ALPHA ARTS AND SCIENCE COLLEGE
PORUR, CHENNAI
(A Christian Minority Institution)
(Reaccredited by NAAC and Affiliated to University of Madras)

Department of Computer Science – Shift I

Notes of Lesson

Introduction to Data Science
(SE26B)

Prepared By

Mr.M.Mohamed Suhail,
Co-convener, Career Development, Training and Placement Cell,
SPOC, Naan Mudhalvan Scheme,
Assistant Professor & Placement Coordinator,
Department of Computer Science,
Alpha Arts and Science College, Chennai

Class: III B.Sc Computer Science **Semester : VI**

Syllabus

UNIT-I

Introduction to Data Science – Benefits and uses – Facets of data – Data science process – Big data ecosystem and data science

1.1 Introduction to Data Science and Big Data

What is Data Science?

Data science is the study of data to extract meaningful insights for business. It is a multidisciplinary approach that combines principles and practices from the fields of mathematics, statistics, artificial intelligence, and computer engineering to analyze large amounts of data.

Data science is an evolutionary extension of statistics capable of dealing with the massive amounts of data produced today. It adds methods from computer science to the repertoire of statistics.

What is Big Data?

Big data refers to a massive amount of information that is too large and complex to handle using traditional methods. It includes diverse types of data from various sources, requiring special tools and techniques to process, analyze, and extract valuable insights.

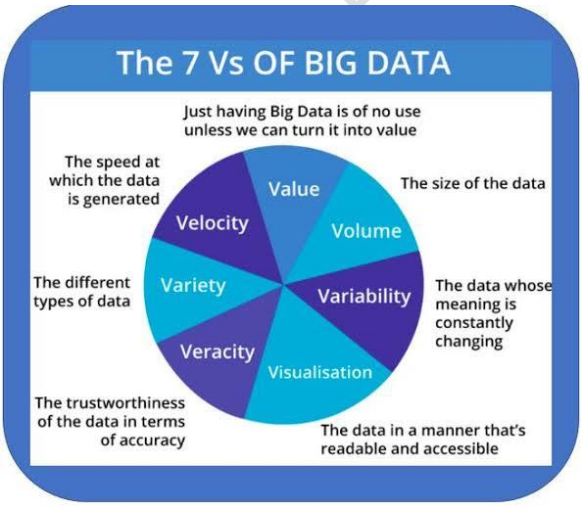


Figure 1.1: 7 Vs of Big Data

1.2 Benefits and uses of Data Science and Big data

Data Science and Big Data offer a wide range of benefits and applications across various industries. Here are some key benefits and uses:

1. **Improved Decision-Making**: Data Science and Big Data enable organizations to make more informed and data-driven decisions. By analyzing large volumes of data, businesses can uncover patterns, trends, and insights that provide valuable information for decision-making processes.

2. **Enhanced Efficiency and Productivity**: Data Science techniques help streamline operations, optimize processes, and identify areas for improvement. By analyzing data, organizations can identify bottlenecks, inefficiencies, and opportunities for automation, leading to increased productivity and cost savings.

3. **Personalized Customer Experiences**: Data Science enables organizations to understand their customers better. By analyzing customer data, businesses can personalize marketing campaigns, tailor products or services, and improve customer satisfaction by offering customized experiences based on individual preferences and needs.

4. **Fraud Detection and Risk Management**: Data Science plays a crucial role in detecting fraudulent activities and managing risks. By analyzing patterns and anomalies in large datasets, organizations can identify potential fraud instances, prevent financial losses, and mitigate risks in sectors such as banking, insurance, and cybersecurity.

5. **Predictive Analytics and Forecasting**: Data Science leverages predictive models to make accurate forecasts and predictions. Organizations can use historical data to anticipate future trends, demand patterns, and market behavior, enabling them to make proactive decisions and optimize resource allocation.

6. **Healthcare and Medical Research**: Data Science and Big Data have significant applications in healthcare and medical research. By analyzing patient records, clinical data, and genetic information, data scientists can gain insights into disease patterns, develop predictive models for early diagnosis, and improve treatment outcomes.

7. **Smart Cities and Internet of Things (IoT)**: Big Data analytics helps in managing and optimizing urban infrastructure. By analyzing data from sensors, devices, and various sources in smart cities, governments and urban planners can improve traffic management, energy efficiency, waste management, and overall urban livability.

8. **Scientific Research and Exploration**: Data Science aids scientific research by processing and analyzing large datasets generated in fields such as astronomy, genomics, climate science, and particle physics. It helps researchers extract insights, discover patterns, and advance knowledge in their respective domains.

   These are just a few examples of the benefits and uses of Data Science and Big Data. The applications are vast and continually expanding as organizations across industries recognize the value of leveraging data for better decision-making, innovation, and competitive advantage.

1.3 Facets of Data – Different types of Data

In data science and big data you'll come across many different types of data, and each of them tends to require different tools and techniques. The main categories of data are these:

- Structured
- Unstructured
- Natural language
- Machine-generated
- Graph-based

■ Audio, video, and images
■ Streaming

Structured Data

- Structured data refers to organized and formatted data that is easily identifiable and stored in a predefined schema or model.
- It is highly organized and typically fits well into traditional databases or spreadsheets.
- Structured data follows a specific data model, where each data element has a defined format, type, and relationship with other elements.
- Structured data is data that depends on a data model and resides in a fixed field within a record.
- As such, it's often easy to store structured data in tables within databases or Excel files (figure 1.1). SQL, or Structured Query Language, is the preferred way to manage and query data that resides in databases.
- You may also come across structured data that might give you a hard time storing it in a traditional relational database. Hierarchical data such as a family tree is one such example.

Characteristics of structured data include:

- **Organized Format**: Structured data is organized in a consistent and predefined structure, following a specific data model such as tables, rows, and columns in a relational database.

- **Clear Schema**: Structured data has a well-defined schema that outlines the structure, relationships, and constraints of the data. This schema provides a blueprint for how the data is organized and represented.

- **Fixed Data Types**: Structured data has fixed and predefined data types for each attribute or column, such as integers, strings, dates, or booleans. These data types determine the nature and format of the data stored in the column.

- **Easy to Query and Analyze**: The organized structure of structured data allows for easy querying and analysis using SQL (Structured Query Language) or other database query languages. Queries can retrieve specific information from the data based on predefined conditions.

- **Examples of structured data include** customer information stored in a customer database (with fields such as name, address, phone number, and email), financial transaction records, inventory lists, and sales reports.

Figure 1.2: An Excel table is an example of structured data

	roll_no	student_name	sem	college_code	college_name	course_name	partner_name	mand
2	unm1429222006735	THAMEEM H	6	unm1429	Alpha Arts and Science College	Java & Programming	Veranda	NO
3	unm1429222006736	KARTHIKEYAN L	6	unm1429	Alpha Arts and Science College	Python	GUVI	NO
4	unm1429222006737	SANOJ S	6	unm1429	Alpha Arts and Science College	Java & Programming	Veranda	NO
5	unm1429222006738	KEERTHANA V	6	unm1429	Alpha Arts and Science College	Matlab	GUVI	NO
6	unm1429222006739	AAKASH K	6	unm1429	Alpha Arts and Science College			NO
7	unm1429222006740	ABINASH S	6	unm1429	Alpha Arts and Science College	Python	GUVI	NO
8	unm1429222006741	ABINESH D	6	unm1429	Alpha Arts and Science College	App Development	Veranda	NO
9	unm1429222006742	AKASH BILL BRIGHT M	6	unm1429	Alpha Arts and Science College	Data Science	Veranda	NO
10	unm1429222006743	ALTHAF HUSSAIN J	6	unm1429	Alpha Arts and Science College	Python	GUVI	NO
11	unm1429222006744	AMRISH KANNAN B	6	unm1429	Alpha Arts and Science College	Java & Programming	Veranda	NO
12	unm1429222006745	ANAND M	6	unm1429	Alpha Arts and Science College	Python	GUVI	NO
13	unm1429222006746	ANTONY JOHNPETER G	6	unm1429	Alpha Arts and Science College	DevOps	Veranda	NO
14	unm1429222006747	ARUNKUMAR S	6	unm1429	Alpha Arts and Science College			NO
15	unm1429222006749	BALAJI R	6	unm1429	Alpha Arts and Science College	App Development	Veranda	NO
16	unm1429222006750	DEEPAKRAM B	6	unm1429	Alpha Arts and Science College	App Development	Veranda	NO
17	unm1429222006751	DHANASEKAR R	6	unm1429	Alpha Arts and Science College	Java & Programming	Veranda	NO
18	unm1429222006752	DHANUSH KODI V	6	unm1429	Alpha Arts and Science College			NO
19	unm1429222006753	DINESH G	6	unm1429	Alpha Arts and Science College	App Development	Veranda	NO
20	unm1429222006754	DINESH KUMAR M	6	unm1429	Alpha Arts and Science College	Python	GUVI	NO
21	unm1429222006755	FRANKLIN JAMES LUKE S	6	unm1429	Alpha Arts and Science College	App Development	Veranda	NO
22	unm1429222006756	GANESH B	6	unm1429	Alpha Arts and Science College			NO
23	unm1429222006758	GOUTHAMAN P	6	unm1429	Alpha Arts and Science College	Python	GUVI	NO

Unstructured Data

- Unstructured data refers to information that does not have a predefined or organized format. It does not fit neatly into traditional rows, columns, or tables like structured data.
- Unstructured data is typically human-generated and can exist in various forms, including text documents, emails, social media posts, audio and video files, images, presentations, and more.

Key characteristics of unstructured data include:

Lack of Organization: Unstructured data does not have a predefined structure or schema. It often consists of free-form text, narratives, or multimedia content without specific rules or patterns.

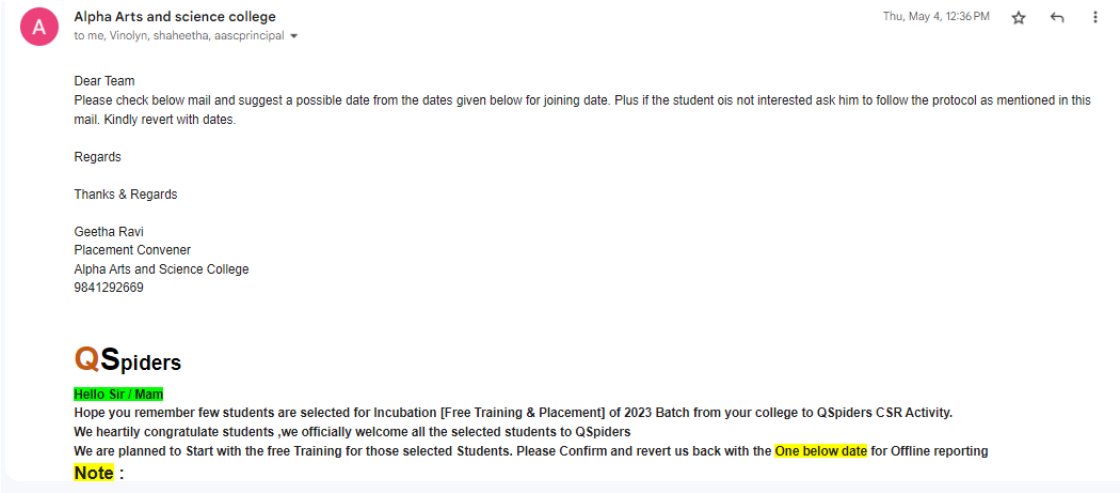
Varied Formats: Unstructured data can take many different formats and can include text, audio, video, images, and combinations thereof. Each format requires specialized techniques and tools for processing and analysis.

High Volume: Unstructured data is generated in large volumes, often exceeding the capacity of traditional data management systems. It poses challenges in terms of storage, processing, and analysis due to its sheer volume.

Difficult to Analyze: Analyzing unstructured data is more complex compared to structured data. It involves techniques such as natural language processing, text mining, sentiment analysis, image recognition, and audio transcription to extract insights and meaning from the data.

Examples of unstructured data include social media posts, customer reviews, emails, medical records, news articles, sensor data, satellite images, and video streams.

Figure: 1.3 Example of Unstructured Data



Alpha Arts and science college
to me, Vinolyn, shaheetha, aascprincipal

Thu, May 4, 12:36 PM

Dear Team
Please check below mail and suggest a possible date from the dates given below for joining date. Plus if the student ois not interested ask him to follow the protocol as mentioned in this mail. Kindly revert with dates.

Regards

Thanks & Regards

Geetha Ravi
Placement Convener
Alpha Arts and Science College
9841292669

QSpiders

Hello Sir / Mam
Hope you remember few students are selected for Incubation [Free Training & Placement] of 2023 Batch from your college to QSpiders CSR Activity.
We heartily congratulate students ,we officially welcome all the selected students to QSpiders
We are planned to Start with the free Training for those selected Students. Please Confirm and revert us back with the One below date for Offline reporting
Note :

Natural language

- Natural Language Processing or NLP in data science is the automatic manipulation of natural languages, like speech and text, by using software that helps computers observe, analyse, understand, and derive valuable meaning from natural or human-spoken languages.
- Natural language is a special type of unstructured data. It's challenging to process because it requires knowledge of specific data science techniques and linguistics.
- In other words, it is a branch of data science that focuses on training computers to process and interpret conversations in text format in a way human do by listening.
- It is a field that is developing methodologies for filling the gap between Data Science and human languages. NLP applications are difficult and challenging during development as computers require humans to interact with them using programming languages like Java, Python, etc.,

Machine-generated data

- Machine-generated data is information that's automatically created by a computer, process, application, or other machine without human intervention. Machine-generated data is becoming a major data resource and will continue to do so.
- The analysis of machine data relies on highly scalable tools, due to its high volume and speed. Examples of machine data are web server logs, call detail records, network event logs, and telemetry as shown in Figure below:

```
CSIPERF:TXCOMMIT;313236
2014-11-28 11:36:13, Info                  CSI    00000153 Creating NT transaction (seq
69), objectname [6]"(null)"
2014-11-28 11:36:13, Info                  CSI    00000154 Created NT transaction (seq 69)
result 0x00000000, handle @0x4e54
2014-11-28 11:36:13, Info                  CSI    00000155@2014/11/28:10:36:13.471
Beginning NT transaction commit...
2014-11-28 11:36:13, Info                  CSI    00000156@2014/11/28:10:36:13.705 CSI perf
trace:
CSIPERF:TXCOMMIT;273983
2014-11-28 11:36:13, Info                  CSI    00000157 Creating NT transaction (seq
70), objectname [6]"(null)"
2014-11-28 11:36:13, Info                  CSI    00000158 Created NT transaction (seq 70)
result 0x00000000, handle @0x4e5c
2014-11-28 11:36:13, Info                  CSI    00000159@2014/11/28:10:36:13.764
Beginning NT transaction commit...
2014-11-28 11:36:14, Info                  CSI    0000015a@2014/11/28:10:36:14.094 CSI perf
trace:
CSIPERF:TXCOMMIT;386259
2014-11-28 11:36:14, Info                  CSI    0000015b Creating NT transaction (seq
71), objectname [6]"(null)"
2014-11-28 11:36:14, Info                  CSI    0000015c Created NT transaction (seq 71)
result 0x00000000, handle @0x4e5c
2014-11-28 11:36:14, Info                  CSI    0000015d@2014/11/28:10:36:14.106
Beginning NT transaction commit...
2014-11-28 11:36:14, Info                  CSI    0000015e@2014/11/28:10:36:14.428 CSI perf
trace:
CSIPERF:TXCOMMIT;375581
```

Graph Based or Network Data

- "Graph data" can be a confusing term because any data can be shown in a graph.
- "Graph" in this case points to mathematical graph theory. In graph theory, a graph is a mathematical structure to model pair-wise relationships between objects. Graph or network data is, in short, data that focuses on the relationship or adjacency of objects.
- The graph structures use nodes, edges, and properties to represent and store graphical data. Graph-based data is a natural way to represent social networks, and its structure allows you to calculate specific metrics such as the influence of a person and theshortest path between two people.
- Examples of graph-based data can be found on many social media websites (figure 1.4). For instance, on LinkedIn you can see who you know at which company.
    - Your follower list on Twitter is another example of graph-based data.
    - The power and sophistication come from multiple, overlapping graphs of the same nodes.
    - For example, imagine the connecting edges here to show "friends" on Facebook. Imagine another graph with the same people which connects business colleagues via LinkedIn.
    - Imagine a third graph based on movie interests on Netflix. Overlapping the three different-looking graphs makes more interesting questions possible.

      Graph databases are used to store graph-based data and are queried with specialized query languages such as SPARQL.
      Graph data poses its challenges, but for a computer interpreting additive and image data, it can be even more difficult.

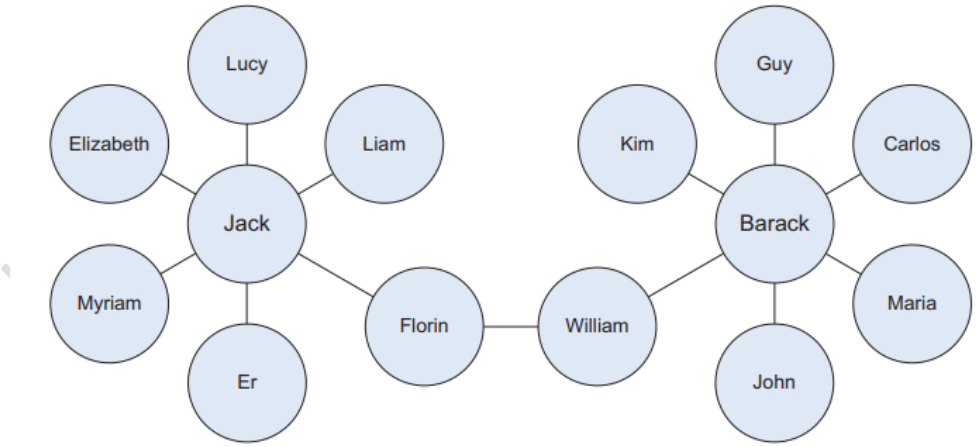


Figure: 1.4 Friends in a social network are an example of graph-based data.

Audio, image, and video

- Audio, image, and video are data types that pose specific challenges to a data scientist.
- Tasks that are trivial for humans, such as recognizing objects in pictures, turn out to be challenging for computers. MLBAM (Major League Baseball Advanced Media) announced in 2014 that they'll increase video capture to approximately 7 TB per game for the purpose of live, in-game analytics.

