Data Visualization in R

Instructor: Mary Yang, PhD

Graduate Assistant: Dan Li and Yifan Zhang

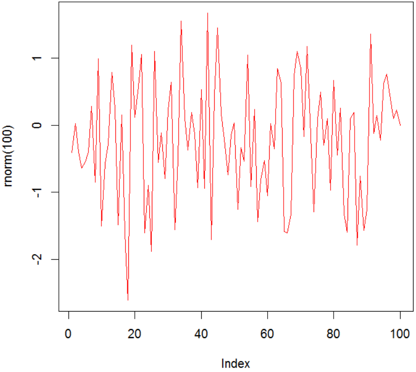
August 2, 2017

Graphics with R

- A great strength of R is visualization
- There are many functions in R that produce graphs, and they range from the very basic to the very advanced
- After a figure is created, you can
  - print directly from the graphics window
  - or copy the graph to the clipboard and paste it into a word processor
  - or save the a graph in many other formats, including pdf, bitmap, metafile, jpeg, or postscript

plot()

Hundred random numbers are plotted by connecting the points by lines in a red color




```
plot (rnorm(100), type="l", col="red")
```

plot()

An example, consider the dataset *Orange* in R

```
> data (Orange)
> Orange
    Tree  age circumference
1      1  118            30
2      1  484            58
3      1  664            87
4      1 1004           115
5      1 1231           120
6      1 1372           142
7      1 1582           145
8      2  118            33
9      2  484            69
10     2  664           111
```

plot()

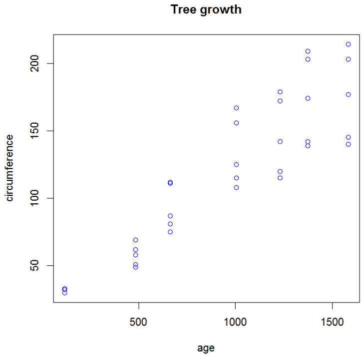
- To visualize the relationship between age and circumference, you can draw a scatter plot

```
> plot (Orange$age, Orange$circumference, col="blue")
```

- Notice that the format here is the first variable is plotted along the horizontal axis and the second variable is plotted along the vertical axis.
- By default, the variable names are listed along each axis
- You can add titles/subtitles, changing the plotting character/color (over 600 colors are available!), etc.
- See **?par** for lists of these options

plot()

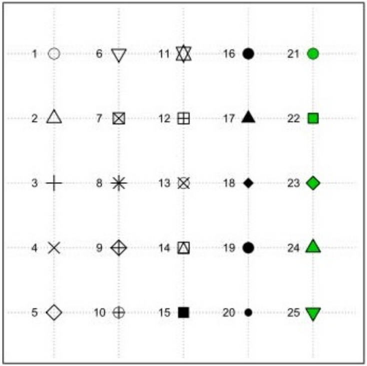
You can add titles/subtitles, changing the plotting character/color (over 600 colors are available!), etc. See **?par** for lists of these options




```
> attach (Orange) # Attach the R object to search path
> plot (age, circumference, col="blue", main = "Tree growth")
```

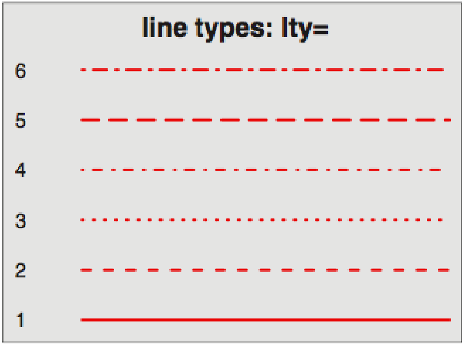
Plot Symbol

- You can specify the **pch** parameter to get different plot symbols
  - a number (pch=1 gives a circle)
  - a text character (pch="v" uses the letter "v")
  - For symbols 21 through 25, specify border color (col=) and fill color (bg=).



