Day 2 – Walk-through

Seminar plotting materials

Use this consolidated walk-through to guide Day 2. Each section corresponds to the standalone notebooks (now under `scripts/individual_notebooks/`) but lives in one file so you can knit or print everything at once.

---

00 Prepare Environment

Use this short notebook (or just run the snippets below) before the workshop to ensure all required packages are installed. Keep it simple: install everything in one go, then load each package once to check that it works.

```
if (!require("BiocManager", quietly = TRUE)) {
  install.packages("BiocManager")
}
BiocManager::install("ComplexHeatmap")
install.packages(c('circlize', 'tidyr', 'dplyr'))
```

```
library(ComplexHeatmap)
library(circlize)
library(tidyr)
library(dplyr)
```

For reference documentation and additional examples, bookmark the official ComplexHeatmap book: https://jokergoo.github.io/ComplexHeatmap-reference/book/.

After installing, you can knit `second_day_part2/data/00_prepare_dataset.Rmd` to create the teaching datasets (if needed) and move on to the exploration scripts.

01 Explore Data

We start the Day 2 session with the already prepared file `second_day_part2/data/dataset1_subset_long.csv`. This document simply shows how to open that file with base R commands and how to inspect what is inside before we move on to plotting. Run `second_day_part2/data/00_prepare_dataset.Rmd` first if you need to regenerate the CSV. Knit it from the repository root with:

```
R -e "rmarkdown::render('second_day_part2/scripts/individual_notebooks/01_explore_data.Rmd')"
```

1. Load the CSV

```
input_path <- file.path('..', 'data', 'dataset1_subset_long.csv')
long_df <- read.csv(input_path, stringsAsFactors = FALSE, check.names = FALSE)

cat('Rows:', nrow(long_df), '\\nColumns:', ncol(long_df), '\\n')
```

```
## Rows: 1536 \nColumns: 7 \n
```

2. Look at column names and a small preview

`colnames()` lists the headers exactly as they appear in the CSV and `head()` prints the first few rows. Encourage learners to read these outputs aloud so they begin to map column names to real-world meaning (genome name, SNP id, mouse ID, day, value, …).

```
colnames(long_df)
```

```
## [1] "Genome"          "snp_id"          "Position"        "value"
## [5] "mouse_id"        "day"             "treatment_group"
```

```
head(long_df)
```

```
##                         Genome      snp_id Position    value mouse_id day
## 1 Akkermansia_muciniphila_YL44  239840-C-G   239840 0.000000     1683   0
## 2 Akkermansia_muciniphila_YL44  241793-A-G   241793 0.049587     1683   0
## 3 Akkermansia_muciniphila_YL44  355328-A-T   355328 0.138182     1683   0
## 4 Akkermansia_muciniphila_YL44  356291-C-A   356291 0.000000     1683   0
## 5 Akkermansia_muciniphila_YL44 2351445-C-T  2351445 0.000000     1683   0
## 6   Bacteroides_caecimuris_I48 1601848-T-C  1601848 0.041609     1683   0
##   treatment_group
## 1         Control
## 2         Control
## 3         Control
## 4         Control
## 5         Control
## 6         Control
```

It can also help to peek at the SNP identifiers themselves to remind everyone what a typical label looks like:

```
head(unique(long_df$snp_id), n = 5)
```

```
## [1] "239840-C-G"  "241793-A-G"  "355328-A-T"  "356291-C-A"  "2351445-C-T"
```

3. Which genomes are present?

We often start by asking "Which genomes are represented here?". `unique()` returns all distinct values, while `table()` counts how many rows belong to each genome.

```
unique(long_df$Genome)
```

```
## [1] "Akkermansia_muciniphila_YL44" "Bacteroides_caecimuris_I48"
```

```
table(long_df$Genome)
```

```
##
## Akkermansia_muciniphila_YL44   Bacteroides_caecimuris_I48
##                          320                         1216
```

Explain to students that each row contains the `Genome`, the unique SNP id, the mouse ID, the day of sampling, and the measured value.

4. How many SNPs per genome?

Counting the unique SNP identifiers per genome shows how much information we keep for each organism. The helper `tapply(values, groups, FUN)` applies a function to each subset of `values` defined by `groups`.

Here we give it the vector of SNP ids and ask it to count how many unique ids (`length(unique(x))`) exist inside each genome group.

```
snp_per_genome <- tapply(long_df$snp_id, long_df$Genome, function(x) length(unique(x)))
snp_per_genome
```

```
## Akkermansia_muciniphila_YL44   Bacteroides_caecimuris_I48
##                            5                           19
```

6. How many measurements per mouse and per day?

```
table(long_df$mouse_id)
```

```
##
## 1683 1688 1692 1699
##  384  384  384  384
```

```
table(long_df$day)
```

```
##
##  0  4  9 14 18 23 30 37 44 49 53 58 63 67 72 79
## 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96
```

For a combined view you can also build a two-way table. Again we use `table()`, this time with `mouse_id` on rows and `day` on columns. Reading across a row shows how many time points we have for a given mouse.

```
table(long_df$mouse_id, long_df$day)
```

```
##
##         0  4  9 14 18 23 30 37 44 49 53 58 63 67 72 79
##   1683 24 24 24 24 24 24 24 24 24 24 24 24 24 24 24 24
##   1688 24 24 24 24 24 24 24 24 24 24 24 24 24 24 24 24
##   1692 24 24 24 24 24 24 24 24 24 24 24 24 24 24 24 24
##   1699 24 24 24 24 24 24 24 24 24 24 24 24 24 24 24 24
```

Pick any cell and inspect those rows directly. For example, mouse 1683 on day 0 has the following measurements (all SNPs for that mouse/day):

```
subset_example <- long_df[long_df$mouse_id == '1683' & long_df$day == 0, ]
subset_example
```

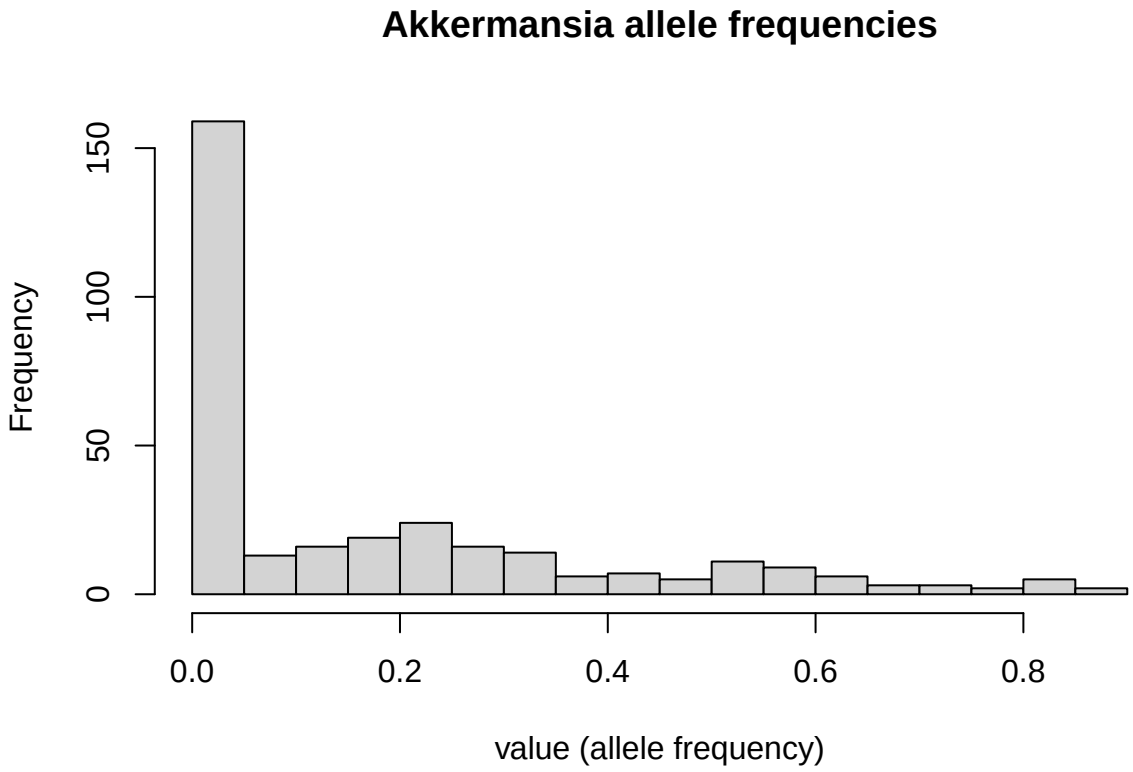
```
##                          Genome      snp_id Position    value mouse_id day
## 1  Akkermansia_muciniphila_YL44  239840-C-G   239840 0.000000     1683   0
## 2  Akkermansia_muciniphila_YL44  241793-A-G   241793 0.049587     1683   0
## 3  Akkermansia_muciniphila_YL44  355328-A-T   355328 0.138182     1683   0
## 4  Akkermansia_muciniphila_YL44  356291-C-A   356291 0.000000     1683   0
## 5  Akkermansia_muciniphila_YL44 2351445-C-T  2351445 0.000000     1683   0
## 6    Bacteroides_caecimuris_I48 1601848-T-C  1601848 0.041609     1683   0
## 7    Bacteroides_caecimuris_I48 1602949-A-G  1602949 0.000000     1683   0
## 8    Bacteroides_caecimuris_I48 2105155-C-T  2105155 0.113208     1683   0
## 9    Bacteroides_caecimuris_I48 2562529-T-C  2562529 0.000000     1683   0
## 10   Bacteroides_caecimuris_I48 2958703-C-A  2958703 0.000000     1683   0
## 11   Bacteroides_caecimuris_I48 2963503-C-A  2963503 0.000000     1683   0
## 12   Bacteroides_caecimuris_I48 2964459-G-T  2964459 0.000000     1683   0
## 13   Bacteroides_caecimuris_I48 2973657-A-G  2973657 0.000000     1683   0
## 14   Bacteroides_caecimuris_I48 2975719-A-G  2975719 0.000000     1683   0
## 15   Bacteroides_caecimuris_I48 2981479-T-G  2981479 0.000000     1683   0
## 16   Bacteroides_caecimuris_I48 2996527-C-A  2996527 0.000000     1683   0
```

```
## 17   Bacteroides_caecimuris_I48 2996770-A-G  2996770 0.000000     1683   0
## 18   Bacteroides_caecimuris_I48 3004527-A-T  3004527 0.000000     1683   0
## 19   Bacteroides_caecimuris_I48 3004659-G-A  3004659 0.000000     1683   0
## 20   Bacteroides_caecimuris_I48 3004706-A-G  3004706 0.000000     1683   0
## 21   Bacteroides_caecimuris_I48 3005011-C-A  3005011 0.000000     1683   0
## 22   Bacteroides_caecimuris_I48 3009589-C-T  3009589 0.000000     1683   0
## 23   Bacteroides_caecimuris_I48 3623124-G-A  3623124 0.173591     1683   0
## 24   Bacteroides_caecimuris_I48 4472955-A-G  4472955 0.000000     1683   0
##    treatment_group
## 1          Control
## 2          Control
## 3          Control
## 4          Control
## 5          Control
## 6          Control
## 7          Control
## 8          Control
## 9          Control
## 10         Control
## 11         Control
## 12         Control
## 13         Control
## 14         Control
## 15         Control
## 16         Control
## 17         Control
## 18         Control
## 19         Control
## 20         Control
## 21         Control
## 22         Control
## 23         Control
## 24         Control
```

6. What do the allele-frequency values look like?

Start by focusing on a single genome (Akkermansia) so the histogram is easy to interpret, then compare it against the combined dataset.

```
akk_values <- long_df$value[long_df$Genome == 'Akkermansia_muciniphila_YL44']
hist(akk_values, breaks = 20, main = 'Akkermansia allele frequencies', xlab = 'value (allele frequency)
```

