3-plotting-1

March 30, 2024

1 3. Plotting

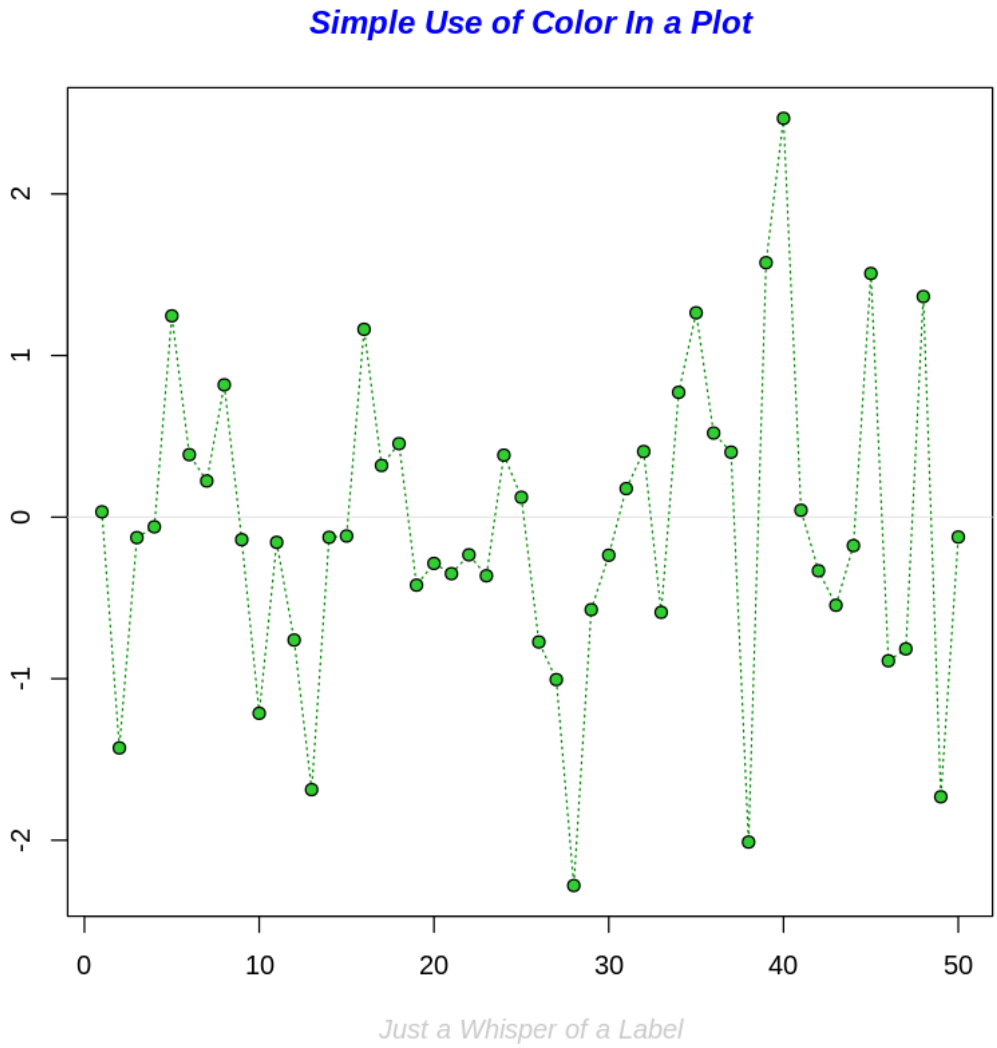
The functions plot(), points(), lines(), text(), mtext(), axis(), identify() etc. form a suite that plots points, lines and text.

To see some of the possibilities that R offers, enter demo(graphics)

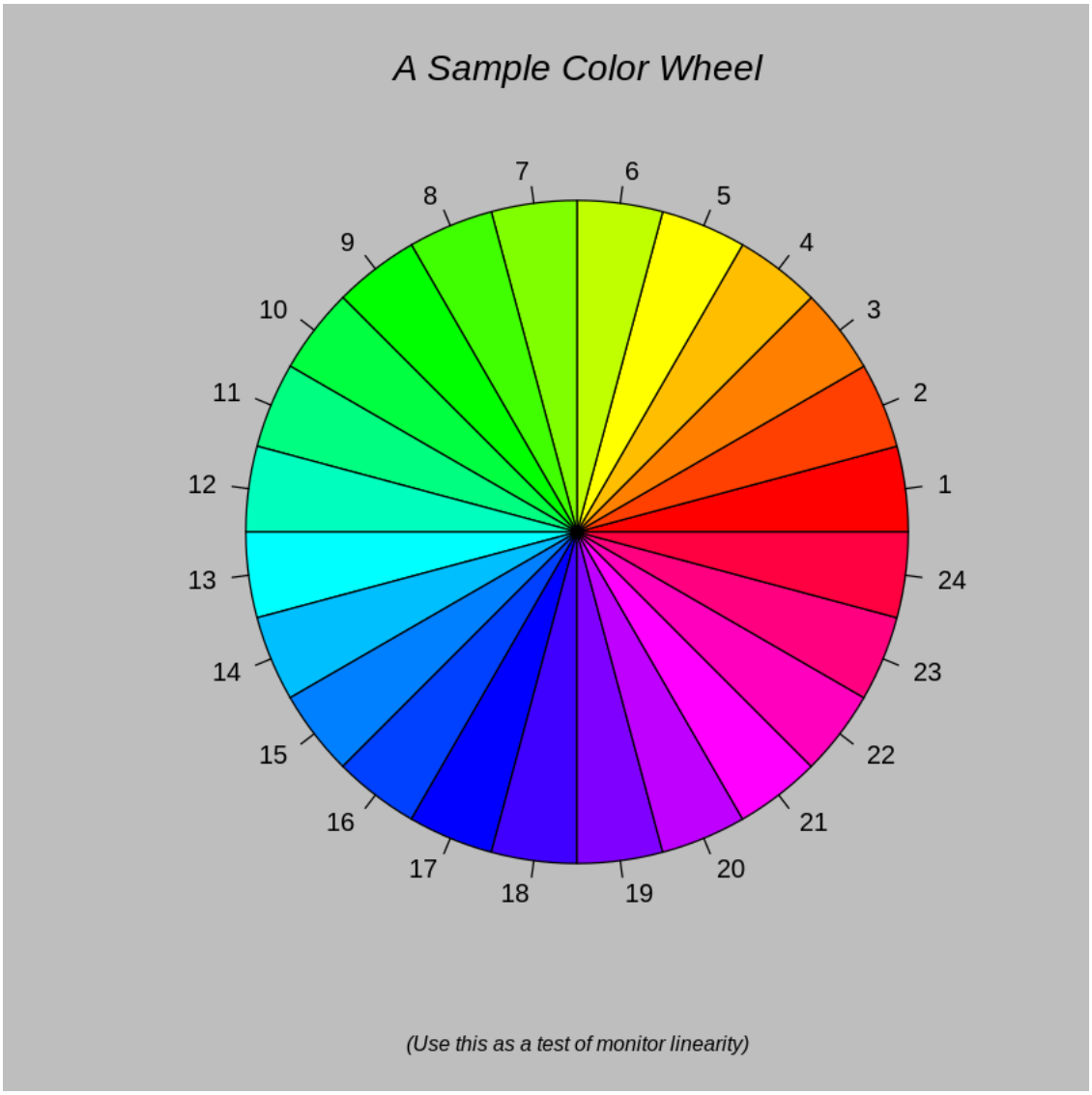
[100]:
```
demo(graphics)
```

```
        demo(graphics)
        ---- ~~~~~~~~

> #  Copyright (C) 1997-2009 The R Core Team
>
> require(datasets)

> require(grDevices); require(graphics)

> ## Here is some code which illustrates some of the differences between
> ## R and S graphics capabilities.  Note that colors are generally specified
> ## by a character string name (taken from the X11 rgb.txt file) and that line
> ## textures are given similarly.  The parameter "bg" sets the background
> ## parameter for the plot and there is also an "fg" parameter which sets
> ## the foreground color.
>
>
> x <- stats::rnorm(50)

> opar <- par(bg = "white")

> plot(x, ann = FALSE, type = "n")

> abline(h = 0, col = gray(.90))

> lines(x, col = "green4", lty = "dotted")

> points(x, bg = "limegreen", pch = 21)
```

```
> title(main = "Simple Use of Color In a Plot",
+       xlab = "Just a Whisper of a Label",
+       col.main = "blue", col.lab = gray(.8),
+       cex.main = 1.2, cex.lab = 1.0, font.main = 4, font.lab = 3)

> ## A little color wheel.       This code just plots equally spaced hues in
> ## a pie chart.       If you have a cheap SVGA monitor (like me) you will
> ## probably find that numerically equispaced does not mean visually
> ## equispaced.  On my display at home, these colors tend to cluster at
> ## the RGB primaries.  On the other hand on the SGI Indy at work the
> ## effect is near perfect.
>
> par(bg = "gray")

> pie(rep(1,24), col = rainbow(24), radius = 0.9)
```

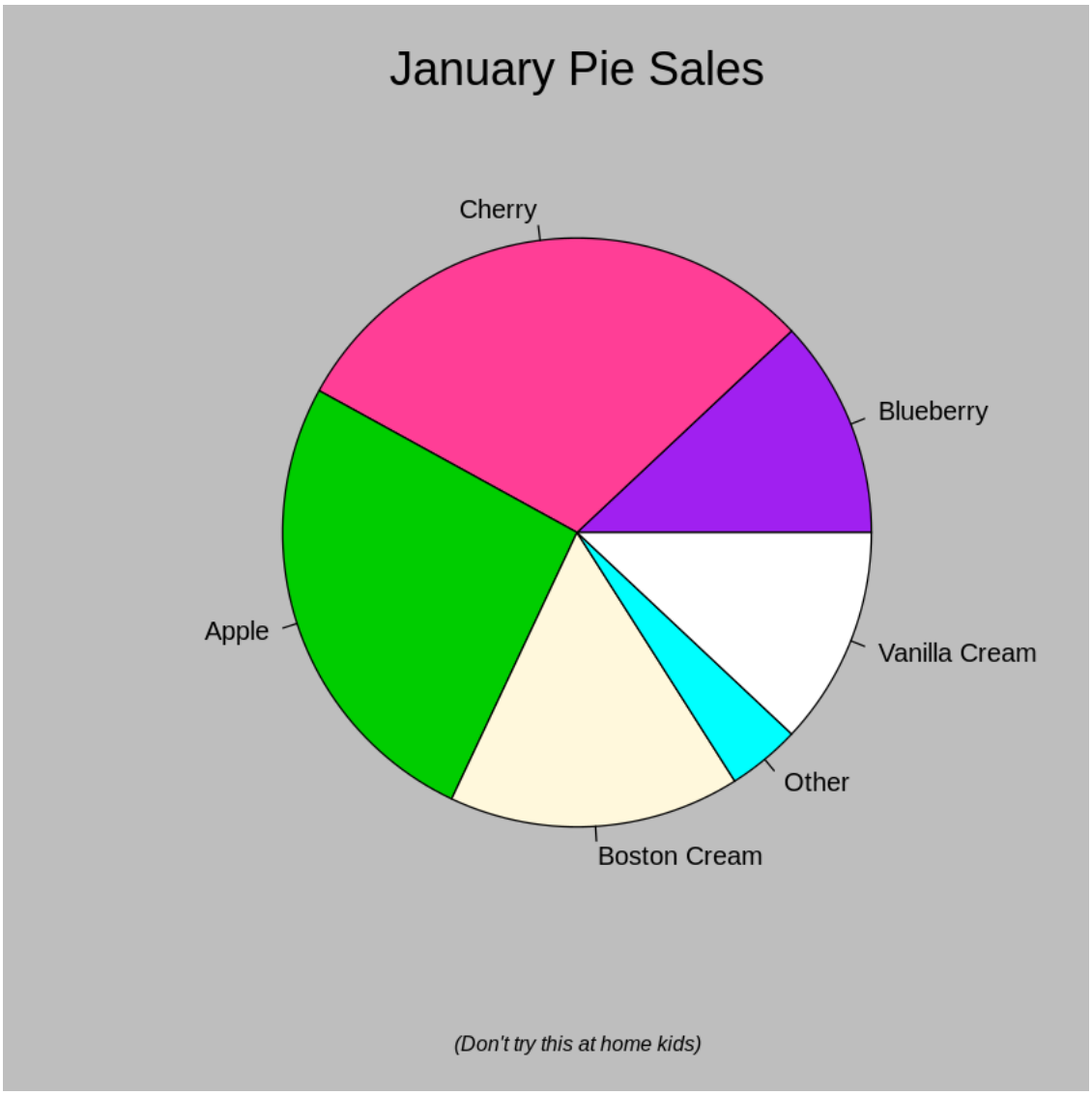


```
> title(main = "A Sample Color Wheel", cex.main = 1.4, font.main = 3)

> title(xlab = "(Use this as a test of monitor linearity)",
+       cex.lab = 0.8, font.lab = 3)

> ## We have already confessed to having these.  This is just showing off X11
> ## color names (and the example (from the postscript manual) is pretty "cute".
>
> pie.sales <- c(0.12, 0.3, 0.26, 0.16, 0.04, 0.12)

> names(pie.sales) <- c("Blueberry", "Cherry",
+                       "Apple", "Boston Cream", "Other", "Vanilla Cream")

> pie(pie.sales,
+     col = c("purple","violetred1","green3","cornsilk","cyan","white"))
```



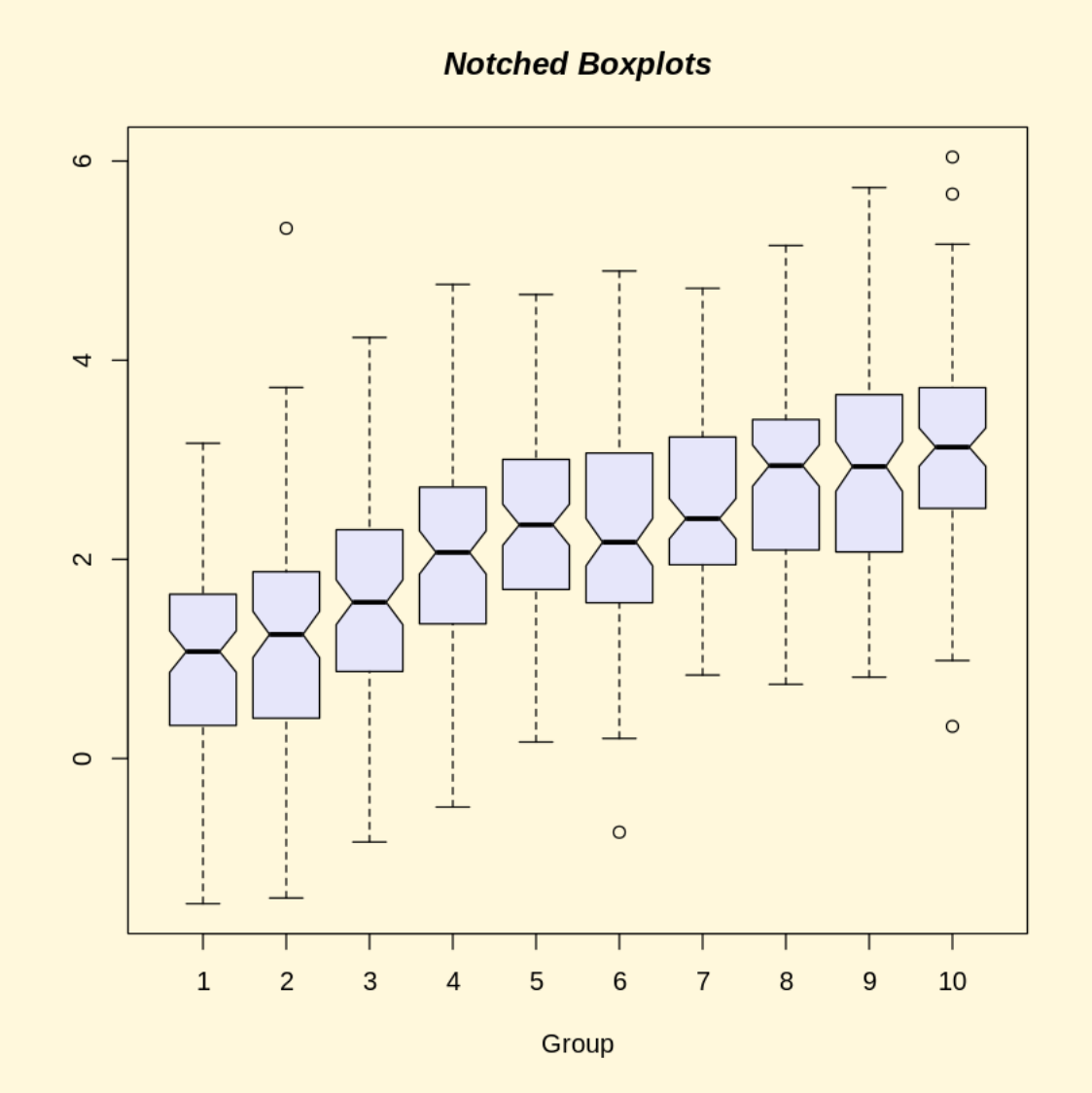

```
> title(main = "January Pie Sales", cex.main = 1.8, font.main = 1)

> title(xlab = "(Don't try this at home kids)", cex.lab = 0.8, font.lab = 3)

> ## Boxplots:  I couldn't resist the capability for filling the "box".
> ## The use of color seems like a useful addition, it focuses attention
> ## on the central bulk of the data.
>
> par(bg="cornsilk")

> n <- 10
```

```
> g <- gl(n, 100, n*100)

> x <- rnorm(n*100) + sqrt(as.numeric(g))

> boxplot(split(x,g), col="lavender", notch=TRUE)
```



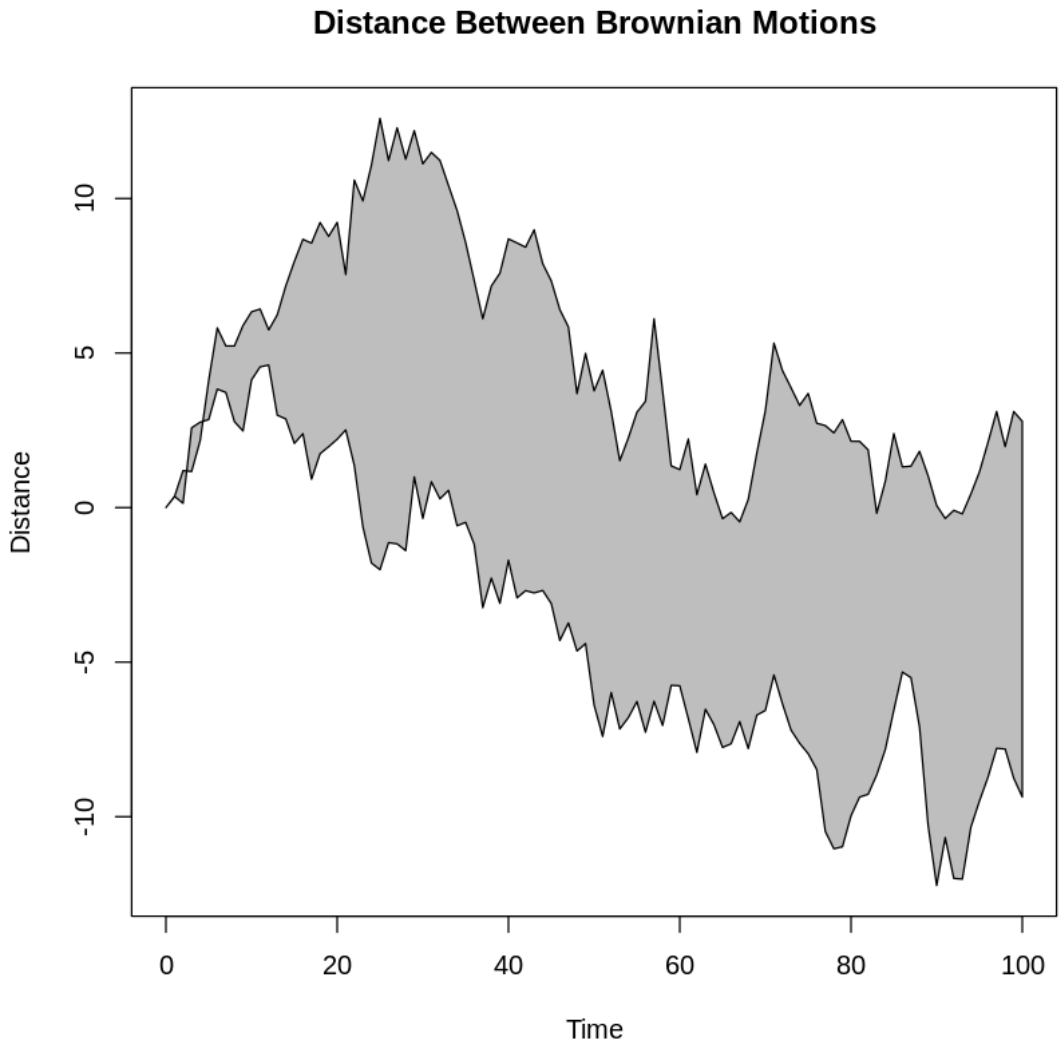

```
> title(main="Notched Boxplots", xlab="Group", font.main=4, font.lab=1)

> ## An example showing how to fill between curves.
>
> par(bg="white")

> n <- 100
```

```
> x <- c(0,cumsum(rnorm(n)))
> y <- c(0,cumsum(rnorm(n)))
> xx <- c(0:n, n:0)
> yy <- c(x, rev(y))
> plot(xx, yy, type="n", xlab="Time", ylab="Distance")
```

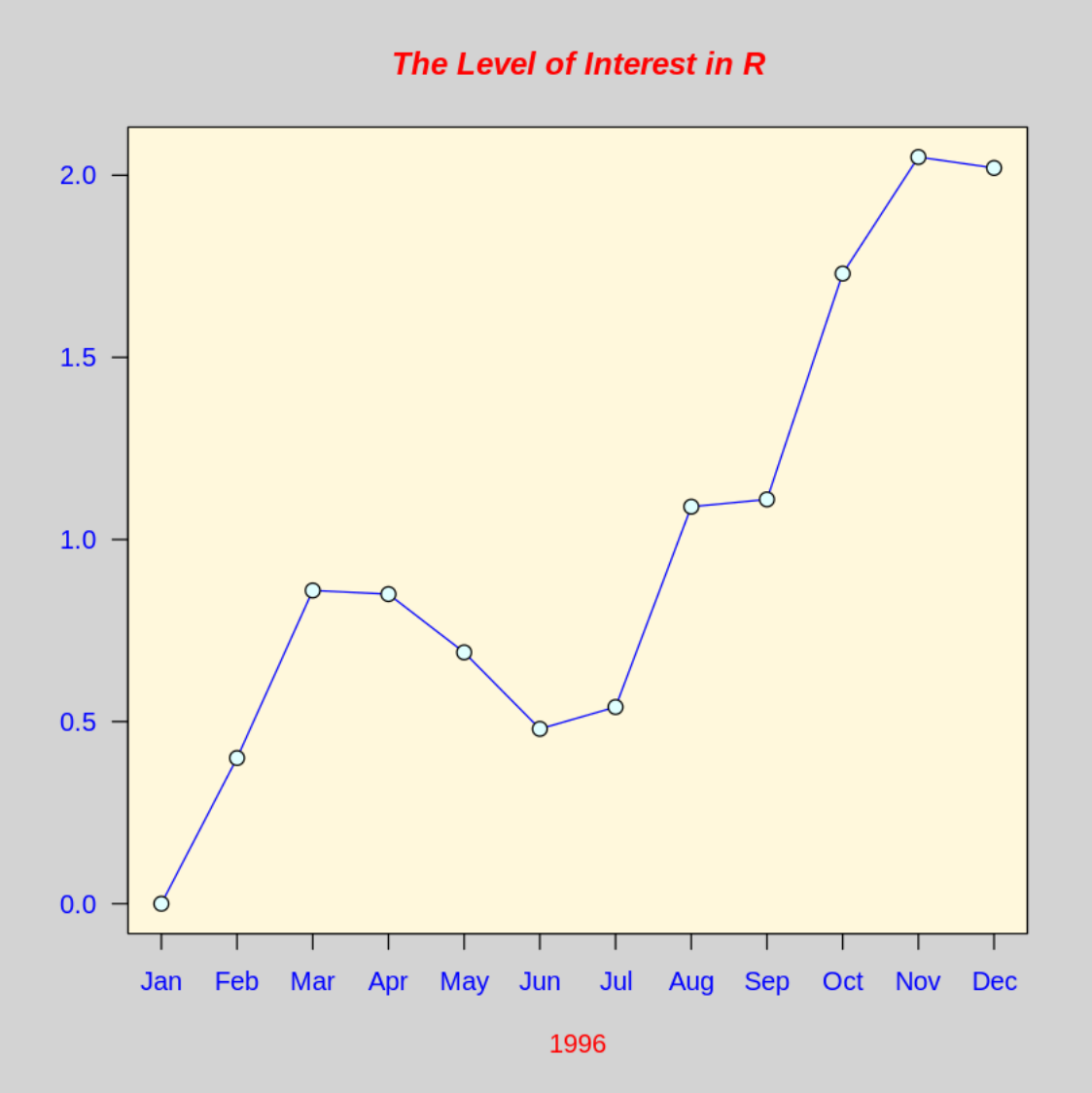



```
> polygon(xx, yy, col="gray")
```

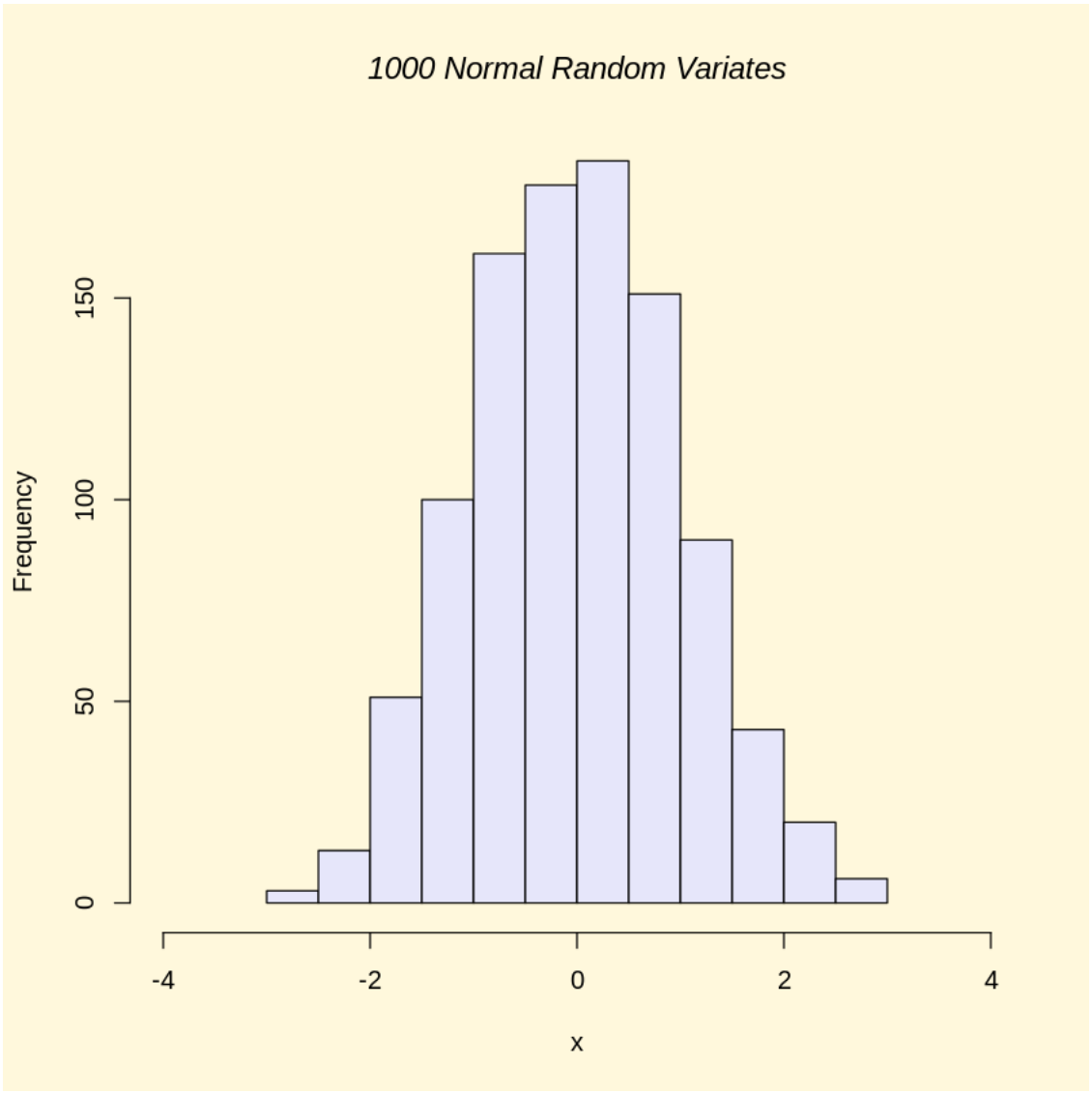
```
> title("Distance Between Brownian Motions")

> ## Colored plot margins, axis labels and titles.       You do need to be
> ## careful with these kinds of effects.       It's easy to go completely
> ## over the top and you can end up with your lunch all over the keyboard.
> ## On the other hand, my market research clients love it.
>
> x <- c(0.00, 0.40, 0.86, 0.85, 0.69, 0.48, 0.54, 1.09, 1.11, 1.73, 2.05, 2.02)

> par(bg="lightgray")

> plot(x, type="n", axes=FALSE, ann=FALSE)
```



