MDS and PCoA

Sebastian Heucke

10/1/2019

Contents

MDS and PCoA 1
Session information 8

MDS and PCoA

This tutorial shows how to do MDS (multi-dimensional scaling) and PCoA (principal coordinate analysis) by converting this video https://www.youtube.com/watch?v=pGAUHhLYp5Q&list=PLblh5JKOoLUJJpBNfk8_YadPwDTO2SCbx&index=3&t=0s into a markdown document.

First we load in ggplot2 since we'll need it later to draw fancy looking graphs. Then we generate some fake data. The data will consist of a matrix with 10 columns, corresponding to 10 samples, and 100 rows, corresponding to measurements from 100 genes. The first 5 columns will be "wt" (or wild-type) samples and the last 5 columns will be "ko" (knock-out) samples. the genes will have names like "gene1" and "gene2".

```
library(ggplot2)

# In this example, the data is in a matrix called
# data.matrix
# columns are individual samples (i.e. cells)
# rows are measurements taken for all the samples (i.e. genes)
# Just for the sake of the example, here's some made up data...
data.matrix <- matrix(nrow=100, ncol=10)
colnames(data.matrix) <- c(
  paste("wt", 1:5, sep=""),
  paste("ko", 1:5, sep=""))
rownames(data.matrix) <- paste("gene", 1:100, sep="")
for (i in 1:100) {
  wt.values <- rpois(5, lambda=sample(x=10:1000, size=1))
  ko.values <- rpois(5, lambda=sample(x=10:1000, size=1))

  data.matrix[i,] <- c(wt.values, ko.values)
}
head(data.matrix)
```

```
##       wt1  wt2  wt3  wt4  wt5 ko1 ko2 ko3 ko4 ko5
## gene1 975  955  933  952  974  41  47  41  49  49
## gene2 226  215  180  192  209 824 867 797 807 818
## gene3 897  802  857  868  840 534 568 527 534 516
```

```
## gene4 974 1020 1010 1000 1008 188 170 172 163 171
## gene5  34   25   23   31   23 735 766 751 759 751
## gene6 717  686  714  714  736 241 229 260 241 258
```

```
dim(data.matrix)
```

```
## [1] 100  10
```

Now, just for comparsion, we'll do PCA on the dataset.

