HOCHSCHULE HEIDELBERG
Intelligence in Learning

STAATLICH ANERKANNTE HOCHSCHULE

Data Management I

(Data Profiling and Quality Dimensions)

Data Set: Clothing Sales data

Author:

Rohan Raj, 11011896

Contents

Introduction

This report presents the data management concepts, how data profiling is performed and how the cleaning is performed based on the valuable insight of data profiling. The Clothing Sales Data is used as a dataset. The cleaning was done on the data-set taking into consideration some or all the different data quality dimensions.

The first task is to understand and analyze the data-set. Then, data profiling had to be performed (using Talend Data Quality tool) based on this understanding. Also, similar data-set was searched for (online), to get a detailed understanding of the columns. After data profiling, the irregularities were noted down and cleaning was done on the data-set using Trifacta, OpenRefine, and Talend. A detailed explanation of the steps performed in each tool will be given in the later sections of the report.

About the dataset

Clothing Sales Data is the Dataset consist of 13 Columns with 1.048 Million observations (1048544). The metadata/column header is also available. By looking the header one can tell this the dataset consist of the sales transaction. Based on Hypothesis - Dataset is a combination of 3 Master data and 1 transaction data:

1. Employee Master data – EmployeeID, Employee's First, Middle and last name.
2. Customer Master data – CustomerID, Customer's First, Middle and last name.
3. Product data – ProductID,ProductName,ProductPrice.
4. Transaction data – Sales ID, Quantity.

Transaction data is associated with Product, Customer and Employee data.

Tools Used

- Talend Data Quality Management tool- This tool is used to perform data profiling on a given dataset. More details are documented in the Data Profiling section.
- Open Refine – This tool is used for data cleaning. It provides us to use the script GREL/ Python.
- Talend Data Preparation- This tool also used for data cleaning. Although the data limit is 30000, so Random data is selected for understanding the functionality of this tool.
- Cloud Data Prep by Trifacta- This tool used for data cleaning, if the dataset is large enough one can load only sample data. Sample data can be selected based on different approach like Cluster based, random selection, based on column uniqueness etc. In this method, some of the garbage values or outliers were not taken into consideration in the sample hence complete cleaning was not possible in one execution unless one has an idea about all the pattern error.
- MySQL Workbench- After performing basic cleaning, the data set is exported to the MySQL database for further analysis.

Data Profiling

Data Profiling is the process of examining the data available from an existing information source (e.g., a database or a file) and collecting statistics or informative summaries about the data. Talend Data Quality Management tool was used to perform data profiling for this project. Column and Table Analysis was performed during this process.

Summary of each column for the dataset, explaining the three basic statistics of the data

1. Categorical (Nominal) or Numerical (Continuous).
2. Data Type (Integer or string)
3. Missing values in each column.

Variable	Value	Data Type	Missing Count
CustomerID	continuous	integer	0
Customer_LastName	nominal	string	1553
Customer_Middle	nominal	string	484621
Customer_Name	nominal	string	1103
Employee_ID	continuous	integer	223
Employee_Lastname	nominal	string	9157
Employee_Middle	nominal	string	4793
Employee_Name	nominal	string	9130
ProductID	continuous	integer	46
Product_Name	nominal	string	521
Product_Price	nominal	string	358
Quantity	nominal	string	22
SalesID	continuous	integer	0

Fig 3.1 Summary of data

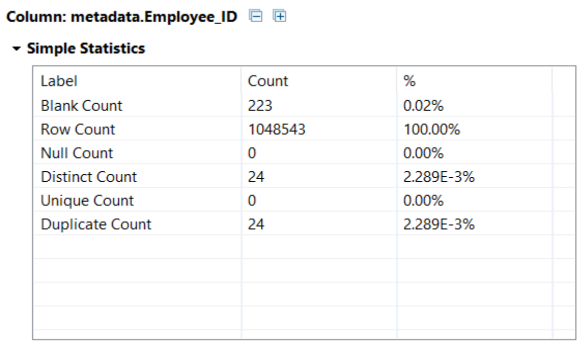
Column Analysis

Column analysis generates a full-frequency distribution and examines all values for a column to infer its definition and properties such as statistical measures and minimum and maximum values.

Simple Statistics

Column analysis gives the insight about the Blank values (No value, whitespace), Row count (column population), Null Count, Distinct count, Unique value, Duplicate value (More than one occurrence). Let's see simple statistics for some of the column to have a basic understanding of the data.

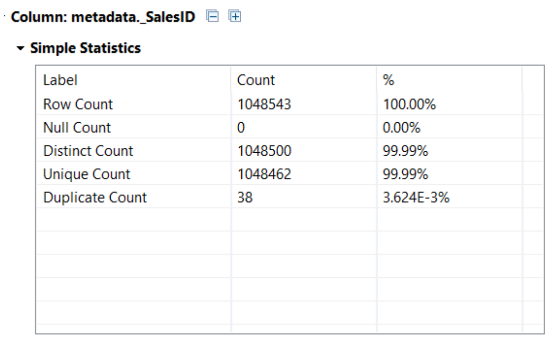
- Employee ID
  Total Employee is only 23 but here distinct value showing 24, that means the blank value is also included in the distinct count.
  The duplicate count is also 24 means, it means there are multiple blank values. Total blank values are 223. Missing or unknown data helps you identify potential data issues



Column: metadata.Employee_ID

Simple Statistics

Label	Count	%
Blank Count	223	0.02%
Row Count	1048543	100.00%
Null Count	0	0.00%
Distinct Count	24	2.289E-3%
Unique Count	0	0.00%
Duplicate Count	24	2.289E-3%



