Epigenomics for Social Scientists

02 quality control of datasets

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

Last compiled on July 15, 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.

```
library(minfi)
library(MASS)
library(abind)
library(sva)
library(Hmisc)
library(ggplot2)
library(ggsci)
library(tidyverse)
library(data.table)
```

Setting file paths for data

```
# Setting file paths for data

# The rest of this script assumes that your data are in a folder called "project" on the Cloud.
# As you work on your own computer, you will need to specify the folder locations.

# Folder location of the data files
data_dir <- "E:/GESS/2060001/Data/"# switch back to "/cloud/project/Data"
data_dir
```

```
## [1] "E:/GESS/2060001/Data/"
```

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.

```
load(file.path(data_dir, "RGset.rda"))
pd <- RGset@colData@listData ; setDT(pd)
```

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.

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

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

```
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: NULL
## colData names(11): GEOID celltype ... Batch filenames
## Annotation
##   array: IlluminaHumanMethylation450k
##   annotation: ilmn12.hg19
## Preprocessing
##   Method: Raw (no normalization or bg correction)
##   minfi version: 1.34.0
##   Manifest version: 0.4.0
```

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

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

```
##            [,1] [,2]  [,3] [,4] [,5]
## cg00050873  514  270 26002 1073  396
## cg00212031  170  400   342  242  420
## cg00213748  312  490  1997  237  286
## cg00214611  181  349   312  349  262
## cg00455876  295  497  5458  681  470
```

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

```
##            [,1] [,2] [,3] [,4] [,5]
## cg00050873  432  348 5595  571  355
## cg00212031  494  463 8375  272  478
## cg00213748  299  476  688  358  301
## cg00214611  362  459 5644  269  576
## cg00455876 1183  922 3139 1248 1312
```

