```
rmarkdown::render("xx.Rmd")
rows: rbind(df1, df2)
cols: cbind()
do.call(rbind,
  lapply(sprintf("covid-0%d.csv", 1:6), read.csv) )
lapply(x, fun)     return list
sapply(x, fun)     return vector/matrix
tapply(x, index, fun)
aggregate(deceased, list(years), mean)   return df
  group deceased by years and give mean
```

```
  Group.1   x
1      11 0.5
2      12 0.5
3      13 1.0
```

```
read.csv("xx.csv", nrow=10, colClasses=c(),
         header=T/F, col.names=c() when no header)
readLines("xx.csv", n=10)
  return vector of 10 lines (each is a character value)

subset(df, Month == 1)
barplot(table(df$Month))
plot(density(df$ArrDelay, na.rm=TRUE))
hist(df$DepTime)
par(mfrow=c(2,1)) 2 rows, 1 col
abline(fit, col="red")

b <- cut(df$DepTime, breaks=seq(0, 100, 10))
  cut: convert numeric to factor
o <- order(xTest)
lines(xTest[o], predGLM[o])
read.fwf("fwf.txt", widths=c(), skip=7, colClasses=c())
library(RSQLite)
con <- dbConnect(SQLite(), dbname="surftemp.sqlite3")
dbGetQuery(con, "SELECT AVG(DepDelay) FROM surftemp")
dbDisconnect(con)
```

Modelling

```
df <- data.frame(deceased, years)[!is.na(years), ]
N <- nrow(df)
labels <- rep(1:10, length.out=N)
groups <- sample(labels)
logloss <- function(i, formula) {
    testSet <- groups == i
    trainSet <- groups != i
    fit <- glm(formula, family="binomial",
               data.frame(x=df$years[trainSet],
                          y=df$deceased[trainSet]),
               na.action=na.exclude))
    pred <- predict(fit, data.frame(x=df$years[testSet]),
                    type="response")
    y <- df$deceased[testSet]
    -mean(y*log(pred) + (1 - y)*log(1 - pred)) # logloss
    sqrt(mean((pred - y)^2))                   # RMSE
}
mean(sapply(1:10, logloss, y ~ x))    # y ~ 1
  mean(unlist(mclapply(1:10, logloss, y ~ x, mc.cores=10)))
pSimple <- mean(df$deceased)
props <- aggregate(df$deceased, list(df$years), mean)
plot(props)                                    # circles
pred <- predict(glm(deceased ~ years, df, family="binomial"),
                type="response")
lines(sort(df$years), pred[order(df$years)], col="red")
abline(h=pSimple, col="grey")
```

Data Formats

```
JSON [{}]      => R data frame (nested)  flatten()
R named vector => JSON [] (names ignored)
R list         => JSON [[]]
R named list   => JSON named {}
R matrix       => JSON array of rows [[]]
R data frame   => JSON array of rows [{}]

JSON must use "" for fields
library(jsonlite)    fromJSON("")
```

```
[{ "x": 11,
   "y": { "a": 1,
          "b": "A" }
},
{ "x": 12,
  "y": { "a": 2,
         "b": "B" }
}]
```

```
  x y.a y.b
1 11   1   A
2 12   2   B
```

```
library(mongolite)
m <- mongo("COVID")
m$find(query='{"location": {"$ne": "World"},
               "pop": {"$gt": 1000}}',
       fields='{"_id": 0, "data.x": 1}',
       sort='{"pop": -1}', limit=10)
  -1 means descending order
  return data frame

library(xml2)    pets <- read_xml("pets.xml")   read_html()
xml_text(xml_find_all(pets, "//row[pets_adopted > 100]/month"),
         trim=TRUE)
f <- function(c) {
  country <- xml_find_first(c, "ancestor::country")
  xml_attr(country, "name") }
countryNames <- sapply(cities, f)
```

```
is.finite() not Inf, NA, NaN
is.na()     if is NA, NaN
```

```
unlist()
as.Date(d,
format="%d.%m.%Y")
identical()
summary(covid$num)
.Last.value
```

```
system("wc -l xx.csv")   source("boot.R") run R file in R
ls linux\ prosper\ script.sh
ls -latr
  -l long format
  -a show hidden file/folder
