BDE TASK SOLUTION

Introduction

At the beginning, we have 3 datasets to be used as a source.

The given datasets are:

1. website_dataset.csv
2. facebook_dataset.csv
3. google_dataset.csv

These datasets contain information about companies from different sources like Facebook, Google and websites. The main task of this project is to combine data from 3 given input datasets in a most accurate way, resulting in the 4th dataset with cleaned and transformed data.

Terminology & Details

Language and environment: Pyspark (python+spark) - jupyter notebook – databricks – azure datalake

web_data, website_data - referring to the website dataset (website_dataset.csv)

fb_data, facebook_data - referring to the facebook dataset (facebook_dataset.csv)

gg_data, google_data - referring to the google dataset (google_dataset.csv)

{dataset}_missing - referring to null value percentage of dataset for each column

{dataset}_distinct - referring to unique value percentage of dataset for each column

web_fb - website_dataset and facebook_dataset

web_fb_left - website_dataset and facebook_dataset join with left operation

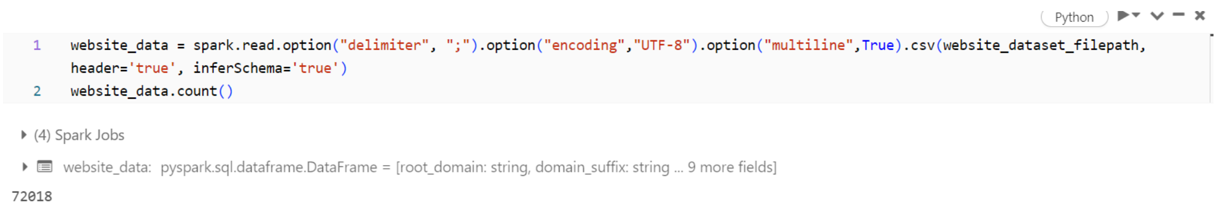
web_fb_inner - website_dataset and facebook_dataset join with inner operation

remaining_gg - non-matched data in google_dataset after join operation

web_fb_gg_full - website_dataset, facebook_dataset, google_dataset are merged into that dataset

Reading Data

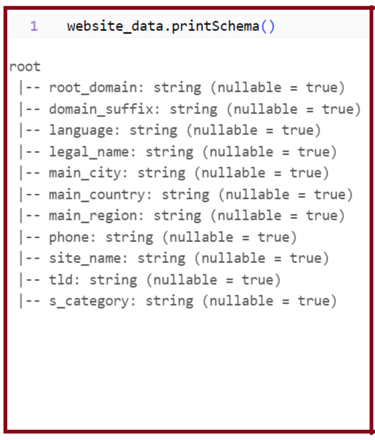
I read data from databricks environment after uploading csv files into filestore. For handling new line characters in csv file, i used multiline option. Also i used encoding option with UTF-8 to handle different language characters for example taiwan, japan alphabet.



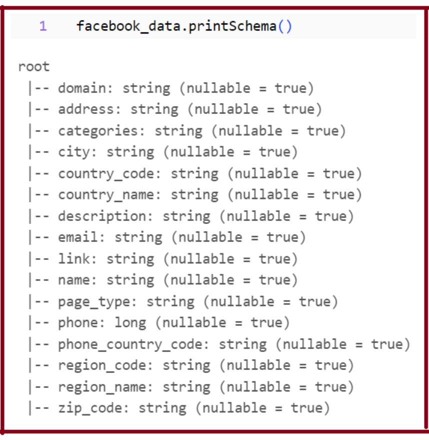
```
website_data = spark.read.option("delimiter", ";").option("encoding","UTF-8").option("multiline",True).csv(website_dataset_filepath,
header='true', inferSchema='true')
website_data.count()
```

```
(4) Spark Jobs
website_data:  pyspark.sql.dataframe.DataFrame = [root_domain: string, domain_suffix: string ... 9 more fields]
72018
```