Column: metadata._SalesID

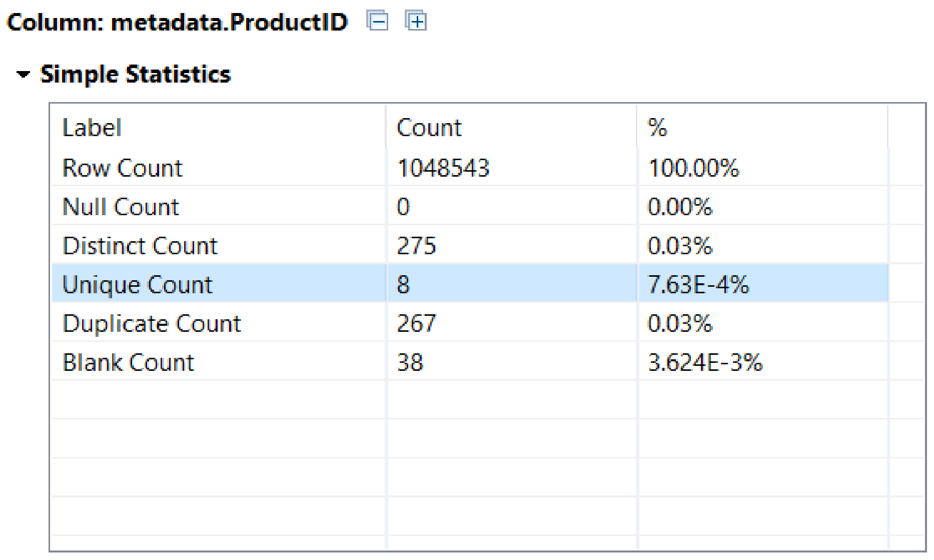
Simple Statistics

Label	Count	%
Row Count	1048543	100.00%
Null Count	0	0.00%
Distinct Count	1048500	99.99%
Unique Count	1048462	99.99%
Duplicate Count	38	3.624E-3%

Fig 3.2 Employee ID and Sales ID Column analysis

- Sales ID
  Sales ID is the candidate key for the dataset as this uniquely identify each row. It has 38 duplicate columns which we need to exclude after checking the validity of data. Analyzing the number of distinct values within the column will help identify possible unique *keys* within the source data.

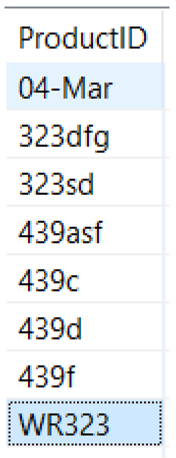
- Product ID
  Product ID has some Unique values, these values appear only once in the observation. Values can be valid or invalid. But after viewing the value, one can tell its invalid for ProductID column.



Column: metadata.ProductID

Simple Statistics

Label	Count	%
Row Count	1048543	100.00%
Null Count	0	0.00%
Distinct Count	275	0.03%
Unique Count	8	7.63E-4%
Duplicate Count	267	0.03%
Blank Count	38	3.624E-3%



ProductID
04-Mar
323dfg
323sd
439asf
439c
439d
439f
WR323

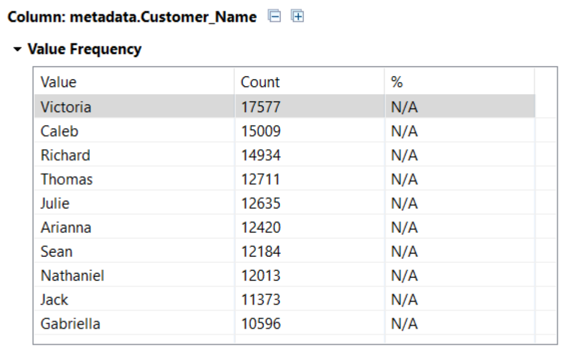
Fig 3.3Product ID Simple Statistics

Nominal Analysis

If the column contains categorical or factor values, then discrete data analysis gives a clear understanding of the columns. If the column has many different formats for the same factors, then the most frequent occurring value-type or format can be used for generalizing the factors in the column. The same can be seen from the below explained analysis done on some of the columns:

Customer Name, Product Name

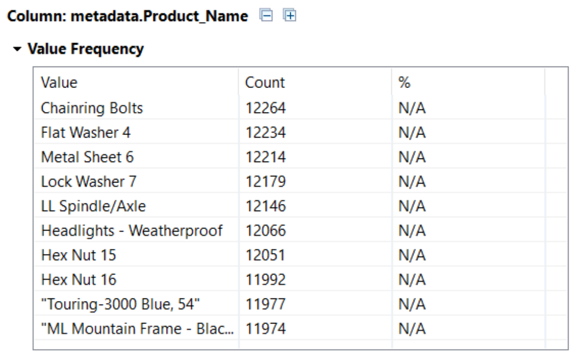
Top 10 Customer who purchased the product can be seen using the Frequency table and top 10 most product sold list. This Column will be used for Hypothetical use case, how important are the customer and most sold product.



Column: metadata.Customer_Name

Value Frequency

Value	Count	%
Victoria	17577	N/A
Caleb	15009	N/A
Richard	14934	N/A
Thomas	12711	N/A
Julie	12635	N/A
Arianna	12420	N/A
Sean	12184	N/A
Nathaniel	12013	N/A
Jack	11373	N/A
Gabriella	10596	N/A



Column: metadata.Product_Name

Value Frequency

Value	Count	%
Chainring Bolts	12264	N/A
Flat Washer 4	12234	N/A
Metal Sheet 6	12214	N/A
Lock Washer 7	12179	N/A
LL Spindle/Axle	12146	N/A
Headlights - Weatherproof	12066	N/A
Hex Nut 15	12051	N/A
Hex Nut 16	11992	N/A
"Touring-3000 Blue, 54"	11977	N/A
"ML Mountain Frame - Blac...	11974	N/A

Fig 3.4Customer and Product Name Nominal

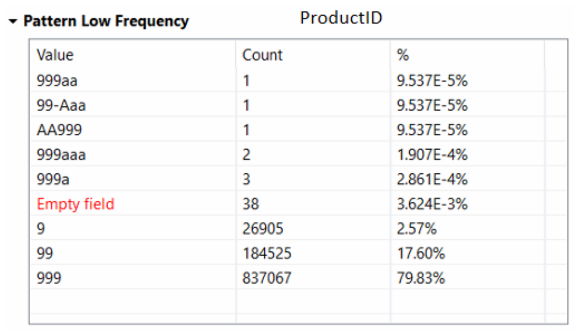
Data Types and Patterns

This method is mostly used on columns which have string values. The pattern followed by each column can be seen, and we can check the values which are not following a pattern. We can find patterns sorted by highest frequency as well as the lowest frequency. Low-frequency analysis can be very helpful in finding pattern mismatch. There are also other options like East-Asia Pattern, CI Word Pattern, Date Pattern and CS Word Pattern which can be used based on the requirement. Pattern Frequency Analysis done on some columns are as explained below:

- ProductID and Quantity Column

  ProductID has Integer Data Type with range 1-999. There is some value which is inaccurate, it should be of integer type and value is inconsistence. Quantity column has also Integer Data Type, there are some invalid entries as quantity should be positive only.

