1. R as a Calculator (for Scalars)

Command	Meaning	Example
Arithmetic:		
`x [+-*/^] y`	$x+y, x-y, xy, x/y, x^y$	`7 / 3`, `8^(1/3)`
`x %/% y`	integer division	`7 %/% 3`
`x %% y`	modulo (remainder)	`7 %% 3`
Calculator functions:		
`exp()`	exponential	`exp(1)`
`log(x, base = exp(1))`	logarithm	`log(9, base = 3)`
(“=” indicates default)		`e = exp(1); log(e^2)`
`cos(), sin(), tan()`	trigonometry	`sin(pi/2)`
`sqrt()`	square root	`sqrt(9)`
Other easy functions:		
`abs(x)`	absolute value	`abs(-3)`
`floor(x)`	greatest int ≤ x	`floor(-1.5)`
`ceiling(x)`	smallest int ≥ x	`ceiling(-1.5)`
`round(x, digits = 0)`	round to #decimal places	`round(4/3, 2)`
`signif(x, digits = 6)`	round to #significant	`signif(4/3, 2)`
Statistics distributions:		
`dnorm(x, mean = 0, sd = 1)`	$f(x)$	`dnorm(0)          # density`
`pnorm(q, mean = 0, sd = 1)`	$P(X \leq q)$ for $X \sim N$(mean, sd)	`pnorm(-1, 0, 1)  # probability`
`qnorm(p, mean = 0, sd = 1)`	x with $P(X \leq x) = p$	`qnorm(.16, 0, 1) # quantile`
`rnorm(n, mean = 0, sd = 1)`	random from $N(0,1)$	`rnorm(1, 7, .01) # random`
`[dpqr][t,chisq,f,binom]()`	other distributions	`?pt`, `pt(-2, 100)`
Miscellaneous:		
`?name`	`help("name")`	`?pt` (help includes Description, Usage, Arguments, Value, Examples)
`??topic`	`help.search("topic")`	`??deviation`
`<-` (or `=`)	assign variable	`x <- 3` (or `x = 3`)
`variable.name`	`print(variable.name)`	`x`
`ls()`	list variables	
`rm(list = ls())`	clear all variables	
`list.files()`	list all files	
`#`	comment rest of line	`N <- 3 # number of points`
`quit()`	quit R	
`demo(topic)`	run demo code	`demo("graphics"), demo("plotmath")`
`source(file)`	read code from `file`	`source("quiz1.R")`
`setwd(dir)`	set working directory	`setwd("C:/Users/jg/Desktop/327")`
Shortcuts		Help > Keyboard Shortcuts
...		
↑,↓ (up-, down-arrow)	previous command, next	
`Esc`	interrupt current command	
...		

2. Vector

A *vector* (or one-dimensional *array*) `v` is a collection of *values* (or *elements*) of the same type, each identified by an *index* in the range `1` to `length(v)`. *Combine* values into a vector with `c(...)`. e.g.

```
v <- c(2.71, 5, 3.14)
length(v)
v
```

index `i`	value `v[i]`
1	2.71
	5
3	

```
words <- c("tree", "ant", "chainsaw")
length(words)
words
```

index `i`	value `words[i]`
1	
2	`"ant"`
	`"chainsaw"`

Basic Vector Types, and Specifying Constants of These Types

- `numeric` (real number): digits with optional decimal point, with optional suffix of `E` or `e` for *exponent* digits (scientific notation); e.g. `3.14e2` is _______
- `character` (which should have been called *character string*): a *string* (or word) in double or single quotes, `"..."` or `'...'`. (*Escape sequences* include `\"` (double quote), `\'` (single quote), `\n` (newline), `\t` (tab), and `\\` (backslash).)

  `paste(..., sep = " ")` makes a string from its arguments, separated by `sep`. e.g.

  ```
  oak <- 70
  text = paste(sep="", "Tree names include \"oak.\"\nOak weighs ", oak, " lbs/ft^3.\n")
  ```

  `cat(..., sep = " ")` pastes and writes to console, interpreting escape sequences. e.g.

  ```
  cat(text)
  cat(sep = "", "oak=", oak, "\n") # display variable with helpful label
  ```

- `logical`: `TRUE` and `FALSE` (which become `1` and `0` when used in arithmetic)

  `any(v)` is `TRUE` if any of the values in `v` is `TRUE`; `all(v)` is `TRUE` if all are

  e.g. `v > 3`, `words == "ant"`, `sum(v > 3)`, `sum(words == "ant")`

`vector(mode="logical", length=0)` creates a vector of the given `mode` and `length`.

To change a vector's type, use `as.numeric()`, `as.character()`, or `as.logical()`. (There are three other basic types we will not use much: `integer`, `complex`, and `raw`.)

