Data Analysis and Visualization

Data import in R

Daniel Bader, Matthias Heinig

Overview: Data import

- Flat files (.txt, .csv, .tsv)
- Excel files
- Websites (XML)
- Relational database systems (SQL)
- readRDS() vs load()

How to get flat files into R

Get some file

Titanic passengers list

- Assign variables

```
# target directory
DATADIR <- file.path('./extdata')
# url of source file
titanic_url <- paste0('https://public.tableau.com/s/sites/default/files/media/',
                      'titanic%20passenger%20list.csv')
# target file for downloaded
titanic_file <-  file.path(DATADIR, 'titanic.csv')
```

- Download source to target

```
download.file(titanic_url, destfile = titanic_file, method='curl')
list.files(DATADIR, pattern="tita")
```

```
## [1] "titanic.csv"
```

Read a text file

- The R command `readLines` is fastest for doing this.

Whole file at once

```
titanic_vector <- readLines(titanic_file)
head(titanic_vector)
```

```
## [1] "\"pclass\",\"survived\",\"name\",\"sex\",\"age\",\"sibsp\",\"parch\",\"ticket\",\"fare\",\"cabin\",\"embarked\",
## [2] "1,1,\"Allen, Miss. Elisabeth Walton\",\"female\",29,0,0,\"24160\",211.3375,\"B5\",\"S\",\"2\",,\"St Louis, MO\""
## [3] "1,1,\"Allison, Master. Hudson Trevor\",\"male\",0.92,1,2,\"113781\",151.5500,\"C22 C26\",\"S\",\"11\",,\"Montrea
## [4] "1,0,\"Allison, Miss. Helen Loraine\",\"female\",2,1,2,\"113781\",151.5500,\"C22 C26\",\"S\",,,\"Montreal, PQ / C
## [5] "1,0,\"Allison, Mr. Hudson Joshua Creighton\",\"male\",30,1,2,\"113781\",151.5500,\"C22 C26\",\"S\",,\"135\",\"Mo
## [6] "1,0,\"Allison, Mrs. Hudson J C (Bessie Waldo Daniels)\",\"female\",25,1,2,\"113781\",151.5500,\"C22 C26\",\"S\",
```

```
str(titanic_vector)
```

```
##  chr [1:1310] "\"pclass\",\"survived\",\"name\",\"sex\",\"age\",\"sibsp\",\"parch\",\"ticket\",\"fare\",\"cabin\",\"e
```

Read a text file II

For line-by-line processing of large files

- get number of lines to read

```
cmd <- paste0("wc -l ", titanic_file, " | awk '{ print $1 }'")
n <- system(command = cmd, intern=TRUE)
```

- Loop over a file connection

```
con <- file(description=titanic_file, open="r") # open connection
for(i in 1:n) {
  tmp <- readLines(con, n=1) # do something on a line of data
}
close(con)
```

Basic tables

Read a flat file table into R with `read.table()`

```
titanic_df <- read.table(titanic_file, sep=',', header=TRUE)
class(titanic_df)
```

```
## [1] "data.frame"
```

- `read.table()` has many arguments, most have useful defaults
- Important parameters
  - `header`: Does the first line contain column names? [`FALSE`]
  - `sep`: What is the separator character [`" "`]
  - `dec`: What is the decimal point character (Germans!) [`"."`]

Some properties of a data.frame

```
dim(titanic_df)      # Dimensions
```

```
## [1] 1309   14
```

```
colnames(titanic_df) # Column names
```

```
##  [1] "pclass"    "survived"  "name"      "sex"       "age"
##  [6] "sibsp"     "parch"     "ticket"    "fare"      "cabin"
## [11] "embarked"  "boat"      "body"      "home.dest"
```

```
titanic_df[1:6, 1:4] # index based access
```

```
##   pclass survived                                            name    sex
## 1      1        1                         Allen, Miss. Elisabeth Walton female
## 2      1        1                        Allison, Master. Hudson Trevor   male
## 3      1        0                          Allison, Miss. Helen Loraine female
## 4      1        0                 Allison, Mr. Hudson Joshua Creighton   male
## 5      1        0 Allison, Mrs. Hudson J C (Bessie Waldo Daniels) female
## 6      1        1                                  Anderson, Mr. Harry   male
```

```
head(titanic_df, n=6)     # first six rows
```

More of the `read.*` family

- Basic function to read tables: `read.table`
- children
  - `read.csv(file, header=TRUE, sep = ",", dec = ".", ...)`
  - `read.delim(file, header=TRUE, sep = "\t", dec = ".", ...)`
- Frequently used parameters:
  - `stringsAsFactors` Should strings be coerced to factors [`TRUE`]
  - `check.names` ensure that they are syntactically valid variable names [`TRUE`]
  - `na.strings` which strings are read as `NA` [`"NA"`]
  - `comment.char` which lines are comments and therefore dropped [`"#"`]
  - `colClasses` A vector of classes to be assumed for the columns. Unless colClasses is specified, all columns are read as character columns and then converted using `type.convert()` to a simpler class.