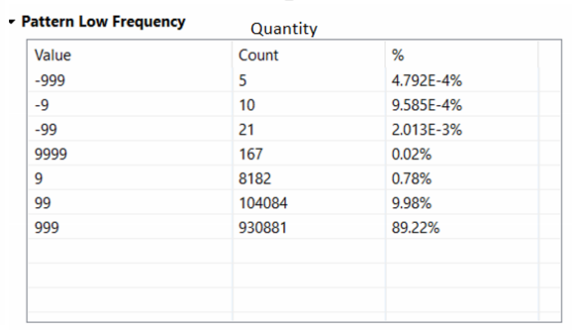
  Analyzing string lengths of the source data is a valuable step in selecting the most appropriate data types and sizes in the target database.



Pattern Low Frequency

ProductID

Value	Count	%
999aa	1	9.537E-5%
99-Aaa	1	9.537E-5%
AA999	1	9.537E-5%
999aaa	2	1.907E-4%
999a	3	2.861E-4%
Empty field	38	3.624E-3%
9	26905	2.57%
99	184525	17.60%
999	837067	79.83%



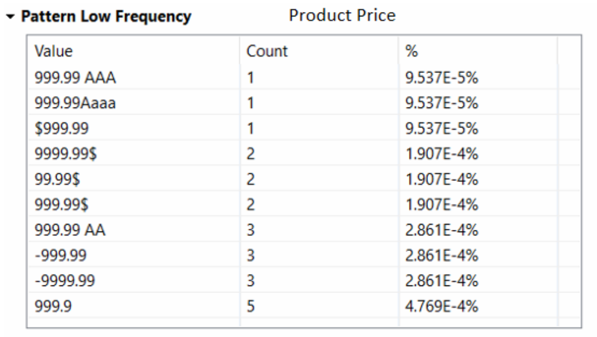
Pattern Low Frequency

Quantity

Value	Count	%
-999	5	4.792E-4%
-9	10	9.585E-4%
-99	21	2.013E-3%
9999	167	0.02%
9	8182	0.78%
99	104084	9.98%
999	930881	89.22%

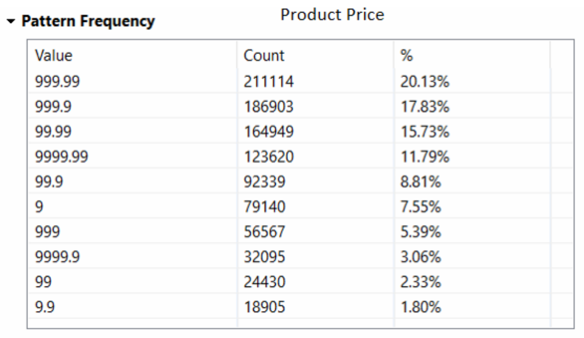
Fig 3.5Product ID and Quantity Pattern

- Product Price has Data Type Integer, it has lots of junk value append as a prefix and suffix. Some value has no currency indicator, some have Either $ or €. The currency symbol is also attached as prefix and suffix to price value.
  Reducing the column widths to be just large enough to meet current and future requirements will improve query performance by minimizing table scan time.



Pattern Low Frequency Product Price

Value	Count	%
999.99 AAA	1	9.537E-5%
999.99Aaaa	1	9.537E-5%
$999.99	1	9.537E-5%
9999.99$	2	1.907E-4%
99.99$	2	1.907E-4%
999.99$	2	1.907E-4%
999.99 AA	3	2.861E-4%
-999.99	3	2.861E-4%
-9999.99	3	2.861E-4%
999.9	5	4.769E-4%



Pattern Frequency Product Price

Value	Count	%
999.99	211114	20.13%
999.9	186903	17.83%
99.99	164949	15.73%
9999.99	123620	11.79%
99.9	92339	8.81%
9	79140	7.55%
999	56567	5.39%
9999.9	32095	3.06%
99	24430	2.33%
9.9	18905	1.80%

Fig 3.5Product Price Pattern

Table Analysis

Table analysis is used to analyze data from a table perspective. The identification of candidate key for the table also confirmation of Primary key. Once the primary key is identified, the duplicate check can be performed.

Functional Dependency Analysis

Functional Dependency analysis finds how one column value functionally determines another column value which is dependent.

Introduction section, a hypothesis about 3 master data are mention. In all the master data there is one primary key and candidate key for the whole dataset. Customer ID, Employee ID, and Product ID are the primary key for their respective master data table.

For whole Data Set- Customer ID, Employee ID, Product ID, and Sales ID or the combination of these can be the primary key.

- Customer ID functionally determines Customer Name with 93.61% dependency strength. It'll also be able to determine Customer middle and last name. Product ID functionally determines Product Name with 93.66% dependency strength. It'll also be able to determine Product price.

