CT5102: Programming for Data Analytics

Lecture 1: Introduction to R and Atomic Vectors

Dr. Jim Duggan,

School of Engineering & Informatics

National University of Ireland Galway.

https://github.com/JimDuggan/PDAR

https://twitter.com/_jimduggan

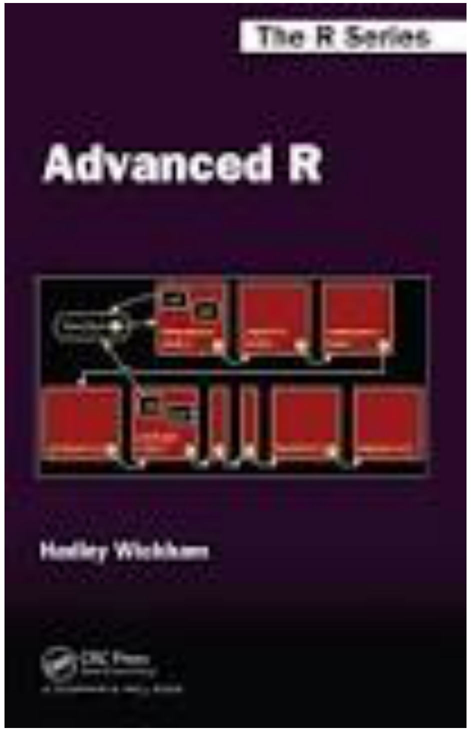
Course Overview

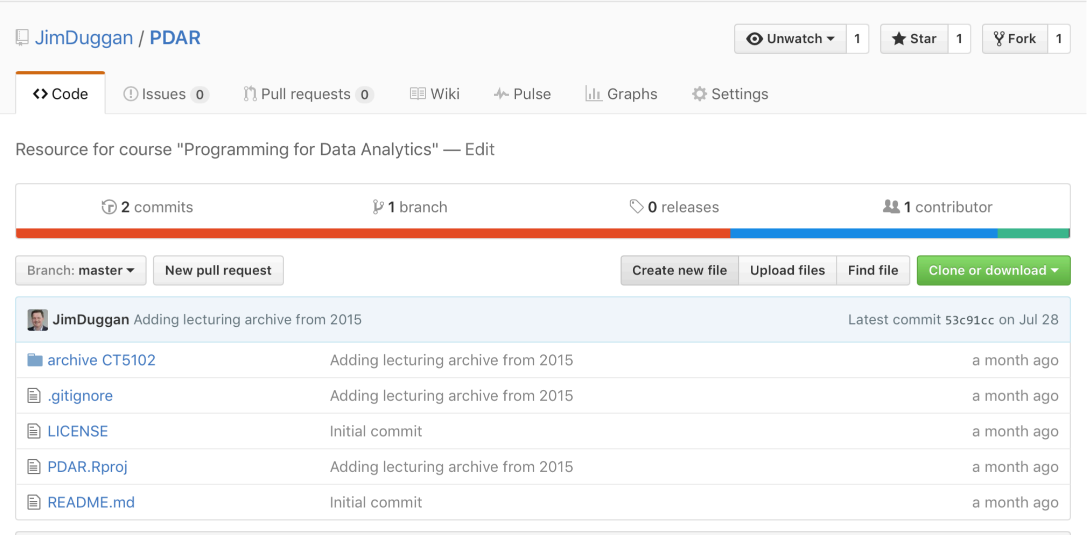
- Data Analytics with R
  - Weeks 1-8
  - Dr. Jim Duggan
- Data Analytics with Python
  - Weeks 9-12
  - Dr. Michael Madden

- Coursework (R)
  - MCQ after each lecture
  - Weekly Assignment

Main Text Book & Resources

http://adv-r.had.co.nz





JimDuggan / PDAR

Resource for course "Programming for Data Analytics" — Edit

2 commits · 1 branch · 0 releases · 1 contributor

Branch: master · New pull request · Create new file · Upload files · Find file · Clone or download

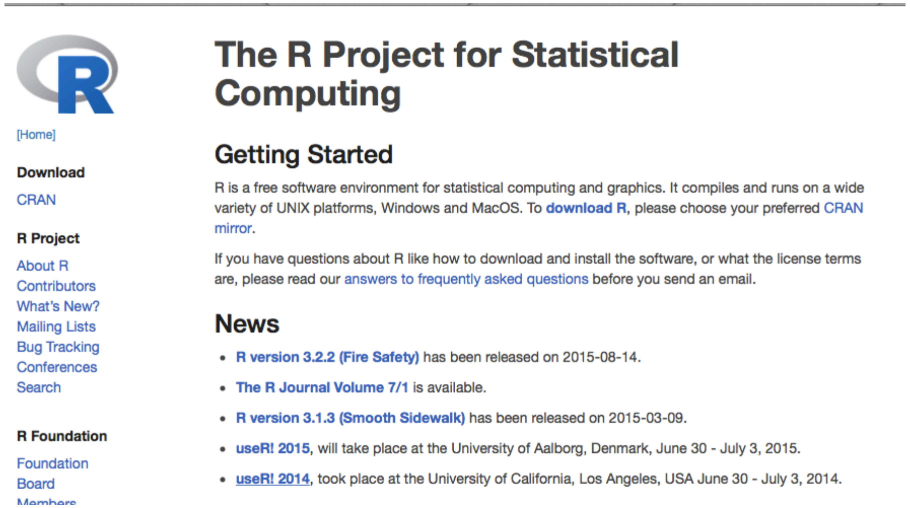
JimDuggan Adding lecturing archive from 2015 · Latest commit 53c91cc on Jul 28

archive CT5102	Adding lecturing archive from 2015	a month ago
.gitignore	Adding lecturing archive from 2015	a month ago
LICENSE	Initial commit	a month ago
PDAR.Rproj	Adding lecturing archive from 2015	a month ago
README.md	Initial commit	a month ago

https://github.com/JimDuggan/PDAR

Overview

- R's *mission* is to enable the best and most thorough exploration of data possible (Chambers 2008).
- It is a dialect of the S language, developed at Bell Laboratories
- ACM noted that *S "will forever alter the way people analyze, visualize, and manipulate data"*



The R Project for Statistical Computing

[Home]

Download

CRAN

R Project

About R
Contributors
What's New?
Mailing Lists
Bug Tracking
Conferences
Search

R Foundation

Foundation
Board
Members

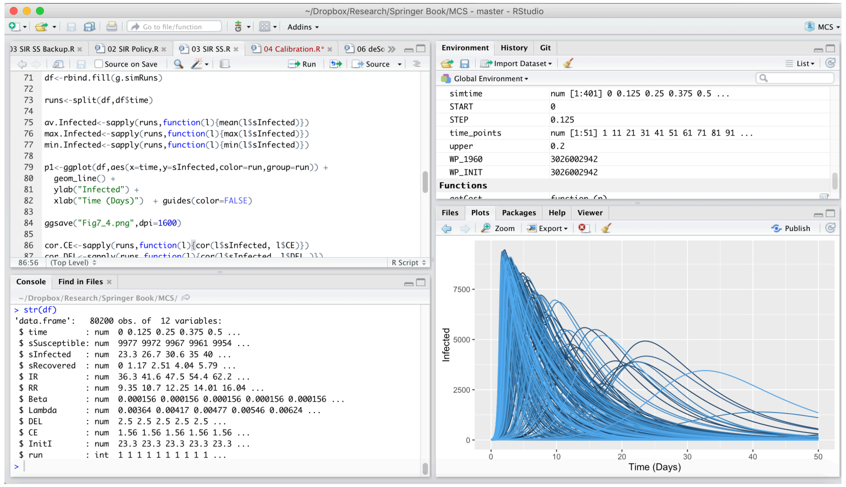
Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To **download R**, please choose your preferred CRAN mirror.

If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

News

- **R version 3.2.2 (Fire Safety)** has been released on 2015-08-14.
- **The R Journal Volume 7/1** is available.
- **R version 3.1.3 (Smooth Sidewalk)** has been released on 2015-03-09.
- **useR! 2015**, will take place at the University of Aalborg, Denmark, June 30 - July 3, 2015.
- **useR! 2014**, took place at the University of California, Los Angeles, USA June 30 - July 3, 2014.



