3

Divide and Recombine: Approach for Detailed Analysis and Visualization of Large Complex Data

Ryan Hafen

CONTENTS

3.1 Introduction ................................................................. 35
3.2 Context: Deep Analysis of Large Complex Data .................. 36
  3.2.1 Deep Analysis ....................................................... 36
  3.2.2 Large Complex Data ............................................... 37
  3.2.3 What is Needed for Analysis of Large Complex Data? ........ 37
3.3 Divide and Recombine ...................................................... 38
  3.3.1 Division Approaches ............................................. 39
  3.3.2 Recombination Approaches ..................................... 39
  3.3.3 Data Structures and Computation ............................... 40
  3.3.4 Research in D&R .................................................. 41
3.4 Trelliscope ...................................................................... 41
  3.4.1 Trellis Display/Small Multiples ................................ 41
  3.4.2 Scaling Trellis Display ............................................ 41
  3.4.3 Trelliscope ............................................................ 42
3.5 Tessera: Computational Environment for D&R .................... 43
  3.5.1 Front End ............................................................. 43
  3.5.2 Back Ends ............................................................. 43
    3.5.2.1 Small Scale: In-Memory Storage with R MapReduce ...... 44
    3.5.2.2 Medium Scale: Local Disk Storage with Multicore R
       MapReduce .............................................................. 44
    3.5.2.3 Large Scale: HDFS and Hadoop MapReduce via RHIPE . 44
    3.5.2.4 Large Scale in Memory: Spark .............................. 44
3.6 Discussion ...................................................................... 45
References ............................................................................ 45

3.1 Introduction

The amount of data being captured and stored is ever increasing, and the need to make
sense of it poses great statistical challenges in methodology, theory, and computation. In this
chapter, we present a framework for statistical analysis and visualization of large complex
data: divide and recombine (D&R).

In D&R, a large dataset is broken into pieces in a meaningful way, statistical or
visual methods are applied to each subset in an embarrassingly parallel fashion, and the
results of these computations are recombined in a manner that yields a statistically valid result. We introduce D&R in Section 3.3 and discuss various division and recombination schemes.

D&R provides the foundation for Trelliscope, an approach to detailed visualization of large complex data. Trelliscope is a multipanel display system based on the concepts of Trellis display. In Trellis display, data are broken into subsets, a visualization method is applied to each subset, and the resulting panels are arranged in a grid, facilitating meaningful visual comparison between panels. Trelliscope extends Trellis by providing a multipanel display system that can handle a very large number of panels and provides a paradigm for effectively viewing the panels. Trelliscope is introduced in Section 3.4.

In Section 3.5, we present an ongoing open source project working toward the goal of providing a computational framework for D&R and Trelliscope, called Tessera. Tessera provides an R interface that flexibly ties to scalable back ends such as Hadoop or Spark. The analyst programs entirely in R, large distributed data objects (DDOs) are represented as native R objects, and D&R and Trelliscope operations are made available through simple R commands.

3.2 Context: Deep Analysis of Large Complex Data

There are many domains that touch data, and hence several definitions of the terms data analysis, visualization, and big data. It is useful therefore to first set the proper context for the approaches we present in this chapter. Doing so will identify the attributes necessary for an appropriate methodology and computational environment.

3.2.1 Deep Analysis

The term analysis can mean many things. Often, the term is used for tasks such as computing summaries and presenting them in a report, running a database query, processing data through a set of predetermined analytical or machine learning routines. While these are useful, there is in them an inherent notion of knowing a priori what is the right thing to be done to the data. However, data most often do not come with a model. The type of analysis we strive to address is that which we have most often encountered when faced with large complex datasets—analysis where we do not know what to do with the data and we need to find the most appropriate mathematical way to represent the phenomena generating the data. This type of analysis is very exploratory in nature. There is a lot of trial and error involved. We iterate between hypothesizing, fitting, and validating models. In this context, it is natural that analysis involves great deal of visualization, which is one of the best ways to drive this iterative process, from generating new ideas to assessing the validity of hypothesized models, to presenting results. We call this type of analysis deep analysis.

