Chapter 7 examples

Brooke Anderson

4/2/2020

```
library(tidyverse)
```

Weighted PCA example (7.8.3)

```
# Uncomment and run the next line if you don't have the `Hiiragi2013` package
# BiocManager::install("Hiiragi2013")
```

There's a vignette with more details on this package available here. It sounds like this data includes 66 wild-type animals and 34 of a type of mutant (35 FGF4-KO mutants) of the same animal (mice, maybe?). It looks like, for each animal, they're measuring levels of gene expression (mRNA, maybe?).

Load and check out the data we're using for this exercise:

```
data("x", package = "Hiiragi2013")
class(x)
```

```
## [1] "ExpressionSet"
## attr(,"package")
## [1] "Biobase"
```

```
str(x, max.level = 2)
```

```
## Formal class 'ExpressionSet' [package "Biobase"] with 7 slots
##   ..@ experimentData   :Formal class 'MIAME' [package "Biobase"] with 13 slots
##   ..@ assayData        :<environment: 0x7fec9ca78268>
##   ..@ phenoData        :Formal class 'AnnotatedDataFrame' [package "Biobase"] with 4 slots
##   ..@ featureData      :Formal class 'AnnotatedDataFrame' [package "Biobase"] with 4 slots
##   ..@ annotation       : chr "mouse4302"
##   ..@ protocolData     :Formal class 'AnnotatedDataFrame' [package "Biobase"] with 4 slots
##   ..@ .__classVersion__:Formal class 'Versions' [package "Biobase"] with 1 slot
```

```
x@phenoData # Remember that you can use `@` to extract an element from an S4 object
```

```
## An object of class 'AnnotatedDataFrame'
##   sampleNames: 1 E3.25 2 E3.25 ... 101 E4.5 (FGF4-KO) (101 total)
##   varLabels: File.name Embryonic.day ... sampleColour (8 total)
##   varMetadata: labelDescription
```

```
str(x@phenoData)
```

```
## Formal class 'AnnotatedDataFrame' [package "Biobase"] with 4 slots
##   ..@ varMetadata      :'data.frame':    8 obs. of  1 variable:
##   .. ..$ labelDescription: chr [1:8] NA NA NA NA ...
##   ..@ data             :'data.frame':    101 obs. of  8 variables:
##   .. ..$ File.name            : chr [1:101] "1_C32_IN" "2_C32_IN" "3_C32_IN" "4_C32_IN" ...
##   .. ..$ Embryonic.day        : Factor w/ 3 levels "E3.25","E3.5",..: 1 1 1 1 1 1 1 1 1 1 ...
##   .. ..$ Total.number.of.cells: Factor w/ 11 levels "32","33","34",..: 1 1 1 1 1 1 1 1 1 1 ...
##   .. ..$ lineage              : chr [1:101] "" "" "" "" ...
##   .. ..$ genotype             : Factor w/ 2 levels "FGF4-KO","WT": 2 2 2 2 2 2 2 2 2 2 ...
##   .. ..$ ScanDate             : Factor w/ 9 levels "2010-06-30","2010-07-01",..: 6 6 6 6 6 6 6 6 6
##   .. ..$ sampleGroup          : chr [1:101] "E3.25" "E3.25" "E3.25" "E3.25" ...
##   .. ..$ sampleColour         : chr [1:101] "#CAB2D6" "#CAB2D6" "#CAB2D6" "#CAB2D6" ...
##   ..@ dimLabels        : chr [1:2] "sampleNames" "sampleColumns"
##   ..@ .__classVersion__:Formal class 'Versions' [package "Biobase"] with 1 slot
##   .. .. ..@ .Data:List of 1
##   .. .. .. ..$ : int [1:3] 1 1 0
```

It looks like `genotype` in the `phenoData` slot is giving whether the sample was wild type ("WT") or a mutant ("FGF4-K0"). There is some other data here about each sample, too, like the date when the sample was scanned, the sample color (?), the file name (probably, the equipment output one file per sample, so those are serving as the input in creating the ExpressionSet data in `x`), the total number of cells in the sample, and the embryonic day.

