Creating Dynamic Documents with RStudio by Unifying Computing and Typesetting

Boyan Kostadinov, BMCC Summer Institute

June 2, 2015

Contents

R Markdown — Dynamic Documents for R

R Markdown is an authoring format that enables easy creation of dynamic documents, presentations, and reports from R. It combines the core syntax of markdown (an easy-to-write plain text format) with embedded R code chunks that are run so their output can be included in the final document. R Markdown documents are fully reproducible (they can be automatically regenerated whenever underlying R code or data changes).

R Markdown offers the following features:

- Many available output formats including HTML, PDF, and MS Word.
- Support for creating Beamer, ioslides, and Slidy presentations.
- New markdown syntax including expanded support for tables and bibliographies.
- (Advanced) Hooks for customizing HTML and PDF output (include CSS, headers, and footers).
- Include raw LaTeX within markdown for advanced customization of PDF output.
- Compile HTML, PDF, or MS Word notebooks from R scripts.
- (Advanced) Create interactive R Markdown documents using Shiny.

Note that PDF output (including Beamer slides) requires a full installation of TeX.

We will attempt to give short introductions to the three main technologies that are required for creating dynamic documents: **R Markdown**, **R** and **LaTeX**.

Quick Tour

Installation of LaTeX, R and RStudio

Software Installers:

- MiKTeX for Windows installation: http://miktex.org/download
- MacTeX for Mac installation: http://tug.org/mactex/
- R installation for Windows and Mac: http://cran.stat.ucla.edu/
- RStudio installation for Windows and Mac: http://www.rstudio.com/products/rstudio/download/

You can install the R Markdown package from CRAN as follows: `install.packages("rmarkdown")` in RStudio.

Markdown Basics

Markdown is a simple formatting language designed to make authoring content easy for everyone. Rather than writing complex markup code (e.g. HTML or LaTeX), Markdown enables the use of a syntax much more like plain-text email. When you click the **Knit** button in RStudio a document will be generated that includes both content as well as the output of any embedded R code chunks within the document.

R Code Chunks

Within an R Markdown file, R Code Chunks can be embedded using the native Markdown syntax.

Inline R Code

You can also evaluate R expressions inline by enclosing the expression within a single back-tick qualified with 'r'. For example, the following inline R code:

```
`r exp(sqrt(2))`
```

will compute `exp(sqrt(2))` and return the number 4.1132504.

As another example, the following expression:

```
I counted `r 77/11 + 2^3` red trucks on the highway.
```

Results in this output: "I counted 15 red trucks on the highway."

Embedding Equations

You can embed LaTeX or MathML equations in R Markdown files using the following syntax:

- `$equation$` for inline equations (note there must not be white space adjacent to the $ delimiters).
- `$$ equation $$` for display equations.

For example, the expression: `The arithmetic mean is equal to $\frac{1}{n}\sum_{i=1}^n x_i$.`

results in this output:

The arithmetic mean is equal to $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$.

As an example of display style equations, the TeX expression `$$ E(X)=\int_{-\infty}^{\infty} xf_X(x)dx $$`

results in the following typeset formula:

$$E(X) = \int_{-\infty}^{\infty} x f_X(x) dx$$

Rendering Output

There are two ways to render an R Markdown document into it's final output format. If you are using RStudio, then the "Knit" command (Ctrl+Shift+K) will render the document and display a preview of it.

Output Options

R Markdown documents can contain a metadata section that includes both title, author, and date information as well as options for customizing output. For example, this metadata included at the top of an Rmd file adds a table of contents and chooses a different HTML theme:

```
---
title: "Sample Document"
output:
  html_document:
    toc: true
    theme: united
---
```

R Markdown has built in support for several output formats (HTML, PDF, and MS Word documents as well as Beamer presentations). These formats can also be specified in metadata, for example:

```
---
title: "Sample Document"
output:
  pdf_document:
    toc: true
    highlight: zenburn
---
```

Markdown Basics

Emphasis

These expressions

```
*italic*  and  **bold**
```

give the following:

italic and **bold**

Headers

```
# Header 1

## Header 2

### Header 3
```

Lists

Unordered List:

```
* Item 1
* Item 2
    + Item 2a
    + Item 2b
```

produces:

- Item 1
- Item 2
  - Item 2a
  - Item 2b

Ordered List:

```
1. Item 1
2. Item 2
3. Item 3
    + Item 3a
    + Item 3b
```

produces:

1. Item 1
2. Item 2
3. Item 3
   - Item 3a
   - Item 3b

R Code Chunks

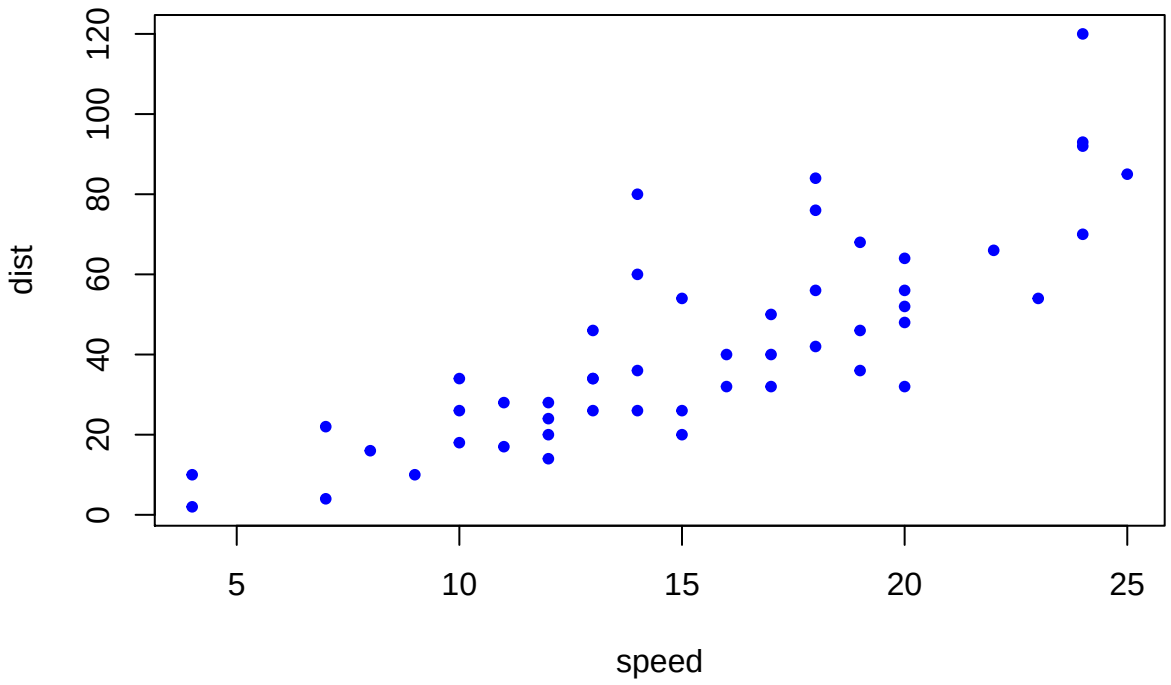
R code will be evaluated and printed

```
summary(cars$speed)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     4.0    12.0    15.0    15.4    19.0    25.0
```

Plots created with R code are seamlessly embedded into the final document:

```
plot(cars,pch=20,col="blue")
```



Note that the `echo = FALSE` parameter can be used in the code chunk to prevent printing of the R code that generated the plot.

Links

Use a plain http address or add a link to a phrase:

http://google.com

Google

Images

Images on the web or local files in the same directory:

```
![text](http://example.com/logo.png)

![text](figures/img.png)
```

Blockquotes

```
A friend once said:

> It's always better to give
> than to receive.
```

gives:

A friend once said:

> It's always better to give than to receive.

Horizontal Rule

Three or more asterisks:

---

or dashes:

Tables

First Header	Second Header
Content Cell	Content Cell
Content Cell	Content Cell

Miscellaneous

```
superscript^2^

~~strikethrough~~
```

gives:

superscript^2

~~strikethrough~~

End a line with two or more spaces:

Roses are red,
Violets are blue.

LaTeX Basics

- Short Math Guide for LaTeX
- LaTeX Mathematics
- LaTeX Templates

Typesetting Mathematical Expressions

R Markdown offers a limited TeX functionality but good enough for creating any kind of teaching materials. There are two main ways to introduce mathematical expressions in LaTeX:

1. Inline TeX expressions: the code `$\sin(\alpha)$` gives $\sin(\alpha)$
2. Display TeX expressions: the code `$$\sin(\alpha)$$` gives

$$\sin(\alpha)$$

R Markdown does not support equation arrays and more complex LaTeX structures because it was designed to serve as a quick development platform but this feature also makes it perfect for both faculty and student when it comes to creating educational materials, project modules and reports.

All standard functions such as all trigonometric functions (sin, cos, tan), logarithms and exponentials (log, exp), etc. are operators in LaTeX defined as commands:

- `$\cos (2\theta) = \cos^2 \theta - \sin^2 \theta$` produces: $\cos(2\theta) = \cos^2\theta - \sin^2\theta$

For certain operators such as limits, the subscript is placed underneath the operator:

- `$\lim_{x \to \infty} \exp(-x) = 0$` produces: $\lim_{x\to\infty} \exp(-x) = 0$

In order for some operators, such as `$\lim$` or `$\sum$` to be displayed correctly inline, it might be convenient to add the `\displaystyle` command. Doing so will cause exponents and indices to be displayed correctly for some math operators. For example, the `$\sum$` will print a smaller $\sum$ and `$\displaystyle \sum$` will print a bigger one $\displaystyle\sum$ as in display style equations.

