Chapter 9

Data Visualization

Visualizing data is key in effective data analysis. It is useful for the following purposes:

1. initially investigating datasets,

2. confirming or refuting data models, and

3. elucidating mathematical or algorithmic concepts.

In most of this chapter we explore different types of data graphs using the R programming language which has excellent graphics functionality. We end the chapter with a description of Python's matplotlib module - a popular Python tool for data visualization.

9.1 Graphing Data in R

We focus on two R graphics packages: graphics and ggplot2. The graphics package contains the original R graphics functions and is installed and loaded by default. Its functions are easy to use and produce a variety of useful graphs. The ggplot2 package provides alternative graphics functionality based on Wilkinson's grammar of graphics [31]. To install it and bring it to scope type the following commands.

```
install.packages('ggplot2')
library(ggplot2)
```

When creating complex graphs, the ggplot2 syntax is considerable simpler than the syntax of the graphics package. A potential disadvantage of ggplot2 package is that rendering graphics using ggplot2 may be substantially slower.

9.2 Datasets

We use three datasets to explore data graphs. The `faithful` dataframe is a part of the datasets package that is installed and loaded by default. It has has two variables: eruption time and waiting time to next eruption (both in minutes) of the Old Faithful geyser in Yellowstone National Park, Wyoming, USA. The code below displays the variable names and the corresponding summary statistics.

```
names(faithful)  # variable names
## [1] "eruptions" "waiting"
summary(faithful)  # variable summary
##    eruptions        waiting
##  Min.   :1.600   Min.   :43.0
##  1st Qu.:2.163   1st Qu.:58.0
##  Median :4.000   Median :76.0
##  Mean   :3.488   Mean   :70.9
##  3rd Qu.:4.454   3rd Qu.:82.0
##  Max.   :5.100   Max.   :96.0
```

The `mtcars` dataframe, which is also included in the `datasets` package, contains information concerning multiple car models extracted from 1974 *Motor Trend* magazine. The variables include model name, weight, horsepower, fuel efficiency, and transmission type.

```
summary(mtcars)
##       mpg             cyl
##  Min.   :10.40   Min.   :4.000
##  1st Qu.:15.43   1st Qu.:4.000
##  Median :19.20   Median :6.000
##  Mean   :20.09   Mean   :6.188
##  3rd Qu.:22.80   3rd Qu.:8.000
##  Max.   :33.90   Max.   :8.000
##       disp             hp
##  Min.   : 71.1   Min.   : 52.0
##  1st Qu.:120.8   1st Qu.: 96.5
##  Median :196.3   Median :123.0
##  Mean   :230.7   Mean   :146.7
##  3rd Qu.:326.0   3rd Qu.:180.0
##  Max.   :472.0   Max.   :335.0
##       drat             wt
##  Min.   :2.760   Min.   :1.513
##  1st Qu.:3.080   1st Qu.:2.581
##  Median :3.695   Median :3.325
##  Mean   :3.597   Mean   :3.217
##  3rd Qu.:3.920   3rd Qu.:3.610
##  Max.   :4.930   Max.   :5.424
##       qsec             vs
##  Min.   :14.50   Min.   :0.0000
##  1st Qu.:16.89   1st Qu.:0.0000
```

```
##  Median :17.71   Median :0.0000
##  Mean   :17.85   Mean   :0.4375
##  3rd Qu.:18.90   3rd Qu.:1.0000
##  Max.   :22.90   Max.   :1.0000
##        am              gear
##  Min.   :0.0000   Min.   :3.000
##  1st Qu.:0.0000   1st Qu.:3.000
##  Median :0.0000   Median :4.000
##  Mean   :0.4062   Mean   :3.688
##  3rd Qu.:1.0000   3rd Qu.:4.000
##  Max.   :1.0000   Max.   :5.000
##       carb
##  Min.   :1.000
##  1st Qu.:2.000
##  Median :2.000
##  Mean   :2.812
##  3rd Qu.:4.000
##  Max.   :8.000
```

The `mpg` dataframe is a part of the `ggplot2` package and it is similar to `mtcars` in that it contains fuel economy and other attributes, but it is larger and it contains newer car models extracted from the website http://fueleconomy.gov.

```
names(mpg)
##  [1] "manufacturer" "model"
##  [3] "displ"        "year"
##  [5] "cyl"          "trans"
##  [7] "drv"          "cty"
##  [9] "hwy"          "fl"
## [11] "class"
```

More information on any of these datasets may be obtained by typing `help(X)` with `X` corresponding to the dataframe name when the appropriate package is in scope.

9.3 Graphics and ggplot2 Packages

The `graphics` package contains two types of functions: high-level functions and low-level functions. High level functions produce a graph, while low level functions modify an existing graph. The primary high level function, `plot`, takes as arguments one or more dataframe columns representing data and other arguments that modify its default behavior (some examples appear below).

Other high-level functions in the `graphics` package are more specialized and produce a specific type of graph, such as `hist` for producing histograms, or `curve` for producing curves. We do not explore many high level functions as they are generally less convenient to use than the corresponding functions in the ggplot2 package.

Examples of low-level functions in the `graphics` package are:

- `title` adds or modifies labels of title and axes,
- `grid` adds a grid to the current figure,
- `legend` displays a legend connecting symbols, colors, and line-types to descriptive strings, and
- `lines` adds a line plot to an existing graph.

The two main functions in the ggplot2 package are `qplot` and `ggplot`. The `qplot` function accepts as arguments one or more data variables assigned to the variables `x`, `y`, and `z` (in some cases only one or two of these arguments are specified). The more complex function `ggplot` accepts as arguments a dataframe and an object returned by the `aes` function which accepts data variables as arguments.

In contrast to `qplot`, `ggplot` does not create a graph and returns instead an object that may be modified by adding layers to it using the + operator. After appropriate layers are added the object may be saved to disk or printed using the `print` function. The layer addition functionality applies to `qplot` as well.

The ggplot2 package provides automatic axes labeling and legends. To take advantage of this feature the data must reside in a dataframe with informative column names. We emphasize this approach as it provides more informative dataframes column names, in addition to simplifying the R code.

For example, the following code displays a scatter plot of the columns of a hypothetical dataframe `dataframe` containing two variables `col1` and `col2` using the `graphics` package and then adds a title to the graph.

```
plot(x = dataframe$col_1, y = dataframe$col_2)
title(main = "figure title")  # add title
```

To create a similar graph using the `qplot` function use the following code.

```
qplot(x = x1,
      y = x2,
      data = DF,
      main = "figure title",
      geom = "point")
```

The corresponding `ggplot` code appears below.

```
ggplot(dataframe, aes(x = col1, y = col2)) + geom_point()
```

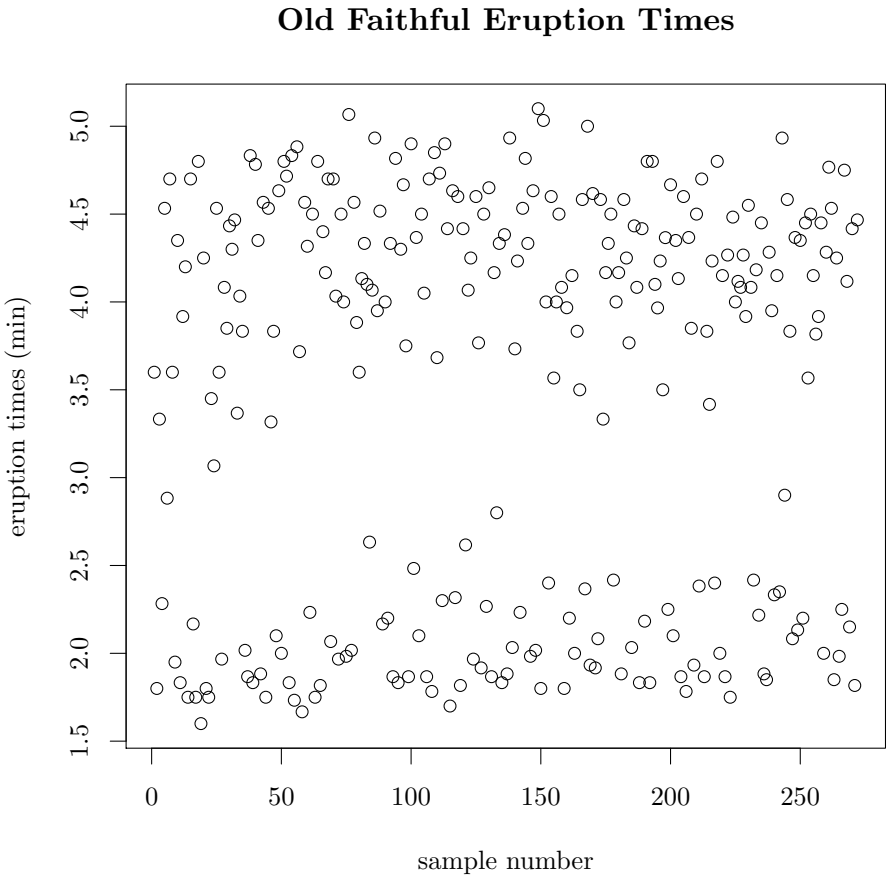
In the following sections we describe several different types of data graphs.

9.4 Strip Plots

The simplest way to graph one-dimensional numeric data is to graph them as points in a two-dimensional space, with one coordinate corresponding to the index of the data point, and the other coordinate corresponding to its value.

To plot a strip plot using the graphics package, call `plot` with a single numerical dataframe column. The resulting x-axis indicates the row number, and the y-axis indicates the numeric value. The `xlab`, `ylab`, and `main` parameters modify the x-label title, y-label title, and figure title.

```
plot(faithful$eruptions,
     xlab = "sample number",
     ylab = "eruption times (min)",
     main = "Old Faithful Eruption Times")
```



We conclude from the figure above that Old Faithful has two typical eruption times — a long eruption time around 4.5 minutes, and a short eruption time

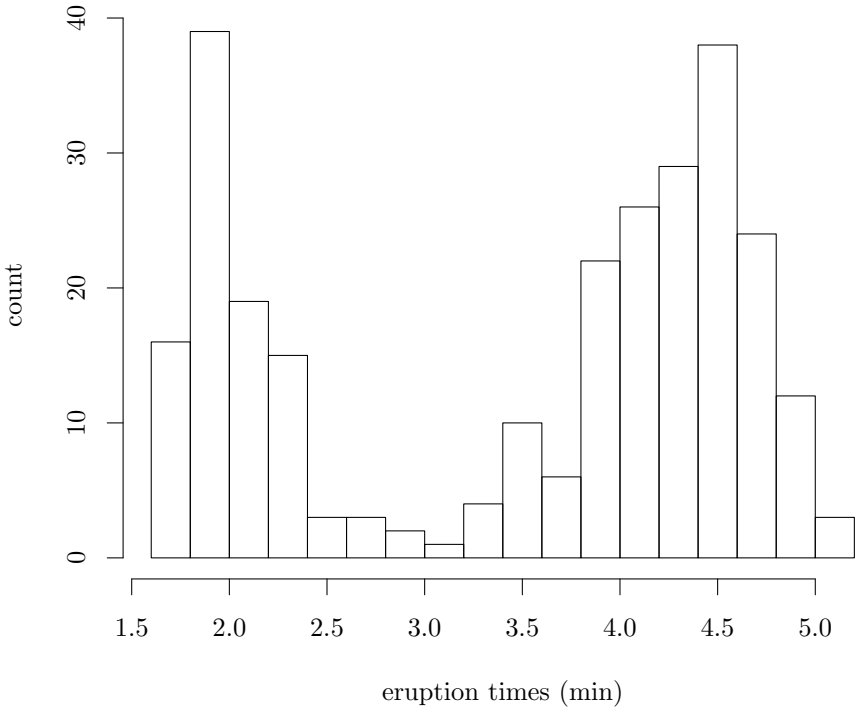
around 1.5 minutes. It also appears that the order in which the dataframe rows are stored is not related to the eruption variable.

9.5 Histograms

An alternative way to graph one dimensional numeric data is using the histogram graph. The histogram divides the range of numeric values into bins and displays the number of data values falling within each bin. The width of the bins influences the level of detail. Very narrow bins maintain all the information present in the data but are hard to draw conclusions from, as the histogram becomes equivalent to a sorted list of data values. Very wide bins lose information due to overly aggressive smoothing. A good bin width balances information loss with useful data aggregation. Unlike strip plots that contain all of the information present in the data, histograms discard the ordering of the data points, and treat samples in the same bin as identical.

The `hist(data, breaks = num_bins)` function within the graphics package can be used to display a histogram. The `xlab`, `ylab`, and `main` parameters described in Section 9.4 can be added as optional parameters to `hist`. See `help(hist)` for more details on the different parameters, and in particular for assistance on controlling the bin widths. For example, the code below displays a histogram of eruption times of Old Faithful using 20 bins.

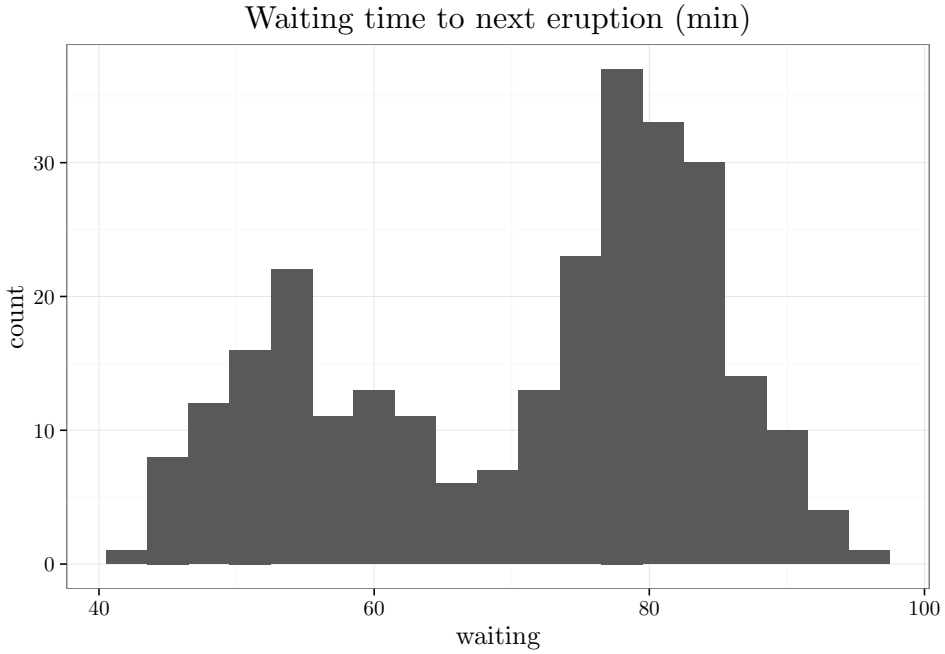
```
hist(faithful$eruptions,
     breaks = 20,
     xlab = "eruption times (min)",
     ylab = "count",
     main = "")
```



We see a nice correspondence between the above histogram and the strip plot in Section 9.4. There are clearly two typical eruption times – one around 2 minutes and one around 4.5 minutes.

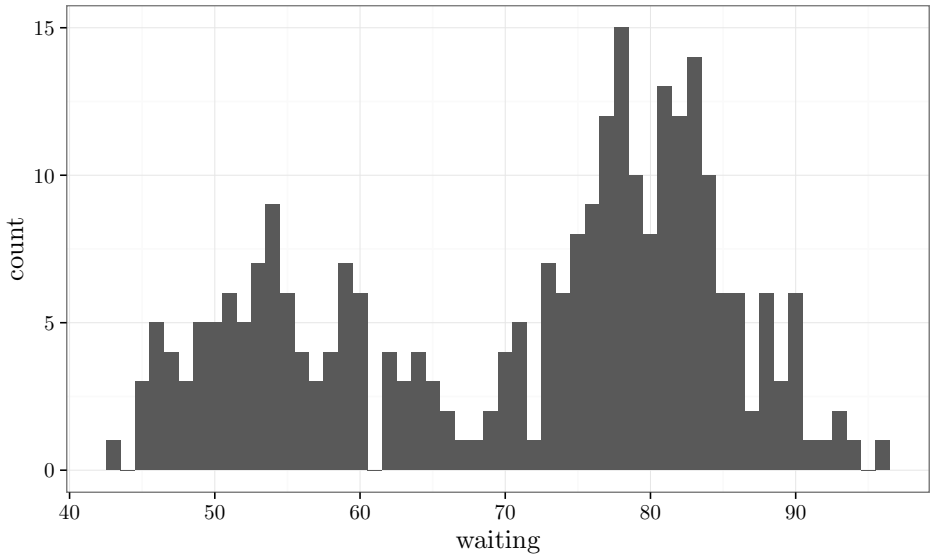
To graph a histogram with the ggplot2 package, call `qplot` with two parameters: a dataframe column (assigned to the `x` argument) and a name of the dataframe variable (assigned to the `data` argument). The two axes are automatically labeled based on the names of the variables that they represent. For example, the code below displays a histogram of the waiting time variable using `qplot`.

```
qplot(x = waiting,
      data = faithful,
      binwidth = 3,
      main = "Waiting time to next eruption (min)")
```



To create a histogram with the `ggplot` function, we pass an object returned from the `aes` function, and add a histogram geometry layer using the + operator.