You can use the function `exprs` to extract the data from

```
x %>%
  exprs() %>%
  `[`(1:10, 1:10) # Using a trick to pipe into the `[ , ]`-style subsetting function
```

```
##                 1 E3.25   2 E3.25   3 E3.25   4 E3.25   5 E3.25   6 E3.25
## 1415670_at     4.910459  7.526672  6.956328  6.424048  5.105808  5.856685
## 1415671_at     9.768979  9.144228  9.295010 11.059831  9.376749  9.681017
## 1415672_at    10.411893 10.918942  9.495738 10.317996 11.143684 10.234943
## 1415673_at     5.618108  6.439416  6.730465  4.914527  5.619778  7.188673
## 1415674_a_at   7.541891  8.380285  8.480580  7.977363  8.650312  8.639637
## 1415675_at     8.590070  7.661697  8.741957  9.147643  8.868919  8.595630
## 1415676_a_at  10.577587 10.862713  9.584166 10.961204 10.859650 10.832484
## 1415677_at     5.077198  3.653334  3.875342  3.967968  3.919776  4.157619
## 1415678_at    11.458378 10.630368 11.128941 11.386041 10.873872 10.847419
## 1415679_at    10.166575  9.835880  9.519152 10.588845  9.536259  9.697174
##                 7 E3.25   8 E3.25   9 E3.25  10 E3.25
## 1415670_at     5.059961  4.574661  8.123073  5.464257
## 1415671_at     7.665886  9.325743  9.724729  9.389818
## 1415672_at    10.642970  9.304958 11.037632  9.754123
## 1415673_at     6.395441  6.405085  6.542729  6.842668
## 1415674_a_at   7.645431  5.265520  6.748849  6.951920
## 1415675_at     8.266214  8.359522  8.896249  9.503661
## 1415676_a_at  10.607946 10.615123 10.764935 11.070344
## 1415677_at     3.726769  4.387729  5.163832  4.048134
## 1415678_at    11.801188 11.340817 11.534948 12.082674
## 1415679_at     9.037631 10.023927  9.970339  9.805573
```

```
                  # It's an "in-fix" function (like "+" and "/"), so you usually use
                  # it within a line of code (instead of with parentheses). However,
                  # all of those will also work like regular functions if you surround
                  # the name with backticks, so '1 + 2' is the same in R as '`[`(1, 2)'.
```

Functions like `exprs`, which exist only to extract some of the data stored in a certain type of object, are called *extractor* functions. If you need to get all the way to a dataframe, run `as_tibble` (from the tidyverse) or `as.data.frame` (from base R) right after you extract these data.

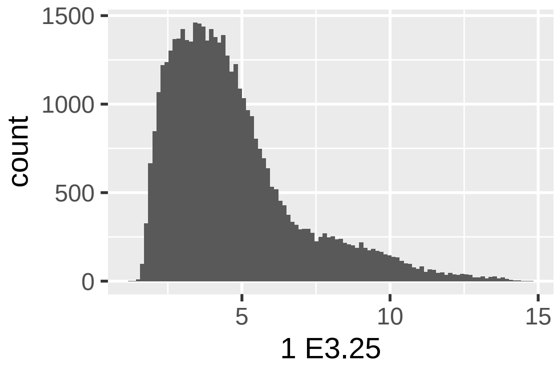
You can use `ggplot2` to explore the data, although keep in mind that the column names aren't in a standard formula. Instead, they start with numbers and have spaces:

```r
x %>%
  exprs() %>%
  colnames() %>%
  head()
```

```
## [1] "1 E3.25" "2 E3.25" "3 E3.25" "4 E3.25" "5 E3.25" "6 E3.25"
```

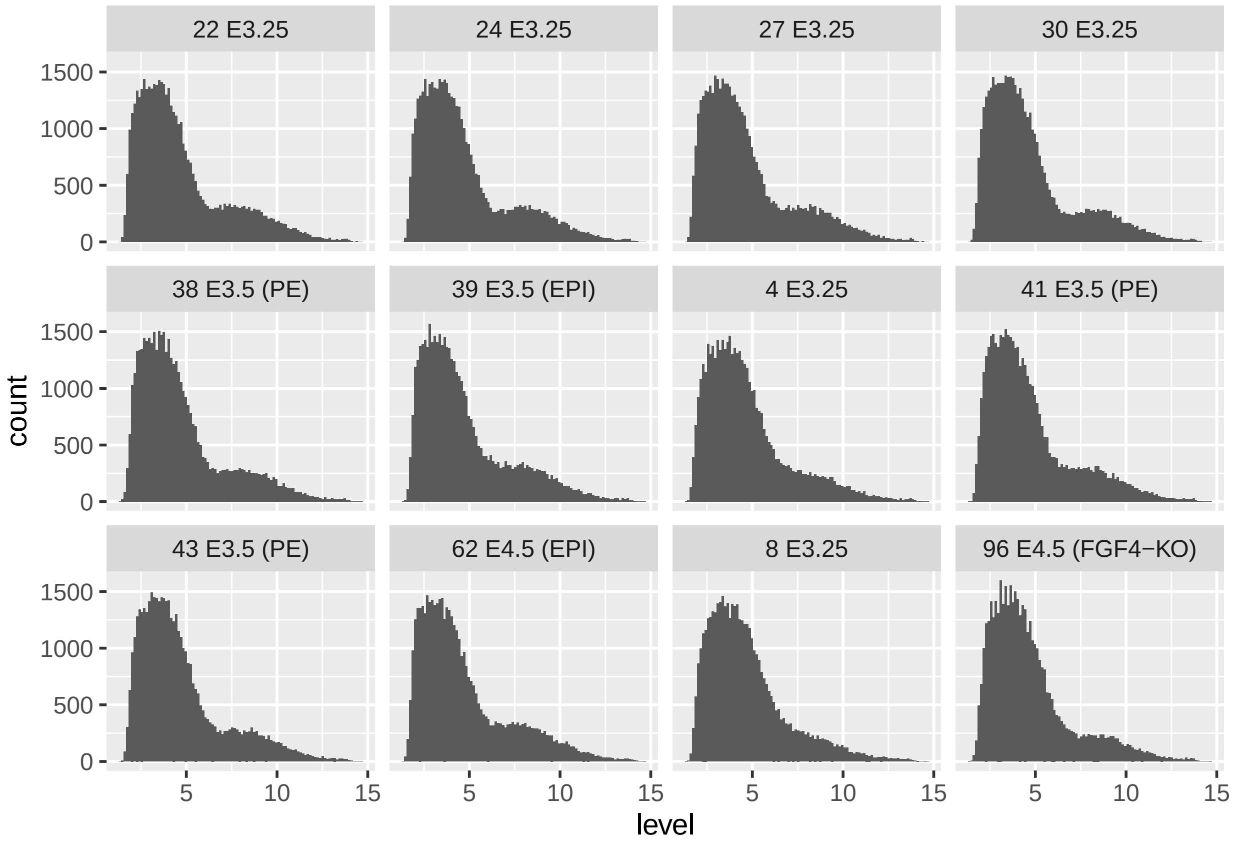
That means that you'll need to "protect" the column name in any tidyverse code, by using backticks around the column name. For example, you can run the following to create a histogram of expression levels in the first column (the first animal sample?):

```r
x %>%
  exprs() %>%
  as_tibble() %>%
  ggplot(aes(x = `1 E3.25`)) +
  geom_histogram(bins = 100)
```



