PrivJail: 差分プライバシーを強制する Python ライブラリ

椎名 峻平 [1]　中谷 翔 [2]　平岡 拓海 [3]
田浦 健次朗 [4]

本稿では，プライバシーを侵害する可能性のあるデータの出力を禁止し，常に差分プライバシーが保証された結果のみを出力する Python ライブラリ，PrivJail を提案する．類似の既存システムは独自のクエリ言語や SQL に限定したものが大部分であるが，PrivJail ではデータ解析者に馴染みのある Python 上で Numpy や Pandas の記法を用いた柔軟なデータ処理が可能である．Python 上で差分プライバシーの適用を強制するため，PrivJail では (i) 複数のデータ変換操作に対する感度（sensitivity）の追跡により適切な大きさのノイズを加えることを保証し，(ii) Python 上での想定外の操作による生データの盗み見を防止する設計を行った．評価では，実際に PrivJail を用いて決定木構築アルゴリズムを実装し，プログラムの記述性と性能について議論する．

1　序論

データは閉じ込めておくだけでは価値はなく，解析処理によってはじめて価値が生まれる．しかし，解析処理ためのデータ提供にはデータ流出のリスクを伴う．特に，個人情報を含むデータの場合，プライバシー侵害のリスクが大きく，円滑なデータ流通の妨げになっている．

本稿では，個人のプライバシーを侵害し得る情報を閉じ込めたまま，プライバシー保護された解析結果のみを外部に出力することを保証する Python ライブラリ，PrivJail を提案する．個人データの管理者（curator）は自身の所有するサーバ上にデータを配置し，PrivJail を介してデータ解析者（analyst）に解析を依頼する．データ解析者はプライバシーが保証される限りにおいて Python 上で柔軟なデータ解析が可能であるが，個人データの中身を直接出力することはできない．

プライバシー保護の判定には，プライバシー漏洩について厳密な数学的定義の存在する**差分プライバシー** [6] を用いる．差分プライバシーの基本は，個人データを集計した値にランダムな摂動（ノイズ）を加えることであり，それによって任意の 1 個人の有無を結果から推測することを困難にする．詳しくは 2.1 節で述べるが，結果に加えるノイズの大きさはプライバシー保護の強度に関するパラメタ ϵ を与えることで決定される．1 回ノイズを加えて差分プライバシーを満たす結果を出力する度に，その際与えた ϵ の値が「消費」されたと見なす．その度に消費された ϵ の値は積算され，一定の**プライバシー予算**の上限を超えるとそれ以上の解析は不可能になる．

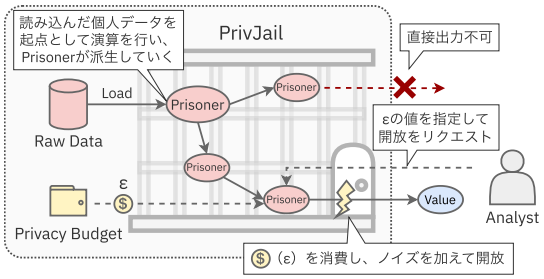

図 1　PrivJail の概念図

図 1 に PrivJail の概念図を示す．まず，PrivJail で読み込んだ個人データは「囚人（prisoner）」としてデータ解析者から隔離される．個人データから派生した値も同様に「囚人」として扱われ，差分プライバシーを満たすノイズを加えるまで出力は禁止される．データ解析者は適切な ϵ を「支払う」ことで「囚人」を開放可能であり，それによってノイズが付加された具体的な値を知ることができる．データ提供者の側では，許容するプライバシー流出の度合い，つまりプライバシー予算の上限を適切に定める必要がある．

PrivJail では，大きく分けて 2 段階の仕組みによって差分プライバシーを強制する．2 つの段階では，それぞれ (i) ライブラリが明示的に提供する呼び出しのみを用いている限り必ず差分プライバシーが保証された結果のみが出力されること，(ii) ライブラリが明示的に提供する呼び出し以外の経路で「囚人」への直接アクセスが行われないことを保証する．

段階 (i) では，各ライブラリ呼び出しが「囚人」の値をそのまま出力せず，指定した ϵ の値に応じた適切な大きさのノイズを加えた結果のみを出力することを保証する．例えば，「ある条件に合致する人数」を返すクエリと「ある年代の課金額の合計」を返すクエリはいずれも実数値を返すが，同じ ϵ の値でも加えるべきノイズの大きさは異なる．なぜなら，差分プライバシーは任意の 1 個人の有無の推測を困難にするため，任意の 1 個人の有無によって変化し得る最大の値の変化分を覆い隠す程度の大きさのノイズを加える必要があるためである．上の例では，人数に関するクエリでは 1，合計値に関するクエリでは課金額の最大値だけ 1 個人の有無によって値が変動し得る．この値をクエリの**感度**（sensitivity）と呼ぶ．実数値に対するノイズの大きさは，基本的には感度と ϵ の値の両方から決定される．

差分プライバシーを保証するには，それ単体で差分プライバシーを満たす演算の感度をライブラリ側で予め適切に定義しておけば良い．しかし，実際には単体では差分プライバシーを満たさない複数のデータ変換演算を組み合わせて適用した後に差分プライバシーを満たすノイズを加える場合が多い．その場合，個々の演算の感度からそれらを組み合わせた演算全体の感度をライブラリ側で自動で導出する必要がある．その過程を**感度追跡**と呼ぶ．

[1] 非会員　トヨタ自動車株式会社
shumpei_shiina@mail.toyota.co.jp
[2] 正会員　株式会社 SecDevLab（本研究はトヨタ自動車株式会社在籍時の実施）
sho.nakatani@secdevlab.com
[3] 非会員　株式会社 Acompany（本研究は東京大学在学時の実施）
takumi.hiraoka@acompany-ac.com
[4] 非会員　東京大学
tau@eidos.ic.i.u-tokyo.ac.jp

感度追跡は，静的な型検査によって [1, 14, 24, 29, 34]，あるいは動的に [2, 4, 15, 27] 行う手法が提案されてきた．静的な感度追跡は実行前に差分プライバシーを保証するが，Python のような動的言語への適用が困難である．そのため，PrivJail では動的な感度追跡手法を用いている．

PrivJail の 1 つの特徴はデータ解析者に馴染みのある Python ライブラリ記法を用いることが可能な点であり，本稿では特に Pandas [31] のデータフレーム演算に着目する．データフレームとは表形式の 2 次元の行・列から成るデータ構造であり，関係データベースとは異なり行および列に順序が存在する．既存の感度追跡手法はレコード間の順序を考慮しておらず，そのままでは大部分のデータフレーム演算に対応できない．そのため，本稿では，レコード（行）間の順序を考慮したデータフレーム演算の感度追跡に関する規則について議論し，その演算規則を整理する（4 節）．

段階 (ii) では，段階 (i) で想定する「囚人」に対する演算のみを許可し，その他の想定外の経路を介した「囚人」へのアクセスを禁止する．例えば，Python 上ではクラスオブジェクトのメンバへのアクセスの禁止は困難であり，秘匿すべき「囚人」の値を簡単に出力可能である．Python のような汎用言語上でこのような抜け道を塞ぐことは難しく，既存の汎用言語上のライブラリ実装 [1, 2, 4, 15, 24, 27] では必ずしも抜け道は塞がれていない．Python のサンドボックスを用いて抜け道を塞ぐ実装 [25] も存在するが，2.4 節で議論するように，Python の言語仕様や実装に依存した形で全ての抜け道を塞ぎ切ることは現実的ではない．本稿では，言語処理系の実行から「囚人」の値をプロセスレベルで隔離し，許可された関数呼び出しのみをプロセス間の Remote Procedure Call (RPC) を介して受理するシステム設計を提案する（6 節）．この設計では，言語非依存な RPC を介してのみ「囚人」に対する演算が許可され，言語処理系の実装に依存しない強力なプライバシー保護が可能になる．

本研究における主要な技術的な貢献を以下にまとめる．

- 動的プログラミング言語上における差分プライバシーの感度追跡を Pandas に代表される順序付きデータフレームに適用するための演算規則の整理（4 節）．
- 背反なデータ分割を考慮したプライバシー予算管理（5 節）．
- 言語処理系の実行から「囚人」をプロセスレベルで隔離し，RPC を介してのみ「囚人」に対する演算を許すことで攻撃界面を小さく保つプライバシー保護手法（6 節）．
- PrivJail を用いて実際に記述した差分プライバシー決定木学習 [13] プログラムの記述性と性能の評価（3 節，8 節）．