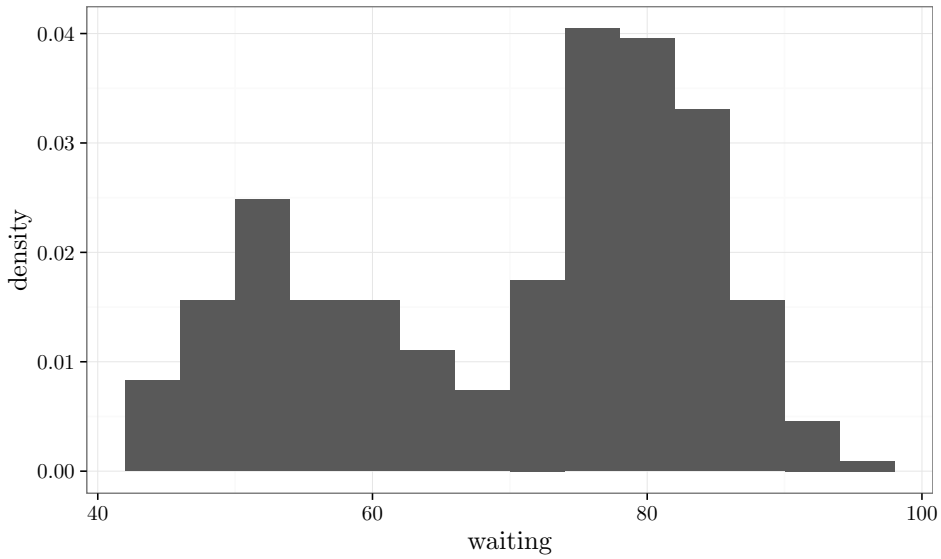
```
ggplot(faithful ,aes(x = waiting)) +
  geom_histogram(binwidth = 1)
```



Traditionally, the y axis in a histogram displays counts. An alternative is to display the frequency by surrounding the data variable with `..` on both sides. In

this case, the height of each bin times its width equals the count of samples falling in the bin divided by the total number of counts. As a result, the area under the graph is 1 making it a legitimate probability density function (see TAOD volume 1, Chapter 2 for a description of the probability density function). This probability interpretation is sometimes advantageous, but it may be problematic in that it masks the total number of counts.

```
ggplot(faithful, aes(x = waiting, y = ..density..)) +
    geom_histogram(binwidth = 4)
```



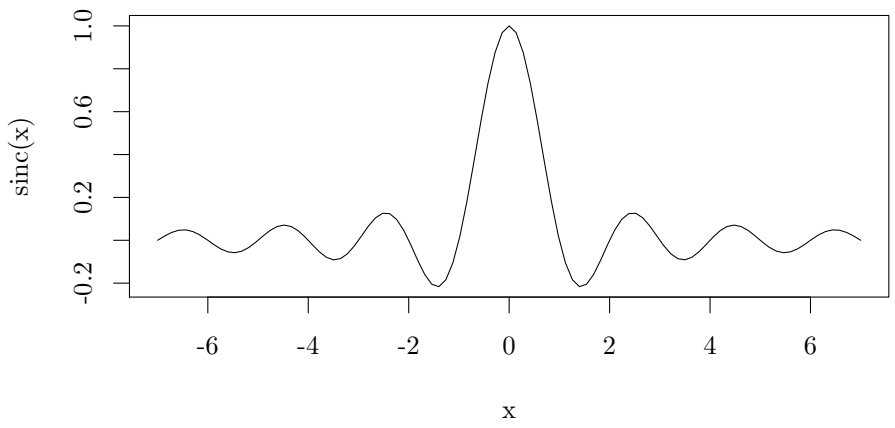
Note that selecting a wider bandwidth (as in the figure above) produces a smoother histogram as compared to the figure before that. Selecting the best bandwidth to use when graphing a specific dataset is difficult and usually requires some trial and error.

9.6 Line Plots

A line plot is a graph displaying a relation between x and y as a line in a Cartesian coordinate system. The relation may correspond to an abstract mathematical function or to relation present between two variables in a specific dataset.

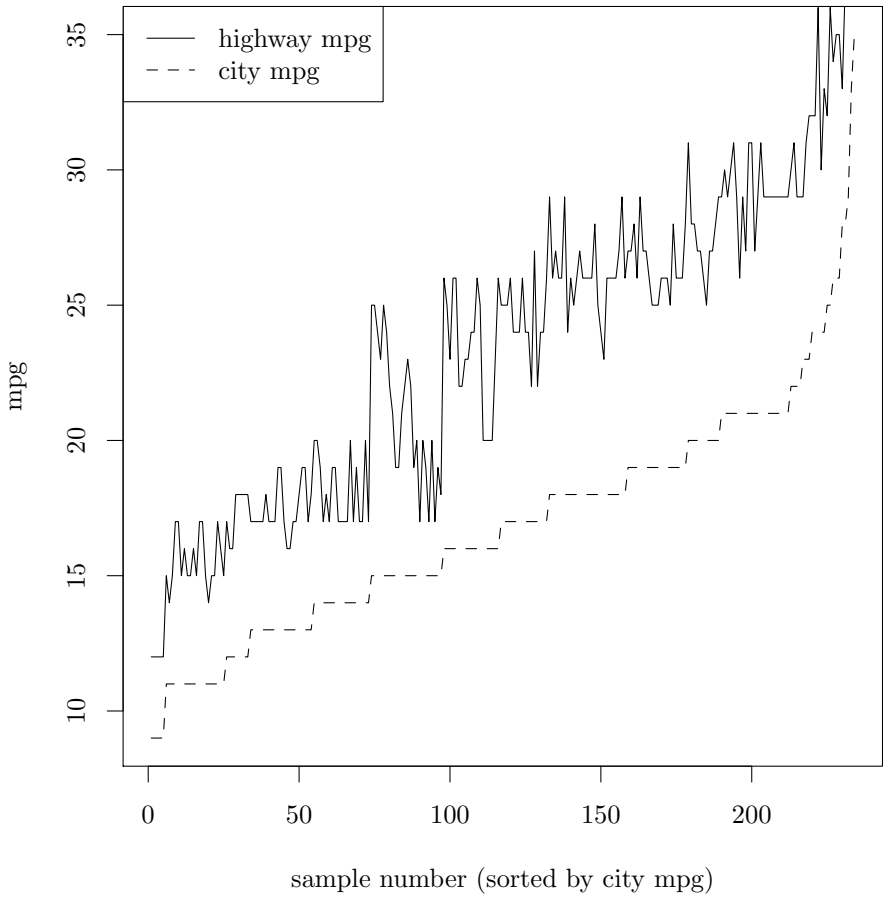
The function `curve` in the graphics package displays mathematical functions. In the example below, the first line defines a new function called `sinc` while the second line plots it. Note the automatic labeling of the axes.

```
sinc = function(x) {
  return(sin(pi * x) / (pi * x))
}
curve(sinc, -7, +7)
```



Another option to display a line plot with the graphics package is to use `plot` but with a `type="l"` parameter, as below. The variable `lty` allows us to display different line types (dashed, dotted, etc.). We demonstrate this below by plotting hwy mpg and city mpg as line plots, where the samples are sorted by city mpg.

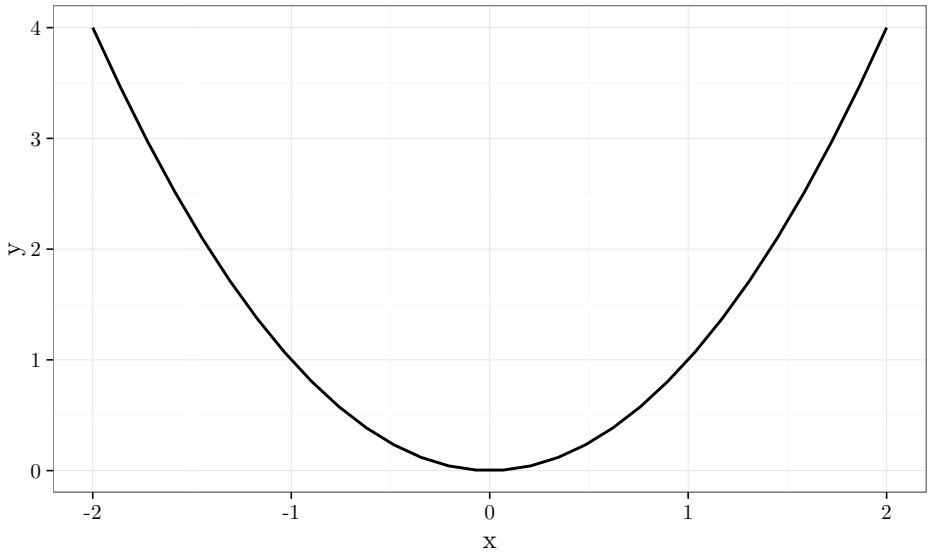
```
S = sort.int(mpg$cty, index.return = T)
#  S$x holds the sorted values of city mpg
#  S$ix holds the indices of the sorted values of city mpg
#  First plot the sorted city mpg values with a line plot
plot(S$x,
     type = "l",
     lty = 2,
     xlab = "sample number (sorted by city mpg)",
     ylab = "mpg")
# add dashed line of hwy mpg
lines(mpg$hwy[S$ix] ,lty = 1)
legend("topleft", c("highway mpg", "city mpg"), lty = c(1, 2))
```



We can conclude from the plot above that (i) highway mpg tends to be higher than city mpg, (ii) highway mpg tends to increase as city mpg increases, and (iii) the difference between the two quantities is less significant for cars with lower fuel efficiency.

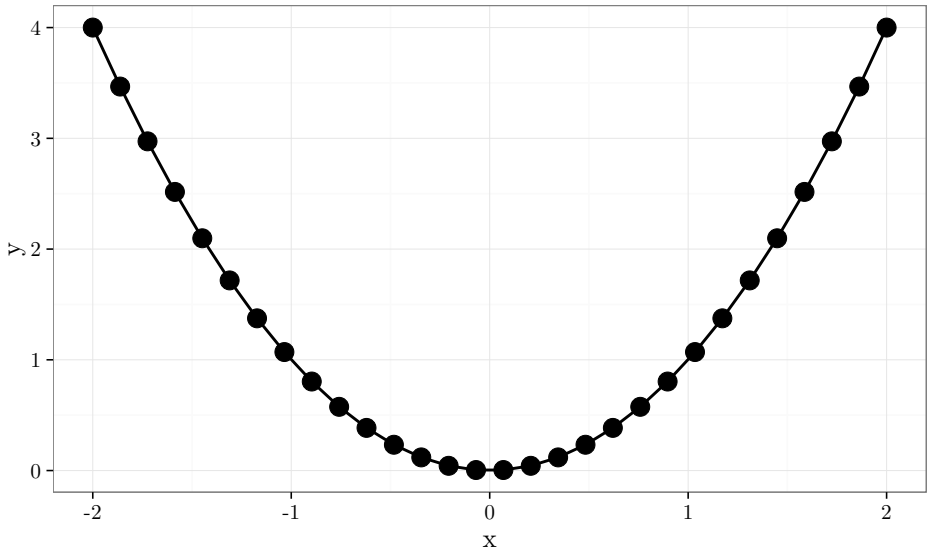
The `qplot` function creates a line plot when passed a `geom=line` parameter.

```
x = seq(-2, 2, length.out = 30)
y = x^2
qplot(x, y, geom = "line")
```



Below is a similar example where multiple geometries are present.

```
x = seq(-2, 2, length.out = 30)
y = x^2
qplot(x, y, geom = c("point", "line"))
```



The function `ggplot` creates the same plot by adding a line geometry layer `geom_line()` using the + operator.

```
# new data frame with variables x,y = x^2
dataframe = data.frame(x = x, y = y)
ggplot(dataframe, aes(x = x, y = y)) + geom_line() + geom_point()
```

9.7 Smoothed Histogram

An alternative to the histogram is the smoothed histogram. Denoting n data points by $x^{(1)}, \ldots, x^{(n)}$, the smoothed histogram is the following function $f_h : \mathbb{R} \to \mathbb{R}_+$

$$f_h(x) = \frac{1}{n} \sum_{i=1}^{n} K_h(x - x^{(i)})$$

where the kernel function $K_h : \mathbb{R} \to \mathbb{R}$ typically achieves its maximum at 0, and decreases as $|x - x^{(i)}|$ increases. We also assume that the kernel function integrates to one $\int K_h(x)\, dx = 1$ and satisfies the relation

$$K_h(r) = h^{-1} K_1(r/h).$$

We refer to K_1 as the base form of the kernel and denote it as K.

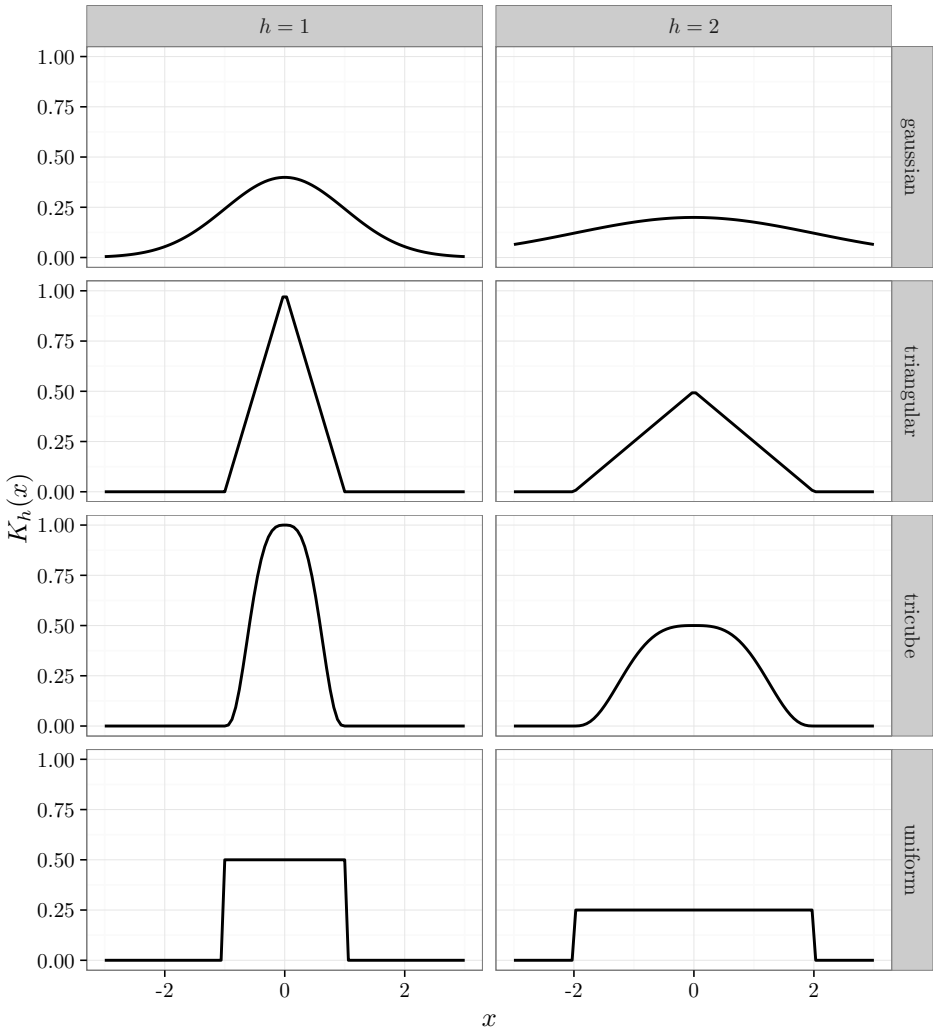
Four popular kernel choices are the tricube, triangular, uniform, and Gaussian kernels, defined as $K_h(r) = h^{-1}K(r/h)$ where the $K(\cdot)$ functions are respectively

$$\begin{aligned}
K(r) &= (1 - |r|^3)^3 \cdot 1_{\{|r|<1\}} && \text{(Tricube)} \\
K(r) &= (1 - |r|) \cdot 1_{\{|r|<1\}} && \text{(Triangular)} \\
K(r) &= 2^{-1} \cdot 1_{\{|r|<1\}} && \text{(Uniform)} \\
K(r) &= \exp(-x^2/2)/\sqrt{2\pi} && \text{(Gaussian).}
\end{aligned}$$

The following R code displays these kernels for $h = 1$ and $h = 2$. Note how the kernel corresponding to $h = 2$ is twice as wide as the kernels corresponding to $h = 1$. The technique used to display multiple panels in the same graph is called faceting, and is described in the next section.

```
x_grid = seq(-3, 3, length.out = 100)
K1 = function(x) {
  ind = abs(x) > 1
  x = x * 0 + 1/2
  x[ind] = 0
  return(x)
}
K2 = function(x) {
  ind = abs(x) > 1
  x = 1 - abs(x)
  x[ind] = 0
  return(x)
}
```

```
K3 = function(x) dnorm(x)
K4 = function(x) {
  ind = abs(x) > 1
  x = (1 - abs(x)^3)^3
  x[ind] = 0
  return(x)
}
R = stack(list('uniform' = K1(x_grid),
               'triangular' = K2(x_grid),
               'gaussian' = K3(x_grid),
               'tricube' = K4(x_grid),
               'uniform' = K1(x_grid / 2) / 2,
               'triangular' = K2(x_grid / 2) / 2,
               'gaussian' = K3(x_grid / 2) / 2,
               'tricube' = K4(x_grid / 2) / 2))
head(R)  # first six lines
##   values     ind
## 1      0 uniform
## 2      0 uniform
## 3      0 uniform
## 4      0 uniform
## 5      0 uniform
## 6      0 uniform
names(R) = c('kernel.value', 'kernel.type')
R$x = x_grid
R$h[1:400] = '$h=1$'
R$h[401:800] = '$h=2$'
head(R)  # first six lines
##   kernel.value kernel.type         x     h
## 1            0     uniform -3.000000 $h=1$
## 2            0     uniform -2.939394 $h=1$
## 3            0     uniform -2.878788 $h=1$
## 4            0     uniform -2.818182 $h=1$
## 5            0     uniform -2.757576 $h=1$
## 6            0     uniform -2.696970 $h=1$
qplot(x,
      kernel.value,
      data = R,
      facets = kernel.type~h,
      geom = "line",
      xlab = "$x$",
      ylab = "$K_h(x)$")
```



Above, we use LaTeX code (text strings containing $ symbols that surround equation code in the example above) to annotate the axes labels or titles with equations.

