Introduction to R Programming

Data Science Dojo

datasciencedojo
data science for everyone

Agenda

- R data types
- Basic operations
- Reading and writing data
- Basic statistics
- Basic plotting systems (optional)

What is R?

R is a language and environment for statistical computing and graphics....R provides a wide variety of statistical (linear and nonlinear modeling, classical statistical tests, time-series analysis, classification, clustering, ...) and graphical techniques, and is highly extensible. The S language is often the vehicle of choice for research in statistical methodology, and R provides an Open Source route to participation in that activity.
—*The R Project for Statistical Computing, http://www.r-project.org/*

Open source + highly extensibility + era of data science
→ Over **6400 packages in CRAN** package repository (the official one)

R is vectorized

```
f = 1 * 11 + 2 * 22 + 3 * 33
```

```
> A = c(1, 2, 3)
> B = c(11, 22, 33)

> f = A * B
> f
[1] 11 44 99
```

Good for statistics
Good for data

R for data science

- Advantages
  - Designed for Statistical Analysis
    - Many built-in functions
  - Large number of libraries
  - Mature open source project
- Disadvantages
  - Overhead (Does not scale well to very large data)
- Use R as a “data sandbox” to play with a sample

Hello world

```
c <- “Hello World” # type Enter
print(c) # print some text
# anything after a hash (#) is a comment
```

R DATA TYPES

Atomic Classes

- Character
  - Strings
  - Ex “Survived”
- Numeric
  - Floating point type, default for all numbers
  - Ex. “3.147292”
- Integer
  - Append “L” to specify (1L is an integer, 1 is numeric)
  - Ex. “3”
- Complex
  - Two numeric pieces
  - Ex. “3 + 2i”
- Logical
  - TRUE/FALSE

Compound objects

Vector,
List,
Factor,
Matrix,
Data frame, etc.

Vector

The most basic object is a vector

A vector can only contain objects of the same class

Vector – Creating vectors

Empty vectors can be created with the vector() function.

```
> x <- vector("numeric", length = 10)
> x
[1] 0 0 0 0 0 0 0 0 0 0
```

Vector – Creating vectors

The c() function can be used to create vectors of objects.

```
> x  <-  c(0.5,  0.6) #number
> x  <-  c(TRUE,  FALSE) # logical
> x  <-  c(T,  F) #logical
> x  <-  c("a",  "b",  "c") #character
> x  <-  c(1+0i,  2+4i) #complex
```

Vector – Creating vectors

: operator

```
> x = 1:20
> x
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

The : operator is used to create integer sequences.

List

List is a special type of vector:

1. Can contain elements of different classes (either basic class or compound class)
2. Each element of list can have a name.

```
> x <- list(1, "a", TRUE, 1 + 4i)
> x
> [[1]]
[1] 1
> [[2]]
[1] "a"
> [[3]]
[1] TRUE
> [[4]]
[1] 1+4i
```

Factor

Factors are used to represent categorical data.

Factors can be unordered or ordered. One can think of a factor as an integer vector where each integer has a label. Using factors with labels is better than using integers because factors are self-describing; having a variable that has values "Male" and "Female" is better than a variable that has values 1 and 2.

```
> x <- factor(c("yes", "yes", "no", "yes", "no"))
> x
[1] yes yes no yes no
Levels: no yes
> table(x)
x
 no yes
  2   3
```

Factor - changing the order of levels

The order of the levels can be set using the levels argument to factor().

This can be important in linear modeling because the first level is used as the baseline level.

```
> x <- factor(c("yes", "yes", "no", "yes", "no"))
> x
[1] yes yes no  yes no
Levels: yes no
> x <- factor(x,levels=c("no","yes"))
> x
[1] yes yes no  yes no
Levels: no yes
```

Matrix

Matrix is a 2-dimensional vector.

The dimension is an attribute, an integer vector of length 2 (nrow, ncol).

```
> m <- matrix(nrow = 2, ncol = 3)
> m
     [,1] [,2] [,3]
[1,]   NA   NA   NA
[2,]   NA   NA   NA
```

```
> dim(m)
[1] 2 3
> attributes(m)
$dim
[1] 2 3
```

Matrix - cbind() and rbind()

Matrix can be created by *column-binding* or *row-binding* with cbind() and rbind().