2 背景，関連研究

2.1 差分プライバシー（Differential Privacy; DP）

差分プライバシー [6] は，プライバシー保護について明確な数学的定義を持つ．まず，以下にその定義を述べる．なお，本稿で用いる主要な記号や記法は表 1 に示した．

あるデータベース D に加えた演算 M（メカニズム）の結果がある出力集合 S に含まれる確率を $P[M(D) \in S]$ と書く．ここで，D に対しある 1 個人に関する行の追加/削除を行ったデータベース D' を考える．これを**隣接データベース**と呼び，その関係を $D \sim D'$ と表す．D' に対し同様にメカニズム M を適用し，同一の出力集合 S に含まれる結果を得る確率は $P[M(D') \in S]$ であるが，この確率が $P[M(D) \in S]$ と十分に近ければ，出力結果からある個人の存在を推測することが困難になる．以上の議論を踏まえて差分プライバシーの定義式を書くと，

$$P[M(D) \in S] \leq \exp(\epsilon) \cdot P[M(D') \in S] \tag{1}$$

と表される．この不等式が任意の隣接データベース $D \sim D'$ と任意の出力集合 S に対して成り立つとき，そのメカニズム M は ϵ-**差分プライバシー**を満たす．この ϵ は数式上の ϵ の値と対応しており，プライバシー保護の度合いを示すパラメタとして与えられる．

表 1　主要な記号の意味，記法

$\mathbb{R}$, $\mathbb{R}_{>0}$, $\mathbb{Z}$, $\mathbb{N}_0$	実数，正の実数，整数，0 以上の整数全体の集合
$\mathbb{D}$	データベース全体の集合
$D \sim D'$	隣接データベースの関係（$D, D' \in \mathbb{D}$）
$\mathcal{DF}$, $\mathcal{S}$	データフレーム，シリーズ全体の集合
$\mathcal{X}$	ドメイン（ある列が取り得る値の集合）
$[k_i \mapsto v_i]$	k_i をキー，v_i を値とする辞書
$v : \tau$	型 τ の値 v
$\langle v \rangle_d$	真の値 v を秘匿し，距離 d が紐づいた秘匿値
$\langle df \rangle_d^{p,[c_i \mapsto \mathcal{X}_i]}$	行保存のタグ（p），各列のドメイン（$[c_i \mapsto \mathcal{X}_i]$，$c_i$ は列名）を持つデータフレームの秘匿値
$\langle s \rangle_d^{p,\mathcal{X}}$	行保存のタグ（p），ドメイン（$\mathcal{X}$）を持つシリーズの秘匿値
$\langle \tau \rangle$	型 τ の値を秘匿した秘匿値の型

ϵ-差分プライバシーを満たすメカニズムとしては，**ラプラスメカニズム**（Laplace mechanism）が代表的である．ラプラスメカニズムでは，ラプラス分布に従うランダムノイズが実数値に対して加えられる．ノイズの大きさは，結果の実数値を出力する関数の**感度**（sensitivity）と ϵ の値によって決定される．データベースを入力として実数値を返す関数 $f : \mathbb{D} \to \mathbb{R}$ の感度 Δf は，任意の隣接データベース $D \sim D'$ に f を適用した結果 $f(D)$，$f(D')$ の値の差の最大値として定義される．すなわち，感度は以下の式で定義される [6].

$$\Delta f = \max_{D \sim D'} |f(D) - f(D')| \tag{2}$$

なお，[6] では関数の出力は実数ベクトルであるが，ここでは簡単のためスカラー値としている．感度 Δf の関数 f に対してラプラス分布 $Lap(\Delta f / \epsilon)$ に従うランダムノイズを加えると ϵ-差分プライバシーを満たすことが知られている．

感度はラプラスメカニズム以外のメカニズムでも用いられる．例えば**指数メカニズム**（exponential mechanism）[26] は N 個の選択肢の中から確率的に 1 つ選ぶ演算である．N 個の選択肢をそれぞれ k_i とし，k_i に対して関数 f を適用した結果の実数値 $f(k_i)$ を k_i が選ばれる確率の重みとして用いる．具体的には，関

数 f の感度を Δf として，$\exp(\epsilon f(k_i)/2\Delta f)$ に比例する確率で k_i を選べば ϵ-差分プライバシーを満たす．

2.2 感度（sensitivity）の定義の一般化

上で述べた通り，感度を実数で定義可能な限り，任意の関数 f に対してラプラスメカニズムや指数メカニズムを適用することで ϵ-差分プライバシーを満たすメカニズムを構成できる．しかし，実際には関数 f は単体で与えられず，複数の演算を組み合わせて表現されることが多い．その場合，元のデータベースを起点として加えられた演算全体の感度を導出する必要がある．

最終的に求めたい感度は関数 $F : \mathbb{D} \to \mathbb{R}$ の感度 ΔF であり，途中過程の演算から全体の感度 ΔF を導出する事を考える．Reed & Pierce [34] は任意の入出力型を取る関数 $f : \tau_1 \to \tau_2$ に対して一般化した感度の定義を導入し，途中過程の演算に対しても感度を定義可能にした．型 τ_1，τ_2 の定義域における距離関数をそれぞれ $d_{\tau_1}(x, x')$，$d_{\tau_2}(x, x')$ とすると，

$$\forall x, x'. \left(d_{\tau_2}(f(x), f(x')) \leq c \cdot d_{\tau_1}(x, x')\right) \tag{3}$$

ならば，f の感度は高々 c である（*c-sensitive* である）．これは，入力 x，x' の距離が関数適用前後で高々 c 倍にしか変化しないことを表す．個々の演算に対してこの一般化した感度が定義されていれば，全体の演算 $F : \mathbb{D} \to \mathbb{R}$ の感度 ΔF を導出可能になる．具体的な感度追跡手法については 4 節で述べる．

2.3 合成定理・プライバシー予算

あるデータベース D に対して ϵ_1-差分プライバシーを満たすメカニズム M_1 と ϵ_2-差分プライバシーを満たすメカニズム M_2 を逐次的に適用した場合，全体では $(\epsilon_1 + \epsilon_2)$-差分プライバシーを満たす．これを**逐次合成定理**（sequential composition theorem）と言う．この性質により，インタラクティブにクエリを実行した場合に各ステップにおける ϵ の値を積算することで全体のプライバシー予算を管理することが可能になる．

あるデータベース D を（個人の重複がないように）背反に分割した結果をそれぞれ D_1，D_2 とすると，ϵ_1-差分プライバシーを満たすメカニズム M_1 を D_1 に，ϵ_2-差分プライバシーを満たすメカニズム M_2 を D_2 にそれぞれ適用した場合，全体では $\max(\epsilon_1, \epsilon_2)$-差分プライバシーを満たす．これを**並列合成定理**（parallel composition theorem）と言う．この性質を活用することで全体の ϵ の消費を抑えることができる．

2.4 関連研究

Diffprivlib [19] や PyDP [11] などのライブラリは差分プライバシーを満たすメカニズムを部品として提供する．しかし，それらの部品を正しく扱うことはユーザの責任であり，プログラム全体の差分プライバシーを保証しない．実際に，ある差分プライバシーのツールのユーザビリティスタディではユーザによる間違いが発生しやすいことが報告されている [30]．以下では，複数の演算を柔軟に組み合わせた場合にも全体として差分プライバシーを保証するシステムについて議論する．

差分プライバシーを保証する SQL 処理系 [5, 20, 22, 37] はその一例である．SQL では関係代数の演算の組み合わせによって宣言的にクエリが表現されるが，SQL の表現能力は限定的である．例えば，3.3 節に示す決定木学習の例を SQL で記述することは難しい．

