R Programming and Data Analysis
Intermediate R Programming

Introduction

1. Numerical tools
    a. Optimization
    b. Integration
    c. Root finding
    d. Solving systems of equations
2. Debugging
3. Reading and Writing Data
    a. Saving and loading workspaces
    b. CSV files
    c. HDF5 files
    d. SAS files
    e. Databases
    f. Web services
    g. Web scraping
    h. Big datasets
4. Data Manipulation with dplyr

Numerical Tools

Numerical Tools

. . . Demonstration . . .

(See `numerical.Rmd`)

Debugging

Debugging

- Debugging is an integral part of writing nontrivial programs.
- It is rare to write a perfect program on the first attempt.
- When running a new program, we might see
  a. errors or warnings.
  b. an obviously incorrect result (e.g. `NA` or a negative probability).
  c. a subtly incorrect result.
  d. a result whose correctness we are not certain of.
  e. problems only for certain inputs or random draws.
- Some basic debugging techniques can help us track down the sources of these issues.

Debugging

- Develop code "interactively".
  a. Write a few lines of code, run them, and examine the output.
  b. Prevents some obvious errors, and quickly get a working program.
- Use print statements to report important values.
  a. Especially useful for longer programs that cannot be run interactively.
  b. For example, to see how an optimization is working, we can print the result before returning it.

```
f <- function(x) {
    fx <- 1 - sum(x^2)
    cat("x = (", paste(round(x, 4), collapse = ","), "),
        fx =", fx, "\n")
    return(fx)
}
```

```
> x0 <- rep(0, 5)
> res <- optim(x0, f, control = list(fnscale = -1))
...
x = ( 0.005,0.051,0.011,0.011,0.011 ), fx = 0.997011
x = ( 0.022,0.0244,-0.0916,0.0484,0.0484 ), fx = 0.985845
x = ( 0.0055,0.0061,0.0521,0.0121,0.0121 ), fx = 0.9969253
...
```

Debugging

- Sometimes it is helpful to leave print statements in program after debugging is complete.
- These functions are useful for basic logging

```
printf <- function (msg, ...)  {
    cat(sprintf(msg, ...))
}

logger <- function (msg, ...) {
    sys.time <- as.character(Sys.time())
    cat(sys.time, "-", sprintf(msg, ...))
}
```

```
> printf("Convergence Status: %d\n", res$convergence)
Convergence Status: 0
> logger("Starting %d reps of MCMC\n", R)
2016-07-25 15:55:01 - Starting 1000 reps of MCMC
```

- The `log4r` package is a bit more sophisticated. Supports multiple logging levels (`INFO`, `WARN`, `ERROR`, etc).

Debugging

- R has an interactive debugger to step through running programs.
- Can ask R to start debugger when a particular function is called in the current session. This can be any function in R, not only ours.

```
> debug(optim)
> x0 <- rep(0, 5)
> res <- optim(x0, f, control = list(fnscale = -1))
debugging in: optim(x0, f, control = list(fnscale = -1))
debug: {
... [optim function contents are shown] ...
}
```

- Now program is paused and we can use regular R commands.

```
Browse[2]> ls()
[1] "control" "fn" "gr" "hessian" "lower" "method" "par" "upper"
Browse[2]> print(control)
$fnscale
[1] -1

Browse[2]>
```

- Can step to the next line, the next breakpoint, or stop debugger.
- Changes made to workspace may be discarded after exiting debugger.
- Use `undebug(optim)` to stop watching `optim` calls.

Debugging

- Another way to invoke the debugger is to put a `browser` call in your program. `R` starts the debugger when it encounters this statement.

```
f <- function(x) {
    z <- t(x) %*% x
    browser()
    return(z)
}
```

```
> x <- c(1,2,3)
> f(x)
Called from: f(x)
Browse[1]> ls()
[1] "x" "z"
Browse[1]> x
[1] 1 2 3
Browse[1]> z
     [,1]
[1,]   14
Browse[1]> Q
```

- For more information about debugging in `R`, see `www.stats.uwo.ca/faculty/murdoch/software/debuggingR`.

