Learning R

Patrick Lam

setting up

- download R from CRAN
- work in the console (code not saved)
- open a script, type code in script, and save as a .R file

Example code is in red

R as calculator

```
> 5 + 4
[1] 9

> 8 * 2 - sqrt(9)
[1] 13

> log(4)/9^2
[1] 0.01711475
```

objects

R is an object-oriented programming language. Use `<-` as assignment operator for objects.

```
> 5 + 4
[1] 9
> my.sum <- 5 + 4
> my.sum
[1] 9
```

```
> my.name <- "Patrick"
> my.name
[1] "Patrick"
```

vectors

All objects consist of one or more **vectors**.

vector: a combination of elements (i.e. numbers, words), usually created using `c()`, `seq()`, or `rep()`

```
> empty.vector <- c()
> empty.vector
NULL

> one.to.five <- c(1, 2, 3, 4, 5)
> one.to.five
[1] 1 2 3 4 5

> poli.sci <- c("theory", "amer.", "comp.", "ir")
> poli.sci
[1] "theory" "amer."  "comp."  "ir"
```

```
> one.to.ten <- 1:10
> one.to.ten
 [1]  1  2  3  4  5  6  7  8  9 10

> two.to.five <- seq(from = 2, to = 5, by = 1)
> two.to.five
[1] 2 3 4 5

> all.fours <- rep(4, times = 5)
> all.fours
[1] 4 4 4 4 4
```

All elements in a vector must be of the same data type!

data types

- numeric
- character
- logical

numeric: numbers

```
> three <- 3
> three
[1] 3
```

```
> is.numeric(three)
[1] TRUE
```

```
> as.numeric("3")
[1] 3
```

character: for example, words or phrases (must be in ’”’)

```
> president <- "Barack Obama"
> president
[1] "Barack Obama"
```

```
> is.character(president)
[1] TRUE
```

```
> as.character(3)
[1] "3"
```

logical: `TRUE (T)` or `FALSE (F)`

```
> num.vec <- c(5, 6, 4)
> logical.vec <- num.vec == 6
> logical.vec
[1] FALSE  TRUE FALSE
> is.logical(logical.vec)
[1] TRUE
```

can also be represented as numeric 1 or 0:

```
> as.numeric(logical.vec)
[1] 0 1 0
```

All elements in a vector must be of the same data type!

- if a vector has a character element, all elements become character

```
> mixed.vec <- c(5, "Patrick", TRUE)
> mixed.vec
[1] "5"       "Patrick" "TRUE"
```

- if a vector has both numeric and logical elements, all elements become numeric

```
> mixed.vec2 <- c(10, FALSE)
> mixed.vec2
[1] 10  0
```

character > numeric > logical

object classes

All objects consist of one or more vectors.

In addition to vector, objects can be of one of the following classes:

- matrix
- array
- dataframe
- list

matrix

A matrix is a two-dimensional ($r \times c$) object (think a bunch of stacked or side-by-side vectors).

```
> a.matrix <- matrix(c(1, 2, 3, 4), nrow = 2, ncol = 2)
> a.matrix

     [,1] [,2]
[1,]    1    3
[2,]    2    4

> class(a.matrix)

[1] "matrix"
```

All elements in a matrix must be of the same data type.
character > numeric > logical

array

An array is a three-dimensional ($r \times c \times h$) object (think a bunch of stacked $r \times c$ matrices).

All elements in an array must be of the same data type (character $>$ numeric $>$ logical).

```
> an.array <- array(0, dim = c(2, 2, 3))
> an.array
, , 1

     [,1] [,2]
[1,]    0    0
[2,]    0    0

, , 2

     [,1] [,2]
[1,]    0    0
[2,]    0    0

, , 3

     [,1] [,2]
[1,]    0    0
[2,]    0    0
```

dataframe

A dataframe is a two-dimensional ($r \times c$) object (like a matrix).

- each column must be of the same data type, but data type may vary by column
- regression and other statistical functions usually use dataframes
- use `as.data.frame()` to convert matrices to dataframes

list

A list is a set of objects.

Each element in a list can be a(n):

