《数据安全》实验报告

姓名： 费泽锟 学号： 2014074 班级： 信安班

实验名称：

交互式发布 DP 方案评估

实验要求：

参考教材实验 5.1，对交互式发布方案进行 DP 方案设计，指定隐私预算为 0.1，支持查询次数为 20 次，对 DP 发布后的结果进行评估说明隐私保护的效果。

实验过程：

1. 配置实验环境

虽然我们已经配置过了 Ubuntu 18.04 虚拟机以及基本的环境，但是为了防止实验过程之中出现一些奇怪的环境问题，我们还是跟着实验视频的指导进行实验环境的配置与检查。

使用如下命令安装 gcc 以及对应的依赖库：

sudo apt-get install build-essential

```
[sudo] password for zzekun:
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version (12.4ubuntu1).
The following packages were automatically installed and are no longer required:
  fonts-liberation2 fonts-opensymbol gir1.2-gst-plugins-base-1.0
  gir1.2-gstreamer-1.0 gir1.2-gudev-1.0 gir1.2-udisks-2.0
  grilo-plugins-0.3-base gstreamer1.0-gtk3 libboost-date-time1.65.1
  libboost-filesystem1.65.1 libboost-iostreams1.65.1 libboost-locale1.65.1
  libcdr-0.1-1 libclucene-contribs1v5 libclucene-core1v5 libcmis-0.5-5v5
  libcolamd2 libdazzle-1.0-0 libe-book-0.1-1 libedataserverui-1.2-2 libeot0
  libepubgen-0.1-1 libetonyek-0.1-1 libevent-2.1-6 libexiv2-14
  libfreerdp-client2-2 libfreerdp2-2 libgc1c2 libgee-0.8-2 libgexiv2-2
  libgom-1.0-0 libgpgmepp6 libgpod-common libgpod4 liblangtag-common
  liblangtag1 liblirc-client0 liblua5.3-0 libmediaart-2.0-0 libmspub-0.1-1
  libodfgen-0.1-1 libqqwing2v5 libraw16 librevenge-0.0-0 libsgutils2-2
  libssh-4 libsuitesparseconfig5 libvncclient1 libwinpr2-2 libxapian30
  libxmlsec1-nss linux-hwe-5.4-headers-5.4.0-84 lp-solve media-player-info
  python3-mako python3-markupsafe syslinux syslinux-common syslinux-legacy
  usb-creator-common
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 98 not upgraded.
```

可见我们之前已经成功地完成了这一步。

我们使用 gcc -v 命令来查看我们的编译器是否已经配置成功：

```
zzekun@ubuntu:~/Differential-Privacy$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/7/lto-wrapper
OFFLOAD_TARGET_NAMES=nvptx-none
OFFLOAD_TARGET_DEFAULT=1
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 7.5.0-3ubuntu1~18
.04' --with-bugurl=file:///usr/share/doc/gcc-7/README.Bugs --enable-languages=c,
ada,c++,go,brig,d,fortran,objc,obj-c++ --prefix=/usr --with-gcc-major-version-on
ly --program-suffix=-7 --program-prefix=x86_64-linux-gnu- --enable-shared --enab
le-linker-build-id --libexecdir=/usr/lib --without-included-gettext --enable-thr
eads=posix --libdir=/usr/lib --enable-nls --enable-bootstrap --enable-clocale=gn
u --enable-libstdcxx-debug --enable-libstdcxx-time=yes --with-default-libstdcxx-
abi=new --enable-gnu-unique-object --disable-vtable-verify --enable-libmpx --ena
ble-plugin --enable-default-pie --with-system-zlib --with-target-system-zlib --e
nable-objc-gc=auto --enable-multiarch --disable-werror --with-arch-32=i686 --wit
h-abi=m64 --with-multilib-list=m32,m64,mx32 --enable-multilib --with-tune=generi
c --enable-offload-targets=nvptx-none --without-cuda-driver --enable-checking=re
lease --build=x86_64-linux-gnu --host=x86_64-linux-gnu --target=x86_64-linux-gnu
Thread model: posix
gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)
```

