

Plotting Rainplots

Mir Henglin

July 09, 2019

To construct our plots in R, we will be using the `ggplot2` package. To perform data manipulation, we will be using the `dplyr` package.

```
library(dplyr)
library(ggplot2)
```

Data Format

The plots we will make will summarize the results of multiple models at the same time. In order to plot those results in `ggplot2`, they must be properly formatted. Specifically, the data must be organized in a `data.frame` with columns indicating:

- The model that the results a row correspond to
- The dependent variable term
- The P-value
- The effect size estimate

Other columns can be included in the data, but for the purpose of this tutorial we will focus on the four necessary columns.

Tutorial Dataset

The dataset we will be using contains regression coefficients and P-values for models evaluating the relationship between the measured levels of metabolites found in the blood and biological measures of interest. A small portion of that dataset is shown below.

```
plot_data

## # A tibble: 168 x 4
##   response      term      estimate p.value
##   <chr>      <chr>      <dbl>   <dbl>
## 1 Body Mass Index Metabolite 17      0.4 5.74e-52
## 2 Framingham Risk Score Metabolite 17      0.4 4.80e-46
## 3 Age          Metabolite 09      0.3 1.87e-30
## 4 Female Sex    Metabolite 07      0.6 2.47e-25
## 5 Metabolic Syndrome Metabolite 17      0.7 1.78e-23
## 6 Female Sex    Metabolite 17     -0.5 3.92e-18
## 7 Age          Metabolite 07      0.2 2.96e-17
## 8 Framingham Risk Score Metabolite 09      0.2 8.91e-17
## 9 Age          Metabolite 04      0.2 1.02e-15
## 10 Body Mass Index Metabolite 14     -0.2 1.47e-13
## # ... with 158 more rows
```

In this dataset, `estimate` and `p.value` indicate the effect size estimate and the P-value of that estimate, `term` indicates the ID of the metabolite, and `response` indicates the biological measure of interest. For example, if `response = Body Mass Index` and `term = mzid_396.271758_6.3118`, that row in the dataset corresponds to the model;

$$BodyMassIndex = \beta_0 + \beta_1 * mzid_396.271758_6.3118 + X\beta + \epsilon$$

Where $X\beta$ corresponds to control variables included in the model and ϵ is the error term. `estimate` is the estimated value of β_1 and `p.value` is the corresponding P-value.

Before plotting, we will first transform `p.value` onto the negative-log scale. This will allow the smallest P-values, which are often of greater interest, to have the largest values when plotted.

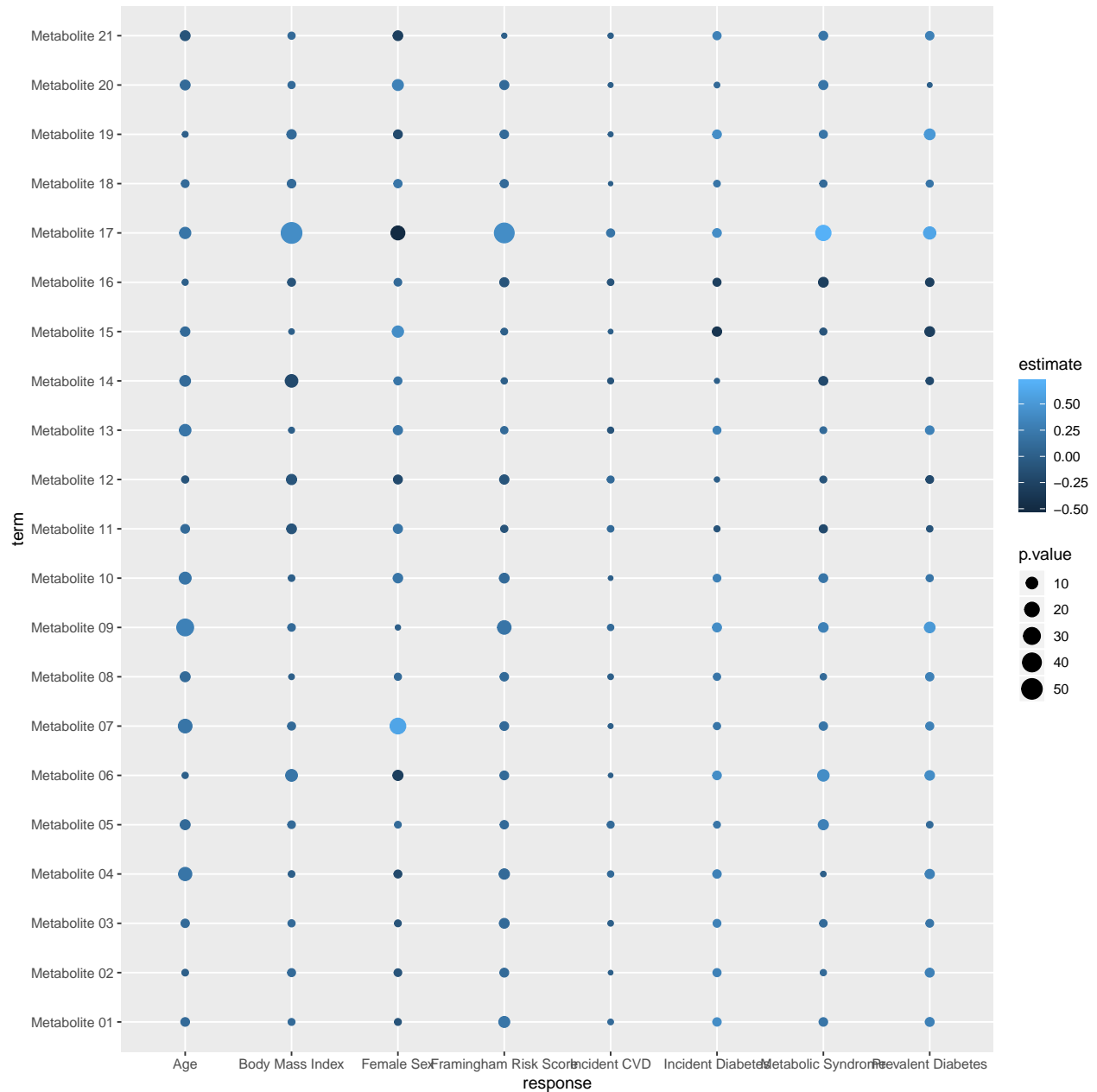
```
plot_data <-
  plot_data %>%
  mutate(p.value = -1 * log10(p.value))
```

Plotting

A basic rainplot can be constructed in only two lines of `ggplot2` code!

```
rainplot <-
  ggplot(plot_data) +
  geom_point(aes(x = response, y = term, colour = estimate, size = p.value))

rainplot
```



