Global Pharmaceutical Barcode Retrieval and Analysis for Enhanced Supply Chain Management

Tejesh Varma Rudraraju

Index

1. Introduction

1.1 Geographic Scope

The project focuses on two Latin American countries where oversees hospitals: Chile and Mexico. These countries were selected due to their significant presence international operations and the unique market characteristics they present. By narrowing the scope to these countries, the project aims to streamline the retrieval of barcodes for pharmaceutical products international item master data specific to the Latin American context.

1.2 Barcode Differences

Due to the variations in market characteristics between the Latin American countries and the United States, the barcodes assigned to products differ in these regions. This discrepancy poses a challenge for managing inventory and supply chain processes, as it requires specific barcode identification for each country. The project aims to address this challenge by utilizing web scraping techniques to retrieve the barcodes for pharmaceutical products in international item master data, specifically focusing on the barcode differences between the Latin American countries and the United States.

1.3 Data Sources

The project utilizes data provided by Healthcare Data from Chile and Mexico as the primary sources for analysis. This data includes comprehensive information related to product details, vendor information, location data, transaction records, and product categories. Leveraging these datasets, the project aims to identify the barcode information associated with the pharmaceutical products in the international item master data. By utilizing these data sources, the project ensures access to accurate and reliable information for barcode retrieval.

2.4 Objectives and Deliverables

The main objective of the project is to retrieve barcodes for pharmaceutical products in international item master data using web scraping techniques. The project aims to gather additional information, including serial numbers, manufacturing names, and actual product descriptions, to enhance the accuracy of barcode identification.

The deliverables of the project include a refined dataset with barcode information, insights into barcode differences between Latin American countries and the United States, and recommendations for improving inventory and supply chain processes.

3. Data Definition

The dataset utilized in this project consists of comprehensive data obtained , specifically from their operations in Chile and Mexico. The dataset contains 391,203 rows and 43 columns, encompassing various aspects of the inventory and supply chain processes. These include product details, vendor information, location data, transaction records, and product categories.

. By leveraging this dataset, the project aims to analyze the data, uncover patterns, and extract meaningful information to facilitate barcode retrieval and improve operational efficiency.

1. Product Manufacture details	**2. Vendor**
Product - ERP Manufacturer Name	Vendor - Vendor Tax Id
Product - ERP Catalog Number	Vendor - Supplier ID
Product - Manufacturer Name (Complete)	Vendor – Name
Product - Manufacturer Name (Parent)	Vendor - Invoice Number
Product - Manufacturer Name (Division)	Vendor City
Product - Manufacturer Catalog Number	Vendor Country
	Vendor Region
3. Location	**4. Transaction**
Country	Accounting Date
Data Source	Transaction - Invoice ID
Entity Hospital	Transaction Po Date
Entity - Facility Name	Transaction - PO ID

Entity City Entity Country Entity Region Entity - Hospital Cost Center	Transaction - PO Line Number Transaction - Account ID Transaction - Account Name Transaction Currency Transaction - Contracted Price Transaction – Quantity Transaction Total Value No Iva Local Currency Transaction Total Value No Iva USD

5. Product

Product – Part Number

Product – Description

Product Category

Product Category – UNSPSC

Product Category – UNSPSC – 1 – Segment

Product Category – UNSPSC – 2 – Family

Product Category – UNSPSC – 3 – Class

Product Category – UNSPSC – 4 – Commodity

Product - Unit of Measure

4. Data Analysis

4.1 Dataset Overview

The dataset consists of 391,203 rows and 43 columns, encompassing data from two countries, Mexico and Chile. The distribution is as follows: 84.43% of the data is from Mexico, while the remaining 15.57% is from Chile.

