Basic Usage of R

Data Analysis with R and Python

Deepayan Sarkar

R is a full programming language

- Variables
- Functions
- Control flow structures
  - For loops, while loops
  - If-then-else (branching)

R is a full programming language

- Variables
- Functions
- Control flow structures
  - For loops, while loops
  - If-then-else (branching)
- Distinguishing features
  - Focus on *vectors* and *vectorized operations*
  - Treatment of *functions* at par with other object types

R is easily extensible

- Most standard data analysis methods are already implemented
- Can be extended by writing add-on packages
- Thousands of add-on packages are available

Major concepts we need to know

- Variables (in the context of programming)

Major concepts we need to know

- Variables (in the context of programming)
- Data structures needed for data analyis

Major concepts we need to know

- Variables (in the context of programming)
- Data structures needed for data analyis
- Functions (set of instructions for performing a procedure)

Variables

- Variables are symbols that may be associated with different values
- Computations involving variables are done using their current value

```
sqrt(x)
```

```
Error: object 'x' not found
```

Variables

- Variables are symbols that may be associated with different values
- Computations involving variables are done using their current value

```
x <- 10 # assignment
sqrt(x)
```

```
[1] 3.162278
```

```
x <- -1
sqrt(x)
```

```
Warning in sqrt(x): NaNs produced
```

```
[1] NaN
```

```
x <- -1+0i
sqrt(x)
```

```
[1] 0+1i
```

Data structures for data analysis

- Vectors
- Matrices
- Data frames (a spreadsheet-like data set)
- Lists (general collection of objects)

Atomic vectors

- Indexed collection of homogeneous scalars, can be
  - Numeric / Integer / Complex
  - Character
  - Logical (`TRUE` / `FALSE`)

Atomic vectors

- Indexed collection of homogeneous scalars, can be
  - Numeric / Integer / Complex
  - Character
  - Logical (`TRUE` / `FALSE`)
- Missing values are allowed, indicated as `NA`

Atomic vectors

- Indexed collection of homogeneous scalars, can be
  - Numeric / Integer / Complex
  - Character
  - Logical (`TRUE` / `FALSE`)
- Missing values are allowed, indicated as `NA`
- Elements are indexed starting from 1

Atomic vectors

- Indexed collection of homogeneous scalars, can be
  - Numeric / Integer / Complex
  - Character
  - Logical (`TRUE` / `FALSE`)
- Missing values are allowed, indicated as `NA`
- Elements are indexed starting from 1
- ith element of vector `x` can be extracted using `x[[i]]`
- There are also more sophisticated forms of (vector) indexing

Atomic vectors: examples

```
month.name # built-in
```

```
 [1] "January"   "February"  "March"     "April"     "May"       "June"      "July"
 [8] "August"    "September" "October"   "November"  "December"
```

Atomic vectors: examples

```
month.name # built-in
```

```
 [1] "January"   "February"  "March"     "April"     "May"       "June"      "July"
 [8] "August"    "September" "October"   "November"  "December"
```

```
x <- rnorm(10)
x
```

```
 [1]  0.1835270  0.6246906 -1.7680396  1.1349834 -0.9381819  0.3000937  0.7678891
 [8]  1.5019791 -1.4474599  0.1974796
```

Atomic vectors: examples

```
month.name # built-in
```

```
 [1] "January"   "February"  "March"     "April"     "May"       "June"      "July"
 [8] "August"    "September" "October"   "November"  "December"
```

```
x <- rnorm(10)
x
```

```
 [1]  0.1835270  0.6246906 -1.7680396  1.1349834 -0.9381819  0.3000937  0.7678891
 [8]  1.5019791 -1.4474599  0.1974796
```

```
x[[3]] # third element of x
```

```
[1] -1.76804
```

Atomic vectors: examples

```
str(x) # useful function
```

```
num [1:10] 0.184 0.625 -1.768 1.135 -0.938 ...
```

```
str(month.name)
```

```
chr [1:12] "January" "February" "March" "April" "May" "June" "July" "August" ...
```

Creating atomic vectors

- Constructor functions

```
numeric(10)
```

```
 [1] 0 0 0 0 0 0 0 0 0 0
```

```
logical(5)
```

```
[1] FALSE FALSE FALSE FALSE FALSE
```

```
character(5)
```

```
[1] "" "" "" "" ""
```

