



DATA QUALITY AND DATA WRANGLING

LIBFOARPDLBSDQDW01

LIBF

LEARNING OBJECTIVES

Due to the ever-increasing amount of complex data, it is important to understand the different concepts of **Data Quality and Data Wrangling** from theoretical and practical perspectives. You will begin this course with an introduction to data quality, where you will learn about the different aspects and metrics, and the different dimensions of data quality. This will include an explanation of what that means for humans and for computers. To set these concepts into practice, different activities and processes will be presented in the context of data quality management. You will learn about different roles within this change process and how they perform within the International Organization for Standardization (ISO) 8000-61 standard for data quality management.

You will also get to know the basic Python libraries for scraping different web technologies. A variety of use cases will show you how to extract, structure and clean data from formats, such as Hypertext Markup Language (HTML), Extensible Markup Language (XML), and JavaScript Object Notation (JSON). The latter will be particularly important when extracting data from application programming interfaces (APIs), which you will learn about alongside an introduction to extracting data from portable document formats (PDFs).

The handling of text-based formats (such as comma separated values [CSV], XML, and JSON) and binary formats (such as HDF5, Parquet, and Arrow) will be introduced, and you will learn the differences between them. Upon completion of this course book, you will be able to execute reading, transformation, and writing operations with those formats. You will gain an overview on how to create tidy data, which encompasses data structuring, cleansing, and enrichment. This will include methods such as data profiling, advanced character matching, statistical methods for standardization and normalization, and addition of new data.

UNIT 1

DATA QUALITY


STUDY GOALS

On completion of this unit, you will be able to ...

- define data quality.
- categorize data as an asset.
- identify the dimensions of data quality.
- reflect on causes for data decay.
- assess the behavior of individuals in regard to data quality.

1. DATA QUALITY

Introduction

When the National Aeronautics and Space Administration (NASA) lost the connection to their Mars Climate Orbiter in 1999, **not only the aeronautics community was shocked** (King & Schwarzenbach, 2020). The small orbiter, whose mission was to circle around Mars and collect valuable climate data, did not send any signals back to Earth after it reached its expected distance from Mars. Soon, the engineers identified the issue which led to the loss of the \$327.6 million vehicle. While NASA developed their systems using their measures in Newton seconds, another system, which was delivered by the supplier, Lockheed Martin, used pound-force seconds. Since neither team developed proper data specification which could have been processed by a computer to convert these measures (and therefore avoid the incident), the orbiter actually approached Mars at a distance of 57 km instead of the anticipated 150 km. Since the atmosphere is denser at this distance, the orbiter immediately burned down to its core (King & Schwarzenbach, 2020). 

There are several examples, though not as expensive as the orbiter in most cases, where a lack of data specification leads to system or process failure. In the corporate data universe, for example, where you can find ecosystems of databases linked by different processes, data batches flow between different systems. The more complex the systems, and the more people involved in the development or data integration process, the more the overall amount of information increases, though the quality of the collected data decays. This data decay leads to a loss in business value, as well as a loss in the stakeholders' trust in the delivered data products; only the combination of high-quality data, effective technology, and a data driven mindset will allow organizations to benefit from the data asset. To access those benefits, it is crucial to understand the different components of data quality and be able to identify where data decay happens (Maydanchik, 2007).

In this unit, several definitions of data quality will be presented and the concept of data as an asset will be introduced. After the identification of differences between data management and data quality management, the different data quality dimensions will be displayed. These dimensions will be put into context, and several causes of data decay will be identified. Additionally, different types of human behavior will be presented in the light of data usage to display the impact humans can have on data quality.

1.1 Introduction to Data Quality

With the steady progress in the fields of data engineering, data analytics, and data science, combined with numerous technological advances, the perception of data quality in research has changed over time. Newly developed methods to acquire, store, and edit data, as well as the growing demand for data products, has increased the complexity faced by researchers. Since data quality is a multidimensional concept that can be approached from several different directions, it is challenging to find a detailed definition

(Batini & Scannapieco, 2006). For this reason, several definitions, which have appeared over time from different researchers, take a general approach and do not consider specific technologies.

For example, Wang and Strong (1996) define data quality as “data that are fit for use by data consumers” (p. 6), and Olson (2002) states that “data has quality if it satisfied the requirements of its intended use” (p. 24). Redman (2001) includes aspects of the nature of the data by stating that “data are fit for use if they are free of defects and possess desired features” (p. 241). King and Schwarzenbach (2020) try a more practical approach by defining data quality as “the right data being available at the right time to the right users, to make the right decision and achieve the right outcome” (p. 11). They extend this definition by considering data quality attributes, such as “safe, legal, and processed fairly, correctly, and securely” (King & Schwarzenbach, 2020, p. 11). These definitions of data quality share the assumption that several expectations exist regarding the data, which can be defined formally or informally. These expectations derive from either the data consumers, the specifications of the data itself, or the intended use of the data (Fürber, 2016). While these definitions show the general intentions of data quality, it also becomes clear that data quality is always measured against the requirements in every individual case (Redman, 2001).

While, in theory, the target is to achieve perfect data quality, this is utopian in a practical environment. Due to several reasons, including time, budget, and the actual expectations of the data, there will always be a tradeoff between data quality and efficient data processes (King & Schwarzenbach, 2020).

Data as an Asset

In many different industries, the amount of collected data has increased over the past decades, and data are now considered more important than ever before. Almost every organization tries to gain a competitive advantage by implementing data-driven business practices. While technical challenges limit the potential of a truly data-driven philosophy, the potential also depends on the mindset of the people working within an organization. King and Schwarzenbach (2020) make the case that data should be regarded as an asset similar to physical assets or human resources, since they share a lot of characteristics.

Data can be highly valuable to any organization by improving existing processes, identifying opportunities, and enabling the creation of new products. This can enhance business performance and reliability by ensuring that decision-making is based on evidence instead of predictions. Furthermore, data can be assessed using quality measures, which help to avoid high business costs.

The main difference between data and any other physical asset is the potential cost of failure. Since data support the strategic decision-making process, wrong insights can have long-lasting and potentially expensive impacts on an organization. Another difference is that the data assets are not destroyed after they are used. This creates more value since data assets can be utilized for several processes or products, and therefore gain value over time.

Data Quality Metrics

To measure the quality of data, a set of metrics is required for monitoring purposes. Pipino et al. (2002) describe those metrics in a general approach by measuring “the ratio of desired outcomes to total outcomes” (p. 213). Since this approach is closely linked to the defined set of requirements, Heinrich and Klier (2010) take the real-world item into account. By defining data quality as the degree of harmonization between the data stored in a database and the corresponding real-world object, they move the focus away from the self-defined expectations to an appropriate representation.

From a more practical perspective, it is important to note that the metrics are not only linked to the data itself, but also to the production, access, and presentation processes. Since the metrics depend on the specific use case, the following examples can serve as a starting point (Olson, 2002):

- number of entries with at least one wrong value
- number of key errors (nonredundant primary keys and primary or foreign key orphans)
- total volume of data stored
- time spent handling data
- number of unauthorized accesses

Once data quality metrics are defined and implemented in a system, they are regularly evaluated, and actions are taken based on the measured results. Therefore, the developed metrics can be regarded as the starting point for any further analysis of data quality decay (Fürber, 2016).

Data quality metrics can be a good indicator of the progress made toward reaching a higher overall data quality. They will show, for example, whether or not a set of guidelines actually has an impact. Additionally, they can be used to evaluate new data sources that will be migrated into the system. The downside of such metrics is that they are difficult to define and hard to measure, and they create an awareness of problems without solving them. It is difficult to identify every error in a database, so a list of potential errors will never be complete. Another issue with data quality metrics is that they are often regarded as a one-time action; however, in order to be able to profit from such metrics, they must be measured and improved continuously. The required resources are not always available, which often turns those efforts into a one-time project (Olson, 2002; Pipino et al., 2002).

Data Quality Management versus Data Management

Data life cycle

In a data life cycle, a document can pass through the following phases: create, store, publish, update, supersede, retire, and dispose.

While data management can be regarded as the overall handling of the **data life cycle** within an organization, data quality management focuses on one characteristic of the data asset. Other characteristics, or **knowledge areas**, in the field of data management are as follows:

- data governance
- data architecture
- data modeling and design
- data storage and operations

- data security
- data integration and interoperability
- document and content management
- reference and master data
- data warehousing and business intelligence
- metadata
- data quality

Knowledge areas
 The Data Management Body of Knowledge (DAMA-DMBOK) identifies 11 knowledge areas within data management, which aim to support effective implementation (King & Schwarzenbach, 2020).

The narrow field of data quality management is described as the “coordinated activities to direct and control an organization with regard to data quality” (King & Schwarzenbach, 2020, p. 14). This shows that data quality measures must be taken in to account at every single step at which data are processed. This includes the root cause analysis of errors, setting of standards, and developing guidelines.

1.2 Data Quality Dimensions and Issue Types

The degree to which data quality is achieved is closely linked to the requirements defined for the data. This leads to data being used for different use cases with different requirements within one organization, which can be a problem. This is the reason that data quality has to be tested to determine whether it is fit for its purpose. Considering that this fitness is defined by a set of requirements, it is key to be able to characterize these requirements in the most specific way possible. This is done along several data quality dimensions. In this section, two different approaches to data quality dimensions are presented, alongside their strength and weaknesses. The first approach involves the data quality dimensions according to Wang and Strong (1996) with a focus on the data consumer, and the second is a more compressed approach by King and Schwarzenbach (2020).

Data Quality Dimensions

Wang and Strong (1996) define four categories (intrinsic, contextual, representational, and accessibility) that should be considered when defining the quality of a dataset or source. Within these categories, they define several dimensions in order to give guidance, from a consumer point of view, concerning data evaluation (Wang & Strong, 1996).

Intrinsic

While accuracy and objectivity seem to be a presumed aspect of intrinsic data quality, Wang and Strong (1996) also identify the dimensions of believability and reputation of the data source as key indicators of data quality. While building a data product, information system professionals need to take the following dimensions into account in order to meet the demands of data consumers:

- believability. This is the degree to which the users trust the data and data source, and accept them as correct. Users must believe that when data arrives, they are fit for purpose.
- accuracy. This is the degree to which the users consider the data to be error-free and accept that they can rely on them.
- objectivity. This is the degree to which the data are free of any prejudices and detached from any assumptions.
- reputation. This is the degree to which the users trust the original data source and, therefore, the delivered data.

Contextual

In their research, Wang and Strong (1996) identify the need to put the data quality dimension in the context of the specific use case. Since the nature of tasks vary in different contexts, this aspect of data quality is quite challenging in a research setting. Wang and Strong (1996) therefore parameterize the context into the following five aspects:

1. Value-added. This is the degree to which data deliver a measurable benefit to the user.
2. Relevancy. This is the degree to which data are fit for the intended purpose.
3. Timeliness. This is the degree to which the data are delivered to the user at the correct time.
4. Completeness. This is the degree to which data display a holistic picture of the situation while fulfilling the scope for the intended use.
5. Appropriate amount of data. This is the degree to which data are available in an appropriate quantity.

Representational

The format and the meaning of the data are related to representational data quality. From a technical perspective, this could include the syntax and the semantics of the data. The following aspects are taken into account:

- interpretability. This is the degree to which data are specified and defined clearly, encompassing a well-defined language and appropriate measurement units.
- ease of understanding. This is the degree to which data are comprehensible in a set context.
- representational consistency. This is the degree to which data follow a consistent format that will not change over time.
- concise representation. This is the degree to which data can be presented to the user without creating a daunting amount of information.

Accessibility

In their research, Wang and Strong (1996) discover that data consumers often presume that the data are always available and can be accessed easily. They also identify the demand for security measures to keep authority over the data. These requirements are explained as follows:

- Accessibility is the degree to which data can be retrieved in a convenient and efficient manner.
- Access security is the degree to which data access is controlled and monitored in order to maintain the integrity of the data.

This research mainly focuses on the needs and perception of the data consumer, which can be misleading, since data consumers often do not possess the knowledge to distinguish between hardware, application, and data when considering the data quality. If a data user, for example, accesses a dataset and misses the most current data, it will require some time to investigate and identify the issue. If the user does not possess the knowledge to identify the issue, it can lead to a bad user experience. Additionally, data consumers might overlook data quality problems, such as data redundancy, which could be uncovered by a data producer or custodian.

In order to include other perspectives on data quality, a framework that builds on the fundamentals of computer science was defined. This framework, which is defined in the ISO 8000-8 standard, identifies three types of data quality: syntactic, semantic, and pragmatic (King & Schwarzenbach, 2020). Since these three types of data quality can be considered abstract, a more popular approach was developed, which uses accuracy, completeness, consistency, validity, timeliness, and uniqueness. The following list gives a brief explanation of each of these dimensions (King & Schwarzenbach, 2020):

- Accuracy is the extent to which a data object is a proper representation of the real-world object.
- Completeness is the extent to which a relevant data object is captured and all necessary attributes are stored.
- Consistency is the extent to which a data object in one data store is comparable to the same data object in another data store.
- Validity is the extent to which the data object follows set data specifications in terms of format or measurement units.
- Timeliness is the extent to which the data object reflects the current status of the real-world object, as well as an appropriate amount of time to receive the data.
- Uniqueness is the extent to which each data object only represents one real-world object and is not redundant in the data store.

The presented dimensions of data quality according to Wang and Strong (1996) and King and Schwarzenbach (2020) each have their strengths and weaknesses, depending on the perspective. However, all of these dimensions require the analysis of data requirements. Additionally, these dimensions might be replaced by the data specifications defined in the data quality assessment process.

Causes of Data Quality Problems

Before data are incorporated into a data product, they pass through numerous processes that can impact the quality of the data. Maydanchik (2007) identifies three groups of processes that are prone to create different kinds of data quality problems.

The first group describes processes that import data from the outside world into the system. This step could be executed either manually or through various data integration techniques. Sometimes, data quality issues in this group of processes arise due to poor data collection or misconfiguration of the data extraction, transformation, and loading (ETL) process.

A second group of issues arises from processes that alter the data within the data storage. This includes automating processes or system updates, and redesigning the data structure. While good practices exist to carry out these processes, a lack of time and resources often leads to a variety of problems and decaying data quality.

Processes that restructure, rearrange, or change over time are collected in the third group. These problems appear when use cases change without adjusting the data acquisition process. In these scenarios, the old data lose their value over time due to the changed requirements.

Group One: Processes Bridging Data from Outside

Initial data conversion

Usually, when a new data store (e.g., a database) is created, there are existing data that need to be introduced into it. These data could stem from a new data collection process or from old “legacy” systems. In both scenarios, there is often a lack of metadata about the old source, and information can get lost during the import process. This is particularly prevalent when it comes to the conversion of legacy systems where a huge variety of business rules can be applied to the data elements, and the data quality therefore decays.

System consolidations

When combining different systems, the same problems appear as in the initial data conversion. However, during system consolidations, both systems (the old one and the new one) are non-empty and follow their own logic. Therefore, a conflict can be expected. The complexity increases when there is a certain overlap between the two sources, and one has to handle duplicates or deviant designation of data entries. While methods exist for handling these kinds of issues (e.g., the winner-loser matrix), a short timeframe or lack of resources may not allow this issue to be addressed properly.

Manual data entry

Even though a series of highly automated processes that bring data into a database exist, many data still enter a data store through manual input by a human being. Since humans do make errors, there is always a risk that the syntax or the semantic structure of these data entries is wrong. Common problems are misspelled words, blank values, or default values that are not overwritten. Most data entry interfaces and the data stores themselves often include type checks, which take care of misspellings or formats. Nevertheless, these techniques are not the standard in all data entry processes and are seldom sufficient to completely eradicate issues associated with manual data entries.

Batch feeds

For the exchange of data between two systems, batch feeds can be used. These are processes that automatically send data from one system to another with ever increasing amounts of data. Since systems alter over time, information about these changes is often neither properly documented nor propagated downstream to the receiving system. This can lead to falsely structured data entries.

Real-time interfaces

In many systems, the exchange of data happens in real-time, or near real-time, to meet the demands of the data consumers. While real-time data seem desirable at first, one of the main challenges is that data might propagate too fast without passing through proper verification methods. While the validity of new data entries might be checked, there is often no process to check for rejected inputs. This shows that fast data are not automatically good data. With the increase of execution speed, data quality often suffers, and a trade-off has to be considered for a given use case.

Group Two: Processes Causing Data Decay

Changes not captured

Communication is often considered to be the key for high data quality. If, for example, a real-world item changes its status, but this change is not properly propagated to the database, data quality decays. Therefore, it has to be ensured that the data acts as a virtual twin of real-world entities. In reality, this seems utopian since data acquisition processes are not perfect and humans might not propagate the changes properly. This often leads to mismatching information in different systems.

System upgrades

Every system needs to be upgraded from time to time in order to meet new technological standards and handle defects or security risks within the system. Upgrades normally do not have an extreme impact on the quality of the data in comparison to system conversions or consolidations, since they focus mainly on the systems architecture and not the data within it. Nevertheless, if the data are wrongly defined or used for a different purpose, upgrades can disrupt systems. This often derives from a lack of metadata, which define the proper datatypes and purpose of the data. If such metadata are missing, the upgraded system is tested against the expected data rather than the actual structure of the data. A lot of work is required to solve this mismatch and bring the system back.

New data uses

Since data quality is measured against the data usage requirements, a change in data usage has a considerable impact on the quality. While a 20 percent error on people's addresses might be acceptable for marketing purposes, this error might not be acceptable when the data are used for billing these people. In addition, the granularity of the data might be adequate for one use case, but not detailed enough for another.

Loss of expertise

When using data, it is necessary to possess a certain degree of knowledge about that particular data. Since systems often develop over time, the knowledge of people working with these datasets increases over time. Long-term users of the data know which fields to use and why or where data might not be collected properly. While ideally this knowledge is well-documented, in many cases, this knowledge is only available in the minds of those who worked with the data. As a consequence, this can lead to data decay as expertise is lost over time when experts on the data move on to new positions or retire.

Process automation

In most cases, process automation can have a positive impact on data quality, for example, by standardized input formulas or automated error checking; however, process automation can also be challenging. Automated processes, for example, lack the proper judgment to identify false data for scenarios that are not defined or new to the system. Furthermore, a system's information might be available to more people due to automation, which can lead to the exposure of data issues and complaints from users. While this can be regarded as a good thing, it can easily create additional workload, especially when users are not properly trained to understand the data.

Group Three: Processes Changing Data from Within

Data processing

There are different types of data processing, ranging from massive overall aggregations and calculations to minor adjustments of data entries. While, in theory, these processes seem to be repetitive, this is not the case in a real-world environment. One source of issues is when systems change over time. Even small changes that are not properly tested can have a huge impact on the data quality. A small bug can alter many entries, especially when specific processes are triggered by specific inputs. The key to avoiding such problems is clear communication with all systems, proper documentation of the changes and the system itself, and clear standards for each data entry.

Data cleansing

Data cleansing can be a double-edged sword when it comes to data quality. On the one hand, it obviously removes or alters incorrect or wrongly created data entries. On the other hand, this could lead to further errors or an incorrect interpretation of the data. Due to the advances in automated data cleansing with a variety of software products, data cleansing is conducted in a rushed manner without taking the complexity or dependencies of the data entries into account.

Data purging is a necessary step in data governance in order to avoid overloading the databases with negligible information. However, it can also be the source of a variety of issues. If data are deleted, there is always the chance that they might be needed again in

the future due to changing requirements. Additionally, in some scenarios, there may be dependencies on legacy data, which, if deleted, will have a negative impact on the data quality.

Technical and Human Aspects in Data Quality

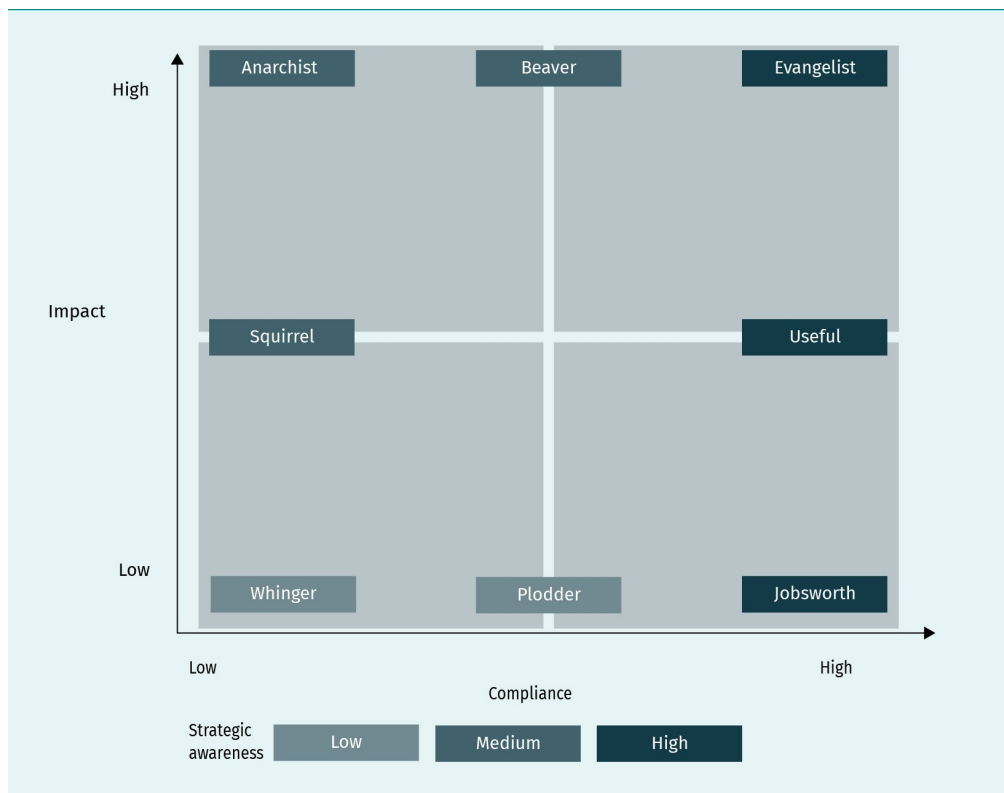
Humans have a major impact on data quality. The impact can come from manually entered data; changing systems' configurations, such as data object specifications; or incorrect data collection processes. In an organization, many different touchpoints exist between humans and data, and data decay often depends on the view of the individual working with it. The concept of the data zoo aims to highlight the different behaviors of humans in this context.

The Data Zoo

Depending on the individual's behavior in their business activities (e.g., the attention to detail or whether rules are followed), data quality can increase or decay. King and Schwarzenbach (2020) developed a framework, called the data zoo, by categorizing the behavior of individuals along three main dimensions: compliance, impact, and strategic awareness. Compliance describes to what degree people follow the written and unwritten rules or standards of an association. The effect of the data-related actions is measured in the impact dimension, and can include a variety of steps within several data processes. Strategic awareness measures an individual's knowledge of how the contribution will help or hinder the achievement of the goals of the organization.

King and Schwarzenbach (2020) identify eight exaggerated types of individuals that can appear within an organization.

Figure 1: The Data Zoo



Source: Niklas Wietreck (2021), based on King and Schwarzenbach (2020).

Jobsworth

A jobsworth is typically unproductive, since they always raise questions of ownership and clarity in data, processes, or systems in order to avoid work. While these traits might be good in highly standardized environments, it can hinder progress in scenarios where requirements frequently change. A jobsworth has a low impact, is highly compliant, and has general strategic awareness.

Evangelist

The data evangelist understands the importance of data standards and processes while keeping the big picture in mind. They will try to improve the overall performance of the data activities by encouraging others, and can be considered data leaders. An evangelist has a high impact, is highly compliant, and has high strategic awareness.

Plodder

Data plodders can be regarded as minimalist when it comes to data activities. They will follow corporate standards, as long as minimal energy is required. Plodders tend to settle with an acceptable job, which can result in introduced data errors that will not be solved since these issues are not brought forward. Plodders have low impact, are a medium level of compliant, and lack general strategic awareness.

Beaver

Data beavers are hard workers, and they are vocal about that. It may be that the data beaver does not follow the standard because of a lack of strategic awareness, as well as misguided enthusiasm. High impact with a medium level of compliance characterizes the beaver, combined with a medium level of strategic awareness.

Whinger

Data whingers lack the overall awareness of why data are important for the organization, and prefer to complain about standards, processes, and systems. They not only avoid engaging in any data-related activity, but also have a demotivating impact on the people around them. Getting a data whinger on board requires a lot of energy and endurance. Therefore, the data whinger is characterized by low compliance, low impact, and low strategic awareness.

Anarchist

Data anarchists tend to set up their own infrastructure since they do not agree with the corporate data system (sometimes for good reason). Nevertheless, the effort they put into their data activities results in negative or no impact, since they create duplicate datasets. They are characterized by a low level of compliance, high impact, and a medium level of strategic awareness.

Useful person

The useful person follows the rules while working enthusiastically. They will run through procedures and will become vocal about any data issue that arises. Colleagues see the useful person as valuable and a positive teammate. A useful person delivers stand out results; they are highly compliant, and have a medium impact and high strategic awareness.

Data squirrel

Data squirrels tend not to follow standards, technologies, or processes. Similar to the anarchist, they will set up their own way of working with a “knowledge is power” approach, which means that they will hoard their knowledge and only share it if requested. From an organizational perspective, this can be critical since valuable knowledge might be buried somewhere on a local computer. Data squirrels have a low level of compliance, medium impact, and a medium level of strategic awareness.

There are seven other types in this characterization of individuals, which are a combination of the tendencies within the aforementioned types. They are the drill, prophet of doom (PoD), innovator, agnostic, somebody else's problem (SEP), hedgehog, and not bovered (King & Schwarzenbach, 2020).

The data zoo aims to identify traits in individuals when it comes to handling data. This can be challenging since the aforementioned types are exaggerations of tendencies, and a clear separation is not usually possible. Identifying those traits helps to match the individual to the appropriate role, and steps can be taken to address specific behaviors that don't improve the data quality. The concept of the data zoo can also play an important role when a team is formed within an organization. Since successful teams are normally highly diverse, including members with a variety of personality traits (Belbin, 2013), the data zoo can support the selection of the right members for the team.

SUMMARY

There are a variety of definitions for data quality, and most literature on the topic defines it differently. Despite their differences, all definitions tie data quality to specific requirements concerning data usage, which must be described in an accurate manner.

Data are an asset similar to physical resources, since they share some characteristics, including the measurement of quality or value creation. Differences between data assets and other assets include that the impacts of data are more long-lasting, resulting in a higher cost of failure. Additionally, data are not necessarily destroyed after use, and can therefore be used repeatedly, requiring proper data management.

Data management combines 11 different knowledge areas of data-related activities, including data quality, data governance, and data security (King & Schwarzenbach, 2020). There is a considerable overlap between those knowledge areas, meaning that any data quality activity has some data governance aspects.

Wang and Strong (1996) classify the data quality dimensions in four different groups: intrinsic, contextual, representational, and accessibility. They look at it from a user perspective, omitting some important aspects that are internal to data systems. King and Schwarzenbach (2020) aim to improve this model by identifying accuracy, completeness, consistency, validity, timeliness, and uniqueness, which can help to find weak spots in the system where data decay could appear.

These weak spots can be put into three groups: combining processes where data are bridged from the outside world into the data store, combining general processes within the organization or system that cause data decay, and focusing on processes that change the data objects

themselves. It is necessary to understand the types of individual interacting with the data. The individuals can be analyzed by the impact of their work, their level of compliance, and their strategic awareness.



UNIT 2

DATA QUALITY MANAGEMENT

STUDY GOALS

On completion of this unit, you will be able to ...

- define the main principles of data quality management.
- categorize data management-related terms.
- assess the data quality maturity of an organization.
- implement processes to improve data quality.
- interpret data as a service and data virtualization.

2. DATA QUALITY MANAGEMENT

Introduction

In the United Kingdom (UK), local government units are required to monitor certain activities and report the resulting measures in the form of a key performance indicator report (King & Schwarzenbach, 2020). This became too extensive for the local units and involved an increasing amount of people and data sources. The data were collected into spreadsheets and then sent to the respective organization within the central government. The central organization knew that this process was prone to data errors since numerous weak spots exist. Therefore, the central government agreed to test the implementation of a standardized data collection and reporting system in the form of a data warehouse with a business intelligence solution. The required steps were carried out in order to implement the business rules in the system. The result was an efficient and effective enterprise business intelligence system prototype (King & Schwarzenbach, 2020).

An additional outcome was that the concept showed that the existing method of reporting fails to accurately display the current statutory performance. Each local unit had developed their own business rules over the years by treating data entries differently by, e.g., rounding numbers up or down differently, or using different naming conventions. Through a series of high-level meetings, decision-makers had to agree on how to handle the existing data and how to switch to the more standardized approach.

This use case demonstrates the impact of a lack of policies, standardization, continuous monitoring processes, and clear data specifications. This unit will focus on the distinctions between different terms when it comes to data quality management, as well as the description of a variety of roles within data management. The main principles of data quality management will be presented in line with the ISO 8000-61 standard (King & Schwarzenbach, 2020). Several processes are introduced, which can be used as a guideline to achieve higher capability levels regarding the quality of the data. To conclude this unit, the concepts of data as a service and data virtualization are introduced as specifically practical means of data quality enhancement by increasing data accessibility.

