Comprehensive Analysis of DNA Methylation Using the Illumina HumanMethylation450 BeadChip Platform

Keshani, K., Castro Vargas, P., Gustini, B., Aynede, A., Dobrokhotov, I., & Otoijamun, F.

2025-06

Contents

Keshani, K., Castro Vargas, P., Gustini, B., MohammadNejad Aynede, A., Dobrokhotov, I., & Otoijamun, F. (2025). DRD_2025_Project: Educational pipeline for DNA methylation analysis (Version 1.0.0) [Computer software]. https://github.com/kianinsilico/DRD_2025_Project

1 Introduction

This project focuses on the analysis of DNA methylation data generated using the Illumina HumanMethylation450 BeadChip (450K array) to explore epigenetic differences between sample groups. Utilizing statistical tests, principal component analysis, and heatmap visualizations, the study aims to identify significant methylation patterns that may contribute to biological insights. This work was conducted as part of the DNA/RNA Dynamics course at the Università di Bologna, combining practical bioinformatics techniques with real epigenomic datasets to deepen understanding of epigenetic regulation mechanisms.

2 Analytical pipeline and R code

2.1 STEP 0. BASIC PREPARATION (Environment Setting and Importing Libraries)

Set the proper working directory where the data is stored

```
#setwd("~/DRD_2025_Project")
```

Install and load the necessary packages

```
library(minfi)
library(minfiData)
library(IlluminaHumanMethylation450kmanifest)
library(IlluminaHumanMethylation450kanno.ilmn12.hg19)
library(shinyMethyl)
library(AnnotationDbi)
library(sva)
library(gplots)
library(qqman)
library(tinytex)
install.packages("../lib/SummarizedExperiment_1.38.0.tar.gz", repos = NULL)
```

2.2 STEP 1. DATA IMPORT AND OBJECT CREATION

Load raw data with minfi and create an object called RGset storing the RGChannelSet object.

List files in data directory

```
list.files("../data/raw/")
```

Load the sample sheet

```
SampleSheet <- read.csv("../data/raw/SampleSheet_Report_II.csv", header =
 ↪  TRUE)
SampleSheet
```

Load sample sheet using minfi's function

```
baseDir <- ("../data/raw")
targets <- read.metharray.sheet(baseDir)
targets
```

Create RGChannelSet object

```
RGset <- read.metharray.exp(targets = targets)
save(RGset, file = "../data/processed/RGChannelSet.RData")
load("../data/processed/RGChannelSet.RData")
```

Explore the RGset object

```
RGset
```

```
str(RGset)
```

2.3 STEP 2. EXTRACT RED AND GREEN CHANNELS

Create dataframes to store the red and green fluorescences.

Extract fluorescence intensity information for Red and Green channels

```
Red <- data.frame(getRed(RGset))
dim(Red)
head(Red)
Green <- data.frame(getGreen(RGset))
dim(Green)
head(Green)
```

2.4 STEP 3. PROBE INFORMATION AND ADDRESS CHECKS

Determine whether address 18756452 belongs to a Type I or Type II probe by retrieving the red and green fluorescence values for this address and checking the manifest file.

Explore manifest and probe information

```
getManifest(RGset)
getManifestInfo(RGset)
getProbeInfo(RGset)
ProbeInfo_I <- data.frame(getProbeInfo(RGset))
dim(ProbeInfo_I)
head(getProbeInfo(RGset, type = "II"))
ProbeInfo_II <- data.frame(getProbeInfo(RGset, type = "II"))
dim(ProbeInfo_II)
```

Check probe with address 18756452

```
ProbeInfo_I[ProbeInfo_I$AddressA == "18756452",]
ProbeInfo_I[ProbeInfo_I$AddressB == "18756452",]
ProbeInfo_II[ProbeInfo_II$AddressB == "18756452",]
ProbeInfo_II[ProbeInfo_II$AddressA == "18756452",]
```

There is a Type II probe at address 18756452, identified by the name cg01523029.

Extract Red/Green fluorescence for address 18756452

```
Probe_Info_18756452 <- ProbeInfo_II[ProbeInfo_II$AddressA == "18756452",]
Red_fluorescences <- Red[rownames(Red) == "18756452",]
Green_fluorescences <- Green[rownames(Green) == "18756452",]
```

Use sample names from the column names of Red/Green (same order) and create summary dataframe to fill the requested table

```
sample_names <- colnames(Red_fluorescences)
df_summary_probes <- data.frame(
  "Sample" = sample_names,
  "Red Fluorescence" = as.numeric (Red_fluorescences[1, ]),
  "Green Fluorescence" = as.numeric (Green_fluorescences[1,]),
  "Type" = rep("II", length(sample_names)),
  "Color" = rep("Both", length(sample_names))
)
print(df_summary_probes)
```

2.5 STEP 4. Creation of the MSet.raw oject

Extract methylated and unmethylated signals

```
MSet.raw <- preprocessRaw(RGset)
MSet.raw
save(MSet.raw, file = "../data/processed/MSet_raw.RData")
Meth <- as.matrix(getMeth(MSet.raw))
str(Meth)
head(Meth)
Unmeth <- as.matrix(getUnmeth(MSet.raw))
str(Unmeth)
head(Unmeth)
```

Check probe cg01523029 in MethylSet

```
Unmeth[rownames(Unmeth) == "cg01523029",]
Meth[rownames(Meth) == "cg01523029",]

load("../data/raw/Illumina450Manifest.RData")
Illumina450Manifest_clean <- Illumina450Manifest[Illumina450Manifest$CHR
 ↪  != "",]
Illumina450Manifest_clean <- droplevels(Illumina450Manifest_clean)
Probe_Info_cg01523029 <-
 ↪  Illumina450Manifest_clean[Illumina450Manifest_clean$IlmnID ==
 ↪  "cg01523029",]
```

2.6 5. Quality control

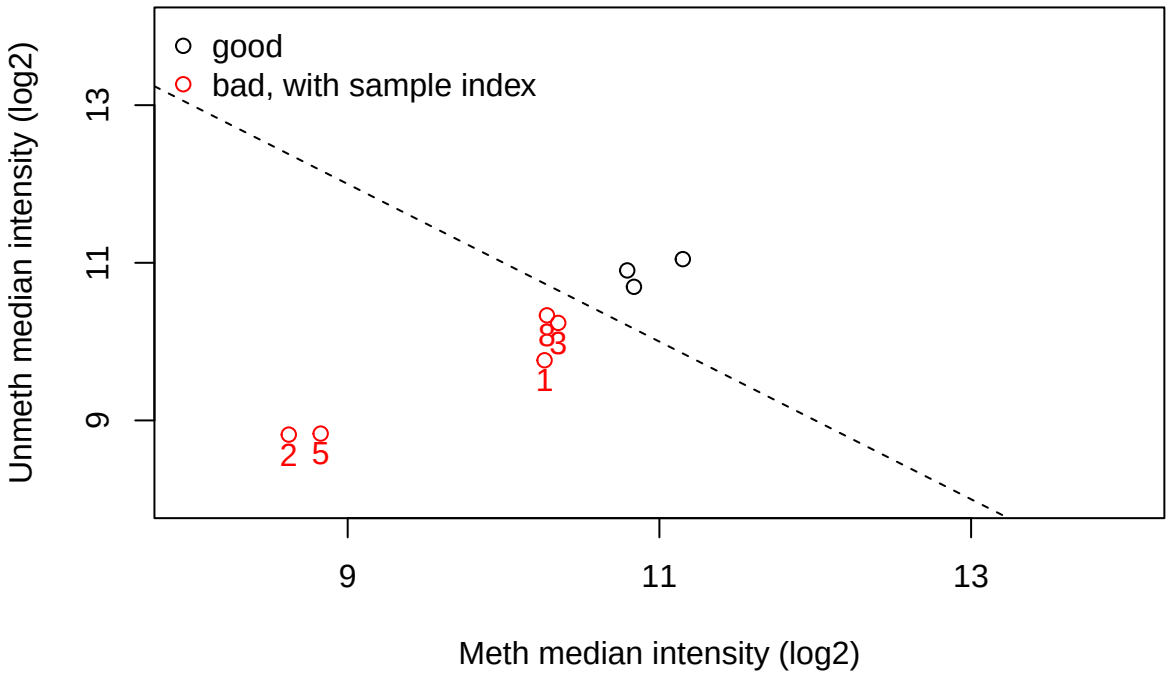
Perform quality checks and provide a brief comment to each step.

2.6.1 a. QCplot

```
qc <- getQC(MSet.raw)
qc
```

```
## DataFrame with 8 rows and 2 columns
##                                       mMed      uMed
##                                  <numeric> <numeric>
## GSM5319592_200121140049_R01C02   10.26327   9.76321
## GSM5319603_3999356129_R02C02      8.62205   8.82018
## GSM5319604_3999547012_R02C01     10.35094  10.23721
## GSM5319607_3999547012_R03C02     10.79360  10.90313
## GSM5319609_3999547016_R02C01      8.82655   8.83289
## GSM5319613_3999547017_R02C01     10.83684  10.69523
## GSM5319615_3999547017_R04C01     11.15038  11.04644
## GSM5319616_3999547017_R06C01     10.27845  10.33316
```

```
plotQC(qc)
```



Comment: A QC plot displays the distribution of methylated and unmethylated signal medians. High median values for both signals indicate high-quality data. In this case, the samples are spread apart, and most show low median values for both signals, suggesting poor quality. Out of the eight samples, only three are considered to be of good quality.

