Data Visualization ENV+WRE Seminar

Jonathan Boualavong

2/9/2022

About

This code is the example code used for the presentation on Data Visualization by Jonathan Boualavong and Sarah Torhan for the Environmental and Water resources engineering graduate seminar held on 9 Feb 2022. The code is intended to be used as a template for other data visualization projects to help streamline figure making and ensure that the figures are consistent.

This notebook has examples for making: a scatter plot with a fit line, a multi-bar plot, and a heat map/contour plot. These three figures also demonstrate labeling a figure with a trend line and text, separating plots into subplots using facets, and combining separate figures, respectively.

For each example, data is imported from a .csv file included with this project, and figures are exported into common formats (.jpg, .svg, and .pdf). The data used in these examples are artifically generated datasets if not properly cited.

Packages used

The packages used in this example code are: * ggplot2 : general plotting framework * dplyr : dataframe manipulation * lemon : additional options when create subplots of similar types * patchwork : allows combination of different figures * svglite : exporting as .svg (vector graphics) files

Both ggplot2 and dplyr are also under the package 'tidyverse', a commonly used collection of other packages. For this example, the packages will be loaded separately for simplicity, but installing and loading tidyverse will also be sufficient.

```
rm(list = ls())
if (!require('ggplot2')) install.packages('ggplot2'); library(ggplot2)
if (!require('dplyr')) install.packages('dplyr'); library(dplyr)

if (!require('patchwork')) install.packages('patchwork'); library(patchwork)
if (!require('lemon')) install.packages('lemon'); library(lemon)

if (!require('svglite')) install.packages('svglite'); library(svglite)
```

Plot formatting functions

By setting up universal formatting options as functions, you can call them later for every figure to ensure all figures are consistent. Similarly, you can set up things such as color schemes, marker sizes, or line weights ahead of time.

```
plot.paper = function (){ # Based on bbc_style(), then modified for style
    font = "sans" # Helvetica
    sz.big = 13
```

```
    sz.small = 12
    ggplot2::theme(
      # Plot settings
      plot.title    = ggplot2::element_text(family = font, size = sz.big, face = "bold",
                                            color = "black"),
      plot.subtitle = ggplot2::element_text(family = font, size = sz.big, margin =
                                              ggplot2::margin(9, 0, 9, 0)),
      plot.caption  = ggplot2::element_blank(),
      # Legend settings
      legend.position    = "right", legend.text.align = 0,
      legend.background  = ggplot2::element_blank(),
      legend.title       = ggplot2::element_text(family = font, size = sz.big,
                                                 color = "black"),
      legend.key         = ggplot2::element_blank(),
      legend.text        = ggplot2::element_text(family = font, size = sz.small,
                                                 color = "black"),
      legend.key.height  = unit(0.5, "line"),
      legend.key.width   = unit(0.75, "line"),
      legend.margin      = margin(t = -10, unit = 'pt'),
      # Axis settings
      axis.title  = ggplot2::element_text(family = font, size = sz.big, color = "black"),
      axis.text   = ggplot2::element_text(family = font, size = sz.small, color = "black"),
      axis.text.x = ggplot2::element_text(margin = ggplot2::margin(5, b = 10)),
      axis.line   = ggplot2::element_line(color = "black"),
      # Panel (background) settings
      panel.grid.minor   = ggplot2::element_blank(),
      panel.grid.major.y = ggplot2::element_blank(),
      panel.grid.major.x = ggplot2::element_blank(),
      panel.background   = ggplot2::element_blank(),
      # Strip (facet or subplot) settings
      strip.background = ggplot2::element_rect(fill = 'white', linetype = 1),
      strip.text       = ggplot2::element_text(size = sz.small, hjust = 0.5),
      strip.placement  = 'outside',
      # Aspect ratio: for papers, typically 3x4
      aspect.ratio     = 3/4
      )
}

plot.presentation = function (){ # Based on bbc_style(), then modified for style
    font = "sans" # Helvetica
    sz.big = 18
    sz.small = 16

    ggplot2::theme(
      # Plot settings
      plot.title    = ggplot2::element_text(family = font, size = sz.big, face = "bold",
                                            color = "black"),
      plot.subtitle = ggplot2::element_text(family = font, size = sz.big, margin =
                                              ggplot2::margin(9, 0, 9, 0)),
      plot.caption  = ggplot2::element_blank(),
      # Legend settings
      legend.position    = "right", legend.text.align = 0,
      legend.background  = ggplot2::element_blank(),
```

```
      legend.title       = ggplot2::element_text(family = font, size = sz.big,
                                                 color = "black"),
      legend.key         = ggplot2::element_blank(),
      legend.text        = ggplot2::element_text(family = font, size = sz.small,
                                                 color = "black"),
      legend.key.height  = unit(0.5, "line"),
      legend.key.width   = unit(0.75, "line"),
      legend.margin      = margin(t = -10, unit = 'pt'),
      # Axis settings
      axis.title  = ggplot2::element_text(family = font, size = sz.big, color = "black"),
      axis.text   = ggplot2::element_text(family = font, size = sz.small, color = "black"),
      axis.text.x = ggplot2::element_text(margin = ggplot2::margin(5, b = 10)),
      axis.line   = ggplot2::element_line(color = "black"),
      # Panel (background) settings
      panel.grid.minor   = ggplot2::element_blank(),
      panel.grid.major.y = ggplot2::element_blank(),
      panel.grid.major.x = ggplot2::element_blank(),
      panel.background   = ggplot2::element_blank(),
      # Strip (facet or subplot) settings
      strip.background = ggplot2::element_rect(fill = 'white', linetype = 1),
      strip.text       = ggplot2::element_text(size = sz.small, hjust = 0.5),
      strip.placement  = 'outside',
      # Aspect ratio: for papers, typically prefer 4/9
      aspect.ratio     = 4/9
      )
}

# Set figure widths and heights for paper exports
fig.pap.width = 4*1.4
fig.pap.height = 3*1.4

# Set figure widths and heights for presentation exports
fig.pre.width = 8*1.4
fig.pre.height = 4*1.4
```

Scatter plot with fit line

Importing labeled scatterplot data

The raw data for this section is from: Sigvart Evjen, Anne Fiksdahl, Diego D.D. Pinto, Hanna K. Knuutila. "New polyalkylated imidazoles tailored for carbon dioxide capture." International Journal of Greenhouse Gas Control, vol 76, 2018, pp. 167-174. DOI: 10.1016/j.ijggc.2018.06.017

With additional post-processing as detailed in: Jonathan Boualavong, Christopher A. Gorski. "Electrochemically Mediated CO2 Capture Using Aqueous Cu(II)/Cu(I) Imidazole Complexes." ACS ES&T Engineering, vol 1, no 7, 2021, pp. 1084-1093. DOI: 10.1021/acsestengg.1c00068

The data relates 2 chemical properties, the equilibrium constant for protonation and that of carbamate formation, of the class of compounds called imidazoles.

This data is split into 2 files: one with the raw measurement data, and one with the fit line since finding the fit line is outside of the scope of this notebook. In this case, the best-fit line is:

$$y = exp(1.29x - 9.4)$$

with a $R^2 = 0.80$

Where y is the equilibrium constant of carbamate formation (K_c) and x is the proton affinity (pK_a).

The raw data ('EvjenData-Processed.csv') includes additional columns for labeling the plot as an example of showing text on the figure.

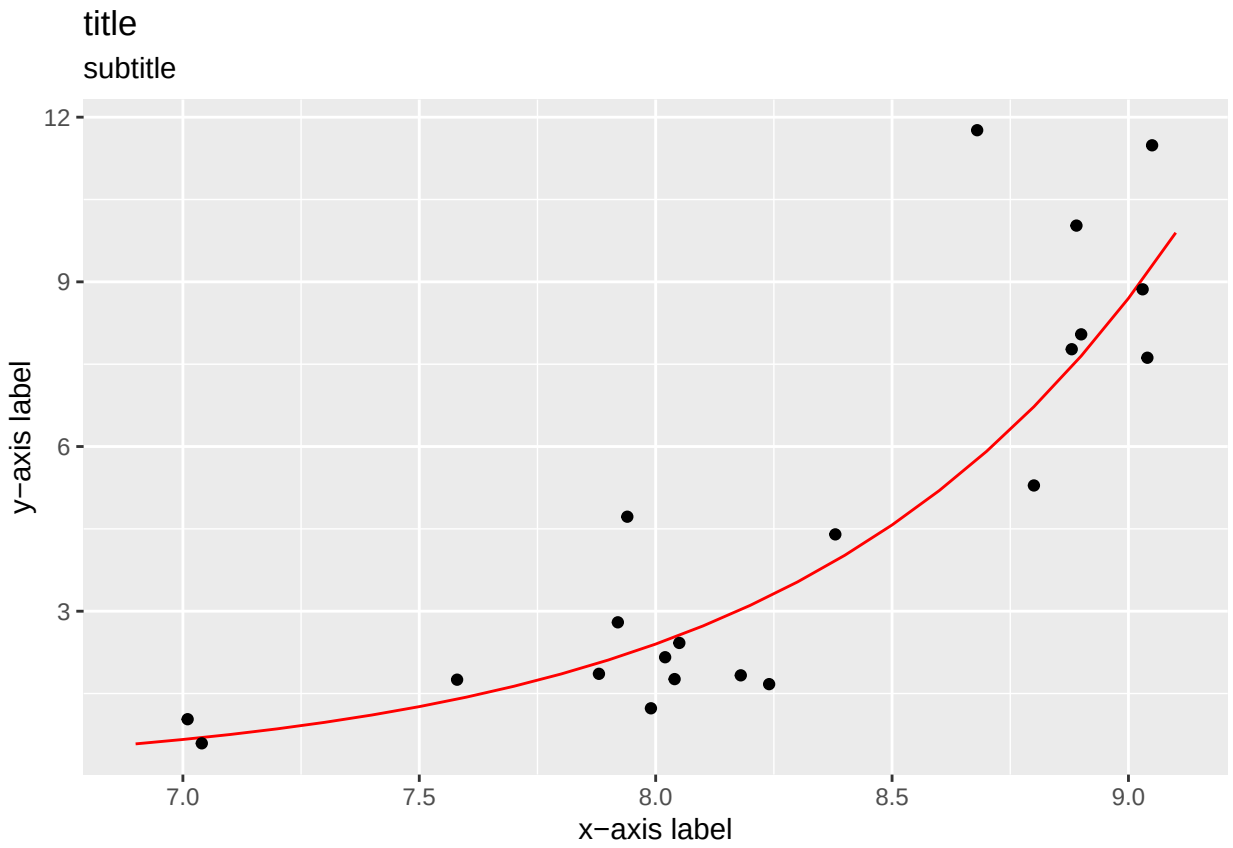
```
# Data import
points = read.csv('EvjenData-Processed.csv')
fit.plot = read.csv('EvjenDataFit.csv')

points
```

```
##                         Substitution  pka    Keq.C25  adj.x adj.y hjust
## 1                       Unsubstituted 7.01  1.0325203  0.025   0.7   0.1
## 2                           1-methyl 7.04  0.5936076     NA    NA    NA
## 3                           2-methyl 7.92  2.7975740     NA    NA    NA
## 4                            2-ethyl 7.88  1.8594898     NA    NA    NA
## 5                           4-methyl 7.58  1.7525595     NA    NA    NA
## 6                         1,2-dimethyl 8.05  2.4217760  0.025   0.1   0.0
## 7                    2-ethyl-4-methyl 8.38  4.3994825     NA    NA    NA
## 8                     2,4,5-trimethyl 8.90  8.0437408 -0.045   0.2   1.0
## 9               4-ethyl-2,5-dimethyl 8.88  7.7726541     NA    NA    NA
## 10                   1,4,5-trimethyl 8.02  2.1617161     NA    NA    NA
## 11               1-ethyl-4,5,-methyl 8.24  1.6713260     NA    NA    NA
## 12               4,5-methyl-1-propyl 8.18  1.8315898     NA    NA    NA
## 13          1-isopropyl-4,5-dimethyl 8.04  1.7630970     NA    NA    NA
## 14 1-(2-hydroxyethyl)-4,5-dimethyl 7.94  4.7215222     NA    NA    NA
## 15                     2,4,5-triethyl 7.99  1.2314761     NA    NA    NA
## 16               1,2,4,5-tetramethyl 9.05 11.4875431 -0.030  -0.4   1.0
## 17           1-ethyl-2,4,5-trimethyl 8.68 11.7624206     NA    NA    NA
## 18          2,4,5-trimethyl-1-propyl 8.89 10.0248671     NA    NA    NA
## 19      1-isopropyl-2,4,5-trimethyl 8.80  5.2906533     NA    NA    NA
## 20           2-ethyl-1,4,5-trimethyl 9.03  8.8639686     NA    NA    NA
## 21         1,4,5-trimethyl-2-pyorpyl 9.04  7.6177392     NA    NA    NA
```

