Data Science Team
Quality Analytics
R&D Quality Community
Rrober14@its.jnj.com

Repository and Coding – Data Theory

1-Governance

1.1 Situation

Data Science, Advanced Analytics, and Data Management Automation teams are still operating as industries that are incredibly young, and accelerating in methods, processes, and technologies very quickly. The definition of "reading a big data set fast" quantitatively is continuing improving. The continual technological progression of what is feasible drives the business definition of staying competitive and serving the client-customer. Coming back to the technical lens, this means not just developing innovative code, and deploying developed code, but maintaining code in production and/or deployed environments. While the data science and advanced analytics space have the benefit of leveraging experiences, examples and instruction provided from software engineering bodies of knowledge, there is still the need to acknowledge that writing inference code and writing software (even analytical software) are not synonymous.

Even with all the decades of experience doing software development, as much as 90 percent of the development effort on a typical software system comes after its initial release, with two-thirds being typical (Pigoski, 1997). Machine learning, generative artificial intelligence, black box models are all tools that arrived after the above cited research. Safe estimates place the effort at much higher (and thus extreme) values of resourcing required.

Data Science and Advanced Analytics not having ubiquitous language, terminology or generally accepted best practices results in a much wider variation in work and outcomes compared to more mature professional fields like statistics, engineering, or medicine. While this is a natural reality of any industry in its early times, it is still one that we must navigate, mitigate and given the uncertainty, do risk assessment and planning for. Technical debt for maintenance and improvement increases exponentially as analytical models and the environments leveraging them become more complex.

Fred Brooks said that software development is like farming, hunting werewolves, or drowning with dinosaurs in a tar pit. There is an opportunity to increase crop yields, set traps that don't require our presence, and identify tar before we are standing in it.

1.2 Governance Objective

Provide work instructions to data engineers, analytical engineers, and data scientists working with advanced analytics teams in support of development activities. Provide product standards to DevOps and MLOPs engineers and developers in support of staging and production maintenance and improvement activities.

Provide interoperable work instructions, product standards and documentation guidance for advanced analytics team with respect to the scope of development, staging and productized code and code management.

1.3 Governance Documentation

Documentation requirements to support code base are the following:

- Link to the products documentation on confluence, which includes
    - Contact information for current technical individual/team/organization maintaining code
    - Catalogue information for current business owners of code in production/deployment
    - Information for location of code base
- Information/Instructions attaining access to code base /confluence
- Information regarding Data Theory
    - Training videos
    - Repository/coding guidance documentation
    - MLOPS documentation

Data Science Team
Quality Analytics
R&D Quality Community
Rrober14@its.jnj.com

1.4 Dynamic VS. Static Information:

Dynamic information are data that are subject to updates, for example, contact information of code owners or business stake holders, or code or model version. To avoid constant need to update the info in repositories, we save these details in products info page (on confluence).

On the contrary, address to the product page or training information do not need frequent updates (their content get updates but not the pointer address to them). These details are example of static information and exist in the governance section of the repository.

1.5 Governance Assumptions & Constraints

There is an opportunity in providing techniques that help individuals constructing data science and advanced analytics code find answers. This is the very definition of a heuristic. Now heuristics tell you how to look, not what to find. You will find that most of the explanations and definitions tell you how to look for something, how to decide, or "find" out what you and your adjacent peers are doing. The utility is a streamlined language purposed to support Data Science initiatives. This work is not designed to challenge the wider-scoped and deeper body of knowledge from software development.

1.6 Governance Code Defined

1.6.1 Governance Explained

Governance is the first data process activity and can be defined as any one of the following:

- Templates, frameworks, containers or supporting technology required by business. An example might be the requirement of a data version control solution like DeltaLake or .DVC
- Coding/logic operating on the code base in support of business implementation requirements. An example might be logic that that writes documentation within source code.
- Coding/logic running conformance or compliance checks on a code base. An example might be code ensuring that a code base is formatted a specific way PEP8, or meets certain security, privacy and legal requirements.

1.6.2 Assumptions & Constraints

Documentation assumes individual has taken data theory courses or reviewed documentation for digital twin design architecture.

This document is designed both as an independently modular tool for work instruction guidance and by default a supporting tool for product standards.

