Next steps in R

Final presentations

- Thursday, Dec. 14, 2:00-4:00 pm
- Mayor of Old Town after? (My treat for first 1/2 pint or non-alcoholic drink of your choice)
- Course surveys online this year– details to come

Final presentations / reports

- *Rationale:* Explain what you were hoping to achieve in writing the functions / app framework that your group created.
- *Idea development:* Describe the different ideas your group explored. What were the biggest challenges in this stage? For any ideas that didn't pan out, what were the key constraints? Also describe how you would tackle this problem if you were starting over.

Final presentations / reports

- *Key functions:* Describe the final functions / app framework you decided on. Explain why you picked these. For functions, include documentation for the functions:
  - Write a brief title for the function ($<$ 8 words) and a brief description (3–4 sentences).
  - Define all parameters. For example, if you have a `df` parameter, explain that this is the dataframe that will be modeled / visualized. If it must have certain column with certain names, specify that.
  - Define what the functions will output (e.g., "A ggplot object showing ..." or "The model output object from running a ...").
  - If you have a reference (e.g., for a model you're fitting in the function), you can include that
  - If you want an extra challenge, try to use the Roxygen2 syntax in writing these descriptions. Otherwise, you can write them in code comments.

Final presentations/ reports

- *Room for errors:* So far, we have focused on getting working prototypes, without making sure they're error-proof and robust to a user doing something non-standard. Identify three things a user could do that could make your functions "break" (i.e., either return an error message or return something other than what you hope they will).

Final presentations / reports

- *Next steps:* Include a section where you describe what you think are interesting next steps, i.e., what you would pursue next if you were continuing work on this project. Lay out explicitly a few ideas (2–3) that you think would be helpful. Be sure, when relevant, to describe how feedback from the project researchers helped in forming these ideas for next steps.

Final presentations / reports

- The presentation should be 15–18 minutes per group.
- For functions, you should show both some of the code and an example of output during the presentation.
- For the final report, the functions and their documentation should be in ".R" files.
- For the final report, everything else will be in a Word report. You may reference the functions you created by name in this report. The report should be no longer than 5 pages (using default font size, line spacing, etc. for RMarkdown documents; if you include figures or tables that show output from your functions, these do not need to count towards that page limit).
- Be sure to show examples of using your functions in your Word report.

Unsupervised learning

Example applications:

- Look for subgroups among samples of breast cancer patients
- ID shoppers with similar browsing and purchase histories (recommendation system)

Challenges:

- No well-defined goal
- Hard to assess if a technique did well
- Hard to determine if clusters just result from noise
- Choices like number of clusters, dissimilarity measure, and type of linkage can have a big influence on results

NCI-60 panel of cancer cells

This example is from James et al., Ch. 10 (*An Introduction to Statistical Learning*).

There is an R package called `ISLR` that includes datasets to go along with the James textbook.

```
library(ISLR)
## ?NCI60
```

The `NCI60` dataset is microarray data from the National Cancer Institute, with expression levels on 6,830 genes from 64 cancer cells.

NCI-60 panel of cancer cells

```
## [1] "data" "labs"
```

```
## List of 2
##  $ data: num [1:64, 1:6830] 0.3 0.68 0.94 0.28 0.485 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:64] "V1" "V2" "V3" "V4" ...
##   .. ..$ : chr [1:6830] "1" "2" "3" "4" ...
##  $ labs: chr [1:64] "CNS" "CNS" "CNS" "RENAL" ...
```

The dataset has two parts:

- `labs`: Labels with the cancer type for each cell line. Vector, length 64.
- `data`: Dataframe, 64 rows, 6,830 columns.

NCI-60 panel of cancer cells

```
nci_labs <- NCI60$labs
nci_data <- NCI60$data
```

```
nci_labs[1:4]
```

```
## [1] "CNS"   "CNS"   "CNS"   "RENAL"
```

NCI-60 panel of cancer cells

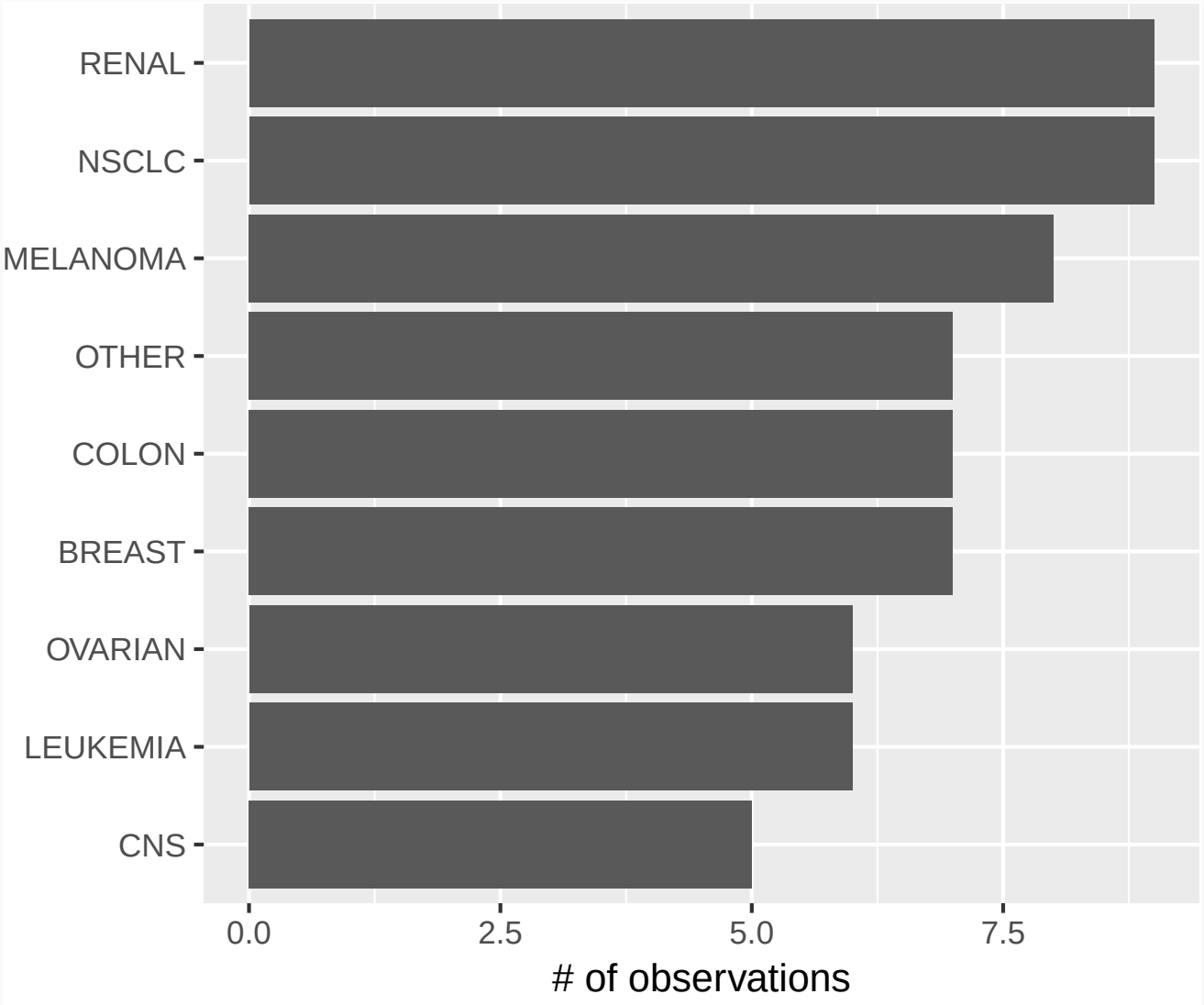
```
data_frame(cancer = nci_labs) %>% group_by(cancer) %>%
  summarize(n = n()) %>% arrange(desc(n)) %>% kable()
```

cancer	n
NSCLC	9
RENAL	9
MELANOMA	8
BREAST	7
COLON	7
LEUKEMIA	6
OVARIAN	6
CNS	5
PROSTATE	2
K562A-repro	1
K562B-repro	1
MCF7A-repro	1
MCF7D-repro	1

NCI-60 panel of cancer cells

```
library(forcats)
data_frame(cancer = nci_labs) %>%
  mutate(cancer = fct_lump(cancer, n = 8,
                           other_level = "OTHER")) %>%
  group_by(cancer) %>%
  summarize(n = n()) %>%
  ggplot(aes(x = fct_reorder(cancer, n), y = n)) +
  geom_col() +
  coord_flip() +
  labs(x = "", y = "# of observations")
```

NCI-60 panel of cancer cells



NCI-60 panel of cancer cells

For the `nci_data` part of the dataset, each row is one of the cell lines and each column gives a measure of gene expression.