Simple figure example

Note that to obtain both the scatter plot and line, you simply have to "add" the two objects together. For both 'geom_line' and 'geom_point', you feed it the source data (as a dataframe), then map its "aesthetics" (x and y axes, for instance) to the names of the columns of that dataframe. Axis and title labeling (as well as legends and captions if desired) is done with the 'labs()' function in a similar fashion.

```
ggplot() +
  # Plot the raw data and line
  geom_line(data = fit.plot, mapping = aes(x = x, y = y), color = "red") +
  geom_point(data = points, mapping = aes(x = pka, y = Keq.C25), color = "black") +
  # Labeling
  labs(x = 'x-axis label', y = 'y-axis label', subtitle = 'subtitle', title = 'title')
```



Note that the default setting has: * a grey background to the plot * relatively small font sizes * grid lines for the x- and y-axes

These are all manipulated with the 'theme()' function, which can be added on at the end. There are also a number of pre-set setting such as 'theme_bw()' and 'theme_classic()'. However, for this, I constructed my own setting function as 'plot.paper()' and 'plot.presentation()' as settings for papers and presentations, respectively, based on my style preferences.

You can use the function 'View()' to look at the underlying code for a built-in function (eg. 'View(theme_bw)') gives you the underlying code for this theme function. This can be useful for seeing how the built-in themes work so you can build your own.

Manipulating the figure

Demo of the code to make a labeled figure, showing it for both the presentation and paper settings. Note that the only difference between the code for these two figures is the last line ('plot.presentation()' vs 'plot.paper()') and the values of the size variables ('ptsz' and 'label.sz').

To obtain only labels for the selected points (those with the 'adj' variables), I use the 'filter()' function from the 'dplyr' package. This function takes a data frame such as the imported data and filters it to those that satisfy a list of conditions, eg.:

'filter(dataframe1, colName1 > X)'

will filter 'dataframe1' to only rows where 'colName1' is greater than X.

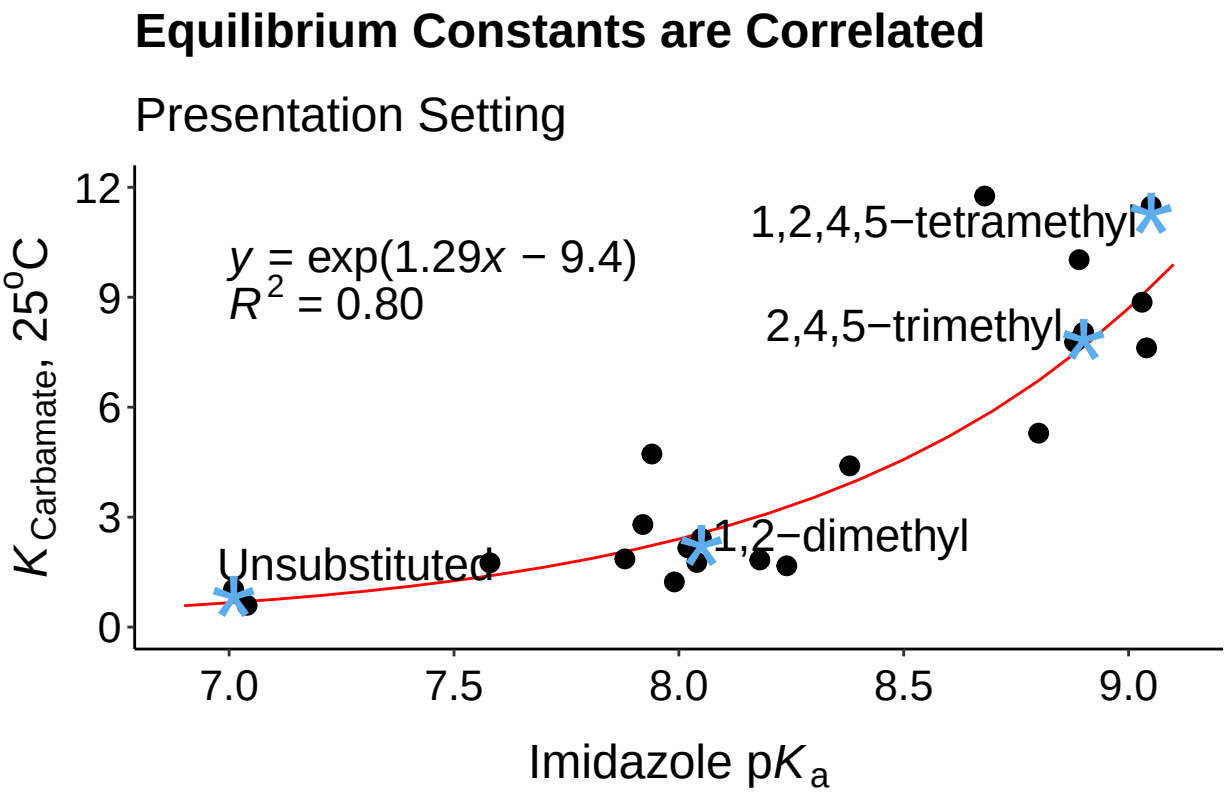
In this case, since only rows with labels we want are filled, this is done with:

'filter(points, !is.na(adj.x))'

where 'is.na(x)' checks where 'x' has a value of 'NaN', which, for a dataframe, means the row is empty. Putting the '!' in front means 'NOT' in formal logic, meaning this line of code filters 'points' to rows that have a filled value for 'adj.x'.

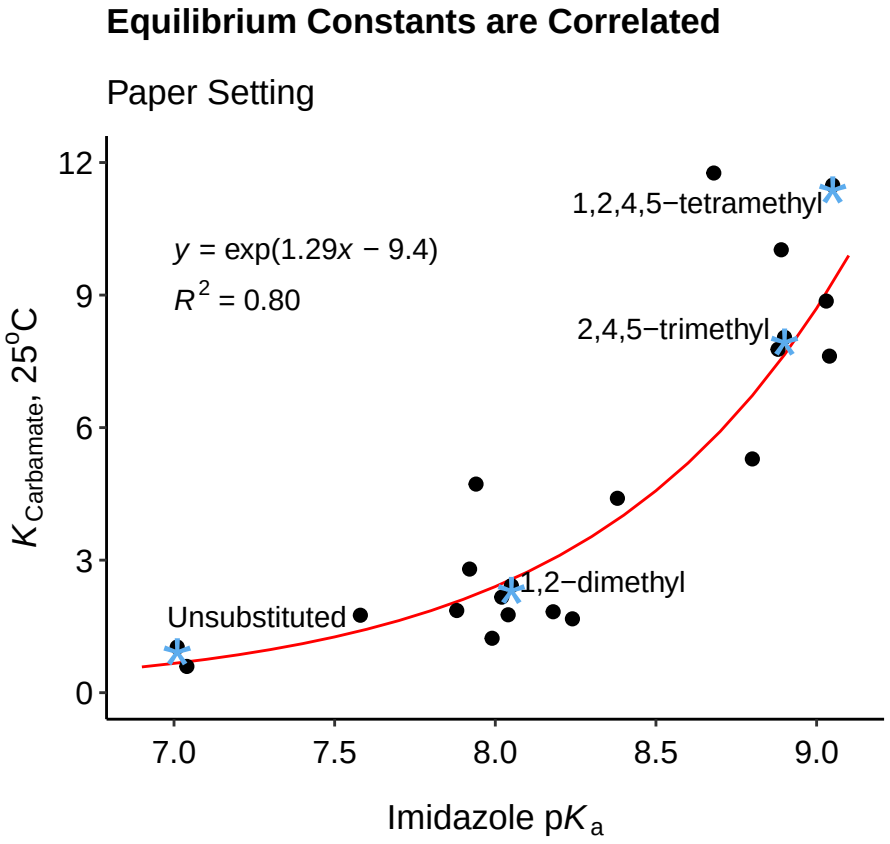
This can be used for plotting a subset of data for highlighting and/or labeling, as shown below.

```
# Relevant plotting info: point and label sizes. For the presentation format:
ptsz = 3
label.sz = 6

ggplot() +
  # Plot the raw data and line
  geom_line(data = fit.plot, mapping = aes(x = x, y = y), color = "red") +
  geom_point(data = points, mapping = aes(x = pka, y = Keq.C25), color = "black",
             size = ptsz) +

  # Label the fit line and R squared
  geom_text(mapping = aes(x = 7, y = 10), label =
            expression(italic(y)*" = exp(1.29"*italic(x)*" - 9.4)"),
            hjust = 0, size = label.sz) +
  geom_text(mapping = aes(x = 7, y = 9), label = expression(italic(R)^2*" = 0.80"),
            hjust = 0, size = label.sz) +

  # Label specific points
  geom_text(data = filter(points, !is.na(adj.x)),
            mapping = aes(x = pka, y = Keq.C25, label = Substitution),
            # Can "nudge" the points slightly as well as control the line adjustment
            # This is best done one-at-a-time, but for demonstration,
            # these have already been tuned for visuals
            nudge_y = filter(points, !is.na(adj.x))$adj.y,
            nudge_x = filter(points, !is.na(adj.x))$adj.x,
            hjust = filter(points, !is.na(adj.x))$hjust,
            size = label.sz) +
  geom_point(data = filter(points, !is.na(adj.x)),
             mapping = aes(x = pka, y = Keq.C25), color = 'steelblue2',
             size = ptsz*5, shape = '*') +
  # Labels: the expression() function allows for sub/superscripts and italics as needed
  labs(x = expression("Imidazole p"*italic(K)*""[a]),
       y = expression(italic(K)[Carbamate]*", 25"^o*"C"),
       subtitle = 'Presentation Setting', title = 'Equilibrium Constants are Correlated') +
  # Set the scale and labels of the y-axis. The same can be done with the
  # x-axis by replacing the 'y' with 'x'
  scale_y_continuous(limits = c(0, 12), breaks = c(0, 3, 6, 9, 12)) +
  # Apply the presentation setting
  plot.presentation()
```




```
# Relevant plotting info: point and label sizes. For the paper format:
ptsz = 2
label.sz = 4

ggplot() +
  # Plot the raw data and line
  geom_line(data = fit.plot, mapping = aes(x = x, y = y), color = "red") +
  geom_point(data = points, mapping = aes(x = pka, y = Keq.C25), color = "black",
             size = ptsz) +
  # Label the fit line and R squared
  geom_text(mapping = aes(x = 7, y = 10), label =
              expression(italic(y)*" = exp(1.29"*italic(x)*" - 9.4)"),
            hjust = 0, size = label.sz) +
  geom_text(mapping = aes(x = 7, y = 9), label = expression(italic(R)^2*" = 0.80"),
            hjust = 0, size = label.sz) +
  # Label specific points
  geom_text(data = filter(points, !is.na(adj.x)),
            mapping = aes(x = pka, y = Keq.C25, label = Substitution),
            nudge_y = filter(points, !is.na(adj.x))$adj.y,
            nudge_x = filter(points, !is.na(adj.x))$adj.x,
            hjust = filter(points, !is.na(adj.x))$hjust,
            size = label.sz) +
  geom_point(data = filter(points, !is.na(adj.x)),
             mapping = aes(x = pka, y = Keq.C25), color = 'steelblue2',
             size = ptsz*5, shape = '*') +
  # Labels: the expression() function allows for sub/superscripts and italics as needed
  labs(x = expression("Imidazole p"*italic(K)*""[a]),
       y = expression(italic(K)[Carbamate]*", 25"^o*"C"),
       subtitle = 'Paper Setting', title = 'Equilibrium Constants are Correlated') +
```

```
  scale_y_continuous(limits = c(0, 12), breaks = c(0, 3, 6, 9, 12)) +
  # Apply the paper setting
  plot.paper()
```