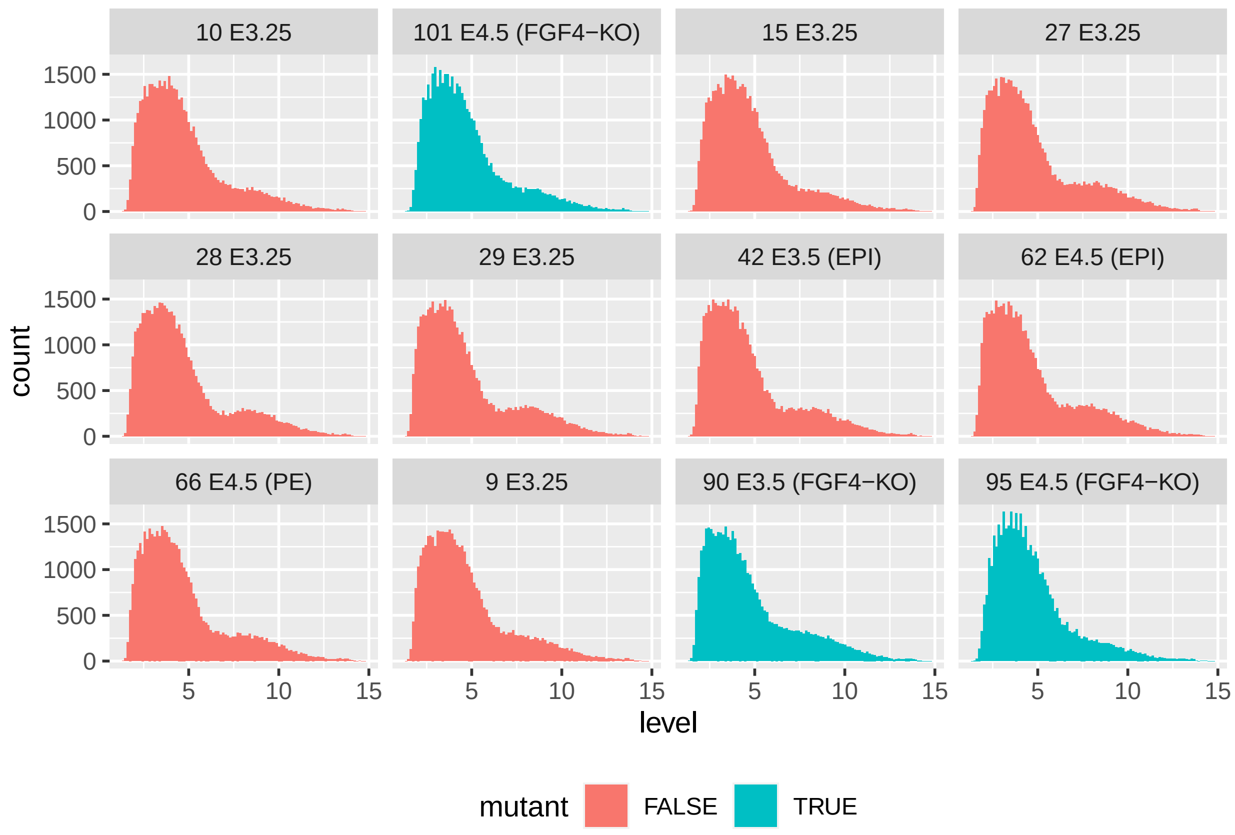
If you'd like to get histograms for several of these, you can take advantage of `pivot_longer` and facetting to do that. For example, to create these for the a random sample of twelve columns, run:

```r
x %>%
  exprs() %>%
  as_tibble() %>%
  select(sample(1:ncol(.), size = 12)) %>%  # Sample twelve columns. The `.` is a "pronoun"--
                                             # it refers to the dataframe you've just piped in
  pivot_longer(cols = 1:12, names_to = "sample", values_to = "level") %>%
  ggplot(aes(x = level)) +
  geom_histogram(bins = 100) +
  facet_wrap(~ sample)
```



