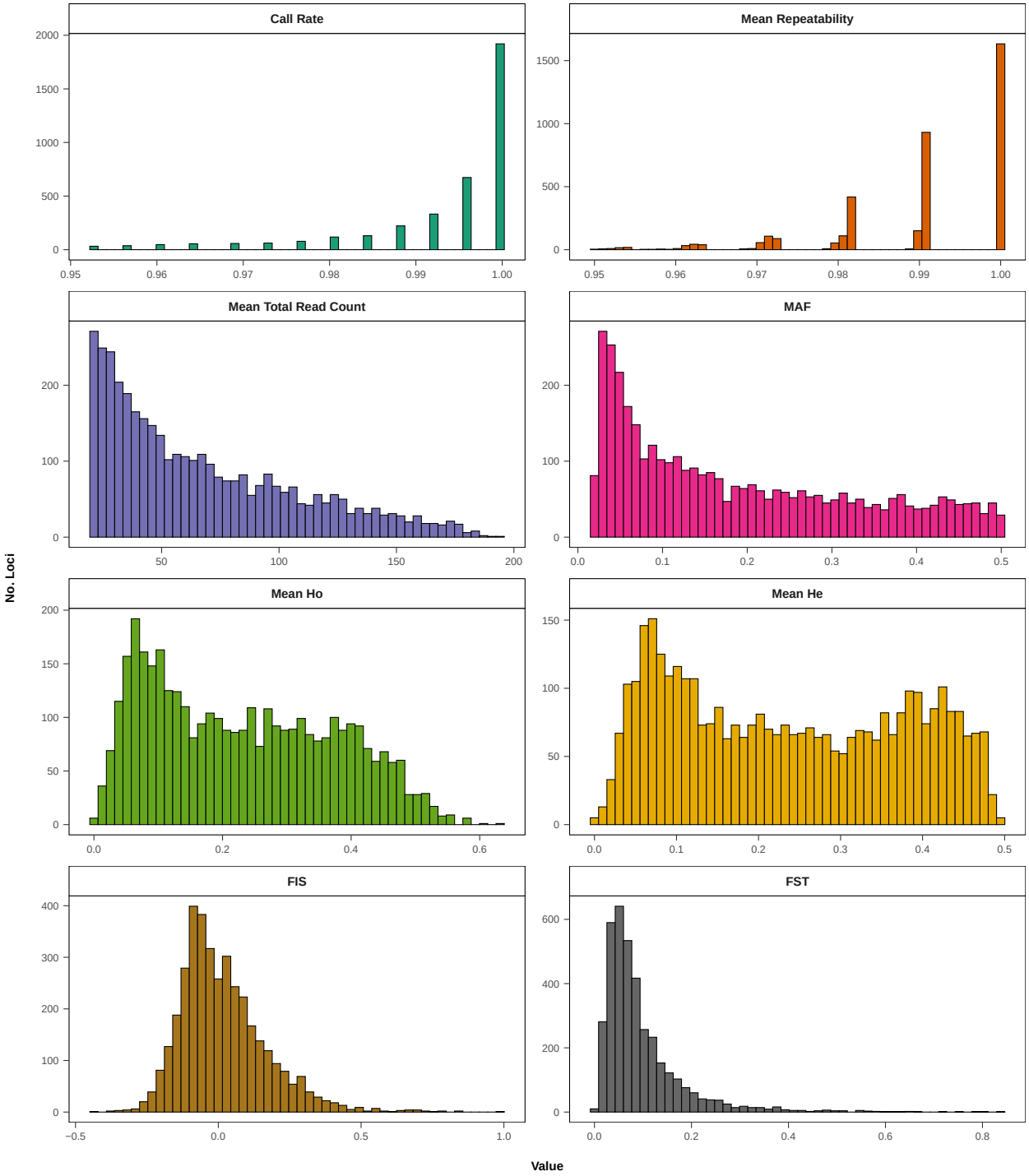

Figure 18: Post-filtering summary statistics: histograms.

