Università degli Studi di Salerno

Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica

Software Engineering for AI project report

Data Smell Detection+

Califano Adriano Emanuele
a.califano88@studenti.unisa.it

D'Antuono Francesco Paolo
f.dantuono10@studenti.unisa.it

Fabiano Daniele
d.fabiano3@studenti.unisa.it

Academic Year: 2023-2024

Contents

CHAPTER 1

Introduction

1.1 Background

Some of the recent studies have focused their research on **Data Smells**, which are defined as *"signals or clues about the presence of latent problems or anomalies in data"*. Their study requires careful and detailed analysis.
For this reason, several tools that enable smell detection have been developed, one of them is **DSD (Data Smell Detection)**[1]. It is based on the open-source data validation tool **Great Expectations**[2] and it uses rule-based techniques for detecting smells by analyzing a dataset. **DSD+**[1] aims to extend the existing smell detection suite, and it is also intended to improve the current dataset reporting mechanism by integrating the calculation of data quality dimensions metrics, along with additional measurements.
In addition, it will be possible to visualize the results of the tool's execution, to track changes obtained over time as a result of new analyses and to form a basis for indications of quality improvement/deterioration of the modified dataset.

1.1.1 Data Quality Dimensions

Defined as *a characteristic or part of the information for classifying information and data requirements*[3], a **Data Quality Dimension (DQD)** offers a way for measuring and managing data quality. From a research perspective, several DQDs have been identified due to the various information data can store and exhibit.
The International Data Management Association (DAMA) established a set of 6 principal DQDs[4]: **consistency**, **uniqueness**, **timeliness**, **accuracy**, **validity** and **completeness**.
As part of this project, a subset of these DQDs will be implemented to provide more insight into the datasets.

[1] `https://github.com/CpDant/DSD-plus`

1.2 Goals of the Project

The main goals for this project are:

- **Extension of the data smell detection suite**: At least one new data smell detector should be implemented to extend the detection suite that makes the tool available.
- **Mechanism for calculating data quality dimension metrics**: Related to creating a mechanism for calculating a subset of data quality dimensions referring to the results obtained from the tool's analysis done on the specific dataset.
- **Improved reporting system**: Add the ability to display on the screen the results obtained from the analysis of a dataset and show over time the change in the dataset, related smells identified, and metrics/measures calculated.

CHAPTER 2

Methodology

This section will explain in more detailed parts what the methodological steps are. The first part will explain the current system, including its features and problems, with a focus on the changes that will be implemented. Then we will introduce DQDs and Data Smell that will be implemented. Lastly, a brief description of the data that will be used for the validation and evaluation of the changes.

2.1 Analysis of the Existing System

DSD (Data Smell Detection) is a tool developed to enable the identification of data smells within datasets. Its functioning is based on the open-source library called **Great Expectations** that has as its main goal the possibility to create, test, and document data quality.
Its major features are the possibility to register or login to the system and the ability to upload and scan datasets in search of data smell with several levels of accuracy and sensitivity.
The following sections will thoroughly explain each system's functionality. Major details on the system analysis are documented for the IGES course.

2.1.1 Data Smell Detection

The main functionality of the system is the detection of data smells, and it consists of the following main steps:

1. **Upload dataset**: In the first step, after choosing the dataset to analyze, it must be inserted into the system. Being the main functionality, it is accessible from the home page.
2. **Customization**: The second step concerns the customization of the detection. In it, it is possible to choose the various types of data smells to be identified within the dataset. There will be a page with checkboxes that allow this selection. Furthermore, there is the possibility of using two different detection modes:
   - **Easy mode**: Simply allows for detection with presetting of three types that provide how the detection will be carried out.
   - **Expert mode**: Allows you to choose manually how you want the detection to be carried out, and therefore modify the values of specific parameters that define how the detection is done.
3. **Results**: After confirming the chosen customization type, you can proceed with the final step regarding the visualization of the results obtained from the detection performed on the dataset. In this phase, the system allows you to view the columns of the dataset, also allowing you to customize the display with the choice of the column.

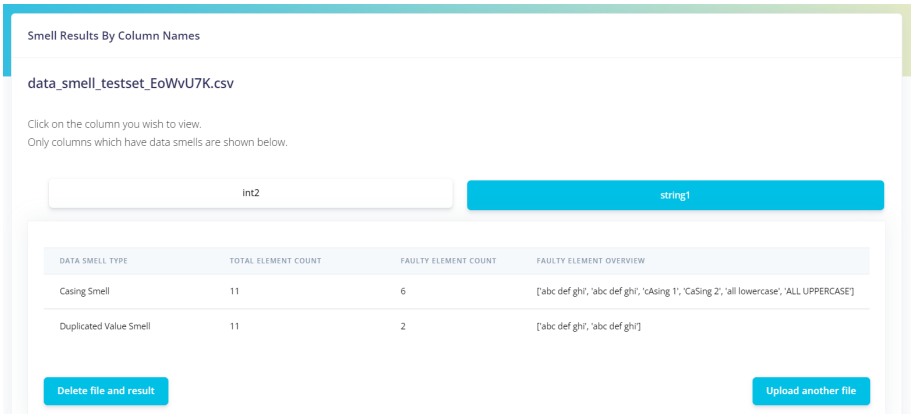

Figure 2.1: Results section after dataset analysis

2.1.2 Reporting System

Once the detection is completed, you can view the results obtained in the relevant *"Saved Results"* section, which is only available to registered users. This section shows the columns where the smells have been identified. You can view a table broken down by rows concerning the identified smells and show the following columns:

- **Data Smell Type**, related to the data smell type identified.
- **Total Element Count**, related to the number of elements that have been scanned.
- **Faulty Element Count**, related to the items in the dataset that are affected by the smell.
- **Faulty Element Overview**, a list of the values that are affected by the smell.

Finally, it is possible to delete the results or start another analysis by uploading a new dataset.

