NOTE:

Unit 1,2,5 is available.

Question Bank – Big Data Analytics

Unit I

1. Define big data. Why is big data required? How does traditional BI environment differ from big data environment?
2. What are the challenges with big data?
3. Define big data. Why is big data required? Write a note on data warehouse environment.
4. What are the three characteristics of big data? Explain the differences between Bl and Data Science.
5. Describe the current analytical architecture for data scientists.
6. What are the key roles for the New Big Data Ecosystem?
7. What are key skill sets and behavioral characteristics of a data scientist?
8. What is big data analytics? Explain in detail with its example.
9. Write a short note on Classification of Analytics.
10. Describe the Challenges of Big Data.
11. What is big data analytics? Also write and explain importance of big data.
12. Write a short note on data science and data science process.
13. Write a short note on Soft state eventual consistency.
14. What are different phases of the Data Analytics Lifecycle? Explain each in detail.

Unit II

1. What is clustering? Explain in detail. Also explain any two of its applications.
2. Describe the steps to find k clusters using k-means algorithm.
3. How to generalize the k-means algorithm? Also write a short note on determining the number of clusters.
4. Write a short note on association rules.
5. What is the role of support in apriori algorithm? Also explain how the Apriori property works with a neat diagram.
6. Find the associative rule using Apriori algorithm; if there are four transactions – T1, T2, T3 and T4 for itemsets {A,B,C},{A,C},{A,D} and {B,E,F} respectively and minimum support and confidence are 50 %.
7. What is Linear regression? Explain in detail. Also explain any two of its applications.

8. Write a short note on linear regression model. Also apply Ordinary least Squares (OLS) technique to estimate the parameters.
9. Explain Linear Regression Model with Normally Distributed Errors.
10. What is Logistic regression? Explain in detail. Also explain any two of its applications.
11. Describe logistic regression model with respect to logistic function.

Unit III

1. Write a short note on decision tree.
2. How to to predict whether customers will buy a product or not? Explain with respect to decision tree.
3. Explain a probabilistic classification method based on Naive Bayes' theorem.
4. John flies frequently and likes to upgrade his seat to first class. He has determined that if he checks in for his flight at least two hours early, the probability that he will get an upgrade is 0.75; otherwise, the probability that he will get an upgrade is 0.35. With his busy schedule, he checks in at least two hours before his flight only 40% of the time. Suppose John did not receive an upgrade on his most recent attempt. What is the probability that he did not arrive two hours early? Find it with respect to on Bayes' theorem.
5. Describe additional classification methods other than decision tree and Bayes' theorem.
6. How to model a structure of observations taken over time? Explain with respect to Time series analysis. Also explain any two of its applications.
7. What are the components of time series? Explain each of them. Also write the main steps of Box-Jenkins methodology for time series analysis.
8. Explain Autoregressive Integrated Moving Average Model in detail.
9. Explain additional time series methods other than Box-Jenkins methodology and Autoregressive Integrated Moving Average Model.
10. What are major challenges with text analysis? Explain with examples.
11. What are various text analysis steps? Explain in detail.
12. Describe ACME's Text Analysis Process.
13. What is the use of Regular Expressions? Explain any five regular expressions with its description and example.

14. How to normalize the text using tokenization and case folding? Explain in detail. Also explain about Bag-of-words approach.
15. How to retrieve information and applying text analysis? Explain with respect to Term Frequency.
16. What is the critical problem in using Term frequency? How can it be fixed?
17. How to categorize documents by topics? Explain in detail.
18. What is sentiment analysis? How it can be carried out? Explain it in detail.

Unit IV

1. What is data science pipeline? Explain in detail with a neat diagram.
2. How to refactor the data science pipeline into an iterative model? Explain all its phases with a neat diagram.
3. List the requirements of distributed system in order to perform computation at scale. How Hadoop addresses these requirements?
4. Write a short note on Hadoop architecture.
5. Explain with a neat diagram a small Hadoop cluster with two master nodes and four workers nodes that implements all six primary Hadoop services.
6. Write a short note on Hadoop Distributed File System.
7. How basic interaction can be done in Hadoop distributed file system? Explain any five basic file system operations with its appropriate command.
8. What are various types of permissions in Hadoop distributed file system? What are different access levels? Write and explain commands to set various types and access levels. What is a caveat with file permissions on HDFS?
9. Explain functionality of map() function and reduce() function with a neat diagram in a MapReduce context.
10. How MapReduce can be implemented on a Cluster? Explain its all phases with a neat diagram.
11. Explain the details of data flow in a MapReduce pipeline executed on a cluster of a few nodes with a neat diagram.
12. Write a short note on job chaining.
13. Demonstrate the process of Hadoop streaming in a MapReduce context.
14. Demonstrate the process of Computing on CSV Data with Hadoop Streaming.
15. Demonstrate the process of executing a Streaming job on a Hadoop cluster.
16. Write a short note on Combiners in advanced MapReduce context.

17. Write a short note on Partitioners in advanced MapReduce context.
18. Write a short note on Job Chaining in advanced MapReduce context.
19. Write in brief about Spark. Also write and explain its primary components.

Unit V

1. Explain in detail how Keys allow parallel reduction by partitioning the keyspace to multiple reducers.
2. What is the functionality of the explode mapper? Explain in detail with example.
3. What is the functionality of the filter mapper? Explain in detail with example.
4. What is the functionality of the identity pattern? Explain in detail with example.
5. Write in brief about design pattern. Explain each of its category.
6. Consider a specific example where we have a dataset that originates from news articles or blog posts and a prediction task where we want to determine the number of comments in the next 24 hours. Then how the data flow will be?
7. Write a command for the following in Hive Query Language:
   i) changing directory to HIVE_HOME
   ii) creating a database
   iii) creating a table
   iv) loading data in a table
   v) counting number of rows in a table and
   vi) exiting the Hive CLI
8. Write and explain with suitable example any three data analysis commands with Hive.
9. What is the major drawback of conventional relational approach for many data analytics applications? How can it be resolved? Explain in detail.
10. How HBase schema can be created? How data can be inserted? and Cell values can be fetched? Explain with suitable example.
11. Which different types of filters can be used in HBase? Explain its entire procedure with appropriate commands.
12. Write the entire procedure with appropriate commands for importing data from MySQL to HDFS.
13. Write the entire procedure with appropriate commands for importing data from MySQL to Hive.
14. Write the entire procedure with appropriate commands for importing data from MySQL to HBase.

15. Explain Flume Data Flows with a neat diagram.
16. How to construct a simple singleagent Flume data flow to consume events from an Apache access log and write the log events to HDFS? Explain with a neat diagram.
17. Explain with example Relations, tuples and Filtering in context of Pig.
18. Explain with example Projection in context of Pig.
19. Explain with example Grouping and joiningin context of Pig.
20. Explain with example Storing and outputting datain context of Pig.
21. Write and describe various Pig relational operators.
22. Explain Spark SQL interface architecture with a neat diagram.

Unit I

Chapter I -Introduction toBig Data

What's in Store?

Irrespective of the size of the enterprise whether it is big or small, data continues to be a precious and irreplaceable asset. Data is present in homogeneous sources as well as in heterogeneous sources. The need of the hour is to understand, manage, process, and take the data for analysis to draw valuable insights. Digital data can be structured, semi-structured or unstructured data.

Data generates information and from information we can draw valuable insight. As depicted in Figure 1.1, digital data can be broadly classified into structured, semi-structured, and unstructured data.

l. Unstructured data: This is the data which does not conform to a data model or is not in a form which can be used easily by a computer program. About 80% data of an organization is in this format; for example, memos, chat rooms, PowerPoint presentations, images, videos, letters. researches, white papers, body of an email, etc.

2. Semi-structured data: Semi-structured data is also referred to as self-describing structure.This is the data which does not conform to a data model but has some structure. However, it is not in a form which can be used easily by a computer program. About 10% data of an organization is in this format; for example, HTML, XML, JSON, email data etc.

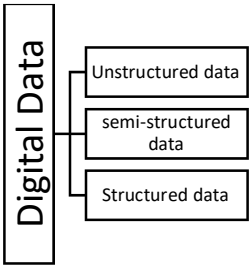


Figure 1.1 classification of digital data

3. Structured data: When data follows a pre-defined schema/structure we say it is structured data. This is the data which is in an organized form (e.g., in rows and columns) and be easily used by a computer program. Relationships exist between entities of data, such as classes and their objects. About 10% data of an organization is in this format. Data stored in databases is an example of structured data.

1.1 Introduction to Big Data

The "Internet of Things" and its widely ultra-connected nature are leading to a burgeoning rise in big data. There is no dearth of data for today's enterprise. On the contrary, they are mired in data and quite deep at that. That brings us to the following questions:

1. Why is it that we cannot forego big data?
2. How has it come to assume such magnanimous importance in running business?
3. How does it compare with the traditional Business Intelligence (BI) environment?
4. Is it here to replace the traditional, relational database management system and data warehouse environment or is it likely to complement their existence?"

Data is widely available. What is scarce is the ability to draw valuable insight.

Some examples of Big Data:

There are some examples of Big Data Analytics in different areas such as retail, IT infrastructure, and social media.

Retail: As mentioned earlier, Big Data presents many opportunities to improve sales and marketing analytics.

An example of this is the U.S. retailer Target. After analyzing consumer purchasing behavior, Target's statisticians determined that the retailer made a great deal of money from three main life-event situations.

Marriage, when people tend to buy many new products

Divorce, when people buy new products and change their spending habits

Pregnancy, when people have many new things to buy and have an urgency to buy them. The analysis target to manage its inventory, knowing that there would be demand for specific products and it would likely vary by month over the coming nine- to ten-month cycles

IT infrastructure: MapReduce paradigm is an ideal technical framework for many Big Data projects, which rely on large data sets with unconventional data structures.

One of the main benefits of Hadoop is that it employs a distributed file system, meaning it can use a distributed cluster of servers and commodity hardware to process large amounts of data.

Some of the most common examples of Hadoop implementations are in the social media space, where Hadoop can manage transactions, give textual updates, and develop social graphs among millions of users.

Twitter and Facebook generate massive amounts of unstructured data and use Hadoop and its ecosystem of tools to manage this high volume.

social media: It represents a tremendous opportunity to leverage social and professional interactions to derive new insights.

LinkedIn represents a company in which data itself is the product. Early on, Linkedln founder Reid Hoffman saw the opportunity to create a social network for working professionals.

As of 2014, Linkedln has more than 250 million user accounts and has added many additional features and data-related products, such as recruiting, job seeker tools, advertising, and lnMaps, which show a social graph of a user's professional network.

1.2 Characteristics of Data:

As depicted in Figure 1.2, data has three key characteristics:

1. Composition: The composition of data deals with the structure of data, that is, the sources of data, the granularity, the types, and the nature of data as to whether it is static or real-time streaming.
2. Condition: The condition of data deals with the state of data, that is, "Can one use this data as is foranalysis?" or "Does it require cleansing for further enhancement and enrichment?"
3. Context: The context of data deals with "Where has this data been generated?" "Why was this datagenerated?" How sensitive is this data?" "What are the events associated with this data?" and so on.

Small data (data as it existed prior to the big data revolution) is about certainty. It is about known datasources; it is about no major changes to the composition or context of data.

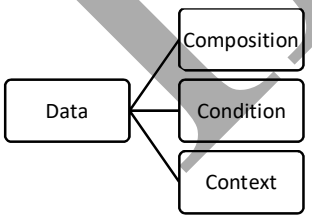


Figure 1.2 Characteristics of data

Most often we have answers to queries like why this data was generated, where and when it was generated, exactly how we would like to use it, what questions will this data be able to answer, and so on. Big data is about complexity. Complexity in terms of multiple and unknown datasets, in terms of exploding volume, in terms of speed at which the data is being

generated and the speed at which it needs to be processed and in terms of the variety of data (internal or external, behavioural or social) that is being generated.

1.3 Evolution of Big Data:

1970s and before was the era of mainframes. The data was essentially primitive and structured. Relational databases evolved in 1980s and 1990s. The era was of data intensive applications. The World Wide Web (WWW) and the Internet of Things (IOT) have led to an onslaught of structured, unstructured, and multimedia data. Refer Table 1.1.

	DATA GENERATIONAND STORAGE	**DATA UTILZATION**	**DATA DRIVEN**
COMPLEX AND UNSTRUCTURED			Structured data, Unstructured data, Multimedia data
COMPLEX AND RELATIONAL		Relational databases: Data-intensive applications	
PRIMITIVE AND STRUCTURED	Mainframes: Basic data storage		
	1970 and before	**Relational (1980 and 1990s)**	**2000 and beyond**

Table 1.1 The evolution of big data

1.4 Definition of Big Data:

Big data is high-velocity and high-variety information assets that demand cost effective, innovative forms of information processing for enhanced insight and decision making.

Big data refers to datasets whose size is typically beyond the storage capacity of and also complex for traditional database software tools

Big data is anything beyond the human & technical infrastructure needed to support storage, processing and analysis.

It is data that is big in volume, velocity and variety. Refer to figure 1.3

Variety: Data can be structured data, semi-structured data and unstructured data. Data stored in a database is an example of structured data.HTML data, XML data, email data,

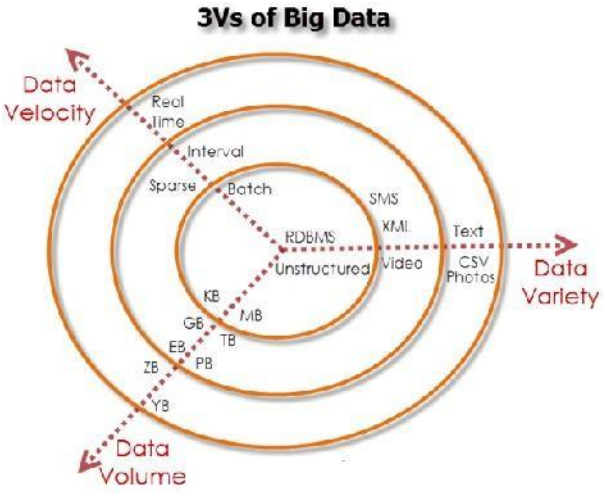


Figure 1.3 Data: Big in volume, variety and volume

CSV files are the examples of semi-structured data.Power point presentation, images, videos, researches, white papers, body of email etc are the examples of unstructured data.

Velocity: Velocity essentially refers tothe speed at which data is being created in real-time. We have moved from simple desktop applications like payroll application to real-time processing applications.

Volume: Volume can be in Terabytes or Petabytes or Zettabytes.

Gartner Glossary **Big data** is high-volume, high-velocity and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight and decision making.

For the sake of easy comprehension, we will look at the definition in three parts. Refer Figure 1.4.

Part I of the definition: "Big data is high-volume, high-velocity, and high-variety information assets" talks about voluminous data (humongous data) that may have great variety (a good mix of structured, semi-structured. and unstructured data) and will require a good speed/pace for storage, preparation, processing and analysis.

Part II of the definition: "cost effective, innovative forms of information processing" talks about embracing new techniques and technologies to capture (ingest), store, process, persist, integrate and visualize the high-volume, high-velocity, and high-variety data.
Part III of the definition: "enhanced insight and decision making" talks about deriving deeper, richer and meaningful insights and then using these insights to make faster and better decisions to gain business value and thus a competitive edge.

Data —> Information —> Actionable intelligence —> Better decisions —>Enhanced business value

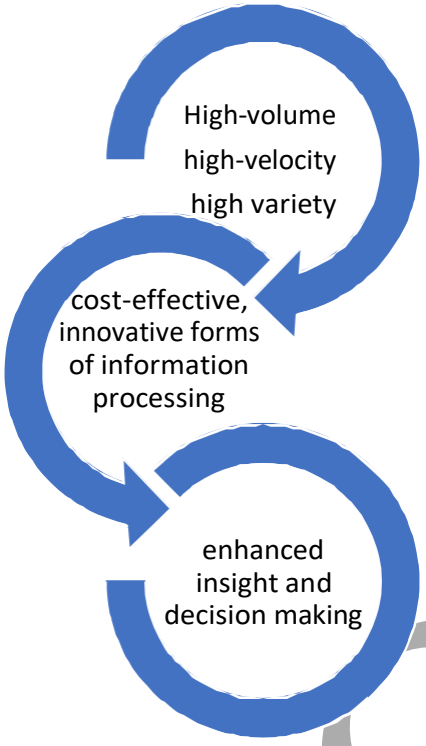


Figure 1.4 Definition of big data - Gartner

1.5 Challenges with big data

Refer figure 1.5. Following are a few challenges with big data:

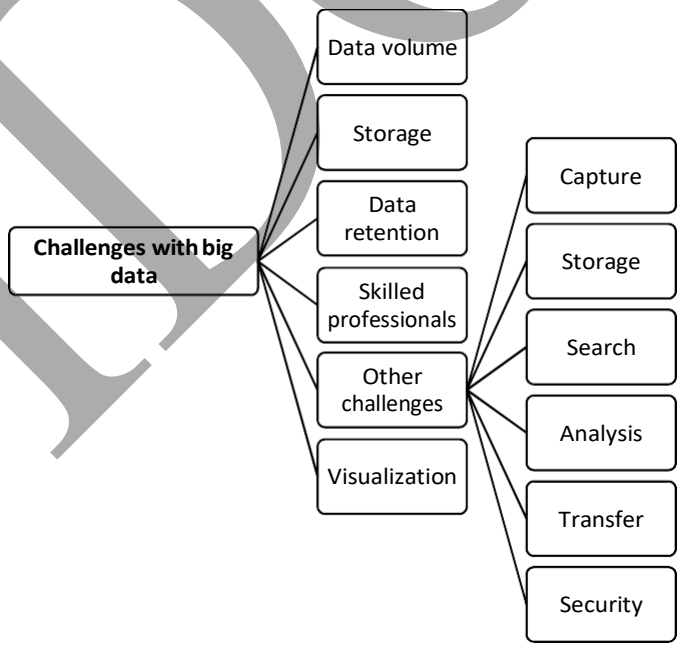


Figure 1.5. Challenges with big data

Data volume:Data today is growing at an exponential rate. This high tide of data will continue to rise continuously. The key questions are –

"will all this data be useful for analysis?",

"Do we work with all this data or subset of it?",

"How will we separate the knowledge from the noise?" etc

Storage: Cloud computingis the answer to managing infrastructure for big data as far as cost-efficiency, elasticity and easy upgrading / downgrading is concerned. This further complicates the decision to host big data solutions outside the enterprise.

Data retention: How long should one retain this data? Some data may require for log-term decision, but some data may quickly become irrelevant and obsolete.

Skilled professionals: In order to develop, manage and run those applications that generate insights, organizations need professionals who possess a high-level proficiency in data sciences.

Other challenges: Other challenges of big data are with respect to capture, storage, search, analysis, transfer and security of big data.

Visualization: Big data refers to datasets whose size is typically beyond the storage capacity of traditional database software tools. There is no explicit definition of how big the data set should be for it to be considered bigdata.Data visualization(computer graphics) is becoming popular as a separate discipline. There are very few data visualization experts.

1.6 Why Big data?

The more data we have for analysis, the greater will be the analytical accuracy and the greater would be the confidence in our decisions based on these analytical findings. The analytical accuracy will lead a greater positive impact in terms of enhancing operational efficiencies, reducing cost and time, and originating new products, new services, and optimizing existing services. Refer Figure 1.6.

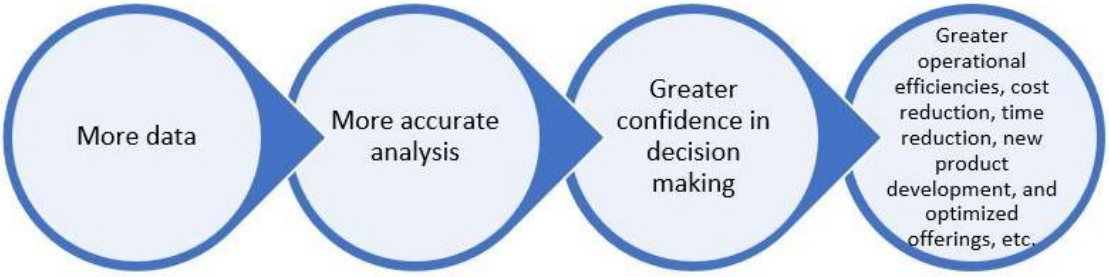


Figure 1.6: Why big data?

1.7 Data Warehouse Environment

Operational or transactional or day-to-day business data is gathered from Enterprise Resource Planning (ERP) systems, Customer Relationship Management (CRM), Legacy systems, and several third-party applications.

The data from these sources may differ in format.

This data is then integrated, cleaned up, transformed, and standardized through the process of Extraction, Transformation, and Loading (ETL).

The transformed data is then loaded into the enterprise data warehouse (available at the enterprise level) or data marts (available at the business unit/ functional unit or business process level).

Business intelligence and analytics tools are then used to enable decision making from the use of ad-hoc queries, SQL, enterprise dashboards, data mining, Online Analytical Processing etc. Refer Figure 1.7.

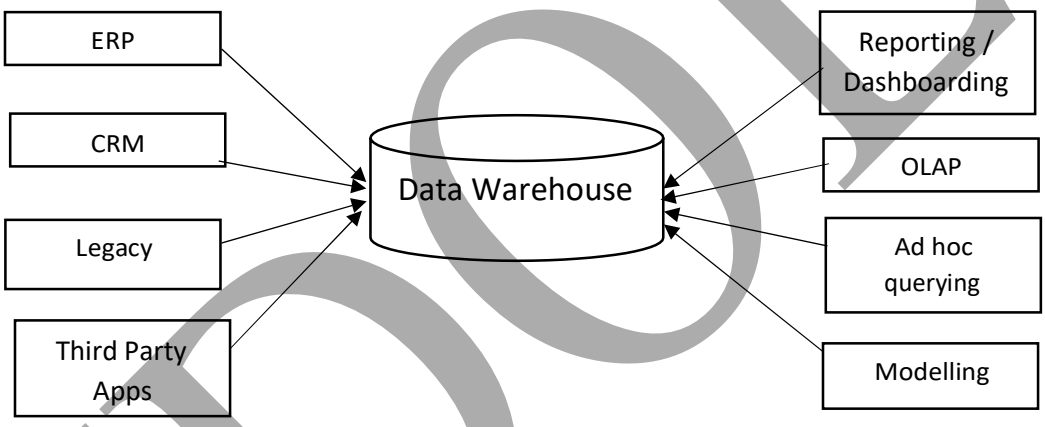


Figure 1.7: Data Warehouse Environment

1.8 Traditional Business Intelligence (Bi) Versus Big Data

Following are the differences that one encounters dealing with traditional Bl and big data.

In traditional BI environment, all the enterprise's data is housed in a central server whereas in a big data environment data resides in a distributed file system. The distributed file system scales by scaling in(decrease) or out(increase) horizontally as compared to typical database server that scales vertically.

In traditional BI, data is generally analysed in an offline mode whereas in big data, it is analysed in both real-time streaming as well as in offline mode.

Traditional Bl is about structured data and it is here that data is taken to processing functions (move data to code) whereas big data is about variety: Structured, semi-structured, and unstructured data and here the processing functions are taken to the data (move code to data).

1.9 State of the Practice in Analytics

Current business problems provide many opportunities for organizations to become more analytical and data driven, as shown in Table 1 ·2.

Business Driver	Examples
Optimize business operations	Sales, pricing, profitability, efficiency
Identify business risk	Customer churn, fraud, default
Predict new business opportunities	Upsell, cross-sell, best new customer prospects
Comply with laws or regulatory requirements	Anti-Money Laundering, Fair Lending,

TABLE 1-2 Business Drivers for Advanced Analytics

The first three examples do not represent new problems.

Organizations have been trying to reduce customer churn, increase sales, and cross-sell customers for many years.

What is new is the opportunity to fuse advanced analytical techniques with Big Data to produce more impactful analyses for these traditional problems.

The last example portrays emerging regulatory requirements.

Many compliance and regulatory laws have been in existence for decades, but additional requirements are added every year, which represent additional complexity and data requirements for organizations.

Laws related to anti-money laundering (AML) and fraud prevention require advanced analytical techniques to comply with and manage properly.

Different types of analytics:

1.9.1 BI Versus Data Science

1.9.2 Current Analytical Architecture (data flow)

1.9.3 Drivers of Big Data

1.9.4 Emerging Big Data Ecosystem and a New Approach to Analytics

1.9.1 BI Versus Data Science: Refer figure 1.8 for comparing BI with Data Science

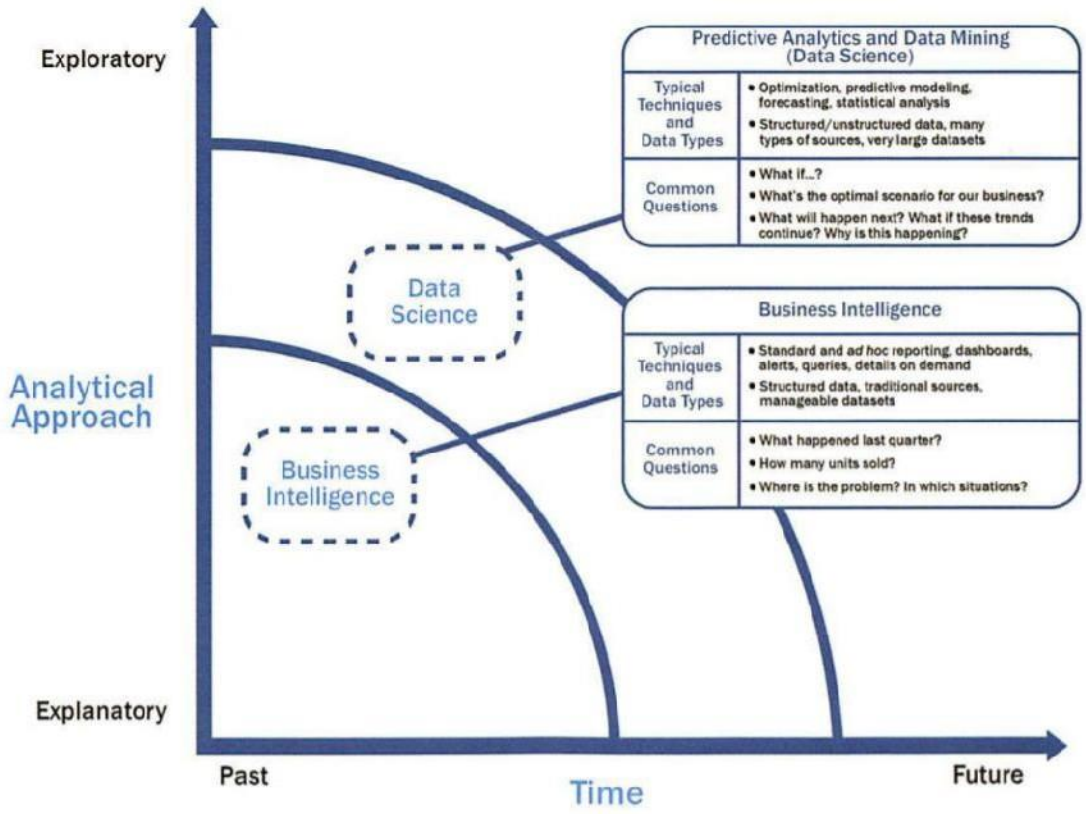


Figure 1.8 Comparing BI with Data Science

Tables – 1-3 and 1-4 explain the comparison between BI and Data Science.

Predictive Analytics and Data Mining (Data Science)	
Typical Techniques and Data Types	• Optimization. predictive modelling, forecasting. statistical analysis • Structured/unstructured data, many types of sources, very large datasets
Common Questions	• What if ... ? • What's the optimal scenario for our business? • What will happen next? What if these trends continue? Why is this happening?

Table 1-3: Data Science

Business Intelligence	
Typical Techniques and Data Types	• Standard and ad hoc reporting, dashboards, alerts, queries, details on demand • Structured data. traditional sources. manageable datasets
Common Questions	• What happened last quarter? • How many units sold? • Where is the problem? Hey in which situation?

Table 1-4: BI

1.9.2 Current Analytical Architecture: Figure 1.9 explains a typical analytical architecture.

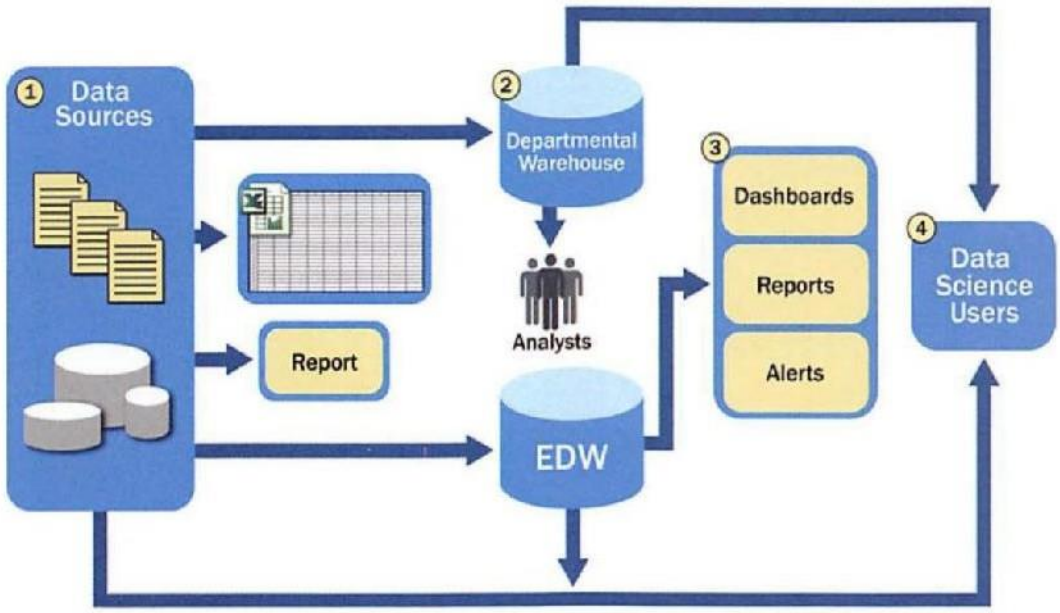


Figure 1.9 – Typical Analytical Architecture

1. For data sources to be loaded into the data warehouse, data needs to be well understood, structured and normalized with the appropriate data type definitions.
2. As a result of this level of control on the EDW(enterprise data warehouse-on server or on cloud), additional local systems may emerge in the form of departmental warehouses and local data marts that business users create to accommodate their need for flexible analysis. However, these local systems reside in isolation, often are not synchronized or integrated with other data stores and may not be backed up.
3. In the data warehouse, data is read by additional applications across the enterprise for Bl and reporting purposes.
4. At the end of this workflow, analysts get data from server. Because users generally are not allowed to run custom or intensive analytics on production databases, analysts create data extracts from the EDW to analyze data offline in R or other local analytical tools to store and process critical data, supporting enterprise applications and enabling corporate reporting activities.

Although reports and dashboards are still important for organizations, most traditional data architectures prevent data exploration and more sophisticated analysis.

1.9.3 Drivers of Big Data

As shown in Figure 1-10, in the 1990s the volume of information was often measured in terabytes. Most organizations analyzed structured data in rows and columns and used relational databases and data warehouses to manage large amount of enterprise information.

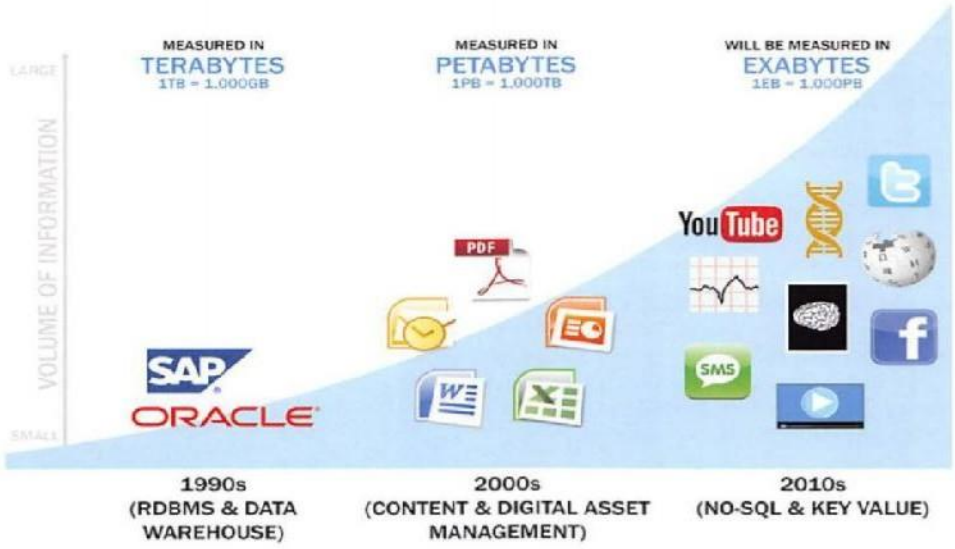


Figure 1.10 – Data Evolution and the Rise of Big Data Sources

The following decade (2000) saw different kinds of data sources-mainly productivity and publishing tools such as content management repositories and networked attached storage systems-to manage this kind of information, and the data began to increase in size and started to be measured at petabyte scales.

In the 2010s, the information that organizations try to manage has broadened to include many other kinds of data. In this era, everyone and everything is leaving a digital footprint. These applications, which generate data volumes that can be measured in exabyte scale, provide opportunities for new analytics and driving new value for organizations. The data now comes from multiple sources, like Medical information, Photos and video footage, Video surveillance, Mobile devices, Smart devices, Nontraditional IT devices etc.

1.9.4 Emerging Big Data Ecosystem and a New Approach to Analytics

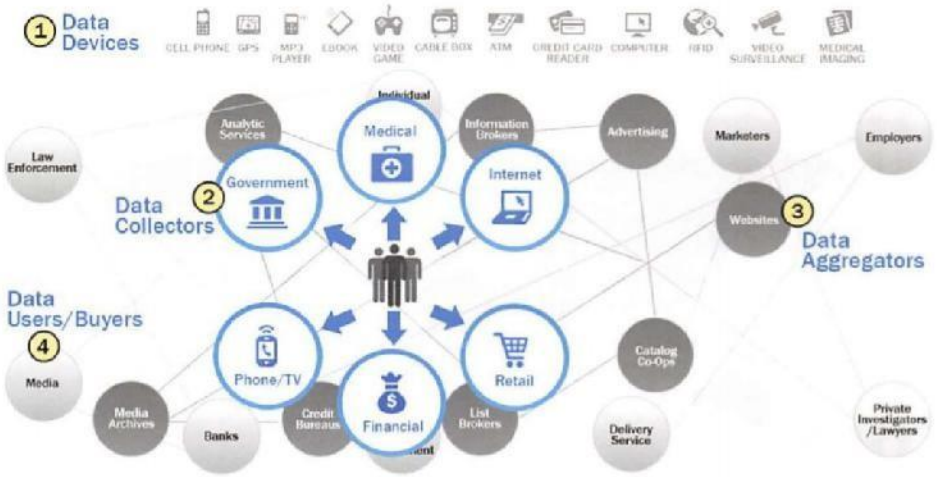


Figure 1.11 – Emerging Big Data Ecosystem

As the new ecosystem takes shape, there are four main groups of players within this interconnected web. These are shown in Figure 1-11.

1. Data devices and the "Sensornet" gather data from multiple locations and continuously generate new data about this data. For each gigabyte of new data created, an additional petabyte of data is created about that data.

   For example, consider someone playing an online video game through a PC, game console, or smartphone. In this case, the video game provider captures data about the skill and levels attained by the player. Intelligent systems monitor and log how and when the user plays the game. As a consequence, the game provider can fine-tune the difficulty of the game, suggest other related games that would most likely interest the user, and offer additional equipment and enhancements for the character based on the user's age, gender, and interests. This information may get stored locally or uploaded to the game provider's cloud to analyze the gaming habits and opportunities for upsell and cross-sell and identify typical profiles of specific kinds of users.

   Smartphones provide another rich source of data. In addition to messaging and basic phone usage, they store and transmit data about Internet usage, SMS usage, and real-time location. This metadata can be used for analyzing traffic patterns by scanning the density of smartphones in locations to track the speed of cars or the relative traffic congestion on busy roads. In this way, GPS devices in cars can give drivers real-time updates and offer alternative routes to avoid traffic delays.

   Retail shopping loyalty cards record not just the amount an individual spends, but the locations of stores that person visits, the kinds of products purchased, the stores where goods are purchased most often, and the combinations of products purchased together. Collecting this data provides insights into shopping and travel habits and the likelihood of successful advertisement targeting for certain types of retail promotions.

2. Data collectors include sample entities that collect data from the device and users.

   Data results from a cable TV provider tracking the shows a person watches, which TV channels someone will and will not pay for to watch on demand, and the prices someone is willing to pay for premium TV content

   Retail stores tracking the path a customer takes through their store while pushing a shopping cart with an RFID chip so they can gauge which products get the most foot traffic using geospatial data collected from the RFID chips

3. Data aggregators make sense of the data collected from the various entities from the "SensorNet" or the "Internet of Things." These organizations compile data from the devices and usage patterns collected by government agencies, retail stores and websites. ln turn, they can choose to transform and package the data as products to sell to list brokers, who may want to generate marketing lists of people who may be good targets for specific ad campaigns.
4. Data users / buyers: These groups directly benefit from the data collected and aggregated by others within the data value chain.

   Retail banks, acting as a data buyer, may want to know which customers have the highest likelihood to apply for a second mortgage or a home equity line of credit. To

provide input for this analysis, retail banks may purchase data from a data aggregator. This kind of data may include demographic information about people living in specific locations; people who appear to have a specific level of debt, yet still have solid credit scores (or other characteristics such as paying bills on time and having savings accounts) that can be used to infer credit worthiness; and those who are searching the web for information about paying off debts or doing home remodeling projects. Obtaining data from these various sources and aggregators will enable a more targeted marketing campaign, which would have been more challenging before Big Data due to the lack of information or high-performing technologies.

Using technologies such as Hadoop to perform natural language processing on unstructured, textual data from social media websites, users can gauge the reaction to events such as presidential campaigns. People may, for example, want to determine public sentiments toward a candidate by analyzing related blogs and online comments. Similarly, data users may want to track and prepare for natural disasters by identifying which areas a hurricane affects first and how it moves, based on which geographic areas are tweeting about it or discussing it via social media.

1.10 Key Roles for the New Big Data Ecosystem

Refer figure 1.12 for Key roles of the new big data ecosystems.