4.2 Proportions of Data from Mexico and Chile

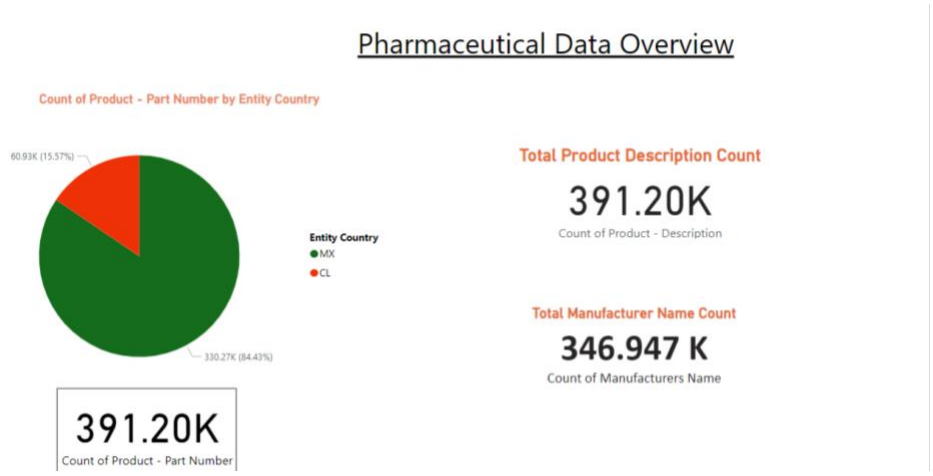
The dataset's composition indicates a significant presence of Mexican data compared to Chilean data. This discrepancy may impact the analysis and findings, necessitating appropriate consideration during the project.

4.3 Key Fields for Barcode Retrieval

Among the various fields in the dataset, the primary focus for barcode retrieval lies in the "Product Description" and "Manufacturer Name." These fields were identified as crucial for locating the barcodes effectively.

4.4 Visualization of Product Description and Manufacturer Name

To gain insights into the dataset, visualizations were created for the "Product Description" and "Manufacturer Name" fields. The visualization revealed that there were 391.20K unique product descriptions, while the count of unique manufacturer names was 346.947K.



4.5 Data Cleaning and Preprocessing

To ensure data quality and eliminate redundancies, a data cleaning process was performed. Several substrings, such as "REF," "REF.," "REFR," and others, were identified and removed from the dataset. This cleaning process aimed to enhance the accuracy of subsequent analyses.

Frequency	Word
227955	mg
203484	ml
101668	ref
65724	amp
32466	tab
32293	de
32121	gr

It converts the value to uppercase, removes trailing spaces and periods, and splits the value into a list of words. It then joins the words back together, removes any asterisks ('*'), and replaces certain substrings from the "removelist" with an empty string. The resulting modified value is added to the "newset" set.

After processing all the values, the code determines the number of unique elements in the "newset" set and prints the count.Next, the code converts the "newset" set into a list, sorts it in alphabetical order, and compares the strings by removing spaces.