差分プライバシーを保証するプログラミング言語としては，Fuzz [17, 34] やその後継の DFuzz [14]，Duet [29] などが提案された．これらの言語では，静的な型システムによって感度を追跡し，プログラム全体として差分プライバシーを満たすことを保証する．しかし，これらの型システムは高度な線形型を必要とし，現在主要なプログラミング言語上での実現が難しい．近年，線形型を必要とせず感度追跡を行う型システムとして Solo [1] や Spar [24] が提案され，いずれも Haskell 上のライブラリとして実装されている．しかし，いずれにせよ Python をはじめとする動的な言語に適用することは難しい．

プログラムの実行中に動的に差分プライバシーの保証を行うライブラリとして，C#では PINQ [27]，Python では DDUO [2]，OpenDP [15]，Tumult Analytics [4] などが存在する．これらのシステムでは，入力のデータベースを起点として適用される演算の感度を動的に追跡し，演算全体の感度を決定する．PrivJail ではこれらの動的感度追跡の考え方をベースに設計されているが，これらのシステムと異なり，データフレームの行順序を考慮した感度の追跡が可能である．PINQ や OpenDP はレコード間の順序のない関係データベースのみを扱っている．DDUO や Tumult Analytics は外形的にはデータフレームを提供するが，実質的に行順序のないデータフレームとして扱っており，提供されているデータフレーム演算の種類は限定的である．

Antigranular [25] は Pandas ライクなデータフレーム演算を提供しており，表面的には PrivJail と類似した Python ライブラリである．しかし，Antigranular は感度追跡を行っておらず，基本的には単体で差分プライバシーを満たす演算のみが提供されており，複数の演算の柔軟な組み合わせが難しい．

次に，各システムにおけるデータ解析者への信頼度の違いと，それに応じたシステム上の設計の違いを議論する．まず，入力のクエリを SQL や独自のプログラミング言語（Fuzz [17, 34] など）に制限する場合は，データ解析者を信頼しないシステムを構成しやすい．一方，汎用言語上のライブラリ [1, 2, 24, 27] ではデータ解析者をある程度信頼し，意図しない差分プライバシー実装の誤りを防ぐことを目的とすることが多い（*Honest-but-fallible* モデル [1, 2]）．このようなシステムでは，秘匿された値への直接アクセスの抜け道は必ずしも塞がれていない．

Antigranular [25] は Python 上のライブラリであるが，上述の抜け道を塞ぐよう設計されている．Antigranular では，想定外の操作を禁止するため，Python のサンドボックス実行環境である RestrictedPython [12] 上でユーザの与えたスクリプトを実行する．しかし，これはサンドボックス実装の正しさに大きく依存した手法であり，安全性が低い．実際に，現時点で RestrictedPython には脆弱性が数件報告されている [12]．一般に，Python のように元々セキュアに設計されていない汎用言語上のサンドボックスは実現が難しい．なぜなら，言語仕様や処理系実装に大きく依存するため攻撃界面が大きく，言語自体の進化への追従が難しいためである．PrivJail では，6 節で詳しく述べる通り，言語処理系に依存しない形で想定外の抜け道を塞いでいる．PrivJail の脅威モデルは 7 節 で述べる．

3 PrivJail の概要

本節では，PrivJail ライブラリの概要と基本的な使用方法を説明し，決定木学習のプログラムを例に PrivJail の記述性について議論する．

3.1 PrivJail のスコープ

PrivJail の扱うデータは Pandas データフレームとして読み込み可能な形式（CSV など）として，ある 1 個人がある 1 行と一対一対応すること（event-level/row-level DP）を想定する．現時点では 1 個人が複数行に対応する場合（user-level DP [8]）を想定しない．また，高度な逐次合成定理（advanced composition [9] など）や近似差分プライバシー [7] を考えず，純粋な差分プライバシー（pure DP）の範囲で議論する．

3.2 PrivJail ライブラリの基本的な使用方法

PrivJail で Pandas ライクな記法を用いてデータフレーム演算を記述するには，Python 上で以下のようにインポートする．

```
import privjail as pj
from privjail import pandas as pd
```

PrivJail の Pandas API は可能な範囲でオリジナルの Pandas API を踏襲する．ただし，差分プライバシーを保証できない演算を除く必要がある他，ϵ などの追加の引数が必要になる演算も存在するため，完全には Pandas 互換ではない．

ある CSV ファイルを読み込むには，

```
>>> df = pd.read_csv("filename.csv")
```

とする．戻り値のデータフレームは `PrivDataFrame` クラスのオブジェクトであり，具体的な中身を直接出力できない．

```
>>> print(df)
Prisoner(<class 'pandas.core.frame.DataFrame'>, distance=1)
```

このような値を PrivJail では**秘匿値**と呼び，その他の通常の値（**公開値**）と区別する．秘匿値のクラス（例えば `PrivDataFrame`）は `Prisoner` クラスを継承する．秘匿値のデータフレームを特に**秘匿データフレーム**と呼ぶ．秘匿値には距離（`distance`）の情報が紐づけられるが，距離の詳細については 4 節で述べる．

秘匿データフレームの行数を取得するには，

```
>>> df.shape[0]
Prisoner(<class 'int'>, distance=1)
```

とする．`df.shape` は行数と列数の組であり，行数（`shape[0]`）は `int` 型の秘匿値である．秘匿値から値を出力するには，差分プライバシーを保証するメカニズムを適用する．ラプラスメカニズムを適用してデータフレームの行数を出力するには，

```
>>> pj.laplace_mechanism(df.shape[0], eps=0.1)
32561.65454862431
```

とする．ここでは ϵ の値として 0.1 を指定した．この出力は単なる float 型の値であり，以降は公開値として使用できる．

他にも，例えば 40 歳を超す人数をカウントするには以下のように Pandas ライクなフィルタリングの記述が可能である．

```
>>> pj.laplace_mechanism(df[df["age"] > 40].shape[0], eps=0.1)
13445.401235025374
```

各データソースに対して消費されたプライバシー予算は

```
>>> pj.consumed_privacy_budget()
{'filename.csv': 0.2}
```

の呼び出しで確認できる．現段階では未実装であるが，適切なユーザ認証を行い，データ管理者が予め設定した単位（ユーザ/グループ）ごとにプライバシー予算を管理することを想定している．プライバシー予算の上限を設定し，上限を超えるプライバシー予算の消費を禁止することも可能である．

次に，`mean()` に ϵ の値を与えて年齢の平均を計算する．

```
>>> df["age"].mean(eps=0.1)
privjail.util.DPError: The domain is unbounded. Use clip().
```

しかし，定義域（ドメイン）が有界でない旨のエラーが発生する．これは，age 列の値の下限と上限が不明であり，感度を定義できないことを意味する．実際のデータの最小値と最大値をそのまま用いることはそれ自体が差分プライバシー違反である．

そこで，例えば `clip()` で値の下限と上限を明示的にユーザが与えることで有界な定義域を設定し，平均値を求める．

```
>>> df["age"].clip(0, 120).mean(eps=0.1)
38.52147065037341
```

この例では年齢の値を 0 歳から 120 歳の間にクリッピングし，ノイズを加えた上で平均年齢を出力している．

秘匿データフレームの各列には定義域が紐づいており，`df.domains` で参照可能である．なお，明示的にクリッピングを行う他にも，各列のスキーマを記述したファイル（json ファイルなど）を別途配置しておき，それをデータソースと紐づけて定義域として読み込むことも可能である．スキーマファイルには，列ごとの値の型，実数の範囲，カテゴリ値の取り得る値のリストなどを定義可能である．

合計値を求める演算 `sum()` も同様に有界な定義域を必要とするが，`sum()` では秘匿値を返すのに対し，`mean()` は ϵ の値を受け取り ϵ-差分プライバシーを満たした公開値を返す．このインターフェースの違いは，`sum()` では感度を有用な形でバウンド可能な一方，`mean()` では感度を有用な形で（十分小さく）定義することが難しいためである．そのため，`mean()` は秘匿値を経由せず，差分プライバシーを満たす平均値アルゴリズム [23] を直接用いることで有用な結果を得る．

3.3 決定木学習の PrivJail プログラム例

より実用的な PrivJail のプログラム例として，図 2 に差分プライバシー決定木構築アルゴリズム（DiffPID3 [13]）を忠実に再現したプログラムを示す．入力の秘匿データフレーム（`df`）の要素は全てカテゴリ値であり，定義域は定義済みとする．

