

ADDISON
WESLEY
DATA &
ANALYTICS
SERIES



PRACTICAL DATA SCIENCE
with
Hadoop[®]
and
Spark

Designing
and Building
Effective Analytics
at Scale

OFER MENDELEVITCH
CASEY STELLA
DOUGLAS EADLINE

FREE SAMPLE CHAPTER

SHARE WITH OTHERS



Practical Data Science with Hadoop[®] and Spark

Practical Data Science with Hadoop[®] and Spark

Designing and Building Effective
Analytics at Scale

Ofer Mendelevitch
Casey Stella
Douglas Eadline

◆ Addison-Wesley

Boston • Columbus • Indianapolis • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City
São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2016955465

Copyright © 2017 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearsoned.com/permissions/.

ISBN-13: 978-0-13-402414-1

ISBN-10: 0-13-402414-1

Contents

Foreword **xiii**

Preface **xv**

Acknowledgments **xxi**

About the Authors **xxiii**

I Data Science with Hadoop—An Overview **1**

1 Introduction to Data Science **3**

What Is Data Science? **3**

Example: Search Advertising **4**

A Bit of Data Science History **5**

 Statistics and Machine Learning **6**

 Innovation from Internet Giants **7**

 Data Science in the Modern Enterprise **8**

Becoming a Data Scientist **8**

 The Data Engineer **8**

 The Applied Scientist **9**

 Transitioning to a Data Scientist Role **9**

 Soft Skills of a Data Scientist **11**

Building a Data Science Team **12**

The Data Science Project Life Cycle **13**

 Ask the Right Question **14**

 Data Acquisition **15**

 Data Cleaning: Taking Care of Data Quality **15**

 Explore the Data and Design Model Features **16**

 Building and Tuning the Model **17**

 Deploy to Production **17**

Managing a Data Science Project **18**

Summary **18**

2 Use Cases for Data Science **19**

Big Data—A Driver of Change **19**

 Volume: More Data Is Now Available **20**

 Variety: More Data Types **20**

 Velocity: Fast Data Ingest **21**

Business Use Cases	21
Product Recommendation	21
Customer Churn Analysis	22
Customer Segmentation	22
Sales Leads Prioritization	23
Sentiment Analysis	24
Fraud Detection	25
Predictive Maintenance	26
Market Basket Analysis	26
Predictive Medical Diagnosis	27
Predicting Patient Re-admission	28
Detecting Anomalous Record Access	28
Insurance Risk Analysis	29
Predicting Oil and Gas Well Production Levels	29
Summary	29
3 Hadoop and Data Science	31
What Is Hadoop?	31
Distributed File System	32
Resource Manager and Scheduler	34
Distributed Data Processing Frameworks	35
Hadoop's Evolution	37
Hadoop Tools for Data Science	38
Apache Sqoop	39
Apache Flume	39
Apache Hive	40
Apache Pig	41
Apache Spark	42
R	44
Python	45
Java Machine Learning Packages	46
Why Hadoop Is Useful to Data Scientists	46
Cost Effective Storage	46
Schema on Read	47
Unstructured and Semi-Structured Data	48
Multi-Language Tooling	48
Robust Scheduling and Resource Management	49
Levels of Distributed Systems Abstractions	49

Scalable Creation of Models	50
Scalable Application of Models	51
Summary	51

II Preparing and Visualizing Data with Hadoop 53

4 Getting Data into Hadoop	55
Hadoop as a Data Lake	56
The Hadoop Distributed File System (HDFS)	58
Direct File Transfer to Hadoop HDFS	58
Importing Data from Files into Hive Tables	59
Import CSV Files into Hive Tables	59
Importing Data into Hive Tables Using Spark	62
Import CSV Files into HIVE Using Spark	63
Import a JSON File into HIVE Using Spark	64
Using Apache Sqoop to Acquire Relational Data	65
Data Import and Export with Sqoop	66
Apache Sqoop Version Changes	67
Using Sqoop V2: A Basic Example	68
Using Apache Flume to Acquire Data Streams	74
Using Flume: A Web Log Example Overview	76
Manage Hadoop Work and Data Flows with Apache Oozie	79
Apache Falcon	81
What's Next in Data Ingestion?	82
Summary	82
5 Data Munging with Hadoop	85
Why Hadoop for Data Munging?	86
Data Quality	86
What Is Data Quality?	86
Dealing with Data Quality Issues	87
Using Hadoop for Data Quality	92
The Feature Matrix	93
Choosing the "Right" Features	94
Sampling: Choosing Instances	94
Generating Features	96
Text Features	97

- Time-Series Features **100**
- Features from Complex Data Types **101**
- Feature Manipulation **102**
- Dimensionality Reduction **103**
- Summary **106**

6 Exploring and Visualizing Data 107

- Why Visualize Data? **107**
 - Motivating Example: Visualizing Network Throughput **108**
 - Visualizing the Breakthrough That Never Happened **110**
- Creating Visualizations **112**
 - Comparison Charts **113**
 - Composition Charts **114**
 - Distribution Charts **117**
 - Relationship Charts **118**
- Using Visualization for Data Science **121**
- Popular Visualization Tools **121**
 - R **121**
 - Python: Matplotlib, Seaborn, and Others **122**
 - SAS **122**
 - Matlab **123**
 - Julia **123**
 - Other Visualization Tools **123**
- Visualizing Big Data with Hadoop **123**
- Summary **124**

III Applying Data Modeling with Hadoop 125

7 Machine Learning with Hadoop 127

- Overview of Machine Learning **127**
- Terminology **128**
- Task Types in Machine Learning **129**
- Big Data and Machine Learning **130**
- Tools for Machine Learning **131**
- The Future of Machine Learning and Artificial Intelligence **132**
- Summary **132**

8	Predictive Modeling	133
	Overview of Predictive Modeling	133
	Classification Versus Regression	134
	Evaluating Predictive Models	136
	Evaluating Classifiers	136
	Evaluating Regression Models	139
	Cross Validation	139
	Supervised Learning Algorithms	140
	Building Big Data Predictive Model Solutions	141
	Model Training	141
	Batch Prediction	143
	Real-Time Prediction	144
	Example: Sentiment Analysis	145
	Tweets Dataset	145
	Data Preparation	145
	Feature Generation	146
	Building a Classifier	149
	Summary	150
9	Clustering	151
	Overview of Clustering	151
	Uses of Clustering	152
	Designing a Similarity Measure	153
	Distance Functions	153
	Similarity Functions	154
	Clustering Algorithms	154
	Example: Clustering Algorithms	155
	<i>k</i> -means Clustering	155
	Latent Dirichlet Allocation	157
	Evaluating the Clusters and Choosing the Number of Clusters	157
	Building Big Data Clustering Solutions	158
	Example: Topic Modeling with Latent Dirichlet Allocation	160
	Feature Generation	160
	Running Latent Dirichlet Allocation	162
	Summary	163

10	Anomaly Detection with Hadoop	165
	Overview	165
	Uses of Anomaly Detection	166
	Types of Anomalies in Data	166
	Approaches to Anomaly Detection	167
	Rules-based Methods	167
	Supervised Learning Methods	168
	Unsupervised Learning Methods	168
	Semi-Supervised Learning Methods	170
	Tuning Anomaly Detection Systems	170
	Building a Big Data Anomaly Detection Solution with Hadoop	171
	Example: Detecting Network Intrusions	172
	Data Ingestion	172
	Building a Classifier	176
	Evaluating Performance	177
	Summary	179
11	Natural Language Processing	181
	Natural Language Processing	181
	Historical Approaches	182
	NLP Use Cases	182
	Text Segmentation	183
	Part-of-Speech Tagging	183
	Named Entity Recognition	184
	Sentiment Analysis	184
	Topic Modeling	184
	Tooling for NLP in Hadoop	184
	Small-Model NLP	184
	Big-Model NLP	186
	Textual Representations	187
	Bag-of-Words	187
	Word2vec	188
	Sentiment Analysis Example	189
	Stanford CoreNLP	189
	Using Spark for Sentiment Analysis	189
	Summary	193

12 Data Science with Hadoop—The Next Frontier	195
Automated Data Discovery	195
Deep Learning	197
Summary	199
A Book Web Page and Code Download	201
B HDFS Quick Start	203
Quick Command Dereference	204
General User HDFS Commands	204
List Files in HDFS	205
Make a Directory in HDFS	206
Copy Files to HDFS	206
Copy Files from HDFS	207
Copy Files within HDFS	207
Delete a File within HDFS	207
Delete a Directory in HDFS	207
Get an HDFS Status Report (Administrators)	207
Perform an FSCK on HDFS (Administrators)	208
C Additional Background on Data Science and Apache Hadoop and Spark	209
General Hadoop/Spark Information	209
Hadoop/Spark Installation Recipes	210
HDFS	210
MapReduce	211
Spark	211
Essential Tools	211
Machine Learning	212
Index	213

This page intentionally left blank

Foreword

Hadoop and data science have been sought after skillsets respectively over the last five years. However, few publications have attempted to bring the two together, teaching data science within the Hadoop context. For practitioners looking for an introduction to data science combined with solving those problems at scale using Hadoop and related tools, this book will prove to be an excellent resource.

The topic of data science is introduced with topics covered including data ingest, munging, feature extraction, machine learning, predictive modeling, anomaly detection, and natural language processing. The platform of choice for the examples and implementation of these topics is Hadoop, Spark, and the other parts of the Hadoop ecosystem. Its coverage is broad, with specific examples keeping the book grounded in an engineer's need to solve real-world problems. For those already familiar with data science, but looking to expand their skillsets to very large datasets and Hadoop, this book is a great introduction.

Throughout the text it focuses on concrete examples and providing insight into business value with each approach. Chapter 5, "Data Munging with Hadoop," provides particularly useful real-world examples on using Hadoop to prepare large datasets for common machine learning and data science tasks. Chapter 10 on anomaly detection is particularly useful for large datasets where monitoring and alerting are important. Chapter 11 on natural language processing will be of interest to those attempting to make chatbots.

Ofer Mendeleevitch is the VP of Data Science at Lendup.com and was previously the Director of Data Science at Hortonworks. Few others are as qualified to be the lead author on a book combining data science and Hadoop. Joining Ofer is his former colleague, Casey Stella, a Principal Data Scientist at Hortonworks. Rounding out these experts in data science and Hadoop is Doug Eadline, frequent contributor to the Addison-Wesley Data & Analytics Series with the titles *Hadoop Fundamentals Live Lessons*, *Apache Hadoop 2 Quick-Start Guide*, and *Apache Hadoop YARN*. Collectively, this team of authors brings over a decade of Hadoop experience. I can imagine few others that have as much knowledge on the subject of data science and Hadoop.

I'm excited to have this addition to the Data & Analytics Series. Creating data science solutions at scale in production systems is an in-demand skillset. This book will help you come up to speed quickly to deploy and run production data science solutions at scale.

—Paul Dix
Series Editor

This page intentionally left blank

Preface

Data science and machine learning are at the core of many innovative technologies and products and are expected to continue to disrupt many industries and business models across the globe for the foreseeable future. Until recently though, most of this innovation was constrained by the limited availability of data.

With the introduction of Apache Hadoop, all of that has changed. Hadoop provides a platform for storing, managing, and processing large datasets inexpensively and at scale, making data science analysis of large datasets practical and feasible. In this new world of large-scale advanced analytics, data science is a core competency that enables organizations to remain competitive and innovate beyond their traditional business models. During our time at Hortonworks, we have had a chance to see how various organizations tackle this new set of opportunities and help them on their journey to implementing data science at scale with Hadoop and Spark. In this book we would like to share some of this learning and experiences.

Another issue we also wish to emphasize is the evolution of Apache Hadoop from its early incarnation as a monolithic MapReduce engine (Hadoop version 1) to a versatile data analytics platform that runs on YARN and supports not only MapReduce but also Tez and Spark as processing engines (Hadoop version 2). The current version of Hadoop provides a robust and efficient platform for many data science applications and opens up a universe of opportunities to new business use cases that were previously unthinkable.

Focus of the Book

This book focuses on real-world practical aspects of data science with Hadoop and Spark. Since the scope of data science is very broad, and every topic therein is deep and complex, it is quite difficult to cover the topic thoroughly. We approached this problem by attempting a good balance between the theoretical coverage of each use case and the example-driven treatment of practical implementation.

This book is not designed to dig deep into many of the mathematical details of each machine learning or statistical approach but rather provide a high-level description of the main concepts along with guidelines for its practical use in the context of the business problem. We provide some references that offer more in-depth treatment of the mathematical details of these techniques in the text and have compiled a list of relevant resources in Appendix C, “Additional Background on Data Science and Apache Hadoop and Spark.”

When learning about Hadoop, access to a Hadoop cluster environment can become an issue. Finding an effective way to “play” with Hadoop and Spark can be challenging

for some individuals. At a minimum, we recommend the Hortonworks virtual machine sandbox for those that would like an easy way to get started with Hadoop. The sandbox is a full single-node Hadoop installation running inside a virtual machine. The virtual machine can be run under Windows, Mac OS, and Linux. Please see <http://hortonworks.com/products/sandbox> for more information on how to download and install the sandbox. For further help with Hadoop we recommend *Hadoop 2 Quick-Start Guide: Learn the Essentials of Big Data Computation in the Apache Hadoop 2 Ecosystem* (and supporting videos), all mentioned in Appendix C.

Who Should Read This Book

This book is intended for those readers who are interested to learn more about what data science is and some of the practical considerations of its application to large-scale datasets. It provides a strong technical foundation for readers who want to learn more about how to implement various use cases, the tools that are best suited for the job, and some of the architectures that are common in these situations. It also provides a business-driven viewpoint on when application of data science to large datasets is useful to help stakeholders understand what value can be derived for their organization and where to invest their resources in applying large-scale machine learning.

There is also a level of experience assumed for this book. For those not versed in data science, some basic competencies are important to have to understand the different methods, including statistical concepts (for example, mean and standard deviation), and a bit of background in programming (mostly Python and a bit of Java or Scala) to understand the examples throughout the book.

For those with a data science background, you should generally be comfortable with the material, although there may be some practical issues such as understanding the numerous Apache projects. In addition, all examples are text-based, and some familiarity with the Linux command line is required. It should be noted that we did not use (or test) a Windows environment for the examples. However, there is no reason to assume they will not work in that and other environments (Hortonworks supports Windows).

In terms of a specific Hadoop environment, all the examples and code were run under Hortonworks HDP Linux Hadoop distribution (either laptop or cluster). Your environment may differ in terms of distribution (Cloudera, MapR, Apache Source) or operating systems (Windows). However, all the tools (or equivalents) are available in both environments.

How to Use This Book

We anticipate several different audiences for the book:

- data scientists
- developers/data engineers
- business stakeholders

While these readers come at the Hadoop analytics from different backgrounds, their goal is certainly the same—running data analytics with Hadoop and Spark at scale. To this end, we have designed the chapters to meet the needs of all readers, and as such readers may find that they can skip areas where they may have a good practical understanding. Finally, we also want to invite novice readers to use this book as a first step in their understanding of data science at scale. We believe there is value in “walking” through the examples, even if you are not sure what is actually happening, and then going back and buttressing your understanding with the background material.

Part I, “Data Science with Hadoop—An Overview,” spans the first three chapters.

Chapter 1, “Introduction to Data Science,” provides an overview of data science and its history and evolution over the years. It lays out the journey people often take to become a data scientist. For those not versed in data science, this chapter will help you understand why it has evolved into a powerful discipline and provide some insight into how a data scientist designs and refines projects. There is also some discussion about what makes a data scientist and how to best plan your career in that direction.

Chapter 2, “Use Cases for Data Science,” provides a good overview of how business use cases are impacted by the volume, variety, and velocity of modern data streams. It also covers some real-world data science use cases in order to help you gain an understanding of its benefits in various industries and applications.

Chapter 3, “Hadoop and Data Science,” provides a quick overview of Hadoop, its evolution over the years, and the various tools in the Hadoop ecosystem. For first-time Hadoop users this chapter can be a bit overwhelming. There are many new concepts introduced including the Hadoop file system (HDFS), MapReduce, the Hadoop resource manager (YARN), and Spark. While the number of sub-projects (and weird names) that make up the Hadoop ecosystem may seem daunting, not every project is used at the same time, and the applications in the later chapters usually focus on only a few tools at a time.

Part II, “Preparing and Visualizing Data with Hadoop,” includes the next three chapters.

Chapter 4, “Getting Data into Hadoop,” focuses on data ingestion, discussing various tools and techniques to import datasets from external sources into Hadoop. It is useful for many subsequent chapters. We begin with describing the Hadoop data lake concept and then move into the various ways data can be used by the Hadoop platform. The ingestion targets two of the more popular Hadoop tools—Hive and Spark. This chapter focuses on code and hands-on solutions—if you are new to Hadoop, its best to also consult Appendix B, “HDFS Quick Start,” to get you up to speed on the HDFS file system.

