LDP-Shuffle-Parameters

June 19, 2025

1 VLDP - Geolife & Smart Meter dataset

1.0.1 Dataset preprocessing and computing parameters

In this notebook, we show how we created the datasets as used in the paper from the original open datasets. Moreover, we compute the value of the γ parameter for our use case evaluations.

1.0.2 Privacy Parameters

Before going into the specifics of the use cases, we first define the randomizer algorithms for real values and histograms (following the paper).

The code below also determines the privacy amplification we get through shuffling, for some arbitrarily chosen parameters. Give the target overall ϵ and δ and the number of participants n, it gives the ϵ_0 needed by the local randomizer.

We make use of the shuffleddp repository [Balle'19] to determine the bounds of our LDP randomizers and implement them in Python.

Let's first import all the required packages.

[Balle'19] Balle, B., Bell, J., Gascón, A. and Nissim, K., 2019. The privacy blanket of the shuffle model. In Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part II 39 (pp. 638-667). Springer International Publishing.

[5]:
```
import numpy as np
import csv
import os
import pandas as pd
import itertools
import kagglehub
from urllib.request import urlretrieve, urlcleanup
from geopy.geocoders import Nominatim
from geopy.exc import GeocoderTimedOut
from zipfile import ZipFile
import datetime
from shuffleddp.mechanisms import *
from shuffleddp.amplification_bounds import *
```

1.0.3 Algorithm for Real Values

Given $x \in [0, 1]$, the following algorithm calculates the sum of i such values by first encoding them with precision k and then applying the LDP and the Analyzer algorithm given in Section 4.1 of [Balle'19].

[6]:
```
# Target (eps, delta)-guarantee required
eps = 0.1
delta = 1e-6

n = 5000  # number of participants
k = 100  # precision level
rrk = RRMechanism(k=k + 1)  # we have the range of {0, 1, ..., k + 1}

bound_types = [Hoeffding, BennettExact]
all_bounds = []
for B in bound_types:
    all_bounds.append(B(rrk))

print(f"Epsilon: {eps}", eps)
print(f"Delta: {delta}")
print(f"Number of participants: {n}")
bounds = {b.get_name(): b.get_eps0(eps, n, delta) for b in all_bounds}
print(f"Bounds: {bounds}")

gamma = rrk.get_gamma()[0]
print(f"Gamma: {gamma}")


# The first part of the randomizer (float encoding as int)
def encode(x, k):
    p = x * k - np.floor(x * k)
    x_enc = np.floor(x * k) + np.random.binomial(1, p)
    return x_enc


# second part of the randomizer (randomized response)
def RRMech(x, gamma, k):
    if not np.random.binomial(1, gamma):
        return x
    else:
        return np.random.randint(k + 1)


# apply float encoding to random inputs (as example)
true_vals = np.random.rand(1, n)
encode_v = np.vectorize(encode)
enc_true_vals = encode_v(true_vals, k)
```

```
# apply randomized response
RRMech_v = np.vectorize(RRMech)
received_vals = RRMech_v(enc_true_vals[0], gamma, k)

# compute outpus
sample_sum = sum(received_vals)
estimate = (sample_sum / k - gamma * n / 2) / (1 - gamma)
print(f"Estimate: {estimate}")
print(f"Actual: {sum(true_vals[0])}")
print(f"Received sum divided by k: {sample_sum / k}")
```

```
Epsilon: 0.1 0.1
Delta: 1e-06
Number of participants: 5000
Bounds: {'Hoeffding, RR-101': 0.9959842619145971, 'Bennett, RR-101':
2.5523496569511277}
Gamma: 0.89509463046459
Estimate: 2425.1706558514124
Actual: 2515.453082378226
Received sum divided by k: 2492.15
```

1.0.4 Algorithm for Histogram

Given $x \in [k]$, the following algorithm calculates the histogram of values by applying the LDP algorithm given in Section 3.1 of [Balle'19]. Note that the values are already integers.

