Solidify Test Project

# Presentation

[soleadify\_test\_project\_presentation](https://docs.google.com/presentation/d/1g2QOvGpoyxp2wIlX0P_LrLo0gpi4WR093ud9PwfLBuk/edit?usp=sharing)

# Introduction

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

The given datasets are:

* facebook\_dataset.csv
* website\_dataset.csv
* google\_dataset.csv

These data 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. *Python* programming language has been chosen to complete this task.

# Terminology

* *Python* - programming language used to complete the project
* *Pandas* - a Python library used for working with data sets
* *Re* (regular expression) - a sequence of characters that specifies a match pattern in text.

# Reading (RAW part)

Firstly, we need to read the data to create well-formed CSV files, as for Facebook and Google datasets data is stored in a way that quoted fields has separators inside its data and this can lead to fields’ number explosion without additional setup. So, using *pandas,* we have a possibility to read such type of data without unexpected behavior. Also, inside some fields, there is an escape character represented by “\”.  
  
Example of reading Facebook dataset to ignore escape character and separator signs met in quoted fields:  
**pd.read\_csv(SOURCE\_FB\_PATH, quotechar='"', escapechar='\\', doublequote=False)**

# Data Cleaning and Transformations (TRANSFORM part)

So, once we have got well-formed CSV files, let’s continue with required data cleaning and transformations to prepare the data for further merging.

Web:

* Columns renaming. Below is the renaming mapping:

{

"root\_domain": "web\_domain",

"domain\_suffix": "web\_domain\_suffix",

"language": "web\_language",

"legal\_name": "web\_company\_name",

"main\_city": "web\_city",

"main\_country": "web\_country",

"main\_region": "web\_region",

"phone": "web\_phone",

"site\_name": "web\_site\_name",

"tld": "web\_tld",

"s\_category": "web\_category"

}

* Lowercase columns. Example below”

Input value → J & W Foods Inc.

Output value → j & w foods inc.

* Escape special characters, spaces in some columns. Also removing suffixes like (ltd, co …) in company names. Example below:

Input value → j & w foods inc.

Output value → j\_w\_foods

* Filtering some columns based on regular expressions to get rid of rubbish rows.

For example, domain cannot include any other characters except for letters, numbers, dot and dash. So if such one met, we filter such row out.

* Dropping NaNs from join columns
* Cast “phone” column type to float
* Dropping duplicates

Facebook:

* Columns renaming. Below is the renaming mapping:

{

"domain": "fb\_domain",

"address": "fb\_address",

"categories": "fb\_categories",

"city": "fb\_city",

"country\_code": "fb\_country\_code",

"country\_name": "fb\_country",

"description": "fb\_description",

"email": "fb\_email",

"link": "fb\_link",

"name": "fb\_company\_name",

"page\_type": "fb\_page\_type",

"phone": "fb\_phone",

"phone\_country\_code": "fb\_phone\_country\_code",

"region\_code": "fb\_region\_code",

"region\_name": "fb\_region",

"zip\_code": "fb\_zip\_code"

}

* Lowercase columns. Example below”

Input value → J & W Foods Inc.

Output value → j & w foods inc.

* Split and Explode fb\_categories column into multiple rows, as it is represented as string split by “|”. Example:

Input value → fb\_categories

clothing stores|motorcycle dealers & rentals

Output value → fb\_categories

clothing stores

motorcycle dealers & rentals

* Escape special characters, spaces in some columns. Also removing suffixes like (ltd, co …) in company names. Example below:

Input value → j & w foods inc.

Output value → j\_w\_foods

* Filtering some columns based on regular expressions to get rid of rubbish rows.

For example, domain cannot include any other characters except for letters, numbers, dot and dash. So if such one met, we filter such row out.

* Dropping NaNs from join columns
* Dropping duplicates

Google:

* Columns renaming. Below is the renaming mapping:

{

"address": "gg\_address",

"category": "gg\_category",

"city": "gg\_city",

"country\_code": "gg\_country\_code",

"country\_name": "gg\_country",

"name": "gg\_company\_name",

"phone": "gg\_phone",

"phone\_country\_code": "gg\_phone\_country\_code",

"raw\_address": "gg\_raw\_address",

"raw\_phone": "gg\_raw\_phone",

"region\_code": "gg\_region\_code",

"region\_name": "gg\_region",

"text": "gg\_text",

"zip\_code": "gg\_zip\_code",

"domain": "gg\_domain"

}

* Lowercase columns. Example below”

Input value → J & W Foods Inc.

Output value → j & w foods inc.

* Escape special characters, spaces in some columns. Also removing suffixes like (ltd, co …) in company names. Example below:

Input value → j & w foods inc.

Output value → j\_w\_foods

* Filtering some columns based on regular expressions to get rid of rubbish rows.