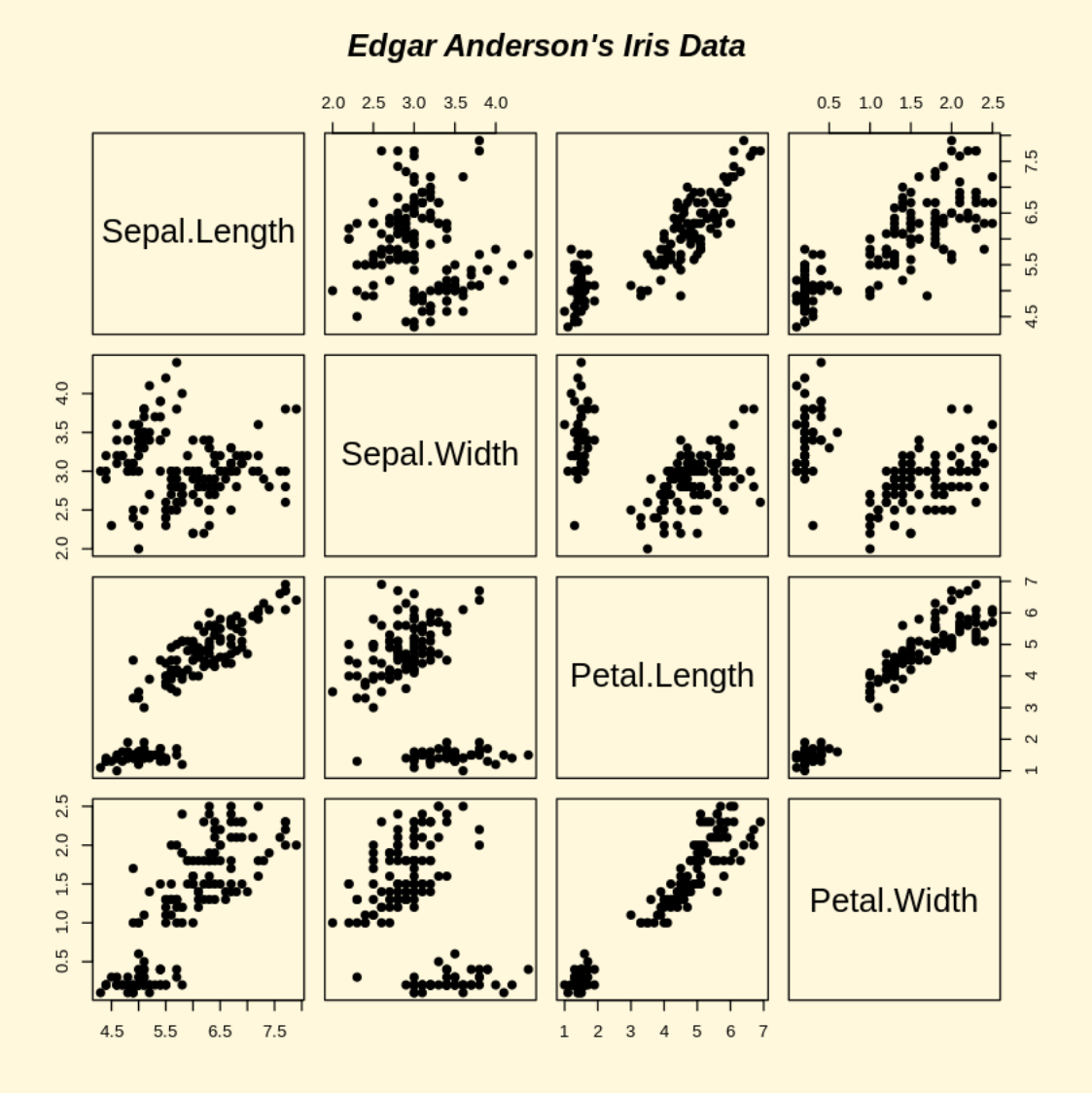
```
> usr <- par("usr")

> rect(usr[1], usr[3], usr[2], usr[4], col="cornsilk", border="black")

> lines(x, col="blue")

> points(x, pch=21, bg="lightcyan", cex=1.25)

> axis(2, col.axis="blue", las=1)

> axis(1, at=1:12, lab=month.abb, col.axis="blue")

> box()

> title(main= "The Level of Interest in R", font.main=4, col.main="red")

> title(xlab= "1996", col.lab="red")

> ## A filled histogram, showing how to change the font used for the
> ## main title without changing the other annotation.
>
> par(bg="cornsilk")

> x <- rnorm(1000)

> hist(x, xlim=range(-4, 4, x), col="lavender", main="")
```




```
> title(main="1000 Normal Random Variates", font.main=3)

> ## A scatterplot matrix
> ## The good old Iris data (yet again)
>
> pairs(iris[1:4], main="Edgar Anderson's Iris Data", font.main=4, pch=19)
```

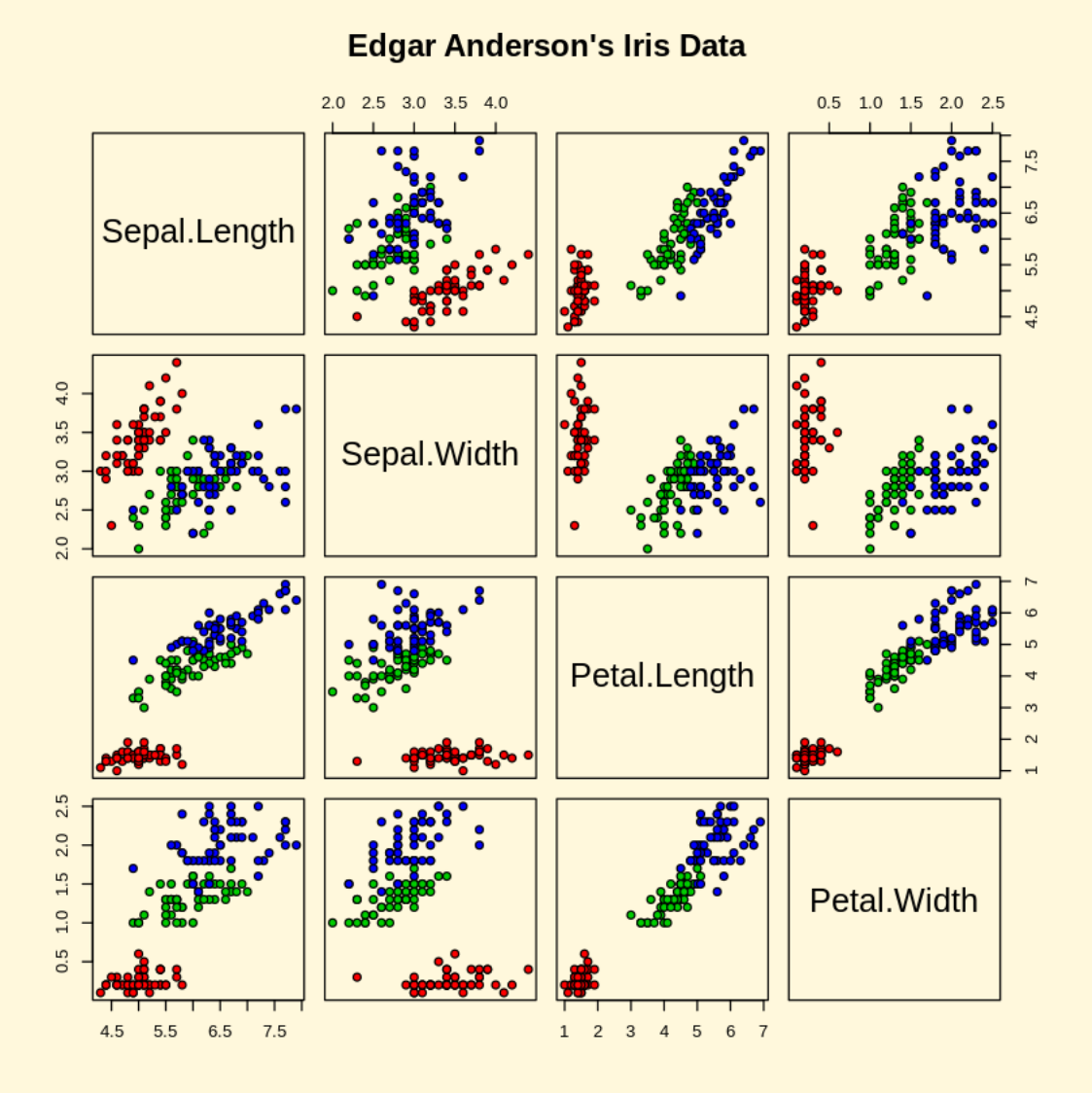



```
> pairs(iris[1:4], main="Edgar Anderson's Iris Data", pch=21,
+       bg = c("red", "green3", "blue")[unclass(iris$Species)])
```



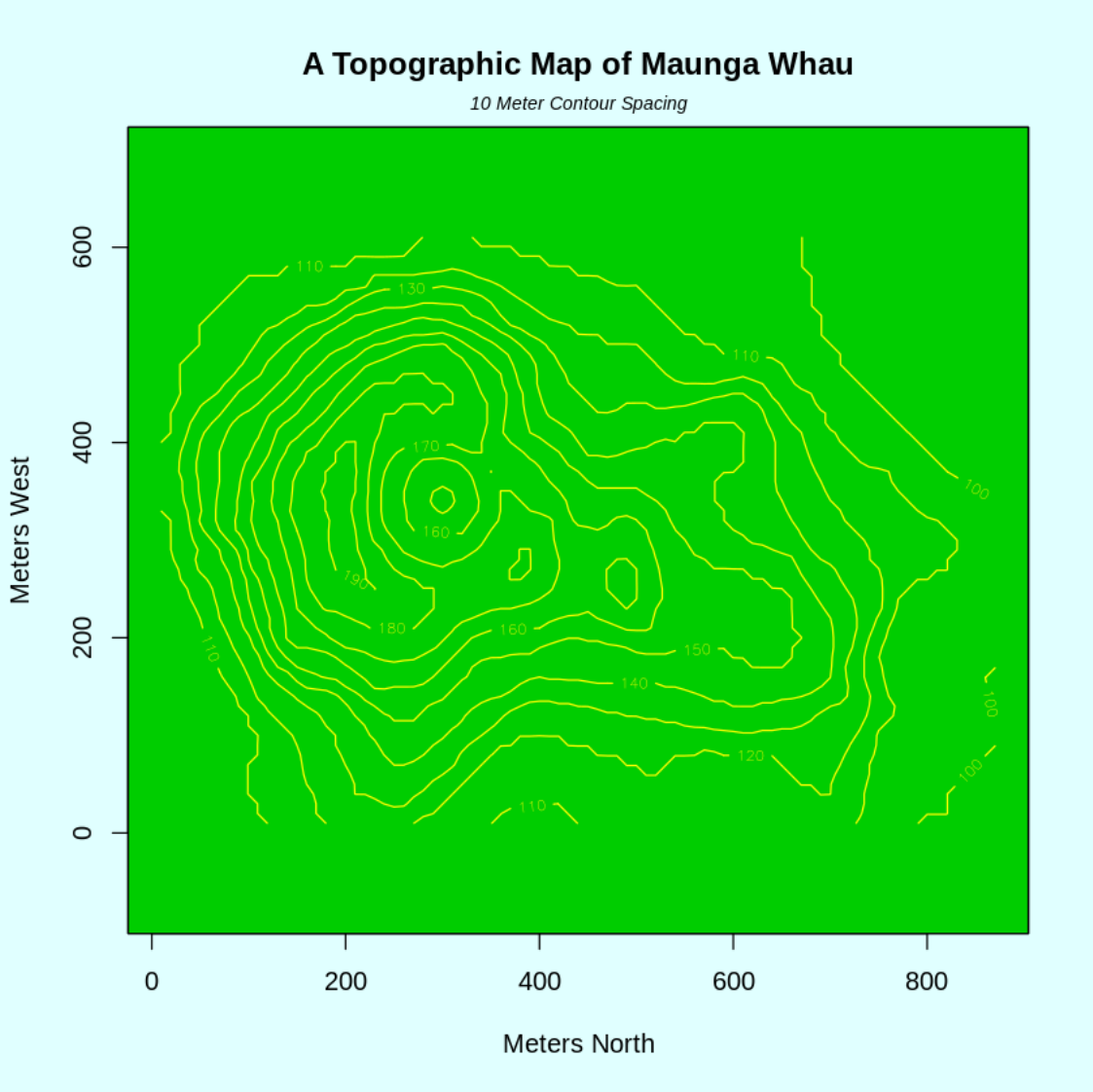

```
> ## Contour plotting
> ## This produces a topographic map of one of Auckland's many volcanic "peaks".
>
> x <- 10*1:nrow(volcano)

> y <- 10*1:ncol(volcano)

> lev <- pretty(range(volcano), 10)

> par(bg = "lightcyan")

> pin <- par("pin")
```

```
> xdelta <- diff(range(x))

> ydelta <- diff(range(y))

> xscale <- pin[1]/xdelta

> yscale <- pin[2]/ydelta

> scale <- min(xscale, yscale)

> xadd <- 0.5*(pin[1]/scale - xdelta)

> yadd <- 0.5*(pin[2]/scale - ydelta)

> plot(numeric(0), numeric(0),
+      xlim = range(x)+c(-1,1)*xadd, ylim = range(y)+c(-1,1)*yadd,
+      type = "n", ann = FALSE)
```

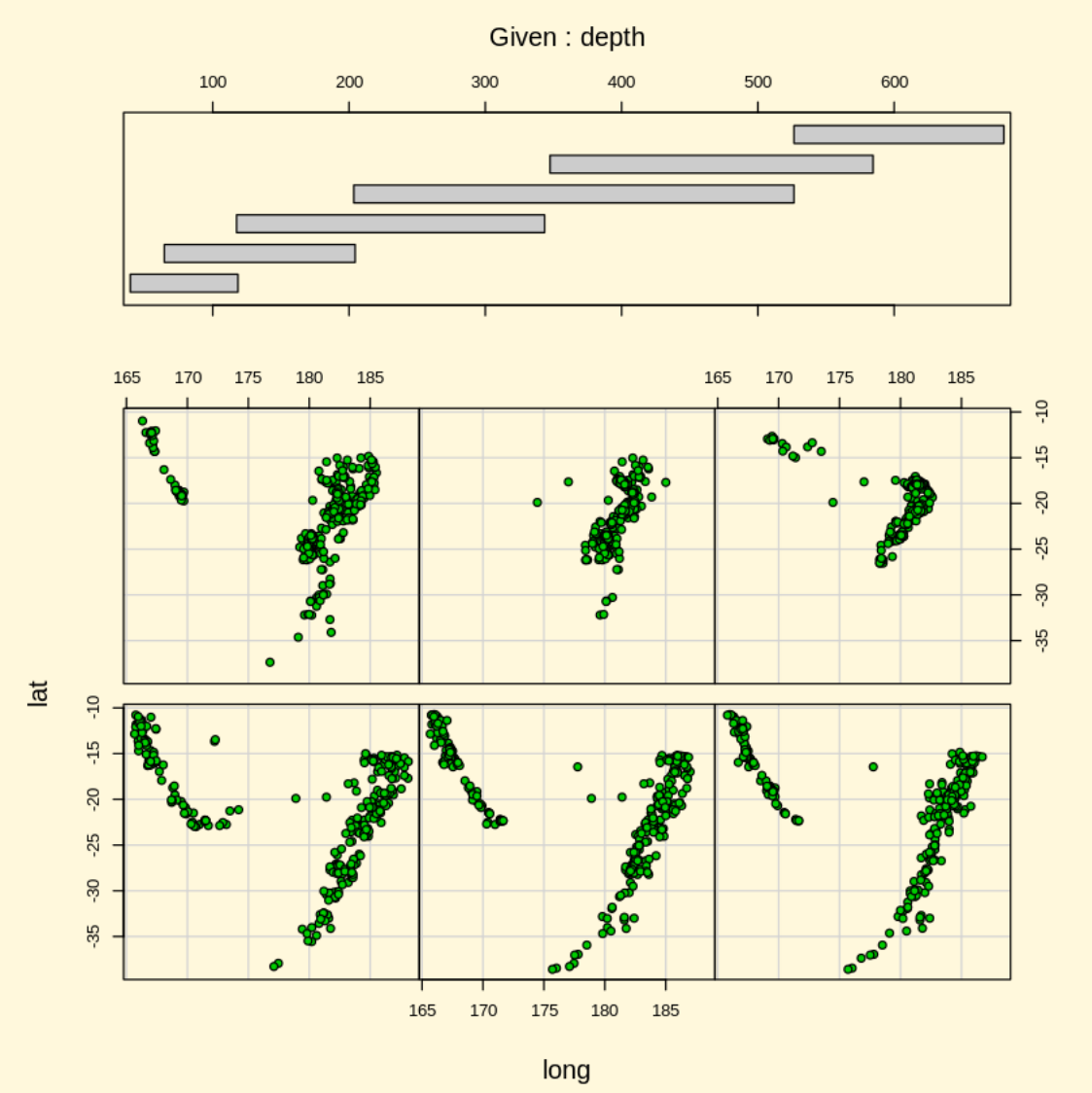



```
> usr <- par("usr")

> rect(usr[1], usr[3], usr[2], usr[4], col="green3")

> contour(x, y, volcano, levels = lev, col="yellow", lty="solid", add=TRUE)

> box()

> title("A Topographic Map of Maunga Whau", font= 4)

> title(xlab = "Meters North", ylab = "Meters West", font= 3)
```

```
> mtext("10 Meter Contour Spacing", side=3, line=0.35, outer=FALSE,
+       at = mean(par("usr")[1:2]), cex=0.7, font=3)

> ## Conditioning plots
>
> par(bg="cornsilk")

> coplot(lat ~ long | depth, data = quakes, pch = 21, bg = "green3")
```




```
> par(opar)
```

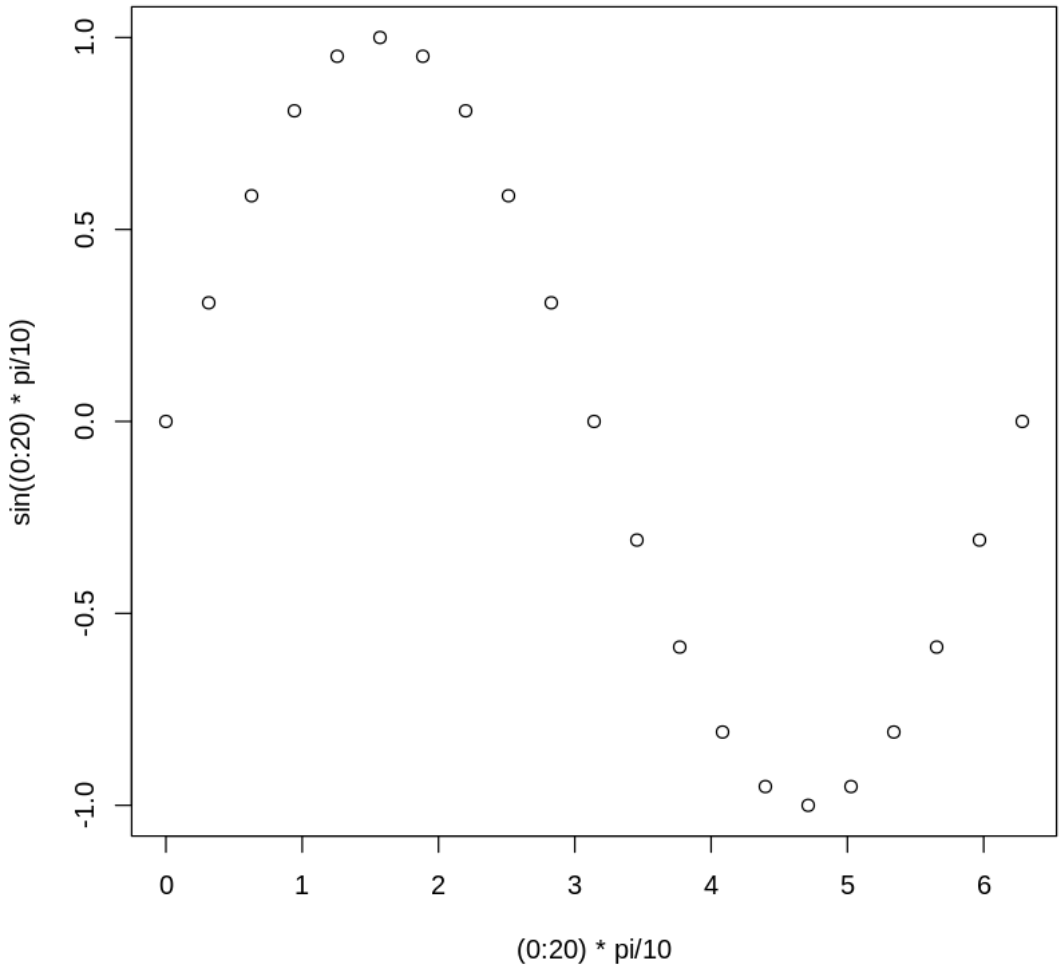


1.1 3.1 plot () and allied functions

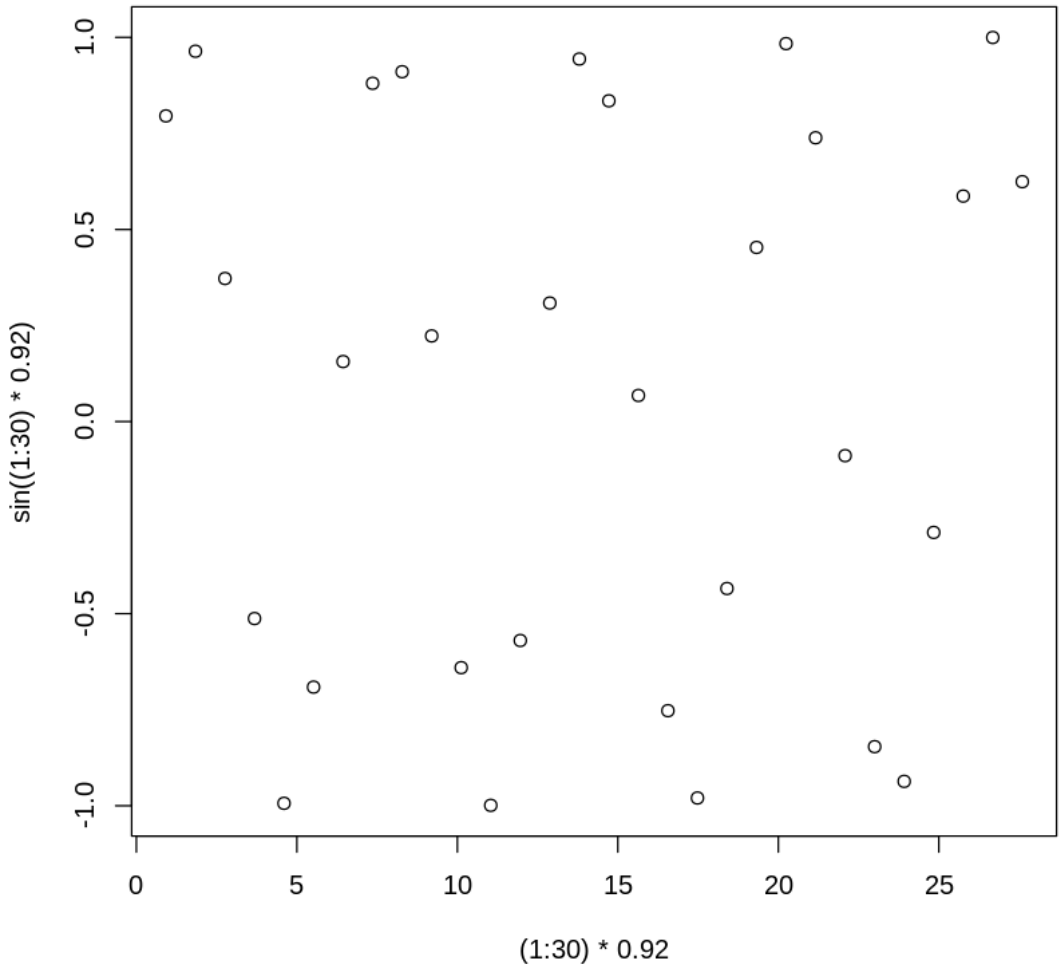
The following both plot y against x:

```
plot(y ~ x) # Use a formula to specify the graph
plot(x, y) # Obviously x and y must be the same length.
```

[101]:
```
plot((0:20)*pi/10, sin((0:20)*pi/10))
```



[102]:
```
plot((1:30)*0.92, sin((1:30)*0.92))
```



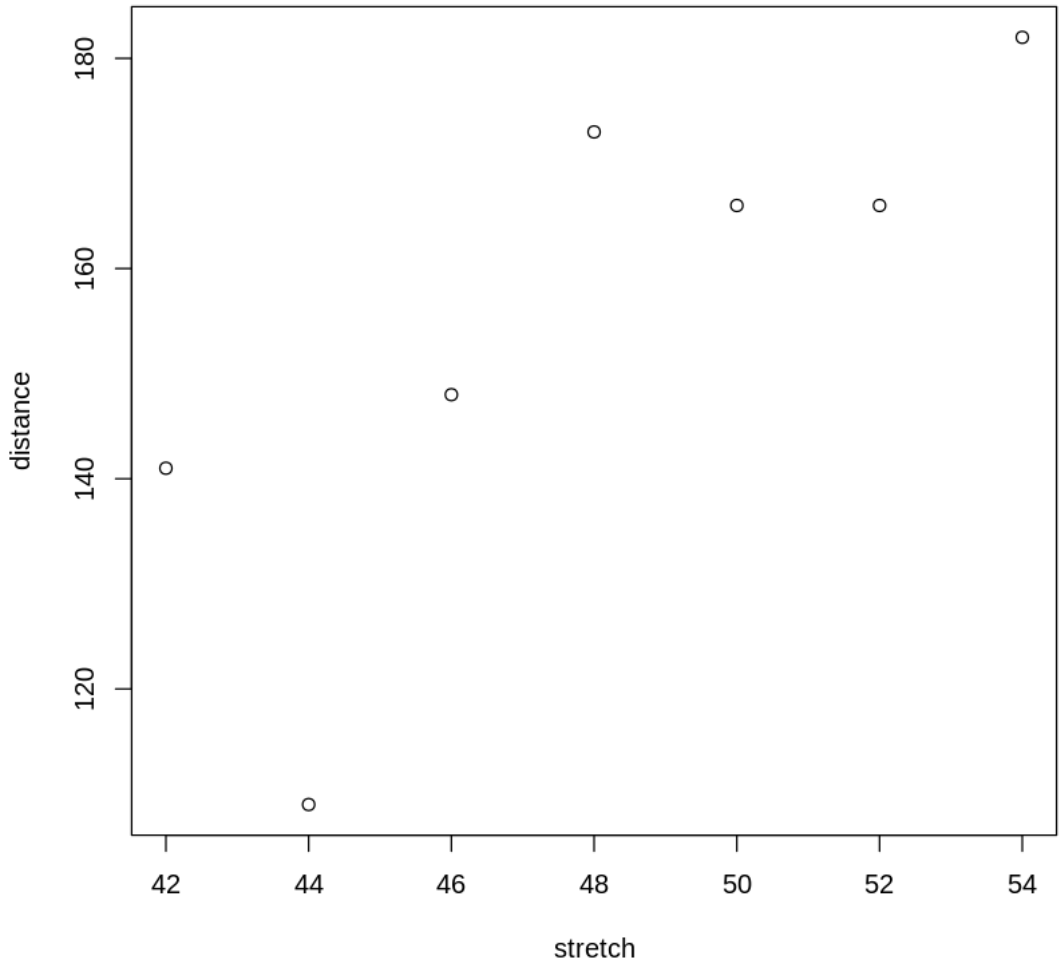
Appearance of the graphs:

The points are scattered and do not appear to lie on a sine curve. The graph is too tall and narrow, making it difficult to see the shape of the data. **Making it obvious:**

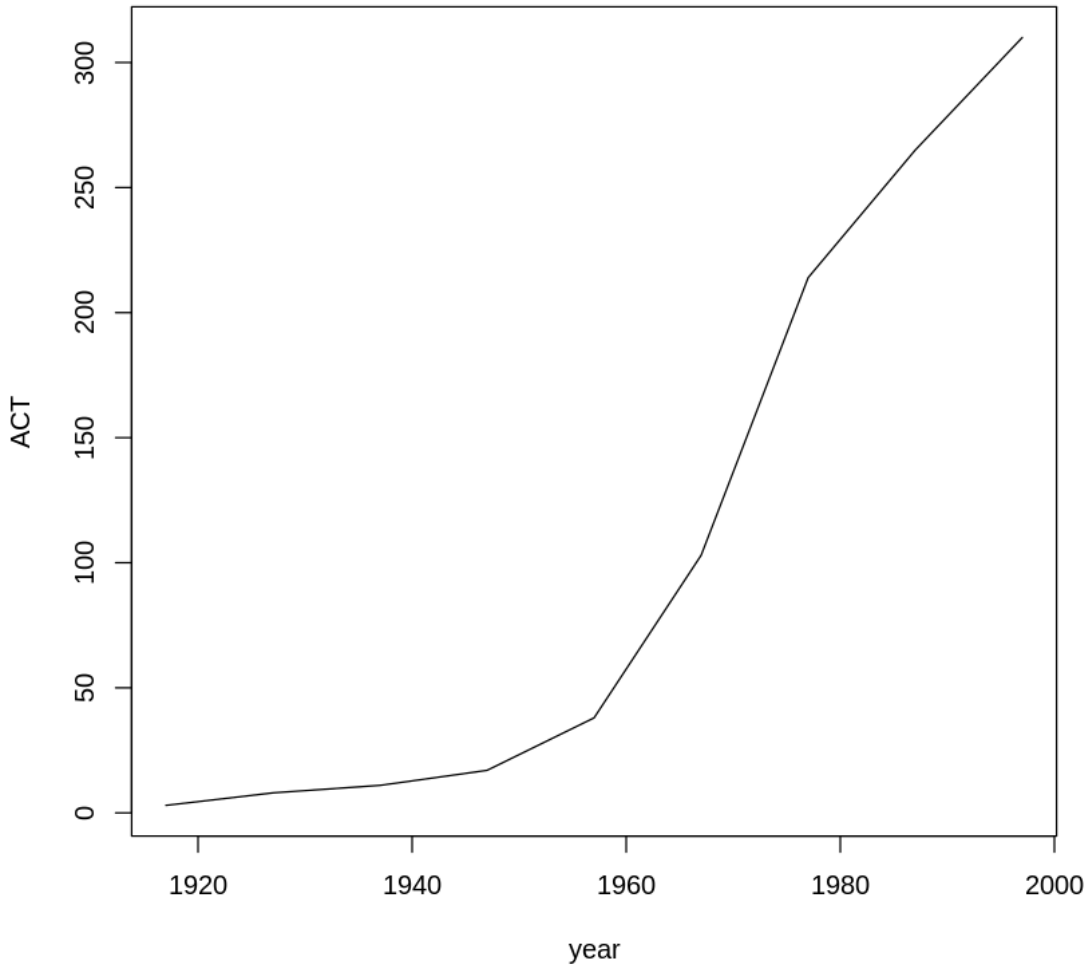
- **Adjust the aspect ratio**: Drag the lower border of the graph sheet upwards to make it shorter and wider. This will make it easier to see the shape of the data.
- **Add a smooth curve:** Use interpolation to connect the points with a smooth curve. This will make it clear that the points lie on a sine curve.

