

Data science

Data science is an interdisciplinary science that incorporates scientific fields of data such as data engineering, information science, computer science, statistics, artificial intelligence, machine learning, data mining, and predictive analytics.

Data analytics is the science of fact-finding analysis of raw data, with the goal of drawing conclusions from the data lake.

Machine learning is the capability of systems to learn without explicit software development. It evolved from the study of pattern recognition and computational learning theory.

UNIT 1

Data Science Technology Stack

Describe data science technology stack

The Data Science Technology Stack covers the data processing requirements in the Rapid Information Factory ecosystem.

The Rapid Information Factory ecosystem is a convention of techniques used for processing developments.

Data Science Storage Tools

This data science ecosystem has a series of tools that you use to build your solutions

Schema-on-Write and Schema-on-Read

There are two basic methodologies that are supported by the data processing tools.

Schema-on-Write Ecosystems

A traditional relational database management system (RDBMS) requires a schema before you can load the data. To retrieve data from my structured data schemas, you may have been running standard SQL queries for a number of years. Benefits include the following:

- In traditional data ecosystems, tools assume schemas and can only work once the schema is described, so there is only one view on the data.
- The approach is extremely valuable in articulating relationships between data points, so there are already relationships configured.
- It is an efficient way to store “dense” data.
- All the data is in the same data store.

On the other hand, schema-on-write isn’t the answer to every data science problem. Among the downsides of this approach are that

- Its schemas are typically purpose-built, which makes them hard to change and maintain.
- It generally loses the raw/atomic data as a source for future analysis.
- It requires considerable modeling/implementation effort before being able to work with the data.
- If a specific type of data can’t be stored in the schema, you can’t effectively process it from the schema. At present, schema-on-write is a widely adopted methodology to store data.

Schema-on-Read Ecosystems

This alternative data storage methodology does not require a schema before you can load the data. Fundamentally, you store the data with minimum structure. The essential schema is applied during the query phase.

Benefits include the following:

- It provides flexibility to store unstructured, semi-structured, and disorganized data.
- It allows for unlimited flexibility when querying data from the structure.

- Leaf-level data is kept intact and untransformed for reference and use for the future.
- The methodology encourages experimentation and exploration.
- It increases the speed of generating fresh actionable knowledge.
- It reduces the cycle time between data generation to availability of actionable knowledge.

A hybrid between schema-on-read and schema-on-write ecosystems is recommended for effective data science and engineering

Define (Differentiate) data lake, data vault and data warehouse

Data Lake

A data lake is a storage repository for a massive amount of raw data. It stores data in native format, in anticipation of future requirements. While a schema-on-write data warehouse stores data in predefined databases, tables, and records structures, a data lake uses a less restricted schema-on-read-based architecture to store data. Each data element in the data lake is assigned a distinctive identifier and tagged with a set of comprehensive metadata tags. A data lake is typically deployed using distributed data object storage, to enable the schema-on-read structure. This means that business analytics and data mining tools access the data without a complex schema. Using a schema-on-read methodology enables you to load your data as is and start to get value from it instantaneously. For deployment onto the cloud, it is a cost-effective solution to use Amazon's Simple Storage Service (Amazon S3) to store the base data for the data lake.

Data Vault

Data vault modeling, designed by Dan Linstedt, is a database modeling method that is intentionally structured to be in control of long-term historical storage of data from multiple operational systems. The data vaulting processes transform the schema-on-read data lake into a schema-on-write data vault. The data vault is designed into the schema-on-read query request and then executed against the data lake. The structure is built from three basic data structures: hubs, links, and satellites

Hubs Hubs contain a list of unique business keys with low propensity to change. They contain a surrogate key for each hub item and metadata classification of the origin of the business key. The hub is the core backbone of your data vault.

Links Associations or transactions between business keys are modeled using link tables. These tables are essentially many-to-many join tables, with specific additional metadata. The link is a singular relationship between hubs to ensure the business relationships are accurately recorded to complete the data model for the real-life business

Satellites Hubs and links form the structure of the model but store no chronological characteristics or descriptive characteristics of the data. These characteristics are stored in appropriated tables identified as satellites. Satellites are the structures that store comprehensive levels of the information on business characteristics and are normally the largest volume of the complete data vault data structure. The appropriate combination of hubs, links, and satellites helps the data scientist to construct and store prerequisite business relationships. This is a highly in-demand skill for a data modeler.

Data Warehouse Bus Matrix

The Enterprise Bus Matrix is a data warehouse planning tool and model created by Ralph Kimball and used by numerous people worldwide over the last 40+ years. The bus matrix and architecture builds upon the concept of conformed dimensions that are interlinked by facts. The data warehouse is a major component of the solution required to transform data into actionable knowledge. This schema-on-write methodology supports business intelligence against the actionable knowledge. A data warehouse is a consolidated, organized and structured repository for storing data.

Data Science Processing Tools

Explain any five data processing tools in data science technology stack

The next step involves processing tools to transform your data lakes into data vaults and then into data warehouses. These tools are the workhorses of the data science and engineering ecosystem. Following are the recommended foundations for the data tools

Spark

Apache Spark is an open source cluster computing framework. Originally developed at the AMP Lab of the University of California, Berkeley, the Spark code base was donated to the Apache Software Foundation, which now maintains it as an open source project. Spark offers an interface for programming distributed clusters with implicit data parallelism and fault-tolerance. Spark is a technology that is becoming a de-facto standard for numerous enterprise-scale processing applications

Spark Core

Spark Core is the foundation of the overall development. It provides distributed task dispatching, scheduling, and basic I/O functionalities. This enables you to offload the comprehensive and complex running environment to the Spark Core. This safeguards that the tasks you submit are accomplished as anticipated.

Spark SQL

Spark SQL is a component on top of the Spark Core that presents a data abstraction called Data Frames. Spark SQL makes accessible a domain-specific language (DSL) to manipulate data frames.

Spark Streaming

Spark Streaming leverages Spark Core's fast scheduling capability to perform streaming analytics. Spark Streaming has built-in support to consume from Kafka, Flume, Twitter, ZeroMQ, Kinesis, and TCP/IP sockets. The process of streaming is the primary technique for importing data from the data source to the data lake. Streaming is becoming the leading technique to load from multiple data sources. There are connectors available for many data sources.