~/Dropbox/Research/Springer Book/MCS - master - RStudio

```
df<-rbind.fill(g.simRuns)

runs<-split(df,df$time)

av.Infected<-sapply(runs,function(l){mean(l$sInfected)})
max.Infected<-sapply(runs,function(l){max(l$sInfected)})
min.Infected<-sapply(runs,function(l){min(l$sInfected)})

p1<-ggplot(df,aes(x=time,y=sInfected,color=run,group=run)) +
  geom_line() +
  ylab("Infected") +
  xlab("Time (Days)")  + guides(color=FALSE)

ggsave("Fig7_4.png",dpi=1600)

cor.CE<-sapply(runs,function(l){cor(l$sInfected, l$CE)})
cor.DEL<-sapply(runs,function(l){cor(l$sInfected, l$DEL )})
```

```
simtime      num [1:401] 0 0.125 0.25 0.375 0.5 ...
START        0
STEP         0.125
time_points  num [1:51] 1 11 21 31 41 51 61 71 81 91 ...
upper        0.2
WP_1960      3026002942
WP_INIT      3026002942
Functions
getCost      function (n)
```

```
> str(df)
'data.frame':   80200 obs. of  12 variables:
 $ time       : num  0 0.125 0.25 0.375 0.5 ...
 $ sSusceptible: num  9977 9972 9967 9961 9954 ...
 $ sInfected  : num  23.3 26.7 30.6 35 40 ...
 $ sRecovered : num  0 1.17 2.51 4.04 5.79 ...
 $ IR         : num  36.3 41.6 47.5 54.4 62.2 ...
 $ RR         : num  9.35 10.7 12.25 14.01 16.04 ...
 $ Beta       : num  0.000156 0.000156 0.000156 0.000156 0.000156 ...
 $ Lambda     : num  0.00364 0.00417 0.00477 0.00546 0.00624 ...
 $ DEL        : num  2.5 2.5 2.5 2.5 2.5 ...
 $ CE         : num  1.56 1.56 1.56 1.56 1.56 ...
 $ InitI      : num  23.3 23.3 23.3 23.3 23.3 ...
 $ run        : int  1 1 1 1 1 1 1 1 1 1 ...
>
```

https://www.safaribooksonline.com/library/view/hands-on-programming-with/9781449359089/pr02.html

Programming in R (Grolemund 2014)

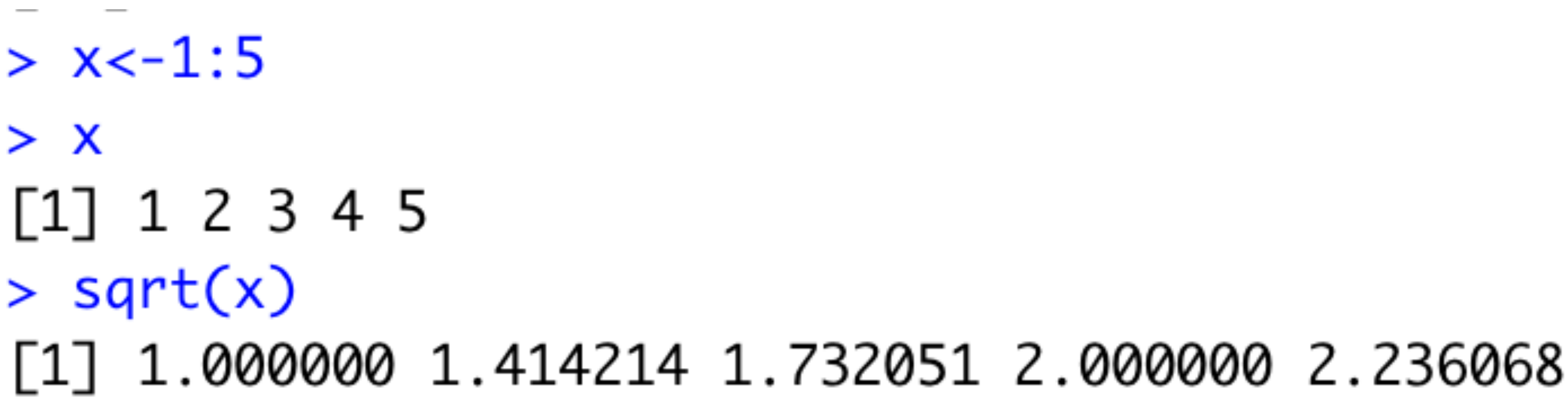
- **However, learning to program *should* be on every data scientist's to-do list**. Knowing how to program will make you a more flexible analyst and augment your mastery of data science in every way. My favorite metaphor for describing this was introduced by Greg Snow on the R help mailing list in May 2006. Using the functions in R is like riding a bus. Writing programs in R is like driving a car.
- *Busses are very easy to use, you just need to know which bus to get on, where to get on, and where to get off (and you need to pay your fare). Cars, on the other hand, require much more work: you need to have some type of map or directions (even if the map is in your head), you need to put gas in every now and then, you need to know the rules of the road (have some type of drivers license). The big advantage of the car is that it can take you a bunch of places that the bus does not go and it is quicker for some trips that would require transferring between busses.*
- *Using this analogy, programs like SPSS are busses, easy to use for the standard things, but very frustrating if you want to do something that is not already preprogrammed.*
- ***R is a 4-wheel drive SUV (though environmentally friendly) with a bike on the back, a kayak on top, good walking and running shoes in the passenger seat, and mountain climbing and spelunking gear in the back.***
- *R can take you anywhere you want to go if you take time to learn how to use the equipment, but that is going to take longer than learning where the bus stops are in SPSS.*

Understanding Computations in R

“Everything that exists is an object.

Everything that happens is a function call.”

John Chambers

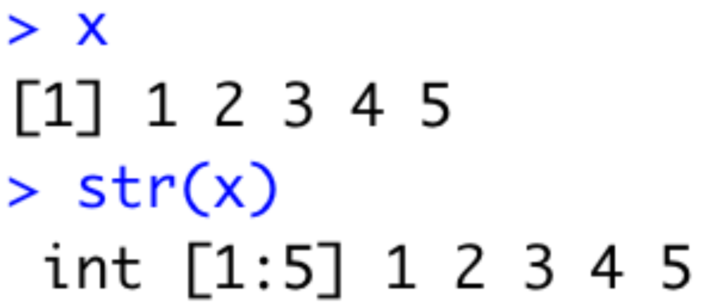


```
> x<-1:5
> x
[1] 1 2 3 4 5
> sqrt(x)
[1] 1.000000 1.414214 1.732051 2.000000 2.236068
```

R – Data Types

	Homogenous	Heterogenous
1d	Atomic Vector	List
2d	Matrix	Data Frame
nd	Array	

- To understand a data structure, use the str() function



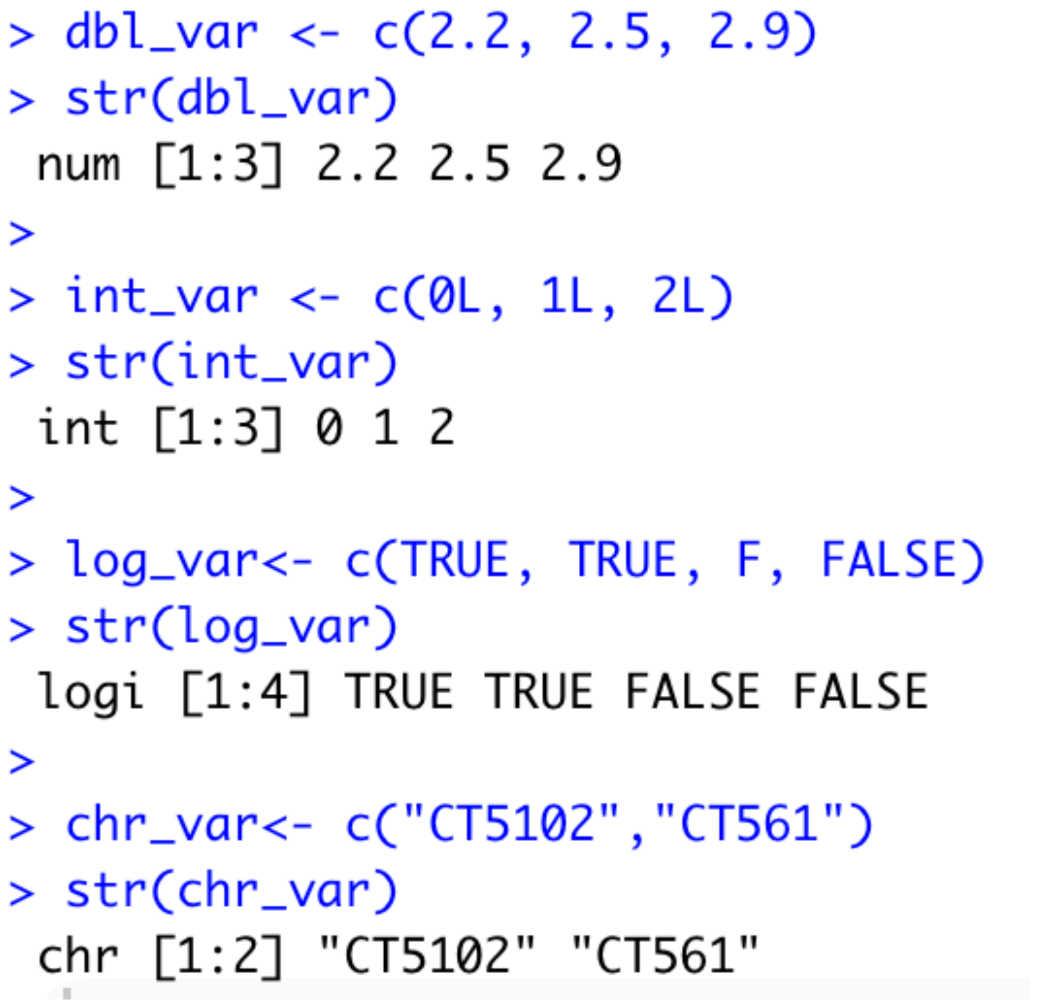
```
> x
[1] 1 2 3 4 5
> str(x)
 int [1:5] 1 2 3 4 5
```

