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 it in their operations. Yet, there is multiple evidence that doing so brings 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.

Disclaimer: since this course is during the first semester of the master, its objective is not to consider all possible advanced techniques and tools throughout the Data Science pipelines (to which you will be introduced throughout the master as you progress) but to identify the main problems behind the operationalization of Data Science projects and also endorse you with good practices about data management for Data Science. We will encourage you to use new solutions and tools but, for example, distributed solutions (databases and infrastructure) are topics covered in the second semester and therefore not tackled in this course.

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 a 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 into 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.

Then, you have to select the **data sources**, which are the locations (repositories, APIs, databases, etc.) from which you can extract data. Since you most probably do not have access to data sources, you may want to explore popular repositories to find interesting data 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. From these data sources you will obtain **datasets**. By dataset we refer to a *type* of data (i.e. a set of attributes/columns) to be found in the source (the source might offer several datasets or only one). Moreover, we can have different **versions** of the same dataset (i.e. same attributes/columns but representing data of different timeframes).

For instance, the U.S. government's open data portal (data.gov) can be a data source for your project. There, you can find a lot of datasets, with information about a wide variety of topics. We decide to use a dataset, *us_states_dem*, that presents demographic information for each state, updated on the first day of each month. Hence, the data of January 1st 2024 (*us_states_dem_01_2024*) and the data of February 1st 2024 (*us_states_dem_02_2024*) are two versions of the same dataset.

Constraint 1. Your project must consider, at least, **two different data sources containing, each, two different versions of the same dataset** (e.g., following the previous example, *us_states_dem_01_2024* and *us_states_dem_02_2024* would be two versions of the same dataset. Now, we would need to repeat this for another source). You will have to integrate this data during the project. The datasets must meet the following quality criteria:

- They refer to the same domain.
- The datasets from the different sources are complementary and provide different data variables for the same object of study that, once integrated, provide added value to the analyst (e.g., we could match *us_states_dem* with a dataset from another source that contains economic information of each state, *us_states_ec*)
- They can be merged together, which implies the existence of a common attribute you can join through and augment the instances of one dataset with the instances of the other (e.g., we can join *us_states_dem* and *us_states_ec* via the column that identifies the states).
- The datasets provide response variables (i.e., variables of interest for the analysis, typically continuous variables) and predictor variables (i.e., variables that can be measured and predict the response variable) and therefore enables meaningful

analysis (e.g., by joining g *us_states_dem* and *us_states_ec* we can analyze how a given economical metric is altered by the characteristics of the population of each state).
- Ideally, the datasets should provide meaningful data variables enabling the possibility to derive interesting business KPIs (e.g., income inequality, employment to population ratio, etc.).

Note: if the dataset that you are using does not provide different versions of the data, you can generate the versions yourself. For instance, you can partition a dataset into several chunks and assume that the data of the different partitions is obtained at different days. This is not an issue, as our primary focus in this first part of the project is for you to learn how to handle different versions and integrate them. Just make sure that the separation makes sense.

During the first step of the project, identifying adequate datasets is crucial. Be sure to conduct descriptive analysis to guarantee the above quality criteria are met **before** starting the project.

2.2 The Data Management Backbone

The data management backbone (and, generally, the entire pipeline) is separated into zones. Each zone is responsible for applying certain transformations to the data, ensuring that the data is progressively refined and prepared for specific analytical tasks. Each zone is independent of each other, which implies that changes or issues in one zone do not directly affect the functioning of others. This separation allows for greater flexibility, as different teams or processes can work on individual zones without disrupting the overall pipeline. Moreover, this modular approach enables better fault tolerance, as errors can be isolated and resolved within a specific zone, minimizing the risk of data corruption or delays propagating throughout the system. It also facilitates incremental upgrades and testing, where new tools or methods can be deployed within a zone without impacting downstream or upstream processes.