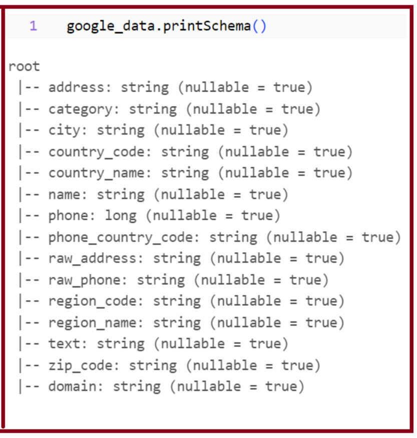
Understanding Data



```
website_data.printSchema()

root
 |-- root_domain: string (nullable = true)
 |-- domain_suffix: string (nullable = true)
 |-- language: string (nullable = true)
 |-- legal_name: string (nullable = true)
 |-- main_city: string (nullable = true)
 |-- main_country: string (nullable = true)
 |-- main_region: string (nullable = true)
 |-- phone: string (nullable = true)
 |-- site_name: string (nullable = true)
 |-- tld: string (nullable = true)
 |-- s_category: string (nullable = true)
```

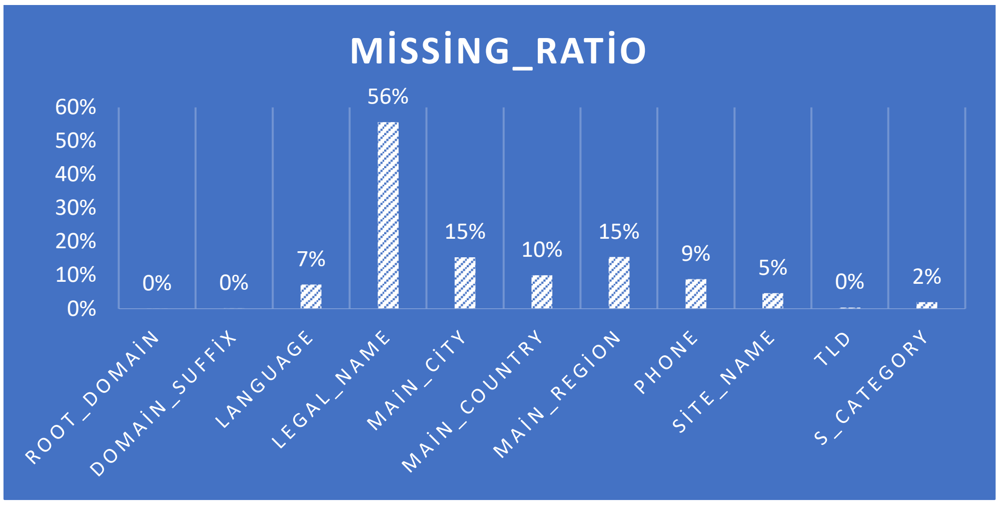


```
facebook_data.printSchema()

root
 |-- domain: string (nullable = true)
 |-- address: string (nullable = true)
 |-- categories: string (nullable = true)
 |-- city: string (nullable = true)
 |-- country_code: string (nullable = true)
 |-- country_name: string (nullable = true)
 |-- description: string (nullable = true)
 |-- email: string (nullable = true)
 |-- link: string (nullable = true)
 |-- name: string (nullable = true)
 |-- page_type: string (nullable = true)
 |-- phone: long (nullable = true)
 |-- phone_country_code: string (nullable = true)
 |-- region_code: string (nullable = true)
 |-- region_name: string (nullable = true)
 |-- zip_code: string (nullable = true)
```