Line

lty: Specify line types



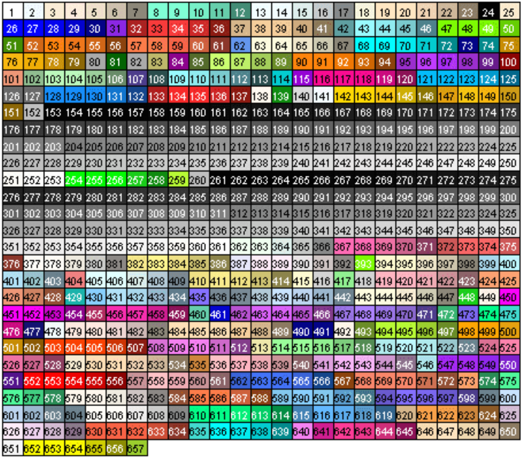
lwd:specify line width relative to the default (default=1). 2 is twice as wide as the default

Text and Symbol Size

Option	Description
cex	number indicating the amount by which plotting text and symbols should be scaled relative to the default. 1: default, 1.5 is 50% larger, 0.5 is 50% smaller, etc.
cex.axis	magnification of axis annotation relative to cex
cex.lab	magnification of x and y labels relative to cex
cex.main	magnification of titles relative to cex
cex.sub	magnification of subtitles relative to cex

Colors

- Specify colors in R by index, name, hexadecimal, or RGB. For example col=1, col="white", and col="#FFFFFF" are equivalent.



- Create a vector of n contiguous colors using the functions **rainbow(n)**, **heat.colors(n)**, **terrain.colors(n)**, **topo.colors(n)**, and **cm.colors(n)**.

Colors

Option	Description
col	Default plotting color. Some functions (e.g. lines) accept a vector of values that are recycled.
col.axis	color for axis annotation
col.lab	color for x and y labels
col.main	color for titles
col.sub	color for subtitles

Useful Functions

Function	Operation
`abline()`	adds a straight line with specified intercept and slope (or draw a vertical or horizontal line)
`arrows()`	adds an arrow at a specified coordinate
`lines()`	adds lines between coordinates
`points()`	adds points at specified coordinates (also for overlaying scatterplots)
`segments()`	similar to `lines()` above
`text()`	adds text (possibly inside the plotting region)
`title()`	adds main titles, subtitles, etc. with other options

Changing Graphics Parameters

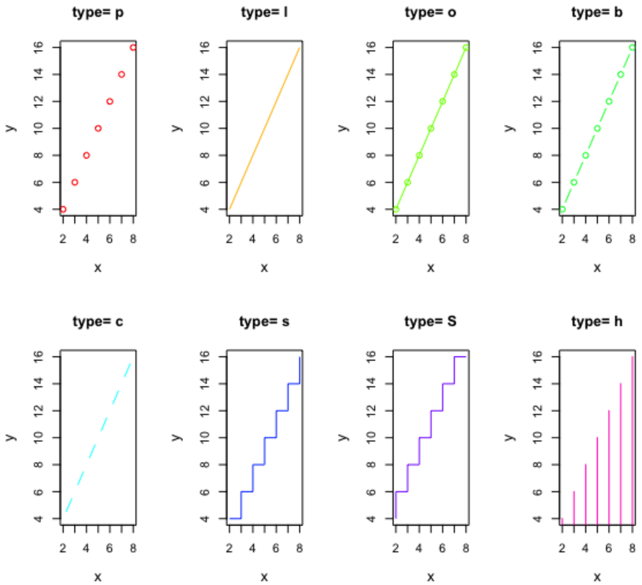
- The default graphical parameters can be changed using the **par()** function
- There are over 70 graphics parameters that can be adjusted
- Some very useful ones are given below:

```
# view current settings
> par()

# make a copy of current settings
> opar = par()

# gives a 2 x 2 layout of plots
> par(mfrow = c(2, 2))

# plots drawn with this colored background
>par(bg = "cornsilk")

# restore original settings
> par (opar)
```

Example: plot()

```
x = 2:8
y = 2 * x
# Layout of sub-figures
par(bg = "cornsilk")
par(mfrow=c(2,4))
opts = c("p","l","o","b","c","s","S","h")
cols = rainbow(8)
for(i in 1:length(opts)){
  title = paste("type=",opts[i])
  plot(x, y, type="n", main=title, col = cols[i])
  lines(x, y, type=opts[i], col = cols[i])
}
```

Example



type= p
type= l
type= o
type= b
type= c
type= s
type= S
type= h
y
x
4 6 8 10 12 14 16
2 4 6 8

Add legend to a figure

legend() function

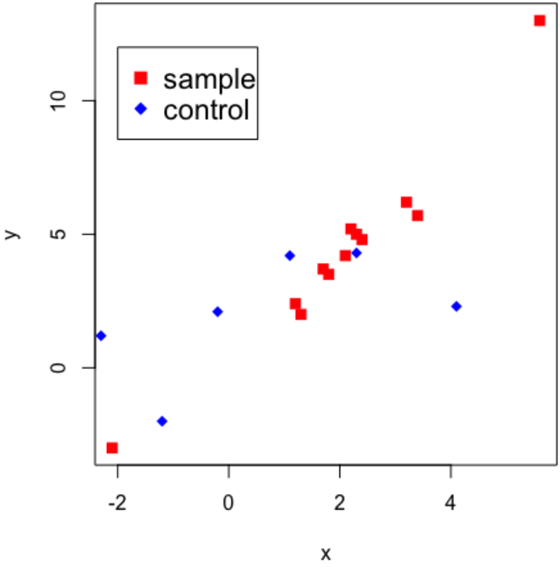
- When more than one set of data or group is incorporated into a graph, a legend can help you to identify whats being represented by each bar, pie slice, or line, etc.