Powers and indices are equivalent to superscripts and subscripts in normal text mode. The **caret** (^) character is used to raise something, and the underscore (_) is for lowering. If more than one expression is raised or lowered, they should be grouped using curly braces {}.

- `$k_{n+1} = n^3 + k_n^2 - k_{n-1}$` produces: $k_{n+1} = n^3 + k_n^2 - k_{n-1}$

For powers with more than one digit, surround the power with {}.

- `$n^{12}$` produces n^{12}

An underscore (_) can be used with a vertical bar (|) to denote evaluation using subscript notation:

- `$(4n^2 + 1)|_{n=5}$` produces $(4n^2 + 1)|_{n=5}$

A fraction is created using the `$\frac{numerator}{denominator}$` command. Likewise, the binomial coefficient may be written using the `$\binom{}{}$` command:

- `$\displaystyle\frac{n!}{k!(n-k)!} = \binom{n}{k}$` produces $\displaystyle\frac{n!}{k!(n-k)!} = \binom{n}{k}$

You can embed fractions within fractions:

- `$\displaystyle\frac{\frac{1}{x}+\frac{1}{y}}{y-z}$` produces $\displaystyle\frac{\frac{1}{x}+\frac{1}{y}}{y-z}$

For relatively simple fractions, especially within the text, it may be better to use powers and indices:

- `$^3/_7$` produces $^3/_7$ or a tighter version with negative space`\!` after 3 and before underscore `$^3\!/\!_7$` produces $^3\!/\!_7$, as opposed to the simple fraction `$3/7$`, which produces 3/7.

It is very useful to note that special LaTeX packages can be included by adding to the header of the Rmd document a special line specifying the LaTeX package. However, including extra LaTeX packages in the Rmd document works only of the output is pdf, rather than html. For example the line `header-includes: \usepackage{xfrac}` will include the **xfrac** package that includes the command `\sfrac{}{}` for a tight, inline fractions. In particular, the code `$\sfrac{3}{7}$` will work but only when the output is pdf.

The `\sqrt{}` command creates a square root surrounding an expression inside the curly braces. It accepts an optional argument specified in square brackets [and] to change magnitude:

- `$\displaystyle\sqrt{\frac{a}{b}}$` produces $\displaystyle\sqrt{\frac{a}{b}}$
- `$\displaystyle\sqrt[5]{1+x+x^2+x^3+\ldots}$` produces $\displaystyle\sqrt[5]{1+x+x^2+x^3+\ldots}$

The `\sum` and `\int` commands insert the sum and integral symbols respectively, with limits specified using the caret (^) and underscore (_).

- `$\sum_{i=1}^{10} x_i$` produces $\sum_{i=1}^{10} x_i$
- `$\displaystyle\sum_{i=1}^{10} x_i$` produces $\displaystyle\sum_{i=1}^{10} x_i$

The limits for the integrals follow the same notation. For some publications it maybe important to represent the integration variables with an upright d, which in math mode is obtained through the `\mathrm{}` command, and with a small space separating it from the integrand, which is attained with the `\,` command.

- `$\int_0^\infty \mathrm{e}^{-x}\,\mathrm{d}x$` produces $\int_0^\infty \mathrm{e}^{-x}\,\mathrm{d}x$
- `$\displaystyle\int_0^\infty \mathrm{e}^{-x}\,\mathrm{d}x$` produces $\displaystyle\int_0^\infty \mathrm{e}^{-x}\,\mathrm{d}x$

If you want the limits of an integral to be specified above and below the symbol, use the `\limits` command.

- `$\int\limits_a^b f(x)\,dx$` produces $\int\limits_a^b f(x)\,dx$
- `$\displaystyle\int\limits_a^b f(x)\,dx$` produces $\displaystyle\int\limits_a^b f(x)\,dx$

Often mathematical expressions will differ in size, in which case the delimiters surrounding the expression should vary accordingly. This can be done automatically using the `\left`, `\right`, and `\middle` commands.

- `$P\left(X=5\middle|\frac{X^2}{2Y}>1\right)$` produces $P\left(X=5\middle|\frac{X^2}{2Y}>1\right)$

If a delimiter on only one side of an expression is required, then an invisible delimiter on the other side may be denoted using a period (.).

- `$\displaystyle\left.\frac{x^3}{3}\right|_0^1$` produces $\displaystyle\left.\frac{x^3}{3}\right|_0^1$

To typeset an interval use the `\in` command.

- `$x \in [-1,1)$` produces $x \in [-1,1)$

When writing matrices, it is common to use horizontal, vertical and diagonal triplets of dots to fill in certain columns and rows. These can be specified using the `\cdots`, `\vdots` and `\ddots` respectively:

```
\[
A_{m,n} =
 \begin{pmatrix}
  a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\
  a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\
  \vdots  & \vdots  & \ddots & \vdots  \\
  a_{m,1} & a_{m,2} & \cdots & a_{m,n}
 \end{pmatrix}
\]
$$
```

produces the matrix:

$$A_{m,n} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$

When having fractions inside a matrix, an additional leading space maybe added with the optional parameter to the `\\` command:

```
\[
M = \begin{bmatrix}
       \frac{5}{6} & \frac{1}{6} & \frac{7}{11} \\[0.3em]
       \frac{5}{6} & -7          & \frac{1}{6}  \\[0.3em]
       0           & \frac{5}{6} & \frac{1}{6}
     \end{bmatrix}
\]
```

produces:

$$M = \begin{bmatrix} \frac{5}{6} & \frac{1}{6} & \frac{7}{11} \\ \frac{5}{6} & -7 & \frac{1}{6} \\ 0 & \frac{5}{6} & \frac{1}{6} \end{bmatrix}$$

The math environment differs from the text environment in the representation of text. The typical way to add text in math mode is to put the text inside the `\text{...}` command.

- `$$10 \text{ apples} \times 200 \text{ apples} = \text{lots of apples}^2$$` produces

$$10 \text{ apples} \times 200 \text{ apples} = \text{lots of apples}^2$$

A single equation can be boxed using the `\boxed{}` command.

```
$$ \boxed{x^n + y^n = z^n} $$
```

produces:

$$\boxed{x^n + y^n = z^n}$$

The `\begin{align} ...\end{align}` environment is used for two or more equations when vertical alignment is desired; usually binary relations such as equal signs are aligned.

```
\[
\begin{align*}
 f(x) & = (x+a)(x+b) \\
      & = x^2 + (a+b)x + ab
\end{align*}
\]
```

produces:

$$\begin{aligned}
 f(x) &= (x+a)(x+b) \\
      &= x^2 + (a+b)x + ab
\end{aligned}$$

```
\[
\begin{align*}
 f(x) & = (x+a)(x+b)        \\
      & = x^2 + (a+b)x + ab
\end{align*}
\]

\[
\begin{align*}
 f(x)  & = a x^2 + b x + c \quad \text{ and }  &   g(x)  &= d x^3 \\
 f'(x) & = 2 a x + b       \quad \text{ and }  &   g'(x) &= 3 d x^2
\end{align*}
\]
```

produces:

$$\begin{aligned}
 f(x) &= a x^2 + b x + c \quad \text{ and } &   g(x) &= d x^3 \\
 f'(x) &= 2 a x + b   \quad \text{ and } &   g'(x) &= 3 d x^2
\end{aligned}$$

```
\[
\begin{align*}
 f(x)  & = a x^2 + b x + c \quad \text{ and }  &   g(x)  &= d x^3 \\
 f'(x) & = 2 a x + b       \quad \text{ and }  &   g'(x) &= 3 d x^2
\end{align*}
\]
```

The `\begin{cases}...\end{cases}` environment allows the writing of piece-wise functions:

```
\[
 |x| =
  \begin{cases}
   x     & \text{if } x \geq 0 \\
   -x    & \text{if } x < 0
  \end{cases}
\]
```

produces:

$$|x| = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{if } x < 0 \end{cases}$$

Greek Letters

- `\alpha` produces α.
- `\beta` produces β.
- `\gamma` produces γ.
- `\delta` produces δ.
- `\epsilon` produces ϵ.
- `\zeta` produces ζ.
- `\eta` produces η.
- `\theta` produces θ.
- `\mu` produces μ.
- `\nu` produces ν.
- `\xi` produces ξ.
- `\pi` produces π.
- `\rho` produces ρ.
- `\sigma` produces σ.
- `\tau` produces τ.
- `\upsilon` produces υ.
- `\phi` produces ϕ.
- `\chi` produces χ.
- `\psi` produces ψ.
- `\omega` produces ω.

Computing and Graphics Basics

R References

Selected R books and web-based references:

1. R Cookbook
2. Using R for Introductory Statistics
3. A (Very) Short Introduction to R

Calculating Probabilities for Discrete and Continuous Distributions

Goal: to calculate either the simple or the cumulative probability associated with a discrete random variable X.

For a simple probability, $P(X = x)$, we use the PDF function. All built-in probability distributions have a PDF function whose name is "d" prefixed to the distribution name (or rather abbreviation). For example, `dbinom()` for the Binomial distribution, `dpois()` for the Poisson distribution, and `dnorm()` for the Normal distribution.

For a cumulative probability, $P(X \leq x)$, we use the CDF function. All built-in probability distributions have a CDF function whose name is "p" prefixed to the distribution name; thus, `pbinom()` is the CDF function for the Binomial distribution, `ppois()` for the Poisson distribution, and `dnorm()` for the Normal distribution.

In each case, we need to provide the value of the random variable and specify all parameters that determine the distribution. One can see which parameters must be specified by looking up the function signature using `help()` or `?`. For example, typing `?pbinom` shows the help page for this function.