This document is designed to be implicitly interoperable with the most prevalent MLOPs and DevOPs methodologies and Ways of Working (WOWs) and is explicitly designed to be interoperable with the Data Theory – MLOPS Documentation Template

1.6.3 Unified DS Repository Framework Explained

Naming convention for **RDQ QANA Data Science Team** is a 3-section nomenclature. To align with SCM Automation requirements, there can be no usage of special characters. This includes periods and hyphens. All characters are lowercase. The name can be no longer than 22 characters. Again, while it is possible to create names using hyphens and periods, this naming convention is not compatible with the automatic generation of storage with matching naming.

design-team-name   business-unit [or] project-name   .data-activity-name

1.Design Business Unit Name   2a. Client Business Unit Name   2b. Client Project Name   3. Analytical Design Activity Name

Data Science Team
Quality Analytics
R&D Quality Community
Rrober14@its.jnj.com

Examples:

rdqqds mdsafetycda

rdqqds supplytrainingdataedaimple

rdqqds qualitycomplianceteamdsreport

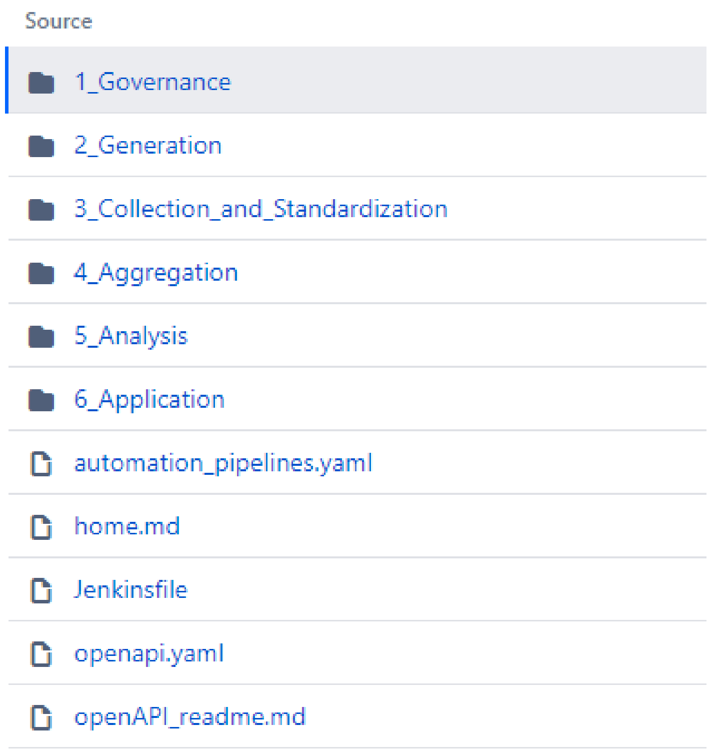
rdqqds enterprisetoweranalyticalsuite

rdqqdsimpctestmitgrec

rqdqdstrndnmlyasocrulz

Multiword components, like a multiword business unit name, or a multiword client name or project name utilizes hyphens.

Repositories are structured into six subfolders representing the 6 data activities; 1-Governance, 2-Generation, 3-Collection_Standardization, 4-Aggregation (enrichment), 5-Analysis, and 6-Application. Looking like this:


Source

1_Governance
2_Generation
3_Collection_and_Standardization
4_Aggregation
5_Analysis
6_Application
automation_pipelines.yaml
home.md
Jenkinsfile
openapi.yaml
openAPI_readme.md

There should be nothing outside of the folder structure unless outside dependency requirements justify it. An example would be if adjacent tools do not allow that orchestration (application) instructions inside sub-directories, requires certain .app, .src and .yaml files be at the main level inside the code repository.

1.6.4 Persisted Data-Object Explained

There are three types of persisted data-objects; raw, working and master. Raw data-objects originate from source or staging locations. A source location is one where you as the coder have control for when you access the location, and you as the coder have control as to how often (weekly, daily, hourly). A staging location is one where the access to the data is outside of your control. An example is a storage site where a human is tasked with uploading a document into the storage site. Effectively, whether there is data to access is decided by an additional dependency. Naturally a repository can have numerous raw data inputs of both types (source and stage: see 2-Generation for source and stage definitions). A repository with two sources of raw data might look like this:

Acrobat Document Acrobat Document

<insert image from Summit Data Set showing two raw file>

Any change by the code base results in the file no longer being "raw". The file can keep the main naming convention, but now transitions into a "working" data object. Let the business case decide how often a new structure should be persisted. It is quite reasonable that extended, complex logic may result in numerous working files that, for unit testing purposes all need to be persisted.