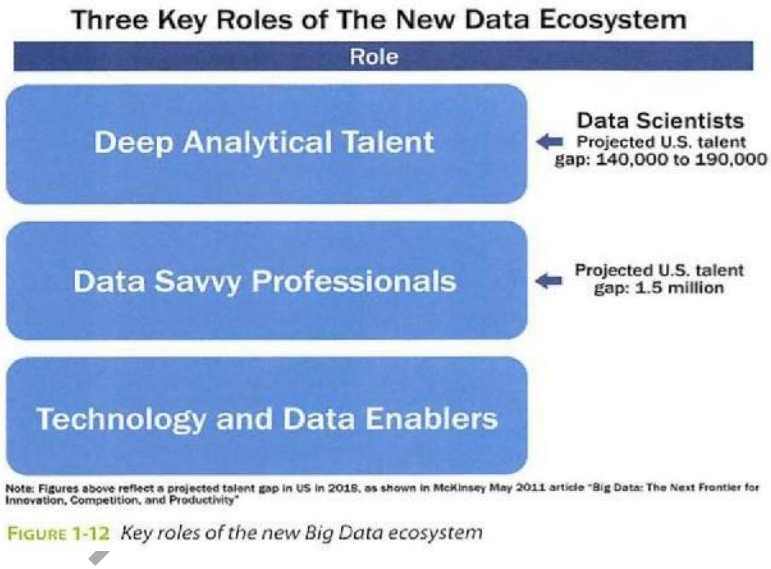


FIGURE 1-12 *Key roles of the new Big Data ecosystem*

Figure 1.12 – Key roles of the new big data ecosystems

1. Deep Analytical Talent is technically savvy, with strong analytical skills. Members possess a combination of skills to handle raw, unstructured data and to apply complex analytical techniques at massive scales.

   This group has advanced training in quantitative disciplines, such as mathematics, statistics, and machine learning. To do their jobs, members need access to a robust

analytic sandbox or workspace where they can perform large-scale analytical data experiments.

Examples of current professions fitting into this group include statisticians, economists, mathematicians, and the new role of the Data Scientist.

2. Data Savvy Professionals-has less technical depth but has a basic knowledge of statistics or machine learning and can define key questions that can be answered using advanced analytics.

   These people tend to have a base knowledge of working with data, or an appreciation for some of the work being performed by data scientists and others with deep analytical talent.

   Examples of data savvy professionals include financial analysts, market research analysts, life scientists, operations managers, and business and functional managers.

3. Technology and Data Enablers- This group represents people providing technical expertise to support analytical projects, such as provisioning and administrating analytical sandboxes, and managing large-scale data architectures that enable widespread analytics within companies and other organizations.

   This role requires skills related to computer engineering, programming, and database administration.

   These three groups must work together closely to solve complex Big Data challenges.

   Most organizations are familiar with people in the latter two groups mentioned, but the first group, Deep Analytical Talent, tends to be the newest role for most and the least understood.

   For simplicity, this discussion focuses on the emerging role of the Data Scientist. It describes the kinds of activities that role performs and provides a more detailed view of the skills needed to fulfill that role.

Activities of data scientist:

There are three recurring sets of activities that data scientists perform:

Reframe business challenges as analytics challenges. Specifically, this is a skill to diagnose business problems, consider the core of a given problem, and determine which kinds of analytical methods can be applied to solve it.

Design, implement, and deploy statistical models and data mining techniques on Big Data. This set of activities is mainly what people think about when they consider the role of the Data Scientist: namely, applying complex or advanced analytical methods to a variety of business problems using data.

Develop insights that lead to actionable recommendations. It is critical to note that applying advanced methods to data problems does not necessarily drive new business value. Instead, it is important to learn how to draw insights out of the data and communicate them effectively.

Profile of a data scientist:

Data scientists are generally thought of as having five main sets of skills and behavioral characteristics, as shown in Figure 1-13:

Quantitative skill: such as mathematics or statistics

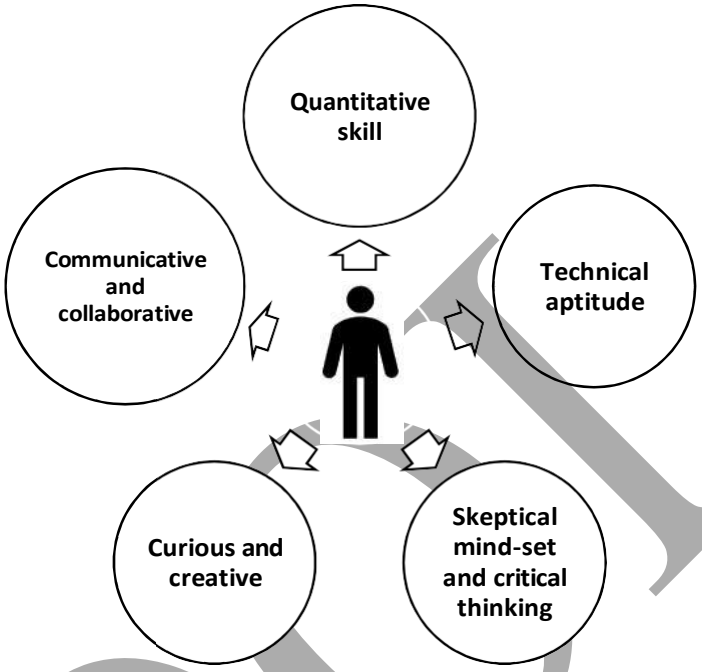


Figure 1.13 - Data scientist

Technical aptitude: namely, software engineering, machine learning, and programming skills

Skeptical mind-set and critical thinking: It is important that data scientists can examine their work critically rather than in a one-sided way.

Curious and creative: Data scientists are passionate about data and finding creative ways to solve problems and portray information.

Communicative and collaborative: Data scientists must be able to understand the business value in a clear way and collaboratively work with other groups, including project sponsors and key stakeholders.

Data scientists are generally comfortable using this blend of skills to acquire, manage, analyze, and visualize data and tell compelling stories about it.

Unit I

Chapter II -Big Data Analytics

What's in Store?

2.1 Introduction to big data analytics

2.2 Classification of Analytics

2.3 Challenges of Big Data

2.4 Importance of Big Data

2.5 Big Data Technologies

2.6 Data Science

2.7 Responsibilities

2.8 Soft state eventual consistency

2.9 Data Analytics Life Cycle

Big Data is creating significant new opportunities for organizations to derive new value and create competitive advantage from their most valuable asset: information. For businesses, Big Data helps drive efficiency, quality, and personalized products and services, producing improved levels of customer satisfaction and profit. For scientific efforts, Big Data analytics enable new avenues of investigation with potentially richer results and deeper insights than previously available. In many cases, Big Data analytics integrate structured and unstructured data with Realtime feeds and queries, opening new paths to innovation and insight.

2.1 Introduction to big data analytics

Big Data Analytics is...

1. Technology-enabled analytics: Quite a few data analytics and visualization tools are available in the market today from leading vendors such as IBM, Tableau, SAS, R Analytics, Statistical, World Programming Systems (WPS), etc. to help process and analyze your big data.
2. About gaining a meaningful, deeper, and richer insight into your business to steer it in the right direction. understanding the customer's demographics to cross-sell and up-sell to them, better leveraging the services of your vendors and suppliers, etc.

Author's experience: The other day I was pleasantly surprised to get a few recommendations via email from one of my frequently visited online retailers. They had recommended clothing line from my favorite brand and also the color suggested was one to my liking. How did they arrive at this? In the recent past, I had been buying clothing line of a particular brand and the color

preference was pastel shades. They had it stored in their database and pulled it out while making recommendations to me.

3. About a competitive edge over your competitors by enabling you with findings that allow quicker and better decision-making.
4. A tight handshake between three communities: IT, business users, and data scientists. Refer Figure 3.3.
5. Working with datasets whose volume and variety exceed the current storage and processing capabilities and infrastructure of your enterprise.

About moving code to data. This makes perfect sense as the program for distributed processing is tiny (just a few KBs) compared to the data (Terabytes or Petabytes today and likely to be Exabytes or Zettabytes in the near future).

2.2 Classification of Analytics

There are basically basically two schools of thought:

2.2.1 Those that classify analytics into basic, operationalized, advanced and Monetized.

2.2.2 Those that classify analytics into analytics 1.0, analytics 2.0, and analytics 3.0.

2.2.1 First School of Thought

It includes Basic analytics, Operationalized analytics, Advanced analytics and Monetized analytics.

Basic analytics: This primarily is slicing and dicing of data to help with basic business insights. This is about reporting on historical data, basic visualization, etc.

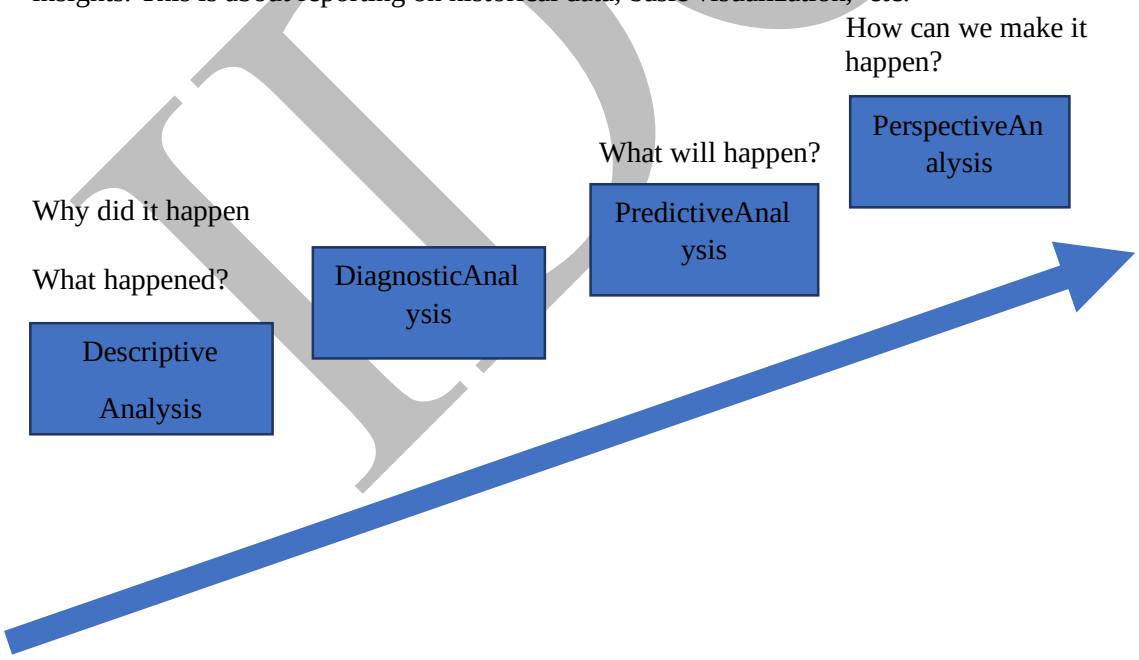


Figure 2.1 Analytics 1.0, 2.0 and 3.0

Operationalized analytics: It is operationalized analytics if it gets woven into the enterprises business processes.

Advanced analytics: This largely is about forecasting for the future by way of predictive and prescriptive modelling.

Monetized analytics: This is analytics in use to derive direct business revenue.

2.2.2 Second School of Thought

Let us take a closer look at analytics 1.0, analytics 2.0, and analytics 3.0. Refer Table 2.1. Figure 2.1 shows the subtle growth of analytics from Descriptive → Diagnostic → Predictive → Perspective analytics.

Analytics 1.0	**Analytics 2.0**	**Analytics 3.0**
Era: mid 1990s to 2009 Descriptive statistics (report on events,occurrences, etc. of the past)	**2005 to 2012** Descriptive statistics predictivestatistics (use data from the past to make predictions for the future)	**2012 to present** Descriptive + predictive + prescriptive statistics (use data from the past to make prophecies for the future and at the same time make recommendations to leverage the situation to one's advantage)
key questions asked: What happened? Why did it happen?	key questions asked: What happened? Why will it happen?	Key questions asked: What will happen? When will it happen? Why will it happen? What should be the action taken to take advantage of what will happen?
Data from legacy systems. ERP, CRM, and 3rd party applications.	Big data	A blend of big data and data from legacy systems, ERP, CRM, and 3rd party applications.
Small and structured data sources. Data stored in enterprise data warehouses or data marts.	Big data is being taken up seriously. Data is mainly unstructured, arriving at a much higher pace. This fast flow of data entailed that the influx of big volume data had to be stored and processed rapidly, often on massive parallel servers running Hadoop.	A blend of big data and traditional analytics to yield insights and offerings with speed and impact.
Data was internally sourced.	Data was often externally sourced.	Data is both being internally and externally sourced.
Relational databases	Database appliances, Hadoop clusters, SQL to Hadoop environments, etc.	In memory analytics, in database processing, agile analytical methods, machine learning techniques etc.

Table 2.1Analytics 1.0, 2.0 and 3.0

2.3 Challenges of Big Data

There are mainly seven challenges of big data: scale, security, schema, Continuous availability, Consistency, Partition tolerant and data quality.

Scale: Storage (RDBMS (Relational Database Management System) or NoSQL (Not only SQL)) is one major concern that needs to be addressed to handle the need for scaling rapidly and elastically. The need of the hour is a storage that can best withstand the attack of large volume, velocity and variety of big data. Should you scale vertically or should you scale horizontally?

Security: Most of the NoSQL big data platforms have poor security mechanisms (lack of proper authentication and authorization mechanisms) when it comes to safeguarding big data. A spot that cannot be ignored given that big data carries credit card information, personal information and other sensitive data.

schema: Rigid schemas have no place. We want the technology to be able to fit our big data and not the other way around. The need of the hour is dynamic schema. Static (pre-defined schemas) are obsolete.

Continuous availability: The big question here is how to provide 24/7 support because almost all RDBMS and NoSQL big data platforms have a certain amount of downtime built in.

Consistency: Should one opt for consistency or eventual consistency?

Partition tolerant: How to build partition tolerant systems that can take care of both hardware and software failures?

Data quality: How to maintain data quality- data accuracy, completeness, timeliness, etc.? Do we have appropriate metadata in place?

2.4Importance of Big Data

Let us study the various approaches to analysis of data and what it leads to.

Reactive-Business Intelligence: What does Business Intelligence (BI) help us with? It allows the businesses to make faster and better decisions by providing the right information to the right person at the right time in the right format. It is about analysis of the past or historical data and then displaying the findings of the analysis or reports in the form of enterprise dashboards, alerts, notifications, etc. It has support for both pre-specified reports as well as ad hoc querying.

Reactive - Big Data Analytics: Here the analysis is done on huge datasets but the approach is still reactive as it is still based on static data.

Proactive - Analytics: This is to support futuristic decision making by use of data mining predictive modelling, text mining, and statistical analysis on. This analysis is not on big data as it still the traditional database management practices on big data and therefore has severe limitations on the storage capacity and the processing capability.

Proactive - Big Data Analytics: This is filtering through terabytes, petabytes, exabytes of information to filter out the relevant data to analyze. This also includes high performance analytics to gain rapid insights from big data and the ability to solve complex problems using more data.

2.5 Big Data Technologies

Following are the requirements of technologies to meet challenges of big data:

The first requirement is of cheap and ample storage.

We need faster processors to help with quicker processing of big data.

Affordable open source distributed big data platforms, such as Hadoop.

Parallel processing, clustering, virtualization, large grid environments (to distribute processing to a number of machines), high connectivity, and high throughputs(rate at which something is processed).

Cloud computing and other flexible resource allocation arrangements.

2.6 Data Science

Data science is the science of extracting knowledge from data. In other words, it is a science of drawing out hidden patterns amongst data using statistical and mathematical techniques.

It employs techniques and theories drawn from many fields from the broad areas of mathematics, statistics, information technology including machine learning, data engineering, probability models, statistical learning, pattern recognition and learning, etc.

Data Scientist works on massive datasets for weather predictions, oil drillings, earthquake prediction, financial frauds, terrorist network and activities, global economic impacts, sensor logs, social media analytics, customer churn, collaborative filtering(prediction about interest on users), regression analysis, etc. Data science is multi-disciplinary. Refer to Figure 2.2.

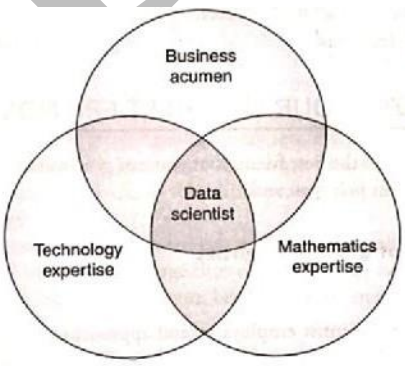


Figure 2.2 Data Scientist

2.6.1 Business Acumen(expertise) Skills

A data scientist should have following ability to play the role of data scientist.

- Understanding of domain
- Business strategy
- Problem solving
- Communication
- Presentation
- Keenness

2.6.2 Technology Expertise

Following skills required as far as technical expertise is concerned.

- Good database knowledge such as RDBMS.
- Good NoSQL database knowledge such as MongoDB, Cassandra, HBase, etc.
- Programming languages such as Java. Python, C++, etc.
- Open-source tools such as Hadoop.
- Data warehousing.
- Data mining
- Visualization such as Tableau, Flare, Google visualization APIs, etc.

2.6.3 Mathematics Expertise

The following are the key skills that a data scientist will have to have to comprehend data, interpret it and analyze.

- Mathematics.
- Statistics.
- Artificial Intelligence (AI).
- Algorithms.
- Machine learning.
- Pattern recognition.
- Natural Language Processing.
- To sum it up, the data science process is
- Collecting raw data from multiple different data sources.
- Processing the data.
- Integrating the data and preparing clean datasets.
- Engaging in explorative data analysis using model and algorithms.
- Preparing presentations using data visualizations.
- Communicating the findings to all stakeholders.
- Making faster and better decisions.

2.7 Responsibilities

Refer figure 2.3 to understand the responsibilities of a data scientist.

Data Management: A data scientist employs several approaches to develop the relevant datasets for analysis. Raw data is just "RAW", unsuitable for analysis. The data scientist

works on it to prepare to reflect the relationships and contexts. This data then becomes useful for processing and further analysis.

Analytical Techniques: Depending on the business questions which we are trying to find answers to and the type of data available at hand, the data scientist employs a blend of analytical techniques to develop models and algorithms to understand the data, interpret relationships, spot trends, and reveal patterns.

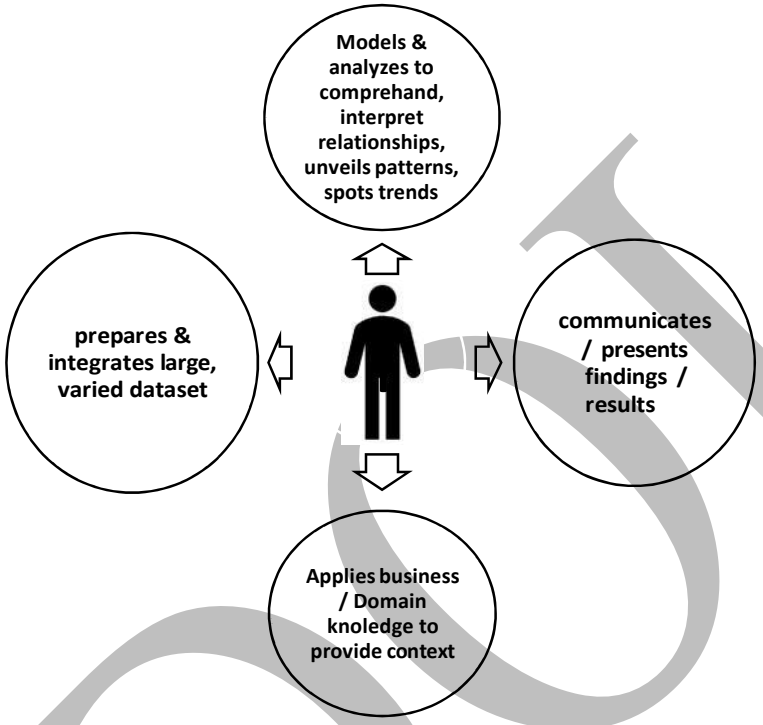


Figure 2.3 – data scientist: your new best friend!!!

Business Analysis: A data scientist is a business analyst who distinguishes cool facts from insights and is able to apply his business expertise and domain knowledge to see the results in the business context.

Communicator: He is a good presenter and communicator who is able to communicate the results of his findings in a language that is understood by the different business stakeholders.

2.8 Soft State Eventual Consistency

ACID property in RDBMS:

Atomicity: Either the task (or all tasks) within a transaction are performed or none of them are. This is the all-or-none principle. If one element of a transaction fails the entire transaction fails.

Consistency: The transaction must meet all protocols or rules defined by the system at all times. The transaction does not violate those protocols and the database must remain in a

consistent state at the beginning and end of a transaction; there are never any half-completed transactions.

Isolation: No transaction has access to any other transaction that is in an intermediate or unfinished state. Thus, each transaction is independent unto itself. This is required for both performance and consistency of transactions within a database.

Durability: Once the transaction is complete, it will persist as complete and cannot be undone; it will survive system failure, power loss and other types of system breakdowns.

BASE (Basically Available, Soft state, Eventual consistency). In a system where BASE is the prime requirement for reliability, the activity/potential (p) of the data (H) changes; it *essentially* slows down.

Basically Available: This constraint states that the system does guarantee the availability of the data as regards CAP Theorem; there will be a response to any request. But, that response could still be 'failure' to obtain the requested data or the data may be in an inconsistent or changing state, much like waiting for a check to clear in your bank account.

Eventual consistency: The system will *eventually* become consistent once it stops receiving input. The data will propagate to everywhere it should sooner or later, but the system will continue to receive input and is not checking the consistency of every transaction before it moves onto the next one. Werner Vogel's article "Eventually Consistent – Revisited" covers this topic is much greater detail.

Soft state: The state of the system could change over time, so even during times without input there may be changes going on due to 'eventual consistency,' thus the state of the system is always 'soft.'

2.9 Data Analytics Life Cycle

Here is a brief overview of the main phases of the Data Analytics Lifecycle:

Phase 1- Discovery: In Phase 1, the team learns the business domain, including relevant history such as whether the organization or business unit has attempted similar projects in the past from which they can learn. The team assesses the resources available to support the project in terms of people, technology, time and data. Important activities in this phase include framing the business problem as an analytics challenge that can be addressed in subsequent phases and formulating initial hypotheses (IHs) to test and begin learning the data.

Phase 2- Data preparation: Phase 2 requires the presence of an analytic sandbox, in which the team can work with data and perform analytics for the duration of the project. The team needs to execute extract, load, and transform (ELT) or extract, transform and load (ETL) to get data into the sandbox. The ELT and ETL are sometimes abbreviated as ETLT. Data should be transformed in the ETLT process so the team can work with it and analyze it. In

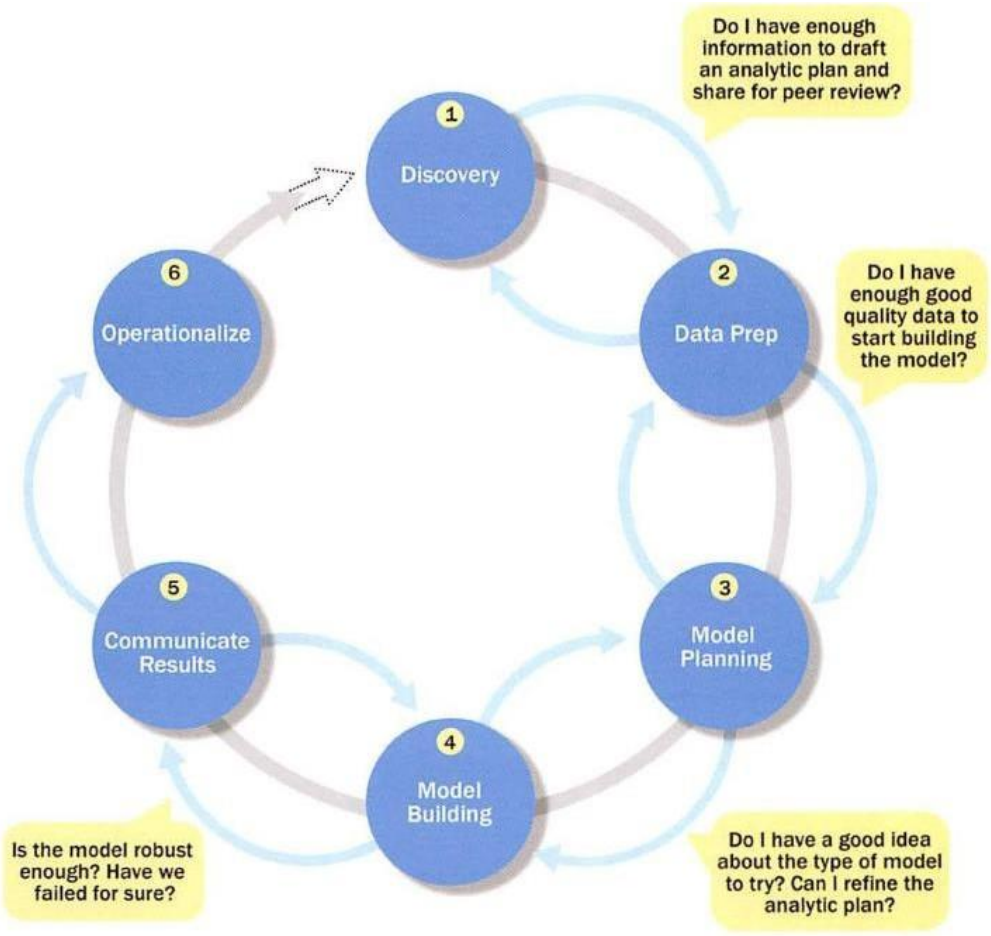


Figure 2.4 - Overview of Data Analytical Lifecycle

this phase, the team also needs to familiarize itself with the data thoroughly and take steps to condition the data.

Phase 3-Model planning: Phase 3 is model planning, where the team determines the methods, techniques and workflow it intends to follow for the subsequent model building phase. The team explores the data to learn about the relationships between variables and subsequently selects key variables and the most suitable models.

Phase 4-Model building: In Phase 4, the team develops data sets for testing, training, and production purposes. In addition, in this phase the team builds and executes models based on the work done in the model planning phase. The team also considers whether its existing tools will suffice for running the models, or if it will need a more robust environment for executing models and workflows (for example, fast hardware and parallel processing, if applicable).

Phase 5-Communicate results: In Phase 5, the team, in collaboration with major stakeholders, determines if the results of the project are a success or a failure based on the criteria developed in Phase 1. The team should identify key findings, quantify the business value, and develop a narrative to summarize and convey findings to stakeholders.

Phase 6-0perationalize: In Phase 6, the team delivers final reports, briefings, code and technical documents. In addition, the team may run a pilot project to implement the models in a production environment.

Unit II
Chapter I

Objectives:

To Study and Understand the following concept

- Analytical Theory and Methods:
- Clustering and Associated Algorithms
- Association Rules
- Apriori Algorithm
- Candidate Rules
- Applications of Association Rules
- Validation and Testing
- Diagnostics

3.1-Overview of Clustering

In general, clustering is the use of *unsupervised* techniques for grouping similar objects. In machine learning, unsupervised refers to the problem of finding hidden structure within unlabeled data. Clustering techniques are unsupervised in the sense that the data scientist does not determine, in advance, the labels to apply to the clusters. The structure of the data describes the objects of interest and determines how best to group the objects.

Clustering is a method often used for exploratory analysis of the data. In clustering, there are no predictions made. Rather, clustering methods find the similarities between objects according to the object attributes and group the similar objects into clusters. Clustering techniques are utilized in marketing, economics, and various branches of science. A popular clustering method is k-means.

3.2-K-means

Given a collection of objects each with n measurable attributes, *k-means* is an analytical technique that, for a chosen value of k, identifies k clusters of objects based on the objects' proximity to the center of the k groups. The center is determined as the arithmetic average (mean) of each cluster's n-dimensional vector of attributes. Below figure illustrates three dusters of objects with two attributes. Each object in the dataset is represented by a small dot color-coded to the closest large dot, the mean of the cluster.

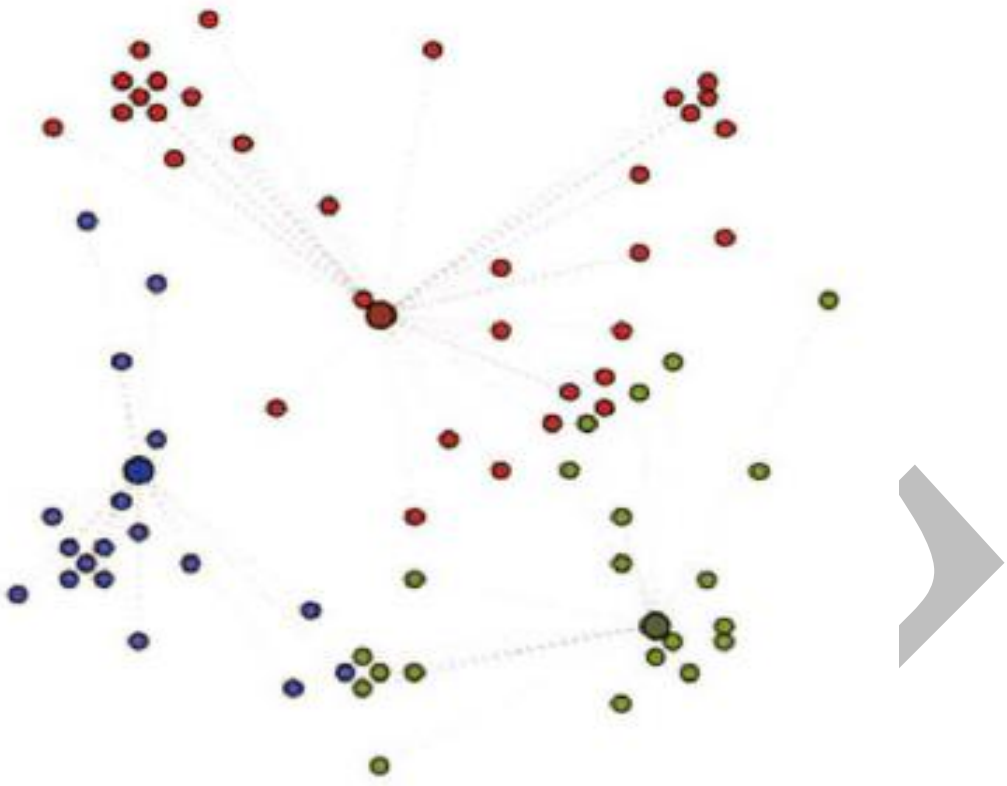


Figure 3.1 Possible K-means clusters for k=3

3.2.1-Use Cases

Clustering is often used as a lead-in to classification. Once the clusters are identified, labels can be applied to each cluster to classify each group based on its characteristics. Some specific applications of k-means are image processing, medical and customer segmentation.

Image Processing

Video is one example of the growing volumes of unstructured data being collected. Within each frame of a video, k-means analysis can be used to identify objects in the video. For each frame, the task is to determine which pixels are most similar to each other. The attributes of each pixel can include brightness, color, and location, the x and y coordinates in the frame. With security video images, for example, successive frames are examined to identify any changes to the clusters. These newly identified dusters may indicate unauthorized access to a facility.

Medical

Patient attributes such as age, height, weight, systolic and diastolic blood pressures, cholesterol level, and other attributes can identify naturally occurring clusters. These dusters could be used to target individuals for specific preventive measures or clinical trial participation. Clustering, in general, is useful in biology for the classification of plants and animals as well as in the field of human genetics.

Customer Segmentation

Marketing and sales groups use k-means to better identify customers who have similar behaviors and spending patterns. For example, a wireless provider may look at the following customer attributes: monthly bill, number of text messages, data volume consumed, minutes used during various daily periods, and years as a customer. The wireless company could then look at the naturally occurring clusters and consider tactics

to increase sales or reduce the customer *churn rate,* the proportion of customers who end their relationship with a particular company.

3.2.2-Overview of the Method

To illustrate the method to find k clusters from a collection of M objects with n attributes, the two-dimensional case (n = 2) is examined. It is much easier to visualize the k-means method in two dimensions.

Because each object in this example has two attributes, it is useful to consider each object corresponding to the point (x_i, y_i), where x and y denote the two attributes and i = 1, 2 ... M. For a given cluster of m points (m ≤ M), the point that corresponds to the cluster's mean is called a *centroid.*

The k-means algorithm to find k dusters can be described in the following four steps.

1. Choose the value of k and the k initial guesses for the centroids.

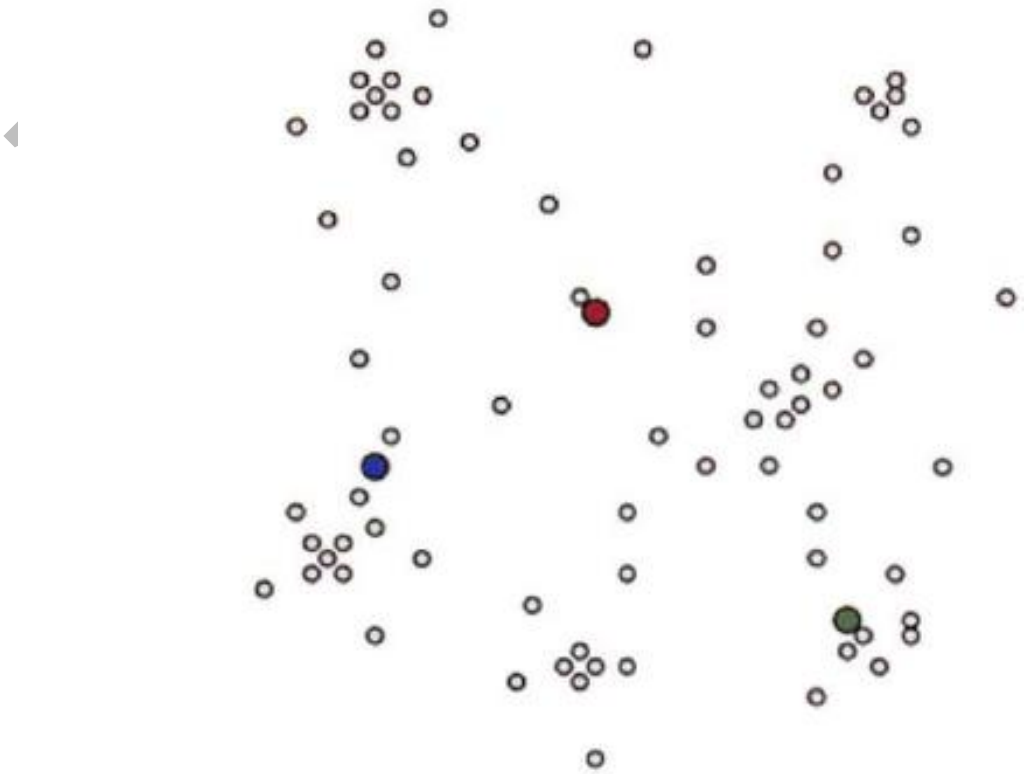
   In this example, k = 3, and the initial centroids are indicated by the points shaded in red, green, and blue in figure 2.2.

2. Compute the distance from each data point (x_i ,y_i) to each centroid. Assign each point to the closest centroid. This association defines the first k dusters.

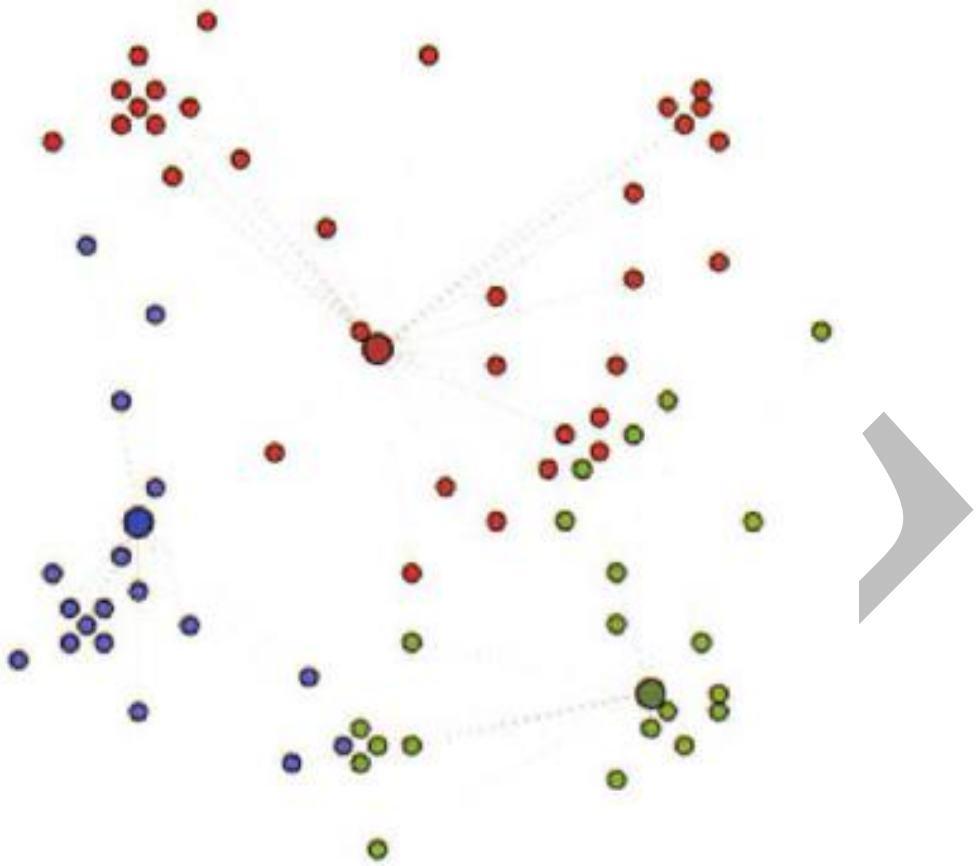
   In two dimensions, the distance, *d,* between any two points, (x_1,y_1) and (x_2,y_2), in the Cartesian plane is typically expressed by using the Euclidean distance measure provided in Equation.

   $$d= \sqrt{(x1 - y1)^2 + (x2 - y2)^2}$$

   In figure 3.2, the points closest to a centroid are shaded the corresponding color.



3.2-Initial starting points for the centroids

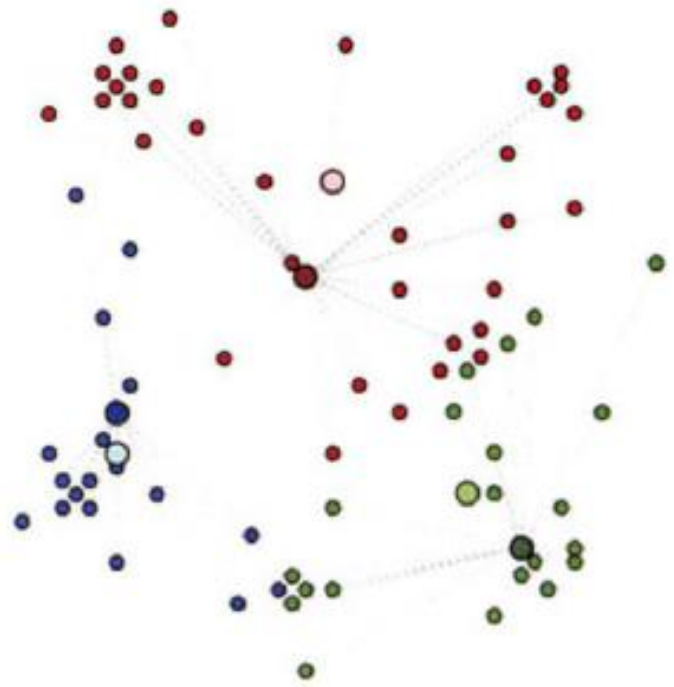


3.3-Points are assigned to the closest centroid

3. Compute the centroid, the center of mass, of each newly defined cluster from Step *2*.
   In Figure 2-4, the computed centroids in Step 3 are the lightly shaded points of the corresponding color. In two dimensions, the centroid (x_c,y_c) of the m points in a k-means duster is calculated as follows in Equation.

$$(x_c, y_c) = \left(\frac{\sum_{i=1}^{m} x_i}{m}, \frac{\sum_{i=1}^{m} y_i}{m} \right)$$

   Thus, (x_c,y_c) is the ordered pair of the arithmetic means of the coordinates of the m points in the cluster. In this step, a centroid is computed for each of the k clusters.



