```
download.file("https://raw.githubusercontent.com/ErinBecker/2018-03-20-genentech/gh-pages/sampleData.csv", "data/sampleData.csv")

download.file("https://raw.githubusercontent.com/ErinBecker/2018-03-20-genentech/gh-pages/countData.csv", "data/countData.csv")

sampleData <- read.csv("data/sampleData.csv", row.names = 1, comment.char = "#")
countData <- read.csv("data/countData.csv", row.names = 1)

# The data used in this workflow is stored in the airway package that summarizes an RNA-seq experiment
# wherein airway smooth muscle cells were treated with dexamethasone,
# a synthetic glucocorticoid steroid with anti-inflammatory effects (Himes et al. 2014).
# Glucocorticoids are used, for example, by people with asthma to reduce inflammation of the airways.
# In the experiment, four primary human airway smooth muscle cell lines were treated with
# 1 micromolar dexamethasone for 18 hours.
# For each of the four cell lines, we have a treated and an untreated sample.


#check dimension of countData
dim(countData)

#check structure of sampleData
str(sampleData)
## all strings are converted to factors automatically

#check out cell column
sampleData$cell

#subset sample data to extract rows for only cell line N052611
sampleData_N052611 <- sampleData[sampleData$cell == "N052611", ]

# subset sample data to extract rows for only trt
sampleData_trt <- sampleData[sampleData$dex == "trt", ]

# use dplyr
#library(dplyr)

#sampleData_trt <- sampleData %>% filter(dex, "trt")

#subset sample data to extract rows for only cell lines N052611 and N080611
sampleData_N052611_N080611 <- sampleData[sampleData$cell %in% c("N052611","N080611"),]

## countData
# explore countData
head(countData)
min(countData$SRR1039508)
max(countData$SRR1039508)
mean(countData$SRR1039508)
```

```
sum(countData$SRR1039508)

## let's visualize counts for one sample using ggplot2
library(ggplot2)

## histogram of raw counts
ggplot(countData, aes(SRR1039508)) + geom_histogram()

## what are the problems? too many zero counts and large scale

## get rid of zero and low read counts
countData_filtered <- countData[rowSums(countData) > 10,]

## exercise
ggplot(countData_filtered, aes(SRR1039508)) + geom_histogram()

## log transform
countData_filtered_transformed <- log2(countData_filtered + 1)
ggplot(countData_filtered_transformed, aes(SRR1039508)) + geom_histogram(fill="salmon")

# can we compare expressions of two or more genes for one sample?
# no - gene length
# normalize for gene length

# can we compare expression of one gene across samples?
# no - library size (sequencing depth)

## library sizes
librarySizes <- colSums(countData)
## first plot only, then make the pdf file, change file size, change color ...
pdf(file="plots/LibrarySizes.pdf", width=9, height=6)
barplot(librarySizes, cex.names=0.7, col="blue")
dev.off()


### DESeq2 package
## starting from the raw count data

# biocLite() is the recommended way to install Bioconductor packages.
# There are several reasons for preferring this to the 'standard' way in which R pacakges are installed via install.packages().
# Bioconductor has a repository and release schedule that differs from R
# (Bioconductor has a 'devel' branch to which new packages and updates are introduced,
# and a stable 'release' branch emitted once every 6 months to which bug fixes but not new features are introduced).
# A consequence of the mismatch between R and Bioconductor release schedules is that
# the Bioconductor version identified by install.packages() is sometimes not the most recent 'release' available.
# For instance, an R minor version may be introduced some months before the next Bioc release.
```

```
#After the Bioc release the users of the R minor version will be pointed to an out-of-date version of Bioconductor.

