2013
Electronics and
Telecommunications
Trends

빅데이터 처리 및
분석 기술 특집

Ⅰ. What is R
Ⅱ. Why use R in Statistics
Ⅲ. How R Works
Ⅳ. Statistical Analysis with R
Ⅴ. Data Visualization with R
Ⅵ. Producing Graphics in R
Ⅶ. Working with Large Datasets

R 프로그래밍: 통계 계산과 데이터 시각화를 위한 환경

R programming: Language and Environment for Statistical Computing and Data Visualization

이두호 (D.H. Lee) BigData 시스템구조연구팀 선임연구원
Ye Ren* KAIST 그리드미들웨어센터 위촉연구원

* Chapters 5 and 6 were written in cooperation with Ye Ren.

The R language is an open source programming language and a software environment for statistical computing and data visualization. The R language is widely used among a lot of statisticians and data scientists to develop statistical software and data analysis. The R language provides a variety of statistical and graphical techniques, including basic descriptive statistics, linear or nonlinear modeling, conventional or advanced statistical tests, time series analysis, clustering, simulation, and others. In this paper, we first introduce the R language and investigate its features as a data analytics tool. As results, we may explore the application possibility of the R language in the field of data analytics.

Ⅰ. What is R

The R language (hereinafter, R) is one of the world's most popular platforms for developing statistical software. Archaeologists use it to track the spread of ancient civilizations, drug companies use it to discover which medications are safe and effective, and actuaries use it to assess financial risks and keep markets running smoothly. R is an open source programming language for statistical analyses and data visualization which is created by Ross Ihaka and Robert Gentleman in 1996 [1]. R is both a software and a language regarded as a dialect of the S language developed by the Bell Labs. S is available as the software S-PLUS commercialized by Insightful [2]. On the other hand, R is freely distributed under the terms of the GNU *General Public License*: its development and distribution are carried out by several statisticians known as the R Development Core Team.

R is available in several forms: the sources (written mainly in C and some routines in Fortran), essentially for Unix and Linux machines, or some pre-compiled binaries for Windows, Linux, and Mac OS. The files needed to install R, either from sources or from the pre-compiled binaries, are distributed from the website of comprehensive R archive network (CRAN) [3] where the instruction manuals for installation are also available. For the distributions of Linux, the binaries are generally available for the most recent versions. For more details, refer to CRAN website.

R allows the users to program loops to successively analyze several datasets. It is also possible that a single program combines with different statistical functions to perform more complex analyses. R users can benefit from a number of programs written in the S language, most of which are can be used directly with R.

Ⅱ. Why Use R in Statistics

What makes R so useful and helps explain its quick acceptance is that statisticians, engineers and scientists can improve the R software's code or write variations for specific tasks. Packages written for R add advanced algorithms, colored and textured graphs and mining techniques to dig deeper into databases. For this reason, global big data companies such as Google, Facebook, Oracle, IBM, and SAP have adopted R as their big data analytics tool and engine. This seems unavoidable because R is not only an open source language but also a powerful tool for data analyses. R has many features to recommend it [4]:

- R is a public domain implementation of the widely regarded S language, and the R/S platform is a de facto standard among statisticians.
- R is comparable, and often superior, in power to commercial products in most of the significant senses—variety of operations available, programmability, graphics, and so on.
- In addition to providing statistical operations, R is a general purpose programming language, so one can use it to automate analyses and create new functions that extend the existing language features.
- R saves datasets between sessions, so it is not

necessary to reload them each time. It saves command history automatically.

- R is available for Windows, Mac, and Linux OS.
- Due to an open source language, it is easy to get help from user community. Also, lots of new functions and state of the art algorithms are contributed by users and developers, many of whom are prominent statisticians.
- R incorporates features found in object-oriented and functional programming language.

The terms object-oriented programming and functional programming were mentioned above. The following sections provide an overview of two topics.

