PRINCIPLES AND PRACTICE OF BIG DATA

PREPARING, SHARING, AND ANALYZING COMPLEX INFORMATION

SECOND EDITION

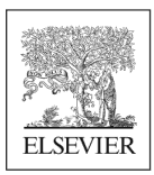
JULES J. BERMAN

Principles and Practice of Big Data

Preparing, Sharing, and Analyzing Complex Information

Second Edition

Jules J. Berman



ACADEMIC PRESS
An imprint of Elsevier

Academic Press is an imprint of Elsevier
125 London Wall, London EC2Y 5AS, United Kingdom
525 B Street, Suite 1650, San Diego, CA 92101, United States
50 Hampshire Street, 5th Floor, Cambridge, MA 02139, United States
The Boulevard, Langford Lane, Kidlington, Oxford OX5 1GB, United Kingdom

Notices
Knowledge and best practice in this field are constantly changing. As new research and experience broaden our understanding, changes in research methods, professional practices, or medical treatment may become necessary.

Practitioners and researchers must always rely on their own experience and knowledge in evaluating and using any information, methods, compounds, or experiments described herein. In using such information or methods they should be mindful of their own safety and the safety of others, including parties for whom they have a professional responsibility.

To the fullest extent of the law, neither the Publisher nor the authors, contributors, or editors, assume any liability for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions, or ideas contained in the material herein.

Library of Congress Cataloging-in-Publication Data
A catalog record for this book is available from the Library of Congress

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library

ISBN: 978-0-12-815609-4

Publisher: Mara Conner
Acquisition Editor: Mara Conner
Editorial Project Manager: Mariana L. Kuhl
Production Project Manager: Punithavathy Govindaradjane
Cover Designer: Matthew Limbert

Typeset by SPi Global, India

Contents

This page intentionally left blank

About the Author

Jules J. Berman received two baccalaureate degrees from MIT; in Mathematics, and in Earth and Planetary Sciences. He holds a PhD from Temple University, and an MD, from the University of Miami. He was a graduate student researcher in the Fels Cancer Research Institute, at Temple University, and at the American Health Foundation in Valhalla, New York. His postdoctoral studies were completed at the US National Institutes of Health, and his residency was completed at the George Washington University Medical Center in Washington, DC. Dr. Berman served as Chief of Anatomic Pathology, Surgical Pathology, and Cytopathology at the Veterans Administration Medical Center in Baltimore, Maryland, where he held joint appointments at the University of Maryland Medical Center and at the Johns Hopkins Medical Institutions. In 1998, he transferred to the US National Institutes of Health, as a Medical Officer, and as the Program Director for Pathology Informatics in the Cancer Diagnosis Program at the National Cancer Institute. Dr. Berman is a past president of the Association for Pathology Informatics, and the 2011 recipient of the Association's Lifetime Achievement Award. He has first-authored over 100 scientific publications and has written more than a dozen books in the areas of data science and disease biology. Several of his most recent titles, published by Elsevier, include:

Taxonomic Guide to Infectious Diseases: Understanding the Biologic Classes of Pathogenic Organisms (2012)
Principles of Big Data: Preparing, Sharing, and Analyzing Complex Information (2013)
Rare Diseases and Orphan Drugs: Keys to Understanding and Treating the Common Diseases (2014)
Repurposing Legacy Data: Innovative Case Studies (2015)
Data Simplification: Taming Information with Open Source Tools (2016)
Precision Medicine and the Reinvention of Human Disease (2018)

This page intentionally left blank

Author's Preface to Second Edition

Everything has been said before, but since nobody listens we have to keep going back and beginning all over again.
Andre Gide

Good science writers will always jump at the chance to write a second edition of an earlier work. No matter how hard they try, that first edition will contain inaccuracies and misleading remarks. Sentences that seemed brilliant when first conceived will, with the passage of time, transform into examples of intellectual overreaching. Points too trivial to include in the original manuscript may now seem like profundities that demand a full explanation. A second edition provides rueful authors with an opportunity to correct the record.

When the first edition of Principles of Big Data was published in 2013 the field was very young and there were few scientists who knew what to do with Big Data. The data that kept pouring in was stored, like wheat in silos, throughout the planet. It was obvious to data managers that none of that stored data would have any scientific value unless it was properly annotated with metadata, identifiers, timestamps, and a set of basic descriptors. Under these conditions, the first edition of the Principles of Big Data stressed the proper and necessary methods for collecting, annotating, organizing, and curating Big Data. The process of preparing Big Data comes with its own unique set of challenges, and the First Edition was peppered with warnings and exhortations intended to steer readers clear of disaster.

It is now five years since the first edition was published and there have since been hundreds of books written on the subject of Big Data. As a scientist, it is disappointing to me that the bulk of Big Data, today, is focused on issues of marketing and predictive analytics (e.g., "Who is likely to buy product x, given that they bought product y two weeks previously?"); and machine learning (e.g., driverless cars, computer vision, speech recognition). Machine learning relies heavily on hyped up techniques such as neural networks and deep learning; neither of which are leading to fundamental laws and principles that simplify and broaden our understanding of the natural world and the physical universe. For the most part, these techniques use data that is relatively new (i.e., freshly collected), poorly annotated (i.e., provided with only the minimal information required for one particular analytic process), and not deposited for public evaluation or for re-use. In short, Big Data has followed the path of least resistance, avoiding most of the tough issues raised in the first edition of this book; such as the importance of sharing data with the public, the value of finding relationships (not similarities) among data objects, and the heavy, but inescapable, burden of creating robust, immortal, and well-annotated data.

It was certainly my hope that the greatest advances from Big Data would come as fundamental breakthroughs in the realms of medicine, biology, physics, engineering, and chemistry. Why has the focus of Big Data shifted from basic science over to machine learning? It may have something to do with the fact that no book, including the first edition of this book, has provided readers with the methods required to put the principles of Big Data into practice. In retrospect, it was not sufficient to describe a set of principles and then expect readers to invent their own methodologies.

Consequently, in this second edition, the publisher has changed the title of the book from "The Principles of Big Data," to "The Principles AND PRACTICE of Big Data." Henceforth and herein, recommendations are accompanied by the methods by which those recommendations can be implemented. The reader will find that all of the methods for implementing Big Data preparation and analysis are really quite simple. For the most part, computer methods require some basic familiarity with a programming language, and, despite misgivings, Python was chosen as the language of choice. The advantages of Python are:

- Python is a no-cost, open source, high-level programming language that is easy to acquire, install, learn, and use, and is available for every popular computer operating system.
- Python is extremely popular, at the present time, and its popularity seems to be increasing.
- Python distributions (such as Anaconda) come bundled with hundreds of highly useful modules (such as numpy, matplot, and scipy).
- Python has a large and active user group that has provided an extraordinary amount of documentation for Python methods and modules.
- Python supports some object-oriented techniques that will be discussed in this new edition

As everything in life, Python has its drawbacks:

- The most current versions of Python are not backwardly compatible with earlier versions. The scripts and code snippets included in this book should work for most versions of Python 3.x, but may not work with Python versions 2.x and earlier, unless the reader is prepared to devote some time to tweaking the code. Of course, these short scripts and snippets are intended as simplified demonstrations of concepts, and must not be construed as application-ready code.
- The built-in Python methods are sometimes maximized for speed by utilizing Random Access Memory (RAM) to hold data structures, including data structures built through iterative loops. Iterations through Big Data may exhaust available RAM, leading to the failure of Python scripts that functioned well with small data sets.
- Python's implementation of object orientation allows multiclass inheritance (i.e., a class can be the subclass of more than one parent class). We will describe why this is problematic, and the compensatory measures that we must take, whenever we use our Python programming skills to understand large and complex sets of data objects.

The core of every algorithm described in the book can be implemented in a few lines of code, using just about any popular programming language, under any operating system,

on any modern computer. Numerous Python snippets are provided, along with descriptions of free utilities that are widely available on every popular operating system. This book stresses the point that most data analyses conducted on large, complex data sets can be achieved with simple methods, bypassing specialized software systems (e.g., parallelization of computational processes) or hardware (e.g., supercomputers). Readers who are completely unacquainted with Python may find that they can read and understand Python code, if the snippets of code are brief, and accompanied by some explanation in the text. In any case, readers who are primarily concerned with mastering the principles of Big Data can skip the code snippets without losing the narrative thread of the book.

This second edition has been expanded to stress methodologies that have been overlooked by the authors of other books in the field of Big Data analysis. These would include:

– **Data preparation.**

How to annotate data with metadata and how to create data objects composed of triples. The concept of the triple, as the fundamental conveyor of meaning in the computational sciences, is fully explained.

– **Data structures of particular relevance to Big Data**

Concepts such as triplestores, distributed ledgers, unique identifiers, timestamps, concordances, indexes, dictionary objects, data persistence, and the roles of one-way hashes and encryption protocols for data storage and distribution are covered.

– **Classification of data objects**

How to assign data objects to classes based on their shared relationships, and the computational roles filled by classifications in the analysis of Big Data will be discussed at length.

– **Introspection**

How to create data objects that are self-describing, permitting the data analyst to group objects belonging to the same class and to apply methods to class objects that have been inherited from their ancestral classes.

– **Algorithms that have special utility in Big Data preparation and analysis**

How to use one-way hashes, unique identifier generators, cryptographic techniques, timing methods, and time stamping protocols to create unique data objects that are immutable (never changing), immortal, and private; and to create data structures that facilitate a host of useful functions that will be described (e.g., blockchains and distributed ledgers, protocols for safely sharing confidential information, and methods for reconciling identifiers across data collections without violating privacy).

– **Tips for Big Data analysis**

How to overcome many of the analytic limitations imposed by scale and dimensionality, using a range of simple techniques (e.g., approximations, so-called back-of-the-envelope

tricks, repeated sampling using a random number generator, Monte Carlo simulations, and data reduction methods).

– **Data reanalysis, data repurposing, and data sharing**

Why the first analysis of Big Data is almost always incorrect, misleading, or woefully incomplete, and why data reanalysis has become a crucial skill that every serious Big Data analyst must acquire. The process of data reanalysis often inspires repurposing of Big Data resources. Neither data reanalysis nor data repurposing can be achieved unless and until the obstacles to data sharing are overcome. The topics of data reanalysis, data repurposing, and data sharing are explored at length.

Comprehensive texts, such as the second edition of the Principles and Practice of Big Data, are never quite as comprehensive as they might strive to be; there simply is no way to fully describe every concept and method that is relevant to a multi-disciplinary field, such as Big Data. To compensate for such deficiencies, there is an extensive Glossary section for every chapter, that defines the terms introduced in the text, providing some explanation of the relevance of the terms for Big Data scientists. In addition, when techniques and methods are discussed, a list of references that the reader may find useful, for further reading on the subject, is provided. Altogether, the second edition contains about 600 citations to outside references, most of which are available as free downloads. There are over 300 glossary items, many of which contain short Python snippets that readers may find useful.

As a final note, this second edition uses case studies to show readers how the principles of Big Data are put into practice. Although case studies are drawn from many fields of science, including physics, economics, and astronomy, readers will notice an overabundance of examples drawn from the biological sciences (particularly medicine and zoology). The reason for this is that the taxonomy of all living terrestrial organisms is the oldest and best Big Data classification in existence. All of the classic errors in data organization, and in data analysis, have been committed in the field of biology. More importantly, these errors have been documented in excruciating detail and most of the documented errors have been corrected and published for public consumption. If you want to understand how Big Data can be used as a tool for scientific advancement, then you must look at case examples taken from the world of biology, a well-documented field where everything that can happen has happened, is happening, and will happen. Every effort has been made to limit Case Studies to the simplest examples of their type, and to provide as much background explanation as non-biologists may require.

Principles and Practice of Big Data, Second Edition, is devoted to the intellectual conviction that the primary purpose of Big Data analysis is to permit us to ask and answer a wide range of questions that could not have been credibly approached with small sets of data. There is every reason to hope that the readers of this book will soon achieve scientific breakthroughs that were beyond the reach of prior generations of scientists. Good luck!

Author's Preface to First Edition

We can't solve problems by using the same kind of thinking we used when we created them.
Albert Einstein

Data pours into millions of computers every moment of every day. It is estimated that the total accumulated data stored on computers worldwide is about 300 exabytes (that's 300 billion gigabytes). Data storage increases at about 28% per year. The data stored is peanuts compared to data that is transmitted without storage. The annual transmission of data is estimated at about 1.9 zettabytes or 1,900 billion gigabytes [1]. From this growing tangle of digital information, the next generation of data resources will emerge.

As we broaden our data reach (i.e., the different kinds of data objects included in the resource), and our data timeline (i.e., accruing data from the future and the deep past), we need to find ways to fully describe each piece of data, so that we do not confuse one data item with another, and so that we can search and retrieve data items when we need them. Astute informaticians understand that if we fully describe everything in our universe, we would need to have an ancillary universe to hold all the information, and the ancillary universe would need to be much larger than our physical universe.

In the rush to acquire and analyze data, it is easy to overlook the topic of data preparation. If the data in our Big Data resources are not well organized, comprehensive, and fully described, then the resources will have no value. The primary purpose of this book is to explain the principles upon which serious Big Data resources are built. All of the data held in Big Data resources must have a form that supports search, retrieval, and analysis. The analytic methods must be available for review, and the analytic results must be available for validation.

Perhaps the greatest potential benefit of Big Data is its ability to link seemingly disparate disciplines, to develop and test hypothesis that cannot be approached within a single knowledge domain. Methods by which analysts can navigate through different Big Data resources to create new, merged data sets, will be reviewed.

What exactly, is Big Data? Big Data is characterized by the three V's: volume (large amounts of data), variety (includes different types of data), and velocity (constantly accumulating new data) [2]. Those of us who have worked on Big Data projects might suggest throwing a few more v's into the mix: vision (having a purpose and a plan), verification (ensuring that the data conforms to a set of specifications), and validation (checking that its purpose is fulfilled).

Many of the fundamental principles of Big Data organization have been described in the "metadata" literature. This literature deals with the formalisms of data description (i.e., how to describe data); the syntax of data description (e.g., markup languages such as eXtensible Markup Language, XML); semantics (i.e., how to make computer-parsable statements that convey meaning); the syntax of semantics (e.g., framework specifications such as Resource Description Framework, RDF, and Web Ontology Language, OWL); the creation of data objects that hold data values and self-descriptive information; and the deployment of ontologies, hierarchical class systems whose members are data objects.

The field of metadata may seem like a complete waste of time to professionals who have succeeded very well, in data-intensive fields, without resorting to metadata formalisms. Many computer scientists, statisticians, database managers, and network specialists have no trouble handling large amounts of data, and they may not see the need to create a strange new data model for Big Data resources. They might feel that all they really need is greater storage capacity, distributed over more powerful computers that work in parallel with one another. With this kind of computational power, they can store, retrieve, and analyze larger and larger quantities of data. These fantasies only apply to systems that use relatively simple data or data that can be represented in a uniform and standard format. When data is highly complex and diverse, as found in Big Data resources, the importance of metadata looms large. Metadata will be discussed, with a focus on those concepts that must be incorporated into the organization of Big Data resources. The emphasis will be on explaining the relevance and necessity of these concepts, without going into gritty details that are well covered in the metadata literature.

When data originates from many different sources, arrives in many different forms, grows in size, changes its values, and extends into the past and the future, the game shifts from data computation to data management. I hope that this book will persuade readers that faster, more powerful computers are nice to have, but these devices cannot compensate for deficiencies in data preparation. For the foreseeable future, universities, federal agencies, and corporations will pour money, time, and manpower into Big Data efforts. If they ignore the fundamentals, their projects are likely to fail. On the other hand, if they pay attention to Big Data fundamentals, they will discover that Big Data analyses can be performed on standard computers. The simple lesson, that data trumps computation, will be repeated throughout this book in examples drawn from well-documented events.

There are three crucial topics related to data preparation that are omitted from virtually every other Big Data book: identifiers, immutability, and introspection.

A thoughtful identifier system ensures that all of the data related to a particular data object will be attached to the correct object, through its identifier, and to no other object. It seems simple, and it is, but many Big Data resources assign identifiers promiscuously, with the end result that information related to a unique object is scattered throughout the resource, attached to other objects, and cannot be sensibly retrieved when needed. The concept of object identification is of such overriding importance that a Big Data resource can be usefully envisioned as a collection of unique identifiers to which complex data is attached.

Immutability is the principle that data collected in a Big Data resource is permanent, and can never be modified. At first thought, it would seem that immutability is a ridiculous and impossible constraint. In the real world, mistakes are made, information changes, and the methods for describing information changes. This is all true, but the astute Big Data manager knows how to accrue information into data objects without changing the pre-existing data. Methods for achieving this seemingly impossible trick will be described in detail.

Introspection is a term borrowed from object-oriented programming, not often found in the Big Data literature. It refers to the ability of data objects to describe themselves when interrogated. With introspection, users of a Big Data resource can quickly determine the content of data objects and the hierarchical organization of data objects within the Big Data resource. Introspection allows users to see the types of data relationships that can be analyzed within the resource and clarifies how disparate resources can interact with one another.

Another subject covered in this book, and often omitted from the literature on Big Data, is data indexing. Though there are many books written on the art of the science of so-called back-of-the-book indexes, scant attention has been paid to the process of preparing indexes for large and complex data resources. Consequently, most Big Data resources have nothing that could be called a serious index. They might have a Web page with a few links to explanatory documents, or they might have a short and crude "help" index, but it would be rare to find a Big Data resource with a comprehensive index containing a thoughtful and updated list of terms and links. Without a proper index, most Big Data resources have limited utility for any but a few cognoscenti. It seems odd to me that organizations willing to spend hundreds of millions of dollars on a Big Data resource will balk at investing a few thousand dollars more for a proper index.

Aside from these four topics, which readers would be hard-pressed to find in the existing Big Data literature, this book covers the usual topics relevant to Big Data design, construction, operation, and analysis. Some of these topics include data quality, providing structure to unstructured data, data deidentification, data standards and interoperability issues, legacy data, data reduction and transformation, data analysis, and software issues. For these topics, discussions focus on the underlying principles; programming code and mathematical equations are conspicuously inconspicuous. An extensive Glossary covers the technical or specialized terms and topics that appear throughout the text. As each Glossary term is "optional" reading, I took the liberty of expanding on technical or mathematical concepts that appeared in abbreviated form in the main text. The Glossary provides an explanation of the practical relevance of each term to Big Data, and some readers may enjoy browsing the Glossary as a stand-alone text.

The final four chapters are non-technical; all dealing in one way or another with the consequences of our exploitation of Big Data resources. These chapters will cover legal, social, and ethical issues. The book ends with my personal predictions for the future of Big Data, and its impending impact on our futures. When preparing this book, I debated whether these four chapters might best appear in the front of the book, to whet the reader's

appetite for the more technical chapters. I eventually decided that some readers would be unfamiliar with some of the technical language and concepts included in the final chapters, necessitating their placement near the end.

Readers may notice that many of the case examples described in this book come from the field of medical informatics. The healthcare informatics field is particularly ripe for discussion because every reader is affected, on economic and personal levels, by the Big Data policies and actions emanating from the field of medicine. Aside from that, there is a rich literature on Big Data projects related to healthcare. As much of this literature is controversial, I thought it important to select examples that I could document from reliable sources. Consequently, the reference section is large, with over 200 articles from journals, newspaper articles, and books. Most of these cited articles are available for free Web download.

Who should read this book? This book is written for professionals who manage Big Data resources and for students in the fields of computer science and informatics. Data management professionals would include the leadership within corporations and funding agencies who must commit resources to the project, the project directors who must determine a feasible set of goals and who must assemble a team of individuals who, in aggregate, hold the requisite skills for the task: network managers, data domain specialists, metadata specialists, software programmers, standards experts, interoperability experts, statisticians, data analysts, and representatives from the intended user community. Students of informatics, the computer sciences, and statistics will discover that the special challenges attached to Big Data, seldom discussed in university classes, are often surprising; sometimes shocking.

By mastering the fundamentals of Big Data design, maintenance, growth, and validation, readers will learn how to simplify the endless tasks engendered by Big Data resources. Adept analysts can find relationships among data objects held in disparate Big Data resources if the data is prepared properly. Readers will discover how integrating Big Data resources can deliver benefits far beyond anything attained from stand-alone databases.

References

[1] Martin Hilbert M, Lopez P. The world's technological capacity to store, communicate, and compute information. Science 2011;332:60–5.

[2] Schmidt S. Data is exploding: the 3V's of Big Data. Business Computing World; 2012. May 15.

1

Introduction

OUTLINE

Section 1.1. Definition of Big Data

It's the data, stupid.

Jim Gray

Back in the mid 1960s, my high school held pep rallies before big games. At one of these rallies, the head coach of the football team walked to the center of the stage carrying a large box of printed computer paper; each large sheet was folded flip-flop style against the next sheet and they were all held together by perforations. The coach announced that the athletic abilities of every member of our team had been entered into the school's computer (we were lucky enough to have our own IBM-360 mainframe). Likewise, data on our rival team had also been entered. The computer was instructed to digest all of this information and to produce the name of the team that would win the annual Thanksgiving Day showdown. The computer spewed forth the aforementioned box of computer paper; the very last output sheet revealed that we were the pre-ordained winners. The next day, we sallied forth to yet another ignominious defeat at the hands of our long-time rivals.

Fast-forward about 50 years to a conference room at the National Institutes of Health (NIH), in Bethesda, Maryland. A top-level science administrator is briefing me. She explains that disease research has grown in scale over the past decade. The very best research initiatives are now multi-institutional and data-intensive. Funded investigators are using high-throughput molecular methods that produce mountains of data for every tissue sample in a matter of minutes. There is only one solution; we must acquire supercomputers and a staff of talented programmers who can analyze all our data and tell us what it all means!

The NIH leadership believed, much as my high school coach believed, that if you have a really big computer and you feed it a huge amount of information, then you can answer almost any question.

Principles and Practice of Big Data. https://doi.org/10.1016/B978-0-12-815609-4.00001-7

That day, in the conference room at the NIH, circa 2003, I voiced my concerns, indicating that you cannot just throw data into a computer and expect answers to pop out. I pointed out that, historically, science has been a reductive process, moving from complex, descriptive data sets to simplified generalizations. The idea of developing an expensive supercomputer facility to work with increasing quantities of biological data, at higher and higher levels of complexity, seemed impractical and unnecessary. On that day, my concerns were not well received. High performance supercomputing was a very popular topic, and still is. [Glossary Science, Supercomputer]

Fifteen years have passed since the day that supercomputer-based cancer diagnosis was envisioned. The diagnostic supercomputer facility was never built. The primary diagnostic tool used in hospital laboratories is still the microscope, a tool invented circa 1590. Today, we augment microscopic findings with genetic tests for specific, key mutations; but we do not try to understand all of the complexities of human genetic variations. We know that it is hopeless to try. You can find a lot of computers in hospitals and medical offices, but the computers do not calculate your diagnosis. Computers in the medical workplace are relegated to the prosaic tasks of collecting, storing, retrieving, and delivering medical records. When those tasks are finished, the computer sends you the bill for services rendered.

Before we can take advantage of large and complex data sources, we need to think deeply about the meaning and destiny of Big Data.

Big Data is defined by the three V's:

1. Volume—large amounts of data;.
2. Variety—the data comes in different forms, including traditional databases, images, documents, and complex records;.
3. Velocity—the content of the data is constantly changing through the absorption of complementary data collections, the introduction of previously archived data or legacy collections, and from streamed data arriving from multiple sources.

It is important to distinguish Big Data from "lotsa data" or "massive data." In a Big Data Resource, all three V's must apply. It is the size, complexity, and restlessness of Big Data resources that account for the methods by which these resources are designed, operated, and analyzed. [Glossary Big Data resource, Data resource]

The term "lotsa data" is often applied to enormous collections of simple-format records. For example: every observed star, its magnitude and its location; the name and cell phone number of every person living in the United States; and the contents of the Web. These very large data sets are sometimes just glorified lists. Some "lotsa data" collections are spreadsheets (2-dimensional tables of columns and rows), so large that we may never see where they end.

Big Data resources are not equivalent to large spreadsheets, and a Big Data resource is never analyzed in its totality. Big Data analysis is a multi-step process whereby data is extracted, filtered, and transformed, with analysis often proceeding in a piecemeal, sometimes recursive, fashion. As you read this book, you will find that the gulf between "lotsa data" and Big Data is profound; the two subjects can seldom be discussed productively within the same venue.

Section 1.2. Big Data Versus Small Data

> *Actually, the main function of Big Science is to generate massive amounts of reliable and easily accessible data.... Insight, understanding, and scientific progress are generally achieved by 'small science.'*
>
> *Dan Graur, Yichen Zheng, Nicholas Price, Ricardo Azevedo, Rebecca Zufall, and Eran Elhaik [1].*

Big Data is not small data that has become bloated to the point that it can no longer fit on a spreadsheet, nor is it a database that happens to be very large. Nonetheless, some professionals who customarily work with relatively small data sets, harbor the false impression that they can apply their spreadsheet and database know-how directly to Big Data resources without attaining new skills or adjusting to new analytic paradigms. As they see things, when the data gets bigger, only the computer must adjust (by getting faster, acquiring more volatile memory, and increasing its storage capabilities); Big Data poses no special problems that a supercomputer could not solve. [Glossary Database]

This attitude, which seems to be prevalent among database managers, programmers, and statisticians, is highly counterproductive. It will lead to slow and ineffective software, huge investment losses, bad analyses, and the production of useless and irreversibly defective Big Data resources.