Names attribute

`names(x)` gets or sets a vector of `character` (strings) corresponding to values in x. e.g.

```
names(v) = c("e", "five", "pi"); v  # set names
names(v) = NULL; v                  # remove names
```

Names can also be set with `c()` by specifying "name=value" pairs. e.g. `y = c(burger=2.50, fries=1.50); y`

A Few Functions

e.g. `x <- c(12, 11, 16, 11)`

`sum(x)`, `max(x)`, `mean(x)`, `median(x)`, `sd(x)`

Operators (which act element-wise on vectors)

- arithmetic: `+ - * / ^` (and, for integer division, `%/%` is quotient, `%%` is remainder)

  e.g. The sample standard deviation of $x_1, x_2, \cdots, x_n$ is $s_x = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$:

  ```
  n = length(x)
  sqrt(sum((x - mean(x))^2) / (n-1))
  ```

- relation: `> >= < <= == !=` (last two are *equals* and *is not equal to*)

- logic:

! ("not")	T	F
	F	T

& ("and")	T	F
T	T	F
F	F	F

| ("or")	T	F
T	T	T
F	T	F

  e.g. `!(v > 3)`, `v < 4`, `(v > 3) & (v < 4)`, `(v > 3) | (v < 4)`

- assignment: `<-` (or `=`, which is not `==`)

- sequence: `:` (colon); e.g. `11:14` is `c(   ,   ,   ,   )`

  `seq()` is a related function:

  - `seq(from=1, to=1, by)`, e.g. `seq(10, 15, by=2)` is `c(   ,   ,   )`
  - `seq(from=1, to=1, length.out)`, e.g. `seq(10, 15, length.out=3)` is `c(   ,   ,   )`

- matching: `%in%`, e.g. `1:3 %in% c(2, 7)` is `c(   ,   ,   )`

Indexing

- For a vector `v` of positive `integer`, `x[v]` is those elements of `x` with indices in `v`; e.g. for `x <- 11:20` and `v <- c(1, 2, 10)`, `x[v]` is `c(   ,   ,   )`; `x[3]` is `c(   )` (or ________)

- For a vector `v` of negative `integer`, `x[v]` is those elements of `x` *excluding* those with indices in `v`; e.g. for `x <- 11:20` and `v <- c(-1, -2, -10)`, `x[v]` is ______________

- For a vector `v` of `logical`,

  - `which(v)` is a vector of *indices* for which `v[i]` is `TRUE`; e.g.

    `indices = which(x < 14)  # c(   ,   ,   )`

    Now use the indices: `x[indices]` is `c(   ,   ,   )`

  - `x[v]` is those elements of `x` corresponding to `TRUE` values in `v`. e.g.
    `x[x < 14]` is `c(   ,   ,   )`
    (so "`which`" could have been omitted in previous example)

  e.g. `x[(x %% 2) == 0]`

- For a vector `v` of `character` names, `x[v]` is those elements of `x` whose names are in `v`; e.g.
  `x <- 1:3; names(x) <- c("one", "two", "Fred"); v <- c("Fred", "one"); x[v]` is __________

3. Vector (continued) and List

Vector (continued)

Sorting functions

`sort(x, decreasing = FALSE)` returns a sorted copy of `x`. e.g. `x = c(12, 11, 16, 11); sort(x)`

`v = rank(x, ties.method = "average")`: `v[i]` is rank of `x[i]`; also try `ties.method = "first"`

`v = order(x, ..., decreasing = FALSE)`: `v[i]` is index of *i*th smallest value in `x`, so `x[v]` is sorted. (`order()` sorts data frames, coming soon. More vectors may be given in `...` to break ties.)

Structure, summary, quantile

`str(object)` displays the structure of `object`; e.g. `str(x)`

`summary(object)` summarizes it; e.g. `s = summary(x)`

`v = quantile(x, probs = seq(0, 1, 0.25))`: `v[i]` is the quantile corresponding to `probs[i]`

NULL and NA special values

`NULL` is the *null object* returned by expressions and functions whose value is undefined; e.g. `names(x)`

`NA` ("not available") indicates a missing data value. `NA` propagates in calculations. e.g.

```
x[3] = NA; sum(x); sum(x, na.rm=TRUE) # "na.rm=FALSE" is a common function argument
```

Test for these values with `is.null()` and `is.na()`, not with "`== NULL`" or "`== NA`"

(Vector) File input/output

`scan(file = "", what = numeric())` reads a vector of `numeric` from `file`, which defaults to `stdin` (the console). `what` options include `logical()`, `numeric()`, and `character()`.

`write(x, file = "data")` writes vector `x` to `file` (which defaults to `"data"`). e.g.

```
fifties = 50:59; write(x=fifties, file="50s.txt"); y = scan(file="50s.txt", what=integer())
```

List

A list is an ordered collection of components not necessarily of the same type. e.g.

```
x = list("rope", 72, c(5, 7, 10), TRUE); length(x)
```

List components can be named via name=value pairs in `list()` or via `names()`.

```
y = list(self = "John",
         spouse = "Michele",
         kids.ages = c(0, 2, 5, 7, 9, 11)
        )
```

List indexing with single brackets returns a sub-list

(Recall: vector indexing with `[...]` returns a sub-vector.) The index can be a vector of `integer` (select), negative `integer` (exclude), `logical` (select `TRUE`), or `character` names (select). e.g.

```
str(y[2]); str(y[2:3]); str(y[c(-2, -3)]); str(y[c("spouse", "kids.ages")])
```

Indexing with double brackets or $ returns one component, dropping names

e.g. `x[4]` vs. `x[[4]]`

e.g. `str(y[[2]]); str(y[["spouse"]])`

The `$` operator used in the form `list.name$component.name` is the same as `list.name[[component.name]]`, except that `$` doesn't allow a computed index.

e.g. `y$spouse; y$kids.ages; y$kids.ages[3]`

e.g. `z = "spouse"; y[[z]]; y$z`

Add a component to a list by assigning it:

```
y$kids.names = c("Teresa", "Margaret", "Monica", "Andrew", "Mary", "Philip")
```

Remove a component from a list by setting it to `NULL`: `y$self = NULL`

e.g. Sort names alphabetically; then sort ages to keep up:

```
indices.ordered.by.name = order(y$kids.names); indices.ordered.by.name
(names.sorted = y$kids.names[indices.ordered.by.name]) # (...) calls print(...)
(ages.sorted.by.name = y$kids.ages[indices.ordered.by.name])
```

Convert a list to a vector

`unlist(x, use.names=TRUE)` simplifies a list `x` to a vector (where possible). e.g. `unlist(y)`

A *data frame*, coming soon, is (≈) a list of equal-length vectors (like a spreadsheet).