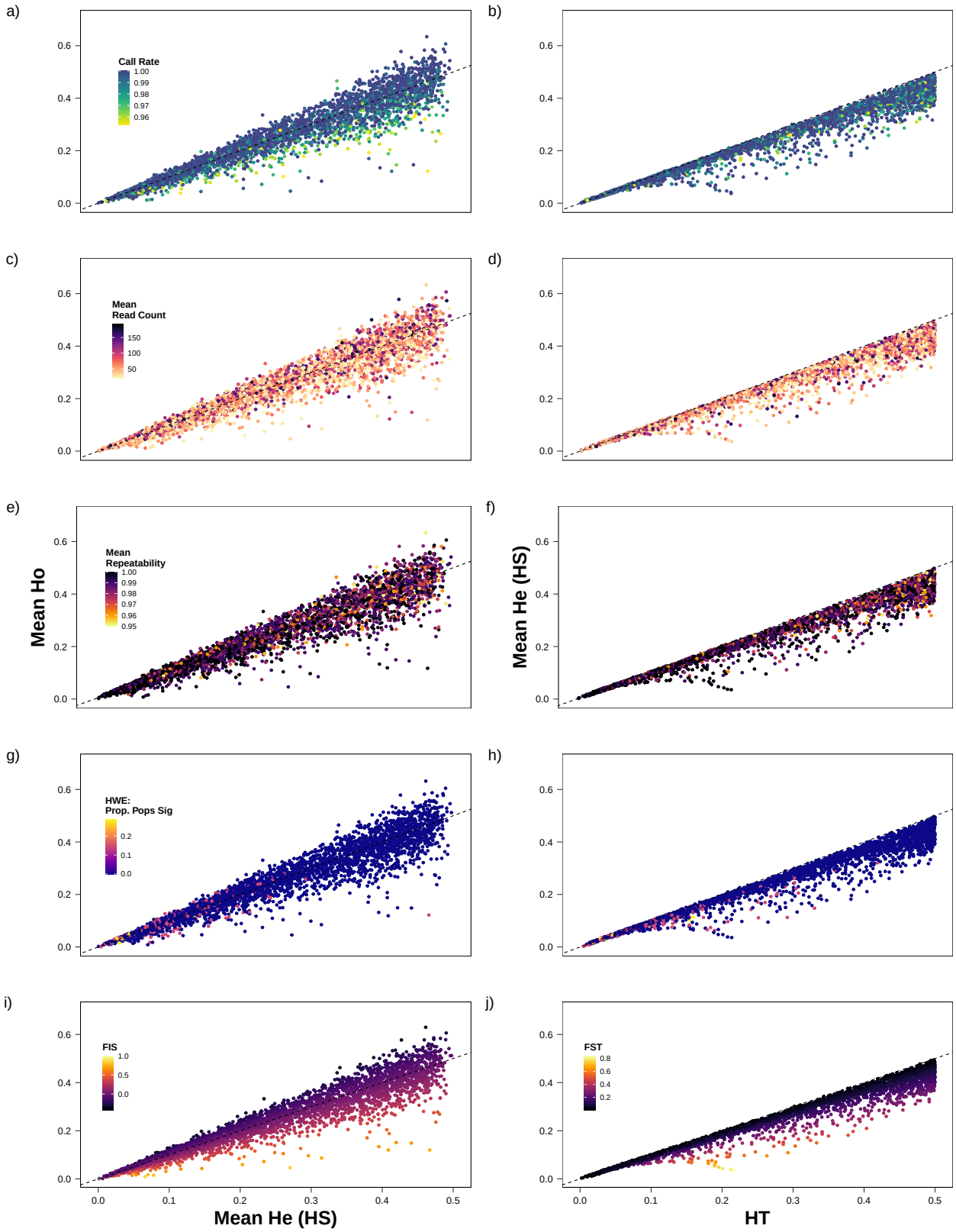

Figure 19: Post-filtering scatter plots: mean Ho, HS, HT.