Option	Description
location	There are several ways to indicate the location of the legend. You can give an x,y coordinate for the upper left hand corner of the legend. You can also use the keywords "bottom", "bottomleft", "left", "topleft", "top", "topright", "right", "bottomright", or "center"
fill	Fill the legend box with color
legend	A character vector with the labels
col	Color of the legend content
border	Border color (when legend box is filled)
lty,lwd	Line types and widths of the legend
pch	The plotting symbols appearing in the legend

Example: legend()

```
x = c(1.2,3.4,1.3,-2.1,5.6,2.3,3.2,2.4,2.1,1.8,1.7,2.2)
y = c(2.4,5.7,2.0,-3,13,5,6.2,4.8,4.2,3.5,3.7,5.2)
plot(x,y,cex=1.2,pch=15,xlab="x",ylab="y",col="red")

#Use add() add more another data set to the plot
x2 <- c(4.1,1.1,-2.3,-0.2,-1.2,2.3)
y2 <- c(2.3,4.2,1.2,2.1,-2,4.3)
points(x2,y2,cex=1.2,pch=18,col="blue")

legend(x=-2,y=12,c("sample","control"),cex=1.4, col=c("red","blue
    "),pch=c(15,18))
```

Example: legend()



hist()

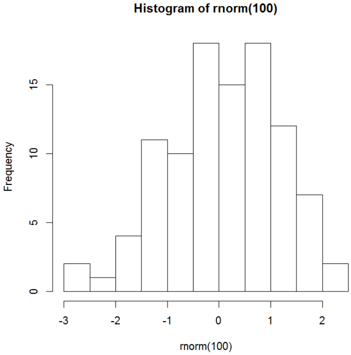
- You can generate a histogram plot to visualize distribute of the data **hist(x, breaks, freq, col, main, xlim, ylim, xlab , ylab ...)**

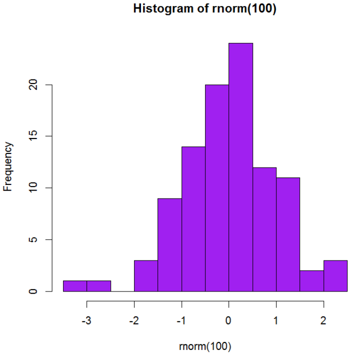
  - **x** : a vector of values for which the histogram is desired.
  - **breaks** one of:
    - a **vector** giving the breakpoints between histogram cells,
    - a **function** to compute the vector of breakpoints,
    - a **single number** giving the number of cells for the histogram
  - **freq**: if TRUE, the histogram graphic is a representation of frequencies. If FALSE, probability densities are plotted.
  - **col**: a colour to be used to fill the bars. The default of NULL yields unfilled bars.
  - **main**: title of the plot
  - **x/ylim**: limits of the x axis or y axis
  - **x/ylab**: a label for the x axis or y axis

- **hist()**

```
# histogram of random variable
> hist (rnorm(100))
> hist (rnorm (100), breaks = 20, col="purple")
```

hist()

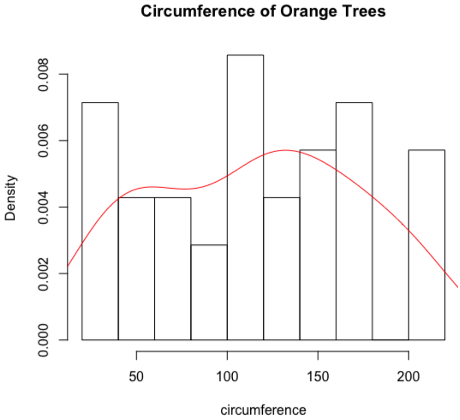





```
> hist (rnorm(100))
```

```
> hist (rnorm(100), breaks =
20, col="purple")
```

