Lecture 4: High Quality Graphics in R

Jing Ma, Statistics, TAMU

16 September, 2019

Recap

- Statistical modeling
- Maximum likelihood estimation
- Markov chains for dependency

This lecture

- Create beautiful and intuitive plots for scientific presentations and publications
- **ggplot2**
- Discuss some options of interactive graphics

R packages we will use

```
# Bioconductor
library("GenomicRanges")
library("Hiiragi2013")
library("ggbio")

# CRAN
library("dplyr")
library("ggplot2")
library("reshape2")
library("Hmisc")
library("ggbeeswarm")
library("extrafont")
library("pheatmap")
library("magrittr")
library("RColorBrewer")
library("hexbin")
```

Base R plotting

The most basic function is `plot`.

We use an example data from R.

```
head(DNase)
```

```
##   Run       conc density
## 1   1 0.04882812   0.017
## 2   1 0.04882812   0.018
## 3   1 0.19531250   0.121
## 4   1 0.19531250   0.124
## 5   1 0.39062500   0.206
## 6   1 0.39062500   0.215
```

Base R plotting

```
plot(DNase$conc, DNase$density)
```

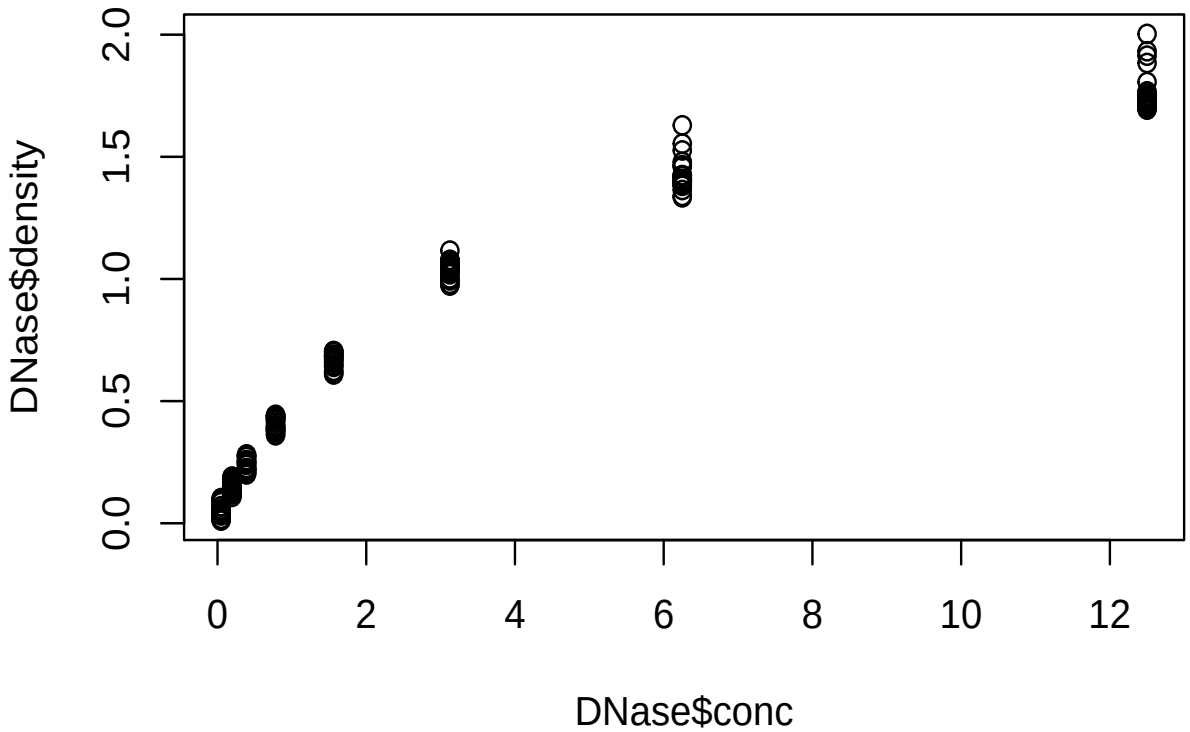


Fig. 1: Plot of concentration vs. density for an ELISA assay of DNase.

Base R plotting can be customized

We can redefine the axis labels, the point shapes, point colors, etc.

```
plot(DNase$conc, DNase$density,
     ylab = attr(DNase, "labels")$y,
     xlab = paste(attr(DNase, "labels")$x, attr(DNase, "units")$x),
     pch = 3,
     col = "blue")
```

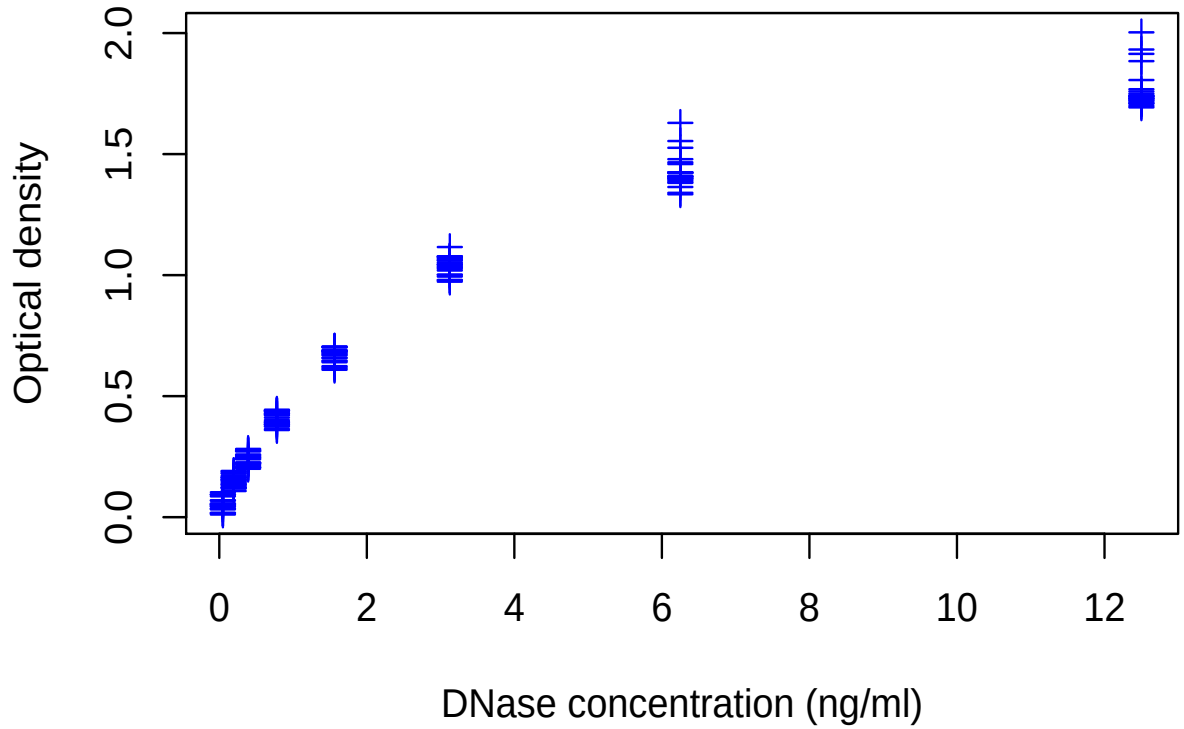


Fig. 2: Same data as in Fig. 1 but with better axis labels and a different plot symbol.

Histograms

```
hist(DNase$density, breaks=25, main = "")
```

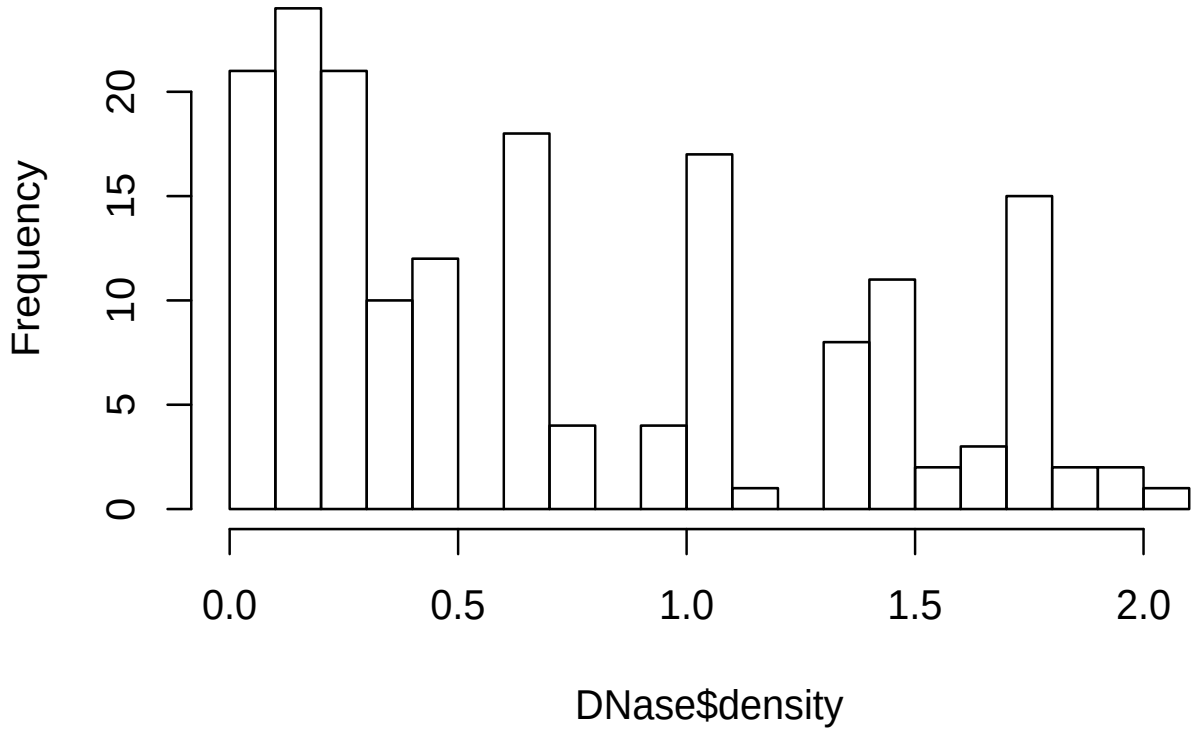


Fig. 3: Histogram of the density from the ELISA assay.

Boxplots

```
boxplot(density ~ Run, data = DNase)
```

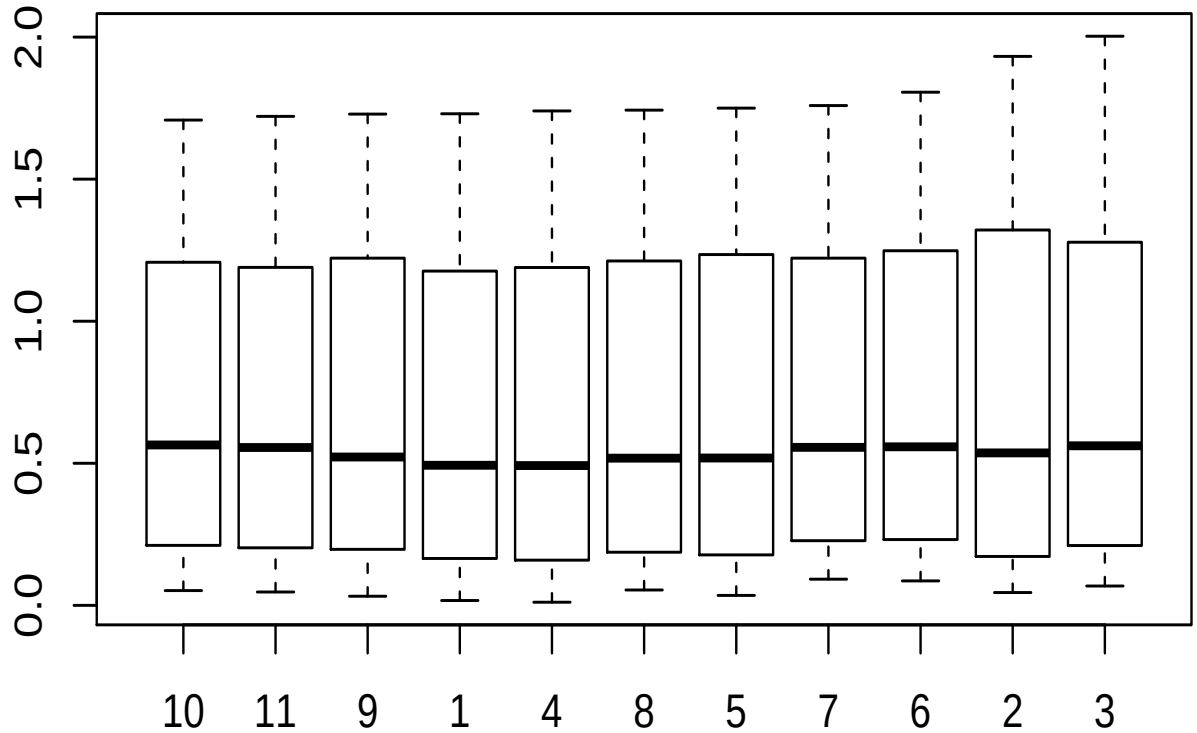


Fig. 4: Boxplots of the density from the ELISA assay stratified by the assay run.

