Procurement POC

Main Brief

A large manufacturing company's Procurement department is kicking off a digitalization journey. Their category managers have hit a wall – they can't properly analyze spend because their supplier database is cluttered with messy, duplicate, and outdated entries. Meanwhile, leadership is pushing hard for a clear cost-saving strategy for next year. On top of that, there's interest in exploring sustainability in the supply chain, but they just don't have the resources to prioritize it right now.

Summary

We have a csv file named **presales_data_sample.csv**. In it, we have 2951 rows, each representing a result for a company search query. Each search query has 5 companies attached to it (5 rows). The purpose of the POC is to build a matching algorithm for every search query, to keep the company results with a high enough score, to discard the others if no good match is found, and to clean the final data and present it.

Building the POC

Building this POC was done using the Anaconda/Python stack. The first phase of the build was done using the Spyder environment. However, due to the constraints of needing to present the work in a step-by-step manner, I found Spyder woefully inadequate at this. As a consequence, creating this file using Jupyer Notebook is the second phase of the build. If I had more time available, I would've started the entire project in Jupyter Notebook. Sadly, I haven't used it before, and had no idea how useful it can be. Oh well, I'll keep this in mind for the future.

Dataset & preparation

We make a python file called **pre_processing.py** . This file will be used to do all the work necessary to create the dataset with companies that will be analyzed and sent to the client.

Importing all the necessary libraries in python

In [ ]:
```
import os
import chardet
import pandas as pd
import numpy as np
import Levenshtein as levenshtein
from difflib import SequenceMatcher as SM
```

Checking the integrity of the data

Immediately after opening the file **presales_data_sample.csv** in Notepad, we notice a problem. The cells are separated by commas, and there are additional commas in multiple fields. We have a solution for this, which I have successfully used in the past: we download LibreOffice (the open source alternative to the Office suite) and use LibreCalc to open the file. LibreCalc, unlike Excel, has a very user-friendly way to change the cell separator. We change it from comma (,) to semicolon (;). However, before that, we do need to check if there are any semicolons in the file. For that, we import the file in python using the following code:

In [ ]:
```
# Open the file in read mode ('r')
with open('presales_data_sample.csv', 'r') as file:
    # Read the entire content into a string variable
    csv_as_text = file.read()
    count = file.read().count(';')
```

It gives an error due to the file encoding. Therefore, we need to check the file encoding using the following code:

In [ ]:
```
# Open in binary mode ('rb')
with open('presales_data_sample.csv', 'rb') as file:
    # Read the first 1,000,000 bytes for a quick check
    raw_data = file.read(1000000)
    result = chardet.detect(raw_data)
```

We find out that the encoding is **utf-8**, therefore we modify the initial code to check for semicolons as such:

In [ ]:
```
# Open the file in read mode ('r')
with open('presales_data_sample.csv', 'r', encoding='utf-8') as file:
    # Read the entire content into a string variable
    csv_as_text = file.read()
    count = file.read().count(';')
```

There are no semicolons anywhere in the file. Perfect. It means that we can go to the next step, that of opening the file **presales_data_sample.csv** in LibreCalc, and changing the separators from commas to semicolons. We visually inspect the file after doing it, checking for any discrepancies. None are found, so we move on.

