Summer Internship Project
Report

Know Your Customer (KYC)

Submitted by

Othmane Cherai
1^{st} year student
Ecole Nationale Supérieure de l'Electronique et de ses Applications
(ENSEA)

Under the guidance of

Tarik Bouramdan
Directeur Edition
(failed to find an english equivalent)

IT Dev Team
Perenity Software. LLC
N° 42 Bd Abdelmoumen, Casablanca

Abstract

Being a first year student at **ENSEA** which is a french IT school, I am obliged, minimally, to a 1 month job-shadowing internship in a certain company. The main purpose of the said internship is to help cultivate the student beyond the academics taught within the school. In fact, only through such experience, will he acquire a broad vision of what the work routine really looks like, and soak in the very essence of his career's culmination: "a job". Given the said above, all the seemingly trivial parameters are to be factored in when in search of an internship, for the said internship will greatly help guide him. In my case, Despite being in an electronics and IT school, I have chosen, weirdly enough, to orient myself to the financial sector and more precisely to market finance's sector. Within this vision, this internship can be considered a first step towards my conversion for the following criteria are all met.

1. A company operating in the financial sector.
2. My internship subject has to involve maths/statistics and coding.
3. Reinforcing the still-building finance's knowledge through discussions with active-within-this-sector employees.

Perenity Software offers an asset managing solution: **Manar**. The said solution allows a wide range of functionalities than can be used by groups such as hedge funds and investment banks as well as portfolio managers or simple individuals. For instance, the module **Collatéral** permits easy management of collateral when a certain operation involves lending an asset. "Repo" or "Repurchase" contracts could be the ideal case for using this module as it guarantees a certain procedure will be triggered if ever a party defaults from the initial contract.
Given the scope of the financial domain, a set of procedures is in order so as to make sure no wanted individual has access to the financial systems. These individuals'data can be given by governmental or international institutions(MFA for example) and are mostly confidential. They are used to establish similarity procedures that deem an individual's right to access a given service. **Know Your Customer(KYC)** is the solution developed by **Perenity Software**.

Abstract(Français)

Étant actuellement élève en première année à l'**ENSEA**, Je suis dans l'obligation de faire un stage ouvrier d'une durée minimal d'un mois. L'objectif dudit stage est de permettre à l'élève de peaufiner les acquis académique dont il s'est imprégné le long de sa première année. Car, sans telle expérience, le concerné ne pourra jamais saisir ce qu'est l'essence de ce qu'on appel "travail". Ainsi, l'élève devra considérer dans sa recherche de stage l'ensemble des paramètres caractérisant les proposition qui lui sont présentés. En ce qui me concerne, ayant décidé de me convertir vers le monde de la finance quantitative, le stage que j'entretiendrai devais répondre à plusieurs critères:

1. La société doit être active dans le secteur financier.
2. Le stage devrait inclure dans les missions qui me seront attribuées, les maths/statistiques et de la programmation.
3. Renforcer si ce n'est ancrer les connaissances, progressivement acquise à travers mes lectures, en finance.

Perenity Software offre une solution de gestion des actifs intitulée **Manar**. Celle-ci offre un éventail de fonctionnalités que trouvera fort utile les fonds de couvertures ainsi que les banques d'investissement. Le module **Collatéral** permet de gérer les contrats de mise en pension dans le cas où l'une des partie de respectent pas les termes du contrats.
Étant donnée les enjeux du domaine financier, certaines procédures sont de rigueur afin d'empêcher certains profiles notamment les recherchés de l'intégrer. Des organismes gouvernementaux et d'autres internationaux, afin de permettre aux sociétés de satisfaire à cette obligation, fournissent les listes des personnes recherché afin de permettre de développer des méthodes d'identification et de calcul de similitude. **Know Your Customer(KYC)** est dans le cas de la société hôte, la solution développé afin de remedier à ce besoin.

Contents

Chapter 1

Data Analysis

All tests and explanations will be based on the UN watch list (which can be accessed on the UN website) for reasons of secrecy, as the other watch lists are covered by the confidentiality agreement I signed. In addition, portions of the code - particularly those that process data from restricted-access watchlists - will not be revealed. However, I will grant complete access to the section responsible for processing the UN watchlist. This section will discuss the steps I took to develop a solid method for calculating the similarity rate between two individuals.
The first challenge that arises when attempting to evaluate similarity is determining the appropriate criteria; should the name be used? Should the birth date be heavily considered? Should eye color be taken into consideration? Or, should the place of birth be the primary criterion? Should we have considered the name? Many individuals share the same name but spell it differently, while others have many aliases. Others have complex names, while some have simple names. Consequently, a detailed analysis was required. But first, let's establish a set of universal parameters that will be used to determine the desired degree of similarity. These are the parameters: **Full name, Nationality, Address, ID number, Date of Birth, and Passport Number**. These factors will be explained in the preceding sequence to facilitate reading.

The name is the first parameter to be considered. A thorough examination of the UN watch lists provides several insights into the factors that control the entity 'Name'. Clearly, the greatest obstacle will be taking into consideration the aliases that many persons appear to have. In reality, over sixty percent of the desired profiles had aliases. The range is between 1 and 7. In some circumstances, these aliases are entirely distinct from the original name, while in others, they are created by adding, deleting, or exchanging words, such as **"El, Al, Sheikh"** for arabs and **"Luo, Ol, Pak"** and other short words for asian names. In addition, the average name length on this list appears to be 2.3 words.

Nationality, Address, Id number, Birth date and Passport Number can be considered random and do not require special attention when analysing them, which is as expected. One sole thing is worth mentioning: the dates in the UN watch list range from the early 60s.

Chapter 2

Weight Calculus and MonteCarlo Simulation

The weight calculation is a crucial stage since the more accurate and realistic it is, the more trustworthy the findings will be. Therefore, a probabilistic technique was required to compute them. I will employ the same dichotomy as in the previous chapter when it comes to order.

2.1 Probability calculus

2.1.1 Name

The name's weight is a crucial criterion, if not the most crucial, as it is the only distinguishing characteristic of the vast majority of profiles. When other parameters are included, it is possible to resolve the few instances in which two persons have the same name. Calculating the name's weight relative to other factors is equivalent to determining the likelihood of two individuals sharing the same name and comparing it to the chance of two individuals sharing the same birth date, address, etc. At the end of this chapter, the weight calculus will be explained in further depth.
Given that we have a complete list of names, the simple but incorrect technique for calculating the probability would be to assume that the names are distributed linearly. Even though the concept is appealing and time-saving, it is nonetheless incorrect. In addition, I did not have access to an exhaustive list of names. I have spent a great deal of time attempting to find a study that examines the distribution of names, but I have been unable to locate any, or at least I do not have access to them. I then resorted to the Zipf law on the assumption that names are essentially words derived from a dictionary, but this did not appear to be accurate. As for the normal distribution, I have not used it since it requires so many parameters, but it has been utilised in the Monte Carlo Simulation for the production of names.

