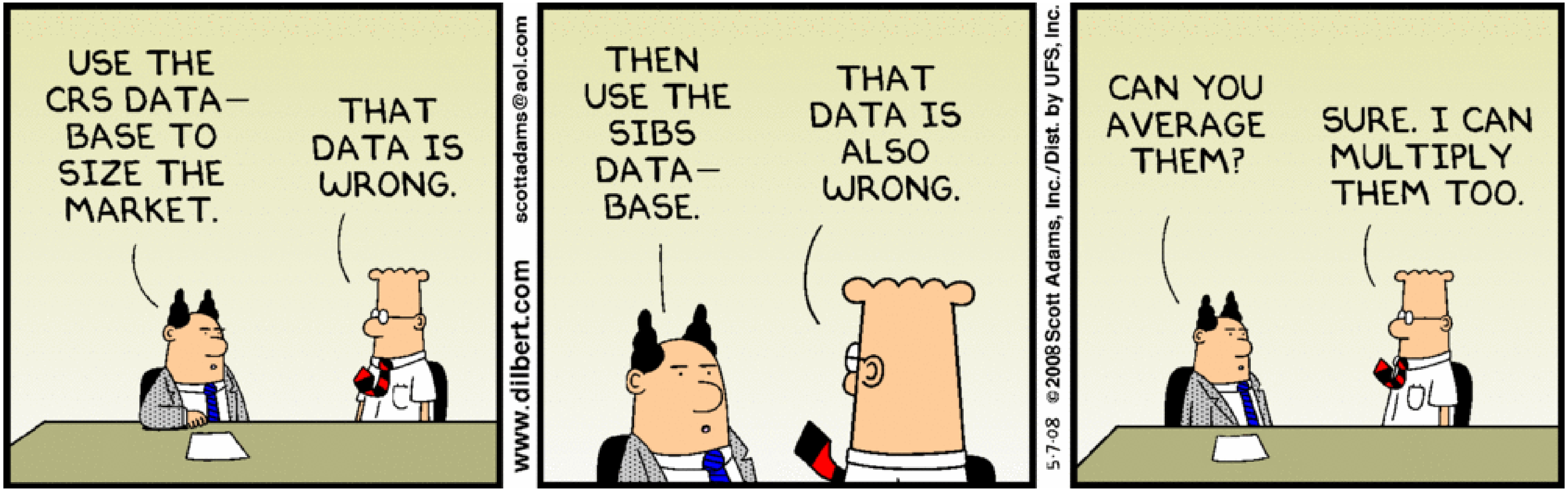
PART 2. DATA MANAGEMENT


USE THE CRS DATABASE TO SIZE THE MARKET.
THAT DATA IS WRONG.
THEN USE THE SIBS DATABASE.
THAT DATA IS ALSO WRONG.
CAN YOU AVERAGE THEM?
SURE. I CAN MULTIPLY THEM TOO.

www.dilbert.com scottadams@aol.com

5-7-08 ©2008 Scott Adams, Inc./Dist. by UFS, Inc.