This is a good start, but we want to clean up layout and presentation. We can do this by creating a custom `ggplot2` theme and adjusting scales and layout. One thing to ensure is to represent P-values (the size of the plotted points) by area instead of radius. When comparing two points of different size, humans perceive the area of points, not their radius, when comparing them. Thus a value that is twice another should have twice as much area, not double the radius. This is ensured by using `scale_size_area`.

```
thm <-
  # Good starting theme + set text size
  theme_light(base_size = 18) +
  theme(
    # Remove axis ticks and titles
    axis.title.x = element_blank(),
    axis.ticks.x = element_blank(),
```

```

axis.title.y = element_blank(),
axis.ticks.y = element_blank(),

# Remove gridlines and boxes
panel.grid.major = element_blank(),
panel.grid.minor = element_blank(),
axis.line = element_blank(),
legend.key = element_blank(),

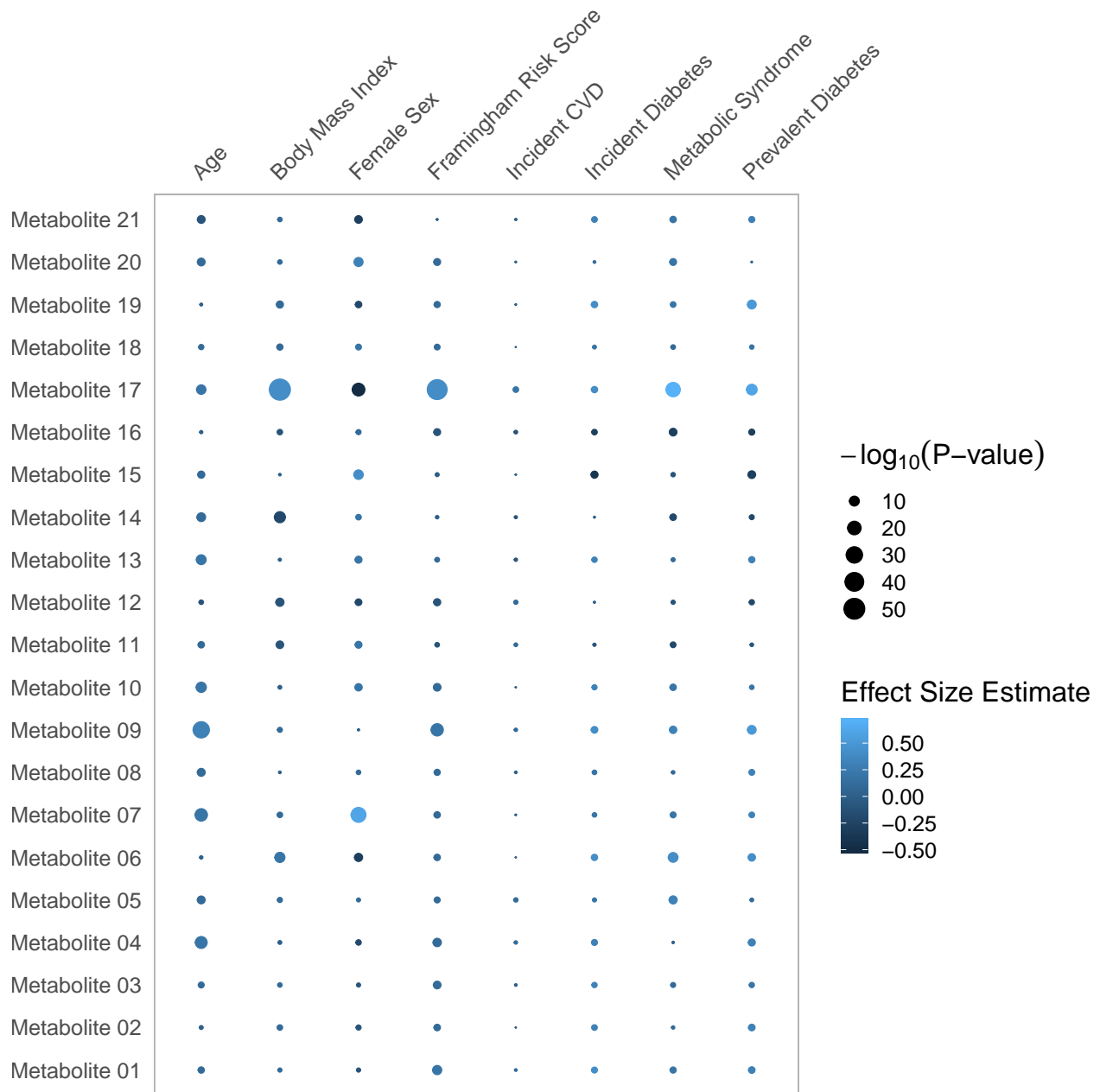
# White backgrounds
panel.background = element_rect(fill = 'white'),
plot.background = element_rect(fill = 'white'),
legend.background = element_rect(fill = 'white'),

# Angle text
axis.text.x.top = element_text(angle = 45, hjust = 0)
)

rainplot <-
  rainplot +
  thm +
  # Move x-axis label to top of plot
  scale_x_discrete(position = 'top') +
  scale_size_area(expression(paste(-log[10]('P-value')))) +
  scale_color_continuous('Effect Size Estimate')

rainplot