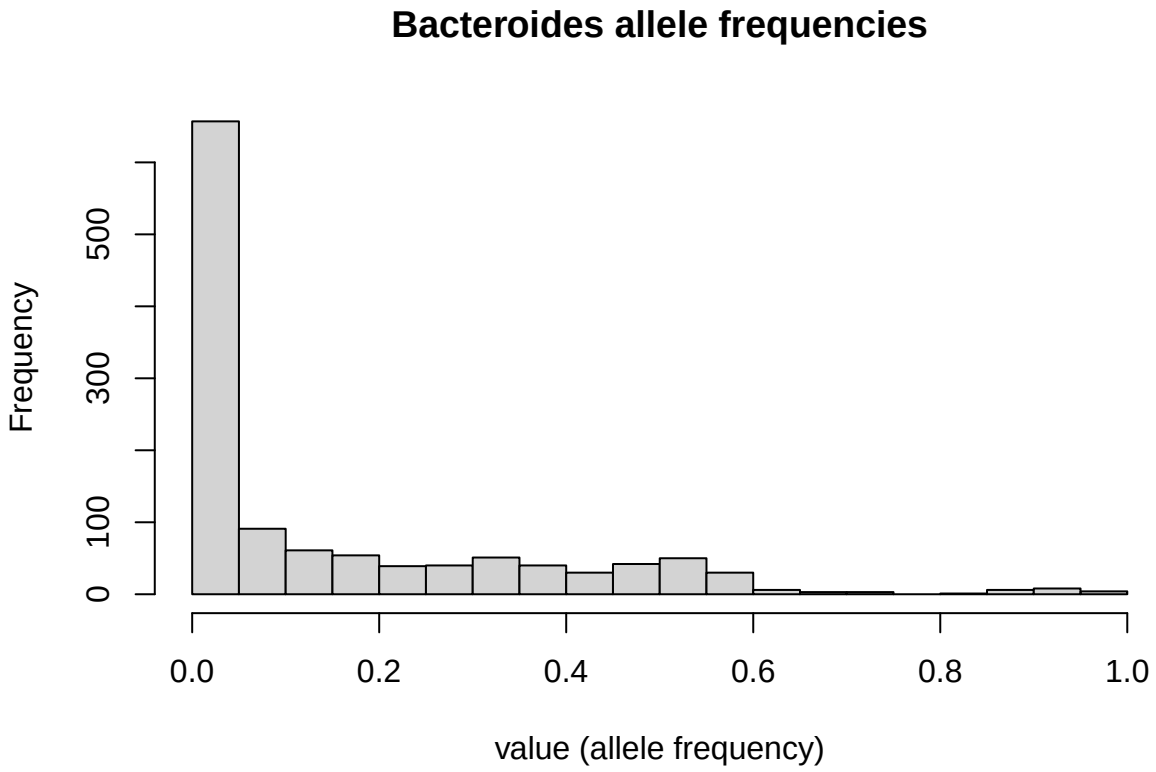



```
summary(akk_values)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.00000 0.05231 0.17419 0.27145 0.87125
```

Do the same for Bacteroides (the other genome in this dataset).

```
bacto_values <- long_df$value[long_df$Genome == 'Bacteroides_caecimuris_I48']
hist(bacto_values, breaks = 20, main = 'Bacteroides allele frequencies', xlab = 'value (allele frequenc
```

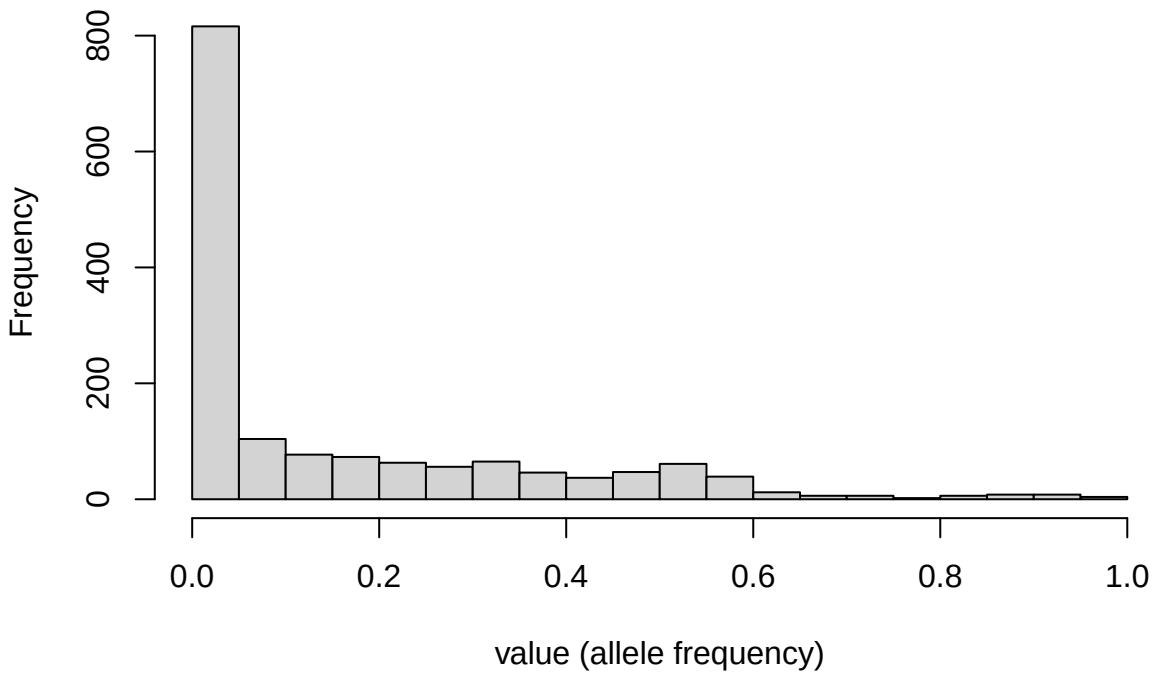



```
summary(bacto_values)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.00000 0.03383 0.14854 0.26792 0.99122
```

Now look at the complete long table to see how the mixture of genomes changes the distribution.

```
hist(long_df$value, breaks = 30, main = 'All genomes: allele-frequency histogram', xlab = 'value (allele
```




```
summary(long_df$value)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.00000 0.03941 0.15388 0.27070 0.99122
```

To compare genomes explicitly, break the summary down by genome using `aggregate(value ~ Genome, ...)`. Read `value ~ Genome` as "value grouped by Genome". The column on the left must be numeric because we are calling `summary()` on it.

```
value_by_genome <- aggregate(value ~ Genome, data = long_df, function(x) summary(x))
value_by_genome
```

```
##                         Genome value.Min. value.1st Qu. value.Median value.Mean
## 1 Akkermansia_muciniphila_YL44  0.0000000     0.0000000    0.0523090  0.1741893
## 2   Bacteroides_caecimuris_I48  0.0000000     0.0000000    0.0338305  0.1485411
##   value.3rd Qu. value.Max.
## 1     0.2714463  0.8712450
## 2     0.2679232  0.9912180
```

7. How many missing values?

Count how many `NA` values appear in `value` and, if any, locate them by mouse/day.

```
na_total <- sum(is.na(long_df$value))
na_total
```

```
## [1] 0
```

```
na_by_mouse_day <- with(long_df, tapply(value, list(mouse_id, day), function(x) sum(is.na(x))))
na_by_mouse_day
```

```
##      0 4 9 14 18 23 30 37 44 49 53 58 63 67 72 79
## 1683 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

```
## 1688 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0
## 1692 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0
## 1699 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

8. Treatment-group overview

The long table now contains a `treatment_group` column. Start by listing the distinct labels and counting how many rows belong to each group.

```
unique(long_df$treatment_group)
```

```
## [1] "Control"       "Ciprofloxacin" "Tetracyclin"   "Vancomycin"
```

```
table(long_df$treatment_group)
```

```
##
## Ciprofloxacin       Control   Tetracyclin    Vancomycin
##           384           384           384           384
```

Which treatment groups appear for each mouse? Use a cross-tabulation to reinforce how mice were assigned.

```
table(long_df$mouse_id, long_df$treatment_group)
```

```
##
##        Ciprofloxacin Control Tetracyclin Vancomycin
##   1683             0     384           0          0
##   1688           384       0           0          0
##   1692             0       0         384          0
##   1699             0       0           0        384
```

You can also inspect specific days. For example, day 30 only:

```
subset_day30 <- long_df[long_df$day == 30, c('mouse_id', 'treatment_group')]
unique(subset_day30)
```

```
##      mouse_id treatment_group
## 97       1683         Control
## 481      1688   Ciprofloxacin
## 865      1692     Tetracyclin
## 1249     1699      Vancomycin
```

9. Focus on a single genome (Akkermansia)

Subsetting is a great way to answer specific questions. Below we keep only `Akkermansia_muciniphila_YL44` (using a logical comparison inside the square brackets) and inspect the resulting table. This demonstrates how to focus on one organism without altering the original data frame.

Explain that `long_df$Genome == 'Akkermansia_muciniphila_YL44'` evaluates to a vector of TRUE/FALSE values—TRUE where the genome matches that text and FALSE everywhere else. When we place that logical vector inside `[...]`, R keeps only the TRUE rows.

```
akk_df <- long_df[long_df$Genome == 'Akkermansia_muciniphila_YL44', ]
cat('Rows for Akkermansia:', nrow(akk_df), '\n')
```

```
## Rows for Akkermansia: 320
```

```
head(akk_df)
```

```
##                          Genome      snp_id Position    value mouse_id day
## 1  Akkermansia_muciniphila_YL44  239840-C-G   239840 0.000000     1683   0
```

```
## 2  Akkermansia_muciniphila_YL44  241793-A-G   241793 0.049587     1683   0
## 3  Akkermansia_muciniphila_YL44  355328-A-T   355328 0.138182     1683   0
## 4  Akkermansia_muciniphila_YL44  356291-C-A   356291 0.000000     1683   0
## 5  Akkermansia_muciniphila_YL44 2351445-C-T  2351445 0.000000     1683   0
## 25 Akkermansia_muciniphila_YL44  239840-C-G   239840 0.000000     1683  14
##    treatment_group
## 1          Control
## 2          Control
## 3          Control
## 4          Control
## 5          Control
## 25         Control
```

```
# After subsetting we can still use `table()` to see how measurements are
# distributed for this genome alone.
table(akk_df$mouse_id, akk_df$day)
```

```
##
##        0 4 9 14 18 23 30 37 44 49 53 58 63 67 72 79
##   1683 5 5 5  5  5  5  5  5  5  5  5  5  5  5  5  5
##   1688 5 5 5  5  5  5  5  5  5  5  5  5  5  5  5  5
##   1692 5 5 5  5  5  5  5  5  5  5  5  5  5  5  5  5
##   1699 5 5 5  5  5  5  5  5  5  5  5  5  5  5  5  5
```

```
head(long_df$Genome == 'Akkermansia_muciniphila_YL44')
```

```
## [1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
```

Reading this out loud ("TRUE, TRUE, FALSE, …") reinforces that the comparison creates a logical mask; only the TRUE rows survive when we subset with `[...]`.

Having these quick checks documented makes it easy to reassure learners that we understand the input file before moving into the plotting notebook (`second_day_part2/scripts/individual_notebooks/02_simple_heatmap.Rmd`).

02 Simple Heatmap

A minimal example that turns the prepared wide table (`dataset1_subset.csv`) into a heatmap with practically no extra formatting. This is the first step before moving on to the fully annotated version in `03_heatmap_annotations.Rmd`.