Reading and Writing Data

Objects and the Workspace

- The entities that `R` creates and manipulates are known as *objects*.
- These may be variables, arrays of numbers, character strings, functions, or more general structures built from such components.
- The collection of objects currently stored in `R` is called the *workspace*.
- The workspace can be saved to disk, and loaded back into `R` in a new or existing session.
- Workspace does not store packages that were loaded — we have to reload them ourself.

Objects and the Workspace

- The commands `object` or `ls` can be used to display the objects currently loaded in `R`.

- Use the function `rm` to remove objects from your workspace.

```
> x <- rnorm(5)
> x
[1] -0.8287666  0.8377261 -0.3695485  1.3661922  1.8287291
> y <- x + 10
> y
[1]  9.171233 10.837726  9.630451 11.366192 11.828729
> objects()
[1] "x" "y"
> ls()
[1] "x" "y"
> rm(list = ls(all = TRUE))
> ls()
character(0)
```

Note that `character(0)` represents a string vector of length zero.

Saving the Workspace

- R designates a directory on the computer to be the "current working directory". This is where output files will be written by default.

```
> getwd()
[1] "/path/to/here"
```

- To get/set the working directory with getwd/setwd

```
> setwd("/path/to/here")
> getwd()
[1] "/path/to/here"
```

- When we exit a session, R asks if we wish to save the workspace.
  - If we say "yes", a binary file .RData will be created; this contains all the objects in our workspace (therefore file might be large).
  - A text file .Rhistory may be also be created; contains our history of commands.
  - Next time we launch R from that directory, our workspace will return to the same state.
  - We can also load the state files manually.

Saving the Workspace

We can save the workspace at any time (not just when quitting), and using any filename we wish, using `save.image`.

```
> x <- c(1,2,3)
> A <- matrix(1:9, nrow = 3, ncol = 3)
> B <- diag(10)
> ls()
[1] "A" "B" "x"
> save.image(file = "myworkspace.Rdata")
```

We can also save specific objects from the workspace using `save`.

```
> ls()
character(0)
> x <- 10
> y <- 11
> z <- 12
> save(list = c("x","y"), file = "myvariables.Rdata")
```

Loading the Workspace

Saved `R` data can be loaded at any time using `load`. Be aware that objects in your current environment may be overwritten.

```
> x <- 55
> y <- 77
> load("myworkspace.Rdata")
> ls()
[1] "A" "B" "x" "y"
> x
[1] 1 2 3
> y
[1] 77
```

CSV Files

- Recall the CO2 dataset.

```
> CO2
   Plant        Type  Treatment conc uptake
1    Qn1      Quebec nonchilled   95   16.0
2    Qn1      Quebec nonchilled  175   30.4
...
83   Mc3 Mississippi    chilled  675   18.9
84   Mc3 Mississippi    chilled 1000   19.9
```

- Write it to a CSV file.

```
> write.table(CO2, file = "CO2.csv", sep = ",")
> getwd()
[1] "/path/to/file"
```

- Check contents of the file.

```
[username@localhost ~]$ cat /path/to/file/CO2.csv
"Plant","Type","Treatment","conc","uptake"
"1","Qn1","Quebec","nonchilled",95,16
"2","Qn1","Quebec","nonchilled",175,30.4
...
"83","Mc3","Mississippi","chilled",675,18.9
"84","Mc3","Mississippi","chilled",1000,19.9
```

CSV Files

- Read the file into R

```
> dat <- read.table("/path/to/file/CO2.csv", sep = ",")
> print(dat)
   Plant        Type  Treatment conc uptake
1    Qn1      Quebec nonchilled   95   16.0
2    Qn1      Quebec nonchilled  175   30.4
...
83   Mc3 Mississippi    chilled  675   18.9
84   Mc3 Mississippi    chilled 1000   19.9
```

R: Data Input

read.table {utils} R Documentation

Data Input

Description

Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.