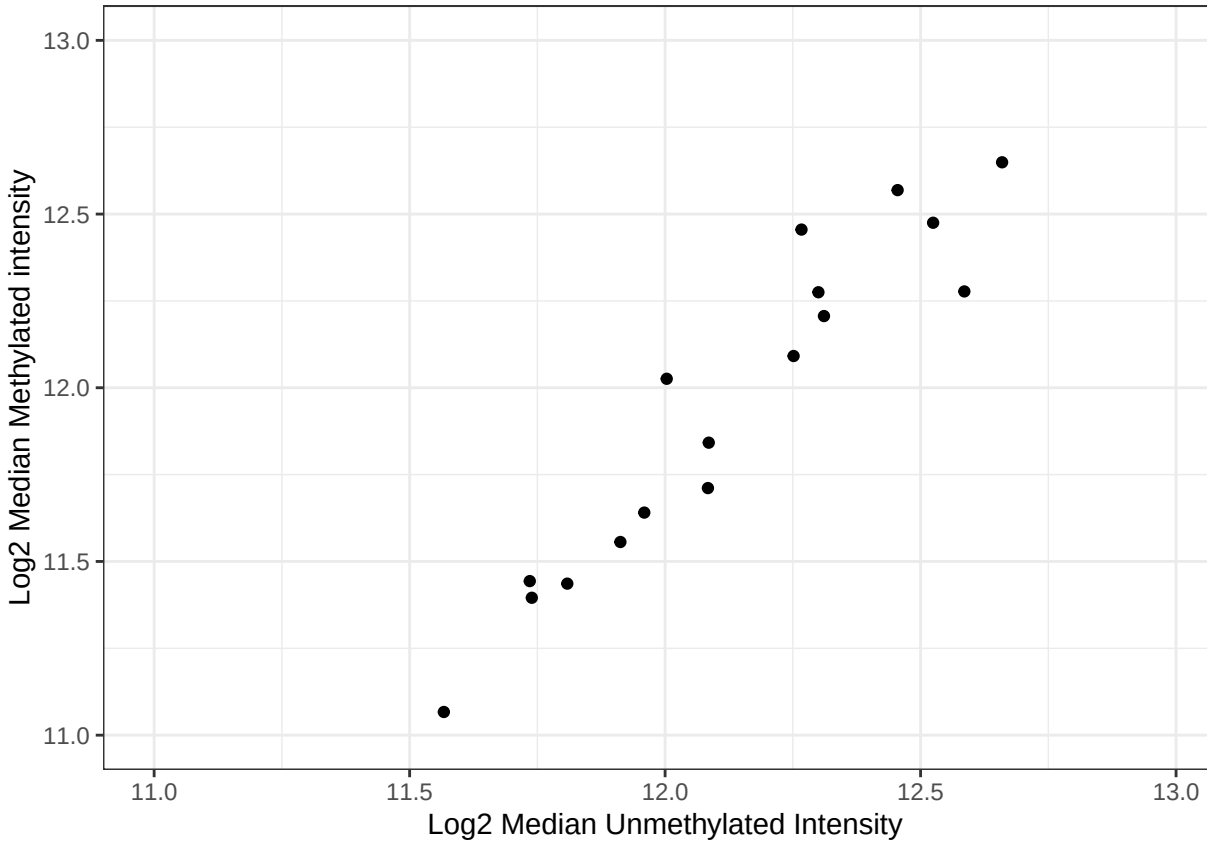
Visualize raw intensities

```
# Overall intensity: M vs. U
pd[, MQC := log2(colMedians(Meth))] # := lets us assign a new column within pd; in r base this would be
```

```
pd[, UQC := log2(colMedians(Unmeth))]

pd[, Slide := factor(Slide)] #If we don't change datatypes then R will think we want slide and batch as
pd[, Batch := factor(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.

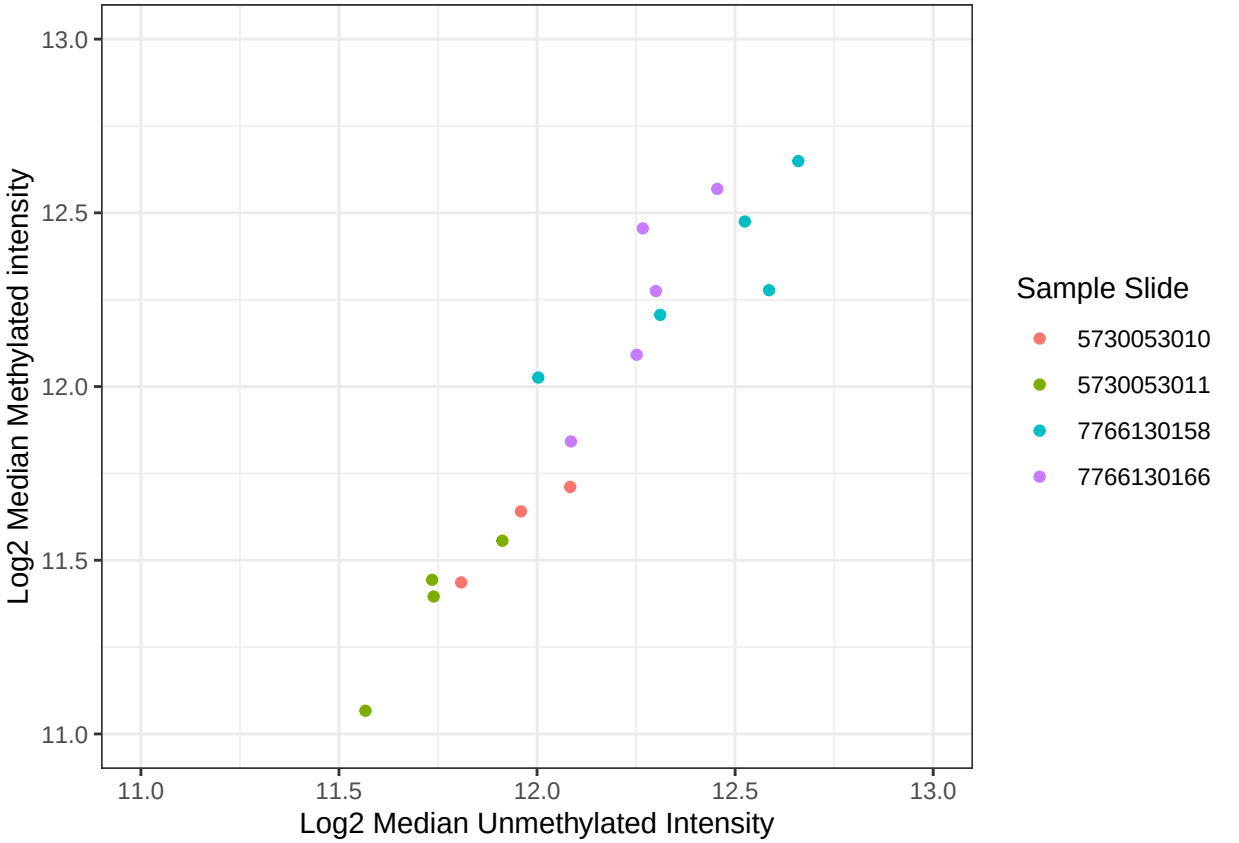
```
intenseplot <- function(col = ""){
  ggplot(pd, aes_string(x = "UQC", y = "MQC", color = col)) +
    geom_point() +
    coord_cartesian(xlim = c(11, 13), ylim = c (11, 13)) +
    labs(x = "Log2 Median Unmethylated Intensity", y = "Log2 Median Methylated intensity",
         color = sprintf("Sample %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")