- vector
- matrix
- array
- dataframe
- list

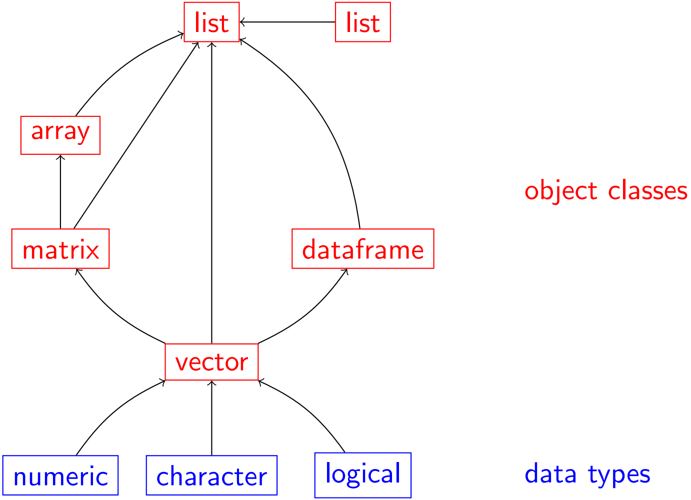
```
> a.vec <- 6:10
> a.matrix <- matrix(3, nrow = 2, ncol = 2)
> a.dataframe <- as.data.frame(a.matrix)
> a.list <- list(a.vec, a.matrix, a.dataframe)
> a.list

[[1]]
[1]  6  7  8  9 10

[[2]]
     [,1] [,2]
[1,]    3    3
[2,]    3    3

[[3]]
  V1 V2
1  3  3
2  3  3
```

brief review



exercises

1. Create a vector of integers from 1 to 20.
2. In one line of code, add 2, multiply by 5, take the square root, and then take the log of each element in the vector.
3. Create a vector of your 5 favorite cities.
4. Create a 3×3 matrix where each element of every column corresponds to the column number.
5. Convert this matrix into a dataframe.
6. Create a $3 \times 5 \times 2$ array of all 0s.
7. Create a list containing your array, your dataframe and your two vectors.

solutions

1. `> ans.1 <- 1:20`
2. `> ans.2 <- log(sqrt((ans.1 + 2) * 5))`
3. ```
   > ans.3 <- c("Los Angeles", "Las Vegas", "Hong Kong", "San Francisc
   +      "Boston")
   ```
4. `> ans.4 <- matrix(c(1, 2, 3), ncol = 3, nrow = 3, byrow = T)`
5. `> ans.5 <- as.data.frame(ans.4)`
6. `> ans.6 <- array(0, dim = c(3, 5, 2))`
7. `> ans.7 <- list(ans.6, ans.5, ans.3, ans.2)`
   ```

combining objects

To combine vectors together or lists together, use `c()`

```
> vec1 <- c(4, 6, 9)
> vec2 <- 10:15
> comb.vec <- c(vec1, vec2)
> comb.vec
[1]  4  6  9 10 11 12 13 14 15
```

To combine matrices or dataframes with other matrices, dataframes, or vectors, use `cbind()` or `rbind()`

```
> a.matrix <- matrix(0, nrow = 2, ncol = 3)

> rbind(a.matrix, 1:3)

     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
[3,]    1    2    3

> cbind(a.matrix, a.matrix)

     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    0    0    0    0    0    0
[2,]    0    0    0    0    0    0
```

Dimensions must match (think layered cake, not wedding cake).

names

It's helpful to give names to elements or rows/columns within objects (i.e. variable names).

Use

- `names()` for vectors, dataframes and lists
- `rownames()` and `colnames()`for matrices and dataframes
- `dimnames()` for arrays

```
> leaders <- c("Obama", "Brown", "Merkel")
> names(leaders) <- c("US", "UK", "Germany")
> leaders
      US        UK   Germany
 "Obama"  "Brown" "Merkel"
```

```
> country.names <- names(leaders)
> country.names
[1] "US"      "UK"      "Germany"
```

```
> leader <- c("Obama", "Brown", "Merkel")
> year <- rep(2009, times = 3)
> dataset <- data.frame(cbind(leader, year))

> names(dataset) <- c("leader", "year")
> dataset

  leader year
1  Obama 2009
2  Brown 2009
3 Merkel 2009
```

- `names()` and `colnames()` are the same for dataframes only
- must use `colnames()`for matrices

indexing

Elements within objects are indexed using `[]` and `[[]]`.

- vectors: `[i]` for the ith element
- matrices and dataframes: `[i,j]` for the ith row, jth column
- arrays: `[i,j,k]` for the ith row, jth column, kth level
- lists: [[i]] for the ith element

vectors:

```
> leaders
      US        UK   Germany
 "Obama"  "Brown" "Merkel"

> leaders[2]
     UK
"Brown"

> leaders[c(1, 3)]
      US   Germany
 "Obama" "Merkel"
```

```
> leaders["US"]
     US
"Obama"

> leaders[-3]
     US      UK
"Obama" "Brown"
```

matrices and dataframes:

```
> dataset

  leader year
1  Obama 2009
2  Brown 2009
3 Merkel 2009
```

```
> dataset[2, 1]

[1] Brown
Levels: Brown Merkel Obama
```

```
> dataset[2, ]

  leader year
2  Brown 2009
```

```
> dataset[, "year"]

[1] 2009 2009 2009
Levels: 2009
```

```
> dataset[, -1]

[1] 2009 2009 2009
Levels: 2009
```

for datasets:

```
> dataset$leader

[1] Obama  Brown  Merkel
Levels: Brown Merkel Obama
```

lists:

```
> my.list <- list(mat = a.matrix, vec = leaders)
> my.list

$mat
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0

$vec
      US        UK   Germany
 "Obama"  "Brown" "Merkel"

> names(my.list)

[1] "mat" "vec"
```

```
> my.list[[2]]

      US        UK   Germany
 "Obama"  "Brown" "Merkel"


> my.list[[2]][2]

     UK
"Brown"


> my.list$mat

     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
```

editing objects

Now that we know indexing, editing and subsetting objects is trivial.

