Exploratory data analysis, dimension reduction, and latent variable models

null

null

Homework Problems

Within this notebook, there are *five homework problems* for you to complete. These problems are written in a blockquote:

> *Homework Problem Example 1:* Compute SVD.

Make sure your figures are named "yourlastname_problem1.pdf", "yourlastname_problem2.pdf", "yourlastname_problem3.pdf", "yourlastname_problem4.pdf", "yourlastname_problem5.pdf".

Dependencies

Some packages must be downloaded from CRAN or Bioconductor. R packages on CRAN can be installed with `install.packages()`. Bioconductor packages are installed by using `BiocManager::install()`. There may be challenges in installation procedures. So if basic commands don't work, please search.

```
library(devtools)
library(Biobase)
library(limma)
library(edge)
library(genefilter)
library(qvalue)
library(tidyverse)
library(data.table)
library(corpcor)
```

Load the `ExpressionSet` data

We use the mouse RNA-seq data from the last week. We load the `ExpressionSet` dataset that was saved from the previous week. Please look at the previous week's notebook.

Evaluating gene expression in C57BL/6J and DBA/2J mouse striatum using RNA-Seq and microarrays.

Make sure to apply log2 transformation and remove genes whose expression levels are below a threshold, 10:

```
load(file="bottomly.Rdata")
ls()
```

```
## [1] "bottomly.eset"
```

```
edata <- as.matrix(exprs(bottomly.eset))
dim(edata)
```

```
## [1] 36536    21
```

```
edata[1:5,1:5]
```

```
##                     SRX033480 SRX033488 SRX033481 SRX033489 SRX033482
## ENSMUSG00000000001       369       744       287       769       348
## ENSMUSG00000000003         0         0         0         0         0
## ENSMUSG00000000028         0         1         0         1         1
## ENSMUSG00000000031         0         0         0         0         0
## ENSMUSG00000000037         0         1         1         5         0
```

```
edata <- edata[rowMeans(edata) > 10, ]
edata <- log2(as.matrix(edata) + 1)
```

Create a heatmap with and without clustering the columns. Observe the genes are highly correlated, revealing the systematic variation in the clustered heatmap:

```
library(RColorBrewer)
library(gplots)
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

```
my_palette <- colorRampPalette(c("blue", "white", "orange"))(n = 299)

#My machine requires this option for gplots, ggplot2, etc. to work
options(bitmapType = "cairo")

png("bottomly_heatmap_raw.png",height=700,width=700)
heatmap.2(edata,
          main = "Bottomly et al. Raw", # heat map title
          notecol="black",      # change font color of cell labels to black
          density.info="none",  # turns off density plot inside color legend
          trace="none",         # turns off trace lines inside the heat map
          margins =c(12,9),     # widens margins around plot
          col=my_palette,       # use on color palette defined earlier
          dendrogram="none",     # only draw a row dendrogram
          scale = "row",
          Colv=FALSE)
dev.off()
```

```
## pdf
##   2
```

```
png("bottomly_heatmap_clustered.png",height=700,width=700)
heatmap.2(edata,
          main = "Bottomly et al. Clustered", # heat map title
          notecol="black",      # change font color of cell labels to black
          density.info="none",  # turns off density plot inside color legend
          trace="none",         # turns off trace lines inside the heat map
          margins =c(12,9),     # widens margins around plot
          col=my_palette,       # use on color palette defined earlier
          dendrogram="none",     # only draw a row dendrogram
          scale = "row")
dev.off()
```

```
## pdf
##   2
```

Homework Problem 1: Make one heatmap of the aforementioned Bottomly data with the following options: a) both rows and columns are clustered, b) show a dendrogram only on the columns., and c) scale in the column direction. Send only one heatmap. If you are unsure, check the help document on this function by typing ?heatmap.2

```
pdf("Bielecki_problem1.pdf")
heatmap.2(  #Options inherited from previous heatmap.2() calls
          edata,
          notecol="black",      # change font color of cell labels to black
          density.info="none",  # turns off density plot inside color legend
          trace="none",         # turns off trace lines inside the heat map
          margins =c(12,9),     # widens margins around plot
          col=my_palette,       # use on color palette defined earlier
            #Custom options
          main = "Bottomly et al., clustered H/V",  # heat map title
          Rowv=TRUE,                                # cluster by row
          Colv=TRUE,                                 # cluster by column
          dendrogram="column",                      # column dendrogram
          scale = "column"                           # scale by column
          )
dev.off()
```

```
## pdf
##   2
```

Singular value decomposition (SVD)

Singular value decomposition gives us the left and right singular vectors, and the singluar values (`d`). It is a computationally efficient way to compute principal component analysis.

```
edata <- t(scale(t(edata), scale=FALSE, center=TRUE))
svd.out <- svd(edata)
names(svd.out)
```

```
## [1] "d" "u" "v"
```

```
print(paste("Dimension of left singular vectors:", dim(svd.out$u)))
```

```
## [1] "Dimension of left singular vectors: 8544"
## [2] "Dimension of left singular vectors: 21"
```

```
print(paste("Length of singular values:",length(svd.out$d)))
```

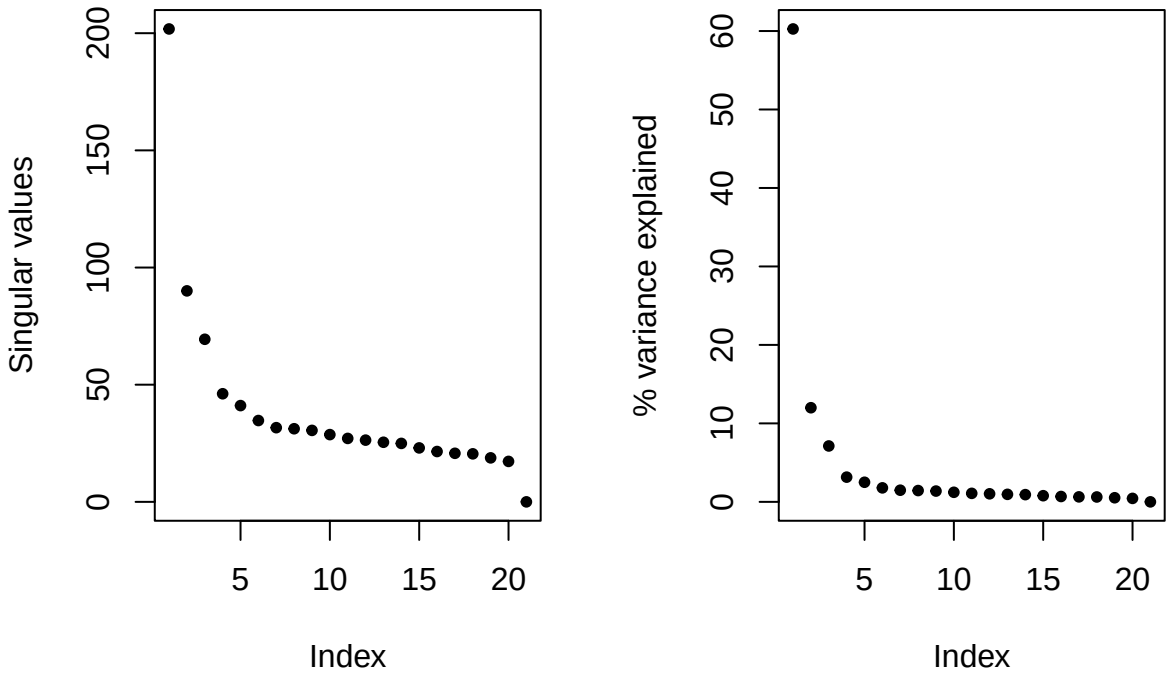
```
## [1] "Length of singular values: 21"
```

```
print(paste("Dimension of right singular vectors:",dim(svd.out$v)))
```

```
## [1] "Dimension of right singular vectors: 21"
## [2] "Dimension of right singular vectors: 21"
```

The key choice one has to make when conducting PCA for genomic data is the dimension `r`. Likely, you would want `r` to be (much) smaller than `min(n,m)` such that you achieve dimension reduction. The first step is looking at the scree plot, that is the variance explained. Often, one want to identify the elbow of the scree plot.