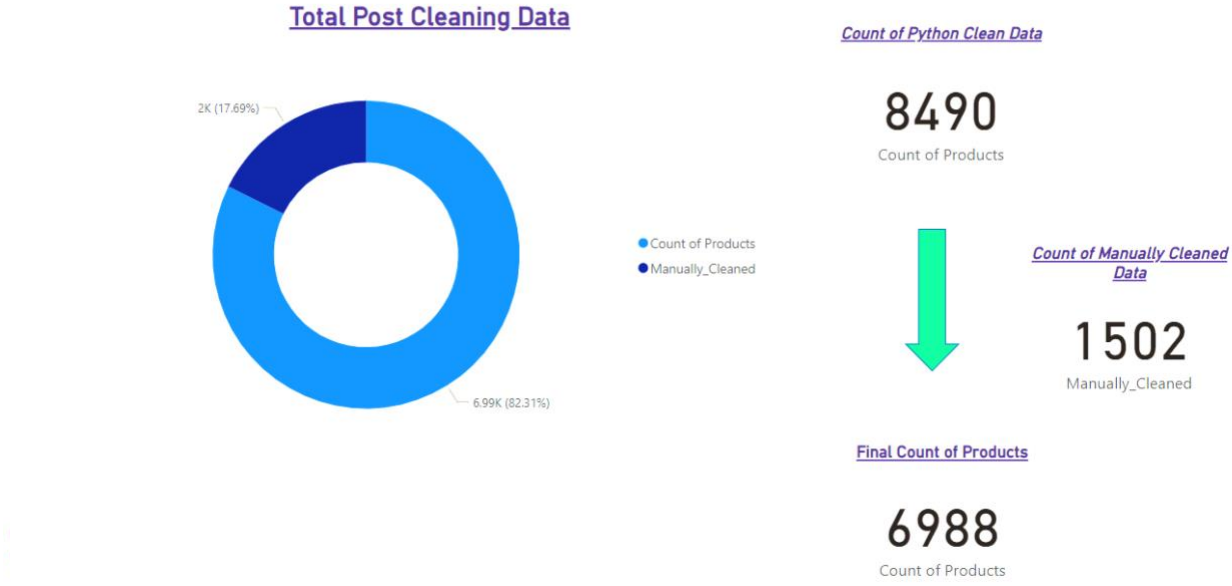
Check the code below.

```
import pandas as pd
from datetime import datetime


# Read CSV file into a Pandas DataFrame
df = pd.read_csv('product.csv',encoding= 'unicode_escape')

# Create a new set and lu to store elements
newset = set()
lu = {}

#removing the extra information that contains reference numbers
removelist = ['REF','REF.','REF ','REFR']
for value in df.iloc[:,0]:

    temp = value.upper().rstrip(" .").split()
    temp = ' '.join(temp)
    temp = temp.replace("*","")
    temp.replace

    for i in removelist:

        index = temp.find(i)
        if(index > 3):
            temp = temp[:index]

    newset.add(temp)

print(len(newset),"elements after set")

newlist = list(newset)
newlist.sort()
# comparing strings by removing spaces
for i in newlist:
    value = "".join(i.split())
    if (value in lu):
        continue
    else:
        lu[value] = i
print("after dictionary",len(lu))

#writing output in file
file = open("output_mod_1.csv","w")
for i in lu:
    print(lu[i])
    file.write(lu[i]+"\n")
```

```
8992
8992 elements after set
after dictionary 8541
100080983 DUOVISC 0,55 ML
1010129123D CONCENTRADO ACIDO DIALOG
2400291 DICHLOREX SOL 2% COLOREADA 125CC
```

4.6 Final Dataset Size

After the data cleaning and preprocessing steps, the dataset was reduced to 8,490 records. Additionally, some manual cleanup was conducted, resulting in approximately 7,000 records that were considered for further analysis and barcode identification.



5. Methodology and Implementation

5.1 Selenium

Selenium is an invaluable tool for web scraping projects due to its versatility and robust capabilities. As a powerful browser automation framework, Selenium allows developers to interact with websites just as a human user would, enabling them to extract data from dynamic web pages with ease. With its support for multiple programming languages, Selenium provides flexibility and accessibility to developers of various backgrounds. By simulating user actions such as clicking buttons, filling out forms, and navigating through pages, Selenium ensures that web scraping scripts can effectively handle JavaScript-driven content and AJAX requests. Its ability to handle complex web interactions, coupled with features like headless browsing and proxy support, makes Selenium an indispensable choice for web scraping projects, empowering developers to extract valuable data from modern websites efficiently and accurately.

5.2 Target Websites and Selection Criteria

For this project, two specific websites, namely "**farmaciasespecializadas**" and "**farmaciagloria**," were selected as the target websites for data extraction. The selection of these websites was based on the following criteria:

Relevance to the Problem Statement: The selected websites were relevant to the problem statement of extracting barcode information and manufacturer details of the products.

Availability of Desired Data: The target websites were chosen because they were known to provide the desired data, specifically barcode information and manufacturer details. Through careful examination and prior knowledge, it was determined that these websites maintain accurate and up-to-date data related to pharmaceutical products.

Website Reliability and Credibility: The selected websites were established and reputable sources in the pharmaceutical industry. They have a proven track record of providing reliable and trustworthy information. The reliability and credibility of the websites were important factors considered during the selection process.

Accessibility and Data Structure: The target websites were evaluated for their accessibility and the structure of the data. It was ensured that the websites allowed programmatic access and provided data in a structured format, facilitating the extraction process. This ensured that the required data could be efficiently retrieved and processed.