Suppose we have a Binomial random variable $X \sim B(n = 10, p = 0.5)$ over 10 Bernoulli trials, where each trial has a success probability of 0.5. We can calculate the probability of observing $X = 7$ by calling the PDF function `dbinom`:

```
dbinom(7, size=10, prob=0.5)
```

```
## [1] 0.1171875
```

Suppose we have a Normal r.v. $X \sim N(\mu = 15, \sigma = 5)$. We can calculate the probability $F_X(25) = P(X \leq 25)$ by calling the CDF function `pnorm`:

```
pnorm(25,mean=15,sd=5)
```

```
## [1] 0.9772499
```

All these functions are vectorized, which means that we can compute a vector of probabilities for a given vector of values. For example, if we need both probabilities $P(X \leq 25)$ and $P(X \leq 15)$, we can make the call:

```
pnorm(c(15,25),mean=15,sd=5)
```

```
## [1] 0.5000000 0.9772499
```

and if we want to compute the probability for an interval, say $P(15 < X \leq 25) = P(X \leq 25) - P(X \leq 15)$, then we can just take the difference using `diff()` of the above vector:

```
diff(pnorm(c(15,25),mean=15,sd=5))
```

```
## [1] 0.4772499
```

Converting Probabilities to Quantiles

Given a probability p and a distribution, you want to determine the corresponding quantile for p, that is the value x such that $F_X(x) = P(X \leq x) = p$.

Every built-in distribution includes a quantile function or inverse CDF that converts probabilities to quantiles. The function's name is "q" prefixed to the distribution name. For example, `qnorm` is the quantile or inverse CDF function F_X^{-1} for the Normal distribution:

$$F_X^{-1}(p) = \text{qnorm}(p, \mu, \sigma), \text{ for } X \sim N(\mu, \sigma)$$

The first argument of the quantile function is the probability. The remaining arguments are the distribution's parameters; for example mean and standard deviation for the Normal distribution, etc.

Suppose we want to compute the quantile x_0 (value of X) for which $F_X(x_0) = P(X \leq x_0) = 0.05$ and the r.v. $X \sim N(\mu = 100, \sigma = 15)$, then we make the following call:

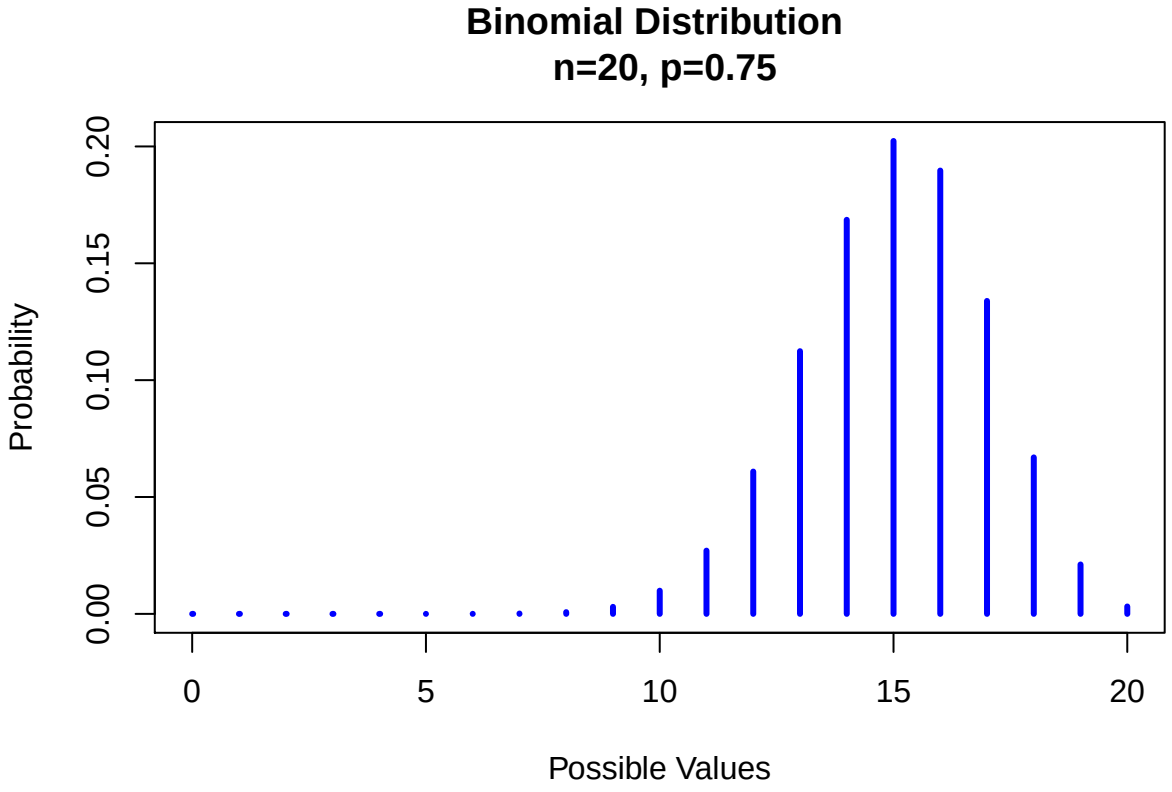
```
qnorm(0.05, mean=100, sd=15)
```

```
## [1] 75.3272
```

Plotting a Density Function

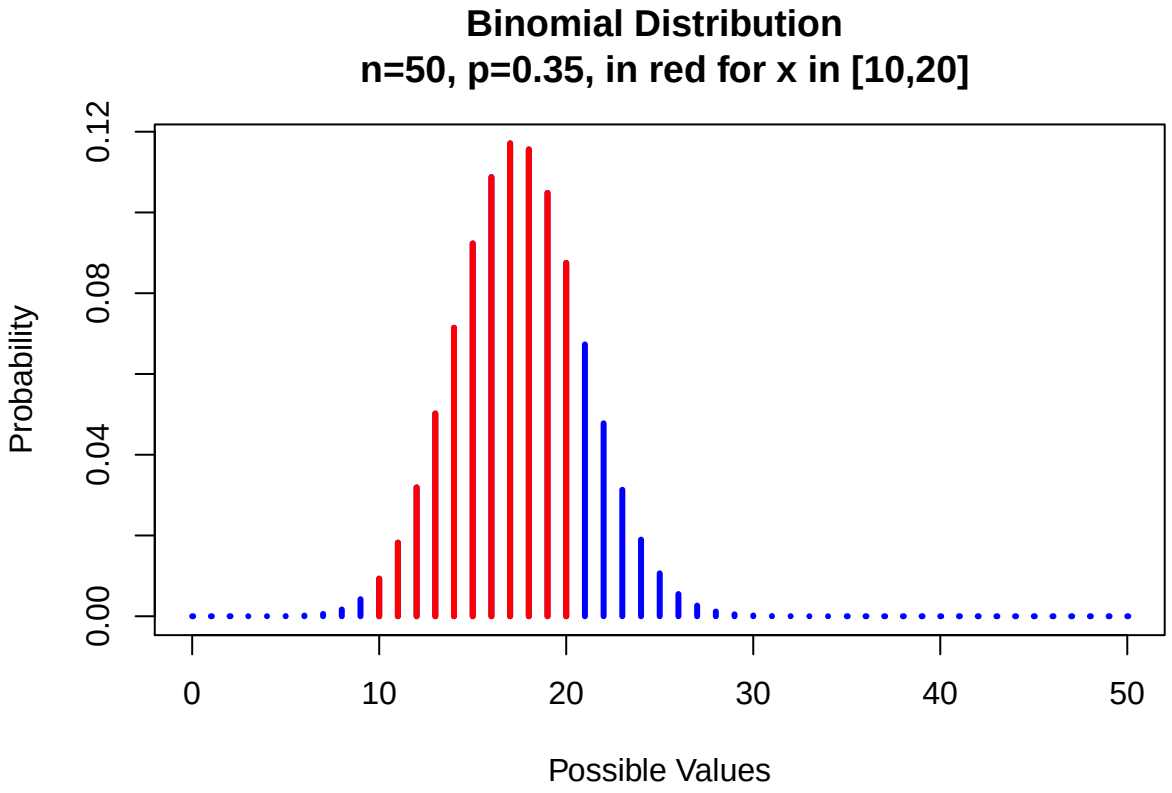
For a discrete r.v. we need to compute the vector of probabilities corresponding to the vector of all possible values of the random variable, and plot the pairs as a histogram style plot.

```
n<-20 # number of Bernoulli trials
p<-0.75 # prob. of success
x<-0:n # possible values
y<-dbinom(x,size=n,prob=p) # vector of prob. for the vector x
plot(x,y,type="h",col="blue",lwd=3,main="Binomial Distribution \n n=20, p=0.75",
     ylab="Probability",xlab="Possible Values")
```



We can color differently a subset of probabilities corresponding to a given subset of possible value of the r.v.

```
n<-50 # number of Bernoulli trials
p<-0.35 # prob. of success
x<-0:n # vector of all possible values
y<-dbinom(x,size=n,prob=p) # vector of prob. for the vector x
plot(x,y,type="h",col="blue",lwd=3,
     main="Binomial Distribution \n n=50, p=0.35, in red for x in [10,20]",
     ylab="Probability",xlab="Possible Values")
x.interval<-x[10<=x & x<=20] # subset from x satisfying the condition
y.interval<-y[10<=x & x<=20] # subset from y satisfying the condition
points(x.interval,y.interval,type="h",lwd=3,col="red") # add points to the plot
```