2.1 Data Quality and Stewardship

It is worth noting that many data-related terms are not used consistently in popular media and real-world marketing. The scientific community, however, has clearly separable definitions for most of these terms. The definitions often vary in their precision since some researchers broadly define their terms to make them applicable in different scenarios. Quality data, for example, are defined as “data that are fit for use by data consumers” (Wang & Strong, 1996, p. 6), and “data has quality if it satisfied the requirements of its intended use” (Olson, 2002, p. 24). While data quality can be considered an attribute of a specific dataset, it can also be seen as a knowledge area, as defined in the Data Manage-

ment Body of Knowledge (DAMA-DMBOK) (King & Schwarzenbach, 2020). This defines a collection of data management techniques, with the emphasis on improving the overall quality of the data.

The implementation of these techniques is called data quality management and offers a framework with 20 lower-level processes (King & Schwarzenbach, 2020). These processes will be described in detail alongside the key principles behind data quality management. In data quality management, data are regarded as an asset when combined with adequate data management practices to harvest any potential value within the data.

Data quality management is often equated with data governance, and there is a considerable overlap when it comes to the creation of policies and standards. However, since data governance is a knowledge area itself (like data quality, data architecture, data security, and data warehousing), it focuses on different aspects in more detail. As the name “governance” already indicates, there is a focus on setting formal policies across an organization. This includes strategic activities, such as defining policies and how to enforce them with the general theorem of centralizing governance while delegating execution. Data quality management, on the other hand, focuses on the delivery of a concrete process framework, activities, and guidelines to improve the quality of the data. While some strategic processes exist, most of them focus on the execution.

One aspect that is present in both knowledge areas is data stewardship. This is often embodied in the role of a data steward who has oversight over any data activities and is in charge of delivering “fit for purpose” data at the desired time to the intended user. Data stewards are specialists in data governance and defining data quality processes by developing and delivering process frameworks, policies, and guidelines (Marco & Jennings, 2004). Additionally, they ensure that all data-related activities are in line with the main principles of data quality management.

Data Steward, Data Owner, and Chief Data Officer

Within the data management field, there are a variety of roles that distinguish themselves by the practicality or strategic nature of their activities: data steward, Chief Data Officer (CDO), and data owner. The data steward normally has a “doer” mentality. Ideally, data stewards sit across various professional positions throughout the organization and execute the guidelines decided upon by the CDO (Samitsch, 2015). The role of the CDO was defined when data became an asset for many organizations. The purpose of the CDO is to encourage each department to take advantage of existing data while identifying new business opportunities. This is often regarded as an exclusively strategic role, but the CDO is also accountable for the efficient storage of the data as well as their quality. Additionally, the CDO has decision-making competency regarding any changes or new policies that need to be set up (Treder, 2020). The CDO is in charge of effective overall storage management, but the ownership of specific datasets will most likely be delegated to the data owner. Organizations struggle to identify data owners who solely own a dataset. The reason for this is that datasets are often the outcome of several processes that combine different inputs. If the data owner possesses the responsibility for the final dataset with only a small influence on the individual process, it will result in poor data quality. In order to achieve a certain level of data quality, it is therefore suggested to strive for process aspects

of data ownership. This way responsibilities are easier to distinguish. Data ownership is also often confused with data storage management, which is merely a support function of data quality management (King & Schwarzenbach, 2020).

Change Management

It is important to note that assessment and improvement of data quality require a certain degree of change in the organization. Since this change can have a strong influence on the overall business activities of the organization, it needs to be managed carefully. The research field of change management combines a collection of methods and practices to guide organization during the change process. In general, changes can include a new organizational structure, a change of production techniques, and the introduction of new technologies (Hashim, 2013).

These changes can be triggered by a variety of events which can be put in to two groups: external and internal triggers. External triggers include the development of new technologies, customer behavior, government policies, or competition. Internal triggers are, for example, the emergence of innovation within the own organization or new top-level management who aim to improve business practices (Hashim, 2013). Since data quality management is a continuous process, it can be regarded as a long-term change management project that has to follow stages and a set of principles (King & Schwarzenbach, 2020).

Top-Down and Bottom-Up

When it comes to the aforementioned changes there is always the question of who drives them. Though triggers for change can differ, two different approaches exist. In the top-down approach, change is driven by the strategic management level, while the bottom-up change is driven by the operational layer.

In the top-down approach, top-level management specifies the business problem that needs to be solved, and goals are set to find solutions. It is crucial to properly communicate these goals to the bottom levels in order to include each individual in the change process. The bottom-up approach has the operational work in mind. The individuals doing the work create and drive innovation in order to improve the overall business process (Watson & McGivern, 2016).

Since there are different motives for change behind the two approaches (strategic business needs versus operational tools and processes), it is necessary to calibrate both to get the best from both sides (Watson & McGivern, 2016).

Main Principles

When carrying out data quality management, five distinct principles can be applied to enhance the quality of the data. While working with data includes a variety of technical skills, the following principles take a holistic approach by being technology agnostic (King & Schwarzenbach, 2020).

Implement a process-centric approach

Any data that run within a system follow a certain process. While these processes often fail to reach data quality standards, it is necessary to align the data quality management activities along these processes. This includes adjusting the processes to monitor, test, and improve the processes and the data.

Aim for progressive maturity

Since, like systems, requirements change over time, a realistic way to achieve good data quality is a “plan, do, check, act” cycle. Therefore, it is necessary to understand the maturity level at the beginning and end of the cycle. This iterative process will introduce improvements until the target maturity is reached. The target maturity should be chosen realistically since data without any errors are unobtainable.

Create explicit data specifications

To address correct requirements toward the data, it is important to define clear and interpretable data specification. This includes the data providers and the data users. Focus should be on the communication between all individuals in contact with the data, especially when specifications change over time.

Roles and responsibilities for end users

In an organizational set up, almost any individual is capable of influencing the data in some way. To address the potential issues that could arise, it is important to create awareness of data quality and communicate responsibilities. This is often done by assigning roles to individuals, who then receive the appropriate training.

Data quality management should not be led by the IT department

Traditionally, everything related to data was handled within the information technology (IT) department. While this might make sense from a technological perspective, it fails to address the holistic view on company data. To address this, organizations might introduce a CDO who focuses on the creation of business value and the sharing and utilization of the data asset. It is important to keep in mind that working with data involves more than setting up a database.

2.2 Activities and Processes

Before starting with any data quality activities, a few considerations must be made and challenges have to be addressed. First of all, data and knowledge about them are often distributed and not consolidated in one place. This can require a lot of effort when data must be used in an integrative manner within a project, especially when the amount of

different potential data sources grows rapidly and ownership or stewardship are not clearly defined. Some data sources might also be held by different software systems, which might not be directly accessible or interoperable.

Since different individuals are involved in different processes, it is important to create a collective view of the data. Historical definitions or specifications in different parts of an organization can hinder the data quality activities. Human factors must also be taken into account since not everyone in the organization has the same view on the importance of data.

Data Quality Management Capability Levels

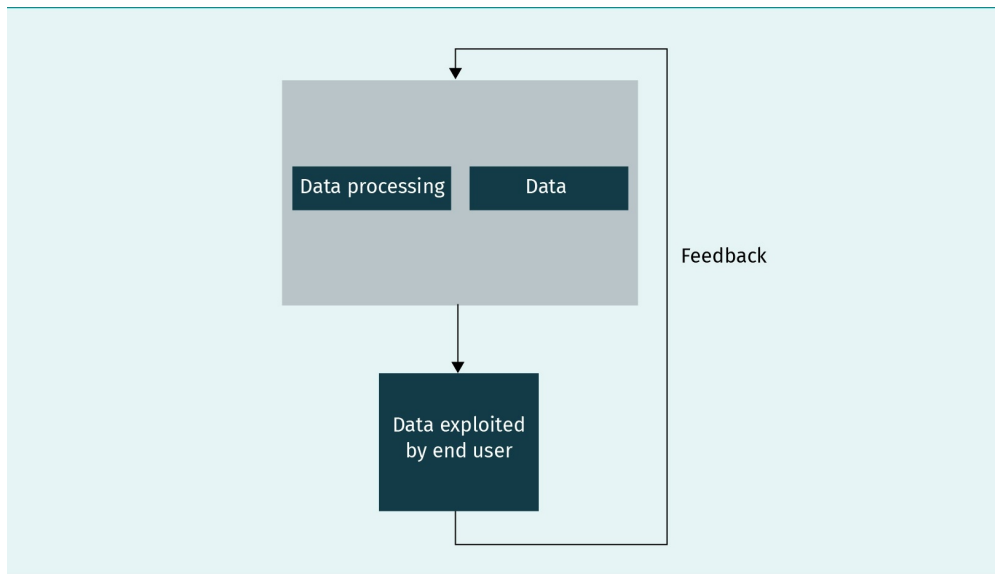
To be able to define which activities or processes need to be executed to improve the overall data quality level, it is important to understand which measures the organization is currently taking. For this reason, the capability level framework was developed. It describes several stages, which identify the activities an organization is capable of executing in order to maintain high quality data.

Since data quality management follows an iterative approach, the aforementioned stages are not meant to be implemented within one project. A practical setting is more accurately reflected when data quality management is considered as a journey along different capabilities. It is expected that a higher capability level will identify new opportunities based on the activities carried out at the lower levels (King & Schwarzenbach, 2020). These capability levels offer a clear and intuitive description of the state of an organization concerning the overall quality of its data. Here, data quality is said to enable the organization to achieve higher reaching goals with its data. This conceptual visualization of the abstract concept of data quality helps to identify aspects that can be improved for this organization.

Capability level one

The first level focuses on reliable data processing as the fundament of data quality management.

Figure 2: Capability Level One of Data Quality Management



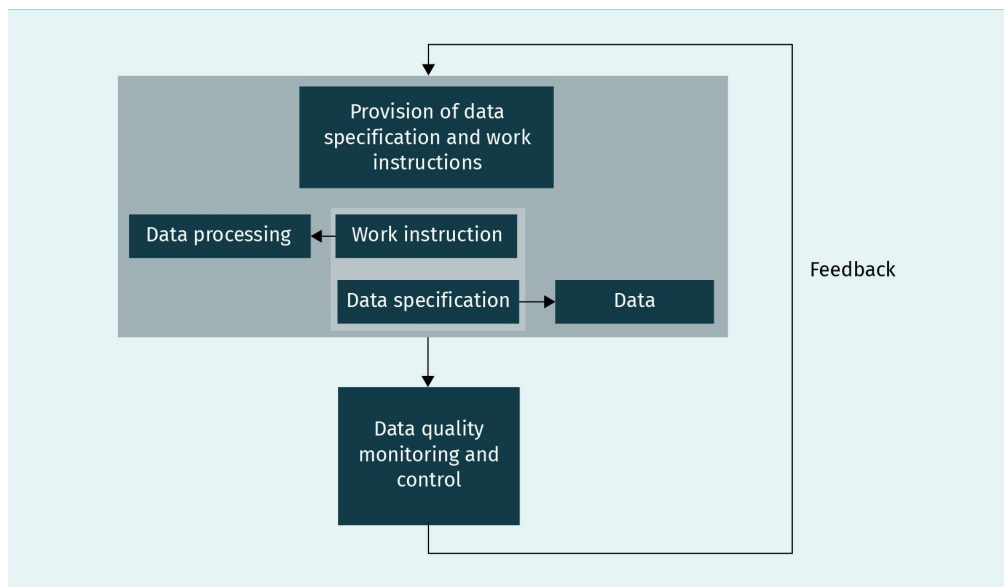
Source: Niklas Wietreck (2021), based on King and Schwarzenbach (2020).

On this level, the focus lies on the data that are delivered to the end user to support a decision-making process. The focus is normally on current decisions that need to be made, but the data should also be able to meet future requirements. Technically skilled individuals with an understanding of the data need to check whether the delivered data meet the demands of the end users. If this knowledge is not already available from the data provider, it must be created. At this level, a feedback mechanism should also exist. This should provide information about whether the delivered data matched the end users' requirements.

Capability level two

While requirements are gathered informally at the first capability level, the second level constitutes a structured approach to understand the needs of the end user in a formal manner.

Figure 3: Capability Level Two of Data Quality Management



Source: Niklas Wietreck (2021), based on King and Schwarzenbach (2020).

The different data processes, as well as the data objects themselves, are defined to ensure that they deliver the expected output. The formal and systematic analysis of data specifications are the basis of this stage and can be combined to formulate engineering requirements. Together with the formal description of the overall process, a holistic view of the activities will be displayed. Additionally, capability level two requires a data quality monitoring and control component to assess whether the data and processes are fit for their expected purposes.

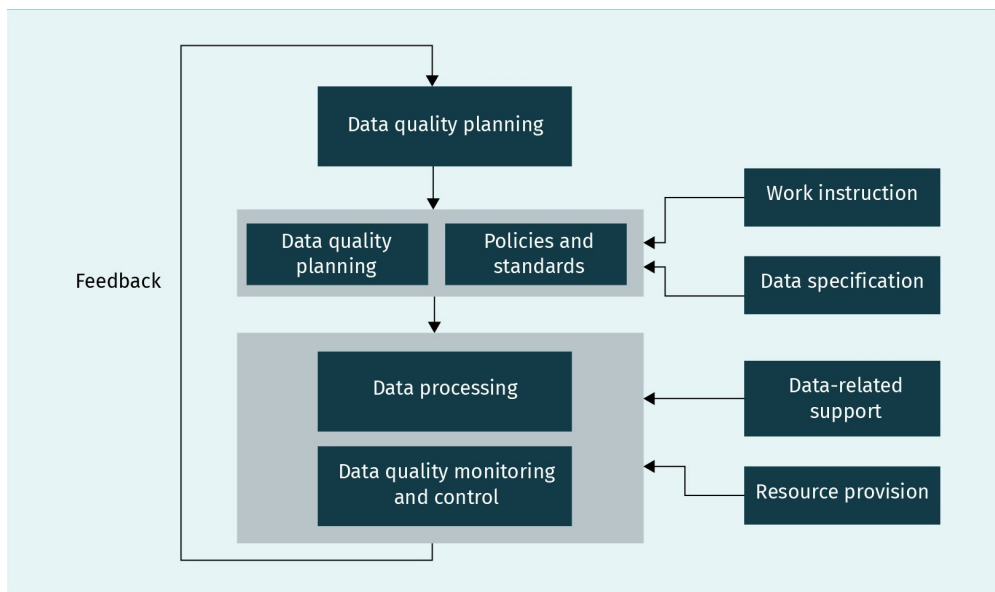
Capability level three

The focus of the third capability level is the planning of data quality. It aims to identify opportunities where data quality management could be carried out and to deliver the right policies and standards to consolidate the existing processes as much as possible. It can be regarded as a control mechanism for the activities that are carried out on levels one and two.

Since an iterative approach is followed, standards and policies that are defined later will propagate back to the initial processes and levels. While the standards are usually broad at the beginning, they turn into concrete data specifications over time.

Additionally, level three provides data-related support components and resources. This can include, for example, providing different types of infrastructure, a recruitment program for new team members, or the establishment of a continuous training program.

Figure 4: Capability Level Three of Data Quality Management

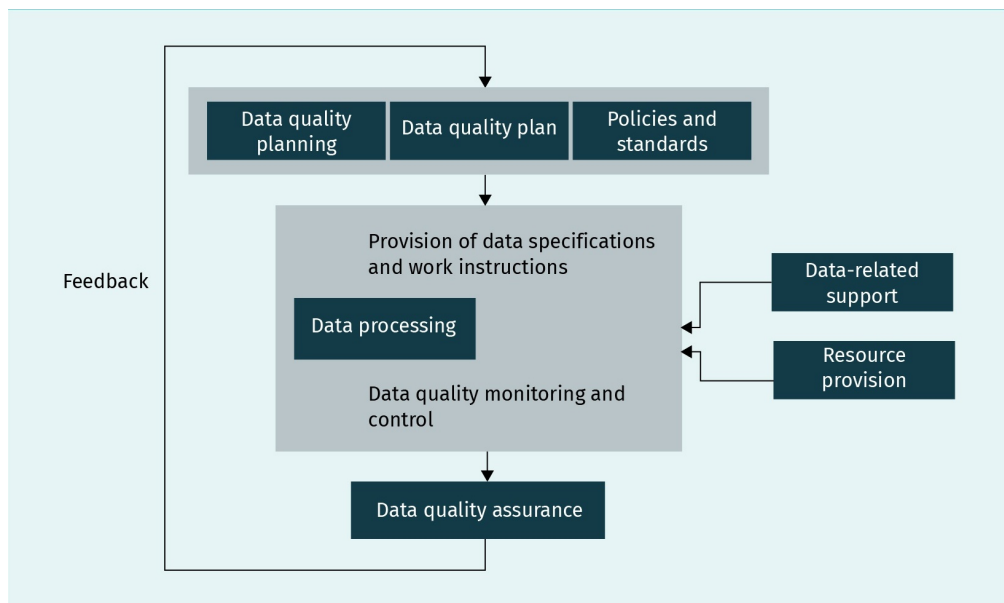


Source: Niklas Wietreck (2021), based on King and Schwarzenbach (2020).

Capability level four

The fourth capability level does not focus on a particular data transformation process. Instead, it focuses on the overall effectiveness and efficiency of data quality management activities. It aims to recognize when processes within the system do not follow the standards, and where improvements and alignments must be conducted. On this level, the characteristics of the whole system are assessed and the performance of the data quality activities is measured against those characteristics. The result of this data quality assurance test is then propagated to the other levels to promote the execution of adequate actions.

Figure 5: Capability Level Four of Data Quality Management

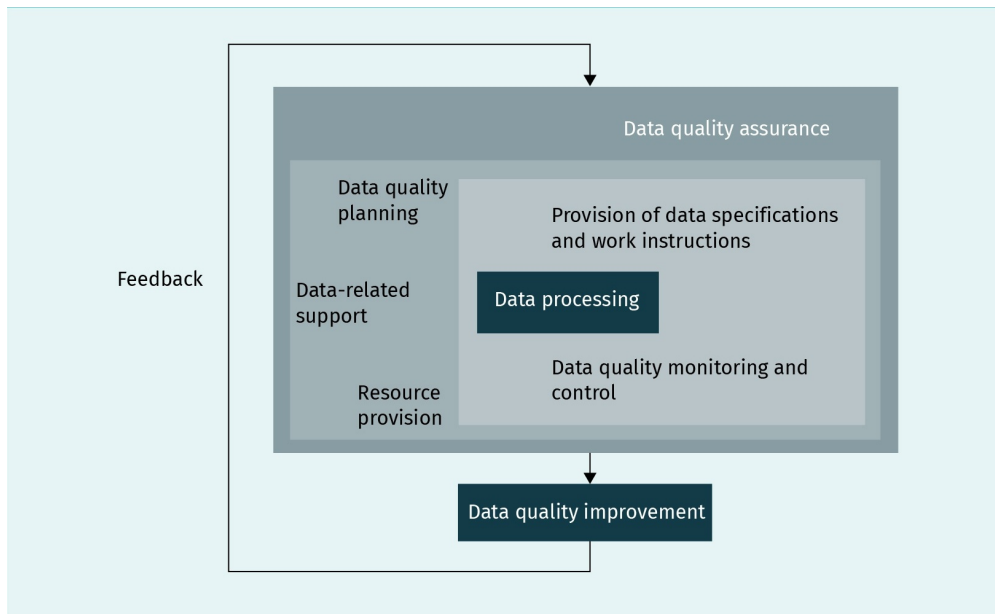


Source: Niklas Wietreck (2021), based on King and Schwarzenbach (2020).

Capability level five

The high-end capability level aims to create an understanding of how to improve the data quality based on a variety of root cause analyses. The identification of sources that create errors is the focus of this stage, while enabling the organization as a whole to raise awareness and become more data-driven.

Figure 6: Capability Level Five of Data Quality Management

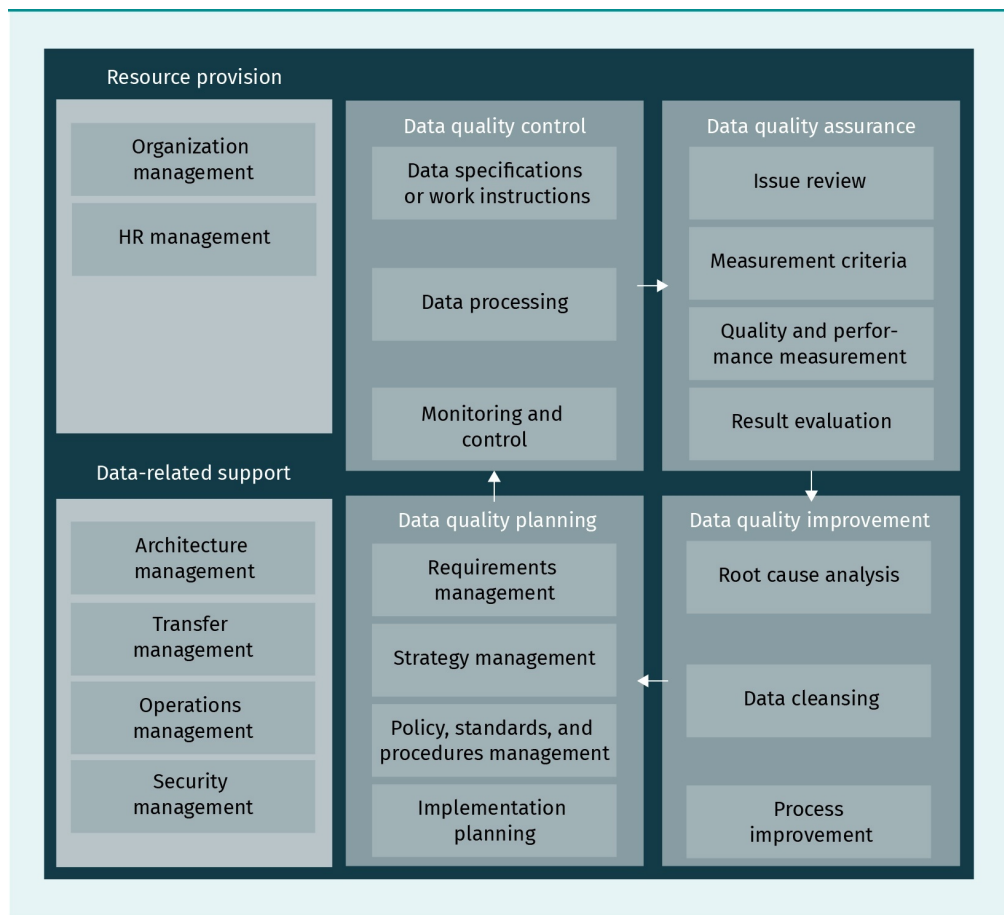


Source: Niklas Wietreck (2021), based on King and Schwarzenbach (2020).

ISO 8000-61

To give a practical guideline of how to improve the capability level of an organization, the ISO 8000-61 norm was developed as a holistic standard for data quality management. It follows the progressive improvements approach, using the “plan, do, check, act” steps. Each process in one of the five capability levels is also placed in the ISO 8000-61 process model. These steps can be formally described in four core and two high-level support processes. Data quality control, planning, assurance, and improvement are the core processes, while data-related support and resource provision have a supportive role (King & Schwarzenbach, 2020).

Figure 7: ISO 8000-61 Processes



Source: Niklas Wietreck (2021), based on King and Schwarzenbach (2020).

Data Quality Control

Each process needs a control mechanism that checks whether the requirements for the data are met. This is especially crucial when data or systems are frequently updated.

Data processing

This is the only process for capability level one. It aims to “satisfy the information needs of identified processes operated by an identified set of users” (King & Schwarzenbach, 2020, p. 69). In basic terms, this process creates data objects that support the end users in their decision-making processes. Since technology advances rapidly, the standard does not address specific actions. The focus is on identifying areas where data processing takes place and how they could potentially influence the data quality.

Provision of data specifications

This process belongs to the second capability level. It delivers artifacts that contain specific work instructions and data specification standards. The development of the artifacts should be conducted in close cooperation with domain experts from the respective field to ensure that the process steps occur in the right order and that data attributes are labeled correctly.

Work instructions in a data process include the definition of an objective of the process, an execution plan, definition of ownership, definition of required resources, identification of interfaces to stakeholders, documentation requirements, and a list of threats and opportunities.

For data specifications, the standard mainly focuses on the **syntax, semantics, and pragmatic** considerations of the data. What is included in the data specifications depends on the use case and the end user. For some recipients, it might be enough to deliver high-level specifications, while others might require additional information.

Syntax, semantic, and pragmatic

The syntactic aspects focus on the correct format of the data; semantics focus on a consistent, common interpretation; and pragmatic considerations are related to the degree the data are useful for the intended recipients.

Data quality monitoring and control

This process is linked to the second capability level. The process aims to identify deviations between the expected and anticipated output. This results in a confident data foundation for the decision-making process. In this process, the methods, tools, techniques, and technologies are defined to test and validate the outcomes against the predefined data specifications. The goal is to standardize methods and tools across an organization. While it seems desirable to automate as much of the verification and monitoring as possible within a variety of computer processes, it is important to keep in mind that there will always be some manual effort included for auditing.

To make the monitoring process more visual, people in charge of data quality often introduce data quality dashboards to create awareness of the topic and instigate competition within the organization for the highest data quality rating. Through such a display of data quality, teams within an organization are encouraged to perform better and gain higher ranks in the dashboard's metrics or leaderboards. Those dashboards are especially useful to monitor progress over time and see whether the data management measures have an effect.

Data Quality Planning

In the data quality planning processes, the requirements are formally defined and agreed upon by the relevant stakeholders. This includes the general objective of the data quality management efforts, as well as the desired data quality maturity. This process can be divided into four individual processes.

Requirements management

The first of ten processes is associated with the third capability level. Its purpose is to create an understanding of the expectation and demands of the stakeholders. Ideally, this process delivers a set of specific requirements that address the needs of all stakeholders while keeping the general organizational objectives in mind. Since such a set can be extensive, prioritization and containment will help to focus on the requirements with the highest impact.

This process includes a lot of communication between the data users, data providers, and strategic management. Each requirement must be analyzed and set into a holistic organizational perspective. Especially in large organizations, this process can be extensive and will increase in complexity when the number of stakeholders increases. Therefore, some data teams have a dedicated requirements engineer whose main task is to gather the information from the different stakeholders and translate it into technical terms.

Data quality strategy management

As part of capability level three, data quality strategy management encourages organizational leaders to create awareness and commitment to data quality. By delivering a long-term roadmap, they understand how the organization will progressively increase their data quality capabilities. This roadmap is partly based on the requirements collected in the previous process.

In some organizations, especially those with a digital business case, data are the key to success, and are understood as such. However, other organizations need to be inspired to execute data quality measures. Therefore, an accessible road map is required to get the relevant individuals on board to drive future activities.

Data quality policy, standards, and procedures management

This process is also part of capability level three, and its purpose is to create a set of specific activities that can be carried out repeatedly to create a common understanding of data quality management and reduce costs.

Outcomes of this process are guidelines and data specifications on how to act in specific scenarios. While on capability level two, the individual processes were the focus, a more holistic approach is taken on level three. Individual processes are consolidated as much as possible without losing any knowledge. The organization as a whole can then profit from pre-existing data processes. Also, the comparability of data from different providers can be improved.

This process reduces the administrative labor since redundant activities are avoided. For example, a cleaning process must be maintained in only one place where several teams can access the output, so the data cleaning process does not have to be done in every single team.

Data quality implementation planning

As the fourth process associated with capability level three, the data quality implementation planning gives an outline of how to deliver the knowledge created in the data quality policy, standards, and procedures management process across the entire organization. Here, it is identified how guidelines are deployed and who is responsible. A detailed schedule to carry out those activities is created.

The outcomes of this process can monitor the anticipated implementation activities. Whether defined resource requirements and the timeframe are met can be checked. Therefore, this process must be adjusted to the individual use case so that the ISO 8000-61 model is still followed.

Data Quality Assurance

Before starting the process to improve the data quality, it is important to understand the current data quality status.

Review of data quality issues

This is the first of four processes associated with capability level four. The review of data quality issues addresses the need to identify general data quality insufficiencies at the beginning of the improvement process. The outcome of this process is a set of data quality issues that can be prioritized and then handled during the data quality improvement step.

Since issues could occur on all levels, an appropriate prioritization should be developed. Prioritizing the impactful criteria and frequency of the issues is a common way to identify most critical issues. While the focus of data quality monitoring and control is on the individual processes, data quality assurance takes a more holistic approach by trying to find the root causes and measure the overall effects of data quality management.

Provision of measurement criteria

The focus of the second process at capability level four is the provision of appropriate measurement criteria. As an outcome, criteria are delivered that measure the data quality over time. After improvements have been applied, they are measured against this set of criteria to see the progress. This can identify trends or patterns, improving the analysis of root causes of data-related issues. Even though most criteria measure the outputs of several data processing steps, the required time to conduct them, and the required resources, should be measured to deliver a holistic view on the improvements. This includes the outcomes of the previous processes since data specifications themselves can be regarded as data.

Measurement of data quality and process performance

The purpose of the third process associated with capability level four is to deliver a framework to measure the performance of data quality, as well as the data processes themselves. The outcome is a report that displays the progression over time. Such displays cre-

ate awareness within the organization and can be used to measure the effectiveness of the executed improvements. It therefore focuses on laying out a quantitative basis based on the identified measurement criteria from the previous process.

In this process, every data object is assessed alongside every single aforementioned process. If, for example, a high impact error occurs, it is important to understand whether the staff possess the appropriate competences to tackle these issues, and, if not, the appropriate training can be scheduled.