Identifying the functions $g_i(x) \stackrel{\text{def}}{=} K_h(x - x^{(i)})$ we see that f_h is an average of the g_i functions, $i = 1, \ldots, n$. Since the functions $g_i(x)$ are centered at $x^{(i)}$ and decay with the distance between x and $x^{(i)}$, their average f_h will be high in areas containing many data points and low in areas containing a few data points. The role of the denominator n in f_h is to ensure that the integral of f_h is 1, making f_h a formal estimator of the underlying distribution (see TAOD volume 1, chapter 2).

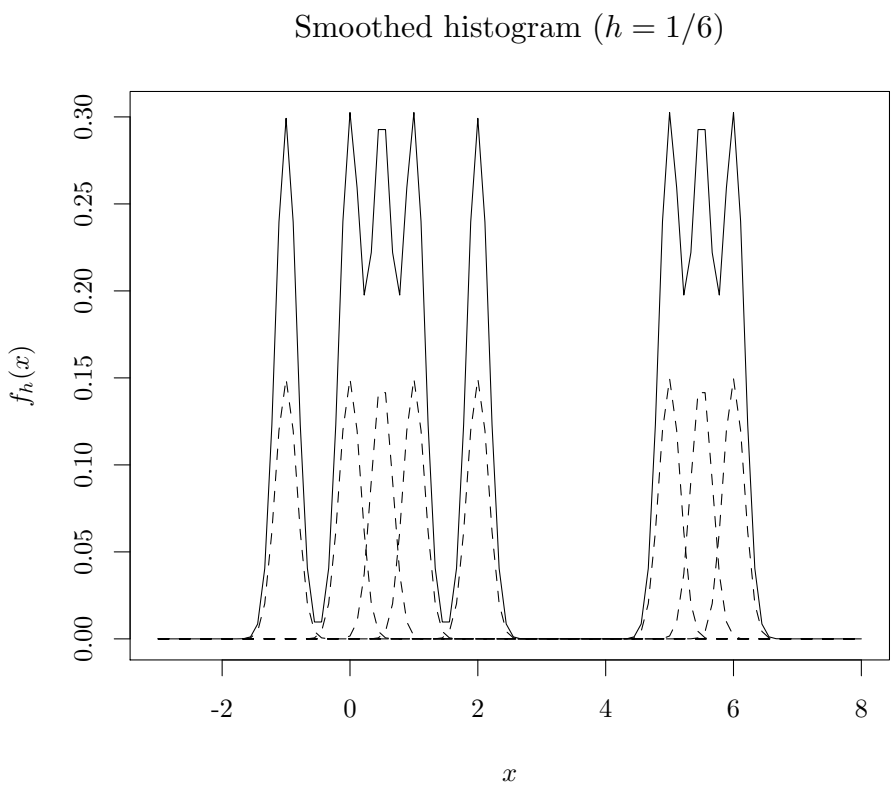
The R code below graphs the smoothed histogram of the data

$$\{-1, 0, 0.5, 1, 2, 5, 5.5, 6\}$$

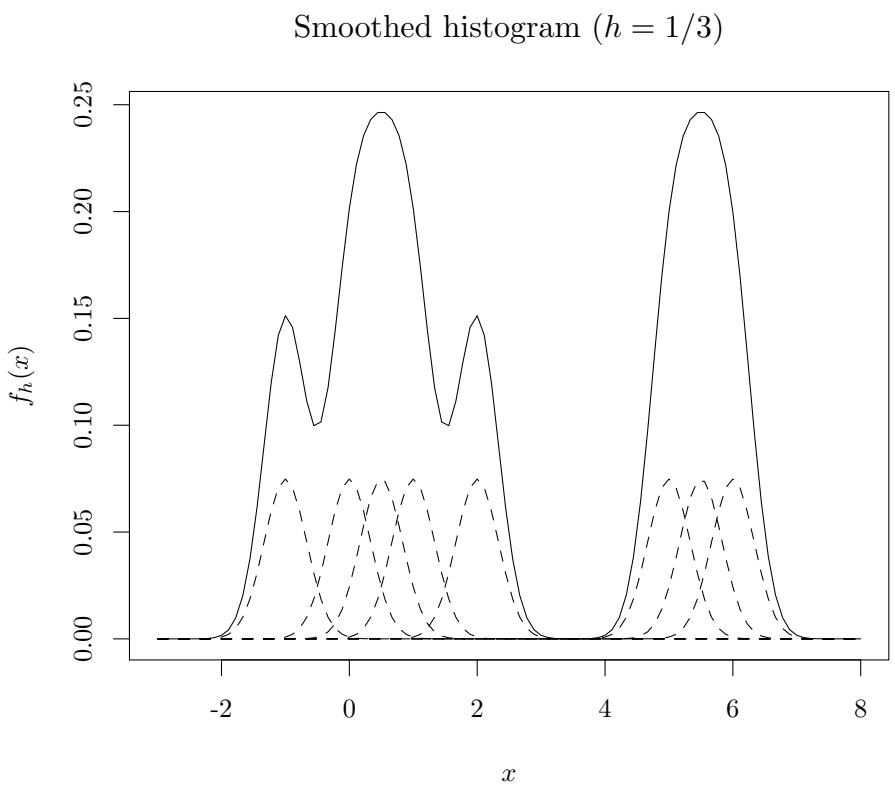
using the Gaussian kernel. The graphs show f_h as a solid line and the g_i functions as dashed lines (scaled down by a factor of 2 to avoid overlapping solid and dashed lines).

In the first graph below, the h value is relatively small ($h = 1/6$), resulting in a f_h close to the a sequence of narrow spikes centered at the data points. In the second graph h is larger ($h = 1/3$) showing a multimodal shape that is significantly different from the first case. In the third case, h is relatively large ($h = 1$), resulting in a f_h that resembles two main components. For larger h, f_h will resemble a single unimodal shape.

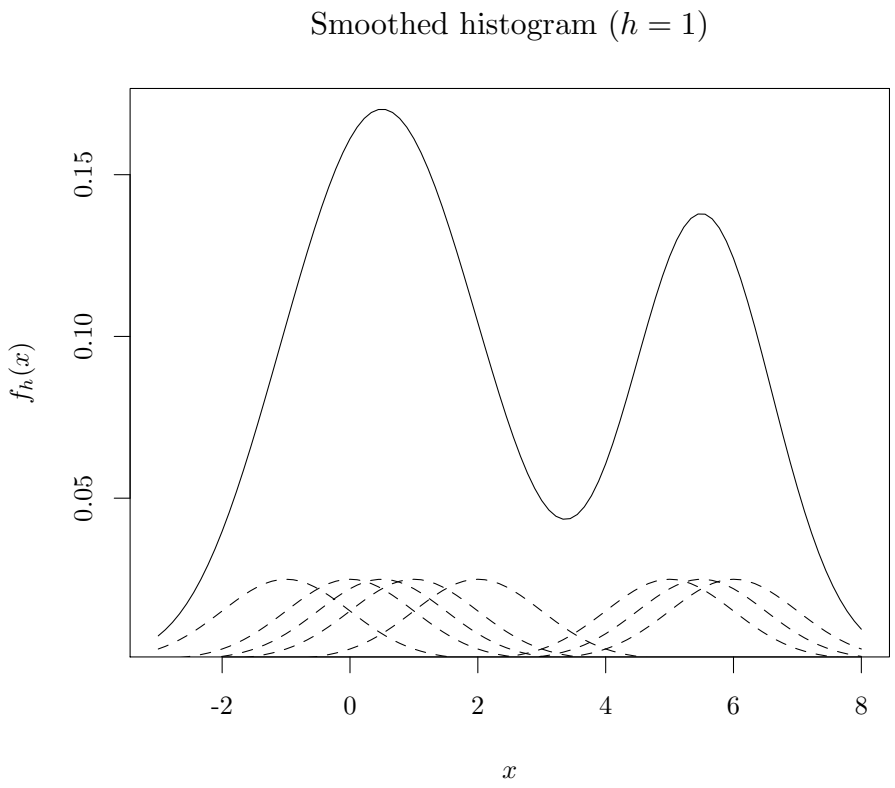
```
data = c(-1, 0, 0.5, 1, 2, 5, 5.5, 6)
data_size = length(data)
x_grid = seq(-3, data_size, length.out = 100)
kernel_values = x_grid %o% rep(1, data_size)
f = x_grid * 0
for(i in 1:data_size) {
  kernel_values[,i] = dnorm(x_grid, data[i], 1/6)/data_size
  f = f + kernel_values[,i]
}
plot(x_grid, f, xlab = "$x$", ylab = "$f_h(x)$", type = "l")
for (i in 1:data_size) lines(x_grid, kernel_values[,i]/2, lty = 2)
title("Smoothed histogram ($h=1/6$)", font.main = 1)
```




```
f = x_grid * 0
for(i in 1:data_size) {
  kernel_values[,i] = dnorm(x_grid, data[i], 1/3)/data_size
  f = f + kernel_values[,i]
}
plot(x_grid, f, xlab = "$x$", ylab = "$f_h(x)$", type = "l")
for (i in 1:data_size) lines(x_grid, kernel_values[,i]/2, lty = 2)
title("Smoothed histogram ($h=1/3$)", font.main = 1)
```




```
f = x_grid * 0
for(i in 1:data_size) {
  kernel_values[,i] = dnorm(x_grid, data[i], 1)/data_size
  f = f + kernel_values[,i]
}
plot(x_grid, f, xlab = "$x$" , ylab = "$f_h(x)$", type = "l")
for (i in 1:data_size) lines(x_grid, kernel_values[,i]/2, lty = 2)
title("Smoothed histogram ($h=1$)", font.main = 1)
```



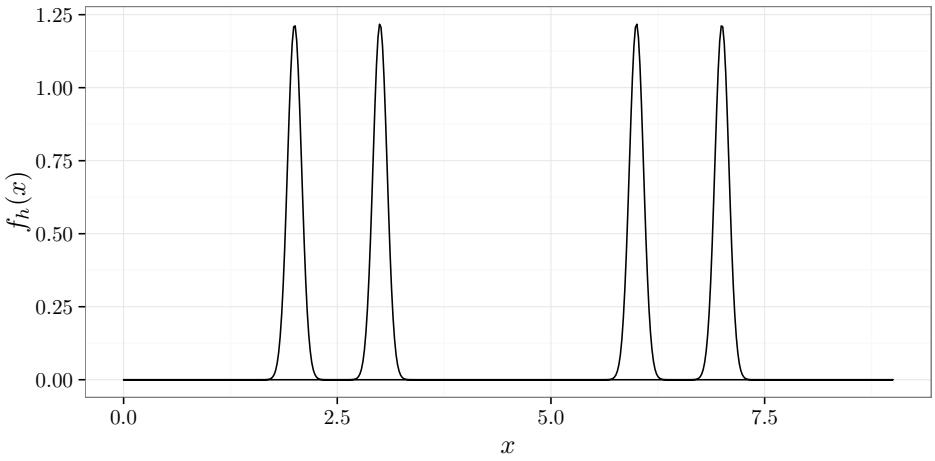
The smoothed histogram is often more intuitive and informative than the non-smoothed histogram. Just as the bin-width selection is crucial for obtaining an informative histogram, selecting the kernel function is very important. Selecting a value h that is too large will have the effect of over-smoothing, causing the resulting $\hat{f}$ to be nearly constant. On the other, hand selecting a value of h that is too small will have an under-smoothing effect, and result in a wiggly and noisy curve.

The ggplot2 package incorporates smoothed histogram graphing into the `qplot` and `ggplot` functions. The value of h is controlled via the `adjust` parameter that assigns h to be the corresponding multiple of an automatic value determined by R. For example, setting `adjust=1` uses R's automatic h value, while setting `adjust=2` multiplies R's automatic h value by 2.

In the first graph below we use a small h value, causing f_h to exhibit four clearly separated peaks — each corresponding to a separate g_i function. This choice of a small h corresponds to a histogram with a very narrow bin-width.

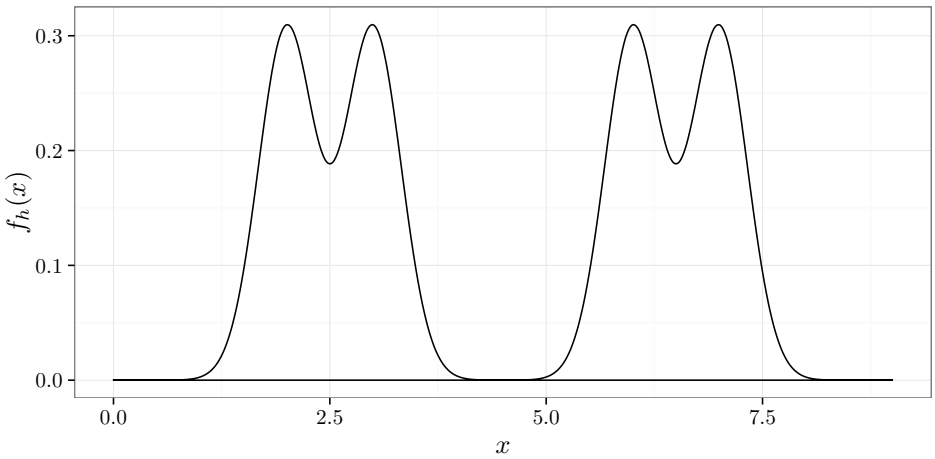
```
qplot(x = c(2, 3, 6, 7),
      y = ..density..,
      geom = c("density"),
```

```
      kernel = "gaussian",
      adjust = 0.05,
      xlab = "$x$",
      ylab = "$f_h(x)$",
      xlim = c(0, 9))
```



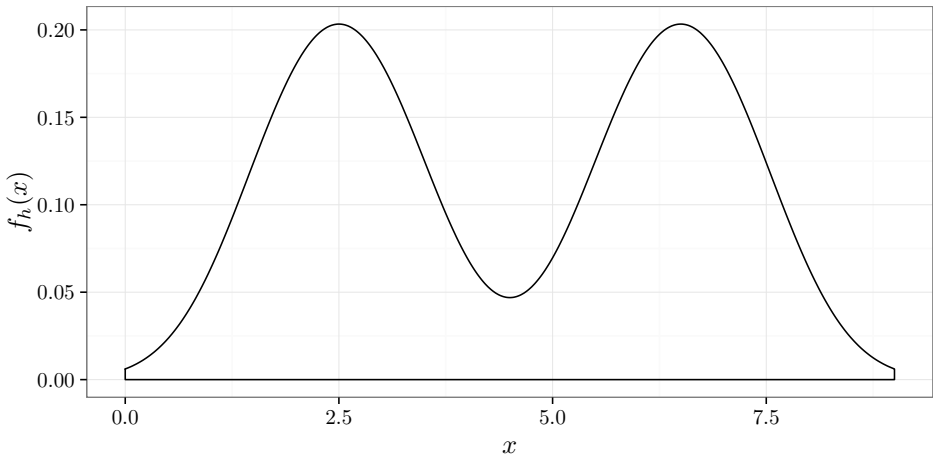
Increasing the value of h by increasing the `adjust` parameter diffuses the g_i functions, causing them to overlap more.

```
qplot(x = c(2,3,6,7),
      y = ..density..,
      geom = c("density"),
      kernel = "gaussian",
      adjust = 0.2,
      xlab = "$x$",
      ylab = "$f_h(x)$",
      xlim = c(0,9))
```