基本的には通常の Pandas プログラムと記法は同一であるが，部分的に PrivJail による拡張を用いている．例えば，差分プライバシーの適用（`pj.laplace_mechanism()` や `pj.exponential_mechanism()`）や，`df.domains` を介した各カラムのカテゴリ数の取得などは PrivJail による拡張である．

このプログラムのポイントを整理すると，

- 具体的な入力データに応じた条件分岐を含む再帰処理であり，静的なプログラム解析やトレーシングが困難なため，動

```python
def DiffPID3(df, attrs, class_attr, d, B):
    # df         : 入力のPrivDataFrame（すべてカテゴリ値）
    # attrs      : 説明変数の属性名（カラム名）の集合
    # class_attr : 目的変数の属性名（カラム名）
    # d          : 決定木の深さの最大値
    # B          : 全体のプライバシー予算
    eps = B / (2*(d+1))
    return build_DiffPID3(df, attrs, class_attr, d, eps)

def build_DiffPID3(df, attrs, class_attr, d, eps):
    t = max([len(df.domains[a].categories) for a in attrs])
    C = len(df.domains[class_attr].categories)
    N = noisy_count(df, eps)

    if len(attrs) == 0 or d == 0 or N/(t*C) < (2**0.5)/eps:
        # 葉ノードにclass_attr属性の最頻値を記録
        class_counts = {c: noisy_count(df_c, eps) \
                        for c, df_c in df.groupby(class_attr)}
        best_class = max(class_counts, key=class_counts.get)
        return LeafNode(best_class)

    # 最良の分割を与える属性（best_attr）を選択
    qs = {a: quality_fn(df, a, class_attr) for a in attrs}
    best_attr = pj.exponential_mechanism(qs, eps=eps)

    # best_attr属性の値に応じてデータを分割し，再帰的に木を生成
    node = InnerNode(best_attr)
    for category, df_child in df.groupby(best_attr):
        child_node = build_DiffPID3(df_child, \
                attrs - {best_attr}, class_attr, d - 1, eps)
        node.add_child(category, child_node)
    return node

# PrivDataFrameの行数にノイズを加えた0以上の実数を出力
def noisy_count(df, eps):
    return max(0, pj.laplace_mechanism(df.shape[0], eps=eps))

# Quality functionとしてMax operatorを実装
def quality_fn(df, split_attr, class_attr):
    s = 0
    for _, df_c in df.groupby(split_attr):
        s += df_c[class_attr].value_counts(sort=False).max()
    return s
```

図 2　PrivJail で決定木学習（DiffPID3 [13]）を記述する例

的な差分プライバシーの保証が有効である．

- 感度追跡（4 節）により，秘匿値を秘匿したまま複数の演算を適用した上で差分プライバシーを満たすノイズを加えることが可能になっている．例えば，`quality_fn()` では秘匿値同士の演算が複数組み合わされ，その結果を秘匿値のまま指数メカニズムの引数として与えている．
- `df.groupby()` で指定した列のカテゴリ値に応じてデータフレームを排他的に分割し，分割後の各データフレームに対して再帰呼び出しを行っている．期待通りプライバシー予算 `B` を全体で消費したと判断するには，排他的なデータ分割を考慮した感度追跡（4.3 節），および並列合成定理を考慮したプライバシー予算管理（5 節）が必要になる．

4　PrivJail における感度追跡

本節では，動的な感度追跡手法の基本的な考え方を説明した後，実用的なデータフレーム処理を記述するために必要になる演算に対して感度追跡手法を拡張し，演算規則を整理する．

4.1　動的な感度追跡の概要

まず，既存の動的な感度追跡手法 [2,4,15,27] をベースとした一般的な感度追跡について，演算規則を導入する．感度は 1 つの秘匿値を入力として 1 つの秘匿値を返す関数に対して定義されるが，複数の秘匿値を入力とするような関数に対しては定義できない．そこで，本稿では，入力と出力の秘匿値に紐づけられた**距離**の対応関係を演算規則として用いる．ここでいう距離とは，隣接データベースが入力として与えられた場合にその秘匿値の真の値が最大で離れ得る距離のことである．距離関数は 2.2 節で議論したように，型ごとに定義される．

秘匿値を真の値 v と距離 d の組 $\langle v\rangle_d$ で表し，型 τ の値を秘匿した秘匿値の型を $\langle\tau\rangle$ と表す（表 1 を参照）．ある秘匿値の表記 $\langle v\rangle_d : \langle\tau\rangle$ は，その真の値が $v : \tau$ であり，任意の隣接データベース $D \sim D'$ を入力とした演算の結果 v，v' の距離 $d_\tau(v, v')$ が最大で d であることを表している．すなわち，入力データベース D を入力として v を得るまでに適用された一連の演算を合成した関数を f とすると，

$$\forall D, D' \in \mathbb{D}.\big(D \sim D' \implies d_\tau(f(D), f(D')) \le d\big) \tag{4}$$

が成り立つことを表す．これは関数 $f : \mathbb{D} \to \mathbb{R}$ に対する感度の定義（式 (2)）と対応しており，関数 f の感度が $\Delta f \le d$ であることを示している．よって，秘匿値 $\langle v\rangle_d : \langle\mathbb{R}\rangle$ に対して感度を d としてラプラスメカニズムを適用することで差分プライバシーを満たすことが保証される．このとき，ノイズを加える時点で具体的な関数 f の構成を知ることは不要であり，秘匿値だけを見て感度を決定できる．なお，距離が 0 である秘匿値 $\langle v\rangle_0 : \langle\tau\rangle$ は公開値と同義であり，単に $v : \tau$ と表記する．

データベース D を読み込んだとき，そのデータベースの値は $\langle D\rangle_1 : \langle\mathbb{D}\rangle$ と表され，これが秘匿値の初期値となる．ここで，距離を 1 としているのは隣接データベースを考えるためである．秘匿値の初期値 $\langle D\rangle_1$ を起点として，加えられる演算に応じて秘匿値が派生していく．以下の説明では，プログラム中で読み込まれるデータベースはただ 1 つであることを仮定する．

まず，1 入力 1 出力の秘匿値の演算規則を以下に表す．

$$\langle v_1\rangle_{d_1} : \langle\tau_1\rangle \xrightarrow{f} \langle v_2\rangle_{d_2} : \langle\tau_2\rangle$$

関数 f の表記は必要に応じて省略される．演算規則中の距離 d_1，d_2 は以下の条件を満たす必要がある．

$$\forall x, x'.\big(d_{\tau_1}(x, x') \le d_1 \implies d_{\tau_2}(f(x), f(x')) \le d_2\big) \tag{5}$$

これは，出力の秘匿値についても式 (4) を満たす必要があるためである．特に，式 (3) が成り立つ，すなわち関数 f の感度が c ならば，$\langle v_1\rangle_d : \langle\tau_1\rangle \xrightarrow{f} \langle v_2\rangle_{c\cdot d} : \langle\tau_2\rangle$ と書ける．

これを複数の秘匿値を入力とする演算に拡張する．

$$\langle v_1\rangle_{d_1} : \langle\tau_1\rangle, \ldots, \langle v_n\rangle_{d_n} : \langle\tau_n\rangle \xrightarrow{f} \langle v_o\rangle_{d_o} : \langle\tau_o\rangle$$

式 (5) に対応する入出力の距離の条件を以下に示す．

$$\begin{aligned}\forall x_1, x_1', \ldots, x_n, x_n'.\big(&\\ d_{\tau_1}(x_1, x_1') \le d_1 \wedge \ldots \wedge d_{\tau_n}(x_n, x_n') \le d_n&\\ \implies d_{\tau_o}(f(x_1, \ldots, x_n), f(x_1', \ldots, x_n')) \le d_o\big)&\end{aligned} \tag{6}$$

例として，表 2 の上部に実数の秘匿値に対する演算規則を示す．これらの演算規則はいずれも式 (6) を満たす．なお，実数の

秘匿値同士の乗算については出力の距離がバウンドできないため，秘匿値と公開値の乗算のみが定義されている．

ラプラスメカニズムの演算規則は，

$$\langle v \rangle_d : \langle \mathbb{R} \rangle,\ \epsilon : \mathbb{R}_{>0} \ \rightsquigarrow\ v + Lap(d/\epsilon) : \mathbb{R}$$