Vectors

- The basic data structure in R is the Vector
- Vectors come in two flavours
  - Atomic vectors
  - Lists
- They have 3 common properties
  - Type, typeof(), what it is
  - Length, length(), how many elements
  - *Attributes, attributes, additional metadata*

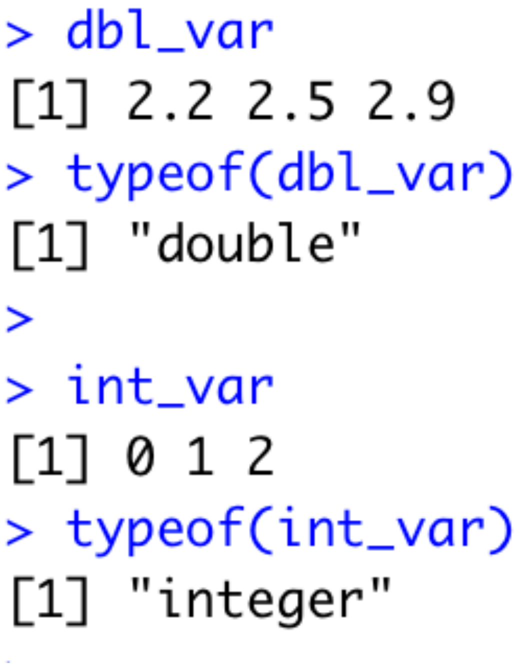
Atomic Vectors

- A sequence of elements that have the same data type (Matloff 2009)
- Four common types
  - logical
  - integer
  - double (or numeric)
  - character
- Usually created with c() – short for combine

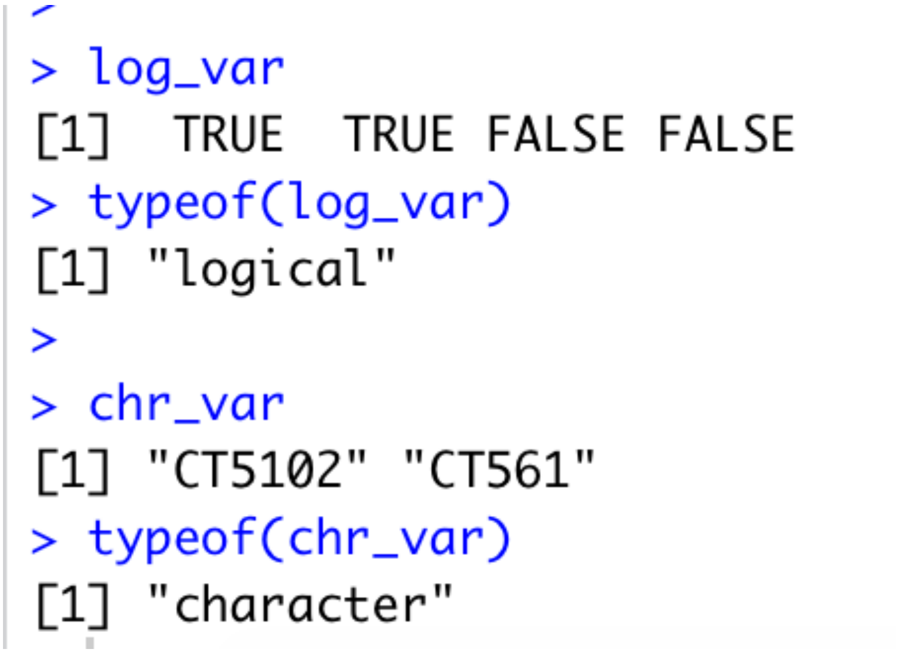


```
> dbl_var <- c(2.2, 2.5, 2.9)
> str(dbl_var)
 num [1:3] 2.2 2.5 2.9
>
> int_var <- c(0L, 1L, 2L)
> str(int_var)
 int [1:3] 0 1 2
>
> log_var<- c(TRUE, TRUE, F, FALSE)
> str(log_var)
 logi [1:4] TRUE TRUE FALSE FALSE
>
> chr_var<- c("CT5102","CT561")
> str(chr_var)
 chr [1:2] "CT5102" "CT561"
```

Atomic vector types



```
> dbl_var
[1] 2.2 2.5 2.9
> typeof(dbl_var)
[1] "double"
>
> int_var
[1] 0 1 2
> typeof(int_var)
[1] "integer"
```



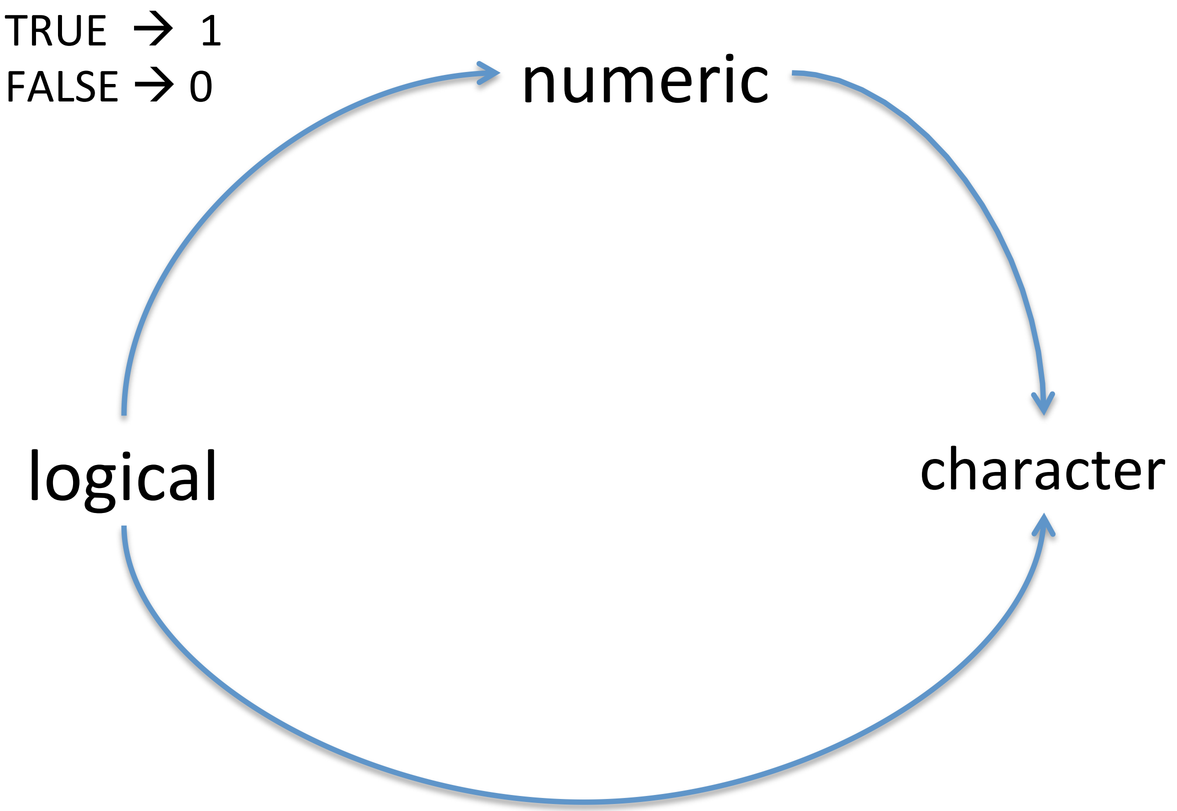
```
> log_var
[1]  TRUE  TRUE FALSE FALSE
> typeof(log_var)
[1] "logical"
>
> chr_var
[1] "CT5102" "CT561"
> typeof(chr_var)
[1] "character"
```

Coercion of atomic vectors

- All elements of an atomic vector MUST be of the same type
- When different type are combined, they will be coerced into the most flexible types
- What will be the type and values of
  - c(1L, T, F)
  - c(1,T,F)