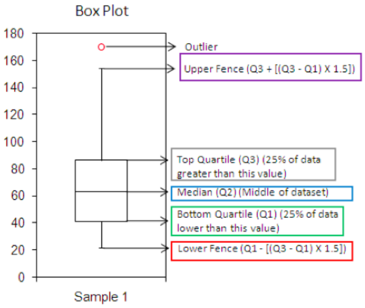
Add density plot




```
> hist (circumference, freq = FALSE, main = "Circumference of
   Orange Trees")
> lines (density (circumference), col = "red")
```

boxplot()

- Box plot shows 5 statistically significant numbers
  - the minimum
  - the 25th percentile
  - the median
  - the 75th percentile
  - the maximum
- It is useful for visualizing the spread of the data is and deriving inferences accordingly



boxplot()

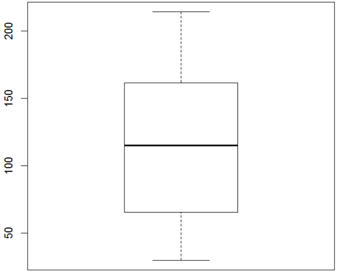
- **boxplot()**
  - The function **boxplot ()** will construct a single boxplot if the argument passed is a single vector
  - If many vectors are contained (or if a data frame is passed), a boxplot for each variable is produced on the same graph.

```
> boxplot (rnorm(100))

> boxplot (circumference)

> boxplot (Orange)
```

- See **?boxplot** for ways to add titles/color, changing the orientation, etc.
- Exercise: Make a box plot of **Orange** data. You need to add colors for boxes in the figure, and add a title for your figure. Save your figure.

boxplot()

```
> boxplot (circumference)
```




```
> boxplot (Orange)
```

barplot()

The generic function to creates a bar plot with vertical or horizontal bars.
barplot(height, col, main , xlim, ylim, xlab , ylab ...)

- **height** : either a vector or matrix of values describing the bars which make up the plot.
  - If height is a **vector**, the plot consists of a sequence of rectangular bars with heights given by the values in the vector.
  - If height is a **matrix** then each bar of the plot corresponds to a column of height, with the values in the column giving the heights of stacked sub-bars making up the bar.
- **col** : a colour to be used to fill the bars. The default of NULL yields unfilled bars.
- **main**: title of the plot
- **x/ylim** : limits of the x axis or y axis
- **x/ylab** : a label for the x axis or y axis

barplot()

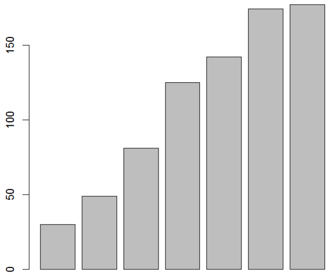
- **barplot()**
  For discrete or categorical data, you can display the information using the **barplot()** function.

```
# Bar plot of circumference of tree 5
# Create tree5 in your directory:
> tree5 = Orange[Tree == 5,]
> barplot (tree5$circumference)

> barplot (tree5$circumference, names.arg = tree5$age, las =
    1, xlab = "age", ylab = "circumference")
```

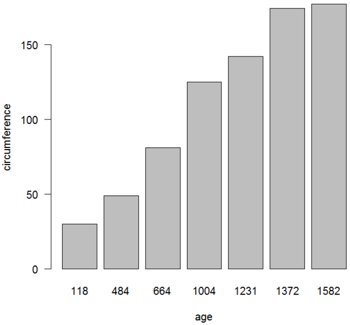
- Exercise: Add a title to the figure, and save the figure

barplot()

```
> barplot
(tree5$circumference)
```




```
> barplot
(tree5$circumference,
names.arg = tree5$age, las =
1, xlab = 'age', ylab =
'circumference')
```

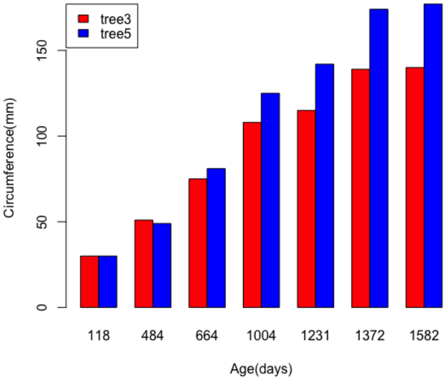


barplot()

