POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Data preparation pipeline

Course: Data and information quality

Authors: **Sara Resta 10667600**
Francesco Scroccarello 10723028

Project ID: **20**

Dataset: **9 - Comune di Milano Esercizi di vicinato in sede fissa**

Academic year: **2024-25**

Contents

1 | Setup choiches

1.1. Libraries

The work was performed in a `Python` environment (version 3.11.3). The setup for the work includes a wide variety of libraries that are commonly used in the data science world. Here follows the list together with a brief motivation:

- `pandas` (version 2.2.3): it allows to store the dataset in a `DataFrame` object that models data as a relation of a structured database, having also useful methods to access and modify values in a very simple way
- `recordlinkage` (version 0.16): it contains the most important methods for the duplicate detection phase
- `scikit-leatrn` (version 1.6.0), `mlxtend` (version 0.23.3): they contain ready-to-use feature extraction techniques, machine learning models and data mining algorithms
- `gensim` (version 4.3.3): contains models for word embedding, such as `Word2Vec`
- `numpy` (version 1.26.4), `scipy` (version 1.13.1): they are scientific computing libraries for linear algebra and statistics
- `matplotlib` (version 3.9.4), `seaborn` (version 0.13.2): libraries for visualization

1.2. Dataset

The dataset under analysis is publicly available, and can be found here together with some quite useful information about it. Unfortunately, it can't be directly sourced using the `pandas.read_csv` function passing it the link, so it has been downloaded locally and loaded directly from the working directory. From a first inspection of the dataset at the available link, the column `ZD` seemed to contain numbers from `1` to `9`. From a web inspection indeed it has been understood that those integers represent the administrative divisions of *Comune di Milano*[2], as shown in Figure 1.1. As a result, column `ZD` represents an ordinal variable and thus it has to be imported as object and not as number.

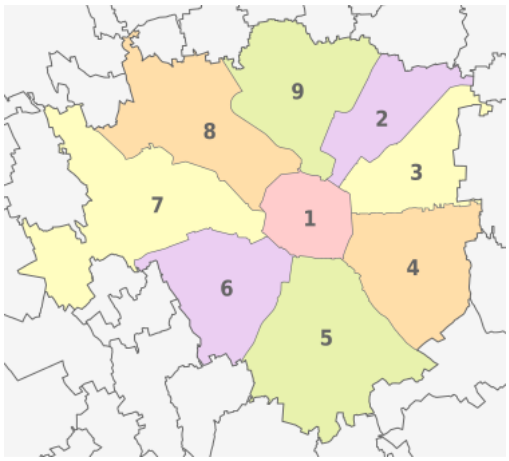

Figure 1.1: Milan administrative divisions

Another useful mention to do is that the people involved in the dataset creation process decided to use as separator the semicolon instead of the usual comma. For these reasons, the correct load of the dataset can be performed in this way:

```
df = pd.read_csv("Comune-di-Milano-Esercizi-di-vicinato-in-sede-fissa-dirty.csv", sep=
    ';', encoding='latin1', dtype={'ZD': str})
```

1.3. Project outlines

The project consisted in implementing a standard data preparation pipeline, plus some extra that have been developed to gain more confidence about the results. The core work can be divided in 2 macro-moments the data exploration phase and the data cleaning phase.

The data exploration phase was a trial to figure out what kind of data were object of the work, together with their semantic. The dataset was also profiled to collect as many metadata as possible to use for the following phases. At the end of the phase, the quality of the dataset was assessed.

During the data cleaning phase, all the key operations were performed in order and in the most efficient and effective manner: data transformation, error correction, and data deduplication. Special care was taken to strike a balance between identifying all errors while preserving the generality of the data. This means that the data was predominantly processed using appropriate algorithms, with manual intervention only in specific cases.

As mentioned, the last project activity was an analysis just to have more practical sense about the direction of the work, which was quite a useful reference point to think about that guided decisions when facing uncertain situations. All details about the implementation in the following section.

2 | Pipeline implementation

2.1. Data Profiling

Data profiling focused on gathering metadata providing valuable information to guide the subsequent steps of data cleaning. It included:

- Basic profiling operations:
- Single column analysis
  - Cardinalities
  - Values distributions
- Dependency discovery
  - Functional dependencies
  - Correlations between numeric variables
  - Association rules

It is important to notice that despite the existance of automatic tools for profiling like `ydata-profiling`, manual operations have been preferred to be more flexible in the metadata collection, to customize the profiling on the basis of the needings and at the same time explore the content of the dataset in depth to become more familiar with it for later work.

2.1.1. Basic profiling operations

Dataset shape and column types were inspected. The initial dataset contained `24136` rows and `13` columns. Out of these, 9 columns had `dtype object` (strings) while 4 columns contained `float` values.

- **Categorical variables**: Settore Merceologico, Insegna, Ubicazione, Tipo via, VIa, Civico, Codice via, ZD, Settore Storico Cf Preval.

- **Numerical variables**: Superficie Vendita, Superficie Altri Usi, Superficie Tabelle Speciali, Superficie Totale

2.1.2. Single columns analysis

Single column analysis was aimed at computing **cardinalities** and inspecting **columns distributions**.

Cardinalities

For each column the number of missing values, actual values and unique values were computed. Results are shown in Table 2.1

	Settore Merceologico	Insegna	Ubicazione	Tipo via	Via	Civico	Codice via	ZD	Settore Storico Cf Preval	Superficie Vendita	Superficie Altri Usi	Superficie Tabelle Speciali	Superficie Totale
Missing	105	17147	1	2	1	650	2	2	33	30	15098	23009	36
Actual	24031	6989	24135	24134	24135	23486	24134	24134	24103	24106	9038	1127	24100
Cardinality	56	5583	16907	20	2331	386	2348	10	2228	250	438	117	642

Table 2.1: Overview of Missing, Actual, and Unique values for single columns

Numeric columns distributions

Two subsets of the dataset were created, one for numerical variables and another for categorical variables, based on the DataFrame's structure. The subset of the numeric variables was inspected with the function `describe` which computed count, mean, standard deviation, minimum value, maximum value, first quartile, second quartile and third quartile (Table 2.2). Of a note, the maximum value for `Superficie Vendita` is `250` coherently

	Superficie Vendita	Superficie Altri Usi	Superficie Tabelle Speciali	Superficie Totale
count	24106.0	9038.0	1127.0	24100.0
mean	56.76	67.85	37.14	98.35
std	49.28	280.89	29.82	266.69
min	0.0	0.0	1.0	0.0
25%	25.0	10.0	13.0	35.0
50%	40.0	21.0	30.0	55.0
75%	70.0	50.0	52.5	100.0
max	250.0	9056.0	230.0	11942.0

Table 2.2: Summary Statistics of Superficies

with the maximum dimension reported here. Figure 2.1 shows histograms of value distributions of numeric variables. Values distributions were plotted using `histplot` from

