Data Visualization with ggplot2 (Part 3)

Plots for Specific Data Types (Chapter 2)

Seun Odeyemi

2019-04-14

Contents

Load Libraries

```
library(readr)
library(dplyr)
library(ggplot2)
# library(ggplot2movies)
library(tidyr)
library(skimr)
library(knitr)
library(kableExtra)
library(RColorBrewer)
library(grid)
library(ggthemes)
library(forcats)
library(GGally)
library(here)
library(hexbin)
```

The Graphics of Large Data

In this chapter we will continue our discussion of specialty plot types turning our attention to those suited for specific data types. My goal here is to familiarize you with unique plot types so that you may call upon them we you have suitable data even when that may not be very often. Remember to think of the purpose of a plot: *the more plot types you have in your data viz bag of tricks, the creative and fitting your data visualizations will be.*

To lead us into this topic we review and round out knowledge of working with large data sets. So far we've dealt with relatively small data sets, but once we start working on large data sets we'll run into a number of issues. How we define a large dataset will dictate the problems we'll encounter for visualizations. For example, large can refer to the:

1. Many observations (rows, records, etc.)

a. Very high resolution time series
b. Large surveys
c. Website analytics

2. Many variables (features, characteristics, columns, etc.)

- Multidimensional data

3. A combination

Each situation demands its own solution. There is an obvious overlap between this topic and data handling and storage for large data sets. However, data munging details are beyond this course. We'll focus on the visualization aspects.

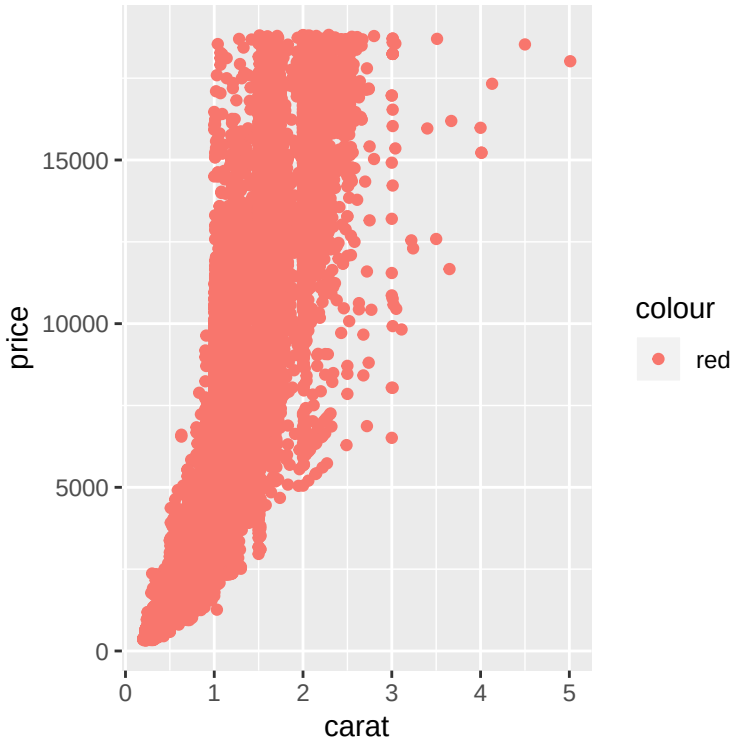
Many Observations

Let's consider many observations. The most straightforward thing to do is to adjust the `geom` we are using. Using the `diamonds` data set as an example—it contains about 50,000 observations relating to diamonds such as price, carat, color, and clarity. For example in this case of high density points such as the scatter plot shown below, we can adjust things like the plotting symbol, point size, and alpha blending to make trends in our data set visible. In the last chapter, we learned that we can also use a 2D densit plot as shown below with contour lines. However, we should realize that this is not a one-stop solution.

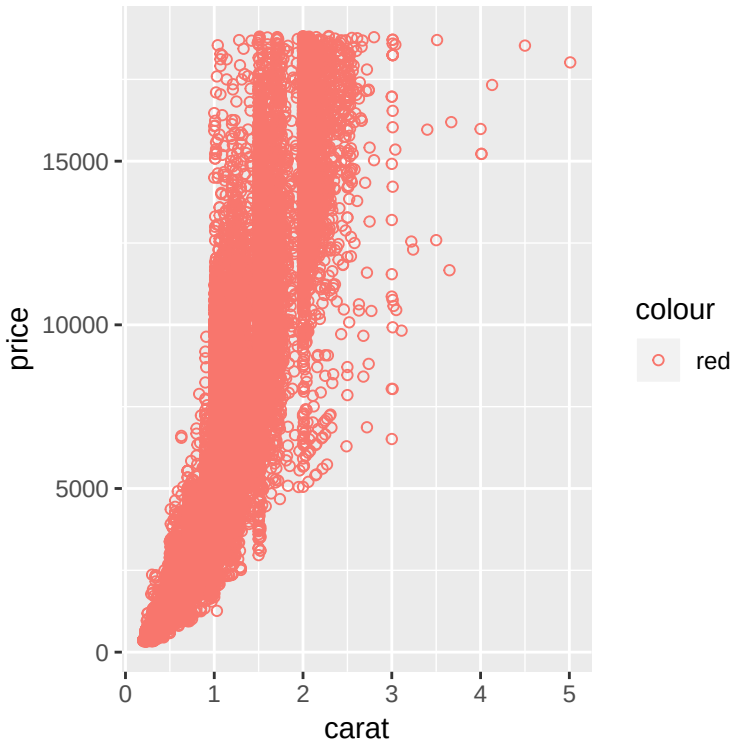
In this case, there is such a high density at the lower end of the scales that we lose too much detail in the rest of the plot. Recall that we can map the actual density, shown by each contour line, to a continuous color scale. However, comparing these plots to the scatter plot with alpha blending shows that neither of 2D density plots really shows a accurate representation of the distribution. An alternative to 2D density plots is to simply bin the values into a grid. This is simply a 2D version of a histogram, which means we can change the `bin` number to increase or decrease the resolution of our plot just like a 1D histogram.