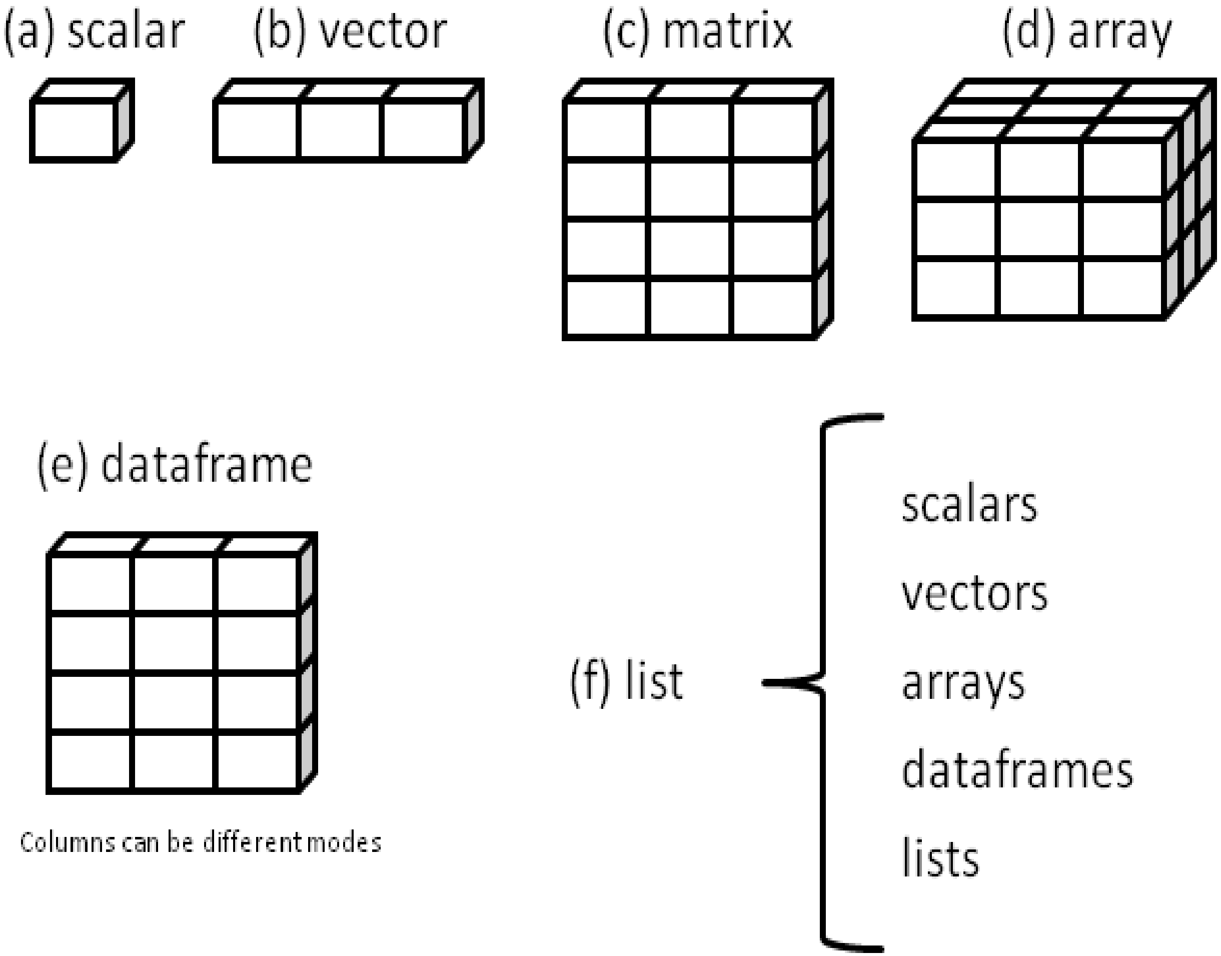
CREATING A DATASET

Topics

- R data structures
  - vectors, matrices, arrays,
    data frames, factors, lists

- Data Input
  - text file
    Excel
    Stat packages (SAS, SPSS, Stata)
    DBMS

R Data Structures



Vectors

One dimensional arrays

```
a <- c(1, 2, 5, 3, 6, -2, 4)
b <- c("one", "two", "three")
c <- c(TRUE, TRUE, TRUE, FALSE, TRUE, FALSE)
```

Vectors(2)

Identifying elements

```
a <- c(1, 2, 5, 3, 6, -2, 4)

a[3]
[1] 5

a[c(1, 3, 5)]
[1] 1 5 6

a[2:6]
[1]  2  5  3  6 -2
```

Matrices

Two dimensional arrays where each element has same mode (numeric, character, or logical)

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

Creating a Matrix

```
y <- matrix(1:20, nrow=5)
y
```

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

ncol=4 would work too

Creating a Matrix

```
y <- matrix(1:20, nrow=5, byrow=TRUE)
y
```

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

Creating a Matrix

```
data <- c(3615, 365, 2212, 3624, 6315, 4530)
rnames <- c("Alabama", "Alaska", "Arizona")
cnames <- c("Population", "Income")
y <- matrix(data, ncol=2,
                dimnames=list(rnames, cnames))
```

```
        Population Income
Alabama       3615   3624
Alaska         365   6315
Arizona       2212   4530
```

Using Matrix Subscripts

```
x <- matrix(1:10, nrow = 2)
x
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

```
x[2, ]
[1]  2  4  6  8 10
x[, 2]
[1] 3 4
x[1, 4]
[1] 7
x[1, c(4, 5)]
[1] 7 9
```

Data frame

- Rectangular array of data
- More general than a matrix - different columns can contain different modes of data (numeric, character, etc.)
- Similar to datasets in SAS, SPSS, and Stata

mydata <- data.frame(*col1*, *col2*, …, *coln*)

Creating a data frame