Importing the data in python

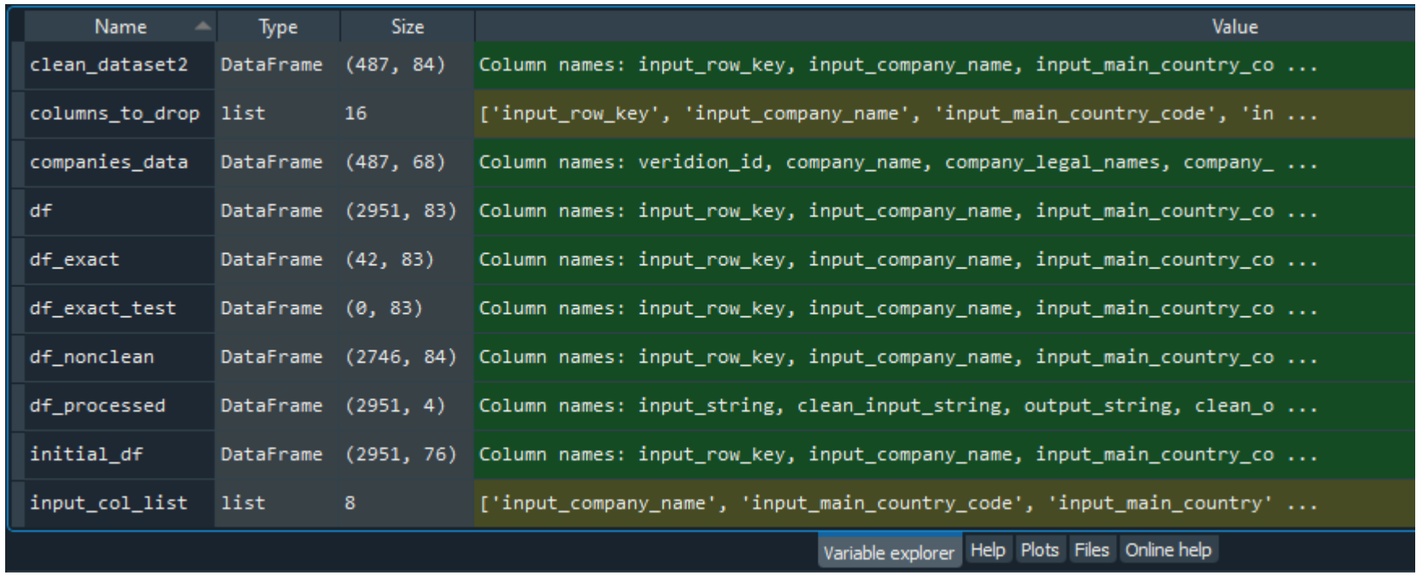
We import the **presales_data_sample.csv** dataset in python as a pandas dataframe.

We do not forget about specifying both the separators and the encoding:

In [ ]:
```
initial_df = pd.read_csv('presales_data.csv', sep=';', encoding='utf-8')
df = pd.read_csv('presales_data.csv', sep=';', encoding='utf-8')
```

df is the dataset that we will work on. We also keep **initial_df** as a copy in memory as a variable, just in case we will need to make visual inspections of it in the future.

Sneak peak of how messy the Spyder variable explorer can end up looking when you're trying to juggle everything everywhere all at once:



Name	Type	Size	Value
clean_dataset2	DataFrame	(487, 84)	Column names: input_row_key, input_company_name, input_main_country_co ...
columns_to_drop	list	16	['input_row_key', 'input_company_name', 'input_main_country_code', 'in ...
companies_data	DataFrame	(487, 68)	Column names: veridion_id, company_name, company_legal_names, company_ ...
df	DataFrame	(2951, 83)	Column names: input_row_key, input_company_name, input_main_country_co ...
df_exact	DataFrame	(42, 83)	Column names: input_row_key, input_company_name, input_main_country_co ...
df_exact_test	DataFrame	(0, 83)	Column names: input_row_key, input_company_name, input_main_country_co ...
df_nonclean	DataFrame	(2746, 84)	Column names: input_row_key, input_company_name, input_main_country_co ...
df_processed	DataFrame	(2951, 4)	Column names: input_string, clean_input_string, output_string, clean_o ...
initial_df	DataFrame	(2951, 76)	Column names: input_row_key, input_company_name, input_main_country_co ...
input_col_list	list	8	['input_company_name', 'input_main_country_code', 'input_main_country' ...

Variable explorer | Help | Plots | Files | Online help

Preparing for matching

In order to create a matching algorithm between input values and output companies, we need to consider several things:

1. We have 8 input columns with data
2. We have 3 different output columns for company names. For convenience and speed of prototyping, we will use the **company_name** column and discard the others, even if we lose information by doing so
3. There are many matching algorithms for strings. For convenience, we will use two of them, the **Levenshtein distance** and the **Gestalt pattern matching**

For more info on fuzzy string matching, check the following links:

Fuzzy string matching in python (https://typesense.org/learn/fuzzy-string-matching-python/)

Levenshtein distance (https://en.wikipedia.org/wiki/Levenshtein_distance)

Levenshtein distance is a string metric for measuring the difference between two sequences by calculating the minimum number of single-character edits—insertions, deletions, or substitutions—required to transform one string into another

Gestalt pattern matching (https://en.wikipedia.org/wiki/Gestalt_pattern_matching)

Gestalt pattern matching, also Ratcliff/Obershelp pattern recognition, is a string-matching algorithm for determining the similarity of two strings. It was developed in 1983 by John W. Ratcliff and John A. Obershelp and published in the Dr. Dobb's Journal in July 1988.

Processing the inputs and the outputs to match

We first create a list of all columns in the dataset, then a list of input columns, and then we make a dictionary mapping in order to map them with output columns.

