Notes of Data Science Unit I as per NEP

Master of information technology (University of Mumbai)

Data Science

MODULE-1: Data Science Storage Tools

Pushpa Mahapatro	**Do subscribe to my channel for useful educational contents: https://www.youtube.com/@mahapatrotutorials**

1. Rapid Information Factory Ecosystem or Data Science Storage Tools

This data science ecosystem has a series of tools that you use to build your solutions. This environment is undergoing a rapid advancement in capabilities, and new developments are occurring every day.

1.1 Schema-on-Write and Schema-on-Read

There are two basic methodologies that are supported by the data processing tools. Following is a brief outline of each methodology and its advantages and drawbacks.

- **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 oncethe schema is described, so there is only one view on the data.
- The approach is extremely valuable in articulating relationships between datapoints, 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 ableto 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.

- ***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.

1.2 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. You will acquire insights from this book on why this is extremely important for practical data science and engineering solutions. 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.

1.3Data 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.

Results stored in a schema-on-write format, to persist the results for future queries. The structure is built from three basic data structures: hubs, inks, 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.

- **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.

1.4 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.

2. Data Science Processing Tools

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.

2.1 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. This tool is evolving at an incredible rate.

IBM is committing more than 3,500 developers and researchers to work on Spark- related projects and formed a dedicated Spark technology center in San Francisco to pursue Spark-based innovations.

SAP, Tableau, and Talend now support Spark as part of their core software stack. Cloudera, Hortonworks, and MapR distributions support Spark as a native interface.

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.

The following modules are part of technology toolkit.

- **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. The distributed nature of the Spark ecosystem enables you to use the same processing request on a small Spark cluster, then on a cluster of thousands of nodes, without any code changes.

- **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. This feature of Spark enables ease of transition from your traditional SQL environments into the Spark environment. I have recognized its advantage when you want to enable legacy applications to offload the data from their traditional relational-only data storage to the data lake ecosystem.

- **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. I have found that there are connectors available for many data sources. There is a major drive to build even more improvements on connectors, and this will improve the ecosystem even further in the future.

- **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.
In Spark 2.0, a new library, **spark.mk**, was introduced to replace the RDD-based data processing with a DataFrame-based model. It is planned that by the introduction of Spark 3.0, only DataFrame-based models will exist.

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, andrandom 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

2.2GraphX

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 machinelearning algorithms.

2.3 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.

2.4 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.

2.5 Cassandra

Apache Cassandra is a large-scale distributed database supporting multi-data center replication for availability, durability, and performance. The standard Apache Cassandra open source version works just as well, minus some extra management it does not offer as standard.

2.6 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.
The Confluent Platform is the main commercial supporter for Kafka (see www. confluent.io/).
Most of the Kafka projects I am involved with now use this 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.

2.7 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.

2.8 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 thoseR 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 stringoperations
- *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 thegrammar 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):* Leo Breiman and Adele Cutler's random forests for classification and regression
- *gbm (generalized boosted regression models):* Yoav Freund and Robert Schapire's AdaBoost algorithm and Jerome Friedman's gradient boosting machine

2.9 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.

2.10 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).

2.11 MQTT (MQ Telemetry Transport)

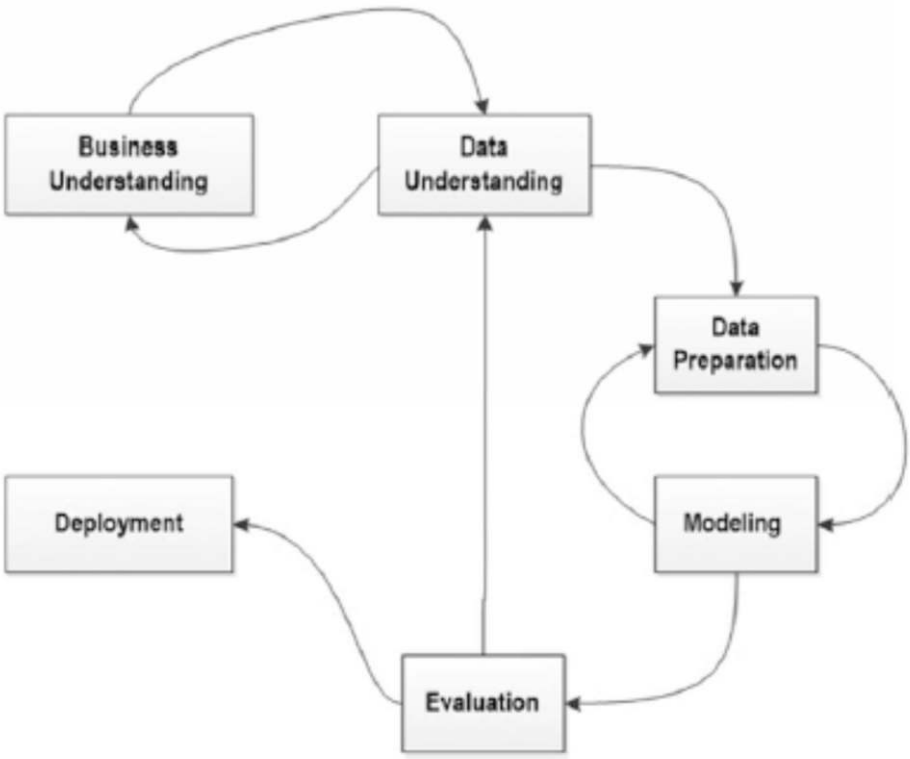
MQTT stands for *MQ Telemetry Transport.* The protocol uses publish and subscribe, extremely simple and lightweight messaging protocols. It was intended for constrained devices and low-bandwidth, high-latency, or unreliable networks. This protocol is perfect for machine-to-machine- (M2M) or Internet-of-things-connected devices.