MLlib

Machine Learning Library Spark MLlib is a distributed machine learning framework used on top of the Spark Core by means of the distributed memory-based Spark architecture. Common machine learning and statistical algorithms have been implemented and are shipped with MLlib, which simplifies large-scale machine learning pipelines, including

- Dimensionality reduction techniques, such as singular value decomposition (SVD) and principal component analysis (PCA)
- Summary statistics, correlations, stratified sampling, hypothesis testing, and random data generation
- Collaborative filtering techniques, including alternating least squares (ALS)
- Classification and regression: support vector machines, logistic regression, linear regression, decision trees, and naive Bayes classification
- Cluster analysis methods, including k-means and latent Dirichlet allocation (LDA)
- Optimization algorithms, such as stochastic gradient descent and limited-memory BFGS (L-BFGS)
- Feature extraction and transformation functions

GraphX

GraphX is a powerful graph-processing application programming interface (API) for the Apache Spark analytics engine that can draw insights from large data sets. GraphX provides outstanding speed and capacity for running massively parallel and machine-learning algorithms. The introduction of the graph-processing capability enables the processing of relationships between data entries with ease.

Mesos

Apache Mesos is an open source cluster manager that was developed at the University of California, Berkeley. It delivers efficient resource isolation and sharing across distributed applications. The software enables resource sharing in a fine-grained manner, improving cluster utilization.

Akka

The toolkit and runtime methods shorten development of large-scale data-centric applications for processing. Akka is an actor-based message-driven runtime for running concurrency, elasticity, and resilience processes. The use of high-level abstractions such as actors, streams, and futures facilitates the data science and engineering granularity processing units. The use of actors enables the data scientist to spawn a series of concurrent processes by using a simple processing model that employs a messaging technique and specific predefined actions/behaviors for each actor. This way, the actor can be controlled and limited to perform the intended tasks only.

Cassandra

Apache Cassandra is a large-scale distributed database supporting multi-data center replication for availability, durability, and performance.

Kafka

This is a high-scale messaging backbone that enables communication between data processing entities. The Apache Kafka streaming platform, consisting of Kafka Core, Kafka Streams, and Kafka Connect, is the foundation of the Confluent Platform. Kafka components empower the capture, transfer, processing, and storage of data streams in a distributed, fault-tolerant manner throughout an organization in real time.

Kafka Core At the core of the Confluent Platform is Apache Kafka. Confluent extends that core to make configuring, deploying, and managing Kafka less complex.

Kafka Streams Kafka Streams is an open source solution that you can integrate into your application to build and execute powerful stream-processing functions.

Kafka Connect This ensures Confluent-tested and secure connectors for numerous standard data systems. Connectors make it quick and stress-free to start setting up consistent data pipelines. These connectors are completely integrated with the platform, via the schema registry. Kafka Connect enables the data processing capabilities that accomplish the movement of data into the core of the data solution from the edge of the business ecosystem.

Elastic Search

Elastic search is a distributed, open source search and analytics engine designed for horizontal scalability, reliability, and stress-free management. It combines the speed of search with the power of analytics, via a sophisticated, developer-friendly query language covering structured, unstructured, and time-series data.

R

R is a programming language and software environment for statistical computing and graphics. The R language is widely used by data scientists, statisticians, data miners, and data engineers for developing statistical software and performing data analysis. The capabilities of R are extended through user-created packages using specialized statistical techniques and graphical procedures. A core set of packages is contained within the core installation of R, with additional packages accessible from the Comprehensive R Archive Network (CRAN). Knowledge of the following packages is a must:

- **sqldf** (data frames using SQL): This function reads a file into R while filtering data with an sql statement. Only the filtered part is processed by R, so files larger than those R can natively import can be used as data sources.
- **forecast** (forecasting of time series): This package provides forecasting functions for time series and linear models.
- **dplyr** (data aggregation): Tools for splitting, applying, and combining data within R

- stringr (string manipulation): Simple, consistent wrappers for common string operations
- RODBC, RSQLite, and RCassandra database connection packages: These are used to connect to databases, manipulate data outside R, and enable interaction with the source system.
- lubridate (time and date manipulation): Makes dealing with dates easier within R
- ggplot2 (data visualization): Creates elegant data visualizations, using the grammar of graphics. This is a super-visualization capability.
- reshape2 (data restructuring): Flexibly restructures and aggregates data, using just two functions: melt and dcast (or acast).
- randomForest (random forest predictive models): random forests for classification and regression
- gbm (generalized boosted regression models)

Scala

Scala is a general-purpose programming language. Scala supports functional programming and a strong static type system. Many high-performance data science frameworks are constructed using Scala, because of its amazing concurrency capabilities. Parallelizing masses of processing is a key requirement for large data sets from a data lake. Scala is emerging as the de-facto programming language used by data-processing tools. Scala is also the native language for Spark, and it is useful to master this language.

Python

Python is a high-level, general-purpose programming language created by Guido van Rossum and released in 1991. It is important to note that it is an interpreted language: Python has a design philosophy that emphasizes code readability. Python uses a dynamic type system and automatic memory management and supports multiple programming paradigms (object-oriented, imperative, functional programming, and procedural). Thanks to its worldwide success, it has a large and comprehensive standard library.

MQTT (MQ Telemetry Transport)

MQTT stands for *MQ Telemetry Transport*. The protocol uses publish and subscribe, extremely simple and lightweight messaging protocols. It was intended for constrained devices and low-bandwidth, high-latency, or unreliable networks. This protocol is perfect for machine-to-machine- (M2M) or Internet-of-things-connected devices. MQTT-enabled devices include handheld scanners, advertising boards, footfall counters, and other machines. The apt use of this protocol is critical in the present and future data science.