と表される．秘匿値を出力する演算規則とは異なり，“$\rightsquigarrow$”で確率的な演算を表す．ここでは，秘匿値の距離 d を感度としてラプラスノイズを加えた値を出力している．出力は ϵ-差分プライバシーを満たす値であり，以降は公開値として扱われる．

4.2 データフレーム演算に対する感度追跡

次に，Pandas ライクなデータフレームを扱うため，データフレームに対する感度追跡について議論する．

4.2.1 データフレーム間の距離の定義

ここでは，データフレーム演算を含めた感度追跡のため，データフレーム間の距離関数を定義する．Fuzz [34] などではデータベースは順序なしの行の集合（multiset）として表され，データベース間の距離は集合の対称差の要素数で表される．しかし，多くのデータフレーム実装は順序を持つ行のリストとして扱われる [33]．例えば，Pandas には `df.iloc[a:b]` などのスライス記法を用いて a から b 行目の行を取得する演算が存在し，行の順序を考慮した距離の定義が必要になる．

そこで，PrivJail では編集距離を用いてデータフレーム間の距離を定義する．ここで言う編集距離とは，文字の一致をデータフレームの行の一致に置き換えたものである．特に，1 行の追加/削除で距離が 1 だけ増加する LCS (Longest Common Subsequence) 距離を用いる．LCS 距離では 1 行の置換には追加と削除の 2 ステップ必要であり，距離が 2 増加する．この定義は，1 行の追加/削除で隣接データベースを定義する Unbounded DP の定義 [21] と整合する．LCS 距離の定義を用い，データフレーム $df_1, df_2 \in \mathcal{DF}$ 間の距離を以下に定義する．

$$d_{\mathcal{DF}}(df_1, df_2) = |df_1| + |df_2| - 2 \cdot |LCS(df_1, df_2)|$$

ここで，$|df|$ はデータフレームの行数，$LCS(df_1, df_2)$ は df_1，df_2 間の最長共通部分列を表す．

この LCS 距離によるデータフレーム間の距離の定義を用いれば，例えばフィルタリング演算の感度は順序なしデータベースと同様に 1 と定義できる．加えて，ある行番号の範囲に含まれる行を取得する演算（`df.iloc[a:b]`，`df.head()`，`df.tail()` など）の感度は 2 であると分かる．これは，1 行の追加/削除によって，ある固定の行番号の範囲に対し最大で 1 行の追加と 1 行の削除が同時に発生するためである．ソートについても，安定ソートであれば入力の 1 行の追加/削除は他の行の順序に影響を与えないため，感度は 1 であると分かる．

LCS 距離の定義により，以上の議論を踏まえると，例えば「年収上位 100 人の平均年齢」を出力するクエリを以下のように標準的な Pandas 記法を用いて記述可能になる．

```
df.sort_values("income").tail(100)["age"].mean(eps=0.1)
```

4.2.2 データフレーム同士の行単位演算

次に，データフレーム同士の演算について議論する．例えば，3.2 節で紹介したクエリ `df[df["age"] > 40].shape[0]` は，以下の演算の組み合わせとして分解できる．

```
s1  = df["age"] # 列名による列（シリーズ）の選択
s2  = s1 > 40   # シリーズの各要素ごとにスカラー値（40）との比較
df1 = df[s2]    # bool値のシリーズによるdfの行のフィルタリング
df1.shape[0]    # フィルタリング後のデータフレームの行数の取得
```

Pandas では単一の列から成るデータフレームを特にシリーズ（series）と呼ぶ．上の `df[s2]` はデータフレームに対して各要素の値が bool 値であるシリーズを与え，値が True である位置にある行のみをフィルタリングする演算である．しかし，例えばある $df \in \mathcal{DF}$ に対して距離が 1 だけ離れているシリーズ $s, s' \in \mathcal{S}$ を考えると，$df[s]$ と $df[s']$ の距離はバウンドできないことが分かる．なぜなら，例えば s の偶数番目の要素は True，奇数番目は False として，s' は s の先頭に 1 要素を追加したものとすると，s と s' の距離は 1 であるが，$df[s]$ と $df[s']$ はほぼ全ての行が異なるデータフレームとなるためである．

このフィルタリング記法は Pandas プログラムでは頻出の表現である．他にも `df1 + df2` などの要素ごとの演算など，データフレーム（シリーズ）同士で行単位の演算を行う演算では同様の問題が存在する．出力の距離がバウンドできなければこれらの演算を PrivJail で扱うことができないため，条件付きで距離をバウンド可能にすることを考える．

そこでまず，データフレームの「行が保存」する関係を定義する．関数 f の演算規則が以下のように表されるとすると，

$$\langle df_1 \rangle_{d_1} : \langle \mathcal{DF} \rangle, \ldots, \langle df_n \rangle_{d_n} : \langle \mathcal{DF} \rangle \xrightarrow{f} \langle df_o \rangle_{d_o} : \langle \mathcal{DF} \rangle$$

入力 $df_1, \ldots, df_n$ が全て「行が保存」した関係にある場合，出力のデータフレーム df_o も同様に「行が保存」した関係を持つための条件は以下の通り定義される．

- 行単位の演算であり，$df_1, \ldots, df_n$ の i 行目を入力とし，df_o の i 行目を出力とする．
- i 行目の演算において，i 行目以外の行の値や行番号 i を用いていない．

すなわち，ある 2 つのデータフレームが「行が保存」した関係にあるならば，その 2 つのデータフレームの行数は等しく，同一の行の対応関係を持つ．行が保存したデータフレーム間で行単位の演算を行う場合，出力の距離をバウンド可能になる．その詳しい理由は付録 付録 A に述べる．

演算規則中では，行が保存した関係にある秘匿データフレーム（またはシリーズ）に対し同一のタグ p を割り当て，$\langle df \rangle^p_d : \langle \mathcal{DF} \rangle$（または $\langle s \rangle^p_d : \langle \mathcal{S} \rangle$）と表す．例えば，データフレームの要素ごとの加算（`df1 + df2`）の演算規則は

$$\langle df \rangle^p_d : \langle \mathcal{DF} \rangle, \langle df' \rangle^p_d : \langle \mathcal{DF} \rangle \rightarrow \langle df + df' \rangle^p_d : \langle \mathcal{DF} \rangle$$

と表される．これは，入力のデータフレームは同一のタグ p を持っている必要があり，そのタグは出力のデータフレームにも引き継がれることを表している．もし異なるタグを持つデータフレームが入力された場合はその時点でエラーを返す．なお，同一のタグ p を持っているならば距離 d も保存する．一方，出力の行

