Differential Privacy Temporal Map Challenge (DeID2)-Sprint 3

Title: TaxiTrip - Synthesizer

Team: GooseDP-PSA3

NOTE: The three sections (**Background**, **Our Algorithm**, **Privacy Discussion**), discuss at a high level these aspects of our approach. A more formal description is provided in section **Formal Approach**, which follows the outline from: `https://community.drivendata.org/t/important-regarding-final-submission-write-ups/6131`.

Main Idea

To create usable private data while guaranteeing data contributor privacy, we have designed a synthetic data generation pipeline which seeks to maintain key data characteristics while only making low-sensitivity queries to private records. To do this, we construct profiles of representative record distributions, which we call *archetypes*, from the public data. Using these archetypes, in addition to a small number of private queries, we can construct a set of partial synthetic records. With additional pre and post processing using the public data, as well as domain knowledge, we can create a complete synthetic dataset with the missing attribute values.

Background

To protect the privacy of the data contributors, we use user level (ϵ, δ)-differential privacy to measure the privacy loss. Informally, a database D is *differentially private* [2, 3] towards a randomized algorithm $\mathcal{M}$ if changing one element of the database does not significantly change the output of the mechanism. For user level differential privacy, changing the contributions of one user must not significantly change the output of the mechanism.

Definition 1. (ϵ, δ)-Differential Privacy
A randomized algorithm $\mathcal{M}$ is (ϵ, δ)-differentially private if for all $\mathcal{S} \in \text{Range}(\mathcal{M})$:

$$Pr\left[\mathcal{M}(D) \in \mathcal{S}\right] \leq e^{\epsilon} Pr\left[\mathcal{M}(D') \in \mathcal{S}\right] + \delta$$

for any pair of neighbouring databases D, D' that differ by the records from a single data contributor.

In this work, we define neighbouring databases D, D' as those which differ by the addition or subtraction of a single data contributor.

The sensitivity of a differentially private algorithm is related to the maximum impact an individual record or user can have on the output. The user-level sensitivity of creating a histogram is equal to the number of histogram bins that can be changed in the worst case when one individual's records are changed.

Definition 2. Sensitivity
The sensitivity over a set of functions F is defined as

$$S(F) = \max ||F(D) - F(D')||_2$$

where D and D' are two datasets that differ by adding/removing one individual's records and $||.||_2$ is the ℓ_2-norm of the matrix.

One way to achieve differential privacy is to use the Analytic Gaussian Mechanism [1].

Definition 3. Analytic Gaussian Mechanism
Let $f : \mathbb{N}^{|\mathcal{X}|} \to \mathbb{R}^d$ be an arbitrary d-dimensional function, and define the ℓ_2 to be $\Delta_2 f = \max_{\text{adjacent } x,y} ||f(x) - f(y)||_2$. The Gaussian mechanism $\mathcal{M}(x) = f(x) + Z$ where $Z \sim N\left(0, \sigma^2\right)$ is (ϵ, δ)-DP if and only if

$$\Phi\left(\frac{\Delta_2 f}{2\sigma} - \frac{\epsilon\sigma}{\Delta_2 f}\right) - e^{\epsilon}\Phi\left(-\frac{\Delta_2 f}{2\sigma} - \frac{\epsilon\sigma}{\Delta_2 f}\right) \leq \delta.$$

[1] gives an algorithm for calculating σ for a given $\epsilon, \delta, \Delta_2 f$, which we use in our code.

Our Algorithm: TaxiTrip - Synthesizer

A block diagram of our approach is shown in Figure 1. It shows the three steps in the pipeline, as well as at which steps queries are made to the public and private data.