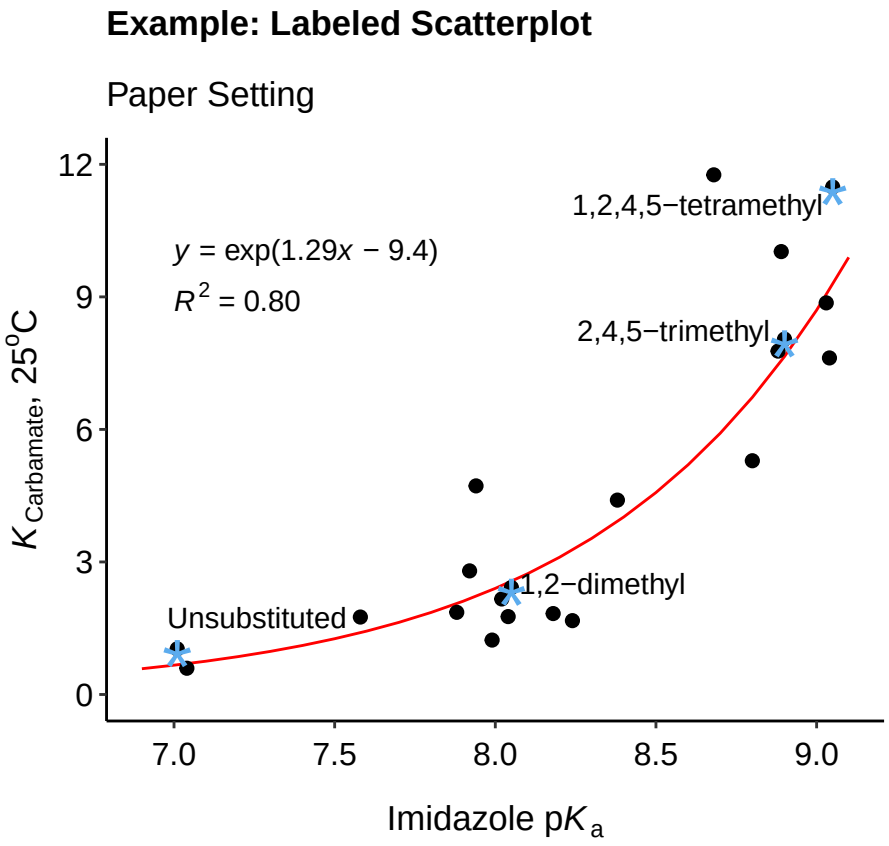


Saving figures

To save the previous figure, use 'ggsave(filename, width, height, units, dpi)'. This produces a file of 'filename' with the width and height specified, using 'units' ('in', 'cm', or 'mm' for inches, centimeters, or millimeters, respectively) at 'dpi' resolution. When using this format, the 'filename' requires the filetype extension, most commonly as '.png', '.jpg', '.pdf', '.svg'. Note that saving to '.svg' may require additional packages such as 'svglite' depending on the operating system and version.

```
ggplot() +
  # Plot the raw data and line
  geom_line(data = fit.plot, mapping = aes(x = x, y = y), color = "red") +
  geom_point(data = points, mapping = aes(x = pka, y = Keq.C25), color = "black",
             size = ptsz) +
  # Label the fit line and R squared
  geom_text(mapping = aes(x = 7, y = 10), label =
              expression(italic(y)*" = exp(1.29"*italic(x)*" - 9.4)"),
            hjust = 0, size = label.sz) +
  geom_text(mapping = aes(x = 7, y = 9), label = expression(italic(R)^2*" = 0.80"),
            hjust = 0, size = label.sz) +
  # Label specific points
  geom_text(data = filter(points, !is.na(adj.x)),
            mapping = aes(x = pka, y = Keq.C25, label = Substitution),
            nudge_y = filter(points, !is.na(adj.x))$adj.y,
```

```
            nudge_x = filter(points, !is.na(adj.x))$adj.x,
            hjust = filter(points, !is.na(adj.x))$hjust,
            size = label.sz) +
  geom_point(data = filter(points, !is.na(adj.x)),
             mapping = aes(x = pka, y = Keq.C25), color = 'steelblue2',
             size = ptsz*5, shape = '*') +
  # Labels: the expression() function allows for sub/superscripts and italics as needed
  labs(x = expression("Imidazole p"*italic(K)*""[a]),
       y = expression(italic(K)[Carbamate]*", 25"^o*"C"),
       subtitle = 'Paper Setting', title = 'Example: Labeled Scatterplot') +
  scale_y_continuous(limits = c(0, 12), breaks = c(0, 3, 6, 9, 12)) +
  # Apply the paper setting
  plot.paper()
```




```
ggsave(filename = 'Figures/01ScatterExample.pdf',
       width = fig.pap.width, height = fig.pap.height, dpi = 300, units = 'in')
ggsave(filename = 'Figures/01ScatterExample.png',
       width = fig.pap.width, height = fig.pap.height, dpi = 300, units = 'in')
ggsave(filename = 'Figures/01ScatterExample.svg',
       width = fig.pap.width, height = fig.pap.height, dpi = 300, units = 'in')
```

```
rm(points, fit.plot)
```

Multi-bar plot

This data is derived from unpublished data with relevant variables removed. Broadly, it describes the probability that a specific model makes an error, testing it across different dataset sampling methods, test cases, and model construction. For ease of understanding, the variable names have been adjusted accordingly to 'Model1' to 'Model3' for instance. The error rates include a standard deviation, which will be used to show how to put error bars on this type of figure.

Import example data

```
data = read.csv('Data_MultiBar.csv')
data
```

```
##      model   sample test.sys   error.rate    error.sd
## 1  Model 1 Sample 1   Test 1 0.2288386769 0.042008515
## 2  Model 1 Sample 2   Test 1 0.1776874382 0.038224941
## 3  Model 1 Sample 3   Test 1 0.2383660462 0.042608412
## 4  Model 1 Sample 1   Test 2 0.0009052678 0.003007405
## 5  Model 1 Sample 2   Test 2 0.0004255632 0.002062479
## 6  Model 1 Sample 3   Test 2 0.0346196002 0.018281434
## 7  Model 1 Sample 1   Test 3 0.0665151121 0.024918036
## 8  Model 1 Sample 2   Test 3 0.0193136796 0.013762508
## 9  Model 1 Sample 3   Test 3 0.0409598644 0.019819726
## 10 Model 2 Sample 1   Test 1 0.1147000000 0.031865955
## 11 Model 3 Sample 1   Test 1 0.1332000000 0.033979076
## 12 Model 2 Sample 2   Test 1 0.0999000000 0.029986662
## 13 Model 3 Sample 2   Test 1 0.1479000000 0.035500083
## 14 Model 2 Sample 3   Test 1 0.1098000000 0.031264030
## 15 Model 3 Sample 3   Test 1 0.1376000000 0.034447967
## 16 Model 2 Sample 1   Test 2 0.1340000000 0.034065232
## 17 Model 3 Sample 1   Test 2 0.1252000000 0.033094555
## 18 Model 2 Sample 2   Test 2 0.1357000000 0.034246972
## 19 Model 3 Sample 2   Test 2 0.1986000000 0.039894616
## 20 Model 2 Sample 3   Test 2 0.1409000000 0.034791837
## 21 Model 3 Sample 3   Test 2 0.1249000000 0.033060549
## 22 Model 2 Sample 1   Test 3 0.1340000000 0.034065232
## 23 Model 3 Sample 1   Test 3 0.1252000000 0.033094555
## 24 Model 2 Sample 2   Test 3 0.1357000000 0.034246972
## 25 Model 3 Sample 2   Test 3 0.1986000000 0.039894616
## 26 Model 2 Sample 3   Test 3 0.1409000000 0.034791837
## 27 Model 3 Sample 3   Test 3 0.1249000000 0.033060549
```

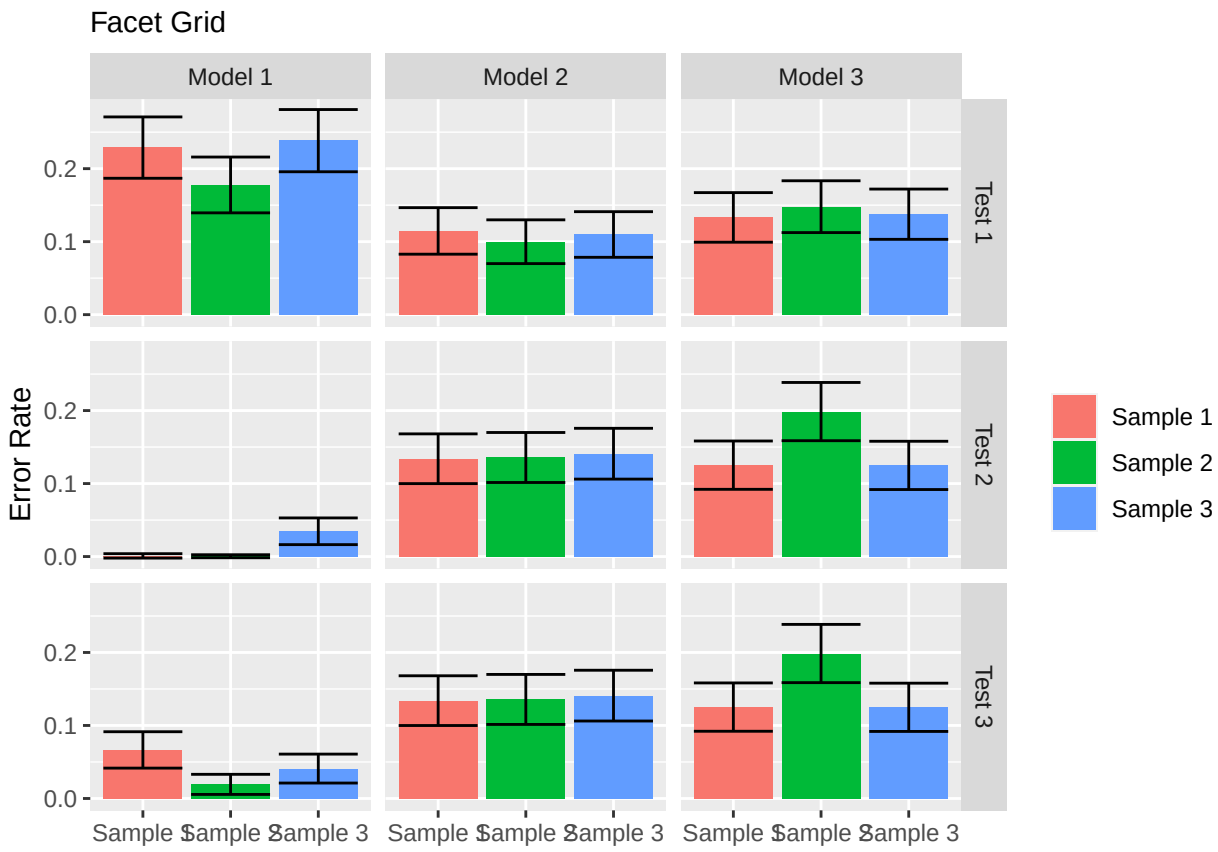
Example default figure

Given that the figure has multiple categorical independent variables, it makes sense to figure out a way to convey the effect of all of these variables in a matrix form than try to isolate single variable effects. This is done with facets, which are automatically-created subplots in 'ggplot2'.