In [ ]:
```
#create a list of all columns
col_list = df.columns.tolist()

# create input_cols list
input_cols = [col for col in df.columns if 'input' in col]

# Convert col_list to a set for O(1) lookup speed
col_match_set = set(col_list)

# Create a dictionary mapping: {'input_NAME': 'NAME'}
col_matches = {col: col[6:] for col in input_cols if col[6:] in col_match_set}
```

We then build an input string and an output string for comparison.

We do this by creating a mask between the input column list and output column list, which we take from the dictionary.

We also need to remove any NaN values and not take them into account. The problem that we have is that:

1. Not all input columns have values
2. Not all matching / output columns have values The values which have no equivalent on the other side need to be disregarded when building the matching string.

This is accomplished in the following way:

In [ ]:
```
# Build input and output string for comparison
# Make list for input and output cols matched
input_col_list = list(col_matches.keys())
output_col_list = list(col_matches.values())

# Check if input is valid AND output is valid.
# We use .values to ignore column names and compare by position.
valid_mask = df[input_col_list].notna().values & df[output_col_list].notna().values

# Apply mask: Keep valid values, replace invalid pairs with empty string ''
clean_inputs = np.where(valid_mask, df[input_col_list], '')
clean_outputs = np.where(valid_mask, df[output_col_list], '')
```

What does the above code achieve?

If input company address has a value (let's say 'Street X'), but the output company address has no value, then the company address string will not be concatenated to the input string. This is valid for all other equivalent columns.

Concatenating the resulting strings, building the input and output comparison strings

In [ ]:
```
# Concatenate the cleaned arrays row-by-row
# We wrap in DataFrame just to use the convenient .agg method
df['input_string'] = pd.DataFrame(clean_inputs).astype(str).agg(''.join, axis=1)
df['output_string'] = pd.DataFrame(clean_outputs).astype(str).agg(''.join, axis=1)
```

Nice. We now have raw strings to compare. However, we still need to process them:

1. Transform all letters to lowercase
2. Remove any whitespaces
3. Remove any commas, dots and punctuation marks

This process will somewhat lower the informational content of the input. However, it can be seen as a tradeoff, since many companies can have punctuation marks in their Brand name but not in their Official name, and many similar cases can appear. For convenience and speed, it's best we do this at this stage.

There are many ways to accomplish the above, one of the most straightforward ways can be using a regular expression, which is what we're doing with the following code:

In [ ]:
```
# Regex explanation:
# \W matches anything that is NOT a letter, number, or underscore (including whitespace & punctuation)
# _ matches the underscore explicitly (often treated as a word char, so we add it to remove it)
df['clean_input_string'] = df['input_string'].str.lower().str.replace(r'[\W_]+', '', regex=True)
df['clean_output_string'] = df['output_string'].str.lower().str.replace(r'[\W_]+', '', regex=True)
```

We also create new dataframe from the processed columns just to visually check them

In [ ]:
```
#create new dataframe from processed columns just to visually check them
df_processed = df[['input_string','clean_input_string','output_string','clean_output_string']]
```

It should look something like this:

df_processed - DataFrame

Index	input_string	clean_input_string	output_string	clean_output_string
0	24-SEVEN MEDIA NETWORK (PRIVATE) LIMITEDPKPakistanSindhKarachi	24sevenmedianetworkprivatelimitedpkpakistan…	New Millennium NetworkPKPakistanSindhKarachi	newmillenniumnetworkpkpakistansindhkarachi
1	24-SEVEN MEDIA NETWORK (PRIVATE) LIMITEDPKPakistanSindhKarachi	24sevenmedianetworkprivatelimitedpkpakistan…	Private Helipad Ali VillaPKPakistanSindhKarachi	privatehelipadalivillapkpakistansindhkarachi
2	24-SEVEN MEDIA NETWORK (PRIVATE) LIMITEDPKPakistanSindhKarachi	24sevenmedianetworkprivatelimitedpkpakistan…	24seven Research NetworkINIndiaHaryanaFaridabad	24sevenresearchnetworkinindiaharyanafaridab…
3	24-SEVEN MEDIA NETWORK (PRIVATE) LIMITEDPKPakistanSindh	24sevenmedianetworkprivatelimitedpkpakistan…	EmerioSoftPKPakistanSindh	emeriosoftpkpakistansindh
4	24-SEVEN MEDIA NETWORK (PRIVATE) LIMITEDPKPakistanSindhKarachi	24sevenmedianetworkprivatelimitedpkpakistan…	Asiatic Public Relations NetworkPKPakistanSindhKarachi Division	asiaticpublicrelationsnetworkpkpakistansind…
5	2OPERATE A/SDKDenmarkNorth Denmark RegionAalborg9220Niels Jernes Vej10	2operateasdkdenmarknorthdenmarkregionaalbor…	2operateDKDenmarkNorth Denmark RegionAalborg Municipality9220Niels Jernes Vej10	2operatedkdenmarknorthdenmarkregionaalborgm…
6	2OPERATE A/S	2operateas	CO 2 Operate	co2operate
7	2OPERATE A/SDKDenmarkNorth Denmark RegionAalborg9220Niels Jernes Vej10	2operateasdkdenmarknorthdenmarkregionaalbor…	ConzentrateDKDenmarkCapital Region Of DenmarkHerlev2730Lyskær8B	conzentratedkdenmarkcapitalregionofdenmarkh…
8	2OPERATE A/SDKDenmarkNorth Denmark RegionAalborg9220Niels Jernes Vej10	2operateasdkdenmarknorthdenmarkregionaalbor…	2A Pharma ApS.DKDenmarkNorth Denmark RegionAalborg9220Niels Jernes Vej10	2apharmaapsdkdenmarknorthdenmarkregionaalbo…
9	2OPERATE A/SDKDenmarkNorth Denmark RegionAalborg9220Niels Jernes Vej10	2operateasdkdenmarknorthdenmarkregionaalbor…	RenSams SolutionsDKDenmarkNorth Denmark RegionAalborg9220Niels Jernes Vej10	rensamssolutionsdkdenmarknorthdenmarkregion…
10	ACCENTURE SERVICES ASNONorwayVikenBærum	accentureservicesasnonorwayvikenbærum	AccenturesIEIrelandLeinsterDublin 2	accenturesieirelandleinsterdublin2
11	ACCENTURE SERVICES ASNONorwayVikenBærum1364Rolfsbuktveien2	accentureservicesasnonorwayvikenbærum1364ro…	Accenture Bratislava Campus OfficeSKSlovaki…	accenturebratislavacampusofficeskslovakiare…
12	ACCENTURE SERVICES ASNONorwayVikenBærum1364Rolfsbuktveien	accentureservicesasnonorwayvikenbærum1364ro…	AccentureMUMauritiusMauritiusQuatre Bornes72208Ebene City Boulevard	accenturemumauritiusmauritiusquatrebornes72…
13	ACCENTURE SERVICES ASNONorwayVikenBærum1364Rolfsbuktveien2	accentureservicesasnonorwayvikenbærum1364ro…	Accenture NorgeNONorwayOsloOslo158Rådhusgata27	accenturenorgenonorwayoslooslo158rådhusgata…
14	ACCENTURE SERVICES ASNONorwayVikenBærum1364Rolfsbuktveien2	accentureservicesasnonorwayvikenbærum1364ro…	AccentureCACanadaQuebecMontrealH3B 2G2Esplanade Place Ville Marie5	accenturecacanadaquebecmontrealh3b2g2esplan…
15	COMBA TELECOM LIMITEDHKHong KongHong KongHong Kong Island	combatelecomlimitedhkhongkonghongkonghongko…	Comba Telecom Inc.USUnited StatesCaliforniaMilpitas	combatelecomincusunitedstatescaliforniamilp…
16	COMBA TELECOM LIMITEDHKHong KongHong KongHong Kong Island	combatelecomlimitedhkhongkonghongkonghongko…	Vasav TelecomHKHong KongHong KongKwai Chung	vasavtelecomhkhongkonghongkongkwaichung
17	COMBA TELECOM LIMITEDHKHong KongHong KongHong Kong Island	combatelecomlimitedhkhongkonghongkonghongko…	MecadliberHKHong KongHong KongKowloon	mecadliberhkhongkonghongkongkowloon
18	COMBA TELECOM LIMITEDHKHong KongHong KongHong Kong Island	combatelecomlimitedhkhongkonghongkonghongko…	Comba TelecomHKHong KongHong KongNew Territories	combatelecomhkhongkonghongkongnewterritories
19	COMBA TELECOM LIMITEDHKHong KongHong KongHong Kong Island	combatelecomlimitedhkhongkonghongkonghongko…	CombaHKHong KongHong KongTai Po	combahkhongkonghongkongtaipo
20	Comengineering Sdn. Bhd.MYMalaysiaSelangorS…	comengineeringsdnbhdmymalaysiaselangorsuban…	Metro Com Engineering S.p.A.ITItalyPiedmont…	metrocomengineeringspaititalypiedmontgarbag…
21	Comengineering Sdn. Bhd.MYMalaysia	comengineeringsdnbhdmymalaysia	Adorea SystemsMYMalaysia	adoreasystemsmymalaysia
22	Comengineering Sdn. Bhd.MYMalaysiaSelangorS…	comengineeringsdnbhdmymalaysiaselangorsuban…	Seliyan Com. EngineeringMYMalaysiaSelangorP…	seliyancomengineeringmymalaysiaselangorpuch…
23	Comengineering Sdn. Bhd.MYMalaysiaSelangorS…	comengineeringsdnbhdmymalaysiaselangorsuban…	COMENGINEERING SDN BHD.MYMalaysiaSelangorSu…	comengineeringsdnbhdmymalaysiaselangorsuban…
24	Comengineering Sdn. Bhd.MYMalaysiaSelangorS…	comengineeringsdnbhdmymalaysiaselangorsuban…	COM Engineering Ltd.GBUnited KingdomWalesFlintshireCH4 8SBRiver Lane	comengineeringltdgbunitedkingdomwalesflints…
25	COMMSCOPE SOLUTIONS SINGAPORE PTE. LTD.SGSi…	commscopesolutionssingaporepteltdsgsingapor…	All Property Solutions SingaporeSGSingapore…	allpropertysolutionssingaporesgsingaporecen…
26	COMMSCOPE SOLUTIONS SINGAPORE PTE. LTD.SGSingaporeCentral68804Shenton Way2	commscopesolutionssingaporepteltdsgsingapor…	CommScope Solutions S Pte Ltd.SGSingaporeSo…	commscopesolutionsspteltdsgsingaporesouthwe…
27	COMMSCOPE SOLUTIONS SINGAPORE PTE. LTD.SGSi…	commscopesolutionssingaporepteltdsgsingapor…	CommScopeMXMexicoTamaulipasReynosa88780Aven…	commscopemxmexicotamaulipasreynosa88780aven…