Coercion Rules
Least to most flexible

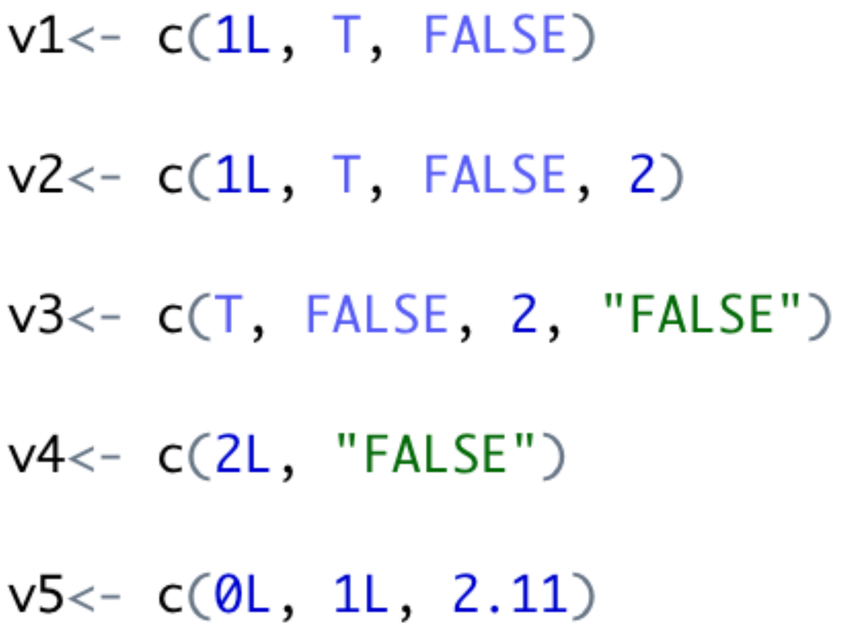
- logical
- integer
- double
- character



Grolemund (2014) p 52

Challenge 1.1

- Determine the types for each of the following (coerced) vectors



```
v1<- c(1L, T, FALSE)

v2<- c(1L, T, FALSE, 2)

v3<- c(T, FALSE, 2, "FALSE")

v4<- c(2L, "FALSE")

v5<- c(0L, 1L, 2.11)
```

Creating atomic vectors using sequences

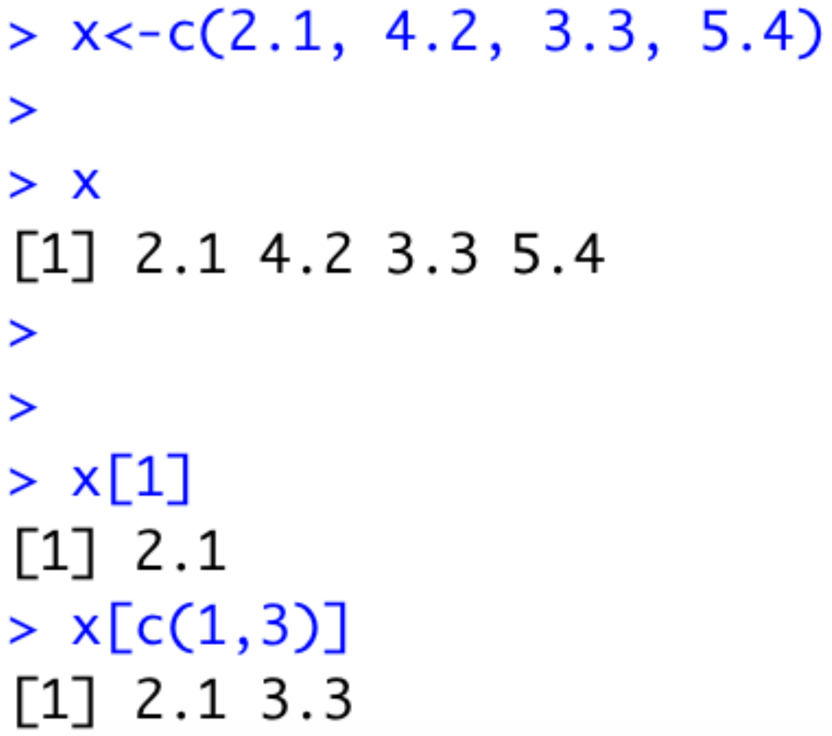
The colon operator (:) generates regular sequences (atomic vectors) within a specified range.

```
> v1<-1:10
> v1
 [1]  1  2  3  4  5  6  7  8  9 10

> v2<-3:13
> v2
 [1]  3  4  5  6  7  8  9 10 11 12 13
```

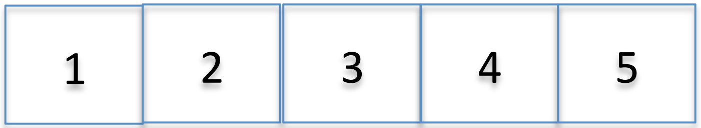
Subsetting Atomic Vectors

- R's subsetting operators are powerful and fast
- For atomic vectors, the operator [is used
- There are six ways to subset an atomic vector in R
- In R, the index for a vector starts at 1



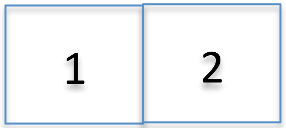
```
> x<-c(2.1, 4.2, 3.3, 5.4)
>
> x
[1] 2.1 4.2 3.3 5.4
>
>
> x[1]
[1] 2.1
> x[c(1,3)]
[1] 2.1 3.3
```

(1) Positive integers

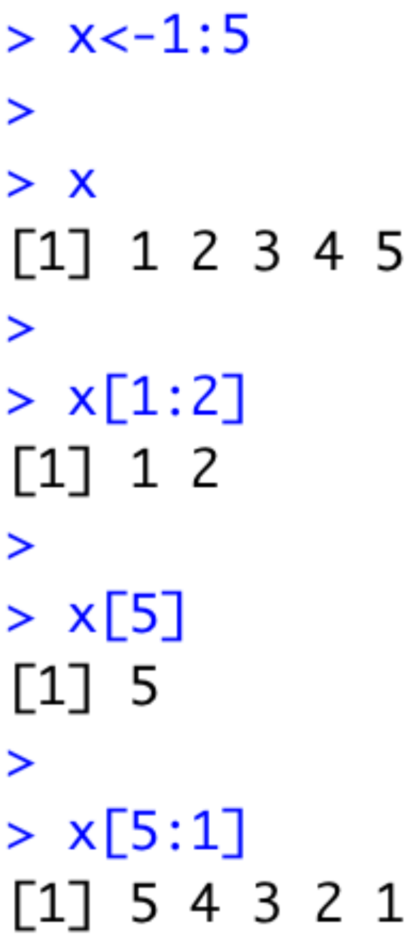


1 2 3 4 5

Positive integers return elements at the specified position



1 2



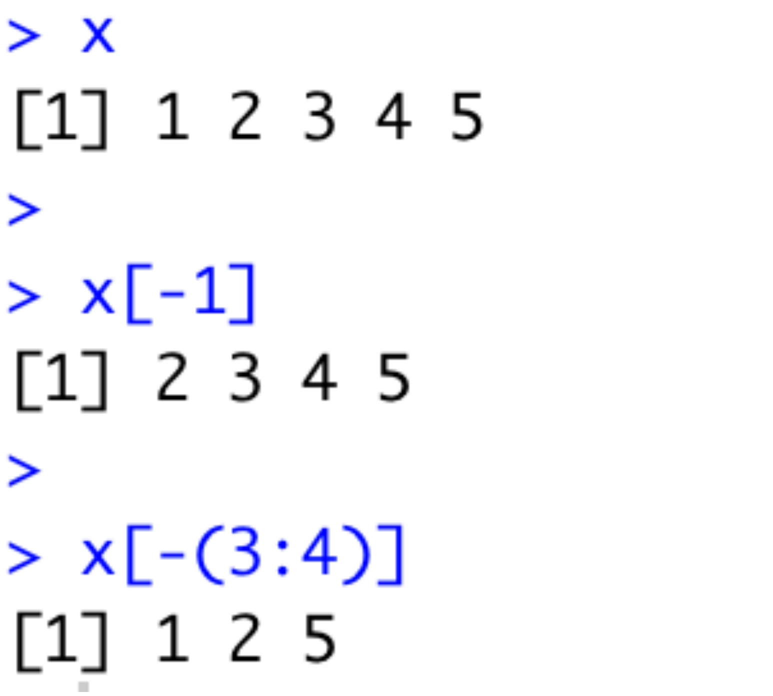
```
> x<-1:5
>
> x
[1] 1 2 3 4 5
>
> x[1:2]
[1] 1 2
>
> x[5]
[1] 5
>
> x[5:1]
[1] 5 4 3 2 1
```