2.6.2 b. Check the intensity of negative controls

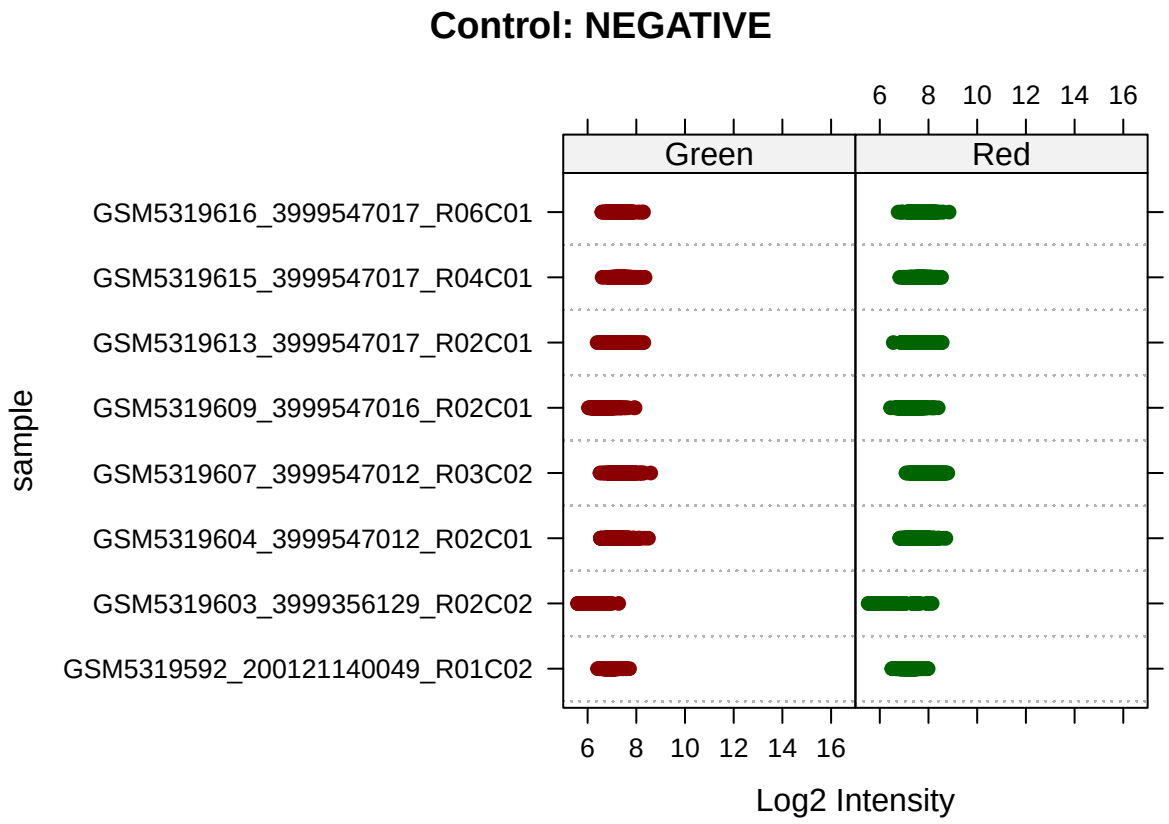
```
NegativeControl <- data.frame(getProbeInfo(RGset, type = "Control"))
table(NegativeControl$Type)
```

```
##
##  BISULFITE CONVERSION I BISULFITE CONVERSION II           EXTENSION
##                      12                       4                   4
##         HYBRIDIZATION              NEGATIVE       NON-POLYMORPHIC
##                       3                     613                   4
##                NORM_A                NORM_C                NORM_G
##                      32                    61                    32
##                NORM_T           RESTORATION          SPECIFICITY I
##                      61                     1                   12
##        SPECIFICITY II              STAINING        TARGET REMOVAL
##                       3                     4                   2
```

```r
head(NegativeControl)
```

```
##    Address      Type  Color  ExtendedType
## 1 27630314  STAINING    Red    DNP (High)
## 2 43603326  STAINING Purple     DNP (Bkg)
## 3 41666334  STAINING  Green Biotin (High)
## 4 34648333  STAINING   Blue  Biotin (Bkg)
## 5 63642461 EXTENSION    Red Extension (A)
## 6 47640365 EXTENSION Purple Extension (T)
```

```r
controlStripPlot(RGset, controls = "NEGATIVE")
```



Comment: In this case, the negative control signals show intensity values (log2) below 10 in both the red and green channels, which is considered acceptable.

2.6.3 c. Calculation of detection p-values

```r
Detection_pvalue <- detectionP(RGset)
save(Detection_pvalue, file = "../data/processed/Detection_pvalue.RData")
Over_Threshold <- Detection_pvalue > 0.05
head(Over_Threshold)
```

```
##        GSM5319592_200121140049_R01C02 GSM5319603_3999356129_R02C02
## cg00050873                          FALSE                          FALSE
## cg00212031                          FALSE                          FALSE
## cg00213748                          FALSE                          FALSE
## cg00214611                          FALSE                          FALSE
## cg00455876                          FALSE                          FALSE
## cg01707559                          FALSE                          FALSE
##         GSM5319604_3999547012_R02C01 GSM5319607_3999547012_R03C02
## cg00050873                        FALSE                          FALSE
## cg00212031                         TRUE                          FALSE
## cg00213748                        FALSE                          FALSE
## cg00214611                         TRUE                          FALSE
## cg00455876                        FALSE                          FALSE
## cg01707559                        FALSE                          FALSE
##         GSM5319609_3999547016_R02C01 GSM5319613_3999547017_R02C01
## cg00050873                        FALSE                          FALSE
## cg00212031                        FALSE                          FALSE
## cg00213748                         TRUE                           TRUE
## cg00214611                         TRUE                           TRUE
## cg00455876                        FALSE                          FALSE
## cg01707559                        FALSE                          FALSE
##         GSM5319615_3999547017_R04C01 GSM5319616_3999547017_R06C01
## cg00050873                         TRUE                          FALSE
## cg00212031                         TRUE                          FALSE
## cg00213748                         TRUE                           TRUE
## cg00214611                         TRUE                           TRUE
## cg00455876                        FALSE                          FALSE
## cg01707559                        FALSE                          FALSE
```

```
table(Over_Threshold)
```

```
## Over_Threshold
##   FALSE    TRUE
## 3877813    6283
```

```
sum(Over_Threshold)
```

```
## [1] 6283
```

```
failed_probe_message <- sprintf(
  "There are %s failed probes with a detection p-value higher than 0.05",
  sum(Over_Threshold)
)
Over_Threshold_Per_Sample <- colSums(Over_Threshold)
Over_Threshold_Per_Sample
```

```
## GSM5319592_200121140049_R01C02   GSM5319603_3999356129_R02C02
##                             202                             306
##   GSM5319604_3999547012_R02C01   GSM5319607_3999547012_R03C02
##                            1156                             366
##   GSM5319609_3999547016_R02C01   GSM5319613_3999547017_R02C01
##                            2735                             460
##   GSM5319615_3999547017_R04C01   GSM5319616_3999547017_R06C01
##                             384                             674
```

```
summary(Over_Threshold)
```

```
##  GSM5319592_200121140049_R01C02 GSM5319603_3999356129_R02C02
##  Mode :logical                  Mode :logical
##  FALSE:485310                   FALSE:485206
##  TRUE :202                      TRUE :306
##  GSM5319604_3999547012_R02C01 GSM5319607_3999547012_R03C02
##  Mode :logical                Mode :logical
##  FALSE:484356                 FALSE:485146
##  TRUE :1156                   TRUE :366
##  GSM5319609_3999547016_R02C01 GSM5319613_3999547017_R02C01
##  Mode :logical                Mode :logical
##  FALSE:482777                 FALSE:485052
##  TRUE :2735                   TRUE :460
##  GSM5319615_3999547017_R04C01 GSM5319616_3999547017_R06C01
##  Mode :logical                Mode :logical
##  FALSE:485128                 FALSE:484838
##  TRUE :384                    TRUE :674
```

```
failed <- data.frame(
  Sample = names(Over_Threshold_Per_Sample),
  n_Failed_Positions = Over_Threshold_Per_Sample
)
print(failed)
```

```
##                                                 Sample
## GSM5319592_200121140049_R01C02 GSM5319592_200121140049_R01C02
## GSM5319603_3999356129_R02C02     GSM5319603_3999356129_R02C02
## GSM5319604_3999547012_R02C01     GSM5319604_3999547012_R02C01
## GSM5319607_3999547012_R03C02     GSM5319607_3999547012_R03C02
## GSM5319609_3999547016_R02C01     GSM5319609_3999547016_R02C01
## GSM5319613_3999547017_R02C01     GSM5319613_3999547017_R02C01
## GSM5319615_3999547017_R04C01     GSM5319615_3999547017_R04C01
## GSM5319616_3999547017_R06C01     GSM5319616_3999547017_R06C01
##                                n_Failed_Positions
## GSM5319592_200121140049_R01C02                202
## GSM5319603_3999356129_R02C02                  306
```

```
## GSM5319604_3999547012_R02C01                  1156
## GSM5319607_3999547012_R03C02                   366
## GSM5319609_3999547016_R02C01                  2735
## GSM5319613_3999547017_R02C01                   460
## GSM5319615_3999547017_R04C01                   384
## GSM5319616_3999547017_R06C01                   674
```

```
print(failed_probe_message)
```

```
## [1] "There are 6283 failed probes with a detection p-
value higher than 0.05"
```

Comment: There are 6283 failed probes with a detection p-value higher than 0.05 and the number of failed positions ranges from 202-2735 per sample.

2.7 STEP 6. Calculation of Beta and M values, Plot Densities

Calculate raw beta and M values and plot the densities of mean methylation values, dividing the samples in CTRL and DIS (suggestion: subset the beta and M values matrixes in order to retain CTRL or DIS subjects and apply the function mean to the 2 subsets).