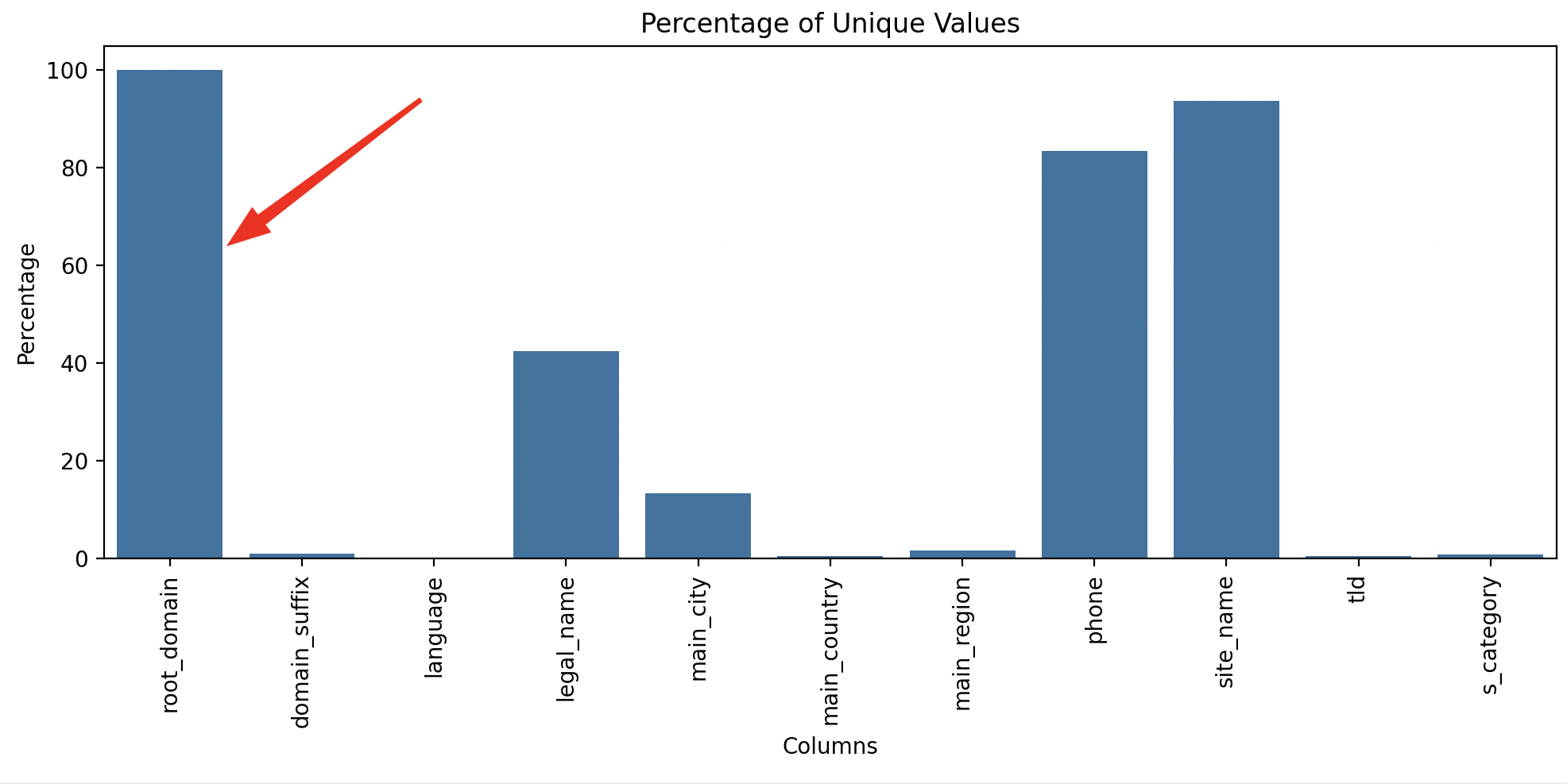
For example, domain cannot include any other characters except for letters, numbers, dot and dash. So if such one met, we filter such row out.