Chapter 5, “Data Munging with Hadoop,” focuses on data munging with Hadoop or how to identify and handle data quality issues, as well as pre-process data and prepare it for modeling. We introduce the concepts of data completeness, validity, consistency, timeliness, and accuracy. Examples of feature generation using a real data set are provided. This chapter is useful for all types of subsequent analysis and, like Chapter 4, is a precursor to many of the techniques mentioned in later chapters.

An important tool in the process of data munging is visualization. Chapter 6, “Exploring and Visualizing Data,” discusses what it means to do visualization with big data. As background, this chapter is useful for reinforcing some of the basic concepts behind data visualization. The charts presented in the chapter were generated using R. Source code for all the plots is available so readers can try these charts with their own data.

Part III, “Applying Data Modeling with Hadoop,” encompasses the final six chapters.

Chapter 7, “Machine Learning with Hadoop,” provides an overview of machine learning at a high level, covering the main tasks in machine learning such as classification and regression, clustering, and anomaly detection. For each task type, we explore the problem and the main approaches to solutions.

Chapter 8, “Predictive Modeling,” covers the basic algorithms and various Hadoop tools for predictive modeling. The chapter includes an end-to-end example of building a predictive model for sentiment analysis of Twitter text using Hive and Spark.

Chapter 9, “Clustering,” dives into cluster analysis, a very common technique in data science. It provides an overview of various clustering techniques and similarity functions, which are at the core of clustering. It then demonstrates a real-world example of using topic modeling on a large corpus of documents using Hadoop and Spark.

Chapter 10, “Anomaly Detection with Hadoop,” covers anomaly detection, describing various types of approaches and algorithms as well as how to perform large-scale anomaly detection on various datasets. It then demonstrates how to build an anomaly detection system with Spark for the KDD99 dataset.

Chapter 11, “Natural Language Processing,” covers applications of data science to the specific area of human language, using a set of techniques commonly called natural language processing (NLP). It discusses various approaches to NLP, open-source tools that are effective at various NLP tasks, and how to apply NLP to large-scale corpuses using Hadoop, Pig, and Spark. An end-to-end example shows an advanced approach to sentiment analysis that uses NLP at scale with Spark.

Chapter 12, “Data Science with Hadoop—The Next Frontier,” discusses the future of data science with Hadoop, covering advanced data discovery techniques and deep learning.

Consult Appendix A, “Book Webpage and Code Download,” for the book web page and code repository (the web page provides a question and answer forum). Appendix B, as mentioned previously, provides a quick overview of HDFS for new users and the aforementioned Appendix C provides further references and background on Hadoop, Spark, HDFS, machine learning, and many other topics.

Book Conventions

Code and file references are displayed in a monospaced font. Code input lines that wrap because they are too long to fit on one line in this book are denoted with this symbol ➤ at the start of the next line. Long output lines are wrapped at page boundaries without the symbol.

Accompanying Code

Again, please see Appendix A, “Book Web Page and Code Download,” for the location of all code used in this book.

Register your copy of *Practical Data Science with Hadoop® and Spark* at informit.com for convenient access to downloads, updates, and corrections as they become available. To start the registration process, go to informit.com/register and log in or create an account. Enter the product ISBN (9780134024141) and click Submit. Once the process is complete, you will find any available bonus content under “Registered Products.”

This page intentionally left blank

Acknowledgments

Some of the figures and examples were inspired and copied from Yahoo! (yahoo.com), the Apache Software Foundation (<http://www.apache.org>), and Hortonworks (<http://hortonworks.com>). Any copied items either had permission from the author or were available under an open sharing license.

Many people have worked behind the scenes to make this book possible. Thank you to the reviewers who took the time to carefully read the rough drafts: Fabricio Cannini, Brian D. Davison, Mark Fenner, Sylvain Jaume, Joshua Mora, Wendell Smith, and John Wilson.

Ofer Mendelevitch

I want to thank Jeff Needham and Ron Lee who encouraged me to start this book, many others at Hortonworks who helped with constructive feedback and advice, John Wilson who provided great constructive feedback and industry perspective, and of course Debra Williams Cauley for her vision and support in making this book a reality. Last but not least, this book would not have come to life without the loving support of my beautiful wife, Noa, who encouraged and supported me every step of the way, and my boys, Daniel and Jordan, who make all this hard work so worthwhile.

Casey Stella

I want to thank my patient and loving wife, Leah, and children, William and Sylvia, without whom I would not have the time to dedicate to such a time-consuming and rewarding venture. I want to thank my mother and grandmother, who instilled a love of learning that has guided me to this day. I want to thank the taxpayers of the State of Louisiana for providing a college education and access to libraries, public radio, and television; without which I would have neither the capability, the content, nor the courage to speak. Finally, I want to thank Debra Williams Cauley at Addison-Wesley who used the carrot far more than the stick.

Douglas Eadline

To Debra Williams Cauley at Addison-Wesley, your kind efforts and office at the GCT Oyster Bar made the book-writing process almost easy (again!). Thanks to my support crew, Emily, Carla, and Taylor—yet another book you know nothing about. Of course, I cannot forget my office mate, Marlee, and those two boys. And, finally, another big thank you to my wonderful wife, Maddy, for her constant support.

This page intentionally left blank

About the Authors

Ofer Mendelevitch is Vice President of Data Science at Lendup, where he is responsible for Lendup's machine learning and advanced analytics group. Prior to joining Lendup, Ofer was Director of Data Science at Hortonworks, where he was responsible for helping Hortonwork's customers apply Data Science with Hadoop and Spark to big data across various industries including healthcare, finance, retail, and others. Before Hortonworks, Ofer served as Entrepreneur in Residence at XSeed Capital, Vice President of Engineering at Nor1, and Director of Engineering at Yahoo!.

Casey Stella is a Principal Data Scientist at Hortonworks, which provides an open source Hadoop distribution. Casey's primary responsibility is leading the analytics/data science team for the Apache Metron (Incubating) Project, an open source cybersecurity project. Prior to Hortonworks, Casey was an architect at Explorys, which was a medical informatics startup spun out of the Cleveland Clinic. In the more distant past, Casey served as a developer at Oracle, Research Geophysicist at ION Geophysical, and as a poor graduate student in Mathematics at Texas A&M.

Douglas Eadline, PhD, began his career as an analytical chemist with an interest in computer methods. Starting with the first Beowulf how-to document, Doug has written hundreds of articles, white papers, and instructional documents covering many aspects of HPC and Hadoop computing. Prior to starting and editing the popular ClusterMonkey.net website in 2005, he served as editor in chief for *ClusterWorld Magazine* and was senior HPC editor for *Linux Magazine*. He has practical hands-on experience in many aspects of HPC and Apache Hadoop, including hardware and software design, benchmarking, storage, GPU, cloud computing, and parallel computing. Currently, he is a writer and consultant to the HPC/analytics industry and leader of the Limulus Personal Cluster Project (<http://limulus.basement-supercomputing.com>). He is author of the *Hadoop Fundamentals LiveLessons* and *Apache Hadoop YARN Fundamentals LiveLessons* videos from Pearson, and is co-author of *Apache Hadoop YARN: Moving beyond MapReduce and Batch Processing with Apache Hadoop 2* and author of *Hadoop 2 Quick Start Guide: Learn the Essentials of Big Data Computing in the Apache Hadoop 2 Ecosystem*, also from Addison-Wesley, and *High Performance Computing for Dummies*.

This page intentionally left blank



Preparing and Visualizing Data with Hadoop

This page intentionally left blank

Getting Data into Hadoop

*You can have data without information,
but you cannot have information without data.*

Daniel Keys Moran

In This Chapter:

- The data lake concept is presented as a new data processing paradigm.
- Basic methods for importing CSV data into HDFS and Hive tables are presented.
- Additional methods for using Spark to import data into Hive tables or directly for a Spark job are presented.
- Apache Sqoop is introduced as a tool for exporting and importing relational data into and out of HDFS.
- Apache Flume is introduced as a tool for transporting and capturing streaming data (e.g., web logs) into HDFS.
- Apache Oozie is introduced as workflow manager for Hadoop ingestion jobs.
- The Apache Falcon project is described as a framework for data governance (organization) on Hadoop clusters.

No matter what kind of data needs processing, there is often a tool for importing such data from or exporting such data into the Hadoop Distributed File System (HDFS). Once stored in HDFS the data may be processed by any number of tools available in the Hadoop ecosystem.

This chapter begins with the concept of the Hadoop data lake and then follows with a general overview of each of the main tools for data ingestion into Hadoop—Spark, Sqoop, and Flume—along with some specific usage examples. Workflow tools such as Oozie and Falcon are presented as tools that aid in managing the ingestion process.

Hadoop as a Data Lake

Data is ubiquitous, but that does not always mean that it's easy to store and access. In fact, many existing pre-Hadoop data architectures tend to be rather strict and therefore difficult to work with and make changes to. The data lake concept changes all that.

So what is a data lake?

With the more traditional database or data warehouse approach, adding data to the database requires data to be transformed into a *pre-determined* schema before it can be loaded into the database. This step is often called “extract, transform, and load” (ETL) and often consumes a lot of time, effort, and expense before the data can be used for downstream applications. More importantly, decisions about how the data will be used must be made during the ETL step, and later changes are costly. In addition, data are often discarded in the ETL step because they do not fit into the data schema or are deemed un-needed or not valuable for downstream applications.

One of the basic features of Hadoop is a central storage space for all data in the Hadoop Distributed File Systems (HDFS), which make possible inexpensive and redundant storage of large datasets at a much lower cost than traditional systems.

This enables the Hadoop data lake approach, wherein all data are often stored in raw format, and what looks like the ETL step is performed when the data are processed by Hadoop applications. This approach, also known as schema on read, enables programmers and users to enforce a structure to suit their needs when they access data. The traditional data warehouse approach, also known as schema on write, requires more upfront design and assumptions about how the data will eventually be used.

For data science purposes, the capability to keep all the data in raw format is extremely beneficial since often it is not clear up front which data items may be valuable to a given data science goal.

With respect to big data, the data lake offers three advantages over a more traditional approach:

- All data are **available**. There is no need to make any assumptions about future data use.
- All data are **sharable**. Multiple business units or researchers can use all available data¹, some of which may not have been previously available due to data compartmentalization on disparate systems.
- All **access methods** are available. Any processing engine (MapReduce, Tez, Spark) or application (Hive, Spark-SQL, Pig) can be used to examine the data and process it as needed.

1. The capability to use all available data is, of course, governed, as you might expect, by the appropriate security policy with Hadoop tools such as Apache Ranger. The point here is that there is no technical hurdle to data sharing, as is often the case with traditional data architectures.

To be clear, data warehouses are valuable business tools, and Hadoop is designed to complement them, not replace them. Nonetheless, the traditional data warehouse technology was developed before the data lake began to fill with such large quantities of data. The growth of new data from disparate sources including social media, click streams, sensor data, and others is such that we are starting to quickly fill the data lake. Traditional ETL stages may not be able to keep up with the rate at which data are entering the lake. There will be overlap, and each tool will address the need for which it was designed.

The difference between a traditional data warehouse and Hadoop is depicted in Figure 4.1.

Different data sources (A, B, C) can be seen entering either an ETL process or a data lake. The ETL process places the data in a schema as it stores (writes) the data to the relational database. The data lake stores the data in raw form. When a Hadoop application

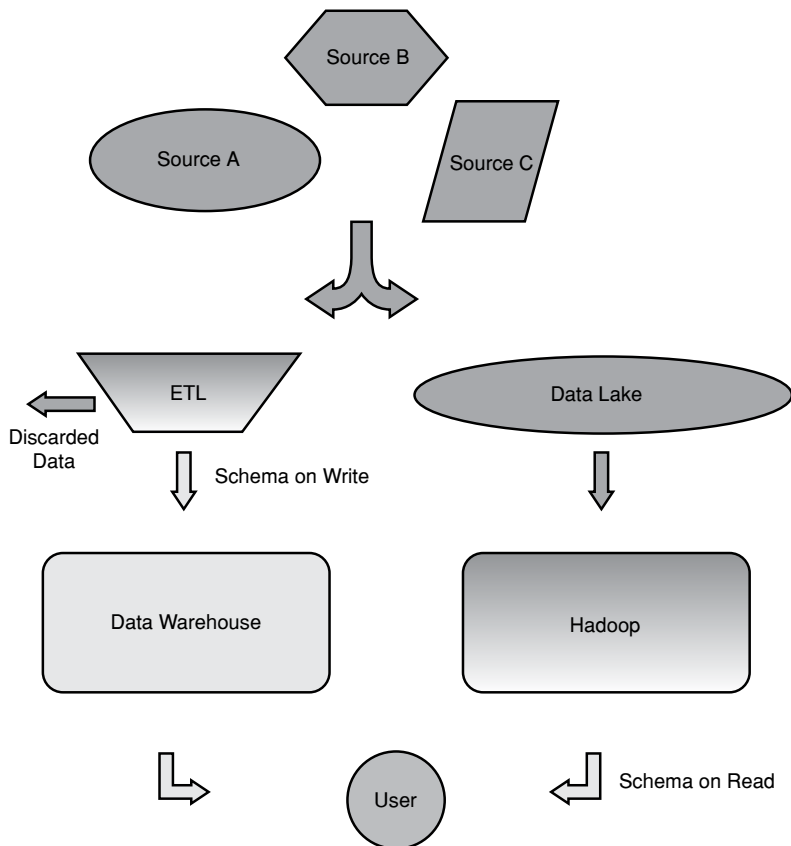


Figure 4.1 The data warehouse versus the Hadoop data lake.

uses the data, the schema is applied to data as they are read from the lake. Note that the ETL step often discards some data as part of the process. In both cases the user accesses the data they need. However, in the Hadoop case it can happen as soon as the data are available in the lake.

The Hadoop Distributed File System (HDFS)

Virtually all Hadoop applications operate on data that are stored in HDFS. The operation of HDFS is separate from the local file system that most users are accustomed to using. That is, the user must explicitly copy to and from the HDFS file system. HDFS is not a general file system and as such cannot be used as a substitute for existing POSIX (or even POSIX-like) file systems.

In general, HDFS is a specialized streaming file system that is optimized for reading and writing of large files. When writing to HDFS, data are “sliced” and replicated across the servers in a Hadoop cluster. The slicing process creates many small sub-units (blocks) of the larger file and *transparently* writes them to the cluster nodes. The various slices can be processed in parallel (at the same time) enabling faster computation. The user does not see the file slices but interacts with whole files in HDFS like a normal file system (i.e., files can be moved, copied, deleted, etc.). When transferring files out of HDFS, the slices are assembled and written as one file on the host file system.

The slices or sub-units are also replicated across different servers so that the failure of any single server will not result in lost data. Due to its design, HDFS does not support random reads or writes to files but does support appending a file. Note that for testing purposes it is also possible to create a single instance of HDFS on a single hard drive (i.e., a laptop or desktop computer), and in this situation there is no file slicing or replication performed on the file.

Direct File Transfer to Hadoop HDFS

The easiest way to move data into and out of HDFS is to use the native HDFS commands. These commands are wrappers that interact with the HDFS file system. Local commands, such as `cp`, `ls`, or `mv` will only work on local files. To copy a file (`test`) from your local file system to HDFS, the following `put` command can be used:

```
$ hdfs dfs -put test
```

To view files in HDFS use the following command. The result is a full listing similar to a locally executed `ls -l` command:

```
$ hdfs dfs -ls
-rw-r--r--  2 username hdfs      497 2016-05-11 14:32 test
```

To copy a file (`another-test`) from HDFS to your local file system, use the following `get` command:

```
$ hdfs dfs -get another-test
```

Other HDFS commands will be introduced in the examples. Appendix B “HDFS Quick Start,” provides basic command examples including listing, copying, and removing files in HDFS.

Importing Data from Files into Hive Tables

Apache Hive is an SQL-like tool for analyzing data in HDFS. Data scientists often want to import data into Hive from existing text-based files exported from spreadsheets or databases. These file formats often include tab-separated values (TSV), comma-separated values (CSV), raw text, JSON, and others. Having the data in Hive tables enables easy access to it for subsequent modeling steps, the most common of which is feature generation, which we discuss in Chapter 5, “Data Munging with Hadoop.”

Once data are imported and present as a Hive table, it is available for processing using a variety of tools including Hive’s SQL query processing, Pig, or Spark.