3. Definition of Data Science Framework

Data science is a series of discoveries. You work toward an overall business strategy 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.

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

An overview of the basics is provided in Figure.



re 3-1. CRISP-DM flowchart

4.1 Business Understanding

This initial phase indicated in Figure 3-1 concentrates on discovery of the data science goals and requests from a business perspective. Many businesses use a decision model or process mapping tool that is based on the Decision Model and Notation (DMN) standard. DMN is a standard published by the Object Management Group . The DMN standard offers businesses a modeling notation for describing decision management flows and business rules.

4.2Data 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.

4.3 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.

4.4 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.

4.5 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.

4.6 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.

5. 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 HURUS- to-data-format converter.

Table 3-1 demonstrates how by using of the hub-and-spoke methodology on text files to reduce the amount of convertor scripts you must achieve an effective data science solution. You can see that at only 100 text data sets, you can save 98% on a non-hub-and- spoke framework. I advise that you invest time in getting the hub-and-spoke framework to work in your environment, as the savings will reward anytime you invest. Similar deductions are feasible by streamlining other data types.

Data Formats	Normal Convertors Required	HORUS Convertors Required (hub-and-spoke)	% Saved
2	4	4	0.00%
3	9	6	33.33%
4	16	8	50.00%
5	25	10	60.00%

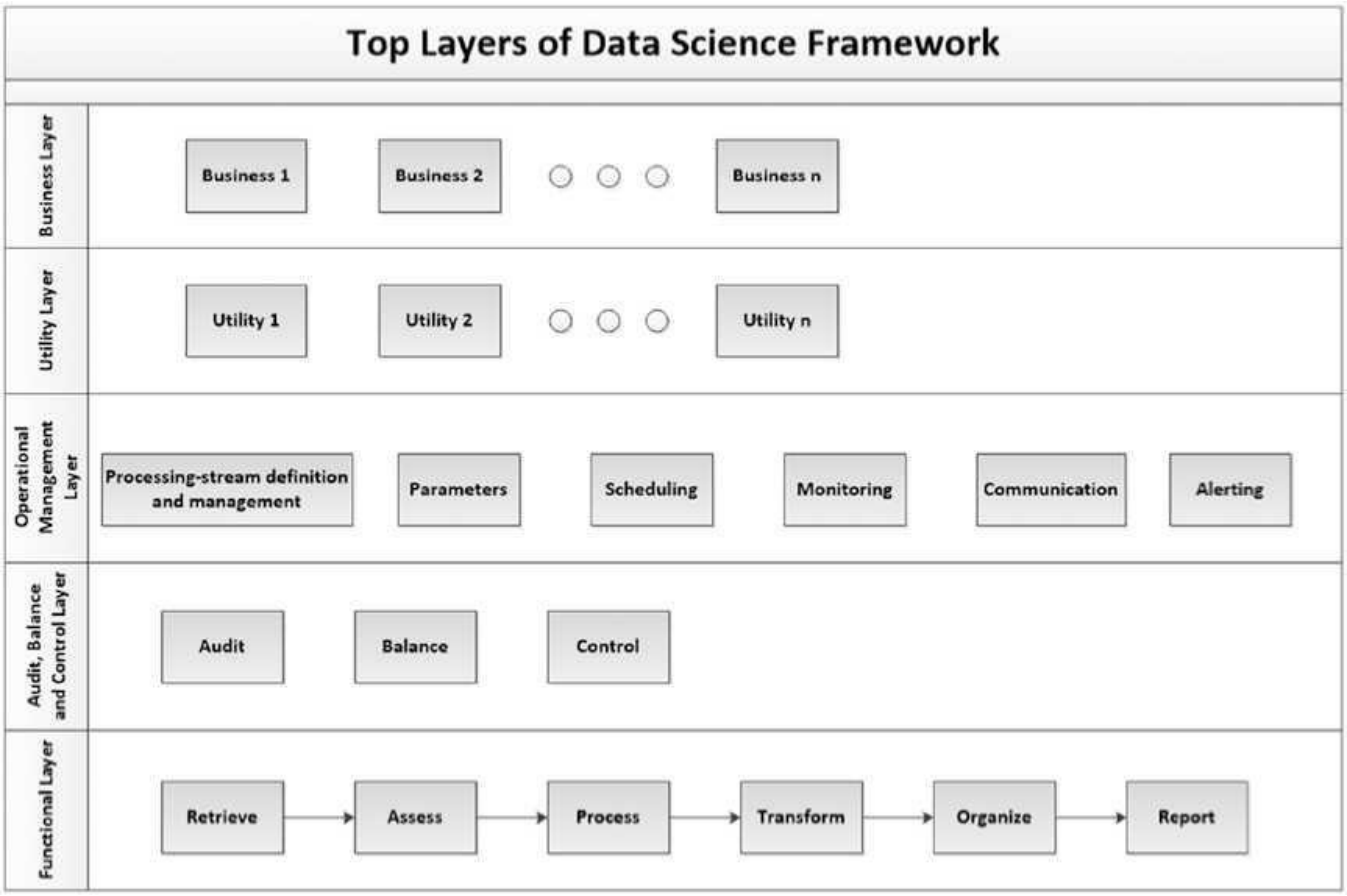
6	**36**	**12**	**66.67%**
7	**49**	**14**	**71.43%**
8	**64**	**16**	**75.00%**
9	**81**	**18**	**77.78%**
10	**100**	**20**	**80.00%**
25	**625**	**50**	**92.00%**
50	**2500**	**100**	**96.00%**
100	**10000**	**200**	**98.00%**