Data preparation

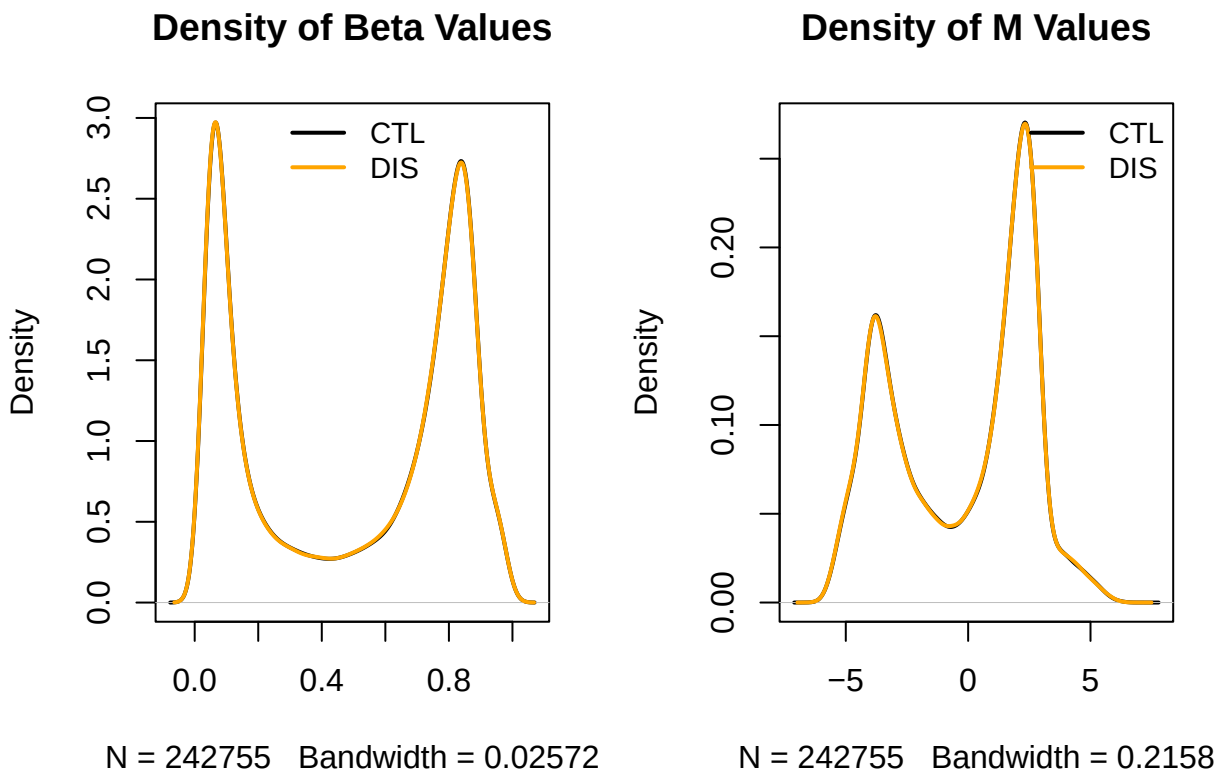
```
beta <- getBeta(MSet.raw)
M <- getM(MSet.raw)
beta_df <- data.frame(beta)
M_df <- data.frame(M)

pheno <- read.csv("../data/raw/SampleSheet_Report_II.csv", header = TRUE,
 ↪  stringsAsFactors = TRUE)

SampleSheet$Group
```

```
## [1] "CTRL" "DIS"  "DIS"  "CTRL" "DIS"  "DIS"  "CTRL" "CTRL"
```

```
beta_ctrl <- beta_df[SampleSheet$Group == "CTRL",]
beta_dis <- beta_df[SampleSheet$Group == "DIS",]
M_ctrl <- M_df[SampleSheet$Group == "CTRL",]
M_dis <- M_df[SampleSheet$Group == "DIS",]

mean_of_beta_ctrl <- apply(beta_ctrl, 1, mean, na.rm = TRUE)
mean_of_beta_dis <- apply(beta_dis, 1, mean, na.rm = TRUE)
mean_of_M_ctrl <- apply(M_ctrl, 1, mean, na.rm = TRUE)
mean_of_M_dis <- apply(M_dis, 1, mean, na.rm = TRUE)
```

Plotting the density of mean methylation values.

```r
plot_density <- function() {
  par(mfrow = c(1, 2), mar = c(5, 4, 4, 2))
  plot(density(mean_of_beta_ctrl, na.rm = TRUE), main = "Density of Beta
   ↪  Values",
       col = "black", lwd = 2)
  lines(density(mean_of_beta_dis, na.rm = TRUE), col = "orange", lwd = 2)
  legend("top", legend = c("CTL", "DIS"),
         col = c("black", "orange"), lwd = 2,
         cex = 0.9, bty = "n", inset = c(-0.01, -0.01))
  plot(density(mean_of_M_ctrl, na.rm = TRUE), main = "Density of M Values",
   ↪  col = "black", lwd = 2)
  lines(density(mean_of_M_dis, na.rm = TRUE), col = "orange", lwd = 2)
  legend("topright", legend = c("CTL", "DIS"),
         col = c("black", "orange"), lwd = 2,
         cex = 0.9, bty = "n", inset = c(-0.01, -0.01))
}
plot_density()
```



Comment: Based on the density plots of mean methylation, no clear differences were observed between the control and disease groups. This could suggest that there are no significant differences in methylation levels between the two. However, further statistical analyses are necessary to better interpret the data and potentially uncover subtle or localized differences.

2.8 STEP 7. Data normalization with preprocessNoob

Normalize the data using preprocessNoob and compare raw data and normalized data. Produce a plot with 6 panels in which, for both raw and normalized data, you show the density plots of beta mean values according to the chemistry of the probes, the density plot of beta standard deviation values according to the chemistry of the probes and the boxplot of beta values.

Subset beta values by probe type

```
dfI <- Illumina450Manifest_clean[Illumina450Manifest_clean$
 ↪  Infinium_Design_Type == "I",]
dfI <- droplevels(dfI)
dfII <- Illumina450Manifest_clean[Illumina450Manifest_clean$
 ↪  Infinium_Design_Type == "II",]
dfII <- droplevels(dfII)

beta_I <- beta[rownames(beta) %in% dfI$IlmnID,]
beta_II <- beta[rownames(beta) %in% dfII$IlmnID,]

mean_of_beta_I <- apply(beta_I, 1, mean)
mean_of_beta_II <- apply(beta_II, 1, mean)
d_mean_of_beta_I <- density(mean_of_beta_I, na.rm = TRUE)
d_mean_of_beta_II <- density(mean_of_beta_II, na.rm = TRUE)
```

Calculation of the densities of the standard deviations This can be calculated using the function sd():

```
sd_of_beta_I <- apply(beta_I, 1, sd, na.rm = TRUE)
sd_of_beta_II <- apply(beta_II, 1, sd, na.rm = TRUE)
d_sd_of_beta_I <- density(sd_of_beta_I,)
d_sd_of_beta_II <- density(na.omit(sd_of_beta_II))
```

Use of preprocessNoob Preparation of the data.

```
preprocessNoob_results <- preprocessNoob(RGset)
beta_preprocessNoob <- getBeta(preprocessNoob_results)
save(beta_preprocessNoob, file =
 ↪  "../data/processed/beta_preprocessNoob.RData")
```

Divide normalized beta matrix by probe type and calculate statistics.

```
beta_preprocessNoob_I <- beta_preprocessNoob[rownames(beta_preprocessNoob)
 ↪  %in% dfI$IlmnID,]
beta_preprocessNoob_II <-
 ↪  beta_preprocessNoob[rownames(beta_preprocessNoob) %in% dfII$IlmnID,]
mean_of_beta_preprocessNoob_I <- apply(beta_preprocessNoob_I, 1, mean)
mean_of_beta_preprocessNoob_II <- apply(beta_preprocessNoob_II, 1, mean)
d_mean_of_beta_preprocessNoob_I <- density(mean_of_beta_preprocessNoob_I,
 ↪  na.rm = TRUE)
d_mean_of_beta_preprocessNoob_II <-
 ↪  density(mean_of_beta_preprocessNoob_II, na.rm = TRUE)
```