Evaluation of measurement results

The last process of capability level four considers the implications that derive from the measurement of data quality and process performance step. As an outcome, a recommendation for improving several data quality processes is provided. Ultimately, this process creates action items, for example, in the form of a data quality management health checklist, based on the assessment of all previous processes. Although this is not the case when initially conducting the process, three further processes follow for capability level five.

Data Quality Improvement

This includes activities to define specific measures to improve data quality. These activities start with an in-depth analysis of the issues and, ultimately, implement a solution.

Root cause analysis and solution development

The first of three process associated with capability level five is the analysis of root causes and the development of solutions. It combines the in-depth understanding for the problem and the delivery of appropriate guidance to solve it. This guidance can include discrete steps to improve data transformation, advice on what technology is most suitable, or consultation on how to implement specific processes.

These processes go beyond identifying incorrect data values. When an error occurs, it takes people, processes, technologies, and organizational structures into account and identifies the root of the problem instead of treating symptoms. There are several approaches to root cause analysis, including **Ishikawa** (or fishbone) and the **5-Whys**. Root causes can vary from poor training to semantic errors in scripts and it should be solved at the lowest possible level.

Ishikawa

This provides a root cause analysis based on people, processes, data, and technology.

5-Whys

They describe an iterative process where the answer from one question creates the next question.

With the generated deep understanding of the causes, developed solutions aim to avoid the same or similar errors in the future. It is important to note that some solutions involve a long-term commitment to specific activities. If the staff, for example, are not properly educated, continuous training will solve the problem, but it will take time.

Data cleansing

Data cleansing is the second process associated with capability level five. It aims to reduce the number of false data entries to prevent the propagation of those errors down the data stream. A dataset can be considered clean when the dataset passes the requirement tests for accuracy, consistency, and coherence.

Data cleansing can occur in several scenarios throughout the whole organization and is therefore crucial for a successful data quality management strategy. While, ideally, only clean data derive in the database, this is impossible especially when data are migrated from external data sources. Even though there are commercial tools that support the data cleansing process, the definition of an error must usually be provided manually by a human being.

Additionally, it must be considered that a former “wrong” entry in the dataset might contain information that can be used in the future. Imagine, for example, a machine in a factory that creates a logfile data entry with the execution time of null because an error occurred. This null value can alter an analysis for the average execution time and will therefore be removed. While this is necessary for this analysis, a report on the occurrences of errors will need those null values.

Process improvement for data nonconformity prevention

The last process of capability level five considers all of the defined processes and aims to identify weaknesses and opportunities to improve the processes for the next iteration. The outputs are optimized processes for each capability level with a focus on the identification of underperforming processes.

This process evaluates the former processes from a holistic perspective and supports the improvement by uncovering performance gaps, as well as promoting discussion about further improvements. This process keeps the other processes aligned on the journey to higher data quality.

Data-Related Support

This group of activities can be watched from a technical perspective. A suitable data infrastructure must be in place before one can carry out data quality improvements. While the general architecture is key, processes concerning how to transfer, operate, and secure the data need to be defined.

Data architecture management

As the fifth process of capability level three, data architecture management aims to create a common perspective of the data. This includes the creation and formal definition of data objects, relationships, and attributes.

Data models
A data model can be conceptual, logical, or physical, and describes the data in different stages of their life cycle.

While the elements of technology involved will be identified (e.g., data stores, data input devices, data output devices, and data transfer mechanisms), the outcome of this process focuses on how these elements are linked and the impact they have on the overall system. There are attempts to visualize the architecture through a variety of **data models**, which will help the stakeholders to understand the architecture. While this is a valid approach, data models are more concerned with the representation of the data, while data architecture focuses on the underlying tools, platforms, and infrastructures.

This leads to the identification of sources of master data. This will help to understand which sources are linked to them, and estimate the impact of their update.

Data transfer management

The sixth process on capability level three is used to create an understanding of data flows across the organization. While this is partly displayed in the data architecture management process, data transfer management delivers a detailed definition of when and how data flows, and what mechanism a data flow triggers.

While data transfer can be initiated through a variety of actions (human or system), it aims to remove data transfer errors as much as possible while creating new data flows to make data available to a wider range of users. One example of an activity in this process is the creation of extraction transfer load (ETL) pipelines to address required data transformations or make data more accessible.

Data operations management

As the seventh part of capability level three, data operations management creates and maintains the infrastructure that is fundamental to data integrity within the organization. The outputs are technological solutions that are fit for purpose, but also scalable.

While the data operations are use case dependent, the activities can be grouped as follows: database management, user access management, capacity management, and maintenance (backing up databases, system optimization, etc.). Even though these activities usually only impact the user when a failure arises or performance is low, they are fundamental to a running system and crucial for meeting service level agreements.

Data security management

Another data-related support process, and the eighth process associated with capability level three, is data security management. Here, the purpose is to ensure the confidentiality, integrity, and availability of the data objects throughout the organization. Data security is a knowledge area, as defined in the Data Management Body of Knowledge (DAMA-DMBOK), since external malicious attacks, as well as internal individuals, can harm the integrity of the data (King & Schwarzenbach, 2020). While users might experience data security mainly as hindering because they might have to choose complex passwords, it adds value by trusting the data and ensuring that developed data products are properly

secured. There is an extensive set of activities that can be carried out within the data security management process. Most of them aim to be compliant with specific norms, such as the ISO/IEC 27000 standard (King & Schwarzenbach, 2020).

Resource Provision

This group of activities includes human factors in the processes. While planning the improvement process, the development of data-related skills needs to be taken into account.

Data quality organization management

The ninth process on capability level three aims to enable the organization as a whole to effectively and efficiently carry out data quality management practices. Outcomes are mostly organizational structures that promote good data quality on all levels of the organization. This includes the transformation of existing roles or the creation of new roles, which are engaged with containing a certain element of data quality management. This process can vary depending on the organization and the maturity of the data processes. For example, requirements and the actions of this process might differ between a digital company and a traditional manufacturer.

Human resource management

Human resource management is the tenth and final process at capability level three. Its purpose is to either fill roles with people that possess the required skills or to develop those skills within the existing staff. The outcome of this process is workers who possess the adequate knowledge to execute data quality management in line with the previously specified policies, standards, and procedures.

Challenges include the identification of the required skills and the level of each skill; it is not necessary for every person in a data role to possess a master's degree in data engineering. There are several frameworks, such as the **SFIAplus**, which support the identification of adequate skill levels for a given role. In this scenario, the skill level contains the technological abilities to execute certain processes and the domain knowledge in the respective field. To achieve high value outcome, it is suggested to keep a healthy balance between the two.

SFIAplus
This includes different levels of expertise for information-related roles.

Data-as-a-Service (DaaS)

In the past decade, an increasing number of organizations are moving toward a service-oriented architecture to optimize their internal processes and use their resources efficiently. One trend that has emerged over the past decades is the use of data-as-a-service (DaaS). DaaS is an information provision and distribution model that enables customers to access data. In basic terms, DaaS enhances on-demand delivery of data to the data customer (Søilen, 2016). Within the cloud environment, DaaS is closely related to **infrastructure-as-a-service** (IaaS) and **software-as-a-service** (SaaS), as it can be integrated into those two services.

Infrastructure-as-a-service

This provides hardware in virtual form to the customer. This can include storage, processing capacity, or a network.

Software-as-a-service

This provides on-demand software to a customer via the browser.

There are two categories of DaaS services: Read-only DaaS, and create, read, update, and delete (CRUD) DaaS. Read-only DaaS provides data based on existing data sources which can be accessed via a technology like application programming interface (API) where the customer only consumes the data. CRUD DaaS, however, can be linked to databases. CRUD therefore mainly provides data storage, which is closely related to IaaS (Truong & Dustdar, 2009).

The architecture behind DaaS is based on three different stages. In the gathering stage, data are retrieved from a variety of data sources and organized. Afterward, the process stage consolidates the different inputs into a reasonable format, such as a database. In the final stage (the publish stage), the data are provided to the end user via a web server where the finished data product can be accessed. Therefore, DaaS can be described as a ready-made intelligence package (Søilen, 2016).

An advantage of DaaS is the reduction of costs, since certain datasets are rented and stored outside the own organization. The acquired data are also easier to analyze since they were previously processed (Truong & Dustdar, 2009). Additionally, flexibility and availability are two features that make organizations decide to move towards a DaaS approach.

According to Sharma and Trivedi (2014), the reasons that organizations refuse to use DaaS practices are security and privacy. Since DaaS can make use of sensitive data, companies are concerned that competitors could access those data. Hackers who want to monetize on data leaks are also a constant threat; when DaaS is used as a storage solution, the data owner does not have full control of the data. Providers try to build trust by creating a variety of protocols and encryption, but according to Sharma and Trivedi (2014), companies are still concerned. Privacy concerns challenge organizations, meaning that they need to protect data and comply with privacy rules, especially with person-specific data (Sarkar, 2015). Since this highly depends on the use case, organizations need to find solutions that meet the demands of the data consumers while also preserving the privacy of the data (Sharma & Trivedi, 2014).

Data Virtualization

Data that are contained in different sources (databases, Application Programming Interface [API] web services, Extensible Markup Language [XML], etc.) can be abstracted to a level that they can be accessed without considering the storage technology, which is a process called data virtualization. It builds on the idea of data federation, in which data sources are distributed while the user gets access through a single interface. The query entered into the interface will then retrieve the information from a variety of systems. This requires that the returned data are consistent, regardless of the original structure of the data source. Data virtualization also makes it also possible to include transformations during the extraction process, which can lead to higher data quality. Especially in a multi-source environment where data sources differ from each other, a consistent format or data structure can be introduced. This creates a higher degree of flexibility where new data sources can be added or old ones can be removed. This can result in a service where the only data that are available are those actually requested by the users (Bologa & Bologa, 2011).

The process of data virtualization includes two phases: the identification of data sources and the application of the data model. In the first phase, the data sources and their related attributes are identified in order to define a data model. A data virtualization tool will help to create a mapping from the original data source to the abstracted level. In the second phase, the data model is applied to a variety of data sources in order to retrieve the queried information in real time. The language of the query can be defined within the data virtualization tool (Bologa & Bologa, 2011). There are a variety of providers for data virtualization tools, including IBM and Oracle.



SUMMARY

Data quality management requires individuals to lead change within an organization. The CDO sets the strategic vision, data stewards execute this vision, and data or process owners are in charge of specific procedures, but they must follow the same principles. The goals of data quality management are to implement a process-centric approach, attain progressive maturity, create explicit data specification, assign roles and responsibilities to end users, and not have data quality management led by the IT department. It is important to understand the current capabilities of the organization when considering these goals.

The capabilities can be divided into five levels. The first capability level focuses on general data processing, and the second level includes the formal specification of the data and the creation of guidelines. Components of quality monitoring and control are also included. The third level plans future data quality activities while providing data-related support and required resources. The fourth level manages quality assurance and an organization-wide standard. On the fifth level, data quality improvement takes place.

Each stage contains specific processes, and twenty processes can be put into six different groups: data quality planning, data quality control, data quality assurance, data quality improvement, data-related support, and resource provision. Processes range from management of requirements to data cleansing, and offer a variety of improvements to data quality.

Data-as-a-service architecture can reduce the required resources for the data infrastructure. By providing data users easy access to the data via a web server, on demand data products can be built to enhance decision-making.

Data virtualization models can abstract data to a level where the physical storage layer is no longer relevant to the data user. Modern data virtualization tools can be used to map and give users a single point of access to multiple data sources.

UNIT 3

DATA ACQUISITION

STUDY GOALS

On completion of this unit, you will be able to ...

- define web scraping, screen reading, and web spiders.
- implement basic web scraping technologies.
- acquire information from PDFs.
- manage the connection to data APIs.

3. DATA ACQUISITION

Introduction

Historians consider the current historical period the information age. A lot of things have changed since the first computers were developed, including the amount of data generated around the globe, which is increasing on a daily basis. According to the International Data Group (IDG), the data that will be generated in 2025 will reach up to 163 zettabytes. This is ten times more than the data that were generated in 2016 (Reinsel et al., 2017). Data accumulate in a high variety of use cases from different sectors. This can include data derived from social media, public transportation data, and clinical data from hospitals. In the past decade, organizations tried to gain value from data they collect themselves, or that were provided by specialized data providers.

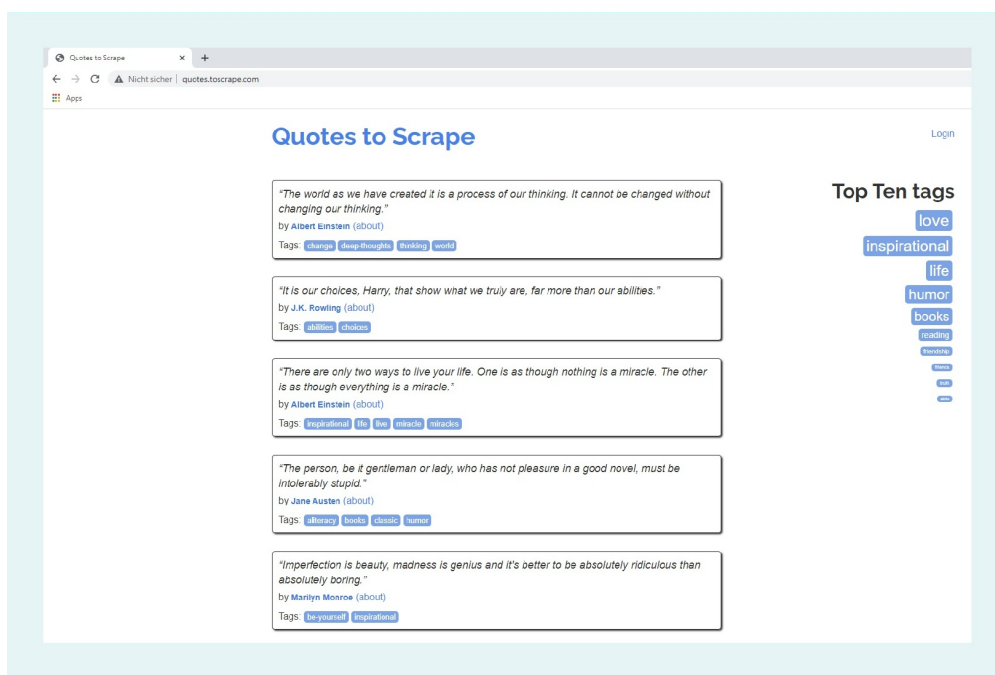
To make use of data, they must first be acquired. This unit will introduce two methods of data acquisition. The first section will focus on web scraping as a method for data extraction from the web. Since there are a variety of technologies on the web, different web scraping technologies will be presented and their uses will be demonstrated using Python. Advanced methods of web scraping in form of web spiders will also be presented, along with legal considerations. In the second section, Python will be used to connect to a data application programming interface (API) to extract data from publicly available sources, and the different models will be presented.

3.1 Web Scraping

While search engine web crawlers are capturing the general structure and identification of keywords and topic groups, it is possible to create more use case specific web scrapers to extract certain data from a website. There are many interesting web scraping projects on the internet, ranging from price monitoring on Amazon to extracting information from news pages. Scrapers, especially small one-page crawlers, often use Python to extract the desired information. Python is used since it has a variety of libraries that support the developers in web scraping.

One of the oldest methods uses the `requests` library, which is a simple but powerful Hypertext Transfer Protocol (HTTP) library that accesses a website and returns the raw data (for example, the Hypertext Markup Language [HTML] code). The simplicity of the library is its biggest strength, and can be used to access application programming interfaces (APIs). `requests` only requires a Uniform Resource Locator (URL) and will return the content of the respective page. In the following example, the scraper extracts the entire content of the website, which displays quotes from different people on different pages, as shown in the figure below.

Figure 8: Scraping Example Webpage



Source: ScrapingHub (n.d.-a). Fair use.

Code

```
# load packages
import requests

# load a web page
page = requests.get('http://quotes.toscrape.com/')

# extract the page's content
contents = page.content
```

Since `requests` returns the raw information from the website, it is necessary to parse the data to make it easier to access certain information. The most common parsing libraries are `BeautifulSoup` (with a focus on HTML-documents) and `lxml` (with a focus on Extensible Markup Language [XML] documents).

With the advances of web technologies, especially the widespread use of JavaScript, new challenges for web scraping have emerged and solutions have been created. Information that is dynamically received from the server (e.g., when scrolling to a certain point of the website) is not properly captured from, for example, the `requests` library. In such scenarios, the library `selenium` can be used to mimic user behavior, which can include filling out forms, logging into an account, or selecting elements from a list. `selenium` uses a web driver that can be used to automate browsers.

Even though `selenium` will extract the requested information in most cases, there is a web scraping framework that helps to set up a systematic way to derive information from a website. This framework is implemented in the library `scrapy`. In the following, we will learn about the most useful and frequently used libraries and techniques for web scraping using Python.

Requests

As shown in the code example above, `requests` is able to extract the raw information, i.e., the HTML code, from the example website, which can be displayed using the method `content`. While this works for this example, it is sometimes necessary to check the `status_code` of the request, which helps to identify wrongly stated requests.

Code

```
>>> page.status_code
200
```

A status code 200 indicates a successful request. Other status codes include a bad request (400), unauthorized access (401), and the occurrence of an internal server error (500). There is a huge variety of different status codes listed in the request for comments (RFCs) standards provided by the Internet Engineering Task Force (**IETF**) (Internet Engineering Task Force, 2021).

IETF
The Internet Engineering Task Force develops high quality technical documentations. This supports people who design, use, and manage the internet.

BeautifulSoup4

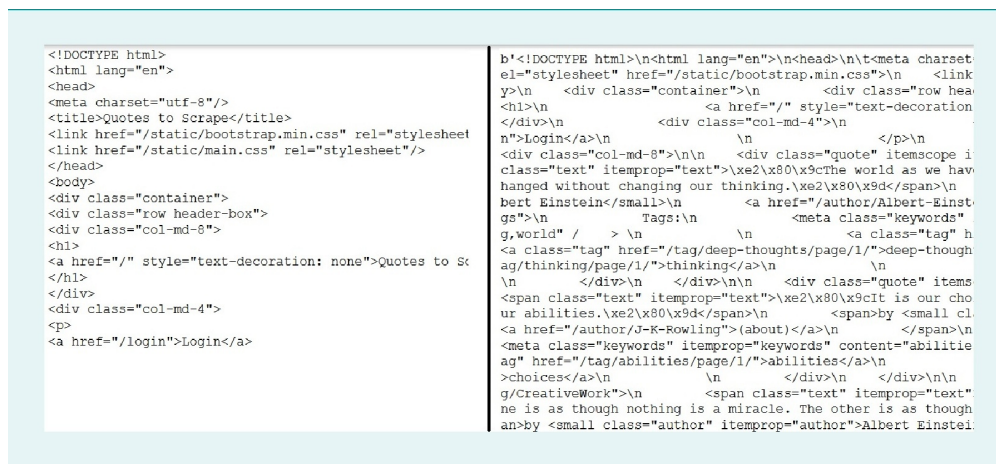
When the raw data are extracted, the information can be parsed. In this example, `BeautifulSoup` is used for this step. By importing the library and passing the raw page content, the website is parsed and can now be read as a proper HTML document.

Code

```
>>> from bs4 import BeautifulSoup
>>> soup = BeautifulSoup(page.content)
```

In the figure below, the outputs of `BeautifulSoup` (left) and `requests` (right) are displayed. An HTML-like structure can be identified from the output of `BeautifulSoup`, while the raw output contains several unnecessary elements.

Figure 9: Parsed HTML and Raw HTML



Source: Niklas Wietreck (2021).

BeautifulSoup allows us to access different tag objects and retrieve the contained information. The following piece of code returns the first tag object of the type link. In this link, the information about the location of the Cascading Style Sheets (CSS) stylesheet is stored.

Code

```
>>> soup.link
<link href="/static/bootstrap.min.css" rel="stylesheet"/>
```

To access a specific attribute that gives the tag its meaning and context, the object can be regarded as a dictionary that can be accessed via its key. The code now only returns the URL of the CSS file's location.

Code

```
>>> soup.link['href']
'/static/bootstrap.min.css'
```

While this example shows how to access the first tag, it is often necessary to search for a specific object within the parsed data. For this purpose, there are several search methods that can identify an object based on its tag, values, name, and text.

While a command like `soup.find("link")` will return the same first occurrence of the link tag as `soup.link`, `find()` lets us pass additional settings. The first quote for example, can be accessed by finding the tag with the attribute `class` and the value "quote". As a result, the complete tag is returned from the data.

Figure 10: BeautifulSoup Code

```
<div class="quote" itemscope="" itemtype="http://schema.org/CreativeWork">
  <span class="text" itemprop="text">The world as we have created it is a
    process of our thinking. It cannot be changed without
    changing our thinking."
  </span>
  <span>by <small class="author" itemprop="author">Albert Einstein</small>
    <a href="/author/Albert-Einstein">(about)</a>
  </span>
  <div class="tags"> Tags: <meta class="keywords" content="change, deep-thoughts, thinking, world" itemprop="keywords"/>
    <a class="tag" href="/tag/change/page/1/">change</a>
    <a class="tag" href="/tag/deep-thoughts/page/1/">deep-thoughts</a>
    <a class="tag" href="/tag/thinking/page/1/">thinking</a>
    <a class="tag" href="/tag/world/page/1/">world</a>
  </div>
</div>
```

Source: Niklas Wietreck (2021).

When searching for a string, it is important to keep in mind that `find()` is case sensitive. `Soup.find(class_ = "Text")`, for example, would return `None`. Regular expressions can be used to handle this.

To access specific elements of the found tag, it is possible to use the aforementioned methods. The following code will retrieve the value of the `href` attribute of the first tag within the tag with a `class_` named `quote`. The link to the author's information is returned.

Code

```
>>> soup.find(class_ = "quote").a["href"]
'/author/Albert-Einstein'
```

So far, the example returned the first occurrence of the searched object. Since the goal is to retrieve all quotes and their respective authors, it is necessary to retrieve all occurrences of an object. In `BeautifulSoup4`, the `find_all()` method can be used to do this. `soup.find_all(class_ = "quote")` returns a list of all tags with the `class_` named `quote`.

We can also use `find_all()` to combine search strings and, for example, find all attributes `class_` with the name `text` or `author`. This will return a list that contains all tags with the defined class name.

```
>>> soup.find_all(class_ = ["text", "author"])
```

The output is difficult to handle since it contains all tag information and is just a long list of all quotes and authors without any proper structure. One way to clean this is to iterate every quote element and find the elements of interest. The method `getText()` can help to retrieve the text of the tag. To make the handling easier, the information can be stored in a list which makes it easy to read into, for example, a **pandas DataFrame**.

pandas DataFrame
The `DataFrame` creates a table format based on multi-dimensional arrays.

Code

```
>>> quoteList = []
>>> for e in soup.find_all(class_ = "quote"):
    author = e.find(class_ = "author").getText()
    text = e.find(class_ = "text").getText()
    quoteList.append([author, text])
```

So far, only the first page of the website has been crawled. Since we want to extract the quotes and authors from all pages, we must find a way to iterate through all existing pages. In the URL, the page number is added at the end. Page one can be viewed when accessing the link ending `/page/1/` (Scrapinghub, n.d.-a), while page two simply uses the same link ending `/page/2/` (Scrapinghub, n.d.-b). The easiest way to crawl this would be iterating through all pages using a while loop and using a count to switch between the pages. The following code snippet accesses the first five pages and retrieves the information as shown in the previous example. After extracting the information from one page, the count is increased and the next page is crawled. The output will be a list with 50 elements, each containing a name and a quote.

Code

```
# crawl multiple pages of a URL and extract authors and
# quotes
url = "https://quotes.toscrape.com/page/"
quoteList = []

# iterate over all pages of the URL
for i in range(0,4):

    cur_url = "http://quotes.toscrape.com/page/" + str(i)
    page = requests.get(cur_url)

    # extract and parse the pages content
    contents = page.content
    soup = BeautifulSoup(page.content)

    # iterate over all tags of class 'quote'
    for e in soup.find_all(class_ = "quote"):

        # extract author's names and quotes
        author = e.find(class_ = "author").getText()
        text = e.find(class_ = "text").getText()
        quoteList.append([author, text])
```

To see how many quotes were extracted, we use the following line of code:

Code

```
print(len(quoteList))
# console output: 30
```

While the data are now properly extracted from the website, the challenge is deciding how many pages should be crawled. Depending on how the website was developed, there are a variety of ways to indicate that all pages have been crawled. The easiest scenario is when the server returns a status code with an error 404, which indicates that the page was not found. With a simple `if` statement, this error can be caught and the while loop can be terminated. This is a robust and flexible solution since it responds directly to the status code of the server.

In another scenario, the last page is indicated through a pagination element, which allows the user to get from one page to another by clicking on the specific page number or a next button. Then, the last element can be accessed through the respective tag and the highest number can be retrieved. This requires the existence of a pagination element, which can be challenging, especially with modern JavaScript-based sites.

Another approach is checking for the existence of specific content on the page. In this example, all pages have a “Next” button. If this button is not present, the last page has been reached. Therefore, `find` can be used to search for the text “Next,” and if the returned value is “None,” the while loop will be terminated. For the count check of the while loop, an appropriate high number has to be selected. In this scenario, 50 was chosen. This is a simple solution, and it is easy to implement, although it can break easily if the content of the web page changes (which occurs quite frequently). It can also be challenging to identify an object that is a proper candidate for such an approach since there must be a good understanding of the overall page structure. In this example, we use the text “Next” with an additional space at the end in the `find` statement. It is important to search for the object as specifically as possible in order to reduce the number of results and find the piece we are actually looking for.

Code

```
# crawl multiple pages of an URL which contain a
# 'Next' button
url = "https://quotes.toscrape.com/page/"
quoteList = []

# iterate over all pages of the URL
for i in range(0, 49):

    cur_url = "http://quotes.toscrape.com/page/" + str(i)
    page = requests.get(cur_url)

    # extract and parse the pages content
    contents = page.content
    soup = BeautifulSoup(page.content)

    # break out of the loop if there is no 'Next' button
    if soup.find(text = "Next ") == None:
        break

    # iterate over all tags of class 'quote'
```

```

for e in soup.find_all(class_ = "quote"):
    author = e.find(class_ ="author").getText()
    text = e.find(class_ = "text").getText()
    quoteList.append([author, text])

```

Again, we check the number of extracted quotes.

Code

```

print(len(quoteList))
# console output: 90

```

The search including the trailing whitespace is potentially harmful as it could be ambiguous and susceptible to web page changes. To handle such input, regular expressions can be used.

Regular Expressions

When data from a scraped page do not follow a strict pattern, regular expressions come in handy. They check whether a string contains a set of specified rules that can help to find and modify certain elements. The syntax used in regular expressions contains few commands, but longer expressions built from these simple rules can be difficult to write and read. In regular expressions, there is a group of metacharacters, which are described in the table below.

Table 1: Regular Expressions Metacharacters

Regex	Description	Example
[]	This specifies a character class that contains a set of characters you wish to match.	[abc] will match any characters a, b, or c. [a-c] does the same. Only lowercase letters would match.
^	This matches characters that start with the symbol.	^a will match any character that starts with a lower case a.
.	This matches anything except a newline character.	.at matches any single character before at, for example, cat.
\$	This matches if this is the end of a string.	cat\$ matches cat, but not catfish.
{}	This is used to specify quantifiers that describe the amount of previously defined characters that will be matched.	876-[0-9]{3}-[0-9]{4} matches a phone number in a specific format (the leading numbers 876, followed by a dash). Then, three numbers between zero and nine will follow. After another dash, four more numbers between zero and nine will follow.
	This is interpreted as “or” and stops at the first match.	.bc .cd would match both abc and bcd. The first match will be handled.
()	This is used to group parts of the regular expression that can be used to apply multiple operations on one string.	O(range ld) will match Orange and Old.

Regex	Description	Example
?	This matches zero or one occurrences of the previous character.	<i>ab?</i> would match <i>a, ab</i> but not <i>abb</i> .
*	This matches zero or more occurrences of the previous character.	<i>ab*</i> would match <i>a, ab, abbb</i> , but not <i>cb</i> .
\w	This matches an American Standard Code for Information Interchange (ASCII) character, such as a letter, digit, or underscore.	<i>\w-\w</i> would match <i>a-b</i> .
\d	This matches a digit character between zero and nine.	<i>word_\d\d</i> would match <i>word_27</i> .
+	This matches one or more occurrences of the previous character.	<i>\w-\w+</i> would match <i>a-bbab</i>
\	This is used to escape meta characters. Escaping means that a character, such as [(a reserved character for regex), can be searched for in a string.	If a square bracket [needs to be matched in a string, it can be written as \[.

Source: Niklas Wietreck (2021).

Based on these metacharacters, a variety of string searches can be conducted. After importing the `re` module for handling regular expressions, the following code checks whether a string starts with the characters “The,” has zero or more of any character and then ends with the characters “Germany.” The substring, or, in this case, the whole string, is returned from this search.

Code

```
# load packages
import re

# generate sample data
txt = "The rain in Germany"

# apply RegEx to find patterns in the data
re.findall("^The.*Germany$", txt)
# console output: ['The rain in Germany']
```

The method `findall()` is used to find all occurrences within the string. In the following example, it will return all occurrences of the substring “ai” from the text. In this case this matches “ai” in the words “rain” and in “Spain.”