Usage

```
read.table(file, header = FALSE, sep = "", quote = "\"'",
           dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),
           row.names, col.names, as.is = !stringsAsFactors,
           na.strings = "NA", colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill = !blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#",
           allowEscapes = FALSE, flush = FALSE,
           stringsAsFactors = default.stringsAsFactors(),
           fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)

read.csv(file, header = TRUE, sep = ",", quote = "\"",
         dec = ".", fill = TRUE, comment.char = "", ...)

read.csv2(file, header = TRUE, sep = ";", quote = "\"",
          dec = ",", fill = TRUE, comment.char = "", ...)

read.delim(file, header = TRUE, sep = "\t", quote = "\"",
           dec = ".", fill = TRUE, comment.char = "", ...)
```

CSV Files

- The readr package provides more sophisticated file parsing tools. Often faster than the usual read.table, read.csv, etc.
- We will generate a large CSV to demonstrate.

```
library(readr)

n <- 5000000
y <- sample(c("YES", "NO"), size = n, replace = TRUE, prob = c(0.1,
     0.9))
DF <- data.frame(
    x = rnorm(n),
    y = y,
    z = rpois(n, 10)
)
write.table(DF, file = "mydata.dat", sep = ",", row.names = FALSE)
```

- This produces a ~124 MB file.

CSV Files

```
> system.time(dat1 <- read.csv("mydata.dat"))
   user  system elapsed
 31.166   8.792  46.799
> head(dat1, 3)
           x   y  z
1  2.4157666  NO  5
2 -0.1859725  NO  8
3 -0.3424828 YES  7
```

```
> system.time(dat2 <- read_csv("mydata.dat"))
Parsed with column specification:
cols(
  x = col_double(),
  y = col_character(),
  z = col_integer()
)
|======================================================| 100%  123 MB
   user  system elapsed
  7.890   1.702  10.677
> head(dat2, 3)
# A tibble: 6 x 3
           x     y      z
       <dbl> <chr> <int>
1  2.4157666    NO      5
2 -0.1859725    NO      8
3 -0.3424828   YES      7
```

HDF5 Files

- HDF5 (`www.hdfgroup.org/HDF5`) flexible library for storing and managing data.
- Portable file format with interfaces in `C/C++`, `Fortran`, `Python`, `R`, and others.
- An HDF5 file has a hierarchical format like a filesystem.
  1. *Groups* are like directories.
  2. *Datasets* are like files, but have a well-defined structure.
  3. *Attributes* are metadata which can be attached to groups and datasets.
- `rhdf5` is an `R` package for manipulating HDF5 files.

```
source("https://bioconductor.org/biocLite.R")
biocLite("rhdf5")
```

HDF5 Files

```
library(rhdf5)
h5createFile("mydata.h5")

y <- mtcars$mpg
X <- model.matrix(~ cyl + disp + hp, data = mtcars)
h5createGroup("mydata.h5","mtcars")
h5write(mtcars, "mydata.h5","mtcars/mtcars")
h5write(y, "mydata.h5","mtcars/y")
h5write(X, "mydata.h5","mtcars/X")

y <- CO2$conc
X <- model.matrix(~ Plant + Type + Treatment + uptake, data = CO2)
h5createGroup("mydata.h5","CO2")
h5write(mtcars, "mydata.h5","CO2/CO2")
h5write(y, "mydata.h5","CO2/y")
h5write(X, "mydata.h5","CO2/X")
```

```
> h5ls("mydata.h5")
    group   name       otype   dclass     dim
0       /    CO2   H5I_GROUP
1    /CO2    CO2 H5I_DATASET COMPOUND      32
2    /CO2      X H5I_DATASET    FLOAT 84 x 15
3    /CO2      y H5I_DATASET    FLOAT      84
4       / mtcars   H5I_GROUP
5 /mtcars      X H5I_DATASET    FLOAT  32 x 4
6 /mtcars mtcars H5I_DATASET COMPOUND      32
7 /mtcars      y H5I_DATASET    FLOAT      32
> H5close()
```