5.3 Data Extraction Techniques

5.3.1 Selenium class for Farmacias Especializadas

Selenium is a web automation tool that can be used to interact with websites and retrieve information. In this case, a Selenium class was used to automate interactions with the Farmacias Especializadas website. This class contains various methods that can be used to navigate to specific pages, input search queries, and extract information from the website. Approximately 75% of the overall scarped data was found utilizing the Farmacias Especializadas

A step by step break down of the process for the scraping algorithm is as follows: First, the input CSV file is read into a Pandas DataFrame. Then, the test is set up, and the page will wait for it to load. Next, each element in the DataFrame, starting from the specified start line, is processed. The element is searched on the website using the

searchElement() method, and if the search is unsuccessful, the page is refreshed before retrying. If one or more search results are found, the similarity score between each result and the input element is calculated using the surityscore() function. If the maximum similarity score meets the required threshold, the company name, barcode, and full name of the result are retrieved using the getDetails() method. The extracted information, along with the normalized input element, is written to the output CSV file. If no search results are found, only the normalized input element is written to the output file. The page is refreshed if the refresh interval has passed. Finally, the output file is closed, and the test is terminated.

```
        try:
            test.searchElement(element)
        except:
            try:
                print("search not found retrying going back  ...")
                test.driver.back()
                test.driver.refresh()
                test.searchElement(element)
            except:
                time.sleep(3)
                test.searchElement(element)

        fullnames = test.findElementFromList()
        similar =None
        print(f"Line {startline+count-1}: ",end = "")
        """
            if any items are found, find the item with
            maximum surity and checked if it is greater than
            required similarity.
        """
        if fullnames:

            surity_indexes = [surityscore(element,target) for target in fullnames]
            max_surity_indexes = max(surity_indexes) if len(fullnames)>0 else 0
            if max_surity_indexes >= required_similarity:
                similar = surity_indexes.index(max_surity_indexes)
        """
            if any max surity element is found ,
            retrieve the company name , barcode ,
            fullname found and store in the output file in csv format
        """
        if similar!= None:
            test.clickTile(similar)
            time.sleep(1)
            upc,company = test.getDetails()
            found +=1
            fullnames[similar] = " ".join(fullnames[similar].split(","))
            company = " ".join(company.split(","))
            print(f"{normalize(element)},{normalize(fullnames[similar])},{upc},{normalize(company)},{max_surity_indexes
            outfile.write(f"{normalize(element)},{normalize(fullnames[similar])},{upc},{normalize(company)},{max_surity
        else:
            print(element,"Notfound")
            outfile.write(f"{normalize(element)}\n")

        if count % refresh == 0:
            print(f"\n ******** {found} found, out of {count} ********\n")
            test.driver.refresh()
    # close file and driver
    outfile.close()
    test.teardown_method(None)
except Exception as e:
    print(e)
    outfile.close()
```

5.3.2 Selenium class for Farmacias Gloria

Similar to the first web scraping that was completed above the code below utilized the selenium package. The output of this scraping resulted in less products from the dataset in comparison to the first website. Approximately 25% of the overall scarped data was found utilizing the Farmacias gloria website.