Let us look at a few of the general differences that can help distinguish Big Data and small data.

– **Goals**

small data—Usually designed to answer a specific question or serve a particular goal.

Big Data—Usually designed with a goal in mind, but the goal is flexible and the questions posed are protean. Here is a short, imaginary funding announcement for Big Data grants designed "to combine high quality data from fisheries, coast guard, commercial shipping, and coastal management agencies for a growing data collection that can be used to support a variety of governmental and commercial management studies in the Lower Peninsula." In this fictitious case, there is a vague goal, but it is obvious that there really is no way to completely specify what the Big Data resource will contain, how the various types of data held in the resource will be organized, connected to other data resources, or usefully analyzed. Nobody can specify, with any degree of confidence, the ultimate destiny of any Big Data project; it usually comes as a surprise.

– **Location**

small data—Typically, contained within one institution, often on one computer, sometimes in one file.

Big Data—Spread throughout electronic space and typically parceled onto multiple Internet servers, located anywhere on earth.

– **Data structure and content**

small data—Ordinarily contains highly structured data. The data domain is restricted to a single discipline or sub-discipline. The data often comes in the form of uniform records in an ordered spreadsheet.

Big Data—Must be capable of absorbing unstructured data (e.g., such as free-text documents, images, motion pictures, sound recordings, physical objects). The subject matter of the resource may cross multiple disciplines, and the individual data objects in the resource may link to data contained in other, seemingly unrelated, Big Data resources. [Glossary Data object]

– **Data preparation**

small data—In many cases, the data user prepares her own data, for her own purposes.

Big Data—The data comes from many diverse sources, and it is prepared by many people. The people who use the data are seldom the people who have prepared the data.

– **Longevity**

small data—When the data project ends, the data is kept for a limited time (seldom longer than 7 years, the traditional academic life-span for research data); and then discarded.

Big Data—Big Data projects typically contain data that must be stored in perpetuity. Ideally, the data stored in a Big Data resource will be absorbed into other data resources. Many Big Data projects extend into the future and the past (e.g., legacy data), accruing data prospectively and retrospectively. [Glossary Legacy data]

– **Measurements**

small data—Typically, the data is measured using one experimental protocol, and the data can be represented using one set of standard units. [Glossary Protocol]

Big Data—Many different types of data are delivered in many different electronic formats. Measurements, when present, may be obtained by many different protocols. Verifying the quality of Big Data is one of the most difficult tasks for data managers. [Glossary Data Quality Act]

– **Reproducibility**

small data—Projects are typically reproducible. If there is some question about the quality of the data, the reproducibility of the data, or the validity of the conclusions drawn from the data, the entire project can be repeated, yielding a new data set. [Glossary Conclusions]

Big Data—Replication of a Big Data project is seldom feasible. In general, the most that anyone can hope for is that bad data in a Big Data resource will be found and flagged as such.

– **Stakes**

small data—Project costs are limited. Laboratories and institutions can usually recover from the occasional small data failure.

Big Data—Big Data projects can be obscenely expensive [2,3]. A failed Big Data effort can lead to bankruptcy, institutional collapse, mass firings, and the sudden disintegration

of all the data held in the resource. As an example, a United States National Institutes of Health Big Data project known as the "NCI cancer biomedical informatics grid" cost at least $350 million for fiscal years 2004–10. An ad hoc committee reviewing the resource found that despite the intense efforts of hundreds of cancer researchers and information specialists, it had accomplished so little and at so great an expense that a project moratorium was called [4]. Soon thereafter, the resource was terminated [5]. Though the costs of failure can be high, in terms of money, time, and labor, Big Data failures may have some redeeming value. Each failed effort lives on as intellectual remnants consumed by the next Big Data effort. [Glossary Grid]

– **Introspection**

small data—Individual data points are identified by their row and column location within a spreadsheet or database table. If you know the row and column headers, you can find and specify all of the data points contained within. [Glossary Data point]

Big Data—Unless the Big Data resource is exceptionally well designed, the contents and organization of the resource can be inscrutable, even to the data managers. Complete access to data, information about the data values, and information about the organization of the data is achieved through a technique herein referred to as introspection. Introspection will be discussed at length in Chapter 6. [Glossary Data manager, Introspection]

– **Analysis**

small data—In most instances, all of the data contained in the data project can be analyzed together, and all at once.

Big Data—With few exceptions, such as those conducted on supercomputers or in parallel on multiple computers, Big Data is ordinarily analyzed in incremental steps. The data are extracted, reviewed, reduced, normalized, transformed, visualized, interpreted, and re-analyzed using a collection of specialized methods. [Glossary Parallel computing, MapReduce]

Section 1.3. Whence Comest Big Data?

All I ever wanted to do was to paint sunlight on the side of a house.

Edward Hopper

Often, the impetus for Big Data is entirely ad hoc. Companies and agencies are forced to store and retrieve huge amounts of collected data (whether they want to or not). Generally, Big Data come into existence through any of several different mechanisms:

– An entity has collected a lot of data in the course of its normal activities and seeks to organize the data so that materials can be retrieved, as needed.

The Big Data effort is intended to streamline the regular activities of the entity. In this case, the data is just waiting to be used. The entity is not looking to discover anything or to do anything new. It simply wants to use the data to accomplish what it has always been doing;

only better. The typical medical center is a good example of an "accidental" Big Data resource. The day-to-day activities of caring for patients and recording data into hospital information systems results in terabytes of collected data, in forms such as laboratory reports, pharmacy orders, clinical encounters, and billing data. Most of this information is generated for a one-time specific use (e.g., supporting a clinical decision, collecting payment for a procedure). It occurs to the administrative staff that the collected data can be used, in its totality, to achieve mandated goals: improving quality of service, increasing staff efficiency, and reducing operational costs. [Glossary Binary units for Big Data, Binary atom count of universe]

– An entity has collected a lot of data in the course of its normal activities and decides that there are many new activities that could be supported by their data.

Consider modern corporations; these entities do not restrict themselves to one manufacturing process or one target audience. They are constantly looking for new opportunities. Their collected data may enable them to develop new products based on the preferences of their loyal customers, to reach new markets, or to market and distribute items via the Web. These entities will become hybrid Big Data/manufacturing enterprises.

– An entity plans a business model based on a Big Data resource.

Unlike the previous examples, this entity starts with Big Data and adds a physical component secondarily. Amazon and FedEx may fall into this category, as they began with a plan for providing a data-intense service (e.g., the Amazon Web catalog and the FedEx package tracking system). The traditional tasks of warehousing, inventory, pick-up, and delivery, had been available all along, but lacked the novelty and efficiency afforded by Big Data.

– An entity is part of a group of entities that have large data resources, all of whom understand that it would be to their mutual advantage to federate their data resources [6].

An example of a federated Big Data resource would be hospital databases that share electronic medical health records [7].

– An entity with skills and vision develops a project wherein large amounts of data are collected and organized, to the benefit of themselves and their user-clients.

An example would be a massive online library service, such as the U.S. National Library of Medicine's PubMed catalog, or the Google Books collection.

– An entity has no data and has no particular expertise in Big Data technologies, but it has money and vision.

The entity seeks to fund and coordinate a group of data creators and data holders, who will build a Big Data resource that can be used by others. Government agencies have been the major benefactors. These Big Data projects are justified if they lead to important discoveries that could not be attained at a lesser cost with smaller data resources.

Section 1.4. The Most Common Purpose of Big Data Is to Produce Small Data

> *If I had known what it would be like to have it all, I might have been willing to settle for less.*
>
> *Lily Tomlin*

Imagine using a restaurant locater on your smartphone. With a few taps, it lists the Italian restaurants located within a 10-block radius of your current location. The database being queried is big and complex (a map database, a collection of all the restaurants in the world, their longitudes and latitudes, their street addresses, and a set of ratings provided by patrons, updated continuously), but the data that it yields is small (e.g., five restaurants, marked on a street map, with pop-ups indicating their exact address, telephone number, and ratings). Your task comes down to selecting one restaurant from among the five, and dining thereat.

In this example, your data selection was drawn from a large data set, but your ultimate analysis was confined to a small data set (i.e., five restaurants meeting your search criteria). The purpose of the Big Data resource was to proffer the small data set. No analytic work was performed on the Big Data resource; just search and retrieval. The real labor of the Big Data resource involved collecting and organizing complex data, so that the resource would be ready for your query. Along the way, the data creators had many decisions to make (e.g., Should bars be counted as restaurants? What about take-away only shops? What data should be collected? How should missing data be handled? How will data be kept current? [Glossary Query, Missing data]

Big Data is seldom, if ever, analyzed *in toto*. There is almost always a drastic filtering process that reduces Big Data into smaller data. This rule applies to scientific analyses. The Australian Square Kilometre Array of radio telescopes [8], WorldWide Telescope, CERN's Large Hadron Collider and the Pan-STARRS (Panoramic Survey Telescope and Rapid Response System) array of telescopes produce petabytes of data every day. Researchers use these raw data sources to produce much smaller data sets for analysis [9]. [Glossary Raw data, Square Kilometer Array, Large Hadron Collider, WorldWide Telescope]

Here is an example showing how workable subsets of data are prepared from Big Data resources. Blazars are rare super-massive black holes that release jets of energy that move at near-light speeds. Cosmologists want to know as much as they can about these strange objects. A first step to studying blazars is to locate as many of these objects as possible. Afterwards, various measurements on all of the collected blazars can be compared, and their general characteristics can be determined. Blazars seem to have a gamma ray signature that is not present in other celestial objects. The WISE survey collected infrared data on the entire observable universe. Researchers extracted from the Wise data every celestial body associated with an infrared signature in the gamma ray range that was suggestive of blazars; about 300 objects. Further research on these 300 objects led the researchers to

believe that about half were blazars [10]. This is how Big Data research often works; by constructing small data sets that can be productively analyzed.

Because a common role of Big Data is to produce small data, a question that data managers must ask themselves is: "Have I prepared my Big Data resource in a manner that helps it become a useful source of small data?"

Section 1.5. Big Data Sits at the Center of the Research Universe

Physics is the universe's operating system.

Steven R Garman

In the past, scientists followed a well-trodden path toward truth: hypothesis, then experiment, then data, then analysis, then publication. The manner in which a scientist analyzed his or her data was crucial because other scientists would not have access to the same data and could not re-analyze the data for themselves. Basically, the results and conclusions described in the manuscript was the scientific product. The primary data upon which the results and conclusion were based (other than one or two summarizing tables) were not made available for review. Scientific knowledge was built on trust. Customarily, the data would be held for 7 years, and then discarded. [Glossary Results]

In the Big data paradigm the concept of a final manuscript has little meaning. Big Data resources are permanent, and the data within the resource is immutable (See Chapter 6). Any scientist's analysis of the data does not need to be the final word; another scientist can access and re-analyze the same data over and over again. Original conclusions can be validated or discredited. New conclusions can be developed. The centerpiece of science has moved from the manuscript, whose conclusions are tentative until validated, to the Big Data resource, whose data will be tapped repeatedly to validate old manuscripts and spawn new manuscripts. [Glossary Immutability, Mutability]

Today, hundreds or thousands of individuals might contribute to a Big Data resource. The data in the resource might inspire dozens of major scientific projects, hundreds of manuscripts, thousands of analytic efforts, and millions or billions of search and retrieval operations. The Big Data resource has become the central, massive object around which universities, research laboratories, corporations, and federal agencies orbit. These orbiting objects draw information from the Big Data resource, and they use the information to support analytic studies and to publish manuscripts. Because Big Data resources are permanent, any analysis can be critically examined using the same set of data, or re-analyzed anytime in the future. Because Big Data resources are constantly growing forward in time (i.e., accruing new information) and backward in time (i.e., absorbing legacy data sets), the value of the data is constantly increasing.

Big Data resources are the stars of the modern information universe. All matter in the physical universe comes from heavy elements created inside stars, from lighter elements. All data in the informational universe is complex data built from simple data. Just as stars

can exhaust themselves, explode, or even collapse under their own weight to become black holes; Big Data resources can lose funding and die, release their contents and burst into nothingness, or collapse under their own weight, sucking everything around them into a dark void. It is an interesting metaphor. In the following chapters, we will see how a Big Data resource can be designed and operated to ensure stability, utility, growth, and permanence; features you might expect to find in a massive object located in the center of the information universe.

Glossary

Big Data resource A Big Data collection that is accessible for analysis. Readers should understand that there are collections of Big Data (i.e., data sources that are large, complex, and actively growing) that are not designed to support analysis; hence, not Big Data resources. Such Big Data collections might include some of the older hospital information systems, which were designed to deliver individual patient records upon request; but could not support projects wherein all of the data contained in all of the records were opened for selection and analysis. Aside from privacy and security issues, opening a hospital information system to these kinds of analyses would place enormous computational stress on the systems (i.e., produce system crashes). In the late 1990s and the early 2000s data warehousing was popular. Large organizations would collect all of the digital information created within their institutions, and these data were stored as Big Data collections, called data warehouses. If an authorized person within the institution needed some specific set of information (e.g., emails sent or received in February, 2003; all of the bills paid in November, 1999), it could be found somewhere within the warehouse. For the most part, these data warehouses were not true Big Data resources because they were not organized to support a full analysis of all of the contained data. Another type of Big Data collection that may or may not be considered a Big Data resource are compilations of scientific data that are accessible for analysis by private concerns, but closed for analysis by the public. In this case a scientist may make a discovery based on her analysis of a private Big Data collection, but the research data is not open for critical review. In the opinion of some scientists, including myself, if the results of a data analysis are not available for review, then the analysis is illegitimate. Of course, this opinion is not universally shared, and Big Data professionals hold various definitions for a Big Data resource.

Binary atom count of universe There are estimated to be about 10^80 atoms in the universe. Log2(10) is 3.32192809, so the number of atoms in the universe is 2^80*3.32192809 or 2^266 atoms.

Binary units for Big Data Binary sizes are named in 1000-fold intervals: 1 bit = binary digit (0 or 1); 1 byte = 8 bits (the number of bits required to express an ascii character); 1000 bytes = 1 kilobyte; 1000 kilobytes = 1 megabyte; 1000 megabytes = 1 gigabyte; 1000 gigabytes = 1 terabyte; 1000 terabytes = 1 petabyte; 1000 petabytes = 1 exabyte; 1000 exabytes = 1 zettabyte; 1000 zettabytes = 1 yottabyte.

Conclusions Conclusions are the interpretations made by studying the results of an experiment or a set of observations. The term "results" should never be used interchangeably with the term "conclusions." **Remember, results are verified. Conclusions are validated** [11].

Data Quality Act In the United States the data upon which public policy is based must have quality and must be available for review by the public. Simply put, public policy must be based on verifiable data. The Data Quality Act of 2002 requires the Office of Management and Budget to develop government-wide standards for data quality [12].

Data manager This book uses "data manager" as a catchall term, without attaching any specific meaning to the name. Depending on the institutional and cultural milieu, synonyms and plesionyms (i.e., near-synonyms) for data manager would include: technical lead, team liaison, data quality manager, chief curator, chief of operations, project manager, group supervisor, and so on.

Data object As used in this book, a data object consists of a unique object identifier along with all of the data/metadata pairs that rightly belong to the object identifier, and that includes one data/metadata pair that tells us the object's class.

```
75898039563441
  name                G. Willikers
  gender              male
  age                 35
  is_a_class_member   cowboy
```

In this example, the object identifier, 75898039563441, is followed by its data/metadata pairs, including the one pair that tells us that the object (a 35-year-old man named G. Willikers) belongs to the class of individuals known as "cowboy."

The utility of data objects, in the field of Big Data, is discussed in Section 6.2.

Data point The singular form of data is datum. Strictly speaking, the term should be datum point or datumpoint. Most information scientists, myself included, have abandoned consistent usage rules for the word "data." In this book, the term "data" always refers collectively to information, numeric or textual, structured or unstructured, in any quantity.

Data resource A collection of data made available for data retrieval. The data can be distributed over servers located anywhere on earth or in space. The resource can be static (i.e., having a fixed set of data), or in flux. Plesionyms for data resource are: data warehouse, data repository, data archive, and data store.

Database A software application designed specifically to create and retrieve large numbers of data records (e.g., millions or billions). The data records of a database are persistent, meaning that the application can be turned off, then on, and all the collected data will be available to the user.

Grid A collection of computers and computer resources (typically networked servers) that is coordinated to provide a desired functionality. In the most advanced Grid computing architecture, requests can be broken into computational tasks that are processed in parallel on multiple computers and transparently (from the client's perspective) assembled and returned. The Grid is the intellectual predecessor of Cloud computing. Cloud computing is less physically and administratively restricted than Grid computing.

Immutability Immutability is the principle that data collected in a Big Data resource is permanent and can never be modified. At first thought, it would seem that immutability is a ridiculous and impossible constraint. In the real world, mistakes are made, information changes, and the methods for describing information changes. This is all true, but the astute Big Data manager knows how to accrue information into data objects without changing the pre-existing data. Methods for achieving this seemingly impossible trick are described in Chapter 8.

Introspection Well-designed Big Data resources support introspection, a method whereby data objects within the resource can be interrogated to yield their properties, values, and class membership. Through introspection the relationships among the data objects in the Big Data resource can be examined and the structure of the resource can be determined. Introspection is the method by which a data user can find everything there is to know about a Big Data resource without downloading the complete resource.

Large Hadron Collider The Large Hadron Collider is the world's largest and most powerful particle accelerator and is expected to produce about 15 petabytes (15 million gigabytes) of data annually [13].

Legacy data Data collected by an information system that has been replaced by a newer system, and which cannot be immediately integrated into the newer system's database. For example, hospitals regularly replace their hospital information systems with new systems that promise greater efficiencies, expanded services, or improved interoperability with other information systems. In many cases, the new system cannot readily integrate the data collected from the older system. The previously collected

data becomes a legacy to the new system. In such cases, legacy data is simply "stored" for some arbitrary period of time in case someone actually needs to retrieve any of the legacy data. After a decade or so the hospital may find itself without any staff members who are capable of locating the storage site of the legacy data, or moving the data into a modern operating system, or interpreting the stored data, or retrieving appropriate data records, or producing a usable query output.

MapReduce A method by which computationally intensive problems can be processed on multiple computers, in parallel. The method can be divided into a mapping step and a reducing step. In the mapping step a master computer divides a problem into smaller problems that are distributed to other computers. In the reducing step the master computer collects the output from the other computers. Although MapReduce is intended for Big Data resources, and can hold petabytes of data, most Big Data problems do not require MapReduce.

Missing data Most complex data sets have missing data values. Somewhere along the line data elements were not entered, records were lost, or some systemic error produced empty data fields. Big Data, being large, complex, and composed of data objects collected from diverse sources, is almost certain to have missing data. Various mathematical approaches to missing data have been developed; commonly involving assigning values on a statistical basis; so-called imputation methods. The underlying assumption for such methods is that missing data arises at random. When missing data arises nonrandomly, there is no satisfactory statistical fix. The Big Data curator must track down the source of the errors and somehow rectify the situation. In either case the issue of missing data introduces a potential bias and it is crucial to fully document the method by which missing data is handled. In the realm of clinical trials, only a minority of data analyses bothers to describe their chosen method for handling missing data [14].

Mutability Mutability refers to the ability to alter the data held in a data object or to change the identity of a data object. Serious Big Data is not mutable. Data can be added, but data cannot be erased or altered. Big Data resources that are mutable cannot establish a sensible data identification system, and cannot support verification and validation activities. The legitimate ways in which we can record the changes that occur in unique data objects (e.g., humans) over time, without ever changing the key/value data attached to the unique object, is discussed in Section 8.2.

For programmers, it is important to distinguish data mutability from object mutability, as it applies in Python and other object-oriented programming languages. Python has two immutable objects: strings and tuples. Intuitively, we would probably guess that the contents of a string object cannot be changed, and the contents of a tuple object cannot be changed. This is not the case. Immutability, for programmers, means that there are no methods available to the object by which the contents of the object can be altered. Specifically, a Python tuple object would have no methods it could call to change its own contents. However, a tuple may contain a list, and lists are mutable. For example, a list may have an append method that will add an item to the list object. You can change the contents of a list contained in a tuple object without violating the tuple's immutability.

Parallel computing Some computational tasks can be broken down and distributed to other computers, to be calculated "in parallel." The method of parallel programming allows a collection of desktop computers to complete intensive calculations of the sort that would ordinarily require the aid of a supercomputer. Parallel programming has been studied as a practical way to deal with the higher computational demands brought by Big Data. Although there are many important problems that require parallel computing, the vast majority of Big Data analyses can be easily accomplished with a single, off-the-shelf personal computer.

Protocol A set of instructions, policies, or fully described procedures for accomplishing a service, operation, or task. Protocols are fundamental to Big Data. Data is generated and collected according to protocols. There are protocols for conducting experiments, and there are protocols for measuring the results. There are protocols for choosing the human subjects included in a clinical trial, and there are protocols for interacting with the human subjects during the course of the trial. All network

communications are conducted via protocols; the Internet operates under a protocol (TCP-IP, Transmission Control Protocol-Internet Protocol).

Query The term "query" usually refers to a request, sent to a database, for information (e.g., Web pages, documents, lines of text, images) that matches a provided word or phrase (i.e., the query term). More generally a query is a parameter or set of parameters that are submitted as input to a computer program that searches a data collection for items that match or bear some relationship to the query parameters. In the context of Big Data the user may need to find classes of objects that have properties relevant to a particular area of interest. In this case, the query is basically introspective, and the output may yield metadata describing individual objects, classes of objects, or the relationships among objects that share particular properties. For example, "weight" may be a property, and this property may fall into the domain of several different classes of data objects. The user might want to know the names of the classes of objects that have the "weight" property and the numbers of object instances in each class. Eventually the user might want to select several of these classes (e.g., including dogs and cats, but excluding microwave ovens) along with the data object instances whose weights fall within a specified range (e.g., 20–30 pound). This approach to querying could work with any data set that has been well specified with metadata, but it is particularly important when using Big Data resources.

Raw data Raw data is the unprocessed, original data measurement, coming straight from the instrument to the database with no intervening interference or modification. In reality, scientists seldom, if ever, work with raw data. When an instrument registers the amount of fluorescence emitted by a hybridization spot on a gene array, or the concentration of sodium in the blood, or virtually any of the measurements that we receive as numeric quantities, the output is produced by an algorithm executed by the measurement instrument. Pre-processing of data is commonplace in the universe of Big Data, and data managers should not labor under the false impression that the data received is "raw," simply because the data has not been modified by the person who submits the data.

Results The term "results" is often confused with the term "conclusions." Interchanging the two concepts is a source of confusion among data scientists. In the strictest sense, "results" consist of the full set of experimental data collected by measurements. In practice, "results" are provided as a small subset of data distilled from the raw, original data. In a typical journal article, selected data subsets are packaged as a chart or graph that emphasizes some point of interest. Hence, the term "results" may refer, erroneously, to subsets of the original data, or to visual graphics intended to summarize the original data. Conclusions are the inferences drawn from the results. Results are verified; conclusions are validated.

Science Of course, there are many different definitions of science, and inquisitive students should be encouraged to find a conceptualization of science that suits their own intellectual development. For me, science is all about finding general relationships among objects. In the so-called physical sciences the most important relationships are expressed as mathematical equations (e.g., the relationship between force, mass and acceleration; the relationship between voltage, current and resistance). In the so-called natural sciences, relationships are often expressed through classifications (e.g., the classification of living organisms). Scientific advancement is the discovery of new relationships or the discovery of a generalization that applies to objects hitherto confined within disparate scientific realms (e.g., evolutionary theory arising from observations of organisms and geologic strata). Engineering would be the area of science wherein scientific relationships are exploited to build new technology.

Square Kilometer Array The Square Kilometer Array is designed to collect data from millions of connected radio telescopes and is expected to produce more than one exabyte (1 billion gigabytes) every day [8].

Supercomputer Computers that can perform many times faster than a desktop personal computer. In 2015 the top supercomputers operate at about 30 petaflops. A petaflop is 10 to the 15 power floating point operations per second. By my calculations a 1 petaflop computer performs about 250,000 operations in the time required for my laptop to finish one operation.

WorldWide Telescope A Big Data effort from the Microsoft Corporation bringing astronomical maps, imagery, data, analytic methods, and visualization technology to standard Web browsers. More information is available at: http://www.worldwidetelescope.org/Home.aspx

References

[1] Graur D, Zheng Y, Price N, Azevedo RB, Zufall RA, Elhaik E. On the immortality of television sets: "function" in the human genome according to the evolution-free gospel of ENCODE. Genome Biol Evol 2013;5:578–90.

[2] Whittaker Z. UK's delayed national health IT programme officially scrapped. ZDNet, September 22, 2011.

[3] Kappelman LA, McKeeman R, Lixuan Zhang L. Early warning signs of IT project failure: the dominant dozen. Information Systems Management 2006;23:31–6.

[4] An assessment of the impact of the NCI cancer Biomedical Informatics Grid (caBIG). Report of the Board of Scientific Advisors Ad Hoc Working Group. National Cancer Institute; March 2011.

[5] Komatsoulis GA. Program announcement to the CaBIG community. National Cancer Institute. https://cabig.nci.nih.gov/program_announcement [viewed August 31, 2012].

[6] Freitas A, Curry E, Oliveira JG, O'Riain S. Querying heterogeneous datasets on the linked data web: challenges, approaches, and trends. IEEE Internet Computing 2012;16:24–33.

[7] Drake TA, Braun J, Marchevsky A, Kohane IS, Fletcher C, Chueh H, et al. A system for sharing routine surgical pathology specimens across institutions: the Shared Pathology Informatics Network (SPIN). Hum Pathol 2007;38:1212–25.