Large tables ($> 10^6$ rows)

The basic R functions can have problems, when handling large data. This holds for many tasks you want to do in R, but the solution stays the same:

Get the right package!

For large tables this is `data.table` with `data.table::fread()` function

```
library(data.table)
titanic_dt = fread(titanic_url)
class(titanic_dt)
```

```
## [1] "data.table" "data.frame"
```

```
titanic_df[1:3, 1:4]
```

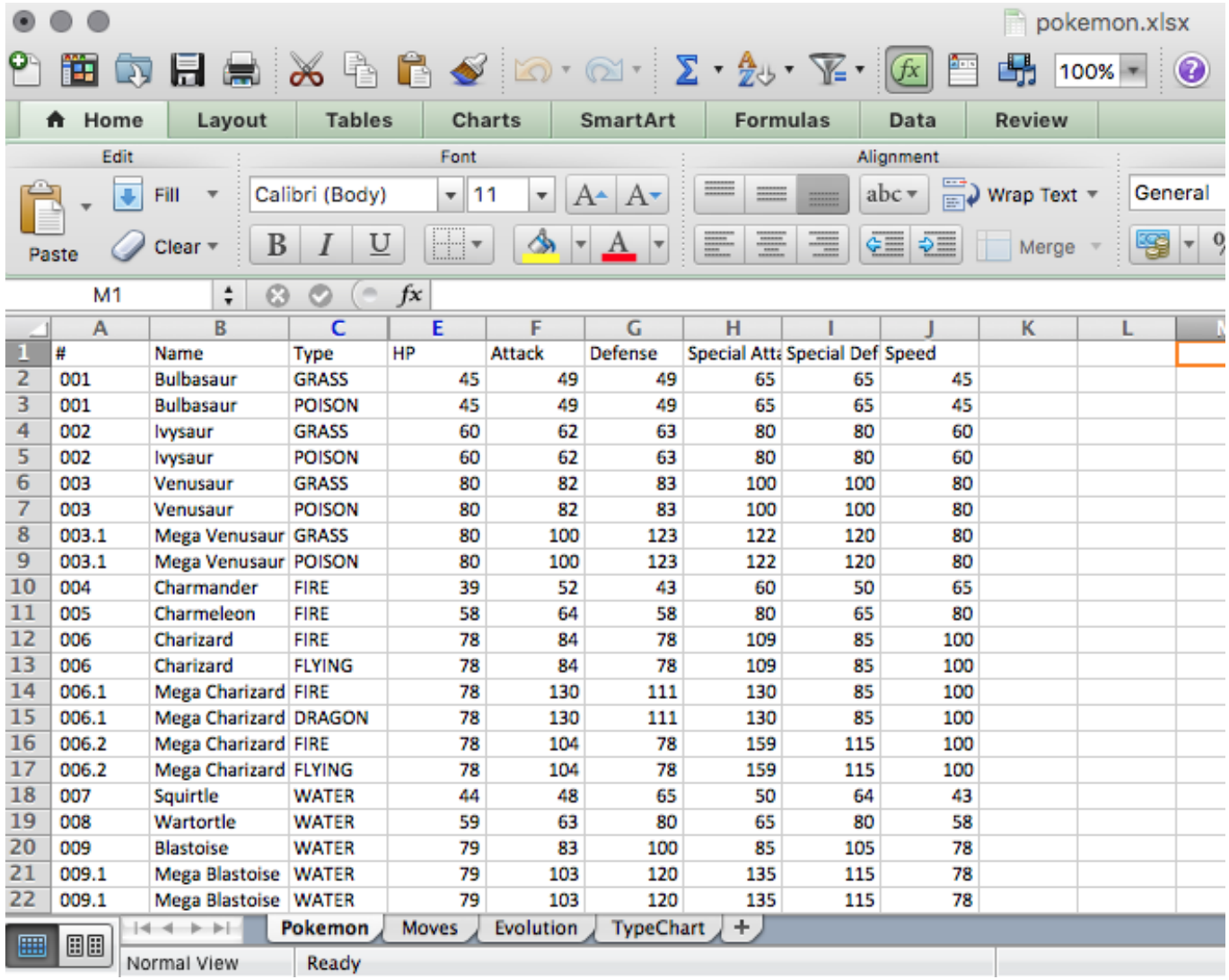
```
##   pclass survived                             name    sex
## 1      1        1  Allen, Miss. Elisabeth Walton female
## 2      1        1 Allison, Master. Hudson Trevor   male
## 3      1        0   Allison, Miss. Helen Loraine female
```