1. Packages and paths

```
library(ComplexHeatmap)
library(circlize)
library(viridisLite)
subset_path <- file.path('..', 'data', 'dataset1_subset.csv')
long_path <- file.path('..', 'data', 'dataset1_subset_long.csv')
pdf_path <- file.path('..', 'pdf', 'dataset1_heatmap_basic.pdf')
target_group <- 'Control'  # change this to view other treatment groups
na_color <- '#dcdcdc'
```

2. Load, subset by treatment group, and convert to matrices

```
wide_df <- read.csv(subset_path, check.names = FALSE, stringsAsFactors = FALSE)
long_df <- read.csv(long_path, check.names = FALSE, stringsAsFactors = FALSE)
```

```r
sample_meta <- unique(long_df[, c('mouse_id', 'day', 'treatment_group')])
sample_meta$sample_id <- paste(sample_meta$mouse_id, sample_meta$day, sep = '-')
sample_cols <- setdiff(names(wide_df), c('Genome', 'snp_id', 'Position'))

mat_all <- as.matrix(wide_df[, sample_cols])
mode(mat_all) <- 'numeric'
rownames(mat_all) <- paste(wide_df$Genome, wide_df$snp_id, sep = ' | ')
colnames(mat_all) <- sample_cols

keep_cols <- intersect(sample_cols, sample_meta$sample_id[sample_meta$treatment_group == target_group])
if (!length(keep_cols)) {
  stop('No samples found for treatment group: ', target_group)
}
heatmap_matrix <- mat_all[, keep_cols, drop = FALSE]
colnames(heatmap_matrix) <- keep_cols
```

3. Choose a data-aware color scale

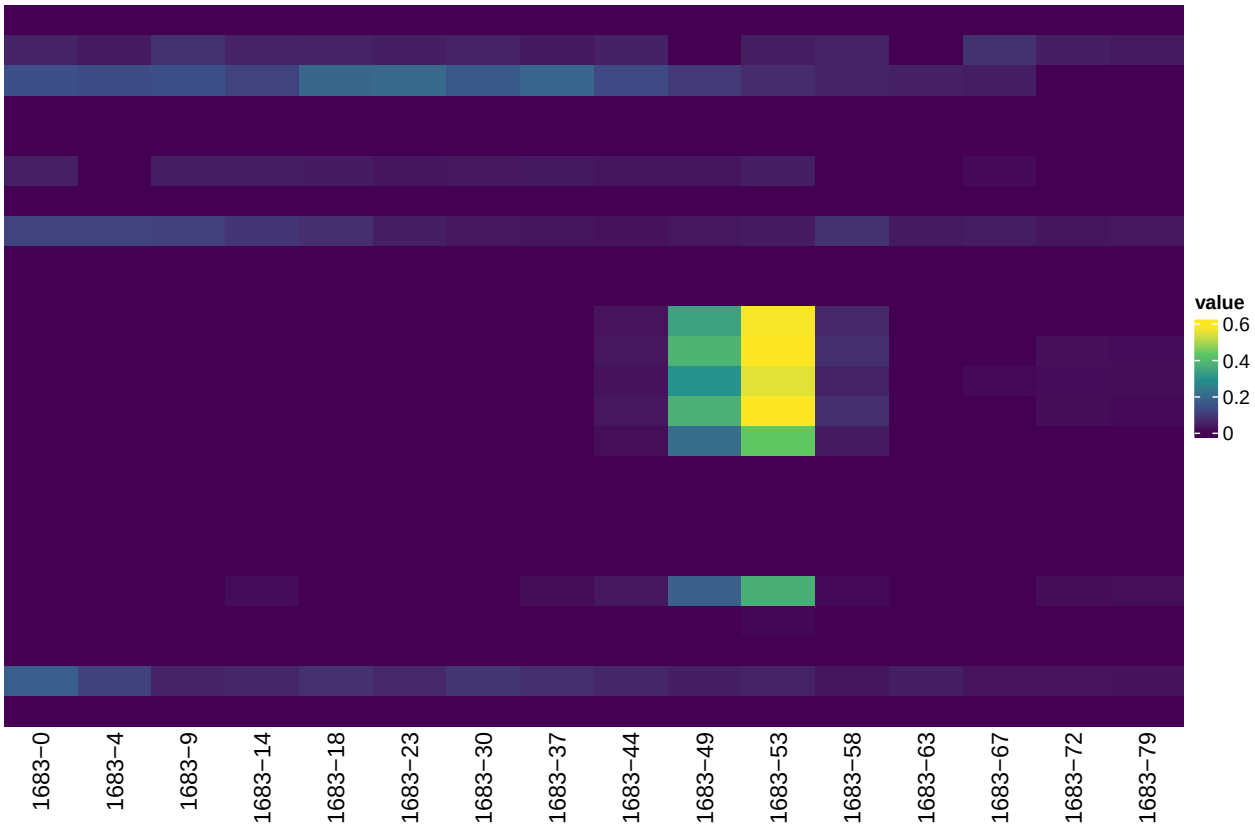
Allele frequencies naturally lie between 0 and 1. Using a diverging palette with an assumed midpoint (e.g., 0.5) suggests there is a special reference level, which is misleading for these measurements. Instead, derive a sequential palette from the observed range so color intensity increases smoothly with the allele frequency.

```r
value_range <- range(heatmap_matrix, na.rm = TRUE)
color_breaks <- seq(value_range[1], value_range[2], length.out = 5)
color_fun <- circlize::colorRamp2(color_breaks, viridisLite::viridis(5))
```

Feel free to swap in other sequential palettes (e.g., `RColorBrewer`) as long as the hues progress monotonically with the data.

4. Draw the simplest possible heatmap

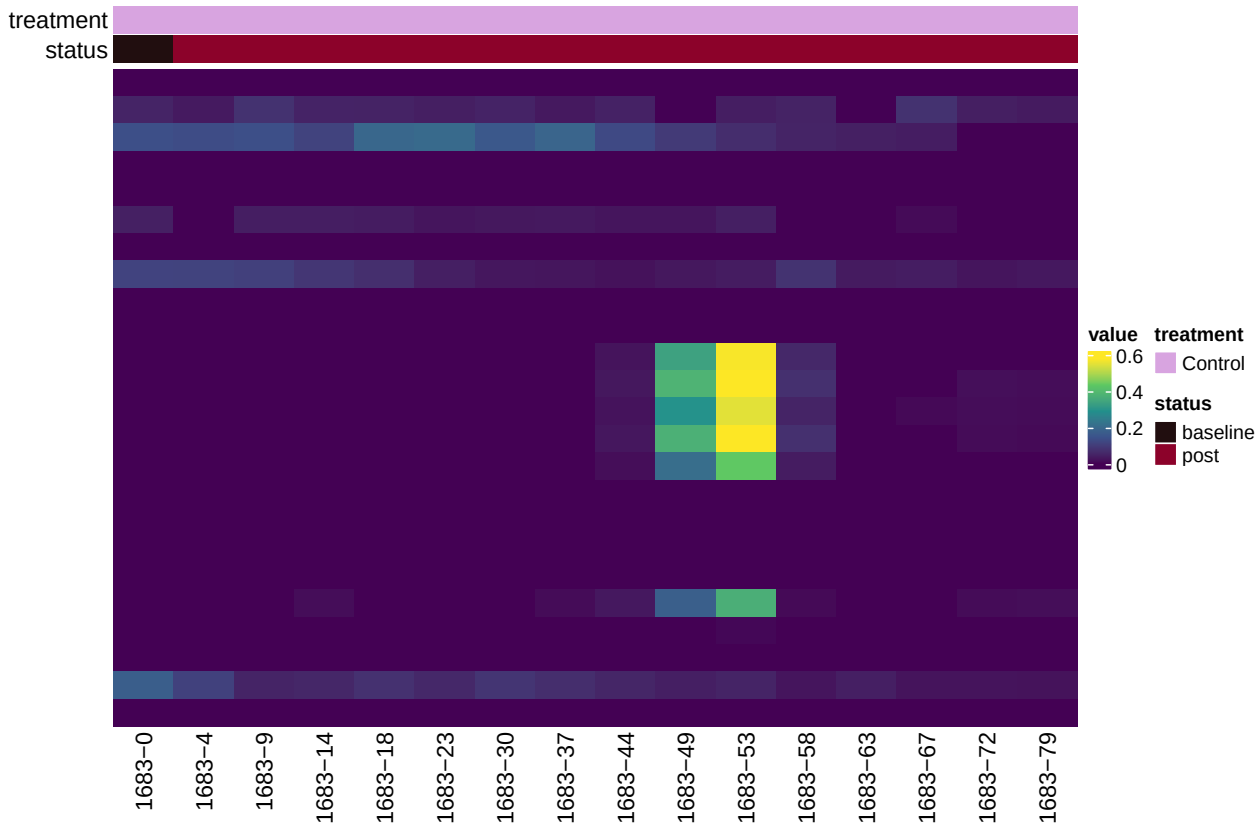
```r
ht <- Heatmap(
  heatmap_matrix,
  name = 'value',
  col = color_fun,
  cluster_rows = FALSE,
  cluster_columns = FALSE,
  show_row_names = FALSE,
  show_column_names = TRUE,
  na_col = na_color
)

draw(ht)
```



5. Optional: highlight treatment and baseline/post status

Before moving on, you can add a light-weight column annotation labeling each sample with its treatment group and whether it was collected at baseline (day 0) or post-antibiotic (day > 0).

```
sample_info <- sample_meta[sample_meta$sample_id %in% colnames(heatmap_matrix), ]
sample_info <- sample_info[match(colnames(heatmap_matrix), sample_info$sample_id), ]
sample_info$post_ab <- ifelse(sample_info$day == 0, 'baseline', 'post')

col_ann <- HeatmapAnnotation(
  treatment = sample_info$treatment_group,
  status = sample_info$post_ab,
  annotation_name_side = 'left'
)

ht_ann <- Heatmap(
  heatmap_matrix,
  name = 'value',
  col = color_fun,
  cluster_rows = FALSE,
  cluster_columns = FALSE,
  show_row_names = FALSE,
  show_column_names = TRUE,
  na_col = na_color,
  top_annotation = col_ann
)

draw(ht_ann)
```



6. Optional: compare two treatment groups side-by-side

If you want to compare two treatment groups visually, filter the full matrix to those samples and split the columns by treatment. This keeps the code simple while showing how cohorts differ.

```
compare_groups <- c('Control', 'Ciprofloxacin')
keep_compare <- sample_meta$sample_id[sample_meta$treatment_group %in% compare_groups]
compare_mat <- mat_all[, keep_compare, drop = FALSE]
group_factor <- factor(sample_meta$treatment_group[match(colnames(compare_mat), sample_meta$sample_id)]
                       levels = compare_groups)
Heatmap(compare_mat,
        name = 'value',
        col = color_fun,
        column_split = group_factor,
        cluster_rows = FALSE,
        cluster_columns = FALSE,
        show_row_names = FALSE,
        na_col = na_color) |>
  draw()
```