The final file for the section of code is called "master". Now master data object naming can be leveraged holographically. This means, you can expect to see a 2.Generation.master-dataframe object if a sole source/stage, is ingested. It then makes sense that this generation stage data is ingested as an input into 3-Collection_standardization code, resulting in one or more data objects that have been refactored and normalized. These data objects in normal operations be expected to be ingested in one or more places across 4-aggregation, 5-analysis and even 6-analysis (example stop word list, or list of ingested users and their roles). All of these at the code sub-directory level for the 6 data process-activities could be called masters, with a final-ultimate outcome dataset consisting of content and information across the repository being a "repository/project.name-master" data object.

1.6.5 Data Provenance Code Explained

Data provenance code is code that ensures instructions from the business are in fact being followed. This covers data integrity code, data quality code and impact analysis code. All of these examples are related to governance. This code is referenced here in this section of the documentation but is directly detailed inside the 6-application sub-directory for unit testing.

1.7 Governance Code and Scripts

1.7.1 Orchestrator

An orchestrator code module or script is one that performs two or more separately distinguishable tasks within this document. This can be seen as a data_cleaner_orchestrator that performs a series of refactor and standardization operations. This could also be seen as a summary-stats_generator_orchestrator that appends to a data-object the mean, median and mode, with each of the three being separate functions within themselves.

Orchestrators can exist at various levels. It is possible (and expected) to see orchestrators like the

summary-stats_generator_orchestrator, which would like be code that physically resides in the

3-collection_standardization repository sub-directory, and then see a 4-aggregation-generator-orchestrator that runs both summary-stats_generator_orchestrator and Imputation-generator_orchestrator. Note: this is separate from where code can be called from. *The focus is where the code "lives", not where code is called from*. Continuing this example it would be expected that an orchestrator for all the code that does data enrichment in a deployment setting would be called from the 6-application sub-directory.

1.7.2 "via methodology"

Via-methodology code and scripts cover situations where it is generally known and accepted that there is a variation in ways to accomplish a task. This about supervised clustering analysis. There is a short, yet widely known list of ways that the measurements for deciding clusters assignment can be attained (radian, Euclidean, Manhattan to name a few). There will likely be cases where communicating explicitly the approach or method makes for cleaner communication across designers and design teams. To say "I am standardizing the time" can mean a near infinite permutation of operations you performed. Did you standardize by Unix time? Using Greenwich Mean Time?

Coordinated Universal Time? By what method did you utilize to standard the time. Standardizing time is a topic or concept, while UTC versus Unix time is your method.

2-Generation

2.1 Generation Code Defined

2.1.1 Generation Explained

Generation is the second process activity and can be defined as the origination of data within a defined scope of operation. Imagine you have cloud storage that is empty. If you:

- upload a file that you have had for 3 years on your host workstation to this currently empty cloud storage.
- run a query from a database and persist the results in your cloud storage
- run a query script on an OLAP database and persist the results in a working directory on your host machine
- perform a coding operation that creates new output data and save this in a subdirectory folder of your code

These are all generation activities. The definition of generation is with respect to the owner-creators of the body of code logic. Whether the owner of the code personally created or acquired from an external source it is generation. Also, whether the external source is an on demand asset like a database where the owner creator can choose to engage the external source every minute or every hour, or a static source like a text file that is provided by an adjacent business department via email “near weekly”, these are all generation activities from the perspective of the owner-creator of the code base, and the users of the code base when the code base is operating.

2.1.2 Assumptions and Constraints

Documentation assumes individual has taken data theory courses or reviewed documentation for digital twin design architecture.

2.2 Code and Scripts

2.2.1 ReadMe

Minimum requirements are:

- Link to catalogue or metadata about name of files
- Link to catalogue or metadata about the location of files
- Link to catalogue or metadata about business group name, business point of contact
- Link to catalogue or metrics including but not limited to time coverage, constraints for use or applicability, known conflicts about data that can pulled from different points of access resulting in variation of data

2.2.2 Read Functions/scripts

Read functions and scripts ingest data. Different libraries and functions under the hood may perform sub-tasks like locating, and confirming, but this is not meant challenge more holistic coding methodologies. The lens of view here is within the scope of the data activity of generation. Either a source or staging area from data is having data retrieved by the acting script or function. Again, different languages may use terms like ‘retrieve’ or ‘create’ or ‘load’, but from a data theory lens all of these are synonymous with ‘read’.

2.2.3 Write Functions/Scripts

Write functions and scripts persist data. This includes copies that are indistinguishable from the original.