  -t (modified time) recent first
  -r reverse order
mkdir    chmod  modify permissions
head -10   tail -10
ls > ./ls1.txt && ls .. > ./ls2.txt
  && means if previous succeed, run next
ls -lh $(find ./ -name "*.csv" | grep -v [0-9].csv)
  -v exclude matching items

wc $(ls) count each file
ls | wc  count output of ls
wc -l line  -w word  -c character
wc counts white spaces and new lines (as #characters)
wc keeps only one line of file in memory at a time to count
  R needs to load whole thing to count

for i in /Data/*.csv
do
  wc $i
  filename=$(basename $i)
  shuf -n100 $i > $filename    randomly select 100 rows
done
tar cvfz ../xx.tar.gz .  zip current repo
tar ztvf ../xx.tar.gz    list content
tar zxvf ../xx.tar.gz    extract content
```

Text Processing

```
                match LHS
RHS <- gsub("^.+- *", "", age)
paste("","","")  paste0(1:10, ".csv")
find /course/ -name "*.csv" -size +1G
grep is case-sensitive
grep -c 'Spring' *.csv  -c #lines only   -R recursively
grep '^[^0-9]' xx.csv | wc -l
  #lines not start with a digit
grep("[a-z]", df$age)  return indices where item match pattern

awk -F, -e '{print(NF)}' xx.csv     '{if (NR > 1) print($1)}'
  -F\"   '{print}'   'NF % 2 == 0'   'NR == 1 {print(NF)}'
  '/"M."/ {print}'  M plus one more character
NR current row id   NF #fields
$0 whole line       $1 first field
default delimiter for fields is whitespaces
```

Memory Usage

```
du -sh . size of current repo  (in KB)
df -h    disk usage            (in KB)
free -h  RAM usage             (in KB)
top      monitor RAM
who      show users

logical, integer           4 bytes   48 bytes overhead
numeric, character pointer 8 bytes    object.size()

class(1:10 + 1) "numeric"  integer can easily become numeric
factor(rep("a", 10000)) 40496 bytes  each integer represents
the factor level

R will only actually copy when modified
x<-seq(1e6) only start&end    y<-x+1 actual values generated

gc() Vcells
gc(reset=TRUE) set max used to current used    rm()

library("profmem")
options(profmem.threshold = 100)   profmem({}, threshold=100)
p <- profmem(f())  return data frame
p   print(p, expr = FALSE)   total(p)   p[p$bytes>100, ]

lm() add one factor variable with 5 levels = add 4 columns
(exclude first level)

/usr/bin/time -f "%M" Rscript -e 'invisible(NULL)'  (in KB)
```

```
sqrt(0)   0       moderate skew
log(0)    -Inf    greater skew  (removes 0 - problem)
```

```
d/- rwx  rwx  rwx  pmur002   4000      Oct 23
   user group other      size(bytes) modification date
```

```
XPath: /a/b[1] first b  /a[@lang='en']  /a[@year>2000]
/a/b[last() - 1]  second last b
/a/b[contains(@id, 'paul')]
/a/b/following-sibling::*  at same level, after b
//c[ancestor::b@lang = 'en']

library(httr)
read.csv("http…")   download.file("http…", "xx.csv")
res <- GET("http…", authenticate("user", "passwd"))
headers(res)$`content-type`
content(res, as="text")    turn binary json into text

library(rvest)
html_table(content(res))  return list of data frames
htmls <- lapply(urls, GET)
df <- do.call(rbind, lapply(htmls,
                  function(x) html_table(content(x))[[1]]))
fromJSON(content(x, as="text"))

library(xml2)   read_xml("")
read_html("http…") ⇨ content(GET())      xml_structure()
xml_find_all( content(GET("http…xml")), "/*" )
------------------------------------------------ Large Data
Large data solution Trade Offs:   correctness
easy to write & understand, simple, flexible, consistent
Always develop code on a small subset of data.