An extension of this, which is popular in infographics is the use of hex binning. This looks just like the binning we've already seen only using hexagons instead of squares. We can adjust the bin size here also. Methods for dealing with many observations are basically concerned with reducing over-plotting or reducing the amount of information that is plotted. This is done by aggregating the data into two-dimensions further removing us from the large amount of raw observations. This is in the hopes of seeing some interesting revelations in the data set. The choice will depend on the data set at hand. So, its worth experimenting with a wide variety of geoms.

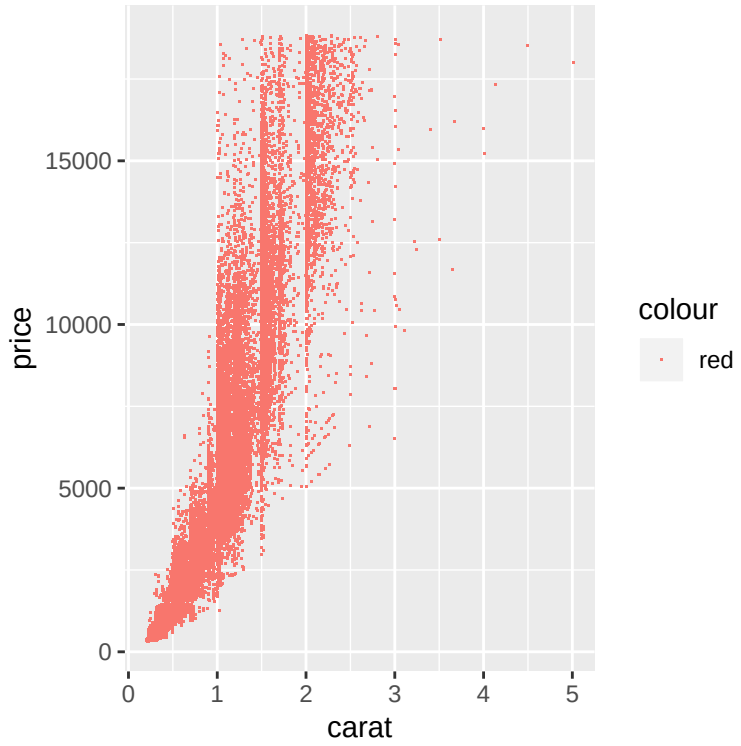
```
ggplot(diamonds, aes(x = carat, y = price, color = "red")) +
  geom_point() #simplified
```



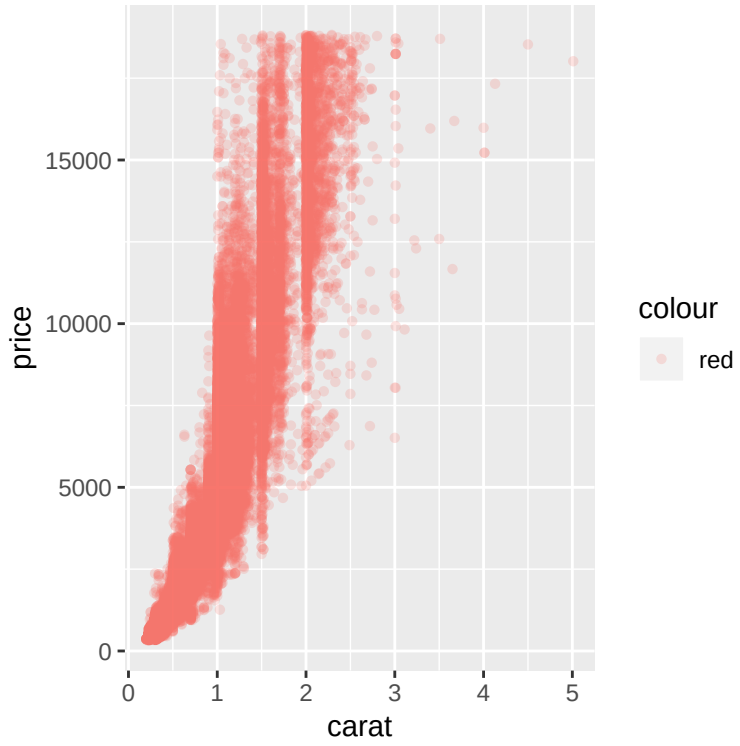

```
ggplot(diamonds, aes(x = carat, y = price, color = "red")) +
  geom_point(shape = 1) #plotting symbol adjusted using shape in the geom layer
```



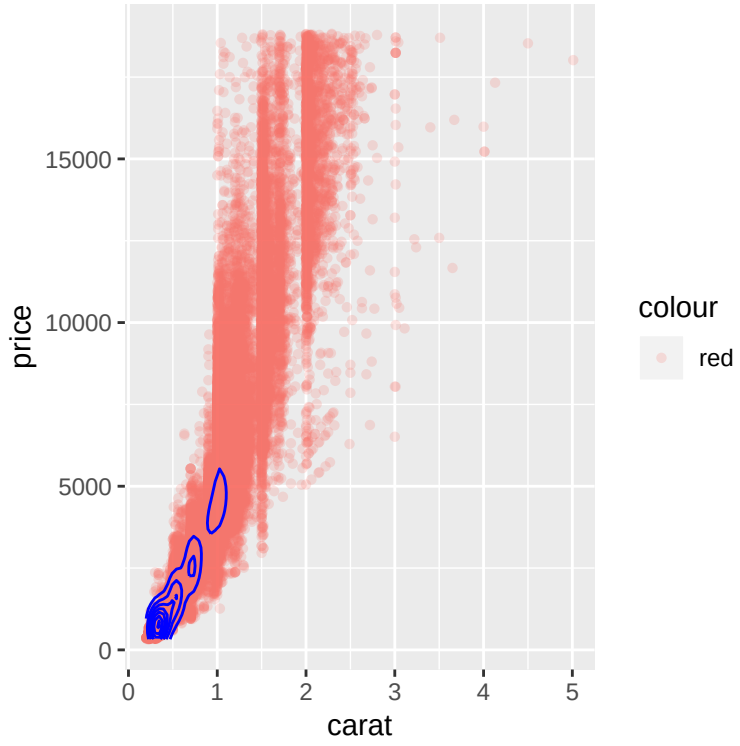
```
ggplot(diamonds, aes(x = carat, y = price, color = "red")) +
  geom_point(shape = ".") #pointsize adjusted
```