For example, we can add to an object (such as adding a variable to a dataset):

```
> dataset$europe <- c(0, 0, 1)
> dataset

  leader year europe
1  Obama 2009      0
2  Brown 2009      0
3 Merkel 2009      1
```

or edit an object:

```
> dataset[2, "europe"] <- 1
> dataset

  leader year europe
1  Obama 2009      0
2  Brown 2009      1
3 Merkel 2009      1
```

subsetting objects

with logical statements:

```
> dataset$europe == 1
[1] FALSE  TRUE  TRUE
> dataset[dataset$europe == 1, ]
  leader year europe
2  Brown 2009      1
3 Merkel 2009      1
```

==, ! =, >, <, >=, <=
&, |, %in%

the R environment

Any objects you create will be stored in the R environment.

To see all the objects in your environment:

```
> ls()
 [1] "a.dataframe"     "a.list"          "a.matrix"        "a.vec"
 [5] "add.vec"         "all.fours"       "an.array"        "ans.1"
 [9] "ans.2"           "ans.3"           "ans.4"           "ans.5"
[13] "ans.6"           "ans.7"           "beta.func"       "beta.hat"
[17] "comb.vec"        "country.names"   "data.samp"       "dataset"
[21] "diff.func"       "empty.vector"    "first.draws"     "geomean.func"
[25] "i"               "j"               "just.trade"      "leader"
[29] "leaders"         "logical.vec"     "macro"           "macro.subset"
[33] "means"           "medians"         "mixed.vec"       "mixed.vec2"
[37] "my.list"         "my.name"         "my.sum"          "new.data"
[41] "new.macro"       "num.vec"         "one.to.five"     "one.to.ten"
[45] "PErisk"          "poli.sci"        "population"      "president"
[49] "pull.out.trade"  "quants"          "random.numbers"  "results"
[53] "results.vec"     "row.numbers"     "row.samp"        "samp"
[57] "samp.data"       "samp.w.probs"    "samp.w.rep"      "samp.wo.rep"
[61] "sds"             "second.draws"    "tally"           "test.mat"
[65] "test.matrix"     "test.vec"        "three"           "triangle.func"
[69] "trim.func"       "trimmed.1"       "trimmed.25"      "two.func"
```

packages

To use packages, you need to install them (do this once) and load them (every time you open R).

To install a package named `foo`:

1. type `install.packages("foo")`
2. choose a CRAN repository

To load a package named `foo`:

1. type `library(foo)`

loading datasets

Suppose you want to load the `foo` dataset.

If the dataset is in

- an existing R package, load the package and type `data(foo)`
- .RData format, type `load(foo)`
- .txt or other text formats, type `read.table("foo.txt")`
- .csv format, type `read.csv("foo.csv")`
- .dta (Stata) format, load the `foreign` library and type `read.dta("foo.dta")`

To save objects into these formats, use the equivalent `write.table()`, `write.csv()`, etc. commands.

working directory

When loading or saving a dataset or object, R will look in the current working directory.

If your working directory is not where the file is at, R will not find it, so make sure you change the working directory.

- to change to the `foo` working directory, use `setwd("foo")`
- to see the current working directory, type `getwd()`

exercises

1. Load the `macro` dataset from the `Zelig` package.
2. Change the name of the “year” variable in the dataset to “date”.
3. Add a column of just 1s to the left of the dataset.
4. Create a vector with just the “trade” variable from the dataset.
5. Create a new dataset with all the observations where ”gdp” is greater than 3.25 and “unem” is less than 5.
6. Write this new smaller dataset as a separate file into your working directory in any format (i.e. .csv, .dta, .txt)
7. Store the large dataset, the “trade” vector, and the new smaller dataset in a list with appropriate names. Then extract the “trade” vector from the list.

solutions

1. ```
   > library(Zelig)
   > data(macro)
   ```
2. ```
   > names(macro)[2] <- "date"
   ```
3. ```
   > new.macro <- cbind(1, macro)
   ```
4. ```
   > just.trade <- macro$trade
   ```
5. ```
   > macro.subset <- macro[macro$gdp > 3.25 & macro$unem < 5,
   +     ]
   ```
6. ```
   > write.csv(macro.subset, file = "macrosubset.csv")
   ```
7. ```
   > my.list <- list(large = new.macro, small = macro.subset,
   +     trade = just.trade)
   > pull.out.trade <- my.list$trade
   ```
   ```

analyzing vectors

- `mean()`
- `median()`
- `sd()`
- `var()`
- `cor()`
- `cov()`
- `quantile()`
- `max()`
- `min()`