[7]:
```
# Target (eps, delta)-guarantee required
eps = 0.2
delta = 1e-6

n = 1000  # number of participants
k = 100  # precision level
rrk = RRMechanism(k=k)  # we have the range of {0, 1, ..., k}

bound_types = [Hoeffding, BennettExact]
all_bounds = []
for B in bound_types:
    all_bounds.append(B(rrk))

print(f"Epsilon: {eps}", eps)
print(f"Delta: {delta}")
print(f"Number of participants: {n}")
bounds = {b.get_name(): b.get_eps0(eps, n, delta) for b in all_bounds}
print(f"Bounds: {bounds}")

gamma = rrk.get_gamma()[0]
print(f"Gamma: {gamma}")
```

```python
# the randomizer algorithm (randomized response)
def RRMechHist(x, gamma, k):
    b = np.random.binomial(1, gamma)
    if not b:
        return x
    else:
        return np.random.randint(1, k + 1)


# generate random inputs
true_vals = np.random.choice(np.arange(1, k + 1), n)

# apply randomizer
RRMechHist_v = np.vectorize(RRMechHist)
received_vals = RRMechHist_v(true_vals, gamma, k)

# compute outputs
_unique, true_counts = np.unique(true_vals, return_counts=True)
_unique, est_counts = np.unique(received_vals, return_counts=True)
print(f"Estimate: {est_counts}")
print(f"Actual: {true_counts}")
```

```
Epsilon: 0.2 0.2
Delta: 1e-06
Number of participants: 1000
Bounds: {'Hoeffding, RR-100': 0.9026877389112389, 'Bennett, RR-100':
2.3257610671469986}
Gamma: 0.9154619731616345
Estimate: [21 16  9 10 11 16  8 14 10  7 10 10  6 20 11 13 12 12 13  9  7 16 15
13
  7 13  8  7 11  6 14  8  4  9 11  8 10 13 11 10 10  8  9  8 18 12  5  5
  8 12 11  5  9  8  9 14 10 14  9  3 10 10 11 11  5  8  9 12  7  7 12 10
 11 10 10  8 11  4 11 13  7 10  6 11  7 12 15  7  9  3 12 12 10  3 13 13
  5  8 12  9]
Actual: [ 7 13  7  5 11 13  9 12 13  8  6 16 10 12  8 11 13 11  4  9 14 11 11 10
 11  6  8 11  8 10 11 13 16  6 16 10 14 11 17 10 12 10 11  8  8 10 17  5
 16  8  4  8 10  9  8 10 10 12 13 13  9 10 10 12  8  3 10 10 15  3  7  7
 13 11  6 10  8 10 10 10 11  8 15  7  8 12  7 10  7  9 12  7 10  5 13  6
 10 11 14 12]
```

1.0.5 Smart Meter Data (Use Case 1)

In this Section we describe the dataset preparation and show how we determine the bounds/run a DP example on this dataset.

The dataset is an application of the mechanism for summing up real numbers (from which we can

obtain the average as well), i.e., Algorithms for Real Values, as mentioned above. The dataset is taken from: https://www.kaggle.com/datasets/jeanmidev/smart-meters-in-london. In particular, the dataset `daily_dataset.csv` is used.

Note: to download the original dataset from Kaggle using the code below, one MIGHT (often it is not needed) has to make an account and set up a token following the "Installation" and "Authentication" sections on https://www.kaggle.com/docs/api. One can then uncomment the specified line below and use the information from the token to login to the Kaggle API.

[8]:
```
# NOTE: IF KAGGLE API KEY IS NEEDED UNCOMMENT BELOW
# kagglehub.login()

dataset_path = kagglehub.dataset_download("jeanmidev/smart-meters-in-london")
dataset = "daily_dataset.csv"
df = pd.read_csv(os.path.join(dataset_path, dataset))
```