Hive supports two types of tables. The first type of table is an *internal table* and is fully managed by Hive. If you delete an internal table, both the definition in Hive *and* the data will be deleted. Internal tables are stored in an optimized format such as ORC and thus provide a performance benefit. The second type of table is an *external table* that is not managed by Hive. External tables use only a metadata description to access the data in its raw form. If you delete an external table, only the definition (metadata about the table) in Hive is deleted and the actual data remain intact. External tables are often used when the data resides outside of Hive (i.e., some other application is also using/creating/managing the files), or the original data need to remain in the underlying location even after the table is deleted.

Due to the large number of use cases, we do not cover all the input methods available to Hive, and instead just a basic example of CSV file import is described. Interested readers can consult the Hive project page, <https://hive.apache.org>, for more information.

Import CSV Files into Hive Tables

The following example illustrates how a comma delimited text file (CSV file) can be imported into a Hive table. The input file (`names.csv`) has five fields (Employee ID, First Name, Title, State, and type of Laptop). The first five lines of the file are as follows:

```
10,Andrew,Manager,DE,PC
11,Arun,Manager,NJ,PC
12,Harish,Sales,NJ,MAC
13,Robert,Manager,PA,MAC
14,Laura,Engineer,PA,MAC
```

The first input step is to create a directory in HDFS to hold the file. Note that, like most Hadoop tools, Hive input is directory-based. That is, input for an operation is taken as all files in a given directory. The following command creates a `names` directory in the `users` HDFS directory.

```
$ hdfs dfs -mkdir names
```

In this example, one file is used. However, any number of files could be placed in the input directory. Next the `names.csv` file is moved into the HDFS `names` directory.

```
$ hdfs dfs -put name.csv names
```

Once the file is in HDFS, we first load the data as an external Hive table. Start a Hive shell by typing `hive` at the command prompt and enter the following commands. Note, to cut down on clutter, some of the non-essential Hive output (run times, progress bars, etc.) have been removed from the Hive output.

```
hive> CREATE EXTERNAL TABLE IF NOT EXISTS Names_text(
  > EmployeeID INT,FirstName STRING, Title STRING,
  > State STRING, Laptop STRING)
  > COMMENT 'Employee Names'
  > ROW FORMAT DELIMITED
  > FIELDS TERMINATED BY ','
  > STORED AS TEXTFILE
  > LOCATION '/user/username/names';
OK
```

If the command worked, an `OK` will be printed. The various fields and the comma delimiter are declared in the command. The final `LOCATION` statement in the command tells Hive where to find the input files. The import can be verified by listing the first five rows in the table:

```
hive> Select * from Names_text limit 5;
OK
10      Andrew  Manager DE      PC
11      Arun     Manager NJ      PC
12      Harish   Sales   NJ      MAC
13      Robert   Manager PA     MAC
14      Laura    Engineer PA     MAC
```

The next step is to move the external table to an internal Hive table. The internal table must be created using a similar command. However, the `STORED AS` format offers new options. There are four main file formats for Hive tables in addition to the basic text format. The choice of format depends on the type of data and analysis, but in most cases either ORC or Parquet are used as they provide the best compression and speed advantages for most data types.

- Text file—All data are stored as raw text using the Unicode standard.
- Sequence file—The data are stored as binary key/value pairs.
- RCFile—All data are stored in a column optimized format (instead of row optimized).
- ORC—An optimized row columnar format that can significantly improve Hive performance.
- Parquet—A columnar format that provides portability to other Hadoop tools including Hive, Drill, Impala, Crunch, and Pig.

The following command creates an internal Hive table that uses the ORC format:

```
hive> CREATE TABLE IF NOT EXISTS Names(  
  > EmployeeID INT,FirstName STRING, Title STRING,  
  > State STRING, Laptop STRING)  
  > COMMENT 'Employee Names'  
  > STORED AS ORC;  
OK
```

To create a table using one of the other formats, change the `STORED AS` command to reflect the new format. Once the table is created, the data from the external table can be moved to the internal table using the command,

```
hive> INSERT OVERWRITE TABLE Names SELECT * FROM Names_text;
```

As with the external table, the contents can be verified using the following command:

```
hive> Select * from Names limit 5;  
OK  
10      Andrew  Manager DE      PC  
11      Arun    Manager NJ      PC  
12      Harish   Sales   NJ      MAC  
13      Robert  Manager PA      MAC  
14      Laura   Engineer PA     MAC
```

Hive also supports partitions. With partitions, tables can be separated into logical parts that make it more efficient to query a portion of the data. For example, the internal Hive table created previously can also be created with a partition based on the state field. The following command creates a partitioned table:

```
hive> CREATE TABLE IF NOT EXISTS Names_part(  
  > EmployeeID INT,  
  > FirstName STRING,  
  > Title STRING,  
  > Laptop STRING)  
  > COMMENT 'Employee names partitioned by state'  
  > PARTITIONED BY (State STRING)  
  > STORED AS ORC;  
OK
```

To fill the internal table from the external table for those employed from PA, the following command can be used:

```
hive> INSERT INTO TABLE Names_part PARTITION(state='PA')  
  > SELECT EmployeeID, FirstName, Title, Laptop FROM Names_text WHERE  
  > state='PA';  
...  
OK
```

This method requires each partition key to be selected and loaded individually. When the number of potential partitions is large, this can make data entry inconvenient. To address this issue Hive now supports **dynamic-partition insert** (or multi-partition insert) that is designed to solve this problem by dynamically determining which partitions should be created and populated while scanning the input table.

Importing Data into Hive Tables Using Spark

Apache Spark is a modern processing engine that is focused on in-memory processing. Spark's primary data abstraction is an immutable distributed collection of items called a resilient distributed dataset (RDD). RDDs can be created from Hadoop input formats (such as HDFS files) or by transforming other RDDs. Each dataset in an RDD is divided into logical partitions, which may be transparently computed on different nodes of the cluster.

The other important data abstraction is Spark's DataFrame. A DataFrame is built on top of an RDD, but data are organized into named columns similar to a relational database table and similar to a data frame in R or in Python's Pandas package.

Spark DataFrames can be created from different data sources such as the following:

- Existing RDDs
- Structured data files
- JSON datasets
- Hive tables
- External databases

Due to its flexibility and friendly developer API, Spark is often used as part of the process of ingesting data into Hadoop. With Spark, you can read data from a CSV file, external SQL or NO-SQL data store, or another data source, apply certain transformations to the data, and store it onto Hadoop in HDFS or Hive. Similar to the Hive examples, a full treatment of all Spark import scenarios is beyond the scope of this book. Consult the Apache Spark project page, <http://spark.apache.org>, for more information.

The following sections provide some basic usage examples of data import using PySpark (Spark via the Python API), although these steps can also be performed using the Scala or Java interfaces to Spark. Each step is explained. However, a full description of the Spark commands and API are beyond the scope of this book.

All the examples assume the PySpark shell (version 1.6) has been started using the following command:

```
$ pyspark
Welcome to

  ____
 /  __ \   ___  /  ___/  /  ___/
 \    /  /  _ \  /  /    /  /  ___/
 /___/  /  ___/ \___/  /___/  /___/
 /___/

 version 1.6.2
```

```
Using Python version 2.7.9 (default, Apr 14 2015 12:54:25)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
```

Import CSV Files into HIVE Using Spark

Comma-separated value (CSV) files and, by extension, other text files with separators can be imported into a Spark DataFrame and then stored as a HIVE table using the steps described. Note that in this example we show how to use an RDD, translate it into a DataFrame, and store it in HIVE. It is also possible to load CSV files directly into DataFrames using the `spark-csv` package.

1. The first step imports functions necessary for Spark DataFrame operations:

```
>>> from pyspark.sql import HiveContext
>>> from pyspark.sql.types import *
>>> from pyspark.sql import Row
```

2. Next, the raw data are imported into a Spark RDD. The input file, `names.csv`, is located in the users local file system and does not have to be moved into HDFS prior to use. (Assuming the local path to the data is `/home/username`.)

```
>>> csv_data = sc.textFile("file:///home/username/names.csv")
```

3. The RDD can be confirmed by using the `type()` command:

```
>>> type(csv_data)
<class 'pyspark.rdd.RDD'>
```

4. The comma-separated data are then split using Spark's `map()` function that creates a new RDD:

```
>>> csv_data = csv_data.map(lambda p: p.split(","))
```

Most CSV files have a header with the column names. The following steps remove this from the RDD,

```
>>> header = csv_data.first()
>>> csv_data = csv_data.filter(lambda p:p != header)
```

5. The data in the `csv_data` RDD are put into a Spark SQL DataFrame using the `toDF()` function. First, however, the data are mapped using the `map()` function so that every RDD item becomes a `Row` object which represents a row in the new DataFrame. Note the use of the `int()` to cast for the employee ID as an integer. All other columns default to a string type.

```
>>> df_csv = csv_data.map(lambda p: Row(EmployeeID = int(p[0]),
↳ FirstName = p[1], Title=p[2], State=p[3], Laptop=p[4])).toDF()
```

The `Row()` class captures the mapping of the single values into named columns in a row and subsequently transforms the complete data into a DataFrame.

6. The structure and data of the first five rows of the `df_csv` DataFrame are viewed using the following command:

```
>>> df_csv.show(5)
+-----+-----+-----+-----+-----+
|EmployeeID|FirstName|Laptop|State|  Title|
+-----+-----+-----+-----+-----+
|         10|   Andrew|    PC|   DE| Manager|
|         11|    Arun|    PC|   NJ| Manager|
|         12|   Harish|   MAC|   NJ|   Sales|
|         13|   Robert|   MAC|   PA| Manager|
|         14|    Laura|   MAC|   PA| Engineer|
+-----+-----+-----+-----+-----+
only showing top 5 rows
```

7. Similarly, if you'd like to inspect the DataFrame schema, use the `printSchema()` command:

```
>>> df_csv.printSchema()
root
 |-- EmployeeID: long (nullable = true)
 |-- FirstName: string (nullable = true)
 |-- Laptop: string (nullable = true)
 |-- State: string (nullable = true)
 |-- Title: string (nullable = true)
```

8. Finally, to store the DataFrame into a Hive table, use `saveAsTable()`:

```
>>> from pyspark.sql import HiveContext
>>> hc = HiveContext(sc)
>>> df_csv.write.format("orc").saveAsTable("employees")
```

Here we create a `HiveContext` that is used to store the DataFrame into a Hive table (in ORC format), by using the `saveAsTable()` command.

Import a JSON File into HIVE Using Spark

Spark can import JSON files directly into a DataFrame. The following is a JSON formatted version of the `names.csv` file used in the previous examples. Note that by entering the `EmployeeID` as an unquoted integer, it will be input as an integer.

```
{ "EmployeeID":10, "FirstName": "Andrew", "Title": "Manager", "State": "DE",
  ➤ "Laptop": "PC" }
{ "EmployeeID":11, "FirstName": "Arun", "Title": "Manager", "State": "NJ",
  ➤ "Laptop": "PC" }
{ "EmployeeID":12, "FirstName": "Harish", "Title": "Sales", "State": "NJ",
  ➤ "Laptop": "MAC" }
```

Also note that Spark expects each line to be a separate JSON object, so it will fail if you try to load a fully formatted JSON file.

1. The first step imports the needed functions and creates a HiveContext.

```
>>> from pyspark.sql import HiveContext
>>> hc = HiveContext(sc)
```

Similar to the CSV example, the data file is located in the users local file system.

```
>>> df_json = hc.read.json("file:///home/username/names.json")
```

2. The first five rows of the DataFrame can be viewed using the `df_json.show(5)` command:

```
>>> df_json.show(5)
+-----+-----+-----+-----+-----+
|EmployeeID|FirstName|Laptop|State| Title|
+-----+-----+-----+-----+-----+
|          10| Andrew| PC| DE| Manager|
|          11|  Arun| PC| NJ| Manager|
|          12| Harish| MAC| NJ| Sales|
|          13| Robert| MAC| PA| Manager|
|          14|  Laura| MAC| PA| Engineer|
+-----+-----+-----+-----+-----+
only showing top 5 rows
```

3. To confirm that the `EmployeeID` was indeed cast as an integer, the `df_json.printSchema()` command can be used to inspect the DataFrame schema:

```
>>> df_json.printSchema()

root
 |-- EmployeeID: long (nullable = true)
 |-- FirstName: string (nullable = true)
 |-- Laptop: string (nullable = true)
 |-- State: string (nullable = true)
 |-- Title: string (nullable = true)
```

4. Similar to the CSV example, storing this DataFrame back to Hive is simple:

```
>>> df_json.write.format("orc").saveAsTable("employees")
```

Using Apache Sqoop to Acquire Relational Data

In many enterprise environments, a lot of data that is required for data science applications resides inside of database management systems such as Oracle, MySQL, PostgreSQL, or DB2. Before we can use this data in the context of a data science application, we need to ingest such data into Hadoop.

Sqoop is a tool designed to transfer data between Hadoop and relational databases. You can use Sqoop to import data from a relational database management system (RDBMS) into the Hadoop Distributed File System (HDFS) or export data from Hadoop back into an RDBMS.

Sqoop can be used with any JDBC-compliant database and has been tested on Microsoft SQL Server, PostgreSQL, MySQL, and Oracle. In the remainder of this section, a brief overview of how Sqoop works with Hadoop is provided. In addition, a basic Sqoop example walk-through is demonstrated. To fully explore Sqoop, more information can be found by consulting the Sqoop project website at <http://sqoop.apache.org>.

Data Import and Export with Sqoop

Figure 4.2 describes the process of importing data into HDFS using Sqoop, which includes two steps. In the first step, Sqoop examines the database to gather the necessary metadata for the data that are to be imported. The second step is a map-only² (no reduce step) Hadoop job that Sqoop submits to the cluster. This is the job that does the actual data transfer using the metadata captured in the previous step. Note that each node doing the import must have access to the database.

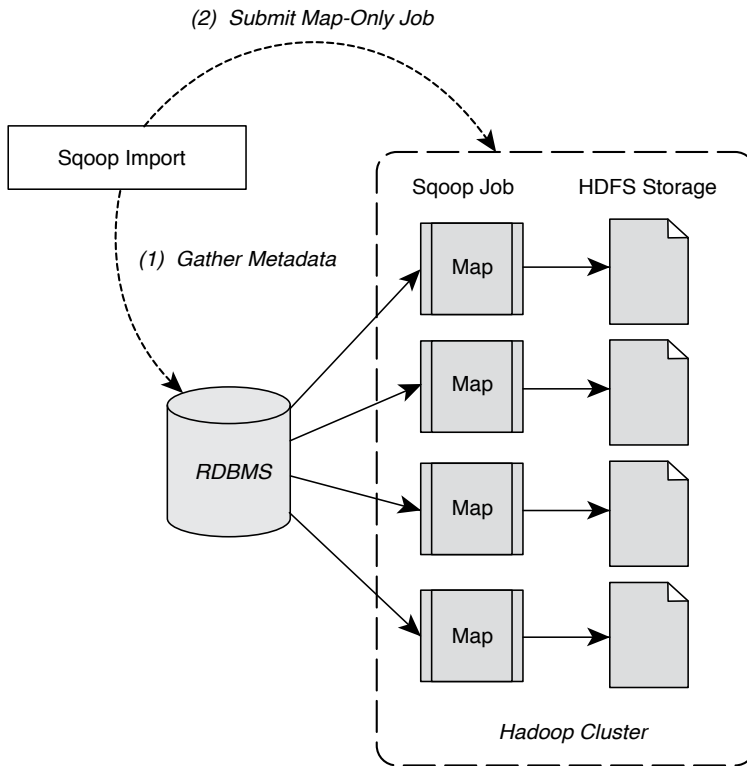


Figure 4.2 Two-step Apache Sqoop data import method.

2. A **map-only job** is a term used in the Hadoop ecosystem to refer to a map-reduce job that has some logic implemented in the map stage, and nothing (no-op) in the reduce job.