1. Object-Oriented Programming

The advantages of object-oriented programming can be explained by example. Consider statistical regression. When performing a regression analysis with other statistical packages such as statistical analysis system (SAS) or statistical package for social science (SPSS), one may get abundant of outputs on the screen. By contrast, if calling the `lm()` regression function in R, the function returns the object which contains all results including the estimated coefficients, their standard errors, residuals, and other statistics. User then picks and chooses, programmatically, which parts of that object to extract. This type of R's approach makes programming much easier because R offers a certain uniformity of access to data. This uniformity stems from the fact that R is polymorphic, which means that a single function can be applied to different types of inputs, which processes in the appropriate way. Such a function is called a generic function which is similar concept of virtual function in C language.

2. Functional Programming

As is typical in functional programming language, a common theme in R programming is avoidance of explicit iteration. Instead of coding loops, users exploit R applied to different types of inputs, which piterative behavior implicitly. This may lead to code that executes even more efficiently, and it can make a huge timing difference when running R on a large dataset. The functional programming nature of R offers some advantages: (i) clearer and more compact code, (ii) potentially much faster execution speed, (iii) less debugging due to simpler code, and (iv) easier transition to parallel programming.

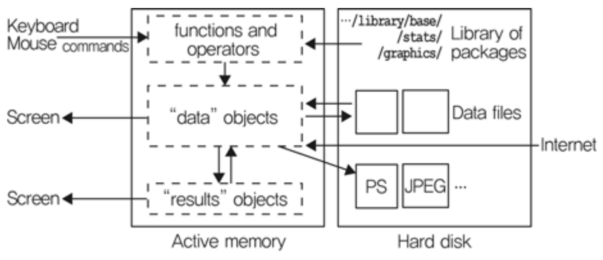
Ⅲ. How R Works?

Once R is installed, the software is executed by launching the corresponding executable. The prompt, by default '>', indicated that R is waiting for your commands. At this stage, a new user is likely to wonder "What can I do with R?" It is indeed very useful to have a few ideas on how R works, and this is what we will see now.

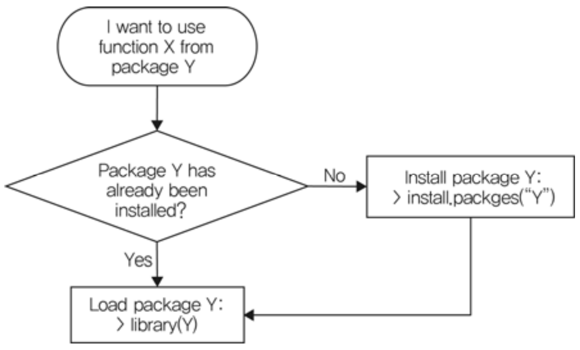
First, R is an interpreted language, not a compiled one, which means that all commands typed on the keyboard are directly executed without requiring building a complete program like in C, Fortran, Pascal, and so on. Second, R's syntax is very simple and intuitive. For example, a linear regression can be conducted by typing the command **lm(y~x1+x2)** which means "fitting a linear model with **y** as a response variable and **x1** and **x2** as explanatory variables." In R, all functions need to written with parentheses, even if there is nothing within them (for example, **help()**). If one types the name of a function without parentheses, R just shows the content of the function.

When R is running, variables, datasets, functions, results, and all other things are stored in the active memory of the system in the form of objects (see (Figure 1)). The readings and writings of files are used for input and output of data and results including graphics. The user executes the functions by typing some commands. The results are displayed on the screen, stored in an object, or written on the disk. Since the results are themselves are objects, they can be considered as data and analyzed as such. Data files can be read from the local disk or from a remote server through internet.

The functions available to the users are stored in a library localized on the disk in a directory. This directory contains packages. A package is a collection of pre-programmed functions, often including functions for specific tasks. There are two types of packages: those that come with the base installation of R and those that users can manually



(Figure 1) Schematic view of how R works.


