Data Science
UNIT - I

chapter 1 : Data Science Technology Stack:

Rapid Information Factory Ecosystem:
Data science ecosystem has a series of tools that you use to build your solutions.

Data Science Storage Tools:
Schema-on-Write and Schema-on-Read:
These 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 structured data schemas we need to run standard SQL queries.
- **Benefits:**
- Tools can only work once the schema is described, so there is only one view on the data.
- Already relationships configured.
- An efficient way to store "dense" data.
- All the data is in the same data store.
- **Drawbacks:**
- Schemas are purpose-built, which makes them hard to change and maintain.
- Loses the raw/atomic data as a source for future analysis.
- 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.

- **Schema-on-Read Ecosystems:**
- Does not require a schema before you can load the data.
- You store the data with minimum structure.
- Schema is applied during the query phase.
- **Benefits:**
- Provides flexibility to store unstructured, semi-structured, and disorganized data.
- 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.
- Increases the speed of generating fresh actionable knowledge.
- Reduces the cycle time between data generation to availability of actionable knowledge.

- **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.
- 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.
- Deployed using distributed data object storage.
- 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.

- **Data Vault:**
- 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, inks, and satellites.
- 1) Hubs:
- Hubs contain a list of unique business keys.
- 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.
- 2) 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.
- 3) Satellites:
- Hubs and links form the structure of the model but store no descriptive characteristics of the data.
- These characteristics are stored in appropriated tables identified as satellites.
- The appropriate combination of hubs, links, and satellites helps the data scientist to construct and store prerequisite business relationships.

- **Data Warehouse Bus Matrix:**
- It is a data warehouse planning tool and model.
- 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.

- **Data Science Processing Tools:**
- They transform your data lakes into data vaults and then into data warehouses.
- These tools are the workhorses of the data science and engineering ecosystem.
- **Spark:**
- Apache Spark is an open source cluster computing framework.
- It offers an interface for programming distributed clusters with implicit data parallelism and fault-tolerance.
- Spark Core:
- Spark Core is the foundation of the overall development.
- It provides distributed task dispatching, scheduling, and basic I/O functionalities.
- 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.
- Spark Streaming:
- Spark Streaming has built-in support to consume from Twitter 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.
- 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.
- Ex: Collaborative filtering techniques, Cluster analysis methods, Optimization algorithms and 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.

- **Mesos**
- Apache Mesos is an open source cluster manager.
- It delivers efficient resource isolation and sharing across distributed applications.
- The software enables resource sharing in a fine-grained manner, improving cluster utilization.

- **Akka:**
- Akka is an actor-based message-driven runtime for running concurrency and elasticity.
- 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.
- It is a type of NoSQL database.
- A NoSQL database (sometimes called as NotOnlySQL) is a database that provides a mechanism to store and retrieve data other than the tabular relations used in relational databases.
- These databases are schema free, support easy replication, have simpleAPI, eventually consistent, and can handle huge amounts of data.

- **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.
- Kafka Core:
- 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.
- 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.

- 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.
- • forecast (forecasting of time series).
- • 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

- **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.

- **Python:**
- Python is a high-level, general-purpose programming language.
- It is an interpreted language.
- Uses dynamic type system and automatic memory management and supports multiple programming paradigms (object-oriented, functional programming, and procedural).
- The Python Package Index (PyPI) (https://pypi.python.org/pypi) supplies thousands of third-party modules ready for use for your data science projects.

- **MQTT:**
- MQTT stands for Message Queuing 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.

chapter 2 : Layered Framework:

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.

- **Cross-Industry Standard Process for Data Mining(CRISP-DM):**
- **Business Understanding:**
- 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.
- The DMN standard offers businesses a modeling notation for describing decision management flows and business rules.
-

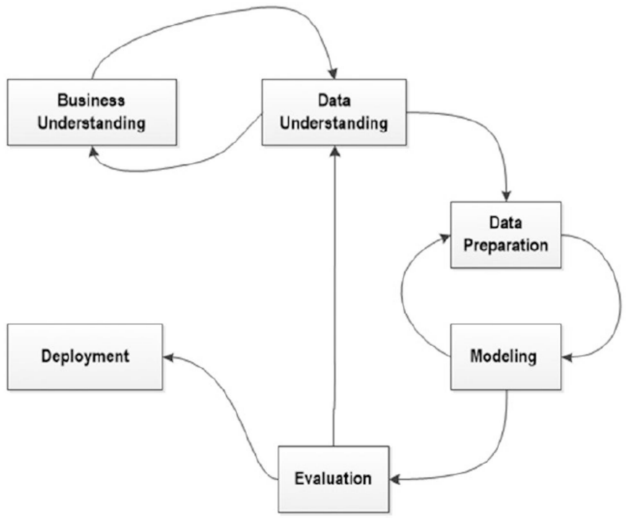


Figure 3-1. CRISP-DM flowchart

- **Data Understanding:**
- 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:**
- Covers all activities to construct the final data set for modeling tools.
- This phase is used in a cyclical order with the modelling phase, to achieve a complete model.
- This ensures you have all the data required for your data science.
- **Modeling:**
- 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:**
- 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.