More `data.table` magic yet to come in the next lecture ...

Exercise: Flat files

How to get EXCEL files into R

Excel files



	A	B	C	E	F	G	H	I	J	K	L
1	#	Name	Type	HP	Attack	Defense	Special Atta	Special Def	Speed		
2	001	Bulbasaur	GRASS	45	49	49	65	65	45		
3	001	Bulbasaur	POISON	45	49	49	65	65	45		
4	002	Ivysaur	GRASS	60	62	63	80	80	60		
5	002	Ivysaur	POISON	60	62	63	80	80	60		
6	003	Venusaur	GRASS	80	82	83	100	100	80		
7	003	Venusaur	POISON	80	82	83	100	100	80		
8	003.1	Mega Venusaur	GRASS	80	100	123	122	120	80		
9	003.1	Mega Venusaur	POISON	80	100	123	122	120	80		
10	004	Charmander	FIRE	39	52	43	60	50	65		
11	005	Charmeleon	FIRE	58	64	58	80	65	80		
12	006	Charizard	FIRE	78	84	78	109	85	100		
13	006	Charizard	FLYING	78	84	78	109	85	100		
14	006.1	Mega Charizard	FIRE	78	130	111	130	85	100		
15	006.1	Mega Charizard	DRAGON	78	130	111	130	85	100		
16	006.2	Mega Charizard	FIRE	78	104	78	159	115	100		
17	006.2	Mega Charizard	FLYING	78	104	78	159	115	100		
18	007	Squirtle	WATER	44	48	65	50	64	43		
19	008	Wartortle	WATER	59	63	80	65	80	58		
20	009	Blastoise	WATER	79	83	100	85	105	78		
21	009.1	Mega Blastoise	WATER	79	103	120	135	115	78		
22	009.1	Mega Blastoise	WATER	79	103	120	135	115	78		

Read Excel worksheet in R

- Read 1 sheet into a data.frame directly from file
- useful parameters: range to specify the cell range (rows and cols) to load

```
library(readxl)
```

```
## Warning: package 'readxl' was built under R version 3.3.2
```

```
poke_file <- file.path(DATADIR, 'pokemon.xlsx')
poke_df <- read_excel(poke_file, sheet=1)
head(poke_df)
```

```
## # A tibble: 6 x 10
##   `#`   Name  Type  Total    HP Attack Defense `Special Attack`
##   <chr> <chr> <chr> <dbl> <dbl>  <dbl>   <dbl>             <dbl>
## 1  001  Bulb… GRASS   318    45     49      49                65
## 2  001  Bulb… POIS…   318    45     49      49                65
## 3  002  Ivys… GRASS   405    60     62      63                80
## 4  002  Ivys… POIS…   405    60     62      63                80
## 5  003  Venu… GRASS   525    80     82      83               100
## 6  003  Venu… POIS…   525    80     82      83               100
## # ... with 2 more variables: `Special Defense` <dbl>, Speed <dbl>
```

Read single Excel sheet in R

The `sheet` parameter accepts indices as well as names (case sensitive) for the desired sheet.

```
poke_df2 <- read_excel(poke_file, sheet='Pokemon')
identical(poke_df, poke_df2)
```

```
## [1] TRUE
```

The default recognizes only empty cells as missing values. If you have a Excel table where **NA** is already explicitely specified, e.g. from an former R output, it will be recognized as character.
You can specify string(s) for missing values:

```
poke_df2 <- read_excel(poke_file, sheet='Pokemon', na="NA")
```

Excel for Non-data-scientists

Checklist for Excel formatting adjusted from datacamp.com

- first row is usually reserved for the header
- first column is used to identify the rows, i.e. the "ID" or "NAME"
- concatenate words by inserting a `"_"` in between two words instead of a space
- Avoid symbols such as `?`, `$`, `%`, `^`, `&`, `*`, `(`, `)`, `–`, `#`, `?`, `,`, `<`, `>`, `/`, `|`, `\`, `[`, `]`, `{`, and `}`
- Delete any comments to avoid extra columns or NA's
- any missing values in your data set are indicated with `NA`

Exercise: Excel

What is XML? How can I get it into R?