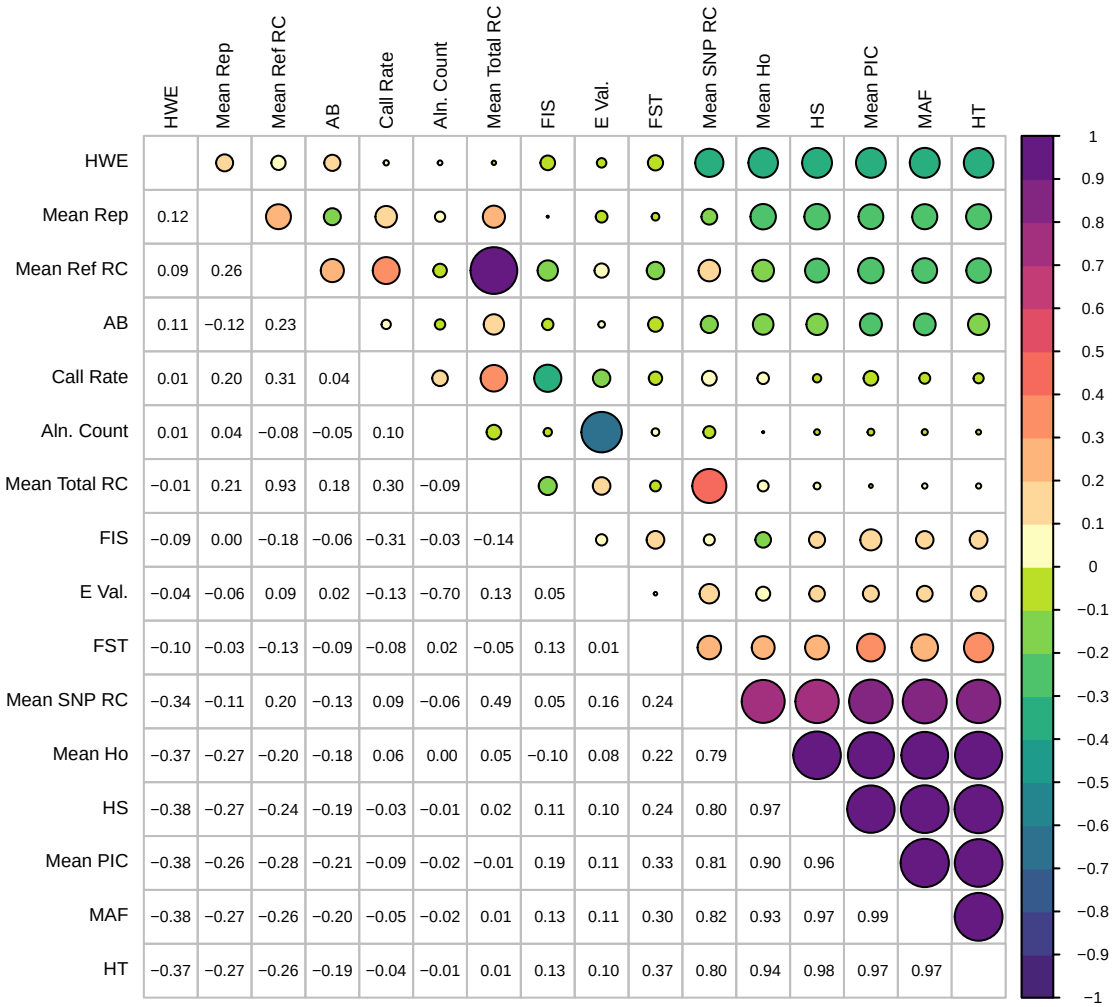

Figure 20: Post-filtering correlation plot

Figure 21 shows that the pattern found in the original PCoA hasn't really changed, although the PCs now explain slightly more variation (i.e. some of the noise has been removed).

This process will likely have to be repeated several times to get the thresholds right. Remember to filter to retain as many SNPs as possible, while also removing artifacts (driven by issues with data quality, sequencing error and non-independence) from the data. It's a trade off and as long as decisions are justifiable the end result will be a powerful, unbiased data set. In other words, don't choose thresholds based on the biological results expected, base them on quantifiable metrics that describe the quality of the data.