HDF5 Files

```
> library(rhdf5)
> h5read("mydata.h5","mtcars/y")
 [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8
[12] 16.4 17.3 15.2 10.4 10.4 14.7 32.4 30.4 33.9 21.5 15.5
[23] 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7 15.0 21.4

> h5read("mydata.h5","mtcars/mtcars")
    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
1  21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
2  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
...
31 15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
32 21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2

> h5read("mydata.h5","mtcars/X")
      [,1] [,2]  [,3] [,4]
 [1,]    1    6 160.0  110
 [2,]    1    6 160.0  110
...
[31,]    1    8 301.0  335
[32,]    1    4 121.0  109

> h5read("mydata.h5","mtcars/X", index=list(1:5,1:3))
     [,1] [,2] [,3]
[1,]    1    6  160
[2,]    1    6  160
[3,]    1    4  108
[4,]    1    6  258
[5,]    1    8  360
```

SAS Files

- `SAS` is a commercial software suite which is popular with data analysts, corporations, and government agencies.

- The main format for `SAS` files is a proprietary `sas7bdat` format.
  1. The `foreign` and `Hmisc` packages can read them, but require `SAS` installed on the computer.
  2. The `sas7bdat` and `haven` packages can read them without `SAS`.

- The `XPORT` format is intended to be more interoperable.
  1. Can be read by the `foreign` package.

- The `foreign` package can also read/write data for statistical packages such as `Minitab`, `S`, `SAS`, `SPSS`, `Stata`, `Systat`.

SAS Files

```
> library(haven)
> url1 <- "http://www.principlesofeconometrics.com/sas/yield.sas7bdat"
> yield <- read_sas(url1)
> class(yield)
[1] "tbl_df"     "tbl"        "data.frame"
> tail(yield)
# A tibble: 6 x 5
     YIELD     T     GROW     GERM   FLOWER
     <dbl> <dbl>    <dbl>    <dbl>    <dbl>
1 1.080829    34 1.354559 1.155823 1.208213
2 1.831250    35 1.005840 0.991480 0.939275
3 1.028031    36 1.288040 0.366613 0.810828
4 1.465217    37 0.828458 1.385182 0.698436
5 1.706897    38 0.772018 0.837972 0.586044
6 1.988593    39 1.052202 0.895763 1.111877
```

```
> url2 <- "http://www.principlesofeconometrics.com/sas/vote.sas7bdat"
> vote <- read_sas(url2)
> tail(vote)
# A tibble: 6 x 7
          STATE  VOTE INCOME SCHOOL URBAN NORTHEAST SOUTHEAST
          <chr> <dbl>  <dbl>  <dbl> <dbl>     <dbl>     <dbl>
1       Vermont     0 12.415   12.5   0.0         1         0
2      Virginia     0 14.579   12.4  65.6         1         0
3    Washington     0 14.962   12.7  71.1         0         0
4  WestVirginia     1 12.007   12.1  36.1         1         0
5     Wisconsin     1 15.064   12.5  63.0         0         0
6       Wyoming     0 14.784   12.6   0.0         0         0
```

SAS Files

```
> library(sas7bdat)
> yield <- read.sas7bdat(url1)
> at <- attributes(yield)
> names(at)
 [1] "names"         "row.names"     "class"         "pkg.version"
 [5] "column.info"   "date.created"  "date.modified" "SAS.release"
 [9] "SAS.host"      "OS.version"    "OS.maker"      "OS.name"
[13] "endian"        "winunix"
> at$SAS.host
[1] "WIN"
> at$SAS.release
[1] "9.0000M0"
> at$date.created
[1] "2008-05-13 17:02:32 EDT"
> at$date.modified
[1] "2008-05-13 17:02:32 EDT"
> str(at$column.info[[1]])
List of 11
 $ name  : chr "YIELD"
 $ offset: int 0
 $ length: int 8
 $ type  : chr "numeric"
 $ fhdr  : int 0
 $ foff  : int 0
 $ flen  : int 0
 $ label : chr "wheat yield, tonnes per hectare"
 $ lhdr  : int 0
 $ loff  : int 36
 $ llen  : int 31
```