XML basics

- XML stands for EXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to store and transport data
- XML was designed to be self-descriptive
- XML is a World Wide Web Consortium (W3C) Recommendation

Difference between XML and HTML

- XML was designed to carry data - with focus on what data is
- HTML was designed to display data - with focus on how data looks
- XML tags are not predefined like HTML tags are

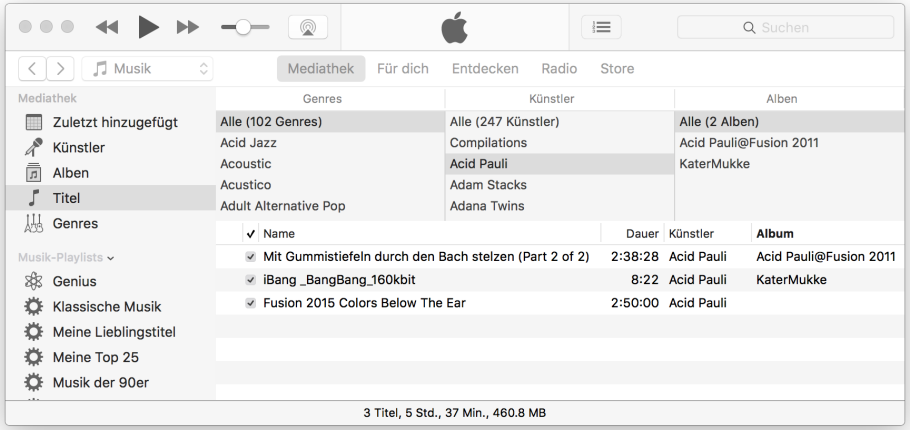
Usefull references and tutorials:

- http://www.w3schools.com/xml/ (http://www.w3schools.com/xml/)
- https://www.stat.berkeley.edu/~statcur/Workshop2/Presentations/XML.pdf (https://www.stat.berkeley.edu/%7Estatcur/Workshop2/Presentations/XML.pdf)

XML Examples

- Scraping HTML
- PubMed articles/abstracts
- European Bank exchange rates
- itunes - CDs, tracks, play lists, ...
- SBML - Systems biology markup language
- Books - Docbook
- SOAP - eBay, KEGG, ...

iTunes XML




```
<key>6110</key>
<dict>
  <key>Track ID</key><integer>6110</integer>
  ...
  <key>Name</key>
  <string>Fusion 2015 Colors Below The Ear</string>
  <key>Artist</key><string>Acid Pauli</string>
  <key>Kind</key><string>AAC-Audiodatei</string>
  <key>Location</key><string>file:///Users/...</string>
</dict>
```

XML Syntax (1)

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

- XML Documents Must Have a Root Element (here: note)
- All XML Elements Must Have a Closing Tag
- XML Tags are Case Sensitive
- XML Elements Must be Properly Nested
- Some special characters need to be escaped: Only < and & are strictly illegal in XML

XML Syntax (2)

```
<bookstore>
  <book category="children">
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title>Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

- XML elements are everything from the element start to the end tag
- An element can contain:
  - text
  - attributes
  - other elements
  - or a mix of the above

XML namespaces

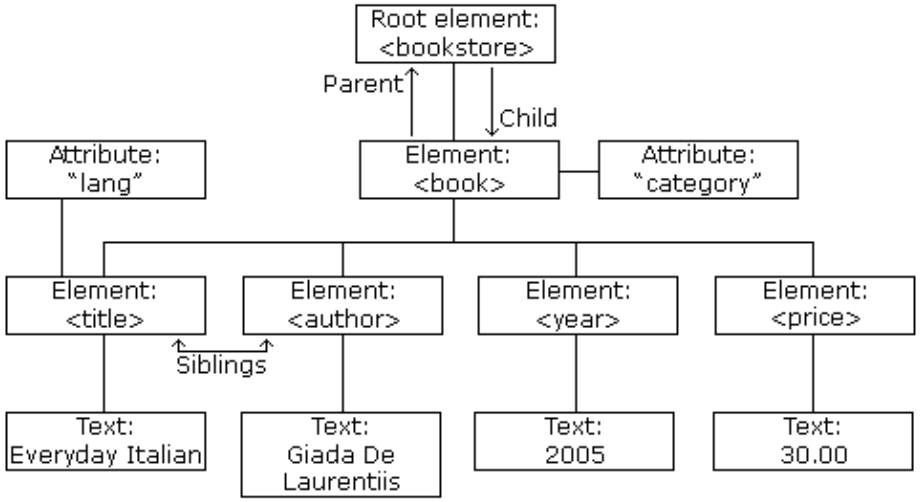
```
<root>
<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table xmlns:f="http://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
</root>
```

- This XML carries information about a HTML table and about a table (a piece of furniture)
- Solving the Name Conflict Using a Prefix
- Namespaces can alternatively be defined in the document root

