R DATA STRUCTURES

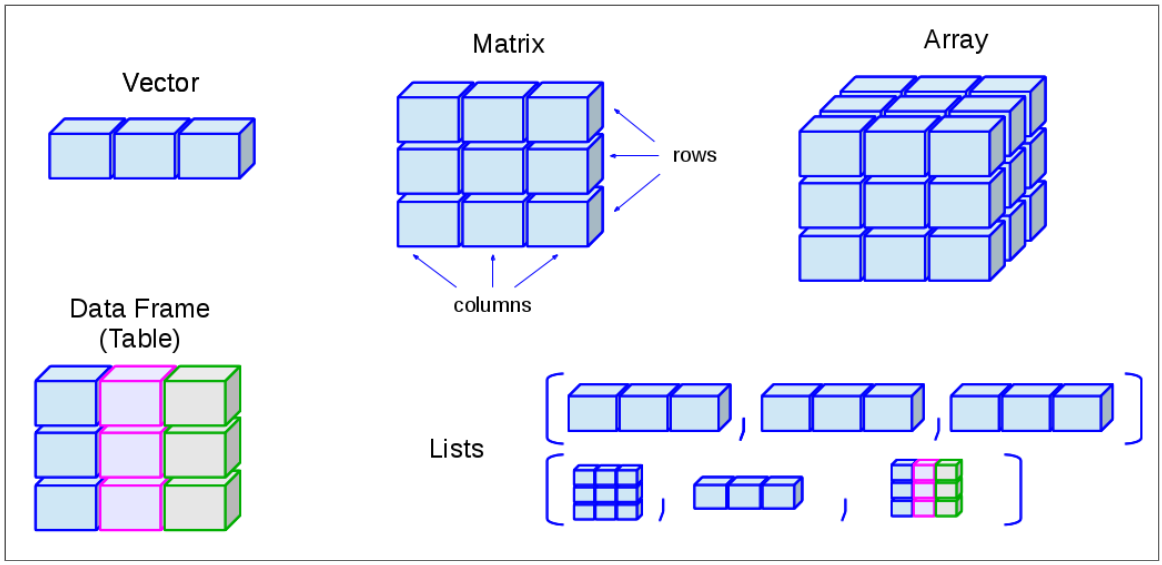
(DSC 101)
MATLOFF / BIRKENKRAHE

Created: 2021-09-21 Di 10:28

WHAT WILL YOU LEARN?

- Preview of important data structures
- Vectors and scalars
- Character strings
- Matrices
- Lists
- Data Frames
- Classes
- Extended example

OVERVIEW



VECTORS AND SCALARS

VECTORS

- Storage modes: check `?mode`
- Functions: `mode`, `storage.mode`, `typeof`
- E.g. `numeric` (`double` or `integer`)
- Create a numeric vector of three elements!

```
x <- c(1,2,3)  # integer
y <- rnorm(3)  # double
z <- 1:3       # integer

## print all three
x; y; z

## check mode
mode(x)
storage.mode(x)
typeof(x)

## check mode
mode(y)
storage.mode(y)
typeof(y)
```

SCALARS

- There are no scalars (numbers)
- Scalars are one-element vectors
- How could you show that?

```
s <- 1
s # prints vector of length 1

## change rownumber display
Nile[1:17]
options(width=100)
Nile[1:17]
```

CHARACTER STRINGS

- Single-element vectors of mode `character`
- Assign `x <- letters[1:3]` and print `x`
- Check the mode of `x`

```
x <- letters[1:3]
x
mode(x)
```

STRING MANIPULATION

- Create one numeric, two character vectors
- Concatenate character vectors with `paste`
- Split character vector with `strsplit`

```
## define vectors
x <- c(5,12,13)  # create numeric vecto
x                # print x
length(x)        # print length of x
mode(x)          # print mode of x

y <- "abc"       # create character st
y
length(y)
mode(y)

z <- c("abc", "29 88")
z
length(z)
mode(z)
```

CONVERSION VS. COERCION

- `character` conversion: `as.character`
- `numeric` conversion: `as.numeric`
- Change numeric vector to character
- Change character vector to numeric

```
y # three real numbers
yc <- as.character(y)
yc
mode(yc)

x # three letters
xn <- as.numeric(x)
xn
mode(xn)
```

MATRICES

- A matrix is a rectangular array of numbers
- Matrices are vectors with rows and column attributes

CREATE MATRICES WITH `matrix`

- `matrix` creates a matrix from input values

```
A <- matrix()  # an empty 1 x 1 matrix
A
dim(A) # rows x columns

B <- matrix(NA) # an empty 1 x 1 matrix
B

C <- matrix(c(1,2)) # a 2 x 1 matrix
C
is.matrix(C)   # check if it's a matrix
```

ATTACHING ROWS AND COLUMNS

- `rbind` attaches rows
- `cbind` attaches columns

```
D <- rbind(c(1,4),c(2,2))
D

E <- cbind(c(1,4),c(2,2))
E
```