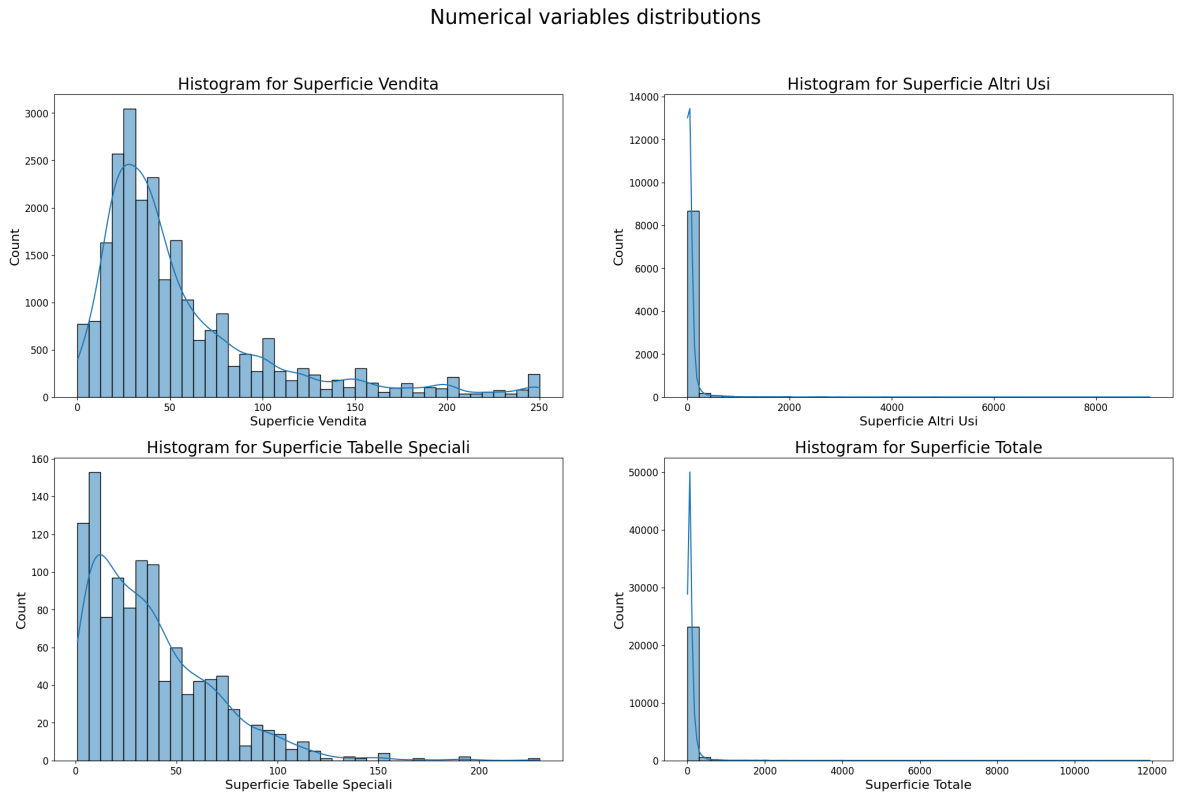

Figure 2.1: Numeric variables distributions

`seaborn`, the parameter `kde` was set to `True` to display the Kernel Density Estimate.

Categoric columns distributions

Categorical variables distributions were inspected with the function `barplot` from `seaborn`. For columns containing many distinct values, only values above a certain occurrence-threshold were shown for a better visualization. Bigger barplots are reported in **Appendix A**.

Settore Merceologico Figure A.1 represents the occurrences of different values for the column `Settore Merceologico`. Some shops may have multiple values, which are separated by the `;` semicolon symbol. Even though there are formatting differences and some values are concatenated, it is possible to identify five distinct product categories:

- `alimentare`
- `non alimentare`
- `tabella speciale monopolio`
- `tabella speciale farmacie`

- `tabella speciale carburanti`

Insegna represents the store sign. For this reason, it shows very different values, as it can potentially be different for each shop. However, some "more generic" values of shop sign are present for more shops, for example `farmacia` (166), `bar tabacchi` (94), `tabacchi` (55), `panificio pasticceria` (42). The `barplot` representing the most frequent shop signs is shown in Figure A.2 (only values with at least four occurrences are shown for dimensionality issues).

Tipo via is a three-letters code that identifies the kind of street of the address. Figure 2.2 shows the occurrences of different kind of street in addresses. Of a note, there are two occurrences for `LARGO` which has lenght `5` instead of `3` and one occurrence for `7` which has lenght `1` and it is a digit. Additionally, four occurrences of `VIE` are present which may be typos of `VIA`. The most frequent values are `VIA` (17379 occurrences), `VLE` (2699 occurrences) and `CSO` (1819 occurrences).

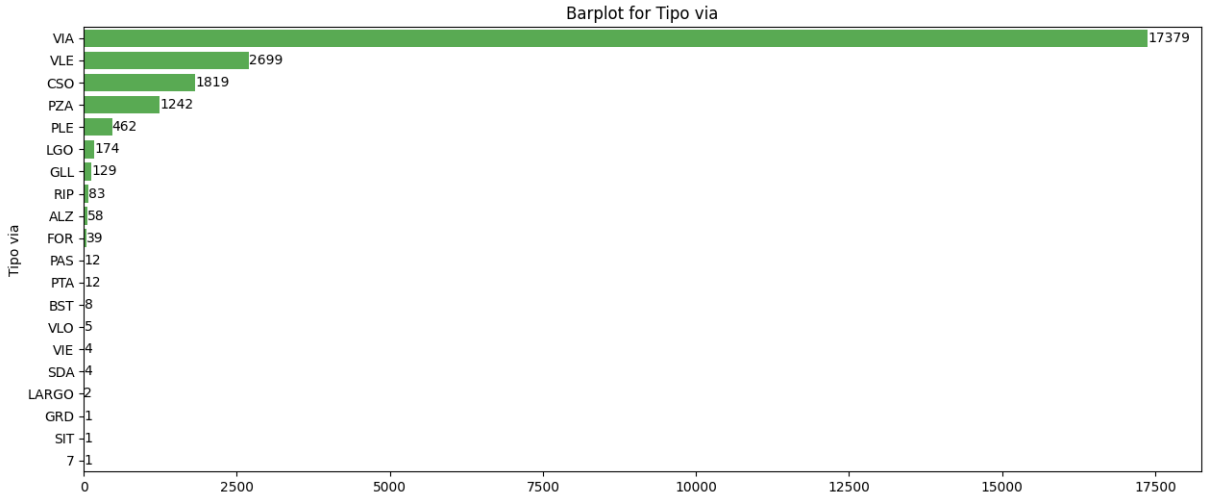

Figure 2.2: Barplot of column `Tipo via`

ZD represents the administrative division of *Comune di Milano*. Frequencies of this column are shown in Figure 2.3. The most frequent value is `1` (5732) occurrences, indicating that a large proportion of the shops are located in the centre of Milan, as expected. Although only values from `1` to `9` should be present, there is one occurrence for `60` which may be a typo.

Settore Storico Cf Preval represents the product category with respect to historical product category table. Although it is semantically similar to the column `Settore`

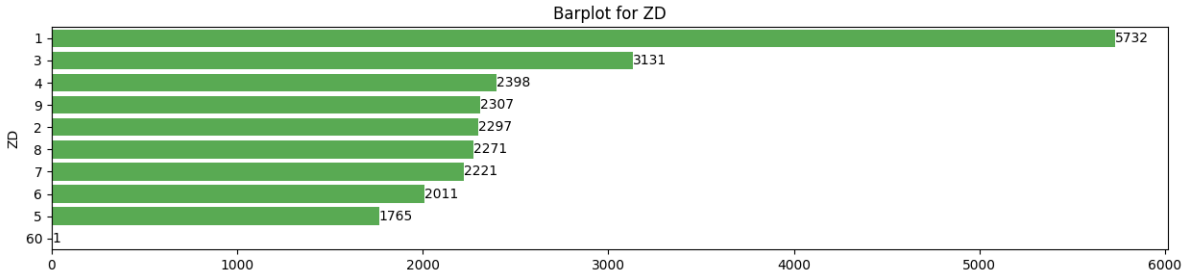

Figure 2.3: Barplot of column `ZD`

`Merceologico` its values show higher variability. It seem to represent shops' product categories in a more detailed way. Frequencies are shown in Figure A.3. Values with the highest occurrencies are `abbigliamento` (2163), `bar tabacchi` (576) and `cartoleria cancelleria` (570).

Ubicazione, Via, Civico, Codice via barplots of these columns are not shown because they would not be very informative as values are almost unique for each shop.

2.1.3. Dependency discovery

Ideally, all the functional dependencies in the dataset would have been extracted using `tane` or `fd_mine`. Unfortunately, the complexities of these algorithms were too high to give a result in a reasonable time on the dataset, in particular the time complexity is $O(2^n m)$, where `n` is the number of attributes of the dataset and `m` is the total number of tuples. Thus, after some basic reasoning about the semantic of the columns, dependencies have been extracted "the hard way", trying to look at specific columns and using specific methods.

Inclusion dependencies

From a visual inspection it was clear that the column `Ubicazione` was likely to contain a combination of `Tipo via`, `Via`, `Civico` and `ZD`. Indeed, it was true for the 97.7% of the rows, turning out as a relaxed inclusion dependency.

Correlations between numeric variables

Pearson's and Spearman's correlations between numeric variables were computed using the function `corr` from `pandas`. The correlation p-values were computed using `pearsonr` and `spearmanr` from `scipy`. Only not null values were included in the analysis. Pearson's correlation test measures only linear relationships between variables, whereas Spearman's

correlation test evaluates all monotonic relationships. Correlations with a p-value lower than `0.05` were considered statistically significant. The correlations are summarized in Figure 2.4. Significant strong positive (correlation coefficient higher than `0.6`) Pearson's correlation were found between:

- `Superficie Vendita` and `Superficie Tabelle Speciali`
- `Superficie Altri Usi` and `Superficie Totale`

A significant low positive correlation was present between `Superficie Vendita` and `Superficie Totale`. As for Spearman's correlations, all correlations were statistically significant; `Superficie Vendita` and `Superficie Altri Usi` showed a strong positive correlation with `Superficie Totale` and `Superficie Tabelle Speciali` showed a strong positive correlation with `Superficie Vendita`. The correlation between `Superficie Totale` and the other "superfici" fields like `Superficie Vendita` and `Superficie Altri Usi`, confirmed a first hypothesis formulated after a visual inspection, that is the column could possibly represent the sum of the other columns. What seemed strange instead was the correlation between `Superficie Vendita` and `Superficie Tabelle Speciali`. A first suspect was that, due to the meaning of the column name, its values could have been included in the measurements about `Superficie Vendita`, since they are closely related. With a bit more of exploration it has been found out that this was true in 660 rows, while it was false in just 16; it seemed reasonable to mark also this as a relaxed inclusion dependency.