```

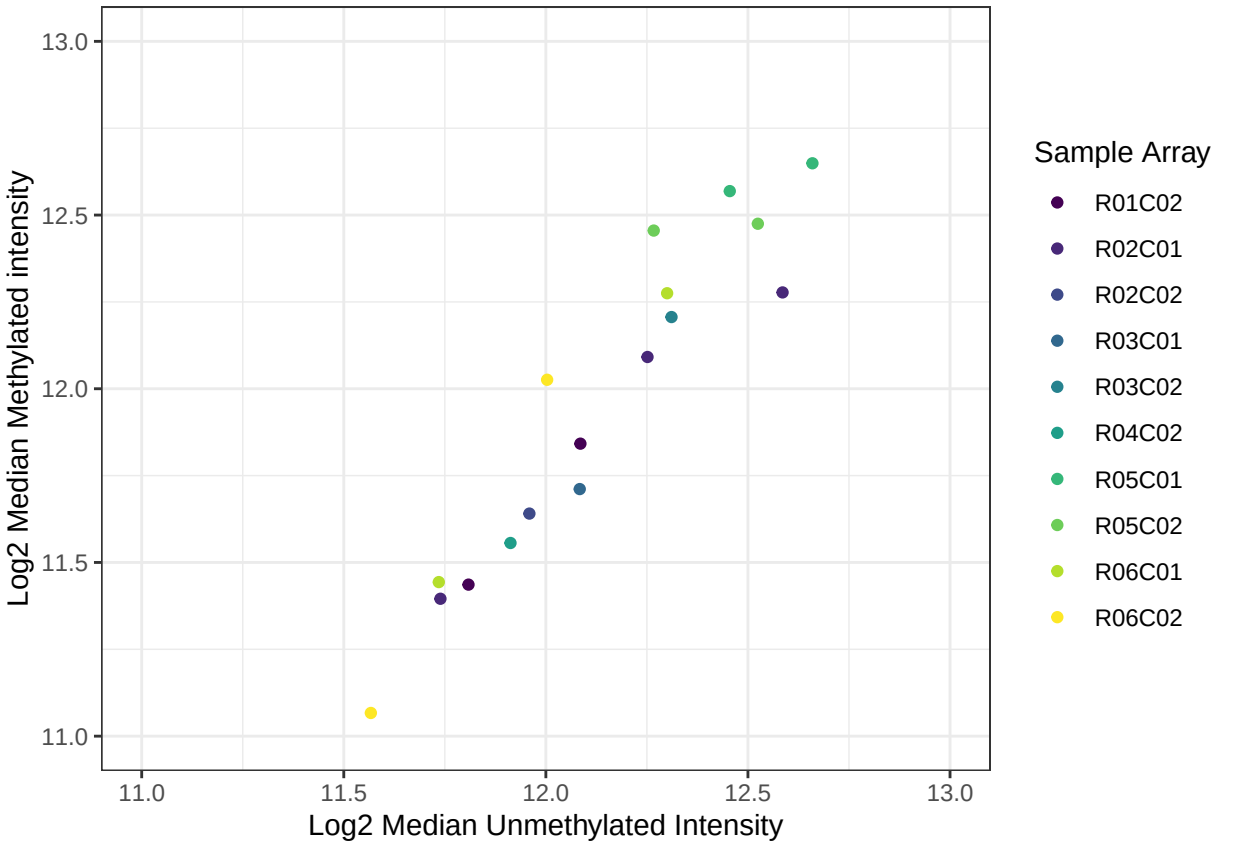


Well/Array We are using GEO data with the 450k chip. On the 450k the wells are arranged on a slide with 6 rows and 2 columns for a total of 12 different possible positions. On the more recent EPIC chip they reduced the slide to a single line of 8 positions.

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.

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

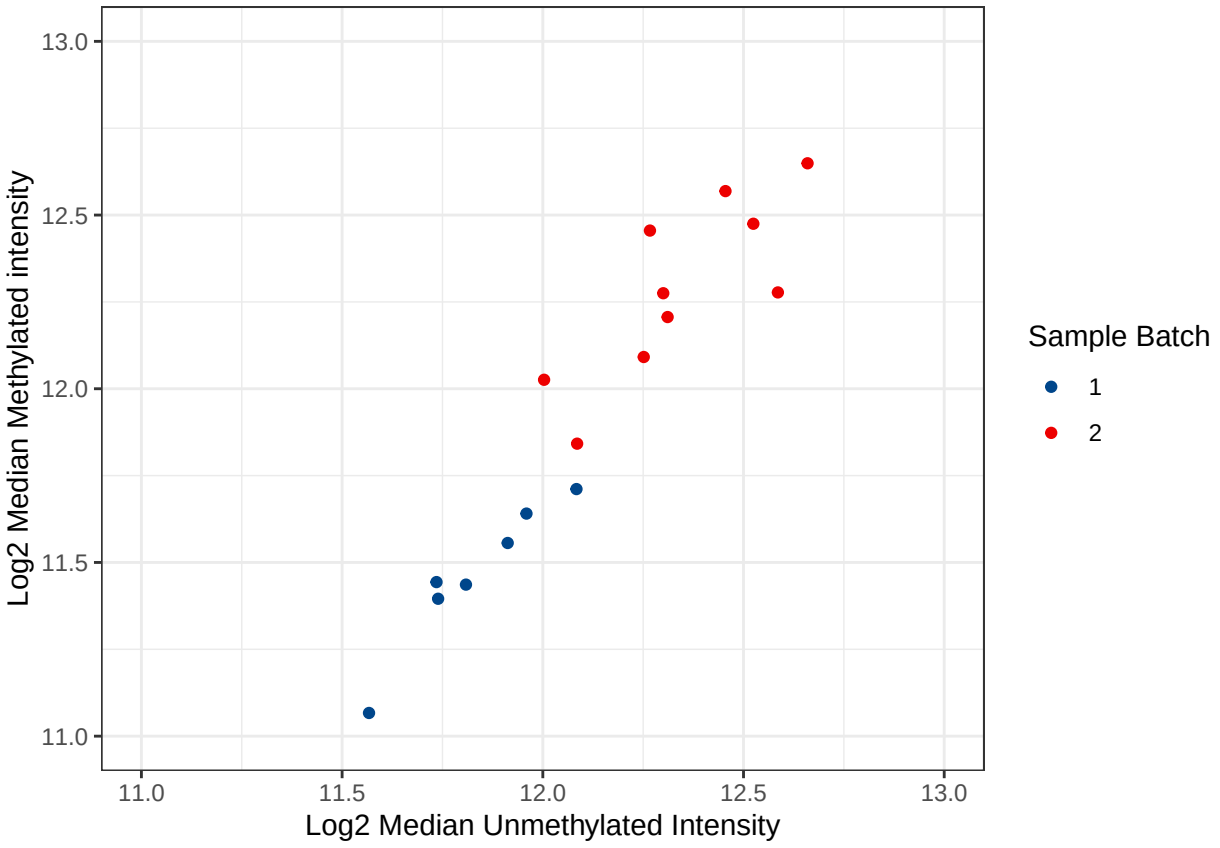


Plate/Batch We now look at Plate which is often called batch as well although batch and plate are not necessarily the exact same thing. In our data (450K chip) there are up to 8 slides in a plate. Within EPIC it switches to 12 slides per plate. So for 450K array a plate is up to 8 slides of 12 samples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