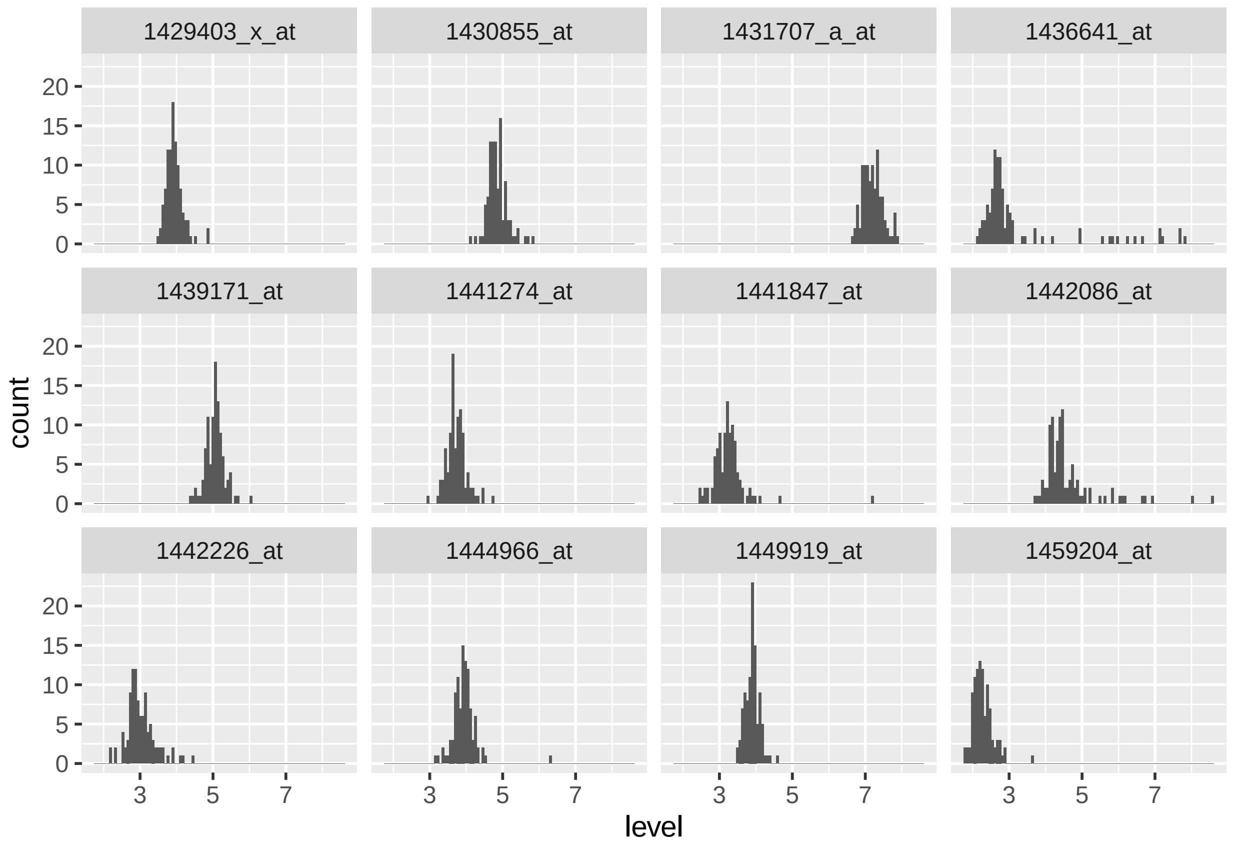
It looks like each of the original column names includes "FGF4-KO" if the sample animal is a mutant, rather than a wild-type. We might want to use the fill of the bars to show which samples are wild-type versus mutant. Once you've made the data longer, you can use regular expressions to determine, based on whether a label includes "FGF4-KO", if the animal is a mutant, and then use that when you plot:

```r
x %>%
  exprs() %>%
  as_tibble() %>%
  select(sample(1:ncol(.), size = 12)) %>%
  pivot_longer(cols = 1:12, names_to = "sample", values_to = "level") %>%
  mutate(mutant = str_detect(sample, "(FGF4-KO)")) %>% # Use regular expressions here
  ggplot(aes(x = level, fill = mutant)) +  # Add the mapping to fill for the `mutant` column
  geom_histogram(bins = 100) +
  facet_wrap(~ sample) +
  theme(legend.position = "bottom")
```



It looks like each row is for a separate gene (? mRNA? transcript?). You might want to instead get histograms for each of those (instead of by sample). I think the easiest way to do that would be to transpose the data first (`t`—that is, flip the rows and columns) and then continue from there:

```r
x %>%
  exprs() %>%
  t() %>% # Here's where I'm switching rows and columns
  as_tibble() %>%
  select(sample(1:ncol(.), size = 12)) %>%
  pivot_longer(cols = 1:12, names_to = "transcript", values_to = "level") %>%
  ggplot(aes(x = level)) +
  geom_histogram(bins = 100) +
  facet_wrap(~ transcript)
```