Format Resize Background color Column min/max Save and Close Close

Great! We are now ready to test the matching algorithms and construct a matching score

Calculating Levenshtein distance and Levenshtein ratio

In [ ]:
```
#calculating levenshtein distance and ratio for processed strings
df['lev_distance'] = [levenshtein.distance(a,b) for a, b in zip(df['clean_input_string'], df['clean_output_string'])]
df['lev_ratio'] = [levenshtein.ratio(a,b) for a, b in zip(df['clean_input_string'], df['clean_output_string'])]
```

Great! Now let's check if there are any companies that have perfect matching between their strings - which means a Levenshtein distance of 0

In [ ]:
```
#get companies with exact name. It seems there are 42 of them. Great, I found the meaning of life.
df_exact = df[df['lev_ratio'] == 1]
```

Perfect. We have 42 companies with exact names.

We will drop them at the moment, since we want to keep working on the rest of the companies, the one with imperfect matching.

In [ ]:
```
#need to drop companies that have exact names
values_to_drop = df.loc[df['lev_ratio'] == 1, 'input_row_key']
df_nonclean = df[~df['input_row_key'].isin(values_to_drop)]

#testing if it dropped the correct ones
df_exact_test = df_nonclean[df_nonclean['lev_ratio'] == 1]
```

We also perform some tests, just to visually check around which Levenshtein ratio most of the companies could be kept in the final dataset, and get a mental estimation of a 'good enough' ratio. From past experience, 0.7 would be the minimum acceptable ratio. Any lower than that, and the matching is too weak and too many wrong results would be added to the final data. This 0.7 ratio is not purely scientific / methodological, it's just a rule of thumb based on previous usage.

In [ ]:
```
#counting number of rows with levenshtein distance larger than a few values
count_lev_9 = (df_nonclean['lev_ratio'] > 0.9).sum()
count_lev_8 = (df_nonclean['lev_ratio'] > 0.8).sum()
count_lev_7 = (df_nonclean['lev_ratio'] > 0.7).sum()
```

Name	Type	Size	
count_lev_7	int64	1	961
count_lev_8	int64	1	573
count_lev_9	int64	1	210

We are now ready to move to the next step.

Calculating similarity ratio based on Gestalt pattern matching

In [ ]:
```
#calculating similarity ratio based on Ratcliff/Obershelp string matching algorithm
df_nonclean['similarity_ratio'] = [SM(None, a, b).ratio() for a, b in zip(df_nonclean['clean_input_string'],
                                                                          df_nonclean['clean_output_string'])]
```

Doing the same tests as above, to visually check how many rows we have for different similarity ratios.