2.2.4 Connector Functions/Scripts

From a data science and data theory lens, a connector is not separable from a read, write or identify function. As a standalone activity this is task that a database administration team would be performing possibly in support of data observability activities, a task that a data science team would not ideally be performing. Data Analysis and Data Science "connect to read" or "connect to write", never "connect to ensure connection is available/accessible".

Identifier Functions/Scripts

Identifiers are functions are scripts that check that an object meets certain data integrity requirements. This can include name-matching, type confirmation, or even metrics within columns and rows (missingness). These functions do not act on the information acquired but rather are tasked with observing and orientation diagnostics. Identifiers should be evaluated as possible unit-testing candidates.

3-Collection & Standardization

3.1 Collection and Standardization Code Defined

3.1.1 Collection and Standardization Explained

Collection and Standardization is third data process activity and can be defined as any one of the following:

- Joining data from two or more data objects by either column or row on a unifying key or keys.
- Joining data from two or more data objects by harmonizing feature fields with respect to granularity/measurement
- Altering dimensions of a data object column-wise or row-wise
- Sorting observations

3.1.2 Assumptions and Constraints

Documentation assumes individual has taken data theory courses or reviewed documentation for digital twin design architecture.

3.2 Code and Scripts

3.2.1 ReadMe

Minimum requirements are:

- Link to catalogue or metadata about function names and types of transformations performed

3.2.2 Refactor

Refactor occurs where you have a harmonized (homogeneous) representation that is transformed to a different representation with respect to granularity.

- Transforming the weight of patients from a dataset from kilograms to pounds
- Transforming the date-time stamp from GMT to UTC
- Transforming patient's weight from numerous data objects where some observations are recorded in pounds and other observations are recorded in kilograms, all into ounces
- Transforming the name of a column header to a more generalized name
- Transforming the name of a column header to be all uppercase or lowercase
- Transforming text to be all upper case or lower
- Stripping specific characters or words from text

3.2.3 Standardize

Standardization occurs where you have two or more representations (heterogeneous) of a single data feature or observation from one or more data objects. The activity performed is a transformation to a single harmonized (homogenous) representation. Examples would include:

- Harmonizing the weight of patients from a single object where some observations are recorded in pounds and other observations are recorded in kilograms, all into kilograms

- Harmonizing the name of a column header from two or data objects into a unified column header name.
- Harmonizing numerical fields expressed as two or more decimal of precision to a unified number of decimal points of precision

3.2.4 Filter

Its first important to repeat that this is not to compete with software engineering best practices. This lens of conversation is for analytics, advanced analytics and data management. The definition of a filter is when your focus attention for the code is what you are removing. Said another way, you filter *out.* Filters are to ensure something is NOT being utilized.

3.2.5 Selector

Its first important to repeat that this is not to compete with software engineering best practices. This lens of conversation is for analytics, advanced analytics and data management. The definition of a filter is when your focus attention for the code is what you are keeping. Said another way, you select *in.* Selectors are used to ensure that something IS being utilized.

4-Aggregation

4.1 Aggregation Code Defined

4.1.1 Aggregation Explained

Aggregation is fourth data process activity and can be defined as enriching a data object through the creation of additional information derived from apply logic operations on the existing data object. This can mean populating existing column or row fields, and/or creating new column or row fields. The source data leveraged must derive solely from the data object being enriched. If a data object is being attained additional inputs, fields, columns or rows from a second or more data object, this would be a *collection and standardization* activity (see section 3.1.1).

Aggregation activities can technically be the same as Analysis activities. The business difference is whether the functioning code ***delivers*** the primary business deliverable or ***supports*** the delivery of the primary deliverable. Imagine a supply shop requesting an diagnostic analytic that tells them if their supplies are currently within acceptable thresholds to support business needs for the quarter. Their primary interests is a representation of “yes we are within acceptable thresholds or no we are not within acceptable thresholds.” (NOTE: for this simplified exampled the thresholds are already provided so there is no confidence interval provided and therefore not a predictive)

In this above example, we could easily see doing several steps:

1) Take current supply value and add expected supply additions for the quarter.
2) Take current demand and add expected demand additions for the quarter
3) Subtract ***step 2*** value from ***step 1*** value
4) Conduct Binary check if ***step 3*** value is within provided thresholds
5) Persist answer from ***step 4*** for later leveraging by an application-6 activity