```
par(mfrow=c(1,2))
plot(svd.out$d, pch=20, ylab="Singular values")
plot(svd.out$d^2/sum(svd.out$d^2)*100, pch=20, ylab="% variance explained")
```

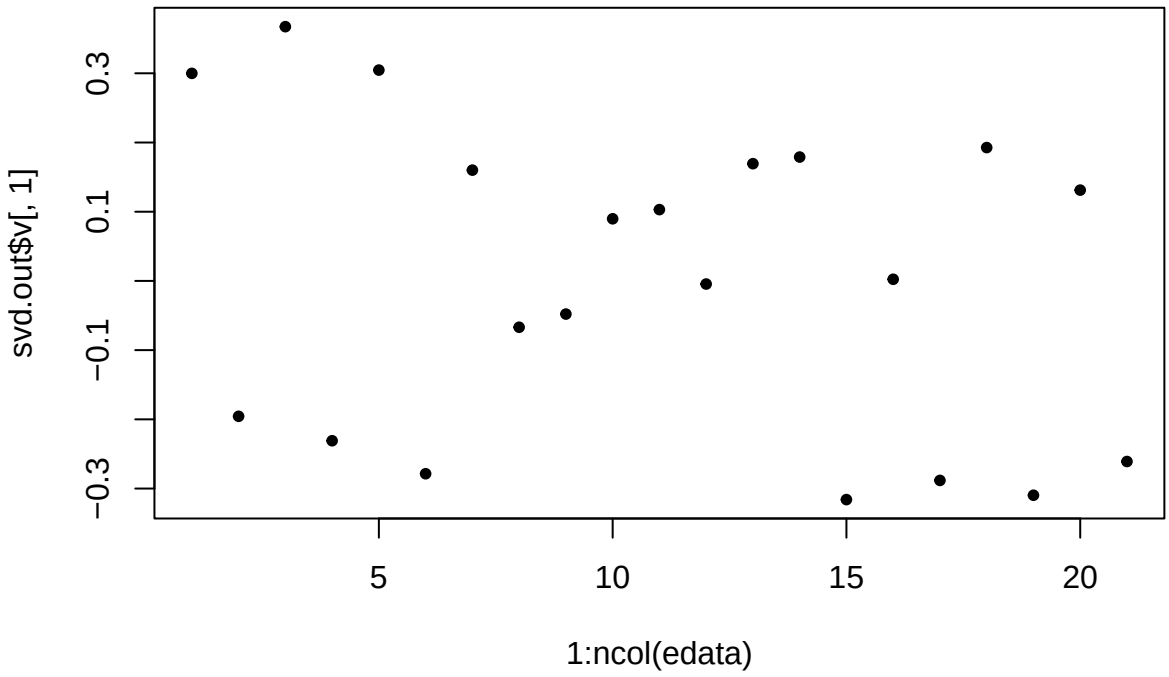


In practice, these results may not be consistent and some of them may not be suitable for your data. If a statistical method, your data may not meet the assumption made by it. Often, consult the biologists or biological knowledge.

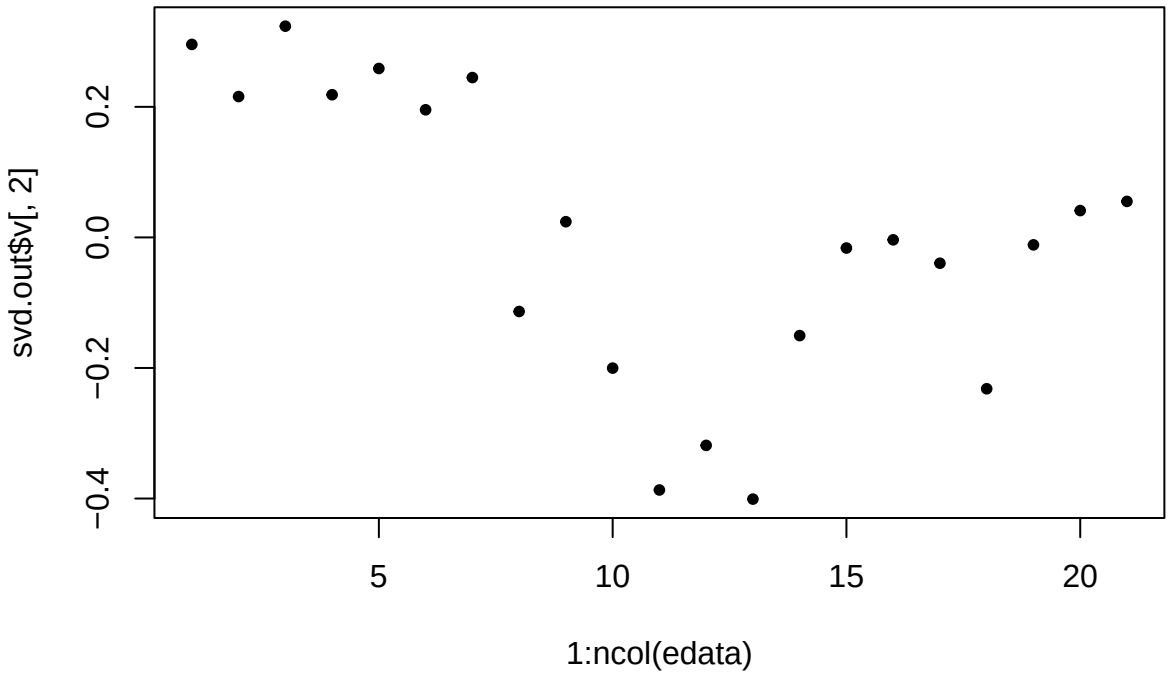
Scatter plots using right Singular Vectors (Principal Components)

Exactly what are principal components and their corresponding loadings depends on the orientation of the input data. In our case, the genes/variables are rows whereas the samples/observations are columns. Then, PCs equals the corresponding singular values times the right singular vectors. These are sometimes called eigengenes to denote that they represents a weighted linear sum of genes (rows). We look at the top 3 right singular vectors:

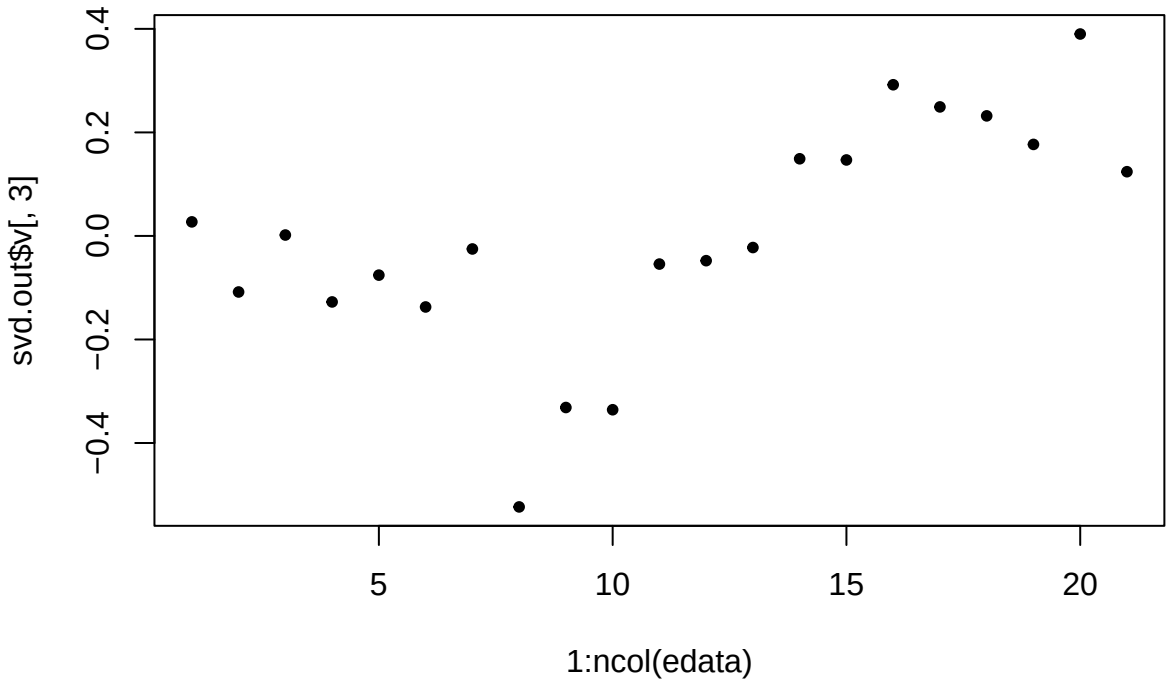
```
plot(1:ncol(edata), svd.out$v[,1],pch=20)
```