Layered Framework

Definition of Data Science Framework

Data science is a series of discoveries of converting raw unstructured data from your data lake into actionable business data. This process is a cycle of discovering and evolving your understanding of the data you are working with to supply you with metadata that you need. We need to build a basic framework that is used for data processing. This will enable you to construct a data science solution and then easily transfer it to your data engineering environments. The following is a framework for projects. It works for projects from small departmental to at-scale internationally distributed deployments, as the framework has a series of layers that enables you to follow a logical building process and then use your data processing and discoveries across many projects

Cross-Industry Standard Process for Data Mining (CRISP-DM)

Write short note on CRISP-DM

CRISP-DM was generated in 1996, and by 1997, it was extended via a European Union project, under the ESPRIT funding initiative. It was the majority support base for data scientists until mid-2015. The web site that was driving the Special Interest Group disappeared on June 30, 2015, and has since reopened. Since then, however, it started losing ground against other custom modeling methodologies. The basic concept behind the process is still valid, but you will find that most companies do not use it as is, in any projects, and have some form of modification that they employ as an internal standard.

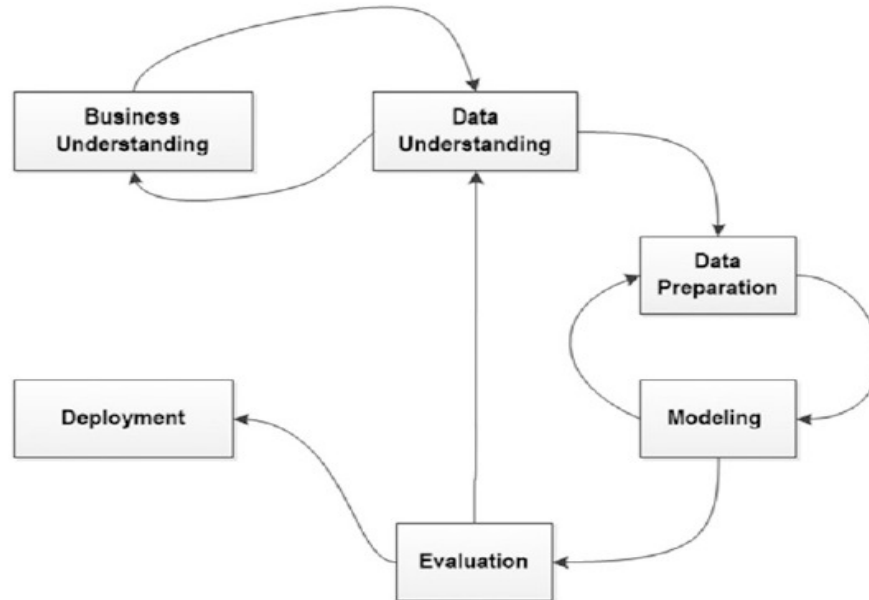


Figure 3-1. CRISP-DM flowchart

Business Understanding

This initial phase indicated in Figure 3-1 concentrates on discovery of the data science goals and requests from a business perspective.

Data Understanding

The data understanding phase noted in Figure 3-1 starts with an initial data collection and continues with actions to discover the characteristics of the data. This phase identifies data quality complications and insights into the data.

Data Preparation

The data preparation phase indicated in Figure 3-1 covers all activities to construct the final data set for modeling tools. This phase is used in a cyclical order with the modeling phase, to achieve a complete model. This ensures you have all the data required for your data science.

Modeling

In this phase noted in Figure 3-1, different data science modeling techniques are nominated and evaluated for accomplishing the prerequisite outcomes, as per the business requirements. It returns to the data preparation phase in a cyclical order until the processes achieve success.

Evaluation

At this stage shown in Figure 3-1, the process should deliver high-quality data science. Before proceeding to final deployment of the data science, validation that the proposed data science solution achieves the business objectives is required. If this fails, the process returns to the data understanding phase, to improve the delivery.

Deployment

Creation of the data science is generally not the end of the project. Once the data science is past the development and pilot phases, it has to go to production.

Homogeneous Ontology for Recursive Uniform Schema

The Homogeneous Ontology for Recursive Uniform Schema (HORUS) is used as an internal data format structure that enables the framework to reduce the permutations of transformations required by the framework. The use of HORUS methodology results in a hub-and-spoke data transformation approach. External data formats are converted to HORUS format, and then a HORUS format is transformed into any other external format. The basic concept is to take native raw data and then transform it first to a single format. That means that there is only one format for text files, one format for JSON or XML, one format for images and video. Therefore, to achieve any-to-any transformation coverage, the framework's only requirements are a data-format-to-HORUS and HORUS-to- data-format converter.

The Top Layers of a Layered Framework

Explain in brief the top layers of data science framework

The top layers are to support a long-term strategy of creating a Center of Excellence for your data science work. The layers shown in Figure 3-1 enable you to keep track of all the processing and findings you achieve. The framework will enable you to turn your small project into a big success, without having to do major restructuring along the route to production.

