# Epigenomics for Social Scientists

02 quality control of datasets

Kelly Bakulski, Shan Andrews, John Dou, Jonah Fisher, Erin Ware

Last compiled on July 29, 2021

## Setup

### Load relevant packages

This should be done whenever you start a new r session. For this script we add several more functions that are useful for data processing, quality control, and figure plotting.

```r
library(minfi)
library(MASS)
library(abind)
library(sva)
library(Hmisc)
library(ggplot2)
library(ggsci)
library(tidyverse)
library(here)
```

## Minfi preprocessing

There are various quality control steps that need to be executed before we can say that the data is clean. Initially we will use the minfi package to process some of our large data sets and understand the quality of the data.

```r
load(here("Data", "RGset.rda"))
pd <- data.frame(RGset@colData@listData)
pd$sex <- pd$gender
pd$gender <- NULL
```

### Extract methylated and unmethylated signals

Here we extract the signal intensity from each sample. Low signal intensity is sign of a poor sample. Additionally, we can stratify signal intensity by variables such as batch to get an idea about the technical noise in our sample.

```r
# MethylSet (Mset) contains metylated and unmethylated signals made using preprocessRaw()
rawMSet <- preprocessRaw(RGset)
```

```
## Loading required package: IlluminaHumanMethylation450kmanifest
```

```r
rawMSet
```

```
## class: MethylSet
## dim: 485512 17
## metadata(0):
## assays(2): Meth Unmeth
## rownames(485512): cg00050873 cg00212031 ... ch.22.47579720R
##   ch.22.48274842R
## rowData names(0):
## colnames(17): GSM1051870_7766130158_R03C02 GSM1052024_5730053010_R01C02
##   ... GSM1052032_5730053011_R06C01 GSM1052037_5730053011_R06C02
## colData names(11): GEOID celltype ... Batch filenames
## Annotation
##   array: IlluminaHumanMethylation450k
##   annotation: ilmn12.hg19
## Preprocessing
##   Method: Raw (no normalization or bg correction)
##   minfi version: 1.38.0
##   Manifest version: 0.4.0
```

```
# save(rawMSet, file = "rawMSet.rda")

# M signal per probe, per sample
Meth <- getMeth(rawMSet)
Meth[1:5, 1:5]
```

```
##            GSM1051870_7766130158_R03C02 GSM1052024_5730053010_R01C02
## cg00050873                          514                          270
## cg00212031                          170                          400
## cg00213748                          312                          490
## cg00214611                          181                          349
## cg00455876                          295                          497
##            GSM1052035_5730053011_R04C02 GSM1051874_7766130166_R02C01
## cg00050873                        26002                         1073
## cg00212031                          342                          242
## cg00213748                         1997                          237
## cg00214611                          312                          349
## cg00455876                         5458                          681
##            GSM1051871_7766130158_R05C02
## cg00050873                          396
## cg00212031                          420
## cg00213748                          286
## cg00214611                          262
## cg00455876                          470
```

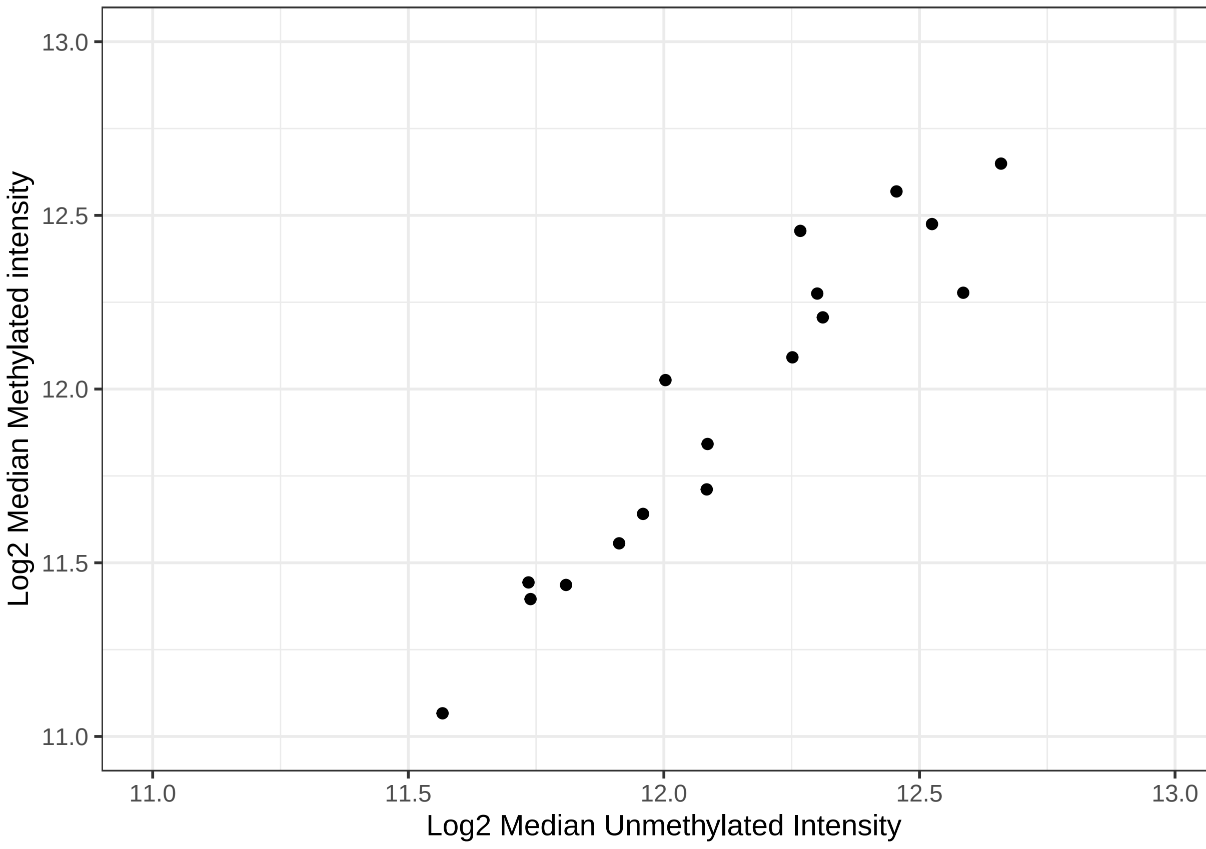
```
# U signal per probe, per sample
Unmeth <- getUnmeth(rawMSet)
Unmeth[1:5, 1:5]
```

```
##            GSM1051870_7766130158_R03C02 GSM1052024_5730053010_R01C02
## cg00050873                          432                          348
## cg00212031                          494                          463
## cg00213748                          299                          476
## cg00214611                          362                          459
## cg00455876                         1183                          922
##            GSM1052035_5730053011_R04C02 GSM1051874_7766130166_R02C01
## cg00050873                         5595                          571
## cg00212031                         8375                          272
## cg00213748                          688                          358
## cg00214611                        5644                             269
## cg00455876                        3139                            1248
##              GSM1051871_7766130158_R05C02
## cg00050873                         355
## cg00212031                         478
## cg00213748                         301
## cg00214611                         576
## cg00455876                        1312
```

**Visualize raw intensities**

```
# Overall intensity: M vs. U
pd$MQC <- log2(colMedians(Meth))
pd$UQC <- log2(colMedians(Unmeth))

pd$Slide <- factor(pd$Slide) #If we don't change datatypes then R will think we want slide and batch as
pd$Batch <- factor(pd$Batch)
```

```
ggplot(pd, aes(UQC, MQC)) +
  geom_point() +
  coord_cartesian(xlim = c(11, 13), ylim = c (11, 13)) +
  labs(x = "Log2 Median Unmethylated Intensity", y = "Log2 Median Methylated intensity") +
  theme_bw()
```



**Raw intensities**

We want to now visualize the intensity split by different technical variables that we often adjust for in analyses.

All illumina 450K and EPIC samples assayed have a well position (often called array), a slide, and a larger plate (often called batch) onto which the slide is placed.

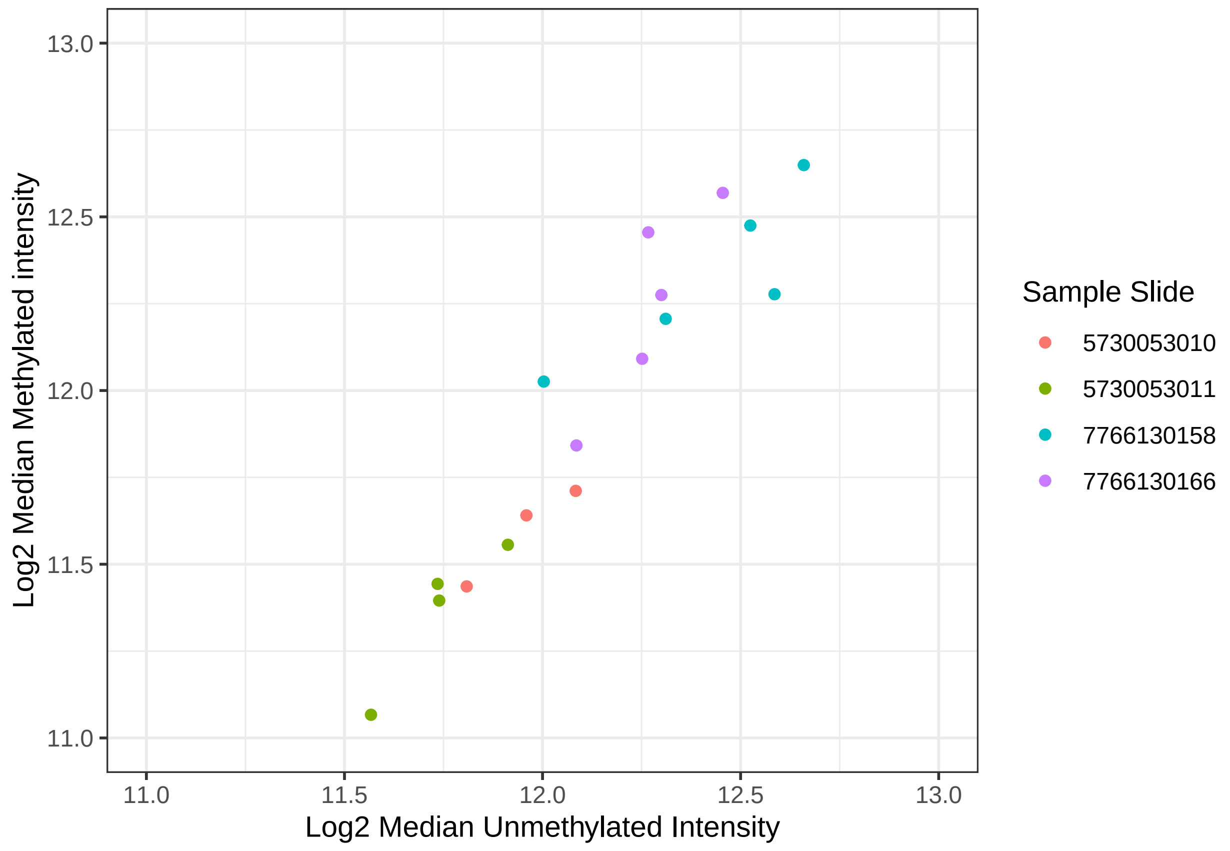
We are going to replicate that graph 3 times, by each of those 3 variables (well, slide, plate). Writing a function with a factor variable as input will help. Writing a function for repeated processes is usually good because it is safer and quicker than copying and pasting the same code each time.

```
intenseplot <- function(col = ""){
  ggplot(pd, aes_string(x = "UQC", y = "MQC", color = col)) +
    geom_point() +
    coord_cartesian(xlim = c(11, 13), ylim = c (11, 13)) +
    labs(x = "Log2 Median Unmethylated Intensity", y = "Log2 Median Methylated intensity",
         color = sprintf("Sample %s", col)) + #sprintf() %s means 'insert a character string here' and
    theme_bw()
}
```

Let's also include some different graphing palettes.