Scalars are also vectors

- "Scalars" are just vectors of length 1

```
str(numeric(2))
```

```
num [1:2] 0 0
```

```
str(numeric(1))
```

```
num 0
```

```
str(0)
```

```
num 0
```

Vectors can have zero length

- Vectors can have length zero

```
numeric(0)
```

```
numeric(0)
```

```
logical(0)
```

```
logical(0)
```

```
length(character(0))
```

```
[1] 0
```

Vectors can have zero length

- Vectors can have length zero

```
numeric(0)
```

```
numeric(0)
```

```
logical(0)
```

```
logical(0)
```

```
length(character(0))
```

```
[1] 0
```

```
length(NULL)
```

```
[1] 0
```

Combining vectors using `c()`

- Vectors can also be created by combining smaller vectors
- For example, vectors `x` and `y` can be combined using `c(x, y)`

```
c(1:5, numeric(3))
```

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

Combining vectors using `c()`

- Vectors can also be created by combining smaller vectors
- For example, vectors `x` and `y` can be combined using `c(x, y)`

```
c(1:5, numeric(3))
```

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

- Any number of vectors can be combined
- This is a common way to build up a vector using scalars

```
c(2, 4, 6, 9, 11)
```

```
[1]  2  4  6  9 11
```

Combining vectors of different types

- Atomic vectors of different types cannot be combined
- Attempting to do so will convert into one of the types

```
c(1:5, c(TRUE, FALSE))
```

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

```
c(1:5, month.name[[1]])
```

```
[1] "1"       "2"       "3"       "4"       "5"       "January"
```

Combining vectors of different types

- Atomic vectors of different types cannot be combined
- Attempting to do so will convert into one of the types

```
c(1:5, c(TRUE, FALSE))
```

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

```
c(1:5, month.name[[1]])
```

```
[1] "1"       "2"       "3"       "4"       "5"       "January"
```

```
c(1:5, c(TRUE, FALSE), month.name[[1]])
```

```
[1] "1"       "2"       "3"       "4"       "5"       "TRUE"    "FALSE"   "January"
```

```
c(c(1:5, TRUE, FALSE), month.name[[1]])
```

```
[1] "1"       "2"       "3"       "4"       "5"       "1"       "0"       "January"
```

Example: Our first dataset

- Life expectancy in different countries over time

year	Australia	France	India	Zimbabwe
1952	69.120	67.410	37.373	48.451
1957	70.330	68.930	40.249	50.469
1962	70.930	70.510	43.605	52.358
1967	71.100	71.550	47.193	53.995
1972	71.930	72.380	50.651	55.635
1977	73.490	73.830	54.208	57.674
1982	74.740	74.890	56.596	60.363
1987	76.320	76.340	58.553	62.351
1992	77.560	77.460	60.223	60.377
1997	78.830	78.640	61.765	46.809
2002	80.370	79.590	62.879	39.989
2007	81.235	80.657	64.698	43.487

Life Exepectancy in France

```
year <- c(1952, 1957, 1962, 1967, 1972, 1977,
          1982, 1987, 1992, 1997, 2002, 2007)
year
```

```
[1] 1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 2002 2007
```

Life Exepectancy in France

```
year <- c(1952, 1957, 1962, 1967, 1972, 1977,
          1982, 1987, 1992, 1997, 2002, 2007)
year
```

```
[1] 1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 2002 2007
```

```
lexp_france <- c(67.41, 68.93, 70.51, 71.55, 72.38, 73.83, 74.89, 76.34,
              77.46, 78.64, 79.59, 80.657)
lexp_france
```

```
[1] 67.410 68.930 70.510 71.550 72.380 73.830 74.890 76.340 77.460 78.640 79.590 80.657
```