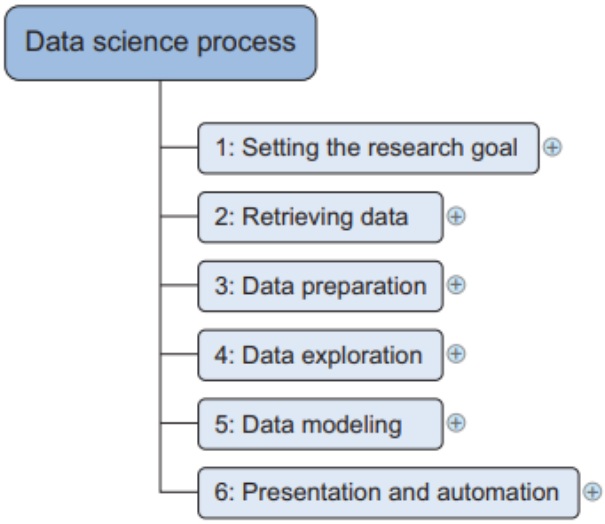
- High-speed cameras at stadiums will capture ball and athlete movements to calculate in real time, for example, the path taken by a defender relative to two baselines.
- Recently a company called DeepMind succeeded at creating an algorithm that's capable of learning how to play video games.
- This algorithm takes the video screen as input and learns to interpret everything via a complex process of deep learning.
- It's a remarkable feat that prompted Google to buy the company for their own Artificial Intelligence (AI) development plans.
- The learning algorithm takes in data as it's produced by the computer game; it's streaming data

Streaming data

- While streaming data can take almost any of the previous forms, it has an extra property. The data flows into the system when an event happens instead of being loaded into a data store in a batch.
- Although this isn't really a different type of data, we treat it here as such because you need to adapt your process to deal with this type of information.
- Examples are the "What's trending" on Twitter, live sporting or music events, and the stock market.

1.4 Data Science Process

The data science process typically consists of six steps, as you can see in the mind map in figure given below



The data science process typically involves several iterative steps to extract valuable insights and knowledge from data. While the specific steps may vary depending on the context and project requirements, here are six common steps in the data science process:

Setting the research goal

Data science is mostly applied in the context of an organization. When the business asks you to perform a data science project, you'll first prepare a project charter. This charter contains information such as what you're going to research, how the company benefits from that, what data and resources you need, a timetable, and deliverables.

Retrieving data

The second step is to collect data. You've stated in the project charter which data you need and where you can find it. In this step you ensure that you can use the data in your program, which means checking the existence of, quality, and access to the data. Data can also be delivered by third-party companies and takes many forms ranging from Excel spreadsheets to different types of databases.

Data preparation

Data collection is an error-prone process; in this phase you enhance the quality of the data and prepare it for use in subsequent steps. This phase consists of three subphases: data cleansing removes false values from a data source and inconsistencies across data sources, data integration enriches data sources by combining information from multiple data sources, and data transformation ensures that the data is in a suitable format for use in your models.

Data exploration

Data exploration is concerned with building a deeper understanding of your data. You try to understand how variables interact with each other, the distribution of the data, and whether there are outliers. To achieve this you mainly use descriptive statistics, visual techniques, and simple modeling. This step often goes by the abbreviation EDA, for Exploratory Data Analysis.

Data modeling or model building

In this phase you use models, domain knowledge, and insights about the data you found in the previous steps to answer the research question. You select a technique from the fields of statistics, machine learning, operations research, and so on. Building a model is an iterative process that involves selecting the variables for the model, executing the model, and model diagnostics.

Presentation and automation

Finally, you present the results to your business. These results can take many forms, ranging from presentations to research reports. Sometimes you'll need to automate the execution of the process because the business will want to use the insights you gained in another project or enable an operational process to use the outcome from your model

It's important to note that the data science process is iterative, and steps may be revisited as new insights or challenges arise. Data scientists often iterate between different stages, refining models, and adjusting based on feedback and continuous learning from the data.

1.5 Big Data Ecosystem and Data Science

- Currently many big data tools and frameworks exist, and it's easy to get lost because new technologies appear rapidly.
- It's much easier once you realize that the big data ecosystem can be grouped into technologies that have similar goals and functionalities, which we'll discuss in this section.
- Data scientists use many different technologies, but not all of them as of now
- The mind map in figure 1.5 shows the components of the big data ecosystem and where the different technologies belong.
- Let's look at the different groups of tools in this diagram and see what each does.

1.5.1 Distributed file systems

- A distributed file system is similar to a normal file system, except that it runs on multiple servers at once. Because it's a file system, you can do almost all the same things you'd do on a normal file system.

- Actions such as storing, reading, and deleting files and adding security to files are at the core of every file system, including the distributed one.
- Distributed file systems have significant advantages:
    - ■ They can store files larger than any one computer disk.
    - ■ Files get automatically replicated across multiple servers for redundancy or parallel operations while hiding the complexity of doing so from the user.
    - ■ The system scales easily: you're no longer bound by the memory or storage restrictions of a single server
- The best-known distributed file system at this moment is the **Hadoop File System (HDFS)**. It is an open source implementation of the Google File System. However, many other distributed file systems exist: Red Hat Cluster File System, Ceph File System and Tachyon File System.

1.5.2 Distributed programming framework

- Once you have the data stored on the distributed file system, you want to exploit it.
- One important aspect of working on a distributed hard disk is that you won't move your data to your program, but rather you'll move your program to the data.
- When you start from scratch with a normal general-purpose programming language such as C, Python, or Java, you need to deal with the complexities that come with distributed programming, such as restarting jobs that have failed, tracking the results from the different subprocesses, and so on.
- Luckily, the open source community has developed many frameworks to handle this for you, and these give you a much better experience working with distributed data and dealing with many of the challenges it carries.

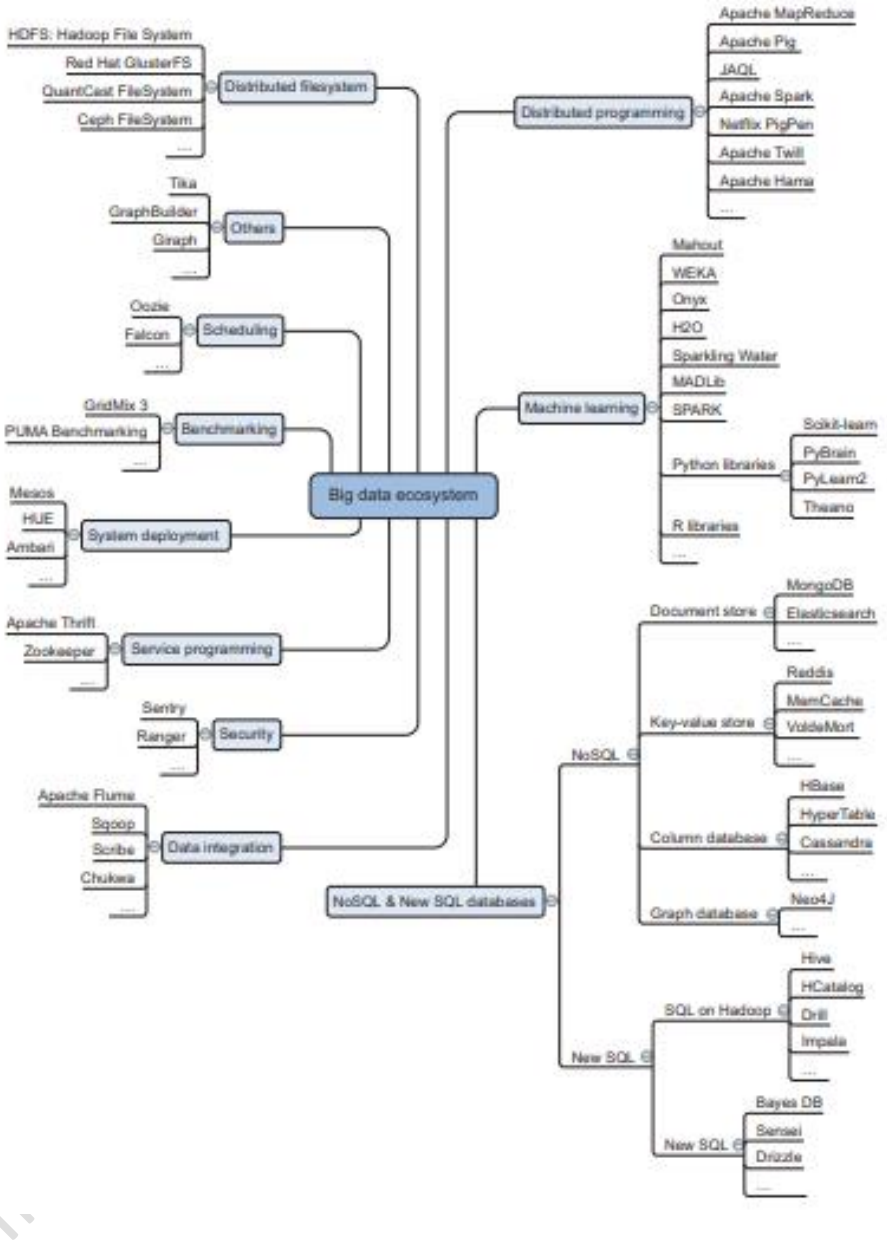


Figure: 1.5 Big data technologies can be classified into a few main components.

1.5.3 Data integration framework

Once you have a distributed file system in place, you need to add data. You need to move data from one source to another, and this is where the data integration frameworks such as Apache Sqoop and Apache Flume excel. The process is similar to an extract, transform, and load process in a traditional data warehouse.

1.5.4 Machine learning frameworks

When you have the data in place, it's time to extract the coveted insights. This is where you rely on the fields of machine learning, statistics, and applied mathematics. With the amount of data, we need to analyze today, this becomes problematic, and specialized frameworks and libraries are required to deal with this amount of data. The most popular machine-learning library for Python is Scikit-learn. It's a great machine-learning toolbox, and we'll use it later in the book. There are, of course, other Python libraries:

PyBrain for neural networks—Neural networks are learning algorithms that mimic the human brain in learning mechanics and complexity. Neural networks are often regarded as advanced and black box.

NLTK or Natural Language Toolkit—As the name suggests, its focus is working with natural language. It's an extensive library that comes bundled with a number of text corpuses to help you model your own data.

Pylearn2—Another machine learning toolbox but a bit less mature than Scikit-learn.

TensorFlow—A Python library for deep learning provided by Google

Spark is a new Apache licensed machine-learning engine, specializing in real-learn-time machine learning.

1.5.5 NoSQL databases

NoSQL (Not Only SQL) databases are a type of database management system that provides a non-relational approach to data storage and retrieval.

Unlike traditional relational databases, which are based on structured query language (SQL) and use a tabular format with predefined schemas, NoSQL databases offer a flexible and scalable alternative for managing large volumes of unstructured or semi-structured data.

Many different types of databases have arisen, but they can be categorized into the following types:

- **Column databases** —Data is stored in columns, which allows algorithms to perform much faster queries. Newer technologies use cell-wise storage. Table-like structures are still important.
- **Document stores**—Document stores no longer use tables, but store every observation in a document. This allows for a much more flexible data scheme.
- **Streaming data**—Data is collected, transformed, and aggregated not in batches but in real time. Although we've categorized it here as a database to help you in tool selection, it's more a particular type of problem that drove creation of technologies such as Storm.
- **Key-value stores**—Data isn't stored in a table; rather you assign a key for every value, such as org. marketing. sales.2015: 20000. This scales well but places almost all the implementation on the developer
- **SQL on Hadoop**—Batch queries on Hadoop are in a SQL-like language that uses the map-reduce framework in the background.
- **New SQL**—This class combines the scalability of NoSQL databases with the advantages of relational databases. They all have a SQL interface and a relational data model.
- **Graph databases**—Not every problem is best stored in a table. Particular problems are more naturally translated into graph theory and stored in graph databases. A classic example of this is a social network.

1.5.6 Scheduling tools

Scheduling tools help you automate repetitive tasks and trigger jobs based on events such as adding a new file to a folder. These are similar to tools such as CRON on Linux but are specifically developed

for big data. You can use them, for instance, to start a MapReduce task whenever a new dataset is available in a directory.

1.5.7 Benchmarking tools

This class of tools was developed to optimize your big data installation by providing standardized profiling suites. A profiling suite is taken from a representative set of big data jobs. Benchmarking and optimizing the big data infrastructure and configuration aren't often jobs for data scientists themselves but for a professional specialized in setting up IT infrastructure; thus they aren't covered in this book. Using an optimized infrastructure can make a big cost difference. For example, if you can gain 10% on a cluster of 100 servers, you save the cost of 10 servers.

1.5.8 System deployment

Setting up a big data infrastructure isn't an easy task and assisting engineers in deploying new applications into the big data cluster is where system deployment tools shine. They largely automate the installation and configuration of big data components. This isn't a core task of a data scientist.

1.5.9 Service programming

Suppose that you've made a world-class soccer prediction application on Hadoop, and you want to allow others to use the predictions made by your application. However, you have no idea of the architecture or technology of everyone keen on using your predictions. Service tools excel here by exposing big data applications to other applications as a service. Data scientists sometimes need to expose their models through services. The best-known example is the REST service; REST stands for representational state transfer. It's often used to feed websites with data.

1.5.10 Security

Big data security tools allow you to have central and fine-grained control over access to the data. Big data security has become a topic in its own right, and data scientists are usually only confronted with it as data consumers; seldom will they implement the security themselves.

Syllabus

Unit -2

The Data science process – Overview – research goals - retrieving data - transformation – Exploratory Data Analysis – Model building

2.1 Overview of the Data Science Process

- Following a structured approach to data science helps you to maximize your chances of success in a data science project at the lowest cost.
- It also makes it possible to take up a project as a team, with each team member focusing on what they do best.
- However, this approach may not be suitable for every type of project or be the only way to do good data science.
- The typical data science process consists of six steps through which you'll iterate, as shown in figure 2.1.

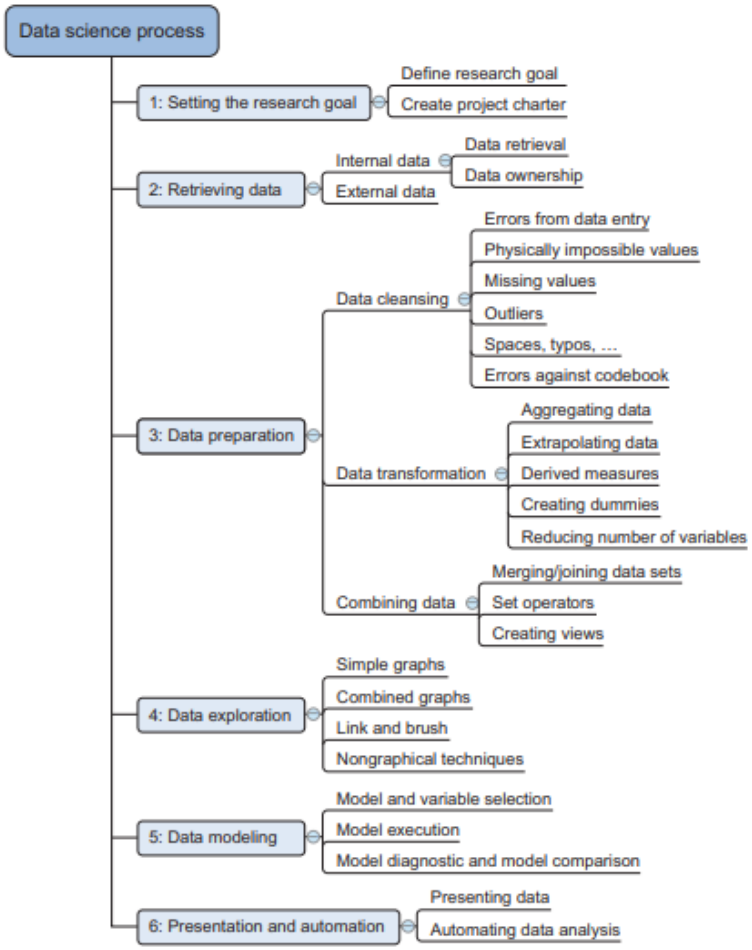


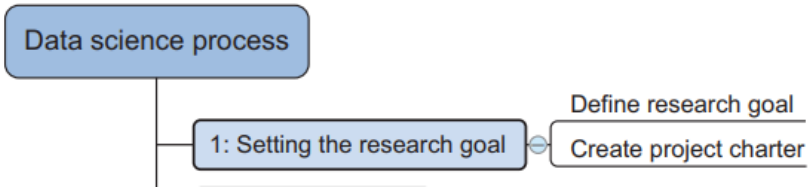
Figure 2.1 The six steps of the data science process

- Figure 2.1 summarizes the data science process and shows the main steps and actions you'll take during a project.

- The first step of this process is setting a research goal. The main purpose here is making sure all the stakeholders understand the what, how, and why of the project. In every serious project this will result in a project charter.
- The second phase is data retrieval. You want to have data available for analysis, so this step includes finding suitable data and getting access to the data from the data owner. The result is data in its raw form, which probably needs polishing and transformation before it becomes usable.
- The third step, now that you have the raw data, it's time to prepare it. This includes transforming the data from a raw form into data that's directly usable in your models. To achieve this, you'll detect and correct different kinds of errors in the data, combine data from different data sources, and transform it. If you have successfully completed this step, you can progress to data visualization and modelling.
- The fourth step is data exploration. The goal of this step is to gain a deep understanding of the data. You'll look for patterns, correlations, and deviations based on visual and descriptive techniques. The insights you gain from this phase will enable you to start modelling.
- The Fifth step is the model building (often referred to as "data modeling" throughout this book). It is now that you attempt to gain the insights or make the predictions stated in your project charter. Now is the time to bring out the heavy guns, but remember research has taught us that often (but not always) a combination of simple models tends to outperform one complicated model. If you've done this phase right, you're almost done.
- The last step of the data science model is presenting your results and automating the analysis, if needed. One goal of a project is to change a process and/or make better decisions. You may still need to convince the business that your findings will indeed change the business process as expected. This is where you can shine in your influencer role. The importance of this step is more apparent in projects on a strategic and tactical level. Certain projects require you to perform the business process over and over again, so automating the project will save time.
- Following these six steps pays off in terms of a higher project success ratio and increased impact of research results. This process ensures you have a well-defined research plan, a good understanding of the business question, and clear deliverables before you even start looking at data.

2.2 Defining research goals and creating a project charter

- A project starts by understanding the what, the why, and the how of your project (figure 2.2). What does the company expect you to do? And why does management place such a value on your research?
- Answering these three questions (what, why, how) is the goal of the first phase, so that everybody knows what to do and can agree on the best course of action.
- The outcome should be a clear research goal, a good understanding of the context, well-defined deliverables, and a plan of action with a timetable.
- This information is then best placed in a project charter. The length and formality can, of course, differ between projects and companies.



2.2.1 Spend time understanding the goals and context of your research

An essential outcome is the research goal that states the purpose of your assignment in a clear and focused manner. Understanding the business goals and context is critical for project success. Continue asking questions and devising examples until you grasp the exact business expectations, identify how your project fits in the bigger picture, appreciate how your research is going to change the business, and understand how they'll use your results.

2.2.2 Create a project charter

By creating a project charter specific to data science projects, stakeholders gain a shared understanding of the project's purpose, scope, and objectives. It sets the foundation for effective collaboration, resource allocation, and decision-making throughout the data science process, ensuring a successful and well-managed project.

A project charter requires teamwork, and your input covers at least the following:

- A clear research goals
- The project mission and context
- How you're going to perform your analysis
- What resources you expect to use
- Proof that it's an achievable project, or proof of concepts
- Deliverables and a measure of success
- A timeline

Project Title and Description: The project charter begins by providing a clear and concise title that reflects the data science project's purpose and a brief description of the problem it aims to address or the goal it seeks to achieve. It outlines the business context, the specific area of focus, and the expected outcomes.

