Data Frames

- Data frames are used to store tabular data in R. Hadley Wickham's package `dplyr35` has an optimized set of functions designed to work efficiently with data frames.
- Data frames are represented as a special type of list where every element of the list has to have the same length. Each element of the list can be thought of as a column and the length of each element of the list is the number of rows.
- Unlike matrices, data frames can store different classes of objects in each column. Matrices must have every element be the same class (e.g. all integers or all numeric).

Data Frames

- In addition to column names, indicating the names of the variables or predictors, data frames have a special attribute called `row.names` which indicate information about each row of the data frame.
- Data frames are usually created by reading in a dataset using the `read.table()` or `read.csv()` .However, data frames can also be created explicitly with the `data.frame()` function.

Data Frames: Construction

- `x <- data.frame(foo = 1:4, bar = c(T, T, F, F))`
- `x`

```
   foo  bar
1  1    TRUE
2  2    TRUE
3  3    FALSE
4  4    FALSE
```

- `nrow(x)`

```
[1] 4
```

- `ncol(x)`

```
[1] 2
```

Accessing Data Frame

- vector1 <- c(10,30,40,15); vector1
- vector2 <- vector1 * 2
- df1 <- data.frame(vector1,vector2); df1
- vector3 <- c(1,2,3)
- vector3 <- c(1,2,3,4)
- df1 <- data.frame(vector1,vector2,vector3); df1 # gives error
- df1[1,2]
- df1[,2]

Accessing Data Frame

- df1$vector1 #similar to the use of $ in the case of list, because df is a list
- df1[df1$vector1<20,2]
- df1
- df1$vector1<20
- df1[1:3,2]
- is.list(df1$vector1[1:2])
- is.vector(df1$vector1[1:2])
- str(df1)

Removing NAs in Data Frames

- `head(airquality)`
- `good <- complete.cases(airquality)`
- `head(airquality[good, ])`
- `x <- airquality[,-1] # x is a regression design matrix`
- `y <- airquality[, 1] # y is the corresponding response`
- `stopifnot(complete.cases(y) != is.na(y))`
- `ok <- complete.cases(x, y)`
- `sum(!ok) # how many are not "ok" ?`
- `x <- x[ok,]`
- `y <- y[ok]`

Reading and Writing Data

There are a few principal functions reading data into R.

- `read.table, read.csv` , for reading tabular data
- `readLines` , for reading lines of a text file

Reading and Writing Data

There are analogous functions for writing data to files

- `write.table` , for writing tabular data to text files (i.e. CSV) or connections
- `writeLines` , for writing character data line-by-line to a file or connection

read.table – important parameters

- `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
- `na.strings –` the set of strings to be considered as NA

Reading Data

- tab1 <- read.table('house_copy.txt')
- tab1
- getwd()
- str(tab1)
- tab1 <- read.table(file='house_copy.txt',header = T)
- tab1 <- read.table(file='house_copy.txt',header = T, sep = "",colClasses = c('double','double'))
- tab1 <- read.table(file='house_copy.txt',header = T, sep = "",colClasses = c('double','double'), na.strings = c('220.0'))
- tab1 <- read.table("home/user/house.txt") #using relative path notation
- #please experiment with relative/absolute path in Windows - use getwd() for hints
- #experiment with read.csv and csv files (comma-separated files like from excel)
- vector1 <- c(10,30,40,15); vector1
- vector1
- writeLines(as.character(vector1), 'vector1.txt')
- getwd()
- write.table(tab1,file = 'area_sales.txt',sep=',')
- write.table(tab1,file = 'area_sales.txt',sep=',',quote = F,row.names = F)

Exercise

- Experiment with variations in write.table function (command) to get the data in the orginal format as shown below:

```
area,sale.price
694,192
905,215
802,215
1366,274
716,112.7
963,185
821,212
714,220
1018,276
887,260
790,221.5
696,255
771,260
1006,293
1191,375
```

- Experiment reading CSV files using read.csv function and explore usage of its various parameters (arguments)