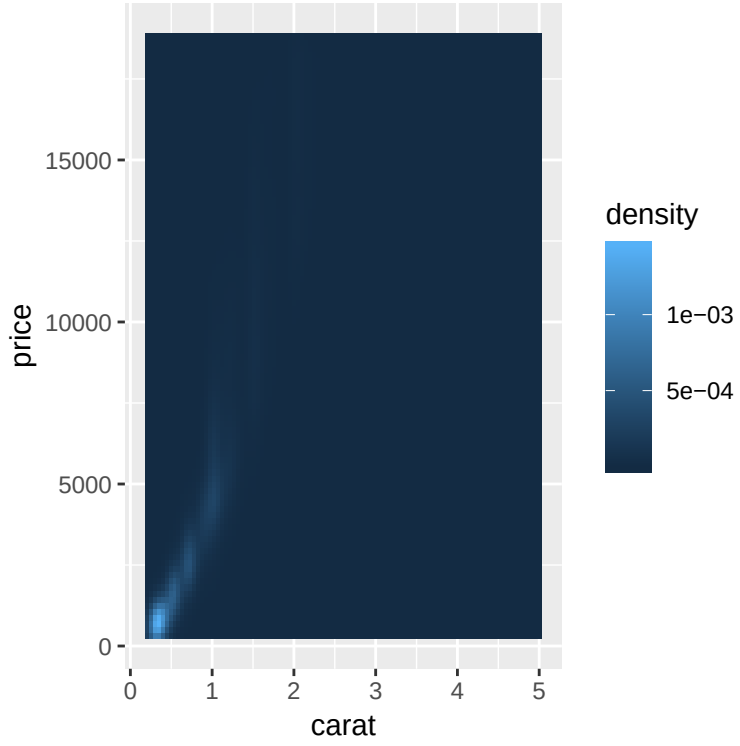

```
ggplot(diamonds, aes(x = carat, y = price, color = "red")) +
  geom_point(shape = 16, alpha = 0.2) #alpha blending
```



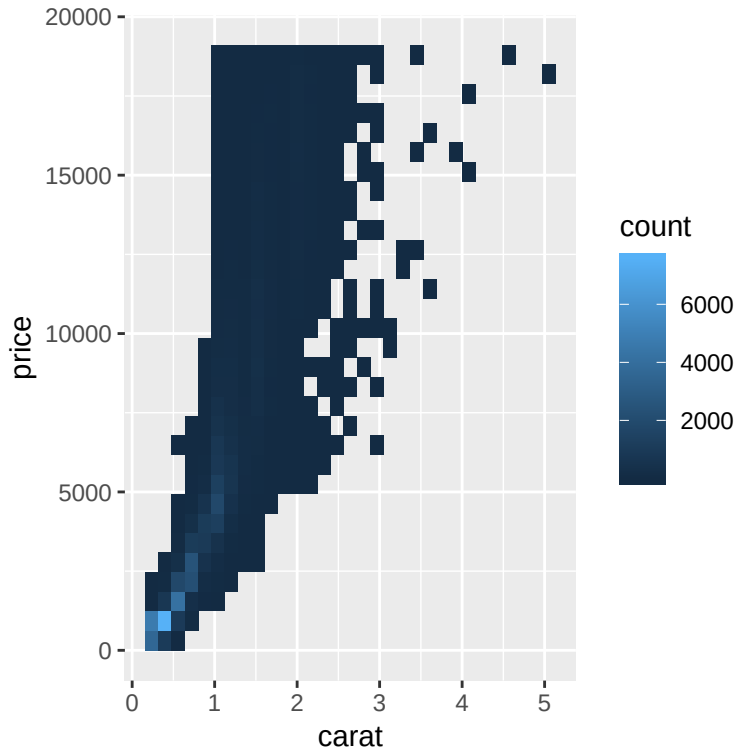

```
ggplot(diamonds, aes(x = carat, y = price, color = "red")) +
  geom_point(shape = 16, alpha = 0.2) + #alpha blending
  stat_density_2d(color = "blue")
```



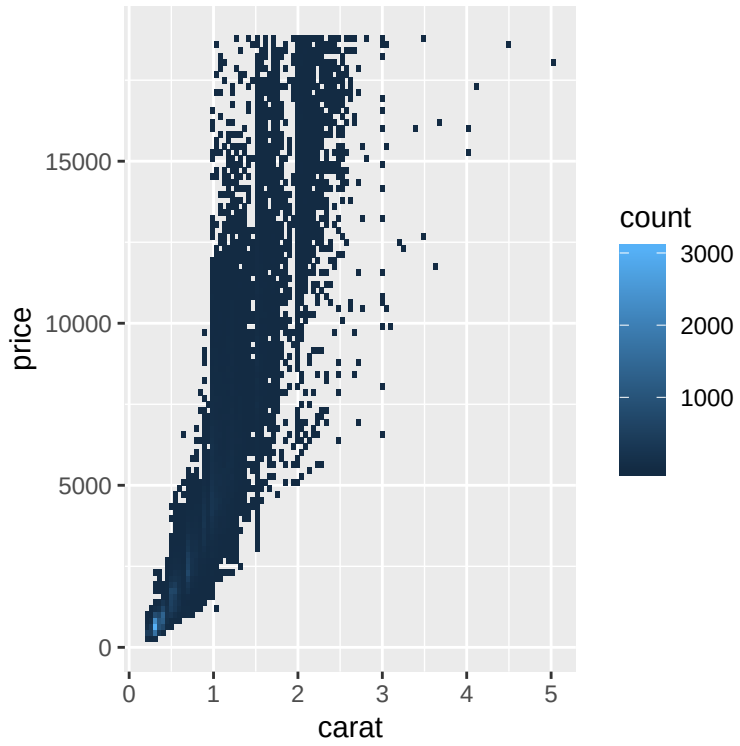
```r
# diamonds <- diamonds %>% mutate(category = cut(carat, breaks = 3, labels = c("low", "medium", "high"))
#
# ggplot(diamonds, aes(x = carat, y = price, color = "red")) +
#   geom_point(shape = 16, alpha = 0.2) + #alpha blending
#   stat_density2d(aes(fill = ..density..))

ggplot(diamonds, aes(x = carat, y = price)) +
 stat_density2d(geom = "tile",
                aes(fill = ..density..),
                contour = FALSE)
```