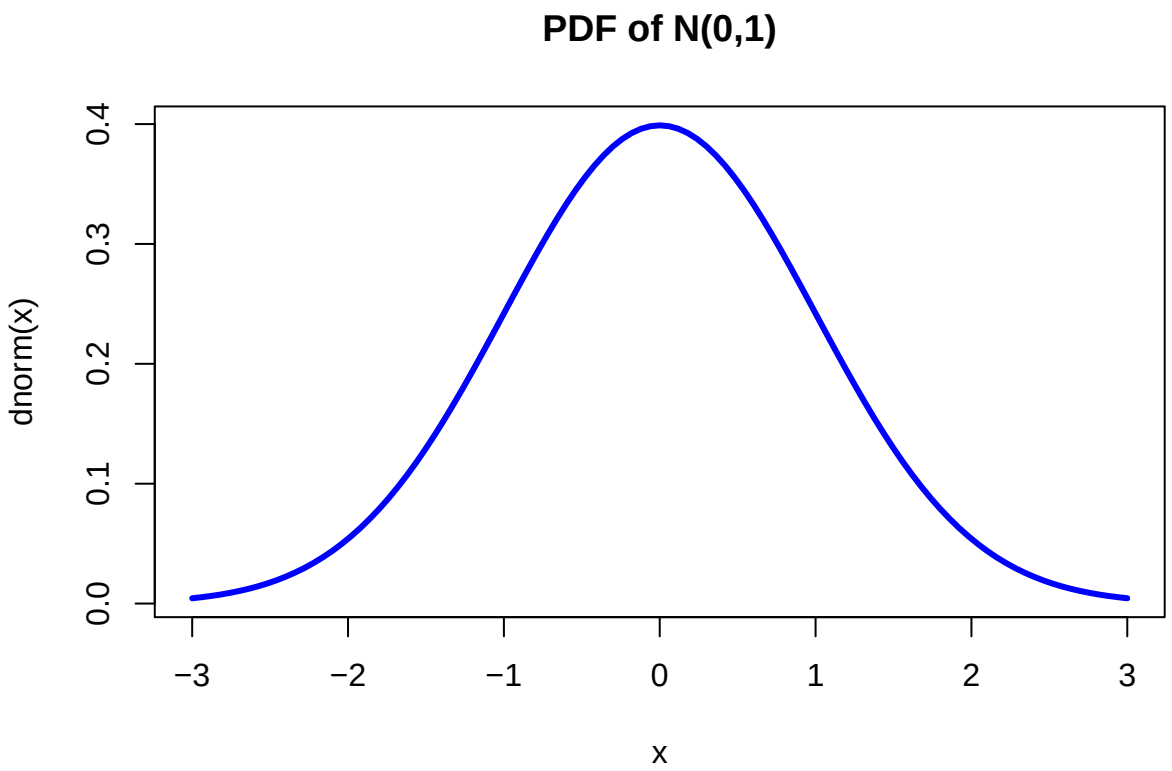


The sum of all probabilities for all possible values should be 1, and we can confirm this by the code `sum(y)`, which gives 1 indeed. The probability $P(10 \leq X \leq 20)$ we get by calling `sum(y.interval)`, which gives 0.8072472.

It's easier to plot the PDF for continuous distributions.

The code snippet below plots the standard normal distribution over the 6-sigma interval $(-3, 3)$.

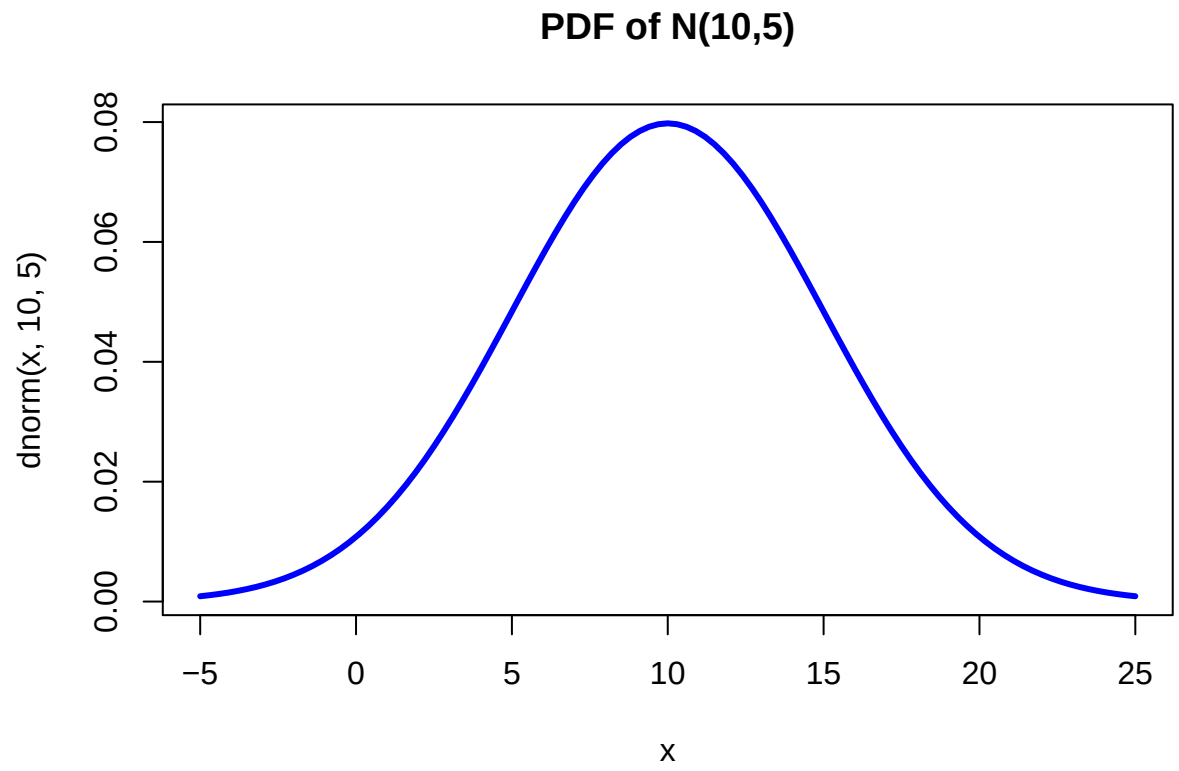
```
curve(dnorm,-3,3,main="PDF of N(0,1)",col="blue",lwd=3)
```



What if we want to plot the PDF of $N(\mu = 10, \sigma = 5)$ random variable?

We can bind the additional arguments by making the call `dnorm(x,10,5)` inside `curve` and plot over the 6-sigma interval:

```
curve(dnorm(x,10,5),-5,25,main="PDF of N(10,5)",col="blue",lwd=3)
```



Density Plots with Shaded Regions

We can create a density plot with a shaded region under the pdf, a useful way to visualize probabilities interpreted as areas under the pdf, in the case of continuous distributions.

We start by first plotting the density and then shading a region with the `polygon()` function. The polygon function draws a series of lines around the highlighted area and fills it.

We can draw the standard Normal density curve by first creating vector of values for the x and y coordinates of the PDF.

```
x <- seq(from=-3, to= 3, length.out=200) # a vector of size 200 with numbers in [-3,3]
y <- dnorm(x) # y is a vector of the same size as x, with values from the PDF
```

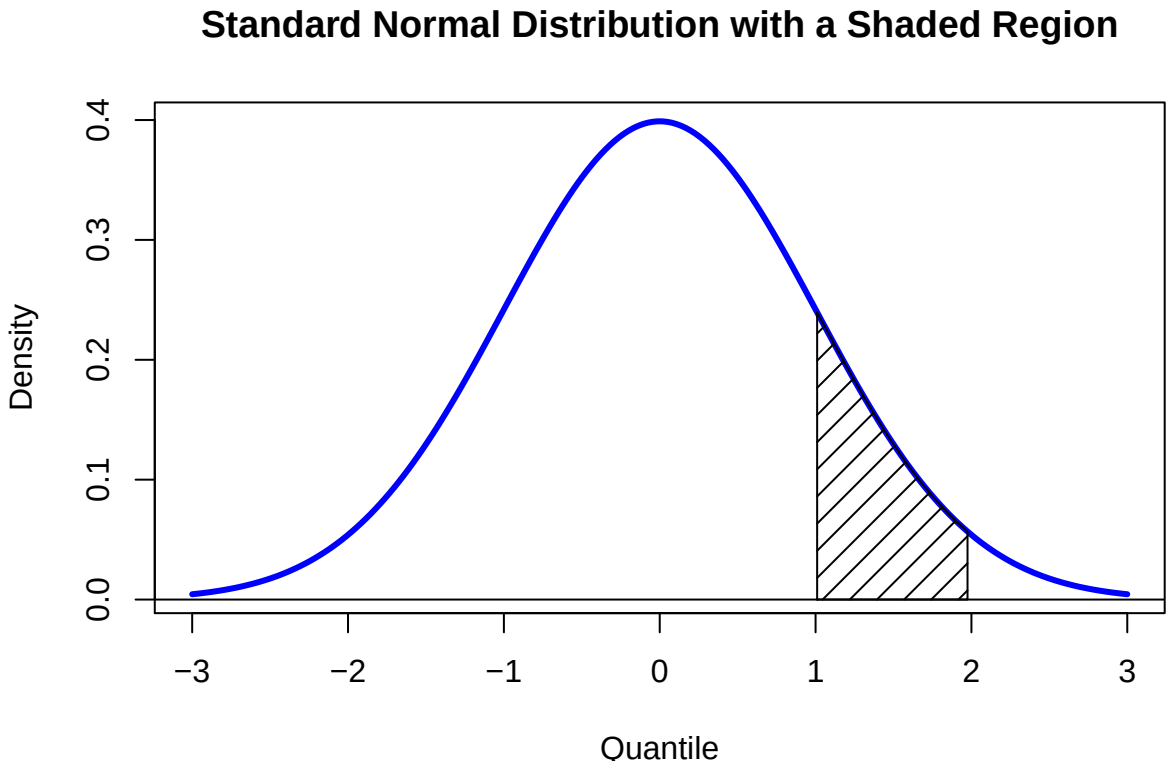
Next, we define the region of interest by a series of line segments. The line segments, in turn, are defined by a series of (x, y) coordinates. The polygon function will connect the first and last (x, y) points to close the polygon:

```
# The polygon follows the density curve where 1 <= x <= 2
region.x <- x[1 <= x & x <= 2] # subset from x satisfying the condition
region.y <- y[1 <= x & x <= 2] # subset from y satisfying the condition
# We add initial and final segments, which drop down to the Y axis
region.x <- c(region.x[1], region.x, tail(region.x,1))
region.y <- c(0,region.y,0)
```

Note that `tail(region.x,1)` gives the last element in the vector `region.x`.