4. Data Frame, Factor, Formula

Data Frame (R's fundamental data structure)

A `data.frame` is (≈) a list of equal-length vectors.

e.g. `mtcars` is a built-in `data.frame`: `mtcars`, `?mtcars`, `str(mtcars)`, `summary(mtcars)`.

Factor

A *factor* represents a vector of categorical values. Categorical data must be converted to factors for R's summary, plotting, and modeling functions to work correctly.

`factor(x, levels, labels = levels)` makes a factor from vector `x`, using levels in `levels` (or the unique strings in `as.character(x)` by default), using optional `labels` to make the categories more readable. e.g.

```
m = mtcars          # work on a copy
m$vs = factor(mtcars$vs, labels=c("V", "straight"))
m$am = factor(mtcars$am, levels=c("0", "1", "2"), labels=c("auto", "man", "CVT"))
# suppose level "2" and label "CVT" are useful even though they are not in mtcars
str(m)
summary(m) # now categorical variables are handled well
```

`table(...)` makes a contingency table of counts of each combination of factors in `...`. e.g. `table(m$vs)`, `table(m$vs, m$am)`

Data frame manipulation examples

```
m$mpg               # mpg column
m[ , 1]             # all rows, 1st column (mpg again)
m[1:3, 1:3]         # rows 1:3, columns 1:3
dim(m)              # dimensions
n.rows = dim(m)[1]
n.cols = length(m)  # or dim(m)[2]
tail(m)
rownames(m)[n.rows - 2] = "Monica's present"
m$hp[30] = 25
M = median(m$hp)
mean(m$mpg[m$hp > M])      # high-power mileage
mean(m$mpg[m$hp < M])      # low-power mileage
m$price = 1000*(1:n.rows)  # add column
m$vs = NULL                # delete column
sorted = m[order(m$cyl, m$disp), ] # sort by cyl, then by disp
```

(Data frame) File input and output (and ".csv" for Excel)

- `write.table(x, file = "", ...)` writes `x` to `file`. Variants include `write.csv(x, file = "")`.

  e.g. `write.csv(m, file = "mtcarsMonica.csv")` saves `m` (our corrupted `mtcars`) as comma-separated values (csv)

- `table = read.table(...)` reads from a file into a `data.frame`. Variants include
  - `table = read.csv(file, header = TRUE, row.names = 1)` for a `file` of comma-separated values with a header row of column names and a first column of row names; e.g.

    `monica = read.csv("mtcarsMonica.csv", row.names = 1)`
  - `table = read.csv(file, header = FALSE, col.names = c(...), row.names = c(...))` for a `file` of unlabeled data

Formula

A *formula* of the form `y ~ model` indicates that `y` depends on the variables in `model`. e.g. Here's a preview of the use of formulas in the coming handouts on graphics and regression.

- Here's a lousy boxplot that obscures the dependence of flower length on flower species:

  ```
  flowers = read.csv("flowers.csv")
  str(flowers) # note the factor
  boxplot(flowers$Flower.Length,
          main="Flower Length Without Regard for Species", ylab="Length (mm)")
  ```

  Improve the graph: `boxplot(formula, ...)` makes multiple plots of data specified by `formula`. e.g. This triple plot reveals the dependence of length on species as a grouping variable:

  ```
  boxplot(flowers$Flower.Length ~ flowers$Species,
          main="Flower Length by Species", xlab = "Species", ylab="Length (mm)")
  ```

- Here are similar examples using `mtcars`:

  ```
  boxplot(m$disp)
  boxplot(m$disp ~ m$am)
  ```

- We'll use formulas in linear regression soon:
  - `y ~ x` indicates that y depends linearly on x, as in the simple linear regression model, $y = a_1 + a_2 x$
  - `y ~ x1 + x2 + x3 + x1*x2` indicates that y depends linearly on x_1, x_2, x_3, and $x_1 \cdot x_2$, as in the multiple linear regression model, $y = a_1 + a_2 x_1 + a_3 x_2 + a_4 x_3 + a_5 x_1 \cdot x_2$.

5. (Base) Graphics

Common parameters

- `formula, data`: a formula of the form `y ~ model` and a data frame containing the variables
- `main`, `sub`; `xlab`, `ylab`: main title, subtitle; x-axis, y-axis labels
- `xlim`, `ylim`, each a 2-vector (low, high): x-axis, y-axis limits
- `pch`: plotting character (see `?points`)
- `cex` (symbols), `cex.axis`, `cex.lab`, `cex.main`, `cex.sub`: character expansion (relative to `1`)
- see `?par` for others

Numeric data

- `boxplot(x)` makes a boxplot from vector `x`; `boxplot(x ~ g)` groups by factor `g`; e.g.