An example dataset

Our example dataset is from the Bioconductor package **Hiiragi2013**.

```
library("Hiiragi2013")
data("x")
dim(Biobase::exprs(x))
```

```
## [1] 45101   101
```

Let's have a look at what information is available about the samples.

```
head(pData(x), n = 2)
```

```
##         File.name Embryonic.day Total.number.of.cells lineage genotype
## 1 E3.25  1_C32_IN         E3.25                    32               WT
## 2 E3.25  2_C32_IN         E3.25                    32               WT
##           ScanDate sampleGroup sampleColour
## 1 E3.25 2011-03-16       E3.25      #CAB2D6
## 2 E3.25 2011-03-16       E3.25      #CAB2D6
```

An example dataset

Using the following code (we'll explain shortly), we define a small dataframe `groups` that contains summary information for each group: the number of cells and the preferred color.

```
library("dplyr")
groups = group_by(pData(x), sampleGroup) %>%
  summarise(n = n(), color = unique(sampleColour))
groups
```

```
## # A tibble: 8 x 3
##   sampleGroup          n color
##   <chr>            <int> <chr>
## 1 E3.25               36 #CAB2D6
## 2 E3.25 (FGF4-KO)     17 #FDBF6F
## 3 E3.5 (EPI)          11 #A6CEE3
## 4 E3.5 (FGF4-KO)       8 #FF7F00
## 5 E3.5 (PE)           11 #B2DF8A
## 6 E4.5 (EPI)           4 #1F78B4
## 7 E4.5 (FGF4-KO)      10 #E31A1C
## 8 E4.5 (PE)            4 #33A02C
```

R code: %>%

The pipe %>% is useful for making nested function calls easier to read for humans. For example, the following two lines of code are equivalent in R.

```
f(x) %>% g(y) %>% h
h(g(f(x), y))
```

What it means: Evaluate `f(x)`, then pass the result to function `g` as the first argument, while `y` is passed to `g` as the second argument. Then pass the output of `g` to the function `h`.

R code: group_by

```
groups = group_by(pData(x), sampleGroup) %>%
  summarise(n = n(), color = unique(sampleColour))
```

The group_by function simply "marks" the dataframe with a note that all subsequent operations should not be applied to the whole dataframe at once, but to blocks defined by the sampleGroup factor.

R code: `summarise`

Finally, `summarise` computes summary statistics; this could be, e.g., the `mean`, `sum`; in this case, we just compute the number of rows in each block, `n()`, and the prevalent color.

ggplot2

- A package by Hadley Wickham (Wickham 2016) that implements the idea of grammar of graphics.
- Free online tutorials are available.
- If you have not installed **ggplot2**, use the following command to install the package:

ggplot2 makes plotting easy!

We can easily reproduce our previous figure that used base plotting functions.

```
library("ggplot2")
ggplot(DNase, aes(x = conc, y = density)) + geom_point()
```

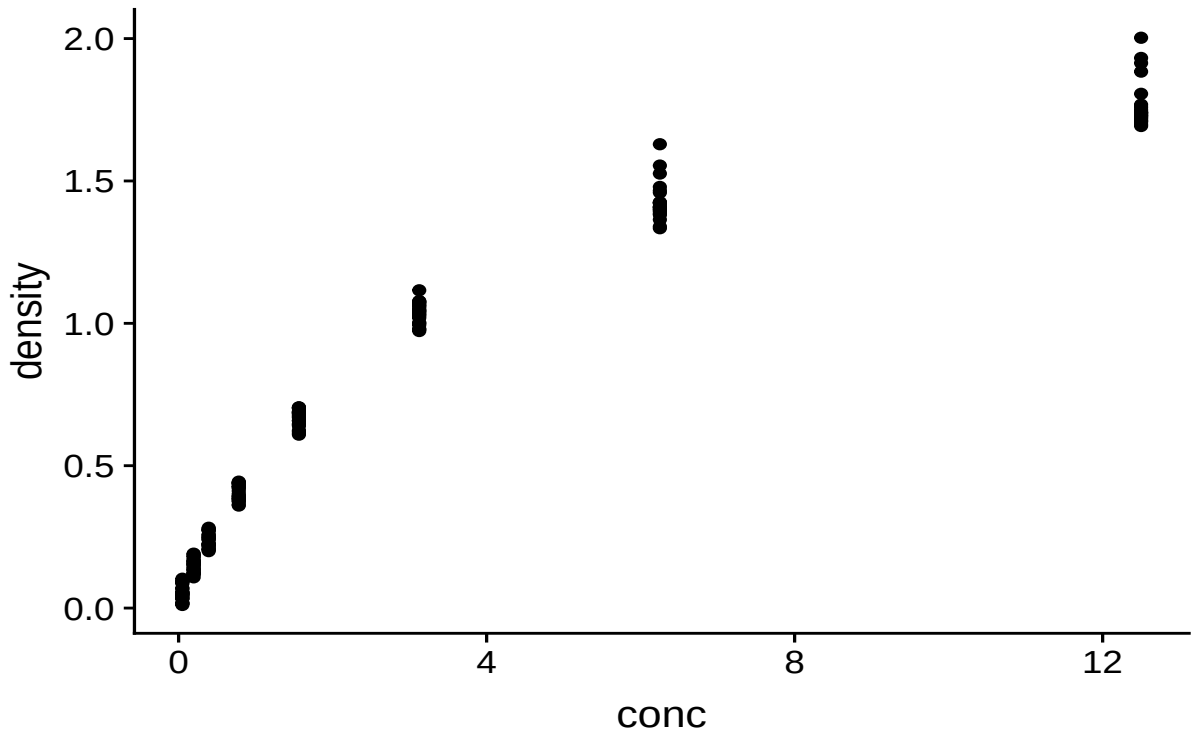


Fig. 5: Our first ggplot2 figure, similar to the base graphics Figure 2.

ggplot2 makes plotting easy!

We can also plot the number of samples for each of the 8 groups in the mouse single cell data.

```
ggplot(groups, aes(x = sampleGroup, y = n)) +
  geom_bar(stat = "identity")
```

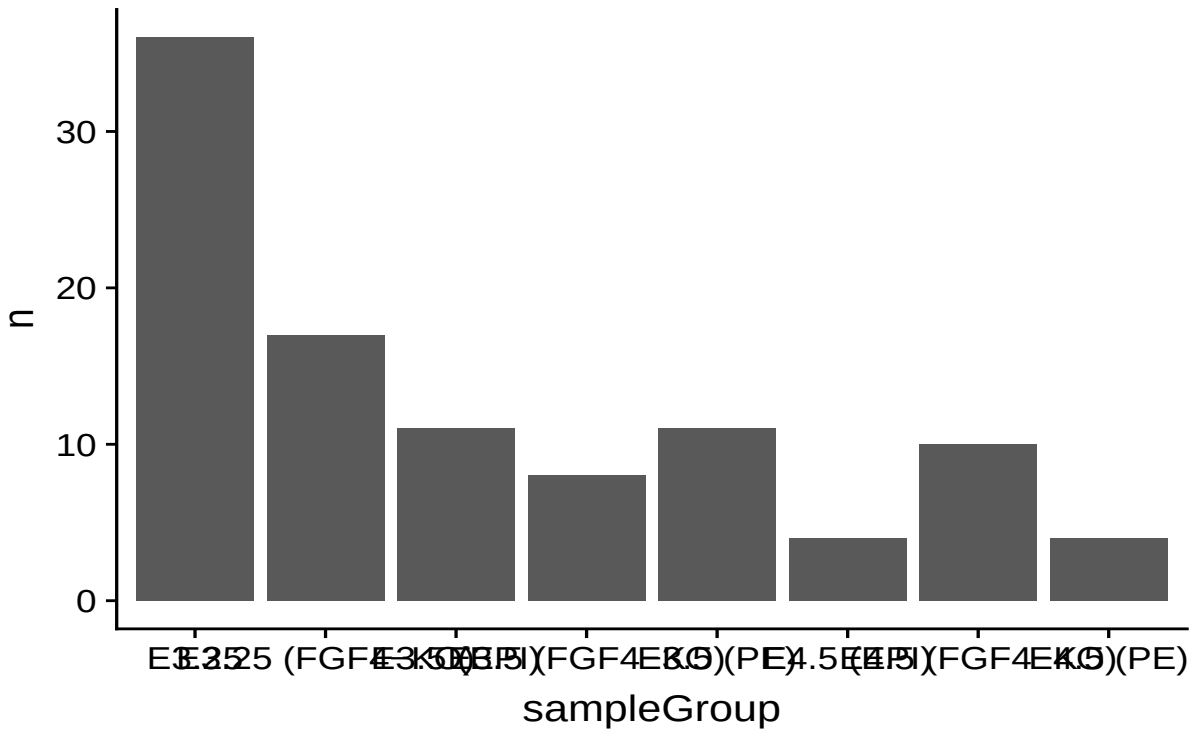


Fig. 6: A barplot, produced with the ggplot function from the table of group sizes in the mouse single cell data.

Adjusting the style

We can use color for the bars to help us quickly see which bar corresponds to which group.

- Need to define our colors.
- Want the bar labels more readable.

```
groupColor = setNames(groups$color, groups$sampleGroup)
ggplot(groups, aes(x = sampleGroup, y = n, fill = sampleGroup)) +
  geom_bar(stat = "identity") +
  scale_fill_manual(values = groupColor, name = "Groups") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

Adjusting the style

```
groupColor = setNames(groups$color, groups$sampleGroup)
ggplot(groups, aes(x = sampleGroup, y = n, fill = sampleGroup)) +
  geom_bar(stat = "identity") +
  scale_fill_manual(values = groupColor, name = "Groups") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

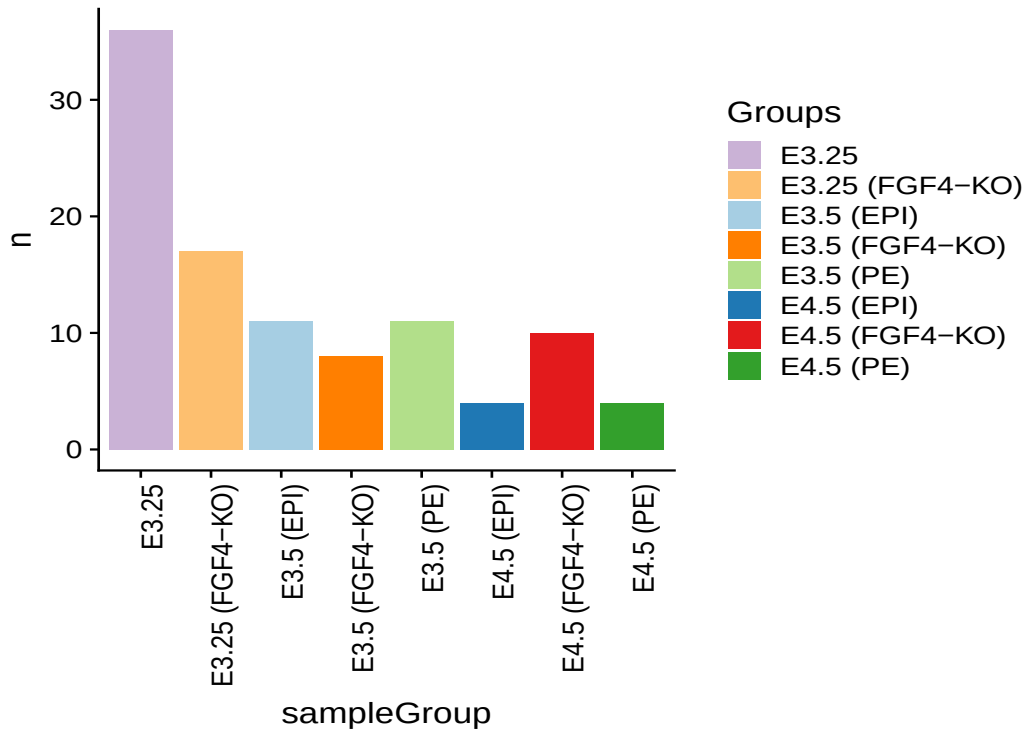


Fig. 7: Similar to Fig. 6, but with colored bars and better bar labels.

Saving figures

The prefered format in R for saving plots into a vector graphics format is PDF.

```
gg = ggplot(DNase, aes(x = conc, y = density)) + geom_point()
ggsave("DNAse-histogram-demo.pdf", plot = gg, width=6,height=4)
```

Grammar of graphics

The components of **ggplot2**'s grammar of graphics are

1. One or more datasets
2. One or more geometric objects, for instance, points, lines, rectangles, contours,
3. Descriptions of how the variables in the data are mapped to visual properties (aesthetics) of the geometric objects, and an associated scale (e.g., linear, logarithmic, rank)
4. One or more coordinate systems,
5. Statistical summarization rules,
6. A facet specification,
7. Optional parameters for layout and rendering, e.g. text size, alignment, legend positions.

The first three items are required.