You are recommended to implement the pipeline at your UPC Drive, creating different folders for each zone and employing Google Colab notebooks to define the code to apply the transformations between the zones. Nonetheless, this decision is up to you as long as it meets the requirements set by the project. If you plan on using specific environments or tools be sure that the supervisor can access them for the correction. We describe possible implementations further down.

The **landing zone** serves as the entrypoint for data into the system. Here, we are going to work with the raw data, and just apply some structuring. This zone is typically implemented by means of a distributed file system. We will avoid distributed systems so far, since this is a topic covered during the second semester of the master, therefore, a centralized file system is acceptable (even if it would generate problems easily, as you will experience). 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/). Here, all the raw files coming from the data sources will be dumped, waiting to be further processed and, then, deleted from the system.
- The **persistent landing** gets the data from the temporal landing and applies some organization. In a given folder, (e.g., /landing/persistent/) generate a structure with subfolders (e.g., per source) 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 at this stage is: *dataSourceName + timestamp* (at ingestion time). You may add other metadata for your project if that helps automation.

The **formatted zone** unifies the formats of all the files of the system, preventing us from having to manage data that is structured in different manners. In this project, we will choose the relational model as the 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.

Traditional relational databases are not the best option in a Data Science project, as they are meant for addressing transactional loads. Our recommendation is an analytical database, such as DuckDB. An analytical database allows you to conduct analytical tasks (e.g., a clustering algorithm or typical data preparation tasks) within the database (which provides ad-hoc features to support these tasks). DuckDB is an interesting case of study because despite being a full-fledged relational database it implements most of the NoSQL advances (you will study them later in the master as well), but keeping a purely relational interface while incorporating some nice features specifically thought for Data Science (e.g., connection to Pandas dataframes). Thus, we especially recommend taking a look at DuckDB for this project.

One of the advantages of DuckDB is that it is an embedded database (sqlite and other traditional databases are also embedded). This implies that it runs directly within your application, without the need for a separate server. Hence, a simple way to implement the formatted zone (and the subsequent zones) is to define a folder in your Google Drive and create a DuckDB database there, allowing you to modify the data by connecting to it via the notebooks. This is the recommended way to proceed, as the supervisor will be able to easily access the database.

Nonetheless, you can also set up these databases locally in your computers and connect to them via Google Colab (instructions on how to do so can be found in the LearnSQL platform). This will prevent the supervisor from accessing them, but you can still connect with your notebooks and store them in Google Drive. To account for this, we ask you to perform extensive data profiling and description of the databases in the notebooks. Another option would be to set up a server, define the databases there and connect them via the notebooks, which would also allow the supervisor to access the databases.

As rule of thumb, in the formatted zone, there should be one table per data source version (i.e., a dataset from a given data source and a timestamp) because along time the source schema and its data might vary and, therefore, it is safer to implement a table per version. In Data Science projects it is quite usual to extract data from a data source recurrently. For example, you may extract a dataset from X every Sunday night (via its API). At each dataset you will have *intensional* (i.e., the schema) and *extensional* (i.e., instances) information. Note that the schema may vary between datasets (e.g., when a new variable is added / removed between two API releases) as well as data.

The **trusted zone** is meant for conducting data quality processes. The core element is a database, where there should be a single table per data source (thus, you need to handle potentially several datasets coming from a data source and decide how to homogenize their schema and instances into a single table). 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 (realize the trusted table schema is typically a superset of the datasets homogenized into that table).
- A set of processes that run over the trusted zone and control data quality. These data quality tasks differ from those performed in the data analysis backbone (2nd part of the project). Here, we conduct generic data quality, which implies that modifications applied here should not interfere with any posterior analytical process. Another way to think about this is as fixing errors that the data might have rather than preparing them to train a specific predictive model. 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.
    - **Misspellings**: when a categorical value contains a typo.
    - **Consistent formatting:** for instance, we want all time variables to have the same format and describe the same timeframe (e.g. dd/mm/yyyy vs. mm/dd/yyyy)
    - **Data normalization**: when a tuple contains an array of objects (e.g., a citizen and an array of all the town council services used). Since the array is of different sizes, it typically adds noise to data management and we might be interested in *exploding* the array.