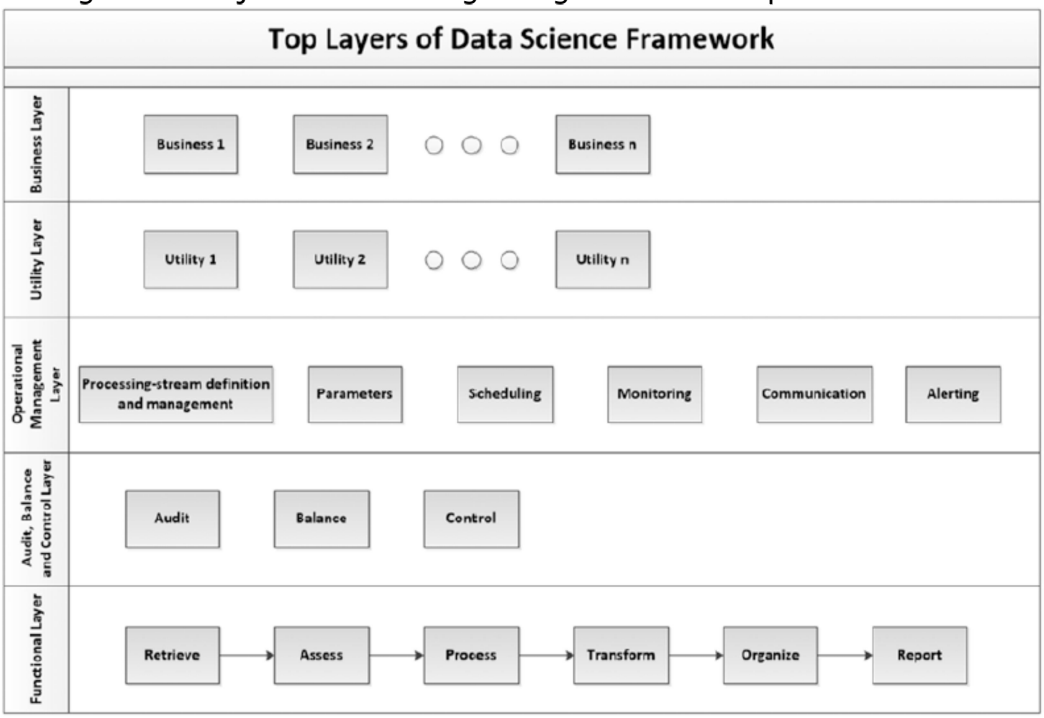
Homogeneous Ontology (study of what kinds of things exist) for Recursive Uniform Schema:

- It is used as an internal data format structure that enables the framework to reduce the permutations of transformations required by the framework.
- 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.
- The use of HORUS methodology results in a hub-and-spoke data transformation approach.

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 layers enable you to keep track of all the processing and findings you achieve.

- Using the framework will prepare you to work in an environment, with multiple data scientists working on a common ecosystem.
- 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.



- **The Basics for Business Layer:**
- It 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:**
- A utility is a process that is used by many data science projects to achieve common processing methodologies.
- It is a common area in which you store all your utilities.
- Collect your utilities (including source code) in one central location.
- Keep detailed records of every version.
- 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.
- **The Basics for Operational Management Layer:**
- 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 Basics for Audit, Balance, and Control Layer:**
- The audit, balance, and control layer, 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 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 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.
- You can balance your processing requirements by removing or adding resources dynamically as you move through the processing pipe.
- **Control:**
- The control sublayer, controls the execution of the current active data science processes in a production ecosystem.
- 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.
- **The Basics for Functional Layer:**
- The main layer of programming required.
- It consists of several structures.
- • Data models
- • Processing algorithms
- • Provisioning of infrastructure
- **six supersteps of processing:**
- 1. Retrieve: for retrieving data from the raw data lake via a more structured format.
- 2. Assess: for quality assurance and additional data enhancements.
- 3. Process: for building the data vault.
- 4. Transform: for building the data warehouse.
- 5. Organize: for building the data marts.
- 6. Report: for building virtualization and reporting the actionable knowledge.

- **Layered Framework for High-Level Data Science and Engineering:**
- The layered framework is engineered with the following structures.
- The following scripts will create for you a complete framework in the C:\VKHCG\05-DS\00-Framework directory, if you are on Windows, or ./VKHCG/05-DS/00-Framework, if you are using a Linux environment.
- **Windows**
- If you prefer Windows and Python, in the source code, you will find the relevant Python script, identified as Build_Win_Framework.py.
- If you prefer Windows and R, in the source code , you will find the relevant Python script, identified as Build_Win_Framework.r.
- **Linux**
- If you prefer Linux and Python, in the source code you will find the relevant Python script, identified as Build_ Linux _Framework.py.
- If you prefer Linux and R, in the source code, you will find the relevant Python script, identified as Build_ Linux_Framework.r.

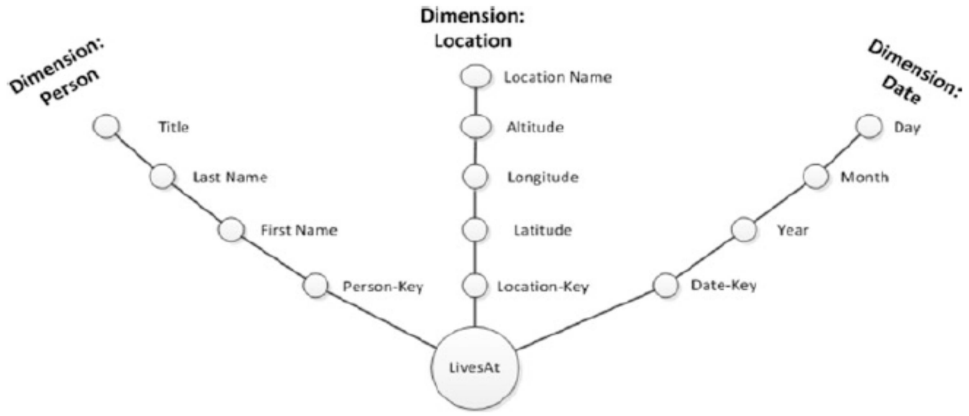
Chapter 3: Business Layer:

- **Business Layer:**
- 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:**
- 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."