Databases

- R can interact with both SQL databases (DBs) and NoSQL DBs by using appropriate packages.
- SQL DBs
  - Store data in a highly structured relational DB
  - Tables are optimized for storage and efficient merging. This usually requires careful planning.
  - Use SQL (Structured Query Language) to query and modify the DB.
  - Examples include MySQL, Oracle, PostgreSQL, SQLite, and SQLServer.
- NoSQL DBs
  - Less structured than relational DBs (e.g. key-value pairs), but more flexible.
  - Use an alternative query language to SQL.
  - Examples include MongoDB and Google BigTable.

Databases

- We will focus on SQL databases.
- The `DBI` package provides a generic interface to SQL DBs.
- Other packages build on `DBI` for specific database implementations. For example, `RMySQL`, `ROracle`, `RPostgreSQL`, `RSQLite`, `RSQLServer`.
- SQLite
  - A “self-contained, serverless, zero-configuration, transactional SQL database engine”.
  - Databases are stored in ordinary files.
  - Good for local storage in an application (e.g. storing website cookies in Firefox).

SQL Databases

```
> library(DBI)
> con <- dbConnect(RSQLite::SQLite(), "mydata.sqlite")

> dbListTables(con)
character(0)
> dbWriteTable(con, "mtcars", mtcars)
[1] TRUE
> dbListTables(con)
[1] "mtcars"

> dbListFields(con, "mtcars")
 [1] "row_names" "mpg"     "cyl"     "disp"    "hp"      "drat"    "wt"      "qsec"
 [9] "vs"        "am"      "gear"    "carb"
> dbReadTable(con, "mtcars")
                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
...
Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

SQL Databases

```
> res <- dbSendQuery(con, "select * from mtcars where cyl = 4")
> qry <- dbFetch(res)
> dbGetInfo(res)$fields
        name    Sclass type len
1  row_names character TEXT  NA
2        mpg    double REAL   8
3        cyl    double REAL   8
4       disp    double REAL   8
5         hp    double REAL   8
6       drat    double REAL   8
7         wt    double REAL   8
8       qsec    double REAL   8
9         vs    double REAL   8
10        am    double REAL   8
11      gear    double REAL   8
12      carb    double REAL   8

> dbGetRowCount(res)
[1] 11
> qry$row_names
 [1] "Datsun 710"     "Merc 240D"      "Merc 230"       "Fiat 128"
 [5] "Honda Civic"    "Toyota Corolla" "Toyota Corona"  "Fiat X1-9"
 [9] "Porsche 914-2"  "Lotus Europa"   "Volvo 142E"
> qry$mpg
 [1] 22.8 24.4 22.8 32.4 30.4 33.9 21.5 27.3 26.0 30.4 21.4
> dbClearResult(res)
[1] TRUE
```

SQL Databases

```
> res <- dbSendQuery(con, "update mtcars set mpg = -22 where cyl = 4")
> dbClearResult(res)
[1] TRUE
> dbReadTable(con, "mtcars")
                  mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4        21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag    21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710      -22.0   4 108.0  93 3.85 2.320 18.61  1  1    4    1
...
Volvo 142E      -22.0   4 121.0 109 4.11 2.780 18.60  1  1    4    2