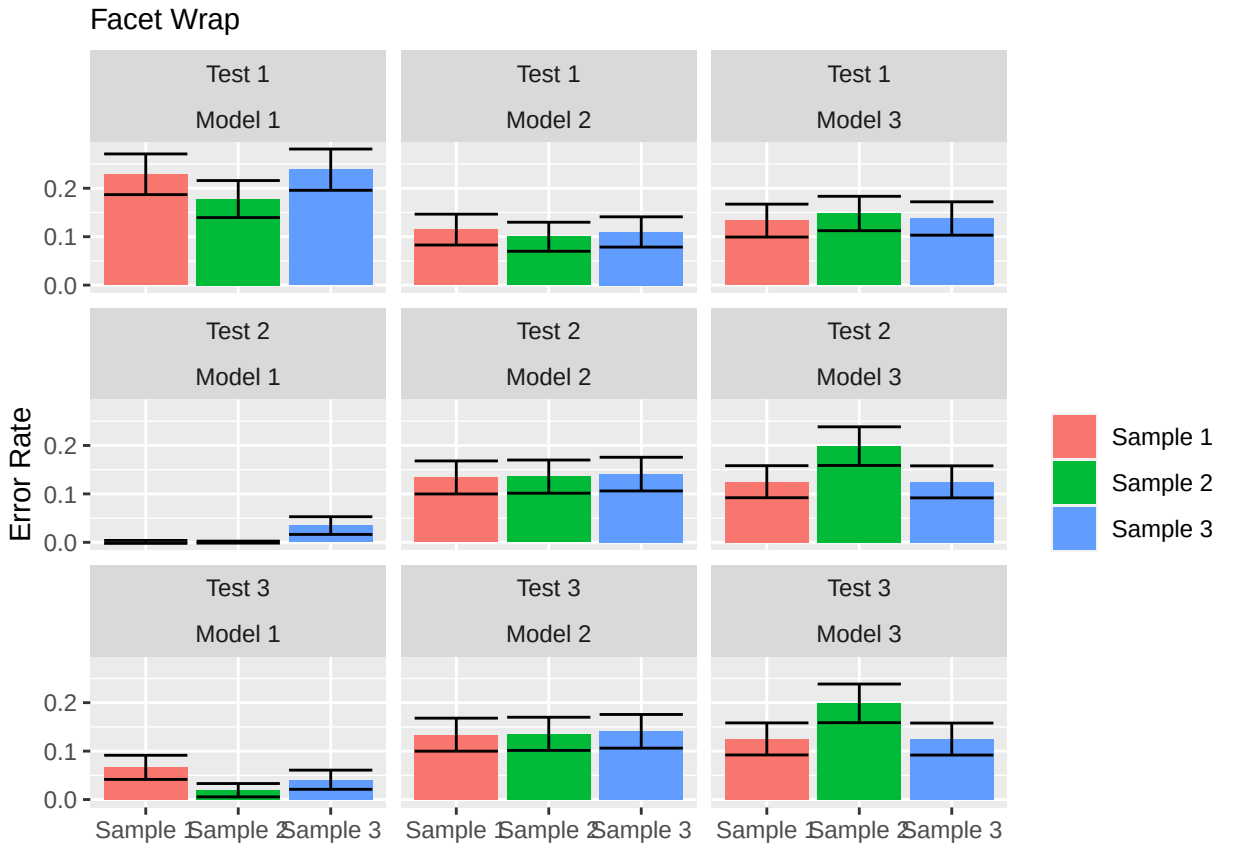
If we set one of the independent variables to the x-axis, the other two can define a matrix of figures according to the rows and columns using 'facet_grid(colVar~rowVar)'. For ease of visualization, it helps for categorical variables to be linked to a fill color as well, in this case using the x-axis variable.

This can also be achieved with 'facet_wrap()', but this function does not force the same grid structure, instead just wrapping according to the number of rows you specific (if none specified, it automatically tries to optimize space filling according to number of subplots it finds).

```
ggplot(data) +
  geom_col(mapping = aes(x = sample, y = error.rate, fill = sample)) +
  geom_errorbar(mapping = aes(x = sample, y = error.rate,
                ymin = error.rate - error.sd, ymax = error.rate + error.sd)) +
  facet_grid(test.sys~model) +
  labs(x = NULL, y = 'Error Rate', fill = NULL, subtitle = 'Facet Grid')
```




```
ggplot(data) +
  geom_col(mapping = aes(x = sample, y = error.rate, fill = sample)) +
  geom_errorbar(mapping = aes(x = sample, y = error.rate,
                ymin = error.rate - error.sd, ymax = error.rate + error.sd)) +
  facet_wrap(test.sys~model) +
  labs(x = NULL, y = 'Error Rate', fill = NULL, subtitle = 'Facet Wrap')
```



Notes: * 'facet_wrap()' leads to a 2-line title, one associated with each of the variables defined for the wrap function. While it does grid nicely according to the variables (eg. the left column is model 1), this is coincidental due to the perfectly symmetric experiment design. * All subplots are forced to the same scale, which can be useful for comparison but make visualization difficult (see Test 2-Model 1, for instance). * The error bars are as wide as the columns, which make them slightly difficult to see for some, as they look almost like a top border. * Only the bottom and left subplots actually have axes. This is more obvious when applying the 'plot.presentation()' or 'plot.paper()' settings, but evident here. * The axes are scaled such that the top and bottom have a spacing of about 5% of the total range of the data. While fine for a scatter plot, for a bar plot, you typically want the bars to start from the bottom.

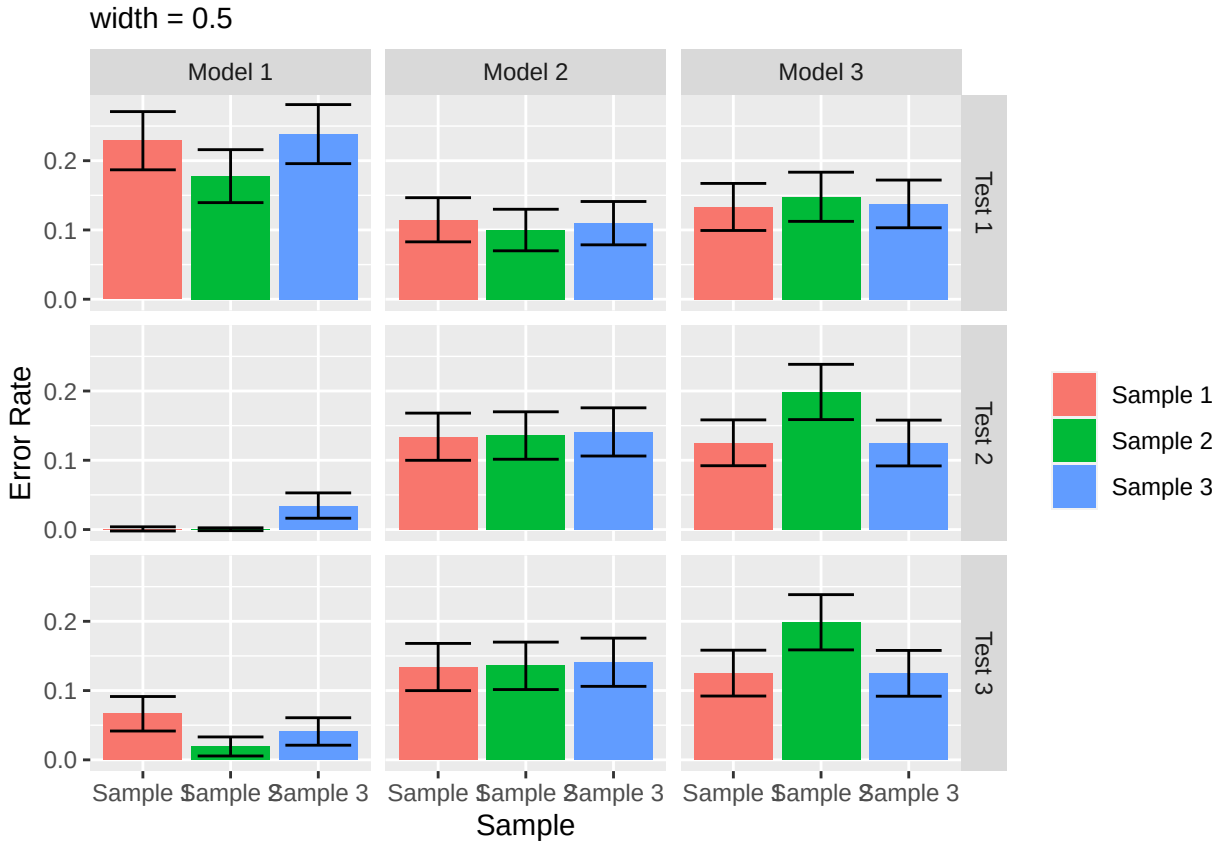
Manipulation of the figure

- For errorbars, there is a width setting that can be used to adjust how wide they are. The default is a width of 1, which is the same width as the bar.
- Because the colors are mapped to the same variable as the x-axis, the x-axis is redundant, and the legend should be positioned closer to it.
- Forcing all subplots to have axes is best done with the 'lemon' package, which includes the functions 'facet_rep_wrap()' and 'facet_rep_grid()'. These work exactly as the 'ggplot2' versions, but can request that tick labels are repeated.
- Asymmetrically adjusting the y-axis needs to be done with the 'expand = expansion(mult = c(a, b))' setting in the y-scale. The first number of 'mult' variable ('a') is the lower bound expansion (0 if none) and 'b' is the upper bound.
- Switching the location of the facet label (ie. from the top and right side to the bottom or left side) is done with by using the 'switch' input to 'facet_grid' or 'facet_wrap'. When switching them, I recommend applying 'theme(strip.placement = "outside")' to ensure that they are outside of the axis labels (this is already included in the 'plot.paper()' and 'plot.presentation()' themes).
- The panels (subplots) may be too close. Adjust them with 'theme(panel.spacing.y = unit(),

panel.spacing.x = unit())', where 'unit()' takes a number followed by the units it should refer to. I recommend applying this after applying the setting functions, as the font sizes have a substantial impact on them. It will also look different if the dimensions of the exported figure are different.

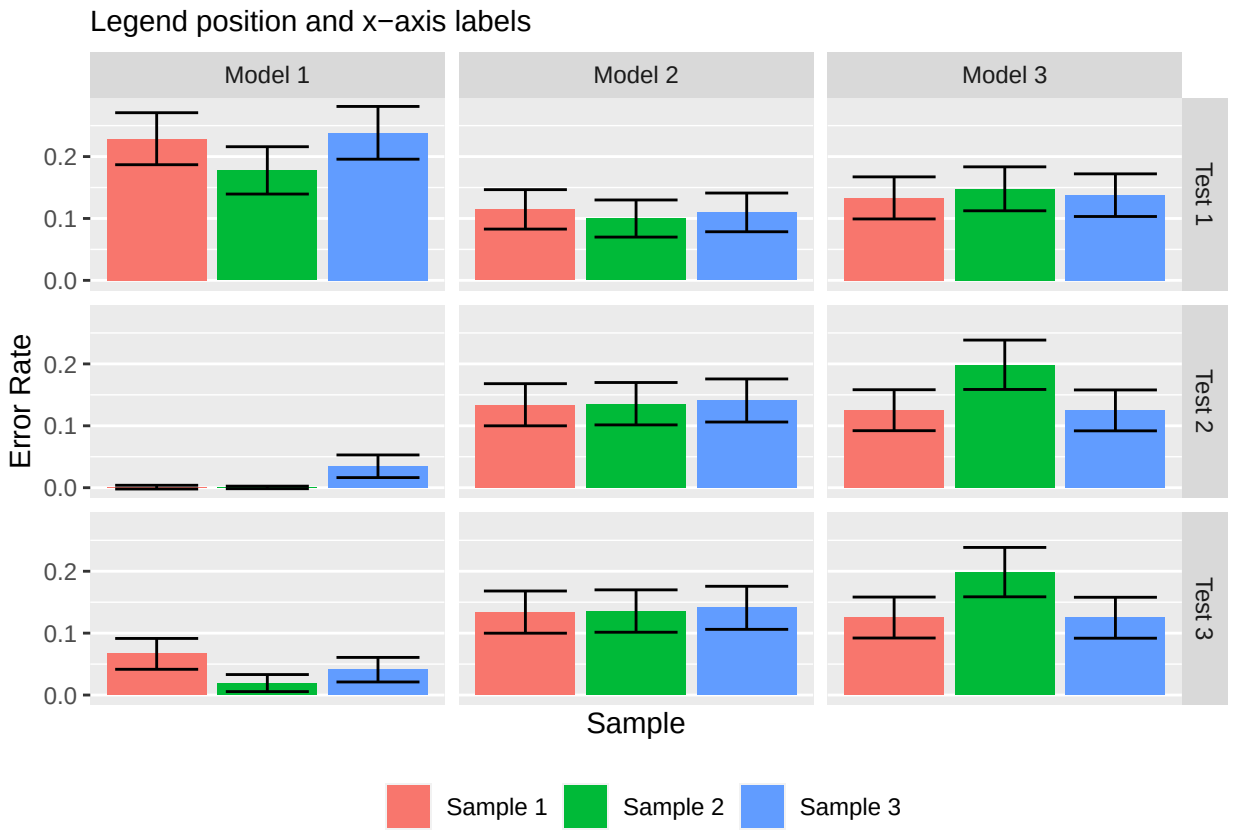
- 'scale_fill_manual(values, labels)' can be used to set color palettes ('values', See: http://sape.inf.usi.ch/sites/default/files/ggplot2-colour-names.png for a list of built-in color names.) as well as adjust labels manually. When adjusting labels, unless linked together in a list, the labels are listed alphabetically.

```
# Bar width
bar.width = 0.75
ggplot(data) +
  # Base figure
  geom_col(mapping = aes(x = sample, y = error.rate, fill = sample)) +
  geom_errorbar(mapping = aes(x = sample, y = error.rate,
    ymin = error.rate - error.sd, ymax = error.rate + error.sd), width = bar.width) +
  facet_grid(test.sys~model) +
  labs(x = 'Sample', y = 'Error Rate', fill = NULL) +
  labs(subtitle = 'width = 0.5')
```