```
ptID      <- c(111, 208, 113, 408)
age       <- c(25, 34, 28, 52)
diabetes  <- c("Type1", "Type2", "Type1", "Type1")
status    <- c("Poor", "Improved", "Excellent", "Poor")
ptdata    <- data.frame(ptID, age, diabetes, status)
ptdata
```

```
       ptID age diabetes    status
1       111  25    Type1      Poor
2       208  34    Type2  Improved
3       113  28    Type1 Excellent
4       408  52    Type1      Poor
```

Specifying elements of a data frame

```
ptdata[1:2]
        ptID   age
1       111     25
2       208     34
3       113     28
4       408     52
```

```
ptdata[c("diabetes","status")]
 diabetes    status
1   Type1      Poor
2   Type2      Improved
3   Type1      Excellent
4   Type1      Poor
```

```
ptdata$age
[1] 25 34 28 52
```

Specifying elements of a data frame

```
ptdata[1:2]
        ptID  age
1       111    25
2       208    34
3       113    28
4       408    52
```

```
ptdata[2:3, 1:2]
        ptID  age
2       208    34
3       113    28
```

```
ptdata[c(1,3), 1:2]

        ptID  age

1       111    25

3       113    28
```

With

```
summary(mtcars$mpg)
plot(mtcars$mpg, mtcars$disp)

with(mtcars, {
    summary(mpg)
    plot(mpg, disp)
})
```

Factors

- Data structure specifying categorical (nominal) or ordered categorical (ordinal) variables
- Tells R how to handle that variable in analyses
- Very important and misunderstood
- Any variable that is categorical or ordinal should usually be stored as a factor.

Factors (2)

```
ptdata$sex <- c(1, 1, 2, 5)

ptdata$sex <- factor(sex, levels=c(1, 2),
     labels=c("Male", "Female"))
```

associates 1=Male, 2=Female

Treats sex as a categorical variable in all analyses

What happens to sex=5?

Factors (2)

```
ptdata$status <- c(1, 2, 3, 1)

ptdata$status <- factor(ptdata$status,
       ordered=TRUE,
       levels=c(1, 2, 3)
       levels=c("Poor", "Improved", "Excellent")
```

associates 1=Poor, 2=Improved, 3=Excellent
Treats status as an ordinal variable in all analyses

Lists

- Ordered collection of objects (components)
- Many important functions return lists

```
mylist <- list(name1=object1, name2=object2, …)
```

List Example

```
g <- "My First List"
h <- c(25, 26, 18, 39)
j <- matrix(1:10, nrow = 5)
k <- c("one", "two", "three")

mylist <- list(title = g,
               ages = h,
               mymatrix = j,
               mystrings = k)
```

mylist

```
$title
[1] "My First List"

$ages
[1] 25 26 18 39

$mymatrix
     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10

$mystrings
[1] "one"   "two"   "three"
```

List Example

```
mylist[[2]]
[1] 25 26 18 39
```

```
mylist[["ages"]]
[1] 25 26 18 39
```

```
mylist$ages
[1] 25 26 18 39
```

mylist

```
$title
[1] "My First List"