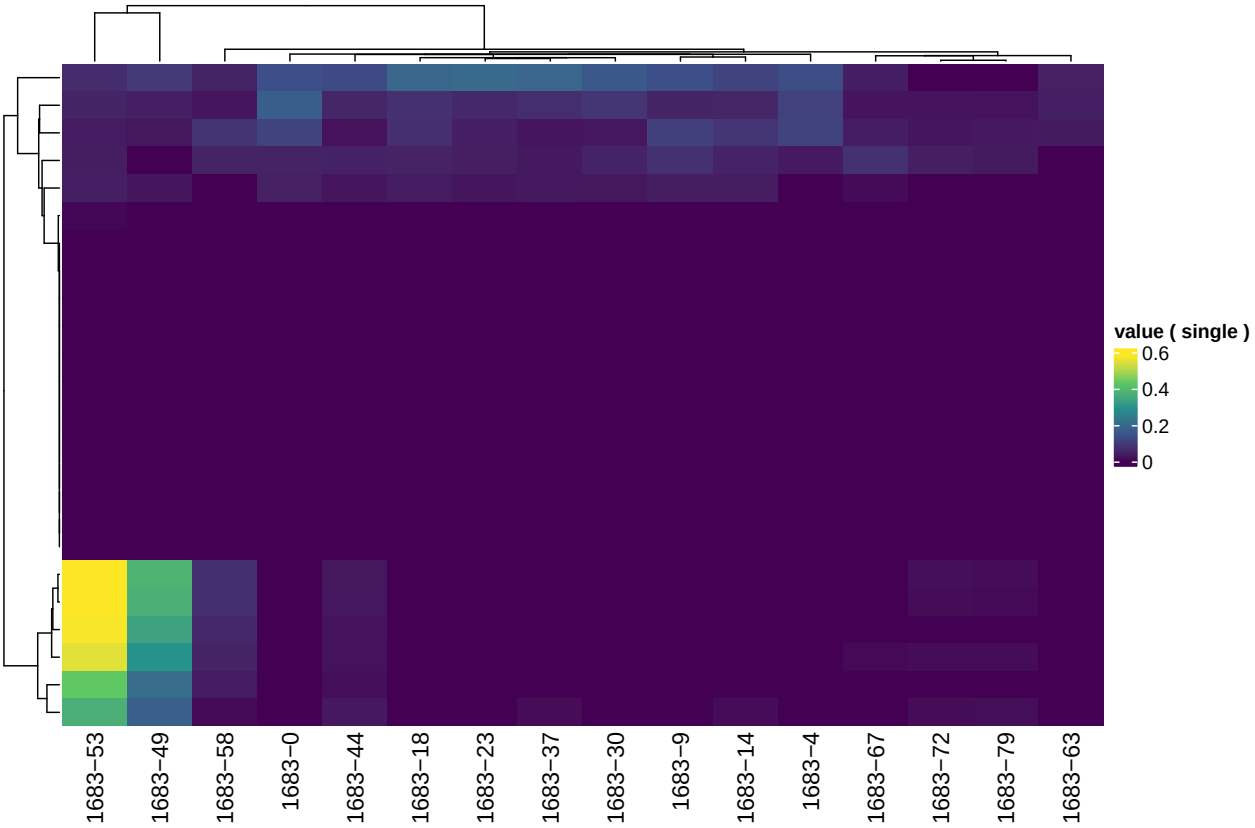
7. Optional: save to PDF

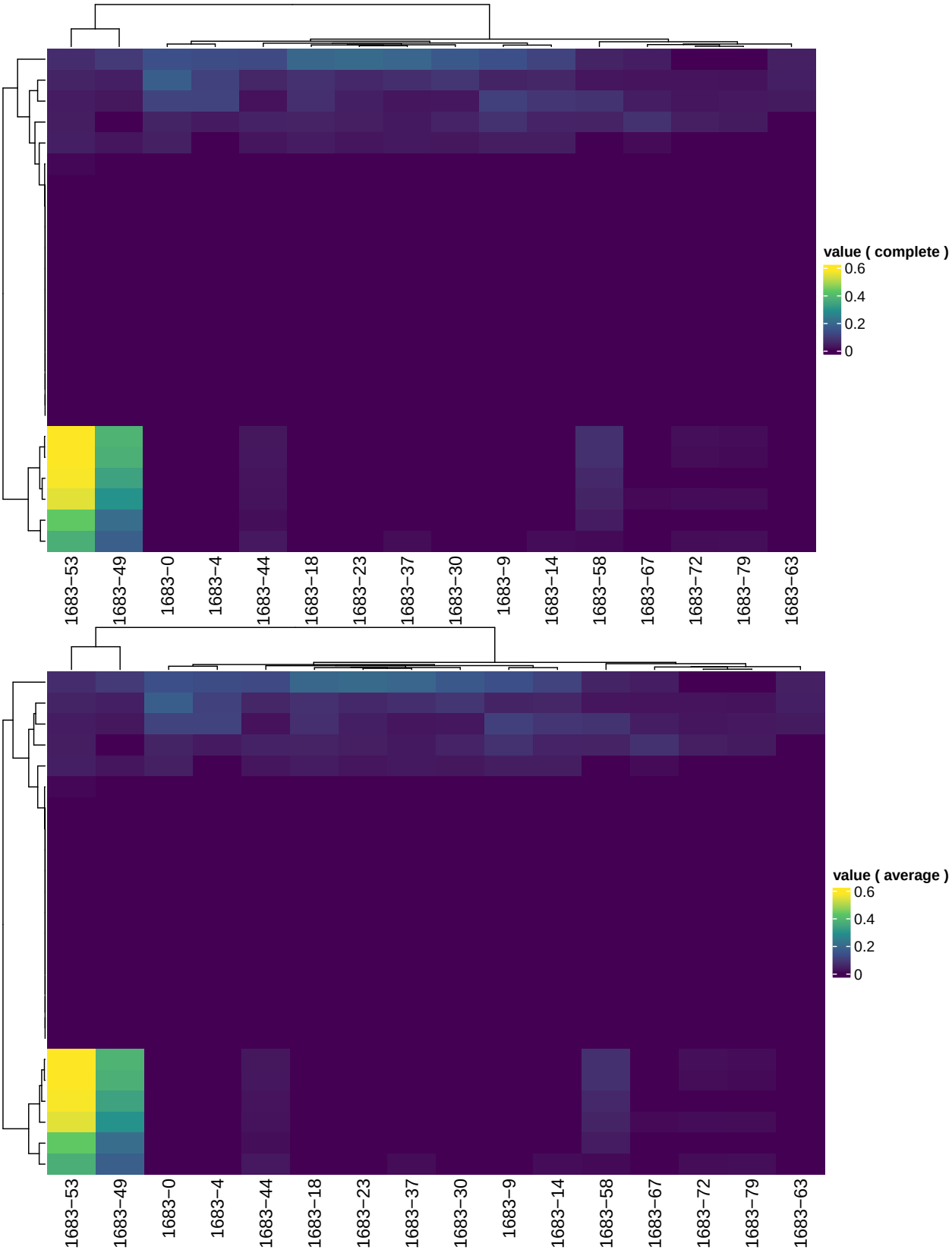
```
pdf(pdf_path, width = 10, height = 7)
draw(ht)
dev.off()
cat('Saved basic heatmap to', pdf_path, '\n')
```

8. Try a few clustering methods

Even though this notebook is meant to stay simple, it is still useful to peek at what happens when we let ComplexHeatmap cluster rows/columns. Below are three variants that only differ in the linkage method.

```
methods <- c('single', 'complete', 'average')
for (m in methods) {
  ht_clust <- Heatmap(
    heatmap_matrix,
    name = paste('value (', m, ')'),
    col = color_fun,
    cluster_rows = TRUE,
    cluster_columns = TRUE,
    clustering_method_rows = m,
    clustering_method_columns = m,
    show_row_names = FALSE,
    show_column_names = TRUE,
    na_col = na_color
  )
  grid::grid.newpage()
  draw(ht_clust, main_heatmap = 1)
}
```

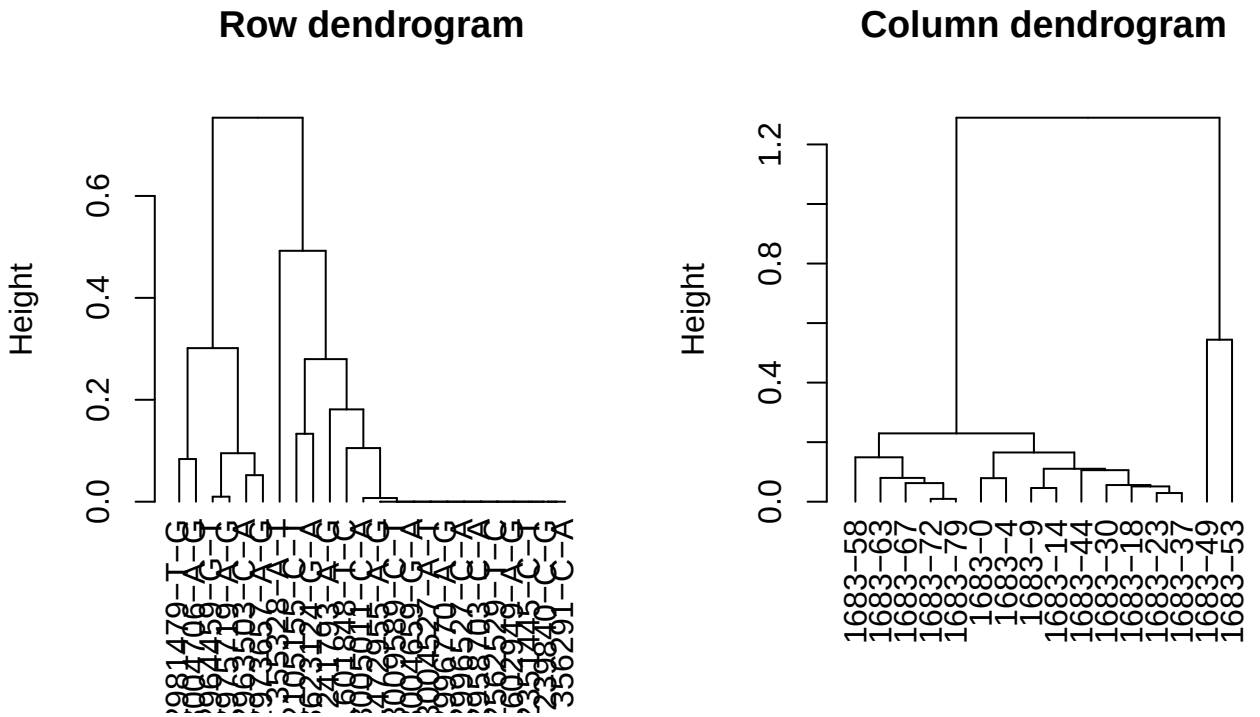



value (complete)
0.6
0.4
0.2
0
1683−53
1683−49
1683−0
1683−4
1683−44
1683−18
1683−23
1683−37
1683−30
1683−9
1683−14
1683−58
1683−67
1683−72
1683−79
1683−63
value (average)
0.6
0.4
0.2
0
1683−53
1683−49
1683−0
1683−4
1683−44
1683−18
1683−23
1683−37
1683−30
1683−9
1683−14
1683−58
1683−67
1683−72
1683−79
1683−63

9. Optional: inspect dendrograms directly

Sometimes you just want to see the clustering tree without the heatmap. You can reuse the ordered matrix to build a dendrogram using base R functions:

```
row_dend <- as.dendrogram(hclust(dist(heatmap_matrix), method = 'complete'))
col_dend <- as.dendrogram(hclust(dist(t(heatmap_matrix)), method = 'complete'))

par(mfrow = c(1, 2))
plot(row_dend, main = 'Row dendrogram', ylab = 'Height')
plot(col_dend, main = 'Column dendrogram', ylab = 'Height')
```