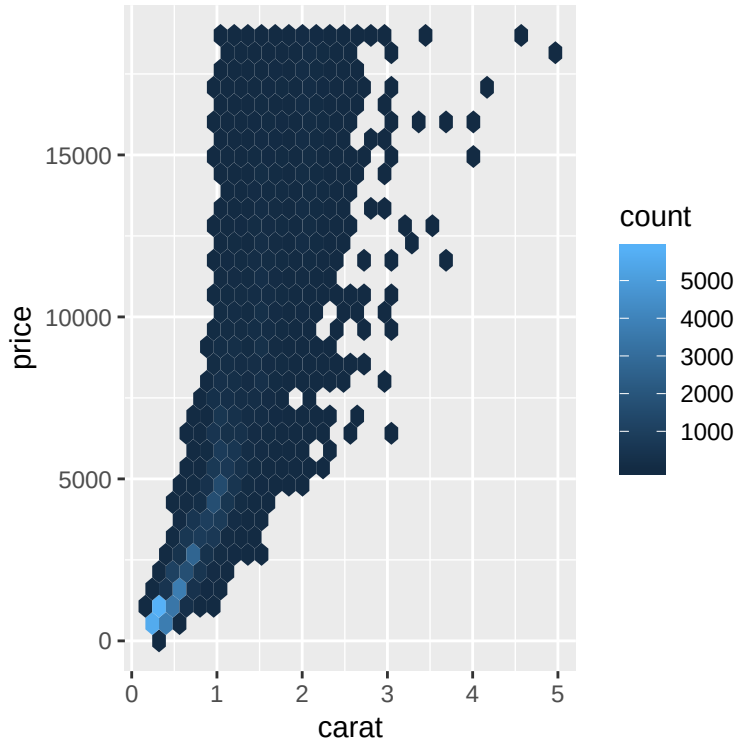

```r
# 2D histogram
ggplot(diamonds, aes(x = carat, y = price)) +
  geom_bin2d()
```



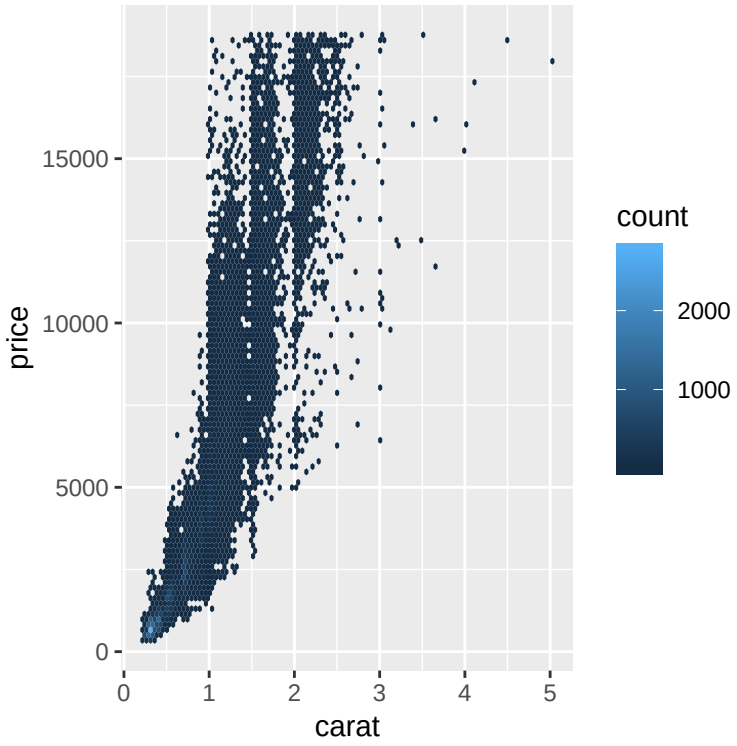

```
ggplot(diamonds, aes(x = carat, y = price)) +
  geom_bin2d(bins = 100)
```



```
# hex binning
ggplot(diamonds, aes(x = carat, y = price)) +
  geom_hex()
```




```
ggplot(diamonds, aes(x = carat, y = price)) +
  geom_hex(bins = 100)
```

