R403: The R Language for Statistical Computing

Topic 3: Reading from and Writing to External Data Sources

Kaloyan Ganev

2022/2023

Lecture Contents

Introduction

Introductory notes

- In statistics, we rarely work with small amounts of data allowing manual entry
- Most often data are contained in readily-available tables
- This information needs to be brought inside R so that it is processed and analysed
- Analysis of output is also substantial so there is the need to export it quickly and flawlessly to outer information storage
- We will demonstrate a number of tools designed for the above-mentioned tasks

Reading and writing text files

The `read.table()` function

- We take a real dataset: Airline routes database (from `http://openflights.org/data.html`)
- The file name is `routes.txt`
- Note: You can also read files directly from their url!
- Take a look at the file in a text editor (get a decent one, such as Notepad++, for example, don't count on Notepad)
- We will read the data in the following way:

```
routes.df <- read.table("routes.txt",
  delim = ",", header = FALSE)
```

- We used only three (of many) arguments of this function
- The command provides a very high level of flexibility
- Now read the associated help page

```
?read.table
```

`read.csv()`, `read.csv2()`, and `read.delim()`

- Wrappers ('shortcuts') for using `read.table()`
- `read.csv()` and `read.csv2()`: For cases when data are respectively comma-separated or semi-column-separated
- Note that for example MS Excel saves `*.csv` files using semi-columns instead of commas
- `read.delim()`: For cases when data are tab-separated (if you use the command `read.table()` with such data, you have to use the `sep = "\t"` option)
- (There is also the `read.fwf()` command for cases when columns in data have fixed width, which we will not discuss as it is rarely used)

Writing to text files

- Analogical commands are available for writing:
  - `write.table()`: for all kinds of text files
  - `write.csv()` and `write.csv2()`: for csv files
- Files are written in the current working directory unless specified otherwise

```
write.table(routes.df, "routes.tsv", sep = "\t")
```

Reading and writing foreign formats

The **foreign** package

- For reading data files in other applications' formats
- The package is a system one
- We will review only some of the most popular formats, for the remaining see the documentation
- First, load the package so that it can be used:

```
library(foreign)
```

- Then use the appropriate options to read your file

The **foreign** package: Examples

- Read SAS XPORT file on alcohol use in the U.S. (http://wwwn.cdc.gov/nchs/nhanes/search/DataPage.aspx?Component=Questionnaire&CycleBeginYear=2009):

```
sas1 <- read.xport("ALQ_F.XPT")
```

- (Also check out `read.ssd()` for reading native SAS datasets)
- Read Weka ARFF file (http://storm.cis.fordham.edu/~gweiss/data-mining/datasets.html):

```
weka1 <- read.arff("cpu.with.vendor.arff")
```

The **foreign** package: Examples (2)

- Read Stata file (Wooldridge, 2012, ch. 18):

```
stata1 <- read.dta("phillips.dta")
```

- Read SPSS file (http://calcnet.mth.cmich.edu/org/spss/Prjs_DataSets.htm):

```
spss1 <- read.spss("MathAssess-SpssFormat.sav", to.data.frame =
    TRUE)
```

The **foreign** package: Writing to files

- Options available for some formats only
- Examples:

```
write.arff(routes.df, "routes.arff")
write.dta(routes.df, "routes.dta") # will likely produce an error
    so
library(haven)
write_dta(routes.df, "routes.dta")
write.foreign(sas1, "stata2.csv", "stata2.do", package = "Stata")
    # SAS and SPSS are also possible
```

Reading and writing MS Excel files

- Not built-in in base R
- Contributed packages such as **xlsx** or **readxl** need to be installed
- Another (newer) one that provides good functionality is also **openxlsx** (we will use it for exercises)
- Reading (Wooldridge, 2012):

```
library(openxlsx)
xlsx1 <- read.xlsx("benefits.xlsx", sheetName = "benefits")
```

- or:

```
library(readxl)
xlsx2 <- read_excel("data/benefits.xlsx", sheet = "benefits",
    range = "A1:S1849")
```

Reading and writing MS Excel files (2)

- Writing:

```
write.xlsx(sas1, "sas1.xlsx") # With the xlsx package
library(writexl)
write_xlsx(sas1, "sas2.xlsx") # With the writexl package
```