Finally, we call `plot` and `polygon()` to plot the PDF and the boundary of the region to be shaded:

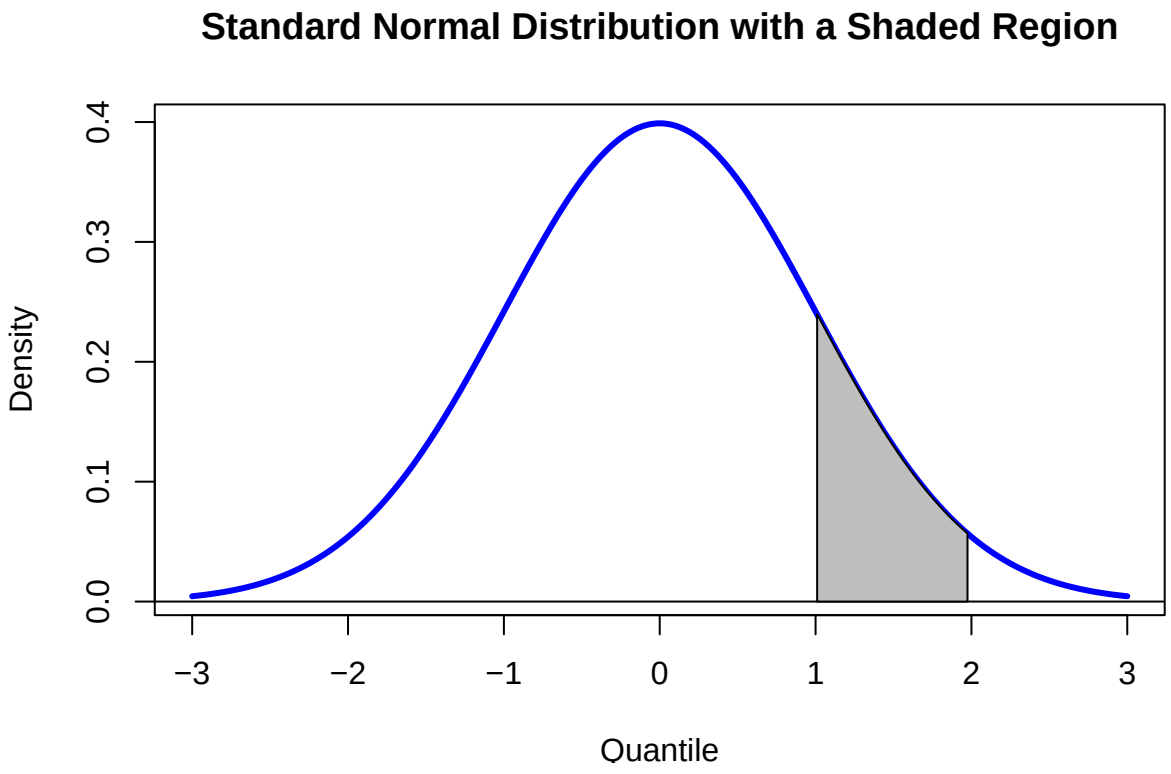
```
plot(x, y, main="Standard Normal Distribution with a Shaded Region",
     type='l', ylab="Density", xlab="Quantile",lwd=3,col="blue")
abline(h=0) # adds the x-axis to the plot
polygon(region.x, region.y, density=10) # the polygon is added to the active plot
```



By default, `polygon` does not fill the region. Setting `density=10` turns on filling with thin lines at a 45° angle.

To fill with some color instead, we would set `density=-1` and set `col` to the desired color, say gray:

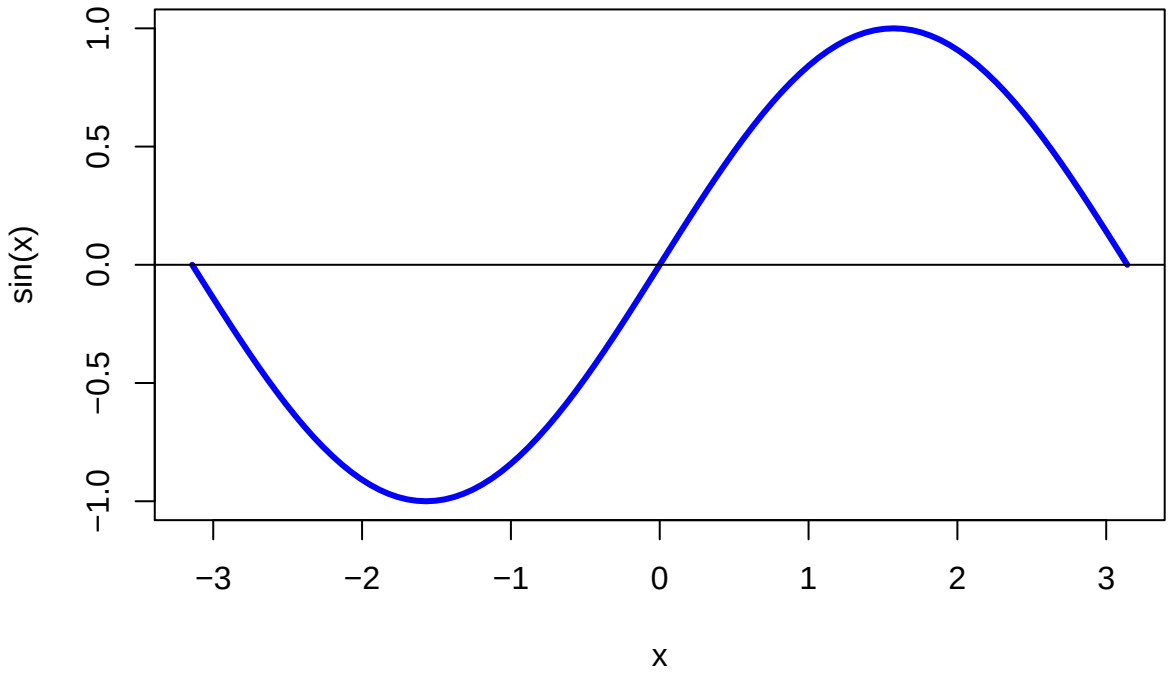
```
plot(x, y, main="Standard Normal Distribution with a Shaded Region",
     type='l', ylab="Density", xlab="Quantile",lwd=3,col="blue")
abline(h=0) # adds the x-axis to the plot
polygon(region.x, region.y, density=-1, col="gray")
```



Graphing Functions

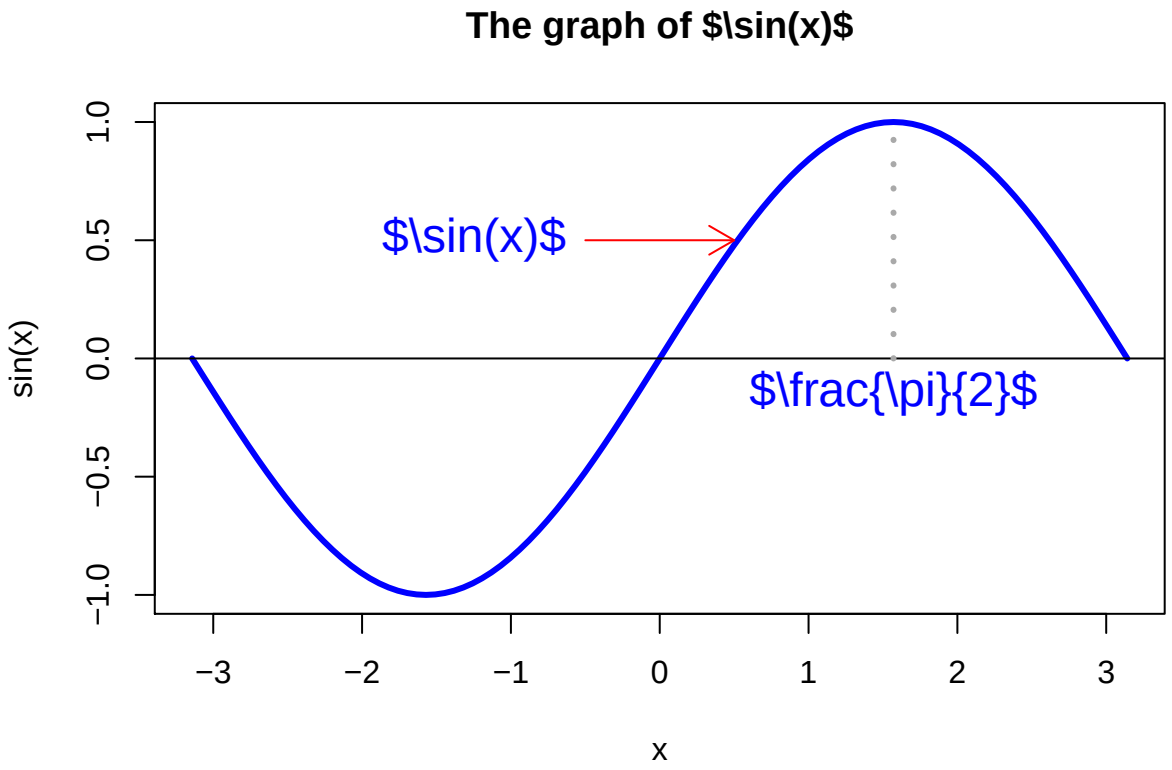
Any built-in R function can be plotted using the name of the function and `curve()`. For example, here is a plot of $y(x) = \sin(x)$ over the interval $[-\pi, \pi]$.

```
curve(sin,-pi,pi,lwd=3,main="The graph of sin(x)",col="blue") # plot of sin(x) over [-pi,pi]
abline(h=0) # add x-axis
```