[8] Francis M. Future telescope array drives development of exabyte processing. Ars Technica; 2012. April 2.

[9] Markoff J. A deluge of data shapes a new era in computing. The New York Times; 2009. December 15.

[10] Harrington JD, Clavin W. NASA's WISE mission sees skies ablaze with blazars. NASA Release 12-109; 2002. April 12.

[11] Committee on Mathematical Foundations of Verification, Validation, and Uncertainty Quantification; Board on Mathematical Sciences and Their Applications, Division on Engineering and Physical Sciences, National Research Council. Assessing the reliability of complex models: mathematical and statistical foundations of verification, validation, and uncertainty quantification. National Academy Press; 2012. Available from: http://www.nap.edu/catalog.php?record_id=13395. [viewed January 1, 2015].

[12] Data Quality Act. 67 Fed. Reg. 8,452, February 22, 2002, addition to FY 2001 Consolidated Appropriations Act (Pub. L. No. 106-554. Codified at 44 U.S.C. 3516).

[13] Worldwide LHC Computing Grid. European Organization for Nuclear Research. Available from: http://public.web.cern.ch/public/en/lhc/Computing-en.html; 2008 [viewed September 19, 2012].

[14] Carpenter JR, Kenward MG. Missing data in randomised control trials: a practical guide. November 21. Available from:http://www.hta.nhs.uk/nihrmethodology/reports/1589.pdf; 2007.

This page intentionally left blank

2

Providing Structure to Unstructured Data

OUTLINE

Section 2.1. Nearly All Data Is Unstructured and Unusable in Its Raw Form

> *I was working on the proof of one of my poems all the morning, and took out a comma. In the afternoon I put it back again.*
>
> *Oscar Wilde*

In the early days of computing, data was always highly structured. All data was divided into fields, the fields had a fixed length, and the data entered into each field was constrained to a pre-determined set of allowed values. Data was entered into punch cards with pre-configured rows and columns. Depending on the intended use of the cards, various entry and read-out methods were chosen to express binary data, numeric data, fixed-size text, or programming instructions. Key-punch operators produced mountains of punch cards. For many analytic purposes, card-encoded data sets were analyzed without the assistance of a computer; all that was needed was a punch card sorter. If you wanted the data card on all males, over the age of 18, who had graduated high school, and had passed their physical exam, then the sorter would need to make 4 passes. The sorter would pull every card listing a male, then from the male cards it would pull all the cards of people over the age of 18, and from this double-sorted sub-stack, it would pull cards that met the next criterion, and so on.

Principles and Practice of Big Data. https://doi.org/10.1016/B978-0-12-815609-4.00002-9

As a high school student in the 1960s, I loved playing with the card sorters. Back then, all data was structured data, and it seemed to me, at the time, that a punch-card sorter was all that anyone would ever need to analyze large sets of data. [Glossary Binary data]

How wrong I was! Today, most data entered by humans is unstructured in the form of free-text. The free-text comes in email messages, tweets, and documents. Structured data has not disappeared, but it sits in the shadows cast by mountains of unstructured text. Free-text may be more interesting to read than punch cards, but the venerable punch card, in its heyday, was much easier to analyze than its free-text descendant. To get much informational value from free-text, it is necessary to impose some structure. This may involve translating the text to a preferred language; parsing the text into sentences; extracting and normalizing the conceptual terms contained in the sentences; mapping terms to a standard nomenclature; annotating the terms with codes from one or more standard nomenclatures; extracting and standardizing data values from the text; assigning data values to specific classes of data belonging to a classification system; assigning the classified data to a storage and retrieval system (e.g., a database); and indexing the data in the system. All of these activities are difficult to do on a small scale and virtually impossible to do on a large scale. Nonetheless, every Big Data project that uses unstructured data must deal with these tasks to yield the best possible results with the resources available. [Glossary Parsing, Nomenclature, Nomenclature mapping, Thesaurus, Indexes, Plain-text]

Section 2.2. Concordances

> *The limits of my language are the limits of my mind. All I know is what I have words for. (Die Grenzen meiner Sprache bedeuten die Grenzen meiner Welt.)*
>
> *Ludwig Wittgenstein*

A concordance is a list of all the different words contained in a text with the locations in the text where each word appears. Concordances have been around for a very long time, painstakingly constructed from holy scriptures thought to be of such immense value that every word deserved special attention. Creating a concordance has always been a straightforward operation. You take the first word in the text and you note its location (i.e., word 1, page 1); then onto the second word (word 2 page 1), and so on. When you come to a word that has been included in the nascent concordance, you add its location to the existing entry for the word. Continuing thusly, for a few months or so, you end up with a concordance that you can be proud of. Today a concordance for the Bible can be constructed in a small fraction of a second. [Glossary Concordance]

Without the benefit of any special analyses, skimming through a book's concordance provides a fairly good idea of the following:

– The topic of the text based on the words appearing in the concordance. For example, a concordance listing multiple locations for "begat" and "anointed" and "thy" is most likely to be the Old Testament.

- The complexity of the language. A complex or scholarly text will have a larger vocabulary than a romance novel.
- A precise idea of the length of the text, achieved by adding all of the occurrences of each of the words in the concordance. Knowing the number of items in the concordance, multiplied by the average number of locations of concordance items, provides a rough estimate of the total number of words in the text.
- The care with which the text was prepared, achieved by counting the misspelled words.

Here, in a short Python script, concord_gettysbu.py, that builds a concordance for the Gettysburg address, located in the external file "gettysbu.txt": [Glossary Script]

```
import re, string
word_list=[];word_dict={};key_list=[]
count=0; word=""
in_text_string = open('gettysbu.txt', "r").read().lower()
word_list = re.split(r'[^a-zA-z\_\-]+',in_text_string)
for word in word_list:
   count = count + 1
   if word in word_dict:
     word_dict[word] = word_dict[word] + ',' + str(count)
   else:
     word_dict[word] = str(count)
key_list = list(word_dict)
key_list.sort()
for key in key_list:
   print(key + " " + word_dict[key])
```

The first few lines of output are shown:

```
a 14,36,59,70,76,104,243
above 131
add 136
advanced 185
ago 6
all 26
altogether 93
and 3,20,49,95,122,248
any 45
are 28,33,56
as 75
battlefield 61
be 168,192
before 200
birth 245
```

```
brave 119
brought 9
but 102,151
by 254
can 52,153
cannot 108,111,114
```

The numbers that follow each item in the concordance correspond to the locations (expressed as the nth words of the Gettysburg address) of each word in the text.

At this point, building a concordance may appear to an easy, but somewhat pointless exercise. Does the concordance provide any functionality beyond that provided by the ubiquitous "search" box. There are five very useful properties of concordances that you might not have anticipated.

- You can use a concordance to rapidly search and retrieve the locations where single-word terms appear.
- You can always reconstruct the original text from the concordance. Hence, after you've built your concordance, you can discard the original text.
- You can merge concordances without forfeiting your ability to reconstruct the original texts, provided that you tag locations with some character sequence that identifies the text of origin.
- With a little effort a dictionary can be transformed into a universal concordance (i.e., a merged dictionary/concordance of every book in existence) by attaching the book identifier and its concordance entries to the corresponding dictionary terms.
- You can easily find the co-locations among words (i.e., which words often precede or follow one another).
- You can use the concordance to retrieve the sentences and paragraphs in which a search word or a search term appears, without having access to the original text. The concordance alone can reconstruct and retrieve the appropriate segments of text, on-the-fly, thus bypassing the need to search the original text.
- A concordance provides a profile of the book and can be used to compute a similarity score among different books.

There is insufficient room to explore all of the useful properties of concordances, but let us examine a script, concord_reverse.py, that reconstructs the original text, in lowercase, from the concordance. In this case, we have pasted the output from the concord_gettysbu.py script (vida supra) into the external file, "concordance.txt".

```
import re, string
concordance_hash = {} ; location_array = []
in_text = open('concordance.txt', "r")
for line in in_text:
   line = line.replace("\n","")
   location_word, separator, location_positions = line.partition(" ")
   location_array = location_positions.split(",")
```

```
    location_array = [int(x) for x in location_array]
    for location in location_array:
        concordance_hash[location] = location_word
for n in range(300):
    if n in concordance_hash:
        print((concordance_hash[n]), end = " ")
```

Here is the familiar output:

> *four score and seven years ago our fathers brought forth on this continent a new nation conceived in liberty and dedicated to the proposition that all men are created equal now we are engaged in a great civil war testing whether that nation or any nation so conceived and so dedicated can long endure we are met on a great battlefield of that war we have come to dedicate a portion of that field as a final resting-place for those who here gave their lives that that nation might live it is altogether fitting and proper that we should do this but in a larger sense we cannot dedicate we cannot consecrate we cannot hallow this ground the brave men living and dead who struggled here have consecrated it far above our poor power to add or detract the world will little note nor long remember what we say here but it can never forget what they did here it is for us the living rather to be dedicated here to the unfinished work which they who fought here have thus far so nobly advanced it is rather for us to be here dedicated to the great task remaining before us–that from these honored dead we take increased devotion to that cause for which they gave the last full measure of devotion–that we here highly resolve that these dead shall not have died in vain that this nation under god shall have a new birth of freedom and that government of the people by the people for the people shall not perish from the earth*

Had we wanted to write a script that produces a merged concordance, for multiple documents, we could have simply written a loop that repeated the concordance-building process for each text. Within the loop, we would have tagged each word location with a short notation indicating the particular source book. For example, locations from the Gettysburg address could have been prepended with "G:" and locations from the Bible might have been prepended with a "B:".

We have not finished with the topic of concordances. Later in this chapter (Section 2.8), we will show how concordances can be transformed to speed-up search and retrieval operations on large bodies of text.

Section 2.3. Term Extraction

> *There's a big difference between knowing the name of something and knowing something.*
>
> *Richard Feynman*

One of my favorite movies is the parody version of "Hound of the Baskervilles," starring Peter Cooke as Sherlock Holmes and Dudley Moore as his faithful hagiographer, Dr. Watson. Sherlock, preoccupied with his own ridiculous pursuits, dispatches Watson to the Baskerville family manse, in Dartmoor, to undertake urgent sleuth-related activities. The hapless Watson, standing in the great Baskerville Hall, has no idea how to proceed with the investigation. After a moment of hesitation, he turns to the incurious maid and commands, "Take me to the clues!"

Building an index is a lot like solving a fiendish crime; you need to know how to find the clues. For informaticians, the terms in the text are the clues upon which the index is built. Terms in a text file do not jump into your index file; you need to find them. There are several available methods for finding and extracting index terms from a corpus of text [1], but no method is as simple, fast, and scalable as the "stop word" method [2]. [Glossary Term extraction algorithm, Scalable]

The "stop word" method presumes that text is composed of terms that are somehow connected into sequences known as sentences. [Glossary Sentence]

Consider the following:

The diagnosis is chronic viral hepatitis.

This sentence contains two very specific medical concepts: "diagnosis" and "chronic viral hepatitis." These two concepts are connected to form a sentence, using grammatical bric-a-brac such as "the" and "is", and the sentence delimiter, ".". These grammatical bric-a-brac are found liberally sprinkled in every paragraph you are likely to read.

A term can be defined as a sequence of one or more uncommon words that are demarcated (i.e., bounded on one side or another) by the occurrence of one or more very common words (e.g., "and", "the", "a", "of") and phrase delimiters (e.g., ".", ",", and ";")

Consider the following:

An epidural hemorrhage can occur after a lucid interval.

The medical concepts "epidural hemorrhage" and "lucid interval" are composed of uncommon words. These uncommon word sequences are bounded by common words (i.e., "the", "an", "can", "a") or a sentence delimiter (i.e., ".").

If we had a list of all the words that were considered common, we could write a program that extracts the all the concepts found in any text of any length. The concept terms would consist of all sequences of uncommon words that are uninterrupted by common words. Here is an algorithm for extracting terms from a sentence:

1. Read the first word of the sentence. If it is a common word, delete it. If it is an uncommon word, save it.
2. Read the next word. If it is a common word, delete it, and place the saved word (from the prior step, if the prior step saved a word) into our list of terms found in the text. If it

is an uncommon word, concatenate it with the word we saved in step one, and save the 2-word term. If it is a sentence delimiter, place any saved term into our list of terms, and stop the program.

3. Repeat step two.

This simple algorithm, or something much like it, is a fast and efficient method to build a collection of index terms. The following list of common words might be useful: "about, again, all, almost, also, although, always, among, an, and, another, any, are, as, at, be, because, been, before, being, between, both, but, by, can, could, did, do, does, done, due, during, each, either, enough, especially, etc, for, found, from, further, had, has, have, having, here, how, however, i, if, in, into, is, it, its, itself, just, kg, km, made, mainly, make, may, mg, might, ml, mm, most, mostly, must, nearly, neither, no, nor, obtained, of, often, on, our, overall, perhaps, pmid, quite, rather, really, regarding, seem, seen, several, should, show, showed, shown, shows, significantly, since, so, some, such, than, that, the, their, theirs, them, then, there, therefore, these, they, this, those, through, thus, to, upon, use, used, using, various, very, was, we, were, what, when, which, while, with, within, without, would."

Such lists of common words are sometimes referred to as "stop word" lists or "barrier word" lists, as they demarcate the beginnings and endings of extraction terms. Let us look at a short Python script (terms.py) that uses our list of stop words (contained in the file stop.txt) and extracts the terms from the sentence: "Once you have a method for extracting terms from sentences the task of creating an index associating a list of locations with each term is child's play for programmers"

```
import re, string
stopfile = open("stop.txt",'r')
stop_list = stopfile.readlines()
stopfile.close()
item_list = []
line = "Once you have a method for extracting terms from \
sentences the task of creating an index associating a list \
of locations with each term is child's play for programmers"
for stopword in stop_list:
   stopword = re.sub(r'\n', '', stopword)
   line = re.sub(r' *\b' + stopword + r'\b *', '\n', line)
item_list.extend(line.split("\n"))
item_list = sorted(set(item_list))
for item in item_list:
   print(item)
```

Here is the output:

```
Once
child's play
creating
```

```
extracting terms
index associating
list
locations
method
programmers
sentences
task
term
```

Extracting terms is the first step in building a very crude index. Indexes built directly from term extraction algorithms always contain lots of unnecessary terms having little or no informational value. For serious indexers, the collection of terms extracted from a corpus, along with their locations in the text, is just the beginning of an intellectual process that will eventually lead to a valuable index.

Section 2.4. Indexing

> *Knowledge can be public, yet undiscovered, if independently created fragments are logically related but never retrieved, brought together, and interpreted.*
>
> *Donald R. Swanson [3]*

Individuals accustomed to electronic media tend to think of the Index as an inefficient or obsolete method for finding and retrieving information. Most currently available e-books have no index. It is far easier to pull up the "Find" dialog box and enter a word or phrase. The e-reader can find all matches quickly, providing the total number of matches, and bringing the reader to any or all of the pages containing the selection. As more and more books are published electronically, the book Index, as we have come to know it, may cease to be.

It would be a pity if indexes were to be abandoned by computer scientists. A well-designed book index is a creative, literary work that captures the content and intent of the book and transforms it into a listing wherein related concepts are collected under common terms, and keyed to their locations. It saddens me that many people ignore the book index until they want something from it. Open a favorite book and read the index, from A to Z, as if you were reading the body of the text. You will find that the index refreshes your understanding of the concepts discussed in the book. The range of page numbers after each term indicates that a concept has extended its relevance across many different chapters. When you browse the different entries related to a single term, you learn how the concept represented by the term applies itself to many different topics. You begin to understand, in ways that were not apparent when you read the book as a linear text, the versatility of the ideas contained in the book. When you have finished reading the index, you will notice that the indexer exercised great restraint when selecting terms.

Most indexes are under 20 pages. The goal of the indexer is not to create a concordance (i.e., a listing of every word in a book, with its locations), but to create a keyed encapsulation of concepts, sub-concepts and term relationships.

The indexes we find in today's books are generally alphabetized terms. In prior decades and prior centuries, authors and editors put enormous effort into building indexes, sometimes producing multiple indexes for a single book. For example, a biography might contain a traditional alphabetized term index, followed by an alphabetized index of the names of the people included in the text. A zoology book might include an index specifically for animal names, with animals categorized according to their taxonomic order. A geography index might list the names of localities sub-indexed by country, with countries sub-indexed by continent. A single book might have 5 or more indexes. In nineteenth century books, it was not unusual to publish indexes as stand-alone volumes. [Glossary Taxonomy, Systematics, Taxa, Taxon]

You may be thinking that all this fuss over indexes is quaint, but it cannot apply to Big Data resources. Actually, Big Data resources that lack a proper index cannot be utilized to their full potential. Without an index, you never know what your queries are missing. Remember, in a Big Data resource, it is the relationship among data objects that are the keys to knowledge. Data by itself, even in large quantities, tells only part of a story. The most useful Big Data resources have electronic indexes that map concepts, classes, and terms to specific locations in the resource where data items are stored. An index imposes order and simplicity on the Big Data resource. Without an index, Big Data resources can easily devolve into vast collections of disorganized information. [Glossary Class]

The best indexes comply with international standards (ISO 999) and require creativity and professionalism [4]. Indexes should be accepted as another device for driving down the complexity of Big Data resources. Here are a few of the specific strengths of an index that cannot be duplicated by "find" operations on terms entered into a query box:

- An index can be read, like a book, to acquire a quick understanding of the contents and general organization of the data resource.
- Index lookups (i.e., searches and retrievals) are virtually instantaneous, even for very large indexes (see Section 2.6 of this chapter, for explanation).
- Indexes can be tied to a classification. This permits the analyst to know the relationships among different topics within the index, and within the text. [Glossary Classification]
- Many indexes are cross-indexed, providing relationships among index terms that might be extremely helpful to the data analyst.
- Indexes from multiple Big Data resources can be merged. When the location entries for index terms are annotated with the name of the resource, then merging indexes is trivial, and index searches will yield unambiguously identified locators in any of the Big Data resources included in the merge.
- Indexes can be created to satisfy a particular goal; and the process of creating a made-to-order index can be repeated again and again. For example, if you have a Big Data

resource devoted to ornithology, and you have an interest in the geographic location of species, you might want to create an index specifically keyed to localities, or you might want to add a locality sub-entry for every indexed bird name in your original index. Such indexes can be constructed as add-ons, as needed. [Glossary Ngrams]

- Indexes can be updated. If terminology or classifications change, there is nothing stopping you from re-building the index with an updated specification. In the specific context of Big Data, you can update the index without modifying your data. [Glossary Specification]
- Indexes are created after the database has been created. In some cases, the data manager does not envision the full potential of the Big Data resource until after it is created. The index can be designed to facilitate the use of the resource in line with the observed practices of users.
- Indexes can serve as surrogates for the Big Data resource. In some cases, all the data user really needs is the index. A telephone book is an example of an index that serves its purpose without being attached to a related data source (e.g., caller logs, switching diagrams).

Section 2.5. Autocoding

The beginning of wisdom is to call things by their right names.

Chinese proverb

Coding, as used in the context of unstructured textual data, is the process of tagging terms with an identifier code that corresponds to a synonymous term listed in a standard nomenclature. For example, a medical nomenclature might contain the term renal cell carcinoma, a type of kidney cancer, attaching a unique identifier code for the term, such as "C9385000." There are about 50 recognized synonyms for "renal cell carcinoma." A few of these synonyms and near-synonyms are listed here to show that a single concept can be expressed many different ways, including: adenocarcinoma arising from kidney, adenocarcinoma involving kidney, cancer arising from kidney, carcinoma of kidney, Grawitz tumor, Grawitz tumour, hypernephroid tumor, hypernephroma, kidney adenocarcinoma, renal adenocarcinoma, and renal cell carcinoma. All of these terms could be assigned the same identifier code, "C9385000". [Glossary Coding, Identifier]

The process of coding a text document involves finding all the terms that belong to a specific nomenclature, and tagging each term with the corresponding identifier code.

A nomenclature is a specialized vocabulary, usually containing terms that comprehensively cover a knowledge domain. For example, there may be a nomenclature of diseases, of celestial bodies, or of makes and models of automobiles. Some nomenclatures are ordered alphabetically. Others are ordered by synonymy, wherein all synonyms and plesionyms (near-synonyms) are collected under a canonical (i.e., best or preferred) term. Synonym indexes are always corrupted by the inclusion of polysemous terms (i.e., terms with multiple meanings). In many nomenclatures, grouped synonyms are collected under

a so-called code (i.e., a unique alphanumeric string) assigned to all of the terms in the group.

Nomenclatures have many purposes: to enhance interoperability and integration, to allow synonymous terms to be retrieved regardless of which specific synonym is entered as a query, to support comprehensive analyses of textual data, to express detail, to tag information in textual documents, and to drive down the complexity of documents by uniting synonymous terms under a common code. Sets of documents held in more than one Big Data resource can be harmonized under a nomenclature by substituting or appending a nomenclature code to every nomenclature term that appears in any of the documents. [Glossary Interoperability, Data integration, Plesionymy, Polysemy, Vocabulary, Uniqueness, String]

In the case of "renal cell carcinoma," if all of the 50+ synonymous terms, appearing anywhere in a medical text, were tagged with the code "C938500," then a search engine could retrieve documents containing this code, regardless of which specific synonym was queried (e.g., a query on Grawitz tumor would retrieve documents containing the word "hypernephroid tumor"). To do so the search engine would simply translate the query word, "Grawitz tumor" into its nomenclature code "C938500" and would pull every record that had been tagged by the code.

Traditionally, nomenclature coding, much like language translation, has been considered a specialized and highly detailed task that is best accomplished by human beings. Just as there are highly trained translators who will prepare foreign language versions of popular texts, there are highly trained coders, intimately familiar with specific nomenclatures, who create tagged versions of documents. Tagging documents with nomenclature codes is serious business. If the coding is flawed the consequences can be dire. In 2009 the Department of Veterans Affairs sent out hundreds of letters to veterans with the devastating news that they had contracted Amyotrophic Lateral Sclerosis, also known as Lou Gehrig's disease, a fatal degenerative neurologic condition. About 600 of the recipients did not, in fact, have the disease. The VA retracted these letters, attributing the confusion to a coding error [5]. Coding text is difficult. Human coders are inconsistent, idiosyncratic, and prone to error. Coding accuracy for humans seems to fall in the range of 85%–90% [6]. [Glossary Accuracy versus precision]

When dealing with text in gigabyte and greater quantities, human coding is simply out of the question. There is not enough time or money or talent to manually code the textual data contained in Big Data resources. Computerized coding (i.e., autocoding) is the only practical solution.

Autocoding is a specialized form of machine translation, the field of computer science wherein meaning is drawn from narrative text. Not surprisingly, autocoding algorithms have been adopted directly from the field of machine translation, particularly algorithms for natural language processing. A popular approach to autocoding involves using the natural rules of language to find words or phrases found in text and matching them to nomenclature terms. Ideally the terms found in text are correctly matched to their equivalent nomenclature terms, regardless of the way that the terms were expressed in the text.

For instance, the term "adenocarcinoma of lung" has much in common with alternate terms that have minor variations in word order, plurality, inclusion of articles, terms split by a word inserted for informational enrichment, and so on. Alternate forms would be "adenocarcinoma of the lung," "adenocarcinoma of the lungs," "lung adenocarcinoma," and "adenocarcinoma found in the lung." A natural language algorithm takes into account grammatical variants, allowable alternate term constructions, word roots (i.e., stemming), and syntax variation. Clever improvements on natural language methods might include string similarity scores, intended to find term equivalences in cases where grammatical methods come up short. [Glossary Algorithm, Syntax, Machine translation, Natural language processing]

A limitation of the natural language approach to autocoding is encountered when synonymous terms lack etymologic commonality. Consider the term "renal cell carcinoma." Synonyms include terms that have no grammatical relationship with one another. For example, hypernephroma, and Grawitz tumor are synonyms for renal cell carcinoma. It is impossible to compute the equivalents among these terms through the implementation of natural language rules or word similarity algorithms. The only way of obtaining adequate synonymy is through the use of a comprehensive nomenclature that lists every synonym for every canonical term in the knowledge domain.

Setting aside the inability to construct equivalents for synonymous terms that share no grammatical roots, the best natural language autocoders are pitifully slow. The reason for the slowness relates to their algorithm, which requires the following steps, at a minimum: parsing text into sentences; parsing sentences into grammatical units; re-arranging the units of the sentence into grammatically permissible combinations; expanding the combinations based on stem forms of words; allowing for singularities and pluralities of words, and matching the allowable variations against the terms listed in the nomenclature. A typical natural language autocoder parses text at about 1 kilobyte per second, which is equivalent to a terabyte of text every 30 years. Big Data resources typically contain many terabytes of data; thus, natural language autocoding software is unsuitable for translating Big Data resources. This being the case, what good are they?

Natural language autocoders have value when they are employed at the time of data entry. Humans type sentences at a rate far less than 1 kilobyte per second, and natural language autocoders can keep up with typists, inserting codes for terms, as they are typed. They can operate much the same way as auto-correct, auto-spelling, look-ahead, and other commonly available crutches intended to improve or augment the output of plodding human typists.

– **Recoding and speed**

It would seem that by applying the natural language parser at the moment when the data is being prepared, all of the inherent limitations of the algorithm can be overcome. This belief, popularized by developers of natural language software, and perpetuated by a generation of satisfied customers, ignores two of the most important properties that must be preserved in Big Data resources: longevity, and curation. [Glossary Curator]

Nomenclatures change over time. Synonymous terms and the codes will vary from year to year as new versions of old nomenclature are published and new nomenclatures are developed. In some cases, the textual material within the Big Data resource will need to be annotated using codes from nomenclatures that cover informational domains that were not anticipated when the text was originally composed.