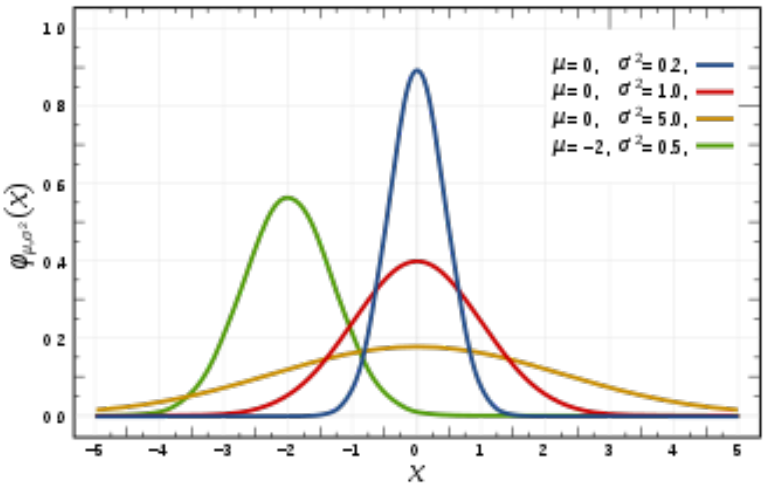


Figure 2.1: Probability density function for a normal distribution. The red curve is that of the standard normal distribution.

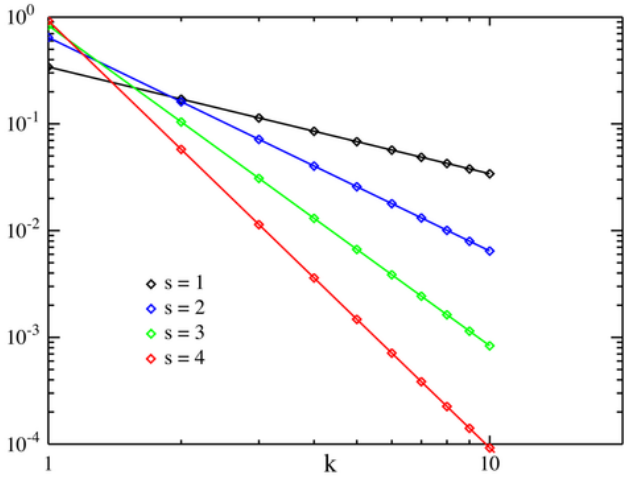


Figure 2.2: Zipf PMF for N = 10 on a log–log scale. The horizontal axis is the index k . (Note that the function is only defined at integer values of k. The connecting lines do not indicate continuity.)

Having found no ready-to-use result, The following demonstration was established. Let there be a group of n persons.

Define K the names set and card(K) = k.

- X_i: The random variable: person **i**'s name, where i ∈ K.
- A: The event $\exists i \neq j, X_i \neq X_j$ where i,j ∈ K
- $\overline{A}$: A's contrary event
- $P_{n,k}$: A's probability

Define "i" ∈ K and Z_i, the number of persons sharing the same name as "i"($Z_i = \text{card}(A_i)$
As such $A_i \neq \emptyset \Leftrightarrow Z_i \geq 1$
A and $\overline{A}$ are complementary event, thus $P(A) = P_{n,k} = 1 - \overline{P_{n,k}} = 1 - P(\overline{A})$

For easy calculus, consider $P(\overline{A})$

$$P(\overline{A}) = \frac{n(n-1)(n-2)...(n-k+1)}{n^k} \tag{2.1}$$

$$1 - P_{n,k} = (1 - \frac{1}{n})(1 - \frac{2}{n})...(1 - \frac{k+1}{n}) \tag{2.2}$$

Considering the proportion(n >> K), e^t's Taylor's expansion gives:

$$P_{n,k} = 1 - \prod_{n=0}^{k-1} e^{-\frac{i}{n}} = 1 - e^{\frac{\sum_n^{k-1} i}{n}} = 1 - e^{-\frac{k(k-1)}{2n}} \tag{2.3}$$

It is evident that the computed probability is large if 'n' and 'k' have the same magnitude.

2.1.2 Date

As surprising as it may seem, the probability of two persons having the same name with a set of K names and N persons is exactly the same as that of finding two persons with the same birth date in a set of N persons and K dates. What differs in this case, is the number "K". After analysing the watch list, K was deduced to be equal to $365 \times (2022 - 18 - 1960)$ (wanted lists profiles'ages are all above 18). The aforementioned calculation is applicable to this scenario and will be incorporated into the final weights distribution.

2.1.3 Nationality, Address, ID number, Passport Number

All of the aforementioned parameters are assumed to follow a linear distribution. They differ solely in the universe. Normally, the probability that two people share a nationality should be determined by factoring in the population of each country, however this will not be done because this parameter is of little value for KYC reasons. It is assumed that the likelihood is 1 over the number of nations in the world.
Similarly, the remaining parameters are likewise determined. In accordance with ICAO requirements, the length of a passport number must be nine characters. In actuality, however, a statistical analysis employing the passport numbers available at this link: `https://github.com/vaasha/Data-Analysis_in-Examples/blob/master/EDA_Passport_Numbers/data.csv` demonstrates that passport length may range from 3 to 17 digits, but infrequently. Despite this, the likelihood that a wanted individual will present themselves to a particular agency using his "wanted" passport number is so remote that the weight of this criterion is trivially reduced. If a match occurs, however, the similarity should be set to a high percentage. The chance that the desired individual is using another person's passport is still considered, thus the similarity is not necessarily 100%

2.2 Weight calculus

Consider a collection of parameters with associated probabilities. Intuitively, the higher the chance that two individuals share one of the evoked factors, the lower the weight should be applied to that parameter, as KYC is about identifying minor similarities that might assure that two individuals are the same or different. Police and forensics utilise DNA due to the extremely low likelihood of two individuals sharing even a single base pair of their DNA. On the other hand, hair colour is not a reliable identifier due to the frequency with which two individuals have the same hair colour. Using this rule as a guide, the following calculus was

developed.
Consider P_{x_i} the probability of two persons sharing the parameter x_i.