Table 4-1. MoSCoW Options

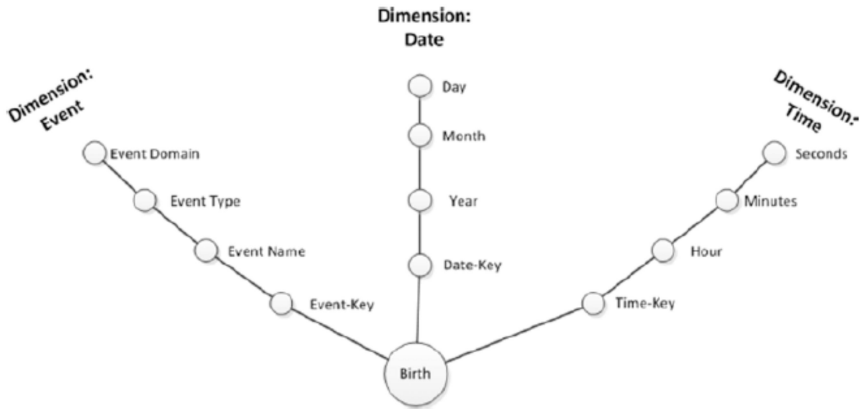
Must have	Requirements with the priority "must have" are critical to the current delivery cycle.
Should have	Requirements with the priority "should have" are important but not necessary to the current delivery cycle.
Could have	Requirements prioritized as "could have" are those that are desirable but 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 thoseidentified by stakeholders as the least critical, lowest payback requests, or just not appropriate at that time in the delivery cycle.

- **General Functional Requirements:**
- As a [user role] I want [goal] so that [business value] is achieved.
- **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.
- 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.
- 2) Sun Models:
- Learn to evolve continually to survive!
- 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.
- Ex: This sun model is for fact LivesAt and supports three dimensions: person, location, and date.
- **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.
-

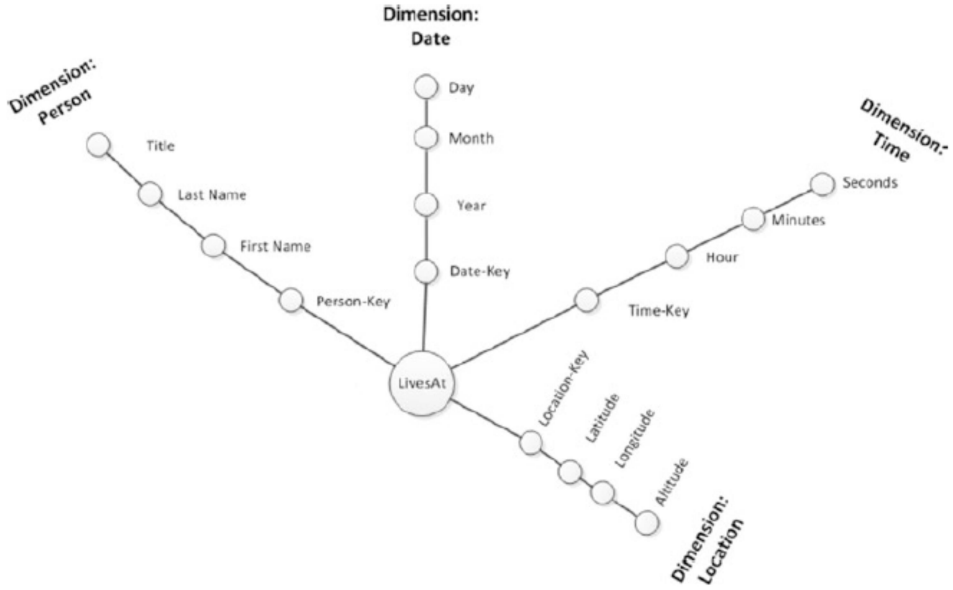


- ***ure 4-1.*** *Typical sun model*
- **SCD Type 1—Only Update:**
- This dimension contains only the latest value for specific business information.
- Example: Dr Jacob Roggeveen lived on Rapa Nui, Easter Island, but now lives in Middelburg, Netherlands.
- Load First Name Last Name Address
- 1 Jacob Roggeveen Rapa Nui, Easter Island
- 2 Jacob Roggeveen Middelburg, Netherlands
- The SCD Type 1 records only the first place Dr Jacob Roggeveen lived and ignores the second load's changes.
- Person Location
- First Name Last Name Address
- Jacob Roggeveen Rapa Nui, Easter Island
- **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
- SCD Type 2 records only changes to the places Dr Jacob Roggeveen has lived, each time it loads the three data loads, using flagging.
- Person Location

- First Name Last Name Flag Address Flag
- Jacob Roggeveen 1 Middelburg, Netherlands 0
  Rapa Nui, Easter Island 0
  Middelburg, Netherlands 1
- SCD Type 2 records only changes in the location Dr Jacob Roggeveen lived, as it loads the three data loads, using effective date.
- Person Location
- First Name Last Name Effective Date Address Effective Date
- Jacob Roggeveen 5 April 1722 Middelburg, Netherlands 1 February 1659
  Rapa Nui, Easter Island 5 April 1722
  Middelburg, Netherlands 31 January 1729
- **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.
- 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
- **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.
- **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.
- Let’s assume our complete set of sun models is only three models.
- Sun Model One
- records the relationship for Birth, with its direct dependence on the selectors Event, Date, and Time.