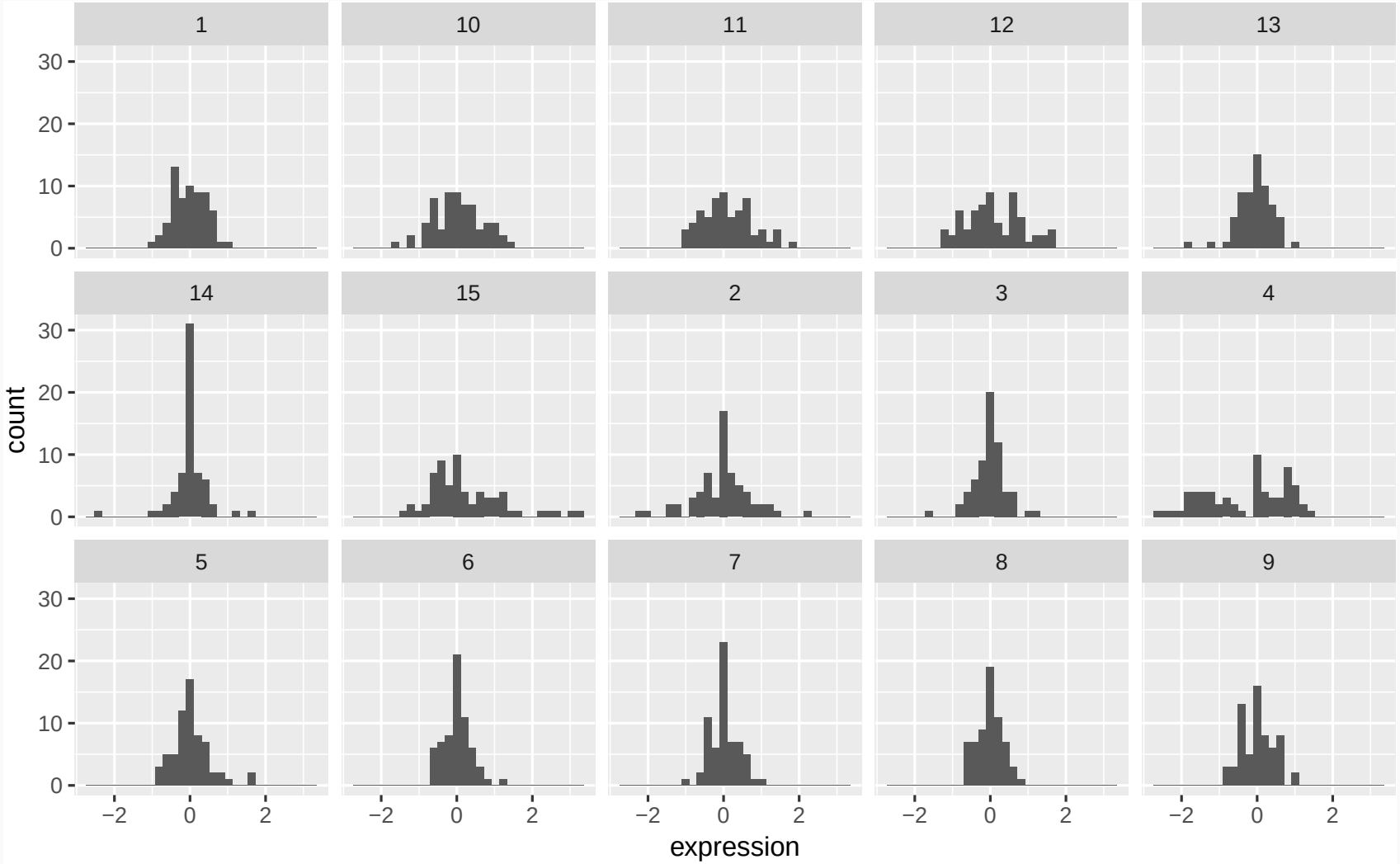
```
nci_data[1:5, 1:5]
```

```
##            1         2         3         4         5
## V1 0.300000  1.180000  0.550000  1.140000 -0.265000
## V2 0.679961  1.289961  0.169961  0.379961  0.464961
## V3 0.940000 -0.040000 -0.170000 -0.040000 -0.605000
## V4 0.280000 -0.310000  0.680000 -0.810000  0.625000
## V5 0.485000 -0.465000  0.395000  0.905000  0.200000
```

Distributions of gene expressions

```
nci_data %>%
  as_tibble() %>%
  select(1:20) %>%
  gather(key = "gene", value = "expression") %>%
  ggplot(aes(x = expression)) +
  geom_histogram() +
  facet_wrap(~ gene)
```

Distributions of gene expressions



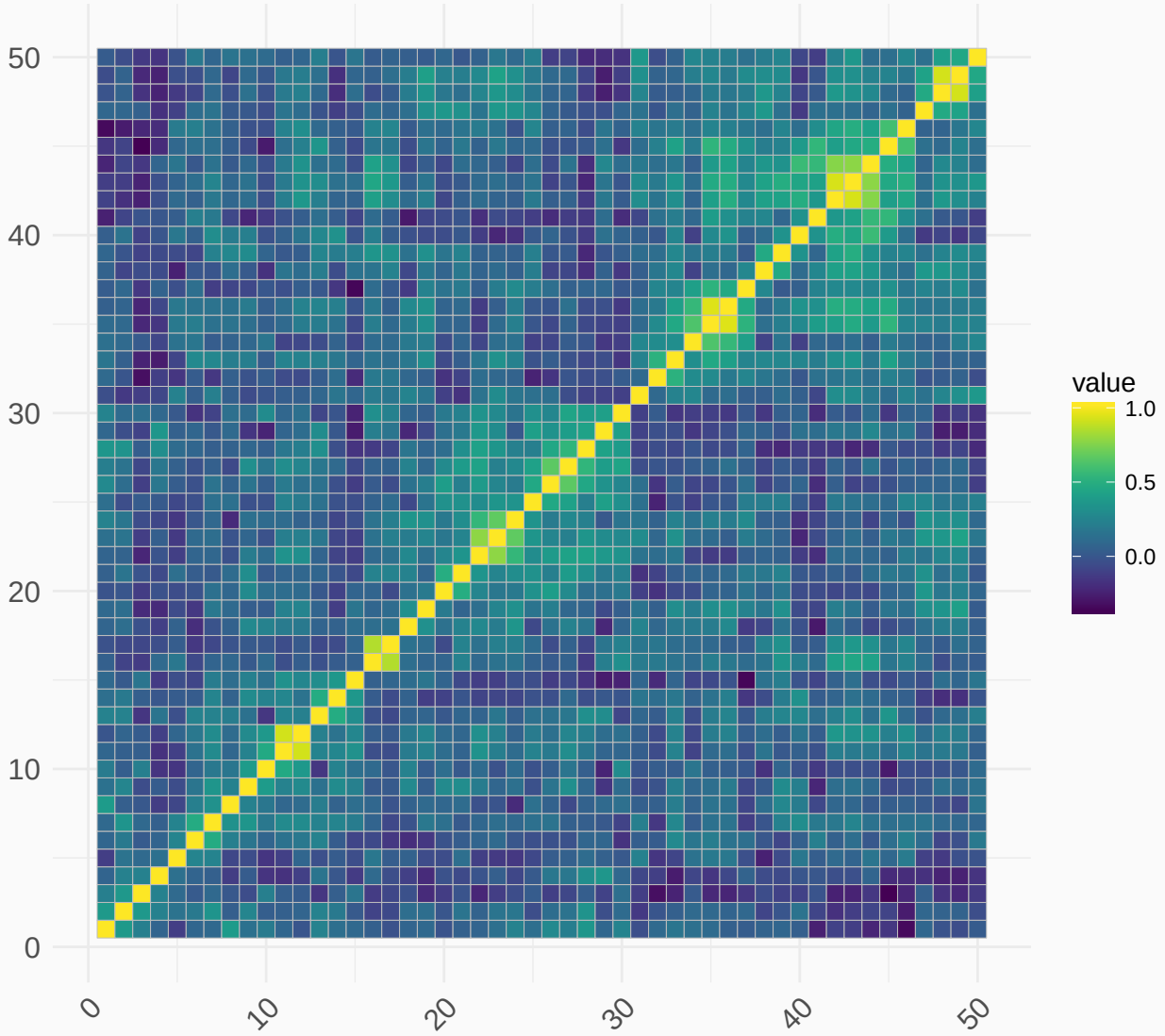
Correlation between gene expressions

```
##             1         2         3         4          5
## 1   1.0000000 0.3695226 0.2248030 0.0725251 -0.1116516
## 2   0.3695226 1.0000000 0.3650872 0.2347458  0.1564618
## 3   0.2248030 0.3650872 1.0000000 0.2344635  0.1051751
## 4   0.0725251 0.2347458 0.2344635 1.0000000  0.1392269
## 5 -0.1116516 0.1564618 0.1051751 0.1392269  1.0000000
```

