

UNIVERSITAT POLITÈCNICA DE CATALUNYA

ALGORITHMS, DATA STRUCTURES AND DATABASES

End-to-End Data Science Project

First Part

Miona Dimic

`miona.dimic@estudiantat.upc.edu`

Míriam Méndez

`miriam.mendez.serrano@estudiantat.upc.edu`



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Facultat d'Informàtica de Barcelona



Curs 2022/2023 Q1

Instructions to access to the platforms

Datasets from corresponding chosen data sources are stored at UPC Drive which is accessible through following link: [ADSDB/Mimi/UPCDrive](#)

The rest is on github in a private repository, located at the following link¹: <https://github.com/mionaD-upc/mimi-adsdb>

To execute the code follow this instructions:

1. Intall the pip requirements: `pip -r requirements.txt`
2. Execute the `__init__.py` script contained in the Operators folder: `python3 __init__.py`

Moreover, you can take a look at all the notebooks, which are executed in the following order:

1. Landing zone/landing_zone.ipynb
2. Formatted zone/formatted_zone.ipynb
3. Trusted zone/trusted_zone.ipynb
 - (a) Trusted zone/household/household_profiling.ipynb
 - (b) Trusted zone/household/household_duplicates.ipynb
 - (c) Trusted zone/household/household_missings.ipynb
 - (d) Trusted zone/household/household_outliers.ipynb
 - (a) Trusted zone/nationalities/nationalities_deduplicates.ipynb
 - (b) Trusted zone/nationalities/nationalities_profiling.ipynb
 - (c) Trusted zone/nationalities/nationalities_missings.ipynb
 - (d) Trusted zone/nationalities/nationalities_outliers.ipynb
4. Exploitation zone/dataQuality_integration.ipynb
5. Exploitation zone/exploitation_zone.ipynb

¹if you do not have access send an email to the owners of this repository.

Contents

| | | |
|----------|---------------------------------|----------|
| 1 | Context | 1 |
| 2 | Data management backbone | 2 |
| 2.1 | Landing zone | 2 |
| 2.2 | Formatted zone | 2 |
| 2.3 | Trusted zone | 5 |
| 2.3.1 | Household | 6 |
| 2.3.2 | Nationalities | 6 |
| 2.4 | Exploitation zone | 7 |
| 3 | Operations | 8 |
| 4 | Conclusions | 8 |

1 Context

As a starting point for this project, two datasets are chosen from different data sources. Both of them rely on the domain of demography in the municipality of Madrid.

Open data sources and selected datasets are available at the following links:

Dataset1 - *Households by Size, Composition of the household, Nationality and Section according to District in Madrid*

Dataset2 - *Residents in the Municipality of Madrid by Nationality*

To simulate the versions, we choose those datasets that correspond to years 2018, 2019 and 2020, which were added in Python with their download url:

```
DATA = {  
    'household2018.xls': 'https://datos.madrid.es/egob/catalogo/300438-9-hogares-tama%C3%B1o.xls',  
    'household2019.xls': 'https://datos.madrid.es/egob/catalogo/300438-10-hogares-tama%C3%B1o.xls',  
    'household2020.xls': 'https://datos.madrid.es/egob/catalogo/300438-11-hogares-tama%C3%B1o.xls',  
    'nationalities2018.xls': 'https://www.madrid.org/iestadis/.../padron/descarga/pc18t18z4_secci.xls',  
    'nationalities2019.xls': 'https://www.madrid.org/iestadis/.../padron/descarga/pc19t19z4_secci.xls',  
    'nationalities2020.xls': 'https://www.madrid.org/iestadis/.../padron/descarga/pc20t20z4_secci.xls',  
}
```

Both have in common that they include the districts and sections of Madrid but with a small difference that the nationalities repository includes the entire autonomous community of Madrid while the household repository only includes the city of Madrid. Based on this, specifically in the exploitation zone, we will only keep the data that belongs to the latter one.

Main idea is to join these two datasets with the aim of extracting useful information about the nationality of the people living in these districts and the composition of their household (if they live alone, there are children, they come with their partner, etc.)

This insight into the distribution by nationality and type of household allows us to define infrastructure and service needs in a specific district of Madrid. As an example, we could be able to determine which district would need more kindergartens, international schools, clubs etc.