- Sun Model Two
- records the relationship for LivesAt, with its direct dependence on the selectors Person, Date, Time, and Location.



- Sun Model Three:
- records the relationship for Owns, with its direct dependence on the selectors Object, Person, Date, and Time.
- consolidate the different dimensions and facts onto a single matrix:

	Date (SCD 0)	Time (SCD 0)	Person (SCD 2)	Object (SCD 2)	Location (SCD 2)	Event (SCD 2)
Birth	X	X				X
LivesAt	X	X	X		X	
Owns	X	X	X	X		

- **The Nonfunctional Requirements:**
- 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.

- **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.
- The distributed and fault-tolerant nature of the data lake technology would ensure a highly available data lake.
- 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.
- 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.
- You can clearly see that these requirements are now more precise than the simple general requirement.
- 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.
- Please ensure that you can restore your backups in an effective and efficient manner.
- The process is backup-and-restore.
- Just generating backups does not ensure survival.

- **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.

- **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.

- **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.

- **Year-on-Year Growth Requirements:**
- The biggest growth in capacity will be for long-term storage.
- These requirements are specified as how much capacity increases over a period.
- The correct way to record the requirement is
- Component C will be responsible for necessary growth capacity to handle additional M MB of data within a period of T.

- **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.

- **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.
- Acceptance script S completes and reports every one of the X faults it generates.
- As a data scientist, you are required to log any failures of the system, to ensure that no unwanted side effects are generated that may cause a detrimental impact to your customers.

- **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.
- 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.
- **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:**
- List 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.
- 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.

- **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.

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

- **Scalability:**
- Scalability is how you get the data science ecosystem to adapt to your requirements.
- 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.

- **Security:**
- One of the most important nonfunctional requirements is security.
- specify security requirements at three levels.

- **Privacy:**
- 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.

- **Physical:**
- 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.

- **Access:**
- Specify detailed access requirements with defined account types/groups and their precise access rights.

- **Testability:**
- 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:**
- 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.
- **Isolate Ability:**
- 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.

- **Understandability:**

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

- **Automatability:**
- The degree to which one can automate testing of the code directly impacts the effective and efficient testing of the algorithms in the ecosystem.
- Ensure that the new code has not altered the previously verified code.

- **Common Pitfalls with Requirements:**
- **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"?
- • User-friendly report layouts
- What is "user-friendly"?
- • Secure access to systems.
- What is "secure?
- • All data must be immediately reported.
- What is "all"? What is "immediately"?
- Make sure the wording of your requirements is precise and specific.

- **Unbounded Lists:**
- An unbounded list is an incomplete list of items. Examples include the following:
- • Accessible at least from London and New York.
- Do I connect only London to New York? What about the other 20 branches?
- • Including, but not limited to, London and New York offices must have access.
- So, is the New Delhi office not part of the solution?
- Make sure your lists are complete and precise.
- **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"?
- • What belongs to the collection of "other network protocols"?
- • What specific protocols of TCP/IP are included?
- Make sure your collections are explicit and precise.

- **Ambiguity:**
- Ambiguity occurs when a word within the requirement has multiple meanings.
- Examples are listed following.
- **Vagueness:**

- The system must pass between 96–100% of the test cases using current standards for data science.
- What are the "current standards"? This is an example of an unclear requirement!
- **Subjectivity:**
- The report must easily and seamlessly integrate with the web sites.
- "Easily" and "seamlessly" are highly subjective terms where testing is concerned.
- **Optionality:**
- The solution should be tested under as many hardware conditions as possible.
- "As possible" makes this requirement optional.
- What if it fails testing on every hardware setup? Is that okay with your customer?
- **Under-specification:**
- The solution must support Hive 2.1 and other database versions.
- Do other database versions only include other Hive databases, or also others such as HBase version 1.0 and Oracle version 10i?
- **Under-reference:**
- Users must be able to complete all previously defined reports in less than three minutes 90% of the day.

Engineering a Practical Business Layer:

- The business layer follows general business analysis and project management principals.
- A practical business layer consist of a minimum of three primary structures.
- For the business layer use a directory structure.
- This enables you to keep your solutions clean and tidy for a successful interaction with a standard version-control system.
- **Requirements:**
- Every requirement must be recorded with full version control, in a requirement-per-file manner.
- Suggested a numbering scheme of 000000-00, which supports up to a million requirements with up to a hundred versions of each requirement.

