APPENDIX A
Solutions to Exercises

Chapter 1

1. Assuming an individual can contribute at most one record, which of the following database pairs are adjacent? In the case where a user may add one record, datasets differ by the number of additions and removals necessary to convert one dataset to another.
   a. Adjacent - two datasets are considered adjacent so long as their distance is *less than or equal to* the max number of contributions (in this case, one). The distance between these datasets is zero.
   b. Not adjacent - the distance between these datasets is two: the removal of B and addition of D.
   c. Adjacent - they differ only by C, assuming insensitivity to order
   d. Adjacent - they differ only by G, assuming insensitivity to order
   e. Not adjacent - two instances of A are added
   f. Adjacent - again, the dataset distance is zero. The types of the instances don't matter, so long as they can be compared for equality.

If you instead consider data sets adjacent by *changing* one record, as was covered in the chapter, then .. Adjacent - it takes zero edits to translate X into Y, so the dataset distance is zero. .. Adjacent - assuming insensitivity to ordering, the two datasets differ only by the change of B to D .. Not adjacent - the distance between these datasets is not well defined. No number of changes can convert X into Y. .. Not adjacent - similarly, the datasets have different sizes, so are not comparable by changes. .. Not adjacent - by similar logic .. Adjacent - dataset distance is zero

2. `COUNT(x)` is implemented as `len(x)` in Python.

   a. Create the test data set via `range`:

   ```
   x = range(0,11)
   print(len(x))
   ```

   b. Compute the distance between counts on all adjacent data sets with one element removed:

   ```
   for i, element in enumerate(x):
       y = x[:i] + x[i+1:]
       print(len(y))
       print('distance:', abs(len(y)-len(x)))
   ```

   c. For notational convenience, let `f(x) = COUNT(x)`. The global sensitivity of the count is the smallest number d such that $|f(x) - f(x')| \leq d$ for any pair of adjacent data sets x, x'.

   x and x' may differ by either removing one row or adding one row. First construct x' by *removing* any single value from x. If we assume $f(x) = n$, then $f(x') = n - 1$. Therefore $|f(x) - f(x')| = |n - (n-1)| = 1$, for any data set that differs by a removal.

   Then construct x' by *adding* any single value from x, so that $f(x') = n + 1$. Therefore $|f(x) - f(x')| = |n - (n+1)| = 1$ for any data set that differs by an addition.

   Therefore, for any choice of adjacent data sets, the sensitivity of `COUNT(x)` when an individual contributes at most one record is one.

3. Consider the "worst case" scenario: you have a data set $[0, 0, 0 \ldots . 0, 110]$ of size 10 with mean $\frac{110}{10} = 11$. One adjacent data set is $[0, 0, 0 \ldots . 0]$ of size 10 and mean 0. The difference in these means is 11.

   a. Using this fact, and knowing the data is bounded on $[0, 110]$, calculate the global sensitivity:

   $$\left|\overline{x} - \overline{x'}\right| = \left|\frac{110}{10}\right| = 11$$

   b. As the maximum score increases, the global sensitivity increases linearly:

   $$\frac{120}{10}, \frac{130}{10}, \frac{140}{10}, \ldots = 12, 13, 14, \ldots$$

   c. No, the sensitivity approaches ∞ as the maximum value increases.

4. Consider a data set x of n elements where each element is in $[0, U]$ and an adjacent data set x' with one changed element. Then the absolute value of the difference in their means is

$$\left|\bar{x} - \overline{x'}\right| = \left|\frac{1}{n}\left(\sum_{i=1}^{n-1} x_i + x_n\right) - \frac{1}{n}\left(\sum_{i=1}^{n-1} x_i + x'_n\right)\right|$$
$$= \frac{1}{n}\left|x_n - x'_n\right|$$

Since all $x_i \in [0, U]$, we know that $|x_n - x'_n|$ is no greater than U. Therefore the sensitivity is:

$$\max_{x \sim x'} \left|\bar{x} - \overline{x'}\right| = \frac{U}{n}$$

5. Continuing the statement from the previous exercise. Since all $x_i \in [L, U]$, we know that $|x_n - x'_n|$ is no greater than $U - L$. Therefore the sensitivity is:

$$\max_{x \sim x'} \left|\bar{x} - \overline{x'}\right| = \frac{U - L}{n}$$

Of note, the sensitivity is undefined when datasets may differ in the number of records, and an adjacent dataset may be empty.

Chapter 2

1. A binary array is an array where each element is either 0 or 1.
   a. Using the definition of global sensitivity, let's first work out the sensitivity under the assumption that neighboring datasets differ by the addition or removal of one record. First consider, without loss of generality, the case where one record (x_{n+1}) is added to the end of x.

$$\max_{x \sim x'} \left|f(x) - f(x')\right| = \max_{x \sim x'} \left|\sum_{i=1}^{n} 2 \cdot x_i + 1 - \sum_{i=1}^{n+1} 2 \cdot x_i - 1\right| = \max_{x \sim x'} 2 \cdot x_{n+1}$$

Since $x_{n+1} \in \{0, 1\}$, then $\max_{x \sim x'} |f(x) - f(x')| = 2$. By a similar logic, consider the case where the last record x_n is removed:

$$\max_{x \sim x'} |f(x) - f(x')| = \max_{x \sim x'} \left| \sum_{i=1}^{n} 2 \cdot x_i + 1 - \sum_{i=1}^{n-1} 2 \cdot x_i - 1 \right| = \max_{x \sim x'} 2 \cdot x_n$$

Since $x_n \in \{0, 1\}$, then $\max_{x \sim x'} |f(x) - f(x')| = 2$. Together with the case where one record is added, the sensitivity of the sum when any one record is added or removed is $\max_{x \sim x'} |f(x) - f(x')| = \max(2, 2) = 2$

It is left to you to work out the sensitivity of $f(\cdot)$ when datasets differ by *changing* one record. In this setting it is useful to assume, without loss of generality, that only the last record is changed.

b. Following a similar line of reasoning from the last exercise, and under the assumption that neighboring datasets differ by the addition or removal of one record, first consider the case where one record is added:

$$\max_{x \sim x'} |f(x) - f(x')| = \max_{x \sim x'} \left(\sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n+1} x_i^2 \right) = \max_{x \sim x'} \left| -x_{n+1}^2 \right| \leq 1$$

Then consider the case where one record is removed:

$$\max_{x \sim x'} |f(x) - f(x')| = \max_{x \sim x'} \left(\sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n-1} x_i^2 \right) = \max_{x \sim x'} \left| x_n^2 \right| \leq 1$$

Therefore, the total sensitivity, assuming boolean data is one (the maximum of both cases).