[103]:
```
elasticband <- data.frame(stretch=c(46,54,48,50,44,42,52),
 distance=c(148,182,173,166,109,141,166))
attach(elasticband) # R now knows where to find distance & stretch
```

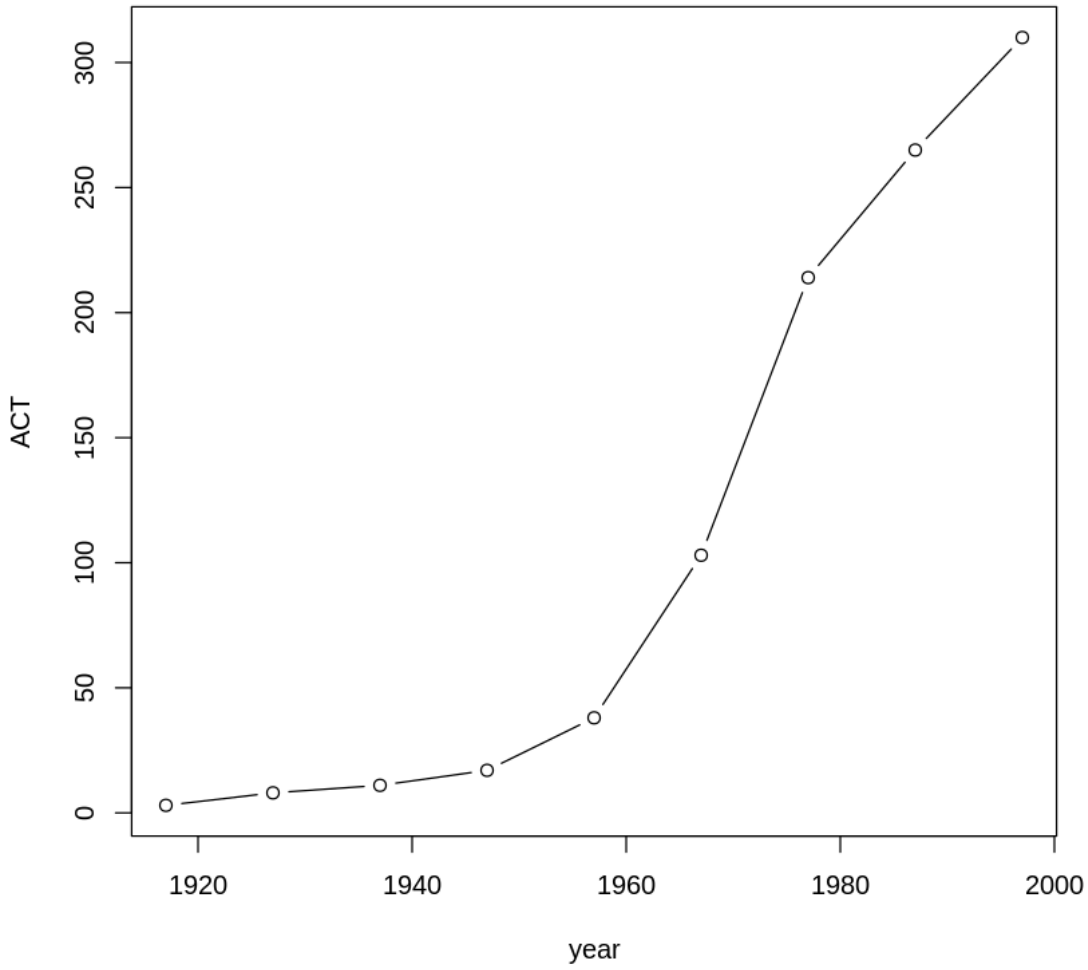
[104]:
```
plot(distance ~ stretch)
```



[106]:
```
austpop <- read.csv("/content/austpop.csv")
plot(ACT ~ year, data=austpop, type="l")
```



[107]:
```
plot(ACT ~ year, data=austpop, type="b")
```



Summary of plotting functions:

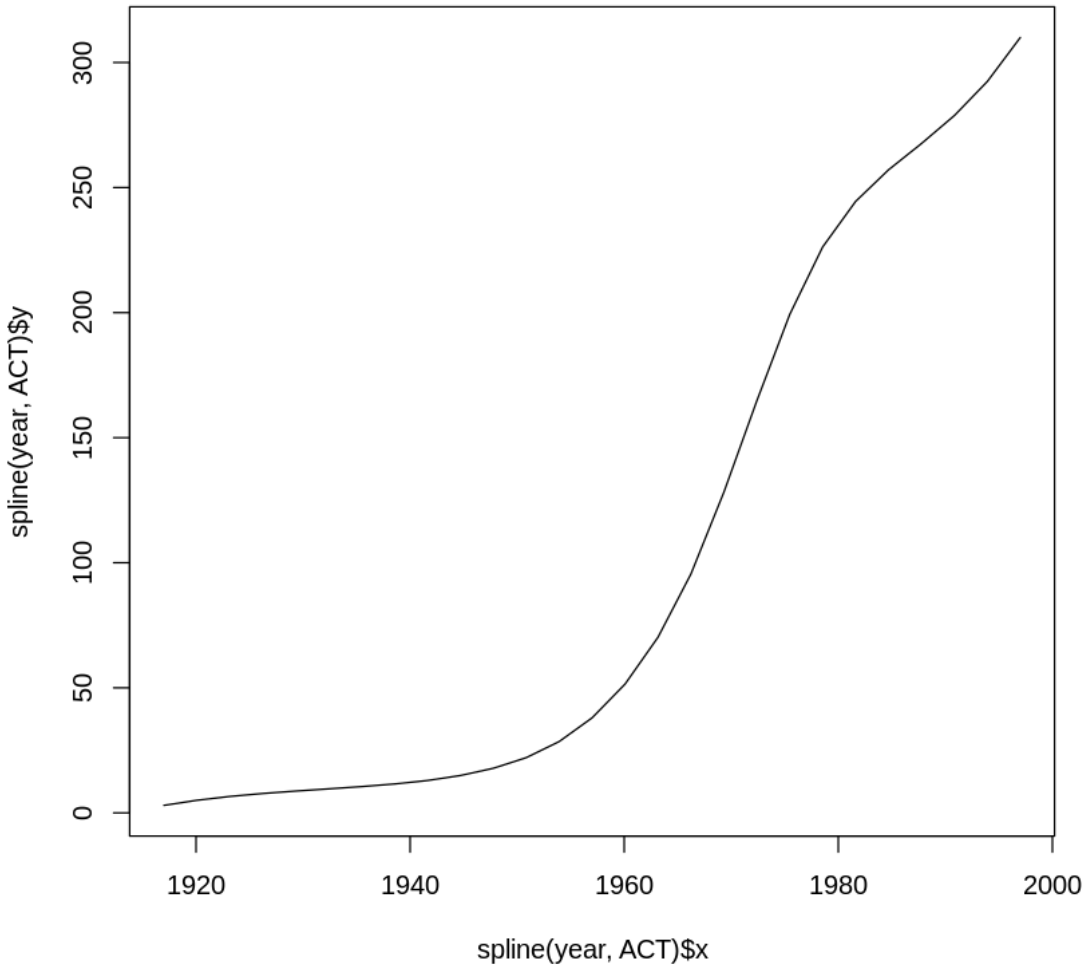
- points(): Adds points to a plot.
- lines(): Adds lines to a plot.
- text(): Adds text at specified locations on the plot.
- mtext(): Places text in one of the margins of the plot.
- axis(): Gives fine control over the appearance of the axes, including tick marks and labels.

points() and lines() are used to plot data. text() and mtext() are used to add annotations to plots. axis() is used to customize the appearance of the axes.

[108]:
```
attach(austpop)
plot(spline(year, ACT), type="l") # Fit smooth curve through points
```

```
The following object is masked from ais (pos = 6):
```

```
    rownames

The following object is masked from ais (pos = 7):

    rownames

The following object is masked from ais (pos = 8):

    rownames

The following object is masked from ais (pos = 9):

    rownames

The following object is masked from ais (pos = 10):

    rownames

The following object is masked from ais (pos = 11):

    rownames
```



[109]:
```
detach(austpop) # In S-PLUS, specify detach("austpop")
```

3.1.1 Plot methods for other classes of object

The plot function is a generic function that has special methods for "plotting" various different classes of object. For example, plotting a data frame gives, for each numeric variable, a normal probability plot. Plotting the lm object that is created by the use of the lm() modelling function gives diagnostic and other information that is intended to help in the interpretation of regression results

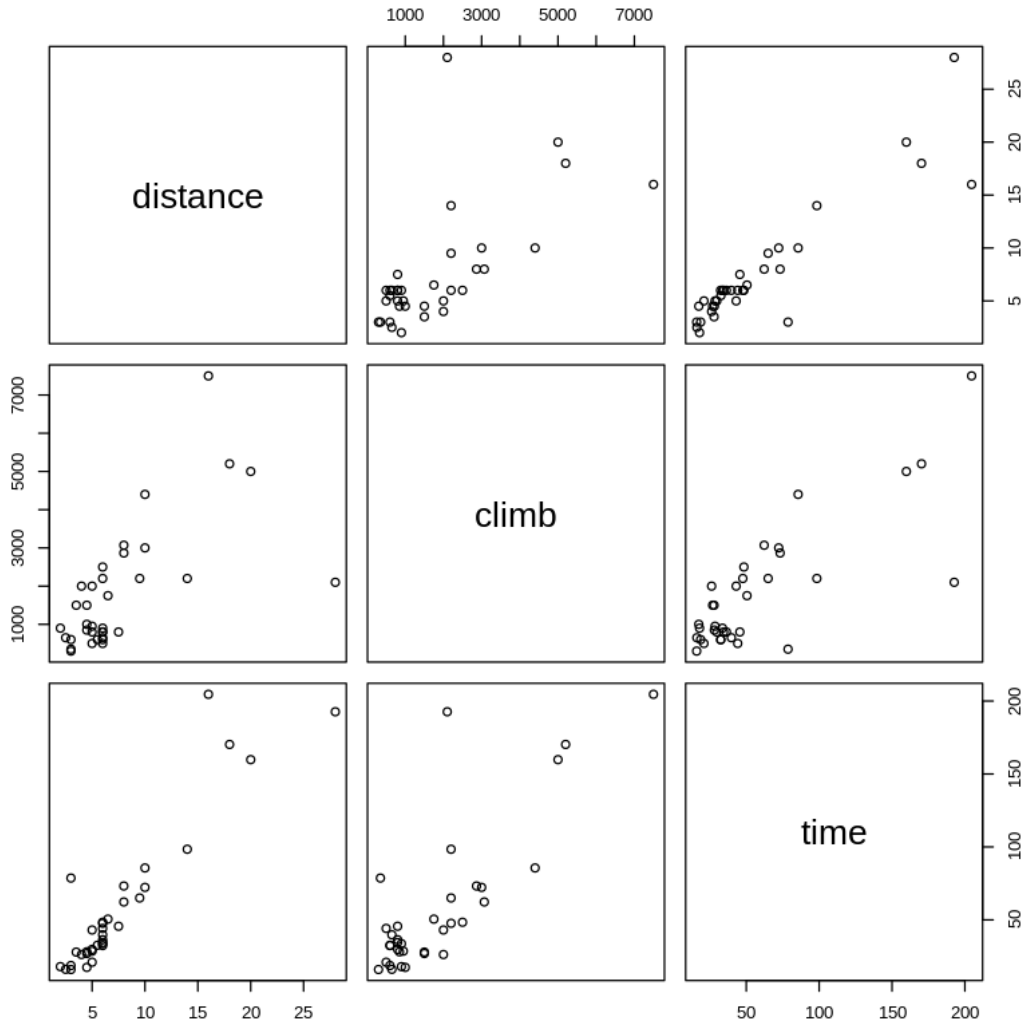
[111]:
```
# Load the RData file
load("/content/hills.RData")

# Print the contents of the loaded object
```

```
hills
```

A data.frame: 35 × 3

	distance <dbl>	climb <dbl>	time <dbl>
Greenmantle	2.5	650	16.083
Carnethy	6.0	2500	48.350
Craig Dunain	6.0	900	33.650
Ben Rha	7.5	800	45.600
Ben Lomond	8.0	3070	62.267
Goatfell	8.0	2866	73.217
Bens of Jura	16.0	7500	204.617
Cairnpapple	6.0	800	36.367
Scolty	5.0	800	29.750
Traprain	6.0	650	39.750
Lairig Ghru	28.0	2100	192.667
Dollar	5.0	2000	43.050
Lomonds	9.5	2200	65.000
Cairn Table	6.0	500	44.133
Eildon Two	4.5	1500	26.933
Cairngorm	10.0	3000	72.250
Seven Hills	14.0	2200	98.417
Knock Hill	3.0	350	78.650
Black Hill	4.5	1000	17.417
Creag Beag	5.5	600	32.567
Kildcon Hill	3.0	300	15.950
Meall Ant-Suidhe	3.5	1500	27.900
Half Ben Nevis	6.0	2200	47.633
Cow Hill	2.0	900	17.933
N Berwick Law	3.0	600	18.683
Creag Dubh	4.0	2000	26.217
Burnswark	6.0	800	34.433
Largo Law	5.0	950	28.567
Criffel	6.5	1750	50.500
Acmony	5.0	500	20.950
Ben Nevis	10.0	4400	85.583
Knockfarrel	6.0	600	32.383
Two Breweries	18.0	5200	170.250
Cockleroi	4.5	850	28.100
Moffat Chase	20.0	5000	159.833

[112]:
```
plot(hills)  # has the same effect as pairs(hills)
```



1.2 3.2 Fine control – Parameter settings

The default settings of parameters, such as character size, are often adequate. When it is necessary to change parameter settings for a subsequent plot, the par() function does this. For example,

[113]:
```
par(cex=1.25) # character expansion increases the text and plot symbol size 25%␣
↪above the default.
```

On the first use of par() to make changes to the current device, it is often useful to store existing settings, so that they can be restored later. For this, specif

[114]:
```
oldpar <- par(cex=1.25, mex=1.25) # mex=1.25 expands the margin by 25%
oldpar
```

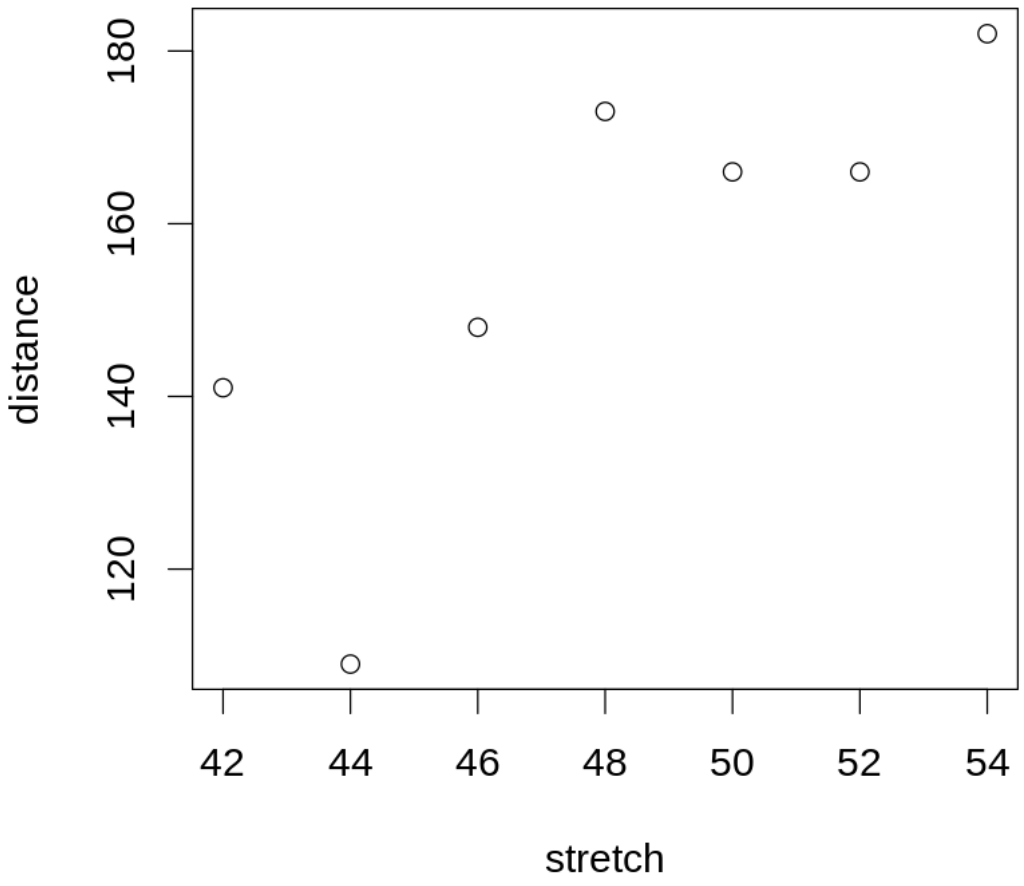
$cex 1

$mex 1

This stores the existing settings in oldpar, then changes parameters (here cex and mex) as requested. To restore the original parameter settings at some later time, enter par(oldpar). Here is an example:

[115]:
```
attach(elasticband)
oldpar <- par(cex=1.5)
plot(distance ~ stretch)
```

```
The following objects are masked from elasticband (pos = 3):

    distance, stretch
```



[116]:
```
par(oldpar) # Restores the earlier settings
```

[117]:
```
detach(elasticband)
```

[118]:
```
# Inside a function specify, e.g
oldpar <- par(cex=1.25)
on.exit(par(oldpar))
```

Type in help(par) to get details of all the parameter settings that are available with par()

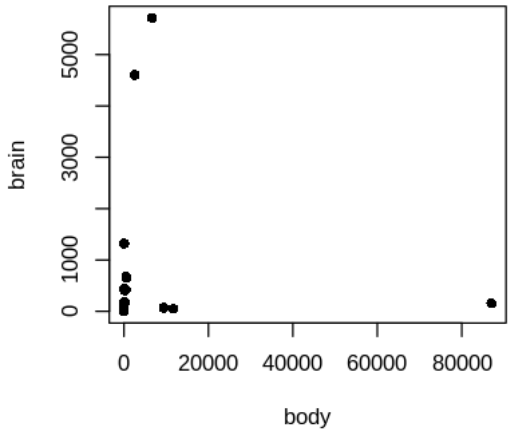
3.2.1 Multiple plots on the one page

The parameter mfrow can be used to configure the graphics sheet so that subsequent plots appear row by row, one after the other in a rectangular layout, on the one page. For a column by column layout, use mfcol instead. In the example below we present four different transformations of the primates data, in a two by two layout:

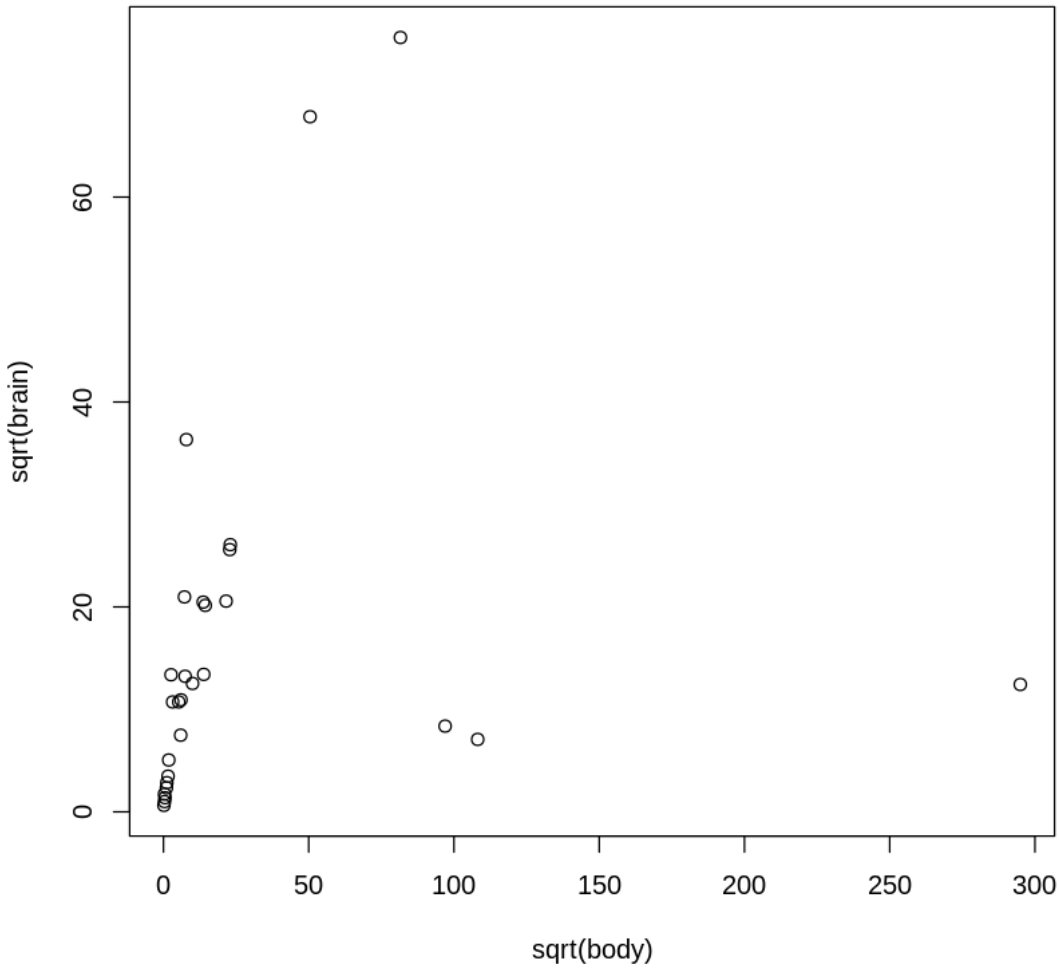
[119]:
```
par(mfrow=c(2,2), pch=16)
Animals <- read.csv("/content/Animals.csv")
attach(Animals) # This dataset is in the MASS package, which must be attached
plot(body, brain)
```

```
The following object is masked from ais (pos = 6):

    rownames


The following object is masked from ais (pos = 7):

    rownames


The following object is masked from ais (pos = 8):

    rownames


The following object is masked from ais (pos = 9):

    rownames


The following object is masked from ais (pos = 10):

    rownames
```

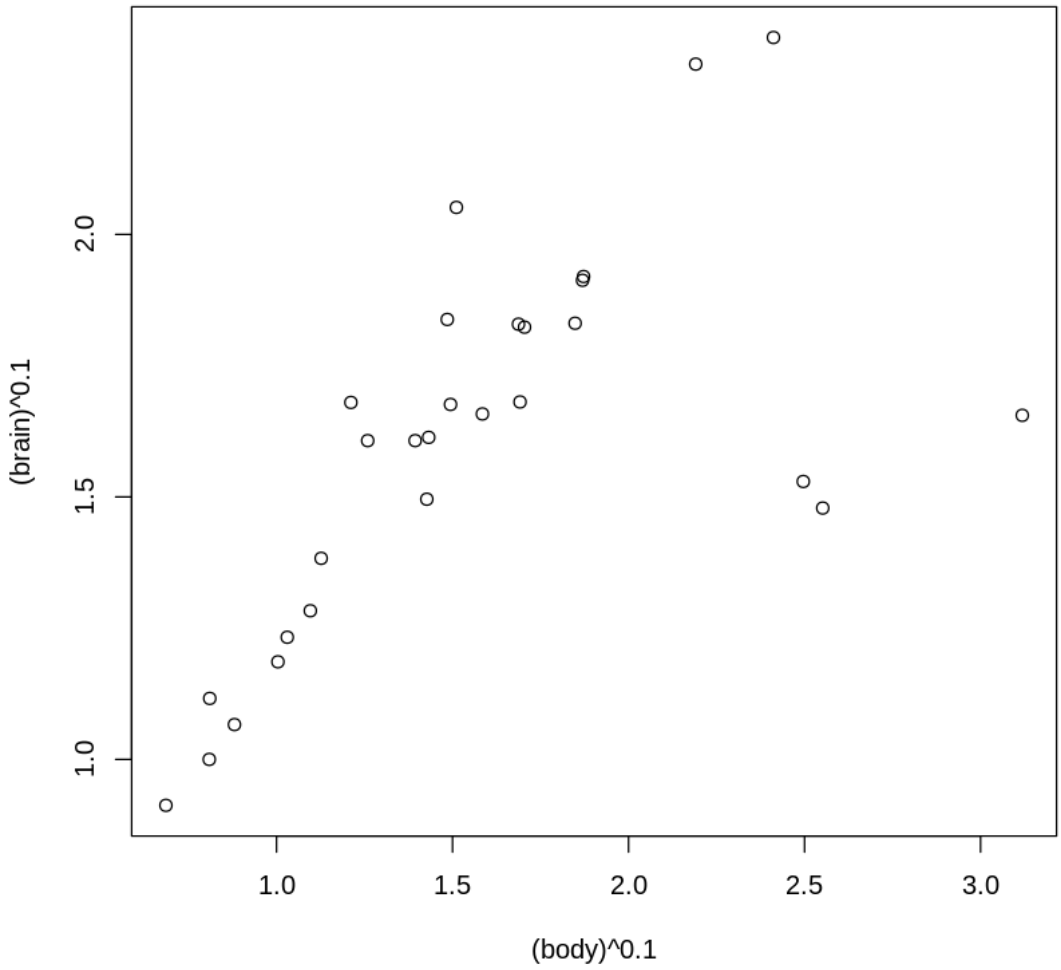
```
The following object is masked from ais (pos = 11):

    rownames
```



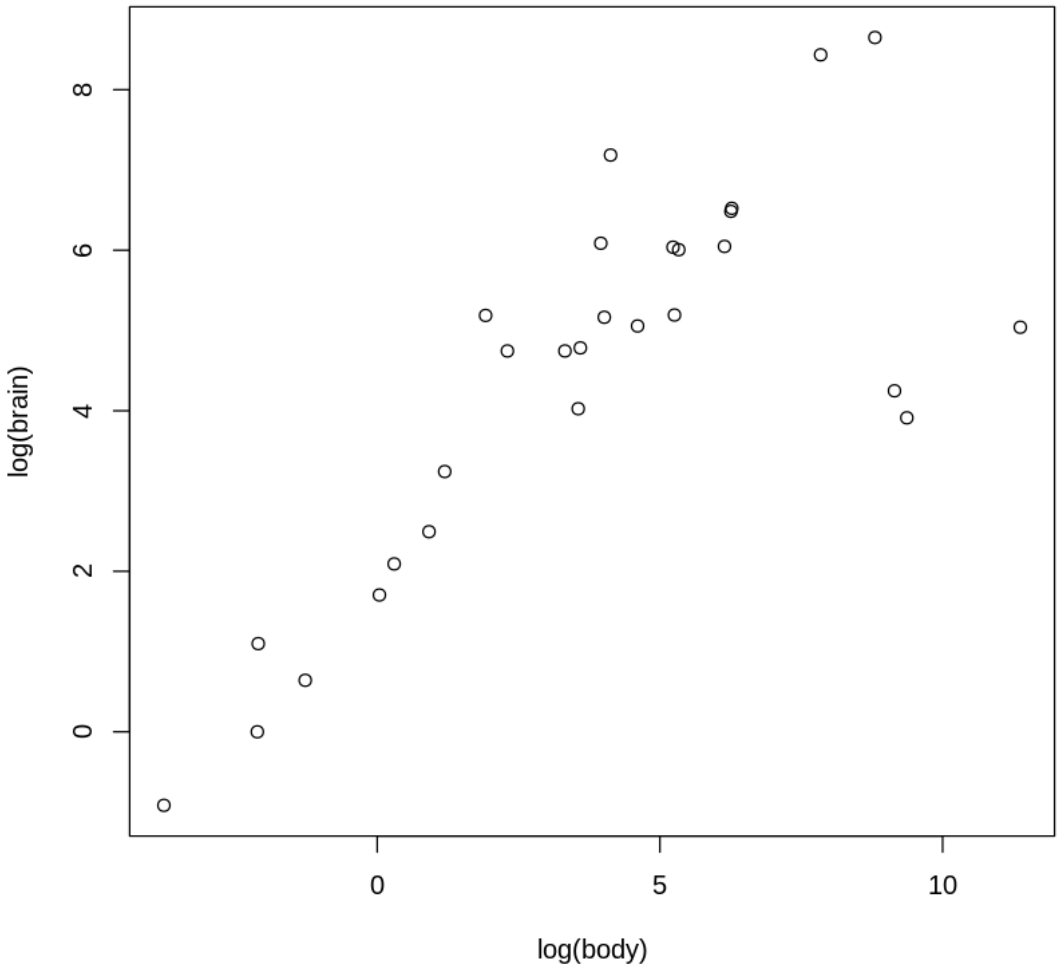
[120]:
```
plot(sqrt(body), sqrt(brain))
```



[121]:
```
plot((body)^0.1, (brain)^0.1)
```



[122]:
```
plot(log(body),log(brain))
```



[123]:
```
detach(Animals)
par(mfrow=c(1,1), pch=1) # Restore to 1 figure per page
```

3.2.2 The shape of the graph sheet

Controlling the shape of the graph page:

You can control the shape of the graph page by setting the width, height, and point size when you open the graphics device. For example, the following code opens a graphics device that is 6 inches wide, 4 inches high, and uses a point size of 12:

```
x11(width = 6, height = 4, pointsize = 12)
```

This will make the individual plots rectangular rather than square. **Explanation:**

- The win.graph() and x11() functions are used to open a graphics device on Windows and X11

(Unix) systems, respectively.

- These functions take several parameters, including width, height, and pointsize.
- The width and height parameters specify the dimensions of the graphics device in inches.
- The pointsize parameter specifies the size of the points used to draw the graphics, in 1/72 of an inch.
- By setting the width and height parameters, you can control the aspect ratio of the individual plots.
- For example, setting the width to a larger value than the height will make the plots wider than they are tall.

1.3 3.3 Adding points, lines and text

Here is a simple example that shows how to use the function text() to add text labels to the points on a plot.

[124]:
```
primates <- read.csv("/content/primates.csv")
primates
```

A data.frame: 5 × 3

rownames <chr>	Bodywt <dbl>	Brainwt <int>
Potar monkey	10.0	115
Gorilla	207.0	406
Human	62.0	1320
Rhesus monkey	6.8	179
Chimp	52.2	440

[125]:
```
# Row names can be created in several different ways. They can be assigned␣
 ↪directly, e.g.
row.names(primates) <- c("P-monkey","Gorilla","Human", "Rhesus monkey","Chimp")
primates
```

A data.frame: 5 × 3

	rownames <chr>	Bodywt <dbl>	Brainwt <int>
P-monkey	Potar monkey	10.0	115
Gorilla	Gorilla	207.0	406
Human	Human	62.0	1320
Rhesus monkey	Rhesus monkey	6.8	179
Chimp	Chimp	52.2	440

[126]:
```
attach(primates) # Needed if primates is not already attached.
plot(Bodywt, Brainwt)
text(x=Bodywt, y=Brainwt, labels=row.names(primates), pos=4)
 # pos=4 positions text to the right of the point
```

```
The following object is masked from ais (pos = 6):

    rownames
```

```
The following object is masked from ais (pos = 7):

    rownames

The following object is masked from ais (pos = 8):

    rownames

The following object is masked from ais (pos = 9):

    rownames

The following object is masked from ais (pos = 10):

    rownames

The following object is masked from ais (pos = 11):

    rownames
```

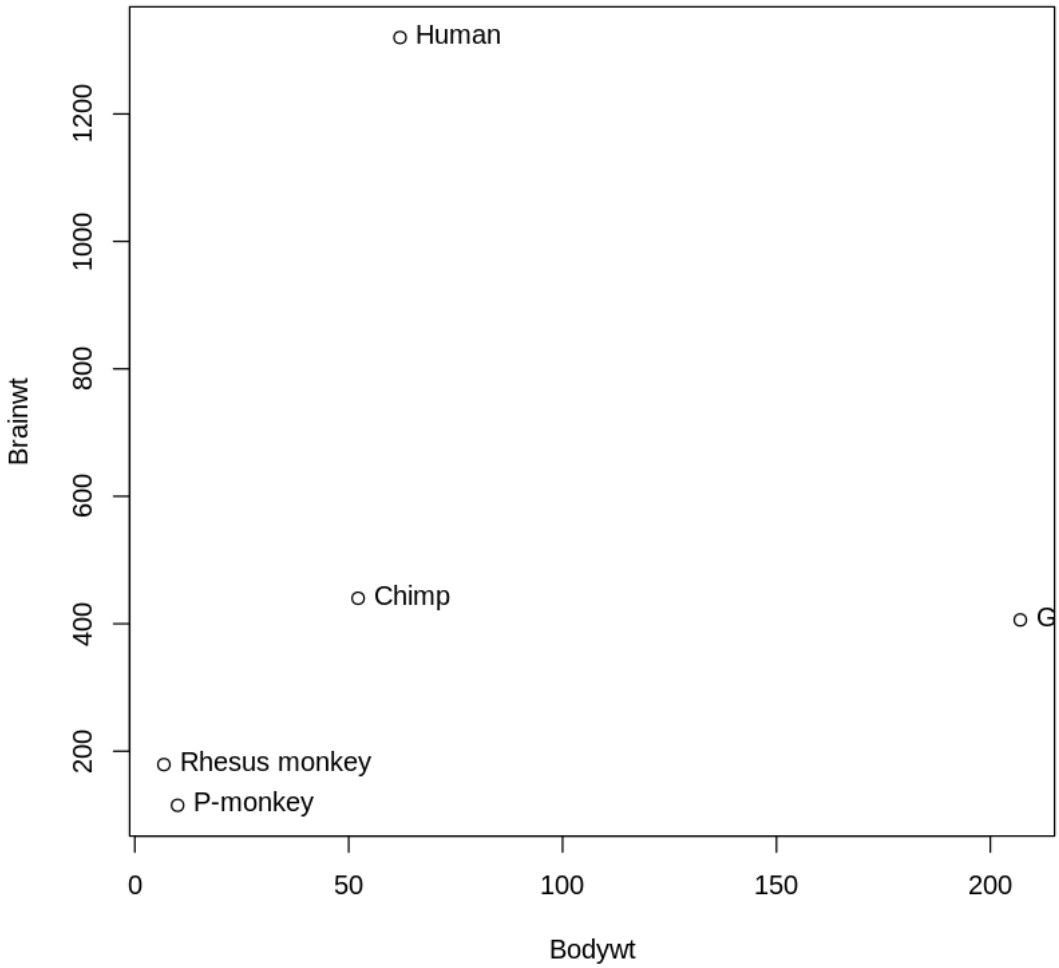
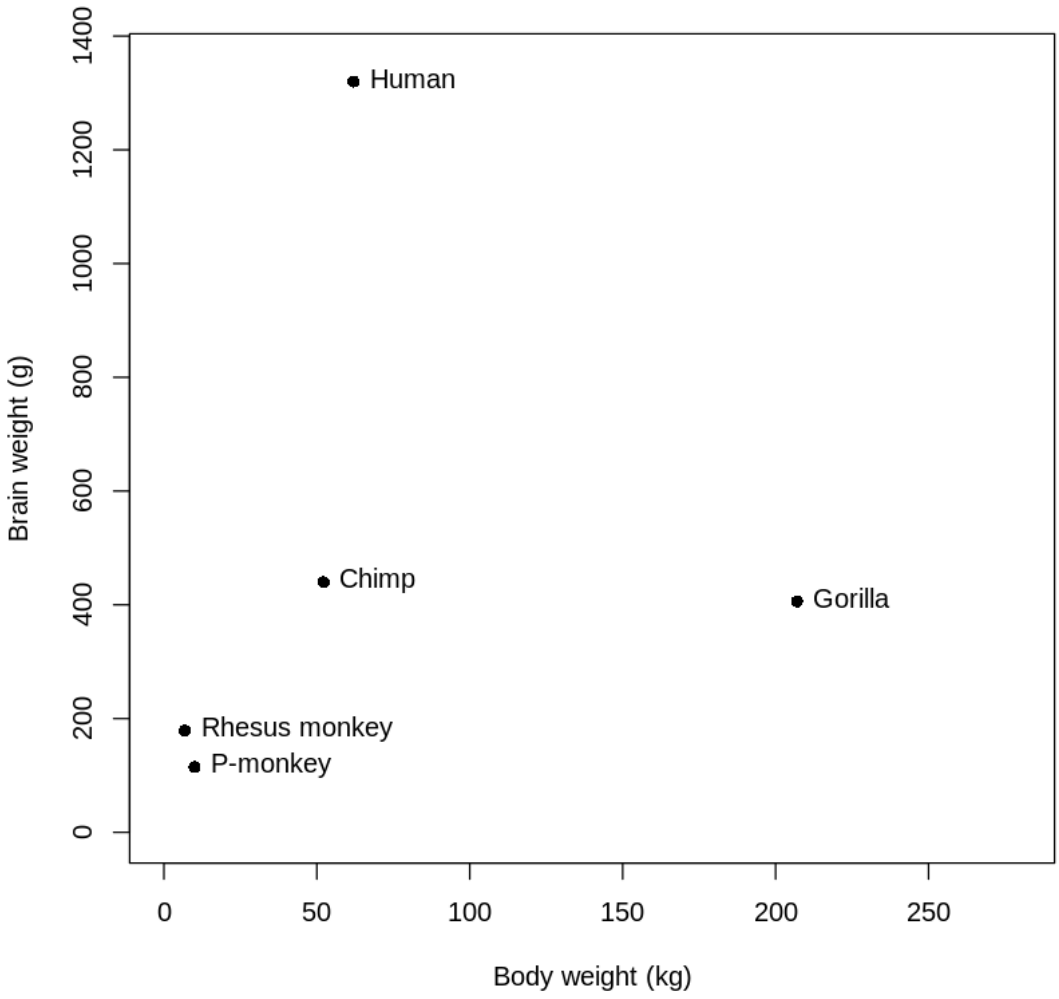


Figure 7A serves its purpose for identifying points but lacks presentation quality. Figure 7B demonstrates improvements. It employs the `xlab` and `ylab` parameters to provide meaningful axis titles. By setting `pos=4`, the axis labeling is positioned to the right of the points. Additionally, `pch=16` is used to depict the plot characters as heavy black dots, enhancing their visibility against the labeling.