$$P_{x_1} = k_{x_1,x_2} \times P_{x_2} \tag{2.4}$$

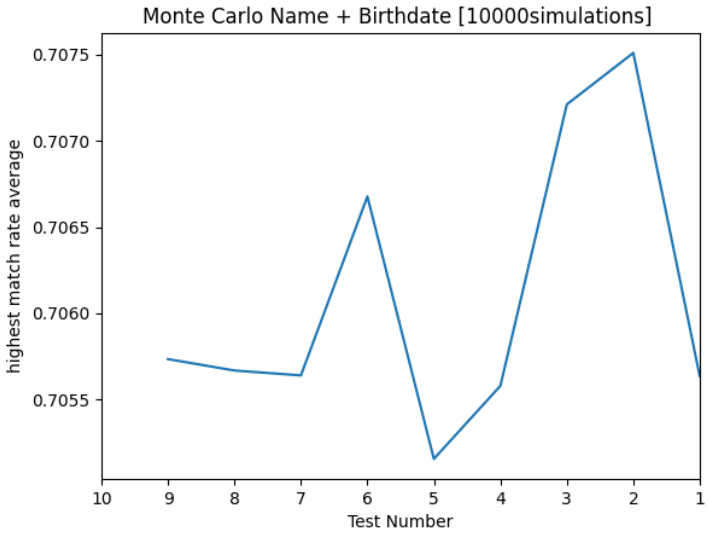
$$P_{x_2} = k_{x_2,x_3} \times P_{x_3} \tag{2.5}$$

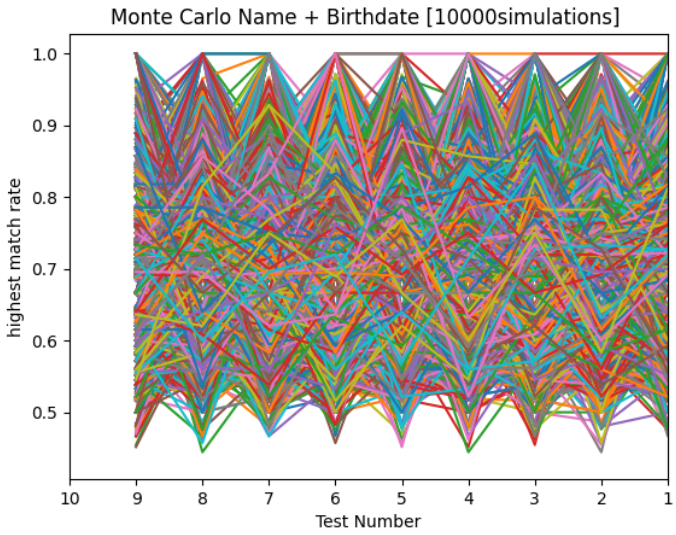
$$P_{x_i} = k_{x_i,x_{i+1}} \times P_{x_{i+1}} \tag{2.6}$$

We then resolve the following system:

$$\begin{cases} P_i = x_{i,j} \times P_j \; i,j \in I = (\text{parameters}), i \neq j \\ \sum_{i \in I} P_i = 100 \end{cases} \tag{2.7}$$

We then inverse the values such as the biggest value is assigned the smallest value.
A MonteCarlo simulation was conducted to validate the results (more than 1 million samples). (the simulation code can be disclosed is not protected by the agreement and will be disclosed if needed.)





2.3 MonteCarlo Simulation

In quantitative fields, MonteCarlo simulation is used to create random samples in order to imitate a real-world event. On the basis of these samples, the hypothesised theory is either verified or denied. If the theory is shown to be erroneous, the initial hypothesis is rewritten and retested while taking the simulation's results into consideration. The greater the number of samples collected, the more accurate and realistic the results will be. In my example, the simulation was used to determine the average similarity rate between names created randomly according to a normal distribution and dates generated linearly over a 60-year span.
The script may be divided into two major sections, the first being the generating process and the second being the testing procedure.

2.3.1 generation process

The generation process may be separated into two major parts: the generation of wanted lists and the generation of tested individuals. Numerous aspects are considered while compiling the wanted list, including the ratio of males to females in the selection of names.
First, we will consider the names. The names are produced using the **Names** library in Python, which is an MIT-licensed project whose primary objective is to provide a library that can generate names based on criteria such as: complete name, initial name, family name, male name, female name... All the names are sourced from the 1990 public domain US census data. As America is a multicultural nation, there is a good likelihood that we may encounter names from all ethnic groups. Given the nature of the project for which this simulation is intended, the use of the aforementioned library can be permitted. The first step was to generate a male names list then a female names lists and finally a family names list.

```
1  def randomMaleName():
2      randFirstNameMale = names.get_first_name(gender = 'male')
3      return randFirstNameMale
4
5  def randomFemaleName():
6      randFirstNameFemale = names.get_first_name(gender = "female")
7      return randFirstNameFemale
8  def randomFamilyName():
9      randFamilyName = names.get_last_name()
10     return randFamilyName
11 for i in range(2000):
12     maleNames.append(randomMaleName())
13     maleNames = list(maleNames)
14 for i in range(2000):
15     femaleNames.append(randomFemaleName())
16     femaleNames = list(femaleNames)
17 for i in range(4000):
18     familyNames.append(randomFamilyName())
19     familyNames = list(familyNames)
```

Listing 2.1: Name Generation

As for the production of the wanted list, one parameter was still needed; male to female proportion in wanted lists. No country or international organisation disclosed this information, and the ratio of males to women on the UN watch list was 95 to 1. Given that individuals in jail were formerly labelled as "wanted", the inconclusive nature of these statistics led me to believe that this ratio should be equal to the male-to-female prison ratio. According to **Prison Studies** `https://www.prisonstudies.org/sites/default/files/resources/downloads/world_prison_population_list_11th_edition_0.pdf`, the percentage of males in incarceration is 93.2% .
The gender of the created individual is decided using a Bernoulli distribution whose success

parameter is equal to that percentage. Using a normal distribution for the names seems to be the most trustworthy approach. The parameters of the distribution were selected so as to make the density function as flat as possible. Then, we would produce random numbers and trap them between 0 and the list's length(-1) to select the desired person's first name. Family names are completely random and are thus generated randomly. In order to duly simulate a watch list, aliases were added by simply doubling a letter in a name. The number of aliases was decided by a normal distribution that went accordingly to the data found in the UN watch list concerning the number of aliases wanted individuals were using. In rare cases, **"EL", "Luo", "Pak", "apostrophe (')"...** were added either in the beginning or between the first and family name for better representation purposes.

```
1       for i in range(listLen):
2           rnd = bernoulli.rvs(maleProportionInPrison, femaleProportionInPrison
    )
3           if rnd == 1:
4               #k = np.random.normal(1000 ,400)
5               k = normalDis1NumberGen(0,len(maleNames)-1, 1000, 600)
6               finalList.append(maleNames[k] + " " + familyNames[random.randint
    (0, len(familyNames)-1)])
7           else:
8               k = normalDis1NumberGen(0, len(femaleNames)-1, 1000, 600)
9               finalList.append(femaleNames[k] + " " + familyNames[random.
    randint(0, len(familyNames)-1)])
10      return finalList
```

Listing 2.2: Wanted list generation

2.3.2 Similarity Calculus

The rate of name similarity is computed using the **Sequence Matcher** function from the difflib package. However, a number of name and known alias combinations are evaluated to get an exhaustive list of potential names the wanted profile may be using. This function's specifics cannot be divulged. As for the Sequence Matcher, it simply separates the word into many sequences while rearranging the component order. Then, it compares each word and returns the proportion of matches. This proportion is then multiplied by the weight of the parameter "Name". The same is true for dates and nationalities, for which similarity is dichotomous; dates are either identical or distinct. The resulting percentage indicates the degree of similarity (or dissimilarity) between the two profiles.