Reading EViews wf1 files

- EViews has been quite popular in FEBA and in public and private institutions in Bulgaria
- To be able to read wf1 files, it is necessary to install the **hexView** package
- Reading is done with (Wooldridge, 2012):

```
library(hexView)
eviews1 <- readEViews("consump.wf1")
```

R and Databases

R and databases: General remarks

- R can successfully perform some tasks performed by DBMS
- However, it is sometimes useful to use the power of relational databases to complement R's capabilities
- Natural setup: your data already resides in a relational database and you need to access it and analyse it
- Why not do it directly from R?

Some SQL basics

- (You have a separate SQL course but we need to get ahead of things now)
- SQL: Structured Query Language
- Not a programming language (like R, Python, C++, etc.)
- Many DBMS implementations, both commercial and free
- We will use SQLite as the alternative (free) that provides the easiest access for our demonstration purposes
- There is a GUI that will be best for our teaching and learning environment: SQLiteStudio
- You can download it from https://github.com/pawelsalawa/sqlitestudio/releases/download/3.2.1/SQLiteStudio-3.2.1.zip and then just unzip it in a folder of your choice

Some SQL basics (2)

- We will be using the SQLite Sample Database (download from http://www.sqlitetutorial.net/sqlite-sample-database/)
- Unzip `chinook.db` and load it in SQLiteStudio
- There, you can inspect the contents of tables
- We will execute an SQL query from R to demonstrate how stuff works
- There basically two ways to connect to a database: through the **DBI** package and through the **RODBC** package

Using the **DBI** package

- **DBI** is decrypted as ‘database interface’
- This interface provides access to many RDBMSs such as MySQL, PostgreSQL, Oracle, etc.
- What **DBI** does is to split the client-server interaction into three parts:
  - Driver
  - Connection
  - Result

Using the **DBI** package (2)

- The driver serves to make easier the communication between R and a particular RDBMS
- Therefore, a driver for each of the listed RDBMS is available
- In our case we will be using the SQLite driver which is provided with the **RSQLite** package
- The connection is a wrapper for the actual connection between R and the RDBMS; it is the means through which all queries to and results from the RDMBS are transported
- The result describes the result of a query or statement (and contains some methods for formatting, printing, summarizing, etc.)

Back to the example: Connecting to the database

- First load the **RSQLite** package:

```
library(RSQLite)
```

- Note that this automatically loads **DBI**, too
- Next, we load the database driver:

```
drv <- dbDriver("SQLite")
```

- Make a character variable to hold the path to the database:

```
dbpath <- "j:/Pcloud/COURSES_SU/MSc/Probabilisitic and
    Statistical Computations with R/Topic 03/data/"
```

- Make the connection:

```
mydb <- dbConnect(drv, dbpath)
```

Exploring the database

- We can now list the database tables from within R;

```
dbListTables(mydb)
```

- Take for example the `artists` table
- We can explore its fields (i.e. names of variables/columns):

```
dbListFields(mydb, "artists")
```

- ...or read the entire table into an R data frame:

```
artists <- dbReadTable(mydb, "artists")
```

- etc.

Query the database from R

- We issue the query and put the result in a data frame in the following way:

```
db_data <- dbGetQuery(mydb, "SELECT trackid, tracks.name AS Track
    , albums.title AS Album, artists.name AS Artist FROM tracks
    INNER JOIN albums ON albums.albumid = tracks.albumid INNER
    JOIN artists ON artists.artistid = albums.artistid WHERE
    artists.artistid = 22;")
```

- (Note that the SQL code should not be split across multiple lines as in the slide! Here it's done only for better visibility)
- You can check what has been extracted by viewing the data frame
- From this point onwards the data is yours to perform statistical analysis on it

A MySQL example

- We will need for this the **RMySQL** package so you have to install it
- Load it and load the respective driver:

```
library(RMySQL)
drv2 = dbDriver("MySQL")
```

- Make the connection (this time the data is on a web server):

```
mydb2 <- dbConnect(drv2, dbname = "ensembl_compara_51", user="
    anonymous", password="", host = "ensembldb.ensembl.org")
```