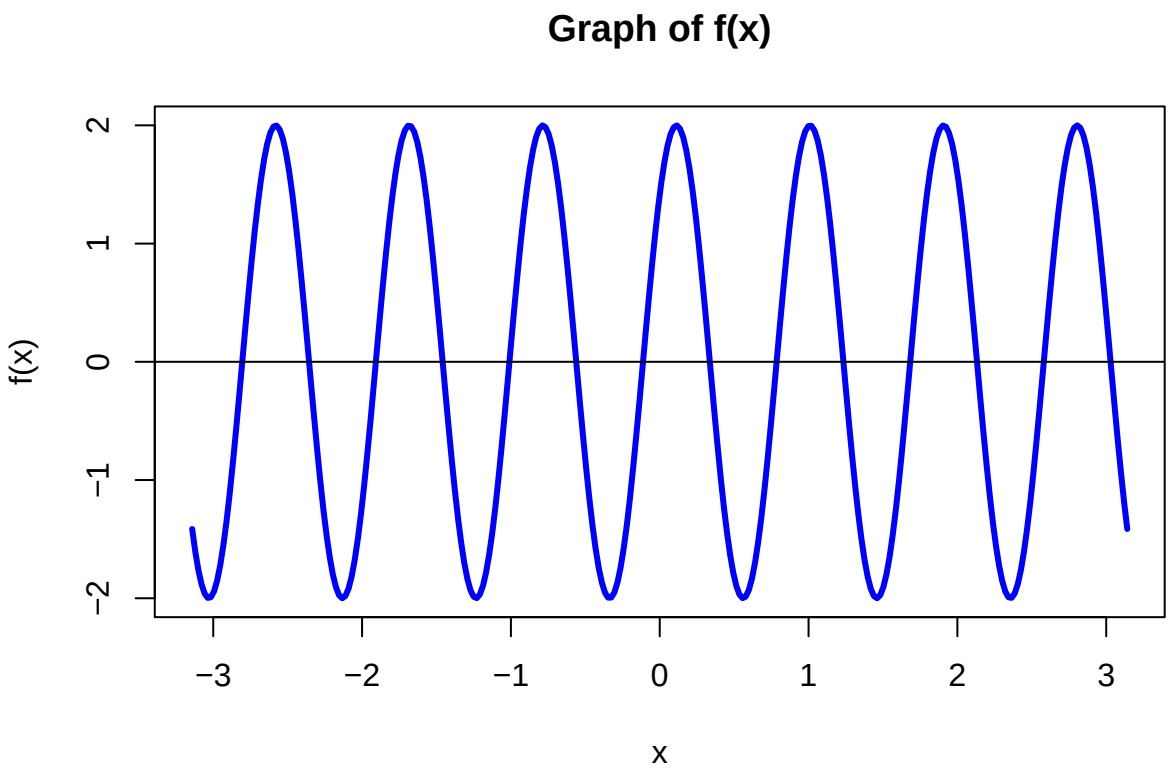
We can add text and draw segments and arrows using the R functions `text()`, `segments()` and `arrows()`. We can also insert LaTeX expressions into the plot but we have to set the device parameter `dev='tikz'` in the R Chunk.However, this will work only if we are generating a pdf document since the 'tikz' device generates a pdf rather than png file, like the other plots embedded in the html. Interestingly, Safari displays also the embedded pdf file, generated by the 'tikz' device, but Chrome and Firefox fail to display the pdf file linked in the html.

```
curve(sin,-pi,pi,lwd=3,main="The graph of $\\sin(x)$",col="blue") # plot of sin(x) over [-pi,pi]
abline(h=0) # add x-axis
text(pi/2,0,"$\\frac{\\pi}{2}$",pos=1,col="blue",cex=1.5)
# pos=1 is below, 2=to the left,3=above,4=to the right
segments(pi/2,0,pi/2,1,lty=3,col="dark gray",lwd=3) # lty=3 is dotted, lty=2 is dashed
arrows(-0.5,0.5,0.5,0.5,length = 0.15, angle = 30,code = 2, col = "red",lwd=1)
text(-0.5,0.5,"$\\sin(x)$",pos=2,col="blue",cex=1.5)
```



If we want to plot a custom function, then we have to first define our function as an R function. For example, let us plot the graph of $f(x) = 2\sin(7x + \frac{\pi}{4})$ over the interval $[-\pi, \pi]$.

```
f<-function(x) 2*sin(7*x+pi/4)
curve(f,-pi,pi,lwd=3,n=300,main="Graph of f(x)",col="blue")
abline(h=0) # add x-axis
```



Application: Adding Harmonics of the same frequency

Let's plot the curve $f(t)$ that results from adding two harmonic waves of the same frequency:

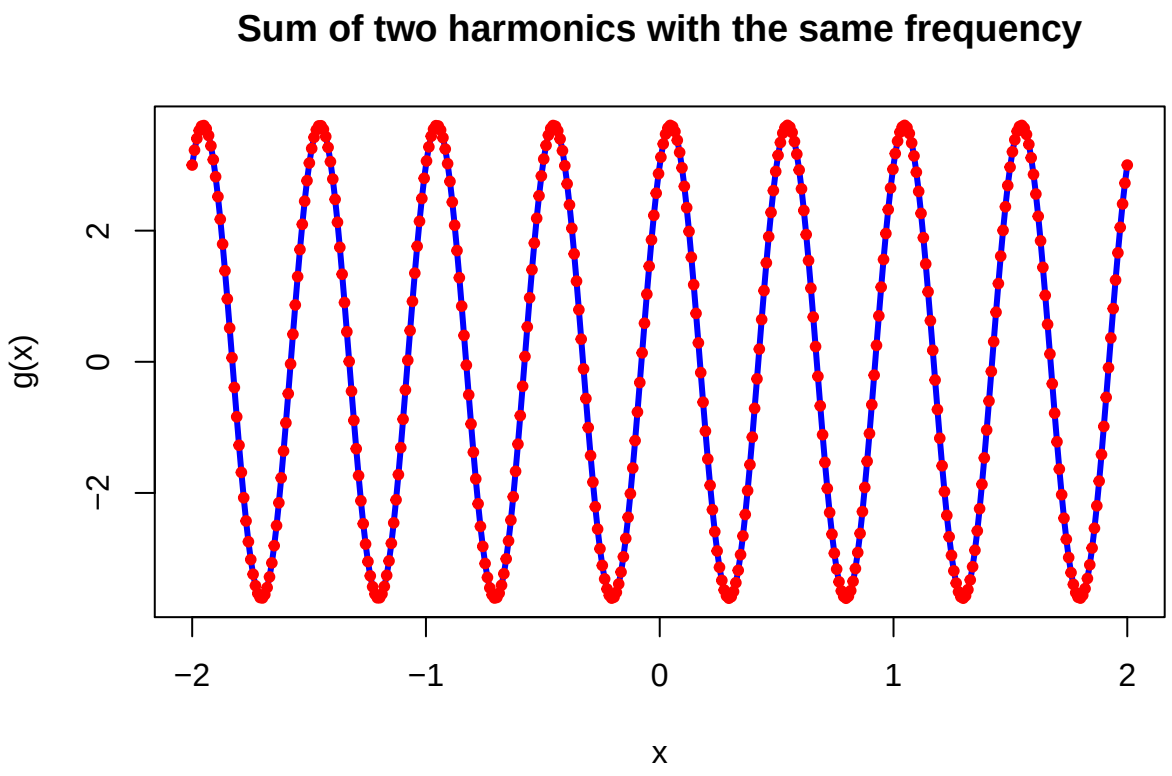
$$f(t) = A\sin(2\pi nt) + B\cos(2\pi nt)$$

It's not immediately obvious that $f(x)$ is another harmonic wave with the same frequency of n cycles per unit time but with a different amplitude C and phase ϕ:

$$f(t) = C\sin(2\pi nt + \phi)$$

It turns out that $C = \sqrt{A^2 + B^2}$ and $\phi = \arctan(\frac{B}{A})$.

```
A<-2; B<-3; n<-2
C<-sqrt(A^2+B^2); phi<-atan(B/A)
f<-function(x)A*sin(2*pi*n*x)+B*cos(2*pi*n*x)
g<-function(x)C*sin(2*pi*n*x+phi)
curve(g,-2,2,col="blue",n=400,lwd=3,main="Sum of two harmonics with the same frequency")
curve(f,-2,2,col="red",n=400,type="p",pch=20,lwd=1,add=T) # plot is added to the first plot
```



Application: Monte Carlo Simulations for Estimating Integrals

Using random sampling, we can estimate difficult integrals with high accuracy in a couple of lines of R code. For example, let's consider the following integral:

$$I = \int_0^\infty \sin(x) e^{-x^2/2} dx$$

The idea is to represent the integral in terms of the expected value of a function of some appropriate random variable.

$$I = \sqrt{2\pi} \int_0^\infty \sin(x)\, f_X(x) dx = \sqrt{2\pi} E[\sin(X) \mathbb{1}(X > 0)],$$

where $X \sim N(0,1)$ and its PDF is $f_X(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$, also $\mathbb{1}(X > 0)$ is the indicator function of the event $(X > 0)$.