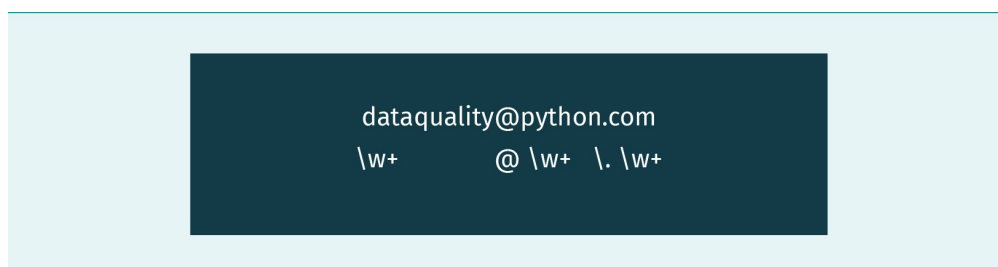
Code

```
txt = "The rain in Spain"
re.findall("ai", txt)
# console output: ['ai', 'ai']
```

There are a variety of other functions within the regular expression module. `search` will return a match object, which indicates whether there is a match within the string. `split` returns a list where the string has been split at each match. `sub` replaces one or many matches with another string.

In the context of a website crawler, a regular expression is commonly used to extract email addresses from website. The example below shows how an email address could be written using regex.

Figure 11: Email Regex Example



Source: Niklas Wietreck (2021).

The following code returns the first email address from an example text. To do this, it uses the special sequence `\w` to select any alphanumeric value that appears at least once, followed by the `@` sign. Afterwards, another alphanumeric sequence is searched for before the dot is escaped. At the end, the email address contains another alphanumeric sequence. In order to use this the regular expression, we need to “compile” it. We can then use it to find the matching characters within a text using `soup.find()`.

Code

```
# create sample HTML
email_example = """<br/>
    <div>
        This is an example HTML document to showcase
        how email addresses can be retrieved using regex
    </div>
    tutor@iu.org
    <div>student@iu.org</div>
    <span>professor@iu.org</span>
    """

# parse the HTML
soup = BeautifulSoup(email_example, "lxml")

# compile a RegEx
regex = re.compile("\w+@\w+\.\w+")

# use the RegEx to extract email addresses from
```

```
# the HTML
print(soup.find(text = regex))
# console output: tutor@iu.org
```

Screen Scrapers and Spiders

BeautifulSoup parses HTML data, which can be retrieved from a webpage as a request object. Advances in web technologies (e.g., post-page-load code and interactive elements) require new forms of web scrapers, which are also capable of imitating user behavior. These types of web scrapers are also called screen readers. One common package to use for screen reading is `selenium`, which basically opens a browser and interacts with the webpage in the way a user would. Methods such as `send_keys()` or `click()` allow the user, for example, to interact with forms. Even scrolling through a page by elements or a defined number of pixels is possible. The following code will open a new tab in Firefox and will open the example webpage. After maximizing the window size using the `maximize_window()` method, the script uses the `execute_script()` method to scroll to the pixel 2000 on the y-axis. `selenium` depends on third party software, such as a valid driver for your browser (e.g., the `geckodriver` for Firefox). Make sure that you have a suitable driver on your machine and set the `PATH` to the binary of this driver in the system's variables to execute the following example.

Code

```
# load packages
from selenium import webdriver

# specify a webdriver
browser = webdriver.Firefox()

# use the browser to go to a web page
browser.get('http://quotes.toscrape.com/')

# maximize the window
browser.maximize_window()

# use a simple script to scroll down 2000 pixels
browser.execute_script("window.scrollTo(0, 2000)")

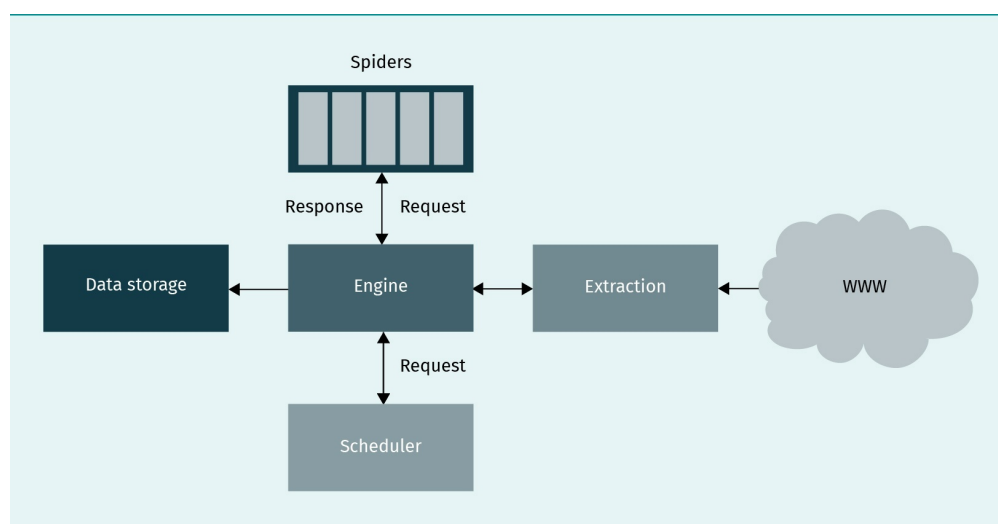
# locate the next button
next_button = browser.\
    find_element_by_partial_link_text('Next')

# click the next button
next_button.click()
```

Web spiders take the topic of web scraping further by delivering a framework that captures data from multiple pages on a site. They are great for retrieving information across a whole domain. In the first step, spiders often help to understand the structure of a site. This can then help to identify new sources of data and efficiently retrieve them in a second step.

A commonly used package for spiders is Scrapy. Scrapy offers a variety of features that support connecting, exporting, and structuring multiple scrapers. Additionally, there are methods and classes that help to handle unsuccessful queries. Scrapy logs errors out-of-the-box, which helps build the scraper in the first place, since debugging can be performed in an efficient manner. This will then lead to more robust scrapers. When building a spider, some more initial organization is required than with scrapers like BeautifulSoup. In Scrapy, there are five different types of spider. The “Spider” class is used to parse a defined set of sites and pages. “CrawlSpider” takes a set of regex rules to identify high value pages. “XMLFeed” spider pulls content from nodes like RSS feeds. The “csvFeed” spider is optimized to extract CSV feeds or row-based content. Finally, the “SiteMap” spider extracts the sitemaps of a given list of domains.

Figure 12: The Scrapy Architecture



Source: Niklas Wietreck (2021), based on Scrapy (n.d.).

The Scrapy architecture can be described as follows. In an initial request, the engine collects a spider from the available set. The engine uses the scheduler to plan the execution. Once the scheduler confirms, the engine sends the request to the extraction step, where the information is retrieved and a response is generated. The response is then sent to the spiders for further processing. As a response, the scraped items are returned and pushed toward the data storage, while new requests are handed over to the engine and scheduler. The following example sets up a simple Scrapy environment. In a first step, a Scrapy project needs to be created. This can be done in the prompt using the following command which should be executed from the terminal (bash, powershell, etc.). This will create a directory with the project name.

```
scrapy startproject quotes
```

Within the directory, there is a sub-directory, “./quotes/quotes/spiders/,” that will store the scrapers created in the next step. This scraper contains a class that names the spider and lists all the pages that need to be scraped. After the spider is instantiated, the “div” container with the name “quote” is selected. From there, the elements for “text” and “author” are extracted. We call this short Python script “Spider1.py.”

Code

```
import scrapy

class QuotesSpider(scrapy.Spider):

    # name the spider
    name = "quotes"

    # specify the URLs to crawl
    start_urls = [
        'http://quotes.toscrape.com/page/5/',
        'http://quotes.toscrape.com/page/6/'
        'http://quotes.toscrape.com/page/7/'
    ]

    # define a parser
    def parse(self, response):

        # get the divs with the tag 'quote'
        quotes = response.css('div.quote')

        # iterate over each quote
        for quote in quotes:

            # extract the text and author of the quote
            yield {
                'text':quote.css('.text::text').get(), \
                'author':quote.css('.author::text').get()
            }
```

From the Scrapy project directory “./quotes/,” the spider can be run using the command line. This will return all quotes from pages five to seven and output the crawls to the JSON-file “quotes.json.” By extending the “urls” list, more complex spiders can be built. This Scrapy project can also be deployed to a server and executed on a regular basis.

```
scrapy crawl quotes -O quotes.json
```

Reading Data from XML

The Extensible Markup Language (XML) is used to represent hierarchical data, which are readable for machines and humans. For the following example, the countries dataset from the XML documentation will be used. It can be downloaded as an XML file online (The Python Software Foundation, n.d.). When opening the data in an editor, the following structure will be shown.

Figure 13: XML Example Countries

```
<?xml version="1.0"?>
<data>
  <country name="Liechtenstein">
    <rank>1</rank>
    <year>2008</year>
    <gdppc>141100</gdppc>
    <neighbor name="Austria" direction="E"/>
    <neighbor name="Switzerland" direction="W"/>
  </country>
  <country name="Singapore">
    <rank>4</rank>
    <year>2011</year>
    <gdppc>59900</gdppc>
    <neighbor name="Malaysia" direction="N"/>
  </country>
  <country name="Panama">
    <rank>68</rank>
    <year>2011</year>
    <gdppc>13600</gdppc>
    <neighbor name="Costa Rica" direction="W"/>
    <neighbor name="Colombia" direction="E"/>
  </country>
</data>
```

Source: Niklas Wietreck (2021).

In the first step, the following code will import the XML ElementTree class. After parsing the data, which are stored as an XML file, the `getroot()` method will get the root of the dataset (in this case, `data`). From there, the tags and attributes of the different child objects are displayed using a for loop. To access a specific value, an array notation can be used. The script will return the name of every single country in the dataset and the value of the of second element in the first child object.

Code

```
# load packages
import xml.etree.ElementTree as ET
```

```

# read the data
tree = ET.parse('data.xml')

# get the root of the XML tree
root = tree.getroot()

# iterate over each child of the root
for child in root:

    # print the child tag and attribute
    print(child.tag, child.attrib)

# console output:
# country {'name': 'Liechtenstein'}
# country {'name': 'Singapore'}
# country {'name': 'Panama'}

```

Legal Aspects of Web Scraping

The legality of web scraping is considered a grey area. There are a variety of aspects that have to be considered when talking about the legal aspects of web scraping. Since there is no legislation that defines the framework of web scraping, the legal discussion is driven by a set of related and fundamental legal theories. These legal theories are highly dependant on local law. Copyright and privacy, for example, are defined differently depending on the country or state in which the developer resides (Krotov & Silva, 2018).

A few aspects that must be considered are the terms of use, copyright, damage of the website, and purpose of the scraping. Website owners can prohibit the systematic extraction of information from their website by stating this in their terms of use. To make this legally prosecutable, the website owner needs to implement a comply mechanism that the user has to go through (e.g., a check box). Using information that is owned by the website owner and explicitly marked as copyrighted could result in a copyright infringement case. In this context, it is difficult to define what falls under the copyright. User-generated content, for example, is not necessarily owned by the website owner. Another example of content that is not under copyright is ideas that have been altered to a certain degree (change in representation or form). Copyrighted material can still be used on a limited scale when following the “fair use” principle.

Another aspect to consider is the purpose of the web scraper. If it is used to access confidential or protected data with the aim to redistribute them, in the US, it falls under the Computer Fraud and Abuse Act (Krotov & Silva, 2018). Victims of such actions can take legal measures to protect themselves. If web scraping overloads or damages the web server, the responsible persons can be prosecuted. It is critical that the damage can be easily proven in order to receive financial compensation (Krotov & Silva, 2018).

Generally, it is important to execute web scraping to a reasonable extent. This includes making sure a server is not overloaded (limiting the amounts of requests), consulting only trustworthy web pages, and only extracting information for ethical and morally supportable causes. Another option is informing the website owner about the web scraping activities and asking for permission to execute them on a regular basis.

Acquire Data from PDFs

Even though it is considered bad practice, data are often delivered in form of a Portable Document Format (PDF) file. PDF files are often hard to parse since they can come in unpredictable formats and are not structured as the reader would expect. There are a variety of approaches to handle PDFs, including converting them to a text. Since there is a number of approaches, there are various Python packages that try to parse PDFs in the most convenient way. For the following example, the package `pdfminer`, which is commonly used to extract text from a PDF, will be used. After installing and importing `pdfminer`, the PDF file can be imported and parsed. For this example, the World Water Development report published by the United Nations is used (The United Nations, 2021). This report can also be downloaded with the following lines of code:

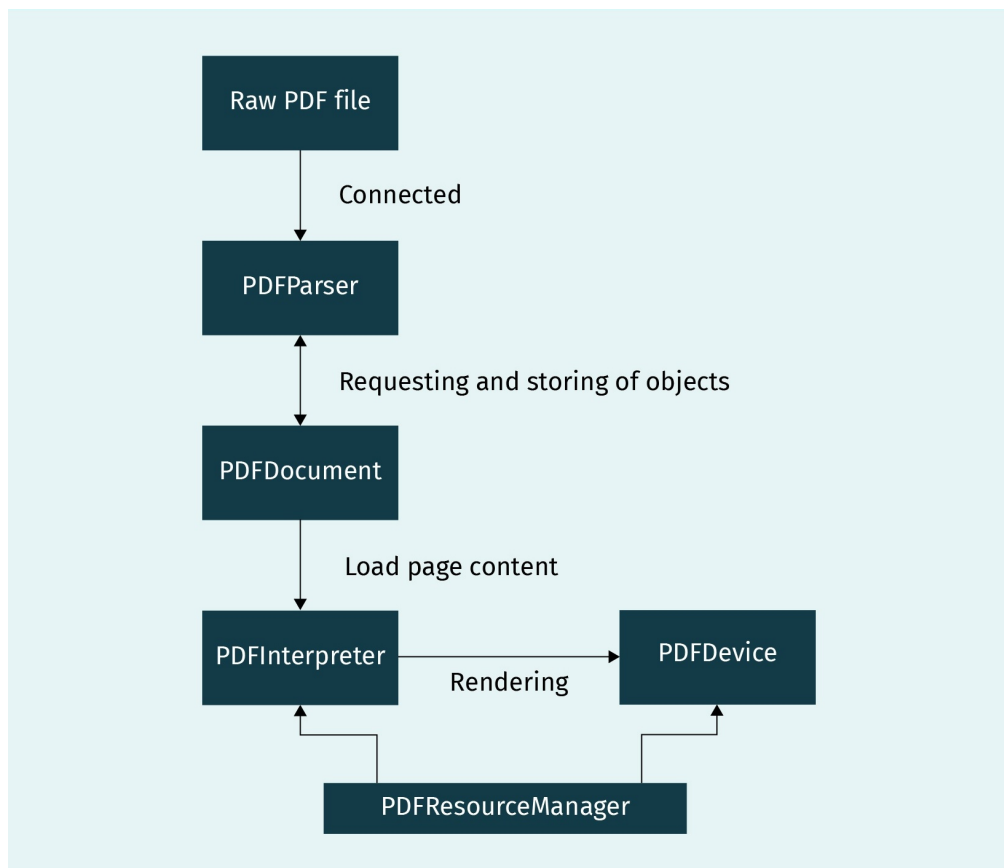
Code

```
# load packages
import urllib.request

# load a sample PDF report
url = "https://unesdoc.unesco.org/in/rest/"
url += "annotationSVC/DownloadWatermarkedAttachment/"
url += "attach_import_db06f7c4-b33f"
url += "-4833-be56-bbf54afdee3f?"
url += "_=375724eng.pdf"
urllib.request.urlretrieve(url, "UN_water_report.pdf")
```

After the file is opened, the “PDFParser” class is used to fetch the file before it is stored in the “PDFDocument” class. The “PDFResourceManager” is used to efficiently store shared resources, such as fonts or images. The “PDFPageInterpreter” processes the explicit contents, while the “PDFDevice” class translates them to the desired output.

Figure 14: PDF Miner Process



Source: Niklas Wietreck (2021), based on Shinyama (n.d.).

Now, the script iterates through every page of a PDF document and extracts the text. To retrieve the text, the `getValue()` method is used. While it is possible to extract the text of only specific pages, the challenge is finding the right elements in the PDF and structuring them properly.

Code

```
file_path = 'UN_water_report.pdf'
output_string = StringIO()
with open(file_path, 'rb') as in_file:
    parser = PDFParser(in_file)
    doc = PDFDocument(parser)
    rsrcmgr = PDFResourceManager()
    device = TextConverter(rsrcmgr, output_string, \
        laparams=LAParams())
    interpreter = PDFPageInterpreter(rsrcmgr, device)
    for page in PDFPage.create_pages(doc):
        interpreter.process_page(page)
```

We can print the text of the whole document with the following line of code:

```
print(output_string.getvalue())
```

To extract only the foreword, the for loop could be refactored using the enumerate operator in combination with an if clause. The pages defined in the pages variable will then limit the number of pages that are extracted.

Code

```
pages = [7,8]
output_string_2 = StringIO()
with open(file_path, 'rb') as in_file:
    parser = PDFParser(in_file)
    doc = PDFDocument(parser)
    rsrcmgr = PDFResourceManager()
    device = TextConverter(rsrcmgr, output_string_2, \
        laparams=LAParams())
    interpreter = PDFPageInterpreter(rsrcmgr, device)
    page_enum = enumerate(PDFPage.create_pages(doc))
    for pagenumber, page in page_enum:
        if pagenumber in pages:
            interpreter.process_page(page)
        else:
            continue
```

It is also possible to extract pictures or tables from PDFs. Since pdfminer mainly focuses on text extraction, other packages (e.g., poppler or PyPDF2) can be used for these purposes. However, as previously mentioned, since PDFs do not arrive in properly structured format, there is always the demand for additional cleaning efforts. This can include removing page headers and footers, encoding special characters, or cleaning when a text cannot properly be separated from a picture. A simpler way to work with data is to receive them in a proper format. One way to access pre-structured data is by using a data application programming interface (API).

In many cases, we are interested in tabular data, which are reported in PDF files. These data are not able to be processed directly from an unstructured format like a PDF. However, there are ways to access numeric data in a structured way. In the next example, we learn how we can extract tabular numeric data from a PDF file. In the UN water report from the previous example, there is a table that shows the estimated walking times to sanitation sites in different countries in South East Asia (The United Nations, 2021). We will extract this table from page 77 to work with it in a structured way in Python using the `tabula` package. This package can be installed with the command “`pip install tabula-py`” (make sure to put the “-py” behind the package name). This package requires Java to be installed. After we have installed `tabula` in our environment, we can load it and use it to extract the table.

Code

```
# extract tables from the PDF
import tabula

# load tables from the PDF
tab = tabula.read_pdf('UN_water_report.pdf', \
    pages = '77')

# show the data from the table as a pandas dataframe
tab[0]

# console output:
#      Country Rural Urban Unnamed: 0
# 0      Cambodia  10.0   3.0      NaN
# 1      Indonesia   3.5   7.5      NaN
# 2      Lao PDR   14.0  10.0      NaN
# 3      Philippines 20.0   9.0      NaN
# 4      Vietnam   6.0  15.0      NaN
# 5 Yunnan (China)  6.0   3.0      NaN
```

From this point forward, we can use the numeric data and analyze them in a systematic and structured way. Before we do that, we should, of course, drop the last column, which does not contain any information.

3.2 Data APIs

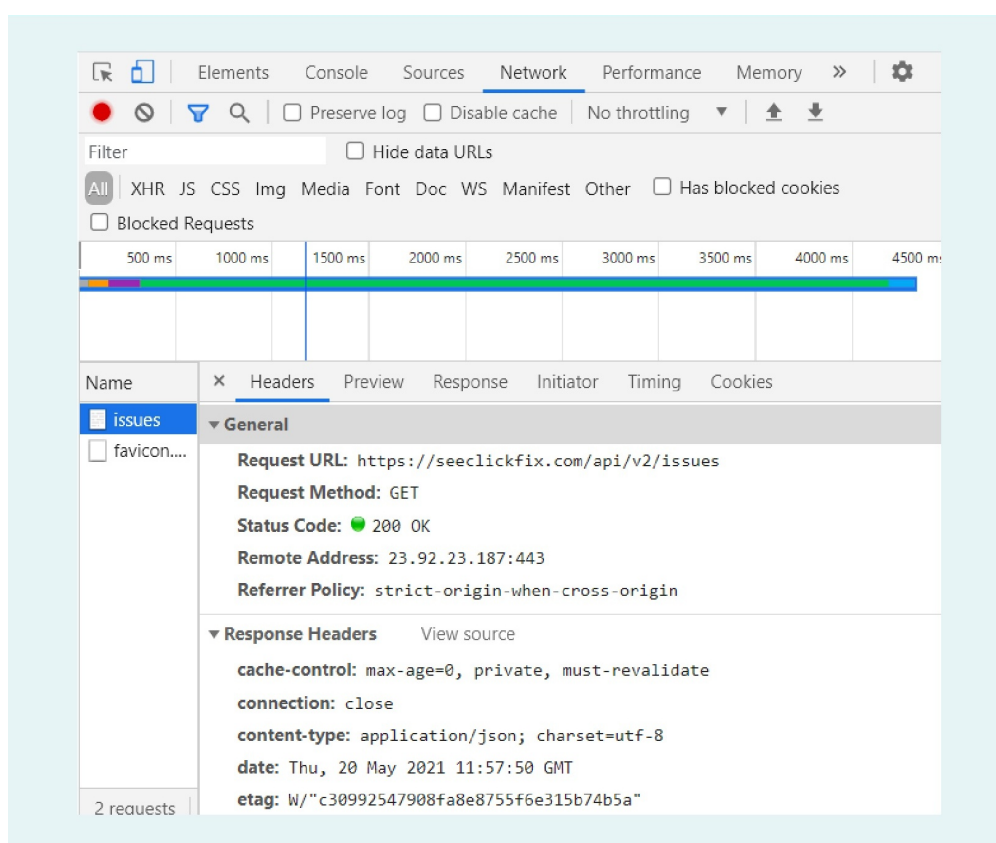
The amount of available data steadily increases. While it was easy in earlier time to access data via static files, nowadays, massive web applications and data archives (e.g., Wikipedia) use another approach. By using APIs, users can retrieve a small portion of the data. Nowadays, there are a variety of APIs that can be accessed for free and are marked as open data APIs. These data are often provided by governments, schools, companies, and online communities. The motivation behind offering free data depends on the data provider. While governments want to provide transparency and encourage researchers to conduct and develop useful applications, commercial organizations often offer a free basic API and a premium version of their product.

There are a variety of projects that collect available open data APIs. One list of publicly available APIs is the GitHub repository of “public-apis” (Github, n.d.-b). There, you can find the documentation and API endpoint specifications for APIs, ranging from public libraries to cryptocurrency market data. An API endpoint is used in this context to define the point where the resource is accessed, and they play a key role in managing user requests in the context of performance and access management. Three examples of APIs will be presented. The bandsintown API allows artists and agencies to make the information about artists and their events publicly available. The Nutritionix API gives access to the world

largest verified food nutrition database. Another interesting API, especially for book lovers, is the Penguin Publishing API, which gives access to information about books, authors, and events (Github, n.d.-b).

In the following examples, the “Seeclickfix” API will be used. This API gives us access to data that are collected by allowing residents of a city to report non-emergency neighborhood issues to the respective government. When communicating with an API, this is normally done via the HTTP protocol, which defines the request and response protocols between the client (e.g., web browser or script) and the server. The most popular type of request is the “get request,” which asks the server for certain data. The request that was sent from the client (in this case, the browser) to the server can be seen in the developer tools of the browser. In this example, which was executed with the Chrome browser, the information was requested (Seeclickfix, n.d.). Within the browser developer tools, the response is returned as shown in the picture below.

Figure 15: Request View in Browser



Source: Niklas Wietreck (2021).

When using Python, the “requests” module can be used to extract the desired information. Therefore, the “requests” module is imported and the API address is defined. The information can be pulled from the API using the `get()` method and is stored in the response object. By using the `json()` method on the response object, the data from the API are

returned in a JavaScript Object Notation (JSON) format, which contains all the data currently stored at the specific API endpoint. The documentation of the SeeClickFix API can help to identify and understand the available endpoints.

Code

```
# load packages
import requests
from pandas.io.json import json_normalize
import pandas as pd

# get data via RESTful API
api_url = "https://seeclickfix.com/api/v2/issues?"
response = requests.get(api_url)
response.json()

# console output:
{'issues': [{'id': 9957288, 'status': 'Open', 'summary': 'Parked Vehicle Concern'...
```

It is possible to hand over additional parameters to the `get()` request in order to obtain specific information from the API. In the following example, the parameters `after`, `before`, and `page` are set to narrow down the returned dataset. `after` and `before` allow us to set a specific time frame, while `page` limits the amount of data entries that are returned.

An ampersand is used to concatenate different parameters. A first analysis of the output has shown us that all issues are stored in a list that can be accessed by using the key “issues.” To access only this list, the “issues” key is accessed after the `json()` method is applied.

Code

```
# get data via RESTful API with specifications
api_url = "https://seeclickfix.com/api/v2/issues?"
api_url += "&after=2021-01-01T00:00:00"
api_url += "&before=2021-01-03T00:00:00"
api_url += "&page=1"
data = requests.get(api_url).json()["issues"]
print(data)

# console output:

[{'id': 9137070, 'status': 'Acknowledged', 'summary': 'Litter - Street Litter...
```

The data element now contains 20 entries, and each is a dictionary. Using the `json_normalize()` function from the `pandas` package, the data can be converted into a tabular `DataFrame` structure.

```
data_df = json_normalize(data)
```

RESTful versus Streaming APIs

APIs often come in two different forms: REST and streaming. Most available APIs are representational state transfer (REST) APIs and have been built to create a stable API architecture. As above, the requests package allows us to access the API and return the queried data. The data from the Seeclickfix API are extracted; they represent the current data that were made available. When querying the API, a whole dataset according to defined parameters is returned.

A streaming API, on the other hand, can be regarded as a real-time service and continuously returns the latest data, which are related to the defined query. In the context of social media, streaming APIs are used to build social listening tools. These tools collect information related to the defined parameters, and the data can then be stored over time and analyzed for trends.

API Rate Limits, Keys, and Tokens

APIs often have access rate limits. One reason for this is that API providers want to efficiently manage the amount of transferred data. If each user could send unlimited requests, the API could crash. Business aspects are another reason for API rate limits. While many APIs offer a specific number of requests for free, the user must pay for additional requests. In the Twitter API, for example, the user can send 900 requests within a 15-minute time frame (Twitter Developer Platform, n.d.). When more requests are required, the user can opt for different pricing models. Once the rate limit is reached, an error will be returned.

To access an API that is not open, keys and tokens are required. The API key normally identifies the application, while the token identifies the user. Additionally, there are API key secrets and token secrets, which, in a broader sense, act as passwords. In the example of the Twitter API, the keys and tokens can be created on the Twitter developer website. After the creation of an account on the Twitter developer platform, we can register a Twitter app (which is used as individual API endpoint), and the keys, tokens, and secrets are returned. This is how the credentials can look when the steps are followed. The following credentials are fictional, but they show the pattern of a real key. When going through this example on your own, you should change these strings according to your credentials. Also, keep in mind to not write credentials in plain text as shown here (use, for instance, system variables, a key vault, or certificate settings). This example is merely for educational purposes and it should never be done like this in a production scenario.

Code

```
API_KEY =  
'PmED9icCc6GsglKyORRuYi0w2'  
API_SECRET =  
'J8QAJDhFA6SHrmW8278wF3vyq8UGNa2rWBOUK8iZY30GcNda07'  
TOKEN_KEY =
```

```
'2407122708-uxlSaXGNsHVpdI6foqMppqQYsA0qqAnAaizZT1a'  
TOKEN_SECRET = '  
N1Q4B2Z1TafwwTwmLWLeFJxSbImjbaGB354XTgiuzYn9'
```

The following example shows how data are retrieved from the hashtag #climatechange. To access this, the Twitter search API endpoint can be used. This can be accessed using the following URL:

Code

```
url = 'https://api.twitter.com/1.1/search/'  
url += 'tweets.json?q=%climatechange'
```

One of the easiest ways to manage requests with authentication is the `oauth2` package. Since each request needs to be authenticated, it is most efficient to define a function for this. In the example below, the `oauth_req()` function takes the URL, access token, and secret in order to return the content. First, a consumer object is defined by handing over the API key and secret. In the second step, a token object is defined, and it takes the token key and secret. The combination of the consumer and token object results in a client object that can be used for authentication for the API. Using the `request()` method on the client object returns the response and content of the requested query.

Code

```
# load packages  
import oauth2  
import json  
from pandas.io.json import json_normalize def  
  
# define a function to connect to the URL  
# using authentication  
oauth_req(url, key, secret, http_method="GET", \  
post_body=b"", http_headers=None):  
  
# set consumer and token and use those to create  
# a client object  
consumer = oauth2.Consumer(key=API_KEY, \  
secret=API_SECRET)  
token = oauth2.Token(key=key, secret=secret)  
client = oauth2.Client(consumer, token)  
  
# access the API  
resp, content = client.request(url, \  
method=http_method, body=post_body, \  
headers=http_headers)  
  
# return the content of the API response  
return content
```

Once this function is defined, the URL can be called. A list of single status posts is returned. It contains the hashtag #climatechange, or parts of it.