2.4-Compute the mean of each cluster

4. Repeat Steps 2 and 3 until the algorithm converges to an answer

    a. Assign each point to the closest centroid computed in Step 3.
    b. Compute the centroid of newly defined clusters.
    c. Repeat until the algorithm reaches the final answer.

3.2.3-Determining the Number of Clusters

In k-means, k clusters can be identified in a given dataset, but what value of k should be selected? The value of k can be chosen based on a reasonable guess or some predefined requirement. However, even then, it would be good to know how much better or worse having k clusters versus k-1 or k+1 cluster would be in explaining the structure of the data. Next, a heuristic using the Within Sum of Squares (WSS) metric is examined to determine a reasonably optimal value of k. Using the distance function, WSS is defined as shown below.

$$WSS = \sum_{i=1}^{M} d(p_i, q^{(i)})^2 = \sum_{i=1}^{M} \sum_{j=1}^{n} \left(p_{ij} - q_j^{(i)}\right)^2$$

In other words, WSS is the sum of the squares of the distances between each data point and the closest centroid. The term $\boldsymbol{q^{(i)}}$ indicates the closest centroid that is associated with the i^{th} point. If the points are relatively close to their respective centroids, the WSS is relatively small. Thus, if k +1 clusters do not greatly reduce the value of WSS from the case with only k clusters, there may be little benefit to adding another cluster.

3.2.4-Diagnostics

The heuristic using WSS can provide at least several possible k values to consider. When the number of attributes

is relatively small, a common approach to further refine the choice of k is to plot the data to determine how distinct the identified clusters are from each other. In general, the following questions should be considered.

- Are the dusters well separated from each other?
- Do any of the dusters have only a few points?
- Do any of the centroids appear to be too close to each other?

In the first case, ideally the plot would look like the one shown in below figure, when n = 2.

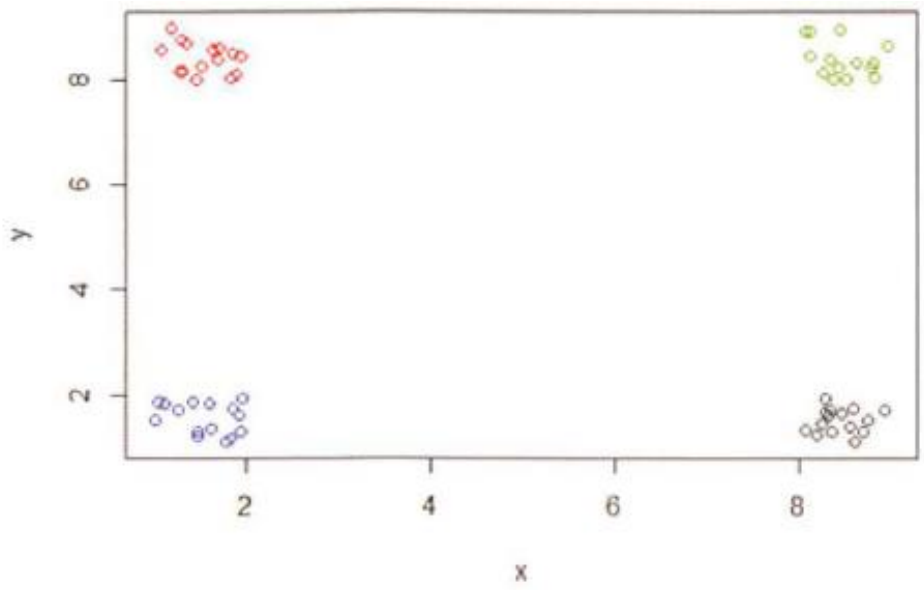


Figure: 3.4 Example of distinct clusters

The clusters are well defined, with considerable space between the four identified clusters. However, in other cases, such as in below figure, the clusters may be close to each other, and the distinction may not be so obvious.

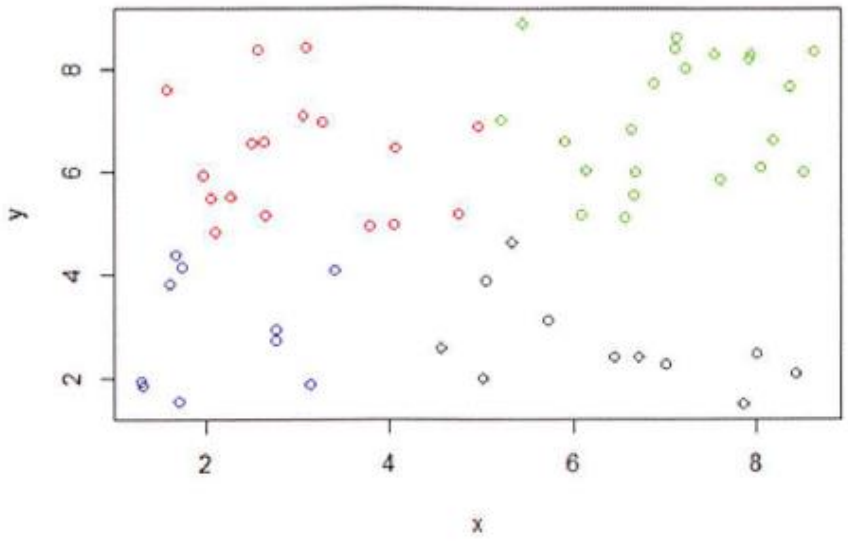


Figure 3.5 Example of less obvious cluster

3.2.5-Reasons to Choose and Cautions

K-means is a simple and straightforward method for defining clusters. Once clusters and their associated centroids are identified, it is easy to assign new objects (for example, new customers) to a cluster based on the object's distance from the closest centroid. Because the method is unsupervised, using k-means helps to eliminate subjectivity from the analysis.

Although k-means is considered an unsupervised method, there are still several decisions that the practitioner must make:

a. What object attributes should be included in the analysis?
b. What unit of measure (for example, miles or kilometers) should be used for each attribute?
c. Do the attributes need to be rescaled so that one attribute does not have a disproportionate effect on the results?
d. What other considerations might apply?

a-Object Attributes

Regarding which object attributes (for example, age and income) to use in the analysis, it is important to understand what attributes will be known at the time a new object will be assigned to a cluster. For example, information on existing customers' satisfaction or purchase frequency may be available, but such information may not be available for potential customers.

The Data Scientist may have a choice of a dozen or more attributes to use in the clustering analysis. Whenever possible and based on the data, it is best to reduce the number of attributes to the extent possible. Too many attributes can minimize the impact of the most important variables. Also, the use of several similar attributes can place too much importance on one type of attribute. For example, if five attributes related to personal wealth are included in a clustering analysis, the wealth attributes dominate the analysis and possibly mask the importance of other attributes, such as age.

When dealing with the problem of too many attributes, one useful approach is to identify any highly correlated attributes and use only one or two of the correlated attributes in the clustering analysis.

Another option to reduce the number of attributes is to combine several attributes into one measure. For example, instead of using two attribute variables, one for Debt and one for Assets, a Debt to Asset ratio could be used, This option also addresses the problem when the magnitude of an attribute is not of real interest, but the relative magnitude is a more important measure.

b-Units of Measure

From a computational perspective, the k-means algorithm is somewhat indifferent to the units of measure for a given attribute (for example, meters or centimeters for a patient's height). However, the algorithm will identify different clusters depending on the choice of the units of measure.

For example, suppose that k-means is used to cluster patients based on age in years and height in centimeters. For k=2, below figure illustrates the two clusters that would be determined for a given dataset.

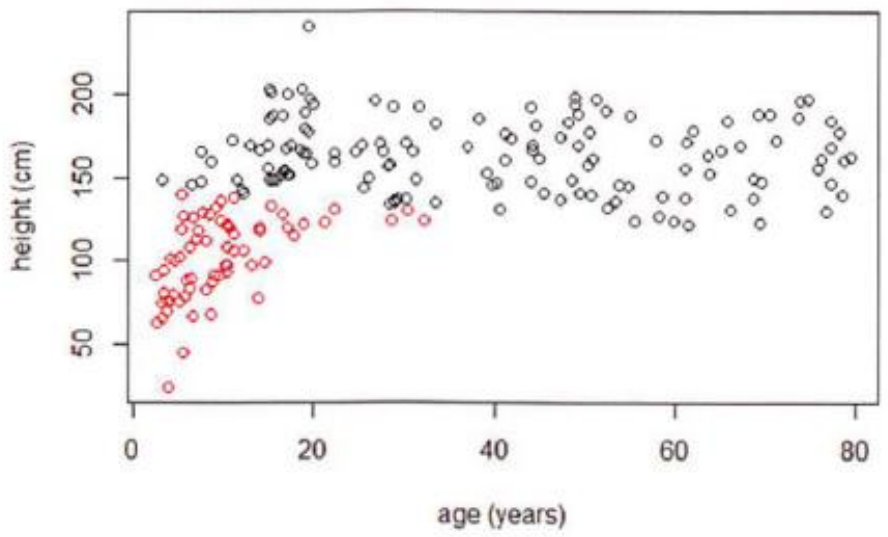


Figure 3.6. Cluster with height expressed in centimeters

But if the height was rescaled from centimeters to meters by dividing by 100, the resulting dusters would be slightly different, as illustrated in below Figure.

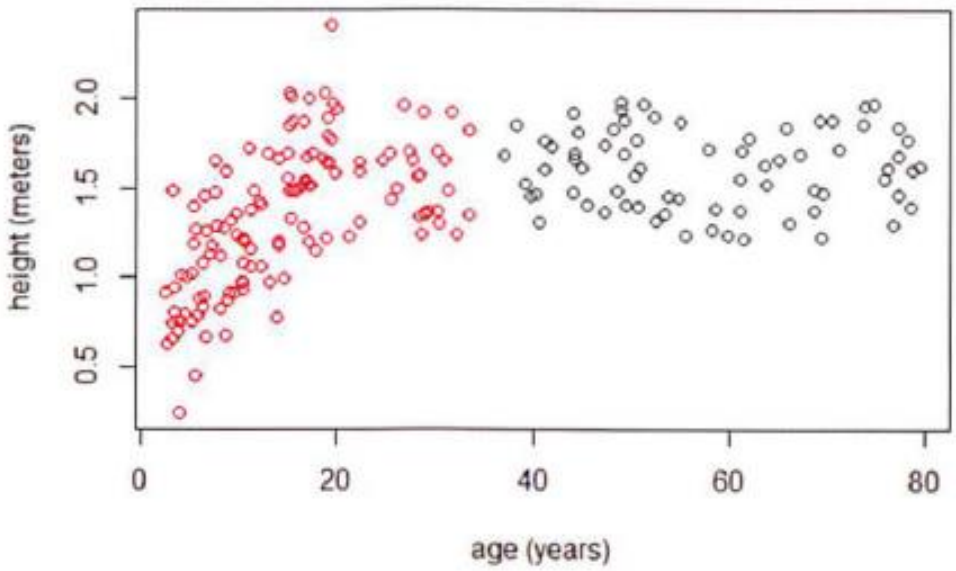


Figure 3.7. Cluster with height expressed in meters

c-Rescaling

Attributes that are expressed in dollars are common in clustering analyses and can differ in magnitude from the other attributes. For example, if personal income is expressed in dollars and age is expressed in years, the income attribute, often exceeding 510,000, can easily dominate the distance calculation with ages typically less than 100 years.

Although some adjustments could be made by expressing the income in thousands of dollars (for example, 10 for 510,000), a more straightforward method is to divide each attribute by the attribute's standard deviation. The resulting attributes will each have a standard deviation equal to 1 and will be without units.

Returning to the age and height example, the standard deviations are 23.1 years and 36.4 cm, respectively. Dividing each attribute value by the appropriate standard deviation and performing the k-means analysis yields the result shown in Figure 3.8.

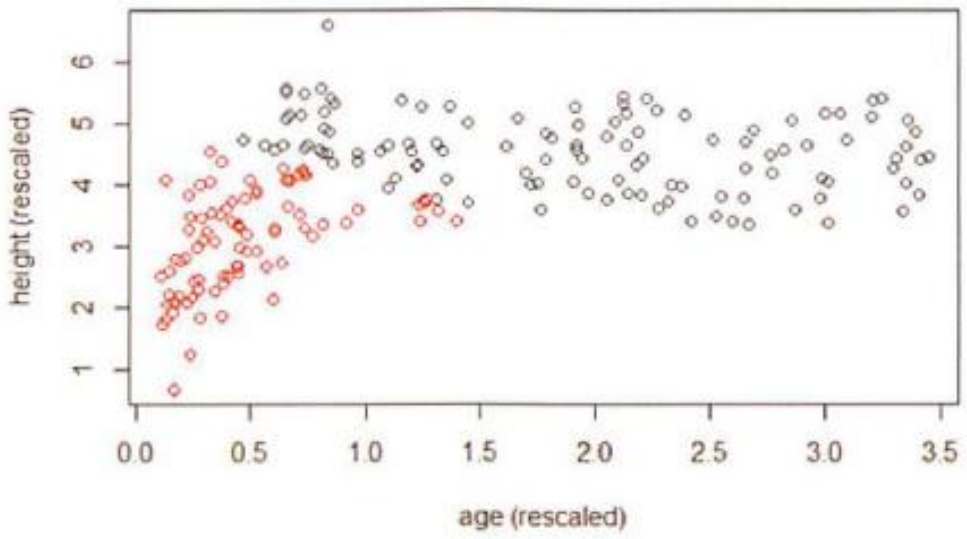


Figure 3.8.Cluster with rescaled attributes

In many statistical analyses, it is common to transform typically skewed data, such as income, with long tails by taking the logarithm of the data. Such transformation can also be applied in k-means, but the Data Scientist needs to be aware of what effect this transformation will have.

d-Additional Considerations

The k-means algorithm is sensitive to the starting positions of the initial centroid. Thus, it is important to rerun the k-means analysis several times for a particular value of k to ensure the cluster results provide the overall minimum WSS. As we know, this task is accomplished in R by using the nstart option in the kmeans () function call.

K-means clustering is applicable to objects that can be described by attributes that are numerical with a meaningful distance measure. Interval and ratio attribute types can certainly be used. However, k-means does not handle categorical variables well. For example, suppose a clustering analysis is to be conducted on new car sales. Among other attributes, such as the sale price, the color of the car is considered important. Although one could assign numerical values to the color, such as red = 1, yellow = 2, and green = 3, it is not useful to consider that yellow is as close to red as yellow is to green from a clustering perspective. In such cases, it may be necessary to use an alternative clustering methodology.

3.3.Association Rules

An unsupervised learning method called association rules. This is a descriptive, not predictive, method often used to discover interesting relationships hidden in a large dataset. The disclosed relationships can be represented as rules or frequent item sets. Association rules are commonly used for mining transactions in databases.

Here are some possible questions that association rules can answer:

- Which products tend to be purchased together?
- Of those customers who are similar to this person, what products do they tend to buy?
- Of those customers who have purchased this product, what other similar products do they tend to view or purchase?

3.3.1- Overview

Below figure shows the general logic behind association rules. Given a large collection of transactions (depicted as three stacks of receipts in the figure), in which each transaction consists of one or more items, association rules go through the items being purchased to see what items are frequently bought together and to discover a list of rules that describe the purchasing behavior. The goal with association rules is to discover interesting relationships among the items. The relationships that are interesting depend both on the business context and the nature of the algorithm being used for the discovery.

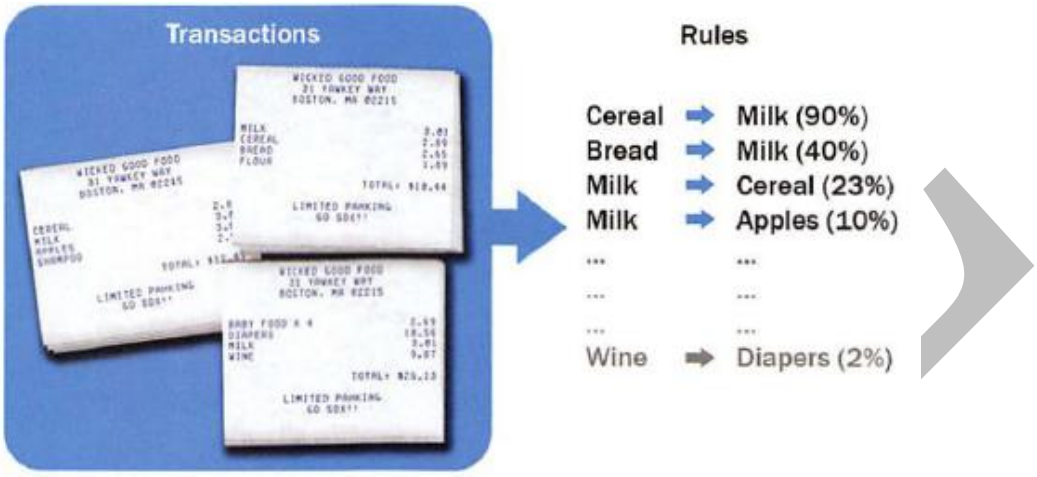


Figure 3.9 The general logic behind association rules

Each of the uncovered rules is in the form X —> Y, meaning that when item X is observed, item Y is also observed. In this case, the left-hand side (LHS) of the rule is X, and the right-hand side (RH5) of the rule is Y.

Using association rules, patterns can be discovered from the data that allow the association rule algorithms to disclose rules of related product purchases. The uncovered rules are listed on the right side of Figure. The first three rules suggest that when cereal is purchased, 90% of the time milk is purchased also. When bread is purchased, 40% of the time milk is purchased also. When milk is purchased, 23% of the time cereal is also purchased.

In the example of a retail store, association rules are used over transactions that consist of one or more items. In fact, because of their popularity in mining customer transactions, association rules are sometimes referred to as ***market basket analysis.*** Each transaction can be viewed as the shopping basket of a customer that contains one or more items. This is also known as an itemset. The term ***itemset*** refers to a collection of items or individual entities that contain some kind of relationship. This could be a set of retail items purchased together in one transaction, a set of hyperlinks clicked on by one user in a single session, or a set of tasks done in one day. An itemset containing ***k*** items is called a ***k-itemset*** denoted by {item1,item 2**,** . . . item k}**.**

3.3.2-Apriori Algorithm

The Apriori algorithm takes a bottom-up iterative approach to uncovering the frequent itemsets by first determining all the possible items (or 1-itemsets, for example {bread}, ***{eggs}, {milk},*** …) and then identifying which among them are frequent.

Assuming the minimum support threshold (or the minimum support criterion) is set at 0.5, the algorithm

identifies and retains those itemsets that appear in at least 50% of all transactions and discards (or "prunes away") the itemsets that have a support less than 0.5 or appear in fewer than 50% of the transactions.

In the next iteration of the Apriori algorithm, the identified frequent 1-itemsets are paired into 2-itemsets (for example, {bread, ***eggs***}, {bread, ***milk}, {eggs, milk},***…) and again evaluated to identify the frequent 2-itemsets among them.

At each iteration, the algorithm checks whether the support criterion can be met; if it can, the algorithm grows the itemset, repeating the process until it runs out of support or until the itemsets reach a predefined length. Let variable C_k be the set of candidate k-itemsets and variable L_k be the set of k-itemsets that satisfy the minimum support. Given a transaction database D, a minimum support threshold δ, and an optional parameter N indicating the maximum length an itemset could reach, Apriori iteratively computes frequent itemsets L_{k+1}, based on L_k.

```
1  Apriori (D, δ, N)
2    k ← 1
3    L_k ← {1-itemsets that satisfy minimum support δ}
4    while L_k ≠ ∅
5      if ∄N ∨ (∃N ∧ k < N)
6        C_{k+1} ← candidate itemsets generated from L_k
7        for each transaction t in database D do
8          increment the counts of C_{k+1} contained in t
9        L_{k+1} ← candidates in C_{k+1} that satisfy minimum support δ
10       k ← k + 1
11   return ⋃_k L_k
```

3.3.3-Evaluation of Candidate Rules

Frequent itemsets from the previous section can form candidate rules such as X implies Y (X —> Y). *Confidence* is defined as the measure of certainty or trustworthiness associated with each discovered rule. Mathematically, confidence is the percent of transactions that contain both X and Y out of all the transactions that contain X

$$Confidence(X \rightarrow Y) = \frac{Support(X \wedge Y)}{Support(X)}$$

For example, if ***{bread, eggs, milk}*** has a support of 0.15 and ***{bread, eggs}*** also has a support of 0.15, the confidence of rule ***{bread, eggs }->{ milk}*** is 1, which means 100% of the time a customer buys bread and eggs, milk is bought as well. The rule is therefore correct for 100% of the transactions containing bread and eggs.

A relationship may be thought of as interesting when the algorithm identifies the relationship with a measure of confidence greater than or equal to a predefined threshold. This predefined threshold is called the *minimum confidence.* A higher confidence indicates that the rule (X —> Y) is more interesting or more trustworthy, based on the sample dataset.

Even though confidence can identify the interesting rules from all the candidate rules, it comes with a problem.

Given rules in the form of X -> Y, confidence considers only the antecedent (X) and the cooccurrence of X and Y; it does not take the consequent of the rule (Y) into concern. Therefore, confidence cannot tell if a rule contains true implication of the relationship or if the rule is purely coincidental. X and Y can be statistically independent yet still receive a high confidence score. Other measures such as lift and leverage are designed to address this issue.

Lift measures how many times more often X and Y occur together than expected if they are statistically independent of each other. Lift is a measure of how X and Y are really related rather than coincidentally happening together

$$Lift(X \rightarrow Y) = \frac{Support(X \wedge Y)}{Support(X) * Support(Y)}$$

Lift is 1 if X and Y are statistically independent of each other. In contrast, a lift of X —> Y greater than 1 indicates that there is some usefulness to the rule. A larger value of lift suggests a greater strength of the association between X and Y.

Assuming 1,000 transactions, with (milk, eggs} appearing in 300 of them, {milk} appearing in 500, and {eggs} appearing in 400, then *Lift(milk—>eggs)* = 0.3/(0.5*0.4) = 1.5. If {bread} appears in 400 transactions and {milk, bread} appears in 400, then *Lift(milk —>bread)* = 0.4/(0.5*0.4) = 2. Therefore it can be concluded that milk and bread have a stronger association than milk and eggs.

Leverage is a similar notion, but instead of using a ratio, leverage uses the difference. Leverage measures the difference in the probability of X and Y appearing together in the dataset compared to what would be expected if X and Y were statistically independent of each other.

Leverage(X —> Y) = Support(X ∧Y) - Support(X)* Support(Y)

In theory, leverage is 0 when X and Y are statistically independent of each other. If X and Y have some kind of relationship, the leverage would be greater than zero. A larger leverage value indicates a stronger relationship between X and Y. For the previous example, *Leverage{milk —> eggs)* = 0.3-(0.5*0.4) = 0.1 and *Leverage(milk -> bread*)=0.4 - (0.5 * 0.4) = 0.2. It again confirms that milk and bread have a stronger association than milk and eggs.

Confidence is able to identify trustworthy rules, but it cannot tell whether a rule is coincidental.

3.3.4-Applications of Association Rules

The term *market basket analysis* refers to a specific implementation of association rules mining that many companies use for a variety of purposes, including these:

- Broad-scale approaches to better merchandising—what products should be included in or excluded from the inventory each month
- Cross-merchandising between products and high-margin or high-ticket items
- Physical or logical placement of product within related categories of products
- Promotional programs—multiple product purchase incentives managed through a loyalty card program

Besides market basket analysis, association rules are commonly used for recommender systems and clickstream analysis.

Many online service providers such as Amazon and Netflix use recommender systems. Recommender systems

can use association rules to discover related products or identify customers who have similar interests. For example, association rules may suggest that those customers who have bought product A have also bought product B, or those customers who have bought products A, B, and C are more similar to this customer. These findings provide opportunities for retailers to cross-sell their products.

Clickstream analysis refers to the analytics on data related to web browsing and user clicks, which is stored on the client or the server side. Web usage log files generated on web servers contain huge amounts of information, and association rules can potentially give useful knowledge to web usage data analysts. For example, association rules may suggest that website visitors who land on page X click on links A, B, and C much more often than links D, E, and F. This observation provides valuable insight on how to better personalize and recommend the content to site visitors.

3.3.5-Validation and Testing

After gathering the output rules, it may become necessary to use one or more methods to validate the results in the business context for the sample dataset. The first approach can be established through statistical measures such as confidence, lift, and leverage. Rules that involve mutually independent items or cover few transactions are considered uninteresting because they may capture spurious relationships.

Confidence measures the chance that X and Y appear together in relation to the chance X appears. Confidence can be used to identify the interestingness of the rules.

Lift and leverage both compare the support of X and Y against their individual support. While mining data with association rules, some rules generated could be purely coincidental. For example, if 95% of customers buy X and 90% of customers buy Y, then X and Y would occur together at least 85% of the time, even if there is no relationship between the two.

Another set of criteria can be established through subjective arguments. Even with a high confidence, a rule may be considered subjectively uninteresting unless it reveals any unexpected profitable actions. For example, rules like {***paper}->{pencil***} may not be subjectively interesting or meaningful despite high support and confidence values. In contrast, a rule like ***{diaper}->{beer}*** that satisfies both minimum support and minimum confidence can be considered subjectively interesting because this rule is unexpected and may suggest a cross-sell opportunity for the retailer.

3.3.6-Diagnostics

Although the Apriori algorithm is easy to understand and implement, some of the rules generated are uninteresting or practically useless. Additionally, some of the rules may be generated due to coincidental relationships between the variables. Measures like confidence, lift, and leverage should be used along with human insights to address this problem.

Another problem with association rules is that, in Phase 3 and 4 of the Data Analytics Lifecycle, the team must specify the minimum support prior to the model execution, which may lead to too many or too few rules. In related research, a variant of the algorithm can use a predefined target range for the number of rules so that the algorithm can adjust the minimum support accordingly.

Apriori algorithm is one of the earliest and the most fundamental algorithms for generating association rules. The Apriori algorithm reduces the computational workload by only examining itemsets that meet the specified minimum threshold. However, depending on the size of the dataset, the Apriori algorithm can be computationally expensive. For each level of support, the algorithm requires a scan of the entire database to obtain the result. Accordingly, as the database grows, it takes more time to compute in each run. Here are some approaches to

improve Apriori's efficiency:

- **Partitioning:** Any itemset that is potentially frequent in a transaction database must be frequent in at least one of the partitions of the transaction database.
- **Sampling:** This extracts a subset of the data with a lower support threshold and uses the subset to perform association rule mining.
- **Transaction reduction:** A transaction that does not contain frequent fc-itemsets is useless in subsequent scans and therefore can be ignored.
- **Hash-based itemset counting:** If the corresponding hashing bucket count of a fc-itemset is below a certain threshold, the/c-itemset cannot be frequent.
- **Dynamic itemset counting:** Only add new candidate itemsets when all of their subsets are estimated to be frequent.

Chapter IV

Objectives:

To Study and Understand the following concept

- Regression
- Linear Regression
- Logistic Regression
- Additional Regression Models.

4-Regression Analysis

In general, regression analysis attempts to explain the influence that a set of variables has on the outcome of another variable of interest. Often, the outcome variable is called a *dependent variable* because the outcome depends on the other variables. These additional variables are sometimes called the *input variables* or the *independent variables*. Regression analysis is useful for answering the following kinds of questions:

- What is a person's expected income?
- What is the probability that an applicant will default on a loan?

Linear regression is a useful tool for answering the first question, and logistic regression is a popular method for addressing the second.

Regression analysis is a useful explanatory tool that can identify the input variables that have the greatest statistical influence on the outcome. With such knowledge and insight, environmental changes can be attempted to produce more favorable values of the input variables. For example, if it is found that the reading level of 10-year-old students is an excellent predictor of the students' success in high school and a factor in their attending college, then additional emphasis on reading can be considered, implemented, and evaluated to improve students' reading levels at a younger age.

4.1-Linear Regression

Linear regression is an analytical technique used to model the relationship between several input variables and a continuous outcome variable. A key assumption is that the relationship between an input variable and the outcome variable is linear. Although this assumption may appear restrictive, it is often possible to properly transform the input or outcome variables to achieve a linear relationship between the modified input and outcome variables.

A linear regression model is a probabilistic one that accounts for the randomness that can affect any particular outcome. Based on known input values, a linear regression model provides the expected value of the outcome

variable based on the values of the input variables, but some uncertainty may remain in predicting any particular outcome.

4.1.1-Use Cases

Linear regression is often used in business, government, and other scenarios. Some common practical applications of linear regression in the real world include the following:

- **Real estate:** A simple linear regression analysis can be used to model residential home prices as a function of the home's living area. Such a model helps set or evaluate the list price of a home on the market. The model could be further improved by including other input variables such as number of bathrooms, number of bedrooms, lot size, school district rankings, crime statistics, and property taxes
- **Demand forecasting:** Businesses and governments can use linear regression models to predict demand for goods and services. For example, restaurant chains can appropriately prepare for the predicted type and quantity of food that customers will consume based upon the weather, the day of the week, whether an item is offered as a special, the time of day, and the reservation volume. Similar models can be built to predict retail sales, emergency room visits, and ambulance dispatches.
- **Medical:** A linear regression model can be used to analyze the effect of a proposed radiation treatment on reducing tumor sizes. Input variables might include duration of a single radiation treatment, frequency of radiation treatment, and patient attributes such as age or weight.

4.1.2-Model Description

As the name of this technique suggests, the linear regression model assumes that there is a linear relationship between the input variables and the outcome variable. This relationship can be expressed as shown in Equation

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \ldots\ldots\ldots + \beta p\text{-}_1 x\ p\text{-}_1 + \varepsilon$$

where:

y is the outcome variable

x_j are the input variables, for j=1,2,...,p-1

β_0 is the value of y when each x_j equals zero

β_j is the change in y based on a unit change in x_j for j=1,2,...,p-1

ε is a random error term that represents the difference in the linear model and a particular observed value for y.

Suppose it is desired to build a linear regression model that estimates a person's annual income as a function of two variables—age and education—both expressed in years. In this case, income is the outcome variable, and the input variables are age and education.

However, it is also obvious that there is considerable variation in income levels for a group of people with identical ages and years of education. This variation is represented by εin the model. So, in this example, the model would be expressed as shown in Equation.

$$\text{Income} = \beta_0 + \beta_1 \text{Age} + \beta_2 \text{Education} + \varepsilon$$

Linear Regression Model (Ordinary Least Squares)

In the linear model, the β_2s represent the unknown p parameters. The estimates for these unknown parameters are chosen so that, on average, the model provides a reasonable estimate of a person's income based

on age and education. In other words, the fitted model should minimize the overall error between the linear model and the actual observations. Ordinary Least Squares (OLS) is a common technique to estimate the parameters.

To illustrate how OLS works, suppose there is only one input variable, x, for an outcome variable y. Furthermore, *n* observations of (x, *y)* are obtained and plotted in below Figure.

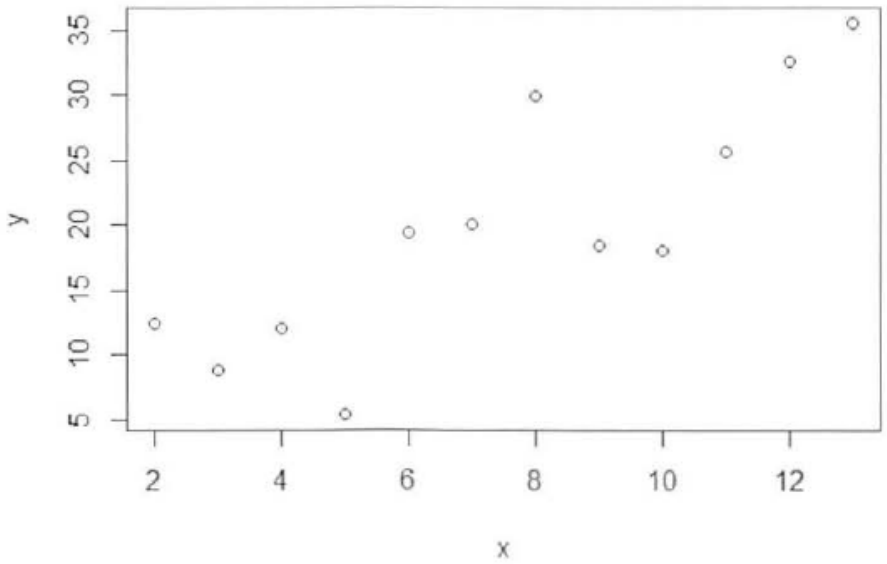


Figure 4.1. Scatterplot of y versus x

The goal is to find the line that best approximates the relationship between the outcome variable and the input variables. With OLS, the objective is to find the line through these points that minimizes the sum of the squares of the difference between each point and the line in the vertical direction. In other words, find the values of β_0 and β_1, such that the summation shown in Equation is minimized.

$$\sum_{i=1}^{n} [y_i - (\beta_0 + \beta_1 x_i)]^2$$

The n individual distances to be squared and then summed are illustrated in below figure. The vertical lines represent the distance between each observed y value and the line $y = \beta_0 + \beta_1 \mathbf{x_1}$

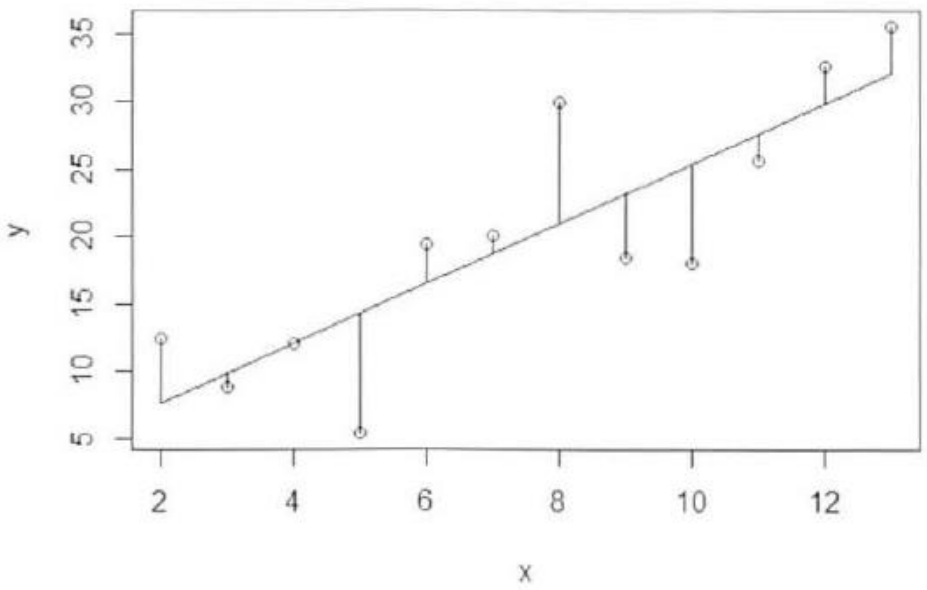


Figure 4.2: Scatterplot of y versus x with vertical distance from the observed points to a fitted line

Linear Regression Model (with Normally Distributed Errors)

In the normal model description, there were no assumptions made about the error term; no additional assumptions were necessary for OLS to provide estimates of the model parameters. However, in most linear regression analyses, it is common to assume that the error term is a normally distributed random variable with mean equal to zero and constant variance. Thus, the linear regression model is expressed as shown in Equation.

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \ldots\ldots\ldots + \beta_{p-1} x_{p-1} + \varepsilon$$

where:

y is the outcome variable

x_j are the input variables, for $j=1,2,...,p-1$

β_0 is the value of y when each x_j equals zero

β_j is the change in y based on a unit change in x_j for $j=1,2,...,p-1$

$\varepsilon \sim N(0,\sigma^2)$ and the εs are independent of each other

This additional assumption yields the following result about the expected value of y, E(y) for given $(x_1, x_2, \ldots x_{p-1})$:

$$\begin{aligned} E(y) &= E(\beta_0 + \beta_1 x_1 + \beta_2 x_2 \ldots + \beta_{p-1} x_{p-1} + \varepsilon) \\ &= \beta_0 + \beta_1 x_1 + \beta_2 x_2 \ldots + \beta_{p-1} x_{p-1} + E(\varepsilon) \\ &= \beta_0 + \beta_1 x_1 + \beta_2 x_2 \ldots + \beta_{p-1} x_{p-1} \end{aligned}$$

Because β_j and x_j are constants, the E(y) is the value of the linear regression model for the given $(x_1, x_2, \ldots x_{p-1})$. Furthermore, the variance of y, V(y), for given $(x_1, x_2, \ldots x_{p-1})$ is this:

$$V(y) = V(\beta_0 + \beta_1 x_1 + \beta_2 x_2 \ldots + \beta_{p-1} x_{p-1} + \varepsilon)$$
$$= 0 + V(\varepsilon) = \sigma^2$$

Thus, for a given (x_1, x_2,... x_{p-1}), y is normally distributed with mean $\beta_0 + \beta_1 x_1 + \beta_2 x_2 \ldots\ldots + \beta_{p-1} x_{p-1}$ and variance σ^2. For a regression model with just one input variable, below figure illustrates the normality assumption on the error terms and the effect on the outcome variable, y, for a given value of x.