Objectives: The project charter defines the specific objectives of the data science project. These objectives should be specific, measurable, achievable, relevant, and time-bound (SMART). They define what the project aims to accomplish in terms of insights, predictions, recommendations, or other desired outcomes.

Scope and Deliverables: The project charter clearly defines the scope of the data science project, including the data sources, variables, and timeframe. It outlines the boundaries within which the project will operate and the specific deliverables expected from the project, such as reports, models, visualizations, or other artefacts.

Stakeholders: The project charter identifies the key stakeholders involved in the data science project. This includes individuals or teams from the business side, data owners, subject matter experts, project sponsors, and other relevant parties. It defines their roles, responsibilities, and levels of engagement throughout the project.

Project Team: The project charter lists the members of the data science project team, including data scientists, analysts, engineers, and any other required roles. It outlines their responsibilities, reporting structure, and the collaboration mechanisms among team members.

Timeline and Milestones: The project charter includes a high-level timeline for the data science project, outlining major milestones and key activities. It provides an overview of the project's duration, important dates, and any critical dependencies or dependencies on external factors.

Constraints and Assumptions: The project charter acknowledges any known constraints or limitations that may impact the data science project. This could include data availability, budgetary constraints, resource limitations, or technical constraints. It also identifies any assumptions made during the project planning phase.

Risks and Mitigation Strategies: The project charter highlights potential risks and uncertainties associated with the data science project. It includes an initial assessment of risks and outlines strategies for mitigating those risks. This allows the project team to proactively address potential challenges and develop contingency plans.

Your client can use this information to make an estimation of the project costs and the data and people required for your project to become a success.

2.3 Retrieving data

- The next step in data science is to retrieve the required data
- Sometimes you need to go into the field and design a data collection process yourself, but most of the time you won't be involved in this step.
- Many companies will have already collected and stored the data for you, and what they don't have can often be bought from third parties.
- Don't be afraid to look outside your organization for data, because more and more organizations are making even high-quality data freely available for public and commercial use.

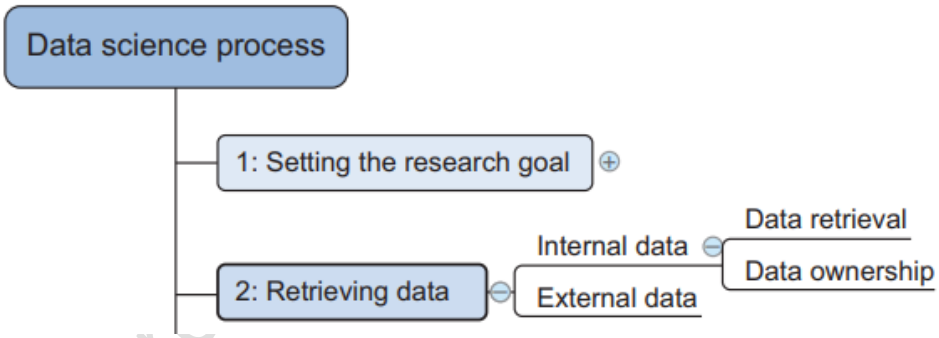


Figure 2.3 Retrieving Data

- Data can be stored in many forms, ranging from simple text files to tables in a database. The objective now is acquiring all the data you need.

2.3.1 Start with data stored within the company

- Your first act should be to assess the relevance and quality of the data that's readily available within your company. Most companies have a program for maintaining key data, so much of the cleaning work may already be done. This data can be stored in official data repositories such as databases, data marts, data warehouses, and data lakes maintained by a team of IT professionals
- A data mart is a subset of the data warehouse and geared toward serving a specific business unit. While data warehouses and data marts are home to pre-processed data, data lakes contain data in its natural or raw format.

- Finding data even within your own company can sometimes be a challenge. As companies grow, their data becomes scattered around many places. Knowledge of the data may be dispersed as people change positions and leave the company
- Getting access to data is another difficult task. Organizations understand the value and sensitivity of data and often have policies in place so everyone has access to what they need and nothing more. These policies translate into physical and digital barriers called Chinese walls.

2.3.2 Don't be afraid to shop around

- If data isn't available inside your organization, look outside your organization's walls. Many companies specialize in collecting valuable information. For instance, Nielsen and GFK are well known for this in the retail industry. Other companies provide data so that you, in turn, can enrich their services and ecosystem. Such is the case with Twitter, LinkedIn, and Facebook.
- Although data is considered an asset more valuable than oil by certain companies, more and more governments and organizations share their data for free with the world.
- This data can be of excellent quality; it depends on the institution that creates and manages it. The information they share covers a broad range of topics such as the number of accidents or amount of drug abuse in a certain region and its demographics.
- This data is helpful when you want to enrich proprietary data but also convenient when training your data science skills at home.
- Table 2.1 shows only a small selection from the growing number of open-data providers.

Table 2.1 A list of open-data providers that should get you started

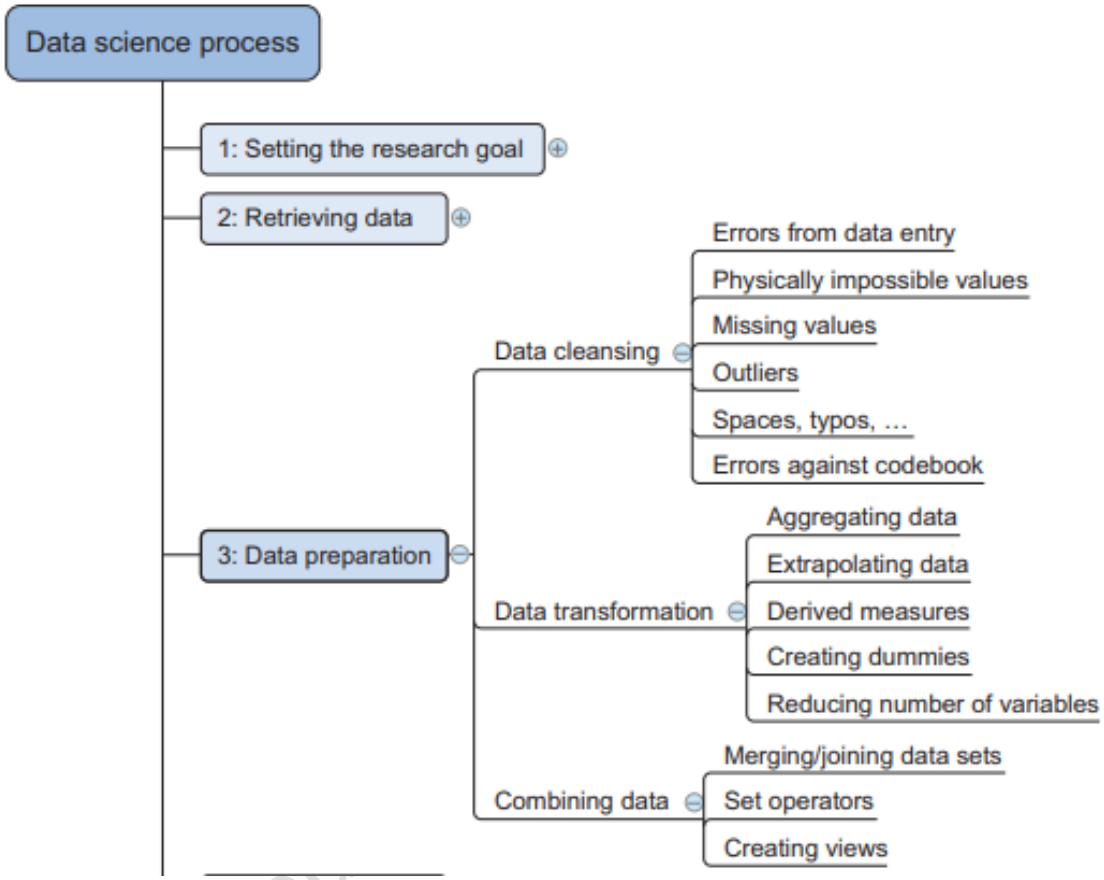
Open data site	Description
Data.gov	The home of the US Government's open data
https://open-data.europa.eu/	The home of the European Commission's open data
Freebase.org	An open database that retrieves its information from sites like Wikipedia, MusicBrains, and the SEC archive
Data.worldbank.org	Open data initiative from the World Bank
Aiddata.org	Open data for international development
Open.fda.gov	Open data from the US Food and Drug Administration

2.3.3 Do data quality checks now to prevent problems later

- Expect to spend a good portion of your project time doing data correction and cleansing, sometimes up to 80%. The retrieval of data is the first time you'll inspect the data in the data science process. Most of the errors you'll encounter during the datagathering phase are easy to spot, but being too careless will make you spend many hours solving data issues that could have been prevented during data import.
- During data retrieval, you check to see if the data is equal to the data in the source document and look to see if you have the right data types.

- With data preparation, you do a more elaborate check. If you did a good job during the previous phase, the errors you find now are also present in the source document.
- During the exploratory phase your focus shifts to what you can learn from the data. Now you assume the data to be clean and look at the statistical properties such as distributions, correlations, and outliers.

2.4 Cleansing, integrating, and transforming data



2.4.1 Data Cleansing

Data cleansing is a subprocess of the data science process that focuses on removing errors in your data so your data becomes a true and consistent representation of the processes it originates from.

By "**true and consistent representation**" we imply that at **least two types of errors exist**. The first type is the **interpretation error, such as when you take the value in your data for granted, like saying that a person's age is greater than 300 years.**

The second type of error points to inconsistencies between data sources or against your company's standardized values. An example of this class of errors is putting "Female" in one table and "F" in another when they represent the same thing: that the person is female.

Table 2.2 An overview of common errors

General solution	
Try to fix the problem early in the data acquisition chain or else fix it in the program.	
Error description	**Possible solution**
Errors pointing to false values within one data set	
Mistakes during data entry	Manual overrules
Redundant white space	Use string functions
Impossible values	Manual overrules
Missing values	Remove observation or value
Outliers	Validate and, if erroneous, treat as missing value (remove or insert)
Errors pointing to inconsistencies between data sets	
Deviations from a code book	Match on keys or else use manual overrules
Different units of measurement	Recalculate
Different levels of aggregation	Bring to same level of measurement by aggregation or extrapolation

- Sometimes you'll use more advanced methods, such as simple modelling, to find and identify data errors; diagnostic plots can be especially insightful.
- For example, in figure 2.5 we use a measure to identify data points that seem out of place. We do a regression to get acquainted with the data and detect the influence of individual observations on the regression line

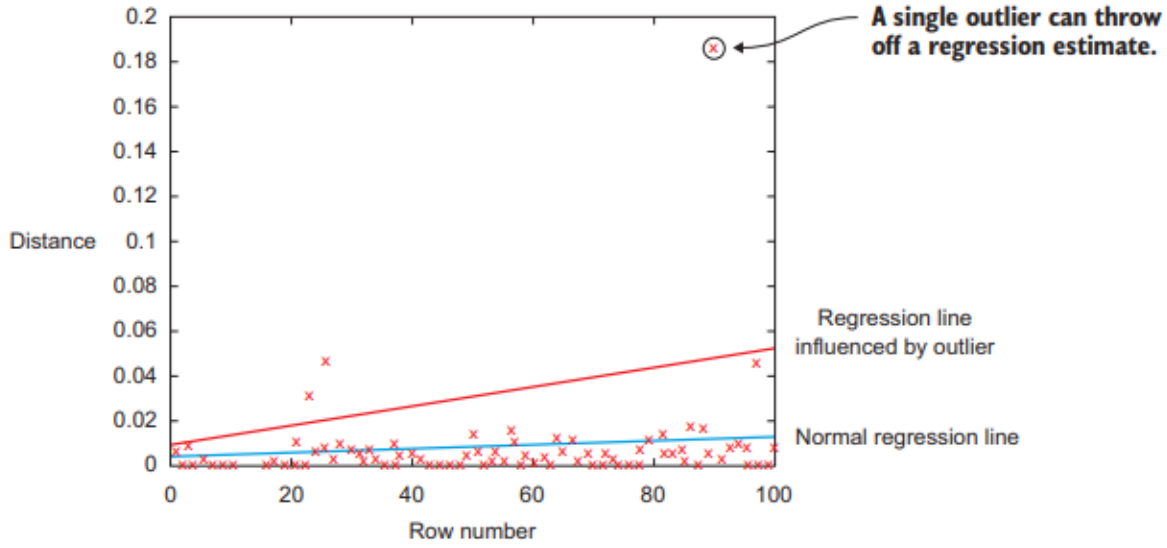


Figure 2.5 The encircled point influences the model heavily and is worth investigating because it can point to a region where you don't have enough data or might indicate an error in the data, but it also can be a valid data point.

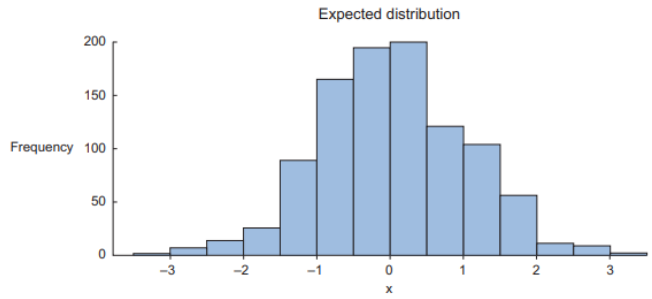
Data entry errors

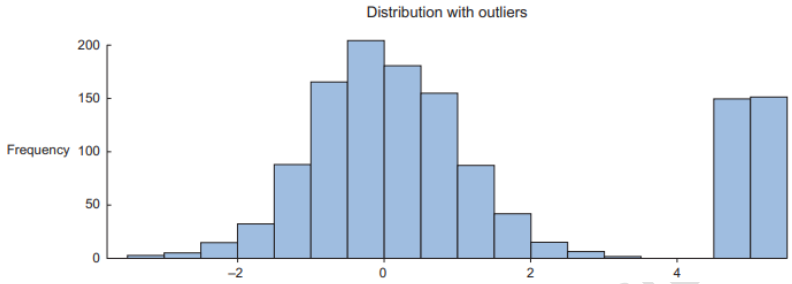
- Data entry errors can significantly impact the quality and reliability of data used in data science projects.
- These errors can occur due to various reasons, such as human mistakes, faulty equipment, or inconsistencies in data collection processes.
- It's important to be aware of and address data entry errors to ensure the accuracy and validity of data used in data science analysis.
- For small data sets you can check every value by hand. Detecting data errors when the variables you study don't have many classes can be done by tabulating the data with counts. When you have a variable that can take only two values: "Good" and "Bad", you can create a frequency table and see if those are truly the only two values present. In table 2.3, the values "Godo" and "Bade" point out something went wrong in at least 16 cases.

Table 2.3 Detecting outliers on simple variables with a frequency table

Value	Count
Good	1598647
Bad	1354468
Godo	15
Bade	1

- Here are some common types of data entry errors in data science:
    - **Typos and Misspellings:** Human errors during manual data entry can lead to typos and misspellings. These errors can introduce inconsistencies and make it difficult to accurately analyze and interpret the data.
    - **Transposition Errors:** Transposition errors occur when digits or characters are unintentionally switched, leading to incorrect values. For example, entering "125" instead of "152" or "abc" instead of "cab".
    - **Redundant whitespaces:** Whitespaces tend to be hard to detect but cause errors like other redundant characters would. Ex: "FR " instead of "FR"
    - **Fixing Capital Letter Mismatches :** Capital letter mismatches are common.
      Most programming languages make a distinction between "Brazil" and "brazil". In this case you can solve the problem by applying a function that returns
      both strings in lowercase, such as .lower() in Python. "Brazil".lower() ==
    - "brazil".lower() should result in true.
    - **IMPOSSIBLE VALUES AND SANITY CHECKS:** Sanity checks are another valuable type of data check. Here you check the value against physically or theoretically impossible values such as people taller than 3 meters or someone with an age of 299 years.
      Sanity checks can be directly expressed with rules:
      check = 0 <= age <= 120
    - **Outliers**: An outlier is an observation that seems to be distant from other observations or, more specifically, one observation that follows a different logic or generative process than the other observations. The easiest way to find outliers is to use a plot or a table with the minimum and maximum values.





- **Data Formatting Errors:** Inconsistent data formatting can lead to errors, especially when dealing with dates, currency, or numeric values. Different formats or units used for the same data can cause problems during analysis.
- **Missing Data:** Incomplete data entry or missing values can introduce biases and affect the statistical validity of analyses. Missing data points need to be handled appropriately to avoid skewing results or introducing inaccuracies.

Table 2.4 An overview of techniques to handle missing data

Technique	Advantage	Disadvantage
Omit the values	Easy to perform	You lose the information from an observation
Set value to `null`	Easy to perform	Not every modeling technique and/or implementation can handle `null` values
Impute a static value such as 0 or the mean	Easy to perform You don't lose information from the other variables in the observation	Can lead to false estimations from a model
Impute a value from an estimated or theoretical distribution	Does not disturb the model as much	Harder to execute You make data assumptions
Modeling the value (nondependent)	Does not disturb the model too much	Can lead to too much confidence in the model Can artificially raise dependence among the variables Harder to execute You make data assumptions

- **DEVIATIONS FROM A CODE BOOK:** Deviating from the codebook refers to situations where the actual data collected or used in the analysis differ from the specifications outlined in the codebook
- **DIFFERENT UNITS OF MEASUREMENT:** When integrating two data sets, you have to pay attention to their respective units of measurement. An example of this would be when you study the prices of gasoline in the world. To do this you gather data from

different data providers. Data sets can contain prices per gallon and others can contain prices per litre.

- **DIFFERENT LEVELS OF AGGREGATION:** Having different levels of aggregation is similar to having different types of measurement. An example of this would be a data set containing data per week versus one containing data per work week. This type of error is generally easy to detect, and summarizing (or the inverse, expanding) the data sets will fix it.
- **Incorrect Data Types**: Data entry errors can result in assigning incorrect data types to variables. For instance, storing a numeric value as a string or vice versa. Incorrect data types can lead to computation errors or unexpected results during analysis.
- **Duplication and Duplicates:** Entering duplicate data or duplicate records can distort analysis results, skew statistical measures, and create redundancies in the dataset.
- **Outliers and Aberrant Values:** Errors during data entry can introduce outliers or aberrant values that are significantly different from the rest of the dataset. These outliers can impact statistical analysis and lead to misleading insights if not properly identified and addressed.

2.4.2 Correct errors as early as possible

- A good practice is to handle data errors as early as possible in the data collection chain and to fix as little as possible inside your program while fixing the origin of the problem.
- Retrieving data is a difficult task, and organizations spend millions of dollars on it in the hope of making better decisions.
- The data collection process is errorprone, and in a big organization it involves many steps and teams.
- Data should be cleansed when acquired for many reasons:

    - Not everyone spots the data abnormality.
    - Decision-makers may make costly mistakes on information based on incorrect data from applications that fail to correct for the faulty data.
    - If errors are not corrected early on in the process, the cleansing will have to be done for every project that uses that data.
    - Data errors may point to a business process that isn't working as designed.
    - For instance, both authors worked at a retailer in the past, and they designed a couponing system to attract more people and make a higher profit.
    - During a data science project, we discovered clients who abused the couponing system and earned money while purchasing groceries.