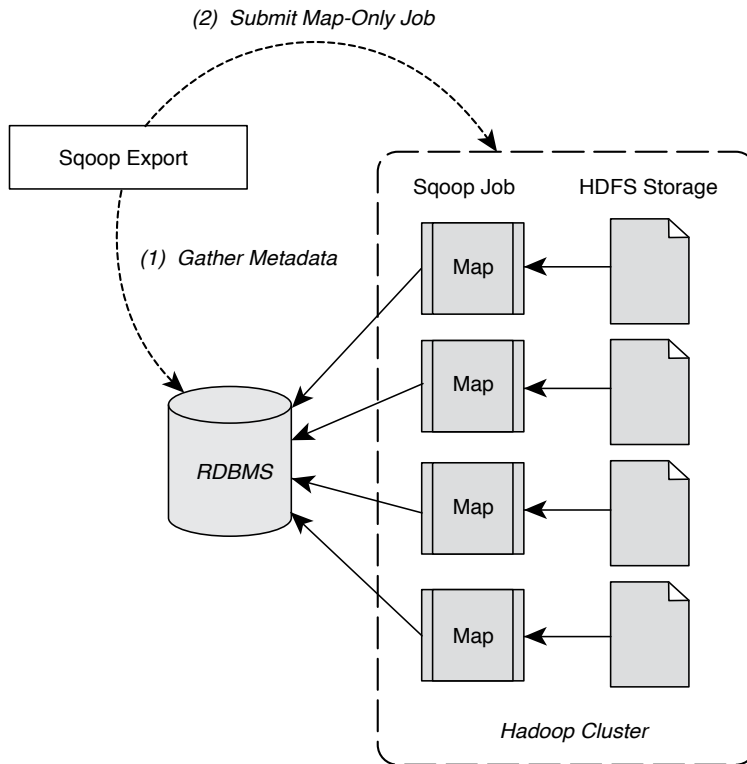


Figure 4.3 Two-step Sqoop data export method.

The imported data is saved in an HDFS directory. Sqoop will use the database name for the directory or the user can specify any alternative directory where the files should be populated. By default, these files contain comma-delimited fields, with new lines separating different records. You can easily override the format in which data is copied over by explicitly specifying the field separator and record terminator characters. Once placed in HDFS, the data are ready for further processing.

Data export from the cluster works in a similar fashion. The export is done in two steps as shown in Figure 4.3. Like the import process, the first step is to examine the database for metadata, followed by the export step that is again a map-only Hadoop job to write the data to the target database. Sqoop divides the input dataset into splits and then uses individual map tasks to push the splits to the database. Again, this process assumes the map tasks have access to the database.

Apache Sqoop Version Changes

Two versions of Sqoop are in general use within the Hadoop ecosystem. Many users have found the features removed in version 2 to be useful and continue to use the first version. Sqoop version 2 will be used for the examples.

Table 4.1 Apache Sqoop version comparison.

Feature	Sqoop Version 1	Sqoop Version 2
Connectors for all major RDBMS	Supported	Not supported. Use the generic JDBC Connector.
Kerberos Security Integration	Supported	Not supported
Data transfer from RDBMS to Hive or HBase	Supported	Not supported. First import data from RDBMS into HDFS, then load data into Hive or HBase manually.
Data transfer from Hive or HBase to RDBMS	Not supported. First export data from Hive or HBase into HDFS, and then use Sqoop for export.	Not supported. First export data from Hive or HBase into HDFS, and then use Sqoop for export.

Sqoop version 1 uses specialized connectors to access external database systems. These are often optimized for various RDBMS systems or those that do not support JDBC (Java Database Connectivity). Connectors are plug-in components based on Sqoop's extension framework and can be added to any existing Sqoop installation. Once a connector is installed, Sqoop can use it to efficiently transfer data between Hadoop and the external store supported by the connector. By default, Sqoop version 1 includes connectors for various popular databases such as MySQL, PostgreSQL, Oracle, SQL Server, and DB2. Sqoop version 1 also supports direct transfer to and from the RDBMS for HBase or Hive.

In order to streamline the Sqoop input methods (the issues cited were increasingly complex command lines, security, and the need to understand too many low-level issues), Sqoop version 2 no longer supports specialized connectors or direct import into HBase or Hive or direct data transfer from Hive or HBase to your RDBMS. There are more generalized ways to accomplish these tasks in version 2. All import and export is done through the JDBC interface. Table 4.1 summarizes the changes. Due to these changes, any new development should be done with attention to Sqoop version 2 capabilities.

Using Sqoop V2: A Basic Example

To better understand how to use Sqoop in practice, we're going to demonstrate how to configure and use Sqoop version 2 via a simple example. The example can then be extended as needed to explore the other capabilities offered by Apache Sqoop. More detailed information can be found at the Sqoop website at <http://sqoop.apache.org>.

The following steps will be performed:

1. Download and load sample MySQL data
2. Add Sqoop user permissions for local machine and cluster
3. Import data from MySQL to HDFS
4. Export data from HDFS to MySQL

Step 1: Download a Sample MySQL Database

For this example, we assume MySQL is installed on the Sqoop node and will use the world example database from the MySQL site (<http://dev.mysql.com/doc/world-setup/en/index.html>). The database has three tables:

- Country—Information about countries of the world.
- City—Information about some of the cities in those countries.
- CountryLanguage—Languages spoken in each country.

1. To get the database, use `wget`³ to download and then extract the file:

```
$ wget http://downloads.mysql.com/docs/world.sql.gz
$ gunzip world.sql.gz
```

2. Next, log into MySQL (assumes you have privileges to create a database) and import that database by entering the following commands:

```
$ mysql -u root -p
mysql> CREATE DATABASE world;
mysql> USE world;
mysql> SOURCE world.sql;
mysql> SHOW TABLES;
+-----+
| Tables_in_world |
+-----+
| City             |
| Country         |
| CountryLanguage |
+-----+
3 rows in set (0.01 sec)
```

3. The following MySQL commands will let you see the details for each table (output omitted because of space considerations):

```
mysql> SHOW CREATE TABLE Country;
mysql> SHOW CREATE TABLE City;
mysql> SHOW CREATE TABLE CountryLanguage;
```

Step 2: Add Sqoop User Permissions for Local Machine and Cluster

Sqoop often needs to talk to MySQL from the Hadoop cluster. Thus, there needs to be permissions added to MySQL so that these conversations can take place. Depending on your installation, you may need to add several privileges for Sqoop requests based on the location (hosts or IP addresses) from where the request originates. For example, the following permissions were assigned for the example.

3. `wget` is a command line tool for Unix/Linux environments that directly downloads files from a valid URL. If using a Windows environment, consider `Winwget` or a browser. If using a Macintosh environment, consider using `curl -O <url>` or a browser.

```
mysql> GRANT ALL PRIVILEGES ON world.* To 'sqoop'@'localhost'
➤ IDENTIFIED BY 'sqoop';
mysql> GRANT ALL PRIVILEGES ON world.* To 'sqoop'@'_HOSTNAME_'
➤ IDENTIFIED BY 'sqoop';
mysql> GRANT ALL PRIVILEGES ON world.* To 'sqoop'@'_SUBNET_'
➤ IDENTIFIED BY 'sqoop';
FLUSH PRIVILEGES;
mysql> quit
```

The `_HOSTNAME_` is the name of the host on which a user has logged in. The `_SUBNET_` is the subnet of the cluster (for example `10.0.0.%`, defines `10.0.0.0/24` network). These permissions allow any node in the cluster to execute MySQL commands as user `sqoop`. Also, for the purposes of this example, the Sqoop password is “sqoop.”

Next, log in as user `sqoop` to test the MySQL permissions.

```
$ mysql -u sqoop -p
mysql> USE world;
mysql> SHOW TABLES;
+-----+
| Tables_in_world |
+-----+
| City             |
| Country          |
| CountryLanguage |
+-----+
3 rows in set (0.01 sec)

mysql> quit
```

Step 3: Import Data Using Sqoop

As a check of Sqoop’s capability to read the MySQL database, we can use Sqoop to list the databases in MySQL.

1. Enter the following commands. The results are after the warnings at the end of the output. Note the use of local `_HOSTNAME_` in the JDBC statement. Extra notifications have been removed from the output (represented by `...`).

```
$ sqoop list-databases --connect jdbc:mysql://_HOSTNAME_/world
➤ --username sqoop --password sqoop
...
information_schema
test
world
```

2. In a similar fashion, Sqoop can connect to MySQL and list the tables in the world database.

```
$ sqoop list-tables --connect jdbc:mysql://_HOSTNAME_/world
➤ --username sqoop --password sqoop
...
City
Country
CountryLanguage
```

3. In order to import data, we need to make a directory in HDFS:

```
$ hdfs dfs -mkdir sqoop-mysql-import
```

4. The following command will import the Country table into HDFS:

```
$ sqoop import --connect jdbc:mysql://_HOSTNAME_/world --username
➤ sqoop --password sqoop --table Country -m 1 --target-dir
➤ /user/username/sqoop-mysql-import/country
```

The option `--table` signifies the table to import, `--target-dir` is the directory created above, and `-m 1` tells sqoop to use a single map task (which is enough in our example since it is only a small table) to import the data.

5. The import can be confirmed by examining HDFS:

```
$ hdfs dfs -ls sqoop-mysql-import/country
Found 2 items
-rw-r--r--  2 username hdfs          0 2014-08-18 16:47 sqoop-mysql-
➤ import/world/_SUCCESS
-rw-r--r--  2 username hdfs      31490 2014-08-18 16:47 sqoop-mysql-
➤ import/world/part-m-00000
```

6. The file can be viewed using the `hdfs -cat` command:

```
$ hdfs dfs -cat sqoop-mysql-import/country/part-m-00000
ABW,Aruba,North America,Caribbean,193.0,null,103000,78.4,828.0,793.0,
➤ Aruba,Nonmetropolitan Territory of The Netherlands,Beatrix,129,AW
...
ZWE,Zimbabwe,Africa,Eastern Africa,390757.0,1980,11669000,37.8,
➤ 5951.0,8670.0,Zimbabwe,Republic,Robert G. Mugabe,4068,ZW
```

To make Sqoop commands more convenient, an options file may be created and used in the command line. This file will help you avoid having to rewrite the same options. For example, a file called `world-options.txt` with the following contents will include the import command, `--connect`, `--username`, and `--password` options:

```
import
--connect
jdbc:mysql://_HOSTNAME_/world
--username
sqoop
--password
sqoop
```

The same import command from the preceding can be performed with the following shorter line:

```
$ sqoop --options-file world-options.txt --table City -m 1 --target-dir
➤ /user/username/sqoop-mysql-import/city
```

It is also possible to include an SQL Query in the import step. For example, if we want just cities in Canada:

```
SELECT ID,Name from City WHERE CountryCode='CAN'
```

Then we can include the `--query` option in the Sqoop import request. In the following query example, a single mapper task is designated with the `-m 1` option:

```
sqoop --options-file world-options.txt -m 1 --target-dir
➔ /user/username/sqoop-mysql-import/canada-city --query
➔ "SELECT ID,Name from City
➔ WHERE CountryCode='CAN' AND \$CONDITIONS"
```

Inspecting the results shows only cities from Canada are imported.

```
$ hdfs dfs -cat sqoop-mysql-import/canada-city/part-m-00000
```

```
1810,Montréal
1811,Calgary
1812,Toronto
...
1856,Sudbury
1857,Kelowna
1858,Barrie
```

Since there was only one mapper process, only one copy of the query needed to be run on the database. The results are also reported in single file (`part-m-0000`). Multiple mappers can be used to process the query if the `--split-by` option is used. The `split-by` option is a way to parallelize the SQL query. Each parallel task runs a subset of the main query with results partitioned by bounding conditions inferred by Sqoop. Your query must include the token `$CONDITIONS`; this is a placeholder for Sqoop to put in unique condition expression based on the `--split-by` option, and Sqoop automatically populates this with the right conditions for each mapper task. Sqoop will try to create balanced sub-queries based on a range of your primary key. However, it may be necessary to split on another column if your primary key is not uniformly distributed.

The following example will help illustrate the `-split-by` option. First, remove the results of the previous query.

```
$ hdfs dfs -rm -r -skipTrash sqoop-mysql-import/canada-city
```

Next, run the query using four mappers (`-m 4`) where we split by the ID number (`--split-by ID`).

```
sqoop --options-file world-options.txt -m 4 --target-dir
➔ /user/username/sqoop-mysql-import/canada-city --query "SELECT ID,
➔ Name from City WHERE CountryCode='CAN' AND \$CONDITIONS" --split-by ID
```

If we look at the number of results files, we find four files corresponding to the four mappers we requested in the command. There is no need to combine these files into one entity because all Hadoop tools can manage multiple files as input.

```
$ hdfs dfs -ls sqoop-mysql-import/canada-city
Found 5 items
-rw-r--r-- 2 username hdfs 0 2014-08-18 21:31 sqoop-mysql-import/canada-
city/_SUCCESS
-rw-r--r-- 2 username hdfs 175 2014-08-18 21:31 sqoop-mysql-import/canada-
city/part-m-00000
```

```
-rw-r--r--  2 username hdfs      153 2014-08-18 21:31 sqoop-mysql-import/canada-
city/part-m-00001
-rw-r--r--  2 username hdfs      186 2014-08-18 21:31 sqoop-mysql-import/canada-
city/part-m-00002
-rw-r--r--  2 username hdfs      182 2014-08-18 21:31 sqoop-mysql-import/canada-
city/part-m-00003
```

Step 4: Export Data Using Sqoop

The first step when exporting data with Sqoop is to create tables in the target database system for the exported data. There are actually two tables needed for each exported table. The first is a table to hold the exported data (e.g., CityExport) and the second is a table to be used for staging the exported data (e.g., CityExportStaging).

1. Using the following MySQL commands, you can create the tables:

```
mysql> USE world;
mysql> CREATE TABLE `CityExport` (
  `ID` int(11) NOT NULL AUTO_INCREMENT
  `Name` char(35) NOT NULL DEFAULT '',
  `CountryCode` char(3) NOT NULL DEFAULT '',
  `District` char(20) NOT NULL DEFAULT '',
  `Population` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`ID`));
mysql> CREATE TABLE `CityExportStaging` (
  `ID` int(11) NOT NULL AUTO_INCREMENT,
  `Name` char(35) NOT NULL DEFAULT '',
  `CountryCode` char(3) NOT NULL DEFAULT '',
  `District` char(20) NOT NULL DEFAULT '',
  `Population` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`ID`));
```

2. Next, create a cities-export-options.txt file similar to the world-options.txt file created above, using the export instead of import command. The following will export the cities data we imported above back into MySQL:

```
sqoop --options-file cities-export-options.txt --table CityExport
➤ --staging-table CityExportStaging --clear-staging-table -m 4
➤ --export-dir /user/username/sqoop-mysql-import/city
```

3. Finally, to make sure everything worked, check the table in MySQL to see if the cities are in the table.

```
$ mysql> select * from CityExport limit 10;
+-----+-----+-----+-----+-----+
| ID | Name          | CountryCode | District    | Population |
+-----+-----+-----+-----+-----+
| 1 | Kabul         | AFG         | Kabul       | 1780000    |
| 2 | Qandahar      | AFG         | Qandahar    | 237500     |
| 3 | Herat         | AFG         | Herat       | 186800     |
| 4 | Mazar-e-Sharif | AFG         | Balkh       | 127800     |
| 5 | Amsterdam     | NLD         | Noord-Holland | 731200     |
| 6 | Rotterdam     | NLD         | Zuid-Holland | 593321     |
+-----+-----+-----+-----+-----+
```

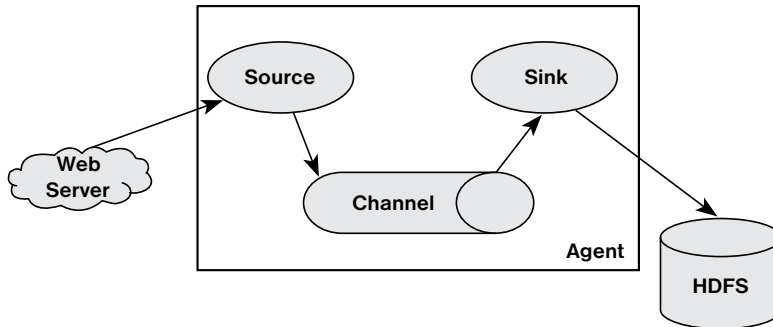



Figure 4.4 Flume Agent with Source, Channel, and Sink.

in the channel until a sink removes the data. By default, the data in a channel is kept in memory but optionally may be stored on disk to prevent data loss in the event of a network failure.

As shown in Figure 4.5, Flume agents may be placed in a pipeline. This configuration is normally used when data is collected on one machine (e.g., a web server) and sent to another machine that has access to HDFS.

In a Flume pipeline, the sink from one agent is connected to the source of another. The data transfer format normally used by Flume is called Apache Avro⁴ and provides several useful features. First, Avro is a data serialization/deserialization system that uses a compact binary format. The schema is sent as part of the data exchange and is defined using JavaScript Object Notation (JSON). Avro also uses remote procedure calls (RPC) to send data. That is, an Avro sink will contact an Avro source to send data.

Another useful Flume configuration is shown in Figure 4.6. In this configuration, Flume is used to consolidate several data sources before committing them to HDFS.

There are many possible ways to construct Flume transport networks.

