Project 3

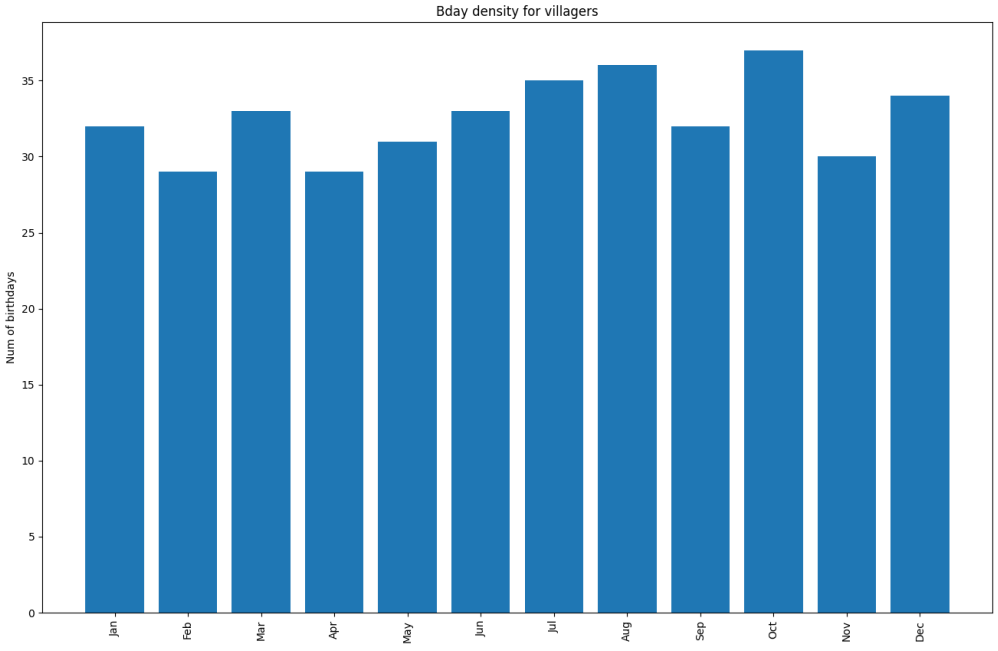
Chase lane cdl52

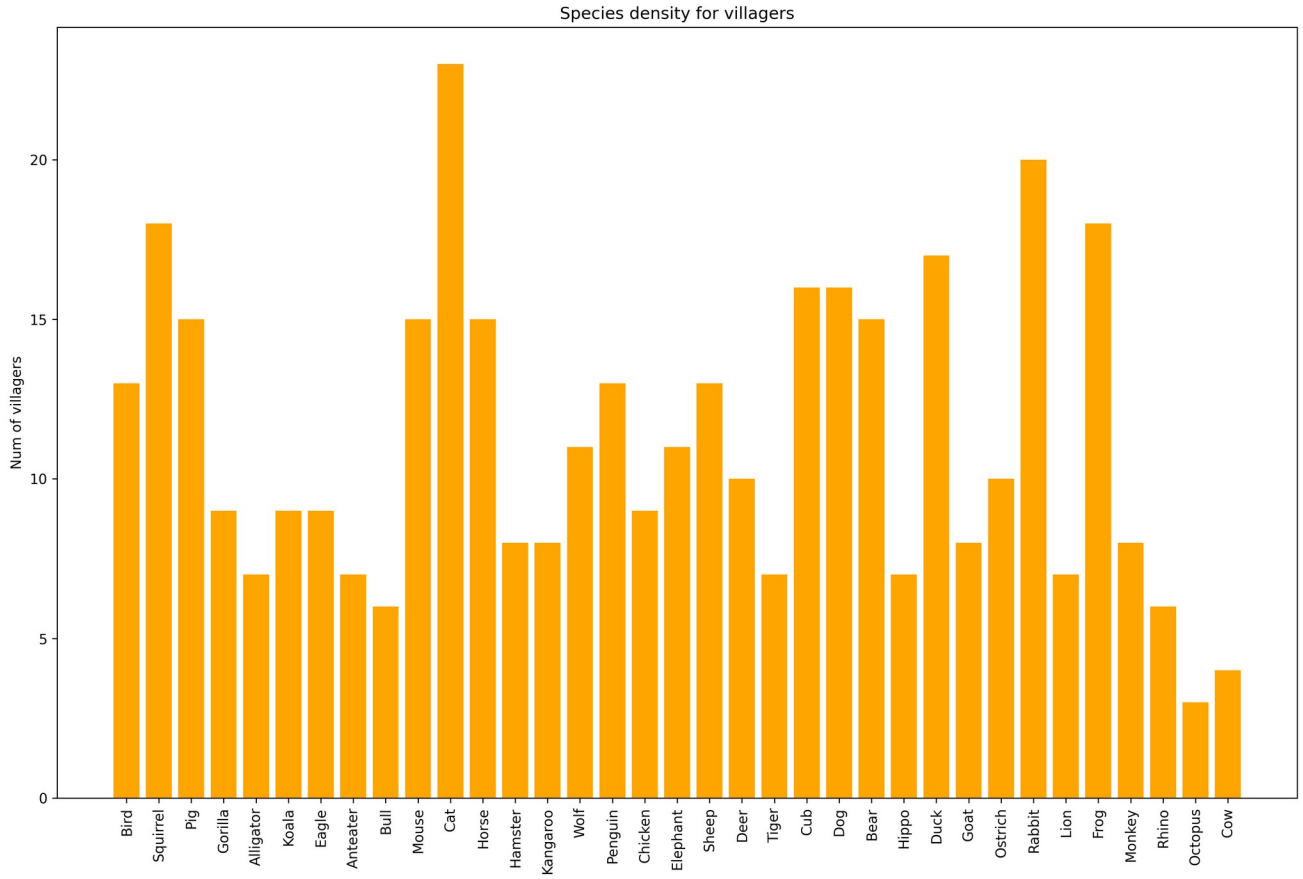
Task W0

I chose to look at a data set containing all the villagers of the game Animal Crossing New Horizon. The game has you move into a desert island and essentially build up civilization attracting new people to come live there. The villagers data set **(1)** contains information pertaining to villagers such as species, gender, and information about their favorite songs and colors. The original data set contains the villager's names but for the purpose of this project, I am omitting this column and proceeding from there. I am also removing columns wallpaper, flooring, furniture list, style, and filename as they will be irrelevant for the analysis. In the end the columns that remain are: Species, Gender, Personality, Hobby, Birthday, Catchphrase, Favorite Song

Task W1