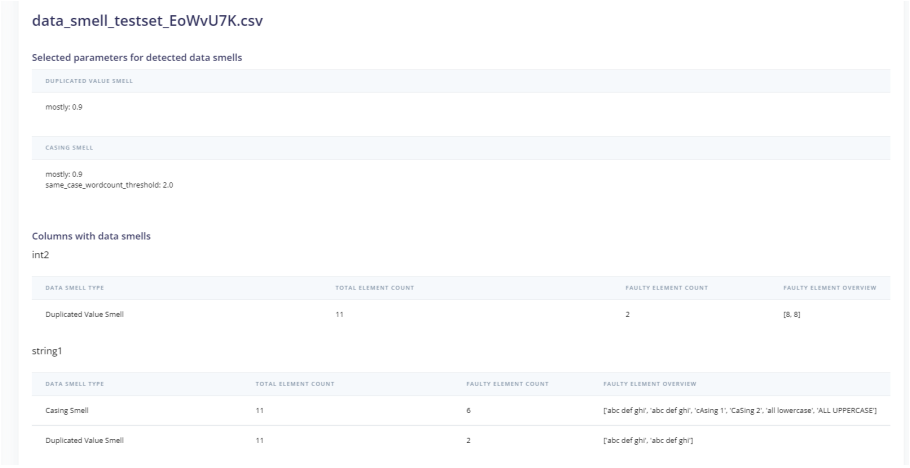

Figure 2.2: Example of results shown in the results page

2.2 Change Requests

The changes that will be made to the system mainly concern the reporting system and the addition of DQD calculations. As also shown in the system analysis, there is a misalignment between the results immediately visible after the analysis and those displayed on the analysis history page. From a usability standpoint, it is much more convenient to maintain the structure of the initial results display. However, it is important to express the following critical issues:

- **The same dataset analyzed multiple times will generate different records**.
- **The visualization of values affected by smell may be compromised when considering cases where there are many analyzed values**.

Furthermore, it is necessary to integrate an overview of the calculated metrics and, if possible, the corresponding graphs for a better analysis of the current reporting system.
All these changes and improvements are shown in Figure **2.3**.

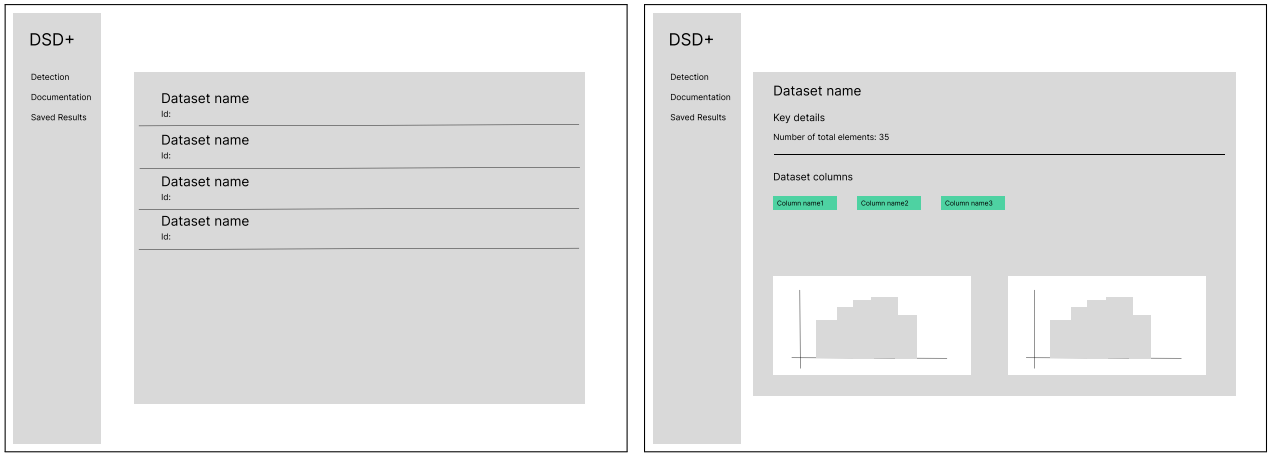

Figure 2.3: Results page and dataset info mockup

2.3 Choice of DQDs and Data Smells

In the following section, we will go into detail about the metrics that will be used to validate the datasets. Furthermore, we also describe the new smells that will be added to the set of data smell detection already present.

2.3.1 Data Quality Dimensions

As previously explained in Section 1.1.1, a Data Quality Dimension offers a way for measuring and managing data quality. In detail, there are 6 data quality dimensions identified by Ramasamy et al. [4]:

- **Completeness**: The proportion of stored data against the potential of "100% complete".
- **Uniqueness**: Nothing will be recorded more than once based upon how that thing is identified
- **Timeliness**: The degree to which data represent reality from the required point in time.
- **Validity**: Data are valid if they conform to the syntax (format, type, range) of their definition.
- **Accuracy**: The degree to which data correctly describes the "real world" object or event being described.
- **Consistency**: The absence of difference when comparing two or more representations of a thing against a definition.

Based on the studies carried out by Elouataoui et al. [5] and Recupito et al. [6] and relying on the analyses performed on the tool and the type of detection that is carried out, the following metrics have been chosen to be calculated on the datasets:

- **Completeness**: Calculated using the results of the implemented **Missing Value** smell:

$$\frac{Number\_of\_non\_empty\_values}{(Total\_values)} * 100$$

- **Uniqueness**: Calculated using the results of the implemented **Duplicated Value** smell:

$$\frac{Number\_of\_unique\_rows}{(Total\_rows)} * 100$$

- **Validity**: Calculated using the results of three data smells, in particular the number of *valid_values* that are not detected as non-valid by **Integer as String**, **Floating-Point Number as String** and **Integer as Floating-Point Number** smells. More generalized:

$$\frac{Number\_of\_valid\_values}{(Total\_values)} * 100$$

2.3.2 Data Smells

Data smells are signals or hints of the presence of underlying issues or anomalies in the data, that require careful and detailed analysis. As shown in [1], there are various categories of data smells:

- **Believability Smells**: Encompass issues related to the credibility and trustworthiness of data.
- **Encoding Understandability Smells**: Highlight problems like representing integers as strings or vice versa, which can lead to confusion during data processing.
- **Syntactic Understability Smells**: Cover issues such as special characters, spacing inconsistencies, and data values that are too long to understand.
- **Consistency Smells**: Refers to the uniformity and consistency of data elements.