通过上图我们能够确认，我们的编译器环境已经配置成功了。

在准备进行实验的文件夹内打开终端，解压提供的 test.tar.gz 文件。我们在命令行中输入命令：tar -xzvf ./test.tar.gz，就可以成功完成所有的实验准备了。

~~~~~~~

2. 复现非交互式发布 DP 方案

首先，我们先使用 make 命令来完成对我们的 project 的编译：

```
zzekun@ubuntu:~/Differential-Privacy/experiment1$ make
gcc -I./include -c csvpackage.c
gcc -I./include -c laplace.c -lm
gcc -I./include -c testraw.c
gcc csvpackage.o laplace.o testraw.o -o testraw  -lm
gcc -I./include -c testhist.c
gcc csvpackage.o laplace.o testhist.o -o testhist -lm
zzekun@ubuntu:~/Differential-Privacy/experiment1$
```

我们可以看到对应的执行命令，我们在对应的工作目录下也能够成功看到能够运行的可执行程序了。我们先不这样直接运行可执行程序，我们来进行分析，看看源代码到底做了些什么事儿。

我们来看看拉普拉斯噪声到底是如何生成的，具体的源代码如下图所示：
~~~~~~~

```c
/*
函数功能：  求解laplace分布概率累积的反函数，利用该反函数产生laplace分布的随机数
输入参数说明：
beta        拉普拉斯分布参数
seed        长整型指针变量，  *seed 为伪随机数的种子
*/
double laplace_data(double beta, long int * seed)
{
    double u1,u2, x;
    u1 = uniform_data(0.0, 1.0, seed);
    u2 = uniform_data(0.0, 1.0, seed);
    if (u1 < 0.5)
    {
        x = beta * (log(2*u1)+u2);
    }
    else
    {
        x = u2 - (beta * log(2*(1-u1)));
    }

    return x;
}
```

其中的 u1，u2 等数均为生成的使用混合同余的方法生成的(a, b)区间上均匀分布的随机数，具体的原理可见第五章的实验讲解，然后我们通过如下公式来基于随机数种子以及均匀分布的随机数来生成拉普拉斯噪声：

$$x = \begin{cases} \beta \ln(2u_1) + u_2 & u_1 \leqslant 0.5 \\ u_2 - \beta \ln\left(2(1-u_1)\right) & u_1 > 0.5 \end{cases}$$

我们能够清晰地看到 laplace_data 函数实现的正是上述公式中的内容。因为我们不需要复现直方图发布的部分，也就是说我们不需要考虑对数据进行分组分桶后的加噪情况，所以我们这里仅仅复现非交互式发布的 DP 方案，我们进一步来分析 testraw.c 文件中的内容。

```c
void csv_analysis(char* path, double beta, long int seed)
{
    FILE *original_file = fopen(path,"r+"); //读取指定路径的数据集
    struct Animals * original_data = NULL;
    original_data = csv_parser(original_file);
    int sum=0,i=0;
    double x = 0;
    while(original_data[i].name)  //循环为原始数据集内各条数据去除标识（动物名）、生成拉普拉斯噪音并加噪
    {
        x = laplace_data(beta,&seed); //产生拉普拉斯随机数
        printf("Added noise:%f\t%s %d\t%f\n",x,"Animal",i+1,original_data[i].carrots+x); //此处分别列出了每条具体添加的
        if(original_data[i].carrots+x>=55)
        {
            sum++;
        }
        i++;
    }
    printf("Animals which carrots cost > 55 (Under DP): %d\n",sum); //输出加噪后的数据集中，每日食用胡萝卜大于55的动物个数
}
```

在 testraw.c 文件之中最重要的就是 csv_analysis 函数了，该函数会读取 csv 文件并将其中的数据转化为 struct 结构化的数据，接着我们对动物名这一列进行遍历，注意，这里我们使用的非交互式的方案，所以我们会直接对数据集进行操作，得到一个加入噪