[127]:
```
plot(x=Bodywt, y=Brainwt, pch=16,
 xlab="Body weight (kg)", ylab="Brain weight (g)",
 xlim=c(0,280), ylim=c(0,1350))
# Specify xlim so that there is room for the labels
text(x=Bodywt, y=Brainwt, labels=row.names(primates), pos=4)
```



[128]:
```
detach(primates)
```

To place the text to the left of the points, specify

```
text(x=Bodywt, y=Brainwt, labels=row.names(primates), pos=2)
```

3.3.1 Size, colour and choice of plotting symbol

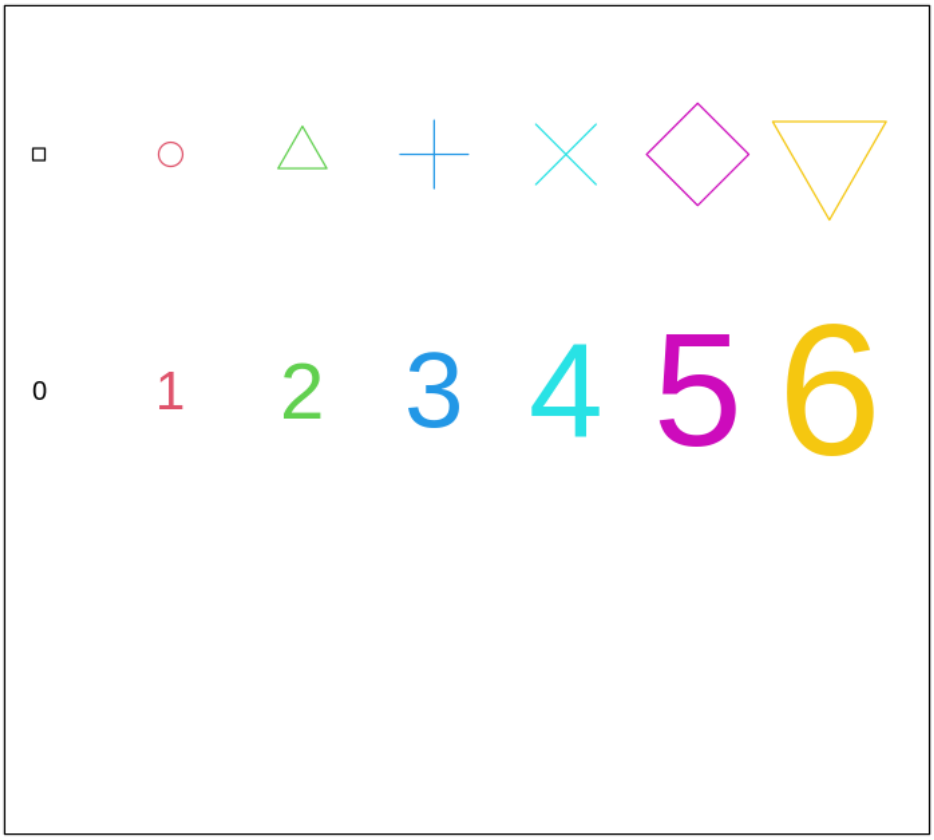
Size: Use cex (character expansion) to control the size of points and symbols in plot() and points().

Color: Use col (colour) to choose the color of points and symbols in plot() and points().

Symbol: Use pch to select the shape of points and symbols in plot() and points().

Text size and color: cex and col can also be used with text() to control the size and color of text elements.

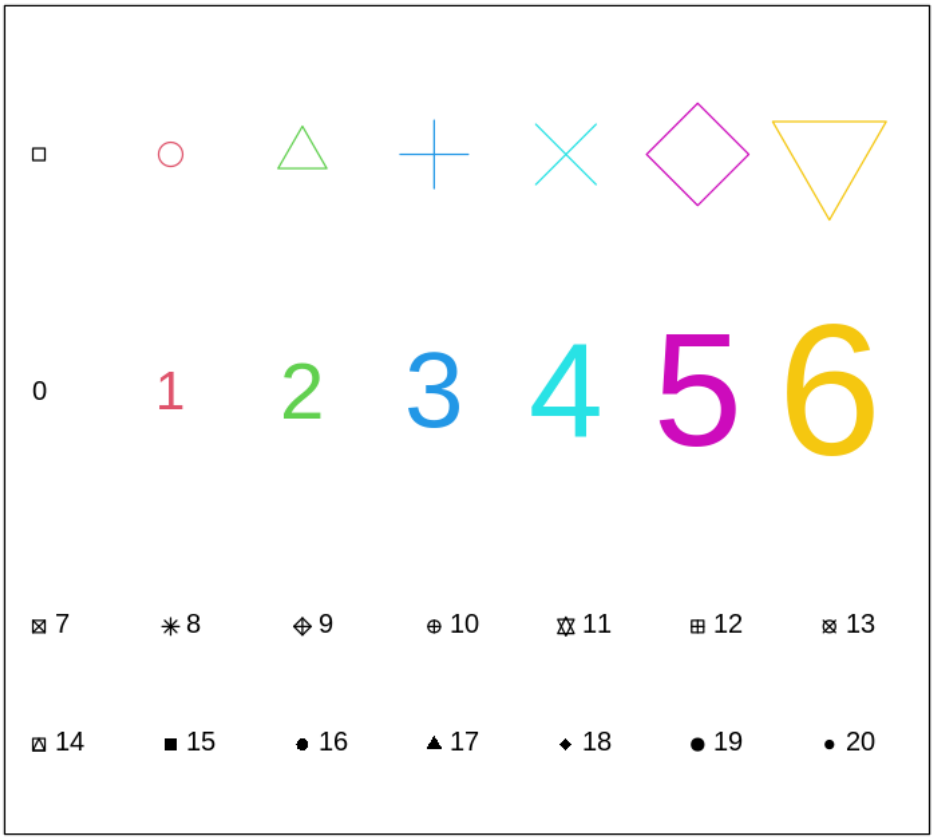
[129]:
```
plot(1, 1, xlim=c(1, 7.5), ylim=c(1.75,5), type="n", axes=F, xlab="",
 ylab="") # Do not plot points
 box()
 points(1:7, rep(4.5, 7), cex=1:7, col=1:7, pch=0:6)
text(1:7,rep(3.5, 7), labels=paste(0:6), cex=1:7, col=1:7)
```



The following, added to the plot that results from the above three statements, demonstrates other choices of pch.

[130]:
```
plot(1, 1, xlim=c(1, 7.5), ylim=c(1.75,5), type="n", axes=F, xlab="",
 ylab="") # Do not plot points
 box()
 points(1:7, rep(4.5, 7), cex=1:7, col=1:7, pch=0:6)
```

```
text(1:7,rep(3.5, 7), labels=paste(0:6), cex=1:7, col=1:7)
points(1:7,rep(2.5,7), pch=(0:6)+7) # Plot symbols 7 to 13
text((1:7), rep(2.5,7), paste((0:6)+7), pos=4) # Label with symbol number
points(1:7,rep(2,7), pch=(0:6)+14) # Plot symbols 14 to 20
text((1:7), rep(2,7), paste((0:6)+14), pos=4) # Labels with symbol number
```



[131]:
```
#A variety of color palettes are available. Here is a function that displays␣
 ↪some of the possibilities:
view.colours <- function(){
plot(1, 1, xlim=c(0,14), ylim=c(0,3), type="n", axes=F,
 xlab="",ylab="")
text(1:6, rep(2.5,6), paste(1:6), col=palette()[1:6], cex=2.5)
text(10, 2.5, "Default palette", adj=0)
```

```
rainchars <- c("R","O","Y","G","B","I","V")
text(1:7, rep(1.5,7), rainchars, col=rainbow(7), cex=2.5)
text(10, 1.5, "rainbow(7)", adj=0)
cmtxt <- substring("cm.colors", 1:9,1:9)
 # Split "cm.colors" into its 9 characters
text(1:9, rep(0.5,9), cmtxt, col=cm.colors(9), cex=3)
text(10, 0.5, "cm.colors(9)", adj=0)
}
#To run the function, enter
view.colours()
```

3.3.2 Adding Text in the Margin

To adds text in the margin of the current plot. The sides are numbered 1(xaxis), 2(y-axis), 3(top)

and 4.

`mtext(side, line, text, ..)`

1.4 3.4 Identification and Location on the Figure Region

identify():

- Draw the graph.
- Call identify().
- Move the cursor near the point you want to identify.
- Click the left mouse button.
- The point will be labeled with its coordinates.

locator():

- Draw the graph.
- Call locator().
- Move the cursor to the location for which you want to know the coordinates.
- Click the left mouse button.
- The coordinates of the clicked location will be printed.

Note:

- Both functions require the user to click the left mouse button.
- identify() labels the points, while locator() only prints their coordinates.

3.4.1 identify()

identify() function lets you label points on a graph by clicking on them. Provide three vectors as input: x, y, and text strings for labels. Example: Plot votes for Buchanan vs. Bush in Florida, then use identify() to label counties.

Short notes: Click on points to label them. Requires x, y, and label vectors. Example: Label counties on a Florida election map.

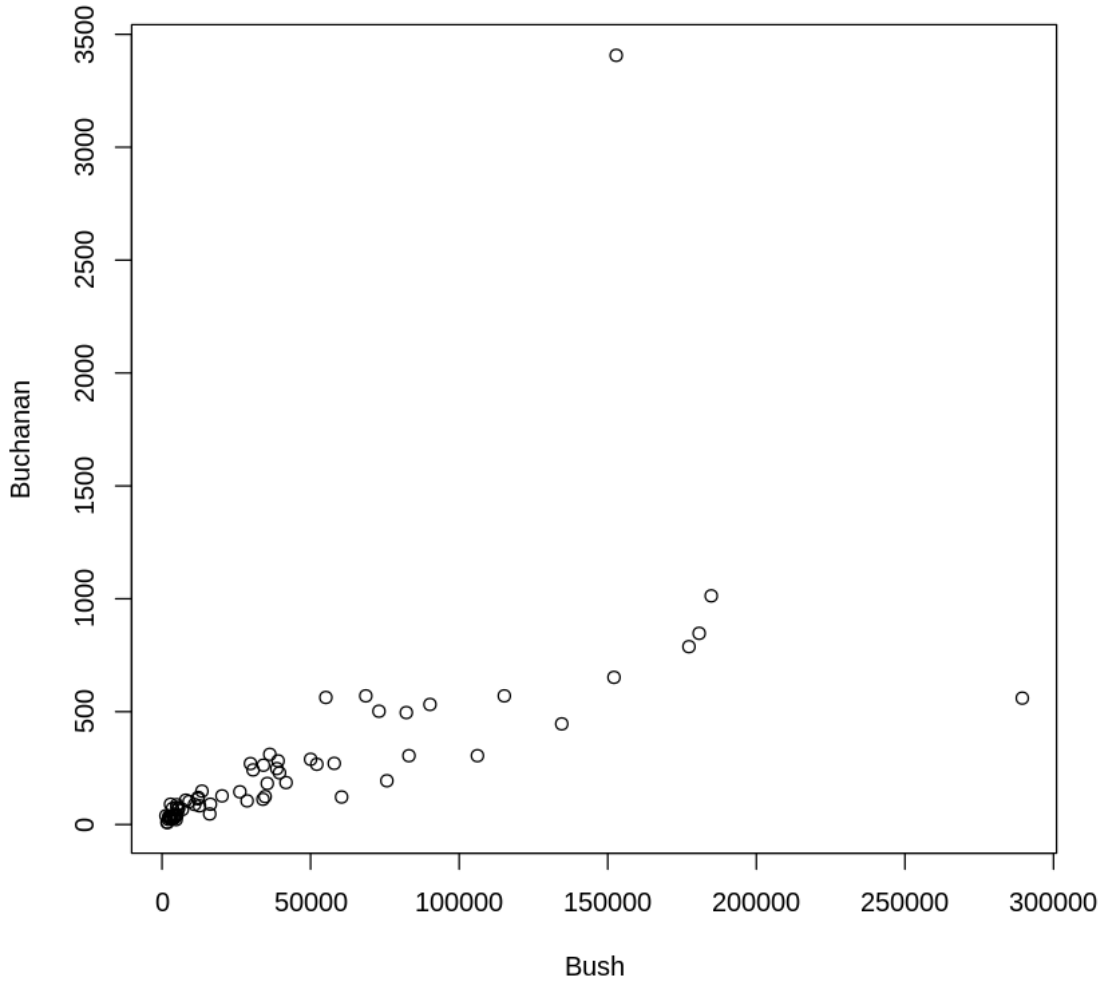
[17]:
```
florida <- read.csv("/content/florida.csv")
florida
```

A data.frame: 67 × 7

county <chr>	Clinton96 <int>	Dole96 <int>	Perot96 <int>	Bush00 <int>	Gore00 <int>	Buchanan00 <int>
Alachua	40144	25303	8072	34124	47365	263
Baker	2273	3684	667	5610	2392	73
Bay	17020	28290	5922	38637	18850	248
Bradford	3356	4038	819	5414	3075	65
Brevard	80416	87980	25249	115185	97318	570
Broward	320736	142834	38964	177323	386561	788
Calhoun	1794	1717	630	2873	2155	90
Charlotte	27121	27836	7783	35426	29645	182
Citrus	22042	20114	7244	29765	25525	270
Clay	13246	30332	3281	41736	14632	186
Collier	23182	42590	6320	60433	29918	122
Columbia	6691	7588	1970	10964	7047	89
Desoto	3219	3272	965	4256	3320	36
Dixie	1731	1398	652	2697	1826	29
Duval	112258	126857	13844	152098	107864	652
Escambia	37768	60839	8587	73017	40943	502
Flagler	9583	8232	2185	12613	13897	83
Franklin	2095	1563	878	2454	2046	33
Gadsden	9405	3813	938	4767	9735	38
Gilchrist	1985	1939	841	3300	1910	29
Glades	1530	1361	521	1841	1442	9
Gulf	2480	2424	1054	3550	2397	71
Hamilton	1734	1518	406	2146	1722	23
Hardee	2417	2926	851	3765	2339	30
Hendry	3882	3855	1135	4747	3240	22
Hernando	28520	22039	7272	30646	32644	242
Highlands	14244	15608	3739	20206	14167	127
Hillsborough	144223	136621	25154	180760	169557	847
Holmes	2310	3248	1208	5011	2177	76
IndianRiver	16373	22709	4635	28635	19768	105
Liberty	868	913	376	1317	1017	39
Madison	2791	2195	578	3038	3014	29
Manatee	41835	44059	10360	57952	49177	271
Marion	37033	41397	11340	55141	44665	563
Martin	20851	28516	5005	33970	26620	112
Miami-Dade	317378	209634	24722	289492	328764	560
Monroe	15219	12021	4817	16059	16483	47
Nassau	7276	12134	1657	16280	6879	90
Okaloosa	16434	40631	5432	52093	16948	267
Okeechobee	4824	3415	1666	5057	4588	43
Orange	105513	106026	18191	134517	140220	446
Osceola	21870	18335	6091	26212	28181	145
PalmBeach	230621	133762	30739	152846	268945	3407
Pasco	66472	48346	18011	68582	69564	570
Pinellas	184728	152125	36990	184823	200629	1013
Polk	66735	67943	14991	90180	75193	532
Putnam	12008	9781	3272	13447	12102	148
SantaRosa	10923	26244	4957	36274	12802	311
Sarasota	63648	69198	14939	83100	72853	305
Seminole	45051	59778	9357	75677	59174	194

[19]:
```
attach(florida)
plot(Bush00, Buchanan00, xlab="Bush", ylab="Buchanan")
identify(Bush00, Buchanan00, county)
```

```
The following objects are masked from florida (pos = 3):

    Buchanan00, Bush00, Clinton96, county, Dole96, Gore00, Perot96


The following objects are masked from florida (pos = 4):

    Buchanan00, Bush00, Clinton96, county, Dole96, Gore00, Perot96
```



[20]:
```
detach(florida)
```

When you're labeling data points, you can click slightly to the left, right, above, or below the point to place the label.

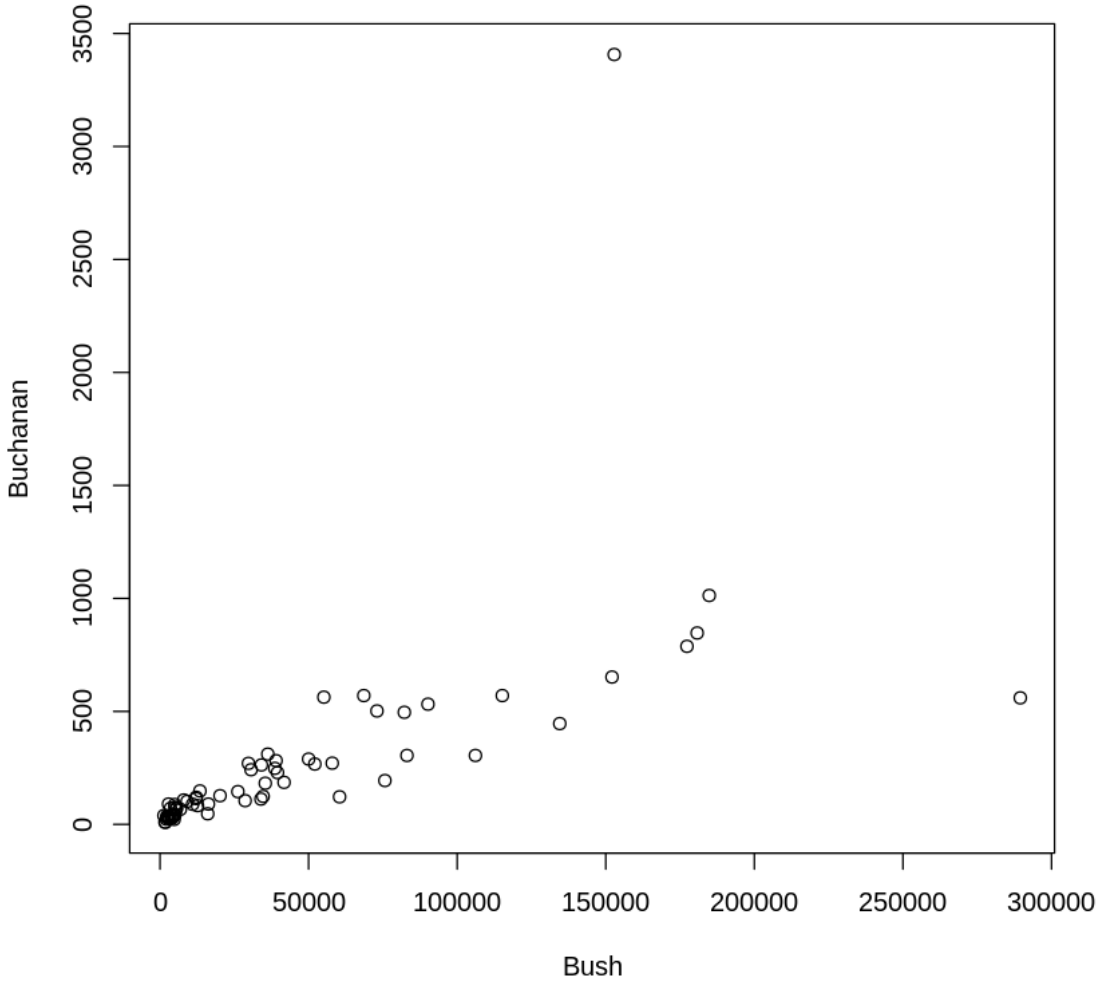
When you're done labeling, right-click to finish. The row numbers of all the data points you labeled will be printed on the screen in the order you labeled them.

3.4.2 locator()

Left click at the locations whose coordinates are required

[21]:
```
attach(florida) # if not already attached
plot(Bush00, Buchanan00, xlab="Bush", ylab="Buchanan")
locator()
```

```
The following objects are masked from florida (pos = 3):

    Buchanan00, Bush00, Clinton96, county, Dole96, Gore00, Perot96


The following objects are masked from florida (pos = 4):

    Buchanan00, Bush00, Clinton96, county, Dole96, Gore00, Perot96
```



[22]:
```
detach(florida)
```

The function can be used to mark new points (specify type=”p”) or lines (specify type=”l”) or both points and lines (specify type=”b”).

1.5 3.5 Plots that show the distribution of data values

We discuss histograms, density plots, boxplots and normal probability plots.

3.5.1 Histograms and density plots

The shapes of histograms depend on the placement of the breaks, as Figure 9 illustrates:

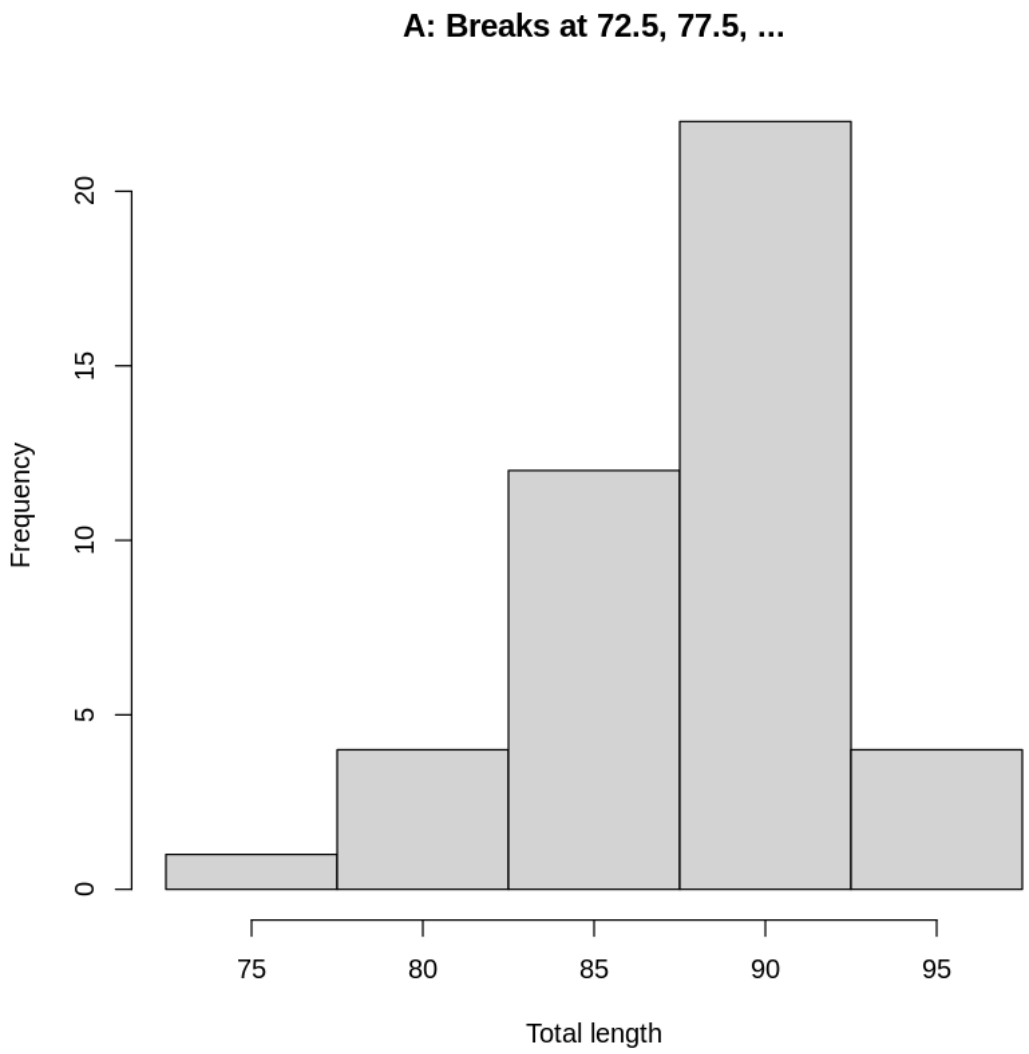
In Figure 9:

Panel A shows a histogram with a frequency scale. This means that the height of each bar represents the number of data points in that bin.

Panel B is drawn with a density scale. This means that the height of each bar represents the proportion of data points in that bin. This allows a density curve to be easily superimposed on the histogram.

Panel C has different breakpoints for the bins. This means that the bins are different sizes. This results in a different impression of the distribution of the data. The density curve is again superimposed on the histogram.

[24]:
```r
#Code to plot the histogram
possum <- read.csv("/content/possum.csv")
attach(possum)
here <- sex == "f"
hist(totlngth[here], breaks = 72.5 + (0:5) * 5, ylim = c(0, 22),
 xlab="Total length", main ="A: Breaks at 72.5, 77.5, ...")
```



Density plots are often preferred over histograms because they are not affected by the choice of breakpoints. This means that they give a more accurate representation of the distribution of the data.

However, density plots do depend on the choice of window width and type. This controls the amount of smoothing applied to the data. A wider window will result in a smoother plot, while a narrower window will result in a more jagged plot.