From a more technical view, DSD's smells are built either with **rule-based** techniques [7] or with **machine learning models** [8]. After an analysis phase on both implementations, we summarized implemented/unimplemented smells in figure **2.4**, where the light blue cells are rule-based smells, while the green ones are with ML models.

Believability Smells	Encoding Unserstandability Smells	Syntactic Understandability Smells	Consistency Smells
Dummy Value Smell	Date as DateTime	Ambiguous Date/Time Format	Abbreviation Inconsistency
Duplicated Value Smell	Date as String	Ambiguous Value	Casing Inconsistency
Extreme Value Smell	Date/Time as String	Casing	Class Inconsistency
Meaningless Value Smell	Floating-point Number as String	Contracting	Date/Time Format Inconsistency
Misspelling Smell	Integer as Floating-point Number	Extraneous Value	Missing Value Inconsistency
Suspect Class Value Smell	Integer as String	Intermingled Data Type	Separating Inconsistency
Suspect Date Value Smell	Suspect Character Encoding	Long Data Value	Spacing Inconsistency
Suspect Date/Time Interval Smell	Time as String	Missing Value	Special Character Inconsistency
Suspect Distribution Smell		Separating	Syntax Inconsistency
Suspect Sign Smell		Spacing	Transposition Inconsistency
		Special Character	Unit Inconsistency
		Synonym	
		Small Number	
		Tagging	

Figure 2.4: Implemented/Unimplemented smells in DSD

All implemented data smells are documented within the tool itself, accessible through a dedicated section in the web application labeled "Documentation."
Recent studies [6], have demonstrated the existence of new categories of smells in addition to those already existing, which are:

- **Redundant Value Smells**: Point to data elements or features that provide little to no additional information for the training of AI models.
- **Distribution Smells**: Related to the values in terms of the whole range of values represented.
- **Miscellaneous Smells**: Covers various data quality issues that do not fit into the previous categories.

After a careful analysis of the tool and the approach used for inserting a new detection of data smell (which we recall to be rule-based, with Great Expectations), the following data smells that have not yet been implemented in the tool have been identified:

- **Spacing**: This smell arises when data values contain an uncommon pattern of spaces (Trailing Space, Leading Spaces, Multiple Spaces, Missing Spaces). Example: “John Smith “, " John Smith”, ” John Smith “, "JohnSmith"
- **Special Character**: This smell occurs when data values contain special characters (non-alphanumeric) like Commas, Dots, Hyphens, Apostrophes, Tab Char, Punctuation, Parentheses, Dashes, accented Letters, etc. Example: "Hello, how are you - today", "New.York.", "John’s house"

2.3.3 New Data Smells Implementation

To implement the new data smells, we will use Great Expectations Library, that uses a rule-based approach. In particular, in our case, we will use the so called "**Custom Regex Based Column Map Expectation**" template that uses the regular expressions to determine the rules for detecting the new data smells. In particular, we have:

- **Spacing**: For this data smell, we have used the following regex `^\s|\s\s+|\s$`. This regex let us detect the smell when the space character is at the beginning of the word, at the end of the word or there are two or more consecutive spaces.

```
class ExpectColumnValuesToNotContainSpacingSmell(ColumnMapExpectation, DataSmell):

    data_smell_metadata = DataSmellMetadata(
        data_smell_type=DataSmellType.SPACING_SMELL,
        profiler_data_types={ProfilerDataType.STRING}
    )

    default_kwarg_values = {
        "catch_exceptions": True,
        "regex": r'^\s|\s\s+|\s$',
        "mostly": 1
    }
```

Figure 2.5: Code snippet for the Spacing Smell detection

- **Special Character**: For this data smell, we have used the following regex `[^a-zA-Z0-9\s]`. This regex let us detect the smell when the word does not contain alphanumerical characters.

```
class ExpectColumnValuesToNotContainSpecialCharacterSmell(ColumnMapExpectation, DataSmell):

    data_smell_metadata = DataSmellMetadata(
        data_smell_type=DataSmellType.SPECIAL_CHARACTER_SMELL,
        profiler_data_types={ProfilerDataType.STRING}
    )

     default_kwarg_values = {
        "catch_exceptions": True,
        "regex": r'[^a-zA-Z0-9\s]',
        "mostly": 1
    }
```

Figure 2.6: Code snippet for the Special Character Smell detection

2.4 Data Collection

In this phase, the initial step is to collect data that will be used for the analysis. Our goal is to test all the improvements and new metrics, so a targeted choice for the datasets fits the best to our purpose. In particular, datasets will be chosen to test the two new smells that will be implemented, and then some datasets for testing data quality metrics.
Since our improvements are on an existing project we will use the same data of the previous version of the tool (**60** datasets, which cover most application domains), using a small subset of datasets chosen from the table **2.1**.