Correlations between gene expressions

```
library(ggcorrplot)
library(viridis)
nci_data %>%
  as_tibble() %>%
  select(1:50) %>%
  cor() %>%
  ggcorrplot() +
  scale_fill_viridis()
```

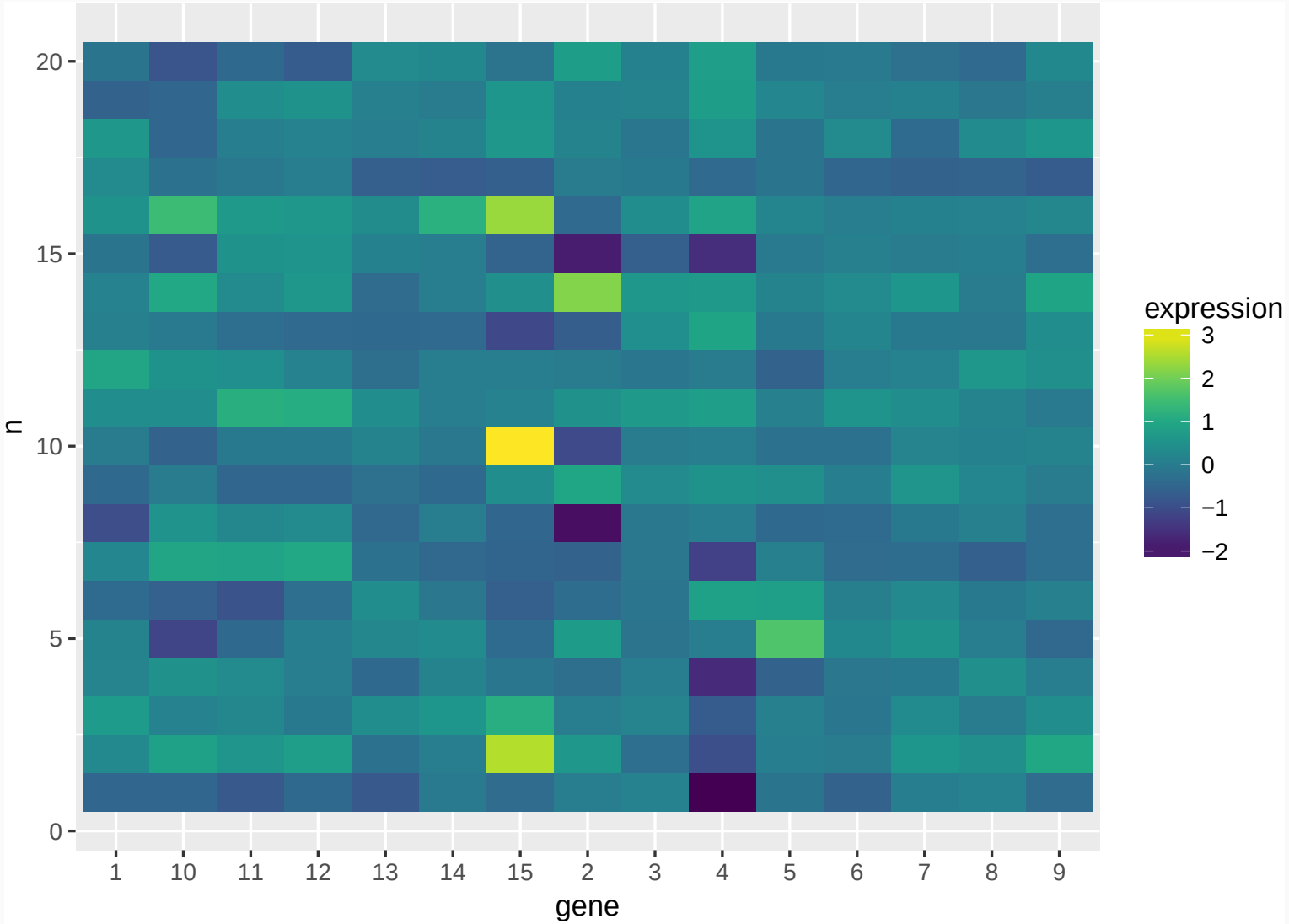
Correlations between gene expressions



Heatmap

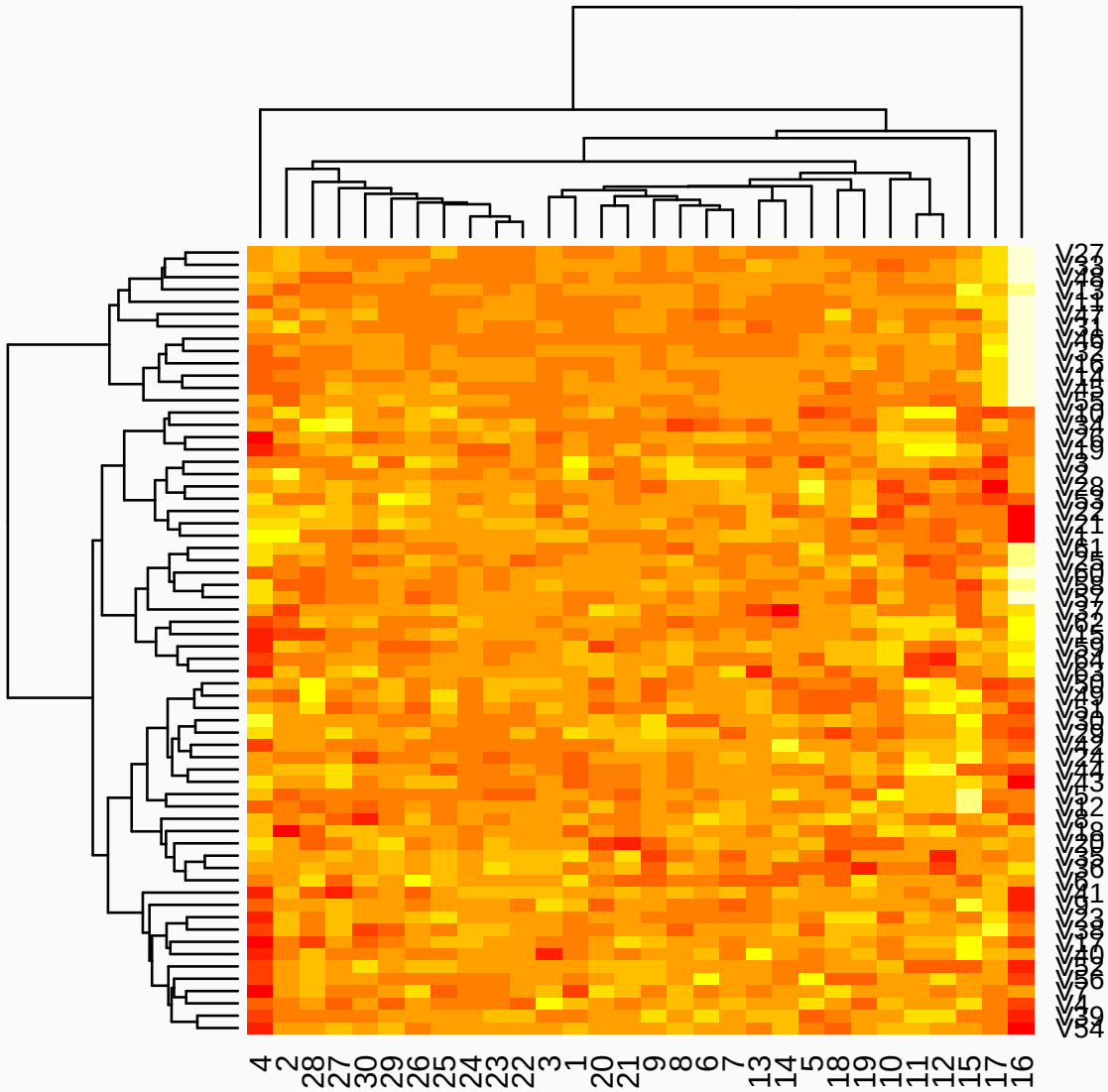
```
nci_data %>%
  as_tibble() %>%
  select(1:15) %>%
  sample_n(20) %>%
  mutate(n = 1:n()) %>%
  gather(key = "gene", value = "expression", -n) %>%
  ggplot(aes(x = gene, y = n, fill = expression)) +
  geom_tile() +
  scale_fill_viridis()
```