表 2　PrivJail における秘匿値の演算規則（単純化したものを一部のみ掲載）

実数に対する演算	
加算 (+)	$\langle v\rangle_d : \langle\mathbb{R}\rangle,\ \langle v'\rangle_{d'} : \langle\mathbb{R}\rangle \ \rightarrow\ \langle v+v'\rangle_{d+d'} : \langle\mathbb{R}\rangle$
乗算 (*)	$\langle v\rangle_d : \langle\mathbb{R}\rangle,\ v' : \mathbb{R} \ \rightarrow\ \langle v\cdot v'\rangle_{d\cdot\lvert v'\rvert} : \langle\mathbb{R}\rangle$
最大値 (pj.max())	$\langle v\rangle_d : \langle\mathbb{R}\rangle,\ \langle v'\rangle_{d'} : \langle\mathbb{R}\rangle \ \rightarrow\ \langle \max(v,v')\rangle_{\max(d,d')} : \langle\mathbb{R}\rangle$
最小値 (pj.min())	$\langle v\rangle_d : \langle\mathbb{R}\rangle,\ \langle v'\rangle_{d'} : \langle\mathbb{R}\rangle \ \rightarrow\ \langle \min(v,v')\rangle_{\max(d,d')} : \langle\mathbb{R}\rangle$
ラプラスメカニズム (pj.laplace_mechanism())	$\langle v\rangle_d : \langle\mathbb{R}\rangle,\ \epsilon : \mathbb{R}_{>0} \ \rightsquigarrow\ v + Lap(\Delta/\epsilon) : \mathbb{R}$ where $d=(\eta, C),\ \Delta = \max\eta$ subject to C
指数メカニズム (pj.exponential_mechanism())	$[k_j \mapsto \langle v_j\rangle_{d_j}] : dict[\tau, \langle\mathbb{R}\rangle],\ \epsilon : \mathbb{R}_{>0} \ \rightsquigarrow\ k^{best} : \tau$ where $d_j=(\eta_j, C_j),\ \Delta_j = \max\eta_j$ subject to $C_j,\ \Delta = \max_j \Delta_j,$ $k^{best} \leftarrow$ choose k_j with probability proportional to $\exp(\epsilon v_j/2\Delta)$
データフレームに対する演算	
CSV の読み込み (pd.read_csv())	$filename : str,\ [c_i \mapsto \mathcal{X}_i] : dict[str, domain] \ \rightarrow\ \langle df\rangle_1^{p^{fresh},[c_i\mapsto\mathcal{X}_i]} : \langle\mathcal{DF}\rangle$
Shape の取得 (df.shape)	$\langle df\rangle_d^{p,[c_i\mapsto\mathcal{X}_i]} : \langle\mathcal{DF}\rangle \ \rightarrow\ (\langle n^{rows}\rangle_d,\ n^{cols}) : \langle\mathbb{N}_0\rangle\times\mathbb{N}_0$ where $(n^{rows}, n^{cols}) \leftarrow df.shape$
列の取得 (df[column_name])	$\langle df\rangle_d^{p,[c_i\mapsto\mathcal{X}_i]} : \langle\mathcal{DF}\rangle,\ c_k : str \ \rightarrow\ \langle df[c_k]\rangle_d^{p,\mathcal{X}_k} : \langle\mathcal{S}\rangle$
列の代入 (df[column_name] = s)	$\langle df\rangle_d^{p,[c_i\mapsto\mathcal{X}_i]} : \langle\mathcal{DF}\rangle,\ c_k : str,\ \langle s\rangle_d^{p,\mathcal{X}} : \langle\mathcal{S}\rangle \ \rightarrow\ \langle df'\rangle_d^{p,[c_{i(\neq k)}\mapsto\mathcal{X}_i, c_k\mapsto\mathcal{X}]} : \langle\mathcal{DF}\rangle$
セルごとの加算 (df1 + df2)†	$\langle df\rangle_d^{p,[c_i\mapsto\mathcal{X}_i]} : \langle\mathcal{DF}\rangle,\ \langle df'\rangle_d^{p,[c_i\mapsto\mathcal{X}'_i]} : \langle\mathcal{DF}\rangle \ \rightarrow\ \langle df+df'\rangle_d^{p,[c_i\mapsto\mathcal{X}''_i]} : \langle\mathcal{DF}\rangle$ where $\mathcal{X}''_i = \{x+x' \mid x\in\mathcal{X}_i, x'\in\mathcal{X}'_i\}$
セルごとの比較 (df1 == df2)†	$\langle df\rangle_d^{p,[c_i\mapsto\mathcal{X}_i]} : \langle\mathcal{DF}\rangle,\ \langle df'\rangle_d^{p,[c_i\mapsto\mathcal{X}'_i]} : \langle\mathcal{DF}\rangle \ \rightarrow\ \langle df == df'\rangle_d^{p,[c_i\mapsto bool]} : \langle\mathcal{DF}\rangle$
行のフィルタリング (df[s_bool])†	$\langle df\rangle_d^{p,[c_i\mapsto\mathcal{X}_i]} : \langle\mathcal{DF}\rangle,\ \langle s\rangle_d^{p,bool} : \langle\mathcal{S}\rangle \ \rightarrow\ \langle df[s]\rangle_d^{p^{fresh},[c_i\mapsto\mathcal{X}_i]} : \langle\mathcal{DF}\rangle$
クリッピング (df.clip(a, b))†	$\langle df\rangle_d^{p,[c_i\mapsto\mathcal{X}_i]} : \langle\mathcal{DF}\rangle,\ a:\mathbb{R},\ b:\mathbb{R} \ \rightarrow\ \langle df^{clipped}\rangle_d^{p,[c_i\mapsto\mathcal{X}'_i]} : \langle\mathcal{DF}\rangle$ if $\forall i.\ \mathcal{X}_i\subseteq\mathbb{R}$ where $df^{clipped} \leftarrow df.clip(a,b),\ \mathcal{X}'_i = \{x\in\mathcal{X}_i \mid a\le x\le b\}$
列の合計値 (df.sum(axis=0))	$\langle df\rangle_d^{p,[c_i\mapsto\mathcal{X}_i]} : \langle\mathcal{DF}\rangle \ \rightarrow\ \mathcal{S}[c_i \mapsto \langle v_i^{sum}\rangle_{d_i}] : \mathcal{S}$ if $\forall i.\ \mathcal{X}_i\subseteq\mathbb{R}\wedge\mathcal{X}_i$ is bounded where $\mathcal{S}[c_i\mapsto v_i^{sum}] \leftarrow df.sum(axis=0),\ d_i = d\cdot\max(\lvert\sup\mathcal{X}_i\rvert, \lvert\inf\mathcal{X}_i\rvert)$
列の平均値 (df.mean(axis=0))	$\langle df\rangle_d^{p,[c_i\mapsto\mathcal{X}_i]} : \langle\mathcal{DF}\rangle,\ \epsilon:\mathbb{R}_{>0} \ \rightsquigarrow\ \mathcal{S}[c_i\mapsto v_i^{mean}] : \mathcal{S}$ if $\forall i.\ \mathcal{X}_i\subseteq\mathbb{R}\wedge\mathcal{X}_i$ is bounded where $d=(\eta,C),\ \Delta=\max\eta$ subject to $C,\ \Delta^{rows}=\Delta,\ \Delta_i^{sum} = \Delta\cdot\max(\lvert\sup\mathcal{X}_i\rvert, \lvert\inf\mathcal{X}_i\rvert),$ $(n^{rows}, n^{cols}) \leftarrow df.shape,\ \mathcal{S}[c_i\mapsto v_i^{sum}] \leftarrow df.sum(axis=0),\ \epsilon' = \epsilon/(n^{cols}+1),$ $n^{rows\prime} \leftarrow n^{rows} + Lap(\Delta^{rows}/\epsilon'),\ v_i^{mean} \leftarrow (v_i^{sum} + Lap(\Delta_i^{sum}/\epsilon'))/n^{rows\prime}$
Slice による行の取得 (df.iloc[a:b])†	$\langle df\rangle_d^{p,[c_i\mapsto\mathcal{X}_i]} : \langle\mathcal{DF}\rangle,\ (a,b):\mathbb{Z}\times\mathbb{Z} \ \rightarrow\ \langle df[a:b]\rangle_{2\cdot d}^{p^{fresh},[c_i\mapsto\mathcal{X}_i]} : \langle\mathcal{DF}\rangle$
安定ソート (df.sort_values(column_name))†	$\langle df\rangle_d^{p,[c_i\mapsto\mathcal{X}_i]} : \langle\mathcal{DF}\rangle,\ c_k : str \ \rightarrow\ \langle df^{sorted}\rangle_d^{p^{fresh},[c_i\mapsto\mathcal{X}_i]} : \langle\mathcal{DF}\rangle$ where $df^{sorted} \leftarrow df.sort\_values(c_k, kind = \text{"}stable\text{"})$
列の値によるデータフレーム分割 (df.groupby(column_name))‡	$\langle df\rangle_d^{p,[c_i\mapsto\mathcal{X}_i]} : \langle\mathcal{DF}\rangle,\ c_k : str \ \rightarrow\ [v_j \mapsto \langle df_j\rangle_{d_j}^{p_j^{fresh},[c_i\mapsto\mathcal{X}_i^{(j)}]}] : groups[\langle\mathcal{DF}\rangle]$ if $\mathcal{X}_k$ is finite where $v_j\in\mathcal{X}_k,\ df_j \leftarrow \mathcal{DF}[row\in df \mid row[c_k] = v_j],\ \mathcal{X}_k^{(j)} = \{v_j\},\ \mathcal{X}_{i(\neq k)}^{(j)} = \mathcal{X}_i,$ $d=(\eta,C),\ d_j = (\eta_j, C\cup\{\sum_j\eta_j\le\eta\}),\ \eta_j \leftarrow$ a fresh distance variable
シリーズに対する演算	
Shape の取得 (s.shape)	$\langle s\rangle_d^{p,\mathcal{X}} : \langle\mathcal{S}\rangle \ \rightarrow\ (\langle n^{rows}\rangle_d,) : \langle\mathbb{N}_0\rangle\times\emptyset$ where $(n^{rows},) \leftarrow s.shape$
合計値 (s.sum())	$\langle s\rangle_d^{p,\mathcal{X}} : \langle\mathcal{S}\rangle \ \rightarrow\ \langle s.sum()\rangle_{d\cdot\max(\lvert\sup\mathcal{X}\rvert,\lvert\inf\mathcal{X}\rvert)} : \langle\mathbb{R}\rangle$ if $\mathcal{X}\subseteq\mathbb{R}\wedge\mathcal{X}$ is bounded
平均値 (s.mean())	$\langle s\rangle_d^{p,\mathcal{X}} : \langle\mathcal{S}\rangle,\ \epsilon:\mathbb{R}_{>0} \ \rightsquigarrow\ v^{mean} : \mathbb{R}$ if $\mathcal{X}\subseteq\mathbb{R}\wedge\mathcal{X}$ is bounded where $d=(\eta,C),\ \Delta=\max\eta$ subject to $C,\ \Delta^{rows}=\Delta,\ \Delta^{sum} = \Delta\cdot\max(\lvert\sup\mathcal{X}\rvert,\lvert\inf\mathcal{X}\rvert),$ $\epsilon' = \epsilon/2,\ v^{mean} \leftarrow (s.sum() + Lap(\Delta^{sum}/\epsilon'))/(s.shape[0] + Lap(\Delta^{rows}/\epsilon'))$
値の出現頻度カウント (s.value_counts(sorted=False))‡	$\langle s\rangle_d^{p,\mathcal{X}} : \langle\mathcal{S}\rangle \ \rightarrow\ \mathcal{S}[v_j\mapsto\langle n_j\rangle_{d_j}] : \mathcal{S}$ if $\mathcal{X}$ is finite where $v_j\in\mathcal{X},\ n_j \leftarrow \lvert\{v\in s \mid v = v_j\}\rvert$ $d=(\eta,C),\ d_j = (\eta_j, C\cup\{\sum_j\eta_j\le\eta\}),\ \eta_j \leftarrow$ a fresh distance variable