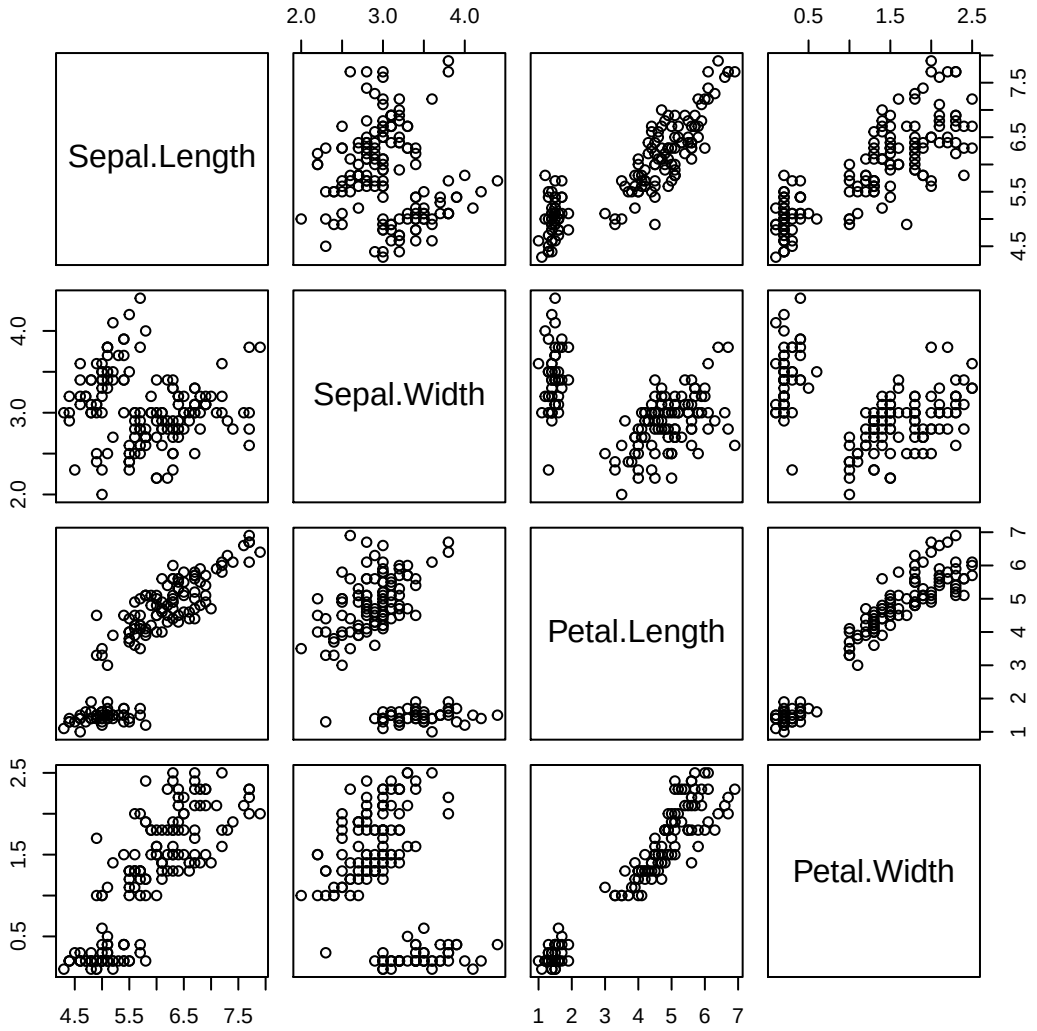


Many variables

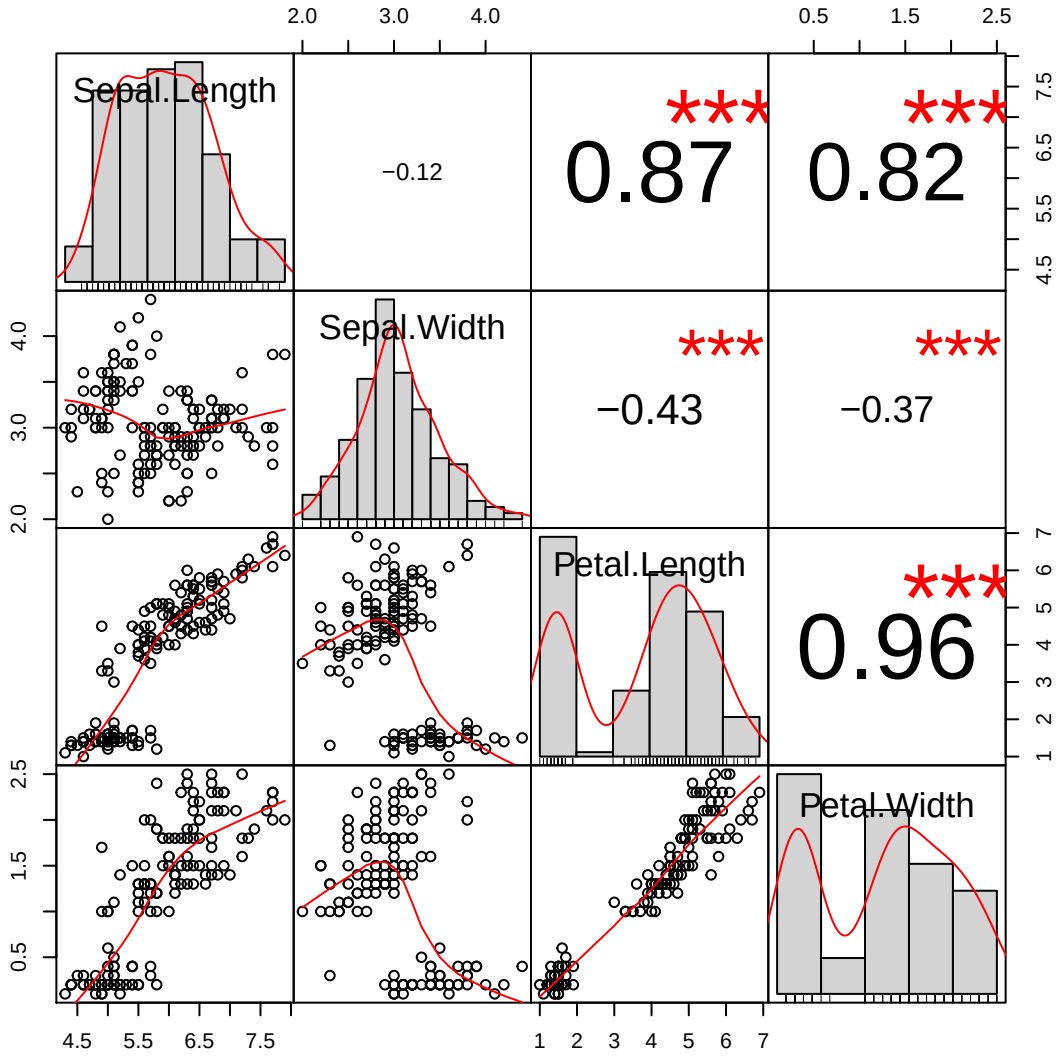
The second type of large data is when we have many variables referred to as multi-variate or high-dimensional data. Before, we begin plotting there are many steps we can take in dealing with the understanding of relationships between the variables like data reduction methods such as principal component analysis (PCA). We do consider data visualization methods for PCA results later on in our section on diagnostic plots. We've already seen examples of how to deal with many dimensions using `facets`, which allow us to treat levels within a factor variable as components in rows or columns of plots. What happens when we have many levels? Facetting then becomes cumbersome and computationally ineffective. We'll see a nice solution to this problem when we deal we discuss animations in the next chapter.