An example

```
ggplot(groups, aes(x = sampleGroup, y = n, fill = sampleGroup)) +
  geom_bar(stat = "identity")
```

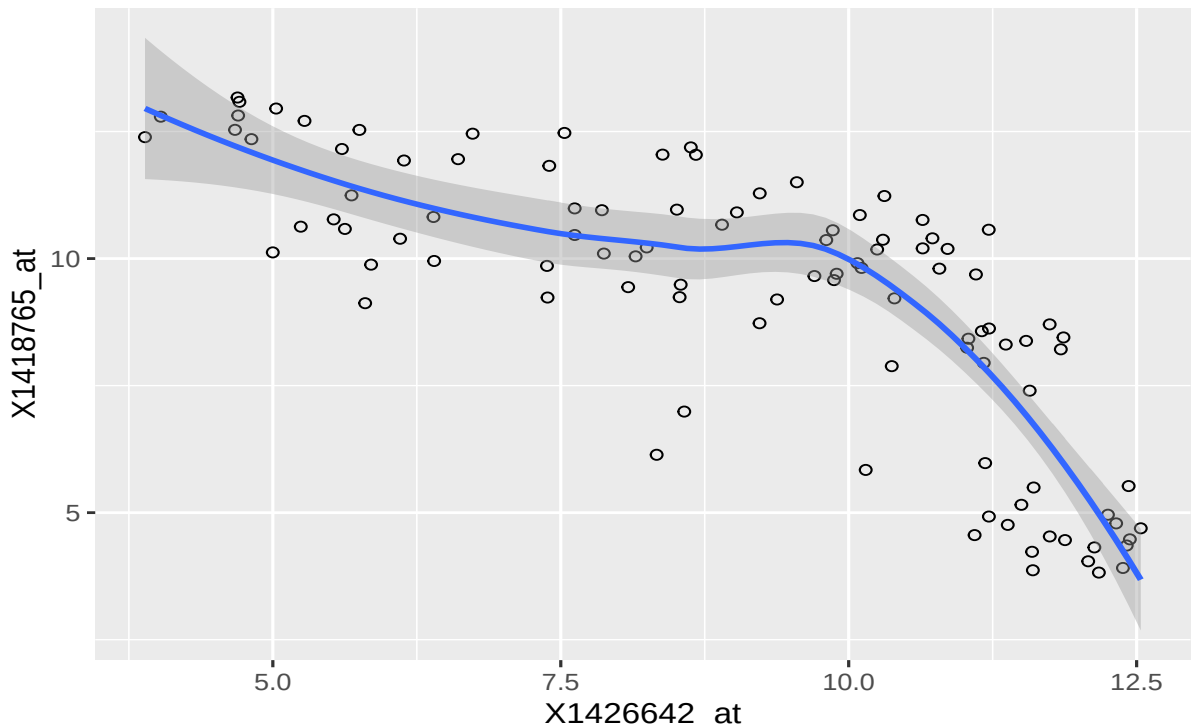
1. The dataset was `groups`.
2. The geometric object was the rectangular bar.
3. The variables were the numeric values (`n`) and the names of groups, which we mapped to the aesthetics y-axis and x-axis respectively. The scale was linear on the y and rank-based on the x-axis (bars are ordered alphanumerically and each has the same width).

Question: can we use multiple geometric objects?

Multiple geometric objects

The code below uses three types of geometric objects: points, a line and a confidence band.

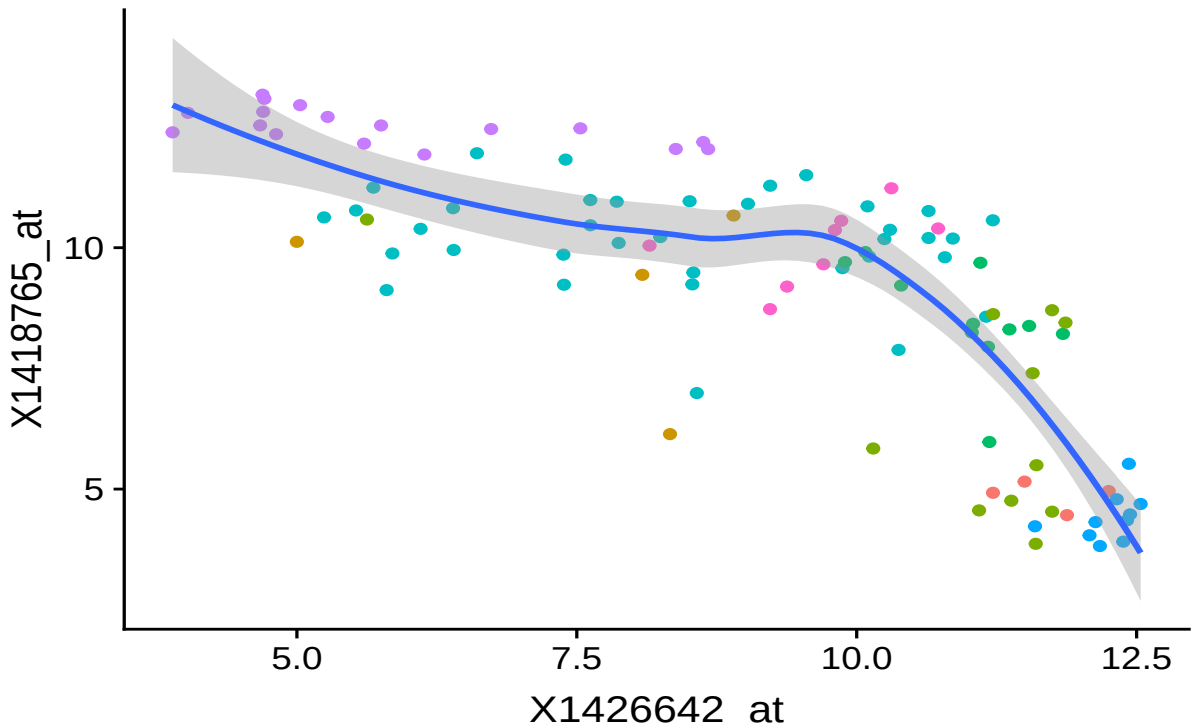
```
dftx = data.frame(t(Biobase::exprs(x)), pData(x))
ggplot( dftx, aes( x = X1426642_at, y = X1418765_at)) +
  geom_point( shape = 1 ) +
  geom_smooth( method = "loess" )
```



Multiple geometric objects

We can further enhance the plot by using colors.

```
ggplot( dftx, aes( x = X1426642_at, y = X1418765_at ))  +
  geom_point( aes( color = sampleColour), shape = 19 ) +
  geom_smooth( method = "loess" ) +
  scale_color_discrete( guide = FALSE )
```



Stepwise ggplot

We can start with an empty plot and build up the ggplot step by step.

```
pb = ggplot(groups, aes(x = sampleGroup, y = n))
class(pb)
```

```
## [1] "gg"     "ggplot"
```

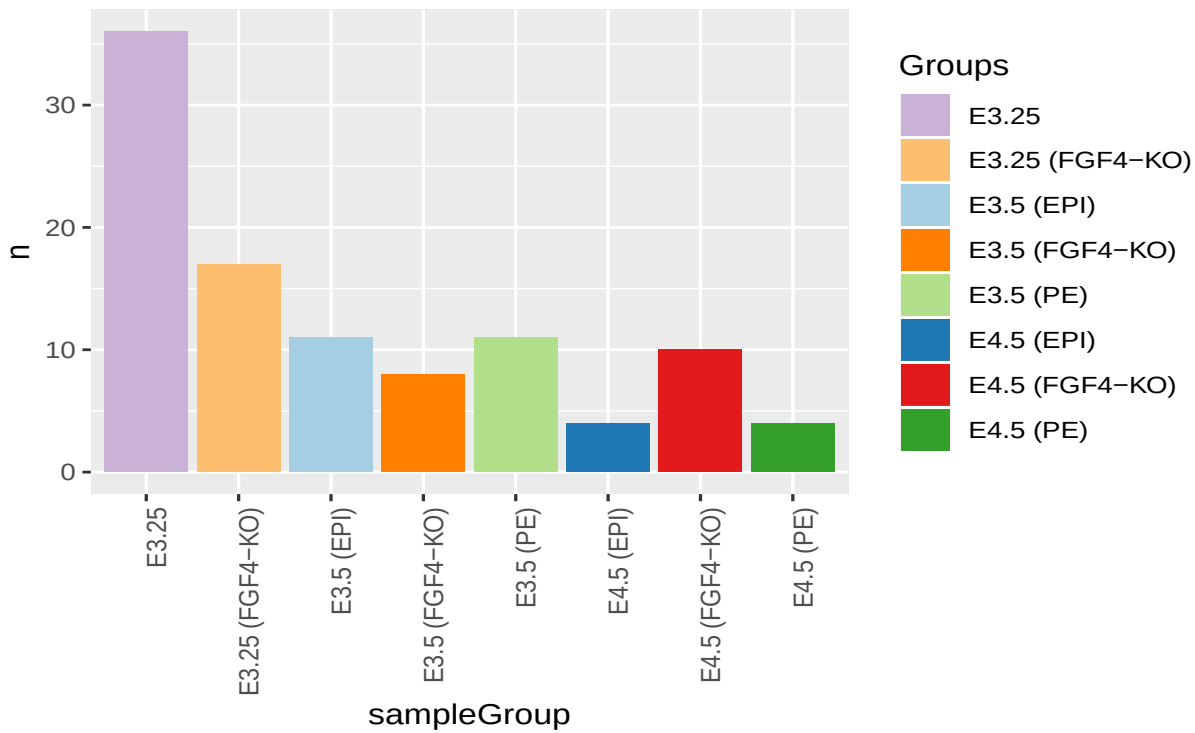
```
pb
```

Fig. 8: 'pb': without a geometric object, the plot remains empty.

Stepwise ggplot

We can simply add on the other components of our plot through the + operator.

```
pb = pb + geom_bar(stat = "identity")
pb = pb + aes(fill = sampleGroup)
pb = pb + theme(axis.text.x = element_text(angle = 90, hjust = 1))
pb = pb + scale_fill_manual(values = groupColor, name = "Groups")
pb
```



More options

```
pb.polar = pb + coord_polar() +
  theme(axis.text.x = element_text(angle = 0, hjust = 1),
        axis.text.y = element_blank(),
        axis.ticks = element_blank()) +
  xlab("") + ylab("")
pb.polar
```

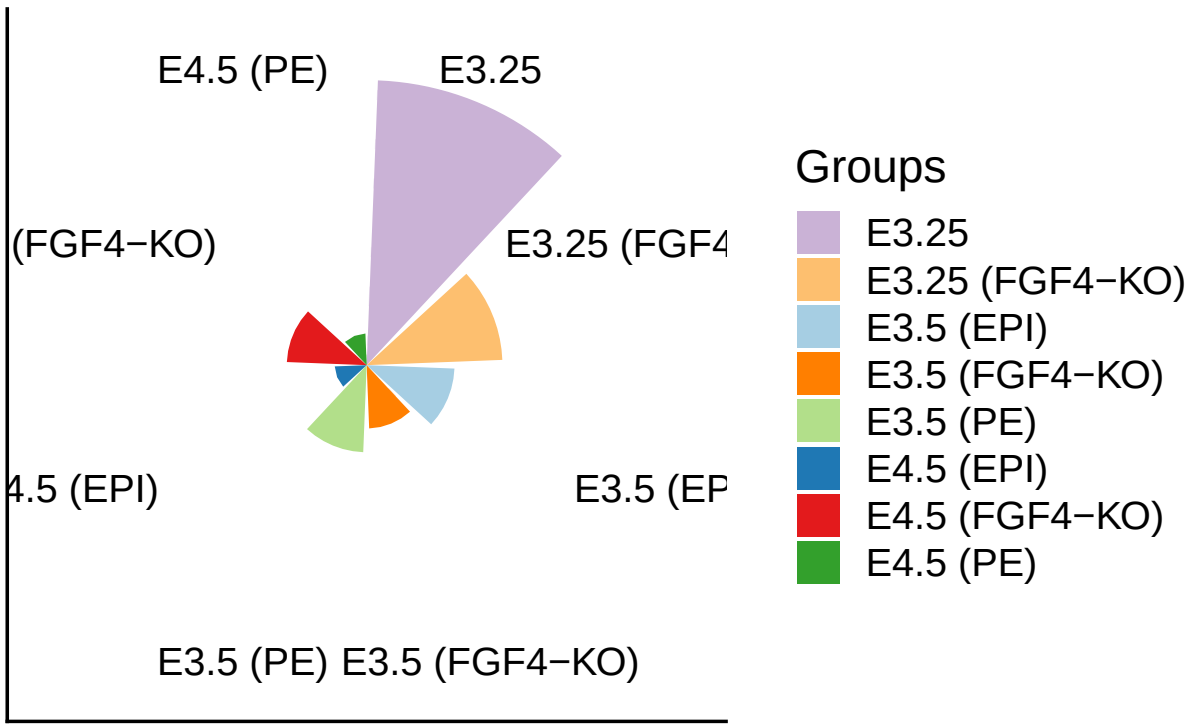


Fig. 9: A barplot in a polar coordinate system.