2. $\epsilon = 0.1$ provides a stronger privacy guarantee - remember, a larger value of ϵ corresponds allows for greater privacy loss.

3. Using SmartNoise:

```
from snsql import from_connection, Privacy

csv = 'student-mat.csv'
student = pd.read_csv(csv)
privacy = Privacy(epsilon=0.1)
reader = from_df(student, metadata=None, privacy=privacy)

result = reader.execute('''SELECT COUNT(*) AS n
    FROM student WHERE school = "GP" ''')
```

Accordingly:

```
from snsql import from_connection, Privacy
```

```
csv = 'student-mat.csv'
student = pd.read_csv(csv)
privacy = Privacy(epsilon=1.0)
reader = from_df(student, metadata=None, privacy=privacy)

result = reader.execute('''SELECT COUNT(*) AS n
    FROM student WHERE school = "GP" GROUP BY family''')
```

4. Given that the mechanism M is ϵ-DP, we know that, for all adjacent data sets x and x', and any outcome z:

$$\Pr\left[M(x) = z\right] \leq \Pr\left[M(x') = z\right] \cdot e^{\epsilon}$$

Now consider the postprocessing $f(M(x)) = n \cdot M(x)$ for some n. Does $f(M(x))$ satisfy the conditions of ϵ-DP as stated above? Let's find out:

$$\Pr\left[f(M(x)) = z\right] = \Pr\left[n \cdot M(x) = z\right] = \Pr\left[M(x) = \frac{z}{n}\right]$$

Since we know that M satisfies ϵ-DP:

$$\Pr\left[M(x) = \frac{z}{n}\right] \leq Pr\left[M(x') = \frac{z}{n}\right] \cdot e^{\epsilon}$$

Therefore, $f(M(x)) = n \cdot M(x)$ satisfies ϵ-DP. This can be proven more generally for all functions $f(\cdot)$, which is what gives differential privacy immunity from postprocessing.

Chapter 3

1. This answer will vary depending on how you consider datasets to be neighboring.

   If you consider datasets to be neighboring under the addition or removal of an individual who may contribute up to d records, then the following function, written in Python, checks that the symmetric distance between x and x' is no greater than d:

```
def check_if_adjacent_Sym(x, x_p, d):
    """check if data sets x and x' differ
    by at most d wrt the symmmetric distance
    """
    return d_Sym(x, x_p) <= d
```

   This uses the function `d_Sym` from ???.

2. Show that metrics in this chapter (absolute distance, symmetric distance and Hamming distance) satisfy non-negativity, symmetry and the triangle inequality

   a. Non-negativity of the absolute distance follows from the definition of the absolute value. Since $x - y = -(y - x)$, symmetry also follows from the definition of the absolute value. For the triangle inequality, first consider any three points x, y and z. If y is in $[x, z]$, then $|x - z| = |x - y| + |y - z|$, strictly preserving the triangle inequality. If y is not in $[x, z]$, then \(|x - z| \lt |x - y| + |y - z|\).

   b. Non-negativity of the symmetric distance follows from the definition of cardinality, which is never negative. Symmetry of the symmetric distance follows from symmetry of the symmetric distance: the set of elements not in either set. For the triangle inequality, first consider any element $a \in |X \triangle Z|$ and any other set Y. If a is in Y, then a is either in $|X \triangle Y|$ or in $|Y \triangle Z|$ exactly once, strictly preserving the triangle inequality. If a is not in Y, then a is in both $|X \triangle Y|$ and $|Y \triangle Z|$, also preserving the triangle inequality.

   c. The Hamming distance is non-negative because the distance is computed as the sum of non-negative values. Symmetry of the Hamming distance follows from symmetry of the equality operator. For the triangle inequality, consider any three vectors X, Y and Z of length n, as well as any index $i \leq n$. If $X_i = Z_i$, then the distance cannot decrease. If $X_i \neq Z_i$, then either $X_i \neq Y_i$, $Y_i \neq Z_i$, or both, meaning the distance cannot decrease.

3. Recall that an aggregate converts a data set into a summary or statistic.

   a. General transformation

   b. Aggregate

   c. General transformation

   d. General transformation

4. This function is 2-stable under both the symmetric distance and Hamming distance. We'll take a closer look at the symmetric distance. Recall that the symmetric distance counts the number of elements that appear in one of two multisets, but not in both. Consider an example:

   Given $x = \{1, 2, 3\}$ and $x' = \{1, 2\}$, then `duplicate(x)` $= \{1, 2, 3, 1, 2, 3\}$ and `duplicate(x')` $= \{1, 2, 1, 2\}$. The symmetric difference between `duplicate(x)` and `duplicate(x')` is:

   `duplicate(x)` $\triangle$ `duplicate(x')` $= \{1, 2, 3, 1, 2, 3\} \triangle \{1, 2, 1, 2\} = \{3, 3\}$.

   The symmetric distance is the cardinality of this multiset: $|\{3, 3\}| = 2$.

Notice that it doesn't make sense to use the absolute distance metric, because you can't compute the absolute distance between two vectors.

5. Derive the sensitivity of the mean transformation, in the setting where the input data is bounded, the data set size is not known, but a lower bound M on the number of records in the data is known.

$$
\begin{aligned}
& \max_{x \sim x'} d_{Abs}(\text{mean}(x), \text{mean}(x')) && \text{by definition of stability} \\
= \; & \max_{x \sim x'} \left| \sum_{i=1}^{N} \frac{x_i}{N} - \sum_{i=1}^{N'} \frac{x'_i}{N'} \right| && \text{substitute } \text{mean}(x) = \sum_{i=1}^{N} \frac{x_i}{N} \\
\leq \; & \frac{1}{\max(N - b_{in}, 0)} \max_{x \sim x'} \left| \sum_{i=1}^{N} x_i - \sum_{i=1}^{N'} x'_i \right| && \text{factor out } \frac{1}{N} \\
= \; & \frac{b_{in}}{\max(N - b_{in}, 0)} \cdot \max(|L|, U) && \text{by previous sum sensitivity} \\
\leq \; & \frac{b_{in}}{\max(M - b_{in}, 0)} \cdot \max(|L|, U) && \text{since } N \geq M
\end{aligned}
$$

This sensitivity can be useful if you have a tight lower bound on the data set size. You could even use a preprocessing transformation that resizes the data to have at least M records, imputing if necessary.