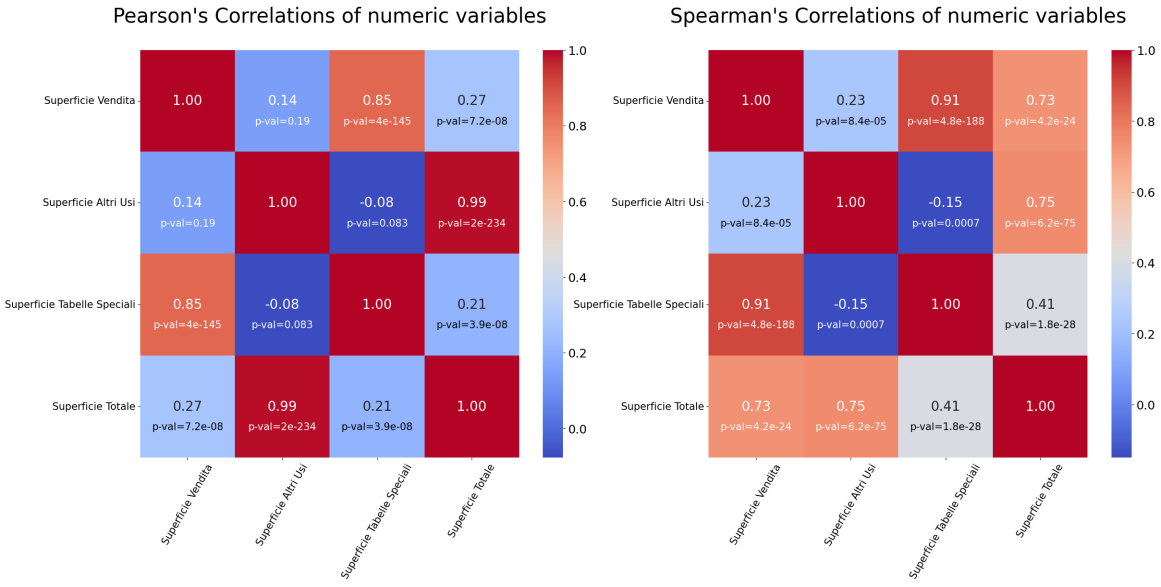

Figure 2.4: **Left**: Heatmap with Pearson's correlations between numeric variables. **Right**: Heatmap with Spearman's correlations between numeric variables.

Association rules

One of the most effective tools to look for patterns among categorical variables is **association rule mining**, which has been applied to discover relationships between suspected columns. For reusability purposes, a single function was defined (code in section B.1) to extract frequent itemsets and generate rules. These rules were stored in a supporting DataFrame, designed to be reusable when needed and formatted as follows:

```
{attribute_1} : {value_1} => {attribute_2} : {value_2}
```

Notice that the function implements the `fpgrowth` algorithm to extract the frequent itemsets since it is the most efficient one for big datasets. Also, due to the untransactional nature of the items, the trend was to use low `min_sup` and high `min_conf` to get usable results. Rules that are relevant where:

- **Codice Via and ZD**: since city's routes are identified by a unique `Codice Via`, they must correspond to the same `ZD`. An association rule model as been fit to extract the most frequent patterns, so that it was less probable the propagation of inconsistent pairs due to association errors or typos.
- **Via and ZD**: for a similar reason it was useful to extract rules between `Via` and `ZD` since italian route names in the same city must be unique.
- **Settore Storico Cf Preval and Settore Merceologico**: it was useful to discover relationships between `Settore Storico Cf Preval` and `Settore Merceologico` as the semantics of the 2 are commonly perceived as related.

It is worth mentioning that all the meaningful combinations of attributes have been tested, but lots of them returned either empty lists of rules or pure noise rules involving `NaN` values mostly, thus they were discarded to maximize the quality of the metadata.

2.2. Data quality assessment

The quality of the data set was assessed with:

- **Cardinality dimensions** were computed for each column (results in Table 2.3):

$$uniqueness(col) = \frac{Cardinality(col)}{n_{rows}}$$
$$distinctness(col) = \frac{Cardinality(col)}{\#notNaN}$$
$$constancy(col) = \frac{Mode(col)}{n_{rows}}$$

- **Completeness** defined as
$$\frac{\#NotNaNcells}{n_{rows} \times n_{cols}}$$
- **Consistency**: the dependencies spotted visually and the dependencies spotted with association rules were verified for each row in the dataset. 8 checks were performed for each row, for each verified check 0.125 was added to the sum of consistencies. In the end that sum was divided for the number of rows, so that the consistency has been computed as:
$$\frac{consistent\_tuples\_scores\_sum}{n_{tuples}}$$
More precisely, these checks were performed:

  1-4 The information present in the columns `Tipo via`, `Via`, `Civico`, `ZD` must be contained in the column `Ubicazione`. A different check was performed for each of the four columns.

  5 Values in the columns `ZD` must be digits from 1 to 9.

  6 Association rules found between `ZD` and `Codice via`.

  7 The sum of the values in `Superficie Vendita` and `Superficie Altri Usi` must be equal to `Superficie Totale`.

  8 The value of `Superficie Tabelle Speciali` is included into `Superficie Vendita`.

- **Accuracy** could not be computed as no ground truth was provided with the dataset.
- **Timeliness** could not be computed since no temporal information about `currency` and `volatility` of data have been provided.

Overall, the dataset showed `0.8212` completeness and `0.6913` consistency. Table 2.3 reports uniqueness, distinctness and constancy for each column.

	Settore Merceologico	Insegna	Ubicazione	Tipo via	Via	Civico	Codice via	ZD	Settore Storico Cf Preval	Superficie Vendita	Superficie Altri Usi	Superficie Tabelle Speciali	Superficie Totale
Uniqueness	0.0023	0.2313	0.7005	0.0008	0.0966	0.0160	0.0973	0.0004	0.0923	0.0104	0.0181	0.0048	0.0266
Distinctness	0.0023	0.7988	0.7005	0.0008	0.0966	0.0164	0.0973	0.0004	0.0924	0.0104	0.0485	0.1038	0.0266
Constancy	0.6823	0.0069	0.0010	0.7200	0.0121	0.0623	0.0121	0.2375	0.0896	0.0516	0.0242	0.0043	0.0426

Table 2.3: Overview of Metrics for Uniqueness, Distinctness, and Constancy for each column

2.3. Data Wrangling

Data wrangling was applied only to the columns containing string objects as the columns containing float values (the ones named `Superficie...`) did not show any particular formatting issue. The operations performed on each column are listed below.

- **Settore Merceologico**: as mentioned in section 2.1.2, there were only few semantically different values and in some cases entries contained more than one value concatenated with `;`. Thus, this column format was standardized as follows:
  - All values were put to lowercase
  - The first reported product category was assumed to be the main one. Thus, two new columns were created: `Settore Merceologico Principale` containing only the first value and `Settore Mercologico Secondario` containing the remaining values (if present, otherwise it was assigned to `NaN`). The `;` was replaced with `,` when more than one value was present in `Settore Mercologico Secondario`. This was aimed to highlight the main product category of each shop and simplify imputation of missing values. In fact `Settore Merceologico Principale` has now only `5` possible values, making the classification of shops according to the main product category potentially easier.
  - The original column `Settore Merceologico` was dropped.
- **Insegna**: values were put to lowercase and characters that were not alphanumeric or dots were replaced with spaces.
- **Ubicazione**: as mentioned in section 2.1.2, it contains information related to `Tipo via`, `Via`, `Civico`, `ZD`. For most columns, information was separated by `;` semicolon symbol. As this column may be useful for studying inconsistencies and performing missing values imputation, the information was extracted using `regular expressions` and saved into a temporary dataframe. The procedures used to extract different columns are listed below, the code with all the regular expressions used is reported in section B.2.
  - `Tipo via tmp`: extracted as the first word in the string. Additional space characters were then removed.
  - `Via tmp`: extracted as a string starting from characters after `Tipo via` and ending before `N. or some other non-caps character`. Space characters at the beginning or end of string were removed.