While almost always useful in scientific disciplines, deep exploratory analysis and model building is not always the right approach. When the goal is pure classification or prediction accuracy, we may not care as much about understanding the data as we do about simply choosing the algorithm with the best performance. But even in these cases, a more open-ended approach that includes exploration and visualization can yield vast improvements. For instance, consider the case where one might choose the best performer from a collection of algorithms, which are all poor performers due to their lack of suitability to the data, and this lack of suitability might be best determined through exploration. Or consider an
analyst with domain expertise who might be able to provide insights based on explorations that vastly improve the quality of the data or help the analyst look at the data from a new perspective. In the words of the father of exploratory data analysis, John Tukey:

Restricting one's self to planned analysis – failing to accompany it with exploration – loses sight of the most interesting results too frequently to be comfortable. [17]

This discussion of deep analysis is nothing new to the statistical practitioner, and to such our discussion may feel a bit belabored. But in the domain of big data, its practice severely lags behind the other analytical approaches and is often ignored, and hence deserves attention.

### 3.2.2 Large Complex Data

Another term that pervades the industry is big data. As with the term analysis, this also can mean a lot of things. We tend to use the term large complex data to describe data that poses the most pressing problems for deep analysis. Large complex data can have any or all of the following attributes: a large number of records, many variables, complex data structures that are not readily put into a tabular form, or intricate patterns and dependencies that require complex models and methods of analysis.

Size alone may not be an issue if the data are not complex. For example, in the case of tabular i.i.d data with a very large number of rows and a small number of variables, analyzing a small sample of the data will probably suffice. It is the complexity that poses more of a problem, regardless of size.

When data are complex in either structure or phenomena generating the data, we need to analyze the data in detail. Summaries or samples will generally not suffice. For instance, take the case of analyzing computer network traffic for thousands of computers in a large enterprise. Because of the large number of actors in a computer network, many of which are influenced by human behavior, there are so many different kinds of activity that can be observed and modeled such that downsampling or trying to summarize will surely result in lost information. We must address the fact that we need statistical approaches to deep analysis that can handle large volumes complex data.

### 3.2.3 What is Needed for Analysis of Large Complex Data?

Now that we have provided some context, it is useful to discuss what is required to effectively analyze large complex data in practice. These requirements provide the basis for the approaches proposed in the remainder of the chapter.

By our definition of deep analysis, many requirements are readily apparent. First, due to the possibility of having several candidate models or hypotheses, we must have at our fingertips a library of the thousands of statistical, machine learning, and visualization methods. Second, due to the need for efficient iteration through the specification of different models or visualizations, we must also have access to a high-level interactive statistical computing software environment in which simple commands can execute complex algorithms or data operations and in which we can flexibly handle data of different structures.

There are many environments that accommodate these requirements for small datasets, one of the most prominent being R, which is the language of choice for our implementation and discussions in this chapter. We cannot afford to lose the expressiveness of the high-level computing environment when dealing with large data. We would like to be able to handle data and drive the analysis from a high-level environment while transparently harnessing
distributed storage and computing frameworks. With big data, we need a statistical methodology that will provide access to the thousands of methods available in a language such as R without the need to reimplement them. Our proposed approach is D&R, described in Section 3.3.

3.3 Divide and Recombine

D&R is a statistical framework for data analysis based on the popular split-apply-combine paradigm [20]. It is suited for situations where the number of cases outnumbers the number of variables. In D&R, cases are partitioned into manageable subsets in a meaningful way for the analysis task at hand, analytic methods (e.g., fitting a model) are applied to each subset independently, and the results are recombined (e.g., averaging the model coefficients from each subset) to yield a statistically valid—although not always exact—result. The key to D&R is that by computing independently on small subsets, we can scalably leverage all of the statistical methods already available in an environment like R.

Figure 3.1 shows a visual illustration of D&R. A large dataset is partitioned into subsets where each subset is small enough to be manageable when loaded into memory in a single process in an environment such as R. Subsets are persistent, and can be stored across multiple disks and nodes in a cluster. After partitioning the data, we apply an analytic method in parallel to each individual subset and merge the results of these computations in the recombination step. A recombination can be an aggregation of analytic outputs to provide a statistical model result. It can yield a new (perhaps smaller) dataset to be used for further analysis, or it can even be a visual display, which we will discuss in Section 3.4.

In the remainder of this section, we provide the necessary background for D&R, but we point readers to [3,6] for more details.

FIGURE 3.1
Diagram of the D&R statistical and computational framework.

Downloaded By: 10.3.98.93 At: 00:52 10 Oct 2018; For: 9781482249088, chapter3, 10.1201/b19567-6
3.3.1 Division Approaches

Divisions are constructed by either conditioning-variable division or replicate division.