* Dropping NaNs from join columns
* Dropping duplicates

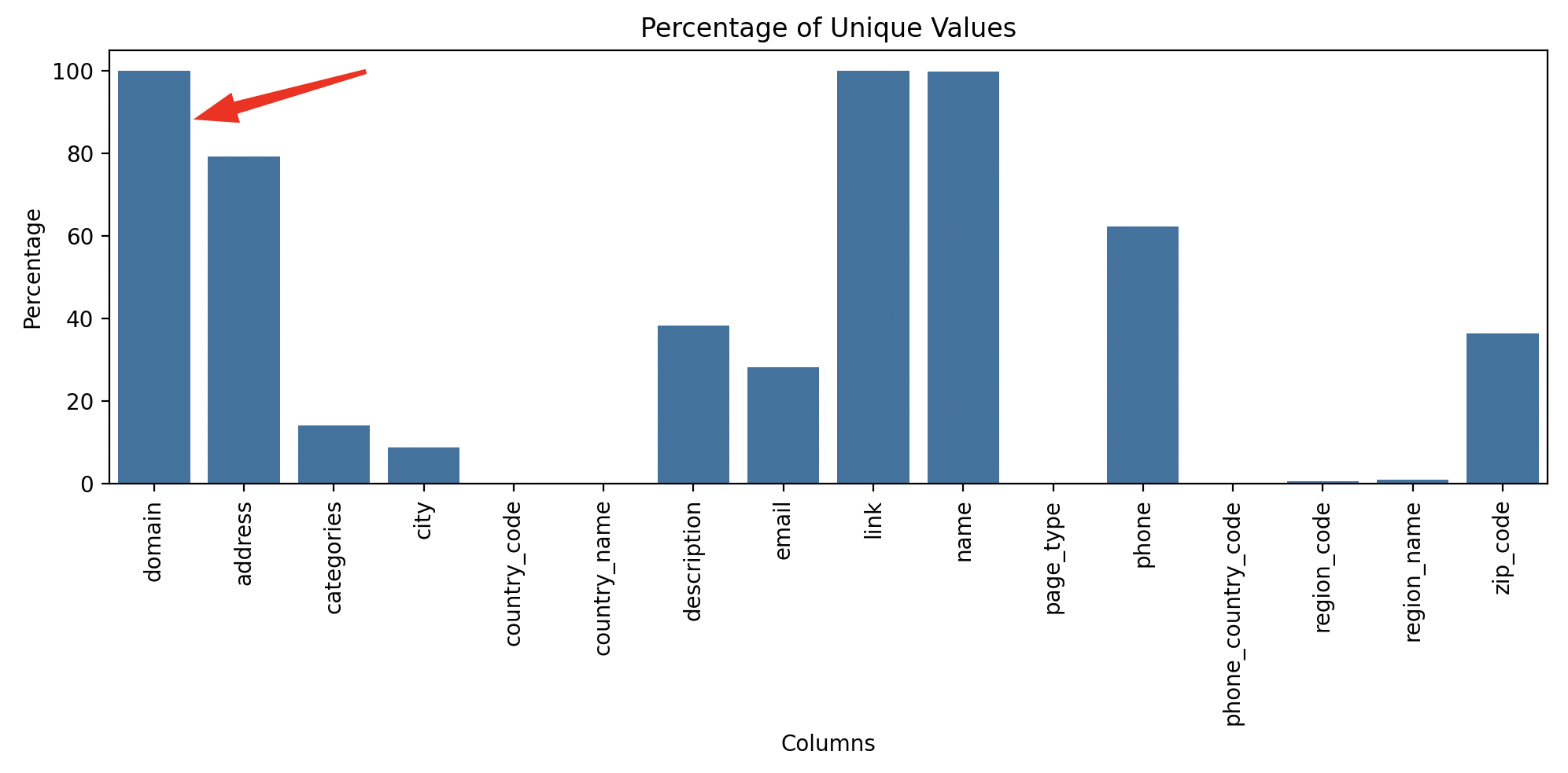
# 

# Merging Datasets (PROCESS part)

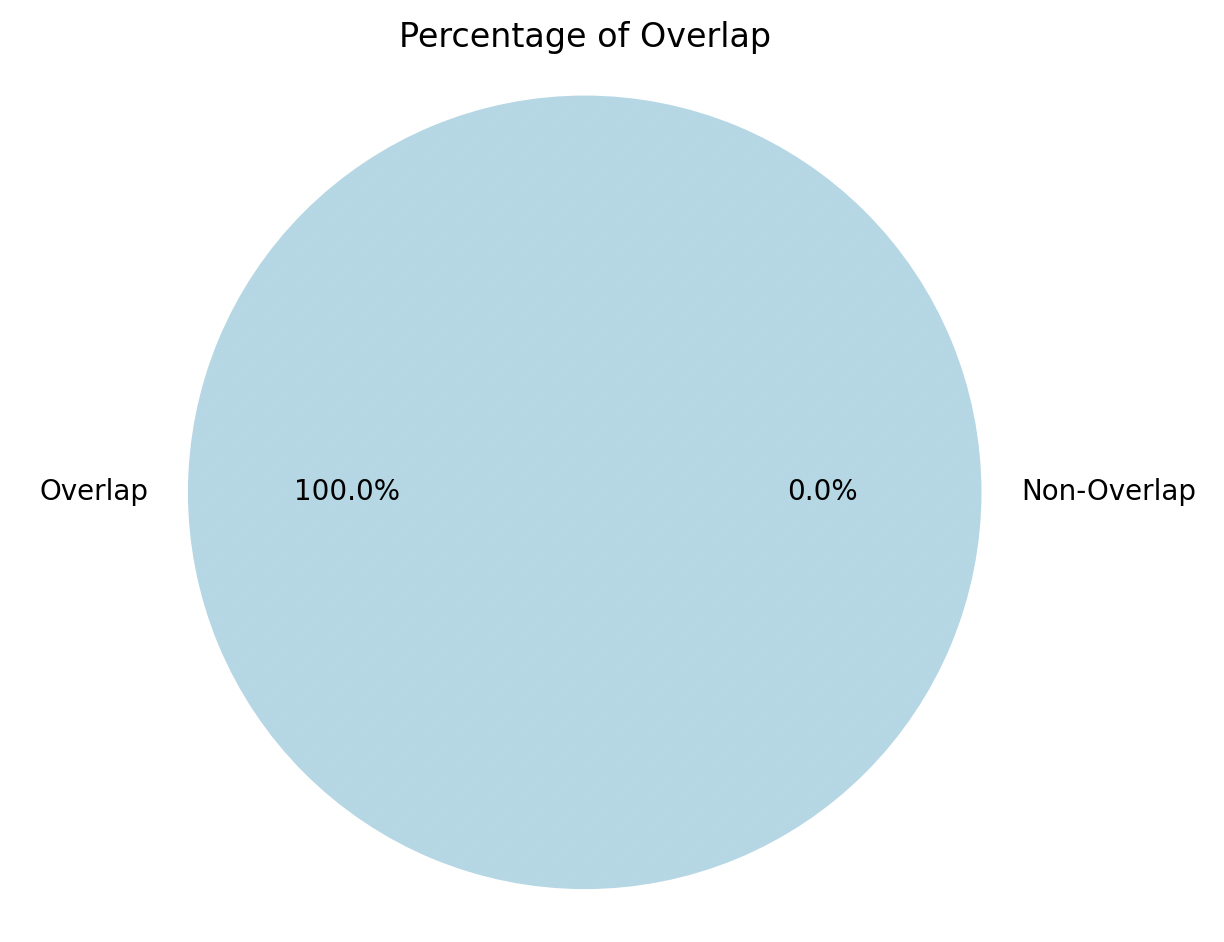
It is time now to merge our cleaned and transformed datasets into the final one. Looking at the data, we can see that Web and Facebook datasets can be joined by domain column as in both datasets it has almost 100% of unique values and also all domains from Web datasets exist in Facebook dataset. Here are evidences:



***Web dataset unique values percentage***



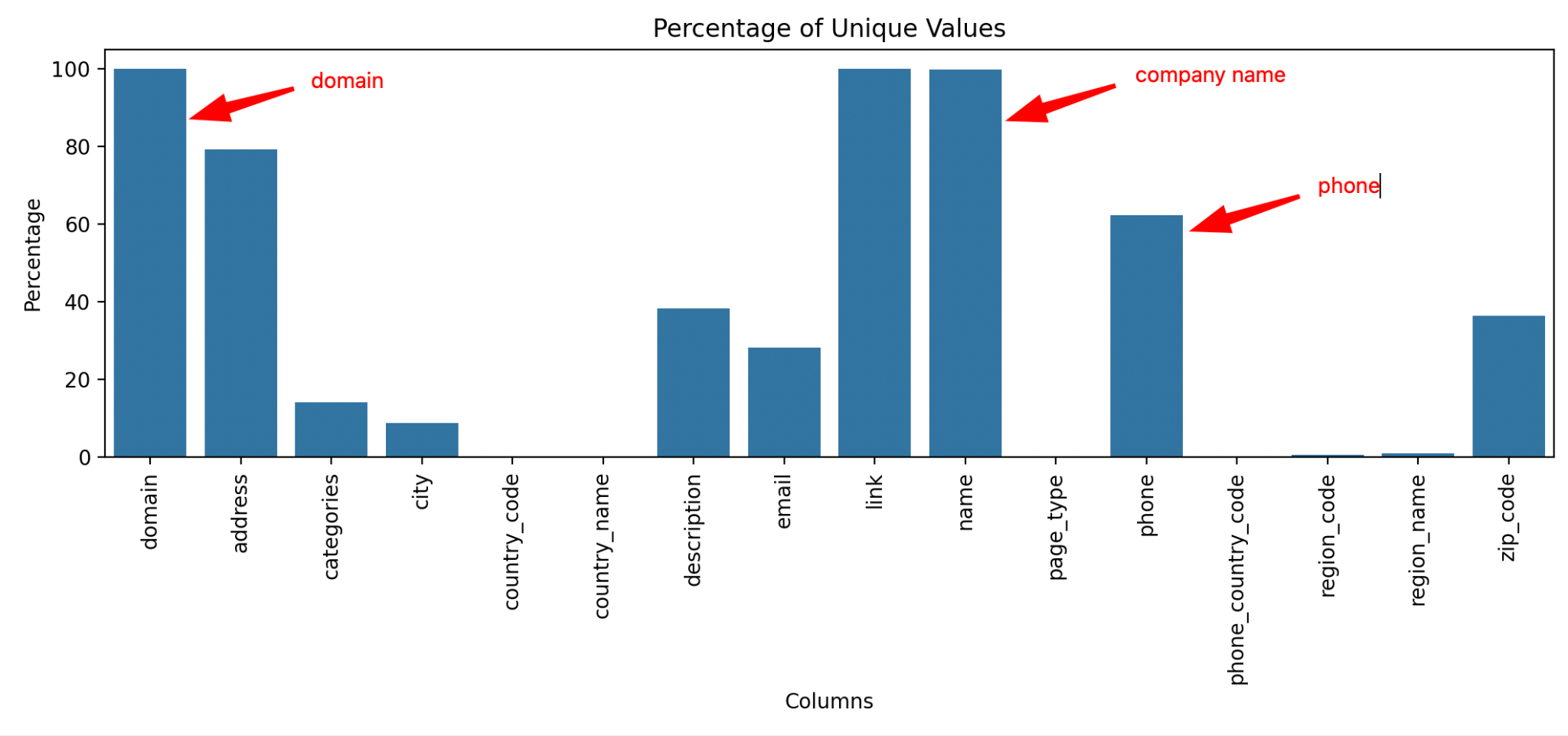
***Facebook dataset unique values percentage***



