Alex Miller
Assignment 7 Writeup
Collaborators: AC Fields (acfields), Ryan Li (rli3)

Problem 1.1

32559 Individuals could be uniquely identified
Only 2 could not be uniquely identified

Problem 1.2

My solution has (n^2) complexity. I determined this by considering how the function 'groupby' works. It iterates through the date frame and pulls rows that match the parameters given and groups them together. In the worse case, when there are no groupable rows, the function, for a data frame of n rows, has to iterate through n rows, then n-1 rows, then n-2 rows, and so on until it reaches the n^{th} row, at which point it iterates through n-n rows. It, therefore, iterates through ~$n^2/2$ rows, giving it n^2 complexity.

Problem 1.3

num_bachelors returned that there are 5355 individuals with exactly a bachelor's level of education in the *adult* dataset.
The sensitivity of the num_bachelors query is 1; the biggest possible change of adding one more row of an individual with a bachelor's level education to the dataset is 1.

Problem 1.4

The more overlap there is between two histograms, the more privacy there is for individuals; this is because more overlap between queries implies that two queries are more indistinguishable.

Problem 1.5

The area of the error plot decreases as epsilon decreases; this implies that, as the privacy of individuals in the dataset increases, the error bounds of the dataset increases. When trying to choose an epsilon, I would weigh the privacy I want and the usability I need in order to come to a solution.

Problem 2

The attack in the paper violates Differential Privacy because the probability that any given answer is the result of some input is greater than the probability that is the result of a neighboring input.
My attack ran as followed:

1. I computed a random number x as well as a counterpart x' = exp(log(x)). I then took note of how many bits they differed by. I repeated this for a reasonable number of trials and concluded that they differed by ~6 bits

2. Using this difference in bits, I saw if I could, given a y and scale, re-compute the x value in y = (2b-1)*scale*log(x). In order to do this, I put in place the following procedure:
    a. Given y, I sent y to a negative value and I computed x' = exp(y/scale)
    b. Taking note of the difference in bits, I set the last six bits of x' to 0
    c. I then incremented the integer representation of x' (x_int) by one until I found one such that the float interpretation of x_int (x_float), when used to calculate a value y_test = scale*log(x_float), resulted in y_test == y.
    d. If this was achieved, it means the x_float was a valid originator of the value of y generated by the my_laplace function. If this was not achieved for a given y, that meant that the given y couldn't have been generated solely by my_laplace.
    e. I then took this value and modded it by 2^{-53} in order to ascertain if that x was generated by the numpy random number generator (it only generates floats in multiples of 2^{-53}). If the result was 0, the x_float was a valid result of simply using my_laplace without a shift in value; y was the result of my_laplace. If the result was anything else, this implies that y was not the result of solely using my_leplace; it was the result of 1.0 + my_laplace.

In the case that my implementation was checking y values generated by my_laplace, it generated 858 true positives out of 1000 trials.
In the case that my implementation was checking y values generated by 1.0 + my_laplace, it generated 308 false positives out of 1000 trials.

Sources

https://docs.python.org/3/library/

https://unix.stackexchange.com/questions/48535/can-grep-return-true-false-or-are-there-alternative-methods

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.count.html#pandas.Series.count

https://math.stackexchange.com/questions/556807/what-is-the-sum-of-n-1n-2-n-k

https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.random.laplace.html

https://en.wikipedia.org/wiki/Double-precision_floating-point_format#IEEE_754_double-precision_binary_floating-point_format:_binary64

https://pynative.com/python-range-for-float-numbers/