Most of the people who work within an information-intensive society are accustomed to evanescent data; data that is forgotten when its original purpose is served. Do we really want all of our old e-mails to be preserved forever? Do we not regret our earliest blog posts, Facebook entries, and tweets? In the medical world, a code for a clinic visit or a biopsy diagnosis, or a reportable transmissible disease will be used in a matter of minutes or hours; maybe days or months. Few among us place much value on textual information preserved for years and decades. Nonetheless, it is the job of the Big Data manager to preserve resource data over years and decades. When we have data that extends back, over decades, we can find and avoid errors that would otherwise reoccur in the present, and we can analyze trends that lead us into the future.

To preserve its value, data must be constantly curated, adding codes that apply to currently available nomenclatures. There is no avoiding the chore; the entire corpus of textual data held in the Big Data resource needs to be recoded again and again, using modified versions of the original nomenclature, or using one or more new nomenclatures. This time, an autocoding application will be required to code huge quantities of textual data (possibly terabytes), quickly. Natural language algorithms, which depend heavily on regex operations (i.e., finding word patterns in text) are too slow to do the job. [Glossary RegEx]

A faster alternative is so-called lexical parsing. This involves parsing text, word by word, looking for exact matches between runs of words and entries in a nomenclature. When a match occurs, the words in the text that matched the nomenclature term are assigned the nomenclature code that corresponds to the matched term. Here is one possible algorithmic strategy for autocoding the sentence: "Margins positive malignant melanoma." For this example, you would be using a nomenclature that lists all of the tumors that occur in humans. Let us assume that the terms "malignant melanoma," and "melanoma" are included in the nomenclature. They are both assigned the same code, for example "Q5673013," because the people who wrote the nomenclature considered both terms to be biologically equivalent.

Let us autocode the diagnostic sentence, "Margins positive malignant melanoma":

1. Begin parsing the sentence, one word at a time. The first word is "Margins." You check against the nomenclature, and find no match. Save the word "margins." We will use it in step 2.
2. You go to the second word, "positive" and find no matches in the nomenclature. You retrieve the former word "margins" and check to see if there is a 2-word term, "margins positive." There is not. Save "margins" and "positive" and continue.
3. You go to the next word, "malignant." There is no match in the nomenclature. You check to determine whether the 2-word term "positive malignant" and the 3-word term "margins positive malignant" are in the nomenclature. They are not.

4. You go to the next word, "melanoma." You check and find that melanoma is in the nomenclature. You check against the two-word term "malignant melanoma," the three-word term "positive malignant melanoma," and the four-word term "margins positive malignant melanoma." There is a match for "malignant melanoma" but it yields the same code as the code for "melanoma."
5. The autocoder appends the code, "Q5673013" to the sentence, and proceeds to the next sentence, where it repeats the algorithm.

The algorithm seems like a lot of work, requiring many comparisons, but it is actually much more efficient than natural language parsing. A complete nomenclature, with each nomenclature term paired with its code, can be held in a single variable, in volatile memory. Look-ups to determine whether a word or phrase is included in the nomenclature are also fast. As it happens, there are methods that will speed things along. In Section 2.7, we will see a 12-line autocoder algorithm that can parse through terabytes of text at a rate that is much faster than commercial-grade natural language autocoders [7]. [Glossary Variable]

Another approach to the problem of recoding large volumes of textual data involves abandoning the attempt to autocode the entire corpus, in favor of on-the-fly autocoding, when needed. On-the-fly autocoding involves parsing through a text of any size, and searching for all the terms that match one particular concept (i.e., the search term).

Here is a general algorithm on-the-fly coding [8]. This algorithm starts with a query term and seeks to find every synonym for the query term, in any collection of Big Data resources, using any convenient nomenclature.

1. The analyst starts with a query term submitted by a data user. The analyst chooses a nomenclature that contains his query term, as well as the list of synonyms for the term. Any vocabulary is suitable, so long as the vocabulary consists of term/code pairs, where a term and its' synonyms are all paired with the same code.
2. All of the synonyms for the query term are collected together. For instance the 2004 version of a popular medical nomenclature, the Unified Medical Language System, had 38 equivalent entries for the code C0206708, nine of which are listed here:

```
C0206708|Cervical Intraepithelial Neoplasms
C0206708|Cervical Intraepithelial Neoplasm
C0206708|Intraepithelial Neoplasm, Cervical
C0206708|Intraepithelial Neoplasms, Cervical
C0206708|Neoplasm, Cervical Intraepithelial
C0206708|Neoplasms, Cervical Intraepithelial
C0206708|Intraepithelial Neoplasia, Cervical
C0206708|Neoplasia, Cervical Intraepithelial
C0206708|Cervical Intraepithelial Neoplasia
```

If the analyst had chosen to search on "Cervial Intraepithelial Neoplasia," his term will be attached to the 38 synonyms included in the nomenclature.

3. One-by-one, the equivalent terms are matched against every record in every Big Data resource available to the analyst.
4. Records are pulled that contain terms matching any of the synonyms for the term selected by the analyst.

In the case of the example, this would mean that all 38 synonymous terms for "Cervical Intraepithelial Neoplasms" would be matched against the entire set of data records. The benefit of this kind of search is that data records that contain any search term, or its nomenclature equivalent, can be extracted from multiple data sets in multiple Big Data resources, as they are needed, in response to any query. There is no pre-coding, and there is no need to match against nomenclature terms that have no interest to the analyst. The drawback of this method is that it multiplies the computational task by the number of synonymous terms being searched, 38-fold in this example. Luckily, there are published methods for conducting simple and fast synonym searches, using precompiled concordances [8].

Section 2.6. Case Study: Instantly Finding the Precise Location of Any Atom in the Universe (Some Assembly Required)

> *There's as many atoms in a single molecule of your DNA as there are stars in the typical galaxy. We are, each of us, a little universe.*
>
> *Neil deGrasse Tyson, Cosmos*

If you have sat through an introductory course in Computer Science, you are no doubt familiar with three or four sorting algorithms. Indeed, most computer science books devote a substantial portion of their texts to describing sorting algorithms. The reason for this infatuation with sorting is that all sorted lists can be searched nearly instantly, regardless of the size of the list. The so-called binary algorithm for searching a sorted list is incredibly simple. For the sake of discussion, let us consider an alphabetically sorted list of 1024 words. I want to determine if the word "kangaroo" is in the list; and, if so, its exact location in the list. Here is how a binary search would be conducted.

1. Go to the middle entry of the list.
2. Compare the middle entry to the word "kangaroo." If the middle entry comes earlier in the alphabet than "kangaroo," then repeat step 1, this time ignoring the first half of the list and using only the second half of the list (i.e., going to the middle entry of the second half of the file). Otherwise, go to step 1, this time ignoring the second half of the list and using only the first half.

These steps are repeated until you come to the location where kangaroo resides, or until you have exhausted the list without finding your kangaroo.

Each cycle of searching cuts the size of the list in half. Hence, a search through a sorted list of 1024 items would involve, at most, 10 cycles through the two-step algorithm (because $1024 = 2^{10}$).

Every computer science student is expected to write her own binary search script. Here is a simple script, binary.py, that does five look-ups through a sorted numeric list, reporting on which items are found, and which items are not.

```
def Search(search_list, search_item):
  first_item = 0
  last_item = len(search_list)-1
  found = False
  while (first_item <= last_item) and not found:
    middle = (first_item + last_item)//2
    if search_list[middle] == search_item:
      found = True
    else:
      if search_item < search_list[middle]:
        last_item = middle - 1
      else:
        first_item = middle + 1
  return found
sorted_list = [4, 5, 8, 15, 28, 29, 30, 45, 67, 82, 99, 101, 1002]
for item in [3, 7, 28, 31, 45, 1002]:
  print(Search(sorted_list, item))

output:
False
False
True
False
True
True
```

Let us say, just for fun, we wanted to search through a sorted list of every atom in the universe. First we would take each atom in the universe and assign it a location. Then we would sort the locations based on their distances from the center of the center of the universe, which is apparently located at the tip of my dog's left ear. We could then substitute the sorted atom list for the sorted_list in the binary.py script, shown above.

How long would it take to search all the atoms of the universe, using the binary.py script. As it happens, we could find the list location for any atom in the universe, almost instantly. The reason is that there are only about 2 ̂260 atoms in the known universe. This means that the algorithm would required, at the very most, 260 2-step cycles. Each cycle is very fast, requiring only that we compare the search atom's distance from my dog's ear, against the middle atom of the list.

Of course, composing the list of atom locations may pose serious difficulties, and we might need another universe, much larger than our own, to hold the sorted list that we

create. Nonetheless, a valid point emerges; that binary searches are fast, and the time to completion of a binary search is not significantly lengthened by any increase in the number of items in the list. Had we chosen, we could have annotated the items of sorted_list with any manner of information (e.g., locations in a file, nomenclature code, links to web addresses, definitions of the items, metadata), so that our binary searches would yield something more useful than the location of the item in the list.

Section 2.7. Case Study (Advanced): A Complete Autocoder (in 12 Lines of Python Code)

Software is a gas; it expands to fill its container.

Nathan Myhrvold

This script requires two external files:

1. The nomenclature file that will be converted into a Python dictionary, wherein each term is a dictionary key, and each nomenclature code is a value assigned to a term. [Glossary Dictionary]

   Here are a few sample lines from the nomenclature file (nomenclature_dict.txt, in this case):

   ```
   oropharyngeal adenoid cystic adenocarcinoma , C6241000
   peritoneal mesothelioma , C7633000
   benign tumour arising from the exocrine pancreas , C4613000
   basaloid penile squamous cell cancer , C6980000
   cns malignant soft tissue tumor , C6758000
   digestive stromal tumour of stomach , C5806000
   bone with malignancy , C4016000
   benign mixed tumor arising from skin , C4474000
   ```

2. The file containing a corpus of sentences that will be autocoded by the script.

   Here are a few sample lines from the corpus file (tumorabs.txt, in this case):

   ```
   local versus diffuse recurrences of meningiomas factors correlated
   to the extent of the recurrence

   the effect of an unplanned excision of a soft tissue sarcoma on
   prognosis

   obstructive jaundice associated burkitt lymphoma mimicking
   pancreatic carcinoma

   efficacy of zoledronate in treating persisting isolated tumor
   cells in bone marrow in patients with breast cancer a phase ii pilot
   study
   ```

```
metastatic lymph node number in epithelial ovarian carcinoma does it
have any clinical significance

extended three dimensional impedance map methods for identifying
ultrasonic scattering sites

aberrant expression of connexin 26 is associated with lung metastasis
of colorectal cancer
```

The 19-line python script, autocode.txt, produces a sentence-by-sentence list of extracted autocoded terms:

```
outfile = open("autocoded.txt", "w")
literalhash = {}
with open("nomenclature_dict.txt") as f:
    for line in f:
       (key, val) = line.split(" , ")
       literalhash[key] = val
corpus_file = open("tumorabs.txt", "r")
for line in corpus_file:
    sentence = line.rstrip()
    outfile.write("\n" + sentence[0].upper() + sentence[1:] + "." +
    "\n")
    sentence_array = sentence.split(" ")
    length = len(sentence_array)
    for i in range(length):
      for place_length in range(len(sentence_array)):
          last_element = place_length + 1
          phrase = ' '.join(sentence_array[0:last_element])
          if phrase in literalhash:
             outfile.write(phrase + " " + literalhash[phrase])
      sentence_array.pop(0)
```

The first seven lines of code are housekeeping chores, in which the external nomenclature is loaded into a Python dictionary (literalhash, in this case), and an external file composed of lines, with one sentence on each line, is opened and prepared for reading, and which another external file, autocoded.txt, is created to accept the script's output. We will not count these first seven lines as belonging to our autocoder because, in all fairness, they are not doing any of the work of autocoding. The meat of the script is the next twelve lines, beginning with "for line in corpus_file."

Here is a sample of the output:

```
Obstructive jaundice associated burkitt lymphoma mimicking pancreatic
carcinoma.
   burkitt lymphoma C7188000
```

```
   lymphoma C7065000
   pancreatic carcinoma C3850000
   carcinoma C2000000

Littoral cell angioma of the spleen.
   littoral cell angioma C8541100
   littoral cell angioma of the spleen C8541100
   angioma C3085000
   angioma of the spleen C8541000

Isolated b cell lymphoproliferative disorder at the dura mater with b
cell chronic lymphocytic leukemia immunophenotype.
   lymphoproliferative disorder C4727100
   b cell chronic lymphocytic leukemia C3163000
   chronic lymphocytic leukemia C3163000
   lymphocytic leukemia C7539000
   leukemia C3161000
```

By observing a few samples of autocoded lines of text, we can see that the autocoder extracts all cancer terms, and supplies its nomenclature code, regardless of whether a term is contained within a longer term.

For example, the autocoder managed to find four terms within the sentence "Littoral cell angioma of the spleen," these being: littoral cell angioma, littoral cell angioma of the spleen, angioma, and angioma of the spleen. The ability to extract every valid term, even when they are subsumed by larger terms, guarantees that a query term and all its synonyms will always be retrieved, if the query term happens to be a valid nomenclature term.

This short autocoding script comes with a few advantages that are of particular interest to Big Data professionals:

– Scalable to any size

All nomenclatures are small. Most of us have a working vocabulary of a few thousand words. Most dictionaries are smaller, containing maybe 60,000 words. The most extreme case of verbiage about verbiage is The 20-volume Oxford English Dictionary, which contains about 170,000 entries. Even in this case, slurping the entire list of Oxford English dictionary items would be a simple matter for any modern computer.

Most importantly, the autocoding algorithm imposes no limits on the size of the Big Data corpus. The software proceeds line-by-line until the task is complete. Memory requirements and other issues of scalability are not a problem.

– Fast

On my modest desktop computer, the 12-line autocoding algorithm processes text at the rate of 1 megabyte every two seconds. A fast and powerful computer, using the same algorithm, would be expected to parse at rates of 1 gigabytes of text per second, or greater.

– Repeatable

Code a gigabyte of data in the morning. Do it all over again in the afternoon. Use another version of the nomenclature, or use a different nomenclature, entirely. Recoding is not a problem.

– Simple and adaptable, with easily maintained code

The larger the program, the more difficult it is to find bugs, or to recover from errors produced when the code is modified. It is nearly impossible to inflict irreversible damage upon a simple, 12-line script. As a general rule, tiny scripts are seldom a problem if you maintain records of where the scripts are located, how the scripts are used, and how the scripts are modified over time.

– Reveals the dirty little secret that every programmer knows, but few are willing to admit.

Virtually all useful algorithms can be implemented in a few lines of code; autocoders are no exception. The thousands, or millions, of lines of code in just about any commercial software application are devoted, in one way or another, to the graphic user interface.

Section 2.8. Case Study: Concordances as Transformations of Text

Interviewer: Is there anything from home that you brought over with you to set up for yourself? Creature comforts?
Hawkeye: I brought a book over.
Interviewer: What book?
Hawkeye: The dictionary. I figure it's got all the other books in it.
*Interview with the character Hawkeye, played by Alan Alda, from television show M*A*S*H*

A transform is a mathematical operation that takes a function, a signal, or a set of data and changes it into something else, that is easier to work with than the original data. The concept of the transform is a simple but important idea that has revolutionized many scientific fields including electrical engineering, digital signal processing, and data analysis. In the field of digital signal processing, data in the time domain (i.e., wherein the amplitude of a measurement varies over time, as in a signal), is commonly transformed into the frequency domain (i.e., wherein the original data can be assigned to amplitude values for a range of frequencies). There are dozens, possibly hundreds, of mathematical transforms that enable data analysts to move signal data between forward transforms (e.g., time domain to frequency domain), and their inverse counterparts (e.g., frequency domain to time domain). [Glossary Transform, Signal, Digital signal, Digital Signal Processing, DSP, Fourier transform, Burrows-Wheeler transform]

A concordance is transform, for text. A concordance takes a linear text and transforms it a word-frequency distribution list; which can reversed as needed. Like any good transform, we can expect to find circumstances when it is easier to perform certain types of operations on the transformed data than on the original data. [Glossary Concordance]

Here is an example, from the Python script proximate_words.py, where we use a concordance to list the words in close proximity to the concordance entries (i.e., the words contained in the text). In this script, we use the previously constructed (vida supra) concordance of the Gettysburg address.

```
import string
infile = open ("concordance.txt", "r")
places = []
word_array = []
concordance_hash = {}
words_hash = {}
for line in infile:
   line = line.rstrip()
   line_array = line.split(" ")
   word = line_array[0]
   places = line_array[1]
   places_array = places.split(",")
   words_hash[word] = places_array
   for word_position in places_array:
     concordance_hash[word_position] = word
for k, v in words_hash.items():
   print(k, end=" - \n")
   for items in v:
     n=0
     while n < 5:
       nextone = str(int(items) + n)
       if nextone in concordance_hash:
         print(concordance_hash[nextone], end=" ")
       n=n+1
     print()
   print()
```

The script produces a list of the words from the Gettysburg address, along with short sequences of the text that follow each occurrence of the word in the text, as shown in this sampling from the output file:

```
to -
to the proposition that all
to dedicate a portion of
```

```
to add or detract. The
to be dedicated here to
to the unfinished work which
to be here dedicated to
to the great task remaining
to that cause for which

dedicated -
dedicated to the proposition that
dedicated can long endure. We
dedicated here to the unfinished
dedicated to the great task
```

Inspecting some of the output, we see that the word "to" appears 8 times in the Gettysburg address. We used the concordance to reconstruct four words that follow the word "to" wherever it occurs in the text. Likewise we see that the word "dedicated" occurs 4 times in the text, and the concordance tells us the four words that follow at each of the locations where "dedicated" appears. We can construct these proximity phrases very quickly, because the concordance tells us the exact location of the words in the text. If we were working from the original text, instead of its transform (i.e., the concordance), then our algorithm would run much more slowly, because each word would need to be individually found and retrieved, by parsing every word in the text, sequentially.

Section 2.9. Case Study (Advanced): Burrows Wheeler Transform (BWT)

> *All parts should go together without forcing. You must remember that the parts you are reassembling were disassembled by you. Therefore, if you can't get them together again, there must be a reason. By all means, do not use a hammer.*
>
> *IBM Manual, 1925*

One of the most ingenious transforms in the field of data science is the Burrows Wheeler transform. Imagine an algorithm that takes a corpus of text and creates an output string consisting of a transformed text combined with its own word index, in a format that can be compressed to a smaller size than the compressed original file. The Burrows Wheeler Transform does all this, and more [9,10]. A clever informatician may find many ways to use the BWT transform in search and retrieval algorithms and in data merging projects [11]. Using the BWT file, you can re-compose the original file, or you can find any portion of a file preceding or following any word from the file [12]. [Glossary Data merging, Data fusion]

Excellent discussions of the algorithm are available, along with implementations in several languages [9,10,13]. The Python script, bwt.py, shown here, is a modification of a script available on Wikipedia [13]. The script executes the BWT algorithm in just three

lines of code. In this example, the input string is a excerpt from Lincoln's Gettysburg address [12].

```
input = "four score and seven years ago our fathers brought forth upon"
input = input + " this continent a new nation conceived in liberty and"
input = input + "\0"
table = sorted(input[i:] + input[:i] for i in range(len(input)))
last_column = [row[-1:] for row in table]
print("".join(last_column))
```

Here is the transformed output:

```
dtsyesnsrtdnwaordnhn  efni n snenryvcvnhbsn  uatttgl tthe oioe oaai
eogipccc
fr fuuuobaeoerri nhra naro ooieet
```

Admittedly, the output does not look like much. Let us juxtapose our input string and our BWT's transform string:

```
four score and seven years ago our fathers brought forth upon this
continent a new nation conceived in liberty and
dtsyesnsrtdnwaordnhn  efni n snenryvcvnhbsn  uatttgl tthe oioe oaai
eogipcccfr fuuuobaeoerri nhra naro ooieet
```

We see that the input string and the transformed output string both have the same length, so there doesn't seem to be any obvious advantage to the transform. If we look a bit closer, though, we see that the output string consists largely of runs of repeated individual characters, repeated substrings, and repeated spaces (e.g., "ttt" "uuu"). These frequent repeats in the transform facilitate compression algorithms that hunt for repeat patterns. BWT's facility for creating runs of repeated characters accounts for its popularity in compression software (e.g., the Bunzip compression utility).

The Python script, bwt_inverse.py, computes the inverse BWT to re-construct the original input string. Notice that the inverse algorithm is implemented in just the last four lines of the python code (the first five lines re-created the forward BWT transform) [12]

```
input = "four score and seven years ago our fathers brought forth upon"
input = input + " this continent a new nation conceived in liberty and"
input = input + "\0"
table = sorted(input[i:] + input[:i] for i in range(len(input)))
last_column = [row[-1:] for row in table]
#The first lines re-created the bwt transform

#The next four lines compute the inverse transform
table = [""] * len(last_column)
for i in range(len(last_column)):
   table = sorted(last_column[i] + table[i] for i in range(len(input)))
print([row for row in table if row.endswith("\0")][0])
```

As we would expect, the output of the bwt_inverse.py script, is our original input string:

```
four score and seven years ago our fathers brought forth upon this
continent a new nation conceived in liberty and
```

The charm of the BWT transform is demonstrated when we create an implementation that parses the input string word-by-word; not character-by-character.

Here is the Python script, bwt_trans_inv.py, that transforms an input string, word-by-word, producing its transform; then reverses the process to yield the original string, as an array of words. As an extra feature, the script produces the first column, as an array, of the transform table [12]. [Glossary Numpy]

```
import numpy as np
input = "\0 four score and seven years ago our fathers brought forth upon"
input = input + " this continent a new nation conceived in liberty and"
word_list = input.rsplit()
table = sorted(word_list[i:] + word_list[:i] for i in range(len
(word_list)))
last_column = [row[-1:] for row in table]
first_column = [row[:1] for row in table]
print("First column of the transform table:\n" + str(first_column) +
"\n")
table = [""] * len(last_column)
for i in range(len(last_column)):
   table = sorted(str(last_column[i]) + " " + str(table[i]) for i in
range(len(word_list)))
original = [row for row in table][0]
print("Inverse transform, as a word array:\n" + str(original))
```

Here is the output of the bwt_trans_inv.py script. Notice once more that the word-by-word transform was implemented in 3 lines of code, and the inverse transform was implemented in four lines of code.

```
First column of the transform table:
[['\x00'], ['a'], ['ago'], ['and'], ['and'], ['brought'],
['conceived'], ['continent'], ['fathers'], ['forth'], ['four'],
['in'], ['liberty'], ['nation'], ['new'], ['our'], ['score'],
['seven'], ['this'], ['upon'], ['years']]

Inverse transform, as a word array:
['\x00'] ['four'] ['score'] ['and'] ['seven'] ['years'] ['ago']
['our'] ['fathers'] ['brought'] ['forth'] ['upon'] ['this']
['continent'] ['a'] ['new'] ['nation'] ['conceived'] ['in']
['liberty'] ['and']
```

The first column of the transform, created in the forward BWT, is a list of the words in the input string, in alphabetic order. Notice that words that occurred more than one time in the input text were repeated in the first column of the transform table (i.e., [and], [and] in the example sentence). Hence, the transform yields all the words from the original input, along with their frequency of occurrence in the text. As expected, the inverse of the transform yields our original input string.

Glossary

Accuracy versus precision Accuracy measures how close your data comes to being correct. Precision provides a measurement of reproducibility (i.e., whether repeated measurements of the same quantity produce the same result). Data can be accurate but imprecise. If you have a 10 pound object, and you report its weight as 7.2376 pounds, on every occasion when the object is weighed, then your precision is remarkable, but your accuracy is dismal.

Algorithm An algorithm is a logical sequence of steps that lead to a desired computational result. Algorithms serve the same function in the computer world as production processes serve in the manufacturing world and as pathways serve in the world of biology. Fundamental algorithms can be linked to one another, to create new algorithms (just as biological pathways can be linked). Algorithms are the most important intellectual capital in computer science. In the past half century, many brilliant algorithms have been developed for the kinds of computation-intensive work required for Big Data analysis [14,15].

Binary data Computer scientists say that there are 10 types of people. Those who think in terms of binary numbers, and those who do not. Pause for laughter and continue. All digital information is coded as binary data. Strings of 0s and 1s are the fundamental units of electronic information. Nonetheless, some data is more binary than other data. In text files, 8-bite sequences are converted into decimals in the range of 0–256, and these decimal numbers are converted into characters, as determined by the ASCII standard. In several raster image formats (i.e., formats consisting of rows and columns of pixel data), 24-bit pixel values are chopped into red, green and blue values of 8-bits each. Files containing various types of data (e.g., sound, movies, telemetry, formatted text documents), all have some kind of low-level software that takes strings of 0s and 1s and converts them into data that has some particular meaning for a particular use. So-called plain-text files, including HTML files and XML files are distinguished from binary data files and referred to as plain-text or ASCII files. Most computer languages have an option wherein files can be opened as "binary," meaning that the 0s and 1s are available to the programmer, without the intervening translation into characters or stylized data.

Burrows-Wheeler transform Abbreviated as BWT, the Burrows-Wheeler transform produces a compressed version of an original file, along with a concordance to the contents of the file. Using a reverse BWT, you can reconstruct the original file, or you can find any portion of a file preceding or succeeding any location in the file. The BWT transformation is an amazing example of simplification, applied to informatics. A detailed discussion of the BWT is found in Section 2.9, "Case Study (Advanced): Burrows Wheeler Transform."