6. Define $f(x) = COUNT(NODES(x))$. `NODES(x)` is a 1-stable function from a graph to a vector of nodes. The addition or removal of d nodes to the graph will always result in the addition or removal of d nodes to the vector of nodes. As previously shown, `COUNT(x)` is a 1-stable function from a vector to a scalar. Since `f(x)` is the functional composition of two 1-stable functions, it is also 1-stable.

7. The forward pass of neural networks are row-by-row when each of the layers are row-by-row. Many common layer types are row-by-row, like linear transformations, activation functions and convolutions. In this manner, inference on a sensitive dataset by a neural network, such as one used for sentiment analysis with row-by-row layers, is a 1-stable row-by-row transformation. This is general technique can allow you to apply DP to microdata from many new data domains (text, audio, video, etc). First use a neural network as a stable transformation of such data into a rectangular form, and then apply DP.

8. In order for a data set to be analyzed with differential privacy, you must know a limit on how much any one individual can influence a data set. If your data is tabular, you might know that each individual may only contribute or change

a limited number of records. If your data is aggregated counts, you might know that the counts may only differ by a limited amount. If your data is images, you might know that each individual may only contribute a limited number of images.

Differential privacy can be applied to any data set where the influence, or difference, on a data set by an individual can be bounded.

Chapter 4

1. First, construct a 1-stable transformation that rounds each sensitive answer to the nearest bound.

```
from random import uniform, randint
from statistics import mean

def make_randomized_round(bounds):
    dp.assert_features("contrib")
    domain, metric = dp.atom_domain(T=type(bounds[0])), dp.discrete_distance()
    return dp.t.make_user_transformation(
        domain, metric, domain, metric,
        function=lambda arg: bounds[int(uniform(*bounds) < arg)],
        stability_map=lambda b_in: b_in
    )
```

   This transformation is used to create a measurement for releasing either the lower or upper bound for each individual.

```
bounds = [0, 100]
rr_mean = make_randomized_round(bounds) \
          >> make_randomized_response_multi(.75, bounds)

# if you know the data is actually distributed evenly between bounds,
# the estimator is unbiased
mock_dataset = [randint(*bounds) for _ in range(10_000)]
priv_dataset = [rr_mean(x) for x in mock_dataset]

print(mean(mock_dataset))
print(mean(priv_dataset))
```

   The local-DP releases of bounds are then postprocessed into an estimate of the mean.

2. When writing a DP algorithm, you should put yourself in the mindset of an adversary. Say you were to set the allowed categories to $\{A, A, A, B\}$. Then $\Pr[M(A) = A] = p + (1-p) \cdot 2/3$, $\Pr[M(B) = A] = (1-p)/3$, and the resulting privacy loss would be $\ln\left(\frac{3p + 2(1-p)}{1-p}\right)$. Unfortunately, this is a higher privacy loss than what the mechanism promises: $\ln\left(\frac{3p}{1-p}\right)$. The privacy loss is too high

because the mechanism is too likely to return A, when an individual's response was A.

3. The global sensitivity of precision is undefined, just like the mean. The denominator goes to zero when all private predicted values are false, regardless of expected values: precision($[\perp, ..., \perp]$) = 0/0. A simple alternative approach is to privately release the count of true positives and privately release the count of false positives. We'll need to create a new preprocessing labeler transformation, and a postprocessor that computes the precision.

```
def make_tp_fp_tn_fn(expect):
    def label(e_i, p_i):
        return ("T" if e_i == p_i else "F") + ("P" if p_i else "N")

    n = len(expect)
    return dp.t.make_user_transformation(
        input_domain=dp.vector_domain(dp.atom_domain(T=bool), size=n),
        input_metric=dp.symmetric_distance(),
        output_domain=dp.vector_domain(dp.atom_domain(T=str), size=n),
        output_metric=dp.symmetric_distance(),
        function=lambda actual: [label(e, p)
                                 for e, p in zip(expect, actual)],
        stability_map=lambda b_in: b_in  # 1-stable
    )

def precision_postprocess(tp_fp_tn_fn):
    tp, fp, *_ = tp_fp_tn_fn
```

The complete private mechanism labels each sample, counts the number of occurrences of each label, privatizes with discrete Laplacian noise, and then postprocesses the counts into a precision.

```
import numpy as np
expect = np.random.choice(a=[False, True], size=1000)
actual = np.random.choice(a=[False, True], size=1000)

categories = ["TP", "FP", "TN", "FN"]
meas = (
    make_tp_fp_tn_fn(expect) >> # 1-stable labeling
    dp.t.then_count_by_categories(categories, null_category=False) >>
    dp.m.then_laplace(2.) >>
    precision_postprocess
)
print(meas(actual))     # ~> 0.4763 precision
```

4. A vector of per-customer daily sales arrives once-a-day.

   a. Adjust the transition function to clamp and sum the output of the stream.

```
def make_sum_stream(bounds):
    dp.assert_features("contrib")
```

```
    T, L, U = type(bounds[0]), *bounds
    def f_sum_stream(stream):
        transition = lambda data: \
            sum(max(min(v, U), L) for v in stream(data))
        return dp.new_queryable(transition, Q=dp.Vec[T], A=T)

    stream_output_domain = dp.vector_domain(dp.atom_domain(bounds=bounds))
    return dp.t.make_user_transformation(
        input_domain=queryable_domain(stream_output_domain),
        input_metric=dp.symmetric_distance(), # technically l-infty of sym dists
        output_domain=queryable_domain(dp.atom_domain(T=float)),
        output_metric=dp.linf_distance(T=T, monotonic=True),
        function=f_sum_stream,
        stability_map=lambda b_in: b_in * max(abs(L), U)
    )
```

Adjust the stability map to account for how much total sales can vary per-day (sensitivity).

