Next Generation Sequencing Technologies: Data Analysis and Applications

Sample specific bias correction

Dr. Riddhiman Dhar, **Department of Biotechnology**

Indian Institute of Technology, Kharagpur

Good day, everyone. Welcome to the course on next-generation sequencing technologies, data analysis, and applications. We have started the hands-on transcriptomic data analysis, and the goal is to identify differentially expressed genes among samples that come from different conditions. So, in this class, we will be talking about sample-specific bias correction. So, if you remember in the last hands-on, we looked at some of the preliminary analysis; we looked at the distance-based correlation or heat map. Now, what we will do is actually generate a PCA plot.

So, that is also part of the preliminary analysis, and then we will move on to the sample-specific bias correction. So, again, here we are right. So, we have started working on preliminary data analysis in the last class. We will complete that in this class and then we will move on to the bias correction part.

Once we have done that, we can go into the differential expression analysis, and then we will move on to the visualization of the results of the functional analysis over the subsequent classes. So, let us move to R and let us complete the preliminary analysis part and see if PCA gives us the same data as the distance-based clustering method, and then we will do the bias correction. So, let's move. Let us move into the R terminal, ok? Here we are from the last class; this is where we ended, and we can now generate the PCA plot. So, one of the things I just wanted to mention is that we used Euclidean distance.

So, first, we did something called an rlog transformation here and then calculated the Euclidean distance between the samples. Now, what will the log transformation do? It will reduce variability in the sense that, as we saw in our data for the genes that have low read counts, we reduce variability in the data. So, in that case, we can use the Euclidean distance, but if we want to do this analysis on the raw data counts, we have to calculate something called the poisson distance. Why? Because of this raw data count, there will be a lot more variability, and there is no transformation done right. So, we have to calculate something called poisson distance, and we can calculate that using this library, which is a specific library built for this purpose, and we follow the same steps.

```
library("PoiClaClu")
poisd <- PoissonDistance(t(counts(dds)))

samplePoisDistMatrix <- as.matrix( poisd$dd )
rownames(samplePoisDistMatrix) <- paste( rld$bin, rld$Sample, sep="-" )
colnames(samplePoisDistMatrix) <- NULL
pheatmap(samplePoisDistMatrix, clustering_distance_rows=poisd$dd, clustering_distance_cols=poisd$dd, col
=colors)
```

So, instead of this function that we use, we use this poisson distance function that comes from this library, and then we will generate the sample poisson distance matrix, and we will use this heatmap as before to generate this distance-based clustering or distance-based heat map. So, I will go through this very quickly just to compare the results with our earlier results. So, and we will see, we will get very similar results, ok? Whether you go through the rlog transformation and the

Euclidean distance, or whether you go through the raw count data and use the poisson distance, we get the same results. So, I will simply copy this code because this part is the same as before.

So, instead of this, right? So, we are just using this poisson distance, although we are taking the names from the RLD dollar bin, but that is ok. We are just labeling the row names right with this bin and sample, but the distance data comes from the poisson distance. So, I will do it, and then I will generate this plot. You can see there is a bit of difference, ok? So, I will close this, and then I will generate it fresh, and you can see this data now ok? This is slightly different from before, ok, and again, we have these group names and the sample names, ok.

So, you can see that these G 1, S 5, and G 1, S 9 are the sample names, and again, you notice the same thing: these G 1 samples are very close to each other. So, darker colors mean smaller distances, and white colors mean larger distances, and here the distance is in terms of counts, right? And as you can see, the rest of the groups cluster together very nicely, like G 2, G 3, and G 4, and as before, there is an outlier, G 4: S 16. This sample looks like an outlier; it does not cluster with this group here or with the G 1 group here, ok? So, this gives the same results. So, what you can do now is go into the PCA, right?

So, we can generate this PCA plot, okay? So, to do that, we again have this RLD. We can actually go with this RLD data and this RLOG transformation that we did, and we can use this function called plotPCA. So, this will generate this PCA plot from the data and it will give these PCA percentages as well, ok? So, this is a very easy way to do this, and it will also label based on this bin and the sample. So, let us try this command first.

```
plotPCA(rld, intgroup = c("bin", "Sample"))
```

Based on this log transform data, you see this: we see this PC1 here, and PC1 explains about 63 percent variance, and then you have PC2 on the y axis, which explains about 14 percent variance, and you can see that these G1 groups are around here, right?

So, the samples from the G1 group are clustered together in this PCA plot, and the other groups are clustered around here, as you noticed before. Probably G4: S16 is somewhere here, right? So, from all this analysis, it is clear that we should perhaps discard S16 from our analysis because it looks like an outlier. So, this is something we can do right, and we can come to this conclusion based on these two different preliminary analyses. We can also do this slightly differently, ok? We can generate this PCA data, calculate this percent variance, and use this ggplot library to actually plot the PCA data. So, we generate this PCA data from this plotPCA function right again using the log transform data, and this will store this data inside this PCA data variable.

So, we can run this first command here, ok? This will do the PCA and store the results in this PCA data variable, ok? So, this is done now, right? We have stored this, and it will also calculate this PCA data percentage variation. So, the percentage variation that is explained by this PCA data is okay. So, in our PCA data, when you do PCA analysis, you might have 10 PCs or 100 PCs, and for each of them, you will have this percent variation or percentage variance calculated or explained by that PC. So, we can simply store this as a percentage from this PCA data analysis.

```
pcaData <- plotPCA(rld, intgroup = c( "bin", "Sample"), returnData=TRUE)
percentVar <- round(100 * attr(pcaData, "percentVar"))
```

```
> plotPCA(rld, intgroup = c("bin", "Sample"))
> pcaData <- plotPCA(rld, intgroup = c( "bin", "Sample"), returnData=TRUE)
> percentVar <- round(100 * attr(pcaData, "percentVar"))
> pcaData
           PC1          PC2  group bin Sample name
S1  -20.240039   2.73269148  G1:S1  G1     S1   S1
S5  -12.424835   0.65283797  G1:S5  G1     S5   S5
S9  -13.866712   3.26227616  G1:S9  G1     S9   S9
S13 -16.890199   2.63949785 G1:S13  G1    S13  S13
S2   -1.535204   0.33778676  G2:S2  G2     S2   S2
S6    3.542939  -2.25372221  G2:S6  G2     S6   S6
S10  -1.741976  -1.23485120 G2:S10  G2    S10  S10
S14   7.087036   1.95562439 G2:S14  G2    S14  S14
S3    4.895294   0.25530633  G3:S3  G3     S3   S3
S7    6.674392  -1.68161541  G3:S7  G3     S7   S7
S11   3.449832   1.38354826 G3:S11  G3    S11  S11
S15  13.216794   4.66302504 G3:S15  G3    S15  S15
S4   11.764510   1.69188925  G4:S4  G4     S4   S4
S8   11.101928   0.09073137  G4:S8  G4     S8   S8
S12   7.725211   2.70887633 G4:S12  G4    S12  S12
S16  -2.758973 -17.20390236 G4:S16  G4    S16  S16
```