声之后的数据集也可以说是净化数据集，在加入噪声之后如果投喂的胡萝卜数大于 55 则进行计数。

```c
int main()
{
    long int seed;
    int sen = 1;  //对于一个单属性的数据集，其敏感度为1
    double beta;
    srand((unsigned)time( NULL )); //生成基于时间的随机种子（srand方法）
    beta = 0;
    printf("Please input laplace epsilon:");
    scanf("%lf", &beta);
    if(beta<=0 || !beta)//当输入的beta值无效时，默认设定beta值为1
    {
        beta = 1.0;
    }
    printf("Under privacy budget %f, sanitized original data with fake animal name and laplace noise:\n",beta);
    beta = sen / beta; //拉普拉斯机制下，实际公式的算子beta为敏感度/预算
    seed = rand()%10000+10000; //随机种子产生
    csv_analysis("./zoo.csv",beta,seed); //先调用原始数据集
    printf("==================Using neighbour dataset==================\n");
    seed = rand()%10000+10000; //随机种子更新
    csv_analysis("./zoo_nb.csv",beta,seed); //再调用相邻数据集
    return 0;
}
```

我们接着来看看主函数中的内容，首先我们就需要一个随机数种子，这是不可或缺的，因为我们的数据集之中只需要比对一个属性列即可，所以这里我们使用一维的拉普拉斯全局敏感度就行，也就是说这里我们的全局敏感度为 1。

在主函数之中还加入了对无效输入的检查，增强了程序的鲁棒性。我们需要注意的是，在生成拉普拉斯噪声时，需要的参数 b，当参数 b 越大的时候对应的生成更大的随机噪声的概率就越大。我们这里的参数 b 与隐私预算的关系是倒数关系，所以需要进行一步转换操作。

因为是非交互式 DP 发布方案，我们只需要得到原数据集进行加噪后的查询语句的结果和只差一条数据的相邻数据集加噪后的查询结果，然后将查询结果对比就能够看出我们差分隐私策略是否有效了，我们已经了解了 DP 的具体原理，我们来实践一下可执行程序的运行效果。

```
zzekun@ubuntu:~/Differential-Privacy/experiment1$ ./testraw
Please input laplace epsilon:10
Under privacy budget 10.000000, sanitized original data with fake animal name and laplace noise:
Animals which carrots cost > 55 (original): 90
Added noise:0.330554    Animal 1        1.330554
Added noise:0.621333    Animal 2        88.621333
Added noise:0.835958    Animal 3        35.835958
Added noise:0.696807    Animal 4        99.696807
Added noise:-0.333855   Animal 5        68.666145
Added noise:0.239808    Animal 6        14.239808
```

在原数据集上得到的统计原始结果为 90，我们来看看加噪后的数据，因为我们选择的隐私预算为 10，隐私预算越大加噪的效果越好，我们可以将加入噪声的大小与隐私预算为 1 时的情况进行比较。

```
Added noise:0.567642    Animal 180      55.567642
Added noise:0.754931    Animal 181      60.754931
Added noise:-0.005748   Animal 182      6.994252
Animals which carrots cost > 55 (Under DP): 90
```

在原数据集上在添加过噪声后的数据集上进行统计得到的结果仍然为 90，数据的可用性好，因为这种情况大部分的噪声在[-1, 1]区间，那么只有在 55 附近的数据才大概率有可能会被噪声影响，从而影响统计结果。

```
==================Using neighbour dataset==================
Animals which carrots cost > 55 (original): 89
Added noise:0.377327    Animal 1        1.377327
Added noise:0.002789    Animal 2        88.002789
Added noise:0.486052    Animal 3        35.486052
Added noise:0.428282    Animal 4        99.428282
Added noise:0.274579    Animal 5        69.274579
Added noise:0.000891    Animal 6        14.000891
Added noise:0.000608    Animal 7        77.000608
Added noise:1.215431    Animal 8        54.215431
```