You can compare the circumference difference between trees. For example,

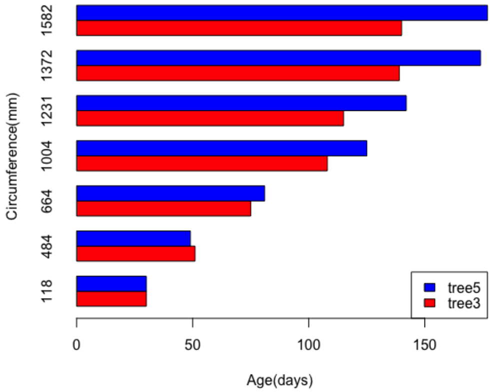
```
> tree3 = Orange [Tree == 3,]

> tree3_5 = rbind (tree3$circumference, tree5$circumference)

> barplot(tree3_5,beside = T, names.arg = tree5$age, legend.text
    = c("tree3", "tree5"), col= c("red", "blue"), args.legend =
    list(x="topleft"), xlab = "Age(days)", ylab = "Circumference(
    mm)")
```

barplot()




```
> barplot(tree3_5,beside = T, names.arg = tree5$age, legend.text = c("tree3
    ", "tree5"), col= c("red", "blue"), args.legend = list(x="topleft"), xlab =
    "Age(days)", ylab = "Circumference(mm)")
```

barplot()




```
> barplot(tree3_5,beside = T, names.arg = tree5$age, legend.text = c("tree3", "tree5"), col= c("red", "blue"), args.legend = list(x="bottomright"), xlab = "Age(days)", ylab = "Circumference(mm)", horiz = T)
```

attach(), detach()

- **attach()**: makes the data available to the R search path, it is possible to refer to the variables in the data frame by their names alone, rather than as components of the data frame - *age* rather than *Orange$age*

```
> attach (Orange)
```

- Caution: if there is already a variable called *age* in the local workspace, issuing *attach*(*Orange*), may not mean that *age* references *Orange$age*.
  - Name conflicts of this type are a common problem with *attach*()
- **detach()**: reverse the process

```
> detach (Orange)
```

scatter plot

- A scatterplot is a useful way to visualize the relationship between two variables.
- Similar to correlations, scatterplots are often used to make initial diagnoses before any statistical analyses are conducted.
- The basic function in R for drawing scatter plot is **plot ()**, directly graph two variables using the default settings

plot(x, y,main,sub,pch, ...)

- **x**: the x coordinates of points in the plot
- **y**: the y coordinates of points in the plot

plot(df, main,sub,pch, ...)

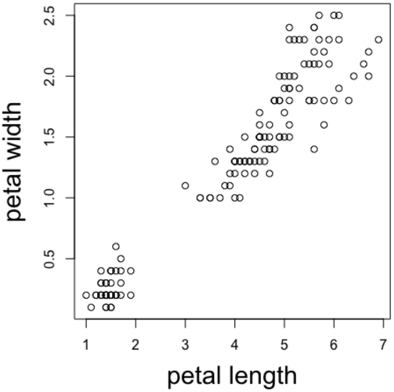
- **bf**: a data frame for plot and the function will use each pair of column as x and y coordinates to generate multiple plots

- **main**: the title for the plot (displayed at the top)
- **sub**: the subtitle for the plot (displayed at the bottom)
- **pch** : type of symbol of each point in the plot
- **col** : color of the point

scatter plot

```
> data ("iris")
> ?iris
> class (iris)
[1] "data.frame"
> dim (iris)
[1] 150   5
> colnames (iris)
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
[5] "Species"
> iris
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1            5.1         3.5          1.4         0.2     setosa
2            4.9         3.0          1.4         0.2     setosa
3            4.7         3.2          1.3         0.2     setosa
4            4.6         3.1          1.5         0.2     setosa
5            5.0         3.6          1.4         0.2     setosa
.
.
.
```

scatter plot

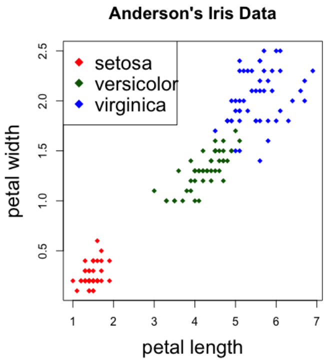



```
> plot(x = iris$Petal.Length, y = iris$Petal.Width , xlab = "
   petal length",ylab = "petal width")
```