† シリーズに対してもデータフレームと同様
‡ 排他的な分割

が保存しない演算では出力に新しくタグを割り当てる．演算規則中では新しいタグを p^{fresh} で表し，その時点までに生成したタグと重複しない一意の ID が割り当てられる．例えば行のフィルタリングでは 2 つの入力の行は保存している必要があるが，入力に対して出力の行は保存せず，演算規則は以下のように表される．

$$\langle df \rangle_d^p : \langle \mathcal{DF} \rangle, \langle s \rangle_d^p : \langle \mathcal{S} \rangle \to \langle df[s] \rangle_d^{p^{fresh}} : \langle \mathcal{DF} \rangle$$

4.2.3 データフレームの列のドメイン管理

3.2 節で述べた通り，秘匿データフレームの各列は定義域（ドメイン）をメタデータとして持ち，`df.domains`（シリーズに対しては `s.domain`）で参照できる．表 1 に示したように，演算規則中では秘匿データフレームのドメインは $[c_i \mapsto \mathcal{X}_i]$（列名 c_i をキーとした辞書）として表され，$\langle df \rangle_d^{p,[c_i \mapsto \mathcal{X}_i]} : \langle \mathcal{DF} \rangle$ のように秘匿データフレームに紐づけられる．秘匿シリーズは単一のドメインが紐づくため，$\langle s \rangle_d^{p,\mathcal{X}} : \langle \mathcal{S} \rangle$ と表される．

表 2 にクリッピングや合計値，平均値など，各列のドメインを扱う演算規則を示す．例えば，クリッピングでは実数のドメインがある範囲にバウンドされ，そのドメインが合計値や平均値の演算に用いられる．`df.groupby()` はある列の値に応じてデータフレームを分割する演算，`s.value_counts()` はシリーズの各値の出現回数をカウントする演算であるが，これらの演算の条件として，カテゴリ値などの有限集合をドメインとして持つ必要がある．なぜなら，「ある値が存在しなかったこと」も秘匿すべき情報であり，取り得る値全てに対して秘匿値を紐づけて出力する必要があるためである．よって，元の Pandas の挙動とは異なり，対象の値が実際のデータ中に存在していなかったとしても，要素数 0 のデータフレームやカウント 0 を秘匿値として返す．

4.3 排他的な分割を考慮した距離の追跡

例えば `df.groupby()` のように，排他的にデータフレームを分割（partition）する演算の感度は 1 と定義できる．このような分割演算は，入力のデータフレーム df をある基準によって排他的に分割し，データフレームのリスト（$[df_i]$）を出力する．異なるリスト間の距離を L1 ノルムで定義すると，入力 df の距離は出力 $[df_i]$ においても変化しない．つまり，演算規則で表すと，$\langle df \rangle_d^p : \langle \mathcal{DF} \rangle \to \langle [df_i] \rangle_d : \langle [\mathcal{DF}] \rangle$ と書ける．なぜなら，df への ある 1 行の追加/削除は，リストの要素のうちただ 1 つの df_i における 1 行の追加/削除に対応するためである．

ここで，リストからある要素を取り出す演算の感度の定義を考える．保守的に感度を定義するなら，ある要素を取得する演算の感度も 1 であり，演算規則は $\langle [df_i] \rangle_d : \langle [\mathcal{DF}] \rangle$, $k : \mathbb{N}_0 \to \langle df_k \rangle_d^{p^{fresh}} : \langle \mathcal{DF} \rangle$ と書ける．しかし，例えば各データフレーム df_i を取り出し，それぞれの行数 n_i をカウントし，n_i をすべての i $(0 \le i < N)$ について足し合わせると，最終的な結果の距離は $d \cdot N$ となる．しかし，この行数の合計値 $\sum_i n_i$ は分割前のデータフレーム df の行数と等しい値であり，素直に df の行数を取得した場合の距離は d である．この距離の不必要な増幅は，データフレームのリストから各データフレームを取り出す演算の感度の定義によるものである．

この問題は，Spar [24] の距離変数と距離制約の考え方を導入することで解決できる．Spar では距離を型パラメータとして表現し，静的に追跡している．リストから各要素を取り出す演算において，あるリスト $[n_i]$ の距離が d だった場合，制約 $\sum_i d_i = d$ の下で各要素 n_i に対して距離 d_i が割り当てられる．よって，$\sum_i n_i$ の距離は $\sum_i d_i$ と表現され，制約 $\sum_i d_i = d$ の下で距離が d であると判断できる．

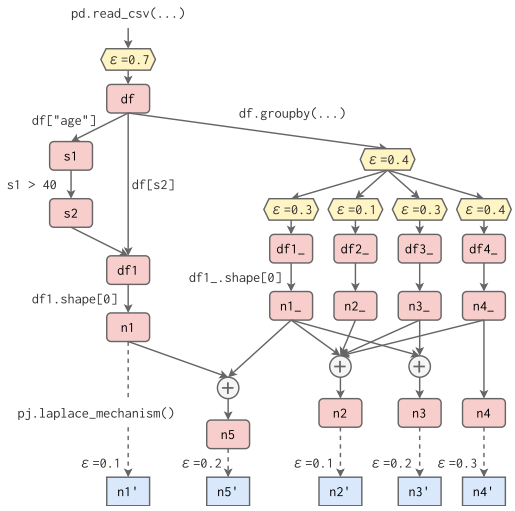

図 3 並列合成定理を考慮したプライバシー予算管理

PrivJail では，「距離」を距離表現（expression）η と距離制約（constraints）C の組として，$d = (\eta, C)$ と表記する．例えば，データフレームの排他的な分割では，分割後の各要素 df_i に対してそれぞれ距離変数 η_i を新しく割り当て，その距離変数間の制約 $\sum_i \eta_i \le \eta$ の下で df_i の距離表現を η_i として与える．表 2 において `groupby()`，および `value_counts()` ではこの排他的な分割による距離制約を導入している．