**Slide** This is the base ggplot2 palette

```
intenseplot(col = "Slide")
```

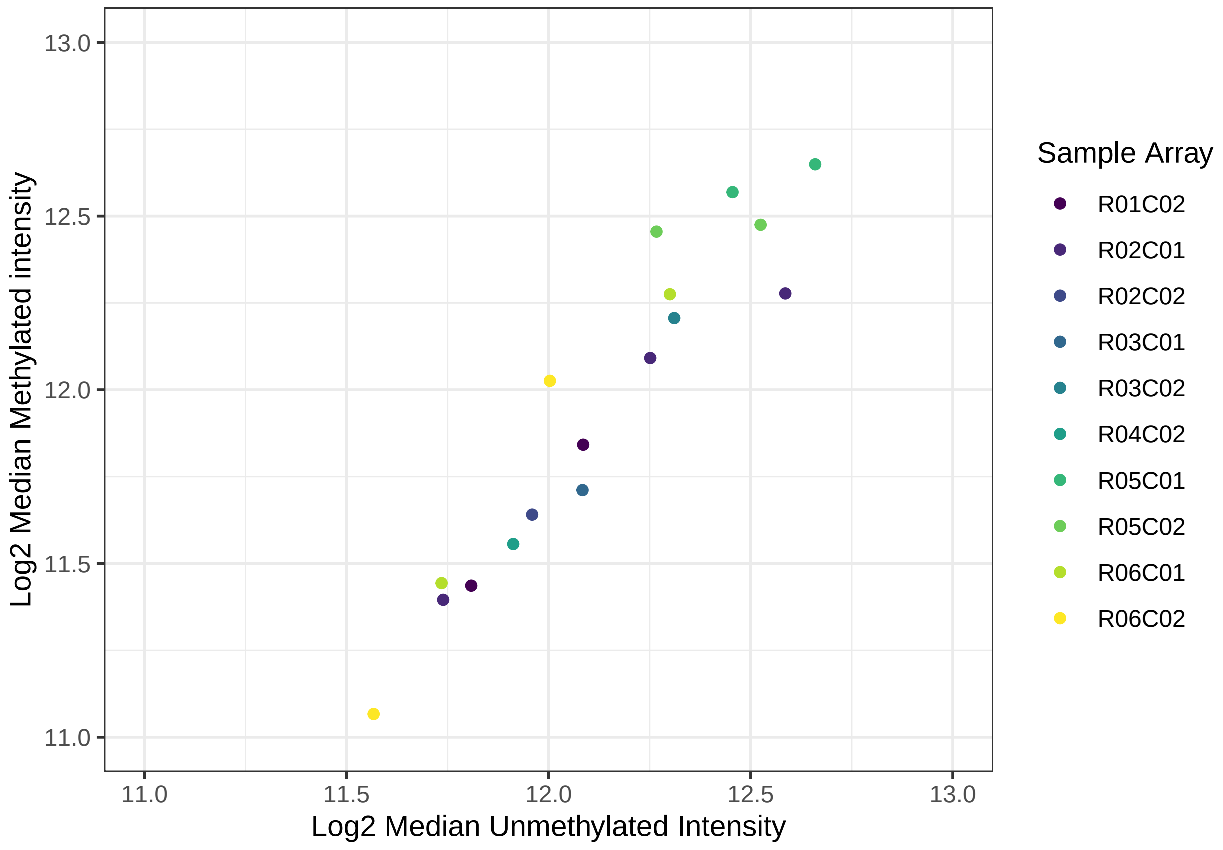


**Well/Array** We are using GEO data with the 450k chip. On the 450k the wells are arranged on a slide with 6 rows and 2 columns for a total of 12 different possible positions. On the more recent EPIC chip they reduced the slide to a single line of 8 positions.

Sample well may also be called Array as it is in our dataset.

This palette the ggplot2 implementation of the viridis package. Viridis is designed to be sensitive to colorblind people and holds up with many different types of colorblindness.

```r
intenseplot(col = "Array") + scale_color_viridis_d()
```

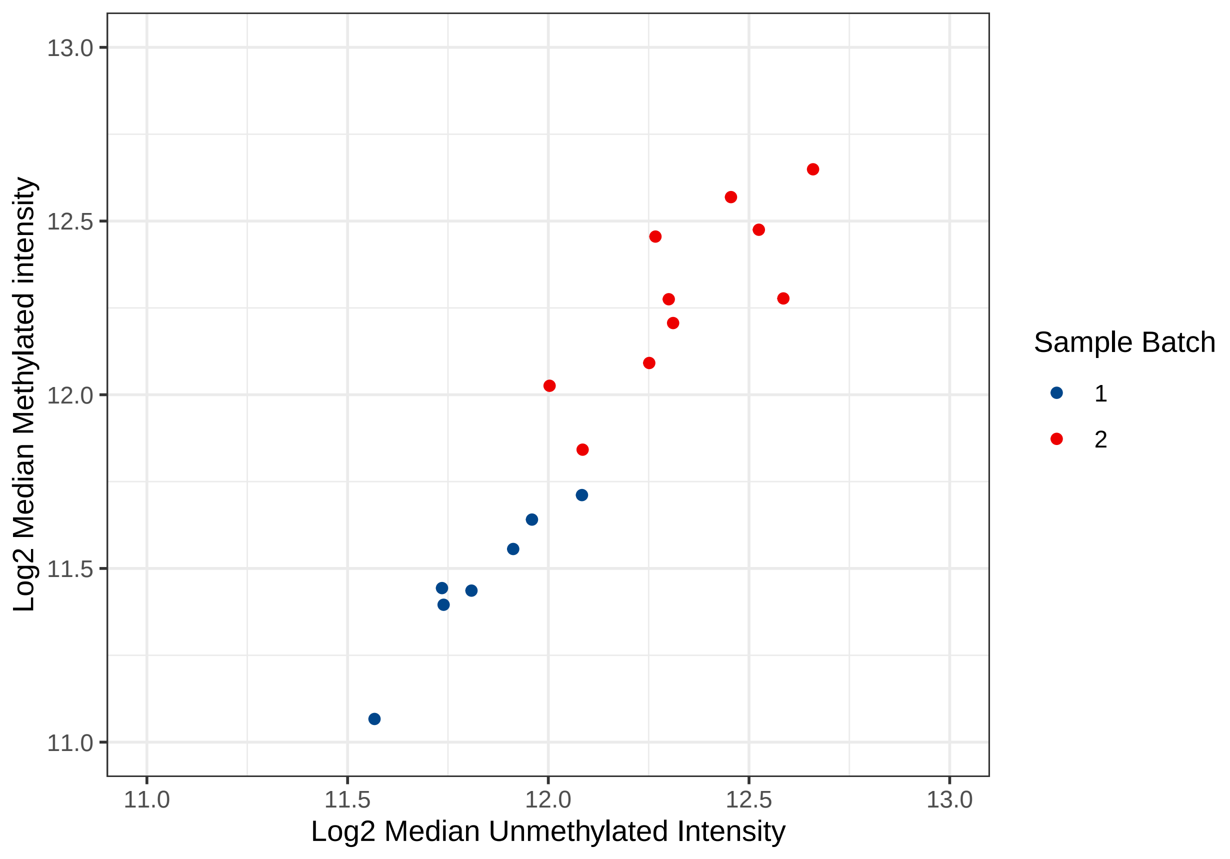


**Plate/Batch** We now look at Plate which is often called batch as well although batch and plate are not necessarily the exact same thing. In our data (450K chip) there are up to 8 slides in a plate. Within EPIC it switches to 12 slides per plate. So for 450K array a plate is up to 8 slides of 12 samples each for 96 total samples. In EPIc it is *12 slides* and *8 samples for slide* which arrives at the same total of up to 96 samples per plate.

Batch often refers to plate in DNAm but it can also mean a bit more abstract. For instance, if 3 plates are run in June and then 3 plates are run later in October then we may consider the 2 sets different batches.

We can see below that the 2 different batches show **large** separation in their median intensities. This underscores the need for smart sample plating before the samples are assayed. We can adjust for batch effects using covariates, ComBat software, or sva software. These methods will be explained in the next lab.

```r
intenseplot(col = "Batch") + ggsci::scale_color_lancet()
```



**Drop or flag low intensity samples**

Here we have no low intensity samples. Also note that the cutpoint of 11 for UQC and MQC is not set in stone as that value; it may depend on your data.

```
# Drop (or if really small sample: watch out for): Samples with UQC<11 & MQC<11
# Note the cutoff value (here, 11) would depend on your data and array (EPIC/450k)
sum(pd$UQC < 11)
```

```
## [1] 0
```

```
sum(pd$MQC < 11)
```

```
## [1] 0
```

```
rm(Meth, Unmeth) # Clean up our coding environment
```

## Ewastools Sample checks

### Create raw methyl dataset

```
library(ewastools)
```

```
##
## Attaching package: 'ewastools'

## The following object is masked from 'package:Hmisc':
##
```

```
##     mask
## The following object is masked from 'package:minfi':
##
##     detectionP
## The following object is masked from 'package:Biostrings':
##
##     mask
```

```r
meth <- read_idats(here("Data", "idats", pd$Basename)) %>% detectionP()
```

```
## [1] 622399
##   |                                                                              |
```

## Illumina control metrics

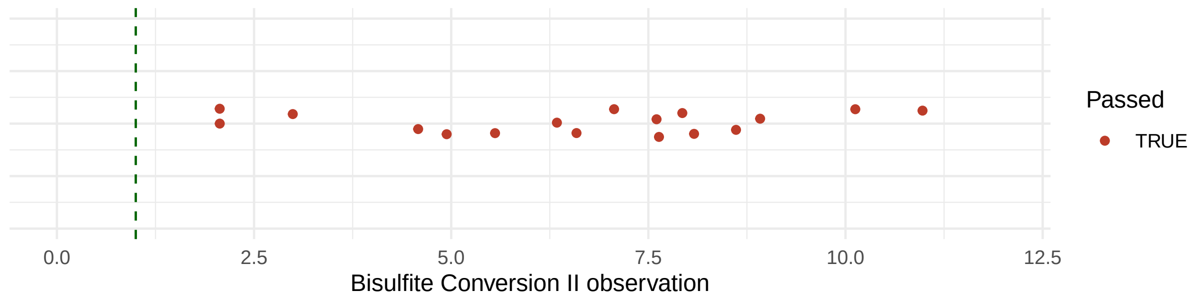
Illumina includes 17 control metrics to check for sample quality. We can use ewastools to find any samples that fail these metrics. These samples should be flagged. They can be cut in later analyses if a more stringent sample filtering is desired. In our case all 17 samples pass every control metric.