Heatmap


20
15
10
5
0
n
1
10
11
12
13
14
15
2
3
4
5
6
7
8
9
gene
expression
3
2
1
0
−1
−2

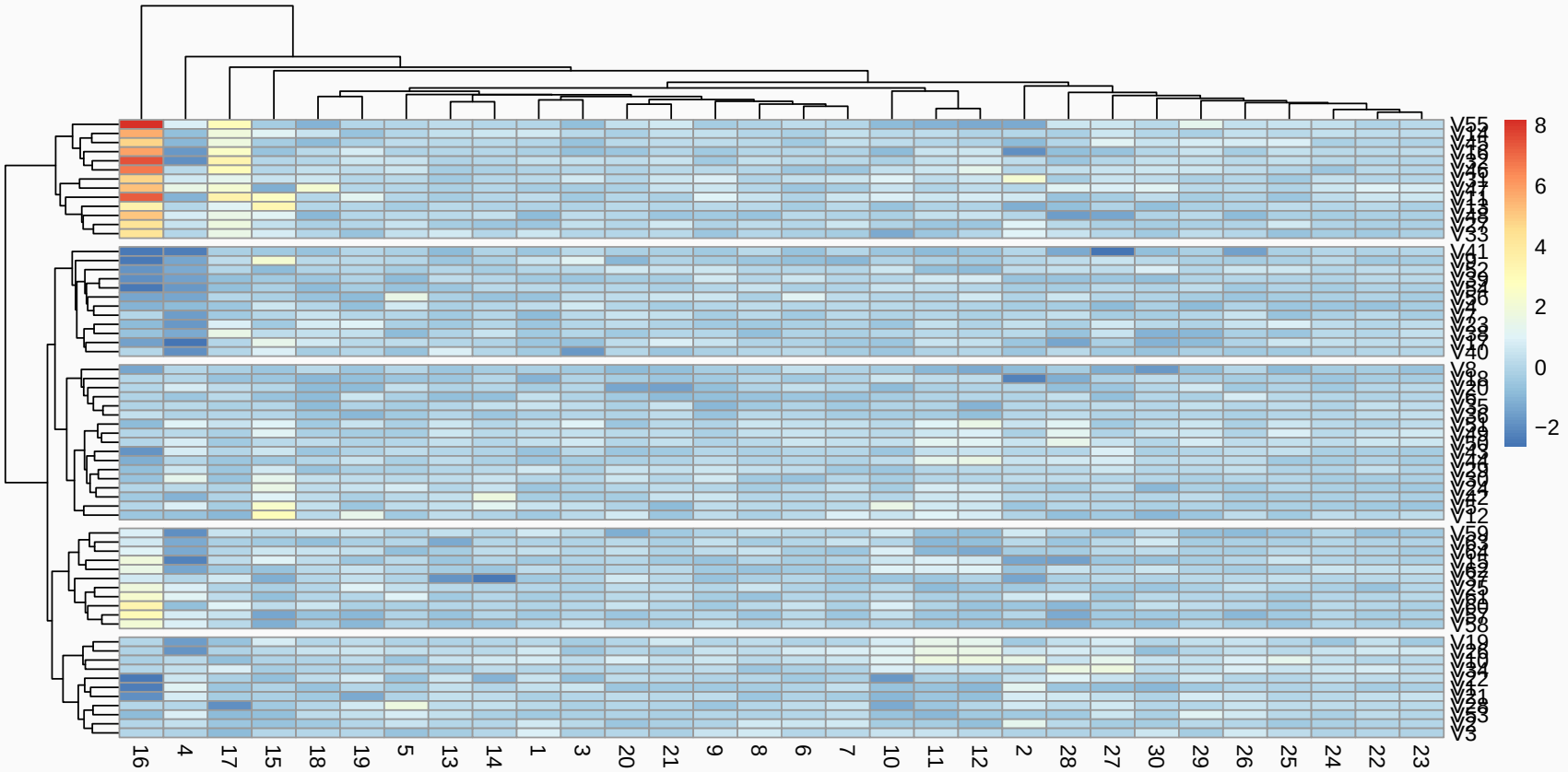
Heatmap

```
heatmap(nci_data[, 1:30])
```



Heatmap

```
library(pheatmap)
pheatmap(nci_data[, 1:30], cutree_rows = 5)
```



Heatmap

R has some other packages for making heatmaps, including:

- `d3heatmap`: Create interactive heatmaps
- `ComplexHeatmap`: Create very complex heatmaps (on Bioconductor)

PCA

Principal components analysis (PCA) can help with exploratory data analysis. If you did pairwise scatterplots of all 6830 gene expressions, you would need loads of plots:

$$\frac{p(p-1)}{2} = \frac{6830(6830-1)}{2} = 2.3321035 \times 10^7$$

PCA

Instead, you can plot pairwise scatterplots of the first few principal components loadings. Based on James et al.,

> *PCA helps create a "low-dimensional representation of the data that captures as much of the information as possible".*

PCA

You can use the `prcomp` function to perform a principal components analysis on a data matrix:

```
pr_out <- prcomp(nci_data, scale = TRUE)
class(pr_out)
```

```
## [1] "prcomp"
```

The output from this function has the class `prcomp`.