```
<root
xmlns:h="http://www.w3.org/TR/html4/"
xmlns:f="http://www.w3schools.com/furniture">
```

XML tree structure

```
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```



Document Object model (DOM)

- The XML DOM defines a standard way for accessing and manipulating XML documents.
- All XML elements can be accessed through the XML DOM.
- The XML DOM defines the objects, properties and methods of all XML elements.
- The XML DOM is:
  - A standard object model for XML
  - A standard programming interface for XML
  - Platform- and language-independent
  - A W3C standard

In other words: The XML DOM is a standard for how to **get, change, add, or delete XML elements**

XML DOM in R

Extract the information about the author of a book from the bookstore example.

```
library(XML)
doc = xmlTreeParse("extdata/books.xml", useInternalNodes = TRUE)
root = xmlRoot(doc)
xmlElementsByTagName(root, "author", recursive=T)
```

```
## $book.author
## <author>Giada De Laurentiis</author>
##
## $book.author
## <author>J K. Rowling</author>
##
## $book.author
## <author>Erik T. Ray</author>
```

The option `useInternalNodes` specifies that the XML document is represented only by an internal C data structure and not as `R` object. It is required for using the DOM functions to access parent and ancestor nodes.

Useful functions to access DOM objects

FUNCTION	DESCRIPTION
xmlValue	get or set the text value of an element
xmlName	returns the name of the element
xmlAttrs	get or set attributes of an element
xmlParent	returns the direct parent element
xmlAncestors	returns all ancestor elements
xmlChildren	returns all child elements
xmlToDataFrame	transforms an element to a data.frame
xmlToList	transforms an element to a list

Question 1:

Here is again the bookstore example:

```
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
...
```

What does the following code return on the bookstore XML?

```
doc = xmlTreeParse("extdata/books.xml", useInternalNodes = TRUE)
root = xmlRoot(doc)
xmlAttrs(xmlParent(xmlElementsByTagName(root, "year", recursive = TRUE)[[1]]))
```

- A cooking
- B category
- C lang
- D en

Submit Show Hint Show Answer Clear

XPath

EXPRESSION	DESCRIPTION
/node	top-level node
//node	node at any level
node[@attr-name]	node that has an attribute named "attr-name"
node[@attr-name='bob']	node that has attribute named attr-name with value 'bob'
node/@x	value of attribute x in node with such attr.
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes
all above	Returns a collection of nodes, attributes, etc.

Wildcards can be used as well

WILDCARD	DESCRIPTION
*	Matches any element node
@*	Matches any attribute node
node()	Matches any node of any kind

XPath Predicates

XPATH EXPRESSION	RESULT
/bookstore/book[1]	Selects the first book element that is the child of the bookstore element
/bookstore/book[last()]	Selects the last book element that is the child of the bookstore element
/bookstore/book[last()-1]	Selects the last but one book element that is the child of the bookstore element
/bookstore/book[position()<3]	Selects the first two book elements that are children of the bookstore element
//title[@lang]	Selects all the title elements that have an attribute named lang
//title[@lang='en']	Selects all the title elements that have a "lang" attribute with a value of "en"
/bookstore/book[price>35.00]	Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00
/bookstore/book[price>35.00]/title	Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 35.00

XPath in R

Apply the function `xmlValue` to all nodes of type title

```
doc = xmlTreeParse("extdata/books.xml", useInternal = TRUE)
xpathApply(doc, "//title", xmlValue)
```

```
## [[1]]
## [1] "Everyday Italian"
##
## [[2]]
## [1] "Harry Potter"
##
## [[3]]
## [1] "Learning XML"
```

Find the first name of authors of all books that appeared before 2004

```
doc = xmlTreeParse("extdata/books.xml", useInternal = TRUE)
xp = "//book[year < 2004]/author"
xpathApply(doc, xp, function(x) strsplit(xmlValue(x), " ")[[1]][1])
```

```
## [[1]]
## [1] "Erik"
```

Question 2:

Here is again the bookstore example:

```
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
...
```

What does the XPath expression `//year[1]/../@category` return?