  `boxplot(mtcars$mpg, main="Gas mileage", ylab="miles per gallon", ylim=c(0,40))`

  `boxplot(mpg ~ factor(cyl), data=mtcars, xlab="cylinders", ylab="miles per gallon")`
- `stripchart(x, method="overplot")` makes a dot plot of `x` (better than boxplot for small sample); `stripchart(x ~ g)` groups by `g`; `method` handles duplicates: `"overplot"`, `"jitter"`, or `"stack"`; e.g. `stripchart(mpg ~ factor(am), data=mtcars, method="stack")`
- `hist(x, breaks="Sturges", freq=NULL)`, makes a histogram from `x`, where `breaks` is a vector of bin boundaries (or, as in the default `"Sturges"`, the name of a bin algorithm); `freq=FALSE` gives density histogram instead of frequency; e.g. `hist(mtcars$mpg)`
- `plot(x, y)` makes a scatterplot from vectors `x` and `y`; e.g. `x = 1:5; y = 2*x; plot(x, y)`, `plot(x, y, xlim=c(0,10), ylim=c(0,10))`
- `points(x, y)` adds points to a plot, and `lines(x, y)` adds line segments; e.g. `points(x, x, pch=15)`; `lines(x=c(1,3,5,7,9), y=c(8,1,4,1,8), col="red")`
- `plot(density(x))` makes a *density plot* (usually better than a histogram) from `x`; `rug(x)` adds the data points; e.g. `plot(density(mtcars$mpg)); rug(mtcars$mpg)`

  (note: `density(x)` estimates density $f(x)$, returning a list including `(x, y)`, where $y \approx f(x)$)
- `pairs(x)` makes a matrix of scatterplots of pairs of columns of data frame `x`; e.g. `pairs(mtcars)`
- `curve(expr, from=NULL, to=NULL, n=101, add=FALSE, type="l")` draws a curve of `expr` over `[from, to]` (`add=TRUE` $\Longrightarrow$ add to existing plot); e.g.

  `curve(expr=x*sin(1/x), from=-pi/6, to=pi/6, n=200); curve(expr=x*1, add=TRUE, col="red")`

Legends; math expressions in titles and labels

`legend(x, y, legend, col=par("col"), lty, pch)` makes a legend at `(x, y)` (or `x` can be one of {`"bottomright"`, etc.}: see `?legend`) using labels, colors, line types, and plotting characters in vectors `legend`, `col`, `lty`, and `pch`; e.g.

```
legend("top", legend=c("x*sin(1/x)", "x"), col=c("black", "red"), lty=c(1, 1))
```

Use `expression(...)` in character string used as `main` or `xlab` or `ylab`; see `?plotmath`. e.g.

```
legend("top", legend=c(expression(x*sin(frac(1,x))), "x"), col=c("black", "red"), lty=c(1, 1))
```

Categorical data

- `barplot(height, names.arg = NULL)` makes a barplot of the counts in `height`, with (optional) bar labels in `names.arg`; e.g.

  ```
  counts = table(mtcars$cyl); barplot(counts)
  ```

- `mosaicplot(x)` makes a mosaic plot from a table of counts from `table()`; e.g.

  ```
  counts = table(mtcars$cyl, mtcars$gear); mosaicplot(counts)
  ```

Multiple figures

`matrix(data, nrow, ncol, byrow=FALSE)` fills an `nrow` $\times$ `ncol` matrix by column from `data`

`layout(mat)`, for matrix `mat`, divides graph so ith figure is drawn where `mat==i` (`0` $\implies$ blank)

`layout.show(n=1)` shows outlines of next `n` figures; e.g.

```
m = matrix(data=c(1, 0, 2, 3, 3, 3), nrow=2, ncol=3, byrow=TRUE)
layout(m)
layout.show(3)
hist(mtcars$mpg)              # 1st plot: (frequency) histogram alone
plot(density(mtcars$mpg))     # 2nd plot: density plot alone
hist(mtcars$mpg, freq=FALSE)  # 3rd plot: density histogram
lines(density(mtcars$mpg))    # add density plot to (3rd plot) histogram
layout(matrix(data=1, nrow=1, ncol=1)) # reset graphics device
```

Write graphical output to a file

- Open a graphical output file with, e.g., `pdf("file.pdf")`, `png("file.png")`, `jpeg("file.jpg")`, `bmp("file.bmp")`, `postscript("file.ps")`, `tiff("file.tif")`
- Make graph
- Close the file with `dev.off()`

6. Statistical Tests and Confidence Intervals

One Mean or the Difference of Two Means

`out = t.test(x, y = NULL, alternative = "two.sided", mu = 0, conf.level = .95)` tests $H_0 : \mu_X = \mu_0 =$ `mu` for a sample `x` from a normal population; or, if `y` is given, $H_0 : \mu_X - \mu_Y = \mu_0 =$ `mu`, for samples `x` and `y` from normal populations. `out` is a list containing (among other things):

- `$parameter`: degrees of freedom ($n - 1$, where $n =$ `length(x)`, if `y == NULL`; or a mess)
- `$statistic`: Student's t test statistic, $t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}}$; or $t = \frac{(\bar{x} - \bar{y}) - \mu_0}{\sqrt{\frac{s_X^2}{n_X} + \frac{s_Y^2}{n_Y}}}$
- `$p.value`: probability of a value at least as extreme as t under H_0
- `$conf.int`: confidence interval for μ_X (or $\mu_X - \mu_Y$) corresponding to H_1 in `alternative`
- `$estimate`: $\bar{x}$ (or $\bar{x}$ and $\bar{y}$)

Other `alternative` choices are `"less"` and `"greater"`. e.g.

```
x = rnorm(n = 10, mean = 0, sd = 1); (out = t.test(x))
x = rnorm(10, 0, 1); (out = t.test(x, mu = 2))  # rnorm() isn't part of the test!
x = rnorm(10, 0, 1); y = rnorm(10, 2, 1); (out = t.test(x, y))
x = rnorm(10, 0, 1); y = rnorm(10, 2, 1); (out = t.test(x, y, mu = -2))
```