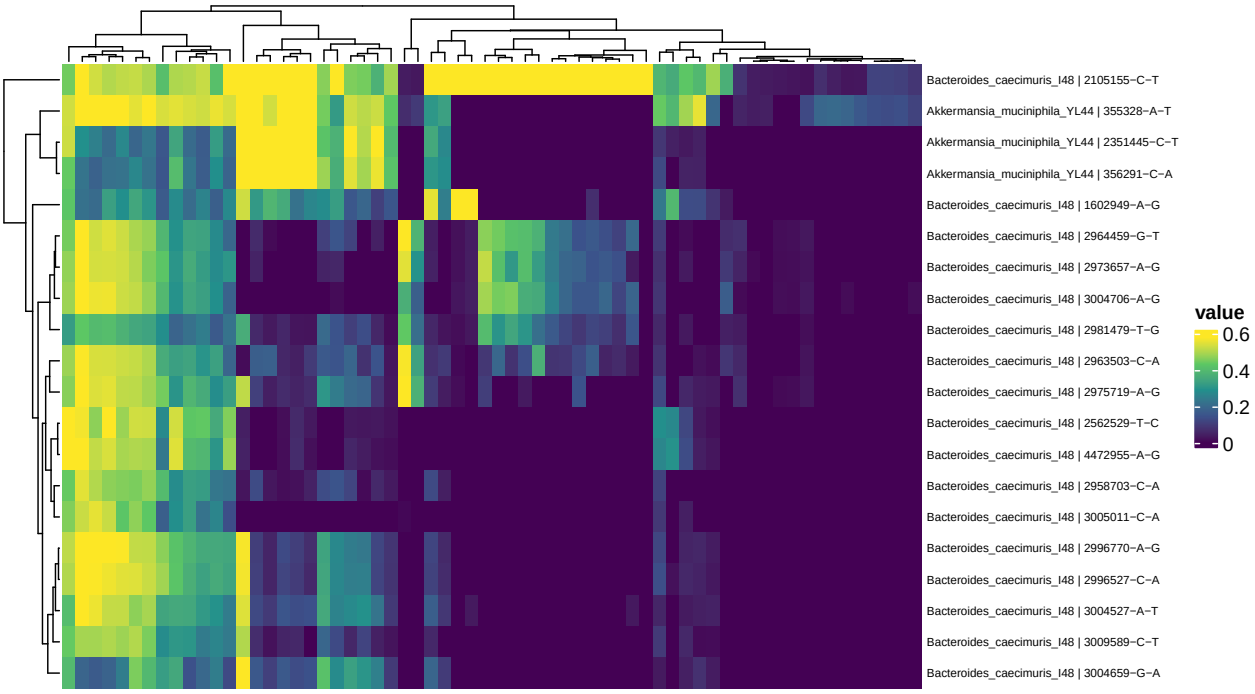



```
par(mfrow = c(1, 1))
```

10. Which SNPs vary the most?

To focus on the most dynamic rows, compute the variance of each SNP and plot a small heatmap of the top 20. This highlights where the biggest changes occur before moving to the advanced notebook.

```
row_var <- apply(mat_all, 1, var, na.rm = TRUE)
top_idx <- order(row_var, decreasing = TRUE)[seq_len(min(20, length(row_var)))]
ht_top <- Heatmap(
  mat_all[top_idx, , drop = FALSE],
  name = 'value',
  col = color_fun,
  show_row_names = TRUE,
  row_names_gp = grid::gpar(fontsize = 6),
  show_column_names = FALSE,
  na_col = na_color
)
draw(ht_top)
```



Ready for more control? Open `03_heatmap_annotations.Rmd` to add ordering, annotations, and publication polish.

03 Heatmap + Annotations

This notebook incrementally builds on the basic heatmap from `02_simple_heatmap.Rmd`. Each step introduces one new detail so students can see how changes affect the final figure.

We assume `second_day_part2/data/00_prepare_dataset.Rmd` has already produced the CSVs.

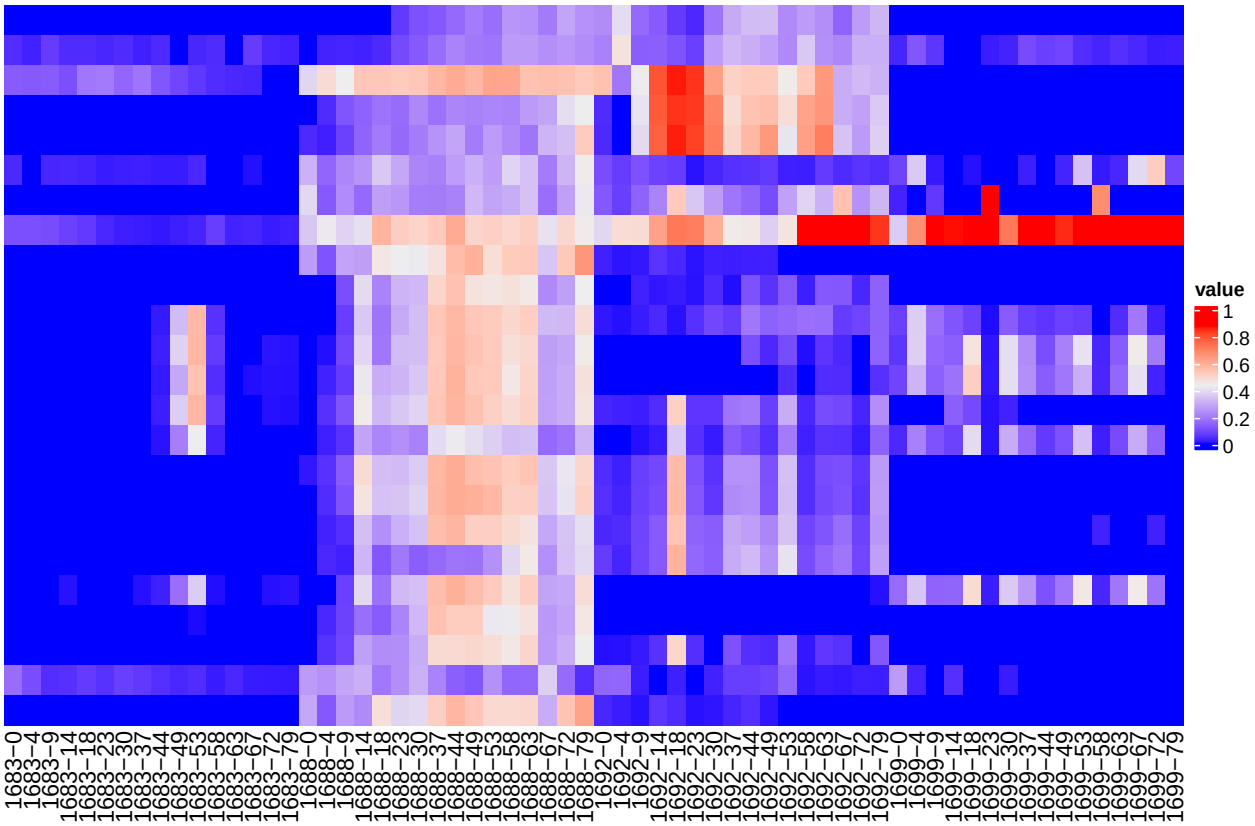
1. Load packages and data

```
library(ComplexHeatmap)
library(circlize)
subset_path <- file.path('..', 'data', 'dataset1_subset.csv')
wide_df <- read.csv(subset_path, check.names = FALSE, stringsAsFactors = FALSE)
sample_cols <- setdiff(names(wide_df), c('Genome', 'snp_id', 'Position'))
mat <- as.matrix(wide_df[, sample_cols])
mode(mat) <- 'numeric'
rownames(mat) <- paste(wide_df$Genome, wide_df$snp_id, sep = ' | ')
```

2. Step 1 – Plain heatmap (same as 02_simple_heatmap)

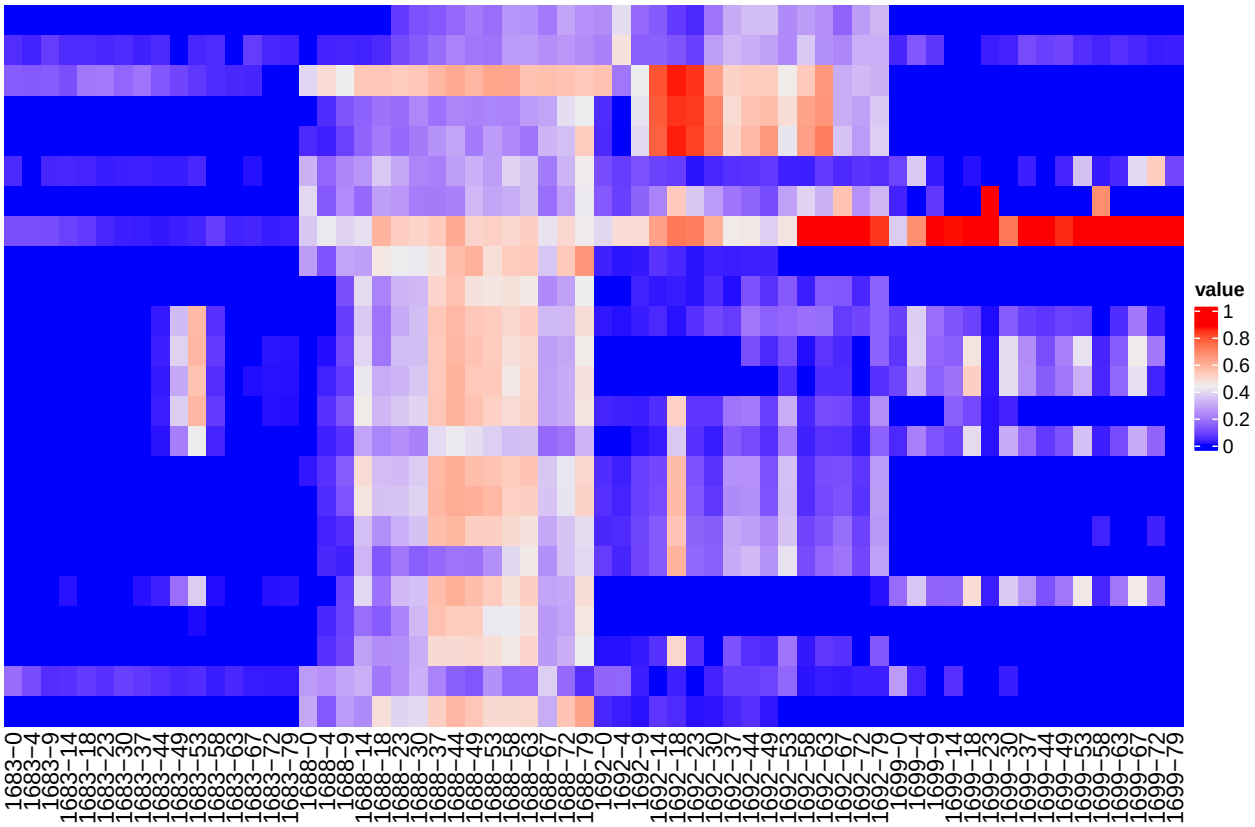
```
ht_plain <- Heatmap(
  mat,
  name = 'value',
  cluster_rows = FALSE,
  cluster_columns = FALSE,
  show_row_names = FALSE,
  show_column_names = TRUE
```

```
)
draw(ht_plain)
```



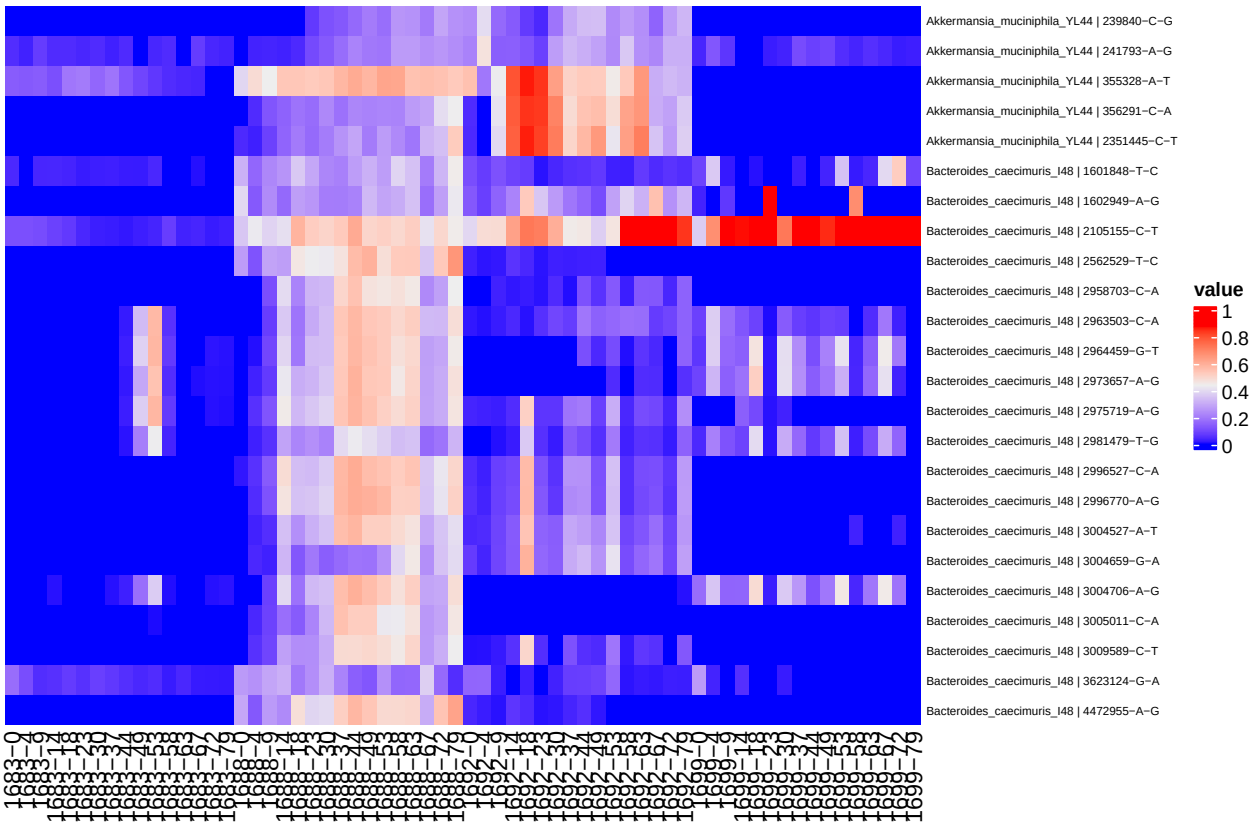
3. Step 2 – Reorder columns (mouse/day ordering)

```
sample_meta <- data.frame(sample_id = sample_cols, stringsAsFactors = FALSE)
split_ids <- strsplit(sample_meta$sample_id, '-', fixed = TRUE)
sample_meta$mouse_id <- vapply(split_ids, `[[`, character(1), 1)
sample_meta$day <- as.integer(vapply(split_ids, function(x) if (length(x) >= 2) x[[2]] else NA_characte
order_idx <- order(sample_meta$mouse_id, sample_meta$day, sample_meta$sample_id)
mat_ordered <- mat[, order_idx, drop = FALSE]
ht_ordered <- Heatmap(
  mat_ordered,
  name = 'value',
  cluster_rows = FALSE,
  cluster_columns = FALSE,
  show_row_names = FALSE,
  show_column_names = TRUE
)
draw(ht_ordered)
```