The full scope of Flume functionality is beyond the scope of this book, and there are many additional features in Flume such as plug-ins and interceptors that can enhance

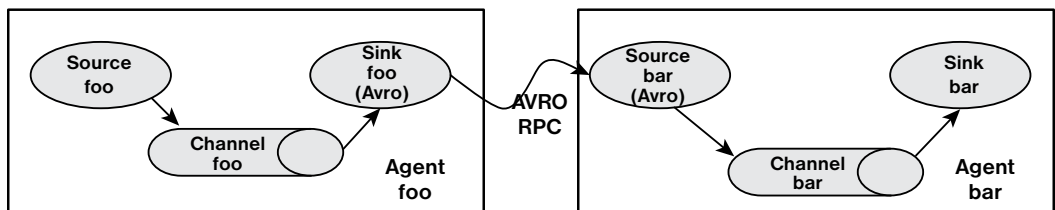


Figure 4.5 Pipeline created by connecting Flume agents.

4. <https://avro.apache.org/>

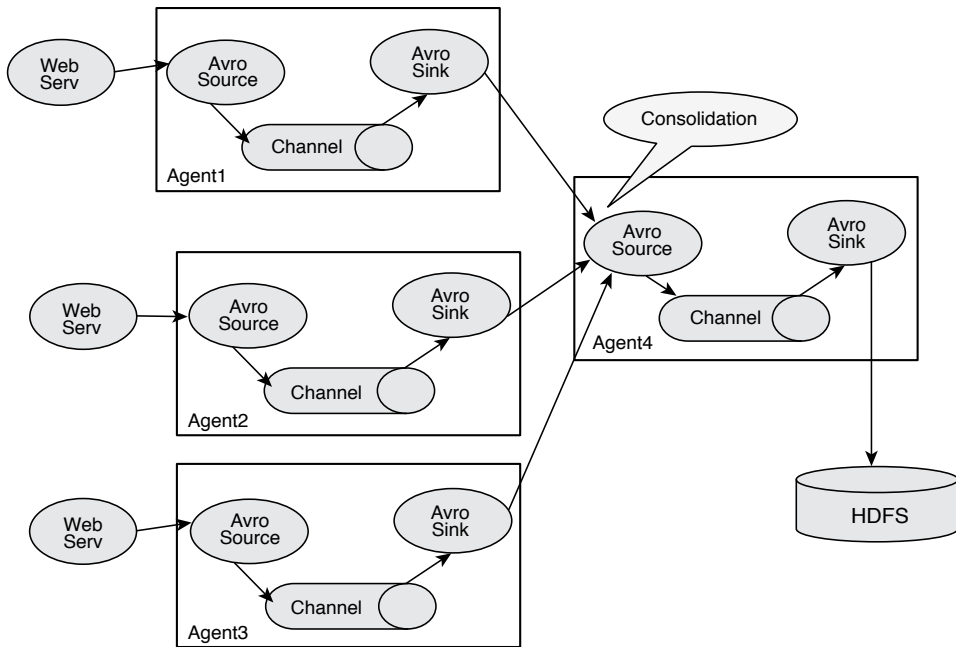


Figure 4.6 A Flume consolidation network.

Flume pipelines. For more information and example configurations, please see the Flume Users Guide at <https://flume.apache.org/FlumeUserGuide.html>.

Using Flume: A Web Log Example Overview

In this example web logs from the local machine will be placed into HDFS using Flume. This example is easily modified to use other web logs from different machines. The full source code and further implementation notes are available from the book web page in Appendix A, “Book Web Page and Code Download.” Two files are needed to configure Flume. (See the sidebar “Flume Configuration Files.”)

- `web-server-target-agent.conf`—The target Flume agent that writes the data to HDFS
- `web-server-source-agent.conf`—The source Flume agent that captures the web log data

The web log is also mirrored on the local file system by the agent that writes to HDFS.

1. To run the example, create the directory as root.

```
# mkdir /var/log/flume-hdfs
# chown hdfs:hadoop /var/log/flume-hdfs/
```


2. Next, as user `hdfs`, make a Flume data directory in HDFS.

```
$ hdfs dfs -mkdir /user/hdfs/flume-channel/
```

3. Now that the data directories are created, the Flume target agent can be started (as user `hdfs`).

```
$ flume-ng agent -c conf -f web-server-target-agent.conf -n collector
```

This agent writes the data into HDFS and should be started before the source agent. (The source reads the web logs.)

Note

In some Hadoop distributions, Flume can be started as a service when the system boots, such as “service start flume.” This configuration allows for automatic use of the Flume agent. The `/etc/flume/conf/{flume.conf,flume-env.sh.template}` files need to be configured for this purpose. For this example, the `/etc/flume/conf/flume.conf` file can be the same as the `web-server-target.conf` file (modified for your environment).

The source agent can be started as root, which will start to feed the web log data to the target agent. Note that the source agent can be on another machine:

```
# flume-ng agent -c conf -f web-server-source-agent.conf -n source_agent
```

To see if Flume is working, check the local log by using `tail`. Also check to make sure the `flume-ng` agents are not reporting any errors (filename will vary).

```
$ tail -f /var/log/flume-hdfs/1430164482581-1
```

The contents of the local log under `flume-hdfs` should be identical to that written into HDFS. The file can be inspected using the `hdfs -tail` command. (filename will vary). Note, while running Flume, the most recent file in HDFS may have a `.tmp` appended to it. The `.tmp` indicates that the file is still being written by Flume. The target agent can be configured to write the file (and start another `.tmp` file) by setting some or all of the `rollCount`, `rollSize`, `rollInterval`, `idleTimeout`, and `batchSize` options in the configuration file.

```
$ hdfs dfs -tail flume-channel/apache_access_combined/150427/FlumeData.  
1430164801381
```

Both files should have the same data in them. For instance, the preceding example had the following in both files:

```
10.0.0.1 - - [27/Apr/2015:16:04:21 -0400] "GET /ambarinagios/nagios/nagios_alerts  
.php?q1=alerts&alert_type=all HTTP/1.1" 200 30801 "-" "Java/1.7.0_65"  
10.0.0.1 - - [27/Apr/2015:16:04:25 -0400] "POST /cgi-bin/rrd.py HTTP/1.1" 200 784  
"- " "Java/1.7.0_65"  
10.0.0.1 - - [27/Apr/2015:16:04:25 -0400] "POST /cgi-bin/rrd.py HTTP/1.1" 200 508  
"- " "Java/1.7.0_65"
```

Both the target and source file can be modified to suit your system.

Flume Configuration Files

A complete explanation of Flume configuration is beyond the scope of this chapter. The Flume website has additional information on Flume configuration at <http://flume.apache.org/FlumeUserGuide.html#configuration>.

The two files describe two Flume agents that have separate Source/Channel/Sink configurations. Some of the important settings used in the example above are as follows:

In `web-server-source-agent.conf`, the following lines set the source. Note that the web log is acquired by using the `tail` command to record the log file.

```
source_agent.sources = apache_server
source_agent.sources.apache_server.type = exec
source_agent.sources.apache_server.command = tail -f /etc/httpd/logs/access_log
```

Further down in the file, the sink is defined. The parameter `source_agent.sinks.avro_sink.hostname` is used to assign the Flume node that will write to HDFS. The port number is also set in the target configuration file.

```
source_agent.sinks = avro_sink
source_agent.sinks.avro_sink.type = avro
source_agent.sinks.avro_sink.channel = memoryChannel
source_agent.sinks.avro_sink.hostname = 192.168.93.24
source_agent.sinks.avro_sink.port = 4545
```

The HDFS settings are placed in the `web-server-target-agent.conf` file. Note the path that was used in the previous example and the data specification.

```
collector.sinks.HadoopOut.type = hdfs
collector.sinks.HadoopOut.channel = mc2
collector.sinks.HadoopOut.hdfs.path = /user/hdfs/flume-channel/{log_type}/
%y%m%d
collector.sinks.HadoopOut.hdfs.fileType = DataStream
```

The target file also defines the port and two channels (`mc1` and `mc2`). One of the channels writes the data to the local file system and the other writes to HDFS. The relevant lines are shown in the following:

```
collector.sources.AvroIn.port = 4545
collector.sources.AvroIn.channels = mc1 mc2

collector.sinks.LocalOut.sink.directory = /var/log/flume-hdfs
collector.sinks.LocalOut.channel = mc1
```

The HDFS file rollover counts create a new file when a threshold is exceeded. In this example, allow any file size and write a new file after 10,000 events or 600 seconds.

```
collector.sinks.HadoopOut.hdfs.rollSize = 0
collector.sinks.HadoopOut.hdfs.rollCount = 10000
collector.sinks.HadoopOut.hdfs.rollInterval = 600
```

A full discussion of Flume can be found on the website at <https://flume.apache.org>.

Manage Hadoop Work and Data Flows with Apache Oozie

Apache Oozie is a workflow scheduler system designed to run and manage multiple related Apache Hadoop jobs. For instance, complete data input and analysis may require several discrete Hadoop jobs to be run as a workflow where the output of one job will be the input for a successive job. Oozie is designed to construct and manage these workflows.

Oozie is not a substitute for the YARN scheduler mentioned previously. That is, YARN manages resources for individual Hadoop jobs, and Oozie provides a way to connect and control multiple Hadoop jobs on the cluster.

Oozie workflow jobs are represented as DAGs of actions. There are three types of Oozie jobs:

- **Workflow:** A specified sequence of Hadoop jobs with outcome-based decision points and control dependency. Progress from one action to another cannot happen until the first action is complete.
- **Coordinator:** A scheduled workflow job that can run at various time intervals or when data becomes available.
- **Bundle:** A higher-level Oozie abstraction that will batch a set of coordinator jobs.

Oozie is integrated with the rest of the Hadoop stack supporting several types of Hadoop jobs out of the box (such as Java MapReduce, Streaming MapReduce, Pig, Hive, Spark, and Sqoop) as well as system-specific jobs (such as Java programs and shell scripts). Oozie also provides a CLI and a Web UI for monitoring jobs. An example of a simple Oozie workflow is shown in Figure 4.7. In this example, Oozie runs a basic MapReduce operation. If the application was successful the job ends; if there was an error, the job is killed.

Oozie workflow definitions are written in Hadoop Process Definition Language (hPDL), which is an XML-based process definition language. Oozie workflows contain several types of nodes.

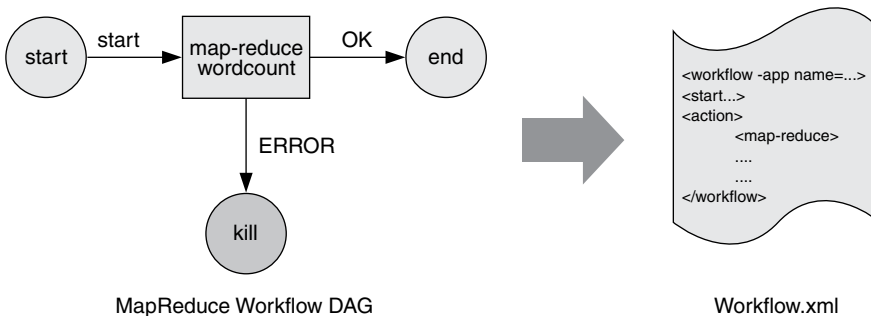


Figure 4.7 A simple Oozie DAG workflow.

- **Start/Stop control flow nodes** define the beginning and the end of a workflow. These include start, end, and optional fail nodes.
- **Action nodes** are where the actual processing tasks are defined. When an action node finishes, the remote systems notify Oozie and the next node in the workflow is executed. Action nodes can also include HDFS commands.
- **Fork/join nodes** allow parallel execution of tasks in the workflow. The fork node allows two or more tasks to run at the same time. A join node represents a rendezvous point that must wait until all forked tasks complete.
- **Control flow nodes** enable decisions to be made about the previous task. Control decisions are based on the results of the previous action (e.g. file size or file existence). Decision nodes are essentially switch-case statements that use JSP EL (Java Server Pages-Expression Language) that evaluates to either true or false.

A more complex workflow that uses all the above nodes is shown in the example workflow in Figure 4.8. More information on Oozie can be found at <http://oozie.apache.org/docs/4.0.0/index.html>.

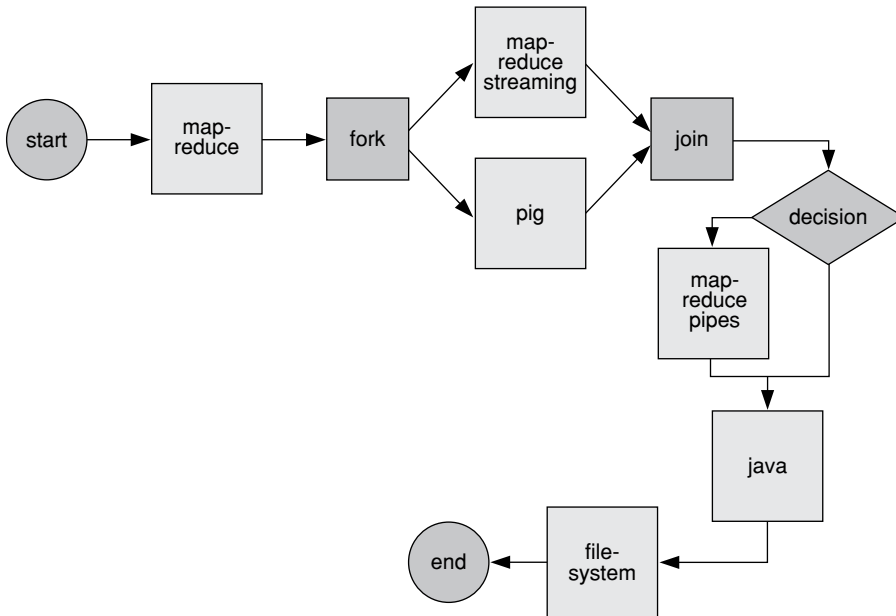


Figure 4.8 A more complex Oozie DAG workflow.

Apache Falcon

Apache Falcon simplifies the configuration of data motion by providing replication, life cycle management, lineage, and traceability. These features provide data governance consistency across Hadoop components that is not possible using Oozie. For instance, Falcon allows Hadoop administrators to centrally define their **data pipelines**, and then Falcon uses those definitions to auto-generate workflows in Apache Oozie. In simple terms, proper use of Falcon helps keep your active Hadoop cluster from becoming a confusing mess.

For example, Oozie lets you define Hadoop processing through workflow and coordinator (a recurring workflow) jobs. The input datasets for data processing are often described as part of coordinator jobs that specify properties such as path, frequency, schedule runs, and so on. If there are two coordinator jobs that depend on the same data, these details have to be defined and managed twice. If you want to add shared data deletion or movement, a separate coordinator is required. Oozie will certainly work in these situations, but there is no easy way to define and track the entire data life cycle or manage multiple independent Oozie jobs.

Oozie is useful when initially setting up and testing workflows and can be used when the workflows are independent and not expected to change often. If there are multiple dependencies between workflows or there is a need to manage the entire data life cycle, then Falcon should be considered.

As mentioned, as Hadoop's high-level workflow scheduler, Oozie may be managing hundreds to thousands of coordinator jobs and files. This situation results in some common mistakes. Processes might use the wrong copies of datasets. Datasets and processes may be duplicated, and it becomes increasingly more difficult to track down where a particular dataset originated. At that level of complexity, it becomes difficult to manage so many dataset and process definitions.

To solve these problems, Falcon allows the creation of a pipeline that is defined by three key attributes:

- A **cluster** entity that defines where data, tools, and processes live on your Hadoop cluster. A cluster entity contains things like the namenode address, Oozie URL, etc., which it uses to execute the other two entities: feeds and processes.
- A **feed** entity defines where data live on your cluster (in HDFS). The feed is designed to designate to Falcon where your new data (that's either ingested, processed, or both) live so it can retain (through retention policies) and replicate (through replication policies) the data on or from your Cluster. A feed is typically (but doesn't have to be) the output of a process.
- A **process** entity defines what action or "process" will be taking place in a pipeline. Most typically, the process links to an Oozie workflow, which contains a series of actions to execute (such as shell scripts, Java Jars, Hive actions, Pig actions, Sqoop Actions, you name it) on your cluster. A process also, by definition, takes feeds as inputs or outputs and is where you define how often a workflow should run.

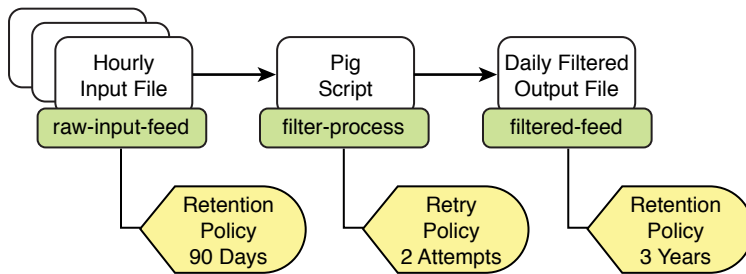


Figure 4.9 A simple Apache Falcon workflow.