Code

```
# call the API
data = oauth_req(url, TOKEN_KEY, TOKEN_SECRET)

# console output:
# b'{"statuses":[{"created_at":"Sat May 22 13:02:01 +0000 # 2021","id":1396088961030893568,"id_
# "1396088961030893568","text":"RT @UNUCPR: New article
# out now: There needs to be a reality check
# on#climate heading into #COP26, write @AdamDayNYC and
# Mayesha Alam.
# The\u2026","truncated":false,"entities":
# {"hashtags":[{"text":"climate","indices":[70,78]},
# {"text":"COP26","indices":[92,98]}],"symbols":[]...

# convert the resulting string to JSON
data_json = json.loads(data)

# convert response to a pandas DataFrame
data_df = json_normalize(data_json)
```

The API call, as defined above, has a limit, since it only collects 15 tweets. Using the Tweepy package, we can extract more information and state advanced queries. Tweepy is an open-source Python package that supports efficient access to the Twitter API. After installing and importing Tweepy, we authenticate the API as we did above.

Code

```
import tweepy
auth = tweepy.OAuthHandler(API_KEY, API_SECRET)
auth.set_access_token(TOKEN_KEY, TOKEN_SECRET)
api = tweepy.API(auth)
```

The `tweepy.API` object can take different arguments, which gives the developer the possibility to customize the request. Common parameters which are used to avoid rate limits are `retry_count`, `retry_delay`, and `wait_on_rate_limit`. The latter will wait until the rate limit has been reset so that additional requests can be posted. The following example does not define those parameters and will therefore use the default values. After defining the search query, Tweepy uses a cursor (which is commonly used in databases) to return an iterator on a per-item or per-page level. The additional argument “lang” limits the results to tweets in English. The for loop will then extract every item from the returned page.

Code

```
query = '#climatechange'  
cursor = tweepy.Cursor(api.search, q=query, lang="en")  
for page in cursor.pages():  
    for item in page:  
        print(item._json)
```

By iterating over the different pages, the information can be retrieved by accessing the `_json` key. The data can then be stored in a DataFrame or database. Depending on the query, the amount of data varies. It is important to take care of rate limits, but also to limit the request so only valuable information is attained.

Tweepy can also be used to access the streaming API endpoint of Twitter. To do so, we create a “Listener” class, which reacts to incoming tweets by printing them, and returns “True” as a flag for a successful process. After a “Listener” class that connects to the streaming API is defined, it can be used to extract the tweets according to a defined query. The query is used as a filter on the stream and will continuously return only relevant tweets.

Code

```
# load packages  
from tweepy.streaming import StreamListener  
from tweepy import OAuthHandler, Stream  
  
# create a listener class  
class Listener(StreamListener):  
    def on_data(self, data):  
        print(data)  
        return True  
  
# listen to a topic within the stream  
stream = Stream(auth, Listener())  
stream.filter(track = ["climatechange"])
```



SUMMARY

The ever-increasing amount of available data has created a demand for efficient data acquisition methods, such as web scraping, which extracts data from a website.

Python can be used to create an HTTP request, which is sent by the browser. Since the returned data cannot be analyzed directly, Python libraries parse the data. To support the extraction of the required information, regular expressions are used to identify the correct elements and tags flexibly.

Since web technologies advanced over time (e.g., the emergence of JavaScript), there was a need for additional web scraping capabilities, such as mimicking the behavior of users by clicking or scrolling. Libraries like Scrapy allow users to set up multi-page scrapers that enable developers to build complex data acquisition systems.

When web scraping, it is important to consider legal and ethical aspects. Since there is no specific legislation, web scraping is a grey zone. Nevertheless, there are a variety of laws (e.g., the Computer Fraud and Abuse Act) that regulate the use of web scraping technologies. Therefore, developers should take care not to damage web applications and must respect copyright and privacy aspects.

A common way to acquire data is via a data application programming interface. An API can be accessed using the requests library and returns the data, most likely in form of a JSON file. While there are a variety of open APIs that can be accessed freely, API providers often require authentication before data can be retrieved. This is commonly done by using keys and tokens, which help to identify users and limit the amount of data that can be retrieved. In general, there are two types of APIs. Streaming APIs focus on data that are created continuously and can be collected using a listener. Rest API can be queried for specific datasets.

UNIT 4

WORKING WITH COMMON DATA FORMATS

STUDY GOALS

On completion of this unit, you will be able to ...

- define different data formats.
- apply basic data operations with Python.
- distinguish between different binary data models.
- manage binary data formats using Python.

4. WORKING WITH COMMON DATA FORMATS

Introduction

Data can be stored in a variety of ways. Since the emergence of the computer, the number of different formats has been steadily increasing. Most of the time, these formats are written and structured so that they are easy for machines to understand. Comma-separated values (CSV), JavaScript Object Notation (JSON), and Extensible Markup Language (XML) are examples of machine-readable formats. Over time, a few formats have been chosen for specific use cases. CSV files are often used to share small datasets, which can easily be handled in a single file and are then used in different tools for further processing. JSON and XML are most dominant and can be used in a variety of different scenarios. Those file formats, together with their underlying encoding, are presented in the first section of this unit. Along with explicit Python examples, we will be able to handle those file formats.

With the ever-increasing amount of data, new technologies had to be developed to store data more efficiently. The second section of this unit focuses on technologies that were developed in the context of the big data movement. Based on hierarchical data format version five (HDF5), Parquet, and Arrow, three modern data formats are presented alongside tangible Python examples.

4.1 Text-Based Formats (CSV, XML, JSON)

In many real-world use cases, provided data are in a text-based format. Since computers store information in a binary manner, different representations have been developed. One of the oldest representation formats is the American Standard Code for Information Interchange (ASCII) format.

ASCII

In order for computers to process a variety of types of information, a way of storing this information is needed. Since computers can only store zeros and ones in the form of a bit, it was necessary to find a binary representation of characters in a text. To translate the 26 letters, the 10 numerical values, and some special characters into a bit notation, a combination of bits is required. The table below shows how bits can be translated into decimal values.

Table 2: Bits to Decimal Values

Bit	7	6	5	4	3	2	1	0
-----	---	---	---	---	---	---	---	---

Max number	$2^7 = 128$	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$	
Example	0	1	0	0	1	1	0	1	SUM
	0	64	0	0	8	4	0	1	=77

Source: Niklas Wietreck (2021).

A byte, which is a block of 8 bits, represents $2^8 = 256$ values. To translate those bit values into a human readable format, the ASCII standard is used. ASCII was developed to translate binary digits into characters and back. Original ASCII is a seven-bit code that is able to represent 128 different characters by translating decimals into binary values. Other standards used the additional eighth bit to represent even more characters and make a complete byte. The figure below shows how the different ASCII characters translate into binary and decimal notation. The binary values in the first column are a direct translation of the decimal values. The first 32 characters are non-printing characters, ranging from a backspace to an escape. The following 32 characters include collating characters, such as ., +, space, and :, as well as the 10 numeric values. The 26 alphabetic values are listed in combination with some special characters, such as @ and [. The last 26 characters are the alphabetic values in lowercase with a few other special characters like { or | (Interface Age Staff, 1980).

Figure 16: The ASCII Table

Decimal Binary Octal Hex ASCII																			
Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	*	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	\$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00010000	010	08	BS	40	00101000	050	28	(72	01001000	110	48	H	104	01101000	150	68	h
9	00010001	011	09	HT	41	00101001	051	29)	73	01001001	111	49	I	105	01101001	151	69	i
10	00010010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00010011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00011000	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00011001	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00011010	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00011011	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00100000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00100001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00100010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00100011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00101000	024	14	DC4	52	00101000	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00101001	025	15	NAK	53	00101001	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00101010	026	16	SYN	54	00101010	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00101011	027	17	ETB	55	00101011	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00110000	030	18	CAN	56	00110000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00110001	031	19	EM	57	00110001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00110010	032	1A	SUB	58	00110010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00110011	033	1B	ESC	59	00110011	073	3B	;	91	01011011	133	5B	[123	01111011	173	7B	{
28	00110100	034	1C	FS	60	00111000	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	
29	00110101	035	1D	GS	61	00111001	075	3D	=	93	01011101	135	5D]	125	01111101	175	7D	}
30	00110110	036	1E	RS	62	00111100	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00110111	037	1F	US	63	00111101	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

Source: Weiman (2010). CC BY 3.0.

ASCII offers a high variety of characters to the user, but was limited to the eight-bit encoding. Many efforts were made to define standards that can represent more characters. One of the most popular ones is the UTF-8 encoding, which can also represent Greek letters,

Chinese characters, and mathematical signs. The Universal Coded Character Set Transformation Format (UTF), follows the ASCII notation for the first 128 characters. In this section, we will focus on text-based data, which can be seen as an encoded ASCII representation. When working with a text file, it is important to understanding what kind of encoding was used.

CSV

One of the most common data formats is the comma-separated values (CSV) format. As the name indicates, the format contains values separated by a separator. The comma is used as a standard separator, but the semicolon is also widely used. The type of separator is arbitrary and is unimportant to the data format. Regardless of the used separator, the file extension is the same (.csv). A file that is separated by a tab can also be saved with the file extension “.tsv”. CSV files are just simple ASCII text files where a separator character is used to separate a certain set of characters.

To read or write CSV files, Python has some built-in methods contained in the CSV package. As CSV files are simple text files, we can also use the `open()` function to interact with these text files as shown in the example below. Any text file can be read using a “with” statement to open the text file. The `read()` method can be used to extract the content, which is then returned through a print statement.

Code

```
with open('example.txt') as f:
    content = f.read()
    print(content)
```

While this is already useful and will allow users to manipulate CSV files, there is another way to work with such files. The pandas package, which is widely utilized in the data analytics community, is used in the following examples to demonstrate how to work with tabular data, such as CSV. Pandas is an abbreviation of panel data.

Before loading the data into Python, inspecting the CSV data is recommended. This can be done, for example, with an editor, and will show the following data for the Islands.csv file. The Islands.csv file can be created by opening the editor on the local computer, copying the following text, and then saving the file as .csv (Countrymeters, 2021).

Code

```
Island;Year;Residents;Capital;Continent
Cape Verde;2005;471000;Praia;Africa
Cape Verde;2010;509000;Praia;Africa
Cape Verde;2015;546000;Praia;Africa
Fiji;2005;837000;Suva;Oceania
Fiji;2010;876000;Suva;Oceania
Fiji;2015;910000;Suva;Oceania
```

```
Isle of Skye;2005;10300;Portree;Europe
Isle of Skye;2010;10500;Portree;Europe
Isle of Skye;2015;10800;Portree;Europe
```

A header with the names of the columns can be identified at the top of the dataset, and every data row consists of values separated by a semicolon. The dataset has nine rows showing the data for three different islands for three different years.

In order to use this dataset in Python, we read the data with the pandas method `read_csv()` to load the data into a pandas DataFrame. In order to load the data into a DataFrame, we specify the location of the file and the separator of the CSV file. The separator can be defined in a parameter called `sep`. As a result, we obtain a well-structured tabular representation of the CSV data, including column names and a row index.

Code

```
import pandas as pd
data = pd.read_csv("Islands.csv", sep = ";")
```

Figure 17: Island DataFrame

	Island	Year	Residents	Capital	Continent
0	Cape Verde	2005	471,000	Praia	Africa
1	Cape Verde	2010	509,000	Praia	Africa
2	Cape Verde	2015	546,000	Praia	Africa
3	Fiji	2005	837,000	Suva	Oceania
4	Fiji	2010	876,000	Suva	Oceania
5	Fiji	2015	910,000	Suva	Oceania
6	Isle of Skye	2005	10,300	Portree	Europe
7	Isle of Skye	2010	10,500	Portree	Europe
8	Isle of Skye	2015	10,800	Portree	Europe

Source: Niklas Wietreck (2021).

The DataFrame takes the first row of the CSV file and defines it as the header. If the CSV file is correctly structured, it can help to address certain columns. Unfortunately, there are scenarios where this does not return the expected result. In the following scenario, there is no header present in the CSV file, so this has to be added manually. The example input CSV could be as follows:

Code

```
Cape Verde;2005;471000;Praia;Africa
Cape Verde;2010;509000;Praia;Africa
Cape Verde;2015;546000;Praia;Africa
```

```
Fiji;2005;837000;Suva;Oceania
Fiji;2010;876000;Suva;Oceania
Fiji;2015;910000;Suva;Oceania
Isle of Skye;2005;10300;Portree;Europe
Isle of Skye;2010;10500;Portree;Europe
Isle of Skye;2015;10800;Portree;Europe
```

To bring it into the desired structure, it is necessary to hand over the names argument, together with the expected column names in the right order. This code returns the same results as in the example above.

Code

```
column_names = ["Island", "Year", "Residents", \
                "Capital","Continent"]
data = pd.read_csv("Islands_noHeader.csv", \
                  names = column_names, sep = ";")
```

Next, we see another common example where the `read_csv()` method has to be modified. This can be the case when there are some specifications or metadata above the actual data, such as an export header for this file. This is often the case when working with data that are extracted directly from a system. The header might include the date of the extraction, the specific source, etc. The following CSV file contains a header which indicates the extraction date, the extraction format, and the requester. Since there is only one value in those rows, the separator will separate empty values from each other to meet the overall number of expected separators in each row.

Code

```
Extraction Date: 24.05.2016 12:02:21;;;
Data Format: csv;;;
Requester: IU;;;
```

```
Island;Year;Residents;Capital;Continent
Cape Verde;2005;471000;Praia;Africa
Cape Verde;2010;509000;Praia;Africa
Cape Verde;2015;546000;Praia;Africa
Fiji;2005;837000;Suva;Oceania
Fiji;2010;876000;Suva;Oceania
Fiji;2015;910000;Suva;Oceania
Isle of Skye;2005;10300;Portree;Europe
Isle of Skye;2010;10500;Portree;Europe
Isle of Skye;2015;10800;Portree;Europe
```

To skip the first four rows, the `read_csv()` method takes a `skiprows` argument, which anticipates the integer of each row that should be skipped. In this case, the first four rows are skipped using the range function. Functions can be used to identify the header row, for

example, by name. This provides additional flexibility since automatically generated headers do not always have the same number of rows. The following line of code returns the same results as in the previous examples:

Code

```
data = pd.read_csv("Islands_meta.csv", sep = ";", \
    skiprows = range(0,4))
```

Encoding errors can occur, especially with outputs from old systems. These are sometimes difficult to identify and solve. The following example uses the encoding parameter to make sure that the CSV is read as UTF-8.

Code

```
data = pd.read_csv("Islands.csv", sep = ";", \
    skiprows = range(0,4), encoding = "utf-8")
```

Now, as the data are loaded into a pandas DataFrame, they can be filtered, sorted, grouped, and transformed. As we have seen, when working with CSV data, it is important to understand the structure of the file to be used. Header rows and export headers need to be identified and handled. When reading or writing data back to a CSV file, encoding must also be considered. The following example writes the DataFrame into a CSV file called `Islands_output.csv` with a separator `;`:

```
data.to_csv("Islands_output.csv", sep = ";")
```

JSON

The JavaScript Object Notation (JSON) is a data format most commonly used for transfers in the web (Müller & Guido, 2017). It became popular for its clean, easy to read structure, and for the possibility of flexible application, especially for nested semi- and unstructured data. JSON data are similar to a Python dictionary, and consist of a combination of key value pairs. JSON files start with curly brackets, then the key and value are defined, separated by a colon. Entries are separated by a comma, which indicates the start of the next element. At the baseline, a JSON file is similar to a CSV file: an ASCII text file in a human-readable representation. In the following JSON file, we can see the weather for London on May 24, 2021. The data are returned from the openweathermap API (Open Weather, n.d.), which is a publicly available API for global weather data.

Code

```
{"coord": {"lon": -0.1257, "lat": 51.5085},
"weather": [{"id": 500,
"main": "Rain",
"description": "light rain",
"icon": "10d"}],
"base": "stations",
"main": {"temp": 282.22,
"feels_like": 279.41,
```

```
"temp_min": 280.64,  
"temp_max": 283.86,  
"pressure": 1005,  
"humidity": 80},  
"visibility": 10000,  
"wind": {"speed": 5.43,  
"deg": 305,  
"gust": 11.39},  
"rain": {"1h": 0.12},  
"clouds": {"all": 90},  
"dt": 1621884916,  
"sys": {"type": 2,  
"id": 268730,  
"country": "GB",  
"sunrise": 1621828607,  
"sunset": 1621886294},  
"timezone": 3600,  
"id": 2643743,  
"name": "London",  
"cod": 200}
```

When reading this file in Python, the JSON package is used. After reading the file using Python's built-in `open()` and `read()` functions, the JSON package is used to load the data. While the return of `json_data` is a string, the data object itself will be a Python dictionary.

Code

```
import json  
json_data = open('LondonWeather.json').read()  
data = json.loads(json_data)
```

As we can see in the code above, JSON files can be nested. In order to access elements of a specific level in this nested structure, we can use the top level key of this dictionary for our query.

Code

```
data["main"]  
  
# console output:  
{'temp': 282.22,  
'feels_like': 279.41,  
'temp_min': 280.64,  
'temp_max': 283.86,  
'pressure': 1005,  
'humidity': 80}
```

Such a structure can then easily be put into a DataFrame as this is no longer nested, but is in flat format. Even though pandas has its own method to read JSON files, it can be difficult to put an unstructured format like JSON into a column-based format. When handing over the “main” section of our JSON example to a DataFrame, the keys will be used as columns. In order to give more context to the weather data, the name of the city is passed as the row index.

Code

```
weather_data = pd.DataFrame(data["main"], \
    index = [data["name"]])
```

Alternatively, we can also normalize the JSON directly to a pandas DataFrame using the `json_normalize()` function.

```
pd.io.json.json_normalize(data['main'])
```

In this example, the data from a JSON file were used. When accessing the API directly, the data for each city can be retrieved and the value can be stored in a data structure like a DataFrame. Together with additional information, such as the time of the measurement, a script can be built that periodically extracts weather information from the API, brings it into the desired order, and stores it in a DataFrame or database.

XML

Hierarchical data from the web are often represented using the Extensible Markup Language (XML). XML data are made readable for machines and humans. The following example XML file, which can be downloaded as an XML file from the XML module documentation, shows information about different countries (Python Software Foundation, n.d.). When opening the data in an editor, the following structure will appear.

Figure 18: XML Example Countries

```
<?xml version="1.0"?>
<data>
  <country name="Liechtenstein">
    <rank>1</rank>
    <year>2008</year>
    <gdppc>141100</gdppc>
    <neighbor name="Austria" direction="E"/>
    <neighbor name="Switzerland" direction="W"/>
  </country>
  <country name="Singapore">
    <rank>4</rank>
    <year>2011</year>
    <gdppc>59900</gdppc>
    <neighbor name="Malaysia" direction="N"/>
  </country>
  <country name="Panama">
    <rank>68</rank>
    <year>2011</year>
    <gdppc>13600</gdppc>
    <neighbor name="Costa Rica" direction="W"/>
    <neighbor name="Colombia" direction="E"/>
  </country>
</data>
```

Source: Niklas Wietreck (2021).

The XML ElementTree class will first be imported before the XML file is parsed. The `getroot()` method is used to get the root of the hierarchical data structure. In this case, this is the tag `<data>`. A for loop is used to display the tags and attributes of each child object using the attribute notation. A specific value can be accessed using the array notation. The following script will return the name of every single country in the dataset and the value of the of second element in the first child.

Code

```
import xml.etree.ElementTree as ET
tree = ET.parse('data.xml')
root = tree.getroot()
for child in root:
    print(child.tag, child.attrib)
print(root[0][1].text)

# console output:
# country {'name': 'Liechtenstein'}
# country {'name': 'Singapore'}
# country {'name': 'Panama'}
```

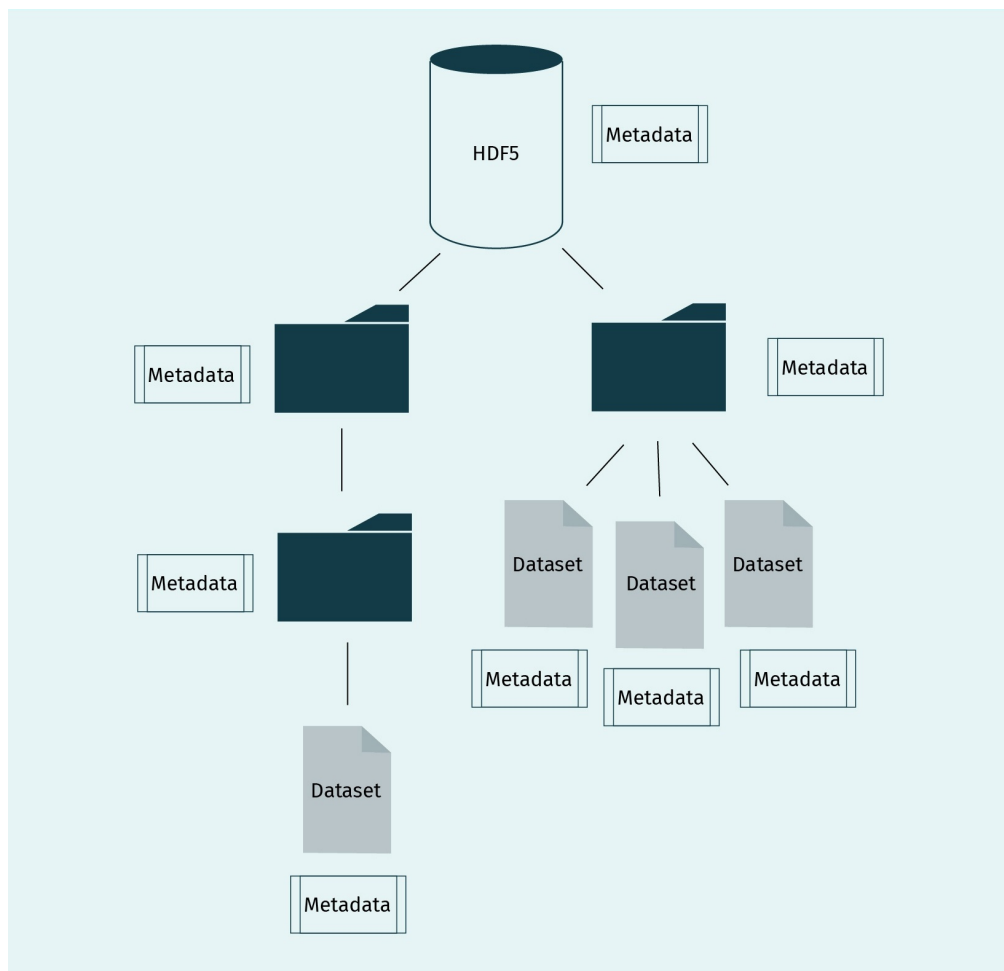
4.2 Binary Formats (HDF5, Parquet, Arrow)

In addition to the more traditional data formats, there are a variety of more modern storage methods that can store data more efficiently in specific use cases. In this section, HDF5, Parquet, and Arrow are introduced.

HDF5

The Hierarchical Data Format (HDF) contains a model that manages and stores large volumes of data. It is often used for complex and heterogeneous data. The HDF5 format is a file system that is contained and described in a single file. The contained file system works like the folder structure on a computer, though it has a different name: groups. Groups can contain a number of different files, which are called datasets. The figure below shows an example structure of HDF5.

Figure 19: HDF5 Data Model



Source: Niklas Wietreck (2021), based on mjones (2020).

HDF5 is a self-describing format, which means that each element (group or dataset) has an associated metadata file that contains information about the data. This includes names, descriptions, and any other documentation the user wants to add. This additional and separate metadata document allows users to automate processes. Other advantages of HDF5 include the storage of large complex and heterogeneous data, since the format was designed to compress a high quantity of data. Additionally it supports data slicing, which means that only the files used for analysis are read into the computer's memory.

In the following example, the `h5py` module is used to create a HDF5 dataset in Python. After instantiating the HDF5 file, the `create_dataset()` method is used to build the dataset. Parameters for names, shape, and integer type are also handed over in that step. Attributes of the dataset can be accessed on the object. In the following example, the shape, name, and parent are returned; the created empty dataset does not belong to a group, i.e., parent, but we see that this data format allows us to group and nest our datasets and introduce a structuring hierarchy.

Code

```
# load packages
import h5py

# create an HDF5 file
file = h5py.File('iu.h5','w')

# create an empty dataset in the HDF5 file
dataset = file.create_dataset("iu", (4, 6))

# print information about the dataset
print("Dataset shape is", dataset.shape )
print("Dataset name is", dataset.name)
print("Dataset is a member of the group", \
      dataset.parent )

# close the file.close()
```

In the following example, the previously created file is opened and the created dataset is accessed. Multidimensional example data are created using NumPy and then written into the dataset. From there, the data can now be read again.

Code

```
# load packages
import numpy as np

# read/write HDF5 file
file = h5py.File('iu.h5','r+')

# list existing datasets in the file
list(file.keys())

# create an empty dataset in the HDF5 file
# if this dataset does not already exist
if not "iu_numbers" in list(file.keys()):
    dataset = file.create_dataset("iu_numbers", (4, 6))
else:
    dataset = file['/iu_numbers']

# generate sample data
data = np.random.rand(4*6).round(2).reshape(4, 6)

# add the data to the HDF5 dataset
dataset[...] = data

# read the data back from the HDF5 file
data_read = dataset[...]
print(data_read)
```

```
# console output:
# [[0.65 0.96 0.77 0.33 0.19 0.93]
# [0.11 0.31 0.99 0.01 0.61 0.48]
# [0.09 0.79 0.4 0.15 0.3 0.35]
# [0.97 0.36 0.27 0.45 0.21 0.59]]

# close the file
file.close()
```

Metadata can be stored in the file using the `.attrs[]` method. The following script can be added to the example above to add metadata information to the file.

```
dataset.attrs["User"] = "ME"
```

In order to access all metainformation, it is possible to iterate over the keys and return the values for each key.

Code

```
for k in dataset.attrs.keys():
    print(k, dataset.attrs[k])
```

Parquet

As another available data representation, Parquet was built within the Hadoop ecosystem to take advantage of compressed and efficient data with a columnar structure. Parquet possesses a complex nested data structure and utilizes the record shredding and assembly algorithm described in the **Dremel** query system. This means that in comparison to row-based formats like CSV, Parquet is a hybrid of column- and row-oriented formats, resulting in a structure making use of row groups and column chunks. This allows for a combination of the desirable advantages of horizontal (suitable for low-latency transactions) and vertical partitioning (suitable for data analysis). Another characteristic of Parquet is that the metadata include the schema and structure. This makes it a self-describing file format, which provides us with information about how best to read the file in order to minimize input/output.

Dremel

As a scalable query system for data analysis, Dremel mainly focuses on read-only nested data.

Table 3: Row-Based versus Columnar Storage

Row-based storage	Columnar storage
<ol style="list-style-type: none"> Germany, Berlin, 83 France, Paris, 67 Italy, Rome, 60 	Country: Germany, France, Italy Capital: Berlin, Paris, Rome Inhabitants: 83, 67, 60

Source: Niklas Wietreck (2021).

Advantages of Parquet include the high compression capabilities, which can be chosen in a flexible manner. Different types of compression methods, in cooperation with extendable encoding schemas, are used to make the file as small as possible. Encoding methods

include dictionary encoding, bit packing, and run length encoding (RLE). Dictionary encoding is normally used for datasets where unique values are only available in a small number. Bit packing is used for the storage of integers with dedicated 32 or 64 bits per integer. This efficiently stores small integers. When a value occurs multiple times, RLE stores only one entry along with the number of occurrences.

Another advantage of Parquet is that it only retrieves the relevant data in an efficient manner since the amount of scanned data is small. This occurs because of the self-describing format where each file contains metadata, as well as the data themselves. When querying the dataset, only the required columns and their respective values are loaded into memory. Since Parquet is designed to keep the metadata separated from the actual data, it is possible to split columns into separate files that are then referenced using the metadata file. This enables the efficient and flexible handling of the data.

In Parquet, there are two main configurations that enable the optimization of the files. One configuration is related to the row group size, which allows the data to be chunked into larger pieces. The data page size configuration enables a single row lookup. By optimizing this, the space overhead is reduced.

To create, access, and write a Parquet file, there are a variety of possibilities in Python, and, for this example, the capabilities of pandas will be explored. First, the required modules are important. Pyarrow will be used in this example for the communication with pandas. After creating a sample dataset, it is written to an Arrow table. The `write_table()` method can be used to create a new Parquet file. Additional configuration parameters, such as the `data_page_size`, can be added.

Code

```
# load packages
import pyarrow.parquet as pq
import pyarrow as pa
import pandas as pd
import numpy as np

# create a pandas DataFrame
df = pd.DataFrame(np.random.randn(100).\
    reshape(25,4), columns = ["one", "two", \
    "three", "four"])

# convert the pandas DataFrame to a parquet table
tableToWrite = pa.Table.from_pandas(df)

# write the parquet table to file
pq.write_table(tableToWrite, "myPQFile.parquet")
```

To read from a Parquet file, the `read_table()` method is used. After applying the `to_pandas()` method, the data are back in a DataFrame. By adding values in the `columns` parameter, it is possible to read only a subset of the Parquet file.

Code

```
tableToRead = pq.read_table("parquet.writeTable")
tableToRead.to_pandas()
```

Attributes like `.metadata` or `.schema` will return information about the file, e.g., the number of columns, rows, groups, or version.