F Test for Equality of Variances

`out = var.test(x, y, ratio = 1, alternative = "two.sided", conf.level = .95)` tests $H_0 : \frac{\sigma_X^2}{\sigma_Y^2} =$ `ratio` for two samples `x` and `y` from normal populations. `out` is a list containing:

- `$parameter`: degrees of freedom ($n_X - 1$ and $n_Y - 1$, where $n_X =$ `length(x)` and $n_Y =$ `length(y)`)
- `$statistic`: F test statistic, $f = \frac{s_X^2/\sigma_X^2}{s_Y^2/\sigma_Y^2} = \frac{s_X^2}{s_Y^2} \cdot \frac{1}{\texttt{ratio}}$
- `$p.value`: probability of a value at least as extreme as f under H_0
- `$conf.int`: confidence interval for $\frac{\sigma_X^2}{\sigma_Y^2}$
- `$estimate`: $\frac{s_X^2}{s_Y^2}$

e.g. `x = rnorm(100, 0, 1); y = rnorm(10, 0, 2); (out = var.test(x, y, ratio = 1))`

e.g. `x = rnorm(100, 0, 1); y = rnorm(10, 0, 2); (out = var.test(x, y, ratio = .25))`

Chi-Squared Tests

- Goodness-of-fit:

  `counts = c(...); probs = c(...); (out = chisq.test(x = counts, p = probs))` tests H_0 : "`counts` came from a distribution with probabilities `probs`". e.g.

  ```
  counts=c(12,15,17,6); probs=c(.20,.25,.40,.15); (out=chisq.test(x=counts, p=probs))
  ```

  `out` is a list containing (among other things):

  - `$parameter`: degrees of freedom (#categories − 1 == `length(x) - 1`)
  - `$statistic`: χ^2 test statistic for goodness-of-fit of observed `counts` to proposed `probs`
  - `$p.value`: probability of a value at least as extreme as χ^2 under H_0

- Independence / Homogeneity

  e.g. Consider the counts in this contingency table:

	Smoking status			
Education	Nonsmoker	Former	Moderate	Heavy
Primary	56	54	41	36
Secondary	37	43	27	32
University	53	28	36	16

  To get data into the test, we need x = `matrix(data, nrow, ncol, byrow = FALSE)`, which fills an `nrow` by `ncol` matrix `x`, by column, from the vector `data`. Note that `x[,c]` is the c$^{\text{th}}$ column of `x`, and `x[r,]` is the r$^{\text{th}}$ row. e.g.

  ```
  (x = matrix(data = c(56,37,53, 54,43,28, 41,27,36, 36,32,16), nrow=3, ncol=4))
  ```

  The χ^2 test `out = chisq.test(x)` is for H_0 : "row and column variables are independent" (or H_0 : "the column populations have the same distribution with respect to the row variable").

  `out` is a list containing (among other things):

  - `$parameter`: degrees of freedom, (#rows − 1) × (#columns − 1)
  - `$statistic`: χ^2 test statistic for independence of row and column variables (or for homogeneity of column populations with respect to row variable)
  - `$p.value`: probability of a value at least as extreme as χ^2 under H_0
  - `$expected`: expected counts under H_0

  To use `chisq.test()` on variables in a data frame, recall that `table(...)` makes a contingency table of counts of each combination of factors in `...`. e.g.

  ```
  table(mtcars$cyl)
  table(mtcars$cyl, mtcars$gear)
  ```

One Proportion or the Difference of Two Proportions

- For integers `x` and `n`, `out = prop.test(x, n, p, alternative = "two.sided", conf.level = .95)` tests $H_0 : p = p_0 =$ `p` for a sample containing `x` successes in `n` trials. e.g.

  `x = 800; n = 1000; p0 = .77; (out = prop.test(x, n, p0, correct=FALSE))`

  (`correct=FALSE` disables a good continuity correction that would add to the explanation.)

  `out` is a list containing (among other things):

  - `$parameter`: degrees of freedom (#categories $- 1 = 2$ (success and failure) $- 1 = 1$)
  - `$statistic`: χ^2 test statistic for goodness-of-fit of observed counts, `x` successes (800) and `n-x` failures ($1000 - 800 = 200$), to the distribution with expected counts, `n*p` successes ($770 = .77 \times 1000$) and `n*(1-p)` failures ($230 = (1 - .77) \times 1000$).
  - `$p.value`: probability of a value at least as extreme as χ^2 under H_0
  - `$conf.int`: confidence interval for p
  - `$estimate`: $\hat{p} =$ `x/n`

  (I teach an equivalent z-test for this one-proportion test:

  `phat = x/n; z = (phat - p0) / sqrt(p0*(1-p0)/n); (p.value = 2*pnorm(-abs(z)))`

  Then `z^2` matches `out$statistic` above, and the P-values are the same.

)

- For 2-vectors `x` and `n`, `out = prop.test(x, n, alternative = "two.sided", conf.level = .95)` tests $H_0 : p_1 - p_2 = 0$ (or $p_1 = p_2$) for samples from two populations containing `x[1]` successes in `n[1]` trials and `x[2]` successes in `n[2]` trials, respectively. e.g.