- Fixing the data as soon as it's captured is nice in a perfect world. Sadly, a data scientist doesn't always have a say in the data collection and simply telling the IT department to fix certain things may not make it so. If you can't correct the data at the source, you'll need to handle it inside your code. Data manipulation doesn't end with correcting mistakes; you still need to combine your incoming data.

2.4.3 Combining data from different data sources

- Your data comes from several different places, and in this sub step we focus on integrating these different sources.
- Data varies in size, type, and structure, ranging from databases and Excel files to text documents.

- **There are two different ways of combining Data:**
  - **Joining the tables**
  - **Appending or stacking tables**

- When you combine data, you have the option to create a new physical table or a virtual table by creating a view. The advantage of a view is that it doesn't consume more disk space.

Joining Tables

- Joining tables allows you to combine the information of one observation found in one table with the information that you find in another table.
- The focus is on enriching a single observation. Let's say that the first table contains information about the purchases of a customer and the other table contains information about the region where your customer lives.
- Joining the tables allows you to combine the information so that you can use it for your model, as shown in figure 2.7.
- To join tables, you use variables that represent the same object in both tables, such as a date, a country name, or a Social Security number. These common fields are known as keys. When these keys also uniquely define the records in the table they are called primary keys.
- One table may have buying behavior and the other table may have demographic information on a person.
- In figure 2.7 both tables contain the client name, and this makes it easy to enrich the client expenditures with the region of the client. The number of resulting rows in the output table depends on the exact join type that you use

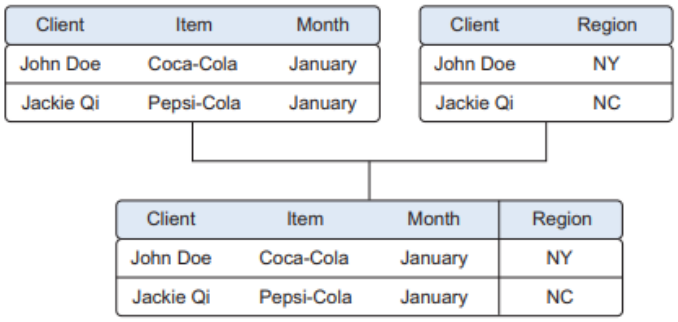


Figure 2.7 Joining two tables on the Item and Region keys

APPENDING TABLES

- Appending or stacking tables is effectively adding observations from one table to another table.
- Figure 2.8 shows an example of appending tables.
- One table contains the observations from the month January and the second table contains observations from the month February.
- The result of appending these tables is a larger one with the observations from January as well as February.
- The equivalent operation in set theory would be the union, and this is also the command in SQL, the common language of relational databases. Other set operators are also used in data science, such as set difference and intersection.

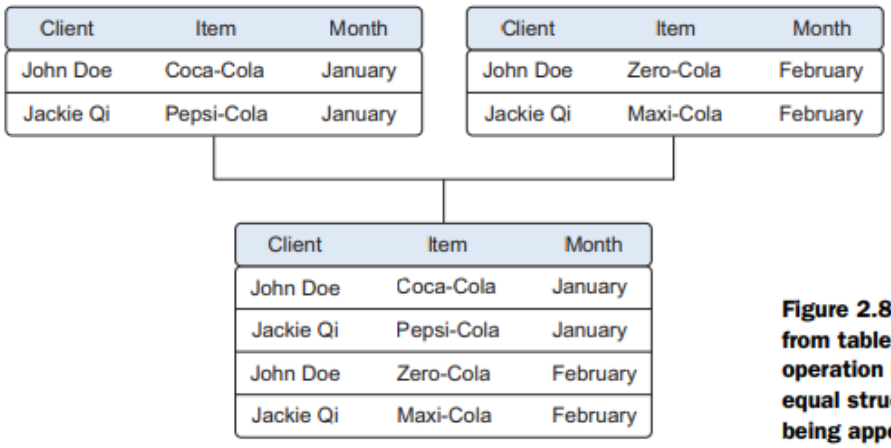


Figure 2.8 Appending data from tables is a common operation but requires an equal structure in the tables being appended.

USING VIEWS TO SIMULATE DATA JOINS AND APPENDS

- To avoid duplication of data, you virtually combine data with views.
- Figure 2.9 shows how the sales data from the different months is combined virtually into a yearly sales table instead of duplicating the data. Views do come with a drawback.
- While a table join is only performed once, the join that creates the view is recreated every time it's queried, using more processing power than a pre-calculated table would have.

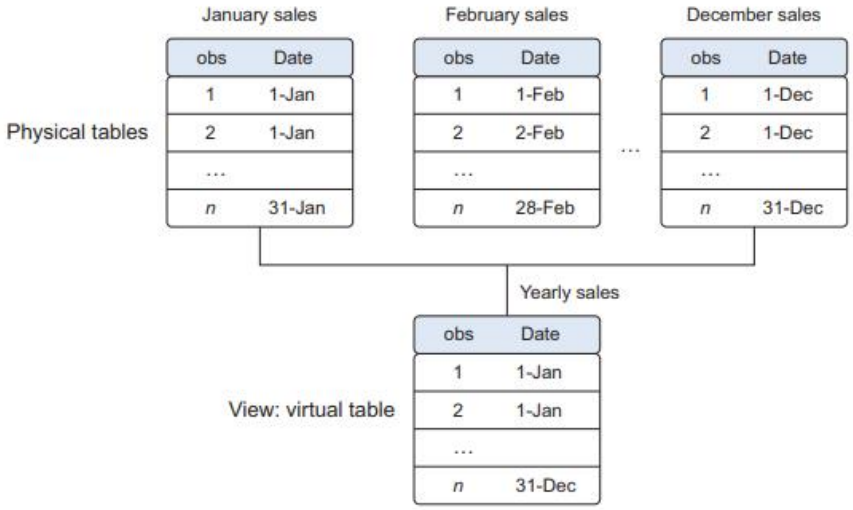


Figure 2.9 A view helps you combine data without replication.

2.4.4 Transforming data

Data Transformation

In data science, data transformation refers to the process of converting or manipulating raw data into a suitable format for analysis or modeling. It involves applying various operations or techniques to modify the structure, content, or representation of the data.

Reducing Number of variables

- Reducing the number of variables in data transformation is often referred to as dimensionality reduction.

- It involves reducing the number of features or variables in a dataset while preserving the important information and minimizing the loss of useful patterns or relationships.
- Sometimes you have too many variables and need to reduce the number because they don't add new information to the model. Having too many variables in your model makes the model difficult to handle, and certain techniques don't perform well when you overload them with too many input variables.
- **Dimensionality reduction can be beneficial in various scenarios, such as:**
    - **Curse of Dimensionality –** High Dimensional Datasets
    - **Overfitting -** When the number of variables is large compared to the number of observations, it can lead to overfitting
- **There are two main approaches to dimensionality reduction:**
    - **Feature Selection** - Feature selection methods aim to select a subset of the original features based on their relevance to the target variable or their ability to represent the underlying data.
    - **Feature Extraction -** Feature extraction methods transform the original features into a lower-dimensional space by creating new features that capture the most important information in the data. Principal Component Analysis (PCA) is a widely used technique that identifies linear combinations of features (principal components) that explain the maximum variance in the data.

Turning variables into dummies

- Turning variables into dummies, also known as one-hot encoding or creating dummy variables, is a common technique used in data transformation. It is primarily used to represent categorical variables numerically, allowing them to be used as inputs in various statistical analyses or machine learning models.
- Suppose you have a dataset that contains information about fruits, including their names and colors. The color variable is categorical and can take three values: "Red," "Green," and "Blue." Here's a sample of the dataset:

Fruit	Color
Apple	Red
Banana	Yellow
Grape	Green
Orange	Orange

- To transform the categorical variable "Color" into dummy variables, you would create binary variables for each unique category. In this case, you would create three dummy variables: "Red," "Green," and "Blue."
- Each dummy variable will indicate whether a fruit belongs to that category or not.
- After applying dummy variable transformation, the dataset would look like this:

Fruit	Color_Red	Color_Green	Color_Blue
Apple	1	0	0
Banana	0	0	0
Grape	0	1	0
Orange	0	0	0

Now, let's break down the transformation for each fruit:

1. Apple:
   - The "Color" variable for Apple is "Red."
   - The "Color_Red" dummy variable is set to 1, indicating that the fruit is red.
   - The "Color_Green" and "Color_Blue" dummy variables are set to 0 since the fruit is not

2.5 Exploratory data analysis (EDA)

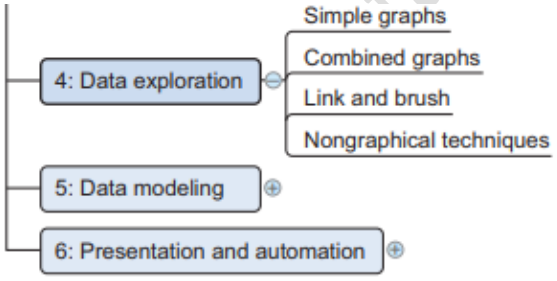


Figure 2.14 Step 4: Data exploration

- EDA stands for Exploratory Data Analysis. It refers to the process of analyzing and summarizing data sets to gain insights, discover patterns, detect anomalies, and formulate hypotheses. EDA is often the first step in the data analysis pipeline, helping data scientists and analysts understand the underlying structure and characteristics of the data.

- During exploratory data analysis you take a deep dive into the data (see figure 2.14)
- Information becomes much easier to grasp when shown in a picture, therefore you mainly use graphical techniques to gain an understanding of your data and the interactions between variables.

  The primary objectives of EDA are:

  Data Familiarity: EDA helps in becoming familiar with the data by examining its size, shape, and general properties. It involves understanding the variables, their data types, and the overall structure of the dataset.

  Data Quality Assessment: EDA allows for the identification and assessment of data quality issues such as missing values, outliers, inconsistencies, or data entry errors. By visualizing and summarizing the data, EDA helps in assessing data completeness and identifying potential data quality problems.

Data Distribution and Summary Statistics: EDA involves calculating summary statistics (such as mean, median, mode, variance) and visualizing data distributions. It helps in understanding the central tendency, spread, and shape of the data, enabling data scientists to make informed decisions and draw preliminary conclusions.

Pattern Discovery: EDA techniques, including data visualization, help in identifying patterns, trends, or relationships within the data. By exploring variables and their interactions, analysts can uncover valuable insights and formulate hypotheses for further investigation.

Outlier Detection: EDA facilitates the identification of outliers, which are data points that deviate significantly from the majority of the data. Outliers can provide important information or indicate data quality issues, and their detection is crucial for accurate analysis and modeling.

Variable Relationships: EDA explores the relationships between variables, allowing for the identification of associations or dependencies. It helps in understanding how variables interact and influence each other, which can guide feature selection, model building, or further analysis.

EDA techniques often involve visualizations such as histograms, box plots, scatter plots, line plots, and correlation matrices. These visual representations enable data scientists to grasp complex patterns or trends in the data more easily.

Overall, EDA plays a vital role in understanding the data, gaining insights, and formulating hypotheses.

It helps in guiding subsequent analysis steps, model selection, and feature engineering, contributing to effective decision-making and problem-solving in data science projects.

- The visualization techniques you use in this phase range from simple line graphs or histograms, as shown in figure 2.15

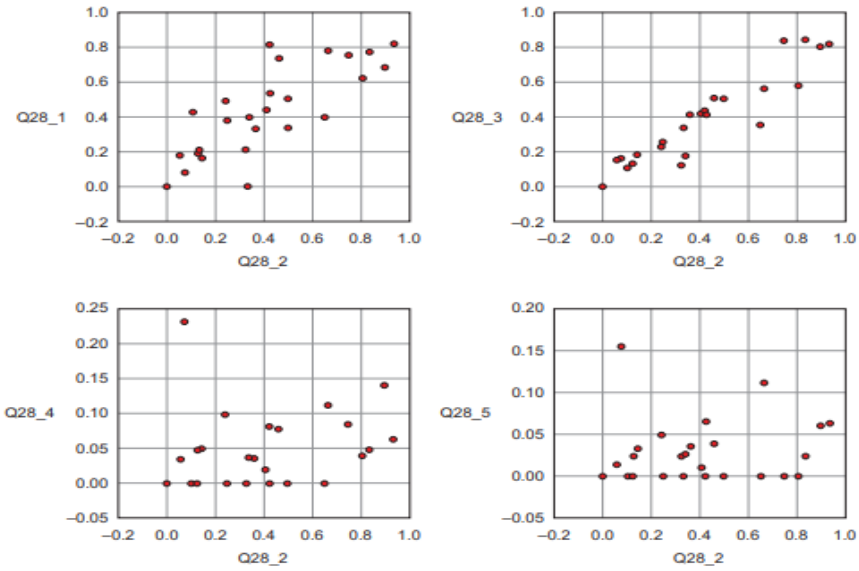


Figure 2.16 Drawing multiple plots together can help you understand the structure of your data over multiple variables.

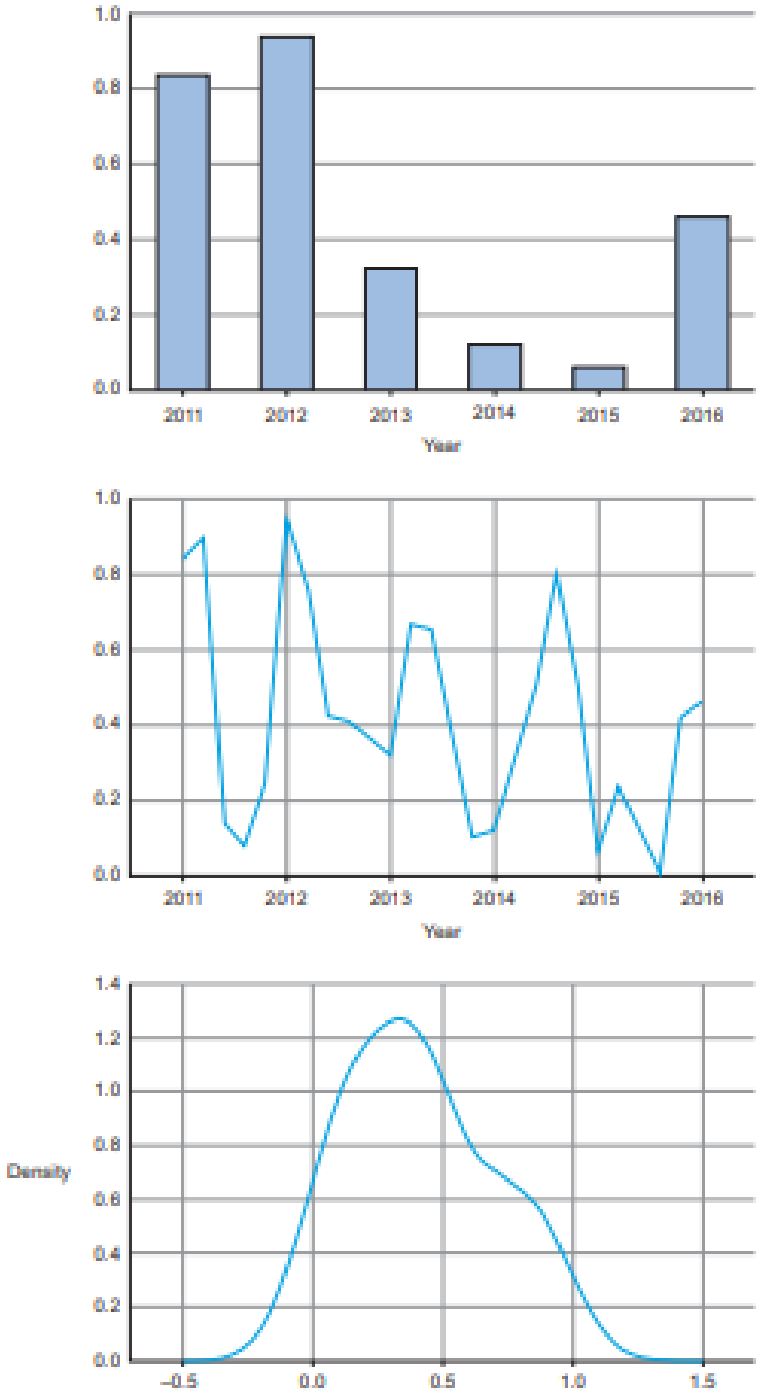


Figure 2.15 From top to bottom, a bar chart, a line plot, and a distribution are some of the graphs used in exploratory analysis.

2.6: Building Models

What is model building?

Building models in data science refers to the process of creating mathematical or statistical representations that can predict outcomes, explain relationships, or uncover patterns in a given dataset. It involves selecting an appropriate algorithm, training the model using available data, and evaluating its performance to make accurate predictions or gain insights from new data.

- Building a model is an iterative process.
- Most models consist of the following main steps:
    1. **Selection of a modelling technique and variables to enter in the model**
    2. **Execution of the model**
    3. **Diagnosis and model comparison**

2.6.1 Model and variable selection

- You'll need to select the variables you want to include in your model and a modelling technique.
- Your findings from the exploratory analysis should already give a fair idea of what variables will help you construct a good model.
- Many modelling techniques are available, and choosing the right model for a problem requires judgment on your part.
- You'll need to consider model performance and whether your project meets all the requirements to use your model, as well as other factors:
    - Must the model be moved to a production environment and, if so, would it be easy to implement?
    - How difficult is the maintenance on the model: how long will it remain relevant if left untouched?
    - Does the model need to be easy to explain?

2.6.2 Model execution

- Once you've chosen a model you'll need to implement it in code.
- Luckily, most programming languages, such as Python, already have libraries such as StatsModels or Scikit-learn. These packages use several of the most popular techniques like regression, classification and much more machine learning algorithms
- **Example of Executing a linear prediction model on semi random data:**

Listing 2.1 Executing a linear prediction model on semi-random data

```
import statsmodels.api as sm
import numpy as np
predictors = np.random.random(1000).reshape(500,2)
target = predictors.dot(np.array([0.4, 0.6])) + np.random.random(500)
lmRegModel = sm.OLS(target,predictors)
result = lmRegModel.fit()
result.summary()
```

Imports required Python modules.

Creates random data for predictors (x-values) and semi-random data for the target (y-values) of the model. We use predictors as input to create the target so we infer a correlation here.

Fits linear regression on data.

Shows model fit statistics.

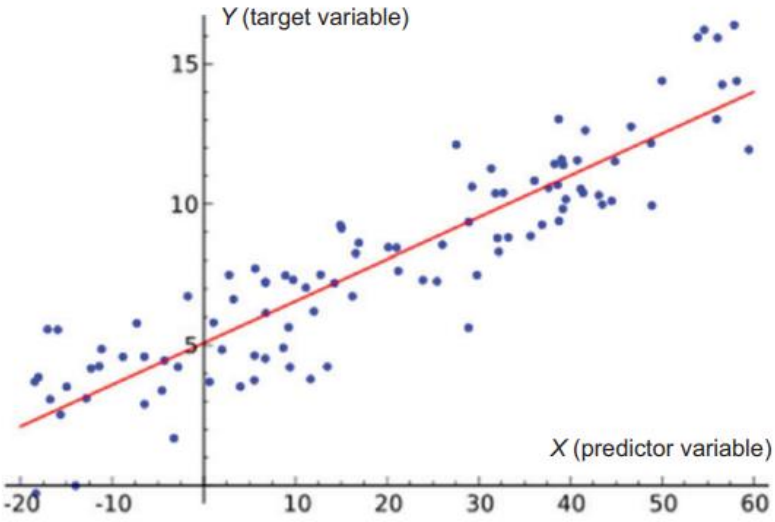


Figure 2.22 Linear regression tries to fit a line while minimizing the distance to each point

We, however, created the target variable, based on the predictor by adding a bit of randomness. It shouldn't come as a surprise that this gives us a well-fitting model. The `results.summary()` outputs the table in figure 2.23. Mind you, the exact outcome depends on the random variables you got.