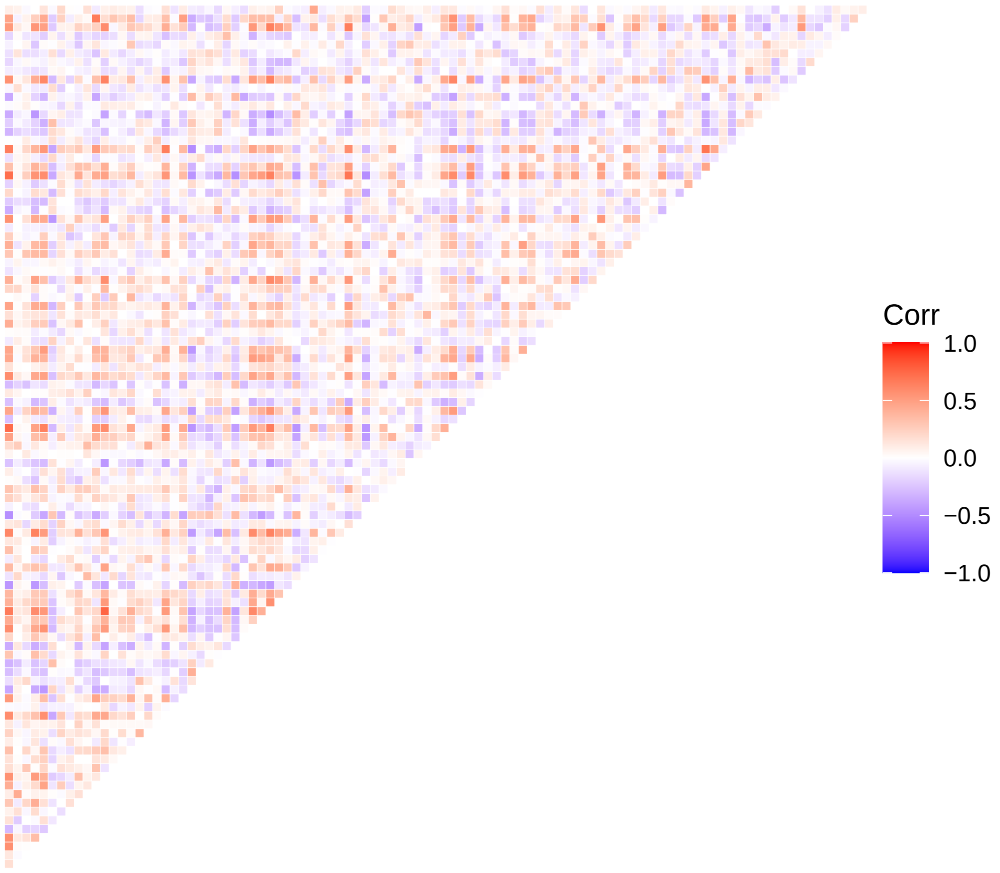
Correlation matrices might be interesting here, too. The `ggcorrplot` package has some nice functions for making those. First, check out the size of the data:

```
x %>%
  exprs() %>%
  dim()
```

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

We could probably fit 101 values in a correlation plot, so we could do one of all the samples, but we probably can't for all the gene expression levels (over 45,000!). We can look to see correlation patterns in that, but we probably should look at just a sample, not everything at once.

```
library(ggcorrplot)

x %>%
  exprs() %>%
  t() %>%
  as_tibble() %>%
  select(sample(1:ncol(.), size = 100)) %>%
  cor() %>%  # Calculate the correlation matrix
  ggcorrplot(outline.col = "white", type = "upper") +
  theme_void() # No point in having x and y labels right now---they'll be too small to see
```



It looks like you probably have some columns that are pretty strongly correlated with each other, both negatively and positively.

Tidier version of code in book

The code in the book walks you through doing a weighted PCA. They first recommend that you limit the data to the wild-type samples and then select the 100 features (genes?) with the highest overall variance. Here's a "tidier" way to do that than in the book.