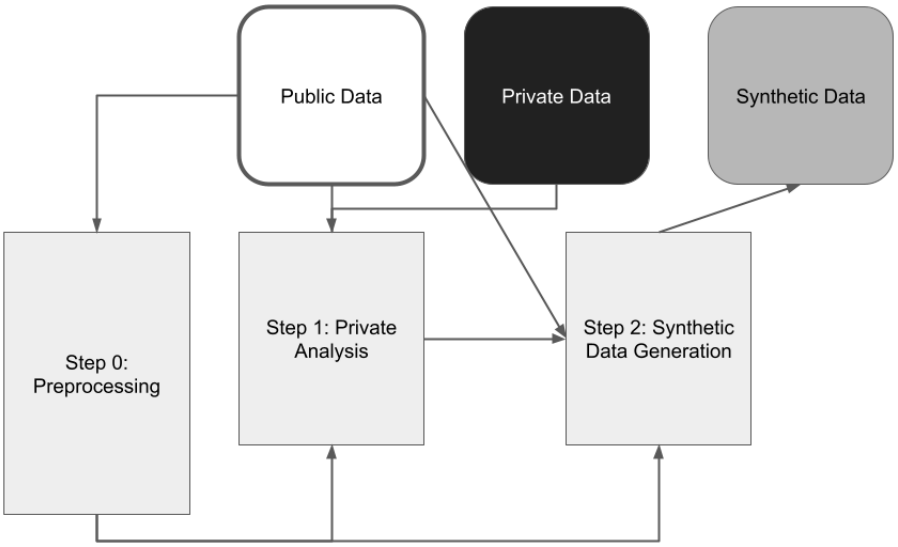

Figure 1: A Block Diagram of the Data Synthesis Pipeline

Step 0: Preprocessing
Archetype Generation. The first step of our pipeline is creating a set of representative profiles, called archetypes, for groups of similar individuals in the data. In terms of `taxi_ids`, these can be thought of as groups of similar drivers such as *airport drivers* or *daytime downtown drivers*. They are used as templates from which individual synthetic `taxi_ids` may be constructed.

In our submission, the archetypes are constructed using a clustering technique called Gaussian Mixture Models [4], on the public data set. We found experimentally that using 10 clusters would allow our pipeline to get stronger results. These archetypes could then be improved using refining techniques and domain knowledge, but our submission does not make use of these.

Each archetype consists of the 3-way marginals of (`shift`, `pickup_community_area`, `dropoff_community_area`) and the total number of taxis and taxi rides associated with it in the public data. We also generate a set of weights for a classification function, which will be able to label a record as belonging to an archetype. As well, a csv file storing the 3-way marginals distribution of (`shift`, `pickup_community_area`, `dropoff_community_area`) per archetype on the public data is saved, so it does not need to be calculated during processing.

Step 1: Private Analysis
Our private analysis consists of two histogram releases from the private data.

For the first, we associate each `taxi_id` in the private data with an archetype and count the number of `taxi_ids` associated with each archetype. Creating this histogram is a sensitivity 1 query with respect to our unit of privacy analysis, `taxi_id`, so we then privatize these counts with the Gaussian mechanism using privacy parameters ϵ_1, δ_1.

For the second, we start by associating each `taxi_id` in the private data with a single `company_id`. We do this by choosing the most common `company_id` for the trips of each `taxi_id`, breaking ties at random. We then construct a histogram yielding the number of unique `taxi_id` associated with each `company_id`. As before, this histogram is a sensitivity 1 query with respect to `taxi_id` because we ensure that each `taxi_id` is associated with exactly 1 `company_id`. We make sure to include every `company_id` from *parameters.json* in the histogram domain, so `company_ids` from *parameters.json* that do not appear in the private data have a count of 0 in our histogram. Finally, we privatize these counts with the Gaussian mechanism using privacy parameters ϵ_2, δ_2.

Appealing to basic composition of (ϵ, δ)-DP mechanisms, we say that the composition of these two releases respects $(\epsilon_1 + \epsilon_2, \delta_1 + \delta_2)$-DP.

Step 2: Synthetic Data Generation
The goal of Step 2 is to synthesize the remaining columns of the data records by sampling from the archetypes generated in Step 0 and making use of the differentially private counts queried in Step 1. The generated columns in this step are `taxi_id`, `company_id`, `shift`, `pickup_community_area`, and `dropoff_community_area`.

