BDSD 2020

Introduction to ggplot2

Andrew Skol

Department of Pathology and Laboratory Medicine

askol@luriechildrens.org

1 Introduction: What is ggplot2

ggplot2 is an R package that, for all intents and purposes, has supplanted the native R plotting functions. ggplot2 is a very structed set of plotting functions that share substantial syntax across functions, making it easier to transfer knowledge for different plotting purposes. The functions also provide fine control over the look of plots, both in terms of layout, plot aethetics, and labelling.

The goal of this tutorial is to gain experience with some of the most common plotting function, learn how to modify common plotting features, learn how to control color, and how to massage the data into the correct format for plotting.

You will gain experience with histograms, distributions, points, bar plots, violin plots, and heatmaps.

This tutorial is based on the “show as you go” philosophy, meaning instead of saying what I’m going to show you and why and what it does, I will instead show you how I did something and then we can talk about how it works.

2 Get this tutorial

The github repository for this tutorial can be found here:

https://github.com/askol/BDSD2020

And the data can be found here:

https://www.dropbox.com/s/2q6qiva7myc4zjn/betas.rds?dl=0
https://www.dropbox.com/s/cg8ayt3zx3jnwjr/ps.rds?dl=0
https://www.dropbox.com/s/0glmrh7zl3j35gg/sampleSize.rds?dl=0

3 Data set: GTEx data (Genotype Tissue Expression Study (v7))

3.1 Data availability and content

Data is accessible from: https://gtexportal.org/home/datasets

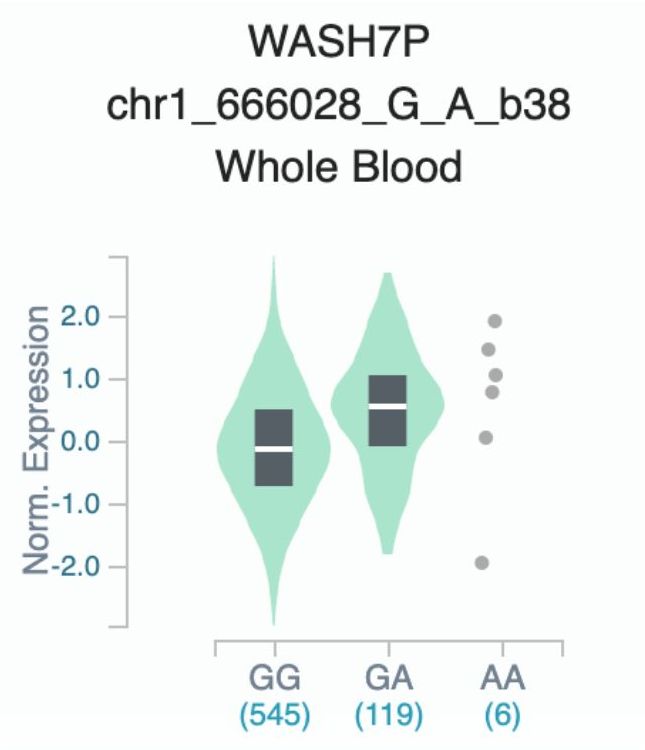
The actual file is:
https://storage.googleapis.com/gtex_analysis_v7/single_tissue_eqtl_data/GTEx_Analysis_v7_eQTL.tar.gz

The unzipped file contain information about eQTL from 48 tissues.

Information includes:

1. Name of genetic variant
2. Name of gene
3. pval_nominal
4. slope

3.2 What is an eQTL



3.3 Data construction

The data from each tissue has been read in and stored in three data.frames. The code to do this is here:

```
dataDir <- "/Users/askol/Dropbox (Personal)/GTEx/V7/GTEx_Analysis_v7_eQTL/"
files <- dir(dataDir, pattern = "pairs")
tissues <- gsub("\\.v7.+","", files)
data <- list()

sampFile <- paste0(dataDir,
"GTEx_Analysis_v8_Annotations_SampleAttributesDS.txt")
sampInfo <- read.table(file = sampFile, header=T, as.is=T, sep="\t", quote = "")
sampInfo <- sampInfo %>% mutate(ID = gsub("-.+-.+$","",SAMPID))
ID <- sapply(sampInfo$SAMPID, strsplit, split="-")
ID <- sapply(ID, function(x){paste(x[1],x[2], sep="-")})
ID <- unlist(ID)
sampInfo$ID <- ID
```

```
a <- sampInfo %>% distinct(ID, SMTSD)
tissueN <- table(sampInfo$SMTSD)
names(tissueN) <- gsub(" - " , "_" , names(tissueN))
names(tissueN) <- gsub(" " , "_" , names(tissueN))
names(tissueN) <- gsub("\\(" , "" , names(tissueN))
names(tissueN) <- gsub("\\)" , "" , names(tissueN))
names(tissueN) <- gsub("Cells_Cultured_fibroblasts",
                       "Cells_Transformed_fibroblasts",
                       names(tissueN))
tissueN <- as.data.frame(tissueN)
names(tissueN) <- c("tissue", "N")
saveRDS(file = "Data/sampleSize.rds", tissueN)

for (i in 1:length(files)){
   file <- paste0(dataDir, files[i])
   print(paste0("Reading file", file))
   data[[i]] <-  fread(file, header=T)
}

## collect p-values
ps <- data[[1]][,c("variant_id","symbol", "pval_nominal")] %>%
  mutate(chr = gsub("_.+","",variant_id)) %>% filter(chr %in% c(1,6,20))
ps <- ps %>% rename(!!tissues[1] := pval_nominal)

for (i in 2:length(data)){
  print(paste0("Merging ",tissues[i]))
  tmp <- data[[i]][, c("variant_id","symbol","pval_nominal")] %>%
    mutate(chr = gsub("_.+","",variant_id)) %>% filter(chr %in% c(1,6,20))
  tmp <- tmp %>% rename(!!tissues[i] := pval_nominal)
  ps <- full_join(ps, tmp, by=c("variant_id" , "symbol", "chr"))
}

## collect betas
betas <- data[[1]][,c("variant_id","symbol", "slope")] %>%
  mutate(chr = gsub("_.+","",variant_id)) %>% filter(chr %in% c(1,6,20))
betas <- betas %>% rename(!!tissues[1] := slope)

for (i in 2:length(data)){
  print(paste0("Merging ",tissues[i]))
  tmp <- data[[i]][, c("variant_id","symbol","slope")] %>%
    mutate(chr = gsub("_.+","",variant_id)) %>% filter(chr %in% c(1,6,20))
```

```
  tmp <- tmp %>% rename(!!tissues[i] := slope)
  betas <- full_join(betas, tmp, by=c("variant_id" , "symbol", "chr"))
}
## save ps
psFile <- paste0(dataDir, "ps.rds")
saveRDS(file = psFile, ps)

## save betas
betasFile = paste0(dataDir,"betas.rds")
saveRDS(file = betasFile, betas)
```

3.4 Loading the data

Let's load the ps and betas variables

```
betasFile = "Data/betas.rds"
psFile <- "Data/ps.rds"
nFile <- "Data/sampleSize.rds"
betas = readRDS(file = betasFile)
ps = readRDS(file = psFile)
tissueN = readRDS(file = nFile)

## substitute gene names for ensembl ids in ps and betas
betas <- betas %>% mutate(gene_id = gsub("\\..+","", gene_id))
betas <- left_join(betas, grch38[,c("ensgene","symbol")], by = c("gene_id" =
"ensgene")) %>% mutate(symbol = ifelse(is.na(symbol), gene_id, symbol))

ps <- ps %>% mutate(gene_id = gsub("\\..+", "", gene_id))
ps <- left_join(ps, grch38[,c("ensgene","symbol")], by=c("gene_id" = "ensgene"))
%>%
  mutate(symbol = ifelse(is.na(symbol), gene_id, symbol))

ps[1:4,]

betas[1:4,]

tissueN[1:4,]
```

4 Setting up: the ggplot function

Now that we have some data to play with, we can start exploring how to perform some simple plotting functions using ggplot2.

4.1 Number of eQTL as a function of tissue

First we will plot the number of eQTL as a function of tissue using a histogram. Since a non-NA cell indicates that a variant is an eQTL, all we have to do is sum the number of non-NA cells in a column (tissue) to count the number of eQTL.

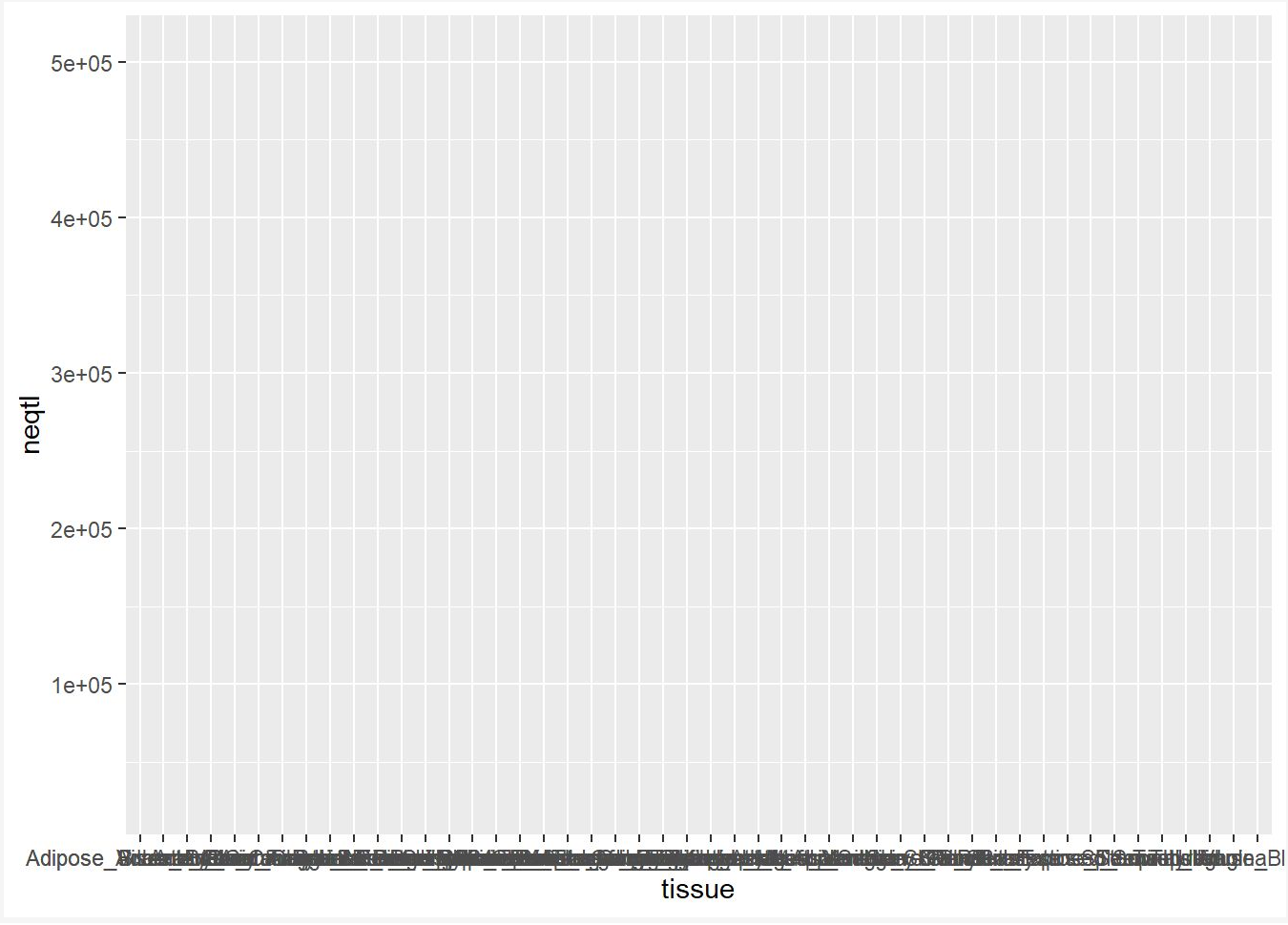
```
## remove unwanted columns
neqtl <- ps %>% select(-chr, -variant_id, -gene_id, -symbol)
## count number of eqtl in each column(tissue)
neqtl <- colSums(!is.na(neqtl))
## convert output to a data.frame
neqtl <- data.frame(tissue = names(neqtl), neqtl = neqtl)

neqtlchr <- c()
for (c in unique(ps$chr)){

  tmp <- ps %>% filter(chr == c) %>%
    select(-chr, -variant_id, -gene_id, --symbol)
  tmp <- colSums(!is.na(tmp))
  tmp <- data.frame(chr = c, tissue = names(tmp), neqtl=tmp)
  neqtlchr = rbind(neqtlchr, tmp)
}
```

4.2 The ggplot function

Any ggplot plot type will start with the ggplot function. It is used to define the dataset and the variables that will define the x and y axis (and other possible attributes). Aes is short of aesthetics, which defines the relationship between the variables in the

dataframe and plot features.

```
options(width=100)
p <- ggplot(data=neqtl, aes(x=tissue, y=neqtl))
plot(p)
```

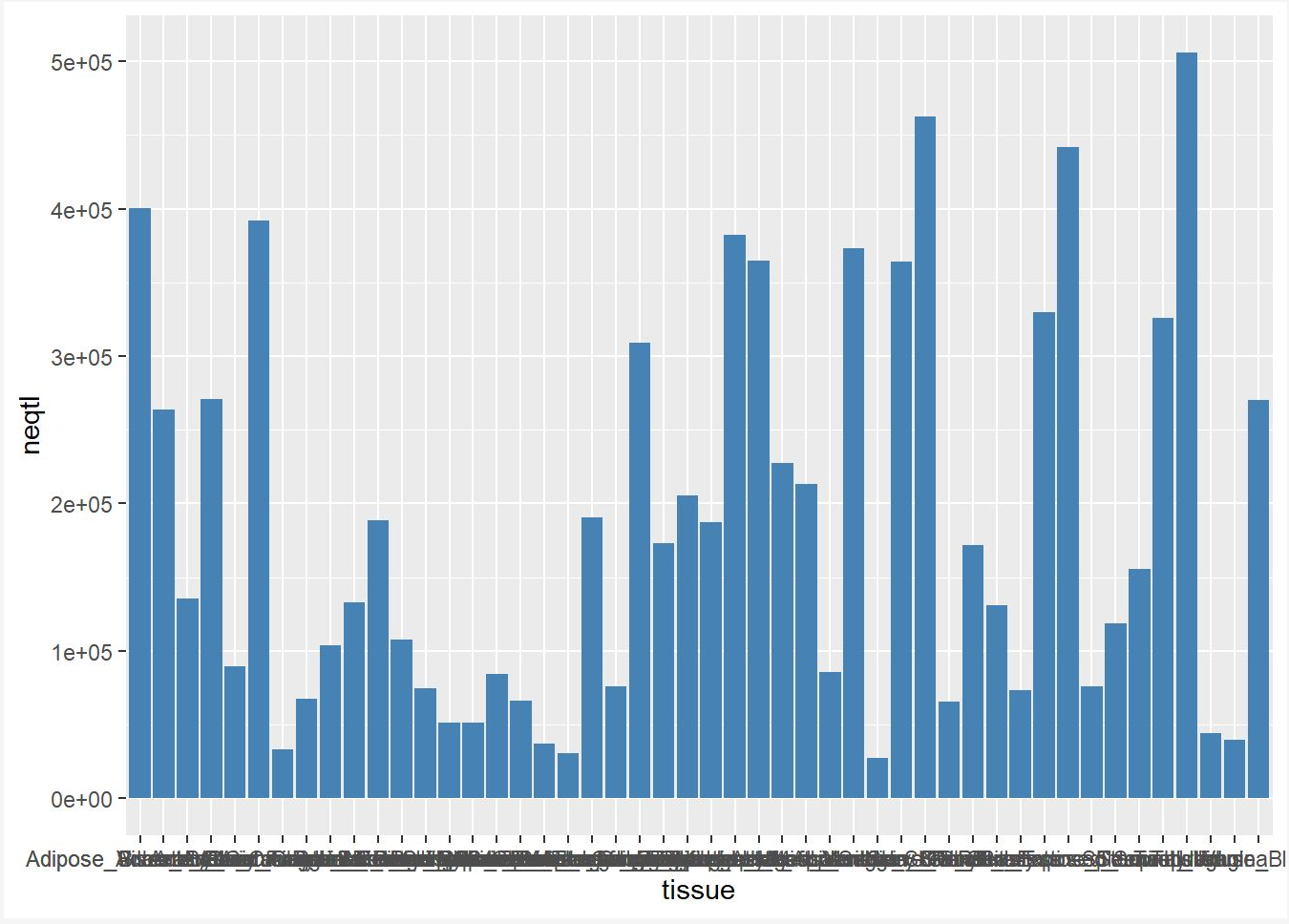


4.3 The barplot: geom_bar()

Next we add the geom_ function that defines the type of plot we want. Here we use geom_bar, because we want to make a barplot. These functions take arguments that are specific to the plot type and graphical elements specific to the plot type.

```
options(width=100)
p <- p + geom_bar(stat="identity", fill = "steelblue")
```

```
print(p)
```



stat="identity" means that the values in the data.frame are the values to be plotted. The default is stat="count", in which case it will count the number of elements in the data.frame column. fill specifies the color of the bars. Later we'll see that fill can be used in another way to assign color to an attribute of the dataset.