So, if you look at this PCA data component, So, again, you will have these different components PC1, PC2, group bin, etcetera, and from this PCA data, we are taking this attribute percent variance. So, this is the percentage of variance that is explained by each PC, and we are storing it inside this variable. So, now we can load this and use this PCA data and percent variance to actually plot this, and you will see we will get a very nice plot because we want these groups to be colored similarly. So, you see this here in this plot. It is ok; we can see these are the different group 1 samples on this side, but maybe if we could color them with the same color, it would be actually better.

So, let us try that with this ggplot2, and the commands for doing that are: with this ggplot, we take the PCA data that we generated. The x axis and y axis will be PC1 and PC2; these are the first two principal components that explain the largest amount of variation in the data. The color equals bin right. So, color the points according to the bin right? So, part of which group they are and shape of the points should also be according to the bin, ok? Then you have this plus geom points of points size of this data points, then the x xlab the level of x axis right, and it should say it should read PC. 1 percent variance 1 right percent variance 1 this is this comes from this variable percentVar ok.

So, this is the percent variance, which means this is the variance explained by the first PC. Then

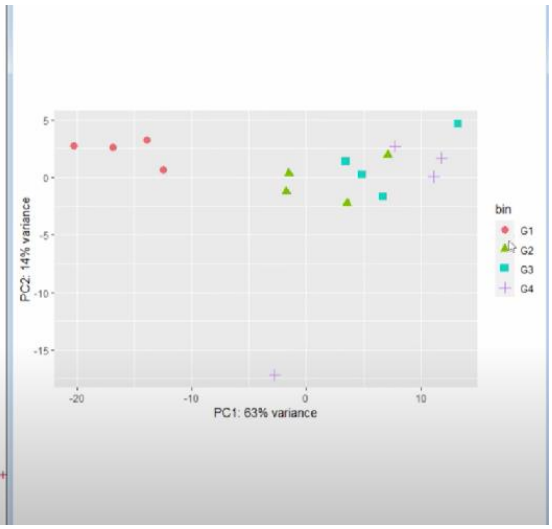
similarly, we have percent variance 2, which is the percentage of variance explained by the PC2. So, these are simply referring to the x level and y level, right, the level of the axis, and then we have the coordinates fixed, ok? So, if you are not familiar with ggplot, do not worry. You can go and look up a bit in this ggplot 2 manual, and you will see that this is how we can specify all these axes, the point size, etcetera. It is a bit different from what we have used before, right?

```
library("ggplot2")

ggplot(pcaData, aes(PC1, PC2, color=bin, shape=bin)) + geom_point(size=3) + xlab(paste0("PC1: ",percentV
ar[1],"% variance")) + ylab(paste0("PC2: ",percentVar[2],"% variance")) +
  coord_fixed()
```

So, for the plot function, inside this plot we mention OK, where the x limit equals to something, and right y limit equals to something. The x level and the y level are everything we mention inside this plot function right in R, but ggplot is different. So, we have this plus, and then whatever you want to add, we just add it with this plus sign. You can see this for all of them, right? So, if you have the geometric point, you have this x level, this y level, and so on, and you can, of course, add a lot more to this and learn that right by exploring these functions. So, we now run this command to see if we can get a nice PCA plot of what is happening. So, what I will do is then think the copying did not work.