> res <- dbSendQuery(con, "insert into mtcars
+ (row_names, mpg, cyl, disp, hp, drat, wt, qsec, vs, am, gear, carb)
+ values ('ABCD', 40.0, 4, 100.0, 200, 4.00, 1.5, 20.0, 1, 1, 5, 1)")
> dbClearResult(res)
[1] TRUE
> dbReadTable(con, "mtcars")
                  mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4        21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag    21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
...
Volvo 142E      -22.0   4 121.0 109 4.11 2.780 18.60  1  1    4    2
ABCD             40.0   4 100.0 200 4.00 1.500 20.00  1  1    5    1
> dbDisconnect(con)
[1] TRUE
```

Web Services

- Web services are collections of functions that can be called through the web by user programs.
- User programs call functions via Application Programmer Interface (API).
- A simple API, useful for querying, is REST (Representational State Transfer). REST embeds queries in standard (HTTP) web requests.
- JavaScript Object Notation (JSON) is a popular format for returning data.
- The `R` package `jsonlite` can be used to query a REST service that returns JSON.

Web Services

- Transport for Finland lets us query real-time positions of its trams.
- Users often have to register for an API key before using web service. No key is required for this one.
- For more information about the service, see `http://digitransit.fi/en/developers/services-and-apis`.
- If we access the URL directly,

```
$ curl http://api.digitransit.fi/realtime/vehicle-positions/v1/hfp/
    journey/tram/#
{"/hfp/journey/tram/RHKL00401/1007A/2/XXX/2247/undefined
    /60;24/29/17/50":{"VP":{"desi":"1007A","dir":"2","oper":"XXX","
    veh":"RHKL00401","tst":"2016-08-02T01:42:55.000Z","tsi
    ":1470102175,"spd":0,"hdg":210,"lat":60.215367,"long
    ":24.970981,"dl":567,"oday":"XXX","jrn":"XXX","line":"1007A","
    start":"2247","stop_index":17}}}
```

Web Services

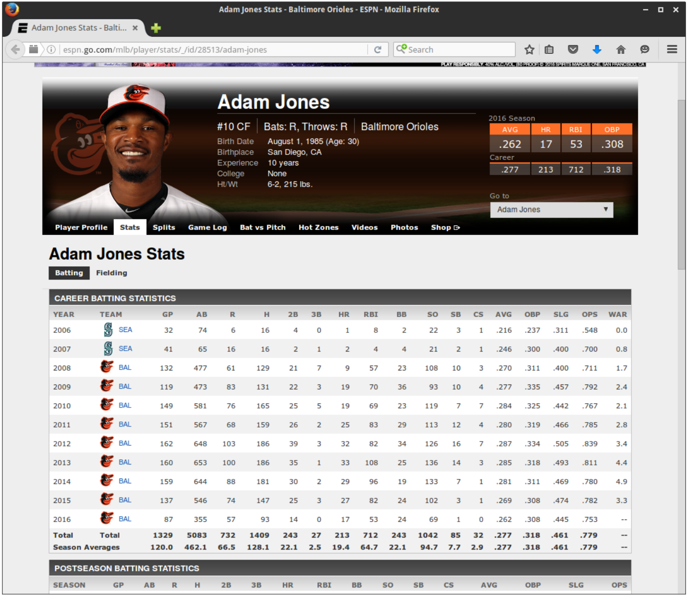
- Reading it with `jsonlite` package,