Replicate division creates partitions using random sampling of cases without replacement, and is useful for many analytic recombination methods that will be touched upon in Section 3.3.2.

Very often the data are embarrassingly divisible, meaning that there are natural ways to break the data up based on the subject matter, leading to a partitioning based on one or more of the variables in the data. This constitutes a conditioning-variable division. As an example, suppose we have 25 years of 90 daily financial variables for 100 banks in the United States. If we wish to study the behavior of individual banks and then make comparisons across banks, we would partition the data by bank. If we are interested in how all banks behave together over the course of each year, we could partition by year. Other aspects such as geography and type or size of bank might also be valid candidates for a division specification.

A critical consideration when specifying a division is to obtain subsets that are small enough to be manageable when loaded into memory, so that they can be processed in a single process in an environment like R. Sometimes, a division driven by subject matter can lead to subsets that are too large. In this case, some creativity on the part of the analyst must be applied to further break down the subsets.

The persistence of a division is important. Division is an expensive operation, as it can require shuffling a large amount data around on a cluster. A given partitioning of the data is typically reused many times while we are iterating over different analytical methods. For example, after partitioning financial data by bank, we will probably apply many different analytical and visual methods to that partitioning scheme until we have a model we are happy with. We do not want to incur the cost of division each time we want to try a new method.

Keeping multiple persistent copies of data formatted in different ways for different analysis purposes is a common practice with small data, and for a good reason. Having the appropriate data structure for a given analysis task is critical, and the complexity of the data often means that these structures will be very different depending on the task (e.g., not always tabular). Thus, it is not generally sufficient to simply have a single table that is indexed in different ways for different analysis tasks. The notion of possibly creating multiple copies of a large dataset may be alarming to a database engineer, but should not be surprising to a statistical practitioner, as it is a standard practice with small datasets to have different copies of the data for different purposes.

3.3.2 Recombination Approaches

Just as there are different ways to divide the data, there are also different ways to recombine them, as outlined in Figure 3.1. Typically for conditioning-variable division, a recombination is a collation or aggregation of an analytic method applied to each subset. The results often are small enough to investigate on a single workstation or may serve as the input for further D&R operations.

With replicate division, the goal is usually to approximate an overall model fit to the entire dataset. For example, consider a D&R logistic regression where the data are randomly partitioned, we apply R’s glm() method to each subset independently, and then we average the model coefficients. The result of a recombination may be an approximation of the exact result had we been able to process the data as a whole, as in this example, but a potentially
small loss in accuracy is often a small price to pay for the simple, fast computation. A more lengthy discussion of this can be found in [6], and some interesting research is discussed in Section 3.3.4.

Another crucial recombination approach is visual recombination, which we discuss in Section 3.4.

### 3.3.3 Data Structures and Computation

In addition to the methodological concerns of D&R, there are also computational concerns. Here, we define the minimal conditions required for a D&R computational environment.

Data structures are the first important consideration. Division methods can result in partitions where subsets can have nontabular data structures. A generic storage mechanism for data with potentially arbitrary structure is a *key-value store*. In a key-value store, each key-value pair constitutes a subset of the data, and typically the key is a unique identifier or object that describes the subset, and the value contains the data for the subset. When the data are large, there are many distributed key-value store technologies that might be utilized such as the Hadoop Distributed File System (HDFS) [14]. It is important to have the ability to store data in a persistent state on these systems, and useful to have fast random lookup of any subset by key.

A data set that has been partitioned into a collection of subsets stored as key-value pairs, potentially distributed across machines or disks of a cluster, is called a *distributed data object* (DDO). When the data for each subset of a DDO is a slice of a data frame, we can more specifically call such an object a *distributed data frame*.

For D&R computation, we need to compute on a distributed key-value store in parallel and in a way that allows us to shuffle data around in a division step, apply analytic methods to the subsets, and combine results in a recombination step. Environments that implement MapReduce [4] are sufficient for this. In MapReduce, a *map* operation is applied to a collection of input key-value pairs in parallel. The map step outputs a transformed set of key-value pairs, and these results are *shuffled*, with the results being grouped by the map output keys. Each collection of data for a unique map output key is then sent to a *reduce* operation, again executed in parallel. All D&R operations can be carried out through this approach. Systems such as Hadoop [8] that run MapReduce on HDFS are a natural fit for D&R computation.