The stream privatization mechanism is simply the chaining of the stream summing transformation and the above threshold mechanism.

```
dp.enable_features("honest-but-curious", "contrib", "floating-point")
meas = make_sum_stream((0., 10.)) >> \
       make_above_threshold(threshold=100., scale=10., monotonic=True)
```

This mechanism is likely to detect day 3.

b. To construct a mechanism that detects sales less than 10, simply negate the output of the sum transformation, and choose a threshold of -10. This mechanism is likely to detect day 2.

Chapter 5

1. You can manage timing side-channels by privatizing the execution time itself. That is, the elapsed time could be made into a DP release via the truncated Laplace mechanism. To do so, first choose an expected `delay`. Assuming the execution time may only differ by t seconds on adjacent data sets, execute the timing-vulnerable mechanism. Now draw a sample from the truncated Laplace distribution, with radius no greater than `delay`, centered at `delay`, and sleep for this many seconds.

```
def make_private_timing(meas, t, scale, delay):
    dp.assert_features("contrib")
    import time
    if meas.output_measure.type.origin != \
            "FixedSmoothedMaxDivergence":
        raise ValueError("measurement must give "
                         "privacy guarantees wrt (ε, δ)")

    tlap = make_truncated_laplace(scale, delay)
```

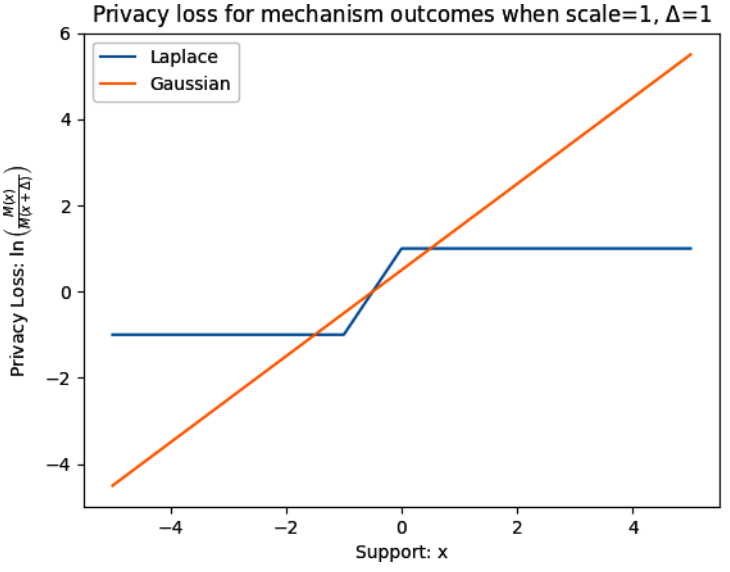
```
def f_add_timing_delay(arg):
    time.sleep(tlap(delay)) # random delay privatizes elapsed time
    return meas(arg)

def privacy_map(b_in):
    (eps_1, del_1), (eps_2, del_2) = meas.map(b_in), tlap.map(t)
    return eps_1 + eps_2, del_1 + del_2

return dp.m.make_user_measurement(
    input_domain=meas.input_domain,
    input_metric=meas.input_metric,
    output_measure=meas.output_measure,
    function=f_add_timing_delay,
    privacy_map=privacy_map)
```

The final privacy consumption is the composition of the privacy loss of the mechanism of interest, as well as the privacy loss of releasing the elapsed time.

2. The privacy loss of the Gaussian mechanism is linear: the multiplicative difference between the tails of distributions on adjacent data sets increases linearly as more extreme values are sampled.



Unlike the Laplace mechanism, whose privacy loss never exceeds its precalibrated value, the privacy loss of the Gaussian mechanism is unbounded! Fortunately, it becomes very unlikely to sample an extreme value with large

privacy loss: so unlikely that you can bound the probability of encountering a privacy loss greater than ϵ with a small δ parameter.

3. The following script derives a privacy map that accounts for the privacy loss of multiple applications of the Gaussian mechanism.

```
from math import sqrt
import opendp.prelude as dp


def plrv_gaussian_composition(scales):
    def privacy_map(b_in):
        eta_g = sum(b_in**2 / (2 * s_i**2) for s_i in scales)
        scale_g = sqrt(b_in**2 / (2 * eta_g))

        space = dp.atom_domain(T=float), dp.absolute_distance(T=float)
        return dp.m.make_base_gaussian(*space, scale_g).map(b_in)

    return privacy_map
```

4. Privacy guarantees get progressively weaker from left to right. Each conversion from one definition to the next is lossy in some way.

```
flowchart LR
    B[Bounded Range] --> P[Pure]
    P --> Z[zCDP]
    Z --> R[Renyi]
    R --> A[Approx/f-DP]
    P --> A
```

5. The following function provides a conversion from a privacy profile (a set of potential ϵ, δ guarantees) to an f-DP trade-off curve:

```
def f_approxDP(profile):
    def curve(alpha):
        return max(max(
            0,
            1 - delta - np.exp(epsilon) * alpha,
            np.exp(-epsilon) * (1 - delta - alpha),
        ) for epsilon, delta in profile)
    return curve
```

An f-DP curve can be instantiated with a trivial profile of one approx-DP parameter pair:

```
curve = f_approxDP([(1., 1e-6)])
alpha = .2
beta = curve(alpha)  # -> 0.4563426343081909
```

When the probability that an adversary falsely claims Alice is not in the data is .2, the adversary would falsely claim Alice is in the data with probability .456. For this choice of privacy parameters, the adversary remains particularly disadvantaged.