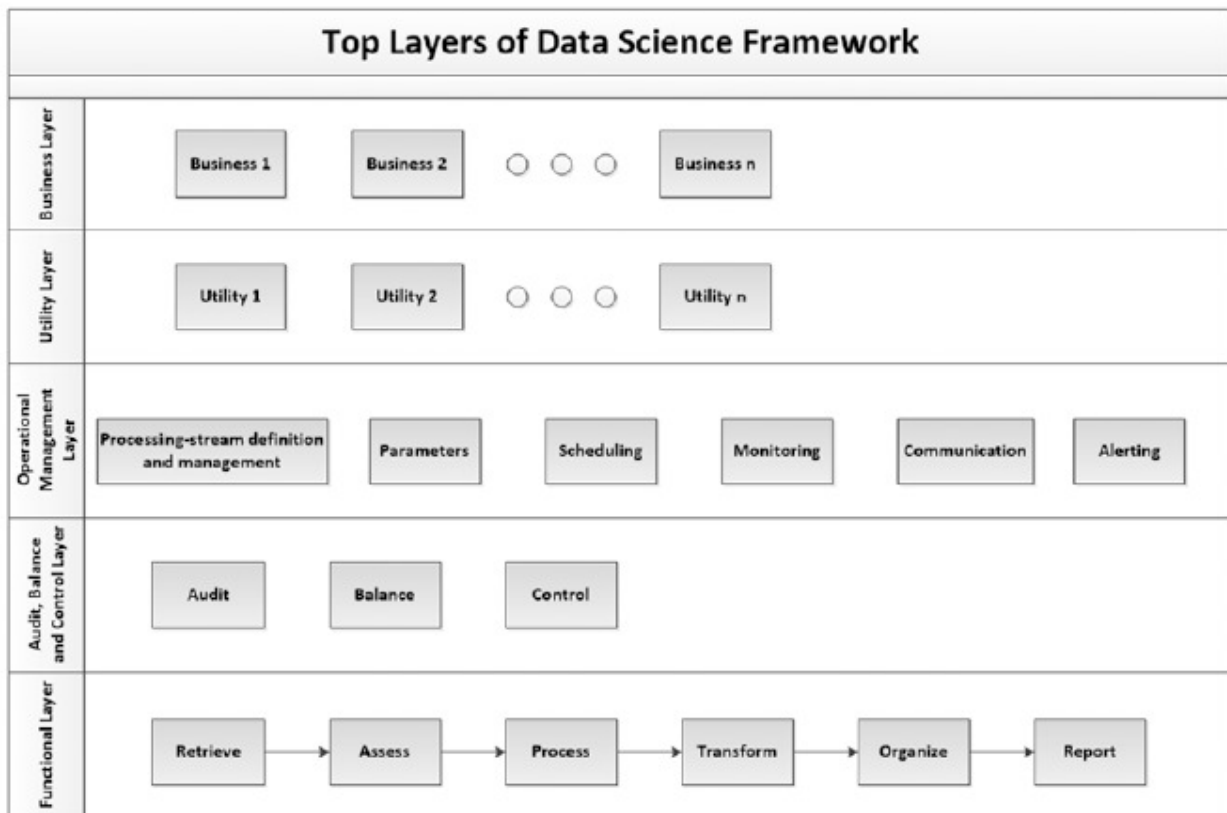


Figure 3-2. Top layers of the data science framework

The Basics for Business Layer

The business layer indicated in Figure 3-2 is the principal source of the requirements and business information needed by data scientists and engineers for processing. Material you would expect in the business layer includes the following:

- Up-to-date organizational structure chart

- Business description of the business processes you are investigating
- List of your subject matter experts
- Project plans
- Budgets
- Functional requirements
- Nonfunctional requirements
- Standards for data items

The Basics for Utility Layer

I define a utility as a process that is used by many data science projects to achieve common processing methodologies. The utility layer, as shown in Figure 3-2, is a common area in which you store all your utilities. You will be faced regularly with immediate requirements demanding that you fix something in the data or run your “magical” algorithm.

The Basics for Operational Management Layer

As referred to in Figure 3-2, operations management is an area of the ecosystem concerned with designing and controlling the process chains of a production environment and redesigning business procedures. This layer stores what you intend to process. It is where you plan your data science processing pipelines. The operations management layer is where you record

- Processing-stream definition and management
- Parameters
- Scheduling
- Monitoring
- Communication
- Alerting

The operations management layer is the common location where you store any of the processing chains you have created for your data science.

The Basics for Audit, Balance, and Control Layer

The audit, balance, and control layer, represented in Figure 3-2, is the area from which you can observe what is currently running within your data science environment. It records

- Process-execution statistics
- Balancing and controls
- Rejects and error-handling
- Codes management

The three subareas are utilized in following manner.

Audit

The audit sublayer, indicated in Figure 3-2, records any process that runs within the environment. This information is used by data scientists and engineers to understand and plan improvements to the processing. The use of the native audit capability of the Data Science Technology Stack will provide you with a quick and effective base for your auditing. The audit information I find essential and useful is discussed in detail in Chapter 6. Following is a quick introduction to what you could expect in this topic.

Balance

The balance sublayer in Figure 3-2 ensures that the ecosystem is balanced across the available processing capability or has the capability to top-up capability during periods of extreme processing. The processing on demand capability of a cloud ecosystem is highly desirable for this purpose. Using the audit trail, it is possible to adapt to changing requirements and forecast what you will require to complete the schedule of work you submitted to the ecosystem. In the always-on and top-up ecosystem you can build, you can balance your processing requirements by removing or adding resources dynamically as you move through the processing pipe. An example would be during end-of-

month processing, you increase your processing capacity to sixty nodes, to handle the extra demand of the end-of-month run. The rest of the month, you run at twenty nodes during business hours. During weekends and other slow times, you only run with five nodes. Massive savings can be generated in this manner.

Control

The control sublayer, indicated in Figure 3-2, controls the execution of the current active data science processes in a production ecosystem. The control elements are a combination of the control element within the Data Science Technology Stack's individual tools plus a custom interface to control the primary workflow. The control also ensures that when processing experiences an error, it can attempt a recovery, as per your requirements, or schedule a clean-up utility to undo the error. This enables data scientists to concentrate on the models and processing and not on complying with the more controlled production requirements.

The Basics for Functional Layer

The functional layer of the data science ecosystem shown in Figure 3-2 is the main layer of programming required. The functional layer is the part of the ecosystem that executes the comprehensive data science. It consists of several structures.

- Data models
- Processing algorithms
- Provisioning of infrastructure

Processing algorithm is spread across six supersteps of processing, as follows:

1. *Retrieve*: This super step contains all the processing chains for retrieving data from the raw data lake via a more structured format.
2. *Assess*: This superstep contains all the processing chains for quality assurance and additional data enhancements.
3. *Process*: This superstep contains all the processing chains for building the data vault.
4. *Transform*: This superstep contains all the processing chains for building the data warehouse.
5. *Organize*: This superstep contains all the processing chains for building the data marts.
6. *Report*: This superstep contains all the processing chains for building virtualization and reporting the actionable knowledge.

Business Layer

The business layer is the transition point between the nontechnical business requirements and desires and the practical data science, where, I suspect, most readers of this book will have a tendency to want to spend their careers, doing the perceived more interesting data science. The business layer does not belong to the data scientist 100%, and normally, its success represents a joint effort among such professionals as business subject matter experts, business analysts, hardware architects, and data scientists. The business layer is where we record the interactions with the business. This is where we convert business requirements into data science requirements

The Functional Requirements

Describe the functional requirements in the business layer of the data science framework

Functional requirements record the detailed criteria that must be followed to realize the business's aspirations from its real-world environment when interacting with the data science ecosystem. These requirements are the business's view of the system, which can also be described as the "Will of the Business." The MoSCoW method (Table 4-1) is a prioritization technique, to indicate how important

each requirement is to the business.