```
plot(1:ncol(edata), svd.out$v[,2],pch=20)
```

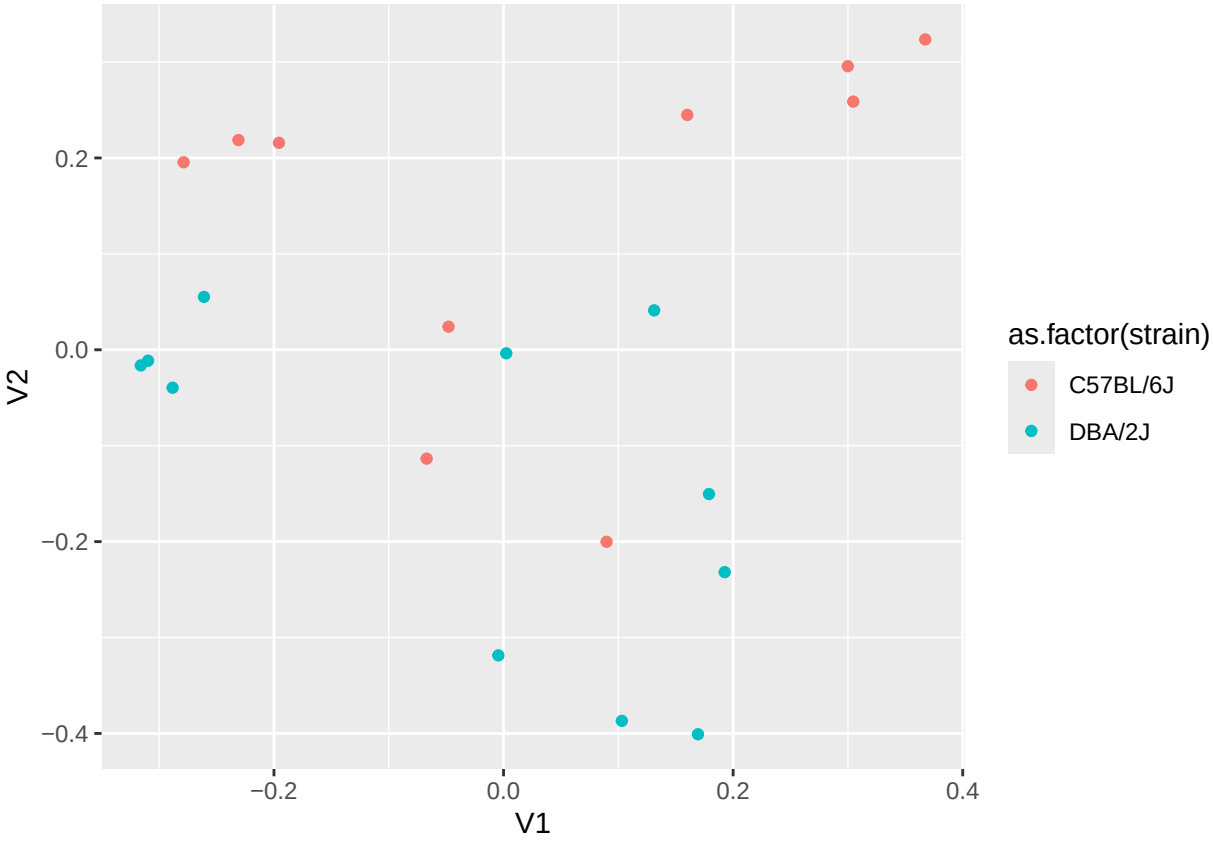