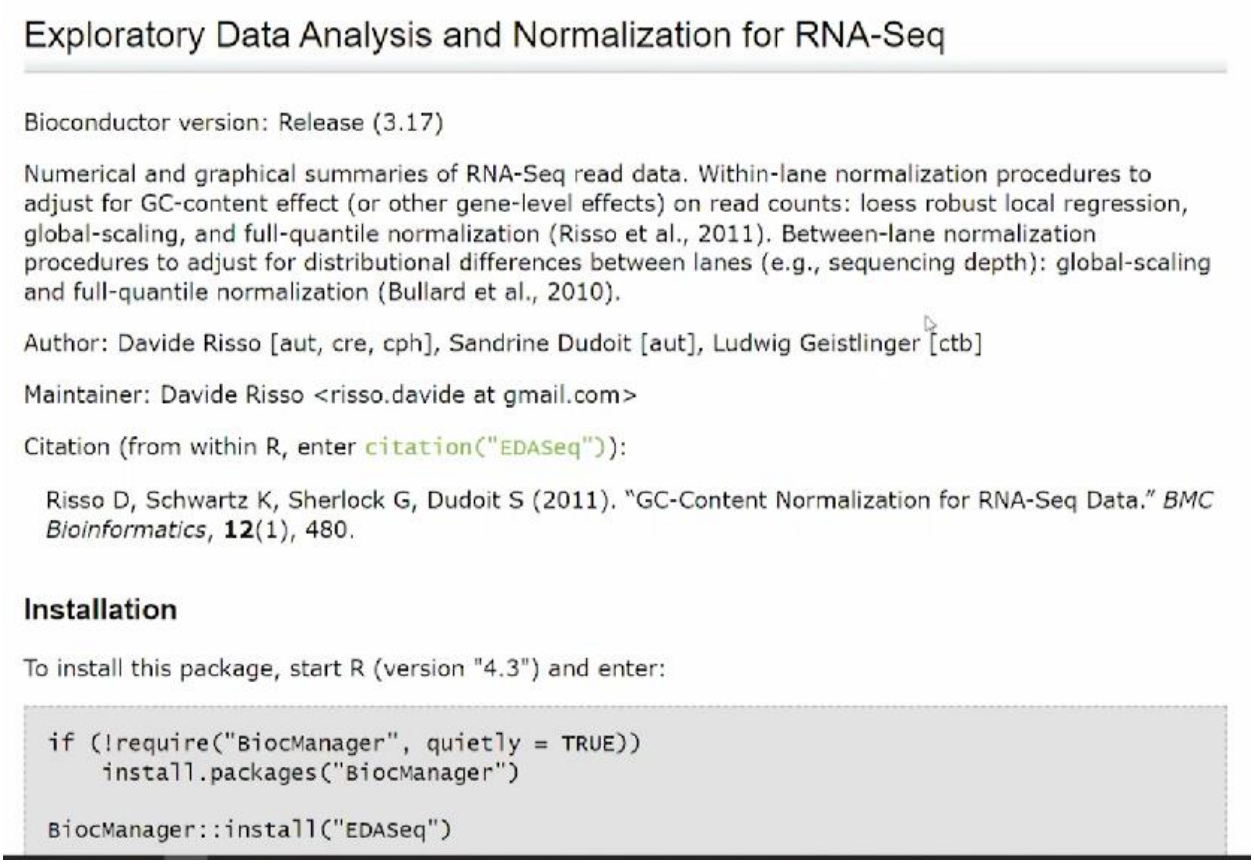
```
null device
          1
> plotPCA(rld, intgroup = c("bin", "Sample"))
> pcaData <- plotPCA(rld, intgroup = c( "bin", "Sample"), returnData=TRUE)
> percentVar <- round(100 * attr(pcaData, "percentVar"))
> pcaData
          PC1          PC2  group bin Sample name
S1  -20.240039   2.73269148  G1:S1  G1     S1   S1
S5  -12.424835   0.65283797  G1:S5  G1     S5   S5
S9  -13.866712   3.26227616  G1:S9  G1     S9   S9
S13 -16.890199   2.63949785 G1:S13  G1    S13  S13
S2   -1.535204   0.33778676  G2:S2  G2     S2   S2
S6    3.542939  -2.25372221  G2:S6  G2     S6   S6
S10  -1.741976  -1.23485120 G2:S10  G2    S10  S10
S14   7.087036   1.95562439 G2:S14  G2    S14  S14
S3    4.895294   0.25530633  G3:S3  G3     S3   S3
S7    6.674392  -1.68161541  G3:S7  G3     S7   S7
S11   3.449832   1.38354826 G3:S11  G3    S11  S11
S15  13.216794   4.66302504 G3:S15  G3    S15  S15
S4   11.764510   1.69188925  G4:S4  G4     S4   S4
S8   11.101928   0.09073137  G4:S8  G4     S8   S8
S12   7.725211   2.70887633 G4:S12  G4    S12  S12
S16  -2.758973 -17.20390236 G4:S16  G4    S16  S16
> library("ggplot2")
>
> ggplot(pcaData, aes(PC1, PC2, color=bin, shape=bin)) + geom_point(size=3) +
+   coord_fixed()
>
```



So, I will just copy the code again, and I will run this code here, and you can see now that this is actually better because each group is colored with one single color, and then also because the shapes are different, and again, this one sample is an outlier here, which we may have to discard

before we actually go into the differential expression analysis. So, I think this preliminary analysis actually gives us a very nice idea about which samples to take for analysis and whether there is an outlier, and it also now tells us G1 is the interesting group, okay, because it stands out from the rest of the samples. So, if you are going for differential expression analysis, we should be considering G1 versus the rest of them, or maybe G1 versus G2, and so on. So, we have now finished this preliminary data analysis, and we can now move on to the bias correction part. So, we have discussed the different types of biases that we can see in RNA-seq data, and we can try to at least identify or minimize some of these biases.

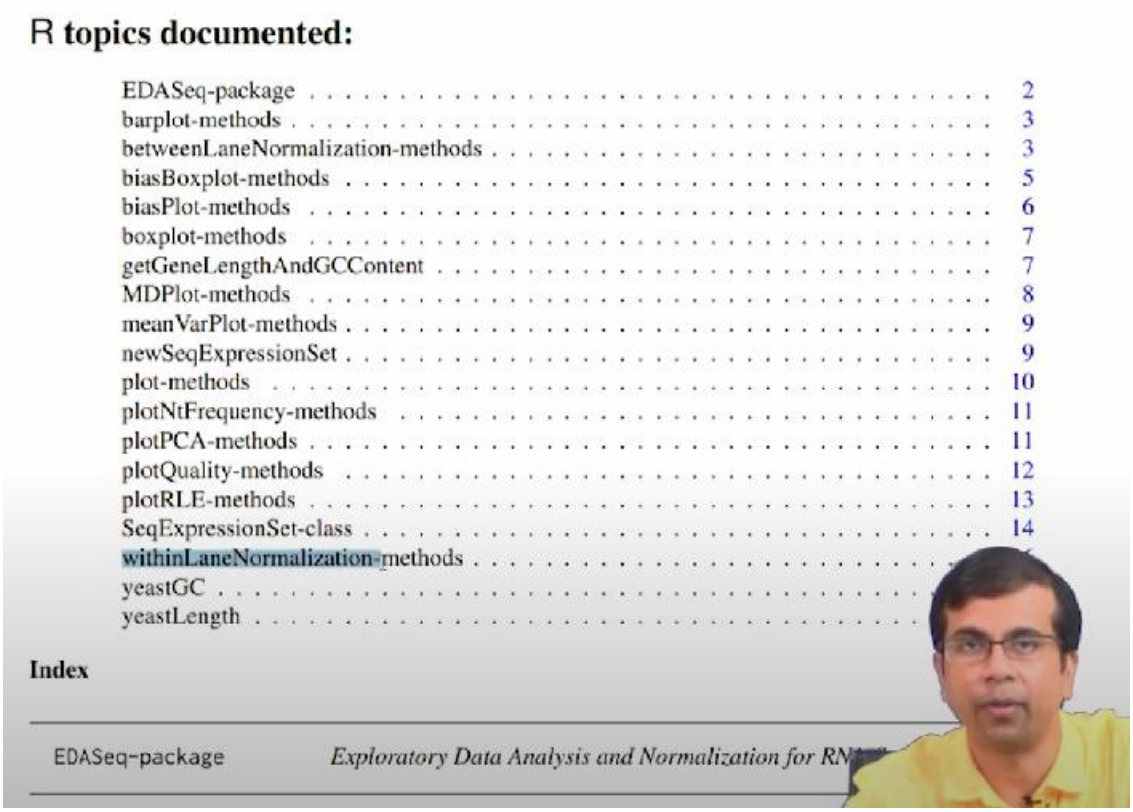
So, that is what we are going to do now in the next step, ok, and for this bias correction part, we will use this package called EDASeq ok. So, again, I will show you how this package looks like it is part of a bioconductor. So, we can simply search here, ok? So, you can see this first result for the bioconductor EDASeq, ok? We have talked about this very briefly in the theory class, right?