上图展示的是在仅差一条数据的相邻数据集上的统计结果，我们看到最终的统计结果为 89，也就是删除那条数据大于 55，这个时候我们来看看加噪后的效果。

```
Added noise:0.874574    Animal 177      7.874574
Added noise:-0.118356   Animal 178      41.881644
Added noise:0.008315    Animal 179      55.008315
Added noise:0.397842    Animal 180      60.397842
Added noise:0.137524    Animal 181      7.137524
Animals which carrots cost > 55 (Under DP): 89
zzekun@ubuntu:~/Differential-Privacy/experiment1$
```

我们可以看到加入噪声后的统计结果仍然为 89，也就是说如果面对这条查询语句的差分攻击，10 的隐私预算是满足不了我们的安全性要求的。我们来试试隐私预算为 1 的时候，能够得到怎样的结果。

这里就不给出俩数据集的原来的统计结果了，我们直接来看加噪后的统计结果：

```
Added noise:0.973318    Animal 179      42.973318
Added noise:3.717662    Animal 180      58.717662
Added noise:0.774347    Animal 181      60.774347
Added noise:0.740910    Animal 182      7.740910
Animals which carrots cost > 55 (Under DP): 89
==================Using neighbour dataset==================
```

在原数据集之中，经过隐私预算为 1 的拉普拉斯噪声的加入，我们最终得到的统计结果为 89，并且我们能够对比对应噪声的大小，可以看到隐私预算为 1 的噪声很明显的比隐私预算为 10 的噪声大。

```
Added noise:-2.307784   Animal 177      4.692216
Added noise:-0.006617   Animal 178      41.993383
Added noise:1.294693    Animal 179      56.294693
Added noise:-1.074592   Animal 180      58.925408
Added noise:-0.128672   Animal 181      6.871328
Animals which carrots cost > 55 (Under DP): 89
```

在相邻数据集之中，最终的统计结果没有变，仍然为 89，但是就算是这样我们也达到了抵御差分攻击的功能，攻击者如果在不知道加入噪声的情况下，会认为相差的那

条数据小于 55，提供给了他一个错误的结果，这也说明了隐私预算越小数据保护的效果越好，但是数据可用性越差。

~~~~~~~

3. 交互式发布 DP 方案模拟实现

在这里我们需要进行多轮的查询进行实验，显然在 csvparser 中的输出的信息就会使我们的输出信息看起来十分的冗余，所以我们这里把 csvparser 中的 printf 语句都给注释掉。

我们来看看在交互式发布 DP 方案之中需要注意的点，最重要的一点就是在差分隐私之中分为串行组合性与并行组合性，在非交互式发布 DP 方案之中，我们可以理解为并行组合性，也就是说我们只需要将数据集加噪后，进行的查询不会造成每一次查询的隐私损失。

但是我们在交互式发布 DP 方案之中，需要注意的就是每一次查询其实都会造成对应的隐私损失，这里给出每次隐私损失的份额。隐私预算ε刻画了一类查询任务的总体允许的隐私泄露程度。如果仅仅将ε作为生成拉普拉斯噪音的参数的话，很容易受到重复推测攻击，在我们的实验之中首先会实现模拟交互式发布 DP 方案，并且在设定的隐私预算下进行重复推测攻击。

ε	噪声绝对值的数据分布				多次查询添加噪声的平均值落在危险区间内的概率			
	90%	95%	99%	99.9%	100	1000	10000	10000
1	2.29	2.98	4.50	6.43	100.00	100.00	100.00	100.00
0.1	23.24	29.99	45.51	66.56	25.59	73.75	99.99	100.00
0.01	227.97	296.22	463.48	677.26	2.72	9.12	27.85	73.70

我们从教材的学习之中，能够得到对应的理论推导值，也就是多次查询添加噪声的平均值落在危险区间的概率，我们可以以此为根据来实现我们的重复推测攻击。

要考虑保护多次查询的话，需要为每次查询进行预算分配：假定隐私预算为ε，允许的查询次数为 k，则每次查询分配的预算为$\frac{\varepsilon}{k}$，这样才能达到ε-差分隐私的目标，也就是说其实我们在交互式发布的过程之中，其实每次查询的结果加入的噪声都较大，这一点我们将会在接下来的实验之中看到。

因此，对于统计查询而言，如果在查询结果上进行反馈，则需要定义所能支持的次数，
~~~~~~~