We can visualize a control metric. Each control has a threshold that a samples observation must exceed in order to pass

```r
ctrls <- control_metrics(meth)
pd$ctrlfail <- ewastools::sample_failure(ctrls)
table(pd$ctrlfail)
```

```
##
## FALSE
##    17
```

```r
bisulfite <- data.frame(measures = unlist(ctrls["Bisulfite Conversion II"]))
thresh <- base::attr(ctrls$`Bisulfite Conversion II`, "threshold")

ggplot(bisulfite, aes(measures, 1, color = measures > thresh)) +
  geom_jitter(height = 0.03) +
  coord_cartesian(ylim = c(0.8, 1.2), xlim = c(0, 12)) +
  geom_vline(xintercept = thresh, linetype = "dashed", color = "darkgreen") +
  theme_minimal() +
  theme(axis.text.y = element_blank()) +
  scale_color_nejm() +
  labs(y = "", x = "Bisulfite Conversion II observation", color = "Passed")
```

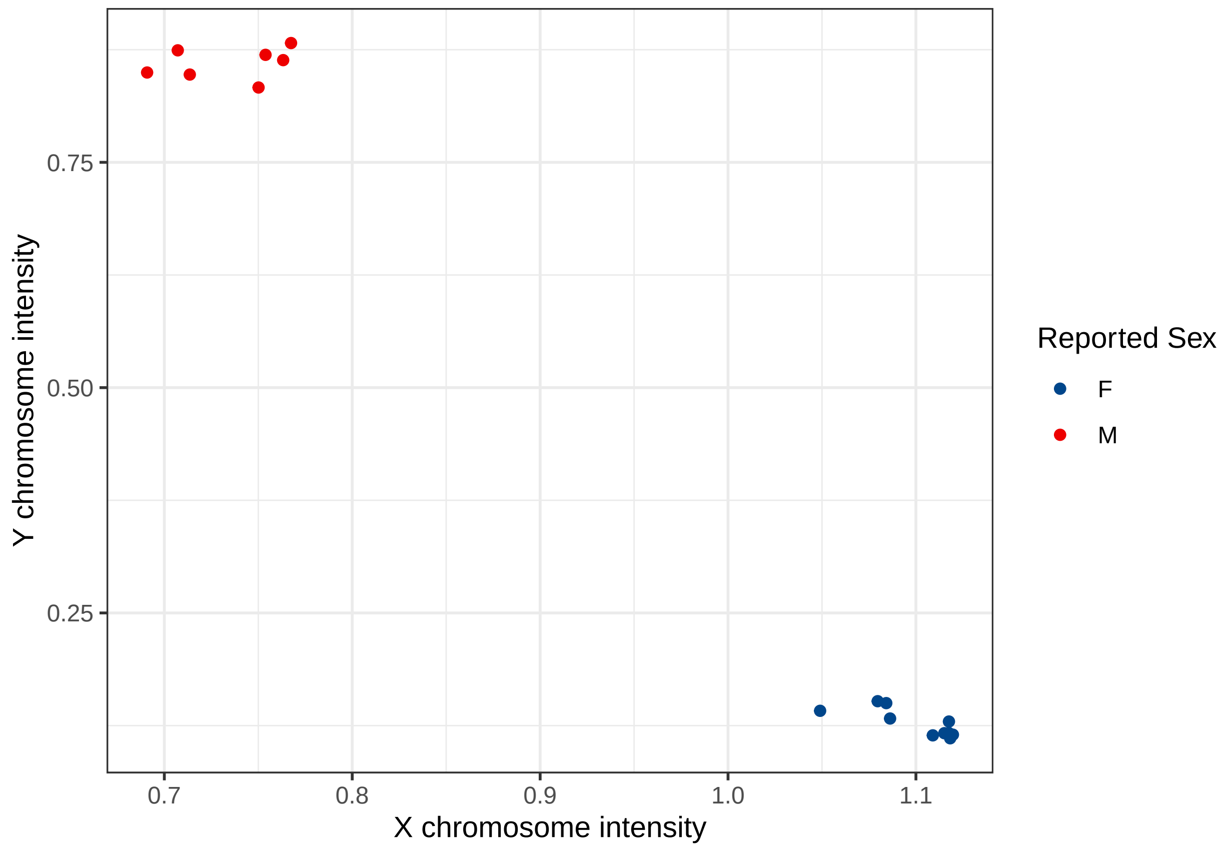


```r
rm(ctrls)
```

## Check sex

We can check conflicts between the reported and predicted sex of each sample. Predicted sex is derived from looking at the signal intensities of the sex chromosomes. This can help us to identify poorly assayed samples or misplates.

```
sex <- meth %>% correct_dye_bias() %>% check_sex()
pd[c("X","Y")] <- bind_rows(sex)

ggplot(pd, aes(X,Y, color = sex)) +
  geom_point() +
  labs(x = "X chromosome intensity", y = "Y chromosome intensity", color = "Reported Sex") +
  theme_bw() +
  scale_color_lancet()
```



There are no discrepancies between reported sex and sex chromosome intensity.

## DNAm relatedness

Several dozen of the probes are placed at to a SNP. This makes the methylation expression mostly dependent on the underlying genotype. With this information we can make a rough inference as to genetic relatedness between different samples. This can be helpful to discover misplates or duplicate samples.

```
snps <- meth$manifest[probe_type == "rs", index]

#This runs the raw methylation set through a dye bias correction, detectionP masking before converting
#All of these steps will be discussed later in the lab
geno <- meth %>% correct_dye_bias() %>% mask(0.01) %>% dont_normalize %>% .[snps,] %>% call_genotypes()
```

```
check_snp_agreement(geno, pd$GEOID, pd$GEOID)
```

```
## NULL
```

The function returns NULL. This means that there are no conflicts in genetic relatedness. All of our 17 look genetically unrelated.

## Derive beta matrix

The equation for a beta matrix is: $Beta = Methyl/(Methyl + Unmethyl)$ where $Methyl$ refers to total methylated intensity and $Unmethyl$ refers to unmethylated intensity. We see that this $Beta$ value is really just the proportion of total intensity that is methylated intensity

```
#Dye bias correction seeks to fix the difference in overall intensity between red and greed flourescenc
#dont_normalize means that there will be no inter-sample normalization as it's been found to mute genui
beta <- meth %>% correct_dye_bias %>% dont_normalize
```

### To mask or not to mask, that is a question.

Figure 1: He'sn't not sure either

Masking here refers to deleting certain observations by rewriting them as NA. If you have masked values you will not have equal observations for all samples, which may be problematic. This can be fixed by data imputation.

```
#Mask all observations with DetectionP > 0.01
#beta2 <- meth %>% correct_dye_bias %>% mask(0.01) %>% dont_normalize
```

## Sample and probe filtering with detectionP and nBeads

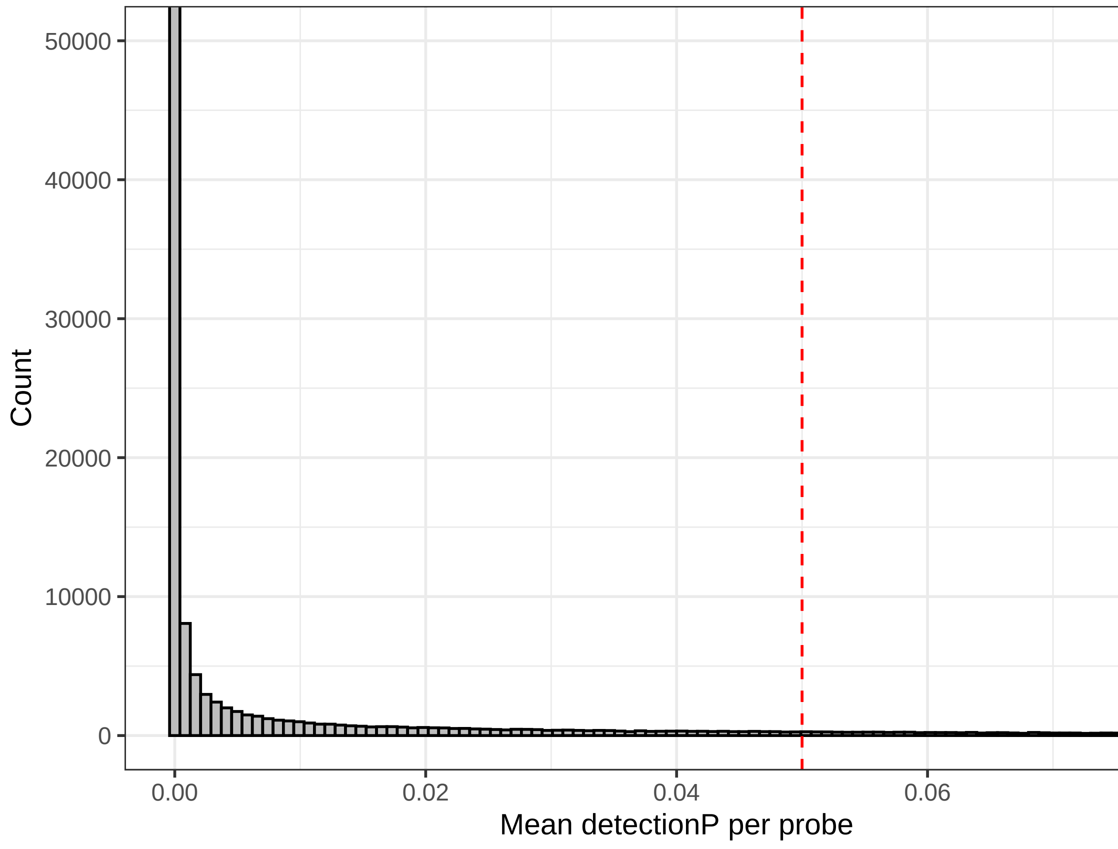
### DetectionP

DetectionP gives us a metric for assessing signal/noise ratios. More specifically, it measures the amount of background flourescence. Both samples and probes with higher detection p levels (indicative of unacceptable levels of noise) are cut or flagged.