- A cooking
- B category
- C lang
- D en

Submit Show Hint Show Answer Clear

Getting data from HTML tables

- Most websites are optimized for human readers, not R programs.
- So they hold information in HTML tables.
- HTML (or XHTML) is a XML dialect, so there is a very convenient function in the XML package: `readHTMLTable`

```
library(RCurl)
```

```
## Loading required package: bitops
```

```
library(XML)
tables = readHTMLTable(getURL("https://en.wikipedia.org/wiki/Oktoberfest"))
tail(tables[[3]])
```

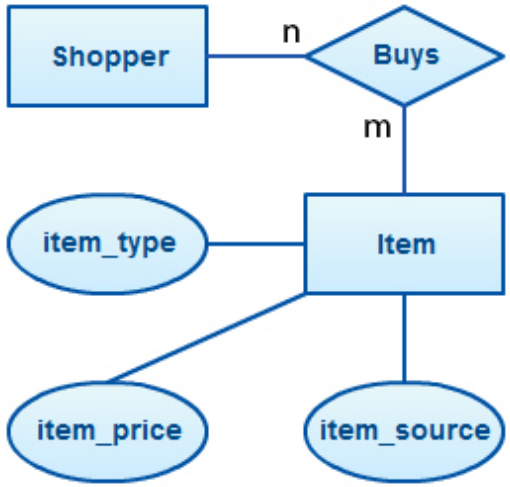
```
##   V1
## 1
##
## 1 This section needs additional citations for verification. Please help improve this article by adding citations to r
```

Exercise: XML

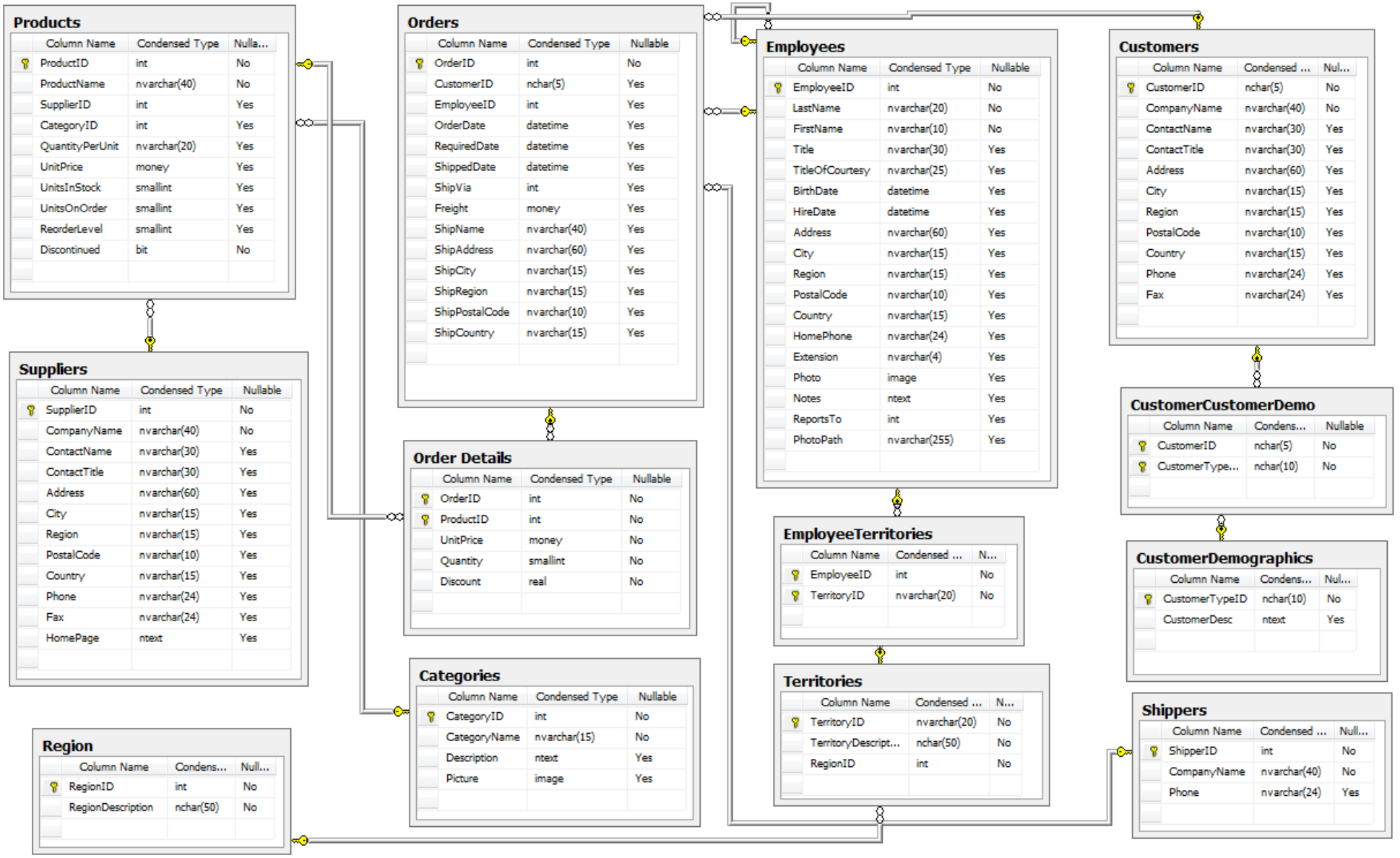
Relational data base recap & Queries in R?