4.4 Changing the look and feel of a plot using theme()

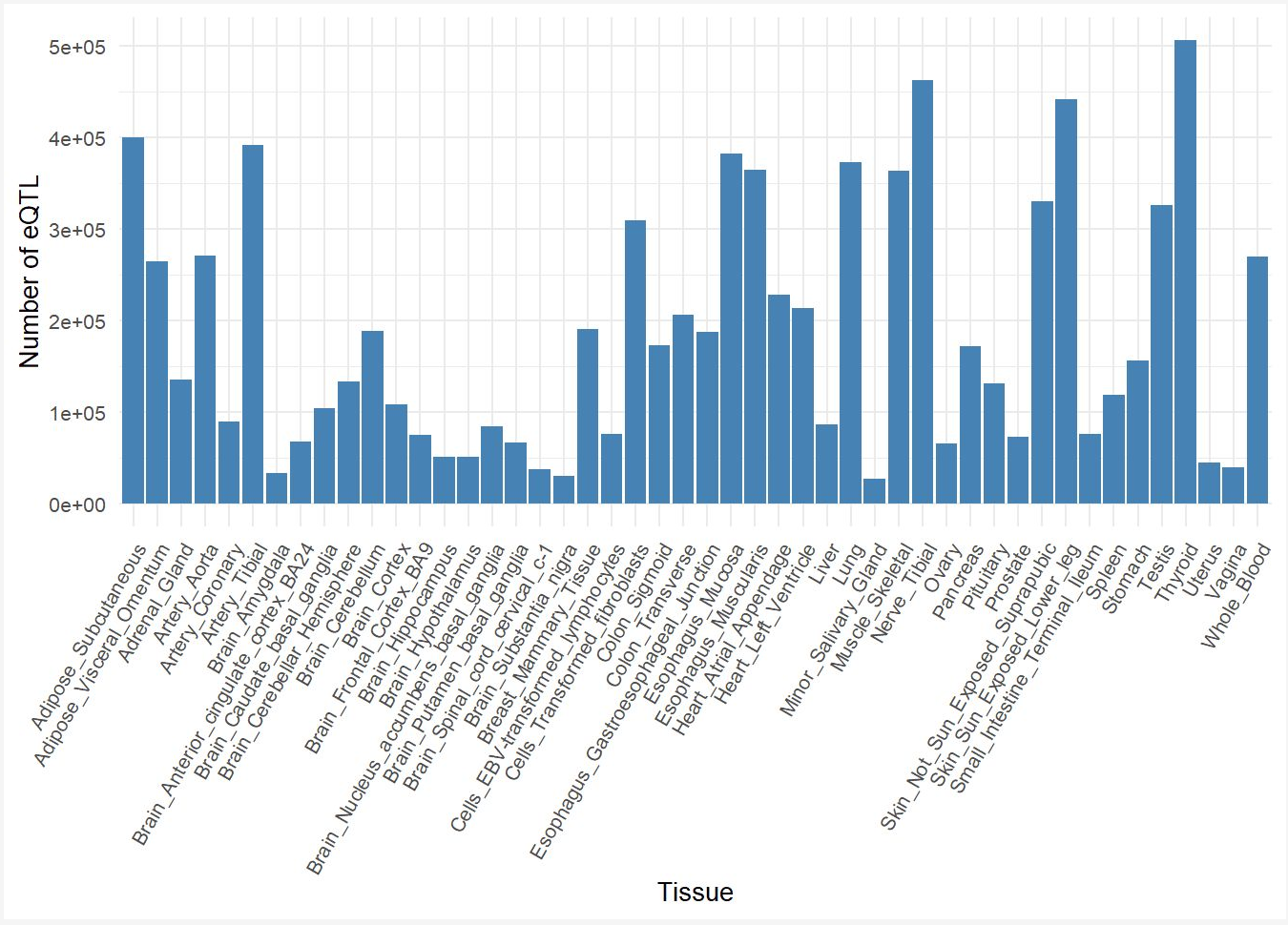
There are a few things that we can do to make this plot look nicer.

1. I find the grey background unattractive. There are predefined themes that provide alternatives. The one I use often is theme_minimal. Others to look at

include: theme_gray, theme_bw, theme_light, theme_dark, theme_classic, theme_void. Others have created packages with other themes. You can also make your own.

2. Angle the x-axis labels (tissues) and and adjust the font size so that the labels are legible. These are placed in the theme function.
3. Rename the x- and y-axis.

```
options(width=100)
p <- ggplot(data=neqtl, aes(x=tissue, y=neqtl)) +
  geom_bar(stat="identity", fill = "steelblue") +
  theme_minimal() +
  theme(text=element_text(size=10),
        axis.text.x=element_text(angle=60, hjust=1)) +
  ylab("Number of eQTL") +
  xlab("Tissue")
  print(p)
```



One nice thing about ggplot is that additional plotting commands can be added to the plot variable (p). This allows conditional statements to help determine what will be contained in the plot.

4.4.1 Sorting factor variables and overlaying plots

One challenge of plotting categorical variables is deciding how to order them on the x-axis in a meaningful or attractive manner. Here, because different tissues have different numbers of samples, it may be reasonable to organize the tissues by samples size. First, we will merge the eqtl number data with the sample size data.

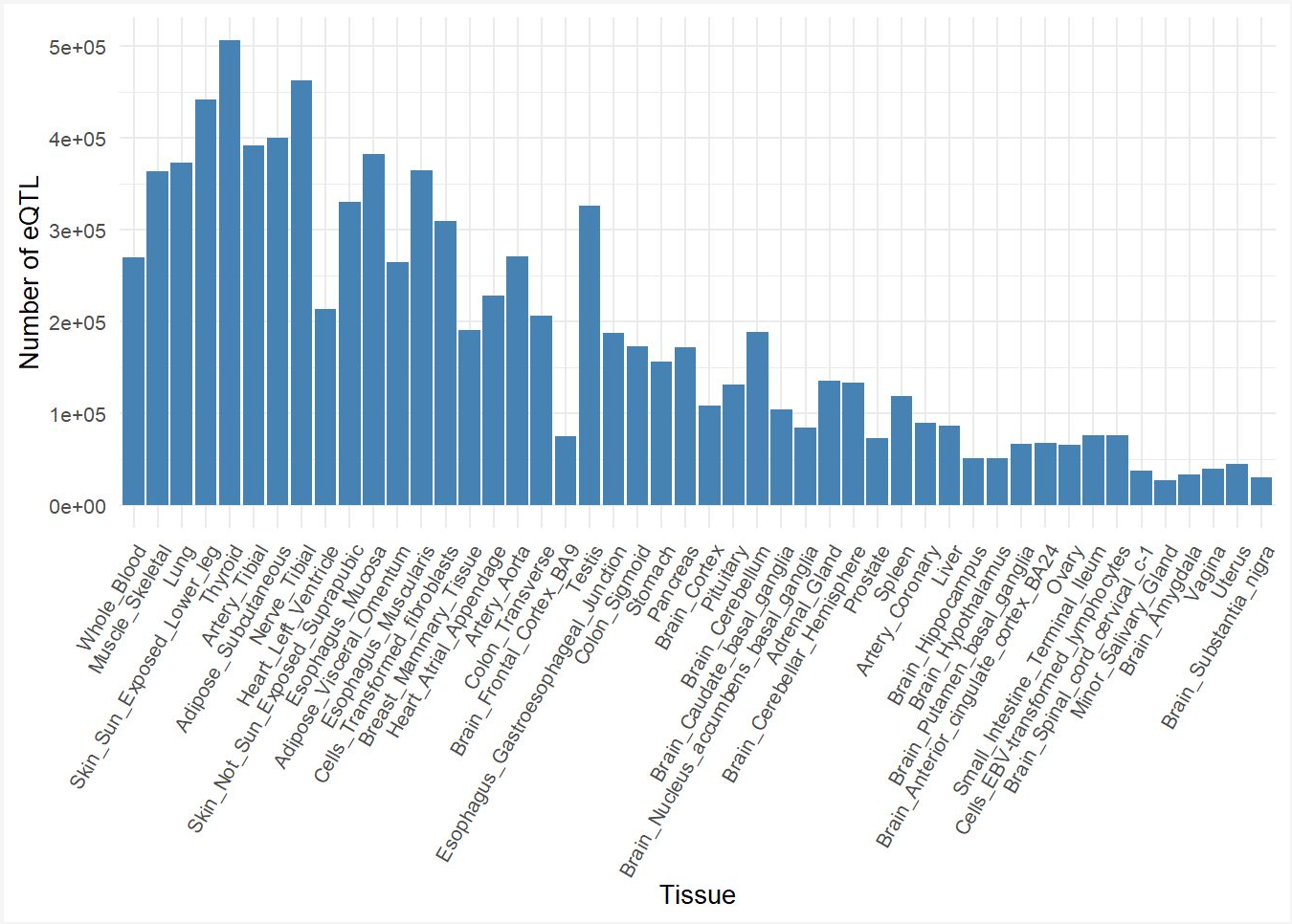
```
## merge neqtl and tissueN data.frames
neqtl <- left_join(neqtl, tissueN, by="tissue")
```

```
## Warning: Column `tissue` joining factors with different levels, coercing to
character vector
```

```
neqtl$tissue <- factor(neqtl$tissue,
                       levels = neqtl$tissue[order(neqtl$N, decreasing = T)])
```

Next we repeat the plot above.

```
options(width=100)
p <- ggplot(neqtl, aes(x=tissue, y=neqtl)) +
  geom_bar(stat="identity", fill = "steelblue") +
  theme_minimal() +
  theme(text=element_text(size=10),
        axis.text.x=element_text(angle=60, hjust=1)) +
  xlab("Tissue") +
  ylab("Number of eQTL")
print(p)
```



O.k. That's much better. But if we really want to know the relationship between sample size and the number of eQTL, then let's plot the number of samples of each tissue type on the same plot. To do this we have to do two things. First is to plot the sample size. We'll use geom_points to do this. The second is to scale the value of the number of samples, sincce the number of eQTL and number of samples is quite different.

```
range(tissueN$N)
```

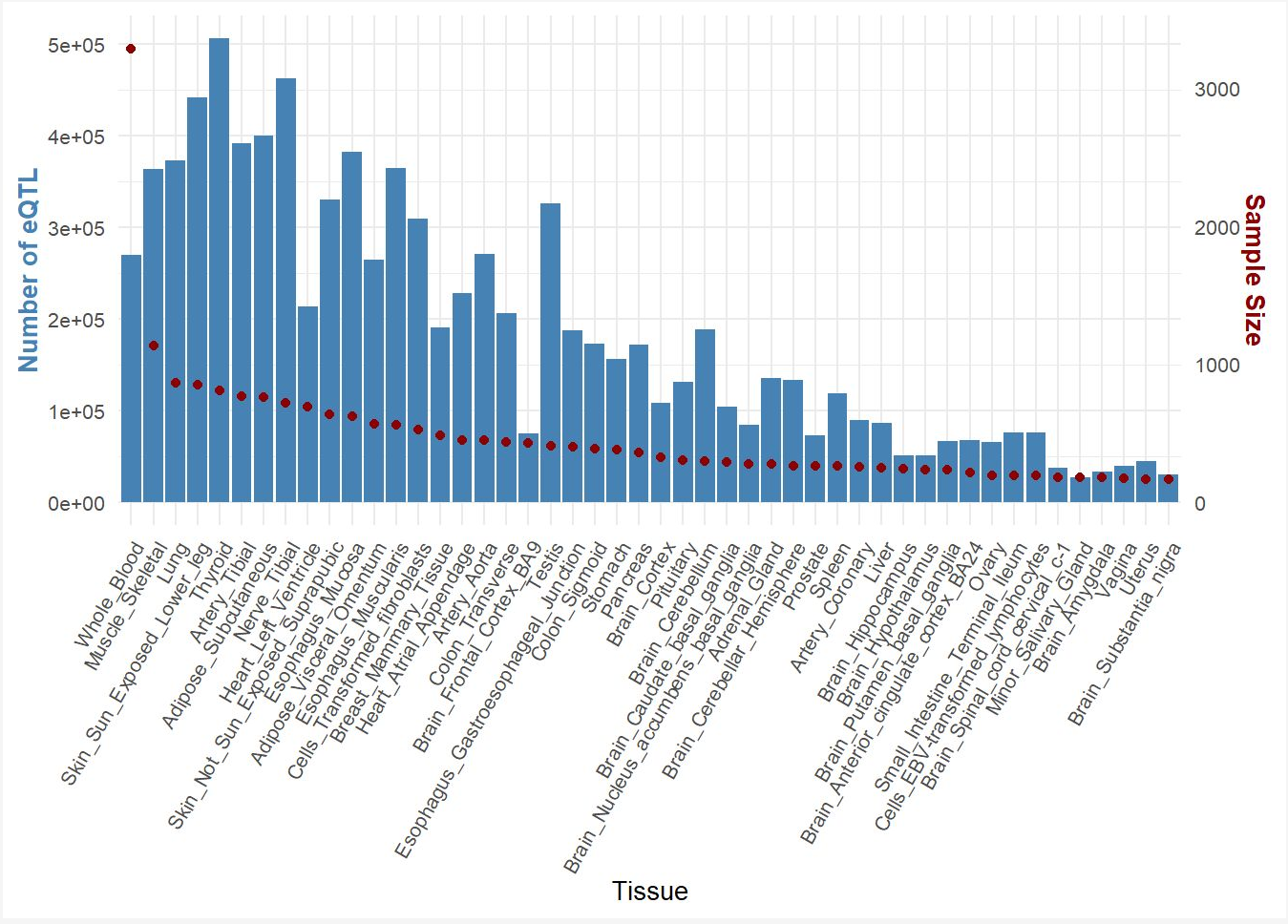
```
## [1]    4 3288
```

```
range(neqtl$neqtl)
```

```
## [1]  27146 505772
```

```
factorN <- 494913/3288
```

```
options(width=100)
p <- p + geom_point(aes(x=tissue, y=N*factorN), color="darkred") +
  scale_y_continuous(sec.axis = sec_axis(~./factorN, name = "Sample Size")) +
  theme(axis.title.y.right = element_text(color = "darkred", face="bold"),
        axis.title.y.left = element_text(color = "steelblue", face="bold"))
print(p)
```



4.5 Distributions

Now we'll consider the effect sizes of eQTL in order to see how to plot distributions (the relative frequency of the variable's values). Let's first use the histogram to look at the distribution of eQTL effects. To make things simpler, and to demonstrate how to compare distributions, we will focus on two tissues: Skeletal Muscle and Subcutaneous Adipose

4.5.1 Preparing the data for plotting

```
## Choose tissues of interest
TissSel <- c("Muscle_Skeletal", "Adipose_Subcutaneous")
```

```
## select only the tissues of interest (!! is a pretty handy trick to stick in your back pocket)
bs <- betas %>% select(!!TissSel)
## remove rows where variant is not an eqtl in either tissue
bs <- bs[rowSums(!is.na(bs)) > 0,]
bs <- as.data.table(bs)
##
bs <- melt(bs, measure=1:2, variable.name="tissue", value.name = "beta", na.rm=TRUE)
```

The function melt is very important for preparing data for ggplot. It transforms a data.frame from a wide rectangle to a long rectangle, whereby a subset of the column headers are converted to a column of values (specified by variable.name), and the values in the converted columns are converted to another column (specified by value.name). If there are columns that are not to be “longified” they are specified by id. We don’t need it above, but we’ll use it later. Below is what the data looks like before and after melting.

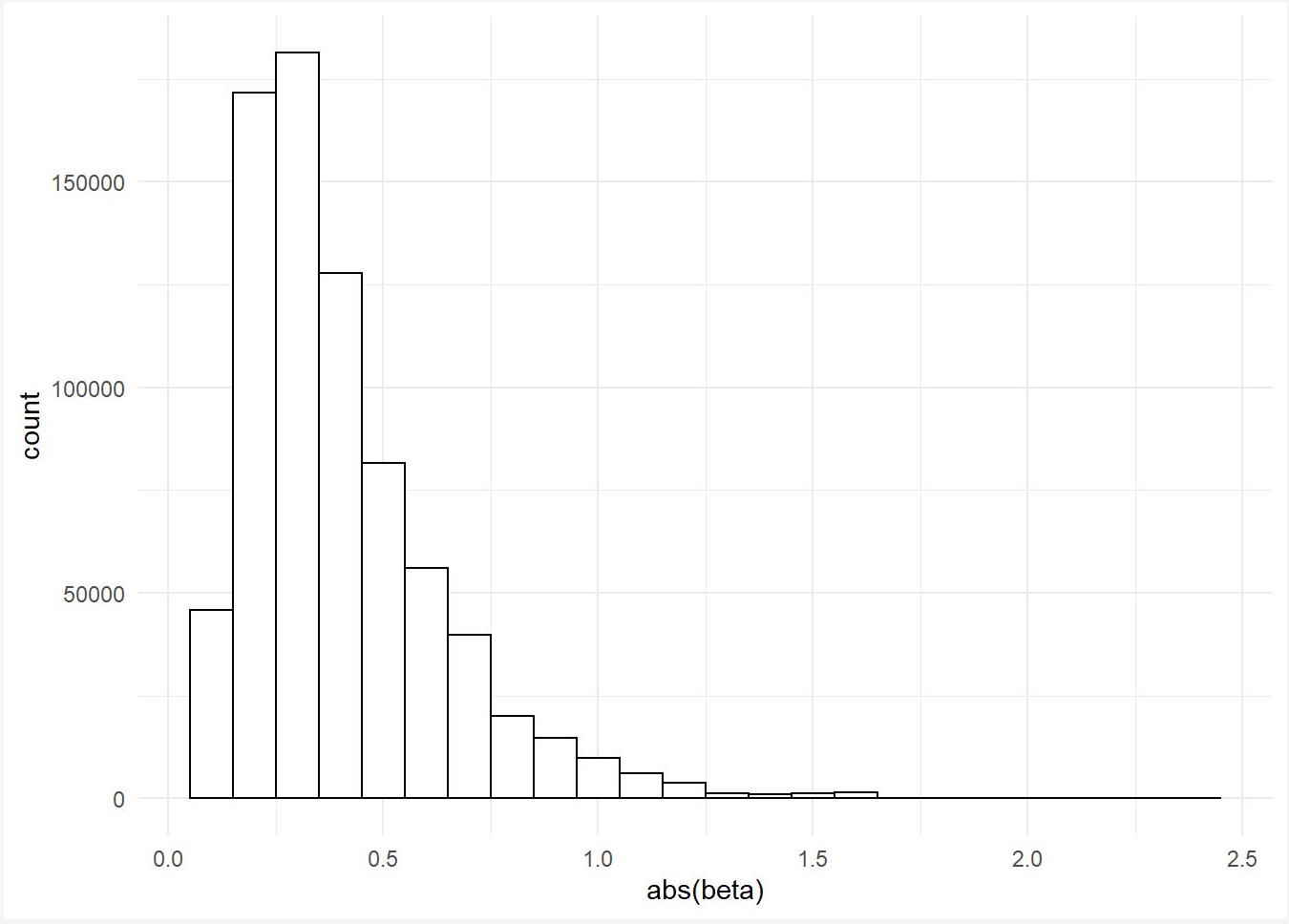
```
betas[1:4,TissSel]
```

```
bs[1:4,]
```

4.5.2 histogram function

A histogram of the distribution of all the values is created using geom_hist

```
options(width=100)
p <- ggplot(data=bs, aes(x = abs(beta))) +
  geom_histogram(binwidth=.1, colour="black", fill="white") +
  theme_minimal()

print(p)
```