Table 4-1. MoSCoW Options

Must have	Requirements with the priority “must have” are critical to the current delivery cycle.
Should have	Requirements with the priority “should have” are important but not necessary to the current delivery cycle.
Could have	Requirements prioritized as “could have” are those that are desirable but not necessary, that is, nice to have to improve user experience for the current delivery cycle.
Won't have	Requirements with a “won't have” priority are those identified by stakeholders as the least critical, lowest payback requests, or just not appropriate at that time in the delivery cycle.

Specific Functional Requirements

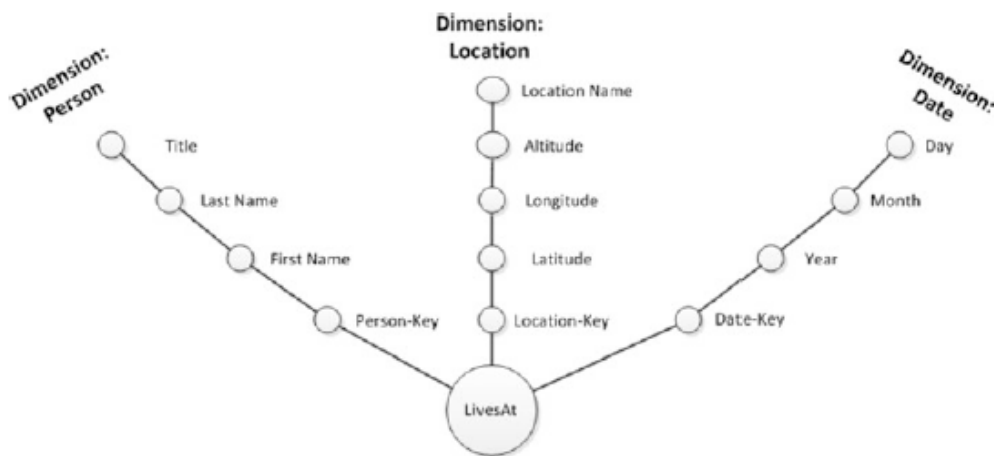
The following requirements specific to data science environments will assist you in creating requirements that enable you to transform a business’s aspirations into technical descriptive requirements.

Data Mapping Matrix

The data mapping matrix is one of the core functional requirement recording techniques used in data science. It tracks every data item that is available in the data sources.

Sun Models

The sun models is a requirement mapping technique that assists you in recording requirements at a level that allows your nontechnical users to understand the intent of your analysis, while providing you with an easy transition to the detailed technical modeling of your data scientist and data engineer. This sun model is for fact LivesAt and supports three dimensions: person, location, and date. Figure 4-1 shows a typical sun model.



Dimensions

A dimension is a structure that categorizes facts and measures, to enable you to respond to business questions. A slowly changing dimension is a data structure that stores the complete history of the data loads in the dimension structure over the life cycle of the data lake. There are several types of Slowly Changing Dimensions (SCDs) in the data warehousing design toolkit that enable different recording rules of the history of the dimension.

Facts

A fact is a measurement that symbolizes a fact about the managed entity in the real world.

Intra-Sun Model Consolidation Matrix

The intra-sun model consolidation matrix is a tool that helps you to identify common dimensions between sun models

The Nonfunctional Requirements

List out the non-functional requirements in the business layer of the data science framework

Nonfunctional requirements record the precise criteria that must be used to appraise the operation of a data science ecosystem.

Accessibility Requirements

Accessibility can be viewed as the “ability to access” and benefit from some system or entity. The concept focuses on enabling access for people with disabilities, or special needs, or enabling access through assistive technology.

Assistive technology covers the following:

- *Levels of blindness support*: Must be able to increase font sizes or types to assist with reading for affected people
- *Levels of color-blindness support*: Must be able to change a color palette to match individual requirements
- *Use of voice-activated commands to assist disabled people*: Must be able to use voice commands for individuals that cannot type commands or use a mouse in normal manner

Audit and Control Requirements

Audit is the ability to investigate the use of the system and report any violations of the system’s data and processing rules. Control is making sure the system is used in the manner and by whom it is pre-approved to be used. An approach called role-based access control (RBAC) is the most commonly used approach to restricting system access to authorized users of your system. RBAC is an access-control mechanism formulated around roles and privileges. The components of RBAC are role-permissions—user-role and role-role relationships that together describe the system’s access policy. These audit and control requirements are also compulsory, by regulations on privacy and processing.

Availability Requirements

Availability is as a ratio of the expected uptime of a system to the aggregate of the downtime of the system. For example, if your business hours are between 9h00 and 17h00, and you cannot have more than 1 hour of downtime during your business hours, you require 87.5% availability. Take note that you specify precisely at what point you expect the availability.

Record your requirements in the following format: *Component C will be entirely operational for P% of the time over an uninterrupted measured period of D days*. Your customers will understand this better than the general “24/7” or “business hours”. No system can achieve these general requirement statements. The business will also have periods of high availability at specific periods during the day, week, month, or year. An example would be every Monday morning the data science results for the weekly meeting has to be available. This could be recorded as the following: *Weekly reports must be entirely operational for 100% of the time between 06h00 and 10h00 every Monday for each office*

Identify single points of failure (SPOFs) in the data science solution. Ensure that you record this clearly, as SPOFs can impact many of your availability requirements indirectly. Highlight that those dependencies between components that may not be available at the same time must be recorded and requirements specified, to reflect this availability requirement fully.

Backup Requirements

A backup, or the process of backing up, refers to the archiving of the data lake and all the data science programming code, programming libraries, algorithms, and data models, with the sole purpose of restoring these to a known good state of the system, after a data loss or corruption event. Remember:

Even with the best distribution and self-healing capability of the data lake, you have to ensure that you have a regular and appropriate backup to restore. Remember a backup is only valid if you can restore it. The merit of any system is its ability to return to a good state. This is a critical requirement. For example, suppose that your data scientist modifies the system with a new algorithm that erroneously updates an unknown amount of the data in the data lake.