The following example will help explain how Falcon is used. Assume there is raw input data that arrives every hour. These data are processed with a Pig script and the results saved for later processing. At a simple level an Oozie workflow could easily manage the task. However, high-level features, not available in Oozie, are needed to automate the process. First, the input data have a retention policy of 90 days, after which old data are discarded. Second, the processing step may have a certain number of retries should the process fail. And, finally, the output data have a retention policy of three years (and location). It is also possible to query data lineage with Falcon (i.e., Where did this data come from?). The simple job flow is shown in Figure 4.9.

What's Next in Data Ingestion?

As the Hadoop platform continues to evolve, innovation in ingestion tools continues. Two important new tools are now available to ingestion teams that we would like to mention:

- **Apache Nifi** is a recent addition to the data ingestion toolset. Originally created at the NSA and recently open sourced and added to the Apache family, Nifi provides a scalable way to define data routing, transformation, and system mediation logic. An excellent UI makes building data flows in Nifi fast and easy. Nifi provides support for lineage tracking and the security and monitoring capability that make it a great tool for data ingestion, especially for sensor data.
- **Apache Atlas** provides a set of core data governance services that enables enterprises to effectively deal with compliance requirements on Hadoop.

Summary

In this chapter

- The Hadoop data lake concept was presented as a new model for data processing.
- Various methods for making data available to several Hadoop tools were outlined. The examples included copying files directly to HDFS, importing CSV files to Apache Hive and Spark, and importing JSON files into HIVE with Spark.

- Apache Sqoop was presented as a tool for moving relational data into and out of HDFS.
- Apache Flume was presented as tool for capturing and transporting continuous data, such as web logs, into HDFS.
- The Apache Oozie workflow manager was described as a tool for creating and scheduling Hadoop workflows.
- The Apache Falcon tool enables a high-level framework for data governance (end-to-end management) by keeping Hadoop data and tasks organized and defined as pipelines.
- New tools like Apache Nifi and Atlas were mentioned as options for governance and data flow on a Hadoop cluster.

This page intentionally left blank

Index

A

Accuracy

- in data quality, 87
- in predictive modeling, 137

ACM-SIGKDD conference, 6

Action nodes, Oozie, 80

Affinity analysis. *See* Market basket analysis.

Agents, 39–40

AI (artificial intelligence), 6. *See also* Machine learning.

Algorithms

- clustering. *See* Clustering algorithms.
- FFT algorithm, invention of, 6
- machine learning, 132

Algorithms, supervised learning

- association rules, 132
- clustering, 132
- collaborative filtering, 132
- decision trees, 140
- GLMs (generalized linear models), 140
- important considerations, 141
- k -nearest-neighbor, 140
- LASSO (least absolute shrinkage, and selection operator), 140
- linear regression, 140
- logistic regression, 140
- neural network, 140
- random forest, 140
- tree ensembles, 140

Amazon in the history of data science, 7–8

Analysis of variance, history of, 6

Anomaly detection

- building a big data solution, 171–172
- with clustering, 152
- collective anomalies, 167, 169
- conditional anomalies, 167
- contextual anomalies, 166–167
- Distributed Processing System, 171
- Event Store systems, 171

- example, detecting network intrusions, 172–178

extreme value analysis, 165

global anomalies, 166–167

machine learning, 129

outlier analysis, data quality issues, 90–91

overview, 165

point anomalies, 166–167

sequence anomalies, 167, 169

sketch techniques, 171

time-series data, 168

tuning, 170

types of anomalies, 166–167

uses for, 166

Anomaly detection, approaches to

k -means clustering, 169–170

k -medoids clustering, 170

k -nearest neighbor, 170

local outlier factor, 169–170

multivariate datasets with clustering, 169–170

PAM (partition around medoids), 170

rules-based methods, 167

semi-supervised methods, 170

statistical identification of point anomalies, 168–169

supervised learning methods, 168

unsupervised learning methods, 168–170

Apache Atlas tool, 82

Apache Nifi tool, 58–59

Applied scientists, description, 8–9

AR (autoregressive datasets), 101

ARIMA (autoregressive integrated moving average), 101

Artificial intelligence (AI), 6. *See also* Machine learning.

Association rules, machine learning, 132

Association rules algorithm, 132

AUC (area-under-the-curve), 139

- Audio data, in the history of data science, 21
- Automated data discovery, 195–197
- Avro, with Flume, 75–76
- Awadallah, Amr, 38
- Ayasdi, 196
- B**
- Bag-of-words model
for NLP, 186, 187–188
for text, 97
- BalderSchweilier, Eric, 31
- Bar charts
examples, 113–114
misuse of, 111
stacked, 115–116
stacked area, 116–117
- Batch prediction, 143–144
- Benford, Frank, 88–89
- Benford’s Law, 88–89
- Big-model NLP, 184, 186–187
- Bing in the history of data science, 7–8
- Bisciglia, Christophe, 38
- Books and publications. *See also* Online resources.
“Efficient Estimation of Word Representations in Vector Space,” 188
“The Future of Data Analysis,” 6
Programming HIVE, 41
Programming Pig, 42
“Statistical Modeling: The Two Cultures,” 7
“Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks,” 198
- Box plot, invention of, 6
- Box-and-whisker charts, 118–119
- Breiman, Leo, 7
- Bundle jobs, Oozie, 79
- C**
- Cafarella, Mike, 37–38
- Casey, Stella
about the author, xxiii
becoming a data scientist, 11
- Categorical values
clustering, similarity measures, 154
data munging feature matrix, 103
predictive modeling, 134
- Cell-based rules, data quality issues
description, 88
example, 92
- Centers for Medicare and Medicaid Services (CMS), 28, 85–86
- Channel component of Flume, 74
- Charts. *See* Visualization; *specific charts*.
- chi-squared test, data quality issues, 90
- Chomsky, Noam, 182
- Classification
in machine learning, 129
vs. regression, 134–135
- Classifiers
building, predictive modeling example, 149–150
predictive modeling, performance metrics, 136–139
- Classifying text elements into categories, 99–100
- Cleaning and normalizing data. *See* Data munging.
- Click-through-rate (CTR), 9
- Cloudera, founding of, 38
- Cluster analysis. *See* Clustering.
- Cluster entities, Falcon, 81–82
- Clustering
anomaly detection, 169–170
for anomaly detection, 152
choosing K, 158
in the data science project lifecycle, 16
end-to-end architecture, 158–160
evaluating clusters, 157–158
for exploratory analysis, 152
in machine learning, 129
overview, 151–152
for pre-processing data, 152
uses for, 152
- Clustering, similarity measures
categorical variables, 154
distance functions, 153–154
Euclidean distance, 153
Hamming distance, 153
Jaccard coefficient, 154
Manhattan distance, 153
Minkowski distance, 153–154
mixed feature sets, 154
similarity functions, 154
Taxicab distance, 153

- Clustering algorithms
 - density-based, 155
 - graph based, 155
 - grid-based, 155
 - hard *vs.* soft clustering, 155
 - hierarchical, 155
 - machine learning, 132
 - model-based, 155
 - partitioning-based, 154
 - Clustering algorithms, examples
 - k*-means clustering, 155
 - k*-means++ clustering, 156
 - k*-medians clustering, 156
 - k*-medoids clustering, 156
 - LDA (Latent Dirichlet Allocation), 157, 160–163
 - PAM (partition around medoids), 156
 - topic modeling, 160–163
 - CMS (Centers for Medicare and Medicaid Services), 28, 85–86
 - Collaborative filtering, 17, 132
 - Collective anomalies, 167, 169
 - Comma-separated values (CSV), importing to Hive tables, 59–62
 - Common words, removing, 97–99
 - Comparison charts, 111–114
 - Completeness, in data quality, 86–87
 - Complex data types
 - data munging, feature matrix, 100
 - feature extraction, 101–102
 - Complex features, 97. *See also* Data munging, feature matrix.
 - Composition charts
 - definition, 112
 - description, 114–117
 - pie charts, 115
 - Composition charts, bar charts
 - stacked, 115–116
 - stacked area, 116–117
 - Conditional anomalies, 167
 - Configuration files, Flume, 76–77, 78
 - Confusion matrix, 136–139
 - Consistency, in data quality, 87
 - Consolidation network, Flume, 75–76
 - Contextual anomalies, 166–167
 - Contingency table. *See* Confusion matrix.
 - Control flow nodes, Oozie, 80
 - Coordinator jobs, Oozie, 79
 - Copying files to/from HDFS, 206–207
 - CoreNLP package, 46
 - Correlation, history of, 6
 - Credit card fraud, costs of, 25
 - Cross validation, predictive modeling
 - performance metrics, 139–140
 - CSV (comma-separated values), importing to Hive tables, 59–62
 - CTR (click-through-rate), 9
 - Curiosity, attribute of data scientists, 11
 - Curse of dimensionality, 104
 - Customer churn analysis, use case for, 22
 - Customer segmentation, use case for, 22–23
 - Cutting, Doug, 37–38
- D**
- DAG (directed acyclic graph), 37
 - Data acquisition in the data science project lifecycle, 15
 - Data cleaning in the data science project lifecycle, 15–16
 - Data engineers, description, 8–9
 - Data ingestion, HDFS
 - Apache Atlas tool, 58–59
 - Apache Nifi tool, 58–59
 - direct file transfer, 58–59
 - slicing and dicing input, 58
 - Data ingestion, increased velocity of, 21. *See also* Importing data.
 - Data lake
 - advantages of, 56–58
 - availability of access methods, 56–57
 - data availability, 56–57
 - data sharability, 56–57
 - description, 19, 56–58
 - ETL (extract, transform, and load), 56
 - predetermined schema, 56
 - preprocessing data, 56
 - value of, 15
 - Data manipulation and analysis, Python language, 45
 - Data motion, configuring, 81–82
 - Data munging
 - advantages of Hadoop, 86
 - definition, 85
 - feature engineering, 85
 - feature generation, 85
 - for predictive modeling, 142

- Data munging, data quality
 - accuracy, 87
 - Benford's Law, 88–89
 - cell-based rules, 88, 92
 - chi-squared test, 90
 - completeness, 86–87
 - consistency, 87
 - definition, 86–87
 - differential “value distribution” rules, 89–90
 - distance between two distributions, 90
 - fraud detection, 89
 - with Hadoop, 92–93
 - Kullback-Leibler divergence pseudo-metric., 90
 - missing value imputation, 91
 - outlier analysis, 90–91
 - overview, 87
 - static “value distribution” rules, 88–89
 - timeliness, 87
 - validity, 87
- Data munging, feature matrix
 - aggregated features, 96
 - categorical values, 103
 - choosing instances, 94
 - choosing the “right” feature, 94
 - complex data types, 100. *See also specific types.*
 - complex features, 97
 - creating, 85
 - curse of dimensionality, 104
 - dimensionality reduction, 103–105
 - examples, 94
 - extracting text from PDFs, 101
 - feature manipulation, 102–103
 - feature projection, 105
 - feature value discretization, 102–103
 - feature vectors, 93
 - generating features, example with Hive, 96–97
 - generating features, overview, 96–97
 - image data, 102
 - measuring stock prices over time, 100
 - one-hot encoding, 103
 - overview, 93–94
 - sampling, 94–95
 - scaling feature values, 103
 - simple features, 96
 - text features. *See* Data munging, text features.
 - thresholding, 102–103
 - time-series features, 100–101
 - video data, 102
- Data munging, text features
 - bag-of-words text model, 97
 - classifying text elements into categories, 99–100
 - dimensionality reduction, Spark example, 105
 - extracting text from PDFs, 101
 - hashing trick, 99, 105–106
 - ICA (independent component analysis), 105
 - identifying intent, 100
 - LDA (linear discriminant analysis), 105
 - LSI (latent semantic indexing), 105
 - NER (named entity extraction), 99–100
 - overview, 97–102
 - PCA (principal component analysis), 105
 - removing common words, 97–99
 - stemming, 105
 - stop words, removing, 105
 - term frequency, 99
 - TF-IDF (term frequency-inverse document frequency), 97–99
 - word distribution. *See* Hashing trick; Zipf's Law.
 - word vectorization, 100
 - Zipf's Law, 105
- Data pipelines, creating, 81–82
- Data preparation. *See also* Feature extraction.
 - in big data clustering solutions, 158–159
 - predictive modeling example, 145–146
 - for scalable models, 50
 - in sentiment analysis, example, 145–146
- Data quality
 - in the data science project lifecycle, 15–16
 - issues. *See* Data munging, data quality.
- Data science
 - computational topology, 8
 - deep learning, 197–199
 - deep nets, 197–199
 - definition, 3–4
 - future of, 195–199

- history of. *See* History of data science.
- iterative discovery process, 4
- in modern enterprise, 8
- representation learning, 197
- semantic hashing, 197
- Data science project lifecycle
 - classification *vs.* regression, 17
 - cluster analysis, 16
 - collaborative filtering, 17
 - data acquisition, 15
 - data cleaning, 15–16
 - data quality, 15–16
 - deploying to production, 17
 - design model features, 16
 - diagram, 14
 - exploring data, 16
 - feature engineering, 16
 - machine learning model, building, 17
 - modeling, 16
 - predictive modeling, 16
 - recommender system, building, 16
 - setting success criteria, 14–15
 - supervised learning, 17
 - tuning a machine learning model, 17
 - unsupervised learning, 17
- Data science projects, managing, 18
- Data science teams, building
 - competition for qualified applicants, 13
 - from existing employees, 13
 - hiring new employees, 12–13
 - lack of experience among hiring management, 13
 - obstacles to, 12–13
 - shortage of qualified applicants, 12–13
- Data scientists
 - applied scientists, 8–9
 - data engineers, 8–9
 - hard skills, 9–10
 - required skillset, 9–10
 - research scientists, 8–9
 - soft skills, 11–12
 - strategies for becoming, 9–10
- Data scientists, key attributes
 - curiosity, 11
 - love of learning, 11
 - persistence, 12
 - story-telling, 12
- Data variety in the history of data science, 20–21
- Data velocity in the history of data science, 21
- Data volume in the history of data science, 20
- Databases, listing, 70–71
- DataFrames
 - data sources, 62
 - definition, 42
 - description, 62
- Decision trees
 - in the history of data science, 6
 - supervised learning algorithms, 140
- Deep learning, 7, 128, 197–199
- Deep nets, 197–199
- Deleting
 - directories in HDFS, 207
 - files in HDFS, 207
- Density-based clustering algorithm, 155
- Dependent variables in predictive modeling, 134
- Deploying to production in the data science project lifecycle, 17
- Design model features in the data science project lifecycle, 16
- Detecting network intrusions, anomaly detection example, 172–178
- Differential “value distribution” rules, 89–90
- Dimensionality reduction, 103–105
- Direct file transfer with Sqoop, 58–59
- Directed acyclic graph (DAG), 37
- Directories, creating in HDFS, 206
- Distance between two distributions, 90
- Distributed Processing System for anomaly detection, 171
- Distributed systems abstraction, levels of, 49–50
- Distribution charts
 - box-and-whisker charts, 118–119
 - common types of, 117–118
 - definition, 112
 - description, 117–118
 - dot charts, example, 117–118
 - histograms, example, 117–118
- Dixon, James, 19
- Dot charts, example, 117–118
- DStream (discretized stream), 43
- Dynamic time warping, 101

E

Eadline, Douglas

about the author, xxiii

becoming a data scientist, 12

eBay, in the history of data science, 7–8

“Efficient Estimation of Word

Representations in Vector Space,” 188

Electronic patient records, 27–28

ELIZA, 182

English, translating to Russian, 182

Error matrix. *See* Confusion matrix.