Table 2.1: Dataset Metadata

Reference	Owner	Creator	Nr. files
manjeetsingh/retaildataset	Manjeet Singh	Manjeet Singh	3
nicapotato/womens-ecommerce-clothing-reviews	nicapotato	nicapotato	1
kaggle/sf-salaries	Kaggle	Paul	1
shivamb/real-or-fake-fake-jobposting-prediction	Shivam Bansal	Shivam Bansal	1
new-york-city/nyc-property-sales	City of New York	Aleksey Bilogur	1
utkarshxy/who-worldhealth-statistics-2020-complete	Zeus	Zeus	39
rtatman/lego-database	Rachael Tatman	Rachael Tatman	8
AnalyzeBoston/crimes-in-boston	Analyze Boston	rgriffin	2
grosvenpaul/family-income-and-expenditure	Francis Paul Flores	Francis Paul Flores	1
crowdflower/twitter-user-gender-classification	Figure Eight	Ed King	1
new-york-city/nyc-inspections	City of New York	Jacob Boysen	1
tanmoyx/covid19-patient-precondition-dataset	Tanmoy Mukherjee	Tanmoy Mukherjee	1
sohier/allenunger-global-commodity-prices	Sohier Dane	Sohier Dane	1
gpreda/chinese-mnist	Gabriel Preda	Gabriel Preda	1
navinmundhra/daily-power-generation-in-india-20172020	Navin Mundhra	Navin Mundhra	3
lalish99/covid19-mx	Eduardo Rojas	Eduardo Rojas	11
garystafford/environmental-sensor-data-132k	Gary A. Stafford	Gary A. Stafford	1
carlolepelaars/toy-dataset	Carlo Lepelaars	Carlo Lepelaars	1
usaf/world-war-ii	United States Air Force	Abigail Larion	1
hacker-news/hacker-news-posts	Hacker News	Anthony Goldbloom	1
gdaley/hkracing	Graham Daley	Timo Bozsolik	2
kaggle/hillary-clinton-emails	Kaggle	Timo Bozsolik	4
faa/wildlife-strikes	Federal Aviation Administration	Abigail Larion	1
brunotly/foreign-exchange-rates-per-dollar-20002019	Bruno Ferreira	Bruno Ferreira	1
momanyc/museum-collection	The Museum of Modern Art	Abigail Larion	2
kevinarvai/clinvar-conflicting	Kevin Arvai	Kevin Arvai	1
casimian2000/aws-honeypot-attack-data	casimian2000	casimian2000	1
imls/museum-directory	Institute of Museum and Library Services	Abigail Larion	1
fda/adverse-food-events	Food and Drug Administration	Jacob Boysen	1
zusmani/mygenome	Zeeshan-ul-hassan Usmani	Zeeshan-ul-hassan Usmani	7
usfundamentals/us-stocks-fundamentals	usfundamentals	Timo Bozsolik	2
miker400/washington-state-home-mortgage-hdma2016	Miker400	Miker400	1
cityofLA/los-angeles-traffic-collision-data	City of Los Angeles	Kaggle Team	1
fizzbuzz/freesound-prediction-file	Zafar	Zafar	10

Continued on next page

Table 2.1 – continued from previous page

Reference	Owner	Creator	Nr. files
theworldbank/world-banks-major-contracts	World Bank	Sohier Dane	1
chicago/chicago-food-inspections	City of Chicago	Kaggle Team	1
new-york-city/ny-bus-breakdown-and-delays	City of New York	Kaggle Team	1
adamschroeder/crimes-new-york-city	def love(x):	def love(x):	3
gustavomodelli/monthly-salary-of-public-worker-in-brazil	Luís Gustavo Modelli	Luís Gustavo Modelli	1
new-york-city/ny-2015-street-tree-census-tree-data	City of New York	Kaggle Team	1
mariopasquato/star-cluster-simulations	Mario Pasquato	Mario Pasquato	19
us-drought-monitor/united-states-droughts-by-county	United States Drought Monitor	Timo Bozsolik	2
rtatman/trademark-application	Rachael Tatman	Rachael Tatman	1
city-of-seattle/seattle-road-weather-information-stations	City of Seattle	Kaggle Team	1
jboysen/austin-waste	Jacob Boysen	Jacob Boysen	1
arashnic/covid19-case-surveillance-public-use-dataset	Möbius	Möbius	1
fcc/robocall-complaints	Federal Communications Commission	Sohier Dane	1
jboysen/sf-street-trees	Jacob Boysen	Jacob Boysen	1
metmuseum/the-metropolitan-museum-of-art-open-access	The Metropolitan Museum of Art	Myles O'Neill	1
therohk/ireland-historical-news	Rohit Kulkarni	Rohit Kulkarni	2
gomes555/road-transport-brazil	Fellipe Gomes	Fellipe Gomes	1
new-york-city/nyc-registration-contacts	City of New York	Noah Daniels	1
chicago/chi-restaurant-inspections	City of Chicago	Jacob Boysen	1
new-york-state/nys-salary-information-for-the-public-sector	State of New York	Kaggle Team	4
sohier/large-purchases-by-the-state-of-ca	Sohier Dane	Sohier Dane	1
nypl/whats-on-the-menu	New York Public Library	Timo Bozsolik	4
bls/producer-price-index	US Bureau of Labor Statistics	Aleksey Bilogur	76
chicago/chicago-red-light-and-speed-camera-data	City of Chicago	Kaggle Team	4
chicago/chicago-business-licenses-and-owners	City of Chicago	Kaggle Team	2
new-york-state/nys-spill-incidents	State of New York	Kaggle Team	1

2.5 Detector Testing

In this section, we want to explain the testing procedure used for the new smells. Each data smell was tested using the classical testing procedure suggested by the Great Expectation library[1].

2.5.1 Input Data

Data smells' input data domain was split in order to be able to test the regexes as fully and functionally as possible. Each subset of data is defined as follows:

2.5.1.1 Spacing Smell

1. **begin_space**: A list of elements in which the first element contains a value beginning with a single space.
2. **multiple_begin_space**: A list of elements in which the first element contains a value beginning with some spaces.
3. **inner_space**: A list of elements in which the first element contains a value with more than one consecutive space among two words.
4. **end_space**: A list of elements in which the first element contains a value ending with a single space.
5. **multiple_end_space**: A list of elements in which the first element contains a value ending with some spaces.
6. **no_spacing_smell**: A list of all correct elements.

```
"data": {
        "begin_space": [" test", "test", "test"],
        "multiple_begin_space": ["      test", "test", "test"],
        "inner_space": ["test    test", "test", "test"],
        "end_space": ["test ", "test", "test"],
        "multiple_end_space": ["test      ", "test", "test"],
        "no_spacing_smell": ["test", "test test", "test"],
}
```

Figure 2.7: Spacing smell test data

[1] https://docs.greatexpectations.io/docs/oss/guides/expectations/creating_custom_expectations/how_to_create_custom_regex_based_column_map_expectations

2.5.1.2 Special Character Smell

1. **punctuation_special_character**: A list of values containing one punctuation character per element.
2. **stressed_letter_special_character**: A list of values containing one stressed character per element.
3. **no_special_character_smell**: A list of all correct elements.

```
"data": {
        "punctuation_special_character": ["te§t", "t&st", "¿test?", "test ¶", "˜test"],
        "stressed_letter_special_character": ["tæst", "tëst", "tÉst", "test Å", "çtest"],
        "no_special_character_smell": ["22", "hello", "hello22", "HELLO", "HELLO 22"]
}
```