So, here again, it can adjust for GC content gene level effects on the counts using different types of methods. So, it has like at least 3 methods that it can do right now: it can do a lowest regression,

global scaling or full quantile normalization, and it can also do lane normalization or means between sample normalization. So, we will just take the within-lane normalization because we want to correct for the biases. These are sample-specific biases that we can correct through the within-lane normalization. So, this is what we are going to do. We will try with the GC content normalization and we will first have to install this package. Let us see if it is there or not, and then if it is not, we can try to install it, and then we will run the course to correct the data. So, let us remove all the figures, etcetera, that we have generated and just also clear the window, and then we can start a fresh one.

So, library EDASeq here, ok? So, it is installed; otherwise, you can simply install it with this command line using the biocmanager, and then we can now use this EDASeq to do the bias correction. So, we will try this for the GC content, and we can again look at the manual and all the functions that are there. So, one of the things we need to do is this within lane normalization, right?



This is the function that we will use for our bias correction part, and as you can see again, we need to create a data format-specific data format called SeqExpressionSet. So, I mentioned that for many packages, you will see that they require a specific data format to be created from the count data or matrix data that they were working with. So, we will do that first, and then we will run this within-Lane normalization, and then we will export the data OK in a text file or a tab-delimited file as we have been doing right now.

```
data <- newSeqExpressionSet(counts, phenoData=AnnotatedDataFrame(data.frame(conditions=cond)))
```

```
withinLaneNormalization(x, y, which=c("loess","median","upper","full"), offset=FALSE, num.bins=10, ro
```

We have seen this count data in a tab-delimited file in a text file. We will do that after we do the bias correction, and we will work with that for differential expression analysis.

So, let us try this part, ok? There are a lot of other functions as well, and you can see some of these functions overlap with other functions. For example, we have the plot PCA and the plot RLE. So, we will talk about plotRLE. Maybe later we will have used plotRLE, like I have shown you some RLE plots before, but we can also use this later on when you do the differential expression analysis to look at the normalized data etcetera. So, again, we can go to this within-lane normalization method or maybe seek expressions set part right. This is a class of data right that we need to create from our count data, and again, this is mentioned as how you can create this right and what kind of attributes or methods you need to include in it. So, and there are some examples of how we can create this right. So, the function is newSeqExpression set right.

So, we have to determine the counts, then you have to give the phenoData variable, and then we have to generate this and assign it to this data variable. So, we will do that, and then once we do that, we need this within-lane normalization method function. Okay, this will do this normalization function, and you can see these different methods that can be used for doing this within-lane normalization or bias correction which will give you that right, and then x is the data that we want right and y is the variable for which we want to do the bias correction right. So, we will take the example of GC content here, and we will do the bias correction for GC content in our data. So, let us move on to our data, ok, and then I have again prepared a set of codes that we can use, ok.

Otherwise, it will not be possible to complete this within this class if you want to search the codes and then search the commands and set up everything. That might take quite a bit of time, ok?

```
rdhar@LAPTOP-3K4C9VBI:/mnt/c/Users/Dhar/Desktop/NGS_Data_Analysis_HandsOn2/Test$ ls
Analyze_RNAseq_data.R    RUN3_all_S1-S16_analysis.txt  SampleInformation.txt
Prelim_analysis_Rcode.R  Rcode_bias_correction.R       steps
rdhar@LAPTOP-3K4C9VBI:/mnt/c/Users/Dhar/Desktop/NGS_Data_Analysis_HandsOn2/Test$ vi Rcode_bias_correctio
n.R
```

```
data=read.table("RUN3_all_S1-S16_analysis.txt",header=T)

library(EDASeq)
data(yeastGC)
data(yeastLength)


sub <- intersect(rownames(data), names(yeastGC))

mat <- as.matrix(data[sub, ])

newdata <- newSeqExpressionSet(mat,normalizedCounts = matrix(data=NA, nrow=nrow(mat), ncol=ncol(mat), di
mnames=dimnames(mat)),
phenoData=AnnotatedDataFrame(
data.frame(conditions=factor(c("G1", "G1","G1","G1","G2", "G2", "G2","G2","G3","G3","G3","G3","G4","G4",
"G4","G4")),
row.names=colnames(data))),
featureData=AnnotatedDataFrame(data.frame(gc=yeastGC[sub])))

norm <- withinLaneNormalization(newdata, "gc", which="full", round=TRUE, offset=FALSE)


biasPlot(newdata,"gc")

norm2 <- as.data.frame(normCounts(norm))
write.table(norm2,"BiasCorrected_RUN3_all_S1-S16_analysis.txt",quote=F,sep="\t")

"Rcode_bias_correction.R" [noeol][dos] 28L, 847B                              1,1           Top
```

So, here it is, right? So, we are loading this data right. This is the raw count data that it will take, and then it will take the library right. So, we will load the library EDASeq, and then we will load these two data sets that are part of this EDASeq library. So, each GC and each length are okay. So, one thing I should mention now is that the data set that we are working with is yeast data. So, it is Saccharomyces cerevisiae data.

So, we can utilize this for each GC content and each length data. So, each GC is length, which is very simple. What is being looked at is the GC content of the genes as well as the length of the genes. So, we need this GC content information if you want to normalize based on the GC content. We can also normalize with the length data, but for this purpose, we will just focus on the GC content-based bias correction. So, let's run this, okay? So, the first thing is we load the data. Press

right in this data variable; maybe we have already loaded, but I just want to run this again to make sure we have everything in order.

So, this data is here. The next part is to load this library, and these two data sets are okay. So, first we load the library EDASeq, then we get these two data points that we will also load ok data yeastGC, and each length is ok; each length is probably not required for this hands-on, but anyway, we will load the library EDASeq, and then these are the data points that are present in the EDASeq library. So, we can simply load the yeastGC data that we use for our analysis. So, the next part is actually creating all the data in the right data format. As I mentioned, we want to create this data format, which is the Seq ExpressionSet, before we can run this within-Lane normalization. So, you can probably see these two parts right there. The first part is looking at the intersection between the row names of the data.