```
> library(jsonlite)
> url <- "http://api.digitransit.fi/realtime/vehicle-positions/v1/
    hfp/journey/tram/#"
> req <- fromJSON(url)
> str(req)
List of 1
 $ /hfp/journey/tram/RHKL00401/1007A/2/XXX/2247/undefined
    /60;24/29/17/50:List of 1
  ..$ VP:List of 16
  .. ..$ desi      : chr "1007A"
  .. ..$ dir       : chr "2"
  .. ..$ veh       : chr "RHKL00401"
  .. ..$ tst       : chr "2016-08-02T01:39:24.000Z"
  .. ..$ tsi       : int 1470101964
  .. ..$ hdg       : int 210
...
  .. ..$ lat       : num 60.2
  .. ..$ long      : num 25
  .. ..$ dl        : int 567
  .. ..$ line      : chr "1007A"
  .. ..$ start     : chr "2247"
  .. ..$ stop_index: int 17
```

Web Scraping

- Data on the web is often prepared for viewing (rendered in HTML) rather than for analysis (e.g. a CSV or Excel file).
- "Web scraping" is the process of writing a script to extract the data from the HTML.
- The `rvest` package supports web scraping in `R`.

Web Scraping



Adam Jones Stats

Batting Fielding

CAREER BATTING STATISTICS

YEAR	TEAM	GP	AB	R	H	2B	3B	HR	RBI	BB	SO	SB	CS	AVG	OBP	SLG	OPS	WAR
2006	SEA	32	74	6	16	4	0	1	8	2	22	3	1	.216	.237	.311	.548	0.0
2007	SEA	41	65	16	16	2	1	2	4	4	21	2	1	.246	.300	.400	.700	0.8
2008	BAL	132	477	61	129	21	7	9	57	23	108	10	3	.270	.311	.400	.711	1.7
2009	BAL	119	473	83	131	22	3	19	70	36	93	10	4	.277	.335	.457	.792	2.4
2010	BAL	149	581	76	165	25	5	19	69	23	119	7	7	.284	.325	.442	.767	2.1
2011	BAL	151	567	68	159	26	2	25	83	29	113	12	4	.280	.319	.466	.785	2.8
2012	BAL	162	648	103	186	39	3	32	82	34	126	16	7	.287	.334	.505	.839	3.4
2013	BAL	160	653	100	186	35	1	33	108	25	136	14	3	.285	.318	.493	.811	4.4
2014	BAL	159	644	88	181	30	2	29	96	19	133	7	1	.281	.311	.469	.780	4.9
2015	BAL	137	546	74	147	25	3	27	82	24	102	3	1	.269	.308	.474	.782	3.3
2016	BAL	87	355	57	93	14	0	17	53	24	69	1	0	.262	.308	.445	.753	--
Total	**Total**	**1329**	**5083**	**732**	**1409**	**243**	**27**	**213**	**712**	**243**	**1042**	**85**	**32**	**.277**	**.318**	**.461**	**.779**	--
Season Averages		**120.0**	**462.1**	**66.5**	**128.1**	**22.1**	**2.5**	**19.4**	**64.7**	**22.1**	**94.7**	**7.7**	**2.9**	**.277**	**.318**	**.461**	**.779**	--

POSTSEASON BATTING STATISTICS

SEASON	GP	AB	R	H	2B	3B	HR	RBI	BB	SO	SB	CS	AVG	OBP	SLG	OPS

Web Scraping

```
library(rvest)

my.html <- read_html("http://espn.go.com/mlb/player/stats/_/id/28513/
     adam-jones")

all.stats <- my.html %>%
    html_table(head = TRUE)

stats <- all.stats[[2]]
View(stats)
```

The %>% operator is defined in the `magrittr` package.

- Feeds data into a function.
- Many of them can be strung together.
- Behaves like the "pipe" operator on the Linux command line.

Web Scraping

stats

Filter

	CAREER BATTING STATISTICS	CAREER BATTING STATISTICS	CAREER BATTING STATISTICS	CAREER BATTING STATISTICS	CAREER BATTING STATISTICS	CAREER BATTING STATISTICS	CAREER BATTING STATISTICS	CAREE BATTIN STATIS
1	YEAR	TEAM	GP	AB	R	H	2B	3B
2	2006	SEA	32	74	6	16	4	0
3	2007	SEA	41	65	16	16	2	1
4	2008	BAL	132	477	61	129	21	7
5	2009	BAL	119	473	83	131	22	3
6	2010	BAL	149	581	76	165	25	5
7	2011	BAL	151	567	68	159	26	2
8	2012	BAL	162	648	103	186	39	3
9	2013	BAL	160	653	100	186	35	1
10	2014	BAL	159	644	88	181	30	2
11	2015	BAL	137	546	74	147	25	3
12	2016	BAL	87	355	57	93	14	0
13	Total	Total	1329	5083	732	1409	243	27
14	Season Averages	Season Averages	120.0	462.1	66.5	128.1	22.1	2.5

Showing 1 to 14 of 14 entries

Web Scraping