Code

```
file = pq.ParquetFile("myPQFile.parquet")
file.metadata
```

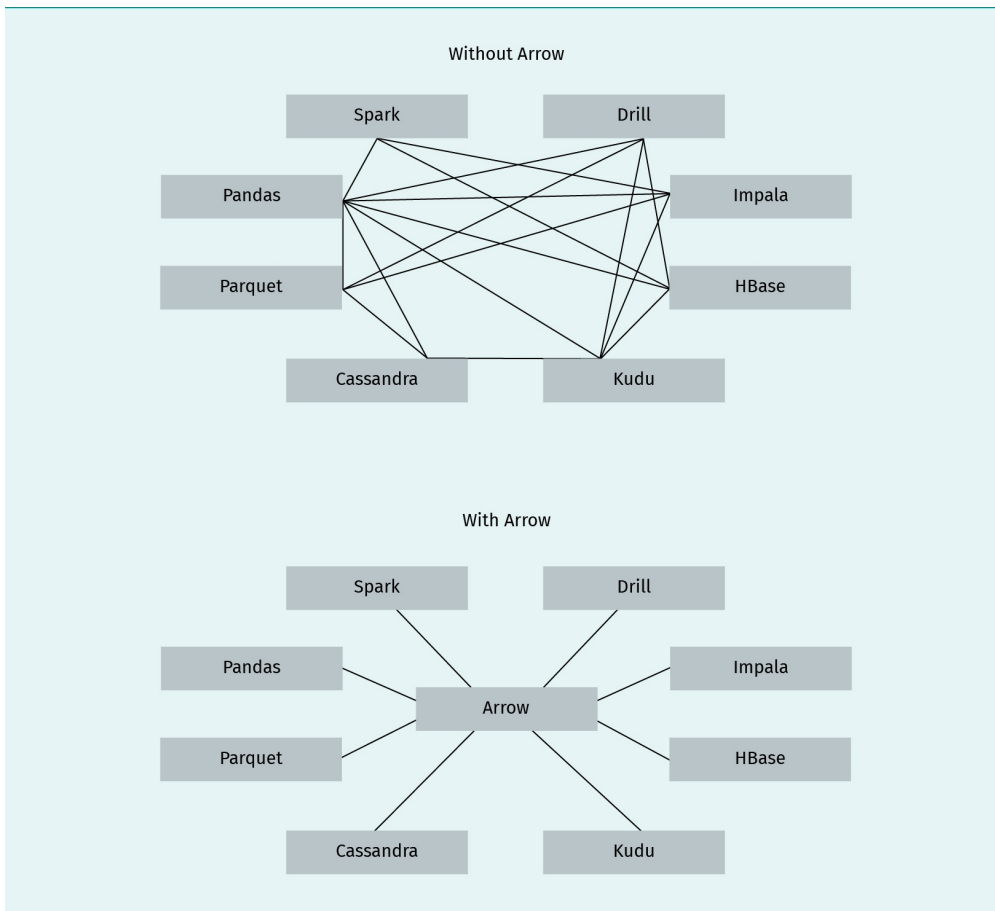
```
<pyarrow._parquet.FileMetaData object at 0x00000279639E7E28>
  created_by: parquet-cpp-arrow version 4.0.1
  num_columns: 4
  num_rows: 25
  num_row_groups: 1
  format_version: 1.0
  serialized_size: 2739
```

Arrow

The development platform Arrow was built for in-memory analytics, which allows the processing and moving of data in a big data environment. This is especially useful when data need to be exchanged with a low overhead. This results in a high query performance, even for complex analytical tasks. It is optimized for the analysis of columnar data formats, either as flat files or hierarchical data. The project is language agnostic, which allows a practical application in a variety of use cases. This facilitates efficient communication between many components. In the following figure, Arrow helps to connect the following:

- Spark (framework for cluster computing)
- Drill (supports data intensive distributed applications)
- Impala (used for querying Hadoop)
- HBase (distributed relational database)
- Kudu (column-based data storage)
- Cassandra (database management system)
- Parquet (explained above)
- pandas (Python library for data analysis)

Figure 20: Arrow Overview



Source: Niklas Wietreck (2021), based on The Apache Software Foundation (n.d.).

The following figure shows how the data can look in memory.


```
# convert the DataFrame to an arrow table
table = pa.Table.from_pandas(df)
```

To make the process of reading data into an Arrow table as efficient as possible, there are a variety of methods, including `read_csv()` and `read_json()`. Additionally, the communication with the Parquet data format is possible, which provides a standardized open-source columnar storage format. Arrow also makes it possible to write to and retrieve data from HDF5 files and can therefore be used as a link between HDFs, Parquet, and other data models.

Methods like `to_pandas()` allow users to return data to a pandas DataFrame.

```
df_new = table.to_pandas()
```

In addition, it is possible to read many files in a directory as one dataset. This enables the analysis of even larger datasets. While this example only showed the main function of Arrow (to create the Arrow object), there are a couple of advantages over DataFrames. Besides containing extensible metadata information of the flat or nested data types, it is also possible to create user-defined types. Pandas objects tend to perform poorly when it comes to database or flatfile ingestions, or export. Arrow, on the other hand, with its streaming and chunk-based oriented design, enables the movement of and access to large datasets at maximum speeds. Overall, Arrow is designed for analytical processing performance and can be used to process very large datasets in a more efficient way than pandas DataFrames.



SUMMARY

There are many common formats that can store and handle data. Before data can be stored on a disc, they must be machine-readable. A common standard for de- and encoding is the eight-bit ASCII standard, which represents 128 characters, e.g., alphabetic, numeric, collating, or non-printing. The UTF-8 standard is also capable of displaying Greek letters and Chinese characters.

One of the most common data formats is the comma-separated values (CSV) format. It stores the values in a given structure, using a separator for the values. This results in a tabular layout that can easily be read, for example, using the pandas `read_csv()` method in Python. Afterwards, Python allows the data to be sorted and filtered while keeping the native tabular structure.

JSON does not have a tabular structure, and is mainly used for the transfer of data on the web. It follows a clean, easy-to-read structure, and can be adjusted to any type of data. The structure can be used like a Python dictionary.

The HDF5 format is newer than CSV. It consists of a data model and a storage model, which enable the efficient manipulation, storing, and transfer of data. The file format is based on groupings, which creates a hierarchy of different datasets. HDF5 can be created and managed using Python with the h5py library.

Parquet also emerged in the context of big data. Parquet uses a columnar storage approach where data are nested along the column headers. This enables the retrieval of data that are actually needed, resulting in higher performance and efficient use of memory.

To facilitate the communication between different types of formats, Arrow can be used. While it is optimized for the handling of columnar data, it can also be applied to hierarchical data. Arrow is language agnostic and can be used in many use cases.

UNIT 5

TIDY DATA

STUDY GOALS

On completion of this unit, you will be able to ...

- identify issues in data.
- chose appropriate solutions to solve issues in data.
- implement solutions using Python.
- execute data enrichment actions.

5. TIDY DATA

Introduction

Organizations use a lot of their resources to create tidy data. According to Dasu and Johnson (2003), 80 percent of an analyst's time is spent cleaning and restructuring the dataset. Wickham (2014) says that tidy data have a specific structure, are easily manipulated, and are simple to visualize and model, which therefore keeps the requirements quite general. He adds that tidy data are the basis of each analysis, so there is no way to bypass this time intensive step. The only way to use this time as efficiently as possible and reduce the resources spent on the cleansing process is to use proven methods and technologies (Dasu & Johnson, 2003). However, the research in this field is quite limited in comparison to research on other elements of the data analytics process. One reason for this is that the process of creating tidy data includes a wide range of different activities, such as outlier detection, restructuring of data, and addition of new data (Wickham, 2014). Additionally, the data cleansing processes is highly use case-specific with a high number and variety of problems that can appear. Although many people regard the creation of tidy data as a one-time action, it is important to note that there is a need for robust solutions that will last over time. This is particularly true if new data need to be added in an efficient and structured manner (Kazil & Jarmul, 2016).

In this unit, the topic of tidy data will be divided into three different elements. First, different data structuring methods will be described. To put this into context, the data workflow as defined by Rattenbury et al. (2017) will be presented. After the data are properly structured, the focus will shift to the cleansing of the data. A variety of methods to remove duplicates, detect outliers and bad data, execute advanced matching algorithms, use standardization, and execute data imputation will be presented, alongside practical coding examples. Different methods of enriching a dataset are explained in the final section. The concepts of feature construction, feature transformation, and feature learning will also be introduced. The unit will close with a collection of different methods that are able to add new data to the dataset.

5.1 Structuring

According to Rattenbury et al. (2017), the process of structuring data is the combination of different actions that change the form or schema of the data. The author distinguishes between two types of structuring actions. The first type is intrarecord structuring transformation, which includes the reordering of columns, the creation of new columns by extraction values, and the combination of multiple columns into one. The second type of structuring transformations is interrecord structuring. This includes the filtering of a dataset and the shift of granularity through aggregations and pivots (Rattenbury et al., 2017).

Intrarecord Structuring

New columns are often created by extracting elements from an existing column. A common use case is the extraction of a substring into a new column. The easiest way to get a substring is extraction based on the position. In this case, the start point and the end point are defined, and the requested substring is stored in a new column (Rattenbury et al., 2017). A common scenario for this is the modification of a date or time field. In the following example, a DataFrame was created that contains different date values in the column “date,” which are in the format “mmddy” (month-day-year). A new column “day” is created, which takes the existing dates and extracts only the day from each string. The value of the day starts with the second and ends with the fourth element. First, the `str()` method is used to ensure that the value is actually a string before the `slice()` method extracts the requested elements. The new DataFrame now contains two columns: date and day.

Code

```
# generate data with date strings
dates = pd.DataFrame([ \
    "02082021", "02152021", "02122021", \
    "02212021", "02012021", "02062021"], \
    columns = ["Date"])

# extract the day from a date string
dates['Day'] = dates['Date'].str.slice(2, 4)

print(dates)
# console output:
#      Date Day
# 0 02082021 08
# 1 02152021 15
# 2 02122021 12
# 3 02212021 21
# 4 02012021 01
# 5 02062021 06
```

Another method of extracting sub-strings is based on a pattern. This method can use a variety of rules in combination with a good understanding of the data. This will then allow the extraction of the desired substring. In the following example, the goal is to extract the numerical values from a given string. Based on the data, the following rule could be formed: From the column transactions, extract the all digits that are followed by a dot, and then take two more digits afterwards. **Regular expressions** are often used to convert such rules to a format that machines can understand (Kazil & Jarmul, 2016).

Regular expressions
These allow computers to match, find, and remove certain patterns in strings.

Table 4

Bank Account Transactions
Transactions

Paid in \$100.00 birthday

Paid out \$40.50 cash withdrawal

Paid in \$500.21 salary

Source: Niklas Wietreck (2021).

The Python code to extract the values from the first row could be as follows:

Code

```
# Using regex to find the value in a specific format\ within a string.
txt = "Paid in $100.00 birthday"
re.findall("\d*\.\d{2}", txt)
# console output:
#[ '100.00' ]
```

When structuring the data, it can be necessary to process hierarchical structures within the dataset. This mainly occurs when the data were created by machines and often appear in the JavaScript Object Notation (JSON) format. JSON data generally come in two different types: JSON array and JSON map. JSON arrays are similar to a list in Python, containing a sequence of different entries separated by commas and enclosed in square brackets (Rattenbury et al., 2017). Double quotes are often used to enclose the different values. JSON map data, on the other hand, contain a combination of key-value pairs. They are initiated by curly brackets and the keys are separated from the values with a colon. Key-value pairs are separated from each other using commas. Since those key-value pairs often do not follow an ideal structure for an analytics task, they need to be formatted into a column-based structure. This will result in a table format, where each key is a column by itself. The problem that often occurs in this context is that in JSON, key-value pairs can be values, as a nested structure is allowed and even desired in this data format. A possible data structure could be as follows:

Code

```
{key: value,
  key: {key, value},
  key: {key: {key, value}}}
```

Another action within the intrarecord structuring process is the combination of multiple fields. These actions are basically the opposite of the aforementioned extraction activities (Rattenbury et al., 2017). In the following example, the columns for street name and zip code should be combined into an address field. When combining two columns, the question is whether there should be a separator (a comma or space) between the two elements of the new field. The following example creates a DataFrame with two different addresses in three different columns. There are columns for the street name, the zip code, and the city name. The “+” symbol can be used to concatenate each element into a new column named “address.” To make it easier to read, additional separators are added between each field.

Code

```
# generate sample data
customers = pd.DataFrame([ \
    ["P. Sherman 42 Wallaby Way", "2000", "Sydney"], \
    ["221B Baker Street", "NW1 6XE", "London"]], \
    columns = ["Street", "Zip code", "City"])

# combine fields to a human readable address
customers["Address"] = customers["Street"].astype(str) + \
    ' , ' + customers["Zip code"] + ' ' + \
    customers["City"]
```

Table 5: Raw Input of Customers

Index	Street	Zip code	City
0	P. Sherman 42 Wallaby Way	2000	Sydney
1	221B Baker Street	NW1 6XE	London

Source: Niklas Wietreck (2021).

Table 6: Combined Customer Address

Index	Address
0	P. Sherman 42 Wallaby Way , 2000 Sydney
1	221B Baker Street , NW1 6XE London

Source: Niklas Wietreck (2021).

Interrecord Structuring

Interrecord structuring changes the overall structure of the dataset. While filtering is mainly used for data cleansing, it can also be used to change the granularity of a dataset. This method is called record-based filtering, and it filters a specific record in order to focus on the data. If a dataset, for example, contains the brand of a car in one column and the name of car in another, there might be a variety of brands. By filtering a single brand, the newly produced dataset will belong to only one category and will therefore increase the focus (Rattenbury et al., 2017).

The following example creates the DataFrame “cars,” which contains columns for the brand and the name of the model. After the DataFrame is created, the `loc` method can be used to filter the dataset based on a specific condition. In this example, only rows in which the “brand” includes “VW” are selected. To work with this newly derived dataset, it is useful to store it in a new DataFrame.

Code

```
# generate the data
cars = pd.DataFrame([ \
    ['VW', 'Golf'], \
    ['VW', 'Passat'], \
    ['VW', 'Polo'], \
    ['Mercedes', 'A-Class'], \
    ['Mercedes', 'AMG'], \
    ['Tesla', 'Model 3'], \
    ['Tesla', 'Cybertruck'], \
    ], columns=['Brand', 'Model']
)

# filter the data by brand
cars.loc[cars.Brand == "VW"]
```

Table 7: Filter Data by Brand

Brand	Model
VW	Gold
VW	Passat
VW	Polo

Source: Niklas Wietreck (2021).

Another method for interrecord structuring is the shift of granularity using aggregations and pivots. A typical use case for aggregations is when data are summarized for different time periods (Rattenbury et al., 2017). The granularity can be set to year, and all values of the dataset will be aggregated to a yearly sum, mean, or other specified function. In the following example, there are values for each month of the first quarters of 2020 and 2021. The `groupby()` method is used to define the level on which the DataFrame should be aggregated. In this case, the column year is selected. In order to sum the values of the sales, the method `sum()` is used. The script returns a dataset that includes the sales for the first quarter for each year.

Code

```
# generate the sample data
sales = pd.DataFrame([ \
    [2020, "January", 212182], \
    [2020, "February", 221921], \
    [2020, "March", 152281], \
    [2021, "January", 243822], \
    [2021, "February", 123212], \
    [2021, "March", 162319]
    ], columns = ["Year", "Month", "Sales"]
)
```

```
# aggregate by group
sales.groupby("Year").sum()
```

Table 8: Aggregations with Groupby

Year	Sales
2020	586384
2021	529353

Source: Niklas Wietreck (2021).

The method of column-to-row pivots maps each input record to multiple output records where each output record only maps back to one input record. This results in a subset of the input record fields. The following table shows three basketball players and the points they won in two National Basketball Association (NBA) seasons, 2017 and 2018, as it can be found in the NBA stats database (The National Basketball Association, n.d.).

Table 9: NBA Player Statistics Season 2017 and 2018 as Columns

Player	2017	2018
LeBron James	2251	1505
Kawhi Leonard	210	2040
Anthony Davis	2110	1452

Source: Niklas Wietreck (2021), based on The National Basketball Association (n.d.).

When the method of column-to-row pivots is applied, the table looks like the one below. The table was restructured so that each row shows a unique combination of features. This is also called a long format as the resulting table is longer (it has more rows) than before.

Table 10: NBA Player Statistics Season 2017 and 2018 as Rows

Player	Year	Points
LeBron James	2017	2251
LeBron James	2018	1505
Kawhi Leonard	2017	210
Kawhi Leonard	2018	2040
Anthony Davis	2017	2110
Anthony Davis	2018	1452

Source: Niklas Wietreck (2021), based on The National Basketball Association (n.d.).

This process can be called unpivoting or denormalization, and creates a row-based format. It is commonly used when the goal is to create a new dataset that allows users to easily summarize values (Rattenbury et al., 2017).

Row-to-column pivots do exactly the opposite, where multiple input records can result in one output record. Output records can involve aggregations (sum or max). As an example, the following table shows a list of earnings paid into a bank account.

Table 11: Account Overview: Rows

Account: NR	Job	Amount
A0023	Ice cream vendor	450
A0023	Ice cream vendor	200
A0024	Barkeeper	350
A0024	Barkeeper	350
A0024	Ice cream vendor	200
A0024	Waiter	250

Source: Niklas Wietreck (2021).

A row-to-column pivot will restructure the table so that two new columns are created, displaying the sum of each job in each bank account. The resulting data structure is also called a wide format as the new table has more columns than before.

Table 12: Account Overview: Aggregated

Account: NR	Sum of ice cream job	Sum of barkeeper job	Sum of waiter job
A0023	650	0	0
A0024	200	700	250

Source: Niklas Wietreck (2021).

The following Python example executes the aforementioned steps. First, the DataFrame is created. This has a similar layout to the table above. In the second step, the `pivot_table()` method is used to create the aggregated view. After adding the previously created DataFrame, the column containing the values must be defined. Additional parameters for the index column and the new columns need to be defined. The way the values are aggregated is defined in the “aggfunc” parameter.

Code

```
# generate the sample data
df = pd.DataFrame({ \
    "Account: NR": [ \
```

```

    "A0023", "A0023", "A0024", \
    "A0024", "A0024", "A0024"], \
    "Job": [ \
        "Ice cream vendor", "Ice cream vendor", \
        "Barkeeper", "Barkeeper", \
        "Ice cream vendor", "Waiter"], \
    "Amount": [450, 200, 350, 350, 200, 250]})

# pivot the table
import numpy as np
pivot = pd.pivot_table(df, values="Amount", \
    index=["Account: NR"], columns=["Job"], \
    aggfunc=np.sum)

print(pivot)
# console output:
# Job          Barkeeper Ice cream vendor Waiter
# Account: NR
# A0023          NaN          650.0      NaN
# A0024          700.0          200.0    250.0

```

Data Profiling

Before the transformation and analysis of the data can take place, it is necessary to create a good understanding of the data. The process of getting to know what type of data are at hand, how they are structured, and what kind of bad entries might be included, is called data profiling. While data profiling can initially be used to understand the data, it can also be used to monitor the data quality of a specific dataset over time after a variety of transformations have been applied. With the ever-growing amount of data, the profiling of the dataset is increasingly crucial since an analyst will not be able to regard each single entry individually. The process of profiling can be started from two different perspectives: Values can be regarded individually (which is often impossible), or several values of the dataset can be considered at the same time as a summary. Both views are often created in form of a list of data values or statistical summary tables. Simple visualizations can help to create a better understanding of the data and support the process of profiling (Abedjan et al., 2017).

The individual value view includes two types of checks that can be applied to the data. **Syntactic profiling** describes the analysis of the literal values that are present in each field. Such values are often represented in a list. Boolean values, for example, will be represented in a list of Boolean values (e.g., true and false). Analysts are challenged when the number of distinct values is high. The daily sales at every store of a global supermarket chain could create thousands of entries that cannot be regarded individually. Though this list can be quite extensive, it will help to create a better understanding for the selected field. **Semantic profiling** is related to the meaning of the values and checks, if a value satisfies a given constraint. If a field, for example, is related to the age of a customer who filled out a questionnaire, the value for a customer who did not respond could be defined as -1. Even though this value is not a proper representation of age and therefore not syntac-

Syntactic profiling

The syntax describes the constraints on each value that is valid within a field.

Semantic profiling

The meaning of a value is contained in the semantic field.

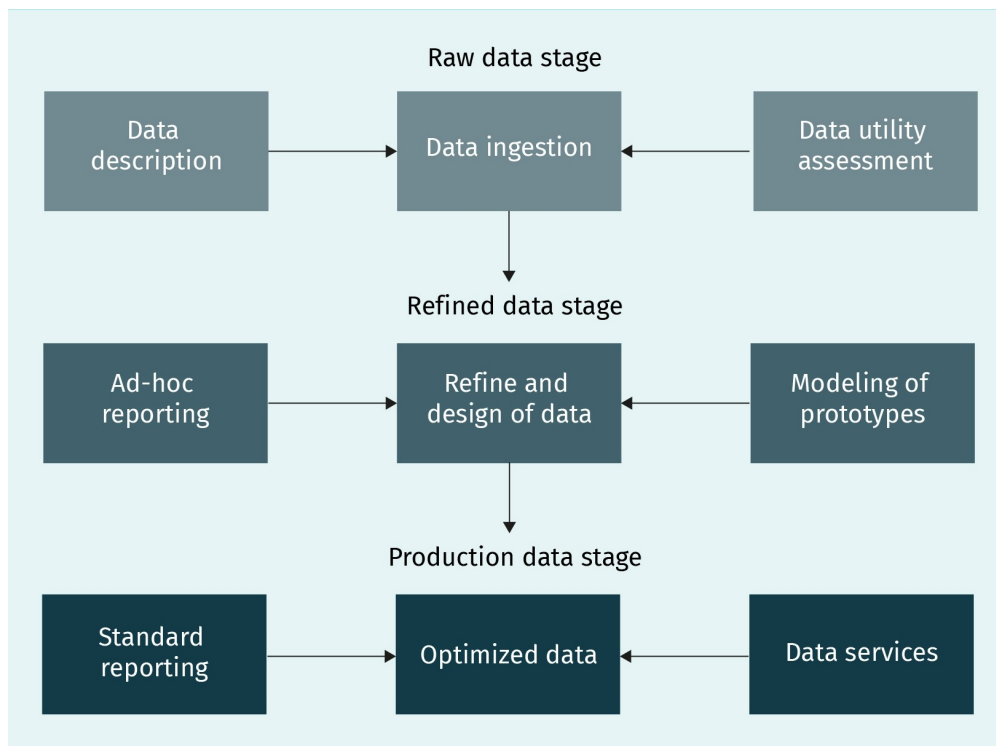
tically correct, it can be interpreted semantically. This could then be translated into a new field that contains a value for the case where the customer reported the age or did not. In another scenario, semantic profiling can be used to transform a variable, such as age, into a categorical set, e.g., life stages (child, adult, and senior). While executing semantic profiling new fields that contain the encoded interpretation of the original field are often created. The encoded interpretation is then usually combined with statistical measures that return the number of occurrences or the number of invalid entries.

The shape and the distribution of values are analyzed in the set-based profiling. This type of profiling includes the relationships between multiple fields and is often displayed in the form of different visualizations. Besides descriptive statistics, such as the sum, mean, maximum, or minimum, statistical measures can help to identify unexpected distributions or outliers. Visualization methods, such as histograms, boxplots, or density function plots, can support the identification process.

The Data Workflow

Rattenbury et al. (2017) define a holistic data workflow framework using three different stages. The data flows from the raw stage to the refined stage and ends in the production stage. Each stage contains a set of individual activities which are described below. As shown in the figure below, the data flow from top to bottom. This is a simplified way of displaying the process since data often have a bidirectional flow or contain feedback loops.

Figure 22: The Data Workflow



Source: Niklas Wietreck (2021), based on Rattenbury et al. (2017).

In the raw stage, three actions can be defined: ingest data, describe data, and assess data utility. These actions mainly aim to ingest the data and the creation of metadata is the second priority. During the ingestion action, the data are brought into the system. The process is highly dependent on the data source since there are a variety of formats. Data can be provided in simple files (e.g., comma-separated values [CSV]) or in complex and extensive databases. Proprietary ~~Extract Transform Load~~ (ETL) tools, such as Informatica, Alteryx, Data Factory, Feature Manipulation Engine (FME), and Talend, are often used during this action to access and transfer the data. These tools create additional value by executing initial data transformations, such as mappings, to the existing data structure (Rattenbury et al., 2017). The data that are imported at this point are often already well known. The actions become more complex when organizations try to import unknown data. This requires close communication with domain experts who understand the data best, the proper description of the data, the specifications of the characteristics, and the assessment of the utility. Both actions will lead to the creation of metadata.

Extract Transform Load
These operations describe the variety of actions executed in the three stages of data ingestion, processing, and preparation.

When describing data, there are different dimensions that can be used to create a better understanding. The structure of a dataset refers to the format, the encoding of the fields, and the values. Traditionally, datasets are presented in a table format with a set number of rows and columns, which results in a consistent data structure. When the structure has a more heterogeneous structure, such as a JavaScript Object Notation (JSON) or Extensible Markup Language (XML) format, the format is not consistent and needs to be described

accordingly (Rattenbury et al., 2017). When talking about encoding the dataset, analysts are often challenged with specific date and time zones like Coordinated Universal Time (UTC), text encoding like UTF-8, and values in different currencies.

Another dimension used to describe the data is granularity, which refers to contained information within each value. Granularity is often described using the terms “coarseness” and “finesness.” Both relate to the level of depth or the number of unique entities that a single record represents. A bank transaction where each transaction has its own entity can be described to have a fine granularity. However, the monthly balance after all transactions have been combined has a coarse granularity.

The accuracy can be used to describe the quality of the data where each field should achieve a certain level of accuracy (Rattenbury et al., 2017). Typical examples of inaccuracies are misspellings, out-of-range numerical values, and missing components within a specific field. To properly assess the accuracy of a given dataset, the scope must be clearly defined.

The scope can be determined by the distinct attributes per dataset field, or the attribute-by-attribute population coverage. The former is related to, e.g., a timestamp when a customer has added an item to the cart. Additional details within the row will then create a distinction between each entry. The latter checks whether every attribute of each field is present, or whether some have been excluded for any reason.

Another dimension to assess is the temporality, which refers to the time the data are presented. For time-sensitive datasets, it is crucial to represent the correct state of the dataset (Rattenbury et al., 2017). If, for example, the balance of a bank account represents the status a month ago and the customer continues to spend money, this can result in problems for both the customer and the bank. Therefore, the timeframe the dataset considers must be assessed and described. This will reduce the number of inaccuracies and enhance efficient processes. These dimensions will help to describe the data and create an initial set of metadata, which will ultimately help to create a multi data source environment.

In the refined stage, canonical data are created in order to build **ad-hoc** reports and prototypes. This includes a number of different exploratory analyses. While the initial ingestion of data included as few data transformations as possible, canonical data are adjusted to meet the demand of future analyses. This adjustment includes a variety of transformation steps that are highly dependent on the created metadata (Rattenbury et al., 2017). During this step, deficiencies of each dimension defined in the previous step will be addressed.

Structural issues are often addressed by creating a solid tabular structure. This is the case for hierarchical unstructured data. A solid tabular structure might include a large number of data transformations, since a structure has to be defined and the respective values have to be accessed and stored. The creation of additional fields by separating categorical data can also alter the initial structure of the dataset.

Granularity issues are addressed by building a dataset on the finest granularity required for the analysis. While it is recommended to keep the finest granularity available, a specific analysis may only require an aggregation of the dataset (Rattenbury et al., 2017). The

dataset for a monthly balance report of every customer does not need to represent each individual record. In this case, a coarser granularity can derive a result faster. When handling inaccurate entries, the following three strategies are used:

1. The records containing inaccurate values are removed.
2. The inaccurate records are retained, but an additional field is created to mark them as inaccurate.
3. The inaccurate values are replaced with default or estimated values.

The complexity in this step comes from the definition of accurate data, which was done in the previous stage. Scope related issues are handled by including the full set of records and fields in the dataset. This can include the combination of different datasets to form a new dataset that includes all required information (Rattenbury et al., 2017).

After addressing these issues, the two analytical actions can be executed. The creation of ad hoc reports helps answer specific question using the created data. Classical reporting, as done by business intelligence units, can be seen as an example of such actions. These analyses mainly examine the current data and answer questions about the past and the present. This is likely to result in indirect value through supporting people in their decision-making process (Rattenbury et al., 2017). The questions that will be answered by the ad hoc report are highly use case specific and can range from “How many products did I sell in Q1?” to “Where are the majority of my users from?” While those ad hoc reports focus on the past, modeling, simulation, and forecasting analysis look into the future. “How many sales will product X generate if I use marketing strategy Y?” Such a question could be roughly estimated using historical data in combination with forecasting models or simulations. The aim of the model is to understand how different factors are influencing the outcome and how this can be controlled.

In the final stage, the production stage, the aim is to create production data and build an automated system. The prototypes and ad-hoc analysis from the previous stage will be used to build robust reporting systems while excluding the one-time analyses. This robust system needs to be built on an optimized data structure, which is then scheduled and monitored to create continuous reports and data products. Optimizing the refined data includes steps to simplify and specify the data as much as possible. This can include specifications related to the resources used for storage or processing. Those specifications also aim to create a dataset that is optimized for a narrow set of analyses (Rattenbury et al., 2017).