I want to use function X from package Y
Package Y has already been installed?
No
Install package Y:
> install.packges("Y")
Yes
Load package Y:
> library(Y)
(Figure 2) Overview of the process of installing and loading packages in R.

download and install. This is one of the big advantages of open source programming language: people love to share. Users throughout the world have written their own special purpose R packages, placing them in the CRAN repository and elsewhere. In R, there is a difference between installing and loading packages. Install implies adding the package to R. Load means that users can access all the functions in the package, and are ready to use it. It is impossible to load a package if it is not installed. (Figure 2) summarizes the process of installing and loading packages.

Ⅳ. Statistical Analysis with R

The package **stat** contains functions for a variety of basic statistical analyses such as classical tests, linear and nonlinear model, distributions, hierarchical clustering, time series analysis, and multivariate analysis. Other advanced statistical analyses are carried out in a large number of packages. Some of packages are available with a base installation of R, and many others are contributed and must be installed individually. We will start with a simple example which requires no other package that **stat** to introduce the basic statistical analysis for a given dataset.

In statistics, analysis of variance (ANOVA) is a collection of statistical models, and their associated procedures, in which the observed variance in a particular variable is partitioned into components attributable to different sources of variation. In its simplest form, ANOVA provides a statistical test of whether or not the means of several groups are all equal. The function for the analysis of variance in the **stats** package is **aov()**. Let us take a dataset accompanied by a installation of R: **InsectSpray**. Six insecticides were tested and the observed response was the count of insects. Each insecticide was tested 12 times, thus there are 72 observations. Under the null hypothesis that there is no difference among six insecticides, the analysis is performed as follows:

```
data(InsectSprays) # load a data set
ex<-aov(count~spray, data=InsectSprays) # do ANOVA
```

The results are not showed up because they are assigned to an object called **ex**. To extract the results, one may use **summary()** function for detailed ones:

```
summary(ex)
              Df  Sum Sq  Mean Sq  F value    Pr(>F)
Spray          5  2668.8    533.7   34.702  < 22e-16 ***
Residuals 66      1015.2    15.38
```

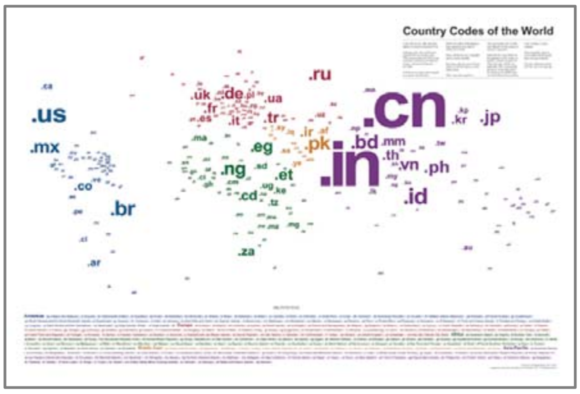
From the above, the null hypothesis is rejected because p-value is smaller than 22e-16. In this case, the test is said to be statistically significant.

V. Data Visualization with R

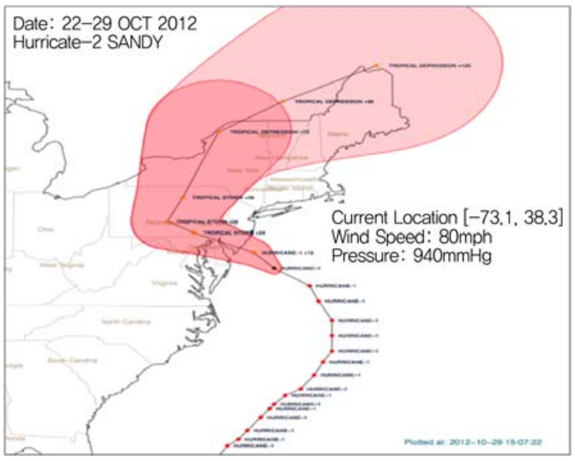
Data visualization is the visual representation of data, by which means the sparse information and potential relationship among aspects of the dataset can get across to people clearly and effectively. Data visualization needs to maintain a balance between being both functional and aesthetic so that meaningful insights into a rather complex dataset can get conveyed in a more intuitive way [5]. In (Figure 3) and (Figure 4), we exemplify several useful cases of data visualization.