other vector functions

- `sum()`
- `prod()`
- `length()`
- `table()`
- `unique()`
- `sort()`
- `order()`
- `which()` with logical statements

analyzing dataframes and matrices

- `head()`
- `tail()`
- `nrow()`
- `ncol()`
- `summary()`
- `colMeans()`
- `rowMeans()`
- `colSums()`
- `rowSums()`
- `View()`
- `edit()`

exercises

1. Reload the `macro` dataset from `Zelig`. Find the mean, median, standard deviation, and 20th and 80th percent quantiles of the unemployment variable.
2. How many observations are there in this dataset?
3. What's the correlation between trade and gdp?
4. Which country-year observation had the highest unemployment rate in the dataset?
5. Which country had the most number of years where gdp > 5?

solutions

1. ```
   > data(macro)
   > mean(macro$unem)
   [1] 4.993873
   > median(macro$unem)
   [1] 4.5
   > sd(macro$unem)
   [1] 3.240486
   > quantile(macro$unem, probs = c(0.2, 0.8))
        20%      80%
   1.880000 8.105564
   ```
2. ```
   > nrow(macro)
   [1] 350
   ```

3.
```
> cor(macro$trade, macro$gdp)
[1] -0.220669
```

4.
```
> macro[macro$unem == max(macro$unem), c("country", "year")]
    country year
119 Belgium 1984
```

5.
```
> tally <- table(macro$country[macro$gdp > 5])
> names(tally)[tally == max(tally)]
[1] "Japan"
```

functions

One last object class that we have yet to mention are **functions**.

Basically everything in R is performing a function on an object.

Recall a function in math:

$$\text{input(s)} \longrightarrow \boxed{\text{function}} \longrightarrow \text{one output}$$

Functions in R:

$$\text{object(s) (arguments)} \longrightarrow \boxed{\text{function}} \longrightarrow \text{one object}$$

Up to now, we've used many canned functions, but we will also need to write our own functions.

$$f(x, y) = x^2 + y^2$$

```
> xy.func <- function(x, y) {
+     f.xy <- x^2 + y^2
+     return(f.xy)
+ }
```

```
> xy.func(x = 3, y = 4)
```

```
[1] 25
```

or

```
> xy.func <- function(vec) {
+     x <- vec[1]
+     y <- vec[2]
+     f.xy <- x^2 + y^2
+     return(f.xy)
+ }
```

```
> xy.func(vec = c(3, 4))
```

```
[1] 25
```

a function that deletes the first *p* percent of observations from a dataframe

```
> trim.func <- function(x, p = .1){
+   n <- nrow(x)  #number of observations
+   trim.number <- round(p*n)  #number to delete (rounded)
+   trimmed.data <- x[-c(1:trim.number),] #delete from top
+   return(trimmed.data)
+ }
```

```
> data(macro)
> trimmed.1 <- trim.func(x = macro)
> trimmed.25 <- trim.func(x = macro, p = 0.25)
```

looking for help

Suppose you want help for a certain function or dataset in R (i.e. what are the arguments of the function, what does the function do, etc.).

To look for help with a function called `foo()`, you can type in one of the following:

- `?foo`
- `help("foo")`

To search for a function by keyword, type in `help.search("keyword")`.

how to read a help file

Most help files follow an approximate format:

- description: briefly describes what the function does
- usage: the syntax for the function as well as arguments and its defaults
- arguments: more specific details about what goes in the arguments
- details: more elaborate description of what the function does
- value: quantities that you can extract from the function (the output)
- more notes and references
- similar functions in R
- example code you can run

exercises

1. Write a function that calculates the geometric mean of a vector of numbers:

$$G = \left(\prod_{i=1}^{n} X_n \right)^{\frac{1}{n}}$$

2. Write a function that takes in a matrix, finds the column of the matrix that has the smallest sum, and then produces a list with the mean, median, standard deviation, and the 25th and 75th percentiles of that column.

solutions

1. ```
   > geomean.func <- function(x){  ## x is a vector
   +   G <- prod(x)^(1/length(x))
   +   return(G)
   + }
   > test.vec <- c(1,2,3)
   > geomean.func(test.vec)
   [1] 1.817121
   ```
   ```

2.
```
> two.func <- function(x){  ## x is a matrix
+   which.col <- which(colSums(x) == min(colSums(x)))
+   column <- x[,which.col] ## extract column
+   output <- list(mean = mean(column), median = median(column),
+   sd=sd(column), percentiles=quantile(column, probs=c(.25,.75)))
+   return(output)
+ }
> random.numbers <- sample(1:100, 81)
> test.mat <- matrix(random.numbers, nrow = 9, ncol = 9)
> two.func(test.mat)

$mean
[1] 43.22222

$median
[1] 39

$sd
[1] 31.29208

$percentiles
25% 75%
 16  68
```

apply()

The `apply()` function takes a function and applies it on each row or column of a matrix, dataframe, or array.