```



To make the presentation of the effect size estimates clearer, we use a diverging red-blue palette. In order to guarantee a symmetric palette, we set the palette limits based on our data.

```
palette <-
  # Blue
  c("#053061",
    "#313695",
    "#4575b4",
    "#74add1",
    "#abd9e9",
    "#e0f3f8",
    "#fee090",
    "#fdae61",
    "#f46d43",
```

```

"#d73027",
"#a50026",
'#67001f')
# Red

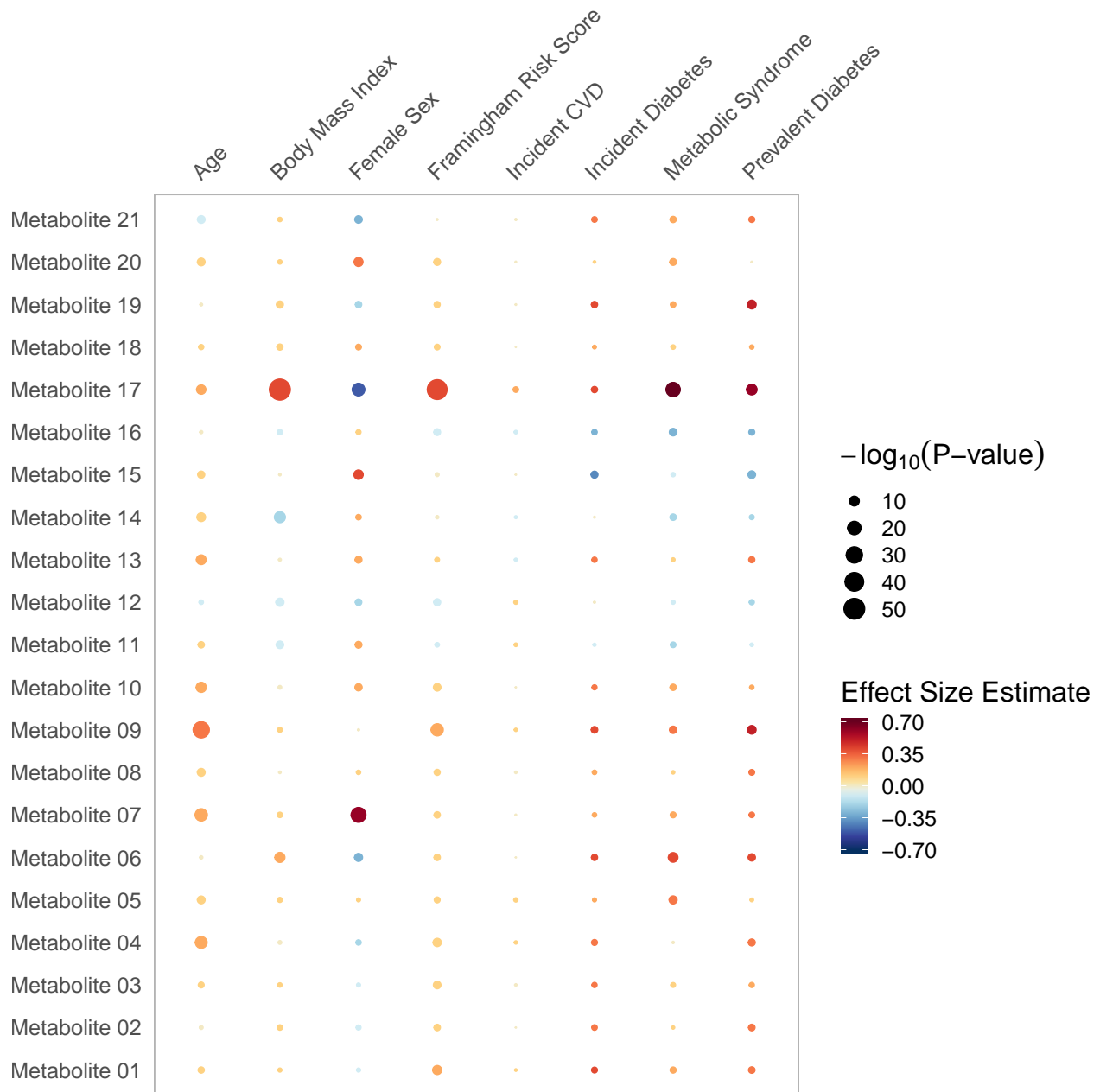
# Calculate symmetric limits based on most extreme value
max_abs_estimate <- max(abs(plot_data$estimate))

max_lim <- max_abs_estimate
min_lim = -1 * max_lim

rainplot <- rainplot +
  scale_color_gradientn(
    'Effect Size Estimate',
    colors = palette,
    limits = c(min_lim, max_lim),
    breaks = c(min_lim, min_lim / 2, 0, max_lim/2, max_lim)
  )

## Scale for 'colour' is already present. Adding another scale for
## 'colour', which will replace the existing scale.
rainplot

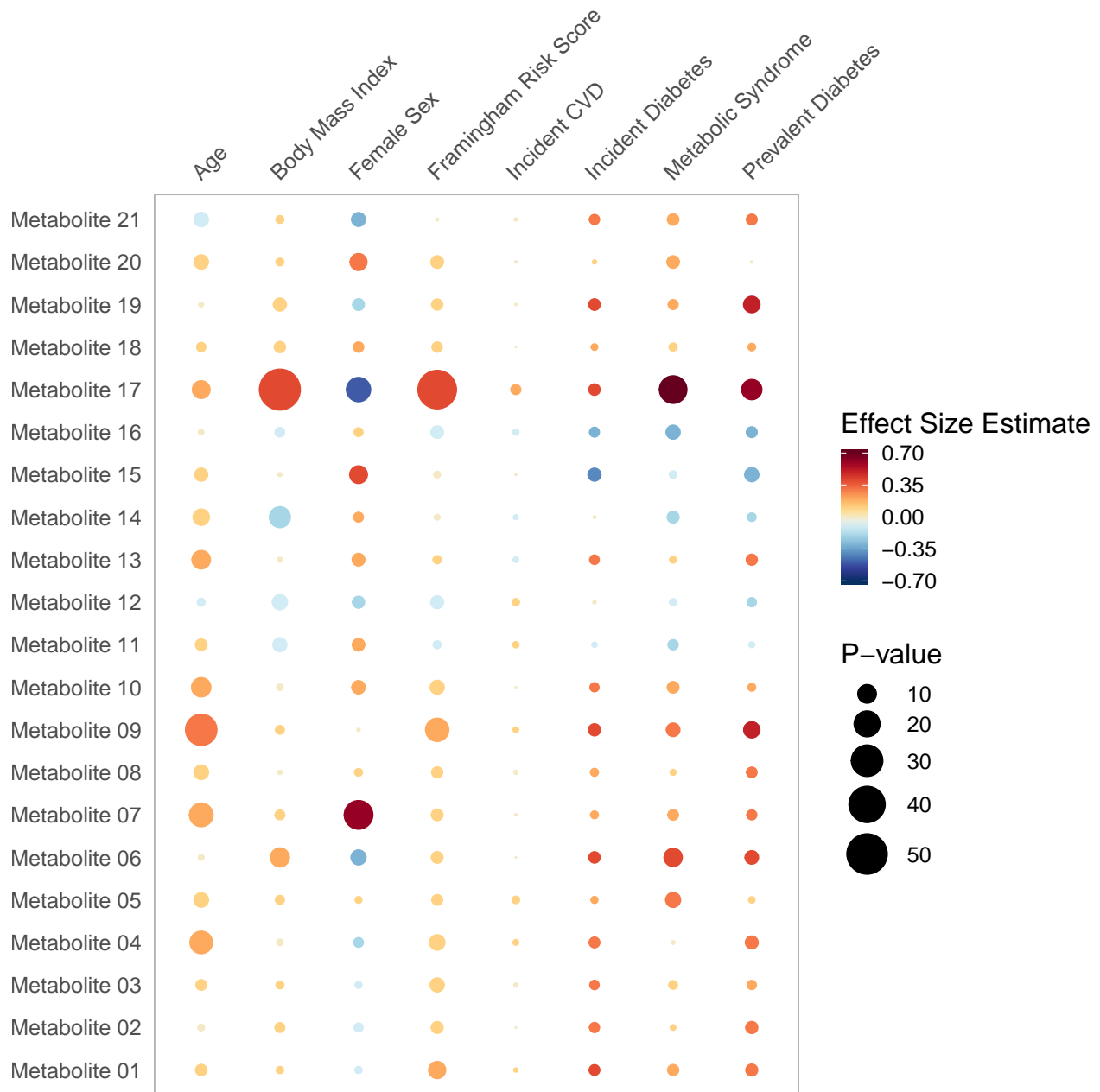
```