So, what are the row names of this data that we have loaded? These are the gene names and the names of the yeast genes. So, this yeastGC data also has names, and these names are gene names, ok, and we are taking only the intersection right. So, wherever we see the common genes across these two data sets, ok. So, this is the command: sub intersect row names data names yeastGC ok. So, here we are, and we can find out right.

```
sub <- intersect(rownames(data), names(yeastGC))
```

So, these are the gene names that are common. So, this is a large majority. As you can see, out of 6800, about 6700 are present in both of these data sets. So, that is good right? So, we have found most of the genes in these two data sets. Now, what we are doing now is creating this matrix, taking only these genes.

```
[6504] "YPL039W"   "YPL041C"   "YPL044C"   "YPL060C-A" "YPL062W"   "YPL067C"   "YPL068C"
[6511] "YPL071C"   "YPL073C"   "YPL077C"   "YPL080C"   "YPL088W"   "YPL102C"   "YPL107W"
[6518] "YPL108W"   "YPL109C"   "YPL113C"   "YPL114W"   "YPL119C-A" "YPL135C-A" "YPL136W"
[6525] "YPL142C"   "YPL150W"   "YPL152W-A" "YPL162C"   "YPL168W"   "YPL182C"   "YPL185W"
[6532] "YPL191C"   "YPL197C"   "YPL199C"   "YPL205C"   "YPL216W"   "YPL222C-A" "YPL225W"
[6539] "YPL229W"   "YPL238C"   "YPL245W"   "YPL247C"   "YPL250W-A" "YPL251W"   "YPL257W"
[6546] "YPL257W-A" "YPL257W-B" "YPL260W"   "YPL261C"   "YPL264C"   "YPL276W"   "YPL277C"
[6553] "YPL278C"   "YPL283W-A" "YPL283W-B" "YGR198W"   "YOL092W"   "YDR352W"   "YPR002C-A"
[6560] "YPR003C"   "YPR010C-A" "YPR011C"   "YPR012W"   "YPR014C"   "YPR015C"   "YPR016W-A"
[6567] "YPR022C"   "YPR027C"   "YPR039W"   "YPR050C"   "YPR053C"   "YPR059C"   "YPR063C"
[6574] "YPR064W"   "YPR071W"   "YPR074W-A" "YPR076W"   "YPR077C"   "YPR078C"   "YPR084W"
[6581] "YPR089W"   "YPR092W"   "YPR096C"   "YPR097W"   "YPR098C"   "YPR099C"   "YDR368W"
[6588] "YPR108W-A" "YPR109W"   "YPR114W"   "YPR117W"   "YPR123C"   "YPR126C"   "YPR127W"
[6595] "YPR130C"   "YPR136C"   "YPR137C-A" "YPR137C-B" "YPR142C"   "YPR145C-A" "YPR146C"
[6602] "YPR147C"   "YPR148C"   "YPR150W"   "YPR153W"   "YPR158C-C" "YPR158C-D" "YPR158W-A"
[6609] "YPR158W-B" "YPR159C-A" "YPR160C-A" "YPR160W-A" "YPR169W-A" "YPR170C"   "YPR170W-A"
[6616] "YPR170W-B" "YPR172W"   "YPR174C"   "YPR177C"   "YPR195C"   "YPR196W"   "YPR197C"
[6623] "YPR202W"   "YPR203W"   "YPR204C-A" "YPR204W"   "YLR120C"   "YLR121C"   "YGL259W"
[6630] "YIR039C"   "YDR349C"   "YFL038C"   "YBR264C"   "YNL304W"   "YER031C"   "YGL210W"
[6637] "YHR105W"   "YKR014C"   "YNL093W"   "YLR262C"   "YML001W"   "YDR381W"   "YKL214C"
[6644] "YDR002W"   "YIL063C"   "YGL164C"   "YDR545W"   "YER190W"   "YGR296W"   "YLR466W"
[6651] "YLR467W"   "YNL339C"   "YPL283C"   "YOR396W"   "YOR172W"   "YBR054W"   "YOR162C"
[6658] "YBR111C"   "YHR017W"   "YHR016C"   "YNL138W-A" "YLR277C"   "YHR155W"   "YDR326C"
[6665] "YOR003W"   "YKR053C"   "YBR148W"   "YBR162W-A" "YMR089C"   "YPL074W"   "YGR270W"
[6672] "YPR107C"   "YOR272W"   "YNL237W"   "YJR099W"   "YJL139C"   "YOR087W"   "YIR026C"
[6679] "YJL056C"   "YMR273C"   "YML109W"   "YOL109W"   "YNL310C"   "YDR285W"   "YGL249W"
[6686] "YGR211W"   "YOL154W"   "YMR243C"   "YNR039C"   "YER033C"   "YGL255W"   "YLR130C"
[6693] "YKL175W"   "YBR046C"   "YGR285C"   "YNL241C"
```

So, this sub is the list of genes, and we want only these genes and their count data for the bias correction, ok?

```
mat <- as.matrix(data[sub, ])
```

So, this is what we are doing here: we are taking this data part of the data right, a subset of the data, and then we are converting this into a matrix and storing it in this math variable. So, math will have this data now, ok? As you can see again, we have this sample; we have the genes. The only difference is that now we are working with only a subset of genes that are common across these two data sets. So, because we cannot do a GC content correction if we do not have the GC content data for any gene.

```
> mat <- as.matrix(data[sub, ])
> head(mat)
         S1  S5  S9 S13   S2  S6  S10  S14   S3   S7  S11  S15   S4  S8  S12 S16
YMR056C 543 521 602 554 1046 775 1211 1159 1019 1111 1273 1390 1102 968 1310 480
YBR085W  72  53 141  63   44  29   57   27   23   32   63   29   13  30   50  30
YJR155W 584 379 317 324  393 293  513  380  336  359  412  347  312 333  369 171
YNL331C 529 481 564 485  350 365  550  384  316  393  348  419  364 384  353 304
YOL165C   2   0   4   2    1   1    4    0    0    4    1    2    0   2    1   0
YGR107W 468 389 303 407  318 304  320  288  283  323  319  269  249 284  339 255
> |
```

So, that is why we need to work with only the intersection between these two data sets. So, now that this is done right, we are ready to create this new seq ExpressionSet, ok? So, again, it might seem a bit complicated, but we are following the manual exactly as it is given in the manual. So, what we will do is create this new data variable, and we will assign this new SeqExpressionSet right that we created to this variable ok?