- the MARGIN argument gets 1 for row and 2 for column
- typically, though not necessarily, the function to be applied is a function that takes in vectors
- any extra arguments to the function being applied can be defined after the FUN argument
- `lapply()`, `sapply()`, and `tapply()` are functions that do similar things

take the median of every row

```
> test.matrix <- matrix(1:9, ncol = 3, nrow = 3)
> test.matrix

     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

> medians <- apply(test.matrix, MARGIN = 1, FUN = median)
> medians

[1] 4 5 6
```

take the geometric mean of every column

```
> geomean.func <- function(x) {
+     G <- prod(x)^(1/length(x))
+     return(G)
+ }
> test.matrix <- matrix(11:19, ncol = 3, nrow = 3)
> apply(test.matrix, MARGIN = 2, FUN = geomean.func)

[1] 11.97216 14.97774 17.98146
```

find the 25th and 75th quantile of every column of `macro`

```
> library(Zelig)
> data(macro)
> ## take out the first two columns, which are country and year
> apply(macro[,-c(1,2)], MARGIN=2, FUN=quantile, probs=c(.25,.75))

         gdp     unem capmob    trade
25% 1.877098 2.099248     -1 41.41939
75% 4.700000 7.300000      0 71.84709

> summary(macro[,-c(1,2)])

      gdp              unem             capmob             trade
 Min.   :-4.300   Min.   : 0.6848   Min.   :-4.0000   Min.   :  9.623
 1st Qu.: 1.877   1st Qu.: 2.0992   1st Qu.:-1.0000   1st Qu.: 41.419
 Median : 3.200   Median : 4.5000   Median :-1.0000   Median : 52.624
 Mean   : 3.254   Mean   : 4.9939   Mean   :-0.8914   Mean   : 57.076
 3rd Qu.: 4.700   3rd Qu.: 7.3000   3rd Qu.: 0.0000   3rd Qu.: 71.847
 Max.   :12.800   Max.   :13.0000   Max.   : 0.0000   Max.   :146.020
```

sampling

Sampling from a vector can be done with the `sample()` function.

```
> population <- c(1, 2, 3, 4, 5)
> samp.w.rep <- sample(population, size = 3, replace = T)
> samp.w.rep

[1] 5 2 3
```

```
> samp.wo.rep <- sample(population, size = 3, replace = F)
> samp.wo.rep

[1] 5 3 2
```

```
> samp.w.probs <- sample(population, size = 3, replace = T,
+     prob = c(0.8, 0.05, 0.05, 0.05, 0.05))
> samp.w.probs

[1] 1 1 1
```

for loops

Use a `for` loop to repeat some code over and over again.

- typically good for things like sampling multiple times
- very computationally intensive, so use as last resort
- many things can be done using `apply()` instead

```
> vector <- c(1, 5, 8, 3, 5, 2, 97, 430)
> for (i in vector) {
+     print(i)
+ }

[1] 1
[1] 5
[1] 8
[1] 3
[1] 5
[1] 2
[1] 97
[1] 430
```

- each time runs through everything between `{ }`
- each time, "i" is defined to be an element of the vector (first time "i" is the first element, second time "i" is the second element, etc.)
- loop runs as many times as the length of the vector

a loop that adds 5 to each of the loop indices and stores the output in a vector

```
> add.vec <- c()
> for (j in 1:10){
+   ## add 5 to j and put in jth slot of the add.vec vector
+   add.vec[j] <- j + 5
+ }
> add.vec

 [1]  6  7  8  9 10 11 12 13 14 15
```

conditional statements

Conditional statements may come in handy when manipulating data:

- `if(){}` and `else{}`
- `ifelse()`

$$f(x) = \begin{cases} 8x & \text{if } 0 \leq x < 0.25 \\ \frac{8}{3} - \frac{8}{3}x & \text{if } 0.25 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

```
> triangle.func <- function(x) {
+     if (x >= 0 & x < 0.25) {
+         out <- 8 * x
+     }
+     else if (x >= 0.25 & x <= 1) {
+         out <- 8/3 - 8 * x/3
+     }
+     else {
+         out <- 0
+     }
+     return(out)
+ }
```

Create a new variable in `macro` for whether the observation is before or after 1980.

```
> macro$pre1980 <- ifelse(macro$year < 1980, 1, 0)
```

Everything done using `ifelse()` can be done using `if` and `else`, but the code may be less efficient.

exercises

You will need the `macro` dataset from `Zelig`.

1. Write a function that takes in a vector and outputs the maximum of the vector minus the minimum of the vector.
2. Apply the function you wrote above to columns of `macro`, omitting the country and year variables.