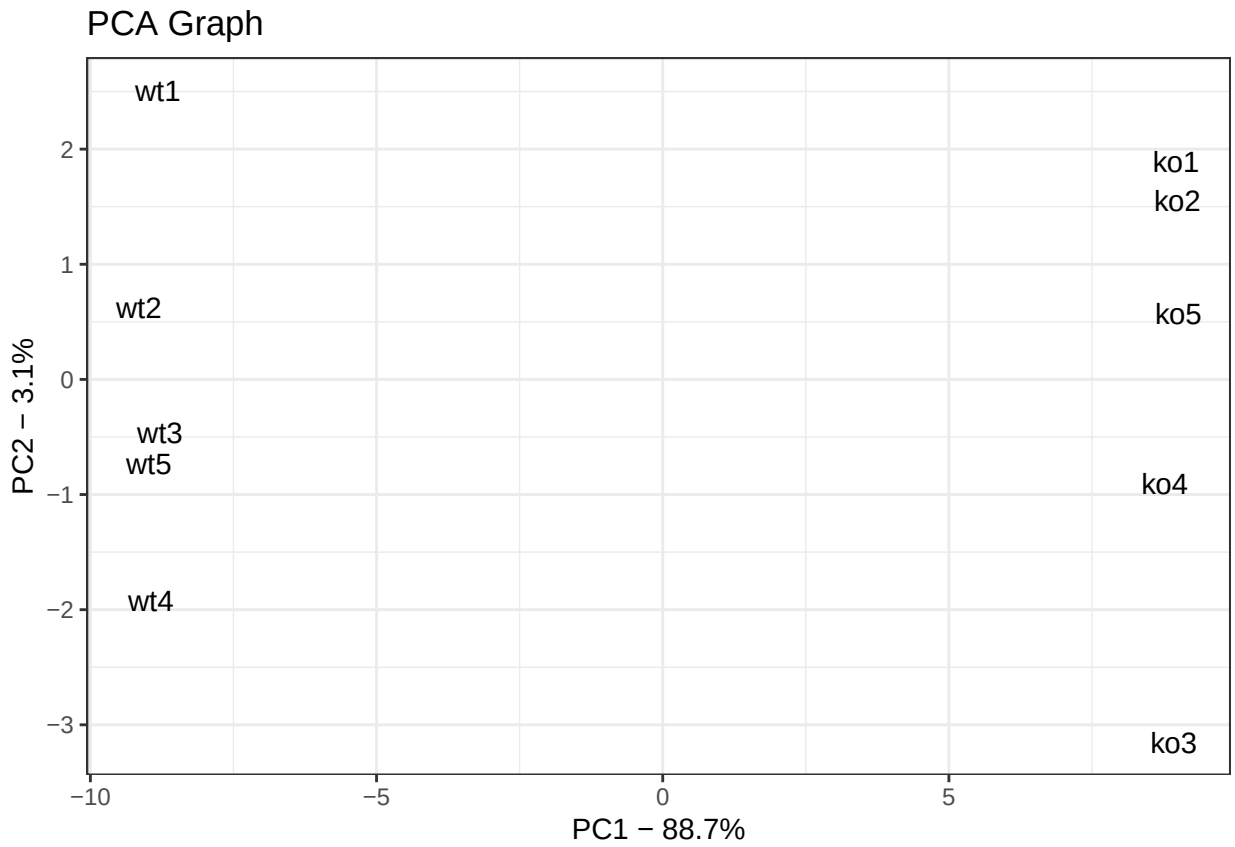
```
pca <- prcomp(t(data.matrix), scale=TRUE, center=TRUE)

## calculate the percentage of variation that each PC accounts for...
pca.var <- pca$sdev^2
pca.var.per <- round(pca.var/sum(pca.var)*100, 1)
pca.var.per
```

```
##  [1] 88.7  3.1  2.3  1.5  1.3  1.0  0.9  0.6  0.5  0.0
```

```
## now make a fancy looking plot that shows the PCs and the variation:
pca.data <- data.frame(Sample=rownames(pca$x),
  X=pca$x[,1],
  Y=pca$x[,2])
pca.data
```

```
##     Sample         X          Y
## wt1    wt1 -8.821724  2.5104874
## wt2    wt2 -9.152043  0.6221173
## wt3    wt3 -8.783428 -0.4577214
## wt4    wt4 -8.942747 -1.9158862
## wt5    wt5 -8.978561 -0.7314830
## ko1    ko1  8.971707  1.8962349
## ko2    ko2  8.994287  1.5545237
## ko3    ko3  8.931241 -3.1517653
## ko4    ko4  8.771129 -0.9033742
## ko5    ko5  9.010138  0.5768668
```

```
ggplot(data=pca.data, aes(x=X, y=Y, label=Sample)) +
  geom_text() +
  xlab(paste("PC1 - ", pca.var.per[1], "%", sep="")) +
  ylab(paste("PC2 - ", pca.var.per[2], "%", sep="")) +
  theme_bw() +
  ggtitle("PCA Graph")
```



Wild-type samples are on the left and the knock-out samples are on the right side. The x-axis, for PC1, accounts for 88% of the variation in the data and the y-axis, for PC2, only accounts for 2.7% of the variation in the data. This means that most of the differences are between the WT and KO samples.

Now draw an MDS plot using the same data and the Euclidean distance. This graph should look the same as the PCA plot.

1. Create a distance matrix. We do this with the **dist()** function. We need to transpose the data, so that samples are rows. We also center and scale the measurements for each gene, which are now columns. Note the **dist()** function has 6 different method to choose from.
2. Perform multi-dimensional scaling on the distance matrix using the **cmdscale()** (**C**lassical **M**ulti-**D**imensional **scaling**) function. We tell **cmdscale()** that we want it to return the eigen values, these will be used to calculate how much variation in the distance matrix each axis in the final MDS plot accounts for. We can also get cmdscale() to return the doubly centered (both rows and columns are centered) version of the distance matrix, this is useful if you want to demonstrate how to do MDS using the eigen() function instead of the cmdscale() function.
3. Calculate the amount of variation each axis in the MDS plot accounts for using the eigen values.
4. Format the data for ggplot and call **ggplot()** to make a graph.

```
## first, calculate the distance matrix using the Euclidian distance.
## NOTE: We are transposing, scaling and centering the data just like PCA.
distance.matrix <- dist(scale(t(data.matrix), center=TRUE, scale=TRUE),
  method="euclidean")

## do the MDS math (this is basically eigen value decomposition)
mds.stuff <- cmdscale(distance.matrix, eig=TRUE, x.ret=TRUE)
```