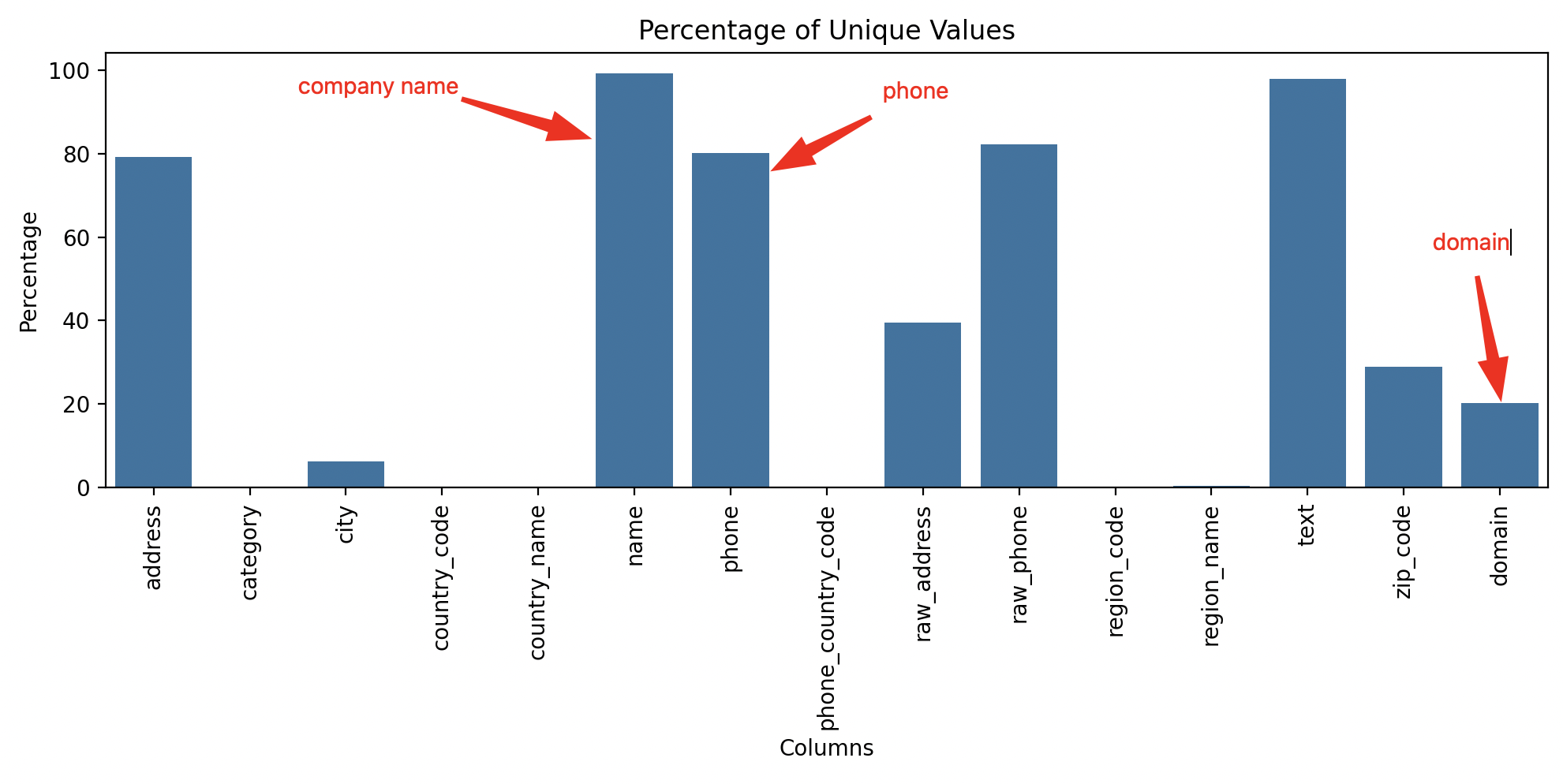
***Overlap of domain values percentage (Web domains inside Facebook domains)***

So, let’s merge our Facebook and Web datasets and using left join to leave companies that don’t exist in Web dataset. Let’s call it further FB\_WEB dataset. The final part is to join Google dataset with already joined one. But what columns to use?

1. Looking at the data, we can see that the best choice to join Facebook and Google data is to use company\_name column as it has pretty large percent of uniqueness and not nulls. We cannot use just domain as in Google dataset it is duplicated a lot.