(Figure 3) [6] maps 245 top-level domain country codes. Each two-digit code is aligned over that country it represents and is sized relative to the population of the country or territory. The below legend provides color coded quick reference. The map is a helpful resource for managers of global web sites or global marketing executives.

(Figure 4) [7] plots Hurricane Sandy data on a slice of the US. In the figure we are able to watch Sandy's past trace and also a forecast for its future activity and the affected area. The figure is informative in terms of conveying important data about Sandy and is intuitively understandable to



(Figure 3) Country codes of the world



(Figure 4) Path & cone of Hurricane Sandy in R.

ordinary people.

Ⅵ. Producing Graphics in R

R is rich with facilities for creating and developing interesting graphics [8]. From an overview, there are four major graphic packages in R: they are **base**, **grid**, **lattice**, and **ggplot2**. The **base** package contains functionality for many plot types and **lattice** and **grid** are supplied with R's recommended packages and are included in every binary distribution. R's **base** graphic system reaches to its limit when one wants to design complex layouts and **grid** was designed to overcome some of these limitations. Using **grid** as the underlying primitives, packages like **lattice** and **ggplot2** are developed. Subsequent part of this paper will put emphasis on the study of **ggplot2** since it is very powerful in producing statistical or data graphics due to a deep underlying grammar.

1. The Grammar of Graphics in ggplot2

The theoretical basis of **ggplot2** is the layered grammar of graphics. Grasping the grammar is very useful for both users and developers. To the user, the grammar suggests the high-level aspects of a plot that can be changed to customize graphics to particular problems: to the developer, it is much easier to add new capabilities to **ggplot2**.

The layered grammar defines a plot as the combination of [9]:

- A default dataset and the set of mappings from variables to aesthetics.
- One or more layers
- A coordinate system
- The faceting specification

The visuality of a plot is rendered by points. To display a point on the screen, the computer needs to know the aesthetic specifications of the point which include color, size, shape and many others. As well as aesthetics that have been mapped to variables, they can also be specified constant. The coordinate system controls how the axes and grid lines are drawn in the plot. The faceting specification creates small multiples each showing a different subset of the whole dataset.

Layering is the mechanism by which additional data elements are added to a plot. There are five components of a layer [9]:

- The data. It must be an R data frame.
- A set of aesthetic mappings. This describes how variables in the data are mapped to aesthetic properties of the layer.
- The **geom**. It defines the set of available aesthetic properties of the layer.
- The **stat**. It transforms the raw data in some useful way and returns a data frame with new

variables that can also be mapped to aesthetics.

- The position adjustment. It adjusts elements to avoid overplotting.

Each layer can convey a different dataset and have a different aesthetic mapping. The **geoms** actually perform the rendering of the data and control the appearance of the plot that we create. Each **geom** has a set of aesthetics that it understands and a set that are required for drawing. The names of generated variables by **stat** must be surrounded with ".." in their later usage instances, in order to prevent confusion with the same name in the original dataset. Every **stat** is associated with a default **geom** and every **geom** with a default **stat** which means that we only need to specify one of **stat** or **geom** to get a completely specified layer.

2. The Usage Examples of ggplot2

Each plot of some dataset actually can be created by many different ways, such as by using commercial statistical packages, or by using the free open-source statistical packages, or by using base R plus self-defined functions. In **ggplot2**, likewise, we may use the function **qplot()** to create a simple yet expressive plot all at one line, and we may also use the function **ggplot()** to create the plot layer by layer. **ggplot()** allows users to create plots that could not be made using **qplot()**, because **qplot()** permits only a single dataset and a single set of aesthetic mappings. Moreover, layers are R objects and can be stored as variables, which is convenient for further enhancement and flexible for reutilization. In following examples, we explain how to generate a graphic by using the provided functions in **ggplot2** and how to realize data visualization by combining **ggplot()** with other packages.