(2) Negative integers

1	2	3	4	5

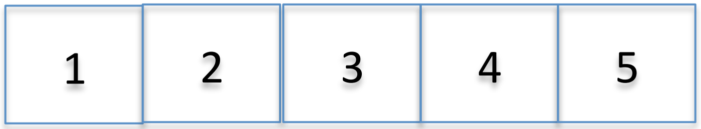
Negative integers omit elements at specified positions

2	3	4	5



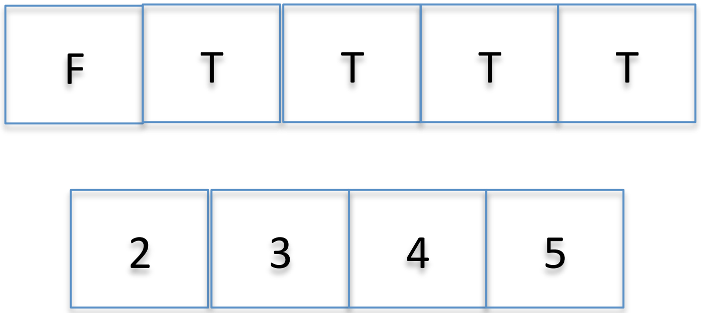
```
> x
[1] 1 2 3 4 5
>
> x[-1]
[1] 2 3 4 5
>
> x[-(3:4)]
[1] 1 2 5
```

(3) Logical Vectors



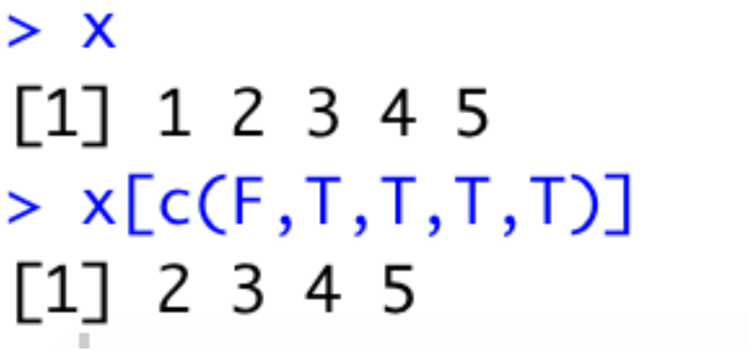
1	2	3	4	5

Select elements where the corresponding logical value is TRUE. This approach supports ***recycling***

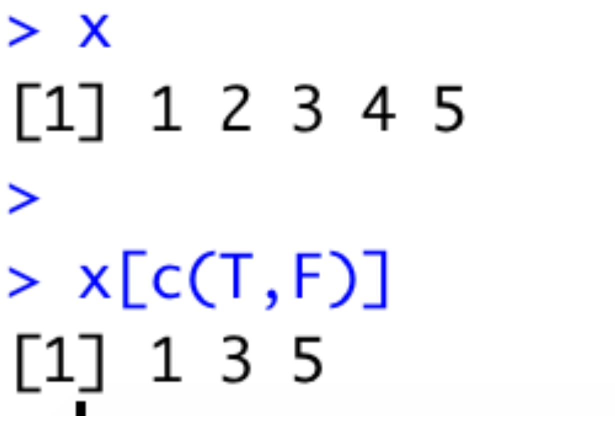


F	T	T	T	T

2	3	4	5



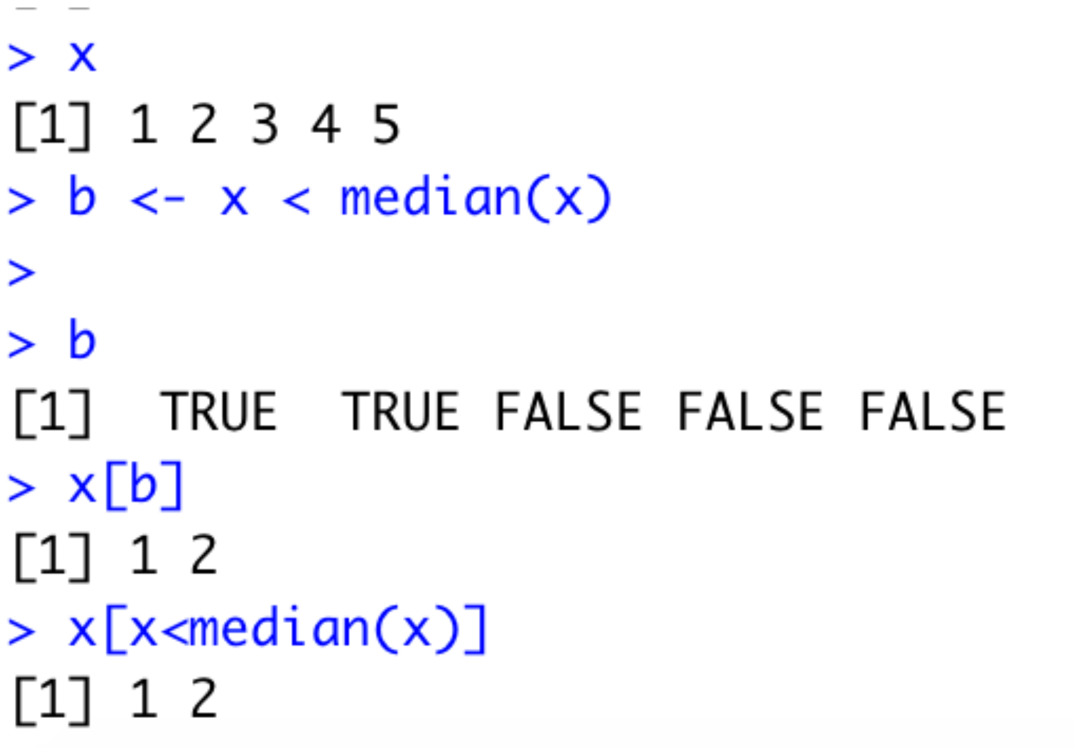
```
> x
[1] 1 2 3 4 5
> x[c(F,T,T,T,T)]
[1] 2 3 4 5
```



```
> x
[1] 1 2 3 4 5
>
> x[c(T,F)]
[1] 1 3 5
```

Logical Vectors - Advantages

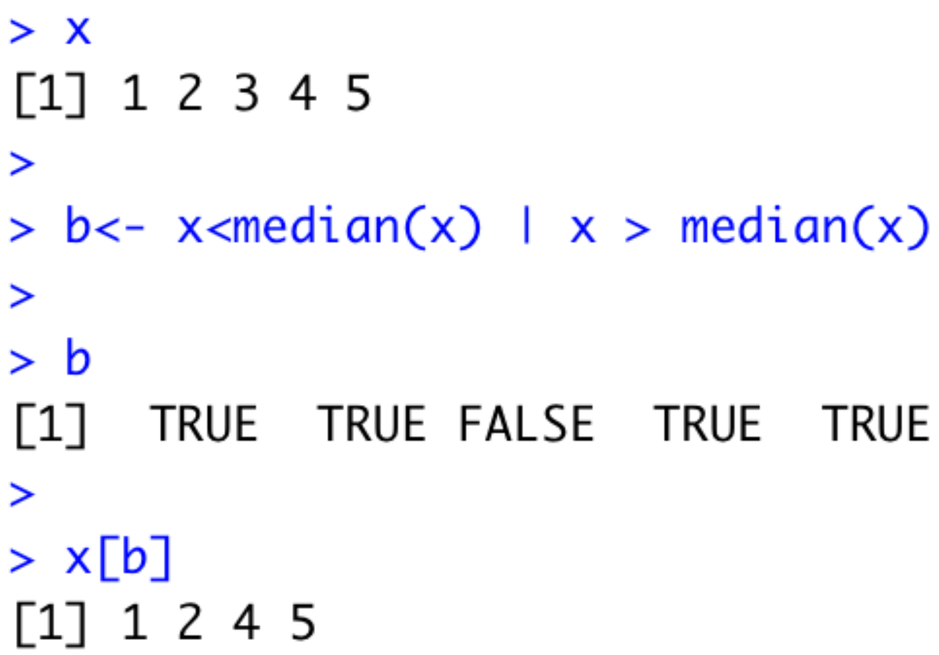
Expressions can be used to create a logical vector



```
> x
[1] 1 2 3 4 5
> b <- x < median(x)
>
> b
[1]  TRUE  TRUE FALSE FALSE FALSE
> x[b]
[1] 1 2
> x[x<median(x)]
[1] 1 2
```

Logical Expressions in R

Operators	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	not x
x | y	x OR y
x & y	x AND y