进而按上述方式对每次查询进行预算的分配。换句话说，这种添加噪音的方式，会使得每次查询都会消耗一定的隐私预算，直到隐私预算都被消耗干净，就再也不能起到保护的作用。

我们来完成我们的交互式发布 DP 方案，首先我们的源码可见同目录附件，我们仅给出关键部分的解析。

```c
double csv_analysis(char* path, double beta, long int seed)
{
    FILE *original_file = fopen(path,"r+"); //读取指定路径的数据集
    struct Animals * original_data = NULL;
    original_data = csv_parser(original_file);
    int sum=0,i=0;
    double x = 0;
    while(original_data[i].name)  //循环为原始数据集内各条数据去除标识（动物名）、生成拉普拉斯噪音并加噪
    {
        if(original_data[i].carrots+x>=55)
        {
            sum++;
        }
        i++;
    }
    // printf("Animals which carrots cost > 55: %d\n",sum); //输出加噪后的数据集中，每日食用胡萝卜大于55的动物个数
    x = laplace_data(beta,&seed); //产生拉普拉斯随机数
    printf("Added noise:%f\t%s \t%f\n",x,"Query Sum (Under DP):", sum+x);
    return sum+x;
}
```

我们首先修改 csv_analysis 函数如上图中的代码所示：我们不对原来的数据集直接添加噪声了，我们则是在统计完 csv 中的数据结果后，在结果上直接添加 Laplace 噪声。

```c
int main()
{
    long int seed;
    int sen = 1;  //对于一个单属性的数据集，其敏感度为1
    double beta;
    srand((unsigned)time( NULL )); //生成基于时间的随机种子（srand方法）
    beta = 0;
    printf("Please input laplace epsilon:");
    scanf("%lf", &beta);

    // input the support times
    int query_frequency = 20;
    // query times: the times we do the query
    int query_times;
    printf("Please input query times:");
    query_times = 1;
    scanf("%d", &query_times);

    if(beta<=0 || !beta)//当输入的beta值无效时，默认设定beta值为1
    {
        beta = 1.0;
    }
    printf("Under privacy budget %f, sanitized query result with laplace noise:\n",beta);
    // each query laplace epsilon
    beta = beta / query_frequency;
    printf("Each query privacy budget %f\n", beta);
    beta = sen / beta; //拉普拉斯机制下，实际公式的算子beta为敏感度/预算
```

在 main 函数之中我们进行一定的修改，首先我们还是需要输入得到期望的隐私预算的也就是 Laplace epsilon 的值，接着我们根据实验的要求会将交互式方案之中支持的查询次数也就是 query frequency 设置为 20，接着我们为了模拟重复推测攻击，我们这里会输入一个执行查询的次数。

```
double* org_data_result;
org_data_result = (double *) malloc(query_times * sizeof(double));
printf("Animals which carrots cost > 55: %d\n",90); // print org result
seed = rand()%10000+10000; //随机种子产生
for(int i = 0; i < query_times; i++){
    double temp = csv_analysis("./zoo.csv",beta,seed); //先调用原始数据集
    org_data_result[i] = temp;
    // printf("Query%d result: \t%f\n", i+1, temp);
}

// compute the average
double org_average = 0;
for(int i = 0; i < query_times; i++){
    org_average += org_data_result[i];
}
printf("Org Sum Query result: \t%f\n", org_average);
org_average = org_average / query_times;
printf("Org Average Query result: \t%f\n", org_average);

printf("==================Using neighbour dataset==================\n");
```