A notable advantage of this website is that it is a traditional multipage HTML website which gives a single data response synchronously. This website is considerably good for scraping because there were no frequent unprecedented changes or updates. No technical problems were faced in scaping this site. The overall procedure in the algorithm remains the same as the website above with the exception of the website utilized to complete the data retrieving.

```
from playsound import playsound
import pandas as pd
inputfile = "bannu_gloria_input.csv"
outfile = open("bannu_gloria_output.csv",'a')
startline = 5
refresh = 1
required_similarity = 63
newset = set(['ENSURE ADVANCE SABOR 237ML',"ENSURE ADVANCE ACTIVE SABOR CHOCOLATE 237ml","ENSURE FOS POLVO SABOR"])
# Read CSV file into a Pandas DataFrame
df = pd.read_csv(inputfile,header=None)
total_len = len(df.index)

scraper = TestGloria()
scraper.setup_method(None)
scraper.initialSetup()
scraper.driver.implicitly_wait(3)

# try:
count = 0
found = 0
for element in df.iloc[startline - 1:,0]:
#     for element in newset:
    count +=1
    scraper.search(element)
#         time.sleep(2)
    fullnames = scraper.getElements()
    similar = None
    product_name = None
    ## do similarity things and store similar into similar
    print(f"Line {startline+count-1}: ",end = "")
    if fullnames:
#             print([i.text for i in fullnames])
        surity_indexes = [surityscore(element,target.text) for target in fullnames]
        max_surity_indexes = max(surity_indexes) if len(fullnames)>0 else 0
#             print(max_surity_indexes)
        if max_surity_indexes >= required_similarity:
            similar = surity_indexes.index(max_surity_indexes)
            product_name = fullnames[similar].text

    if product_name!= None:
        fullnames[similar].click()
        time.sleep(1)
        upc,company = scraper.getBarcode(),scraper.getCompanyName()
        found +=1
        product_name = " ".join(product_name.split(","))
        company = " ".join(company.split(","))
        print(f"{normalize(element)},{normalize(product_name)},{upc},{normalize(company)},{max_surity_indexes}")
        outfile.write(f"{normalize(element)},{normalize(product_name)},{upc},{normalize(company)},{max_surity_indexes}\
    else:
        print(element,"Notfound")
        outfile.write(f"{normalize(element)}\n")
    if count % refresh == 0:
        print(f"\n ******** {found} found, out of {count} , total {total_len} ********\n")
#             scraper.driver.refresh()
outfile.close()
scraper.teardown_method(None)
```

5.4 Core functionality: Search Algorithm

5.4.1 Data normalization

Data normalization is the process of converting data from one format to another to make it consistent and easier to work with. In this case, the data extracted from the websites contained various special characters and diacritics that needed to be normalized. For example, characters like Â, Ć, and Ê were converted to A, C, and E respectively.

Extracting tablet type and count info combinedly that have con,c/,tabcount: This step involves extracting information about the type and count of tablets from the tablet description. This information is typically stored in a format like "X CON Y TAB" or "X/C Y TAB". The code uses regular expressions to extract this information and store it in a dictionary.

```
import unicodedata

## used to normalize data
## like Â ,Ć ,Ê , to A , C , E

def normalize(string):
    return unicodedata.normalize('NFKD', string).encode('ASCII', 'ignore').decode('ASCII')
```

```
import re
def Tab_type_details(string):

    tablet_type_features = []
    ## most found tablets types are stored in below dictionary
    tab_types = {
                  "CREAM":["CREAM","CREMA","POMADA","OINTMENT"],
                  "DROPS":["GTS","GOTAS"],
                  "BOTTLE":["FRESCO","FRASCO","FRASCO AMPOLLA","SOL"],
                  "TABLET":["TABLETAS","TABLETS","TABLET","TABS","TAB","GRAGEAS"],
                  "POWDER" :["SOBRES","SOBRE"],
                  "INJECTION":["INY","I[.]V.","I[.]V","I V","IV","AMP","AMPOLLETAS","AMPOLLA","AMPULA",\
                               "VIAL","VIALS","JER","FA","F[.]A","F A","F[.]A[.]"],
                  "SUPO":["SUPOSITORIS","SUPPOSITORIES","SUPOSITORIO","SUPPOSITORIES"],
                  "CAPSULE":["COMPRIMIDOS","COMPR","COPR","COMP","CPR","CAPSULES","CAPSULAS","CAPS","CAP"]
                 }

    ## Extracting tablet type and count info combinely that have con,c/,tabcount "
    for tab_type in tab_types.keys():
        for name in tab_types[tab_type]:
            found = re.findall(f"C?O?N?/? ?\d+(?:\.\d+)? ?{name}.? ?", string)
            for i in found:
                count = re.findall("(\d+(?:\.\d+)?)",i)[0]
                tablet_type_features.append({"raw_string":i,"count":count,"found_name":name,"type":tab_type})
                string = string.replace(i," ")
    return tablet_type_features,string
```