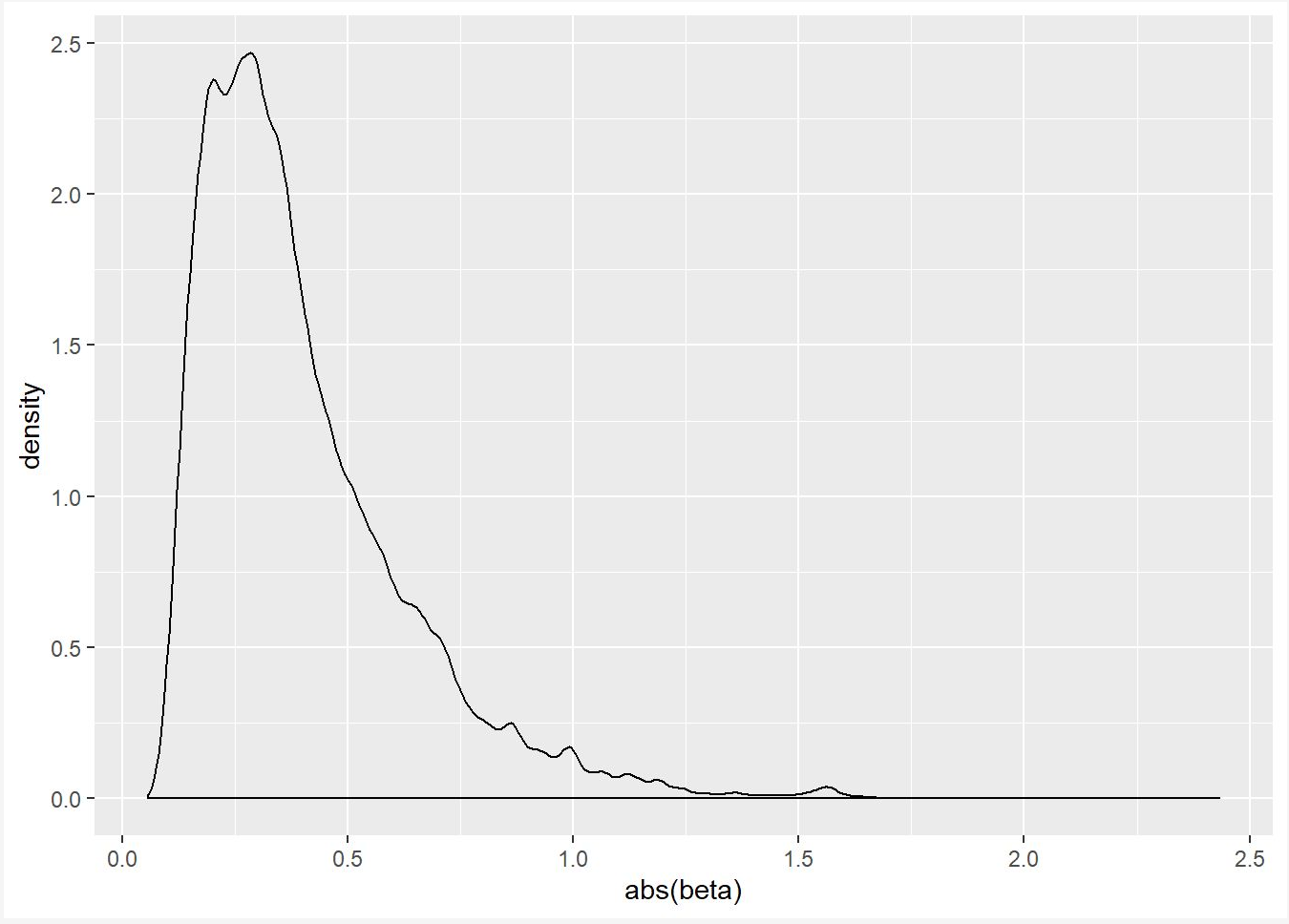


You can also specify the number of bins (bins=), the actualy breaks for the bins (breaks=), and if the intervals are left or right closed (closed=<left/right>)

4.5.3 Density function

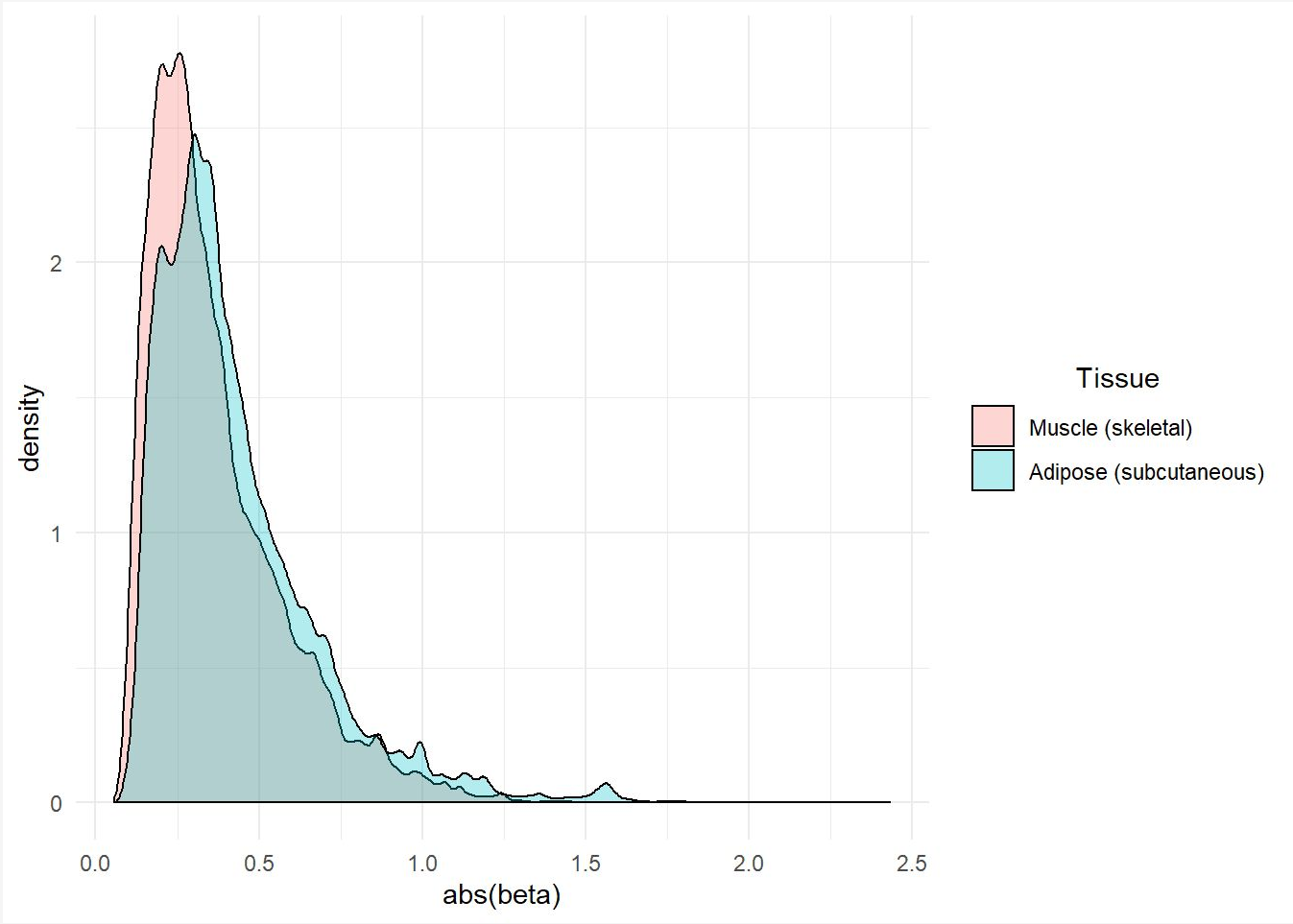
If you prefer density functions, you can use geom_density

```
p <- ggplot(bs, aes(x = abs(beta))) + geom_density(adjust = 1.15)
print(p)
```

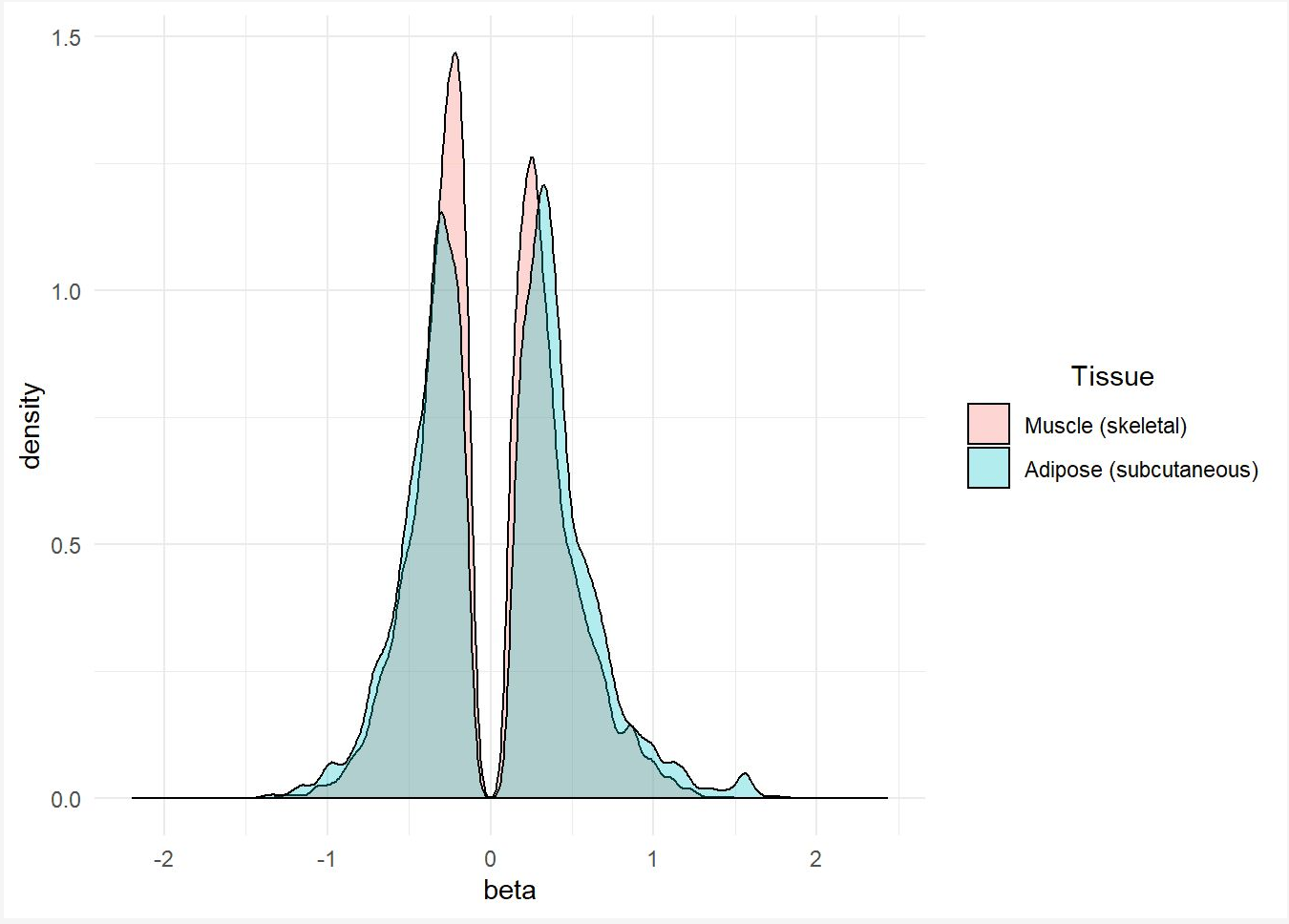


What we really wanted to do was compare the distribution of effect sizes for different tissue types. To do this, we will use the fill attribute of the density function that is available within the ggplot function (or the geom_density function)

```
options(width=100)
p <- ggplot(bs, aes(x=abs(beta), fill=tissue)) + geom_density(alpha=.3) +
  scale_fill_discrete(name = "Tissue",
                      labels=c("Muscle (skeletal)", "Adipose (subcutaneous)")) +
  theme_minimal() + theme(legend.title.align=0.5)
print(p)
```




```
p <- ggplot(bs, aes(x=beta, fill=tissue)) + geom_density(alpha=.3) +
  scale_fill_discrete(name = "Tissue",
                      labels=c("Muscle (skeletal)", "Adipose (subcutaneous)")) +
  theme_minimal() + theme(legend.title.align=0.5)
print(p)
```



4.6 Boxplots and violin plots

What if we wanted to compare more than two tissues. It's not hard to imagine that looking at more than 2 or 3 tissues via overlapping densities will start to get ugly and uninformative quickly. An alternative is to use a box and whisker plot. First we will select a subset of tissues and create an appropriately formatted data.frame. We will also summarize the eQTL effects by gene, by calculating the median eQTL effect for each gene.

4.6.1 Preparing data for box and violin plots

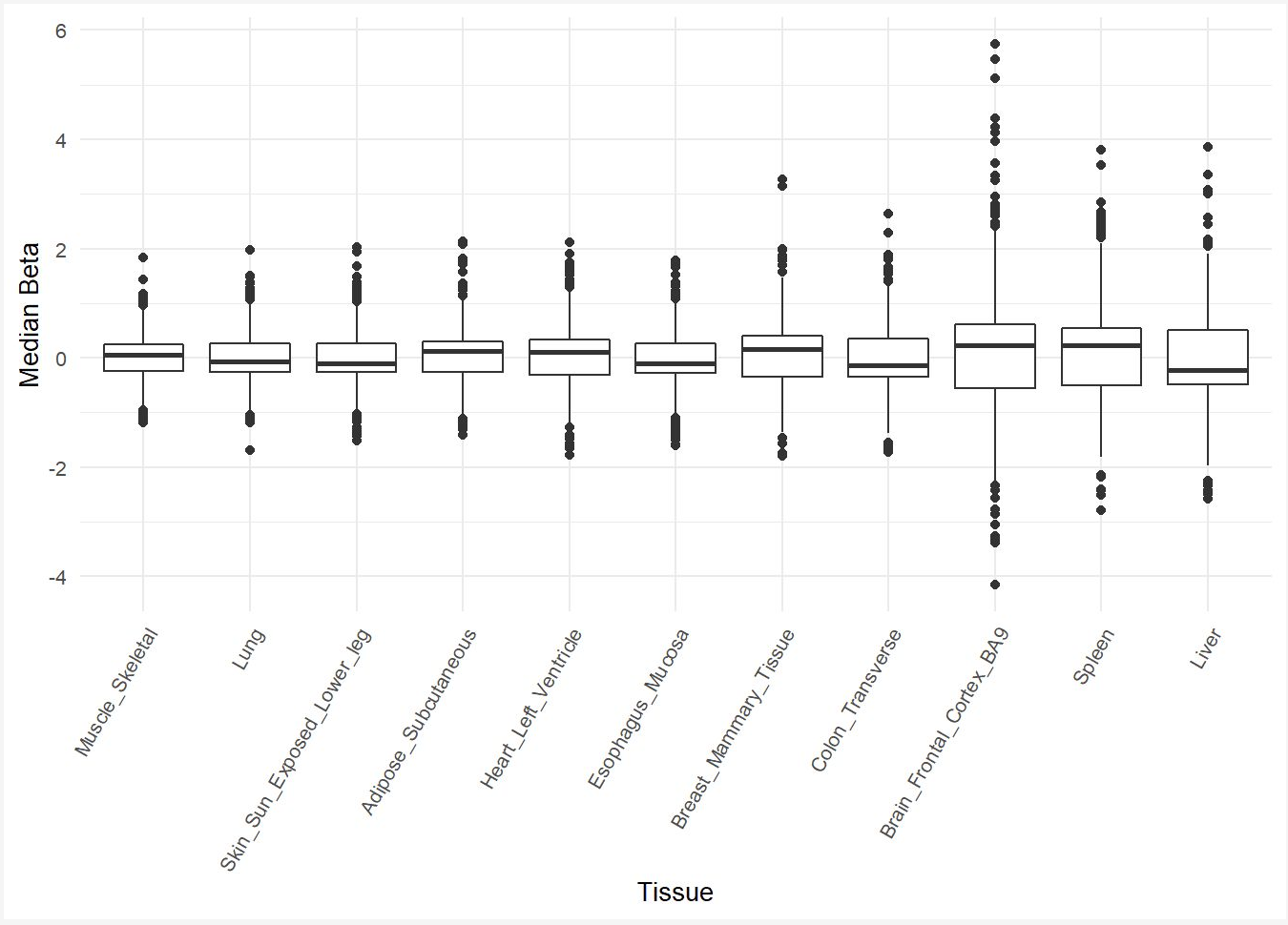
We again get our data in the correct format to plot. Below we will get the number of samples in each of the tissues of interest and order the factor levels in descreasing order. Also, instead of plotting the effect size of all eQTL, we will take the median effect size of all of the eQTLs for a gene in each tissue we consider. To do this we use the group_by function, which is an execptionally useful function.

```
tissSel <- c("symbol", "Adipose_Subcutaneous",
          "Brain_Frontal_Cortex_BA9",
          "Breast_Mammary_Tissue","Colon_Transverse", "Esophagus_Mucosa",
          "Heart_Left_Ventricle","Liver", "Lung", "Muscle_Skeletal",
          "Spleen", "Skin_Sun_Exposed_Lower_leg","Stomach")
## Get sample size information about the tissues in tissSel
tissueNabrv <- tissueN[tissueN$tissue %in% tissSel,]
## Get rid of factor levels that do not exist any more
tissueNabrv$tissue <- droplevels(tissueNabrv$tissue)
## Reorder the tissues levels so they are by samples size
tissueNabrv$tissue <- factor(tissueNabrv$tissue, levels =
                        tissueNabrv$tissue[order(tissueNabrv$N,
                                            decreasing=TRUE)])

## select columns of interest (!! again!)
bs <- betas %>% select(chr, !!tissSel) %>% as.data.table()
bs <- melt(bs, measure=3:length(tissSel), id = 1:2,
         variable.name="tissue", value.name="beta")
## Calcuate the mean effect size of an eQTL for each gene within tissue
bs <- bs %>% group_by(symbol, tissue) %>%
  mutate(m.beta=median(beta, na.rm=T)) %>% filter(row_number() == 1)
%>%
  select(-beta) %>% ungroup()
## Remove rows that contain NAs and reorder the factor levels
bs <- bs %>% filter(!is.na(m.beta)) %>%
  mutate(tissue = factor(tissue, levels = levels(tissueNabrv$tissue)))
```

Now we will plot these gene-based median eQTL effect sizes across tissues.

```
options(width=120)
p <- ggplot(bs, aes(x = tissue, y = m.beta)) +
  geom_boxplot() + theme_minimal() +
  theme(text=element_text(size=10),
        axis.text.x=element_text(angle=60, hjust=1)) +
  xlab("Tissue") +
```

```
  ylab("Median Beta")

print(p)
```