source("https://bioconductor.org/biocLite.R")
biocLite("DESeq2")

library(DESeq2)

dds <- DESeqDataSetFromMatrix(countData = countData,
                              colData = sampleData,
                              design = ~ cell + dex)

dds <- dds[ rowSums(counts(dds)) > 1, ]

# The function rlog returns an object based on the SummarizedExperiment class
# that contains the rlog-transformed values in its assay slot.
rld <- rlog(dds, blind = FALSE)
head(assay(rld), 3)


# A useful first step in an RNA-seq analysis is often to assess overall similarity between samples:
# Which samples are similar to each other, which are different?
# Does this fit to the expectation from the experiment's design? etc
# We use the R function dist to calculate the Euclidean distance between samples.
# To ensure we have a roughly equal contribution from all genes, we use it on the rlog-transformed data.
# We need to transpose the matrix of values using t,
# because the dist function expects the different samples to be rows of its argument,
# and different dimensions (here, genes) to be columns.

sampleDists <- dist(t(assay(rld)))

library("pheatmap")

sampleDistMatrix <- as.matrix( sampleDists )

rownames(sampleDistMatrix) <- paste( rld$dex, rld$cell, sep = " - " )
colnames(sampleDistMatrix) <- NULL

# In order to plot the sample distance matrix with the rows/columns arranged by the distances in our distance matrix,
# we manually provide sampleDists to the clustering_distance argument of the pheatmap function.
# Otherwise the pheatmap function would assume that the matrix contains the data values themselves,
# and would calculate distances between the rows/columns of the distance matrix, which is not desired.

pdf(file="heatmap.pdf")
pheatmap(sampleDistMatrix,
      clustering_distance_rows = sampleDists,
      clustering_distance_cols = sampleDists)
dev.off()
```

```
# We can also manually specify a blue color palette using the colorRampPalette function from the
RColorBrewer package.
library("RColorBrewer")
colors <- colorRampPalette( rev(brewer.pal(9, "Blues")) )(255)

pdf(file="heatmap_blue.pdf", width = 10, height = 8)
pheatmap(sampleDistMatrix,
      clustering_distance_rows = sampleDists,
      clustering_distance_cols = sampleDists,
      col = colors)
dev.off()


## PCA plot
# Another way to visualize sample-to-sample distances is a principal components analysis (PCA).
# In this ordination method, the data points (here, the samples) will be projected onto the 2D plane

# Here, we will use the function plotPCA that comes with DESeq2.
# The two terms specified by intgroup are the interesting groups for labeling the samples;
# they tell the function to use them to choose colors.

pdf("pca.pdf")
plotPCA(rld, intgroup = c("dex", "cell"))
dev.off()

# In this ordination method, the data points (here, the samples) are projected onto the 2D plane
# such that they spread out in the two directions that explain most of the differences (figure below).
# The x-axis is the direction that separates the data points the most.
# The values of the samples in this direction are written PC1.
# The y-axis is a direction (it must be orthogonal to the first direction) that separates the data the second
most.
# The values of the samples in this direction are written PC2.
# The percent of the total variance that is contained in the direction is printed in the axis label.
# Note that these percentages do not add to 100%, because there are more dimensions
# that contain the remaining variance
# (although each of these remaining dimensions will explain less than the two that we see).

# We can also build the PCA plot from scratch using the ggplot2 package - homework


### Differential expression
# As we have already specified an experimental design when we created the DESeqDataSet,
# we can run the differential expression pipeline on the raw counts with a single call to the function DESeq:

dds <- DESeq(dds)

# This function will print out a message for the various steps it performs.
# These are described in more detail in the manual page for DESeq, which can be accessed by typing ?DESeq.
```

```
# Briefly these are: the estimation of size factors (controlling for differences in the sequencing depth of the samples),
# the estimation of dispersion values for each gene, and fitting a generalized linear model.

res <- results(dds, contrast=c("dex","trt","untrt"))

# Calling results without any arguments will extract the estimated log2 fold changes and p values
# for the last variable in the design formula. If there are more than 2 levels for this variable,
# results will extract the results table for a comparison of the last level over the first level.
# The comparison is printed at the top of the output: dex trt vs untrt.

summary(res)

## save differentially expressed genes

res_fdr05 <- results(dds, alpha = 0.05)

summary(res_fdr05)

# write.csv(res_fdr05, file="diff_exp_genes.csv", row.names = TRUE)
```