For each taxi archetype, we create a number of simulated `taxi_id`'s equal to the private count of the size of that archetype from Step 1. Each of the `taxi_id`'s are then assigned a number of trip records proportional to the expected number of records a taxi in that archetype takes in the public dataset. For faster computation, we limit the total number of trips to 2 million and scale down the counts from each archetypes proportionately based on total public count. A record consists of a `shift`, `pickup_community_area`, `dropoff_community_area` triplet sampled from its corresponding taxi's archetype. The values for these simulated trip records are sampled from the 3-way marginals of the three attributes provided by the archetype files from Step 0. Finally, for each record, we sample a `company_id` from the private company counts from Step 1 and assign it that value. A 5-column synthetic dataset is then the output of this step.
Synthesize Remaining Columns.
In this step, we complete synthesizing all other columns (`fare`, `tips`, `trip_total`, `trip_seconds`, `trip_miles`, `payment_type`). We do so row-wise by leveraging the relationships learnt from the public data. We use public k-way marginals to fill up these columns. In Table 1, we describe the combination of columns used to calculate each target marginal.

Target column	Attributes for marginal calculation
`fare`	`shift`, `pickup_community_area`, `dropoff_community_area`
`tips`	`shift`, `pickup_community_area`, `dropoff_community_area`, `fare`
`trip_total`	`fare`, `tips`
`trip_seconds`	`shift`, `pickup_community_area`, `dropoff_community_area`, `fare`
`trip_miles`	`shift`, `pickup_community_area`, `dropoff_community_area`, `fare`
`payment_type`	`fare`, `tips`, `trip_total`

Table 1: Description of k-way marginal calculation

Rounding. Finally, we round the values that are floating point numbers into their nearest integer values and verify that the output value is in the domain of that attribute.

Privacy Discussion

In our pipeline, the only step that makes queries to the private data is Step 1. It makes two histogram queries: one for archetype counts and one for `company_id` counts. We divide the overall privacy budget between these two histograms. In our current design, the archetype count histogram query receives a proportion of 0.1 of the budget in both the ϵ and δ parameters. That is, we give a budget of $(\epsilon_1, \delta_1) = (0.1\epsilon, 0.1\delta)$ to the archetype counting query and $(\epsilon_2, \delta_2) = (0.9\epsilon, 0.9\delta)$ to the `company_id` counting query. As mentioned in **Step 1: Private Analysis**, we then appeal to basic composition and see that the composition of these processes respects (ϵ, δ)-DP.

The sensitivity of the archetype counting histogram is 1, as we ensure that `taxi_id` can only belong to one archetype. The sensitivity of the `company_id` counting histogram is also 1, as we only count the company that is most frequently associated with each `taxi_id` and thus ensure that they can only contribute to the overall count once. We can then use sequential composition, as the two private histogram counting queries are constructed with independent noise to achieve overall $(\epsilon_1 + \epsilon_2, \delta_1 + \delta_2)$-DP.

It is also important to note the trade-off between the number of archetypes and the number of private queries required by the pipeline. If we increase the number of archetypes we consider, we need to make more private queries but those archetypes will be more representative of ground truth `taxi_ids`. Conversely, if we have only a few archetypes we make only a few private queries but the archetypes are less accurate. Thus the utility of our pipeline relies on our ability to find a small amount of highly representative archetypes in the public data set, and the assumption that these archetypes are still present in the private data set.

Finally, we note that we are using the Gaussian mechanism from [3], which is defined only for $\epsilon \in (0, 1)$. Thus, we always use a functional $\epsilon' = \min(\epsilon, 0.9999999999)$ for our privatization. This is done in line 18 of the file `Archetype_Counting/archetype_company_counts.py`

Formal Approach

Pre-processing

Archetype Generation

As a pre-processing step, our approach requires the generation of archetypes from a public data set. To do this, we determine clusters of `taxi_id`s in the public data set and store their representative distributions. We call these the archetypes. The high level flow of the archetype generation pre-processing is as follows:

1. Determine k archetypes by clustering the rows in the public data on key attributes
2. Assign each `taxi_id` in the public data to an archetype
3. Calculate the expected distributions of attributes (1-way marginals) within these public archetypes and store the results to files
4. Save a model that predicts an archetype label for a given row (taxi trip)
5. Calculate the key 3-way marginal distributions within each archetype and save those distributions to another file

We generate these archetypes using a clustering technique called Gaussian Mixture Models (GMM) [4]. This works by assuming that all data values are generated from a combination of k Gaussian distributions, over the selected attributes. The technique then groups the data points to find the centres of these distributions. Unlike the k-means approach, the clusters may end up non-uniform elliptical which allows for more generality in their shape. In our submission, we make use of the `scikit-learn` implementation of Gaussian Mixture Models `https://scikit-learn.org/stable/modules/mixture.html`.