PCA

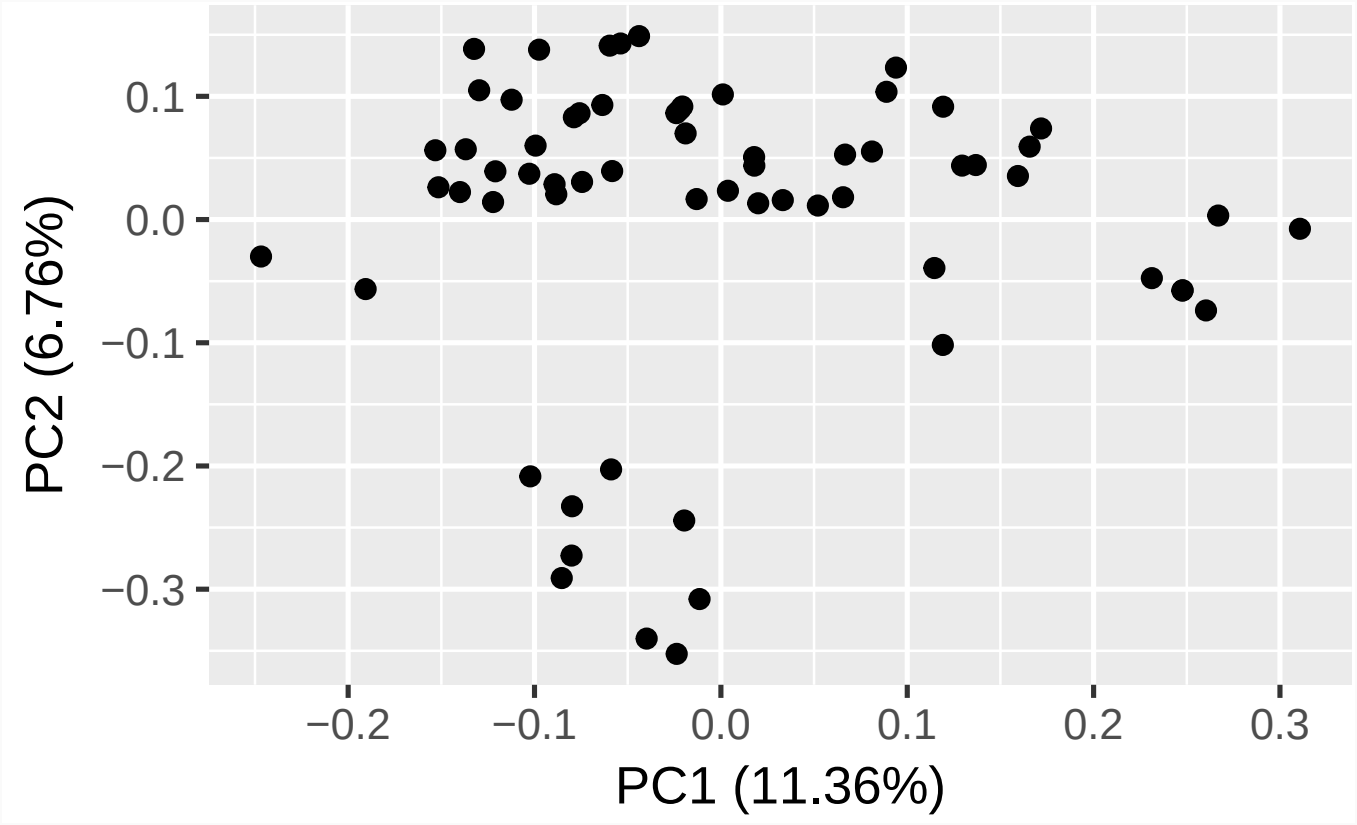
As a reminder, since the output is a special class, it will have special methods of things like `print` and `summary`:

```
## [1] "sdev"       "rotation"   "center"     "scale"      "x"
## [6] "importance"
```

PCA

The `ggfortify` package has an `autoplot` method for `prcomp` class objects. This plots each observation on its values for the first two principal components.

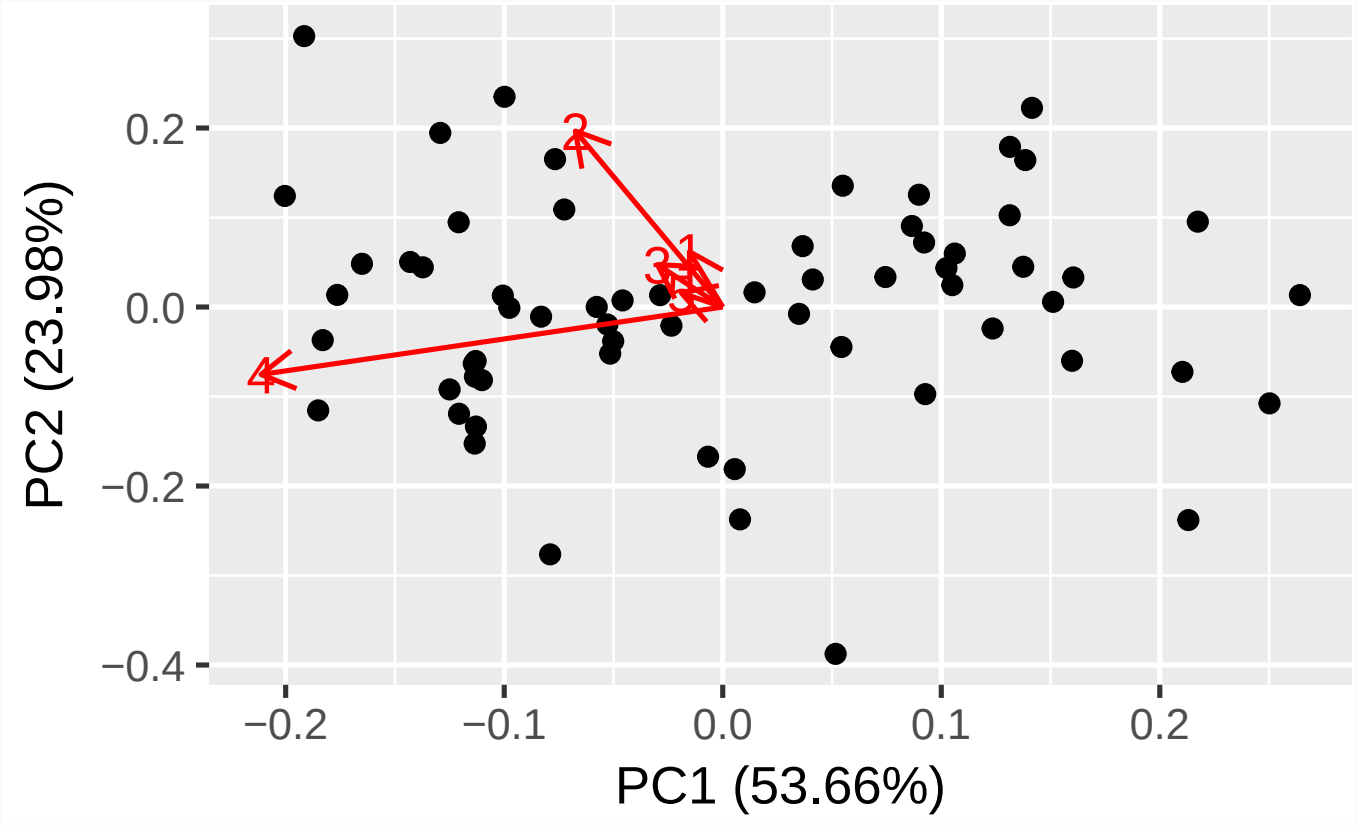
```
library(ggfortify)
autoplot(pr_out)
```



PCA

You can add labels that show the loadings for each variable. However, this is usually only legible if you have a lower number of variables you're considering.

```
autoplot(prcomp(nci_data[ , 1:5]), loadings = TRUE,
         loadings.label = TRUE)
```



PCA

The `$x` element of the output of `prcomp` are the value of the rotated data (i.e., the centered and scaled data multiplied by the rotation matrix):

```
dim(pr_out$x)
```

```
## [1] 64 64
```

```
pr_out$x[1:4, 1:4]
```