```
plot(1:ncol(edata), svd.out$v[,3],pch=20)
```

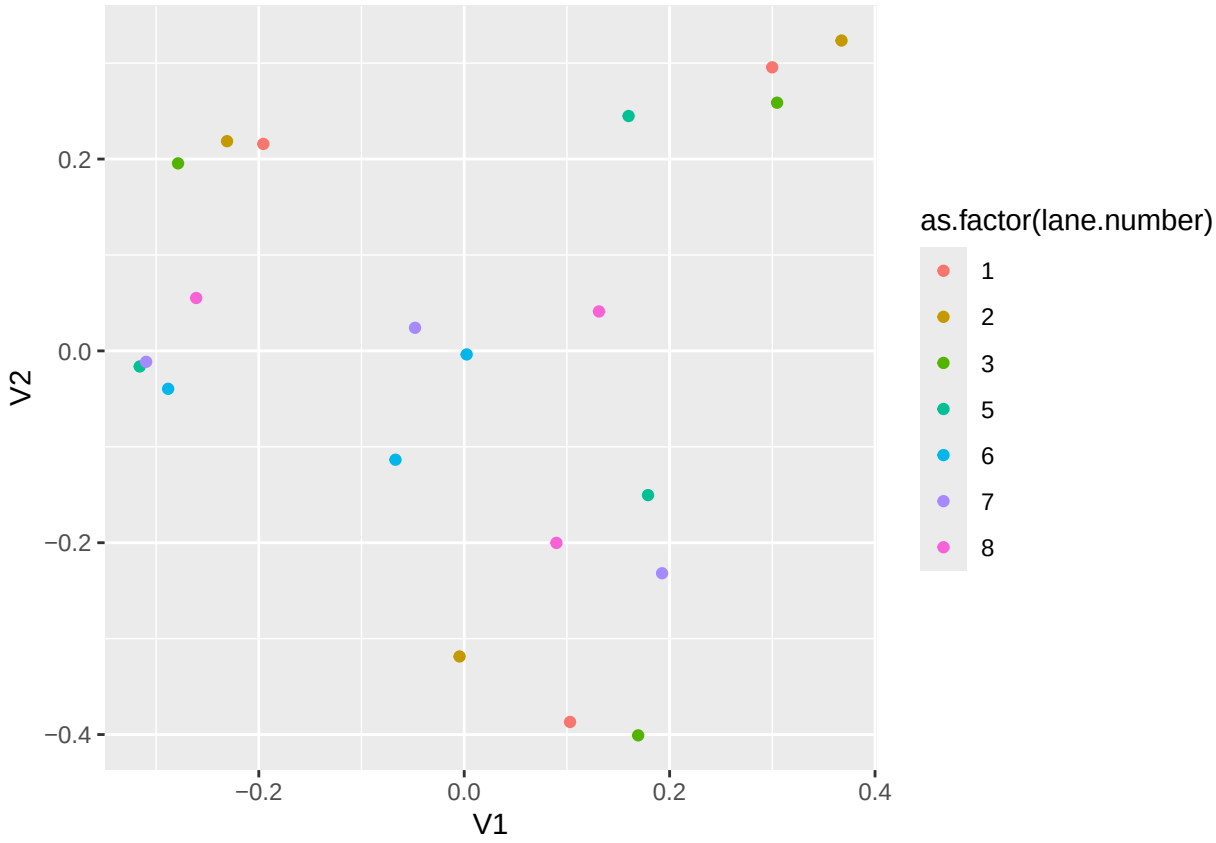


We can make a scatter plot of the top 2 PCs. And using the meta data, we can color each data point accordingly. To do so, we will use ggplot2.

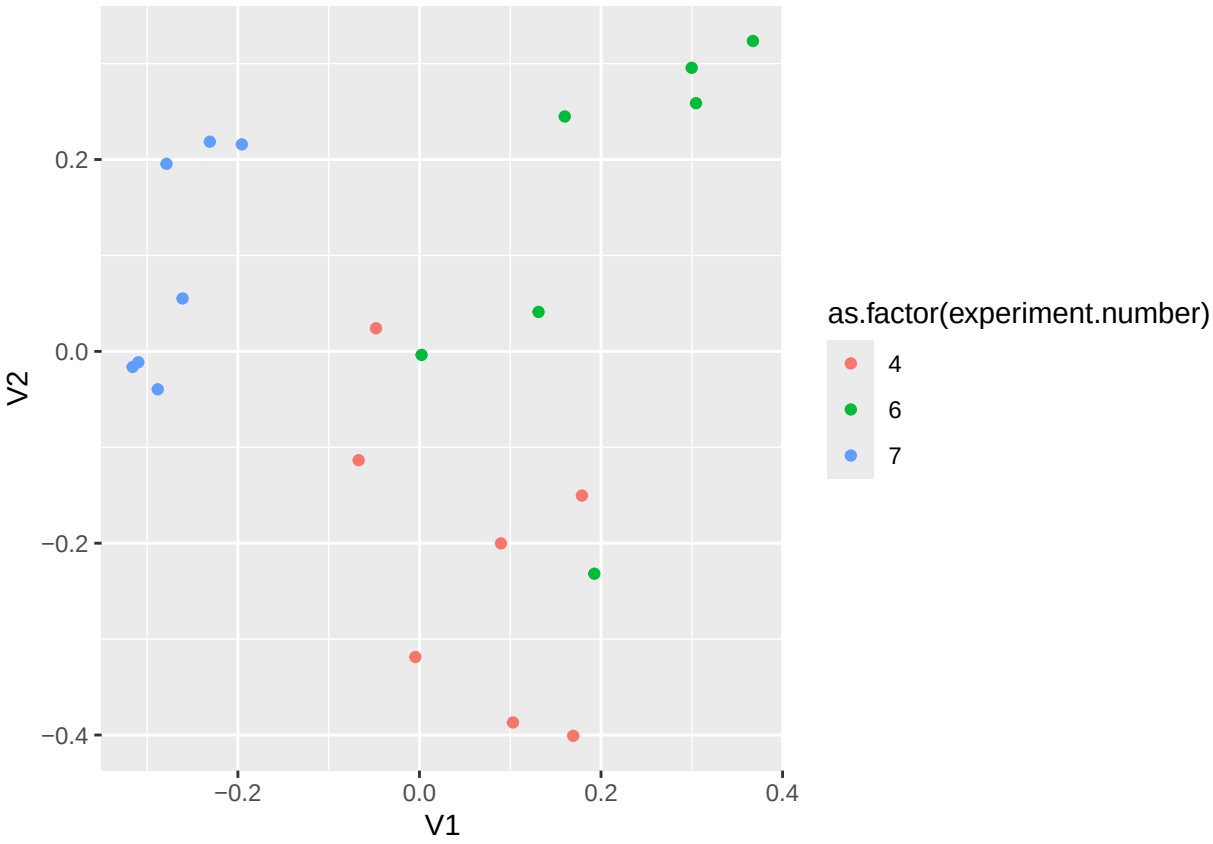
```
PC = data.table(svd.out$v,pData(bottomly.eset))
ggplot(PC) + geom_point(aes(x=V1, y=V2, col=as.factor(strain)))
```



```
ggplot(PC) + geom_point(aes(x=V1, y=V2, col=as.factor(lane.number)))
```




```
ggplot(PC) + geom_point(aes(x=V1, y=V2, col=as.factor(experiment.number)))
```



Homework Problem 2: As shown in the plot above, the projection on the top 2 PCs doesn't show the grouping by the strains. But we have many PCs to explore. Explore different combinations of PCs in scatter plots while coloring the data points by the genetic strains. Find a combination of PCs that separate the strains well. Send only one scatterplot.

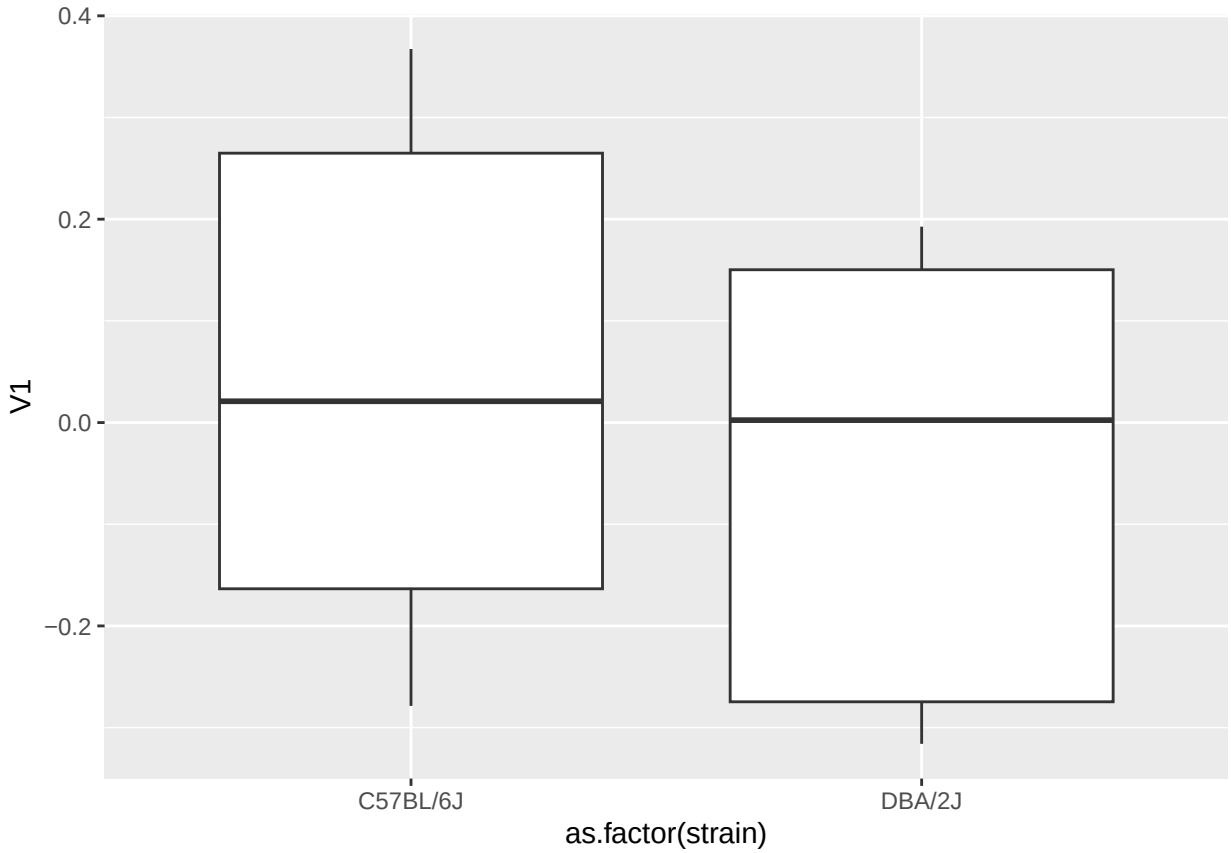
```
pdf("Bielecki_problem2.pdf")
ggplot(PC) + geom_point(aes(x=V1, y=V3, col=as.factor(strain)))
dev.off()
```