When designing a regular report or an automated product, it is important to continuously monitor the flow of data and ensure that the aforementioned constraints on the dimensions (structure, accuracy, etc.) are in place. This can be challenging since newly inserted data might vary from the historical dataset. Those diverting values can be addressed by generalizing the logic of the data wrangling process as much as possible. This can include the extension of value ranges or the handling of duplicate records (Rattenbury et al., 2017).

This process has shown, in a generalized way, how a data workflow can look. It is important to note that this is merely a framework and needs to be modified to fit the use case at hand.

5.2 Cleansing

Even though clean data are the fundament of every analysis, the process of cleansing a dataset is not considered as glamorous as, for example, the creation of an in-depth analysis. Nevertheless, it is crucial to execute efficient data cleansing since the input data are rarely ready for use. There is a need for a consistent and clean format, especially when working in an multi source environment. This can start right at the beginning when data are not correctly joined because the join key is not formatted the same way in both tables. The focus of this section will be on handling duplicates, detecting outliers, matching entries using fuzzy logic, standardizing and normalizing a dataset, and data imputation (Kazil & Jarmul, 2016).

Remove Duplicates

When working with a new dataset, specific entries may appear to occur more than once in a dataset. There are numerous reasons for this, and can, in many cases, be solved using Python (Kazil & Jarmul, 2016). The pandas module contains several methods that support the process of handling duplicates. The following pandas DataFrame contains values for different combinations of first and last names, as well as age:

Code

```
import pandas as pd
df = pd.DataFrame({
    'firstname': ['Peter', 'Peter', 'Bruce', \
                 'Bruce', 'Bruce'],
    'lastname': ['Parker', 'Parker', 'Parker', \
                 'Banner', 'Banner'],
    'age': [24, 24, 23, 35, 25]
})
print(df)
```

```
# console output:
#  firstname lastname age
#  0    Peter   Parker   24
#  1    Peter   Parker   24
#  2    Bruce   Parker   23
#  3    Bruce   Banner   35
#  4    Bruce   Banner   25
```

When applying the `drop_duplicates()` method, every row is checked for duplicates. Since the rows with the index 0 and 1 are identical, the first occurrence is kept and the second one is deleted. This results in a smaller output DataFrame. When examining the index, we can see that the original index was kept.

Code

```
print(df.drop_duplicates())
```

```
# console output:  
#  firstname lastname age  
# 0    Peter   Parker   24  
# 2    Bruce   Parker   23  
# 3    Bruce   Banner   35  
# 4    Bruce   Banner   25
```

While this will compare every single field of the DataFrame, it is possible to apply this method based only on specific columns. The parameter `subset` allows users to submit a list of fields that should be included in the check for duplicates. The following example will therefore only check for duplicates in the column “firstname.” Only the first occurrence will be kept, which will result in a small new DataFrame.

Code

```
print(df.drop_duplicates(subset=['firstname']))
```

```
# console output:  
#  firstname lastname age  
# 0    Peter   Parker   24  
# 2    Bruce   Parker   23
```

The example above only focused on one additional field, while the example below will also include the field “lastname” in the definition of a duplicate. Therefore, all unique combinations of first and last name will be returned in the final DataFrame. By adding an argument, `keep`, users can define which occurrence should be kept. By default, the first occurrence will be kept, but `last` can also be selected as shown in the example below. It is also possible to use `False` as `keep`, which will drop all duplicates.

Code

```
print(df.drop_duplicates(\n    subset=['firstname', 'lastname'], \n    keep='last'))
```

```
# console output:  
#  firstname lastname age  
# 1    Peter   Parker   24  
# 2    Bruce   Parker   23  
# 4    Bruce   Banner   25
```

Detection of Outliers and Bad Data

The detection of outliers and bad data is one of the most challenging tasks for any analyst. It is often unclear at the beginning which values are correct, which data values are errors, and which values are correct but can be considered outliers. To answer these questions, it is important to understand the dataset as a whole, including the data acquisition process (Kazil & Jarmul, 2016). When examining a dataset that contains the number of daily visits at a German supermarket, missing values can, most likely, be found for one day per week on a consistent basis. In Germany, most supermarkets are closed on Sundays, so these “missing” data are actually correct.

Bad data are likely created during the data acquisition process by putting incorrect entries into the dataset. Entries including “Null” values are a common example of data being incorrectly or incompletely stored in the database (Kazil & Jarmul, 2016). One example is an online survey with several questions that are not filled out until the end because the website broke during the process. In the underlying database, some of the fields could be stored while the missing fields will be marked as “null”, not available (NA), or not a number (NaN). The analyst of the questionnaire now has to decide how to handle these bad data. If there is no column indicating that the questionnaire was completed, a solution can be removing the rows that are missing a certain number of values.

The following example creates a DataFrame filled with different numerical values and uses the capabilities of NumPy to simulate “NaN” values. When using the method `dropna()`, all entries are deleted where at least one “NaN” value is present. This process is also called list-wise imputation. This results in a DataFrame in which three out of the four initial rows are deleted.

Code

```
# load packages
import pandas as pd
import numpy as np
from numpy import nan as NA

# generate sample data
from numpy import nan as NA
df = pd.DataFrame([\
    [1.3, 2.3, 5.2], \
    [2.1, NA, NA], \
    [NA,NA,NA], \
    [NA, 9.2, 2.3]])

# print the raw data
print(df)

# console output:
#      0    1    2
# 0  1.3  2.3  5.2
# 1  2.1  NaN  NaN
```

```
# 2 NaN NaN NaN
# 3 NaN 9.2 2.3

# print only rows of non-missing values
print(df.dropna())

# console output:
#      0    1    2
# 0 1.3 2.3 5.2
```

While it can be helpful to delete every row that contains a NaN value, NaN values should often be deleted in a more flexible manner since a NaN value is not automatically a bad value. By adding the argument `how` with a value of `all`, only the rows where every single field contains a NaN value are deleted. This returns a DataFrame that deleted the row with the index two, which only contained NaN values before.

Code

```
print(df.dropna(how='all'))

# console output:
#      0    1    2
# 0 1.3 2.3 5.2
# 1 2.1 NaN NaN
# 3 NaN 9.2 2.3
```

In some cases, it is necessary to delete rows that contain a certain amount of “NaN” values. In the following example, all rows that contain two or more “NaN” values are deleted. This results in a DataFrame where the rows with the index one and two have been deleted.

Code

```
print(df.dropna(thresh=2))

# console output:
#      0    1    2
# 0 1.3 2.3 5.2
# 3 NaN 9.2 2.3
```

Similar to the example for the `drop_duplicates()` method, there is an argument `subset` that deletes the rows based on a specified field. The following example deletes the rows that contain a “NaN” value in the first column. This results in a DataFrame that deleted the rows with the index two and three.

Code

```
print(df.dropna(subset=[0]))

# console output:
```

```
#      0      1      2
# 0  1.3  2.3  5.2
# 1  2.1  NaN  NaN
```

There are many statistical methods to identify outliers, and two are commonly used. One is based on the standard deviation, while the other one uses the median absolute deviations. The following example will use the Z-score, which is the signed number of standard deviations by which a certain entry is above the mean value. The Z-score can be defined as $Zscore = (x - mean) / \text{standard deviation}$ and aims to describe the distance of each value with respect to the center of the distribution of values (Kazil & Jarmul, 2016). Through calculating the Z-score, the data are rescaled and centered. Then, a threshold (often two or three Z-score units) is used to calculate how high the deviation must be in order to be considered an outlier. The following code uses a given list of values and then uses the capabilities of NumPy to calculate the mean and the standard deviation.

Code

```
# generate the sample data
data = [15, 22, 12, 22, 13, 21, 11, \
        140, 12, 22, 21, 33, 21, 11, 22]

# calculate the mean and standard deviation
mean = np.mean(data)
std = np.std(data)
```

In a second step, we calculate the Z-scores and apply a threshold in order to define what is considered as an outlier.

Code

```
# calculate z-scores
data_z = (data - mean)/std

# apply threshold
threshold = 3
data_z[data_z >= threshold]

# console output:
# [3.6733965]
```

We see that there is one value which, based on its Z-score of about 3.67, is considered an outlier. To extract the original value, we can execute the following code:

Code

```
from itertools import compress
list(compress(data, data_z >= threshold))
```

```
# console output:
```

```
# [140]
```

While this example aims to display the way the Z-score works, there are a variety of implementations in different modules. In the module SciPy, there is a stats class that contains a method, `zscore`, that computes the Z-score for each value. Additionally, it contains some useful parameters to handle “NaN” values. Normalization or standardization can also be used to identify outliers.

Fuzzy Matching

When combining two different datasets, there is sometimes a join key that is not exactly the same. A simple string comparison, for example, will not be able to match the text “2 rich girls” to “Two rich Girls,” even though they mean the same thing. To solve this problem, fuzzy matching can be used to determine that both elements are the same by calculating their differences using specific algorithms. The following code creates a list of records where each element is a dictionary. The dictionary contains strings for a song, movie, and book title. In the second step, the `ratio()` method from the `fuzzywuzzy` module is used to calculate the similarity between the two strings (Kazil & Jarmul, 2016). A high similarity is indicated with a value close to 100, while a low similarity will have a value close to zero. The output shows that, although the spelling of the words is different, the algorithm gives them a high similarity score.

Code

```
# load packages
from fuzzywuzzy import fuzz

# generate sample data
my_records = [{ \
    'song': 'Sound of Silence', \
    'movie': '2 fast 2 furious', \
    'book': 'The Hitchhikers Guide to the Galaxy', \
}, { \
    'song': 'The sound of silence', \
    'movie': 'Too fast too furious', \
    'book': 'The Hitchhikers Guide to the Galaxie', \
}]

# calculate the overall similarity between the
# two records
fuzz.ratio(my_records[0], my_records[1])

# console output:
# 91
```

```

# calculate the similarity between songs
fuzz.ratio(my_records[0]['song'], \
           my_records[1]['song'])
# console output:
# 78

# calculate the similarity between movies
fuzz.ratio(my_records[0]['movie'], \
           my_records[1]['movie'])
# console output:
# 78

# calculate the similarity between books
fuzz.ratio(my_records[0]['book'], \
           my_records[1]['book'])
# console output:
# 96

```

When using the `partial_ratio()` method instead of the `ratio()` method, the algorithm focuses on substrings. This results in a higher similarity score since it does not matter if some words are missing or if the punctuation is different. As the example below shows, all comparisons have improved their similarity score.

Code

```

# calculate the partial similarity between songs
fuzz.partial_ratio(my_records[0]['song'], \
                  my_records[1]['song'])
# console output:
# 88

# calculate the partial similarity between movies
fuzz.partial_ratio(my_records[0]['movie'], \
                  my_records[1]['movie'])
# console output:
# 81

# calculate the partial similarity between books
fuzz.partial_ratio(my_records[0]['book'], \
                  my_records[1]['book'])
# console output:
# 97

```

Fuzzy matching can be used in a variety of scenarios, such as joining datasets. Fuzzy-Wuzzy, for example, contains additional methods for sorting strings (`fuzz.token_sort_ratio()`) and assigning a string to the most similar string from a list (`fuzz.process.extract()`).

Normalization and Standardization

From a statistical perspective, normalization creates a new value that is set on a particular scale. If you consider the results of a basketball game, there is a high deviation of how many points a team could have. This score per team could be put on a win or loss scale based on whether or not the team had more points than the opposing team. While this is more of a categorization, a normalization would include putting the scores on a scale from zero to one with high scores close to one and low scores close to zero. Based on this newly created scale, the distribution can be calculated to identify percentiles or outliers. Another reason that we apply standardization or normalization is to achieve comparability between features on very different scales.

There are many ways to achieve this, but we broadly distinguish between standardization and normalization. While the former shifts the data to a scale on which the distribution's mean is at zero and the standard deviation is one, the latter rescales the data to values between zero and one.

This normalization can be applied in Python using the `MinMaxScaler` from the `scikit-learn` module (Kazil & Jarmul, 2016). It will scale all values by putting them in a range between zero and one. Before applying the `MinMaxScaler`, we import the required modules and generate the following sample NumPy array as a test dataset:

Code

```
# load packages
import numpy as np
from sklearn.preprocessing import MinMaxScaler

# generate the sample data
data = np.array([ \
    [201, 0.003], \
    [14, 0.09], \
    [40, 0.003], \
    [78, 0.02], \
    [3, 0.3] \
])
```

Now, the scaler can be instantiated and applied to the data. This application to the data is done using the `fit_transform()` method, which firsts fits the scaler to the data, and then transforms them directly. As a result, all values are now scaled in the range of zero to one.

Code

```
scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(data)

print(data_scaled)
# console output:
```

```
# [[1.          0.          ]
# [0.05555556 0.29292929]
# [0.18686869 0.          ]
# [0.37878788 0.05723906]
# [0.          1.          ]]
```

Standardization, in contrast to normalization, uses the mean for a value of zero, while one represents a standard deviation of one. From an implementation point of view, it follows the same process as the `MinMaxScaler`. An implementation can be found in form of the `StandardScaler`, which is available in the preprocessing class of the scikit-learn module. The example below executes it in the same way as for the `MinMaxScaler`.

Code

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

print(data_scaled)
# console output:
# [[ 1.86587833 -0.70939126]
# [-0.74188884  0.06014789]
# [-0.37931159 -0.70939126]
# [ 0.15060901 -0.55902154]
# [-0.89528691  1.91765618]]
```

Standardization and normalization are often used to prepare datasets for their use in machine learning models.

Data Imputation

The process of filling missing data or replacing inaccurate data with default or estimates is called data imputation. If, in a sales dataset, it was identified that extremely high values are incorrect, they could be exchanged by a set maximum value to insure unbiased aggregations calculations (Kazil & Jarmul, 2016).

The following DataFrame contains four fields that contain random numbers in eight rows. Some of the randomly created values have been exchanged with “NaN” values using the `iloc` method in combination with the capabilities of NumPy.

Code

```
# load packages
import numpy as np
import pandas as pd

# generate a sample dataset
df = pd.DataFrame(np.random.randn(8,4), \
                  columns = ["zero", "one", "two", "three"])
```

```

# add missing values
df.iloc[:6,0] = np.nan
df.iloc[:4,1] = np.nan
df.iloc[:2,2] = np.nan

# show the sample data
print(df)

# console output:
#      zero      one      two      three
# 0      NaN      NaN      NaN -0.722502
# 1      NaN      NaN      NaN  0.461928
# 2      NaN      NaN -1.123492 -0.870373
# 3      NaN      NaN  0.438706 -0.410326
# 4      NaN  1.274561 -0.073225 -0.553929
# 5      NaN  0.569232  1.088357 -0.353911
# 6 -0.065522  0.666667 -1.753920 -2.093555
# 7 -0.558569 -1.302122  0.092451  1.122218

```

Based on this DataFrame, the missing data need to be imputed. The following code exchanges every “NaN” value per field with a specified value using the `fillna()` method. All “NaN” values in column “zero” are replaced by the fixed value of zero, “NaN” values in column “one” are replaced by the maximum value of the same column, and “NaN” values in column “two” are replaced by the mean. This is called substitution by default value, maximum value, and mean, respectively.

Code

```

df.fillna({ \
    "zero":0, \
    "one":df.one.max(), \
    "two":df.two.mean() \
})

      zero      one      two      three
0  0.000000  0.389163 -0.556765  0.263473
1  0.000000  0.389163 -0.556765  1.241951
2  0.000000  0.389163 -0.377334 -0.096307
3  0.000000  0.389163 -2.108044 -0.608818
4  0.000000  0.389163 -0.827452  0.660163
5  0.000000  0.216780  0.957897 -0.071680
6  1.575033 -0.458397 -1.239138 -0.492136
7 -1.324215 -0.605377  0.253480 -1.057012

```

Another simple imputation method is mode substitution. This method substitutes missing values with the most frequent value in this column. This method is particularly useful for categorical data, but can also be applied to numerical data. To see this method in action, we generate another sample dataset.

Code

```
df = pd.DataFrame({ \
    "city": ["tokyo", None, "london", \
            "seattle", "san francisco", "tokyo"], \
    "boolean": ["yes", "no", None, \
               "no", "no", "yes"], \
    "ordinal_column": ["somewhat like", "like", \
                      "somewhat like", "like", \
                      "somewhat like", "dislike"], \
    "quatitative_column": [1, 11, -0.5, 10, None, 20]})
```

We want to use mode substitution to impute the missing value in the city column. To do this, we identify the value that occurs most frequently in this column (the mode).

Code

```
sub_val = df['city'].value_counts().index[0]

print(sub_val)
# console output: 'tokyo'
```

By using the `fillna()` method, all “None” values can be filled with the most common value. This will result in a “city” column, which does not contain any “None” values.

Code

```
df_sub = df.fillna({'city': sub_val})

print(df_sub.city)
# console output:
# 0          tokyo
# 1          tokyo
# 2         london
# 3         seattle
# 4  san francisco
# 5          tokyo
# Name: city, dtype: object
```

In addition to mode substitution, we have already seen an example of mean substitution. Finally, let us notice that mean substitution can also be applied using the “SimpleImputer” in the scikit-learn library.

Code

```
# generate sample data
df = pd.DataFrame({ \
    'age': [34, 27, 89, 23, 34, None, 32], \
    'height': [178, 173, 198, 167, None, 174, 165]
})

# apply mean substitution using the SimpleImputer
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='mean')
imputer.fit_transform(df)

# console output:
# array([[ 34.         , 178.         ],
#        [ 27.         , 173.         ],
#        [ 89.         , 198.         ],
#        [ 23.         , 167.         ],
#        [ 34.         , 175.83333333],
#        [ 39.83333333, 174.         ],
#        [ 32.         , 165.         ]])
```

Apart from these very simple substitution techniques, we already saw list-wise imputation, which is simpler, but also more wasteful, as whole data rows are dropped. There are various imputation techniques that are usually more efficient and elaborate, but are beyond the scope of this course book. These techniques include multiple imputation techniques, which use information from other columns. The most popular among these techniques are nearest neighbors imputation, matrix factorization, multiple imputation by chained equations (MICE), multiple imputations with denoising autoencoders (MIDAS), and geo-interpolations.

5.3 Enrichment

There are a variety of approaches to the field of data enrichment. This section will first focus on different aspects of feature engineering, including feature construction, feature transformation, and feature learning. In the second part of this section, different methods for adding new data, such as joins or unions, will be presented.

Feature Construction

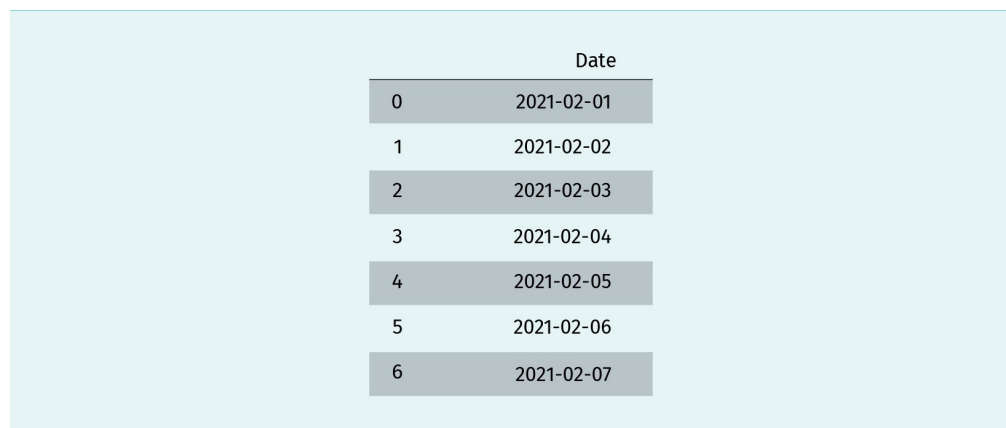
The process of feature construction is used to derive new features based on what is already given within a column in order to enrich the dataset. This can include the transformation of existing features and the addition of new features from other sources. The following example shows how new features are derived from a date column. The following DataFrame contains the first week of February, 2021. The `date_range()` method is used

to build these data by handing over the start date, the amount of entries that should be returned, and the frequency, which is set to “D” for day. Afterwards, the created data are put into the DataFrame.

Code

```
values = pd.date_range("2021-02-01", periods=7, freq='D')
df = pd.DataFrame(values, columns=["Date"])
```

Figure 23: Date DataFrame Raw



	Date
0	2021-02-01
1	2021-02-02
2	2021-02-03
3	2021-02-04
4	2021-02-05
5	2021-02-06
6	2021-02-07

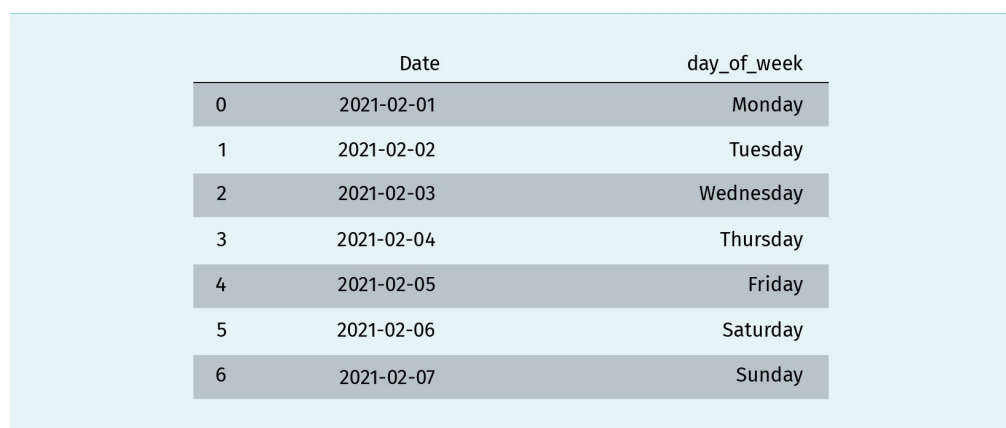
Source: Niklas Wietreck (2021).

In the next step, a new feature is constructed. In this example, the `day_name()` method is used to extract the name of the day. The newly created feature is stored in a new column.

Code

```
df['day_of_week'] = df['Date'].dt.day_name()
```

Figure 24: Date DataFrame with Day of the Week



	Date	day_of_week
0	2021-02-01	Monday
1	2021-02-02	Tuesday
2	2021-02-03	Wednesday
3	2021-02-04	Thursday
4	2021-02-05	Friday
5	2021-02-06	Saturday
6	2021-02-07	Sunday

Source: Niklas Wietreck (2021).

Another useful feature is a column that indicates whether a day is a weekday or weekend day. The following script takes the date value and returns the index value of the date within the week. By applying an integer division of five, it is checked if the value is one, which indicates the weekend.

Code

```
df.weekday = pd.DatetimeIndex(df.Date).dayofweek // 5
```

Figure 25: Date DataFrame with Weekday

	Date	day_of_week	weekday
0	2021-02-01	Monday	0.0
1	2021-02-02	Tuesday	0.0
2	2021-02-03	Wednesday	0.0
3	2021-02-04	Thursday	0.0
4	2021-02-05	Friday	0.0
5	2021-02-06	Saturday	1.0
6	2021-02-07	Sunday	1.0

Source: Niklas Wietreck (2021).

The scripts above created new features based on an existing one. By using additional enrichment methods, such as joins, additional information like holidays can be added.

One kind of feature construction method involves the calculation or derivation of values and can be separated into generic and proprietary derivations. Generic derivations can be used to encode temporal or geographical information. When working with geographical data, this can, for example, include transforming an address to latitude and longitude coordinates in order to display it on a map. This example shows that the information is already there, but needs to be enriched in order to be useful. Analysis tools often contain functionalities to handle this kind of enrichment. Proprietary derivations describe custom derivation calculations that are only applicable in specific scenarios. An organization that measures product prices on different e-commerce platforms might have different values for the same product. In a new column, “minimum price,” a function could write the minimum price across all platforms. This newly created column can then be used for other analysis tasks. In analysis tools, values are often stored in user-defined functions (UDFs) and follow a syntax set by the tool (Visochek, 2017).

Feature Transformation

There are a variety of methods that aim to transform the features. These transformations can include the change of a unit (Fahrenheit to Celsius); the combination of values into bins, like in a histogram, to understand the underlying distribution; and different statistical transformations to prepare the data for further analysis methods, such as machine

learning. The following examples will introduce these different types of transformation methods. The first example simply applies a function to the given DataFrame, which transforms a temperature value available in Celsius into Fahrenheit. After the DataFrame is created, a new column is defined, which takes the Celsius value and recalculates it using a formula

Code

```
# load libraries
import pandas as pd

# create sample data
df = pd.DataFrame([30.5, 23.2, 16.6, \
                  8.5, 0.1, -6.5], \
                  columns = ["temp_C"])

# transform units
df["temp_F"] = (df["temp_C"] * 9 / 5) + 32
```

Figure 26: Celsius to Fahrenheit Transformation

	Temperature in Celsius	Temperature in Fahrenheit
0	30.5	86.90
1	23.2	73.76
2	16.6	61.88
3	8.5	47.30
4	0.1	32.18
5	-6.5	20.30

Source: Niklas Wietreck (2021).

The next example groups values into a predefined number of bins. After a DataFrame is filled with random numbers, the `cut()` method can be used to divide the values in to four groups as defined in the `bins` parameter. The ranges of the different groups are returned. The first group ranges from 0.902 to 25.5. By adding the `value_counts()` method, it is possible to identify the number of values contained in each bin. In this example, the most values are contained in the bin with a range of 74.5 to 99.0. Based on this, the distribution can be analyzed.

Code

```
import numpy as np
df = pd.DataFrame(np.random.randint(0,100, \
                                  size=(100)), columns=["Values"])

pd.cut(df['Values'], bins=4).value_counts()
```

```
#Output
# (74.5, 99.0]    33
# (0.902, 25.5]  27
# (25.5, 50.0]   20
# (50.0, 74.5]   20
# Name: Values, dtype: int64
```

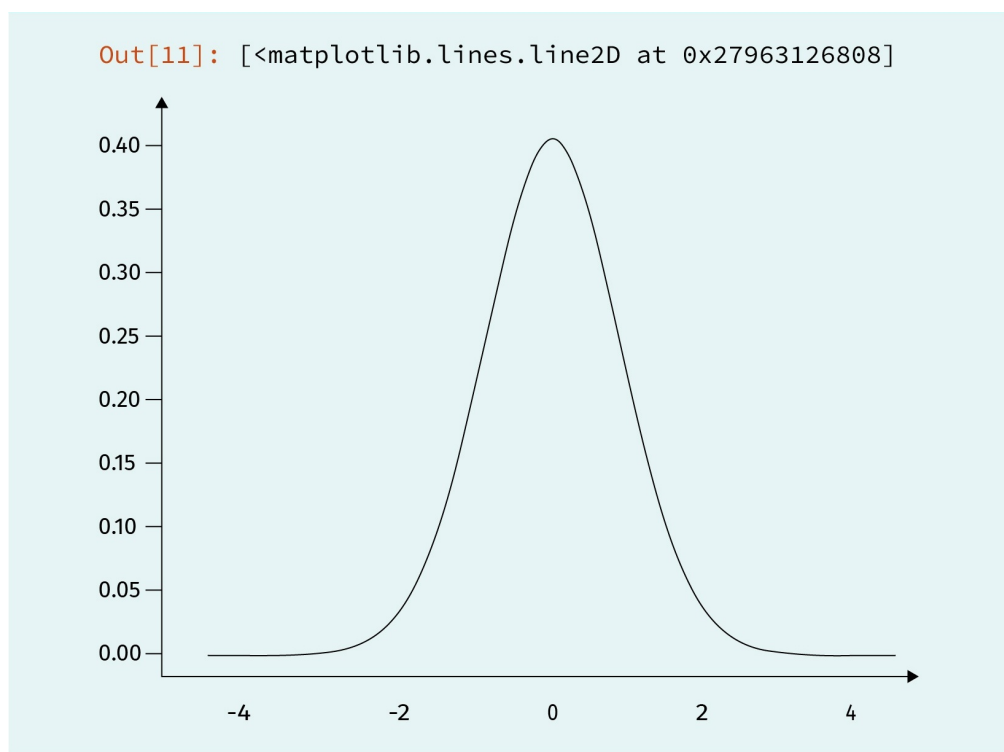
Another group of methods that transform the data is linked to the distribution of the data. When working with data, it is important to know how each individual datapoint is distributed along the value range. One important type of distribution to know is the normal distribution (or Gaussian distribution). It describes a probability distribution that is symmetric around the mean. This means that values occur more frequently around the mean, which will result in a bell curve as displayed in the figure below. In a normal distribution, there are two parameters: the mean and the standard deviation. In the normal distribution, 68 percent of the data values are within one standard deviation, 95 percent are within two standard deviations, and 99.7 percent are within three deviations. A figure like the one below can be created using the following code:

Code

```
# load libraries
from scipy import stats
from matplotlib import pyplot as plt

