Privacy

Masters Report

Thomas Zamojski

Anurag Miglani

Professor: Tristan Allard

tristan.allard@irisa.fr

Abstract

The ability of a person or a group to seclude themselves, or information about themselves, which could describe them selectively. Measures and the effects of privacy are rather subjective rather than being descriptive. For many, it hold a different definition in different contexts. In this course we will discuss the approaches to seclude information from being selective to generic.

You can’t have 100% security and 100% privacy, and also zero inconvenience.
Barack Obama

Introduction

We have used a python notebook for the creation of this project. How to use python (Anaconda)? We will answer this. After. We go to the implementation of the algorithms. I hope you will like what we have done. So just sit back and let's do some Privacy.

Implementation

Code can be downloaded from the shared folder:

Step1: *Download and install Anaconda for Python. Anaconda in nothing but a more precise system which comes along with many packages per-installed for python such that we don't have to do it manually.*

Step2: *Once you have installed Python, download the source code and copy the code in a new directory.*

Step3: *After-that go to the directory in which you have copied the code. Open the terminal from that directory and give the command. "jupyter notebook".*

Step4: *A new pop-up should open in the web-browser which have basically opened the directory in the browser and in there you will see the files of our code.*

Step5: *Click on the files to have a look at them, later we provide a full description of what it is doing.*

- Mondrian.ipynb
- Differential.ipynb

Mondrian

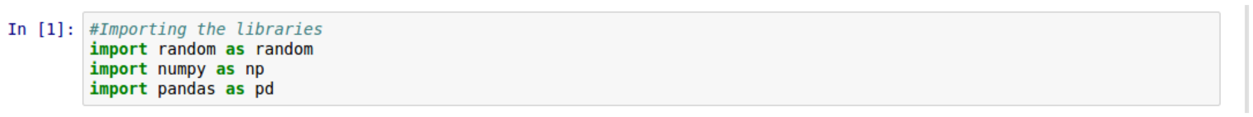
Mondrian was a contributor to the *De Stijl* art movement and group, which was founded by Theo van Doesburg. He evolved a non-representational form which he termed neoplasticism. This consisted of white ground, upon which he painted a grid of vertical and horizontal black lines and the three primary colors. From there the word developed Mondrian. Now Mondrian is used for Privacy discrepancy. Let's see what it does.

There are many variants of Mondrian these days, we will implement the one which was given in the class.

Algorithm Designed:

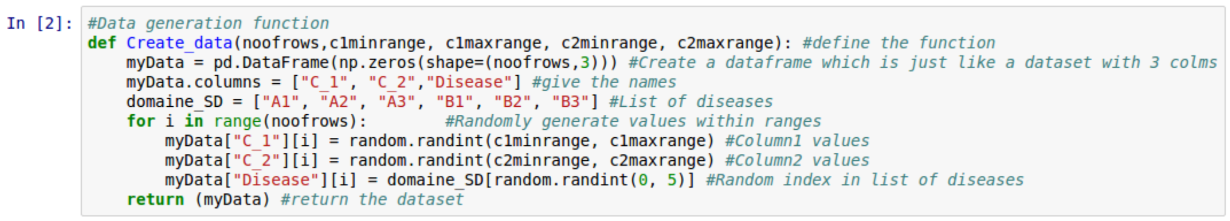
When you click on the file Mondrian “*Mondrian.ipynb*” you will be taken to the next browser having such a window: Each and every cell is well commented.

Importing the libraries:



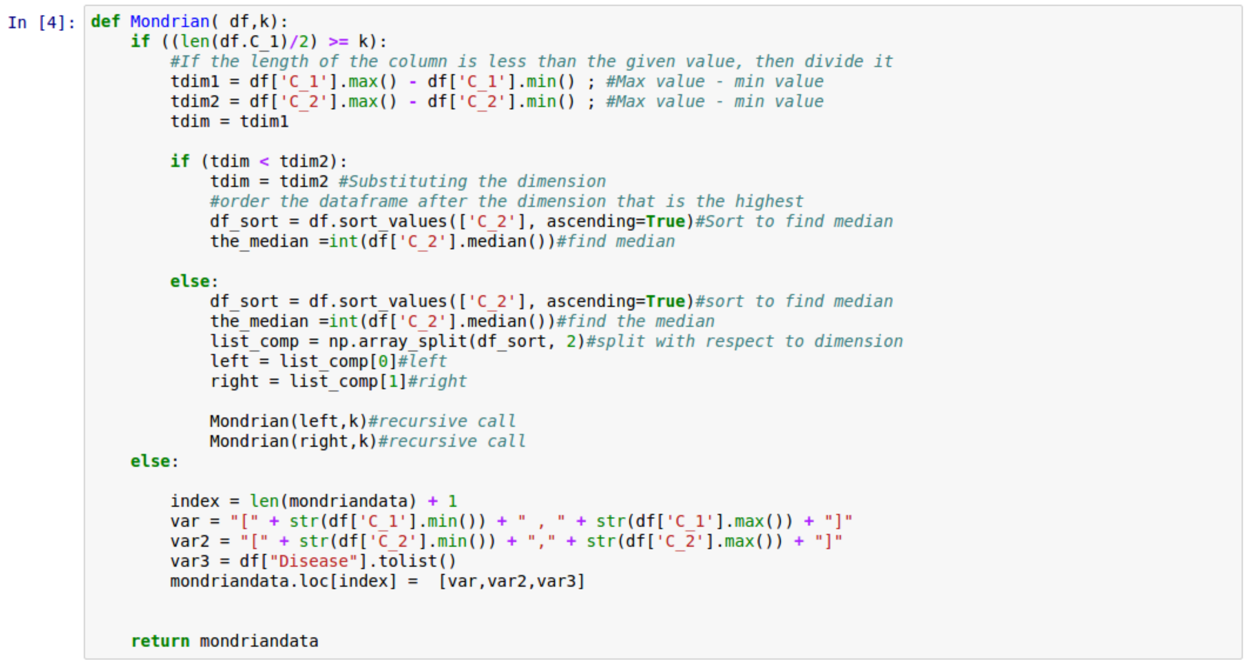
```
In [1]: #Importing the libraries
        import random as random
        import numpy as np
        import pandas as pd
```

Data Generation:



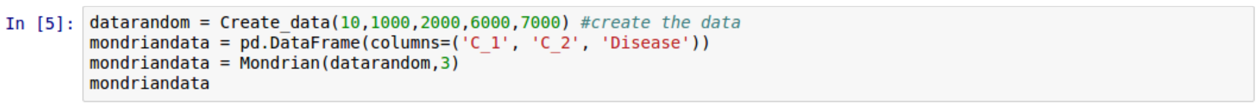
```
In [2]: #Data generation function
        def Create_data(noofrows,c1minrange, c1maxrange, c2minrange, c2maxrange): #define the function
            myData = pd.DataFrame(np.zeros(shape=(noofrows,3))) #Create a dataframe which is just like a dataset with 3 colms
            myData.columns = ["C_1", "C_2","Disease"] #give the names
            domaine_SD = ["A1", "A2", "A3", "B1", "B2", "B3"] #List of diseases
            for i in range(noofrows):        #Randomly generate values within ranges
                myData["C_1"][i] = random.randint(c1minrange, c1maxrange) #Column1 values
                myData["C_2"][i] = random.randint(c2minrange, c2maxrange) #Column2 values
                myData["Disease"][i] = domaine_SD[random.randint(0, 5)] #Random index in list of diseases
            return (myData) #return the dataset
```

Algorithm Defined:



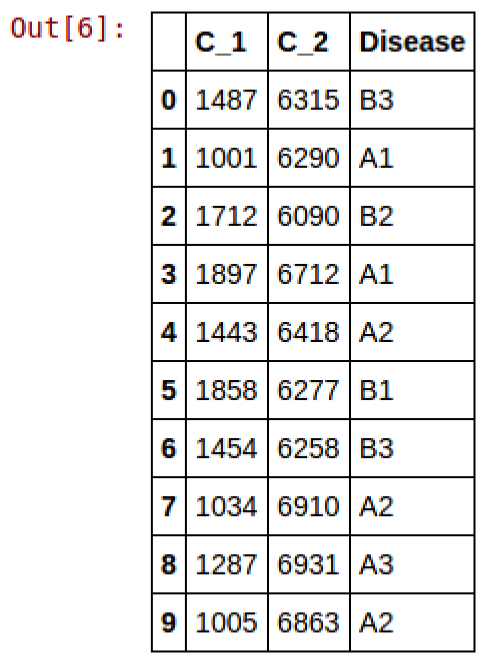
```
In [4]: def Mondrian( df,k):
            if ((len(df.C_1)/2) >= k):
                #If the length of the column is less than the given value, then divide it
                tdim1 = df['C_1'].max() - df['C_1'].min() ; #Max value - min value
                tdim2 = df['C_2'].max() - df['C_2'].min() ; #Max value - min value
                tdim = tdim1

                if (tdim < tdim2):
                    tdim = tdim2 #Substituting the dimension
                    #order the dataframe after the dimension that is the highest
                    df_sort = df.sort_values(['C_2'], ascending=True)#Sort to find median
                    the_median =int(df['C_2'].median())#find median

                else:
                    df_sort = df.sort_values(['C_2'], ascending=True)#sort to find median
                    the_median =int(df['C_2'].median())#find the median
                    list_comp = np.array_split(df_sort, 2)#split with respect to dimension
                    left = list_comp[0]#left
                    right = list_comp[1]#right

                    Mondrian(left,k)#recursive call
                    Mondrian(right,k)#recursive call
            else:

                index = len(mondriandata) + 1
                var = "[" + str(df['C_1'].min()) + " , " + str(df['C_1'].max()) + "]"
                var2 = "[" + str(df['C_2'].min()) + "," + str(df['C_2'].max()) + "]"
                var3 = df["Disease"].tolist()
                mondriandata.loc[index] =  [var,var2,var3]


            return mondriandata
```