The section that follows is essentially a compilation of all the data (name, date, and country) and is displayed below. Using the same weights, the two functions provide a means of comparing the influence on similarity if two parameters are given the same weight.

```
1def rdmSmlrtyNameDateprofile(testNumber : int):
2       global wantedListNameDate
3       rate = 0
4       rates = [[] for _ in range(testNumber)]
5       for i in range(testNumber):
6           randomIndividual = [(randomMaleName() + randomFamilyName()).lower(),
    randomCountryGenerator(1)]
7           for j in range(len(wantedListNameDate)):
8               rates[i].append(SequenceMatcher(None, randomIndividual[0],
    wantedListNameDate[j][0]).ratio() * 0.5 + (randomIndividual[1][0].__eq__(
    wantedListNameDate[j][1]))* 0.5)
9           rates[i] = sum(rates[i])/len(rates[i])
```

```
10
11      return rates
12
13  #a = rdmSmlrtyNameDateprofile(10000)
14
15
16
17  def rdmSmlrtyNameCntryprofile(testNumber : int):
18      global wantedListNameCountry
19      rate = 0
20      rates = [[] for _ in range(testNumber)]
21      for i in range(testNumber):
22          randomIndividual = [(randomMaleName() + randomFamilyName()).lower(),
    randomCountryGenerator(1)]
23          for j in range(len(wantedListNameCountry)):
24              rates[i].append(SequenceMatcher(None, randomIndividual[0],
    wantedListNameCountry[j][0]).ratio() * 0 + (randomIndividual[1][0].__eq__(
    wantedListNameCountry[j][1]))* 1)
25          rates[i] = sum(rates[i])/len(rates[i])
26
27      return rates
```

Listing 2.3: Similarity test using Name and Date vs Name and Country

Other tests were conducted so as to see to what would the average name similarity look like. The function responsible for that is **tstSmlrtyRdmNameInList** which is as follow.

```
1   def tstSmlrtyRdmNameInList(randomMaleFemalListK: list):
2       # Generate Random name
3       global maleNames
4       global femaleNames
5       global familyNames
6
7       maleProportionInAfrica, femaleProportioninAfrica = 0.499, 0.501
8       #Generate rate table:
9       rates = np.zeros((maxSims,maxNames))
10
11      for i in range(maxSims):
12          testNames = []
13          for l in range(maxNames):
14              rnd = bernoulli.rvs(maleProportionInAfrica,
    femaleProportioninAfrica)
15              if rnd == 1:
16                  k = normalDis1NumberGen(0, len(maleNames), 1000, 300)
17                  testNames.append(maleNames[k] + " " + familyNames[random.
    randint(0, len(familyNames)-1)])
18              else:
19                  k = normalDis1NumberGen(0, len(femaleNames), 1000, 300)
20                  testNames.append(femaleNames[k] + " " + familyNames[random.
    randint(0, len(familyNames)-1)])
21          for j in range(len(testNames)):
22              for k in range(len(randomMaleFemalListK)):
23                  #r = compareName(testNames[j], randomMaleFemaleList[k])
24                  r = SequenceMatcher(None, testNames[j], randomMaleFemalListK
    [k]).ratio()
25                  if r > rates[i][j]:
26                      rates[i][j] = r
27          print(rates)
28          plt.plot(range(len(testNames)),rates[i])
29      plt.show()
30
```

```
31        fig2 = plt.figure()
32        plt.title("Monte Carlo Name + Birthdate [" + str(maxSims) + "simulations
    ] ")
33        plt.xlabel("Test Number")
34        plt.ylabel("highest match rate average")
35        plt.xlim(maxNames)
36
37        avRates = []
38        ratesK = np.array(rates)
39        ratesK = np.transpose(ratesK)
40        for i in range(len(ratesK)):
41            avRates.append(sum(ratesK[i]) / len(ratesK[i]))
42
43        plt.plot(range(len(testNames)), avRates)
44        plt.show()
```

Listing 2.4: "tstSmlrtyRdmNameInList" code. The Similarity calculus function has been ommited and replaced with a simple Sequence matcher for confidentiality purposes.

Chapter 3

Python Code(UN list case)

3.1 XML to SQL

the available data list found in the following link `https://scsanctions.un.org/resources/xml/en/consolidated.xml`. The data contained in this xml file was parsed using an automaton then placed in two SQL tables who were linked with a one-to-many relationship; the tables being named watchList and watchlistDetail. The requests used for making the tables are available as well as the .xls files grouping the data.
watchList includes generic information such as first name, last name, UN list category, and list ID. In contrast, watchListDetail organizes all other accessible information on watchList profiles. The concept of two tables originates from the fact that not all profiles share the same data type, since some profiles have nationalities and parents' names while others do not. However, processing data using both of these tables proved to be extremely challenging, especially when working with a SQL database containing many lines with the same ID. Considering how difficult it would be to approach it with python code (the script would have to reread name and id many times for certain persons), I've chosen to combine these two tables into a single one that would include all of the data. As with empty cells, the value "None" will be assigned. The SQL query used to combine these two tables is:

```
1  MERGE INTO    tpc.watch_list dest
2  USING    (SELECT * FROM tpc.watch_list_detail) src
3  ON       (dest.code = src.code AND dest.type_data = src.type_data)
4  WHEN MATCHED
5  THEN UPDATE SET
6                  dest.data_1 = src.data_1,
7                  dest.data_2 = src.data_2
8  WHEN NOT MATCHED
9  THEN
10 INSERT   (code,
11           type_compte,
12           nom,
13           prenom,
14           listed_on,
15           code_liste,
16           type_data,
17           data_1,
18           data_2)
19 VALUES   (src.code,
20           (SELECT DISTINCT type_compte FROM tpc.watch_list w WHERE w.code =
   src.code),
21           (SELECT DISTINCT nom FROM tpc.watch_list w WHERE w.code = src.code),
```

```
22          (SELECT DISTINCT prenom FROM tpc.watch_list w WHERE w.code = src.
    code),
23          (SELECT DISTINCT listed_on FROM tpc.watch_list w WHERE w.code = src.
    code),
24          src.code_liste,
25          src.type_data,
26          src.data_1,
27          src.data_2);
28
29  DELETE FROM tpc.watch_list a
30  WHERE a.code in (SELECT DISTINCT d.code FROM tpc.watch_list_detail d)
31  AND a.type_data is NULL
32  AND a.data_1 is NULL
33  AND a.data_2 is NULL;
```

