# Brundle Example

*Andrew Holding*

*8/8/2017*

## Brundle Examples

This markdown provides an example of a workflow using Brundle applied to a minmal dataset as included in the BrudleData package.

The packages are found on GitHub as AndrewHolding/Brundle & AndrewHolding/BrundleData. They can be installed with the following code.

```
install.packages("devtools")
library(devtools)

install_github("AndrewHolding/packagename")
```

Once installed we do not need to install them again and can load them as normal.

```
library(Brundle)
library(BrundleData)
```

The intial steps of the Brundle Pipleline are to set the varibles. Here we are using the data from the BundleData package which contains two samplesheets formatted as required by DiffBind. They both refer to the same data, but one provides BED files for only the CTCF peaks while the provides BED files for only the ER regions. The CTCF regions are to provide our control peaks, while the ER binding provides our experimental peaks changes. In this example we have treated MCF7 cells with Fulvestrant.

```
#Set up the intial varible
jg.controlMinOverlap      <- 5
jg.controlSampleSheet     <-
    system.file("extdata", "samplesheet_SLX14438_hs_CTCF_DBA.csv", package =
                    "Brundle")
jg.experimentSampleSheet  <-
    system.file("extdata", "samplesheet_SLX14438_hs_ER_DBA.csv", package =
                    "Brundle")
jg.treatedCondition       =  "Fulvestrant"
jg.untreatedCondition     =  "none"
```

Once configured we load the data from the samples sheets as normal with DiffBind. This provides us with two DiffBind objects. One Experimental and one Control.

```
setwd(system.file("extdata",package="BrundleData"))

dbaExperiment <- jg.getDba(jg.experimentSampleSheet, bRemoveDuplicates=TRUE)
```

```
## 1a MCF7 ERCTCF none  1 macs

## 1b MCF7 ERCTCF Fulvestrant  1 macs

## 2a MCF7 ERCTCF none  2 macs

## 2b MCF7 ERCTCF Fulvestrant  2 macs

## 3b MCF7 ERCTCF Fulvestrant  3 macs

## 3a MCF7 ERCTCF none  3 macs
```

```r
dbaControl    <- jg.getDba(jg.controlSampleSheet, bRemoveDuplicates=TRUE)
```

```
## 1a MCF7 ERCTCF none  1 bed

## 1b MCF7 ERCTCF Fulvestrant  1 bed

## 2a MCF7 ERCTCF none  2 bed

## 2b MCF7 ERCTCF Fulvestrant  2 bed

## 3b MCF7 ERCTCF Fulvestrant  3 bed

## 3a MCF7 ERCTCF none  3 bed
```

We then use Brundle to extract the data from the DiffBind object to generate a peakset. This provides us with the read count at each peak location for each sample.

```r
jg.experimentPeakset <- jg.dbaGetPeakset(dbaExperiment)
jg.controlPeakset    <- jg.dbaGetPeakset(dbaControl)
```

To normalise the data we need to counts of the control and treated samples seperately. This uses the original information we provided at the start of the script to split the control samples into two matrices. For convience we also record the names of the samples relating to each condition.