```
# Fix the header
colnames(stats) <- stats[1,]

# Remove the extra first row, and the row of totals and averages
stats <- stats[-c(1,13,14),]

# Change columns 3, ..., k to numeric
k <- ncol(stats)
for (j in 3:k) {
    stats[,j] <- as.numeric(stats[,j])
}
```

stats

Filter

	YEAR	TEAM	GP	AB	R	H	2B	3B	HR	RBI	BB	SO	SB	CS	AVG	OBP	SLG	OPS	WAR
2	2006	SEA	32	74	6	16	4	0	1	8	2	22	3	1	0.216	0.237	0.311	0.548	0.0
3	2007	SEA	41	65	16	16	2	1	2	4	4	21	2	1	0.246	0.300	0.400	0.700	0.8
4	2008	BAL	132	477	61	129	21	7	9	57	23	108	10	3	0.270	0.311	0.400	0.711	1.7
5	2009	BAL	119	473	83	131	22	3	19	70	36	93	10	4	0.277	0.335	0.457	0.792	2.4
6	2010	BAL	149	581	76	165	25	5	19	69	23	119	7	7	0.284	0.325	0.442	0.767	2.1
7	2011	BAL	151	567	68	159	26	2	25	83	29	113	12	4	0.280	0.319	0.466	0.785	2.8
8	2012	BAL	162	648	103	186	39	3	32	82	34	126	16	7	0.287	0.334	0.505	0.839	3.4
9	2013	BAL	160	653	100	186	35	1	33	108	25	136	14	3	0.285	0.318	0.493	0.811	4.4
10	2014	BAL	159	644	88	181	30	2	29	96	19	133	7	1	0.281	0.311	0.469	0.780	4.9
11	2015	BAL	137	546	74	147	25	3	27	82	24	102	3	1	0.269	0.308	0.474	0.782	3.3
12	2016	BAL	87	355	57	93	14	0	17	53	24	69	1	0	0.262	0.308	0.445	0.753	NA

Showing 1 to 11 of 11 entries

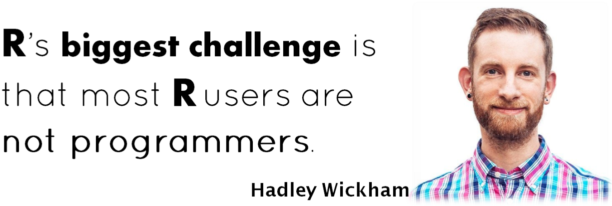
Big Datasets

- Base `R` expects all objects (vectors, matrices, data frames) to be completely loaded into memory.
- Modern datasets may be too large to fit into memory.
- The `bigmemory` and `ff` packages store matrices and data frames on disk, but allow them to be accessed somewhat like regular `R` objects.
  1. The `ff` project: `http://ff.r-forge.r-project.org`
  2. The `bigmemory` project: `http://www.bigmemory.org`

Data Manipulation with dplyr

Tidyverse

- "an opinionated collection of R packages designed for data science. All packages share an underlying philosophy and common APIs."



- Tidyverse includes:
  a. ggplot2 - grammar for plotting.
  b. dplyr - grammar for data manipulation. $(***)$
  c. readr, readxl, haven - reading data from files.
  d. magrittr - provides the pipe operator (%>%).
- Many tidyverse packages are discussed in the book *R for Data Science* (**?**), which is available online.

Data Manipulation with dplyr

. . . Demonstration . . .

(See `dplyr.Rmd`)

References I