We get a detectionP value for every observation of all probes in all samples. These are dichotomized as pass/fail with a chosen threshold. We will choose a bit more of a conservative threshold of 0.01.

```r
detp <- meth$detP
save(detp, file = here("Data", "detP.rda"))
```

```r

detprobe <- data.frame(pmean = rowMeans(detp, na.rm = T))
probecut <- 0.05

ggplot(detprobe, aes(x = pmean)) +
  geom_histogram(color = "black", fill = "grey", bins = 1000) +
  coord_cartesian(xlim = c(0, 0.08), ylim = c(0, 5e4)) +
  geom_vline(xintercept = probecut, color = "red", linetype = "dashed") +
  theme_bw() +
  labs(x = "Mean detectionP per probe", y = "Count")
```



#### Plot probe detectionP

```r
table(detprobe$pmean > 0.05)
```

```
##
```

```
##  FALSE   TRUE
## 460143  25434
```

```
table(detprobe$pmean > 0.05) %>% '/'(nrow(detprobe)) %>% round(digits = 3)
```

```
##
## FALSE  TRUE
## 0.948 0.052
```

```
rm(detprobe)
```

We can see that only about 5% of the probes are over the threshold

## nBeads

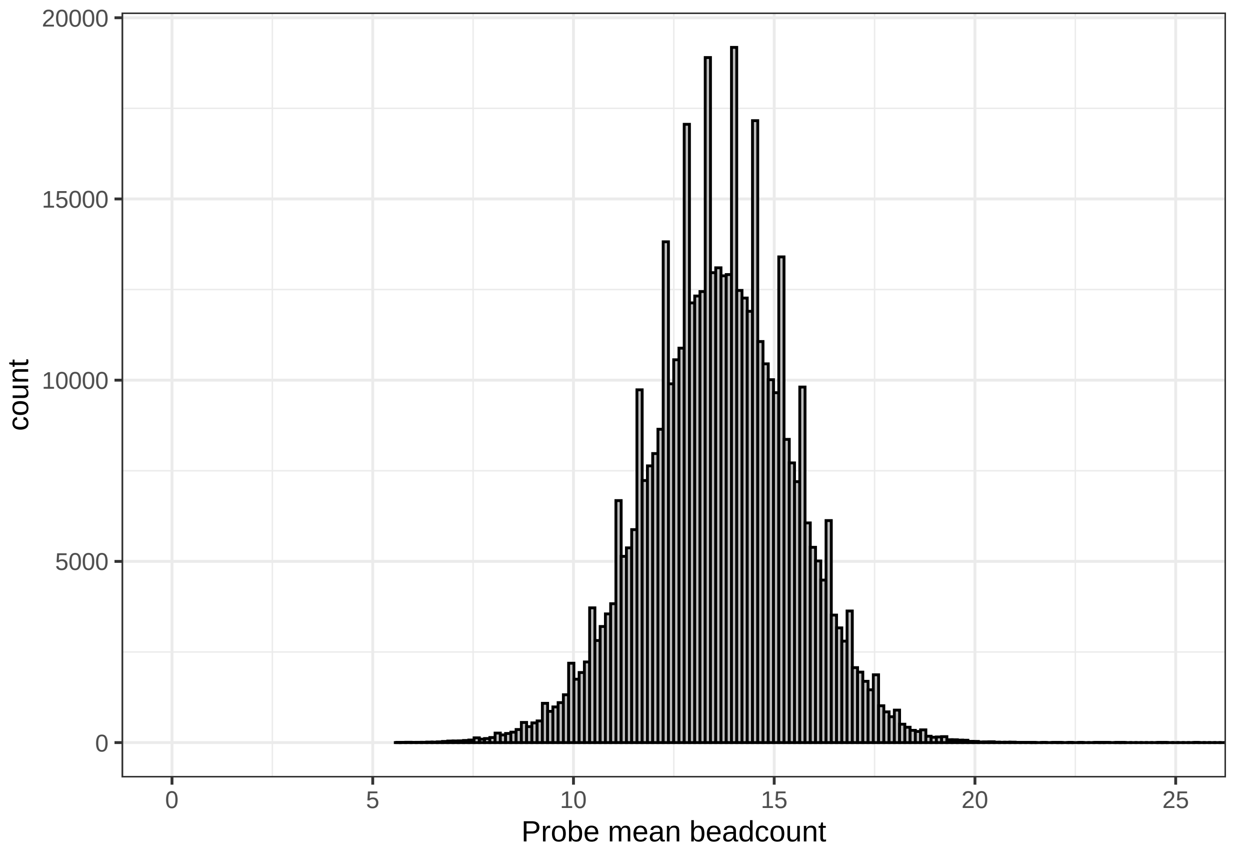
nBeads refers to the number of hybridizing beads that were responsible for the beta methylation observation. Low bead numbers (n<4) indicate observations that are likely of lower quality.

The bead numbers follow a normal distribution.

```
mani <- ewastools:::manifest_450K
nbeads <- wateRmelon::beadcount(RGset)
```

```
## No methods found in package 'RSQLite' for request: 'dbListFields' when loading 'lumi'
```

```
nbeads <- nbeads[match(mani[mani$probe_type != "rs", probe_id],rownames(nbeads)),]

beadmean <- data.frame(bmean = rowMeans(nbeads, na.rm = T))

ggplot(beadmean, aes(x = bmean)) +
  geom_histogram(bins = 200, color = "black", fill = "grey") +
  coord_cartesian(xlim = c(0, 25)) +
  labs(x = "Probe mean beadcount") +
  theme_bw()
```



```
(table(nbeads < 5) / length(unlist(nbeads))) %>% round(digits = 3)
```

```
##
## FALSE  TRUE
## 0.991 0.008
```

```
detp <- detp[!grepl("rs", meth$manifest$probe_id),] #Removing the 65 SNP probes
beta <- beta[!grepl("rs", meth$manifest$probe_id),]

rm(RGset,beadmean)
```

Only a small proportion of the observations fail this check

## Filter probes

All observations with a detectionP value of <0.05 and/or a beadcount of under 5 will be labeled as unreliable. All probes with over 5% unreliable values will be cut.

```
beadmin <- 5
detpmax <- 0.01
pthresh <- 0.05

reliable <- (detp > detpmax) & (nbeads < beadmin)
reliable[is.na(reliable)] <- F

dim(beta)
```

```
## [1] 485512     17
```

```
beta <- beta[rowMeans(reliable) < pthresh,] ; reliable <- reliable[rowMeans(reliable) < pthresh,]

dim(beta)
```

```
## [1] 482946     17
```

### Filter samples

After bad probes are filtered out we can continue to filter out samples. We set a less stringent cutoff value of 0.1.

```
sthresh <- 0.1

beta <- beta[,colMeans(reliable, na.rm = T) < sthresh] ; reliable <- reliable[,colMeans(reliable, na.rm

dim(beta)
```

```
## [1] 482946     17
```

```
rm(reliable)
```

## Cross-reactive probes

Cross-reactive probes are shown to 'co-hybridize' onto multiple different sites on the epigenome. This means that we can't know for sure whether the methylation measures are measuring the site we actually want it to.

```
load(here("Data", "cross.probes.info.rda"))

beta <- beta[!rownames(beta) %in% as.character(cross.probes.info$TargetID),]

rm(cross.probes.info)
```

## Gap probes

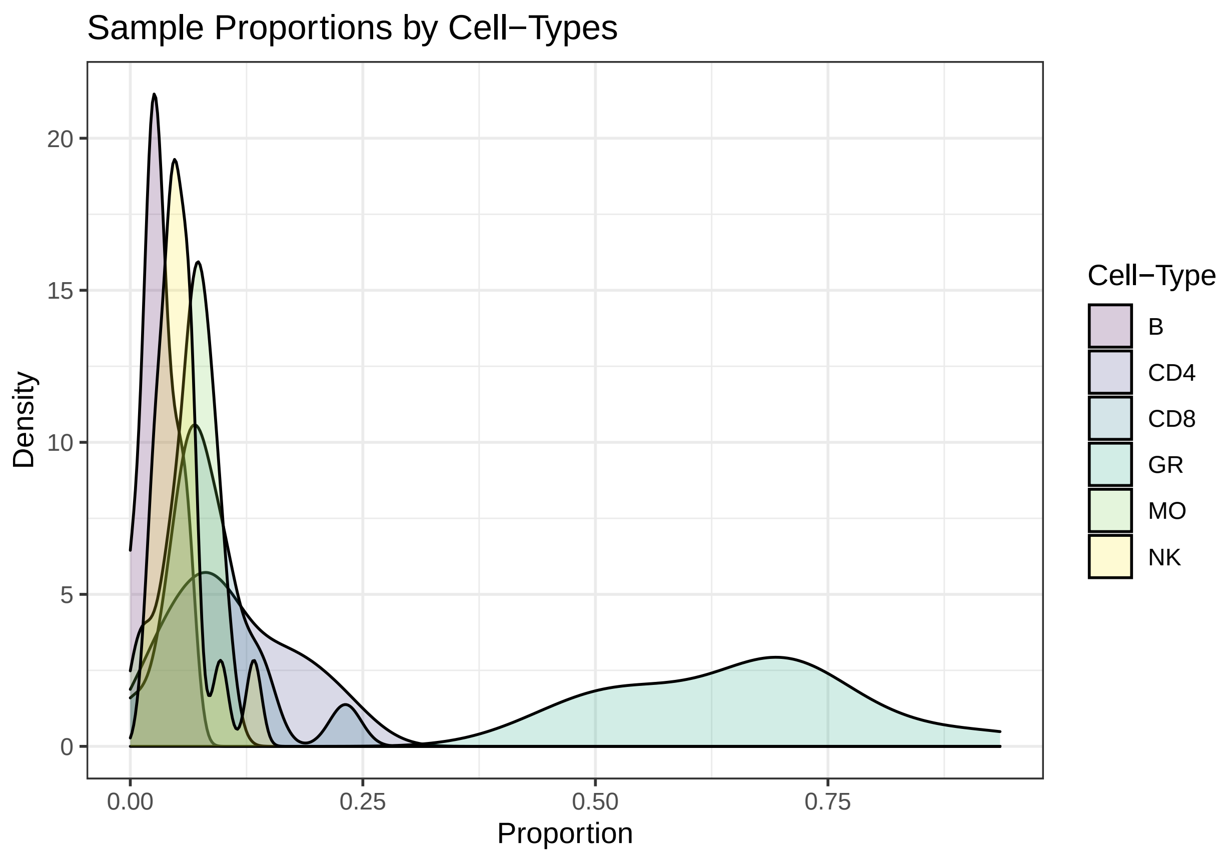
Gap probes are probes for which the methylation beta clusters into discrete groups– these typically have their methylation driven by underlying SNPs. So with these probes the underlying genotype explains the beta methylation value. These probes do not necessarily need to be cut but it's good to be aware of them. With a very low sample

```
gaps <- gaphunter(beta, outCutoff = 0.1, threshold = 0.15) # We set a very high outlier cutoff because

str(gaps)

#Get a probe in the list as an example and plot the beta distribution.

gap1 <- beta[rownames(gaps$sampleresults)[6],]
gap1 <- data.frame(gap1)

ggplot(gap1, aes(x = rownames(gaps$sampleresults)[6], y = gap1)) +
  geom_jitter(width = 0.01) +
  labs(title = sprintf("Methylation Beta Values for probe %s", rownames(gaps$sampleresults)[2]), x = "
  coord_cartesian(ylim = c(0, 1)) +
  theme_bw()

rm(gap1)
```

## Cell type proportions

Cell type proportions are a strong confounder of DNA methylation. Different cell types have varying levels of methylation in different probes so we can use these probes to disentangle cell types proportions from unrelated DNA methylation differences.

```
cells <- estimateLC(beta, ref = "salas", constrained = T)

ggplot(pivot_longer(cells, cols = 1:6), aes(x = value, y = ..density.., fill = name)) +
  geom_density(alpha = 0.2) +
  scale_fill_viridis_d() +
  theme_bw() +
  labs(x = "Proportion", y = "Density", title = "Sample Proportions by Cell-Types", fill = "Cell-Type")
```



```
pd <- data.frame(pd, cells)
```