$ages
[1] 25 26 18 39

$mymatrix
     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10

$mystrings
[1] "one"   "two"   "three"
```

List Example

```
mylist$ages[2]
[1] 26

mylist[[2]][2]
[1] 26

 mylist$mymatrix[2,2]
 [1] 7
 mylist[[3]][,2]
 [1]  6  7  8  9 10
```

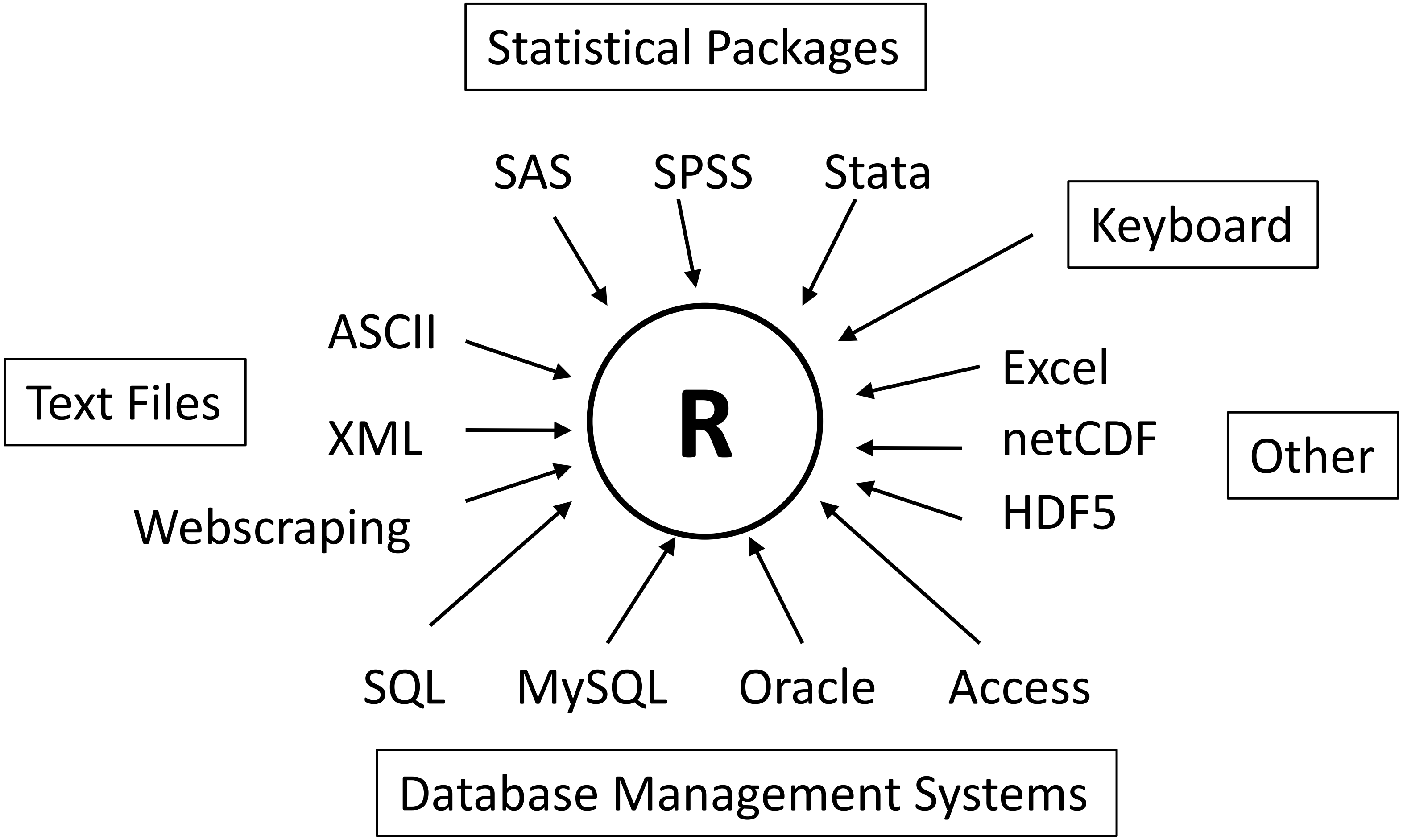
```
mylist
$title
[1] "My First List"

$ages
[1] 25 26 18 39

$mymatrix
     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10

$mystrings
[1] "one"   "two"   "three"
```

Data Input


Statistical Packages
SAS
SPSS
Stata
Keyboard
ASCII
Text Files
XML
Webscraping
R
Excel
netCDF
HDF5
Other
SQL
MySQL
Oracle
Access
Database Management Systems

Import from Delimited Text File

First row are variable names

```
library(readr)

# comma separated values
mydataframe <- read_csv("file")

# tab separated values
mydataframe <- read_tsv("file")

# semicolon separated values
mydataframe <- read_csv2("file")
```

option:
col_names=FALSE

Import from Delimited Text File

First row are variable names

```
library(readr)

# space delimited values
mydataframe <- read_table("file")

# like read.table(), it allows any number
# whitespace characters between columns, and
# the lines can be of different lengths
mydataframe <- read_table2("file")
```

option:
col_names=FALSE

Import from Delimited Text File

```
library(readr)

# more control over reading file
mydataframe <- read_delim("file",
             delim,
             col_names = TRUE,
             na = c("", "NA"),
             skip = 0,
             n_max = Inf)
```

Importing from Excel