Figure 2.8: Special Character smell test data

2.5.2 Test Suite Definition

According to the documentation, each smell has a particular test suite in order to cover all the possible combinations of arising smells. Given the input data values defined earlier, the test definition followed the same pattern for each test. Following the definition of each smell, a test is considered to pass if it matches the expected output, otherwise it will fail. The expected results for each test case are defined as follows:

- **Expected: False**, if test data contains the smell.
- **Expected: True**, if test data does not contain the smell.

```
"tests": [
            {
                "title": "begin_space_test",
                "exact_match_out": False,
                "include_in_gallery": True,
                "in": {"column": "begin_space", "mostly": 1},
                "out": {"success": False}
            },

            {
                "title": "punctuation_special_character_test",
                "exact_match_out": False,
                "include_in_gallery": True,
                "in": {"column": "punctuation_special_character", "mostly": 1},
                "out": {"success": False}
            }
]
```

Figure 2.9: Test cases examples

2.6 DQDs Testing

The testing phase for the 3 DQDs was executed following these steps:

1. Generate randomly 3 datasets of 3 columns and 30 rows.
2. Modify some values in the dataset by inserting duplicates, removing some values and changing the type of numeric values.
3. Evaluate the metrics calculation and see if they match with the oracle.

Each of the 3 datasets was duplicated and adapted for the specific metric that had to be evaluated. In order to have a correct evaluation, the oracles for each dataset are the ones shown in table **2.2**.

		Columns		
Dataset	**Metric to evaluate**	**first_name**	**last_name**	**street_number**
Dataset #1	Completeness Uniqueness Validity	5 empty 4 duplicates all valid	9 empty 6 duplicates all valid	10 empty 10 duplicates 7 non-valid

		Columns		
Dataset	**Metric to evaluate**	**car_maker**	**color**	**model_year**
Dataset #2	Completeness Uniqueness Validity	10 empty 20 duplicates all valid	10 empty 22 duplicates all valid	10 empty 19 duplicates 5 non-valid

		Columns		
Dataset	**Metric to evaluate**	**ssn**	**currency**	**number**
Dataset #3	Completeness Uniqueness Validity	5 empty 12 duplicates all valid	7 empty 19 duplicates all valid	5 empty 16 duplicates 1 non-valid

Table 2.2: Datasets informations

CHAPTER 3

Results and Findings

3.1 DQDs Analysis

3.1.1 Implementation

The DQDs were calculated through some features implemented in the system, based on Dr. Recupito's studies, which provided mathematical formulas that are applied to the data that have that specific data smell, to calculate those specific DQDs.
As we can see from the figures below **3.1, 3.2** and **3.3**, we have the code snippets for each DQDs and the dependency with a particular type of smell that allows to identify the type of data to be taken into account in the calculation. For example, when we calculate the uniqueness of the dataset, we take the data that was detected as duplicated value, representing the faulty elements for applying later mathematics calculations derived from Recupito's studies, on these data.

```python
# Calculate global and single columns completeness
def calculate_completeness(datasmells):
    completeness_map = {}
    global_elements, global_faulty_elements = 0, 0

    total_rows, total_columns = 0, len(datasmells.keys())
    for column, column_smells in datasmells.items():
        for smell in column_smells:
            total_rows = smell.total_element_count
            break

    for column, column_smells in datasmells.items():
        for smell in column_smells:
            if smell.data_smell_type.smell_type == "Missing Value Smell":
                completeness = ((smell.total_element_count - smell.faulty_element_count) /
                smell.total_element_count) * 100
                completeness_map[column.column_name] = round(completeness, 2)

                global_faulty_elements += smell.faulty_element_count

    global_elements = total_rows * total_columns
    global_completeness = ((global_elements - global_faulty_elements) / global_elements) * 100
    completeness_map["GLOBAL_COMPLETENESS"] = round(global_completeness, 2)
    return completeness_map
```

Figure 3.1: Function for calculating completeness

```python
# Calculate global and single columns uniqueness
def calculate_uniqueness(datasmells):
    uniqueness_map = {}
    global_elements, global_faulty_elements = 0, 0

    total_rows, total_columns = 0, len(datasmells.keys())
    for column, column_smells in datasmells.items():
        for smell in column_smells:
            total_rows = smell.total_element_count
            break

    for column, column_smells in datasmells.items():
        for smell in column_smells:
            if smell.data_smell_type.smell_type == "Duplicated Value Smell":
                uniqueness = ((smell.total_element_count - smell.faulty_element_count) /
                smell.total_element_count) * 100
                uniqueness_map[column.column_name] = round(uniqueness, 2)

                global_faulty_elements += smell.faulty_element_count

    global_elements = total_rows * total_columns
    global_uniqueness = ((global_elements - global_faulty_elements) / global_elements) * 100
    uniqueness_map["GLOBAL_UNIQUENESS"] = round(global_uniqueness, 2)
    return uniqueness_map
```

Figure 3.2: Function for calculating uniqueness

```python
# Calculate global and single columns validity
def calculate_validity(datasmells):
    validity_map = {}
    global_elements, global_faulty_elements = 0, 0

    total_rows, total_columns = 0, len(datasmells.keys())
    for column, column_smells in datasmells.items():
        for smell in column_smells:
            total_rows = smell.total_element_count
            break

    for column, column_smells in datasmells.items():
        for smell in column_smells:
            if smell.data_smell_type.smell_type == "Integer As String Smell" or
                smell.data_smell_type.smell_type == "Floating Point Number As String Smell" or
                smell.data_smell_type.smell_type == "Integer As Floating Point Number Smell":
                validity = ((smell.total_element_count - smell.faulty_element_count) /
                smell.total_element_count) * 100
                validity_map[column.column_name] = round(validity, 2)

                global_faulty_elements += smell.faulty_element_count

    global_elements = total_rows * total_columns
    global_validity = ((global_elements - global_faulty_elements) / global_elements) * 100
    validity_map["GLOBAL_VALIDITY"] = round(global_validity, 2)
    return validity_map
```

Figure 3.3: Function for calculating validity

3.1.2 Testing Results

After the test execution over each dataset, we achieved the results shown in the table **3.1**.