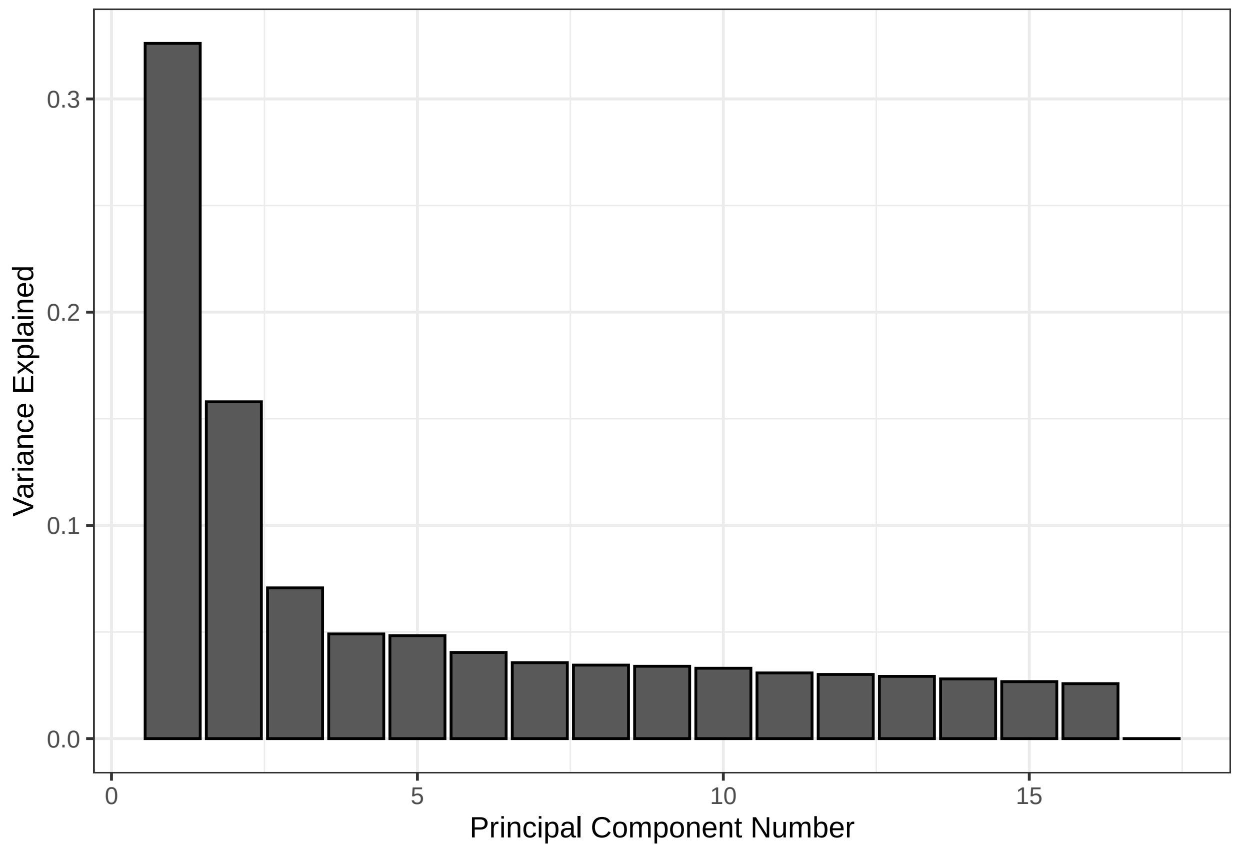
## Drop samples with problematic cell proportions

```
# Pick cutoffs for biological ranges for cell type estimates.
pd <- pd[pd$MO < 0.2, ]
pd$grq <- cut2(pd$GR, g = 4)
beta <- beta[,colnames(beta) %in% pd$Basename]
dim(beta)
```

```
## [1] 453802     17
```

## Principal components on samples from the beta matrix

```r
#This is mean imputation which is pretty imprecise. W
nainds <- which(is.na(beta), arr.ind = T)
beta[nainds] <- rowMeans(beta[nainds[1,],], na.rm = T)

#Remove sex chromosome probes before pca
mani <- ewastools:::manifest_450K
sexprobes <- mani[mani$chr %in% c("X", "Y"),][, "probe_id"]
betapcs <- prcomp(t(beta[!rownames(beta) %in% sexprobes,]), center = T, scale. = F)
out.var <- betapcs$sdev^2 / sum(betapcs$sdev^2)
pcvar <- data.frame(pcs = seq(1, length(out.var)), var = out.var)

ggplot(pcvar, aes(x = pcs, y = var)) +
  geom_bar(stat = "identity", color = "black") +
  theme_bw() +
  labs(x = "Principal Component Number", y = "Variance Explained")
```



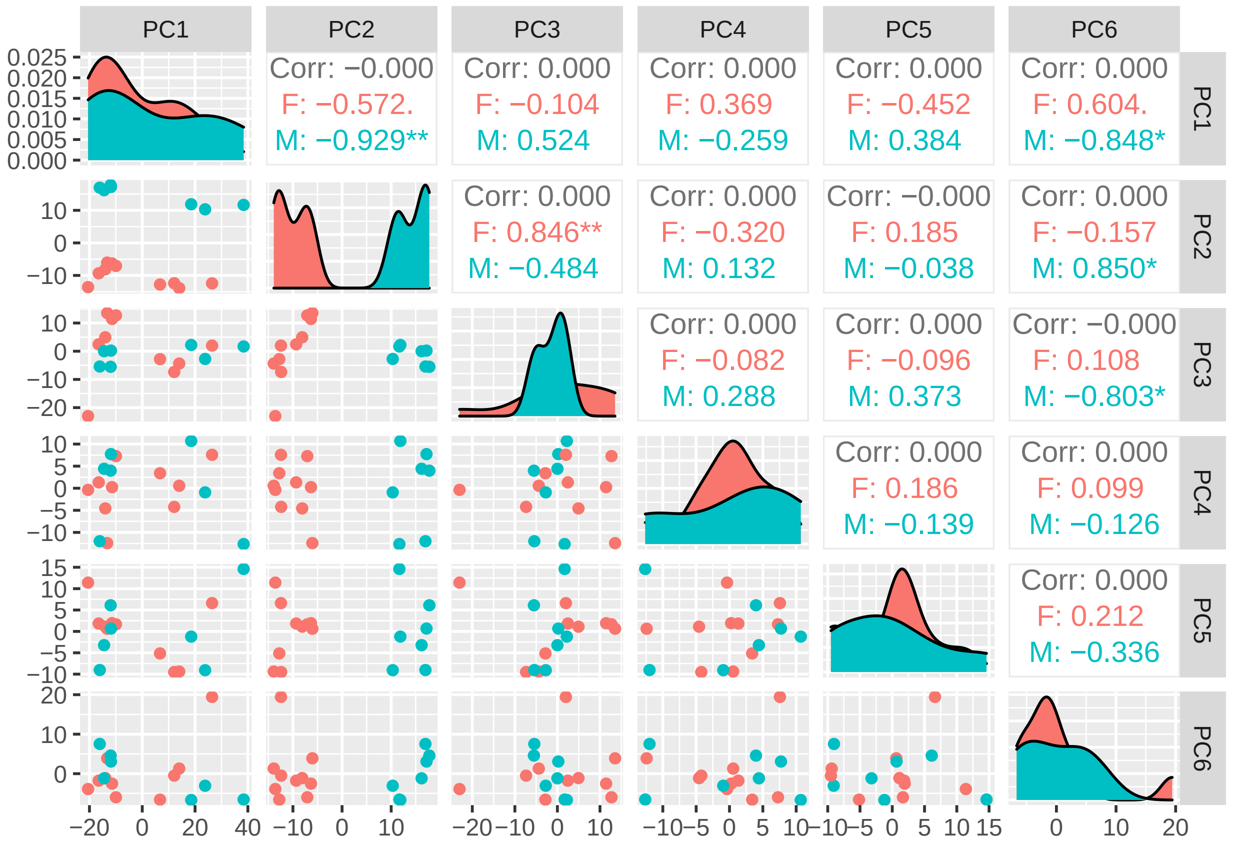
```r
library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```r
#Here we can define another function to use to look at principle component plots by different attribute
pcplot <- function(pheno){
  ggpairs(data.frame(betapcs$x)[, 1:6], aes(color = factor(unlist(pd[pheno]))))
}
```