```
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
```

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
```

```
meth <- read_idats(file.path(data_dir, "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.

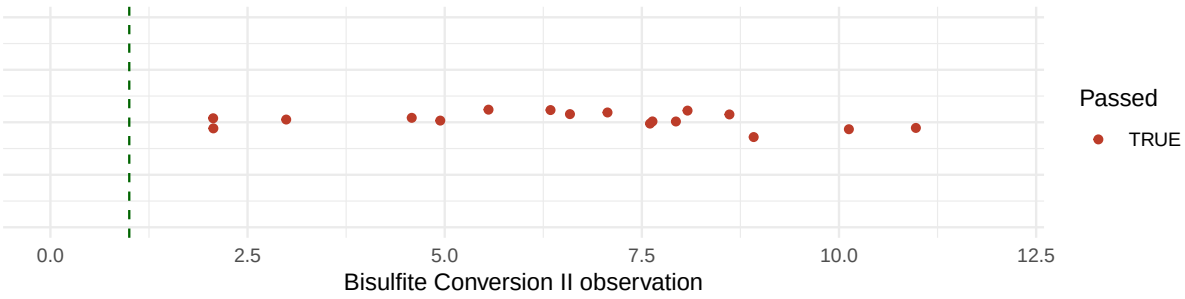
We can visualize a control metric. Each control has a threshold that a samples observation must exceed in order to pass