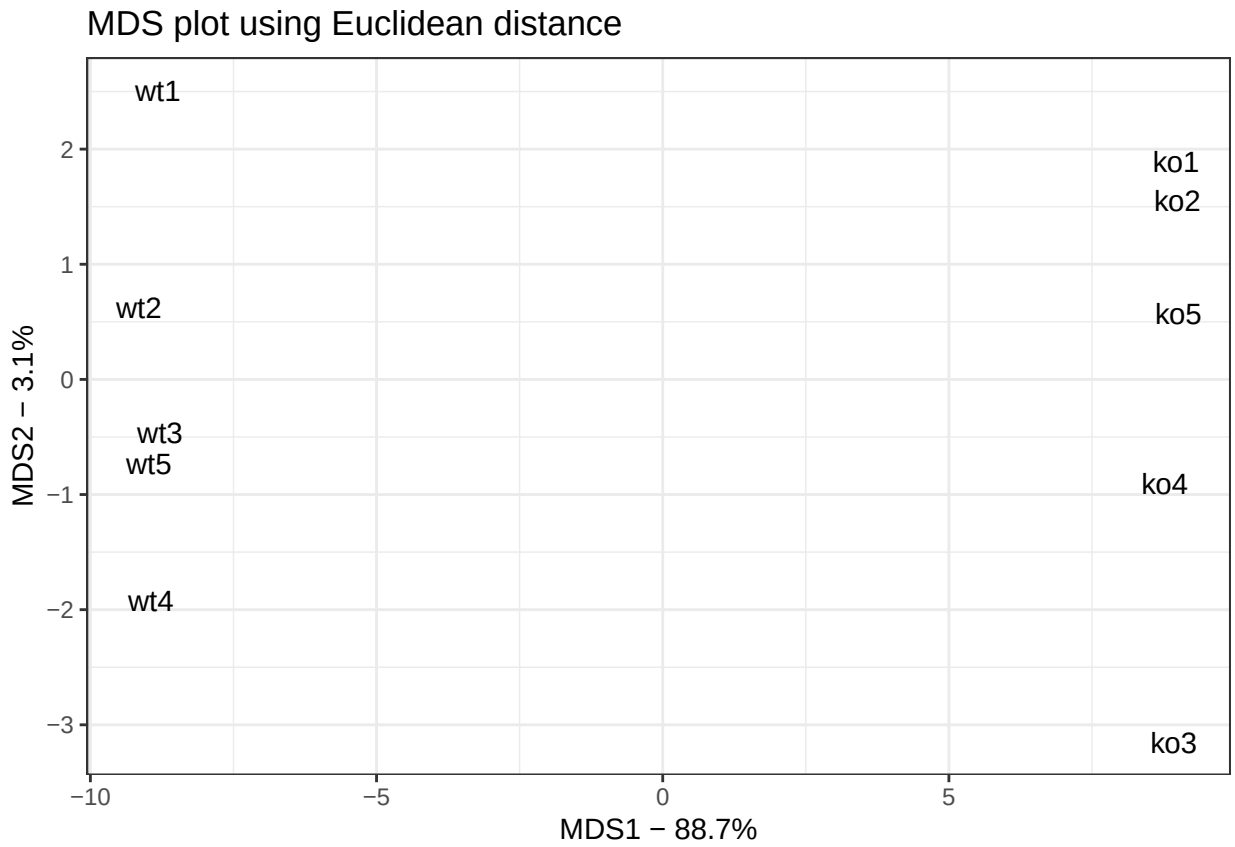
```
## calculate the percentage of variation that each MDS axis accounts for...
mds.var.per <- round(mds.stuff$eig/sum(mds.stuff$eig)*100, 1)
mds.var.per
```

```
##  [1] 88.7  3.1  2.3  1.5  1.3  1.0  0.9  0.6  0.5  0.0
```

```
## now make a fancy looking plot that shows the MDS axes and the variation:
mds.values <- mds.stuff$points
mds.data <- data.frame(Sample=rownames(mds.values),
  X=mds.values[,1],
  Y=mds.values[,2])
mds.data
```

```
##     Sample         X          Y
## wt1    wt1 -8.821724  2.5104874
## wt2    wt2 -9.152043  0.6221173
## wt3    wt3 -8.783428 -0.4577214
## wt4    wt4 -8.942747 -1.9158862
## wt5    wt5 -8.978561 -0.7314830
## ko1    ko1  8.971707  1.8962349
## ko2    ko2  8.994287  1.5545237
## ko3    ko3  8.931241 -3.1517653
## ko4    ko4  8.771129 -0.9033742
## ko5    ko5  9.010138  0.5768668
```

```
ggplot(data=mds.data, aes(x=X, y=Y, label=Sample)) +
  geom_text() +
  theme_bw() +
  xlab(paste("MDS1 - ", mds.var.per[1], "%", sep="")) +
  ylab(paste("MDS2 - ", mds.var.per[2], "%", sep="")) +
  ggtitle("MDS plot using Euclidean distance")
```