[9]:
```
# inspect dataset
df.head
```

[9]:
```
<bound method NDFrame.head of              LCLid         day  energy_median
energy_mean  energy_max  \
0         MAC000131  2011-12-15         0.4850     0.432045       0.868
1         MAC000131  2011-12-16         0.1415     0.296167       1.116
2         MAC000131  2011-12-17         0.1015     0.189812       0.685
3         MAC000131  2011-12-18         0.1140     0.218979       0.676
4         MAC000131  2011-12-19         0.1910     0.325979       0.788
…               …           …              …            …           …
3510428   MAC004977  2014-02-24         0.0950     0.118458       0.580
3510429   MAC004977  2014-02-25         0.0675     0.084208       0.176
3510430   MAC004977  2014-02-26         0.1080     0.120500       0.282
3510431   MAC004977  2014-02-27         0.0720     0.114062       0.431
3510432   MAC004977  2014-02-28         0.0970     0.097000       0.097

          energy_count  energy_std  energy_sum  energy_min
0                   22    0.239146       9.505       0.072
1                   48    0.281471      14.216       0.031
2                   48    0.188405       9.111       0.064
3                   48    0.202919      10.511       0.065
4                   48    0.259205      15.647       0.066
…                  …           …           …           …
3510428             48    0.093814       5.686       0.052
3510429             48    0.037107       4.042       0.046
3510430             48    0.069332       5.784       0.046
3510431             48    0.094482       5.475       0.047
3510432              1         NaN       0.097       0.097

[3510433 rows x 9 columns]>
```

1.0.6 DP Parameters and Example Run (Use Case 1)

[10]:
```
# maximum possible value of energy -- we will normalize using this
max_energy = df["energy_mean"].max()
print("Max energy:", max_energy)

# Total number of households
households = df["LCLid"].unique()
n = len(households)

eps = 0.2  # Target (eps, delta)-guarantee required
delta = 1e-6
k = 10  # precision level
rrk = RRMechanism(k=k + 1)

bound_types = [Hoeffding, BennettExact]
all_bounds = []
for B in bound_types:
    all_bounds.append(B(rrk))

print(f"Epsilon: {eps}", eps)
print(f"Delta: {delta}")
print(f"Number of participants: {n}")
bounds = {b.get_name(): b.get_eps0(eps, n, delta) for b in all_bounds}
print(f"Bounds: {bounds}")

gamma = rrk.get_gamma()[0]
print(f"Gamma: {gamma}")

num_days = int(1 / eps)  # we will run the mechanism a total of 1/eps times

print("k:", k)
print("n:", n)
print("Number of days:", num_days)
print("eps0:", rrk.get_eps0())
print("================\n")

last_day = "2014-02-25"
cur_date = datetime.datetime.strptime(last_day, '%Y-%m-%d').date()
delta = datetime.timedelta(days=1)


# Normalize energy values within [0, 1]
def normalizeVals(vals):
    for i in range(len(vals)):
        vals[i] = vals[i] / (max_energy)
    return vals
```

```
# also store data for writing to csv
energy_vals_for_csv = []

# do an example DP run on this data and simultaneously parse the data
for i in range(num_days):
    day = cur_date.strftime('%Y-%m-%d')
    cur_date -= delta
    df0 = df[['LCLid', 'day', 'energy_mean']]
    df1 = df0[df0['day'] == day]
    energy_vals = [0.0 for i in range(len(households))]
    for j in range(len(households)):
        energy = df1.loc[df1['LCLid'] == households[j], 'energy_mean'].values
        if energy.size != 0:
            energy_vals[j] = energy[0]
    energy_vals = normalizeVals(energy_vals)
    true_vals = energy_vals
    encode_v = np.vectorize(encode)
    enc_true_vals = encode_v(true_vals, k)

    RRMech_v = np.vectorize(RRMech)
    received_vals = RRMech_v(enc_true_vals, gamma, k)

    sample_sum = sum(received_vals)
    # This is the de-biasing step in Algorithm 3 of [Balle'19]
    estimate = (sample_sum / k - gamma * n / 2) / (1 - gamma)
    print(f"Run {i + 1}:")
    print(f"Estimate: {estimate / n}")
    print(f"Actual: {sum(true_vals) / n}")
    print("==================\n")
    energy_vals_for_csv.append(energy_vals)
```

```
Max energy: 6.928250020833329
Epsilon: 0.2 0.2
Delta: 1e-06
Number of participants: 5566
Bounds: {'Hoeffding, RR-11': 1.6116776981605088, 'Bennett, RR-11':
2.48270496095323}
Gamma: 0.5006005204469995
k: 10
n: 5566
Number of days: 5
eps0: 2.482704960952729
================

Run 1:
```

```
Estimate: 0.039475637365392724
Actual: 0.027832181900990727
==================

Run 2:
Estimate: 0.034798802185133454
Actual: 0.028096308785484303
==================

Run 3:
Estimate: 0.007349377242534734
Actual: 0.031060984630618366
==================

Run 4:
Estimate: 0.028251232932770456
Actual: 0.029218913551385617
==================

Run 5:
Estimate: 0.03472685087466791
Actual: 0.028447346459788043
==================
```