D&R is sometimes confused as being equivalent to MapReduce. This is not the case, but D&R operations are *carried out* by MapReduce. For example, a division is typically achieved by a single MapReduce operation—the map step reassigns records to new partitions, the shuffle groups the map output by partition assignment, and the reduce collates each partition. The application of an analytic method followed by recombination is a separate MapReduce operation, where the analytic method is applied in the map and the recombination is done in the shuffle and reduce. Recall that division is independent of recombination and typically a division persists and is used for many different recombinations.

A common question that arises when discussing the use of systems such as Hadoop for D&R computation is that of Hadoop being a *batch* processing system. As we discussed in Section 3.2, this fits the purpose of the type of deep analysis we are doing, which is typically *offline* analysis of historical data. Of course, the models and algorithms that result from a deep analysis be adapted and integrated into a real-time processing environment, but that is not the use case for D&R, where we are doing the actual discovery and validation of the algorithms. But speed is still important, and much work is being done to improve the speed of these type of systems, such as Spark [23], which can keep the data in memory distributed across machines, avoiding the most expensive time cost present in Hadoop: reading and writing data to disk multiple times throughout a MapReduce operation.
3.3.4 Research in D&R

The D&R paradigm provides fertile ground for new research in statistical analysis of big data. Many ideas in D&R are certainly not new, and there is a lot of research independent of ours that fits in the D&R paradigm that should be leveraged in a D&R computational environment.

The key for D&R research is to find pairs of division/recombination procedures that provide good results. Existing independent research that relates to D&R that can serve as a platform for new research includes Bayesian consensus Monte Carlo [13], bag of little bootstraps [11], alternating direction method of multipliers [2], and scatter matrix stability weighting [10]. It is important to note that we seek methods that require a very minimal amount of iteration, preferably none, as every MapReduce step can be very costly. There are also many domains of statistics that remain to be studied, including spatial, time series, and nonparametric statistics. There is a great opportunity for interesting research in this area.

3.4 Trelliscope

Visualization is crucial throughout the analysis process. This could not be more true than in the case of large complex data. Typical approaches to visualizing large data are either very specialized tools for data from a specific domain, or schemes that aggregate various aspects of the data into a single plot. Specialized tools generally do not work for deep analysis because of the flexibility required to make any imaginable plot we might deem useful. Visualizations of summaries are indispensable, but alone are not enough. Summaries can hide important information, particularly when the data are complex. We need flexible, detailed visualization that scales. Trelliscope is an approach that addresses these needs.

3.4.1 Trellis Display/Small Multiples

Trelliscope is based on the idea of Trellis display [1]. In Trellis display, the data are split into meaningful subsets, usually conditioning on variables of the dataset, and a visualization method is applied to each subset. The image for each subset is called a panel. Panels are arranged in an array of rows, columns, and pages, resembling a garden trellis.

The notion of conditioning in Trellis display manifests itself in several other plotting systems, under names such as faceted or small multiple plots. Trellis display for small data has proven to be very useful for uncovering the structure of data even when the structure is complicated and in making important discoveries in data not appreciated in the original analyses [1]. There are several reasons for its success. One is that it allows the analyst to break a larger or higher dimensional dataset into a series of two-dimensional plots, providing more visual detail. Second is the ability to make comparisons across different subsets. Edward Tufte, in discussing multipanel displays as small multiples, supports this benefit, stating that once a viewer understands one panel, they immediately understand all the panels, and that when arranged adjacent, panels directly depict comparisons to reveal repetition and change, pattern and surprise [16].

3.4.2 Scaling Trellis Display

The notion of conditioning to obtain a multipanel display maps naturally to D&R. We can divide the data in any manner and specify a panel plotting function to be applied to each subset with the recombination being a collation of the panels presented to view. But there
is a problem here. Typically in D&R, we are dealing with very large datasets, and typically divisions can result in thousands to hundreds of thousands of subsets. A multipanel display would have as many panels. This can happen even with data of small to moderate size. It is easy to generate thousands of plots, but it is not feasible to look at all of them.

The problem of having more visualizations than humanly possible to cognitively consume is a problem that pioneering statistician John Tukey realized decades ago. Since it is impossible to view every display in a large collection, he put forth the idea of asking the computer to sort out which ones to present by judging the relative interest of showing each of them [18]. He proposed computing diagnostic quantities with the help of a computer to determine which plots to show. In his words, “it seems natural to call such computer guiding diagnostics cognostics. We must learn to choose them, calculate them, and use them. Else we drown in a sea of many displays” [18]. Hence, we use the term cognostic to mean a single metric computed about a single plot that captures a behavior of interest that can be used by the computer to bring interesting plots to our attention. For any collection of plots, we may be interested in several behaviors, and will therefore compute several cognostics.