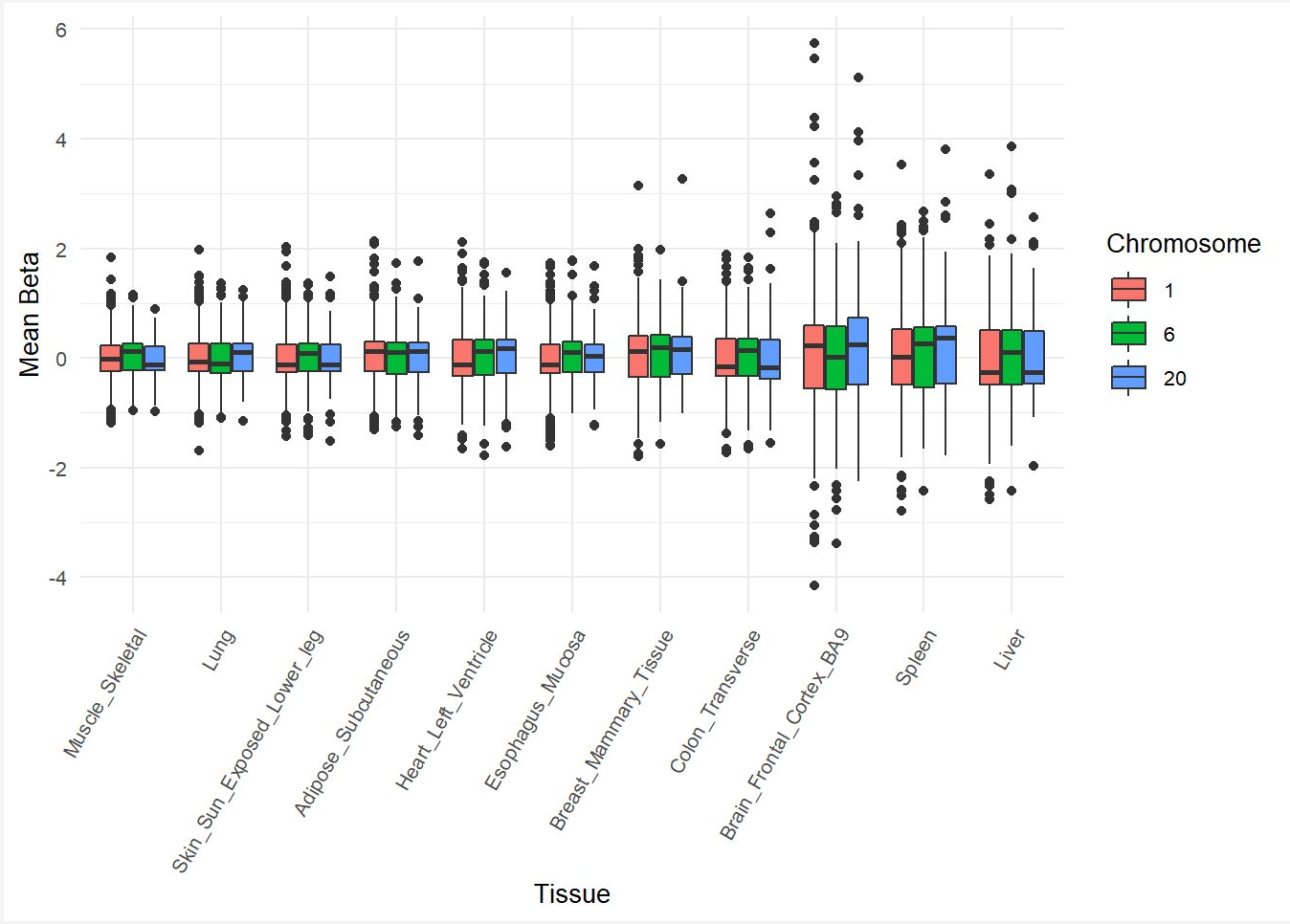


4.6.2 Using additional attributes to communicate additional information

Now, what if we wanted to know if the effect size distribution differs by chromosomal location. To do this, we can use another aesthetic attribute. In this case, we'll use the fill attribute.

```
bs$chr <- factor(bs$chr, levels = c("1","6","20"))
options(width=120)
p <- ggplot(bs, aes(x = tissue, y = m.beta, fill = chr)) +
  geom_boxplot() + theme_minimal() +
  theme(text=element_text(size=10),
```

```
      axis.text.x=element_text(angle=60, hjust=1)) +
  xlab("Tissue") +
  ylab("Mean Beta") +
  scale_fill_discrete(name = "Chromosome")

print(p)
```

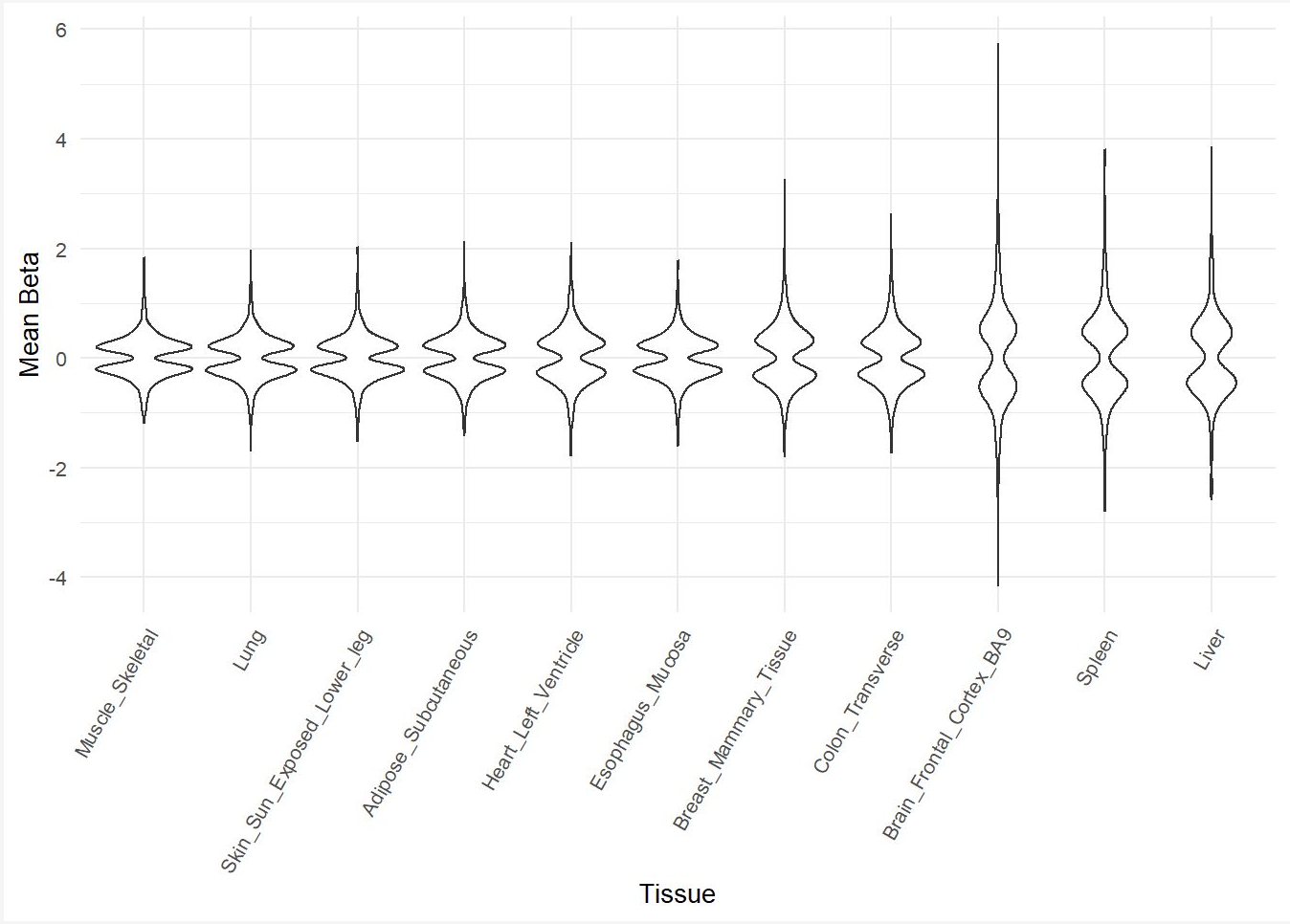


4.6.3 The violin plot

The barplot gives us valuable information about the distribution, but is not as informative as the full distribution. The violin plot is a way to better visualize the full distribution. Not surprisingly, the plot function is geom_violin.

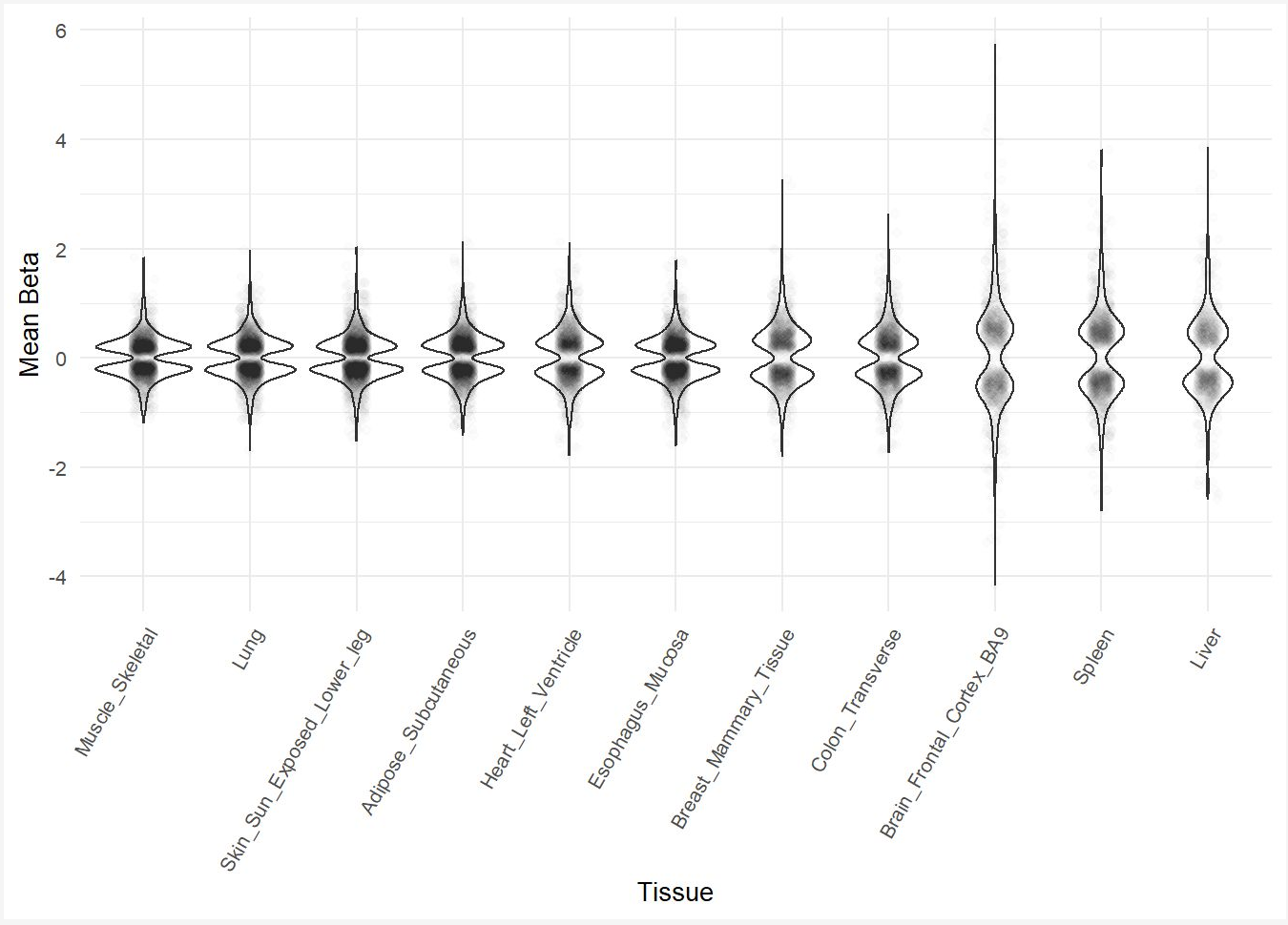
```
p <- ggplot(bs, aes(x = tissue, y = m.beta)) +
  geom_violin() + theme_minimal() +
```

```
  theme(text=element_text(size=10),
        axis.text.x=element_text(angle=60, hjust=1)) +
  xlab("Tissue") +
  ylab("Mean Beta")

print(p)
```



It is not uncommon to plot the points on top of the violins. Our data happens to have a very large number of points. In order to “thin” out the data, we can use the alpha parameter in the geom_jitter function.

```
p <- p + geom_jitter(width = 0.1, alpha=.01)
print(p)
```



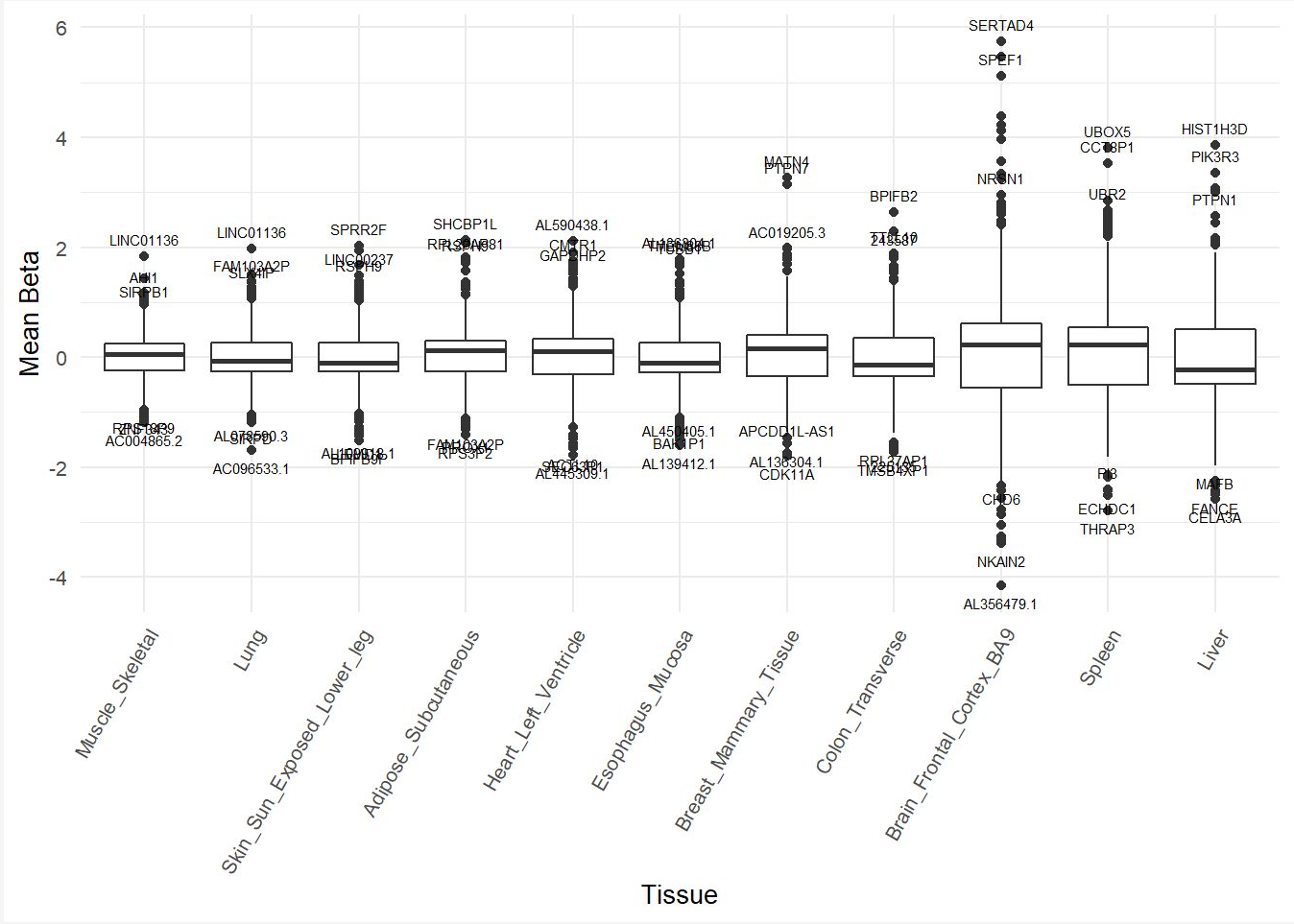
4.7 Adding text to plots

ggplot makes adding labels to plots pretty easy. As an example, let's label the genes that have the larges and smallest value in each tissue on each chromosome. First, it's useful to identify the minimum and maximum value for each tissue/chromosome.

```
bs <- bs %>% group_by(tissue, chr) %>%
  mutate(min.lab = ifelse(m.beta == min(m.beta), gsub("ENSG0+","", symbol),
"""),
        max.lab = ifelse(m.beta == max(m.beta), gsub("ENSG0+","", symbol), """),
        lab = ifelse(m.beta %in% c(max(m.beta), min(m.beta)),
                                gsub("ENSG0+","", symbol), ""))
```

Now we use geom_text.