```
# install.packages(readxl)
library(readxl)
df <- read_excel("myfile.xlsx", 1)
```

Importing from Stat Packages

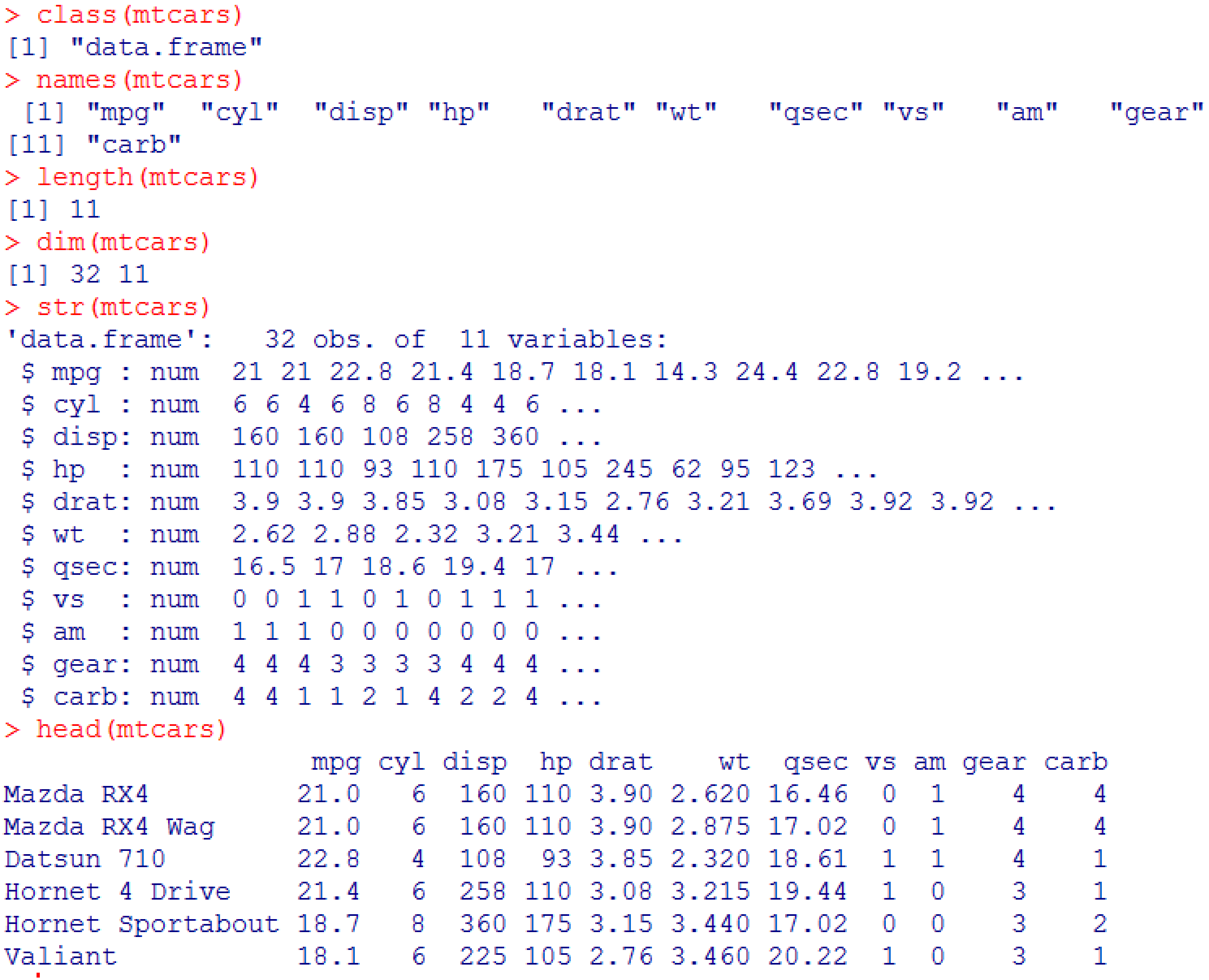
```
# install.packages(haven)
library(haven)
# sas
df <- read_sas("myfile.sas7bdat")
# spss
df <- read_sav("myfile.sav")
# stata
df <- read_stata("myfile.dta")
```

Accessing DBMS

- R can access MS SQL Server, MS Access, MySQL, Oracle, PostgreSQL, DB2, Sybase, Teradata, SQLite, …
- One of the best ways to deal with large datasets

Functions for working with objects

Function	Action
length(object)	number of elements/components
dim(object)	dimensions of an object
str(object)	structure of an object
class(object)	class or type of an object
names(object)	names of components in an object
c(object, object,...)	combines objects into a vector
cbind(object, object, ...)	combines objects as columns
rbind(object, object, ...)	combines objects as rows
object	prints the object
head(object)	list the first part of the object
tail(object)	list the last part of the object
rm(object)	delete an object



