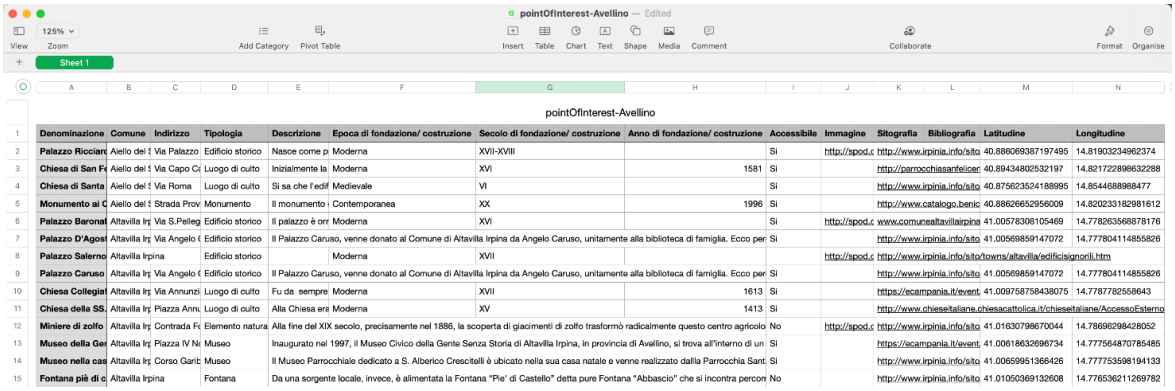
5 DATA PROFILLING

To get familiar with the data we had a look at the different datasets we were working with. The Point of Interests datasets of Avellino, Caserta, AgroNocerino, Montoro show a very similar structure in terms of columns and the associated data contained within them. Whereas the San Nicola dataset has a slightly different structure and contains just a few information about each point of interest.



pointOfInterest-Avellino

Denominazione	Comune	Indirizzo	Tipologia	Descrizione	Epoca di fondazione/ costruzione	Secolo di fondazione/ costruzione	Anno di fondazione/ costruzione	Accessibile	Immagine	Sitografia	Bibliografia	Latitudine	Longitudine
Palazzo Ricciar	Aiello del S	Via Palazzo	Edificio storico	Nasce come p	Moderna	XVII-XVIII		Si	http://spod.c	http://www.irpinia.info/sito		40.88606938197495	14.81903234962374
Chiesa di San Fe	Aiello del S	Via Capo Ca	Luogo di culto	Inizialmente la	Moderna	XVI	1581	Si		http://parrocchiesanfelicer		40.89434802532197	14.821722898632288
Chiesa di Santa	Aiello del S	Via Roma	Luogo di culto	Si sa che l'edif	Medievale	VI		Si		http://www.irpinia.info/sito		40.875623524188995	14.8544688988477
Monumento ai C	Aiello del S	Strada Prov	Monumento	Il monumento	Contemporanea	XX	1996	Si		http://www.catalogo.benic		40.88626652956009	14.820233182981612
Palazzo Baronal	Altavilla Irp	Via S.Pelleg	Edificio storico	Il palazzo è ori	Moderna	XVI		Si	http://spod.c	www.comunealtavillairpina		41.00578306105469	14.778263568878176
Palazzo D'Agost	Altavilla Irp	Via Angelo (	Edificio storico	Il Palazzo Caruso, venne donato al Comune di Altavilla Irpina da Angelo Caruso, unitamente alla biblioteca di famiglia. Ecco per				Si		http://www.irpinia.info/sito		41.00569859147072	14.777804114855826
Palazzo Salerno	Altavilla Irpina		Edificio storico		Moderna	XVII			http://spod.c	http://www.irpinia.info/sito/towns/altavilla/edificisignorili.htm			
Palazzo Caruso	Altavilla Irp	Via Angelo (	Edificio storico	Il Palazzo Caruso, venne donato al Comune di Altavilla Irpina da Angelo Caruso, unitamente alla biblioteca di famiglia. Ecco per				Si		http://www.irpinia.info/sito		41.00569859147072	14.777804114855826
Chiesa Collegiat	Altavilla Irp	Via Annunzi	Luogo di culto	Fu da sempre	Moderna	XVII	1613	Si		https://ecampania.it/event		41.009758758438075	14.7787782558643
Chiesa della SS.	Altavilla Irp	Piazza Annu	Luogo di culto	Alla Chiesa era	Moderna	XV	1413	Si		http://www.chieseitaliane.chiesacattolica.it/chieseitaliane/AccessoEsterno			
Miniere di zolfo	Altavilla Irp	Contrada Fo	Elemento natura	Alla fine del XIX secolo, precisamente nel 1886, la scoperta di giacimenti di zolfo trasformò radicalmente questo centro agricolo				No	http://spod.c	http://www.irpinia.info/sito		41.01630798670044	14.78696298428052
Museo della Ge	Altavilla Irp	Piazza IV N	Museo	Inaugurato nel 1997, il Museo Civico della Gente Senza Storia di Altavilla Irpina, in provincia di Avellino, si trova all'interno di un				Si		https://ecampania.it/event		41.00618632696734	14.777564870785485
Museo nella cas	Altavilla Irp	Corso Garib	Museo	Il Museo Parrocchiale dedicato a S. Alberico Crescitelli è ubicato nella sua casa natale e venne realizzato dalla Parrocchia Sant				Si		http://www.irpinia.info/sito		41.00659951366426	14.777753598194133
Fontana più di c	Altavilla Irpina		Fontana	Da una sorgente locale, invece, è alimentata la Fontana "Pie' di Castello" detta pure Fontana "Abbascio" che si incontra percor				No		http://www.irpinia.info/sito		41.01050369132608	14.776536211269782