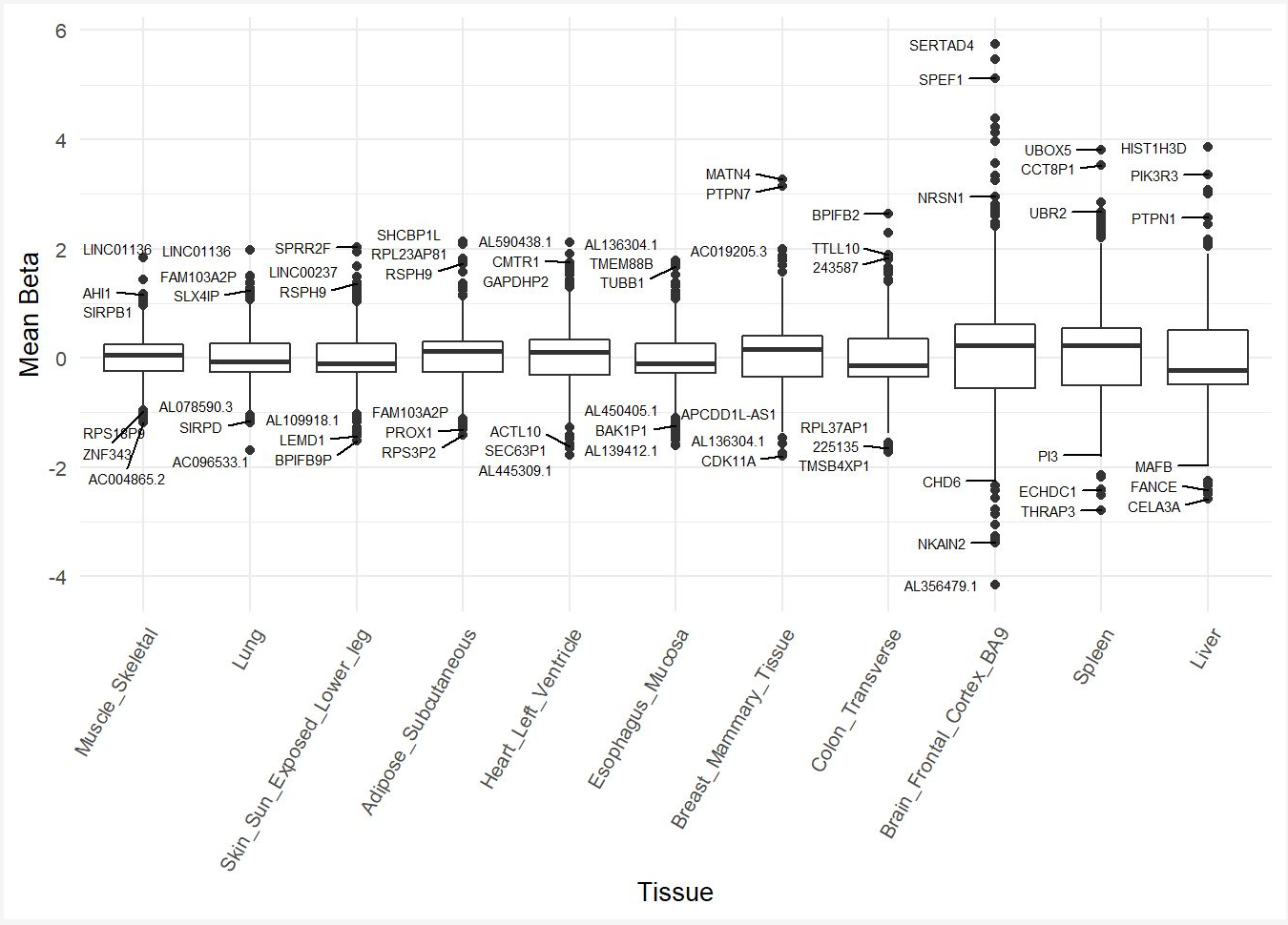
```
options(width=120)
p <- ggplot(bs, aes(x = tissue, y = m.beta)) +
  geom_boxplot() +
  geom_text(aes(label = max.lab), size = 2, angle=0, vjust=-1) +
  geom_text(aes(label = min.lab), size = 2, angle=0, vjust=2) +
  theme_minimal() +
  theme(text=element_text(size=10),
        axis.text.x=element_text(angle=60, hjust=1)) +
  xlab("Tissue") +
  ylab("Mean Beta")

print(p)
```



4.7.1 repel for prettier labels

When points are this close together, it's really trick to get the labels to look nice. There is a fantastic package called repel which can help. Here's how you use it.

```
options(width=120)
p <- ggplot(bs, aes(x = tissue, y = m.beta)) +
  geom_boxplot() +
  ## aes can be used within the plotting functions themselves
  geom_text_repel(aes(label = max.lab), size = 2, box.padding = .1,
                  nudge_x=-.5, angle=0) +
   geom_text_repel(aes(label = min.lab), size = 2, box.padding = .1,
                  nudge_x=-.5, angle=0) +
  theme_minimal() +
  theme(text=element_text(size=10),
        axis.text.x=element_text(angle=60, hjust=1)) +
  xlab("Tissue") +
  ylab("Mean Beta")

print(p)
```

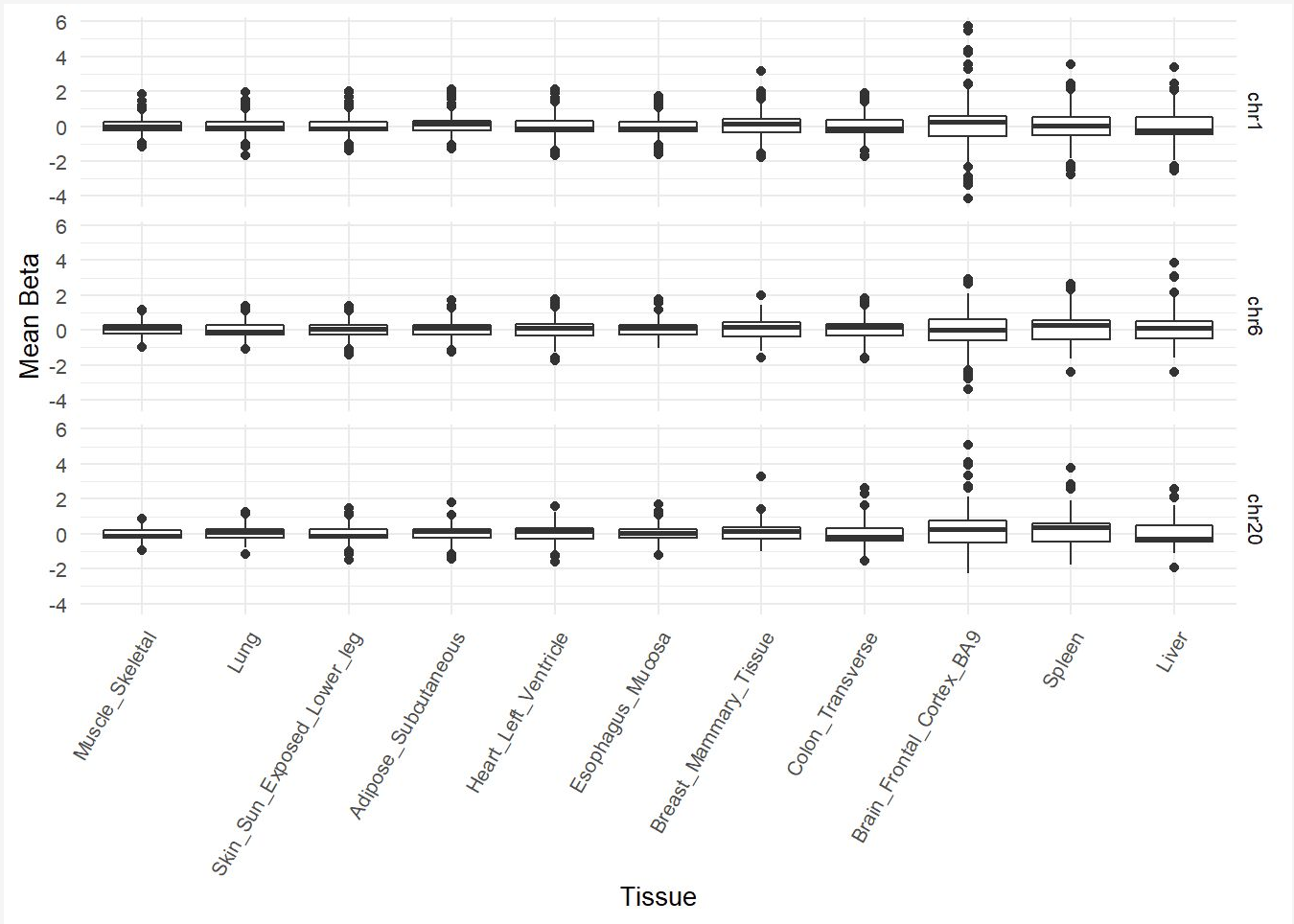


4.8 facet_grid: Creating a matrix of plots to simplify plotting multiple variables

Recall our boxplot that printed the distribution of betas for chromosomes 1, 6 and 20 next to each other. As you can image, if we labelled gene names on that plot it would be a mess. For this reason, and others, sometimes it is better to repeat a simpler plot multiple times as a function of an addition variable (e.g. chromosome) in an organized way. One way to do this is to use facet_grid. First, in order to make the plot look a little nicer we will add “chr” to the front of the chromosome number.

```
bs$chr <- factor(paste0("chr", bs$chr), levels = c("chr1","chr6","chr20"))

options(width=120)
p <- ggplot(bs, aes(x = tissue, y = m.beta)) +
 geom_boxplot() +
 ## the y ~ x is an equation like arguement that shows how to organize the
```

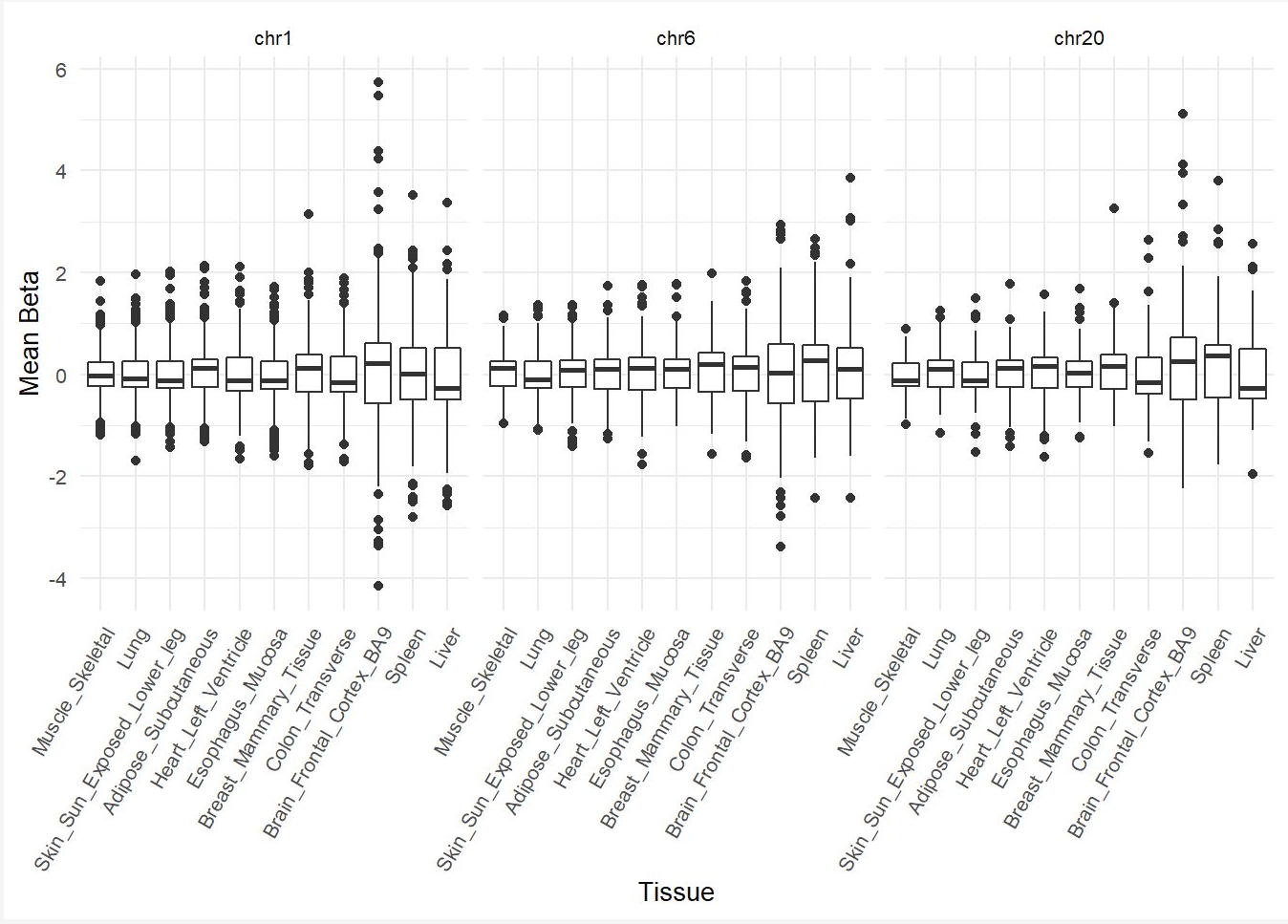
```
variables used to create
  ## the grid of plots. Variables on the y side are organized a rows and on the x
side as columns.
  facet_grid(chr~.) +
  theme_minimal() +
  theme(text=element_text(size=10),
        axis.text.x=element_text(angle=60, hjust=1)) +
  xlab("Tissue") +
  ylab("Mean Beta")

print(p)
```



If you prefer have chromosome along the vertical axis, just change the chr~. to .~chr

```
options(width=120)
p <- ggplot(bs, aes(x = tissue, y = m.beta)) +
  geom_boxplot() +
```

```
  facet_grid(.~chr) +
  theme_minimal() +
  theme(text=element_text(size=10),
        axis.text.x=element_text(angle=60, hjust=1)) +
  xlab("Tissue") +
  ylab("Mean Beta")

print(p)
```



5 Plotting similarity using heatmaps

Next we are going to explore heatmaps, which are useful for comparing the pairwise correlation or other similarity measures between variables. For our data, we will calculate the proportion of eQTL shared between tissues. We can do this using the ps

data.frame which has tissues as columns, variant-gene combinations as rows, and each cell contains a p-value is the variant-gene is an eQTL for the tissue and NA otherwise. We will turn these into proportion of shared eQTL between tissues in the following way:

5.1 Creating data for plotting a heatmap

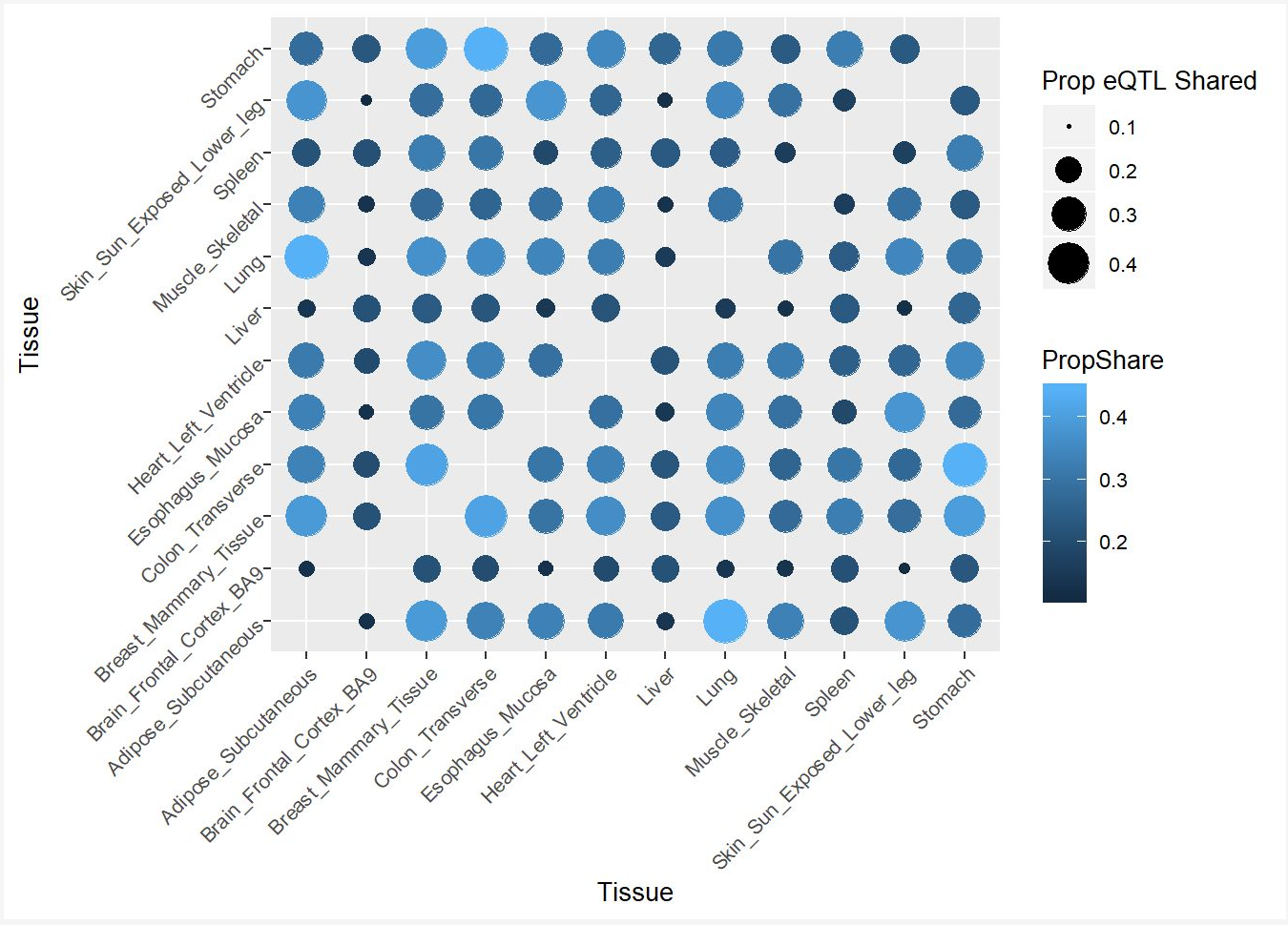
```
tissSel <- c("Adipose_Subcutaneous",
          "Brain_Frontal_Cortex_BA9",
          "Breast_Mammary_Tissue","Colon_Transverse", "Esophagus_Mucosa",
          "Heart_Left_Ventricle","Liver", "Lung", "Muscle_Skeletal",
          "Spleen", "Skin_Sun_Exposed_Lower_leg","Stomach")
tissim <- ps[, tissSel]
## Only keep snp-genes pairs that are eQTLs for at least one tissue
tissim <- tissim[rowSums(!is.na(tissim)) > 0 , ]
## Convert from p-value to 1/0 eQTL yes/no
tissim <- 1*(is.na(tissim) == FALSE)
tissim <- Matrix(tissim, sparse=T)

## Create a matrix that counts the number of eQTL-gene shared between tissues
tisNshare <- t(tissim) %*% tissim
## Determine the total number of eQTL-tissues that were eQTL for at least one of
the two tissues
tisNeither<- nrow(tissim) - t(tissim == 0) %*% (tissim == 0)
## Determine the proportion of eQTL shared between tissues
tisShareProp <- tisNshare/tisNeither
tisShareProp <- as.matrix(tisShareProp)
tisShareProp <- as.data.table(tisShareProp, stringAsFactor=FALSE)
tisShareProp$Tissue1 <- names(tisShareProp)

toplot <- melt(tisShareProp, id = "Tissue1",  variable.name="Tissue2",
value.name = "PropShare")
toplot <- toplot %>% mutate(Tissue1 = factor(Tissue1,
levels=names(tisShareProp)),
                      Tissue2 = factor(Tissue2, levels=names(tisShareProp)))
toplot <- toplot %>% filter(Tissue1 != Tissue2)

## toplot looks like this
```

```
toplot[1:5,]
```

O.k. Now that we've calculated the proportion of eQTL that tissue pairs share in common, we can plot them in a number of ways. One simple way is to use a scatter plot.