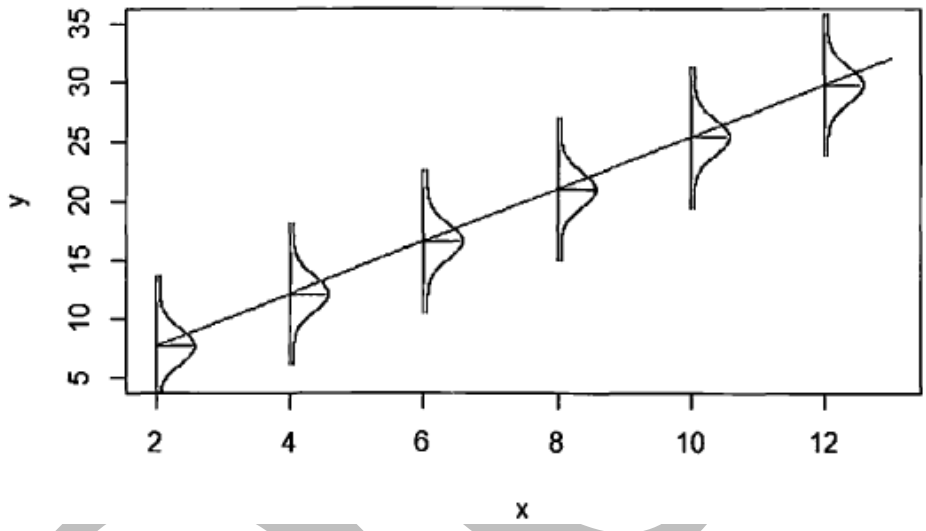


Figure 4.3: Normal distribution about y for a given value of x

4.1.3-Diagnostics

The use of hypothesis tests, confidence intervals, and prediction intervals is dependent on the model assumptions being true. Following are Some tools and techniques that can be used to validate a fitted linear regression model.

a-Evaluating the Linearity Assumption

A major assumption in linear regression modeling is that the relationship between the input variables and the outcome variable is linear. The most fundamental way to evaluate such a relationship is to plot the outcome variable against each input variable. If the relationship between *Age* and *Income* is represented as illustrated in Figure 4.3, a linear model would not apply.

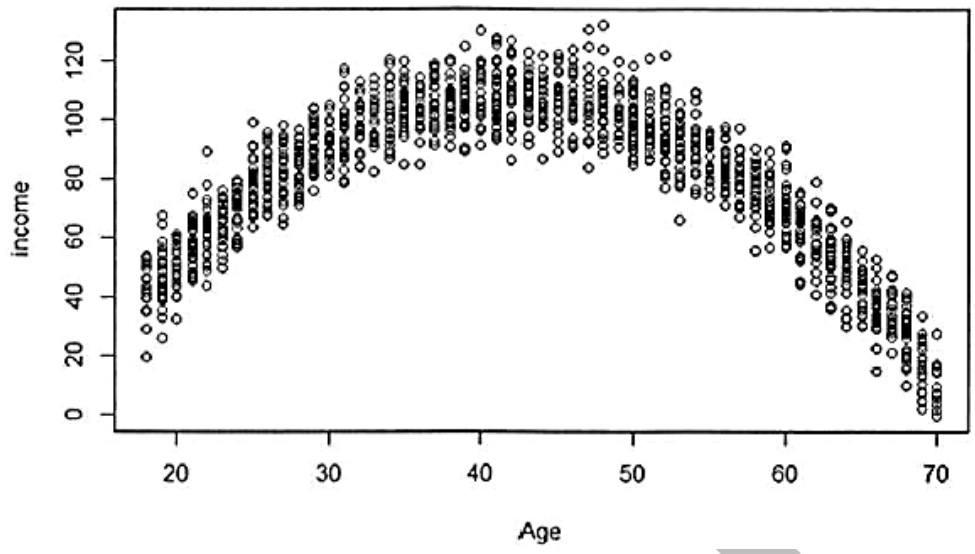


Figure 4.3: Income as a quadratic function of Age

In such a case, it is often useful to do any of the following:

- Transform the outcome variable.
- Transform the input variables.
- Add extra input variables or terms to the regression model.

Common transformations include taking square roots or the logarithm of the variables. Another option is to create a new input variable such as the age squared and add it to the linear regression model to fit a quadratic relationship between an input variable and the outcome.

b-Evaluating the Residuals

As stated previously, it is assumed that the error terms in the linear regression model are normally distributed with a mean of zero and a constant variance. If this assumption does not hold, the various inferences that were made with the hypothesis tests, confidence intervals, and prediction intervals are suspect.

c-Evaluating the Normality Assumption

The residual plots are useful for confirming that the residuals were centered on zero and have a constant variance. However, the normality assumption still has to be validated.

d-N-Fold Cross-Validation

To prevent overfitting a given dataset, a common practice is to randomly split the entire dataset into a training set and a testing set. Once the model is developed on the training set, the model is evaluated against the testing set. When there is not enough data to create training and testing sets, an N-fold cross-validation technique may be helpful to compare one fitted model against another. In N-fold cross-validation, the following occurs:

- The entire dataset is randomly split into N datasets of approximately equal size.
- A model is trained against N - 1 of these datasets and tested against the remaining dataset. A measure of the model error is obtained.
- This process is repeated a total of N times across the various combinations of N datasets taken N - 1 at a time. Recall:

$$\binom{N}{N-1} = N$$

- The observed N model errors are averaged over the N folds.

The averaged error from one mode! is compared against the averaged error from another model. This technique can also help determine whether adding more variables to an existing model is beneficial or possibly overfitting the data.

4.2-Logistic Regression

In linear regression modeling, the outcome variable is a continuous variable. When the outcome variable is categorical in nature, logistic regression can be used to predict the likelihood of an outcome based on the input variables. Although logistic regression can be applied to an outcome variable that represents multiple values, but we will examine the case in which the outcome variable represents two values such as true/false, pass/fail, or yes/no.

For example, a logistic regression model can be built to determine if a person will or will not purchase a new automobile in the next 12 months. The training set could include input variables for a person's age, income, and gender as well as the age of an existing automobile. The training set would also include the outcome variable on whether the person purchased a new automobile over a 12-month period. The logistic regression model provides the likelihood or probability of a person making a purchase in the next 12 months.

4.2.1-Use Cases

The logistic regression model is applied to a variety of situations in both the public and the private sector.

Some common ways that the logistic regression model is used include the following:

- **Medical:** Develop a model to determine the likelihood of a patient's successful response to a specific medical treatment or procedure. Input variables could include age, weight, blood pressure, and cholesterol levels.
- ***Finance:*** Using a loan applicant's credit history and the details on the loan, determine the probability that an applicant will default on the loan. Based on the prediction, the loan can be approved or denied, or the terms can be modified.
- ***Marketing:*** Determine a wireless customer's probability of switching carriers (known as churning) based on age, number of family members on the plan, months remaining on the existing contract, and social network contacts. With such insight, target the high-probability customers with appropriate offers to prevent churn.
- ***Engineering:*** Based on operating conditions and various diagnostic measurements, determine the probability of a mechanical part experiencing a malfunction or failure. With this, probability estimate, schedule the appropriate preventive maintenance activity.

4.2.2-Model Description

Logistic regression is based on the logistic function $f(y)$, as given in Equation 6-7.

$$f(y) = \frac{e^y}{1+e^y} \quad \text{for } -\infty < y < \infty \tag{6-7}$$

Note that as $y \to \infty$, $f(y) \to 1$, and as $y \to -\infty$, $f(y) \to 0$. So, as Figure 6-14 illustrates, the value of the logistic function $f(y)$ varies from 0 to 1 as y increases.

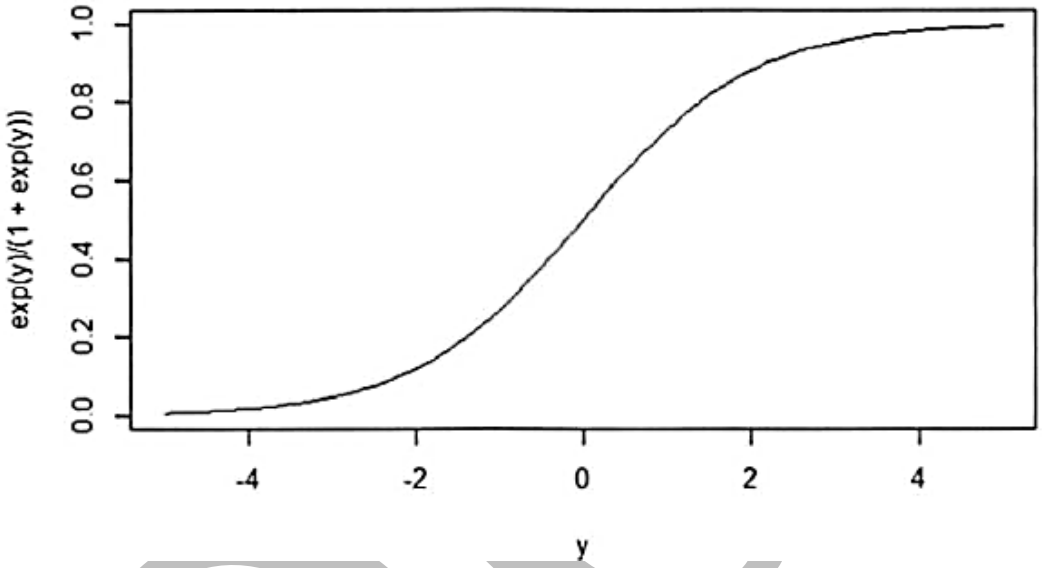


Figure 4.4. The logistic function

Because the range of $f(y)$ is (0, 1), the logistic function appears to be an appropriate function to model the probability of a particular outcome occurring. As the value of y increases, the probability of the outcome occurring increases. In any proposed model, to predict the likelihood of an outcome, y needs to be a function of the input variables. In logistic regression, y is expressed as a linear function of the input variables. In other words, the formula shown in Equation 6-8 applies.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \ldots + \beta_{p-1} x_{p-1} \quad (6\text{-}8)$$

Then, based on the input variables $x_1, x_2, \ldots, x_{p-1}$, the probability of an event is shown in Equation 6-9.

$$p(x_1, x_2, \ldots, x_{p-1}) = f(y) = \frac{e^y}{1+e^y} \quad \text{for } -\infty < y < \infty \quad (6\text{-}9)$$

Equation 6-8 is comparable to Equation 6-1 used in linear regression modeling. However, one difference is that the values of y are not directly observed. Only the value of $f(y)$ in terms of success or failure (typically expressed as 1 or 0, respectively) is observed.

Using p to denote $f(y)$, Equation 6-9 can be rewritten in the form provided in Equation 6-10.

$$ln\left(\frac{p}{1-p}\right) = y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \ldots + \beta_p x_{p-1} \quad (6\text{-}10)$$

The quantity $ln\left(\frac{p}{1-p}\right)$, in Equation 6-10 is known as the log odds ratio, or the logit of p. Techniques such as Maximum Likelihood Estimation (MLE) are used to estimate the model parameters. MLE determines the values of the model parameters that maximize the chances of observing the given dataset.

Customer Churn Example

A wireless telecommunications company wants to estimate the probability that a customer will churn (switch to a different company) in the next six months. With a reasonably accurate prediction of a person's likelihood of churning, the sales and marketing groups can attempt to retain the customer by offering various incentives. Data on 8,000 current and prior customers was obtained. The variables collected for each customer follow:

- *Age* (years)
- *Married*(true/false)
- *Duration* as a customer (years)
- *Churned_contacts*(count)—Number of the customer's contacts that have churned (count)
- *churned* (true/false)—Whether the customer churned

After analyzing the data and fitting a logistic regression model, *Age* and *Churned_contacts* were selected as the best predictor variables. Equation 6-11 provides the estimated model parameters.

$$y = 3.50 - 0.16 * \textbf{\textit{Age}} + 0.38 * \textbf{\textit{Churned\_contacts}} \quad (6.11)$$

Using the fitted model from Equation 4-1, below table provides the probability of a customer churning based on the customer's age and the number of churned contacts.

Table 4.1 Estimated churn probabilities

Customer	Age (Years)	Churned_Contacts	y	Prob. of Churning
1	50	1	–4.12	0.016
2	50	3	–3.36	0.034
3	50	6	–2.22	0.098
4	30	1	–0.92	0.285
5	30	3	–0.16	0.460
6	30	6	0.98	0.727
7	20	1	0.68	0.664
8	20	3	1.44	0.808
9	20	6	2.58	0.930

Based on the fitted model, there is a 93% chance that a 20-year-old customer who has had six contacts churn will also churn.

4.2.3-Diagnostics

Deviance and the Pseudo-R^2

In logistic regression, deviance is defined to be $-2 * logL$, where L is the maximized value of the likelihood function that was used to obtain the parameter estimates. In the R output, two deviance values are provided. The ***null deviance*** is the value where the likelihood function is based only on the intercept term ($y = \beta_0$). The ***residual deviance*** is the value where the likelihood function is based on the parameters in the specified logistic model, shown in Equation 6-12.

$$y = \beta_0 + \beta_1 * Age + \beta_2 * Churned\_contacts \tag{6-12}$$

A metric analogous to R^2 in linear regression can be computed as shown in Equation 6-13.

$$\text{pseudo-}R^2 = 1 - \frac{residual\ dev.}{null\ dev.} = \frac{null\ dev. - res.\ dev.}{null\ dev.} \tag{6-13}$$

The pseudo-R^2 is a measure of how well the fitted model explains the data as compared to the default model of no predictor variables and only an intercept term. A $pseudo - R^2$ value near 1 indicates a good fit over the simple null model.

Deviance and the Log-Likelihood Ratio Test

In the *pseudo* $-R^2$ calculation, the –2 multipliers simply divide out. So, it may appear that including such a multiplier does not provide a benefit. However, the multiplier in the deviance definition is based on the log-likelihood test statistic shown in Equation 6-14:

$$T = -2 * log\left(\frac{L_{null}}{L_{alt.}}\right) = -2 * \log(L_{null}) - (-2) * \log(L_{alt.}) \quad (6\text{-}14)$$

where T is approximately Chi-squared distributed (χ^2_k) with

k *degrees of freedom* $(df) = df_{null} - df_{alternate}$

The previous description of the log-likelihood test statistic applies to any estimation using MLE. As can be seen in Equation 6-15, in the logistic regression case,

$$T = \text{null deviance} - \text{residual deviance} \sim \chi^2_{p-1} \quad (6\text{-}15)$$

where p is the number of parameters in the fitted model.

So, in a hypothesis test, a large value of T would indicate that the fitted model is significantly better than the null model that uses only the intercept term.

In the churn example, the log-likelihood ratio statistic would be this:

$T = 8387.3 - 5359.2 = 3028.1$ with 2 degrees of freedom and a corresponding p-value that is essentially zero.

So far, the log-likelihood ratio test discussion has focused on comparing a fitted model to the default model of using only the intercept. However, the log-likelihood ratio test can also compare one fitted model to another.

Receiver Operating Characteristic (ROC) Curve

Logistic regression is often used as a classifier to assign class labels to a person, item, or transaction based on the predicted probability provided by the model. In the Churn example, a customer can be classified with the label called **Churn** if the logistic model predicts a high probability that the customer will churn. Otherwise, a **Remain** label is assigned to the customer. Commonly, 0.5 is used as the default probability threshold to distinguish between any two class labels. However, any threshold value can be used depending on the preference to avoid false positives (for example, to predict **Churn** when actually the customer will Remain) or false negatives (for example, to predict **Remain** when the customer will actually **Churn).**

Histogram of the Probabilities

It can be useful to visualize the observed responses against the estimated probabilities provided by the logistic regression. Figure 4-2 provides overlaying histograms for the customers who churned and for the customers who remained as customers. With a proper fitting logistic model, the customers.who remained tend to have a low probability of churning. Conversely, the customers who churned have a high probability of churning again. This histogram plot helps visualize the number of items to be properly classified or mis- dassified. In the Churn example, an ideal histogram plot would have the remaining customers grouped at the left side of the plot, the customers who churned at the right side of the plot, and no overlap of these two groups.

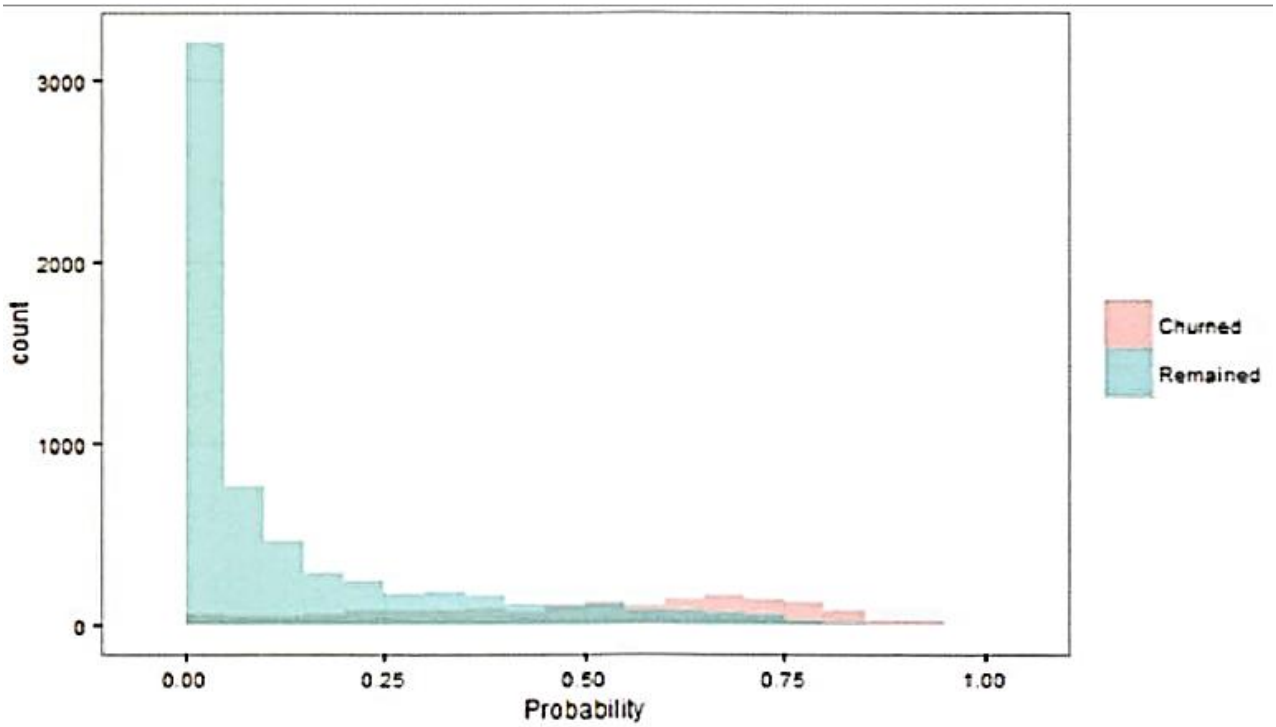


Figure4. 2 : Customer counts versus estimated churn probability

4.3-Reasons to Choose and Cautions

Linear regression is suitable when the input variables are continuous or discrete, including categorical data types, but the outcome variable is continuous. If the outcome variable is categorical, logistic regression is a better choice.

Both models assume a linear additive function of the input variables. If such an assumption does not hold true, both regression techniques perform poorly, Furthermore, in linear regression, the assumption of normally distributed error terms with a constant variance is important for many of the statistical inferences that can be considered. If the various assumptions do not appear to hold, the appropriate transformations need to be applied to the data.

Although a collection of input variables may be a good predictor for the outcome variable, the analyst should not infer that the input variables directly cause an outcome. For example, it may be identified that those individuals who have regular dentist visits may have a reduced risk of heart attacks. However, simply sending someone to the dentist almost certainly has no effect on that person's chance of having a heart attack. It is possible that regular dentist visits may indicate a person's overall health and dietary choices, which may have a more direct impact on a person's health.

Use caution when applying an already fitted model to data that falls outside the dataset used to train the model. The linear relationship in a regression model may no longer hold at values outside the training dataset. For example, if income was an input variable and the values of income ranged from $35,000 to $90,000, applying the model to incomes well outside those incomes could result in inaccurate estimates and predictions.

If several of the input variables are highly correlated to each other, the condition is known as *multicollinearity*. Multicollinearity can often lead to coefficient estimates that are relatively large in absolute magnitude and may be of inappropriate direction (negative or positive sign). When possible, the majority of these correlated variables should be removed from the model or replaced by a new variable that is a function of the correlated variables.

References:

1) Big Data and Analytics, Subhashini Chellappan Seema Acharya, Wiley First addition
2) Data Analytics with Hadoop, *An Introduction for Data Scientists, Benjamin Bengfort and Jenny Kim* O'Reilly 2016
3) Big Data and Hadoop V.K Jain KhannaPublishing First 2018

Unit II
Chapter III

Objectives:

To Study and Understand the following concept

- Analytical Theory and Methods:
- Clustering and Associated Algorithms
- Association Rules
- Apriori Algorithm
- Candidate Rules
- Applications of Association Rules
- Validation and Testing
- Diagnostics

3.1-Overview of Clustering

In general, clustering is the use of *unsupervised* techniques for grouping similar objects. In machine learning, unsupervised refers to the problem of finding hidden structure within unlabeled data. Clustering techniques are unsupervised in the sense that the data scientist does not determine, in advance, the labels to apply to the clusters. The structure of the data describes the objects of interest and determines how best to group the objects.

Clustering is a method often used for exploratory analysis of the data. In clustering, there are no predictions made. Rather, clustering methods find the similarities between objects according to the object attributes and group the similar objects into clusters. Clustering techniques are utilized in marketing, economics, and various branches of science. A popular clustering method is k-means.

3.2-K-means

Given a collection of objects each with n measurable attributes, *k-means* is an analytical technique that, for a chosen value of k, identifies k clusters of objects based on the objects' proximity to the center of the k groups. The center is determined as the arithmetic average (mean) of each cluster's n-dimensional vector of attributes. Below figure illustrates three dusters of objects with two attributes. Each object in the dataset is represented by a small dot color-coded to the closest large dot, the mean of the cluster.

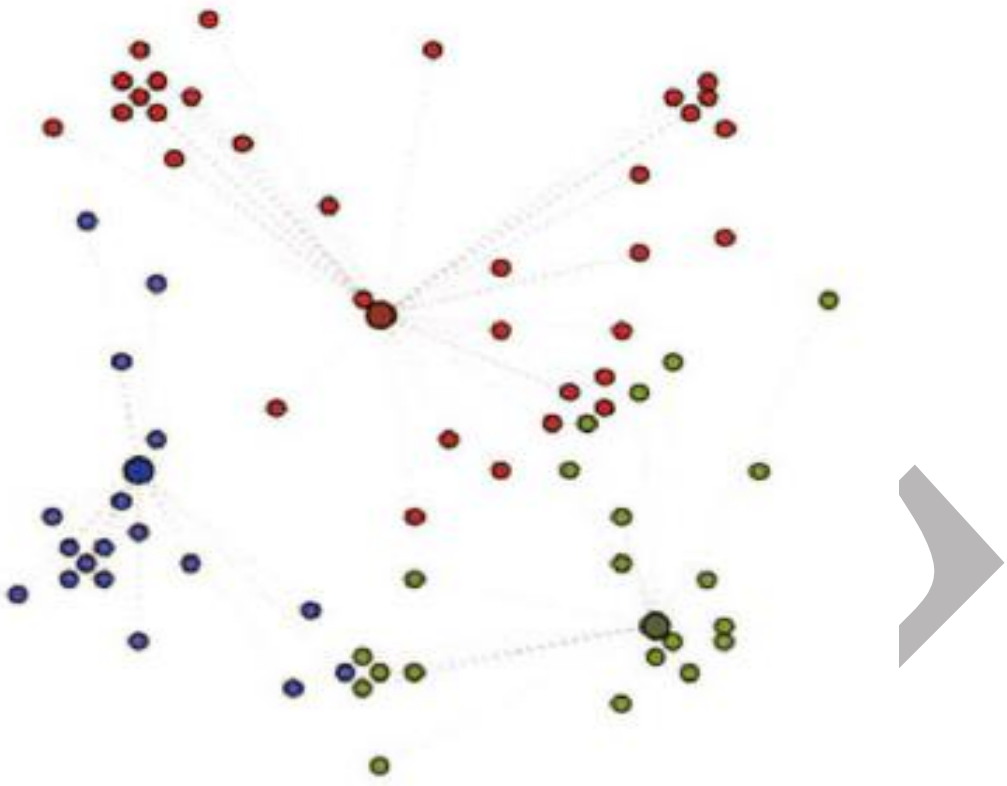


Figure 3.1 Possible K-means clusters for k=3

3.2.1-Use Cases

Clustering is often used as a lead-in to classification. Once the clusters are identified, labels can be applied to each cluster to classify each group based on its characteristics. Some specific applications of k-means are image processing, medical and customer segmentation.

Image Processing

Video is one example of the growing volumes of unstructured data being collected. Within each frame of a video, k-means analysis can be used to identify objects in the video. For each frame, the task is to determine which pixels are most similar to each other. The attributes of each pixel can include brightness, color, and location, the x and y coordinates in the frame. With security video images, for example, successive frames are examined to identify any changes to the clusters. These newly identified dusters may indicate unauthorized access to a facility.

Medical

Patient attributes such as age, height, weight, systolic and diastolic blood pressures, cholesterol level, and other attributes can identify naturally occurring clusters. These dusters could be used to target individuals for specific preventive measures or clinical trial participation. Clustering, in general, is useful in biology for the classification of plants and animals as well as in the field of human genetics.

Customer Segmentation

Marketing and sales groups use k-means to better identify customers who have similar behaviors and spending patterns. For example, a wireless provider may look at the following customer attributes: monthly bill, number of text messages, data volume consumed, minutes used during various daily periods, and years as a customer. The wireless company could then look at the naturally occurring clusters and consider tactics

to increase sales or reduce the customer *churn rate,* the proportion of customers who end their relationship with a particular company.

3.2.2-Overview of the Method

To illustrate the method to find k clusters from a collection of M objects with n attributes, the two-dimensional case (n = 2) is examined. It is much easier to visualize the k-means method in two dimensions.

Because each object in this example has two attributes, it is useful to consider each object corresponding to the point (x_i, y_i), where x and y denote the two attributes and i = 1, 2 ... M. For a given cluster of m points (m ≤ M), the point that corresponds to the cluster's mean is called a *centroid.*

The k-means algorithm to find k dusters can be described in the following four steps.

1. Choose the value of k and the k initial guesses for the centroids.

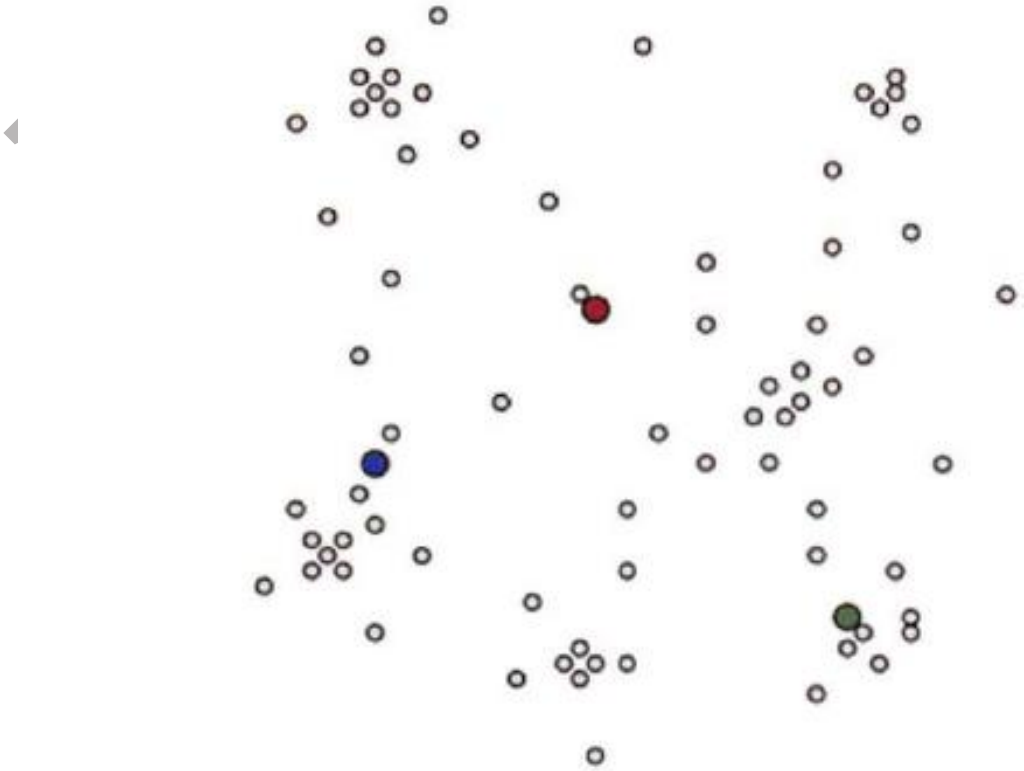
   In this example, k = 3, and the initial centroids are indicated by the points shaded in red, green, and blue in figure 2.2.

2. Compute the distance from each data point (x_i ,y_i) to each centroid. Assign each point to the closest centroid. This association defines the first k dusters.

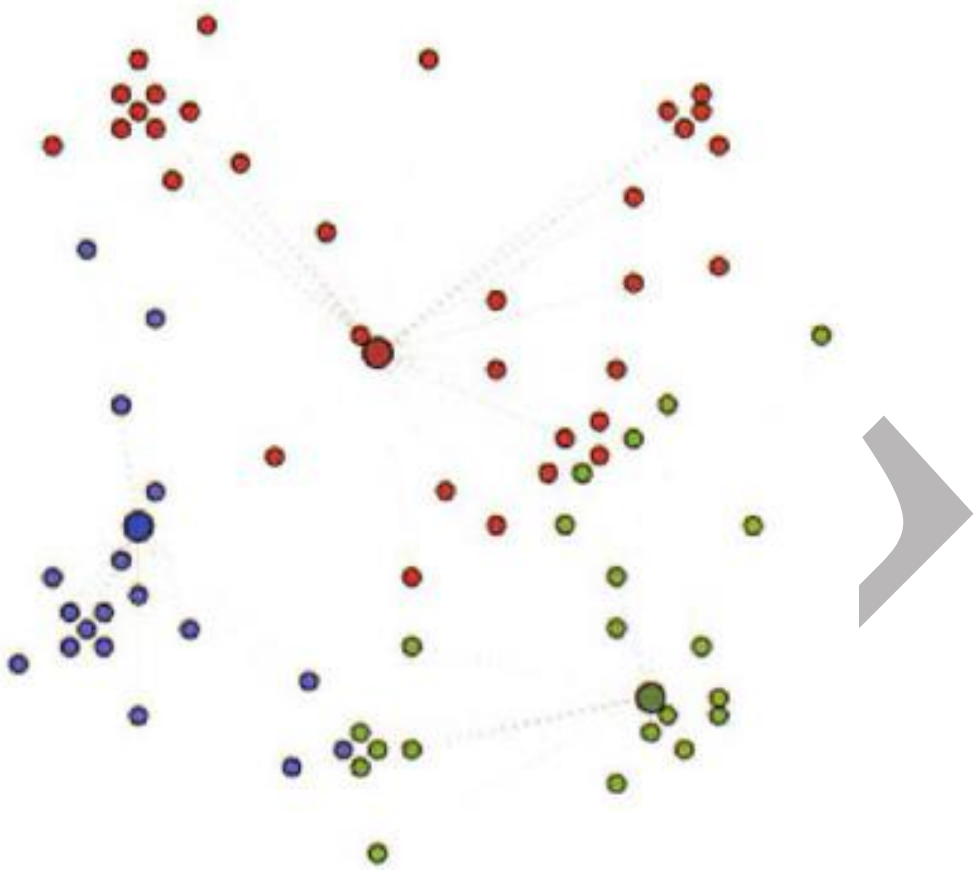
   In two dimensions, the distance, *d,* between any two points, (x_1,y_1) and (x_2,y_2), in the Cartesian plane is typically expressed by using the Euclidean distance measure provided in Equation.

   $$d= \sqrt{(x1 - y1)^2 + (x2 - y2)^2}$$

   In figure 3.2, the points closest to a centroid are shaded the corresponding color.



3.2-Initial starting points for the centroids

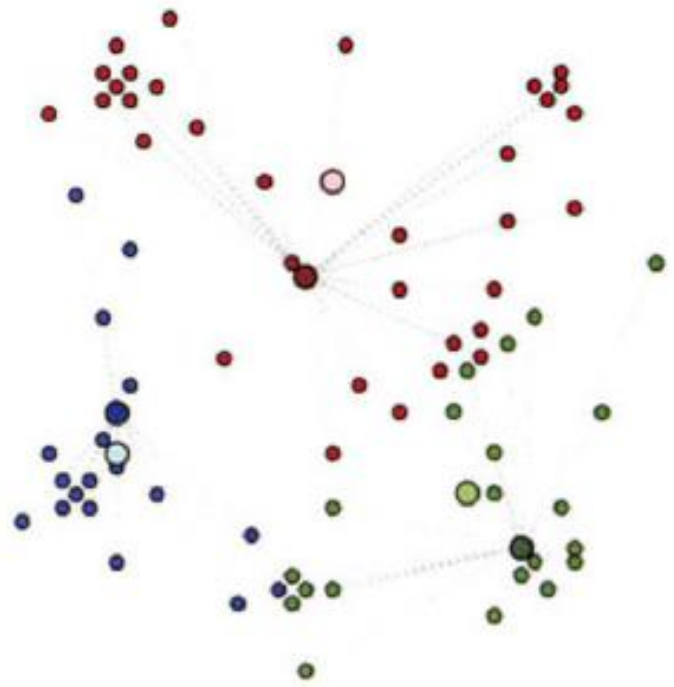


3.3-Points are assigned to the closest centroid

3. Compute the centroid, the center of mass, of each newly defined cluster from Step *2*.
   In Figure 2-4, the computed centroids in Step 3 are the lightly shaded points of the corresponding color. In two dimensions, the centroid (x_c,y_c) of the m points in a k-means duster is calculated as follows in Equation.

$$(x_c, y_c) = \left(\frac{\sum_{i=1}^{m} x_i}{m}, \frac{\sum_{i=1}^{m} y_i}{m} \right)$$

   Thus, (x_c,y_c) is the ordered pair of the arithmetic means of the coordinates of the m points in the cluster. In this step, a centroid is computed for each of the k clusters.



2.4-Compute the mean of each cluster

4. Repeat Steps 2 and 3 until the algorithm converges to an answer

    a. Assign each point to the closest centroid computed in Step 3.
    b. Compute the centroid of newly defined clusters.
    c. Repeat until the algorithm reaches the final answer.

3.2.3-Determining the Number of Clusters

In k-means, k clusters can be identified in a given dataset, but what value of k should be selected? The value of k can be chosen based on a reasonable guess or some predefined requirement. However, even then, it would be good to know how much better or worse having k clusters versus k-1 or k+1 cluster would be in explaining the structure of the data. Next, a heuristic using the Within Sum of Squares (WSS) metric is examined to determine a reasonably optimal value of k. Using the distance function, WSS is defined as shown below.

$$WSS = \sum_{i=1}^{M} d(p_i, q^{(i)})^2 = \sum_{i=1}^{M} \sum_{j=1}^{n} \left(p_{ij} - q_j^{(i)}\right)^2$$

In other words, WSS is the sum of the squares of the distances between each data point and the closest centroid. The term $\boldsymbol{q^{(i)}}$ indicates the closest centroid that is associated with the i^{th} point. If the points are relatively close to their respective centroids, the WSS is relatively small. Thus, if k +1 clusters do not greatly reduce the value of WSS from the case with only k clusters, there may be little benefit to adding another cluster.

3.2.4-Diagnostics

The heuristic using WSS can provide at least several possible k values to consider. When the number of attributes

is relatively small, a common approach to further refine the choice of k is to plot the data to determine how distinct the identified clusters are from each other. In general, the following questions should be considered.

- Are the dusters well separated from each other?
- Do any of the dusters have only a few points?
- Do any of the centroids appear to be too close to each other?

In the first case, ideally the plot would look like the one shown in below figure, when n = 2.

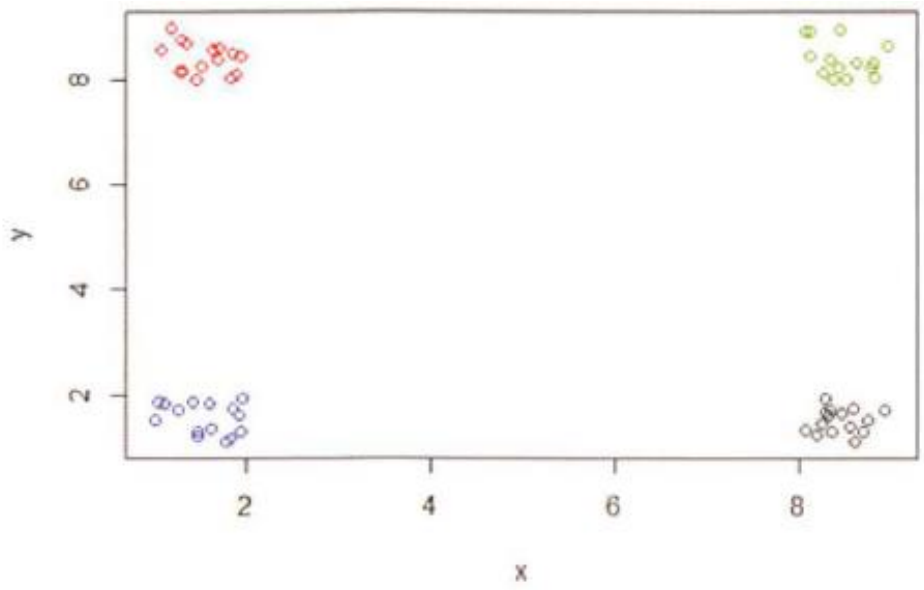


Figure: 3.4 Example of distinct clusters

The clusters are well defined, with considerable space between the four identified clusters. However, in other cases, such as in below figure, the clusters may be close to each other, and the distinction may not be so obvious.

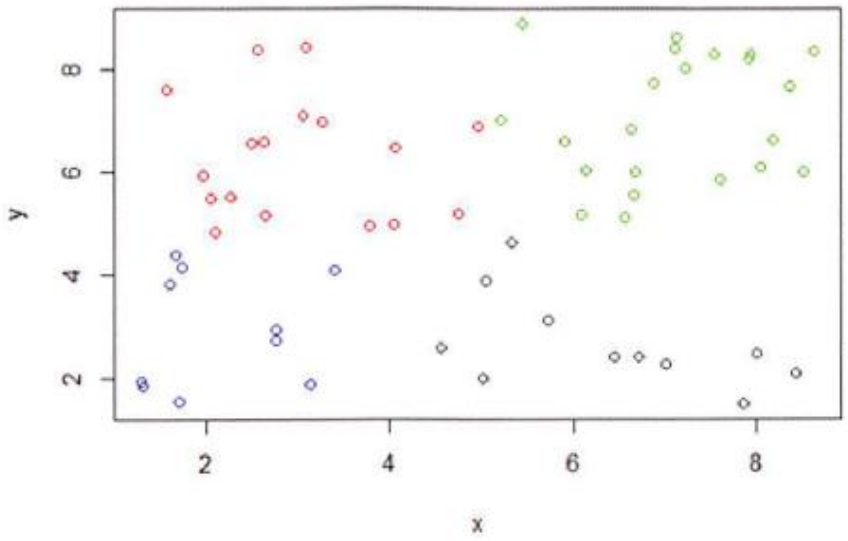


Figure 3.5 Example of less obvious cluster

3.2.5-Reasons to Choose and Cautions

K-means is a simple and straightforward method for defining clusters. Once clusters and their associated centroids are identified, it is easy to assign new objects (for example, new customers) to a cluster based on the object's distance from the closest centroid. Because the method is unsupervised, using k-means helps to eliminate subjectivity from the analysis.