# create a normally distributed dataset
x_data = np.arange(-5, 5, 0.001)
y_data = stats.norm.pdf(x_data, 0, 1)
plt.plot(x_data, y_data)
```

Figure 27: Normal Distribution



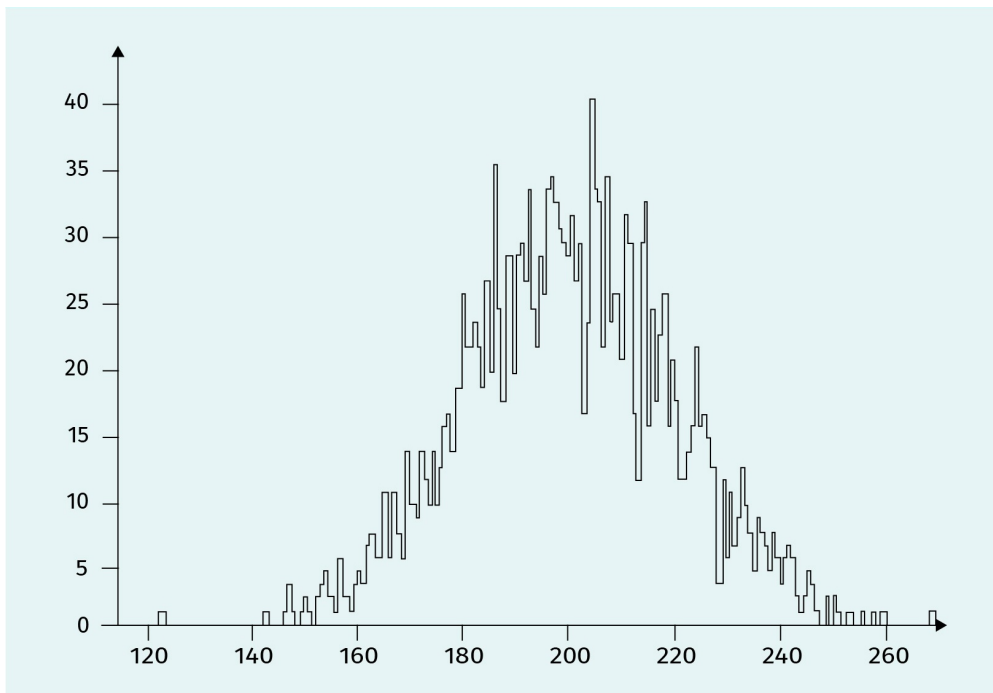
Source: Niklas Wietreck (2021).

The following example will demonstrate how to check for normal distribution. The data for this example are randomly created using the capabilities of NumPy, and will return a histogram that shows the occurrences of the values in a set number of bins.

Code

```
data = 20 * np.random.randn(2000) + 200  
plt.hist(data, bins = 200)
```

Figure 28: Random Numbers for Normal Distribution Test



Source: Niklas Wietreck (2021).

The figure above is reminiscent of the bell-shaped curve in the previous example. Nevertheless, we can see that there are a few gaps in between the individual bins. While this visualization helps to define whether or not it is a normal distribution, this must be described with further statistics. The Shapiro-Wilk test will be used as one statistical testing method. This is optimized for testing a normal distribution and calculates the difference between the expected and actual deviation. This test is made available in the SciPy module and will return two values. The `stat` value returns a test statistic and the `p` returns the corresponding `p`-value. In this example, the `p`-value is higher than a threshold value (normally 0.05), which indicates that we do not have sufficient evidence that the data are not normally distributed. This is not surprising, since the data were created using the NumPy `randn()` method, which creates values according to the normal distribution.

Code

```
from scipy.stats import shapiro
stat, p = shapiro(data)
print(stat, p)

0.9993932247161865 0.7986587882041931
```

Another assumption that can be made about the data is homoscedasticity, which is especially crucial in linear regression models. Homoscedasticity describes a scenario in which the variance is the same across the whole dataset. An example of the exact opposite, heteroscedasticity, is the annual income of people when split by age. While teens and young

adults often earn the minimum wage since they are untrained and inexperienced, salaries often increase in their 20s and 30s, depending on their education. A person who worked as a newspaper deliverer as a teen could continue doing this, or they could become a highly paid lawyer. Therefore, the variance over the years increases and makes the dataset heteroscedastic. Depending on the data and the presence of homoscedasticity, different models can be used to address this conditional variance. One method that works especially well when data are discrete and non-negative is the negative binomial regression model. It can be categorized as a generalized linear model (GLM) and can be implemented using the statsmodel module in Python. As the name suggests, it uses a negative binomial distribution, which is able to address non-normal distributions.

The following section will explain three different methods that can be used to transform data that deviate from the normal distribution. First, the square root method will be applied, which is normally used for moderately deviating data. This method mainly reduces the risk of dealing with skewed data while only having a moderate effect on the distribution shape. The following dataset contains four different data columns where each has a different distribution. The example can be downloaded from Github (n.d.-a). In the example, the dataset is read into a DataFrame. Afterwards, a new column “squared” is created, which contains the square root of the first column. In order to visualize the data, two columns are sliced and visualized using the `hist()` method.

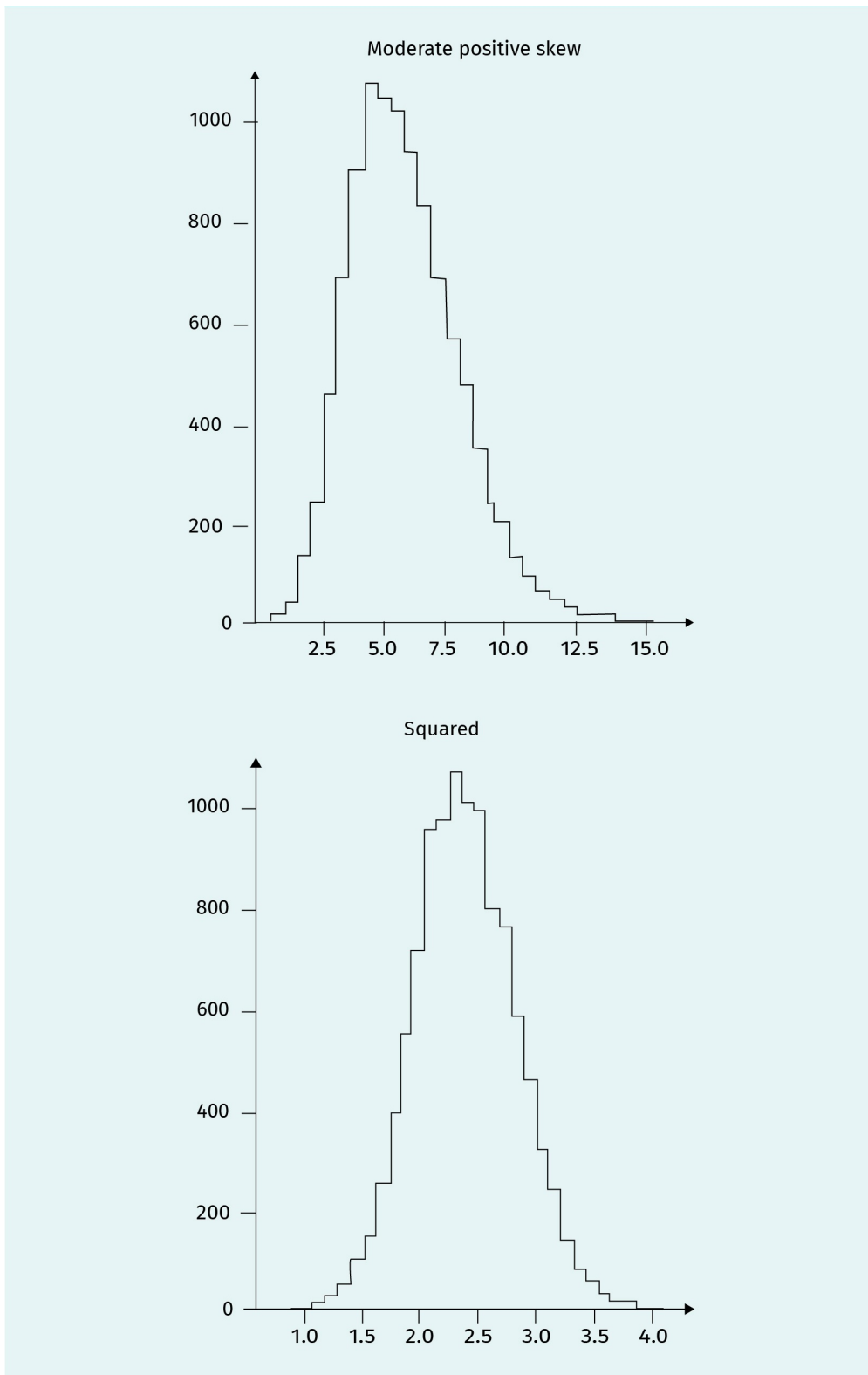
Code

```
# read sample data
url = 'https://raw.githubusercontent.com/marsja/jupyter/'
url += 'master/SimData/Data_to_Transform.csv'
df = pd.read_csv(url)

# apply square transformation
df.insert(len(df.columns), 'Squared', \
         np.sqrt(df['Moderate Positive Skew']))

# create histograms
df_viz = df[["Moderate Positive Skew", "Squared"]]
df_viz.hist(grid=False, figsize=(10, 6), bins=30)
```

Figure 29: Square Root Transformation



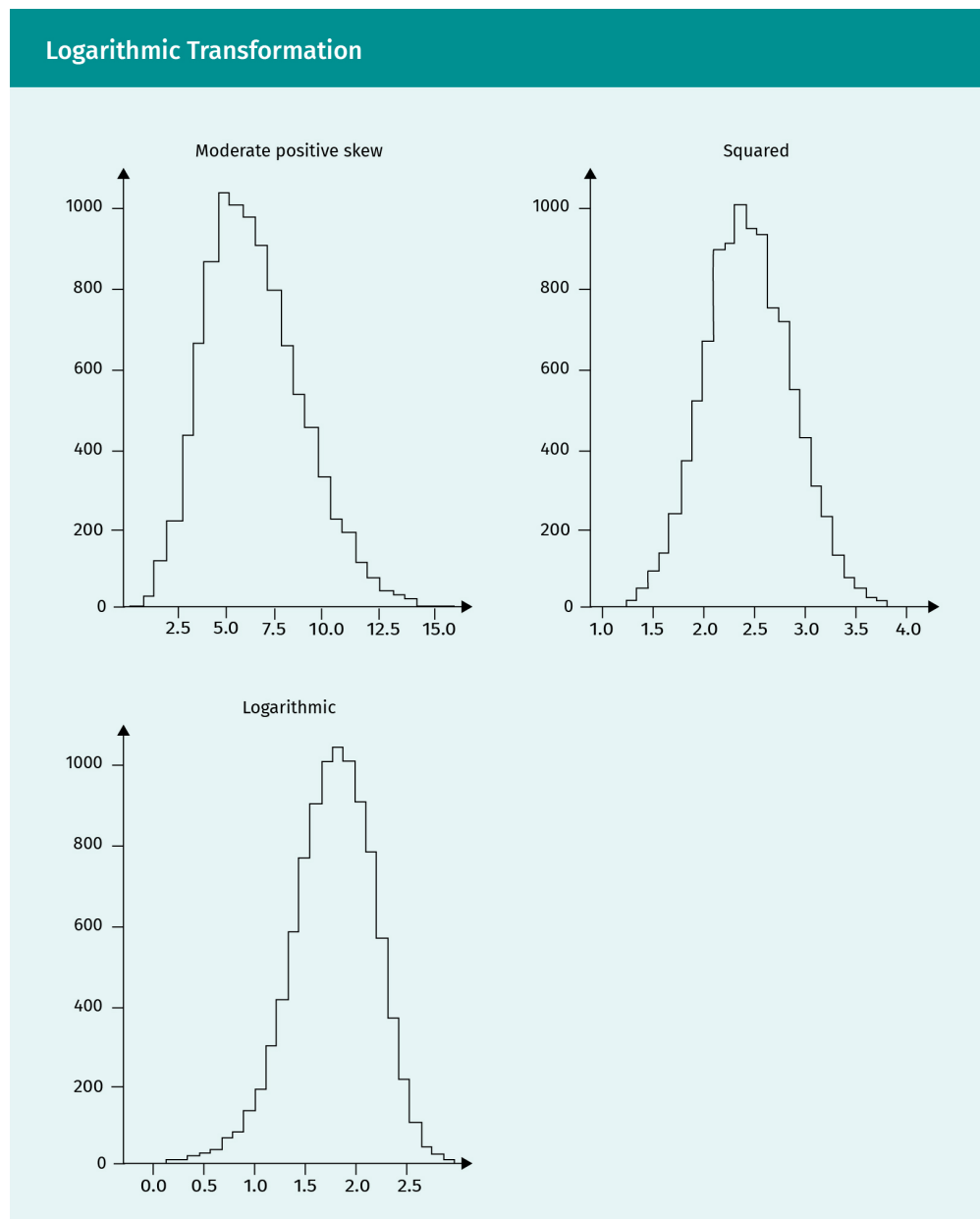
Source: Niklas Wietreck (2021).

Another transformation that has a relatively large effect on the shape of the distribution is the logarithmic transformation. While the square root method was applicable in most scenarios, the `log` method is limited to scenarios where the values are not zero or negative. The example above will be extended with a line of code that uses the `log()` method to transform the data. The figure below shows how the data change in comparison to the input data.

Code

```
df.insert(len(df.columns), 'Logarithmic', \
         np.log(df['Moderate Positive Skew']))
```


Figure 30



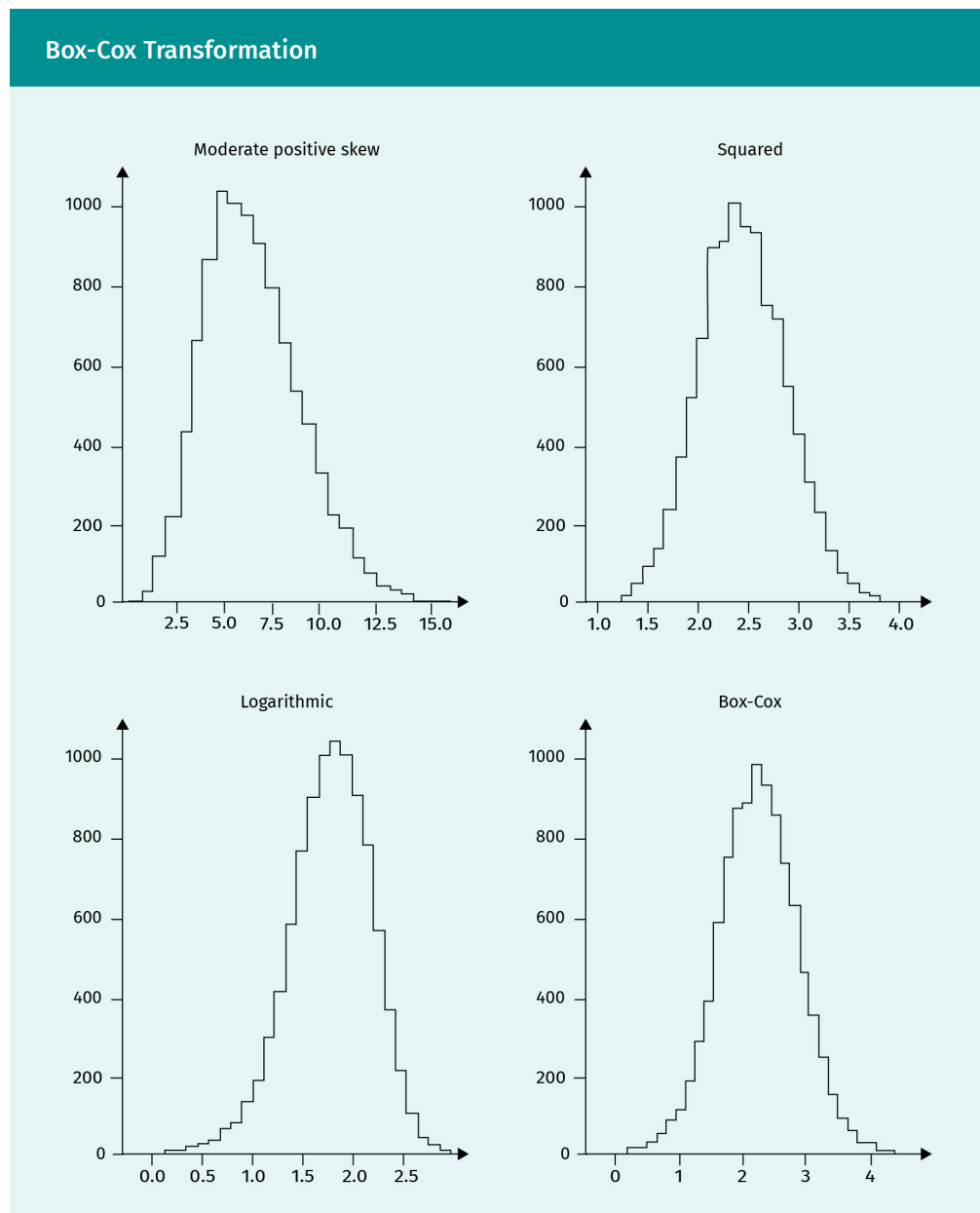
Source: Niklas Wietreck (2021).

The final method is the Box-Cox method. It is also used to transform non-normal data into a normal shape. This method aims to identify a suitable exponent to transform the data. The Box-Cox transformation is implemented in SciPy and needs to be imported. Likewise, it will be added to the example above. Since the return of the `boxcox()` method is a tuple, the first element needs to be selected using square brackets. The resulting transformed data can be seen in the bottom left.

Code

```
from scipy.stats import boxcox
df.insert(len(df.columns), 'Boxcox', \
         boxcox(df['Moderate Positive Skew'])[0])
```

Figure 31



Source: Niklas Wietreck (2021).

After these transformations have been applied, it can be seen that each of them transforms the data slightly differently.

Feature Learning

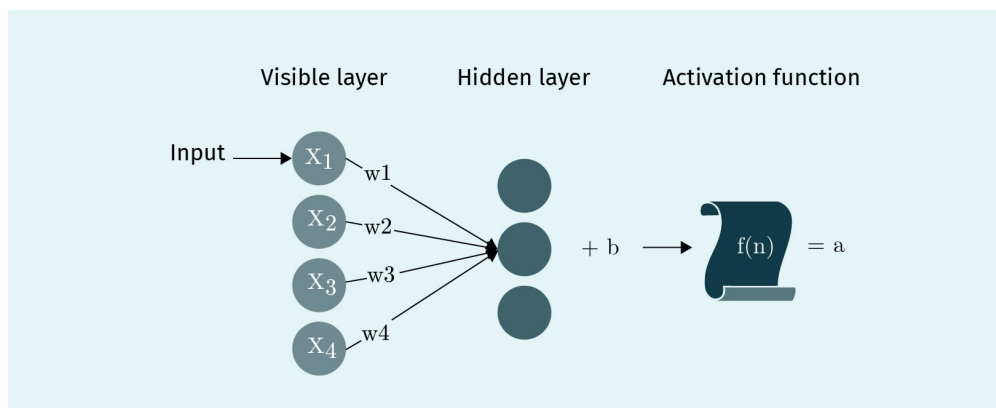
Algorithms in the feature learning class take cleaned data and create brand-new features by identifying latent structures within the data. This is quite similar to feature transformation, though there is a difference in the parametric assumption. This assumption is made by the algorithm about the shape of the transformed dataset. Feature learning algorithms remove this parametric assumption by relying on stochastic learning. Instead of creating a linear equation for each component, feature learning algorithms aim to select the best features to extract by looking at the entries over different epochs (Ozdemir & Susarla, 2018).

To bypass the parametric assumption, deep learning algorithms can be used. The **Restricted Boltzmann Machine (RBM)** is often used in this context to learn a set number of dimensions based on probabilities. In Python, the scikit-learn module offers the BernoulliRBM, which is a non-parametric feature learning algorithm that takes the values of the fields into account (Ozdemir & Susarla, 2018). The following graphic shows the basic structure of the RBM and exemplifies how an input is processed. Each node represents one feature of the dataset, which is weighted based on a defined value for each feature. The weighted sum is added to a bias variable, sent to an activation function and stored in a value a .

Restricted Boltzmann Machine

This is an unsupervised learning algorithm that normally uses a two-layer neural network.

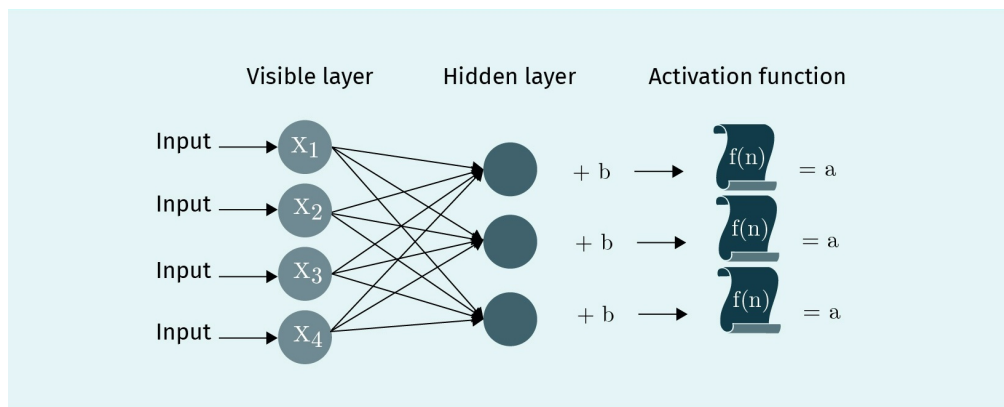
Figure 32: Restricted Boltzmann Machine Value “a”



Source: Niklas Wietreck (2021).

In reality, these nets are more complex and interconnected, as shown in the figure below. There, the hidden layer connects the weighted sum of each value, creating a more complex model.

Figure 33: Restricted Boltzmann Machine Complete Structure



Source: Niklas Wietreck (2021).

This is only one brief conceptual example of how neural networks can be used for feature learning. A full description of this topic is beyond the scope of this unit.

Adding Data

There are a variety of methods that deliver additional information to a dataset. These methods can be separated into three different groups: unions, joins, and calculation of new fields (Kazil & Jarmul, 2016). Unions append additional entries to an existing dataset. This is basically combining two highly related datasets by stacking them on top of each other. One common use case is when new data are created. A website, for example, creates different types of user interactions over time, which are stored in a database (clicks on buttons, purchase of a product, viewing time of a video, etc.). When new data arrive (through a user interaction or as bulk after a set period), it is appended at the end of the dataset in order to increase the available information.

In the following example, there are two DataFrames. The first one, `df_total`, contains the data that were collected for the aforementioned website example on the twenty-first of January, 2021. The next day, two users created new interactions (stored in `df_new`), which should now be combined with the previous dataset.

Code

```
df_total = pd.DataFrame([\
    ["21.01.2021", "UI-92716", "Banner click"], \
    ["21.01.2021", "UI-92716", "Video view"], \
    ["21.01.2021", "UI-92717", "Video view"], \
    ["21.01.2021", "UI-92717", "Form entry"], \
    ["21.01.2021", "UI-92717", "Purchase"]], \
    columns = ["Date", "User-ID", "Action"])

df_new = pd.DataFrame([\
```

```
["22.01.2021", "UI-92718", "contact form"], \
["22.01.2021", "UI-92719", "Video view"]], \
columns = ["Date", "User-ID", "Action"])
```

Since both DataFrames have the same column name, the `concat()` method of pandas can be used to unite both DataFrames. The result will be a combination of both DataFrames. It needs to be noted that the index is kept from the original dataset if the parameter `ignore_index` is not set to `True`.

Code

```
df_total = pd.concat([df_total, df_new], \
    ignore_index=True)
```

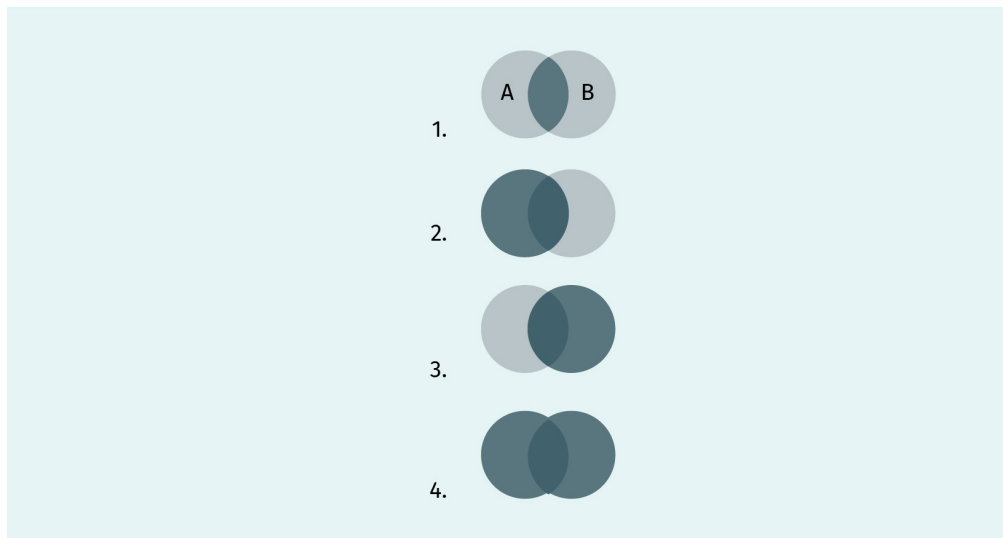
```
print(df_total)
```

```
# console output:
```

```
#      Date  User-ID  Action
# 0 21.01.2021  UI-92716  Banner click
# 1 21.01.2021  UI-92716  Video view
# 2 21.01.2021  UI-92717  Video view
# 3 21.01.2021  UI-92717  Form entry
# 4 21.01.2021  UI-92717  Purchase
# 5 22.01.2021  UI-92718  contact form
# 6 22.01.2021  UI-92719  Video view
# 7 22.01.2021  UI-92718  contact form
# 8 22.01.2021  UI-92719  Video view
```

Joins combine two datasets via a common key. This common key has to be an exact match in order to join the two datasets properly. If there is, for example, a table that contains information about a company but only stores the names of the employees as IDs, another table which contains the employee ID and the name of the employee can be joined to receive the full information. There are four different ways to join datasets. An inner join (1) only returns the data for which keys are available in both datasets. A left join (2) keeps all the records of the initial (left) dataset. This can result in entries with null values when a key was not present in the to be joined dataset (right). The right join (3) does the same thing, but for the right dataset. The full outer join (4) keeps all records from both datasets, even if there is no match.

Figure 34: Join-Types



Source: Arbeck (2013). CC BY 3.0.

The following DataFrames show the data as explained in the example above. The first DataFrame contains the list of employee IDs for each employer while the second DataFrame stores the employee name for each ID.

Code

```
df_company = pd.DataFrame([ \
    ["MI6", "007"], \
    ["MI6", "001"], \
    ["MI6", "006"], \
    ["MI6", "005"], \
    ["MI5", "105"]], \
    columns = ["Employer", "Employee-ID"])

df_employee = pd.DataFrame([ \
    ["007", "James Bond"], \
    ["001", "Edward Donne"], \
    ["005", "Stuart Thomas"], \
    ["0012", "Sam Jonston"]], \
    columns = ["Employee-ID", "Employee-Name"])
```

The `merge()` method of pandas is used to execute the different join operations. The first example displays the inner join, the second the left, the third the right, and the fourth the outer join. Missing values are introduced for samples that do not have a match in the joining table.

Inner join example

Code

```
df_company.merge(df_employee, on='Employee-ID', \
                 how = "inner")
```

```
# console output:
```

```
# Employer Employee-ID Employee-Name
```



```
# 1   MI6   001 Edward Donne
# 2   MI6   005 Stuart Thomas
```

Left join example

Code

```
df_company.merge(df_employee, on='Employee-ID', \
                 how = "left")
```

```
# console output:
```

```
# Employer Employee-ID Employee-Name
```

```
# 1   MI6   001 Edward Donne
# 2   MI6   006          NaN
# 3   MI6   005 Stuart Thomas
# 4   MI5   105          NaN
```

Right join example

Code

```
df_company.merge(df_employee, on='Employee-ID', \
                 how = "right")
```

```
# console output:
```

```
# Employer Employee-ID Employee-Name
```

```
# 1   MI6   001 Edward Donne
# 2   MI6   005 Stuart Thomas
# 3   NaN   0012 Sam Jonston
```

Full (outer) join example

Code

```
df_company.merge(df_employee, on='Employee-ID', \
                 how = "outer")
```

```
# console output:
```

```
# Employer Employee-ID Employee-Name
# 1 MI6 001 Edward Donne
# 2 MI6 006 NaN
# 3 MI6 005 Stuart Thomas
# 4 MI5 105 NaN
# 5 NaN 0012 Sam Jonston
```

 **SUMMARY**

There are a variety of methods used to structure data, which can be grouped into intrarecord- and interrecord-structuring. Intrarecord structuring focuses on reordering columns, creating new columns based on existing columns, and merging multiple columns. Interrecord structuring focuses on filtering the dataset and shifting granularity through aggregations and pivots. To execute both structuring methods, it is important to understand the data. Data profiling can help in this step by using semantic, syntactic, or set-based profiling methods.

While the structuring and the profiling of data are often just the beginning of an analysis, it is important to understand the complete data workflow. This includes several stages through which the data pass. The first stage focuses on the ingestion and understanding of the data, while the second stage refines and designs the data. This includes the creation of first initial data products or prototypes. In the last stage, the data are optimized and put into a production system.

Within these stages, a lot of effort will go into data cleansing. Since there are many possible problems, the unit focused on the main groups of methods. This includes the removal of duplicates, the detection of outliers, and bad data. Normalization and standardization can help define an outlier by transforming the data to an appropriate scale. One method is fuzzy matching, which is an advanced text comparison method. Data imputation can help to fill gaps of missing data based on rules or statistical measures.

Once the data are cleaned, different methods can be applied to enrich them. This unit introduced feature construction, feature transformation, feature learning, and adding data. For the latter, unions and joins can extend a dataset.