Relational database system

- Decompose data into:
  - Entities
  - Relations
- Entities
  - have attributes
- Relations can
  - have attributes
  - be one to one
  - be one to many
  - be many to many



Example data base



Structured query language (SQL)

The `SELECT` statement is the most important for retrieving data

```
SELECT [DISTINCT] column_name,column_name FROM table_name;
```

or

```
SELECT * FROM table_name;
```

Data can be sorted already by the data base engine

```
SELECT column_name, column_name
FROM table_name
ORDER BY column_name ASC|DESC, column_name ASC|DESC;
```

Specific subsets of a data table can be queries with the `WHERE` clause

```
SELECT column_name,column_name FROM table_name WHERE column_name operator value;
```

SQL WHERE clause operators

OPERATOR	DESCRIPTION
=	Equal
<>	Not equal. Note: In some versions of SQL this operator may be written as !=
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

SQL JOIN: combining data from different tables

- The most common is the `INNER JOIN` returning all rows with matches in both tables

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name=table2.column_name;
```

- The `LEFT JOIN` keyword returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is `NULL` in the right side when there is no match. The `RIGHT JOIN` keyword works exactly in the other direction.
- The `FULL OUTER JOIN` keyword returns all rows from the left table (table1) and from the right table (table2). It combines the result of both `LEFT` and `RIGHT` joins.

Relational data base systems (RDBS)

- SQLite
  - lightweight file based SQL data base
  - very useful for local data bases with few (1) user(s)
  - no client / server infrastructure needed
- MySQL
  - very popular for web development
  - powerful client server architecture
  - setup is a bit timeconsuming

Generic interface DBI

- The data base driver is a specific adapter to each data base system
- The data base connection is a specific instance of a data base system

```
library(RSQLite)
drv <- dbDriver("SQLite")
con <- dbConnect(drv, dbname="extdata/Northwind.sl3")
```

DBI methods to inspect the data base

```
dbListTables(con)
```

```
##  [1] "Alphabetical list of products"  "Categories"
##  [3] "Current Product List"           "Customer and Suppliers by City"
##  [5] "CustomerCustomerDemo"           "CustomerDemographics"
##  [7] "Customers"                      "EmployeeTerritories"
##  [9] "Employees"                      "Order Details"
## [11] "Order Details Extended"         "Order Subtotals"
## [13] "Orders"                         "Orders Qry"
## [15] "Products"                       "Products Above Average Price"
## [17] "Products by Category"           "Region"
## [19] "Shippers"                       "Summary of Sales by Quarter"
## [21] "Summary of Sales by Year"       "Suppliers"
## [23] "Territories"                    "copy_of_customers"
```

```
dbListFields(con, "Customers")
```

```
##  [1] "CustomerID"   "CompanyName"  "ContactName"  "ContactTitle"
##  [5] "Address"      "City"         "Region"       "PostalCode"
##  [9] "Country"      "Phone"        "Fax"
```

DBI methods to retrieve data

```
drv <- dbDriver("SQLite")
con <- dbConnect(drv, dbname="extdata/Northwind.sl3")
res <- dbSendQuery(con, "SELECT companyname, contactname, city from customers limit 5")
fetch(res)
```