Although k-means is considered an unsupervised method, there are still several decisions that the practitioner must make:

a. What object attributes should be included in the analysis?
b. What unit of measure (for example, miles or kilometers) should be used for each attribute?
c. Do the attributes need to be rescaled so that one attribute does not have a disproportionate effect on the results?
d. What other considerations might apply?

a-Object Attributes

Regarding which object attributes (for example, age and income) to use in the analysis, it is important to understand what attributes will be known at the time a new object will be assigned to a cluster. For example, information on existing customers' satisfaction or purchase frequency may be available, but such information may not be available for potential customers.

The Data Scientist may have a choice of a dozen or more attributes to use in the clustering analysis. Whenever possible and based on the data, it is best to reduce the number of attributes to the extent possible. Too many attributes can minimize the impact of the most important variables. Also, the use of several similar attributes can place too much importance on one type of attribute. For example, if five attributes related to personal wealth are included in a clustering analysis, the wealth attributes dominate the analysis and possibly mask the importance of other attributes, such as age.

When dealing with the problem of too many attributes, one useful approach is to identify any highly correlated attributes and use only one or two of the correlated attributes in the clustering analysis.

Another option to reduce the number of attributes is to combine several attributes into one measure. For example, instead of using two attribute variables, one for Debt and one for Assets, a Debt to Asset ratio could be used, This option also addresses the problem when the magnitude of an attribute is not of real interest, but the relative magnitude is a more important measure.

b-Units of Measure

From a computational perspective, the k-means algorithm is somewhat indifferent to the units of measure for a given attribute (for example, meters or centimeters for a patient's height). However, the algorithm will identify different clusters depending on the choice of the units of measure.

For example, suppose that k-means is used to cluster patients based on age in years and height in centimeters. For k=2, below figure illustrates the two clusters that would be determined for a given dataset.

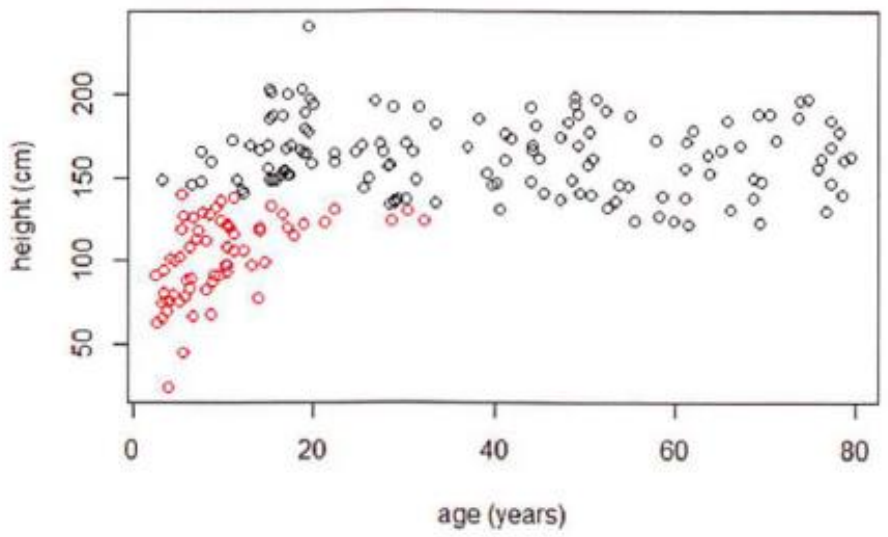


Figure 3.6. Cluster with height expressed in centimeters

But if the height was rescaled from centimeters to meters by dividing by 100, the resulting dusters would be slightly different, as illustrated in below Figure.

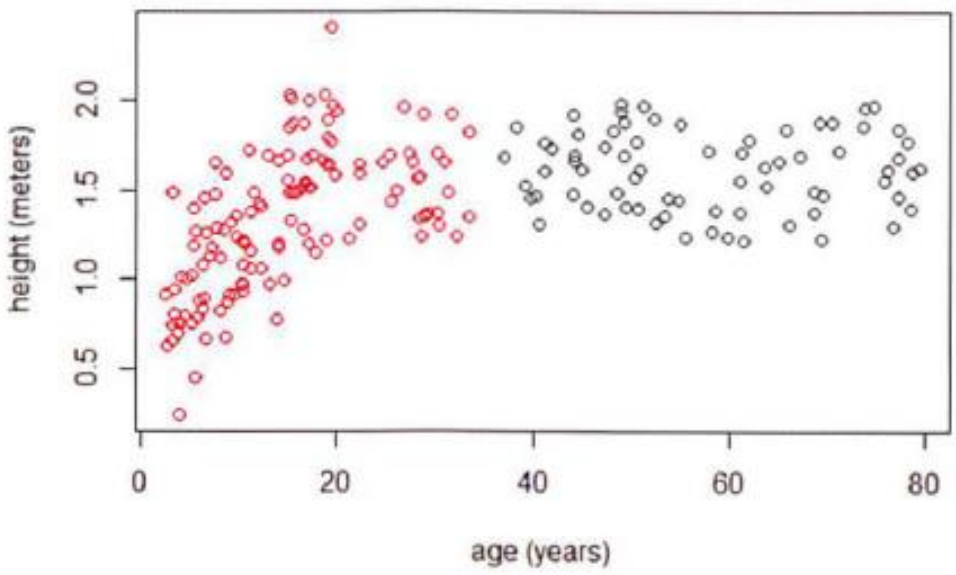


Figure 3.7. Cluster with height expressed in meters

c-Rescaling

Attributes that are expressed in dollars are common in clustering analyses and can differ in magnitude from the other attributes. For example, if personal income is expressed in dollars and age is expressed in years, the income attribute, often exceeding 510,000, can easily dominate the distance calculation with ages typically less than 100 years.

Although some adjustments could be made by expressing the income in thousands of dollars (for example, 10 for 510,000), a more straightforward method is to divide each attribute by the attribute's standard deviation. The resulting attributes will each have a standard deviation equal to 1 and will be without units.

Returning to the age and height example, the standard deviations are 23.1 years and 36.4 cm, respectively. Dividing each attribute value by the appropriate standard deviation and performing the k-means analysis yields the result shown in Figure 3.8.

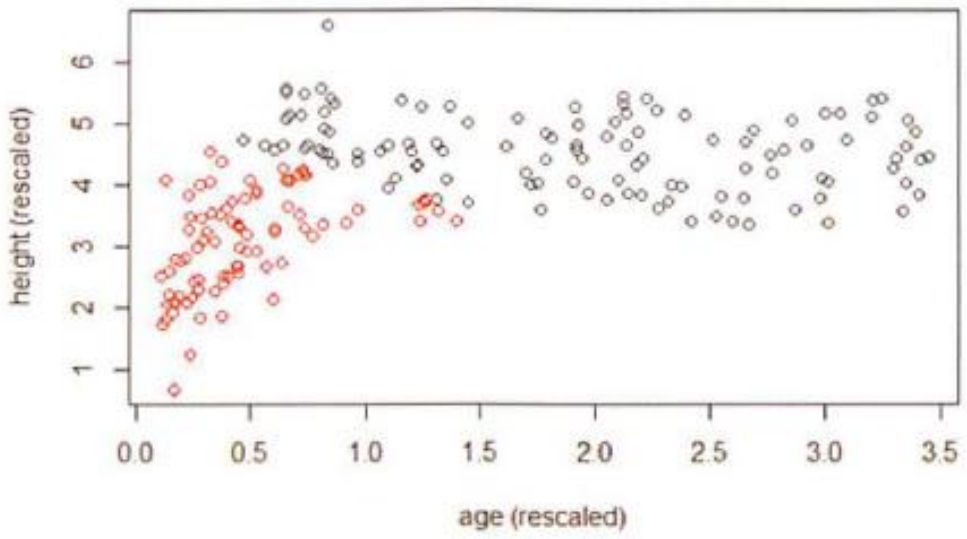


Figure 3.8.Cluster with rescaled attributes

In many statistical analyses, it is common to transform typically skewed data, such as income, with long tails by taking the logarithm of the data. Such transformation can also be applied in k-means, but the Data Scientist needs to be aware of what effect this transformation will have.

d-Additional Considerations

The k-means algorithm is sensitive to the starting positions of the initial centroid. Thus, it is important to rerun the k-means analysis several times for a particular value of k to ensure the cluster results provide the overall minimum WSS. As we know, this task is accomplished in R by using the nstart option in the kmeans () function call.

K-means clustering is applicable to objects that can be described by attributes that are numerical with a meaningful distance measure. Interval and ratio attribute types can certainly be used. However, k-means does not handle categorical variables well. For example, suppose a clustering analysis is to be conducted on new car sales. Among other attributes, such as the sale price, the color of the car is considered important. Although one could assign numerical values to the color, such as red = 1, yellow = 2, and green = 3, it is not useful to consider that yellow is as close to red as yellow is to green from a clustering perspective. In such cases, it may be necessary to use an alternative clustering methodology.

3.3.Association Rules

An unsupervised learning method called association rules. This is a descriptive, not predictive, method often used to discover interesting relationships hidden in a large dataset. The disclosed relationships can be represented as rules or frequent item sets. Association rules are commonly used for mining transactions in databases.

Here are some possible questions that association rules can answer:

- Which products tend to be purchased together?
- Of those customers who are similar to this person, what products do they tend to buy?
- Of those customers who have purchased this product, what other similar products do they tend to view or purchase?

3.3.1- Overview

Below figure shows the general logic behind association rules. Given a large collection of transactions (depicted as three stacks of receipts in the figure), in which each transaction consists of one or more items, association rules go through the items being purchased to see what items are frequently bought together and to discover a list of rules that describe the purchasing behavior. The goal with association rules is to discover interesting relationships among the items. The relationships that are interesting depend both on the business context and the nature of the algorithm being used for the discovery.

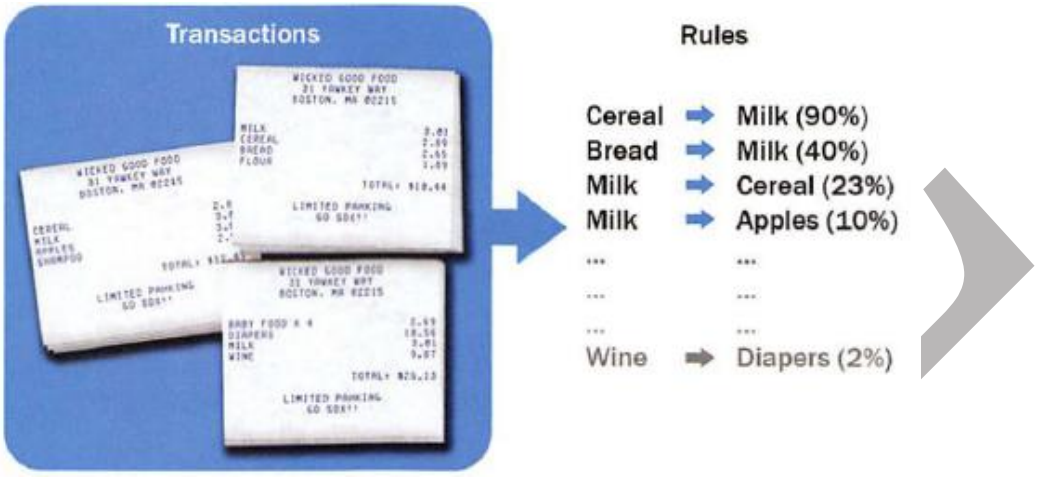


Figure 3.9 The general logic behind association rules

Each of the uncovered rules is in the form X —> Y, meaning that when item X is observed, item Y is also observed. In this case, the left-hand side (LHS) of the rule is X, and the right-hand side (RH5) of the rule is Y.

Using association rules, patterns can be discovered from the data that allow the association rule algorithms to disclose rules of related product purchases. The uncovered rules are listed on the right side of Figure. The first three rules suggest that when cereal is purchased, 90% of the time milk is purchased also. When bread is purchased, 40% of the time milk is purchased also. When milk is purchased, 23% of the time cereal is also purchased.

In the example of a retail store, association rules are used over transactions that consist of one or more items. In fact, because of their popularity in mining customer transactions, association rules are sometimes referred to as ***market basket analysis.*** Each transaction can be viewed as the shopping basket of a customer that contains one or more items. This is also known as an itemset. The term ***itemset*** refers to a collection of items or individual entities that contain some kind of relationship. This could be a set of retail items purchased together in one transaction, a set of hyperlinks clicked on by one user in a single session, or a set of tasks done in one day. An itemset containing ***k*** items is called a ***k-itemset*** denoted by {item1,item 2**,** . . . item k}**.**

3.3.2-Apriori Algorithm

The Apriori algorithm takes a bottom-up iterative approach to uncovering the frequent itemsets by first determining all the possible items (or 1-itemsets, for example {bread}, ***{eggs}, {milk},*** …) and then identifying which among them are frequent.

Assuming the minimum support threshold (or the minimum support criterion) is set at 0.5, the algorithm

identifies and retains those itemsets that appear in at least 50% of all transactions and discards (or "prunes away") the itemsets that have a support less than 0.5 or appear in fewer than 50% of the transactions.

In the next iteration of the Apriori algorithm, the identified frequent 1-itemsets are paired into 2-itemsets (for example, {bread, ***eggs***}, {bread, ***milk}, {eggs, milk},***…) and again evaluated to identify the frequent 2-itemsets among them.

At each iteration, the algorithm checks whether the support criterion can be met; if it can, the algorithm grows the itemset, repeating the process until it runs out of support or until the itemsets reach a predefined length. Let variable C_k be the set of candidate k-itemsets and variable L_k be the set of k-itemsets that satisfy the minimum support. Given a transaction database D, a minimum support threshold δ, and an optional parameter N indicating the maximum length an itemset could reach, Apriori iteratively computes frequent itemsets L_{k+1} , based on L_k.

```
1  Apriori (D, δ, N)
2    k ← 1
3    L_k ← {1-itemsets that satisfy minimum support δ}
4    while L_k ≠ ∅
5      if ∄N ∨ (∃N ∧ k < N)
6        C_{k+1} ← candidate itemsets generated from L_k
7        for each transaction t in database D do
8          increment the counts of C_{k+1} contained in t
9        L_{k+1} ← candidates in C_{k+1} that satisfy minimum support δ
10       k ← k + 1
11   return ⋃_k L_k
```

3.3.3-Evaluation of Candidate Rules

Frequent itemsets from the previous section can form candidate rules such as X implies Y (X —> Y). *Confidence* is defined as the measure of certainty or trustworthiness associated with each discovered rule. Mathematically, confidence is the percent of transactions that contain both X and Y out of all the transactions that contain X

$$Confidence(X \rightarrow Y) = \frac{Support(X \wedge Y)}{Support(X)}$$

For example, if ***{bread, eggs, milk}*** has a support of 0.15 and ***{bread, eggs}*** also has a support of 0.15, the confidence of rule ***{bread, eggs }->{ milk}*** is 1, which means 100% of the time a customer buys bread and eggs, milk is bought as well. The rule is therefore correct for 100% of the transactions containing bread and eggs.

A relationship may be thought of as interesting when the algorithm identifies the relationship with a measure of confidence greater than or equal to a predefined threshold. This predefined threshold is called the *minimum confidence.* A higher confidence indicates that the rule (X —> Y) is more interesting or more trustworthy, based on the sample dataset.

Even though confidence can identify the interesting rules from all the candidate rules, it comes with a problem.

Given rules in the form of X -> Y, confidence considers only the antecedent (X) and the cooccurrence of X and Y; it does not take the consequent of the rule (Y) into concern. Therefore, confidence cannot tell if a rule contains true implication of the relationship or if the rule is purely coincidental. X and Y can be statistically independent yet still receive a high confidence score. Other measures such as lift and leverage are designed to address this issue.

Lift measures how many times more often X and Y occur together than expected if they are statistically independent of each other. Lift is a measure of how X and Y are really related rather than coincidentally happening together

$$Lift(X \rightarrow Y) = \frac{Support(X \wedge Y)}{Support(X) * Support(Y)}$$

Lift is 1 if X and Y are statistically independent of each other. In contrast, a lift of X —> Y greater than 1 indicates that there is some usefulness to the rule. A larger value of lift suggests a greater strength of the association between X and Y.

Assuming 1,000 transactions, with (milk, eggs} appearing in 300 of them, {milk} appearing in 500, and {eggs} appearing in 400, then *Lift(milk—>eggs)* = 0.3/(0.5*0.4) = 1.5. If {bread} appears in 400 transactions and {milk, bread} appears in 400, then *Lift(milk —>bread)* = 0.4/(0.5*0.4) = 2. Therefore it can be concluded that milk and bread have a stronger association than milk and eggs.

Leverage is a similar notion, but instead of using a ratio, leverage uses the difference. Leverage measures the difference in the probability of X and Y appearing together in the dataset compared to what would be expected if X and Y were statistically independent of each other.

Leverage(X —> Y) = Support(X ∧Y) - Support(X)* Support(Y)

In theory, leverage is 0 when X and Y are statistically independent of each other. If X and Y have some kind of relationship, the leverage would be greater than zero. A larger leverage value indicates a stronger relationship between X and Y. For the previous example, *Leverage{milk —> eggs)* = 0.3-(0.5*0.4) = 0.1 and *Leverage(milk -> bread*)=0.4 - (0.5 * 0.4) = 0.2. It again confirms that milk and bread have a stronger association than milk and eggs.

Confidence is able to identify trustworthy rules, but it cannot tell whether a rule is coincidental.

3.3.4-Applications of Association Rules

The term *market basket analysis* refers to a specific implementation of association rules mining that many companies use for a variety of purposes, including these:

- Broad-scale approaches to better merchandising—what products should be included in or excluded from the inventory each month
- Cross-merchandising between products and high-margin or high-ticket items
- Physical or logical placement of product within related categories of products
- Promotional programs—multiple product purchase incentives managed through a loyalty card program

Besides market basket analysis, association rules are commonly used for recommender systems and clickstream analysis.

Many online service providers such as Amazon and Netflix use recommender systems. Recommender systems

can use association rules to discover related products or identify customers who have similar interests. For example, association rules may suggest that those customers who have bought product A have also bought product B, or those customers who have bought products A, B, and C are more similar to this customer. These findings provide opportunities for retailers to cross-sell their products.

Clickstream analysis refers to the analytics on data related to web browsing and user clicks, which is stored on the client or the server side. Web usage log files generated on web servers contain huge amounts of information, and association rules can potentially give useful knowledge to web usage data analysts. For example, association rules may suggest that website visitors who land on page X click on links A, B, and C much more often than links D, E, and F. This observation provides valuable insight on how to better personalize and recommend the content to site visitors.

3.3.5-Validation and Testing

After gathering the output rules, it may become necessary to use one or more methods to validate the results in the business context for the sample dataset. The first approach can be established through statistical measures such as confidence, lift, and leverage. Rules that involve mutually independent items or cover few transactions are considered uninteresting because they may capture spurious relationships.

Confidence measures the chance that X and Y appear together in relation to the chance X appears. Confidence can be used to identify the interestingness of the rules.

Lift and leverage both compare the support of X and Y against their individual support. While mining data with association rules, some rules generated could be purely coincidental. For example, if 95% of customers buy X and 90% of customers buy Y, then X and Y would occur together at least 85% of the time, even if there is no relationship between the two.

Another set of criteria can be established through subjective arguments. Even with a high confidence, a rule may be considered subjectively uninteresting unless it reveals any unexpected profitable actions. For example, rules like {***paper}->{pencil***} may not be subjectively interesting or meaningful despite high support and confidence values. In contrast, a rule like ***{diaper}->{beer}*** that satisfies both minimum support and minimum confidence can be considered subjectively interesting because this rule is unexpected and may suggest a cross-sell opportunity for the retailer.

3.3.6-Diagnostics

Although the Apriori algorithm is easy to understand and implement, some of the rules generated are uninteresting or practically useless. Additionally, some of the rules may be generated due to coincidental relationships between the variables. Measures like confidence, lift, and leverage should be used along with human insights to address this problem.

Another problem with association rules is that, in Phase 3 and 4 of the Data Analytics Lifecycle, the team must specify the minimum support prior to the model execution, which may lead to too many or too few rules. In related research, a variant of the algorithm can use a predefined target range for the number of rules so that the algorithm can adjust the minimum support accordingly.

Apriori algorithm is one of the earliest and the most fundamental algorithms for generating association rules. The Apriori algorithm reduces the computational workload by only examining itemsets that meet the specified minimum threshold. However, depending on the size of the dataset, the Apriori algorithm can be computationally expensive. For each level of support, the algorithm requires a scan of the entire database to obtain the result. Accordingly, as the database grows, it takes more time to compute in each run. Here are some approaches to

improve Apriori's efficiency:

- **Partitioning:** Any itemset that is potentially frequent in a transaction database must be frequent in at least one of the partitions of the transaction database.
- **Sampling:** This extracts a subset of the data with a lower support threshold and uses the subset to perform association rule mining.
- **Transaction reduction:** A transaction that does not contain frequent fc-itemsets is useless in subsequent scans and therefore can be ignored.
- **Hash-based itemset counting:** If the corresponding hashing bucket count of a fc-itemset is below a certain threshold, the/c-itemset cannot be frequent.
- **Dynamic itemset counting:** Only add new candidate itemsets when all of their subsets are estimated to be frequent.

Chapter IV

Objectives:

To Study and Understand the following concept

- Regression
- Linear Regression
- Logistic Regression
- Additional Regression Models.

4-Regression Analysis

In general, regression analysis attempts to explain the influence that a set of variables has on the outcome of another variable of interest. Often, the outcome variable is called a *dependent variable* because the outcome depends on the other variables. These additional variables are sometimes called the *input variables* or the *independent variables*. Regression analysis is useful for answering the following kinds of questions:

- What is a person's expected income?
- What is the probability that an applicant will default on a loan?

Linear regression is a useful tool for answering the first question, and logistic regression is a popular method for addressing the second.

Regression analysis is a useful explanatory tool that can identify the input variables that have the greatest statistical influence on the outcome. With such knowledge and insight, environmental changes can be attempted to produce more favorable values of the input variables. For example, if it is found that the reading level of 10-year-old students is an excellent predictor of the students' success in high school and a factor in their attending college, then additional emphasis on reading can be considered, implemented, and evaluated to improve students' reading levels at a younger age.

4.1-Linear Regression

Linear regression is an analytical technique used to model the relationship between several input variables and a continuous outcome variable. A key assumption is that the relationship between an input variable and the outcome variable is linear. Although this assumption may appear restrictive, it is often possible to properly transform the input or outcome variables to achieve a linear relationship between the modified input and outcome variables.

A linear regression model is a probabilistic one that accounts for the randomness that can affect any particular outcome. Based on known input values, a linear regression model provides the expected value of the outcome

variable based on the values of the input variables, but some uncertainty may remain in predicting any particular outcome.

4.1.1-Use Cases

Linear regression is often used in business, government, and other scenarios. Some common practical applications of linear regression in the real world include the following:

- **Real estate:** A simple linear regression analysis can be used to model residential home prices as a function of the home's living area. Such a model helps set or evaluate the list price of a home on the market. The model could be further improved by including other input variables such as number of bathrooms, number of bedrooms, lot size, school district rankings, crime statistics, and property taxes
- **Demand forecasting:** Businesses and governments can use linear regression models to predict demand for goods and services. For example, restaurant chains can appropriately prepare for the predicted type and quantity of food that customers will consume based upon the weather, the day of the week, whether an item is offered as a special, the time of day, and the reservation volume. Similar models can be built to predict retail sales, emergency room visits, and ambulance dispatches.
- **Medical:** A linear regression model can be used to analyze the effect of a proposed radiation treatment on reducing tumor sizes. Input variables might include duration of a single radiation treatment, frequency of radiation treatment, and patient attributes such as age or weight.

4.1.2-Model Description

As the name of this technique suggests, the linear regression model assumes that there is a linear relationship between the input variables and the outcome variable. This relationship can be expressed as shown in Equation

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \ldots\ldots\ldots + \beta p\text{-}_1 x\ p\text{-}_1 + \mathcal{E}$$

where:

y is the outcome variable

x_j are the input variables, for j=1,2,...,p-1

β_0 is the value of y when each x_j equals zero

β_j is the change in y based on a unit change in x_j for j=1,2,...,p-1

$\mathcal{E}$ is a random error term that represents the difference in the linear model and a particular observed value for y.

Suppose it is desired to build a linear regression model that estimates a person's annual income as a function of two variables—age and education—both expressed in years. In this case, income is the outcome variable, and the input variables are age and education.

However, it is also obvious that there is considerable variation in income levels for a group of people with identical ages and years of education. This variation is represented by $\mathcal{E}$in the model. So, in this example, the model would be expressed as shown in Equation.

$$\text{Income} = \beta_0 + \beta_1 \text{Age} + \beta_2 \text{Education} + \mathcal{E}$$

Linear Regression Model (Ordinary Least Squares)

In the linear model, the β_2s represent the unknown p parameters. The estimates for these unknown parameters are chosen so that, on average, the model provides a reasonable estimate of a person's income based

on age and education. In other words, the fitted model should minimize the overall error between the linear model and the actual observations. Ordinary Least Squares (OLS) is a common technique to estimate the parameters.

To illustrate how OLS works, suppose there is only one input variable, x, for an outcome variable y. Furthermore, *n* observations of (x, *y)* are obtained and plotted in below Figure.

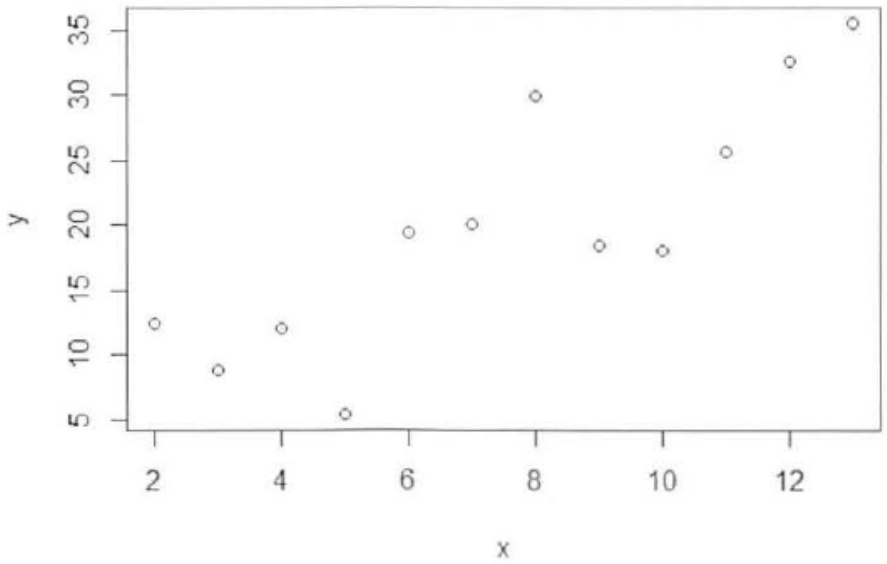


Figure 4.1. Scatterplot of y versus x

The goal is to find the line that best approximates the relationship between the outcome variable and the input variables. With OLS, the objective is to find the line through these points that minimizes the sum of the squares of the difference between each point and the line in the vertical direction. In other words, find the values of β_0 and β_1, such that the summation shown in Equation is minimized.

$$\sum_{i=1}^{n}[y_i-(\beta_0+\beta_1 x_i)]^2$$

The n individual distances to be squared and then summed are illustrated in below figure. The vertical lines represent the distance between each observed y value and the line $y = \beta_0 + \beta_1 x_1$

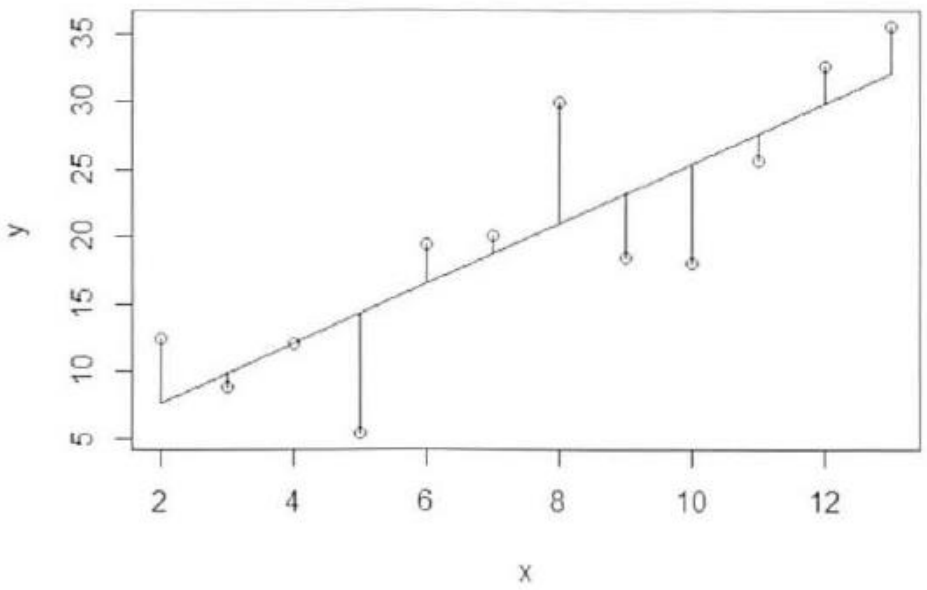


Figure 4.2: Scatterplot of y versus x with vertical distance from the observed points to a fitted line

Linear Regression Model (with Normally Distributed Errors)

In the normal model description, there were no assumptions made about the error term; no additional assumptions were necessary for OLS to provide estimates of the model parameters. However, in most linear regression analyses, it is common to assume that the error term is a normally distributed random variable with mean equal to zero and constant variance. Thus, the linear regression model is expressed as shown in Equation.

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \ldots\ldots\ldots + \beta_{p-1} x_{p-1} + \varepsilon$$

where:

y is the outcome variable

x_j are the input variables, for $j=1,2,...,p-1$

β_0 is the value of y when each x_j equals zero

β_j is the change in y based on a unit change in x_j for $j=1,2,...,p-1$

$\varepsilon \sim N(0,\sigma^2)$ and the εs are independent of each other

This additional assumption yields the following result about the expected value of y, E(y) for given $(x_1, x_2, \ldots x_{p-1})$:

$$\begin{aligned} E(y) &= E(\beta_0 + \beta_1 x_1 + \beta_2 x_2 \ldots + \beta_{p-1} x_{p-1} + \varepsilon) \\ &= \beta_0 + \beta_1 x_1 + \beta_2 x_2 \ldots + \beta_{p-1} x_{p-1} + E(\varepsilon) \\ &= \beta_0 + \beta_1 x_1 + \beta_2 x_2 \ldots + \beta_{p-1} x_{p-1} \end{aligned}$$

Because β_j and x_j are constants, the E(y) is the value of the linear regression model for the given $(x_1, x_2, \ldots x_{p-1})$. Furthermore, the variance of y, V(y), for given $(x_1, x_2, \ldots x_{p-1})$ is this:

$$V(y) = V(\beta_0 + \beta_1 x_1 + \beta_2 x_2 \ldots + \beta_{p-1} x_{p-1} + \varepsilon)$$
$$= 0 + V(\varepsilon) = \sigma^2$$

Thus, for a given (x_1, x_2,... x_{p-1}), y is normally distributed with mean $\beta_0 + \beta_1 x_1 + \beta_2 x_2 \ldots\ldots + \beta_{p-1} x_{p-1}$ and variance σ^2. For a regression model with just one input variable, below figure illustrates the normality assumption on the error terms and the effect on the outcome variable, y, for a given value of x.

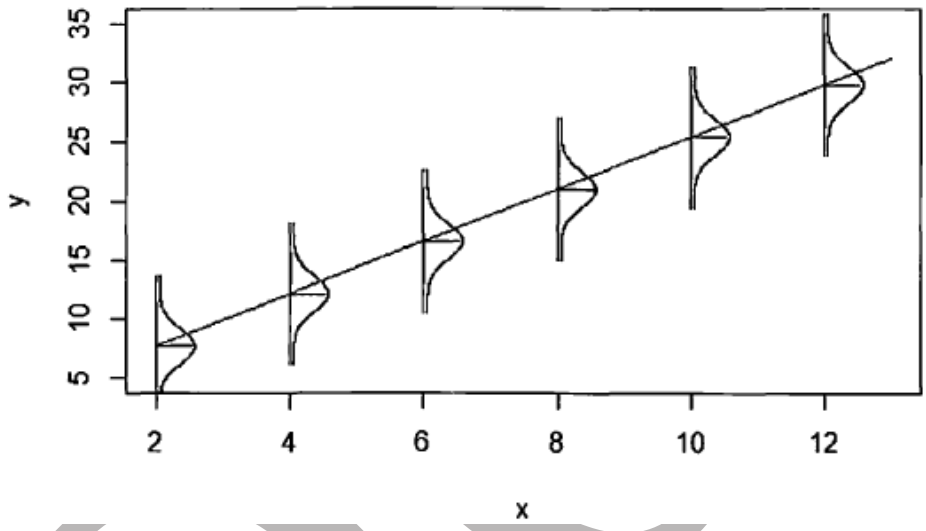


Figure 4.3: Normal distribution about y for a given value of x

4.1.3-Diagnostics

The use of hypothesis tests, confidence intervals, and prediction intervals is dependent on the model assumptions being true. Following are Some tools and techniques that can be used to validate a fitted linear regression model.

a-Evaluating the Linearity Assumption

A major assumption in linear regression modeling is that the relationship between the input variables and the outcome variable is linear. The most fundamental way to evaluate such a relationship is to plot the outcome variable against each input variable. If the relationship between *Age* and *Income* is represented as illustrated in Figure 4.3, a linear model would not apply.

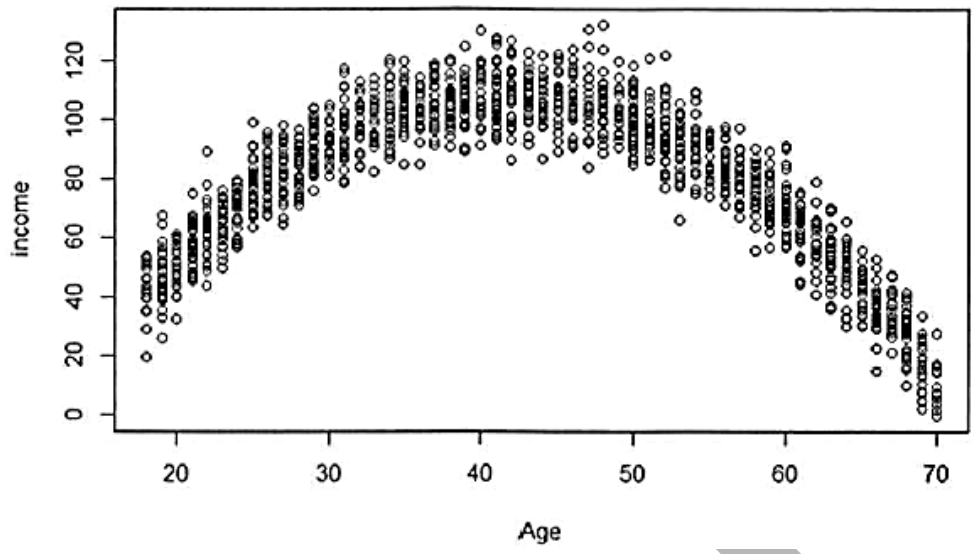


Figure 4.3: Income as a quadratic function of Age

In such a case, it is often useful to do any of the following:

- Transform the outcome variable.
- Transform the input variables.
- Add extra input variables or terms to the regression model.

Common transformations include taking square roots or the logarithm of the variables. Another option is to create a new input variable such as the age squared and add it to the linear regression model to fit a quadratic relationship between an input variable and the outcome.

b-Evaluating the Residuals

As stated previously, it is assumed that the error terms in the linear regression model are normally distributed with a mean of zero and a constant variance. If this assumption does not hold, the various inferences that were made with the hypothesis tests, confidence intervals, and prediction intervals are suspect.

c-Evaluating the Normality Assumption

The residual plots are useful for confirming that the residuals were centered on zero and have a constant variance. However, the normality assumption still has to be validated.

d-N-Fold Cross-Validation

To prevent overfitting a given dataset, a common practice is to randomly split the entire dataset into a training set and a testing set. Once the model is developed on the training set, the model is evaluated against the testing set. When there is not enough data to create training and testing sets, an N-fold cross-validation technique may be helpful to compare one fitted model against another. In N-fold cross-validation, the following occurs:

- The entire dataset is randomly split into N datasets of approximately equal size.
- A model is trained against N - 1 of these datasets and tested against the remaining dataset. A measure of the model error is obtained.
- This process is repeated a total of N times across the various combinations of N datasets taken N - 1 at a time. Recall:

$$\binom{N}{N-1} = N$$

- The observed N model errors are averaged over the N folds.

The averaged error from one mode! is compared against the averaged error from another model. This technique can also help determine whether adding more variables to an existing model is beneficial or possibly overfitting the data.

4.2-Logistic Regression

In linear regression modeling, the outcome variable is a continuous variable. When the outcome variable is categorical in nature, logistic regression can be used to predict the likelihood of an outcome based on the input variables. Although logistic regression can be applied to an outcome variable that represents multiple values, but we will examine the case in which the outcome variable represents two values such as true/false, pass/fail, or yes/no.

For example, a logistic regression model can be built to determine if a person will or will not purchase a new automobile in the next 12 months. The training set could include input variables for a person's age, income, and gender as well as the age of an existing automobile. The training set would also include the outcome variable on whether the person purchased a new automobile over a 12-month period. The logistic regression model provides the likelihood or probability of a person making a purchase in the next 12 months.

4.2.1-Use Cases

The logistic regression model is applied to a variety of situations in both the public and the private sector.

Some common ways that the logistic regression model is used include the following:

- **Medical:** Develop a model to determine the likelihood of a patient's successful response to a specific medical treatment or procedure. Input variables could include age, weight, blood pressure, and cholesterol levels.
- ***Finance:*** Using a loan applicant's credit history and the details on the loan, determine the probability that an applicant will default on the loan. Based on the prediction, the loan can be approved or denied, or the terms can be modified.
- ***Marketing:*** Determine a wireless customer's probability of switching carriers (known as churning) based on age, number of family members on the plan, months remaining on the existing contract, and social network contacts. With such insight, target the high-probability customers with appropriate offers to prevent churn.
- ***Engineering:*** Based on operating conditions and various diagnostic measurements, determine the probability of a mechanical part experiencing a malfunction or failure. With this, probability estimate, schedule the appropriate preventive maintenance activity.