Another step to improve presentation is to increase the maximum size of each point. There will be a bit of trial and error here; if the size threshold is too large, the points will overlap.

```
rainplot +
  scale_size_area('P-value', max_size = 12)
```

```
## Scale for 'size' is already present. Adding another scale for 'size',
## which will replace the existing scale.
```



Additional Plot Adjustments

Outlines

The points on rainplots can be outlined in a color different from the color of the point. To get such a shape, we add the argument `shape = 21` to `geom_point`. Note that when we do this, the color of the point changes from `color` to `fill`. Here, we leave the outlines as the default black.

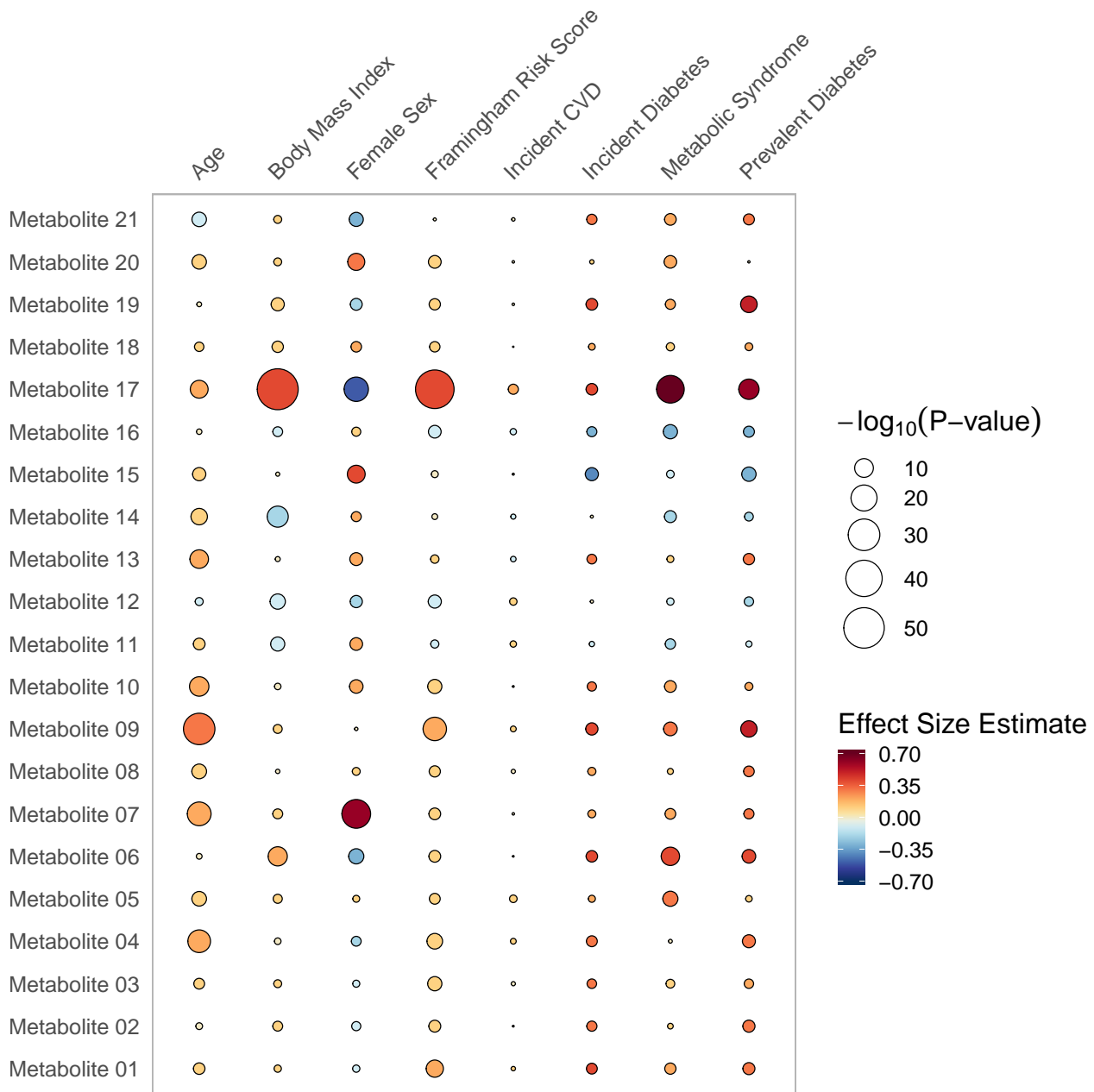
```
rainplot <-
  ggplot(plot_data) +
  geom_point(aes(x = response, y = term, fill = estimate, size = p.value),
             shape = 21) +
  scale_fill_gradientn(
```



```

    'Effect Size Estimate',
    colors = palette,
    limits = c(min_lim, max_lim),
    breaks = c(min_lim, min_lim / 2, 0 , max_lim / 2, max_lim)
  ) +
  scale_size_area(expression(paste(-log[10]('P-value'))), max_size = 12) +
  scale_x_discrete(position = 'top') +
  thm
rainplot

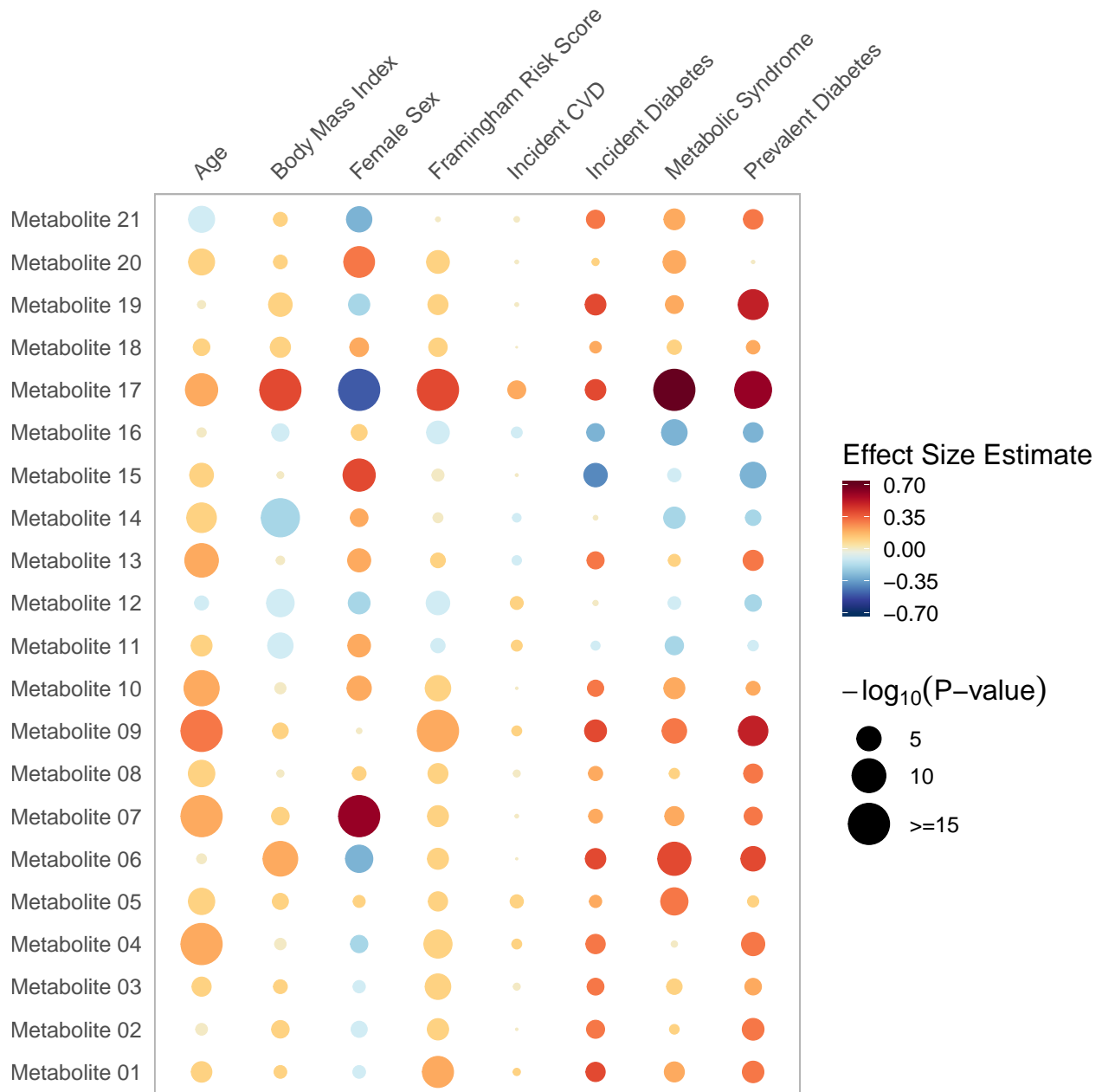
```