5.2 Using geom_point to communicate similarity.

```
p <- ggplot(toplot, aes(x=Tissue1, y=Tissue2)) +
  geom_point(aes(size = PropShare, color = PropShare)) +
  scale_size_continuous(name = "Prop eQTL Shared",
                        range=c(.5,8),
                        limits=c(.1,.45)) +
  theme(axis.text.x = element_text(angle = 45),
        text=element_text(size=10, hjust=1),
        axis.text.y = element_text(angle = 45),
        axis.title = element_text(hjust=.5)) +
  xlab("Tissue") + ylab("Tissue")

  ## COLOR LABELS TO HELP IDENTIFY WHO IS WHO ##
  #theme(axis.text.x = element_text(angle = 90, color = STclr)) +
  #scale_color_manual(values=clrLvl)
print(p)
```

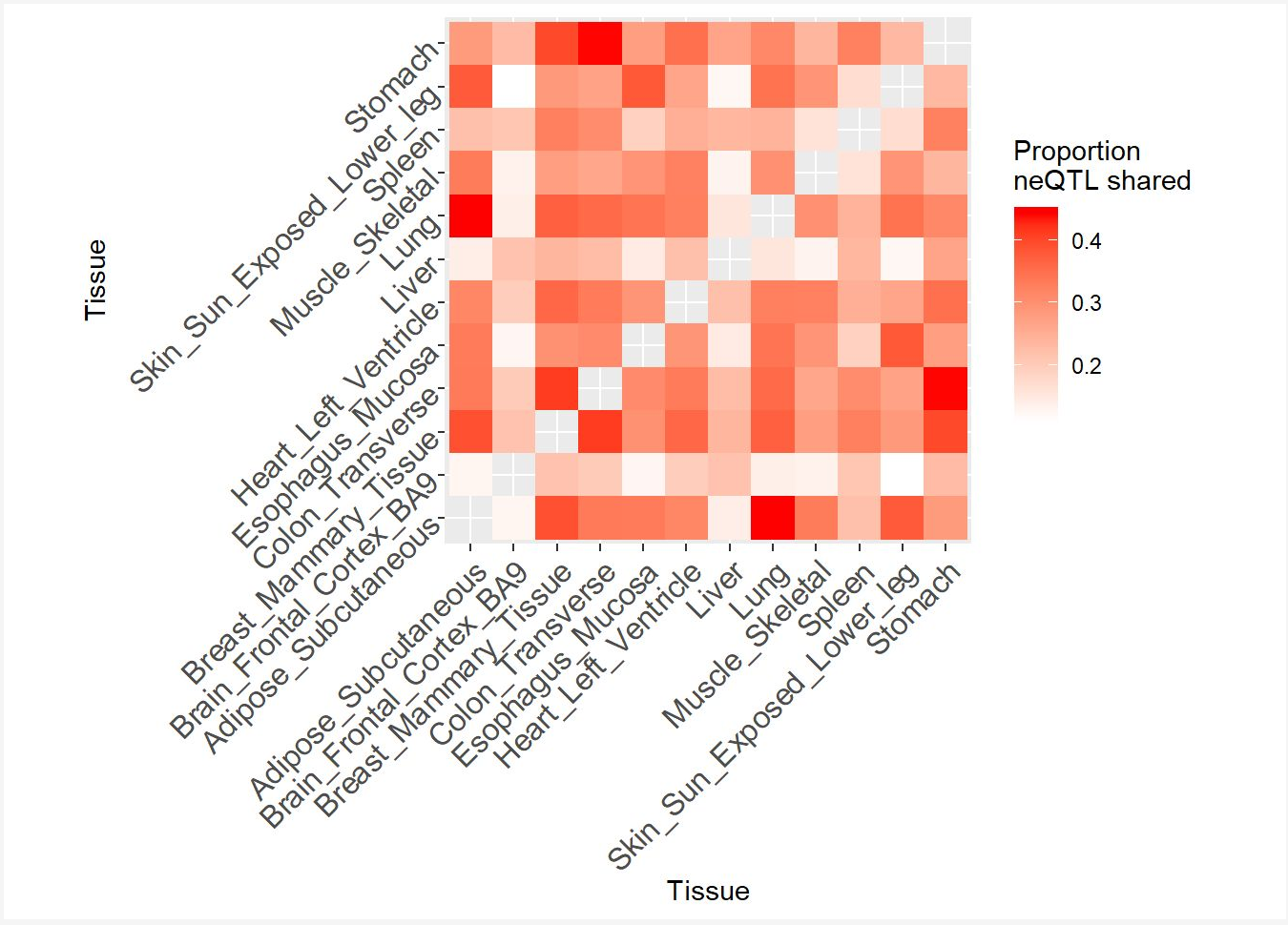


5.3 Using geom_tile to communicate similary.

A more common alternative for plotting similarity data is a heatmap. The way to make a heatmap using ggplot is using geom_tile, although many folks prefer alternative heatmapping functions. We'll explore one of them shortly. geom_tile works very much like all other geom type functions. . .

```
p <- ggplot(toplot, aes(x=Tissue1, y=Tissue2)) +
 geom_tile(aes(fill = PropShare)) +
  theme(axis.text.x = element_text(angle = 45),
      axis.text=element_text(size=12, hjust=1),
      axis.text.y = element_text(angle = 45),
      axis.title = element_text(hjust=.5)) +
 coord_equal() +
 scale_fill_gradient(name = "Proportion\nneQTL shared",
```

```
                low = "white", high = "red") +
 xlab("Tissue") + ylab("Tissue")
print(p)
```

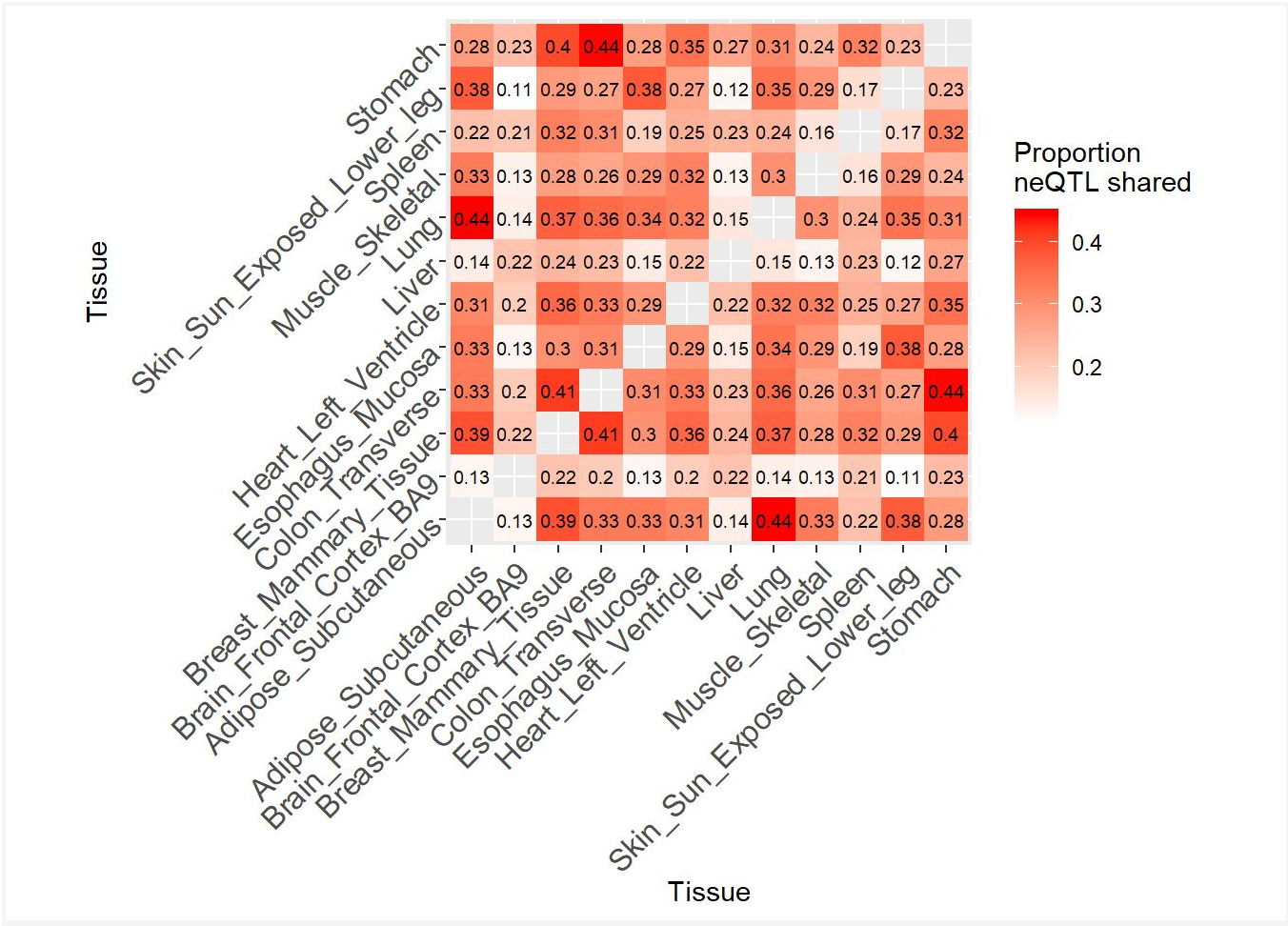


5.3.1 Adding values to cells (geom_text)

Sometimes it's nice to add the actual value in the cell (provided the matrix isn't too large)

```
p <- p <- ggplot(toplot, aes(x=Tissue1, y=Tissue2)) +
 geom_tile(aes(fill = PropShare)) +
 theme(axis.text.x = element_text(angle = 45),
       axis.text=element_text(size=12, hjust=1),
       axis.text.y = element_text(angle = 45),
       axis.title = element_text(hjust=.5)) +
```

```
  coord_equal() +
  scale_fill_gradient(name = "Proportion\nneQTL shared",
                      low = "white", high = "red") +
  xlab("Tissue") + ylab("Tissue") +
  ## add text to cells
  geom_text(aes(label = as.character(round(PropShare, 2))),
            size = 2.5)

print(p)
```



5.4 Organizing rows and columns by similarity

The other common need when plotting similarities is to organize the columns and rows so as to show patterns, which makes interpreting and communicating your results

easier. Some heatmap function in R will do this for you. With a couple extra steps you can do it with geom_tile too. We will employ the hierarchical clustering function,hclust, to do this. First we reshape the data, which we would have to do for any other heatmap function.

```
tp2 <- reshape2::dcast(toplot, Tissue1 ~ Tissue2,
                       value.var = "PropShare")
 rownames(tp2) <- tp2$Tissue1
tp2 <- tp2 %>% select(-Tissue1) %>% as.matrix()
tp2[1:4,1:4]

##                          Adipose_Subcutaneous Brain_Frontal_Cortex_BA9
Breast_Mammary_Tissue Colon_Transverse
## Adipose_Subcutaneous                       NA             0.1268428
0.3908484       0.3338619
## Brain_Frontal_Cortex_BA9          0.1268428                    NA
0.2155646       0.2012800
## Breast_Mammary_Tissue             0.3908484             0.2155646
NA       0.4123290
## Colon_Transverse                  0.3338619             0.2012800
0.4123290            NA
```

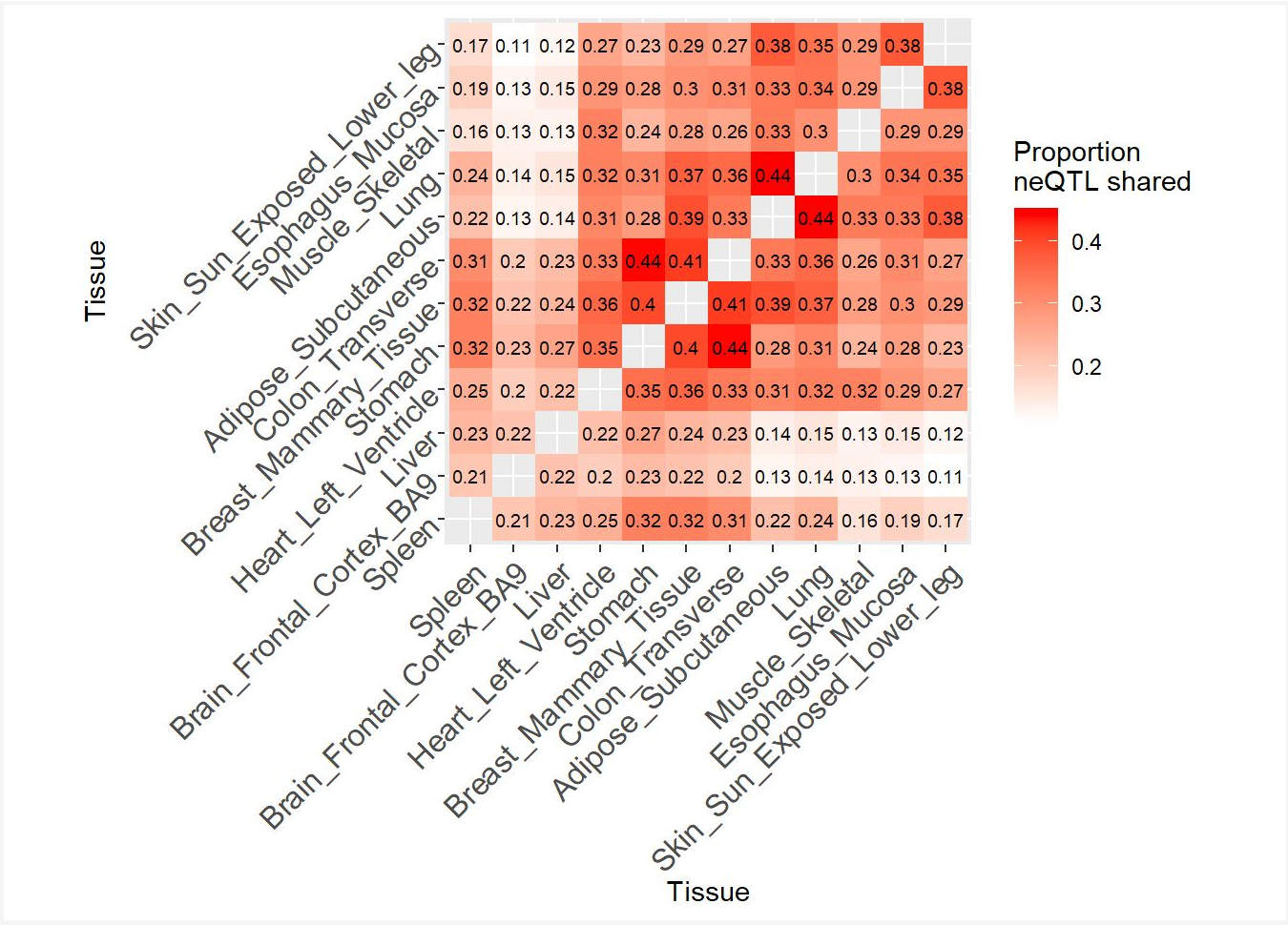
Next we cluster the data and determine the ordering of the rows/columns, and then reorder the tissue variables.

```
 ord <- hclust( dist(tp2, method = "euclidean"), method = "ward.D" )$order

 toplot$Tissue1 <- factor(toplot$Tissue1, levels=colnames(tp2)[ord])
 toplot$Tissue2 <- factor(toplot$Tissue2, levels=colnames(tp2)[ord])
```

Now we perform the geom_tile identically as before, except with the reordered columns.

```
p <- p <- ggplot(toplot, aes(x=Tissue1, y=Tissue2)) +
 geom_tile(aes(fill = PropShare)) +
  theme(axis.text.x = element_text(angle = 45),
      axis.text=element_text(size=12, hjust=1),
      axis.text.y = element_text(angle = 45),
      axis.title = element_text(hjust=.5)) +
 coord_equal() +
 scale_fill_gradient(name = "Proportion\nneQTL shared",
                     low = "white", high = "red") +
```

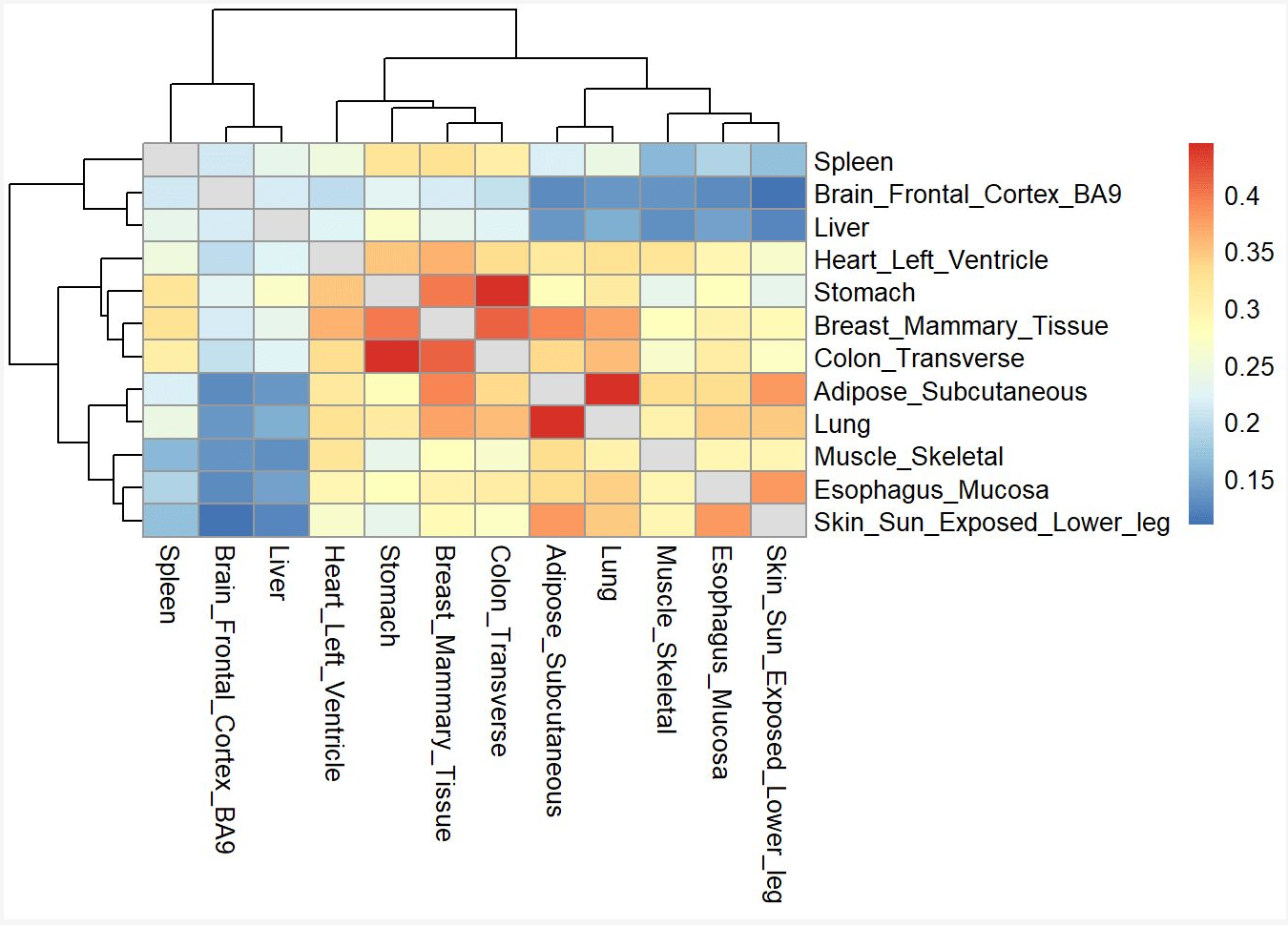
```
  xlab("Tissue") + ylab("Tissue") +
  geom_text(aes(label = as.character(round(PropShare, 2))),
            size = 2.5)

print(p)
```