All we need to do is simulate a large sample from $N(0,1)$, using the random number generator `rnorm` for the Normal distribution, and take the `mean()` of the appropriate function applied to this sample.

```
x<-rnorm(1e6) # a sample of size 10^6 from N(0,1)
sqrt(2*pi)*mean(sin(x)*(x>0)) # estimate for I
```

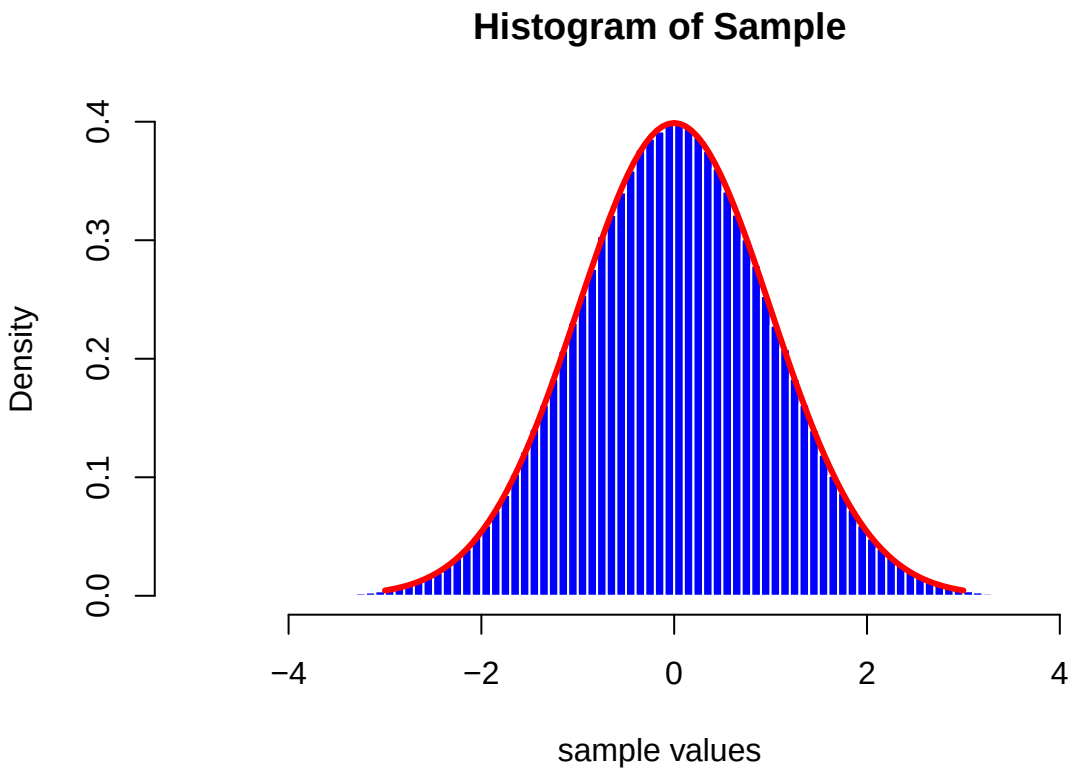
```
## [1] 0.7252405
```

Using **Wolfram Alpha** gives the following result based on deterministic numerical schemes:

```
integral_0^infinity sin(x) e^(-x^2/2) dx = 0.724778
```

We can confirm that a large sample generated from $N(0,1)$ using `rnorm` indeed comes from the Standard Normal Distribution by creating the density histogram of this sample and plotting the theoretical PDF on top of it to make sure they match. A density histogram can be created using the `hist()` function with the `freq` argument set to `FALSE`, that is `freq=FALSE`.

```
hist(rnorm(1e6),breaks=100,freq=FALSE,col="blue",border="white",
     main="Histogram of Sample",xlab="sample values")
curve(dnorm,-3,3,col="red",lwd=3,add=TRUE)
```



We can generate samples of any size from any other distribution implemented in R, and all random number generators' names follow the same convention `rname`, where `name` stands for the name of the distribution. You can see the arguments that the function takes by typing `?rname` at the RStudio Console.

Exploratory Data Analysis (Optional)

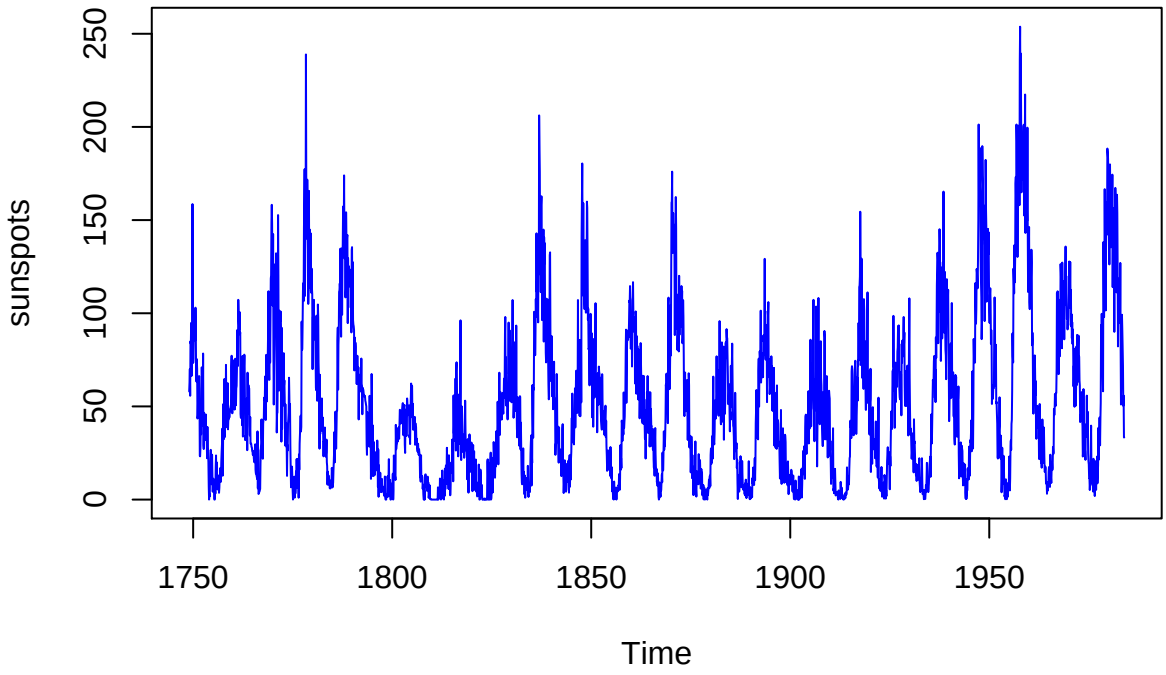
Analysis of Datasets Available in Base R

Base R has datasets that can be used for exploratory data analysis.

Use the code `library(help="datasets")` to see the complete list in `datasets`.

For example, `sunspots` is one of the datasets, and we can load the data using the code `data(sunspots)`.

```
data(sunspots) # sunspots is a time-series data class in R
plot(sunspots,col="blue") # plots the time-series
```



Analysis of an External Dataset

One can load into R datasets in different formats. The most common formats are .txt, .csv and .dat (R data files). The process of importing a data file into RStudio can be automated by using the `Import Dataset` button.

Alternatively,datasets can be read into R using `read.table()` for text files or `read.csv()` for Excel spreadsheets.

We have a data file called `college.txt` as a text file, where the columns (variables) are separated by tabs, and we want to import the data into R for further data analysis.

```
data <- read.table("college.txt",header=T,sep="\t") # load data file
dim(data) # dim of original data
```

```
## [1] 260  11
```

```
names(data) # see the names of all variables
```

```
##  [1] "School"        "Enrollment"    "Tier"          "Retention"
##  [5] "Grad.rate"     "Pct.20"        "Pct.50"        "Full.time"
##  [9] "Top.10"        "Accept.rate"   "Alumni.giving"
```

Each column represents a variable, so this data has 11 variables, and each element in a column is a different observation from this variable, representing a college. We can see the beginning of this data by using `head()`. We are interested only in the first 4 columns of this data, so we can extract them using the `subset()` function.

```
college<-subset(data,select=c(School,Tier,Retention,Grad.rate))
dim(college)
```