```
## pdf
##   2
```

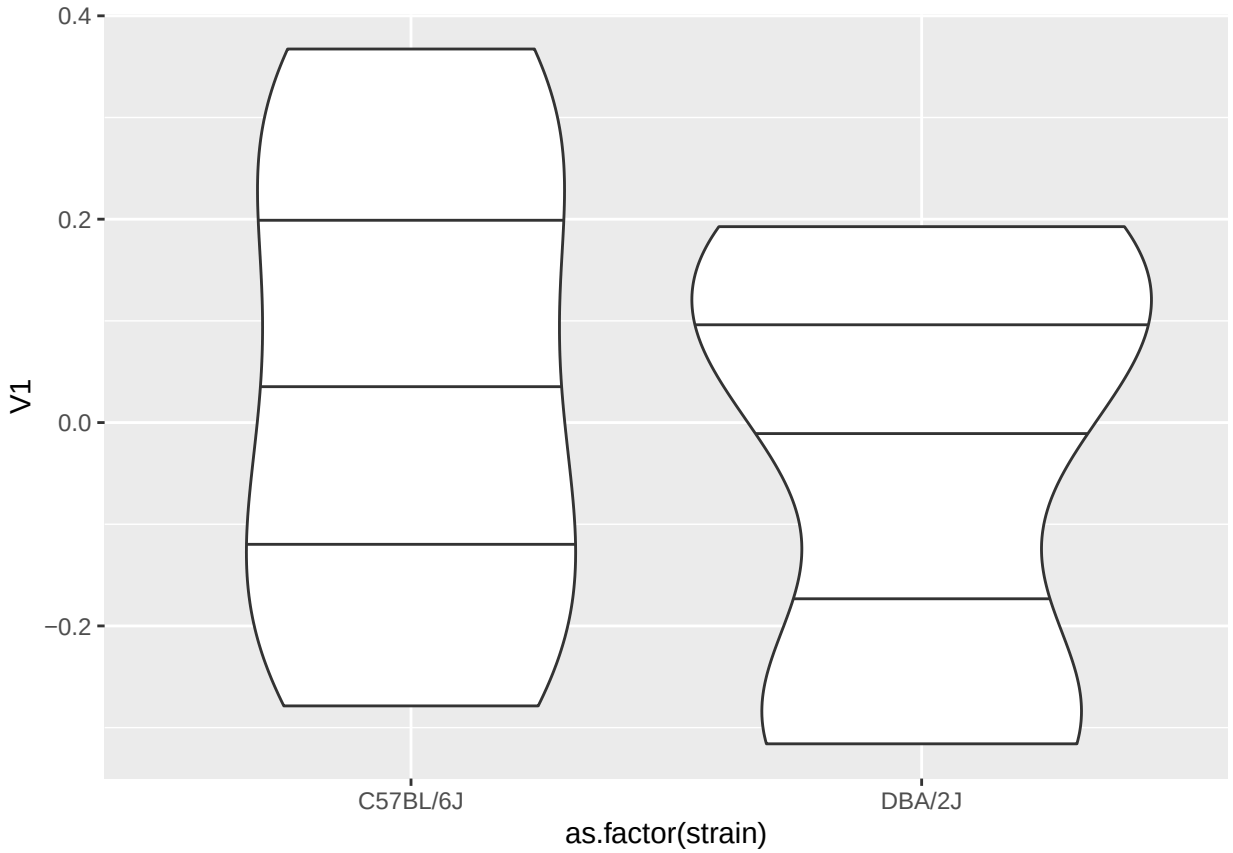
Boxplots and violin plots

Violin plots extend boxplots by showing the density estimates. However, both violin plots and boxplots would be better served when the original values are overlayed (the last plot below).

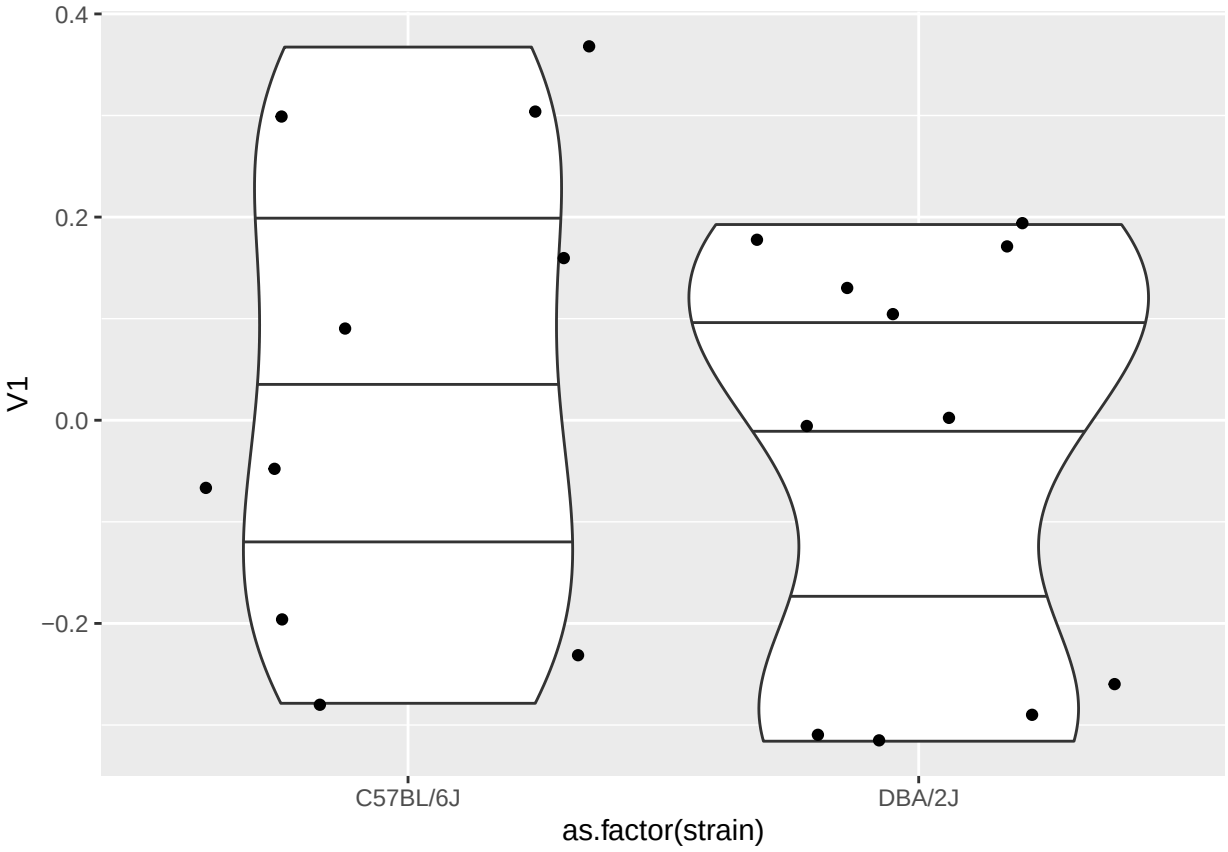
```
ggplot(PC) + geom_boxplot(aes(x=as.factor(strain), y=V1))
```