Table 3-1. Advantages of Using Hub-and-Spoke Data Conversions

6. The Top Layers of a Layered Framework

The top layers are to support a long-term strategy of creating a Center of Excellence for your data science work. The use of a Center of Excellence framework is required, if you are working in a team environment. Using the framework will prepare you to work in an environment, with multiple data scientists working on a common ecosystem.

If you are just data wrangling a few data files, and your data science skills must yield quick insights Remember: All big systems were a small pilot at some point. 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.



6.1 The Basics for Business Layer

The business layer indicated in Above figure is the principal source of the requirements and business information needed by data scientists and engineers for processing.

Data science has matured. The years of providing a quick half-baked analysis, then rushing off to the next analysis are, to a great extent, over. You are now required to ensure the same superior quality and consistency to analysis as you would for any other business aspect.

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

6.2 The Basics for Utility Layer

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. You may have spent hours preparing for this moment, only to find that you cannot locate that awesome utility, or you have a version that you cannot be sure of working properly. Been there—and have got the scars to prove it.

If you have data algorithms that you use regularly, I strongly suggest you keep any proof and credentials that show the process to be a high-quality and industry-accepted algorithm. Painful experience has proven to me that you will get tested, and it is essential to back up your data science 100%. The additional value created is the capability to get multiple teams to work on parallel projects and know that each data scientist or engineer is working with clear standards.

6.3 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.

6.4 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.

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. I suggest that you enable your code to balance its ecosystem as regularly as possible. 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.

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.

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.

Most of my bigger customers have separate discovery environments set up, in which you can run jobs without any limiting control mechanisms. This enables data scientists to concentrate on the models and processing and not on complying with the more controlled production requirements.

6.5 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

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

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

7. The Functional Requirements

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."

I use the MoSCoW method (Below Table) as a prioritization technique, to indicate how important each requirement is to the business. I revisit all outstanding requirements before each development cycle, to ensure that I concentrate on the requirements that are of maximum impact to the business at present, as businesses evolve, and you must be aware of their true requirements.

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

7.1 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.