1.0.7 Writing Extracted Smart Meter Data

The following extracts only relevant information into a CSV file. Namely the average energy consumption per household over the days used in the algorithm.

[11]:
```
# Write the normalized energy values into a csv file
with open("energy_data.csv", "w", newline='') as f:
    wr = csv.writer(f, delimiter=",")
    header_row = ["household", "day", "average energy"]
    wr.writerow(header_row)
    for i in range(len(energy_vals_for_csv)):
        vals = energy_vals_for_csv[i]
        for j in range(len(vals)):
            row = [j, i, vals[j]]
            wr.writerow(row)
```

1.0.8 Geolife GPS Trajectory Dataset (Use Case 2)

In this Section we describe the dataset preparation and show how we determine the bounds/run a DP example on this dataset.

Taken from https://www.microsoft.com/en-us/research/publication/geolife-gps-trajectory-dataset-user-guide/. The following extracts the first longitude, latitude entry on a given date from files corresponding to all users. Not all users have date for each day.

First we download and unpack the original data. (Note: This can take up to 10-15 minutes, since it's a lot of data to unpack)

[12]:
```
zip_path, _headers = urlretrieve(
    "https://download.microsoft.com/download/F/4/8/
 ↪F4894AA5-FDBC-481E-9285-D5F8C4C4F039/Geolife%20Trajectories%201.3.zip")

with ZipFile(zip_path, 'r') as zip_file:
    zip_file.extractall(os.getcwd())

# remove downloaded tmp file
urlcleanup()
```

1.0.9 First Latitude, Longitude Reading from All Users One Day at a Time

We only need a subselection of the data, so we do that as follows. This code takes the first lat long from each user's file of the first 5 days.

[13]:
```
work_dir = "Geolife Trajectories 1.3/Data"
users = ["{:03d}".format(i) for i in range(182)]
sub_dir = "Trajectory"

latLongs = []
days = 5

for user in users:
    cur_dir = os.path.join(work_dir, user, sub_dir)
    files = sorted([filename for filename in os.listdir(cur_dir)])
    for day in range(days):
        if day < len(files):
            cur_file = os.path.join(cur_dir, files[day])
            with open(cur_file, 'r') as f:
                reader = csv.reader(f, delimiter='|')
                rows = list(reader)

            flat_rows = itertools.chain.from_iterable(rows)
            list_rows = [i.strip().split(',') for i in flat_rows]

            df = pd.DataFrame(list_rows[6:])  # first 6 lines are useless

            lat = df.iloc[0][0]
            long = df.iloc[0][1]
            latLongs.append([user, day, lat, long])

print(f"Number of records: {len(latLongs)}")
print("Head:")
print(latLongs[:6])
```

```
Number of records: 851
Head:
[['000', 0, '39.984702', '116.318417'], ['000', 1, '40.008304', '116.319876'],
['000', 2, '39.907414', '116.370017'], ['000', 3, '39.994622', '116.326757'],
['000', 4, '40.01229', '116.297072'], ['001', 0, '39.984094', '116.319236']]
```

1.0.10 Calling the Reverse Geo API

Next we transform the latitudes and longitudes found in the data into postcodes using a geodata lookup.

NOTE: please specify the use agent on line 7, for example use your e-mail or the name of the application. This is necessary to follow the conditions of the API used.