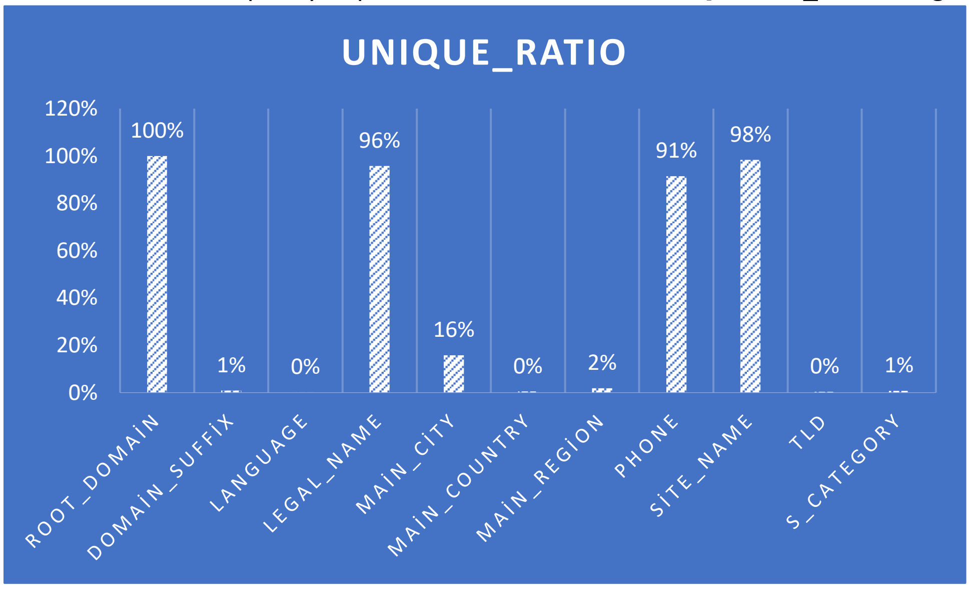


```
google_data.printSchema()

root
 |-- address: string (nullable = true)
 |-- category: string (nullable = true)
 |-- city: string (nullable = true)
 |-- country_code: string (nullable = true)
 |-- country_name: string (nullable = true)
 |-- name: string (nullable = true)
 |-- phone: long (nullable = true)
 |-- phone_country_code: string (nullable = true)
 |-- raw_address: string (nullable = true)
 |-- raw_phone: string (nullable = true)
 |-- region_code: string (nullable = true)
 |-- region_name: string (nullable = true)
 |-- text: string (nullable = true)
 |-- zip_code: string (nullable = true)
 |-- domain: string (nullable = true)
```

Website_data:

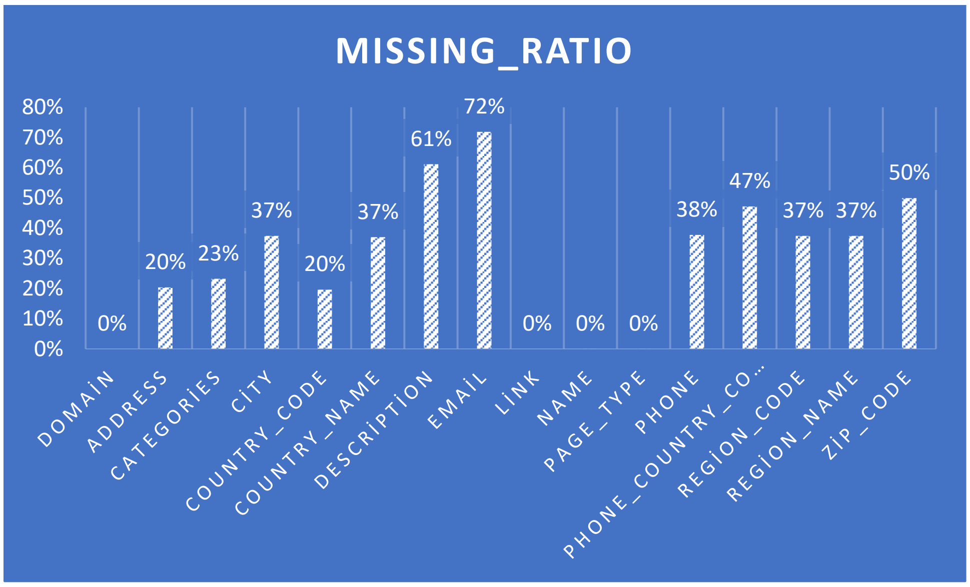


55% of "Legal Name" is missing so we need to use "Site Name" which has only 4% null value. Also when "Site Name" is null then we can use "Root Domain" as company name for keeping all information. We keep only important columns and discard [domain_suffix, language, tld].