Villagers in the game have unique attributes such as favorite song and a catchphrase. The data in the current form reveals this sensitive information through numerous quasi-identifiers such as species, gender, and date of birth. If an attacker successfully knew all three, which could easily be determined by looks and a brief conversation with the person, they could link the information easily. Even with just a subset of these three identifiers, an adversary could still narrow down the choices for favorite song and catchphrases. The identifiers are so revealing because there are 391 unique villagers present, which belong to 35 different species. The first bar graph displays the density of birthdays which is fairly uniform. The second bar graph displays the density for each of the 35 species with some being low relative to others and therefore easier to distinguish.



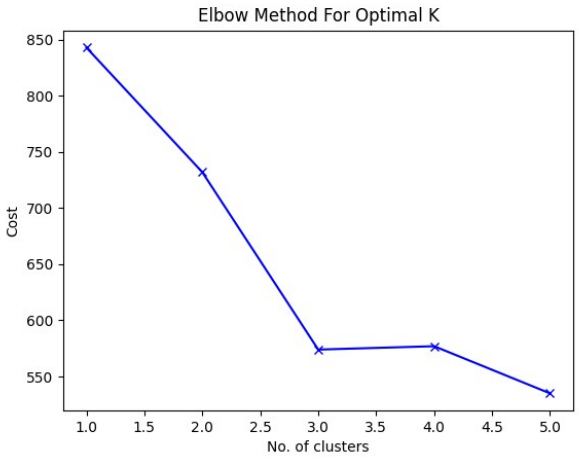

Species density for villagers
Num of villagers
0
5
10
15
20
Bird
Squirrel
Pig
Gorilla
Alligator
Koala
Eagle
Anteater
Bull
Mouse
Cat
Horse
Hamster
Kangaroo
Wolf
Penguin
Chicken
Elephant
Sheep
Deer
Tiger
Cub
Dog
Bear
Hippo
Duck
Goat
Ostrich
Rabbit
Lion
Frog
Monkey
Rhino
Octopus
Cow