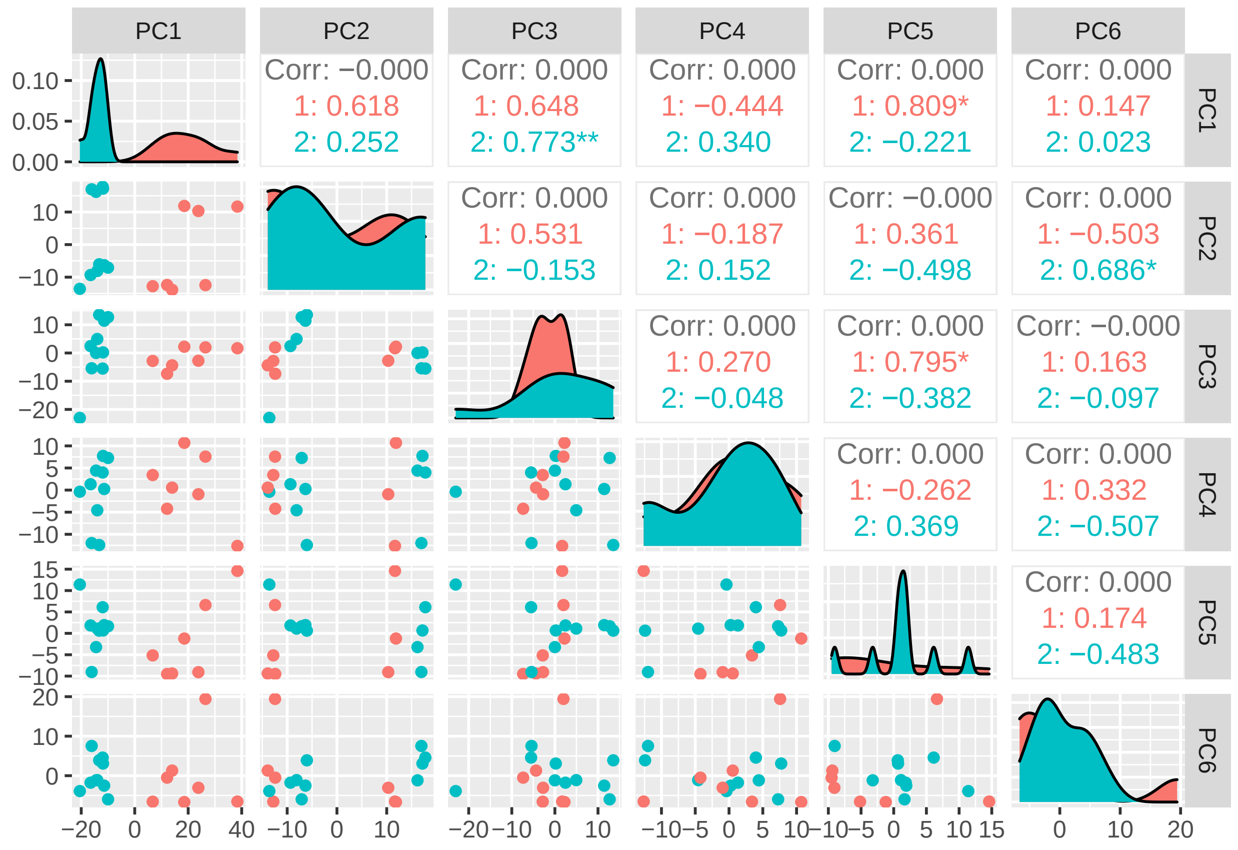
### By sex

```
pcplot("sex")
```



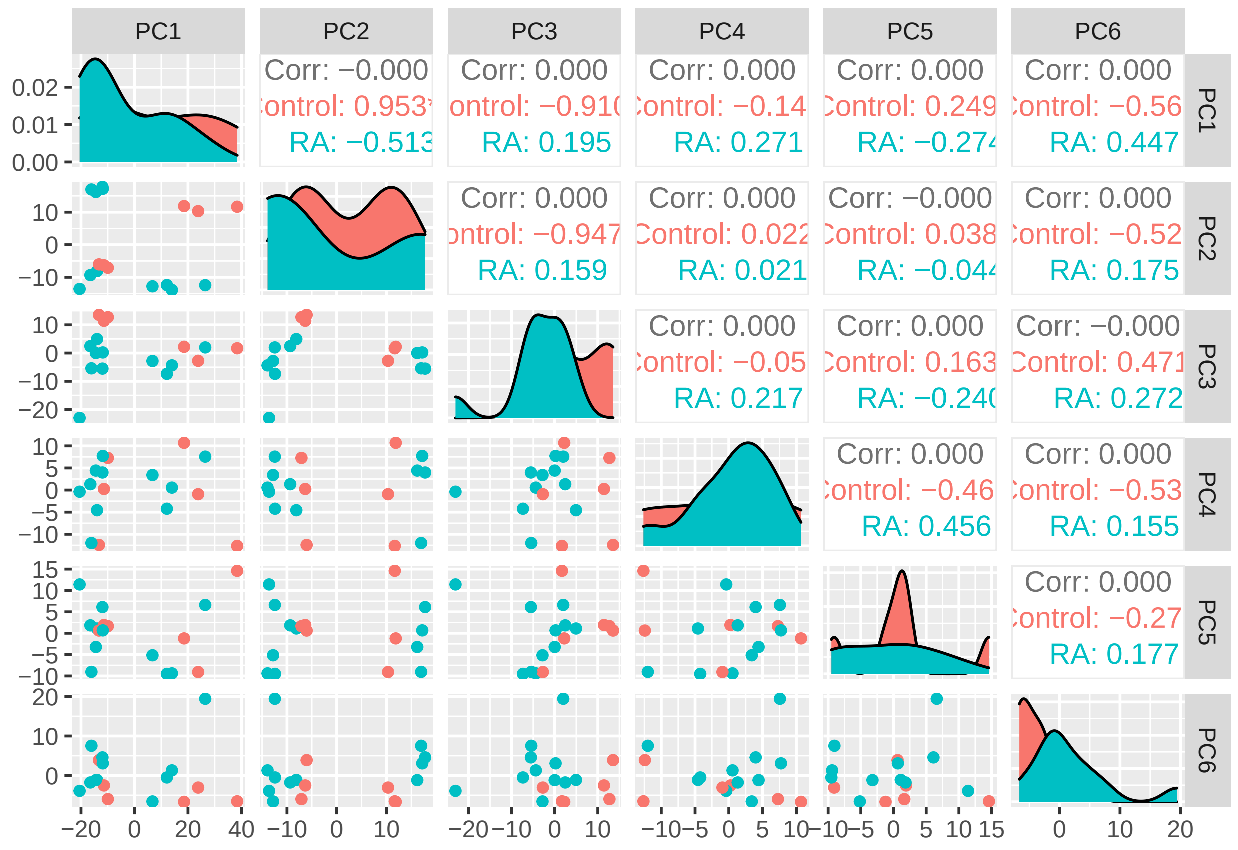
### Batch

```
pcplot("Batch")
```



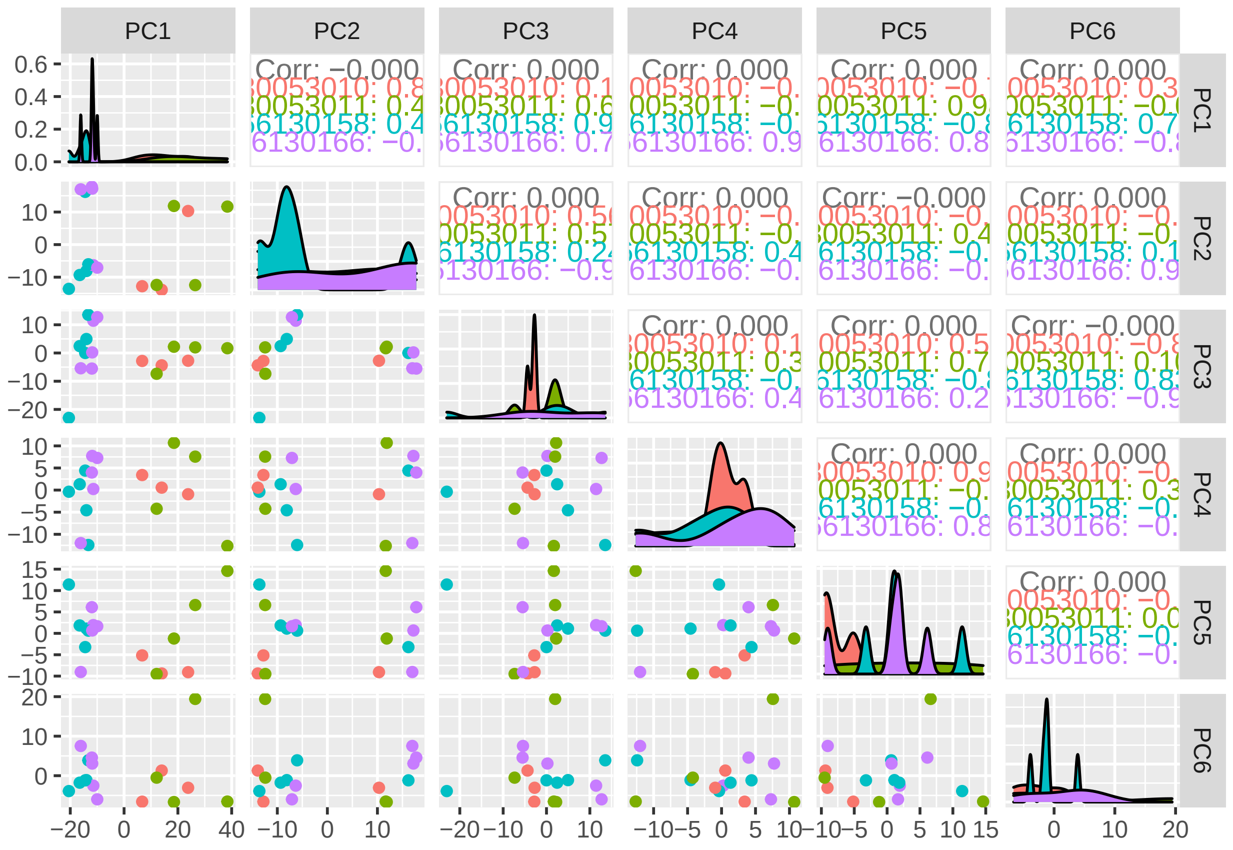
### Case status

```r
pcplot("casestatus")
```



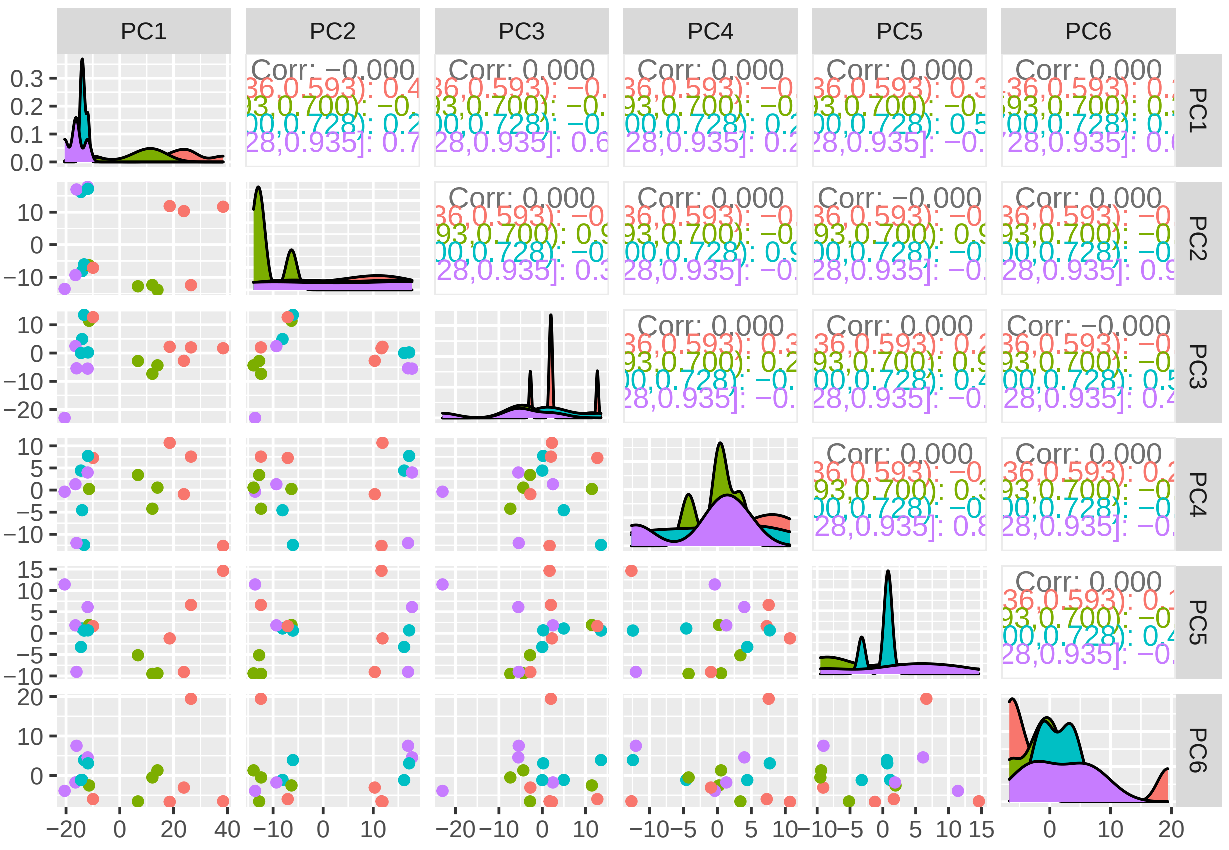
## By Slide

```
pcplot("Slide")
```



## GR quartile

```r
pcplot("grq")
```



## Drop samples with missing data at key covariates

```
# Drop samples with missing data at key covariates
covars <- c("sex", "casestatus", "smoking", "Batch")

pd <- pd[complete.cases(pd[, covars]), ] # Example of how to remove.
```