Just like in the PCA grpah, the wild-type samples are on the left side of the graph and the knock-out samples are on the right side. And just like in the PCA graph, the x-axis accounts for 88% of the variation in the data. Actually, the PCA and the MDS graph don't just look similar, they are exactly the same. This is because we used the Euclidean metric to calculate the distance matrix.

Now let's see what happens when we use a different matric to calculate the distance matrix. Let's use the average of the absolute value of the log fold change. This is what **edgeR** does when you call the plotMDS() function.

The first thing to do is calculate the log2 values of the measurements for each gene. Since the average of absolute values of the log-fold change isn't one of the distance metrics built into the dist() function, we'll create our own distance matrix. After that we perform multi-dimensional scaling on our new distance matrix.

```
# first, take the log2 of all the values in the data.matrix.
# This makes it easy to compute log2(Fold Change) between a gene in two
# samples since...
#
# log2(Fold Change) = log2(value for sample 1) - log2(value for sample 2)
#
log2.data.matrix <- log2(data.matrix)

# now create an empty distance matrix
log2.distance.matrix <- matrix(0,
  nrow=ncol(log2.data.matrix),
  ncol=ncol(log2.data.matrix),
  dimnames=list(colnames(log2.data.matrix),
    colnames(log2.data.matrix)))
```

```
log2.distance.matrix
```

```
##     wt1 wt2 wt3 wt4 wt5 ko1 ko2 ko3 ko4 ko5
## wt1   0   0   0   0   0   0   0   0   0   0
## wt2   0   0   0   0   0   0   0   0   0   0
## wt3   0   0   0   0   0   0   0   0   0   0
## wt4   0   0   0   0   0   0   0   0   0   0
## wt5   0   0   0   0   0   0   0   0   0   0
## ko1   0   0   0   0   0   0   0   0   0   0
## ko2   0   0   0   0   0   0   0   0   0   0
## ko3   0   0   0   0   0   0   0   0   0   0
## ko4   0   0   0   0   0   0   0   0   0   0
## ko5   0   0   0   0   0   0   0   0   0   0
```

```
# now compute the distance matrix using avg(absolute value(log2(FC)))
for(i in 1:ncol(log2.distance.matrix)) {
  for(j in 1:i) {
    log2.distance.matrix[i, j] <-
      mean(abs(log2.data.matrix[,i] - log2.data.matrix[,j]))
  }
}
log2.distance.matrix
```

```
##            wt1        wt2        wt3        wt4      wt5        ko1
## wt1 0.00000000 0.00000000 0.00000000 0.00000000 0.000000 0.00000000
## wt2 0.10095194 0.00000000 0.00000000 0.00000000 0.000000 0.00000000
## wt3 0.10210635 0.09136025 0.00000000 0.00000000 0.000000 0.00000000
## wt4 0.10172682 0.09870606 0.09402319 0.00000000 0.000000 0.00000000
## wt5 0.09464479 0.09608481 0.09407219 0.09693253 0.000000 0.00000000
## ko1 1.28171492 1.30622760 1.30326522 1.30596349 1.292322 0.00000000
## ko2 1.27803864 1.29632586 1.29295554 1.29538721 1.280828 0.09928254
## ko3 1.28920918 1.30270350 1.30148239 1.30041921 1.286099 0.09890886
## ko4 1.28991286 1.30851060 1.30472606 1.30592702 1.292500 0.09460740
## ko5 1.27937269 1.30197480 1.29871242 1.30020167 1.286931 0.08972707
##            ko2        ko3        ko4 ko5
## wt1 0.00000000 0.00000000 0.00000000   0
## wt2 0.00000000 0.00000000 0.00000000   0
## wt3 0.00000000 0.00000000 0.00000000   0
## wt4 0.00000000 0.00000000 0.00000000   0
## wt5 0.00000000 0.00000000 0.00000000   0
## ko1 0.00000000 0.00000000 0.00000000   0
## ko2 0.00000000 0.00000000 0.00000000   0
## ko3 0.10054073 0.00000000 0.00000000   0
## ko4 0.08600566 0.08105311 0.00000000   0
## ko5 0.08522287 0.08296172 0.07591701   0
```