In [ ]:
```
#counting number of rows with similarity ratio larger than a few values
count_sim_9 = (df_nonclean['similarity_ratio'] > 0.9).sum()
count_sim_8 = (df_nonclean['similarity_ratio'] > 0.8).sum()
count_sim_7 = (df_nonclean['similarity_ratio'] > 0.7).sum()
```

count_sim_7	int64	1	922
count_sim_8	int64	1	551
count_sim_9	int64	1	209

We have similar results to the ones obtained with Levenshtein ratio. It checks out, since the Gestalt similarity ratio and Levenshtein ratios are similarily calculated. This is useful for verification.

We also made some tests for rows with similarity ratio larger than 0.8 . The code for that has been left commented out in the **pre_processing.py** file, as a backup in case the client would like a 'stronger' and more exact matching for companies, but which will leave more data out.

Filtering the dataset based on similarity ration larger than 0.7

We are finally ready to extract the companies which are more likely to match the inputs, with more than 0.7 similarity.

Firstly we need to filter the rows of the dataset based on similarity.

In [ ]:
```
#filtering the dataset by similarity ratio >= 0.7
sub_dfnonclean = df_nonclean[df_nonclean['similarity_ratio'] >= 0.7 ]
```

Secondly, we need to find the companies themselves - we need to group by index in the dataset, since the same input string may have more than one result with higher than 0.7 similarity.

For every input, we want a single company result.

In [ ]:
```
#finding companies with similarity ratio >= 0.7
idx = sub_dfnonclean.groupby('input_row_key')['similarity_ratio'].idxmax()
sub_dfsim = sub_dfnonclean.loc[idx]
```

We also remove some variables to reduce the mess in the variable explorer.

In [ ]:
```
#delete useless variables
del clean_inputs, clean_outputs, col_list, col_match_set, col_matches, idx, valid_mask, values_to_drop
```

Constructing the final dataset

At long last, we are going to build the final dataset which will be shown to the client. What we need to do is:

1. Merge the 42 companies which were an exact match with the companies with >= 0.7 similarity
2. Fill the 'similarity_ratio' column with value '1' where 'lev_ratio' is 1
3. Drop the processed columns, such as the strings we made the matching on
4. Generate the final csv file

In [ ]:
```
#merge dataset with exact companies and dataset with >= 0.7 similar companies
clean_dataset = pd.concat([df_exact, sub_dfsim], ignore_index = True)
clean_dataset.loc[clean_dataset['lev_ratio']==1, 'similarity_ratio'] = 1

#indexing the columns to drop in the final dataset
columns_to_drop = clean_dataset.iloc[:, np.r_[0:9, 76:83]].columns.tolist()

#sort the companies data by similarity ratio
companies_data = clean_dataset.drop(columns=columns_to_drop)
companies_data = companies_data.sort_values('similarity_ratio', ascending=False)

#generate the final csv file
companies_data.to_csv('companies_data.csv', index=False, sep=';', encoding='utf-8-sig')
```

Success! We also chose to use 'utf-8-sig' encoding for generating the file, just to be sure that special characters such as Swedish, Danish or German letters are encoded successfully.

We now have the **companies_data.csv** file, which has 487 companies out of the initial 590

103 inputs have remained unmatched, which gives a **82.5%** matching completeness

The focus is on the 82.5% matched companies, since the remaining tail was low quality input data.

Exploring the dataset

Data analysis and insight extraction

We make a python file called **post_processing.py** , in which we will do all the analysis on the data we have previously built.

Firstly, we import all the necessary libraries and the dataset:

In [ ]:
```
import os
import pandas as pd
import numpy as np
import plotly.express as px

initial_df = pd.read_csv('companies_data.csv', sep=';', encoding='utf-8-sig')
df = pd.read_csv('companies_data.csv', sep=';', encoding='utf-8-sig')
```

I realise I forgot to check if the companies are unique by 'veridion_id' column. Maybe the same company has appeared in different input search queries.

In [ ]:
```
#check if companies are unique by veridion_id column
duplicates = df[df.duplicated('veridion_id', keep=False)]
count_duplicates = duplicates['veridion_id'].nunique()
```

Name	Type	Size	
count_duplicates	int	1	11

Suspicion confirmed, we have 11 duplicate companies. We drop them, and we also reindex the dataset:

In [ ]:
```
#drop duplicates
df = df.drop_duplicates(subset=['veridion_id'], keep='first')

#reindexing dataframe
df = df.reset_index(drop=True)
```

And now, we've finally reached the stage where we can generate ideas based on the data we have. I don't have a lot of information about the client, but there are a few major things that I can think from the top of my head.