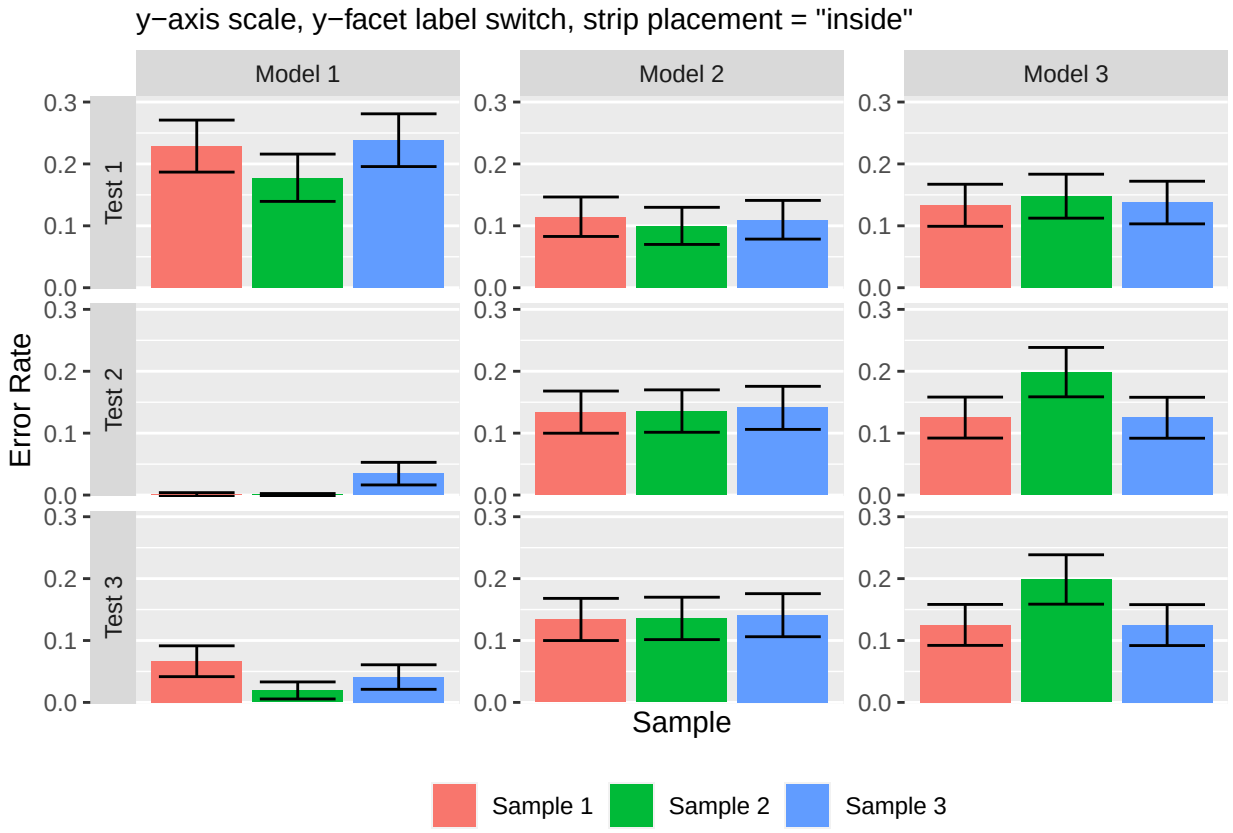



```
# Legend position and orientation, x-axis labels
ggplot(data) +
  # Base figure
  geom_col(mapping = aes(x = sample, y = error.rate, fill = sample)) +
  geom_errorbar(mapping = aes(x = sample, y = error.rate,
    ymin = error.rate - error.sd, ymax = error.rate + error.sd), width = bar.width) +
  facet_grid(test.sys~model) +
  labs(x = 'Sample', y = 'Error Rate', fill = NULL) +
  # Legend position and x-labels
  theme(legend.position = 'bottom', legend.direction = 'horizontal') +
```

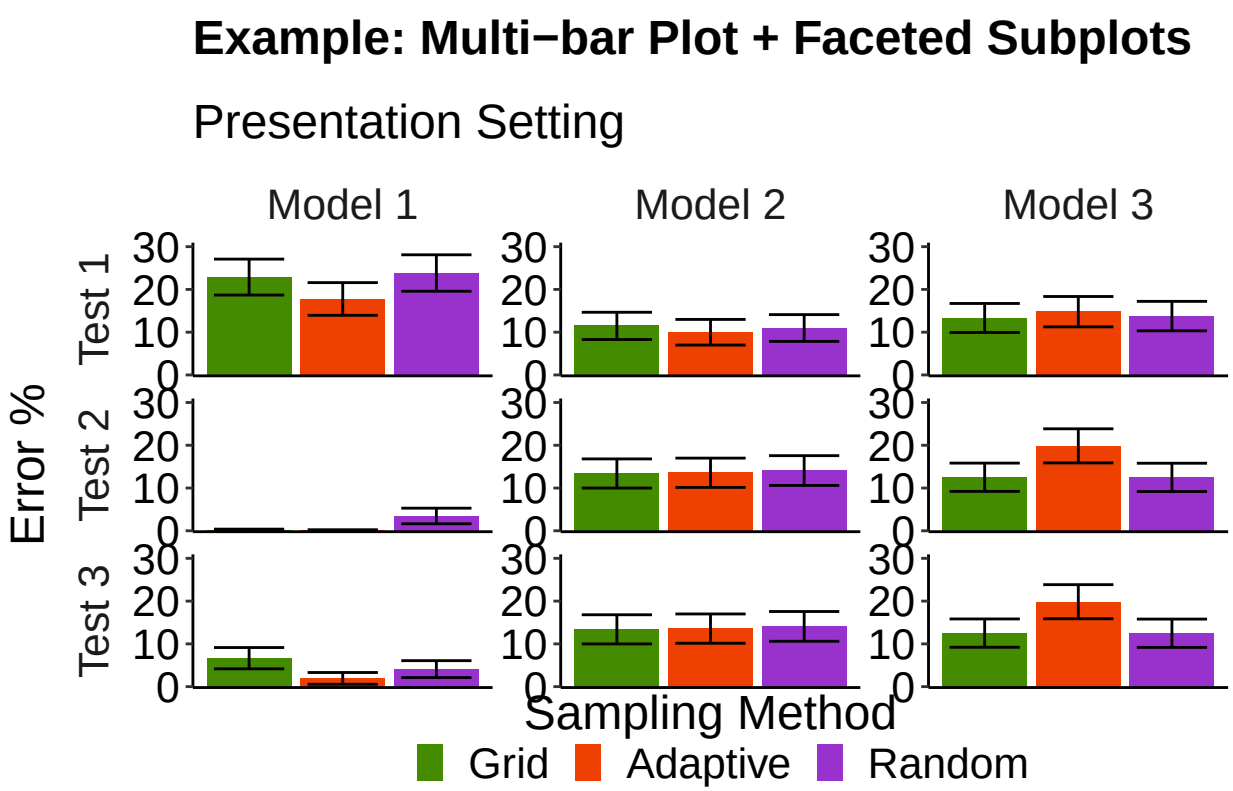
```
  scale_x_discrete(labels = NULL, breaks = NULL) +
  labs(subtitle = 'Legend position and x-axis labels')
```




```
# y-axis range, switch the y facets
ggplot(data) +
  # Base figure
  geom_col(mapping = aes(x = sample, y = error.rate, fill = sample)) +
  geom_errorbar(mapping = aes(x = sample, y = error.rate,
    ymin = error.rate - error.sd, ymax = error.rate + error.sd), width = bar.width) +
  facet_rep_grid(test.sys~model, repeat.tick.labels = TRUE, switch = 'y') +
  labs(x = 'Sample', y = 'Error Rate', fill = NULL) +
  # Legend position and x-labels
  theme(legend.position = 'bottom', legend.direction = 'horizontal') +
  scale_x_discrete(labels = NULL, breaks = NULL) +
  # y scale
  scale_y_continuous(expand = expansion(mult = c(0, 0.1))) +
  labs(subtitle = 'y-axis scale, y-facet label switch, strip placement = "inside"')
```




```
# Apply color palette and the paper formatting option.
color.palette = c('chartreuse4', 'orangered2', 'darkorchid')
ggplot(data) +
  # Base figure
  geom_col(mapping = aes(x = sample, y = error.rate*100, fill = sample)) +
  geom_errorbar(mapping = aes(x = sample, y = error.rate*100,
    ymin = (error.rate - error.sd)*100, ymax = (error.rate + error.sd)*100),
    width = bar.width) +
  scale_fill_manual(values = color.palette,
    # Label adjustment: note that they are specified because they are not in order
                    labels = c('Sample 3' = 'Random',
                               'Sample 2' = 'Adaptive',
                               'Sample 1' = 'Grid')) +
  facet_rep_grid(test.sys~model, repeat.tick.labels = TRUE, switch = 'y') +
  labs(x = 'Sampling Method', y = 'Error %', fill = NULL) +
  # Legend position and x-labels
  plot.presentation() +
  theme(legend.position = 'bottom', legend.direction = 'horizontal',
        panel.spacing.y = unit(0.3, "cm"),
        panel.spacing.x = unit(0.1, "cm"),) +
  scale_x_discrete(labels = NULL, breaks = NULL) +
  # y scale
  scale_y_continuous(expand = expansion(mult = c(0, 0.1))) +
  labs(title = 'Example: Multi-bar Plot + Faceted Subplots',
       subtitle = 'Presentation Setting')
```




```
ggsave(filename = 'Figures/02BarplotExample.pdf',
       width = fig.pre.width, height = fig.pre.height, dpi = 300, units = 'in')
ggsave(filename = 'Figures/02BarplotExample.png',
       width = fig.pre.width, height = fig.pre.height, dpi = 300, units = 'in')
ggsave(filename = 'Figures/02BarplotExample.svg',
       width = fig.pre.width, height = fig.pre.height, dpi = 300, units = 'in')
```

Note that the 'plot.paper()' addition is before the other 'theme()' additions - otherwise, the settings in 'plot.paper()' will overwrite the new theme settings, notably the legend position.

```
rm(data)
```

Heat map

This example will focus on 3 things: * How to create and manipulate a heatmap or contour plot * How to control the axes and legends * How to combine 2 separate plots into a single figure

Import heatmap example data

This data has 2D data (imagine this as latitude and longitude) along a 5 unit x 5 unit square ('input1' and 'input2'), with 2 different "measurements" of interest ('output1' and 'output2').