[14]:
```python
# TODO: set USER AGENT
# initialize Nominatim API
# 1)
from geopy.geocoders import Nominatim
from geopy.exc import GeocoderTimedOut, GeocoderServiceError
from geopy.extra.rate_limiter import RateLimiter
import time

# 2)
geolocator = Nominatim(user_agent="mautadoz@gmail.com", timeout=10)

# 3) RateLimiter
reverse = RateLimiter(
    geolocator.reverse,
    min_delay_seconds=1,
    max_retries=2,
    error_wait_seconds=5
)
for idx, data in enumerate(latLongs):
    print(f"{idx + 1}/{len(latLongs)}")
    lat = data[2]
    long = data[3]
    try:
        location = geolocator.reverse(lat + "," + long, language='en')
        address = location.raw['address']
        if 'postcode' in address:
            data.append(address['postcode'])
        else:
            data.append("None")
    except GeocoderTimedOut as e:
        print("Error: geocode failed on input %s" % (lat + "," + long))
        data.append("TimeOut")
print("done")
```

```
1/851
```

2/851
3/851
4/851
5/851
6/851
7/851
8/851
9/851
10/851
11/851
12/851
13/851
14/851
15/851
16/851
17/851
18/851
19/851
20/851
21/851
22/851
23/851
24/851
25/851
26/851
27/851
28/851
29/851
30/851
31/851
32/851
33/851
34/851
35/851
36/851
37/851
38/851
39/851
40/851
41/851
42/851
43/851
44/851
45/851
46/851
47/851
48/851
49/851

50/851
51/851
52/851
53/851
54/851
55/851
56/851
57/851
58/851
59/851
60/851
61/851
62/851
63/851
64/851
65/851
66/851
67/851
68/851
69/851
70/851
71/851
72/851
73/851
74/851
75/851
76/851
77/851
78/851
79/851
80/851
81/851
82/851
83/851
84/851
85/851
86/851
87/851
88/851
89/851
90/851
91/851
92/851
93/851
94/851
95/851
96/851
97/851

98/851
99/851
100/851
101/851
102/851
103/851
104/851
105/851
106/851
107/851
108/851
109/851
110/851
111/851
112/851
113/851
114/851
115/851
116/851
117/851
118/851
119/851
120/851
121/851
122/851
123/851
124/851
125/851
126/851
127/851
128/851
129/851
130/851
131/851
132/851
133/851
134/851
135/851
136/851
137/851
138/851
139/851
140/851
141/851
142/851
143/851
144/851
145/851

146/851
147/851
148/851
149/851
150/851
151/851
152/851
153/851
154/851
155/851
156/851
157/851
158/851
159/851
160/851
161/851
162/851
163/851
164/851
165/851
166/851
167/851
168/851
169/851
170/851
171/851
172/851
173/851
174/851
175/851
176/851
177/851
178/851
179/851
180/851
181/851
182/851
183/851
184/851
185/851
186/851
187/851
188/851
189/851
190/851
191/851
192/851
193/851

194/851
195/851
196/851
197/851
198/851
199/851
200/851
201/851
202/851
203/851
204/851
205/851
206/851
207/851
208/851
209/851
210/851
211/851
212/851
213/851
214/851
215/851
216/851
217/851
218/851
219/851
220/851
221/851
222/851
223/851
224/851
225/851
226/851
227/851
228/851
229/851
230/851
231/851
232/851
233/851
234/851
235/851
236/851
237/851
238/851
239/851
240/851
241/851

242/851
243/851
244/851
245/851
246/851
247/851
248/851
249/851
250/851
251/851
252/851
253/851
254/851
255/851
256/851
257/851
258/851
259/851
260/851
261/851
262/851
263/851
264/851
265/851
266/851
267/851
268/851
269/851
270/851
271/851
272/851
273/851
274/851
275/851
276/851
277/851
278/851
279/851
280/851
281/851
282/851
283/851
284/851
285/851
286/851
287/851
288/851
289/851