Capacity, Current, and Forecast

Capacity is the ability to load, process, and store a specific quantity of data by the data science processing solution. You must track the current and forecast the future requirements, because as a data scientist, you will design and deploy many complex models that will require additional capacity to complete the processing pipelines you create during your processing cycles.

Capacity

Capacity is measured per the component's ability to consistently maintain specific levels of performance as data load demands vary in the solution. The correct way to record the requirement is *Component C will provide P% capacity for U users, each with M MB of data during a time frame of T seconds.*

Example:

The data hard drive will provide 95% capacity for 1000 users, each with 10MB of data during a time frame of 10 minutes.

Concurrency

Concurrency is the measure of a component to maintain a specific level of performance when under multiple simultaneous loads conditions.

The correct way to record the requirement is *Component C will support a concurrent group of U users running predefined acceptance script S simultaneously*

Throughput Capacity

This is how many transactions at peak time the system requires to handle specific conditions.

Storage (Memory)

This is the volume of data the system will persist in memory at runtime to sustain an effective processing solution.

Storage (Disk)

This is the volume of data the system stores on disk to sustain an effective processing solution.

Storage (GPU)

This is the volume of data the system will persist in GPU memory at runtime to sustain an effective parallel processing solution, using the graphical processing capacity of the solution. A CPU consists of a limited amount of cores that are optimized for sequential serial processing, while a GPU has a massively parallel architecture consisting of thousands of smaller, more efficient cores intended for handling massive amounts of multiple tasks simultaneously.

Configuration Management

Configuration management (CM) is a systems engineering process for establishing and maintaining consistency of a product's performance, functional, and physical attributes against requirements, design, and operational information throughout its life.

Deployment

A methodical procedure of introducing data science to all areas of an organization is required. Investigate how to achieve a practical continuous deployment of the data science models. These skills are much in demand, as the processes model changes more frequently as the business adopts new processing techniques.

Documentation

Data science requires a set of documentation to support the story behind the algorithms.

Disaster Recovery

Disaster recovery (DR) involves a set of policies and procedures to enable the recovery or continuation of vital technology infrastructure and systems following a natural or human-induced disaster.

Efficiency (Resource Consumption for Given Load)

Efficiency is the ability to accomplish a job with a minimum expenditure of time and effort. As a data scientist, you are required to understand the efficiency curve of each of your modeling techniques and algorithms.

Effectiveness (Resulting Performance in Relation to Effort)

Effectiveness is the ability to accomplish a purpose; producing the precise intended or expected result from the ecosystem. As a data scientist, you are required to understand the effectiveness curve of each of your modeling techniques and algorithms. You must ensure that the process is performing only the desired processing and has no negative side effects.

Extensibility

The ability to add extra features and carry forward customizations at next-version upgrades within the data science ecosystem. The data science must always be capable of being extended to support new requirements.

Failure Management

Failure management is the ability to identify the root cause of a failure and then successfully record all the relevant details for future analysis and reporting.

Fault Tolerance

Fault tolerance is the ability of the data science ecosystem to handle faults in the system's processing. In simple terms, no single event must be able to stop the ecosystem from continuing the data science processing.

Latency

Latency is the time it takes to get the data from one part of the system to another. This is highly relevant in the distributed environment of the data science ecosystems. Acceptance script S completes within T seconds on an unloaded system and within T2 seconds on a system running at maximum capacity, as defined in the concurrency requirement.

Interoperability

Insist on a precise ability to share data between different computer systems under this section. Explain in detail what system must interact with what other systems

Maintainability

Insist on a precise period during which a specific component is kept in a specified state. Describe precisely how changes to functionalities, repairs, and enhancements are applied while keeping the ecosystem in a known good state.

Modifiability

Stipulate the exact amount of change the ecosystem must support for each layer of the solution.

Network Topology

Stipulate and describe the detailed network communication requirements within the ecosystem for processing. Also, state the expected communication to the outside world, to drive successful data science

Privacy

Listing the exact privacy laws and regulations that apply to this ecosystem. Make sure you record the specific laws and regulations that apply. Seek legal advice if you are unsure. This is a hot topic worldwide, as you will process and store other people's data and execute algorithms against this data. As a data scientist, you are responsible for your actions.

Quality

Specify the rigorous faults discovered, faults delivered, and fault removal efficiency at all levels of the ecosystem. Remember: Data quality is a functional requirement. This is a nonfunctional requirement that states the quality of the ecosystem, not the data flowing through it.

Recovery/Recoverability

The ecosystem must have a clear-cut mean time to recovery (MTTR) specified. The MTTR for specific layers and components in the ecosystem must be separately specified. I typically measure in hours, but for other extra-complex systems, I measure in minutes or even seconds.

Reliability

The ecosystem must have a precise mean time between failures (MTBF). This measurement of availability is specified in a pre-agreed unit of time. I normally measure in hours, but there are extra sensitive systems that are best measured in years.

Resilience

Resilience is the capability to deliver and preserve a tolerable level of service when faults and issues to normal operations generate complications for the processing. The ecosystem must have a defined ability to return to the original form and position in time, regardless of the issues it has to deal with during processing.

Resource Constraints

Resource constraints are the physical requirements of all the components of the ecosystem. The areas of interest are processor speed, memory, disk space, and network bandwidth, plus, normally, several other factors specified by the tools that you deploy into the ecosystem

Reusability

Reusability is the use of pre-built processing solutions in the data science ecosystem development process. The reuse of preapproved processing modules and algorithms is highly advised in the general processing of data for the data scientists.

Scalability

Scalability is how you get the data science ecosystem to adapt to your requirements. I use three scalability models in my ecosystem: horizontal, vertical, and dynamic (on-demand). Horizontal scalability increases capacity in the data science ecosystem through more separate resources, to improve performance and provide high availability (HA). Vertical scalability increases capacity by adding more resources (more memory or an additional CPU) to an individual machine.

Security

One of the most important nonfunctional requirements is security. I specify security requirements at three levels.- Privacy, Physical(Include physical requirements such as power, elevated floors, extra server cooling, fire prevention systems, and cabinet locks.), Access(specify detailed access requirements with defined account types/groups and their precise access rights)

Testability

International standard IEEE 1233-1998 states that *testability* is the “degree to which a requirement is stated in terms that permit establishment of test criteria and performance of tests to determine whether those criteria have been met.” In simple terms, if your requirements are not testable, do not accept them.