Completeness				
Dataset	**Expected global value**	**Actual global value**	**Expected column values**	**Actual colum values**
Dataset #1	77.33%	77.3%	83.33% / 70% / 66.67%	83.33% / 70% / 66.67%
Dataset #2	66.67%	66.67%	66.67% / 66.67% / 66.67%	66.67% / 66.67% / 66.67%
Dataset #3	81.11%	81.11%	83.33% / 76.67% / 83.33%	83.33% / 76.67% / 83.33%

Uniqueness				
Dataset	**Expected global value**	**Actual global value**	**Expected column values**	**Actual colum values**
Dataset #1	77.78%	77.78%	86.67% / 80% / 66.67%	86.67% / 80% / 66.67%
Dataset #2	33.22%	33.22%	33.33% / 26.67% / 36.67%	33.33% / 26.67% / 36.67%
Dataset #3	47.78%	47.78%	60% / 36.67% / 46.67%	60% / 36.67% / 46.67%

Validity				
Dataset	**Expected global value**	**Actual global value**	**Expected column values**	**Actual colum values**
Dataset #1	92.22%	70%	100% / 100% / 76.66%	100% / 100% / 86.67%
Dataset #2	94.44%	70%	100% / 100% / 83.34%	100% / 100% / 93.33%
Dataset #3	98.89%	98.89%	100% / 100% / 96.67%	100% / 100% / 96.67%

Table 3.1: DQDs test results

As we can see in table **3.1**, after the testing phase on the metrics, we got the same scores (expected and actual results) for both completeness and uniqueness. On the other hand, we got unexpected results over validity due to the wrong assumptions on how the smells related to the metric work. In fact, there is a key difference in how the tests were made on the smells and how the smells work over a dataset. The wrong assumption is that the smell is expected to work as the test case, ignoring the fact that dataset readers can't put different data types into a .csv column, unlike lists with heterogeneous data types. Once you understand this behavior, it is clear that the expected values for validity metric were wrongly calculated, but the effectiveness of the metric still holds since it reveals wrong different data types in the specific column that triggers the data smell.

3.2 Data Smell Analysis

3.2.1 Testing Results

As we can see in the figure **3.4**, we ran the tests that were implemented to verify the correct functioning of the two expectations implemented, and the test did not find any failures, therefore the components for the type of input that were chosen met our expectations.

```
=================== 1 passed, 1 warning in 71.97s (0:01:11) ====================
PASSED [100%]Executing tests for expect_column_values_to_not_contain_special_character_smell
    Executing testcase punctuation_special_character_test
    Executing testcase stressed_letter_special_character_test
    Executing testcase no_special_character_smell_test
Executing tests for expect_column_values_to_not_contain_spacing_smell
    Executing testcase begin_space_test
    Executing testcase multiple_begin_space_test
    Executing testcase inner_space_test
    Executing testcase end_space_test
    Executing testcase multiple_end_space_test
    Executing testcase no_spacing_smell_test
```

Figure 3.4: Data Smells test results

3.3 User Interface Update

3.3.1 Reporting Pages

In order to have a better view of the analysed datasets and their subdivision into groups, as already mentioned and shown in the figure **2.3**, the results page was split into two main pages.

3.3.1.1 Groups Page

Starting from the group's visualization page, it is now possible to view and select the group of analysed datasets instead of the complete list and analysis of all of them, as we can see in the figure **3.5**.

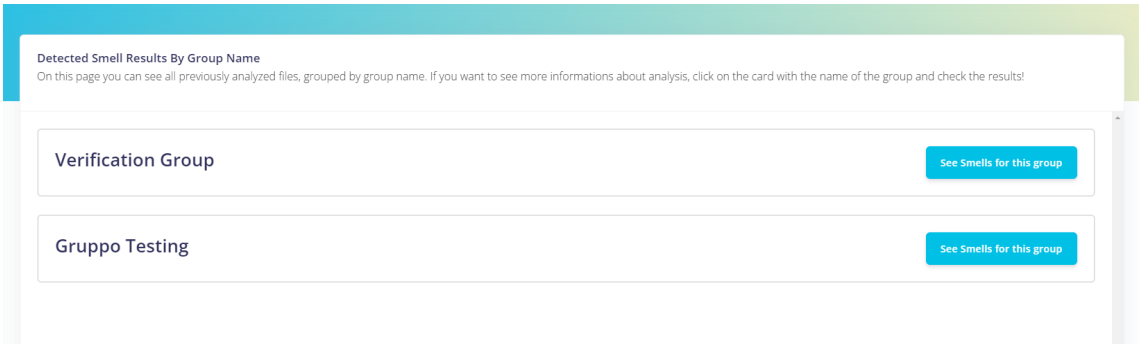

Figure 3.5: Groups Page Interface

3.3.1.2 Groups Results

Next, we have the page on which all the results and analyses for each dataset are shown. The first section includes a quick insight into the metrics' trend and how they change over the datasets analysed, figure **3.6**. It is better to mention that in the following images we used different datasets for the analysis, so the plots do not represent the trend on a single dataset over time. It is up to the user to choose if he wants to analyse a single dataset over time or a group.