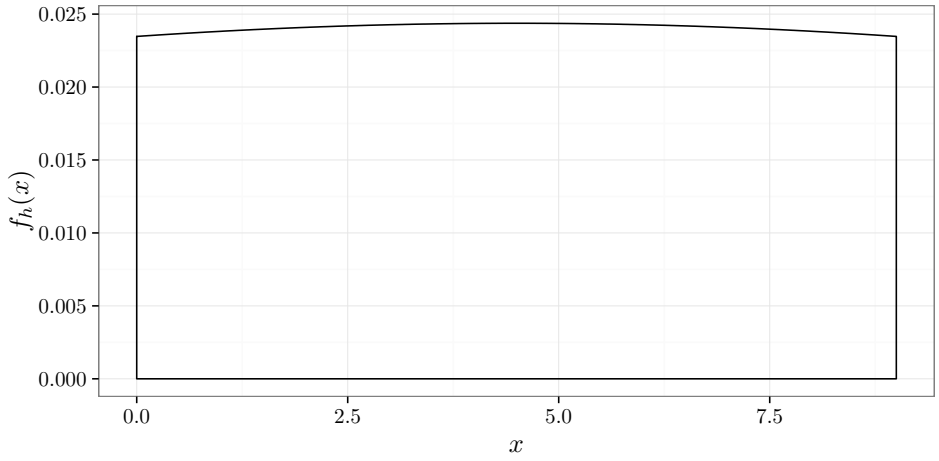
Increasing the value of h further aggregates the four peaks into two peaks, each responsible for a corresponding pair of nearby points.

```
qplot(x = c(2, 3, 6, 7),
      y = ..density..,
      geom = c("density"),
      kernel = "gaussian",
      adjust = 0.5,
      xlab = "$x$",
      ylab = "$f_h(x)$",
      xlim = c(0,9))
```



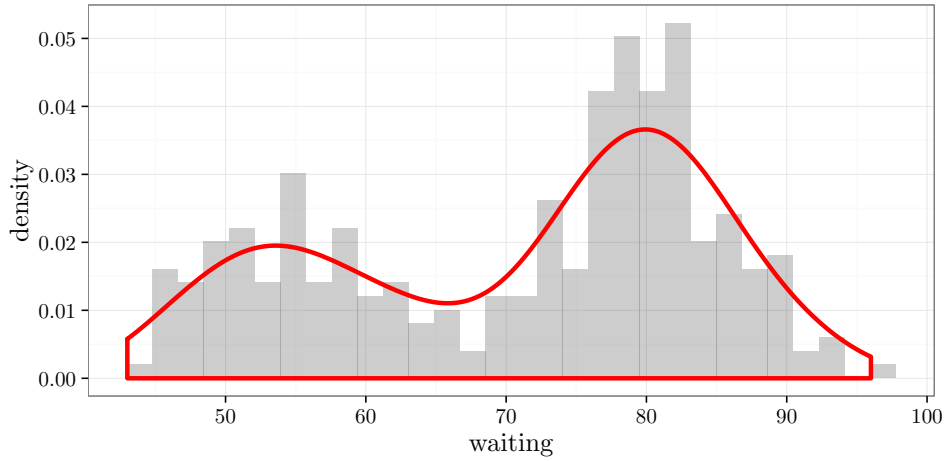
Finally, further increasing the value of h results in nearly constant g_i functions and a nearly constant $\hat{f}$ function. This corresponds to a histogram with very wide bins in which all points fall in the same bin.

```
qplot(x = c(2, 3, 6, 7),
      y = ..density..,
      geom = c("density"),
      kernel = "gaussian",
      adjust = 10,
      xlab = "$x$",
      ylab = "$f_h(x)$",
      xlim = c(0,9))
```



The figure below contrasts a histogram with a smoothed histogram using the `ggplot` function. To enhance the visualization we made the histogram semi-transparent using the `alpha` argument that takes a value between 0 and 1 indicating the transparency level.

```
ggplot(faithful, aes(x = waiting, y = ..density..)) +
  geom_histogram(alpha = 0.3) +
  geom_density(size = 1.5, color = "red")
```



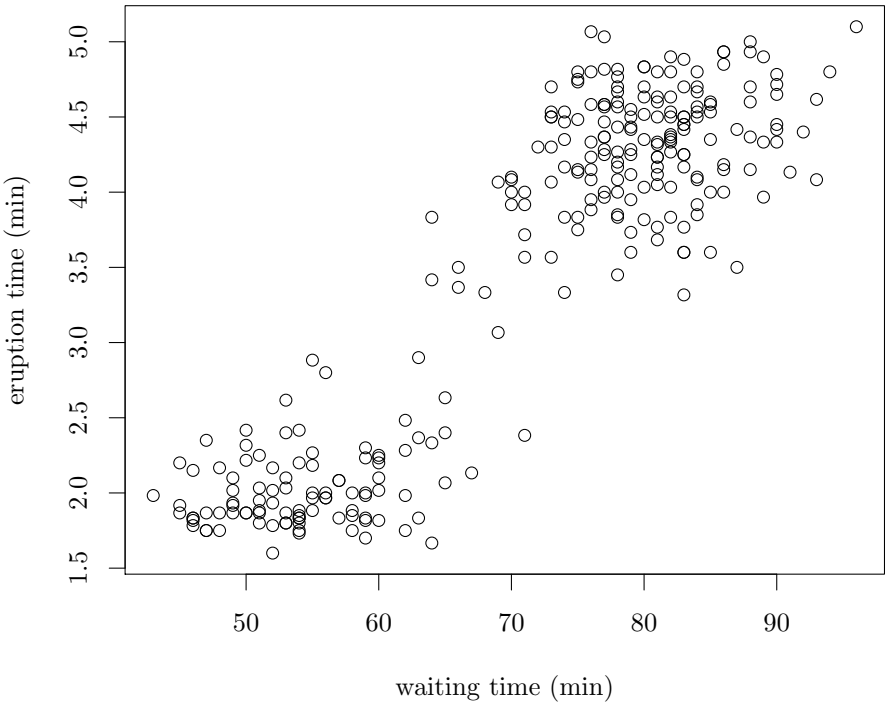
As is apparent from the figure above, smoothed histograms do a better job of uncovering the mathematical relationship between the variable and the frequency of density. This is backed mathematically by the fact that the smoothed histogram is a better estimator for the underlying density than the histogram. On the other hand, histograms features a more direct relationship between the graph and the underlying data that is sometimes important in its own right.

9.8 Scatter Plots

A scatter plot graphs the relationships between two numeric variables. It graphs each pair of variables as a point in a two dimensional space whose coordinates are the corresponding x, y values.

To create a scatter plot with the graphics package call `plot` with two dataframe columns.

```
plot(faithful$waiting,
     faithful$eruptions,
     xlab = "waiting time (min)",
     ylab = "eruption time (min)")
```

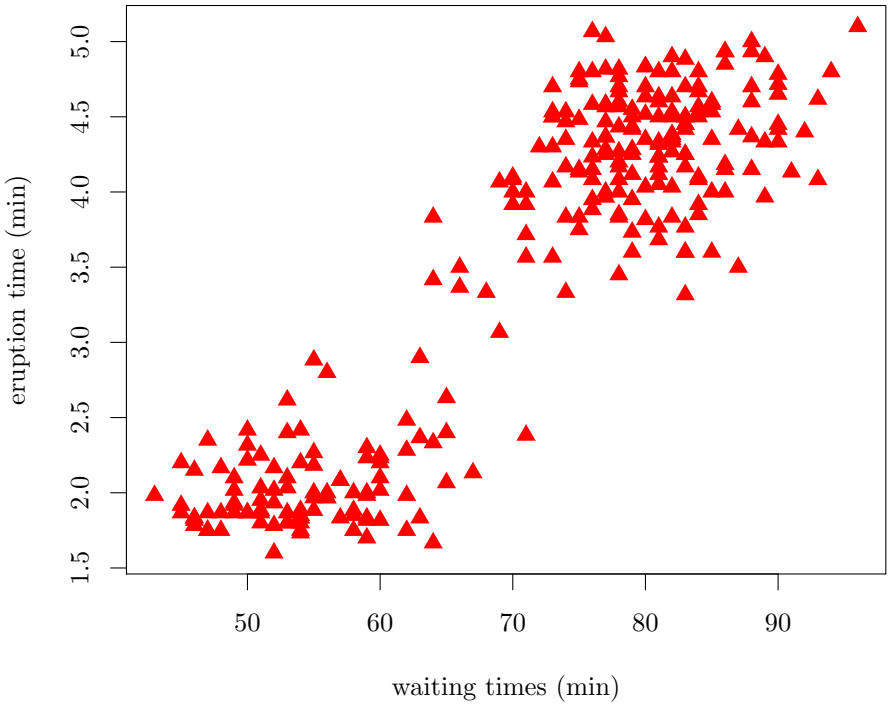


We conclude from the two clusters in the scatter plot above that there are two distinct cases: short eruptions and long eruptions. Furthermore, the waiting times for short eruptions are typically short, while the waiting times for the long eruptions are typically long. This is consistent with our intuition: it takes longer to build the pressure for a long eruption than it does for a short eruption.

The points above are graphed using hollow circular markers. The arguments `pch`, `col`, `cex` modify the marker's shape, color, and size, respectively. Type

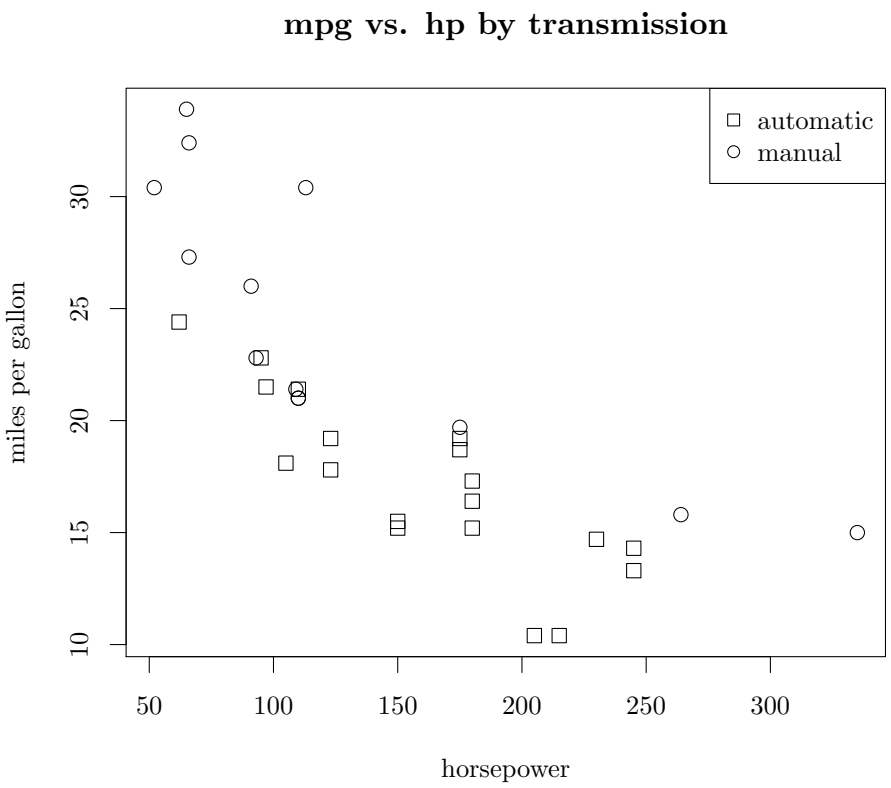
`help(pch)` for more information on setting these values.

```
plot(faithful$waiting,
     faithful$eruptions,
     pch = 17,
     col = 2,
     cex = 1.2,
     xlab = "waiting times (min)",
     ylab = "eruption time (min)")
```



In some cases we wish to plot a scatter plot of one dataframe column vs. another, but distinguish the points based on the value of another dataframe column, typically a factor variable (factors in R take values in an ordered or unordered finite set — see Chapter 6). For example the plot below shows horsepower vs. mile per gallon of cars within the `mtcars` dataset, but distinguishes between automatic and manual transmission using different symbols. Transmission types are encoded through the `am` variable, which takes values 0 or 1 — both legitimate values for the `pch` marker shape argument.

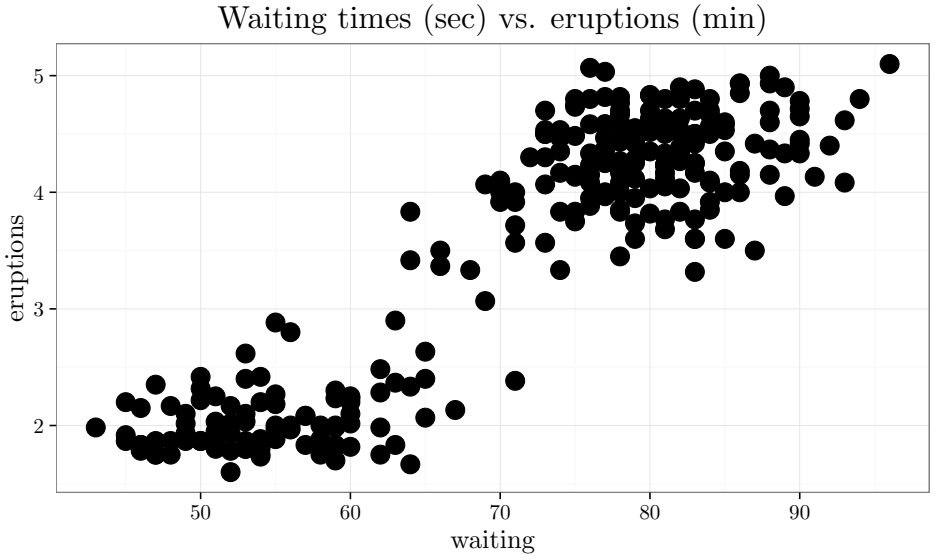
```
plot(mtcars$hp,
     mtcars$mpg,
     pch = mtcars$am,
     xlab = "horsepower",
     cex = 1.2,
     ylab = "miles per gallon",
     main = "mpg vs. hp by transmission")
legend("topright", c("automatic", "manual"), pch = c(0, 1))
```



We draw several conclusions from this graph. First, there is an inverse relationship between horsepower and mpg. Second, for a given horsepower amount, manual transmission cars are generally more fuel efficient. Third, cars with the highest horsepower tend to be manual. Indeed, the two highest horsepower cars in the dataset are Maserati Bora and Ford Pantera, both sports cars with manual transmissions.

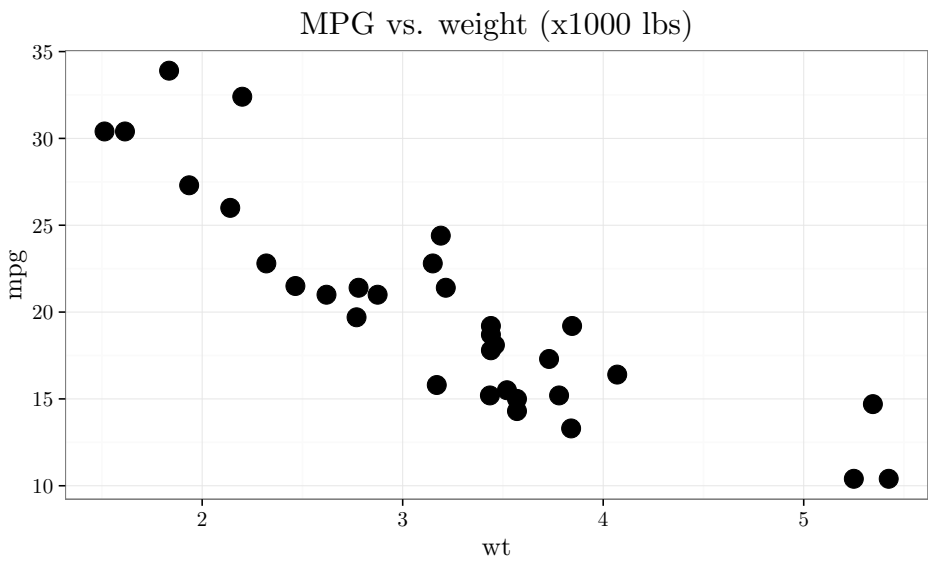
To graph a scatter plot with `qplot` call it with two dataframe column parameters assigned to the `x` and `y` arguments.

```
qplot(x = waiting,
      y = eruptions,
      data = faithful,
      main = "Waiting times (sec) vs. eruptions (min)")
```



The graph below shows a scatter plot of car weights vs mpg.

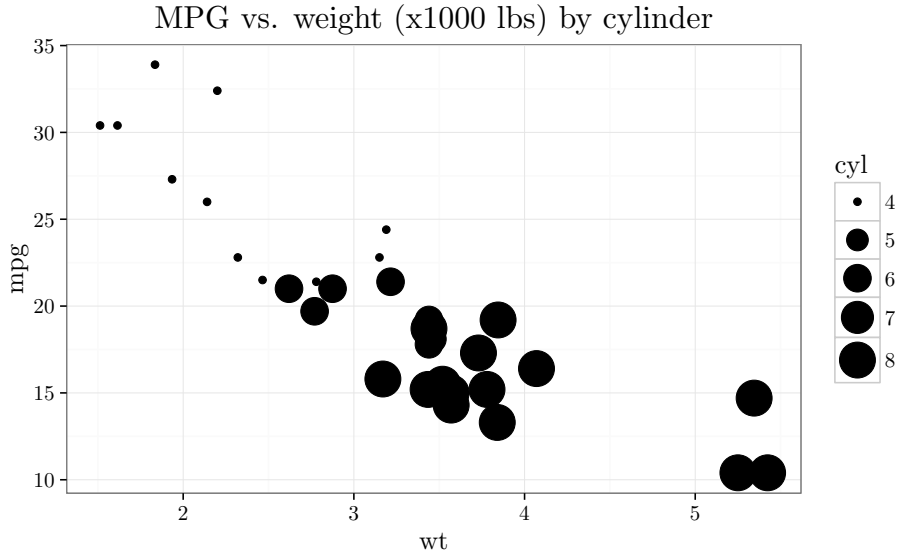
```
qplot(x = wt,
      y = mpg,
      data = mtcars,
      main = "MPG vs. weight (x1000 lbs)")
```



We conclude from the figure above that there exists a somewhat linear trend with negative slope between mpg and weight (though that trend decreases for heavier cars).