  - `Civico tmp`: extracted as digits and characters that follow `N.`, `num.`, `civ.` or `civico`. Space characters at the beginning or end of string were removed and letters were put to uppercase. When a letter was present this was separated by the number with `/`. Zeros at the beginning of the number were removed.
  - `ZD tmp`: extracted as the single digit following `z.d.`

  Four new columns emerged:

  - `Accesso`: extracted as the string following `accesso:`. As the column name is already `Accesso`, values like `accesso interno` and `accesso esterno` were reported as `interno` and `esterno`. Additional spaces at the beginning or end of the string were removed.
  - `Isolato` extracted as the string following `isolato:`. Zeros at the beginning of the string and spaces at the beginning or end of the string were removed.
  - `Website`: extracted as the word beginning with `www`.
  - `Altre info`: used to store additional information.

  In the end, the columns derived from `Ubicazione` were added to the dataset and column `Ubicazione` was dropped.

- **Tipo via, Via**: spaces at the beginning and end of the string were removed and characters were put to uppercase.
- **Civico**: spaces at the beginning and end of the string were removed and characters were put to uppercase. For format consistency, `/` character was added before letters when these were present.
- **Codice via**: spaces at the beginning and end of the string were removed.
- **Settore Storico Cf Preval**: spaces at the beginning and end of the string were removed and characters were put to lowercase.
- **Superfici**: for the last 4 columns of the dataset, namely the ones related to the sufrace metrature registration (`Superficie Vendita`, `Superficie Altri Usi`, `Superficie Tabelle Speciali`, `Superficie Totale`), formats were already standards and values were already in m^2, standard unit of measurements. As already mentioned, no transformation has been applied.

2.4. Inconsistencies handling

The relationships used to compute the total consistency of the dataset in section 2.2 were used to spot inconsistencies that persisted in the wrangled dataset. More precisely, as `Ubicazione` was dropped, columns related to the address (`Tipo via`, `Via`, `Civico`, `ZD`) were compared to the temporary columns extracted from `Ubicazione` (`Tipo via tmp`, `Via tmp`, `Civico tmp`, `ZD tmp`). Additionally `ZD` was checkecked to see if there were rows cotaining different values from digits `1-9`. The association rules found between `Codice via` and `ZD` were also checked. As for surface areas, accordingly to what found in 2.1.3, these checks were performed: $S_{vend} + S_a == S_{tot}$ and $S_{vend} + S_{ts} <= S_{tot} - S_a$ where: S_{vend} is `Superficie Vendita`, S_a is `Superficie Altri Usi`, S_{ts} is `Superficie Tabelle Speciali` and S_{tot} is `Superficie Totale`. Rows containing inconsistencies were appended to a separate `DataFrame` and the column `Reason` was added to describe why the row is inconsistent. Comparisons where `NaN` values were present were not performed and thus were not considered among inconsistencies because they will be handled in section 2.5. Table 2.4 shows the counts of different reasons for inconsistencies. The highest number of inconsistencies are found in the columns related to shops' surface areas sum (235 rows).

Combination of Features	Count
Superficie totale	235
Nome via, Civico	226
Tipo via, Nome via, Civico	167
ZD	155
Tipo via, Nome via	28
Civico	17
Nome via	16
Superficie totale, Superficie tabelle speciali	7
ZD messed zone number	5
Nome via, Civico, Superficie totale	2
Tipo via, Nome via, Civico, ZD	1
Nome via, Civico, ZD	1
Tipo via, Nome via, Superficie totale	1
Civico, Superficie totale	1

Table 2.4: Summary of inconsistency reasons occurrencies

Inconsistencies related to the address include all reasons containing at least one column name related to the address (`Tipo via`, `Via`, `Civico`, `ZD`). Rows containing inconsistencies in columns related to the address were dropped as there was no criteria to establish which address was the correct one. `618` rows were dropped.

Inconsistencies related to the surface areas include rows where at least one of these conditions is verified:

- The sum of `Superficie Vendita` and `Superficie Altri Usi` does not equal `Superficie Totale`.
- The sum of `Superficie Vendita` and `Superficie Tabelle Speciali` is higher than the difference between `Superficie Totale` and `Superficie Altri Usi`.

All inconsistencies were corrected modifying the value of `Superficie Totale` assuming that values reported in `Superficie Vendita` and `Superficie Altri Usi` were correct.

2.5. Missing values handling

2.5.1. Columns related to the address

Columns `Tipo via`, `Via`, `Codice via` and `ZD` did not have missing values. On the other hand, the column `Civico` presented `616` missing values which were imputed using the column `Civico tmp` extracted from `Ubicazione` in section 2.3. After imputation `63` rows still had `NaN` value in `Civico` and were filled with the standard value `SNC` (Senza Numero Civico). Columns `Tipo via tmp`, `Via tmp`, `Civico tmp` and `ZD tmp` were dropped.

Columns `Accesso` and `Isolato` respectively had `30.60%` and `17.83%` of missing values. Thus these columns were not dropped and `NaN` valeus were replaced with the standard value `NP` (Non Presente).

Columns `Website` and `Altre Info` showed `99.87%` and `82.42%` of missing values respectively, thus they were dropped.

2.5.2. Surface Areas

Missing values percentages in surface areas are reported in Table 2.5 Missing values were filled according to the relaxed dependencies found in section 2.1.3. The complete list of performed passages is shown below.

Superficie	NaN %
Vendita	0.11
Tabelle Speciali	95.33
Altri Usi	62.35
Totale	0.14

Table 2.5: NaN Percentages by Superficie

1. `NaN` values in `Superficie Vendita` were filled with the values of `Superficie Tabelle Speciali` (when this information was present).
2. `Superficie Altri Usi` was imputed as $Superficie_{Totale} - Superficie_{Vendita}$.
3. The `median rate` between `Superficie Vendita` and `Superficie Altri Usi` was computed to handle rows contianing only the information about `Superficie Totale`. Then, `Superficie Vendita` and `Superficie Altri Usi` were imputed as follows:
   - $Superficie_{Vendita} = Superficie_{Totale} - Superficie_{Totale}/(median\_rate + 1)$
   - $Superficie_{AltriUsi} = Superficie_{Totale}/(median\_rate + 1)$
4. For rows with no information about surface areas, `Superficie Vendita` and `Superficie Altri Usi` were imputed using the `median`.

2.5.3. Settore Merceologico Secondario, Insegna and Settore Storico Cf Preval

Missing values were replaced with the standard value `NP` (Non Presente). Although `Settore Merceologico Secondario` and `Insegna` showed more than `50%` missing valeus, they were not dropped as the absence of values was still informative. In fact, for `Settore Merceologico Secondario` the standard value may indicate that the shop sells products belonging to only one product category while for `Insegna` the standard value may indicate that there is no shop sign. On the other hand, `Settore Storico Cf Preval` contained only `0.11%` of missing values.

2.5.4. Settore Merceologico Principale

Missing values of this column were handled with machine learning based imputation. Given the discovered associations between `Settore Merceologico` and `Settore Storico Cf Preval` (section 2.1.3) 10 numerical features were generated from `Settore Storico`

`Cf Preval` using `Word2Vec` (library `gensim`). These features were then utilized to train a `RandomForestClassifier` (`scikit-learn` library) for the imputation process. The dataset without rows containing `NaN` values in the target column was used as training set. The training process involved an extensive hyperparameter optimization using `GridSearchCV` with `10-fold cross-validation` to identify the best combination of parameters for the model. The parameter grid is shown in Figure 2.5.

```
param_grid = {
    'n_estimators': [100, 200, 500],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf':  [1, 2, 4]
}
```

Figure 2.5: Parameter grid for RandomForestClassifier

where:

- `n_estimators` is the number of trees.
- `max_depth` is the maximum depth of each tree.
- `min_samples_split` is the minimum number of samples required to split an internal node.
- `min_samples_leaf` is the minimum number of samples required to be at a leaf node.