Calling the Algorithm:



```
In [5]: datarandom = Create_data(10,1000,2000,6000,7000) #create the data
        mondriandata = pd.DataFrame(columns=('C_1', 'C_2', 'Disease'))
        mondriandata = Mondrian(datarandom,3)
        mondriandata
```

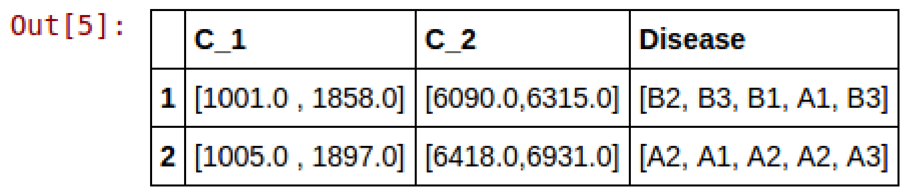
Input:



Out[6]:

	C_1	C_2	Disease
0	1487	6315	B3
1	1001	6290	A1
2	1712	6090	B2
3	1897	6712	A1
4	1443	6418	A2
5	1858	6277	B1
6	1454	6258	B3
7	1034	6910	A2
8	1287	6931	A3
9	1005	6863	A2

Output:



Out[5]:

	C_1	C_2	Disease
1	[1001.0 , 1858.0]	[6090.0,6315.0]	[B2, B3, B1, A1, B3]
2	[1005.0 , 1897.0]	[6418.0,6931.0]	[A2, A1, A2, A2, A3]

Differential Privacy

Given a database containing confidential information, we would like to allow learning of statistical information about the contents of database without violating the privacy of any of its individual entries. The traditional notion of privacy is not suitable, because it only allows negligible information to be revealed from the database. Therefore a new notion of privacy is needed to allow a better trade-off between privacy and utility.

Setting: We have the following setting.

- A sensitive database $x \in X$, where X is the space of databases
- A discrete distance function between databases $\Delta : X \times X \rightarrow N = \{0, 1, \cdots\}$
- Neighboring databases: $x, x' \in X$ s.t. $\Delta(x, x') = 1$

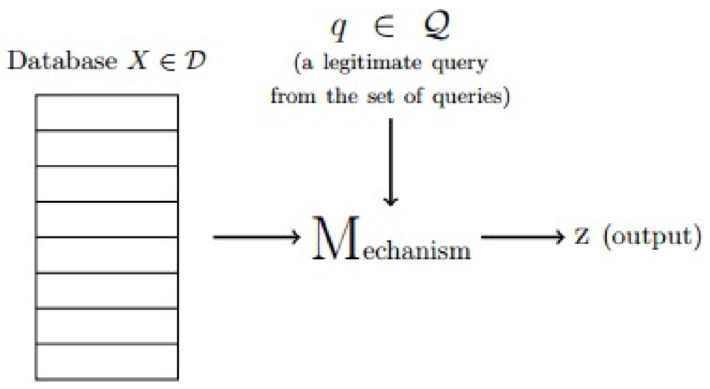

$q \in Q$
(a legitimate query from the set of queries)
Database $X \in \mathcal{D}$
Mechanism
z (output)