Chapter 6

1. The sparse vector technique can be implemented as an interactive postprocessing of above threshold mechanisms. This implementation makes use of a basic compositor to spawn k above threshold mechanisms, and exhausts each in turn.

```
def make_sparse_vector(threshold, scale, k, monotonic=False):
    """Privately find the first k items above `threshold` in a stream."""
    dp.assert_features("contrib", "floating-point")

    meas_AT = make_above_threshold(threshold, scale, monotonic)
    meas_BC = dp.c.make_basic_composition([meas_AT] * k)

    def f_sparse_vector(stream):
        qbls_AT, found = meas_BC(stream), 0
        def transition(query):
            nonlocal found
            assert found < k, "sparse vector mechanism is exhausted"
            if qbls_AT[found](query):
                found += 1
                return True
            return False

        return dp.new_queryable(transition, Q=query_type(stream), A=bool)

    return dp.m.make_user_measurement(
        input_domain=queryable_domain(dp.atom_domain(T=type(threshold))),
        input_metric=dp.linf_distance(T=type(threshold), monotonic=monotonic),
        output_measure=dp.max_divergence(T=float),
        function=f_sparse_vector,
        privacy_map=lambda b_in: b_in * meas_BC.map(1),
    )
```

   a. Remember that the use of basic composition results in a 4x increase of privacy spend.

```
meas = make_sum_stream((0., 10.)) >> \
       make_sparse_vector(threshold=100., scale=10., k=4, monotonic=True)
```

2. Using NumPy:

```
import numpy as np
loose_bounds = -10.0, 10.0
scale = 1.0

data = np.random.normal(1000)
```

```
mean = np.mean(data)                        # A

m_05 = make_private_quantile_in_bounds(loose_bounds, alpha=0.05, scale=scale)
m_95 = make_private_quantile_in_bounds(loose_bounds, alpha=0.95, scale=scale)
p_05, p_95 = m_05(data), m_95(data)         # B

clipped_data = np.clip(data, p_05, p_95)  # C
clipped_mean = np.mean(clipped_data)      # D
```

The sensitivity of `clipped_mean` is unknown, since the clipping bounds are data-dependent.

3. The `make_private_quantile_in_bounds` constructor from ??? can be used to individually release the lower and upper bounds.

```
from ch04_interval_exponential_mechanism import make_private_quantile_in_bounds
import opendp.prelude as dp


def make_private_bounds_via_quantile(loose_bounds, scale):
    return dp.c.make_basic_composition([
        make_private_quantile_in_bounds(loose_bounds, alpha=0.05, scale=scale),
        make_private_quantile_in_bounds(loose_bounds, alpha=0.95, scale=scale),
    ])
```

Basic composition is all that's necessary to construct the bounds estimation mechanism.

4. Parallel composition relies on the property that an individual may only influence a bounded number of partitions. This proof relies on the assumption that any one individual may influence only as many as k partitions. There are n total partitions, each of which are denoted x_i or x_i', respectively.

$$
\begin{aligned}
&\max_{x \sim x'} D(M(x), M(x')) \\
&= \max_{x \sim x'} D(M_1(x_1) \times ... \times M_n(x_n), M_1(x_1') \times ... \times M_n(x_n')) \\
&= \max_{x \sim x'} \sum_{i=1}^{n} D(M_i(x_i), M_i(x_i')) \\
&\leq k \cdot \max_{1 \leq i \leq n} \max_{x_i \sim x_i'} D(M_i(x_i), M_i(x_i')) \qquad x_i = x_i' \text{ in all but k partitions} \\
&= k \cdot \max_{1 \leq i \leq n} \epsilon_i
\end{aligned}
$$

Since the total privacy loss is based on the maximum privacy loss in any one partition, it makes sense to use mechanisms that have the same privacy loss ϵ_0 in each

partition, to maximize utility. If this is the case, then the total privacy loss is simply $k \cdot \epsilon_0$.

Chapter 7

1. Recall the identifier distance metric.
   a. Not adjacent - since these datasets differ by the removal of one id (10) and addition of another (9), the identifier distance is two. Just like in the distinction between bounded-DP and unbounded-DP, you might find it useful to define a metric that considers datasets adjacent if any one individual's data is *changed*. However, even in this setting, these datasets would still not be adjacent, because they have different identifiers. These datasets would only be considered adjacent if the dataset distance could be two: accounting for the case where one individual may influence multiple user ids.
   b. Not adjacent - the dataset difference is now four.

2. Using the browser logs data set:
   a. To demonstrate that the presence of rare events can lead to a user-level privacy violation, you can describe how specific events can be tied to the presence of specific users. For example, in the browser logs data set, personal information may be embedded into a URL that can reveal the presence of an individual in the data set.
   b. In practice, the parameter τ in τ-thresholding is typically chosen to satisfy a given δ guarantee. That is, you first choose suitable privacy parameters, and then solve for corresponding τ parameter. The purpose of thresholding is to exclude *rare* events. The setting of privacy parameters calibrates exactly how rare an event can be. An extreme example from the browser logs data set would be the case where a URL is only accessed by a single user. That URL might either be directly linked to the user (e.g. by containing user credentials) or can be indirectly linked using auxiliary data. Setting a threshold of one would give very poor protections, and commensurately higher δ. Privacy violations are avoided by only releasing URLs that have been accessed by a significant number of individuals.

3. Parallel composition is a property of differential privacy that can impact the privacy analysis from the privacy loss budget perspective.
   a. If the browser logs data set contains a column that provides location information, and each user is assigned to a single location, the location column can be used to partition the data.

b. Partitioning the data enables parallel composition. This means that making queries with the same privacy loss parameter in each data partition results in spending that privacy loss budget only once, as you proved in the final exercise of ???.

Chapter 8

1. As discussed in the chapter, the Theil-Sen estimator takes pairs of sample points $\{(x_i, y_i), (x_j, y_j)\}$, calculates the slope between them $\frac{y_j - y_i}{x_j - x_i}$, and then releases the DP median of the slopes.