Some notes on how the data should look first: * In order for the heatmap/contour plot to appear correctly using 'geom_contour' and 'geom_contour_filled', the data must be collected in a perfectly uniform grid (ie. the 'x' spacing between samples must be constant, as should the 'y' spacing, although the 'dx' and 'dy' can be different) * If 'dx' and 'dy' are not constant, the data needs to be interpolated or fitted first, and then the estimated grid data can be plotted. * The measurement values must be continuous for 'geom_contour' and 'geom_contour_filled'. If the measurements are categorical, the figure can be made using 'geom_raster' or 'geom_tile' instead; these will not be shown as examples for brevity.

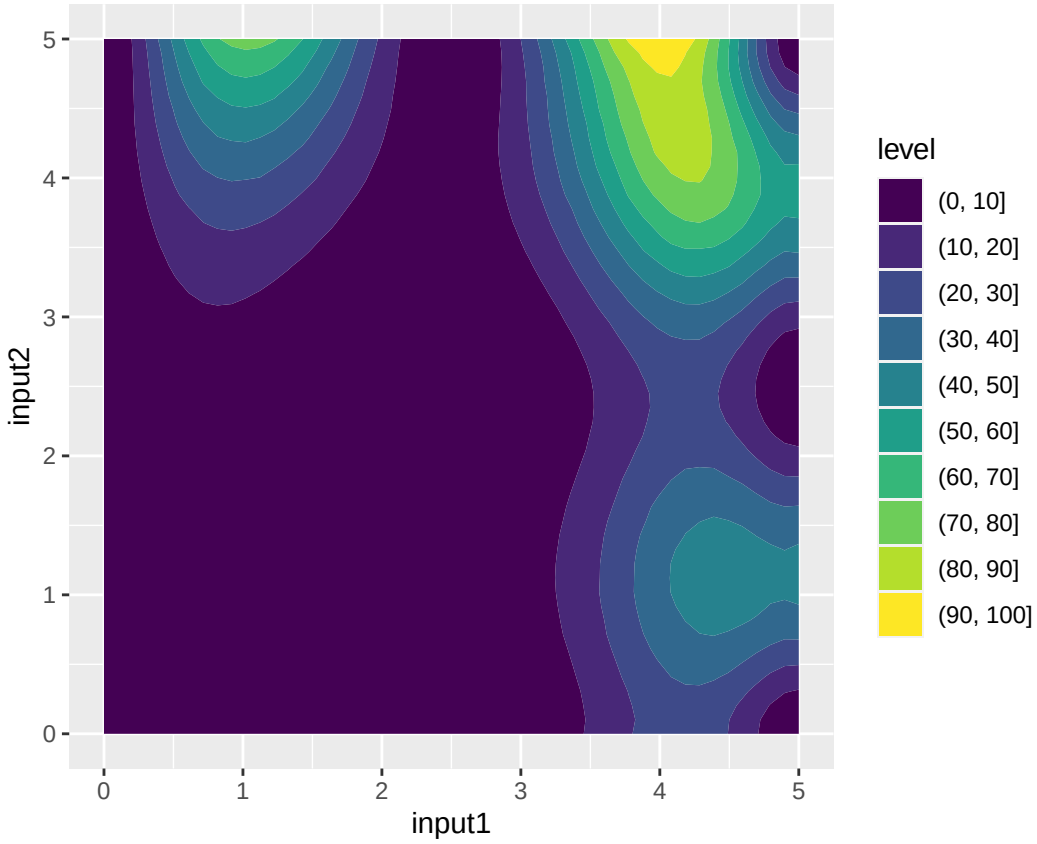
```
data = read.csv('Data_HeatMap.csv')
data[1:10,]
```

```
##       input1 input2      output1       output2
## 1  0.0000000      0 5.337071e-29  0.000000e+00
## 2  0.1020408      0 5.736609e-01 4.515290e-144
## 3  0.2040816      0 1.939479e+00  1.069929e-29
## 4  0.3061224      0 3.578957e+00  2.231781e-10
## 5  0.4081633      0 5.213292e+00  3.219704e-04
## 6  0.5102041      0 6.550154e+00  1.242846e-01
## 7  0.6122449      0 7.557208e+00  2.302028e+00
## 8  0.7142857      0 8.084791e+00  1.112583e+01
## 9  0.8163265      0 8.244515e+00  2.749909e+01
## 10 0.9183673      0 8.088201e+00  4.717320e+01
```

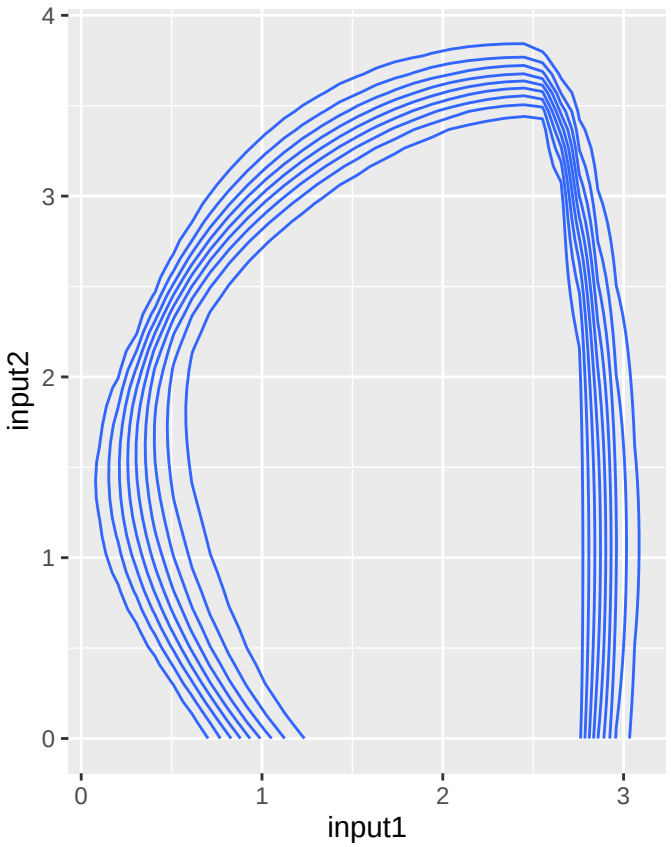
Defaults of a heatmap/contour plot

If you wish for the 'x' and 'y' axes to be at the same scale (eg. if 'x' and 'y' represent miles in the east/west and north/south directions, respectively), the function 'coord_fixed()' will set them to the same scale.

```
# Example heatmap with defaults
ggplot(data) +
  geom_contour_filled(mapping = aes(x = input1, y = input2, z = output1)) +
  coord_fixed()
```




```
ggplot(data) +
  geom_contour(mapping = aes(x = input1, y = input2, z = output2)) +
  coord_fixed()
```



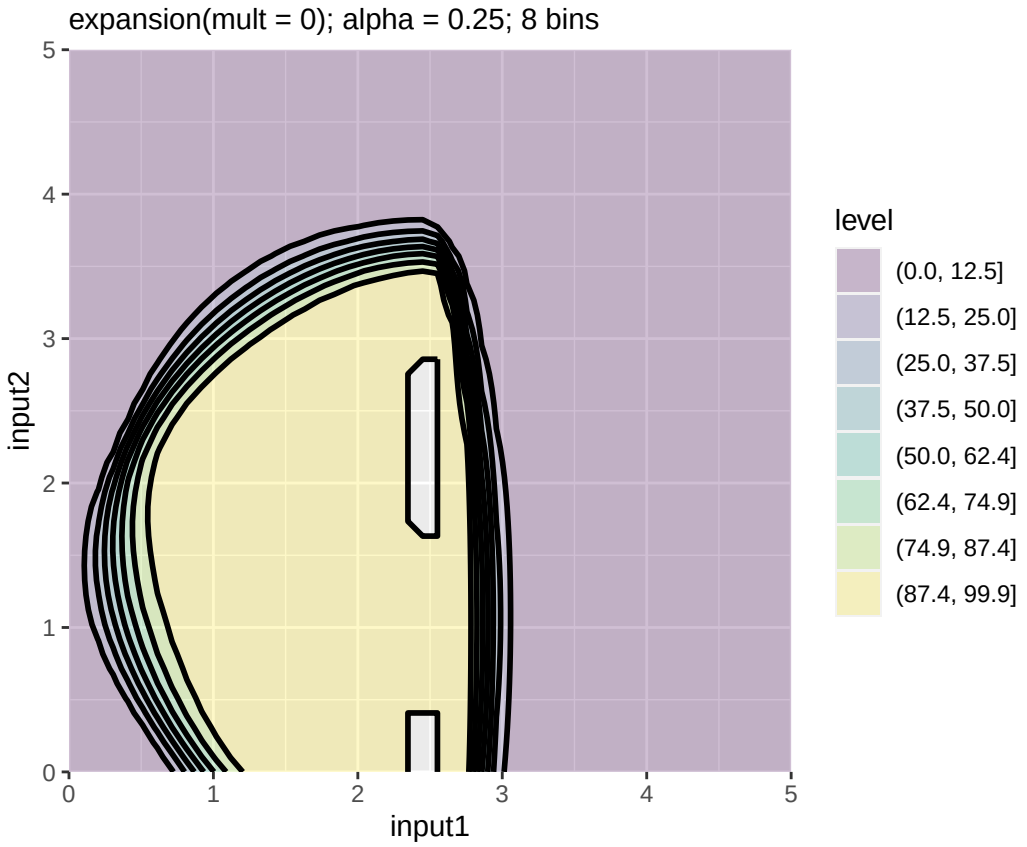
Notes: * 'ggplot2' figures automatically extend the edges of the plot beyond the region of interest (in this case, '[0, 5]'), and thus includes the background grey as a border around it. * Because of the scale of the data, the contour lines that are automatically drawn in the second figure show how this can be disadvantageous since it automatically crops to '[0, 3]' on the x-axis and '[0, 4]' for the y-axis instead of the full extent of the data around '[0, 5]'. * The contour lines default to a blue color, and the heatmap defaults to a purple-green-yellow gradient. Both default to 10 groups/bins.

Manipulation of a heatmap/contour plot

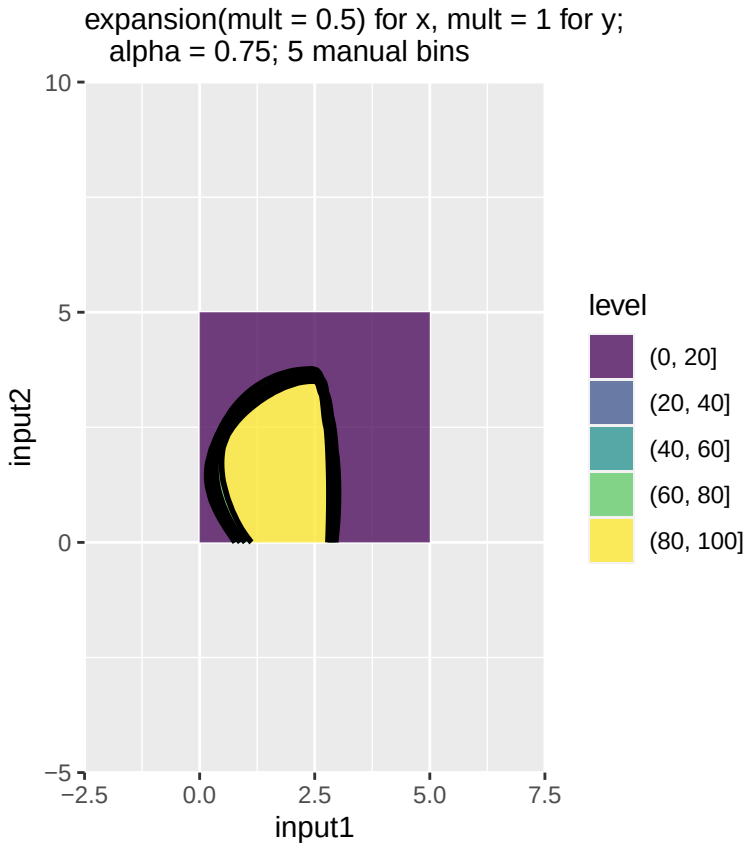
- Eliminating the border around the figure is done with the 'expansion(mult = 0)' option for 'scale_x' and 'scale_y' options. Alternatively, if 'mult = X' where $X > 0$, the bounds will be expanded by a factor of X on each side. The default setting for 'ggplot' is $X = 1.05$
- The 'bins' option controls the number of break points when determining contour lines or colors, not including the minimum. As a result, the number of contour lines or colors you get at the end is actually nbin - 1.
- Sometimes, you want to stack information on top of/below the heatmap. You can control the transparency of the heatmap with 'alpha'. When 'alpha = 1' (default option), the layer is fully opaque; when 'alpha = 0', the layer is transparent (invisible).
- Color schemes can be adjusted with 'scale_fill_brewer(palette)'. See https://www.r-graph-gallery.com/38-rcolorbrewers-palettes.html and https://colorbrewer2.org/ for palette codes.
  - Alternatively, you can create your own gradient with 'scale_fill_gradient()'
- Sometimes the 'nbin' setting will not properly set the bin widths to capture the highest value in the dataset. This often happens when the distribution function tends towards the extremes. You can alternatively use 'breaks' and give a list of the bin boundaries.

```
# Expansion/contraction of the axis boundaries, different alpha values, number of bins
nbin = 8+1
```