Visualizing data in 1D

- Bar plots
- Boxplots
- Violin plots
- Dot plots
- Beeswarm plots
- Density plots

Visualizing data in 1D: Example data

We will use the intensities of the following four genes Fgf4, Gata4, Gata6 and Sox2.

```
selectedProbes = c( Fgf4 = "1420085_at", Gata4 = "1418863_at",
                    Gata6 = "1425463_at",  Sox2 = "1416967_at")
```

We also add a column that provides the gene symbol along with the probe identifiers.

```
library("reshape2")
genes = melt(Biobase::exprs(x)[selectedProbes, ],
             varnames = c("probe", "sample"))
genes$gene = names(selectedProbes)[match(genes$probe, selectedProbes)]
# head(genes)
```

Barplots

Barplots are a popular way to display data.

```
ggplot(genes, aes( x = gene, y = value)) +
  stat_summary(fun.y = mean, geom = "bar")
```

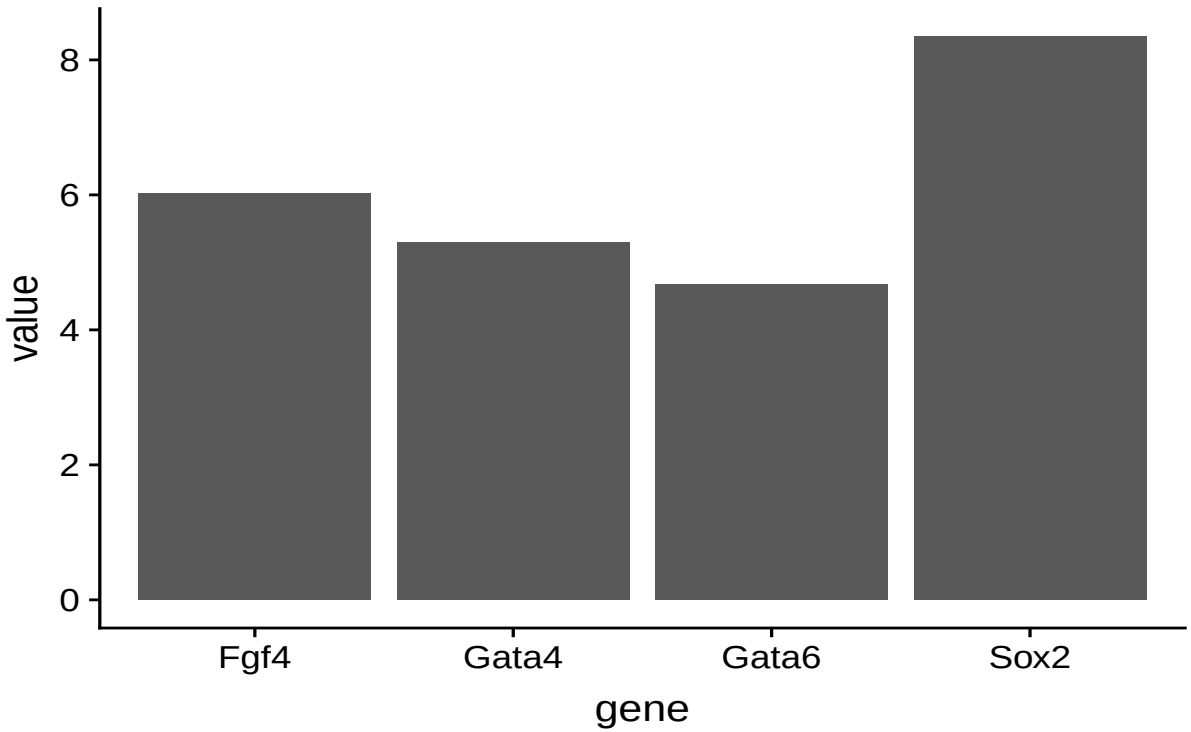


Fig. 10: Barplots showing the means of the distributions of expression measurements from four probes.

Barplots

We may want to add error bars.

```
library("Hmisc")
ggplot(genes, aes( x = gene, y = value, fill = gene)) +
  stat_summary(fun.y = mean, geom = "bar") +
  stat_summary(fun.data = mean_cl_normal, geom = "errorbar",
               width = 0.25)
```

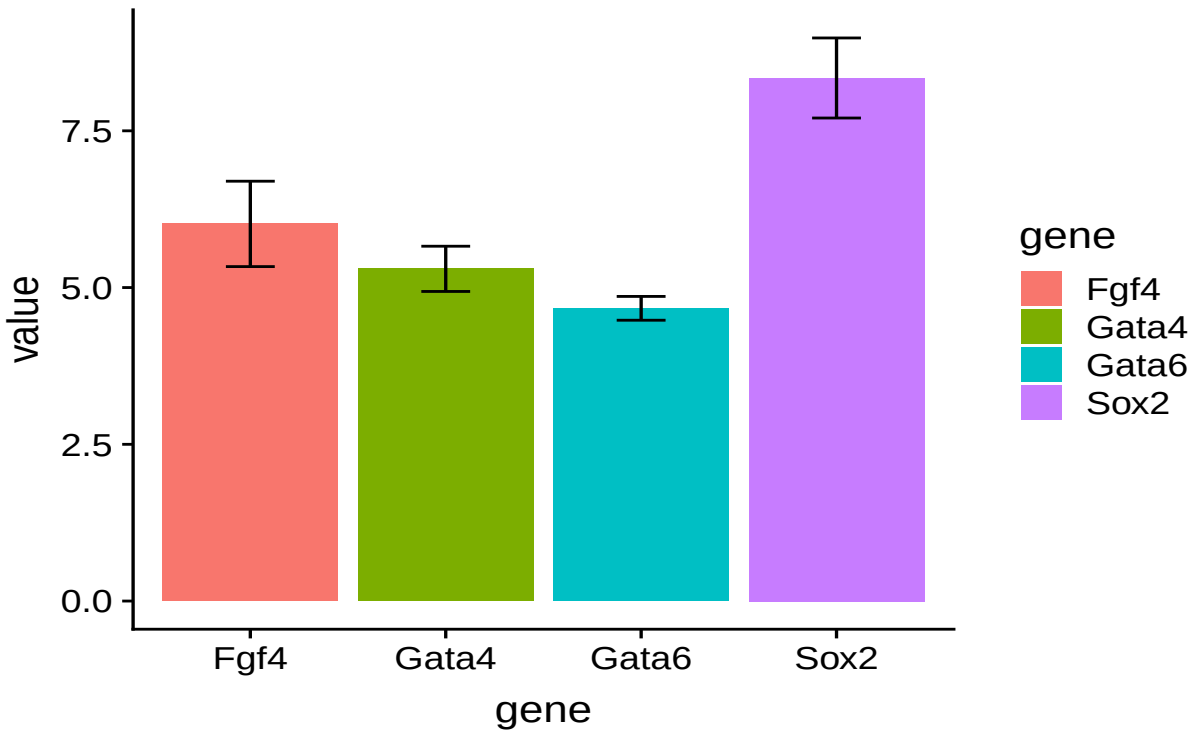


Fig. 11: Barplots with error bars indicating standard error of the mean.

Boxplots

Boxplots take up a similar amount of space as barplots, but are much more informative.

```
p = ggplot(genes, aes( x = gene, y = value, fill = gene))
p + geom_boxplot()
```

Boxplots

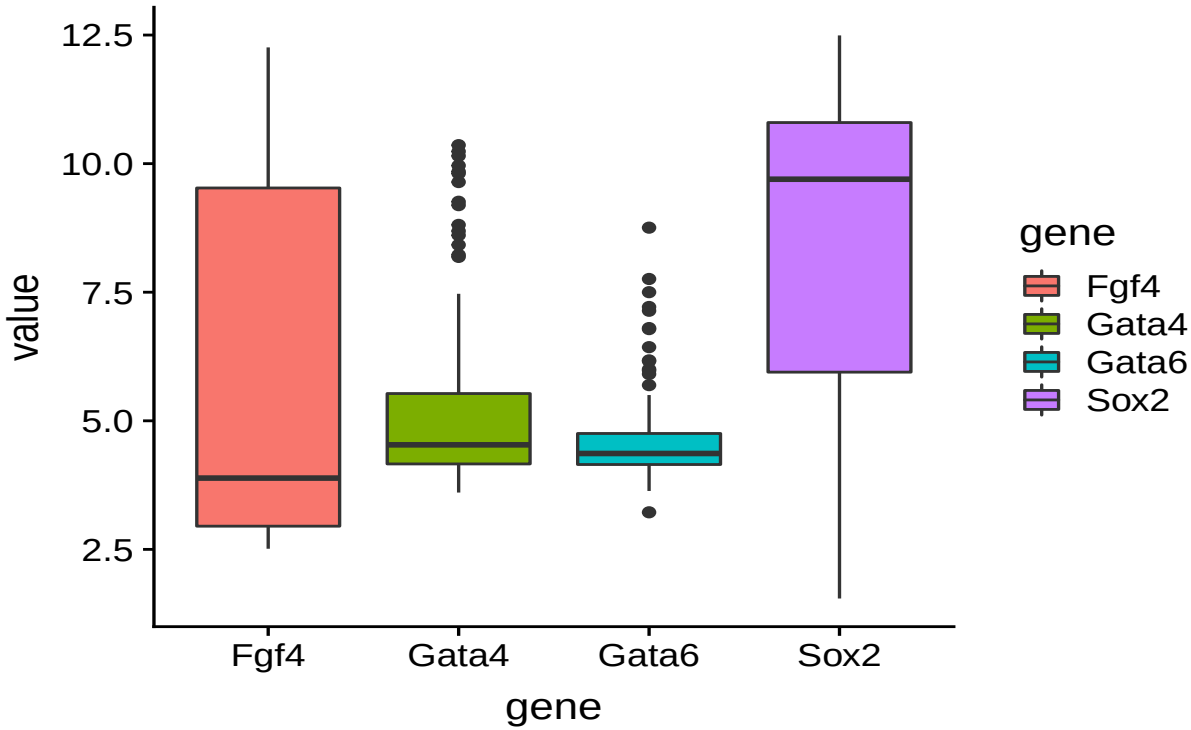


Fig. 12: Boxplots.

- Gata4, Gata6 have concentrated distributions.
- Distribution for Fgf4 is right-skewed.
- Distribution for Sox2 is left-skewed.

Violin plots

A violin plot provides an even more direct representation of the shape of the data distribution.

```
p + geom_violin()
```

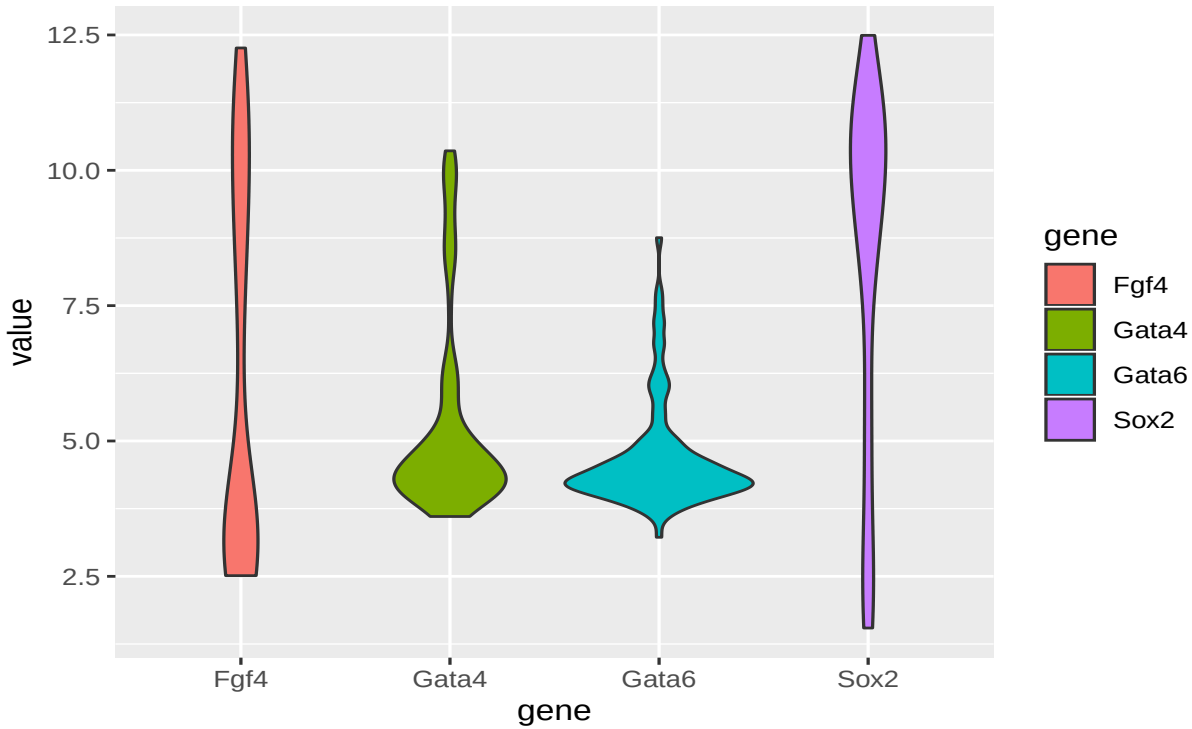


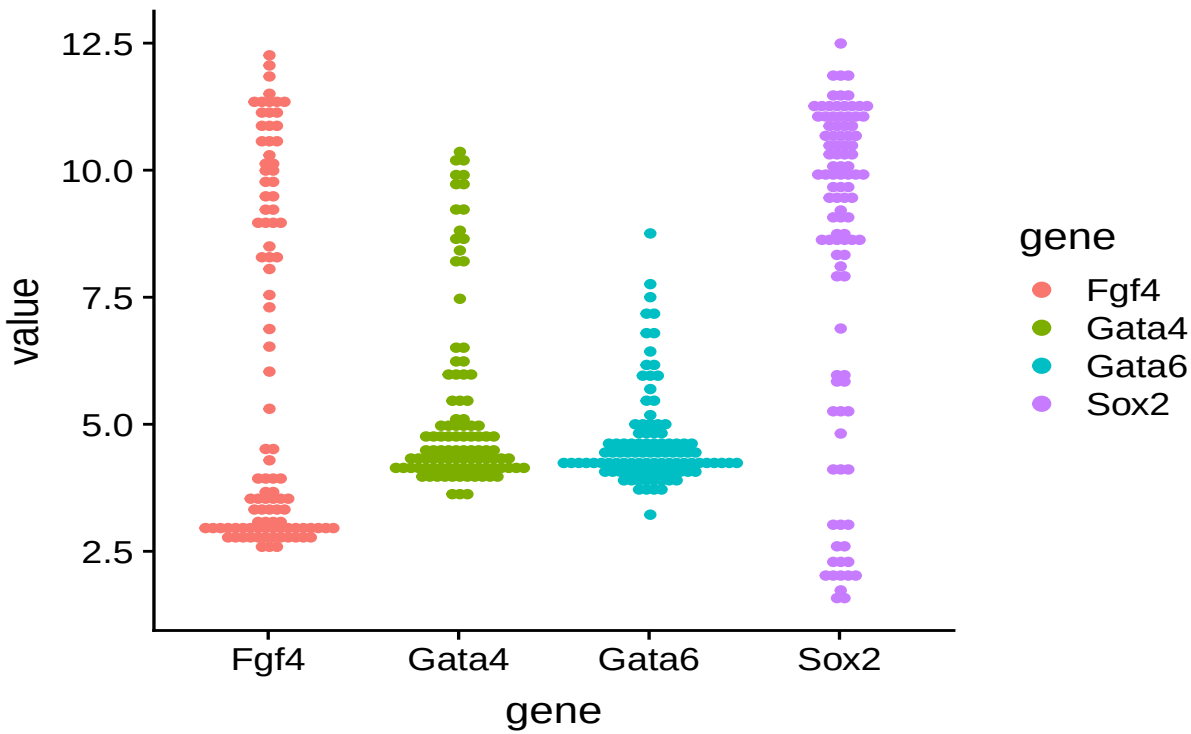
Fig. 13: Violin plots.