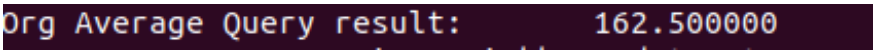
在 main 函数接下来的部分之中，我们会使用动态数组记录每一次查询后的结果进行展示，并且计算出来所有查询结果的求和平均值，对于原始数据集和相邻数据集均进行这样的操作就能够完成我们的交互式发布 DP 方案的模拟了。

为了在 Ubuntu 系统上成功的进行模拟，我们还需要对 Makefile 文件之中进行适当的修改，我们可以简单把所有的 testraw 均换为我们的 test 即可。

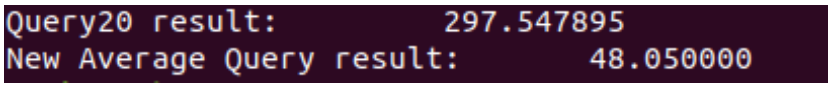
我们来看看实验的结果：

```
zzekun@ubuntu:~/Differential-Privacy/experiment1$ ./test
Please input laplace epsilon:0.1
Please input query times:20
Under privacy budget 0.100000, sanitized query result with laplace noise:
Each query privacy budget 0.005000
Animals which carrots cost > 55: 90
Added noise:302.092206  Query Sum (Under DP):   392.092206
Query1 result:  392.092206
Added noise:-349.631373 Query Sum (Under DP):   -259.631373
Query2 result:  -259.631373
Added noise:4.268834    Query Sum (Under DP):   94.268834
Query3 result:  94.268834
Added noise:33.741452   Query Sum (Under DP):   123.741452
Query4 result:  123.741452
Added noise:32.150340   Query Sum (Under DP):   122.150340
Query5 result:  122.150340
Added noise:341.676759  Query Sum (Under DP):   431.676759
Query6 result:  431.676759
Added noise:459.470731  Query Sum (Under DP):   549.470731
Query7 result:  549.470731
Added noise:105.195052  Query Sum (Under DP):   195.195052
```

我们可以看到我们输入的隐私预算为 0.1，而实际上每次查询的隐私预算仅仅为 0.005，这样的隐私预算带来的噪声是很大的，有相当一部分都在 100 左右，单次查询的保护效果我认为是加强了的。



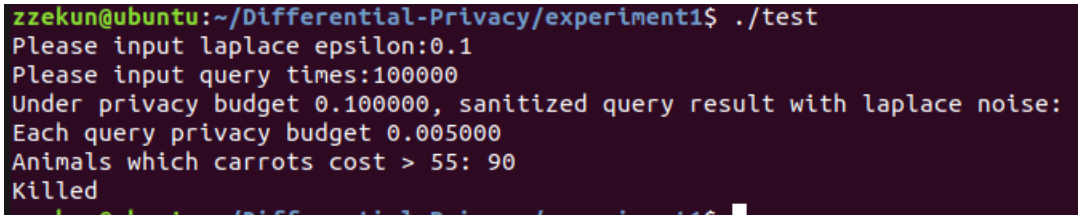
上图展示的为我们在查询 20 次时每次查询结果的求和平均结果，上图为原数据集上测试的结果。