MATRIX ALGEBRA

- Matrices are multiplied with %*%

```
D %*% c(1,1)
E %*% c(1,1)
D %*% E
```

MATRIX INDEXING

- Matrices are indexed with two subscripts

```
D
D[1,2]  # row 1, col 2
D[,2]   # col 2
D[2,2]  # row 2, col 2
D[1,]   # row 1
```

LISTS

- Lists can contain different data types
- This is like a `struct` in C/C++
- Access elements with two-part names

```
x <- list(u=2, v="abc") # number and str
x
mode(x)

x$u # access list element u
x$v # access list element v

y <- paste(x$u,x$v)  # concatenation lea
y
mode(y)
length(y)
```

USE OF LISTS

- Combine multiple values
- Return list by function

```
hist(Nile)         # produces graph
hn <- hist(Nile) # save histogram as lis
mode(hn)           # mode of hn
print(hn)          # print hn (we can alsc
```

- More common way to show structure with `str`

```
str(hn)
```

DATA FRAMES

- Data frames are lists made of vectors
- Vectors can have different modes
- Data frames are rectangular but not matrices

CREATE DATA FRAME

- Turn a list into a data frame using `data.frame`

```
fam <- list(kids=c("Jack","Jill"), ages=
fam
d <- data.frame(fam)
d
```

- Turn vectors directly into a data frame

```
df <- data.frame(kids=c("Jack","Jill"),ages=c(12,10))
df
```

READ DATA FRAME FROM FILE

- Use `read.table` or `read.csv`
- You can read in straight from the web

```
## read csv without header information
pima_raw <-
  read.csv(file=
             "https://raw.githubusercont
           header=FALSE, sep=,)
head(pima_raw)
```

- **Download from Kaggle** and read in from local machine

```
## read csv with header information
pima <- read.csv(file="/home/marcus/GitH
                 header=TRUE,
                 sep=,)
str(pima)
```

CLASSES

- R objects[1] are instances of *classes*
- Classes are *abstract* data types[2]
- Class instances are R lists with a class name

CLASS EXAMPLE: TIME SERIES

- The class of `Nile` is time series or `ts`

```
str(Nile)
class(Nile)
```

CLASS EXAMPLE: HISTOGRAM

- Non-graphical output of `hist()` has a class
- Compare also with `print(hn)`

```
hn <- hist(Nile)  # create a histogram o
mode(hn)          # the object is of moo
class(hn)         # its object class is
```

WHAT ARE CLASSES GOOD FOR?

- Classes are used by *generic* functions (**Chambers, 2002**)
- Generic = defines family of similar functions
- Each function fits a specific class
- This relates to R's package extensibility

GENERIC FUNCTION EXAMPLE: `summary()`

- Invoking `summary()` searches according to class, e.g.
    - Calling `summary()` on the output of `hist()`
    - Calling `summary()` on the output of `lm()` (regression)

```
summary(hn) # summarize histogram of Nil
summary(Nile) # summarize time series of
summary(lm(1:100~Nile)) # summarize line
```

GENERIC FUNCTION EXAMPLE: `plot()`

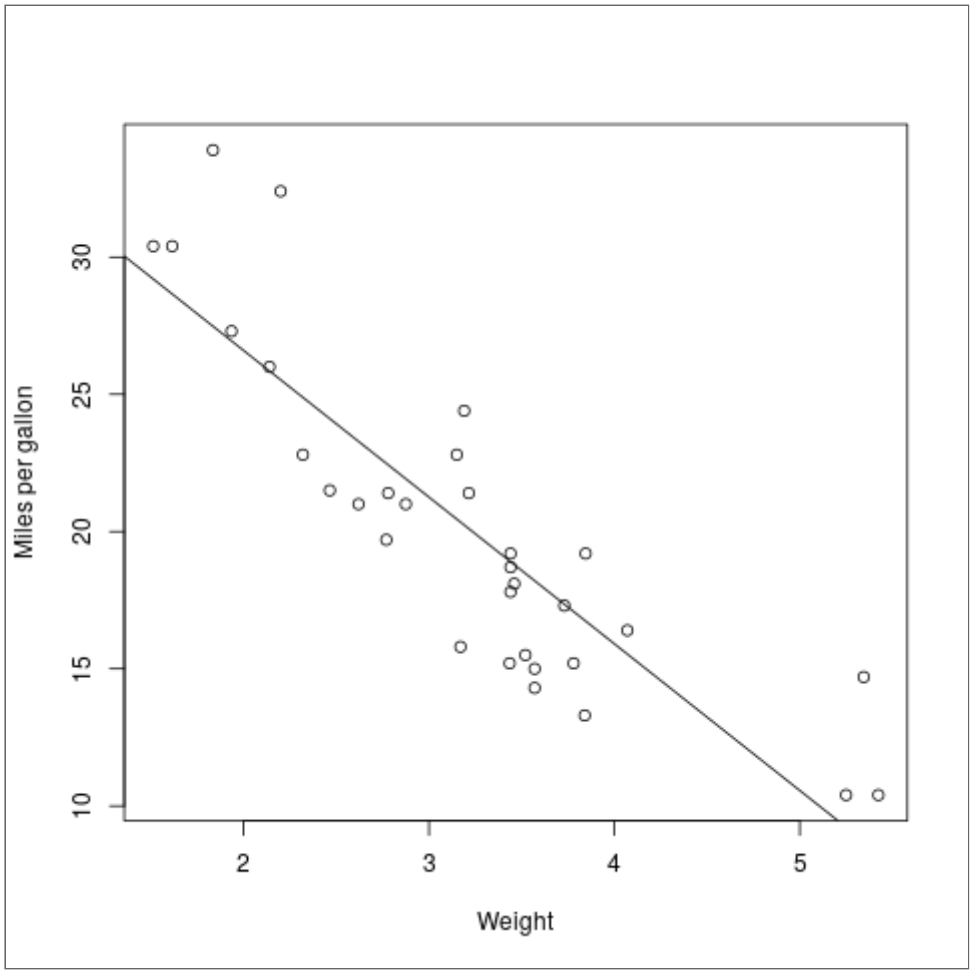
- You can call `plot()` on just about any R object, e.g.
    - Call `plot()` on a time series like `Nile`
    - Call `plot()` on a data frame like `mtcars`

```
plot(Nile) # plot of Nile time series da
```

```
plot(hn)  # plot histogram
```

```
plot(mtcars) # plot of all mtcars variab
```