```
ctrls <- control_metrics(meth)
pd[, ctrlfail := ewastools::sample_failure(ctrls)]
pd[, table(ctrlfail)]
```

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

```
bisulfite <- data.table(measures = ctrls["Bisulfite Conversion II"] %>% unlist)
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")
```



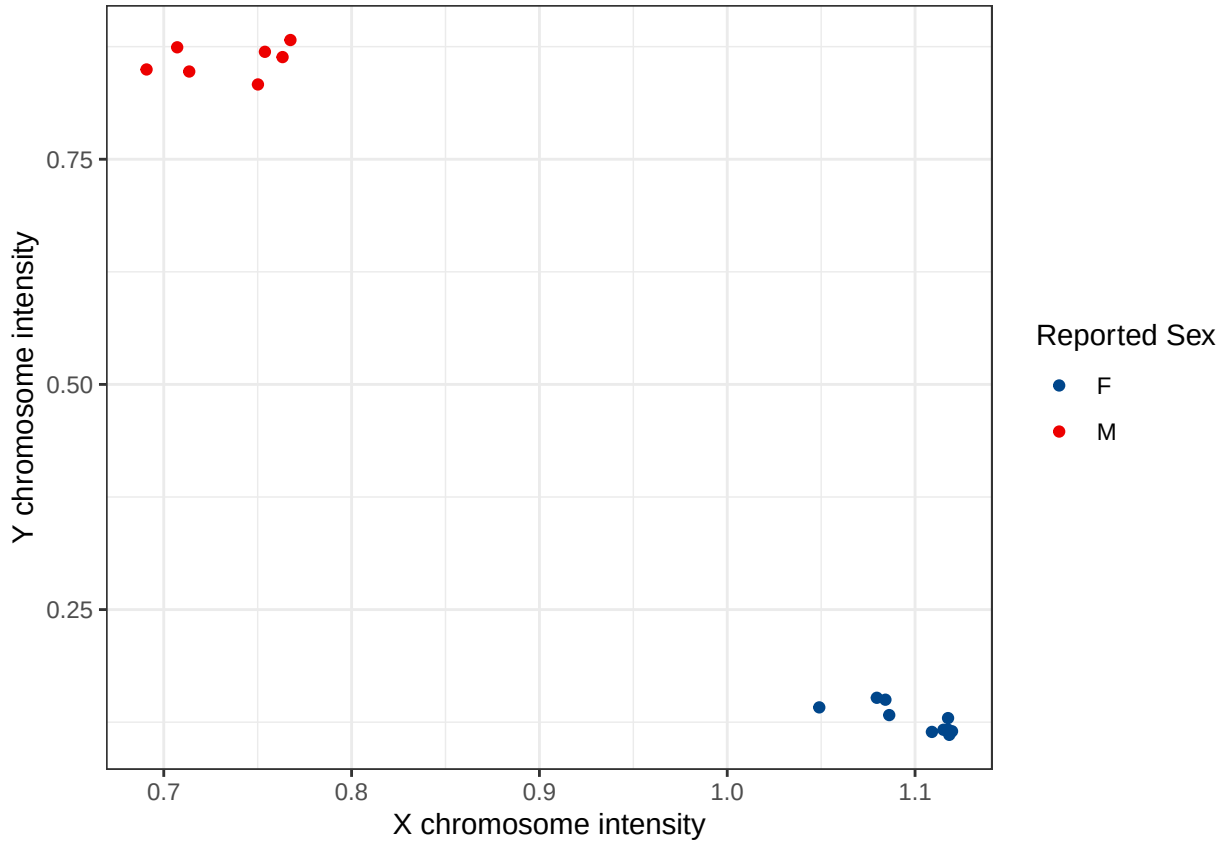