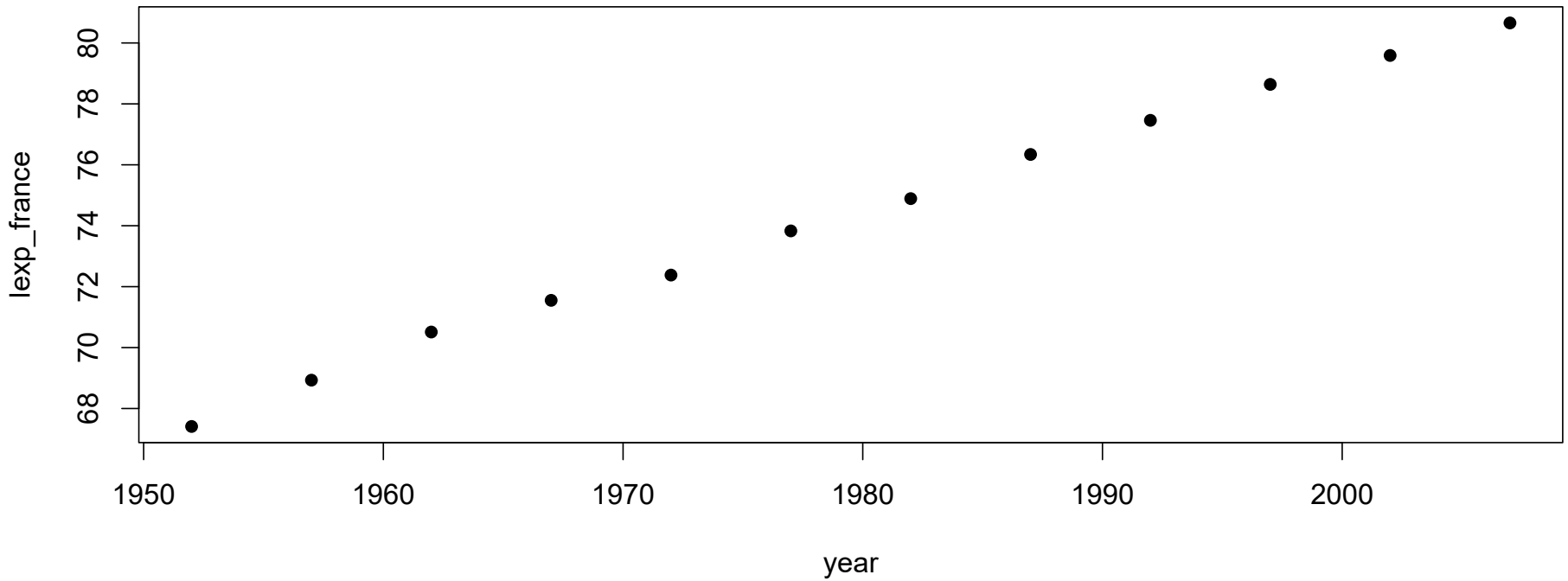
Life Exepectancy in France

```
year <- seq(1952, 2007, by = 5)
year
```

```
[1] 1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 2002 2007
```

Life Exepectancy in France

```
plot(year, lexp_france, pch = 16)
```



Life Exepectancy in France

```
lexp_france[[2]] - lexp_france[[1]]
```

```
[1] 1.52
```

Life Exepectancy in France

```
c(lexp_france[[2]] - lexp_france[[1]], lexp_france[[3]] - lexp_france[[2]],
  lexp_france[[4]] - lexp_france[[3]], lexp_france[[5]] - lexp_france[[4]],
  lexp_france[[6]] - lexp_france[[5]], lexp_france[[7]] - lexp_france[[6]],
  lexp_france[[8]] - lexp_france[[7]], lexp_france[[9]] - lexp_france[[8]],
  lexp_france[[10]] - lexp_france[[9]], lexp_france[[11]] - lexp_france[[10]],
  lexp_france[[12]] - lexp_france[[11]])
```

```
[1] 1.520 1.580 1.040 0.830 1.450 1.060 1.450 1.120 1.180 0.950 1.067
```

Life Exepectancy in France

```
d <- numeric(0)
```

Life Exepectancy in France

```
d <- numeric(0)
```

```
for (i in 1:11) {
    d <- c(d, lexp_france[[i+1]] - lexp_france[[i]])
}
d
```

```
[1] 1.520 1.580 1.040 0.830 1.450 1.060 1.450 1.120 1.180 0.950 1.067
```

Life Exepectancy in France

```
lexp_france[-1] - lexp_france[-12]
```

```
[1] 1.520 1.580 1.040 0.830 1.450 1.060 1.450 1.120 1.180 0.950 1.067
```