scatter plot: iris data

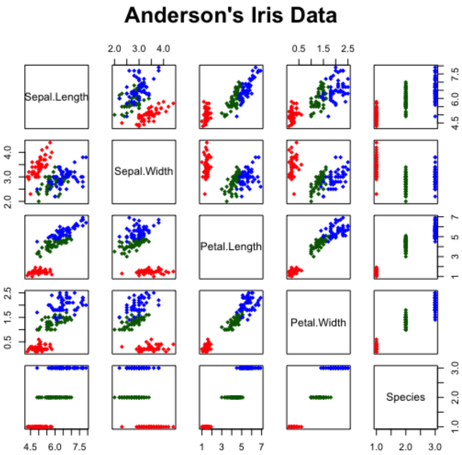
```
> class (iris$Species)
[1] "factor"
> table (iris$Species)

   setosa versicolor virginica
       50 50 50
> levels (iris$Species)
[1] "setosa" "versicolor" "virginica"
> iris.species = levels (iris$Species)
> class (iris.species)
[1] "character"
> iris.color = c("red","darkgreen","blue")[unclass(iris$Species)]
```

scatter plot: iris data




```
> plot(x = iris$Petal.Length, y = iris$Petal.Width , xlab = "
    petal length",ylab = "petal width", col = iris.color, pch =
    18, main="Anderson’s Iris Data")
> legend ("topleft", legend = iris.species, col = c("red","
    darkgreen","blue"), pch = 18)
```

scatter plot: data frame




```
> plot (iris, col = iris.color, pch = 18, main="Anderson's Iris
    Data")
```

Pie chart

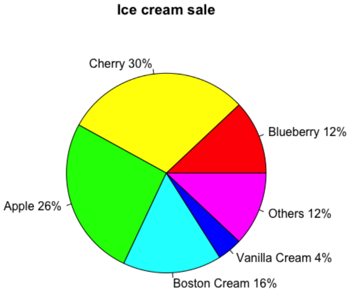
- A pie chart is a circular statistical graphic which is divided into slices to illustrate numerical proportion.
- Pie charts are created with the function **pie(x, labels, ...)**
  - **x**: is a non-negative numeric vector indicating the area of each slice
  - **labels**: a character vector of names for the slices
- Recommend bar or dot plots over pie charts as people can judge length more accurately than volume.

pie()

pie (x, labels, radius, ...)

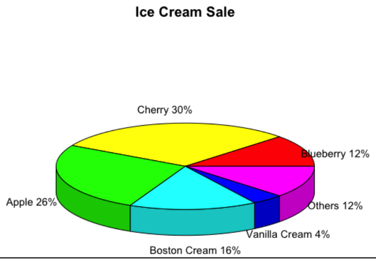
- **x**: a vector of non-negative numerical quantities. The values in x are displayed as the areas of pie slices.
- **labels**: one or more expressions or character strings giving names for the slices.
- **radius**: the pie is drawn centered in a square box whose sides range from -1 to 1. If the character strings labeling the slices are long it may be necessary to use a smaller radius.

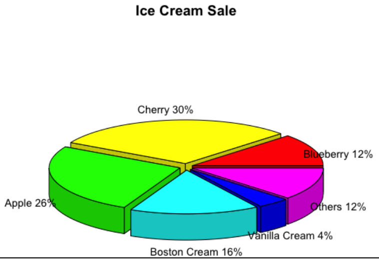
pie()




```
> slices = c(0.12, 0.30, 0.26, 0.16, 0.04, 0.12)
> types= c("Blueberry", "Cherry", "Apple", "Boston Cream", "Vanilla Cream", "
    Others")
> pct = slices * 100
> lbs = paste (types, pct)
> lbs = paste (lbs, "%", sep = "")
> pie(slices,labels = lbs, col=rainbow(length(lbs)), main="Ice Cream Sale")
```

pie3D() in the plotrix package generates 3D exploded pie charts.






```
> install.packages ("plotrix") #install the package
> library (plotrix) #load the package to your workplace

#generate left pie chart
> pie3D(slices,labels=lbs,main="Ice Cream Sale", labelcex = 0.9)

#generate right pie chart
> pie3D(slices,labels=lbs,pie3D(slices,labels=lbs,explode=0.05,
main="Ice Cream Sale", labelcex = 0.9)
```

Advanced graphics

- Advanced graphics
  - grid
  - lattice
  - ggplot2
- Expand on the capabilities of, and correct for deficiencies in, base graphics system

Advanced graphics

- The **grid** graphics system provides low-level access to graphic primitives, giving programmers a great deal of flexibility in the creation of graphic output.
- The **lattice** package provides an intuitive approach for examining multivariate relationships through conditional 1, 2, or 3dimensional graphs called trellis graphs.
- The **ggplot2** package provides a method of creating innovative graphs based on a comprehensive graphical grammar.

