Data Science End-to-End Project

(Part 1: The Data Management Backbone)

1. Motivation

Operationalizing (i.e., systematizing) and automating Data Science is hard. And this is the only reason why most organizations find it difficult to incorporate in their operations. Yet, there are multiple evidences that doing so bring competitive advantage to organizations.

One key aspect that distinguishes Data Science from Statistics or Mathematical Modeling is that Data Scientists build data-intensive software systems and as such, one relevant aspect of their day-by-day is to act as software developers: designing (architectural aspects), coding, testing and evaluating. However, software engineering tasks conducted by Data Scientists differ significantly from those of traditional Computer Science because data is a key aspect of these systems and, as we will explore during this project, it affects quite a few fundamental aspects.

For this project, we want you to get familiar with the main difficulties you will face when participating in Data Science projects, get familiar with DataOps, develop good habits to create data-intensive systems and also assimilate good practices that will facilitate your day-by-day.

2. Project Objectives

We assume a standard Data Science scenario where an organization wants to operationalize and automate as much as possible their day-by-day data analysis.

The main objective of this project is designing a simplified end-to-end system architecture that you should take as baseline when diving into Data Science projects. Of course, being ADSDB a Q1 course, we will not drill down into all the complexities of such an architecture, but present a baseline that, (i) firstly, will provide you with a description of the main modules and concerns a project should cover and, (ii) secondly, promote good practices to enable the evolution of such systems (in Data Science, it is important to start simple and small and grow from there).

More specifically, in this project you must implement the data management backbone (1st part of the project) and, at least, one data analysis backbone for a given problem (2nd part of the project), and put them on operations. We will avoid the data governance layer. The main tasks you must conduct in this project are as follows:

- Define the project context in a given (imaginary) organization,
- Implement the organization data management backbone,
- Implement, at least, one data analysis backbone,
- Explain your decisions and guarantee *reproducibility*.

Importantly, the project must be conducted in groups of two people. **Register your group in Learn-SQL before the deadline**. If you do not have a group by the deadline, you will be automatically assigned to a group. Each group will have a supervisor (lecturer) assigned that will help you throughout this project. Contact your supervisor on-demand as much as you need. We are here to help!

In the next sections we will explain what we expect for the subobjectives related to the first part of the project (the data management aspects). Importantly, to fully make sense of the next sections it is important to understand the project presentation slides (otherwise, please, talk to your lecturers first).

2.1 Define the Project Context

Choose a domain (e.g., retail, videogames, energy, fraud detection, etc.) and an analysis topic you may be interested in.

Since you most probably do not have access to data sources, you may want to explore popular repositories, find interesting datasets you may want to work with and, eventually, pick the domain and decide the analytical task to conduct in the project.

In Learn-SQL you will find some links to interesting data sources that may help you to come up with interesting ideas.

Constraint 1. Your project must consider, at least, **two different datasets from different data sources** that you will have to integrate during the project. You must specify which are the data sources the datasets come from.

2.2 The Data Management Backbone

The **landing zone** is typically implemented by means of a distributed file system. We will avoid distributed systems so far, to keep the complexity of the project low, therefore, a regular file system is acceptable (even if it would generate problems easily, as you will experience).

You are recommended to store your landing zone at your UPC Drive but this decision is up to you as long as it meets the requirements set by the project. If you go for UPC Drive, you may create a folder in your drive, called landing, where all files related to this zone are stored.

The landing zone has two parts:

- The temporal landing can be as simple as a cp into a folder (e.g., /landing/temporal/)
- The persistent landing requires some structure. In a given folder (e.g., /landing/persistent/) generate some structure with subfolders that will contain the persisted files. Both, for the folders and files you need to use some naming convention that will allow you to later automatically explore it. The most usual metadata used at this stage is: *dataSourceName + timestamp* (at ingestion time). You may add other metadata for your project if that helps automation.