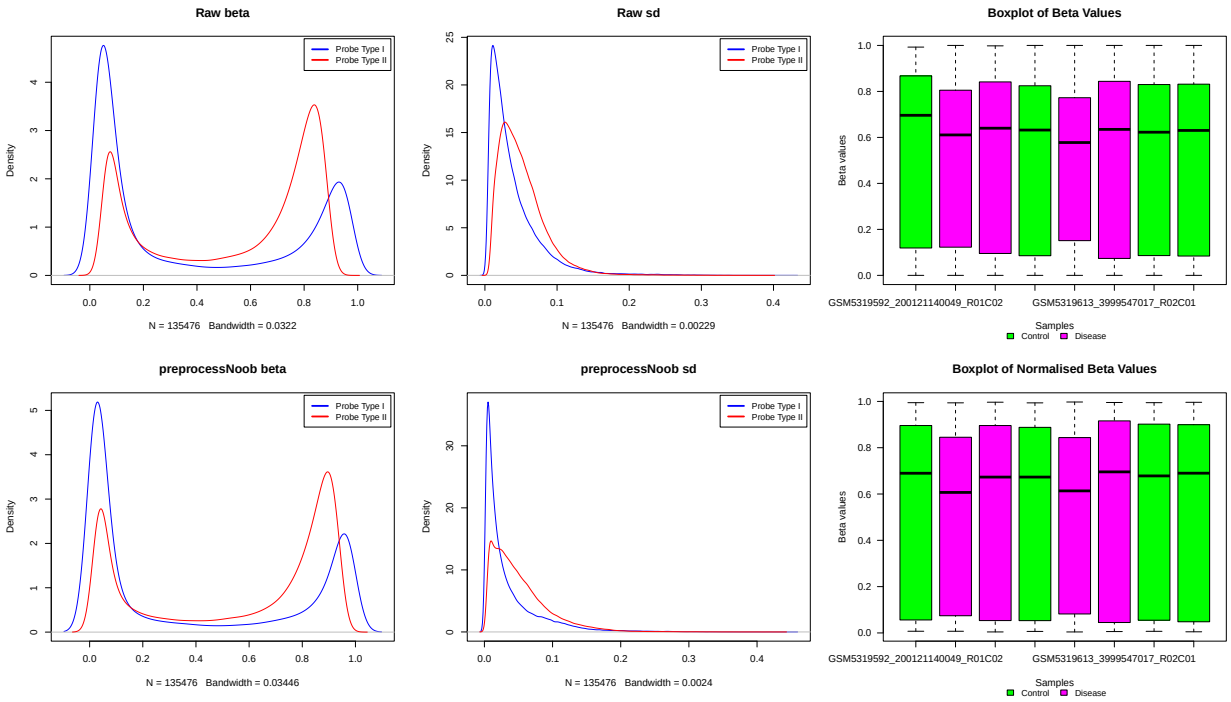
```
sd_of_beta_preprocessNoob_I <- apply(beta_preprocessNoob_I, 1, sd)
sd_of_beta_preprocessNoob_II <- apply(beta_preprocessNoob_II, 1, sd)
d_sd_of_beta_preprocessNoob_I <- density(sd_of_beta_preprocessNoob_I,
 ↪  na.rm = TRUE)
d_sd_of_beta_preprocessNoob_II <- density(sd_of_beta_preprocessNoob_II,
 ↪  na.rm = TRUE)
```

Boxplot visualization by group ### Comparison of Raw and Normalized Beta Values

This section displays the distribution of beta values before and after normalization with `preprocessNoob`. Each row of plots corresponds to raw and normalized data, respectively.

```
par(mfrow = c(2, 3))

# Plot 1
plot(d_mean_of_beta_I, col = "blue", main = "Raw beta")
lines(d_mean_of_beta_II, col = "red")
legend("topright",
       legend = c("Probe Type I", "Probe Type II"),
       col = c("blue", "red"),
       lwd = 2,
       cex = 0.9,
       inset = c(0.01, 0.01))

# Plot 2
plot(d_sd_of_beta_I, col = "blue", main = "Raw sd")
lines(d_sd_of_beta_II, col = "red")
legend("topright",
       legend = c("Probe Type I", "Probe Type II"),
       col = c("blue", "red"),
       lwd = 2,
       cex = 0.9,
       inset = c(0.01, 0.01))

# Plot 3
boxplot(beta,
        las=1,
        col=c("green", "magenta")[pheno$Group],
        ylab='Beta values',
        xlab='Samples',
        main='Boxplot of Beta Values')
legend("bottom",
       legend = c("Control", "Disease"),
       fill = c("green", "magenta"),
       horiz = TRUE,
       bty = 'n',
       inset = -0.25,
       xpd = TRUE,
       cex = 0.9)

# Plot 4
plot(d_mean_of_beta_preprocessNoob_I, col = "blue", main = "preprocessNoob
 ↪  beta")
```

```
lines(d_mean_of_beta_preprocessNoob_II, col = "red")
legend("topright",
       legend = c("Probe Type I", "Probe Type II"),
       col = c("blue", "red"),
       lwd = 2,
       cex = 0.9,
       inset = c(0.01, 0.01))

# Plot 5
plot(d_sd_of_beta_preprocessNoob_I, col = "blue", main = "preprocessNoob
 ↪  sd")
lines(d_sd_of_beta_preprocessNoob_II, col = "red")
legend("topright",
       legend = c("Probe Type I", "Probe Type II"),
       col = c("blue", "red"),
       lwd = 2,
       cex = 0.9,
       inset = c(0.01, 0.01))

# Plot 6
boxplot(beta_preprocessNoob,
        las=1,
        col=c("green", "magenta")[pheno$Group],
        ylab='Beta values',
        xlab='Samples',
        main='Boxplot of Normalised Beta Values')
legend("bottom",
       legend = c("Control", "Disease"),
       fill = c("green", "magenta"),
       horiz = TRUE,
       bty = 'n',
       inset = -0.25,
       xpd = TRUE,
       cex = 0.9)
```



Comment: preprocessNoob is a normalization method designed to correct background fluorescence and dye bias. It is known as a "light" normalization technique, meaning it applies minimal transformation to the data while still addressing key technical artifacts.

After applying it, the density plots show that the probe type distributions are slightly more aligned, indicating improved consistency. However, the boxplots reveal a larger interquartile range and non-uniform medians across samples, suggesting that some variability remains uncorrected.

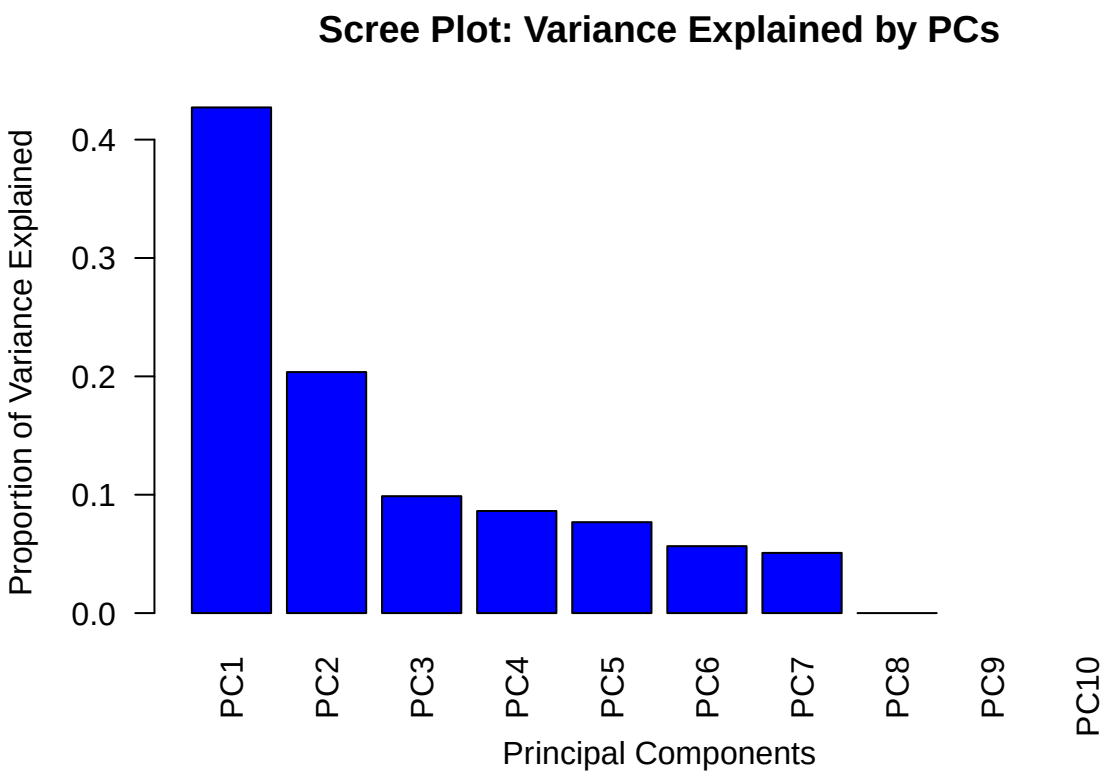
Given these results, preprocessNoob may not be the most appropriate normalization method for this dataset. Although it effectively corrects background noise and dye bias, it does not sufficiently address sample variability. A better alternative could be preprocessQuantile, which applies stratified quantile normalization specifically tailored for Illumina methylation microarrays—the platform used for this dataset.

2.9 STEP 8. Principal Component Analysis

Perform a PCA on the matrix of normalized beta values generated in step 7.

