`fn make_private_group_by`

Michael Shoemate

This proof resides in **"contrib"** because it has not completed the vetting process.

Proves soundness of `fn make_private_group_by` in `mod.rs` at commit 0db9c6036 (outdated[1]).

Vetting History

- Pull Request #512

1 Hoare Triple

Precondition

To ensure the correctness of the output, we require the following preconditions:

- Type `MS` must have trait `DatasetMetric`.
- Type `MI` must have trait `UnboundedMetric`.
- Type `MO` must have trait `ApproximateMeasure`.

Pseudocode

```python
def make_private_group_by(
    input_domain,
    input_metric,
    output_measure,
    plan,
    global_scale,
    threshold,
):
    input_expr, keys, aggs, predicate = match_group_by(plan)

    t_prior = input_expr.make_stable(input_domain, input_metric)  #
    middle_domain, middle_metric = t_prior.output_space()

    grouping_columns = match_grouping_columns(keys)  #

    margin = (
        middle_domain
        .margins
        .get(grouping_columns, Margin.default())
    )  #

    # prepare a joint measurement over all expressions
    expr_domain = ExprDomain(  #
```

[1]See new changes with `git diff 0db9c6036..d0515a7 rust/src/measurements/make_private_lazyframe/group_by/mod.rs`

```
        middle_domain,
        ExprContext.Aggregate(grouping_columns),
    )

    m_exprs = make_basic_composition([
        make_private_expr(
             expr_domain,
             PartitionDistance(middle_metric),
             output_measure,
             expr,
             global_scale,
        ) for expr in aggs
    ])

    # reconcile information about the threshold
    dp_exprs = m_exprs.invoke((input_expr, all()))

    if margin.public_info is not None:
        threshold_info = None
    elif is_threshold_predicate(predicate) is not None:
        name, threshold_value = is_threshold_predicate(predicate)
        noise = find_len_expr(dp_exprs, name)[1]
        threshold_info = name, noise, threshold_value, False
    elif threshold is not None:
        name, noise = find_len_expr(dp_exprs, None)[1]
        threshold_info = name, noise, threshold_value, True
    else:
        raise f"The key set of {grouping_columns} is private and cannot be released."

    # prepare the final_predicate to be used in the function
    if threshold_info is not None:
        name, _, threshold_value, is_present = threshold_info
        if not is_present and predicate is not None:
            final_predicate = threshold_expr.and_(predicate)
        else:
            final_predicate = threshold_expr
    else:
        final_predicate = predicate

    # prepare supporting elements
    def function(arg):  #
        output = DslPlan.GroupBy(
            input=arg,
            keys=keys,
            aggs=m_exprs((arg, all())),
            apply=None,
            maintain_order=False,
        )

        if final_predicate is not None:
            output = DslPlan.Filter(
                input=output,
                predicate=final_predicate,
            )
        return output

    def privacy_map(d_in):  #
        mip = margin.get("max_influenced_partitions", default=d_in)
        mnp = margin.get("max_num_partitions", default=d_in)
        mpc = margin.get("max_partition_contributions", default=d_in)
        mpl = margin.get("max_partition_length", default=d_in)

        l0 = min(mip, mnp, d_in)
        li = min(mpc, mpl, d_in)
```

```
        l1 = l0.inf_mul(li).min(d_in)

        d_out = m_exprs.map((l0, l1, li))

        if threshold is not None:
            _, plugin, threshold_value = threshold_info
            if li >= threshold_value:
                raise f"Threshold must be greater than {li}."
            d_instability = f64.inf_cast(threshold_value.inf_sub(li))
            delta_single = integrate_discrete_noise_tail(plugin.distribution, plugin.scale,
    d_instability)
            delta_joint = 1 - (1 - delta_single).inf_powi(l0)
            d_out = MO.add_delta(d_out, delta_joint)
        elif margin.public_info is None:
            raise "keys must be public if threshold is unknown"

        return d_out

    m_group_by_agg = Measurement(
        middle_domain,
        function,
        middle_metric,
        output_measure,
        privacy_map,
    )

    return t_prior >> m_group_by_agg
```

Postconditions

Theorem 1.1. For every setting of the input parameters (`input_domain`, `input_metric`, `output_measure`, `plan`, `global_scale`, `threshold`) to `make_private_group_by` such that the given preconditions hold, `make_private_group_by` raises an exception (at compile time or run time) or returns a valid measurement. A valid measurement has the following property:

1. (Privacy guarantee). For every pair of elements x, x' in `input_domain` and for every pair (`d_in`, `d_out`), where `d_in` has the associated type for `input_metric` and `d_out` has the associated type for `output_measure`, if x, x' are `d_in`-close under `input_metric`, `privacy_map`(`d_in`) does not raise an exception, and `privacy_map`(`d_in`) $\leq$ `d_out`, then `function`(x), `function`(x') are `d_out`-close under `output_measure`.