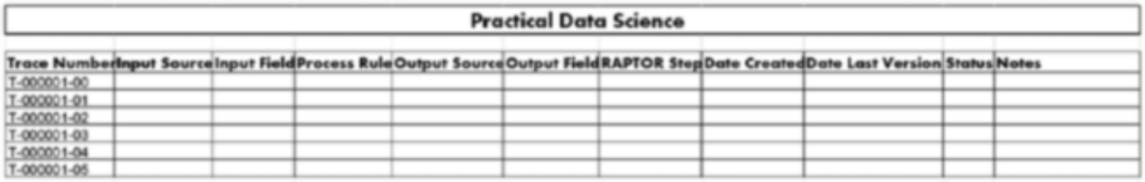
- **Requirements Registry:**
- Keep a summary registry of all requirements in one single file, to assist with searching for specific requirements.
- Suggested to have a column with the requirement number, MoSCoW, a short description, date created, date last version, and status.
- Normally use the following status values:
- • In-Development
- • In-Production
- • Retired
- The register acts as a control for the data science environment's requirements.

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						

- *4-7. Requirements registry template*

- **Traceability 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.



Practical Data Science

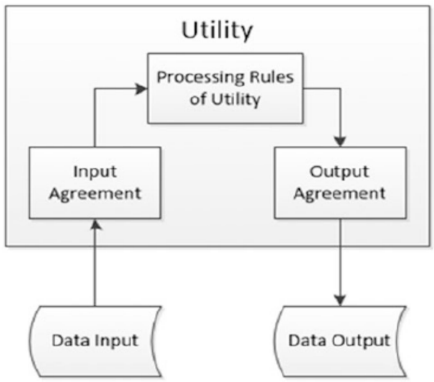
Trace Number	Input Source	Input Field	Process Rule	Output Source	Output Field	RAPTOR Step	Date Created	Date Last Version	Status	Notes
T-000001-00										
T-000001-01										
T-000001-02										
T-000001-03										
T-000001-04										
T-000001-05										

- *4-8. Traceability matrix template*

Chapter 4: 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 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.
- On May 25, 2018, a new European Union General Data Protection Regulation (GDPR) goes into 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.
- • You must support the right to accurate personal data.
- • You must support the right to have personal data erased.
- • 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.
- Use a basic design 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.
- Start your utility layer with a small utility set that you know works well and build the set out as you go along.
- There are three types of utilities
- • Data processing utilities
- • Maintenance utilities
- • Processing utilities
- **1) 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.
- Build all your retrieve utilities to transform the external raw data lake format into the Homogeneous Ontology for Recursive Uniform Schema (HORUS) data format.
- HORUS is core data format.
- It is used to enable the reduction of development work required to achieve a complete solution that handles all data formats.
- **1] Text-Delimited to HORUS:**
- These utilities enable your solution to import text-based data from your raw data sources.
- Example: This utility imports a list of countries in CSV file format into HORUS format.
- import pandas as pd
- sInputFileName='C:/VKHCG/05-DS/9999-Data/Country_Code.csv'
- InputData=pd.read_csv(sInputFileName,encoding="latin-1")
- ProcessData=InputData
- # Remove columns ISO-2-Code and ISO-3-CODE
- ProcessData.drop('ISO-2-CODE', axis=1,inplace=True)
- ProcessData.drop('ISO-3-Code', axis=1,inplace=True)
- # Rename Country and ISO-M49
- ProcessData.rename(columns={'Country': 'CountryName'}, inplace=True)
- ProcessData.rename(columns={'ISO-M49': 'CountryNumber'}, inplace=True)
- # Set new Index
- ProcessData.set_index('CountryNumber', inplace=True)
- # Sort data by CurrencyNumber

- ProcessData.sort_values('CountryName', axis=0, ascending=False, inplace=True)
- print(ProcessData)
- OutputData=ProcessData
- sOutputFileName='C:/VKHCG/05-DS/9999-Data/HORUS-CSV-Country.csv'
- OutputData.to_csv(sOutputFileName, index = False)

- **2] XML to HORUS:**
- These utilities enable your solution to import XML-based data from your raw data sources.
- Example: This utility imports a list of countries in XML file format into HORUS format.
- import pandas as pd
- import xml.etree.ElementTree as ET
- def df2xml(data):
    - header = data.columns
    - root = ET.Element('root')
    - for row in range(data.shape[0]):
        - entry = ET.SubElement(root,'entry')
        - for index in range(data.shape[1]):
            - schild=str(header[index])
            - child = ET.SubElement(entry, schild)
            - if str(data[schild][row]) != 'nan':
                - child.text = str(data[schild][row])
            - else:
                - child.text = 'n/a'
            - entry.append(child)
    - result = ET.tostring(root)
    - return result
- def xml2df(xml:data):
    - root = ET.XML(xml:data)
    - all_records = []
    - for i, child in enumerate(root):
        - record = {}
        - for subchild in child:
            - record[subchild.tag] = subchild.text
        - all_records.append(record)
    - return pd.DataFrame(all_records)
- # Input Agreement ==========
- sInputFileName='C:/VKHCG/05-DS/9999-Data/Country_Code.xml'
- InputData = open(sInputFileName).read()
- # Processing Rules ==========
- ProcessDataXML=InputData
- # XML to Data Frame
- ProcessData=xml2df(ProcessDataXML)
- # Remove columns ISO-2-Code and ISO-3-CODE
- ProcessData.drop('ISO-2-CODE', axis=1,inplace=True)
- ProcessData.drop('ISO-3-Code', axis=1,inplace=True)
- # Rename Country and ISO-M49
- ProcessData.rename(columns={'Country': 'CountryName'}, inplace=True)
- ProcessData.rename(columns={'ISO-M49': 'CountryNumber'}, inplace=True)
- # Set new Index