[25]:
```
plot(density(totlngth[here]),type="l")
```

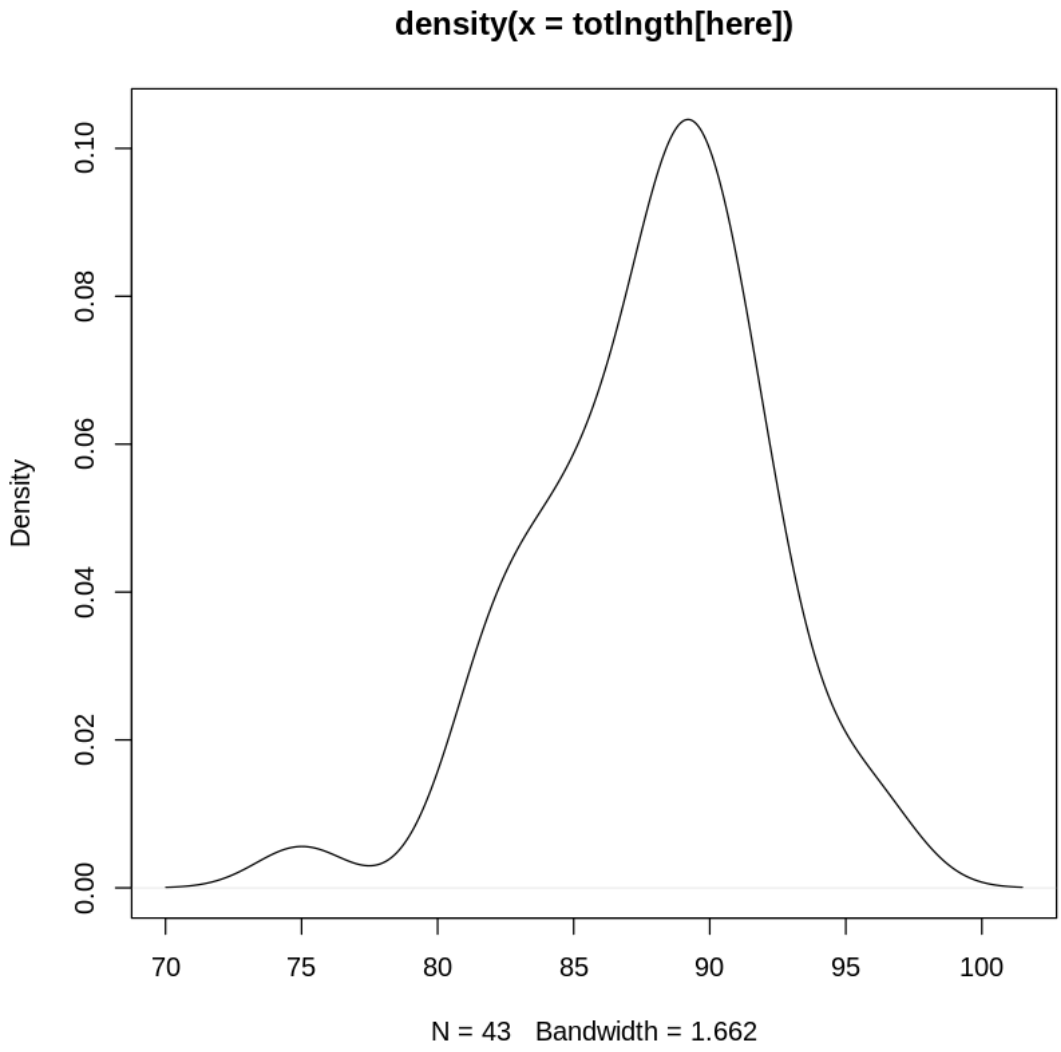
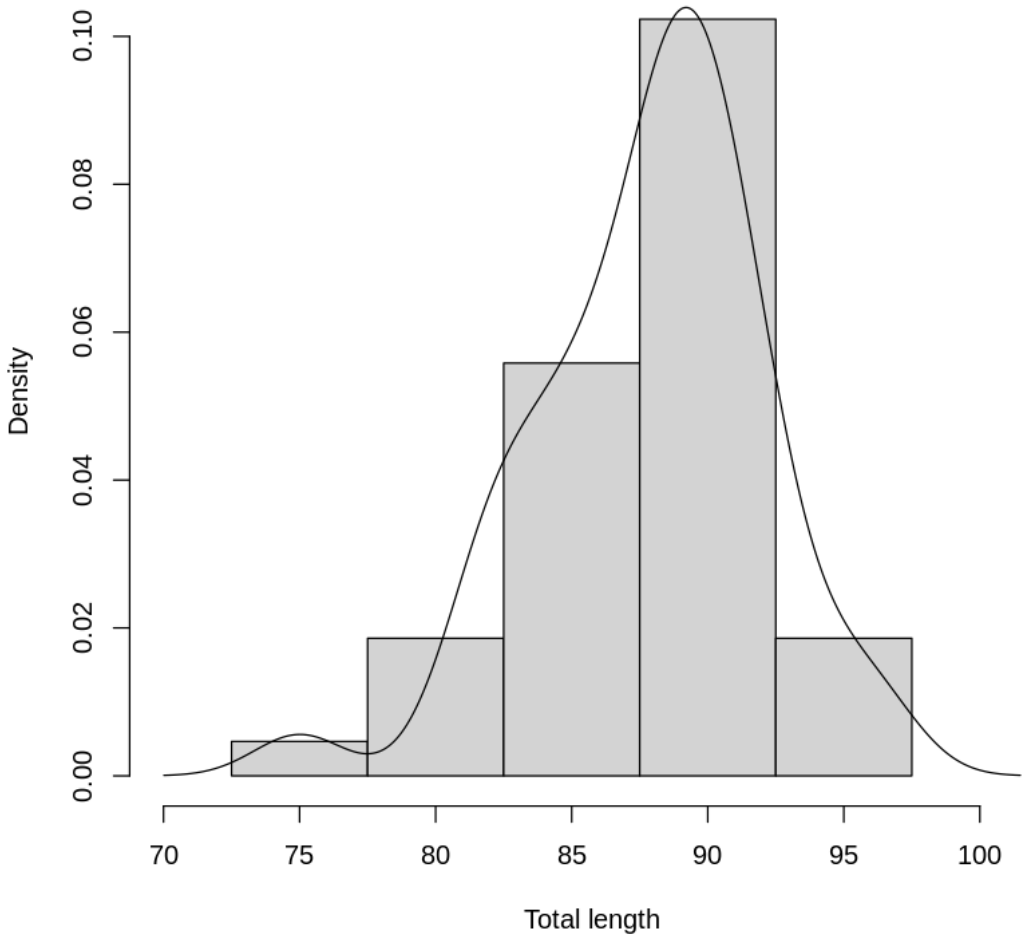


Figure 9B: y-axis labeled for histogram density. Area of rectangle equals frequency divided by width. Suitable for overlaying density curve estimate. Figure 9C uses different breakpoints, altering data distribution impression. Code for Figure 9B.

[26]:
```
here <- sex == "f"
dens <- density(totlngth[here])
xlim <- range(dens$x)
ylim <- range(dens$y)
hist(totlngth[here], breaks = 72.5 + (0:5) * 5, probability = TRUE,
 xlim = xlim, ylim = ylim, xlab="Total length", main="")
lines(dens)
```

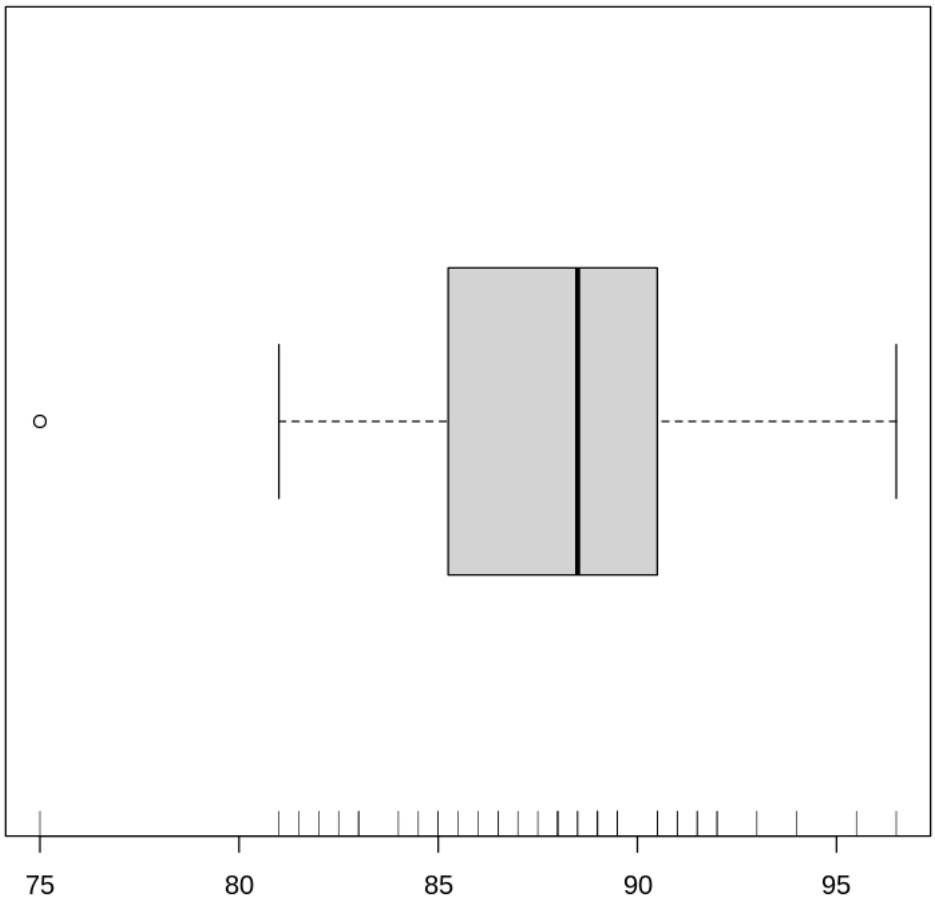


3.5.3 Boxplots

Figure 10: Boxplot of female possum lengths, with additional labelling information.

Code for this boxplot are as:

[29]:
```
boxplot(totlngth[here], horizontal = TRUE)
rug(totlngth[here], side=1)
```



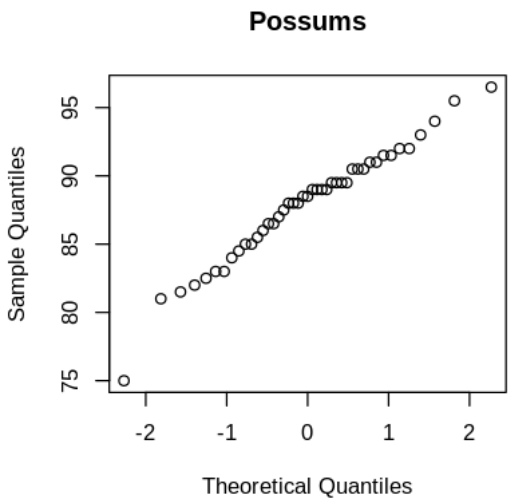
3.5.4 Normal probability plots

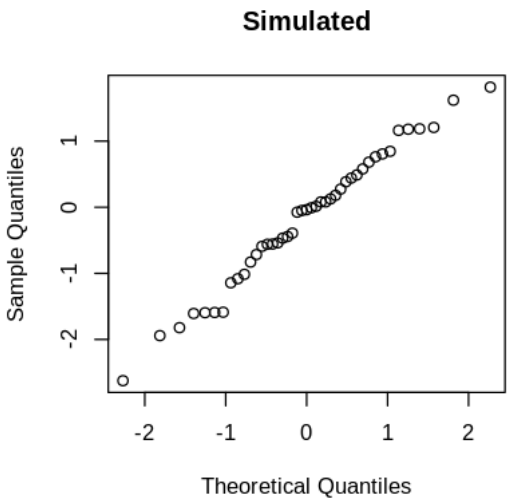
The function qqnorm(y) creates a plot called a normal probability plot for the values in y. If the data follows a normal distribution, the points on this plot will form a straight line. To train the eye to spot deviations from normality, it's useful to generate multiple normal probability plots using random samples of the same size from a normal distribution.

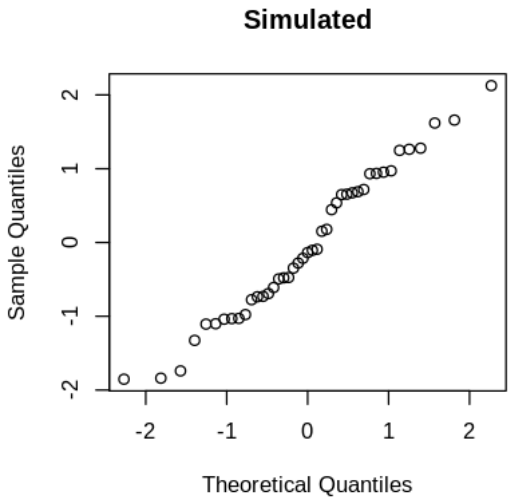
[33]:
```
#x11(width=8, height=6) # This is a better shape for this plot
attach(possum)
here <- sex == "f"
par(mfrow=c(2,2)) # A 2 by 2 layout of plots
y <- totlngth[here]
qqnorm(y, xlab="Theoretical Quantiles", ylab="Sample Quantiles", main="Possums")
for(i in 1:7) {
```

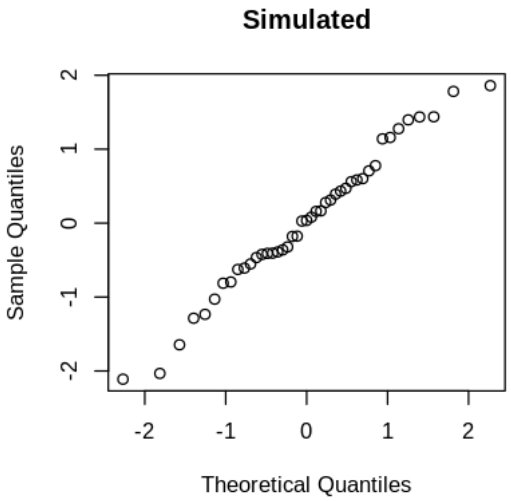
```
  qqnorm(rnorm(43), xlab="Theoretical Quantiles", ylab="Sample Quantiles",
         main="Simulated")
}
```

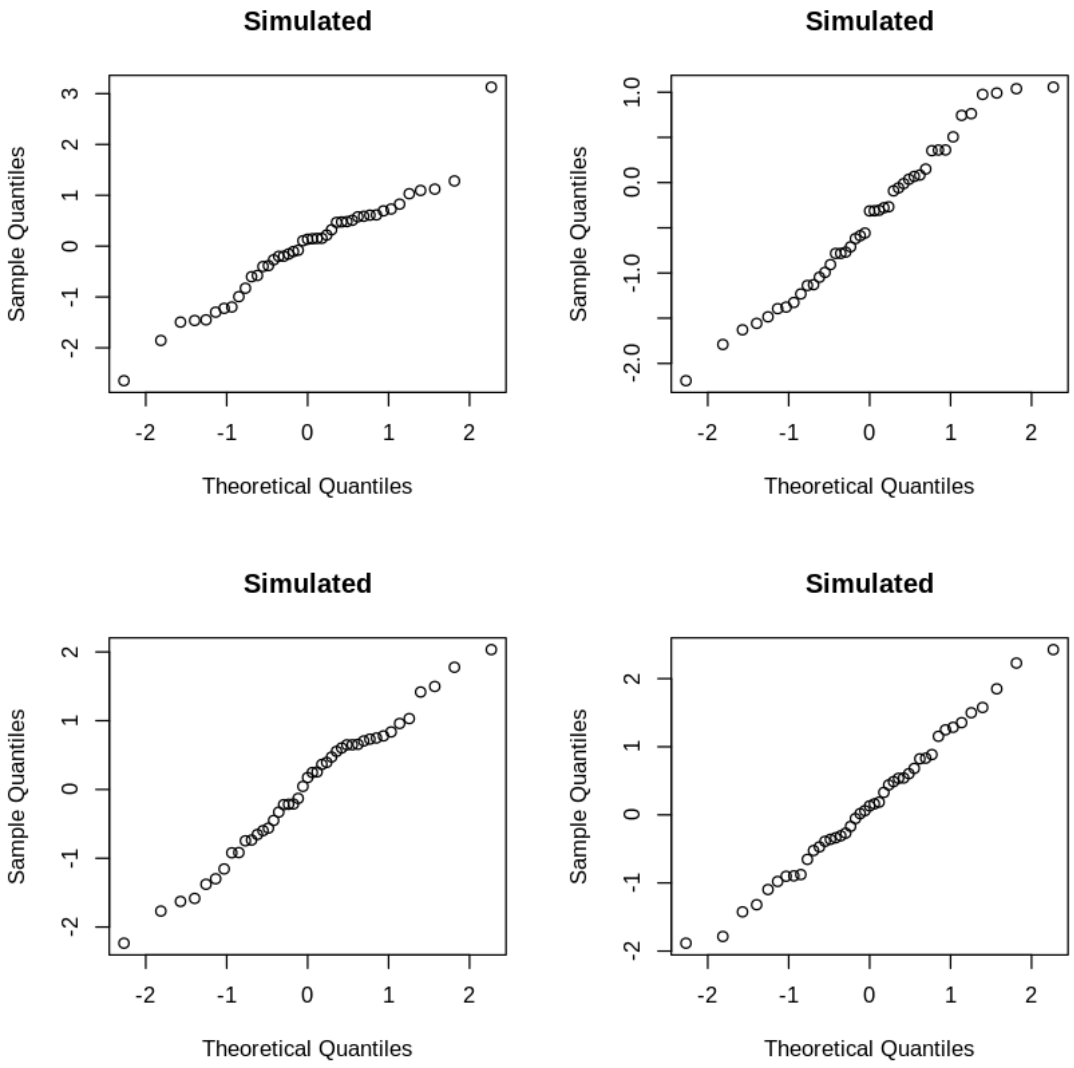
```
The following objects are masked from possum (pos = 3):

    age, belly, case, chest, earconch, eye, footlgth, hdlngth, Pop,
    sex, site, skullw, taill, totlngth


The following objects are masked from possum (pos = 4):

    age, belly, case, chest, earconch, eye, footlgth, hdlngth, Pop,
    sex, site, skullw, taill, totlngth
```


Possums
Sample Quantiles
Theoretical Quantiles
75
80
85
90
95
-2
-1
0
1
2


Simulated
Sample Quantiles
Theoretical Quantiles
-2
-1
0
1
2


Simulated
Sample Quantiles
Theoretical Quantiles
-2
-1
0
1
2


Simulated
Sample Quantiles
Theoretical Quantiles
-2
-1
0
1
2



The concept is significant. To determine if data follows a normal distribution, look at several randomly created samples of equal size from a normal distribution. This process helps develop the ability to visually identify deviations from normality. By default, the function rnorm() produces random samples with an average of 0 and a standard deviation of 1.

[38]:
```
detach(possum)
```

1.6 3.6 Other Useful Plotting Functions

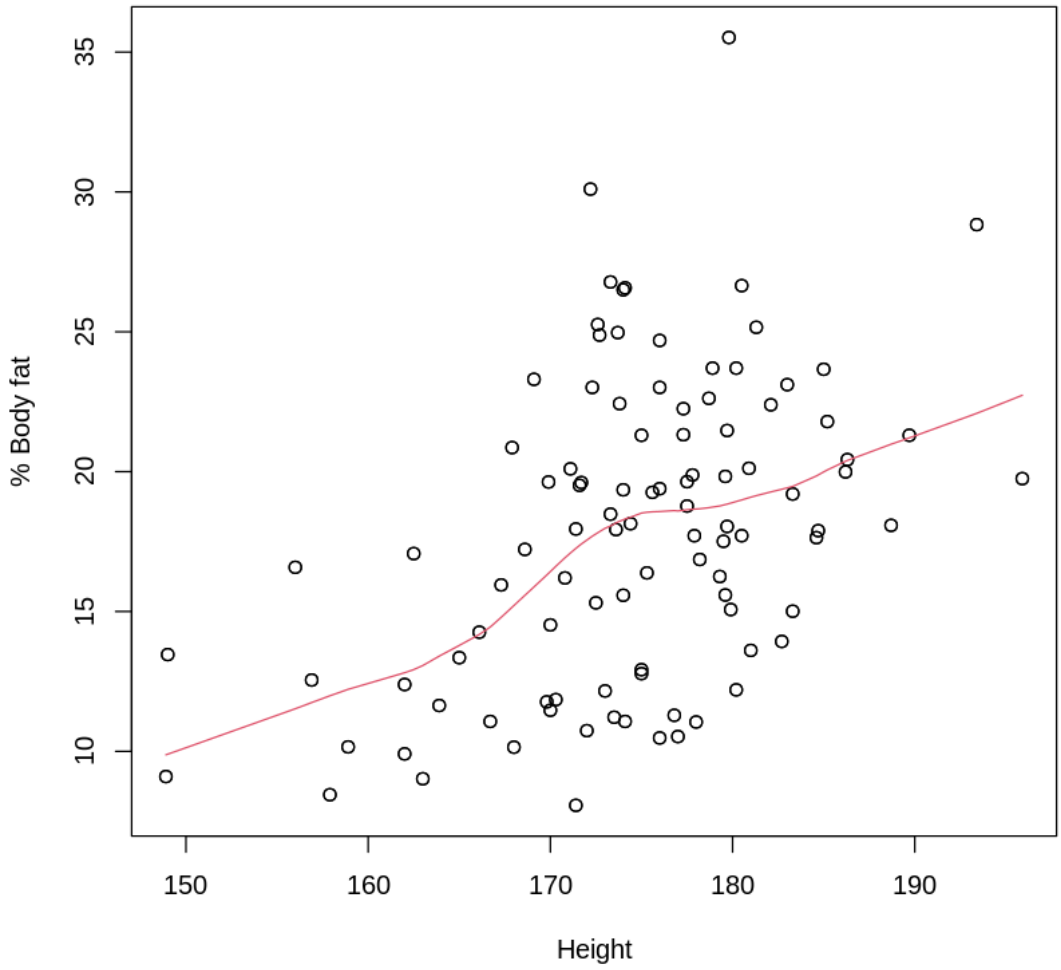
For the functions demonstrated here, we use data on the heights of 100 female athletes.

3.6.1 Scatterplot smoothing

panel.smooth() plots points, then adds a smooth curve through the points. For example:

[41]:
```
ais <- read.csv("/content/ais.csv")
attach(ais)
here <- sex == "f"
plot(pcBfat[here] ~ ht[here], xlab = "Height", ylab = "% Body fat")
panel.smooth(ht[here], pcBfat[here])
```

```
The following objects are masked from ais (pos = 3):

    bmi, ferr, hc, hg, ht, lbm, pcBfat, rcc, rownames, sex, sport, ssf,
    wcc, wt


The following objects are masked from ais (pos = 4):

    bmi, ferr, hc, hg, ht, lbm, pcBfat, rcc, rownames, sex, sport, ssf,
    wcc, wt


The following objects are masked from ais (pos = 5):

    bmi, ferr, hc, hg, ht, lbm, pcBfat, rcc, rownames, sex, sport, ssf,
    wcc, wt


The following objects are masked from ais (pos = 6):

    bmi, ferr, hc, hg, ht, lbm, pcBfat, rcc, rownames, sex, sport, ssf,
    wcc, wt


The following object is masked from possum (pos = 7):

    sex


The following object is masked from possum (pos = 8):

    sex
```



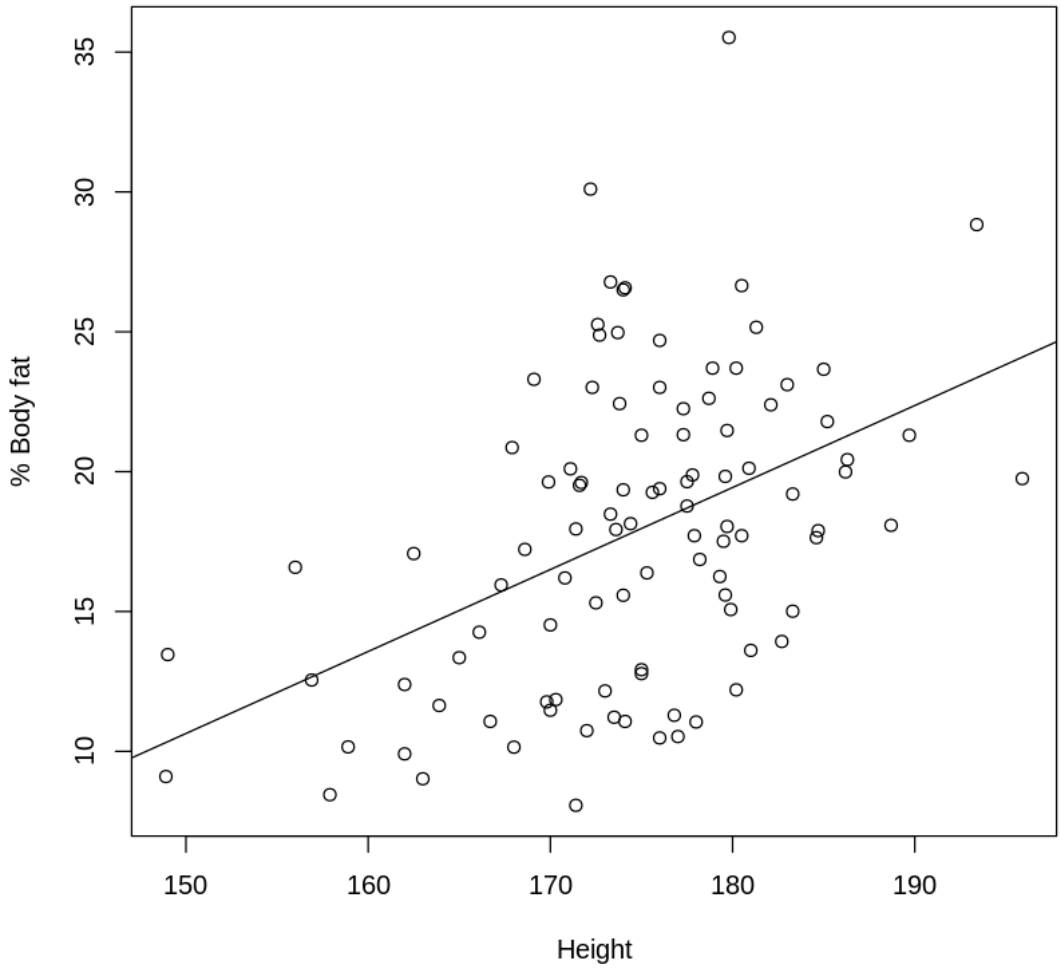
3.6.2 Adding lines to plots

You can employ the function abline() for this purpose. You can specify parameters such as an intercept and slope, a vector containing the intercept and slope, or an lm object. Additionally, you have the option to draw a horizontal line using the argument h = , or a vertical line using v = .

[43]:
```
attach(ais)
here<- sex=="f"
plot(pcBfat[here] ~ ht[here], xlab = "Height", ylab = "% Body fat")
abline(lm(pcBfat[here] ~ ht[here]))
```

```
The following objects are masked from ais (pos = 3):

    bmi, ferr, hc, hg, ht, lbm, pcBfat, rcc, rownames, sex, sport, ssf,
```

```
    wcc, wt


The following objects are masked from ais (pos = 4):

    bmi, ferr, hc, hg, ht, lbm, pcBfat, rcc, rownames, sex, sport, ssf,
    wcc, wt


The following objects are masked from ais (pos = 5):

    bmi, ferr, hc, hg, ht, lbm, pcBfat, rcc, rownames, sex, sport, ssf,
    wcc, wt


The following objects are masked from ais (pos = 6):

    bmi, ferr, hc, hg, ht, lbm, pcBfat, rcc, rownames, sex, sport, ssf,
    wcc, wt


The following objects are masked from ais (pos = 7):

    bmi, ferr, hc, hg, ht, lbm, pcBfat, rcc, rownames, sex, sport, ssf,
    wcc, wt


The following object is masked from possum (pos = 8):

    sex


The following object is masked from possum (pos = 9):

    sex
```

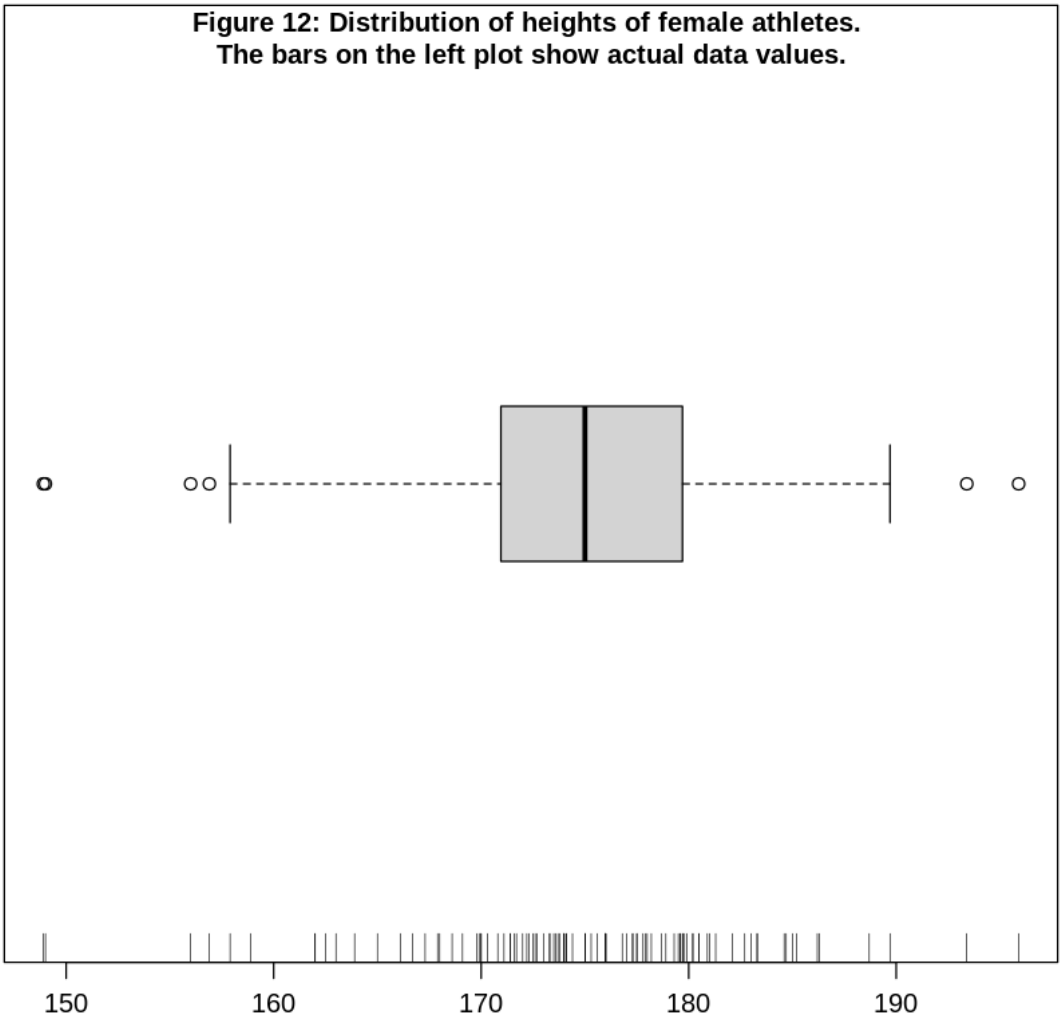


3.6.3 Rugplots

By default, when you use the function rug(x), it places vertical bars along the x-axis of the current plot to display the distribution of values of x. However, it can also be handy for displaying the actual values alongside a boxplot. In Figure 12, there's a boxplot depicting the height distribution of female athletes, with a rugplot added along the y-axis to show the actual values.

[56]:
```
here <- sex == "f"
oldpar <- par(mar=c(4.1,1.1,1.1, 1.1), mgp=c(2.25, 0.75,0))
boxplot(ht[here], boxwex = 0.35, ylab = "Height(cm)", horizontal=TRUE)
rug(ht[here], side = 1)
par(oldpar)
#The parameter boxwex controls the width of the boxplot.
mtext("Figure 12: Distribution of heights of female athletes.
```

```
The bars on the left plot show actual data values.", side = 3, line = 1, font =␣
↪2)
```



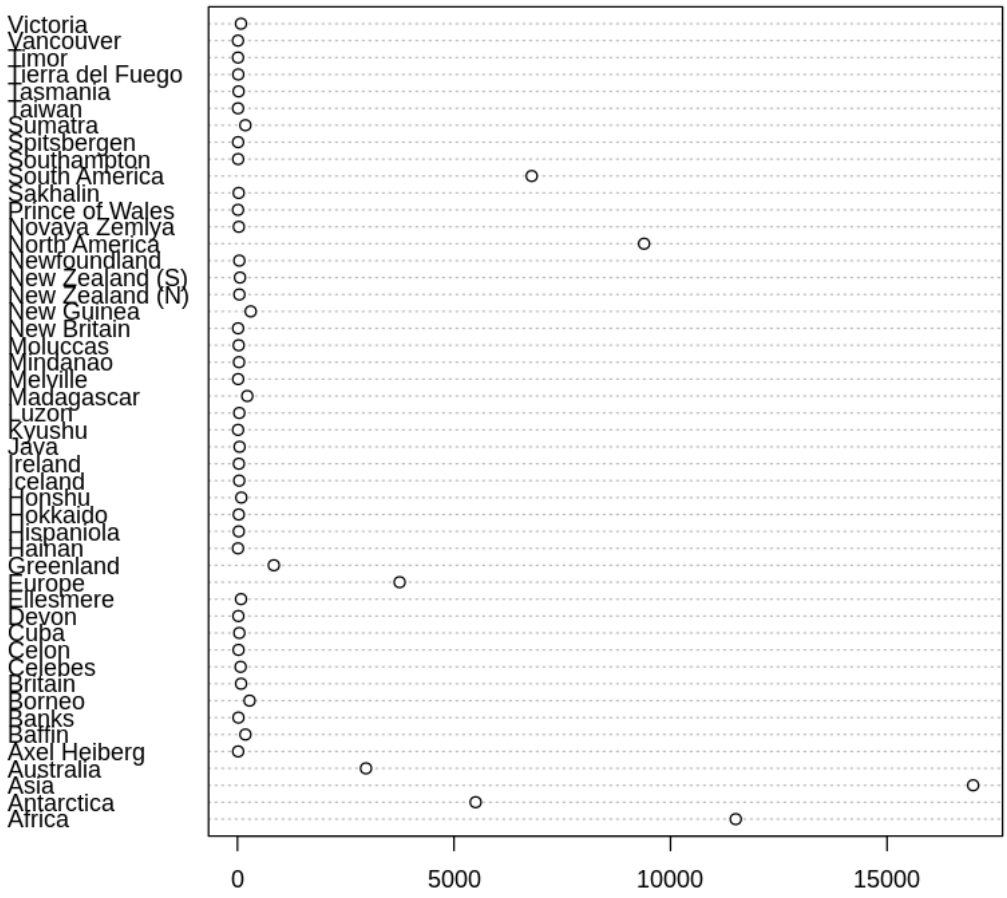
3.6.4 Scatterplot matrices

Section 2.1.3 demonstrated the use of the pairs() function.