Steps 2 and 3 are clearly predictive analytical activities but are not the primary information of interests for the business but rather supporting items necessary as stepping stones to get the primary information of interests. These both would ideally be persisted to a data object and thus be enrichments, or aggregations of a data object. This again applies for the raw numerical value generated in ***step 3***. Pay special attention to the fact that the *check itself in **step 4*** again is not the primary item of interest. The business would not be happy to know solely that the check was performed, it is the answer delivered in step 5 that is of interest. The step 5 item is the analytical. So one way this could be designed is-

- a model-function that triggers orchestration for the activities listed above
- aggregation functions for generating the supply, demand and binary threshold checks
- within the aggregation-4 functions for the supply and demand, sub-models located in analysis-5 would be called (for sub-models see section 5.2.2)

This may seem complex when you only have a single way to perform your subtasks but imagine if you had several methods to forecast the supply and the demand for ***step 1*** and ***step 2***. Imagine if there were numerous threshold functions provided in ***step 4***. These would represent separate ways to attain the information needed and naturally would the change the underlying data utilized to find the answer "Yes or No are we within threshold constraints", but the actual steps to get there, the analytical logic is still the same. Only the methods leveraged to design the dataset (and thus the dataset) inputted into your primary analytic are changing. Remember, *The focus is where the code "lives", not where code is called from*(See section 1.6.1).

Other examples falls into Aggregation:

- Ranking

4.1.2 Assumptions & Constraints

Documentation assumes individual has taken data theory courses or reviewed documentation for digital twin design architecture.

4.2 Code and Scripts

4.2.1 ReadMe

4.2.2 Summary Statistics

4.2.3 Data Scaffolding

4.2.3.1 Rank

4.2.4 Imputation

4.2.5 Cluster Labeling

4.2.6 Generators

5-Analysis

5.1 Analysis Code Defined

5.1.1 Analysis Explained

Analysis is the fifth data process activity and can be defined. <more to come inserted>

5.1.2 Assumptions and Constraints

5.2 Code and Scripts

5.2.1 ReadMe

5.2.2 Model/Sub-Models

5.2.3 Ensemble Models

5.2.4 Unified Model Output Structure

6-Application

6.1 Application Code Defined

6.1.1 Application Explained

Application is the sixth data process activity and can be defined as the client interaction and application layer. Your highest levels are static and dynamic, where both center around the idea of how do client-users interact with the code logic. Client roles and reasons of interest include, but are not limited to:

- **Business customers** to seeking product and service supporting tools (internal efficiency tools, products and services)
- **Managers of primary business customers** seeking to monitor usage of supporting tools, products and services.

- **Technical maintainers of analytical code** (data scientists) seeking to maintain and improve models and model logic within both internal efficiency and client facing tools.
- **Technical maintainers of productization code, orchestration and delivery** (DevOps, MLOps) seeking to maintain and improve ETL logic.
- **Technical managers** seeking to monitor uptime and consistency of deliver and performance of client-facing tools, products and services provided to adjacent business teams.
- **Business Performance Managers** seeking to audit business impact of individuals leveraging internal efficiency tools, products and services from adjacent technical teams.

6.1.2 Assumptions and Constraints

6.2 Code and Scripts

6.2.1 ReadMe

6.2.2 www folder

6.2.3 application-orchestrators

6.2.4 Unit Testing

6.2.4.1 Governance

6.2.4.2 Generation

6.2.4.3 Collection & Standardization

6.2.4.4 Aggregation

6.2.4.5 Analysis

6.2.4.5.1 High Performance Computing Thresholds

6.2.4.6 Application

6.2.4.6.1 Big Data Thresholds

6.2.5 Operational Health

Borrowing from the Microsoft AZ-900 course content, operational health at a highest level can be thought of with respect to the:

- (High) Availability
- Scalability
- Predictability
- Reliability
- Security
- Governance
- Manageability

This covers the application resource requirements both from a machine aspect but also a human resource demand aspect from both users and maintainers, improvers and providers. An application could be argued to be generalized by the gain value it provides users, maintainers, improvers and providers with respect to the cost impact it effects on users, maintainers, improvers and providers.

6.2.5.1 Big Data Diagnostics

6.2.5.2 Data Integrity Diagnostics

6.2.5.3 High Performance Computing Diagnostics

6.2.5.4 Application Diagnostics

6.2.6 Performance Impact

6.2.6.1 User Feedback

6.2.6.2 Unified Behaviors Standard Metrics

6.2.6.3 Data Quality 2.0