Here, we'll take a look at two special plot types that are particularly useful. The first type of plot is called *SPLOM*, which stands for *Scatter Plot Matrix*. Here, I have made a SPLOM using the four continuous variables in the `iris` dataset using the base package `pairs` function. There are many variations of this theme such as this correlation matrix in the `PerformanceAnalytics` package or this method in the `GGally` package, which uses the `mtcars` data set. This function can handle different variable types not just continuous data. Another popular plot type for dealing with many variables is the parallel coordinate plot, which we encountered in the second course. This example takes the four continuous variables in the `iris` data set and places them on parallel vertical axes. Typically, it will be completely taboo to draw lines between individual values in different nominal variables, but in the case as an exception to the rule, we do want to compare many different variables including categorical and continuous variables together. Variables that are on completely different axes can be lumped into one large visualization. The goal here is to look for trends in how particular variables are related. This should give you an idea of the variety of options we have available to visualize large data sets.

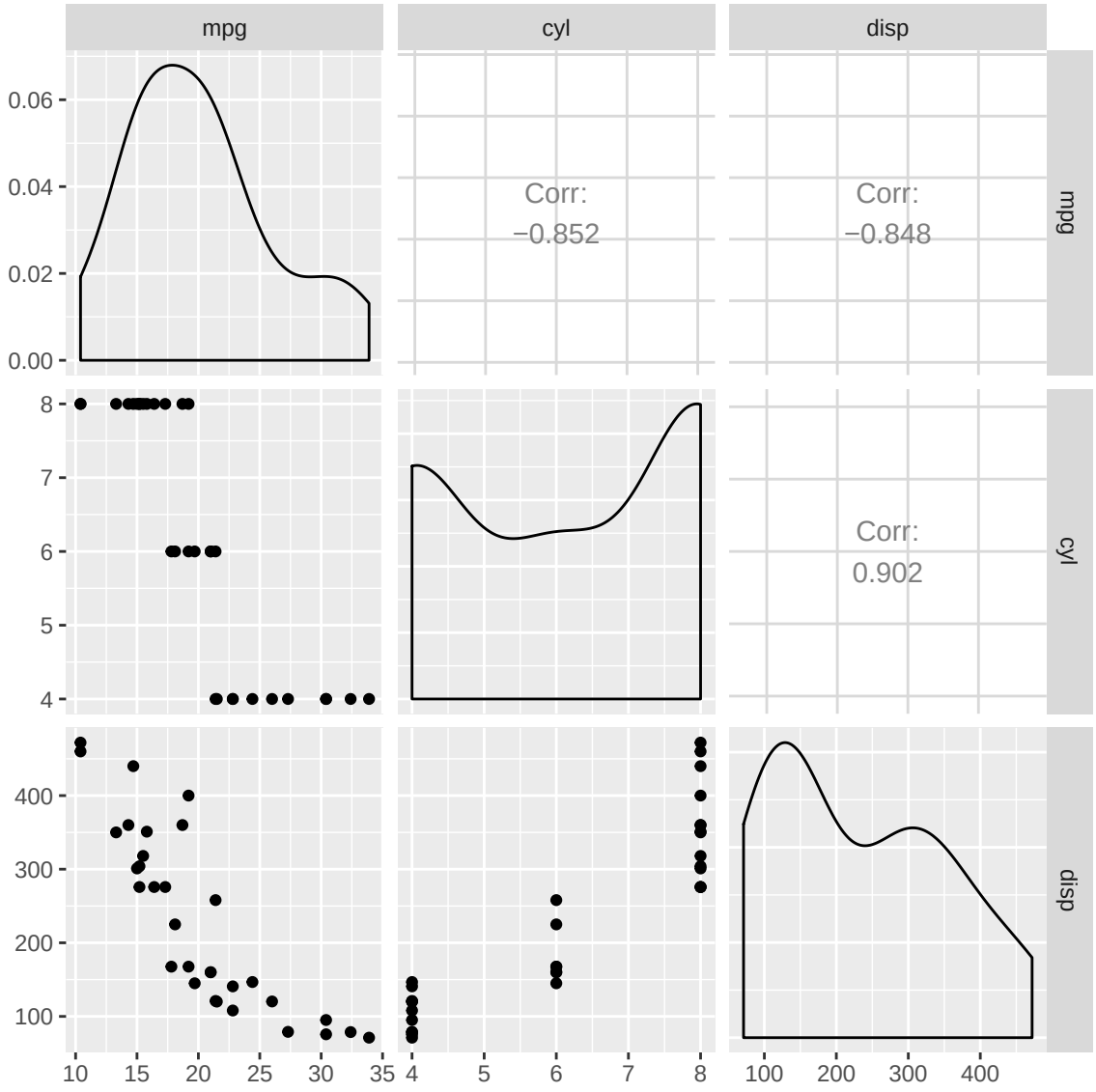
```
pairs(iris[-5])
```



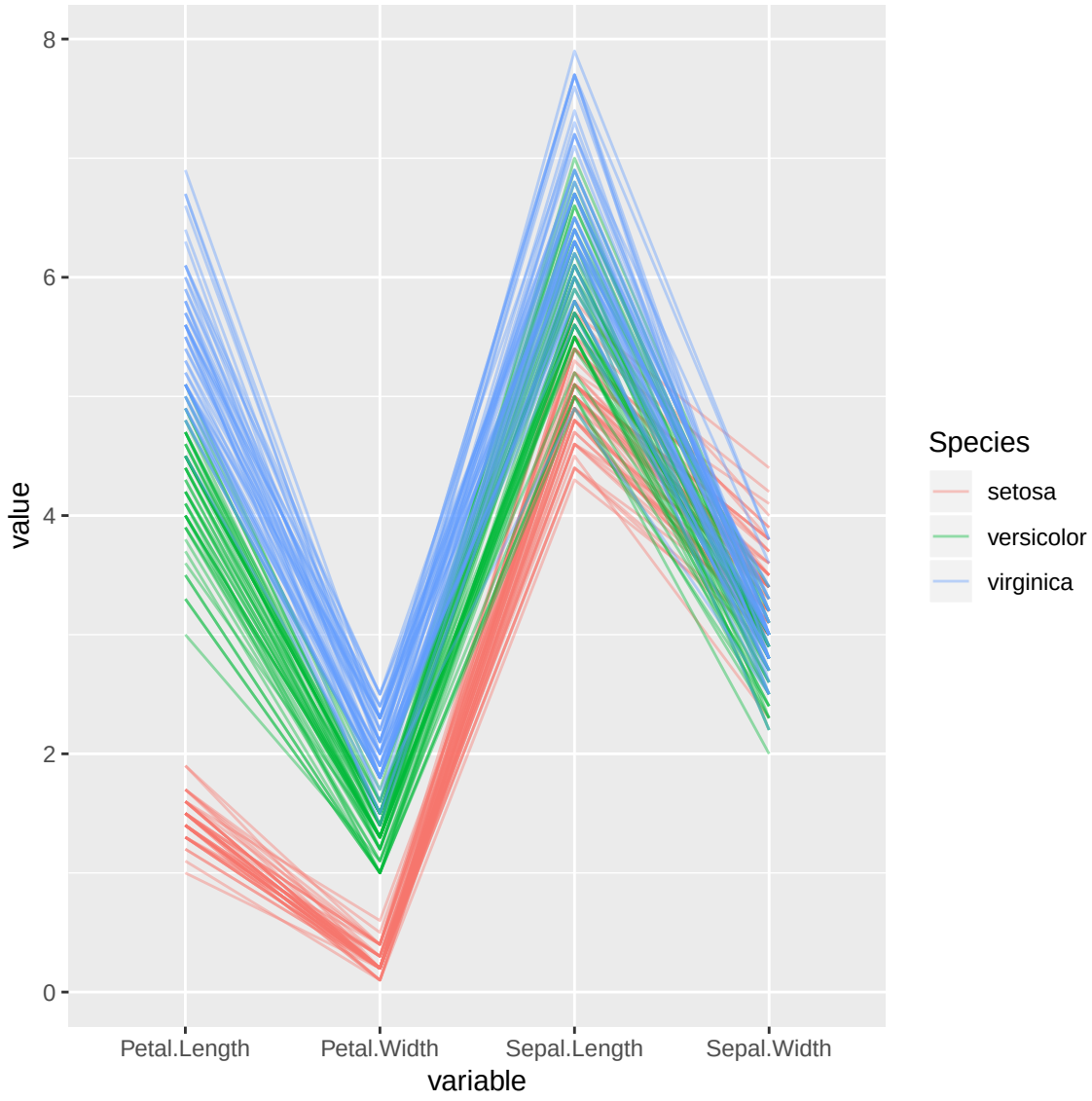

```
library(PerformanceAnalytics)
chart.Correlation(iris[-5],
                  pch = 0:18,
                  cex.labels = 4,
                  labels = c("Sepal Lenght", "Sepal Width", "Petal Length", "Petal Width"))
```




```
library(GGally)
ggpairs(mtcars[1:3])
```




```
ggparcoord(iris, columns = 1:4,
           groupColumn = 5,
           scale = "globalminmax",
           order = "anyClass", alphaLines = 0.4)
```



SPLOM! These are some great-looking *S*catter *PLO*t *M*atrices.

Create a Correlation Matrix in ggplot2