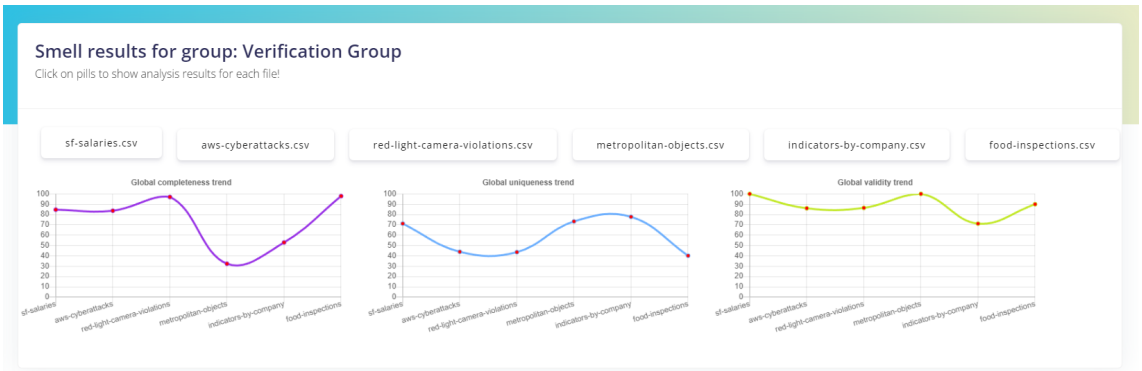

Figure 3.6: Metrics trend over time

The second section includes the datasets names and their columns, as well as the smells configuration. Each clickable pill in the figure **3.7** shows a different table of results and settings depending on the column/smell chosen.

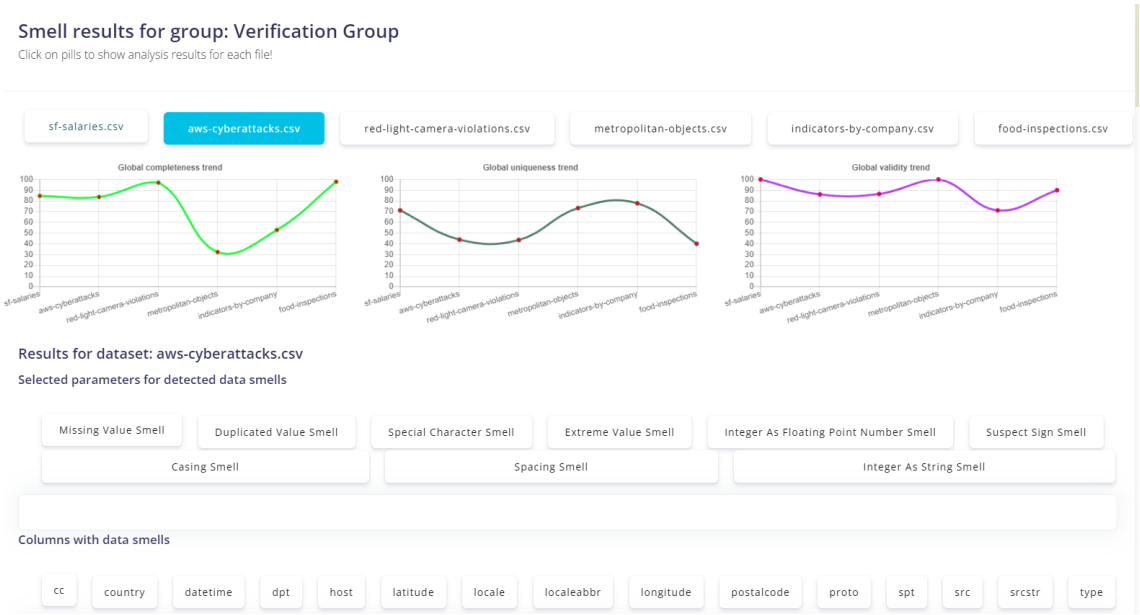

Figure 3.7: Datasets insights

The third and last section is the visualization of the plots for the three DQDs that have been implemented and calculated. As we can see in the figure **3.8**, each plot shows the percentage of valid data for each column and their percentage, plus the complete metric value for the dataset. Just to give an example, we show a particular instance for a single metric, but the design of all of them is the same.

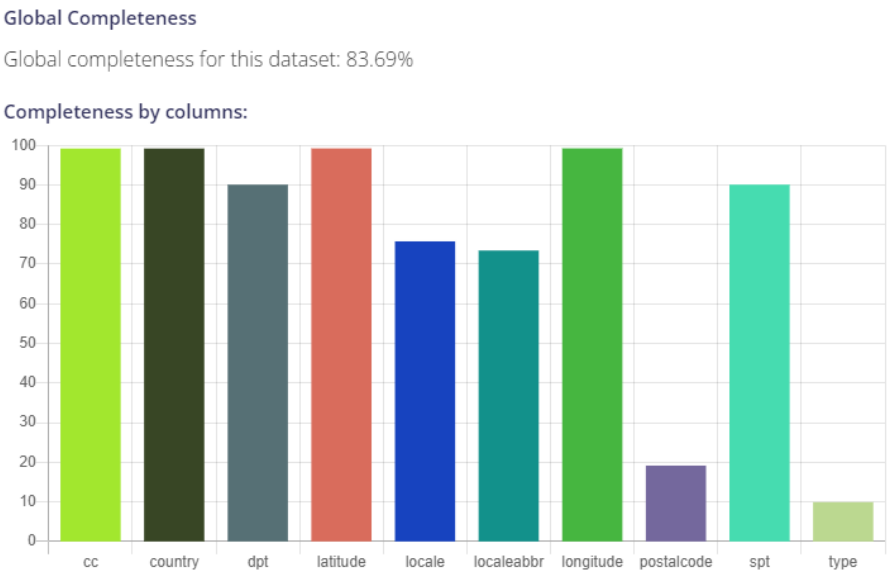

Figure 3.8: Example of a plot for the completeness metric

3.4 Real World Simulation

For the application on real-world data for the two new smells implemented, we chose 6 datasets from the table **2.1**:

- **sf-salaries**: This data contains the names, job title, and compensation for San Francisco city employees on an annual basis from 2011 to 2014
- **aws-cyberattacks**: The AWS Honeypot Database is an open-source database including information on cyberattacks/attempts.
- **indicators-by-company**: This dataset contains US stocks fundamental data, such as the income statement, balance sheet and cash flows.
- **red-light-camera-violations**: This is a dataset hosted by the City of Chicago and updated until 2019. It contains information about traffic light violations recorded by city cameras.
- **food-inspections**: This datae contains information derived from inspections of restaurants and other food establishments in Chicago from January 1, 2010 to 2019.
- **metropolitan-objects**: This dataset contains information on more than 420,000 artworks in The Metropolitan Museum of Art collection for unrestricted commercial and noncommercial use.