  Despite there are a wealth of tools and libraries supporting data quality, such as *PyDeequ* or tools like *OpenRefine* that could be used here, we want you to learn good practices while dealing with them. Thus, for this project, implement your own quality rules by coding them in this zone without using external off-the-shelf tools. Nonetheless, you can check these tools to see what kind of processes they implement.

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 organization of the exploitation zone highly depends on the data that you have, and there is no clear pattern to do so. This lack of a one-size-fits-all approach emphasizes the importance of adapting the structure based on the specific nature of your data. It is useful to think about it as the *ground truth* of the analytical tasks, that is, the data that will be given to the analysts for them to perform whatever processes they need. To improve the usability of the exploitation zone, try to combine similar entities together and compute relevant information for the analysts. This involves grouping data sets that share common characteristics or will likely be used together in analyses, hence minimizing redundant searches and allowing for a seamless analytical process. Additionally, pre-computing relevant metrics, aggregations, or derived values can save valuable time and reduce computational overhead for analysts.

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**. After integrating data, you may need to transform and homogenize certain values (e.g., two different data sources referring to New York as '*NY*' or '*New York City*'.
- **Definition of KPIs** or other data that the analysts might want to use in the analytical backbone.

In essence, the data flows between the trusted and exploitation zone follow the idea of ETLs you will get introduced to in Data Warehousing. 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. If you are embedding the databases in Google Drive, you can decide to place the databases in these folders or create a different folder structure altogether. Just make sure that everything is well organized.

Constraint 3. Use a meaningful name convention for the notebooks and add text and charts in the notebook to explain your processes. You are advised to use Python as the prototyping language, and do so via notebooks.

Constraint 4. When ingesting data into the landing zone, provide the notebooks used to extract data from the sources (if it is needed, such as connecting to an API). If your data does not have different versions, provide a code to break these files into datasets and simulate an incremental data ingestion. The source files from where you will extract your databases must be provided. We expect at least two datasets versions per data source (and at least two data sources) to practice incremental data ingestion. All of this can be stored in a different folder (e.g. *data ingestion*), with one notebook per data ingestion process.

Constraint 5. In the trusted and exploitation zones, different data quality processes must be separated in different notebooks, to ensure correct software practices such as reusability. Realize there is no rule of thumb to decide if two data quality processes should be separated or together in the same notebook. For this reason, it is important you justify the organization of your quality processes for the sake of the non-functional requirements stated in the evaluation section.

Constraint 6. Include a notebook per database (note that there might be more than one database per zone -justify this decision, though-, from formatted to exploitation) that explains its structure (schema) and **profiles** the data in each database. This is needed even if the supervisor can access your databases.

Constraint 7. Justify the organization of your data in the exploitation zone. The exploitation zone is subject-oriented and it is very dependent on the problem at hand. Follow the good practices discussed in the lectures and justify your decisions.

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. Also, if you have set up the databases and the notebooks in your Google Drive, you will need to download them to define the operations code and then upload that to your versioning tools. This might require some changes in the code, such as the path definition to access the databases.

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

Constraint 9. Your operations layer must be able to execute and ingest a new data source and propagate it throughout all layers.

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 on how to access these platforms must be in a mandatory section to be included in the document (first section of the document, in a separate 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, a justification they meet the quality criteria set for them 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 (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 it to add a new variable into an existing source? And add a new source? How easy is it to change the transformations executed between two zones? Up to which extent these tasks are automatically supported without extra code?

Reusability. Code and data are not duplicated and they are well organized. A third person should be able to understand your code and run it easily.

Openness. Your system could be easily extended and evolved with new / advanced aspects. Remember the project created here is a baseline, in a real environment, this system will be extended and evolved significantly along the years.

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.