```
> x <- 1:3
> y <- 10:12
> cbind(x,y)
     x  y
[1,] 1 10
[2,] 2 11
[3,] 3 12
```

```
> rbind(x,y)
  [,1] [,2] [,3]
x    1    2    3
y   10   11   12
```

Matrix - naming rows and columns

```
> m <- matrix(1:4, nrow = 2, ncol = 2)
> dimnames(m)
NULL
> dimnames(m) <- list(c("a", "b"), c("c", "d"))
> m
  c d
A 1 3
b 2 4
```

Data frames

Data frames are used to store tabular data.

Unlike matrices, data frames can store different classes of objects in each column; while matrices must have every element be the same class.

Data frames also have a special attribute called row.names

Data frames are usually created by calling read.table() or read.csv()

```
> x <- data.frame(foo = 6:9, bar = c(T, T, F, F))
> x
  foo   bar
1   6  TRUE
2   7  TRUE
3   8 FALSE
4   9 FALSE
> nrow(x)
[1] 4
> ncol(x)
[1] 2
```

Coercion

```
> y  <-  c(1.7,  "a") #character
> y
[1] "1.7" "a"
>  y  <-  c(TRUE,  2) #numeric
> y
[1] 1 2
```

When different objects are mixed in a vector, coercion occurs so that every element in the vector is of the same class.

Coercion - explicit coercion

Objects can be explicitly coerced from one class to another using the as.* functions.

```
> x  <-  0:6
> class(x)
[1] "integer"

> as.numeric(x)
[1] 0 1 2 3 4 5 6
```

```
> as.logical(x)
 [1] FALSE  TRUE  TRUE  TRUE  TRUE
TRUE  TRUE

> as.character(x)
[1] "0" "1" "2" "3" "4" "5" "6"

> as.complex()
complex(0)

> as.complex(x)
 [1] 0+0i 1+0i 2+0i 3+0i 4+0i 5+0i
6+0i
```

Missing values

Missing values are denoted by NA or NaN

NaN: Not a Number
NA: a missing value and has various forms - NA_integer_, NA_character_, etc.

NaN value is also NA but the converse is not true.

```
> x <- c(1, 2, NA, 10, 3)
> is.na(x)
[1] FALSE FALSE  TRUE FALSE FALSE
> is.nan(x)
[1] FALSE FALSE FALSE FALSE FALSE

> x <- c(1, 2, NaN, NA, 4)
> is.na(x)
[1] FALSE FALSE  TRUE  TRUE FALSE
> is.nan(x)
[1] FALSE FALSE  TRUE FALSE FALSE
```

Summary of R Data Types

1. Basic data types
2. Compound objects
3. Coercion
4. Missing values

Exercise

Declare two vectors with 3 elements each. Call these x and y. Now do the following:

Use rbind() and cbind() to create a matrix from these two variables. Observe the difference in structure resulting from cbind() and rbind() functions.
Name the columns of the matrix.
Create a data frame from x and y and give the columns appropriate names.

Exercise

```
> x <- c(1, 2, 3)
> y <- c(4, 5, 6)
> rbind(x, y)
  [,1] [,2] [,3]
x    1    2    3
y    4    5    6
```

```
> cbind(x, y)
     x y
[1,] 1 4
[2,] 2 5
[3,] 3 6
> a = cbind(x, y)
> colnames(a) <- c("x", "y")
```

BASIC OPERATIONS

Subsetting

There are several operators that used to extract subsets of R objects:
[always returns an object of the same class as the original object. Can be used to select more than one element.
[[is used to extract elements of a list or a data frame
Can only be used to extract a single element and the class of the returned object will not necessarily be a list or data frame.
$ is used to extract elements of a list or data frame by name. Semantics are similar to that of [[.

Subsetting

```
> x  <-  c("a",  "b",  "c",  "c",  "d",  "a")
> x[2]
[1] "b"
> x[1:4]
[1] "a" "b" "c" "c"
> x > "a"
[1] FALSE  TRUE  TRUE  TRUE  TRUE FALSE
> x[x>"a"]
[1] "b" "c" "c" "d"
```

```
> x  <-  matrix(1:6, 2,  3)
> x[1,2]
[1] 3
> x[1,] # Entire first row.
[1] 1 3 5
> x[,2] # Entire second column.
[1] 3 4
```

Subsetting

```
> x  <-  c("a",  "b",  "c",  "c",  "d",  "a")
> x[2]
[1] "b"
> x[1:4]
[1] "a" "b" "c" "c"
> x > "a"
[1] FALSE  TRUE  TRUE  TRUE  TRUE FALSE
> x[x>"a"]
[1] "b" "c" "c" "d"
```