Life Exepectancy in France

```
lexp_france[-1] - lexp_france[-12]
```

```
[1] 1.520 1.580 1.040 0.830 1.450 1.060 1.450 1.120 1.180 0.950 1.067
```

```
diff(lexp_france)
```

```
[1] 1.520 1.580 1.040 0.830 1.450 1.060 1.450 1.120 1.180 0.950 1.067
```

Life Exepectancy in France

```
d <- diff(lexp_france)
median(d)
```

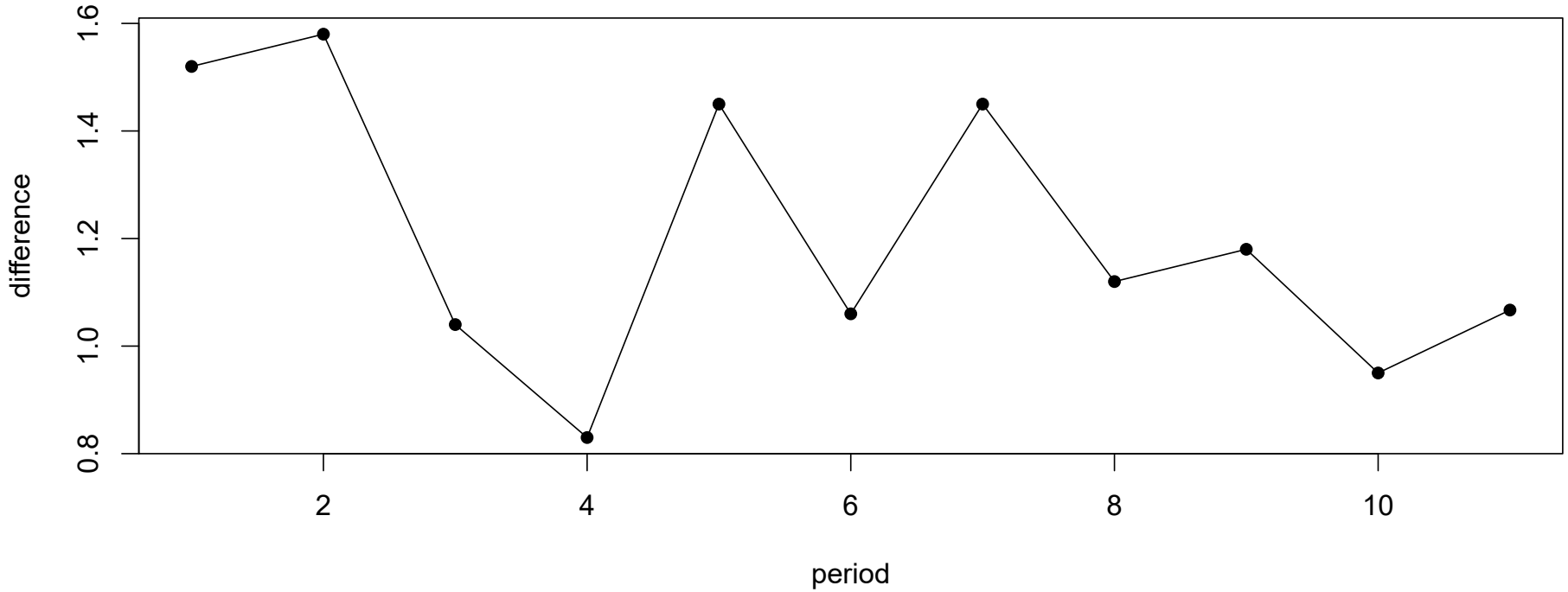
```
[1] 1.12
```

```
mean(d)
```

```
[1] 1.204273
```

Life Exepectancy in France

```
plot(d, pch = 16, type = "o", ylab = "difference", xlab = "period")
```



Types of vector indexing

- Indexing refers to extracting subsets of data
- R supports several kinds of indexing:
  - Indexing with a vector of positive integers
  - Indexing with a vector of negative integers
  - Indexing with a logical vector
  - Indexing with a vector of names

The empty index

- A vector indexing operation has the form `x[index]`

The empty index

- A vector indexing operation has the form `x[index]`
- The most basic form is an empty index, which selects all elements

```
month.name[]
```

```
 [1] "January"   "February"  "March"     "April"     "May"       "June"      "July"
 [8] "August"    "September" "October"   "November"  "December"
```