3.6.5 Dotcharts

Dotcharts can be a great substitute for bar charts. They provide a higher information-to-ink ratio. You can try creating a dot chart using the command `dotchart(islands)`. However, if there are many names or substantial overlap, the points might become too small to see.

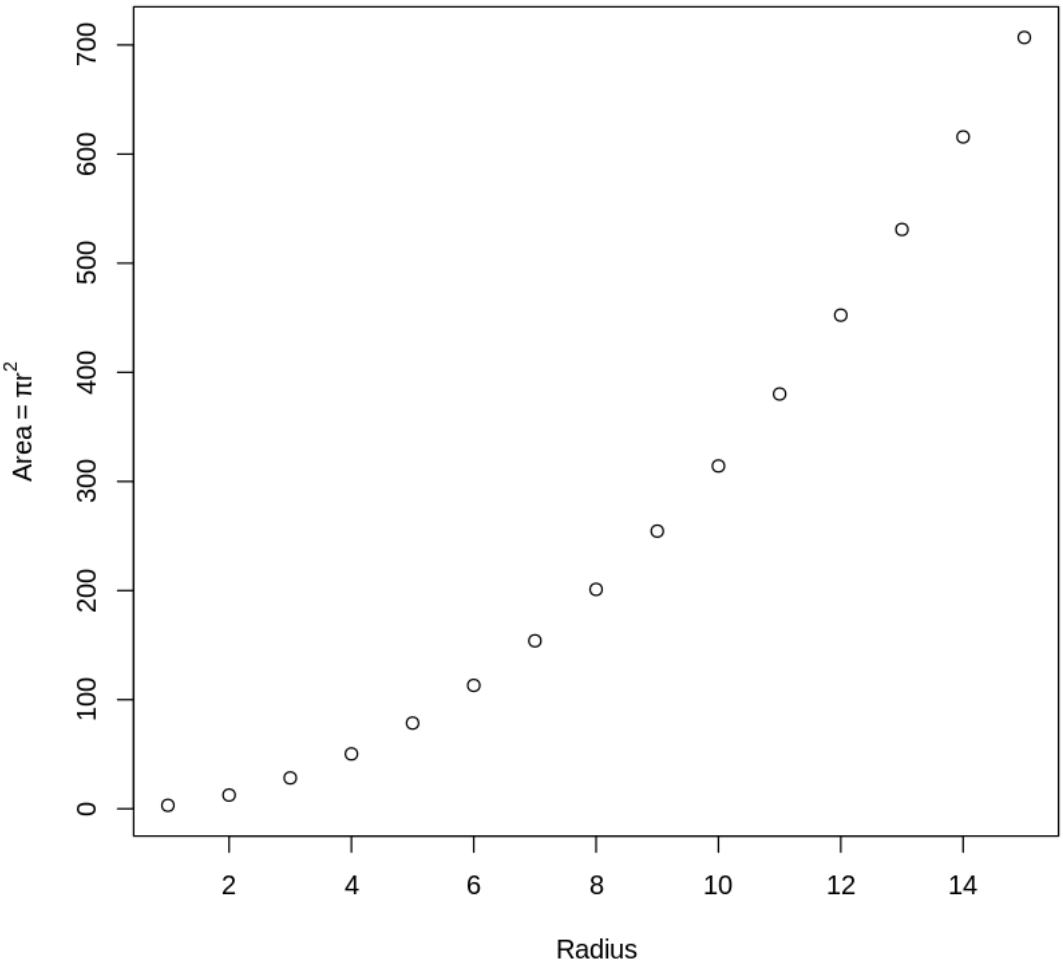
[61]:
```
dotchart(islands,cex=0.9)
```



1.7 3.7 Plotting Mathematical Symbols

Both text() and mtext() will take an expression rather than a text string. In plot(), either or both of xlab and ylab can be an expression

[63]:
```
x <- 1:15
y <- pi * x^2  # y is defined as the area, pi * r^2
plot(x, y, xlab="Radius", ylab=expression(Area == pi*r^2))
```



In the expression `expression(Area == pi*r^2)`, you might notice a double equals sign (`==`). However, when this expression is displayed on the plot, it will appear as `Area = pi*r^2`, with a single equals sign. For more detailed information on how mathematical expressions are plotted in R, you can refer to the documentation by typing

```
help(plotmath)
```

Additionally, there's a related example in Chapter 12. If you run

```
demo(graphics)
```

you'll see the final plot, which showcases various possibilities for plotting mathematical symbols.

1.8 3.8 Guidelines for Graphs

1. Design graphs to convey the message clearly and concisely, minimizing ink usage. Label important features adequately.
2. In scatterplots, use solid, conspicuous points when there's little overlap. When overlap is extensive, opt for open symbols to achieve high ink density.
3. Prefer scatterplots over bar graphs when the horizontal axis represents a quantitative effect.
4. Prioritize graphs that allow direct and easy information reading over those relying on visual impression or perspective.
5. For scientific papers, favor contour plots over surface plots or 2D bar graphs for clarity.
6. Ensure graphs maintain clarity even after reduction and reproduction.
7. Clearly explain the interpretation of error bars—whether they represent SE limits, 95% confidence interval, SD limits, etc., and specify the source of errors.
8. Use color or distinct symbols to differentiate between different groups, ensuring the colors contrast well.

1.9 3.9 Exercises

1. The data set huron that accompanies these notes has mean July average water surface elevations, in feet, 30 IGLD (1955) for Harbor Beach, Michigan, on Lake Huron, Station 5014, for 1860-198622. (Alternatively work with the vector LakeHuron from the datasets package, that has mean heights for 1875-1772 only.)

a) Plot mean.height against year.

b) Use the identify function to label points for the years that correspond to the lowest and highest mean levels. That is, type

```
identify(huron$year,huron$mean.height,labels=huron$year)
```

and use the left mouse button to click on the lowest point and highest point on the plot. To quit, press both mouse buttons simultaneously.

c) As in the case of many time series, the mean levels are correlated from year to year. To see how each year's mean level is related to the previous year's mean level, use

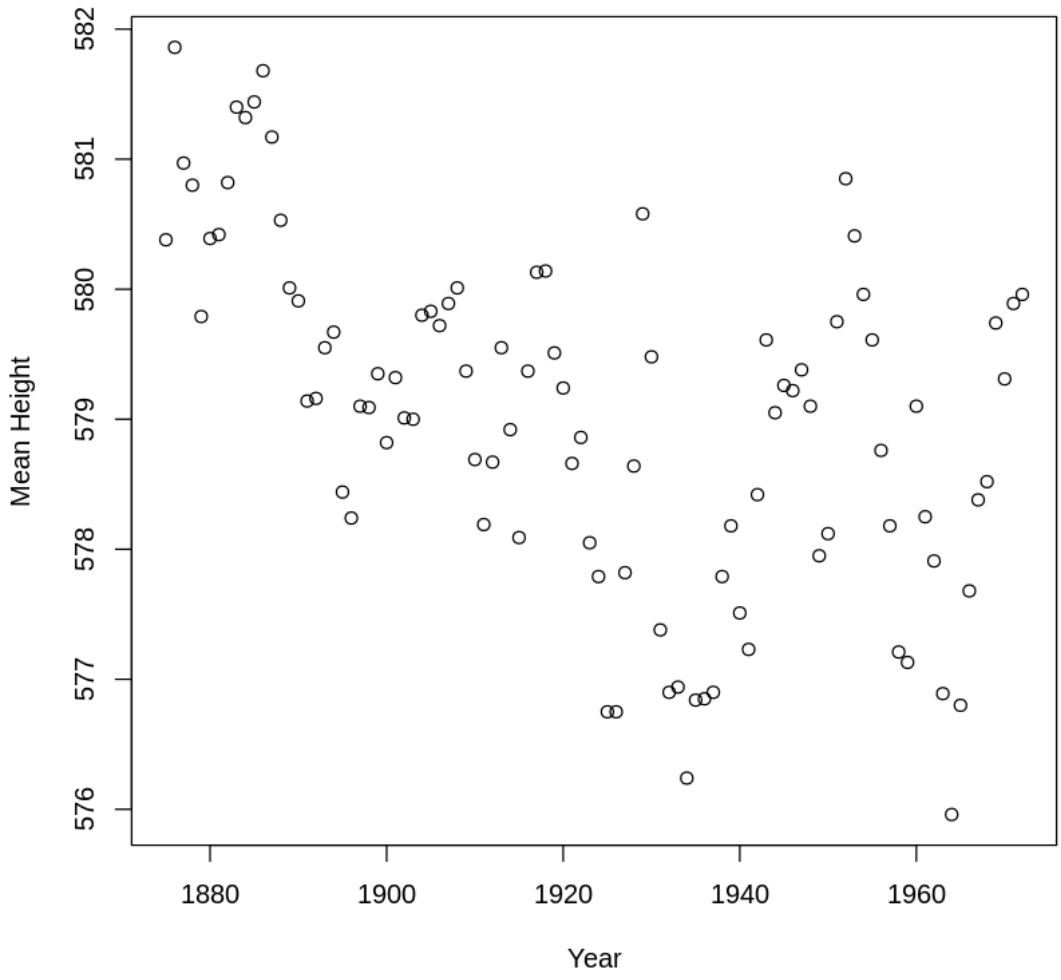
```
lag.plot(huron$mean.height)
```

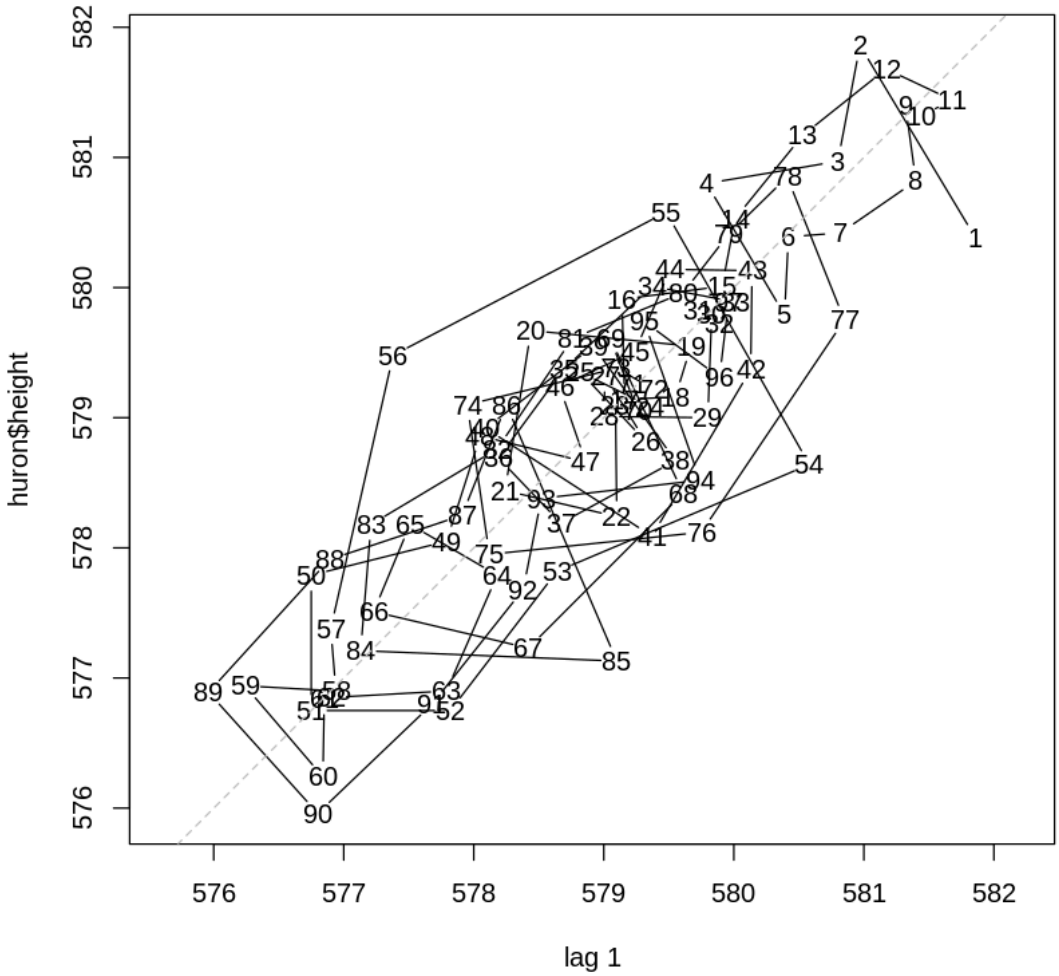
This plots the mean level at year i against the mean level at year i-1

[71]:
```
# Load the dataset huron or use the vector LakeHuron from the datasets package
# Plot mean.height against year
huron <- read.csv("/content/LakeHuron.csv")
colnames(huron) <- c("id","year", "height")
huron
```

id	year	height
<int>	<int>	<dbl>
1	1875	580.38
2	1876	581.86
3	1877	580.97
4	1878	580.80
5	1879	579.79
6	1880	580.39
7	1881	580.42
8	1882	580.82
9	1883	581.40
10	1884	581.32
11	1885	581.44
12	1886	581.68
13	1887	581.17
14	1888	580.53
15	1889	580.01
16	1890	579.91
17	1891	579.14
18	1892	579.16
19	1893	579.55
20	1894	579.67
21	1895	578.44
22	1896	578.24
23	1897	579.10
24	1898	579.09
25	1899	579.35
26	1900	578.82
27	1901	579.32
28	1902	579.01
29	1903	579.00
A data.frame: 98 × 3 30	1904	579.80
69	1943	579.61
70	1944	579.05
71	1945	579.26
72	1946	579.22
73	1947	579.38
74	1948	579.10
75	1949	577.95
76	1950	578.12
77	1951	579.75
78	1952	580.85
79	1953	580.41
80	1954	579.96
81	1955	579.61
82	1956	578.76
83	1957	578.18
84	1958	577.21
85	1959	577.13
86	1960	579.10
87	1961	578.25
88	1962	577.91

[81]:
```r
# Load the dataset huron or use the vector LakeHuron from the datasets package
# Plot mean height against year
plot(huron$year, huron$height, xlab = "Year", ylab = "Mean Height")

# Use identify function to label points for the lowest and highest mean levels
identify(huron$year, huron$height, labels = huron$year)

# Use lag.plot to visualize correlation between mean levels from year to year
lag.plot(huron$height)
```



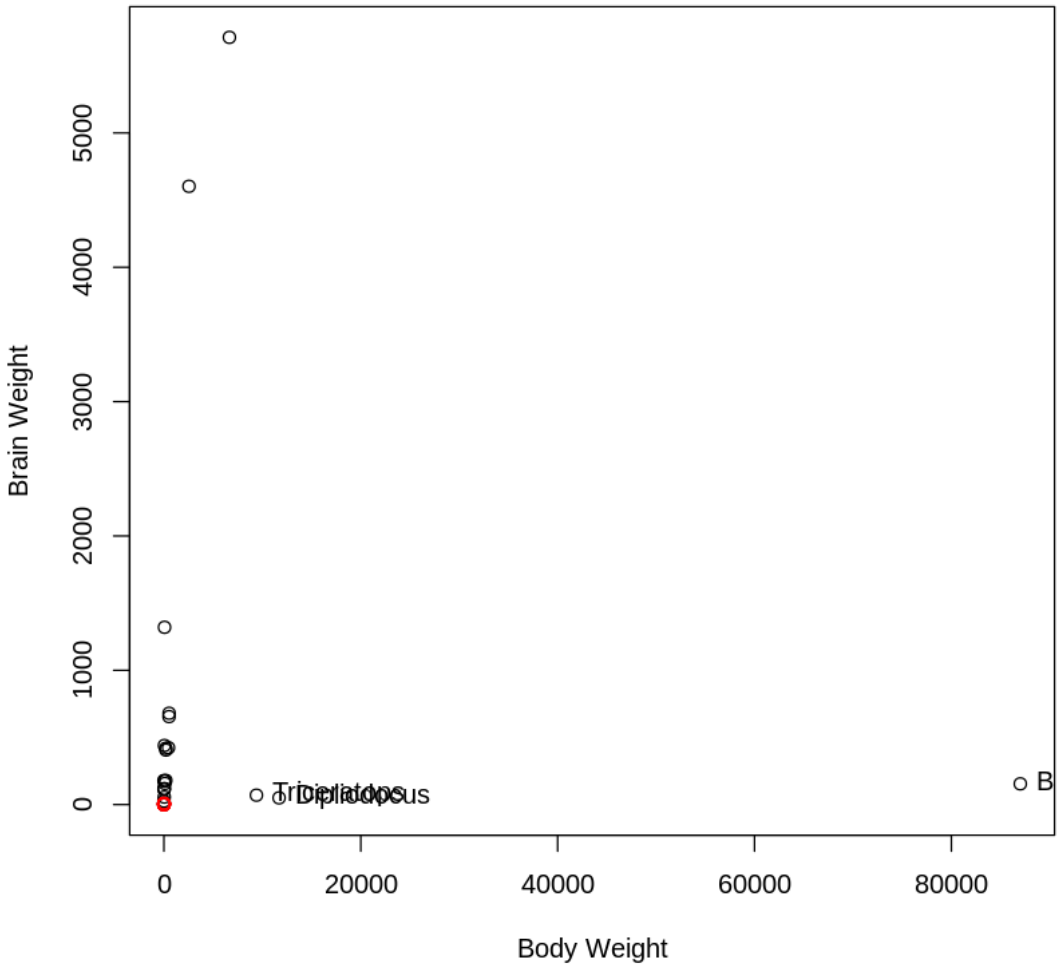


2. Plot the graph of brain weight (brain) versus body weight (body) for the data set Animals from the MASS package. Label the axes appropriately. Alongside on the same graphics page, log(brain weight) versus log(body weight). Use the row labels to label the points with the three largest body weight values.

Label the axes in untransformed units. [To access this data frame, specify library(MASS)]

[82]:
```
library(MASS)

# Load the Animals dataset
data(Animals)

# Plot brain weight (brain) versus body weight (body)
plot(Animals$body, Animals$brain, xlab = "Body Weight", ylab = "Brain Weight")
```

```r
# Log transformation for brain and body weights
log_brain <- log(Animals$brain)
log_body <- log(Animals$body)

# Plot log(brain weight) versus log(body weight)
points(log_body, log_brain, col = "red")

# Identify the row labels for the three largest body weight values
largest_bodies <- tail(order(Animals$body), 3)
text(Animals$body[largest_bodies], Animals$brain[largest_bodies], labels = row.
  ↪names(Animals)[largest_bodies], pos = 4)
```



3. Check the distributions of head lengths (hdlngth) in the possum23 data set that accompanies these notes. Compare the following forms of display:

a) a histogram (hist(possum$hdlngth));

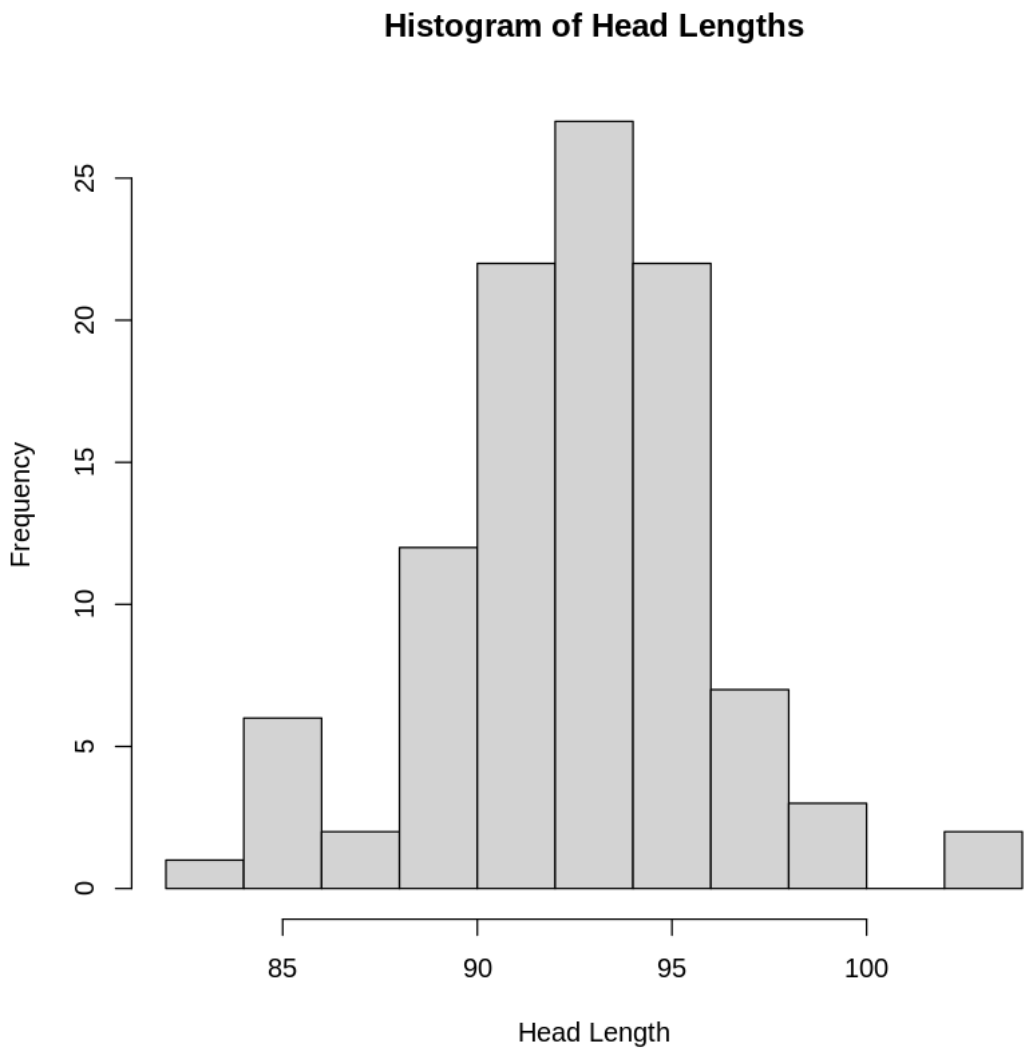
b) a stem and leaf plot (stem(qqnorm(possum$hdlngth));

c) a normal probability plot (qqnorm(possum$hdlngth)); and

d) a density plot (plot(density(possum$hdlngth)).

What are the advantages and disadvantages of these different forms of display?

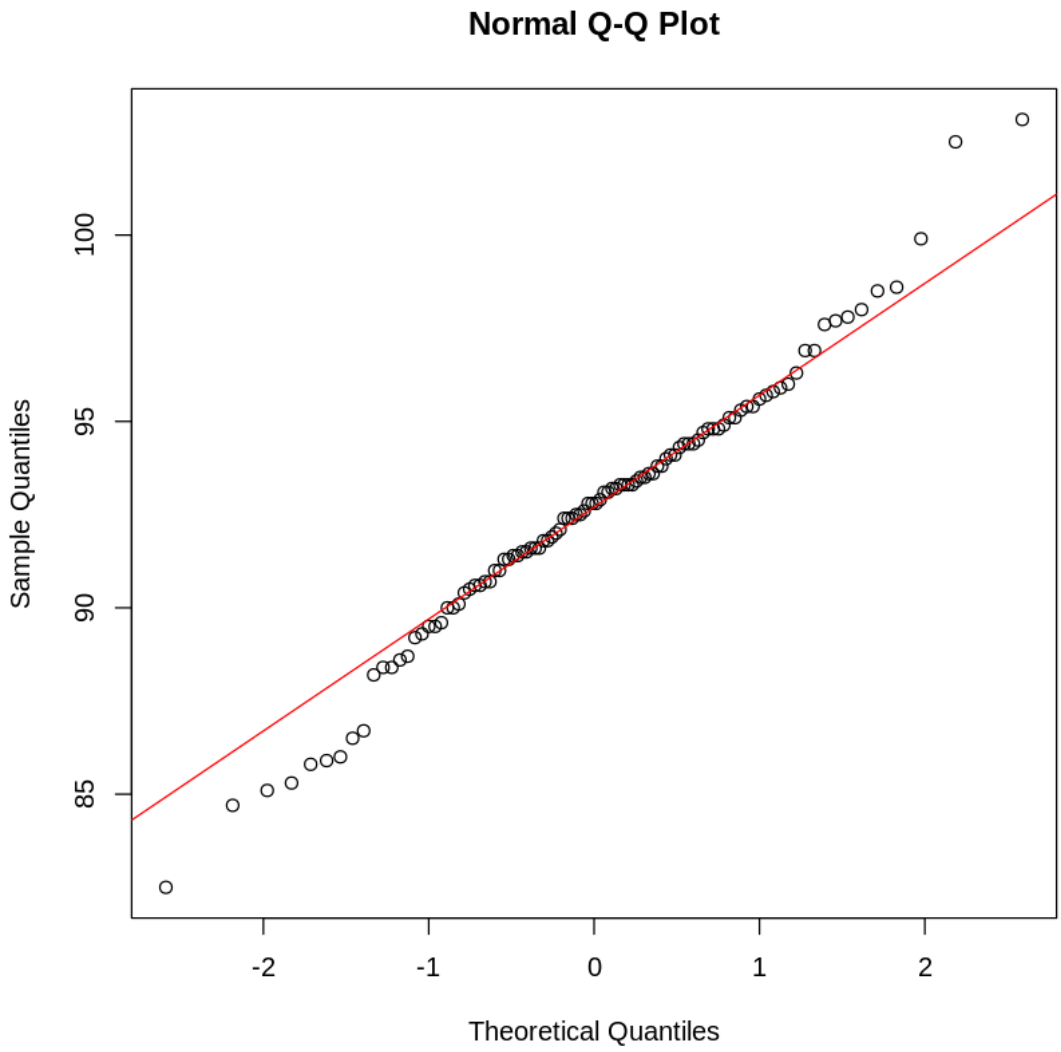
[84]:
```
# Plot histogram
hist(possum$hdlngth, main = "Histogram of Head Lengths", xlab = "Head Length")
```



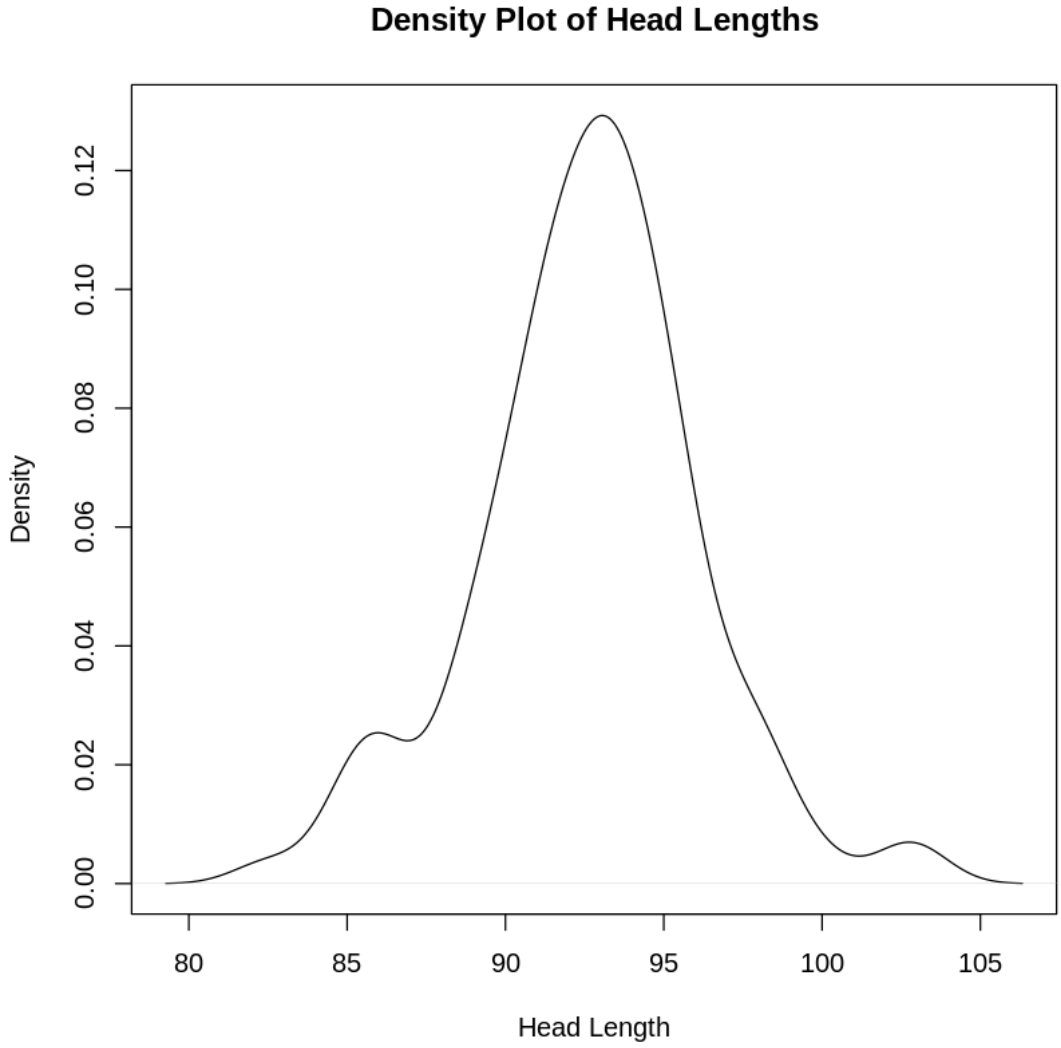
[85]:
```
# Stem and leaf plot
stem(possum$hdlngth)
cat("\n")
```

```
The decimal point is at the |

 82 | 5
 83 |
 84 | 7
 85 | 1389
 86 | 057
 87 |
 88 | 24467
 89 | 23556
 90 | 001456677
 91 | 00334455666889
 92 | 014445568889
 93 | 112233334556688
 94 | 0113444578889
 95 | 113446789
 96 | 0399
 97 | 678
 98 | 056
 99 | 9
100 |
101 |
102 | 5
103 | 1
```

[86]:
```
# Normal probability plot
qqnorm(possum$hdlngth)
qqline(possum$hdlngth, col = "red")
cat("\n")
```



[87]:
```
# Density plot
plot(density(possum$hdlngth), main = "Density Plot of Head Lengths", xlab =␣
↪"Head Length", ylab = "Density")
```



Explanation:

- **Histogram:** Shows the distribution of head lengths in a visually appealing way but may hide specific data patterns in large datasets.
- **Stem and leaf plot:** Provides a detailed view of the distribution, including individual data points, but may be cumbersome to interpret for larger datasets.
- **Normal probability plot:** Useful for assessing if the data follows a normal distribution, but less intuitive for general distribution exploration.
- **Density plot:** Offers a smooth representation of the distribution, allowing for easy identification of peaks and valleys, but may not provide detailed information on individual data points.

Advantages and Disadvantages:

- **Histogram:** Easy to understand and visually appealing, but can obscure details in large

datasets.

- **Stem and leaf plot:** Shows individual data points and provides detailed distribution information, but can be cumbersome for large datasets.
- **Normal probability plot:** Useful for assessing normality, but less intuitive for general distribution exploration.
- **Density plot:** Provides a smooth representation of the distribution and highlights peaks and valleys, but may not show individual data points.