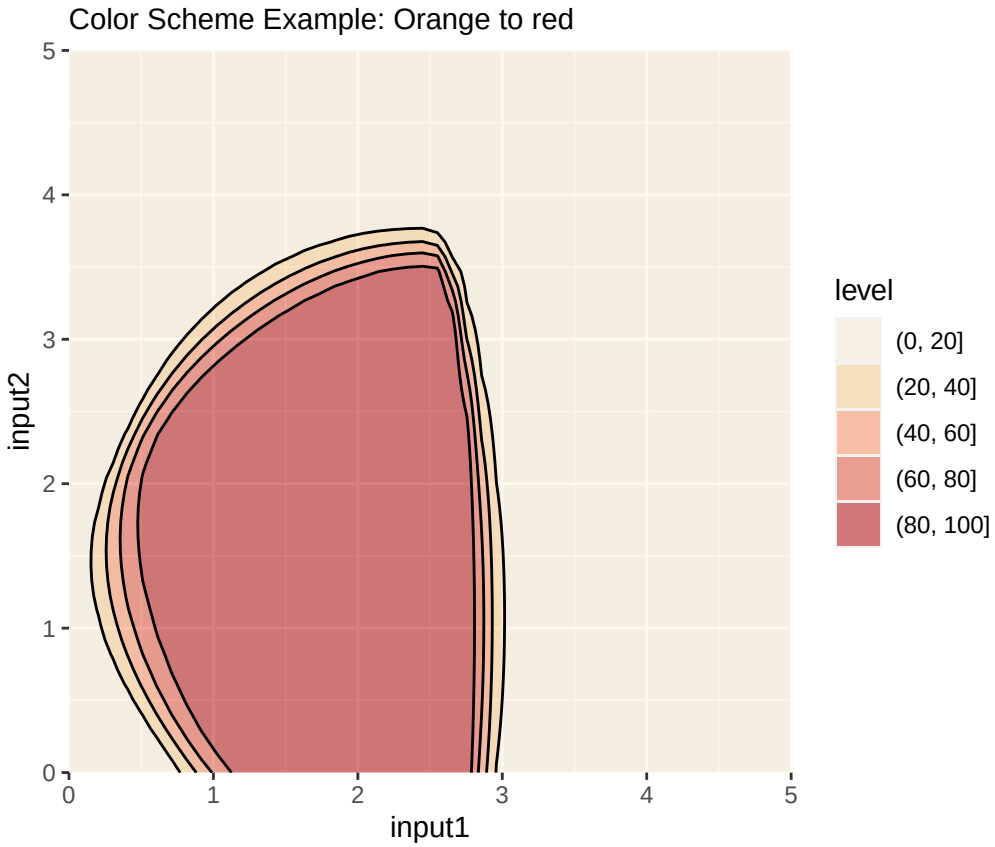
```
alph = 0.25
bound.expand = 0
ggplot(data) +
  geom_contour_filled(mapping = aes(x = input1, y = input2, z = output2),
                      alpha = alph, bins = nbin) +
  geom_contour(mapping = aes(x = input1, y = input2, z = output2),
               size = 1, color = 'black', bins = nbin) +
  scale_x_continuous(expand = expansion(mult = bound.expand)) +
  scale_y_continuous(expand = expansion(mult = bound.expand)) +
  labs(subtitle = 'expansion(mult = 0); alpha = 0.25; 8 bins') +
  coord_fixed()
```




```
# Adjust all of the set values
nbin = 5+1
alph = 0.75
bound.expand = 0.5
ggplot(data) +
  geom_contour_filled(mapping = aes(x = input1, y = input2, z = output2),
    alpha = alph, breaks = seq(from = 0, to = 100, length.out = nbin)) +
  geom_contour(mapping = aes(x = input1, y = input2, z = output2),
    size = 1, color = 'black', breaks = seq(from = 0, to = 100, length.out = nbin)) +
  scale_x_continuous(expand = expansion(mult = bound.expand)) +
  scale_y_continuous(expand = expansion(mult = bound.expand*2)) +
  labs(subtitle = 'expansion(mult = 0.5) for x, mult = 1 for y;
   alpha = 0.75; 5 manual bins') +
  coord_fixed()
```




```
# Adjust the color scheme
bound.expand = 0
alph = 0.5
ggplot(data) +
  geom_contour_filled(mapping = aes(x = input1, y = input2, z = output2),
    alpha = alph, breaks = seq(from = 0, to = 100, length.out = nbin)) +
  geom_contour(mapping = aes(x = input1, y = input2, z = output2),
    size = 0.5, color = 'black', breaks = seq(from = 0, to = 100, length.out = nbin)) +
  scale_x_continuous(expand = expansion(mult = bound.expand)) +
  scale_y_continuous(expand = expansion(mult = bound.expand)) +
  labs(subtitle = 'Color Scheme Example: Orange to red') +
  scale_fill_brewer(palette = 'OrRd') +
  coord_fixed()
```



Combining figures

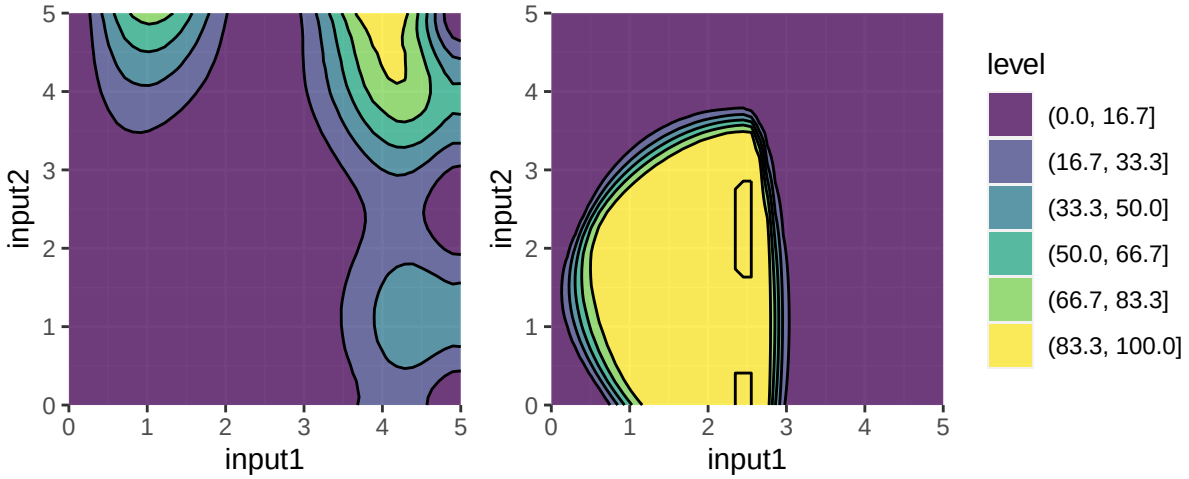
When it is easier to handle the data by placing them in multiple figures instead of using the built-in facet function (for instance, when the figures are different types such as a bar and scatter plot), the 'patchwork' package can be used. Figures can be saved as "objects" under a variable name just like any other variable. The 'patchwork' package allows you to "add" figures together to combine them.

Further detail can be found here: https://patchwork.data-imaginist.com/articles/guides/layout.html%5Bhttps://patchwork.data-imaginist.com/articles/guides/layout.html]

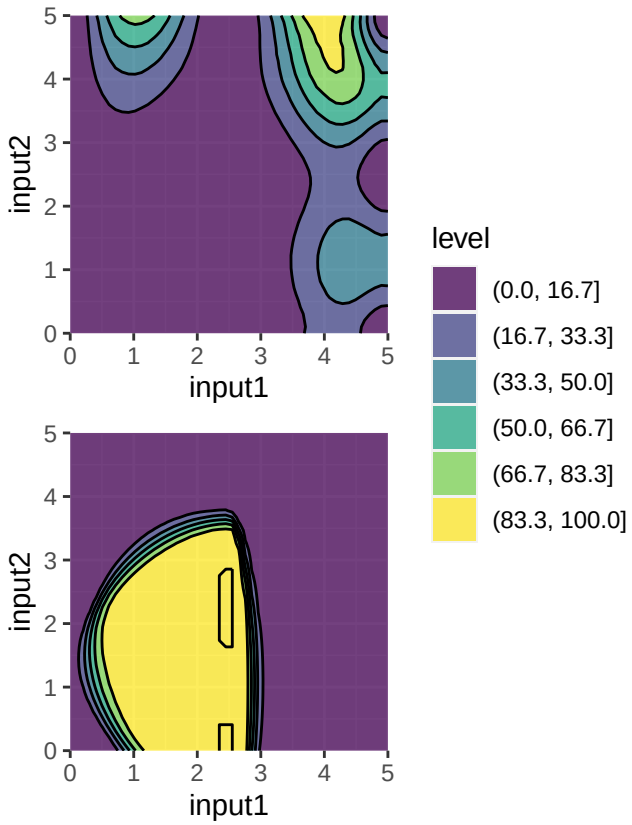
Notes: * If two figures use the same legend, it can be convenient to only include the legend once. This can be done by adding 'plot_layout(guides = "collect")' * When combining multiple figures, it can be useful to control the arrangement of these figures. This can be done with the function 'plot_layout()'. For instance, 'plot_layout(nrow = 2)' ensures there are 2 rows of figures as evenly as possible, filling from left to right, top to bottom, adding blank space at the end as needed. * Sometimes, you want to control where the blank spaces happen. This can be done by using 'plot_spacer()' in the location of the blank space. * If you wish to hide a legend such as the fill colors, use the function 'guides(fill = FALSE)'. These can also be used to manipulate the appearance of the legends - see: https://aosmith.rbind.io/2020/07/09/ggplot2-override-aes/

```
nbin = 6+1; alph = 0.75; ln.wt = 0.5
## Simple addition of 2
g1 = ggplot(data) +
  geom_contour_filled(mapping = aes(x = input1, y = input2, z = output1),
          alpha = alph, breaks = seq(from = 0, to = 100, length.out = nbin)) +
  geom_contour(mapping = aes(x = input1, y = input2, z = output1),
               size = ln.wt, color = 'black', bins = nbin) +
  scale_x_continuous(expand = expansion(mult = bound.expand)) +
  scale_y_continuous(expand = expansion(mult = bound.expand)) +
```