   a. An implementation of the Theil-Sen estimator in Python could look like this:

```
import statistics
from itertools import permutations


def theil_sen(data):
    """
    :param data: form [(x0, y0), ..., (xN, yN)]
    """
    pairs = permutations(data, 2)
    slopes = [(p1[1] - p0[1]) / (p1[0] - p0[0]) for p0, p1 in pairs]
    return statistics.median(slopes)
```

   b. A DP Theil-Sen estimator that uses all possible pairs of points can be implemented as follows:

```
import opendp.prelude as dp
from itertools import permutations

dp.enable_features("honest-but-curious", "contrib")


def make_theil_sen_slopes(size):
    def slope(p0, p1):
        return (p1[1] - p0[1]) / (p1[0] - p0[0])

    return dp.t.make_user_transformation(
        input_domain=dp.np_array2_domain(T=float, num_columns=2, size=size),
        input_metric=dp.symmetric_distance(),
        output_domain=dp.vector_domain(dp.atom_domain(T=float)),
        output_metric=dp.symmetric_distance(),
        function=lambda x: [slope(p0, p1) for p0, p1 in permutations(x, 2)],
        # each point influences n - 1 slopes
        stability_map=lambda b_in: (size - 1) * b_in,
    )


def make_theil_sen(size, bounds, scale):
```

```
        return make_theil_sen_slopes(size) >> make_private_quantile_in_bounds(
            bounds, 0.5, scale=scale
        )


    def dp_theil_sen(size, bounds, epsilon):
        return dp.binary_search_chain(
            lambda s: make_theil_sen(size, bounds, s), d_in=1, d_out=epsilon
        )
```

c. Using Numpy:

```
import numpy as np

# Create an initial dataset of 1000 data points
x = np.arange(0, 1000, 1)
# Use x as an input to a linear equation, with some unknown error
y = 2 * x + 3 + np.random.normal(0., scale=1., size=len(x))
data = np.stack([x, y], axis=1)

m_theil_sen = dp_theil_sen(size=1000, bounds=(0, 10), epsilon=1.)
print(m_theil_sen(data))
```

d. The non-DP Theil-Sen estimator calculates model parameters closer to the true slope and intercept of the data you generated.

   Notice that DP fits of the slope term will diverge if y is a perfect linear function of x (when no error term is added to y). In this setting all slopes are constant, meaning the distance between each slope is zero, so the likelihood of the mechanism selecting the interval between each slope goes to zero. This causes the mechanism to behave poorly: the quantile mechanism always returns a slope uniformly from an interval to the left or right of the constant. There are alternative implementations of the mechanism that provide robustness against this degenerate case.

2. Instead of only releasing the DP Theil-Sen slope, this answer will release both the slope and intercept. The approach still uses pairings of two points, but this time releases the median of the 25th and 75th percentiles of each line. These quantities can then be post-processed into a slope and intercept.

   The following code provides the function for the transformation into data sets of the 25th and 75th percentiles of each line:

```
import opendp.prelude as dp
import numpy as np

alphas = np.array([[0.25], [0.75]])

def f_match(data):
    # keep an even number of rows
    data = np.array(data, copy=True)[:len(data) // 2 * 2]
```

```
    # evenly partition into random pairs
    np.random.shuffle(data)
    p1, p2 = np.array_split(data, 2)

    # compute a vector data set of slopes
    dx, dy = (p2 - p1).T
    slope = dy / dx

    # compute points on each line at 25th and 75th percentiles
    x_bar, y_bar = (p1 + p2).T / 2
    points = slope * (alphas - x_bar) + y_bar

    # only keep well-defined pairings
    return points.T[dx > 0]
```

This can be made into a k-stable transformation by stacking the data points from k random sets of matchings:

```
import opendp.prelude as dp
import numpy as np

alphas = np.array([[0.25], [0.75]])

def f_match(data):
    # keep an even number of rows
    data = np.array(data, copy=True)[:len(data) // 2 * 2]

    # evenly partition into random pairs
    np.random.shuffle(data)
    p1, p2 = np.array_split(data, 2)

    # compute a vector data set of slopes
    dx, dy = (p2 - p1).T
    slope = dy / dx

    # compute points on each line at 25th and 75th percentiles
    x_bar, y_bar = (p1 + p2).T / 2
    points = slope * (alphas - x_bar) + y_bar

    # only keep well-defined pairings
    return points.T[dx > 0]
```

Afterwards, you need a measurement that can, via composition, apply the median mechanism to both the 25 and 75 percentile datasets.

```
def make_private_theil_sen(bounds, k, scale):
    # Released percentiles relate to regression coefficients by a system:
    #     p25 = .25 α + β
    #     p75 = .75 α + β
    # Equivalently:
    #     p = [[.25, 1.0], [.75, 1.0]] @ [[α], [β]]

    # Solve for the regression coefficients by inverting the system:
```

```
    def postprocess(p):
        return -2 * np.array([[1, -1], [-.75, .25]]) @ p

    m_median = make_private_quantile_in_bounds(bounds, 0.5, scale=scale)
    return make_theil_sen_percentiles(k) >> dp.c.make_basic_composition([
        make_select_column(0) >> m_median,
        make_select_column(1) >> m_median,
    ]) >> postprocess
```

This code snip made use of a helper transformation that selects a column out of a numpy array:

```
def make_select_column(j):
    return dp.t.make_user_transformation(
        input_domain=dp.np_array2_domain(T=float, num_columns=2),
        input_metric=dp.symmetric_distance(),
        output_domain=dp.vector_domain(dp.atom_domain(T=float)),
        output_metric=dp.symmetric_distance(),
        function=lambda x: x[:, j],
        stability_map=lambda b_in: b_in)
```

Altogether, the scale parameter can be calibrated to satisfy any level of privacy, given any choice of k.

```
dp.enable_features("honest-but-curious", "contrib")
meas = make_private_theil_sen((-10, 10), k=1, scale=1.)
x = np.random.normal(size=1_000)
y = 2 * x + 3 + np.random.normal(size=1_000)
data = np.stack([x, y], axis=1)

print(meas(data))
```

3. The R^2 statistic is calculated as $1 - \frac{RSS}{TSS}$, where $RSS = \Sigma_i^n (y_i - \widehat{y}_i)^2$ and $TSS = \Sigma_i^n (y_i - \overline{y})^2$. While the sensitivity of the entire statistic is undefined (as the TSS could be zero), the sensitivity of RSS and TSS can each be bounded by clipping the differences. If each difference is bounded between $[-C, C]$, then the sensitivity of the sums of squares would be C^2. The fraction could be released by basic composition.

   Alternatively, the quantity could be released using the sample and aggregate framework discussed in ???.