290/851
291/851
292/851
293/851
294/851
295/851
296/851
297/851
298/851
299/851
300/851
301/851
302/851
303/851
304/851
305/851
306/851
307/851
308/851
309/851
310/851
311/851
312/851
313/851
314/851
315/851
316/851
317/851
318/851
319/851
320/851
321/851
322/851
323/851
324/851
325/851
326/851
327/851
328/851
329/851
330/851
331/851
332/851
333/851
334/851
335/851
336/851
337/851

338/851
339/851
340/851
341/851
342/851
343/851
344/851
345/851
346/851
347/851
348/851
349/851
350/851
351/851
352/851
353/851
354/851
355/851
356/851
357/851
358/851
359/851
360/851
361/851
362/851
363/851
364/851
365/851
366/851
367/851
368/851
369/851
370/851
371/851
372/851
373/851
374/851
375/851
376/851
377/851
378/851
379/851
380/851
381/851
382/851
383/851
384/851
385/851

386/851
387/851
388/851
389/851
390/851
391/851
392/851
393/851
394/851
395/851
396/851
397/851
398/851
399/851
400/851
401/851
402/851
403/851
404/851
405/851
406/851
407/851
408/851
409/851
410/851
411/851
412/851
413/851
414/851
415/851
416/851
417/851
418/851
419/851
420/851
421/851
422/851
423/851
424/851
425/851
426/851
427/851
428/851
429/851
430/851
431/851
432/851
433/851

434/851
435/851
436/851
437/851
438/851
439/851
440/851
441/851
442/851
443/851
444/851
445/851
446/851
447/851
448/851
449/851
450/851
451/851
452/851
453/851
454/851
455/851
456/851
457/851
458/851
459/851
460/851
461/851
462/851
463/851
464/851
465/851
466/851
467/851
468/851
469/851
470/851
471/851
472/851
473/851
474/851
475/851
476/851
477/851
478/851
479/851
480/851
481/851

482/851
483/851
484/851
485/851
486/851
487/851
488/851
489/851
490/851
491/851
492/851
493/851
494/851
495/851
496/851
497/851
498/851
499/851
500/851
501/851
502/851
503/851
504/851
505/851
506/851
507/851
508/851
509/851
510/851
511/851
512/851
513/851
514/851
515/851
516/851
517/851
518/851
519/851
520/851
521/851
522/851
523/851
524/851
525/851
526/851
527/851
528/851
529/851

530/851
531/851
532/851
533/851
534/851
535/851
536/851
537/851
538/851
539/851
540/851
541/851
542/851
543/851
544/851
545/851
546/851
547/851
548/851
549/851
550/851
551/851
552/851
553/851
554/851
555/851
556/851
557/851
558/851
559/851
560/851
561/851
562/851
563/851
564/851
565/851
566/851
567/851
568/851
569/851
570/851
571/851
572/851
573/851
574/851
575/851
576/851
577/851

578/851
579/851
580/851
581/851
582/851
583/851
584/851
585/851
586/851
587/851
588/851
589/851
590/851
591/851
592/851
593/851
594/851
595/851
596/851
597/851
598/851
599/851
600/851
601/851
602/851
603/851
604/851
605/851
606/851
607/851
608/851
609/851
610/851
611/851
612/851
613/851
614/851
615/851
616/851
617/851
618/851
619/851
620/851
621/851
622/851
623/851
624/851
625/851

626/851
627/851
628/851
629/851
630/851
631/851
632/851
633/851
634/851
635/851
636/851
637/851
638/851
639/851
640/851
641/851
642/851
643/851
644/851
645/851
646/851
647/851
648/851
649/851
650/851
651/851
652/851
653/851
654/851
655/851
656/851
657/851
658/851
659/851
660/851
661/851
662/851
663/851
664/851
665/851
666/851
667/851
668/851
669/851
670/851
671/851
672/851
673/851

674/851
675/851
676/851
677/851
678/851
679/851
680/851
681/851
682/851
683/851
684/851
685/851
686/851
687/851
688/851
689/851
690/851
691/851
692/851
693/851
694/851
695/851
696/851
697/851
698/851
699/851
700/851
701/851
702/851
703/851
704/851
705/851
706/851
707/851
708/851
709/851
710/851
711/851
712/851
713/851
714/851
715/851
716/851
717/851
718/851
719/851
720/851
721/851