  `x = c(40, 87); n = c(244, 245); (out = prop.test(x, n, correct=FALSE))`

  `out` is a list containing (among other things):

  - `$parameter`: degrees of freedom, 4 (counts) $-$ 2 (constraints due to the sample sizes) $-$ 1 (parameter estimated, $\hat{p}$) $= 1$

    $$\left(\begin{array}{l|ll} & \text{Sample} & \\ \text{Outcome} & 1 & 2 \\ \hline \text{success} & 40 & 87 \\ \text{failure} & 244 - 40 & 245 - 87 \end{array}\right)$$

  - `$statistic`: χ^2 test statistic for goodness-of-fit of observed counts, `x[1]` and `x[2]` successes (40 and 87) and `n[1]-x[1]` and `n[2]-x[2]` failures ($244 - 40$ and $245 - 87$) to the distribution with corresponding expected counts based on $\hat{p} = \frac{x_1+x_2}{n_1+n_2}$
  - `$p.value`: probability of a value at least as extreme as χ^2 under H_0
  - `$conf.int`: confidence interval for the difference in proportions $p_1 - p_2$
  - `$estimate`: a 2-vector containing $\hat{p}_1 =$ `x[1]/n[1]` and $\hat{p}_2 =$ `x[2]/n[2]`

  (Here, too, I teach an equivalent z-test, with `z^2 == out$statistic`, and the same P-value.)

7. Simple Linear Regression, $y = mx + b$

e.g. `cars` is a built-in data.frame: `cars`, `?cars`, `str(cars)`, `head(cars)`

- (Recall) `plot(x, y)` makes a (base graphics) scatterplot of data in the vectors `x` and `y`; e.g.

  ```
  plot(x=cars$speed, y=cars$dist)
  ```

- `cor(x, y)` gives the correlation of vectors `x` and `y`; e.g. `r = cor(x = cars$speed, y = cars$dist)`

  For data frame `x`, `A = cor(x)` gives a matrix `A` such that `A[i, j] == cor(x[ , i], x[ , j])`, the correlation of `A`'s i^{th} and j^{th} columns; e.g. `cor(mtcars[ , 1:3])`

- `lm(y ~ x, data)` calculates a linear regression model $y = mx + b$ from the `y` and `x` variables in the data.frame `data` (this uses the "`formula, data`" interface mentioned earlier); e.g.

  ```
  m = lm(dist ~ speed, data = cars) # "m" is for "model"
  str(m)
  summary(m) # summary
  anova(m) # ANOVA table
  ```

- `m$coefficients` is a vector containing y-intercept b and slope m:

  ```
  y.intercept = m$coefficients[1]
  slope = m$coefficients[2]
  ```

- `abline(a, b)` adds a line $y =$ `a` $+$ `b`x, and `abline(reg)` adds the line from model `reg`; e.g.

  ```
  abline(a = y.intercept, b = slope) # add regression line
  abline(reg = m) # same as previous line
  abline(a = mean(cars$dist), b = 0, lty = "dashed") # horizontal line through mean y
  ```

- `predict(model, newdata)` gives $\hat{y}$ from `model` evaluated at x (or at $x_1, \ldots, x_p$ in the multiple regression case) in data.frame `newdata`; e.g. Our model's x is `speed`; so put speeds for which we want predictions in a data.frame with a `speed` column:

  ```
  d = data.frame(speed = seq(from=5, to = 25, by = 5))
  y.hat = predict(m, newdata = d)
  # add (x, y) pairs to graph with plotting character 19, scaled by 3
  points(x=d$speed, y=y.hat, pch=19, cex=3)
  ```

- In the simple regression model $y_i = mx_i + b + \varepsilon_i$, errors ε_i are assumed to be random and independent, with $\varepsilon_i \sim N(0, \sigma)$. To check these assumptions, a *residual plot* of points $\{(\text{fitted value} = \hat{y}_i, \text{residual} = e_i = y_i - \hat{y}_i)\}$ should show no pattern (if errors are random and independent) or varying vertical spread (if errors have the same standard deviation σ); e.g.

  ```
  plot(m$fitted.values, m$residuals)
  abline(0, 0) # y = 0 + 0x; errors should have mean 0
  ```

- A *QQ plot* shows quantiles of a data distribution, like our residuals, on the y-axis against the same quantiles of a reference distribution, like $N(\mu =$`mean(residuals)`$, \sigma =$ `sd(residuals)`$)$. If the assumption of normal errors is met, these points should be close to a line. `qqline(x)` adds a line through the first and third quantile pairs. e.g.

```
x = rnorm(n=100); qqnorm(x);                qqline(x) # 100 random N(0, 1) points
w = rexp(100);    qqnorm(w, ylim=c(-1, 5)); qqline(w) # 100 random Exp(1)  points

qqnorm(m$residuals); qqline(m$residuals) # our "dist vs. speed" model
```

  Or use `plot(m)` to see the residual and QQ plots, and two others, in one step:

```
layout(matrix(data=1:4, nrow=2, ncol=2, byrow=TRUE))
plot(m)
layout(matrix(data=1, nrow=1, ncol=1)) # reset graphics device
```