ggplots2 package

- The simplest approach for creating graphs in **ggplot2** is through the **qplot()** (for quick plot).
- The **qplot()** function can be used to create the most common graph types though it does not expose full power of ggplot

```
qplot(x, y, data=, color=, shape=, size=, alpha=, geom=,
method=, formula=, facets=, xlim=, ylim=, xlab=, ylab=, main=,
sub=)
```

ggplot2: qplot()

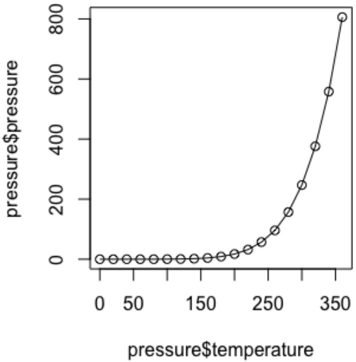
Option	Description
x, y	Specifies the variables placed on the horizontal and vertical axis. For univariate plots (for example, histograms), omit y
xlab, ylab	Character vectors specifying horizontal and vertical axis labels
xlim,ylim	Two-element numeric vectors giving the minimum and maximum values for the horizontal and vertical axes, respectively
data	Specifies a data frame
alpha	Alpha transparency for overlapping elements expressed as a fraction between 0 (complete transparency) and 1 (complete opacity)
facets	describe how data is split into subsets and displayed as multiple small graphs
geom	Specifies the geometric objects that define the graph type. The geom option is expressed as a character vector with one or more entries. geom values include "point", "smooth", "boxplot", "line", "histogram", "density", "bar", and "jitter".
main, sub	Character vectors specifying the title and subtitle

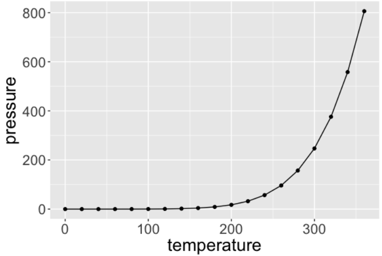
ggplot: pressure data

```
> data (pressure)
> ?pressure
> class (pressure)
[1] "data.frame"
> colnames (pressure)
[1] "temperature" "pressure"
> dim (pressure)
[1] 19 2
```

```
pressure: data on the relation between temperature in degrees
Celsius and vapor pressure of mercury in millimeters (of mercury).

A data frame with 19 observations on 2 variables.
[, 1]  temperature  numeric  temperature (deg C)
[, 2]  pressure  numeric  pressure (mm)
```

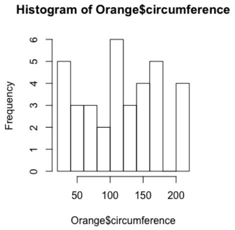
Line Graphs

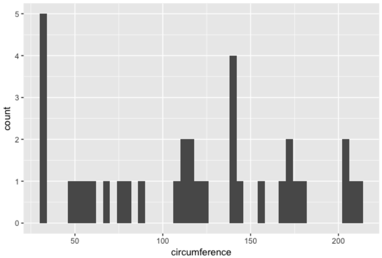





```
# Generate the left figure
> plot(pressure$temperature, pressure$pressure, type="l")
> points(pressure$temperature, pressure$pressure)

#Generate the right figure
ggplot(pressure, aes(x=temperature, y=pressure)) + geom_line() + geom_point() + theme(
    text = element_text(size=20))
```

Histogram





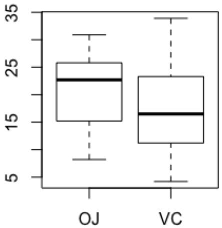

```
> hist (Orange$circumference)

> ggplot(Orange, aes(x=circumference)) + geom_histogram(binwidth
    =4)
```

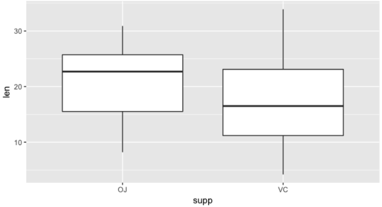
Boxplot: ToothGrowth data

```
> ?ToothGrowth
> dim (ToothGrowth)
[1] 60 3
> class (ToothGrowth)
[1] "data.frame"
> colnames (ToothGrowth)
[1] "len" "supp" "dose"
```

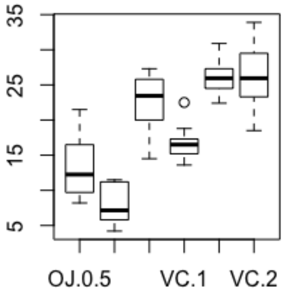
```
ToothGrowth, a data frame with 60 observations on 3 variables.
[,1]  len   numeric  Tooth length
[,2]  supp  factor   Supplement type (VC or OJ).
[,3]  dose  numeric  Dose in milligrams/day
```