Dot plots

If the number of data points is not too large, it is a good practice to show the data points directly.

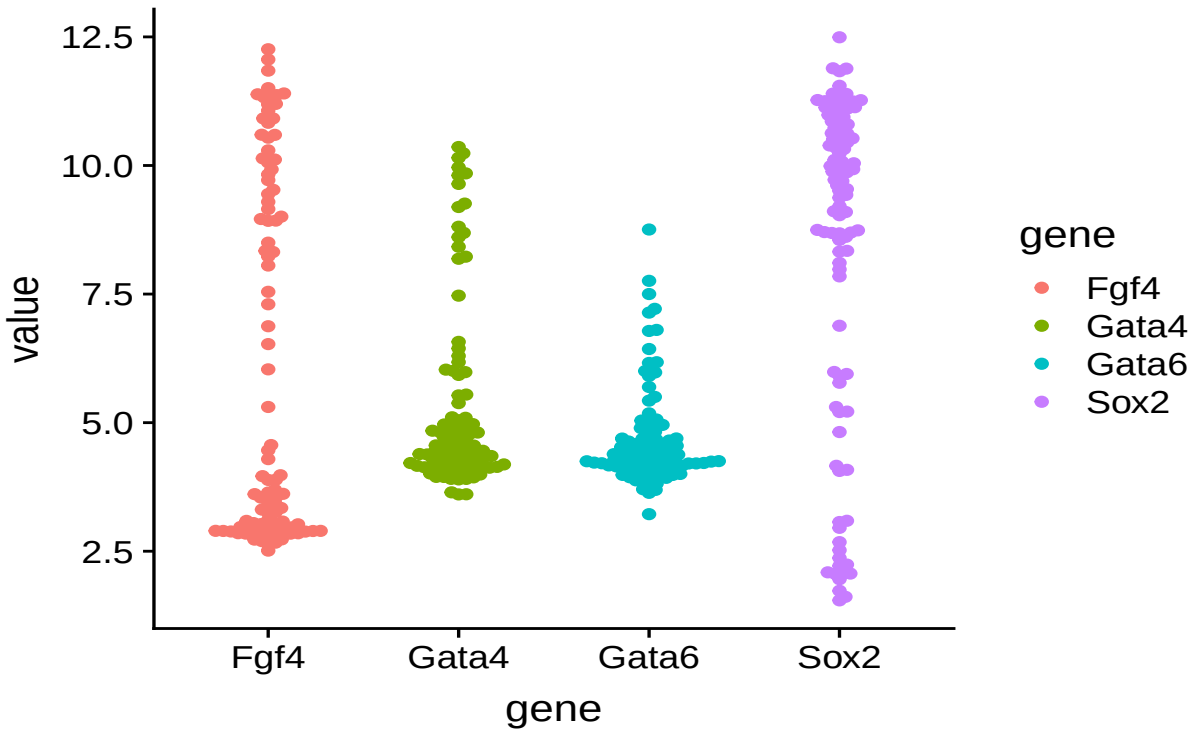
- Caveat: overlapping points can be visually unpleasant, or even obscure the data.

```
p + geom_dotplot(binaxis = "y", binwidth = 1/6, stackdir = "center",
                 stackratio = 0.75, aes(color = gene))
```



Beeswarm plots

```
library("ggbeeswarm")
p + geom_beeswarm(aes(color = gene))
```



Density plots

```
ggplot(genes, aes( x = value, color = gene)) + geom_density()
```

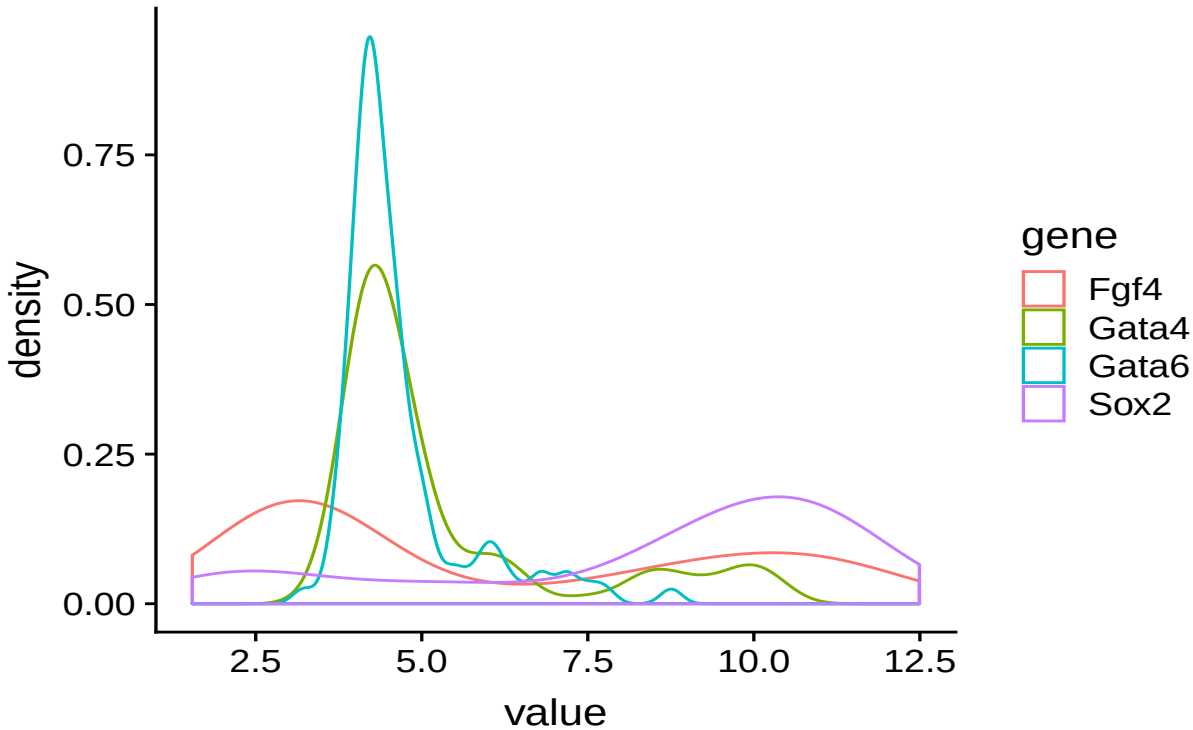


Fig. 14: Density plots.

Visualizing data in 2D

- Scatterplots: useful for visualizing treatment–response comparisons, associations between variables or paired data.
- Contour plots: 2D density plots.

Scatterplots

Let's look at differential expression between a wildtype sample `59 E4.5 (PE)` and an FGF4-KO sample `92 E4.5 (FGF4-KO)`.

```
dfx = as.data.frame(Biobase::exprs(x))
scp = ggplot(dfx, aes(x = `59 E4.5 (PE)` ,
                      y = `92 E4.5 (FGF4-KO)`))
scp + geom_point()
```

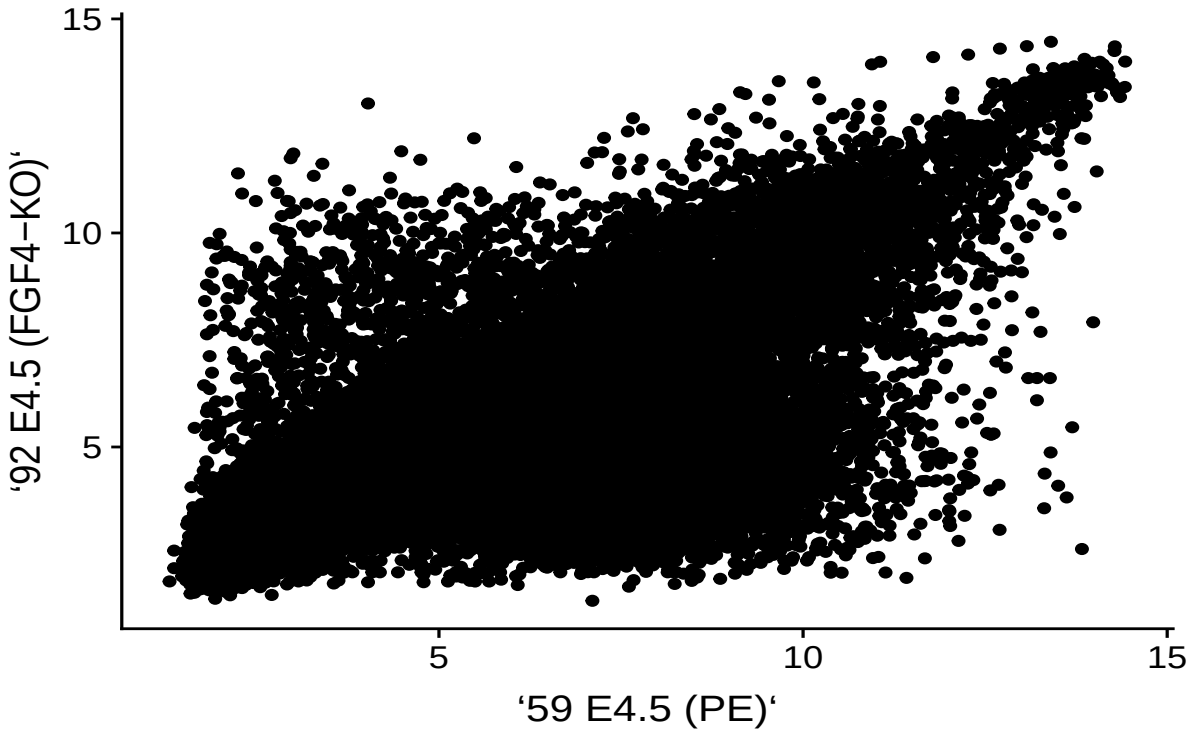


Fig. 15: Scatterplot of 45101 expression measurements for two of the samples.

Scatterplots

To ameliorate overplotting, we can adjust the transparency (alpha value) of the points.

```
scp  + geom_point(alpha = 0.1)
```

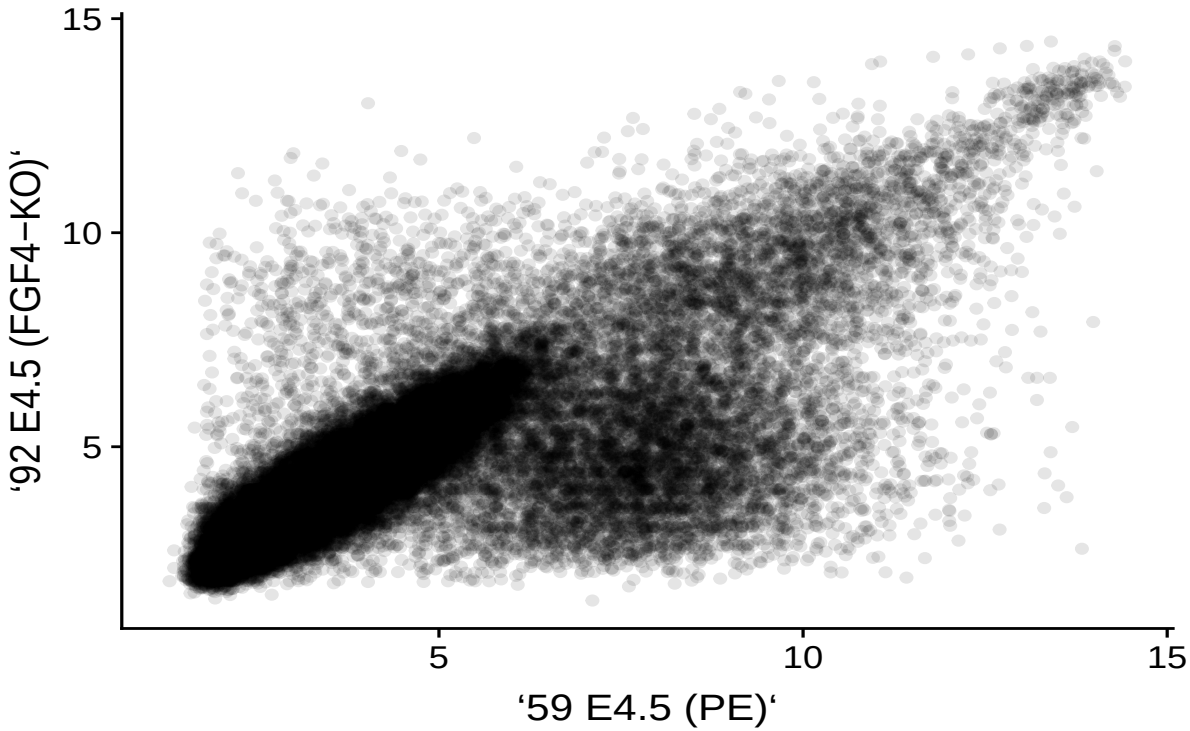


Fig. 16: As Fig. 17, but with semi-transparent points to resolve some of the overplotting.