```
rm(ctrls)
```

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

ggplot(pd, aes(X,Y, color = gender)) +
  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's not sure either

Detection P

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.

```
detp <- meth$detP
save(det, file = file.path(data_dir, "detP.rda"))

det0p01 <- detp < 0.01
```

Plot detection P

```
hist(per.samp, breaks = 40, col = "dodgerblue", cex.lab = 1.3, xlab = "Fraction of failed positions per
abline(v = 0.01, col = "red")

hist(per.probe, breaks = 40, col = "dodgerblue", cex.lab = 1.3, xlab = "Fraction of failed samples per p
abline(v = 0.1, col = "red", lwd = 2)
```

Drop problem samples

```
RGset.drop <- RGset[, !colnames(RGset) %in% names(sample.fail)] # Drop samples that failed by detection
dim(RGset.drop)

rm(RGset, detP, failed, per.samp, per.probe)
```

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(file.path(data_dir, "cross.probes.info.rda")) %>% data.table

# rec is to use the 47 cross probe cutoff
```

Gap probes

Gap probes are probes for which the methylation beta clusters into discrete groups– these typically have their methylation driven by underlying SNPs. These probes do not need to be cut in the way that det

```
gaps <- gaphunter(beta)

str(gaps)

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

gap1 <- beta[rownames(gaps$sampleresults)[42],]
gap1 <- data.frame(gap1)

ggplot(gap1, aes(x = rownames(gaps$sampleresults)[42], y = gap1)) +
  geom_jitter(width = 0.01) +
  labs(title = sprintf("Methylation Beta Values for probe %s", rownames(gaps$sampleresults)[42]), 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.