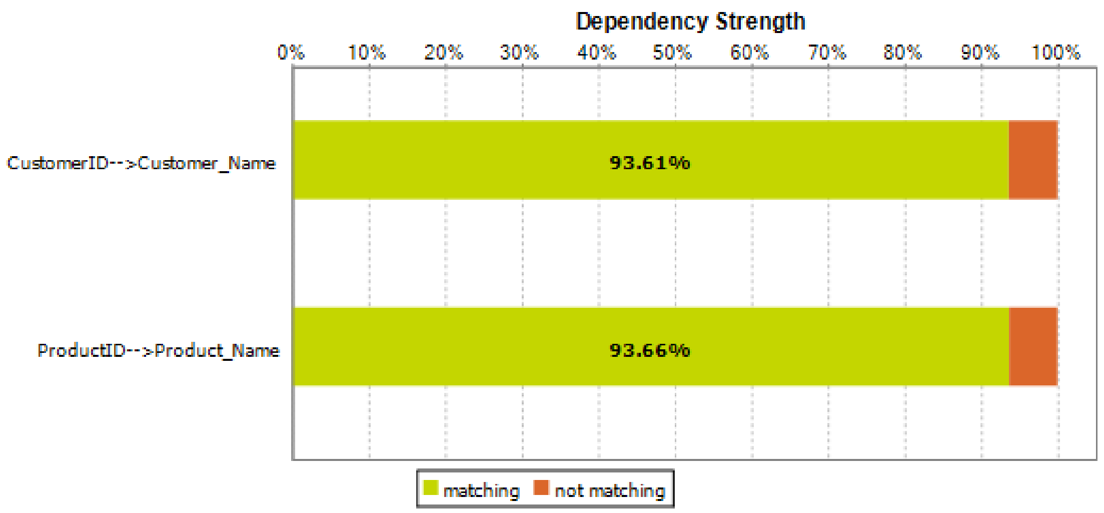


Fig 3.6 ID's and Name Functional Dependency

- Customer ID functionally determines the Product name, Product ID functionally determines the Product price.
  Product Name and Quantity giving the high dependency strength because of multiple transactions has the same quantity.

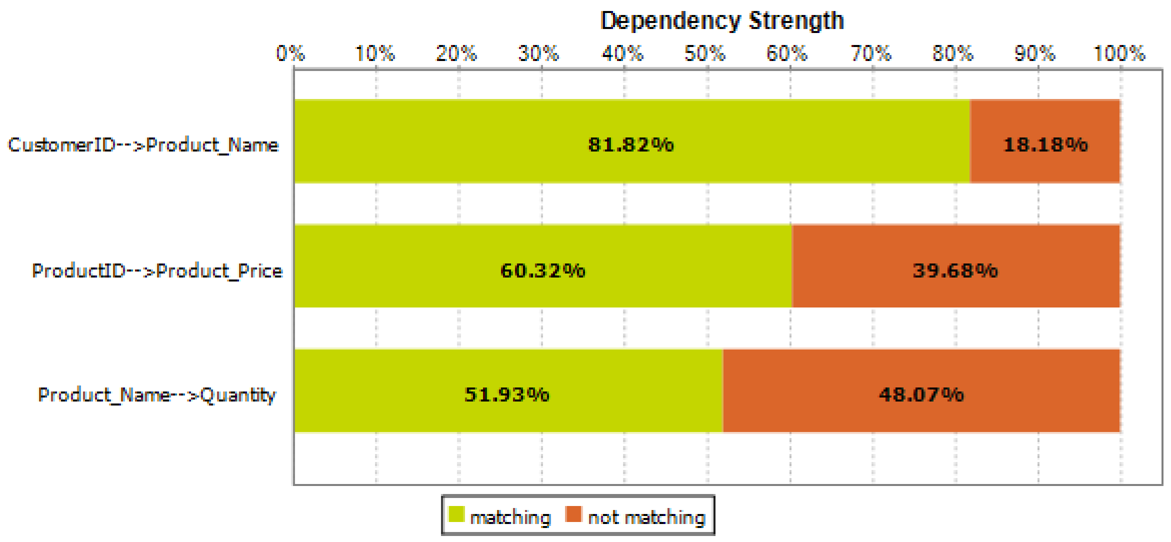


Fig 3.7 Customer ID and product Name Functional Dependency

Data Cleaning

In this section, all the data quality dimension will be mentioned on each column level. After finding the required cleaning steps that must be performed based on the above profiling, the data was cleaned using Cloud-Data prep by Trifacta, Talend and Open Refine. Each of the used tools has its own advantages and disadvantages, which have already been described in the 'Tools Used' section of this project.

After searching from different sources, similar data-file was fetched from websites, to perform comparison and find dis-similarities. Also, a composite primary key consisting of Sales ID, Customer ID ProductID was identified (as these columns had not null values, and when concatenated together had unique values). Before cleaning the data set, each quality dimension check was done for each column and some steps were identified. These steps were then executed and clean- data set was obtained. A detailed explanation of these steps is given below, along with the associated quality dimensions:

Summary of all the column and associated dimension should be performed on that column.

Column Name	Quality Dimension
SalesID	Consistency (Uniqueness), Completeness(mandatory values)
EmployeeID	Completeness(mandatory values)
CustomerID	Completeness (mandatory values)
ProductID	Consistency (Format Consistency),Completeness(mandatory values), Accuracy(should be Integer only)
Quantity	Completeness (mandatory values), Accuracy (Accuracy to reality), Validity (Metadata compliance), Conformity (Internal standard)
ProductName	Completeness, Consistency(Value consistency)
ProductPrice	Validity, Consistency(format consistency), currency (Picked from two different countries), Conformity (External source, Internal standard)
EmployeeName	Accuracy (Precision),Completeness, Validity, Consistency (Value consistency)
EmployeeMiddle, Last Name	Accuracy (Precision), Completeness, Validity, Consistency (Value consistency)
CustomerName	Accuracy (Precision), Completeness, Validity, Consistency (Value consistency)
CustomerMiddle, Last Name	Accuracy (Precision), Completeness, Validity, Consistency (Value consistency)

UseCase: Find all the customer who always makes some purchase or most product has been sold.