5.4.2 Fine tuning

The code utilizes a dictionary named **tab_types** to establish a mapping between common tablet types and their respective variations found in the tablet description. This mapping enables the identification of tablet types like "CREAM," "CREMA," "POMADA," or "OINTMENT" within the description. By iterating through each tablet type in the **tab_types** dictionary, the code applies regular expressions to locate matches in the tablet description. When a match is found, the extracted tablet type, along with the raw string and type category, is added to the **extra_features** dictionary. Furthermore, the original string is modified by replacing the extracted tablet type with a

space. The code also employs regular expressions to identify patterns indicative of tablet counts in the description. Upon extracting the count value, it is added to the **extra_features** dictionary along with the raw string. Similar to the tablet type extraction, the count string is substituted with a space in the original string. Additionally, the code defines a list named **COMMON_TERMS** comprising common terms encountered in tablet descriptions (e.g., "CAJA" for box or "CON" for with). By iterating through the **COMMON_TERMS** list, the code searches for matches in the tablet description using regular expressions. Any common terms are appended to the **extra_features** dictionary, and the original string is altered by replacing the respective term with a space. Finally, the code returns the **extra_features** dictionary containing the extracted tablet types, counts, and common terms, along with the modified string that excludes the extracted features.

```
            }
COMMON_TERMS = ["CAJA","CON","PISA"]

## to check for types
for tab_type in tab_types.keys():
    for name in tab_types[tab_type]:
        found = re.findall(f" {name} ", string)
        if not found:
            found = re.findall(f" {name}\Z",string)
        for i in found:
            extra_features["types"].append({"raw_string":i,"found_name":name,"type":tab_type})
            string = string.replace(i," ")


# to check for attributes like tabtype and tabcount individually

found = re.findall("C?O?N?/? ?\d+(?:\.\d+)?",string)
for i in found:
    string = string.replace(i," ")
    count = re.findall("(\d+(?:\.\d+)?)",i)[0]
    extra_features["counts"].append({"raw_string":i,"count":count})

# extracting some common terms and extra info from the tablet
for name in COMMON_TERMS:
    found = re.findall(f" {name} ", string)

    if not found:
        found = re.findall(f" {name}\Z",string)
    for i in found:
        extra_features["common"].append(i)
        string = string.replace(i," ")
return extra_features,string
```

5.4.3 Extracting additional information

In addition to the tablet type and count, the code also extracts other information like the weight, composition, and volume of the tablets. This information is extracted using regular expressions and stored in a separate dictionary.

```
    for key, i in enumerate(["compositions","weights","volumes"]):
        details[i]=CWV[key]
    details["tab_type_details"],string= Tab_type_details(string)
    extrafeatures,string = extractmorefeatures(string)
    details["name"] = findName(text,CWV,details["tab_type_details"],extrafeatures)
    string = string.replace(details["name"],"")
    string = (" ".join(string.split())).strip()
    string = cleanstring(string)
    details["extrafeatures"] = extrafeatures
    for quant in ["weights","volumes","compositions"]:
        for key,i in enumerate(details[quant]):
            details[quant][key] = standardize(i)
    types = set()
    counts = set()
    for detail in details["tab_type_details"]:
        if len(details["tab_type_details"]) != 0:
            types.add(detail["type"])
            counts.add(detail["count"])
    for detail in details['extrafeatures']['types']:
        types.add(detail["type"])
    for detail in details['extrafeatures']['counts']:
        counts.add(detail["count"])
    details["features"] = {"types":types,"counts":counts,"extra":set(string.split())}
    details.pop('tab_type_details')
    details.pop('extrafeatures')
    return details

Extract_Details("NIQUITIN 21 MG C/7 PARCHES CON PARA")
Extract_Details("EXFORGE HCT 10MG/320MG/25MG CON 28 COMPRIMIDOS TAB")
Extract_Details("SIMPLE 50 ML")
```

```
{'compositions': [],
 'weights': [],
 'volumes': ['50ML'],
 'name': 'SIMPLE',
 'features': {'types': set(), 'counts': set(), 'extra': set()}}
```

5.4.4 Similarity measurement

To compare different tablets and determine if they are the same, the code uses a similarity measurement function. This function compares various attributes like the tablet name, weight, composition, and volume to calculate a similarity score between two tablets. This score is a value between 0 and 1, with higher values indicating greater similarity.