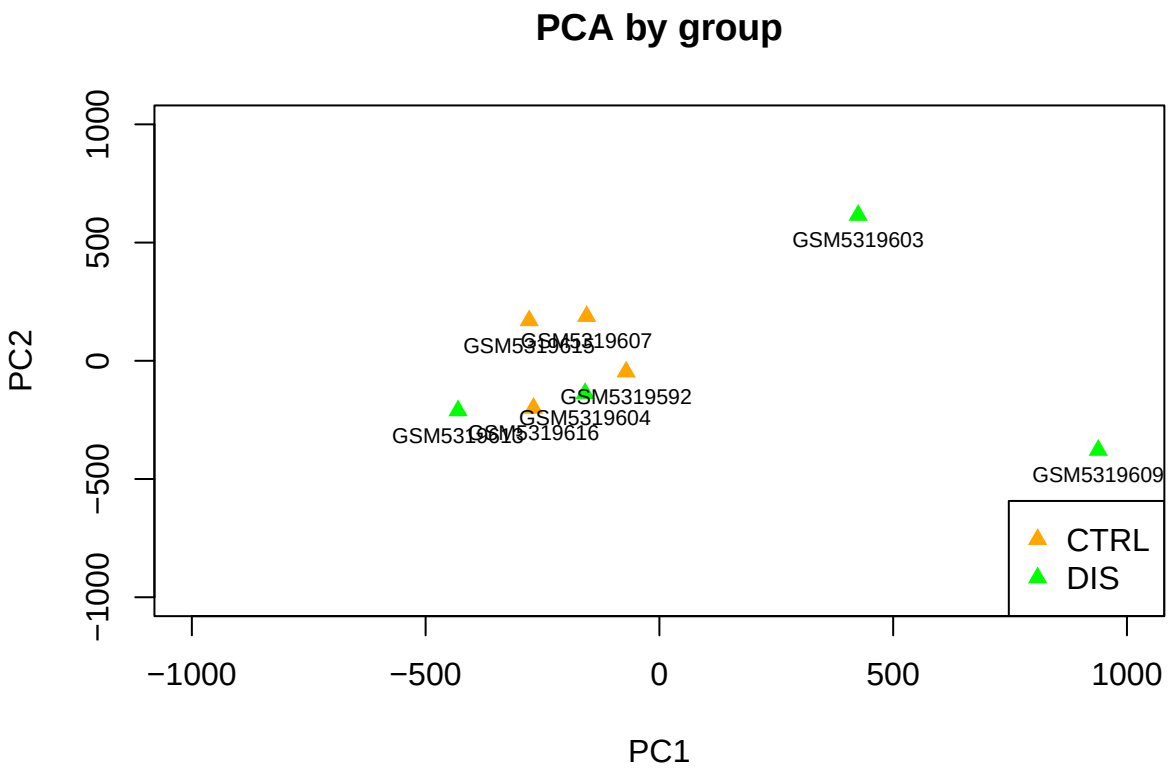
Perform PCA on normalized beta values

```
pca_results <- prcomp(t(beta_preprocessNoob), scale = TRUE)
pca_var <- pca_results$sdev^2
pca_var_explained <- pca_var / sum(pca_var)
barplot(pca_var_explained[1:10],
        names.arg = paste0("PC", 1:10),
        col = "blue",
        main = "Scree Plot: Variance Explained by PCs",
        xlab = "Principal Components",
        ylab = "Proportion of Variance Explained",
        las = 2)
```



Plot PCA by Group

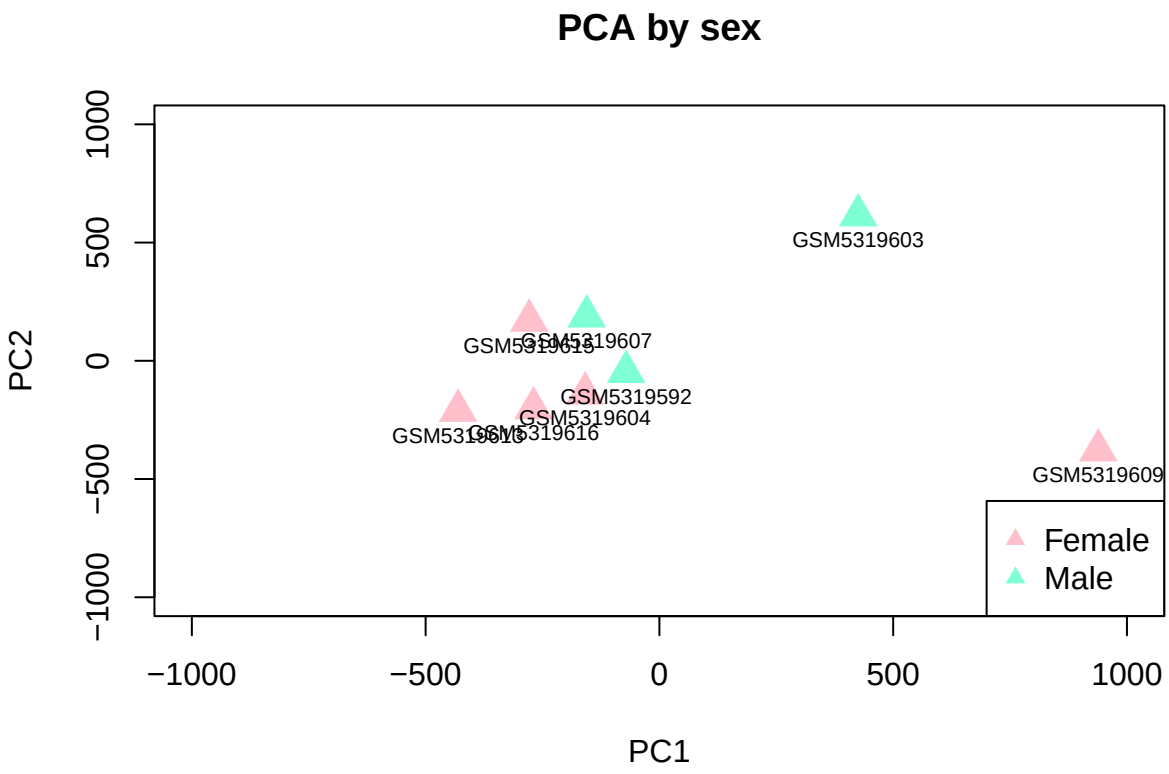
```
par(mfrow = c(1, 1))
palette(c('orange', 'green'))
plot(pca_results$x[, 1], pca_results$x[, 2], cex = 1, pch = 17, col =
 ↪  pheno$Group,
     main = 'PCA by group', xlab = "PC1", ylab = "PC2", xlim = c(-1000,
      ↪  1000), ylim = c(-1000, 1000))
text(pca_results$x[, 1], pca_results$x[, 2], labels = (pheno$SampleID),
 ↪  cex = 0.7, pos = 1)
legend('bottomright', legend = levels(pheno$Group), col =
 ↪  c(1:nlevels(pheno$Group)), pch = 17)
```



Comment: Unfortunately, the PCA analysis does not show a clear separation between the control and disease groups. While the control samples cluster closely together, the disease samples are more dispersed across the plot.

Plot PCA by Sex

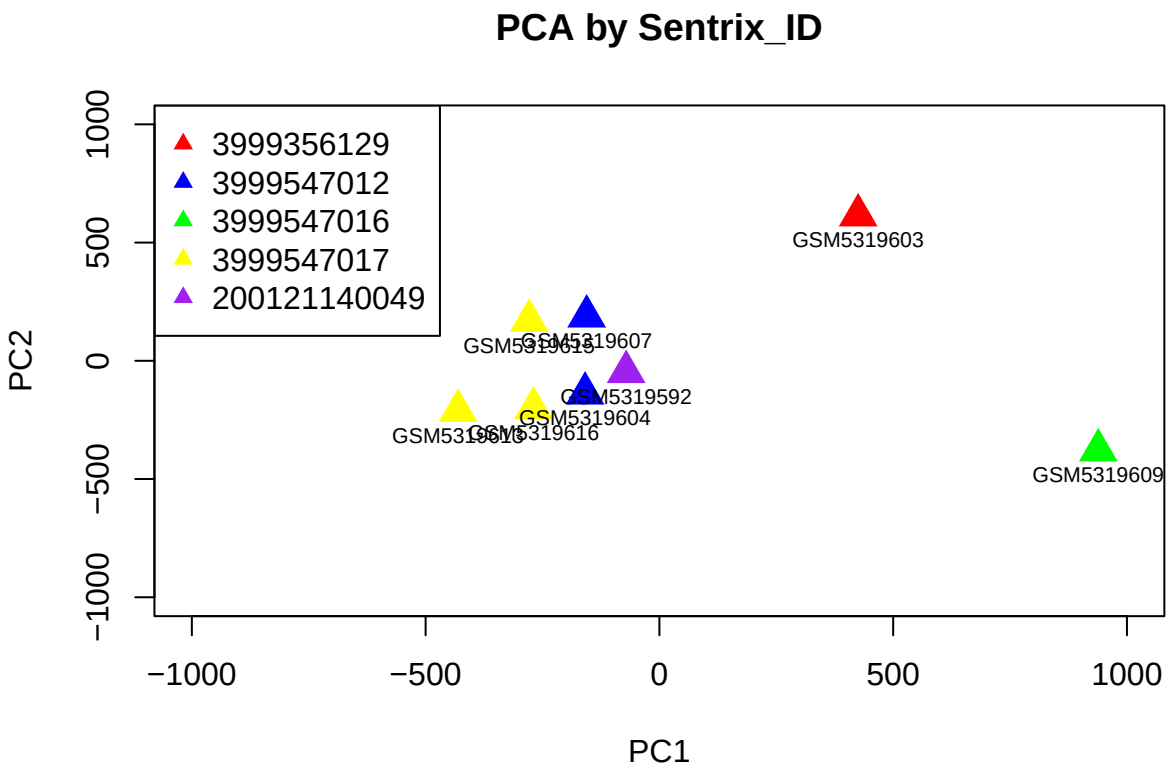
```
pheno$Sex
pheno$Sex <- as.factor(pheno$Sex)
palette(c('pink', 'aquamarine'))
plot(pca_results$x[, 1], pca_results$x[, 2], cex = 2, pch = 17, col =
 ↪  pheno$Sex,
     main = 'PCA by sex', xlab = "PC1", ylab = "PC2", xlim = c(-1000,
      ↪  1000), ylim = c(-1000, 1000))
text(pca_results$x[, 1], pca_results$x[, 2], labels = (pheno$SampleID),
 ↪  cex = 0.7, pos = 1)
legend('bottomright', legend = levels(pheno$Sex), col =
 ↪  c(1:nlevels(pheno$Sex)), pch = 17)
```



Comment: In this PCA analysis, the female samples cluster closely together, with the exception of one outlier (GMS5319609), but there is not a clear separation based on sex.