Instead of using an off-the-shelf correlation matrix function, you can of course create your own plot. Just for fun, in this exercise, you'll re-create the scatterplot you see on the right. The strength of the correlation is depicted by the size and color of the points and labels.

For starters, a correlation matrix can be calculated using, for example, `cor(dataframe)` (if all variables are numerical). Before you can use your data frame to create your own correlation matrix plot, you'll need to get it in the right format.

In the editor, you can see the definition of `cor_list()`, a function that re-formats the data frame `x`. Here, `L` is used to add the points to the lower triangle of the matrix, and `M` is used to add the numerical values as text to the upper triangle of the matrix. With `reshape2::melt()`, the correlation matrices `L` and `M` are each converted into a three-column data frame: the x and y axes of the correlation matrix make up the first two columns and the corresponding correlation coefficient makes up the third column. These become the new variables `"points"` and `"labels"`, which can be mapped onto the `size` aesthetic for the points in the lower

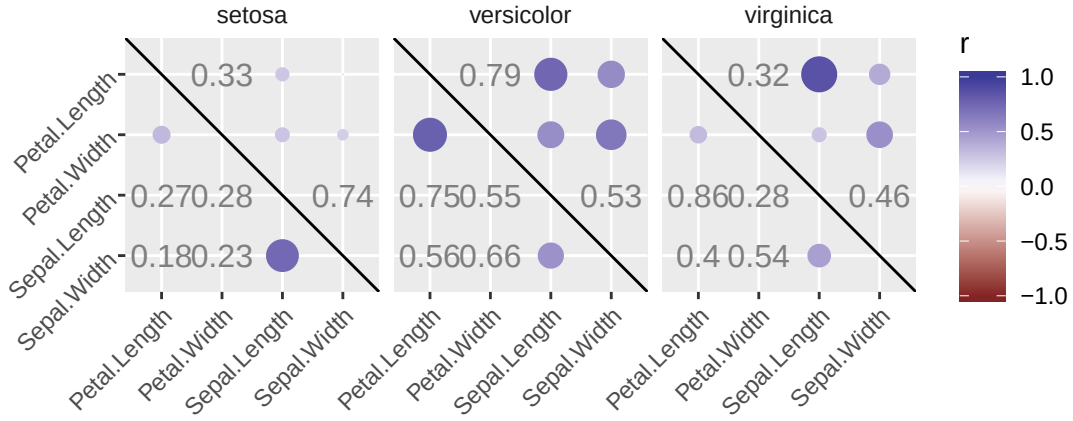
triangle and onto the `label` aesthetic for the text in the upper triangle, respectively. Their values will be the same, but their positions on the plot will be symmetrical about the diagonal! Merging `L` and `M`, you have everything you need.

If you're not familiar with `reshape2` - don't worry, the only reason we use that instead of `tidyr` is that `reshape2::melt()` can handle a matrix, whereas `tidyr::gather()` requires a data frame. At this point you just need to understand how to use the output from `cor_list()`.

You'll first use `dplyr` to execute this function on the continuous variables in the `iris` data frame (the first four columns), but separately for each species. Please refer to the course on dplyr if you are not familiar with these functions.