- ProcessData.set_index('CountryNumber', inplace=True)
- # Sort data by CurrencyNumber
- ProcessData.sort_values('CountryName', axis=0, ascending=False, inplace=True)
- # Output Agreement
  ==========
- OutputData=ProcessData
- sOutputFileName='C:/VKHCG/05-DS/9999-Data/HORUS-XML-Country.csv'
- OutputData.to_csv(sOutputFileName, index = False)

- **3] JSON to HORUS:**
- These utilities enable your solution to import Json-based data from your raw data sources.
- Example: This utility imports a list of countries in JSON file format into HORUS format.
- import pandas as pd
- # Input Agreement
  ==========
- sInputFileName='C:/VKHCG/05-DS/9999-Data/Country_Code.json'
- InputData=pd.read_json(sInputFileName, orient='index', encoding="latin-1")
- # Processing Rules ==========
- ProcessData=InputData
- # Remove columns ISO-2-Code and ISO-3-CODE
- ProcessData.drop('ISO-2-CODE', axis=1,inplace=True)
- ProcessData.drop('ISO-3-Code', axis=1,inplace=True)
- # Rename Country and ISO-M49
- ProcessData.rename(columns={'Country': 'CountryName'}, inplace=True)
- ProcessData.rename(columns={'ISO-M49': 'CountryNumber'}, inplace=True)
- # Set new Index
- ProcessData.set_index('CountryNumber', inplace=True)
- # Sort data by CurrencyNumber
- ProcessData.sort_values('CountryName', axis=0, ascending=False, inplace=True)
- # Output Agreement
  ==========
- OutputData=ProcessData
- sOutputFileName='C:/VKHCG/05-DS/9999-Data/HORUS-JSON-Country.csv'
- OutputData.to_csv(sOutputFileName, index = False)

- **4] Database to HORUS:**
- These utilities enable your solution to import data from existing database sources.
- Example: This utility imports a list of countries in SQLite data format into HORUS format.
- import pandas as pd
- import sqlite3 as sq
- # Input Agreement
  ==========
- sInputFileName='C:/VKHCG/05-DS/9999-Data/utility.db'
- sInputTable='Country_Code'
- conn = sq.connect(sInputFileName)
- sSQL='select * FROM ' + sInputTable + ';'
- InputData=pd.read_sql_query(sSQL, conn)
- # Processing Rules ==========
- ProcessData=InputData
- # Remove columns ISO-2-Code and ISO-3-CODE
- ProcessData.drop('ISO-2-CODE', axis=1,inplace=True)
- ProcessData.drop('ISO-3-Code', axis=1,inplace=True)
- # Rename Country and ISO-M49
- ProcessData.rename(columns={'Country': 'CountryName'}, inplace=True)
- ProcessData.rename(columns={'ISO-M49': 'CountryNumber'}, inplace=True)
- # Set new Index
- ProcessData.set_index('CountryNumber', inplace=True)

- # Sort data by CurrencyNumber
- ProcessData.sort_values('CountryName', axis=0, ascending=False, inplace=True)
- # Output Agreement ==========
- OutputData=ProcessData
- sOutputFileName='C:/VKHCG/05-DS/9999-Data/HORUS-CSV-Country.csv'
- OutputData.to_csv(sOutputFileName, index = False)

- **5] Picture to HORUS:**
- These expert utilities enable your solution to convert a picture into extra data.
- These utilities identify objects in the picture, such as people, types of objects, locations, and many more complex data features.
- Example: This utility imports a picture of a dog called Angus in JPG format into HORUS format.
- from scipy.misc import imread
- import pandas as pd
- import matplotlib.pyplot as plt
- import numpy as np
- # Input Agreement ==========
- sInputFileName='C:/VKHCG/05-DS/9999-Data/Angus.jpg'
- InputData = imread(sInputFileName, flatten=False, mode='RGBA')
- print('Input Data Values ==========')
- print('X: ',InputData.shape[0])
- print('Y: ',InputData.shape[1])
- print('RGBA: ', InputData.shape[2])
- # Processing Rules ==========
- ProcessRawData=InputData.flatten()
- y=InputData.shape[2] + 2
- x=int(ProcessRawData.shape[0]/y)
- ProcessData=pd.DataFrame(np.reshape(ProcessRawData, (x, y)))
- sColumns= ['XAxis','YAxis','Red', 'Green', 'Blue','Alpha']
- ProcessData.columns=sColumns
- ProcessData.index.names =['ID']
- plt.imshow(InputData)
- plt.show()
- # Output Agreement ==========
- OutputData=ProcessData
- print('Storing File')
- sOutputFileName='C:/VKHCG/05-DS/9999-Data/HORUS-Picture.csv'
- OutputData.to_csv(sOutputFileName, index = False)

- **6] Video to HORUS:**
- These expert utilities enable your solution to convert a video into extra data.
- These utilities identify objects in the video frames, such as people, types of objects, locations, and many more complex data features.
- Example: This utility imports a movie in MP4 format into HORUS format.
- The process is performed in two stages.
- Movie to Frames
- # Utility Start Movie to HORUS (Part 1) ==========
- import os
- import shutil
- import cv2
- sInputFileName='C:/VKHCG/05-DS/9999-Data/dog.mp4'
- sDataBaseDir='C:/VKHCG/05-DS/9999-Data/temp'
- if os.path.exists(sDataBaseDir):