The three attributes from the row that we consider during the clustering are `shift, pickup_location, drop_off_location`. Thus each row is grouped into one of $k = 10$ clusters. We can see the code used to determine these clusters in the following code block:

```
df = pd.read_csv("ground_truth.csv")
data = df.to_numpy()
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
gmm = GaussianMixture(n_components=num_clusters)
gmm.fit(scaled_features)
```

To assign `taxi_id`'s to an archetype, we first count the archetype labels of each of the rows associated with a `taxi_id`. Each `taxi_id` is then assigned to the archetype which contains the largest amount of its rows. This process is shown in the following code block:

```
for row in data:
        if row[0] != taxi_id and count > 0:
                taxi_count = taxi_count + 1
                # find the cluster this taxi belongs to
                taxi_cluster = np.argmax(cluster_counts)
                cluster_lists[taxi_cluster].append(taxi_id)
                # clear counters for next taxi
                count = 0
                cluster_counts = np.zeros(num_clusters)
                taxi_id = row[0]
        #Determine the cluster belonging to the next trip
        cluster_counts[labels[overall_count]] += 1
        count += 1
        overall_count = overall_count + 1
```

For each archetype, we generate an archetype file (which is a `.txt` file). This contains the expected number of `shift`s per shift bin ,trips to each `pickup_location` and `drop_off_location` an average taxi in this archetype will take. They also contain the `taxi_id` s of all taxis in the public data that are associated with that archetype, the number of taxis and rows associated with each archetype, and the distribution of `payment_type` for trips in this archetype in the public data. The last line is not used in our current implementation.

As part of this pre-processing step, we also create a predictive model based on the GMM clusters, and save this as a set of weights to a `.pkl` file. This way, the model can be loaded into future files. The model generation step is shown in the following code block:

```
labels = gmm.predict(scaled_features)
pkl_filename = os.path.join(res_dir, "archetypes.pkl")
with open(pkl_filename, 'wb') as file:
    pickle.dump(gmm, file)
```

We also determine the distribution of the `shift, pickup_location, drop_off_location` 3-way marginals per archetype in the public data. To do so, we assign each row in the public data set an the archetype label associated with that row's `taxi_id`. Then we determine the distributions of the 3-way marginals in that archetype, and save those values to an additional `.csv` file. We can see this step in the following code block:

```
archetype_probs = df.groupby('archetype').size().div(len(df)
within_archetype_3way_marginals = df.groupby(['archetype', 'shift',
'pickup_community_area','dropoff_community_area']).size().div(len(df))
.div(archetype_probs, axis=0, level='archetype').reset_index()
# Print marginal distributions to a new file
within_archetype_3way_marginals.to_csv(os.path.join(res_dir, 'archetype_marginals.csv'))
```

Pre-processing corresponds to the first step of our data-synthesis pipeline. The files associated with this step are found in the `Archetype_Generation` folder in our submission. The file used to generate these files is named `k_archetypes.py` and the set of archetype files, indexed by k value are found in the `Results_GMM` folder. We generated many sets of archetype files before our experimental evaluation suggested that $k = 10$ would likely give us the best results.

Privatization

Our approach makes two private queries: the number of taxis per archetype and the number of taxis per `company_id` in the private data. We give a budget of $(\epsilon_1, \delta_1) = (0.1\epsilon, 0.1\delta)$ to the archetype counting query and $(\epsilon_2, \delta_2) = (0.9\epsilon, 0.9\delta)$ to the `company_id` counting query.