```
> class(mtcars)
[1] "data.frame"
> names(mtcars)
 [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
[11] "carb"
> length(mtcars)
[1] 11
> dim(mtcars)
[1] 32 11
> str(mtcars)
'data.frame':   32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Note for programmers

- **The period (.) has no special significance in object names**.
  However, the dollar sign ($) has a somewhat analogous meaning, identifying the parts of an object.
  For example, A$x refers to variable x in data frame A.
- **R does not provide for multi-line or block comments**.
  You must start each line of a multi-line comment with #. For debugging purposes, you can also surround code that you want the interpreter to ignore with the statement if(0){…}.
- **R does not have scalar values**.
  Scalars are represented as one element vectors.

Note for programmers

- **Variables cannot be declared**.
  They come into existence on first assignment.
- **Assigning a value to a non-existent element of a vector, matrix, array, or list will expand that structure to accommodate the new value.**

  ```
  x <- c(2, 6, 4)
  x[7] <- 10
  x
  [1]  2  6  4 NA NA NA 10
  ```

  The vector x has expanded from 3 elements to 7 elements through the assignment.

  x <- x[1:3] would shrink it back to three elements again.
- **Indices in R start at 1, not at zero**.
  In the vector above, x[1] is 2.

DATA MANAGEMENT

Topics

- arithmetic and logical operators
- creating, recoding, renaming variables
- missing values and data values
- type conversions
- sorting, merging, subsetting
- reshaping datasets

Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
^ or **	Exponentiation

Logical Operators

Operator	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

Salaries dataset

data(Salaries, package="car")

names(Salaries)

```
[1] "rank"          "discipline"    "yrs.since.phd" "yrs.service"  
[5] "sex"           "salary"
```

head(Salaries)

```
       rank discipline yrs.since.phd yrs.service  sex salary
1      Prof          B            19          18 Male 139750
2      Prof          B            20          16 Male 173200
3  AsstProf          B             4           3 Male  79750
4      Prof          B            45          39 Male 115000
5      Prof          B            40          41 Male 141500
6 AssocProf          B             6           6 Male  97000
```

Add new variables to a data frame

```
experience <- (yrs.since.phd + yrs.service)/2    # fails

experience <- (Salaries$yrs.since.phd +
          Salariesyrs.service)/2    # doesn't fail but…

Salaries$experience <- (Salaries$yrs.since.phd +
          Salaries$yrs.service)/2    # works

Salaries <- transform(Salaries, experience =
           (yrs.since.phd + yrs.service )/2) # better
```

Using the transform() function

```
Salaries <- transform(Salaries,
        logSalary = log(salary),
        experience = ( yrs.since.phd + yrs.service)/2,
        discipline = factor(discipline, levels=c("A", "B"),
            labels=c("Theoretical", "Applied")),
        salaryCat = cut(salary,
            quantile(salary, probs=c(0, .33, .66, 1)),
            labels=c("low", "med", "high"))
)
```

Renaming variables in a data frame

```
names(Salaries)
[1] "rank"          "discipline"    "yrs.since.phd" "yrs.service"
[5] "sex"        "salary"
```

```
names(Salaries)[3:4] <- c("yrs.post.phd",
                          "yrs.of.service")
```

```
names(Salaries)
[1] "rank"     "discipline"   "yrs.post.phd" yrs.of.service"
[5] "sex"   "salary"
```

Missing values

- coded as NA (no quotation marks)
- test with is.na()
- x == NA doesn't work

Working with missing values

```
data(sleep, package="VIM")
head(sleep)
head(is.na(sleep)
colSums(is.na(sleep))
colMeans(is.na(sleep))
newSleep <- na.omit(sleep)
```

Recoding to missing values

hypothetical example

-
  ```
  df$age <- ifelse(df$age == 99, NA, df$age)
  ```

-
  ```
  df$age <- ifelse(df$age %in% c(99, 999, -1),
                   NA, df$age)
  ```

- delete all missing values
  ```
  newdf <- na.omit(df)  # listwise deletion
  ```

Sorting data frames

```
index <- order(Salaries$rank, Salaries$salary)
Salaries  <- Salaries[index, ]
```