use bigger machine
make data smaller
  query from database
  store data compactly
    Matrix::sparseMatrix(1:10, 1:10, x=1)
    MatrixModels::model.Matrix(y~x, df, sparse=TRUE)
  sample data
  avoid R's copying semantics
  streaming data
    con <- file("numbers.csv", "r")  sum <- 0
    for (i in 1:10) {
        y <- scan(con, sep=",", nlines=1000)
        sum <- sum + sum(y)
        gc() }
    library(biglm)
    df <- read.table(con, sep=",", nrows=1000)
    streamFit <- biglm(V1 ~ V2, df)
    for (i in 1:10) {
        df <- read.table(con, sep=",", nrows=1000)
        streamFit <- update(streamFit, df)
    }
    streamFit <- bigglm(y~x, dataFun, family=binomial())
    close(con)
  leave data on disk
    library(bigmemory) use a connection to the file
use not-R tools (shell & sql)

library(data.table)
  reduce amount of copying, faster to load data
  utilise multiple CPUs at a time (parallel)
dt <- fread("xx.csv", sep=",")
dt[i, j (update/create), by]
dt[grep("[0-9] *- *[0-9]", age),
   monthDelay := mean(DepDelay, na.rm=TRUE), by=Month]

> /dev/null  null file (discard output)
------------------------------------------------ Efficiency
Ctrl c   Ctrl z (put into sleep), ps, kill -9
fg  continue running

system.time({})   replicate()
microbenchmark::microbenchmark(f(x), times=5)

Rprof(filename) start profiling, write to file
Rprof(NULL)     stop

library(profvis)
profvis({})    graphic, stochastic(sample on call stack)

pre-allocate memory
vectorisation
binary version file   readRDS("xx.rda")
use faster functions & data structures

time -p  \     (shell)    real user sys
wc -l xx.csv
```

```
library(parallel)  detectCores() ------------------------ Parallel
mclapply(1:3, function(i) source("boot.R"), mc.cores=3)
  using source() will return list of list
    result[[1]]$value  $visible
mclapply(1:3, function(i) myboot(), mc.cores=3)
  return list (length of 1:3)
mclapply(rep(2000, 3), myboot, mc.cores=3)

mc*() fork session (share RAM read-only)
  - not available on Windows
  + very quick to start worker session
  + less communication from master to worker
-------------------------
cl <- makeCluster(4)
clusterExport(cl, c("myboot", "df"))
parLapply(cl, rep(2000, 3), myboot)  return list
parLapply(cl, 1:3, function(i) myboot())
stopCluster(cl)
  - slower to start worker session (also harder)
  + works everywhere
  + worker session can run on remote machine
-------------------------
RNGkind("L'Ecuyer-CMRG")
  set.seed(123)  mclapply(1:5, coin, mc.cores=3)
  clusterSetRNGStream(cl, 123)  parLapply(cl, 1:5, coin)
we don't want each worker session choose their seed because they
might overlap halfway

Load balacing – run long job first
              - mc.preschedule=FALSE (parLapplyLB())

data.table  automatically runs in parallel
library(caret)  library(doParallel)
caret automatically runs in parallel given a foreach back-end
  foreach (i = values) %do% [or] %dopar% myboot(i)
  registerDoParallel(cores= / cl)
  tc <- trainControl(method="cv", number=10,
        classProbs=TRUE, summaryFunction=mnLogLoss)
  train(y~x,data=df,trControl=tc,method="glm",family="binomial")
    $results$logLoss
------------------------------------------------------- Distributed
Hadoop MapReduce             Rscript -e '' &   & run in background
  data transformation
  disk-based computation (slow)
  keeps calling R functions calculations
Spark
  data transformation & modelling
  in memory (RAM-based) computation
  not only shifts data around, but also do calculations within it
-------------------------
library(rmr2)  s<-to.dfs(1:10)    kv<-from.dfs(s)  kv$key kv$val
csv <- make.input.format("csv", sep=",",
                         stringsAsFactors=FALSE,
                         col.names=c())
oldwd <- setwd(tempdir())
result <- mapreduce(input=small.ints, verbose=FALSE,
                    input.format=csv,
                    map=function(k, v) keyval(v, 1)
                    reduce=function(k, vv) keyval(k, length(vv)) )
setwd(oldwd)
kv <- from.dfs(result)
-------------------------
library(sparklyr)  library(dplyr)
tbl <- spark_read_csv(sc, name="covid",
                      path="/course/COVID/Lab04/covid.csv",
                      header=FALSE,
                      columns=c(outcome="character",age="double"))

tbl <- copy_to(sc, df, "flights")
result <- tbl %>%
          group_by(tailnum) %>%
          summarise(count = n(), dist = mean(distance),
                    delay = mean(arr_delay)) %>%
          filter(count > 20, dist < 2000, !is.na(delay))
result                              # now Spark actually calculates
resultR <- collect(result)

fit <- tbl %>%
       filter(!is.na(dep_delay) & !is.na(dep_time)) %>%
       ml_linear_regression(dep_delay ~ dep_time)
       [or] ml_logistic_regression    [values cannot contain null]

tbl [or] df %>% mutate(deceased = o %in% c()) %>% count(deceased)
df %>% lm(y ~ x, .)
```