3.2 Python Code

3.2.1 Automated Weight Calculus

As seen in the above demonstration, the size of the sampled population is relevant for calculating the probability that two people have the same name. Consequently, the weight of the parameter "name" must be readjusted every time the database size changes. For this reason, the weight calculation has to be automated first. Each time the code is built, the weight is recalculated using the code shown below.

```
1  K = len(DBList)
2  n = 1-exp(-(K*(K-1))/(2*4500**(2.3)))
3  j = 1/365
4  s = 0.5
5  p = 1/247
6
7  toSortList = sorted([n, j, s, p], reverse=True)
8
9  x_nj = n/j
10 x_js = j/s
11 x_sp = s/p
12
13 A = np.array([[1, -x_nj, 0, 0], [0, 1, -x_js, 0],
14              [0, 0, 1, -x_sp], [1, 1, 1, 1]])
15 B = np.array([0, 0, 0, 85])
16 x = sorted(np.linalg.solve(A, B))
17
18 p_n = x[3]
19 p_j = x[toSortList.index(j)-1]
20 p_s = x[1]
21 p_p = x[toSortList.index(p)]
22
23 p_n = x[toSortList.index(n)]
24 p_j = x[toSortList.index(s)]
25 p_s = x[toSortList.index(p)]
26 p_p = x[toSortList.index(j)]
```

Listing 3.1: Automated Weight Calculus

3.2.2 Database formatting within a List

Due to personal preferences and job dexterity considerations, I have chosen to load the DB as a list within the code. This list comprises the information deemed relevant to the calculation of similarity. The list will have the following structure: **[id, account type, [list of names and aliases], codeList, [Nationalities], [Addresses], [ID card number], [Passport number], and [Date of birth]]**. This section of code parses the database line by line and populates each desired person's data in a single line, as opposed to the multiple lines contained in the database. Once this is complete, any empty lines created by this operation are removed. For clarity, please refer to the diagram below.

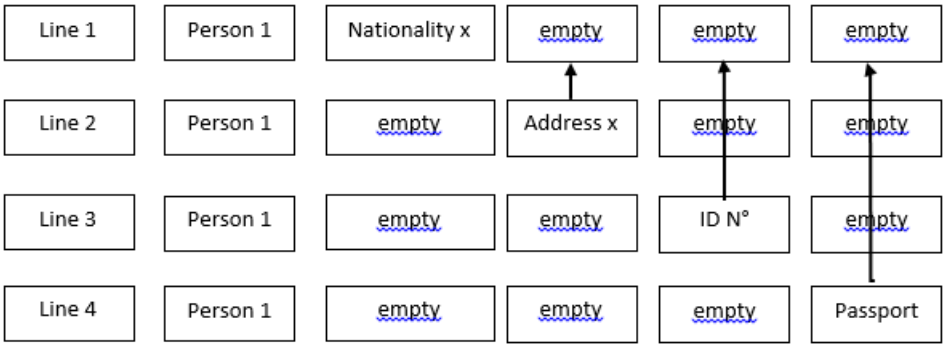


Figure 3.1: Drawing explaining the parsing and filling process

```
### remaker
## Input : DataBase table formated using the SQL request given in the same file
## Output: Databse formated as : ["ID", "Type", "Name", "Codelist", "
    Nationalities", "adresses", "ID card number", "Passport", "BirthDate" ]
## Description : This function is simply a parser that rewrite the DB list in an
     appropriate format for easier computing
def remaker(table: list):       #formatting the DBList to the format I chose
    l = []
    for i in range(len(table)):
        l.append([0,0,[],0,[],[],[],[],[]])
    for i in range(0,len(table)):
        l[i][0] = table[i][0]                           #id
        l[i][1] = table[i][1]                           #type compte
        l[i][2] = table[i][2] + ' '+table[i][3]         #name
        l[i][3] = table[i][5]                           #codeList
        if table[i][6].lower() == 'nationality':
            l[i][4].append(table[i][7])
        if table[i][6].lower() == 'adresse':
            l[i][5].append(table[i][7] + ', ' + table[i][8])
        if table[i][6].lower() == 'piece_identite':
            if table[i][7].lower() == "national identification number":
                l[i][6].append(table[i][8])
            if table[i][7].lower() == "passport":
                l[i][8].append(table[i][8])
        if table[i][6].lower() == 'date_naissance':
            l[i][7].append(table[i][7])
        indice = 0
    while indice < len(l):
        if indice+1 == len(l):
            break
        if l[indice][0] != l[indice+1][0]:
            indice +=1
```

```
31      elif l[indice][0] == l[indice+1][0]:
32          for k in range(4,len(l[0])):
33              #if (l[indice][k]== 0 and l[indice+1][k]!=0):
34              #if (l[indice][k] == [] and l[indice+1][k] != []):
35              if (l[indice][k] != l[indice+1][k] and l[indice+1][k] != []):
36                  l[indice][k].append(l[indice+1][k][0])
37          l.remove(l[indice+1])
38      #if l[indice] == [0,0,0,0,0,0,0,0,0]:
39      #   for i in range(indice, len(l)):
40      #       l.remove(l[indice])
41
42  return l
```

Listing 3.2: remaker function. read comment for better understandinf of inputs and outputs formats.

For consistency with Python syntax and other functions not yet encountered in the remainder of the manual, the **zeroToNone** function was added. It only replaces empty cells with [”**none**”].

```
### perm
## Input  : the name string
## Output : returns a list with all possible permutations than can be made using
    the name list
def perm(name: str) -> list:
    finalList = []
    l = name.split(" ")  # mettre chaque mot de la liste dans une case
    finalList = list(permutations(l))  # liste des permutations
    for i in range(len(finalList)):
        finalList[i] = ' '.join(finalList[i])         # joindre chaque tuple de
   la liste pour former un nom
    return finalList
```

Listing 3.3: perm function code.