Two simple rules on the aspect ratio

- If the variables on the two axes are measured in the same units, then make sure that the same mapping of data space to physical space is used, i.e., use `coord_fixed()`.
- If the variables on the two axes are measured in different units, then we can still relate them to each other by comparing their dimensions.

Visualizing more than two dimensions

- We can show multiple plots that result from repeatedly subsetting (or "slicing") our data based on one (or more) of the variables.
- This is called **faceting** and it enables us to visualize data in up to four or five dimensions.

Faceting

In the following example, note how we first defined a factor `dftx$lineage` with four levels.

```
library("magrittr")
dftx$lineage %<>% sub("^$", "no", .)
dftx$lineage %<>% factor(levels = c("no", "EPI", "PE", "FGF4-KO"))

ggplot(dftx, aes( x = X1426642_at, y = X1418765_at)) +
  geom_point() + facet_grid( . ~ lineage )
```

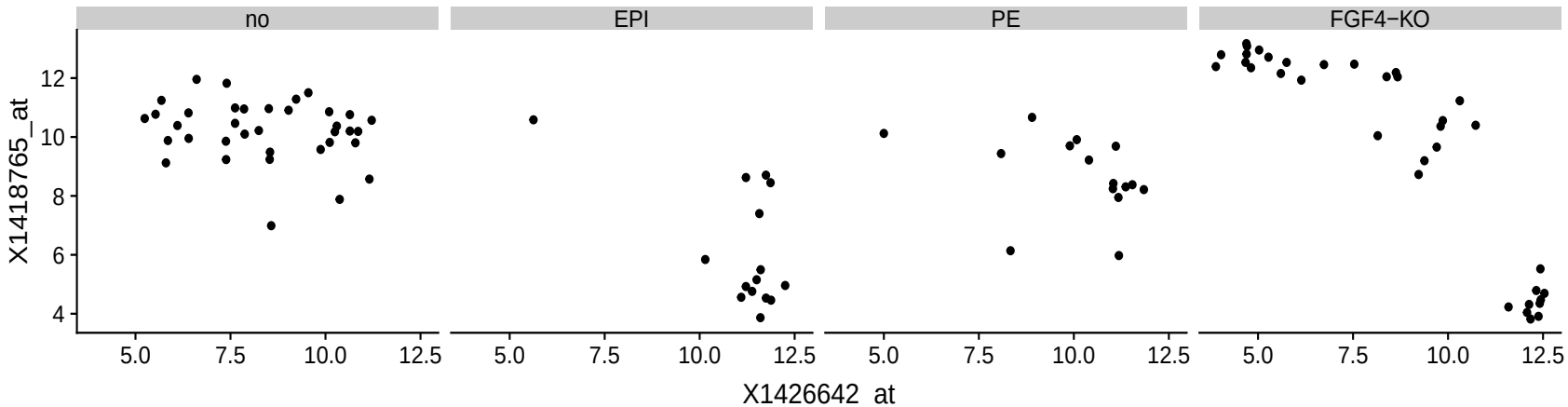


Fig. 17: An example of faceting.

Faceting

We can specify two faceting variables.

```
ggplot(dftx, aes( x = X1426642_at, y = X1418765_at)) +
  geom_point() + facet_grid( Embryonic.day ~ lineage )
```

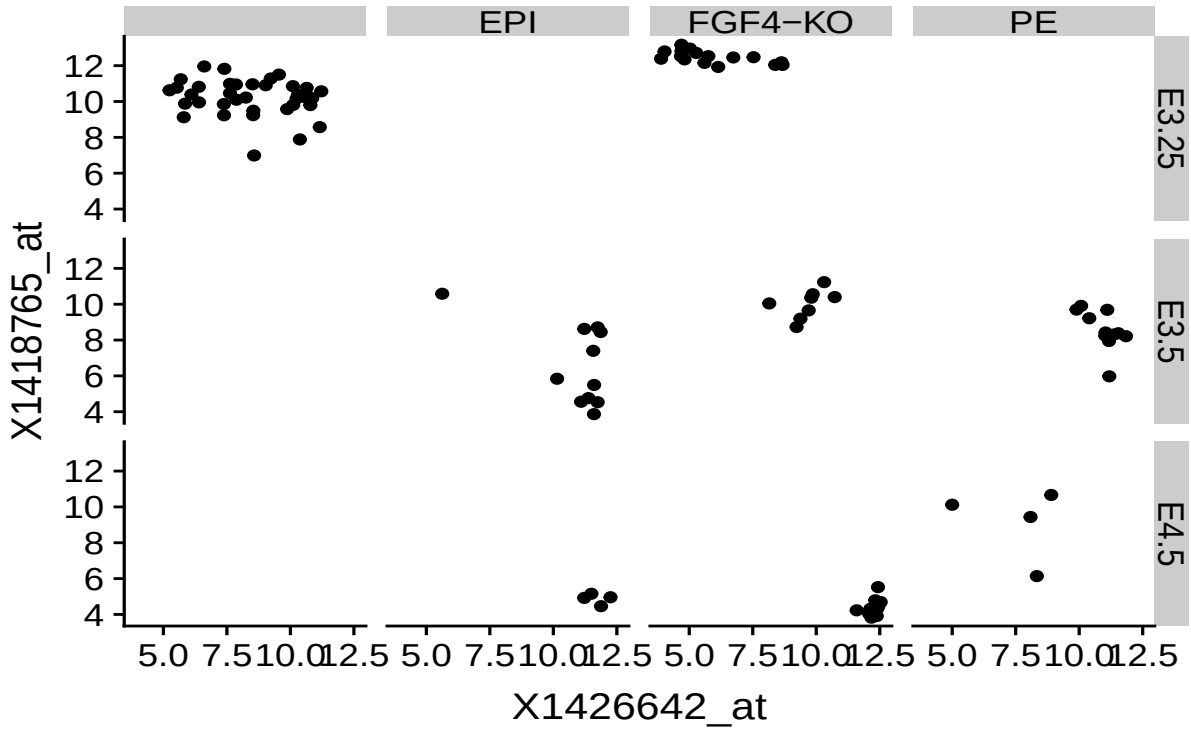


Fig. 18: The same data as in Fig. 22 split by the categorical variables ‘Embryonic.day‘ (rows) and ‘lineage‘ (columns).

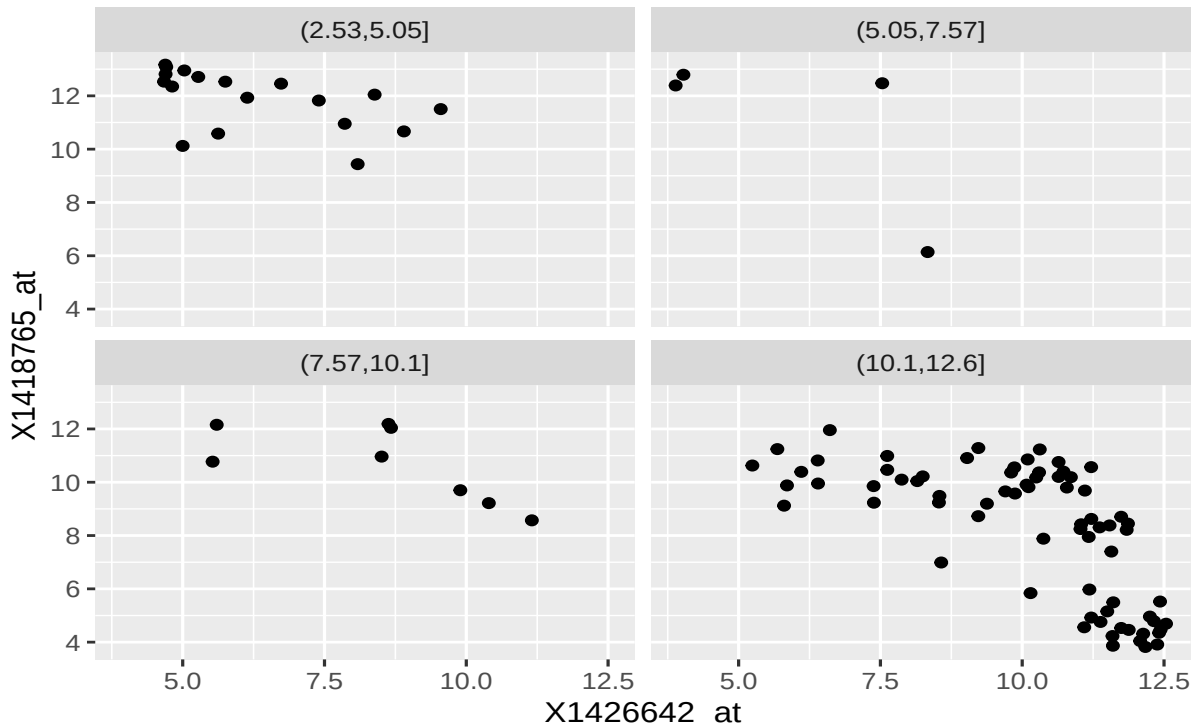
Faceting

Another useful function is `facet_wrap`:

- if the faceting variable has too many levels for all the plots to fit in one row or one column, this function can be used to wrap them into a specified number of columns or rows.

facet_wrap

```
dftx.mutate <- mutate(dftx, Tdgf1 = cut(X1450989_at, breaks = 4))
ggplot(dftx.mutate, aes( x = X1426642_at, y = X1418765_at)) +
  geom_point() +
  facet_wrap( ~ Tdgf1, ncol = 2 ) + theme_set(theme_cowplot())
```



Axes scales

- We have used the same axes scales for all panels.
- We could let them vary by setting `scales` of the `facet_grid` and `facet_wrap` functions.
- The trade-off: adaptive axes scales might offer more detail, but the panels are less comparable across the groupings.

```
ggplot(dftx.mutate, aes( x = X1426642_at, y = X1418765_at)) +
  geom_point() +
  facet_wrap( ~ Tdgf1, ncol = 2, scales = "free")
```

cowplot

The **cowplot** package is a simple add-on to **ggplot2**. It is meant to provide a publication-ready theme for ggplot2, one that requires a minimum amount of fiddling with sizes of axis labels, plot backgrounds, etc.

- The default theme in **ggplot2** is `theme_gray()`.
- The default theme in **cowplot** has no grid and redefined font sizes.
- **cowplot** also allows us to combine multiple plots into one graph via the function `plot_grid()`.

cowplot theme

```
p + geom_violin() + theme_gray()

library(cowplot)
p + geom_violin()
```