- order() returns a permutation which rearranges its argument into ascending or descending order
- default is ascending (reverse with - sign)
- note the comma!

```
index <- order(Salaries$rank, -Salaries$salary)
newdf <- Salaries[index, ]

head(newdf)
```

```
        rank discipline yrs.since.phd yrs.service    sex salary
238 AsstProf          A             7           6 Female  63100
227 AsstProf          A             3           1   Male  63900
65  AsstProf          B             4           3   Male  68404
241 AsstProf          A             5           3   Male  69200
235 AsstProf          A             8           3   Male  69700
50  AsstProf          B             1           1   Male  70768
```

Merging data frames (horizontally)

Use merge()

```
dataframeC <- merge(dataframeA, dataframeB,
    by ="ID")

dataframeC <- merge(dataframeA, dataframeB,
     by=c("ID","Country"))
```

Merging data frames (vertically)

Use rbind()

dataframeC <- rbind(*dataframeA*, *dataframeB*)

both data frames must have save variables (but don't have to be in same order).

Subsetting a data frame

Selecting (excluding variables)

```
df <- Salaries[c("rank", "sex", "salary")]
df <- Salaries[c(1, 5, 6)]
df <- df[-c(2, 3, 4)]
```

Subsetting a data frame

Selecting (excluding) observations

```
newdata <- Salaries[1:5, ]
```

```
newdata <- Salaries[Salaries$sex=="Female"
                    & Salaries$salary > 100000, ]
```

Subsetting a data frame

Selecting observations/variables using subset()

```
newdata <- subset(Salaries,
   salary >= 200000 | salary <  60000,
   select=c(sex, rank, salary))


newdata <- subset(Salaries,
   sex=="Female" & salary > 60000,
   select=yrs.since.phd:salary)
```

Aggregating data

- aggregate(*x*, *by*, *FUN*)
  - *x* is the data object to be collapsed
  - *by* is a list of variables that will be cross to form new observations
  - *FUN* is a scalar function used to calculate summary statistics that will make up the new observation values

Aggregate example

```
aggdata <- aggregate(mtcars[c("mpg", "disp", "wt")],
                     by=list(cylinder=mtcars$cyl, gears = mtcars$gear),
                     FUN=mean, na.rm=TRUE)
 aggdata
```

```
  cylinder gears  mpg disp   wt
1        4     3 21.5  120 2.46
2        6     3 19.8  242 3.34
3        8     3 15.1  358 4.10
4        4     4 26.9  103 2.38
5        6     4 19.8  164 3.09
6        4     5 28.2  108 1.83
7        6     5 19.7  145 2.77
8        8     5 15.4  326 3.37
```

Subgroup processing

- by(*data*, *INDICES*, *FUN*)
  - where *data* is a data frame or matrix
  - *INDICES* is a categorical variable or list of categorical variables that define the groups
  - *FUN* is an arbitrary function

Descriptive statistics by subgroup

by(mtcars[c("mpg", "hp")], mtcars$am, summary)

```
mtcars$am: 0
      mpg               hp
 Min.   :10.4   Min.   : 62
 1st Qu.:14.9   1st Qu.:116
 Median :17.3   Median :175
 Mean   :17.1   Mean   :160
 3rd Qu.:19.2   3rd Qu.:192
 Max.   :24.4   Max.   :245
------------------------------------------------------------
mtcars$am: 1
      mpg             hp
 Min.   :15   Min.   : 52
 1st Qu.:21   1st Qu.: 66
 Median :23   Median :109
 Mean   :24   Mean   :127
 3rd Qu.:30   3rd Qu.:113
 Max.   :34   Max.   :335
```

Reshaping wide to long

```
library(reshape2)
jtrain_long <- reshape(jtrain, idvar = "id",
                    varying = list(c("re74","re75", "re78"),
                                 c("unem74","unem75", "unem78")),
                     v.names = c("re", "unem"),
                     timevar = "year" ,
                     times = c(74, 75, 78),
                     direction = "long")
```

Reshaping long to wide

```
library(reshape2)
jtrain_wide<- reshape(jtrain_long,
                      idvar = "id",
                      v.names = c("re", "unem"),
                      timevar = "year",
                      direction = "wide")
```

Applying function to columns and rows

apply(dataframe, index, function, options)

```
apply(mtcars, 1, mean)  # row means
apply(mtcars, 2, mean, na.rm=TRUE) # column means
```