Multiple Linear Regression, $y = a_0 + a_1x_1 + \cdots + a_px_p$

e.g. `y ~ x1 + x2 + x3 + x1*x2` indicates that y depends linearly on x_1, x_2, x_3, and $x_1 \cdot x_2$, as in the multiple linear regression model, $y = a_0 + a_1x_1 + a_2x_2 + a_3x_3 + a_4x_1 \cdot x_2$.

```
n = 100 # simulate n points, (y, x1, x2, x3), for a "sanity check" example
x1 = rnorm(n=n, mean=0, sd=1); x2 = rnorm(n); x3 = rnorm(n)
y = 3 + 4*x1 + 5*x2 + 6*x3 + 7*x1*x2
m = lm(y ~ x1 + x2 + x3 + x1*x2) # use lm() to discover coefficients from data
summary(m)

y = 3 + 4*x1 + 5*x2 + 6*x3 + 7*x1*x2 + rnorm(n) # add noise to make it harder
m2 = lm(y ~ x1 + x2 + x3 + x1*x2)
summary(m2)

m3 = lm(mpg ~ hp + wt + gear, data=mtcars) # real data from mtcars:
summary(m3)
anova(m3)
```

Inference on the coefficients is facilitated by `summary(model)`, which gives

- estimated coefficients $a_0, a_1, \cdots, a_p$
- estimated standard deviations of coeffiecients, $s_{a_0}, \cdots, s_{a_p}$
- the F statistic and P-value for $H_0 : a_1 = \cdots = a_p = 0$
- for each coefficient a_i, the t statistic and P-value for $H_0 : a_i = 0$

`confint(m, level = .95)` gives confidence intervals for the coefficients

8. Simulation by replicating a calculation

Make random number generation repeatable

`set.seed(seed)`, for integer `seed`, sets starting point of (pseudo-)random number generation. e.g.

```
a = rnorm(1); b = rnorm(1); a == b
set.seed(0); a = rnorm(1); set.seed(0); b = rnorm(1); a == b
```

Repeat a calculation n times

`replicate(n, expr)` returns a vector (or matrix or array) of `n` evaluations of `expr`. e.g.

```
x = replicate(n=4, expr=rnorm(1))       # 4 random samples of size 1
y = replicate(n=4, expr=rnorm(3))       # 4 random samples of size 3
z = replicate(n=4, expr=mean(rnorm(3))) # 4 means of samples of size 3
```

`expr` can be compound in curly braces, `{ ... }`; its value is that of its last expression. e.g.

```
w = replicate(n=4, expr={ mu=7; sigma=3; x=rnorm(n=3, mean=mu, sd=sigma); mean(x) })
```

Distributions

Check `?distributions`. (Recall prefixes `d, p, q, r` for *density*, *probability*, *quantile*, *random*.)

Let's simulate a few distributions.

- $N(\mu, \sigma)$: First, confirm that $\bar{x}$ is close to μ and that s is close to σ:

  `mu = 7; sigma = 3; mean(x <- rnorm(n=1000, mean=mu, sd=sigma)); sd(x)`

  Second, the *Central Limit Theorem* (CLT) says that for a large sample from (almost) any distribution with finite μ and σ, $\bar{X} \approx N(\mu, \frac{\sigma}{\sqrt{n}})$.

  e.g. Consider $U(0,1)$, which has $\mu = \frac{\max - \min}{2} = \frac{1}{2}$ and $\sigma = \sqrt{\frac{(\max - \min)^2}{12}} = \sqrt{\frac{1}{12}}$. Simulate CLT by finding many sample means from samples from $U(0,1)$:

  ```
  curve(dunif(x, min=0, max=1), from=-0.1, to=1.1, ylim=c(0,8), lty=2) # U(0,1)
  n =  30 # sample size (also try n=1 to see CLT fail)
  N = 100 # number of samples
  x.bars = replicate(n=N, expr=mean(runif(n=n, min=0, max=1))) # vector of sample means
  mean(x.bars) # should be near 1/2
  sd(x.bars)   # should be near sqrt(1/12)/sqrt(n), about .0527
  curve(dnorm(x, mean=1/2, sd=sqrt(1/12)/sqrt(n)), from=0, to=1, lty=3, add=TRUE) # CLT
  lines(density(x.bars), lty=1) # sampling distribution of bar(x)
  rug(x.bars)
  legend(x="topright", legend=c("U(0,1)", "CLT", expression(bar(X))), lty=c(2,3,1))
  ```

- t_{n-1}: For a random sample $X_1, \ldots, X_n$ from $N(\mu, \sigma)$, the quantity $T = \frac{\bar{X}-\mu}{s/\sqrt{n}}$ follows the *Student's t* distribution with $n-1$ degrees of freedom, denoted t_{n-1}.

  e.g. Simulate t_{n-1} for $n = 6$:

  ```
  n =   6 # sample size
  N = 100 # number of samples
  mu = 7
  sigma = 3
  t = replicate(N, { x=rnorm(n, mean=mu, sd=sigma); (mean(x) - mu)/(sd(x)/sqrt(n)) })
  plot(density(t)) # sampling distribution of T ~ t_{n-1}
  rug(t)
  curve(dt(x, df=n-1), lty="dashed", add=TRUE) # true t_{n-1}
  curve(dnorm(x, mean=0, sd=1), lty="dotted", add=TRUE) # add N(0, 1) for reference
  legend(x="topright", legend=c(expression("true "*t[n-1]), "simulated t", "N(0, 1)"),
         lty=c("dashed", "solid", "dotted"))
  ```

- χ^2_n: If $Z_1, \cdots, Z_n$ are independent, $N(0, 1)$ random variables, then $X^2 = \sum_{i=1}^n Z_i^2 \sim \chi^2_n$. ...
- F_{n_1,n_2}: If $X \sim \chi^2_{n_1}$ and $Y \sim \chi^2_{n_2}$ are independent, then $\dfrac{X/n_1}{Y/n_2} \sim F(n_1, n_2)$. ...