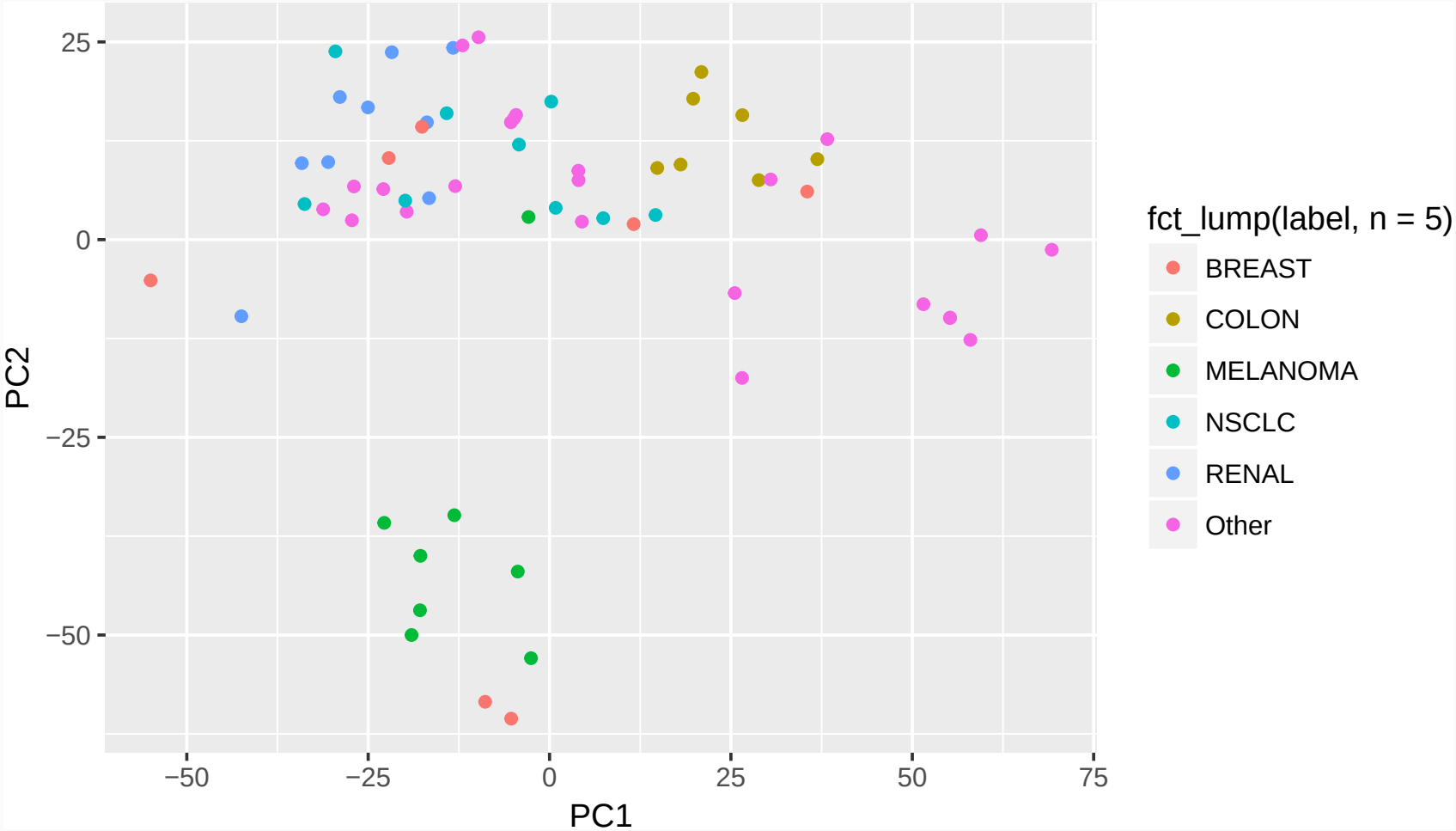
```
##           PC1       PC2          PC3        PC4
## V1 -19.68245  3.527748  -9.7354382  0.8177816
## V2 -22.90812  6.390938 -13.3725378 -5.5911088
## V3 -27.24077  2.445809  -3.5053437  1.3311502
## V4 -42.48098 -9.691742  -0.8830921 -3.4180227
```

PCA

You can pull out the PC values for each observation and plot those, adding the cell type labels with color:

```
library(tibble)
pr_out$x %>%
  as.data.frame() %>%
  select(PC1:PC2) %>%
  rownames_to_column() %>%
  mutate(label = nci_labs) %>%
  ggplot(aes(x = PC1, y = PC2,
             color = fct_lump(label, n = 5))) +
  geom_point()
```

PCA


25
0
−25
−50
PC2
−50
−25
0
25
50
75
PC1
fct_lump(label, n = 5)
BREAST
COLON
MELANOMA
NSCLC
RENAL
Other

PCA

To see the standard deviation explained by the first five components, you can pull out the `sdev` component:

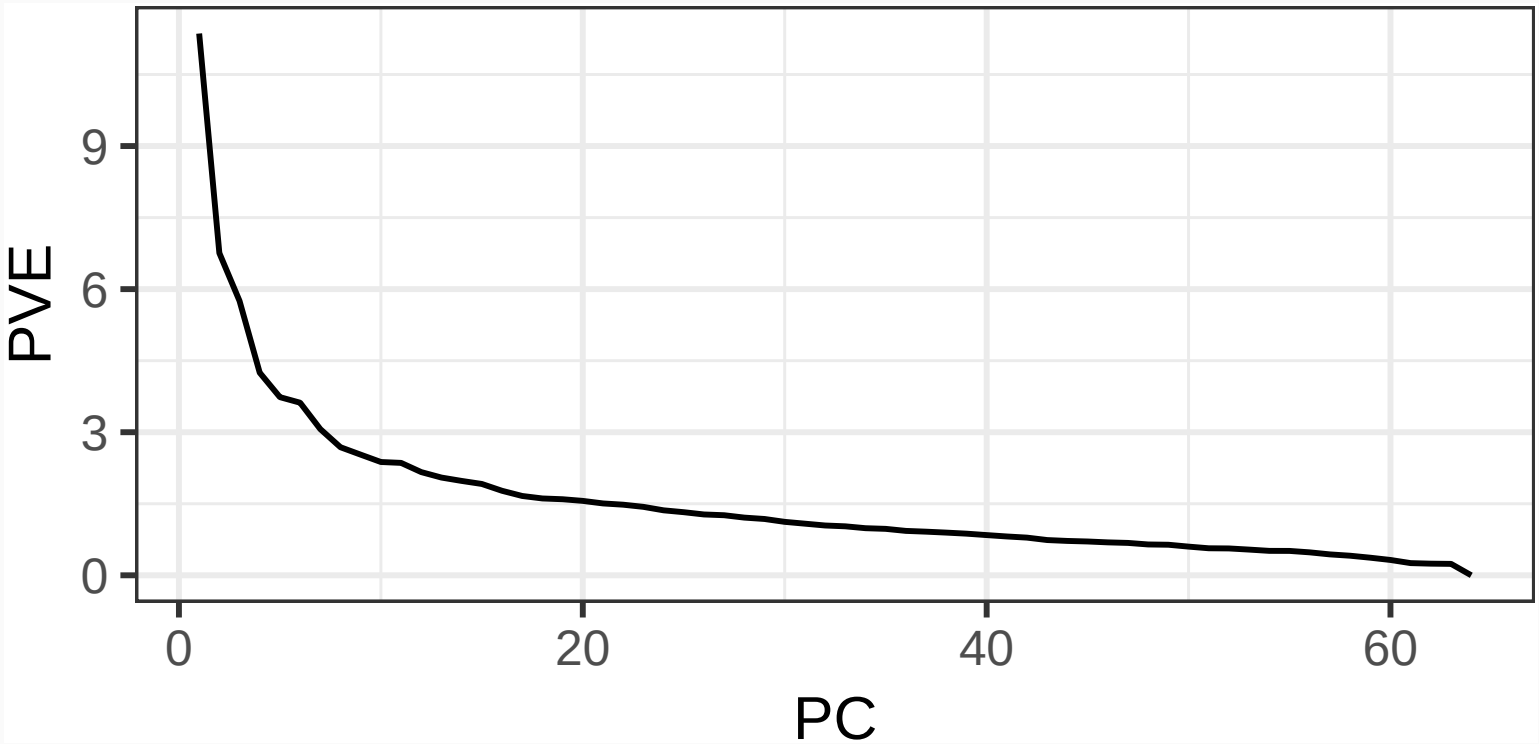
```
pr_out$sdev[1:5]
```

```
## [1] 27.85347 21.48136 19.82046 17.03256 15.97181
```

PCA

To create a scree plot:

```
to_plot <- data.frame(PC = 1:nrow(pr_out$x),
                      PVE = 100 * pr_out$sdev ^ 2 /
                        sum(pr_out$sdev ^ 2))
ggplot(to_plot, aes(x = PC, y = PVE)) + geom_line() +
  theme_bw()
```



PCA

From James et al.:

> *"Unfortunately, there is no well-accepted objective way to decide how many principal components are enough. In Fact, the question of how many principal components are enough is inherently ill-defined, and will depend on the specific area of application and the specific data set."*

Clustering methods

Goal: Create clusters so that the within-cluster variation among observations is as low as possible.