3.2.3 Similitude calculus for Names.

Before the similitude calculus for names could be developed, a collection of previously mentioned functions had to be constructed. They exist to ensure that the similitude calculation for names accounts for all possible outcomes. The comparison approach entails generating an exhaustive list of all potential permutations by combining each word of names and aliases in a series of combinations. The similitude calculation is then regarded credible and completed.

```
1  ### perm
2  ## Input  : the name string
3  ## Output : returns a list with all possible permutations than can be made
   using the name list
4  def perm(name: str) -> list:
5      finalList = []
6      l = name.split(" ")  # mettre chaque mot de la liste dans une case
7      finalList = list(permutations(l))  # liste des permutations
8      for i in range(len(finalList)):
9          finalList[i] = ' '.join(finalList[i])         # joindre chaque tuple
    de la liste pour former un nom
10     return finalList
11
12
13
14
```

```
15  ### compareName
16  ## Input  : finalList        : is the client's name permutation list
17  ##          nameAndAliasList : is the wanted profile's name and alias list
18  ## Output : returns a list [rate, client's name, highest matching wanted
    name]
19  ##          the rate is the highest one found
20  ## Description : compare a name with all possibe combinations that can be
    made with wanted's name and his aliases
21  ### converter
22  ## Input  : a date in string format
23  ## Output : returns that date in EU date format
24  def compareName(finalList: list, nameAndAliasList: list):
25      rate = -1
26      finalListOriginal = finalList.copy()
27      finalList = list(map(lambda x: x.lower(),finalList ))
28      nameAndAliasList = list(map(lambda x: x.lower(),nameAndAliasList ))
29      for i in range(len(finalList)):
30          for j in range(len(nameAndAliasList)):
31              r = SequenceMatcher(None, finalList[i], nameAndAliasList[j]).
    ratio()
32              if rate < r:
33                  rate = r  # on choisit le nom avec le plus de similitude
34                  indexPermList = i
35                  indexNameAndAliasList = j
36                  if r == 1:  # si on a similitude parfaite, pas la peine de
    chercher plus
37                      break
38      return ([rate, finalListOriginal[indexPermList], # renvoie le nom avec
    la plus grande ressemblance, le pourcentage de ressembence
39               nameAndAliasList[indexNameAndAliasList]])   # et le nom de la
    personne
```

Listing 3.4: perm function code.

The retained result of the function has the highest match rate of all those checked. Additional code details are explicitly described in the preceding script (read comments).

3.2.4 Date Comparison

Comparing dates was not a laborious coding effort. The only issue that required consideration was date formats. In actuality, date formats vary from country to country. Therefore, all dates have to be changed to a format (in this case, EU format) before being compared.

```
1   ### compareDate
## Input : birthDate     : is the clients birthdate
##       : birthDateList : is the wanted profile datelist(one profile can have
    many dates)
## Output: return 0 if there is no match
##       : return 1/(len(birthDateList)) if there is a match
6   def compareDate(birthDate, birthDateList) -> int:
7       birthDate = converter(birthDate)
8       birthDateList = converter(birthDateList)
9       if birthDateList == "none" or birthDate == "none":
10          return 0
11
12      birthDate = list(map(lambda x: x.split("-"), birthDate))
13      birthDate = list(map(lambda x: datetime(int(x[2]), int(x[1]), int(x[0])),
    birthDate))
14
```

```
15
16  b = birthDateList
17  b = list(map(lambda x: x.split("-"), b))
18  b = list(map(lambda x:  datetime(int(x[2]), int(x[1]), int(x[0])), b))
19  for i in range(len(b)):
20      if b[i] == birthDate[0]:
21          #normalement si une date est egale, on renvoie juste 1, mais dans l'
    eventualite ou on a plusieures dates de naissances
22          return 1/len(b)
23      else:  # on rajoute cette ligne
24          return 0
```

Listing 3.5: Date comparison function.

3.2.5 Other Data Comparison

Other Data are simply compared one to one. If the data matches, 1 is returned, otherwise 0. In the eventuality where one wanted individual may have used many data of the same type; for instance has had numerous passports, each of these passports will be compared to the client's genuine passport. The other data undergoes the same procedure.

```
1  def comp(s1, s2):
2      if s1 == ['none'] or s2 == ['none']:
3          return 0
4      for j in range(len(s2)):
5          if s1[0].__eq__(s2[j]):
6              return 1
7      return 0
```

Listing 3.6: Other Data comparison function

3.2.6 Resemblance to Data

Resemblance to Data, or as I like to call it, "Where the magic happens," is where all the data are processed and the similarity rate is calculated. Note that the shown weights are not the actual weights utilised. As demonstrated plainly in the code below (line 13), only profiles of the same kind are compared (Entities are only compares to entities and individuals only to individuals). In rare instances, a profile may lack data, necessitating that the parameter's weight be set to 0. Lines 15 to 26 are responsible for carrying out this function.

```
1  ###resemblanceToData
2  ## Input  : listC  : a list containing the profile's data
3  ##        : listCC : a list containing the wanted profile's data
4  ## Output : a resemblance rate
5  ## Description : The main part is the weight calculation. When a resemblance
    criteria is not contained in either the
6  ##               client's data or the wanted profile's data, its weight is
    distributed evenly between all criteria
7  ##               which are contained in both profiles.
8  ##               Comparison is limited to same Type profile. I.e we can only
    compare individuals to individuals and
9  ##               entities to entities
10 def resemblanceToData(listC: list, listCC: list):
11     weight=[37.9478,37.9478,15.8128,3.79478, 3.45705, 1.03977]
12     spareWeight = 1
13     if listCC[6].__eq__('ENTITY'):
```

```
14          weight=[100,0, 0, 0, 0,0]
15      for i in range(0,len(listCC)):
16              #if (listCC[i].__eq__("none") or listC[i].__eq__("none")):
17              if (listCC[i] == ["none"] or listC[i] == ["none"] or listCC[i]
    == "none" or listC[i] == "none"):
18                  #for k in range(len(weight)):
19                      #if not (listCC[k].__eq__("none") or listC[k].__eq__("
    none")) :
20                          #weight[k] += 1/min([len(listCC)-1 - listCC[0:6].
    count("none"),len(listC)-1 - listC[0:6].count("none")])*weight[i])
21                  #weight[i] = 0
22                  spareWeight += weight[i]
23                  weight[i] = 0
24      for k in range(len(weight)):
25          if not(weight[k]==0):
26              weight[k] += spareWeight / (len(weight)-weight.count(0))
27
28
29
30  ##weight format: [name, nationality, address, id card number, birthdate,
    passport number]
31      resemblance = [0,0,0,0,0,0]
32
33      resemblance[0] = compareName(perm(listC[0]), [listCC[0]])[0] * weight[0]
     # name
34      resemblance[1] = comp(listC[1],listCC[1]) * weight[1] # nationality
35      resemblance[2] = comp(listC[2] , listCC[2])  * weight[2] # address
36      resemblance[3] = comp(listC[3] , listCC[3]) * weight[3] # id card number
37      resemblance[4] = compareDate(listC[4], listCC[4]) * weight[4]  #
    birthdate
38      resemblance[5] = comp(listC[5], listCC[5]) * weight[5] # passport number
39
40
41      return(sum(resemblance))
```

Listing 3.7: Resemblance to Data function script.