4. Throughout this book, you have learned about how to make differentially private low-level queries to databases, such as count and sum queries. As seen in this chapter, these low-level simple queries can be used in the construction of differentially private algorithms, such as decision trees. You can obtain a differentially private decision tree algorithm by replacing all access (queries) to the training data set with the equivalent differentially private query. In the case of the decision tree algorithm, a simple way to transform the decision tree into its differentially

private version is to compute all probabilities and conditional probabilities using differentially private counts and sums. By doing that, the remaining steps on the algorithms are all post-processing steps of differentially private sum and counts, According to the post-processing (as you learned in ???), this preserves privacy.

5. In this exercise, The first step for the reader is to identify, for each feature in the *Adult data set*, if the feature is categorical or numerical. For categorical variables, the sensitivity of the conditional probability is 1, according to ???. In the case of numerical features, the reader will need to first guesstimate the minimum and maximum values of the numerical feature, and then proceed computing the sensitivity of the mean and the sensitivity of the standard deviation according to the equations in ???.

Chapter 9

1. As you change the `noise_multiplier` and `max_grad_norm` parameters, you will notice that the model utility (e.g. model accuracy) and privacy loss budget will change. This exercise should be approached as follows:
   a. Keep `max_grad_norm` fixed, while experimenting with different values of the `noise_multiplier`. High values of `noise_multiplier` should result in low model utility and low privacy loss budget. As you decrease `noise_multiplier` you should see an increase in model utility and privacy loss budget.
   b. Similar process to the previous item.
2. To answer this, you'll need to create a measurement that randomly selects hyperparameters, and releases a utility score, fitted model and hyperparameters.

   First, create a DP utility estimate. Counting the number of correct classifications will suffice:

```
def make_count_correct():
    return dp.t.make_user_transformation(
        input_domain=dp.np_array2_domain(T=float, num_columns=2),
        input_metric=dp.symmetric_distance(),
        output_domain=dp.atom_domain(T=float),
        output_metric=dp.absolute_distance(T=float),
        function=lambda x: sum(x[:, 0] == x[:, 1]),
        stability_map=lambda d_in: d_in,
    )
```

   Then, wrap the training process and evaluation in a measurement. In the following script, ``fit_adult_with_opacus`` is a function that wraps the adult model training from earlier in this chapter. The function always consumes the same privacy budget, regardless of the choice of hyperparameters.

```
def make_dpsgd_random_params(score_scale):
    m_acc = make_count_correct() >> dp.m.then_gaussian(score_scale)
    m_acc = make_zCDP_to_RDP(m_acc)

    def f_fit_score(data):
        hyperparams = dict(
            max_grad_norm=np.random.uniform(0.1, 10),
            hidden_size=np.random.randint(1, 500),
            learning_rate=np.random.exponential(0.05),
        )
        train, test = data
        model = fit_adult_with_opacus(train, **hyperparams)
        score = m_acc(np.stack([model(test.x), test.y], axis=1))
        return score, model, hyperparams
    # ...
```

For any setting of the arguments to ``fit_adult_with_opacus``, the history parameters don't change. Therefore, the history of gradient updates used to train the model remains constant, and can be used to create a renyi-DP privacy map. Notice this privacy map accounts for the composition of both the RDP utility estimate and model training.

```
    # ...
    def privacy_map(d_in):
        def rdp_curve(alpha):
            # these constants are from privacy_engine.accountant.history
            eps_train = compute_rdp(
                q=0.00031145508783559825,
                noise_multiplier=1.0,
                steps=30170,
                orders=alpha,
            )

            eps_test = m_acc.map(d_in)(alpha)
            return max(eps_train, eps_test)

        return rdp_curve

    return dp.m.make_user_measurement(
        input_domain=dp.np_array2_domain(T=float),
        input_metric=dp.symmetric_distance(),
        output_measure=renyi_divergence(),
        function=f_fit_score,
        privacy_map=privacy_map,
    )
```

Each time ``m_dpsgd`` is invoked, it will randomly choose hyperparameters, train the model, and evaluate the utility. When ``m_dpsgd_pspc`` is invoked, it will invoke ``m_dpsgd`` a random number of times, and only return the utility, model and selected hyperparameters of the best run.

```
m_dpsgd = make_dpsgd_random_params(score_scale=1.0)
m_dpsgd_pspc = make_pspc_negative_binomial(m_dpsgd, p=0.1, n=2)
data = AdultDataSet("adult.data"), AdultDataSet("adult.test")
score, model, hyperparams = m_dpsgd_pspc(data)
```

Altogether, while PSPC makes the privacy parameters larger, you may find that the automatic selection of hyperparameters results in vastly superior model performance.

3. Either DP-SGD or PATE could prove to be more suitable. In the absence of public data, it would not be feasible to train the student models with PATE. However, with the aid of public data, PATE may out-perform DP-SGD.

Chapter 10

1. First, note that MWEM is a series of T iterations, each containing one application of the exponential mechanism and one application of the Laplace mechanism. Each mechanism has $\frac{\epsilon}{2T}$. By basic composition:

$$\sum_{i}^{T} \frac{\epsilon}{2T} + \sum_{i}^{T} \frac{\epsilon}{2T} = 2T\frac{\epsilon}{2T} = \epsilon$$