2 Data management backbone

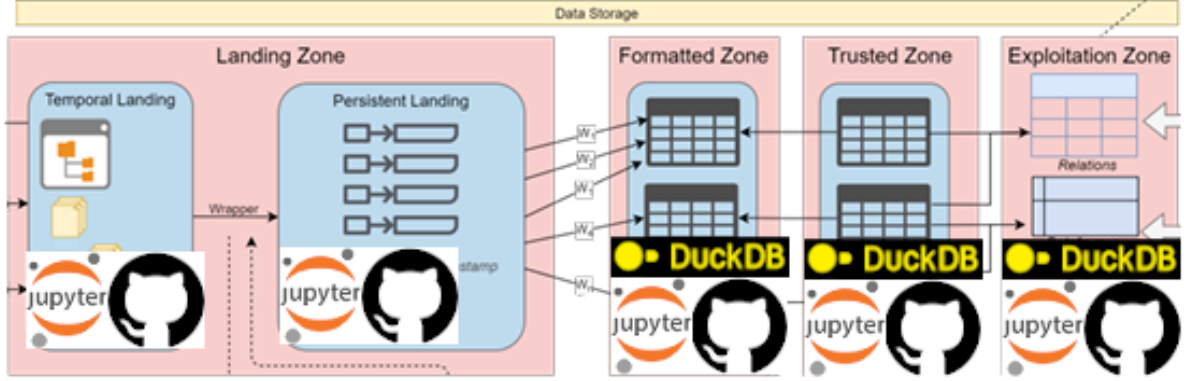


Figure 1: Tools used in the data management backbone

2.1 Landing zone

The landing zone is divided in two: the temporal and the persistent landing zone.

Temporal landing zone creates the `landing/temporal` folder and downloads the data if it does not exist from the original sources through the links of the global `DATA` variable. It has been decided to do it this way to demonstrate that no previous modification is made to the data that has been saved in the temporary area.

Persistent landing zone copies the files found in the `landing/temporal` folder to the persistent storage which will be a new folder called `landing/persistent`, where the files will be renamed by adding the current date to the beginning of the file name.

2.2 Formatted zone

Each excel table found in the `landing/persistent` folder has been created in a relational database. As we have two independent repositories, we decided to create two different databases for each data source. For this we used **DuckDB** because it enables to do connections of Pandas DataFrames and we named created databases `household.duckdb` and `nationalities.duckdb`.

The first step that was done to store the data in the relational database was to read them in pandas DataFrame format at the same time as performing the data homogenization according to a canonical data model.

Household data source was homogenized by coding all its columns, because of the white spaces and the commas that were in the column names and that were confused as another element when the data was transferred to SQL. Also, since we were going to

hard code it, we decided to shorten it and translate it into the international programming language, which is English.

```
household_columns = ['...']

df = pd.read_excel(path, sheet_name='Composicion del hogar', header=[5], names=household_columns)
```

Moreover, it was decided to remove those not homogeneous rows that abstracted the section codes, and get all the data with the same granularity to enable us analyze measures such as: maximum, minimum, average, etc. What was removed can be easily derived using group by and a sum function. So technically we are not losing any information.

Luckily, those rows were easily distinguished in the dataset, since they contain the name of the district (letters) and not the section code (numbers), and they were eliminated as follows:

| | section |
|-----|------------------|
| 0 | Ciudad de Madrid |
| 1 | 01. Centro |
| 2 | 1001.0 |
| 3 | 1002.0 |
| ... | ... |
| 223 | 03.Retiro |
| 224 | 3001.0 |
| ... | ... |

```
df = df[pd.to_numeric(df['section'], \
errors='coerce').notnull()]
```

Table 1: Section column preview

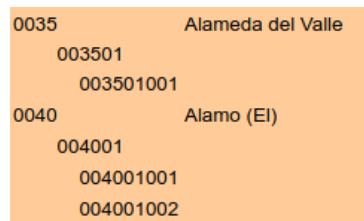
To check that the elimination was done correctly, a small test was created which calculates the difference in rows between the original and the new DataFrame, verifying that the number of rows eliminated is twenty one (the number of districts) plus one (the total of all the districts):

```
assert tmp.shape[0] - df.shape[0] == 22
```

For **nationalities data source** was not necessary to hard code the columns, since none of them had the comma problem, and the homogenization was done through Python functions. The first step was to eliminate the white spaces that were at the beginning and at the end of the name with the function: `df.columns.str.strip()`. After that we replaced the blanks that were in the middle of the name with a `_` which was achieved with: `df.columns.str.replace(' ', '_')`.