Revenue and employee data

Since the client is a manufacturing company, more specifically its Procurement department, which wants to analyze spend, I should first check how many companies have revenue and employees data.

In [ ]:
```
#checking values for employee number and revenue to see how many are NaN
revenue_valid = df['revenue'].notna().mean() * 100
employee_count_valid = df['employee_count'].notna().mean() * 100
```

valid_employee_count	float64	1	57.35294117647059
valid_revenue	float64	1	54.20168067226891

Only 57% of companies have employee count, and only 54% of them have revenue values. Also, some of those values are extracted, and others are modelled by the internal system.

The client should be asked if they want us to specifically flag companies based on these criteria.

Geographical distribution of supplier companies

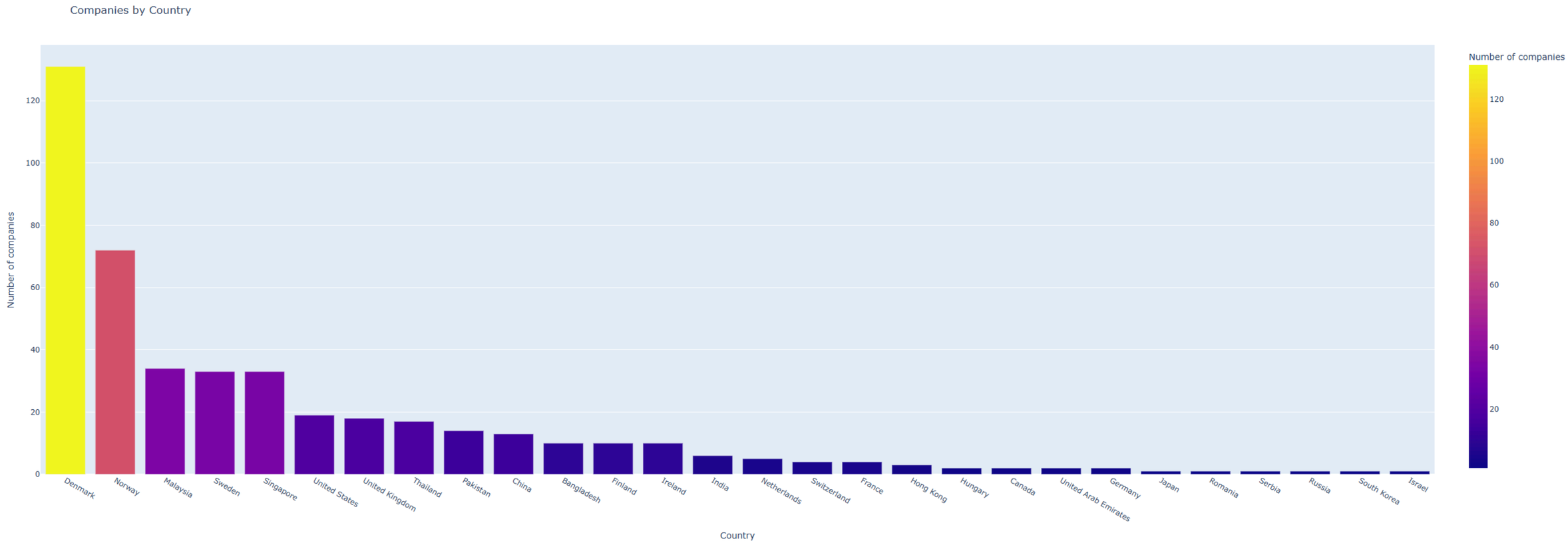
We use the plotly.py charting library to make a chart based on grouping companies by their home country

In [ ]:
```
#generating a chart that shows the number of companies for each country
counts = df['main_country'].value_counts().reset_index()
counts.columns = ['main_country', 'company_name']
fig = px.bar(counts, x='main_country', y='company_name', color='company_name', title="Companies by Country",
             labels={'main_country': 'Country', 'company_name': 'Number of companies'})

fig.write_html("companies_per_country.html")
```

The neat thing about plotly library is that it can save the charts as interactive objects which can be explored in the browser. The following chart can be opened in the browser and examined more closely. Due to time constraints, I don't yet know how to embed it into this current presentation, so I've only inserted a printscreen of it.