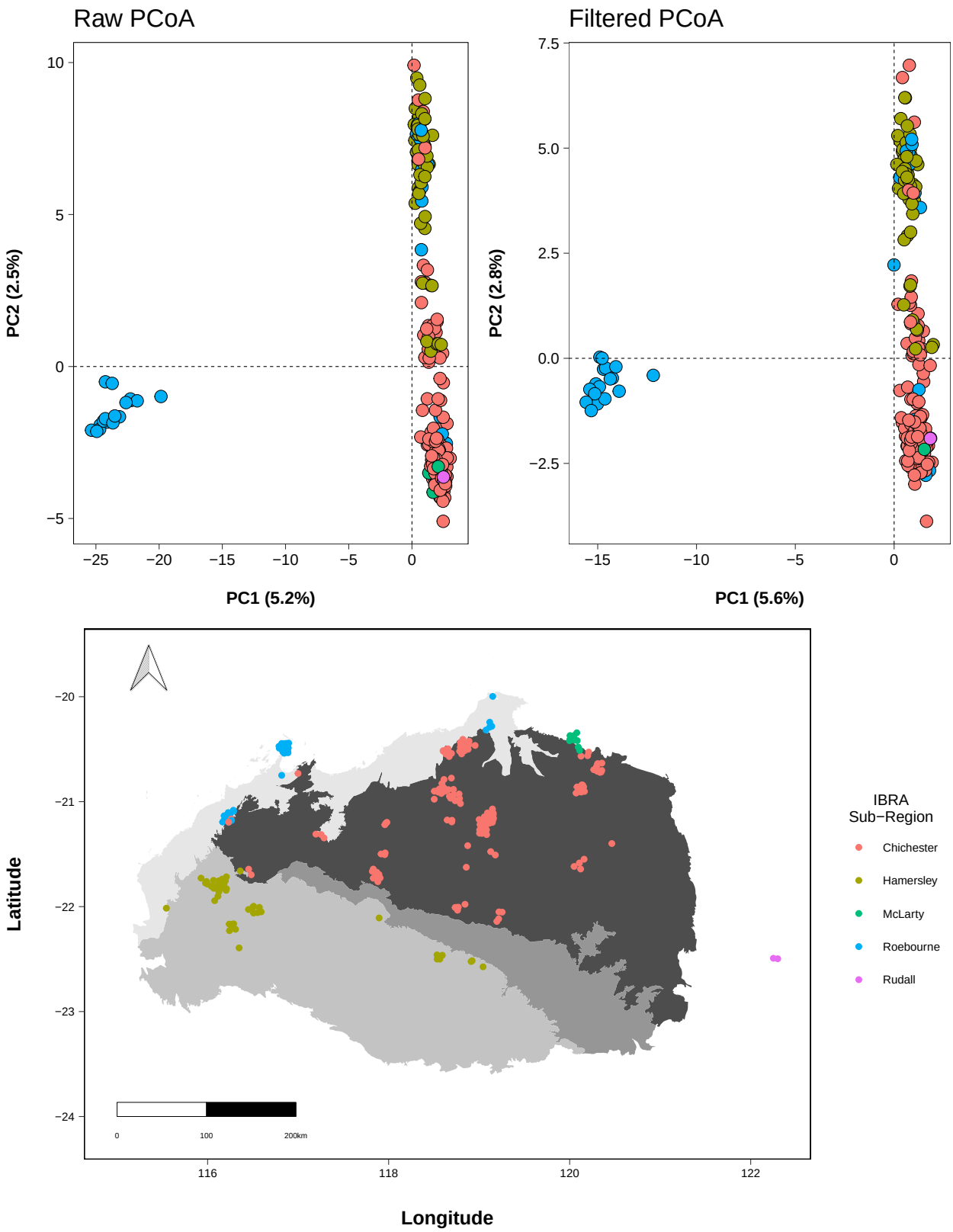

Figure 21: PCoA on final filtered SNP data set, compared to raw data set and sample map.

9.1 Save final filtered data set

```
# Save genlight (so that we can load the genlight into another r session)
gl.FinalFilt <- gl.CR.RC.Rp.MAF.lD.SR.Filt
save(gl.FinalFilt, file = paste0(outpath, "gl.", Outfile_Desc, "_FinalFilt.rdata"))

# Save locus metrics
write.csv(gl.FinalFilt@other$loc.metrics, paste0(outpath, Outfile_Desc,
↪  ".FiltSumStats.csv"),
    row.names = F)
```