Figure 1

The goal is to answer the query q(X), while conserving the privacy. In essence we wish to approximate the true answer q(X) with z, without revealing too much information. Q is defined the set of query functions,

$$Q = \{q : X \to Z\}.$$

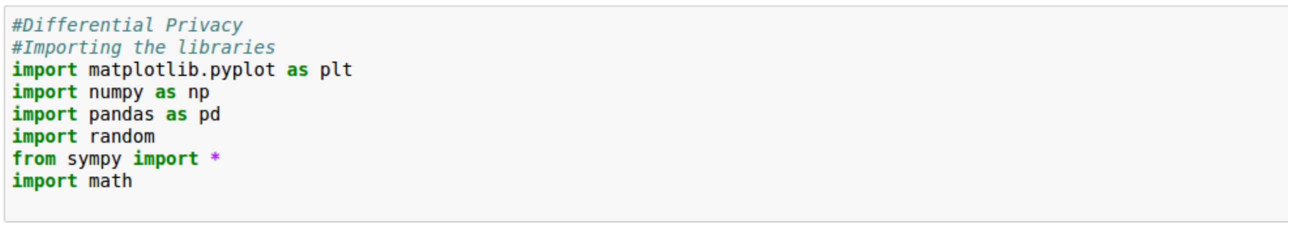
To achieve the goal of privacy, the mechanism, M, should not be deterministic, and it should be randomized. Consequently, let R be the source of randomness with a sample r. Then the output of the query, generated by the mechanism, depends on the randomness source in the mechanism.

$$z = M(X, q; r), \text{ or equivalently } z \leftarrow M(X, q).$$

$$f(x|\mu, b) = \frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right)$$

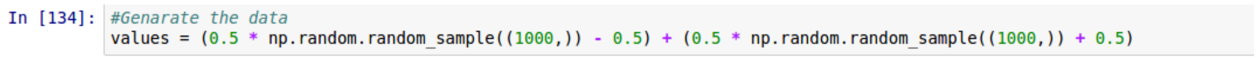
Then, we need to add some noise for the same. Lets see with the help of python, how can we add noise to the data.

First import libraries.



```
#Differential Privacy
#Importing the libraries
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import random
from sympy import *
import math
```

Then generate the data



```
In [134]: #Genarate the data
          values = (0.5 * np.random.random_sample((1000,)) - 0.5) + (0.5 * np.random.random_sample((1000,)) + 0.5)
```

Just as an example try to generate some noise

```
In [140]: ##Generating the noise
          loc = 0.0001
          scale = 1
          s = np.random.laplace(loc, scale, 1000)
```

Then, trying to add noise to the generated data

```
In [160]: ##Mix it with the data
          for i in range(1000):
              values[i] = values[i] + np.random.laplace(1,0.001)


          epsilon = 0.0001
          n = 1000
          bw = np.random.random_sample(0)
          #To generate the perbutations
          avg = 0.0
          power = 0.0
          perturb = 0
          for i in range(6):
              power = math.pow(10.0, i)
              average = 0.0
              perturb = 0
              while(perturb < power):
                  avg += n + np.random.laplace(1,epsilon)
                  avg /= power
                  perturb = perturb + 1


          ##After that some of the graphs watching how is it going
```