Class A class is a group of objects that share a set of properties that define the class and that distinguish the members of the class from members of other classes. The word "class," lowercase, is used as a general term. The word "Class," uppercase, followed by an uppercase noun (e.g., Class Animalia), represents a specific class within a formal classification.

Classification A system in which every object in a knowledge domain is assigned to a class within a hierarchy of classes. The properties of superclasses are inherited by the subclasses. Every class has one immediate superclass (i.e., parent class), although a parent class may have more than one immediate subclass (i.e., child class). Objects do not change their class assignment in a classification, unless there

was a mistake in the assignment. For example, a rabbit is always a rabbit, and does not change into a tiger. Classifications can be thought of as the simplest and most restrictive type of ontology, and serve to reduce the complexity of a knowledge domain [16].

Classifications can be easily modeled in an object-oriented programming language and are non-chaotic (i.e., calculations performed on the members and classes of a classification should yield the same output, each time the calculation is performed). A classification should be distinguished from an ontology. In an ontology a class may have more than one parent class and an object may be a member of more than one class. A classification can be considered a special type of ontology wherein each class is limited to a single parent class and each object has membership in one and only one class.

Coding The term "coding" has three very different meanings depending on which branch of science influences your thinking. For programmers, coding means writing the code that constitutes a computer programmer. For cryptographers, coding is synonymous with encrypting (i.e., using a cipher to encode a message). For medics, coding is calling an emergency team to handle a patient in extremis. For informaticians and library scientists, coding involves assigning a alphanumeric identifier, representing a concept listed in a nomenclature, to a term. For example, a surgical pathology report may includes the diagnosis, "Adenocarcinoma of prostate." A nomenclature may assign a code C4863000 that uniquely identifies the concept "Adenocarcinoma." Coding the report may involve annotating every occurrence of the work "Adenocarcinoma" with the "C4863000" identifier. For a detailed explanation of coding, and its importance for searching and retrieving data, see the full discussion in Section 3.4, "Autoencoding and Indexing with Nomenclatures."

Concordance A concordance is an index consisting of every word in the text, along with every location wherein each word can be found. It is computationally trivial to reconstruct the original text from the concordance. Before the advent of computers, concordances fell into the provenance of religious scholars, who painstakingly recorded the locations of the all words appearing in the Bible, ancient scrolls, and any texts whose words were considered to be divinely inspired. Today, a concordance for a Bible-length book can be constructed in about a second. Furthermore, the original text can be reconstructed from the concordance, in about the same time.

Curator The word "curator" derives from the latin, "curatus," the same root for "curative," indicating that curators "take care of" things. A data curator collects, annotates, indexes, updates, archives, searches, retrieves, and distributes data. Curator is another of those somewhat arcane terms (e.g., indexer, data archivist, lexicographer) that are being rejuvenated in the new millennium. It seems that if we want to enjoy the benefits of a data-centric world, we will need the assistance of curators, trained in data organization.

DSP Abbreviation for Digital Signal Processing.

Data fusion Data fusion is very closely related to data integration. The subtle difference between the two concepts lies in the end result. Data fusion creates a new and accurate set of data representing the combined data sources. Data integration is an on-the-fly usage of data pulled from different domains and, as such, does not yield a residual fused set of data.

Data integration The process of drawing data from different sources and knowledge domains in a manner that uses and preserves the identities of data objects and the relationships among the different data objects. The term "integration" should not be confused with a closely related term, "interoperability." An easy way to remember the difference is to note that **integration applies to data; interoperability applies to software.**

Data merging A nonspecific term that includes data fusion, data integration, and any methods that facilitate the accrual of data derived from multiple sources.

Dictionary In general usage a dictionary is a word list accompanied by a definition for each item. In Python a dictionary is a data structure that holds an unordered list of key/value pairs. A dictionary, as used in Python, is equivalent to an associative array, as used in Perl.

Digital Signal Processing Digital Signal Processing (DSP) is the field that deals with creating, transforming, sending, receiving, and analyzing digital signals. Digital signal processing began as a specialized

subdiscipline of signal processing, another specialized subdiscipline. For most of the twentieth century, many technologic advances came from converting non-electrical signals (temperature, pressure, sound, and other physical signals) into electric signals that could be carried via electromagnetic waves, and later transformed back into physical actions. Because electromagnetic waves sit at the center of so many transform process, even in instances when the input and outputs are non-electrical in nature, the field of electrical engineering and signal processing have paramount importance in every field of engineering. In the past several decades the intermediate signals have been moved from the analog domain (i.e., waves) into the digital realm (i.e., digital signals expressed as streams of 0s and 1s). Over the years, as techniques have developed by which any kind of signal can be transformed into a digital signal, the subdiscipline of digital signal processing has subsumed virtually all of the algorithms once consigned to its parent discipline. In fact, as more and more processes have been digitized (e.g., telemetry, images, audio, sensor data, communications theory), the field of digital signal processing has come to play a central role in data science.

Digital signal A signal is a description of how one parameter varies with some other parameter. The most familiar signals involve some parameter varying over time (e.g., sound is air pressure varying over time). When the amplitude of a parameter is sampled at intervals, producing successive pairs of values, the signal is said to be digitized.

Fourier transform A transform is a mathematical operation that takes a function or a time series (e.g., values obtained at intervals of time) and transforms it into something else. An inverse transform takes the transform function and produces the original function (Fig. 2.1). Transforms are useful when there are operations that can be more easily performed on the transformed function than on the original function. Possibly the most useful transform is the Fourier transform, which can be computed with great speed on modern computers, using a modified form known as the fast Fourier Transform. Periodic functions and waveforms (periodic time series) can be transformed using this method. Operations on the transformed function can sometimes eliminate repeating artifacts or frequencies that occur below a selected threshold (e.g., noise). The transform can be used to find similarities between two signals. When the operations on the transform function are complete, the inverse of the transform can be calculated and substituted for the original set of data (Fig. 2.2).

Identifier A string that is associated with a particular thing (e.g., person, document, transaction, data object), and not associated with any other thing [17]. In the context of Big Data, identification usually involves permanently assigning a seemingly random sequence of numeric digits (0–9) and alphabet characters (a-z and A-Z) to a data object. The data object can be a class of objects.

Indexes Every writer must search deeply into his or her soul to find the correct plural form of "index". Is it "indexes" or is it "indices"? Latinists insist that "indices" is the proper and exclusive plural form. Grammarians agree, reserving "indexes" for the third person singular verb form; "The student indexes his thesis." Nonetheless, popular usage of the plural of "index," referring to the section at the end of a book, is almost always "indexes," the form used herein.

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x)\ e^{-2\pi i x \xi}\,dx$$

$$f(x) = \int_{-\infty}^{\infty} \hat{f}(\xi)\ e^{2\pi i x \xi}\,d\xi$$

FIG. 2.1 The Fourier transform and its inverse. In this representation of the transform, *x* represents time in seconds and the transform variable zeta represents frequency in hertz.

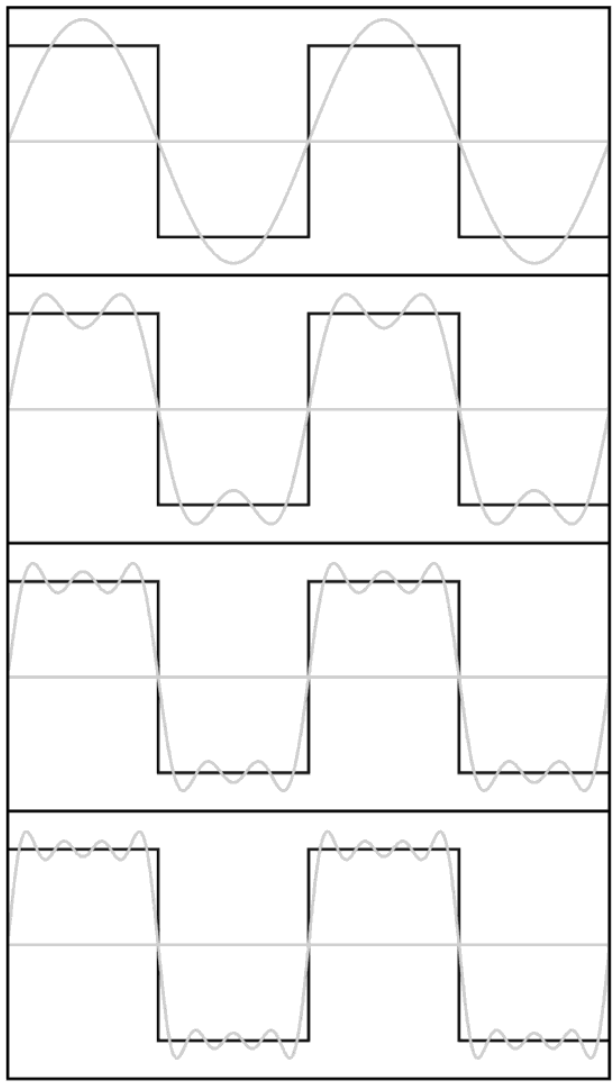


FIG. 2.2 A square wave is approximated by a single sine wave, the sum of two sine waves, three sine waves, and so on. As more components are added, the representation of the original signal or periodic set of data, is more closely approximated. *From Wikimedia Commons.*

Interoperability It is desirable and often necessary to create software that operates with other software, regardless of differences in hardware, operating systems and programming language. Interoperability, though vital to Big Data science, remains an elusive goal.

Machine translation Ultimately, the job of machine translation is to translate text from one language into another language. The process of machine translation begins with extracting sentences from text, parsing the words of the sentence into grammatical parts, and arranging the grammatical parts into an order that imposes logical sense on the sentence. Once this is done, each of the parts can be translated by a dictionary that finds equivalent terms in a foreign language, then re-assembled as a foreign

language sentence by applying grammatical positioning rules appropriate for the target language. Because these steps apply the natural rules for sentence constructions in a foreign language, the process is often referred to as natural language machine translation. It is important to note that nowhere in the process of machine translation is it necessary to find meaning in the source text, or to produce meaning in the output. Good machine translation algorithms preserve ambiguities, without attempting to impose a meaningful result.

Natural language processing A field broadly concerned with how computers interpret human language (i.e., machine translation). At its simplest level this may involve parsing through text and organizing the grammatical units of individual sentences (i.e., tokenization). For example, we might assign the following tokens to the grammatical parts of a sentence: A = adjective, D = determiner, N = noun, P = preposition, V = main verb. A determiner is a word such as "a" or "the", which specifies the noun [18]. Consider the sentence, "The quick brown fox jumped over lazy dogs." This sentence can be grammatically tokenized as:

the::D
quick::A
brown::A
fox::N
jumped::V
over::P
the::D
lazy::A
dog::N

We can express the sentence as the sequence of its tokens listed in the order of occurrence in the sentence: DAANVPDAN. This does not seem like much of a breakthrough, but imagine having a large collection of such token sequences representing every sentence from a large text corpus. With such a data set, we could begin to understand the rules of sentence structure. Commonly recurring sequences, like DAANVPDAN, might be assumed to be proper sentences. Sequences that occur uniquely in a large text corpus are probably poorly constructed sentences. Before long, we might find ourselves constructing logic rules for reducing the complexity of sentences by dropping subsequences which, when removed, yield a sequence that occurs more commonly than the original sequence. For example, our table of sequences might indicate that we can convert DAANVPDAN into NVPAN (i.e., "Fox jumped over lazy dog"), without sacrificing too much of the meaning from the original sentence and preserving a grammatical sequence that occurs commonly in the text corpus.

This short example serves as an overly simplistic introduction to natural language processing. We can begin to imagine that the grammatical rules of a language can be represented by sequences of tokens that can be translated into words or phrases from a second language, and re-ordered according to grammatical rules appropriate to the target language. Many natural language processing projects involve transforming text into a new form, with desirable properties (e.g., other languages, an index, a collection of names, a new text with words and phrases replaced with canonical forms extracted from a nomenclature) [18]. When we use natural language rules to autocode text, the grammatical units are trimmed, reorganized, and matched against concept equivalents in a nomenclature.

Ngrams Ngrams are subsequences of text, of length n words. A complete collection of ngrams consists of all of the possible ordered subsequences of words in a text. Because sentences are the basic units of statements and ideas, when we speak of ngrams, we are confining ourselves to ngrams of sentences. Let us examine all the ngrams for the sentence, "Ngrams are ordered word sequences."

```
Ngrams (1-gram)
are (1-gram)
ordered (1-gram)
word (1-gram)
```

```
sequences (1-gram)
Ngrams are (2-gram)
are ordered (2-gram)
ordered word (2-gram)
word sequences (2-gram)
Ngrams are ordered (3-gram)
are ordered word (3-gram)
ordered word sequences (3-gram)
Ngrams are ordered word (4-gram)
are ordered word sequences (4-gram)
Ngrams are ordered word sequences (5-gram)
```

Here is a short Python script, ngram.py, that will take a sentence and produce a list of all the contained ngrams.

```
import string
text = "ngrams are ordered word sequences"
partslist = []
ngramlist = {}
text_list = text.split(" ")
while(len(text_list) > 0):
   partslist.append(" ".join(text_list))
   del text_list[0]
for part in partslist:
   previous = ""
   wordlist = part.split(" ")
   while(len(wordlist) > 0):
      ngramlist[(" ".join(wordlist))] = ""
      firstword = wordlist[0]
      del wordlist[0]
      ngramlist[firstword] = ""
      previous = previous + " " + firstword
      previous = previous.strip()
      ngramlist[previous] = ""
for key in sorted(ngramlist):
      print(key)
exit
```

```
output:
are
are ordered
are ordered word
are ordered word sequences
ngrams
ngrams are
ngrams are ordered
ngrams are ordered word
ngrams are ordered word sequences
ordered
ordered word
ordered word sequences
sequences
```

```
word
word sequences
```

The ngram.py script can be easily modified to parse through all the sentences of any text, regardless of length, building the list of ngrams as it proceeds.

Google has collected ngrams from scanned literature dating back to 1500. The public can enter their own ngrams into Google's ngram viewer, and receive a graph of the published occurrences of the phrase, through time [18]. We can use the Ngram viewer to find trends (e.g., peaks, valleys and periodicities) in data. Consider the Google Ngram Viewer results for the two-word ngram, "yellow fever" (Fig. 2.3).

We see that the term "yellow fever" (a mosquito-transmitted hepatitis) appeared in the literature beginning about 1800, with several subsequent peaks. The dates of the peaks correspond roughly to outbreaks of yellow fever in Philadelphia (epidemic of 1793), New Orleans (epidemic of 1853), with United States construction efforts in the Panama Canal (1904–14), and with well-documented WWII Pacific outbreaks (about 1942). Following the 1942 epidemic an effective vaccine was available, and the incidence of yellow fever, as well as the literature occurrences of the "yellow fever" n-gram, dropped precipitously. In this case, a simple review of n-gram frequencies provides an accurate chart of historic yellow fever outbreaks [19,18].

Nomenclature A nomenclatures is a listing of terms that cover all of the concepts in a knowledge domain. A nomenclature is different from a dictionary for three reasons: 1) the nomenclature terms are not annotated with definitions, 2) nomenclature terms may be multi-word, and 3) the terms in the nomenclature are limited to the scope of the selected knowledge domain. In addition, most nomenclatures group synonyms under a group code. For example, a food nomenclature might collect submarine sandwich, hoagie, po' boy, grinder, hero, and torpedo under an alphanumeric code such as "F63958." Nomenclatures simplify textual documents by uniting synonymous terms under a common code. Documents that have been coded with the same nomenclature can be integrated with other documents that have been similarly coded, and queries conducted over such documents will yield the same results, regardless of which term is entered (i.e., a search for either hoagie, or po' boy will retrieve the same information, if both terms have been annotated with the synonym code, "F63948"). Optimally, the canonical concepts listed in the nomenclature are organized into a hierarchical classification [20,21,12].

Nomenclature mapping Specialized nomenclatures employ specific names for concepts that are included in other nomenclatures, under other names. For example, medical specialists often preserve their favored names for concepts that cross into different fields of medicine. The term that pathologists use for a certain benign fibrous tumor of the skin is "fibrous histiocytoma," a term spurned by

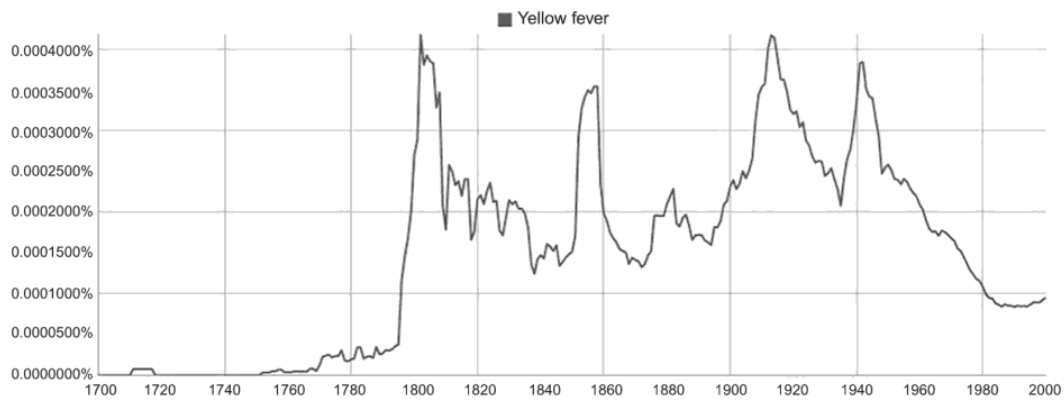


FIG. 2.3 Google Ngram for the phrase "yellow fever," counting occurrences of the term in a large corpus, from the years 1700–2000. Peaks roughly correspond to yellow fever epidemics. *Source: Google Ngram viewer, with permission from Google.*

dermatologists, who prefer to use "dermatofibroma" to describe the same tumor. As another horrifying example, the names for the physiologic responses caused by a reversible cerebral vasoconstricitve event include: thunderclap headache, Call-Fleming syndrome, benign angiopathy of the central nervous system, postpartum angiopathy, migrainous vasospasm, and migraine angiitis. The choice of term will vary depending on the medical specialty of the physician (e.g., neurologist, rheumatologist, obstetrician). To mitigate the discord among specialty nomenclatures, lexicographers may undertake a harmonization project, in which nomenclatures with overlapping concepts are mapped to one another.

Numpy Numpy (Numerical Python) is an open source extension to Python that supports matrix operations, as well as a rich assortment of mathematical functions. Numpy can be easily downloaded from sourceforge.net: http://sourceforge.net/projects/numpy/. Here is a short Python script, numpy_dot.py, that creates a 3x3 matrix, inverts the matrix, and calculates the dot produce of the matrix and its inverted counterpart.

```
import numpy
from numpy.linalg import inv
a = numpy.array([[1,4,6], [9,15,55], [62,-5, 4]])
print(a)
print(inv(a))
c = numpy.dot(a, inv(a))
print(numpy.round_(c))
```

The numpy_dot.py script employs numpy, numpy's linear algebra module, and numpy's matrix inversion method, and the numpy dot product method. Here is the output of the script, displaying the original matrix, its inversion, and the dot product, which happens to be the unity matrix:

```
c:\ftp\py>numpy_dot.py
[[ 1  4  6]
 [ 9 15 55]
 [62 -5  4]]
[[ 4.19746899e-02 -5.76368876e-03 1.62886856e-02]
 [ 4.22754041e-01 -4.61095101e-02 -1.25297582e-04]
 [-1.22165142e-01 3.17002882e-02 -2.63124922e-03]]
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

Parsing Much of computer programming involves parsing; moving sequentially through a file or some sort of data structure and performing operations on every contained item, one item at a time. For files, this might mean going through a text file line by line, or sentence by sentence. For a data file, this might mean performing an operation on each record in the file. For in-memory data structures, this may mean performing an operation on each item in a list or a tuple or a dictionary.

The parse_directory.py script prints all the file names and subdirectory names in a directory tree.

```
import os
for root, dirs, files in os.walk(".", topdown=False):
  for filename in files:
     print(os.path.join(root, filename))
  for dirname in dirs:
     print(os.path.join(root, dirname))
```

Plain-text Plain-text refers to character strings or files that are composed of the characters accessible to a typewriter keyboard. These files typically have a ".txt" suffix to their names. Plain-text files are sometimes referred to as 7-bit ascii files because all of the familiar keyboard characters have

ASCII vales under 128 (i.e., can be designated in binary with, just seven 0s and 1s. In practice, plain-text files exclude 7-bit ascii symbols that do not code for familiar keyboard characters. To further confuse the issue, plain-text files may contain ascii characters above 7 bits (i.e., characters from 128 to 255) that represent characters that are printable on computer monitors, such as accented letters.

Plesionymy Nearly synonymous words, or pairs of words that are sometimes synonymous; other times not. For example, the noun forms of "smell" and "odor" are synonymous. As verb forms, "smell" applies, but odor does not. You can small a fish, but you cannot odor a fish. Smell and odor are plesionyms. Plesionymy is another challenge for machine translators.

Polysemy Occurs when a word has more than one distinct meaning. The intended meaning of a word can sometimes be determined by the context in which the word is used. For example, "She rose to the occasion," and "Her favorite flower is the rose." Sometimes polysemy cannot be resolved. For example, "Eats shoots and leaves."

RegEx Short for Regular Expressions, RegEx is a syntax for describing patterns in text. For example, if I wanted to pull all lines from a text file that began with an uppercase "B" and contained at least one integer, and ended with the a lowercase x, then I might use the regular expression: " B.*[0-9].*x$". This syntax for expressing patterns of strings that can be matched by pre-built methods available to a programming language is somewhat standardized. This means that a RegEx expression in Perl will match the same pattern in Python, or Ruby, or any language that employs RegEx. The relevance of RegEx to Big Data is several-fold. RegEx can be used to build or transform data from one format to another; hence creating or merging data records. It can be used to convert sets of data to a desired format; hence transforming data sets. It can be used to extract records that meet a set of characteristics specified by a user; thus filtering subsets of data or executing data queries over text-based files or text-based indexes. The big drawback to using RegEx is speed: operations that call for many RegEx operations, particularly when those operations are repeated for each parsed line or record, will reduce software performance. RegEx-heavy programs that operate just fine on megabyte files may take hours, days or months to parse through terabytes of data.

A 12-line python script, file_search.py, prompts the user for the name of a text file to be searched, and then prompts the user to supply a RegEx pattern. The script will parse the text file, line by line, displaying those lines that contain a match to the RegEx pattern.

```
import sys, string, re
print("What is file would you like to search?")
filename = sys.stdin.readline()
filename = filename.rstrip()
print("Enter a word, phrase or regular expression to search.")
word_to_search = (sys.stdin.readline()).rstrip()
infile = open (filename, "r")
regex_object = re.compile(word_to_search, re.I)
for line in infile:
  m= regex_object.search(line)
  if m:
     print(line)
```

Scalable Software is scalable if it operates smoothly, whether the data is small or large. Software programs that operate by slurping all data into a RAM variable (i.e., a data holder in RAM memory) are not scalable, because such programs will eventually encounter a quantity of data that is too large to store in RAM. As a rule of thumb, programs that process text at speeds less than a megabyte per second are not scalable, as they cannot cope, in a reasonable time frame, with quantities of data in the gigabyte and higher range.

Script A script is a program that is written in plain-text, in a syntax appropriate for a particular programming language, that needs to be parsed through that language's interpreter before it can be compiled

and executed. Scripts tend to run a bit slower than executable files, but they have the advantage that they can be understood by anyone who is familiar with the script's programming language.

Sentence Computers parse files line by line, not sentence by sentence. If you want a computer to perform operations on a sequence of sentences found in a corpus of text, then you need to include a subroutine in your scripts that list the sequential sentences. One of the simplest ways to find the boundaries of sentences is to look for a period followed by one or more spaces, followed by an uppercase letter. Here's a simple Python demonstration of a sentence extractor, using a few famous lines from the Lewis Carroll poem, Jabberwocky.

