Exadel assignment

# The problem

We have three datasets (Google, Facebook, and Web) and we intend to merge those datasets into one with the highest possible data accuracy.

# The Solution

## Data inspection and load

First, we need to familiarize ourselves with the data. I have been investigating the data in different ways: by just looking using Numbers, printing out some metadata, and diving deep into certain columns that looked like a good candidate for merging columns. Introspection of the datasets was also done in my preferred IDE, which allowed me to properly see what delimiters, quote characters and escape characters were used in each one. This information was useful because now I knew how to load that data using pandas without having to clean data beforehand, create new files, or similar. After loading the data, I exported it to a new CSV file to check and be sure that everything was formatted correctly. It was.

### Google dataset

Google dataset has 356.520 rows with 15 columns. The most obvious column for introspection is the name. It has only 2621 duplicated and just 31 NaN values - almost perfect. Category has 471 unique values, which seems reasonable at first glance.

### Facebook dataset

Starting in the same manner as we did with Google DS, this one has 72.010 rows and 16 columns, with 71.838 unique company names in the name column. Once again a solid number, with only 1 NaN value. The category column, on the other hand, has 10.185 unique values (~7% of the ds) which is too much. Looking at the data itself, we can see that it is formatted in a different way than the Google dataset. This one has multiple categories in the same column, separated by “|”. This is something we will need to bear in mind when we start comparing categories. Phone numbers are interpreted as an int value, in both Facebook and Google DS, so that is another column we will need to preprocess.

### Website dataset

With 72018 entries and 11 columns, this dataset has 30620 unique company names in the column named legal\_name, but also has 40029 NaN values. This is going to force us to consider some other columns for matching with other datasets. One such candidate is the phone number(name of the column: phone), containing 60054 unique values and 6378 NaN values. As far as the category (s\_category) is concerned, we have 563 unique values and 1384 NaN values. This also seems pretty good. This dataset does not contain full addresses, but it has city, country, and region with 11052, 7149, and 11085 NaN values, respectively.

## Data preprocessing

Before attempting to merge datasets, we want to have one row per company in every dataset.

Time for some deduplication. If we had any dates in our data, I would choose to keep the most recent row for the duplicates. Since we don’t, I will keep the one with the most data (the least NaN values in a row). This process will also drop all rows with NaN value in the name column. By checking the counts of NaN values in the name columns of our datasets we can conclude that those cases are in a couple of hundreds, which is safe to ignore on datasets containing almost half a million rows. If there were more NaN values, we would need to keep those rows, I would then assign empty strings to those columns for them to not be dropped, but that would also mean that name is not a good column for matching.

After taking care of duplicates, we need to do a bit of a cleanup in the name column. I deleted the following endings of the company names: llc, inc, co, llp (all case insensitive and with or without a full stop at the end). Those are enough for this assignment, but in the real world, there might be some more to be dropped. I will not drop special characters or whitespace characters in this step. The reason for that lies in the library we are going to use for our fuzzy matching, which brings us to our next chapter.

## Data matching

I have opted for a library called [RapidFuzz](https://github.com/maxbachmann/RapidFuzz), made by Max Bachmann. This library is an alternative to the well-known FuzzyWuzzy. So, why use the alternative? A couple of reasons.

1. Some tests showed that it is 16x faster than FuzzyWuzzy. FuzzyWuzzy is very slow for big datasets, which we must consider. RapidFuzz is faster because it is mostly written in C++.
2. MIT licensed
3. Fixed bugs from FuzzyWuzzy

The speed alone is enough for us to adopt RapidFuzz.

I’ve chosen a library that uses Levenshtein Distance because I consider it to be the standard in use cases such as these. I will not go into a detailed explanation of Levenshtein, since I assume the person reading this does not need clarification.

Finally, RapidFuzz offers the possibility to use processor=fuzz\_utils.default\_process

This is the reason why we did not have to clean special characters and trailing whitespace characters from our name column. This processor will do it for us, removing all non-alphanumeric characters from the string. This creates great conditions for fuzzy matching - all of our names are now lowercase, without any trailing whitespaces, and without any special characters.