Testing and Coercion – compilation

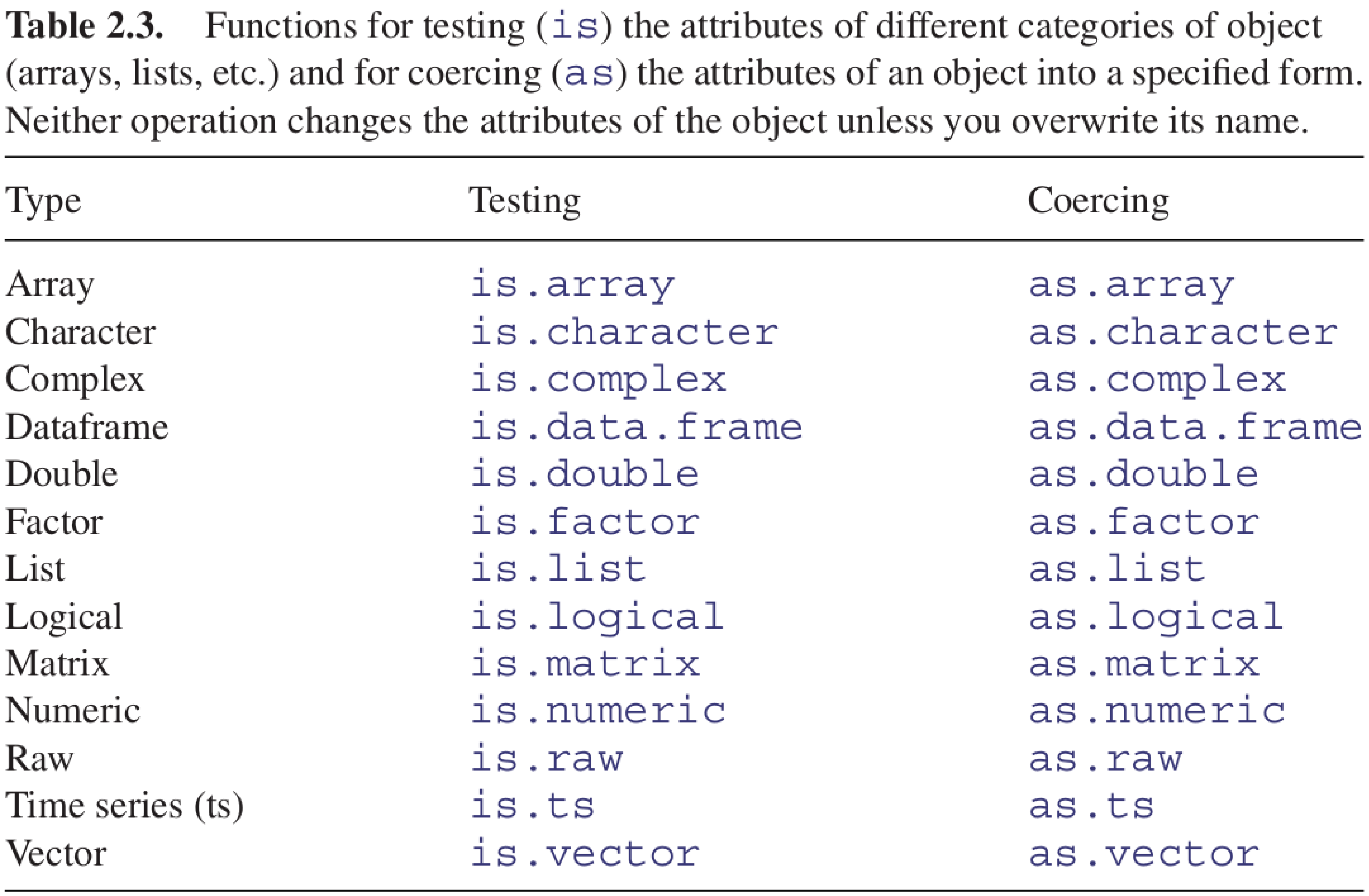

Table 2.3. Functions for testing (`is`) the attributes of different categories of object (arrays, lists, etc.) and for coercing (`as`) the attributes of an object into a specified form. Neither operation changes the attributes of the object unless you overwrite its name.

Type	Testing	Coercing
Array	`is.array`	`as.array`
Character	`is.character`	`as.character`
Complex	`is.complex`	`as.complex`
Dataframe	`is.data.frame`	`as.data.frame`
Double	`is.double`	`as.double`
Factor	`is.factor`	`as.factor`
List	`is.list`	`as.list`
Logical	`is.logical`	`as.logical`
Matrix	`is.matrix`	`as.matrix`
Numeric	`is.numeric`	`as.numeric`
Raw	`is.raw`	`as.raw`
Time series (ts)	`is.ts`	`as.ts`
Vector	`is.vector`	`as.vector`

Table 2.4. Vector functions used in R.

Operation	Meaning
`max(x)`	maximum value in x
`min(x)`	minimum value in x
`sum(x)`	total of all the values in x
`mean(x)`	arithmetic average of the values in x
`median(x)`	median value in x
`range(x)`	vector of min(x) and max(x)
`var(x)`	sample variance of x
`cor(x,y)`	correlation between vectors x and y
`sort(x)`	a sorted version of x
`rank(x)`	vector of the ranks of the values in x
`order(x)`	an integer vector containing the permutation to sort x into ascending order
`quantile(x)`	vector containing the minimum, lower quartile, median, upper quartile, and maximum of x
`cumsum(x)`	vector containing the sum of all of the elements up to that point
`cumprod(x)`	vector containing the product of all of the elements up to that point
`cummax(x)`	vector of non-decreasing numbers which are the cumulative maxima of the values in x up to that point
`cummin(x)`	vector of non-increasing numbers which are the cumulative minima of the values in x up to that point
`pmax(x,y,z)`	vector, of length equal to the longest of x, y or z, containing the maximum of x, y or z for the ith position in each
`pmin(x,y,z)`	vector, of length equal to the longest of x, y or z, containing the minimum of x, y or z for the ith position in each
`colMeans(x)`	column means of dataframe or matrix x
`colSums(x)`	column totals of dataframe or matrix x
`rowMeans(x)`	row means of dataframe or matrix x
`rowSums(x)`	row totals of dataframe or matrix x