Denoting the number of cylinders by size using the `size` argument allows us to investigate the relationship between the three numeric quantities.

```
qplot(x = wt,
      y = mpg,
      data = mtcars,
      size = cyl,
      main = "MPG vs. weight (x1000 lbs) by cylinder")
```



We conclude from the figure that cars with more cylinders tend to have higher weight and lower fuel efficiency. Alternatively, color can be used to encode the number of cylinders using the argument `color`.

```
qplot(x = wt,
      y = mpg,
      data = mtcars,
      color = cyl,
      main = "MPG vs. weight (x1000 lbs) by cylinder")
```

In many cases the data contains a large amount of noise, and graphing it may focus the viewer's attention on the noise while hiding important general trends. One technique to address this issue is to add a smoothed line curve y_S, which is a weighted average of the original data $(y^{(i)}, x^{(i)})$ $i = 1, \ldots, n$:

$$y_S(x) = \sum_{i=1}^{n} \frac{K_h(x - x^{(i)})}{\sum_{i=1}^{n} K_h(x - x^{(i)})} y^{(i)}. \tag{9.1}$$

The K_h functions above are the kernel functions described in Section 9.7.

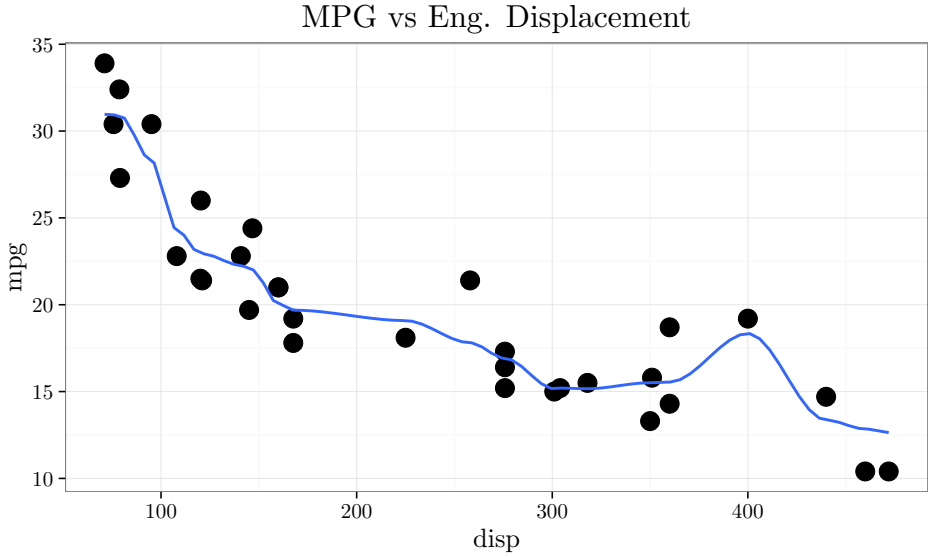
In other words, $y_S(x)$ is an average the $y^{(i)}$ values, weighted in a way that emphasizes $y^{(i)}$ values whose corresponding $x^{(i)}$ values are close to x. The denominator in the definition of y_S ensures that the weights defining the weighted average sum to 1.

Equation (9.1) describes a statistical technique known as locally constant regression that can be used to estimate a relationship between x and y without making parametric assumptions (for example the assumption of a linear relationship).

We demonstrate the smoothed scatter plot with several graphs below. The first two graphs explore different values of h, the parameter that influences the

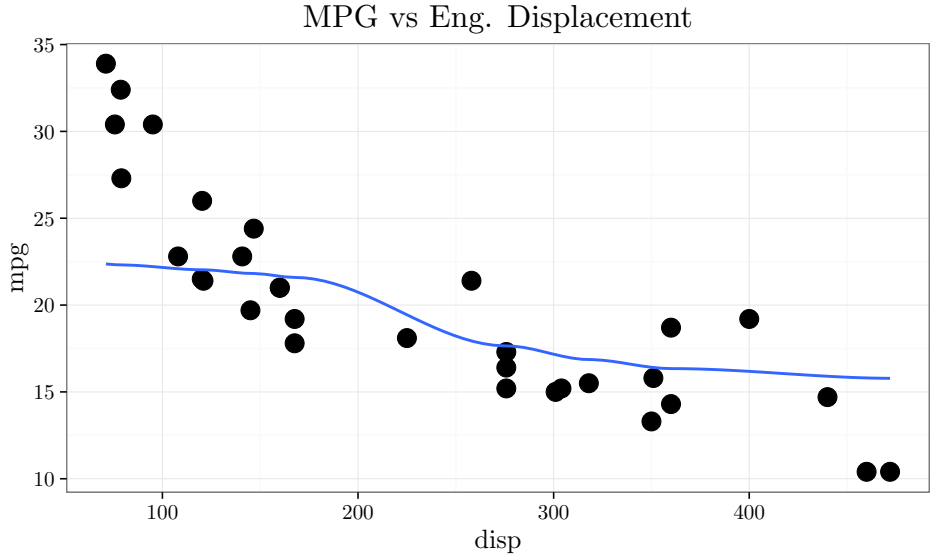
spread or width of the $g_i = K_h(x, x^{(i)})$ functions. To adjust h, we modify the `span` argument that has a similar role to the `adjust` parameter in the discussion of the smoothed histogram above.

```
qplot(disp,
      mpg,
      data = mtcars,
      main = "MPG vs Eng. Displacement") +
  stat_smooth(method = "loess",
              method.args = list(degree = 0),
              span = 0.2,
              se = FALSE)
```



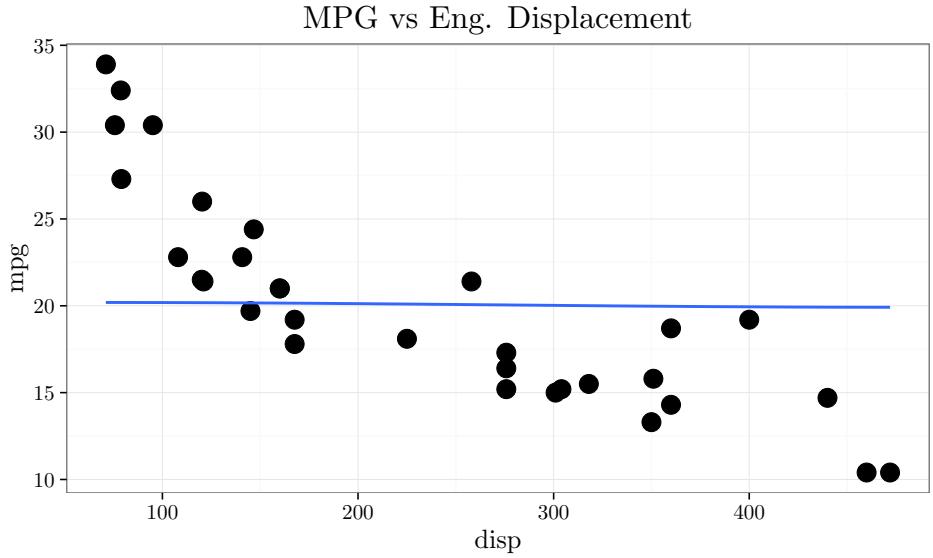
Increasing the value of the `span` parameter increases h, resulting in wider g_i functions and a less wiggly curve.

```
qplot(disp,
      mpg,
      data = mtcars,
      main = "MPG vs Eng. Displacement") +
  stat_smooth(method = "loess",
              method.args = list(degree = 0),
              span = 1,
              se = FALSE)
```



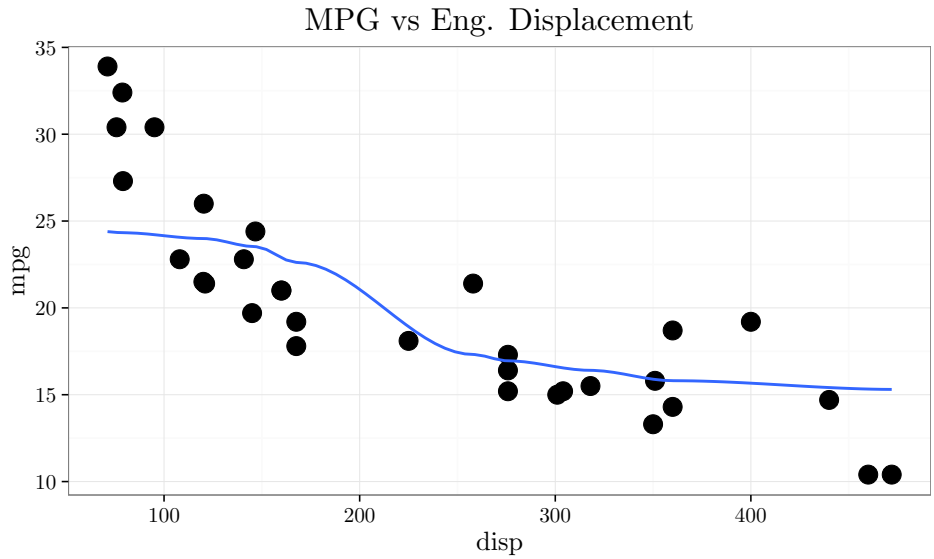
Selecting an even larger h results in a nearly constant line.

```
qplot(disp,
      mpg,
      data = mtcars,
      main = "MPG vs Eng. Displacement") +
  stat_smooth(method = "loess",
              method.args = list(degree = 0),
              span = 10,
              se = FALSE)
```



Omitting this parameter reverts to an automatically chosen value.

```
qplot(disp,
      mpg,
      data = mtcars,
      main = "MPG vs Eng. Displacement") +
  stat_smooth(method = "loess",
              method.args = list(degree = 0),
              se = FALSE)
```

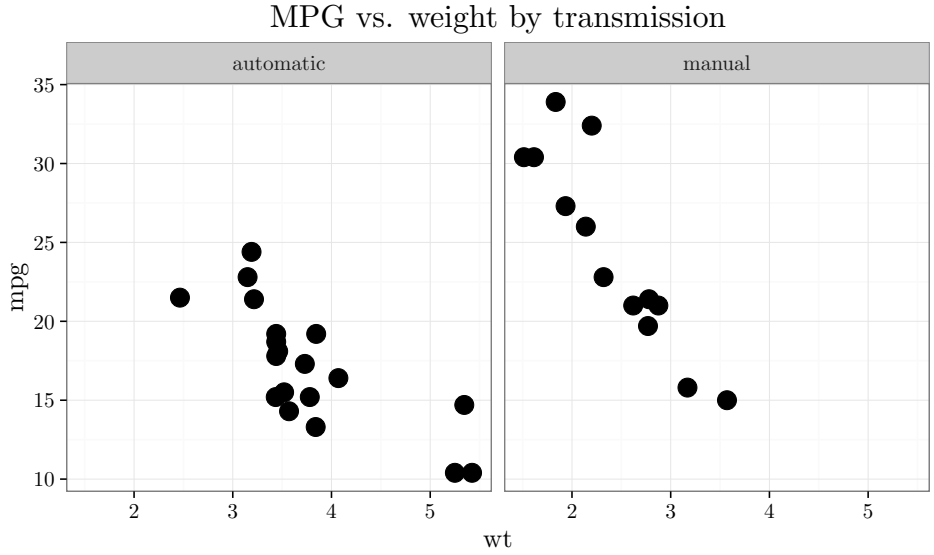


We can conclude from the graph above that mpg decreases as engine displacement volume increases. The trend is nonlinear with a slope that changes as follows: first small slope, then large slope, and then small slope again (in absolute value). This information is readily available from the smoothed y_S but is not easy to discern from the original scatter plot data.

In some cases we want to examine multiple plots with the same x or y axes in different side-by-side panels. The function `qplot` enables this using the `facets` argument which takes a formula of the form $a \sim b$ and creates multiple rows and columns of panels (a determines the row variable and b the column variable).

In the first example below we graph two scatter plots side-by-side: mpg vs. weight for automatic transmission cars and manual transmission cars. Note that the two panels are side-by-side since the `facet` argument is . ~`amf`. The two panels share the same axes' scales, thus facilitating easy comparison. As before, we create new dataframe columns with more appropriate names in order to create more informative axes labeling. Changing the names of existing columns using the function `names` is another option.

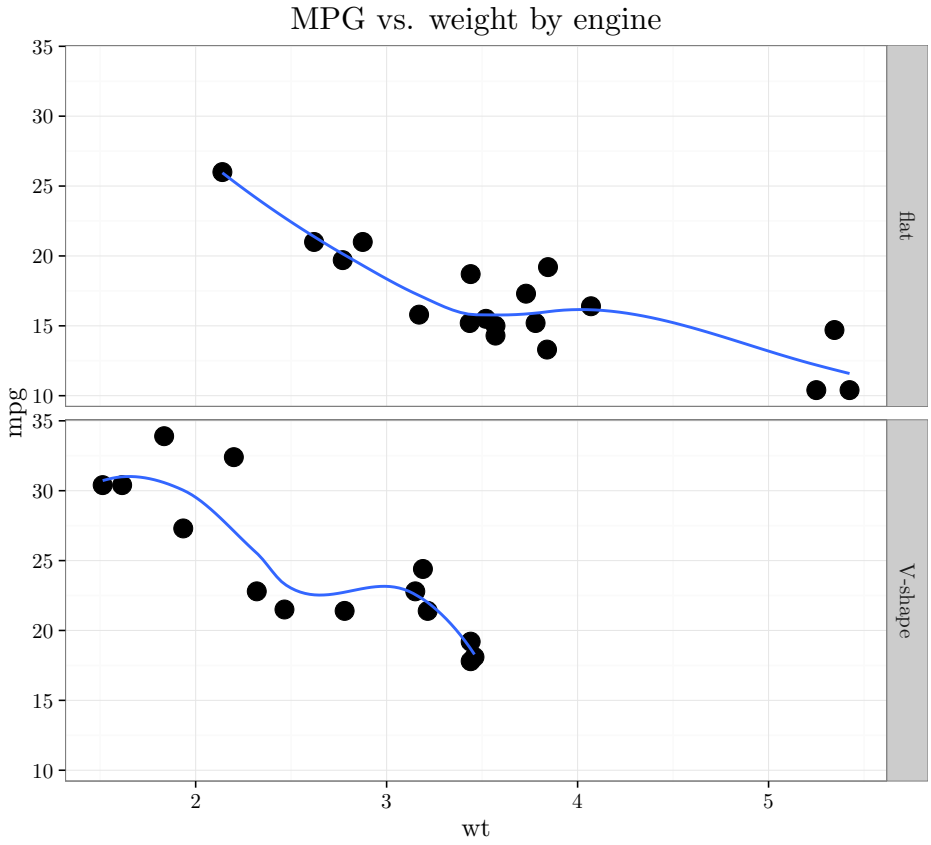
```
# add new dataframe columns with more appropriate names for
# better axes labeling in future graphs
mtcars$amf[mtcars$am==0] = 'automatic'
mtcars$amf[mtcars$am==1] = 'manual'
mtcars$vsf[mtcars$vs==0] = 'flat'
mtcars$vsf[mtcars$vs==1] = 'V-shape'
qplot(x = wt,
      y = mpg,
      facets = .~amf,
      data = mtcars,
      main = "MPG vs. weight by transmission")
```



We conclude from the graph above that manual transmission cars tend to have lower weights and be more fuel efficient.

The graph below plots mpg vs. weight for two panels - one above the other indicating whether or not the engine is a V shape engine.