P-value Thresholding

When a few P-values are much smaller than the majority of the data, a rainplot loses size resolution in the range where most of the data lies. One possible solution is to set all P-values above some ceiling, here chosen to be 15, to the value of the ceiling. The threshold can be set at a level where one considers all P-values more extreme than the threshold to be ‘of interest’.

```
plot_data_thresholded <-  
  plot_data %>%  
  mutate(p.value = ifelse(p.value > 15, 15, p.value))  
  
rainplot <-  
  # Use the thresholded data  
  ggplot(plot_data_thresholded) +  
  geom_point(aes(x = response, y = term, colour = estimate, size = p.value)) +  
  scale_x_discrete(position = 'top') +  
  scale_color_gradientn(  
    'Effect Size Estimate',  
    colors = palette,  
    limits = c(min_lim, max_lim),  
    breaks = c(min_lim, min_lim / 2, 0, max_lim / 2, max_lim)  
  ) +  
  # Set the legend breaks and labels to account for the thresholding  
  scale_size_area(  
    expression(paste(-log[10]('P-value'))),  
    max_size = 12,  
    breaks = c(5, 10, 15),  
    labels = c('5', '10', '>=15')) +  
  thm  
  
rainplot
```



Ordering by P-Value

To make it easier to identify the metabolites that had small P-values in multiple models, we will convert the `term` column into an 'ordered factor' ordered by the average P-value across all models. This will put metabolites with small P-values in multiples models at the top of the plot, and metabolites with large P-values in multiple models at the bottom of the plot.

```
# Calculate mean P-value for each metabolite
mpv <-
  plot_data %>%
  group_by(term) %>%
  summarise(mean_pv = mean(p.value))
```

```

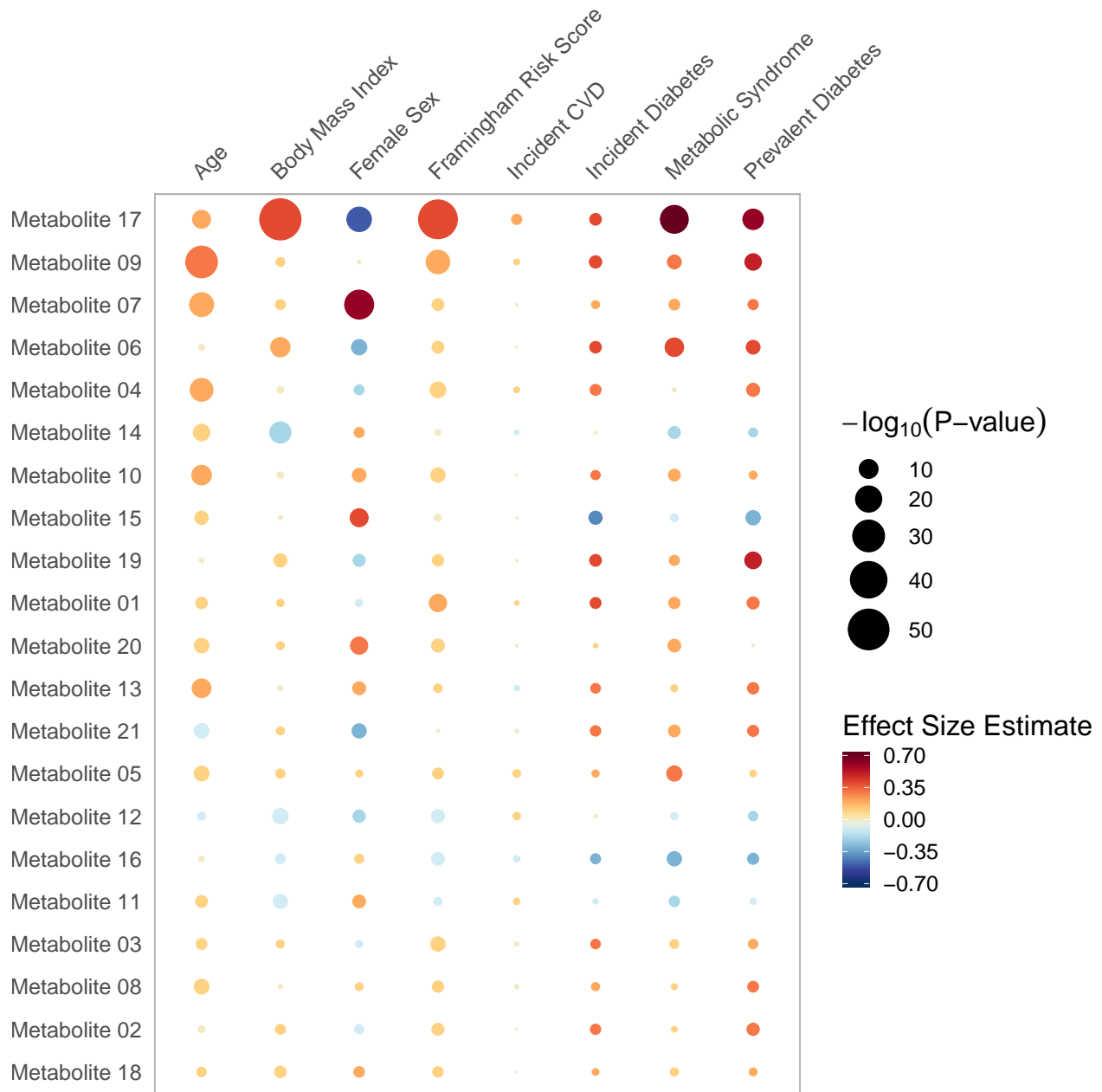
# Order metabolites by average p-value
term_order <-
  mpv %>%
  arrange(mean_pv) %>%
  pull(term)

# Convert term to a factor, ordered by `term_order`
plot_data_pvo <-
  plot_data %>%
  mutate(term = factor(term, levels = term_order))

rainplot <-
  # Use the data with the term column ordered by mean P-value
  ggplot(plot_data_pvo) +
  geom_point(aes(x = response, y = term, colour = estimate, size = p.value)) +
  scale_color_gradientn(
    'Effect Size Estimate',
    colors = palette,
    limits = c(min_lim, max_lim),
    breaks = c(min_lim, min_lim / 2, 0 , max_lim / 2, max_lim)