```
> x  <-  matrix(1:6, 2,  3)
> x[1,2]
[1] 3
> x[1,] # Entire first row.
[1] 1 3 5
> x[,2] # Entire second column.
[1] 3 4
```

Use in data science: drop useless columns

```
titanic.data <- titanic.data[, -c(1, 4, 9, 11)]
titanic.data <- titanic.data[, titanic.data$PassengerId,
titanic.data$Name, titanic.data$Ticket,
titanic.data$Cabin]
```

Built-in functions

Numeric Functions:

abs(x) absolute value
sqrt(x) square root
ceiling(x) ceiling(3.475) is 4
floor(x) floor(3.475) is 3
trunc(x) trunc(5.99) is 5
round(x, digits=n) round(3.475, digits=2) is 3.48
...

Character Functions:

paste(..., sep="") Concatenate strings after using sep string to seperate them.
paste("x",1:3,sep="") returns c("x1","x2" "x3")
paste("x",1:3,sep="M") returns c("xM1","xM2" "xM3")

grep(pattern, x , ignore.case=FALSE, fixed=FALSE)

Search for pattern in x. If fixed =FALSE then pattern is a regular expression. If fixed=TRUE then pattern is a text string. Returns matching indices.
grep("A", c("b","A","c"), fixed=TRUE) returns 2
...

Built-in functions

Statistical Functions:

rnorm(), dunif(), mean(), sum()...

More complete list:

http://www.statmethods.net/management/functions.html

str function

str compactly displays the internal structure of an R object (data object or function)

```
> str(titanic.data)
'data.frame':   891 obs. of  12 variables:
 $ PassengerId: int  1 2 3 4 5 6 7 8 9 10 ...
 $ Survived   : int  0 1 1 1 0 0 0 0 1 1 ...
 $ Pclass     : int  3 1 3 1 3 3 1 3 3 2 ...
 $ Name       : Factor w/ 891 levels "Abbing, Mr. Anthony",..: 109 191 358 277 16 559 520 629 417
81 ...
 $ Sex        : Factor w/ 2 levels "female","male": 2 1 1 1 2 2 2 2 1 1 ...
 $ Age        : num  22 38 26 35 35 NA 54 2 27 14 ...
 $ SibSp      : int  1 1 0 1 0 0 0 3 0 1 ...
 $ Parch      : int  0 0 0 0 0 0 0 1 2 0 ...
 $ Ticket     : Factor w/ 681 levels "110152","110413",..: 524 597 670 50 473 276 86 396 345 133 .
.
 $ Fare       : num  7.25 71.28 7.92 53.1 8.05 ...
 $ Cabin      : Factor w/ 148 levels "","A10","A14",..: 1 83 1 57 1 1 131 1 1 1 ...
 $ Embarked   : Factor w/ 4 levels "","C","Q","S": 4 2 4 4 4 3 4 4 4 2 ...
```

Packages

For almost everything you want to do with R, there's probably a package written to do just that.
A list of packages in the official packages repository CRAN can be found here: http://cran.fhcrc.org/web/packages/.
If you need a package, it can be installed very easily from within R using the command:

```
install.packages("packagename") # if package already installed, it'll bypass
```

Libraries in Github can be installed using devtools library.

Working directory

You can set the working directory from the menu if using the R-gui (*Change dir…*) or from the R command line:

```
setwd("C:\\MyWorkingDirectory")
setwd("C:/MyWorkingDirectory") # can use forward slash
setwd(choose.dir()) # opens a file browser

getwd()  # returns a string with
         # the current working directory
```

To see a list of the files in the current directory:

```
dir() # returns a list of strings of file names
dir(pattern=".R$") # list of files ending in ".R"
dir("C:\\Users") # show files in directory C:\Users
```

Run a script:

```
source("helloworld.R") # execute a script
```

Summary of Basic Operations

1. Sub-setting
2. Built-in functions
3. Packages
4. Working directory

READING AND WRITING DATA

Reading/writing local flat files

read.table, read.csv, and readLines

CSV stands for 'Comma Separated Values'

```
> titanic_data <- read.csv("Titanic.csv")
> head(titanic_data, 3)
  X Class  Sex   Age Survived Freq
1 1   1st Male Child       No    0
2 2   2nd Male Child       No    0
3 3   3rd Male Child       No   35
```

Reading/writing local flat files

Notice the parameters before reading/writing

Ex.: the read.table function is one of the most commonly used functions for reading data. It has a few important arguments:

file, the name of a file, or a connection
header, logical indicating if the file has a header line
sep, a string indicating how the columns are separated
colClasses, a character vector indicating the class of each column in the dataset
nrows, the number of rows in the dataset
comment.char, a character string indicating the comment character
skip, the number of lines to skip from the beginning
stringsAsFactors, should character variables be coded as factors?

Reading/writing Excel files

read.xlsx, write.xlsx,

or read.xlsx2, write.xlsx2 (faster, but unstable)