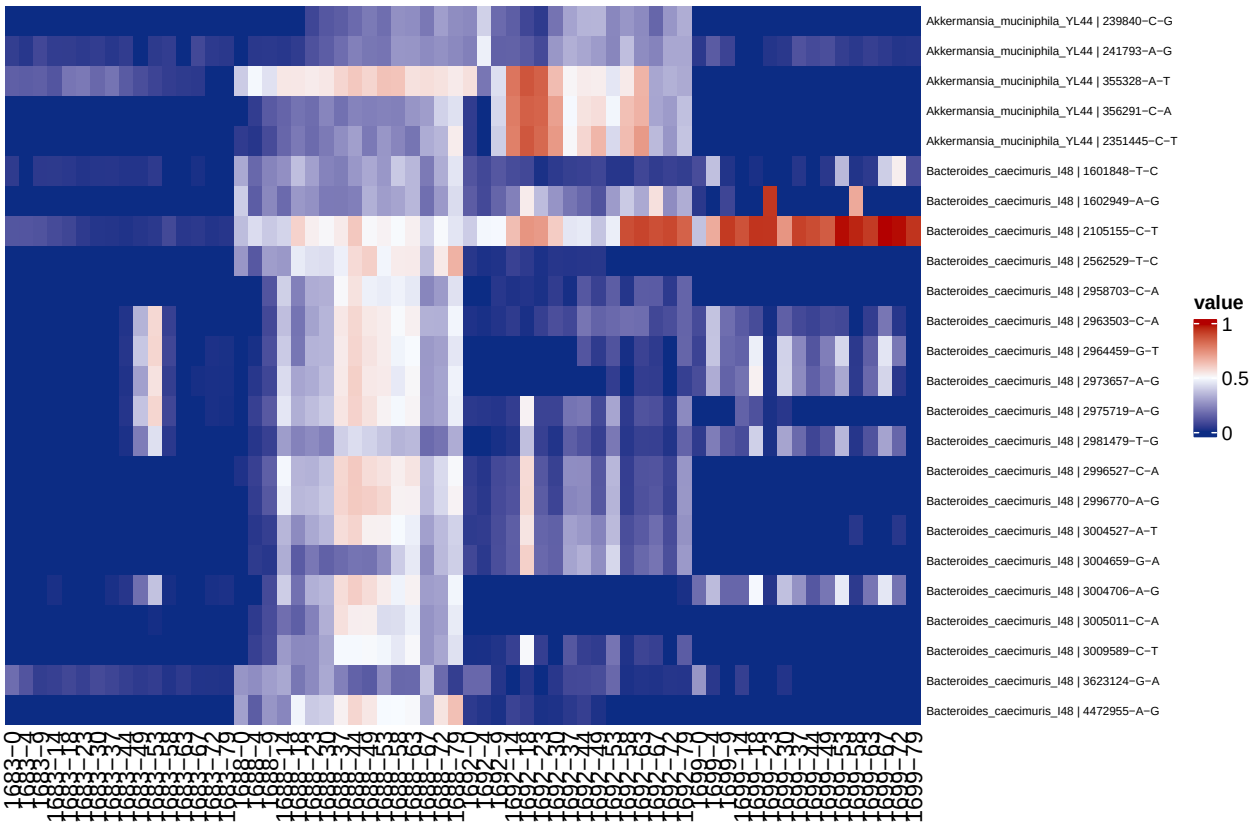
4. Step 3 – Add row labels (Genome | SNP)

```
ht_rows <- Heatmap(
  mat_ordered,
  name = 'value',
  cluster_rows = FALSE,
  cluster_columns = FALSE,
  show_row_names = TRUE,
  show_column_names = TRUE,
  row_names_gp = grid::gpar(fontsize = 6)
)
draw(ht_rows)
```



5. Step 4 – Add simple color control

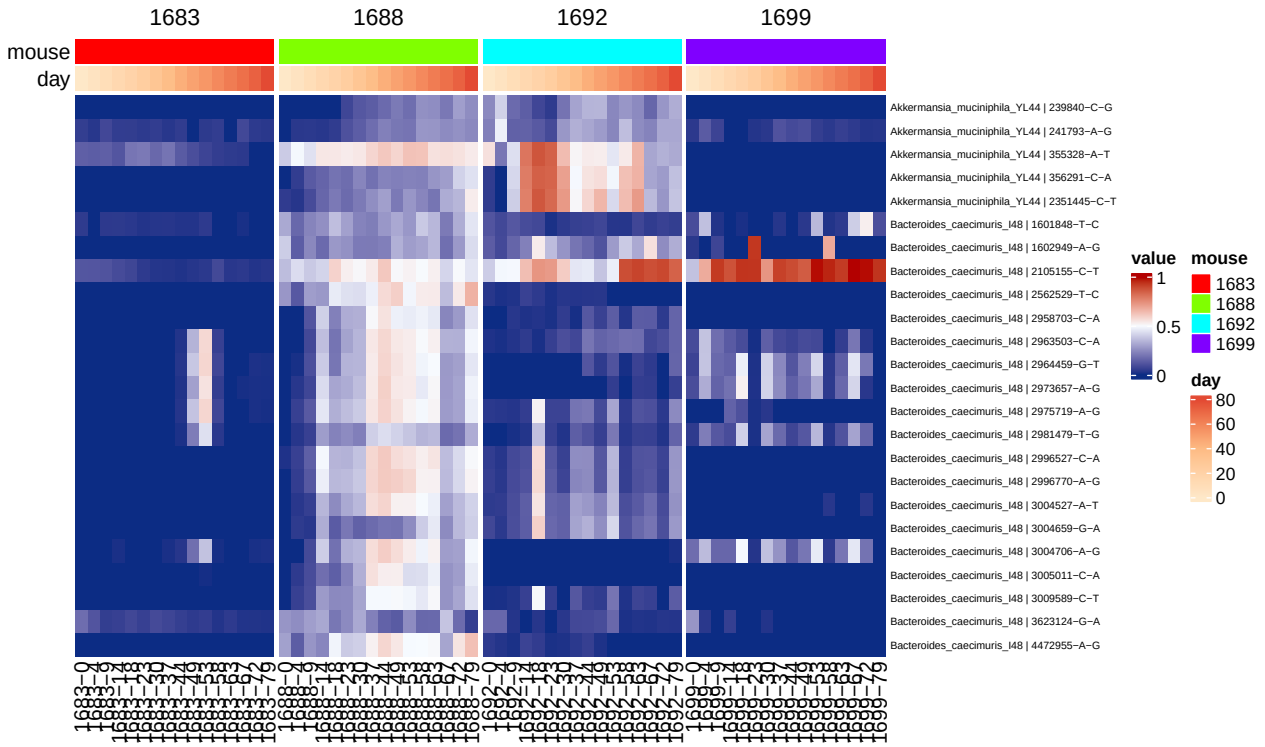
```
min_val <- min(mat_ordered, na.rm = TRUE)
max_val <- max(mat_ordered, na.rm = TRUE)
mid_val <- (min_val + max_val) / 2
col_fun <- circlize::colorRamp2(c(min_val, mid_val, max_val), c('#0c2c84', '#f7fbff', '#b30000'))
ht_colors <- Heatmap(
  mat_ordered,
  name = 'value',
  col = col_fun,
  cluster_rows = FALSE,
  cluster_columns = FALSE,
  show_row_names = TRUE,
  show_column_names = TRUE,
  row_names_gp = grid::gpar(fontsize = 6)
)
draw(ht_colors)
```



6. Step 5 – Add column annotations (mouse ID, day)

```
mouse_levels <- unique(sample_meta$mouse_id)
mouse_colors <- setNames(grDevices::rainbow(length(mouse_levels)), mouse_levels)
min_day <- min(sample_meta$day, na.rm = TRUE)
max_day <- max(sample_meta$day, na.rm = TRUE)
if (min_day == max_day) {
  day_colors <- circlize::colorRamp2(c(min_day, min_day + 1), c('#fee8c8', '#e34a33'))
} else {
  day_colors <- circlize::colorRamp2(seq(min_day, max_day, length.out = 3), c('#fee8c8', '#fdbb84', '#e
}
col_ann <- HeatmapAnnotation(
  mouse = factor(sample_meta$mouse_id, levels = mouse_levels),
  day = sample_meta$day,
  col = list(mouse = mouse_colors, day = day_colors),
  annotation_name_side = 'left'
)
ht_ann <- Heatmap(
  mat_ordered,
  name = 'value',
  col = col_fun,
  top_annotation = col_ann,
  column_split = factor(sample_meta$mouse_id, levels = mouse_levels),
  cluster_rows = FALSE,
  cluster_columns = FALSE,
  show_row_names = TRUE,
  show_column_names = TRUE,
```

```
  row_names_gp = grid::gpar(fontsize = 6)
)
draw(ht_ann, heatmap_legend_side = 'right', annotation_legend_side = 'right')
```



7. Step 6 – Export to PDF

```
pdf(file.path('..', 'pdf', '02_simple_heatmap_with_annotations.pdf'), width = 10, height = 7)
draw(ht_ann, heatmap_legend_side = 'right', annotation_legend_side = 'right')
dev.off()
```

You can extend this notebook further by experimenting with row splits, additional color control, or integrating external annotations. Compare this result with the minimalist `02_simple_heatmap.Rmd` to emphasize how each step adds clarity.

04 Full Heatmap

This notebook takes `dataset3_subset.csv` / `dataset3_subset_long.csv` (all genomes) and gradually moves from a plain heatmap to a more publication-ready version. Each step adds one feature so you can see the incremental effect.