Search and match: Finally, the code searches a CSV file containing a list of known tablets and tries to match the extracted tablets to this list. It does this by calculating the similarity score between each extracted tablet and each tablet in the list, and selecting the one with the highest score. If the score is above a certain threshold, the extracted tablet is considered a match and its details (company name, barcode, etc.) are stored in an output file. If no match is found, the code tries again by going back to the home page and searching for the tablet again.

```
def similarity_type(target_type,src_type):

    """
        This Function considers correlation between two types to
        measure similarity between two types and return output as
        value between 0 and 1

    """
    similarity_dict ={
        "SUPOCREAM" : 0.24,
        "SUPODROPS" : 0.3,
        "SUPOBOTTLE" : 0.24,
        "SUPOTABLET" : 0.35,
        "SUPOPOWDER" : 0.29,
        "SUPOINJECTION" : 0.2,
        "SUPOCAPSULE" : 0.39,
        "CREAMDROPS" : 0.59684736728668213,
        "CREAMBOTTLE" : 0.5013158917427063,
        "CREAMTABLET" : 0.33642226457595825,
        "CREAMPOWDER" : 0.48533108830451965,
        "CREAMINJECTION" : 0.49305346608161926,
        "CREAMCAPSULE" : 0.3910931646823883,
        "DROPSBOTTLE" : 0.7995974540710449,
        "DROPSTABLET" : 0.603466272354126,
        "DROPSPOWDER" : 0.583532989025116,
        "DROPSINJECTION" : 0.6436945199966431,
        "DROPSCAPSULE" : 0.6268869638442993,
        "BOTTLETABLET" : 0.5191242694854736,
        "BOTTLEPOWDER" : 0.6567681431770325,
        "BOTTLEINJECTION" : 0.92448949813843,
        "BOTTLECAPSULE" : 0.5893133878707886,
        "TABLETPOWDER" : 0.5272809267044067,
        "TABLETINJECTION" : 0.5583986043930054,
```

6: Results and findings

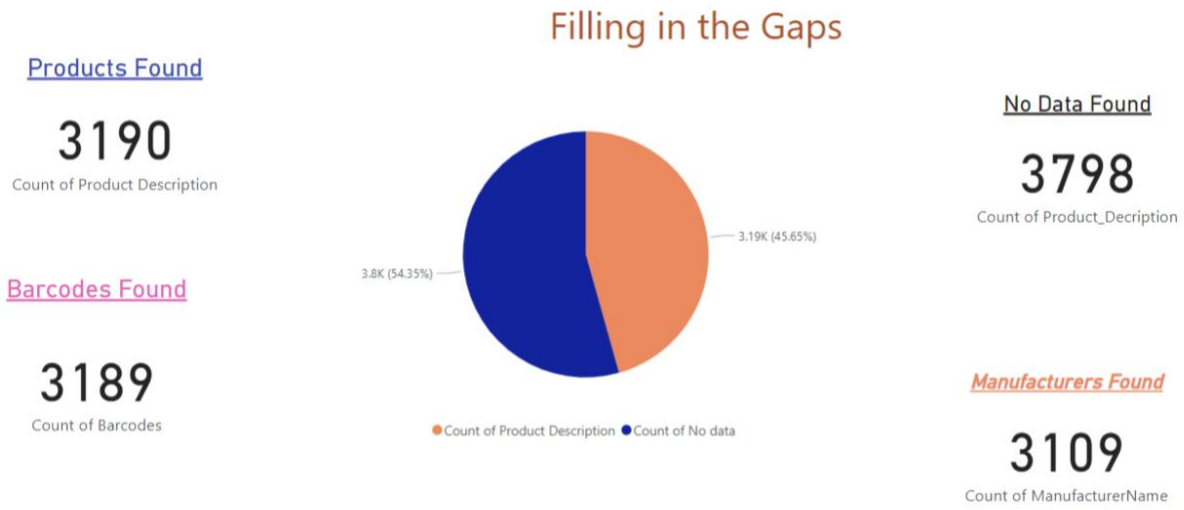
The objective of the project was to scrape **Barcodes** and **Manufacturer Names** and clean redundant data from a given source. The initial data included **391,201** total products and **346,947** manufacturer names. The data was then subjected to a cleaning process, which was performed in two ways: Using Python and manual cleaning.