    - shutil.rmtree(sDataBaseDir)
- if not os.path.exists(sDataBaseDir):
    - os.makedirs(sDataBaseDir)
- vidcap = cv2.VideoCapture(sInputFileName)
- success,image = vidcap.read()
- count = 0
- while success:
    - success,image = vidcap.read()
    - sFrame=sDataBaseDir + str('/dog-frame-' + str(format(count, '04d')) + '.jpg')
    - print('Extracted: ', sFrame)
    - cv2.imwrite(sFrame, image)
    - if os.path.getsize(sFrame) == 0:
        - count += -1
        - os.remove(sFrame)
        - print('Removed: ', sFrame)
    - if cv2.waitKey(10) == 27: # exit if Escape is hit
        - break
    - count += 1
- print('Generated : ', count, ' Frames')
-
- **Frames to Horus**
- from scipy.misc import imread
- import pandas as pd
- import matplotlib.pyplot as plt
- import numpy as np
- import os
- # Input Agreement ==========
- sDataBaseDir='C:/VKHCG/05-DS/9999-Data/temp'
- f=0
- for file in os.listdir(sDataBaseDir):
    - if file.endswith(".jpg"):
    - f += 1
    - sInputFileName=os.path.join(sDataBaseDir, file)
    - print('Process : ', sInputFileName)
    - InputData = imread(sInputFileName, flatten=False, mode='RGBA')
    - print('Input Data Values ==========')
    - print('X: ',InputData.shape[0])
    - print('Y: ',InputData.shape[1])
    - print('RGBA: ', InputData.shape[2])
    - # Processing Rules ==========
    - ProcessRawData=InputData.flatten()
    - y=InputData.shape[2] + 2
    - x=int(ProcessRawData.shape[0]/y)
    - ProcessFrameData=pd.DataFrame(np.reshape(ProcessRawData, (x, y)))
    - ProcessFrameData['Frame']=file
    - print('Process Data Values ==========')
    - plt.imshow(InputData)
    - plt.show()
    - if f == 1:
        - ProcessData=ProcessFrameData
    - else:
        - ProcessData=ProcessData.append(ProcessFrameData)
- if f > 0:
- sColumns= ['XAxis','YAxis','Red', 'Green', 'Blue','Alpha','FrameName']
- ProcessData.columns=sColumns

- ProcessFrameData.index.names =['ID']
- print('Rows: ',ProcessData.shape[0])
- print('Columns :',ProcessData.shape[1])
- # Output Agreement
  ==========
- OutputData=ProcessData
- print('Storing File')
- sOutputFileName='C:/VKHCG/05-DS/9999-Data/HORUS-Movie-Frame.csv'
- OutputData.to_csv(sOutputFileName, index = False)
- print('Processed ; ', f,' frames')

- **7] Audio to HORUS:**
- These expert utilities enable your solution to convert an audio into extra data.
- These utilities identify objects in the video frames, such as people, types of objects, locations, and many more complex data features.
- Example: This utility imports a set of audio files in WAV format into HORUS format.
- from scipy.io import wavfile
- import pandas as pd
- import matplotlib.pyplot as plt
- import numpy as np
- def show_info(aname, a,r):
    - print ("Audio:", aname)
    - print ("Rate:", r)
    - print ("shape:", a.shape)
    - print ("dtype:", a.dtype)
    - print ("min, max:", a.min(), a.max())
    - plot_info(aname, a,r)
- def plot_info(aname, a,r):
    - sTitle= 'Signal Wave - '+ aname + ' at ' + str(r) + 'hz'
    - plt.title(sTitle)
    - sLegend=[]
    - for c in range(a.shape[1]):
        - sLabel = 'Ch' + str(c+1)
        - sLegend=sLegend+[str(c+1)]
        - plt.plot(a[:,c], label=sLabel)
    - plt.legend(sLegend)
    - plt.show()
- sInputFileName='C:/VKHCG/05-DS/9999-Data/2ch-sound.wav'
- print('Processing : ', sInputFileName)
- InputRate, InputData = wavfile.read(sInputFileName)
- show_info("2 channel", InputData,InputRate)
- ProcessData=pd.DataFrame(InputData)
- sColumns= ['Ch1','Ch2']
- ProcessData.columns=sColumns
- OutputData=ProcessData
- sOutputFileName='C:/VKHCG/05-DS/9999-Data/HORUS-Audio-2ch.csv'
- OutputData.to_csv(sOutputFileName, index = False)
- sInputFileName='C:/VKHCG/05-DS/9999-Data/4ch-sound.wav'
- print('Processing : ', sInputFileName)
- InputRate, InputData = wavfile.read(sInputFileName)
- show_info("4 channel", InputData,InputRate)
- ProcessData=pd.DataFrame(InputData)
- sColumns= ['Ch1','Ch2','Ch3', 'Ch4']
- ProcessData.columns=sColumns
- OutputData=ProcessData
- sOutputFileName='C:/VKHCG/05-DS/9999-Data/HORUS-Audio-4ch.csv'
- OutputData.to_csv(sOutputFileName, index = False)