Indexing with an integer vector

- For integer indexing, `index` is an integer vector

```
month.name[c(2, 4, 6, 9, 11)]
```

```
[1] "February"  "April"     "June"      "September" "November"
```

Indexing with an integer vector

- For integer indexing, `index` is an integer vector

```
month.name[c(2, 4, 6, 9, 11)]
```

```
[1] "February"  "April"     "June"      "September" "November"
```

- Elements can be repeated

```
month.name[c(2, 2, 6, 4, 6, 11)]
```

```
[1] "February" "February" "June"     "April"    "June"     "November"
```

Indexing with an integer vector

- "Out-of-bounds" indexing gives `NA` (missing)

```
month.name[13]
```

```
[1] NA
```

Indexing with an integer vector

- "Out-of-bounds" indexing gives `NA` (missing)

```
month.name[13]
```

```
[1] NA
```

```
seq(1, by = 2, length.out = 8)
```

```
[1]  1  3  5  7  9 11 13 15
```

Indexing with an integer vector

- "Out-of-bounds" indexing gives `NA` (missing)

```
month.name[13]
```

```
[1] NA
```

```
seq(1, by = 2, length.out = 8)
```

```
[1]  1  3  5  7  9 11 13 15
```

```
month.name[seq(1, by = 2, length.out = 8)]
```

```
[1] "January"   "March"     "May"       "July"      "September" "November"  NA
[8] NA
```

Indexing with an integer vector

- Indexing with a scalar (vector of length 1) also works:

```
month.name[2]
```

```
[1] "February"
```

Indexing with an integer vector

- Indexing with a scalar (vector of length 1) also works:

```
month.name[2]
```

```
[1] "February"
```

- This is usually the same as `x[[index]]`

```
month.name[[2]]
```

```
[1] "February"
```

Indexing with an integer vector

- Indexing with a scalar (vector of length 1) also works:

```
month.name[2]
```

```
[1] "February"
```

- This is usually the same as `x[[index]]`

```
month.name[[2]]
```

```
[1] "February"
```

- However, these differ in the behaviour when an index is out of bound

```
month.name[15]
```

```
[1] NA
```

```
month.name[[15]]
```

```
Error in month.name[[15]]: subscript out of bounds
```

Indexing with a vector of negative integers

- Negative integers omit the specified entries

```
month.name[-2]
```

```
 [1] "January"   "March"     "April"     "May"       "June"      "July"      "August"
 [8] "September" "October"   "November"  "December"
```

```
month.name[-c(2, 4, 6, 9, 11)]
```

```
[1] "January"  "March"    "May"      "July"     "August"   "October"  "December"
```

Indexing with a vector of negative integers

- Negative integers omit the specified entries

```
month.name[-2]
```

```
 [1] "January"   "March"     "April"     "May"       "June"      "July"      "August"
 [8] "September" "October"   "November"  "December"
```

```
month.name[-c(2, 4, 6, 9, 11)]
```

```
[1] "January"  "March"    "May"      "July"     "August"   "October"  "December"
```

- Cannot be mixed with positive integers

```
month.name[c(2, -3)]
```

```
Error in month.name[c(2, -3)]: only 0's may be mixed with negative subscripts
```

Indexing with 0

- Zero has a special meaning - it doesn't select anything

```
month.name[0]
```

```
character(0)
```

```
month.name[integer(0)] ## same as empty index
```

```
character(0)
```

```
month.name[c(1, 2, 0, 11, 12)]
```

```
[1] "January"  "February" "November" "December"
```

```
month.name[-c(1, 2, 0, 11, 12)]
```

```
[1] "March"     "April"     "May"       "June"      "July"      "August"    "September"
[8] "October"
```

Indexing with a logical vector

- Indexing by logical vector: select `TRUE` elements

```
month.name[c(TRUE, FALSE, FALSE)] # index replicated
```

```
[1] "January" "April"   "July"    "October"
```

Indexing with a logical vector

- Indexing by logical vector: select `TRUE` elements

```
i <- substring(month.name, 1, 1) == "J"
i
```

```
 [1]  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
```

Indexing with a logical vector

- Indexing by logical vector: select `TRUE` elements

```
i <- substring(month.name, 1, 1) == "J"
i
```

```
 [1]  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
```

```
month.name[i]
```

```
[1] "January" "June"    "July"
```