Data Cleaning Process:

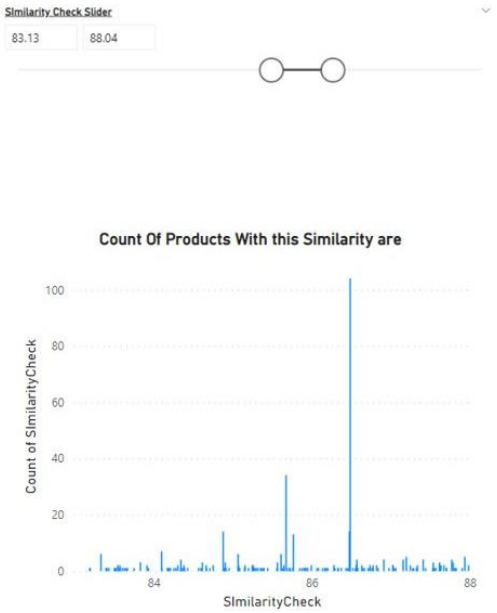
The Python cleaning process yielded an output of **8,490** products, while the manual cleaning process eliminated **1,502** products from the python output. The final total after cleaning was 6988 products.

The Results of the Scraping Process:

The correct number of scraped products was **3190**, with **3189 Barcodes** found and **3109 Manufacturer Names** identified. However, **3798** products were left for scraping.



A Power BI report was utilized to visualize the output of the python induced similarity checker. To visualize each product a table was used with one column containing product description received and the other containing the product description received through the web scraping algorithm. A slider was then utilized with the minimum number set to 63% and the maximum of a 100%. The slider can be utilized to set a percentage for the needed similarity percentage and the table can be used to visualize the set after products. Another visualization seen on this report is a graph that keeps track of the count of products with a specific similarity check. All three visuals work in unison with one another, dependent on one another and constantly changing as indicators for any given one of them changes. Furthermore, these visuals are sourced with a CSV Source which deems them re-useable for any change in future data.



7: Conclusion

7.1 Summary of the Project

In order to fill in the gaps for the data library majority of the bar code information for pharmaceuticals was located utilizing two source websites. The websites that aided in this process were namely ‘Farmacias Especializadas’ and ‘Farmacias Gloria’. In addition to finding barcode information, the accuracy of multiple product descriptions was validated and missing manufacturer names were restored. The success of this overall project consisted of data analysis, pre-processing, web scrapping, quality checking and finally a visual report that was manufactured via PowerBi.

7.2 Achievement of Goals

The overall aim of this project was three-fold. First to locate bar code information, second to update the out-of-date product description and finally to find missing manufacturer names. During this process; 3109 manufacturer names were found, 3190 product descriptions were updated and 3189 barcode records were found. ‘Farmacias Especializadas’ and ‘Farmacias Gloria’ were utilized to find the 3186 barcode records.

7.3 Recommendations for Future Work

The results of the scraping process were satisfactory, with the majority of products correctly identified and their corresponding barcodes and manufacturer names extracted. The remaining products left for scraping can be addressed in future work. In addition, it is recommended to focus on “URC”; Use (Using the search algorithm), Reuse (Reusing the data cleaning script) and most importantly Create (Creating a master data collection for cleaner inputs).

8: Limitations

Some notable limitations were namely; unavailability (“unavailability”: indicates the missing required information that could not be located on the websites for the scraped products), Incompleteness or incorrectness (“Incompleteness \ Incorrectness“: indicates the information provided in the given data set), duplicate data (presence of duplicate data in the data set) and unprecedented website changes and difficulty identifying complete(competent) websites to utilize.