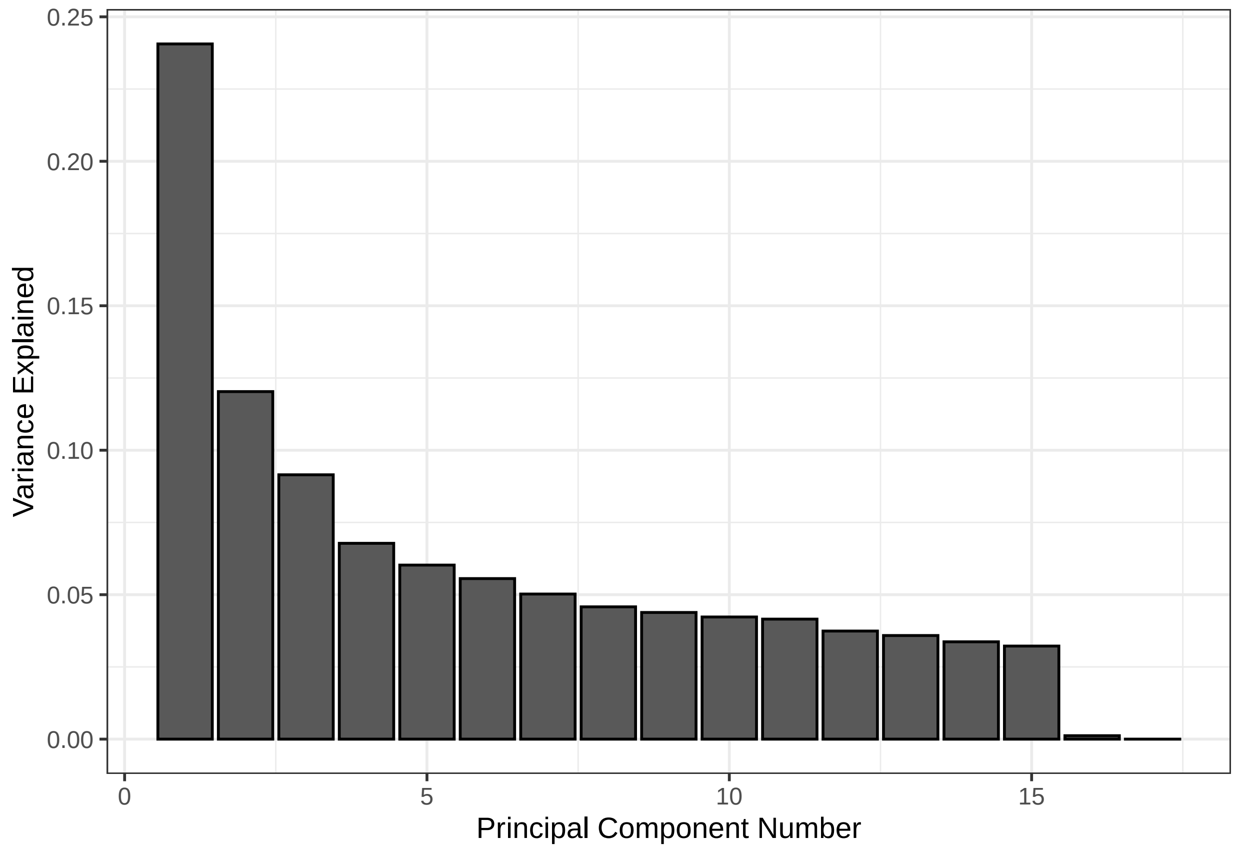
## Use Combat to adjust for batch effects

```
mod <- model.matrix(~ pd$sex + pd$casestatus + pd$smoking)

### These 2 following lines have already been run as they take too much memory for rstudio cloud
# combat.beta <- ComBat(dat = beta, batch = pd$Batch, mod = mod)
# save(combat.beta, file = here("Data", "Premade_Intermediate_Files", "combat-beta.rda"))

load(here("Data", "Premade_Intermediate_Files", "combat-beta.rda"))

combatpcs <- prcomp(t(combat.beta[!rownames(combat.beta) %in% sexprobes,]), center = T, scale. = F)

#add combat pcs to pd file and save
pd <- cbind(pd, combatpcs$x[, 1:5])
save(pd, file = here("Data","pdqc.rda"))
```

```r
out.var <- combatpcs$sdev^2 / sum(combatpcs$sdev^2)
out.var[1:10]
```

```
##  [1] 0.24062209 0.12028506 0.09149081 0.06777526 0.06024429 0.05556304
##  [7] 0.05022073 0.04579136 0.04383466 0.04227327
```

```r
pcvar <- data.frame(pcs = seq(1, length(out.var)), var = out.var)

ggplot(pcvar, aes(x = pcs, y = var)) +
  geom_bar(stat = "identity", color = "black") +
  theme_bw() +
  labs(x = "Principal Component Number", y = "Variance Explained")
```

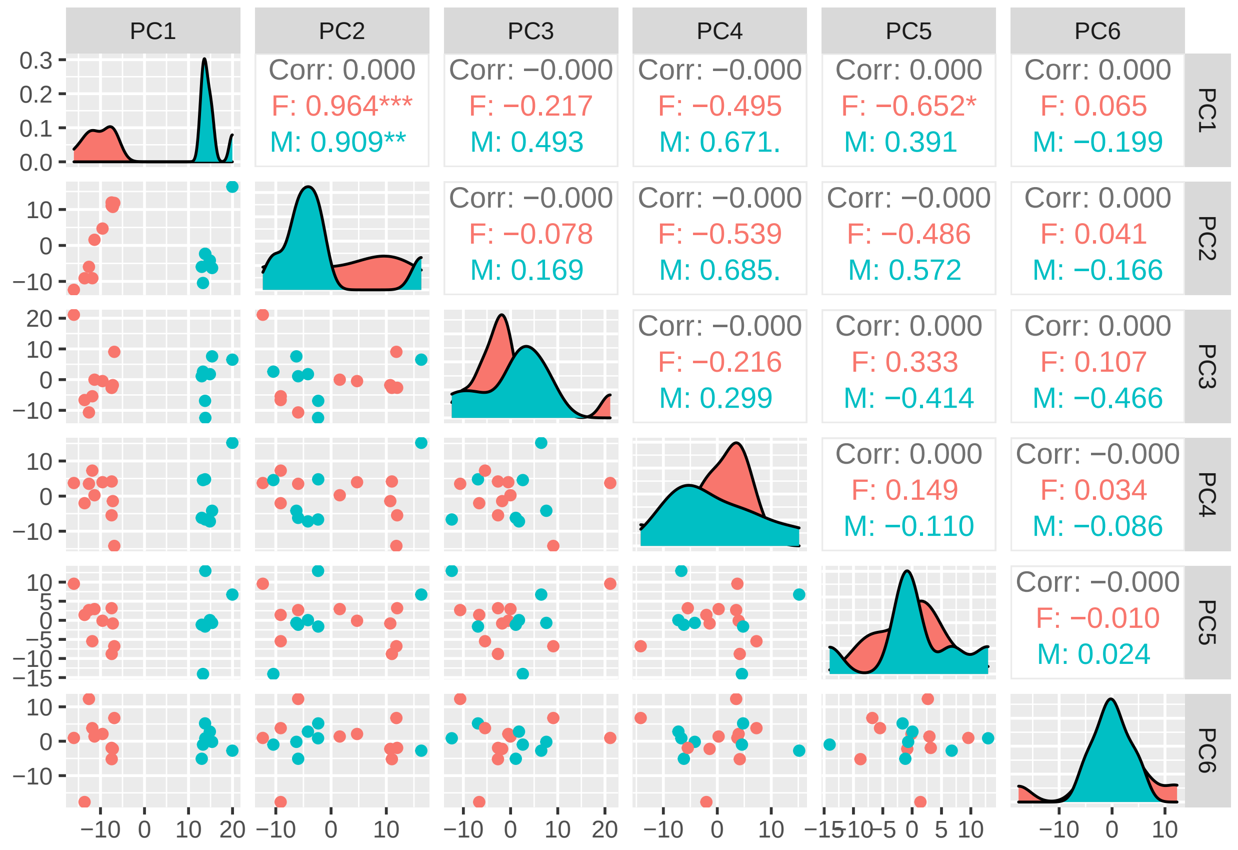


## Plot principal components of combat data

```r
library(GGally)
#Here we can define another function to use to look at principle component plots by different attribute
pcplotc <- function(pheno){
  ggpairs(data.frame(combatpcs$x)[, 1:6], aes(color = factor(unlist(pd[pheno]))))
}
```