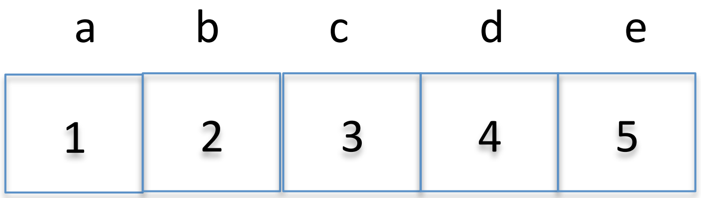


```
> x
[1] 1 2 3 4 5
>
> b<- x<median(x) | x > median(x)
>
> b
[1]  TRUE  TRUE FALSE  TRUE  TRUE
>
> x[b]
[1] 1 2 4 5
```

Challenge 1.2

- Create an R vector of squares of 1 to 10
- Find the minimum
- Find the maximum
- Find the average
- Subset all those values greater than the average

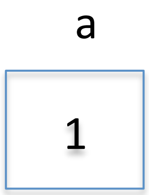
(4) Character vectors



a b c d e

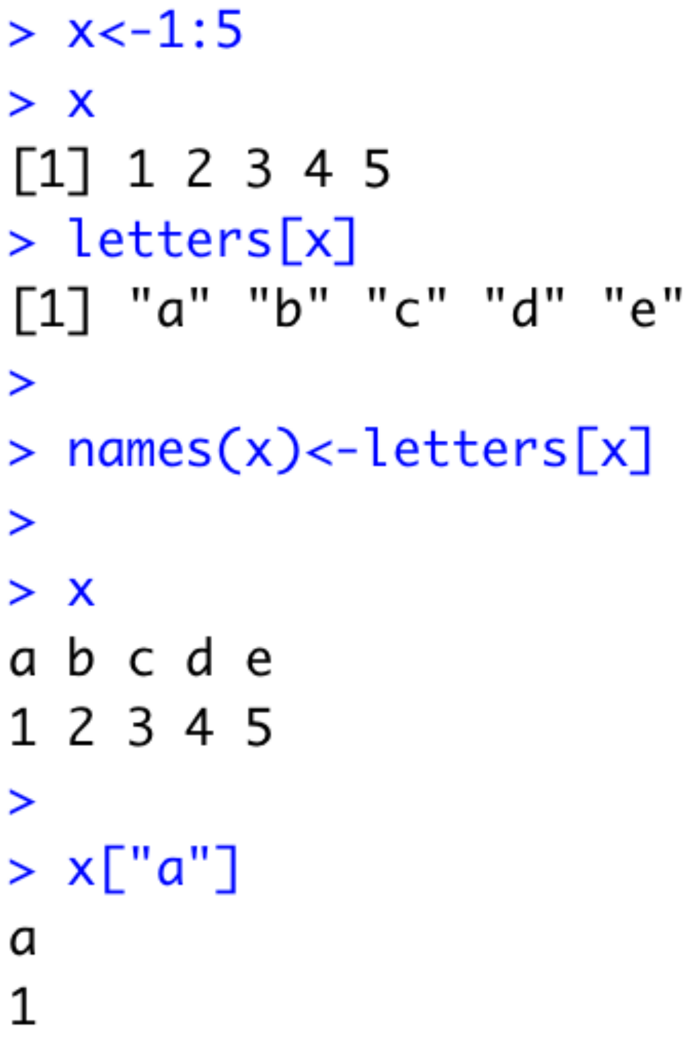
1 2 3 4 5

Return elements with matching names



a

1



```
> x<-1:5
> x
[1] 1 2 3 4 5
> letters[x]
[1] "a" "b" "c" "d" "e"
>
> names(x)<-letters[x]
>
> x
a b c d e
1 2 3 4 5
>
> x["a"]
a
1
```

Vectorization

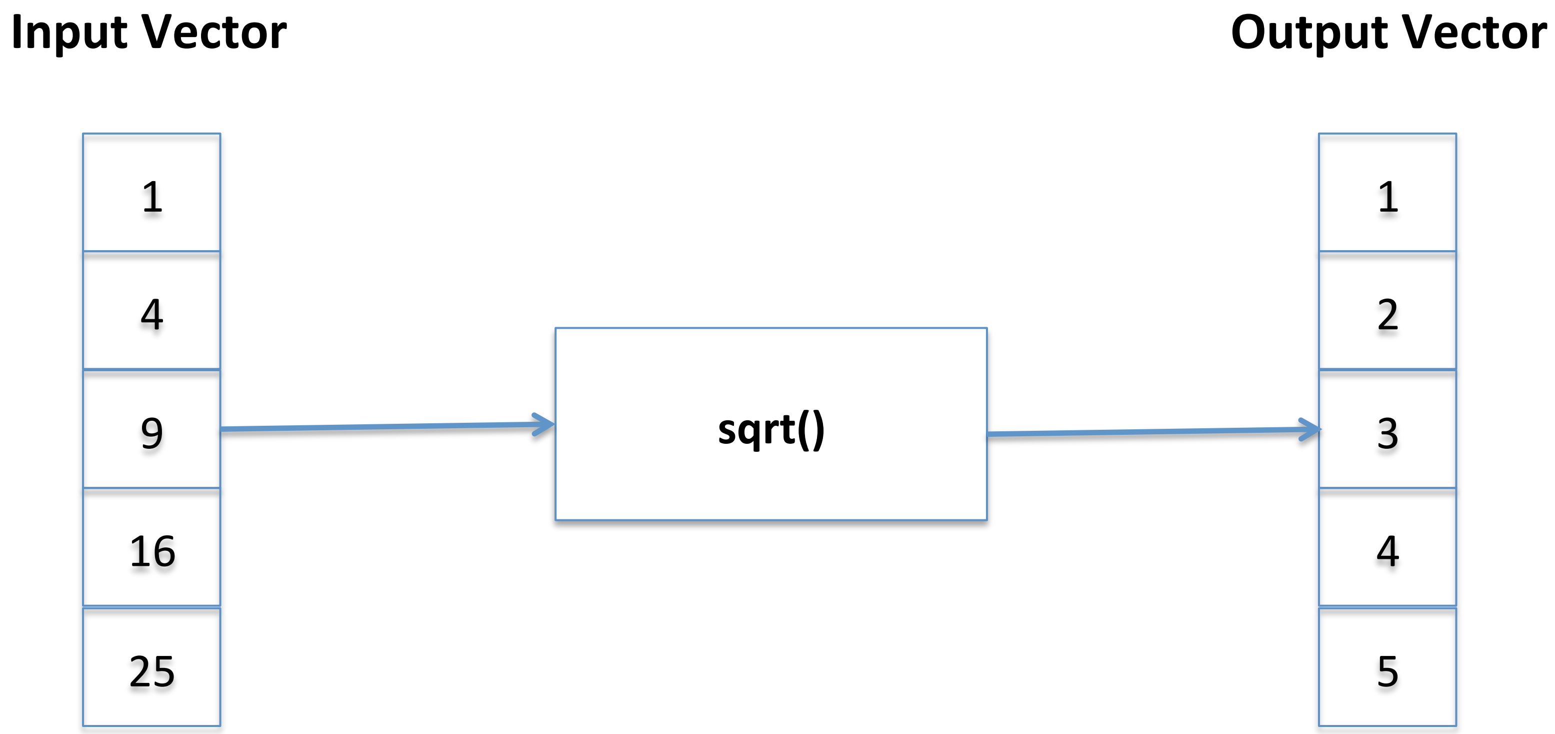
- A powerful feature of R is that it supports *vectorization*
- Functions can operate on every element of a vector, and return the results of each individual operation in a new vector.

```
> v1
[1] 1 2 3 4 5

> r<-sqrt(v1)

> r
[1] 1.000000 1.414214 1.732051 2.000000 2.236068
```

Key Idea



Arithmetic Operators

- Arithmetic operations can also be applied to vectors in an element-wise manner

```
> v1
[1] 1 2 3 4 5
> v2<-3*v1

> v2
[1]  3  6  9 12 15

> v3<-v1+v2
> v3
[1]  4  8 12 16 20
```