Plot PCA by Sentrix_ID

```
pheno$Sentrix_ID
pheno$Sentrix_ID <- as.factor(pheno$Sentrix_ID)
palette(c('red', 'blue', 'green', 'yellow', 'purple'))
plot(pca_results$x[, 1], pca_results$x[, 2], cex = 2, pch = 17, col =
 ↪  pheno$Sentrix_ID,
     main = 'PCA by Sentrix_ID', xlab = "PC1", ylab = "PC2", xlim =
      ↪  c(-1000, 1000),
     ylim = c(-1000, 1000))
text(pca_results$x[, 1], pca_results$x[, 2], labels = (pheno$SampleID),
 ↪  cex = 0.7, pos = 1)
legend('topleft', legend = levels(pheno$Sentrix_ID),
       col = c(1:nlevels(pheno$Sentrix_ID)), pch = 17)
```



Comment: Five batches were identified. The PCA shows clustering; however, there are almost as many batches as there are samples. This makes it difficult to determine whether the variability observed in the experimental results is due to batch effects. A larger sample size would be needed to confirm this.

2.10 STEP 9. Differential Methylation Analysis

Using the matrix of normalized beta values generated in step 7, identify differentially methylated probes between CTRL and DIS groups using a t-test.

Example for a single probe

```
t_test <- t.test(beta_preprocessNoob[1, ] ~ pheno$Group)
t_test
t_test$p.value
```

Define a function to apply test to all probes

```
My_ttest_function <- function(x) {
  t_test <- t.test(x ~ pheno$Group)
  return(t_test$p.value)
}
```

Apply the function to all probes

```
pValues_ttest <- apply(beta_preprocessNoob, 1, My_ttest_function)
```

Create data.frame with beta values and p-values

```
final_ttest <- data.frame(beta_preprocessNoob, pValues_ttest)
```

Order probes by p-value

```
final_ttest <- final_ttest[order(final_ttest$pValues_ttest), ]
head(final_ttest)
```

Filter for significant probes (p < 0.05)

```
significant_ttest <- final_ttest[final_ttest$pValues_ttest <= 0.05, ]
```

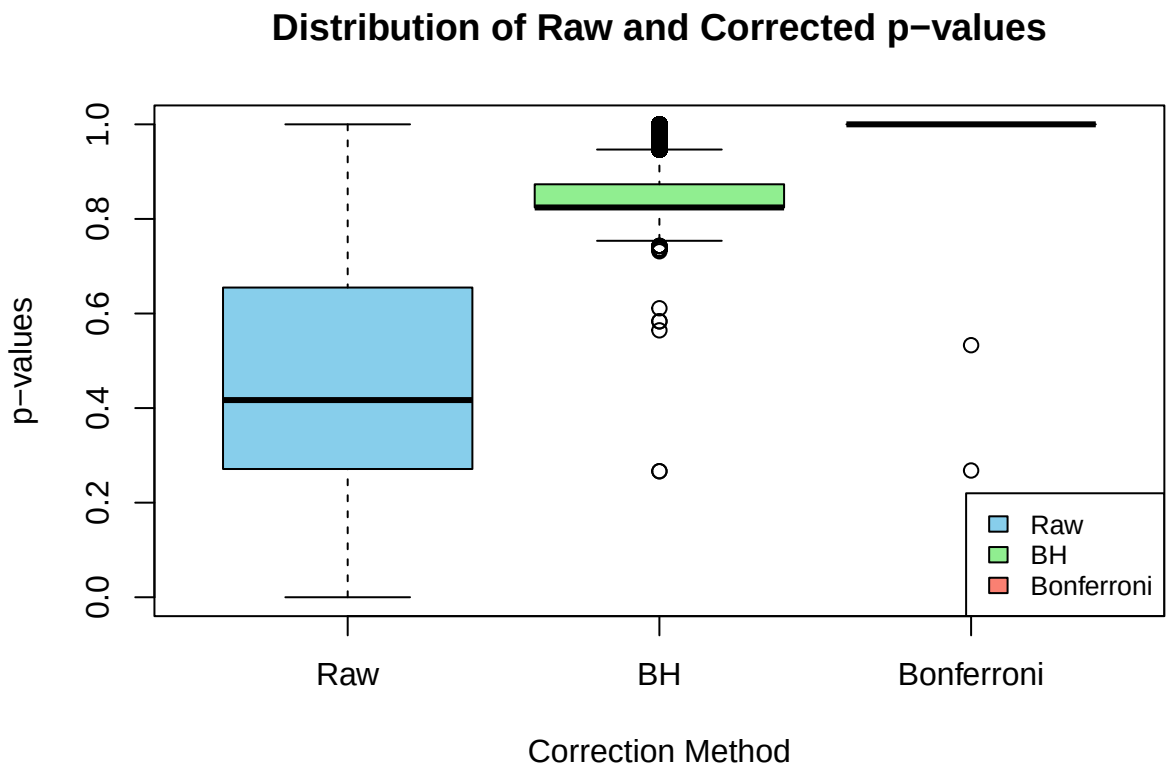
2.11 STEP 10. Multiple Test Correction

Apply multiple test correction and set a significant threshold of 0.05

```
corrected_pValues_BH <- p.adjust(final_ttest$pValues_ttest, "BH")
corrected_pValues_Bonf <- p.adjust(final_ttest$pValues_ttest,
 ↪  "bonferroni")
final_ttest_corrected <- data.frame(final_ttest, corrected_pValues_BH,
 ↪  corrected_pValues_Bonf)
head(final_ttest_corrected)
```

Visualize distributions of p-values and corrected p-values

```
colnames(final_ttest_corrected)
box_colors <- c("skyblue", "lightgreen", "salmon")
boxplot(final_ttest_corrected[, 9:11],
        main = "Distribution of Raw and Corrected p-values",
        ylab = "p-values",
        xlab = "Correction Method",
        col = box_colors,
        names = c("Raw", "BH", "Bonferroni"))
legend("bottomright",
       legend = c("Raw", "BH", "Bonferroni"),
       fill = box_colors,
       cex = 0.8)
```



Determining the amount of probes that remain after the MTC

```
dim(final_ttest_corrected[final_ttest_corrected$pValues_ttest <= 0.05, ])
dim(final_ttest_corrected[final_ttest_corrected$corrected_pValues_BH <=
 ↪  0.05, ])
dim(final_ttest_corrected[final_ttest_corrected$corrected_pValues_Bonf <=
 ↪  0.05, ])
```

Comment: 11,928 probes with a nominal p-value below 0.05 before applying any multiple testing correction were identified. However, after applying Bonferroni and Benjamini-Hochberg (BH) corrections, no probes remained significant at the 0.05 threshold. This suggests that while many probes appeared significant before correction, these results may be due to chance, highlighting the importance of correcting for multiple testing in high-dimensional data like DNA methylation. Furthermore, this could also be an additional sign that the normalization procedure applied to data was not appropriate. Most likely, applying the preprocessQuantile function and then the t-test on data, the multiple testing correction may highlight significant probes.

2.12 STEP 11. Graphic the results of the differential methylation analysis.

Produce a volcano plot and a Manhattan plot of the results of differential methylation analysis.

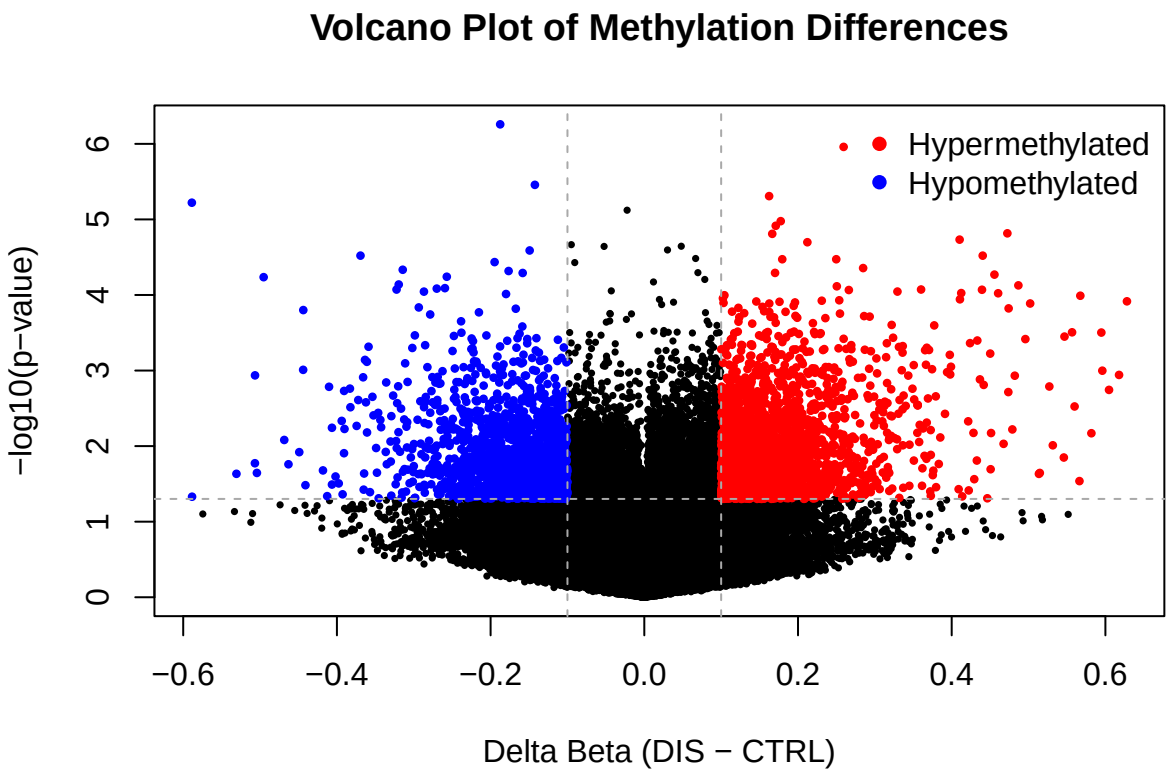
a. Volcano plot

Preparation of the data

```
beta_first <- final_ttest_corrected[, 1:8]
beta_first_ctrl <- beta_first[, pheno$Group == "CTRL"]
beta_first_dis <- beta_first[, pheno$Group == "DIS"]
mean_beta_first_ctrl <- apply(beta_first_ctrl, 1, mean)
mean_beta_first_dis <- apply(beta_first_dis, 1, mean)
delta_first <- mean_beta_first_dis - mean_beta_first_ctrl
toVolcPlot <- data.frame(
  delta_beta = delta_first,
  negLog10p = -log10(final_ttest_corrected$pValues_ttest)
)
```