2 Proof

We now prove the postcondition of `make_private_group_by`.

Proof. Start by establishing properties of the following variables, which hold for any setting of the input arguments.

- By the postcondition of `StableDslPlan.make_stable`, `t_prior` is a valid transformation. 11
- By the postcondition of `match_grouping_columns`, `grouping_columns` holds the names of the grouping columns. `margin` denotes what is considered public information about the key set, pulled from descriptors in the input domain. 14
- By the postcondition of `make_basic_composition`, `m_exprs` is a valid measurement that prepares a batch of expressions that, when executed, satisfies the privacy guarantee of `m_exprs`. 23

We now reconcile information about the censoring threshold. 38 In the setting where grouping keys are considered public, no thresholding is necessary. In the setting where grouping keys are considered private information, threshold information is prepared from either `predicate` or `threshold`. By the post-condition of `find_len_expr`, filtering on `name` can be used to satisfy δ-approximate DP.

The final predicate to be applied is the intersection of conditions necessary for filtering, and initial conditions set in the predicate. The thresholding predicate is only added to the final predicate if the threshold is not already present in the predicate.

We now move on to the implementation of the function. 64 The function returns a DslPlan that applies each expression from `m_exprs` to `arg` grouped by `keys`. `threshold_info` is conveyed into the plan, if set, to ensure that the keys are also privatized if necessary. It is assumed that the emitted DSL is executed in the same fashion as is done by Polars. This proof/implementation does not take into consideration side-channels involved in the execution of the DSL.

We now move on to the implementation of the privacy map. 80 The measurement for each expression expects data set distances in terms of a triple:

- L^0: the greatest number of partitions that can be influenced by any one individual. This is no greater than the input distance (an individual can only ever influence as many partitions as they contribute rows), but could be smaller when supplemented by the `max_influenced_partitions` metric descriptor or `max_num_partitions` domain descriptor.
- L^∞: the greatest number of records that can be added or removed by any one individual in each partition. This is no greater than the input distance, but could be tighter when supplemented by the `max_partition_contributions` metric descriptor or the `max_partition_length` domain descriptor.
- L^1: the greatest total number of records that can be added or removed across all partitions. This is no greater than the input distance, but could be tighter when accounting for the L^0 and L^∞ distances.

By the postcondition of the map on `m_exprs`, the privacy loss, when grouped data sets may differ by this distance triple, is `d_out`.

We now adapt the proof from [Rog23] (Theorem 7). Consider S to be the set of labels that are common between x and x'. Define event E to be any potential outcome of the mechanism for which all labels are in S (where only stable partitions are released). We then lower bound the probability of the mechanism returning an event E. In the following, c_j denotes the exact count for partition j, and Z_j is a random variable distributed according to the distribution used to release a noisy count.

$$\begin{aligned}
\Pr[E] &= \prod_{j \in x \setminus x'} \Pr[c_j + Z_j \le T] \\
&\ge \prod_{j \in x \setminus x'} \Pr[\Delta_\infty + Z_j \le T] \\
&\ge \Pr[\Delta_\infty + Z_j \le T]^{\Delta_0}
\end{aligned}$$

The probability of returning a set of stable partitions ($\Pr[E]$) is the probability of not returning any of the unstable partitions. We now solve for the choice of threshold T such that $\Pr[E] \ge 1 - \delta$.

$$\begin{aligned}
\Pr[\Delta_\infty + Z_j \le T]^{\Delta_0} &= \Pr[Z_j \le T - \Delta_\infty]^{\Delta_0} \\
&= (1 - \Pr[Z_j > T - \Delta_\infty])^{\Delta_0}
\end{aligned}$$

Let `d_instability` denote the distance to instability of $T - \Delta_\infty$. By the postcondition of `integrate_discrete_noise_tail`, the probability that a random noise sample exceeds `d_instability` is at most `delta_single`. Therefore $\delta = 1 - (1 - \texttt{delta\_single})^{\Delta_0}$. This privacy loss is then added to `d_out`.

Together with the potential increase in delta for the release of the key set, then it is shown that `function`(x), `function`(x') are `d_out`-close under `output_measure`.

□

References

[Rog23] Ryan Rogers. A unifying privacy analysis framework for unknown domain algorithms in differential privacy, 2023.