The product is divided into 4 Main categories (1):

1. Bikes
2. Components
3. Clothing
4. Accessories

Dimension on each column

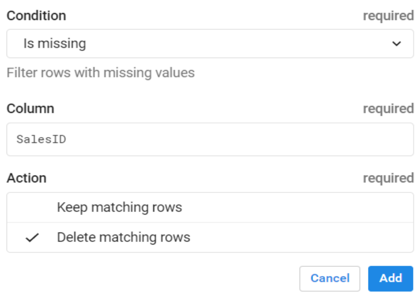
All the relevant data quality dimension based on column:

Sales ID

Dimension:

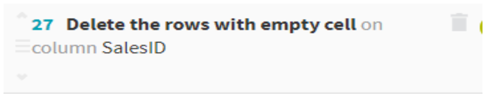
1. *Consistency*: This column has 1048462 unique records, but there are some duplicate records (38) which need to be discarded. Refer-Fig 3.2
   Function used:
   Trifacta- Remove duplicate rows based on Sales ID column
   Talend- Remove duplicate row on the table
   Open Refine- Not able to remove based on the single column, manually removed

2. *Completeness*: This column is uniquely identifiable in each record, so this is the mandatory field for the dataset.
   Function used:
   a. Trifacta- Filter the rows with a missing value, select the required column and delete the matching rows.



Condition: Is missing (required)
Filter rows with missing values
Column: SalesID (required)
Action (required): Keep matching rows [✓ Delete matching rows]
Cancel Add

   b. Talend- Delete the rows with an empty cell on column Sales ID



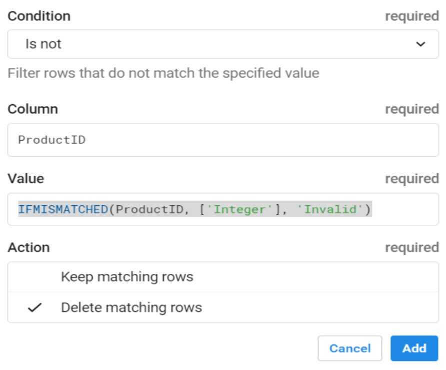
27 **Delete the rows with empty cell** on column SalesID

   c. Open Refine- Cannot perform the facet because of the large dataset

Product ID

Dimension:

1. *Consistency*: Product Id format should be consistent throughout the dataset. It has 8 value were junk value is appended as a prefix and suffix. Please refer Fig 3.2
   Function used:
   a. Trifacta: Filter rows that do not match the specified value, select column Product ID and insert the formula IFMISMATCHED (ProductID, ['Integer'], 'Invalid'). At Last, Delete the matching rows (Because Only 8 value was there).



Condition required

Is not

Filter rows that do not match the specified value

Column required

ProductID

Value required

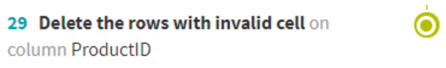
`IFMISMATCHED(ProductID, ['Integer'], 'Invalid')`

Action required

Keep matching rows

✓ Delete matching rows

b. Talend: Delete the rows with an invalid cell



29 **Delete the rows with invalid cell** on column ProductID

c. Open Refine: Not able to remove based on the single column, manually removed

2. *Accuracy*: Product ID should be of integer data type and range from 1-999 as 99.98% percent of data lies in this range.

Quantity

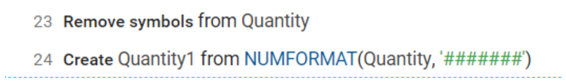
Dimension:

1. *Completeness*: Quantity is mandatory for each transaction, based on product quantity only the price will be calculated. Complete the column by either removing the blank record or fill with the correct value.

2. *Consistency*: Quantity value is inconsistence; some special character is appended around the value.

   Function used:

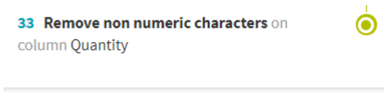
   a. Trifacta: Remove all the symbol from Quantity, however it doesn't remove all the symbol from the value. Next, we can use other function to create a new column based on formula (only allow integer). Then we can delete the missing value.



23 **Remove symbols** from Quantity

24 **Create** Quantity1 from NUMFORMAT(Quantity, '#######')

   b. Talend: Remove non-numeric characters is quite easy here, by just calling one function to remove non-numeric characters.

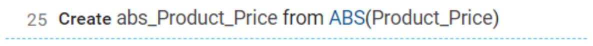


33 **Remove non numeric characters** on column Quantity

   c. Open Refine: Using Python script and Regular expression library, the same function can be achieved.

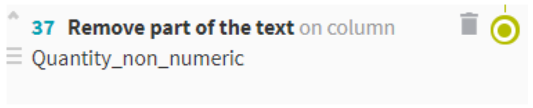
```
Import re
quantity = value
data = re.findall("\d+\.*\d+", s)
return data [0]
```

3. *Validity (Metadata compliance) and Conformity (Internal standard):* Quantity cannot be negative for a given dataset.
   a. Trifacta: Create a new column with ABS function, it'll convert all the



25 Create abs_Product_Price from ABS(Product_Price)

   b. Talend: Using function remove part of the text can remove all the negative sign (-) symbol.



37 Remove part of the text on column Quantity_non_numeric

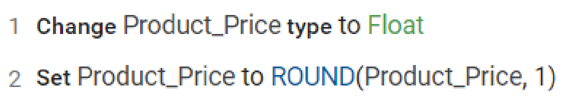
   c. Open Refine: Using python script which treat negative sign (-) also as a non-numeric and remove using same regular expression.

```
 data = re.findall("\d+\.*\d+", s)
```

Product Price

Dimension:

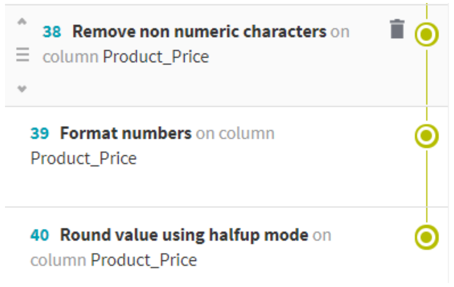
1. *Validity*: Product price has some negative sign (-) value, which invalid as per Standards and Regulatory compliance. All the function is the same what is used in Quantity Validity dimension.
2. *Consistency*(format consistency): Product price made consistence by removing all the special characters. All the function is the same what is used in Quantity Consistency dimension.
3. *Currency*(Picked from a different country): Product Price consisting of different currency (DOLLAR, RS, EURO) is also available in the dataset. Price conversion was not performed because after comparing with the External source get to know that price value is same only Currency symbol is different.
4. *Conformity*(External source, Internal standard) (2) (3): Conformity Check is performed with external Standards to find out the valid price value.
   Function used:
   a. Trifacta: After converting the value to float type, use the round () function to specify the rounding decimal point.