ETL (extract, transform, and load) from data lakes, 56

Euclidean distance, clustering similarity measure, 153

Evaluating predictive modeling classifiers, 136–139

Event Store systems for anomaly detection, 171

Evolution of Hadoop, 37–38

Examples. *See also* Use cases.

bar charts, 113–114

cell-based rules, 92

code, online resources for, 201

data munging, feature matrix, 94

dot charts, 117–118

exporting data with Sqoop, 73–74

Falcon workflow, 81–82

Flume web log, 76–77

in healthcare, validating HCPCS codes, 92–93

histograms, 117–118

Hive, sampling for a feature matrix, 94–95

line charts, 113–114

network throughput, visualization, 108–110

Oozie workflow, 79–80

pie charts, 115

Pig, sampling for a feature matrix, 94–95

predictive modeling, 133–134

predictive modeling, classification *vs.* regression, 135

predictive modeling, design choices, 136

sampling for a feature matrix, 94–95

search advertising, 4–5

sentiment analysis, 189–193

TF-IDF (term frequency-inverse document frequency), 98–99

Examples, clustering algorithms

k-means clustering, 155

k-means++ clustering, 156

LDA (Latent Dirichlet Allocation), 157, 160–163

topic modeling, 160–163

Examples, Spark

dimensionality reduction, 105

removing common words from text, 98–99

sampling for a feature matrix, 94–95

word vectorization, 100

word2vec, 100

Examples, Sqoop

exporting data, 73–74

version 2, 68–74

Examples, text features

classifying text elements into categories, 99

removing common words, 98–99

word vectorization, 100

Exploratory analysis, with clustering, 152

Exporting relational data, 66–67

External Hive tables, 59

Extract, transform, and load (ETL) from data lakes, 56

Extracting text from PDFs, 101

Extreme value analysis, 165

F

Facebook

founding of Cloudera, 38

history of data science, 7–8

Falcon

cluster entities, 81–82

configuring data motion, 81–82

creating data pipelines, 81–82

description, 40

feed entities, 81–82

lifecycle management, 81–82

lineage, 81–82

process entities, 81–82

replication, 81–82

sample workflow, 81–82

traceability, 81–82

Fault tolerance in Hadoop, 31–32

- Feature engineering in the data science project lifecycle, 16. *See also* Data munging, feature matrix.
 - Feature extraction. *See also* Data preparation.
 - in big data clustering solutions, 158–159
 - from complex data types, 101–102
 - for scalable models, 50
 - Feature generation
 - data munging. *See* Data munging, feature matrix.
 - definition, 85
 - predictive modeling example, 146–149
 - Feature learning, 197
 - Feature manipulation, 102–103
 - Feature matrix
 - data munging. *See* Data munging, feature matrix.
 - example, 128–129
 - Feature projection, 105
 - Feature value discretization, 102–103
 - Feature vectors, 93
 - Features
 - generating, 96–97
 - machine learning, 128
 - Feed entities, Falcon, 81–82
 - FFT algorithm, invention of, 6
 - Files
 - copying to/from HDFS, 206–207
 - deleting in HDFS, 207
 - importing into Hive tables, 59
 - viewing, 58
 - First-Digit Law. *See* Benford’s Law.
 - Fisher, Ronald, 6
 - Flink, 37. *See also* MapReduce.
 - Flume
 - acquiring data streams, 74–78
 - Avro, 75–76
 - channel component, 74
 - components, 74
 - configuration files, 76–77, 78
 - consolidation network, 75–76
 - description, 39–40
 - pipeline configuration, 75
 - sink component, 74
 - source component, 74
 - starting at boot, 77
 - web log example, 76–77
 - Fork/join nodes, Oozie, 80
 - Fourier transform, 101
 - Fraud costs
 - credit card fraud, 25
 - healthcare claim fraud, 25
 - Fraud detection
 - Benford’s Law, 89
 - data quality issues, 89
 - use case for, 25
 - fsck HDFS option, 208
 - “The Future of Data Analysis,” 6
- G**
- GadFly, visualization tool, 123
 - Galton, Francis, 6
 - Gas well. *See* Oil and gas well.
 - Georgetown University, 182
 - get HDFS command, 58
 - ggplot Python package, visualization tool, 122
 - ggplot2 package, visualization tool, 122
 - GIF files, extracting time-series data, 101
 - GLMs (generalized linear models), 140
 - Global anomalies, 166–167
 - Google
 - “Efficient Estimation of Word Representations in Vector Space,” 188
 - history of data science, 7–8
 - word vectorization, 100
 - word2vec, 100
 - Graph based clustering algorithm, 155
 - GraphX, 43
 - Graphics, visualization tools, 121–122
 - Graph-N-Go, visualization tool, 122
 - Graphs. *See* Visualization; *specific graphs*.
 - Grid-based clustering algorithm, 155
 - Grosse, Roger, 198
- H**
- Hadoop
 - advantages in data munging, 86
 - definition, 31
 - evolution of, 37–38
 - fault tolerance, 31–32
 - parallel processing and predictive modeling, 143
 - running Microsoft, 49
 - running SAP, 49

- Hadoop (*continued*)
 - running SAS, 49
 - vs.* traditional data warehouses, 56–57
 - working with relational data. *See* Sqoop.
- Hadoop, uses for data scientists
 - cost effective storage, 46–47
 - levels of distributed systems abstraction, 49–50
 - multi-language tooling, 48–49
 - resource management, 49
 - scalable application of models, 51
 - scalable creation of models, 50–51
 - scheduling, 49
 - semi-structured data, 48
 - unstructured data, 48
- Hadoop components
 - distributed data processing frameworks. *See* MapReduce; Parallel MapReduce.
 - distributed file system. *See* HDFS (Hadoop Distributed Filesystem).
 - resource management. *See* YARN (Yet Another Resource Negotiator).
 - scheduling. *See* YARN (Yet Another Resource Negotiator).
 - Spark, 37, 42
 - Tez, 37
- Hadoop Distributed Filesystem (HDFS). *See* HDFS (Hadoop Distributed Filesystem).
- Hadoop Process Definition Language (hPDL), 79
- Hadoop tools for data science
 - aggregating and moving log data into HDFS. *See* Flume.
 - bulk data transfer. *See* Sqoop.
 - data manipulation. *See* R language.
 - data processing and management. *See* Falcon; Oozie.
 - distributed in-memory data processing. *See* Spark.
 - graphical display. *See* Python language, Java language, R language.
 - Java, 8, 31, 40
 - multi-step ETL. *See* Pig.
 - performing calculations. *See* Python language, Java language, R language.
 - programming languages. *See* Python language; R language.
 - Python, 8–9, 42, 45
 - R, 8–9, 44–45
 - SQL analytics. *See* Hive; Spark SQL.
 - statistical analysis. *See* R language.
- Hammerbacher, Jeff, 38
- Hamming distance, clustering similarity measure, 153
- Hard *vs.* soft clustering, 155
- Hashing trick, 99, 105–106. *See also* Zipf’s Law.
- HDFS (Hadoop Distributed Filesystem)
 - copying files, 58
 - description, 32–34
 - and existing POSIX file systems, 58
 - importing data to. *See* Data ingestion, HDFS.
 - online resources, 203
 - viewing files, 58
- HDFS (Hadoop Distributed Filesystem), quick start
 - copying files to/from HDFS, 206–207
 - creating a directory, 206
 - deleting directories, 207
 - deleting files, 207
 - `fsck` option, 208
 - general user commands, 204–205
 - getting a status report, 207–208
 - `hdfs` command, 203
 - health check, 208
 - listing files, 205–206
 - quick command dereference, 204
- `hdfs` command, 203
- Health check on HDFS, 208
- Healthcare
 - claim fraud, costs of, 25
 - detecting anomalous record access, 28
 - electronic patient records, 27–28
 - HEDIS (Healthcare Effectiveness Data and Information Set), 27
 - ICD10 standard for patient records, 27
 - PMI (Precision Medicine Initiative) Cohort program, 27–28
 - predicting patient re-admission, 28
 - predictive medical diagnosis, 27–28
 - validating HCPCS codes, example, 92–93
- HEDIS (Healthcare Effectiveness Data and Information Set), 27

- Help for data scientists. *See* Books and publications; Online resources.
 - Heritage Provider Network, 28
 - Hidden Markov models (HMMs), 101
 - Hierarchical clustering algorithm, 155
 - Hierarchical learning, 197–199
 - Histograms, example, 117–118
 - History of data science
 - AI (artificial intelligence), 6
 - decision trees, 6
 - deep learning, 7
 - KDD workshops, 6
 - k*-means clustering, 6
 - nearest-neighbors, 6
 - neural networks, 6
 - significant contributors, 7–8. *See also individual names.*
 - statistics and machine learning, 6–7
 - support vector machines, 6
 - supporting technological/scientific achievements, 5–6
 - History of data science, role of big data
 - audio data, 21
 - log files, 20
 - sensor data, 20
 - text data, 21
 - variety of data, 20–21
 - velocity of data, 21
 - video data, 21
 - volume of data, 20
 - Hive
 - description, 40–41
 - Programming HIVE*, 41
 - sampling for a feature matrix, example, 94–95
 - SQL analytics, 40–41
 - UDFs (user-defined functions), 40–41
 - User-Defined Aggregation functions, 40–41
 - word-count script, 41
 - Hive tables
 - external, 59
 - importing files into, overview, 59
 - internal, 59
 - Hive tables, importing data from files
 - CSV files, 59–62
 - overview, 59
 - TSV (tab-separated values), 59
 - Hive tables, importing data with Spark
 - CSV files, 63–64
 - overview, 62–63
 - Hivemall, 41
 - HMMs (hidden Markov models), 101
 - Hoaglin, David, 91
 - Hortonworks, founding of, 38
 - HPDL (hadoop Process Definition Language), 79
- I**
- IBM, 182
 - ICA (independent component analysis), 105
 - ICD10 standard for patient records, 27
 - Iglewicz, Boris, 91
 - Image data, data munging feature matrix, 102
 - Image processing, 198
 - Importing data. *See also* Data ingestion.
 - files into Hive tables, 59, 64
 - JSON files, 64
 - relational data, 66–67
 - Independent variables, predictive modeling, 134
 - Installing and configuring Sqoop, 68–74
 - Insurance risk analysis, use case, 29
 - Intent, identifying in text, 100
 - Internal Hive tables, 59
- J**
- Jaccard coefficient, 154
 - Java packages for Hadoop, 46
 - JPEG files, extracting time-series data, 101
 - JSON files
 - extracting time-series data, 101
 - importing into Hive tables, 64
 - Julia language, visualization tool, 123
- K**
- Kaggle.com competition, 28
 - KDD workshops in the history of data science, 6
 - K*-fold cross-validation, predictive modeling
 - performance metrics, 139–140
 - k*-means clustering
 - anomaly detection, 169–170
 - example, 155
 - k*-means++ clustering, example, 156

- K-means clustering in the history of data science, 6
- k*-medians clustering, example, 156
- k*-medoids clustering, anomaly detection, 170
- K*-medoids clustering, example, 156
- k*-nearest-neighbor
 - anomaly detection, 170
 - description, 140
- Kullback-Leibler divergence pseudo-metric., 90
- L**
- LASSO (least absolute shrinkage, and selection operator), 140
- lattice package, visualization tool, 122
- LDA (linear discriminant analysis), 105
- LDA (Latent Dirichlet Allocation), examples, 157, 160–163
- Lee, Honglak, 198
- Libraries
 - Matlab, 123
 - Python language, 45
 - Seaborn, 122
 - Spark. *See* MLlib.
- Libraries, matplotlib
 - plotting library, 45
 - visualization tool, 122
- Lifecycle management, Falcon, 81–82
- Likelihood, history of, 6
- Line charts, examples, 113–114
- Lineage, Falcon, 81–82
- Linear regression, supervised learning
 - algorithms, 140
- LinkedIn in the history of data science, 7–8
- Listing files in HDFS, 205–206
- Local outlier factor, 169–170
- Log files in the history of data science, 20
- Logistic regression, supervised learning
 - algorithms, 140
- Love of learning, attribute of data scientists, 11
- LSI (latent semantic indexing), 105–106
- M**
- MA (moving average) datasets, 101
- Machine learning
 - algorithms, 132
 - association rules algorithm, 132
 - benefits of big data, 130–131
 - clustering algorithm, 132
 - collaborative filtering algorithm, 132
 - deep learning, 128
 - feature matrix, example, 128–129
 - features, definition, 128
 - future of, 132
 - Java packages for, 46
 - neural networks, 128
 - observations, definition, 128
 - overview, 127–128
 - perceptron, 127
 - support vector machines, 128
 - text mining, 46
 - tools for, 131–132
 - unsupervised learning, 129
 - Vowpal Wabbit, 46
 - WEKA pkg, 46
- Machine learning, supervised learning
 - algorithms for, 132, 140–141
 - definition, 129
- Machine learning, targets
 - definition, 128
 - task types, 129
- Machine learning, task types
 - anomaly detection, 129
 - classification, 129
 - clustering, 129
 - market basket analysis, 129
 - predictive modeling, 129
 - recommender systems, 129
 - regression, 129
- Machine learning libraries
 - Python language, 45
 - Spark MLlib, 132
- Machine learning models
 - in the data science project lifecycle, 17
 - tuning, 17
- MAE (mean absolute error), predictive modeling performance metric, 139
- Mahout, 186
- Mallet package, 46
- Managing data science projects *vs.* managing software projects, 18
- Manhattan distance, clustering similarity measure, 153
- Map phase, MapReduce, 35–36
- Map-only jobs, 66–67
- MapReduce. *See also* Flink; Spark; Tez.
 - description, 35–37

- for highly iterative jobs, 36
 - map phase, 35–36
 - phases, 35
 - reduce phase, 35–36
 - shuffle phase, 35
 - word-count script, 35
 - Market basket analysis
 - in machine learning, 129
 - use case for, 26–27
 - Matlab, matrix library, 123
 - matplotlib, 45, 122
 - Mendelevitch, Ofer
 - about the author, xxiii
 - becoming a data scientist, 9
 - work on search advertising, 9
 - Microsoft, running on Hadoop, 49
 - Minkowski distance, clustering similarity
 - measure, 153–154
 - Missing value imputation, 91
 - Mixed feature sets, clustering similarity
 - measure, 154
 - MLlib. *See also* Libraries.
 - description, 43
 - NLP example, 186–187
 - word vectorization, example, 100
 - word2vec, example, 100
 - Model tuning, predictive modeling, 142–143
 - Model-based clustering algorithm, 155
 - Modeling in the data science project lifecycle, 16
 - Models
 - scalable application, 51
 - scalable creation, 50–51
 - Moving average (MA), 101
 - MP3 files, extracting time-series data, 101
 - MP4 files, extracting time-series data, 101
 - Multi-language tooling, 48–49
 - Multi-layer neural networks, 197
 - Multivariate datasets with clustering,
 - anomaly detection, 169–170
 - Munging data. *See* Data munging.
- N**
- Named entity extraction (NER), 99–100
 - Named entity recognition, 184
 - Natural language processing (NLP). *See* NLP (natural language processing).
 - Nearest-neighbors
 - in the history of data science, 6
 - k -nearest neighbor, anomaly detection, 170
 - k -nearest-neighbor algorithm, 140
 - NER (named entity extraction), 99–100
 - Netflix in the history of data science, 7–8
 - Network intrusions, anomaly detection
 - example, 172–178
 - Neural networks
 - in the history of data science, 6
 - multi-layer, 197
 - supervised learning algorithms, 140
 - Ng, Andrew Y., 198
 - NLP (natural language processing)
 - Chomsky, Noam, 182
 - ELIZA, 182
 - Georgetown University, 182
 - historical approaches, 182
 - IBM, 182
 - named entity recognition, 184
 - overview, 181–182
 - Python language, 45
 - sentiment analysis, 184
 - tagging parts of speech, 183
 - text segmentation, 183
 - topic modeling, 184
 - translating Russian to English, 182
 - use cases, 182–183
 - Weizenbaum, Joseph, 182
 - NLP (natural language processing), examples
 - sentiment analysis, 189–193
 - with Spark, 189
 - with Stanford CoreNLP, 189
 - NLP (natural language processing), textual representations
 - bag of words, 186, 187–188
 - “Efficient Estimation of Word Representations in Vector Space,” 188
 - term frequencies, 188
 - term frequency scaled by TF-IDF, 188
 - word2vec, 188
 - NLP (natural language processing), tooling for Hadoop
 - bag of words model, 186
 - big-model NLP, 184, 186–187
 - Mahout, 186
 - small-model NLP, 184–186
 - Spark MLlib, 186–187
 - NLTK, Python language, 45
 - Numeric computing, Python language, 45

NumPy, Python language, 45
Nutch, in the evolution of Hadoop, 37–38

O

Observations, in machine learning, 128
Oil and gas well production predictions, use case for, 29
Olson, Mike, 38
One-hot encoding, 103
Online resources. *See also* Books and publications.
 deep learning, 199
 example code, 201
 Google, 100
 HDFS, 203
 question and answer forum, 201
 Sqoop project website, 66
 TDA (topological data analysis), 197
 webpage for this book, 201
 word2vec, 100

Oozie

 action nodes, 80
 bundle jobs, 79
 control flow nodes, 80
 coordinator jobs, 79
 description, 40
 fork/join nodes, 80
 hPDL (Hadoop Process Definition Language), 79
 job types, 79
 node types, 79–80
 scheduling workflow, 79–80
 start/stop control flow nodes, 80
 workflow example, 79–80
 workflow jobs, 79

OpenNLP package, 46

Outlier. *See* Anomaly.