```
import re
all_text =\
"And, has thou slain the Jabberwock? Come \
to my arms, my beamish boy! O frabjous \
day! Callooh! Callay! He chortled in his \
joy. Lewis Carroll, excerpted from \
Jabberwocky";
sentence_list = re.split(r'[\.\!\?] +(?=[A-Z])', all_text)
print("\n".join(sentence_list))
```

Here is the output:

```
And, has thou slain the Jabberwock
Come to my arms, my beamish boy
O frabjous day
Callooh
Callay
He chortled in his joy
Lewis Carroll, excerpted from Jabberwocky
```

The meat of the script is the following line of code, which splits lines of text at the boundaries of sentences:

```
sentence_list = re.split(r'[\.\!\?] +(?=[A-Z])',in_text_string)
```

This algorithm is hardly foolproof, as periods are used for many purposes other than as sentence terminators. But it may suffice for most purposes.

Signal In a very loose sense a signal is a way of gauging how measured quantities (e.g., force, voltage, or pressure) change in response to, or along with, other measured quantities (e.g., time). A sound signal is caused by the changes in pressure, exerted on our eardrums, over time. A visual signal is the change in the photons impinging on our retinas, over time. An image is the change in pixel values over a two-dimensional grid. Because much of the data stored in computers consists of discrete quantities of describable objects, and because these discrete quantities change their values, with respect to one another, we can appreciate that a great deal of modern data analysis is reducible to digital signal processing.

Specification A specification is a method for describing objects (physical objects such as nuts and bolts or symbolic objects such as numbers). Specifications do not require specific types of information, and do not impose any order of appearance of the data contained in the document. Specifications do not generally require certification by a standards organization. They are generally produced by special interest organizations, and their legitimacy depends on their popularity. Examples of specifications are RDF (Resource Description Framework) produced by the W3C (WorldWide Web Consortium), and TCP/IP (Transfer Control Protocol/Internet Protocol), maintained by the Internet Engineering Task Force.

String A string is a sequence of characters. Words, phrases, numbers, and alphanumeric sequences (e.g., identifiers, one-way hash values, passwords) are strings. A book is a long string. The complete sequence of the human genome (3 billion characters, with each character an A,T,G, or C) is a very long string. Every subsequence of a string is another string.

Syntax Syntax is the standard form or structure of a statement. What we know as English grammar is equivalent to the syntax for the English language. If I write, "Jules hates pizza," the statement would be syntactically valid, but factually incorrect. If I write, "Jules drives to work in his pizza," the statement would be syntactically valid but nonsensical. For programming languages, syntax refers to the enforced structure of command lines. In the context of triplestores, syntax refers to the arrangement and notation requirements for the three elements of a statement (e.g., RDF format or N3 format). Charles Mead distinctly summarized the difference between syntax and semantics: "Syntax is structure; semantics is meaning" [22].

Systematics The term "systematics" is, by tradition, reserved for the field of biology that deals with taxonomy (i.e., the listing of the distinct types of organisms) and with classification (i.e., the classes of organisms and their relationships to one another). There is no reason why biologists should lay exclusive claim to the field of systematics. As used herein, systematics equals taxonomics plus classification, and this term applies just as strongly to stamp collecting, marketing, operations research, and object-oriented programming as it does to the field of biology.

Taxa Plural of taxon.

Taxon A taxon is a class. The common usage of "taxon" is somewhat inconsistent, as it sometimes refers to the class name, and at other times refers to the instances (i.e., members) of the class. In this book, the term "taxon" is abandoned in favor of "class," the plesionym used by computer scientists. Hence, the term "class" is used herein in the same manner that it is used in modern object oriented programming languages.

Taxonomy When we write of "taxonomy" as an area of study, we refer to the methods and concepts related to the science of classification, derived from the ancient Greek taxis, "arrangement," and nomia, "method." When we write of "a taxonomy," as a construction within a classification, we are referring to the collection of named instances (class members) in the classification. To appreciate the difference between a taxonomy and a classification, it helps to think of taxonomy as the scientific field that determines how different members of a classification are named. Classification is the scientific field that determines how related members are assigned to classes, and how the different classes are related to one another. A taxonomy is similar to a nomenclature; the difference is that in a taxonomy, every named instance must have an assigned class.

Term extraction algorithm Terms are phrases, most often noun phrases, and sometimes individual words, that have a precise meaning within a knowledge domain. For example, "software validation," "RDF triple," and "WorldWide Telescope" are examples of terms that might appear in the index or the glossary of this book. The most useful terms might appear up to a dozen times in the text, but when they occur on every page, their value as a searchable item is diminished; there are just too many instances of the term to be of practical value. Hence, terms are sometimes described as noun phrases that have low-frequency and high information content. Various algorithms are available to extract candidate terms from textual documents. The candidate terms can be examined by a curator who determines whether they should be included in the index created for the document from which they were extracted. The curator may also compare the extracted candidate terms against a standard nomenclature, to determine whether the candidate terms should be added to the nomenclature. For additional discussion, see Section 2.3, "Term Extraction."

Thesaurus A vocabulary that groups together synonymous terms. A thesaurus is very similar to a nomenclature. There are two minor differences. Nomenclatures do not always group terms by synonymy; and nomenclatures are often restricted to a well-defined topic or knowledge domain (e.g., names of stars, infectious diseases, etc.).

Transform (noun form) There are three truly great conceptual breakthroughs that have brought with them great advances to science and to civilization. The first two to be mentioned are well known to everyone: equations and algorithms. Equations permit us to relate variable quantities in a highly specific and repeatable way. Algorithms permit us to follow a series of steps that always produce the same

results. The third conceptual breakthrough, less celebrated but just as important, is the transformation; a way of changing things to yield a something new, with properties that provide an advantage over the original item. In the case of reversible transformation, we can return the transformed item to its original form, and often in improved condition, when we have completed our task.

It should be noted that this definition applies only to the noun form of "transform." The meaning of the verb form of transform is to change or modify, and a transformation is the closest noun form equivalent of the verb form, "to transform."

Uniqueness Uniqueness is the quality of being separable from every other thing in the universe. For data scientists, uniqueness is achieved when data is bound to a unique identifier (i.e., a randomly chosen string of alphanumeric characters) that has not, and will never be, assigned to any data. The binding of data to a permanent and inseparable identifier constitutes the minimal set of ingredients for a data object. Uniqueness can apply to two or more indistinguishable objects, if they are assigned unique identifiers (e.g., unique product numbers stamped into identical auto parts).

Variable In algebra, a variable is a quantity, in an equation, that can change; as opposed to a constant quantity, that cannot change. In computer science, a variable can be perceived as a container that can be assigned a value. If you assign the integer 7 to a container named "x," then "x" equals 7, until you re-assign some other value to the container (i.e., variables are mutable). In most computer languages, when you issue a command assigning a value to a new (undeclared) variable, the variable automatically comes into existence to accept the assignment. The process whereby an object comes into existence, because its existence was implied by an action (such as value assignment), is called reification.

Vocabulary A comprehensive collection of the words used in a general area of knowledge. The term "vocabulary" and the term "nomenclature" are nearly synonymous. In common usage, a vocabulary is a list of words and typically includes a wide range of terms and classes of terms. Nomenclatures typically focus on a class of terms within a vocabulary. For example, a physics vocabulary might contain the terms "quark, black hole, Geiger counter, and Albert Einstein"; a nomenclature might be devoted to the names of celestial bodies.

References

[1] Krauthammer M, Nenadic G. Term identification in the biomedical literature. J Biomed Inform 2004;37:512–26.

[2] Berman JJ. Methods in medical informatics: fundamentals of healthcare programming in Perl, Python, and Ruby. Boca Raton: Chapman and Hall; 2010.

[3] Swanson DR. Undiscovered public knowledge. Libr Q 1986;56:103–18.

[4] Wallis E, Lavell C. Naming the indexer: where credit is due. The Indexer 1995;19:266–8.

[5] Hayes A. VA to apologize for mistaken Lou Gehrig's disease notices. CNN; 2009. August 26. Available from: http://www.cnn.com/2009/POLITICS/08/26/veterans.letters.disease [viewed September 4, 2012].

[6] Hall PA, Lemoine NR. Comparison of manual data coding errors in 2 hospitals. J Clin Pathol 1986;39:622–6.

[7] Berman JJ. Doublet method for very fast autocoding. BMC Med Inform Decis Mak 2004;4:16.

[8] Berman JJ. Nomenclature-based data retrieval without prior annotation: facilitating biomedical data integration with fast doublet matching. In Silico Biol 2005;5:0029.

[9] Burrows M, Wheeler DJ. a block-sorting lossless data compression algorithm. SRC Research Report 124, May 10, 1994.

[10] Berman JJ. Perl programming for medicine and biology. Sudbury, MA: Jones and Bartlett; 2007.

[11] Healy J, Thomas EE, Schwartz JT, Wigler M. Annotating large genomes with exact word matches. Genome Res 2003;13:2306–15.

[12] Berman JJ. Data simplification: taming information with open source tools. Waltham, MA: Morgan Kaufmann; 2016.

[13] Burrows-Wheeler transform. Wikipedia. Available at: https://en.wikipedia.org/wiki/Burrows%E2%80%93Wheeler_transform [viewed August 18, 2015].

[14] Cipra BA. The best of the 20th century: editors name top 10 algorithms. SIAM News May 2000;33(4).

[15] Wu X, Kumar V, Quinlan JR, Ghosh J, Yang Q, Motoda H, et al. Top 10 algorithms in data mining. Knowl Inf Syst 2008;14:1–37.

[16] Patil N, Berno AJ, Hinds DA, Barrett WA, Doshi JM, Hacker CR, et al. Blocks of limited haplotype diversity revealed by high-resolution scanning of human chromosome 21. Science 2001;294:1719–23.

[17] Paskin N. Identifier interoperability: a report on two recent ISO activities. D-Lib Mag 2006;12:1–23.

[18] Berman JJ. Repurposing legacy data: innovative case studies. Waltham, MA: Morgan Kaufmann; 2015.

[19] Berman JJ. Principles of big data: preparing, sharing, and analyzing complex information. Waltham, MA: Morgan Kaufmann; 2013.

[20] Berman JJ. Tumor classification: molecular analysis meets Aristotle. BMC Cancer 2004;4:10.

[21] Berman JJ. Tumor taxonomy for the developmental lineage classification of neoplasms. BMC Cancer 2004;4:88.

[22] Mead CN. Data interchange standards in healthcare IT–computable semantic interoperability: now possible but still difficult, do we really need a better mousetrap? J Healthc Inf Manag 2006;20:71–8.

This page intentionally left blank

3

Identification, Deidentification, and Reidentification

OUTLINE

Section 3.1. What Are Identifiers?

Where is the 'any' key?

Homer Simpson, in response to his computer's instruction to "Press any key"

Let us begin this chapter with a riddle. "Is the number 5 a data object?" If you are like most people, you will answer "yes" because "5" is an integer and therefore it is represents numeric data, and "5" is an object because it exists and is different from all the other numbers. Therefore "5" is a data object. This line of reasoning happens to be completely erroneous. Five is not a data object. As a pure abstraction with nothing binding it to a physical object (e.g., 5 pairs of shoes, 5 umbrellas), it barely qualifies as data.

When we speak of a data object, in computer science, we refer to something that is identified and described. Consider the following statements:

```
<f183136d-3051-4c95-9e32-66844971afc5><name><Baltimore>
<f183136d-3051-4c95-9e32-66844971afc5><class><city>
<f183136d-3051-4c95-9e32-66844971afc5><population><620,961>
```

Without knowing much about data objects (which we will be discussing in detail in Section 6.2), we can start to see that these three statements are providing information about Baltimore. They tell us that Baltimore is a city of population 620,961, and that

Principles and Practice of Big Data. https://doi.org/10.1016/B978-0-12-815609-4.00003-0

Baltimore has been assigned an alphanumeric sequence, "f183136d-3051-4c95-9e32-66844971afc5," to which all our available information about Baltimore has been attached. Peeking ahead into Chapter 6, we can now surmise that a data object consists of a unique alphanumeric sequence (the object identifier) plus the descriptive information associated with the identifier (e.g., name, population number, class). We will see that there are compelling reasons for storing all information contained in Big Data resources within uniquely identified data objects. Consequently, one of the most important tasks for data managers is the creation of a dependable identifier system [1]. In this chapter, we will be focusing our attention on the unique identifier and how it is created and utilized in the realm of Big Data.

Identification issues are often ignored by data managers who are accustomed to working on small data projects. It is worthwhile to list, up front, the most important ideas described in this chapter, many of which are counterintuitive and strange to those whose careers are spent outside the confusing realm of Big Data.

- All Big Data resources can be imagined as identifier systems to which we attach our data.
- Without an adequate identification system, a Big Data resource has no value. In this case, the data within the resource cannot be sensibly analyzed.
- Data deidentification is a process whereby links to the public name of the subject of the record are removed.
- Deidentification should not be confused with the act of stripping a record of an identifier. A deidentified record, like any valid data object, must always have an associated identifier.
- Deidentification should not be confused with data scrubbing. Data scrubbers remove unwanted information from a data record, including information of a personal nature, and any information that is not directly related to the purpose of the data record. [Glossary Data cleaning, Data scrubbing]
- Reidentification is a concept that specifically involves personal and private data records. It involves ascertaining the name of the individual who is associated with a deidentified record. Reidentification is sometimes necessary to verify the contents of a record, or to provide information that is necessary for the well-being of the subject of a deidentified data record. Ethical reidentification always requires approval and oversight.
- Where there is no identification, there can be no deidentification and no reidentification.
- When a deidentified data set contains no unique records (i.e., every record has one or more additional records from which it cannot be distinguished, aside from its assigned identifier sequence), then it becomes impossible to maliciously uncover a deidentified record's public name.

Section 3.2. Difference Between an Identifier and an Identifier System

Many errors, of a truth, consist merely in the application the wrong names of things.
Baruch Spinoza

Data identification is among the most underappreciated and least understood Big Data issue. Measurements, annotations, properties, and classes of information have no informational meaning unless they are attached to an identifier that distinguishes one data object from all other data objects, and that links together all of the information that has been or will be associated with the identified data object. The method of identification and the selection of objects and classes to be identified relates fundamentally to the organizational model of the Big Data resource. If data identification is ignored or implemented improperly, the Big Data resource cannot succeed. [Glossary Annotation]

This chapter will describe, in some detail, the available methods for data identification, and the minimal properties of identified information (including uniqueness, exclusivity, completeness, authenticity, and harmonization). The dire consequences of inadequate identification will be discussed, along with real-world examples. Once data objects have been properly identified, they can be deidentified and, under some circumstances, reidentified. The ability to deidentify data objects confers enormous advantages when issues of confidentiality, privacy, and intellectual property emerge. The ability to reidentify deidentified data objects is required for error detection, error correction, and data validation. [Glossary Deidentification, Re-identification, Privacy versus confidentiality, Intellectual property]

Returning to the title of this section, let us ask ourselves, "What is the difference between an identifier and an identifier system?" To answer, by analogy, it is like the difference between having a $100 dollar bill in your pocket and having a savings account with $100 credited to the account. In the case of the $100 bill, anyone in possession of the bill can use it to purchase items. In the case of the $100 credit, there is a system in place for uniquely assigning the $100 to one individual, until such time as that individual conducts an account transaction that increases or decreases the account value. Likewise, an identifier system creates a permanent environment in which the identifiers are safely stored and used.

Every good information system is, at its heart, an identification system: a way of naming data objects so that they can be retrieved by their name, and a way of distinguishing each object from every other object in the system. If data managers properly identified their data, and did absolutely nothing else, they would be producing a collection of data objects with more informational value than many existing Big Data resources.

The properties of a good identifier system are the following:

– **Completeness**

Every unique object in the big data resource must be assigned an identifier.

– **Uniqueness**

Each identifier is a unique sequence.

– **Exclusivity**

Each identifier is assigned to a unique object, and to no other object.

– **Authenticity**

The objects that receive identification must be verified as the objects that they are intended to be. For example, if a young man walks into a bank and claims to be Richie Rich, then the bank must ensure that he is, in fact, who he says he is.

– **Aggregation**

The Big Data resource must have a mechanism to aggregate all of the data that is properly associated with the identifier (i.e., to bundle all of the data that belongs to the uniquely identified objected). In the case of a bank, this might mean collecting all of the transactions associated with an account holder. In a hospital, this might mean collecting all of the data associated with a patient's identifier: clinic visit reports, medication transactions, surgical procedures, and laboratory results. If the identifier system performs properly, aggregation methods will always collect all of the data associated with an object and will never collect any data that is associated with a different object.

– **Permanence**

The identifiers and the associated data must be permanent. In the case of a hospital system, when the patient returns to the hospital after 30 years of absence, the record system must be able to access his identifier and aggregate his data. When a patient dies, the patient's identifier must not perish.

– **Reconciliation**

There should be a mechanism whereby the data associated with a unique, identified object in one Big Data resource can be merged with the data held in another resource, for the same unique object. This process, which requires comparison, authentication, and merging is known as reconciliation. An example of reconciliation is found in health record portability. When a patient visits a hospital, it may be necessary to transfer her electronic medical record from another hospital. Both hospitals need a way of confirming the identity of the patient and combining the records. [Glossary Electronic medical record]

– **Immutability**

In addition to being permanent (i.e., never destroyed or lost), the identifier must never change (see Chapter 6) [2]. In the event that two Big Data resources are merged, or that legacy data is merged into a Big Data resource, or that individual data objects from two different Big Data resources are merged, a single data object will be assigned two

identifiers; one from each of the merging systems. In this case, the identifiers must be preserved as they are, without modification. The merged data object must be provided with annotative information specifying the origin of each identifier (i.e., clarifying which identifier came from which Big Data resource).

– **Security**

The identifier system is vulnerable to malicious attack. A Big Data resource with an identifier system can be irreversibly corrupted if the identifiers are modified. In the case of human-based identifier systems, stolen identifiers can be used for a variety of malicious activities directed against the individuals whose records are included in the resource.

– **Documentation and Quality Assurance**

A system should be in place to find and correct errors in the identifier system. Protocols must be written for establishing the identifier system, for assigning identifiers, for protecting the system, and for monitoring the system. Every problem and every corrective action taken must be documented and reviewed. Review procedures should determine whether the errors were corrected effectively; and measures should be taken to continually improve the identifier system. All procedures, all actions taken, and all modifications of the system should be thoroughly documented. This is a big job.

– **Centrality**

Whether the information system belongs to a savings bank, an airline, a prison system, or a hospital, identifiers play a central role. You can think of information systems as a scaffold of identifiers to which data is attached. For example, in the case of a hospital information system, the patient identifier is the central key to which every transaction for the patient is attached.

– **Autonomy**

An identifier system has a life of its own, independent of the data contained in the Big Data resource. The identifier system can persist, documenting and organizing existing and future data objects even if all of the data in the Big Data resource were to suddenly vanish (i.e., when all of the data contained in all of the data objects are deleted).

In theory, identifier systems are incredibly easy to implement. Here is exactly how it is done:

1. Generate a unique character sequence, such as UUID, or a long random number. [Glossary UUID, Randomness]
2. Assign the unique character sequence (i.e., identifier) to each new object, at the moment that the object is created. In the case of a hospital a patient chart is created at the moment he or she is registered into the hospital information system. In the case of a bank a customer record is created at the moment that he or she is provided with an

account number. In the case of an object-oriented programming language, such as Ruby, this would be the moment when the "new" method is sent to a class object, instructing the class object to create a class instance. [Glossary Object-oriented programming, Instance]

3. Preserve the identifier number and bind it to the object. In practical terms, this means that whenever the data object accrues new data, the new data is assigned to the identifier number. In the case of a hospital system, this would mean that all of the lab tests, billable clinical transactions, pharmacy orders, and so on, are linked to the patient's unique identifier number, as a service provided by the hospital information system. In the case of a banking system, this would mean that all of the customer's deposits and withdrawals and balances are attached to the customer's unique account number.

Section 3.3. Generating Unique Identifiers

A UUID is 128 bits long, and can guarantee uniqueness across space and time.
P. Leach, M. Mealling and R. Salz [3]

Uniqueness is one of those concepts that everyone intuitively understands; explanations would seem unnecessary. Actually, uniqueness in the computational sciences is a somewhat different concept than uniqueness in the natural world. In computational sciences, uniqueness is achieved when a data object is associated with an unique identifier (i.e., a character string that has not been assigned to any other data object). Most of us, when we think of a data object, are probably thinking of a data record, which may consist of the name of a person followed by a list of feature values (height, weight, and age), or a sample of blood followed by laboratory values (e.g., white blood cell count, red cell count, and hematocrit). For computer scientists a data object is a holder for data values (the so-called encapsulated data), descriptors of the data, and properties of the holder (i.e., the class of objects to which the instance belongs). Uniqueness is achieved when the data object is permanently bound to its own identifier sequence. [Glossary Encapsulation]

Unique objects have three properties:

- A unique object can be distinguished from all other unique objects.
- A unique object cannot be distinguished from itself.
- Uniqueness may apply to collections of objects (i.e., a class of instances can be unique).

UUID (Universally Unique IDentifier) is an example of one type of algorithm that creates unique identifiers, on command, at the moment when new objects are created (i.e., during the run-time of a software application). A UUID is 128 bits long and reserves 60 bits for a string computed directly from a computer time stamp, and is usually represented by a sequence of alphanumeric ASCII characters [3]. UUIDs were originally used in the Apollo

Network Computing System and were later adopted in the Open Software Foundation's Distributed Computing Environment [4]. [Glossary Time stamp, ASCII]

Linux systems have a built-in UUID utility, "uuidgen.exe," that can be called from the system prompt.

Here are a few examples of output values generated by the "uuidgen.exe" utility: [Glossary Command line utility, Utility]

```
$ uuidgen.exe
312e60c9-3d00-4e3f-a013-0d6cb1c9a9fe
$ uuidgen.exe
822df73c-8e54-45b5-9632-e2676d178664
$ uuidgen.exe
8f8633e1-8161-4364-9e98-fdf37205df2f
$ uuidgen.exe
83951b71-1e5e-4c56-bd28-c0c45f52cb8a
$ uuidgen -t
e6325fb6-5c65-11e5-b0e1-0ceee6e0b993
$ uuidgen -r
5d74e36a-4ccb-42f7-9223-84eed03291f9
```

Notice that each of the final two examples has a parameter added to the "uuidgen" command (i.e., "-t" and "-r"). There are several versions of the UUID algorithm that are available. The "-t" parameter instructs the utility to produce a UUID based on the time (measured in seconds elapsed since the first second of October 15, 1582, the start of the Gregorian calendar). The "-r" parameter instructs the utility to produce a UUID based on the generation of a pseudorandom number. In any circumstance, the UUID utility instantly produces a fixed length character string suitable as an object identifier. The UUID utility is trusted and widely used by computer scientists. Independent-minded readers can easily design their own unique object identifiers, using pseudorandom number generators, or with one-way hash generators. [Glossary One-way hash, Pseudorandom number generator]

Python has its own UUID generator. The uuid module is included in the standard python distribution and can be called directly from the script.