10 Session Information

Current session info

```
- Session info ---------------------------------------------------------------
 setting  value
 version  R version 4.1.2 (2021-11-01)
 os       macOS Big Sur 10.16
 system   x86_64, darwin17.0
 ui       X11
 language (EN)
 collate  en_AU.UTF-8
 ctype    en_AU.UTF-8
 tz       Australia/Sydney
 date     2022-08-03
 pandoc   2.14.0.3 @ /Applications/RStudio.app/Contents/MacOS/pandoc/ (via rmarkdown)

- Packages -------------------------------------------------------------------
 package       * version   date (UTC) lib source
 abind           1.4-5     2016-07-21 [1] CRAN (R 4.1.0)
 ade4          * 1.7-18    2021-09-16 [1] CRAN (R 4.1.0)
 adegenet      * 2.1.5     2021-10-09 [1] CRAN (R 4.1.0)
 ape             5.6-2     2022-03-02 [1] CRAN (R 4.1.2)
 assertthat      0.2.1     2019-03-21 [1] CRAN (R 4.1.0)
 backports       1.4.1     2021-12-13 [1] CRAN (R 4.1.0)
 base64enc       0.1-3     2015-07-28 [1] CRAN (R 4.1.0)
 bayesm          3.1-4     2019-10-15 [1] CRAN (R 4.1.0)
 bitops          1.0-7     2021-04-24 [1] CRAN (R 4.1.0)
 bookdown        0.26.3    2022-06-07 [1] Github (rstudio/bookdown@169c43b)
 broom           0.8.0     2022-04-13 [1] CRAN (R 4.1.2)
 calibrate       1.7.7     2020-06-19 [1] CRAN (R 4.1.0)
 captioner     * 2.2.3     2015-07-16 [1] CRAN (R 4.1.0)
 car             3.0-13    2022-05-02 [1] CRAN (R 4.1.2)
 carData         3.0-5     2022-01-06 [1] CRAN (R 4.1.2)
 cellranger      1.1.0     2016-07-27 [1] CRAN (R 4.1.0)
 class           7.3-20    2022-01-13 [1] CRAN (R 4.1.2)
 classInt        0.4-7     2022-06-10 [1] CRAN (R 4.1.2)
 cli             3.3.0     2022-04-25 [1] CRAN (R 4.1.2)
 clipr           0.8.0     2022-02-22 [1] CRAN (R 4.1.2)
 cluster         2.1.2     2021-04-17 [1] CRAN (R 4.1.2)
 codetools       0.2-18    2020-11-04 [1] CRAN (R 4.1.2)
 colorspace      2.0-3     2022-01-15 [1] R-Forge (R 4.1.2)
 combinat        0.0-8     2012-10-29 [1] CRAN (R 4.1.0)
 compositions    2.0-4     2022-01-05 [1] CRAN (R 4.1.2)
 corrplot      * 0.92      2021-11-18 [1] CRAN (R 4.1.2)
 cowplot         1.1.1     2022-02-14 [1] Github (wilkelab/cowplot@555c9ae)
 crayon          1.5.1     2022-03-26 [1] CRAN (R 4.1.2)
 dartR         * 2.0.3     2022-03-28 [1] CRAN (R 4.1.2)
 data.table      1.14.2    2021-09-27 [1] CRAN (R 4.1.0)
 DBI             1.1.2     2021-12-20 [1] CRAN (R 4.1.0)
 dbplyr          2.1.1     2021-04-06 [1] CRAN (R 4.1.0)
 DEoptimR        1.0-10    2022-01-03 [1] CRAN (R 4.1.2)
 desc            1.4.1     2022-03-06 [1] CRAN (R 4.1.2)
```

```
details       * 0.3.0     2022-03-27 [1] CRAN (R 4.1.2)
digest          0.6.29    2021-12-01 [1] CRAN (R 4.1.0)
dismo           1.3-5     2021-10-11 [1] CRAN (R 4.1.0)
doParallel      1.0.17    2022-02-07 [1] CRAN (R 4.1.2)
dotCall64       1.0-1     2021-02-11 [1] CRAN (R 4.1.0)
dplyr         * 1.0.9     2022-04-28 [1] CRAN (R 4.1.2)
e1071           1.7-11    2022-06-07 [1] CRAN (R 4.1.2)
ellipsis        0.3.2     2021-04-29 [1] CRAN (R 4.1.0)
evaluate        0.15      2022-02-18 [1] CRAN (R 4.1.2)
fansi           1.0.3     2022-03-24 [1] CRAN (R 4.1.2)
farver          2.1.0     2021-02-28 [1] CRAN (R 4.1.0)
fastmap         1.1.0     2021-01-25 [1] CRAN (R 4.1.0)
fields          13.3      2021-10-30 [1] CRAN (R 4.1.0)
flextable     * 0.7.2     2022-06-12 [1] CRAN (R 4.1.2)
forcats       * 0.5.1     2021-01-27 [1] CRAN (R 4.1.0)
foreach         1.5.2     2022-02-02 [1] CRAN (R 4.1.2)
foreign         0.8-82    2022-01-13 [1] CRAN (R 4.1.2)
formatR       * 1.12.1    2022-08-02 [1] Github (yihui/formatR@942bff2)