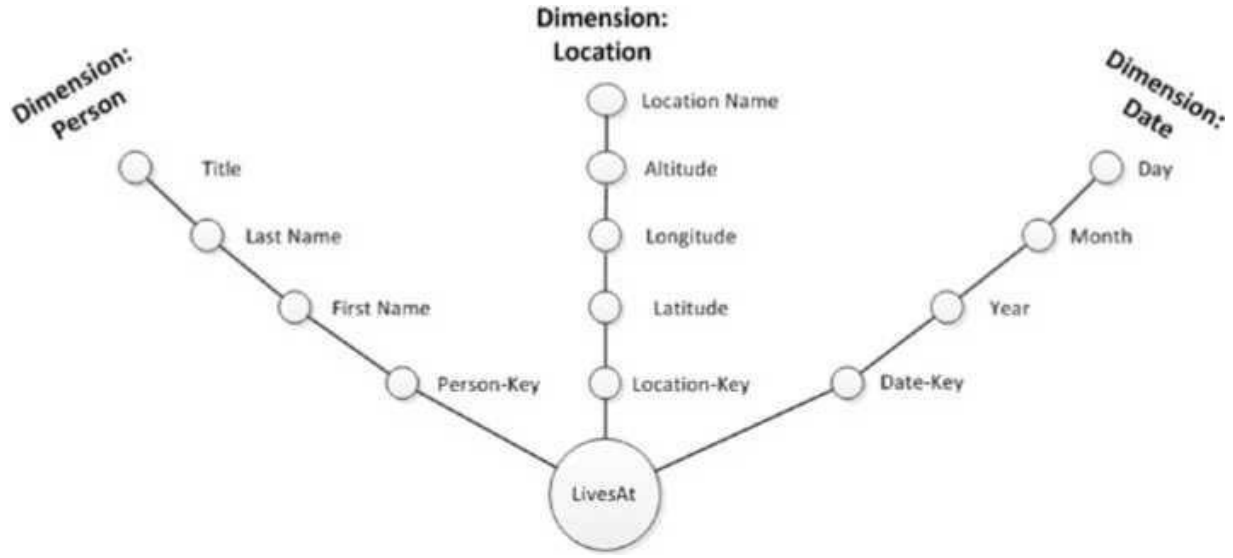
7.1.1 Data Mapping Matrix

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

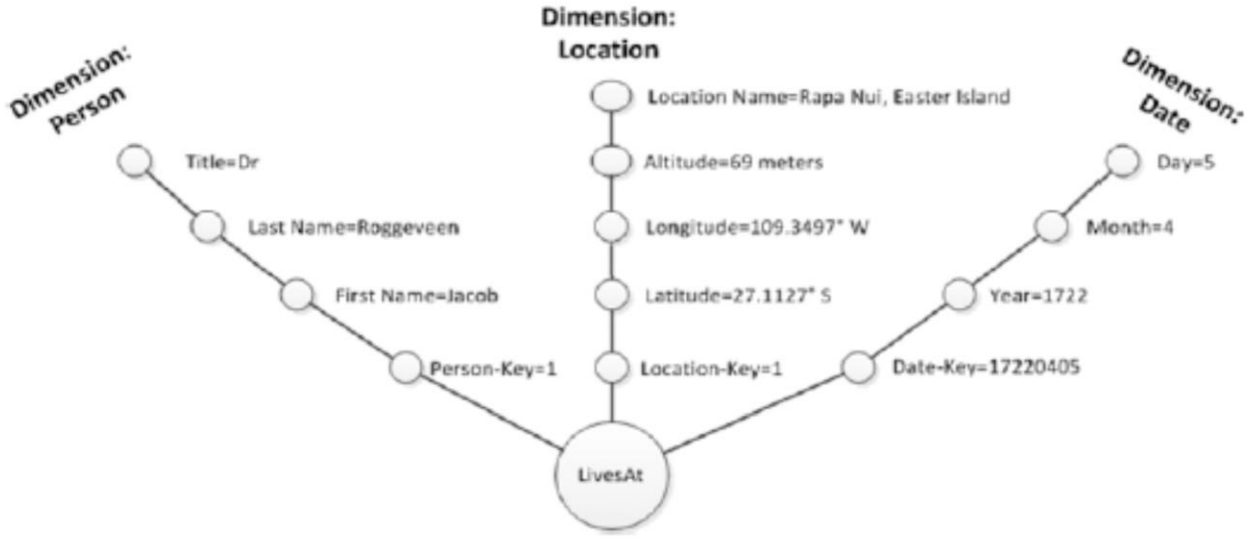
The sun models is a requirement mapping technique that assists you in recording requirements at a level that allows your nontechnical users to understand the intent of your analysis, while providing you with an easy transition to the detailed technical modeling of your data scientist and data engineer.

This sun model is for fact LivesAt and supports three dimensions: person, location, and date. Figure shows a typical sun model.



Typical sun model

This can be used to store the fact that a specific person lived at a specific location since a specific date. So, if you want to record that Dr Jacob Roggeveen lived on Rapa Nui, Easter Island, since April 5, 1722, your data would fit into the sun model, as seen in Figure 4-2.