722/851
723/851
724/851
725/851
726/851
727/851
728/851
729/851
730/851
731/851
732/851
733/851
734/851
735/851
736/851
737/851
738/851
739/851
740/851
741/851
742/851
743/851
744/851
745/851
746/851
747/851
748/851
749/851
750/851
751/851
752/851
753/851
754/851
755/851
756/851
757/851
758/851
759/851
760/851
761/851
762/851
763/851
764/851
765/851
766/851
767/851
768/851
769/851

770/851
771/851
772/851
773/851
774/851
775/851
776/851
777/851
778/851
779/851
780/851
781/851
782/851
783/851
784/851
785/851
786/851
787/851
788/851
789/851
790/851
791/851
792/851
793/851
794/851
795/851
796/851
797/851
798/851
799/851
800/851
801/851
802/851
803/851
804/851
805/851
806/851
807/851
808/851
809/851
810/851
811/851
812/851
813/851
814/851
815/851
816/851
817/851

```
818/851
819/851
820/851
821/851
822/851
823/851
824/851
825/851
826/851
827/851
828/851
829/851
830/851
831/851
832/851
833/851
834/851
835/851
836/851
837/851
838/851
839/851
840/851
841/851
842/851
843/851
844/851
845/851
846/851
847/851
848/851
849/851
850/851
851/851
done
```

[15]:
```
print("Head:")
print(latLongs[:6])
```

```
Head:
[['000', 0, '39.984702', '116.318417', '100190'], ['000', 1, '40.008304',
'116.319876', '100084'], ['000', 2, '39.907414', '116.370017', '100032'],
['000', 3, '39.994622', '116.326757', '100084'], ['000', 4, '40.01229',
'116.297072', '100091'], ['001', 0, '39.984094', '116.319236', '100190']]
```

1.0.11 Data Preparation for CSV

In the code below we make our data ready for the use case. First, we condense the list of postcodes to the top 7 most used ones. The other are aggregated under "all_others". Then we will the missing entries with the "all_others" category as well and write the resulting data to a CSV file.

Note: the resulting .csv file might be slightly different from the one we created, the geodata api calls are not always consistent (timeouts may happen), so this could cause some changes. The overall file will be very similar though.

[16]:
```
df = pd.DataFrame(latLongs, columns=["User", "day", "lat", "long", "postcode"])

# reduce the number of postcodes to the top 7, replace rest by "all_others"
top_postcodes = df.value_counts(subset="postcode").drop("None").nlargest(7).
 ↪keys()
df["postcode"] = df["postcode"].mask(df["postcode"].isin(top_postcodes) ==␣
 ↪False, "all_others")

# fill missing data
users = ["{:03d}".format(i) for i in range(182)]
days = 5
for day in range(days):
    for user in users:
        if not any((df["User"] == user) & (df["day"] == day)):
            df.loc[len(df)] = [user, day, None, None, "all_others"]

# Write to CSV
df.to_csv("geolife-postcodes-condensed-empties.csv", index=False)
```

1.0.12 Inspect the final dataset distribution

Below we look at the true distribution of the resulting dataset

[17]:
```
df = pd.read_csv("geolife-postcodes-condensed-empties.csv")
days = df['day'].unique()
users = df['User'].unique()
counts = {}
for day in days:
    counts[day] = {}
    postcodes = df.loc[df['day'] == day, 'postcode'].unique()
    for postcode in postcodes:
        counts[day][postcode] = df[(df['day'] == day) & (df['postcode'] ==␣
 ↪postcode)].shape[0]

for day in counts:
    c = 0
    for k, v in counts[day].items():
        print(k, v)
```

```
        c = c + v
    print(f"This count: {c}")
    print()
```

```
100190 17
all_others 49
100084 5
100083 5
100098 85
100872 7
100871 2
100081 12
This count: 182

100084 10
all_others 82
100083 12
100081 15
100190 12
100098 36
100872 10
100871 5
This count: 182

all_others 104
100084 13
100083 8
100098 26
100190 10
100081 6
100871 5
100872 10
This count: 182

100084 13
all_others 94
100083 12
100081 7
100098 28
100190 7
100872 13
100871 8
This count: 182

all_others 104
100098 25
100084 8
```

```
100083 6
100190 12
100081 12
100872 12
100871 3
This count: 182
```