***Facebook dataset unique values percentage***



***Google dataset unique values percentage***

1. Even if company name column in Google dataset has almost 100% of uniqueness it is not ideally unique, so to identify the rest of companies’ data uniquely we can use domain and phone columns as domain 100% unique inside Facebook and Web, phone column has scalar values and has the best percent of uniqueness (see pictures above) and not nulls looking across all datasets and columns that can be used to join and identify companies’ data uniquely. So, joining FB\_WEB dataset with Google on

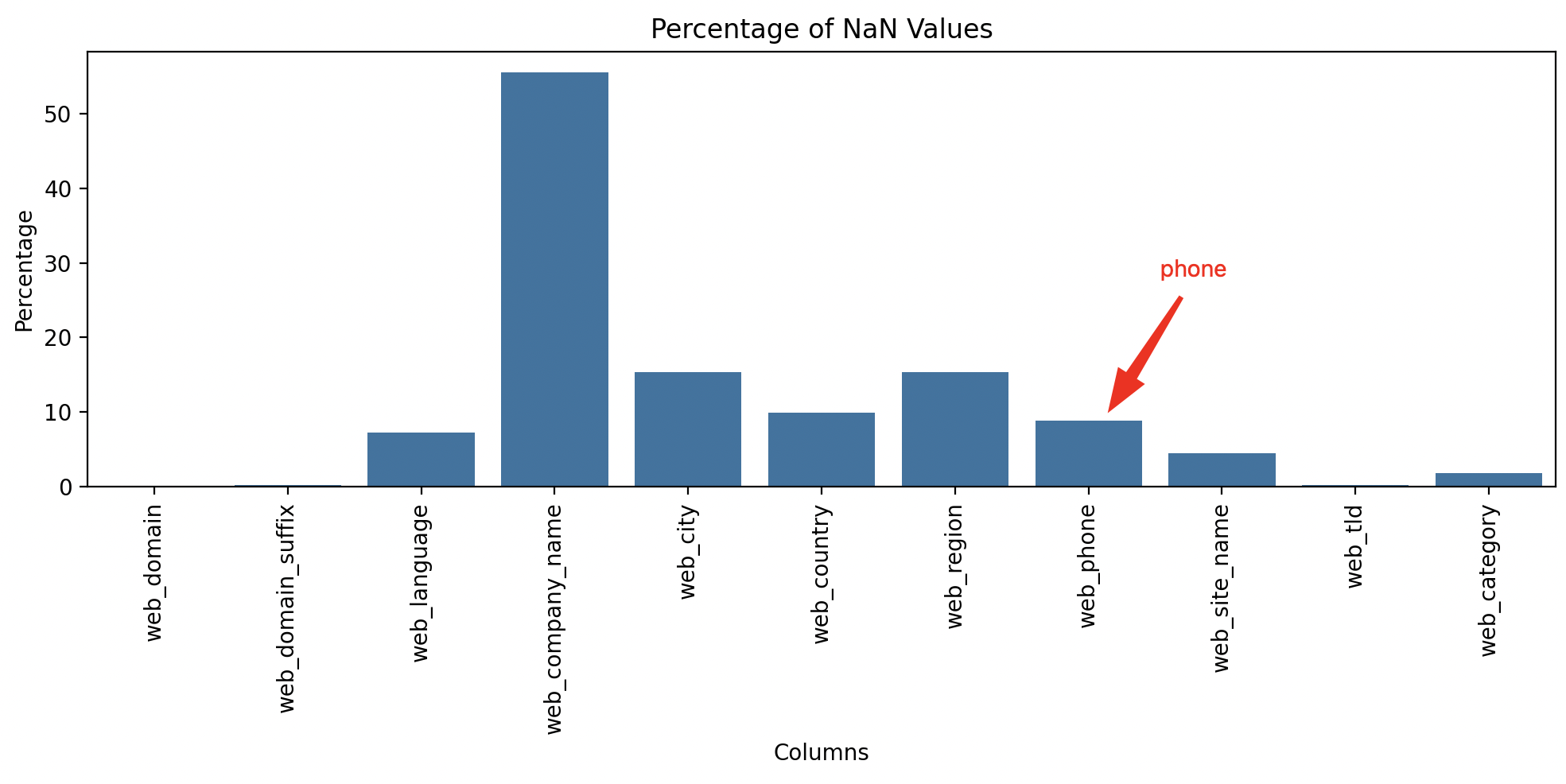
fb\_company\_name=gg\_company\_name

and

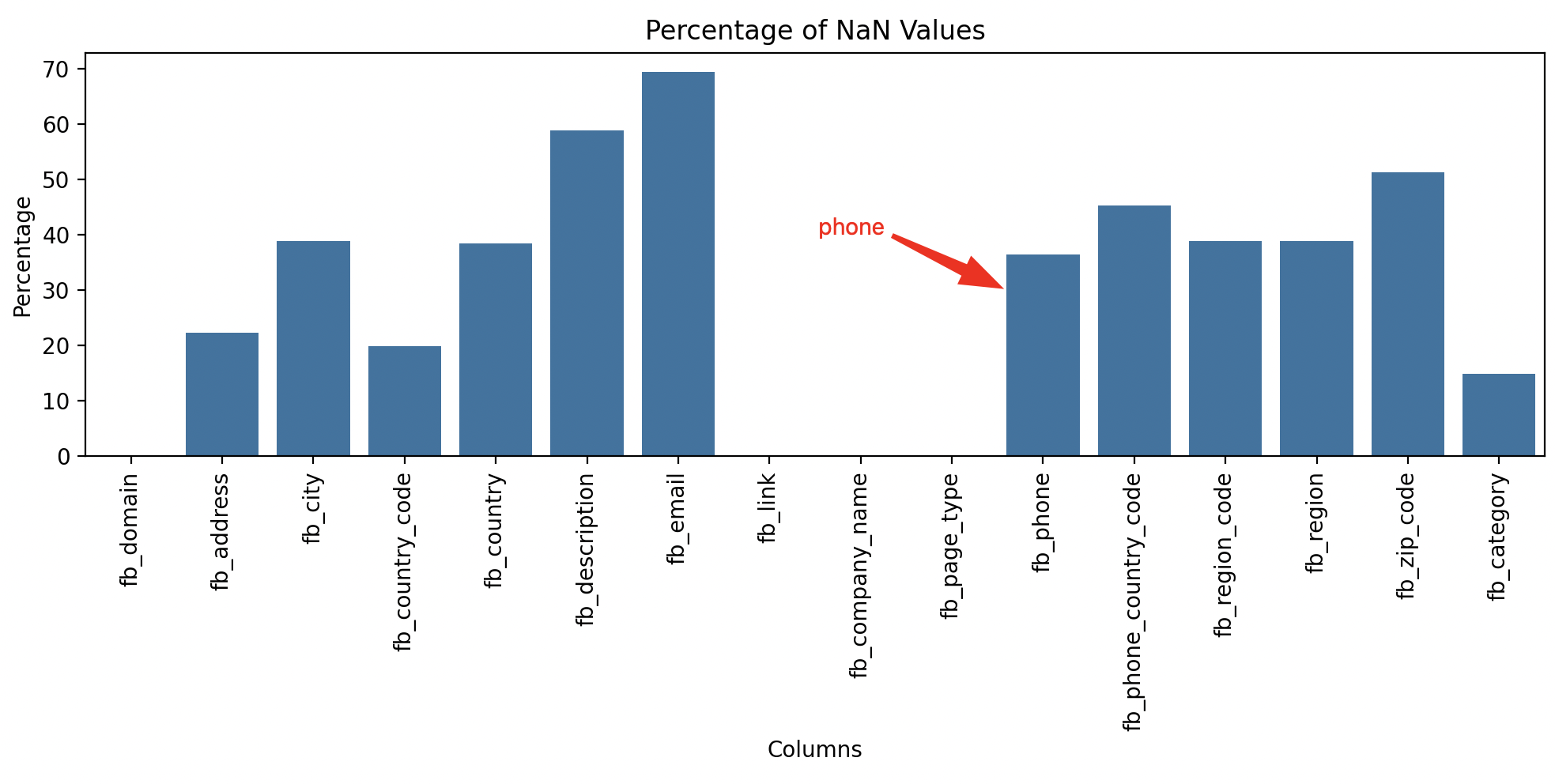
fb\_domain=gg\_domain

and

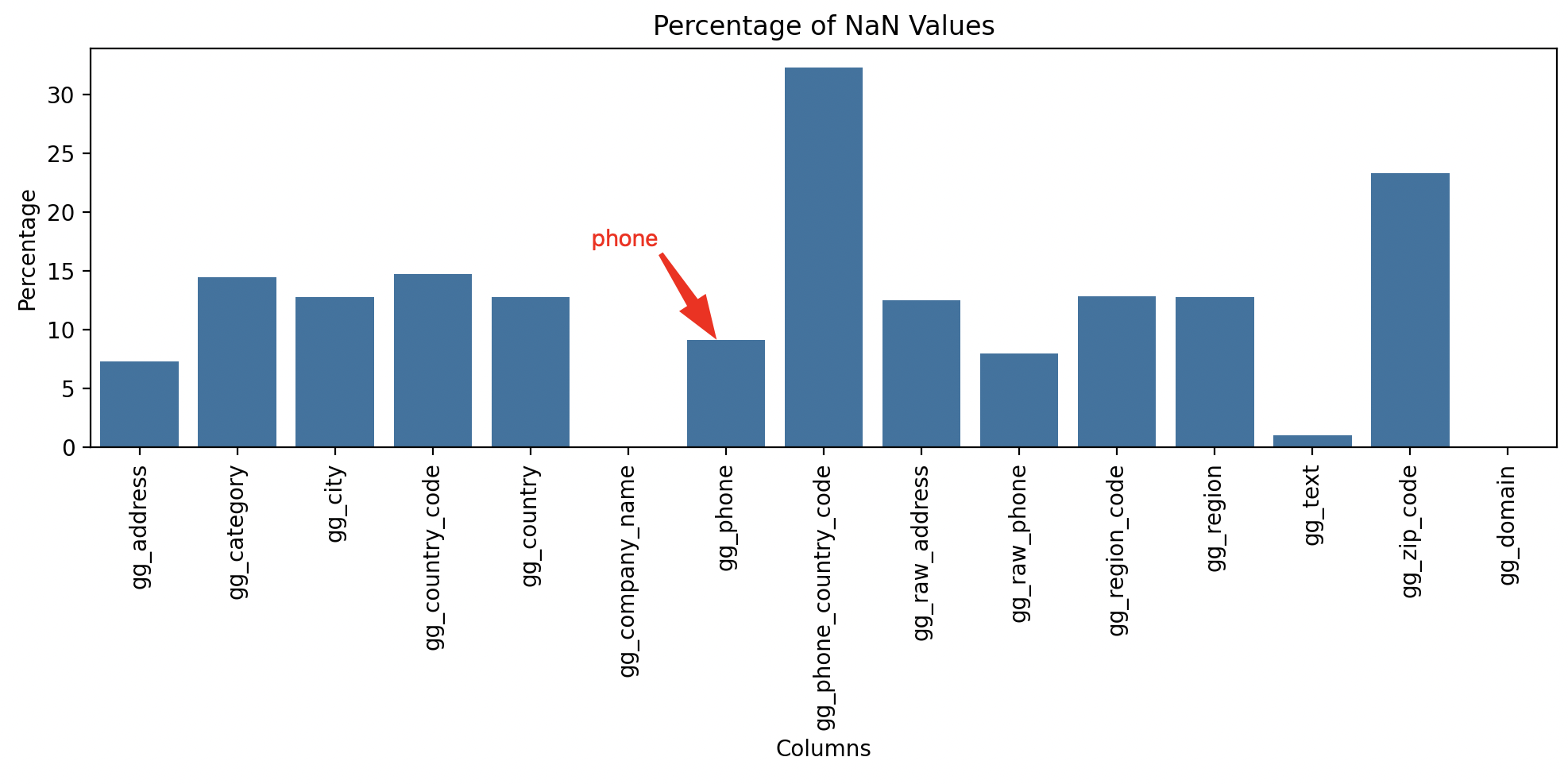
fb\_phone=gg\_phone



***Web NaN percentage***

******

***Facebook NaN percentage***

******

***Google NaN percentage***

1. But what to do with companies from web dataset? It was decided to get rows from FB\_WEB dataset where fb\_company\_name doesn’t equal to web\_company\_name and join it also with Google dataset on

web\_company\_name=gg\_company\_name

and

web\_domain=gg\_domain

and

web\_phone=gg\_phone

1. The final step is to join datasets from step 2 and 3 to get the final dataset.

# Results

* Got final dataset containing information from 3 source datasets
* Final dataset has 44851 entries
* Final dataset has 27288 unique companies