```
# Install the xlsx library
> install.packages('xlsx')
# Load the library
> library(xlsx)
# Now you can Read the Excel file
> titanic_data <- read.xlsx("titanic3.xls", sheetIndex=1)
```

Reading XML/HTML files

```
> library(XML) # you need to install this library
> url <- "http://www.w3schools.com/xml/simple.xml"
> doc <- xmlTreeParse(url, useInternal=TRUE) # also works for html
> rootNode <- xmlRoot(doc)
> rootNode[[1]]
<food>
  <name>Belgian Waffles</name>
  <price>$5.95</price>
  <description>Two of our famous Belgian Waffles with plenty of real maple
syrup</description>
  <calories>650</calories>
</food>
```

Reading/writing JSON

JSON: Javascript object notation.
Common format for data from application programming interfaces (APIs). An alternative to XML

```
> library(jsonlite)
> jsonData <- fromJSON("http://citibikenyc.com/stations/json")
> names(jsonData)
[1] "executionTime"    "stationBeanList"
> jsonData$stationBeanList[1,1:3]
  id      stationName availableDocks
1 72 W 52 St & 11 Ave            31
```

Connect to a data base

JSONPackages: RmySQL, RpostresSQL, RODBC, RMONGO

```
> library(RMySQL)# load the library
> ucscDb <- dbConnect(MySQL(), user="genome", host="genome-mysql.cse.ucsc.edu")
> data <- dbGetQuery(ucscDb, "show databases;") # get the output of SQL query as data frame in R
> head(data)
          Database
1 information_schema
2            ailMel1
3            allMis1
4            anoCar1
5            anoCar2
6            anoGam1
> dbDisconnect(ucscDb) # don't forget to close the connection
```

BASIC STATISTICS

Basic Statistics

Summary(titanic.data)

- Mean
- Quartiles
- Maximum
- Number of NAs