So, the first part here is the math, right? So, this is the matrix that we have just created this is the only count data. You can also define this normalized count variable. So, this normalized count variable is just data in a matrix, right? But there is no data now; it will be created, and the number of rows will be the same as the matrix. So, matrix, let us say, 6700 rows This normalized data matrix, or normalized count matrix, should also have the same number of rows and the same number of genes as the n-th column. So, the number of columns should be the same as the matrix file because we have 16 samples here. We should also get 16 samples, and the dimension names will be the same as the matrix. So, that is why everything should be the same as the matrix; only the data will be different after the bias correction.

Then you have the phenoData, which is given as the annotated data frame right, which actually gives us the conditions or factors of these experiments. So, as you can probably see, the first 4 samples belong to group 1, the next 4 samples belong to group 2, the next 4 belong to group 3, and the final 4 belong to group 4. So, this is what we are mentioning when we are saying this factor C, etcetera, etcetera, ok. Then you have the row names right. So, these are the row names given for this data frame. This will be a data file, right? This is the data variable that you have loaded with the column names of that data.

So, these are the sample names, which will be the row names in the case of phenotypes, right? So, phenoData, and then you have the feature data, which actually belongs to this yeastGC sub. So, this is based on the GC content, right? So, we are looking at GC content here. So, this belongs to the yeastGC sub. Again, it might seem complicated, but again, we are following the functions that are given in the manual, and I will show you in a moment.

```
newdata <- newSeqExpressionSet(mat,normalizedCounts = matrix(data=NA, nrow=nrow(mat), ncol=ncol(mat), di
mnames=dimnames(mat)),
phenoData=AnnotatedDataFrame(
data.frame(conditions=factor(c("G1", "G1","G1","G1","G2", "G2", "G2","G2","G3","G3","G3","G3","G4","G4",
"G4","G4")),
row.names=colnames(data))),
featureData=AnnotatedDataFrame(data.frame(gc=yeastGC[sub])))
```

So, then we can run this in R. It is a big command, right? You can see this in that it goes to multiple lines, but in R it is ok, right? It seems to be see this plus plus right, and then we can run the full command, ok? It has generated this new data. Now we can probably type, and it will show you right: this is the SeqExpressionSet class of data. This is what we wanted, right? This is what we want to create before we can run this within-Lane normalization method. And again, you have some statistics about the data: what are these assay data, how many samples, etcetera, sample names, etcetera. All these things that feature names are all given here, ok? So, what I wanted to show you is that we are running this according to the manual, right?

```
> newdata <- newSeqExpressionSet(mat,normalizedCounts = matrix(data=NA, nrow=nrow(mat), ncol=n$
+ phenoData=AnnotatedDataFrame(
+ data.frame(conditions=factor(c("G1", "G1","G1","G1","G2", "G2", "G2","G2","G3","G3","G3","G3$
+ row.names=colnames(data))),
+ featureData=AnnotatedDataFrame(data.frame(gc=yeastGC[sub])))
> newdata
SeqExpressionSet (storageMode: lockedEnvironment)
assayData: 6696 features, 16 samples
  element names: counts, normalizedCounts, offset
protocolData: none
phenoData
  sampleNames: S1 S5 ... S16 (16 total)
  varLabels: conditions
  varMetadata: labelDescription
featureData
  featureNames: YMR056C YBR085W ... YNL241C (6696 total)
  fvarLabels: gc
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation:
>
```

So, we created this new seek data method, as you can probably see. So, we have this normalization, ok? So, I will show you here perhaps in one of the examples you can see this clearly right this data is created like this you can see this right. So, we have just replaced this with our experimental design and our data.

```
data <- newSeqExpressionSet(mat,
                            phenoData=AnnotatedDataFrame(
                              data.frame(conditions=factor(c("mut", "mut", "wt", "wt")),
                                         row.names=colnames(geneLevelData))),
                            featureData=AnnotatedDataFrame(data.frame(gc=yeastGC[sub])))
```

And then we are following the GC content, right? So, when we want to correct for GC content bias, So, that is what we are going to do after we have created this data set, ok? So, the next step is to actually run this within Lane normalization, whether and then use a specific type of method. We can use the lowest method, or we can use full quantile normalization right and then. So, there are three methods that I mentioned. So, we can use any three of them again; these options are given on top, and you can find them, and then we will get the normalized data that will be created.

```
norm <- withinLaneNormalization(newdata, "gc", which="full", round=TRUE, offset=FALSE)
```

And we can then export that normalized data into a text file, ok? So, let us do that now, ok? So, we have created this new set of data, right? So, this is stored in this variable name data, ok, and we now want to do this within lane normalization, ok. So, that is the goal, which is the next step, which is this normalization path.

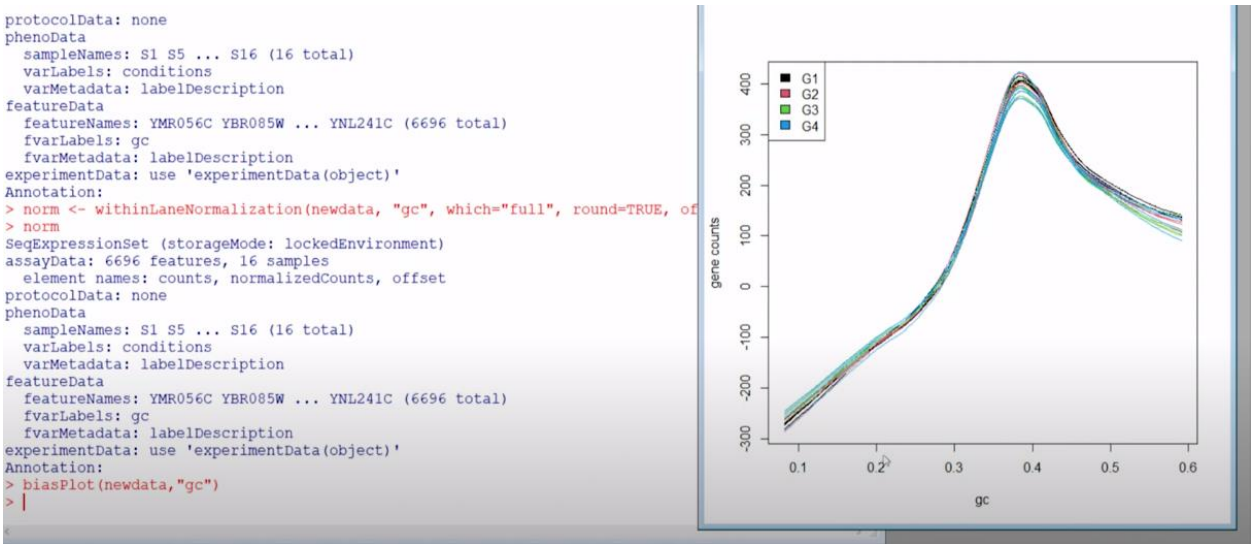
Alright So, let us run this. I have copied this. So, we can run this command, and we can probably see one thing, right? So, this is the data right where we are running this normalization within lane normalization. Based on GC content, we are running full quantile normalization around the values to true because we want these normalized counts to be integers. We will load them in DESeq2, which expects raw counts. So, that is why you want to say round equals true and offset is false. So, there are two options in this package: you can keep the counts the same as the raw count, and then you can set offset values, which will give you the normalized values.

```
> norm <- withinLaneNormalization(newdata, "gc", which="full", round=TRUE, offset=FALSE)
> norm
SeqExpressionSet (storageMode: lockedEnvironment)
assayData: 6696 features, 16 samples
  element names: counts, normalizedCounts, offset
protocolData: none
phenoData
  sampleNames: S1 S5 ... S16 (16 total)
  varLabels: conditions
  varMetadata: labelDescription
featureData
  featureNames: YMR056C YBR085W ... YNL241C (6696 total)
  fvarLabels: gc
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation:
```