1 Change Product_Price type to Float

2 Set Product_Price to ROUND(Product_Price, 1)

b. Talend: Using Round value using half up mode on the price column and mention the precision value.


38 **Remove non numeric characters** on column Product_Price

39 **Format numbers** on column Product_Price

40 **Round value using halfup mode** on column Product_Price

c. Open Refine: Using GREL scripting, the round function helps to round the value. Create a column based on Product Price and use the below script to round up the value.

round(toNumber(value))

Exporting to MySQL

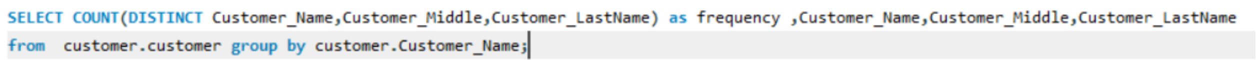
After performing basic cleaning, the data set is exported to the MySQL database for further analysis. For importing the dataset, one schema for whole dataset and three schemas for different master data have been created:

1. Customer master data
2. Employee master data
3. Product master data

Lookup dataset

The reason for creating three separate schemas is to perform the lookup operation. In original dataset, there is some error which can be easily rectified using a lookup table.

Following errors are present in the dataset: For Individual Customer, Employee and Product different names, typo mistake, blank value, error values are present in the dataset.



```
SELECT COUNT(DISTINCT Customer_Name,Customer_Middle,Customer_LastName) as frequency ,Customer_Name,Customer_Middle,Customer_LastName
from  customer.customer group by customer.Customer_Name;
```

This query gives all the Distinct Customer with their frequency, the new Customer master data is created. In this dataset, each entry appears once and has the best possible value from the whole dataset.

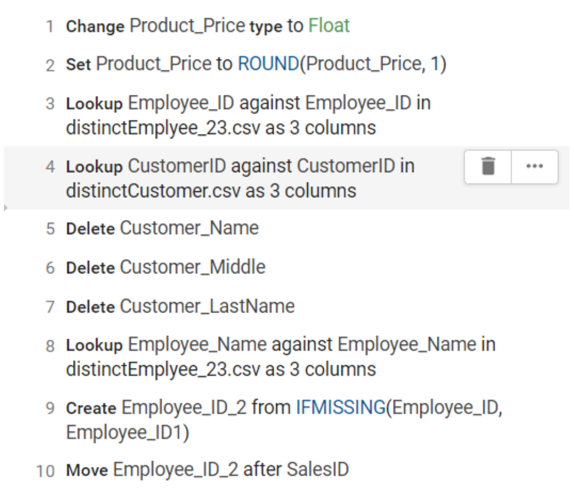
Likewise, for the Employee and the product, the distinct dataset has been created. Manual effort is also required to select the best possible value.

This Lookup dataset will help for multiple data quality dimension:

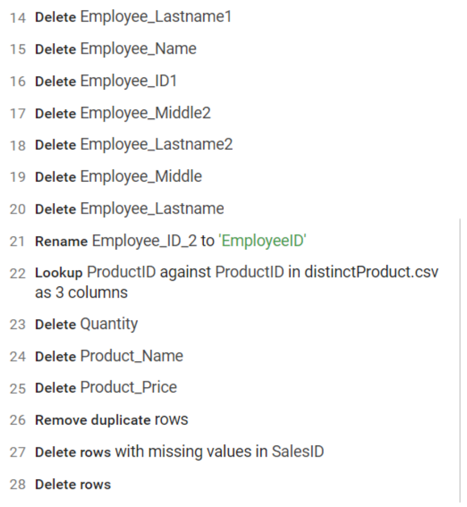
Completeness, Consistency, Validity, and Accuracy.

Trifacta

Recipe

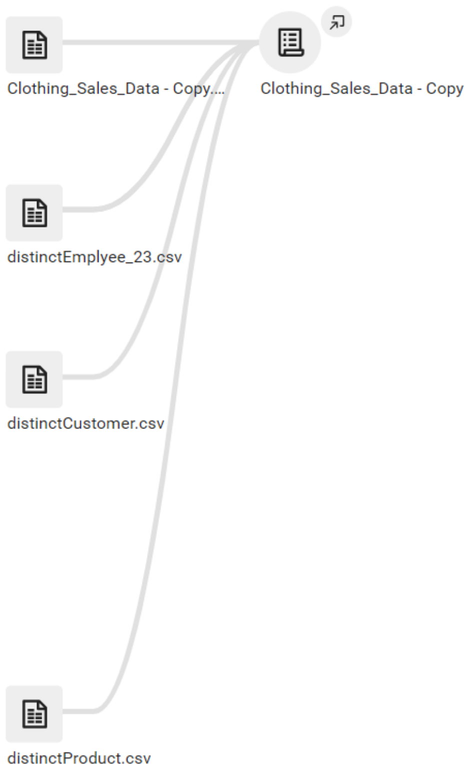


1 **Change** Product_Price **type** to Float
2 **Set** Product_Price to ROUND(Product_Price, 1)
3 **Lookup** Employee_ID against Employee_ID in distinctEmplyee_23.csv as 3 columns
4 **Lookup** CustomerID against CustomerID in distinctCustomer.csv as 3 columns
5 **Delete** Customer_Name
6 **Delete** Customer_Middle
7 **Delete** Customer_LastName
8 **Lookup** Employee_Name against Employee_Name in distinctEmplyee_23.csv as 3 columns
9 **Create** Employee_ID_2 from IFMISSING(Employee_ID, Employee_ID1)
10 **Move** Employee_ID_2 after SalesID