First, you can use one pipeline to make a dataframe that's limited to the 66 wild-type samples and the top 100 features by variance. We're transposing it along the way (`t`), so it will be in the right format for the `dudi.pca` call later:

```
simpl_data <- x %>%
  exprs() %>%
  t() %>%
  as.data.frame() %>%
  rownames_to_column(var = "sample") %>%
  filter(!str_detect(sample, "(FGF4-KO)")) %>%
  pivot_longer(-sample, names_to = "transcript", values_to = "level") %>%
  group_by(transcript) %>%
  nest() %>%
  mutate(var = map_dbl(.x = data, .f = ~ var(.x$level))) %>% # Calculate the variance for
                                                             # each transcript
  ungroup() %>%
  top_n(n = 100, wt = var) %>% # Extract the top 100 transcripts in terms of variance
  select(-var) %>% # We don't need the variance now that we've picked the top 100, go remove it
  unnest(data) %>% # Unnest the data
  pivot_wider(names_from = transcript, values_from = level) # Make wide again

simpl_data
```

```
## # A tibble: 66 x 101
##    sample `1417065_at` `1417185_at` `1417695_a_at` `1417868_a_at` `1418072_at`
##    <chr>         <dbl>        <dbl>          <dbl>          <dbl>        <dbl>
##  1 1 E3.~         3.53         9.51           9.53           7.98         4.33
##  2 2 E3.~         7.08        12.6            8.57           2.00         2.59
##  3 3 E3.~         4.07         9.16           9.18           2.70         9.86
##  4 4 E3.~         3.87        11.1            9.23           2.26         3.70
##  5 5 E3.~         4.28        10.9            8.58           1.81         7.85
##  6 6 E3.~         3.83         9.27           2.51           2.74         1.89
##  7 7 E3.~         3.68        11.8            2.11           8.64         3.44
##  8 8 E3.~         4.71        11.5            9.63           2.46         1.90
##  9 9 E3.~         4.41        12.3            8.77           2.12         8.05
## 10 10 E3~         3.72         6.42           3.05           2.43         8.70
## # ... with 56 more rows, and 95 more variables: `1418153_at` <dbl>,
## #   `1418203_at` <dbl>, `1418209_a_at` <dbl>, `1418486_at` <dbl>,
## #   `1418817_at` <dbl>, `1418887_a_at` <dbl>, `1418914_s_at` <dbl>,
## #   `1419418_a_at` <dbl>, `1419737_a_at` <dbl>, `1419824_a_at` <dbl>,
## #   `1420064_s_at` <dbl>, `1420085_at` <dbl>, `1420086_x_at` <dbl>,
## #   `1420191_s_at` <dbl>, `1420498_a_at` <dbl>, `1421917_at` <dbl>,
## #   `1422325_at` <dbl>, `1422486_a_at` <dbl>, `1422557_s_at` <dbl>,
## #   `1423747_a_at` <dbl>, `1423754_at` <dbl>, `1424349_a_at` <dbl>,
## #   `1424649_a_at` <dbl>, `1425020_at` <dbl>, `1426255_at` <dbl>,
## #   `1426438_at` <dbl>, `1426722_at` <dbl>, `1426990_at` <dbl>,
## #   `1428471_at` <dbl>, `1428572_at` <dbl>, `1428749_at` <dbl>,
## #   `1429177_x_at` <dbl>, `1429388_at` <dbl>, `1429483_at` <dbl>,
## #   `1429654_at` <dbl>, `1430776_s_at` <dbl>, `1431805_a_at` <dbl>,
## #   `1433509_s_at` <dbl>, `1434046_at` <dbl>, `1434170_at` <dbl>,
## #   `1434584_a_at` <dbl>, `1434628_a_at` <dbl>, `1435493_at` <dbl>,
## #   `1435494_s_at` <dbl>, `1436392_s_at` <dbl>, `1436833_x_at` <dbl>,
## #   `1436838_x_at` <dbl>, `1436944_x_at` <dbl>, `1437009_a_at` <dbl>,
## #   `1437301_a_at` <dbl>, `1437308_s_at` <dbl>, `1437325_x_at` <dbl>,
## #   `1437534_at` <dbl>, `1438292_x_at` <dbl>, `1438941_x_at` <dbl>,
## #   `1439036_a_at` <dbl>, `1439148_a_at` <dbl>, `1439255_s_at` <dbl>,
## #   `1439256_x_at` <dbl>, `1440254_at` <dbl>, `1440910_at` <dbl>,
## #   `1443779_s_at` <dbl>, `1444390_at` <dbl>, `1445897_s_at` <dbl>,
## #   `1447640_s_at` <dbl>, `1447845_s_at` <dbl>, `1447997_s_at` <dbl>,
## #   `1448573_a_at` <dbl>, `1448595_a_at` <dbl>, `1448649_at` <dbl>,
## #   `1448666_s_at` <dbl>, `1448830_at` <dbl>, `1449134_s_at` <dbl>,
## #   `1449254_at` <dbl>, `1449732_at` <dbl>, `1449770_x_at` <dbl>,
## #   `1450624_at` <dbl>, `1450793_at` <dbl>, `1450843_a_at` <dbl>,
## #   `1451602_at` <dbl>, `1451791_at` <dbl>, `1452270_s_at` <dbl>,
## #   `1452700_s_at` <dbl>, `1452833_at` <dbl>, `1453304_s_at` <dbl>,
## #   `1454737_at` <dbl>, `1455899_x_at` <dbl>, `1455904_at` <dbl>,
## #   `1456270_s_at` <dbl>, `1456312_x_at` <dbl>, `1456542_s_at` <dbl>,
## #   `1456598_at` <dbl>, `1456661_at` <dbl>, `1460576_at` <dbl>,
## #   `1460605_at` <dbl>
```

You can see our new data frame has 66 samples (just the wild-type samples) and 101 columns (the sample names, then the 100 genes with the largest variance in expression levels).