There has been interesting research on cognostics for scatterplots, called scagnostics, that has yielded metrics that quantify different shapes or behaviors that might present themselves in a scatterplot [21,22]. Beyond scatterplots, there are many useful metrics that might be computed to guide the selection of plots. Many such cognostics may be context-dependent. For example, when dealing with displays of quantile plots, metrics such as the median, first and third quartile, and range might be good cognostics. Or when dealing with time series plots, calculations of slope, autocorrelation coefficient, variance of first-order differences, and so on might be good cognostics. Often, choosing a useful cognostic is heavily based on the subject matter of the data and the particular plot being made. For example, consider a collection quantile plots, one for each county in the United States, showing the age distribution for male and female. For each quantile plot, a useful cognostic might be the difference in median age between genders.

How does the computer use cognostics to determine which plots we should view? There are many possibilities. One is ranking or sorting the plots based on the cognostics. For example, we can effectively understand the data at extremes of the median age difference by gender by calling up panels with the largest or smallest absolute age difference cognostic. Another possibility is sampling plots across the distribution of a set of cognostics, for example, looking at a representative set of panels that spans the age difference distribution. Another action is to filter panels of a display to only panels that have cognostics in a range of interest. There are many possible effective ways to get a good representation of interesting panels in a large display, particularly when combining cognostics. For example, if we want to find interesting panels with respect to a collection of cognostics, we can, for example, compute projections of the set of cognostics into two dimensions, plot the projection as a scatterplot, and select panels based on interesting regions of the projection. Another possibility is to apply a clustering algorithm to a set of cognostics to view panels representative of each of the clusters.

### 3.4.3 Trelliscope

Trelliscope is a computational system that implements the ideas of large-scale multipanel display with cognostics for effective detailed display of large complex data [9]. In Trelliscope, the analyst creates a division of the data, specifies a plotting function to be applied to each subset, and also specifies a cognostics function that for each subset will compute a set of metrics. The cognostics are collected from every subset and are used in an interactive viewer allowing the user to specify different actions with the cognostics (sort, filter, and sample) in different dimensions, thereby arranging panels in a desired way.
Trelliscope’s multipanel visual recombination system has been influenced by work in visualization databases [7]. A visualization database can be thought of as a large collection of many different displays that are created throughout the course of a data analysis, many of which might be multipanel displays. In addition to needing a way to effectively view panels within a single display, we also need ways to store and organize all of the visual artifacts that are created during an analysis. Trelliscope provides a system for storing and tagging displays in a visualization database, so that they can easily be sorted through and retrieved for viewing or sharing with others.

3.5 Tessera: Computational Environment for D&R

Tessera is an open source project that implements D&R in a familiar high-level language at the front end and ties to various distributed storage and computing at back ends. Development of Tessera began out of necessity when tackling applied statistical problems involving big data by researchers at Purdue University and has expanded to include statisticians and computer scientists at Pacific Northwest National Laboratory. Development of Tessera has been part of the US government big data initiatives including funding from Defense Advanced Research Projects Agency and Department of Homeland Security. In this chapter, we introduce and discuss the components of Tessera. For several examples of how to use Tessera, the reader is encouraged to visit the Tessera website: http://tessera.io/. The website is a more suitable medium than this chapter for providing interactive reproducible tutorials and up-to-date examples.

3.5.1 Front End

The front end of Tessera is the R statistical programming environment [12]. R’s elegant design makes programming with data very efficient, and R has a massive collection of statistics, machine learning, and visualization methods. Further supporting this choice is the large R user community and its popularity for statistical analysis of small data. We note, however, that the D&R approach could easily be implemented in other languages as well.

Tessera has two front-end R packages. The first is datadr, which is a domain-specific language for D&R operations. This package provides commands that make the specification of divisions, analytic methods, and recombinations easy. Its interface is abstracted from different back-end choices, so that commands are the same whether running on Hadoop or on a single workstation. In datadr, large DDOs behave like native R objects. In addition to division and recombination methods, datadr also provides several indispensable tools for reading and manipulating data, as well as a collection of division-independent methods that can compute things such as aggregations, quantiles, or summaries across the entire dataset, regardless of how the data are partitioned.