3. Omit the country and year variables from `macro`. Take a sample of 350 observations from the dataset with replacement (hint: sample row numbers). Take the mean of each column in your sample. Do this 1000 times and store your results in a 1000×4 matrix.
4. Take a sample of size 200 from `macro$trade` without replacement. If the absolute value (`abs()`) of the difference between the max and the min of this sample is greater than 3 times the standard deviation of `macro$trade`, then take the median of the sample. Otherwise take the mean. Do this 1000 times and store the results of your 1000 iterations in a vector.
5. Create a new variable in `macro` called "rich" where for each observation, the variable takes on a value of 1 if its gdp is greater than the mean gdp and 0 otherwise.

solutions

1. ```
   > diff.func <- function(x){ ## x is a vector
   +   out <- max(x) - min(x)
   +   return(out)
   + }
   ```

2. ```
   > data(macro)
   > apply(macro[, -c(1, 2)], MARGIN = 2, FUN = diff.func)
   ```

   ```
        gdp      unem    capmob     trade
   17.10000  12.31522   4.00000 136.39729
   ```

3. ```
   > new.macro <- macro[,-c(1,2)]
   > results <- matrix(NA, nrow = 1000, ncol = 4) # results matrix
   > for (i in 1:1000){
   +   row.samp <- sample(c(1:nrow(new.macro)), size = 350,
   +   replace = T)
   +   data.samp <- new.macro[row.samp,]
   +   results[i,] <- colMeans(data.samp)
   + }
   ```
   ```

4. ```
   > results.vec <- c()
   > for (i in 1:1000) {
   +     samp <- sample(macro$trade, size = 200, replace = F)
   +     if (abs(max(samp) - min(samp)) > 3 * sd(macro$trade)) {
   +         results.vec[i] <- median(samp)
   +     }
   +     else {
   +         results.vec[i] <- mean(samp)
   +     }
   + }
   ```

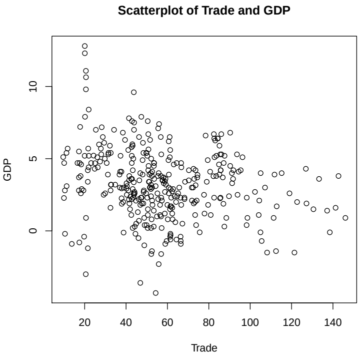
5. ```
   > macro$rich <- ifelse(macro$gdp > mean(macro$gdp), 1, 0)
   ```

plots

- specialized plots: `hist()`, `barplot()`, etc.
- general plot command for lines, points, etc.: `plot()`
- many options using `par()` before `plot()`
- add a legend using `legend()`
- add points with `points()`

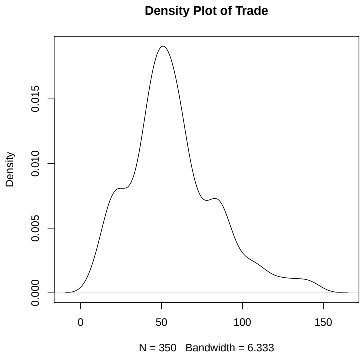
scatterplot:

```
> plot(x = macro$trade, y = macro$gdp, xlab = "Trade", ylab = "GDP",
+     main = "Scatterplot of Trade and GDP")
```



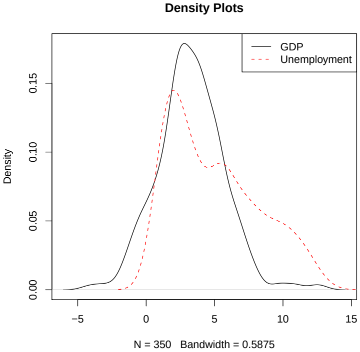
density plot:

```
> plot(density(macro$trade), main = "Density Plot of Trade")
```



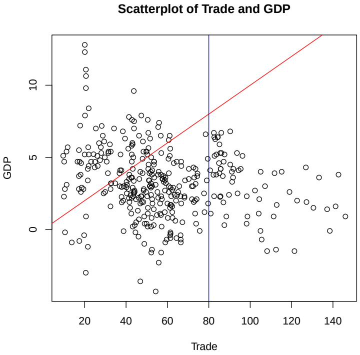
multiple line plots in one (with legend):

```
> plot(density(macro$gdp), main = "Density Plots")
> lines(density(macro$unem), col = "red", lty = "dashed")
> legend(x = "topright", legend = c("GDP", "Unemployment"),
+     lty = c("solid", "dashed"), col = c("black", "red"))
```