We can pull out the group from them (in each sample name, everything from "E" later) using regular expressions:

```r
simpl_data <- simpl_data %>%
  mutate(group = str_extract(sample, "E.+")) # Pull everything starting from "E" in 'sample'

simpl_data %>%
  select(sample, group)
```

```
## # A tibble: 66 x 2
##    sample   group
##    <chr>    <chr>
##  1 1 E3.25  E3.25
##  2 2 E3.25  E3.25
##  3 3 E3.25  E3.25
##  4 4 E3.25  E3.25
##  5 5 E3.25  E3.25
##  6 6 E3.25  E3.25
##  7 7 E3.25  E3.25
##  8 8 E3.25  E3.25
##  9 9 E3.25  E3.25
## 10 10 E3.25 E3.25
## # ... with 56 more rows
```

Next, they want us to create a weight for each sample, as the inverse of how many total samples there are in its group.

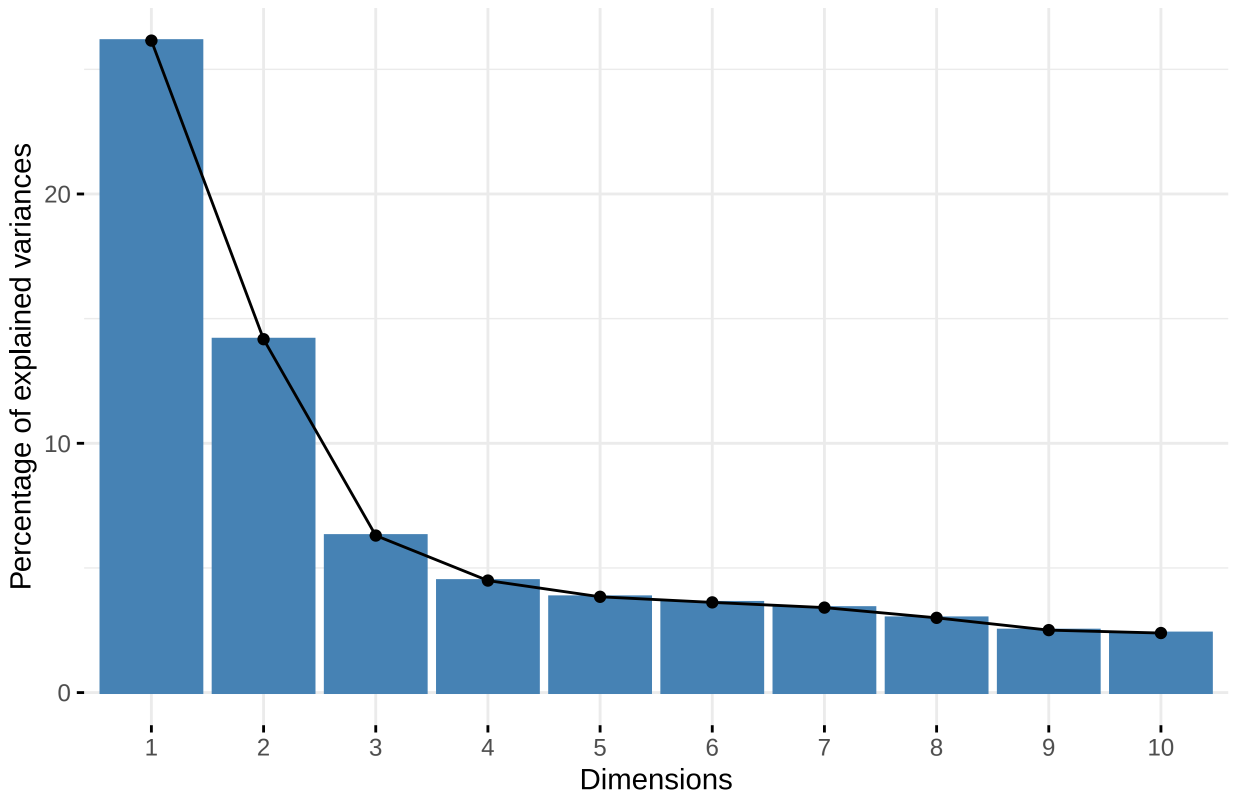
```r
simpl_data <- simpl_data %>%
  group_by(group) %>% # Group by 'group' so we can get the total count for each group
  mutate(n_in_group = n(), # When things are grouped, `mutate` will *add* the summary
                           # information as a column, while keeping the same number of rows
                           # as the original
         weight = 1 / n_in_group) %>% # We want the weight to be 1 / the number of groups
  ungroup()

# Here's an example of what a few of these look like now:
simpl_data %>%
  select(sample, group, n_in_group, weight) %>%
  sample_n(6)
```

```
## # A tibble: 6 x 4
##   sample        group       n_in_group weight
##   <chr>         <chr>            <int>  <dbl>
## 1 7 E3.25       E3.25               36 0.0278
## 2 31 E3.25      E3.25               36 0.0278
## 3 54 E3.5 (PE)  E3.5 (PE)           11 0.0909
## 4 62 E4.5 (EPI) E4.5 (EPI)           4 0.25
## 5 35 E3.25      E3.25               36 0.0278
## 6 8 E3.25       E3.25               36 0.0278
```