What is a P-value?

A P-value is the probability, assuming H_0 is true, of getting data, as summarized by the test statistic, more extreme than the sample data. e.g. Guinness says pouring a glass should take 119.5 seconds. Here's a random sample of times from a server:

`x = c(118, 121, 113, 116, 117, 112, 113)`

Is this server pouring correctly? Test $H_0 : \mu = 119.5$ vs. $H_1 : \mu \neq 119.5$: `(out = t.test(x, mu=119.5))`

Simulate P-value by seeing how often `t`, from random samples, is greater than `out$statistic`:

```
mu = 119.5
sigma = sd(x)
n =  length(x) # sample size
N = 1000 # number of replicates
t = replicate(N, { x=rnorm(n, mean=mu, sd=sigma); (mean(x) - mu)/(sd(x)/sqrt(n)) })
more.extreme = (abs(t) > abs(out$statistic))
(simulated.p.value = sum(more.extreme) / N)
out$p.value

plot(density(t), main=bquote(.(N) * " Simulated t statistics")) # visualize P-value
rug(t)
points(x=out$statistic, y=0, pch=19, col="red")
text(x=out$statistic, y=.02, labels="out$statistic")
```

STAT 327-1 (also -4 and -7): Introductory Data Analysis with R

Course outcome: Students will use R to manipulate data and perform exploratory data analysis using introductory statistics.

Unit	Objectives Students will:	Assessment	Read, View, Do
1 **Build basic R vocabulary**	1. Use R as a calculator. 2. Use Rs distribution functions: calculate probability mass/density, cumulative probability distribution, and quantile functions and generate random numbers. 3. Run a line of R code in the console and a batch from a script. 4. Use R Markdown to write reports integrating text, data, R code, and its output.	Q1 HW1 (trivial script) (Q = online quiz, HW = homework)	1calculator.pdf, lecture 1
2 **Maniuplate data in R**	1. Manipulate data to create vectors, lists, and data frames. 2. Summarize a data set. 3. Select a subset of a data set. 4. Use a factor vector for categorical data. 5. Load clean tabular data sets into R. 6. Save R data sets as text or csv files.	Q2, Q3, Q4 HW2 (donations to 2012 elections), HW3 (Boston housing)	2vector.pdf, lecture 2 3list.pdf, lecture 3 4dataFrame.pdf, lecture 4
3 **Produce graphics**	1. Use the graphics base package to create displays of data: scatterplots, boxplots, histograms and density plots. 2. Customize graphical layout, annotations, and legends.	Q5 HW3, HW4	5graphics.pdf, lecture 5 group work (graphics)
4 **Apply statistical methods**	1. Run classical statistical procedures including confidence intervals, t tests, Z tests for proportions, F tests for variance, and χ^2 tests. 2. Do basic linear regression analysis and ANOVA.	Q6, Q7 HW4 (test, regression, confidence bands)	6test.pdf, lecture 6 7regression.pdf, lecture 7 8simulation.pdf, lecture 8 group work (tests)
5 **Run basic simulations**	1. Replicate a calculation to simulate properties of distributions. 2. Simulate data fitting $N(0,1)$, t, χ^2, and F distributions.		8simulation.pdf, lecture 8
		Written exam	

Prerequisite: Introductory statistics

R Markdown

R Markdown is software included with RStudio that allows you to put text, data, R code, and Latex math notation in the same plain-text file, and then compile it to a nicely formatted file containing text, data, R code, textual output of R code, graphical output of R code, and math notation. By putting all these things in a single file, R Markdown greatly simplifies the otherwise tedious and error-prone process of writing and assembling a statistical report.

Here's all you have to know for STAT 327

- To open a new file, use RStudio's menu "File > New file > R Markdown ...", give your file a name ending "`.Rmd`", choose "HTML" under "Default Output Format:", and click "OK".

- Write R code inside "code chunks" delimited as follows:

````
```{r}
  # R code
```
````

  (These "backquotes" are on the upper-left corner of the keyboard.)

- Write plain text anywhere in the file except in code chunks.

- Click "KnitHTML" to knit together your text, data, R code, and its output into a web page.

- For debugging, run a line of code in the console with "Ctrl-Enter" (Windows) or "Command-Enter" (Mac). See the "Chunks" menu for running a chunk at a time.

To learn more about R Markdown

Use RStudio's "?" menu to choose "Using R Markdown" and "Markdown Quick Reference".

See cheatsheets at `http://www.rstudio.com/resources/cheatsheets`. There are four; start with "R Markdown Cheat Sheet" and "R Markdown Reference Guide".

Latex for mathematical notation (optional)

In R Markdown text, you may use Latex mathematical notation in sections delimited by `$ ... $` to show up inline, or by `$$ ... $$` to show up as a separate paragraph. Here are basics:

Latex	Result
`x^y`	x^y
`x_y`	x_y
`\alpha, \mu, \sigma`	α, μ, σ
`\bar{x}`	$\bar{x}$
`\hat{x}`	$\hat{x}$
`\sqrt{x}`	$\sqrt{x}$
`\sum`	$\sum$
`\frac{x}{y}`	$\frac{x}{y}$

e.g. `$Z = \frac{\bar{x} - \mu_0}{\sigma / \sqrt{n}}$` gives $Z = \frac{\bar{x} - \mu_0}{\sigma / \sqrt{n}}$.

e.g. `$$\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i$$` gives

$$\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i.$$

To learn more about Latex, see `http://en.wikibooks.org/wiki/LaTeX`.