Figure 7 Raw data provided

In order to analyse the data in a more systematic way we used pandas-profiling in order to perform the single-column and multi-columns analysis. But before starting to use “automatic tools”, we had a fine-grained look at the datasets looking for inconsistencies and anomalies in the data. The first thing we discovered was that a lot of fields contained the value “Informazione assente”, so we removed all those fields from the datasets.

The second thing we noticed was that the columns “Secolo di fondazione/ costruzione” and "Anno di fondazione/ costruzione" contained different formats to indicate the same Before Christ acronym: "a.c." and "A.C.". There was also another mismatch in the format for range periods, where some fields used the notation "startYear - endYear" and others used the notation "startYear-endYear". In this case we decided to remove all spaces in the fields and put the before christ acronym always in capital letters.

Another problem we discovered was that the "Geolocalizzazione" column contained both the latitude and the longitude coordinates in the same field. So, we decided to split it into two different columns. The final inconsistency we found was that the Montoro dataset contained the column "Frazione" whereas all other datasets contained the column "Comune" to indicate the town where the POI was located. After this manual preprocessing that we did for the datasets, we started to use pandas profiling.

5.1 Pandas Profiling

The first thing we noticed was that not all missing values were labeled with the string "Informazione assente": other labels, such as "informazione mancante" and "Nessuna informazione" were also used, so we had to clean the dataset again and re-do the analysis.

The second thing we saw from the report that was generated, is that the column "Epoca di fondazione/ costruzione" is highly correlated with the column "Secolo di fondazione/ costruzione" and the column "Tipologia". This was expected since the epoch of a point of interest can easily be determined by the century in which it was “founded” or built and the typology of it is highly affected by the trends of that period."Denominazione del Punto di Interesse" has a very high cardinality, which makes it a candidate primary key.