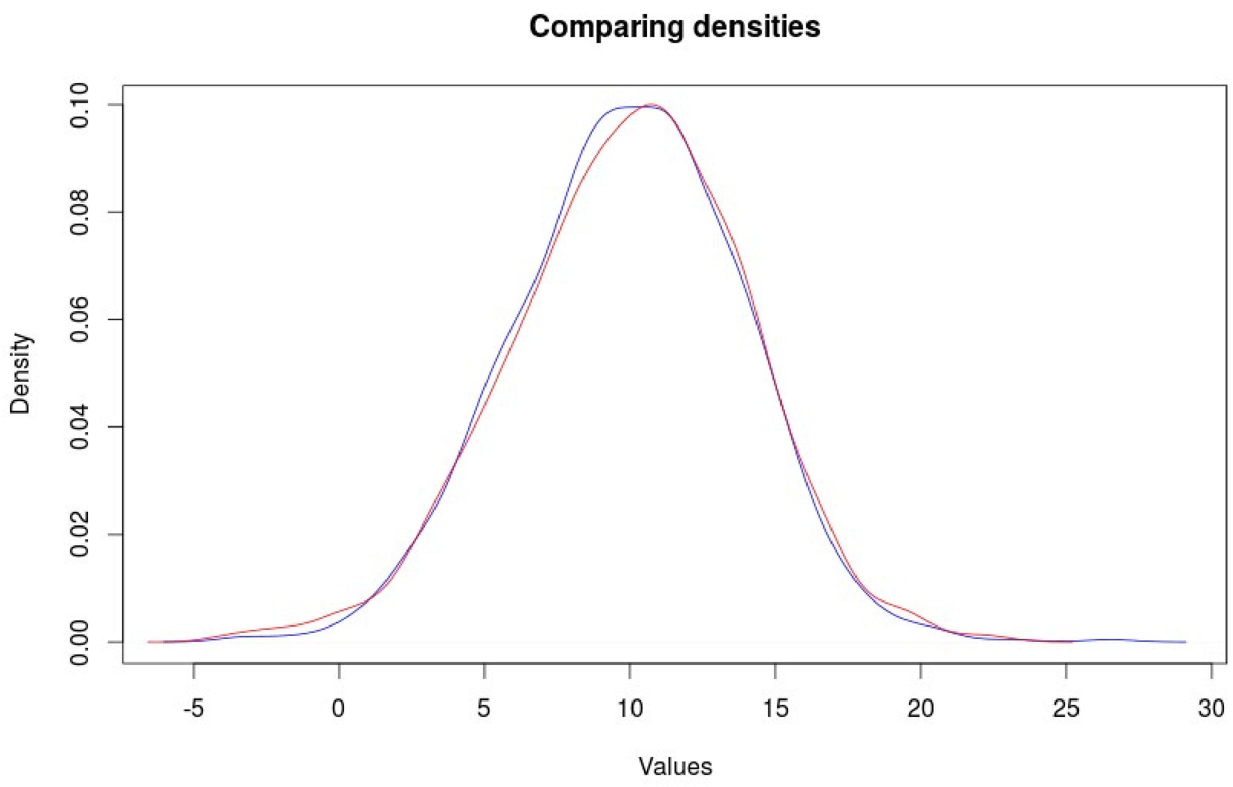
Again, lets see the different values and some graphs to compare the densities

```
In [155]: avg
Out[155]: 0.010010099332450456

In [157]: perturb
Out[157]: 100000

In [161]: power
Out[161]: 100000.0

In [162]: values ##As you could see that the addition of noise and brought them up to be quite generic
```

```
       1.64998129,  1.71384897,  1.5953082 ,  1.2408566 ,  1.67258159,
       1.52789498,  1.3552361 ,  1.49649815,  1.61213102,  1.49134277,
       1.83508875,  1.785479  ,  1.56835576,  1.63133209,  1.60733975,
       1.46753527,  1.68476607,  1.45514534,  1.41898776,  1.39191132,
       1.33400948,  1.53464612,  1.1273924 ,  1.48377565,  1.22118905,
       1.35096474,  1.73041736,  1.56649157,  1.6355702 ,  1.30766552,
       1.39038933,  1.088478  ,  1.44402082,  1.28429913,  1.62596441,
       1.32339754,  1.39334491,  1.21431455,  1.58351925,  1.18031872,
       1.78855554,  1.53173882,  1.38469854,  1.60392858,  1.45850657,
       1.26665728,  1.67469279,  1.36836766,  1.63217778,  1.78114067,
       1.2728166 ,  1.73481598,  1.47486474,  1.52086299,  1.52069613,
       1.48206311,  1.62954777,  1.40511806,  1.67060882,  1.40003914,
       1.4588883 ,  1.54995998,  1.12626126,  1.54543084,  1.45201637,
       1.73129195,  1.73672433,  1.57986617,  1.11652814,  1.27461827,
       1.52937078,  1.83910481,  1.38219512,  1.52782882,  1.63834276,
       1.19962744,  1.09299267,  1.54535236,  1.72498263,  1.88397376,
       1.45109371,  1.95763967,  1.1215812 ,  1.36701445,  1.59172602,
       1.17009398,  1.23957164,  1.85214356,  1.7656455 ,  1.67211702,
       1.51948196,  1.25021396,  1.76184003,  1.60123966,  1.65852387,
       1.10553701,  1.99136305,  1.50163978,  1.50048878,  1.48320968,
```

```
In [ ]: ##Lets compare some of the densities using R studio
```

Comparing the densities.



And this we can say that the model is good.

Conclusion

We have discussed two classes of algorithms used in *Privacy.* Mondrian although apart from being simple yet have some discrepancies which are later resolved from the differential privacy, we could see that adding some noise to the data could simply results in terms of more generic data.