But in our case, we just want to create these normalized counts which we will use for the differential expression analysis. So, we have run this now, and we can say the norm is right, and the data is here, ok. So, how do we actually look at the bias? So, whether there is any bias or not,.

So, you can see this by using this command bias plot. Again, this is part of this package. There are other plotting functions as well, and you can use them, and this will give you the bias plot. So, this is the GC content and the gene counts right based on this bias plot, ok? So, of course, gene counts cannot be negative. So, there is some sort of scaling here, ok?

```
biasPlot(newdata,"gc")
```



So, this is how we can check this out. Now what we will do is export this data, right? So, to export

this data, I will use this command: normCounts norm. So, it will get the normalized count data from this normalized data set, right? So, this norm data is the normalized data, and from norm counts, if you use norm counts on norm, we will get the normalized count.

```
norm2 <- as.data.frame(normCounts(norm))
write.table(norm2,"BiasCorrected_RUN3_all_S1-S16_analysis.txt",quote=F,sep="\t")
```

And we want to store this as a data frame inside this variable, norm2. And in the next step, we will write this norm 2 variable using this command. Write dot table right; this is a very similar command as read dot table. So, read dot table will read the data from a text file, whereas write dot table will write this variable data, this matrix data, in a file, and you can say you can mention this file name right, and then you can tell how you should write this. So, this shape equals a slash, which means you should separate the data. Separate the data in every column using tabs, ok? So, we want this tab-separated file quote to mean:. So, if you do not use this command, you will see that all these values will be within double quotes.

```
> head(norm2)
          S1  S5  S9 S13   S2  S6  S10  S14   S3   S7  S11  S15   S4  S8  S12 S16
YMR056C 695 654 667 672 1017 775 1286 1121 1002 1079 1168 1327 1051 950 1211 631
YBR085W  35  26  74  31   22  16   31   15   15   19   32   17   11  19   26  17
YJR155W 237 245 216 215  270 198  362  254  230  246  274  232  210 218  245 124
YNL331C 380 355 403 349  244 246  389  258  215  268  241  272  245 253  233 224
YOL165C   4   0   5   4    3   3    8    0    0    8    3    4    0   3    3   0
YCR107W 372 322 240 312  263 245  272  235  219  250  264  214  192 209  273 213
```

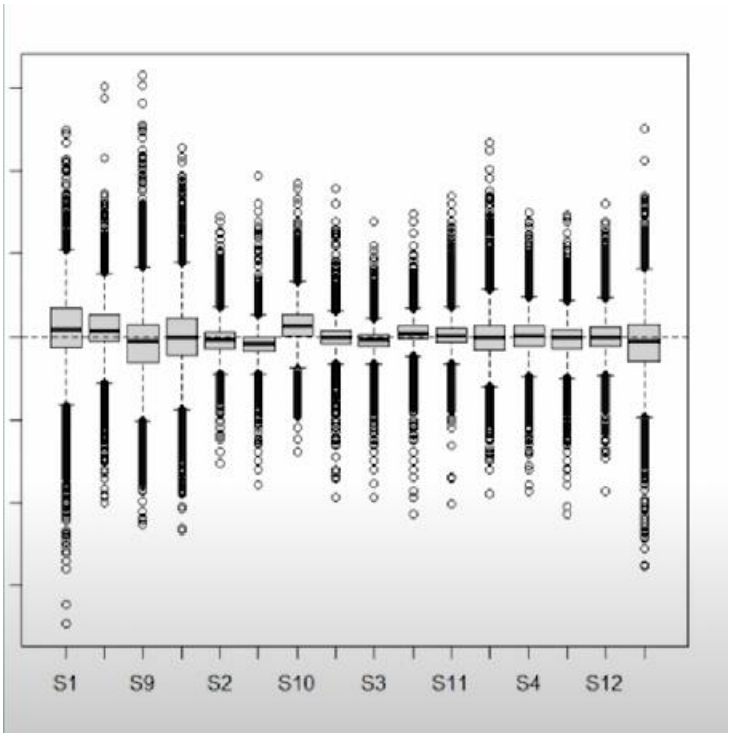
So, we do not want those double quotes, right? So, we will just write. So, a quote equals a false right. So, F just stands for false. Instead of writing the full thing we can simply say F, which means we do not want any quotes when we are writing this data. So, we will run this right, and we will get this normalized count data. And we can check a little bit normalized, ok, maybe we can try head norm 2 right, and you will see this is different from what we have seen earlier. So, from the raw count data, this is actually different, which you can compare, and it should be different because this is corrected for the GC content bias.