Dep. Variable:	y	**R-squared:**	0.893
Model:	OLS	**Adj. R-squared:**	0.893
Method:	Least Squares	**F-statistic:**	2088.
Date:	Fri, 30 Oct 2015	**Prob (F-statistic):**	7.13e-243
Time:	12:44:31	**Log-Likelihood:**	-176.74
No. Observations:	500	**AIC:**	357.5
Df Residuals:	498	**BIC:**	365.9
Df Model:	2		
Covariance Type:	nonrobust		

Model fit: higher is better but too high is suspicious.

p-value to show whether a predictor variable has a significant influence on the target. Lower is better and <0.05 is often considered "significant."

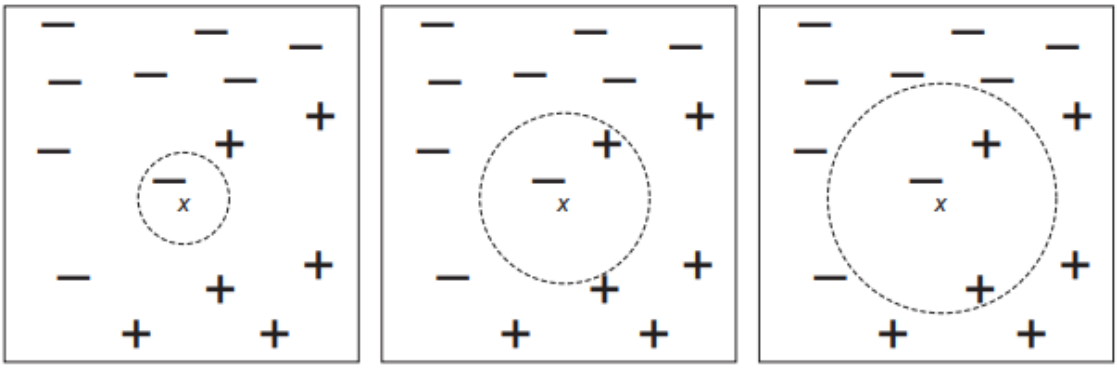
	coef	std err	t	P>|t|	[95.0% Conf. Int.]
x1	0.7658	0.040	19.130	0.000	0.687 0.844
x2	1.1252	0.039	28.603	0.000	1.048 1.202

Omnibus:	34.269	**Durbin-Watson:**	1.943
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	13.480
Skew:	-0.125	**Prob(JB):**	0.00118
Kurtosis:	2.235	**Cond. No.**	2.51

Linear equation coefficients.
$y = 0.7658x1 + 1.1252x2$.

Figure 2.23 Linear regression model information output

- Linear regression works if you want to predict a value, but what if you want to classify something? Then you go to classification models, the best known among them being **k-nearest neighbors.**
- As shown in figure 2.24, k-nearest neighbors looks at labeled points nearby an unlabeled point and, based on this, makes a prediction of what the label should be.



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

Figure 2.24 K-nearest neighbor techniques look at the k-nearest point to make a prediction.

Listing 2.2 Executing k-nearest neighbor classification on semi-random data

```
from sklearn import neighbors
predictors = np.random.random(1000).reshape(500,2)
target = np.around(predictors.dot(np.array([0.4, 0.6])) +
         np.random.random(500))
clf = neighbors.KNeighborsClassifier(n_neighbors=10)
knn = clf.fit(predictors,target)
knn.score(predictors, target)
```

Imports modules.

Creates random predictor data and semi-random target data based on predictor data.

Fits 10-nearest neighbors model.

Gets model fit score: what percent of the classification was correct?

2.6.3 Model diagnostics and model comparison

- You'll be building multiple models from which you then choose the best one based on multiple criteria.
- Working with a holdout sample helps you pick the best-performing model.
- A holdout sample is a part of the data you leave out of the model building so it can be used to evaluate the model afterward.
- The principle here is simple: the model should work on unseen data. You use only a fraction of your data to estimate the model and the other part, the holdout sample, is kept out of the equation. The model is then unleashed on the unseen data and error measures are calculated to evaluate it.
- Multiple error measures are available, and in figure 2.26 we show the general idea on comparing models. The error measure used in the example is the mean square error.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

Figure 2.26 Formula for mean square error

- Mean square error is a simple measure: check for every prediction how far it was from the truth, square this error, and add up the error of every prediction.
- Figure 2.27 compares the performance of two models to predict the order size from the price. The first model is size = 3 * price and the second model is size = 10.

	n	Size	Price	Predicted model 1	Predicted model 2	Error model 1	Error model 2
	1	10	3				
	2	15	5				
	3	18	6				
	4	14	5				
	…	…					
80% train	800	9	3				
	801	12	4	12	10	0	2
	802	13	4	12	10	1	3
	…						
	999	21	7	21	10	0	11
20% test	1000	10	4	12	10	–2	0
				Total		5861	110225

Figure 2.27 A holdout sample helps you compare models and ensures that you can generalize results to data that the model has not yet seen.

Final Conclusion:

- To estimate the models, we use 800 randomly chosen observations out of 1,000 (or 80%), without showing the other 20% of data to the model.
- Once the model is trained, we predict the values for the other 20% of the variables based on those for which we already know the true value, and calculate the model error with an error measure. Then we choose the model with the lowest error.
- **In this example we chose model 1 because it has the lowest total error.**

2.7 Presenting findings and building applications on top of them

- After you've successfully analyzed the data and built a well-performing model, you're ready to present your findings to the world.
- **Presenting your results to the stakeholders and industrializing your analysis process for repetitive reuse and integration with other tools**
- Sometimes people get so excited about your work that you'll need to repeat it over and over again because they value the predictions of your models or the insights that you produced. For this reason, you need to automate your models.
- Sometimes it's sufficient that you implement only the model scoring; other times you might build an application that automatically updates reports, Excel spreadsheets, or PowerPoint

presentations. The last stage of the data science process is where your soft skills will be most useful, and yes, they're extremely important.

Introduction to Data Science – SE26B

Syllabus

UNIT-3

Algorithms - Machine learning algorithms – Modeling process – Types – Supervised – Unsupervised - Semi-supervised

3.1 Algorithms

Machine learning algorithms are computational models or techniques that enable machines to learn from data and make predictions or decisions without being explicitly programmed. These algorithms form the foundation of various machine learning tasks and are designed to extract patterns, relationships, and insights from data

Examples:

- Linear Regression
- Logistic Regression
- Decision Tress
- Random Forest
- Support Vector Machine(SVM)
- Navies Bayes
- KNN
- Neural Networks
- Clustering Algorithms and much more

3.2 What is Machine Learning?

"Machine learning is a field of study that gives computers the ability to learn without being explicitly programmed."

- To achieve machine learning, experts develop general-purpose algorithms that can be used on large classes of learning problems.
- When you want to solve a specific task you only need to feed the algorithm more specific data. In a way, you're programming by example.
- In most cases a computer will use data as its source of information and compare its output to a desired output and then correct for it.
- The more data or "experience" the computer gets, the better it becomes at its designated job, like a human does.

Define Machine Learning Definition as a process

"Machine learning is the process by which a computer can work more accurately as it collects and learns from the data it is given."

- For example, as a user writes more text messages on a phone, the phone learns more about the messages' common vocabulary and can predict (autocomplete) their words faster and more accurately

Applications for machine learning in data science

- Regression and classification are of primary importance to a data scientist. To achieve these goals, one of the main tools a data scientist uses is machine learning. The uses for regression and automatic classification are wide ranging, such as the following:
  - Finding oil fields, gold mines, or archeological sites based on existing sites (classification and regression)
  - Finding place names or persons in text (classification)
  - Identifying people based on pictures or voice recordings (classification)
  - Recognizing birds based on their whistle (classification)
  - Identifying profitable customers (regression and classification)
  - Proactively identifying car parts that are likely to fail (regression)
  - Identifying tumors and diseases (classification)
  - Predicting the amount of money a person will spend on product X (regression)
  - Predicting the number of eruptions of a volcano in a period (regression)
  - Predicting your company's yearly revenue (regression)
  - Predicting which team will win the Champions League in soccer (classification)

Root cause Analysis

- Occasionally data scientists build a model (an abstraction of reality) that provides insight to the underlying processes of a phenomenon. When the goal of a model isn't prediction but interpretation, it's called root cause analysis.
  Here are a few examples:
  - Understanding and optimizing a business process, such as determining which products add value to a product line
  - Discovering what causes diabetes
  - Determining the causes of traffic jams

Python tools used in machine learning

Python has a numerous number of packages that can be used in a machine learning setting. The Python machine learning ecosystem can be divided into three main types of packages, as shown in figure 3.2.

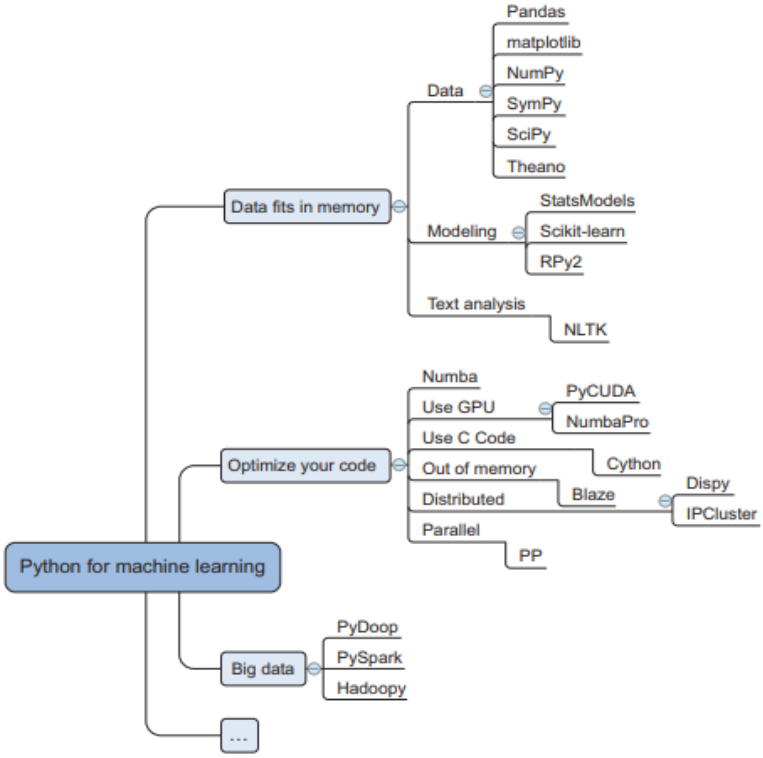


Figure 3.2 Overview of Python packages used during the machine-learning phase

PACKAGES FOR WORKING WITH DATA IN MEMORY

- **SciPy** is a library that integrates fundamental packages often used in scientific computing such as NumPy, matplotlib, Pandas, and SymPy.
- **NumPy** gives you access to powerful array functions and linear algebra functions.
- **Matplotlib** is a popular 2D plotting package with some 3D functionality.
- Pandas is a high-performance, but easy-to-use, data-wrangling package. It introduces dataframes to Python, a type of in-memory data table. It's a concept that should sound familiar to regular users of R.
- **SymPy** is a package used for symbolic mathematics and computer algebra.
- **StatsModels** is a package for statistical methods and algorithms.
- **Scikit-learn** is a library filled with machine learning algorithms.
- **RPy2** allows you to call R functions from within Python. R is a popular open source statistics program.
- **NLTK (Natural Language Toolkit)** is a Python toolkit with a focus on text analytics

OPTIMIZING OPERATIONS

Once your application moves into production, the libraries listed here can help you deliver the speed you need. Sometimes this involves connecting to big data infrastructures such as Hadoop and Spark

- **Numba and NumbaPro**—These use just-in-time compilation to speed up applications written directly in Python and a few annotations.
- **PyCUDA**—This allows you to write code that will be executed on the GPU instead of your CPU and is therefore ideal for calculation-heavy applications
- **Cython, or C for Python**—This brings the C programming language to Python. C is a lower-level language, so the code is closer to what the computer eventually uses (bytecode). The closer code is to bits and bytes, the faster it executes.
- **Blaze**—Blaze gives you data structures that can be bigger than your computer's main memory, enabling you to work with large data sets
- **Dispy and IPCluster**—These packages allow you to write code that can be distributed over a cluster of computers.
- **PP**—Python is executed as a single process by default. With the help of PP you can parallelize computations on a single machine or over clusters.
- **Pydoop and Hadoopy**—These connect Python to Hadoop, a common big data framework.
- **PySpark**—This connects Python and Spark, an in-memory big data framework.

3.3 Modelling process

The modelling phase consists of four steps:

1. **Feature engineering and model selection**
2. **Training the model**
3. **Model validation and selection**
4. **Applying the trained model to unseen data**

Before you find a good model, you'll probably iterate among the first three steps.

Ensemble Learning

It's possible to chain or combine multiple techniques. When you chain multiple models, the output of the first model becomes an input for the second model. When you combine multiple models, you train them independently and combine their results. This last technique is also known as **ensemble learning**

A model consists of constructs of information called features or predictors and a target or response variable. Your model's goal is to predict the target variable

Feature Engineering and Model Selection

With engineering features, you must come up with and create possible predictors for the model. This is one of the most important steps in the process because a model recombines these features to achieve its predictions. Often you may need to consult an expert or the appropriate literature to come up with meaningful features.

Selecting the right model is crucial for obtaining accurate and meaningful predictions. The choice of model depends on various factors, including the type of problem, size and nature of the dataset, and the desired outcome.

Here are some considerations for model selection:

Type of problem: Determine whether it is a classification, regression, clustering, or other types of problems.

Dataset size: Some models work better with large datasets, while others may perform well with smaller datasets.

Dataset characteristics: Consider the data's structure (tabular, text, images, etc.), presence of noise or outliers, and the relationships between features.

Model assumptions: Different models have different assumptions about the data, and it's important to choose a model that aligns with those assumptions.

Complexity vs. interpretability: Some models, like deep neural networks, can capture complex patterns but may lack interpretability, while simpler models like linear regression are more interpretable but may have limitations in capturing complex relationships.

Performance metrics: Consider the evaluation metrics appropriate for the problem, such as accuracy, precision, recall, or mean squared error, and choose a model that optimizes those metrics.

It's important to note that model selection and feature engineering often go hand in hand. Engineers and data scientists often iterate between these steps, trying different models and engineering features until they find the combination that yields the best performance and meaningful insights.

Training your model

- With the right predictors in place and a modeling technique in mind, you can progress to model training.
- In this phase you present to your model data from which it can learn.
- The most common modeling techniques have industry-ready implementations in almost every programming language, including Python.
- These enable you to train your models by executing a few lines of code.
- For more state-of-the art data science techniques, you'll probably end up doing heavy mathematical calculations and implementing them with modern computer science techniques.
- Once a model is trained, it's time to test whether it can be extrapolated to reality: model validation.

Validating a model

- Data science has many modeling techniques, and the question is which one is the right one to use.
- A good model has two properties:
    - It has good predictive power
    - It generalizes well to data it hasn't seen.

- To achieve this, you define an error measure (how wrong the model is) and a validation strategy.
- Two common error measures in machine learning are
    - **classification error rate for classification problems**
    - **Mean squared error for regression problems.**

Classification error rate

The classification error rate is the percentage of observations in the test data set that your model mislabelled. lower is better.

The mean squared error

The Mean Square Error measures how big the average error of your prediction is. Squaring the average error has two consequences: you can't cancel out a wrong prediction in one direction with a faulty prediction in the other direction

Many validation strategies exist, including the following common ones:

- ***Dividing your data into a training set with X% of the observations*** and keeping the rest as a holdout data set (a data set that's never used for model creation)—This is the most common technique.
- ***K-folds cross validation***—This strategy divides the data set into k parts and uses each part one time as a test data set while using the others as a training data set. It has the advantage that you use all the data available in the data set.
- ***Leave-1 out*** —This approach is the same as k-folds but with k=1. You always leave one observation out and train on the rest of the data. This is used only on small data sets, so it's more valuable to people evaluating laboratory experiments than to big data analysts

3.3 Types of machine learning

Broadly speaking, we can divide the different approaches to machine learning by the amount of human effort that's required to coordinate them and how they use labeled data—data with a category or a real-value number assigned to it that represents the outcome of previous observations.

1. **Supervised Learning -** Supervised learning is a machine learning approach where a model is trained on labeled examples or data with known inputs and corresponding outputs. The goal of supervised learning is to learn a mapping or relationship between the input features and the desired output, allowing the model to make accurate predictions or classifications for new, unseen data.
2. **Unsupervised Learning -** Unsupervised learning is a machine learning approach where the model is trained on unlabeled data, without any specific output or target variable provided. The goal of unsupervised learning is to discover patterns, structures, or relationships within the data itself, without any predefined categories or labels.
3. **Semi Supervised Learning -** Semi-supervised learning is a machine learning approach that combines labeled and unlabeled data to train a model. In simple terms, it's like having a

teacher (labeled data) who provides some answers and a lot of ungraded practice exercises (unlabeled data) for the model to learn from.

Supervised Learning

Supervised learning is a machine learning approach where a model is trained on labeled examples or data with known inputs and corresponding outputs. The goal of supervised learning is to learn a mapping or relationship between the input features and the desired output, allowing the model to make accurate predictions or classifications for new, unseen data.

Case Study: DISCERNING DIGITS FROM IMAGES

Supervised learning can be applied to the task of discerning digits from images, which is commonly known as digit recognition or digit classification. In this context, supervised learning involves training a model to accurately classify or predict the correct digit label for a given image.