Dr Jacob Roggeveen record in sun model

Components of Sun Model

1) 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.

A) SCD Type 1—Only Update

This dimension contains only the latest value for specific business information.

Example: Dr Jacob Roggeveen lived on Rapa Nui, Easter Island, but now lives in Middelburg, Netherlands.

oad	First Name	Last Name	Address
	Jacob	Roggeveen	Rapa Nui, Easter Island
	Jacob	Roggeveen	Middelburg, Netherlands

The SCD Type 1 records only the first place Dr Jacob Roggeveen lived and ignores the second load's

son		Location
t Name	Last Name	Address
ob	Roggeveen	Rapa Nui, Easter Island

changes.

This useful for storing the first address you register for a user, or the first product he or she has bought.

B) SCD Type 2—Keeps Complete History

This dimension contains the complete history of the specific business information.
There are three ways to construct a SCD Type 2.

1. *Versioning:* The SCD keeps a log of the version number, allowing you to read the data back in the order in which the changes were made.
2. *Flagging:* The SCD flags the latest version of the business data row as a 1 and the rest as 0. This way, you can always get the latest row.
3. *Effective Date:* The SCD keeps records of the period in which the specific value was valid for the business data row.

Example: Dr Jacob Roggeveen lived in Middelburg, Netherlands, and then in Rapa Nui, Easter Island, but now lives in Middelburg, Netherlands.

Load	First Name	Last Name	Address
1	Jacob	Roggeveen	Middelburg, Netherlands
2	Jacob	Roggeveen	Rapa Nui, Easter Island
3	Jacob	Roggeveen	Middelburg, Netherlands

SCD Type 2 records only changes in the location Dr Jacob Roggeveen lived, as it loads the three data loads, using versioning.

Person			Location	
First Name	Last Name	Version	Address	Version
Jacob	Roggeveen	1	Middelburg, Netherlands	1
			Rapa Nui, Easter Island	2
			Middelburg, Netherlands	3

This useful for storing all the addresses at which your customer has lived, without knowing from what date to what date he or she was at each address.

C) SCD Type 3—Transition Dimension

This dimension records only the transition for the specific business information.

Example: Dr Jacob Roggeveen lived in Middelburg, Netherlands, then in Rapa Nui, Easter Island, but now lives in Middelburg, Netherlands.

Load	First Name	Last Name	Address
1	Jacob	Roggeveen	Middelburg, netherlands
2	Jacob	Roggeveen	Rapa Nw, easter Island
3	Jacob	Roggeveen	Middelburg, netherlands

The transition dimension records only the last address change, by keeping one record with the last change in value and the current value. The three load steps are as follows:

Step 1: Lives in Middelburg, Netherlands

Location	
AddressPrevious	Address
	Middelburg, Netherlands

Step 2: Moves from Middelburg, Netherlands, to Rapa Nui, Easter Island

Location	
AddressPrevious	Address
Middelburg, Netherlands	Rapa Nui, Easter Island

Step 3: Moves from Rapa Nui, Easter Island, to Middelburg, Netherlands

Location	
AddressPrevious	Address
Rapa Nui, Easter Island	Middelburg, Netherlands

I use SCD Type 3 to record the transitions to the current values of the customer.

D) SCD Type 4—Fast-Growing Dimension.

This dimension handles specific business information with a high change rate. This enables the data to track the fast changes, without overwhelming the storage requirements.

2) Facts

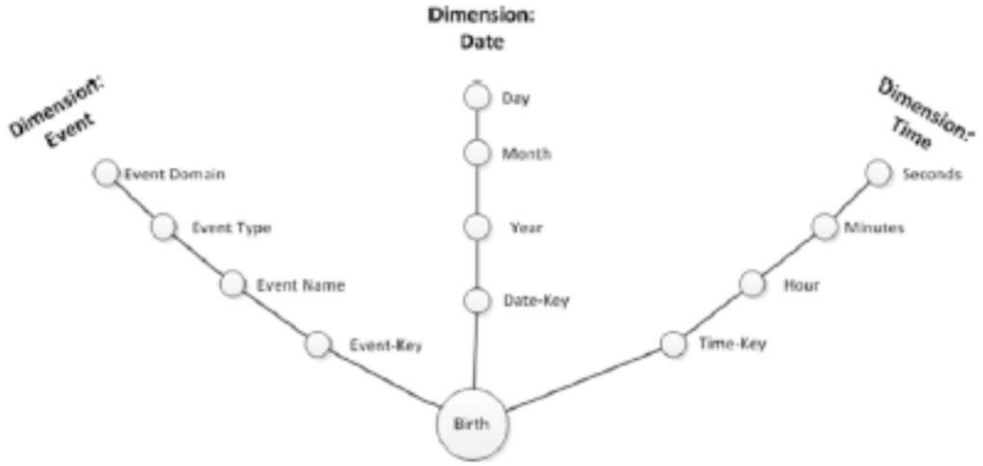
A fact is a measurement that symbolizes a fact about the managed entity in the real world.