Volcano plot to highlight hypermethylated (Delta Beta > 0.1 & p < 0.05) and hypomethylated (Delta Beta < -0.1 & p < 0.05) targets

```
plot(toVolcPlot$delta_beta, toVolcPlot$negLog10p,
     pch = 16, cex = 0.5,
     xlab = "Delta Beta (DIS - CTRL)",
     ylab = "-log10(p-value)",
     main = "Volcano Plot of Methylation Differences")
hyper <- which(toVolcPlot$delta_beta > 0.1 &
 ↪  final_ttest_corrected$pValues_ttest < 0.05)
points(toVolcPlot$delta_beta[hyper],
       toVolcPlot$negLog10p[hyper],
       col = "red", pch = 16, cex = 0.6)
hypo <- which(toVolcPlot$delta_beta < -0.1 &
 ↪  final_ttest_corrected$pValues_ttest < 0.05)
points(toVolcPlot$delta_beta[hypo],
       toVolcPlot$negLog10p[hypo],
       col = "blue", pch = 16, cex = 0.6)

abline(h = -log10(0.05), col = "darkgray", lty = 2)   # p = 0.05
abline(v = c(-0.1, 0.1), col = "darkgray", lty = 2)   # Delta Beta = ±0.1
legend("topright", legend = c("Hypermethylated", "Hypomethylated"),
       col = c("red", "blue"), pch = 16, bty = "n")
```

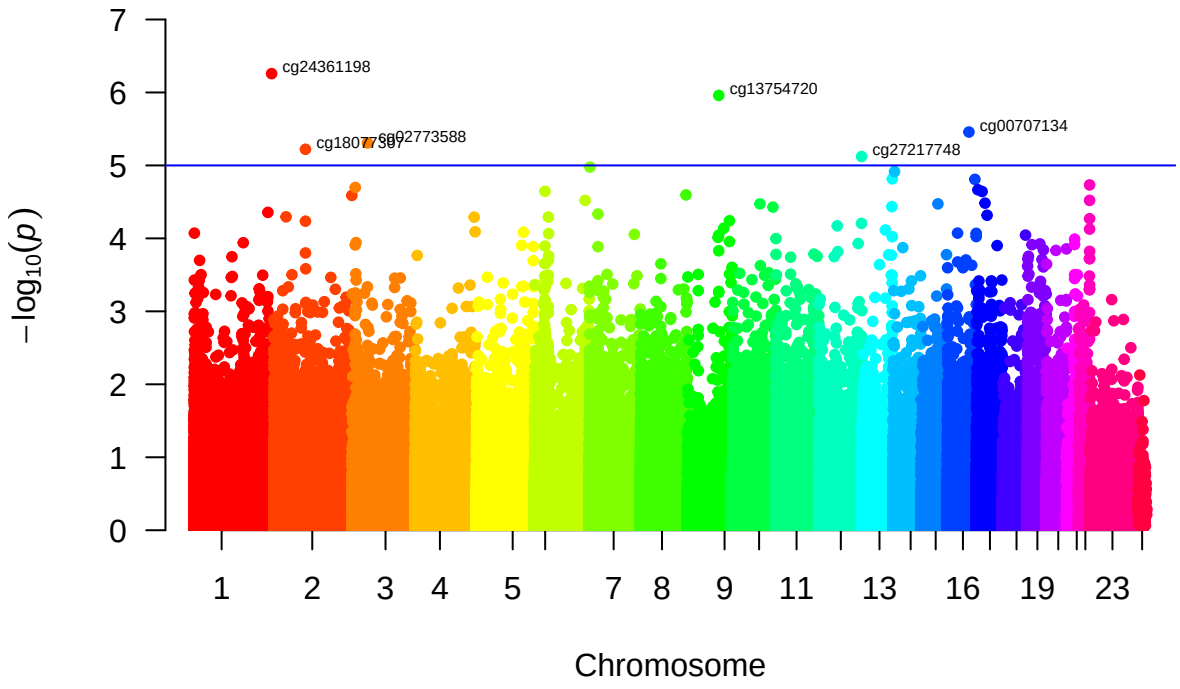


b. Manhattan plot

Annotate dataframe with genome annotation for each CpG probe.

```
final_ttest_corrected <- data.frame(rownames(final_ttest_corrected),
 ↪  final_ttest_corrected)
colnames(final_ttest_corrected)[1] <- "IlmnID"
final_ttest_corrected_clean <- final_ttest_corrected[,
 ↪  !duplicated(colnames(final_ttest_corrected))]
final_ttest_corrected_annotated <- merge(final_ttest_corrected_clean,
 ↪  Illumina450Manifest_clean,
                                         by = "IlmnID")
input_Manhattan <- final_ttest_corrected_annotated[colnames
 ↪  (final_ttest_corrected_annotated) %in%
                      c("IlmnID", "CHR", "MAPINFO", "pValues_ttest")]
order_chr <- c("1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11",
 ↪  "12", "13", "14", "15",
               "16", "17", "18", "19", "20", "21", "22", "X", "Y")
input_Manhattan$CHR <- factor(input_Manhattan$CHR, levels = order_chr)
input_Manhattan$CHR <- as.numeric(input_Manhattan$CHR)
manhattan(input_Manhattan,
          snp = "IlmnID",
          chr = "CHR",
          bp = "MAPINFO",
          p = "pValues_ttest",
          annotatePval = 0.00001,
```

```
          col = rainbow(24))
```



2.13 STEP 12. Heatmap

Produce an heatmap of the top 100 significant, differentially methylated probes.

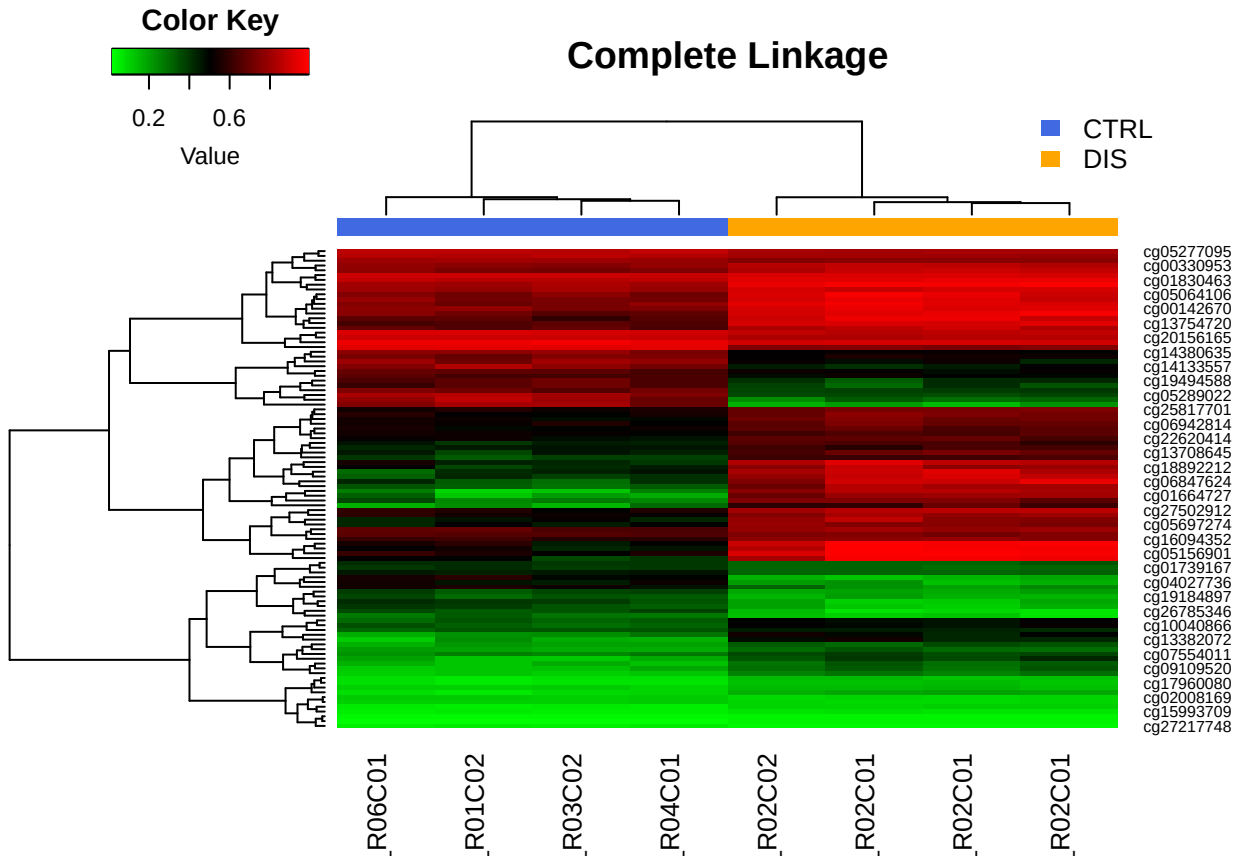
Define colorbar from phenotype group and custom color palette

```
colorbar <- pheno$Group
colorbar <- as.character(factor(colorbar, labels = c("royalblue",
 ↪  "orange")))
input_heatmap = as.matrix(final_ttest[1:100, 1:8])
col2 = colorRampPalette(c("green", "black", "red"))(100)
```