The model's performance was evaluated based on the average accuracy across the 10 folds. The best accuracy score obtained, `0.927`, was achieved with the following hyperparameters:

- `max_depth` = 20
- `min_samples_leaf` = 2
- `min_samples_split` = 2
- `n_estimators` = 500

2.6. Outliers handling

Outliers are the last category of errors to fix in the dataset. Several methods were employed to discover anomalous tuples either at column level or directly at tuple level based on specific features. The methods applied can be grouped into 3 categories:

- **statistical methods**
- **distance-based methods**
- **model-based methods**

2.6.1. Statistical methods

Statistical methods were used to uncover outliers in numerical columns, that were `Superficie Vendita` and `Superficie Altri Usi`. The first approach was to rely on the famous `Grubb's test`, because it is specific for one-outlier detection within a set of points. A custom iterative implementation of it was defined in the function `grubbs_test` (code in section B.3) to return all the outliers contained in the column, meaning that the test is iterated until the null hypothesis is no longer rejected, keeping track of the points for which the alternative hypothesis is true, which are then returned. To enrich the study and have more sources to base the final decision on, 3 of the simplest, still very effective, tests were selected, that are:

- **z-test**: it standardizes the values with the usual formula $\frac{x-\mu}{\sigma}$, where μ is the `sample mean` of the column, while σ is the `sample standard deviation` to then compare it with respect to the given threshold, set to 3 just to be consistent with the usual `3 sigma rule` since no other particular assumptions were meaningful
- **robust z-test**: robust version of the `z-test` where `sample mean` and `sample standard deviation` have been replaced with `median` and `MAD` (Median Absolute Deviation) to have a breakpoint of 50%
- **interquantile-range-test**: chosen because it performs no particular assumptions on distribution of data but it exploits instead the notion of `quantile distances` to catch the points living outside bounds imposed by those values

All of these steps proved to be highly beneficial in the process. The first task, which was also the simplest, involved running all the tests in parallel to identify differences in the results and investigate their causes. It was found that the `Grubb's test` was particularly effective in detecting outliers, while the other tests showed some inconsistencies with each other. Thus, the final strategy implements a cascade of `Grubb's test` followed by a voting mechanism, which will be later referred as `voting test`. The algorithm is pretty simple, first, the `Grubb's test` was run to detect all potential outliers. Then, the other three tests were executed in parallel. The algorithm extracted from the `Grubb's test` results only those where at least two out of the three additional tests agreed, which proved to be a very effective approach. It is worthy mentioning that a trial has been done also

with a full non parametric approach using `Kernel Density Estimation` to estimate the multivariate distribution describing `Superficie Vendita` and `Superficie Altri Usi` without any specific assumption, and then performing thresholding on the values using a `quantile` to get values in low density regions, but this method actually didn't spot any meaningful erroneous vector, thus in the end it resulted in an unused method.

Superficie Vendita

For `Superficie Vendita`, the `Grubb's test` detected no outlier, then the `voting test` has been performed. An exploration of the obtained results, shown in Table 2.6, revealed that there were no outliers, as the detected values did not apppear strange. Consequently, no tuples were removed.

count	1943
mean	186.451879
std	35.834147
min	138
25%	150
50%	180
75%	211
max	250

Table 2.6: Statistics about results of the voting test for Superficie Vendita

Superficie Altri Usi

For `Superficie Altri Usi` instead, the `Grubb's test` actually detected some values. An exploration of the results revealed something quite interesting, as shown in Table 2.7. . A negative `min` value suggests that there was at least 1 negative value, most probably more than 1, that weren't spotted during the exploratory phase though. This means that the values must have appeared in a subsequent phase; in particular, their presence is very likely to be due to the inconsistency correction phase 2.4, meaning that the registration of the surface fields of that tuples couldn't be fixed in any way, and they are actually erroneous tuples. All tuples having negative `Superficie Altri Usi` were dropped. The next step was the `voting test` whose results led to some considerations, Table 2.8. . Values per-se are not that weird considering that the semantic of the column could refer also to some kind of warehouses, but their magnitude seems very strange for the city

count	960
mean	79.750000
std	593.517215
min	-133
25%	0
50%	0
75%	25
max	11736

Table 2.7: Statistics about results of the Grubbs test for Superficie Altri Usi

count	911
mean	647.790340
std	1023.802884
min	156
25%	204
50%	301
75%	630
max	11736

Table 2.8: Statistics about results of the voting test for Superficie Altri Usi

of Milan. After a bit of brainstorming, the final decision was not to drop them since they are in the end believable values, and dropping them just because of some unverified assumptions would have meant a loss of information.

2.6.2. Distance based methods

The notion of distance between points is often very useful to discover tuples living in low density regions, since the density can be defined as a function of the distance between the points. In particular, the most effective methods are the ones based on local distances, which are the most computationally expensive ones though. The selected reference algorithm is the `Local Outlier Factor` [1], which has been used to detect categorical outliers since for string objects there are effective methods to compute distances. However, the computational complexity of executing both `Local Outlier Factor` and a possible string distance function such as `edit distance`, `levenshtein distance` or

`jaro-winkler distance` was too high to get a result for this dataset in a reasonable amount of time. Thus, the strategy implemented relies on `one-hot encoding` of the values which, despite being a non semantic encoding with the vectors mutually orthogonal, produced very accurate results due to the way `lof` computes density scores. The procedure was immediate and the workflow was the same for all the categorical columns: encode the column, run the `lof` algorithm, look at the results, for every entry, if any, decide between 3 options: **drop**, **correct** (with a refactor operation), **do nothing**. The Table 2.9 summarizes the work done column by column.

Column	Result count	Operation done	Motivation
Settore Merceologico Principale	0	nothing	no outlier found
Settore Merceologico Secondario	13	nothing	just false positives
Tipo via	17	refactor	typing issues
Civico	205	refactor	typing issues
Codice via	0	nothing	no outlier found
Accesso	0	nothing	no outlier found
Isolato	0	nothing	no outlier found
Settore Storico Cf Preval	40	drop 6	semantically wrong

Table 2.9: Summary of categorical outlier handling

2.6.3. Model based methods

The features of the dataset are also something that can help spot some erroneous tuples using some machine learning models. In particular, unsupervised learning is very effective because it doesn't need any labels, but instead tries to uncover anomalous patterns based on the input features themselves. The algorithm selected was `Isolation Forest` since, due to its high velocity in both training and inference, it allowed to fit a forest of lots of `isolation trees`, giving output results with a very low variance. The model was fit over `Settore Merceologico Principale` and `Settore Storico Cf Preval`, and it was able to spot something never seen in any exploratory moment: some shops with historical selling area not related to food have been associated with a main selling area of eating products, which could mean that some errors have been made during the insertion phase. Since they were not so much, 18 rows, they were dropped to increase the number of correct tuples.

2.7. Duplicates handling

Data deduplication was the last step of the cleaning process, and of the whole data preparation pipeline, for which the dataset has been lightened from any redundancy left. The first trivial thing done was to drop all the exact duplicates that were very few, just 35 rows. Then, all inexact duplicates have been found using some `record linkage` algorithms, and then a step of `conflict avoidance` has been performed.

2.7.1. Record linkage

Record linkage was used to identify pairs of tuples referring to the same object, and since the duplication at this point is inexact, it was crucial to identify the most discriminant columns by considering their semantics and understand how to relax the comparisons. The choice was to study only matchings regarding categorical variables since surface data can vary over time and might mislead the process of identifying the same shop. The algorithm used to perform the process was the `Sorted Neighbourhood`, despite its computational complexity, since it is one of the most powerful for this task, that can exploit also the sort on a combined key. The sorting key selected was a combination of `Tipo via` + `Via` + `Civico`, and the reason is that if 2 entries refer to the same shop, then they must be very close in terms of locations. Notice though that it cannot be assumed that if a pair of tuples has matching `Tipo via`, `Via` and `Civico` they are automatically the same shop since there could be situations of 2 shops located in the same building. The algorithm extracted the results of the comparisons in the form of a dataframe whose rows identify a pairing and columns contain answers to comparison of the 2 in all the attributes under study. First of all, the rows having 100% of the matches have been marked as duplicated, as they were only 310. Then another strategy has been found to detect the remaining duplicates: if a pair had `Insegna`, `Tipo via`, `Via`, `Civico` matching and their values were not standard values, they were also marked as duplicates. The rationale is that it is very unlikely that 2 shops with the same banner are located in the same building; they were 40 rows. It is worth mentioning that also other combinations have been tested but with no meaningful duplicates caught.

2.7.2. Conflict Avoidance

Once pairs of tuples have been marked as duplicated, the conflict has been solved with the `conflict avoidance` technique, which consists in defining a simple strategy to chose just one of the tuples and discard the other, without caring about how to merge results into a new and more complete representation, since no priors were given about the correctness

of values in the tuples. The most meaningful strategy was to discard the tuple with the smallest index since it is reasonable to assume that data were sequentially collected, meaning that highest indexed data were also the most recent ones.

3 | Results

At the end of the data quality improvement process, it is always useful to perform another round of assesment to analyze the results of the job done.