1. Packages, paths, and helpers

```
library(ComplexHeatmap)
library(circlize)
subset_path <- file.path('..', 'data', 'dataset3_subset.csv')
long_path <- file.path('..', 'data', 'dataset3_subset_long.csv')
pdf_path <- file.path('..', 'pdf', '04_full_heatmap.pdf')
na_color <- '#dcdcdc'
```

2. Load data and basic NA report

```
wide_df <- read.csv(subset_path, check.names = FALSE, stringsAsFactors = FALSE)
long_df <- read.csv(long_path, check.names = FALSE, stringsAsFactors = FALSE)
cat('Wide rows x cols:', nrow(wide_df), ncol(wide_df), '\n')
```

```
## Wide rows x cols: 71 67
```

```
cat('Long rows x cols:', nrow(long_df), ncol(long_df), '\n')
```

```
## Long rows x cols: 4544 7
```

```
na_total <- sum(is.na(long_df$value))
cat('Total NA entries in value:', na_total, '\n')
```

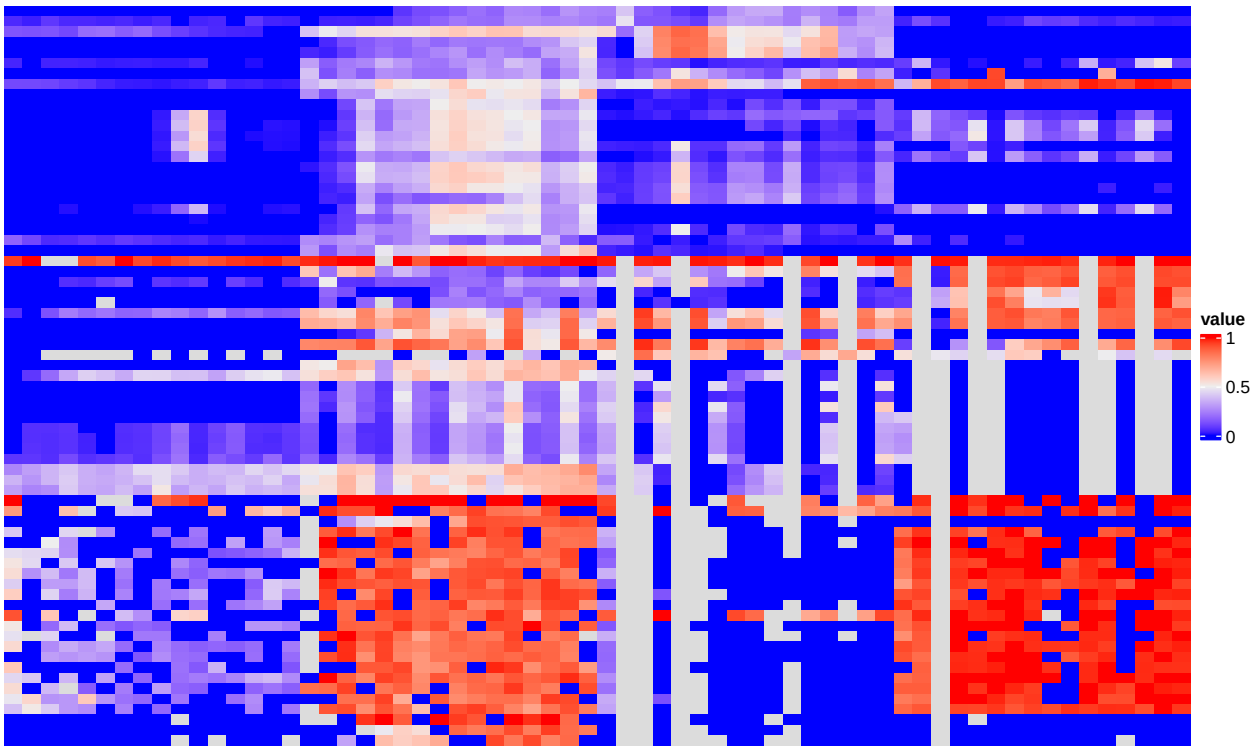
```
## Total NA entries in value: 443
```

```
if (na_total > 0) {
  na_table <- with(long_df, tapply(value, list(mouse_id, day), function(x) sum(is.na(x))))
  print(na_table)
}
```

```
##       0  4  9 14 18 23 30 37 44 49 53 58 63 67 72 79
## 1683  0  1  2  3  4  4  2  0  1  2  1  1  1  0  1  1
## 1688 15  1  1  2  3  0  1  1  0  0  0  1  0  0  1  1
## 1692  0 47 24  0 46 20  6  0  1  9 38  0  0 28  0  0
## 1699  0 23 37  0 24 13  0  0  1  1 23 13  1 23 13  1
```

3. Base heatmap (no annotations)

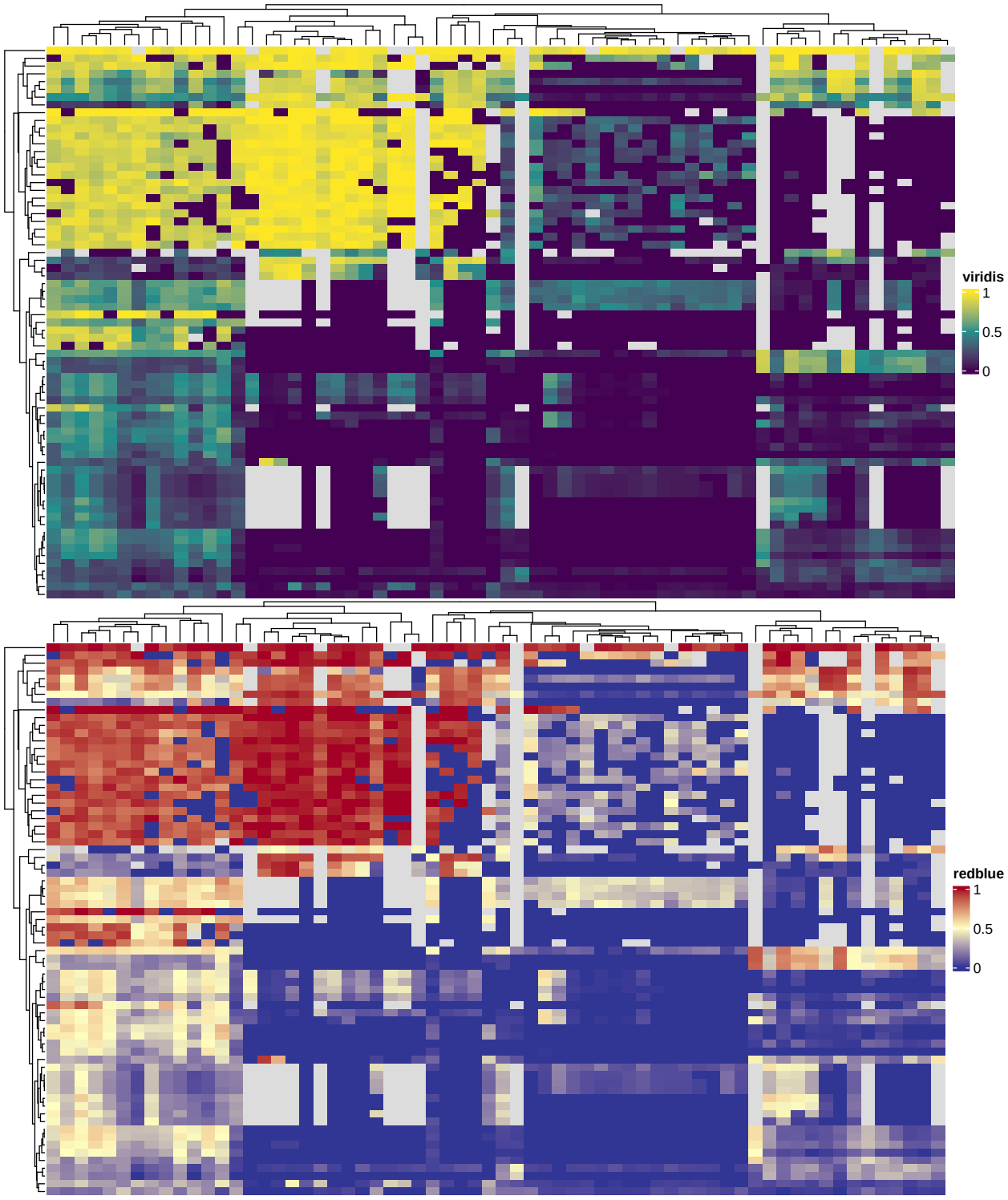
```
sample_cols <- setdiff(names(wide_df), c('Genome', 'snp_id', 'Position'))
mat <- as.matrix(wide_df[, sample_cols])
mode(mat) <- 'numeric'
rownames(mat) <- paste(wide_df$Genome, wide_df$snp_id, sep = ' | ')
colnames(mat) <- sample_cols

ht_base <- Heatmap(
  mat,
  name = 'value',
  cluster_rows = FALSE,
  cluster_columns = FALSE,
  show_row_names = FALSE,
  show_column_names = FALSE,
  na_col = na_color
)
draw(ht_base)
```



4. Explore color scales

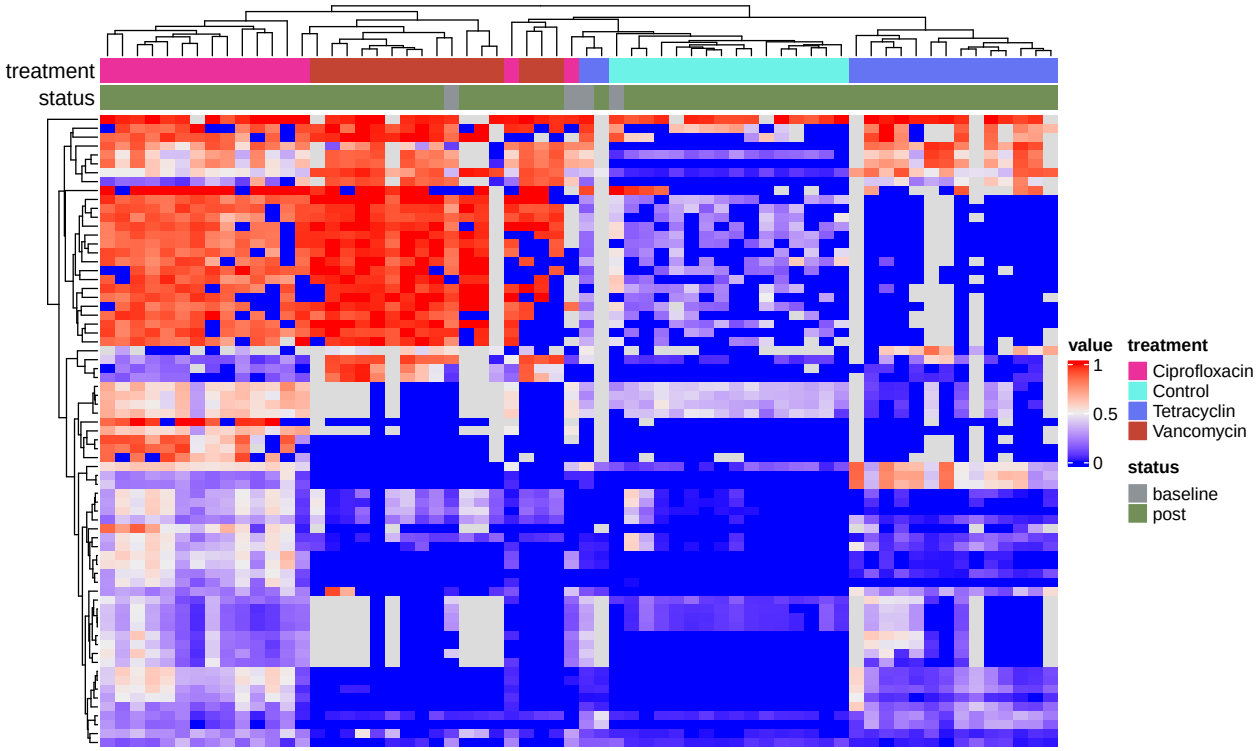
Generate three quick variants to see how different palettes affect the look.

```
mins <- min(mat, na.rm = TRUE)
maxs <- max(mat, na.rm = TRUE)
mids <- (mins + maxs) / 2
palettes <- list(
  viridis = circlize::colorRamp2(c(mins, mids, maxs), viridisLite::viridis(3)),
  redblue = circlize::colorRamp2(c(mins, mids, maxs), c('#313695', '#ffffbf', '#a50026')),
  grayscale = circlize::colorRamp2(c(mins, mids, maxs), c('#f7f7f7', '#cccccc', '#252525'))
)
for (nm in names(palettes)) {
  ht_tmp <- Heatmap(mat, name = nm, col = palettes[[nm]], show_row_names = FALSE, show_column_names = F
  grid::grid.newpage()
  draw(ht_tmp)
}
```