4. Try x <- rnorm(10). Print out the numbers that you get. Look up the help for rnorm. Now generate a sample of size 10 from a normal distribution with mean 170 and standard deviation 4.

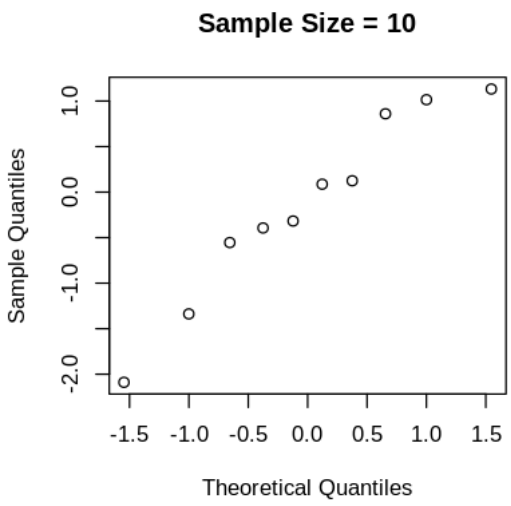
[88]:
```
# Generate a sample of size 10 from a standard normal distribution
x <- rnorm(10)
print(x)

# Generate a sample of size 10 from a normal distribution with mean 170 and␣
 ↪standard deviation 4
y <- rnorm(10, mean = 170, sd = 4)
print(y)
```

```
 [1]  0.68482092 -1.96163576  1.25173954 -0.47359826 -0.17239942  0.07485734
 [7]  0.95463300 -0.87368313  1.59637430  1.92521319
 [1] 174.3178 169.9995 168.3403 174.6476 169.4601 171.9097 164.1072 168.3917
 [9] 167.6327 170.0206
```

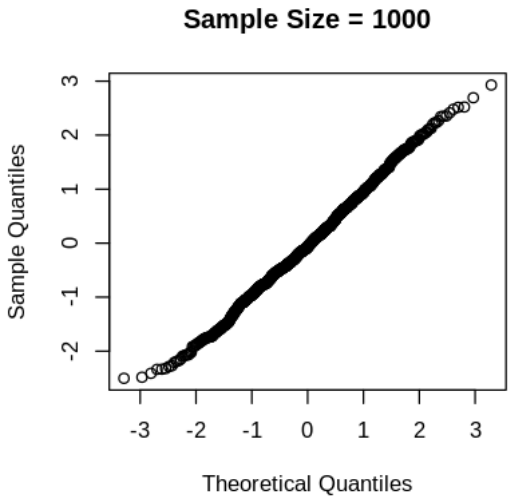
5. Use mfrow() to set up the layout for a 3 by 4 array of plots. In the top 4 rows, show normal probability plots (section 3.4.2) for four separate 'random' samples of size 10, all from a normal distribution. In the middle 4 rows, display plots for samples of size 100. In the bottom four rows, display plots for samples of size 1000. Comment on how the appearance of the plots changes as the sample size changes.

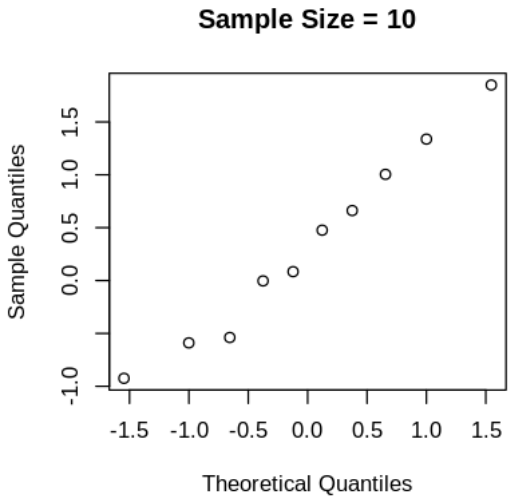
[90]:
```
# Set up the layout for a 3 by 4 array of plots
par(mfrow = c(2, 2))

# Generate and display normal probability plots for samples of size 10, 100,␣
 ↪and 1000
for (i in 1:4) {
  # Sample size 10
  x_10 <- rnorm(10)
  qqnorm(x_10, main = "Sample Size = 10")

  # Sample size 100
  x_100 <- rnorm(100)
  qqnorm(x_100, main = "Sample Size = 100")

  # Sample size 1000
  x_1000 <- rnorm(1000)
  qqnorm(x_1000, main = "Sample Size = 1000")
```

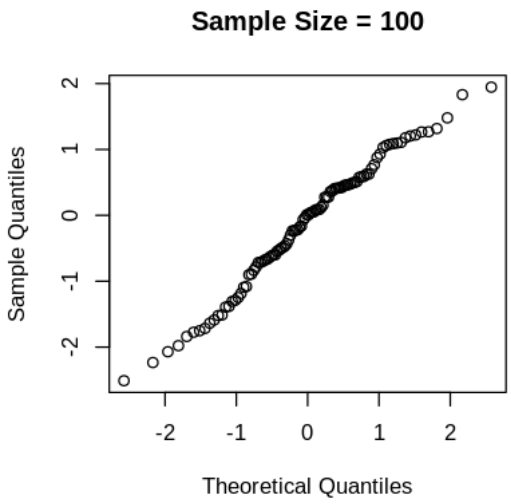
```
}
```

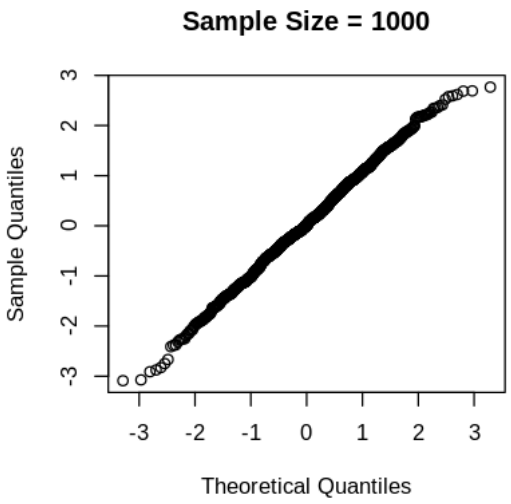

Sample Size = 10
Sample Quantiles
Theoretical Quantiles
-2.0
-1.0
0.0
1.0
-1.5
-1.0
-0.5
0.0
0.5
1.0
1.5

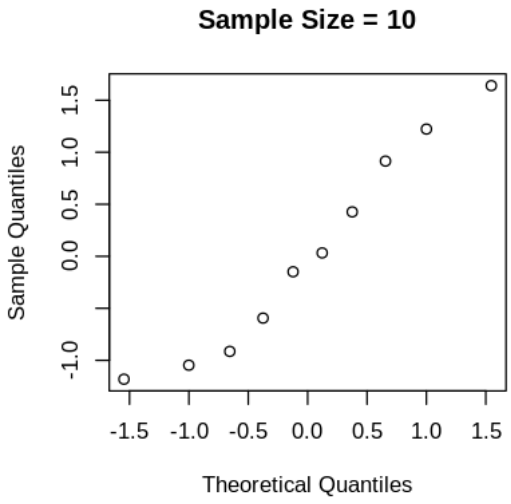

Sample Size = 100
Sample Quantiles
Theoretical Quantiles
-3
-2
-1
0
1
2
3


Sample Size = 1000
Sample Quantiles
Theoretical Quantiles
-2
-1
0
1
2
3

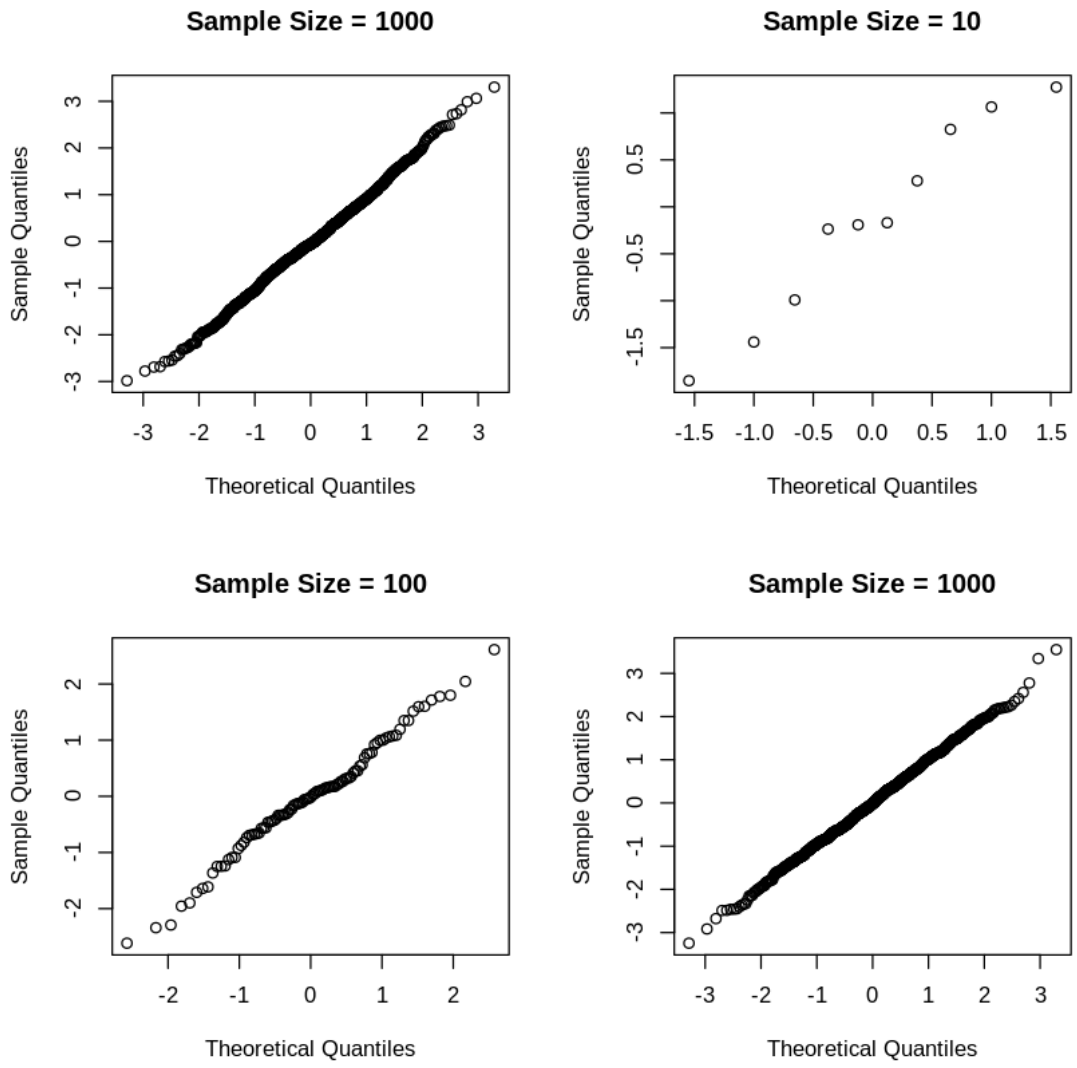

Sample Size = 10
Sample Quantiles
Theoretical Quantiles
-1.0
0.0
0.5
1.0
1.5


Sample Size = 100
Sample Quantiles
Theoretical Quantiles
-2
-1
0
1
2


Sample Size = 1000
Sample Quantiles
Theoretical Quantiles
-3
-2
-1
0
1
2
3


Sample Size = 10
Sample Quantiles
Theoretical Quantiles
-1.0
0.0
0.5
1.0
1.5


Sample Size = 100
Sample Quantiles
Theoretical Quantiles
-2
-1
0
1
2



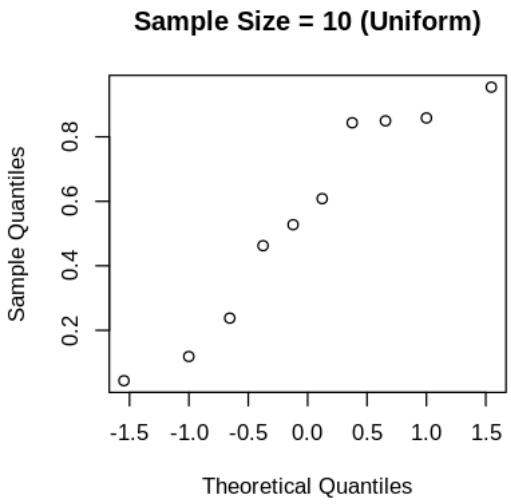
This code sets up a 3 by 4 array of plots using par(mfrow = c(3, 4)). Then, it generates and displays normal probability plots for samples of size 10, 100, and 1000 in the top, middle, and bottom rows respectively. Finally, it comments on how the appearance of the plots changes as the sample size increases.

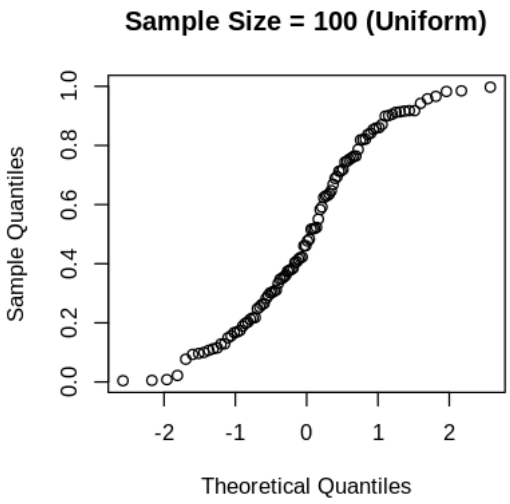
6. The function runif() generates a sample from a uniform distribution, by default on the interval 0 to 1. Try x <- runif(10), and print out the numbers you get. Then repeat exercise 6 above, but taking samples from a uniform distribution rather than from a normal distribution. What shape do the points follow?

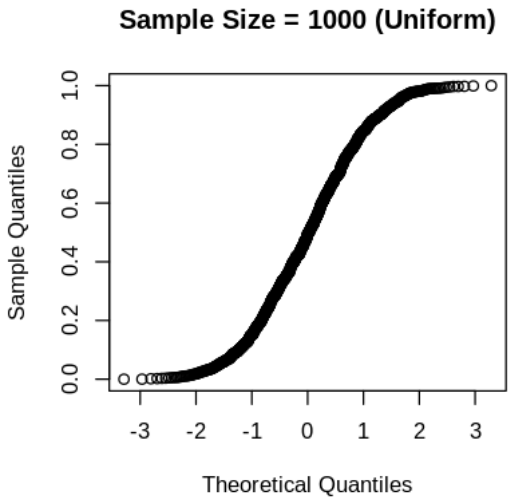
[92]:
```
# Generate a sample of size 10 from a uniform distribution on the interval [0,␣
↪1]
x <- runif(10)
print(x)
```

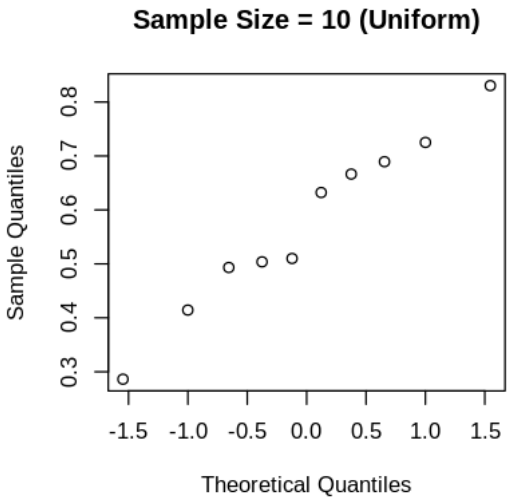
```r
# Set up the layout for a 3 by 4 array of plots
par(mfrow = c(2, 2))

# Generate and display normal probability plots for samples of size 10, 100,␣
  ↪and 1000 from a uniform distribution
for (i in 1:4) {
  # Sample size 10
  x_10 <- runif(10)
  qqnorm(x_10, main = "Sample Size = 10 (Uniform)")

  # Sample size 100
  x_100 <- runif(100)
  qqnorm(x_100, main = "Sample Size = 100 (Uniform)")

  # Sample size 1000
  x_1000 <- runif(1000)
  qqnorm(x_1000, main = "Sample Size = 1000 (Uniform)")
}
```

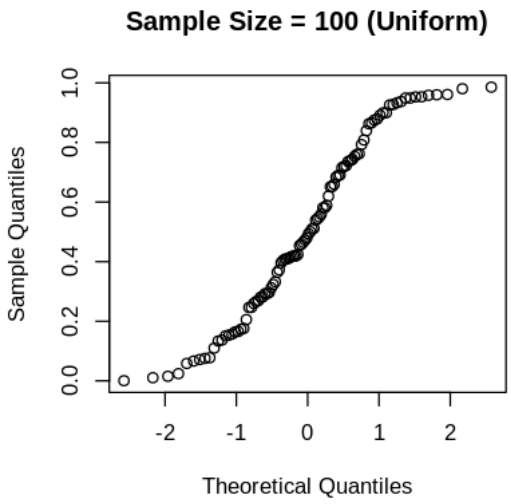
```
 [1] 0.50896998 0.52331279 0.38832991 0.61980015 0.29087819 0.88588327
 [7] 0.31262687 0.08033964 0.77196752 0.91321856
```

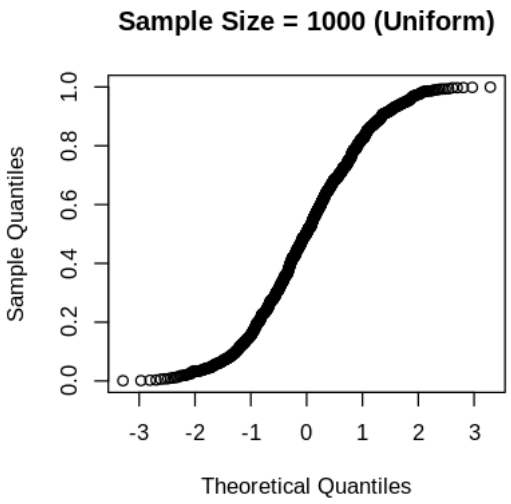

Sample Size = 10 (Uniform)
Sample Quantiles
Theoretical Quantiles
-1.5 -1.0 -0.5 0.0 0.5 1.0 1.5
0.2 0.4 0.6 0.8

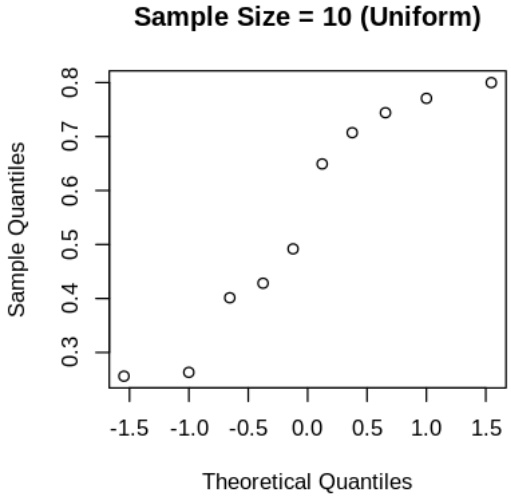

Sample Size = 100 (Uniform)
Sample Quantiles
Theoretical Quantiles
-2 -1 0 1 2
0.0 0.2 0.4 0.6 0.8 1.0

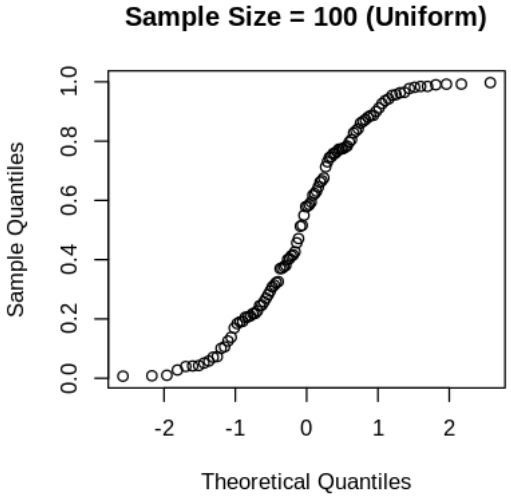

Sample Size = 1000 (Uniform)
Sample Quantiles
Theoretical Quantiles
-3 -2 -1 0 1 2 3
0.0 0.2 0.4 0.6 0.8 1.0

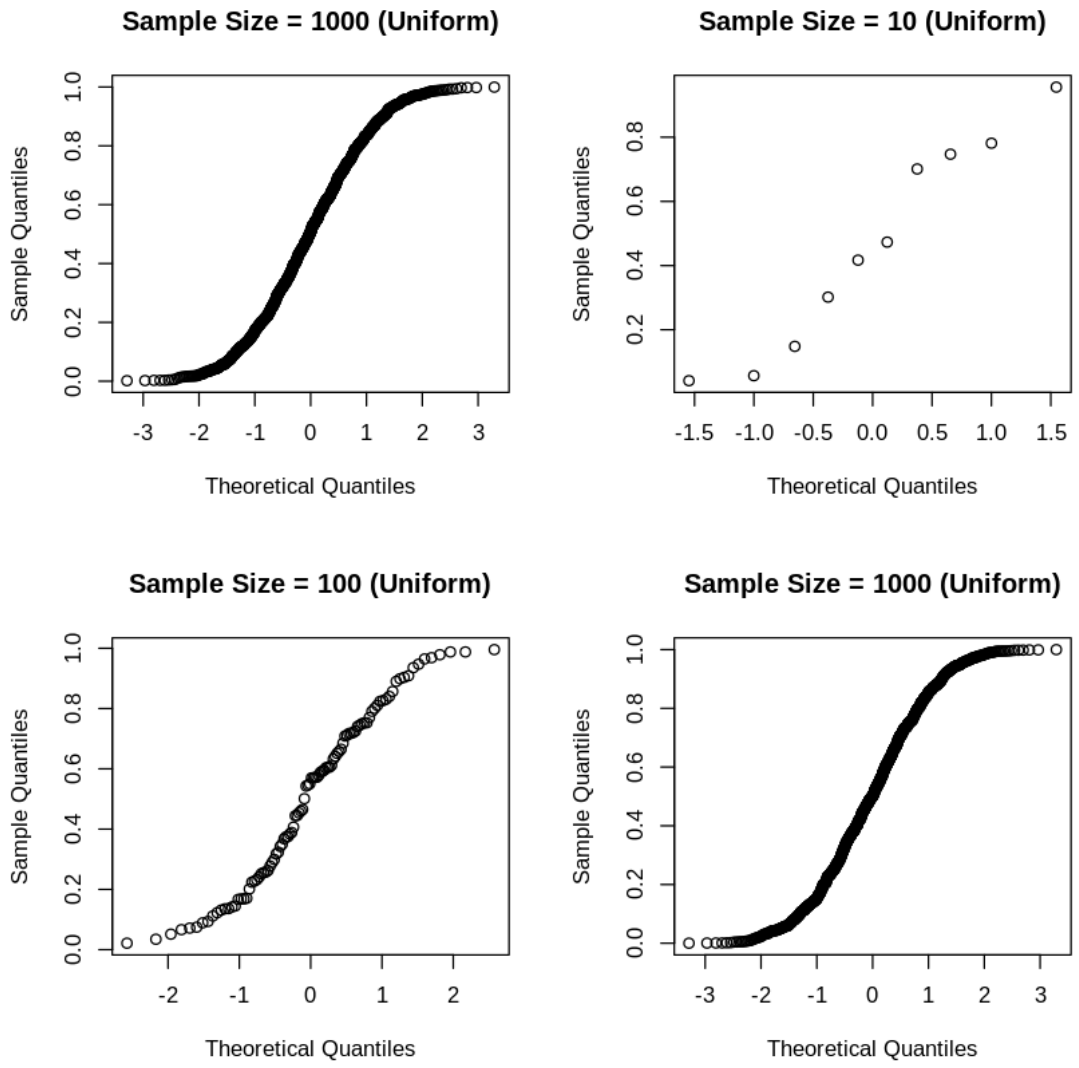

Sample Size = 10 (Uniform)
Sample Quantiles
Theoretical Quantiles
-1.5 -1.0 -0.5 0.0 0.5 1.0 1.5
0.3 0.4 0.5 0.6 0.7 0.8


Sample Size = 100 (Uniform)
Sample Quantiles
Theoretical Quantiles
0.0
0.2
0.4
0.6
0.8
1.0
-2
-1
0
1
2


Sample Size = 1000 (Uniform)
Sample Quantiles
Theoretical Quantiles
0.0
0.2
0.4
0.6
0.8
1.0
-3
-2
-1
0
1
2
3


Sample Size = 10 (Uniform)
Sample Quantiles
Theoretical Quantiles
0.3
0.4
0.5
0.6
0.7
0.8
-1.5
-1.0
-0.5
0.0
0.5
1.0
1.5


Sample Size = 100 (Uniform)
Sample Quantiles
Theoretical Quantiles
0.0
0.2
0.4
0.6
0.8
1.0
-2
-1
0
1
2



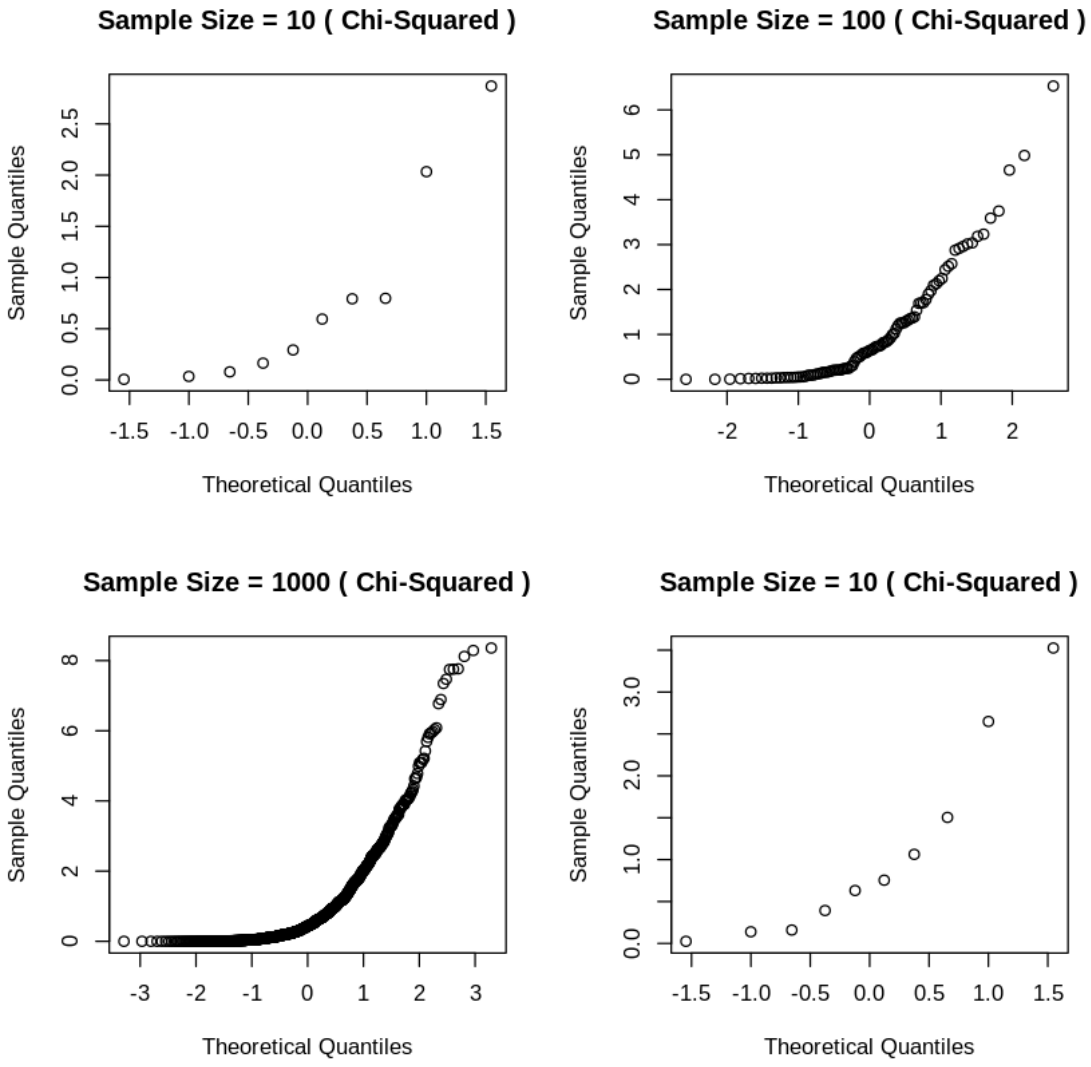
This code first generates a sample of size 10 from a uniform distribution using runif(10) and prints out the numbers. Then, it sets up a 3 by 4 array of plots. For each row, it generates and displays normal probability plots for samples of size 10, 100, and 1000 from a uniform distribution. The shape of the points in the plots follows a straight line, indicating a uniform distribution.

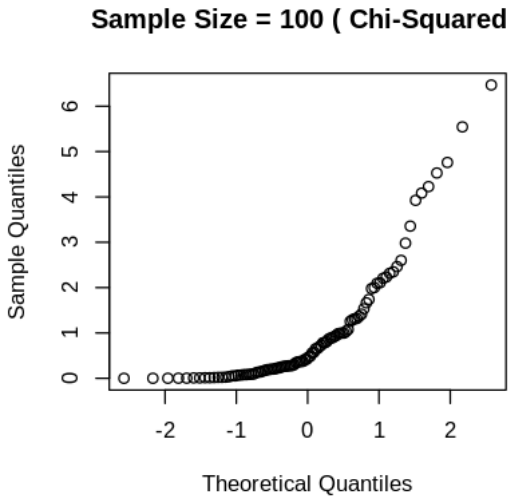
7. If you find exercise 6 interesting, you might like to try it for some further distributions. For example x <- rchisq(10,1) will generate 10 random values from a chi-squared distribution with degrees of freedom 1.

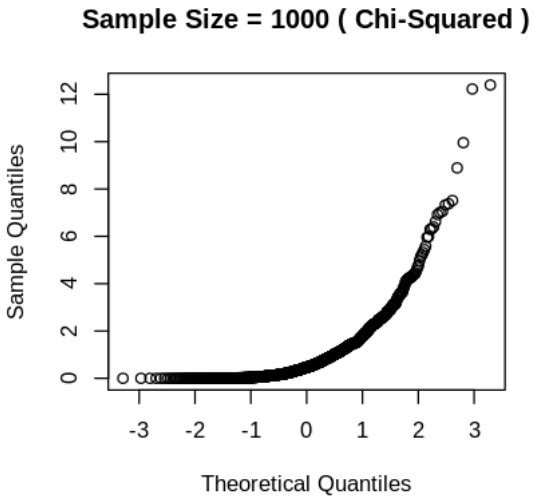
The statement x <- rt(10,1) will generate 10 random values from a t distribution with degrees of freedom one.

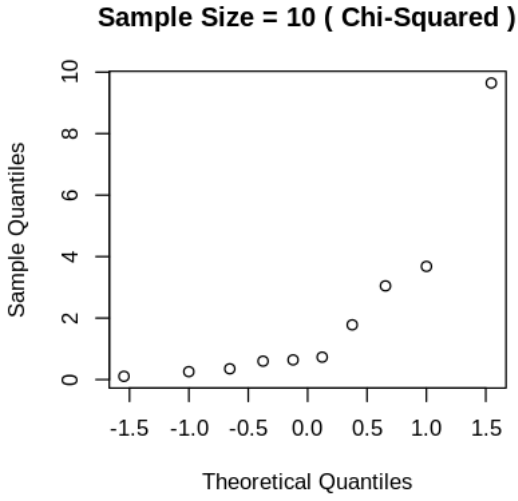
Make normal probability plots for samples of various sizes from these distributions.