```
# Cell type

# library(FlowSorted.Blood.450k)
# cell <- estimateCellCounts(RGset.drop)
# save(cell, file=file.path(data_dir,"cell-type-estimates.rda"))

# cell estimation may take more memory than available in RStudio Cloud, if an issue load premade cell t
load(file.path(data_dir, "Premade_Intermediate_Files/cell-type-estimates.rda"))

dim(cell)
head(cell)
identical(rownames(cell), colnames(beta)) # A quick sanity check to make sure our samples are in the ri
pd.n <- data.frame(pd, cell)

rm(RGset.drop)
```

Principal components on cell proportions

```
prin.cell <- prcomp(t(cell), center = T, scale. = F)
out.var <- prin.cell$sdev^2 / sum(prin.cell$sdev^2)
out.var


screeplot(prin.cell, col = "dodgerblue", xlab = "Principal Components of Estimated Cell Type", main = "

myColors <- c("seagreen3", "dodgerblue", "darkorchid", "firebrick1", "darkorange", "khaki1")
palette(myColors)

par(mar = c(0, 0, 0, 0))
plot.new()
legend("bottom", c("Bcell", "CD4T", "CD8T", "Gran", "Mono", "NK"), fill = myColors, title = "Principal
pairs(prin.cell$x[, 1:6], col = as.factor(rownames(prin.cell$x)), labels = c("PC1", "PC2", "PC3", "PC4"

summary(pd.n$Gran)

hist(pd.n$Gran, breaks = 20, col = "dodgerblue", xlab = "Estimated percent granulocytes", cex.lab = 1.3)
hist(pd.n$Mono, breaks = 20, col = "dodgerblue", xlab = "Estimated percent monocytes", cex.lab = 1.3)
hist(pd.n$NK, breaks = 20, col = "dodgerblue", xlab = "Estimated percent NK cells", cex.lab = 1.3)
hist(pd.n$CD8T, breaks = 20, col = "dodgerblue", xlab = "Estimated percent CD8T", cex.lab = 1.3)
hist(pd.n$CD4T, breaks = 20, col = "dodgerblue", xlab = "Estimated percent CD4T", cex.lab = 1.3)
```

Drop samples with problematic cell proportions

```
# Pick cutoffs for biological ranges for cell type estimates.
pd.n.drop <- pd.n[pd.n$Mono < 0.2, ]
beta <- beta[, colnames(beta) %in% rownames(pd.n.drop)]
dim(beta)
```

Principal components on noob preprocessed data

```
# Noob PCA plots
pd.n.drop$gran.quart <- cut2(pd.n.drop$Gran, g = 4)

prin <- prcomp(t(beta), center = T, scale. = F)
out.var <- prin$sdev^2 / sum(prin$sdev^2)
out.var[1:10]
```

Plot principal components on noob data

```
screeplot(prin, col = "dodgerblue", xlab = "Principal Components of Noob Beta Values", main = "", cex.la
```

By sex

```
plot.new()
palette(myColors)
par(mar = c(0, 0, 0, 0))
legend("bottom", levels(as.factor(pd.n.drop$gender)), fill = myColors, title = "Principal Components by
pairs(prin$x[, 1:6], col = as.factor(pd.n.drop$gender), labels = c("PC1", "PC2", "PC3", "PC4", "PC5", "
```

By batch

```
plot.new()
legend("bottom", levels(as.factor(pd.n.drop$Batch)), fill = myColors, title = "Principal Components by
pairs(prin$x[, 1:6], col = as.factor(pd.n.drop$Batch), labels = c("PC1", "PC2", "PC3", "PC4", "PC5", "PC
```

By case status

```
plot.new()
legend("bottom", levels(as.factor(pd.n.drop$casestatus)), fill = myColors, title = "Principal Component
pairs(prin$x[, 1:6], col = as.factor(pd.n.drop$casestatus), labels = c("PC1", "PC2", "PC3", "PC4", "PC5
```

By array position

```
plot.new()
palette(graphColors)
legend("bottom", levels(as.factor(pd.n.drop$Array)), fill = graphColors, title = "Principal Components
pairs(prin$x[, 1:6], col = as.factor(pd.n.drop$Array), labels = c("PC1", "PC2", "PC3", "PC4", "PC5", "PC
```

By slide