```
ggplot(PC) + geom_violin(aes(x=as.factor(strain), y=V1),draw_quantiles = c(0.25, 0.5, 0.75))
```




```
ggplot(PC) + geom_violin(aes(x=as.factor(strain), y=V1),draw_quantiles = c(0.25, 0.5, 0.75)) + geom_jit
```



Visualize Left Singular Vectors (Loadings)

As we had done with right singular vectors, we can apply the similar exploration and visualization using left singular vectors. The left singular vectors are often called the loadings of PCs.

Homework Problem 3: Make a scatter plot of the top 2 left singular vectors.

```
#Get left singular vectors
loadings = data.table(svd.out$u,pData(bottomly.eset))
```

```
## Warning: Item 2 has 21 rows but longest item has 8544; recycled with remainder.
```

```
#Plot
pdf("Bielecki_problem3.pdf")
ggplot(loadings) + geom_point(aes(x=V1, y=V2))
dev.off()
```

```
## pdf
##   2
```

Homework Problem 4: Make one figure that contains violin plots of the top 5 left singular vectors (loadings). Hint/To-do: Make sure turn the top 5 left singular vectors into a data.table (or a data.frame) and ggplot2 to plot them altogether. Do not send 5 figures!

```
#Reduce data to first 5 vectors + strain
loadings_df = as.data.frame(loadings[,c(1:5,24)])
clr <- c("#E69F00", "#56B4E9", "#009E73", "#D55E00", "#CC79A7")

#Plot
pdf("Bielecki_problem4.pdf")
```

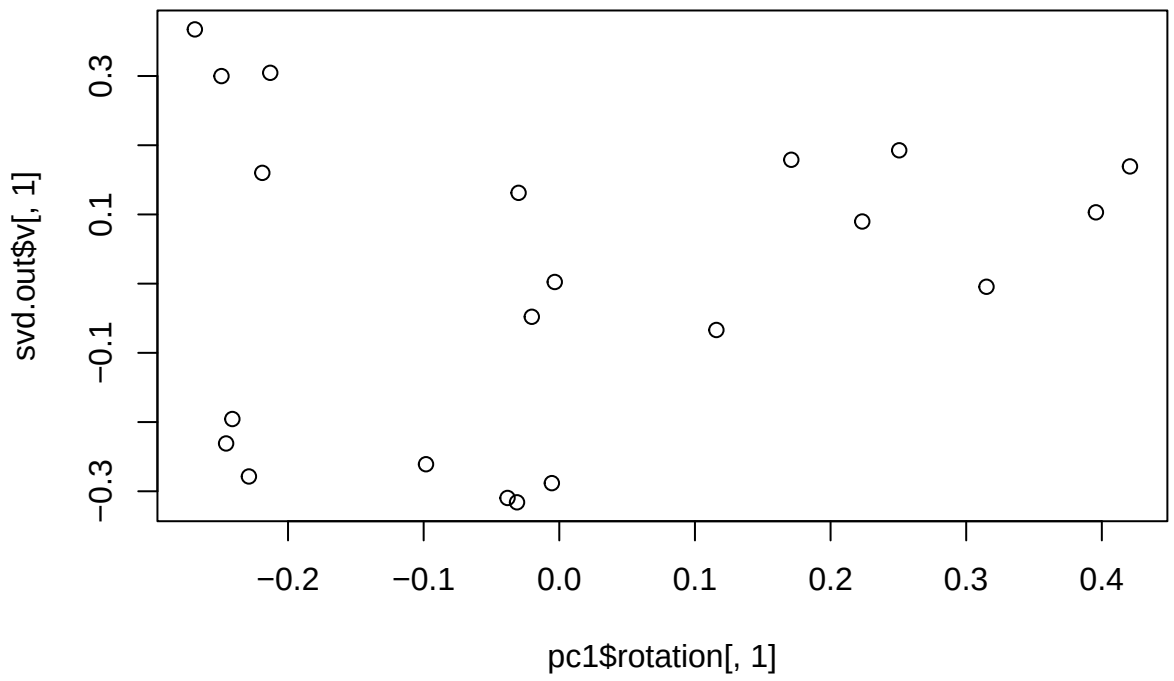
```
ggplot(loadings_df) +
    geom_jitter(aes(x=as.factor(strain), y=V1, color=clr[1])) +
    geom_jitter(aes(x=as.factor(strain), y=V2, color=clr[2])) +
    geom_jitter(aes(x=as.factor(strain), y=V3, color=clr[3])) +
    geom_jitter(aes(x=as.factor(strain), y=V4, color=clr[4])) +
    geom_jitter(aes(x=as.factor(strain), y=V5, color=clr[5])) +
    geom_violin(aes(x=as.factor(strain), y=V1, color=clr[1]),   fill=NA,    draw_quantiles = c(0.25, 0.5
    geom_violin(aes(x=as.factor(strain), y=V2, color=clr[2]),   fill=NA,    draw_quantiles = c(0.25, 0.5
    geom_violin(aes(x=as.factor(strain), y=V3, color=clr[3]),   fill=NA,    draw_quantiles = c(0.25, 0.5
    geom_violin(aes(x=as.factor(strain), y=V4, color=clr[4]),   fill=NA,    draw_quantiles = c(0.25, 0.5
    geom_violin(aes(x=as.factor(strain), y=V5, color=clr[5]),   fill=NA,    draw_quantiles = c(0.25, 0.5
    scale_color_manual(labels = c("V1","V2","V3","V4","V5"), values = clr)
dev.off()
```