Vectorized if/else

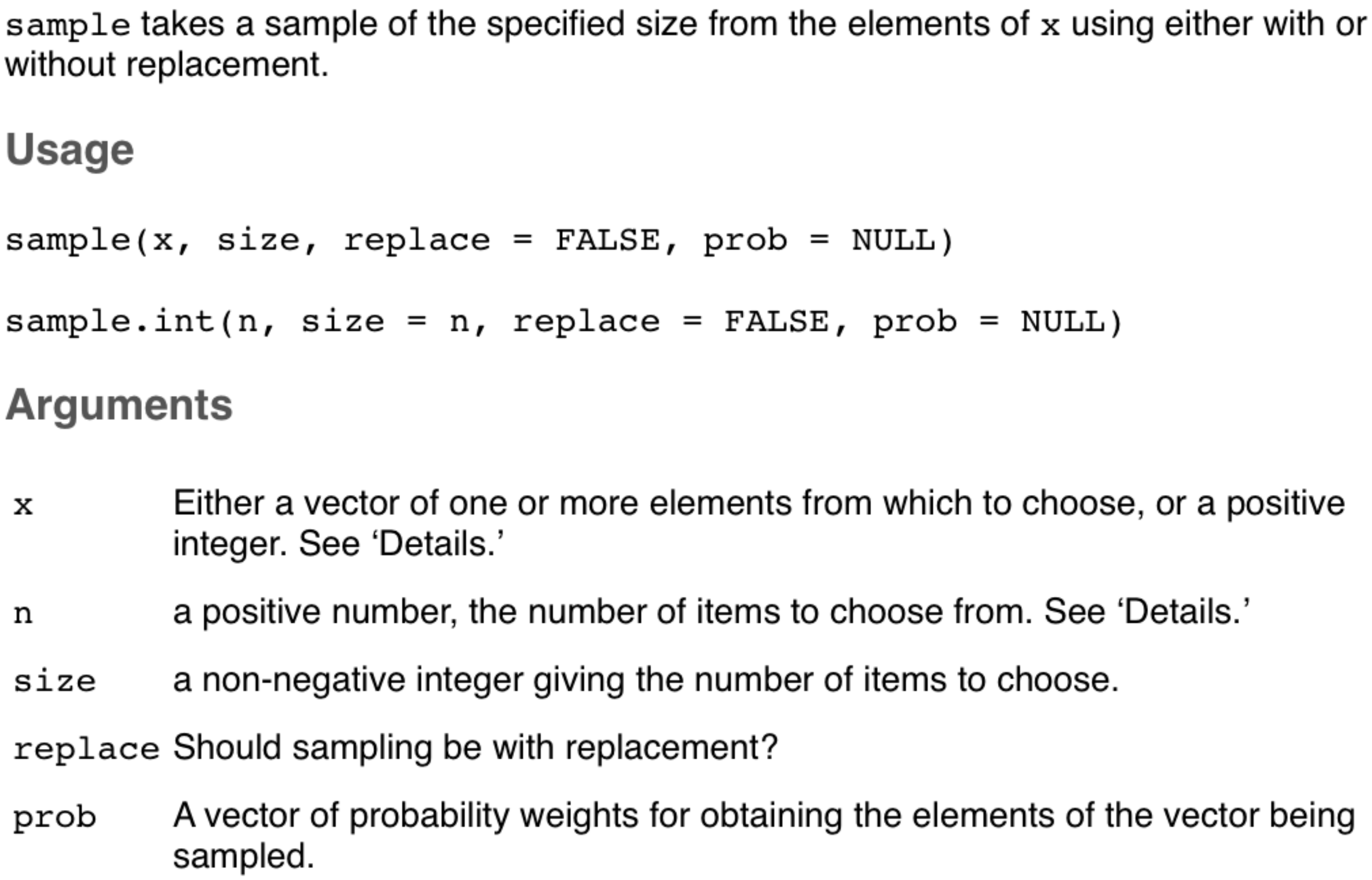
- Vectors can also be processed using the vectorized **ifelse(b,u,v)** function, which accepts a boolean vector **b** and allocates the element-wise results to be either **u** or **v**.

```
> v1
[1]  1  4  9 16 25

> c1<-ifelse(v1%%2==0,"Even","Odd")

> c1
[1] "Odd"  "Even" "Odd"  "Even" "Odd"
```

sample()



sample takes a sample of the specified size from the elements of x using either with or without replacement.

Usage

```
sample(x, size, replace = FALSE, prob = NULL)

sample.int(n, size = n, replace = FALSE, prob = NULL)
```

Arguments

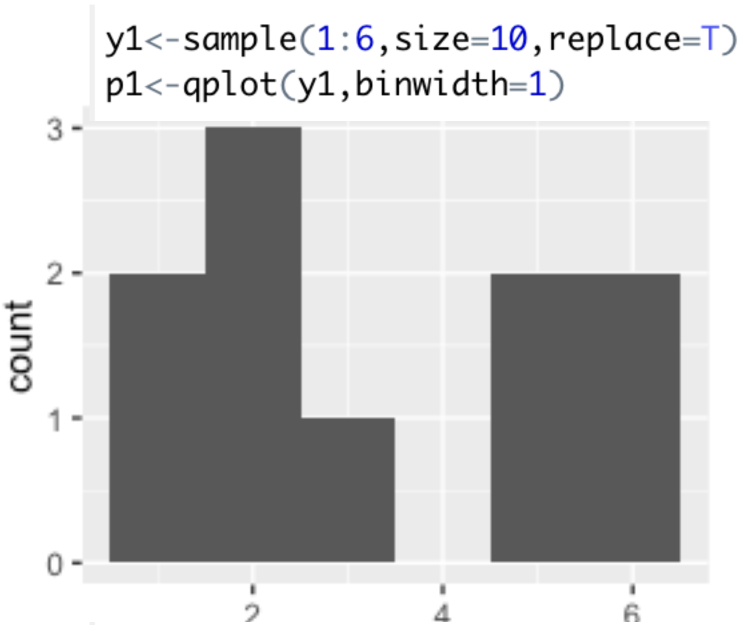
x Either a vector of one or more elements from which to choose, or a positive integer. See 'Details.'

n a positive number, the number of items to choose from. See 'Details.'

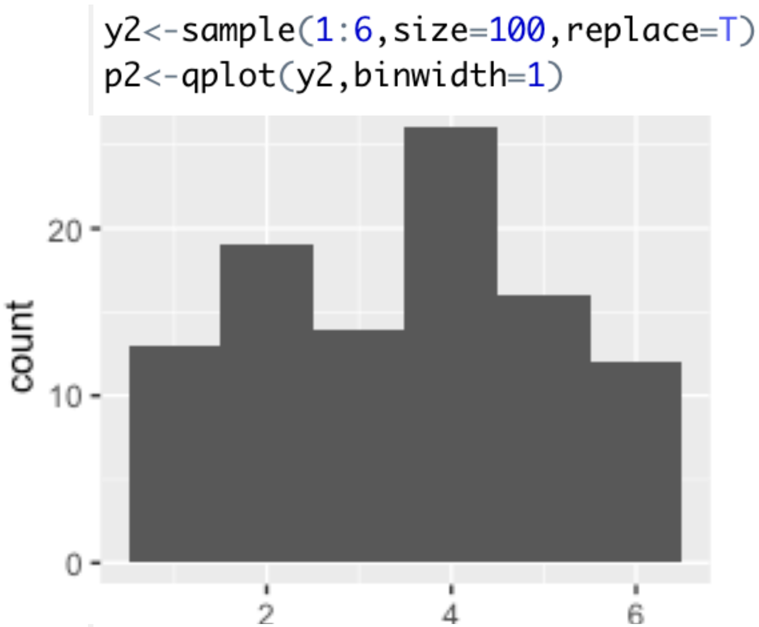
size a non-negative integer giving the number of items to choose.

replace Should sampling be with replacement?

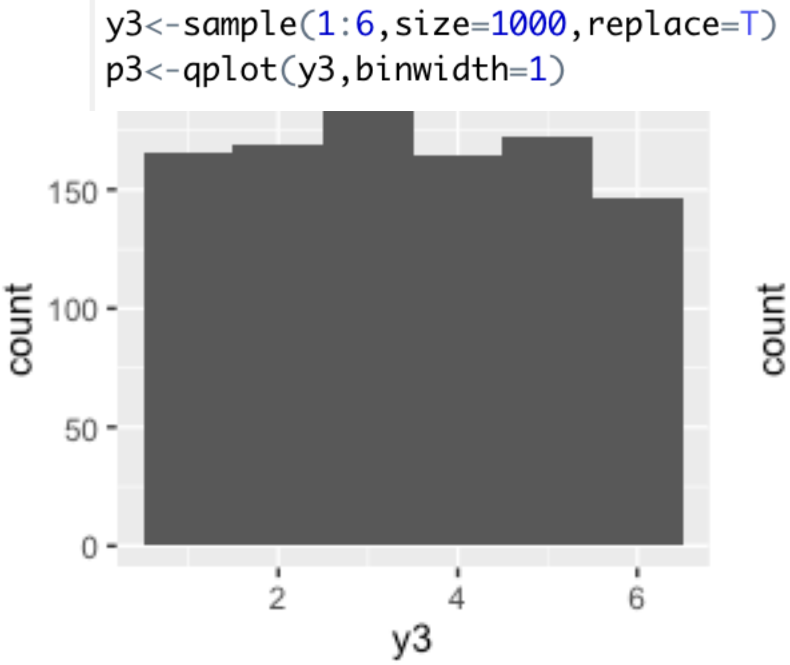
prob A vector of probability weights for obtaining the elements of the vector being sampled.