Moreover, to homogenize the data total columns that abstracted the countries such as

Oceania, Europe, Asia, etc. were dropped as we decided to have all the data in terms of countries not continents.



| | |
|-----------|-------------------|
| 0035 | Alameda del Valle |
| 003501 | |
| 003501001 | |
| 0040 | Alamo (El) |
| 004001 | |
| 004001001 | |
| 004001002 | |

Figure 2: Original section nationalities column

The total rows in the original nationalities table were displayed like the Figure 2 showed on the left, and to keep those values that were not total in the dataframe, we filtered with the code below which drops all the rows that not contain nine digits meaning that the row is an abstraction.

```
df[df['Madrid_section'].apply(lambda x: len(x.strip()) == 9)]
```

Once all the versions have been read and stored in DataFrame format, the following function was called to **create persistent tables in DuckDB** from the DataFrame format.

```
def df_to_DBtable(DB,df,table):
    """
    Creates a persistent table in DuckDB `DB` from the dataframe `df`
    content with the name `table`.
    """
    con = duckdb.connect(DB)
    con.register(table, df)
    con.execute(f'CREATE OR REPLACE TABLE {table} AS SELECT * FROM {table}')
    con.close()
```

To verify that each database contains correct tables, small test was conducted, using following defined function defined in utils.py:

```
def get_tables(DB):
    """
    Gets all the tables from the `DB`
    """
    con = duckdb.connect(DB)
    tables = con.execute(f'SELECT table_name \
        FROM information_schema.tables').df()['table_name']
    con.close()
    return tables
```

```
# check if all tables in household.duckdb are created
household = utils.get_tables('../household.duckdb')
assert household.size == 3
household
```

| | |
|---|---------------|
| 0 | household2019 |
| 1 | household2018 |
| 2 | household2020 |

```
# check if all tables in nationalities.duckdb are created
nationalities = utils.get_tables('../nationalities.duckdb')
assert nationalities.size == 3
nationalities
```

| | |
|---|-------------------|
| 0 | nationalities2019 |
| 1 | nationalities2018 |
| 2 | nationalities2020 |

2.3 Trusted zone

In trusted zone the core element is a database. There should be a single table per source and therefore we should join all the versions in a one table. This was done by firstly transforming tables to DataFrame format with the function:

```
def DBtable_to_df(DB, table):
    """
    Converts the DB `table` in a data frame format
    """
    con = duckdb.connect(DB)
    df = con.execute(f'SELECT * FROM {table}').df()
    con.close()
    return df
```

To each DataFrame, column Year was added to indicate the version of the table. The resulting DataFrames were unified by rows into one using `pd.concat()` function and the result was then transformed into table of corresponding database using function `df_to_DBtable()`.

These steps resulted that our databases now consist of not only tables per versions but also have one table that joined all existing versions.

Data quality processes were done independently for each repository. Nevertheless both were performed with the same pipeline:

Firstly, we did the **data profiling** where the normal distribution was studied with statistical summary of the numerical variables together with graphs that showed their distribution through the estimate of computed kernel density. The correlation heatmap was also made, where the 10 variables with the highest correlation were analyzed.

Secondly, we deal with the **duplicates** where it was checked whether there were repeated rows or duplicate columns that had been wrongly unified during the process of generating a single table form the different versions.

Thirdly, we check if there where **missing values** in the data with the `isnull()` function and if there were not, we analyzed if the zero values were hidden missing values.

Lastly, we manage with the **outliers** and for this we analyze the distribution diagrams and the boxplots.

2.3.1 Household

The data from this dataset appears fairly consistent in the context of Spain households and it seem to have quite logical values. As an example, there is a high positive correlation between `single_women_aged_16_to_64` and `single_men_aged_16_to_64` meaning that the more single men between 16 to 64 years, the more single women with the same age. Also number of missing values was analyzed and none was found to be missing. Zero values were then converted to *NAs* to observe categories with most zero values.

Regarding the outliers, many were found and we are aware that some machine learning models may be very sensitive to this.

As it was presented in the data profiling, many variables were unbalanced and did not follow the normal distribution.

Box-plots and the dist-plots showed lot of whiskers which indicate a lot of variability outside the upper quartiles. It was decided not to eliminate or impute them, but just to acknowledge the existence of these and take it into account for future analysis.

2.3.2 Nationalities

Within table corresponding to nationalities we found duplicates mainly because of different naming conventions between different versions.

Instead of column names `Belarús` and `República Democrática del Congo` version 2018 consisted of column names `Bielorrusia` and `República Democrática del Cong`, and in contrast remaining versions had column names `Belarús` and `República Democrática del Congo`, but not `Bielorrusia` and `República Democrática del Cong`.