Controllability

Knowing the precise degree to which I can control the state of the code under test, as required for testing, is essential. The algorithms used by data science are not always controllable, as they include random start points to speed the process.

Isolate Ability

The specific degree to which I can isolate the code under test will drive most of the possible testing.

Understandability

The degree to which the algorithms under test are documented directly impacts the testability of requirements.

Automatability

The degree to which I can automate testing of the code

What are the common pitfalls in requirements

Common Pitfalls with Requirements

1) Weak Words

Weak words are subjective or lack a common or precise definition. The following are examples in which weak words are included and identified:

- Users must *easily* access the system.

What is “easily”?

- Use *reliable* technology.

What is “reliable”?

- *State-of-the-art* equipment

What is “state-of-the-art”?

- Reports must run *frequently*.

What is “frequently”?

2) Unbounded Lists

An unbounded list is an incomplete list of items. Make sure your lists are complete and precise.

3) Implicit Collections

When collections of objects within requirements are not explicitly defined, you or your team will assume an incorrect meaning. See the following example:

The solution must support TCP/IP and other network protocols supported by existing users with Linux.

- What is meant by “existing user”?

4) Ambiguity

Ambiguity occurs when a word within the requirement has multiple meanings.

- 5) Vagueness (eg current standards)
- 6) Subjectivity (eg easily)
- 7) Optionality (eg as many as possible)
- 8) Under-specification (eg other database versions)
- 9) Under-reference (eg previous reports)

Engineering a Practical Business Layer

Requirements Registry

The register acts as a control for the data science environment’s requirements. An example of a template is shown in Figure

Practical Data Science						
Requirement Number	MoSCoW	Requirement Description	Date Created	Date Last Version	Status	Notes
R-000001-00						
R-000001-01						
R-000001-02						
R-000001-03						
R-000001-04						
R-000001-05						

Traceability matrix template

Trace Number	Input Source	Input Field	Process Rule	Output Source	Output Field	RAPTOR Step	Date Created	Date Last Version	Status	Notes
T-000001-00										
T-000001-01										
T-000001-02										
T-000001-03										
T-000001-04										
T-000001-05										

Create a traceability matrix against each requirement and the data science process you developed, to ensure that you know what data science process supports which requirement. This ensures that you have complete control of the environment. Changes are easy if you know how everything interconnects.

Utility Layer

Explain utility layer

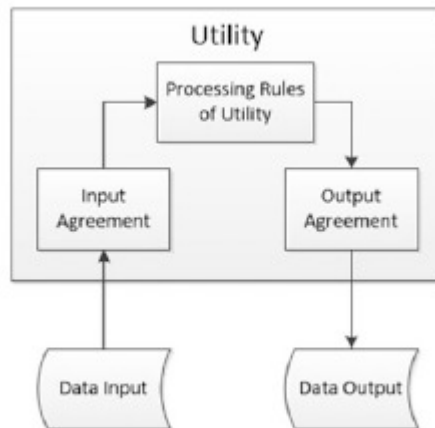
The utility layer is used to store repeatable practical methods of data science. Utilities are the common and verified workhorses of the data science ecosystem. The utility layer is a central storehouse for keeping all one's solutions utilities in one place. Having a central store for all utilities ensures that you do not use out-of-date or duplicate algorithms in your solutions. The most important benefit is that you can use stable algorithms across your solutions. If you use algorithms, keep any proof and credentials that show that the process is a high-quality, industry-accepted algorithm. The additional value is the capability of larger teams to work on a similar project and know that each data scientist or engineer is working to the identical standards. European Union General Data Protection Regulation (GDPR) is in effect. The GDPR has the following rules:

- You must have valid consent as a legal basis for processing
- You must assure transparency, with clear information about what data is collected and how it is processed. Utilities must generate complete audit trails of all their activities
- You must support the right to accurate personal data. Utilities must use only the latest accurate data
- You must support the right to have personal data erased. Utilities must support the removal of all information on a specific person.
- You must have approval to move data between service providers.
- You must support the right not to be subject to a decision based solely on automated processing.

Basic Utility Design

The basic utility must have a common layout to enable future reuse and enhancements. This standard makes the utilities more flexible and effective to deploy in a large-scale ecosystem. basic design for a processing utility, is a three-stage process.

1. Load data as per input agreement.
2. Apply processing rules of utility.
3. Save data as per output agreement.



-1. Basic utility design

The main advantage of this methodology in the data science ecosystem is that you can build a rich set of utilities that all your data science algorithms require. That way, you have a basic pre-validated set of tools to use to perform the common processing and then spend time only on the custom portions of the project.

There are three types of utilities

- Data processing utilities
- Maintenance utilities
- Processing utilities

Data processing utilities perform some form of data transformation within the solutions.

Describe various data processing utilities

Retrieve Utilities

Utilities for this superstep contain the processing chains for retrieving data out of the raw data lake into a new structured format. You can build all your retrieve utilities to transform the external raw data lake format into the Homogeneous Ontology for Recursive Uniform Schema (HORUS) data format. For example, the HORUS format can be selected to be CSV-based.

Assess Utilities

The assess utilities ensure that the data imported via the Retrieve superstep are of a good quality, to ensure it conforms to the prerequisite standards of your solution. There are two types of assess utilities:

Feature Engineering

Feature engineering is the process by which you enhance or extract data sources, to enable better extraction of characteristics you are investigating in the data sets.

Fixers Utilities

Fixers enable your solution to take your existing data and fix a specific quality issue.