- **Hierarchical clustering**: Create a dendrogram that could be used to pick out clusters of any size.
- **K-means clustering**: Split the observations into a certain number of clusters.

You can cluster observations by features or features by observations.

Hierarchical clustering

Start by standardizing the data:

```
sd_data <- scale(nci_data)
```

Then use the `dist` function to measure Euclidean distance:

```
data_dist <- dist(sd_data, method = "euclidean")
class(data_dist)
```

```
## [1] "dist"
```

Other `method` options: “maximum”, “manhattan”, “canberra”, “binary”, “minkowski”.

Hierarchical clustering

`hclust` can be applied to a `dist` object to identify clusters:

```
nci_clusters <- hclust(data_dist)
names(nci_clusters)
```

```
## [1] "merge"       "height"      "order"       "labels"      "
## [6] "call"        "dist.method"
```

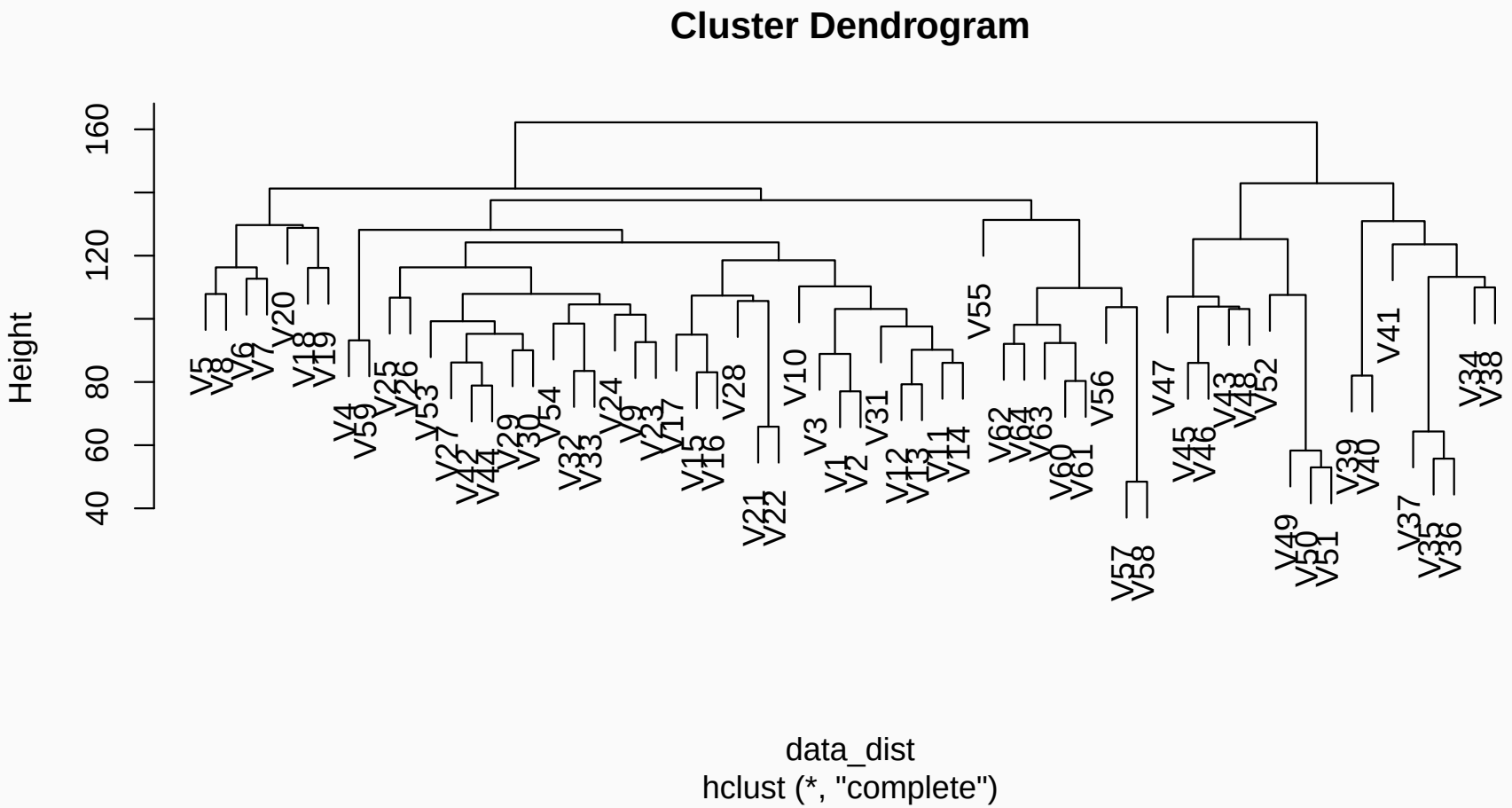
```
class(nci_clusters)
```

```
## [1] "hclust"
```

The default is to cluster using complete linkage.

Hierarchical clustering

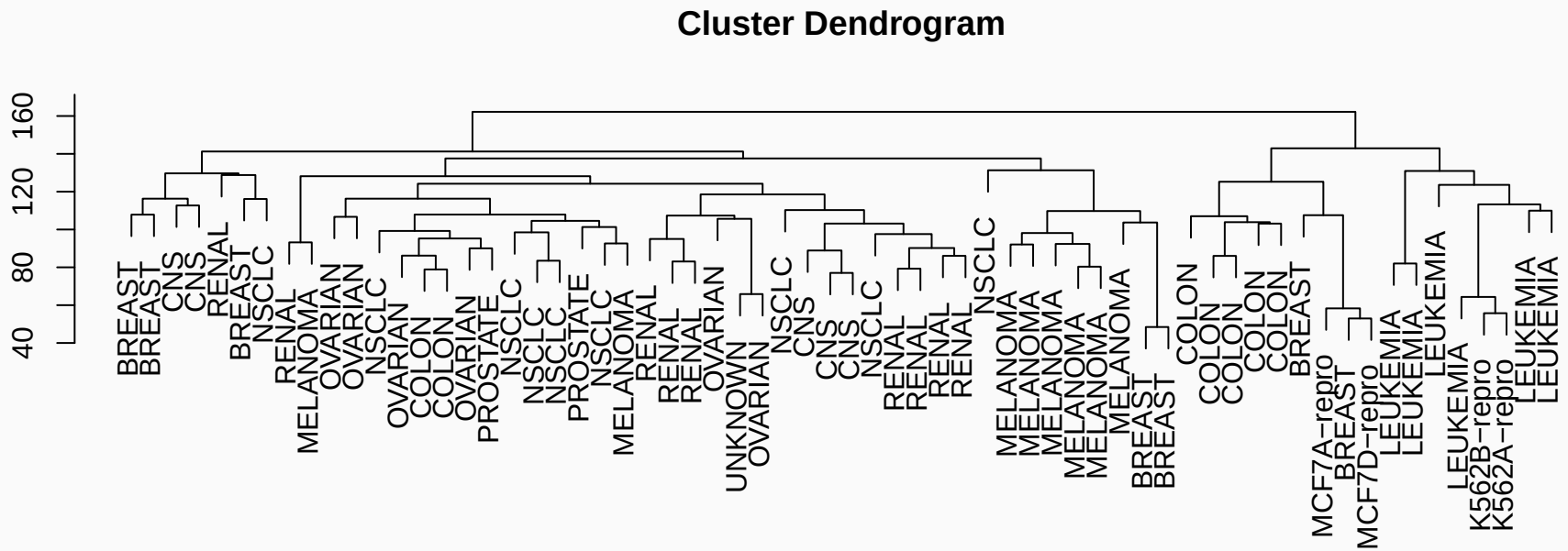
```
plot(nci_clusters)
```



Hierarchical clustering

Use the cancer type for labels:

```
plot(nci_clusters, labels = nci_labs, xlab = "",
     ylab = "", sub = "")
```