```
 PassengerId       Survived          Pclass                                          Name
Min.   :  1.0   Min.   :0.0000   Min.   :1.000   Abbing, Mr. Anthony                    :  1
1st Qu.:223.5   1st Qu.:0.0000   1st Qu.:2.000   Abbott, Mr. Rossmore Edward            :  1
Median :446.0   Median :0.0000   Median :3.000   Abbott, Mrs. Stanton (Rosa Hunt)       :  1
Mean   :446.0   Mean   :0.3838   Mean   :2.309   Abelson, Mr. Samuel                    :  1
3rd Qu.:668.5   3rd Qu.:1.0000   3rd Qu.:3.000   Abelson, Mrs. Samuel (Hannah Wizosky):  1
Max.   :891.0   Max.   :1.0000   Max.   :3.000   Adahl, Mr. Mauritz Nils Martin         :  1
                                                 (Other)                                :885
    Sex           Age              SibSp           Parch             Ticket         Fare
female:314   Min.   : 0.42   Min.   :0.000   Min.   :0.0000   1601     :  7   Min.   :  0.00
male  :577   1st Qu.:20.12   1st Qu.:0.000   1st Qu.:0.0000   347082   :  7   1st Qu.:  7.91
             Median :28.00   Median :0.000   Median :0.0000   CA. 2343:  7   Median : 14.45
             Mean   :29.70   Mean   :0.523   Mean   :0.3816   3101295  :  6   Mean   : 32.20
             3rd Qu.:38.00   3rd Qu.:1.000   3rd Qu.:0.0000   347088   :  6   3rd Qu.: 31.00
             Max.   :80.00   Max.   :8.000   Max.   :6.0000   CA 2144  :  6   Max.   :512.33
             NA's   :177                                      (Other)  :852
```

BASIC PLOTTING SYSTEMS

Basic plotting systems

1. Base R graphics: Simple and allows plotting to mirror the thought process.
2. Lattice graphics: Entire plots created in a simple function call.
3. ggplot2 graphics: an implementation of the Grammar of Graphics by Leland Wikinson. Combines concepts from both base and lattice graphics.
4. Other graphics packages: wait for the bootcamp!

A list of interactive visualization in R at:
http://ouzor.github.io/blog/2014/11/21/interactive-visualizations.html

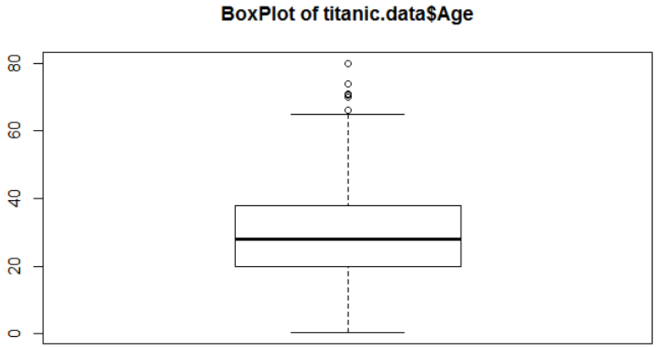
Base R plotting systems

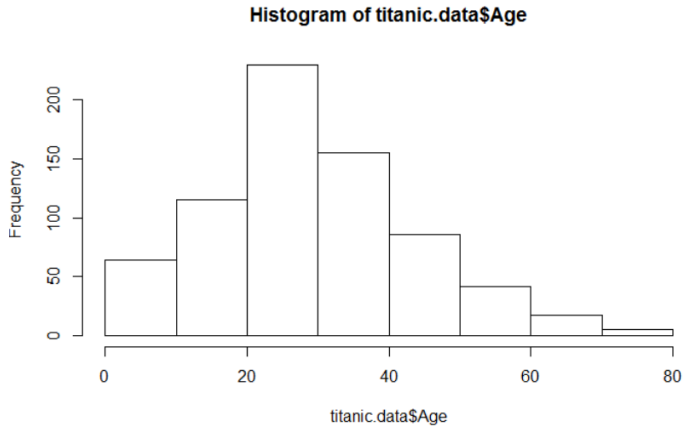
Plotting functions

plot(x, y, …) –> main function for creating scatterplots

boxplot() –> box and whisker plots

hist() –> histograms



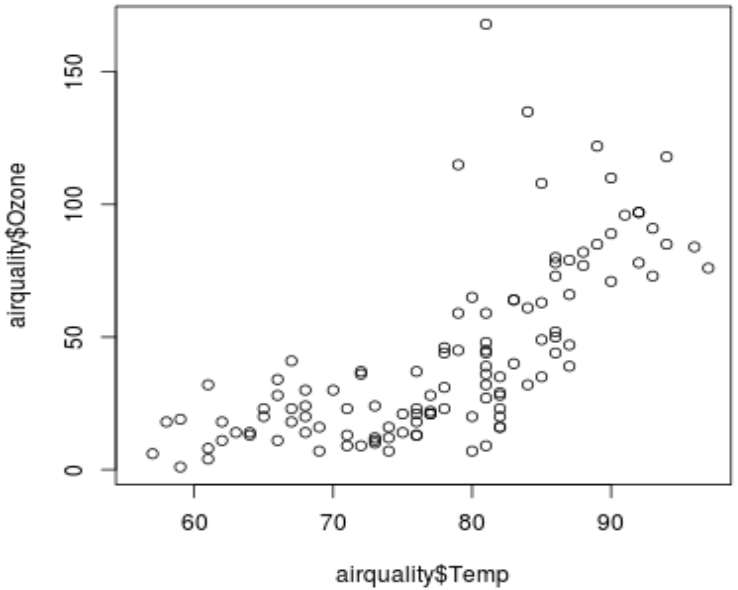


datasciencedojo
data science for everyone

Base R plotting systems

```
> library(datasets)
> names(airquality)
[1] "Ozone"   "Solar.R" "Wind"
"Temp"    "Month"   "Day"
```

> plot(x = airquality$Temp, y = airquality$Ozone)



Basic plotting systems

Additional functions

lines: adds lines to a plot, given a vector of x values and corresponding vector of y values

points: adds a point to the plot

text: add text labels to a plot using specified x,y coordinates

title: add annotations to x,y axis labels, title, subtitles, outer margin

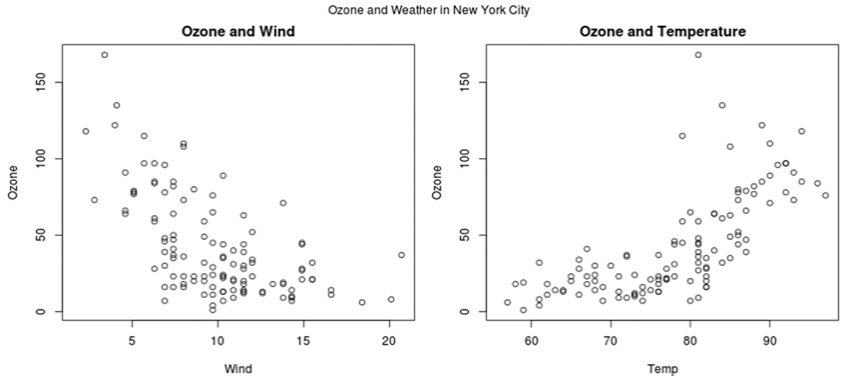
mtext: add arbitrary text to margins (inner or outer) of plot

axis: specify axis ticks

Basic plotting systems

Add more capability with more commands