Examples include

- Removing leading or lagging spaces from a data entry

Example in Python:

```

baddata = "      Data Science with too many spaces is bad!!!      "
print('>',baddata,<')
cleandata=baddata.strip()

```

- Removing nonprintable characters from a data entry

- Reformatting data entry to match specific formatting criteria. Convert 2017/01/31 to 31 January 2017

Adders Utilities

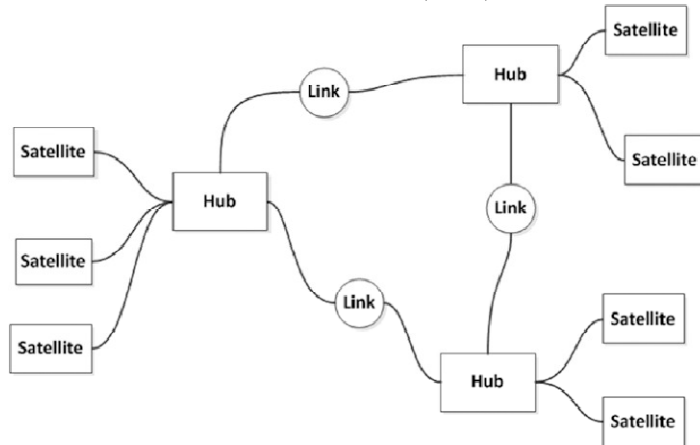
Adders use existing data entries and then add additional data entries to enhance your data

Process Utilities

Utilities for this superstep contain all the processing chains for building the data vault.

Data Vault Utilities

The data vault is a highly specialist data storage technique that was designed by Dan Linstedt. The data vault is a detail-oriented, historical-tracking, and uniquely linked set of normalized tables that support one or more functional areas of business. It is a hybrid approach encompassing the best of breed between 3rd normal form (3NF) and star schema.



Transform Utilities

Utilities for this superstep contain all the processing chains for building the data warehouse from the results of your practical data science. There are two basic transform utilities.

Dimensions Utilities

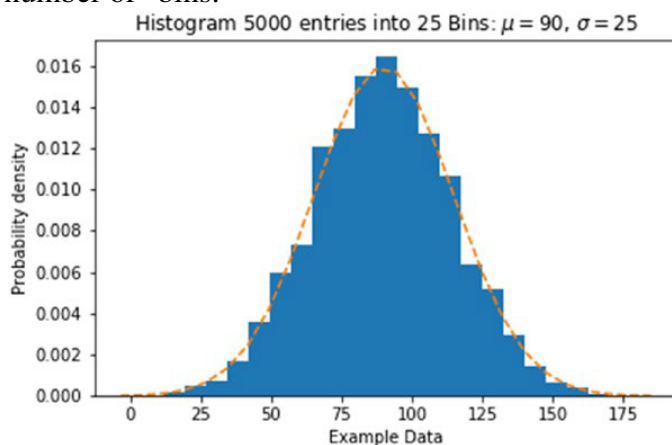
The dimensions use several utilities to ensure the integrity of the dimension structure.

Data Science Utilities

There are several data science-specific utilities that are required for you to achieve success in the data processing ecosystem.

Data Binning or Bucketing

Binning is a data preprocessing technique used to reduce the effects of minor observation errors. Statistical data binning is a way to group a number of more or less continuous values into a smaller number of “bins.”



Averaging of Data

The use of averaging of features value enables the reduction of data volumes in a control fashion to improve effective data processing. This technique also enables the data science to prevent a common issue called overfitting the model

Outlier Detection

Outliers are data that is so different from the rest of the data in the data set that it may be caused by an error in the data source. There is a technique called outlier detection that, with good data science, will identify these outliers.

Organize Utilities

Utilities for this superstep contain all the processing chains for building the data marts. The organize utilities are mostly used to create data marts against the data science results stored in the data warehouse dimensions and facts

Report Utilities

Utilities for this superstep contain all the processing chains for building virtualization and reporting of the actionable knowledge. The report utilities are mostly used to create data virtualization against the data science results stored in the data marts.

What are the maintenance and processing utilities in the Layered Framework utility layer?

Maintenance Utilities

The data science solutions you are building are a standard data system and, consequently, require maintenance utilities, as with any other system.

- Backup and Restore Utilities

These perform different types of database backups and restores for the solution.

- Checks Data Integrity Utilities

These utilities check the allocation and structural integrity of database objects and indexes across the ecosystem, to ensure the accurate processing of the data into knowledge.

- History Cleanup Utilities

These utilities archive and remove entries in the history tables in the databases.

- Maintenance Cleanup Utilities

These utilities remove artifacts related to maintenance plans and database backup files.

- Notify Operator Utilities

Utilities that send notification messages to the operations team about the status of the system are crucial to any data science factory

- Rebuild Data Structure Utilities

These utilities rebuild database tables and views to ensure that all the development is as designed

- Reorganize Indexing Utilities

These utilities reorganize indexes in database tables and views, which is a major operational process when your data lake grows at a massive volume and velocity. The variety of data types also complicates the application of indexes to complex data structures.

- Shrink/Move Data Structure Utilities

These reduce the footprint size of your database data and associated log artifacts, to ensure an optimum solution is executing

- Solution Statistics Utilities

These utilities update information about the data science artifacts, to ensure that your data science structures are recorded. Call it data science on your data science

Processing Utilities

The data science solutions you are building require processing utilities to perform standard system processing. The data science environment requires two basic processing utility types.

- Scheduling Utilities

The scheduling utilities are based on the basic agile scheduling principles.

- Backlog Utilities

Backlog utilities accept new processing requests into the system and are ready to be processed in future processing cycles.

- To-Do Utilities

The to-do utilities take a subset of backlog requests for processing during the next processing cycle. They use classification labels, such as priority and parent-child relationships, to decide what process runs during the next cycle.

- Doing Utilities

The doing utilities execute the current cycle's requests.

- Done Utilities

The done utilities confirm that the completed requests performed the expected processing.

- Monitoring Utilities

The monitoring utilities ensure that the complete system is working as expected.

Engineering a Practical Utility Layer

Create three sublayers to help the utility layer support better future use of the utilities.

Maintenance Utility

Collect all the maintenance utilities in this single directory, to enable the environment to handle the utilities as a collection. Keep a maintenance utilities registry, to enable your entire team to use the common utilities. Include enough documentation for each maintenance utility, to explain its complete workings and requirements

Data Utility

Collect all the data utilities in this single directory, to enable the environment to handle the utilities as a collection. Keep a data utilities registry to enable your entire team to use the common utilities. Include enough documentation for each data utility to explain its complete workings and requirements

Processing Utility

Collect all the processing utilities in this single directory to enable the environment to handle the utilities as a collection. Keep a processing utilities registry, to enable your entire team to use the common utilities. Include sufficient documentation for each processing utility to explain its complete workings and requirements