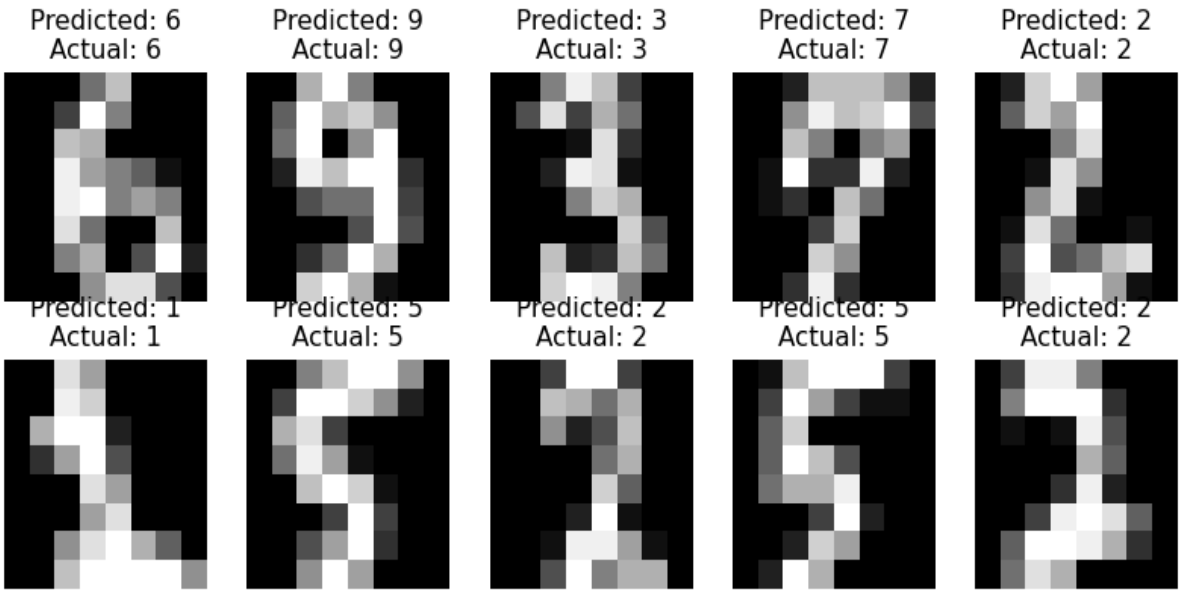
Here's how the process of supervised learning for discerning digits from images typically works:

- **Dataset**: First, you need a labeled dataset for training the model. This dataset consists of a collection of images where each image is associated with a corresponding digit label. For example, you might have a dataset of thousands of handwritten digit images, each labeled with the correct digit (e.g., 0 to 9).
- **Data Preprocessing**: The images in the dataset may need preprocessing before training the model. This could involve tasks like resizing the images to a consistent size, normalizing pixel values, and applying any necessary transformations or enhancements.
- **Model Selection:** Choose an appropriate model architecture for digit recognition. Convolutional Neural Networks (CNNs) are commonly used for image-based tasks like this. CNNs are designed to capture spatial relationships in images and have shown excellent performance in various computer vision tasks.
- **Training:** Split the labeled dataset into training and validation sets. The training set is used to train the model by presenting the images and their associated digit labels to the model. The model learns to extract relevant features and patterns from the images and make predictions. During training, the model's parameters are adjusted iteratively to minimize the difference between predicted and actual digit labels.
- **Evaluation:** After training, evaluate the model's performance using the validation set. Measure metrics such as accuracy, precision, recall, or F1 score to assess how well the model can classify digits in unseen images.
- **Testing:** Once the model is trained and evaluated, it can be used to make predictions on new, unseen images. The model takes an input image and predicts the digit label associated with it.
- **Iteration and Fine-tuning:** Depending on the results, you can iterate and fine-tune the model by adjusting hyperparameters, modifying the model architecture, or incorporating techniques like data augmentation to improve performance.

Supervised learning in the context of discerning digits from images enables the model to learn from labeled examples and develop the ability to recognize and classify digits accurately, even for unseen images.

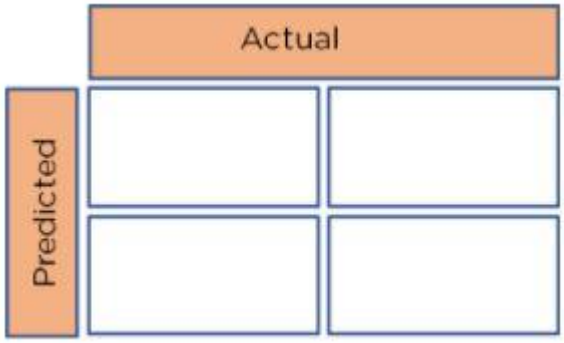
Sample Code:

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn import svm
import matplotlib.pyplot as plt

# Load the digit dataset
digits = load_digits()

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target, test_size=0.2,
random_state=42)
# Create a Support Vector Machine (SVM) classifier
classifier = svm.SVC()
# Train the classifier
classifier.fit(X_train, y_train)
# Make predictions on test data
predictions = classifier.predict(X_test)
# Display a few test images with their predicted labels
fig, axes = plt.subplots(2, 5, figsize=(10, 4))
for i, ax in enumerate(axes.flat):
    ax.imshow(X_test[i].reshape(8, 8), cmap='gray')
    ax.set_title(f"Predicted: {predictions[i]}\nActual: {y_test[i]}")
    ax.axis('off')
plt.show()
```

Confusion Matix

A confusion matrix presents a table layout of the different outcomes of the prediction and results of a classification problem and helps visualize its outcomes. It plots a table of all the predicted and actual values of a classifier.



Create a 2x2 Confusion Matrix

We can obtain four different combinations from the predicted and actual values of a classifier:

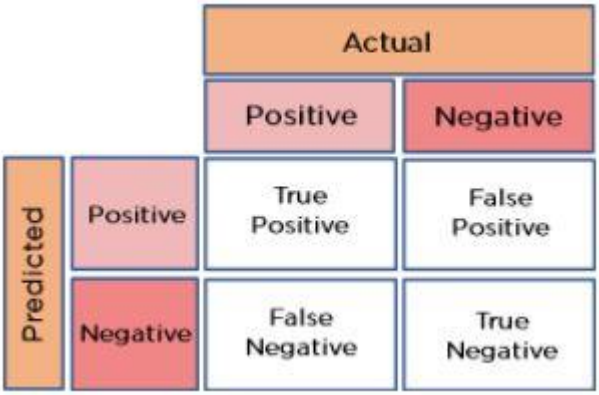


Figure 2: Confusion Matrix

True Positive: The number of times our actual positive values are equal to the predicted positive. You predicted a positive value, and it is correct.

False Positive: The number of times our model wrongly predicts negative values as positives. You predicted a negative value, and it is actually positive.

True Negative: The number of times our actual negative values are equal to predicted negative values. You predicted a negative value, and it is actually negative.

False Negative: The number of times our model wrongly predicts negative values as positives. You predicted a negative value, and it is actually positive.

Example:

n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

Unsupervised learning

Unsupervised learning is a machine learning approach where the model is trained on unlabeled data, without any specific output or target variable provided. The goal of unsupervised learning is to discover patterns, structures, or relationships within the data itself, without any predefined categories or labels.

It's generally true that most large data sets don't have labels on their data, so unless you sort through it all and give it labels, the supervised learning approach to data won't work. Instead, we must take the approach that will work with this data because

■ We can study the distribution of the data and infer truths about the data in different parts of the distribution.

■ We can study the structure and values in the data and infer new, more meaningful data and structure from it.

Many techniques exist for each of these unsupervised learning approaches.

K-Means Clustering - Example

In this example, we generate a simple dataset using make_blobs() from scikit-learn, which creates synthetic data points in the form of blobs or clusters. We generate 100 data points with 3 centers.

We create a K-means clustering model using KMeans() and specify the number of clusters as an argument (in this case, 3 clusters). The model is then fitted to the data using fit().

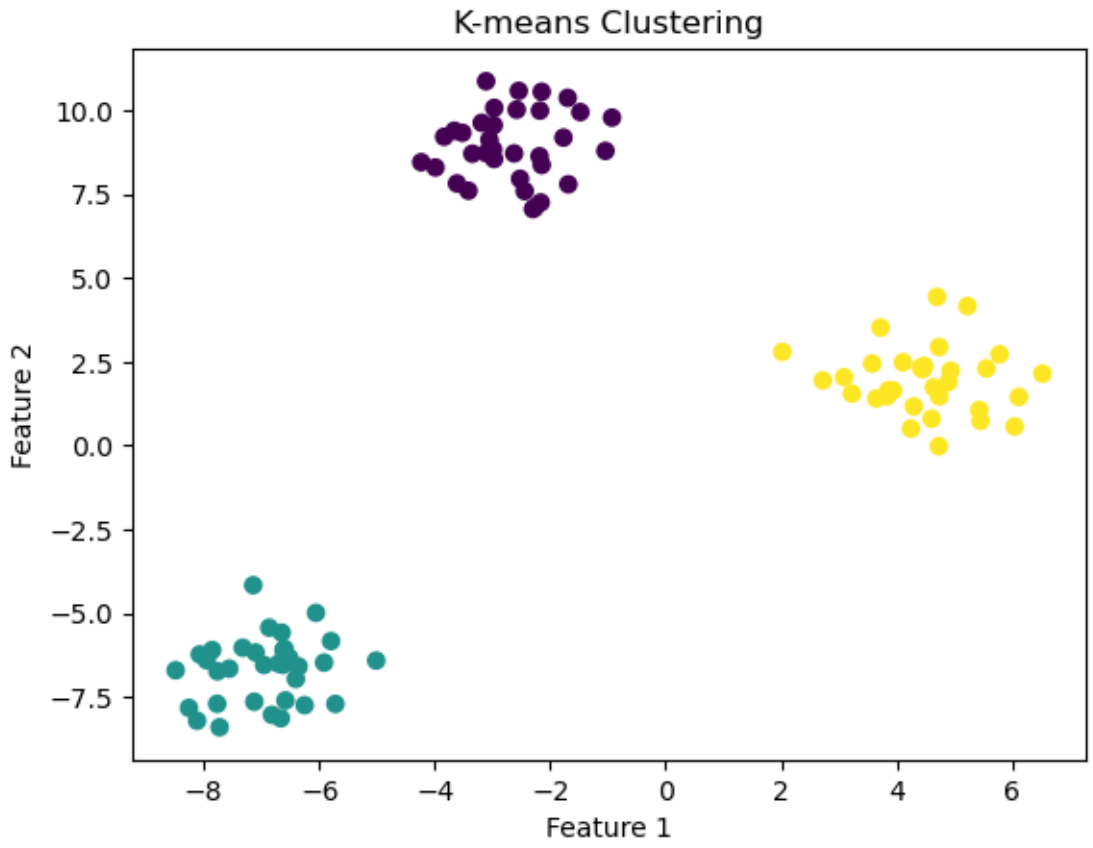
We obtain the cluster labels using labels_, which assigns each data point to a cluster.

To visualize the clustering results, we use plt.scatter() from matplotlib to plot the data points with color-coded clusters based on the labels.

In this simple example, you can observe how the K-means algorithm partitions the data points into three clusters based on their similarities, forming distinct groups in the scatter plot.

Sample Code:

```
# Import necessary libraries
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
# Generate sample data
X, _ = make_blobs(n_samples=100, centers=3, random_state=42)
# Create K-means clustering model
kmeans = KMeans(n_clusters=3)
# Fit the model to the data
kmeans.fit(X)
# Get the cluster labels
labels = kmeans.labels_
# Plot the data points with color-coded clusters
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("K-means Clustering")
plt.show()
```



Semi Supervised Learning

Semi-supervised learning is a machine learning approach that combines labeled and unlabeled data to train a model. In simple terms, it's like having a teacher (labeled data) who provides some answers and a lot of ungraded practice exercises (unlabeled data) for the model to learn from.

Our goal then is to train our predictor models with as little labeled data as possible. This is where semi-supervised learning techniques come in—hybrids of the two approaches we've already seen.

Take for example the plot in figure 3.12. In this case, the data has only two labeled observations; normally this is too few to make valid predictions.

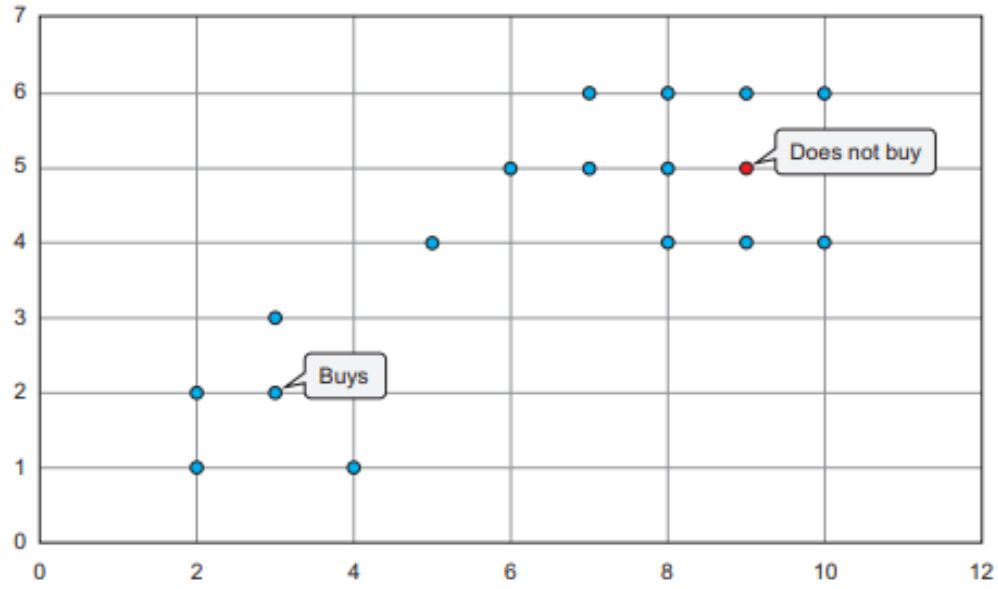


Figure 3.12 This plot has only two labeled observations—too few for supervised observations, but enough to start with an unsupervised or semi-supervised approach.

A common semi-supervised learning technique is label propagation. In this technique, you start with a labeled data set and give the same label to similar data points. This is similar to running a clustering algorithm over the data set and labeling each cluster based on the labels they contain. If we were to apply this approach to the data set in figure 3.12, we might end up with something like figure 3.13.

One special approach to semi-supervised learning worth mentioning here is active learning. In active learning the program points out the observations it wants to see labeled for its next round of learning based on some criteria you have specified.

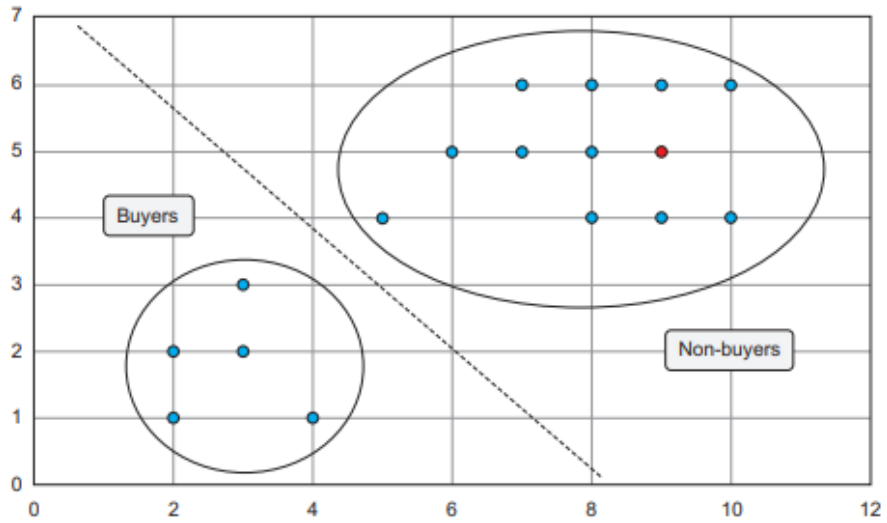


Figure 3.13 **The previous figure shows that the data has only two labeled observations, far too few for supervised learning. This figure shows how you can exploit the structure of the underlying data set to learn better classifiers than from the labeled data only. The data is split into two clusters by the clustering technique; we only have two labeled values, but if we're bold we can assume others within that cluster have that same label (buyer or non-buyer), as depicted here. This technique isn't flawless; it's better to get the actual labels if you can.**

Introduction to Data Science – SE26B

Syllabus

Unit – 4

Introduction to Hadoop – framework – Spark – replacing MapReduce– NoSQL – ACID – CAP – BASE – types

Distributing data storage and processing with frameworks

New big data technologies such as Hadoop and Spark make it much easier to work with and control a cluster of computers. Hadoop can scale up to thousands of computers, creating a cluster with petabytes of storage. This enables businesses to grasp the value of the massive amount of data available.

Hadoop: a framework for storing and processing large data sets

Apache Hadoop is a framework that simplifies working with a cluster of computers. It aims to be all of the following things and more:

Reliable—By automatically creating multiple copies of the data and redeploying processing logic in case of failure.

Fault tolerant—It detects faults and applies automatic recovery.

Scalable—Data and its processing are distributed over clusters of computers (horizontal scaling).

Portable—Installable on all kinds of hardware and operating systems

The core framework is composed of a distributed file system, a resource manager, and a system to run distributed programs. In practice it allows you to work with the distributed file system almost as easily as with the local file system of your home computer. But in the background, the data can be scattered among thousands of servers.

The Different Components of Hadoop

- ***A distributed file system (HDFS)***
- ***A method to execute programs on a massive scale (MapReduce)***
- ***A system to manage the cluster resources (YARN)***

On top of that, an ecosystem of applications arose (figure 5.2), such as the databases. Hive and HBase and frameworks for machine learning such as Mahout. Hive has a language based on the widely used SQL to interact with data stored inside the database

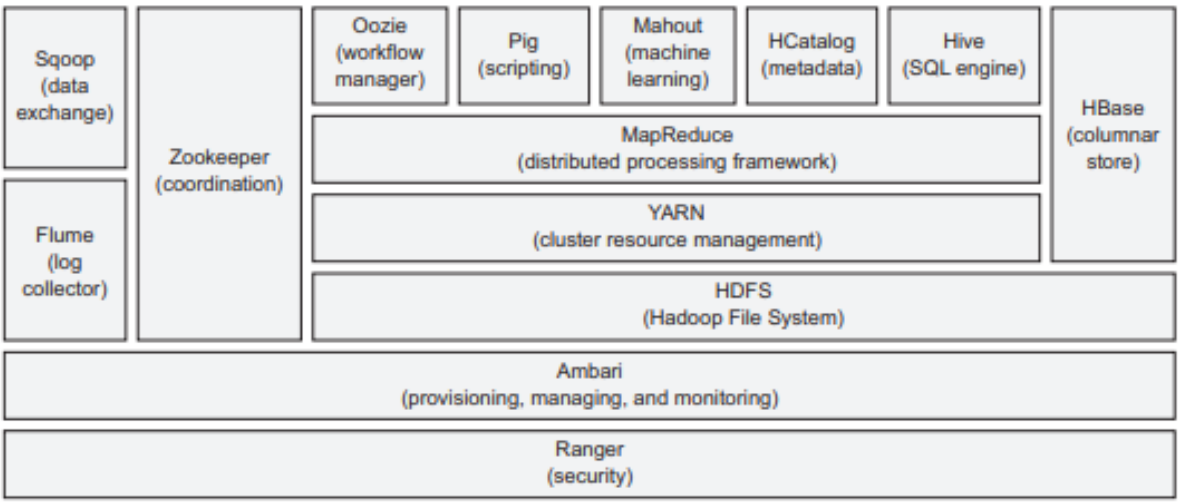


Figure 5.2 A sample from the ecosystem of applications that arose around the Hadoop Core Framework

MAPREDUCE: How Hadoop Achieves Parallelism

Hadoop uses a programming method called MapReduce to achieve parallelism. A MapReduce algorithm splits up the data, processes it in parallel, and then sorts, combines, and aggregates the results back together. However, the MapReduce algorithm isn't well suited for interactive analysis or iterative programs because it writes the data to a disk in between each computational step. This is expensive when working with large data sets.

Map Reduce Algorithms

- MapReduce is a programming model and algorithmic approach designed to process and analyze large amounts of data in a distributed computing environment. It simplifies the task of parallelizing computations across multiple machines, making it more efficient to process big data.
- In simple terms, the MapReduce algorithm consists of two main stages:
    - the map stage
    - the reduce stage.

Map Stage:

- The input data is divided into smaller chunks and distributed to multiple machines in a cluster.
- Each machine independently processes its chunk of data using a "map" function.
- The map function takes the input data, performs some computation on it, and emits intermediate key-value pairs.

Reduce Stage:

- The intermediate key-value pairs generated by the map stage are collected and shuffled based on their keys.
- The shuffled key-value pairs are then sent to different machines, where the "reduce" function is applied.

- The reduce function aggregates and processes the intermediate values associated with each key, producing the final output.

The key idea behind MapReduce is the parallelization of computations across multiple machines, allowing for efficient processing of large datasets. It provides fault tolerance and scalability by distributing the data and computation across the cluster, handling failures, and automatically balancing the workload.

MapReduce has been widely used in big data processing frameworks such as Apache Hadoop, allowing for the processing of massive amounts of data in a distributed manner. It abstracts away the complexities of parallel programming, making it easier to write scalable and efficient data processing applications.

The whole process is described in the following six steps and depicted in figure 5.4.

1 Reading the input files.
2 Passing each line to a mapper job.
3 The mapper job parses the colors (keys) out of the file and outputs a file for each color with the number of times it has been encountered (value). Or more technically said, it maps a key (the color) to a value (the number of occurrences).
4 The keys get shuffled and sorted to facilitate the aggregation.
5 The reduce phase sums the number of occurrences per color and outputs one file per key with the total number of occurrences for each color.
6 The keys are collected in an output file.

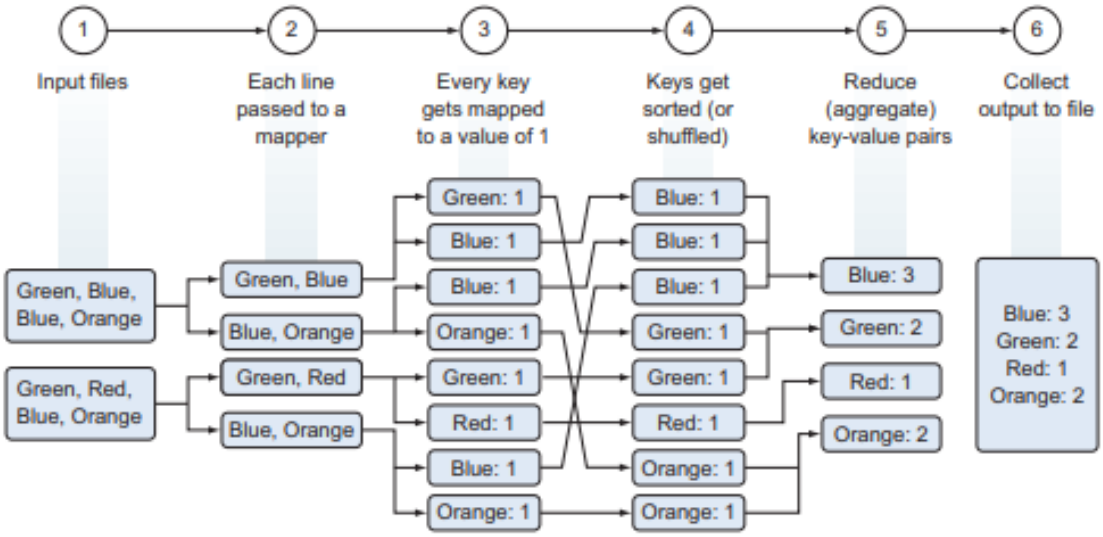


Figure 5.4 An example of a MapReduce flow for counting the colors in input texts

What is Spark?

Spark is a cluster computing framework similar to MapReduce. Spark, however, doesn't handle the storage of files on the (distributed) file system itself, nor does it handle the resource management. For this it relies on systems such as the Hadoop File System, YARN, or Apache Mesos. Hadoop and Spark are thus complementary systems. For testing and development, you can even run Spark on your local system.

How Does Spark Solve the Problems of MapReduce?

- Here are some ways in which Spark replaces or mitigates the limitations of MapReduce:
    1. In-Memory Processing: Spark controls in-memory computing, allowing it to cache intermediate data in memory and perform iterative or interactive computations much faster.
    2. DAG-based Task Execution: Spark uses a Directed Acyclic Graph (DAG) execution engine, which optimizes the execution plan based on the dependencies between tasks.
    3. Resilient Distributed Datasets (RDDs): Spark introduces RDDs, which are fault-tolerant and immutable distributed collections of data. RDDs allow for efficient distributed data processing and provide higher-level abstractions like transformations and actions, making it easier to express complex computations compared to the low-level MapReduce model.
    4. Rich Set of APIs: Spark provides APIs in multiple programming languages, including Python, Java, Scala, and R, making it more accessible to a wider range of developers. These APIs allow users to express computations using high-level abstractions like DataFrames and Datasets, which offer enhanced functionality and optimized execution.
    5. Support for Various Workloads: Spark supports not only batch processing but also interactive queries, streaming data, machine learning, and graph processing. It provides libraries like Spark SQL, Spark Streaming, MLlib, and GraphX, enabling a wide range of data processing and analytics tasks within a unified framework.

Overall, Spark addresses the limitations of MapReduce by leveraging in-memory computing, optimizing task execution with DAGs, introducing RDDs for efficient distributed data processing, providing rich APIs, and supporting diverse workloads. These improvements make Spark a more efficient, flexible, and user-friendly framework for big data processing and analytics.

Different components of the spark ecosystem

Spark core provides a NoSQL environment well suited for interactive, exploratory analysis. Spark can be run in batch and interactive mode and supports Python.

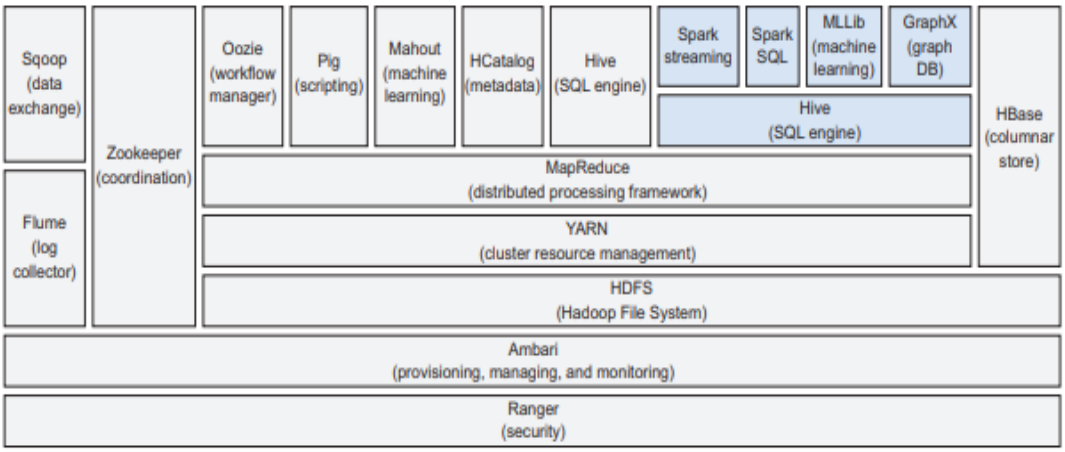


Figure 5.5 The Spark framework when used in combination with the Hadoop framework

Spark has four other large components, as listed below

1. Spark streaming is a tool for real-time analysis.
2. Spark SQL provides a SQL interface to work with Spark.
3. MLLib is a tool for machine learning inside the Spark framework.
4. GraphX is a graph database for Spark.

NoSQL – [Not Only SQL]

What is NOSQL?

A NoSQL (Not Only SQL) database is a type of database management system that provides a non-relational approach to data storage and retrieval. Unlike traditional relational databases, NoSQL databases offer flexible schema design, horizontal scalability, and high-performance data processing.

Some key characteristics and features of NoSQL databases:

1. **Flexible Schema**: NoSQL databases allow for dynamic and flexible schema design, meaning that each record or document can have a different structure. This flexibility is particularly useful in scenarios where the data schema may evolve or vary between different entities.
2. **Horizontal Scalability**: NoSQL databases are designed to scale horizontally, meaning they can distribute data across multiple servers or nodes. This allows for seamless expansion of storage and processing capacity as data volumes grow.
3. **High Performance**: NoSQL databases are optimized for handling large amounts of data and providing high-performance data processing. They often use indexing and caching techniques to enable fast data retrieval and support high throughput.
4. **Distributed Architecture**: NoSQL databases are built to operate in distributed environments, allowing data to be replicated and distributed across multiple nodes. This improves fault tolerance and ensures data availability even in the event of node failures.
5. **Variety of Data Models**: NoSQL databases support different data models, including key-value stores, document databases, columnar databases, and graph databases. Each data model is designed to handle specific types of data and use cases efficiently.

NoSQL databases are commonly used in modern applications that deal with large-scale data, real-time analytics, and high-speed data ingestion. They are suitable for use cases such as web applications, social media platforms, IoT data management, and other scenarios where the flexibility, scalability, and performance advantages of NoSQL databases are beneficial.

ACID: the core principle of relational databases

ACID is an acronym that stands for

- Atomicity
- Consistency
- Isolation
- Durability.

These are the four fundamental properties that ensure the reliability and integrity of data in a transactional relational database management system (RDBMS).

1. **Atomicity**: Atomicity guarantees that a transaction is treated as a single indivisible unit of work. Either all the operations within a transaction are successfully completed, or none of them are. If any part of the transaction fails, the entire transaction is rolled back, and the database returns to its original state before the transaction started. This ensures data integrity and prevents incomplete or inconsistent updates.
2. **Consistency**: Consistency ensures that a transaction brings the database from one valid state to another. It enforces integrity constraints, such as data validation rules, referential integrity, and other defined rules. When a transaction completes successfully, the database is left in a consistent state, preserving the relationships and dependencies between data.
3. **Isolation**: Isolation ensures that concurrent transactions do not interfere with each other. Each transaction operates as if it is executed in isolation, without being aware of other concurrent transactions. This property prevents issues such as data corruption, lost updates, and dirty reads. Isolation levels, such as Read Uncommitted, Read Committed, Repeatable Read, and Serializable, define the degree of isolation provided by the database.
4. **Durability**: Durability guarantees that once a transaction is committed, its changes are permanent and will survive any subsequent system failures, such as power outages or crashes. The committed data is stored in a durable manner, usually on disk or other persistent storage, to ensure its long-term persistence.

By adhering to the ACID properties, relational databases ensure the reliability, consistency, and integrity of data, making them suitable for applications that require strong data guarantees and transactional support.

CAP Theorem: the problem with DBs on many nodes

Once a database gets spread out over different servers, it's difficult to follow the ACID principle because of the consistency ACID promises; the CAP Theorem points out why this becomes problematic. The CAP Theorem states that a database can be any two of the following things but never all three:

- **Partition tolerant**
  The database can handle a network partition or network failure.
- **Available**
  As long as the node you're connecting to is up and running and you can connect to it, the node will respond, even if the connection between the different database nodes is lost.
- **Consistent**
  No matter which node you connect to, you'll always see the exact same data.

For a single-node database it's easy to see how it's always available and consistent:

- Available
  As long as the node is up, it's available. That's all the CAP availability promises.
- Consistent
  There's no second node, so nothing can be inconsistent.

Things get interesting once the database gets partitioned. Then you need to make a choice between availability and consistency, as shown in figure 6.2.

Figure 6.2 CAP Theorem: when partitioning your database, you need to choose between availability and consistency.

- Let's take the example of an online shop with a server in Europe and a server in the United States, with a single distribution center.
- A German named Fritz and an American named Freddy are shopping at the same time on that same online shop. They see an item and only one is still in stock: a bronze, octopus-shaped coffee table.
- Disaster strikes, and communication between the two local servers is temporarily down.
- If you were the owner of the shop, you'd have two options:
  - Availability
    You allow the servers to keep on serving customers, and you sort out everything afterward.
  - Consistency
    You put all sales on hold until communication is re-established. In the first case, Fritz and Freddy will both buy the octopus coffee table, because the last-known stock number for both nodes is "one" and both nodes are allowed to sell it, as shown in figure 6.3.

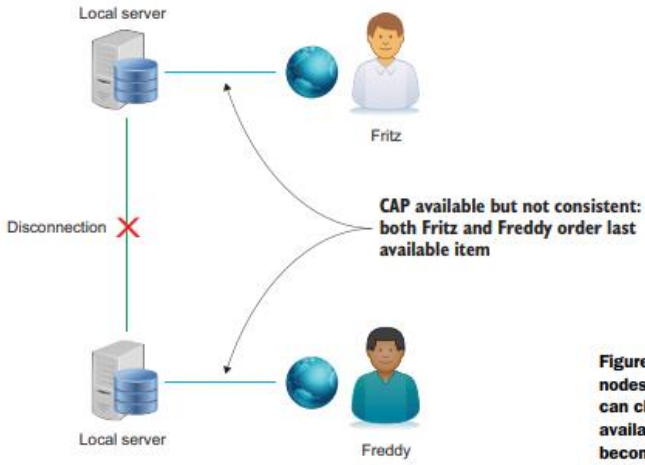


Figure 6.3 CAP Theorem: if nodes get disconnected, you can choose to remain available, but the data could become inconsistent.

If the coffee table is hard to come by, you'll have to inform either Fritz or Freddy that he won't receive his table on the promised delivery date or, even worse, he will never receive it. As a good businessperson, you might compensate one of them with a discount coupon for a later purchase, and everything might be okay after all. The second option (figure 6.4) involves putting the incoming requests on hold temporarily. This might be fair to both Fritz and Freddy if after five minutes the web shop is open for business again, but then you might lose both sales and probably many more. Web shops tend to choose availability over consistency, but it's not the optimal choice in all cases.

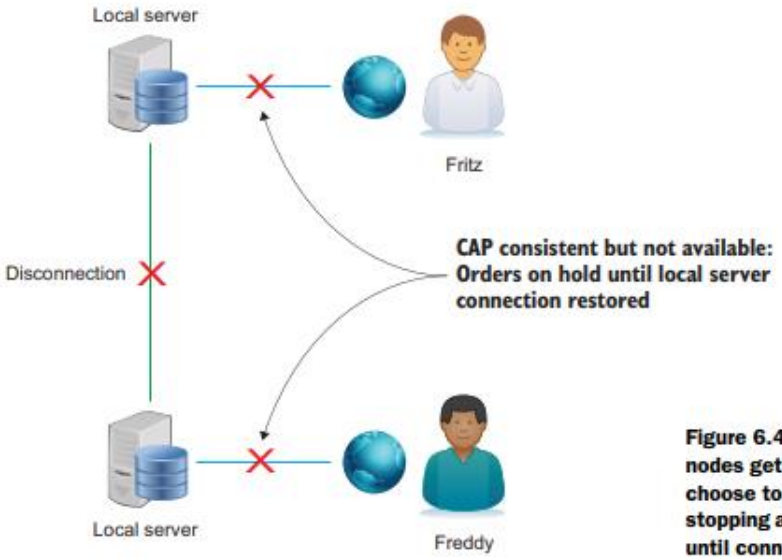


Figure 6.4 CAP Theorem: if nodes get disconnected, you can choose to remain consistent by stopping access to the databases until connections are restored

The BASE principles of NoSQL databases

- RDBMS follows the ACID principles.
- NoSQL databases that don't follow ACID, such as the document stores and key-value stores, follow BASE.
- BASE is a set of much softer database promises:
    - Basically available
    - Soft state

- Eventual consistency

1. **Basically Available**: The principle of basically available means that the system should always be available and responsive to user requests, even in the presence of partial failures or network partitions. It does not guarantee immediate consistency or access to the latest data, but it ensures that the system remains operational and can provide some level of service.
2. **Soft state**: Soft state refers to the idea that the system does not have to be in a globally consistent state at all times. Each node or replica in the system can have its own view of the data, and updates can be propagated asynchronously. As a result, there may be temporary inconsistencies or conflicts between different replicas, but the system will eventually converge to a consistent state.
3. **Eventually consistent**: The principle of eventual consistency means that the system will reach a consistent state over time, given that there are no further updates or conflicts. Updates to the data propagate asynchronously across different nodes or replicas, and conflicts are resolved eventually. This allows for high availability and scalability but sacrifices strong consistency guarantees.

NoSQL database types

There are several types of NoSQL databases, each designed to handle different data models and use cases. The common types of NoSQL databases include:

1. **Key-Value Stores:**
   - Key-value stores are the simplest form of NoSQL databases, where data is stored as a collection of key-value pairs.
   - The database allows efficient retrieval of values based on their corresponding keys.
   - Examples of key-value stores include Redis and Riak.

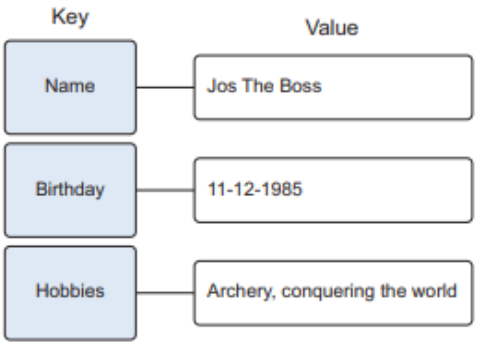


Figure 6.11 Key-value stores store everything as a key and a value.

2. **Document Databases:**
   - Document databases store and retrieve data in the form of JSON-like documents.
   - Documents can vary in structure and are typically self-contained units of data.
   - Document databases provide flexible schema design and support hierarchical and nested data structures.
   - MongoDB and CouchDB are popular examples of document databases.

Document 1

```
{
  "id": "1",
  "name": "John Smith",
  "isActive": true,
  "dob": "1964-30-08"
}
```

Document 2

```
{
  "id": "2",
  "fullName": "Sarah Jones",
  "isActive": false,
  "dob": "2002-02-18"
}
```

Document 3