P**Packages**

 CoreNLP, 46
 ggplot Python visualization tool, 122
 Java for Hadoop, 46
 Java for machine learning, 46
 Mallet, 46
 OpenNLP, 46
 R language, 44
 R package for visualization, 122

 RPlyr, 44

 Vowpal Wabbit, 46

 WEKA, 46

PAM (partition around medoids), 156, 170

“Panama” project, 9

Pandas, Python language, 45

Parallel MapReduce, 35

Partitioning-based clustering algorithm, 154

PayPal in the history of data science, 7–8

PCA (principal component analysis), 105

PDF files, extracting text from, 101

Pearson, Karl, 6

Pentaho, 19

Perceptron, 127

Performance metrics, predictive modeling.

See Predictive modeling, performance metrics.

Persistence, attribute of data scientists, 12

Phases of MapReduce, 35

Piatesky-Shapiro, Gregory, 6

Pie charts, 115

Pig

 description, 41–42

Programming Pig, 42

 sampling for a feature matrix, example, 94–95

 script for word-count, 42

Pig-on-Spark, 41

Pipeline configuration, Flume, 75

PMI (Precision Medicine Initiative) Cohort program, 27–28

PMML (predictive modeling markup language), 143

Point anomalies, 166–167, 168–169

POSIX file systems, and HDFS, 58

Precision in predictive modeling, definition, 137

Precision-recall curve, predictive modeling performance metric, 138–139

Predicting patient re-admission, use case for, 28

Predictive maintenance, use case for, 26

Predictive medical diagnosis, use case for, 27–28

Predictive modeling. *See also* Supervised learning.

 accuracy, definition, 137

 architectural view, 141–143

- batch prediction, 143–144
 - categorical targets, 134
 - cleaning and normalizing data, 142
 - confusion matrix, 136–139
 - in the data science project lifecycle, 16
 - dependent variables, 134
 - examples, 133–134
 - Hadoop parallel processing, 143
 - independent variables, 134
 - in machine learning, 129
 - model tuning, 142–143
 - munging data, 142
 - overview, 133–134
 - PMML (predictive modeling markup language), 143
 - precision, definition, 137
 - real-time prediction, 144
 - recall, definition, 137
 - specificity, definition, 137
 - supervised learning, algorithms for. *See* Algorithms, supervised learning.
 - training sets, 134
 - twitter. *See* Predictive modeling, sentiment analysis example.
 - Predictive modeling, classification
 - example, 135
 - vs.* regression, 134–135
 - Predictive modeling, design choices
 - evaluating classifiers, 136–139
 - example, 136
 - test sets, 136
 - training sets, 136
 - validation sets, 136
 - Predictive modeling, performance metrics
 - AUC (area-under-the-curve), 139
 - classifiers, 136–139
 - cross validation, 139–140
 - k*-fold cross-validation, 139–140
 - MAE (mean absolute error), 139
 - precision-recall curve, 138–139
 - regression models, 139
 - RMSE (root mean squared error), 139
 - ROC (receiver operating characteristic), 138–139
 - ROC curve, 138–139
 - Predictive modeling, regression
 - vs.* classification, 134–135
 - example, 135
 - Predictive modeling, sentiment analysis example
 - building classifiers, 149–150
 - data preparation, 145–146
 - feature generation, 146–149
 - tweets dataset, 145
 - Predictive modeling markup language (PMML), 143
 - Pre-processing data with clustering, 152
 - Principal component analysis (PCA), 105
 - Process entities, Falcon, 81–82
 - Product recommendation, use case for, 21–22
 - Programming HIVE*, 41
 - Programming Pig*, 42
 - put HDFS command, 58
 - PySpark, word-count script, 45–46
 - Python language
 - data manipulation and analysis, 45
 - machine learning library, 45
 - matplotlib, plotting library, 45
 - natural language processing, 45
 - NLTK, 45
 - numeric computing, 45
 - NumPy, 45
 - overview, 45–46
 - Pandas, 45
 - scientific computing, 45
 - scikit-learn, 45
 - SciPy, 45
 - Spacy, 45
 - text mining, 45
 - word-count script, 45–46
 - Python visualization tool, 122
- Q**
- QlikView, visualization tool, 123
- R**
- R Foundation for Statistical Computing, 121–122
 - R language
 - capabilities, 44
 - description, 44–45
 - packages, 44
 - RPLY, 44
 - RHadoop pkg, 44
 - RODBC pkg, 44
 - visualization tool, 121–122
 - word-count script, 44–45

- R Plyr package, 44
 - R project, 45
 - Random forest algorithm for supervised learning, 140
 - Ranganath, Rajesh, 198
 - RDD (resilient distributed dataset)
 - definition, 42
 - source for importing to HDFS, 62–64
 - Reading data into Hadoop. *See* Data ingestion; Importing data.
 - Real-time prediction, predictive modeling, 144
 - Recall in predictive modeling, definition, 137
 - Receiving operating characteristic (ROC), predictive modeling performance metric, 138–139
 - Recommender systems
 - in the data science project lifecycle, 16
 - in machine learning, 129
 - use case for, 21–22
 - Reduce phase, MapReduce, 35–36
 - Regression
 - vs.* classification, 134–135
 - history of, 6
 - in machine learning, 129
 - Regression models, predictive modeling performance metric, 139
 - Relationship charts
 - definition, 112
 - description, 118–120
 - scatter plots, 118–120
 - Removing
 - common words, 97–99
 - stop words, 105
 - Replication, Falcon, 81–82
 - Representation learning, 197
 - Research scientists, description, 8–9
 - Resilient distributed datastore (RDD)
 - definition, 42
 - source for importing to HDFS, 62–64
 - Resource management, 49
 - Resources for data scientists. *See* Books and publications; Online resources.
 - Reusing command options, Sqoop, 71–72
 - RHadoop pkg, 44
 - Right model formulation, identifying, 196
 - Risk analysis, use case for, 29
 - Risk models among auto insurers, 14
 - RMSE (root mean squared error), predictive modeling performance metric, 139
 - ROC (receiving operating characteristic), predictive modeling performance metric, 138–139
 - ROC curve, predictive modeling performance metric, 138–139
 - RODBC pkg, 44
 - Rosenblatt, Frank, 6
 - Rules-based anomaly detection methods, 167
 - Russian, translating to English, 182
- S**
- Sales use cases
 - diversifying sales, 22
 - increasing sales, 22
 - prioritizing leads, 23–24
 - Sampling, data munging feature matrix, 94–95
 - SAP, running on Hadoop, 49
 - Sarkar, Deepayan, 122
 - SAS (Statistical Analysis System)
 - running on Hadoop, 49
 - visualization tools, 122–123
 - SAS/Analyst, visualization tool, 123
 - SAS/Insight, visualization tool, 123
 - SAS/Procs, visualization tool, 123
 - Scaling feature values, data munging feature matrix, 103
 - Scatter plots, 118–120
 - Scheduling, 49
 - Scheduling workflow, Oozie, 79–80
 - Schema-on-read, 47
 - Scientific computing, Python language, 45
 - scikit-learn package, Python language, 45
 - SciPy, Python language, 45
 - Seaborn library, visualization tool, 122
 - Search advertising
 - examples of, 4–5
 - Yahoo! project “Panama,” 9
 - Semantic hashing, 197
 - Semi-structured data, 48
 - Semi-supervised anomaly detection methods, 170
 - Sensitivity. *See* Recall.

- Sensor data in the history of data science, 20
- Sentiment analysis
 - example, 189–193
 - with NLP, 184, 189–193
 - use case for, 24–25
- Sequence anomalies, 167, 169
- Shuffle phase, MapReduce, 35
- Similarity functions, clustering similarity measures, 154
- Similarity measures. *See* Clustering, similarity measures.
- Simple features, 96. *See also* Data munging, feature matrix.
- Sink component, Flume, 74
- Sketch techniques, 171
- Skill required for data scientists, 9–12
- Slicing and dicing input. *See also* Spark.
 - HDFS, 58
 - with Sqoop, 58
- Small-model NLP, 184–186
- Source component, Flume, 74
- Spacy, Python language, 45
- Spark.
 - architecture, 42–43
 - DataFrames, 42
 - description, 37
 - fault tolerant streaming applications. *See* Spark Streaming.
 - machine learning library. *See* MLlib.
 - NLP example, 189
 - Pig-on-Spark, 41
 - R language. *See* SparkR project.
 - RDD (resilient distributed datastore), 42
 - slicing and dicing data, 42
 - SQL analytics. *See* Spark SQL.
 - word-count script, 42–43
- Spark, examples
 - dimensionality reduction, 105
 - removing common words from text, 98–99
 - sampling for a feature matrix, 94–95
 - word vectorization, 100
 - word2vec, 100
- Spark MLlib. *See* MLlib.
- Spark components. *See specific components.*
- Spark SQL, 43
- Spark Streaming, 43
- SparkR project, 45
- Specificity in predictive modeling, definition, 137
- SQL analytics in Hive, 40–41
- SQL for Spark, 43
- Sqoop
 - clean-up commands, 74
 - definition, 65–66
 - description, 39
 - exporting data, example, 73–74
 - importing/exporting relational data, 66–67
 - installing and configuring, 68–74
 - listing databases, 70–71
 - project website, 66
 - reusing command options, 71–72
 - version 2, example, 68–74
 - version 2 *vs.* version 1, 67–68
 - version changes, 67–68
- Stanford CoreNLP, NLP example, 189
- Start/stop control flow nodes, Oozie, 80
- Static “value distribution” rules, 88–89
- Statistical Analysis System (SAS)
 - running on Hadoop, 49
 - visualization tools, 122–123
- Statistical computing, visualization tools, 121–122
- “Statistical Modeling: The Two Cultures,” 7
- Statistics
 - box plot, invention of, 6
 - Tukey’s HSD test, invention of, 6
- Statistics, and machine learning
 - overview, 6–7
 - statistical methods, 6
- Stemming, 105
- Stock prices, measuring over time, 100
- Stop words, removing, 105
- Storage, cost effectiveness, 46–47
- Story-telling, attribute of data scientists, 12
- Streaming, Spark, 43
- Success criteria in the data science project
 - lifecycle, 14–15
- Supervised learning. *See also* Predictive modeling.
 - algorithms for. *See* Algorithms, supervised learning.
 - anomaly detection, 168
 - in the data science project lifecycle, 17
- Support vector machines, 6, 128

T

- Tableau, visualization tool, 123
- Tables. *See* Hive tables.
- Tab-separated values (TSV), importing to
 - Hive tables, 59
- Tagging parts of speech, 183
- TDA (topological data analysis), 195–197
- Term frequency
 - computing, 99
 - definition, 188
 - scaled by TF-IDF, 188
- Test sets in predictive modeling design
 - choices, 136
- Text data in the history of data science, 21
- Text features, 97–102. *See also* Data munging, text features.
- Text mining
 - Python language, 45
 - tools for, 46
- Text segmentation, 183
- Tez, 37. *See also* MapReduce.
- TF-IDF (term frequency-inverse document frequency), 97–99
 - data munging, text features, 97–99
 - scaling term frequency, 188
- Thresholding, data munging feature matrix, 102–103
- Timeliness in data quality, 87
- Time-series data, anomaly detection, 168
- Time-series features. *See also* Data munging, feature matrix.
 - example, 100
 - overview, 100–101
- Tooling, multi-language, 48–49
- Tools. *See also specific tools.*
- Tools, machine learning, 131–132
- Tools, visualization
 - GadFly, 123
 - ggplot Python package, 122
 - ggplot2 package, 122
 - graphics, 121–122
 - Graph-N-Go, 122
 - Julia language, 123
 - lattice package, 122
 - Matlab, matrix library, 123
 - matplotlib library, 122
 - online resources, 122–123
 - Python, 122
 - QlikView, 123
 - R language, 121–122
 - SAS (Statistical Analysis System), 122
 - SAS/Analyst, 123
 - SAS/Insight, 123
 - SAS/Procs, 123
 - Seaborn library, 122
 - statistical computing, 121–122
 - Tableau, 123
- Tools for Hadoop data science
 - aggregating and moving log data into HDFS. *See* Flume.
 - bulk data transfer. *See* Sqoop.
 - data manipulation. Python language; R language.
 - data processing and management. *See* Falcon; Oozie.
 - distributed in-memory data processing. *See* Spark.
 - graphical display. *See* Python language; R language.
 - multi-step ETL. *See* Pig.
 - performing calculations..
 - programming languages. *See* Python language; R language.
 - SQL analytics. *See* Hive; Spark SQL.
 - statistical analysis. *See* R language.
- Topic modeling
 - example, 160–163
 - with NLP, 184
- Topological data analysis (TDA), 195–197
- Training sets
 - in machine learning, 129
 - predictive modeling, 134
 - predictive modeling, design choices, 136
- Translating Russian to English, 182
- Tree ensembles, supervised learning
 - algorithms, 140
- True Negative Rate. *See* Recall.
- True Positive Rate. *See* Recall.
- TSV (tab-separated values), importing to
 - Hive tables, 59
- Tukey, John W., 6, 118
- Tukey's HSD test, invention of, 6

- Tuning
 - anomaly detection, 170
 - predictive models, 142–143
- Tweets dataset, predictive modeling example, 145
- U**
- UDFs (user-defined functions) in Hive, 40–41
- Unstructured data, 48
- Unsupervised detection
 - of collective anomalies, 169
 - of sequence anomalies, 169
- Unsupervised learning
 - anomaly detection, 168–170
 - in the data science project lifecycle, 17
- “Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks,” 198
- Use cases, business
 - customer churn analysis, 22
 - customer segmentation, 22–23
 - fraud detection, 25
 - increasing sales, 22
 - increasing user satisfaction and loyalty, 22
 - insurance risk analysis, 29
 - market basket analysis, 26–27
 - predicting oil and gas well production, 29
 - predictive maintenance, 26
 - product recommendation, 21–22
 - sales diversity, 22
 - sales lead prioritization, 23–24
 - sentiment analysis, 24–25
- Use cases, healthcare
 - detecting anomalous record access, 28
 - electronic patient records, 27–28
 - HEDIS (Healthcare Effectiveness Data and Information Set), 27
 - ICD10 standard for patient records, 27
 - PMI (Precision Medicine Initiative) Cohort program, 27–28
 - predicting patient re-admission, 28
 - predictive medical diagnosis, 27–28
- Use cases, NLP, 182–183
- User satisfaction and loyalty, use case for, 22
- User-based-insurance (UBI). *See* Risk models.
- User-Defined Aggregation functions in Hive, 40–41
- User-defined functions (UDFs) in Hive, 40–41
- V**
- Validation sets, predictive modeling design choices, 136
- Validity in data quality, 87
- Video data
 - data munging, feature matrix, 102
 - in the history of data science, 21
- Viewing files, 58
- Visualization tools
 - GadFly, 123
 - ggplot Python package, 122
 - ggplot2 package, 122
 - graphics, 121–122
 - Graph-N-Go, 122
 - Julia language, 123
 - lattice package, 122
 - Matlab, matrix library, 123
 - matplotlib library, 122
 - online resources, 122–123
 - Python, 122
 - QlikView, 123
 - R language, 121–122
 - SAS (Statistical Analysis System), 122
 - SAS/Analyst, 123
 - SAS/Insight, 123
 - SAS/Procs, 123
 - Seaborn library, 122
 - statistical computing, 121–122
 - Tableau, 123
- Visualizations
 - with Hadoop, 123–124
 - misuse of, 110–112
 - network throughput, example, 108–110
 - uses in data science, 121
- Visualizations, chart types. *See also specific types.*
 - comparing variables. *See* Comparison charts.
 - composition of data items. *See* Composition charts.
 - displaying mean, median, and maximum values. *See* Box-and-whisker chart.

Visualizations, chart types (*continued*)
distribution of data. *See* Distribution charts.
a guide for choosing, online resource, 112
relationships between datasets or variables.
See Relationship charts.
Vowpal Wabbit package, 46

W

WAV files, extracting time-series data, 101
Wavelet transformations, 101
Webpages. *See* Online resources.
Weizenbaum, Joseph, 182
WEKA package, 46
Wickham, Hadley, 122
WMV files, extracting time-series data, 101
Word vectorization, 100
word2vec, 188
Word-count scripts
Hive, 41
MapReduce, 35
Pig, 42

PySpark, 45–46
Python language, 45–46
R language, 44–45
Spark, 42–43
Workflow jobs, Oozie, 79
Workflow sample, Falcon, 81–82

X

XML data, extracting time-series data, 101

Y

Yahoo!
founding of Cloudera, 38
founding of Hortonworks, 38
history of data science, 7–8
project “Panama,” 9
work on search advertising, 9
YARN (Yet Another Resource Negotiator),
34

Z

Zipf’s Law, 105. *See also* Hashing trick.