Another thing we discovered is that most frequent type of POI are "Luogo di culto" and "Edificio storico", with respectively 51,3% and 18,2% of the entire list of POI present in the datasets. The same goes for Epoca di fondazione, with the values:

- Moderna (32,2%)
- Medievale (20,1%)
- Contemporanea (9,7%)

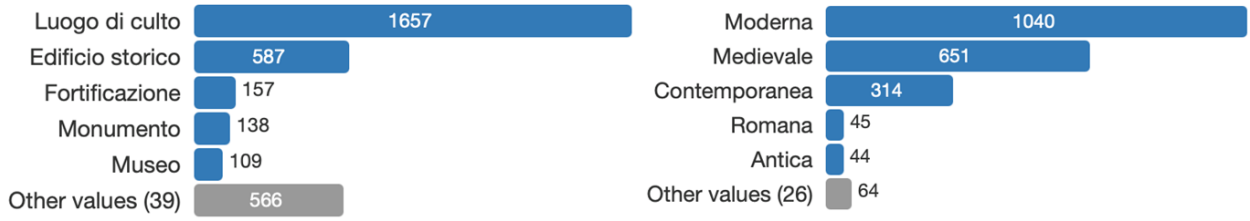


Figure 8 Data distribution types and epochs.

5.2 Profiling algorithms

We developed the following algorithms:

- Unique Column Combinations discovery
- Functional Dependency discovery
- Approximate Functional dependency discovery
- Unary Inclusion Dependency discovery

All these algorithms are based on 3 core principles:

1) **Position List Indices:**
   Split records of data into groups having the same value. They are useful to calculate the distinct count of a column of a dataset, and they can be calculated incrementally from single attribute to multiple attributes combinations.

2) **Prefix tree traversal of the Lattice:**
   The lattice structure enumerates all column combinations in a level wise order, from single attribute combinations to multi attributes combinations. The prefix tree traversal enables us to check each column combination just once.

3) **A-priori property:**

All of the algorithms follow the a-priori algorithm structure: it has a candidates step, where we generate all possible candidates for the current level, and a prune step, where we check each candidate. Since the prune operation is the most expensive one, the algorithm takes advantage of the a-priori property to avoid checking candidates that we know a-priori cannot satisfy our need.

An important aspect to keep in mind is that all of these algorithms find UCCs, FDs and INDs for **specific** dataset instances; All of these features only hold for the dataset they were calculated from, they are not guaranteed to generalize for any new data coming to the dataset. So, to establish Primary Keys, Functional Dependencies and Foreign Keys, a human check is still needed.

UCC discovery

The first algorithm we developed was the Unique Column Combination discovery (UCC discovery) following the HCA algorithm. In order to get all of the UCCs we only need to discover the minimal ones, since all others can be derived from generalization.

We detect UCCs by checking the condition that the stripped PLIs for a given column combination is empty. The algorithm is built upon the a-priori level-wise search on prefix tree. It exploits the a-priori property in order to avoid the checks for column combinations we know a-priori cannot be UCC, due to the fact that UCCs are upward closed and non-UCCs are downward closed.

- If X is a UCC, then XUA is a UCC (UCCs are upward closed)
- If X is not a UCC, then any subset of X cannot be UCCs (non-UCCs are downward closed)