1.0.13 DP Parameters and Example Run (Use Case 2)

[18]:
```python
# Total number of users
users = df["User"].unique()
n = len(users)
postcodes = df["postcode"].unique()
k = len(postcodes)  # histogram of postcodes
eps = 2  # Target (eps, delta)-guarantee required
# over multiple runs the eps add up, e.g., 5 runs => 5*eps
delta = 1e-4

rrk = RRMechanism(k=k)  # we have the range of {0, 1, ..., k}

bound_types = [Hoeffding, BennettExact]
all_bounds = []
for B in bound_types:
    all_bounds.append(B(rrk))

print(f"Epsilon: {eps}", eps)
print(f"Delta: {delta}")
print(f"Number of participants: {n}")
bounds = {b.get_name(): b.get_eps0(eps, n, delta) for b in all_bounds}
print(f"Bounds: {bounds}")

gamma = rrk.get_gamma()[0]
print(f"Gamma: {gamma}")

days = df["day"].unique()


def RRMech(x, gamma, postcodes):
    if not np.random.binomial(1, gamma):
        return x
    else:
        return np.random.choice(postcodes)


orig_dict = {}
syn_dict = {}
```

```
# do an example run
i = 0
for day in days:
    i += 1
    orig_dict[day] = {}
    syn_dict[day] = {}
    df0 = df[['User', 'day', 'postcode']]
    df1 = df0[df0['day'] == day]
    for user in users:
        postcode = df1.loc[df1['User'] == user, 'postcode'].values
        #print(postcode)
        if postcode.size != 0:
            postcode = postcode[0]
            if postcode in orig_dict[day]:
                orig_dict[day][postcode] += 1
            else:
                orig_dict[day][postcode] = 1
            priv_postcode = RRMech(postcode, gamma, postcodes)
            if priv_postcode in syn_dict[day]:
                syn_dict[day][priv_postcode] += 1
            else:
                syn_dict[day][priv_postcode] = 1

    print("Run " + str(i) + ":")
    print("Postcode, DP, Original:")
    print("==========\n")
    for k, v1 in syn_dict[day].items():
        v2 = orig_dict[day][k]
        print(k + ", " + str(v1) + ", " + str(v2))
```

```
Epsilon: 2 2
Delta: 0.0001
Number of participants: 182
Bounds: {'Hoeffding, RR-8': 2.0567127248541412, 'Bennett, RR-8':
2.4982876252697417}
Gamma: 0.4175005627940103
Run 1:
Postcode, DP, Original:
==========

100872, 12, 7
100081, 19, 12
100084, 12, 5
all_others, 43, 49
100098, 56, 85
100190, 21, 17
```

```
100083, 10, 5
100871, 9, 2
Run 2:
Postcode, DP, Original:
==========

100084, 14, 10
100871, 21, 5
all_others, 54, 82
100083, 17, 12
100190, 13, 12
100098, 29, 36
100872, 10, 10
100081, 24, 15
Run 3:
Postcode, DP, Original:
==========

all_others, 75, 104
100083, 20, 8
100084, 12, 13
100098, 24, 26
100081, 8, 6
100190, 16, 10
100872, 13, 10
100871, 14, 5
Run 4:
Postcode, DP, Original:
==========

100084, 14, 13
all_others, 62, 94
100872, 21, 13
100098, 25, 28
100083, 17, 12
100081, 15, 7
100190, 13, 7
100871, 15, 8
Run 5:
Postcode, DP, Original:
==========

all_others, 72, 104
100098, 20, 25
100084, 17, 8
100083, 12, 6
100190, 19, 12
100872, 14, 12
```

```
100081, 18, 12
100871, 10, 3
```