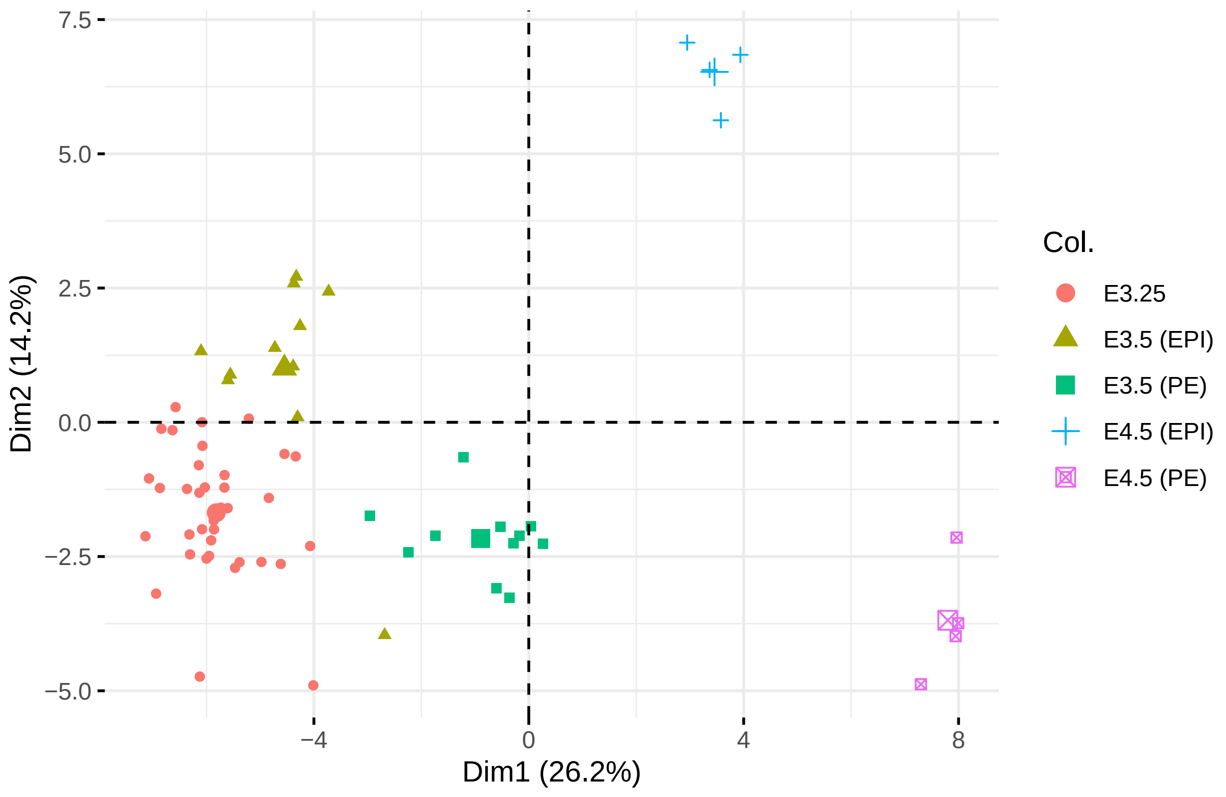
```r
library(ade4)
pcaMouse <- simpl_data %>%
  select(-sample, -group, -n_in_group, -weight) %>%
  dudi.pca(center = TRUE, scale = TRUE, nf = 2, scannf = FALSE,
           row.w = simpl_data$weight)
pcaMouse
```

```
## Duality diagramm
## class: pca dudi
## $call: dudi.pca(df = ., row.w = simpl_data$weight, center = TRUE, scale = TRUE,
##     scannf = FALSE, nf = 2)
##
## $nf: 2 axis-components saved
## $rank: 65
## eigen values: 130.8 70.87 31.5 22.47 19.21 ...
##   vector length mode    content
## 1 $cw    100    numeric column weights
## 2 $lw    66     numeric row weights
## 3 $eig   65     numeric eigen values
##
##   data.frame nrow ncol content
## 1 $tab       66   100  modified array
## 2 $li        66   2    row coordinates
## 3 $l1        66   2    row normed scores
## 4 $co        100  2    column coordinates
## 5 $c1        100  2    column normed scores
## other elements: cent norm
```

```r
library(factoextra)
pcaMouse %>%
  fviz_eig() +
  ggtitle("")
```



```r
pcaMouse %>%
  fviz_pca_ind(geom = "point", col.ind = simpl_data$group) +
  ggtitle("") +
coord_fixed()
```



7.5
5.0
2.5
0.0
−2.5
−5.0
Dim2 (14.2%)
−4
0
4
8
Dim1 (26.2%)
Col.
E3.25
E3.5 (EPI)
E3.5 (PE)
E4.5 (EPI)
E4.5 (PE)