This was handled by firstly crating DataFrames that gathered data from joined table by certain value of Year column. For further explanation these DataFrames were named `n2018`, `n2019` and `n2020`.

Out of `n2018` empty columns were removed (`Belarús` and `República Democrática del Congo`) and remaining column name `Bielorrusia` was changed to `Belarús`, same as `República Democrática del Cong` was renamed to `República Democrática del Congo`. From two of remaining DataFrames empty columns were removed (`Bielorrusia` and `República Democrática del Cong`).

Resulted DataFrames were concatenated into one that was transformed to table `nationalitiesClean` into `nationalities` database as previously explained.

Manipulating over DataFrame type allowed us to easily control every step of the process, and only final result was imported to database as table. This contributed to short connection to the database, rather than keeping connection open for every processing step.

When it comes to data profiling conducted for table referring to nationalities, same as for household variables, plotted distributions were not normal. Right skewed distribution was detected for variables describing total number of Spanish people and total number of foreigners. As there are 193 numerical variables, it was decided to first visualise correlation between first 20 variables and weak correlations between selected nationalities were clearly detected. Missing values were found for ten variables referring to nationalities and those were imputed with zero values. For a selection of variables many outliers were detected and visualised with box plots.

2.4 Exploitation zone

Exploitation zone aims to result with one table that refers to both data sources. In order to achieve this it was necessary to conduct previous step which was trusted zone. Two tables created in trusted zone are corresponding to each data source and don't have any redundant information. Taken these as a starting point, we firstly analyzed which **quality processes** have to be done in order to correctly integrate them into one table.

For table `household` within database `household.duckdb` it was observed that section codes are missing Madrid municipality reference `0796` at the beginning of section code. This was necessary to be modified since table `nationalitesClean` inside database `nationalites.duckdb` refer to all municipalities in Madrid and each has according code prior to section code. After this step, column name describing section was renamed as `Madrid_section` for the purpose of integrating with `nationalitesClean` table, over the same column name. Resulting table `householdClean_Madrid` was added to newly created database `integration.duckdb`.

Table referring to nationalities `nationalitesClean`, as previously described, consisted of more information then needed for integration. Since we only aim to integrate over Madrid municipality code, all section codes beginning with `0796` were extracted. Again some renaming of column names was done to ensure correct integration and created table `nationalitesClean_Madrid` was stored in `integration.duckdb`. Section codes were then compared for both tables and no difference was found. This allowed us to approach to the last step.

Last step in this zone was to finally merge tables `nationalitesClean_Madrid` and

`householdClean_Madrid` into one. This was achieved by inner join over columns referring to section code and year.

3 Operations

To create the operations, we extracted the code from the jupyter notebooks referring to all zones and remodeled it a bit to python format. It was not a very complicated task since we already had the notebooks divided by functions, in order to have a reusable code.

All the Operators were located in a GitHub folder named Operators. The executable script is the `__init__.py` which contains all the functions of the other scripts.

Moreover, a `utils.py` has also been imported, since there are many functions that are shared by all zones, such as obtaining the dataframe of table X from database Y.

The `__init__.py` script calls all zones in the following order:

1. **landing.py:** The same code that was in the jupyter notebook has been copied and pasted.
2. **formatted.py:** Only change that has been done is that now it returns the households and nationalities DataFrames to be able to visualize it in the future with a GUI.
3. **trusted.py:** The trusted is the file that contains the longest code since it includes the joining and all the graphic plots done during the data quality process. At the moment this code does not visualize the graphs but it downloads all that have been executed in the jupyter notebooks such as the correlation matrix, etc.
4. **explotation.py:** All functions from jupyter notebook exploitation were used.

Nevertheless a main function has been added to all the functions, so that they can be executed independently and the code is better structured. Moreover, most of the `print` functions that are showing process steps in console were executed there.

4 Conclusions

All the evaluation criteria of the project have been taken into account, in fact all the code has been structured in functions and independent of the context. What is more, many of

those have been reused in the different notebooks. That is why we find inside the both folders operators and notebook an utils.py file.

The utils file groups all those functions that are reused between the different zones. To carry out in the notebooks the following code has been done:

```
import os, sys
module_path = os.path.abspath(os.path.join('..'))
if module_path not in sys.path:
    sys.path.append(module_path)

import utils
```

In the operators as it follows the structure of a python package, it was done only with the import. Moreover, all python code has been documented and PEP 8 style has been used to make it even easier to read.

We believe that the objectives of this work have been achieved since we achieved to get a minimal viable product that performs the basic tasks needed in a DataOps workflow using DuckDB.