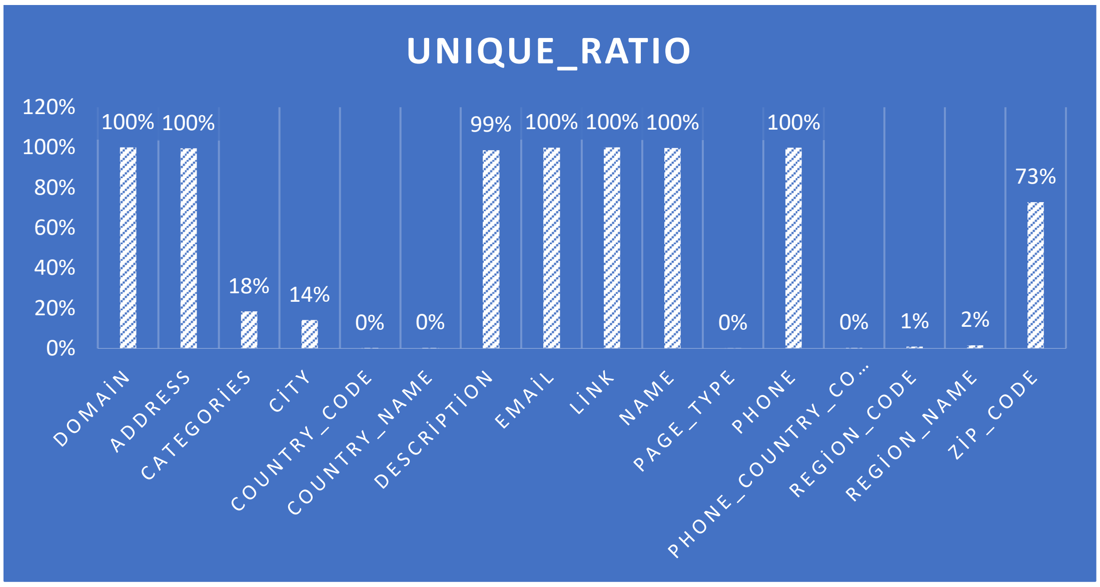


All values for "Root Domain" are unique. 98% of "Site Name" is also unique. Unique percentage of "Legal Name" is 95%, that means 5% of company names have more than 1 website. We can concat domain names if we group data by "Legal Name". However, for getting more information from this data we should use "Root Domain" and "Site Name".

Facebook_data:

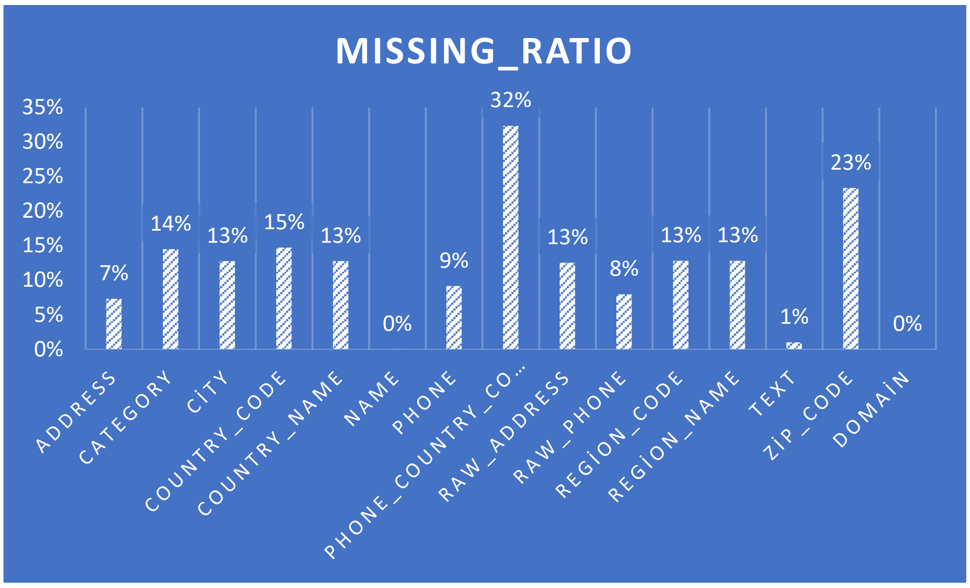


In facebook_data, "Domain" column has 0% null values, and "Name" column has roughly 0% null values. "Categories" column includes rich information but 23% of that column's values are null.

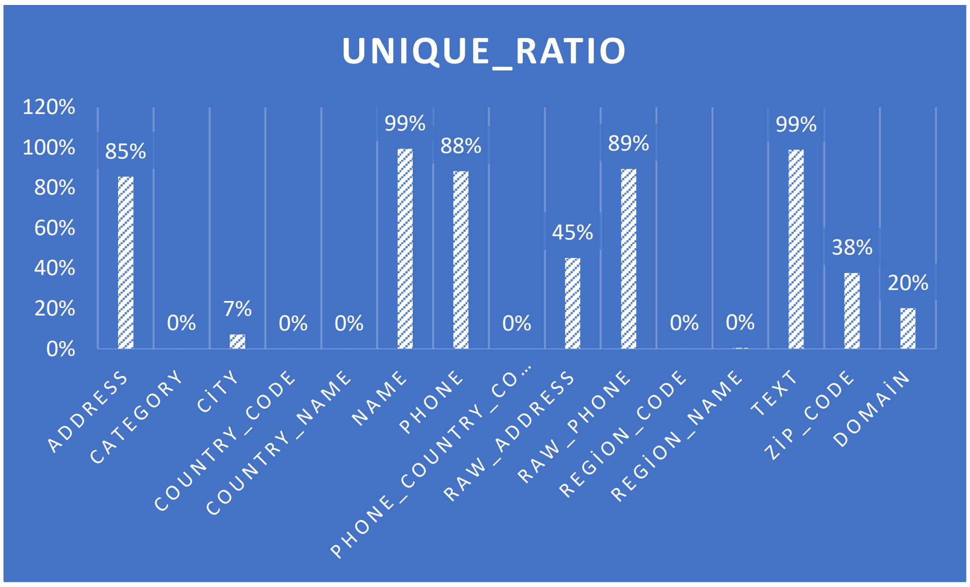


In facebook data, all domain names are unique, also their address, email, name, and phone information have unique values more than 99%. We will keep important columns (domain, name, categories, country_name, region_name, city, address, zip_code, email, phone, page_type) for merging with website data.

Google_data:



In google data, domain and name columns have fewer missing data. However, "domain" column includes aggregator domains like facebook, instagram.



"domain" column has only 20% percent unique values because of aggregator domains. Therefore, we need also to use "phone" and/or "name" column for joining with website and facebook datasets.

Some companies have more than one store, so there are multiple addresses and phone for same domain. We can group it according to domain or we can think that every store is a different business.

Data Preparation and Merging

We rename column names and add dataset prefix to all.

Web-FB join:

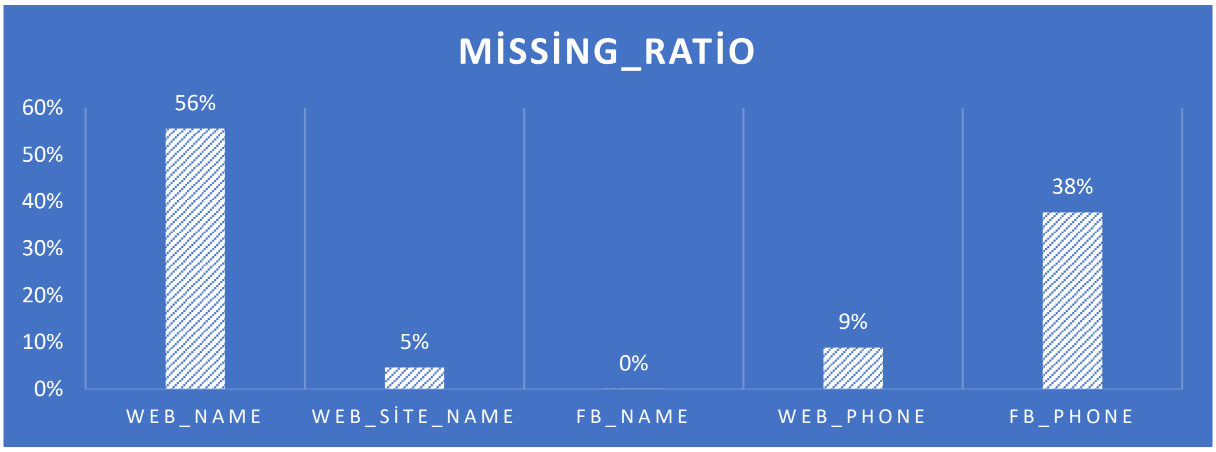
First, we join web and fb datasets using domain column.

```
web_fb_inner = web_data.join(fb_data, on = [web_data.web_domain == fb_data.fb_domain], how = "inner")
web_fb_left = web_data.join(fb_data, on = [web_data.web_domain == fb_data.fb_domain], how = "left")
print("join ratio:",web_fb_inner.count()/web_fb_left.count())
```

```
(6) Spark Jobs
web_fb_inner:  pyspark.sql.dataframe.DataFrame = [web_domain: string, web_name: string ... 17 more fields]
web_fb_left:  pyspark.sql.dataframe.DataFrame = [web_domain: string, web_name: string ... 17 more fields]
join ratio: 0.9998194895720515
```

The join ratio is more than 99%, so domain selection is good for these two datasets.

Google data does not have proper domain name so we will use company names for merging google data with others. First, we will look for name columns in "web_fb_data".



fb_name columns have a very small number of null values but web_name has 55% null values. Therefore, we will use fb_name for matching with google. I will use MinHashLSHModel for finding similarity between google_data names and web_fb_data names. Making cross join is not effective and causes a lot of cost and increases data size. Therefore, I need to join two datasets with a common column to decrase data size. So, i will join using phone then if there is no match i will use cross join to calculate distance metrics.

(Web-fb) and gg data join using phone:

I clean non-numeric data from phone columns:

```
from pyspark.sql import functions as F
# FB phone has well format, if it is available use it. I try both version and fb_phone has bigger coverage rate
web_fb_data = web_fb_data.withColumn('fb_phone', F.regexp_replace(F.col("fb_phone"), '[^0-9]+', ''))
web_fb_data = web_fb_data.withColumn('web_phone', F.regexp_replace(F.col("web_phone"), '[^0-9]+', ''))
web_fb_data = web_fb_data.withColumn('new_phone', F.when(F.isnan("fb_phone") | F.col("fb_phone").isNull(), F.col("web_phone")).otherwise(F.
col("fb_phone")))

gg_data = gg_data.withColumn('gg_phone', F.regexp_replace(F.col("gg_phone"), '[^0-9]+', ''))
```

```
web_fb_gg_inner = web_fb_data.join(gg_data, on = [web_fb_data.new_phone == gg_data.gg_phone], how = "inner")
web_fb_gg_left = web_fb_data.join(gg_data, on = [web_fb_data.new_phone == gg_data.gg_phone], how = "left")
print("join ratio:",web_fb_gg_inner.count()/web_fb_gg_left.count())
print("domain coverage ratio:",web_fb_gg_inner.select("web_domain").distinct().count()/web_fb_data.select("web_domain").distinct().count())
```

```
(19) Spark Jobs
web_fb_gg_inner:  pyspark.sql.dataframe.DataFrame = [web_domain: string, web_name: string ... 27 more fields]
web_fb_gg_left:  pyspark.sql.dataframe.DataFrame = [web_domain: string, web_name: string ... 27 more fields]
join ratio: 0.7339806069447763
domain coverage ratio: 0.7062956483101447
```

Now we have joined all three datasets. Web - Facebook join ratio is 99%. (Web-Facebook) and Google join ratio is 70%. For the other 30%, I will use MinHashLSHModel.