As a reminder, we are using the Gaussian mechanism from [3] to answer our private queries, which is defined only for $\epsilon \in (0, 1)$. Thus, we always use a functional $\epsilon' = \min(\epsilon, 0.9999999999)$ for our privatization. This is done in line 18 of the file `Archetype_Counting/archetype_company_counts.py`

Archetype Counting
We start by using the Gaussian Mixture Model trained on public data to predict the archetypes associated with each trip in our private data. We then assign each `taxi_id` to an archetype by identifying the most common predicted archetype among the trips belonging to the `taxi_id`. Each `taxi_id` is assigned to only 1 archetype. These assignments are not returned, but are used in future computation steps. This takes place in lines 53-65 of `Archetype_Counting/archetype_company_counts.py`.

```
priv_data = pd.read_csv(priv_data_path)
features = priv_data[ ['shift', 'pickup_community_area',
'dropoff_community_area'] ].to_numpy()

# load gmm
with open(os.path.join(archetype_path, 'archetypes.pkl'), 'rb') as f:
    gmm = pickle.load(f)

# predict archetypes for private data
priv_data_archetypes = pd.Series(gmm.predict(features))

# get most likely archetype for each taxi_id
priv_data['archetype'] = priv_data_archetypes
taxi_id_archetypes = priv_data.groupby('taxi_id')['archetype'].agg(lambda x:
x.value_counts().index[0])
```

We then get the private counts for each archetype and make sure that we include each of our possible 10 archetypes in our counts vector, even if it was never predicted. For this histogram counting query, let the bin labels be equal to the set of archetypes : $bin\_labels = \{archetype\_0, archetype\_1, \ldots, archetype\_k\}$. Let x be a database instance. Let $f_{bin\_label}(x)$ be a function that returns a histogram count of the number of `taxi_id` associated with each `bin_label`. We return $\max(\mathcal{M}_{\text{Gauss}}(x, f_{bin\_label}, \epsilon_1, \delta_1), 0)$, which adds proportional noise using the Gaussian mechanism, and projects negative private counts to 0. We can apply this in parallel to each x, as any `taxi_id` can only belong to one x.

This takes place in lines 67-82 of
`Archetype_Counting/archetype_company_counts.py`.

```
'''get archetype counts'''
priv_data_archetype_counts = taxi_id_archetypes.value_counts(sort = False)
for arch in archetype_dict.keys():
    if arch not in priv_data_archetype_counts.index:
       priv_data_archetype_counts = priv_data_archetype_counts.append(
       pd.Series([0], index = [arch]) )
priv_data_archetype_counts = priv_data_archetype_counts.sort_index()
priv_data_archetype_counts.to_csv(os.path.join(private_counts_dir,
f'nonprivate_data_archetype_counts_{epsilon}.csv'), index = False)

'''get DP archetype counts'''
final_priv_archetype_counts = priv_data_archetype_counts + np.random.normal(0,
archetype_sigma, len(archetype_dict.keys()))
final_priv_archetype_counts = np.maximum(0, final_priv_archetype_counts)
priv_count_df = pd.DataFrame({'archetype':
list(range(len(priv_data_archetype_counts))),
                              'count': final_priv_archetype_counts})
priv_count_df.sort_values('archetype', axis = 0)
priv_count_df.reset_index(drop = True, inplace = True)
priv_count_df.to_csv(os.path.join(private_counts_dir,
f'private_data_archetype_counts_{epsilon}.csv'), index = False)
```

The sensitivity of this query is 1. First note that our predictive model was trained on the public data, and thus its properties are completely independent of the private data. We use it to assign each `taxi_id` to an archetype, and then make a histogram query over the set of possible archetypes. Adding or removing any particular `taxi_id` can change the count of at most one histogram bin by at most one.

For a given overall (ϵ, δ), we $\frac{1}{10}$ of each as the privacy budget for this step.