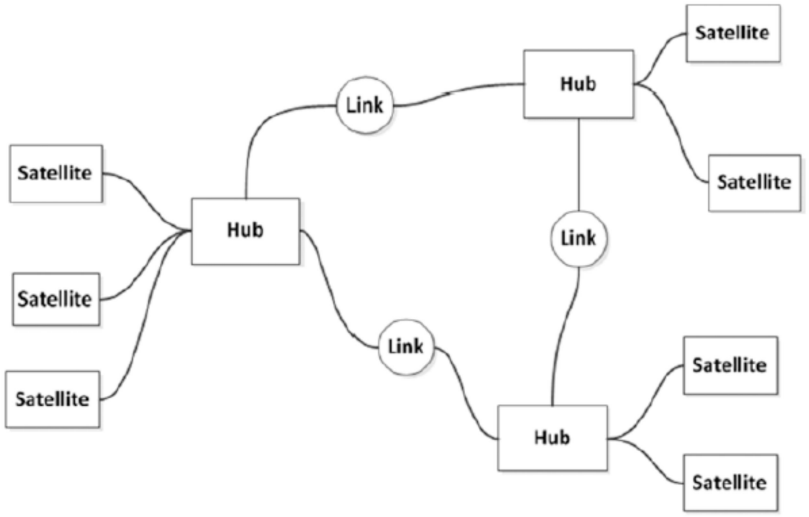
- sInputFileName='C:/VKHCG/05-DS/9999-Data/6ch-sound.wav'
- print('Processing : ', sInputFileName)
- InputRate, InputData = wavfile.read(sInputFileName)
- show_info("6 channel", InputData,InputRate)
- ProcessData=pd.DataFrame(InputData)
- sColumns= ['Ch1','Ch2','Ch3', 'Ch4', 'Ch5','Ch6']
- ProcessData.columns=sColumns
- OutputData=ProcessData
- sOutputFileName='C:/VKHCG/05-DS/9999-Data/HORUS-Audio-6ch.csv'
- OutputData.to_csv(sOutputFileName, index = False)
- sInputFileName='C:/VKHCG/05-DS/9999-Data/8ch-sound.wav'
- print('Processing : ', sInputFileName)
- InputRate, InputData = wavfile.read(sInputFileName)
- show_info("8 channel", InputData,InputRate)
- ProcessData=pd.DataFrame(InputData)
- sColumns= ['Ch1','Ch2','Ch3', 'Ch4', 'Ch5','Ch6','Ch7','Ch8']
- ProcessData.columns=sColumns
- OutputData=ProcessData
- sOutputFileName='C:/VKHCG/05-DS/9999-Data/HORUS-Audio-8ch.csv'
- OutputData.to_csv(sOutputFileName, index = False)

- **8] Data Stream to HORUS:**
- These expert utilities enable your solution to handle data streams.
- Data streams are evolving as the fastest-growing data collecting interface at the edge of the data lake.

- **Assess Utilities**
- Utilities for this superstep contain all the processing chains for quality assurance and additional data enhancements.
- 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.
- Following is a small subset of the utilities you may use:
- **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()
- print('>',cleandata,'<')
- **Adders Utilities**
- Adders use existing data entries and then add additional data entries to enhance your data.
- Examples include
- • Utilities that look up extra data against existing data entries in your solution.
- • Zoning data that is added by extra data entries based on a test.
- The utility can indicate that the data entry is valid, i.e., you found the code in the lookup.
- A utility can indicate that your data entry for bank balance is either in the black or the red.

- **Process Utilities:**
- Utilities for this superstep contain all the processing chains for building the data vault.

- The data vault as a data structure that uses well-structured design to store data with full history.
- The basic elements of the data vault are hubs, satellites, and links.
- There are three basic process utilities.
- **Data Vault Utilities**
- The data vault is a highly specialist data storage technique.
- 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.



2. Simple Data Vault

- **Hub Utilities:**
- Hub utilities ensure that the integrity of the data vault's (Time, Person, Object, Location, Event) hubs is 100% correct, to verify that the vault is working as designed.
- **Satellite Utilities:**
- Satellite utilities ensure the integrity of the specific satellite and its associated hub.
- **Link Utilities:**
- Link utilities ensure the integrity of the specific link and its associated hubs.

- **Transform Utilities:**
- Utilities for this superstep contain all the processing chains for building the data warehouse from the results of your practical data science.
- In the Transform superstep, the system builds dimensions and facts to prepare a data warehouse, via a structured data configuration, for the algorithms in data science to use to produce data science discoveries.
- There are two basic transform utilities.
- **Dimensions Utilities:**
- The dimensions use several utilities to ensure the integrity of the dimension structure.
- Concepts such as conformed dimension, degenerate dimension, role-playing dimension, mini-dimension, outrigger dimension, slowly changing dimension, late-arriving dimension, and dimension types (0, 1, 2, 3).
- **Facts Utilities:**
- These consist of a number of utilities that ensure the integrity of the dimensions structure and the facts.
- There are various statistical and data science algorithms that can be applied to the facts that will result in additional utilities.

- **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.
- **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.
- **Maintenance Utilities:**
- The data science solutions you are building are a standard data system and, consequently, require maintenance utilities, as with any other system.
- Data engineers and data scientists must work together to ensure that the ecosystem works at its most efficient level at all times.
- **Backup and Restore Utilities:**
- These perform different types of database backups and restores for the solution.
- They are standard for any computer system.
- **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.
- **1)Scheduling Utilities:**
- 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.
- **2)Monitoring Utilities**
- The monitoring utilities ensure that the complete system is working as expected.

Engineering a Practical Utility Layer:

- The utility layer holds all the utilities you share across the data science environment.
- 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.
- you 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.
- you 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.