viridis
1
0.5
0
redblue
1
0.5
0

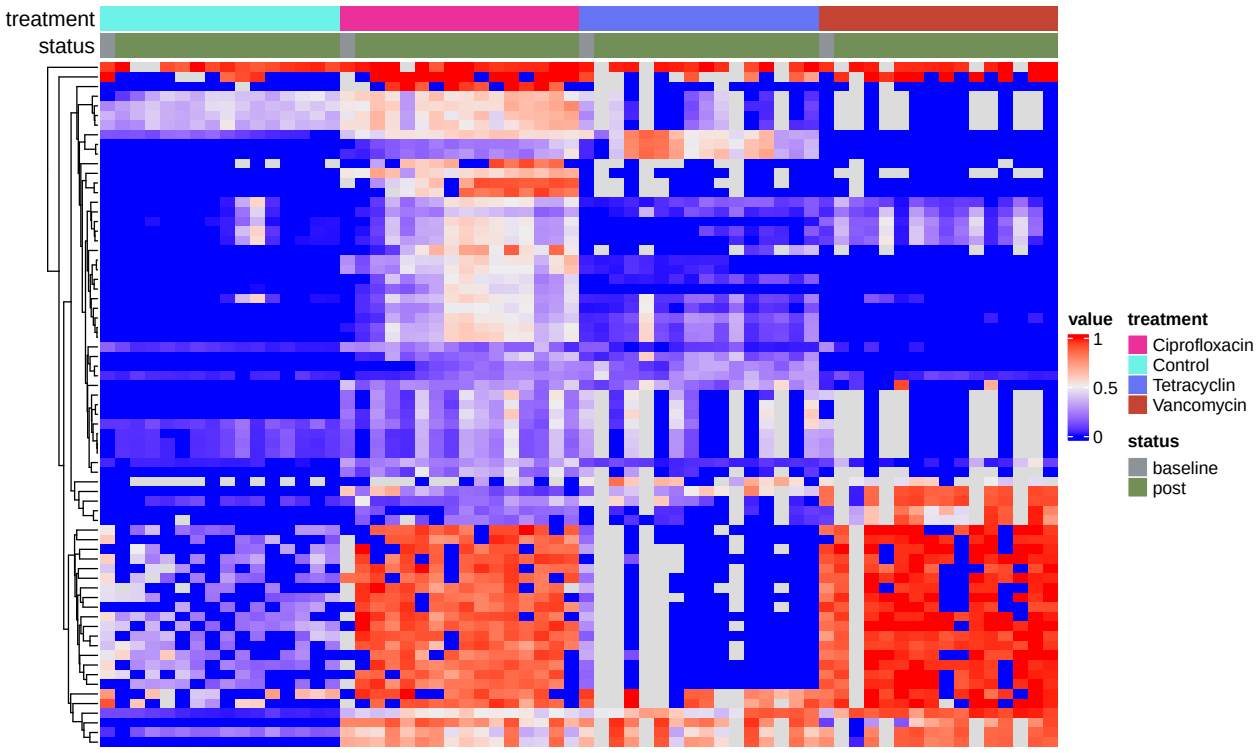
5. Add treatment and baseline/post annotations

```
sample_meta <- unique(long_df[, c('mouse_id', 'day', 'treatment_group')])
sample_meta$sample_id <- paste(sample_meta$mouse_id, sample_meta$day, sep = '-')
sample_meta <- sample_meta[match(colnames(mat), sample_meta$sample_id), ]
sample_meta$post_ab <- ifelse(sample_meta$day == 0, 'baseline', 'post')
col_ann <- HeatmapAnnotation(
  treatment = sample_meta$treatment_group,
  status = sample_meta$post_ab,
  annotation_name_side = 'left'
)

ht_ann <- Heatmap(
  mat,
  name = 'value',
  show_row_names = FALSE,
  show_column_names = FALSE,
  na_col = na_color,
  top_annotation = col_ann
)
draw(ht_ann)
```



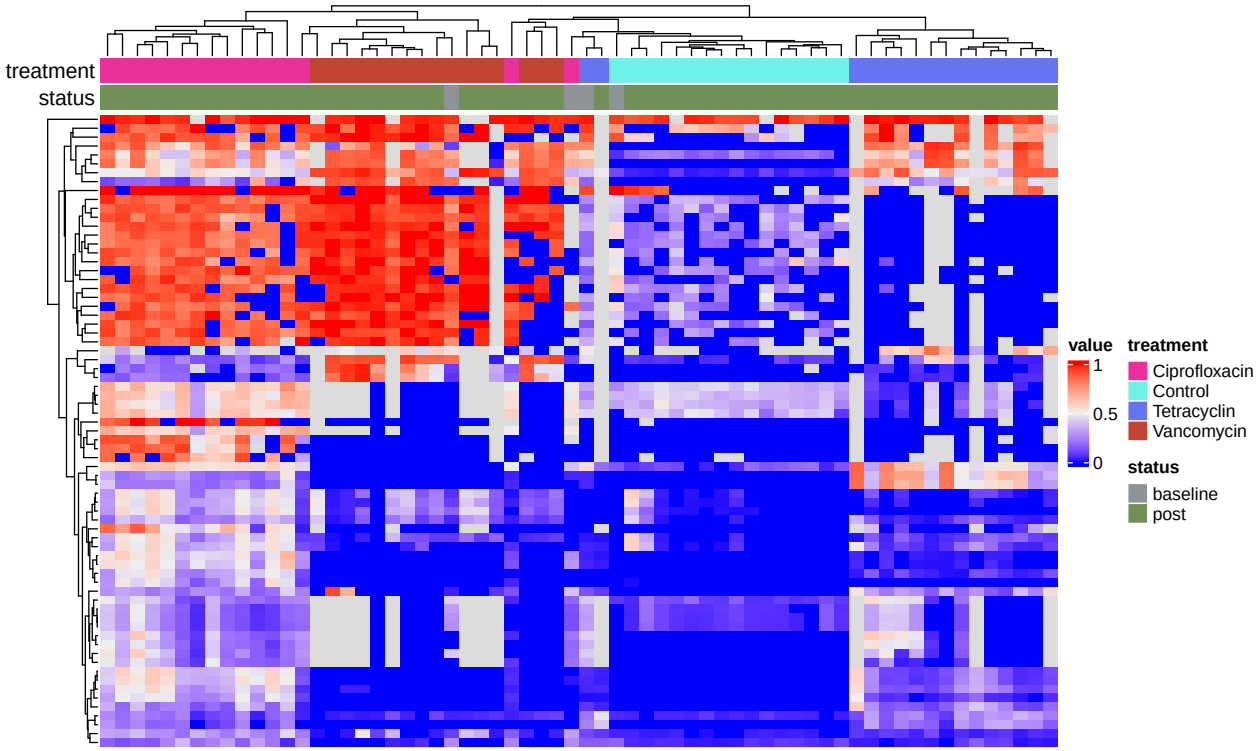
6. Order columns (mouse/day) and cluster rows

```
order_idx <- order(sample_meta$mouse_id, sample_meta$day)
mat_ordered <- mat[, order_idx]
row_dend <- hclust(dist(mat_ordered), method = 'average')
ht_ordered <- Heatmap(
  mat_ordered,
  name = 'value',
  cluster_rows = as.dendrogram(row_dend),
  cluster_columns = FALSE,
  show_row_names = FALSE,
  show_column_names = FALSE,
  na_col = na_color,
  top_annotation = col_ann[order_idx]
)
draw(ht_ordered)
```



7. Row filtering example (top variance)

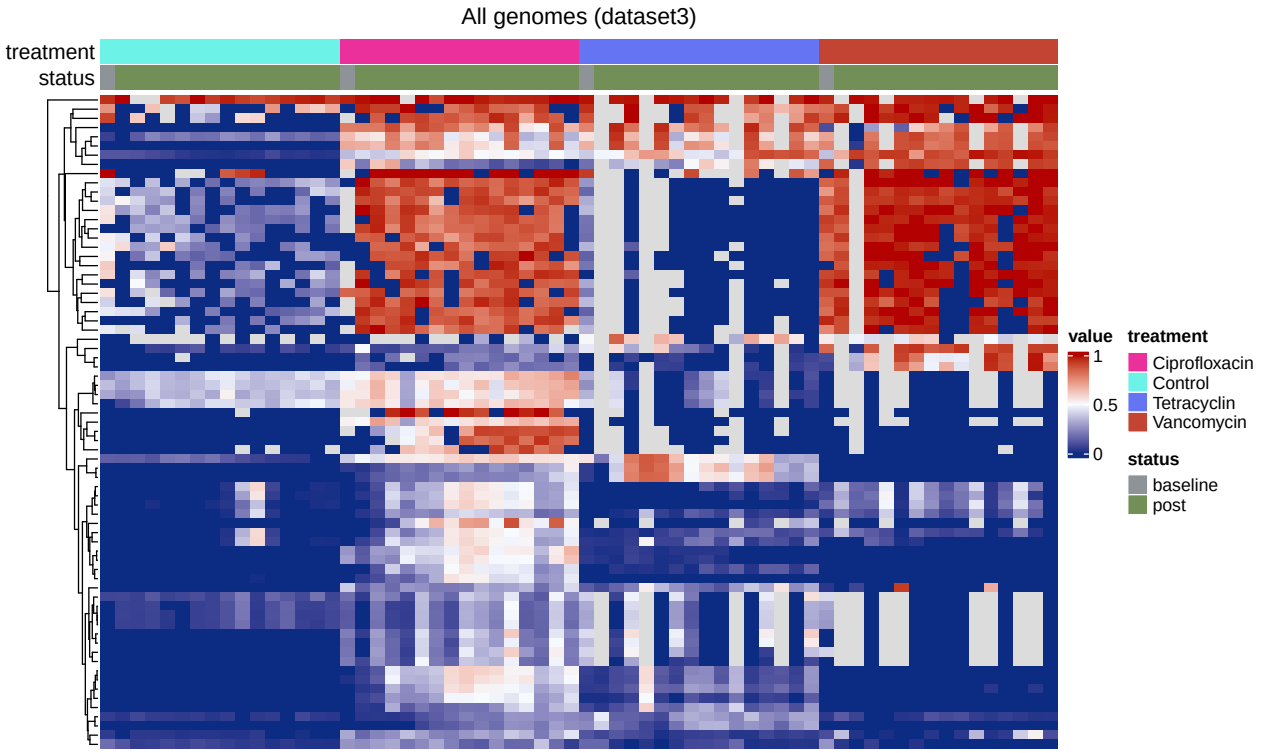
```
row_var <- apply(mat, 1, var, na.rm = TRUE)
keep_idx <- order(row_var, decreasing = TRUE)[seq_len(min(100, nrow(mat)))]
mat_var <- mat[keep_idx, ]
ht_var <- Heatmap(
  mat_var,
  name = 'value',
  show_row_names = FALSE,
  show_column_names = FALSE,
  na_col = na_color,
  top_annotation = col_ann
)
draw(ht_var)
```



8. Final polish and PDF export

Here we combine ordering, clustering, and annotations into one figure and save it.

```
ht_final <- Heatmap(
  mat_ordered,
  name = 'value',
  col = circlize::colorRamp2(c(mins, mids, maxs), c('#0c2c84', '#f7fbff', '#b30000')),
  top_annotation = col_ann[order_idx],
  cluster_rows = TRUE,
  cluster_columns = FALSE,
  show_row_names = FALSE,
  show_column_names = FALSE,
  na_col = na_color,
  column_title = 'All genomes (dataset3)'
)

draw(ht_final)
```




```
pdf(pdf_path, width = 11, height = 7)
draw(ht_final)
dev.off()
cat('Saved final heatmap to', pdf_path, '\n')
```