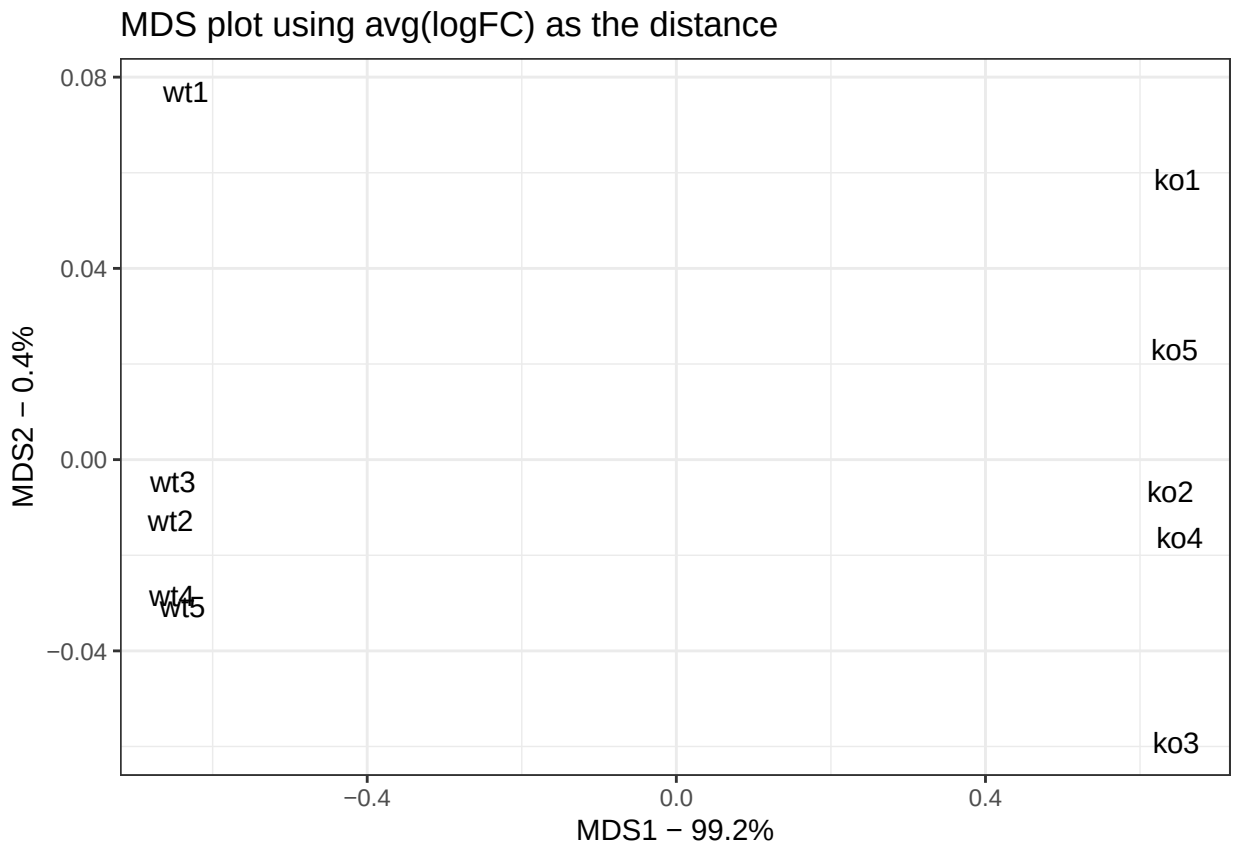
```
# do the MDS math (this is basically eigen value decomposition)
# cmdscale() is the function for "Classical Multi-Dimensional Scalign"
mds.stuff <- cmdscale(as.dist(log2.distance.matrix),
  eig=TRUE,
  x.ret=TRUE)

# calculate the percentage of variation that each MDS axis accounts for...
mds.var.per <- round(mds.stuff$eig/sum(mds.stuff$eig)*100, 1)
mds.var.per
```

```
##  [1] 99.2  0.4  0.1  0.1  0.1  0.1  0.1  0.1  0.0 -0.1
```

```
# now make a fancy looking plot that shows the MDS axes and the variation:
mds.values <- mds.stuff$points
mds.data <- data.frame(Sample=rownames(mds.values),
  X=mds.values[,1],
  Y=mds.values[,2])
mds.data
```

```
##     Sample          X            Y
## wt1    wt1 -0.6344991  0.077109011
## wt2    wt2 -0.6542201 -0.012665831
## wt3    wt3 -0.6513587 -0.004672367
## wt4    wt4 -0.6525721 -0.028315786
## wt5    wt5 -0.6388400 -0.030676183
## ko1    ko1  0.6486198  0.058554056
## ko2    ko2  0.6395361 -0.006722903
## ko3    ko3  0.6469000 -0.059253247
## ko4    ko4  0.6516300 -0.016371789
## ko5    ko5  0.6448041  0.023015040
```

```
ggplot(data=mds.data, aes(x=X, y=Y, label=Sample)) +
  geom_text() +
  theme_bw() +
  xlab(paste("MDS1 - ", mds.var.per[1], "%", sep="")) +
  ylab(paste("MDS2 - ", mds.var.per[2], "%", sep="")) +
  ggtitle("MDS plot using avg(logFC) as the distance")
```