Company ID Counting

We calculate the 1-way marginal for the `company_id` column in the private dataset. We do so by first, associating each `taxi_id` in the private dataset to only one company. Then, we group it based on `taxi_id` and assign it to its most likely company breaking ties at random. These assignments are never returned, but only used in calculations. We then calculate the 1-way marginal. This happens on lines 90-96 of `Archetype_Counting/archetype_company_count`

```
print('getting private company ID counts')
# load company IDs
with open(parameters_path, 'r') as f:
    parameters_dict = json.loads(pathlib.Path(parameters_path).read_text())
company_id_vals = list(parameters_dict['schema']['company_id'].values())[2]

# get most common company ID for each taxi ID, then merge onto original
# private data
top_company_id_modes = priv_data.groupby(['taxi_id'])['company_id']
.agg(lambda x: np.random.choice(pd.Series.mode(x))) # could be multimodal
company_id_counts = top_company_id_modes.value_counts(sort = False)
for id_val in company_id_vals:
    if id_val not in company_id_counts.index:
        id_val_series = pd.Series([0], index = [id_val])
        company_id_counts = company_id_counts.append( id_val_series )
```

Next, we privatize this 1-way marginal by adding Gaussian Noise. As each `company_id` is associated with 1 taxi, the sensitivity is 1. For a given overall (ϵ, δ), we assign $\frac{9}{10}$ of the total budget.

For this histogram counting query, let the bin labels be equal to the set of `company_id` : $bin\_labels = \{0, 1, \ldots, m\}$, where m is the maximum domain value for `company_id`'s retrieved from `parameters.json` . Let x be a database instance. Let $f_{bin\_label,k}(x)$ be a function that returns a histogram count of the number of `taxi_id` associated with each `bin_label` in archetype k. We return $\max(\mathcal{M}_{\text{Gauss}}(x, f_{bin\_label,k}, \epsilon_2, \delta_2), 0)$, which

adds proportional noise using the Gaussian mechanism, and projects negative private counts to 0. We can apply this in parallel to each x, as any `taxi_id` can only belong to one x.

Lines 98 - 105 of `Archetype_Counting/archetype_company_counts.py` privatize the marginal:

```
# privatize counts
priv_company_id_counts = company_id_counts + np.random.normal(0,
company_sigma, len(company_id_counts))
priv_company_id_counts = np.maximum(0, priv_company_id_counts)
priv_company_count_df = pd.DataFrame({'company_id':
priv_company_id_counts.index,
                                      'count': priv_company_id_counts})
priv_company_count_df.sort_values('company_id', axis = 0, inplace = True)
priv_company_count_df.reset_index(drop = True, inplace = True)
priv_company_count_df.to_csv(os.path.join(private_counts_dir,
f'private_data_company_id_counts_{epsilon}.csv'), index = False)
```

Privacy Budget Accounting
We have two private queries in our pipeline: one for archetypes and one for `company_id`. We are allotting $\frac{1}{10}$ and $\frac{9}{10}$ of our ϵ and δ to each, which by basic composition gives us an (ϵ, δ) overall guarantee.

Post-Processing

Sampling Taxi-Trips

We start to generate synthetic data by sampling taxi-trips records, i.e. the columns (`shift`, `pickup_community_area`, `dropoff_community_area`). Then in this step, we associate `company_id` and `taxi_id` with each record. This step takes place in the code file `Step2_sample_triplets/sample_triplets.py`.

This step is only makes use of the (`shift`, `pickup_community_area`, `dropoff_community_area`) 3-way marginal information within each archetype from the public data and the differentially private counts from the privatization step. This step is post-processing since it never looks at the private data. Therefore, we first load the marginal information and the private counts, which takes place in line 53-59:

```
three_way_marginal_public =
    pd.read_csv(os.path.join(archetype_path, f'archetype_marginals.csv'))

private_counts_company =
    pd.read_csv(os.path.join(private_count_path,
            f"private_data_company_id_counts_{epsilon}.csv"))

# get weights from private company counts
private_counts_company_weights = private_counts_company['count'].tolist()
```

Then for each archetype i, we build the 3-way marginal dictionary for sampling where the keys of the dictionary are all combinations of (`shift`, `pickup_community_area`, `dropoff_community_area`) within that archetype and the value corresponding to each key is the probability that the combination appears in that archetype in public data. This part takes place in line 84-90:

```
three_way_marginal_public_cur_archetype =
        three_way_marginal_public[three_way_marginal_public['archetype']==i]
        .reset_index()
keys = list(zip(three_way_marginal_public_cur_archetype['shift'],
        three_way_marginal_public_cur_archetype['pickup_community_area'],
        three_way_marginal_public_cur_archetype['dropoff_community_area']))

three_way_marginal_dict = dict()
for j in range(three_way_marginal_public_cur_archetype.shape[0]):
    three_way_marginal_dict[keys[j]] = three_way_marginal_public_cur_archetype.loc[j,'0']
```