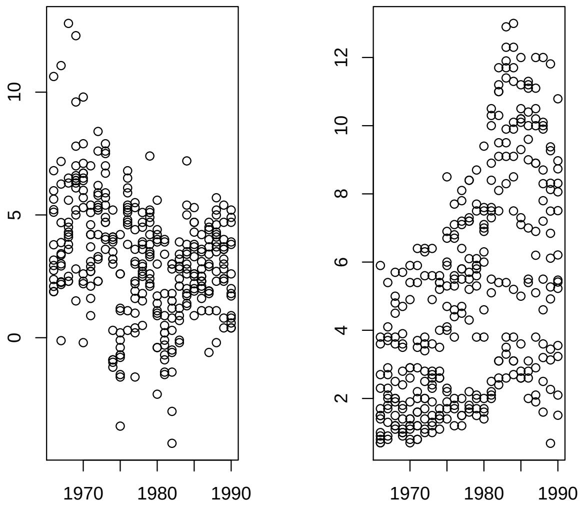
add a straight line with `abline()`:

```
> plot(x = macro$trade, y = macro$gdp, xlab = "Trade", ylab = "GDP",
+     main = "Scatterplot of Trade and GDP")
> abline(a = 0, b = 0.1, col = "red")
> abline(v = 80, col = "blue")
```



multiple plots with `par()` options

```
> par(mfrow = c(1, 2))
> plot(x = macro$year, y = macro$gdp, main = "", xlab = "",
+     ylab = "")
> plot(x = macro$year, y = macro$unem, main = "", xlab = "",
+     ylab = "")
```


0
5
10
1970
1980
1990
2
4
6
8
10
12
1970
1980
1990

saving plots

Two ways of saving plots:

- right-click and save
- using commands
  - before the plot command, use `pdf()`, `png()` etc. with filename
  - after including everything in plot, type `dev.off()`

```
> pdf("filename.pdf")
> plot(x = macro$trade, y = macro$gdp)
> dev.off()
```

```
quartz
     2
```

matrix algebra

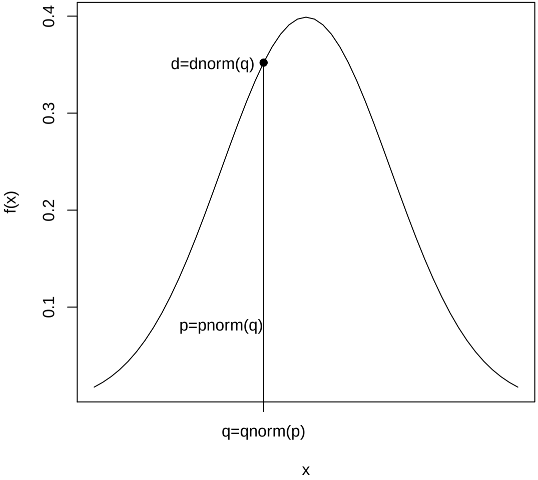
- add/subtract matrices with $+/-$
- matrix multiply with `%*%`
- transpose with `t()`
- invert with `solve()`
- extract diagonal with `diag()`
- determinant with `det()`

probability distributions

For the normal distribution:

- `dnorm()`: density function, gives the height of the density curve
- `pnorm()`: distribution function, gives the area to the left (or right)
- `qnorm()`: quantile function, opposite of `pnorm()`
- `rnorm()`: generate random draws from the distribution

Similar commands for other distributions.



0.4
0.3
0.2
0.1
f(x)
d=dnorm(q)
p=pnorm(q)
q=qnorm(p)
x

final exercises

Write up the following exercises into a LaTeX document. Put any code you use into a `verbatim` environment in the document. All figures and tables should have captions and you should include a very short (1-2 sentence) writeup for everything you report describing what you did.

1. Load the `PErisk` dataset from the `Zelig` package.
2. Which country does the 35th observation belongs to? Use code to identify the country.
3. Create a new dataset that omits Kenya from the dataset.
4. Using the new dataset, now extract the `barb2` and `gdpw2` variables. Find the mean, median, standard deviation, and correlation of these two variables. Present the results in a nicely formatted table in LaTeX.

5. Now let $\mathbf{X}$ be a matrix with two columns: the first column is a column of 1s and the second column is the `gdpw2` variable. Let $\mathbf{y}$ be the `barb2` variable. Create $\mathbf{X}$ and $\mathbf{y}$ and find $\hat{\boldsymbol{\beta}}$, where

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

   Type out this equation in LaTeX and also report the values in $\hat{\boldsymbol{\beta}}$. $\hat{\boldsymbol{\beta}}$ should be a vector of length 2.

6. Create two plots. In the first plot, include the density of the `gdpw2` and `barb2` variables. You should have two curves. Differentiate the curves by color and/or type. In the second plot, do a scatterplot with `gdpw2` on the x-axis and `barb2` on the y-axis. Add a line that has an intercept at the first value of $\hat{\boldsymbol{\beta}}$ and a slope of the second value of $\hat{\boldsymbol{\beta}}$. Include an informative legend for both plots. Put these two plots side-by-side and include them as an image in your document.