```
import uuid
print(uuid.uuid4())
```

When discussing UUIDs the question of duplicates (so-called collisions, in the computer science literature) always arises. How can we be certain that a UUID is unique? Isn't it possible that the algorithm that we use to create a UUID may, at some point, produce the same sequence on more than one occasion? Yes, but the odds are small. It has been estimated that duplicate UUIDs are produced, on average, once every 2.71 quintillion (i.e., 2.71 * 10∧18) executions [5]. It seems that reports of UUID collisions, when investigated, have been attributed to defects in the implementation of the UUID algorithms. The general consensus seems to be that UUID collisions are not worth worrying about, even in the realm of Big Data.

Section 3.4. Really Bad Identifier Methods

I always wanted to be somebody, but now I realize I should have been more specific.
Lily Tomlin

Names are poor identifiers. First off, we can never assume that any name is unique. Surnames such as Smith, Zhang, Garcia, Lo, and given names such as John and Susan are very common. In Korea, five last names account for nearly 50% of the population [6]. Moreover, if we happened to find an individual with a truly unique name (e.g., Mr. Mxyzptlk), there would be no guarantee that some other unique individual might one day have the same name. Compounding the non-uniqueness of names, there is the problem of the many variant forms of a single name. The sources for these variations are many. Here is a partial listing:

1. Modifiers to the surname (du Bois, DuBois, Du Bois, Dubois, Laplace, La Place, van de Wilde, Van DeWilde, etc.).
2. Accents that may or may not be transcribed onto records (e.g., acute accent, cedilla, diacritical comma, palatalized mark, hyphen, diphthong, umlaut, circumflex, and a host of obscure markings).
3. Special typographic characters (the combined "ae").
4. Multiple "middle names" for an individual, that may not be transcribed onto records. Individuals who replace their first name with their middle name for common usage, while retaining the first name for legal documents.
5. Latinized and other versions of a single name (Carl Linnaeus, Carl von Linne, Carolus Linnaeus, Carolus a Linne).
6. Hyphenated names that are confused with first and middle names (e.g., Jean-Jacques Rousseau, or Jean Jacques Rousseau; Louis-Victor-Pierre-Raymond, 7th duc de Broglie, or Louis Victor Pierre Raymond Seventh duc deBroglie).
7. Cultural variations in name order that are mistakenly rearranged when transcribed onto records. Many cultures do not adhere to the Western European name order (e.g., given name, middle name, surname).
8. Name changes; through marriage or other legal actions, aliasing, pseudonymous posing, or insouciant whim.

Aside from the obvious consequences of using names as record identifiers (e.g., corrupt database records, forced merges between incompatible data resources, impossibility of reconciling legacy record), there are non-obvious consequences that are worth considering. Take, for example, accented characters in names. These word decorations wreak havoc on orthography and on alphabetization. Where do you put a name that contains an umlauted character? Do you pretend the umlaut is not there, and alphabetize it according to its plain characters? Do you order based on the ASCII-numeric assignment for the character, in which the umlauted letter may appear nowhere near the plain-lettered words in an alphabetized list. The same problem applies to every special character. [Glossary American Standard Code for Information Interchange, ASCII]

A similar problem exists for surnames with modifiers. Do you alphabetize de Broglie under "D" or under "d" or under "B"? If you choose B, then what do you do with the concatenated form of the name, "deBroglie"? When it comes down to it, it is impossible to satisfactorily alphabetize a list of names. This means that searches based on proximity in the alphabet will always be prone to errors.

I have had numerous conversations with intelligent professionals who are tasked with the responsibility of assigning identifiers to individuals. At some point in every conversation, they will find it necessary to explain that although an individual's name cannot serve as an identifier, the combination of name plus date of birth provides accurate identification in almost every instance. They sometimes get carried away, insisting that the combination of name plus date of birth plus social security number provides perfect identification, as no two people will share all three identifiers: same name, same date of birth, same social security number. This argument rises to the height of folly and completely misses the point of identification. As we will see, it is relatively easy to assign unique identifiers to individuals and to any data object, for that matter. For managers of Big Data resources, the larger problem is ensuring that each unique individual has only one identifier (i.e., denying one object multiple identifiers). [Glossary Social Security Number]

Let us see what happens when we create identifiers from the name plus the birthdate. We will examine name + birthdate + social security number later in this section.

Consider this example. Mary Jessica Meagher, born June 7, 1912 decided to open a separate bank account in each of 10 different banks. Some of the banks had application forms, which she filled out accurately. Other banks registered her account through a teller, who asked her a series of questions and immediately transcribed her answers directly into a computer terminal. Ms. Meagher could not see the computer screen and could not review the entries for accuracy.

Here are the entries for her name plus date of birth:

1. Marie Jessica Meagher, June 7, 1912 (the teller mistook Marie for Mary).
2. Mary J. Meagher, June 7, 1912 (the form requested a middle initial, not name).
3. Mary Jessica Magher, June 7, 1912 (the teller misspelled the surname).
4. Mary Jessica Meagher, Jan 7, 1912 (the birth month was constrained, on the form, to three letters; Jun, entered on the form, was transcribed as Jan).
5. Mary Jessica Meagher, 6/7/12 (the form provided spaces for the final two digits of the birth year. Through a miracle of modern banking, Mary, born in 1912, was re-born a century later).
6. Mary Jessica Meagher, 7/6/2012 (the form asked for day, month, year, in that order, as is common in Europe).
7. Mary Jessica Meagher, June 1, 1912 (on the form, a 7 was mistaken for a 1).
8. Mary Jessie Meagher, June 7, 1912 (Marie, as a child, was called by the informal form of her middle name, which she provided to the teller).
9. Mary Jesse Meagher, June 7, 1912 (Marie, as a child, was called by the informal form of her middle name, which she provided to the teller, and which the teller entered as the male variant of the name).

10. Marie Jesse Mahrer, 1/1/12 (an underzealous clerk combined all of the mistakes on the form and the computer transcript, and added a new orthographic variant of the surname).

For each of these ten examples, a unique individual (Mary Jessica Meagher) would be assigned a different identifier at each of 10 banks. Had Mary re-registered at one bank, ten times, the outcome may have been the same.

If you toss the social security number into the mix (name + birth date + social security number) the problem is compounded. The social security number for an individual is anything but unique. Few of us carry our original social security cards. Our number changes due to false memory ("You mean I've been wrong all these years?"), data entry errors ("Character transpositoins, I mean transpositions, are very common"), intention to deceive ("I don't want to give those people my real number"), or desperation ("I don't have a number, so I'll invent one"), or impersonation ("I don't have health insurance, so I'll use my friend's social security number"). Efforts to reduce errors by requiring patients to produce their social security cards have not been entirely beneficial.

Beginning in the late 1930s, the E. H. Ferree Company, a manufacturer of wallets, promoted their product's card pocket by including a sample social security card with each wallet sold. The display card had the social security number of one of their employees. Many people found it convenient to use the card as their own social security number. Over time, the wallet display number was claimed by over 40,000 people. Today, few institutions require individuals to prove their identity by showing their original social security card. Doing so puts an unreasonable burden on the honest patient (who does not happen to carry his/her card) and provides an advantage to criminals (who can easily forge a card).

Entities that compel individuals to provide a social security number have dubious legal standing. The social security number was originally intended as a device for validating a person's standing in the social security system. More recently, the purpose of the social security number has been expanded to track taxable transactions (i.e., bank accounts, salaries). Other uses of the social security number are not protected by law. The Social Security Act (Section 208 of Title 42 U.S. Code 408) prohibits most entities from compelling anyone to divulge his/her social security number.

Considering the unreliability of social security numbers in most transactional settings, and considering the tenuous legitimacy of requiring individuals to divulge their social security numbers, a prudently designed medical identifier system will limit its reliance on these numbers. The thought of combining the social security number with name and date of birth will virtually guarantee that the identifier system will violate the strict one-to-a-customer rule.

Most identifiers are not purely random numbers; they usually contain some embedded information that can be interpreted by anyone familiar with the identification system. For example, they may embed the first three letters of the individual's family name in the identifier. Likewise, the last two digits of the birth year are commonly embedded in many types of identifiers. Such information is usually included as a crude "honesty" check by people "in the know." For instance, the nine digits of a social security number are divided into an

area code (first three digits), a group number (the next two digits), followed by a serial number (last four digits). People with expertise in the social security numbering system can pry considerable information from a social security number, and can determine whether certain numbers are bogus, based on the presence of excluded sub-sequences.

Seemingly inconsequential information included in an identifier can sometimes be used to discover confidential information about individuals. Here is an example. Suppose every client transaction in a retail store is accessioned under a unique number, consisting of the year of the accession, followed by the consecutive count of accessions, beginning with the first accession of the new year. For example, accession 2010-3518582 might represent the 3,518,582nd purchase transaction in the year 2010. Because each number is unique, and because the number itself says nothing about the purchase, it may be assumed that inspection of the accession number would reveal nothing about the transaction.

Actually, the accession number tells you quite a lot. The prefix (2010) tells you the year of the purchase. If the accession number had been 2010-0000001, then you could safely say that accession represented the first item sold on the first day of business in the year 2010. For any subsequent accession number in 2010, simply divide the suffix number (in this case 3,518,582) by the last accession number of the year, and multiply by 365 (the number of days in a non-leap year), and you have the approximate day of the year that the transaction occurred. This day can easily be converted to a calendar date.

Unimpressed? Consider this scenario. You know that a prominent member of the President's staff had visited a Washington, D.C. Hospital on February 15, 2005, for the purpose of having a liver biopsy. You would like to know the results of that biopsy. You go to a Web site that lists the deidentified pathology records for the hospital, for the years 2000–2010. Though no personal identifiers are included in these public records, the individual records are sorted by accession numbers. Using the aforementioned strategy, you collect all of the surgical biopsies performed on or about February 15, 2010. Of these biopsies, only three are liver biopsies. Of these three biopsies, only one was performed on a person whose gender and age matched the President's staff member. The report provides the diagnosis. You managed to discover some very private information without access to any personal identifiers.

The alphanumeric character string composing the identifier should not expose the patient's identity. For example, a character string consisting of a concatenation of the patient's name, birth date, and social security number might serve to uniquely identify an individual, but it could also be used to steal an individual's identity. The safest identifiers are random character strings containing no information whatsoever.

Section 3.5. Registering Unique Object Identifiers

It isn't that they can't see the solution. It's that they can't see the problem.

G. K. Chesterton

Registries are trusted services that provide unique identifiers to objects. The idea is that everyone using the object will use the identifier provided by the central registry. Unique object registries serve a very important purpose, particularly when the object identifiers

are persistent. It makes sense to have a central authority for Web addresses, library acquisitions, and journal abstracts. Such registries include:

- DOI, Digital object identifier
- PMID, PubMed identification number
- LSID (Life Science Identifier)
- HL7 OID (Health Level 7 Object Identifier)
- DICOM (Digital Imaging and Communications in Medicine) identifiers
- ISSN (International Standard Serial Numbers)
- Social Security Numbers (for United States population)
- NPI, National Provider Identifier, for physicians
- Clinical Trials Protocol Registration System
- Office of Human Research Protections FederalWide Assurance number
- Data Universal Numbering System (DUNS) number
- International Geo Sample Number
- DNS, Domain Name Service
- URL, Unique Resource Locator [Glossary URL]
- URN, Unique Resource Name [Glossary URN]

In some cases the registry does not provide the full identifier for data objects. The registry may provide a general identifier sequence that will apply to every data object in the resource. Individual objects within the resource are provided with a non-unique registry number. A unique suffix sequence is appended locally (i.e., not by a central registrar). Life Science Identifiers (LSIDs) serve as a typical example of a registered identifier. Every LSIDs is composed of the following 5 parts: Network Identifier, root DNS name of the issuing authority, name chosen by the issuing authority, a unique object identifier assigned locally, and an optional revision identifier for versioning information.

In the issued LSID identifier, the parts are separated by a colon, as shown:

urn:lsid:pdb.org:1AFT:1

This identifies the first version of the 1AFT protein in the Protein Data Bank. Here are a few LSIDs:

urn:lsid:ncbi.nlm.nih.gov:pubmed:12571434

This identifies a PubMed citation

urn:lsid:ncbi.nlm.nig.gov:GenBank:T48601:2

This refers to the second version of an entry in GenBank

An OID, short for Object Identifier, is a hierarchy of identifier prefixes. Successive numbers in the prefix identify the descending order of the hierarchy. Here is an example of an OID from HL7, an organization that deals with health data interchanges:

1.3.6.1.4.1.250

Each node is separated from the successor by a dot, Successively finer registration detail leads to the institutional code (the final node). In this case the institution identified by the HL7 OID happens to be the University of Michigan.

The final step in creating an OID for a data object involves placing a unique identifier number at the end of the registered prefix. OID organizations leave the final step to the institutional data managers.

The problem with this approach is that the final within-institution data object identifier is sometimes prepared thoughtlessly, corrupting the OID system [7]. Here is an example. Hospitals use an OID system for identifying images, part of the DICOM (Digital Imaging and Communications in Medicine) image standard. There is a prefix consisting of a permanent, registered code for the institution and the department, and a suffix consisting of a number generated for an image as it is created.

A hospital may assign consecutive numbers to its images, appending these numbers to an OID that is unique for the institution and the department within the institution. For example, the first image created with a CT-scanner might be assigned an identifier consisting of the OID (the assigned code for institution and department) followed by a separator such as a hyphen, followed by "1."

In a worst-case scenario, different instruments may assign consecutive numbers to images, independently of one another. This means that the CT-scanner in room A may be creating the same identifier (OID + image number) as the CT-scanner in Room B; for images on different patients. This problem could be remedied by constraining each CT-scanner to avoid using numbers assigned by any other CT-scanner. This remedy can be defeated if there is a glitch anywhere in the system that accounts for image assignments (e.g., if the counters are re-set, broken, replaced or simply ignored).

When image counting is done properly, and the scanners are constrained to assign unique numbers (not previously assigned by other scanners in the same institution), each image may indeed have a unique identifier (OID prefix + image number suffix). Nonetheless, the use of consecutive numbers for images will create havoc over time. Problems arise when the image service is assigned to another department in the institution, or when departments or institutions merge. Each of these shifts produces a change in the OID (the institutional and departmental prefix) assigned to the identifier. If a consecutive numbering system is used, then you can expect to create duplicate identifiers if institutional prefixes are replaced after the merge. The old records in both of the merging institutions will be assigned the same prefix and will contain replicate (consecutively numbered) suffixes (e.g., image 1, image 2, etc.).

Yet another problem may occur if one unique object is provided with multiple different unique identifiers. A software application may be designed to ignore any previously assigned unique identifier and to generate its own identifier, using its own assignment method. Doing so provides software vendors with a strategy that insulates the vendors from bad identifiers created by their competitor's software, and locks the customer to a vendor's software, and identifiers, forever.

In the end the OID systems provide a good set of identifiers for the institution, but the data objects created within the institution need to have their own identifier systems. Here is the HL7 statement on replicate OIDs:

> *Though HL7 shall exercise diligence before assigning an OID in the HL7 branch to third parties, given the lack of a global OID registry mechanism, one cannot make absolutely certain that there is no preexisting OID assignment for such third-party entity [8].*

It remains to be seen whether any of the registration identifier systems will be used and supported with any serious level of permanence (e.g., over decades and centuries).

Section 3.6. Deidentification and Reidentification

Never answer an anonymous letter.

Yogi Berra

For scientists, deidentification serves two purposes:

- To protect the confidentiality and the privacy of the individual (when the data concerns a particular human subject), and
- To remove information that might bias the experiment (e.g., to blind the experimentalist to patient identities).

Deidentification involves stripping information from a data record that might link the record to the public name of the record's subject. In the case of a patient record, this would involve stripping any information from the record that would enable someone to connect the record to the name of the patient. The most obvious item to be removed in the deidentification process is the patient's name. Other information that should be removed would be the patient's address (which could be linked to the name), the patient's date of birth (which narrows down the set of individuals to whom the data record might pertain), and the patient's social security number. In the United States, patient privacy regulations include a detailed discussion of record deidentification and this discussion recommends 18 patient record items for exclusion from deidentified records [9].

Before going any further, it is important to clarify that deidentification is not achieved by removing an identifier from a data object. In point of fact, nothing good is ever achieved by simply removing an identifier from a data object; doing so simply invalidates the data object (i.e., every data object, identified or deidentified, must have an identifier). Deidentification involves removing information contained in the data object that reveals something about the publicly known name of the data object. This kind of information is often referred to as identifying information, but it would be much less confusing if we used another term for such data, such as "name-linking information." The point here is that we do not want to confuse the identifier of a data object with information contained in a data object that can link the object to its public name.

It may seem counterintuitive, but there is very little difference between an identifier and a deidentifier; under certain conditions the two concepts are equivalent. Here is how a dual identification/deidentification system might work:

1. Collect data on unique object. "Joe Ferguson's bank account contains $100."
2. Assign a unique identifier. "Joe Ferguson's bank account is 7540038947134."
3. Substitute name of object with its assigned unique identifier: "754003894713 contains $100.".

4. Consistently use the identifier with data.
5. Do not let anyone know that Joe Ferguson owns account "754003894713."

The dual use of an identifier/deidentifier is a tried and true technique. Swiss bank accounts are essentially unique numbers (identifiers) assigned to a person. You access the bank account by producing the identifier number. The identifier number does not provide information about the identity of the bank account holder (i.e., it is a deidentifier and an identifier).

The purpose of an identifier is to tell you that whenever the identifier is encountered, it refers to the same unique object, and whenever two different identifiers are encountered, they refer to different objects. The identifier, by itself, should contain no information that links the data object to its public name.

It is important to understand that the process of deidentification can succeed only when each record is properly identified (i.e., there can be no deidentification without identification). Attempts to deidentify a poorly identified data set of clinical information will result in replicative records (multiple records for one patient), mixed-in records (single records composed of information on multiple patients), and missing records (unidentified records lost in the deidentification process).

The process of deidentification is best understood as an algorithm performed on-the-fly, in response to a query from a data analyst. Here is how such an algorithm might proceed.

1. The data analyst submits a query requesting a record from a Big Data resource. The resource contains confidential records that must not be shared, unless the records are deidentified.
2. The Big Data resource receives the query and retrieves the record.
3. A copy of the record is parsed and any of the information within the data record that might link the record to the public name of the subject of the record (usually the name of an individual) is deleted from the copy. This might include the aforementioned name, address, date of birth, and social security number.
4. A pseudo-identifier sequence is prepared for the deidentified record. The pseudo-identifier sequence might be generated by a random number generator, by encrypting the original identifier, through a one-way hash algorithm, or by other methods chosen by the Big Data manager. [Glossary Encryption]
5. A transaction record is attached to the original record that includes the pseudo-identifier, the deidentified record, the time of the transaction, and any information pertaining to the requesting entity (e.g., the data analyst who sent the query) that is deemed fit and necessary by the Big Data resource data manager.
6. A record is sent to the data analyst that consists of the deidentified record (i.e., the record stripped of its true identifier and containing no data that links the record to a named person) and the unique pseudo-identifier created for the record.

Because the deidentified record, and its unique pseudo-identifier are stored with the original record, subsequent requests for the pseudo-identified record can be retrieved and

provided, at the discretion of the Big Data manager. This general approach to data deidentification will apply to requests for a single record or to millions of records.

At this point, you might be asking yourself the following question, "What gives the data manager the right to distribute parts of a confidential record, even if it happens to be deidentified?" You might think that if you tell someone a secret, under the strictest confidence, then you would not want any part of that secret to be shared with anyone else. The whole notion of sharing confidential information that has been deidentified may seem outrageous and unacceptable.

We will discuss the legal and ethical issues of Big Data in Chapters 18 and 19. For now, readers should know that there are several simple and elegant principles that justify sharing deidentified data.

Consider the statement "Jules Berman has a blood glucose level of 85." This would be considered a confidential statement because it tells people something about my medical condition.

Consider the phrase, "Blood glucose 85."

When the name "Jules Berman" is removed, we are left with a disembodied piece of data. "Blood glucose 85" is no different from "Temperature 98.6" or "Apples 2" or "Terminator 3." They are simply raw data belonging to nobody in particular. The act of removing information linking data to a person renders the data harmless. Because the use of properly deidentified data poses no harm to human subjects, United States Regulations allow the unrestricted use of such data for research purposes [9,10]. Other countries have similar provisions.

– **Reidentification**

Because confidentiality and privacy concerns always apply to human subject data, it would seem imperative that deidentification should be an irreversible process (i.e., the names of the subjects and samples should be held a secret, forever).

Scientific integrity does not always accommodate irreversible deidentification. On occasion, experimental samples are mixed-up; samples thought to come from a certain individual, tissue, record, or account, may in fact come from another source. Sometimes major findings in science need to be retracted when a sample mix-up has been shown to occur [11,12,13,14,15]. When samples are submitted, without mix-up, the data is sometimes collected improperly. For example, reversing electrodes on an electrocardiogram may yield spurious and misleading results. Sometimes data is purposefully fabricated and otherwise corrupted, to suit the personal agendas of dishonest scientists. When data errors occur, regardless of reason, it is important to retract the publications [16,17]. To preserve scientific integrity, it is sometimes necessary to discover the identity of deidentified records.

In some cases, deidentification stops the data analyst from helping individuals whose confidentiality is being protected. Imagine you are conducting an analysis on a collection of deidentified data, and you find patients with a genetic marker for a disease that is curable, if treated at an early stage; or you find a new biomarker that determines which

patients would benefit from surgery and which patients would not. You would be compelled to contact the subjects in the database to give them information that could potentially save their lives. Having an irreversibly deidentified data sets precludes any intervention with subjects; nobody knows their identities.

Deidentified records can, under strictly controlled circumstances, be reidentified. Reidentification is typically achieved by entrusting a third party with a confidential list that maps individuals to their deidentified records. Obviously, reidentification can only occur if the Big Data resource keeps a link connecting the identifiers of their data records to the identifiers of the corresponding deidentified record (what we've been calling pseudo-identifiers). The act of assigning a public name to the deidentified record must always involve strict oversight. The data manager must have in place a protocol that describes the process whereby approval for reidentification is obtained. Reidentification provides an opportunity whereby confidentiality can be breached and human subjects can be harmed. Consequently, stewarding the reidentification process is one of the most serious responsibilities of Big Data managers [18].

Section 3.7. Case Study: Data Scrubbing

It is a sin to believe evil of others but it is seldom a mistake.

Garrison Keillor

The term "data scrubbing" is sometimes used, mistakenly, as a synonym for deidentification. It is best to think of data scrubbing as a process that begins where deidentification ends. A data scrubber will remove unwanted information from a data record, including information of a personal nature and any information that is not directly related to the purpose of the data record. For example, in the case of a hospital record a data scrubber might remove the names of physicians who treated the patient; the names of hospitals or medical insurance agencies; addresses; dates; and any textual comments that are inappropriate, incriminating, irrelevant, or potentially damaging. [Glossary Data munging, Data scraping, Data wrangling]

In medical data records, there is a concept known as "minimal necessary" that applies to shared confidential data [9]. It holds that when records are shared, only the minimum necessary information should be released. Any information not directly relevant to the intended purposes of the data analyst should be withheld. The process of data scrubbing gives data managers the opportunity to render a data record that is free of information that would link the record to its subject and free of extraneous information that the data analyst does not actually require. [Glossary Minimal necessary]

There are many methods for data scrubbing. Most of these methods require that data managers develop an exception list of items that should not be included in shared records (e.g., cities, states, zip codes, and names of people). The scrubbing application moves through the records, extracting unnecessary information along the way. The end product is cleaned, but not sterilized. Though many undesired items can be successfully removed,

this approach never produces a perfectly scrubbed set of data. In a Big Data resource, it is simply impossible for the data manager to anticipate every objectionable item and to include it in an exception list. Nobody is that smart.

There is, however, a method whereby data records can be cleaned, without error. This method involves creating a list of data (often in the form of words and phrases) that is acceptable for inclusion in a scrubbed and deidentified data set. Any data that is not in the list of acceptable information is automatically deleted. Whatever is left is the scrubbed data. This method can be described as a reverse scrubbing method. Everything is in the data set is automatically deleted, unless it is an approved "exception."

This method of scrubbing is very fast and can produce an error-free deidentified and scrubbed output [4,19,20]. An example of the kind of output produced by such a scrubber is shown:

*Since the time when * * * * * * * * his own * and the * * * *, the anomalous * * have been * and persistent * * *; and especially * true of the construction and functions of the human *, indeed, it was the anomalous that was * * * in the * the attention, * * that were * to develop into the body * * which we now * *. As by the aid * * * * * * * * * our vision into the * * * has emerged *, we find * * and even evidence of *. To the highest type of * * it is the * the ordinary * * * * *. * to such, no less than to the most *, * * * is of absorbing interest, and it is often * * that the * * the most * into the heart of the mystery of the ordinary. * * been said, * * * * *. * * dermoid cysts, for example, we seem to * * * the secret * of Nature, and * out into the * * of her clumsiness, and * of her * * * *, *, * tell us much of * * * used by the vital * * * * even the silent * * * upon the * * *.*

The reverse-scrubber requires the preexistence of a set of approved terms. One of the simplest methods for generating acceptable terms involves extracting them from a nomenclature that comprehensively covers the terms used in a knowledge domain. For example, a comprehensive listing of living species will not contain dates or zip codes or any of the objectionable language or data that should be excluded from a scrubbed data set. In a method that I have published a list of approved doublets (approximately 200,000 two-word phrases collected from standard nomenclatures) are automatically collected for the scrubbing application [4]. The script is fast, and its speed is not significantly reduced by the size of the list of approved terms.

Here is a short python script. scrub.py, that will take any line of text and produce a scrubbed output. It requires an external file, doublets.txt, containing an approved list of doublet terms.

```
import sys, re, string
doub_file = open("doublets.txt", "r")
doub_hash = {}
for line in doub_file:
  line = line.rstrip()
  doub_hash[line] = " "
```

```
doub_file.close()
print("What would you like to scrub?")
line = sys.stdin.readline()
line = line.lower()
line = line.rstrip()
linearray = re.split(r' +', line)
lastword = "*"
for i in range(0, len(linearray)):
  doublet = " ".join(linearray[i:i+2])
  if doublet in doub_hash:
    print(" " + linearray[i], end="")
    lastword = " " + linearray[i+1]
  else:
    print(lastword, end="")
    lastword = " *"
  if (i == len(linearray) + 1):
    print(lastword, end="")
```

Section 3.8. Case Study (Advanced): Identifiers in Image Headers

Plus ca change, plus c'est la meme chose.

Old French saying ("The more things change, the more things stay the same.")

As it happens, nothing is ever as simple as it ought to be. In the case of an implementation of systems that employ long sequence generators to produce unique identifiers, the most common problem involves indiscriminate reassignment of additional unique identifiers to the same data object, thus nullifying the potential benefits of the unique identifier systems.

Let us look at an example wherein multiple identifiers are redundantly assigned to the same image, corrupting the identifier system. In Section 4.3, we discuss image headers, and we provide examples wherein the ImageMagick "identify" utility could extract the textual information included in the image header. One of the header properties created, inserted, and extracted by ImageMagick's "identify" is an image-specific unique string. [Glossary ImageMagick]

When ImageMagick is installed in our computer, we can extract any image's unique string, using the "identify" utility and the "-format" attribute, on the following system command line: [Glossary Command line]

```
c:\>identify -verbose -format "%#" eqn.jpg
```

Here, the image file we are examining is "eqn.jpg". The "%#" character string is ImageMagick's special syntax indicating that we would like to extract the image identifier from the image header. The output is shown.

```
219e41b4c761e4bb04fbd67f71cc84cd6ae53a26639d4bf33155a5f62ee36e33
```

We can repeat the command line whenever we like, for this image; and the same image-specific unique sequence of characters will be produced.

Using ImageMagick, we can insert text into the "comment" section of the header, using the "-set" attribute. Let us add the text, "I'm modifying myself":

```
c:\ftp>convert eqn.jpg -set comment "I'm modifying myself" eqn.jpg
```

Now, let us extract the comment that we just added, to satisfy ourselves that the "-set" attribute operated as we had hoped. We do this using the "-format" attribute and the "%c" character string, which is ImageMagick's syntax for extracting the comment section of the header.