  ) +
  scale_size_area(expression(paste(-log[10]('P-value'))), max_size = 12) +
  scale_x_discrete(position = 'top') +
  thm

rainplot

```



Ordering by Cluster

rainplots can be ordered by cluster such that similar dependent variable terms are plotted next to one another. We will be using the `hculst` function to cluster the results by effect size estimate. The `term` variable will be converted into an ordered factor, such that clustered terms are plotted next to one another. In order to cluster the data, we will need to reshape it using the `spread` function from the `tidyr` package.

```
library(tidyr)

# Convert to matrix and reshape for clustering.
cluster_data <-
  plot_data %>%
```

```

select(response, term, estimate) %>%
spread(response, estimate)

rnms <-
  cluster_data$term

cluster_data <-
  cluster_data %>%
  select(-term) %>%
  as.matrix()

rownames(cluster_data) <- rnms

# Quick peek to see structure of data
cluster_data[1:5, 1:5]

##           Age Body Mass Index Female Sex Framingham Risk Score
## Metabolite 01 0.1           0.1      -0.1           0.2
## Metabolite 02 0.0           0.1      -0.1           0.1
## Metabolite 03 0.1           0.1      -0.1           0.1
## Metabolite 04 0.2           0.0      -0.2           0.1
## Metabolite 05 0.1           0.1       0.1           0.1
##           Incident CVD
## Metabolite 01           0.1
## Metabolite 02           0.0
## Metabolite 03           0.0
## Metabolite 04           0.1
## Metabolite 05           0.1

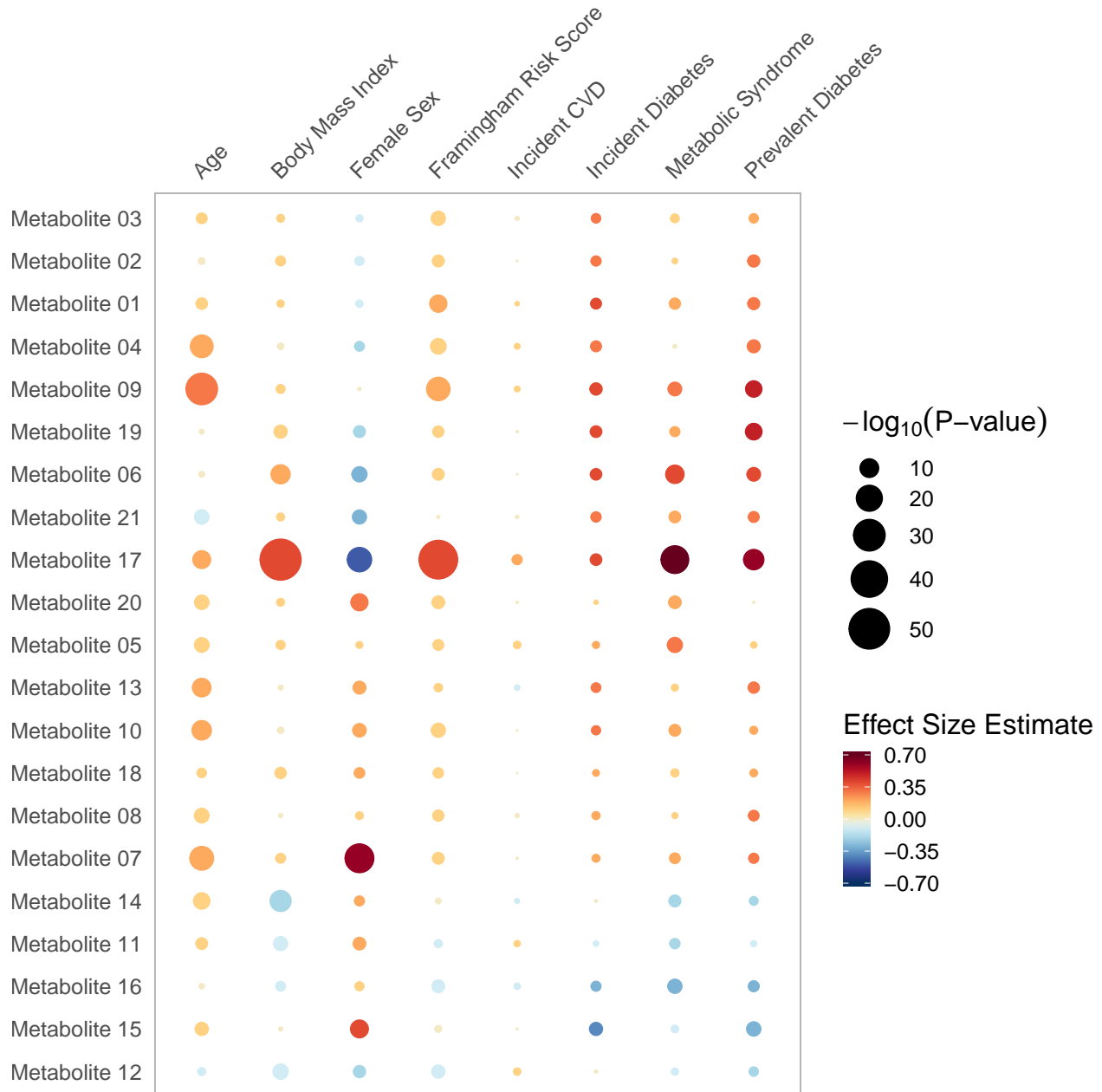
# cluster dependent variable terms
clust <- hclust(dist(cluster_data), method = 'ward.D2')

# `clust$order` orders `term` into clusters
term_order <-
  clust$labels[clust$order]

# Convert term to a factor, ordered by `term_order`
plot_data_clo <-
  plot_data %>%
  mutate(term = factor(term, levels = term_order))

rainplot <-
  # Use cluster ordered data
  ggplot(plot_data_clo) +
  geom_point(aes(x = response, y = term, colour = estimate, size = p.value)) +
  scale_color_gradientn(
    'Effect Size Estimate',
    colors = palette,
    limits = c(min_lim, max_lim),
    breaks = c(min_lim, min_lim / 2, 0, max_lim / 2, max_lim)
  ) +
  scale_size_area(expression(paste(-log[10]('P-value'))), max_size = 12) +
  scale_x_discrete(position = 'top') +
  thm