The **formatted zone**, in this project, will choose the relational model as canonical data model. In reality, there is a wealth of data models that you may use here, but you will find them out bit by bit later in the master. So, for now, let us implement it in a relational database. You may want to use PostgreSQL. However, there are some other interesting options that you are strongly advised to use, such as CockroachDB or DuckDB. Both are full-fledged relational databases implementing most of the NoSQL advances (you will study them later in the master as well, but the interface is purely relational) and also incorporating some nice features specifically thought for Data Science (e.g., connection to Pandas dataframes). We specially recommend taking a look at DuckDB.

You are suggested to set up these databases locally in your computers. You will have full control to configure and tune them up. This will prevent the supervisor to access them, but we will present a solution to this problem later.

As thumb rule, in the formatted zone, there should be one table per data source version (i.e., data source and timestamp) because along time the source schema might vary and therefore, it is safer to implement a table per version (instead of per dataset).

The **trusted zone** is meant for conducting data quality processes. The core element is a database, where there should be a single table per dataset (thus, you need to handle the versions and decide how to homogenize their schema and data). Typically, this is done in two different steps:

- A script extracting data from the formatted zone and loading it in the right definitive trusted table.
- A set of processes that run over the trusted zone and control data quality. Realize it is fine if these processes generate new tables. These processes usually conduct:
    - **Data profiling**: for each table, a profile of each variable is generated to gain insights on the quality of the data. This can be done by means of descriptive multivariate analysis methods to explore the available data, assess its quality and gain knowledge about the value of the analysis that can be extracted from there.
    - **Deduplication**: when generating a single table from different datasets, it is usual to generate duplicates.
    - **Outlier detection**: data might contain errors, noise or outliers. These are typically handled here.
    - **Other data quality tasks**: realize that the data quality processes run here differ from those in the data analysis backbone in the sense that they should be common for any data analysis pipeline, whereas those run in the analysis backbone are specific for the data analysis pipeline at hand.

  There is nowadays a wealth of tools and libraries supporting data quality. Libraries such as PyDeequ or tools like OpenRefine could be used here. Otherwise, you can implement your own quality rules by coding them.

The **exploitation zone** should be a different database containing the relevant entities in the domain. Therefore, it must contain a set of tables that separate the different concepts of interest for later analysis. Recall that the data management backbone should be independent of any analysis that can be performed afterwards. Thus, its design must follow good practices in data management (minimize redundancy, etc). Later, in the analysis backbone, data will be re-organized in an appropriate form according to the specific analysis tasks to be performed. Importantly, you must explain and justify the design of the exploitation zone in the document.

The scripts you need to create to extract the data from the trusted zone and insert it in here must conduct the following relevant steps:

- **Data integration**. A table in the exploitation zone may be fed with data from more than one table in the trusted zone. Thus, data must be first integrated.
- **Data quality processes for integration**. To integrate data, you may need to transform and homogenize certain values.

In essence, the data flows between the trusted and exploitation zone follow the idea of ETLs you will get introduced to in Data Warehousing.

Between each zone described above (landing, formatted, trusted, and exploitation) you must create scripts running the transformations, and also, within a zone to run data quality processes, etc. You are highly advised to use notebooks to write these scripts. One interesting tool for you is Google Colab. Your UPC Google account gives you access to it and it allows you to directly connect to your Drive in case you implemented there your landing zone.

Importantly, the exploitation zone can store relational table(s) that later, in the data analysis backbone you may want to transform into tensors or dataframes.

Constraint 2. Organize your notebooks per zones. For example, if you use Google Drive, create a folder named notebooks and inside group your notebooks in folders per zone: landing zone, formatted zone, trusted zone and exploitation zone.

Constraint 3. Use a meaningful name convention for the notebooks and add text and charts inside the notebook to explain your processes. You are advised to use Python as prototying language.

Constraint 4. In the trusted and exploitation zones, each data quality process must be separated in its own notebook (unless you use an external tool, then, be sure to talk to your supervisor to agree on how to open this information so that it can be evaluated).

Constraint 5. Include a notebook per database (one per each zone, from formatted to exploitation) that explains its structure (schema) and profiles the data in that database. This way, the supervisor does not need access to the databases.