Next, you'll actually plot the resulting data frame with `ggplot2` functions.

```
library(reshape)

cor_list <- function(x) {
  L <- M <- cor(x)

  M[lower.tri(M, diag = TRUE)] <- NA
  M <- melt(M)
  names(M)[3] <- "points"

  L[upper.tri(L, diag = TRUE)] <- NA
  L <- melt(L)
  names(L)[3] <- "labels"

  merge(M, L)
}

# Calculate xx with cor_list
library(dplyr)
xx <- iris %>%
  group_by(Species) %>%
  do(cor_list(.[1:4])) %>%
  select( "Var1" = X1, "Var2" = X2, everything())

# Finish the plot
# We use abs() to get the absolute value here since correlations can be positive or negative,
# but size can only be positive. Don't set
ggplot(xx, aes(x = Var1, y = Var2)) +
  geom_point(aes(col = points, size = abs(points)), shape = 16) +
  geom_text(aes(col = points,  size = abs(points), label = round(labels, 2))) +
  scale_size(range = c(0, 6)) +
  scale_color_gradient2("r", limits = c(-1, 1)) +
  scale_y_discrete("", limits = rev(levels(xx$Var1))) +
  scale_x_discrete("") +
  guides(size = FALSE) +
  geom_abline(slope = -1, intercept = nlevels(xx$Var1) + 1) +
  coord_fixed() +
  facet_grid(. ~ Species) +
  theme(axis.text.y = element_text(angle = 45, hjust = 1),
        axis.text.x = element_text(angle = 45, hjust = 1),
        strip.background = element_blank())
#> Warning: Removed 30 rows containing missing values (geom_point).
#> Warning: Removed 30 rows containing missing values (geom_text).
```

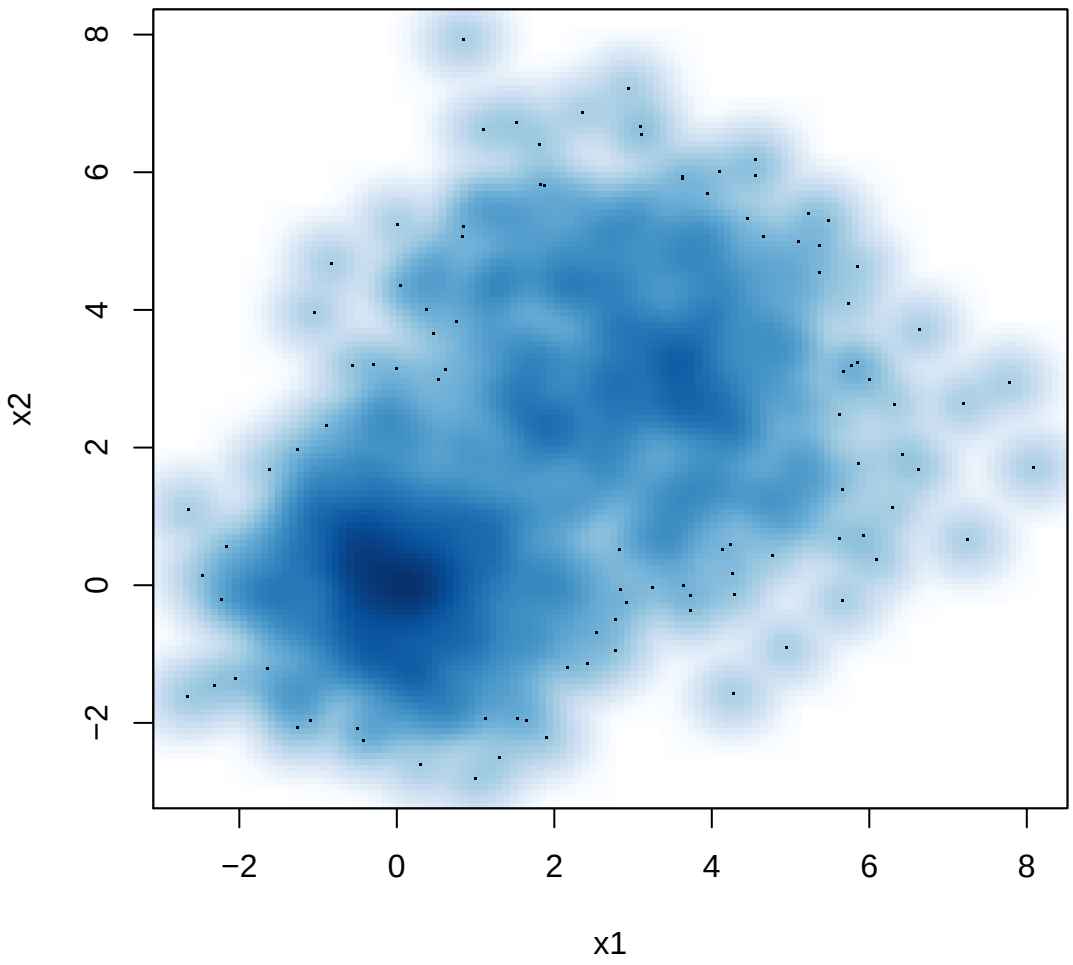


Massive Matrices! Another great custom visualisation!

Ternary Plots

Feel free to use the `knitr` infrastructure with dozens of tunable options in your document.

```
set.seed(123)
n <- 1000
x1  <- matrix(rnorm(n), ncol = 2)
x2  <- matrix(rnorm(n, mean = 3, sd = 1.5), ncol = 2)
x   <- rbind(x1, x2)
head(x)
#>             [,1]        [,2]
#> [1,] -0.56047565 -0.60189285
#> [2,] -0.23017749 -0.99369859
#> [3,]  1.55870831  1.02678506
#> [4,]  0.07050839  0.75106130
#> [5,]  0.12928774 -1.50916654
#> [6,]  1.71506499 -0.09514745
smoothScatter(x, xlab = "x1", ylab = "x2")
```



You can include code snippets of languages other than R, but note that the block header has no curly brackets around the language name.

```
// [[Rcpp::export]]
NumericVector timesTwo(NumericVector x) {
    return x * 2;
}
```

You can also write math expressions, e.g. $Y = X\beta + \epsilon$, footnotes[1], and tables, e.g. using `knitr::kable()`.

[1] A footnote here.

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa

Diagnostic Plots

Please visit the development page of the `prettydoc` package for latest updates and news. Comments, bug reports and pull requests are always welcome.