```



Side-by-side plots

Often, one will want to plot multiple graphs together. We will use the **gridExtra** package to accomplish this. By plotting multiple graphs together, we can easily scale up the information presented in our plots.

A common problem when composing multiple graphs in the same plot is alignment. Graphs will often share common features that need to be aligned, and differences in scale and layout can break that alignment when the graphs are plotted together. Thus, an important consideration when creating our graphs plots will be to match the scale and layout of every graph we want to align. A detailed walkthrough of this process can be found in `aligning_plots.rmd`

```
library(gridExtra)
```

Adding dendrograms

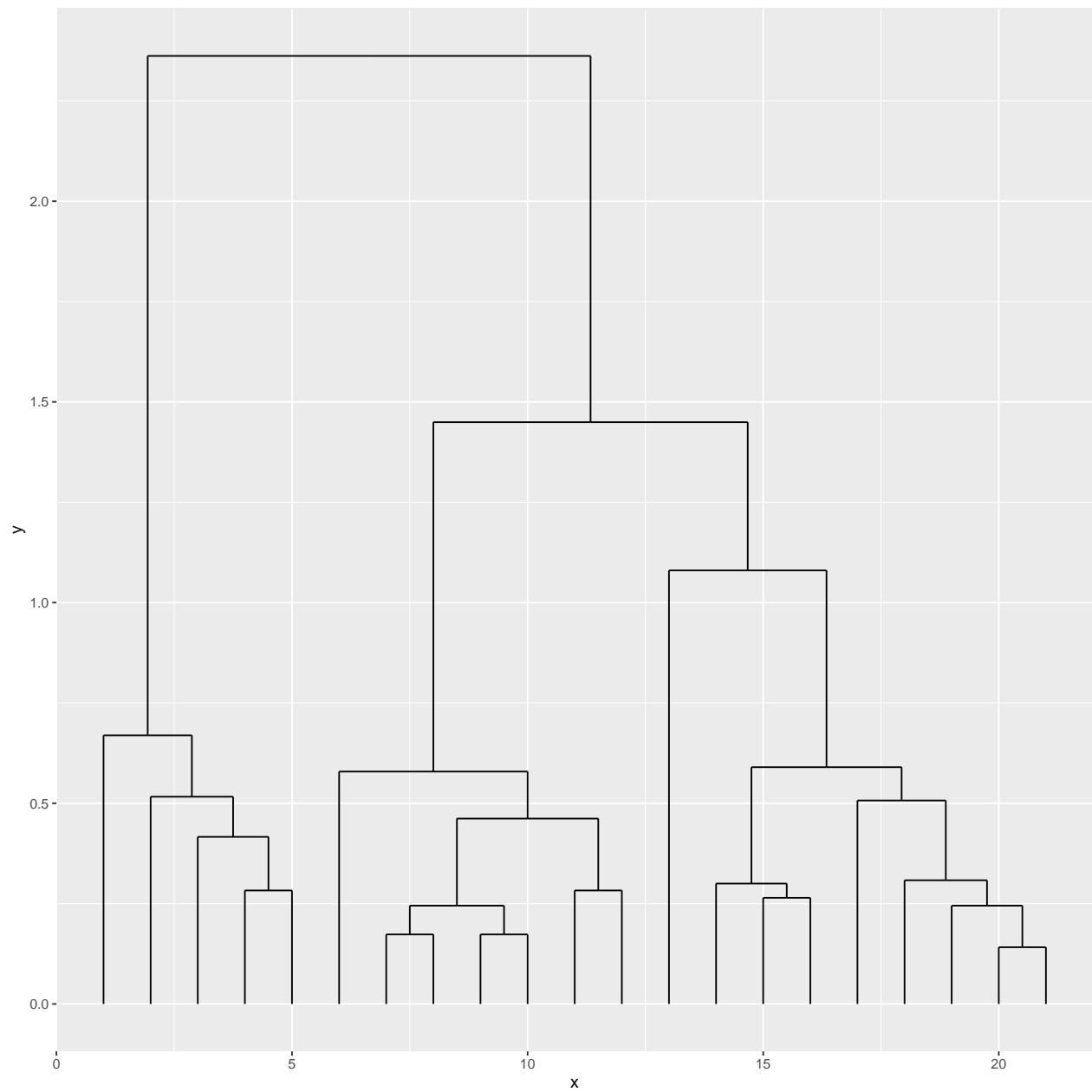
Dendrograms can be added to cluster-ordered `ggplot2` plots using the `ggdendro` package.

```
library(ggdendro)
```

```
# Extract dendrogram data from previous cluster results  
dendro_dat <- segment(dendro_data(clust))
```