```
## pdf
##   2
```

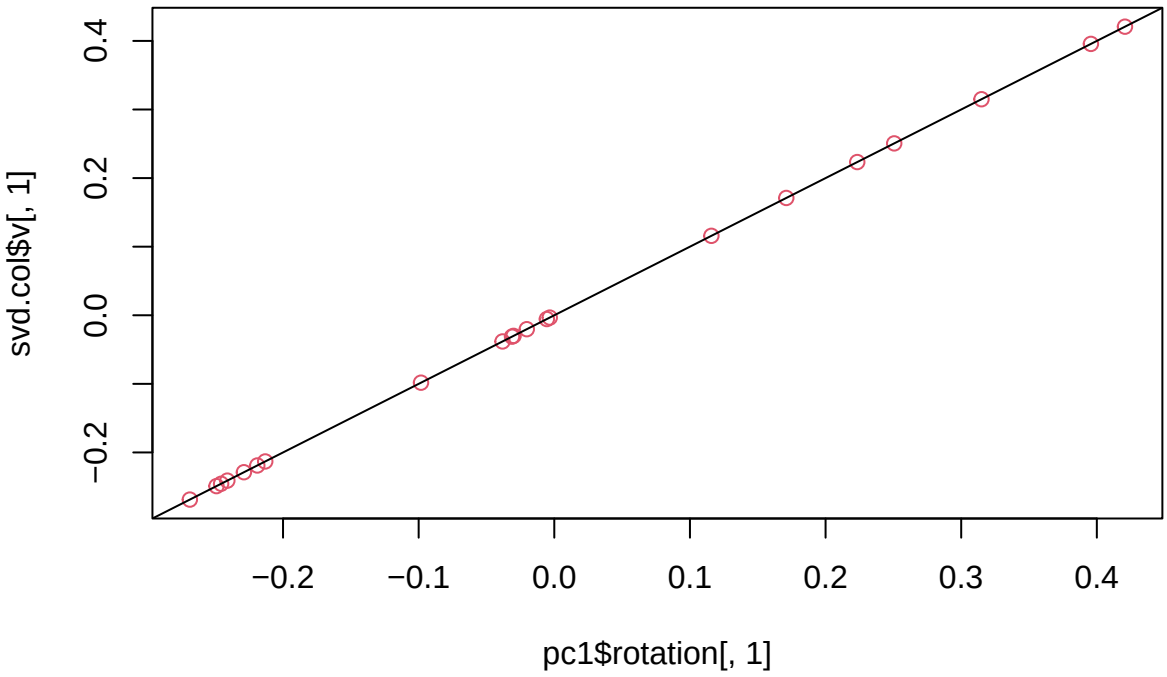
Role of Normalization

PCA has a long history in multivariate analysis. Combined with that of eigendecomposition, singular value decomposition, and related methods, there are confusing terminologies. If you are to use the PCA functions in R, you may get different results.

```
# compute PCs using prcomp and compare it with SVD
pc1 = prcomp(edata)
plot(pc1$rotation[,1],svd.out$v[,1])
```




```
# the results are different, because technically the data should be centered by column means
edata.col <- scale(edata, scale=FALSE, center=TRUE)
svd.col <- svd(edata.col)
plot(pc1$rotation[,1],svd.col$v[,1],col=2)
abline(0,1)
```




```
all(pc1$rotation[,1] == svd.col$v[,1])
```

```
## [1] TRUE
```

However, in genomics and modern high-dimensional data analysis, it's common to perform row-wise centering (and even scaling). Then, SVD is applied and the right singular vectors are often shown as PCs.

Apply truncated SVD approximation

When the data is very large, SVD becomes a computational bottleneck. In fact, in a personal computer, it may not work at all. Since we know that we may be only interested in `r` PCs (or singular vectors), we could use an approximation called truncated SVD/PCA using a package `irlba`:

> The augmented implicitly restarted Lanczos bidiagonalization algorithm (IRLBA) finds a few approximate largest (or, optionally, smallest) singular values and corresponding singular vectors of a sparse or dense matrix using a method of Baglama and Reichel. It is a fast and memory-efficient way to compute a partial SVD.

```
library(irlba)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
tsvd.out <- irlba(edata, nv = 4)
dim(tsvd.out$u)
```

```
## [1] 8544    4
```

```
length(tsvd.out$d)
```

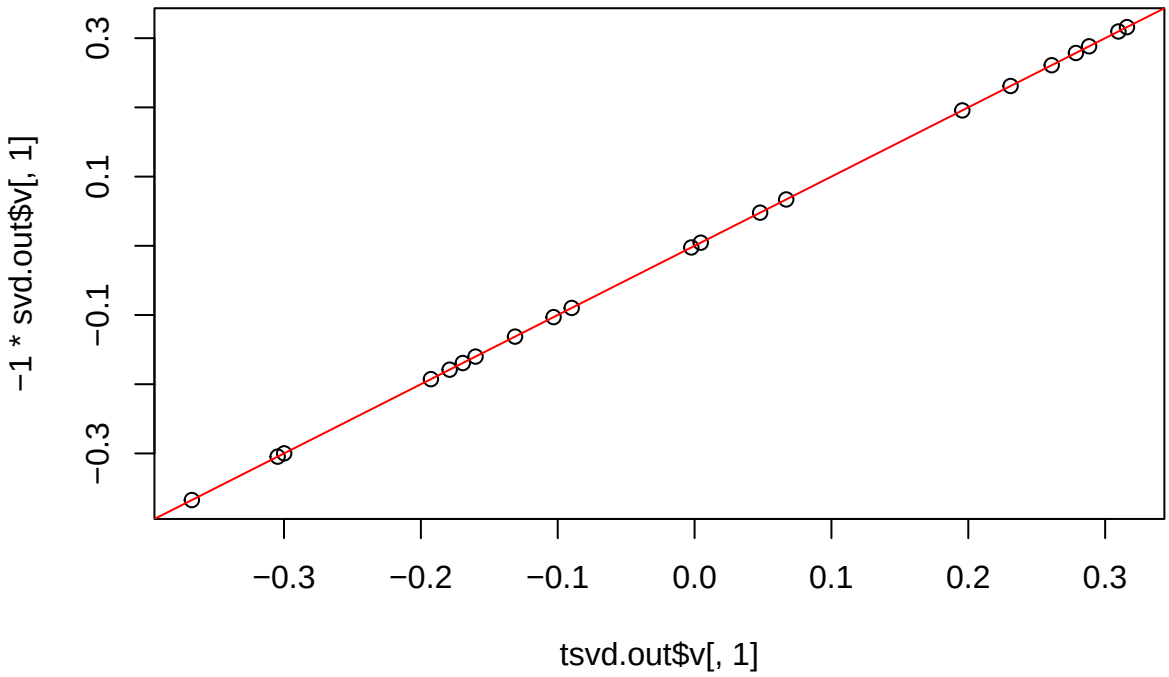
```
## [1] 4
```