Hierarchical clustering

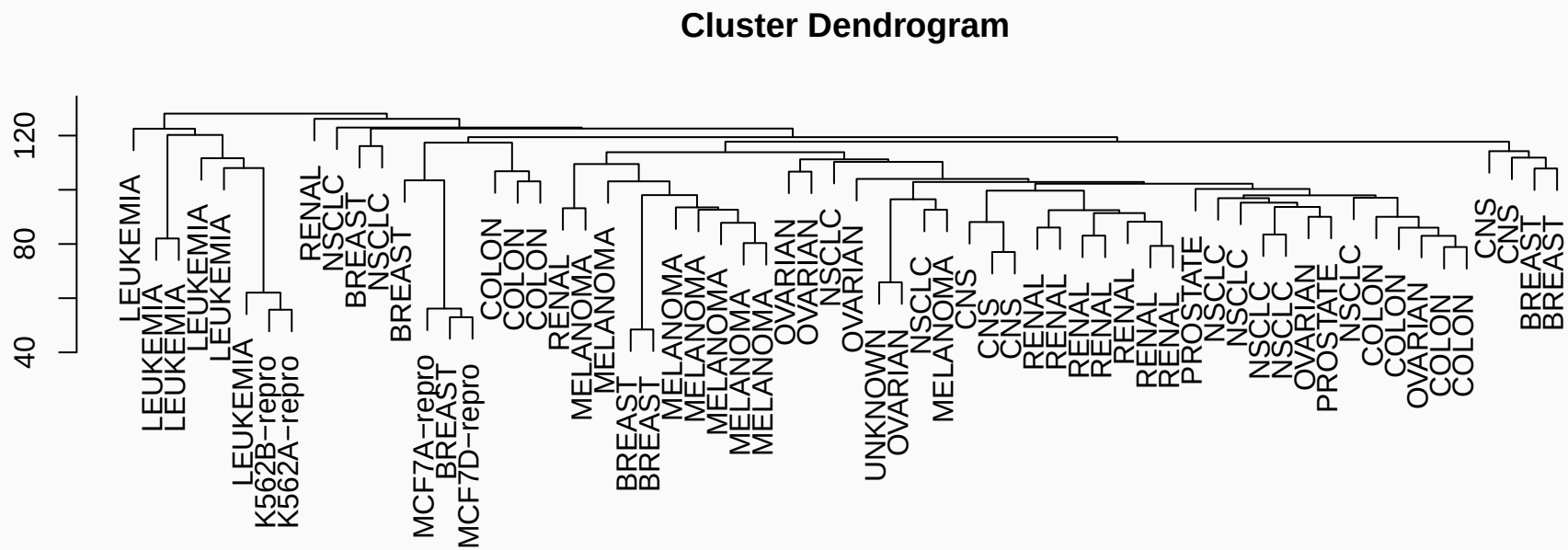
Linkage: The dissimilarity between two groups of observations (see Table 10.2 in James et al.).

- Complete: Largest of all pairwise distances between observations in cluster A and cluster B
- Average: Average of all pairwise distances between observations in cluster A and cluster B
- Single: Smallest of all pairwise distances between observations in cluster A and cluster B
- Centroid: The distance between the centroids of each cluster

Hierarchical clustering

By change the `hclust` arguments, you can use average linkage instead:

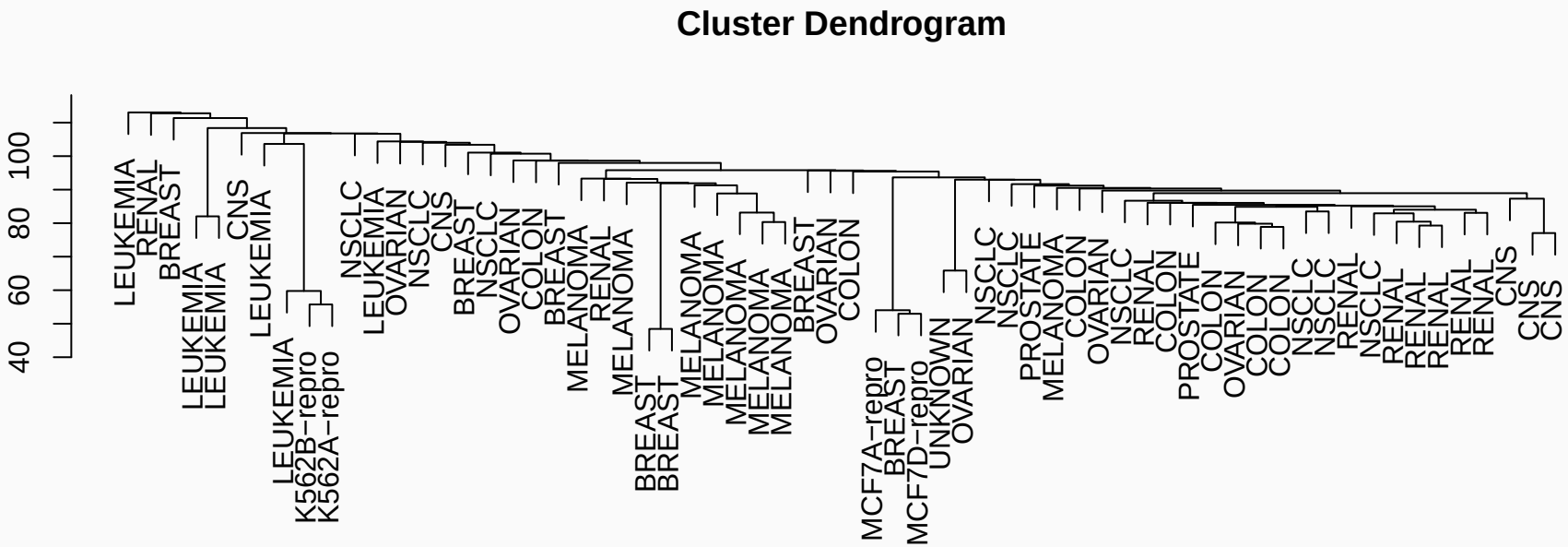
```
plot(hclust(data_dist, method = "average"),
     labels = nci_labs, xlab = "",
     ylab = "", sub = "")
```



Hierarchical clustering

Or single linkage:

```
plot(hclust(data_dist, method = "single"),
     labels = nci_labs, xlab = "",
     ylab = "", sub = "")
```



Cutting down to fewer clusters

You can use the `cutree` function to cut the cluster dendrogram at a certain height to only get a certain number of clusters. For example, to get four clusters:

```
hc_clusters <- cutree(nci_clusters, 4)
hc_clusters
```

```
##  V1  V2  V3  V4  V5  V6  V7  V8  V9 V10 V11 V12 V13 V14 V15 V
##   1   1   1   1   2   2   2   2   1   1   1   1   1   1   1
## V19 V20 V21 V22 V23 V24 V25 V26 V27 V28 V29 V30 V31 V32 V33 V
##   2   2   1   1   1   1   1   1   1   1   1   1   1   1   1
## V37 V38 V39 V40 V41 V42 V43 V44 V45 V46 V47 V48 V49 V50 V51 V
##   3   3   3   3   3   1   4   1   4   4   4   4   4   4   4
## V55 V56 V57 V58 V59 V60 V61 V62 V63 V64
##   1   1   1   1   1   1   1   1   1   1
```

Cutting down to fewer clusters

```
data.frame(cancer = nci_labs, cluster = hc_clusters) %>%
  group_by(cluster) %>%
  summarize(cancers = paste(unique(cancer), collapse = ", ")) %>
  pander(split.cell = 50)
```

cluster	cancers
1	CNS, RENAL, NSCLC, UNKNOWN, OVARIAN, MELANOMA, PROSTATE, COLON, BREAST
2	BREAST, CNS, NSCLC, RENAL
3	LEUKEMIA, K562B-repro, K562A-repro
4	COLON, MCF7A-repro, BREAST, MCF7D-repro