Task C2

Because the birthday months are uniformly distributed, I opted for 4-k anonymity by diving the birthday into four groups: [Jan-Mar], [Apr-Jun], [Jul-Sep], [Oct-Dec]. This clustering based approach only partially obfuscates the quasi-identifiers. The functioning for clustering is described in this code chunk and is called as a part of the k_anonymity() function.

```
# group bday into 4 groups
def group_bday(df):
    gr1 = "[Jan-Mar]"
    gr2 = "[Apr-Jun]"
    gr3 = "[Jul-Sep]"
    gr4 = "[Oct-Dec]"
    for r in df.index:
        curr = df['Birthday'][r]
        if curr == 'Jan' or curr == 'Feb' or curr == 'Mar':
            df['Birthday'][r] = gr1
        if curr == 'Apr' or curr == 'May' or curr == 'Jun':
            df['Birthday'][r] = gr2
        if curr == 'Jul' or curr == 'Aug' or curr == 'Sep':
            df['Birthday'][r] = gr3
        if curr == 'Oct' or curr == 'Nov' or curr == 'Dec':
            df['Birthday'][r] = gr4
```

Because species don't belong to identifiable clusters, I opted to use k-modes clustering for generalizing them. Clustering is a technique to divide the data into a number of groups so that data in a group are more similar to other data points in that group. Because species is a categorical variable, k-modes clustering is used over k-means clustering. First, the optimal number of clusters is chosen using the "Huang" method. This chooses k based on the cost between groups, and when this cost becomes less than strictly decreasing, aka an "elbow," k is chosen at that point. The predictors for the groups utilizes the birthday clustering and species. From this method, k was chosen at 3.




```
# find the optimal K for k modes clustering
def find_optimalK(df, predictors):
    # k modes clustering using birthday and species
    # Elbow curve to find optimal K
    cost = []
    K = range(1,6)
    for num_clusters in list(K):
        kmode = KModes(n_clusters=num_clusters, init="Huang", n_init=1, verbose=1)
        kmode.fit_predict(df[predictors])
        cost.append(kmode.cost_)
    plt.plot(K, cost, 'bx-')
    plt.xlabel('No. of clusters')
    plt.ylabel('Cost')
    plt.title('Elbow Method For Optimal K')
    plt.savefig('./output/optimalK.png')
    # find optimal k based on when the graph stops going from strictly decreasing
    change = 0
    optimal = -1
    for i in range(len(cost)):
        if i > 0:
            curr = cost[i-1]-cost[i]
            if curr < change:
                optimal = i
                break
            change = curr
    return optimal
```

With k chosen at 3, the k modes package **(2)** can now fit the clusters. This is made as simple as calling k_modes(df, k, predictors). The k_anonymity() function does everything needed for birthday grouping and creating the k-modes clusters.

```
def k_anonymity():
    # creating k anonymity for species and birth month using k modes clustering
    kanon = villagerDF.copy()
    group_bday(kanon)
    predictors = ["Birthday", "Species"]
    k = find_optimalK(kanon, predictors)
    fit_kmodes(kanon, k, predictors)
    print(kanon.head(10))
    print("Fit", k, "clusters")
```

Task W3

Original Data

	Gender	Species	Birthday	Personality	Hobby	Catchphrase	Favorite Song	Unique Entry ID
0	Male	Bird	Jan	Cranky	Nature	aye aye	Steep Hill	B3RyfNEqwGmcccRC3
1	Female	Squirrel	Jul	Peppy	Fitness	sidekick	Go K.K. Rider	SGMdki6dzpDZyXAw5
2	Female	Pig	Apr	Big Sister	Play	snuffle	K.K. House	jzWCiDPm9MqtCfecP
3	Male	Gorilla	Oct	Lazy	Fitness	ayyyeee	Go K.K. Rider	LBifxETQJGEaLhBjC
4	Male	Alligator	Jun	Lazy	Play	it'sa me	Forest Life	REpd8KxB8p9aGBRSE
5	Female	Koala	Aug	Normal	Education	guvnor	Aloha K.K.	wkPJDHMMMTK24eqzC
6	Female	Alligator	Nov	Snooty	Fashion	graaagh	K.K. Soul	dcB3BpFLtcXHAbt5i
7	Female	Eagle	Nov	Snooty	Music	cuz	K.K. Condor	gWpcHhfpuN4MSTdhF
8	Female	Anteater	Feb	Peppy	Fashion	snorty	Aloha K.K.	gvdKbp6t9xMtx5swT
9	Male	Bird	Mar	Lazy	Play	chuurp	K.K. Ragtime	T9ya79MPxRxcR4tae