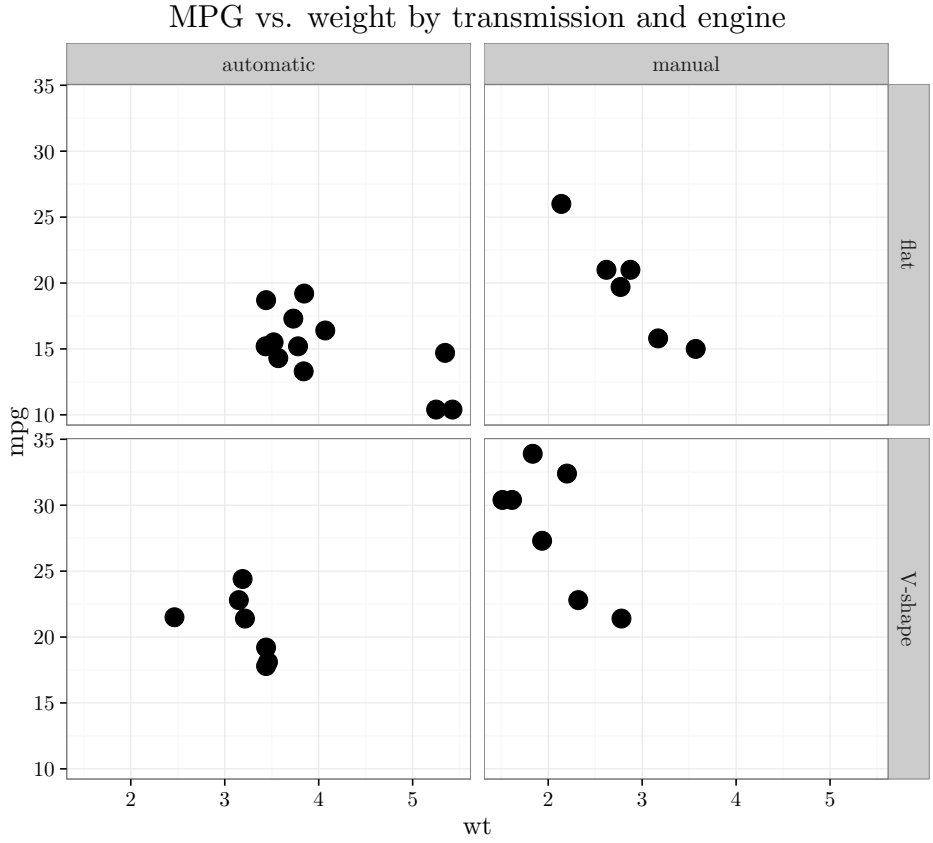
```
mtcars$amf[mtcars$am==0] = 'automatic'
mtcars$amf[mtcars$am==1] = 'manual'
mtcars$vsf[mtcars$vs==0] = 'flat'
mtcars$vsf[mtcars$vs==1] = 'V-shape'
qplot(x = wt,
      y = mpg,
      facets = vsf~.,
      data = mtcars,
      main = "MPG vs. weight by engine") +
  stat_smooth(se = FALSE)
```



We conclude from the graph above that cars with V shape engines tend to weigh less and be more fuel efficient.

The graph below shows a multi-row and multi-column array of panels. It shows that manual transmission and V engine cars tend to be lighter and more fuel efficient. Automatic transmission and non-V engine are heavier and less fuel efficient.

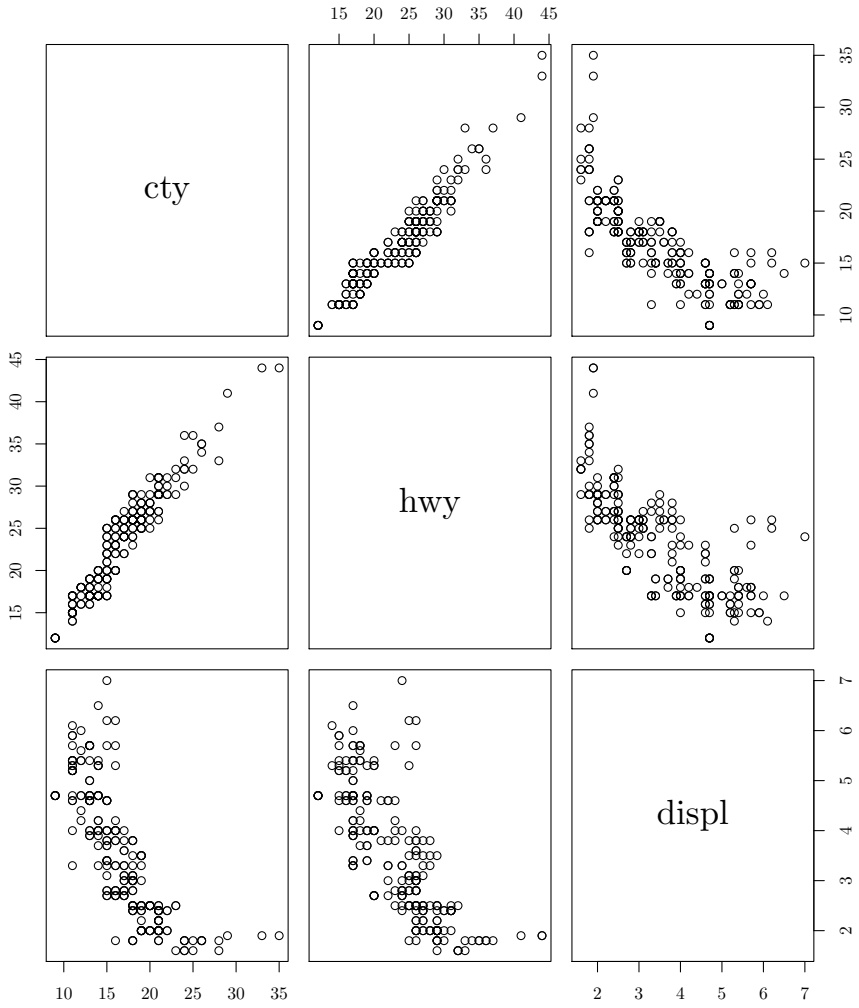
```
mtcars$amf[mtcars$am==0] = 'automatic'
mtcars$amf[mtcars$am==1] = 'manual'
mtcars$vsf[mtcars$vs==0] = 'flat'
mtcars$vsf[mtcars$vs==1] = 'V-shape'
qplot(x = wt,
      y = mpg,
      data = mtcars,
      facets = vsf~amf,
      main = "MPG vs. weight by transmission and engine")
```



The function `plot` can create a similar array of panels with synchronized axes scales when it receives an entire dataframe as an argument. We demonstrate this below by exploring all-pairs relationships between city mpg, highway mpg, and engine displacement volume.

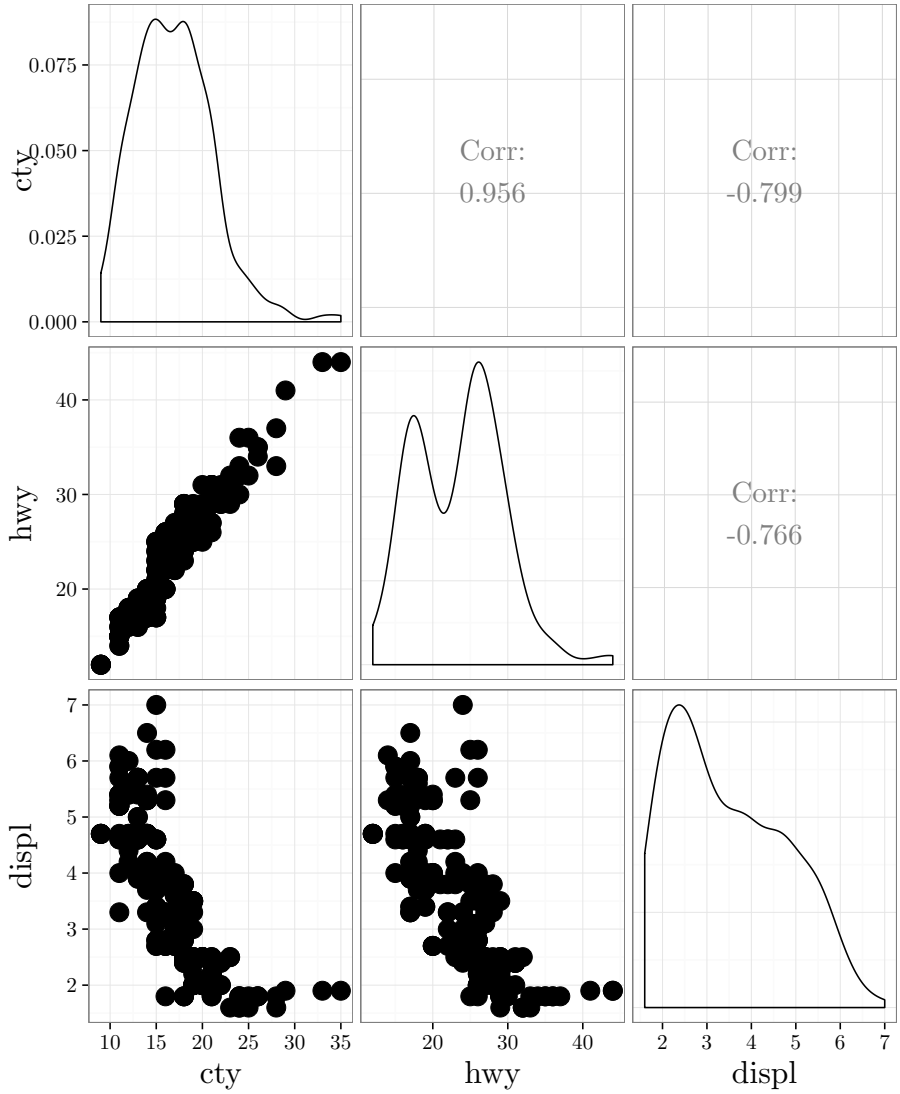
```
# create a new dataframe with three columns: cty, hwy, and disp
DF = mpg[, c("cty", "hwy", "displ")]
plot(DF, main = "City MPG vs. Highway MPG vs. Engine Volume")
```

City MPG vs. Highway MPG vs. Engine Volume



An alternative is the `ggpairs` function from the `GGallly` package. It also displays smoothed histograms of all variables in the diagonal panels and the correlation coefficients in the upper triangle.

```
library(GGally)
ggpairs(DF)
```



We can conclude from the plot above that while highway mpg and city mpg tend to increase together linearly, they are in inverse (non-linear) relationship to the engine displacement volume.

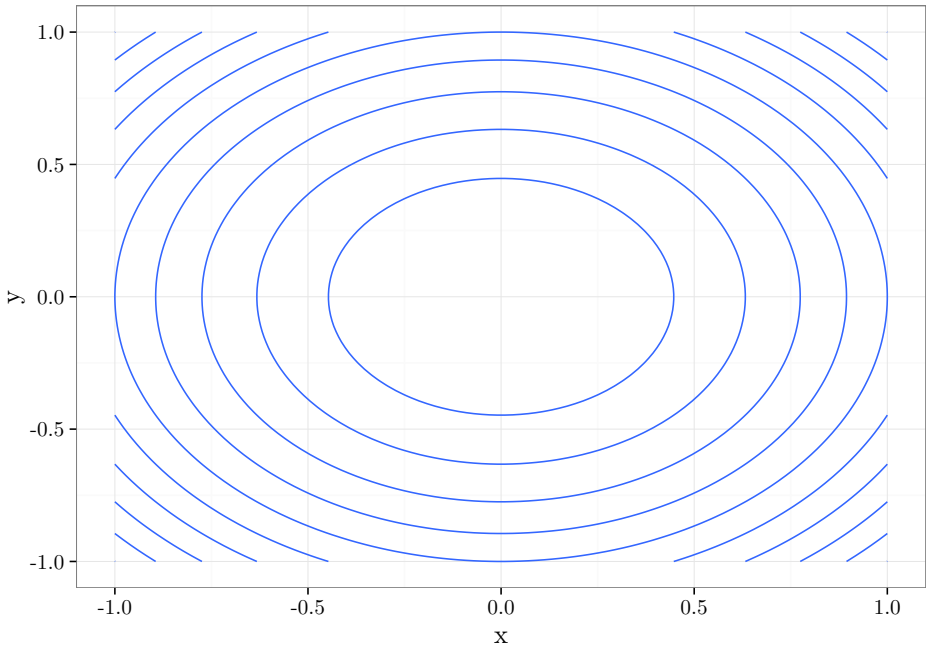
9.9 Contour Plots

The most convenient way to graph a two dimensional function $f(x, y)$ is by graphing its equal height contours

$$z_c = \{(x, y) \in \mathbb{R}^2 : f(x, y) = c\}$$

for different values of c. To graph such a function with the ggplot2 package, create a dataframe with columns corresponding to the `x`, `y`, and `z` values. The `x` and `y` columns should feature all possible combinations of the two coordinates over a certain grid. Then call `ggplot` and add the `stat_contour` layer.

```
x_grid = seq(-1, 1, length.out = 100)
y_grid = x_grid
# create a dataframe containing all possible combinations of x,y
R = expand.grid(x_grid, y_grid)
# number of rows is 100 x 100 - one for each combination
dim(R)
## [1] 10000     2
# modify column names for clear axes labeling
names(R) = c('x', 'y')
R$z = R$x^2 + R$y^2
head(R)
##            x  y        z
## 1 -1.0000000 -1 2.000000
## 2 -0.9797980 -1 1.960004
## 3 -0.9595960 -1 1.920824
## 4 -0.9393939 -1 1.882461
## 5 -0.9191919 -1 1.844914
## 6 -0.8989899 -1 1.808183
ggplot(R, aes(x = x,y = y, z = z)) + stat_contour()
```



9.10 Quantiles and Box Plots

Histograms are very useful for summarizing numeric data in that they show the rough distribution of values. An alternative that is often used in conjunction with the histogram is the box plot. We start with describing the notion of percentiles that plays a central role in our discussion.

The r-percentile of a numeric dataset is the point at which approximately r percent of the data lie underneath, and approximately $100-r$ percent lie above[1]. Another name for the r percentile is the $0.r$ quantile.

```
# display 0 through 100 percentiles at 0.1 increments
# for the dataset containing 1,2,3,4.
quantile(c(1, 2, 3, 4), seq(0, 1, length.out = 11))
##   0%  10%  20%  30%  40%  50%  60%  70%  80%
##  1.0  1.3  1.6  1.9  2.2  2.5  2.8  3.1  3.4
##  90% 100%
##  3.7  4.0
```

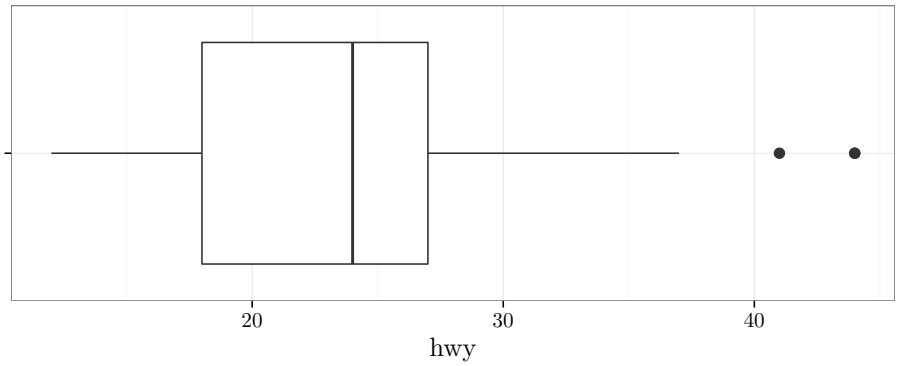
The median or 50-percentile is the point at which half of the data lies underneath and half above. The 25-percentile and 75 percentile are the values below which 25% and 75% of the data lie. These points are also called the first and third quartiles (the second quartile is the median). The interval between the first and third quartiles is called the inter-quartile range (IQR). It is the region covering the central 50% of the data.

The box plot is composed of a box, an inner line bisecting the box, whiskers that extend to either side of the box, and outliers. The box denotes the IQR, with the inner bisecting line denoting the median. The median may or may not be in the geometric center of the box, depending on whether the distribution of values is skewed or symmetric. The whiskers extend to the most extreme point no further than 1.5 times the length of the IQR away from the edges of the box. Data points outside the box and whiskers' range are called outliers and are graphed as separate points. Separating the outliers from the box and whiskers is useful for avoiding a distorted viewpoint where there are a few extreme non-representative values.

The following code graphs a box plot in R using the ggplot2 package. The + operator below adds the box plot geometry, flips the x, y coordinates, and removes the y-axis label.

```
ggplot(mpg, aes("",hwy)) +
  geom_boxplot() +
  coord_flip() +
  scale_x_discrete("")
```

[1]There are several different formal definitions for percentiles. Type `help(quantile)` for several competing definitions that R implements.

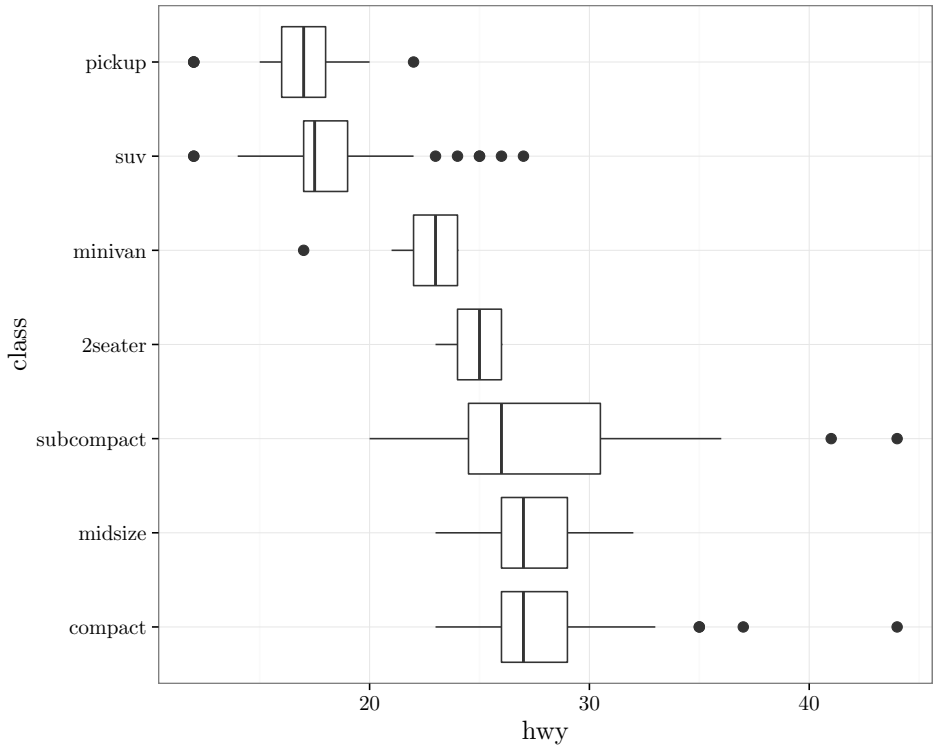


We conclude from this graph that the median highway mpg is around 24, with the central 50% of the data falling within the box that spans the range from 18 to 27. There are two high outliers over 40, but otherwise the remaining data lie within the whiskers between 12 and 37. The fact that the median line is right of the middle of the box hints that the distribution is skewed to the right.