3.2.7 csvCompare

The csv in the function's name is derived from this code's earliest version in which the entry data was written in a csv file, Whereas now they are written in a different DTO format: Json. This function simply compares the data in each line of the list to the data provided by the client. Following data, a rate table is constructed and then reorganized using the pandas package to generate a similarity rate table in decreasing order. This table is then written in the Json output file

```
1  ### csvCompare
2  ## Input  : inputList : is the client's data
3  ##        : DBLits    : is the wanted profile's data
4  ## Output : returns a list of all matching rates, the client's name, the
    highest matching profile's ID and its name
5  ## Desciprtion : takes the client's data and puts what is needed for the
    resemblance is a list. Then does the same
6  ##               for each wanted profile. Calls "cvsCompare" and store the
    rate in a list to be returned in the end.
7  ##               In case you need only the highest matching profile,
    decomment what is commented, and comment lines 266
8  ##               to 269
```

```
9  def csvCompare(inputList: list, DBList: list) -> list:
10
11      global InputList
12      column_set = [
13          "rate",
14          "wantedID",
15          "wantedName"
16          ]
17      DBList = zeroToNone(remaker(DBList))
18      InputList = zeroToNone((InputList))
19      rate = [0]*len(inputList)
20      #finalList = [0]*len(inputList)
21      finalList = []
22      index  = [0]*len(inputList)
23      listCC = [0,0,0,0,0,0,0]
24      listC  = [0,0,0,0,0,0,0]
25      for j in range(len(DBList)):
26
27          listC[0] = InputList[2] #name
28          listC[1] = ["".join(list(map(lambda x: x.lower(),InputList[4])))] #
    nationality
29          listC[2] = ["".join(list(map(lambda x: x.lower(),InputList[5])))] #
    address
30          listC[3] = ["".join(list(map(lambda x: x.lower(),InputList[6])))] #
    id card number
31          listC[4] = converter(InputList[7]) #birthdate
32          listC[5] = ["".join(list(map(lambda x: x.lower(),InputList[8])))] #
    passport number
33          listC[6] = InputList[1] #type
34
35
36
37
38          listCC[0] = DBList[j][2]                                          #
    name
39          if len(DBList[j][4])>1:                                           #
    nationality
40              listCC[1] = (list(map(lambda x: x.lower(),DBList[j][4])))
41          else:
42              listCC[1] = "".join(list(map(lambda x: x.lower(),DBList[j][4]))
    )]
43
44          if len(DBList[j][5])>1:                                           #
    address
45              listCC[2] = (list(map(lambda x: x.lower(),DBList[j][5])))
46          else:
47              listCC[2] = "".join(list(map(lambda x: x.lower(),DBList[j][5]))
    )]
48
49          if len(DBList[j][6])>1:                                           #
    id card number
50              listCC[3] = (list(map(lambda x: x.lower(),DBList[j][6])))
51          else:
52              listCC[3] = "".join(list(map(lambda x: x.lower(),DBList[j][6]))
    )]
53
54          listCC[4] = converter(DBList[j][7])                               #
    birthdate
55
```

```
56          if len(DBList[j][8])>1:                                    #
    passport number
57              listCC[5] = (list(map(lambda x: x.lower(),DBList[j][8])))
58          else:
59              listCC[5] = "".join(list(map(lambda x: x.lower(),DBList[j][8]))
    )]
60
61          listCC[6] = DBList[j][1]                                   #
    type
62
63
64          if str(listCC[6]) == str(listC[6]):
65              finalList.append([round(resemblanceToData(listC,listCC),2),
    DBList[j]
66                                      [0], (DBList[j][2])])
67          if len(finalList) == 0:
68              return None
69      df = (pd.DataFrame((finalList), columns = column_set)).sort_values(by =
    "rate", ascending = False)
70      return df
71
72  print(np.array(csvCompare(inputList, DBList )))
73  with open("jsonComparaisonFile.json", "w", encoding='utf-8') as j:
74      json.dump(csvCompare(inputList, DBList ).to_json(orient = "records"),j,
    ensure_ascii=False, indent=4 )
75  #data = (csvCompare(inputList, DBList )).to_json("./jsonComparaisonFileV1",
    orient = "records")
76  print("---%s seconds ---" % (time.time() - start_time))
77  #s = input("")
```

Listing 3.8: Where real magic happens.

3.2.8 Results

The code has two outputs; a console output and a DTO(Json) Output which are as shown below.

```
In [3]: runfile('C:/Users/Mon PC/.spyder-py3/
resemblanceToDataUpdatedJsonVersion.py', wdir='C:/Users/Mon
PC/.spyder-py3')
[[90.9 '110901' 'AHMADI MOHAMMAD']
 [88.38 '111207' 'MOHAMMADI MOHAMMAD SHAFIQ']
 [87.53 '110577' 'HASSAN MOHAMMAD']
 ...
 [5.13 '6908537' 'YUSUF ABU UBAYDAH']
 [3.68 '6908621' 'Il Kyu Pak']
 [2.49 '6908699' 'Abubakar Abdifatah']]
---0.3697350025177002 seconds ---
```