The second front-end package is trelliscope. This package provides a visualization database system for storing and managing visual artifacts of an analysis, as well as a visual recombination system for creating and viewing very large multipanel displays. Trelliscope displays can be viewed in an interactive viewer that provides several modes for sorting, filtering, and sampling panels of a display.

3.5.2 Back Ends

The datadr and trelliscope packages are interfaces to distributed computing back ends. They were designed to be extensible to be able to harness new technology as it comes along.
As discussed in Section 3.3.3, a back end for D&R needs to be able to do MapReduce over a distributed key-value store. Additionally, there needs to be a mechanism that connects R to these back ends, as we need to be able to run R code inside of the map and reduce steps. Tessera currently has implemented support for four different back ends which we will discuss in the following sections: in-memory storage with R MapReduce (small), local disk storage with multicore R MapReduce (medium), HDFS storage with Hadoop MapReduce via RHIPE (large), and HDFS storage with Spark MapReduce via SparkR (large).

3.5.2.1 Small Scale: In-Memory Storage with R MapReduce

D&R is not just useful for large data. When dealing with small datasets that fit within R’s memory comfortably, the in-memory Tessera back end can be used to store and analyze DDOs (although in this case, the term distributed does not really apply). A nonparallel version of MapReduce has been implemented in R to handle all D&R operations for this back end. A nice feature of the in-memory back end is that the only requirements are the datadr and trelliscope packages, making it an easy way to start getting familiar with Tessera without the need for a cluster of machines or to install other back end components such as Hadoop, which can be a difficult task.

This back end is useful for very small datasets, which in our current experience has typically meant tens or hundreds of megabytes or less. As data gets larger than this, even though it may still be much smaller than the available memory on the machine, the lack of parallel computation and the growing size of objects in memory from making copies of subsets for each thread becomes a problem. Even with parallel in-memory computing in R, we have found the strategy of throwing more memory at the problem to not scale.

3.5.2.2 Medium Scale: Local Disk Storage with Multicore R MapReduce

When the data are large enough that they are difficult to handle in memory, another option is to use the local disk back end. The key-value store in this case is simply a collection of files stored in a directory on a hard drive, one file per subset. Computation is achieved with a parallel version of MapReduce implemented in R, making use of R’s parallel package [12].

As with the in-memory back end, this back end is also useful for a single-workstation setup, although it could conceptually be used in a single network of workstations [15] setting where every workstation has access to the disk.

We have found this back end to be useful for data that is in the range of a few gigabytes.

3.5.2.3 Large Scale: HDFS and Hadoop MapReduce via RHIPE

RHIPE is the R and Hadoop Integrated Programming Environment [5]. RHIPE is an R interface to Hadoop, providing everything from reading, writing, and manipulating data on HDFS, to running Hadoop MapReduce jobs completely from within the R console. Tessera uses RHIPE to access the Hadoop back end for scaling to very large datasets.

Hadoop has been known to scale to data in the range of petabytes. Leveraging RHIPE, Tessera should be able to scale similarly, although the largest amount of data we have routinely used for it is in the multi-terabyte range.

3.5.2.4 Large Scale in Memory: Spark

Another large-scale back-end option that is becoming very popular is Spark. Spark is a general distributed execution engine that allows for keeping data in memory, greatly improving performance. Spark can use HDFS as a distributed storage mechanism. It provides many more data operations than MapReduce, but these operations are a strict superset of MapReduce, and therefore Spark is a suitable back end for Tessera. The additional data
operations are a bonus in that the same data being used for D&R can be used for other parallel computing purposes. Tessera connects to Spark using the SparkR package [19], which exposes the Spark API in the R console.

Support in Tessera for Spark at the time of this writing is very experimental—it has been implemented and works, and adding it to Tessera is a testament of Tessera’s flexibility in being back end agnostic, but it has only been tested with rather small datasets.

3.6 Discussion

In this chapter, we have presented one point of view regarding methodology and computational tools for deep statistical analysis and visualization of large complex data. D&R is attractive because of its simplicity and its ability to make a wide array of methods available without needing to implement scalable versions of them. D&R also builds on approaches that are already very popular with small data, particularly implementations of the split-apply-combine paradigm such as the plyr and dplyr R packages. D&R as implemented in datadr is future proof because of its design, enabling adoption of improved back-end technology as it comes along. All of these factors give D&R a high chance of success. However, there is a great need for more research and software development to extend D&R to more statistical domains and make it easier to program.

References