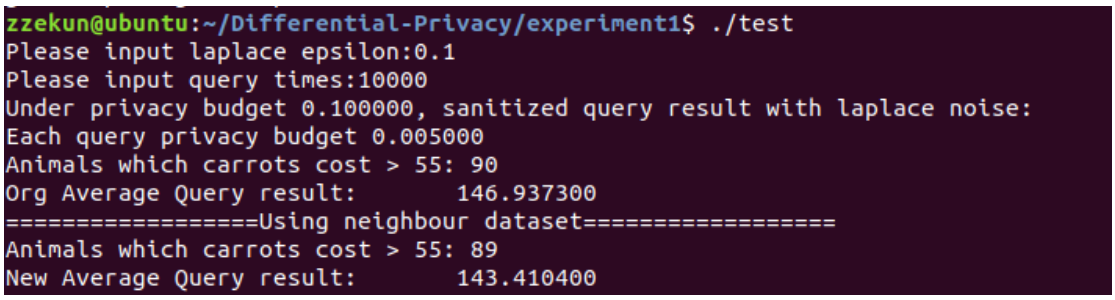
上图展示的是相邻数据集上查询 20 次时每次查询结果的求和平均结果。我们可以看到在原数据集上的测试结果为 162.5，而在相邻数据集上的测试结果为 48.05，这里我们显然不可能现实世界的结果为浮点数，这里我们可以进行强制类型转换为整型。

我们根据上述给出的图表可以得知，在总隐私预算为 0.1 时，经过 10000 次重复查询那么结果落在危险区间的概率为 99.9%，我们的实验要求隐私预算为 0.1，那我们就进行 10000 次查询，但是我们就把中间的 printf 都注释掉即可。（需要进行一定的源码修改，可以将 malloc 动态数组部分该为 double 类型）

我们现在直接输入隐私预算为 0.1，并且我们直接输入查询次数为 10000，20000，30000…进行测试，我们很像进行 100000 次查询模拟，但是 bash 运行时会被 Kill 掉…



接着我们保持总的隐私预算为 0.1 不变，然后我们来改变我们输入的 query times 也就是重复查询次数，我们从 10000 进行模拟，下图为查询次数为 10000 次的结果：



我们可以看到原数据集和相邻数据集上的结果已经十分相近了，仅仅差了 3 点多的差距，这看起来好像还没有那么危险，我们再试试查询 20000 次的结果：

```
zzekun@ubuntu:~/Differential-Privacy/experiment1$ ./test
Please input laplace epsilon:0.1
Please input query times:20000
Under privacy budget 0.100000, sanitized query result with laplace noise:
Each query privacy budget 0.005000
Animals which carrots cost > 55: 90
Org Average Query result:       144.482450
==================Using neighbour dataset==================
Animals which carrots cost > 55: 89
New Average Query result:       143.116100
```

这一次我们可以看到，仅仅相差了 1 点多的差距，严格来说如果取整型，那么就会相差 1 的结果，从攻击者的角度而言，那么其实差分隐私就已经泄露了，但是很奇怪的一点是为什么均值并没有趋近于实际结果 89 与 90，反而在 143-144 之间徘徊，这里怀疑是因为随机数种子与 Laplace 生成算法而导致的。我们来最后看看 30000 次查询后的模拟结果：

```
zzekun@ubuntu:~/Differential-Privacy/experiment1$ ./test
Please input laplace epsilon:0.1
Please input query times:30000
Under privacy budget 0.100000, sanitized query result with laplace noise:
Each query privacy budget 0.005000
Animals which carrots cost > 55: 90
Org Sum Query result:   4369208.982554
Org Average Query result:       145.640299
==================Using neighbour dataset==================
Animals which carrots cost > 55: 89
New Sum Query result:   4339938.493059
New Average Query result:       144.664616
```

我们可以看到原数据集查询的结果为 145.640299，相邻数据集的结果为 144.664616，差距十分逼近 1，个人认为这已经很接近危险区间了。至此，我们的交互式发布 DP 方案就成功完成。

心得体会：

在我们的实验中，我们探索了差分隐私的一个重要概念——Laplace 机制，并使用了拉普拉斯噪声来实现交互式发布的差分隐私方案。

通过我们的实验，我们得出了以下结论：Laplace 机制是一种有效的差分隐私方案，能够在数据发布过程中提供合理的隐私保护。我们还发现，在选择 epsilon 值时需要权衡隐私和数据效用之间的平衡，较小的 epsilon 值提供了更高的隐私保护，但可能降低了查询结果的准确性。最后，我们还模拟了重复推测攻击，进行了攻击测试。