3.1. Dimension definition

Since the dataset has been somehow transformed, it is necessary to define again the quality dimensions, that luckily were almost the same. The absence of a ground truth and of any temporal information forbid again to study both **Accuracy** and **Timeliness**. The only metrics that are possible to evaluate are still **Completeness** and **Consistency**, computed in the same way as before:

$$completeness = \frac{number\_cells\_not\_empty}{total\_number\_of\_cells}$$

$$consistency = \frac{tuples\_consistency\_percentage}{total\_number\_of\_tuples}$$

Notice that some of the consistency checks performed have ben changed due to the modification of the structure of the data schema itself. Each of the checks account now for the 25% of tuple consistency, and the list is:

- `ZD` has to be a number from 1 to 9
- association between `Codice via` and `ZD` consistent with the rules mined previously
- domain constraint that `Superficie Vendita` must be up to 250 m^2
- `Tipo via` is a three letters word

3.2. Single column analysis

Single column analysis was repeated on the cleaned dataset and the following cardinality measures were computed for each column: number of missing values, number of actual

values, number of distinct values. Results are shown in Table 3.1. Of a note, no miss-

	Settore Merceologico Principale	Settore Merceologico Secondario	Insegna	Tipo via	Via	Civico	Codice via	ZD	Accesso	Isolato	Settore Storico Cf Preval	Superficie Vendita	Superficie Altri Usi
Missing	0	0	0	0	0	0	0	0	0	0	0	0	0
Actual	21836	21836	21836	21836	21836	21836	21836	21836	21836	21836	21836	21836	21836
Cardinality	5	16	5130	17	2282	538	2300	9	4	476	2016	250	556

Table 3.1: Overview of Missing, Actual, and Unique values for single columns in the cleaned dataset

ing values were found in any of the column of the cleaned dataset and the number of actual values is `21836` for each column, which corresponds to the number of the rows of the cleaned dataset. For each column the quality metrics uniqueness, distinctness and constancy were assessed again; results are shown in Table 3.2. For most columns,

	Settore Merceologico Principale	Settore Merceologico Secondario	Insegna	Tipo via	Via	Civico	Codice via	ZD	Accesso	Isolato	Settore Storico Cf Preval	Superficie Vendita	Superficie Altri Usi
Uniqueness	0.0002	0.0007	0.2349	0.0008	0.1045	0.0246	0.1053	0.0004	0.0002	0.0218	0.0923	0.0114	0.0255
Distinctness	0.0002	0.0007	0.2349	0.0008	0.1045	0.0246	0.1053	0.0004	0.0002	0.0218	0.0923	0.0114	0.0255
Constancy	0.7476	0.9307	0.7078	0.7195	0.0124	0.0626	0.0124	0.2375	0.6393	0.1805	0.0856	0.0527	0.4562

Table 3.2: Overview of Uniqueness, Distinctness, and Constancy for each column (rounded to 4 decimals)

results are comparable with the ones on the dirty dataset (Table 2.3). This can be somehow expected as columns like `Via`, `Civico`, `Codice via`, `Settore Storico Cf Preval`, `Superficie Vendita`, `Superficie Altri Usi` had many different values, regardless the format. On the other hand, columns `Tipo via` and `ZD` did not show a high cardinality even in the dirty dataset; only few values were refactored during cleaning and so the metrics did not show significant changes. The constancy observed for `Insegna` increased significantly from `0.0069` in the dirty dataset to `0.71` in the cleaned dataset. This improvement is attributed to the introduction of the standard value `NP` for tuples where the shop sign was not reported. The column `Settore Merceologico Principale` showed the highest difference as both uniqueness and distinctness decreased by 10 times in the cleaned dataset. This is due to the split of the original column `Settore Merceologico` in the two columns `Settore Merceologico Principale` and `Settore Merceologico Secondario`. This lead to values refactoring the number of distinct values went from `56` (in the original column `Settore Merceologico` to `5` (in the column `Settore Merceologico Principale`). This allowed to focus on the primary product category of the shop and perform an effective missing values imputation.

3.3. Dimension measurements

The assesment has then been performed and a score has been given to the monitored dimensions:

- Completeness: 1.0
- Consistency: 0.8093

These are very big improvements with respect to the beginning of the work; in particular, 100% of completeness has been achieved through the use of standard values together with more complex imputations to provide to a subsequent work a complete and fully informative representation of the data. The consistency value however didn't achieve 100% due to the presence of `association rules` based checks, which are of course not been extracted for each combination of data in the dataset but only for the most frequent ones, which is not a big deal but is the only reason preventing the check to reach its maximum.

3.4. Clustering analysis

As already mentioned, the project contains some extra work that was helpful to evaluate the quality of the job; it has been selected a `clustering analysis` to uncover hidden patterns that go beyond the simple division in zones performed by the city administration. The usual performance indicator for clustering is the `silhouette coefficient`(from now on, SC) which measures the cohesion and coherency of the obtained clusters. The algorithm selected was `KMeans`, with $k = 9$, and it has been executed both on the dirty dataset and on the cleaned dataset using all the features. The work was succesful since the `SC` of clustered points increased from 0.5 to 0.6 when running the same algorithm with the same parameters and features over the dirty dataset first and then over the cleaned one. To further improve the analysis, it could be done both some `hyperparameter tuning` using for instance the `elbow method` to select an optimal k, together with some `feature selection` to reduce the dimension of the points to cluster, and some `feature engineering` like finding a suitable `embedding` for categorical columns to introduce more dense information rather than the usual sparse dummy encoding. In the end this analysis was still very satisfactory despite being a very basic one since it allowed to visualize clearly the impact of the work done, Figure 3.1.

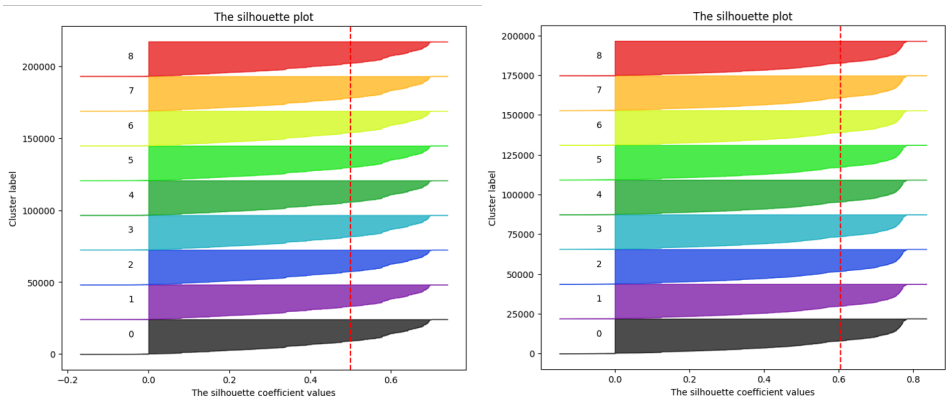

Figure 3.1: Comparison of silhouette plots. **Left**: silhouette plot on clusters obtained from the dirty dataset. **Right**: silhouette plot on clusters obtained the from cleaned dataset.

3.5. Conclusion

The quality of the data has been effectively improved due to rigorous following of the order of the pipeline. The `data wrangling` phase produced well-formatted values for each column, which proved beneficial for post-wrangling operations. This not only ensured good performance during `ML-based imputations`, but also eliminated any syntactic differences, allowing to focus only on identifying real `outliers` and simplifying the process of `duplicate detection`. Then, any try to fix the errors with the removal of `inconsistent values` and `outliers` has maximized the correctness of the tuples, while `data deduplication` and `column drops` has minimized any data redundancy without any significant loss in information. The dataset can finally be considered prepared for any future usage. Some suggestions can be, for instance, to use it either for the training of a model, after a step of `feature engineering`, or to create some dashboards for a `data visualization` activity.

Bibliography

[1] Local outlier factor. https://www.researchgate.net/publication/221214719_LOF_Identifying_Density-Based_Local_Outliers.

[2] Municipi di milano. https://it.wikipedia.org/wiki/Municipi_di_Milano. Accessed: 2025-01-09.