7.1.2 Intra-Sun Model Consolidation Matrix

The intra-sun model consolidation matrix is a tool that helps you to identify common dimensions between sun models. Let's assume our complete set of sun models is only three models.

Sun Model One

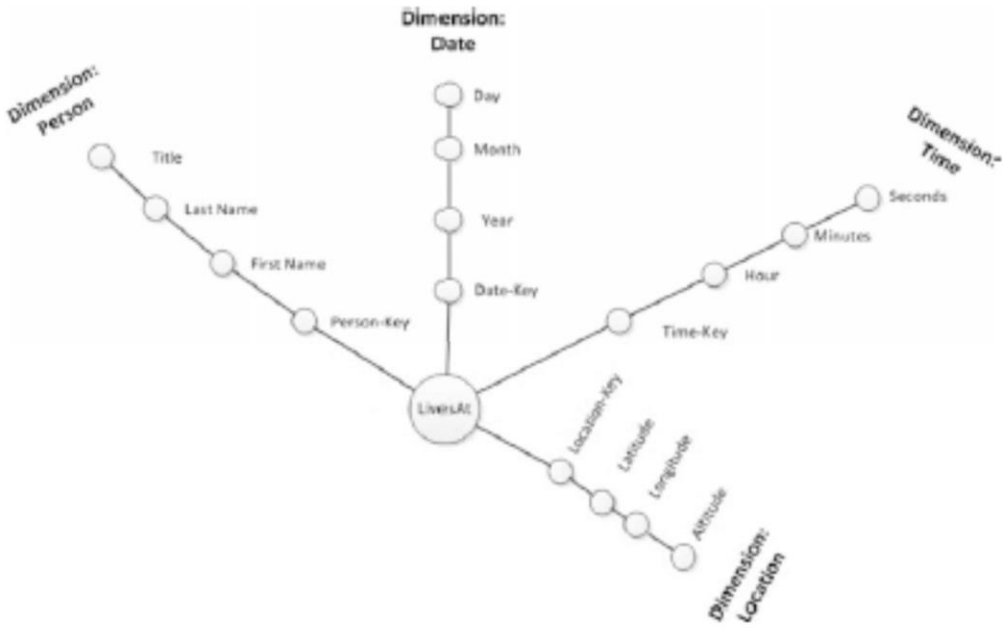
The first sun model (Figure 4-3) records the relationship for Birth, with its direct dependence on the selectors Event, Date, and Time.



ure 4-3. Sun Model One

Sun Model Two

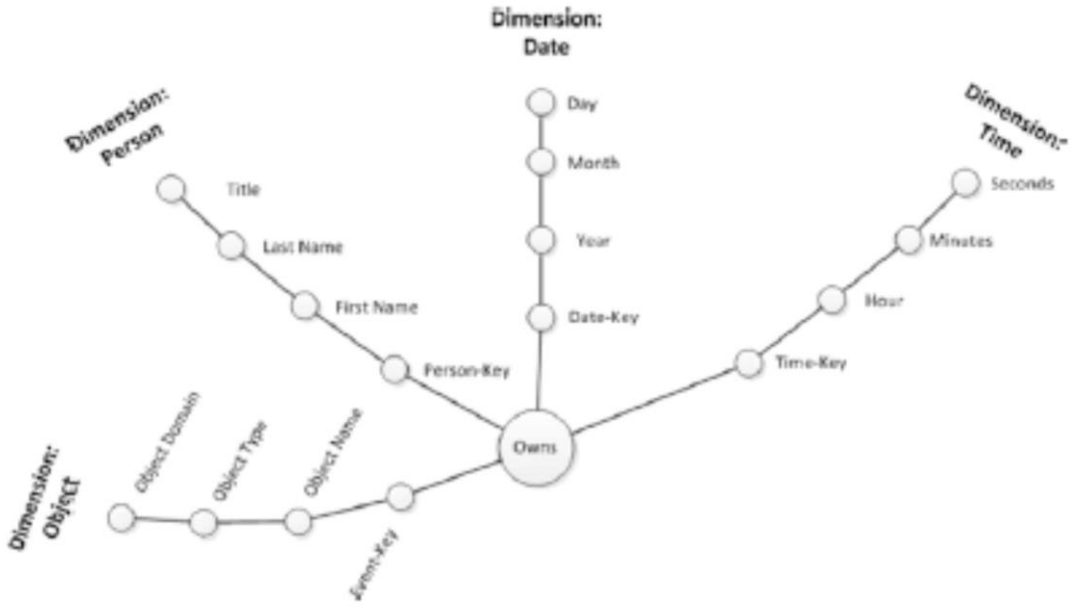
The second sun model (Figure 4-4) records the relationship for LivesAt, with its direct dependence on the selectors Person, Date, Time, and Location.