EXTENDED EXAMPLE: REGRESSION ANALYSIS



OBJECTIVE

We walk through a brief statistical regression analysis - fitting a linear function to a small data set, showing different R objects along the way.

DATA SET

The file `grades.txt` contains grades. The numbers correspond to letter grades on a 5-point scale common in Continental Europe:

LETTER	POINT
A+	0.7
A	1.0
A-	1.3
B+	1.7
B	2.0
B-	2.3
C+	2.7
C	3.0
C-	3.3
D+	3.7
D	4.0
D-	4.3
F	5.0

COLUMN VECTORS

Each row contains the data for one student consisting of the midterm examination grade, the final examination grade, and the average quiz grade. We want to see how well the midterm and quiz grades predict the student's final exam grade. We'll come back to this example when we go deeper into visualization and data interpretation.

READ DATA INTO R

- Make sure you are in the right folder
- Read in data file using `read.table()`
- Don't read the first row as `header` (default)

```
setwd("/home/marcus/GitHub/dsc101/5_data
grades <- read.table(file="./data/grades
```

- Take a look at the data with `head()`
- R assigns default column vector names

```
head(grades)
```

CHECK R DATA

- `grades` is an R object of class `data.frame`
- `str()` also contains this information (and more)

```
class(grades)
str(grades)
```

MODEL DATA

- Predict finals score (V2) from midterm scores (V1)

```
lma <- lm(grades[,2] ~ grades[,1]) # using indices
```

- The `lm()` function fits a linear prediction equation: predicted final = $b_0 + b_1 * \text{midterm}$, where b_0 and b_1 are constant estimated from the data
- Check out `help(lm)` and `example(lm)` for details
  How could we also have extracted the column vectors?

EXTRACTING COLUMN VECTORS WITH $

The accessor operator $ works only for named non-atomic vectors

```
lma <- lm(grades$V2 ~ grades$V1) # using
```

EXPLORE THE `lm` OBJECT

- The fit returned by `lm()` is in an object
- The object is stored in the variable `lma`
- `lma` is an instance of the class `lm`

```
class(lma)
```

LIST COMPONENTS OF THE OBJECT WITH `attributes()`

```
attributes(lma)
```

MORE DETAILS WITH `str()`

```
str(lma)
```

OBJECTS STORE ATTRIBUTES

- Estimated linear coefficients are stored in `lma$coefficients`
- Long names can be shortened (if they're not ambiguous)

```
lma$coef
```

GENERIC FUNCTION `coef`

There is a generic function, `coef`, just for this, too

```
coef(lma)
```

USING GENERIC `print()`

You can also print the object `lma` itself

```
lma
```

- By default, this is the generic `print()` function
- `print()` hands the work over to `print.lm()`

CLASSES CONTAIN `methods()`

- See all methods of `print()` with `methods()`

```
methods("print")
```

MORE STATS WITH `summary()`

- Get more stats info with `summary()`
- It triggers a call to `summary.lm()`

```
summary(lma)
```

IMPROVING THE MODEL

- We can also estimate from both exam 1 and quiz scores
- To make it easier, we name the column vectors first

```
names(grades) <- c("final","midterm", "c
head(grades)
```

ADD PREDICTORS

- Now we use + to add another predictor variable

```
lmb <- lm(grades$final ~ grades$midterm + grades$quiz)
```

CONCEPT SUMMARY

CODE SUMMARY

CODE	DESCRIPTION

REFERENCES

Chambers J (2 Jan 2002). The Definition of Generic Functions and Methods [Website]. **Online: r-project.org.**