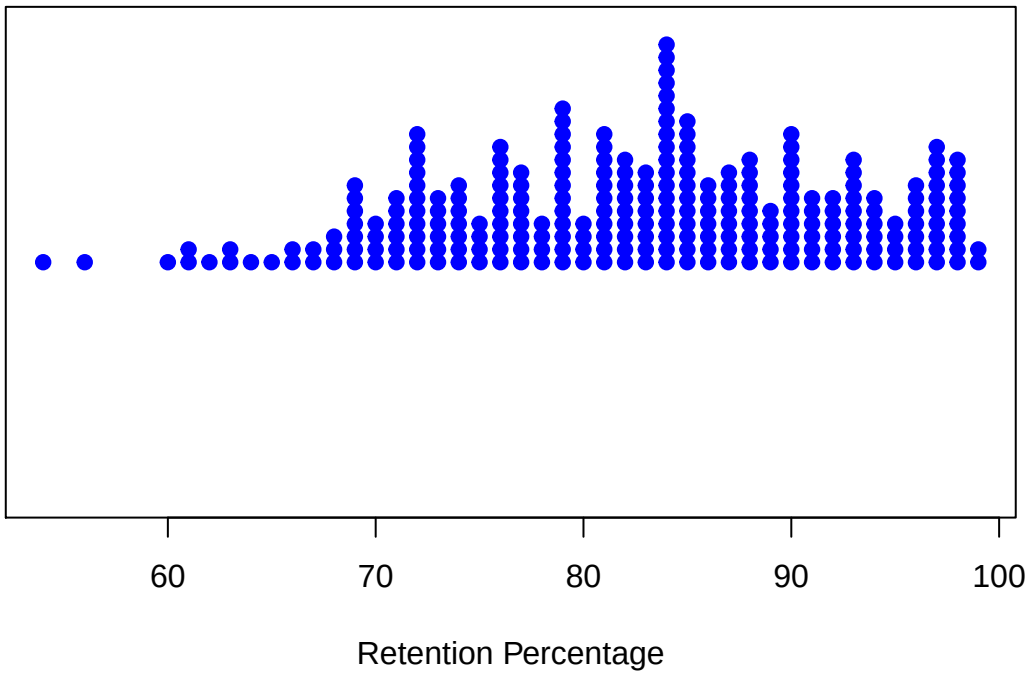
```
## [1] 260   4
```

```
head(college) # show the beginning of college data
```

```
##      School Tier Retention Grad.rate
## 1   Harvard    1        97        98
## 2 Princeton    1        98        96
## 3      Yale    1        99        97
## 4  Cal Tech    1        98        88
## 5       MIT    1        98        94
## 6  Stanford    1        98        94
```

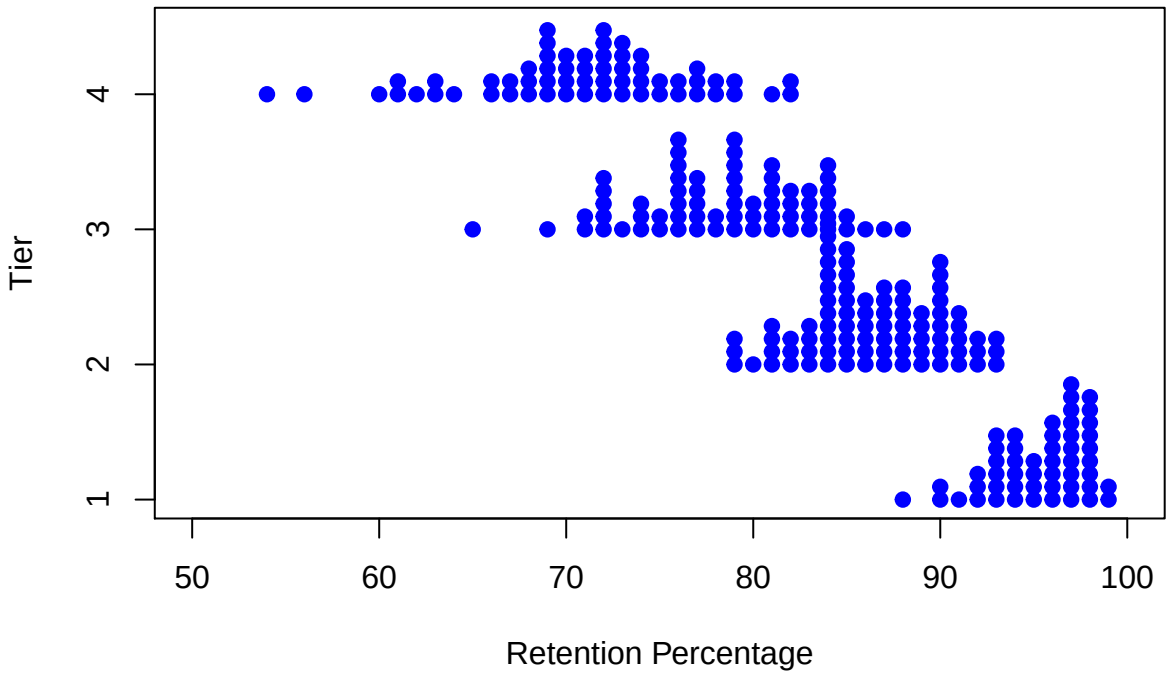
We can easily create a stacked stripchart as a histogram-style chart to visualize the data. We can extract a variable from the data, that is an entire column, by using the `$` operator. For example, `college$Retention` would extract the `Retention` variable from `college`.

```
stripchart(college$Retention,method="stack",pch=19,xlab="Retention Percentage",col="blue")
```



We can also create several stripcharts at once, conditioned on another variable. For example, we can condition the `Retention` variable on the `Tier` variable.

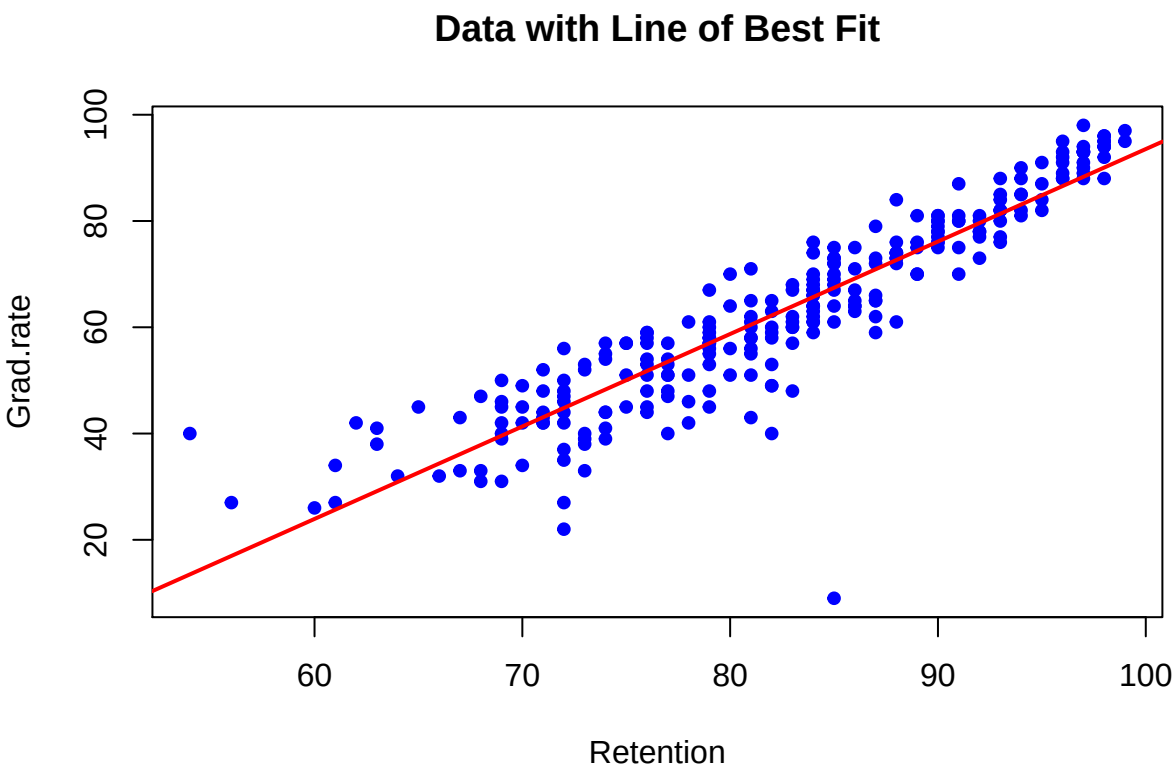
```
stripchart(Retention ~ Tier,method="stack",pch=19,col="blue",xlab="Retention Percentage",
           ylab="Tier",xlim=c(50,100),data=college)
```



Linear Regression Modeling

We consider a linear model, which attempts to explain the graduation rate using the retention rate as a predictor. In other words, we want to fit a linear model to the retention and graduation rates, using the retention rate as the explanatory variable and the graduation rate as the response variable.

```
attach(college) # make the variables in college available
plot(Retention,Grad.rate,pch=20,col="blue",lwd=2,main="Data with Line of Best Fit")
model<-lm(Grad.rate ~ Retention, data=college)
abline(model,col="red",lwd=2)
```



The fitted coefficients for the regression line can be extracted as a vector from the model object using `coef(model)` or `model$coef`.

```
coef(model) # calling the coef() method on the model object
```

```
## (Intercept)   Retention
##  -80.487066    1.740261
```

```
model$coef # extracting the coef component from the model object
```

```
## (Intercept)   Retention
##  -80.487066    1.740261
```

Thus, the regression line is given by the equation:

$$\text{Graduation Rate} = -80.4870656 + 1.7402607 \times \text{Retention Rate}$$

which was typeset by using the expression

```
$$ \text{Graduation Rate} = `r coef(model)[1]` +
  `r coef(model)[2]` \times \text{Retention Rate} $$
```

in the source .Rmd file.

We can easily compute the correlation between Retention and Graduation Rate by using the code

`cor(Retention, Grad.rate,use="complete.obs")`,

which gives

$$\text{cor}(\text{Retention}, \text{Grad.rate}) = 0.914823$$

Note that the optional parameter `use` is set to `"complete.obs"` since we have missing data (NAs).

Once we have the fitted model, we can make predictions using the `predict()` function.

For example, we predict a graduation rate of 58.73% for a retention rate of 80%, which was computed by using the code

```
`r round(predict(model,newdata=data.frame(Retention=80)),2)` # round results to 2 digits
```

Often, it is important to create a plot of the residuals of the linear regression fit. The residuals are contained in the fitted linear model, that is, the `model` object, and can be extracted by `model$residuals`.

A Sneak Peek Into Interactive Web-based Graphics

3D Plots

The **threejs** package includes a 3D scatterplot and 3D globe. You can directly manipulate the scatterplot below with the mouse.

3D Networks

The **networkD3** package provides tools for creating D3 JavaScript network graphs from R.