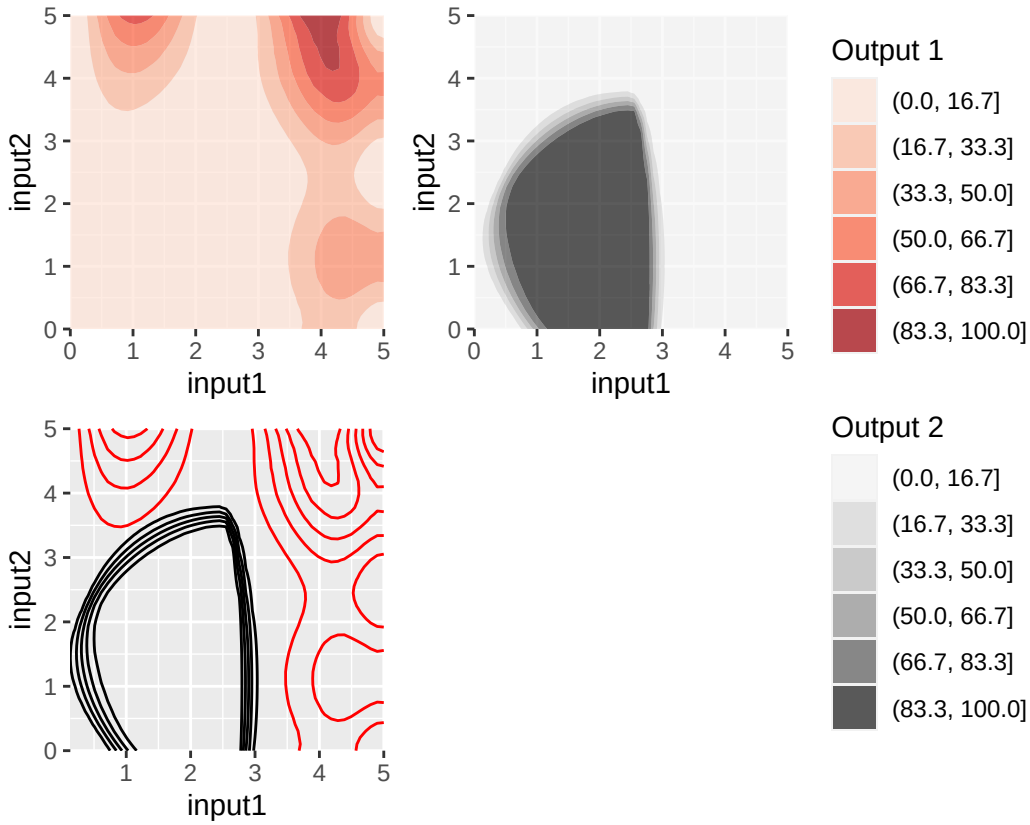
```
  coord_fixed()
g2 = ggplot(data) +
  geom_contour_filled(mapping = aes(x = input1, y = input2, z = output2),
          alpha = alph, breaks = seq(from = 0, to = 100, length.out = nbin)) +
  geom_contour(mapping = aes(x = input1, y = input2, z = output2),
               size = ln.wt, color = 'black', bins = nbin) +
  scale_x_continuous(expand = expansion(mult = bound.expand)) +
  scale_y_continuous(expand = expansion(mult = bound.expand)) +
  coord_fixed()
# Side by side uses addition
g1 + g2 + plot_layout(guides = "collect")
```



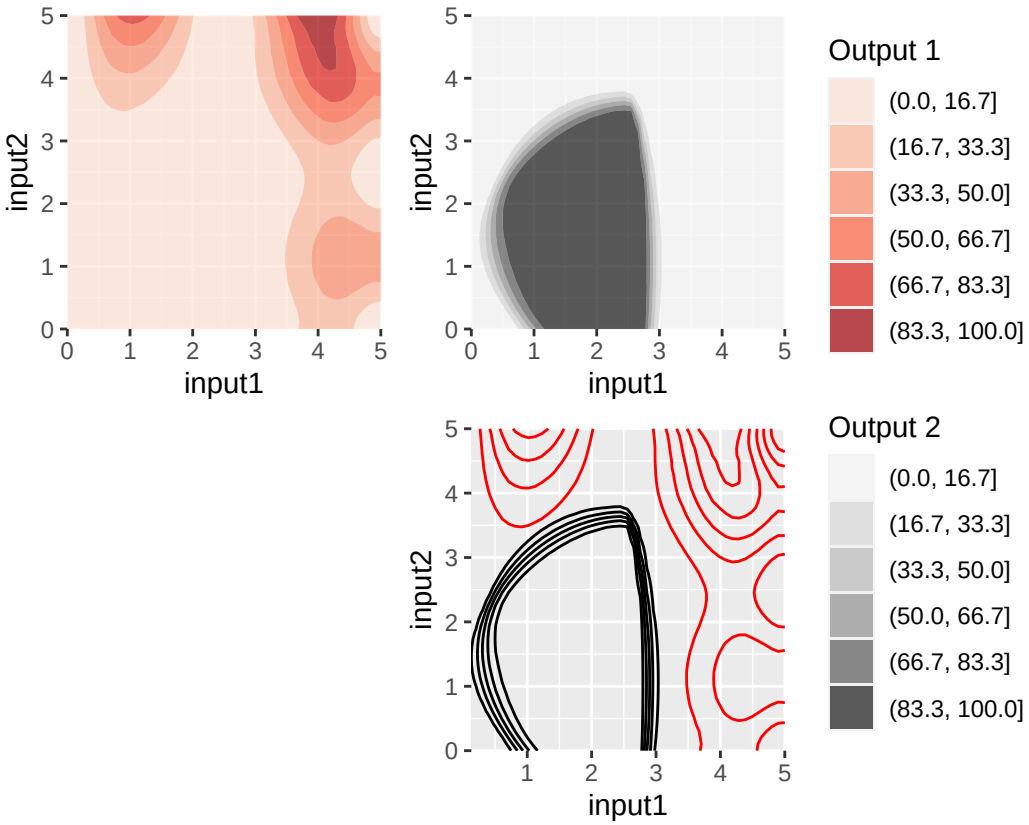

```
# Vertical stacking uses division
(g1 / g2) + plot_layout(guides = "collect")
```




```
## 3 Figures with a spacer
g1 = ggplot(data) +
  geom_contour(mapping = aes(x = input1, y = input2, z = output1),
               size = ln.wt, color = 'red',
               breaks = seq(from = 0, to = 100, length.out = nbin)) +
  geom_contour(mapping = aes(x = input1, y = input2, z = output2),
               size = ln.wt, color = 'black',
               breaks = seq(from = 0, to = 100, length.out = nbin)) +
  scale_x_continuous(expand = expansion(mult = bound.expand)) +
  scale_y_continuous(expand = expansion(mult = bound.expand)) +
  coord_fixed()
g2 = ggplot(data) +
  geom_contour_filled(mapping = aes(x = input1, y = input2, z = output1),
          alpha = alph, breaks = seq(from = 0, to = 100, length.out = nbin)) +
  scale_x_continuous(expand = expansion(mult = bound.expand)) +
  scale_y_continuous(expand = expansion(mult = bound.expand)) +
  coord_fixed() + labs(fill = 'Output 1') +
  scale_fill_brewer(palette = 'Reds')
g3 = ggplot(data) +
  geom_contour_filled(mapping = aes(x = input1, y = input2, z = output2),
          alpha = alph, breaks = seq(from = 0, to = 100, length.out = nbin)) +
  scale_x_continuous(expand = expansion(mult = bound.expand)) +
  scale_y_continuous(expand = expansion(mult = bound.expand)) +
  coord_fixed() + labs(fill = 'Output 2') +
  scale_fill_brewer(palette = 'Greys')
# Example with just setting the number of rows
g2 + g3 + g1 + plot_layout(guides = "collect", nrow = 2)
```



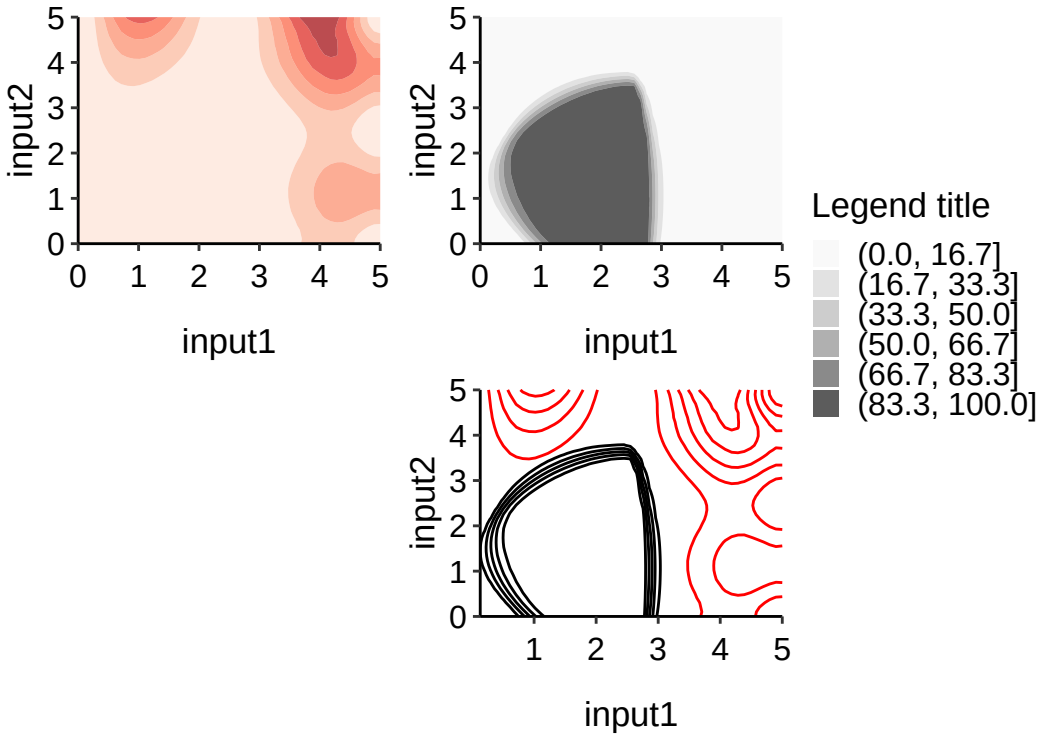

```
# Example placing a spacer so the final figure is shifted by 1
g2 + g3 + plot_spacer() + g1 + plot_layout(guides = "collect", nrow = 2)
```




```
## Hiding a legend, paper presentation mode.
# same g1, just add the paper setting
g1 = g1 + plot.paper()
g2 = ggplot(data) +
  geom_contour_filled(mapping = aes(x = input1, y = input2, z = output1),
          alpha = alph, breaks = seq(from = 0, to = 100, length.out = nbin)) +
  scale_x_continuous(expand = expansion(mult = bound.expand)) +
  scale_y_continuous(expand = expansion(mult = bound.expand)) +
  coord_fixed() +
  scale_fill_brewer(palette = 'Reds') + guides(fill = FALSE) + plot.paper()
g3 = ggplot(data) +
  geom_contour_filled(mapping = aes(x = input1, y = input2, z = output2),
          alpha = alph, breaks = seq(from = 0, to = 100, length.out = nbin)) +
  scale_x_continuous(expand = expansion(mult = bound.expand)) +
  scale_y_continuous(expand = expansion(mult = bound.expand)) +
  coord_fixed() + labs(fill = 'Legend title') +
  scale_fill_brewer(palette = 'Greys') + plot.paper()
# As before, but the red legend has been eliminated. The plot annotation
# function allows for a total figure title; otherwise, the title is
# specific to a single subplot
g2 + g3 + plot_spacer() + g1 + plot_layout(guides = "collect", nrow = 2) +
  plot_annotation(title = 'Example Heatmap/Contour + Patchwork',
                  subtitle = 'Paper setting')
```

Example Heatmap/Contour + Patchwork

Paper setting




```
ggsave(filename = 'Figures/03HeatmapExample.pdf',
       width = fig.pap.width, height = fig.pap.height, dpi = 300, units = 'in')
ggsave(filename = 'Figures/03HeatmapExample.png',
       width = fig.pap.width, height = fig.pap.height, dpi = 300, units = 'in')
ggsave(filename = 'Figures/03HeatmapExample.svg',
       width = fig.pap.width, height = fig.pap.height, dpi = 300, units = 'in')

rm(g1, g2, g3)
```