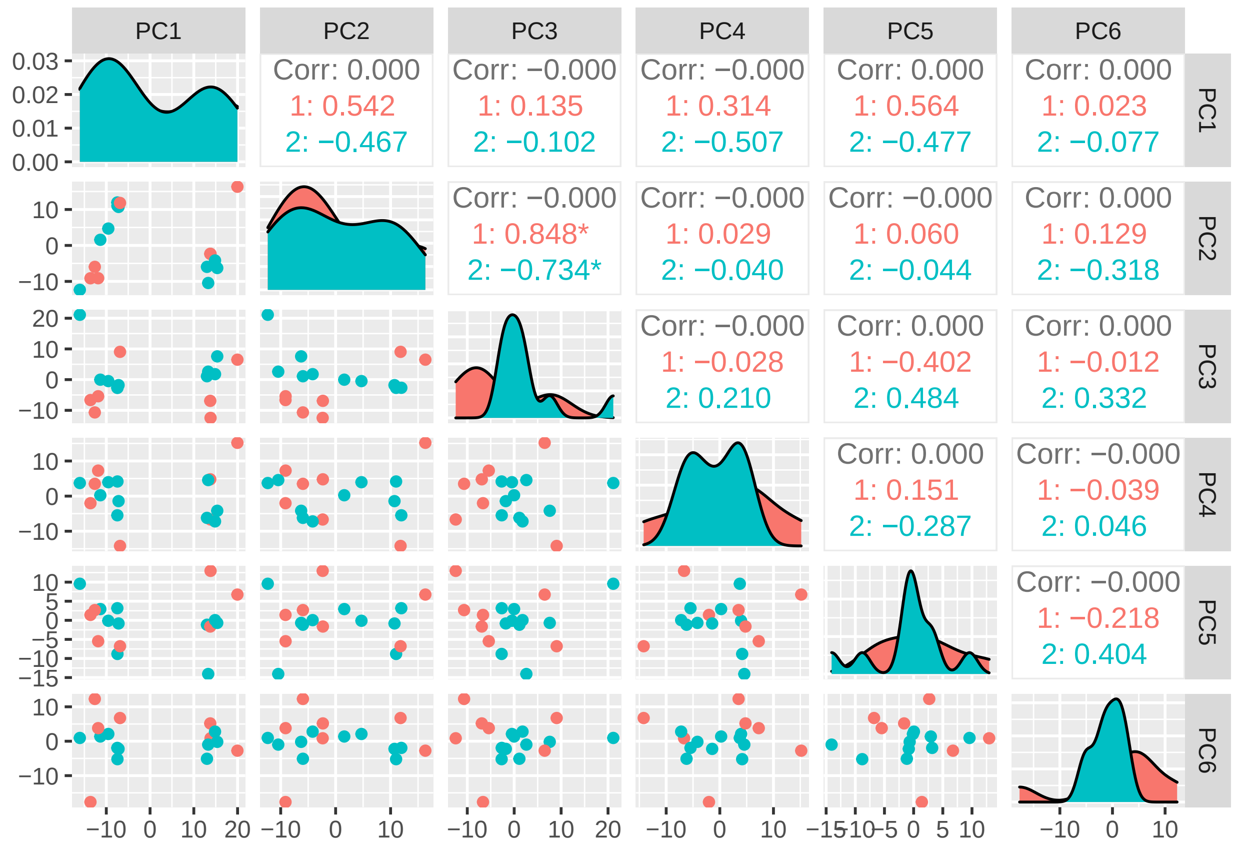
### By sex

```r
pcplotc("sex")
```



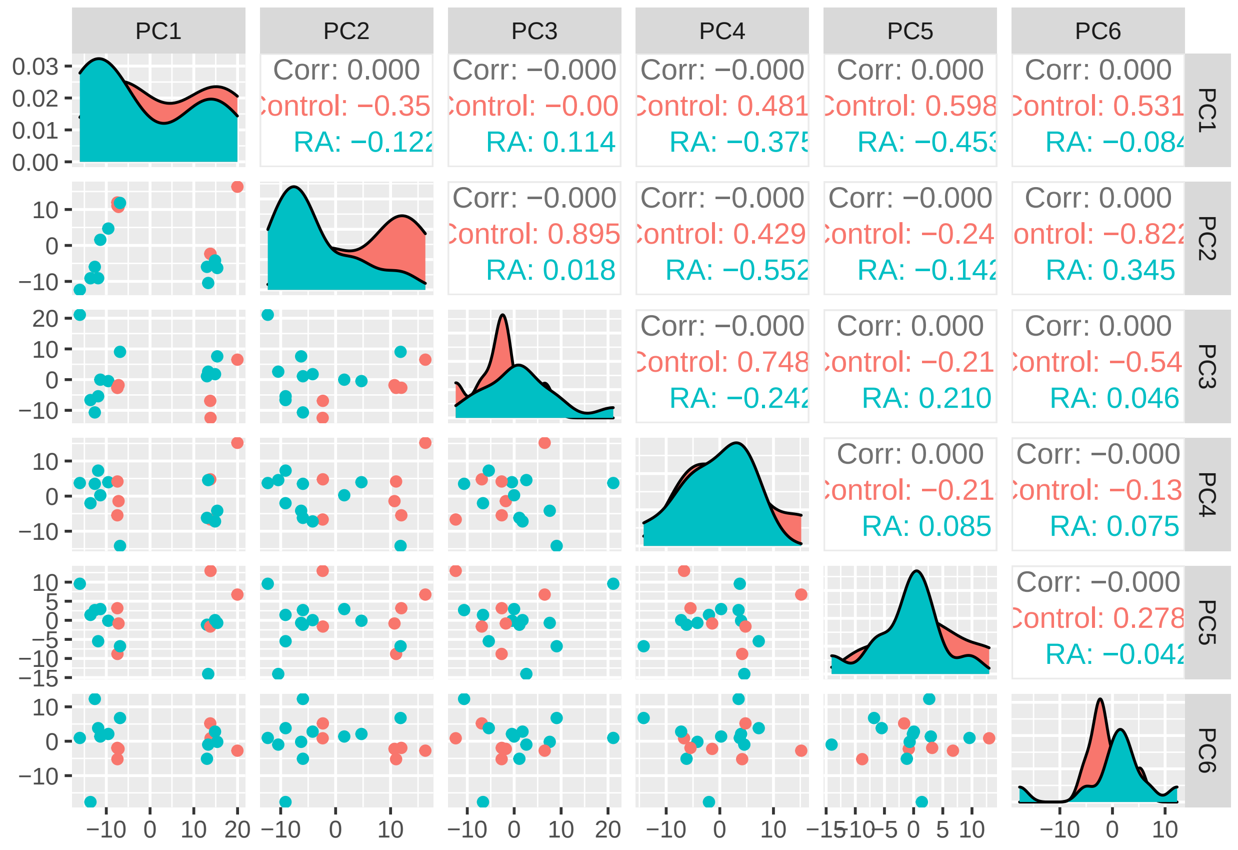
### Batch

```
pcplotc("Batch")
```



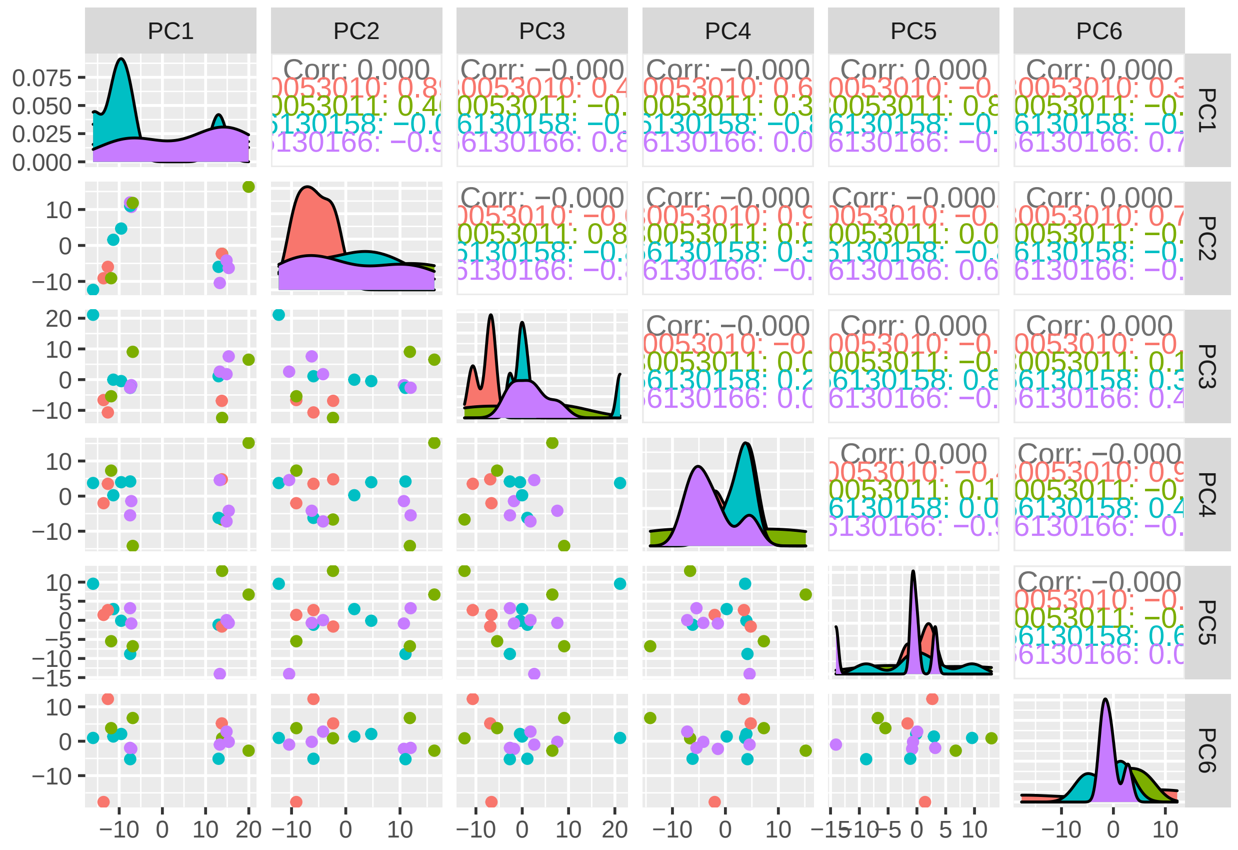
### Case status

```
pcplotc("casestatus")
```



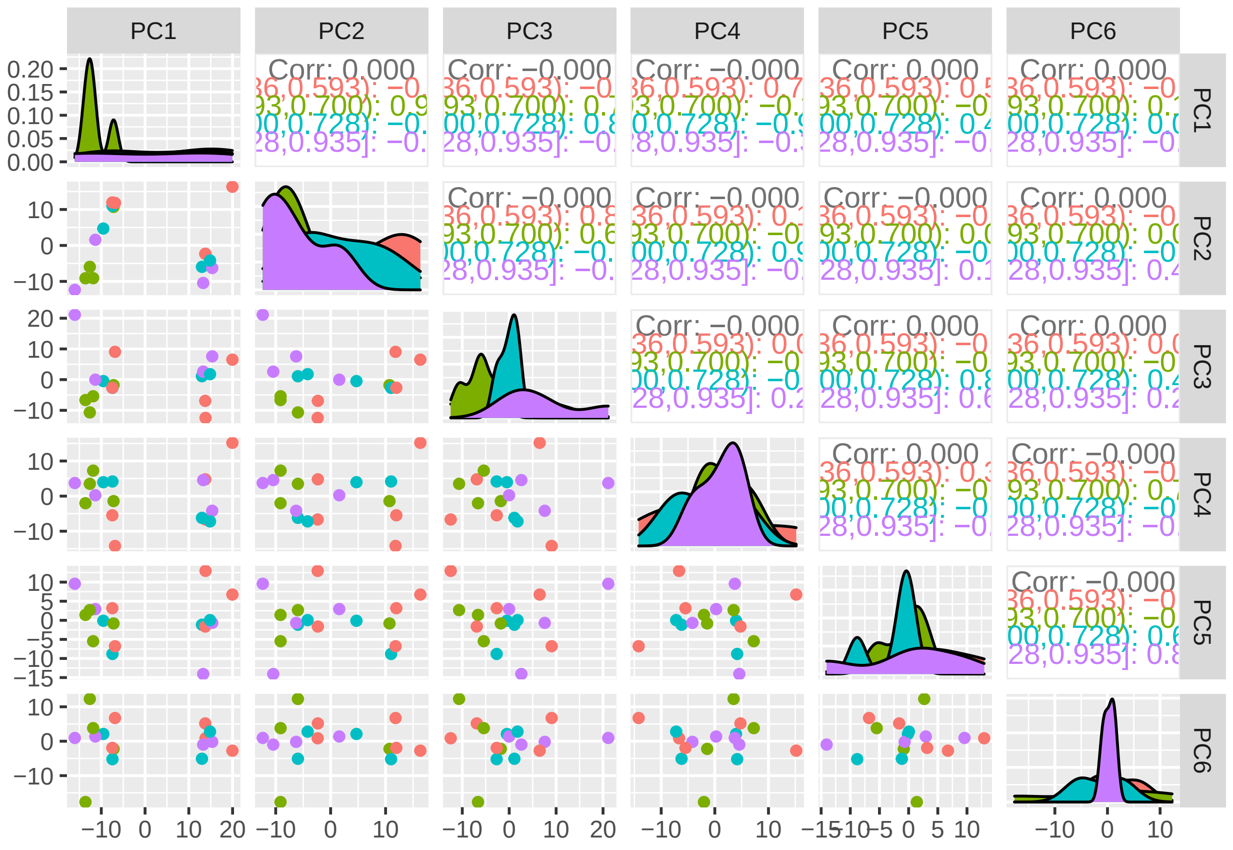
### By Slide

```
pcplotc("Slide")
```



## GR quartile

```
pcplotc("grq")
```


PC1
PC2
PC3
PC4
PC5
PC6
Corr: 0.000
Corr: −0.000