MinHashLSHModel in PySpark:

MinHashLSHModel is part of PySpark's MLlib (Machine Learning Library) and is used for finding approximate similarity between pairs of rows in a dataset. It is particularly useful when dealing with large datasets and is often applied to text data for tasks like record linkage or entity resolution. The MinHash technique is a probabilistic data structure used to estimate the Jaccard similarity between two sets efficiently.

In the context of assignment, MinHashLSHModel is applied to find similarities between names in the Google dataset (gg_name) and names in the combined website and Facebook dataset (web_fb_name).

Tokenization and n-grams: First, company names are tokenized into words, and n-grams (combinations of words) are created. This is done to represent company names as sets of n-grams, which are more responsible to similarity comparisons.

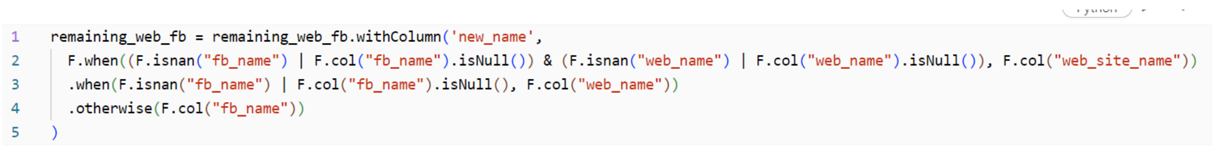
Numerical Vectors: These n-grams are then converted into numerical vectors using techniques like HashingTF (Term Frequency). This step transforms the text data into numerical format.

MinHash LSH Model: The MinHashLSHModel is created, which is essentially a hashing strategy that allows the efficient calculation of Jaccard similarity between sets of data. It is used to find approximate similarities.

Approximate Similarity Join: The model is used to join the two datasets efficiently to find pairs of rows whose Jaccard distance (a measure of dissimilarity) is smaller than a specified threshold. In our case, a threshold of 0.5 is used for string matching.

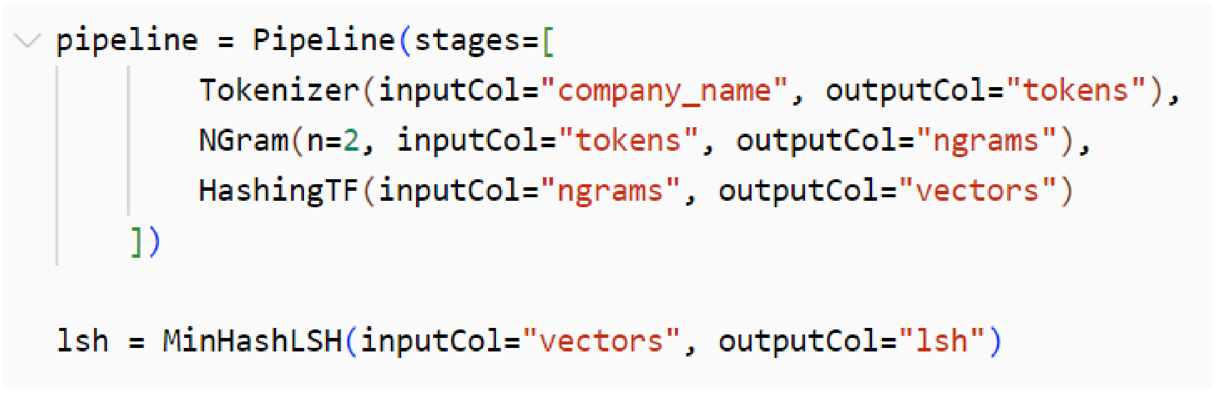
(Web-fb) and gg data join using approxSimilarityJoin:

Fb_name is more valuable than others. We create new_name because if fb_name is null we will use web or web_site name.



```python
remaining_web_fb = remaining_web_fb.withColumn('new_name',
  F.when((F.isnan("fb_name") | F.col("fb_name").isNull()) & (F.isnan("web_name") | F.col("web_name").isNull()), F.col("web_site_name"))
  .when(F.isnan("fb_name") | F.col("fb_name").isNull(), F.col("web_name"))
  .otherwise(F.col("fb_name"))
)
```

We are creating a pipeline for using MinHashLSHModel to calculate distance metric between gg_name and fb_name.



```python
pipeline = Pipeline(stages=[
        Tokenizer(inputCol="company_name", outputCol="tokens"),
        NGram(n=2, inputCol="tokens", outputCol="ngrams"),
        HashingTF(inputCol="ngrams", outputCol="vectors")
    ])

lsh = MinHashLSH(inputCol="vectors", outputCol="lsh")
```

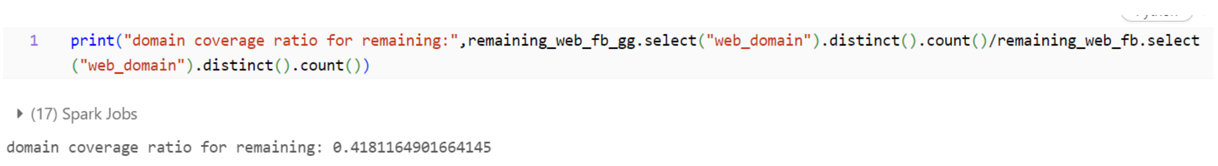
First, we convert company names into word tokens, then using ngram we create combinations of words. Next step is using HashingTF for converting words into numerical vectors. Finally using MinHashLSHModel, we create efficient hashing strategy for those vectors.

At the end, approxSimilarityJoin helps us to join two datasets to approximately find all pairs of rows whose distance are smaller than the threshold.

approxSimilarityJoin is using Jaccard distance approach for calculating distance.

```
matched_df = model.approxSimilarityJoin(df, df2, 0.5, "confidence")
print(matched_df.count())
display(matched_df)
```

We filter data with 0.5 threshold for holding only reliable matches.

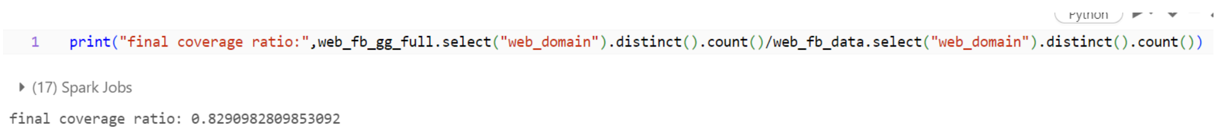


```
print("domain coverage ratio for remaining:",remaining_web_fb_gg.select("web_domain").distinct().count()/remaining_web_fb.select
("web_domain").distinct().count())
```

▸ (17) Spark Jobs

```
domain coverage ratio for remaining: 0.4181164901664145
```

We use 0.5 threshold for string matching, so we cover 41% of remaining data. Now we increase (Web-Facebook) and Google join ratio from 70% to 82%.



```
print("final coverage ratio:",web_fb_gg_full.select("web_domain").distinct().count()/web_fb_data.select("web_domain").distinct().count())
```

▸ (17) Spark Jobs

```
final coverage ratio: 0.8290982809853092
```

Final coverage is 82.9%.

Conclusion

The project successfully merged data from three distinct sources—website, Facebook, and Google—resulting in a comprehensive dataset for analysis. While achieving an 82.9% coverage, it's essential to recognize that this process can be further improved, with various areas for enhancement and optimization. When I encounter very similar data across sources, I can prioritize data from the source with the highest reliability based on my domain-specific knowledge or previous experiences.