A | Appendix A

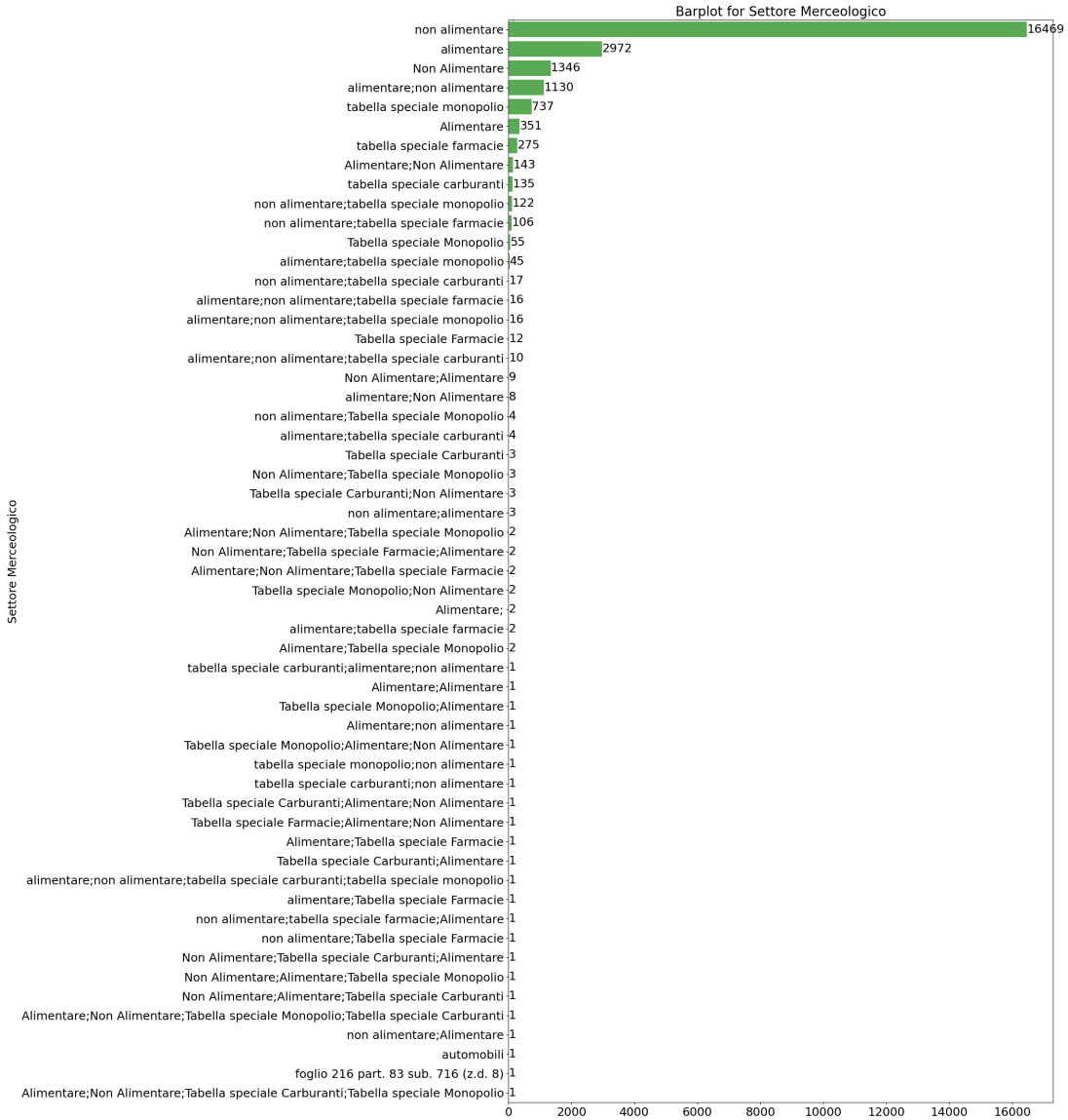

Figure A.1: Barplot of column `Settore Merceologico`

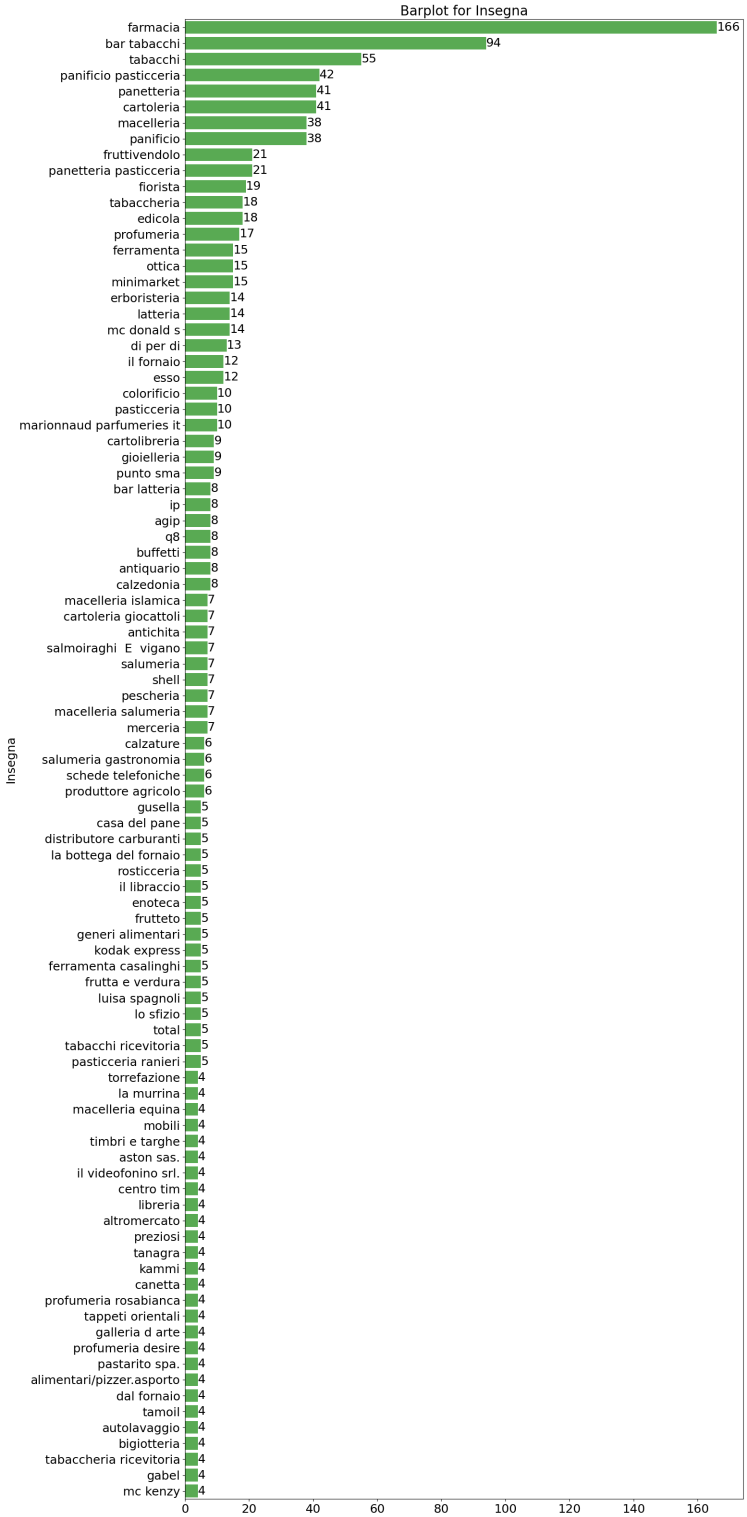

Figure A.2: Barplot of column `Insegna`

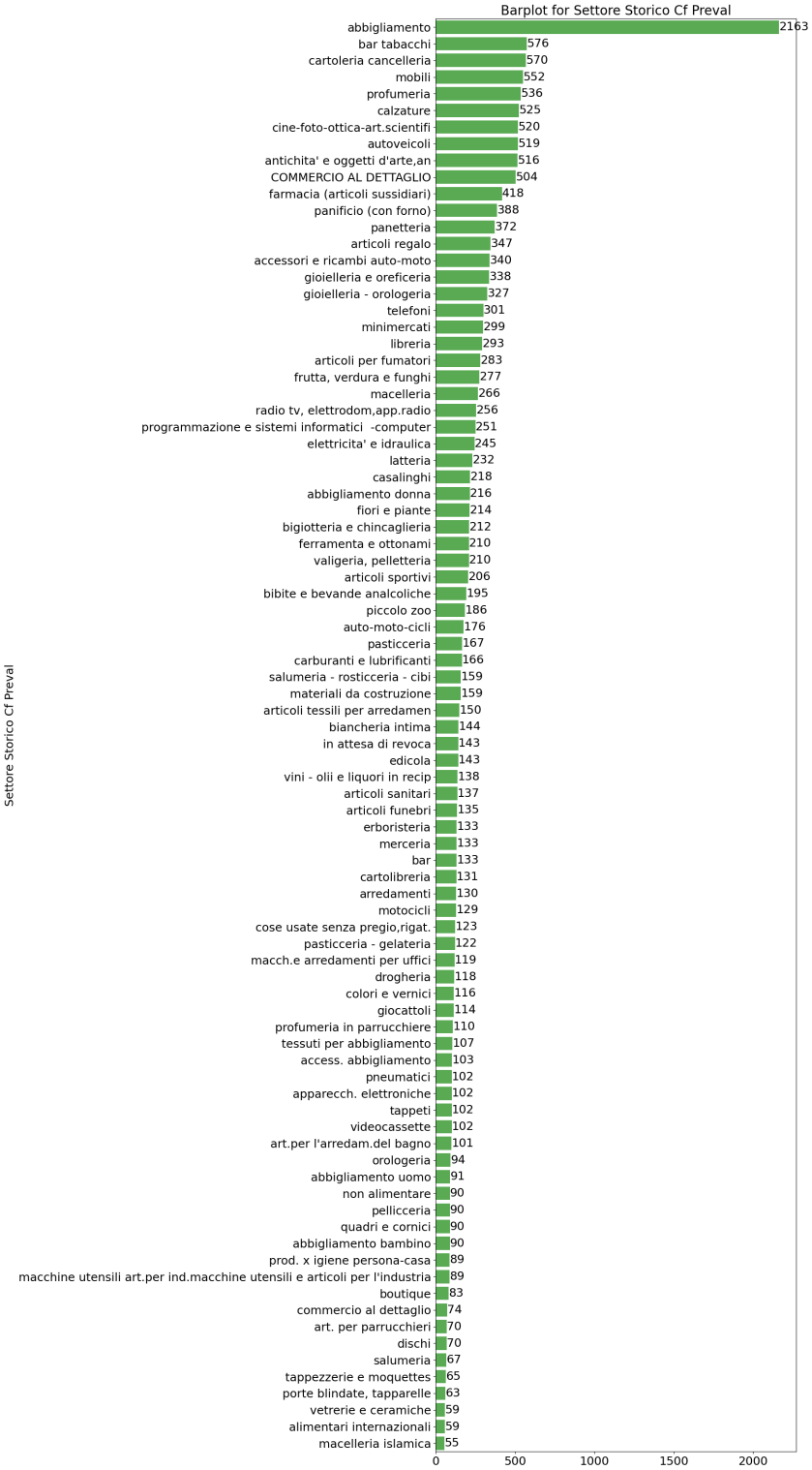

Figure A.3: Barplot of column `Settore Storico Cf Preval`

B | Appendix B

This section contains some code snippets of data processing pipeline.