a. Example of Using qplot() and ggplot()

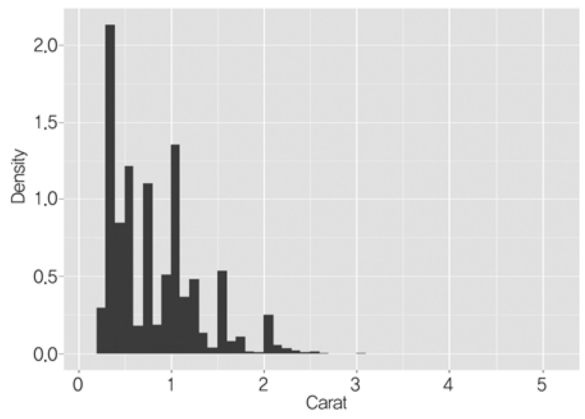
For one of the available data frames named "diamonds" in the package of **ggplot2**, the below three sets commands generate the same plot as shown in (Figure 5).

```
qplot(carat, ..density.., data = diamonds, geom = "histo-
gram", binwidth=0.1)

ggplot(diamonds, aes(carat)) + geom_histogram(aes(y
= ..density..), binwidth=0.1)

ggplot(diamonds, aes(carat)) + stat_bin(aes(y = ..density..),
binwidth = 0.1)
```

By comparing the first and second sets of commands, we can see that in generating the same easy plot, **qplot()** implements the command all at one line while **ggplot()** builds the plot according to the layering mechanism. By comparing the second and the third sets of commands, we can see that by using the default associations we specify either



(Figure 5) The Density of diamonds' weight distribution.

geom or **stat** to render a complete layer. Since the "**density**" is generated variable and not belong to the original data frame, we need to surround it with ".. ".

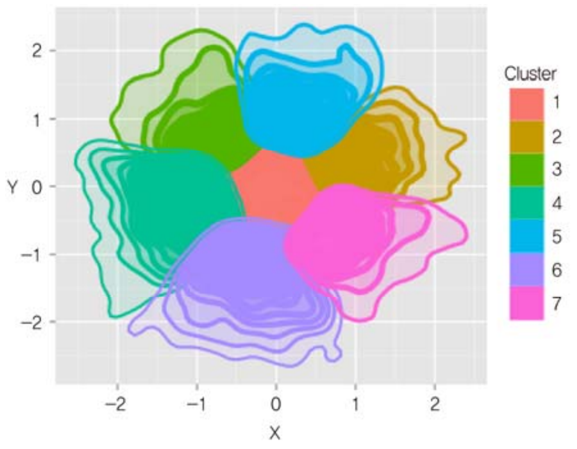
b. Encoding Two-dimensional Density in Graphical Variables [10]

In this example, we first create a sample dataset distributing in two-dimensions, and then we cluster the data points and try to express the estimate of the data's density over the space. After getting the data to be visualized prepared in "**myData**" we sequentially execute the below commands to render the plot. We save the intermediate plot as an R object named **zp4** and add modifications to it step by step.

```
zp4 <- ggplot(myData, aes(x = X, y = Y))

zp4 <- zp4 + stat_density2d(aes(fill = Cluster, colour =
Cluster, alpha = ..level.., size =..level..), geom = "polygon")
```

If we print **zp4** till this step we will get the plot shown in (Figure 6). The created sample data points are clustered into seven clusters and are rendered in different colors. The two-dimensional density estimation of the sample data points is rendered by polygons, where **alpha** and **size** are both mapped to the computed two-dimensional density estimate. **alpha** is the aesthetic deciding the transparency of a color and **size** is the aesthetic for the width of lines. Therefore, the area with darker color and thicker line indicates the space with higher density estimate and the area with transparent color and thinner line means the space with lower density



(Figure 6) Two-dimensional density represented in space.