fs              1.5.2     2021-12-08 [1] CRAN (R 4.1.0)
gap             1.2.3-1   2021-04-21 [1] CRAN (R 4.1.0)
gdata           2.18.0    2017-06-06 [1] CRAN (R 4.1.0)
gdistance       1.3-6     2020-06-29 [1] CRAN (R 4.1.0)
gdsfmt        * 1.30.0    2021-10-26 [1] Bioconductor
gdtools         0.2.4     2022-02-14 [1] CRAN (R 4.1.2)
generics        0.1.2     2022-01-31 [1] CRAN (R 4.1.2)
genetics        1.3.8.1.3 2021-03-01 [1] CRAN (R 4.1.0)
GGally          2.1.2     2021-06-21 [1] CRAN (R 4.1.0)
ggmap           3.0.0     2019-02-05 [1] CRAN (R 4.1.0)
ggplot2       * 3.3.6     2022-05-03 [1] CRAN (R 4.1.2)
ggpubr        * 0.4.0     2020-06-27 [1] CRAN (R 4.1.0)
ggsignif        0.6.3     2021-09-09 [1] CRAN (R 4.1.0)
ggsn          * 0.5.0     2019-02-18 [1] CRAN (R 4.1.0)
ggtern          3.3.5     2021-07-23 [1] CRAN (R 4.1.0)
glue            1.6.2     2022-02-24 [1] CRAN (R 4.1.2)
gridExtra     * 2.3       2017-09-09 [1] CRAN (R 4.1.0)
gtable          0.3.0     2019-03-25 [1] CRAN (R 4.1.0)
gtools          3.9.2     2021-06-06 [1] CRAN (R 4.1.0)
HardyWeinberg   1.7.4     2021-11-26 [1] CRAN (R 4.1.0)
haven           2.5.0     2022-04-15 [1] CRAN (R 4.1.2)
highr           0.9       2021-04-16 [1] CRAN (R 4.1.0)
hms             1.1.1     2021-09-26 [1] CRAN (R 4.1.0)
htmltools       0.5.2     2021-08-25 [1] CRAN (R 4.1.0)
htmlwidgets     1.5.4     2021-09-08 [1] CRAN (R 4.1.0)
httpuv          1.6.5     2022-01-05 [1] CRAN (R 4.1.2)
httr            1.4.3     2022-05-04 [1] CRAN (R 4.1.2)
igraph          1.3.2     2022-06-13 [1] CRAN (R 4.1.2)
iterators       1.0.14    2022-02-05 [1] CRAN (R 4.1.2)
jpeg            0.1-9     2021-07-24 [1] CRAN (R 4.1.0)
jsonlite        1.8.0     2022-02-22 [1] CRAN (R 4.1.2)
KernSmooth      2.23-20   2021-05-03 [1] CRAN (R 4.1.2)
knitr         * 1.39.6    2022-08-02 [1] Github (yihui/knitr@bebf67e)
labeling        0.4.2     2020-10-20 [1] CRAN (R 4.1.0)
later           1.3.0     2021-08-18 [1] CRAN (R 4.1.0)
```

```
latex2exp       0.9.4     2022-03-02 [1] CRAN (R 4.1.2)
lattice         0.20-45   2021-09-22 [1] CRAN (R 4.1.2)
lifecycle       1.0.1     2021-09-24 [1] CRAN (R 4.1.0)
lubridate       1.8.0     2021-10-07 [1] CRAN (R 4.1.0)
magrittr        2.0.3     2022-03-30 [1] CRAN (R 4.1.2)
maps            3.4.0     2021-09-25 [1] CRAN (R 4.1.0)
maptools        1.1-4     2022-04-17 [1] CRAN (R 4.1.2)
MASS            7.3-55    2022-01-13 [1] CRAN (R 4.1.2)
Matrix          1.4-0     2021-12-08 [1] CRAN (R 4.1.0)
mgcv            1.8-39    2022-02-24 [1] CRAN (R 4.1.2)
mice            3.14.0    2021-11-24 [1] CRAN (R 4.1.0)
mime            0.12      2021-09-28 [1] CRAN (R 4.1.0)
mmod            1.3.3     2017-04-06 [1] CRAN (R 4.1.0)
modelr          0.1.8     2020-05-19 [1] CRAN (R 4.1.0)
munsell         0.5.0     2018-06-12 [1] CRAN (R 4.1.0)
mvtnorm         1.1-3     2021-10-08 [1] CRAN (R 4.1.0)
networkD3       0.4       2017-03-18 [1] CRAN (R 4.1.0)
nlme            3.1-155   2022-01-13 [1] CRAN (R 4.1.2)
nnet            7.3-17    2022-01-13 [1] CRAN (R 4.1.2)
officer       * 0.4.3     2022-06-12 [1] CRAN (R 4.1.2)
patchwork       1.1.1     2020-12-17 [1] CRAN (R 4.1.0)
pegas           1.1       2021-12-16 [1] CRAN (R 4.1.0)
permute         0.9-7     2022-01-27 [1] CRAN (R 4.1.2)
pillar          1.7.0     2022-02-01 [1] CRAN (R 4.1.2)
pkgconfig       2.0.3     2019-09-22 [1] CRAN (R 4.1.0)
plyr            1.8.7     2022-03-24 [1] CRAN (R 4.1.2)
png             0.1-7     2013-12-03 [1] CRAN (R 4.1.0)
PopGenReport    3.0.4     2019-02-04 [1] CRAN (R 4.1.0)
promises        1.2.0.1   2021-02-11 [1] CRAN (R 4.1.0)