B.1. Association rules function

```
def mine_association_rules(df,lhs,rhs,min_sup=0.5,min_conf=0.8,num_itemsets=2):

    """This function mines for association rules between arbitrary number of columns
    of a dataframe using fpgrowth algorithm to extract frequent itemsets."""

    #Put all items of each transactions into a list
    #df = df.dropna() # it makes everything explode apparently
    records = []
    for i in range(0,df.shape[0]):
        if type(lhs) == list:
            st1 = df.loc[i,lhs].str.cat(sep = ' ')
        else:
            st1 = str(df.loc[i,lhs])

        if type(rhs) == list:
            st2 = df.loc[i,rhs].str.cat(sep=' ')
        else:
            st2 = str(df.loc[i,rhs])

        records.append([(st1,0),(st2,1)])

    ### ASSOCIATION RULES MINING IN 3 STEPS ###

    # # 0 - generate an encoding of the dataset
    TE = TransactionEncoder()
    array = TE.fit(records).transform(records)
    array_df = pd.DataFrame(array,columns=TE.columns_)

    # 1 - generate the frequent itemset: set of transactions with support >= min_sup
    freq_itemset = fpgrowth(array_df, min_support=min_sup, use_colnames=True)
```

```
    freq_itemset['length'] = freq_itemset['itemsets'].apply(lambda x: len(x)) # additional column for length of itemset

    if not freq_itemset.empty:
        # 2 - generate rules: set of transactions having confidence >= min_conf
        rules = association_rules(freq_itemset, num_itemsets=num_itemsets, metric='confidence', min_threshold=min_conf)
        rules = rules[rules['lift']>1] # filter to keep only interesting rules

        # # clean up a bit the formats
        rules['antecedents'] = rules['antecedents'].apply(lambda x : list(x)[0])
        rules['consequents'] = rules['consequents'].apply(lambda x : list(x)[0])
        rules['antecedents'] = rules['antecedents'].apply(lambda x : tuple((str(lhs),x[0])) if x[1]==0 else tuple((str(rhs),x[0])))
        rules['consequents'] = rules['consequents'].apply(lambda x : tuple((str(lhs),x[0])) if x[1]==0 else tuple((str(rhs),x[0])))

        return pd.DataFrame([f"{x[0]} : {x[1]} => {y[0]} : {y[1]}" for x, y in rules[['antecedents','consequents']].values],columns=['rule'])
    else:
        return pd.DataFrame()
```

B.2. Wrangling of Ubicazione

```
# Create the temporary DataFrame
dft = pd.DataFrame()

# Extract Tipo via
dft['Tipo via tmp'] = df['Ubicazione'].str.extract('([A-Z]+ )',expand=True)
dft['Tipo via tmp'] = dft['Tipo via tmp'].str.strip() # Remove spaces at the begin and end of the string

# Extract Via
dft['Via tmp'] = df['Ubicazione'].str.extract(r'(?:^[A-Z]+ )([A-Z\.\'\s]+[A-MO-Z\']+[\.N]*)(?= N\.|num\.|civ\.|civico|\s?[a-z]|\s?\W\d)', expand=True)
dft['Via tmp'] = dft['Via tmp'].str.strip() # Remove spaces at the begin and end of the string

# Extract Civico
dft['Civico tmp'] = df['Ubicazione'].str.extract(r'(?:N\.|num\.|civ\.|civico)(\s?[\w/\\]+)', expand=True)
dft['Civico tmp'] = dft['Civico tmp'].str.strip() # Remove spaces at the begin and end of the string
dft['Civico tmp'] = dft['Civico tmp'].str.upper() # Put all characters to uppercase
```

```
dft['Civico tmp'] = dft['Civico tmp'].replace(r'^0+', '', regex=True) # Remove all the zeros at the beginning of the string
dft['Civico tmp'] = dft['Civico tmp'].str.replace(r'(\d)([a-zA-Z])', r'\1/\2', regex=True) # Add a / before letters to have uniform strings
dft.loc[dft['Civico tmp'] == '','Civico tmp'] = np.nan # If the string is empty, replace with NaN

# Extract ZD
dft['ZD tmp'] = df['Ubicazione'].str.extract('\(z.d. (\d)\)',expand=True)
dft['ZD tmp'] = dft['ZD tmp'].str.strip() # Remove spaces at the begin and end of the string

# Extract Accesso
dft['Accesso'] = df['Ubicazione'].str.extract(r'accesso:([^;]*)')
dft['Accesso'] = dft['Accesso'].str.strip() # Remove spaces at the begin and end of the string
dft['Accesso'] = dft['Accesso'].replace(r'^accesso ', '', regex=True) # Remove the word accesso from the beginning of the string

# Extract Isolato
dft['Isolato'] = df['Ubicazione'].str.extract(r'isolato:([^;]*)')
dft['Isolato'] = dft['Isolato'].str.strip() # Remove spaces at the begin and end of the string
dft['Isolato'] = dft['Isolato'].replace(r'^0+', '', regex=True) # Remove all the zeros at the beginning of the string

# Extract Website
dft['Website'] = df['Ubicazione'].str.extract(r'(www[\w\.-]+)', expand=True)
dft['Website'] = dft['Website'].str.strip() # Remove spaces at the begin and end of the string

# Extract Altre info
dft['Altre info'] = df['Ubicazione'].replace({
    r'\(z.d. (\d)\)': '',  # Remove ZD info
    r'accesso:([^;]*)': '', # Remove Accesso info
    r'isolato:([^;]*)': '',  # Remove Isolato info
    r'([A-Z\.\'\s]+[A-Z\']+)(?= N\.|num\.|civ\.|civico|\s?[a-z]|\s?\W\d)': '', # remove Via info
    r'(N\.|num\.|civ\.|civico)(\s?[\w/\\]+)': '', # remove Civico info
    r'(www[\w\.-]+)': ''  # Remove Website info
}, regex=True)
dft['Altre info'] = dft['Altre info'].str.strip() # Strip any extra spaces from the remaining string
```

```
dft['Altre info'] = dft['Altre info'].replace(r'^[; ]+|[; ]+$', '', regex=True)  # Remove leading/trailing ; and spaces
dft['Altre info'] = dft['Altre info'].replace(r';+', '', regex=True)  # Replace multiple semicolons with a single one

# Set the rows with empty Altre info as nan
dft.loc[dft['Altre info'] == '','Altre info'] = np.nan
```

B.3. Grubb's Test function

```
def grubbs_test(data,alpha=0.05):
    """It returns all the outliers detected by the Grubbs test. It perform two-sided test."""
    outliers = []
    outlier_indexes = []
    points = list(data)
    reject_null_hp = True
    while reject_null_hp:
        n = len(points)
        mean = np.mean(points)
        stddev = np.std(points)
        index = np.argmax(abs(points-mean))
        point = points[index] # extract the point with max absolute distance from the mean (candidate point)
        G = abs(point-mean)/stddev # compute the test statistic for the candidate point
        t_critical = t_student.ppf(1-alpha/(2*n),n-2) # compute the critical value of t-distribution at alpha/2N
        G_critical = ((n-1)/np.sqrt(n)*np.sqrt(t_critical**2/(n-2+t_critical**2))) # compute the threshold
        if G > G_critical:
            outliers.append(point)
            outlier_indexes.append(index)
            points.remove(points[index])
        else:
            reject_null_hp = False

    return pd.DataFrame(outliers, columns=['Outliers'], index=outlier_indexes)
```

List of Figures

List of Tables