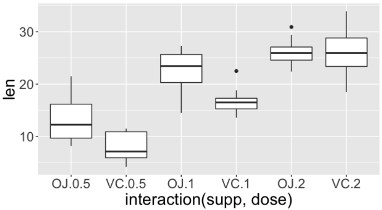
Boxplot: ToothGrowth data






```
# Left figure
> plot(ToothGrowth$supp, ToothGrowth$len)
# Formula syntax
> boxplot(len ~ supp, data = ToothGrowth)

# Right figure
> ggplot(ToothGrowth, aes(x=supp, y=len)) + geom_boxplot()
```






```
# Left figure
#Put interaction of two variables on x−axis
>boxplot(len ~ supp + dose, data = ToothGrowth)

# Right figure
> ggplot(ToothGrowth, aes(x=interaction(supp, dose), y=len)) + geom_
    boxplot() + theme(text = element_text(size=20))
```

Reference

- R Graphics Cookbook by *Winston Wang*
- R in action by *Robert Kabacoff*
- Using R for Data Analysis and Graphics by *J H Maindonald*
- Online resource: Quick R

- What is the standard error (SE) of a mean?
  - The SE measures the amount of variability in the sample mean.
  - It indicated how closely the population mean is likely to be estimated by the sample mean.
- SE is different from Standard Deviation (SD) which measures the amount of variability in the population.
- SE incorporates SD to assess the difference between sample and population measurements due to sampling variation

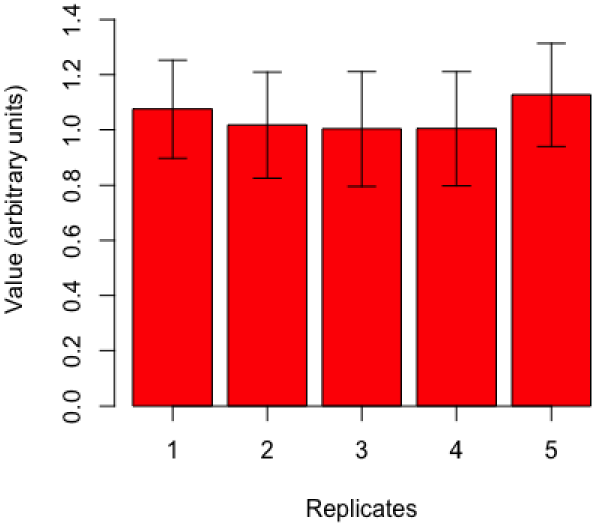
Calculation of SE for mean = $SD/sqrt(n)$

- The sample mean and its SE provide a range of likely values for the true population mean.
- How can you calculate the Confidence Interval (CI) for a mean?
- Assuming a normal distribution, we can state that 95% of the sample mean would lie within 1.96 SEs above or below the population mean, since 1.96 is the 2-sides 5% point of the standard normal distribution.
- Calculation of CI for mean = (mean + (1.96 x SE)) to (mean (1.96 x SE))

Error bar

```
error.bar <- function(x, y, upper, lower=upper, length=0.1,...){
  if(length(x) != length(y) | length(y) !=length(lower) | length(lower) !=
      length(upper))
    stop("vectors must be same length")
  arrows(x,y+upper, x, y-lower, angle=90, code=3, length=length, ...)
}

y <- rnorm(500, mean=1)
y <- matrix(y,100,5)
y.means <- apply(y,2,mean)
y.sd <- apply(y,2,sd)
barx <- barplot(y.means, names.arg=1:5,ylim=c(0,1.5), col="red", axis.lty
     =1, xlab="Replicates", ylab="Value (arbitrary units)")
error.bar(barx,y.means, 1.96*y.sd/10)
```

Error bar



Value (arbitrary units)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1
2
3
4
5
Replicates

Error bar

```
y <- rnorm(500, mean=1)
y <- matrix(y, 100, 5)
y.means <- apply(y, 2, mean)
y.sd <- apply(y, 2, sd)
lower = 1.96*y.sd/10
upper = lower

df <- data.frame(col = c("col1", "col2", "col3", "col4", "col5"), mean = y.means)
ggplot(df, aes(x = col, y = mean)) + geom_col(col = "red", fill = "red") + geom_errorbar(
      aes(ymin=y.means - lower, ymax=y.means + upper), width=0.2)
```

Error bar