The choice of this small subset is simply related to the fact that they represent different domains and fields. Like the other smells already implemented in the previous system, it is reasonable to expect that some of the datasets do not contain at all the smells introduced, since this is only a simulation of usage with real-world data.
For the datasets that contain the smells introduced, tables **3.2** and **3.3** give the full list results for this experiment.

Dataset & Columns		Total Elements Count	Faulty Elements
sf-salaries	EmployeeName	148654	10507
	JobTitle	148654	16
aws-cyberattacks	localeabbr	451581	83
metropolitan-objects	City	448203	7
	Country	448203	8
	Culture	448203	71
	Dimensions	448203	91856
	Dinasty	448203	3
	Excavation	448203	13
	Locale	448203	24
	Locus	448203	5
	Medium	448203	5753
	Portfolio	448203	981
	Region	448203	408
	Reign	448203	3
	River	448203	8
	State	448203	8
	Subregion	448203	530
	Title	448203	4735
food-inspections	Address	153810	153366
	Violations	153810	86170

Table 3.2: Detected spacing smells

Dataset & Columns		Total Elements Count	Faulty Elements
sf-salaries	EmployeeName	148654	4614
	JobTitle	148654	20703
aws-cyberattacks	localeabbr	451581	83
	country	451581	872
	datetime	451581	451581
	host	451581	451581
	srcstr	451581	451581
	locale	451581	11229
red-light-camera-violations	INTERSECTION	521533	11188
	LOCATION	521533	494422
metropolitan-objects	City	448203	1656
	Classification	448203	152157
	Country	448203	2681
	County	448203	1696
	Culture	448203	34459
	Department	448203	12427
	Dimensions	448203	381726
	Dinasty	448203	9013
	Excavation	448203	12608
	Locale	448203	5372
	Locus	448203	2453
	Medium	448203	153559
	Portfolio	448203	12307
	Region	448203	12532
	Reign	448203	2062
	Repository	448203	448203
	River	448203	299
	State	448203	1181
	Subregion	448203	8006
	Title	448203	191502
food-inspections	Address	153810	17136
	Location	153810	153266
	Results	153810	14530
	Risk	153810	153725
	Violations	153810	123012

Table 3.3: Detected special character smells

With this simulation, we can assume that our new smells are very frequent within these datasets, so in most cases, this data could lead to problems.
For example, if we consider a scenario in which a particular datatype needs to be used to train a model and the variables used have data that should represent the same component, but it is represented differently, this leads to evaluating the same data differently (e.g., "John Wayne" and "John Wayne " will be evaluated differently because there's a difference in spaces that brings bias in the model).

CHAPTER 4

Conclusions & Future Works

4.1 Conclusions

Problems related to data debt are common among multiple datasets in every application domain. The improvements made to this tool are clear and give a better view of the results of each analysis, if compared to the previous version.
Both of the newly implemented detectors are correct and effective for their purpose, and the most important part is that they enrich the detector suite provided by the tool. There are still various non-implemented detectors that are documented, so a possible future work is adding even more detectors, maybe using Machine Learning solutions.
Moving to the Data Quality Dimension metrics introduced, they can help the user understand all the problems that are related to them, giving the possibility to check how they evolve over time and analysis. As the detectors, there can be added more DQDs implementations using other detectors as sources of information.
Lastly, there are more possible future works that can be made on this tool in order to enrich even more the tool itself.

4.2 Future Works

There are a lot of features and components that can be added and improved in this tool, which are listed below:

- **Export results in a convenient format**: It could be useful when the results of the detection are provided to export the results, in order to avoid accessing the tool every time, to get the specific information of that detection.
- **Saving custom profiles**: To speed up the process of creating a detection, you can think of saving a specific customization that could be used multiple times to avoid creating multiple different detections with the same parameters while still manually entering all the parameters each time.
- **Extend Data Smell detection suite**: As previously discussed in the previous paragraph, there are some other unimplemented detectors that can be inserted into the detector suite. Surely, it is possible to continue using rule-based approaches, but also adding some ML models can be a valid option.
- **UI improvements**: Some aspects of the UI are still cryptic and less understandable than others. First, a complete redesign of the UI can be done, making the actions that the user can take even more clearly and easily understandable.
- **Full customization over the detection**: One missing feature is the possibility to choose which smell should be searched on each column. In the actual system, the user can only remove columns / data smells from detection.

Bibliography

[1] H. Foidl, M. Felderer, and R. Ramler, "Data smells: categories, causes and consequences, and detection of suspicious data in ai-based systems," in *Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI*, 2022, pp. 229–239. 1, 8

[2] "Have confidence in your data, no matter what," https://greatexpectations.io/. 1

[3] F. Sidi, P. H. Shariat Panahy, L. S. Affendey, M. A. Jabar, H. Ibrahim, and A. Mustapha, "Data quality: A survey of data quality dimensions," in *2012 International Conference on Information Retrieval Knowledge Management*, 2012, pp. 300–304. 1

[4] A. Ramasamy and S. Chowdhury, "Big data quality dimensions: A systematic literature review," *Journal of Information Systems and Technology Management*, vol. 17, no. 0, 2020. [Online]. Available: https://www.tecsi.org/jistem/index.php/jistem/article/view/3126 1, 7

[5] W. Elouataoui, I. El Alaoui, S. El Mendili, and Y. Gahi, "An advanced big data quality framework based on weighted metrics," *Big Data and Cognitive Computing*. [Online]. Available: https://www.mdpi.com/2504-2289/6/4/153 7

[6] G. Recupito, R. Rapacciuolo, D. Di Nucci, and F. Palomba, "Unmasking data secrets: An empirical investigation into data smells and their impact on data quality," 2023. 7, 8

[7] L. G. Martin Kerschbaumer, "rb-data-smell-detection." [Online]. Available: https://github.com/mkerschbaumer/rb-data-smell-detection 8

[8] G. Wenzel, "ml-data-smell-detection." [Online]. Available: https://github.com/georg-wenzel/ml-data-smell-detection 8