2.3 Operations

Realize all the code and repositories developed in the previous items correspond to the development phase of the project. Even more specifically, to the prototyping phase of development (largely based on trial and error). Now, we must generate the operations environment.

At this point, it is important you stop and realize about the project development stage. Each notebook is an isolated piece of code that executes on demand. Notebooks communicate between each other through the data repositories created (i.e., the databases). Therefore, you have pieces that communicate asynchronously via databases. This is ideal for prototyping and doing trial and error. Yet, according to DevOps, this is not acceptable, and we need a running environment based on continuous integration meeting the DevOps requirements. Thus, we should, at least:

- Generate orchestrated executable code from the notebooks,
- Place it into a continuous integration environment (e.g., Gitlab).

Given the time constraint of this project, we will go for some basic integration environment. Specifically, we will focus on generating the orchestrated executable code. In real projects, a continuous integration environment is key.

For this matter, an easy way to obtain Python executable code from the notebooks you created is by using the export functionality of Google Colab (other notebooks have similar operations).

This will generate a .py per notebook. That is why it is important to properly organize your notebooks.

Now, use a tool to version and organize your code. For example, Github. You will need to develop some additional code to orchestrate the execution of all the independent pieces of code. Including a piece of code that reads some new data and executes the created model and makes a prediction / classification, etc. You may want to use Gitlab and runners for that (this is optional).

Generate the executable codes and create a simple user interface to run it in order to simulate an operations environment. There is no need to overcomplicate the interface, we will not assess its quality.

Three aspects that we will positively consider in the operations layer, although they are not mandatory, are:

- Enrich the extracted code from the notebooks with error handling (something typically not considered in development), such as basic unit testing.
- A simple code monitoring the execution of the Data Management Backbone in runtime (processes performance, disk and memory usage, etc.).
- Couple a quality control tool to control the quality of your code (e.g., SonarQube) and report about it.

Constraint 6. Create a code repository orchestrating all the code from your notebooks able to generate your operations environment.

Constraint 7. Your operations layer must be able to execute and ingest a new data source and propagate it throughout all layers, including the execution of the model in runtime upon new arrival of data.

3. Deliverables

The outcome of this first part of the project is twofold:

- An explanatory document (max. 10 pages long),
- The project repository

The document must contain the following sections:

1. Your project supervisor should have access to your **development** (e.g., Drive) and **operations** (e.g., Github) **platforms** where you developed your project. Instructions how to access to these platforms must be in a mandatory section to be included in the document (first section of the document, in a separated page, without numeration) providing the link to these platforms. This section does not count for the maximum number of pages to be used in the document.
2. **Context**. The domain chosen and a description of the original data sources and the variables they provide. Define here the analytical question you want to answer.
3. **The data management backbone**. Using the figure from the slides, instantiate it (i.e., add on top) the tools you used for each element from the architecture, as well as the datasets and transformations of your particular pipeline.

4. **Operations**. Add an explanation of how you have organized your operations for the data management backbone. Specifically, pay special attention to justify the new code added to orchestrate the pieces generated during development.

For items 3 and 4, it is important you discuss the pros and cons of your chosen solution. We will positively assess if you identify the limitations of your current approach (do not worry to state them, it is a baseline, simple solution, it will have plenty). Do not repeat here the information in the notebooks / code. These should be an overall description to understand your development and operations platform and facilitate the supervisor to explore it.

4. Evaluation

The project will be evaluated according to the following criteria:

Dynamicity. How easy is to add a new variable into an existing source? And add a new source? How easy is to change the transformations executed between two zones?

Reusability. Code and data are not duplicated and they are well organized.

Openness. Your system could be easily extended and evolved with new / advanced aspects.

Single source of truth. Analysts can access a single source of truth from where to start the analysis.

Reproducibility. The executables in operations can be easily executed and are properly documented to do so. Similarly, for the notebooks in development.

Soundness. The choice of tools / solutions is adequate for each zone.

Completeness. All constraints in the statement are met.

Rigorous thinking. In the document, there is a detailed discussion about pros and cons of each solution and the students identify room for improvement for advanced solutions.