```
> with(airquality, {
+ plot(Wind, Ozone, main="Ozone and Wind")
+ plot(Temp, Ozone, main="Ozone and Temperature")
+ mtext("Ozone and Weather in New York City", outer=TRUE)
+ })
```



Lattice plotting systems

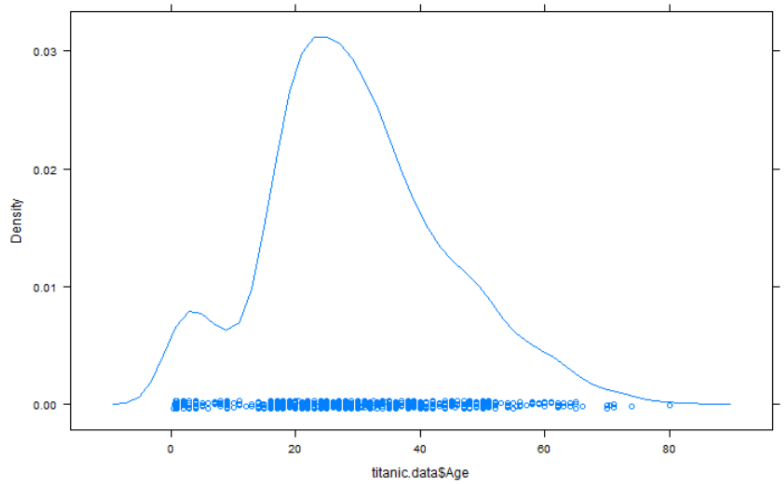
Plotting functions

xyplot(x, data…) –> main function for creating scatterplots

bwplot() –> box and whiskers plots (box plots)

histogram() –> histograms

densityplot() -> shows concentration of numbers



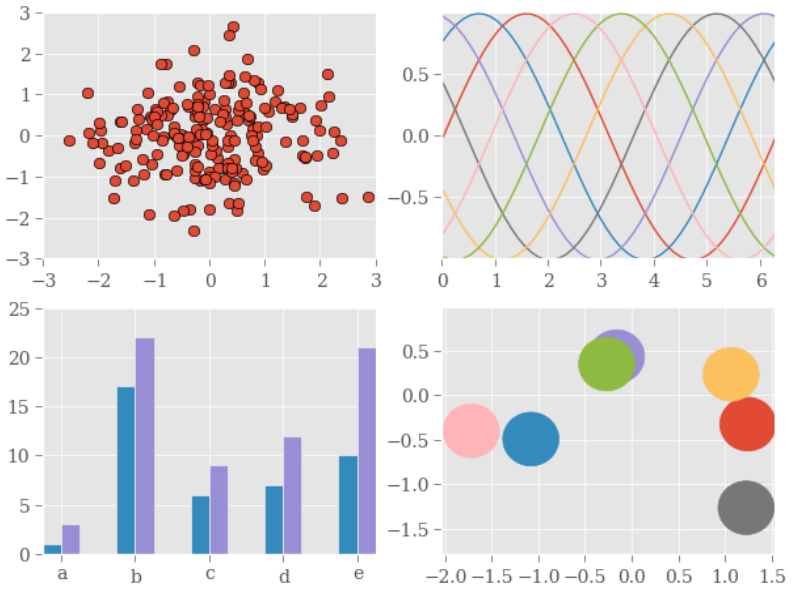
datasciencedojo
data science for everyone

ggplot2 plotting systems

Need to install ggplot2 library

Mix elements of base and lattice

Good tutorial (3 basic plotting systems): https://sux13.github.io/DataScienceSpCourseNotes/4_EXDATA/Exploratory_Data_Analysis_Course_Notes.html



datasciencedojo
data science for everyone

Reference

https://cran.r-project.org/doc/manuals/R-intro.pdf