Then we sample the columns (`shift`, `pickup_community_area`, `dropoff_community_area`) using the marginal dictionary, which is in line with code from line 94-98.

```
sampled_triplets = random.choices(population=list(three_way_marginal_dict.keys()),
                                  weights=three_way_marginal_dict.values(),
```

```
                                        k=int(private_taxicount * private_ride_per_taxi))

    shift, pickup, dropoff = map(list,zip(*sampled_triplets))
```

The private counts of `company_id` give the noisy one-way distribution of that attribute in the private data. Therefore, we sample `company_id` according to this noisy one-way distribution, which takes place in line 101-103. Next, we assign `taxi_id`'s to each record in lines 112-117. To do so, we calculate the average number of trips per taxi in each archetype from the public data and assign this many trips to each `taxi_id`. We choose the number of `taxi_ids` per archetype in our synthetic data by scaling down the private `taxi_id` counts until this strategy gives us our desired number of synthetic records (approximately 2 million).

```
    company_id = random.choices(population = company_id_domain,
                                        weights = private_counts_company_weights,
                                        k = int(private_taxicount * private_ride_per_taxi))
    current_archetype_data = np.array(current_archetype_data).T
    taxi_ids = np.arange(first_taxi_id, first_taxi_id + private_taxicount)
    taxi_id_array = np.array([ [int(t)] * int(private_ride_per_taxi) for t in taxi_ids])
                            .flatten()
    current_archetype_data = np.column_stack([taxi_id_array, current_archetype_data])
    archetype_data_list.append(current_archetype_data)
    first_taxi_id += private_taxicount
```

Public K-Way Marginals
As described in Step 3 of our algorithm, as well as Table 1, we synthesize the columns (`fare`, `tips`, `trip_total`, `trip_seconds`, `trip_miles`, `payment_type`) directly from k-way marginals in the public data. We do so sequentially in the order in which the columns appear above.

We look at `tips` as an illustrative example. First note that, by the time we are synthesizing `tips`, we have already synthesized (`shift`, `pickup_community_area`, `dropoff_community_area`, `company_id`, `fare`), and thus can use these to inform our generation of `tips`. For an arbitrary element in our synthetic data with index i, say we have (`shift`$_i$ = a, `pickup_community_area`$_i$ = b, `dropoff_community_area`$_i$ = c, `company_id`$_i$ = d, `fare`$_i$ = e). We'll ignore the `company_id` information, as we know from table 1 that it is not used in the calculation of `tips`. We then subset our public data to the set of records with (`shift` = a, `pickup_community_area` = b, `dropoff_community_area` = c, `fare`$_i$ = e) and construct an empirical probability probability mass function within the subset for `tips`. We then sample from this distribution to get our synthetic value for `tips`$_i$.

This process is carried out in the `Step3_nonprivate_gen` folder, in the `cc_post_col_generation` file. The `post_col_generation` function constructs dictionaries where the keys are identifying information about how to subset the public data and the values are the empirical probability mass functions over the variable of interest within each subset. It then assigns values to elements in the synthetic via dictionary lookup and sampling from the empirical probability mass function. The function always attempts to sample over the subset defined by the values in the columns defined in table 1. If the particular set of values does not exist in the public data, the function then defaults to using only the columns (`shift`, `pickup_community_area`, `dropoff_community_area`).

It is important to note that all of this sampling happens on the public data, with subsets defined by values synthesized from private counts of (`shift`, `pickup_community_area`, `dropoff_community_area`). Thus, it incurs no privacy loss.

References

[1] B. Balle and Y.-X. Wang. Improving the gaussian mechanism for differential privacy: Analytical calibration and optimal denoising. In *International Conference on Machine Learning*, pages 394–403. PMLR, 2018.

[2] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006.

[3] C. Dwork, A. Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[4] D. A. Reynolds. Gaussian mixture models. *Encyclopedia of biometrics*, 741:659–663, 2009.