However, the file **companies_per_country.html** can be opened separately and examined more closely. At the moment, this is simply a showcase of what plotly can do, and how a business insight can be presented based on the data at hand.


Companies by Country
Number of companies
0
20
40
60
80
100
120
Denmark
Norway
Malaysia
Sweden
Singapore
United States
United Kingdom
Thailand
Pakistan
China
Bangladesh
Finland
Ireland
India
Netherlands
Switzerland
France
Hong Kong
Hungary
Canada
United Arab Emirates
Germany
Japan
Romania
Serbia
Russia
South Korea
Israel
Country
Number of companies

As we can see from the chart, roughly half of the supplying companies are from the Scandinavian countries - Denmark, Norway, Sweden and Finland. Since the customer is also looking at sustainability and supply chain consolidation, this data shows a possible scenario to achieve that.

In the current turbulent times of 2026, the Scandinavian countries operate as a geopolitical bloc from a security perspective. They have a common history and very strong mutual defense pacts. They have been united under the Swedish Empire for more than a century, and have kept close relations between themselves for the past 2 centuries.

Even in an unlikely scenario of NATO being unable to effectively respond to security crises, the Scandinavian countries have significant interoperability between their security apparatuses. Finland has the largest army, Sweden the most equiped and technologically advanced army, Norway is a key supplier of energy, and Denmark has a strong diplomatic presence with key relations with states such as the Netherlands and the United Kingdom. If our customer is looking for security in case of global disruption in supply chains due to global conflict, advising them to focus on procurement from such a secure area would be advisable.

This sort of insight should be pointed out in the presentation to the client. However, I should also mention that it's not clear where the main manufacturing base of the client is located. It is not stated in the task. However, if it is located in Denmark (which may be a lucky guess simply because most of the suppliers are from there), we could advise them to consolidate their purchases in the Scandinavian countries and Northern Europe, and try to move away from Asian suppliers in case Pacific conflicts erupt.

There is also the benefit of geographic proximity for a decision to consolidate supply chains in a single geographic area. The Scandinavian countries are extremely well connected both by rail and road networks, and they are all connected on water by the Baltic Sea and North Sea. Water transport has the added advantage of being by far the cheapest mode of transport both from an Energy Return on Investment (EROI) perspective and from a monetary perspective.

Additional ideas to check with the client

1. Plotly has many types of charting possibilities, which can all be checked on the official website: https://plotly.com/python/ (https://plotly.com/python/) . Experiment with some charts, since the visual presentation is by far the richest type of informational content transmission
2. Charting the companies with their geographical locations based on longitude and latitude is a possibility. Calculations can be made for transport costs, since the distances between them can also be calculated since we have access to that data
3. We can make further error corrections in the data. For example, we could flag companies that have mismatched description and categories - healthcare providers in description , but cleaning services in category columns

---

Possible refinements

1. I am not entirely happy with how I've solved the problems of commas in the csv file. Even if I've used the LibreOffice Calc for it, and replaced all the commas with semicolons, when I am now trying to open the final csv, **companies_data.csv** , I find out it cannot be opened in Excel because it works badly with semicolons. This is a problem, because I cannot expect the client to simply install LibreOffice if they want to explore the data. I need to find a way to further process the data, without removing all the commas from it, in order to export it in a client readable format. I think pandas had some methods such as export_to_excel() or something like that. To keep in mind for the future.
2. The matching algorithm was not perfect. On the one hand, I feel I have left too many companies outside of the final dataset (17.5% are unmatched). On the other hand, I feel that too many of the ones left in it are simply inaccurate. In the future, I think that further experimentation with additional matching algorithms, and trying to make a weighted average of their results, might bring a better matching score and reduce loss of information.
3. I haven't touched web crawling at all in this project. Ideally, I would have tried to write a simple scrapper that could access the company websites that are available in the datase, and try to scrape some metadata and compare it with that is already available in the dataset. Due to the time constraints and the deadline I've imposed on myself for building this POC, I decided against it. However, in a real life scenario, this would've been the sort of decision that I would've discussed with my team lead or in a meeting, since the option of crawling opens up an entirely new avenue of possibilities regarding how this problem could be approached.
4. Some classification algorithm could be implemented for some additional cleaning of data. For example, some datapoints extractor could be implemented on the company description columns, and simply check if it matches with the company categories. Might reduce the noise and reduce the missclassification of companies in wrong categories.

Thank you very much for giving me the opportunity to tackle this problem!