ure 4-4. Sun Model Two

Sun Model Three

The third sun model (Figure 4-5) records the relationship for Owns, with its direct dependence on the selectors Object, Person, Date, and Time.



ıre 4-5. Sun Model Three

8. The Nonfunctional Requirements

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

8.1 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.

8.2 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. Please check with your local information officer which precise rules apply.

8.3 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. If you are measuring at the edge of the data lake, it is highly possible that you will sustain 99.99999% availability with ease. The distributed and fault-tolerant nature of the data lake technology would ensure a highly available data lake. But if you measure at critical points in the business, you will find that at these critical business points, the requirements are more specific for 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" terminology that I have seen used by some of my previous customers. 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.

The correct requirements are

- London's weekly reports must be entirely operational for 100% of the time between 06h00 and 10h00 (Greenwich Mean Time or British Daylight Time) every Monday.
- New York's weekly reports must be entirely operational for 100% of the time between 06h00 and 10h00 (Eastern Standard Time or Eastern Daylight Time) every Monday.

8.4 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. Oh, yes, that silent moment before every alarm in your business goes mad! You want to be able at all times to return to a known good state via a backup.

8.5 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.

Example:

The memory will support a concurrent group of 100 users running a sort algorithm of 1000 records simultaneously.

Concurrency is an important requirement to ensure an effective solution at the start. Capacity can be increased by adding extra processing resources, while concurrency normally involves complete replacements of components.

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.

You will need short-term storage on fast solid-state drives to handle the while-processing capacity requirements.

The next requirement is your long-term storage. The basic rule is to plan for bigger but slower storage. Investigate using clustered storage, whereby two or more storage servers work together to increase performance, capacity, and reliability. Clustering distributes workloads to each server and manages the transfer of workloads between servers, while ensuring availability.

The use of clustered storage will benefit you in the long term, during periods of higher demand, to scale out vertically with extra equipment.

The big data evolution is now bringing massive amounts of data into the processing ecosystem. So, make sure you have enough space to store any data you need.

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.

The big advantage is to connect an effective quantity of very high-speed memory as closely as possible to these thousands of processing units, to use this increased capacity. I am currently deploying systems such as Kinetic DB and MapD, which are GPU-based data base engines. This improves the processing of my solutions by factors of a hundred in speed.

8.6 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.

8.7 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.

8.7 Documentation

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

8.8 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.

8.9 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. As I suggested before, you must practice with your tools at different scales.

8.10 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.

8.11 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.

8.12 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. I have found that most of the tools I would include in my ecosystem have adequate fault management and reporting capability already built in to their native internal processing.

8.13 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.

Here, I normally stipulate the precise operational system monitoring, measuring, and management requirements within the ecosystem, when faults are recorded. Acceptance script S withstands the X faults it generates. As a data scientist, you are required to ensure that your data science algorithms can handle faults and recover from them in an orderly manner.

8.14 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.

8.15 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. I normally investigate areas such as communication protocols, locations of servers, operating systems for different subcomponents, and the now-important end user's Internet access criteria.

8.16 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.

8.17 Modifiability

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

8.18 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.

8.19 Privacy

I suggest 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.

8.20 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.

8.21 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.

8.22 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.

8.23 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.

8.24 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.

8.25 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. The requirement here is that you use approved and accepted standards to validate your own results.

8.26Scalability

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). The ecosystem grows by scale out, by adding more servers to the data science cluster of resources.

Vertical scalability increases capacity by adding more resources (more memory or an additional CPU) to an individual machine.

Dynamic (on-demand) scalability increases capacity by adding more resources, using either public or private cloud capability, which can be increased and decreased on a pay-as-you-go model. This is a hybrid model using a core set of resources that is the minimum footprint of the system, with additional burst agreements to cover any planned or even unplanned extra scalability increases in capacity that the system requires.

8.27 Security

One of the most important nonfunctional requirements is security. I specify security requirements at three levels.