Contrast the box plot above with the smoothed histogram in Page 318 (center panel). The box plot does not convey the multimodal nature of the distribution that the histogram shows. On the other hand, it is easier to read the median and the IQR, which show the center and central 50% range from the box plot.

It is convenient to plot several box plots side by side in order to compare data corresponding to different values of a factor variable. We demonstrate this by graphing below box plots of highway mpg for different classes of vehicles. We flip the box plots horizontally using `coord_flip()` since the text labels display better in this case. Note that we re-order the factors of the `class` variable in order to sort the box plots in order of increasing highway mpg medians. This makes it easier to compare the different populations.

```
ggplot(mpg, aes(reorder(class, -hwy, median), hwy)) +
  geom_boxplot() +
  coord_flip() +
  scale_x_discrete("class")
```



The graph suggests the following fuel efficiency order among vehicle classes: pickups, SUV, minivans, 2-seaters, sub-compacts, midsizes, and compacts. The compact and midsize categories have almost identical box and whiskers but the compact category has a few high outliers. The spread of subcompact cars is substantially higher than the spread in all other categories. We also note that SUVs and two-seaters have almost disjoint values (the box and whisker ranges are completely disjoint) leading to the observation that almost all 2-seater cars in the survey have a higher highway mpg than SUVs.

9.11 qq-Plots

Quantile-quantile plots, also known as qq-plots, are useful for comparing two datasets, one of which may be sampled from a certain distribution. They are essentially scatter plots of the quantiles of one dataset vs. the quantiles of another dataset. The shape of the scatter plot implies the following conclusions (the proofs are straightforward applications of probability theory).

- A straight line with slope[2] 1 that passes through the origin implies that the two datasets have identical quantiles, and therefore that they are sampled from the same distribution.

[2]Slope 1 corresponds to 45 degrees incline from left to right.

- A straight line with slope 1 that does not pass through the origin implies that the two datasets have distributions of similar shape and spread, but that one is shifted with respect to the other.

- A straight line with slope different from 1 that does not pass through the origin implies that the two datasets have distributions possessing similar shapes but that one is translated and scaled with respect to the other.

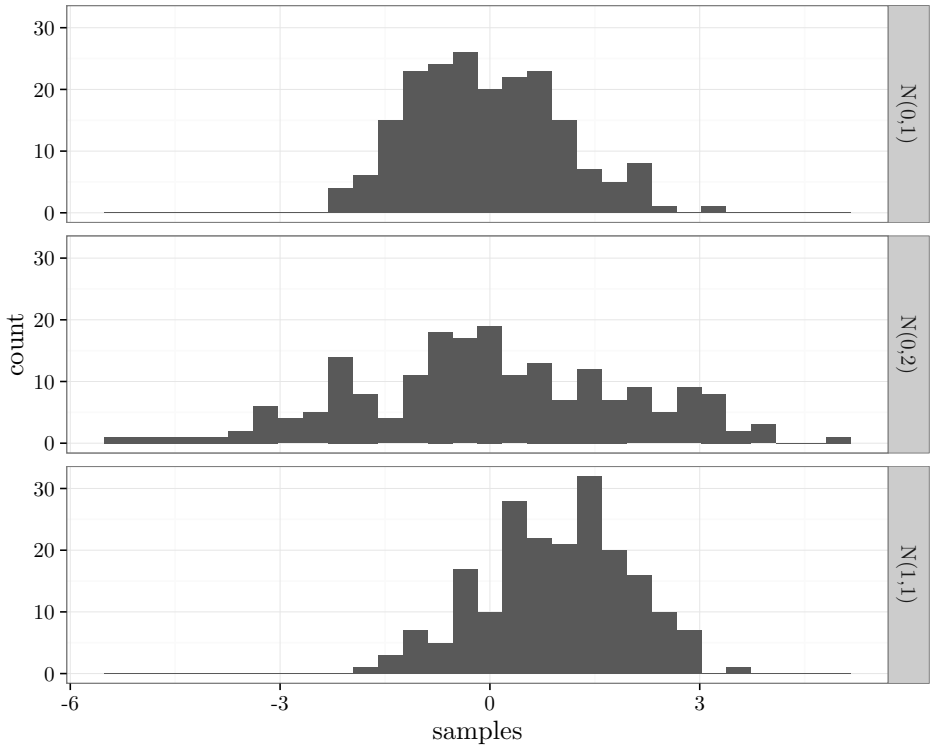
- A non-linear S shape implies that the dataset corresponding to the x-axis is sampled from a distribution with heavier tails than the other dataset.

- A non-linear reflected S shape implies that the dataset whose quantiles correspond to the y-axis is drawn from a distribution having heavier tails than the other dataset.

To compare a single dataset to a distribution we sample values from the distribution, and then display the qq-plots of the two datasets. The quantiles of the sample drawn from the distribution are sometimes called theoretical quantiles.

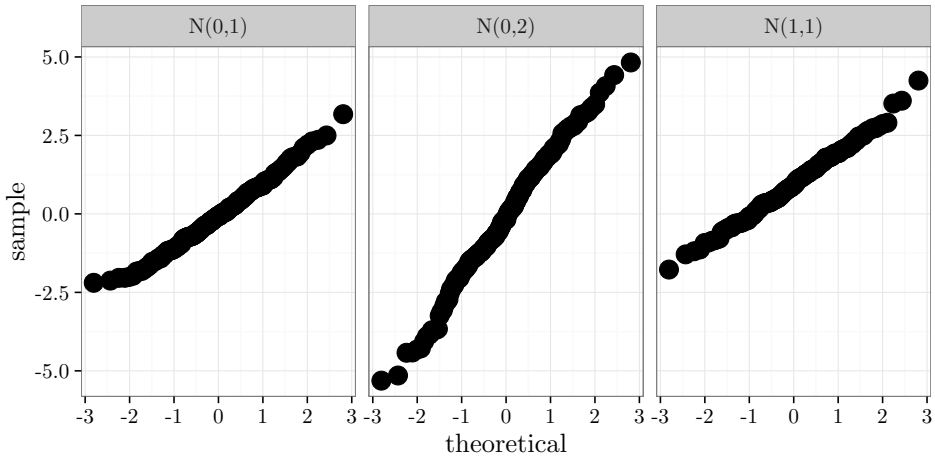
For example, consider the three datasets sampled from three bell-shaped Gaussian distributions $N(0,1)$, $N(0,1)$, and $N(0,2)$ (a precise definition and a discussion of these important distributions appears in TAOD volume 1, Chapter 3). The corresponding histograms appear below.

```
D = data.frame(samples = c(rnorm(200, 1, 1),
                           rnorm(200, 0, 1),
                           rnorm(200, 0, 2)))
D$parameter[1:200]   = 'N(1,1)';
D$parameter[201:400] = 'N(0,1)';
D$parameter[401:600] = 'N(0,2)';
qplot(samples,
      facets = parameter~.,
      geom = 'histogram',
      data = D)
```



We compute below the qq-plots of these three datasets (y axis) vs. a sample drawn from the $N(0,1)$ distribution (x axis).

```
D = data.frame(samples = c(rnorm(200, 1, 1),
                           rnorm(200, 0, 1),
                           rnorm(200, 0, 2)));
D$parameter[1:200]   = 'N(1,1)';
D$parameter[201:400] = 'N(0,1)';
D$parameter[401:600] = 'N(0,2)';
ggplot(D, aes(sample = samples)) +
  stat_qq() +
  facet_grid(.~parameter)
```



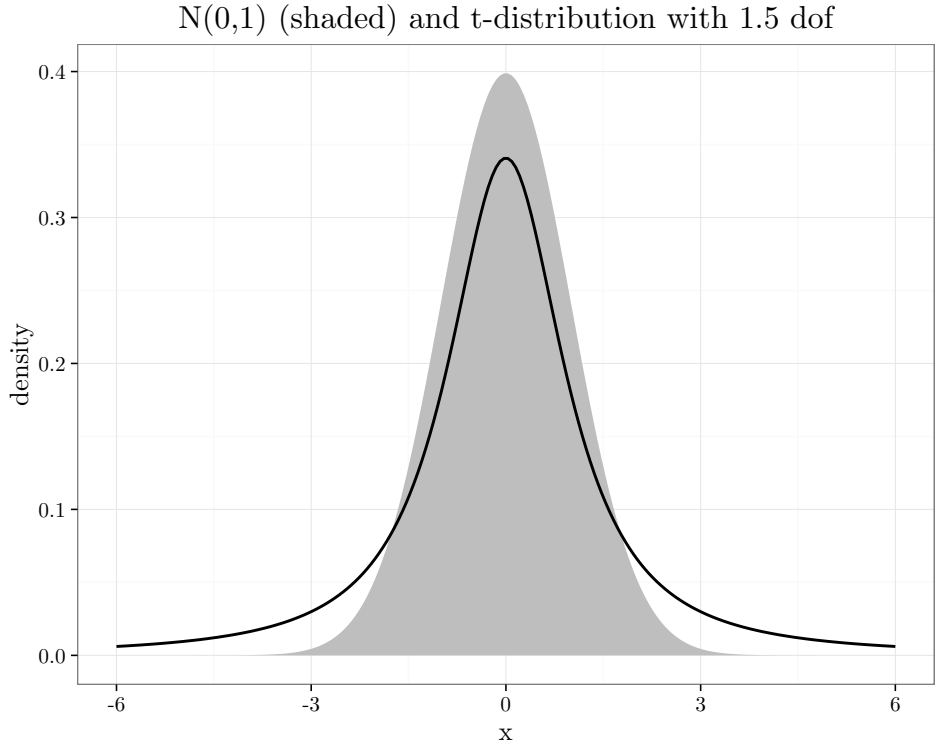
Note how all three plots are linear, implying that the three datasets have distributions similar in shape to the $N(0,1)$ distribution up to translation and scaling. In the left panel we get a linear shape with slope 1 which passes through the origin, since the two datasets were sampled from the same $N(0,1)$ distribution. In the middle panel we get a line passing through the origin but with a steeper slope, since the data was more wide-spread than the $N(0,1)$ distribution. In the right panel we get a slope 1 line that does not pass through the origin since the two distributions have identical spread but are translated with respect to each other.

As a final example we show the qq-plot of a sample from a $N(0,1)$ distribution and a sample from a t-distribution[3] with 1.5 degrees of freedom. In contrast to the Gaussian $N(0,1)$ density whose tails decrease exponentially, the t density has tails that decrease significantly slower at a polynomial rate. As a result the two distributions have fundamentally different shapes and their quantiles are non-linearly related.

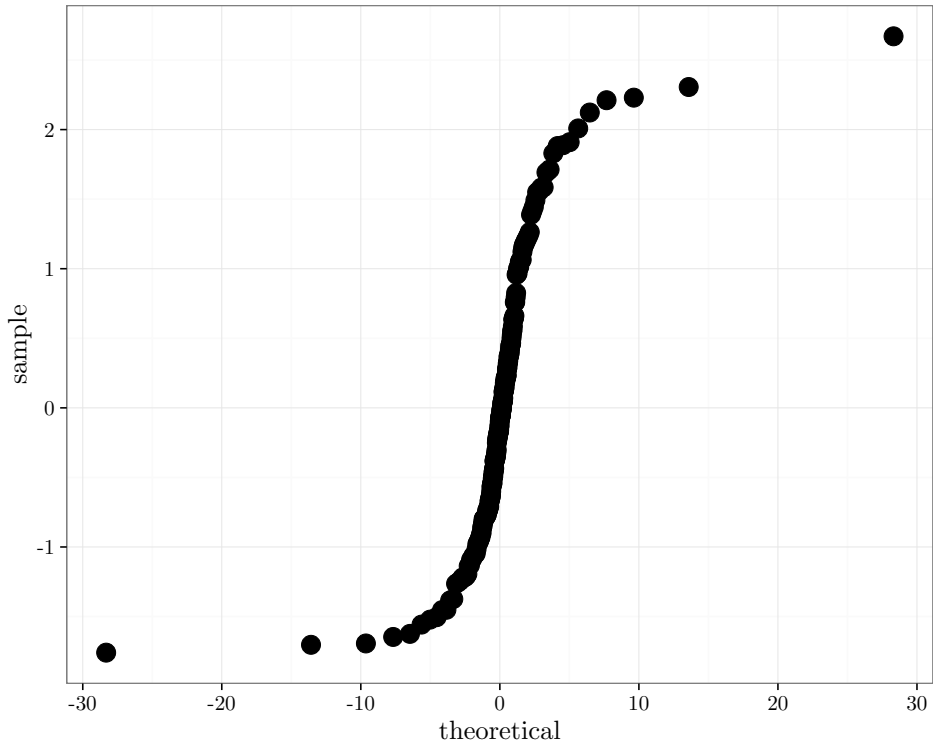
The following graph demonstrates the shape of the densities of the two distribution and how the t-distribution has heavier tails than the Gaussian $N(0,1)$ distribution.

```
x_grid = seq(-6, 6, length.out = 200)
R = data.frame(density = dnorm(x_grid, 0, 1))
R$tdensity = dt(x_grid, 1.5)
R$x = x_grid
ggplot(R, aes(x = x, y = density)) +
  geom_area(fill = I('grey')) +
  geom_line(aes(x = x, y = tdensity)) +
  labs(title = "N(0,1) (shaded) and t-distribution with 1.5 dof")
```

[3]A formal definition of the t-distribution appears in TAOD volume 1, Chapter 3.




```
x_grid = seq(-6, 6, length.out = 200)
R = data.frame(density = dnorm(x_grid, 0, 1))
R$samples = rnorm(200, 0, 1)
pm = list(df = 1.5)
ggplot(R, aes(sample = samples)) +
  stat_qq(distribution = qt, dparams = pm)
```



9.12 Devices

By default, R overwrites the current figure with new plots. The function call `dev.new()` opens a new additional graphics windows. The `ggsave` function within the ggplot2 package saves the active graphics window to a file with the file type (pdf, postscript, jpeg) corresponding to the file name extension.

```
# detects file-type format (PDF) from file name extension
ggsave(file = "myPlot.pdf")
```

To send all future graphics to a single file use one of the following functions: `postscript`, `pdf`, `xfig`, `bmp`, `png`, `jpeg`, or `tiff`. It is essential to call the function `dev.off()` at the end of the graphics session in order to ensure that the graphics file is closed properly. To see a precise list of optional parameters such as font size and compression rate refer to `help(X)` where `X` is one of the device drivers, for example `help("jpeg")`.

The first three drivers (`postscript`, `pdf`, `xfig`) maintain high-resolution graphics using vector graphics. The resulting graphics can be zoomed in to arbitrary precision. Among these formats, pdf is usually preferred, since it is often smaller in file size and since it is accessible by a wide variety of programs.

The latter four drivers (`bmp`, `png`, `jpeg`, `tiff`) produce raster graphics which correspond to pixelized images with fixed resolutions. While vector graphics is generally preferable to raster graphics due to its superior resolution, vector graphics may produce very large files when the graphics contain many objects. In that case a raster graphics file may be preferable due to its smaller file size.

```
# save all future graphics to file myplots.pdf
pdf('myplots', height = 5, width = 5, pointsize = 11)
#  graphics plotting
qplot(...)
qplot(...)
#  close graphics file and return to display driver
dev.off()
```

9.13 Data Preparation

We emphasize in this book graphing data by first creating a dataframe with the appropriate data and informative column names, and then calling `plot`, `qplot`, or `ggplot`. This approach is better than keeping the data in an unannotated array, graphing the values, and then labeling the axes, legends, and facets appropriately. In the examples above we usually started with a ready-made dataframe, but in most data analysis cases the dataframe has to be prepared by the data analyst.

To create a dataframe use the `data.frame` function, for example:

```
R = data.frame(name = vec1, ages = vec2, salary = vec3).
```

If a dataframe already exists, but the variable names are not coherent, change the names using the `names` function to coherent names so that legible axes and legends can be automatically created.

```
names(R) = c("last.name", "age", "annual.income")
```

The functions `rbind` and `cbind` add additional rows or columns to an existing dataframe.

Consider the following example of graphing the line plots of the Gaussian density function (see TAOD volume 1, Chapter 3)

$$f(x) = N(x\,;0,\sigma) = \exp(-x^2/(2\sigma^2))/(\sqrt{2\pi\sigma^2})$$

with the color and line type corresponding to the value of σ among four different values: 1, 2, 3, and 4. Note that this function is also $K_h(x)$ for the Gaussian kernel described earlier in this chapter.

