Rayanna Harduarsingh

September 23rd, 2020

Module 5 Notes

IST 687

Chapter 11/Module 5: Importing Data

- **Connecting R to External Data Sources**
    - R data import/export:
        - Range of methods for obtaining data from a wide variety of programs and formats
    - Two threads:
        - Data in a discrete "flat" file format
        - Data in non-discrete format such as system oriented (relational database)
- Connecting R to Discrete Files
    - R Packages
        - RODBC (Windows)
        - xIsReadWrite
        - xlxs(Mac)
        - XLConnect(Mac)
        - Data
    - Example:
        - Read/load census data via read.xls
        - install.packages('gdata')
            - library('gdata')
            - testFrame <-read.slx('link')
    - Cleansing and Transformation Process
        - Cleansing
            - Remove header rows
            - Remove unneeded columns

        - Remove last few rows
        - Copy first column to a column with a good name
        - **Remove first column in R:**
            - #removing 1st 3 rows
            - testFrame<-testFrame[-1:-3,]
        - **Keeping 1st 5 columns in R:**
            - testFrame <- testFrame[,1:5]
        - **Look at the last 5 rows of testFrame in R:**
            - tail(testFrame,5)
    - Transformation
        - Remove dots on front of state names
        - Convert “factor”/character data to numeric via a custom developed function
        - Recommend viewing ‘testFrame’ at various cleaning and transformation steps to see the affect of the R statement
- Question: Why are reading spreadsheets (or other files such as CSV) sometimes not practical/ appropriate?
    - Some files can actually be so massive and contains huge amounts of data that could lead to stability or development problems. Sorting and reading through spreadsheets can be confusing as well as time consuming. Sometimes the data can be unstructured and cause complications to just read it and understand it. There could also be mistakes such as human errors that can make reading not practical or convenient.

- **SQL from R:**
    - The R client supports sending commands mostly in SQL to the database server. The database server returns a result to the R client, which places it in an R data object (typically a dataframe) for use in further processing or visualization.

        - Returning to R, use install.packages() and library() to prepare the RMySQL package for use. If everything is working the way it should, you should be able to run the following command from the command line:
            - con <- dbConnect(dbDriver(“MySQL”), dbname = “test”)”
            - The dbConnect() function establishes a linkage or connection between R and the database we want to use.
- **JSON:**
    - JSON is a structured, but human readable, way of sending back some data from an application or a website.
    - Nondiscrete Data Access:
        - Remote applications are database “servers”
        - Data too large to store in local memory
        - Data is too large to store on local disk
    - Loading JSON data in R example:
        - > bikeURL <-
        - + “https://feeds.citibikenyc.com/stations/
        - + stations.json”
        - > apiResult <- getURL(bikeURL)
        - > results <- fromJSON(apiResult)
        - > length(results)
        - [1] 2
    - Question: Why is the data available via JSON? What are some other good and bad alternatives? Why did they make citibike available at all?
        - JSON is a way to store information in an organized, easy-to-access manner. In a nutshell, it gives us a human-readable collection of data that we can access in a really logical manner. It’s important for sites to be able to load data quickly and asynchronously. Making the data available makes it more easier for others to use and implement.