2. You should notice that it takes at least 10 minutes for this process to complete, even if you only run the sampler on the first 1000 data points. The next part of the solution explains why.
   a. The efficiency of the MWEM algorithm is sensitive to the dimensions of the data cube of the *Adult data set*. The dimension of the data cube depends on the number of columns in the data set and the cardinality of each column. It is *not* sensitive to the number of rows. Because some columns of the *Adult data set* are numerical, the MWEM algorithm will process such columns as a high cardinality column, resulting in inefficiency. In other words, the process will take a lot longer to complete than you might expect. By transforming numerical columns into categorical columns, the process of generating synthetic data sets with the MWEM algorithms should become considerably more efficient.
   b. This code could look like the following:

```
import pandas as pd
from sklearn.compose import make_column_transformer

from snsynth import Synthesizer

from sklearn.preprocessing import OrdinalEncoder, StandardScaler, Normalizer
```

```
header = ['age', 'workclass', 'fnlwgt', 'education', 'education_num',
          'marital_status', 'occupation', 'relationship',
          'race', 'sex', 'capital_gain', 'capital_loss',
          'hours_per_week', 'native_country', 'income']

df = pd.read_csv('adult.data', header=None, names=header,
                 sep=',\\s', na_values=['?'], engine='python')
df = df.dropna()
df = df.reset_index(drop=True)
df['income'] = df['income'].apply(lambda x: x.replace('.', ''))

# Part 1: no preprocessing
column_names = ['workclass', 'age', 'income', 'hours_per_week']

synth = Synthesizer.create('mwem', epsilon=1.0)
sample = synth.fit_sample(df[column_names], preprocessor_eps=0.2)

sample_conditional = synth.sample_conditional(2, "age < 65 AND income > 0")

print(sample_conditional)

# Part 2: with preprocessing
categorical_columns = ['workclass', 'income']
numerical_columns = ['age', 'hours_per_week']

column_transformer = make_column_transformer(
    (OrdinalEncoder(), categorical_columns),
    (StandardScaler(), numerical_columns),
)

X = column_transformer.fit_transform(df[column_names])
X = Normalizer().fit_transform(X)

for epsilon, preproc_eps in zip([0.1, 1., 10.], [0.02, 0.2, 2.0]):
    print(f'Epsilon: {epsilon}\n')
    synth = Synthesizer.create('mwem', epsilon=1.0)
    sample = synth.fit_sample(X, column_names=[column_names], preprocessor_eps=preproc_eps)
    sample.to_csv(f'sample_epsilon_{epsilon}.csv')
    sample_conditional = synth.sample_conditional(2, "age < 65 AND income > 0")

    print(sample_conditional)
```

c. This exercise should help you to understand how epsilon impacts the quality of the generated synthetic data. Compare the one-way and two-way marginals of the synthetic data sets with the marginals of the *Adult data*. Data sets generated with higher values of epsilon should provide marginals with higher utility.

3. In this exercise the reader will make a similar analysis from the previous exercise, but this time using the *PrivBayes* algorithm. The *PrivBayes* algorithm, as the

MWEM algorithm, synthesizes data based on marginal queries. When an algorithm is *marginal based*, numerical columns are interpreted as columns with high cardinality, impacting utility. However, the *PrivBayes* algorithm usually results in data sets with higher utility compared to MWEM algorithm. The utility analysis should show that for a same epsilon, the *PrivBayes* algorithm achieves better utility than MWEM algorithm.

4. GAN-based synthetic data generators can experience mode collapse, which means that the data generator is unable to learn properly the relationship between variables. This happens for low values of epsilon, but also happens when there is not enough data samples. The *Adult data* can lead to mode collapse, specially for low values of epsilon. By running the proposed experiments a few times, the reader should be able to observe mode collapse.

4. DPCTGAN….

Chapter 11

1. Explain how differential privacy addresses each of the attacks discussed in this chapter.
   a. *Record linkage*. Differential privacy is robust against auxiliary information. An example in practice: Assuming you are using local-DP to gain access to microdata, the greater uncertainty/reduced utility of local-DP makes joins/linkages highly inaccurate.
   b. *Singling out*. Differential privacy is a criterion all queries must pass, including "singling out" queries. In the instance of a count query with a predicate that singles out an individual, the sensitivity of the query works out to the range of the query, resulting in what is essentially a local-DP guarantee that is too noisy to derive any confidence in.
   c. *Differencing*. The effectiveness of differencing is even less than that of singling out since, for the same level of privacy, you now need to add two instances of noise, each with twice as much variance.
   d. *Reconstruction via systems of equations*. Since differential privacy accounts for the release of multiple statistics via composition, the effectiveness of reconstruction attacks via a system of equations deteriorates for the same reason as differencing attacks do, but to an even greater extent, since the privacy budget is further split.
   e. *Tracing*. When the test statistic is released via differential privacy, the high-sensitivity-nature of the statistic dominates the utility of the release. Again, this is due to the fundamental property of differential privacy that guarantees are immune from auxiliary information.

Chapter 12

1. Let's examine each aspect of contextual integrity that you read about in this chapter. The key to this exercise is to note the differences that exist between a human trainer and the app.
   a. The following are the contextual integrity parameters for the app:
      i. data subject - the athlete / app user
      ii. data senders - without the app, this would only include the (human) trainer or the athlete directly sending information. The app can also be a data sender that delivers data to third parties, such as insurance companies and advertisers.
      iii. data recipients - Doctors and other medical practitioners, insurance agencies, and advertisers. Anyone who can receive data from the app becomes a data recipient.
      iv. information type - any personal health data collected by the app. As mentioned in the exercise, this can include resting heart rate, oxygen level, and daily step count, and oxygen level. Notably, this is much more granular, second-by-second data than what may be collected by a human trainer.
      v. transmission principle - personally identifiable information should only be sent to the athlete's healthcare provider with their permission, and these organizations are bound by HIPAA. Aggregate statistics may be sent to third parties, but not microdata. Depending on the app and the privacy policy agreed to by the user, some apps may allow for aggregate statistics (or even microdata) to be sent to advertising companies.
   b. When the app is using DP for aggregate statistics, then one may argue that several of the third parties are no longer data recipients. For example, advertisers and insurance companies that are receiving differentially private aggregate statistics would no longer have the ability to reconstruct microdata from any of the athletes who use the app. Further, the transmission principle can be modified: privatized aggregate statistics can now be shared more widely without the risk of re-identification.
2. This data set consists of bike share rides across the Washington, D.C. metropolitan area during December 2023. Each row has an origin and destination station, as well as their corresponding latitude and longitude points. To understand acceptable values of the privacy parameters, let's imagine the worst privacy violation possible from this data set. Consider a situation where a local business has a loyalty program that logs each user's transactions in the store, along with their location. If this app's data is leaked publicly, then linked with the bike share data,

the most an attacker can learn is the origin of the customer before coming to the store. For this scenario, you might be willing to use larger ϵ and δ values.

   a. Note that each station has unbounded contribution to the data set. To start, you will want to bound the contribution so that you can have a usable measure of sensitivity before adding noise to any of the aggregate statistics you release.

3. As mentioned in the footnotes, this final question is left deliberately vague. For each problem field and context, there will be subtle differences and considerations to be made. Feel free to reach out to *authors@handsondpbook.com* with any questions.