```
dim(tsvd.out$v)
```

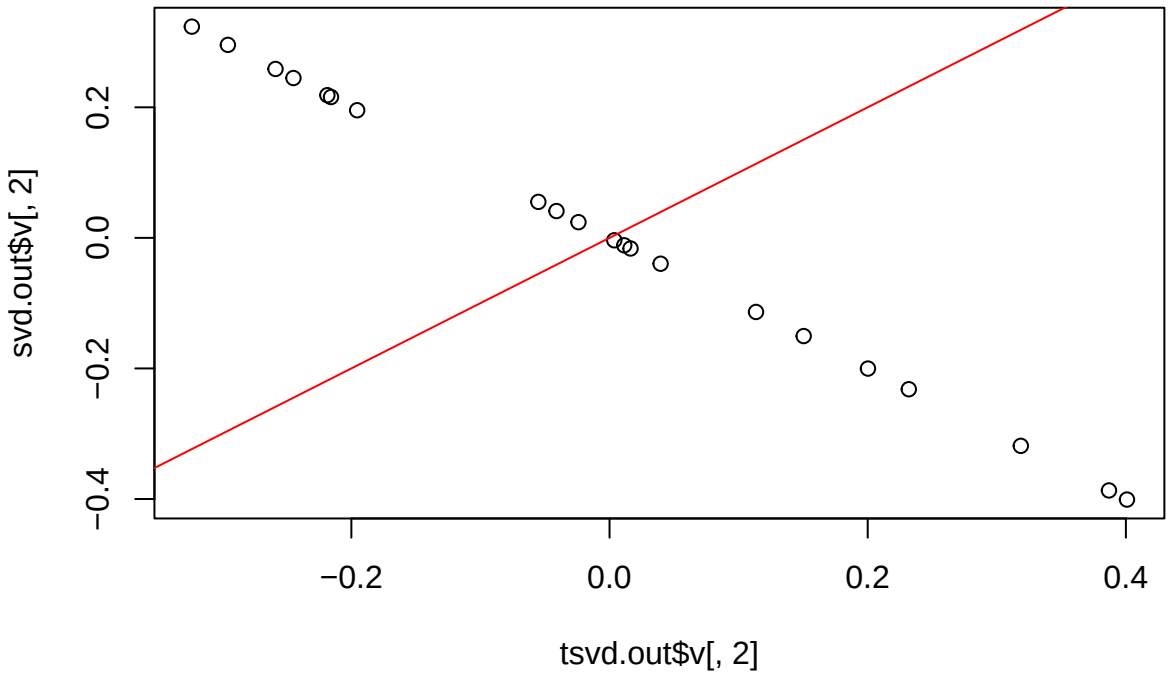
```
## [1] 21  4
```

Compare their approximate singular values. Note that when the data is not that big and the dominant signals, these algorithms produce very good approximations. Note that the signs of PCs are arbitrary and such rotation is not unique. So if you want, you can multiply a PC by -1:

```
plot(tsvd.out$v[,1],-1*svd.out$v[,1]); abline(0,1,col="red")
```




```
plot(tsvd.out$v[,2],svd.out$v[,2]); abline(0,1,col="red")
```



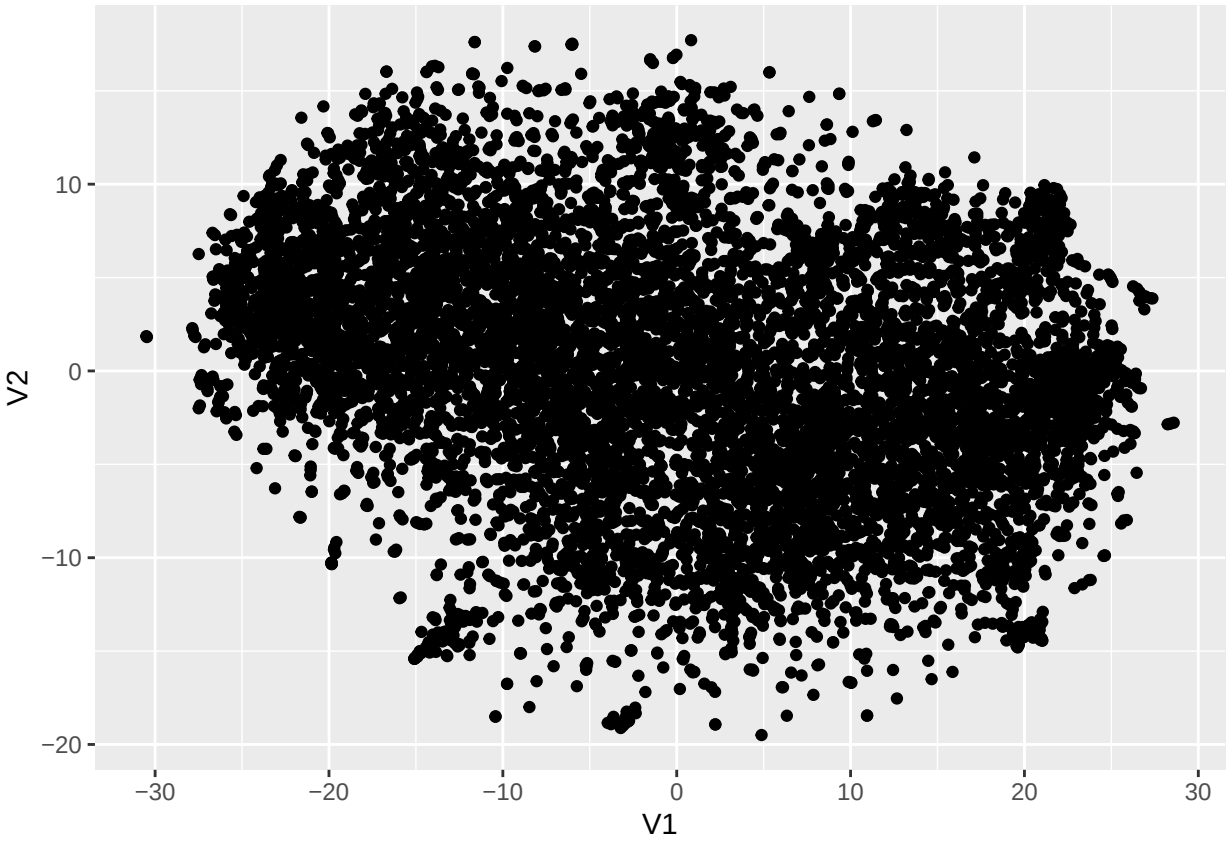
t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-Distributed Stochastic Neighbor Embedding (t-SNE) is an algorithm for dimension reduction and visualization. It's especially popular in machine learning and often uses PCA as a pre-processing step.

In this example, we apply t-SNE among the genes. Each point in a scatter plot then correspond to a gene:

```
library(irlba)
library(Rtsne)

# Set a seed for reproducible results
set.seed(1)
# complexity is a hyperparameter needed for this algorithm. 30 is a default
tsne_out <- Rtsne(edata,pca=FALSE,perplexity=60)
tsne_out = data.table(tsne_out$Y)
ggplot(tsne_out) + geom_point(aes(x=V1, y=V2))
```




```
tsne_pdata = data.table(tsne_out, pData(bottomly.eset))
```

```
## Warning: Item 2 has 21 rows but longest item has 8544; recycled with remainder.
```

Unlike SVD/PCA, t-SNE returns (slightly so) different results everytime it runs on the same dataset, which is why we declared a seed for a random number generator. Nonetheless, t-SNE may provide interesting low-dimensional projection that might be better than PCA.

> *Homework Problem 5:* Cluster the genes (rows) using K-means clustering (function `kmeans()`) on the original data, with `k=5` clusters. Then, create a 2-dimensional t-SNE projection (as done previously) while using the 5 clusters to color the data points corresponding to genes.

```r
#Clustering
clustered = kmeans(edata, centers=5)
tsne_out$cluster = clustered$cluster

#Plot
pdf("Bielecki_problem5.pdf")
ggplot(tsne_out) + geom_point(aes(x=V1, y=V2, color=cluster))
dev.off()
```

```
## pdf
##   2
```

Please make sure your 5 figures (not more, not less) are titled sequentially as "yourlastname_problem1.pdf", "yourlastname_problem2.pdf", "yourlastname_problem3.pdf", "yourlastname_problem4.pdf", "yourlastname_problem5.pdf".

Submit your homeworks via Github repositories.