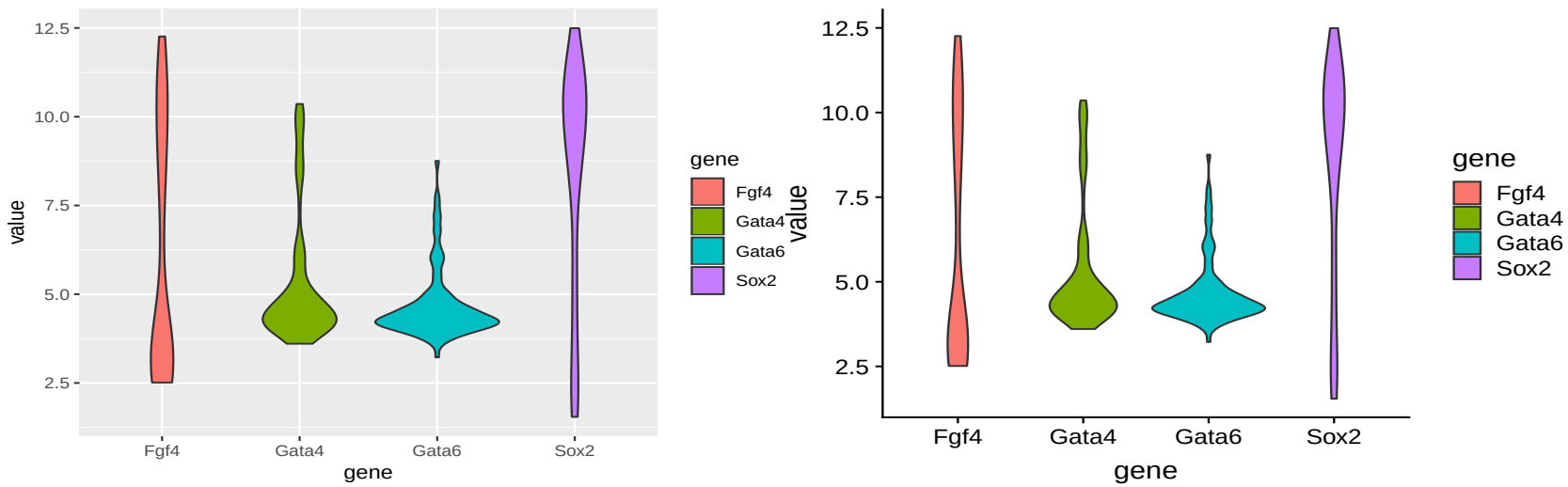
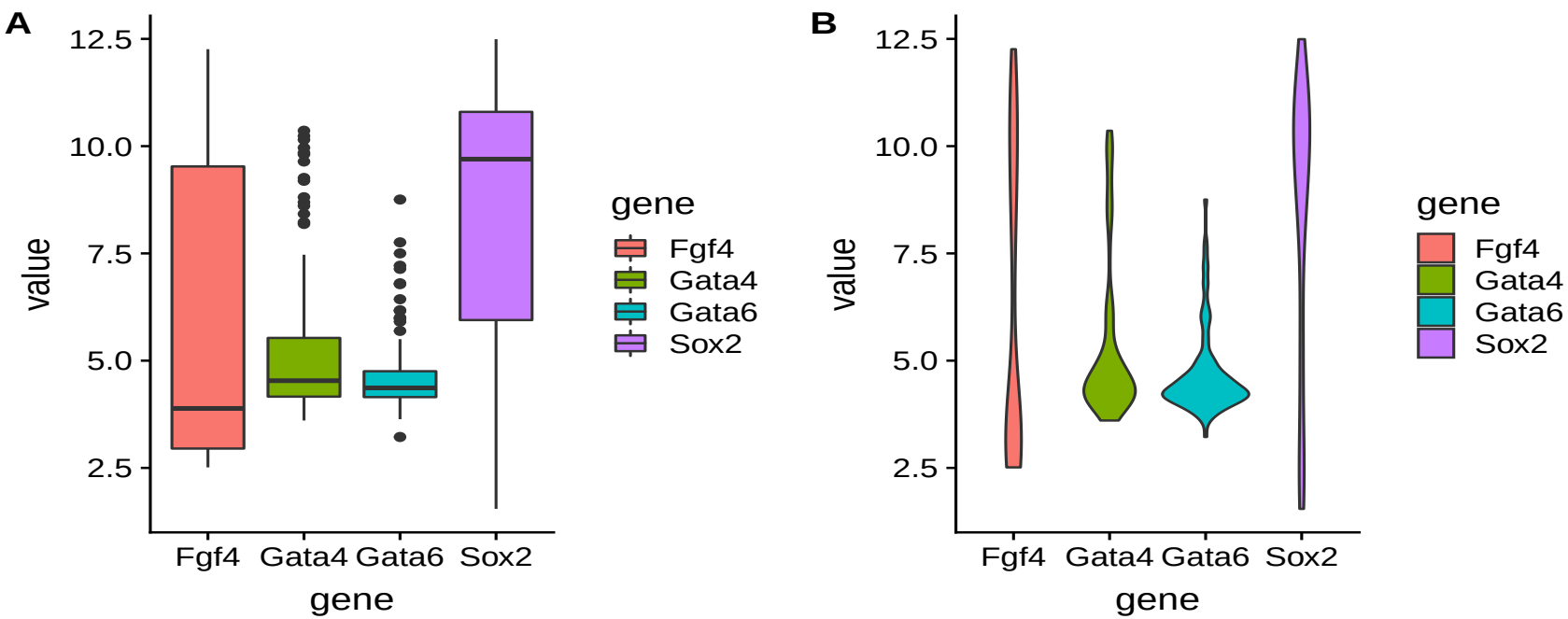


Fig. 19: Violin plots with default theme in the ‘ggplot2‘ package (left) and ‘cowplot‘ package (right).

Arranging graphs into a grid with cowplot

```
plot1 <- p + geom_boxplot()
plot2 <- p + geom_violin()
plot_grid(plot1, plot2, labels = c("A", "B"))
```



Interactive graphics

- **shiny** (demo) is a web application framework for R.
- **ggvis** extends the good features of **ggplot2** into the realm of interactive graphics.
- `plotly` is a great web-based tool for interactive graphic generation. You can view examples online at https://plot.ly/r/.

```
library("plotly")
plot_ly(economics, x = ~ date, y = ~ unemploy / pop)
```

Interactive graphics

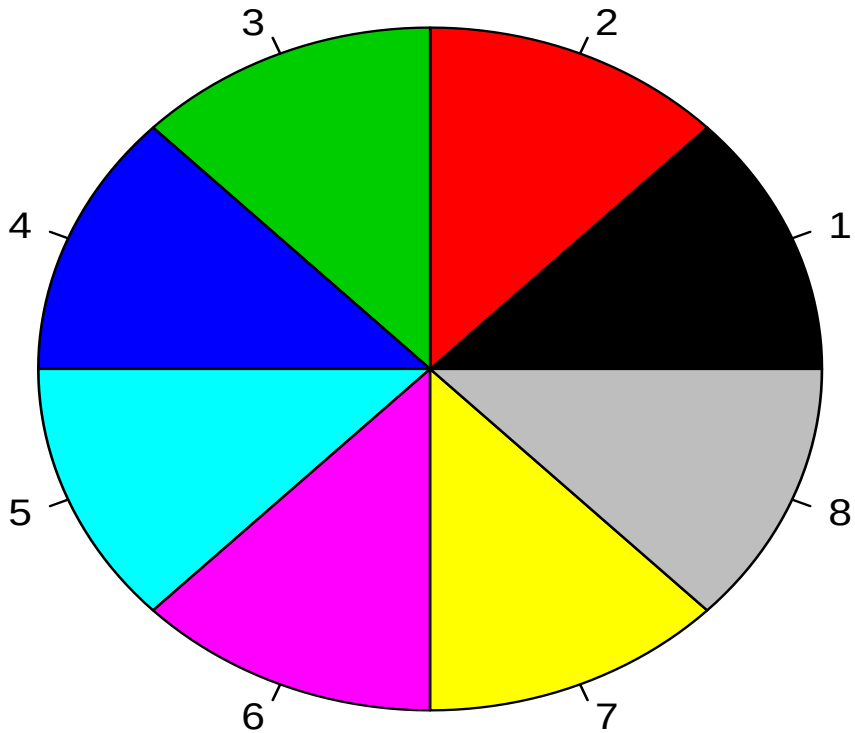
- The package `rgl` can be used to visualize 3D objects (e.g. a geometrical structure). It produces interactive viewer windows in which you can rotate the scene, zoom in and out, et.
- The following rendering requires you to have installed *XQuartz* on a Mac laptop.

```
data("volcano")
volcanoData = list(
  x = 10 * seq_len(nrow(volcano)),
  y = 10 * seq_len(ncol(volcano)),
  z = volcano,
  col = terrain.colors(500)[cut(volcano, breaks = 500)]
)
library("rgl")
with(volcanoData, persp3d(x, y, z, color = col))
```

Basic R colors

An important consideration in making plots is the coloring we use. R has built-in color scheme, such as

```
pie(rep(1, 8), col=1:8)
```

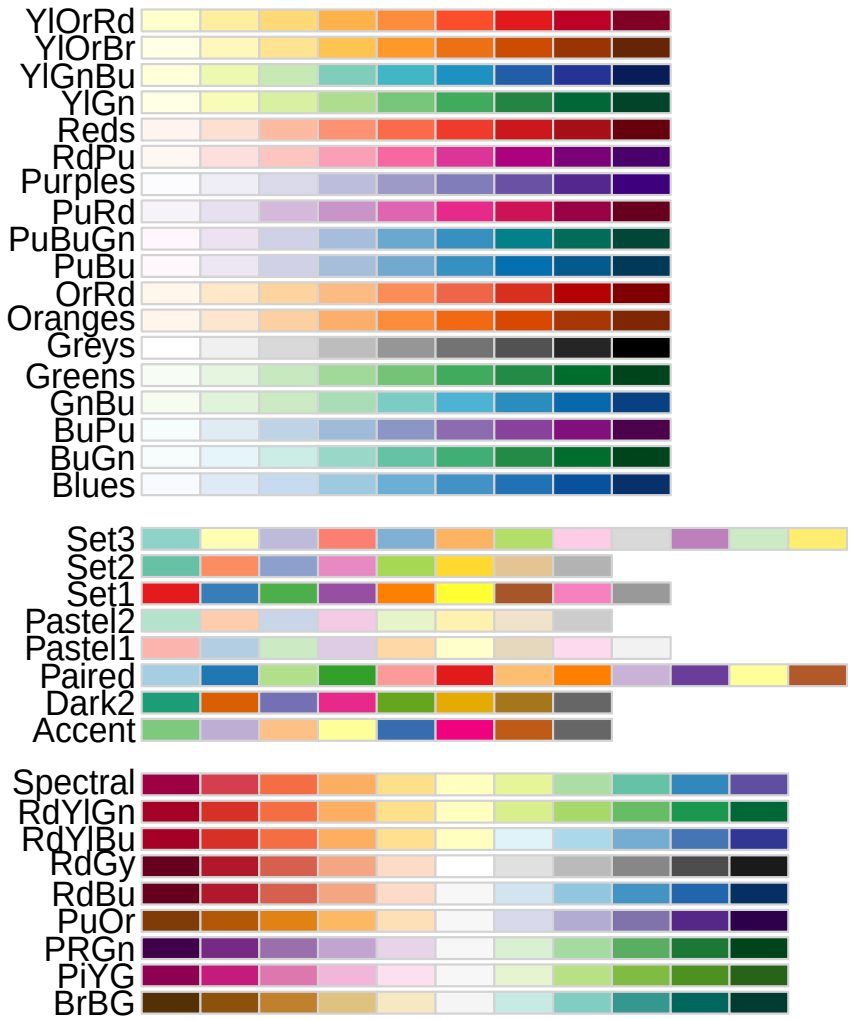


But these colors are harsh on eyes, and there are better alternatives.

RColorBrewer

We can look at all colors using the following function:

```
display.brewer.all()
```



RColorBrewer

We can learn more about them from `brewer.pal.info`.

```
head(brewer.pal.info)
```

```
##      maxcolors category colorblind
## BrBG        11      div       TRUE
## PiYG        11      div       TRUE
## PRGn        11      div       TRUE
## PuOr        11      div       TRUE
## RdBu        11      div       TRUE
## RdGy        11      div      FALSE
```

```
table(brewer.pal.info$category)
```

```
##
##  div qual  seq
##    9    8   18
```

The palettes are divided into three categories: sequential, qualitative and diverging.

RColorBrewer

We can use the function `brewer.pal` to obtain the colors from a particular palette.

```
brewer.pal(4, "RdYlGn")
```

Sometimes we may want more colors (e.g. when we plot a heatmap). We can interpolate using the `colorRampPalette` function.

```
mypalette  = colorRampPalette(c("darkorange3", "white","darkblue"))(100)
# head(mypalette)

par(mai = rep(0.1, 4))
image(matrix(1:100, nrow = 100, ncol = 10), col = mypalette,
        xaxt = "n", yaxt = "n", useRaster = TRUE)
```

Fig. 20: A quasi-continuous color palette derived by interpolating between the colors ‘darkorange3‘, ‘white‘ and ‘darkblue‘.

Heatmaps

Heatmaps are powerful for visualizing large, matrix-like datasets, and provide a quick overview of the patterns that might be present in the data.

```
library("pheatmap")
topGenes = order(rowVars(Biobase::exprs(x)), decreasing = TRUE)[1:500]
rowCenter = function(x) { x - rowMeans(x) }
centered.topGenes <- rowCenter(Biobase::exprs(x)[ topGenes, ] )

pheatmap( centered.topGenes,
          show_rownames = FALSE, show_colnames = FALSE,
          breaks = seq(-5, +5, length = 101),
          annotation_col =
            pData(x)[, c("sampleGroup", "Embryonic.day", "ScanDate") ],
          annotation_colors = list(
            sampleGroup = groupColor,
            genotype = c(`FGF4-KO` = "chocolate1", `WT` = "azure2"),
            Embryonic.day = setNames(brewer.pal(9, "Blues")[c(3, 6, 9)],
                                     c("E3.25", "E3.5", "E4.5")),
            ScanDate = setNames(brewer.pal(nlevels(x$ScanDate), "YlGn"),
                                levels(x$ScanDate))),
          cutree_rows = 4)
```