k-anonymity Data, k=4

Fit 3 clusters

	Cluster	Birthday	Personality	Hobby	Catchphrase	Favorite Song	Unique Entry ID
0	0	[Jan-Mar]	Cranky	Nature	aye aye	Steep Hill	B3RyfNEqwGmcccRC3
1	0	[Jul-Sep]	Peppy	Fitness	sidekick	Go K.K. Rider	SGMdki6dzpDZyXAw5
2	1	[Apr-Jun]	Big Sister	Play	snuffle	K.K. House	jzWCiDPm9MqtCfecP
3	0	[Oct-Dec]	Lazy	Fitness	ayyyeee	Go K.K. Rider	LBifxETQJGEaLhBjC
4	0	[Apr-Jun]	Lazy	Play	it'sa me	Forest Life	REpd8KxB8p9aGBRSE
5	1	[Jul-Sep]	Normal	Education	guvnor	Aloha K.K.	wkPJDHMMMTK24eqzC
6	0	[Oct-Dec]	Snooty	Fashion	graaagh	K.K. Soul	dcB3BpFLtcXHAbt5i
7	0	[Oct-Dec]	Snooty	Music	cuz	K.K. Condor	gWpcHhfpuN4MSTdhF
8	1	[Jan-Mar]	Peppy	Fashion	snorty	Aloha K.K.	gvdKbp6t9xMtx5swT
9	0	[Jan-Mar]	Lazy	Play	chuurp	K.K. Ragtime	T9ya79MPxRxcR4tae

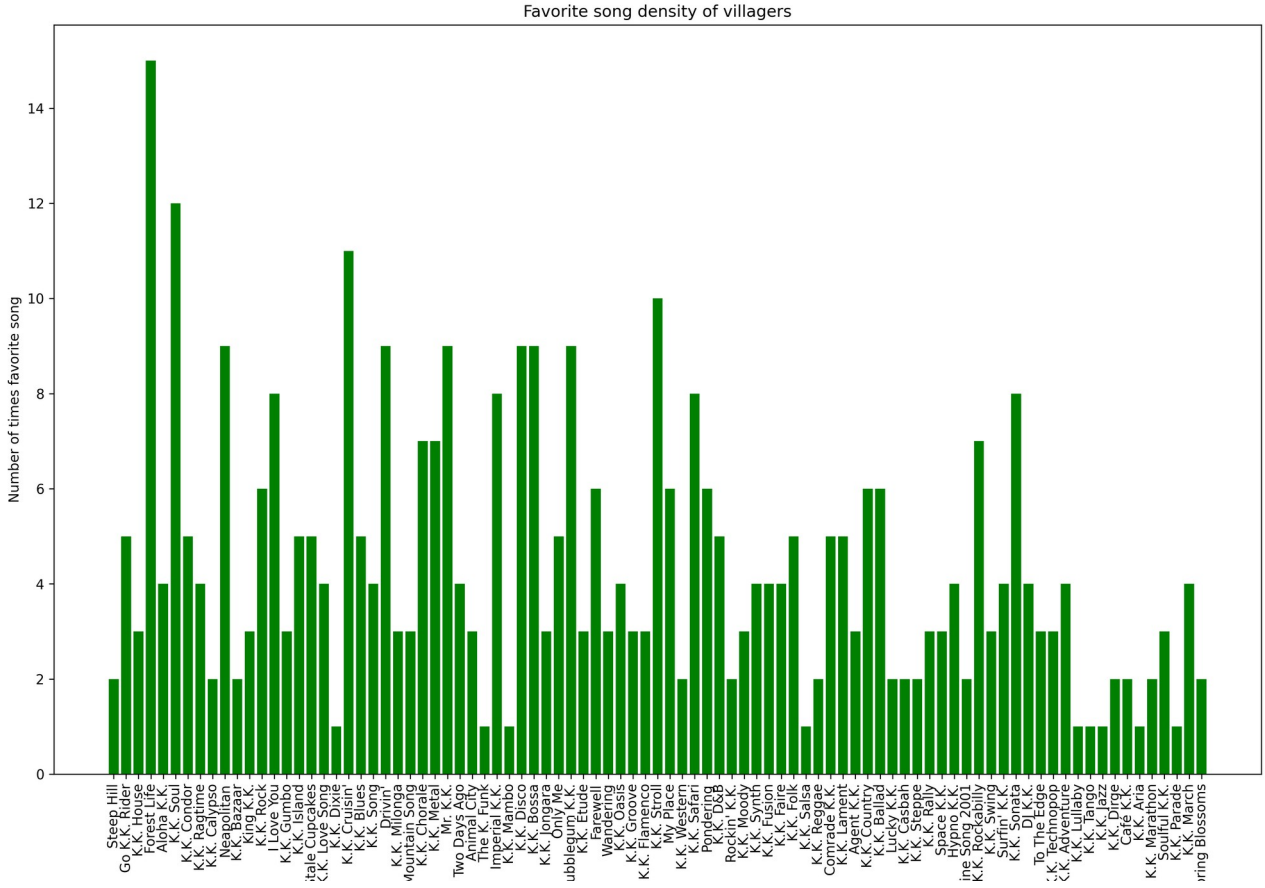
The original data has way too much information revealed through Gender, Species, and Birthday, while the modified k-anonymous data has very limited information revealed. First, the Birthday is grouped in a range to avoid revealing the specific month. Gender and Species were dropped entirely and instead shown as the “Cluster” column. The unique entry ID was still left there to compare the data before and after and can be ignored. One issue with the clusters is the uneven amount of villagers placed in each group. Group 2 being the least common one, which could be used as identification. The cost of the clustering approach comes at less utility. We don’t know how the clusters were generated and therefore are just an arbitrary identifier other than they have some similarity within the cluster. In comparison, we could see which villagers were Male/Female in the original data. If a data set were used for example to test the effectiveness of a new vaccination, we could not apply the clustering approach to important QIs like gender due to the importance of the data. But in the villager data case, we can afford this cost because species and gender aren’t being used in a way that is vital for the data. If an attacker knew some of the villagers habits say by living near them, they could identify some of the sensitive information like their favorite Hobby and use that as an identifier to further dissect the table and narrow options. In this case, if an attacker were well versed on their target, they could use their hobby/favorite song as an identifier.