* The main reason of why company names have duplicated values is that fb\_categories column was exploded into multiple rows to have the same structure as category columns from Web and Google datasets. The other duplicates got from Google dataset as initially for the same company can be different gg\_text, gg\_city, gg\_adress etc. values
* Each row can be identified uniquely by gg\_company\_name, fb\_category and gg\_text
* Final dataset structure:

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 fb\_domain 44851 non-null object

1 web\_domain 44773 non-null object

2 gg\_domain 44851 non-null object

3 fb\_company\_name 44851 non-null object

4 web\_company\_name 22518 non-null object

5 gg\_company\_name 44851 non-null object

6 fb\_category 38832 non-null object

7 web\_category 44134 non-null object

8 gg\_category 41271 non-null object

9 fb\_country 31370 non-null object

10 web\_country 40122 non-null object

11 gg\_country 40904 non-null object

12 fb\_region 31342 non-null object

13 web\_region 38129 non-null object

14 gg\_region 40899 non-null object

15 fb\_city 31345 non-null object

16 web\_city 38133 non-null object

17 gg\_city 40903 non-null object

18 fb\_phone 44202 non-null float64

19 web\_phone 41631 non-null float64

20 gg\_phone 44851 non-null float64

21 fb\_country\_code 44528 non-null object

22 gg\_country\_code 43967 non-null object

23 fb\_region\_code 31342 non-null object

24 gg\_region\_code 40899 non-null object

25 fb\_address 37352 non-null object

26 gg\_address 42568 non-null object