The code for the UCC discovery can be found in "UCC_discovery.py". In order to run it, copy and paste the following commands:

```
cd data-profiling

conda env create -f environment.yml

conda activate data-profiling

Python3 UCC_discovery.py --dataset=satellites.csv

Python3 UCC_discovery.py --dataset=abalone.csv
```

```
(data-profiling) gianluigi@Gianluigis-MacBook-Pro data-profiling % python3 UCC_discovery.py --dataset=abalone.csv

abalone.csv
      0      1      2      3       4       5       6       7   8
0     M  0.455  0.365  0.095   0.514  0.2245   0.101    0.15  15
1     M   0.35  0.265   0.09  0.2255  0.0995  0.0485    0.07   7
2     F   0.53   0.42  0.135   0.677  0.2565  0.1415    0.21   9
3     M   0.44  0.365  0.125   0.516  0.2155   0.114   0.155  10
4     I   0.33  0.255   0.08   0.205  0.0895  0.0395   0.055   7
...  ..    ...    ...    ...     ...     ...     ...     ...  ..
4172  F  0.565   0.45  0.165   0.887    0.37   0.239   0.249  11
4173  M   0.59   0.44  0.135   0.966   0.439  0.2145  0.2605  10
4174  M    0.6  0.475  0.205   1.176  0.5255  0.2875   0.308   9
4175  F  0.625  0.485   0.15  1.0945   0.531   0.261   0.296  10
4176  M   0.71  0.555  0.195  1.9485  0.9455  0.3765   0.495  12

[4177 rows x 9 columns]
9
UCCs found:
[[4, 5, 7], [4, 6, 7], [0, 1, 4, 5], [0, 4, 6, 8], [1, 2, 4, 5], [1, 2, 4, 7], [1, 3, 4, 5], [1, 3, 4, 6], [1, 4,
5, 6], [1, 4, 6, 8], [1, 5, 6, 8], [2, 3, 4, 5], [2, 4, 5, 8], [2, 4, 6, 8], [3, 4, 5, 6], [3, 5, 6, 7], [3, 5, 6,
 8], [4, 5, 6, 8], [0, 1, 3, 5, 6], [0, 1, 6, 7, 8], [0, 2, 3, 5, 6], [1, 2, 3, 4, 8], [1, 2, 3, 6, 8], [1, 2, 6,
7, 8], [2, 3, 4, 7, 8], [0, 1, 2, 3, 5, 7], [0, 1, 2, 3, 5, 8], [0, 1, 2, 3, 6, 7], [1, 2, 3, 5, 7, 8]]
(data-profiling) gianluigi@Gianluigis-MacBook-Pro data-profiling %
```

Figure 9 UCC Discovery output.

FD discovery

The second algorithm we developed was the Functional Dependency discovery (FD discovery) following the TANE algorithm. We only need to find the minimal FDs, since all others can be derived from generalization.

We detect FDs by checking the condition that X implies Y if and only if the number of PLIs(X) is equal to the number of PLIs(XY). The algorithm is built upon the a-priori level-wise search on prefix tree. It exploits the a-priori property in order to avoid the checks for pairs of column combinations we know a-priori cannot be FDs.

The main component of the FD discovery algorithm are Candidate sets. C(X) is the list of RHS for all the possible FD having X as LHS. TANE defines 3 rules in order to obtain the improved candidate set C+(X), which exploits the a-priori property in order to skip the candidates we know a-priori that cannot be FDs.

Rule C1: we remove from the candidate set all columns that can be derived from other columns of X

Rule C2: if there is a column that can be derived from other columns in X, then we remove all R\X columns

Rule C3: the candidate set for a new candidate X gets initialised as the intersection between all candidate sets of the subsets of X (with just one element less)

The algorithm also takes advantage of Key Pruning. Given that a Primary Key functionally determines all other columns (by definition), if we find a key we generate all FDs with the key as LHS, and then prune the key from the candidates generation.

The code for the FD discovery can be found in "FD_discovery.py". In order to run it, copy and paste the following commands:

```
cd data-profiling

Conda env create -f environment.yml

Conda activate data-profiling

Python3 FD_discovery.py --dataset=breast-cancer-wisconsin.csv

Python3 FD_discovery.py --dataset=satellites.csv

Python3 FD_discovery.py --dataset=abalone.csv
```

The results can be checked with those available in the official Hasso-Plattner-Institut reference: https://hpi.de/naumann/projects/repeatability/data-profiling/fds.html