Complete linkage

```
heatmap.2(input_heatmap,
          col = col2,
          Rowv = TRUE,
          Colv = TRUE,
          dendrogram = "both",
          key = TRUE,
          ColSideColors = colorbar,
```

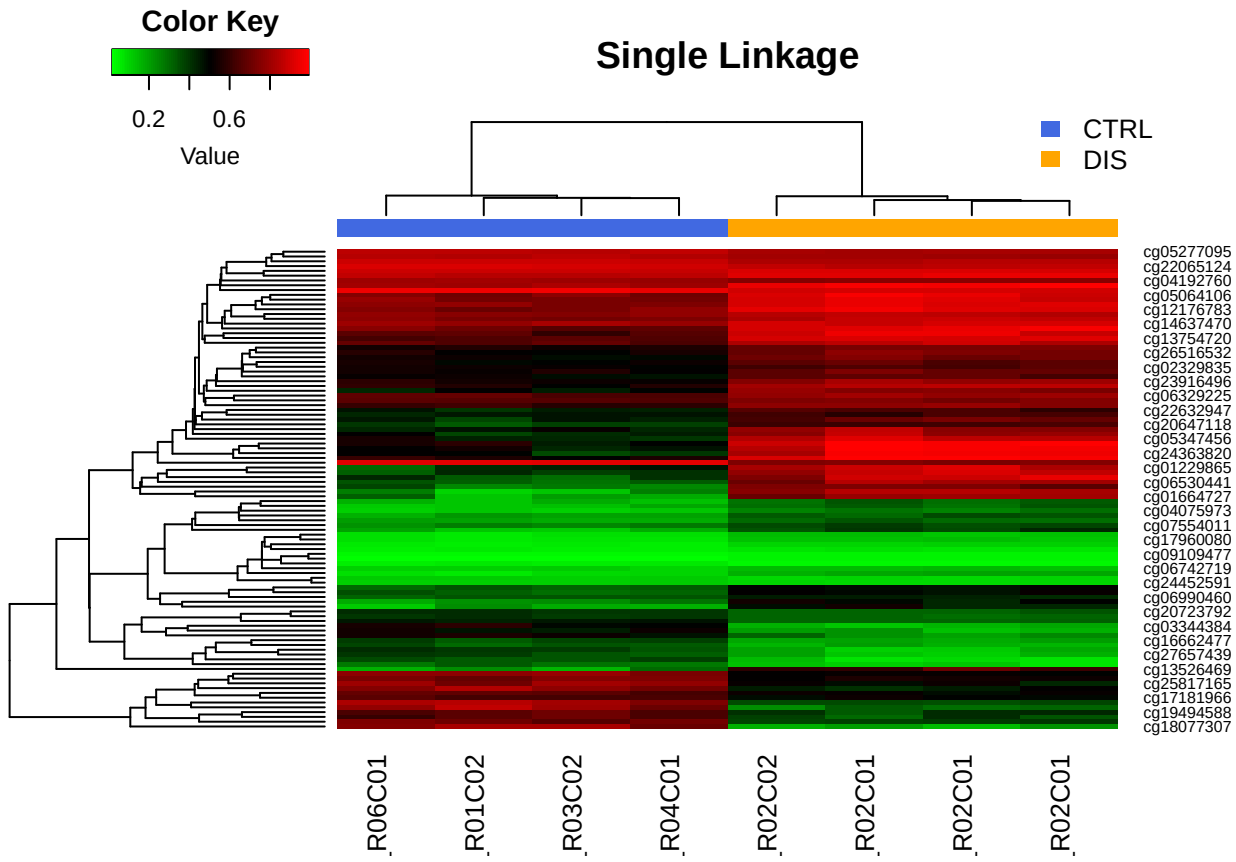
```
          density.info = "none",
          trace = "none",
          scale = "none",
          symm = FALSE,
          main = "Complete Linkage")
legend("topright", legend = c("CTRL", "DIS"), fill = c("royalblue",
 ↪  "orange"), border = NA, bty = "n",
       cex = 0.8, xpd = TRUE, inset = c(0, -0.10))
```



Single linkage

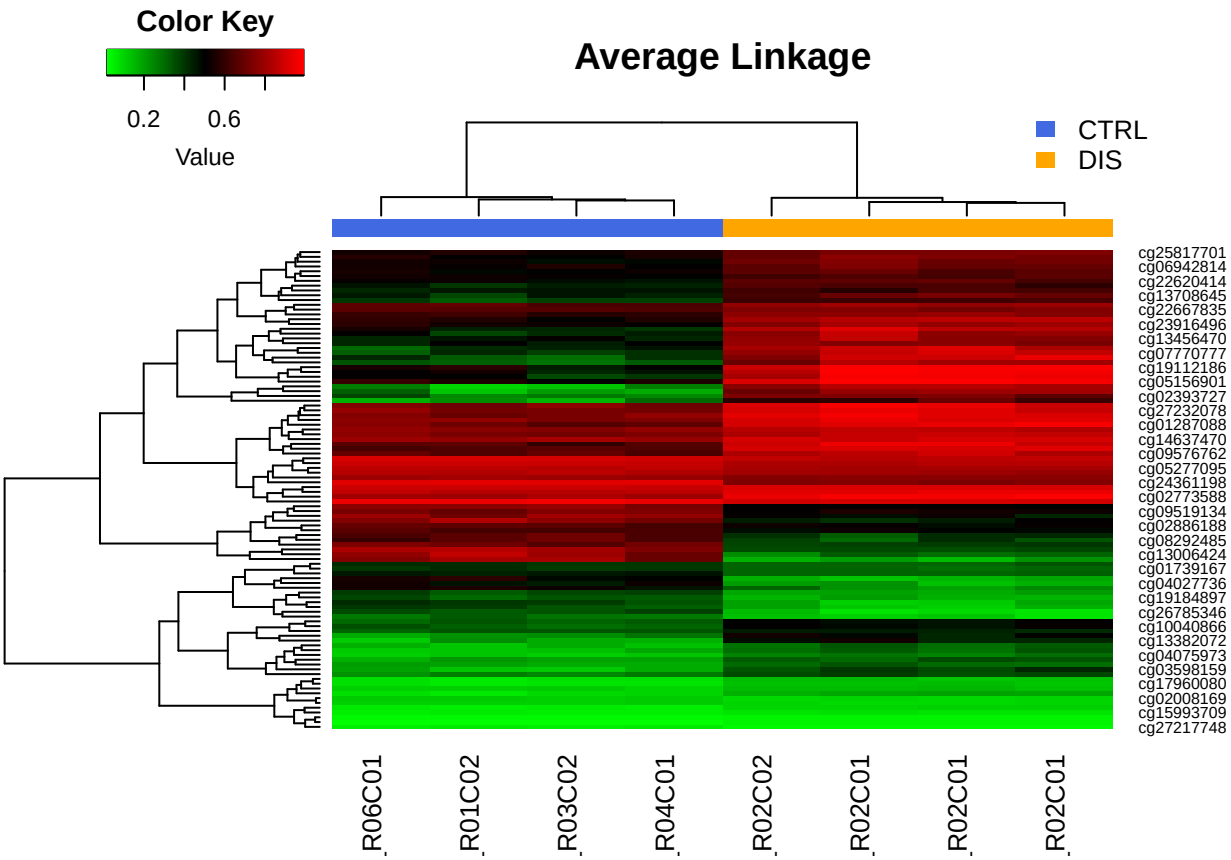
```
heatmap.2(input_heatmap,
          col = col2,
          Rowv = TRUE,
          Colv = TRUE,
          hclustfun = function(x) hclust(x, method = "single"),
          dendrogram = "both",
          key = TRUE,
          ColSideColors = colorbar,
          density.info = "none",
          trace = "none",
          scale = "none",
          symm = FALSE,
          main = "Single Linkage")
legend("topright", legend = c("CTRL", "DIS"), fill = c("royalblue",
 ↪  "orange"), border = NA, bty = "n",
```

```
       cex = 0.8, xpd = TRUE, inset = c(0, -0.10))
```



Average linkage

```
heatmap.2(input_heatmap,
          col = col2,
          Rowv = TRUE,
          Colv = TRUE,
          hclustfun = function(x) hclust(x, method = "average"),
          dendrogram = "both",
          key = TRUE,
          ColSideColors = colorbar,
          density.info = "none",
          trace = "none",
          scale = "none",
          symm = FALSE,
          main = "Average Linkage")
legend("topright", legend = c("CTRL", "DIS"), fill = c("royalblue",
 ↪  "orange"), border = NA, bty = "n",
       cex = 0.8, xpd = TRUE, inset = c(0, -0.10))
```


Color Key
0.2
0.6
Value
Average Linkage
CTRL
DIS
cg25817701
cg06942814
cg22620414
cg13708645
cg22667835
cg23916496
cg13456470
cg07770777
cg19112186
cg05156901
cg02393727
cg27232078
cg01287088
cg14637470
cg09576762
cg05277095
cg24361198
cg02773588
cg09519134
cg02886188
cg08292485
cg13006424
cg01739167
cg04027736
cg19184897
cg26785346
cg10040866
cg13382072
cg04075973
cg03598159
cg17960080
cg02008169
cg15993709
cg27217748
R06C01
R01C02
R03C02
R04C01
R02C02
R02C01
R02C01
R02C01