Indexing with a logical vector

```
(x <- rnorm(20))
```

```
 [1]  0.4551969 -0.4193646  0.1298366  1.7835150 -0.2016912 -0.6367525 -0.4775448
 [8] -0.2687966  0.2538650  0.8750837 -2.1376604 -0.6995476  0.0897731  0.6781549
[15]  0.4202109  0.6365381  1.1706711  0.7683105  0.4760369  1.1587456
```

```
x > 0
```

```
 [1]  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE  TRUE
[15]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
x[x > 0]
```

```
 [1] 0.4551969 0.1298366 1.7835150 0.2538650 0.8750837 0.0897731 0.6781549 0.4202109
 [9] 0.6365381 1.1706711 0.7683105 0.4760369 1.1587456
```

```
mean(x[x > 0])
```

```
[1] 0.6843029
```

Converting a logical index vector to integer

- Logical indexing may be replaced by integer indexing using `which()`

```
i
```

```
 [1]  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
```

```
which(i)
```

```
[1] 1 6 7
```

Converting a logical index vector to integer

- Logical indexing may be replaced by integer indexing using `which()`

```
i
```

```
 [1]  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
```

```
which(i)
```

```
[1] 1 6 7
```

```
month.name[ which(i) ]
```

```
[1] "January" "June"    "July"
```

Converting a logical index vector to integer

- Logical indexing may be replaced by integer indexing using `which()`

```
i
```

```
 [1]  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
```

```
which(i)
```

```
[1] 1 6 7
```

```
month.name[ which(i) ]
```

```
[1] "January" "June"    "July"
```

```
month.name[ -which(i) ] # same as month.name[ !i ]
```

```
[1] "February"  "March"     "April"     "May"       "August"    "September" "October"
[8] "November"  "December"
```

Converting a logical index vector to integer

- But be careful about zero-length indices

```
which(substring(month.name, 1, 1) == "B")
```

```
integer(0)
```

```
month.name[ which( substring(month.name, 1, 1) == "B") ]
```

```
character(0)
```

```
-which(substring(month.name, 1, 1) == "B")
```

```
integer(0)
```

```
month.name[ -which( substring(month.name, 1, 1) == "B") ]
```

```
character(0)
```

Indexing with a vector of names

- Vectors can optionally have names — one for each element
- These are usually informative labels

Indexing with a vector of names

- Vectors can optionally have names — one for each element
- These are usually informative labels
- Example: quantiles of a Normal random sample

```
x <- rnorm(100)
qx <- quantile(x)
qx
```

```
         0%         25%         50%         75%        100% 
-2.45118171 -0.70671319  0.05292767  0.49462338  1.90524820 
```

```
names(qx)
```

```
[1] "0%"   "25%"  "50%"  "75%"  "100%"
```

Indexing with a vector of names

- Vectors can optionally have names — one for each element
- These are usually informative labels
- Example: quantiles of a Normal random sample

```
x <- rnorm(100)
qx <- quantile(x)
qx
```

```
         0%         25%         50%         75%        100% 
-2.45118171 -0.70671319  0.05292767  0.49462338  1.90524820 
```

```
names(qx)
```

```
[1] "0%"   "25%"  "50%"  "75%"  "100%"
```

```
names(x) # no names
```

```
NULL
```

Indexing with a vector of names

- When present, names may be used to identify elements
- Indexing with names works in the same way as positive integers
- Instead of position, the corresponding named element is selected

```
qx[["50%"]] ## extracting a single element using scalar indexing
```

```
[1] 0.05292767
```

```
qx["50%"]   ## extracting a single element with vector indexing
```

```
       50%
0.05292767
```

```
qx[c("25%", "75%")]  ## extracting multiple elements
```

```
       25%        75%
-0.7067132  0.4946234
```

Indexing with a vector of names

- Inter-quartile range

```
diff(qx[c("25%", "75%")])
```

```
     75%
1.201337
```

Indexing with a vector of names

- Inter-quartile range

```
diff(qx[c("25%", "75%")])
```

```
     75%
1.201337
```

```
IQR(x)
```

```
[1] 1.201337
```

Indexing with a vector of names

- Unmatched names are treated like out-of-bound indexes

```
qx[["95%"]]
```

```
Error in qx[["95%"]]: subscript out of bounds
```

```
qx["95%"]
```

```
<NA>
  NA
```