Task C4

For the sensitive information query, I chose to look at villager's favorite songs due to the distribution of them. There is a clear preference for some songs and some songs are not liked nearly as universally. The function find_favorite_song(df, "song") returns the number of villagers who like the song passed in. In my demo, I'm looking at villagers whose favorite song is "Go K.K. Rider"

```
# extract number of villagers who like song ___
def find_favorite_song(df, song):
    num = 0
    for r in df.index:
        if df['Favorite Song'][r] == song:
            num += 1
    return num
num = find_favorite_song(villagerDF,"Go K.K. Rider")
print("Num of villagers whose favorite song is 'Go. K.K. Rider' - ", num)
```

```
Num of villagers whose favorite song is 'Go. K.K. Rider' -  5
```



Task W5

Differential privacy is the act of publishing insights from a data set without depending too heavily on any one user. In our data currently, D, villager SGMdki6dzpDZyXAw5 likes the song “Go K.K. Rider.” In an imaginary data set, D’, this villager does not like the song and instead likes “K.K. Soul.” When the same query is performed above on D’, the result returned will be 4. This means that changing one row has immediate impact on what the information extracted from the data is. If the number was 5, the data was drawn from D. If the number was 4, the data data was drawn from D’. For differential privacy, the attacker should not be able to discern whether the returned result was from D or D’. In simpler words, our villager’s favorite song should not impact the query. An adversary would be able to use this knowledge to his/her advantage. If they know their target, X, likes “K.K. Rider,” and let’s assume they also have some knowledge on the other villagers – including their favorite song, they would be able to count the number of villagers total who like this song. If target x is represented in the data, they would have high confidence that data D would contain their target. If target x is not represented in the data, they would not have confidence that D would contain their target and therefore the data would be D’. In essence, an attacker would be able to tell if their target was present in the data or not, which could be used to their advantage to break the k-anonymity algorithm and determine sensitive information based of the QIs.