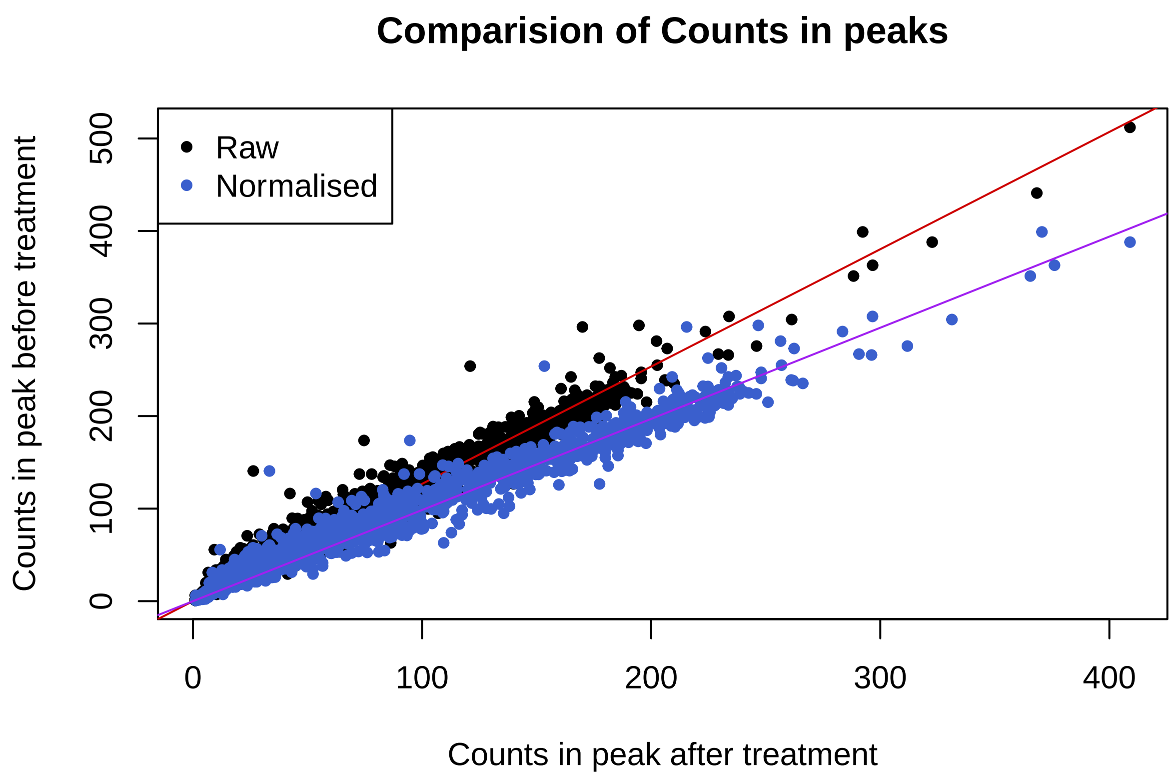
```r
setwd(system.file("extdata",package="BrundleData"))

#Get counts for the treated control samples.
jg.controlCountsTreated<-jg.getControlCounts(jg.controlPeakset,
                                             jg.controlSampleSheet,
                                             jg.treatedCondition )
#Repeat for the untreated/control samples
jg.controlCountsUntreated<-jg.getControlCounts(jg.controlPeakset,
                                               jg.controlSampleSheet,
                                               jg.untreatedCondition)

#Get the sample names for replicates that represent the two conditions.
jg.untreatedNames <- names(jg.controlCountsUntreated)
jg.treatedNames   <- names(jg.controlCountsTreated)
```

Next we generate a normalization coefficient from the data. Typically this is visualized with the included plot function but the step is not required, it can be calculated directly from the data.

```r
jg.plotNormalization(jg.controlCountsTreated,
                     jg.controlCountsUntreated)
```




```
## rowMeans(jg.controlCountsTreated)
##                          1.267618
```

```
#Calculate the noralisation coeffcent
jg.coefficient<-jg.getNormalizationCoefficient(jg.controlCountsTreated,
                                               jg.controlCountsUntreated)
```

To reinsert the data into DiffBind we calculate a correction factor. This is essential as DiffBind will try to normalise our data, this correction factor ensures that our normalization coefficent is applied correctly.

```
setwd(system.file("extdata",package="BrundleData"))

jg.correctionFactor<-jg.getCorrectionFactor(jg.experimentSampleSheet,
                                            jg.treatedNames,
                                            jg.untreatedNames)
```

We then apply the normalisation coefficent and correction factor to the treated samples.

```
jg.experimentPeaksetNormalised<-jg.applyNormalisation(jg.experimentPeakset,
                                                      jg.coefficient,
                                                      jg.correctionFactor,
                                                      jg.treatedNames)
```

For convience the we return the data to DiffBind (using a modified version from https://github.com/andrewholding/flypeaks/tree/master/Diffbind) and use DiffBind to analyze the data. We could then go on to generate a DiffBind report. As this is the analysis of only chromosome 22 we get only a small number of differentially bound sites, nonetheless, the proceedure documented here will work for much larger datasets.

```
jg.dba <- DiffBind:::pv.resetCounts(dbaExperiment,
                                    jg.experimentPeaksetNormalised)

dba.analysis<-dba.analyze(jg.dba)
```

```
## converting counts to integer mode
## gene-wise dispersion estimates
## mean-dispersion relationship
## final dispersion estimates
```

```
dba.plotMA(dba.analysis,bSmooth=FALSE,bFlip = TRUE)
```