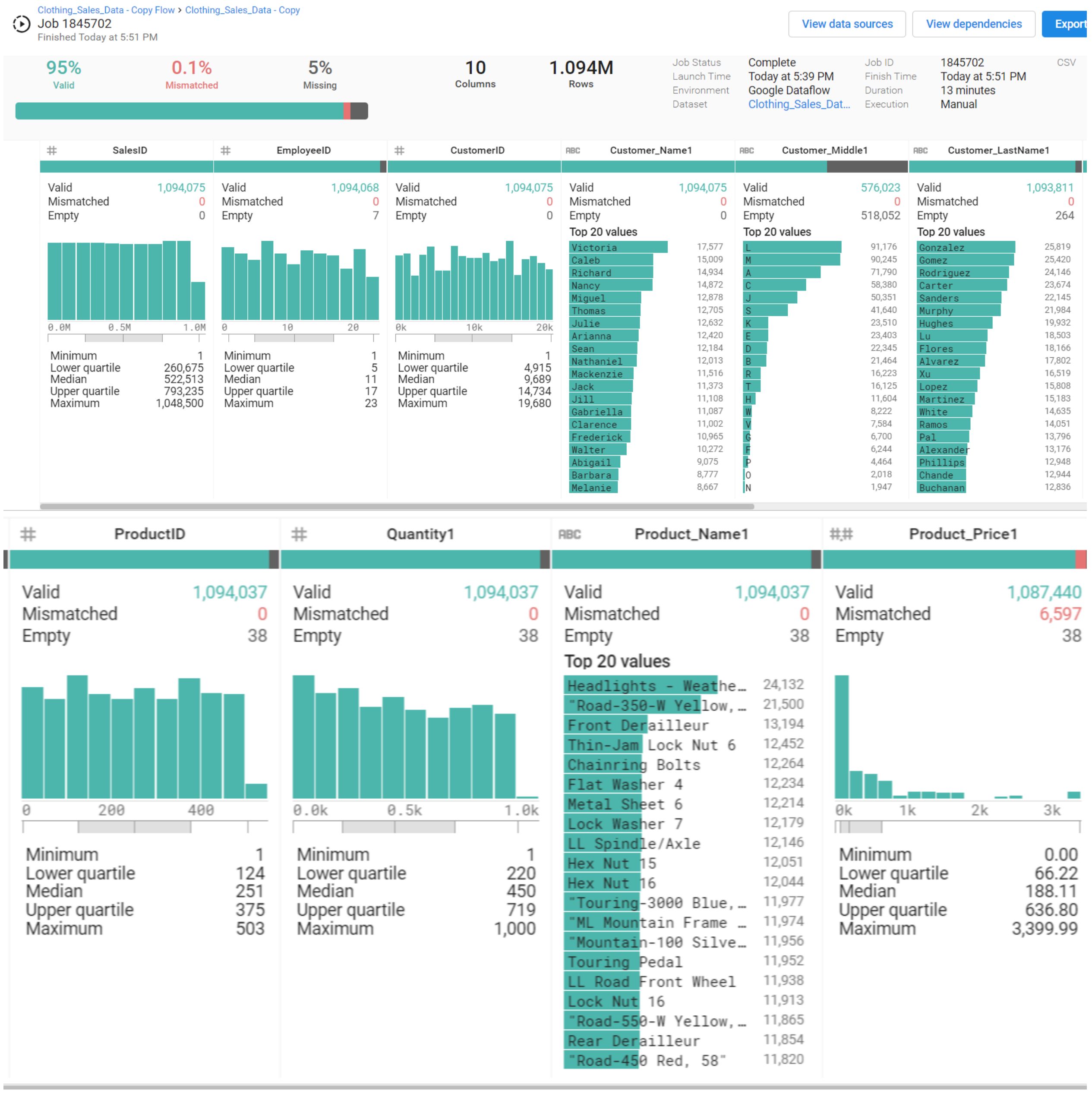


14 **Delete** Employee_Lastname1
15 **Delete** Employee_Name
16 **Delete** Employee_ID1
17 **Delete** Employee_Middle2
18 **Delete** Employee_Lastname2
19 **Delete** Employee_Middle
20 **Delete** Employee_Lastname
21 **Rename** Employee_ID_2 to 'EmployeeID'
22 **Lookup** ProductID against ProductID in distinctProduct.csv as 3 columns
23 **Delete** Quantity
24 **Delete** Product_Name
25 **Delete** Product_Price
26 **Remove duplicate** rows
27 **Delete rows** with missing values in SalesID
28 **Delete rows**

Flow Details



Job Result



Clothing_Sales_Data - Copy Flow › Clothing_Sales_Data - Copy
Job 1845702
Finished Today at 5:51 PM

95% Valid | 0.1% Mismatched | 5% Missing | 10 Columns | 1.094M Rows

Job Status	Complete	Job ID	1845702	CSV
Launch Time	Today at 5:39 PM	Finish Time	Today at 5:51 PM	
Environment	Google Dataflow	Duration	13 minutes	
Dataset	Clothing_Sales_Dat…	Execution	Manual	

	SalesID	EmployeeID	CustomerID	Customer_Name1	Customer_Middle1	Customer_LastName1
Valid	1,094,075	1,094,068	1,094,075	1,094,075	576,023	1,093,811
Mismatched	0	0	0	0	0	0
Empty	0	7	0	0	518,052	264

	SalesID	EmployeeID	CustomerID
Minimum	1	1	1
Lower quartile	260,675	5	4,915
Median	522,513	11	9,689
Upper quartile	793,235	17	14,734
Maximum	1,048,500	23	19,680