4.2.2-Model Description

Logistic regression is based on the logistic function $f(y)$, as given in Equation 6-7.

$$f(y) = \frac{e^y}{1+e^y} \quad \text{for } -\infty < y < \infty \tag{6-7}$$

Note that as $y \to \infty$, $f(y) \to 1$, and as $y \to -\infty$, $f(y) \to 0$. So, as Figure 6-14 illustrates, the value of the logistic function $f(y)$ varies from 0 to 1 as y increases.

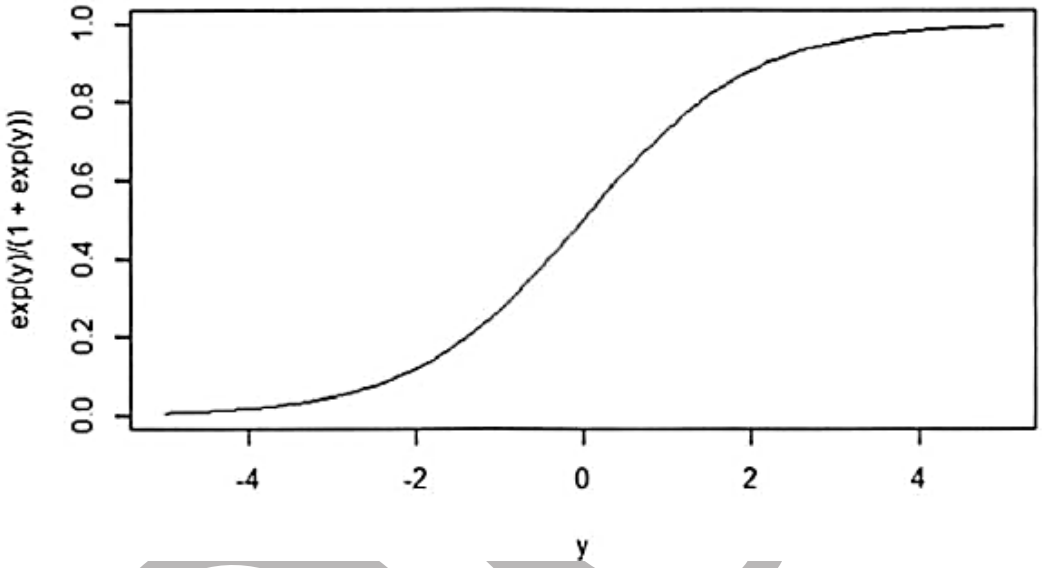


Figure 4.4. The logistic function

Because the range of $f(y)$ is (0, 1), the logistic function appears to be an appropriate function to model the probability of a particular outcome occurring. As the value of y increases, the probability of the outcome occurring increases. In any proposed model, to predict the likelihood of an outcome, y needs to be a function of the input variables. In logistic regression, y is expressed as a linear function of the input variables. In other words, the formula shown in Equation 6-8 applies.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \ldots + \beta_{p-1} x_{p-1} \quad (6\text{-}8)$$

Then, based on the input variables $x_1, x_2, \ldots, x_{p-1}$, the probability of an event is shown in Equation 6-9.

$$p(x_1, x_2, \ldots, x_{p-1}) = f(y) = \frac{e^y}{1+e^y} \quad \text{for } -\infty < y < \infty \quad (6\text{-}9)$$

Equation 6-8 is comparable to Equation 6-1 used in linear regression modeling. However, one difference is that the values of y are not directly observed. Only the value of $f(y)$ in terms of success or failure (typically expressed as 1 or 0, respectively) is observed.

Using p to denote $f(y)$, Equation 6-9 can be rewritten in the form provided in Equation 6-10.

$$ln\left(\frac{p}{1-p}\right) = y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \ldots + \beta_p x_{p-1} \quad (6\text{-}10)$$

The quantity $ln\left(\frac{p}{1-p}\right)$, in Equation 6-10 is known as the log odds ratio, or the logit of p. Techniques such as Maximum Likelihood Estimation (MLE) are used to estimate the model parameters. MLE determines the values of the model parameters that maximize the chances of observing the given dataset.

Customer Churn Example

A wireless telecommunications company wants to estimate the probability that a customer will churn (switch to a different company) in the next six months. With a reasonably accurate prediction of a person's likelihood of churning, the sales and marketing groups can attempt to retain the customer by offering various incentives. Data on 8,000 current and prior customers was obtained. The variables collected for each customer follow:

- *Age* (years)
- *Married*(true/false)
- *Duration* as a customer (years)
- *Churned_contacts*(count)—Number of the customer's contacts that have churned (count)
- *churned* (true/false)—Whether the customer churned

After analyzing the data and fitting a logistic regression model, *Age* and *Churned_contacts* were selected as the best predictor variables. Equation 6-11 provides the estimated model parameters.

$$y = 3.50 - 0.16 * \textbf{\textit{Age}} + 0.38 * \textbf{\textit{Churned\_contacts}} \quad (6.11)$$

Using the fitted model from Equation 4-1, below table provides the probability of a customer churning based on the customer's age and the number of churned contacts.

Table 4.1 Estimated churn probabilities

Customer	Age (Years)	Churned_Contacts	y	Prob. of Churning
1	50	1	–4.12	0.016
2	50	3	–3.36	0.034
3	50	6	–2.22	0.098
4	30	1	–0.92	0.285
5	30	3	–0.16	0.460
6	30	6	0.98	0.727
7	20	1	0.68	0.664
8	20	3	1.44	0.808
9	20	6	2.58	0.930

Based on the fitted model, there is a 93% chance that a 20-year-old customer who has had six contacts churn will also churn.

4.2.3-Diagnostics

Deviance and the Pseudo-R^2

In logistic regression, deviance is defined to be $-2 * logL$, where L is the maximized value of the likelihood function that was used to obtain the parameter estimates. In the R output, two deviance values are provided. The ***null deviance*** is the value where the likelihood function is based only on the intercept term ($y = \beta_0$). The ***residual deviance*** is the value where the likelihood function is based on the parameters in the specified logistic model, shown in Equation 6-12.

$$y = \beta_0 + \beta_1 * Age + \beta_2 * Churned\_contacts \quad (6\text{-}12)$$

A metric analogous to R^2 in linear regression can be computed as shown in Equation 6-13.

$$\text{pseudo-}R^2 = 1 - \frac{residual\ dev.}{null\ dev.} = \frac{null\ dev. - res.\ dev.}{null\ dev.} \quad (6\text{-}13)$$

The pseudo-R^2 is a measure of how well the fitted model explains the data as compared to the default model of no predictor variables and only an intercept term. A $pseudo - R^2$ value near 1 indicates a good fit over the simple null model.

Deviance and the Log-Likelihood Ratio Test

In the *pseudo* $-R^2$ calculation, the –2 multipliers simply divide out. So, it may appear that including such a multiplier does not provide a benefit. However, the multiplier in the deviance definition is based on the log-likelihood test statistic shown in Equation 6-14:

$$T = -2 * log\left(\frac{L_{null}}{L_{alt.}}\right) = -2 * \log(L_{null}) - (-2) * \log(L_{alt.}) \quad (6\text{-}14)$$

where T is approximately Chi-squared distributed (χ_k^2) with

k *degrees of freedom* $(df) = df_{null} - df_{alternate}$

The previous description of the log-likelihood test statistic applies to any estimation using MLE. As can be seen in Equation 6-15, in the logistic regression case,

$$T = \text{null deviance} - \text{residual deviance} \sim \chi^2_{p-1} \quad (6\text{-}15)$$

where p is the number of parameters in the fitted model.

So, in a hypothesis test, a large value of T would indicate that the fitted model is significantly better than the null model that uses only the intercept term.

In the churn example, the log-likelihood ratio statistic would be this:

$T = 8387.3 - 5359.2 = 3028.1$ with 2 degrees of freedom and a corresponding p-value that is essentially zero.

So far, the log-likelihood ratio test discussion has focused on comparing a fitted model to the default model of using only the intercept. However, the log-likelihood ratio test can also compare one fitted model to another.

Receiver Operating Characteristic (ROC) Curve

Logistic regression is often used as a classifier to assign class labels to a person, item, or transaction based on the predicted probability provided by the model. In the Churn example, a customer can be classified with the label called **Churn** if the logistic model predicts a high probability that the customer will churn. Otherwise, a **Remain** label is assigned to the customer. Commonly, 0.5 is used as the default probability threshold to distinguish between any two class labels. However, any threshold value can be used depending on the preference to avoid false positives (for example, to predict **Churn** when actually the customer will Remain) or false negatives (for example, to predict **Remain** when the customer will actually **Churn).**

Histogram of the Probabilities

It can be useful to visualize the observed responses against the estimated probabilities provided by the logistic regression. Figure 4-2 provides overlaying histograms for the customers who churned and for the customers who remained as customers. With a proper fitting logistic model, the customers.who remained tend to have a low probability of churning. Conversely, the customers who churned have a high probability of churning again. This histogram plot helps visualize the number of items to be properly classified or mis- dassified. In the Churn example, an ideal histogram plot would have the remaining customers grouped at the left side of the plot, the customers who churned at the right side of the plot, and no overlap of these two groups.

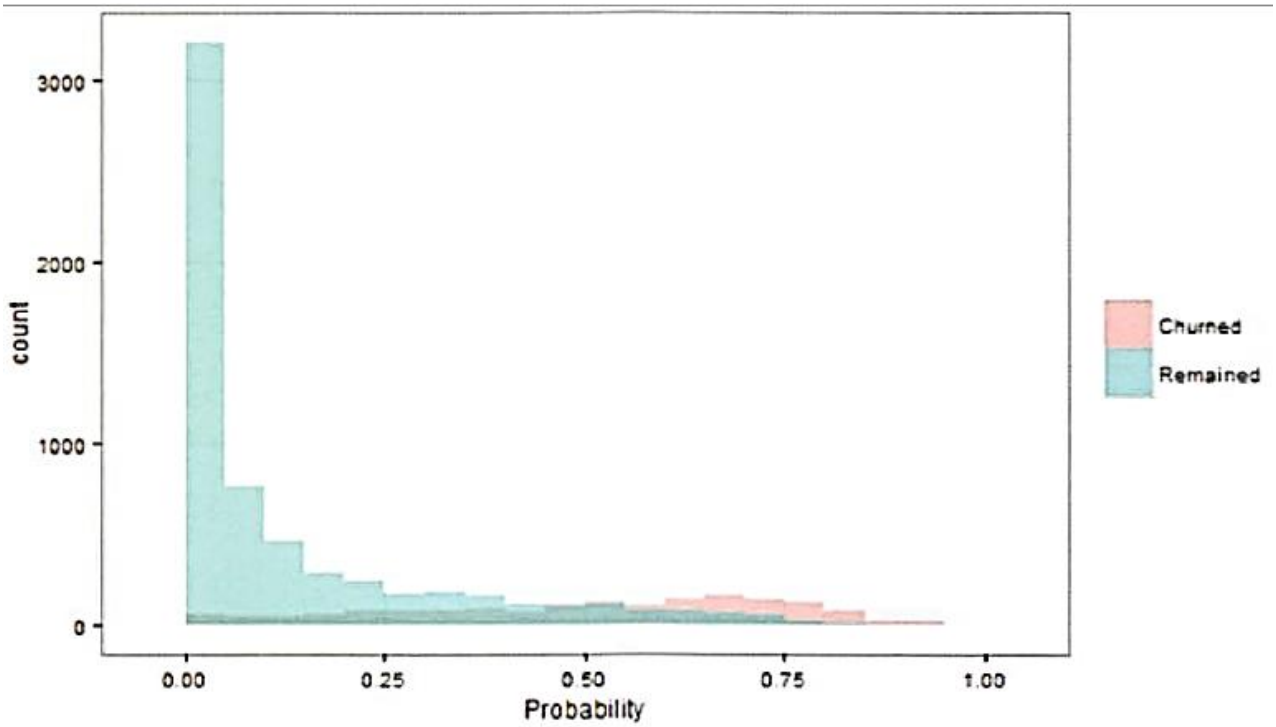


Figure4. 2 : Customer counts versus estimated churn probability

4.3-Reasons to Choose and Cautions

Linear regression is suitable when the input variables are continuous or discrete, including categorical data types, but the outcome variable is continuous. If the outcome variable is categorical, logistic regression is a better choice.

Both models assume a linear additive function of the input variables. If such an assumption does not hold true, both regression techniques perform poorly, Furthermore, in linear regression, the assumption of normally distributed error terms with a constant variance is important for many of the statistical inferences that can be considered. If the various assumptions do not appear to hold, the appropriate transformations need to be applied to the data.

Although a collection of input variables may be a good predictor for the outcome variable, the analyst should not infer that the input variables directly cause an outcome. For example, it may be identified that those individuals who have regular dentist visits may have a reduced risk of heart attacks. However, simply sending someone to the dentist almost certainly has no effect on that person's chance of having a heart attack. It is possible that regular dentist visits may indicate a person's overall health and dietary choices, which may have a more direct impact on a person's health.

Use caution when applying an already fitted model to data that falls outside the dataset used to train the model. The linear relationship in a regression model may no longer hold at values outside the training dataset. For example, if income was an input variable and the values of income ranged from $35,000 to $90,000, applying the model to incomes well outside those incomes could result in inaccurate estimates and predictions.

If several of the input variables are highly correlated to each other, the condition is known as *multicollinearity*. Multicollinearity can often lead to coefficient estimates that are relatively large in absolute magnitude and may be of inappropriate direction (negative or positive sign). When possible, the majority of these correlated variables should be removed from the model or replaced by a new variable that is a function of the correlated variables.

References:

1) Big Data and Analytics, Subhashini Chellappan Seema Acharya, Wiley First addition
2) Data Analytics with Hadoop, *An Introduction for Data Scientists, Benjamin Bengfort and Jenny Kim* O'Reilly 2016
3) Big Data and Hadoop V.K Jain KhannaPublishing First 2018

Unit V
Chapter I

Objectives

To Study and Understand the following concept

- **Distributed Analysis and Patterns,**
- **Computing with Keys**
- **Design Patterns**
- **Last-Mile Analytics**

9-Distributed Analysis and Patterns

MapReduce and Spark allow developers and data scientists the ability to easily conduct *data parallel* operations, where data is distributed to multiple processing nodes and computed upon simultaneously, then reduced to a final output. YARN provides simple *task parallelism* by allowing a cluster to perform multiple different operations simultaneously by allocating free computational resources to perform individual tasks. Parallelism reduces the amount of time required to perform a single computation, thereby unlocking datasets that are measured in petabytes, analyzed at thousands of records per second, or composed of multiple heterogeneous data sources. However, most parallel operations like the ones described to this point are simple, leading to the question, how can data scientists conduct advanced data analysis at scale?

If two operations can be computed simultaneously, but the sequence must maintain the property that data is fed sequentially from an input data source to a final output. For that reason, data flows are described as *directed acyclic graphs* (DAGs). It is important, therefore, to realize that if an algorithm, analysis, or other non-trivial computation can be expressed as a DAG, then it can be parallelized on Hadoop.

Algorithms that cannot be described as a directed data flow include those that maintain or update a single data structure throughout the course of computation (requiring some shared memory) or computations that are dependent on the results of another at intermediate steps (requiring intermediate interprocess communication).

There are tools and techniques that address requirements for cyclicity, shared memory, or interprocess communication in both MapReduce and Spark, but to make use of these tools, algorithms must be rewritten to a distributed form.

It is because of the widespread use of this approach that Hadoop is often said to be a preprocessor that unlocks the potential of large datasets by reducing them into increasingly manageable chunks through every operation. A common rule of thumb is use either MapReduce or Spark to articulate data down to a computational space that can fit into 128 GB of memory. This rule is often called “last-mile” computing because it moves data from an extremely large space to a place close enough, the last mile, that allows for accurate analyses or application-specific computations.

9.1-Computing with Keys

The first step toward understanding how data flows work in practice is to understand the relationship between key/value pairs and parallel computation. In MapReduce, *all data* is structured as key/value pairs in both the map and reduce stages. The key requirement relates primarily to reduction, as aggregation is grouped by the key, and

parallel reduction requires partitioning of the *keyspace*—in other words, the domain of key values such that a reducer task sees all values for that key.

Although often ignored, keys allow the computation to work on sets of data simultaneously. Therefore, a data flow expresses the relation of one set of values to another, which should sound familiar, especially presented in the context of more traditional data management—structured queries on a relational database. Similar to how you would not run multiple individual queries for an analysis of different dimensions on a database like PostgreSQL, MapReduce and Spark computations look to perform grouping operations in parallel, as shown by the mean computation grouped by key in Figure 9-1

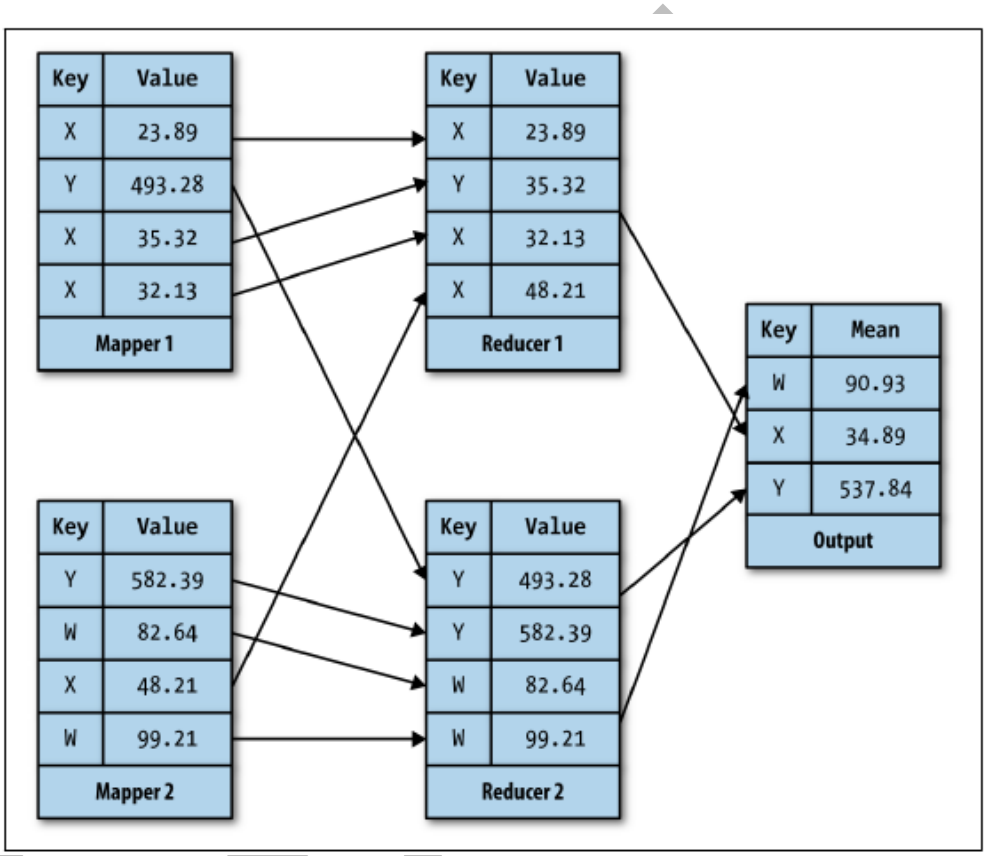


Figure 9.1 Keys allow parallel reduction by partitioning the keyspace to multiple

Moreover, keys can maintain information that has already been reduced at one stage in the data flow, automatically parallelizing a result that is required for the next step in computation. This is done using compound keys. Most Spark applications require them for their analyses, primarily using groupByKey, aggregateByKey, sortByKey and reduceByKey actions to collect and reduce.

9.1.1-Compound Keys

Keys need not be simple primitives such as integers or strings; instead, they can be compound or complex types so long as they are both *hashable* and *comparable.* Comparable types must at the very least expose some mechanism to determine equality and some method of ordering. Comparison is usually accomplished by mapping some type to a numeric value (e.g., months of the year to the integers 1-12) or through a lexical ordering. Hashable types in Python are any immutable type, the most notable of which is the tuple. Tuples can contain mutable types (e.g., a tuple of lists), however, so a hashable tuple is one that is composed of immutable types. Mutable types such as lists and dictionaries can be transformed into immutable tuples:

```
# Transform a list into a tuple
key = tuple(['a', 'b', 'c'])

# Transform a dictionary into a tuple of tuples
key = {'a': 1, 'b': 2}
key = tuple(key.items())
```

Compound keys are used in two primary ways: to facet the keyspace across multiple dimensions and to carry key-specific information forward through computational stages that involve the values alone. Consider web log records of the following form:

```
local - - [30/Apr/1995:21:18:07 -0600] "GET 7448.html HTTP/1.0" 404 -
local - - [30/Apr/1995:21:18:42 -0600] "GET 7448.html HTTP/1.0" 200 980
remote - - [30/Apr/1995:21:22:56 -0600] "GET 4115.html HTTP/1.0" 200 1363
remote - - [30/Apr/1995:21:26:29 -0600] "GET index.html HTTP/1.0" 200 2881
```

Web log records are a typical data source of big data computations on Hadoop, as they represent per-user clickstream data that can be easily mined for insight in a variety of domains; they also tend to be very large, dynamic semistructured datasets, well suited to operations in Spark and MapReduce. Initial computation on this dataset requires a frequency analysis; for example, we can decompose the text into two daily time series, one for local traffic and the other for remote traffic using a compound key:

```
import re
from datetime import datetime

# Parse datetimes in the log record
dtfmt = "%d/%b/%Y:%H:%M:%S %z"

# Parse log records using a regular expression
linre = re.compile(r'^(\w+) \- \- \[(.+)\] "(.+)" (\d+) ([\d\-]+)$')

def parse(line):
    # Match the log record against our regular expression
    match = linre.match(line)
    if match is not None:
        # The regular expression has groups to extract the source, timestamp,
        # the request, the status code, and the byte size of the response.
        parts = match.groups()

        # Parse the datetime and return the source, along with the year and day.
        date  = datetime.strptime(parts[1], dtfmt).timetuple()
        return (parts[0], date.tm_year, date.tm_yday)
```

Mapping yields the following data from the preceding dataset:

```
('local', 1995, 120)    1
('local', 1995, 120)    1
('remote', 1995, 120)    1
('remote', 1995, 120)    1
```

Compound data serialization

The final consideration when using compound keys (and complex values) is to understand *serialization* and *deserialization* of the compound data. *Serialization* is the process of turning an object in memory into a stream of bytes such that it can be written to disk or transmitted across the network *(deserialization* is the reverse process). This process is essential, particularly in MapReduce, as keys and values are written (usually as strings) to disk between map and reduce phases

By default in Spark, the Python API uses the pickle module for serialization, which means that any data structures you use must be pickle-able. With MapReduce Streaming, you must serialize both the key and the value as a string, separated by a specified character, by default a tab (\t).

One common first attempt is to simply serialize an immutable type (e.g., a tuple) using the built-in str function, converting the tuple into a string that can be easily pickled or streamed. The problem then shifts to deserialization; using the ast (abstract syntax tree) in the Python standard library, we can use the literal_eval function to evaluate stringified tuples back into Python tuple types as follows:

```
import ast

def map(key, val):
    # Parse the compound key, which is a tuple.
    key = ast.literal_eval(key)

    # Write out the new key as a string
    return (str(key), val)
```

As both keys and values get more complex. Other data structures for serialization is Base64-encoded JSON because it is compact, uses only ASCII characters, and is easily serialized and deserialized with the standard library as follows:

```
import json
import base64

def serialize(data):
    """
    Returns the Base64-encoded JSON representation of the data (keys or values)
    """
    return base64.b64encode(json.dumps(data))

def deserialize(data):
    """
    Decodes Base64 JSON-encoded data
    """
    return json.loads(base64.b64decode(data))
```

However, take care when using more complex serial representations; often there is a trade-off in the computational complexity of serialization versus the amount of space used.

9.1.2-Keyspace Patterns

The notion of computing with keys allows you to manage sets of data and their relations. However, keys are also a primary piece of the computation, and as such, they must be managed in addition to the data. There several

patterns that impact the *keyspace*, specifically the explode, filter, transform, and identity patterns.

For the following examples, we will consider a dataset of orders whose key is the order ID, customer ID, and timestamp, and whose value is a list of universal product codes (UPCs) for the products purchased in the order as follows:

```
1001, 1063457, 2014-09-16 12:23:33, 098668259830, 098668318865
1002, 0171488, 2014-12-11 03:05:03, 098668318865
1003, 1022739, 2015-01-03 13:01:54, 098668275427, 098668331789, 098668274321
```

Transforming the keyspace

The most common key-based operation is a transformation of the input key domain, which can be conducted either in a map or a reduce. The most common transformation functions are direct assignment, compounding, splitting, and inversion.

Direct assignment drops the input key, which is usually entirely ignored, and constructs a new key from the input value or another source (e.g., a random key). Compounding constructs or adds to a compound key, increasing the faceting of the key relation. Splitting breaks apart a compound key and uses only a smaller piece of it. It is, however, appropriate to also drop unneeded data and eliminate extraneous information via compounding or splitting.

For example, in order to sort a dataset by value rather than by key, it is necessary to first map the inversion of the key and value, perform a sortByKey or utilize the shuffle and sort in MapReduce, then reinvert in the reduce or with another map. Consider a job to sort our orders by the number of products in each order, along with the date, which will use all of the keyspace transformations identified earlier

```
# Load orders into an RDD and parse the CSV
orders = sc.textFile("orders.csv").map(split)

# Key assignment: (orderid, customerid, date), products
orders = orders.map(lambda r: ((r[0], r[1], r[2]), r[3:]))

# Compute the order size and split the key to orderid, date
orders = orders.map(lambda (k, v): ((k[0], parse_date(k[2])), len(v)))

# Invert the key and value to sort
orders = orders.map(lambda (k, v): ((v, k[1]), k[0]))

# Sort the orders by key
orders = orders.sortByKey(ascending=False)

# Reinvert the key/value space so that we key on order ID again
orders = orders.map(lambda (k,v ): (v, k))

# Get the top ten order IDs by size and date
print orders.take(10)
```

This example is perhaps a bit verbose for the required task, but it does demonstrate each type of transformation as follows:

1. First, the dataset is loaded from a CSV using the split method.

2. At this point, orders is only a collection of lists, so we assign keys by breaking the value into the IDs and date as the key, and associate it with the list of products as the value.
3. The next step is to get the length of the products list (number of products ordered) and to parse the date, using a closure that wraps a date format for date time.strptime; note that this method splits the compound key and eliminates the customer ID, which is unnecessary.
4. In order to sort by order size, we need to invert the size value with the key, also splitting the date from the key so we can also sort by date.
5. After performing the sort, this function reinverts so that each order can be identified by size and date.

The following snippet demonstrates what happens to the first record throughout each map in the Spark job:

```
0. "1001, 1063457, 2014-09-16 12:23:33, 098668259830, 098668318865"
1. [1001, 1063457, 2014-09-16 12:23:33, 098668259830, 098668318865]
2. ((1001, 1063457, 2014-09-16 12:23:33), [098668259830, 098668318865])
3. ((1001, datetime(2014, 9, 16, 12, 23, 33), 2)
4. ((2, datetime(2014, 9, 16, 12, 23, 33)), 1001)
5. (1001, (2, datetime(2014, 9, 16, 12, 23, 33)))
```

Through this series of transformations, the client program can then take the top 10 orders by size and date, and print them out after the distributed computation.

The explode mapper

The explode mapper generates multiple intermediate key/value pairs for a single input key. Generally, this is done by a combination of a key shift and splitting of the value into multiple parts. An explode mapper can also generate many intermediate pairs by dividing a value into its constituent parts and reassigning them with the key. We can explode the list of products per order value to order/product pairs, as in the following code:

```
def order_pairs(key, products):
    # Returns a list of order id, product pairs
    pairs = []

    for product in products:
        pairs.append((key[0], product))

    return pairs

orders = orders.flatMap(order_pairs)
```

Applying this mapper to our input dataset produces the following output:

```
1001, 098668259830
1001, 098668318865
1002, 098668318865
1003, 098668275427
1003, 098668331789
1003, 098668274321
```

Note the use of the flatMap operation on the RDD, which is specifically designed for explode mapping. It operates similarly to the regular map; however, the function can yield a sequence instead of a single item, which is then chained into a single collection (rather than an RDD of lists).

The filter mapper

Filtering is often essential to limit the amount of computation performed in a reduce stage, particularly in a big

data context. It is also used to partition a computation into two paths of the same data flow, a sort of data-oriented branching in larger algorithms that is specifically designed for extremely large datasets.

Spark provides a filter operation that takes a function and transforms the RDD such that only elements on which the function returns True are retained. This example shows a more advanced use of a closure and a general filter function that can take any year. The partial function creates a closure whose year argument to year_filter is always 2014, allowing for a bit more versatility. MapReduce code is similar but requires a bit more logic:

```
def YearFilterMapper(Mapper):

    def __init__(self, year, **kwargs):
        super(YearFilterMapper, self).__init__(**kwargs)
        self.year = year

    def map(self):
        for key, value in self:
            if parse_date(key[2]).year == self.year:
                self.emit(key, value)

if __name__ == "__main__":
    mapper = YearFilterMapper(2014)
    mapper.map()
```

It is completely acceptable for a mapper to not emit anything, therefore the logic for a filter mapper is to only emit when the condition is met.

The identity pattern

The final keyspace pattern that is commonly used in MapReduce is the Identity function. This is simply a pass-through, such that identity mappers or reducers return the same value as their input (e.g., as in the *identity function*, f(x) = x). Identity mappers are typically used to perform multiple reductions in a data flow. When an identity reducer is employed in MapReduce, it makes the job the equivalent of a sort on the keyspace. Identity mappers and reducers are implemented simply as follows:

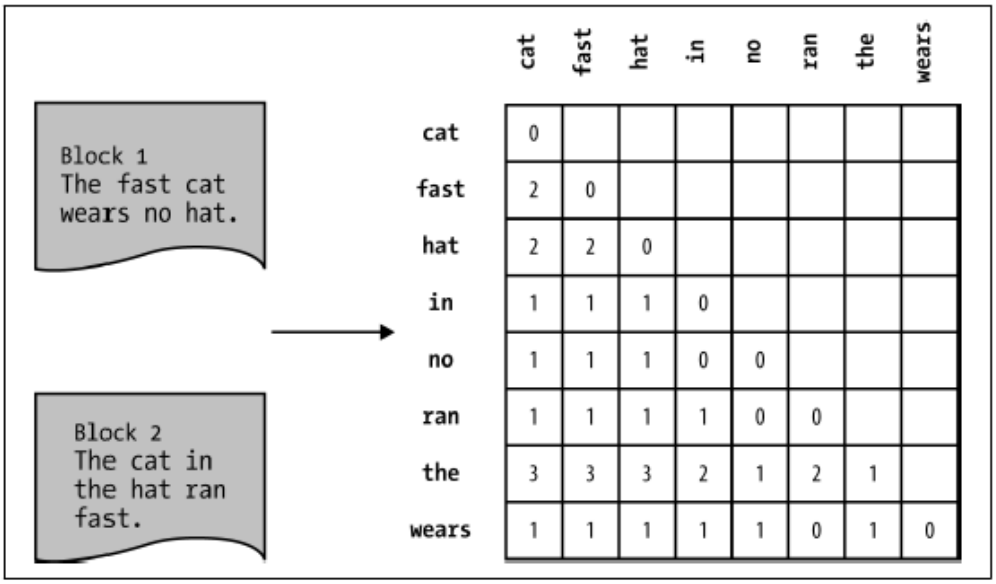
```
class IdentityMapper(Mapper):

    def map(self):
        for key, value in self:
            self.emit(key, value)


class IdentityReducer(Reducer):

    def reduce(self):
        for key, values in self:
            for value in values:
                self.emit(key, value)
```

Identity reducers are generally more common because of the optimized shuffle and sort in MapReduce. However, identity mappers are also very important, particularly in chained MapReduce jobs where the output of one reducer must immediately be reduced again by a secondary reducer.

9.1.3-Pairs versus Stripes

There are two ways that matrices are commonly represented: by *pairs* and by *stripes*. Both pairs and stripes are examples of key-based computation. To explain the motivation behind this example, consider the problem of building a word co-occurrence matrix for a text-based corpus.

The word co-occurrence matrix as shown in Figure 9-2 is a square matrix of size NxN, where N is the vocabulary (the number of unique words) in the corpus. Each cell Wij contains the number of times both word w_i and word w_j appear together in a sentence, paragraph, document, or other fixed-length window. This matrix is sparse, particularly with aggressive stopword filtering because most words only co-occur with very few other words on a regular basis.



	cat	fast	hat	in	no	ran	the	wears
cat	0							
fast	2	0						
hat	2	2	0					
in	1	1	1	0				
no	1	1	1	0	0			
ran	1	1	1	1	0	0		
the	3	3	3	2	1	2	1	
wears	1	1	1	1	1	0	1	0

Figure 9.2 A word co-occurrence matrix demonstrates the frequency of terms appearing together in the same block of text such as a sentence

The *pairs* approach maps every cell in the matrix to a particular value, where the pair is the compound key i, j. Reducers therefore work on per-cell values to produce a final, cell-by-cell matrix. This is a reasonable approach, which yields output where each *Wij* is computed upon and stored separately. Using a sum reducer, the mapper is as follows:

```
from itertools import combinations

class WordPairsMapper(Mapper):

    def map(self):
        for docid, document in self:
            tokens = list(self.tokenize(document))
            for pair in combinations(sorted(tokens), 2):
                self.emit(pair, 1)
```

Here is the input:

```
"See Spot run, run Spot, run!"
```

The pairs of the word co-occurrence matrix would be aggregated as follows:

```
(run, run), 6
(run, see), 3
(run, spot), 6
(see, run), 3
(see, spot), 2
(spot, run), 6
(spot, see), 2
(spot, spot), 1
```

While the pairs approach is easy to implement and understand, it causes a lot of intermediate pairs that must be transmitted across the network both during the MapReduce shuffle and sort phase, and during groupByKey operations to shuffle values between partitions in an RDD. Moreover, the pairs approach is not well suited to computations that require an entire row (or column) of data.

The stripes approach was initially conceived as an optimization to reduce the number of intermediate pairs and reduce network communication in order to make jobs faster. However, it also quickly became an essential tool in many algorithms that need to perform fast per-element computations—for example, relative frequencies or other statistical operations. Instead of pairs, a per-term associative array (a Python dictionary) is constructed in the mapper and emitted as a value:

```
from collections import Counter

class WordStripeMapper(Mapper):

    def map(self):
        for docid, document in self:
            tokens = list(self.tokenize(document))

            for i, term in enumerate(tokens):
```

```
                # Create a new stripe for each term
                stripe = Counter()

                for j, token in enumerate(tokens):
                    # Don't count the term's co-occurrence with itself
                    if i != j:
                        stripe[token] += 1

                # Emit the term and the stripe
                self.emit(term, stripe)


class StripeSumReducer(Reducer):

    def reduce(self):
        for key, values in self:
            stripe = Counter()

            # Add all the mapped counters together
            for value in values:
                for token, count in value.iteritems():
                    # For each token, count add the collector stripe
                    stripe[token] += count

            self.emit(key, stripe)
```

The stripes approach is not only more compact in its representation, but also generates fewer and simpler intermediary keys, thus optimizing sorting and shuffling of data or other optimizations. However, the stripes object is heavier, both in terms of processing time as well as the serialization requirements, particularly if the stripes get very large. There is a limit to the size of a stripe, particularly in very dense matrices, which may take a lot of memory to track single occurrences.

9.2- Design Patterns

Design patterns are a special term in software design: generic, reusable solutions for a particular programming challenge. We can explore functional design patterns for solving parallel computations in both MapReduce and Spark. These patterns show a generic strategy and principle that can be used in more complex or domain-specific roles.

Donald Miner and Adam Shook explore 23 design patterns for common MapReduce jobs. They loosely categorize them as follows:

Summarization

Provide a summary view of a large dataset in terms of aggregations, grouping, statistical measures, indexing, or other high-level views of the data.

Filtering

Create subsets or samples of the data based on a fixed set of criteria, without modifying the original data in any way.

Data Organization

Reorganize records into a meaningful pattern in a way that doesn't necessarily imply grouping. This task is useful as a first step to further computations.

Joins

Collect related data from disparate sources into a unified whole.

Metapatterns

Implement job chaining and job merging for complex or optimized computations. These are patterns associated with other patterns.

Input and output

Transform data from one input source to a different output source using data manipulation patterns, either internal to HDFS or from external sources.

9.2.1-Summarization

Summarization attempts to describe the largest amount of information about a dataset as simply as possible. We are accustomed to executive summaries that highlight the primary take-aways of a longer document without getting into the details. Similarly, descriptive statistics attempt to summarize the relationships between observations by measuring their central tendency (mean, median), their dispersion (standard deviation), the shape of their distribution (skewness), or the dependence of variables on each other (correlation).

MapReduce and Spark in principle apply a sequence of summarizations distilling the most specific form of the data (each individual record) to a more general form. Broadly speaking, we are most familiar with summarization as characterized by the following operations:

- Aggregation (collection to a single value such as the mean, sum, or maximum)
- Indexing (the functional mapping of a value to a set of values)
- Grouping (selection or division of a set into multiple sets)

Aggregation

An aggregation function in the context of MapReduce and Spark is one that takes two input values and produces a single output value and is also *commutative* and *associative* so that it can be computed in parallel. Addition and multiplication are commutative and associative, whereas subtraction and division are not.

Aggregation is the general application of an operation on a collection to create a smaller collection (gathering together), and reduction is generally considered an operation that reduces a collection into a single value. Aggregation can also be thought of as the application of a series of smaller reductions. With this context, it's easy to see why associativity and commutativity are necessary for parallelism.

Consider the standard dataset descriptors: mean, median, mode, minimum, maximum, and sum. Of these, *summation, minimum,* and *maximum* are easily implemented because they are both associative and commutative. *Mean, median,* and *mode,* however, are not. Although there are parallel approximations for these computations, it is important to be aware that some care should be taken when performing these types of analyses.

Statistical summarization

We can simplify and summarize large datasets by grouping instances into keys and describing the per-key properties. Rather than implementing a MapReduce job for each descriptive metric individually (costly), we're going to run all six jobs together in a single batch, computing the count, sum, mean, standard deviation, and range (minimum and maximum).

The basic strategy will be to map a collection of counter values for each computation we want to make on a per-key basis. The reducer will then apply each operation independently to each item in the value collection, using each as necessary to compute the final output (e.g., mean depends on both a count and a sum). Here is the basic outline for such a mapper:

```
class StatsMapper(Mapper):

    def map(self):
        for key, value in self:
            try:
                value = float(value)
                self.emit(key, (1, value, value ** 2))
            except ValueError:
                # Could not parse the value, ignore.
                pass
```

In this case, the three operations that will be directly reduced are count, sum, and sum of squares. Therefore, this mapper emits on a per-key basis, a 1 for count, the value for summation, and the square of the value for the sum of the squares. The reducer uses the count and sum to compute the mean, the value to compute the range, and the count, sum, and sum of squares to compute the standard deviation as follows:

```
from ast import literal_eval as make_tuple

class StatsReducer(Reducer):

    def reduce(self):
        for key, values in self:
            # Parse the values from the mapper
            values = make_tuple(values)

            count   = 0
            delay   = 0.0
            square  = 0.0
            minimum = None
            maximum = None

            for value in values:
                count  += value[0]
                delay  += value[1]
                square += value[2]

            if minimum is None or value[1] < minimum:
                minimum = value[1]

            if maximum is None or value[1] > maximum:
                maximum = value[1]

        mean   = delay / float(count)
        stddev = math.sqrt((square-(delay**2)/count)/count-1)

        self.emit(key, (count, mean, stddev, minimum, maximum))
```

The reducer utilizes the ast.literal_eval mechanism of deserialization to parse the value tuple, then performs a *single* loop over the data values to compute the various sums, minimums, and maximums.