```
c:\ftp>identify -verbose -format "%c" eqn.jpg
```

The output of the command line is:

```
I'm modifying myself
```

Now, let us run, one more time, the command line that produces the unique character string that is unique for the eqn.jpg image file

```
c:\ftp>identify -verbose -format "%#" eqn.jpg
```

The output is:

```
cb448260d6eeeb2e9f2dcb929fa421b474021584e266d486a6190067a278639f
```

What just happened? Why has the unique character string specific for the eqn.jpg image changed? Has our small modification of the file, which consisted of adding a text comment to the image header, resulted in the production of a new image object, worthy of a new unique identifier?

Before answering these very important questions, let us pose the following gedanken question. Imagine you have a tree. This tree, like every living organism, is unique. It has a unique history, a unique location, and a unique genome (i.e., a unique sequence of nucleotides composing its genetic material). In ten years, its leaves drop off and are replaced ten times. Its trunk expands in size and its height increases. In the ten years of its existence, has the identify of the tree changed? [Glossary Gedanken]

You would probably agree that the tree has changed, but that it has maintained its identity (i.e., it is still the same tree, containing the descendants of the same cells that grew within the younger version of itself).

In informatics, a newly created object is given an identifier, and this identifier is immutable (i.e., cannot be changed), regardless of how the object is modified. In the case of the unique string assigned to an image by ImageMagick, the string serves as an authenticator, not as an identifier. When the image is modified a new unique string is created. By comparing the so-called identifier string in copies of the image file, we can determine whether any modifications have been made. That is to say, we can authenticate the file.

Getting back to the image file in our example, when we modified the image by inserting a text comment, ImageMagick produced a new unique string for the image. The identity of the image had not changed, but the image was different from the original image (i.e., no longer authentic). It seems that the string that we thought to be an identifier string was actually an authenticator string. [Glossary Authentication]

If we want an image to have a unique identifier that does not change when the image is modified, we must create our own identifier that persists when the image is modified.

Here is a short Python script, image_id.py, that uses Python's standard UUID method to create an identifier, which is inserted into the comment section of the image's header, and flanking the identifier with XML tags. [Glossary XML, HTML]

```
import sys, os, uuid
my_id = "<image_id>" + str(uuid.uuid4()) + "</image_id>"
in_command = "convert leaf.jpg -set comment \"" + my_id + "\" leaf.jpg"
os.system(in_command)
out_command = "identify -verbose -format \"%c\" leaf.jpg"
print ("\nHere's the unique identifier:")
os.system(out_command)
print ("\nHere's the unique authenticator:")
os.system("identify -verbose -format \"%#\" leaf.jpg")
os.system("convert leaf.jpg -resize 325x500! leaf.jpg")
print ("\nHere's the new authenticator:")
os.system("identify -verbose -format \"%#\" leaf.jpg")
print ("\nHere's the unique identifier:")
os.system(out_command)
```

Here is the output of the image_id.py script:

```
Here's the unique identifier:
<image_id>b0836a26-8f0e-4a6b-842d-9b0dde2b3f59</image_id>

Here's the unique authenticator:
98c9fe07e90ce43f49961ab6226cd1ccffee648edd1a456a9d06a53ad6d3215a

Here's the new authenticator:
017e401d80a41aafa289ae9c2a1adb7c00477f7a943143141912189499d69ad2

Here's the unique identifier:
<image_id>b0836a26-8f0e-4a6b-842d-9b0dde2b3f59</image_id>
```

What did the script do and what does it teach us? It employed the UUID utility to create a unique and permanent identifier for the image (leaf.jpg, in this case), and inserted the unique identifier into the image header. This identifier, "b0836a26-8f0e-4a6b-842d-9b0dde2b3f59," did not change when the image was subsequently modified. A new authenticator string was automatically inserted into the image header, by ImageMagick, when the image was modified. Hence, we achieved what we needed to achieve: a unique

identifier that never changes, and a unique authenticator that changes when the image is modified in any way.

If you have followed the logic of this section, then you are prepared for the following question posed as an exercise for Zen Buddhists. Imagine you have a hammer. Over the years, you have replaced its head, twice, and its handle, thrice. In this case, with nothing remaining of the original hammer, has it maintained its identity (i.e., is it still the same hammer?). The informatician would answer "Yes," the hammer has maintained its unique identity, but it is no longer authentic (i.e., it is what it must always be, though it has become something different).

Section 3.9. Case Study: One-Way Hashes

I live on a one-way street that's also a dead end. I'm not sure how I got there.
Steven Wright

A one-way hash is an algorithm that transforms a string into another string is such a way that the original string cannot be calculated by operations on the hash value (hence the term "one-way" hash). Popular one-way hash algorithms are MD5 and Standard Hash Algorithm (SHA). A one-way hash value can be calculated for any character string, including a person's name, or a document, or even another one-way hash. For a given input string, the resultant one-way hash will always be the same.

Here are a few examples of one-way hash outputs performed on a sequential list of input strings, followed by their one-way hash (md5 algorithm) output.

```
Jules Berman => Ri0oaVTIAilwnS8+nvKhfA
"Whatever" => n2YtKKG6E4MyEZvUKyGWrw
Whatever => OkXaDVQFYjwkQ+MOC8dpOQ
jules berman => SlnuYpmyn8VXLsxBWwO57Q
Jules J. Berman => i74wZ/CsIbxt3goH2aCS+A
Jules J Berman => yZQfJmAf4dIYO6Bd0qGZ7g
Jules Berman => Ri0oaVTIAilwnS8+nvKhfA
```

The one-way hash values are a seemingly random sequence of ASCII characters (the characters available on a standard keyboard). Notice that a small variation among input strings (e.g., exchanging an uppercase for a lowercase character, adding a period or quotation mark) produces a completely different one-way hash output. The first and the last entry (Jules Berman) yield the same one-way hash output (Ri0oaVTIAilwnS8+nvKhfA) because the two input strings are identical. A given string will always yield the same hash value, so long as the hashing algorithm is not altered. Each one-way hash has the same length (22 characters for this particular md5 algorithm) regardless of the length of the input term. A one-way hash output of the same length (22 characters) could have been produced for a string or file or document of any length. Once produced, there is no feasible mathematical algorithm that can reconstruct the input string from its one-way hash output. In our example, there is no way of examining the string "Ri0oaVTIAilwnS8+nvKhfA" and computing the name Jules Berman.

We see that the key functional difference between a one-way hash and a UUID sequence is that the one-way hash algorithm, performed on a unique string, will always yield the same random-appearing alphanumeric sequence. A UUID algorithm has no input string; it simply produces unique alphanumeric output, and never (almost never) produces the same alphanumeric output twice.

One-way hashes values can serve as ersatz identifiers, permitting Big Data resources to accrue data, over time, to a specific record, even when the record is deidentified (e.g., even when its UUID identifier has been stripped from the record). Here is how it works [18]:

1. A data record is chosen, before it is deidentified, and a one-way hash is performed on its unique identifier string.
2. The record is deidentified by removing the original unique identifier. The output of the one-way hash (from step 1) is substituted for the original unique identifier.
3. The record is deidentified because nobody can reconstruct the original identifier from the one-way hash that has replaced it.
4. The same process is done for every record in the database.
5. All of the data records that were associated with the original identifier will now have the same one-way hash identifier and can be collected under this substitute identifier, which cannot be computationally linked to the original identifier.

Implementation of one-way hashes carry certain practical problems. If anyone happens to have a complete listing of all of the original identifiers, then it would be a simple matter to perform one-way hashes on every listed identifier. This would produce a look-up table that can match deidentified records back to the original identifier, a strategy known as a dictionary attack. For deidentification to work, the original identifier sequences must be kept secret.

One-way hash protocols have many practical uses in the field of information science [21,18,4]. It is very easy to implement one-way hashes, and most programming languages and operating systems come bundled with one or more implementations of one-way hash algorithms. The two most popular one-way hash algorithms are md5 (message digest version 5) and SHA (Secure Hash Algorithm). [Glossary HMAC, Digest, Message digest, Check digit]

Here we use Cygwin's own md5sum.exe utility on the command line to produce a one-way hash for an image file, named dash.png:

```
c:\ftp>c:\cygwin64\bin\md5sum.exe dash.png
```

Here is the output:

```
db50dc33800904ab5f4ac90597d7b4ea *dash.png
```

We could call the same command line from a Python script:

```
import sys, os
os.system("c:/cygwin64/bin/md5sum.exe dash.png")
```

The output will always be the same, as long as the input file, dash.png, does not change:

```
db50dc33800904ab5f4ac90597d7b4ea *dash.png
```

OpenSSL contains several one-way hash implementations, including both md5 and several variants of SHA.

One-way hashes on files are commonly used as a quick and convenient authentication tool. When you download a file from a Web site, you are likely to see that the file distributor has posted the file's one-way hash value. When you receive the file, it is a good idea to calculate the one-way hash on the file that you have received. If the one-way hash value is equal to the posted one-way hash value, then you can be certain that the file received is an exact copy of the file that was intentionally sent. Of course, this does not ensure that the file that was intentionally sent was a legitimate file or that the website was an honest file broker. We will be using our knowledge of one-way hashes when we discuss trusted time stamps (Section 8.5), blockchains (Section 8.6) and data security protocols (Section 18.3).

Glossary

ASCII ASCII is the American Standard Code for Information Interchange, ISO-14962-1997. The ASCII standard is a way of assigning specific 8-bit strings (a string of 0s and 1s of length 8) to the alphanumeric characters and punctuation. Uppercase letters are assigned a different string of 0s and 1s than their matching lowercase letters. There are 256 ways of combining 0s and 1s in strings of length 8. This means that that there are 256 different ASCII characters, and every ASCII character can be assigned a number-equivalent, in the range of 0–255. The familiar keyboard keys produce ASCII characters that happen to occupy ASCII values under 128. Hence, alphanumerics and common punctuation are represented as 8-bits, with the first bit, "0", serving as padding. Hence, keyboard characters are commonly referred to as 7-bit ASCII, and files composed exclusively of common keyboard characters are referred to as plain-text files or as 7-bit ASCII files.
These are the classic ASCII characters:

```
!"#$%&'()*+,-./0123456789:;<=>
?@ABCDEFGHIJKLMNOPQRSTUVWXYZ
[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

Python has several methods for removing non-printable characters from text, including the "printable" method, as shown in this short script, printable.py.

```
# -*- coding: iso-8859-15 -*-

import string
in_string = "prinüéâäàtable"
out_string = "".join(s for s in in_string if s in string.printable)
print(out_strung)
output:
printable
```

It is notable that the first line of code violates a fundamental law of Python programming; that the pound sign signifies that a comment follows, and that the Python interpreter will ignore the pound

sign and any characters that follow the pound sign on the line in which they appear. For obscure reasons, the top line of the snippet is a permitted exception to the rule. In nonpythonic language, the top line conveys to the Python compiler that it may expect to find non-ASCII characters encoded in the iso-8859-15 standard.

The end result of this strange snippet is that non-ASCII characters are stripped from input strings; a handy script worth saving.

American Standard Code for Information Interchange Long form of the familiar acronym, ASCII.

Annotation Annotation involves describing data elements with metadata or attaching supplemental information to data objects.

Authentication A process for determining if the data object that is received (e.g., document, file, image) is the data object that was intended to be received. The simplest authentication protocol involves one-way hash operations on the data that needs to be authenticated. Suppose you happen to know that a certain file, named temp.txt will be arriving via email and that this file has an MD5 hash of "a0869a42609af6c712caeba454f47429". You receive the temp.txt file, and you perform an MD5 one-way hash operation on the file.

In this example, we will use the md5 hash utility bundled into the CygWin distribution (i.e., the Linux emulator for Windows systems). Any md5 implementation would have sufficed.

```
c:\cygwin64\bin>openssl md5 temp.txt
MD5(temp.txt)= a0869a42609af6c712caeba454f47429
```

We see that the md5 hash value generated for the received file is identical to the md5 hash value produced on the file, by the file's creator, before the file was emailed. This tells us that the received, temp.txt, is authentic (i.e., it is the file that you were intended to receive) because no other file has the same MD5 hash. Additional implementations of one-way hashes are described in **Section 3.9.**

The authentication process, in this example, does not tell you who sent the file, the time that the file was created, or anything about the validity of the contents of the file. These would require a protocol that included signature, time stamp, and data validation, in addition to authentication. In common usage, authentication protocols often include entity authentication (i.e., some method by which the entity sending the file is verified). Consequently, authentication protocols are often confused with signature verification protocols. An ancient historical example serves to distinguish the concepts of authentication protocols and signature protocols. Since earliest of recorded history, fingerprints were used as a method of authentication. When a scholar or artisan produced a product, he would press his thumb into the clay tablet, or the pot, or the wax seal closing a document. Anyone doubting the authenticity of the pot could ask the artisan for a thumbprint. If the new thumbprint matched the thumbprint on the tablet, pot, or document, then all knew that the person creating the new thumbprint and the person who had put his thumbprint into the object were the same individual. Hence, ancient pots were authenticated. Of course, this was not proof that the object was the creation of the person with the matching thumbprint. For all anyone knew, there may have been a hundred different pottery artisans, with one person pressing his thumb into every pot produced. You might argue that the thumbprint served as the signature of the artisan. In practical terms, no. The thumbprint, by itself, does not tell you whose print was used. Thumbprints could not be read, at least not in the same way as a written signature. The ancients needed to compare the pot's thumbprint against the thumbprint of the living person who made the print. When the person died, civilization was left with a bunch of pots with the same thumbprint, but without any certain way of knowing whose thumb produced them. In essence, because there was no ancient database that permanently associated thumbprints with individuals, the process of establishing the identity of the pot-maker became very difficult once the artisan died. A good signature protocol permanently binds an authentication code to a unique entity (e.g., a person). Today, we can find a fingerprint at the scene of a crime; we can find a matching signature in a database; and we can link the fingerprint to one individual. Hence, in modern times,

fingerprints are true "digital" signatures, no pun intended. Modern uses of fingerprints include keying (e.g., opening locked devices based on an authenticated fingerprint), tracking (e.g., establishing the path and whereabouts of an individual by following a trail of fingerprints or other identifiers), and body part identification (i.e., identifying the remains of individuals recovered from mass graves or from the sites of catastrophic events based on fingerprint matches). Over the past decade, flaws in the vaunted process of fingerprint identification have been documented, and the improvement of the science of identification is an active area of investigation [22].

Check digit A checksum that produces a single digit as output is referred to as a check digit. Some of the common identification codes in use today, such as ISBN numbers for books, come with a built-in check digit. Of course, when using a single digit as a check value, you can expect that some transmitted errors will escape the check, but the check digit is useful in systems wherein occasional mistakes are tolerated; or wherein the purpose of the check digit is to find a specific type of error (e.g., an error produced by a substitution in a single character or digit), and wherein the check digit itself is rarely transmitted in error.

Command line Instructions to the operating system, that can be directly entered as a line of text from the a system prompt (e.g., the so-called C prompt, "c:\>", in Windows and DOS operating systems; the so-called shell prompt, "$", in Linux-like systems).

Command line utility Programs lacking graphic user interfaces that are executed via command line instructions. The instructions for a utility are typically couched as a series of arguments, on the command line, following the name of the executable file that contains the utility.

Data cleaning More correctly, data cleansing, and synonymous with data fixing or data correcting. Data cleaning is the process by which errors, spurious anomalies, and missing values are somehow handled. The options for data cleaning are: correcting the error, deleting the error, leaving the error unchanged, or imputing a different value [23]. Data cleaning should not be confused with data scrubbing.

Data munging Refers to a multitude of tasks involved in preparing data for some intended purpose (e.g., data cleaning, data scrubbing, and data transformation). Synonymous with data wrangling.

Data scraping Pulling together desired sections of a data set or text by using software.

Data scrubbing A term that is very similar to data deidentification and is sometimes used improperly as a synonym for data deidentification. Data scrubbing refers to the removal of unwanted information from data records. This may include identifiers, private information, or any incriminating or otherwise objectionable language contained in data records, as well as any information deemed irrelevant to the purpose served by the record.

Data wrangling Jargon referring to a multitude of tasks involved in preparing data for eventual analysis. Synonymous with data munging [24].

Deidentification The process of removing all of the links in a data record that can connect the information in the record to an individual. This usually includes the record identifier, demographic information (e.g., place of birth), personal information (e.g., birthdate), and biometrics (e.g., fingerprints). The deidentification strategy will vary based on the type of records examined. Deidentifying protocols exist wherein deidentificated records can be reidentified, when necessary.

Digest As used herein, "digest" is equivalent to a one-way hash algorithm. The word "digest" also refers to the output string produced by a one-way hash algorithm.

Electronic medical record Abbreviated as EMR, or as EHR (Electronic Health Record). The EMR is the digital equivalent of a patient's medical chart. Central to the idea of the EMR is the notion that all of the documents, transactions, and all packets of information containing test results and other information on a patient are linked to the patient's unique identifier. By retrieving all data linked to the patient's identifier, the EMR (i.e., the entire patient's chart) can be assembled instantly.

Encapsulation The concept, from object oriented programming, that a data object contains its associated data. Encapsulation is tightly linked to the concept of introspection, the process of accessing the data

4

Metadata, Semantics, and Triples

OUTLINE

Section 4.1. Metadata

Life is a concept.

Patrick Forterre [1]

When you think about it, numbers are meaningless. The number "8" has no connection to anything in the physical realm until we attach some information to the number (e.g., 8 candles, 8 minutes). Some numbers, like "0" or "−5" have no physical meaning under any set of circumstances. There really is no such thing as "0 dollars"; it is an abstraction indicating the absence of a positive number of dollars. Likewise, there is no such thing as "−5 walnuts"; it is an abstraction that we use to make sense of subtractions ($5-10=-5$).

When we write "8 walnuts," "walnuts" is the metadata that tells us what is being referred to by the data, in this case the number "8."

When we write "8 o'clock", "8" is the data and "o'clock" is the metadata.

Section 4.2. eXtensible Markup Language

The purpose of narrative is to present us with complexity and ambiguity.

Scott Turow

XML (eXtensible Markup Language) is a syntax for attaching descriptors (so-called metadata) to data values. [Glossary Metadata]

In XML, descriptors are commonly known as tags.

Principles and Practice of Big Data. https://doi.org/10.1016/B978-0-12-815609-4.00004-2


image
not
available

A namespace is the metadata realm in which a metadata tag applies. The purpose of a namespace is to distinguish metadata tags that have the same name, but different meaning. For example, within a single XML file, the metadata term "date" may be used to signify a calendar date, or the fruit, or the social engagement. To avoid confusion, the metadata term is given a prefix that is associated with a Web document that defines the term within an assigned Web location. [Glossary Namespace]

For example, an XML page might contain three date-related values, and their metadata descriptors:

```
<calendar:date>June 16, 1904</caldendar:date>
<agriculture:date>Thoory</agriculture:date>
<social:date>Pyramus and Thisbe<social:date>
```

At the top of the XML document you would expect to find declarations for the namespaces used in the XML page. Formal XML namespace declarations have the syntax:

```
xmlns:prefix="URI"
```

In the fictitious example used in this section, the namespace declarations might appear in the "root" tag at the top of the XML page, as shown here (with fake web addresses):

```
<root xmlns:calendar="http://www.calendercollectors.org/"
xmlns:agriculture="http://www.farmersplace.org/"
xmlns:social="http://hearts_throbbing.com/">
```

The namespace URIs are the web locations that define the meanings of the tags that reside within their namespace.

The relevance of namespaces to Big Data resources relates to the heterogeneity of information contained in or linked to a resource. Every description of a value must be provided a unique namespace. With namespaces, a single data object residing in a Big Data resource can be associated with assertions (i.e., object-metadata-data triples) that include descriptors of the same name, without losing the intended sense of the assertions. Furthermore, triples held in different Big Data resources can be merged, with their proper meanings preserved.

Here is an example wherein two resources are merged, with their data arranged as assertion triples.

```
Big Data resource 1

29847575938125      calendar:date      February 4, 1986
83654560466294      calendar:date      June 16, 1904

Big Data resource 2

57839109275632      social:date        Jack and Jill
83654560466294      social:date        Pyramus and Thisbe
```

```
Merged Big Data Resource 1 + 2

29847575938125      calendar:date      February 4, 1986
57839109275632      social:date        Jack and Jill
83654560466294      social:date        Pyramus and Thisbe
83654560466294      calendar:date      June 16, 1904
```

There you have it. The object identified as 83654560466294 is associated with a "date" metadata tag in both resources. When the resources are merged, the unambiguous meaning of the metadata tag is conveyed through the appended namespaces (i.e., social: and calendar:)

Section 4.5. Case Study: A Syntax for Triples

> *I really do not know that anything has ever been more exciting than diagramming sentences.*
>
> *Gertrude Stein*

If you want to represent data as triples, you will need to use a standard grammar and syntax. RDF (Resource Description Framework) is a dialect of XML designed to convey triples. Providing detailed instruction in RDF syntax, or its dialects, lies far outside the scope of this book. However, every Big Data manager must be aware of those features of RDF that enhance the value of Big Data resources. These would include:

1. The ability to express any triple in RDF (i.e., the ability to make RDF statements).
2. The ability to assign the subject of an RDF statement to a unique, identified, and defined class of objects (i.e., that ability to assign the object of a triple to a class).

RDF is a formal syntax for triples. The subjects of triples can be assigned to classes of objects defined in RDF Schemas and linked from documents composed of RDF triples. RDF Schemas will be described in detail in Section 5.9.

When data objects are assigned to classes, the data analysts can discover new relationships among the objects that fall into a class, and can also determine relationships among different related classes (i.e., ancestor classes and descendant classes, also known as superclasses and subclasses). RDF triples plus RDF Schemas provide a semantic structure that supports introspection and reflection. [Glossary Child class, Subclass, RDF Schema, RDFS, Introspection, Reflection]

3. The ability for all data developers to use the same publicly available RDF Schemas and namespace documents with which to describe their data, thus supporting data integration over multiple Big Data resources.

This last feature allows us to turn the Web into a worldwide Big Data resource composed of RDF documents.

We will briefly examine each of these three features in RDF. First, consider the following triple:

```
pubmed:8718907    creator    Bill Moore
```

Every triple consists of an identifier (the subject of the triple), followed by metadata, followed by a value. In RDF syntax the triple is flanked by metadata indicating the beginning and end of the triple. This is the <rdf:description> tag and its end-tag </rdf:description). The identifier is listed as an attribute within the <rdf:description> tag, and is described with the rdf:about tag, indicating the subject of the triple. There follows a metadata descriptor, in this case <author>, enclosing the value, "Bill Moore."

```
<rdf:description rdf:about="urn:pubmed:8718907">
  <creator>Bill Moore</creator>
</rdf:description>
```

The RDF triple tells us that Bill Moore wrote the manuscript identified with the PubMed number 8718907. The PubMed number is the National library of Medicine's unique identifier assigned to a specific journal article. We could express the title of the article in another triple.

```
pubmed:8718907, title, "A prototype Internet autopsy database. 1625
consecutive fetal and neonatal autopsy facesheets spanning 20 years."
```

In RDF, the same triple is expressed as:

```
<rdf:description rdf:about="urn:pubmed:8718907">
  <title>A prototype Internet autopsy database. 1625 consecutive
fetal and neonatal autopsy facesheets spanning 20 years</title>
</rdf:description>
```

RDF permits us to nest triples if they apply to the same unique object.

```
<rdf:description rdf:about="urn:pubmed:8718907">
  <author>Bill Moore</author>
  <title>A prototype Internet autopsy database. 1625 consecutive
fetal and neonatal autopsy facesheets spanning 20 years</title>
</rdf:description>
```

Here we see that the PubMed manuscript identified as 8718907 was written by Bill Moore (the first triple) and is titled "A prototype Internet autopsy database. 1625 consecutive fetal and neonatal autopsy facesheets spanning 20 years" (a second triple).

What do we mean by the metadata tag "title"? How can we be sure that the metadata term "title" refers to the name of a document and does not refer to an honorific (e.g., The Count of Monte Cristo or the Duke of Earl). We append a namespace to the metadata. Namespaces were described in Section 4.4.

```
<rdf:description rdf:about="urn:pubmed:8718907">
  <dc:creator>Bill Moore</dc:creator>
  <dc:title>A prototype Internet autopsy database. 1625 consecutive
fetal and neonatal autopsy facesheets spanning 20 years</dc:title>
</rdf:description>
```

In this case, we appended "dc:" to our metadata. By convention, "dc:" refers to the Dublin Core metadata set at: http://dublincore.org/documents/2012/06/14/dces/.

We will be describing the Dublin Core in more detail, in Section 4.6. [Glossary Dublin Core metadata].

RDF was developed as a semantic framework for the Web. The object identifier system for RDF was created to describe Web addresses or unique resources that are available through the Internet. The identification of unique addresses is done through the use of a Uniform Resource Name (URN) [3]. In many cases the object of a triple designed for the Web will be a Web address. In other cases the URN will be an identifier, such as the PubMed reference number in the example above. In this case, we appended the "urn:" prefix to the PubMed reference in the "about" declaration for the object of the triple.

```
<rdf:description rdf:about="urn:pubmed:8718907">
```

Let us create an RDF triple whose subject is an actual Web address.

```
<rdf:Description rdf:about="http://www.usa.gov/">
  <dc:title>USA.gov: The U.S. Government's Official Web Portal</dc:
title>
</rdf:Description>
```

Here we created a triple wherein the object is uniquely identified by the unique Web address http://www.usa.gov/, and the title of the Web page is "USA.gov: The U.S. Government's Official Web Portal." The RDF syntax for triples was created for the purpose of identifying information with its URI (Unique Resource Identifier). The URI is a string of characters that uniquely identifies a Web resource (such as a unique Web address, or some unique location at a Web address, or some unique piece of information that can be ultimately reached through the Worldwide Web). In theory, using URIs as identifiers for triples will guarantee that all triples will be accessible through the so-called "Semantic Web" (i.e., the Web of meaningful assertions) [3]. Using RDF, Big Data resources can design a scaffold for their information that can be understood by humans, parsed by computers, and shared by other Big Data resources. This solution transforms every RDF-compliant Web page into a an accessible database whose contents can be searched, extracted, aggregated, and integrated along with all the data contained in every existing Big Data resource.

In practice, the RDF syntax is just one of many available formats for packaging triples, and can be used with identifiers that have invalid URIs (i.e., that do not relate in any way to Web addresses or Web resources). The point to remember is that Big Data resources that employ triples can port their data into RDF syntax, or into any other syntax for triples, as needed. [Glossary Notation 3, Turtle]


image
not
available

Index

Note: Page numbers followed by *f* indicate figures.