Our strategy is to first compute a list of four vectors containing y values — one vector for each value of σ. The names of the four list elements identify the

σ corresponding to that element. We then use the `stack` function to create the dataframe.

The following code helps illustrate the idea before we move on to the complete example below.

```
A = list(a = c(1, 2), b = c(3, 4), c = c(5, 6))
A
## $a
## [1] 1 2
##
## $b
## [1] 3 4
##
## $c
## [1] 5 6
stack(A)
##   values ind
## 1      1   a
## 2      2   a
## 3      3   b
## 4      4   b
## 5      5   c
## 6      6   c
```

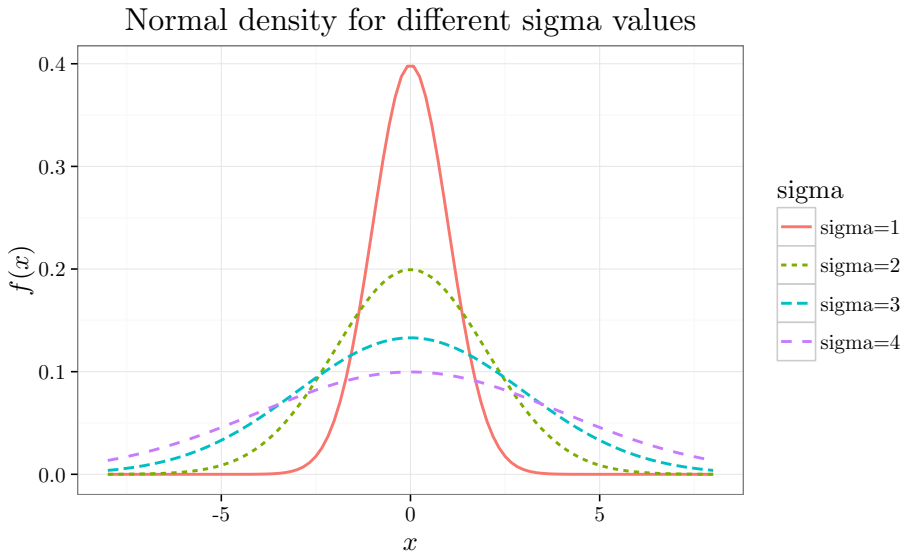
The dataframe above is ready for graphing. The first column contains the values of the variable that is being visualized, and the second column contains a variable that is used to distinguish different graphs using overlays or facets.

The code below provides a complete example.

```
x_grid = seq(-8, 8, length.out = 100)
gaussian_function = function(x, s) exp(-x^2/(2*s^2))/(sqrt(2*pi)*s)
R = stack(list('sigma=1' = gaussian_function(x_grid, 1),
               'sigma=2' = gaussian_function(x_grid, 2),
               'sigma=3' = gaussian_function(x_grid, 3),
               'sigma=4' = gaussian_function(x_grid, 4)))

names(R) = c("y", "sigma");
R$x = x_grid
head(R)
##              y   sigma         x
## 1 5.052271e-15 sigma=1 -8.000000
## 2 1.816883e-14 sigma=1 -7.838384
## 3 6.365366e-14 sigma=1 -7.676768
## 4 2.172582e-13 sigma=1 -7.515152
## 5 7.224128e-13 sigma=1 -7.353535
## 6 2.340189e-12 sigma=1 -7.191919
qplot(x,
      y,
      color = sigma,
```

```
lty = sigma,
geom = "line",
data = R,
main = "Normal density for different sigma values",
xlab = "$x$",
ylab = "$f(x)$")
```



9.14 Python's Matplotlib Module

Despite the fact that R has excellent graphics capabilities it is sometimes desirable to graph data in another programming language. For example, if the entire data analysis process is in Python, it may make sense to graph the data within Python rather than save the data in Python, load it in R, and graph it in R. We describe below matplotlib – a popular Python module for graphing data. Matplotlib features two interfaces: (a) the default object oriented programmatic interface, and (b) pylab - a Matlab-like interface. We focus below on pylab as it is simpler and it may be familiar to readers previously exposed to Matlab. To get access to pylab's API start the interactive IPython program using the command `ipython --pylab`.

9.14.1 Figures

The function `figure` opens a new figure and returns an object that may be used to display graphs in that figure. Multiple `figure` functions may be issued, and by default the figure that was opened last is the active figure. The function `close(X)` closes figure `X` (or by default the active figure if the argument is

omitted). The function `savefigX` saves the active figure to a file whose filename is `X` (the file type is inferred from the filename extension). For example, the following code opens up two figures, saves the second figure and closes it, and then saves and closes the first figure.

```
import matplotlib.pyplot as plt
f1 = plt.figure()  # open a figure
f2 = plt.figure()  # open a second figure
plt.savefig('f2.pdf')  # save fig 2 - the active figure
plt.close(f2)
plt.savefig('f1.pdf')  # save fig 2 - the active figure
plt.close(f1)
```

9.14.2 Scatter-plots, Line-plots, and Histograms

The function `plot(x,y)` displays a scatter plot of the two arrays `x` and `y`, and connects the scatter plot points with lines.

An optional third argument for `plot` is a string containing the color code, followed by marker code that is in turn followed by line-style code. For example the string `'ro--'` corresponds to color red, circular scatter plot markers, and dashed line. If some of these patterns are omitted the default choice is selected. Omitting the scatter plot markers pattern creates a line plot, for example `'r--'` creates a red line plot. Omitting the line-style pattern creates a scatter plot, for example `'ro'` creates a red scatter plot. Multiple consecutive plot functions add additional features to the current figure.

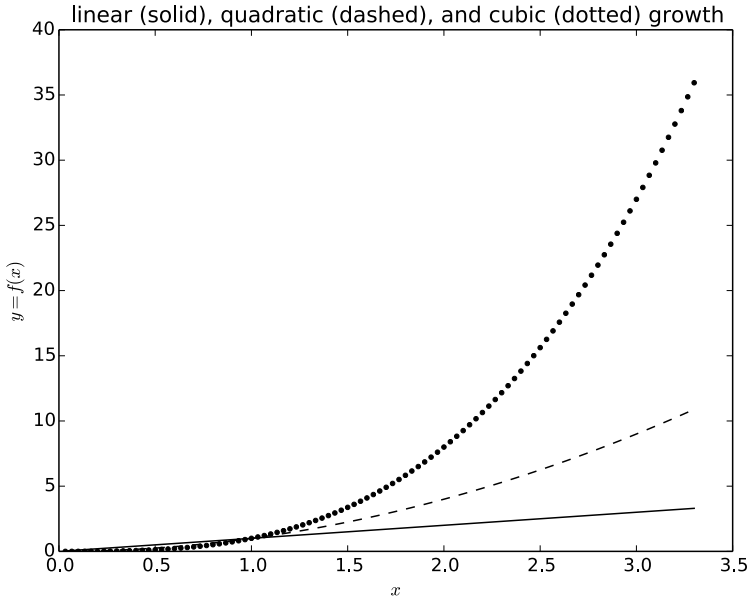
The functions `xlabel`, `ylabel`, and `title` annotates the x-axis, the y-axis and the figure. As in the R examples we can use LaTeX code (text surrounded by $ symbols in the example below) to annotate the axes labels or titles with equations.

The range of values displayed in the x-axis and y-axis can be modified using the function `xlim([min_x, max_x])` and `ylim([min_y, max_y])` functions.

The code below displays three line plots - linear (black solid line), quadratic (black dashed line), and cubic (black dotted line). Since the scatter plot marks patterns are omitted, we get line plots without the scatter plot corresponding to the sampled points.

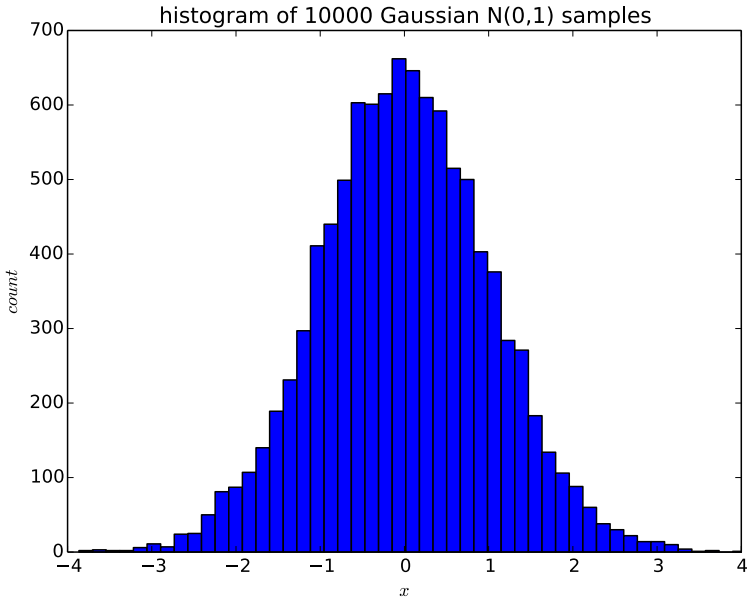
```
import matplotlib.pyplot as plt
x_grid = array(range(1, 100)) / 30.0
plt.figure()  # open a figure
plt.plot(x_grid, x_grid, 'k-')  # adds f(x)=x as dashed
plt.plot(x_grid, x_grid ** 2, 'k--')  # draws f(x)=x^2 as solid line
plt.plot(x_grid, x_grid ** 3, 'k.')  # draws f(x)=x^2 as solid line
plt.xlabel(r'$x$')
plt.ylabel(r'$y=f(x)$')
ttl='linear (solid), quadratic (dashed), and cubic (dotted) growth'
```

```
plt.title(ttl)
```



The function `hist(x,n)` creates a histogram of the data in `x` using `n` bins.

```
import matplotlib.pyplot as plt
data = randn(10000)
hist(data, 50)
plt.xlabel(r'$x$')
plt.ylabel(r'$count$')
plt.title('histogram of 10000 Gaussian N(0,1) samples')
plt.xlim([-4, 4])
```



9.14.3 Contour Plots and Surface Plots

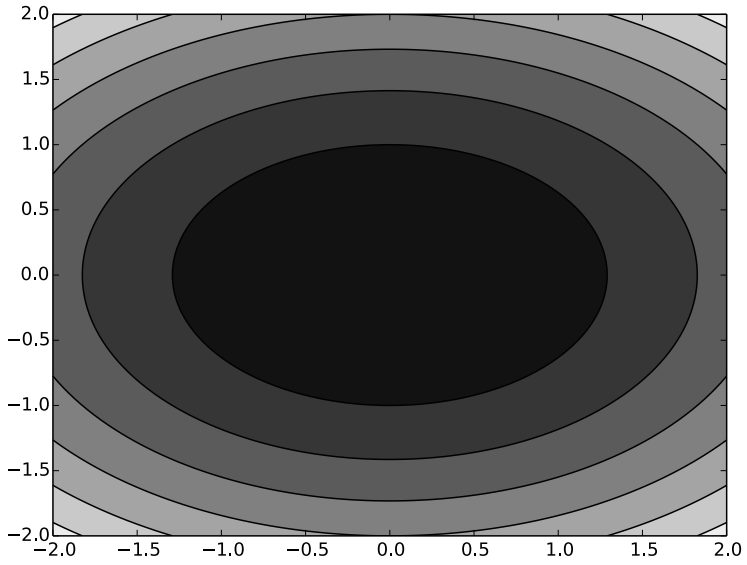
Matplotlib can also graph three dimensional data. We describe below how to create contour plots and surface plots - the two most common 3-D graphs.

To create a contour plot or surface plot of a function $f = (x, y)$, we need to first create two one dimensional grids corresponding to the values of x and y. The function `numpy.meshgrid` takes these two one dimensional grids and returns two two dimensional ndarrays containing the x and y values (the first ndarray has constant columns and the second ndarray has constant rows). We can then create a third ndarray holding the values of $z = f(x, y)$ by operating a two dimensional function on the two ndarrays.

The function `contourf` displays $z = f(x, y)$ as a function of x, y, quantizing the z values into several constant values. The code below shows how to use `contourf` to display a contour plot of the function $z = 3x^2 + 5y^2$.

```
import numpy as np
import matplotlib.pyplot as plt
def f(x, y): return (3*x**2 + 5*y**2)
x_grid = np.linspace(-2, 2, 100)
y_grid = np.linspace(-2, 2, 100)
# create two ndarrays xx, yy containing x and y coordinates
xx, yy = np.meshgrid(x_grid, y_grid)
zz = f(xx, yy)
# draw contour graph with 6 levels using gray colormap
#   (white=high, black=low)
```

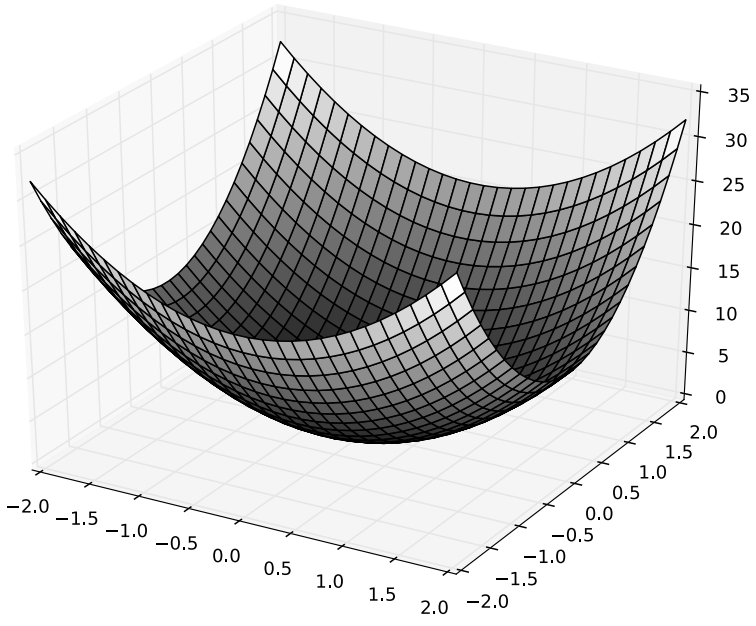
```
plt.contourf(xx,
             yy,
             zz,
             6,
             cmap = 'gray')
# add black lines to highlight contours levels
plt.contour(xx,
            yy,
            zz,
            6,
            colors = 'black',
            linewidth = .5)
```



The function `plot_surface` is similar to `contourf` except that it displays a 3-D surface plot. The code below shows how to use it. Note that before creating a 3-D plot we need to create a 3-D axis object using the function `Axes3D` in `mpl_toolkits.mplot3d`.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
def f(x, y): return (3*x**2 + 5*y**2)
x_grid = np.linspace(-2, 2, 30)
y_grid = np.linspace(-2, 2, 30)
xx, yy = np.meshgrid(x_grid, y_grid)
zz = f(xx, yy)
surf_figure = plt.figure()
```

```
figure_axes = Axes3D(surf_figure)
figure_axes.plot_surface(xx,
                         yy,
                         zz,
                         rstride = 1,
                         cstride = 1,
                         cmap = 'gray')
```



9.15 Notes

Refer to the official R manual at http://cran.r-project.org/doc/manuals/R-intro.html (replace html with pdf for pdf version) and the language reference at http://cran.r-project.org/doc/manuals/R-lang.html (replace html with pdf for pdf version) for more information on R graphics. Detailed information on the ggplot2 package appears in [29] or on the website http://had.co.nz/ggplot2/. A comprehensive description of the grammar of graphics which is the basis of the ggplot2 package is available in [31]. Two useful books on how to construct useful graphs are [3, 4]. A classic book on exploratory data analysis is [27].

The package `lattice` [22] is a popular alternative to graphics and ggplot2. It often creates graphs faster than ggplot2, but its syntax is less intuitive. Many other R packages feature graphics functions for specialized data such as time series, financial data, or geographic maps. A useful resource for exploring such packages is the Task Views http://cran.r-project.org/web/views/.

Python's matplotlib can display additional types of graphs, including bar

charts, two dimensional scatter plots, three dimensional surface plots. See http://matplotlib.org for details. Python's pandas module has some graphics functionality that is useful for graphing dataframes. See http://pandas.pydata.org for details. Python also has additional graphics module for specialized graphics, such as interactive graphics.

9.16 Exercises

1. Using the `mpg` data, describe the relationship between highway mpg and car manufacturer. Describe which companies produce the most and least fuel efficient cars, and display a graph supporting your conclusion.

2. Using the `mpg` data, explore the three-way relationship between highway mpg, city mpg, and model class. What are your observations? Display a graph supporting these observations.

3. What are the pros and cons of using a histogram vs a box plot? Which one will you prefer for what purpose?

4. Generate two sets of N random points using the function `runif` and display a corresponding scatter plot. If you save the file to disk, what is the resulting file size for the following file formats: ps, pdf, jpeg, png? How do these values scale with increasing N?

5. The `diamonds` dataset within ggplot2 contains 10 columns (price, carat, cut, color, etc.) for 53940 different diamonds. Type `help(diamonds)` for more information. Plot histograms for color, carat, and price, and comment on their shapes. Investigate the three-way relationship between price, carat, and cut. What are your conclusions? Provide graphs that support your conclusions. If you encounter computational difficulties, consider using a smaller dataframe whose rows are sampled from the original `diamonds` dataframe. Use the function `sample` to create a subset of indices that may be used to create the smaller dataframe.