Our mapper must extend the value with minimum and maximum counters, such that the minimum and maximum values are tracked with each value through the reduction as follows:

```
def counters(item):
    """
    Parses a key/value pair into the key and summary counters.
    A counter is as follows: (count, total, square, minimum, maximum).
    """
    key, value = item # Break apart the item tuple

    try:
        value = float(value)
        self.emit(key, (1, value, value ** 2, value, value))
    except ValueError:
        # Could not parse the value, ignore.
        pass

def aggregation(first, second):
    """
    For two (key, counter) items, perform summary aggregations.
    """
    count1, total1, squares1, min1, max1 = first
    count2, total2, squares2, min2, max2 = second
```

```
        minimum = min((min1, min2))
        maximum = max((max1, max2))
        count   = count1 + count2
        total   = total1 + total2
        squares = squares1 + squares2

        return (count, total, squares, minimum, maximum)

    def summary(aggregate):
        """
        Compute summary statistics from aggregation.
        """
        (key, (count, total, square, minimum, maximum)) = aggregate

        mean   = total / float(count)
        stddev = math.sqrt((square-(total**2)/count)/count-1)

        return (key, (count, mean, stddev, minimum, maximum))


    def main(sc):
        """
        Primary analysis mechanism for Spark application
        """

        # Given a dataset of key/value pairs, map to counters
        dataset = dataset.map(counters)

        # Perform summary aggregation by key
        dataset = dataset.reduceByKey(aggregation)
        dataset = dataset.map(summary)

        # Write the results out to disk
        dataset.saveAsTextFile("dataset-summary")
```

We can't simply perform our final computation during the aggregation, and another map is needed to finalize the summarization across the (much smaller) aggregated RDD.

The describe example provides a useful pattern for computing multiple features simultaneously and returning them as a vector. This pattern is reused often, particularly in the machine learning context, where multiple procedures might be required in order to produce an instance to train on (e.g., quadratic computations, normalization, imputation, joins, or more specific machine learning tasks). Understanding the difference between aggregation implementations in MapReduce versus Spark can make a lot of difference in tracking down bugs and porting code from MapReduce to Spark and vice versa.

9.2.2-Indexing

In contrast to aggregation-based summarization techniques, *indexing* takes a many- to-many approach. While aggregation collects several records into a single record, indexing associates several records to one or more indices. In databases, an index is a specialized data structure that is used for fast lookups, usually a binary-tree (B-Tree). In Hadoop/Spark, indices perform a similar function, though rather than being maintained and updated, they are typically generated as a first step to downstream computation that will require fast lookups.

Text indexing has a special place in the Hadoop algorithm pantheon due to Hadoop's original intended use for creating search applications. When dealing with only a small corpus of documents, it may be possible to scan the documents looking for the search term like grep does. However, as the number of documents and queries increases, this quickly becomes unreasonable. In this section, we take a look at two types of text- based indices, the more common inverted index, as well as term frequency-inverse document frequency (TF-IDF), a numerical statistic that is associated with an index and is commonly used for machine learning.

Inverted index

An *inverted index* is a mapping from an index term to locations in a set of documents (in contrast to forward indexing, which maps from documents to index terms). In full text search, the index terms are search terms: usually words or numbers with stopwords removed (e.g., very common words that are meaningless in search). Most search engines also employ some sort of stemming or lemmatization: multiple words with the same meaning are categorized into a single word class (e.g., "running", "ran", "runs" is indexed by the single term "run").

The search example shows the most common use case for an inverted index: it quickly allows the search algorithm to retrieve the subset of documents that it must rank and return without scanning every single document. For example, for the query "running bear", the index can be used to look up the intersection of documents that contain the term "running" and the term "bear". A simple ranking system might then be employed to return documents where the search terms are close together rather than far apart in the document (though obviously modern search ranking systems are far more complex than this).

The search example can be generalized, however, to a machine learning context. The index term does not necessarily have to be text; it can be any piece of a larger record. Moreover, the task of using an index to simplify or speed up downstream computation (like the ranking) is common. Depending on how the index is created, there can be a trade-off between performance and accuracy, or, given a stochastic index, between precision and recall.

we would use an identity reducer and the following mapper (note that the same algorithm is easily implemented with Spark):

```
class CharacterIndexMapper(Mapper):

    def map(self):
        for row in self:
            row = row.split("\t")            # split the tab parts
            if not len(row) >= 3: continue   # ensure we have data

            if row[1] != "":
                # If we have a character, emit the name and the docid/lineno.
                self.emit(row[1], row[0])
```

The output of the character index job is a list of character names, each of which corresponds to a list of lines where that character starts speaking. This can be used as a lookup table or as input to other types of analysis.

TF-IDF

Term frequency-inverse document frequency (TF-IDF) is now probably the most commonly used form of text-based summarization and is currently the most commonly used feature of documents in text-based machine learning. TF-IDF is a metric that defines the relationship between a term (a word) and a document that is part of a larger corpus. In particular, it attempts to define how important that word is to that particular document given the word's relative frequency in other documents.

We include this algorithm with indexing for a similar reason that we included the simpler inverted indexing example: it creates a data structure that is typically used for downstream computations and machine learning. Moreover, this more complex example highlights something we've only touched upon in other sections: the use of job chaining to compute a single algorithm. With that in mind, let's take a look at the MapReduce implementation of TF-IDF.

Typically, the tf-idf weight is composed by two terms: the first computes the normalized Term Frequency (TF), aka. the number of times a word appears in a document, divided by the total number of words in that document; the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

- **TF: Term Frequency**, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

  TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

- **IDF: Inverse Document Frequency**, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

  IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

Consider a document containing 100 words wherein the word *cat* appears 3 times. The term frequency (i.e., tf) for *cat* is then (3 / 100) = 0.03. Now, assume we have 10 million documents and the word *cat* appears in one thousand of these. Then, the inverse document frequency (i.e., idf) is calculated as log(10,000,000 / 1,000) = 4. Thus, the Tf-idf weight is the product of these quantities: 0.03 * 4 = 0.12.

9.2.3-Filtering

Filtering is one of the primary methods of coarse-grained data reduction for downstream computation. Unlike aggregation methods, which reduce the input space through a high-level overview over a set of groups, filtering is intended to reduce the computational space through omission. In fact, many filtering tasks are a perfect fit for map-only jobs, which do not require reducers because mappers are so well suited to this task. This can be considered filtering by predicate or by selection, similar to a where clause in a SQL statement.

Other filtering tasks may leverage reducers in order to accumulate a representative dataset or to perform some per-values filtering constraint. Examples of this style of filtering include finding the n-largest or n-smallest values,

performing deduplication, or subselection. A very common filtering task in analytics is sampling: creating a smaller, representative dataset that is well distributed relative to the larger dataset (depending on the type of distribution you are expecting to achieve). Data-oriented subsamples are used in development, to validate machine-learning algorithms (e.g., cross-validation) or to produce other statistical computations (e.g., power).

Generically we might implement filtering as a function that takes a single record as input. If the evaluation returns true, the record is emitted; otherwise, it is dropped. In this section, we explore sortless n-largest/smallest, sampling techniques, as well as more advanced filtering using Bloom filters to improve performance.

Top n records

The top n records (and conversely the bottom n records) methodology is a cardinality comparison filter that requires both a mapper and reducer to work. The basic principle is to have each mapper yield its top n items, and then the reducer will similarly choose the top n items from the mappers. If n is relatively small (at least in comparison to the rest of the dataset) a single reducer should be able to handle this computation with ease because at most n records will come from each mapper:

```
import bisect

class TopNMapper(Mapper):

    def __init__(self, n, *args, **kwargs):
        self.n = n
        super(TopNMapper, self).__init__(*args, **kwargs)

    def map(self):

        items = []
        for value in self:
            # maintain sorted list of items
            bisect.insort(items, value)

        for item in items[-self.n:]:
            # emit the top n values from the mapper
            self.emit(None, item)


class TopNReducer(object):

    def __init__(self, n, *args, **kwargs):
        self.n = n
        super(TopNReducer, self).__init__(*args, **kwargs)

    def reduce(self):
        items = []
        for _, values in self:
            for value in values:
                bisect.insort(items, value)

        for item in items[-self.n:]:
            # emit the top n values from the mapper
            self.emit(None, item)
```

The primary benefit of this methodology is that a complete sort does not have to occur over the entire dataset. Instead, the mappers each sort their own subset of the data, and the reducer sees only n times the number of mappers worth of data.

Simple random sample

Simple random samples are subsets of a dataset where each record is equally likely to belong to the subset. In this case, the evaluation function does not care about the content or structure of the record, but instead utilizes some random number generator to evaluate whether to emit the record. The question is how to ensure that every element has an equal likelihood of being selected.

A first approach if we don't exactly need a specific sample of size n but rather some percentage of records is to simply use a random number generator to produce a number and compare it to the desired threshold size. Generally speaking, random number generators return a value between 0 and 1—so direct comparison to a percentage will yield the intended result! For example, if we want to sample 20% of our dataset, we might write a mapper as follows:

```
import random

class PercentSampleMapper(Mapper):

    def __init__(self, *args, **kwargs):
        self.percentage = kwargs.pop("percentage")
        super(PercentSampleMapper, self).__init__(*args, **kwargs)

    def map(self):
        for _, val in self:
            if random.random() < self.percentage:
                self.emit(None, val)

if __name__ == '__main__':
    mapper = PercentSampleMapper(sys.stdin, percentage=0.20)
    mapper.map()
```

Bloom filtering

A *bloom filter* is an efficient probabilistic data structure used to perform set membership testing. A bloom filter is really no different from any other evaluation function, except that a preliminary computation must be made to gather "hot values", which we would like to filter against. The benefit is that a bloom filter is compact and fast to test membership.

Bloom filters suffer, however, from false positives—in other words, saying something belongs to the set when it does not; however, they guarantee that any exclusion does not belong in the membership set—there are no false negatives). If you're willing to have some fuzziness, most bloom filters can be constructed with a threshold for the probability of a false negative, by increasing or decreasing the size of the bloom filter.

In order to construct a bloom filter, you will first have to build it. Bloom filters work by applying several hashes to input data, then by setting bits in a bit array according to the hash. Once the bit array is constructed, it can be used to test membership by applying hashes to the test data and seeing if the relevant bits are 1 or not. The bit array construction can either be parallelized by using rules to map distinct values to a reducer that constructs the bloom filter, or it can be a living, versioned data structure that is maintained by other processes.

In this example, we will use a third-party library, pybloomfiltermmap, which can be installed using pip. Let's consider an example in which we are including tweets based on whether they contain a hashtag or @ reply that is

in a whitelist of terms and usernames. In order to create the bloom filter, we load our data from disk, and save the bloom filter to mmap file as follows:

```
from pybloomfilter import BloomFilter

bloom = BloomFilter(1000000, 0.1, 'twitter.bloom')

for prefix, path in (('#', 'hashtags.txt'), ('@', 'handles.txt')):
    with open(path, 'r') as f:
        for word in f:
            bloom.add(prefix + word.strip())
```

After reading our hashtags and Twitter handles from files on, our bloom filter will be written to disk in a file called *twitter.bloom*.

To employ this in a Spark context:

```
ELEMS = re.compile(r'[#@][\w\d]+')

def tweet_filter(tweet, bloom=None):
    for elem in ELEMS.findall(tweet['text']):
        if elem in bloom.value:
            return True

# Load the bloom filter from disk and parallelize it
bloom = sc.broadcast(BloomFilter.open('twitter.bloom'))

# Load JSON tweets from disk and parse them
tweets = sc.textFile('tweets').map(json.loads)
tweets = tweets.filter(partial(tweet_filter, bloom=bloom))
```

Bloom filters are potentially the most complex data structure that you will use on a regular basis performing analytics in Hadoop.

9.3- Toward Last-Mile Analytics

Many machine learning techniques use generalized linear models (GLM) under the hood to estimate a response variable given some input data and an error distribution. The most commonly used GLM is a linear regression (others include logistic and Poisson regressions), which models the continuous relationship between a dependent variable *Y* and one or more independent variables, X. That relationship is encoded by a set of *coefficients* and an *error term* as follows:

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_n X_n + \epsilon$$

We can state that the computation of the β coefficients is the primary goal of fitting the model to existing data. This is generally done via an optimization algorithm that finds the set of coefficients that minimizes the amount of error given some dataset with observations for *X* and Y. Note that linear regression can be considered a *supervised* machine learning method, as the "correct" answers are known in advance.

Optimization algorithms like ordinary least squares or stochastic gradient descent are iterative; that is, they make

multiple passes over the data. In a big data context, reading a complete dataset multiple times for each optimization iteration can be prohibitively time consuming, particularly for on-demand analytics or development. Spark makes things a bit better with distributed machine learning algorithms and inmemory computing exposed in its MLlib. However, for extremely large datasets, or smaller time windows, even Spark can take too long; and if Spark doesn't have the model or distributed algorithm you'd like to implement, then the many gotchas of distributed programming could limit your analytical choices.

The general solution is the one: decompose your problem by transforming the input dataset into a smaller one, until it fits in memory. Once the dataset is reduced to an in-memory computation, it can be analyzed using standard techniques, then validated across the entire dataset. For a linear regression, we could take a simple random sample of the dataset, perform feature extraction on the sample, build our linear model, then validate the model by computing the mean square error of the entire dataset.

9.3.1-Fitting a Model

Consider a specific example where we have a dataset that originates from news articles or blog posts and a prediction task where we want to determine the number of comments in the next 24 hours. Given the raw HTML pages from a web crawl, the data flow may be as follows:

1. Parse HTML page for metadata and separate the main text and the comments.
2. Create an index of comments/commenters to blog post associated with a timestamp.
3. Use the index to create instances for our model, where an instance is a blog post and the comments in a 24-hour sliding window.
4. Join the instances with the primary text data (for both comments and blog test).
5. Extract the features of each instance (e.g., number of comments in the first 24 hours, the length of the blog post, bag of words features, day of week, etc.).
6. Sample the instance features.
7. Build a linear model in memory using Scikit-Learn or Statsmodels.
8. Compute the mean squared error or coefficient of determination across the entire dataset of instance features.

At this point, let's assume that through techniques we've already learned we've managed to arrive at a dataset that has all features extracted. Using the sampling technique, we can take a smaller dataset and save it to disk, and build a linear model with Scikit-Learn:

```
import pickle
import numpy as np
from sklearn import linear_model

# Load data from tab-delimited file on disk
data = np.loadtxt('sample.txt')

# The target is the first column (the key) and X is the value
y = data[:,0]
X = data[:,1:]

# Instantiate and fit our linear model
clf = linear_model.Ridge(alpha=1.0, fit_intercept=True)
clf.fit(X, y)
 # Write the model as a pickle to disk
 with open('clf.pickle', 'wb') as f:
     pickle.dump(clf, f)
```

This snippet of code uses the np.loadtxt function to load our sample data from disk, which in this case must be a tab-delimited file of instances where the first column is the target value and the remaining columns are the features. This type of output matches what might happen when key/value pairs are written to disk from Spark or MapReduce, although you will have to collect the data from the cluster into a single file, and ensure it is correctly formatted.

9.3.2-Validating Models

In order to use this model in the cluster to evaluate our performance, we have two choices. First, we could write the Scikit-Learn linear model properties, clf.coef_ (coefficients) and clf.intercept_ (error term) to disk and then load those parameters into our MapReduce or Spark job and compute the error ourselves. However, this requires us to implement a prediction function for every single model we may want to use. Instead, we will use the pickle module to dump the model to disk, then load it to every node in the cluster to make our prediction.

In order to validate our model, we must compute the mean square error (MSE) across the entire dataset. Error is defined as the difference between the actual and predicted values.

```
import pickle

class MSEMapper(Mapper):

    def __init__(self, model, *args, **kwargs):
        super(MSEMapper, self).__init__(*args, **kwargs)

        # Load our model from disk
        with open(model, 'rb') as f:
            self.clf = pickle.load(f)


    def map(self):
        for row in self:
            # Parse the floating-point values in the row
            row = map(float, row)
            y = row[0]
            X = row[1:]

            yhat = self.clf.predict(x)

            self.emit(_, (y-yhat) ** 2)
```

In Spark, we can use an accumulator to sum the square error, and broadcast the model across the cluster as follows:

```
def cost(row, clf=None):
    """
    Computes the square error given the row.
    """
    return (row[0] - clf.predict(row[1:])) ** 2
```

```
def main(sc):
    """
    Primary analysis mechanism for Spark application
    """

    # Load the model from the pickle file
    with open('clf.pickle', 'rb') as f:
```

```
    clf = sc.broadcast(model.load(f))


# Create an accumulator to sum the squared error
sum_square_error = sc.accumulator(0)

# Load and parse the blog data
blogs = sc.textFile("blogData").map(float)

# Map the cost function and accumulate the sum
error = blogs.map(partial(cost, clf=clf))
error.foreach(lambda cost: sum_square_error.add(cost))

# Print and compute the mean
print sum_square_error.value / error.count()
```

Unit V

Chapter II

Objectives

To Study and Understand the following concept

- **Data Mining and Warehousing**
- **Structured Data Queries with Hive**
- **HBase**
- **Data Ingestion**
- **Importing Relational data with Sqoop**
- **Injesting stream data with flume.**
- **Analytics with higher level APIs**
- **Pig**
- **Spark's higher level APIs.**

10-Data Mining and Warehousing

It's estimated that ETL consumes 70–80% of data warehousing costs, risks, and implementation time. This overhead makes it costly to perform even modest levels of data analysis prototyping or exploratory analysis. RDBMSs present another limitation in the face of the rapidly expanding diversity of data types that we need to store and analyze, which can be unstructured (emails, multimedia files) or semi-structured (clickstream data) in nature. The velocity and variety of this data often demands the ability to evolve the schema in a "just-in-time" manner, which is very tough to support in a traditional DW.

It's for these reasons that Hadoop has become the most disruptive technology in the data warehousing and data mining space. Hadoop's separation of storage from processing enables organizations to store their raw data in HDFS without necessitating ETLs to conform the data into a single unified data model. Moreover, with YARN's generalized processing layer, we're able to directly access and query the raw data from multiple perspectives and using different methods (SQL, non-SQL) as appropriate for the particular use case. Hadoop thus not only enables exploratory analysis and data mining prototyping, it opens the floodgates to new types of data and analysis.

10.1-Structured Data Queries with Hive

Apache Hive is a "data warehousing" framework built on top of Hadoop. Hive provides data analysts with a familiar SQL-based interface to Hadoop, which allows them to attach structured schemas to data in HDFS and access and analyze that data using SQL queries. Hive has made it possible for developers who are fluent in SQL

to leverage the scalability and resilience of Hadoop without requiring them to learn Java or the native MapReduce API.

Hive provides its own dialect of SQL called the Hive Query Language, or HQL. HQL supports many commonly used SQL statements, including data definition statements (DDLs), data manipulation statements (DMSs), and data retrieval queries. Hive also supports integration of custom user-defined functions, which can be written in Java or any language supported by Hadoop Streaming, that extend the built-in functionality of HQL.

Hive commands and HQL queries are compiled into an *execution plan* or a series of HDFS operations and/or MapReduce jobs, which are then executed on a Hadoop cluster. Thus, Hive has inherited certain limitations from HDFS and MapReduce that constrain it from providing key online transaction processing (OLTP) features that one might expect from a traditional database management system. In particular, because HDFS is a write-once, read-many (WORM) file system and does not provide in-place file updates, Hive is not very efficient for performing row-level inserts, updates, or deletes.

Additionally, Hive queries entail higher-latency due to the overhead required to generate and launch the compiled MapReduce jobs on the cluster; even small queries that would complete within a few seconds on a traditional RDBMS may take several minutes to finish in Hive.

On the plus side, Hive provides the high-scalability and high-throughput that you would expect from any Hadoop-based application, and as a result, is very well suited to batch-level workloads for online analytical processing (OLAP) of very large datasets at the terabyte and petabyte scale.

10.1.1- The Hive Command-Line Interface (CLI)

Hive's installation comes packaged with a handy command-line interface (CLI), which we will use to interact with Hive and run our HQL statements. To start the Hive CLI from the *$HIVE_HOME*:

```
~$ cd $HIVE_HOME
 /srv/hive$ bin/hive
```

This will initiate the CLI and bootstrap the logger and Hive history file, and finally display a Hive CLI prompt:

```
hive>
```

At any time, you can exit the Hive CLI using the following command:

```
hive> exit;
```

Hive can also run in non-interactive mode directly from the command line by passing the filename option, -f, followed by the path to the script to execute:

```
~$ hive -f ~/hadoop-fundamentats/hive/init.hqt
~$ hive -f ~/hadoop-fundamentats/hive/top_50_ptayers_by_homeruns.hqt >> ~/homeruns.tsv
```

Additionally, the quoted-query-string option, -e, allows you to run inline commands from the command line:

```
~$ hive -e 'SHOW DATABASES;'
```

You can view the full list of Hive options for the CLI by using the -H flag:

```
~$ hive -H

usage: hive
 -d,--define <key=value>          Variable substitution to apply to hive
                                      commands. e.g. -d A=B or --define A=B
```

…..

10.1.2-Hive Query Language (HQL)

Creating a database

Creating a database in Hive is very similar to creating a database in a SQL-based RDBMS, by using the `CREATE DATABASE` or `CREATE SCHEMA` statement:

```
hive> CREATE DATABASE log_data;
```

When Hive creates a new database, the schema definition data is stored in the *Hive metastore*. Hive will raise an error if the database already exists in the metastore; we can check for the existence of the database by using `IF NOT EXISTS`:

```
hive> CREATE DATABASE IF NOT EXISTS log_data;
```

We can then run `SHOW DATABASES` to verify that our database has been created. Hive will return all databases found in the metastore, along with the default Hive database:

```
hive> SHOW DATABASES;
OK
default
log_data
Time taken: 0.085 seconds, Fetched: 2 row(s)
```

Additionally, we can set our working database with the `USE` command:

```
hive> USE log_data;
```

Now that we've created a database in Hive, we can describe the layout of our data by creating table definitions within that database.

Creating tables

Hive provides a SQL-like `CREATE TABLE` statement, which in its simplest form takes a table name and column definitions:

```
CREATE TABLE apache_log (
    host STRING,
    identity STRING,
     user STRING,
     time STRING,
     request STRING,
     status STRING,
     size STRING,
     referer STRING,
     agent STRING
 );
```

However, because Hive data is stored in the file system, usually in HDFS or the local file system, the **CREATE** TABLE command also takes optional clauses to specify the row format with the **ROW FORMAT** clause that tells Hive how to read each row in the file and map to our columns. For example, we could indicate that the data is in a delimited file with fields delimited by the tab character:

```
hive> CREATE TABLE shakespeare (
    lineno STRING,
    linetext STRING
)
ROW FORMAT DELIMITED
    FIELDS TERMINATED BY '\t';
```

Fortunately, Hive provides a way for us to apply a regex to known record formats to *deserialize* or parse each row into its constituent fields. We'll use the Hive serializer-deserializer row format option, SERDE, and the contributed RegexSerDe library to specify a regex with which to deserialize and map the fields into columns for our table. We'll need to manually add the *hive-serde* JAR from the *lib* folder to the current hive session in order to use the RegexSerDe package:

```
hive> ADD JAR /srv/hive/tib/hive-serde-0.13.1.jar;
```

And now let's drop the apache_tog table that we created previously, and re-create it to use our custom serializer:

```
hive> DROP TABLE apache_log;

hive> CREATE TABLE apache_log (
    host STRING,
    identity STRING,
    user STRING,
    time STRING,
    request STRING,
    status STRING,
    size STRING,
    referer STRING,
    agent STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES ("input.regex" = "([^ ]*) ([^ ]*) ([^ ]*) (-|\\[[^\\]]
*\\])([^ \"]*|\"[^\"]*\") (-|[0-9]*) (-|[0-9]*)(?: ([^ \"]*|\".*\") ([^ \"]
```

```
*|\".*\"))?", "output.format.string" = "%1$s %2$s %3$s %4$s %5$s %6$s %7$s
%8$s %9$s")
STORED AS TEXTFILE;
```

Once we've created the table, we can use DESCRIBE to verify our table definition:

```
hive> DESCRIBE apache_log;
OK
host                    string                  from deserializer
identity                string                  from deserializer
user                    string                  from deserializer
time                    string                  from deserializer
request                 string                  from deserializer
status                  string                  from deserializer
size                    string                  from deserializer
referrer                string                  from deserializer
agent                   string                  from deserializer
Time taken: 0.553 seconds, Fetched: 9 row(s)
```

Type	Description	Example
TINYINT	8-bit signed integer, from -128 to 127	127
SMALLINT	16-bit signed integer, from -32,768 to 32,767	32,767
INT	32-bit signed integer	2,147,483,647
BIGINT	64-bit signed integer	9,223,372,036,854,775,807
FLOAT	32-bit single-precision float	1.99
DOUBLE	64-bit double-precision float	3.14159265359
BOOLEAN	True/false	true
STRING	2 GB max character string	hello world
TIMESTAMP	Nanosecond precision	1400561325

Table 10.1 Hive primitive data type

Type	Description	Example
ARRAY	Ordered collection of elements. The elements in the array must be of the same type.	`recipients ARRAY<email:STRING>`
MAP	Unordered collection of key/value pairs. Keys must be of primitive types and values can be of any type.	`files MAP<filename:STRING, size:INT>`
STRUCT	Collection of elements of any type.	`address STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>`

Table 10.2 Hive complex data type

Loading data

Hive does not perform any verification of the data for compliance with the table schema, nor does it perform any transformations when loading the data into a table. Data loading in Hive is done in batch-oriented fashion using a bulk LOAD DATA command or by inserting results from another query with the INSERT command. To start, let's copy our Apache log data file to HDFS and then load it into the table we created earlier:

```
~$ hadoop fs -mkdir statistics
~$ hadoop fs -mkdir statistics/log_data
~$ hadoop fs -copyFromLocal ~/hadoop-fundamentals/data/log_data/apache.log \
    statistics/log_data/
```

You can verify that the *apache.log* file was successfully uploaded to HDFS with the tail command:

```
~$ hadoop fs –tail statistics/log_data/apache.log
```

Once the file has been uploaded to HDFS, return to the Hive CLI and use the log_data database:

```
~$ $HIVE_HOME/bin/hive

hive> use log_data;
OK
Time taken: 0.221 seconds
```

We'll use the LOAD DATA command and specify the HDFS path to the logfile, writing the contents into the apache_log table:

```
hive> LOAD DATA INPATH 'statistics/log-data/apache.log'
OVERWRITE INTO TABLE apache_log;

Loading data to table log_data.apache_log
rmr: DEPRECATED: Please use 'rm -r' instead.
Deleted hdfs://localhost:9000/user/hive/warehouse/log_data.db/apache_log
Table log_data.apache_log stats: [numFiles=1, numRows=0, totalSize=52276758,
rawDataSize=0]
OK
Time taken: 0.902 seconds
```

INPATH takes an argument to a path on the default file system (in this case, HDFS). We can also specify a path on the local file system by using LOCAL INPATH instead.

10.1.3-Data Analysis with Hive

SQL GROUP BY query can be used to computes the number of hits per calendar month:

```
hive> SELECT
        month,
        count(1) AS count
      FROM (SELECT split(time, '/')[1] AS month FROM apache_log) l
      GROUP BY month
      ORDER BY count DESC;
OK
Mar 99717
Sep 89083
Feb 72088
Aug 66058
Apr 64984
May 63753
Jul 54920
Jun 53682
Oct 45892
Jan 43635
Nov 41235
Dec 29789
NULL    1903
Time taken: 84.77 seconds, Fetched: 13 row(s)
```

we can easily perform other ad hoc queries on any of the other fields for example:

```
hive> SELECT host, count(1) AS count FROM apache_log GROUP BY host ORDER BY count;
```

In addition to count, Hive also supports other aggregate functions to compute the sum, average, min, max as well as statistical aggregations for variance, standard deviation, and covariance of numeric columns. When using these built-in aggregate functions, you can improve the performance of the aggregation query by setting the following property to true:

```
hive> SET hive.map.aggr = true;
```

We can create new tables to store the results returned by these queries for later record-keeping and analysis:

```
hive> CREATE TABLE
        remote_hits_by_month
        AS
        SELECT
            month,
            count(1) AS count
        FROM (
            SELECT split(time, '/')[1] AS month
            FROM apache_log
            WHERE host == 'remote'
            ) l
        GROUP BY month
        ORDER BY count DESC;
```

Aggregations and joins

Consider the US flight data *ontime_flights.tsv*. Each row of the on-time flight data in *ontime_flights.tsv* includes an integer value that represents the code for AIRLINE_ID (such as 19805) and a string value that represents the

code for CARRIER (such as "AA"). AIRLINE_ID codes can be joined with the corresponding code in the *airlines.tsv* file in which each row contains the code and corresponding description:

19805 American Airlines Inc.: AA

Accordingly, CARRIER codes can be joined with the corresponding code in *carriers.tsv*, which contains the code and corresponding airline name and effective dates:

AA American Airlines Inc. (1960 -)

Assuming that we've uploaded our data files to HDFS or local file system and created required tables with data. To get a list of airlines and their respective average departure delays, we can simply perform a SQL JOIN on flights and airlines on the airline code and then use the aggregate function AVG() to compute the average depart_delay grouped by the airline description:

```
hive> SELECT
        a.description,
        AVG(f.depart_delay)
      FROM airlines a
      JOIN flights f ON a.code = f.airline_code
      GROUP BY a.description;
```

10.2-HBase

We know that while Hive provides a familiar data manipulation paradigm within Hadoop, it doesn't change the storage and processing paradigm, which still utilizes HDFS and MapReduce in a batch-oriented fashion. HBase is part of the Hadoop ecosystem which offers random real-time read/write access to data in the Hadoop File System.

10.2.1-NoSQL and Column-Oriented Databases

NoSQL is a broad term that generally refers to non-relational databases and encompasses a wide collection of data storage models, including graph databases, document databases, key/value data stores and column family databases. HBase is classified as a column-family or column-oriented database, modeled on Google's BigTable architecture. This architecture allows HBase to provide:

- Random (row-level) read/write access
- Strong consistency
- "Schema-less" or flexible data modeling

The schema-less trait is a result of how HBase approaches data modeling, which is very different from how relational databases approach data modeling. HBase organizes data into *tables* that contain *rows*. Within a table, rows are identified by their unique *row key*, which do not have a data type and are instead stored and treated as a *byte array*. Row keys are similar to the concept of primary keys in relational databases, in that they are automatically indexed; in HBase, table rows are sorted by their row key and because row keys are byte arrays, almost anything can serve as a row key from strings to binary representations of longs or even serialized data structures.

HBase stores its data as key/value pairs, where all table lookups are performed via the table's row key, or unique identifier to the stored record data. Data within a row is grouped into *column families*, which consist of related columns. Visually, you can picture an HBase table that holds census data for a given population where each row represents a person and is accessed via a unique ID rowkey, with column families for personal data which contains columns for name and address, and demographic info which contains columns for birthdate and gender. This

example is shown in Figure 10.3.

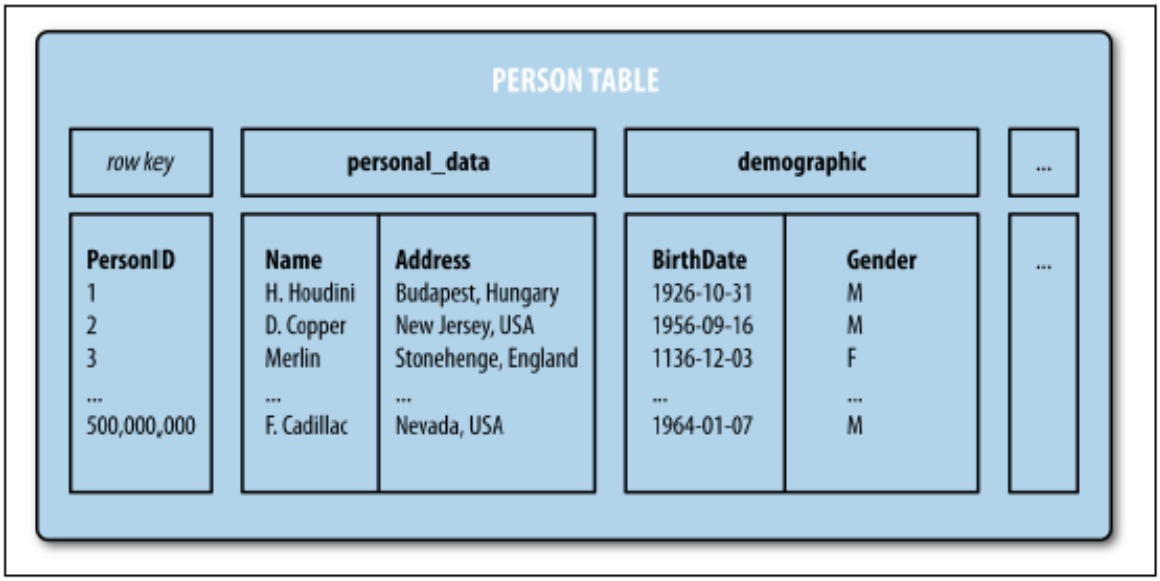
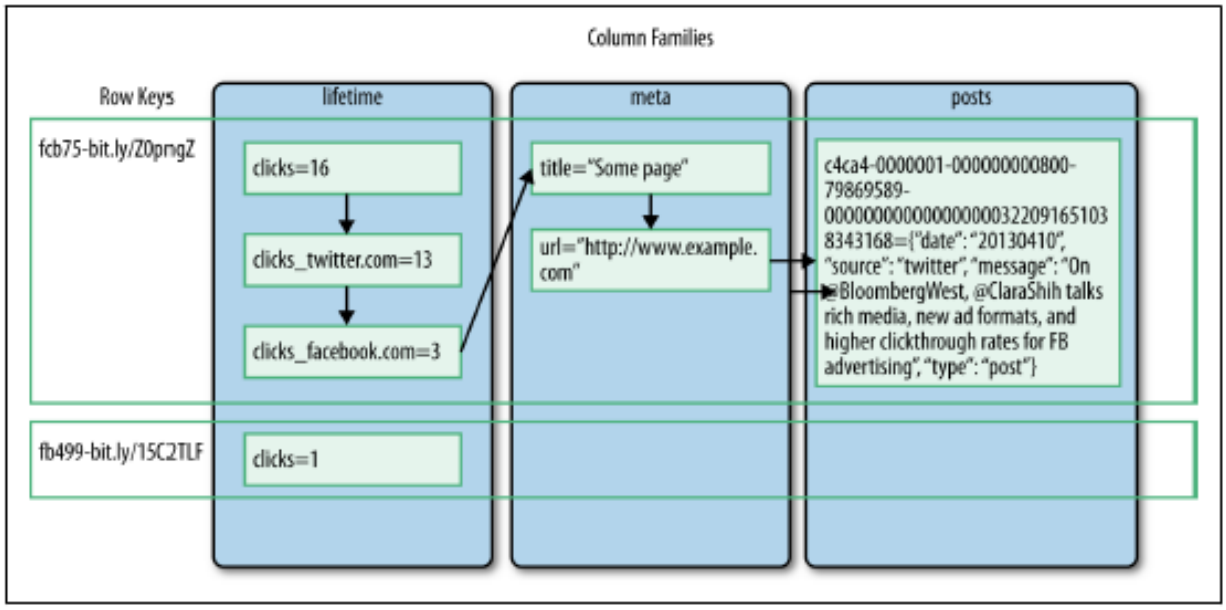


Figure 10.3 Census data as an HBase schema

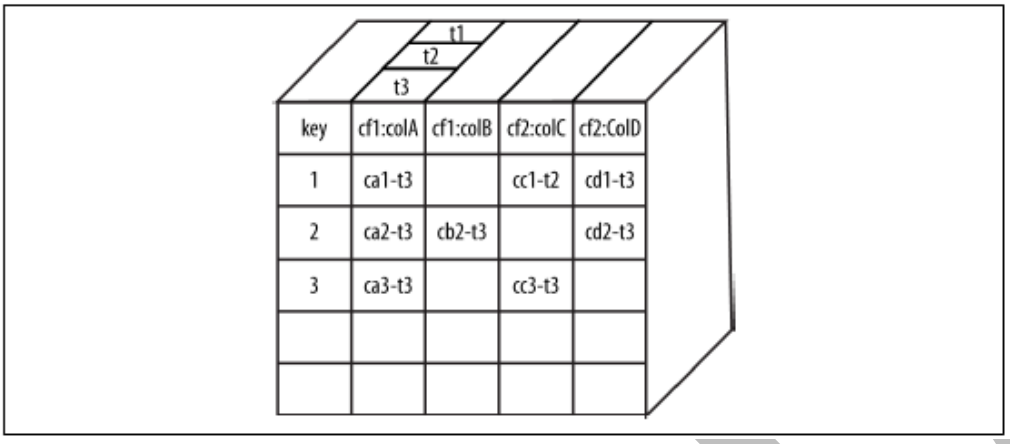
Storing data in columns rather than rows has particular benefits for data warehouses and analytical databases where aggregates are computed over large sets of data with potentially sparse values, where not all columns values are present. However, the actual columns that make up a row can be determined and created on an as-needed basis. In fact, each row can have a different set of columns. Figure 6-2 shows an example HBase table with two rows where first row key utilizes three column families and the second row key utilizes just one column.



10.4 Social media events with sparse columns

Another interesting feature of HBase and BigTable-based column-oriented databases is that the table cells, or the intersection of row and column coordinates, are versioned by timestamp, stored as a long integer representing milliseconds since January 1, 1970 UTC. HBase is thus also described as being a multidimensional map where time provides the third dimension, as shown in Figure 6-3. The time dimension is indexed in decreasing order, so that when reading from an HBase store, the most recent values are found first. The contents of a cell can be

referenced by a *{rowkey, column, timestamp}* tuple, or we can scan for a range of cell values by time range.



10.5 HBase timestamp versioning

10.2.2-Real-Time Analytics with HBase

HBase schemas can be created or updated with the HBase Shell or with the Java API, using the HBaseAdmin interface class. Additionally, HBase supports a number of other clients that can be used to support non-Java programming languages, including a REST API interface, Thrift, and Avro. These clients act as proxies that wrap the native Java API.