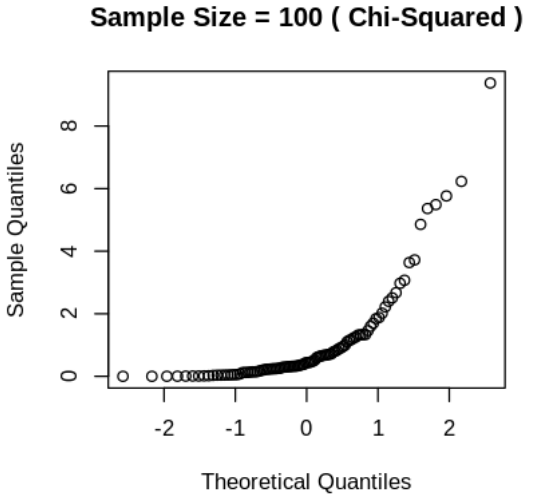
[97]:
```r
# Set up the layout for a 3 by 4 array of plots
par(mfrow = c(2, 2))

# Generate and display normal probability plots for samples of various sizes␣
 ↪from different distributions
distributions <- list("Chi-Squared" = rchisq, "T" = rt)

for (dist_name in names(distributions)) {
  for (i in 1:4) {
    for (sample_size in c(10, 100, 1000)) {
      x <- distributions[[dist_name]](sample_size, df = 1)
      qqnorm(x, main = paste("Sample Size =", sample_size, "(", dist_name, ")"))
    }
  }
}
```

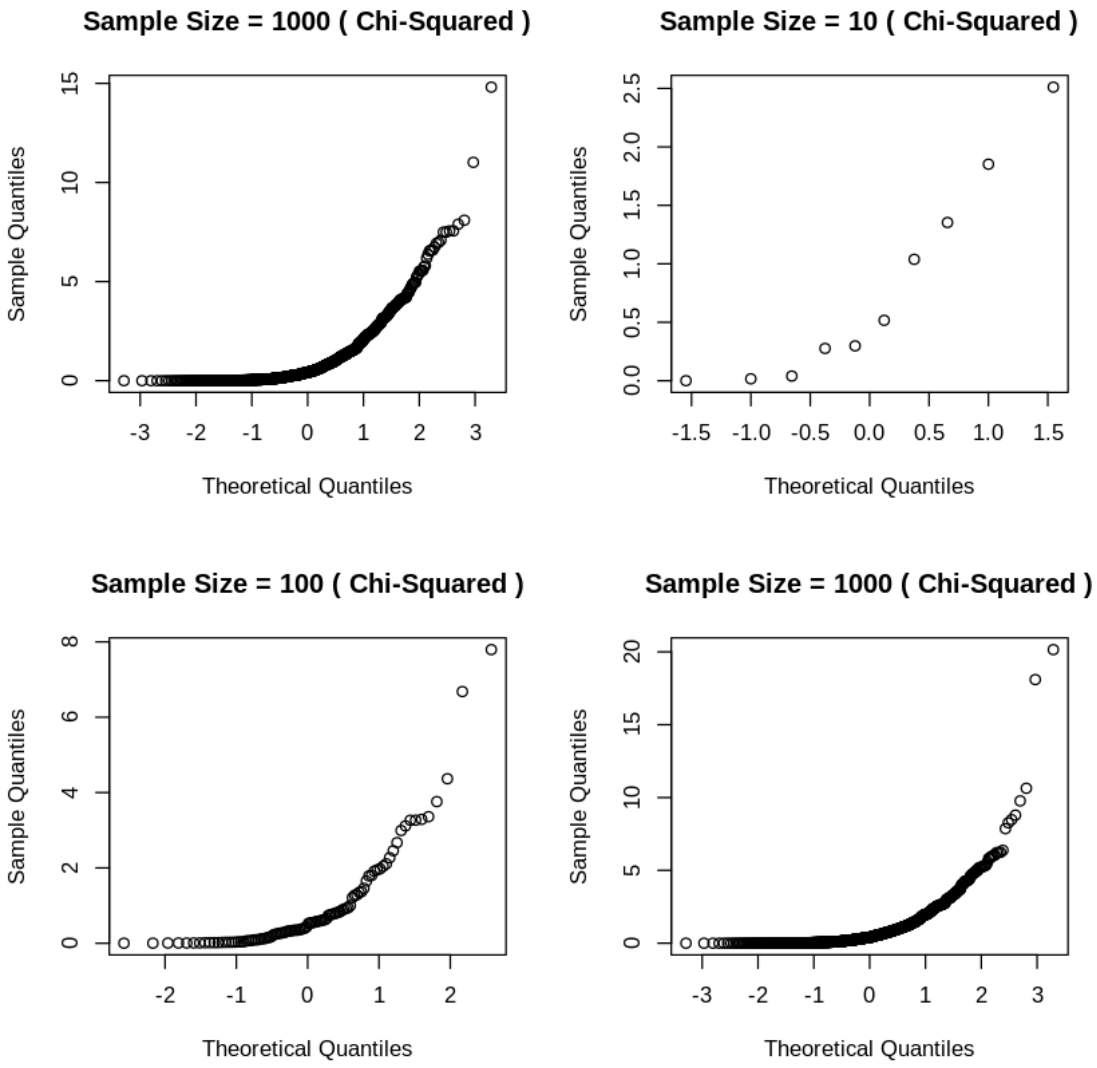


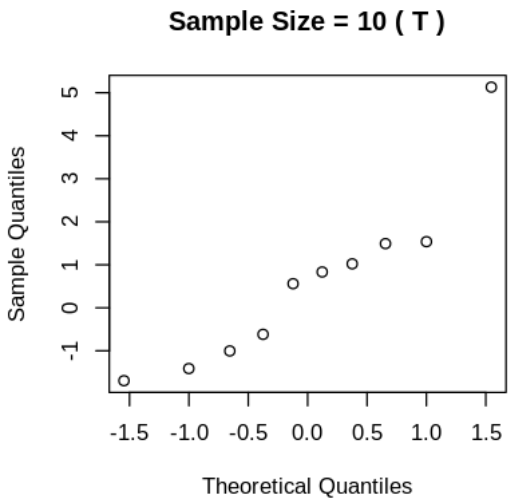

Sample Size = 100 (Chi-Squared)
Sample Quantiles
Theoretical Quantiles
0
1
2
3
4
5
6
-2
-1
0
1
2

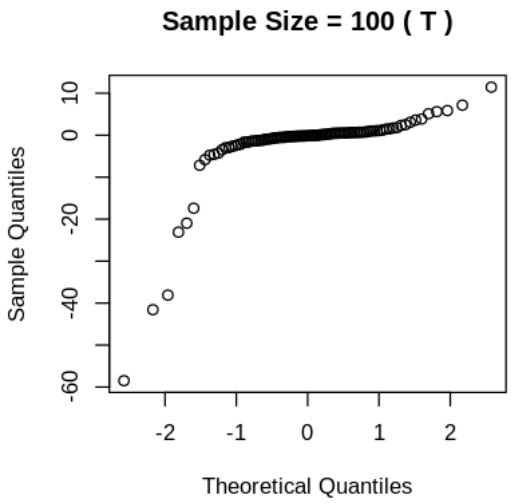

Sample Size = 1000 (Chi-Squared)
Sample Quantiles
Theoretical Quantiles
0
2
4
6
8
10
12
-3
-2
-1
0
1
2
3

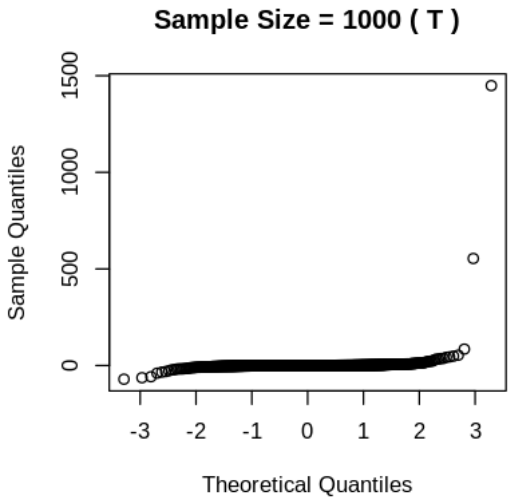

Sample Size = 10 (Chi-Squared)
Sample Quantiles
Theoretical Quantiles
0
2
4
6
8
10
-1.5
-1.0
-0.5
0.0
0.5
1.0
1.5

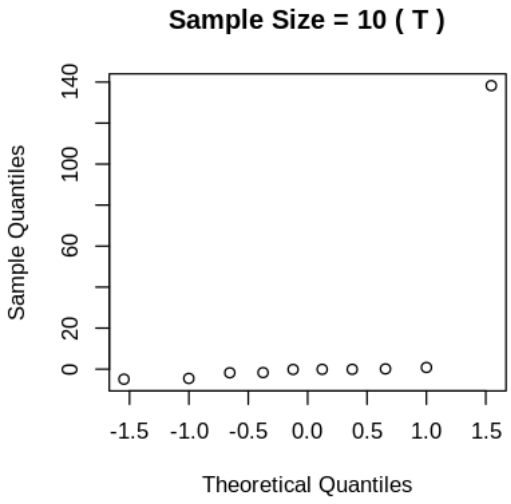

Sample Size = 100 (Chi-Squared)
Sample Quantiles
Theoretical Quantiles
0
2
4
6
8
-2
-1
0
1
2

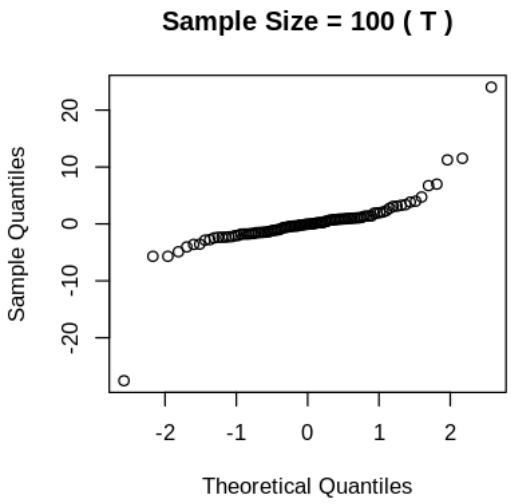

Sample Size = 1000 (Chi-Squared)
Sample Quantiles
Theoretical Quantiles
Sample Size = 10 (Chi-Squared)
Sample Quantiles
Theoretical Quantiles
Sample Size = 100 (Chi-Squared)
Sample Quantiles
Theoretical Quantiles
Sample Size = 1000 (Chi-Squared)
Sample Quantiles
Theoretical Quantiles

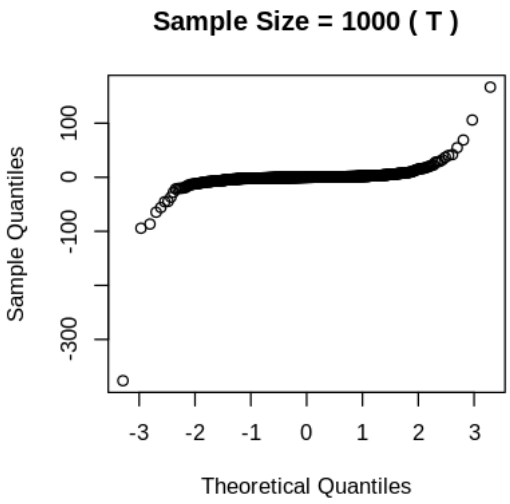

Sample Size = 10 (T)
Sample Quantiles
Theoretical Quantiles
-1
0
1
2
3
4
5
-1.5
-1.0
-0.5
0.0
0.5
1.0
1.5

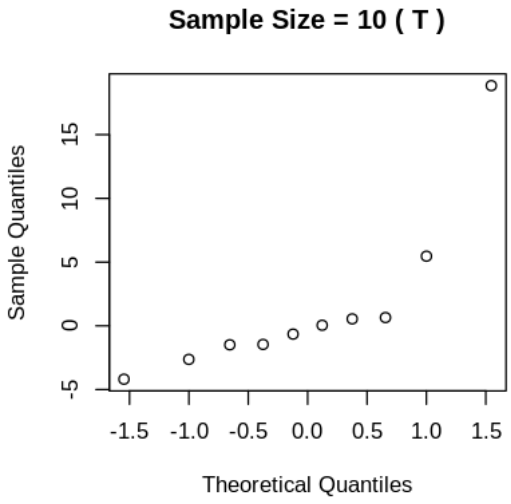

Sample Size = 100 (T)
Sample Quantiles
Theoretical Quantiles
-60
-40
-20
0
10
-2
-1
0
1
2

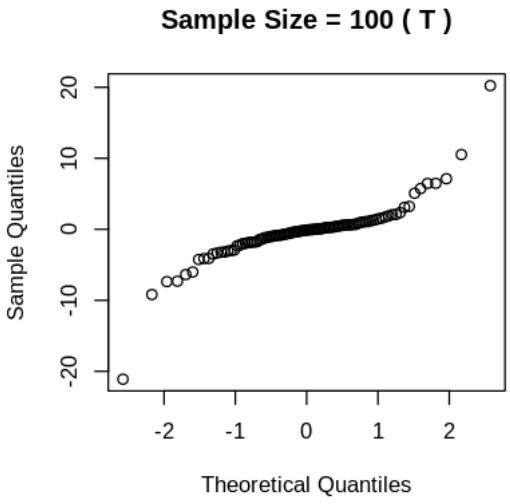

Sample Size = 1000 (T)
Sample Quantiles
Theoretical Quantiles
0
500
1000
1500
-3
-2
-1
0
1
2
3

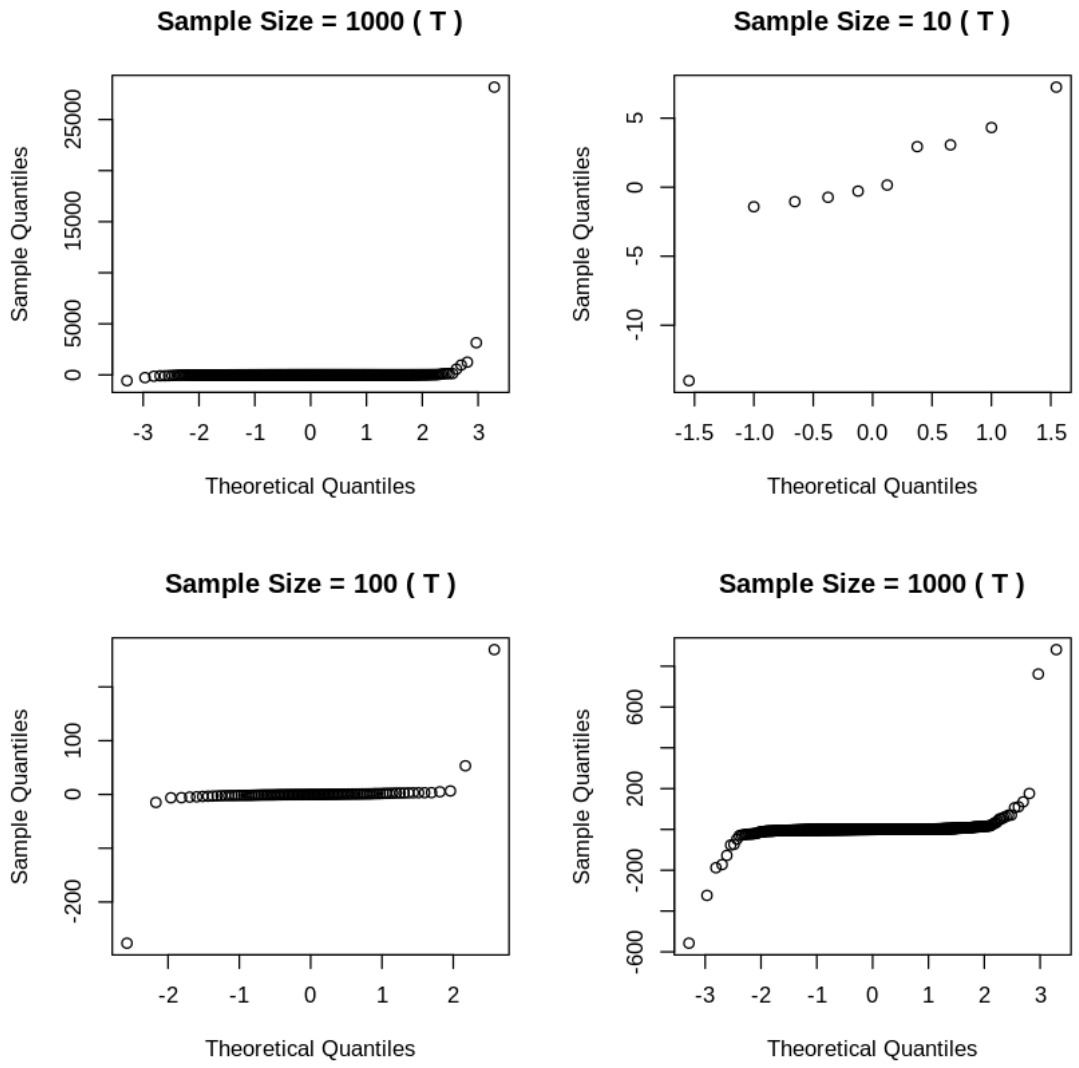

Sample Size = 10 (T)
Sample Quantiles
Theoretical Quantiles
0
20
60
100
140
-1.5
-1.0
-0.5
0.0
0.5
1.0
1.5


Sample Size = 100 (T)
Sample Quantiles
Theoretical Quantiles
-20
-10
0
10
20
-2
-1
0
1
2


Sample Size = 1000 (T)
Sample Quantiles
Theoretical Quantiles
-300
-100
0
100
-3
-2
-1
0
1
2
3


Sample Size = 10 (T)
Sample Quantiles
Theoretical Quantiles
-5
0
5
10
15
-1.5
-1.0
-0.5
0.0
0.5
1.0
1.5


Sample Size = 100 (T)
Sample Quantiles
Theoretical Quantiles
-20
-10
0
10
20
-2
-1
0
1
2

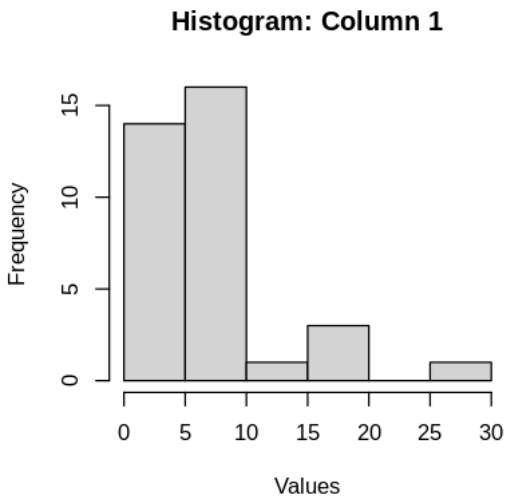


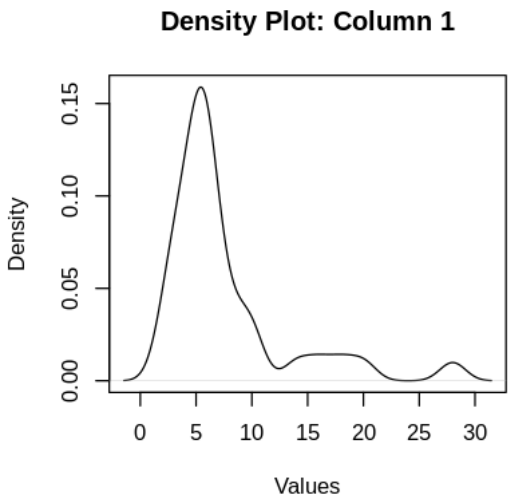
This code sets up a 3 by 4 array of plots. For each distribution (chi-squared and t), it generates and displays normal probability plots for samples of sizes 10, 100, and 1000. The code loops through each distribution and sample size combination to create the plots.

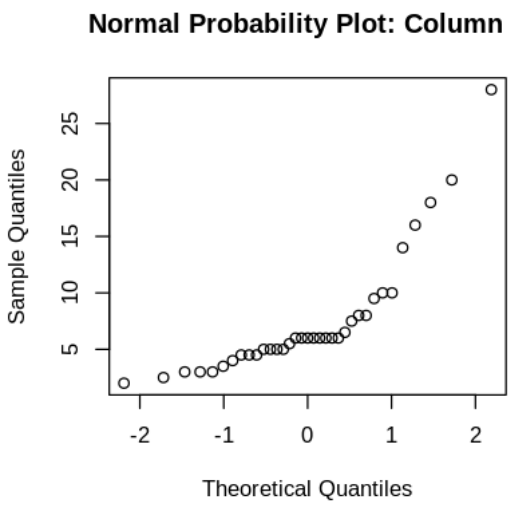
8. For the first two columns of the data frame hills, examine the distribution using: (a) histogram; (b) density plots; (c) normal probability plots. Repeat (a), (b) and (c), now working with the logarithms of the data values.

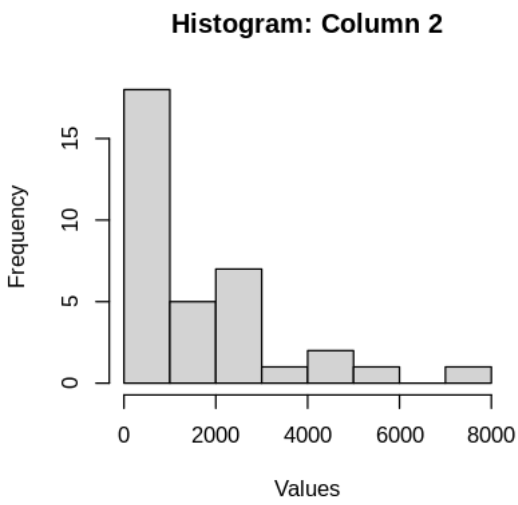
[99]:
```
# Load the hills dataset
data(hills)

# Subset the first two columns of the hills dataset
hills_subset <- hills[, 1:2]
```

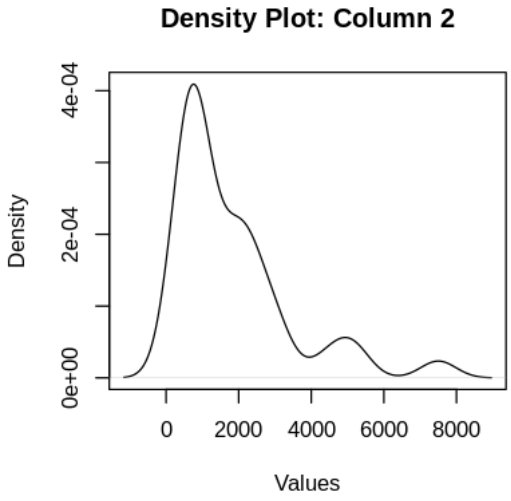
```r
# Set up the layout for a 2 by 2 array of plots
par(mfrow = c(2, 2))

# Histogram for the first column
hist(hills_subset[, 1], main = "Histogram: Column 1", xlab = "Values")

# Density plot for the first column
plot(density(hills_subset[, 1]), main = "Density Plot: Column 1", xlab =␣
 ↪"Values", ylab = "Density")

# Normal probability plot for the first column
qqnorm(hills_subset[, 1], main = "Normal Probability Plot: Column 1")

# Histogram for the second column
hist(hills_subset[, 2], main = "Histogram: Column 2", xlab = "Values")

# Density plot for the second column
plot(density(hills_subset[, 2]), main = "Density Plot: Column 2", xlab =␣
 ↪"Values", ylab = "Density")

# Normal probability plot for the second column
qqnorm(hills_subset[, 2], main = "Normal Probability Plot: Column 2")

# Histogram for the logarithms of the first column
hist(log(hills_subset[, 1]), main = "Histogram: Log(Column 1)", xlab =␣
 ↪"Log(Values)")

# Density plot for the logarithms of the first column
plot(density(log(hills_subset[, 1])), main = "Density Plot: Log(Column 1)",␣
 ↪xlab = "Log(Values)", ylab = "Density")

# Normal probability plot for the logarithms of the first column
qqnorm(log(hills_subset[, 1]), main = "Normal Probability Plot: Log(Column 1)")

# Histogram for the logarithms of the second column
hist(log(hills_subset[, 2]), main = "Histogram: Log(Column 2)", xlab =␣
 ↪"Log(Values)")

# Density plot for the logarithms of the second column
plot(density(log(hills_subset[, 2])), main = "Density Plot: Log(Column 2)",␣
 ↪xlab = "Log(Values)", ylab = "Density")

# Normal probability plot for the logarithms of the second column
qqnorm(log(hills_subset[, 2]), main = "Normal Probability Plot: Log(Column 2)")
```

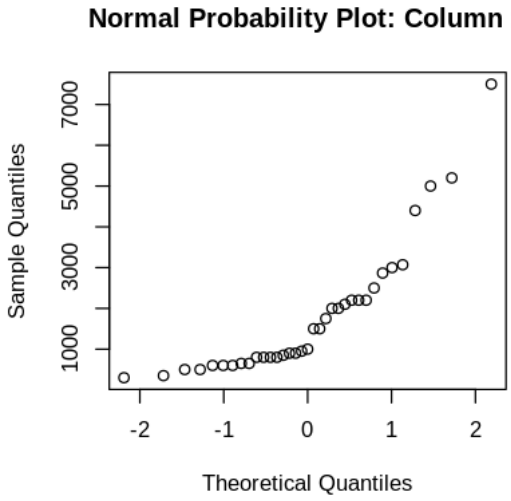

Histogram: Column 1
Frequency
0
5
10
15
0
5
10
15
20
25
30
Values

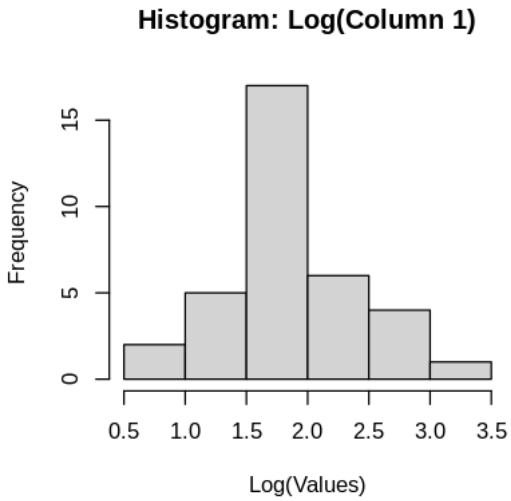

Density Plot: Column 1
Density
0.00
0.05
0.10
0.15
0
5
10
15
20
25
30
Values

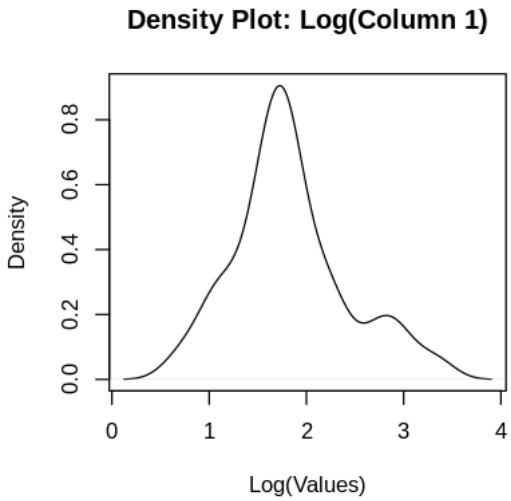

Normal Probability Plot: Column 1
Sample Quantiles
5
10
15
20
25
-2
-1
0
1
2
Theoretical Quantiles

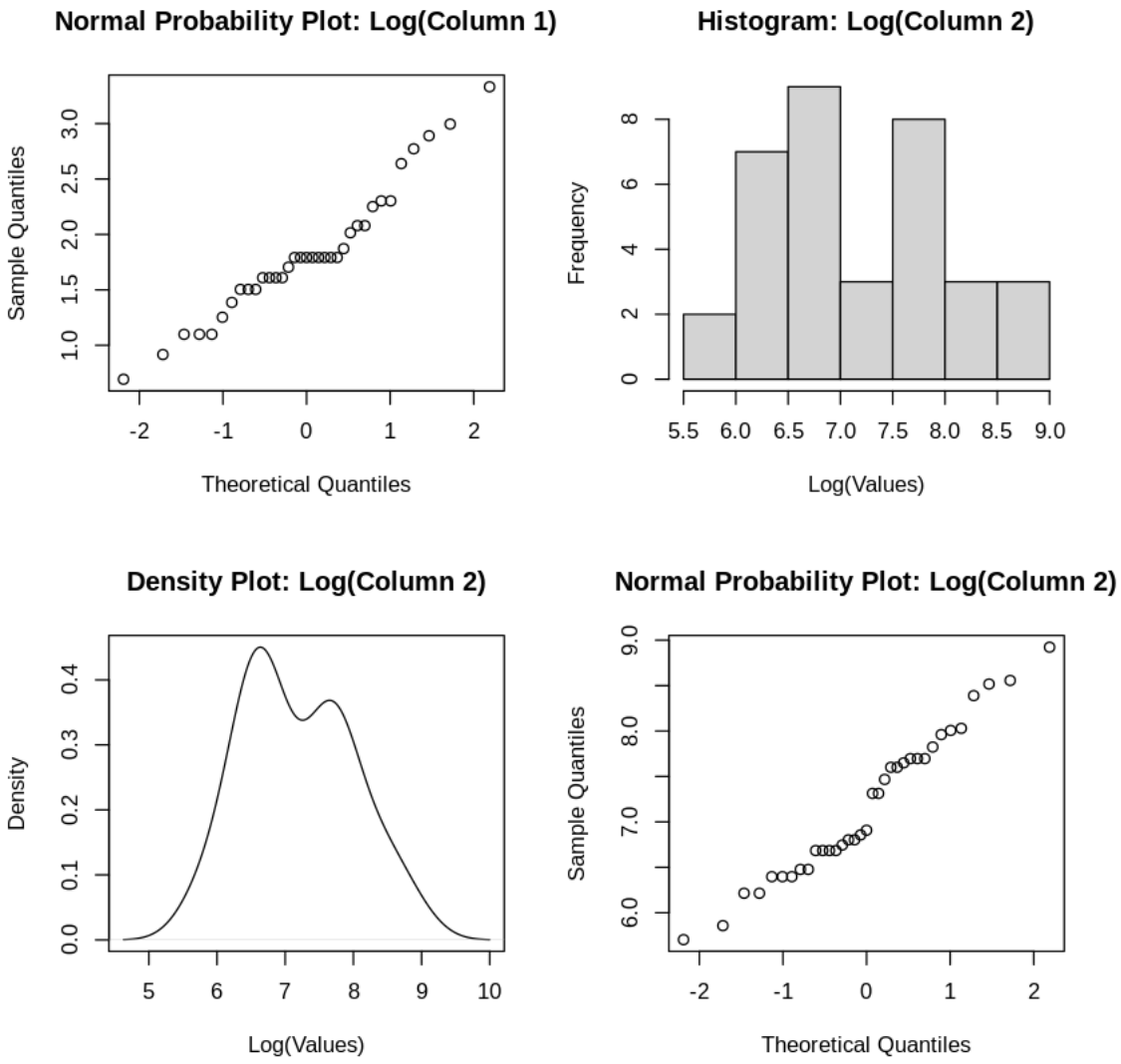

Histogram: Column 2
Frequency
0
5
10
15
0
2000
4000
6000
8000
Values


Density Plot: Column 2
Density
0e+00
2e-04
4e-04
0
2000
4000
6000
8000
Values


Normal Probability Plot: Column 2
Sample Quantiles
1000
3000
5000
7000
-2
-1
0
1
2
Theoretical Quantiles


Histogram: Log(Column 1)
Frequency
0
5
10
15
0.5
1.0
1.5
2.0
2.5
3.0
3.5
Log(Values)


Density Plot: Log(Column 1)
Density
0.0
0.2
0.4
0.6
0.8
0
1
2
3
4
Log(Values)



1.10 3.10 References

The web page http://www.math.yorku.ca/SCS/StatResource.html#DataVis has links to many different collections of information on statistical graphics