Heatmaps

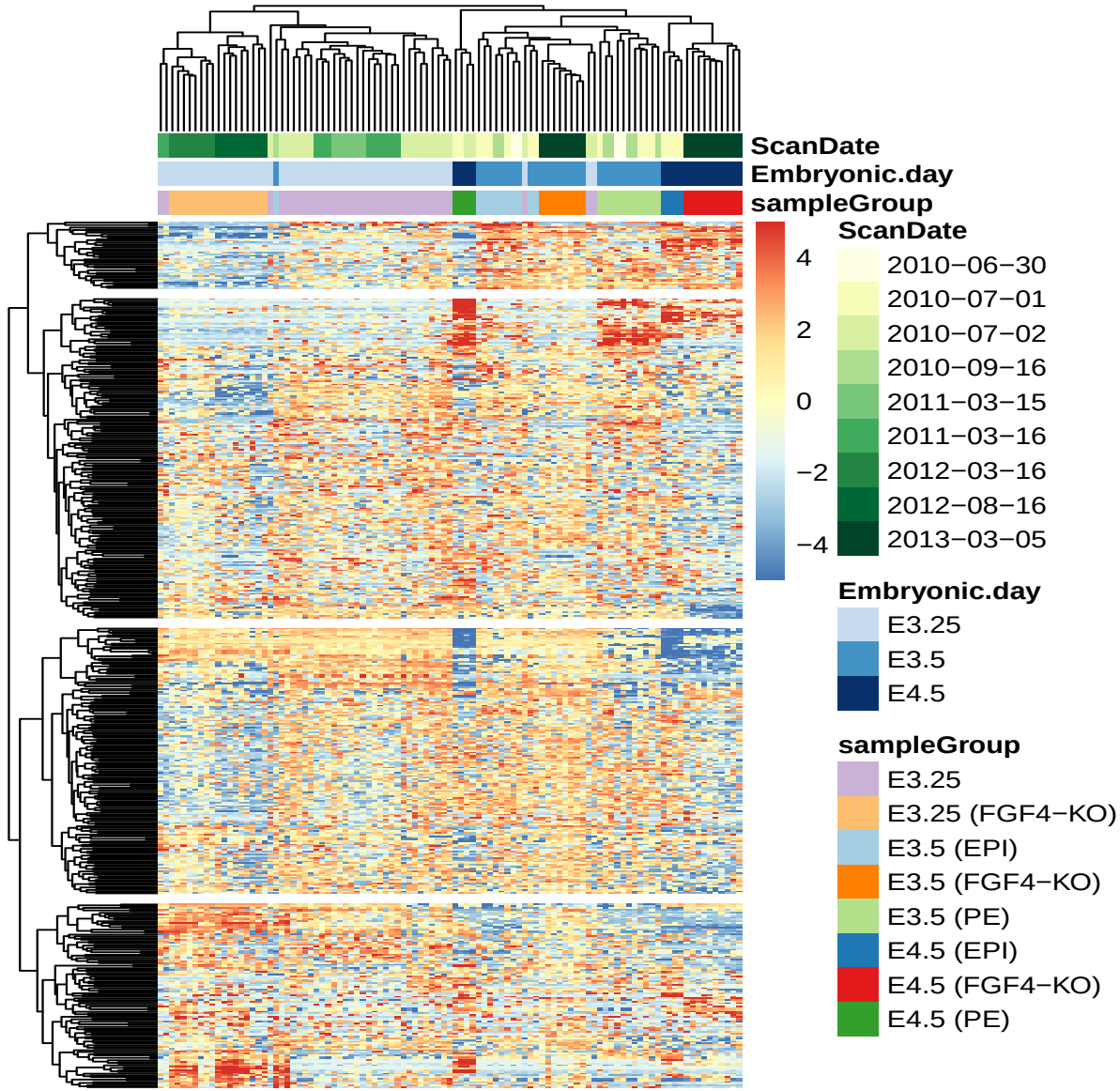


Fig. 21: A heatmap of relative expression values. The color scale uses a diverging palette whose midpoint is at 0.

Heatmaps

```
pheatmap( centered.topGenes,
          show_rownames = FALSE, show_colnames = FALSE,
          breaks = seq(-5, +5, length = 101),
          annotation_col =
            pData(x)[, c("sampleGroup", "Embryonic.day", "ScanDate") ],
          annotation_colors = list(
            sampleGroup = groupColor,
            genotype = c(`FGF4-KO` = "chocolate1", `WT` = "azure2"),
            Embryonic.day = setNames(brewer.pal(9, "Blues")[c(3, 6, 9)],
                                     c("E3.25", "E3.5", "E4.5")),
            ScanDate = setNames(brewer.pal(nlevels(x$ScanDate), "YlGn"),
                                levels(x$ScanDate))),
          cutree_rows = 4)
```

- `show_rownames` and `show_colnames`: whether the row and column names are printed.
- `annotation_col`: annotate samples; can also use `annotation_row`.
- `annotation_colors`
- `cutree_rows`: we cut the row dendrogram into four (an arbitrarily chosen number) clusters.

Dendrogram ordering

- The trees at the left and the top of Fig. 21 represent the result of a hierarchical clustering algorithm and are also called *dendrograms*.
- Ordering the rows and columns by cluster dendrogram is an arbitrary choice, and you could just as well make others.
- Even if you settle on dendrogram ordering, there is an essentially arbitrary choice at each internal branch, as each branch could be flipped without changing the topology of the tree.
- We will learn about clustering and methods for evaluating cluster significance in Chapter 5.

Data transformations

- The data in `dftx` represents logarithm (base 2) transformed microarray fluorescence intensities.
- The logarithm transformation is attractive because it has a definitive meaning - a move up or down by the same amount on a log-transformed scale corresponds to the same multiplicative change on the original scale: $\log(ax) = \log a + \log x$.
- The logarithm transformation is not good enough when the data include zero or negative values, or when even on the logarithmic scale the data distribution is highly uneven.

Data transformations

```
gg = ggplot(tibble(A = Biobase::exprs(x)[, 1], M = rnorm(length(A))),
            aes(y = M))
gg + geom_point(aes(x = A), size = 0.2)
gg + geom_point(aes(x = rank(A)), size = 0.2)
```

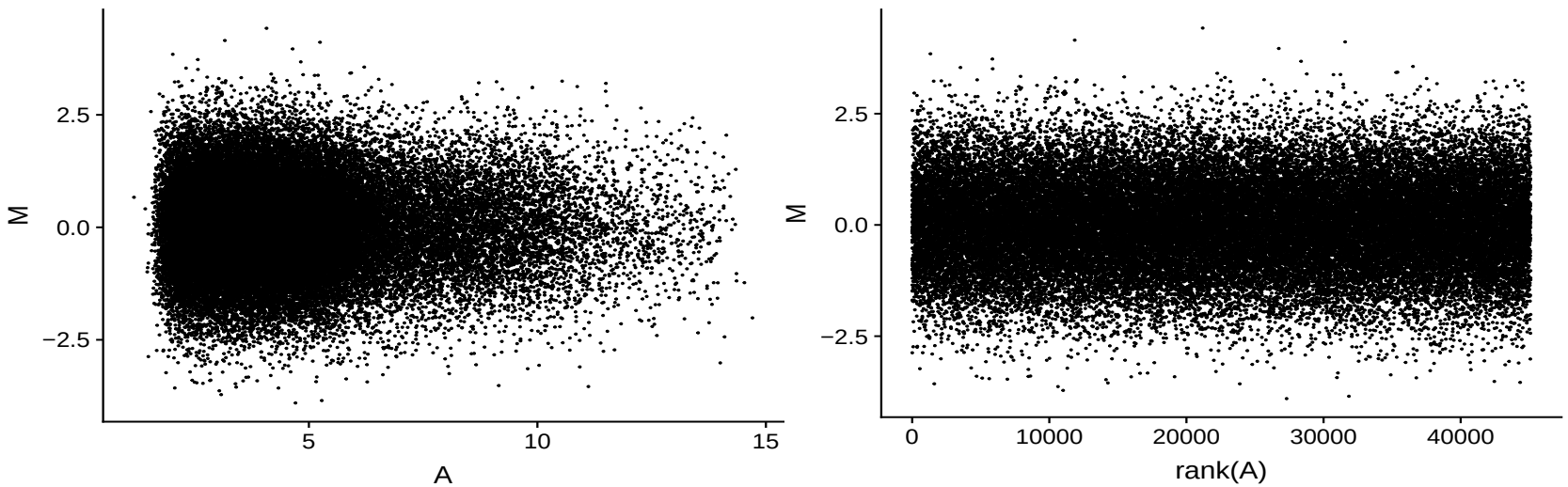


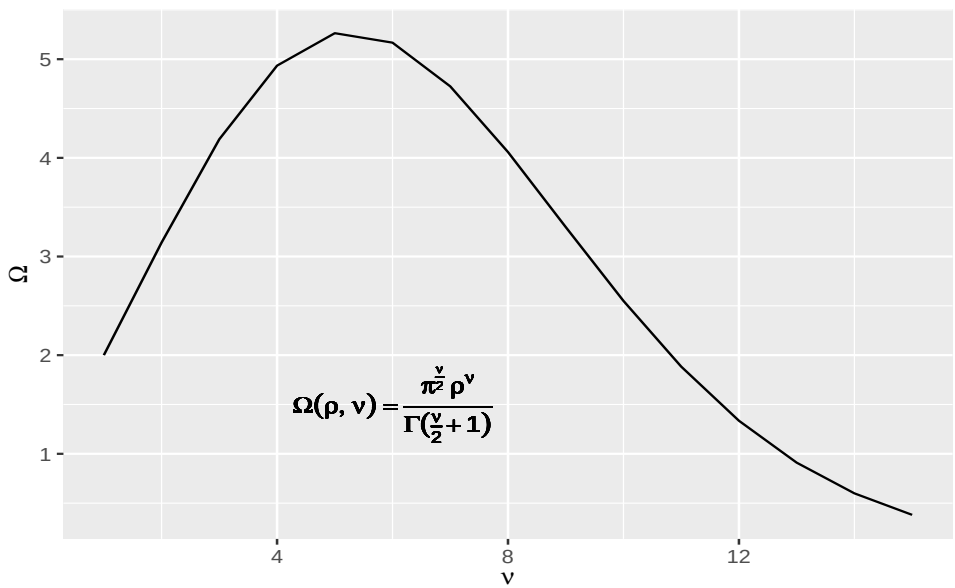
Fig. 22: The effect of rank transformation on the visual perception of dependency.

Mathematical symbols

- We can use mathematical notation in plot labels, using a notation that is a mix of R syntax and LaTeX-like notation.
- See `help(plotmath)` for details.

```
volume = function(rho, nu) pi^(nu/2) * rho^nu / gamma(nu/2+1)

ggplot(tibble(nu = 1:15, Omega = volume(1, nu)), aes(x = nu, y = Omega)) +
  geom_line() +
  xlab(expression(nu)) + ylab(expression(Omega)) +
  geom_text(label="Omega(rho,nu)==frac(pi^frac(nu,2)~rho^nu, Gamma(frac(nu,2)+1))",
            parse = TRUE, x = 6, y = 1.5)
```



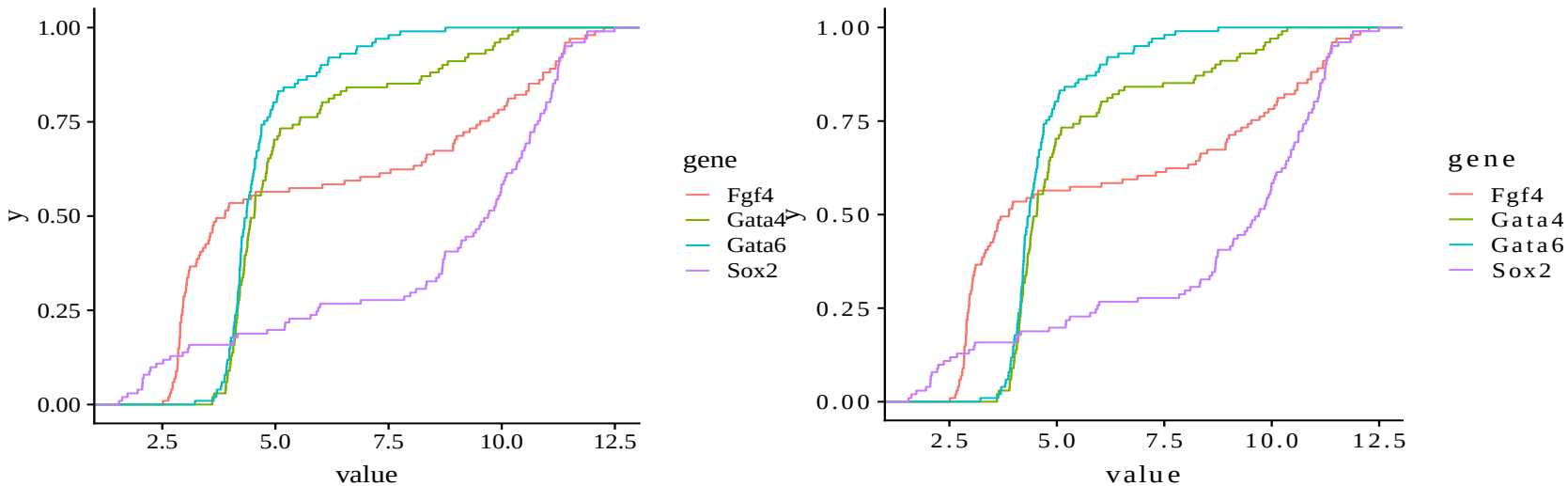
Other fonts

We can also switch to other fonts easily.

```
ggplot(genes, aes( x = value, color = gene)) + stat_ecdf() +
  theme(text = element_text(family = "Times"))

# install.packages("extrafont")
# library(extrafont)
# font_import()

ggplot(genes, aes( x = value, color = gene)) + stat_ecdf() +
  theme(text = element_text(family = "Verdana"))
```



Summary of this lecture

- The grammar of graphics is a powerful set of concepts to reason about graphics and to communicate our intentions for a data visualization to a computer.
- Avoid *laziness* – just using the software's defaults without thinking about the options is dangerous.
- Avoid *getting carried away* – adding lots of visual candy that just clutters up the plot but has no real message
- Look carefully at lots of visualizations made by others.
- Experiment with making your own visualizations.

Further reading

1. Wickham, Hadley. ggplot2: elegant graphics for data analysis. Springer, 2016.
2. https://ggplot2.tidyverse.org/reference/
3. Wilke, Claus O. "cowplot: streamlined plot theme and plot annotations for 'ggplot2'. R package version 0.9.4; 2018." URL: https://cran.r-project.org/package=cowplot.
4. Ohnishi, Yusuke, et al. "Cell-to-cell expression variability followed by signal reinforcement progressively segregates early mouse lineages." Nature cell biology 16.1 (2014): 27.