proto           1.0.0     2016-10-29 [1] CRAN (R 4.1.0)
proxy           0.4-27    2022-06-09 [1] CRAN (R 4.1.2)
purrr         * 0.3.4     2020-04-17 [1] CRAN (R 4.1.0)
R.methodsS3     1.8.1     2020-08-26 [1] CRAN (R 4.1.0)
R.oo            1.24.0    2020-08-26 [1] CRAN (R 4.1.0)
R.utils         2.11.0    2021-09-26 [1] CRAN (R 4.1.0)
R6              2.5.1     2021-08-19 [1] CRAN (R 4.1.0)
raster          3.5-15    2022-01-22 [1] CRAN (R 4.1.2)
RColorBrewer    1.1-3     2022-04-03 [1] CRAN (R 4.1.2)
Rcpp            1.0.8.3   2022-03-17 [1] CRAN (R 4.1.2)
readr         * 2.1.2     2022-01-30 [1] CRAN (R 4.1.2)
readxl          1.3.1     2019-03-13 [1] CRAN (R 4.1.0)
reprex          2.0.1     2021-08-05 [1] CRAN (R 4.1.0)
reshape         0.8.8     2018-10-23 [1] CRAN (R 4.1.0)
reshape2        1.4.4     2020-04-09 [1] CRAN (R 4.1.0)
rgdal         * 1.5-28    2021-12-15 [1] CRAN (R 4.1.0)
rgeos           0.5-9     2021-12-15 [1] CRAN (R 4.1.0)
RgoogleMaps     1.4.5.3   2020-02-12 [1] CRAN (R 4.1.0)
rjson           0.2.21    2022-01-09 [1] CRAN (R 4.1.2)
rlang           1.0.2     2022-03-04 [1] CRAN (R 4.1.2)
rmarkdown       2.14      2022-04-25 [1] CRAN (R 4.1.2)
robustbase      0.93-9    2021-09-27 [1] CRAN (R 4.1.0)
rprojroot       2.0.3     2022-04-02 [1] CRAN (R 4.1.2)
Rsolnp          1.16      2015-12-28 [1] CRAN (R 4.1.0)
```

```
 rstatix         0.7.0     2021-02-13 [1] CRAN (R 4.1.0)
 rstudioapi      0.13      2020-11-12 [1] CRAN (R 4.1.0)
 rvest           1.0.2     2021-10-16 [1] CRAN (R 4.1.0)
 scales          1.2.0     2022-04-13 [1] CRAN (R 4.1.2)
 seqinr          4.2-8     2021-06-09 [1] CRAN (R 4.1.0)
 sessioninfo   * 1.2.2     2021-12-06 [1] CRAN (R 4.1.0)
 sf              1.0-7     2022-03-07 [1] CRAN (R 4.1.2)
 shiny           1.7.1     2021-10-02 [1] CRAN (R 4.1.0)
 SNPRelate     * 1.28.0    2021-10-26 [1] Bioconductor
 sp            * 1.5-0     2022-06-05 [1] CRAN (R 4.1.2)
 spam            2.8-0     2022-01-06 [1] CRAN (R 4.1.2)
 StAMPP          1.6.3     2021-08-08 [1] CRAN (R 4.1.0)
 stringi         1.7.8     2022-07-11 [1] CRAN (R 4.1.2)
 stringr       * 1.4.0     2019-02-10 [1] CRAN (R 4.1.0)
 systemfonts     1.0.4     2022-02-11 [1] CRAN (R 4.1.2)
 tensorA         0.36.2    2020-11-19 [1] CRAN (R 4.1.0)
 terra           1.5-21    2022-02-17 [1] CRAN (R 4.1.2)
 tibble        * 3.1.7     2022-05-03 [1] CRAN (R 4.1.2)
 tidyr         * 1.2.0     2022-02-01 [1] CRAN (R 4.1.2)
 tidyselect      1.1.2     2022-02-21 [1] CRAN (R 4.1.2)
 tidyverse     * 1.3.1     2021-04-15 [1] CRAN (R 4.1.0)
 truncnorm       1.0-8     2018-02-27 [1] CRAN (R 4.1.0)
 tzdb            0.3.0     2022-03-28 [1] CRAN (R 4.1.2)
 units           0.8-0     2022-02-05 [1] CRAN (R 4.1.2)
 utf8            1.2.2     2021-07-24 [1] CRAN (R 4.1.0)
 uuid            1.1-0     2022-04-19 [1] CRAN (R 4.1.2)
 vctrs           0.4.1     2022-04-13 [1] CRAN (R 4.1.2)
 vegan           2.5-7     2020-11-28 [1] CRAN (R 4.1.0)
 viridis       * 0.6.2     2021-10-13 [1] CRAN (R 4.1.0)
 viridisLite   * 0.4.0     2021-04-13 [1] CRAN (R 4.1.0)
 withr           2.5.0     2022-03-03 [1] CRAN (R 4.1.2)
 xfun            0.31      2022-05-10 [1] CRAN (R 4.1.2)
 xml2            1.3.3     2021-11-30 [1] CRAN (R 4.1.0)
 xtable          1.8-4     2019-04-21 [1] CRAN (R 4.1.0)
 yaml            2.3.5     2022-02-21 [1] CRAN (R 4.1.2)
 zip             2.2.0     2021-05-31 [1] CRAN (R 4.1.0)

 [1] /Library/Frameworks/R.framework/Versions/4.1/Resources/library

------------------------------------------------------------------------------
```