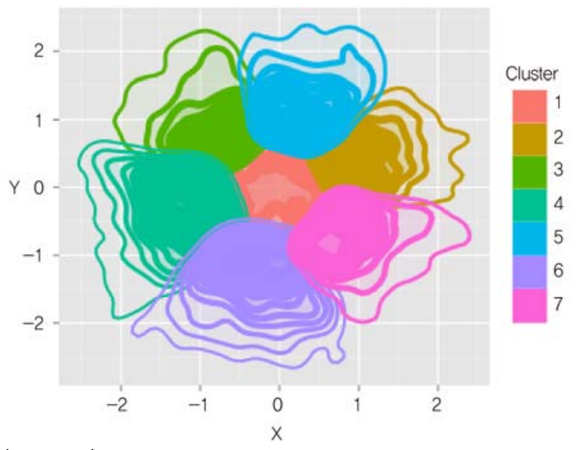
estimate.

```
zp4 <- zp4 + scale_alpha(range = c(0, 1/2), guide = "none")
```

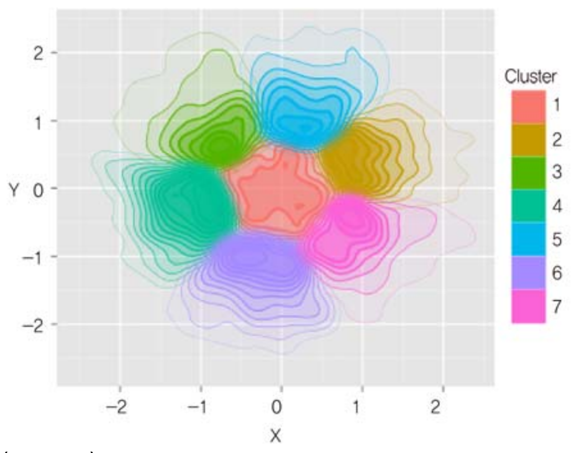
If we print **zp4** till this step we will get the plot shown in (Figure 7). By using the function **scale_alpha()**, we narrow down the **alpha** range from (0,1) to (0,1/2).

```
zp4 <- zp4 + scale_size(range = c(0, 3/2), guide = "none")
```

If we print **zp4** till this step we will get the plot shown in (Figure 8). By using the function



(Figure 7) Two-dimensional density represented in space with scaled transparency.



(Figure 8) Two-dimensional density represented in space with scaled transparency and line width.

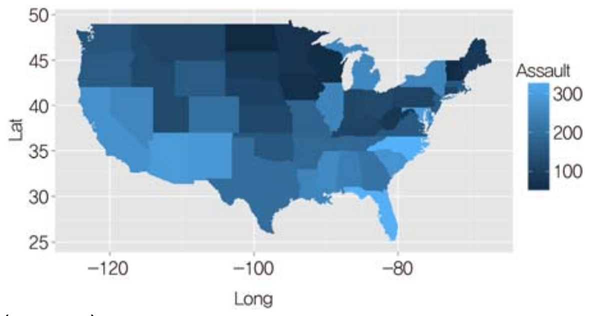
scale_size(), we narrow down the size range to (0,3/2).

We can infer from the above plotting procedures that the functions implementing the grammar of graphics in **ggplot2** can provide flexible and versatile methods to compose functional and beautiful graphics.

c. Data Visualization with ggplot() and Other Package-Matching the Identifiers between our Data and the Map Data [11]

We install the "**maps**" package and use the datasets come within it. By using **map_data()** function in **ggplot2**, we can convert a map into map data in the form of data frame. Then, by using **merge()** we can merge the map data with the data we have surveyed such as the number of assaults happened in each state to produce a choropleth map. The detail R codes can be found in [11] and running the codes will render plots similar to the one shown in (Figure 9).

From (Figure 9), we visualize the surveyed data on assault cases in the US and match that to the



(Figure 9) Choropleth map reflecting number of assaults in the US.

map data. Through such data visualization it is easy for us to have a first impression of which parts of the US are safer than other parts in terms of lower possibility for the occurrence of assault.