Once you have generated this data, we can simply write this table, OK, and we can use this command to write a dot table, and this bias-corrected file will now be generated, OK. And we can

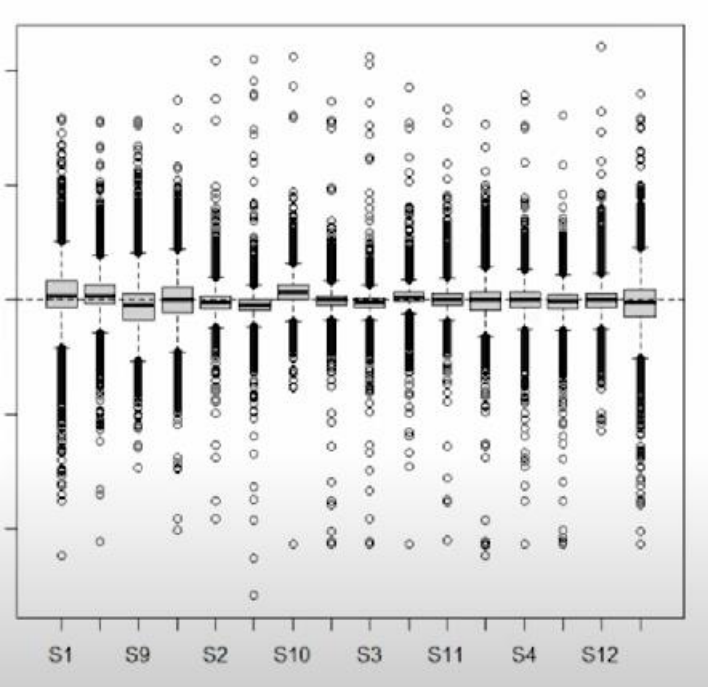
verify this through the command line; we will have a look at that in a moment, ok? Before I look at that file, I just wanted to say that we can also look at this within-lane normalization to see whether this normalization has happened, what kind of normalization has happened, and what kind of adjustment has happened using the RLE plot. So, we have used this command before, and here, also within the package EDASeq, we have this plotRLE command. And we can look at the data before normalization, which is the new data, and also look at the normalized data, which is the norm variable.

```
plotRLE(newdata)
plotRLE(norm)
```

So, we can use this plot for early new data and the plot norm, and we can see very quickly how this normalization changes the data. So, this is what we are going to do, and I will just move this little bit so that you can see the data. So, these are the S1 to S16 samples, and you can see this is before normalization, right?



And after normalization, you can see there is some difference again, depending on the type of normalization. This will change, ok?



You can see the distributions have changed a little bit. So, if this is what a bias correction will do, then we have corrected only for GC bias, and of course, we can take other examples. You can do this for other methods. One final thing we want to see is that we have created the bias-corrected file. So, this code is complete. Now that we have run this code we can come out, and we just want to see that we have created this bias-corrected file. So, here is the bias-corrected file, and we can open this with Vi to see again that we have created this file and saved it in the right format. We will load this data next when we do the differential gene expression analysis, ok?

```
rdhar@LAPTOP-3K4C9VBI:/mnt/c/Users/Dhar/Desktop/NGS_Data_Analysis_HandsOn2/Test$ ls
Analyze_RNAseq_data.R                      RUN3_all_S1-S16_analysis.txt  Steps
BiasCorrected_RUN3_all_S1-S16_analysis.txt  Rcode_bias_correction.R
Prelim_analysis_Rcode.R                    SampleInformation.txt
rdhar@LAPTOP-3K4C9VBI:/mnt/c/Users/Dhar/Desktop/NGS_Data_Analysis_HandsOn2/Test$ vi BiasCorrected_RUN3_a
ll_S1-S16_analysis.txt
```

	S1	S5	S9	S13	S2	S6	S10	S14	S3	S7	S11	S15	S4
YMR056C	695	654	667	672	1017	775	1286	1121	1002	1079	1168	1327
YBR085W	35	26	74	31	22	16	31	15	15	19	32	17
YJR155W	437	245	216	215	270	198	362	254	230	246	274	232
YNL331C	380	355	403	349	244	246	389	258	215	268	241	272
YOL165C	4	0	5	4	3	3	8	0	0	8	3	4
YCR107W	372	322	240	312	263	245	272	235	219	250	264	214
YDL243C	399	296	350	321	256	212	404	190	177	201	245	162
YFL056C	217	159	233	186	113	139	189	143	147	162	155	158
YNL141W	288	504	531	424	511	696	509	696	484	626	402	325
YHR047C	2065	2875	1619	2755	3302	1942	2757	1703	2456	2040	2647	1443
YBL074C	154	119	127	144	104	102	151	124	124	118	114	132
YKL106W	195	220	206	156	194	260	361	341	314	382	447	402
YLR027C	10183	11204	7804	11083	13115	6342	15615	7985	9312	6864	13181	6309
YBR236C	2327	2059	1854	1931	1899	1794	2089	1970	1857	2145	1750	2063
YKL112W	4189	3578	2680	3379	2239	1595	2702	1283	1633	1806	2484	1218
YMR072W	15227	16807	6637	13240	11324	11924	11880	12333	9307	13391	8439	11833
YJR108W	26	27	21	22	19	24	24	23	16	32	23	24
YCR088W	5370	5164	3995	5203	4135	3345	4734	3721	3810	3966	3679	3657
YOR239W	5891	5002	2035	4742	3860	3336	3607	3955	3780	4013	3013	3808
YNR033W	1968	1795	1946	2044	1296	1241	1565	1349	1338	1409	1268	1415
YMR289W	827	782	592	728	715	777	988	919	909	1043	837	1160
YER045C	505	506	456	394	332	236	466	288	304	359	369	219
YGR037C	2815	2590	2052	2327	1360	1346	1320	1360	1214	1316	1119	1334
YNR016C	5899	5410	3960	4480	2254	2351	3282	1578	1919	2631	2513	1805
YLR131C	857	951	949	815	525	486	898	435	313	503	638	452
YLR144C	1252	1426	1064	1090	1034	1077	1611	1278	1118	1349	1324	1249

:se nowrap 1,1 Top

So, to summarize, we have completed the preliminary analysis, and we have seen whether the samples are correlated or whether there is an outlier in the sample based on two types of analysis: distance-based clustering and principal component analysis. And both gave us very similar results, right? So, we get some very useful insights into the data before we actually jump into the differential expression analysis part. And then in the second part we have looked into the bias correction. There are different types of biases that represent the data in this answer. We simply looked at the GC content bias and corrected for that bias, and we have generated this bias corrected data that we can use now for the differential gene expression analysis. Thank you.