Generating a schema

When designing schemas in HBase, it's important to think in terms of the column- family structure of the data model and how it affects data access patterns. Furthermore, because HBase doesn't support joins and provides only a single indexed rowkey, we must be careful to ensure that the schema can fully support all use cases. Often this involves de-normalization and data duplication with nested entities.

But HBase allows dynamic column definition at runtime, we have quite a bit of flexibility even after table creation to modify and scale our schema.

Namespaces, tables, and column families

First, we need to declare the table name, and at least one column-family name at the time of table definition. We can also declare our own optional namespace to serve as a logical grouping of tables, analogous to a database in relational database systems. If no namespace is declared, HBase will use the default namespace.

```
hbase> create 'linkshare', 'link'
0 row(s) in 1.5740 seconds
```

We just created a single table called `linkshare` in the default namespace with one column-family, named `link`. To alter the table after creation, such as changing or adding column families, we need to first disable the table so that clients will not be able to access the table during the `alter` operation:

```
hbase> disable 'linkshare'
0 row(s) in 1.1340 seconds

hbase> alter 'linkshare', 'statistics'
Updating all regions with the new schema...
1/1 regions updated.
Done.
0 row(s) in 1.1630 seconds
```

We can then re-enable the table using the `enable` command:

```
hbase> enable 'linkshare'
0 row(s) in 1.1930 seconds
```

And then use the `describe` command to verify that the table contains the two expected column families with the default configurations:

```
hbase> describe 'linkshare'

Table linkshare is ENABLED
COLUMN FAMILIES DESCRIPTION
{NAME => 'link', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW',
REPLICATION_SCOPE => '0', COMPRESSION => 'NONE', VERSIONS => '1',
TTL => 'FOREVER', MIN_VERSIONS => '0', KEEP_DELETED_CELLS => 'FALSE',
BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE => 'true'}
{NAME => 'statistics', DATA_BLOCK_ENCODING => 'NONE',
BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0', VERSIONS => '1',
COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => 'FOREVER',
KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536',
IN_MEMORY => 'false', BLOCKCACHE => 'true'}
2 row(s) in 0.1290 seconds
```

Row keys

Before we insert row data, we need to determine how to design our row key. By default, HBase stores rows in sorted order by row key, so that similar keys are stored to the same RegionServer. While this enables faster range scans, it could also lead to uneven load on particular servers during read/write operations. For the current example, let's assume that we will use the unique reversed link URL for the row key.

Inserting data with put

We can insert, or put, a value in a cell at the specified table/row/column and optionally timestamp coordinates. To put a cell value into table linkshare at row with row key org.hbase.www under column-family link and column title marked with the current timestamp, do:

```
hbase> put 'linkshare', 'org.hbase.www', 'link:title', 'Apache HBase'
hbase> put 'linkshare', 'org.hadoop.www', 'link:title', 'Apache Hadoop'
hbase> put 'linkshare', 'com.oreilly.www', 'link:title', 'O\'Reilly.com'
```

The put operation works great for inserting a value for a single cell, but for incrementing frequency counters, HBase provides a special mechanism to treat columns as counters. Otherwise, under heavy load, we could face significant contention for these rows as we would need to lock the row, read the value, increment it, write it back, and finally unlock the row for other writers to be able to access the cell.[9]

To increment a counter, we use the command incr instead of put:

```
hbase> incr 'linkshare', 'org.hbase.www', 'statistics:share', 1
(COUNTER VALUE is now 1)

hbase> incr 'linkshare', 'org.hbase.www', 'statistics:like', 1
(COUNTER VALUE is now 1)
```

The last option passed is the increment value, which in this case is 1. Incrementing a counter will return the updated counter value, but you can also access a counter's current value any time using the get_counter command, specifying the table name, row key, and column:

```
hbase> incr 'linkshare', 'org.hbase.www', 'statistics:share', 1
(COUNTER VALUE is now 2)

hbase> get_counter 'linkshare', 'org.hbase.www', 'statistics:share', 'dummy'
COUNTER VALUE = 2
```

The get_counter method is used to decode the byte-array value of the counter and return the integer value. Unfortunately, the latest HBase build includes a bug in the shell command for getting the counter value, which expects a fourth argument that is never used. As a result, we'll need to pass in a fourth dummy value:

```
hbase> get_counter 'linkshare', 'org.hbase.www', 'statistics:share', 'dummy'
COUNTER VALUE = 2
```

HBase provides two general methods to retrieve data from a table: the `get` command performs lookups by row key to retrieve attributes for a specific row, and the `scan` command, which takes a set of filter specifications and iterates over multiple rows based on the indicated specifications.

Get row or cell values

In its simplest form, the `get` command accepts the table name followed by the row key, and returns the most recent version timestamp and cell value for all columns in the row:

```
hbase> get 'linkshare', 'org.hbase.www'

COLUMN                          CELL
 link:title                     timestamp=1422145743298, value=Apache HBase
 statistics:like                timestamp=1422153344211,
     value=\x00\x00\x00\x00\x00\x00\x00\x1F
 statistics:share               timestamp=1422153337498,
     value=\x00\x00\x00\x00\x00\x00\x00\x02
3 row(s) in 0.0310 seconds
```

Note that the `statistics:share` column returns the value in its byte array representation, printing the separate bytes as hexadecimal values. To display the integer representation of the counter value, use the `get_counter` command mentioned in the previous section.

The `get` command also accepts an optional dictionary of parameters to specify the column(s), timestamp, timerange, and version of the cell values we want to retrieve. For example, we can specify the column(s) of interest:

```
hbase> get 'linkshare', 'org.hbase.www', {COLUMN => 'link:title'}
hbase> get 'linkshare', 'org.hbase.www', {COLUMN => ['link:title',
    'statistics:share']}
```

There is also a shortcut to specify column parameters in a `get` by just appending the comma-delimited column names after the row key:

```
hbase> get 'linkshare', 'org.hbase.www', 'link:title'
hbase> get 'linkshare', 'org.hbase.www', 'link:title', 'statistics:share'
hbase> get 'linkshare', 'org.hbase.www', ['link:title', 'statistics:share']
```

To specify a time range of values we are interested in, we pass in a `TIMERANGE` parameter with start and end timestamps in milliseconds:

```
hbase> get 'linkshare', 'org.hbase.www', {TIMERANGE => [1399887705673,
    1400133976734]}
```

Scan rows

Using scan is similar to using the get command; it also accepts COLUMN, TIMESTAMP, TIMERANGE, and FILTER parameters. However, instead of specifying a single row key, you can specify an optional STARTROW and/or STOPROW parameter, which can be used to limit the scan to a specific range of rows. If neither STARTROW nor STOPROW are provided, the scan operation will scan through the entire table.

You can, in fact, call scan with the table name to display all the contents of a table:

```
hbase> scan 'linkshare'

ROW                 COLUMN+CELL
com.oreilly.www     column=link:title, timestamp=1422153270279,
value=O'Reilly.com
org.hadoop.www      column=link:title, timestamp=1422153262507,
value=Apache Hadoop
org.hbase.www       column=link:title, timestamp=1422145743298,
value=Apache HBase
org.hbase.www       column=statistics:like, timestamp=1422153344211,
    value=\x00\x00\x00\x00\x00\x00\x00\x1F
org.hbase.www       column=statistics:share, timestamp=1422153337498,
    value=\x00\x00\x00\x00\x00\x00\x00\x02
3 row(s) in 0.0290 seconds
```

Keep in mind that the rows in HBase are stored in lexicographical order.[10] For example, numbers going from 1 to 100 will be ordered like this:

```
1,10,100,11,12,13,14,15,16,17,18,19,2,20,21,...,9,91,92,93,94,95,96,97,98,99
```

Let's retrieve the link:title column but limit our scan to the rows starting with row key org.hbase.www:

```
hbase> scan 'linkshare', {COLUMNS => ['link:title'], STARTROW => 'org.hbase.www'}

ROW                 COLUMN+CELL
 org.hbase.www      column=link:title, timestamp=1453184861236,
 value=Apache HBase
1 row(s) in 0.0250 seconds
```

Filters

HBase provides a number of filter classes that can be applied to further filter the row data returned from a get or scan operation. These filters can provide a much more efficient means of limiting the row data returned by HBase and offloading the row-filtering operations from the client to the server. Some of HBase's available filters include:

- *RowFilter:* Used for data filtering based on row key values
- *ColumnRangeFilter:* Allows efficient intra-row scanning, can be used to get a *slice* of the columns of a very wide row.
- SingleColumnValueFilter: Used to filter cells based on column value
- *RegexStringComparator:* Used to test if a given regular expression matches a cell value in the column.

The HBase Java API provides a Filter interface and abstract FilterBase class plus a number of specialized Filter subclasses. Custom filters can also be created by subclassing the FilterBase abstract class and implementing the key abstract methods.

To begin, we need to import the necessary classes, including the org.apache.hadoop.hbase.util.Bytes to convert our column family, column, and values into bytes, and the filter and comparator classes:

```
hbase> import org.apache.hadoop.hbase.util.Bytes
hbase> import org.apache.hadoop.hbase.filter.SingleColumnValueFilter
hbase> import org.apache.hadoop.hbase.filter.BinaryComparator
hbase> import org.apache.hadoop.hbase.filter.CompareFilter
```

Next, we'll create a filter that limits the results to rows where the `statistics:like` counter column value is greater than or equal to 10:

```
hbase> likeFilter = SingleColumnValueFilter.new(Bytes.toBytes('statistics'),
    Bytes.toBytes('like'),
    CompareFilter::CompareOp.valueOf('GREATER_OR_EQUAL'),
    BinaryComparator.new(Bytes.toBytes(10)))
```

And because we don't have a value for this column for every row, we need to set a flag that tells this filter to skip any rows without a value in this column:

```
hbase> likeFilter.setFilterIfMissing(true)
```

At this point, we can run our `scan` operation with the filter we configured:

```
hbase> scan 'linkshare', { FILTER => likeFilter }

ROW                 COLUMN+CELL
org.hbase.www       column=link:title, timestamp=1422145743298,
    value=Apache HBase
org.hbase.www       column=statistics:like, timestamp=1422153344211,
    value=\x00\x00\x00\x00\x00\x00\x00\x1F
org.hbase.www       column=statistics:share, timestamp=1422153337498,
   value=\x00\x00\x00\x00\x00\x00\x00\x02
1 row(s) in 0.0470 seconds
```

This should return all rows that contain a column value for `statistics:like` that is greater than or equal to 10; this should include row key `com.oreilly.www` in this example.

10.3.-Data Ingestion

One of Hadoop's greatest strengths is that it's inherently schemaless and can work with any type or format of data regardless of structure (or lack of structure) from any source, as long as you implement Hadoop's Writable or DBWritable interfaces and write your MapReduce code to parse the data correctly. However, in cases where the input data is already structured because it resides in a relational database, it would be convenient to leverage this known schema to import the data into Hadoop in a more efficient manner than uploading CSVs to HDFS and parsing them manually.

Sqoop (SQL-to-Hadoop) is designed to transfer data between relational database management systems (RDBMS) and Hadoop. It automates most of the data transformation process, relying on the RDBMS to provide the schema description for the data to be imported.

While Sqoop works very well for bulk-loading data that already resides in a relational database into Hadoop, many new applications and systems involve fast-moving data streams like application logs, GPS tracking, social media updates, and sensor-data that we'd like to load directly into HDFS to process in Hadoop. In order to handle and process the high-throughput of event-based data produced by these systems, we need the ability to support continuous ingestion of data from multiple sources into Hadoop.

Apache Flume was designed to efficiently collect, aggregate, and move large amounts of log data from many different sources into a centralized data store. While Flume is most often used to direct streaming log data into Hadoop, usually HDFS or HBase, Flume data sources are actually quite flexible and can be customized to transport many types of event data, including network traffic data, social media-generated data, and sensor data into any Flume-compatible consumer.

10.3.1-Importing Relational Data with Sqoop

Sqoop (SQL-to-Hadoop) is a relational database import and export tool created by Cloudera, and is now an Apache top-level project. Sqoop is designed to transfer data between a relational database like MySQL or Oracle, into a Hadoop data store, including HDFS, Hive, and HBase. It automates most of the data transfer process by reading the schema information directly from the RDBMS. Sqoop then uses MapReduce to import and export the data to and from Hadoop.

Sqoop gives us the flexibility to maintain our data in its production state while copying it into Hadoop to make it available for further analysis without modifying the production database.

10.3.2.-Importing from MySQL to HDFS

When importing data from relational databases like MySQL, Sqoop reads the source database to gather the necessary metadata for the data being imported. Sqoop then submits a map-only Hadoop job to transfer the actual table data based on the metadata that was captured in the previous step. This job produces a set of serialized files, which may be delimited text files, binary format (e.g., Avro), or Sequence Files containing a copy of the imported table or datasets. By default, the files are saved as comma-separated files to a directory on HDFS with a name that corresponds to the source table name.

Assuming that you have MySQL with a database called energydata and a table called average_price_by_state:

Before we proceed to run the sqoop import command, verify that HDFS and YARN are started with the jps command:

```
~$ sudo su hadoop
hadoop@ubuntu:~$ jps

4051 NodeManager
31134 Jps
3523 DataNode
3709 SecondaryNameNode
3375 NameNode
3921 ResourceManager
```

At this point, we can import the data in table `average_price_by_state` into HDFS by using the `import` command. We can specify the source database's connection string, username, and tablename with the `--connect` option, `--username` option, and `--table` option, respectively. We'll set the optional `-m` flag to 1 to indicate that this job should use a single map task:

```
/srv/sqoop$ sqoop import --connect jdbc:mysql://localhost:3306/energydata
    --username root --table average_price_by_state -m 1

15/01/20 22:47:35 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6
15/01/20 22:47:35 INFO manager.MySQLManager: Preparing to use a MySQL
    streaming resultset.
15/01/20 22:47:35 INFO tool.CodeGenTool: Beginning code generation
15/01/20 22:47:36 INFO manager.SqlManager: Executing SQL statement:
    SELECT t.* FROM `average_price_by_state` AS t LIMIT 1

(output truncated)

15/01/25 22:47:53 INFO mapreduce.ImportJobBase: Transferred 200.4287 KB in
    15.3718 seconds (13.0387 KB/sec)
15/01/25 22:47:53 INFO mapreduce.ImportJobBase: Retrieved 3272 records.
```

10.2.3-Importing from MySQL to Hive

Sqoop provides a couple ways to do this, either exporting to HDFS first and then loading the data into Hive using the LOAD DATA HQL command in the Hive shell, or by using Sqoop to directly create the tables and load the relational database data into the corresponding tables in Hive.

Sqoop can generate a Hive table and load data based on the defined schema and table contents from a source database, using the import command. However, because Sqoop still actually utilizes MapReduce to implement the data load operation, we must first delete any preexisting data directory with the same output name before running the import tool:

```
/srv/sqoop$ hadoop fs -rm -r /user/hadoop/average_price_by_state
```

We can then run Sqoop's `import` command, passing it the JDBC connection string to the database, the table name, field delimiter, line terminator, and null string value:

```
/srv/sqoop$ sqoop import --connect jdbc:mysql://localhost:3306/energydata
    --username root --table average_price_by_state
    --hive-import --fields-terminated-by ','
    --lines-terminated-by '\n' --null-string 'null' -m 1

(output truncated)

15/01/20 00:14:37 INFO hive.HiveImport: Table default.average_price_by_state stats:
    [numFiles=2, numRows=0, totalSize=205239, rawDataSize=0]
15/01/20 00:14:37 INFO hive.HiveImport: OK
15/01/20 00:14:37 INFO hive.HiveImport: Time taken: 0.435 seconds
15/01/20 00:14:37 INFO hive.HiveImport: Hive import complete.
15/01/20 00:14:37 INFO hive.HiveImport: Export directory is empty, removing it.
```

In local mode, Hive will create a *metastore_db* directory within the file system location from which it was run; After above query *metastore_db* will be created under the *SQOOP_HOME* (*/srv/sqoop*). Open the Hive shell and verify that the table average_price_by_state was created:

```
/srv/sqoop$ hive

hive> DESC average_price_by_state;

OK
year                    int
state                   string
sector                  string
residential             double
commercial              double
industrial              double
transportation          double
other                   double
total                   double
Time taken: 0.858 seconds, Fetched: 9 row(s)
```

10.3.4-Importing from MySQL to HBase

HBase is designed to handle large volumes of data for a large number of concurrent clients that need real-time access to row-level data. Sqoop's import tool allows us to import data from a relational database to HBase. As with Hive, there are two approaches to importing this data. We can import to HDFS first and then use the HBase CLI or API to load the data into an HBase table, or we can use the **--hbase-table** option to instruct Sqoop to directly import to a table in HBase.

In this example, the data that we want to offload to HBase is a table of weblog stats where each record contains a primary key composed of the pipe-delimited IP address and year, and a column for each month that contains the number of hits for that IP and year. You can find the CSV named *weblogs.csv* in the GitHub repo's */data* directory. Download this CSV and load it into a MySQL table. Consider we have table *weblogs* in **logadata** database in MQSQL.

Again, we need to verify that Hadoop is running, as well as HBase daemons:

```
~$ cd $HBASE_HOME
/srv/hbase$ bin/start-hbase.sh
```

We can then run Sqoop's `import` command, passing it the JDBC connection string to the database, the table name, HBase table name, column family name, and row key name:

```
sqoop import --connect jdbc:mysql://localhost:3306/logdata
    --table weblogs --hbase-table weblogs --column-family traffic
    --hbase-row-key ipyear --hbase-create-table -m 1

(output truncated)

15/01/20 00:33:01 INFO mapreduce.ImportJobBase: Transferred 0 bytes in
19.0716 seconds (0 bytes/sec)
15/01/20 00:33:01 INFO mapreduce.ImportJobBase: Retrieved 27300 records.
```

10.4-Ingesting Streaming Data with Flume

Flume is designed to collect and ingest high volumes of data from multiple data streams into Hadoop. A very common use case for Flume is the collection of log data, such as collecting web server log data emitted from multiple application servers, and aggregating it in HDFS for later search or analysis. However, Flume isn't restricted to simply consuming and ingesting log data sources, but can also be customized to transport massive quantities of event data from any custom event source. In both cases, Flume enables us to incrementally and continuously ingest streaming data *as it is written* into Hadoop, rather than writing custom client applications to batch-load the data into HDFS, HBase, or other Hadoop data sink. Flume provides a unified yet flexible method of pushing data from many fast-moving, disparate data streams into Hadoop.

Flume's flexibility is derived from its inherently extensible data flow architecture. In addition to flexibility, Flume is designed to maintain both fault-tolerance and scalability through its distributed architecture. Flume provides multiple failover and recovery mechanisms, although the default "end-to-end" reliability mode that guarantees that accepted events will eventually be delivered is generally the recommended setting.

10.4.1-Flume Data Flows

Flume expresses the data ingestion pathway from origin to destination as a *data flow*. In a data flow, a unit of data or *event* (e.g., a single log statement) travels from a source to the next destination via a sequence of hops. This concept of data flow is expressed even in the simplest entity in a Flume flow, a Flume *agent*. A Flume agent is a single unit within a Flume data flow (actually, a JVM process), through which events propagate once initiated at an external source. Agents consist of three configurable components: the *source*, *channel*, and *sink*, as shown in Figure 10-6.

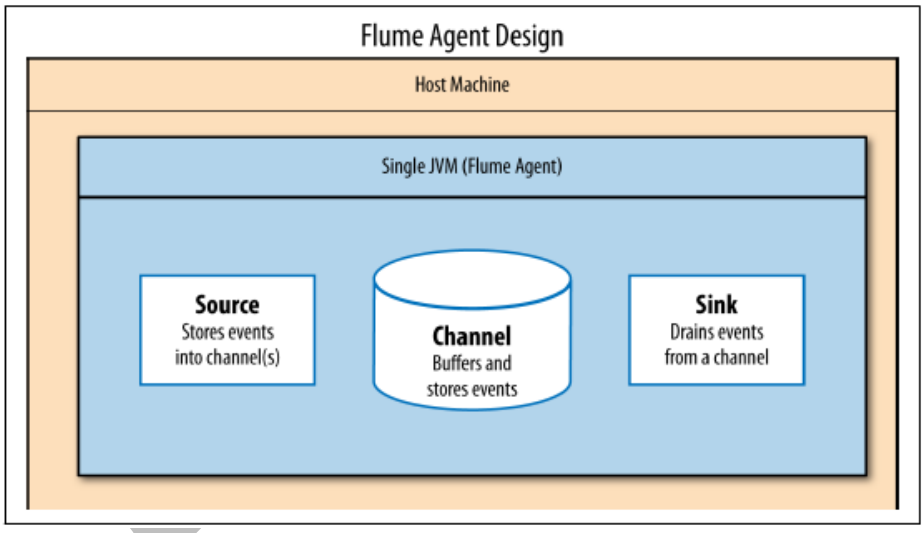


Figure 10.6. flume agent design

A Flume source is configured to listen for and consume events from one or more external data sources (not to be confused with a Flume source), which are configured by setting a name, type, and additional optional parameters for each data source. For example, we could configure up a Flume agent's source to accept events from an Apache access log by running a tail -f /etc/httpd/logs/access_log command. This type of source is called an exec source because it requires Flume to execute a Unix command to retrieve events.

When the agent consumes an event, the Flume source writes it to a channel, which acts as a storage queue that stores and buffers events until they are ready to be read. Events are written to channels transactionally, meaning that a channel keeps all events queued until they have been consumed and the corresponding transactions are

explicitly closed. This enables Flume to maintain durability of data events even if an agent goes down.

Flume sinks eventually read and remove events from the channel and forward them to their next hop or final destination. Sinks can thus be configured to write its output as a streaming source for another Flume agent, or to a data store like HDFS or HBase.

Using this source-channel-sink paradigm, we can easily construct a simple singleagent Flume data flow to consume events from an Apache access log and write the log events to HDFS, as shown in Figure 10-7.

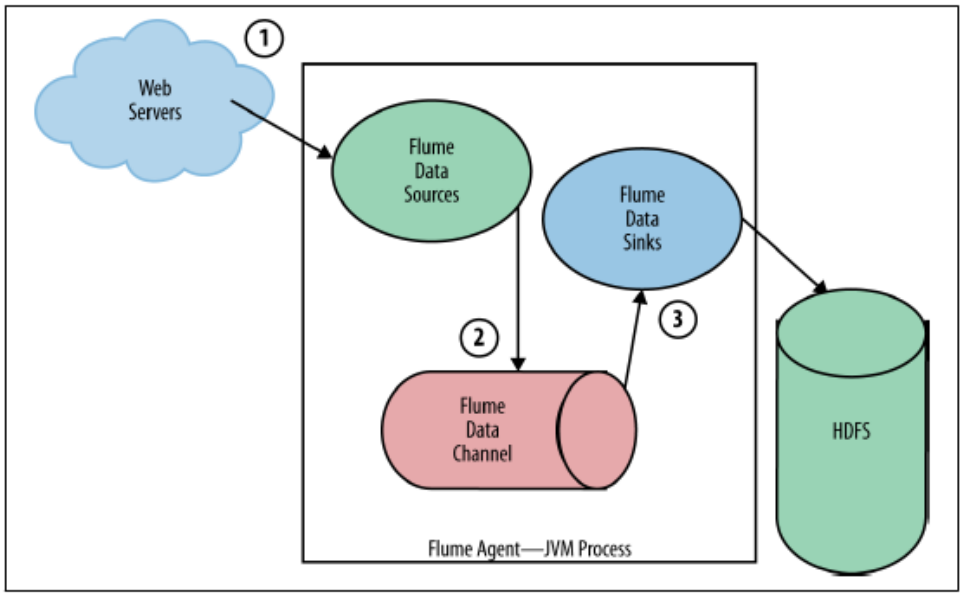


Figure 10.7.Simple Flume data flow

But because Flume agents are so adaptable and can even be configured to have multiple sources, channels, and sinks, we can actually construct multi-agent data flows by chaining several Flume agents together, as shown in Figure 10.8.

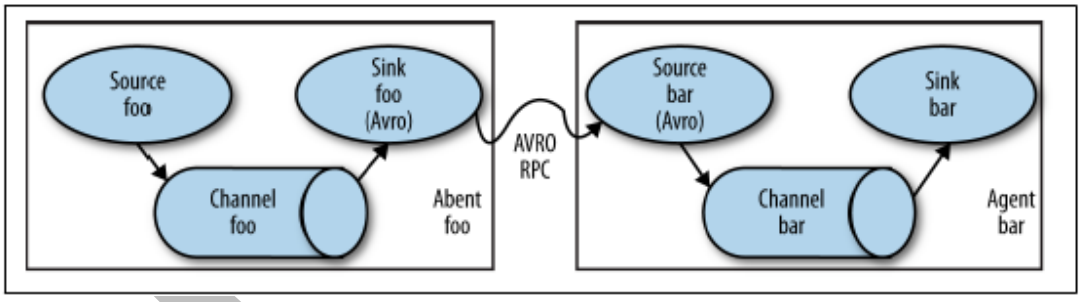


Figure 10.8. Multi-agent Flume data flow

There's almost no boundaries around how Flume agents can be organized into these complex data flows, although certain patterns and topologies of Flume data flows have emerged to handle common scenarios when dealing with a streaming data- processing architecture. For instance, a common scenario in log collection is when a large number of log producing clients are writing events to several Flume agents, which we call "first-tier" agents, as they are consuming data at the layer of the external data source(s). If we want to write these events to HDFS, we can set up each of the first-tier agents' sinks to write to HDFS, but this could present several problems as the first-tier scales out. Because several disparate agents are writing to HDFS independently, this data flow wouldn't be able to handle periodic bursts of data writes to the storage system and could thus introduce spikes in load and latency.

10.5-Analytics with Higher-Level APIs

10.5.1-Pig

Pig, like Hive, is an abstraction of MapReduce, allowing users to express their data processing and analysis operations in a higher-level language that then compiles into a MapReduce job. Pig is now a top-level Apache Project that includes two main platform components:

- Pig Latin, a procedural scripting language used to express data flows.
- The Pig execution environment to run Pig Latin programs, which can be run in local or MapReduce mode and includes the Grunt command-line interface.

Pig Latin is procedural in nature and designed to enable programmers to easily implement a series of data operations and transformations that are applied to datasets to form a data pipeline. While Hive is great for use cases that translate well to SQL-based scripts, SQL can become unwieldy when multiple complex data transformations are required. Pig Latin is ideal for implementing these types of multistage data flows, particularly in cases where we need to aggregate data from multiple sources and perform subsequent transformations at each stage of the data processing flow.

Pig Latin scripts start with data, apply transformations to the data until the script describes the desired results, and execute the entire data processing flow as an optimized MapReduce job. Additionally, Pig supports the ability to integrate custom code with user-defined functions (UDFs) that can be written in Java, Python, or JavaScript, among other supported languages. Pig thus enables us to perform near arbitrary transformations and ad hoc analysis on our big data using comparatively simple constructs.

It is important to remember the earlier point that Pig, like Hive, ultimately compiles into MapReduce and cannot transcend the limitations of Hadoop's batch-processing approach. However, Pig does provide us with powerful tools to easily and succinctly write complex data processing flows, with the fine-grained controls that we need to build real business applications on Hadoop.

10.5.2-Pig Latin

The following script loads Twitter tweets with the hashtag #unitedairlines over the course of a single week. The data file, *united_airlines_tweets.tsv*, provides the tweet ID, permalink, date posted, tweet text, and Twitter username. The script loads a dictionary, *dictionary.tsv*, of known "positive" and "negative" words along with sentiment scores (1 and -1, respectively) associated to each word. The script then performs a series of Pig transformations to generate a sentiment score and classification, either POSITIVE or NEGATIVE, for each computed tweet:

```
grunt> tweets = LOAD 'united_airlines_tweets.tsv' USING PigStorage('\t')
    AS (id_str:chararray, tweet_url:chararray, created_at:chararray,
    text:chararray, lang:chararray, retweet_count:int, favorite_count:int,
    screen_name:chararray);
grunt> dictionary = LOAD 'dictionary.tsv' USING PigStorage('\t')
    AS (word:chararray, score:int);
grunt> english_tweets = FILTER tweets BY lang == 'en';
grunt> tokenized = FOREACH english_tweets GENERATE id_str,
    FLATTEN( TOKENIZE(text) ) AS word;
grunt> clean_tokens = FOREACH tokenized GENERATE id_str,
    LOWER(REGEX_EXTRACT(word, '[#@]{0,1}(.*)', 1)) AS word;
grunt> token_sentiment = JOIN clean_tokens BY word, dictionary BY word;
grunt> sentiment_group = GROUP token_sentiment BY id_str;
grunt> sentiment_score = FOREACH sentiment_group
    GENERATE group AS id, SUM(token_sentiment.score) AS final;
grunt> classified = FOREACH sentiment_score
    GENERATE id, ( (final >= 0)? 'POSITIVE' : 'NEGATIVE' ) AS classification,
    final AS score;
grunt> final = ORDER classified BY score DESC;
grunt> STORE final INTO 'sentiment_analysis';
```

10.5.3-Data Types in pig-

Table 10.1. Pig scalar types

Category	Type	Description	Example
Numeric	int	32-bit signed integer	12
	long	64-bit signed integer	34L
	float	32-bit floating-point number	2.18F
	double	64-bit floating-point number	3e-17
Text	chararray	String or array of characters	hello world
Binary	bytearray	Blob or array of bytes	N/A

Table 10.2 Pig relational operators

Category	Operator	Description
Loading and storing	LOAD	Loads data from the file system or other storage source
	STORE	Saves a relation to the file system or other storage
	DUMP	Prints a relation to the console
Filtering and projection	FILTER	Selects tuples from a relation based on some condition
	DISTINCT	Removes duplicate tuples in a relation
	FOREACH...GENERATE	Generates data transformations based on columns of data.
	MAPREDUCE	Executes native MapReduce jobs inside a Pig script
	STREAM	Sends data to an external script or program
	SAMPLE	Selects a random sample of data based on the specified sample size
Grouping and joining	JOIN	Joins two or more relations
	COGROUP	Groups the data from two or more relations
	GROUP	Groups the data in a single relation
	CROSS	Creates the cross-product of two or more relations
Sorting	ORDER	Sorts the relation by one or more fields
	LIMIT	Limits the number of tuples returned from a relation
Combining and splitting	UNION	Computes the union of two or more relations
	SPLIT	Partitions a relation into two or more relations

10.5.4-User-Defined Functions

Pig provides extensive support for such user-defined functions (UDFs), and currently provides integration libraries for six languages: Java, Jython, Python, JavaScript, Ruby, and Groovy. In this scenario, we would like to write a custom eval UDF in java that will allow us to convert the score classification evaluation into a function.

```
package com.statistics.pig;

import java.io.IOException;

import org.apache.pig.EvalFunc;
import org.apache.pig.backend.executionengine.ExecException;
import org.apache.pig.data.Tuple;

public class Classify extends EvalFunc {

    @Override
    public String exec(Tuple input) throws IOException {
        if (args == null || args.size() == 0) {
            return false;
        }
        try {
            Object object = args.get(0);
            if (object == null) {
                return false;
            }
            int i = (Integer) object;
            if (i >= 0) {
                return new String("POSITIVE");
            } else {
                return new String("NEGATIVE");
            }
        } catch (ExecException e) {
            throw new IOException(e);
        }
    }
}
```

To use this function, we need to compile it, package it into a JAR file, and then register the JAR with Pig by using the REGISTER operator:

```
grunt> REGISTER statistics-pig.jar;
```

We can then invoke the function in a command:

```
grunt> classified = FOREACH sentiment_score GENERATE id,
    com.statistics.pig.Classify(final) AS classification, final AS score;
```

10.5.5.-Wrapping Up

Pig can be a powerful tool for users who prefer a procedural programming model. It provides the ability to control data checkpoints in the pipeline, as well as fine-grained controls over how the data is processed at each step. This makes Pig a great choice when you require more flexibility in controlling the sequence of operations in a data flow (e.g., an extract, form, and load, or ETL, process), or when you are working with semi-structured data that may not lend itself well to Hive's SQL syntax.

10.6-Spark's Higher-Level APIs

In practice, a typical analytic workflow will entail some combination of relational queries, procedural programming, and custom processing, which means that most end-to-end Hadoop workflows involve integrating

several disparate components and switching between different programming APIs. Spark, in contrast, provides two major programming advantages over the MapReduce-centric Hadoop stack:

- Built-in expressive APIs in standard, general-purpose languages like Scala, Java, Python, and R
- A unified programming interface that includes several built-in higher-level libraries to support a broad range of data processing tasks, including complex interactive analysis, structured querying, stream processing, and machine learning.

10.6.1-Spark SQL

Spark SQL is a module in Apache Spark that provides a relational interface to work with structured data using familiar SQL-based operations in Spark. It can be accessed through JDBC/ODBC connectors, a built-in interactive Hive console, or via its built-in APIs. The last method of access is the most interesting and powerful aspect of Spark SQL; because Spark SQL actually runs as a library on top of Spark's Core engine and APIs, we can access the Spark SQL API using the same programming interface that we use for Spark's RDD APIs, as shown in Figure 10-9.

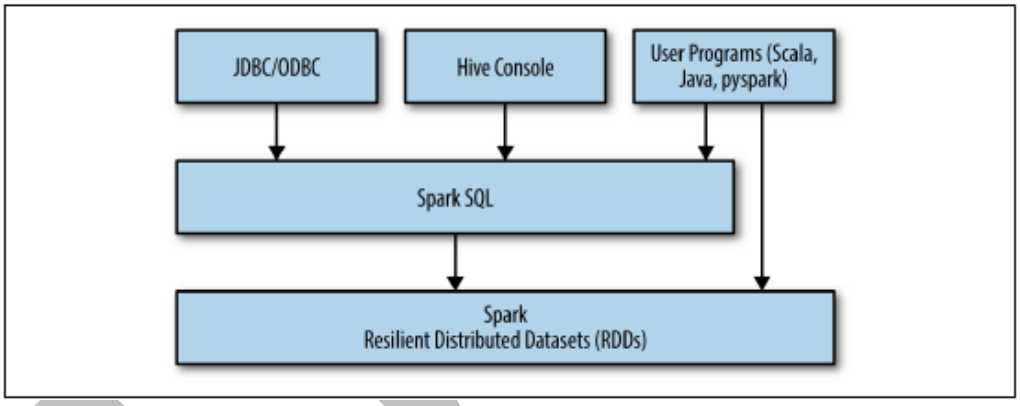


Figure 10.9. Spark SQL interface

This allows us to seamlessly combine and leverage the benefits of relational queries with the flexibility of Spark's procedural processing and the power of Python's analytic libraries, all in one programming environment.

Let's write a simple program that uses the Spark SQL API to load JSON data and query it. You can enter these commands directly in a running pyspark shell or in a Jupyter notebook that is using a pyspark kernel; in either case, ensure that you have a running SparkContext, which we'll assume is referenced by the variable sc.

To begin, we'll need to import the SQLContext class from the *pyspark.sql* package.

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
```

With the file properly formatted, we can easily load its contents by calling sqlCon text.read.json and passing it the path to the file:

```
parking = sqlContext.read.json('../data/sf_parking/sf_parking_clean.json')
```

In order to run a SQL statement against our dataset, we must first register it as a temporary named table:

```
parking.registerTempTable("parking")
```

This allows us to run additional table and SQL methods, including show, which will display the first 20 rows of data in a tabular format:

```
parking.show()
```

To execute a SQL statement on the parking table, we use the sql method, passing it the full query.

10.6.2-DataFrames

DataFrames are the underlying data abstraction in Spark SQL. The data frame concept should be very familiar to users of Python's Pandas or R, and in fact, Spark's DataFrames are interoperable with native Pandas (using pyspark) and R data frames (using SparkR). In Spark, a DataFrame also represents a tabular collection of data with a defined schema. The key difference between a Spark DataFrame and a dataframe in Pandas or R is that a Spark DataFrame is a distributed collection that actually wraps an RDD; you can think of it as an RDD of row objects.

Additionally, DataFrame operations entail many optimizations under the hood that not only compile the query plan into executable code, but substantially improve the performance and memory-footprint over comparable handcoded RDD operations. In fact, in a benchmark test that compared the runtimes between DataFrames code that aggregated 10 million integer pairs against equivalent RDD code, DataFrames were not only found to be up to 4–5x faster for these workloads, but they also close the performance gap between Python and JVM implementations.

The concise and intuitive semantics of the DataFrames API coupled with the performance optimizations provided by its computational engine was the impetus to make DataFrames the main interface for all of Spark's modules, including Spark SQL, RDDs, MLlib, and GraphX. In this way, the DataFrames API provides a unified engine across all of Spark's data sources, workloads, and environments, as shown in Figure 10-10.

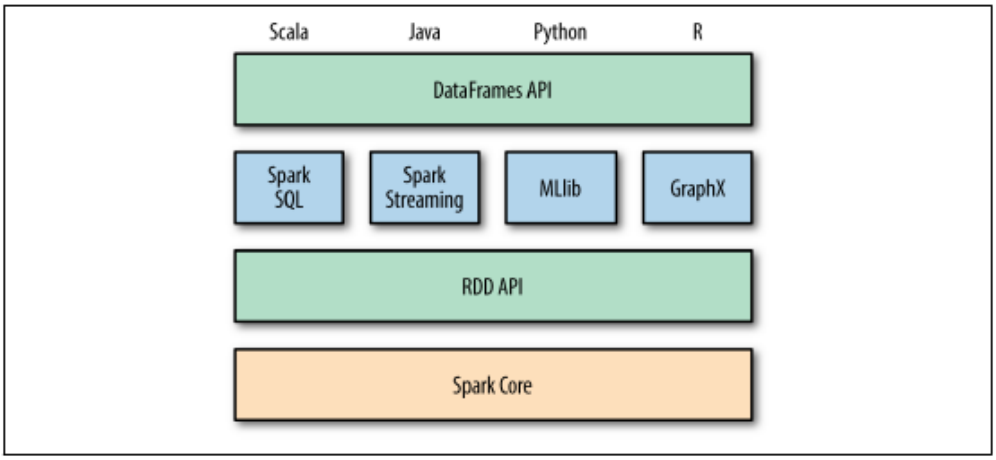


Figure 10.10 Data Frames as sparks unified interface

Example of chaining several simple DataFrame operations:

```
from pyspark.sql import functions as F

aggr_by_type = parking.select("primetype", "secondtype", "regcap") \
                      .where("trim(primetype) != ''") \
                      .groupBy("primetype", "secondtype") \
                      .agg(
                         F.count("*").alias("count"),
                         F.round(F.avg("regcap"), 0).alias("avg_spaces")
                       ) \
                       .sort("count", ascending=False)
```

The advantage of this approach over raw SQL is that we can easily iterate on a complex query by successively chaining and testing operations. Additionally, we have access to a rich collection of built-in functions from the DataFrames API, including the count, round, and avg aggregation functions that we used previously. The pyspark.sql.functions module also contains several mathematical and statistical utilities that include functions for:

- Random data generation
- Summary and descriptive statistics
- Sample covariance and correlation
- Cross tabulation (a.k.a. contingency table)
- Frequency computation
- Mathematical functions