The two different MDS plots (one using the Euclidean distance, the other using the average absolute value of the log fold change) are similar, but not the same. In the new graph, the x-axis accounts more of the variation 99%.

Session information

```
## R version 3.6.1 (2019-07-05)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 18.04.3 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.7.1
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.7.1
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] ggplot2_3.2.0  knitr_1.23     devtools_2.1.0 usethis_1.5.1
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.1        pillar_1.4.2      compiler_3.6.1
##  [4] prettyunits_1.0.2 remotes_2.1.0     tools_3.6.1
##  [7] testthat_2.1.1    digest_0.6.20     pkgbuild_1.0.5
## [10] pkgload_1.0.2     tibble_2.1.3      evaluate_0.14
## [13] memoise_1.1.0     gtable_0.3.0      pkgconfig_2.0.2
## [16] rlang_0.4.0       cli_1.1.0         yaml_2.2.0
## [19] xfun_0.8          dplyr_0.8.3       withr_2.1.2
## [22] stringr_1.4.0     desc_1.2.0        fs_1.3.1
## [25] tidyselect_0.2.5  rprojroot_1.3-2   grid_3.6.1
## [28] glue_1.3.1        R6_2.4.0          processx_3.4.1
## [31] rmarkdown_1.14    sessioninfo_1.1.1 purrr_0.3.2
## [34] callr_3.3.1       magrittr_1.5      backports_1.1.4
## [37] scales_1.0.0      ps_1.3.0          htmltools_0.3.6
## [40] assertthat_0.2.1  colorspace_1.4-1  labeling_0.3
## [43] stringi_1.4.3     lazyeval_0.2.2    munsell_0.5.0
## [46] crayon_1.3.4
```

The document was processed on 2019-10-01.