Figure 3.2: Program Console output

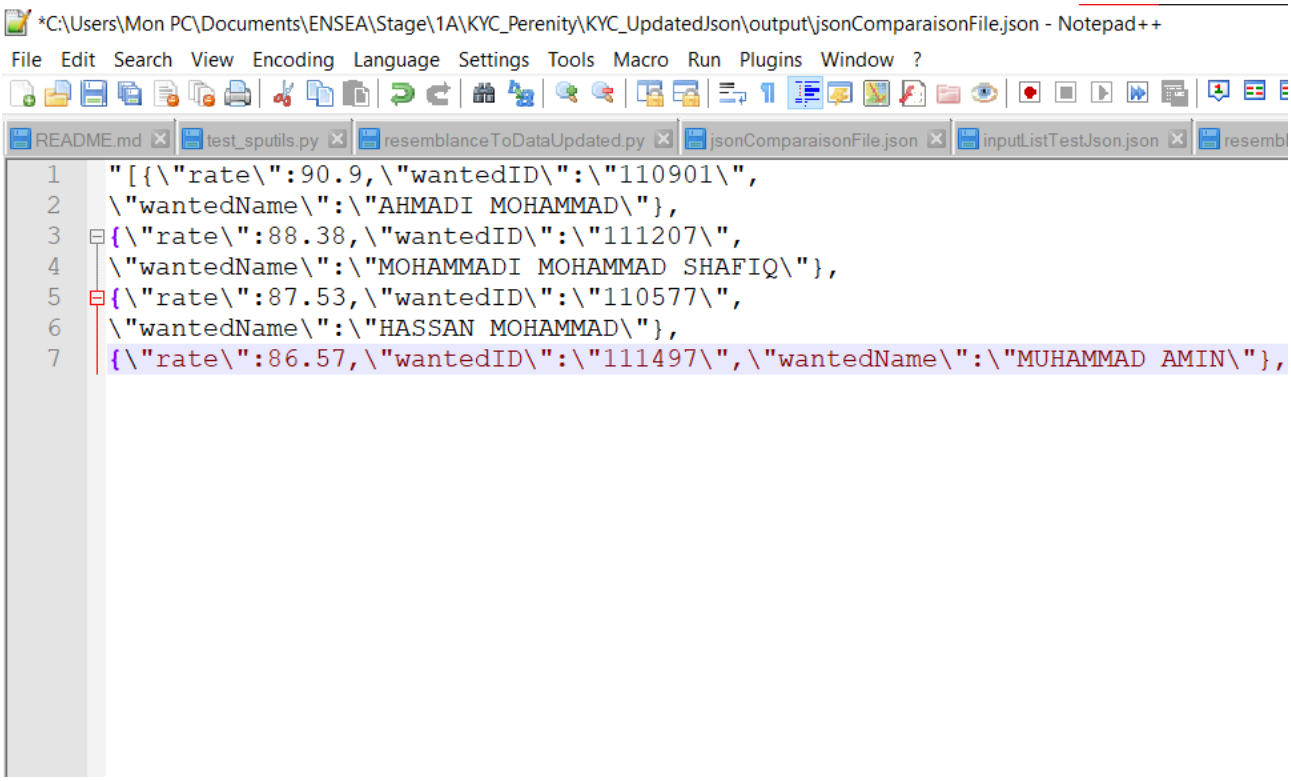


Figure 3.3: Program Json output

Chapter 4

Future Work

This chapter will be devoted entirely to discussing potential code enhancements. Typically, they will pertain to improved weight computation or faster processing time.

1. Since the weight calculation relies primarily on the quantity of viable names, a more extensive name database will be much welcomed. For instance, disposing of names databases for all ethnicities, including how they are dispersed throughout age and region, and storing all potential names'homonymes on that list are beneficial. Currently, storage is far less expensive than computing speed. Therefore, if we are able to retain and utilise these variables, the MonteCarlo simulation will be considerably more accurate in terms of how well it represents reality. Thus, an improved weight arrangement may be tailored to each individual based on a variety of characteristics.

2. A better way to parse data from watch lists can be used. This will rid us of the **remaker** function which is heavy in computation resources.

3. When rebuilding the database, it would be simple to create two lists, one for entities and one for persons. Therefore, if the evaluated profile is an individual, only the list of people will be processed. If the given profile is an entity, then only entities will be processed.

4. Given that passports and identification cards may be replicated in the modern day, a feature that tests if the identity document number is shared by several persons who are labeled as dangerous or not might assist detect if a customer is using a fake identity. Consider the testing of a wanted individual who is using a fraudulent passport. When evaluating his passport number, the application should check to see whether others have the same passport. If so, a notice should appear to alert the user of this anomaly.

5. Features such as the output format, rate list order, and manual weight setting are valued and straightforward to apply.

Chapter 5

Conclusion

Overall, this endeavour was engaging on several levels, whether professional or personal. The similarity calculus proved to be extremely tenacious and still in need of improvements, and the criteria on which it is based require more study to prevent the entry of unwanted profiles into the financial services. . Mastering KYC is in this case synonymous to reducing closed or freezed account, later on the basis of suspicious activity or dangerous funds use.
In addition, this internship was the initial step in my conversion to the quantitative finance profession. It is the field in which I intend to pursue my professional career. In quantitative domains, one must also perform data analysis, generate hypotheses, and confirm those hypotheses using simulations. During this one-month internship, I have improved my proficiency in Python programming, MonteCarlo simulation, data analysis, and statistical and probabilistic methodologies. For this reason alone, this encounter might be considered a success in many respects. The produced solution, which was deemed dependable and permitted to be offered as a package, was valued and represents a personal milestone.

Acknowledgment

This section will be devoted to the whole Perenity Software team, who were incredibly accommodating and inspiring to work with. They were technically informed and helpful. But most importantly, they were friendly and a pleasure to converse with. During my rather brief tenure there, I had the privilege of working with Mr. Toumani Sidibe, a skilled programmer who was also interested in a wide range of topics (Maths, Physics, Finance...).His passion to comprehend things and how they interact is incredibly motivating and quite similar to my own aspirations in life. Frequently, he would ask me things out of the blue, and we would debate ideas until we reached a specific conclusion. Once, out of the blue, he asked me, "Can you define entropy?" My response was essentially physical in nature. He explained that entropy is the amount of data that can be encoded in a tiny instance and that the more data stored in an instance, the more entropy it has, using data encryption as an example. Never had I tough that entropy's meaning could enclose such a wide notion . Mr. Florent was an additional great guy with whom I had the opportunity to collaborate. He is a quiet, good-natured individual. I would go so far as to say that he was too nice for his own good, given that he was a man of honor and integrity. I believe this is the one Creon had in mind when he stated that individuals with principles are destined to suffer the most. I do not mean it a rude or demeaning way. Then there was my tutor and director, Mr Tarik Bouramdan who did not lack in any way when it came to leadership skills. This prompts his teams to be productive and respond effectively to the needed specifications. I had not had the chance to spend that much time with him since he is managing over 30 persons at the same time, but just from observing him(his desk was in front of mine) and the rare exchanges I had with him, he gave off the impression of being quite capable and deserving of his position. Working with him had changed how I viewed a "director" and cleared my mind as to who I want to be in my career: demanding yet appreciated.

I hereby would like to offer my heartfelt appreciation to each and every person who has contributed to this event. There are several individuals whose names I have omitted, but they all contributed to the creation of a pleasant working atmosphere and, most importantly, an engaging experience. I also would like to thanks **ENSEA** for offering me the opportunity to develop my knowledge and broaden my vision.Not only did it make me a better engineer, but also a better human throughout the diverse activities, projects and events it has offered me access to. As such, I also thank all the school staff for such a character-building first year in a foreign country.

References

none