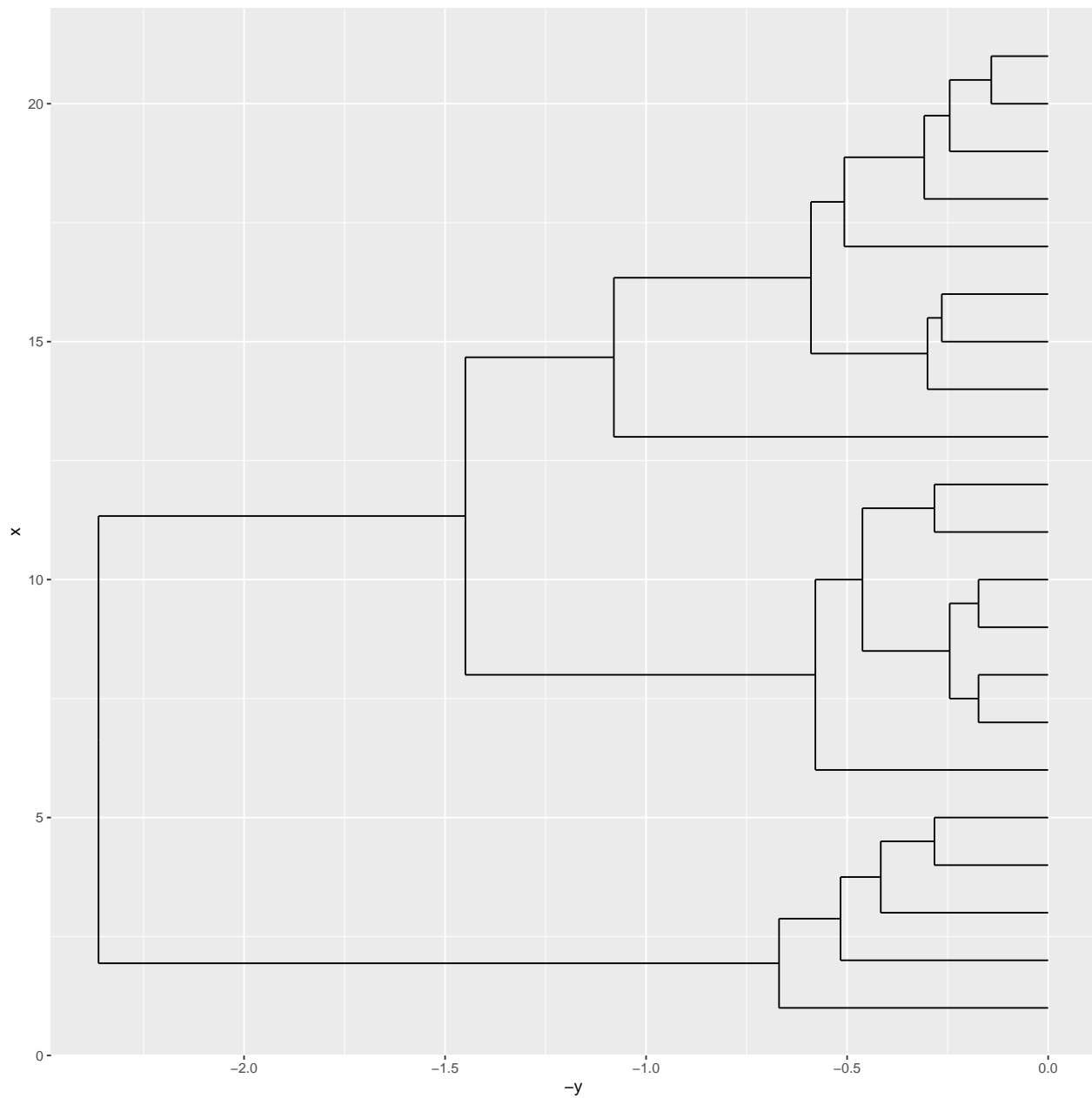
A basic dendrogram can be plotted quite easily.

```
dendro <-  
  ggplot(dendro_dat) +  
    geom_segment(aes(x = x, y = y, xend=xend, yend=yend), colour = 'black')  
  
dendro
```

The default dendrogram points down. To put the dendrogram on the left of our plot, we want it to point to the right. We can do this by switching the x and y coordinates.

```
dendro <-  
  ggplot(dendro_dat) +  
    geom_segment(aes(x = -y, y = x, xend=-yend, yend=xend), colour = 'black')  
dendro
```



Before plotting our dendrogram and rainplot side-by-side, we need to first create a version of our dendrogram with the same scale and layout as our rainplot

```
# Taking only the column with the longest label maintains alignment while
# simplifying plot layout
x_labels <-
  plot_data_clo$response %>%
  unique()

longest_x_label <-
  x_labels[[which.max(nchar(x_labels))]]

longest_x_label_data <-
  plot_data %>%
```

```

filter(response == longest_x_label)

# Align the top of the dendrogram with the x-axis label
max_dendro <-
  max(abs(c(dendro_dat$y, dendro_dat$yend)))

offset <- max_dendro + 1

dendro <-
  # Empty ggplot with same layout and scale as rainplot
  ggplot() +
  geom_blank(aes(x = response, y = term), data = longest_x_label_data) +
  thm +
  # 'expand' controls whitespace around the dendrogram. The non-zero argument
  # may need to be increased if the line thickness of the dendrogram is
  # increased to make sure the entire dendrogram is plotted
  scale_x_discrete(position = 'top', expand = c(0, 0.02, 0, 0)) +
  # Draw dendrogram
  geom_segment(aes(x = (-y + offset), y = x, xend = (-yend + offset), yend=xend),
    colour = 'black',
    data = dendro_dat) +
  # Make invisible unnecessary plot layout.
  theme(legend.position = 'none',
    axis.text.y = element_blank(),
    axis.text.x = element_text(colour = 'white'),
    panel.grid = element_blank(),
    panel.border = element_rect(fill = NA, colour = 'white'))

)

grid.arrange(dendro, rainplot, ncol = 2, widths = c(3, 15))

```