8.28 Privacy

I would specifically note requirements that specify protection for sensitive information within the ecosystem. Types of privacy requirements to note include data encryption for database tables and policies for the transmission of data to third parties.

8.29 Physical

I would specifically note requirements for the physical protection of the system. Include physical requirements such as power, elevated floors, extra server cooling, fire prevention systems, and cabinet locks.

8.30 Access

I purposely specify detailed access requirements with defined account types/groups and their precise access rights.

8.31 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.

8.32 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. Running distributed algorithms is not easy to deal with, as the distribution of the workload is not under your control.

8.33 Isolate Ability

The specific degree to which I can isolate the code under test will drive most of the possible testing. A process such as deep learning includes non-isolation, so do not accept requirements that you cannot test, owing to not being able to isolate them.

8.34 Understandability

I have found that most algorithms have undocumented "extra features" or, in simple terms, "got-you" states. The degree to which the algorithms under test are documented directly impacts the testability of requirements.

8.35 Automatability

I have found the degree to which I can automate testing of the code directly impacts the effective and efficient testing of the algorithms in the ecosystem. I am an enthusiast of known result inline testing. I add code to my algorithms that test specific sub-sessions, to ensure that the new code has not altered the previously verified code.

9. Engineering a Practical Business Layer

The business layer follows general business analysis and project management principals.

9.1Requirements

Every requirement must be recorded with full version control, in a requirement-per-file manner.

9.2Requirements Registry

Keep a summary registry of all requirements in one single file, to assist with searching for specific requirements. I suggest you have a column with the requirement number, MoSCoW, a short description, date created, date last version, and status. I normally use the following status values:

- In-Development
- In-Production
- Retired

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

9.3Traceability Matrix

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. An example of a template is shown in Figure.

Practical Data Science										
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										

10 Utility Layer

The utility layer is used to store repeatable practical methods of data science. The objective of this chapter is to define how the utility layer is used in the ecosystem.

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, I suggest that you keep any proof and credentials that show that the process is a high-quality, industry-accepted algorithm. Hard experience has taught me that you are likely to be tested, making it essential to prove that your science is 100% valid.

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. In several industries, it is a regulated requirement to use only sanctioned algorithms.

On May 25, 2018, a new European Union General Data Protection Regulation (GDPR) goes into effect. The GDPR has the following rules:

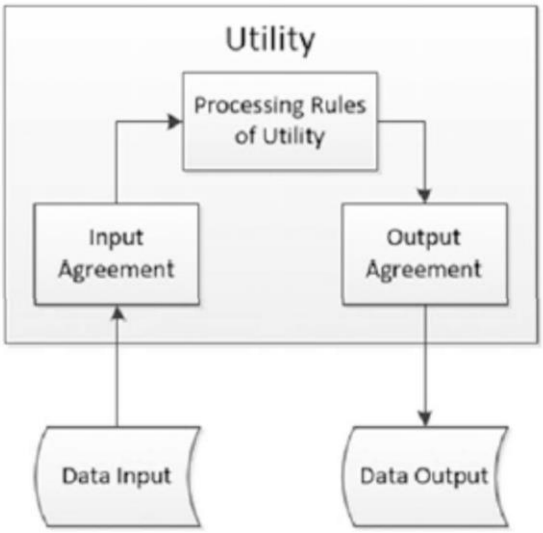
- You must assure transparency, with clear information about what data is collected and howit is processed. Utilities must generate complete audit trails of all their activities.
- You must support the right to accurate personal data. Utilities must use only the latestaccurate data.
- You must support the right to have personal data erased. Utilities must support the removalof all information on a specific person.
- You must have approval to move data between service providers. I advise you to make sure you have 100% approval to move data between data providers. If you move the data from your customer's systems to your own systems without clear approval, bothyou and your customer may be in trouble with the law.
- You must support the right not to be subject to a decision based solely on automatedprocessing.

11 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.

Use a basic design (Figure) for a processing utility, by building it a three-stage process.

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



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.

You can also enhance the processing capability of your entire project collection with one single new utility update.

There are three types of utilities

- Data processing utilities
- Maintenance utilities
- Processing utilities

Data processing utilities

Data processing utilities are grouped for the reason that they perform some form of data transformation within the solutions.

Retrieve Utilities

Utilities for this superstep contain the processing chains for retrieving data out of the raw data lake into a new structured format. I will give specific implementations of these utilities in Chapter 7.

I suggest that you build all your retrieve utilities to transform the external raw data lake format into theHomogeneous Ontology for Recursive Uniform Schema (HORUS) data format that I have been using in my projects.

HORUS is my core data format. It is used by my data science framework, to enable the reduction of development work required to achieve a complete solution that handles all data formats.

---XXX---