References

Alathea, L. (2015). *Captioner: Numbers figures and creates simple captions.* Retrieved from https://CRAN.R-project.org/package=captioner

Auguie, B. (2017). *gridExtra: Miscellaneous functions for "grid" graphics.* Retrieved from https://CRAN.R-project.org/package=gridExtra

Bivand, R., Keitt, T., & Rowlingson, B. (2021). *Rgdal: Bindings for the 'geospatial' data abstraction library.* Retrieved from https://CRAN.R-project.org/package=rgdal

Garnier, Simon, Ross, Noam, Rudis, Robert, ... Cédric. (2021). *viridis - colorblind-friendly color maps for r.* doi: 10.5281/zenodo.4679424

Gohel, D. (2022a). *Flextable: Functions for tabular reporting.* Retrieved from https://CRAN.R-project.org/package=flextable

Gohel, D. (2022b). *Officer: Manipulation of microsoft word and PowerPoint documents.* Retrieved from https://CRAN.R-project.org/package=officer

Gruber, B., Unmack, P. J., Berry, O. F., & Georges, A. (2018). Dartr: An r package to facilitate analysis of SNP data generated from reduced representation genome sequencing. *Molecular Ecology Resources*, *18*, 691–699. doi: 10.1111/1755-0998.12745

Kassambara, A. (2020). *Ggpubr: 'ggplot2' based publication ready plots.* Retrieved from https://CRAN.R-project.org/package=ggpubr

Müller, K., & Wickham, H. (2022). *Tibble: Simple data frames.* Retrieved from https://CRAN.R-project.org/package=tibble

O'Leary, S. J., Puritz, J. B., Willis, S. C., Hollenbeck, C. M., & Portnoy, D. S. (2018). These aren't the loci you'e looking for: Principles of effective SNP filtering for molecular ecologists. *Molecular Ecology*, *27*(16), 3193–3206. doi: 10.1111/mec.14792

Peakall, R., & Smouse, P. E. (2006). GENALEX 6: genetic analysis in Excel. Population genetic software for teaching and research. *Molecular Ecology Notes*, *6*(1), 288–295. doi: 10.1111/j.1471-8286.2005.01155.x

Peakall, R., & Smouse, P. E. (2012). GenAlEx 6.5: genetic analysis in Excel. Population genetic software for teaching and research-an update. *Bioinformatics*, *28*(19), 2537–2539. doi: 10.1093/bioinformatics/bts460

R Core Team. (2021). *R: A language and environment for statistical computing.* Vienna, Austria: R Foundation for Statistical Computing. Retrieved from https://www.R-project.org/

Santos Baquero, O. (2019). *Ggsn: North symbols and scale bars for maps created with 'ggplot2' or'ggmap'.* Retrieved from https://CRAN.R-project.org/package=ggsn

Sidi, J. (2022). *Details: Create details HTML tag for markdown and package documentation.* Retrieved from https://CRAN.R-project.org/package=details

Wei, T., & Simko, V. (2021). *R package 'corrplot': Visualization of a correlation matrix.* Retrieved from https://github.com/taiyun/corrplot

Wickham, H. (2016). *ggplot2: Elegant graphics for data analysis.* Springer-Verlag New York. Retrieved from https://ggplot2.tidyverse.org

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., ... Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, *4*(43), 1686. doi: 10.21105/joss.01686

Wickham, H., Chang, W., Flight, R., Müller, K., & Hester, J. (2021). *Sessioninfo: R session information.* Retrieved from https://CRAN.R-project.org/package=sessioninfo

Wickham, H., François, R., Henry, L., & Müller, K. (2022). *Dplyr: A grammar of data manipulation.* Retrieved from https://CRAN.R-project.org/package=dplyr

Wickham, H., & Girlich, M. (2022). *Tidyr: Tidy messy data.* Retrieved from https://CRAN.R-project.org/package=tidyr

Xie, Y. (2014). Knitr: A comprehensive tool for reproducible research in R. In V. Stodden, F. Leisch, & R. D. Peng (Eds.), *Implementing reproducible computational research.* Chapman; Hall/CRC. Retrieved from http://www.crcpress.com/product/isbn/9781466561595

Xie, Y. (2015). *Dynamic documents with R and knitr* (2nd ed.). Boca Raton, Florida: Chapman; Hall/CRC. Retrieved from https://yihui.org/knitr/

Xie, Y. (2022). *Knitr: A general-purpose package for dynamic report generation in r.* Retrieved from https://yihui.org/knitr/

Zheng, X., Levine, D., Shen, J., Gogarten, S., Laurie, C., & Weir, B. (2012). A high-performance computing toolset for relatedness and principal component analysis of SNP data. *Bioinformatics*, *28*(24), 3326–3328. doi: 10.1093/bioinformatics/bts606