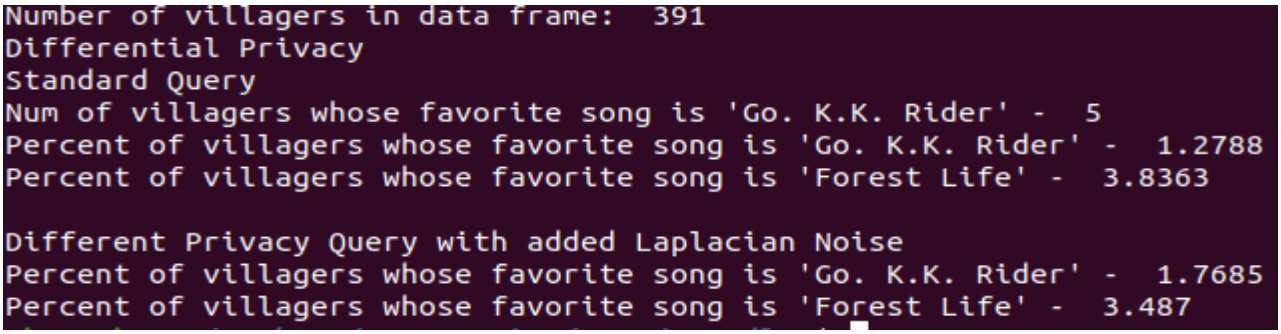
Another example would be if the attacker wants to know if their target likes “K.K. Rider.” The attacker has information on all the other villagers and knows how many like the song, in this example case it would be 4 total. So, if the attacker looks at the data and sees 5 people like the song, they know their target also likes the song, which means sensitive information has just be leaked.

Task C6

A simple solution to obfuscate data D and D' is to add random noise sampled from the Laplacian distribution. I used the numpy random gen package in python to obtain the noise **(3)**. In this query, a single villager can, at most, impact the result by 1. They either like the song or don't. Because of this, in the laplace() formula, the scale will be set at 1. This means the amount of noise added will be just from one sample. To calculate the center of the distribution, the number of villagers total who like the song is calculated first using the standard query above. Finally, the total + noise is converted to a percentage using the total number of villagers present in the data.

```
# implementing differential privacy for favorite song by adding Laplacian Noise
def find_favorite_song_diffpriv(df, song):
    # add laplacian noise using Python random generator
    rng = default_rng()
    # find total number of villagers with this favorite song to center the distr
    mean = find_favorite_song(df, song)
    if mean == -1:
        return -1
    # add noise
    noise = rng.laplace(mean, 1)
    # turn into percentage
    percentage = round(((noise / num_villagers) * 100), 4)
    return percentage
```

Task W7



In this example I chose to represent the differential privacy query using percentages since the song preference values are low in the data and this can better represent how the Laplacian Noise impacts the output. An adversary's knowledge doesn't change with the outcome because they can still calculate the total number because they know the number of villagers in the data set. From the example above, we see that the actually percentage of villagers who like 'Go K.K. Rider' is about 1.2788% while the different privacy outcome had it at 1.3014%. With this outcome, the attacker must decide which is a more probable outcome if the actual total was 5 or 4 (k, k-1). But due to the unique factor of the Laplacian Distribution, the probability the noise was added to 5 or 4 would be the same and therefore the attacker cannot discern whether their target was in the data or not. Now the privacy of the sensitive information is now preserved from an attacker and an attacker can no longer use previous known knowledge to determine something. The utility of this output has decreased because it is no longer the original value and contains noise. This added noise might be a problem if the sensitive data were used for training a model. Each time the code is ran, the differential privacy query percentage changes due to the random sampling of noise. So assuming Python's random number generator is secure, then we can safely say the sensitive information is secure.

Verifying the output above:
True value = 5
Value + noise = (1.7685 / 100) * 391 ≈ 6.9 rounded up to = 7

What is the likelihood the original number was 4 (our target does not like 'K.K. Rider') vs 5 (our target likes the 'K.K. Rider')? In our case, e-differential privacy, we picked a random value using Laplace(1/e), and behind the logic of probabilities for differential privacy, you cannot determine this **(4)**.

Links

1. https://www.kaggle.com/datasets/jessicali9530/animal-crossing-new-horizons-nookplaza-dataset?resource=download
2. https://pypi.org/project/kmodes/
3. https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.laplace.html
4. https://desfontain.es/privacy/differential-privacy-in-practice.html