5.5 Using pheatmap to plot similarity data

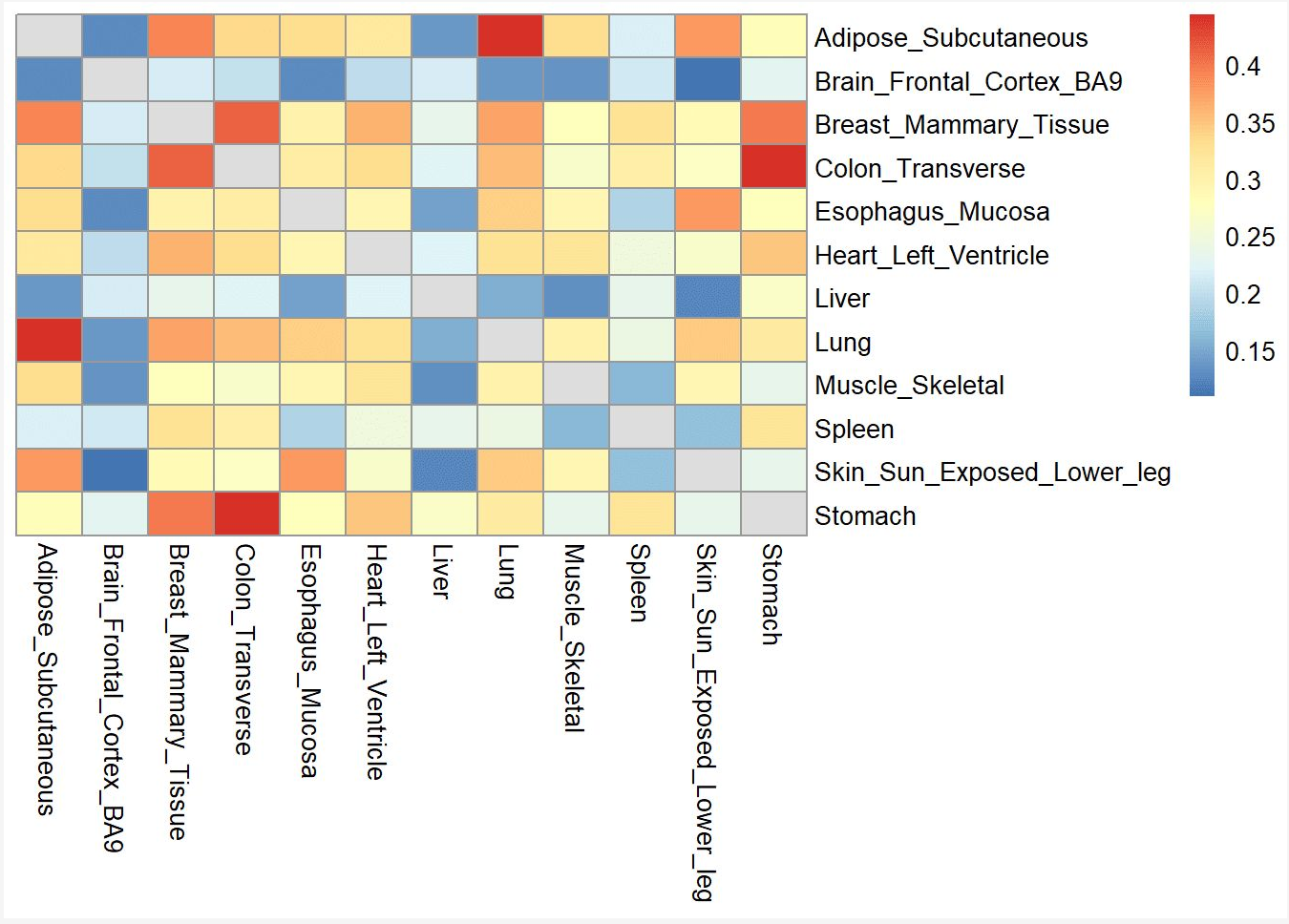
One of the heamap functions that will do the clustering automatically is pheatmap. heatmap.2 is another, which is within the gplots package. pheatmap uses hclust internally and allows you to change hclust arguments inside the pheatmap function.

```
pheatmap(tp2)
```



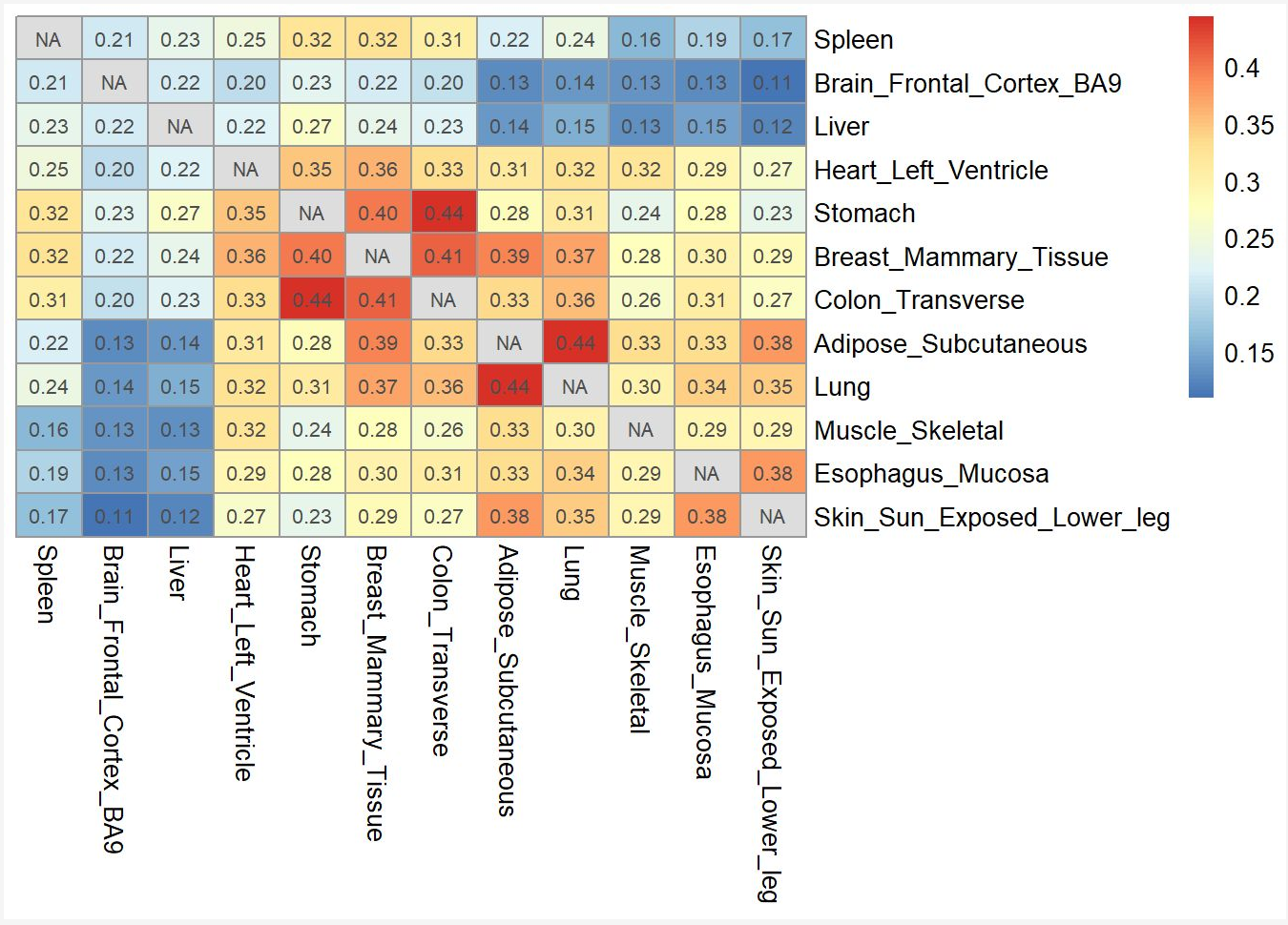
While the cladogram is informative, this might be a more attractive plot without it.

```
pheatmap(tp2, cluster_rows=FALSE, cluster_cols=FALSE)
```



Unfortunately, when you don't cluster the rows or columns pheatmap not only omits the cladogram, but it omits clustering altogether. We can apply the same trick we did previously to reorder the rows and columns, however this time we have to literally reorder the rows and columns. pheatmap also makes it very easy to add the value being plotted into the figure.

```
ord <- hclust( dist(tp2, method = "euclidean"), method = "ward.D" )$order
tp2 <- tp2[ord, ord]
pheatmap(tp2, cluster_rows=FALSE, cluster_cols=FALSE,
display_numbers=TRUE)
```



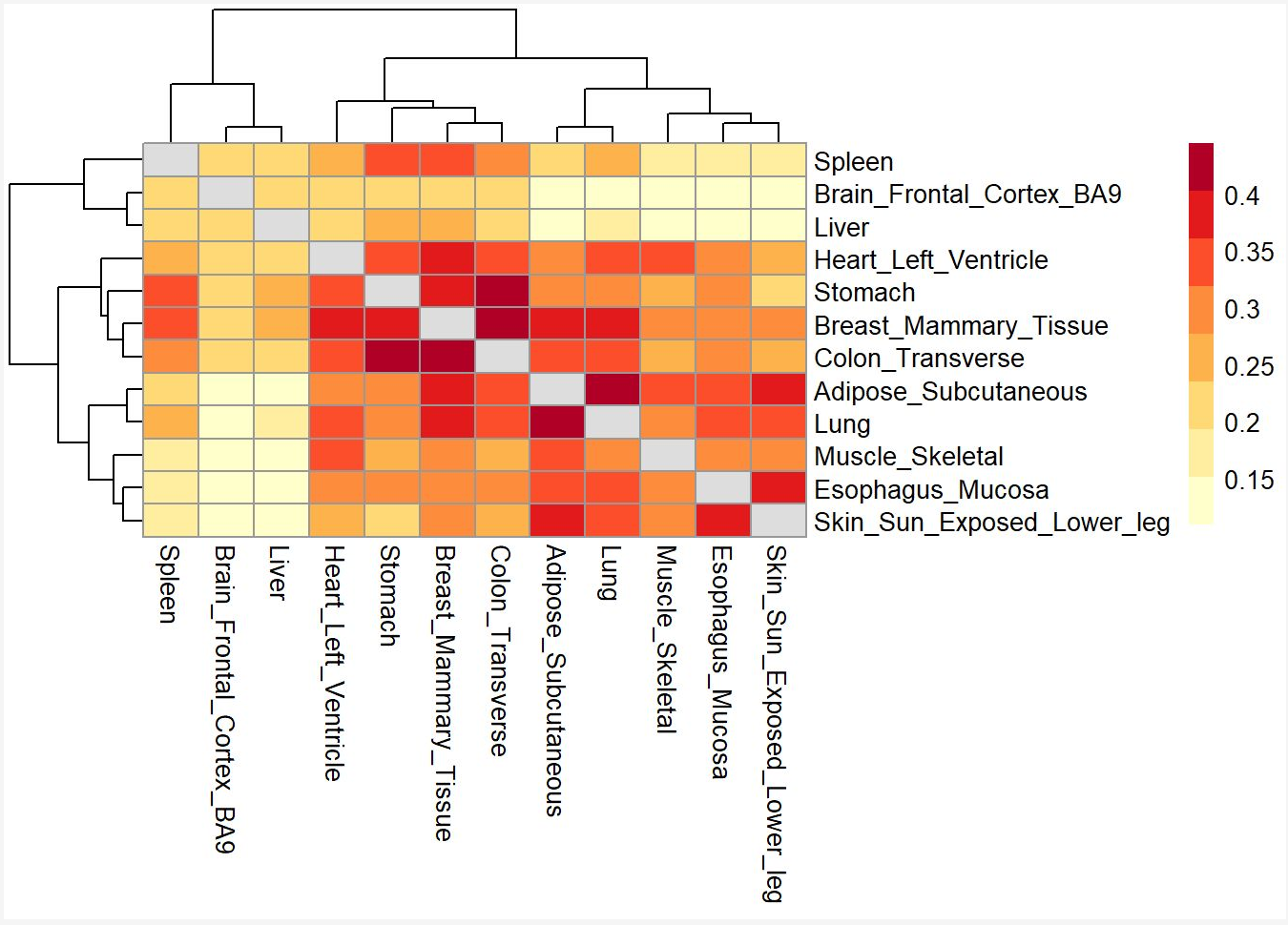
6 Changing colors in heatmaps (and other plots)

Having the correct color scale can make a big difference in the effectiveness of a heatmap, and interestingly, the best colors palettes for a dataset will often differ as a function of the application. RColorBrewer is a very popular package from which to select predefined color palettes. To see what pallets are available, you can. . .

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

To use a palette while plotting, we can do the following. . .

```
pheatmap(tp2, color=brewer.pal(n=8, name="YlOrRd"))
```



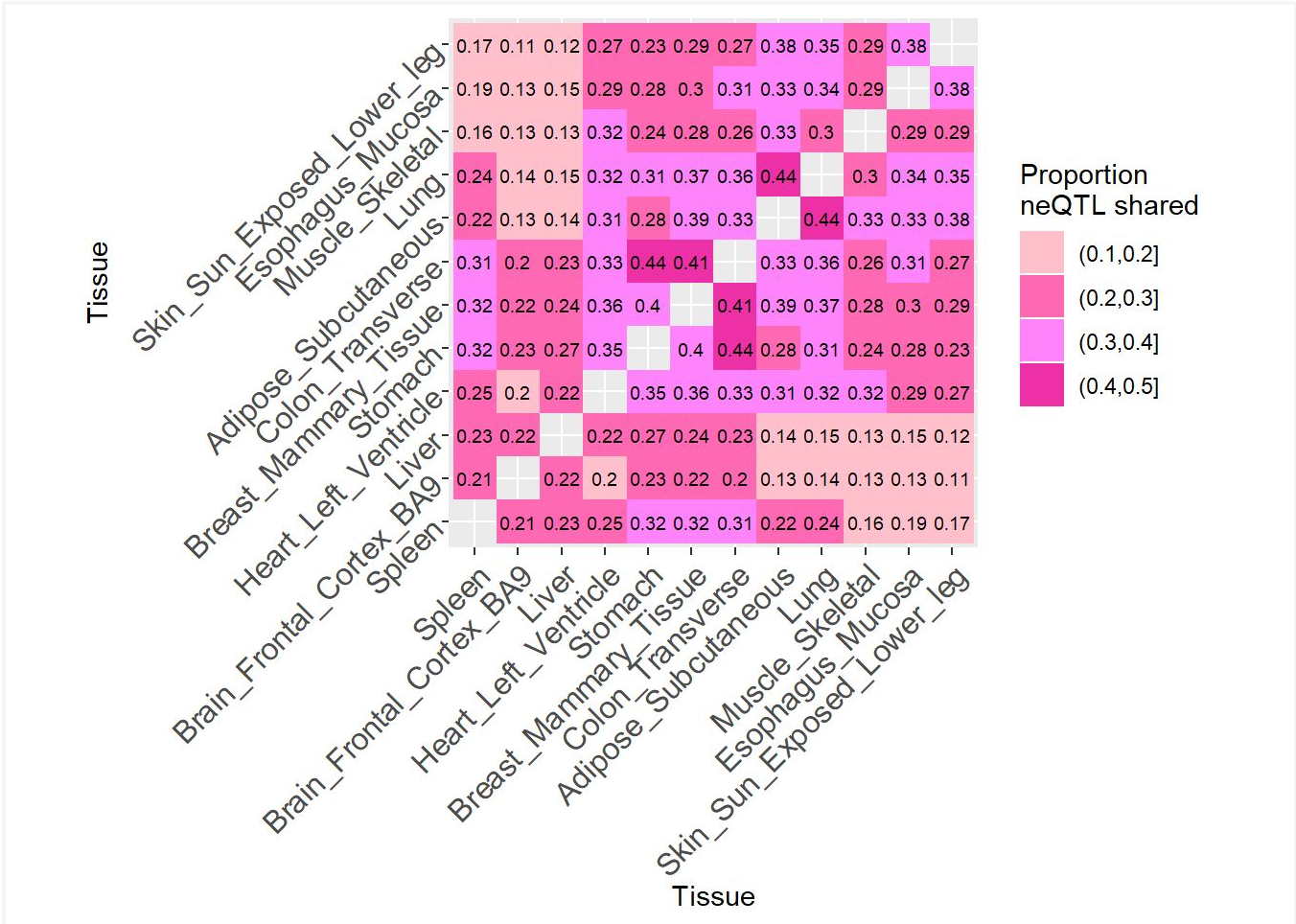
In ggplot there is a lot more flexibility in how colors are chosen. There are four main arguments within ggplot that let you control color:

1. scale_color_manual
2. scale_color_gradient
3. scale_color_gradient2
4. scale_color_gradientn
5. scale_color_brewer
6. scale_color_grey

Below are examples of each:

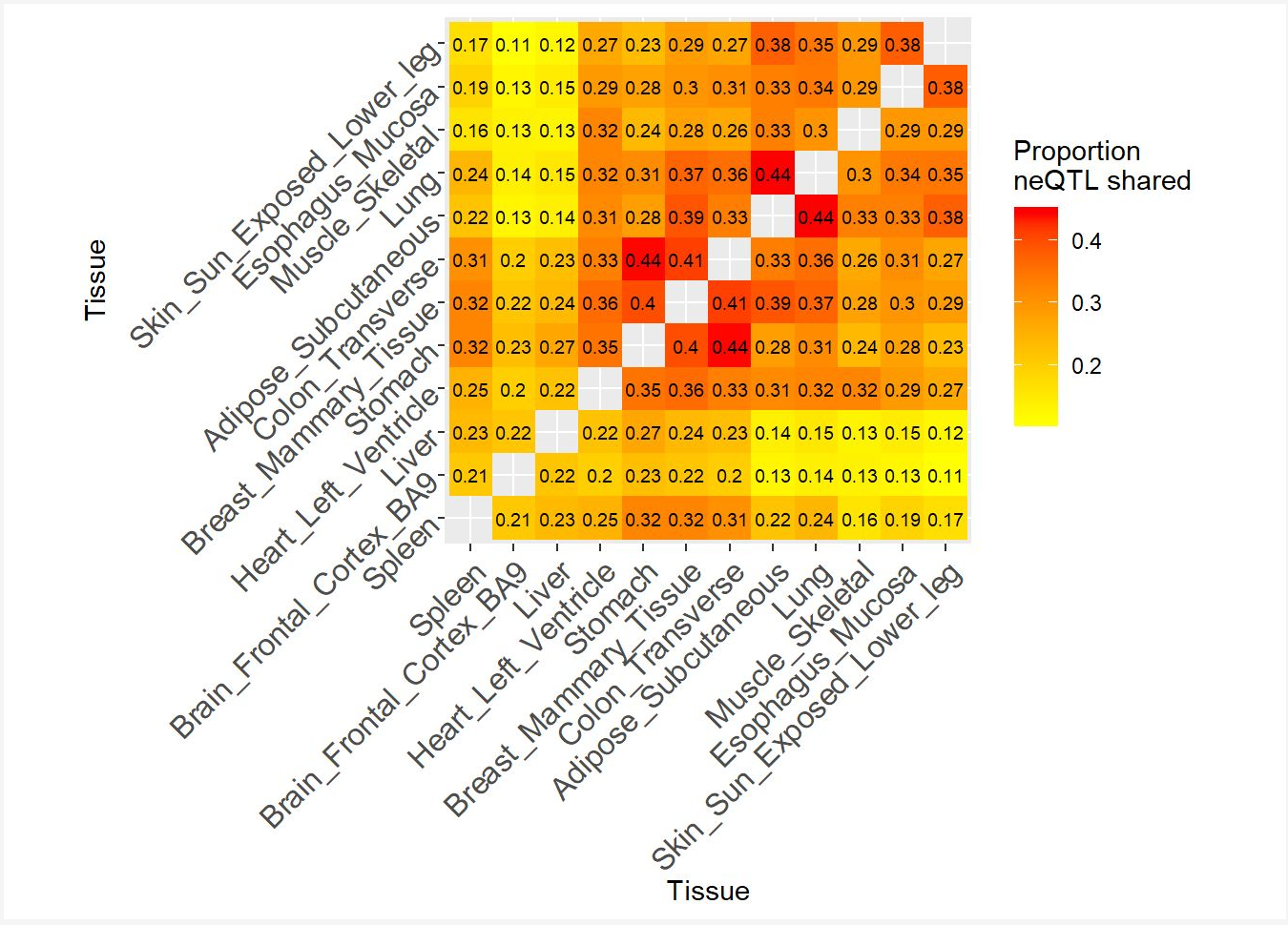
6.0.1 scale_fill_manual

```
toplot <- toplot %>% mutate(PropShareCat = cut(PropShare, breaks =
seq(0,.5,by=.1)))
q <- ggplot(toplot, aes(x=Tissue1, y=Tissue2)) +
  theme(axis.text.x = element_text(angle = 45),
        axis.text=element_text(size=12, hjust=1),
        axis.text.y = element_text(angle = 45),
        axis.title = element_text(hjust=.5)) +
  coord_equal() + xlab("Tissue") + ylab("Tissue")

  p <- q + geom_tile(aes(fill = PropShareCat)) +
  scale_fill_manual(name = "Proportion\nneQTL shared",
                    values = c("pink","hotpink","orchid1","maroon2","red")) +
  geom_text(aes(label = as.character(round(PropShare, 2))),
            size = 2.5)
print(p)
```



6.0.2 scale_fill_gradient: Allows for a continuous gradient between two colors.

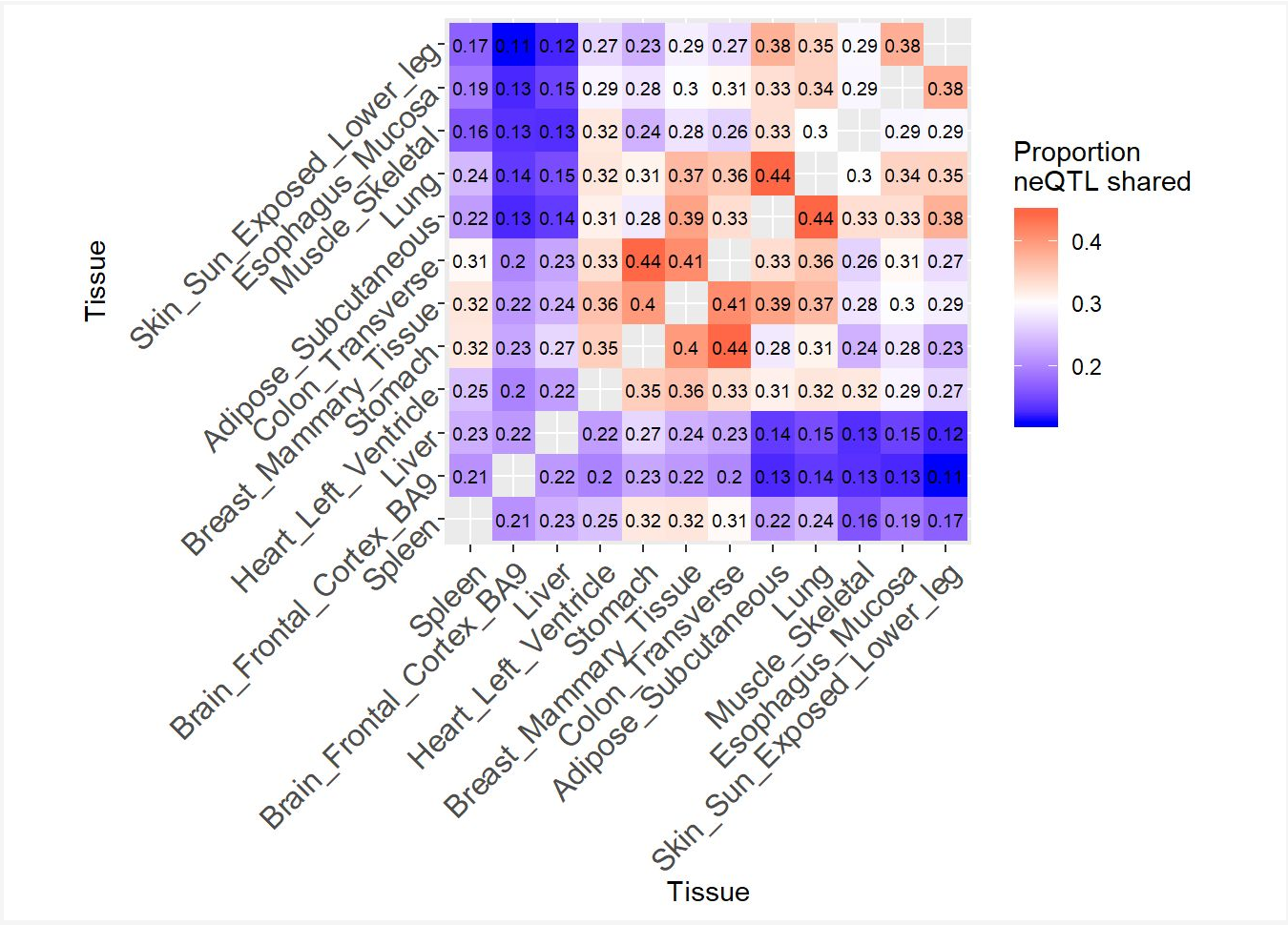
```
p <- q + geom_tile(aes(fill = PropShare)) +
  scale_fill_gradient(name = "Proportion\nneQTL shared",
                      low = "yellow", high="red") +
  geom_text(aes(label = as.character(round(PropShare, 2))),
            size = 2.5)
print(p)
```



6.0.3 scale_fill_gradient2: Allow for a continuous gradient between two sets of color pairs.

```
p <- q + geom_tile(aes(fill = PropShare)) +
  scale_fill_gradient2(name = "Proportion\nneQTL shared",
```

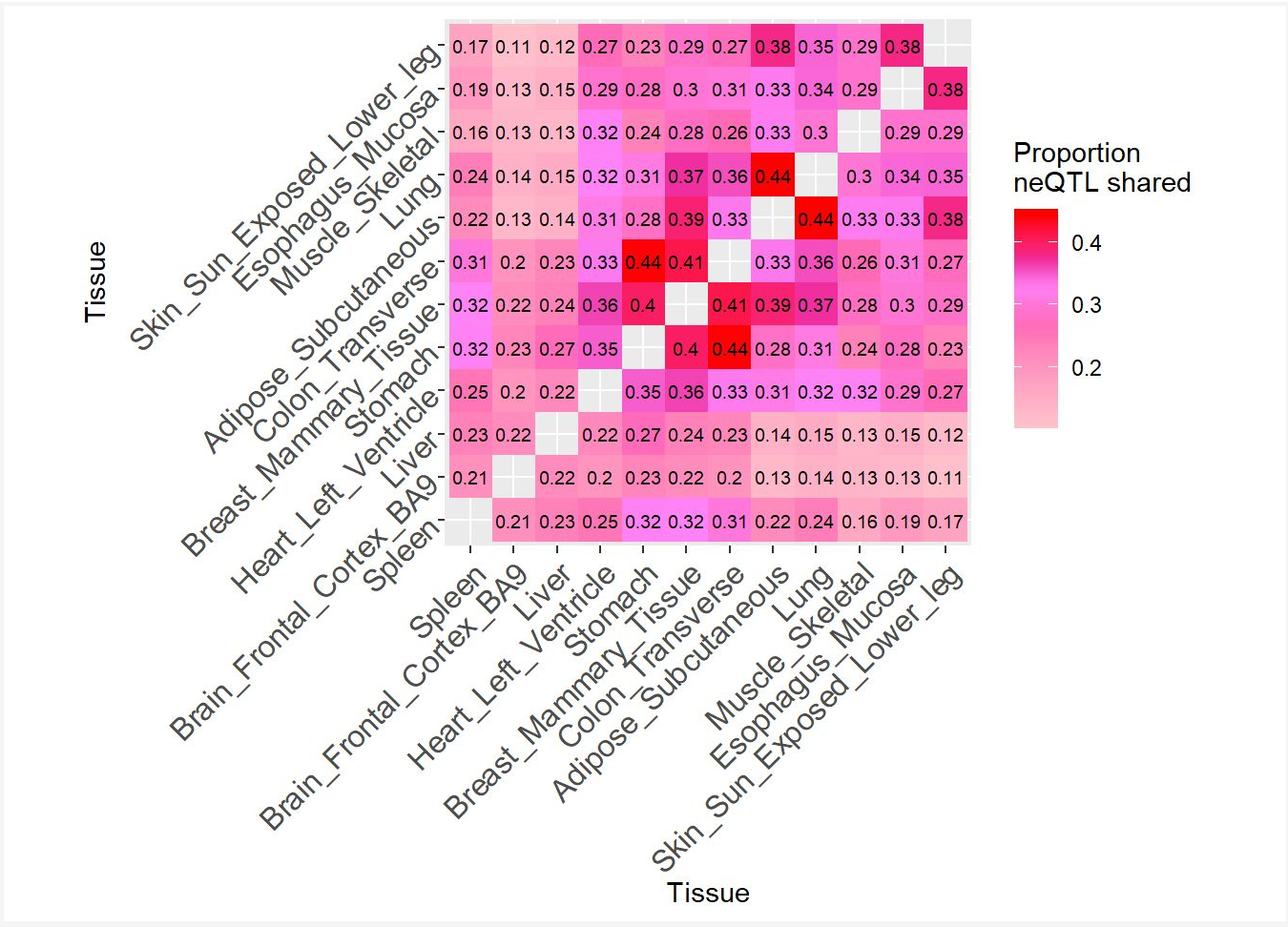
```
                low = "blue", mid= "white", high="red",
                midpoint = .3) +
  geom_text(aes(label = as.character(round(PropShare, 2))),
            size = 2.5)
print(p)
```



6.0.4 scale_fill_gradientn: Allow for a continuous gradient between multiple sets of color pairs.

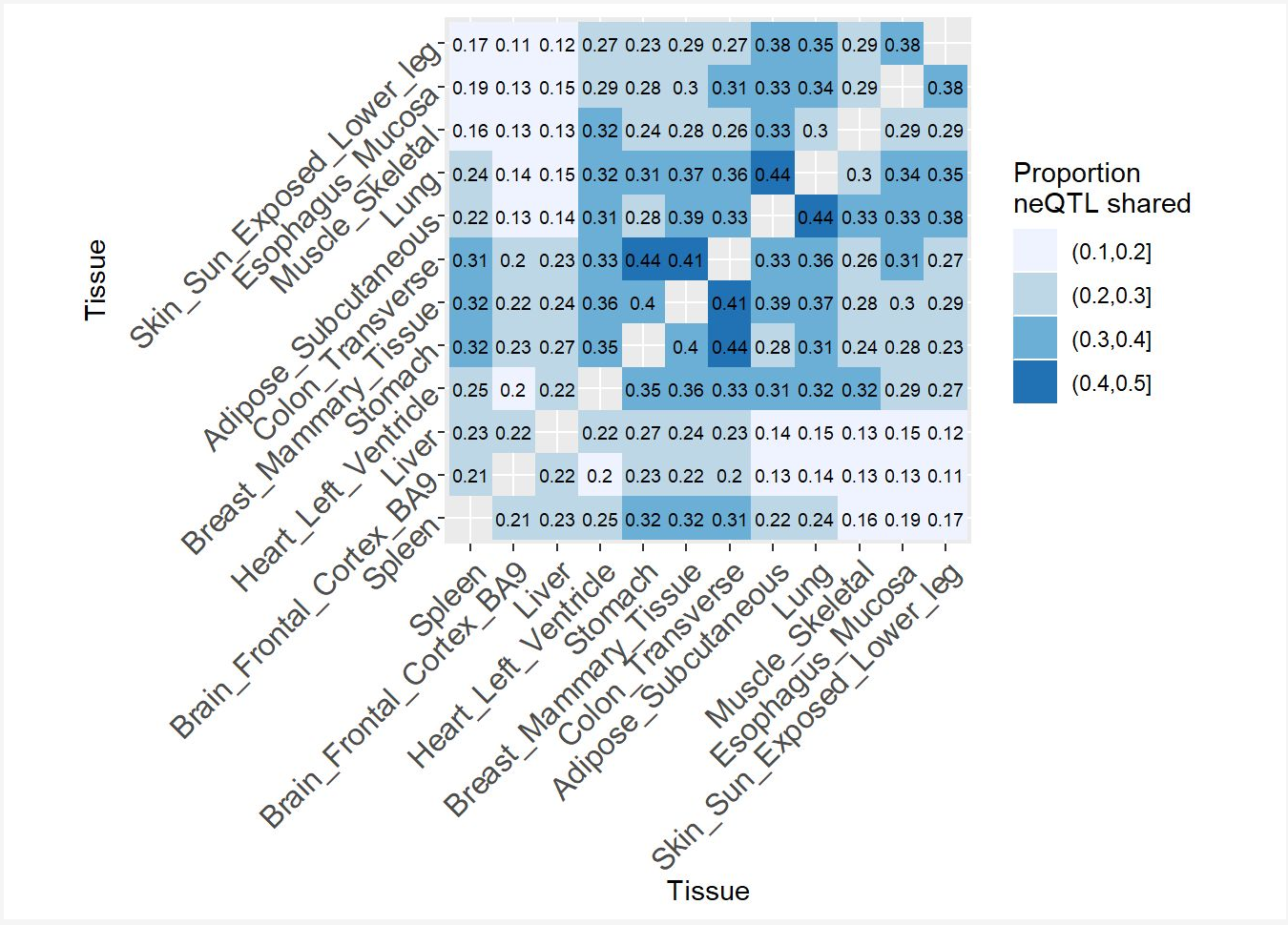
```
p <- q + geom_tile(aes(fill = PropShare)) +
  scale_fill_gradientn(name = "Proportion\nneQTL shared",
                colors= c("pink","hotpink","orchid1","maroon2","red"),
                values = c(0,.4,.6,.7,.8, 1)) +
  geom_text(aes(label = as.character(round(PropShare, 2))),
            size = 2.5)
```

```
print(p)
```



6.0.5 scale_fill_brewer: Allow for easy integration of RColorBrewer palettes. Works on discrete scale only.

```
p <- q + geom_tile(aes(fill = PropShareCat)) +
  scale_fill_brewer(name = "Proportion\nneQTL shared",
                    palette="Blues") +
  geom_text(aes(label = as.character(round(PropShare, 2))),
            size = 2.5)
print(p)
```



7 Summary

ggplot is an extremely powerful and generally intuitive enviroment with which to create a bevy of different types of plots. Most of the plots have a common structure and expect data in a similar organization. There are a number of packages that have been created that integrate with ggplot2, such as repel, which extend the basic (and extensive) ggplot functionality. While the documentation is pretty good for ggplot, sometimes it's nice to see examples. Google is your friend here.

Creating excellent and informative plots is an iterative process, and often time consuming process. Be patient, especially if you are just getting started with ggplot2. When you think you are done, show your plots to a collegue, or your mother, and see if they can understand what you are trying to communicate. No one should ever have to ask you what is plotted on an axis or what a color or a dot size represents. There are

lots of guides out there for effective communication of numbers. There's even a paper in PLoS Computational Biology (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4161295/).

Communicating effectively with plots can be extremely satisfying. Take your time and have fun. Good luck.

8 Useful Links

8.1 ggplot cheat sheets

https://rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf
https://github.com/rstudio/cheatsheets/raw/master/data-visualization-2.1.pdf

8.2 Data manipulation cheat sheet

https://github.com/rstudio/cheatsheets/raw/master/data-transformation.pdf

8.3 RMarkdown cheat sheet

https://github.com/rstudio/cheatsheets/raw/master/rmarkdown-2.0.pdf

8.4 ggplot tutorials

https://tutorials.iq.harvard.edu/R/Rgraphics/Rgraphics.html
http://r-statistics.co/Complete-Ggplot2-Tutorial-Part1-With-R-Code.html
https://cedricscherer.netlify.com/2019/08/05/a-ggplot2-tutorial-for-beautiful-plotting-in-r/
http://www.sthda.com/english/wiki/be-awesome-in-ggplot2-a-practical-guide-to-be-highly-effective-r-software-and-data-visualization

8.5 R tutorials

8.6 Colors in R reference

https://www.nceas.ucsb.edu/~frazier/RSpatialGuides/colorPaletteCheatsheet.pdf