- List tables and fields:

```
dbListTables(mydb2)
dbListFields(mydb2, "genome_db")
```

- Read data into a data frame:

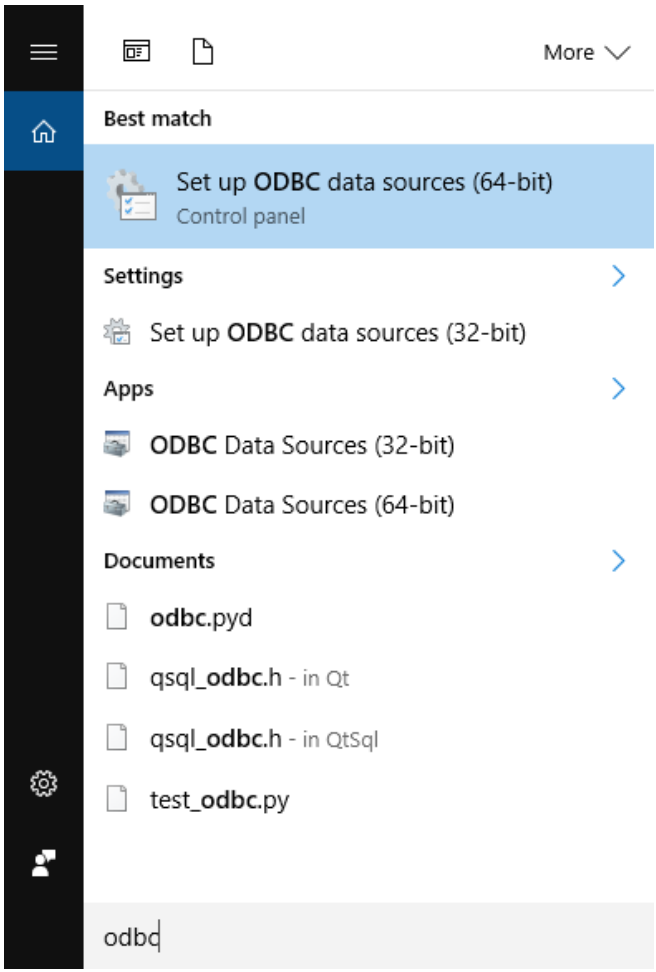
```
genome <- dbReadTable(mydb2, "genome_db")
```

ODBC

- Decrypted as ‘Open Database Connectivity’
- Developed originally by Microsoft in the beginning of the 90s
- Currently available on all major OS (Windows, Linux, Mac)
- Aimed to provide an API for accessing DBMS
- Provides independence from DBMS through the usage of an ODBC driver
- Most DBMS producers provide ODBC connectors
- Besides for DBMS, there are ODBC drivers for MS Excel and even for csv files

Setting up ODBC on Windows

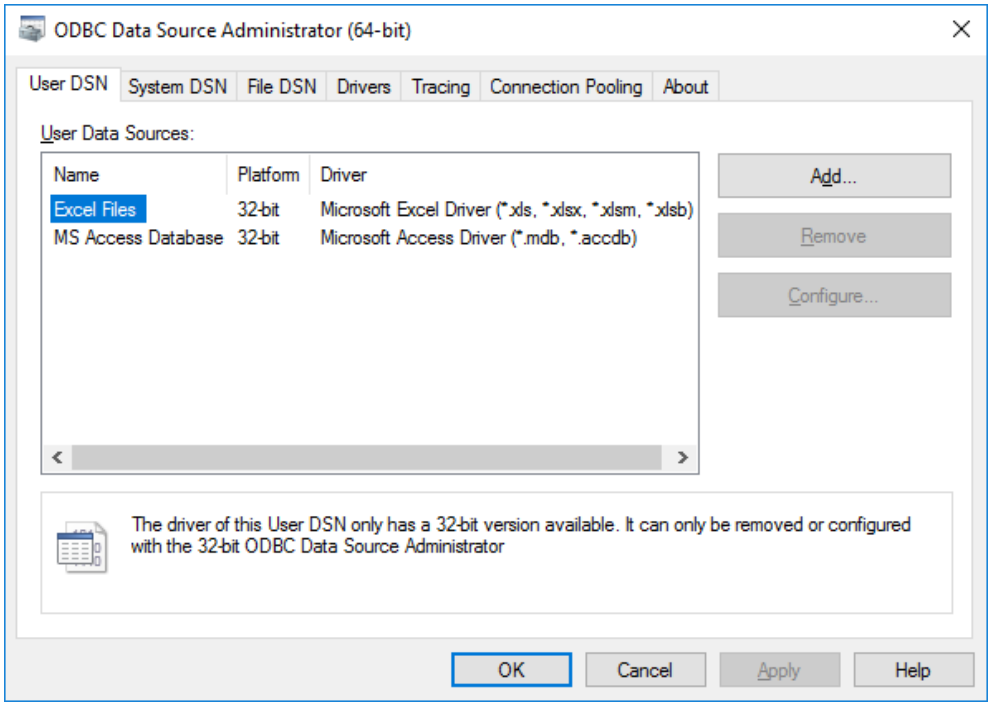
- Type 'odbc' in the search box, this appears:



- Click on `Set up ODBC data sources (64-bit)`

Setting up ODBC on Windows (2)

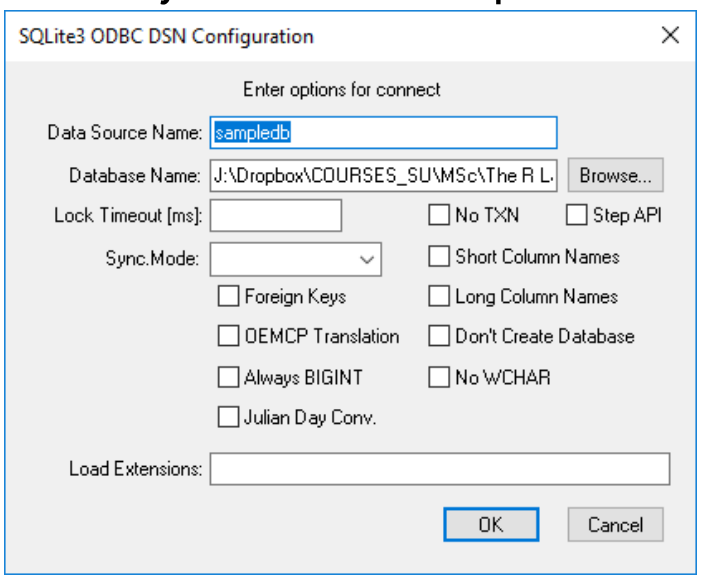
- A window appears showing the available connectors



- There are some more if you click Add
- If you miss something, get it online, e.g. http://www.ch-werner.de/sqliteodbc/

Create a new DSN

- DSN is 'data source name'
- In the last window that opened, click Add, select `SQLite3 ODBC driver`
- Name your DSN and point it to your database in what follows:



- Click OK

The **RODBC** package

- Allows using this common interface through R
- Should be installed additionally as it is not a system package
- We will try to connect to the `sampledb` DSN we created
- Load the package and make the connection:

```
library(RODBC)
odbc1 <- odbcConnect("sampledb", believeNRows = FALSE, rows_at_
    time = 1)
```

- The option `believeNRows = FALSE` checks whether the number of rows returned by the ODBC connection is believable

The **RODBC** package (2)

- List tables and import a whole table as a data frame:

```
sqlTables(odbc1)
odbc_data <- sqlFetch(odbc1, "albums")
```

- make a query (put data in a data frame again)

```
odbc_data2 <- sqlQuery(odbc1, paste("SELECT trackid, tracks.name
    AS Track, albums.title AS Album, artists.name AS Artist FROM
    tracks INNER JOIN albums ON albums.albumid = tracks.albumid
    INNER JOIN artists ON artists.artistid = albums.artistid
    WHERE artists.artistid = 22;"))
```

An important note on R and SQL

- We discussed only how to retrieve data from DBMS
- But the connections allow much more than this
- Depending on your SQL user rights, it is possible to do with a database what you would normally be able in the very DBMS

Further readings

- Packages' documentation
- Spector's book (somewhat obsolete but still useful)
- Nield, T. (2016): *Getting Started with SQL: A Hands-on Approach for Beginners*, O'Reilly