```
##                          CompanyName        ContactName           City
## 1                Alfreds Futterkiste       Maria Anders         Berlin
## 2 Ana Trujillo Emparedados y helados       Ana Trujillo M\xe9xico D.F.
## 3         Antonio Moreno Taquer\xeda     Antonio Moreno M\xe9xico D.F.
## 4                    Around the Horn       Thomas Hardy         London
## 5              Berglunds snabbk\xf6p Christina Berglund       Lule\xe5
```

```
## or shorter
dbGetQuery(con, "SELECT companyname, contactname, city from customers limit 5")
```

```
##                          CompanyName        ContactName           City
## 1                Alfreds Futterkiste       Maria Anders         Berlin
## 2 Ana Trujillo Emparedados y helados       Ana Trujillo M\xe9xico D.F.
## 3         Antonio Moreno Taquer\xeda     Antonio Moreno M\xe9xico D.F.
## 4                    Around the Horn       Thomas Hardy         London
## 5              Berglunds snabbk\xf6p Christina Berglund       Lule\xe5
```

DBI other functions

```
drv <- dbDriver("SQLite")
con <- dbConnect(drv, dbname="extdata/Northwind.sl3")
## conveniently read a table
tab <- dbReadTable(con, "customers")
## conveniently write a table (works only if the table does not exist yet)
if(dbExistsTable(con, "copy_of_customers")) {
   dbRemoveTable(con, "copy_of_customers")
}
```

```
## [1] TRUE
```

```
dbWriteTable(con, "copy_of_customers", head(tab))
```

```
## [1] TRUE
```

```
## or the append argument is TRUE
dbWriteTable(con, "copy_of_customers", tail(tab), append=TRUE)
```

```
## [1] TRUE
```

```
## Free up resources
dbDisconnect(con)
```

```
## [1] TRUE
```

```
dbUnloadDriver(drv)
```

Exercise: relational data bases

Import/export of binary R files

Native R data set import/export

```
poke_rds_file = "extdata/poke.RDS"
saveRDS(poke_df, file = poke_rds_file)
head(readRDS(file = poke_rds_file))
```

```
## # A tibble: 6 x 10
##   `#`   Name  Type  Total    HP Attack Defense `Special Attack`
##   <chr> <chr> <chr> <dbl> <dbl>  <dbl>   <dbl>            <dbl>
## 1  001  Bulb… GRASS   318    45     49      49               65
## 2  001  Bulb… POIS…   318    45     49      49               65
## 3  002  Ivys… GRASS   405    60     62      63               80
## 4  002  Ivys… POIS…   405    60     62      63               80
## 5  003  Venu… GRASS   525    80     82      83              100
## 6  003  Venu… POIS…   525    80     82      83              100
## # ... with 2 more variables: `Special Defense` <dbl>, Speed <dbl>
```

Difference save() vs saveRDS()

`save(object1, object2, ... , list, file="stored_file.Rdata")`

- arbitrary number of objects, i.e. whole R environment
- object saved with name

`load(file, verbose=FALSE)`

- verbose: should item names be printed during loading? If FALSE, stay silent.
- workaround: `new_name <- get(load("stored_file.Rdata"))`

`saveRDS(object, file)`

- exactly one R object, e.g. one RDataSet aka RDS
- only data is stored not the object name --> need to reassign object to variable at `readRDS()`

Supplement

References

- datasets
  - US government: https://catalog.data.gov/dataset (https://catalog.data.gov/dataset)
  - Tableau public: https://public.tableau.com/s/resources (https://public.tableau.com/s/resources)
  - github JSON: https://github.com/jdorfman/awesome-json-datasets (https://github.com/jdorfman/awesome-json-datasets)
  - datahub: https://datahub.io/dataset (https://datahub.io/dataset)
- data import
  - datacamp tutrial (https://www.datacamp.com/community/tutorials/r-data-import-tutorial)
  - R and HDF5 (https://www.bioconductor.org/packages/2.13/bioc/html/rhdf5.html)
  - JSON vs XML (http://www.json.org/xml.html)
  - JSON pkg in R (https://www.dropbox.com/s/m8o3a01wynla9ni/getting_web_data_r5_json_data.pdf)

Requirements for next exercise

Install the following R packages

- data.table
- DT
- ggplot2
- htmlwidgets
- jsonlite
- limma
- plotly
- RColorBrewer
- rhdf5
- slidify
- readxl
- XML