7. Write a function that takes in any $n \times k$ matrix for $\mathbf{X}$ and any $n \times 1$ vector for $\mathbf{y}$ and calculates $\hat{\boldsymbol{\beta}}$.
8. Do the following 1000 times with a for loop: Take a sample of 61 observations from the dataset with replacement. For each sample, calculate $\hat{\boldsymbol{\beta}}$, where $\mathbf{X}$ and $\mathbf{y}$ are defined the same as in question 5. Store your results in a 1000×2 matrix.
9. Using the apply function, find and report the mean, standard deviation, and 2.5% and 97.5% quantiles for the two columns in your matrix.
10. For each column, draw 1000 draws from the normal distribution with the means and standard deviations from question 9. That is, you should draw 1000 draws twice, once with the mean and sd from the first column and once with the mean and sd from the second column. For each vector of 1000 draws, plot the density of the draws and include a vertical line for the mean of the draws. You should be doing this twice, once for each set of the 1000 draws. Include the two plots in your document.

1. ```
   > library(Zelig)
   > data(PErisk)
   ```

2. ```
   > PErisk$country[35]
   ```

   ```
   [1] Malaysia
   62 Levels: Argentina Australia Austria Bangladesh Belgium ... Zimba
   ```

3. ```
   > new.data <- PErisk[PErisk$country != "Kenya", ]
   ```

4. ```
   > variables <- cbind(new.data$barb2, new.data$gdpw2)
   > colMeans(variables)
   ```

   ```
   [1] -2.935360  9.065189
   ```

   ```
   > apply(variables, MARGIN = 2, FUN = sd)
   ```

   ```
   [1] 2.7285678 0.9606462
   ```

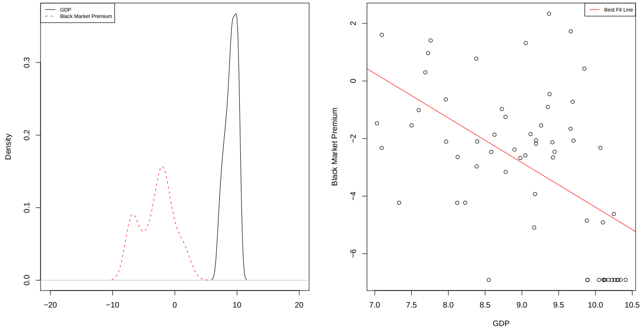
   ```
   > cor(new.data$barb2, new.data$gdpw2)
   ```

   ```
   [1] -0.5456426
   ```

5.
```
> X <- cbind(1, new.data$gdpw2)
> y <- new.data$barb2
> beta.hat <- solve(t(X) %*% X) %*% t(X) %*% y
> beta.hat

          [,1]
[1,] 11.113997
[2,] -1.549814
```

6.
```
> par(mfrow = c(1, 2))
> plot(density(new.data$gdpw2), xlab = "", ylab = "Density",
+     main = "", xlim = c(-20, 20))
> lines(density(new.data$barb2), col = 2, lty = 2)
> legend(x = "topleft", legend = c("GDP", "Black Market Premium"),
+     col = c(1, 2), lty = 1:2, cex = 0.5)
> plot(x = new.data$gdpw2, y = new.data$barb2, main = "",
+     xlab = "GDP", ylab = "Black Market Premium")
> abline(a = beta.hat[1], b = beta.hat[2], col = 2)
> legend(x = "topright", legend = "Best Fit Line", col = 2,
+     lty = 1, cex = 0.5)
```



GDP
Black Market Premium
Density
0.0
0.1
0.2
0.3
−20
−10
0
10
20
Best Fit Line
Black Market Premium
2
0
−2
−4
−6
7.0
7.5
8.0
8.5
9.0
9.5
10.0
10.5
GDP

7.
```
> beta.func <- function(X, y) {
+     beta.hat <- solve(t(X) %*% X) %*% t(X) %*% y
+     return(beta.hat)
+ }
```

8.
```
> results <- matrix(NA, nrow=1000, ncol=2)
> for(i in 1:1000){
+   row.numbers <- sample(1:nrow(new.data), size=61, replace=T)
+   samp.data <- new.data[row.numbers,]  ## sample the data
+   X <- cbind(1, samp.data$gdpw2)
+   y <- samp.data$barb2
+   results[i,] <- beta.func(X=X, y=y)
+ }
```

9.
```
> means <- apply(results, MARGIN = 2, FUN = mean)
> means

[1] 11.157317 -1.552811

> sds <- apply(results, MARGIN = 2, FUN = sd)
> sds

[1] 2.6331276 0.2935984

> quants <- apply(results, MARGIN = 2, FUN = quantile, probs = c(0.
+     0.975))
> quants

           [,1]       [,2]
2.5%   5.911529 -2.0948121
97.5% 16.090756 -0.9442727
```

10.

```
> first.draws <- rnorm(1000, mean = means[1], sd = sds[1])
> second.draws <- rnorm(1000, mean = means[2], sd = sds[2])
> par(mfrow = c(1, 2))
> plot(density(first.draws), xlab = expression(beta[1]), main = "")
> abline(v = mean(first.draws), col = 2)
> plot(density(second.draws), xlab = expression(beta[2]),
+     main = "")
> abline(v = mean(second.draws), col = 2)
```