```
plot.new()
legend("bottom", levels(as.factor(pd.n.drop$Slide)), fill = graphColors, title = "Principal Components
pairs(prin$x[, 1:6], col = as.factor(pd.n.drop$Slide), labels = c("PC1", "PC2", "PC3", "PC4", "PC5", "P
```

By granulocyte proportion quartile

```
plot.new()
legend("bottom", levels(as.factor(pd.n.drop$gran.quart)), fill = graphColors, title = "Principal Compon
pairs(prin$x[, 1:6], col = as.factor(pd.n.drop$gran.quart), labels = c("PC1", "PC2", "PC3", "PC4", "PC5
```

Drop samples with missing data at key covariates

```
# Drop samples with missing data at key covariates
table(pd.n.drop$gender, useNA = "always")
table(pd.n.drop$casestatus, useNA = "always")
table(pd.n.drop$smoking, useNA = "always")
table(pd.n.drop$Batch, useNA = "always")

# No samples missing data, we're okay. But run this line anyway.
pd.complete <- pd.n.drop[!(is.na(pd.n.drop$smoking)), ] # Example of how to remove.
save(pd.complete, file = file.path(data_dir, "pd-complete.rda"))
```

Use Combat to adjust for batch effects

```
betaoob.complete <- beta[, colnames(beta) %in% rownames(pd.complete)]
rm(beta)
mod <- model.matrix(~ pd.complete$gender + pd.complete$casestatus + pd.complete$smoking)

# combat.beta <- ComBat(dat = betaoob.complete, batch = pd.complete$Batch, mod = mod)

# if ComBat requires more memory than available, load premade results:
load(file.path(data_dir, "Premade_Intermediate_Files/combat-beta.rda"))
combat.beta[1:5, 1:5]

# save the cleaned matrix.
save(combat.beta, file = file.path(data_dir, "combat-beta.rda"))

prin <- prcomp(t(combat.beta), center = T, scale. = F)
pd.complete <- data.frame(pd.complete, prin$x)
save(pd.complete, file = file.path(data_dir, "Premade_Intermediate_Files/pd-complete.rda"))

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

Plot principal components of combat data

```
screeplot(prin, col = "dodgerblue", xlab = "Principal Components of Noob Beta Values", main = "", cex.la
```

By sex

```
plot.new()
palette(myColors)
par(mar = c(0, 0, 0, 0))
legend("bottom", levels(as.factor(pd.complete$gender)), fill = myColors, title = "Principal Components
pairs(prin$x[, 1:6], col = as.factor(pd.complete$gender), labels = c("PC1", "PC2", "PC3", "PC4", "PC5",
```

By batch

```
plot.new()
legend("bottom", levels(as.factor(pd.complete$Batch)), fill = myColors, title = "Principal Components by
pairs(prin$x[, 1:6], col = as.factor(pd.complete$Batch), labels = c("PC1", "PC2", "PC3", "PC4", "PC5",
```

By case status

```
plot.new()
legend("bottom", levels(as.factor(pd.complete$casestatus)), fill = myColors, title = "Principal Componer
pairs(prin$x[, 1:6], col = as.factor(pd.complete$casestatus), labels = c("PC1", "PC2", "PC3", "PC4", "PC
```

By array position

```
plot.new()
palette(graphColors)
legend("bottom", levels(as.factor(pd.complete$Array)), fill = graphColors, title = "Principal Components
pairs(prin$x[, 1:6], col = as.factor(pd.complete$Array), labels = c("PC1", "PC2", "PC3", "PC4", "PC5",
```

By slide

```
plot.new()
legend("bottom", levels(as.factor(pd.complete$Slide)), fill = graphColors, title = "Principal Components
pairs(prin$x[, 1:6], col = as.factor(pd.complete$Slide), labels = c("PC1", "PC2", "PC3", "PC4", "PC5",
```

By quartile of granulocyte proportion

```
plot.new()
legend("bottom", levels(as.factor(pd.complete$gran.quart)), fill = graphColors, title = "Principal Compo
pairs(prin$x[, 1:6], col = as.factor(pd.complete$gran.quart), labels = c("PC1", "PC2", "PC3", "PC4", "PC
```