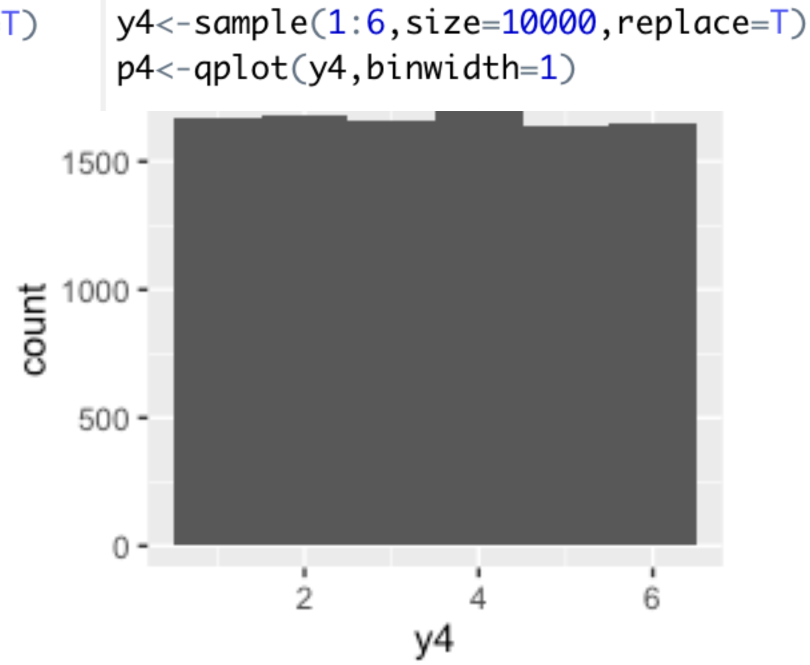
```
y1<-sample(1:6,size=10,replace=T)
p1<-qplot(y1,binwidth=1)
```



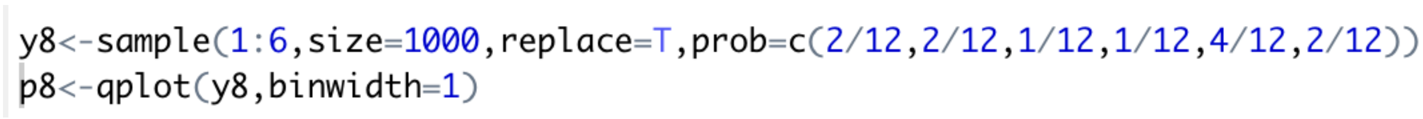
```
y2<-sample(1:6,size=100,replace=T)
p2<-qplot(y2,binwidth=1)
```



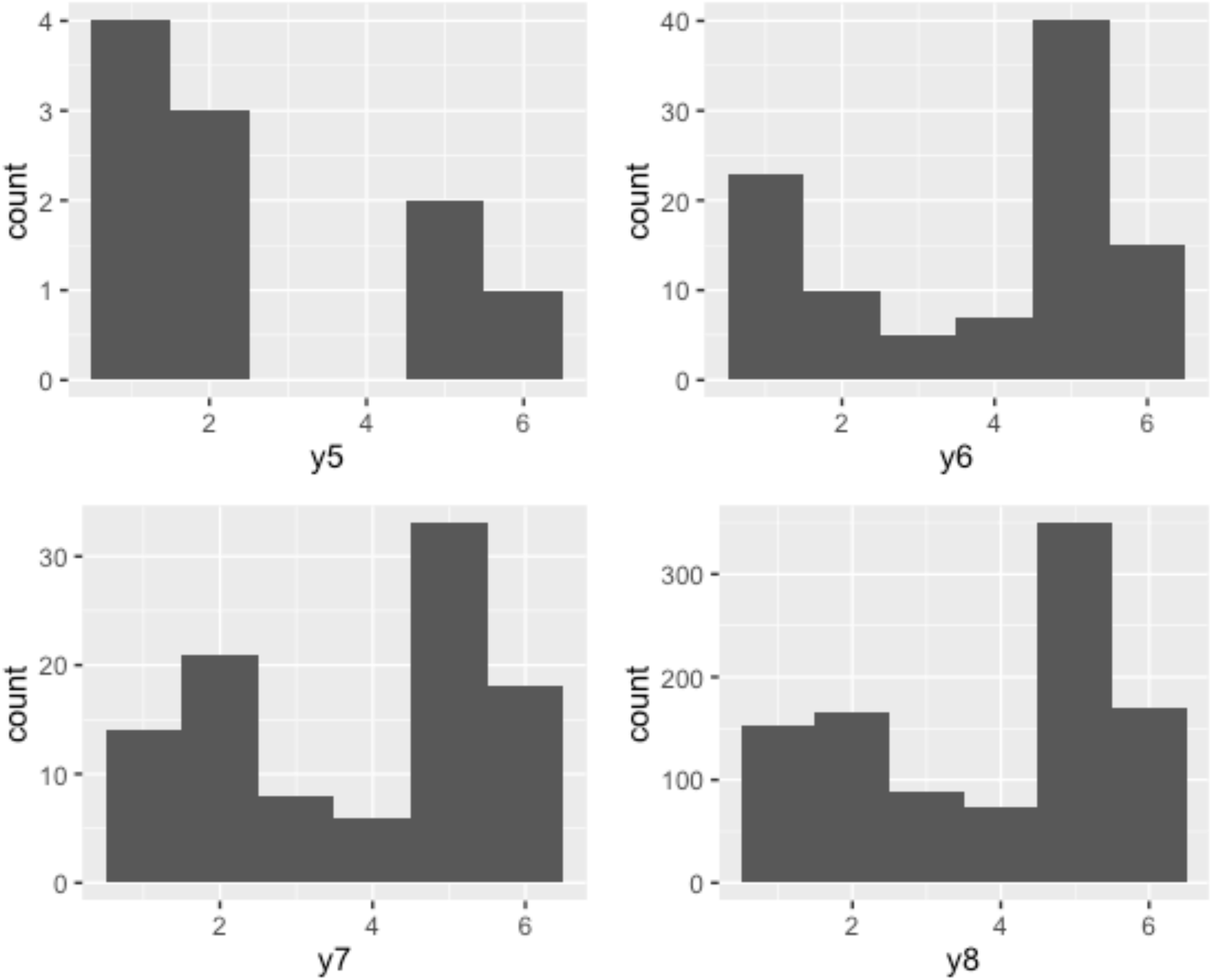
```
y3<-sample(1:6,size=1000,replace=T)
p3<-qplot(y3,binwidth=1)
```



```
y4<-sample(1:6,size=10000,replace=T)
p4<-qplot(y4,binwidth=1)
```



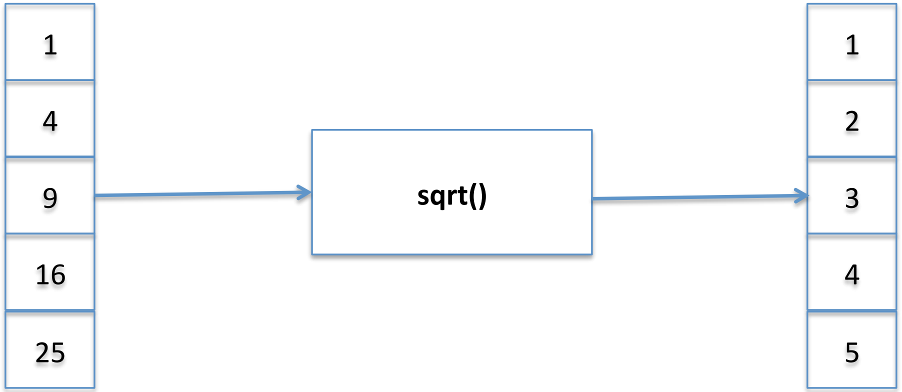
```
y8<-sample(1:6,size=1000,replace=T,prob=c(2/12,2/12,1/12,1/12,4/12,2/12))
p8<-qplot(y8,binwidth=1)
```



Summary

- R Data Types (1d, 2d, nd)
- Atomic Vectors in R
  - 4 main types
  - Rules of coercion
  - Subsetting methods
- Vectorisation
  - Operations and functions
- Useful functions
  - qplot
  - sample()

	Homogenous	Heterogenous
1d	Atomic Vector	List
2d	Matrix	Data Frame
nd	Array	



References

- Chambers, J. 2008. Software for data analysis: programming with R. Springer Science & Business Media. Chicago.
- Grolemund, G. 2014. Hands-On Programming with R. O'Reilly Press.
- Matloff, N. 2009. The Art of R Programming. No Starch Press, San Francisco, CA.
- Wickham, H. 2015. Advanced R. Taylor & Francis