While doing these assessments, I was using significantly smaller datasets than provided. Although I have opted for the most optimized solutions I could find, this process still takes a long time and my laptop could not handle the fuzzy matching of 350k and 70k of rows (I wanted to merge Google and FB first). I would bring up infrastructure to run this, probably an EC2 machine on AWS or something similar. However, I did not have the time or resources to do so for this assignment, so I will just make assumptions based on smaller datasets with the intent of checking if my assumptions are correct.

I used RapidFuzz to see if the name matching would be performed correctly and I was tweaking the threshold. I was outputting the matching score as well as the matched name in a separate column and then looking at those results. I ended up going with 85 as a threshold which did not generate any unwanted matching. Anything below resulted in the matching of different companies that just have something like “technologies” in their name since that accounts for more than half of the name. I also played around with different scorers but ended up using rapidfuzz.fuzz.ratio which calculates normalized Indel distance. That is a slight variation on Levenshtein distance but without the substitution operation.

After confirming that the name can be used for merging the sets, I will perform the merging with a library called fuzzy\_merge for simplicity purposes. I will keep all columns so we can compare the most important columns and determine if the merging was successful. All the columns are renamed so we can compare the most important ones: Category, Address(country, region...), Phone, Company names.

After confirming that matching and merging were valid, we should merge the current dataset with the website dataset. Now, this dataset has a lot of empty names, so we need to find another column for merging. I opted for the domain because that field is unique enough, and has very few NaN values in both the website dataset and our merged dataset. The other column I have considered is phone number, but the domain seemed to give better results (more merged rows).

Now that everything is merged we need a strategy of which columns to keep. My final dataset would have the following columns:

* Name
* Address
* Category
* City
* Country code
* Phone
* Region (either code or name, does not really matter that much)
* Zip code (useful for some countries, for some not that much)
* Domain
* Email

But how would we decide which dataset to trust? Well, now we have everything in one dataset, so let’s run trough the columns I’ve decided to keep.

#### Name

For the name our options are google or facebook, since website has a lot of NaN values. From these two we can keep either one, since we are matching them by name.

#### Address

Even though it is weird that the website dataset does not contain an address, this does make our life a bit easier. We can only choose from GG and FB datasets, I would keep the one that is not NaN, if both have values, that are different, I would keep the Google one, I consider google to be more up-to-date with stuff like this.

#### Category

This is the column that sometimes contains data that says the same - but in a different way. This is why I would compare the values, and if there are multiple different ones, I would concatenate and keep them all separated by |.

#### City, Country Code, Phone, Region, Zip Code, Email

For all of these I would use the same strategy. If there is data on the website dataset, I would consider that to be correct and keep that one. If not, then just keep from GG or FB, whichever is not NaN.

#### Domain

Since we have used domain to match with the website dataset, and that one has the least NaN values in the domain column, I would keep it from website dataset.

In the end, I just want to do a recap on the key questions of this assignment.

1. What column will you use to join?

As shown in the section above, for joining GG and FB dataset, name with fuzzy matching, threshold 85. Then domain column to match that one with the website dataset. I would increase the threshold here to 90-95 since there is little to no room in a domain name (no special characters, no whitespace etc)

2. If you have data conflicts once you join, which one do you believe?

As described above for each column I would keep in the final dataset.

3. If you have very similar data, what information will you keep?

As described above for each column I would keep in the final dataset.

# Final thoughts

I only kept 10 columns in the final dataset because I like to keep the data tidy, cluttering with a bunch of columns can oftentimes create confusion for the person using the data. I would also talk to the person who will use this data, in order to get all of the required information.

I would execute this code in a cloud infrastructure instead of using my own laptop because that proved to be very inefficient and slow. I made some assumptions while using the smaller dataset, but I would check all of those if I had the supporting infra.

The columns for the final dataframe were not implemented in code because I was running low on time and I considered it would be better to describe my thought process than to actually implement it because that part is trivial compared to the rest of the assignment.

Lastly, I have one ask for the person reading this. Would it be possible to see the desired solution for the task? I think it would benefit me for my personal growth if I could get access to the code that you consider to be as close to ideal as you have.