Ⅶ. Working with Large Datasets

This paper aimed at introducing R language and investigating its application in the data visualization. This paper now ends by discussing the analysis of large datasets in R.

As stated earlier, R holds all of its objects in virtual memory. For most of users, this design decision has led to a zippy interactive experience, but for analysts working with large datasets, it can lead to slow program execution and memory-related errors. Memory limits will mainly depend on the R build (32 vs. 64-bit) and for 32-bit Windows, on the OS version involved. Error messages starting with **cannot allocate vector of size** typically indicate a failure to obtain sufficient contiguous memory, while error messages starting with **cannot allocate vector of length** indicate that an address limit has been executed. When working with large datasets, it is recommended to use a 64-

bit build if at all possible. For all builds, the number of elements in a vector is limited to 2,147,483,647 (see **`?Memory`** for more information).

There are three issues to consider when working with large datasets: (i) efficient programming to speed execution, (ii) storing data externally to limit memory issues, and (iii) using specialized statistical packages designed to efficiently analyze massive amounts of data.

There are many programming tips to improve performance when working with large datasets. Those tips include vectorizing calculations, using matrices rather than data frames, testing programs on subsets of the large dataset, and deleting temporary objects that are no longer needed. With large datasets, increasing code efficiency will only get users so far. When bumping up against memory limits, objects can be stored externally or specialized package for large datasets can be used. There are several packages available for storing objects outside of R's main memory. This strategy involves storing objects or data in external database (DB) or in binary flat files on disk, and then accessing portions as they are needed. Useful R packages for solving memory problems include **`ff`**, **`bigmemory`**, **`biganalytics`**, **`filehash`**, **`ncdf`**, **`RODBC`**, **`RMySQL`**, **`ROracle`**, and **`RSQLite`**.

Terminology

Open source software (OSS) The computer software that is available with source code: by using an open source license the copyright holders provide source codes and the rights to study, change and distribute the software to anyone and for any purpose.

Data visualization The study of the visual representation of data, meaning "information that has been abstracted in some schematic form, including attributes or variables for the units of information"

Working with datasets in the gigabyte to terabyte range should be challenging in any language. For more information on the method available within R, see the CRAN Task View: High-Performance and Parallel Computing with R in [12].

Abbreviations

ANOVA	Analysis Of Variance
DB	Database
CRAN	Comprehensive R Archive Network
GNU	GNU is Not Unix
SAS	Statistical Analysis System
SPSS	Statistical Package for Social Science

References

[1] R. Ihaka and R. Gentleman, "R: a Language for Data Analysis and Graphics," *J. Comput. Graph. Stat.*, vol. 5, no. 3, 1996, pp. 299-314.

[2] S-Plus and R Statistics Software. http://stat.ethz.ch/~www/SandR.html

[3] http://cran.r-project.org

[4] N. Matloff, *The Art of R Programming*, No Starch Press, 2011.

[5] V. Friedman, "Data Visualization and Infographics," Graphics, Monday Inspiration, Jan. 14th, 2008. http://www.smashingmagazine.com/2008/01/14/monday-inspiration-data-visualization-and-infographics/

[6] http://www.historyshots.com/OtherArtists/4015.cfm

[7] https://github.com/hrbrmstr/sandy

[8] CRAN, "CRAN Task View: Graphic Display & Dynamic Graphics & Graphic Devices & Visualization," Dec. 13th, 2012. http://cran.r-project.org/web/views/Graphics.html

[9] H. Wickham, *ggplot2 Elegant Graphics for Data Analysis*, Springer Science+Business Media, LLC, 2009, pp. 27-41.

[10] http://www.r-bloggers.com/representing-density-in-two-dimensions/

[11] Hadley Wickham, *ggplot2 Elegant Graphics for Data Analysis*, Springer Science+Business Media, LLC, 2009, pp 78-79.

[12] http://cran.r-project.org/web/views