```
{
  "id": "3",
  "fullName":
  {
    "first": "Adam",
    "last": "Stark"
  },
  "isActive": true,
  "dob": "2015-04-19"
}
```

3. **Columnar Databases:**
   - Columnar databases organize data into columns rather than rows, enabling efficient storage and retrieval of specific columns or attributes.
   - They are suitable for analytical workloads that involve aggregations and large-scale data processing.
   - Apache Cassandra and Apache HBase are commonly used columnar databases.

ROWID	NAME	DATE OF BIRTH	City
1	Suhail	13-11-1985	Madurai
2	Naveen	24-05-2023	Chennai
3	Monish	20-03-2010	Chennai
4	Meghna	10-02-2013	Chennai
5	Humaira	24-10-2019	Chennai

4. **Graph Databases:**
   - The last big NoSQL database type is the most complex one, geared toward storing relations between entities in an efficient manner. When the data is highly interconnected, such as for social networks, scientific paper citations, or capital asset clusters, graph databases are the answer.
   - Graph or network data has two main components:
     ■ Node—The entities themselves. In a social network this could be people.
     ■ Edge—The relationship between two entities.
     This relationship is represented by a line and has its own properties. An edge can have a direction, for example, if the arrow indicates who is whose boss.
   - Examples of graph databases include Neo4j and Amazon Neptune.

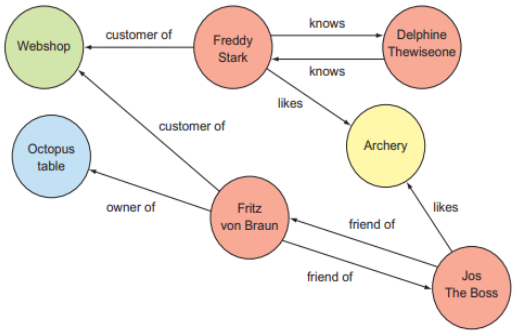


Figure 6.14 Graph data example with four entity types (person, hobby, company, and furniture) and their relations without extra edge or node information

Introduction to Data Science – SE26B

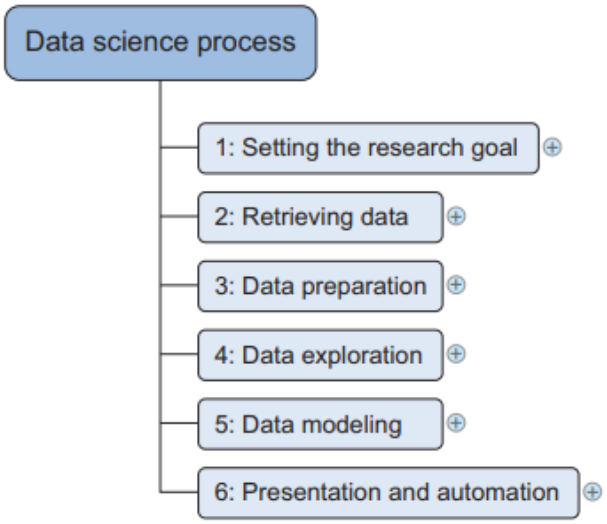
Syllabus

Unit – 5

Case Study – Prediction of Disease - Setting research goals - Data retrieval – preparation - exploration - Disease profiling - presentation and automation

Case Study: Prediction of Disease using data science process

The data science process typically consists of six steps, as you can see in the mind map in figure given below



The data science process typically involves several iterative steps to extract valuable insights and knowledge from data. While the specific steps may vary depending on the context and project requirements, here are six common steps in the data science process:

Problem Statement or Setting Research Goals:

The objective of this case study is to develop a predictive model to identify the likelihood of a patient having a certain disease based on their medical history and other relevant factors. This model can assist healthcare professionals in early detection and diagnosis, leading to timely interventions and improved patient outcomes.

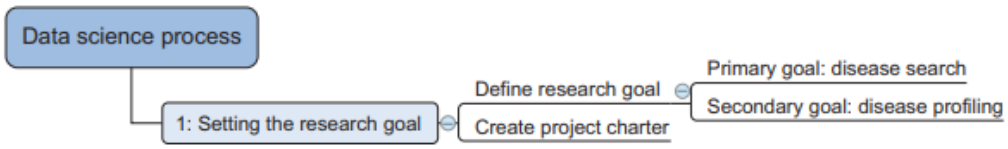


Figure 6.17 Step 1 in the data science process: setting the research goal

- Your primary goal is to set up a disease search engine that would help general practitioners in diagnosing diseases.
- Your secondary goal is to profile a disease: What keywords distinguish it from other diseases?

Retrieving Data or Data Collection:

1. Identify the target variable: Determine the specific disease to be predicted, such as diabetes, heart disease, or cancer.

2. Gather relevant data: Collect a dataset that includes patient information, medical records, lab results, demographic details, and any other factors that could be indicative of the disease.

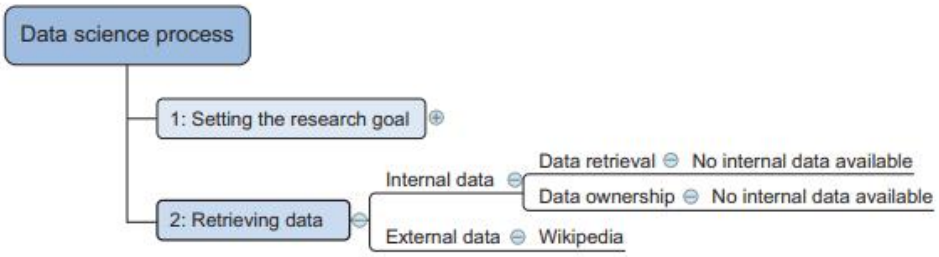


Figure 6.18 Data science process step 2: data retrieval. In this case there's no internal data; all data will be fetched from Wikipedia.

As shown in figure 6.18 you have two possible sources:

■ Internal data—You have no disease information lying around. If you currently work for a pharmaceutical company or a hospital, you might be luckier.

■ External data—All you can use for this case is external data. You have several possibilities, but you'll go with Wikipedia.

When you pull the data from Wikipedia, you'll need to store it in your local Elastic search index, but before you do that you'll need to prepare the data. Once data has entered the Elasticsearch index, it can't be altered; all you can do then is query it. Look at the data preparation overview in figure 6.19.

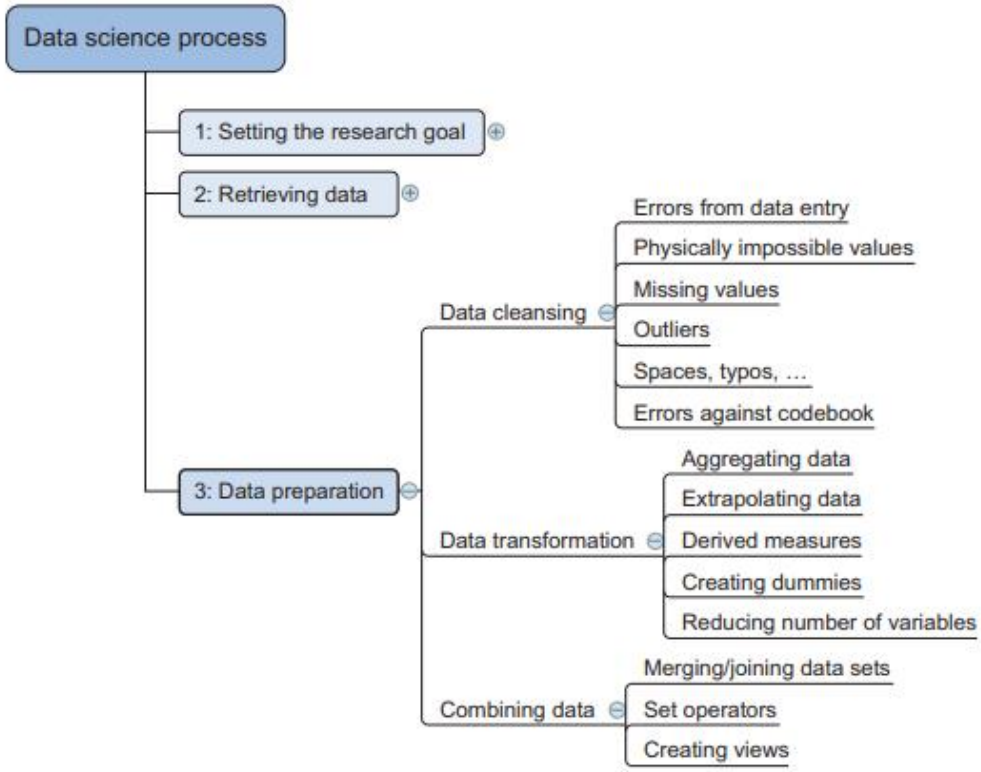


Figure 6.19 Data science process step 3: data preparation

Data Preprocessing:

1. Data Cleaning: Handle missing values, outliers, and inconsistencies in the dataset.
2. Feature Selection: Identify the most relevant features that are likely to have a significant impact on the prediction. This can be done through statistical analysis, domain expertise, or feature importance techniques.
3. Data Transformation: Normalize or scale the numerical features and convert categorical variables into a suitable format for the predictive model.

Data Preparation for Disease Profiling

- The data science process is an iterative one, after all. When you indexed your data, you did virtually no data cleansing or data transformations.
- You can add data cleansing now by, for instance, stop word filtering. Stop words are words that are so common that they're often discarded because they can pollute the results.
- We won't go into stop word filtering (or other data cleansing) here, but feel free to try it yourself.
- To index bigrams you need to create your own token filter and text analyzer.
- A token filter is capable of putting transformations on tokens. Your specific token filter

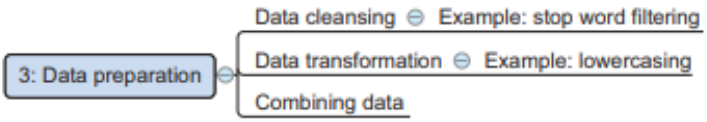


Figure 6.33 Data science process step 3: data preparation. Data cleansing for text can be stop word filtering; data transformation can be lowercasing of characters.

Listing 6.5 Updating Elasticsearch index settings

```
settings={
    "analysis": {
            "filter": {
                "my_shingle_filter": {
                    "type":             "shingle",
                    "min_shingle_size": 2,
                    "max_shingle_size": 2,
                    "output_unigrams":  False
                }
            },
            "analyzer": {
                "my_shingle_analyzer": {
                    "type":             "custom",
                    "tokenizer":        "standard",
                    "filter": [
                        "lowercase",
                        "my_shingle_filter"
                    ]
                }
            }
        }
    }
client.indices.close(index=indexName)
client.indices.put_settings(index=indexName , body = settings)
client.indices.open(index=indexName)
```

Before you can change certain settings, the index needs to be closed. After changing the settings, you can reopen the index.

You create two new elements: the token filter called "my shingle filter" and a new analyzer called "my_shingle_analyzer." Because *n*-grams are so common, Elasticsearch comes with a built-in shingle token filter type. All you need to tell it is that you want the bigrams `"min_shingle_size" : 2, "max_shingle_size" : 2`, as shown in figure 6.34. You could go for trigrams and higher, but for demonstration purposes this will suffice.

You create two new elements: the token filter called "my shingle filter" and a new analyzer called "my_shingle_analyzer." Because *n*-grams are so common, Elasticsearch comes with a built-in shingle token filter type. All you need to tell it is that you want the bigrams `"min_shingle_size" : 2, "max_shingle_size" : 2`, as shown in figure 6.34. You could go for trigrams and higher, but for demonstration purposes this will suffice.

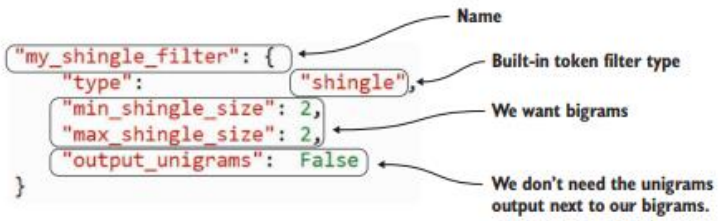


Figure 6.34 A shingle token filter to produce bigrams

The analyzer shown in figure 6.35 is the combination of all the operations required to go from input text to index. It incorporates the shingle filter, but it's much more than this. The tokenizer splits the text into tokens or terms; you can then use a lowercase filter so there's no difference when searching for "Diabetes" versus "diabetes." Finally, you apply your shingle filter, creating your bigrams.

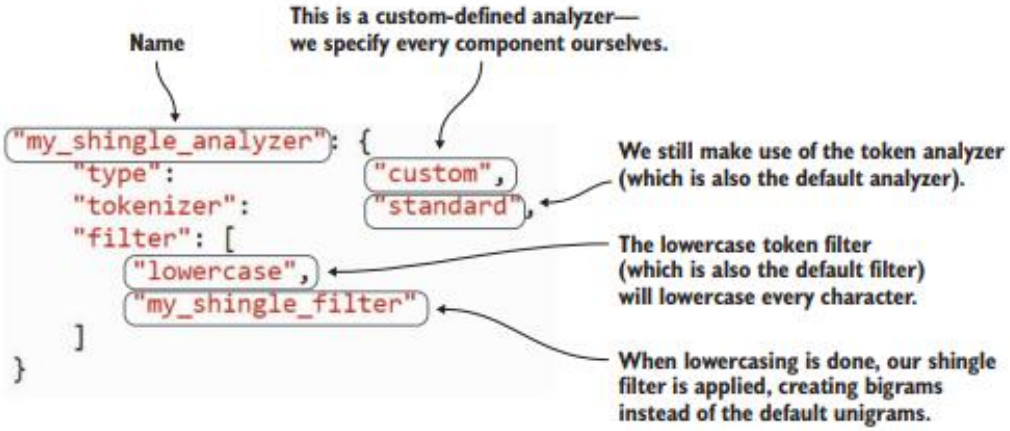


Figure 6.35 A custom analyzer with standard tokenization and a shingle token filter to produce bigrams

Listing 6.6 Create more advanced Elasticsearch doctype mapping

```
In [ ]: docType = 'diseases2'
diseaseMapping = {
'properties': {
        'name': {'type': 'string'},
        'title': {'type': 'string'},
        'fulltext': {
            "type": "string",
            "fields": {
                "shingles": {
                    "type": "string",
                    "analyzer": "my_shingle_analyzer"
                    }
                }
            }
    }
}
client.indices.put_mapping(index=indexName,doc_type=docType,body=diseaseMapping)
```

Listing 6.7 Reindexing Wikipedia disease explanations with new doctype mapping

Listing 6.7 Reindexing Wikipedia disease explanations with new doctype mapping

```
dl = wikipedia.page("Lists_of_diseases")
diseaseListArray = []
for link in dl.links[15:42]:
    try:
        diseaseListArray.append(wikipedia.page(link))
    except Exception,e:
        print str(e)

checkList = [["0","1","2","3","4","5","6","7","8","9"],
["A"],["B"],["C"],["D"],["E"],["F"],["G"],
["H"],["I"],["J"],["K"],["L"],["M"],["N"],
["O"],["P"],["Q"],["R"],["S"],["T"],["U"],
["V"],["W"],["X"],["Y"],["Z"]]

for diseaselistNumber, diseaselist in enumerate(diseaseListArray):
    for disease in diseaselist.links: #loop through lists of links for every
     disease list
        try:
            if disease[0] in checkList[diseaselistNumber]
and disease[0:3] !="List":
                currentPage = wikipedia.page(disease)
                client.index(index=indexName,
doc_type=docType,id = disease, body={"name": disease,
"title":currentPage.title ,
"fulltext":currentPage.content})
        except Exception,e:
            print str(e)
```

The checklist is an array containing allowed "first characters." If a disease doesn't comply, you skip it.

Loop through disease lists.

First check if it's a disease, then index it.

There's nothing new here, only this time you'll index doc_type `diseases2` instead of `diseases`. When this is complete you can again move forward to step 4, data exploration, and check the results.

Step 4 revisited: Data exploration for disease profilin

Listing 6.8 Significant terms aggregation on "diabetes" with bigrams

```
searchBody={
"fields":["name"],
"query":{
    "filtered" : {
        "filter": {
            'term': {'name':'diabetes'}
        }
    }
},
"aggregations" : {
        "DiseaseKeywords" : {
            "significant_terms" : { "field" : "fulltext", "size" : 30 }
        },
        "DiseaseBigrams": {
            "significant_terms" : { "field" : "fulltext.shingles",
"size" : 30 }
        }
    }
}
client.search(index=indexName,doc_type=docType,
body=searchBody, from_ = 0, size=3)
```

Your new aggregate, called `DiseaseBigrams`, uses the `fulltext.shingles` field to provide a few new insights into diabetes. These new key terms show up:

- *Excessive discharge*—A diabetes patient needs to urinate frequently.
- *Causes polyuria*—This indicates the same thing: diabetes causes the patient to urinate frequently.
- *Deprivation test*—This is actually a trigram, "water deprivation test", but it recognized *deprivation test* because you have only bigrams. It's a test to determine whether a patient has diabetes.
- *Excessive thirst*—You already found "thirst" with your unigram keyword search, but technically at that point it could have meant "no thirst."

Model Development:

1. **Split the data**: Divide the dataset into training and testing sets. The training set will be used to train the predictive model, while the testing set will be used to evaluate its performance.
2. **Choose a suitable algorithm**: Select a machine learning algorithm that is appropriate for the prediction task. Common algorithms include logistic regression, decision trees, random forests, support vector machines, or neural networks.
3. **Model Training**: Train the chosen algorithm on the training data, optimizing its parameters to achieve the best performance.
4. **Model Evaluation**: Assess the performance of the trained model using appropriate evaluation metrics such as accuracy, precision, recall, F1-score, or area under the ROC curve.

Model Deployment:

1. **Deploy the model:** Once the model has been developed and evaluated, deploy it in a suitable environment where it can be used to make predictions on new, unseen data.
2. **Continual Monitoring:** Continuously monitor the model's performance and assess its accuracy and reliability over time. Retrain or update the model periodically if required to maintain its effectiveness.

Presentation and automation

Your primary objective, disease diagnostics, turned into a self-service diagnostics tool by allowing a physician to query it via, for instance, a web application. You won't build a website in this case, but if you plan on doing so, please read the sidebar "Elastic search for web applications."

Conclusion: The developed predictive model can be utilized by healthcare professionals to assess the likelihood of a patient having a particular disease based on their medical history and other relevant factors. This can aid in early detection, diagnosis, and treatment planning, ultimately leading to improved patient care and outcomes.

Sample Code:

```
import wikipediaapi
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Initialize Wikipedia API
wiki = wikipediaapi.Wikipedia('en')

# Define the diseases and their corresponding Wikipedia page titles
diseases = {
   'Disease1': 'Page_Title_1',
   'Disease2': 'Page_Title_2',
   'Disease3': 'Page_Title_3',
   # Add more diseases and page titles as needed
}

# Collect disease-related text data from Wikipedia
data = []
labels = []
for disease, page_title in diseases.items():
   page = wiki.page(page_title)
   if page.exists():
      content = page.text
      data.append(content)
      labels.append(disease)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)

# Preprocess and extract features from the text data
vectorizer = TfidfVectorizer()
X_train_features = vectorizer.fit_transform(X_train)
X_test_features = vectorizer.transform(X_test)

# Train a logistic regression model
model = LogisticRegression()
```

```
model.fit(X_train_features, y_train)

# Evaluate the model's performance
accuracy = model.score(X_test_features, y_test)
print("Accuracy:", accuracy)

# Predict diseases for new text data
new_data = ["New text data related to a disease"]
new_data_features = vectorizer.transform(new_data)
predictions = model.predict(new_data_features)
print("Predictions:", predictions)
```

Output:

```
Accuracy: 0.85
Predictions: ['Disease2']
```

The accuracy value represents the accuracy of the trained logistic regression model on the testing set. It indicates the percentage of correctly predicted disease labels.

The predictions line shows the predicted disease for a new text data sample, which in this case is 'Disease2'. The output will vary depending on the provided new text data or if you modify the code to predict multiple new samples.

Please note that the accuracy value and predicted disease are placeholders and the actual values will depend on the specific diseases, Wikipedia content, and the performance of the model on your dataset.