Top 20 values

Customer_Name1		Customer_Middle1		Customer_LastName1	
Victoria	17,577	L	91,176	Gonzalez	25,819
Caleb	15,009	M	90,245	Gomez	25,420
Richard	14,934	A	71,790	Rodriguez	24,146
Nancy	14,872	C	58,380	Carter	23,674
Miguel	12,878	J	50,351	Sanders	22,145
Thomas	12,705	S	41,640	Murphy	21,984
Julie	12,632	K	23,510	Hughes	19,932
Arianna	12,420	E	23,403	Lu	18,503
Sean	12,184	D	22,345	Flores	18,166
Nathaniel	12,013	B	21,464	Alvarez	17,802
Mackenzie	11,516	R	16,223	Xu	16,519
Jack	11,373	T	16,125	Lopez	15,808
Jill	11,108	H	11,604	Martinez	15,183
Gabriella	11,087	W	8,222	White	14,635
Clarence	11,002	V	7,584	Ramos	14,051
Frederick	10,965	G	6,700	Pal	13,796
Walter	10,272	F	6,244	Alexander	13,176
Abigail	9,075	P	4,464	Phillips	12,948
Barbara	8,777	O	2,018	Chande	12,944
Melanie	8,667	N	1,947	Buchanan	12,836

	ProductID	Quantity1	Product_Name1	Product_Price1
Valid	1,094,037	1,094,037	1,094,037	1,087,440
Mismatched	0	0	0	6,597
Empty	38	38	38	38

	ProductID	Quantity1	Product_Price1
Minimum	1	1	0.00
Lower quartile	124	220	66.22
Median	251	450	188.11
Upper quartile	375	719	636.80
Maximum	503	1,000	3,399.99

Top 20 values

Product_Name1	
Headlights - Weathe…	24,132
"Road-350-W Yellow, …	21,500
Front Derailleur	13,194
Thin-Jam Lock Nut 6	12,452
Chainring Bolts	12,264
Flat Washer 4	12,234
Metal Sheet 6	12,214
Lock Washer 7	12,179
LL Spindle/Axle	12,146
Hex Nut 15	12,051
Hex Nut 16	12,044
"Touring-3000 Blue, …	11,977
"ML Mountain Frame …	11,974
"Mountain-100 Silve…	11,956
Touring Pedal	11,952
LL Road Front Wheel	11,938
Lock Nut 16	11,913
"Road-550-W Yellow, …	11,865
Rear Derailleur	11,854
"Road-450 Red, 58"	11,820

Talend

Talend has Import limit, so whatever step has been performed in Trifacta similar steps perform here too.

Recipe

Sales_Data_Unique Preparation

1. Lookup is done with dataset distinctEmplyee_23. Join has been set between Employee_Name and Employee_Name. The column Employee_ID has been added.
2. Fill empty cells with text on column Employee_ID
3. Lookup is done with dataset distinctEmplyee_23. Join has been set between Employee_Lastname and Employee_Lastname. The column Employee_ID has been added.
4. Fill empty cells with text on column Employee_ID
5. Lookup is done with dataset distinctEmplyee_23. Join has been set between Employee_ID and Employee_ID. The columns Employee_Name, Employee_Middle and 1 other(s) have been added.
6. Delete column on column Employee_Name
7. Delete column on column Employee_Lastname
8. Lookup is done with dataset distinctEmplyee_23. Join has been set between Employee_ID and Employee_ID. The columns Employee_Name, Employee_Middle and 1 other(s) have been added.
9. Change data type on column CustomerID
10. Lookup is done with dataset distinct customer. Join has been set between CustomerID and CustomerID. The columns Customer_Name, Customer_Middle and 1 other(s) have been added.
11. Delete these filtered rows on column ProductID
12. Standardize value (fuzzy matching) on column ProductID
13. Lookup is done with dataset distinctProduct1. Join has been set between ProductID and ProductID. The columns Quantity, Product_Name and 1 other(s) have been added.
14. Delete these filtered rows on column ProductID
15. Remove duplicate rows on the table
16. Delete empty rows on the table
17. Delete the rows with an empty cell on column SalesID
18. Remove non-numeric characters on column Quantity

Open Refine

Open Refine is used to perform data-cleaning task. This tool has the extract and applies the feature as a JSON file for the recipe. However, the operations performed are masked and not clearly defined. For e.g. “Mass edit 573 cells in column Product_Price”, this doesn’t explain about the facet and operation. Lots of Recipe is remove was repetitive.

Recipe

1. Create project
2. Create new column Product_Price_Integer based on column Product_Price by filling 17 rows with grel:replace(value, /\d/, '')
3. Remove column Product_Price_Integer
4. Mass edit 390737 cells in column Product_Price
5. Mass edit 24648 cells in column Product_Price
6. Mass edit 573 cells in column Product_Price
7. Mass edit 1133 cells in column Product_Price
8. Create new column Product_Price_New based on column Product_Price by filling 0 rows with jython:import re s = value data = re.findall("\d+\.*\d+", s) return data
9. Text transform on 1048543 cells in column Product_Price_New: ""
10. Text transform on 0 cells in column Product_Price_New: value.toString()
11. Create new column Produc_Price_New based on column Product_Price by filling 1048185 rows with jython:import re s = value data = re.findall("\d+\.*\d+", s) return (", ".join(data))
12. Mass edit 79498 cells in column Produc_Price_New
13. Move column Product_Price to position 14
14. Remove column Product_Price_New
15. Create new column Quantity_New based on column Quantity by filling 1048521 rows with jython:import re s = value data = re.findall("\d+\.*\d+", s) return (", ".join(data))
16. Move column Quantity to position 14
17. Mass edit 8249 cells in column Quantity_New
18. Fill down 1 cell in column Product_Name
19. Flag 9130 rows

Bibliography

(1). Retrieved from http://www.britishmicro.com/MvcDemo/Products

(2). Retrieved from https://demos.componentone.com/ASPNET/AdventureWorks/Products.aspx?Category=Clothing

(3). Retrieved from https://demos.componentone.com/ASPNET/AdventureWorks/Products.aspx?Category=Bikes