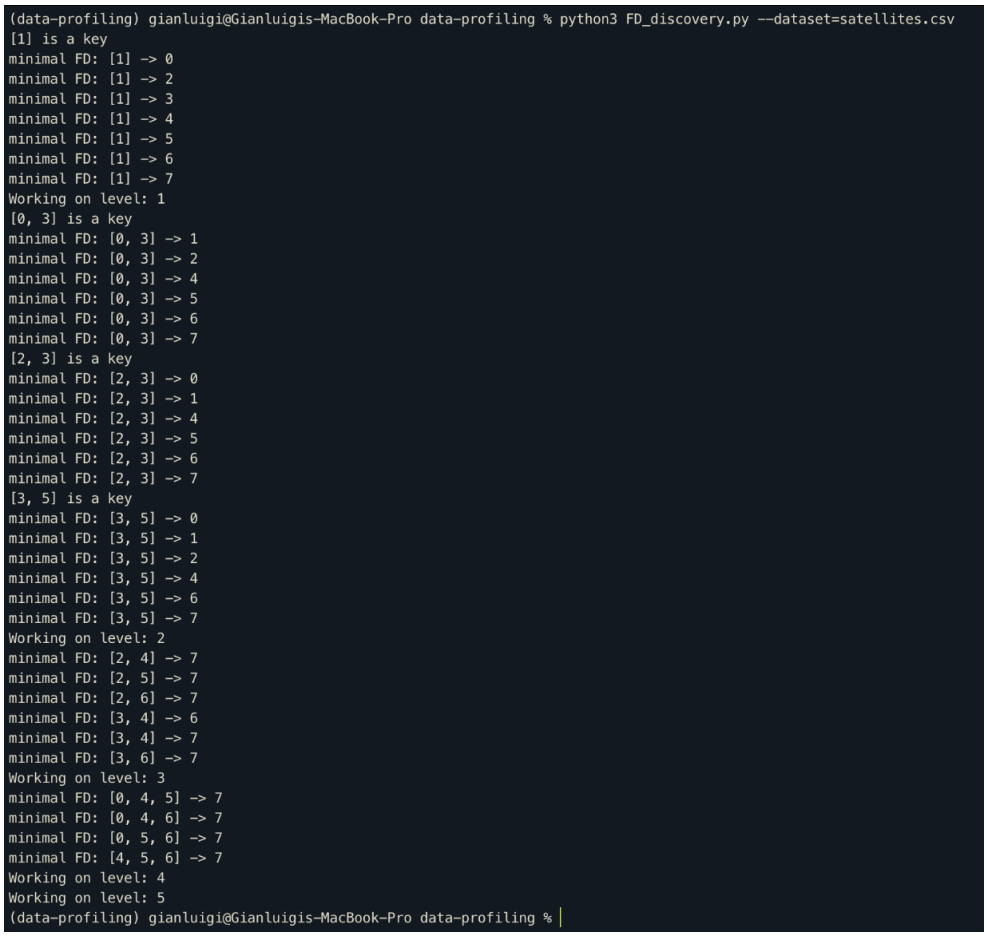


Figure 10 FD Discovery output.

We also developed the algorithm for Approximate FD discovery. We added a s_min property to the a-priori function which determines the amount of records the algorithm can omit while

checking for an FD. In the sample code provided we have set the minimum support to 10% of the data, meaning that if the FD holds for 90% of the records, or more, then it is recognised as a valid FD. Even though we relaxed the condition on the FD discovery, we cannot use the same rule for pruning, because for that case the FD must hold exactly.

The code for the approximate FD discovery can be found in "FD_approximate_discovery.py". In order to run it, copy and paste the following commands:

```
cd data-profiling

conda env create -f environment.yml

conda activate data-profiling

Python3 FD_approximate_discovery.py --dataset=breast-cancer-wisconsin.csv

Python3 FD_approximate_discovery.py --dataset=satellites.csv

Python3 FD_approximate_discovery.py --dataset=abalone.csv
```

IND discovery

The last algorithm we developed was Unary Inclusion Dependency discovery (IND discovery) following the MIND algorithm. We only need to find the minimal INDs, since all others can be derived from generalization.

We detect INDs by checking the condition that every value in X also appears in Y.

We only check columns of the same datatype since incompatible datatypes cannot be included in one another.

The algorithm generates a list of columns of the same datatype and a list of all values contained in these columns; it then builds a binary relationship between the columns and values, making a pair if a certain value appears in a certain column. Eventually, it makes an inverted index of the binary relationship and scan it to look for INDs.

The code for the unary IND discovery can be found in "unary_IND_discovery.py". In order to run it, copy and paste the following commands:

```
cd data-profiling

Conda env create -f environment.yml

Conda activate data-profiling

Python3 unary_IND_discovery.py
```

The example available in the source code was taken from the slides of the course for comparison purposes, precisely in slide 109.