27 fb\_zip\_code 24455 non-null object

28 gg\_zip\_code 34669 non-null object

29 fb\_phone\_country\_code 39516 non-null object

30 gg\_phone\_country\_code 42766 non-null object

31 fb\_description 19440 non-null object

32 fb\_email 15147 non-null object

33 fb\_link 44851 non-null object

34 fb\_page\_type 44851 non-null object

35 web\_domain\_suffix 44725 non-null object

36 web\_language 42589 non-null object

37 web\_site\_name 42567 non-null object

38 web\_tld 44704 non-null object

39 gg\_raw\_address 41721 non-null object

40 gg\_raw\_phone 44851 non-null object

41 gg\_text 44851 non-null object

dtypes: float64(3), object(39)

memory usage: 14.7+ MB

# Problems & Possible Improvements

In my terms of view, the main question about this task was to join Google dataset with other data. As there is no ideal join key between google and other data (as some columns have a lot of NaN values and the others could have a bunch of duplicates and so on) and we can just try different approaches based on what we need in a specific case. Also, one more question is to choose final dataset structure. In my case, I used the approach to save maximum number of columns to have a possibility to choose needed ones from a full set. This can resemble OBT (One Big Table) data modeling approach in some way.

# Useful Links

* Pandas - <https://pandas.pydata.org/docs/>
* Regular expressions - <https://docs.python.org/3/library/re.html>
* Matplotlib - <https://matplotlib.org/stable/users/explain/quick_start.html>