R Cheatsheet

• **read.table**, read.csv, for reading tabular data

• **readLines**, for reading lines of a text file

• **source**, for reading in R code files (inverse of dump)

• **dget**, for reading in R code files (inverse of dput)

• **load**, for reading in saved workspaces

• **unserialize**, for reading single R objects in binary form

• **write.table**, for writing tabular data to text files (i.e. CSV) or connections

• **writeLines**, for writing character data line-by-line to a file or connection

• **dump**, for dumping a textual representation of multiple R objects

• **dput**, for outputting a textual representation of an R object

• **save**, for saving an arbitrary number of R objects in binary format (possibly compressed) to

a file.

• **serialize**, for converting an

help.search(“rnorm”)

The readr package is developed to deal with reading in large flat files quickly. 用read\_table and read\_csv 替代read.table and read.csv

suppose I have a data frame with 1,500,000 rows and 120 columns, all of which are

numeric data.

1,500,000 × 120 × 8 bytes/numeric = 1,440,000,000 bytes

= 1,440,000,000 / 2²⁰ bytes/MB

= 1,373.29 MB

= 1.34 GB

So the dataset would require about 1.34 GB of RAM.

Using Textual and Binary Formats for Storing Data

The dump() and dput() functions are useful because the resulting textual format is editable,

and in the case of corruption, potentially recoverable. Unlike writing out a table or CSV file,

dump() and dput() preserve the metadata (sacrificing some readability), so that another user doesn’t

have to specify it all over again. For example, we can preserve the class of each column of a table or

the levels of a factor variable.