表 2 では，差分プライバシーを適用する演算規則（“$\rightsquigarrow$” が含まれるもの）において距離 d から距離表現 η と距離制約 C を取り出し，C の制約下で η の最大値を求め，その最大値を感度として扱っている．PrivJail の現在の実装では SymPy [36] で距離表現を表現し，線形計画法で距離の最大値を求めている．

演算規則中における距離に対する演算は，$d = (\eta, C)$，$d' = (\eta', C')$ として以下のように定義する．

- **距離の加算**: $d + d' = (\eta + \eta', C \cup C')$
- **距離の乗算**: $d \cdot v = (\eta \cdot v, C) \quad \text{where } v \in \mathbb{R}_{>0}$
- **距離の最大値**: $\max(d, d') = (\max(\eta, \eta'), C \cup C')$

ただし，最後の距離の最大値については，$\max(\eta, \eta')$ が定数にリダクション可能ではない場合（つまり距離変数を含んだ形になっている場合），実装では $\eta + \eta'$ として扱っている．これは，max() を含んだ距離表現は非線形であり，線形計画法を用いることができなくなるためである．

5 プライバシー予算の管理

PrivJail では消費した ϵ の値を動的に積算し，予め設定したプライバシー予算の上限に達した時点でそれ以上の値の出力を禁止

する．Ryan ら [35] はこの仕組みを *privacy odometer and filters* と呼び，様々な種類の DP に対して積算方法を提案した．本稿では pure DP のみを扱うため，単純に消費した ϵ の値の和を積算する．

しかし，既存研究では並列合成定理を考慮しておらず，背反なデータ分割が発生するプログラムの ϵ の値を必要以上に多く積算してしまう．特に，3.3 節で示した決定木学習のプログラムでは `df.groupby()` による背反な分割が頻繁に発生する．

そこで，本稿では並列合成定理を考慮した ϵ の積算手法を提案する．あるデータベース D を背反に分割した結果を $\{D_1, D_2, \ldots, D_n\}$ とする．ここで，各 D_i に適用したメカニズムで消費した ϵ の値を各 D_i ごとにそれぞれ積算し，その値を $E(D_i)$ とする．すると，全体では $\max_i E(D_i)$-差分プライバシーを満たす．これは合成定理より容易に分かる．

次に，メカニズムの適用が各 D_i それぞれに対してではなく，複数の分割後のデータベースの組み合わせ（例えば $D_1 \cup D_2$ など）に対して行われる場合を考える．例えば，$D_1 \cup D_2$，$D_2 \cup D_3$，$D_3 \cup D_1$ に対してそれぞれ ϵ を消費してメカニズムを適用した場合，全体では 3ϵ ではなく 2ϵ-差分プライバシーを満たす．この結果は一般的に知られている並列合成定理（2.3 節）からは自明ではない．本稿では並列合成定理を上のような例も包含するよう一般化し，付録 付録 B にその定理と証明を示す．結論としては，D_i を含むデータに対して適用したメカニズムで消費された ϵ を各 D_i ごとに積算し，その値を $E(D_i)$ とすると，全体では $\max_i E(D_i)$-差分プライバシーを満たす．

この性質により，実際のプライバシー予算管理の実装はシンプルに設計できる．図 3 にプライバシー予算管理の概念図を示す．グラフは秘匿値（赤，角丸）と公開値（青，四角）の親子関係を表し，ϵ の積算値（黄，六角）を ϵ **ノード**として加えている．n1 のように先祖において背反な分割が発生していない場合，根の ϵ ノードに対し直接 ϵ の値を積算する．一方，n2，n3，n4 は背反な分割後のデータフレームを先祖に含んでおり，自身の先祖の分割後データフレームに対応する全ての ϵ ノードに消費した ϵ の値をそれぞれ加算する．背反な分割においては子の ϵ ノードの値の最大値が親の ϵ ノードの値となり，その値はさらにその親（図では根）の ϵ ノードに積算される．

より複雑な例として，n5 は分割前のデータフレームと分割後のデータフレームの両方を先祖に含んでいる．このような場合には，それらの唯一の最小共通祖先，Lowest “Single” Common Ancestor (LSCA) [10] を求め，その LSCA ノード（この例では根ノード）に ϵ の値を積算する．この場合，LSCA ノードの子孫には ϵ の値は積算しない．なお，あるノード a，b の LSCA は根ノードから a および b までの全てのパスが通過するような最小のノードとして定義される．

6 プロセス分離による個人データ隔離

以上で議論した差分プライバシーの保証は，ライブラリが明示的に提供する演算のみを用いた場合に限定される．2.4 節で議論したように，汎用言語上ではライブラリが想定しない操作によって秘匿値への直接アクセスが可能な抜け道が多く存在する．例えば，Python ではクラスオブジェクトのプライベートメンバへのアクセスは禁止できず，容易に秘匿値の中身を出力できてしまう．

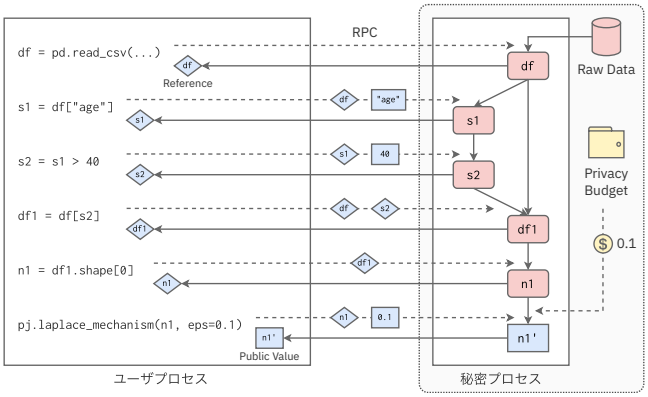

図 4 ユーザプロセスと秘密プロセスの分離

そこで，ユーザの与えるスクリプトと秘匿値をプロセスレベルで分離し，適切な RPC のみを介してのみ秘匿値を操作可能な設計を提案する．図 4 にその概念図を示す．ここでは，ユーザの与えるスクリプトを実行するプロセスを**ユーザプロセス**と呼び，秘匿値を管理するプロセスを**秘密プロセス**と呼ぶ．プロセスの配置方法は様々考えられるが，ここではユーザプロセスはデータ解析者のクライアント端末で実行されており，秘密プロセスはデータ管理者のサーバで動作するものとする．

ユーザプロセス上で発生した PrivJail のライブラリ呼び出しは，ライブラリ内部で RPC リクエストに変換され，秘密プロセスに送信される．秘密プロセス側ではリクエストを検証した後，秘匿値に対し対応する演算を行い，結果の秘匿値への参照をユーザプロセスに返す．参照には秘匿値の具体的な中身は含まれない．秘匿値に対する演算を行う場合は，秘匿値への参照を RPC の引数として送信する．差分プライバシーを適用した結果の公開値はユーザプロセスに（参照ではなく）直接返される．このようにプロセス分離を行うことで，たとえユーザプロセス上の言語処理系に脆弱性があったとしても，それが直ちに秘密プロセス上の秘匿値に影響することはない．PrivJail の RPC 実装には gRPC [16] を用いている．

7 脅威モデルと対策

本稿における脅威モデルと対策について簡潔に述べる．主な脅威として悪意のあるデータ解析者，すなわち PrivJail を悪用しプライバシー侵害を目論む攻撃者を考える．攻撃者は主たる手段として Python プログラムを通じた生データの窃取を試みるが，これは 6 節のプロセス分離により大部分防がれる．他者のクライアント端末に対する攻撃や，データ管理者のサーバへの PrivJail と無関係な攻撃は考慮しない．PrivJail の秘密プロセスに対する攻撃は，ユーザ認証を伴うセキュアな RPC プロトコルで防ぐものとする．ノイズのサンプリングにおける浮動小数点の脆弱性 [28]，およびタイミング攻撃等のサイドチャネル攻撃 [17] については本研究のスコープ外とする．

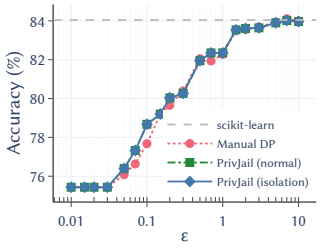

図 5　決定木学習の正解率

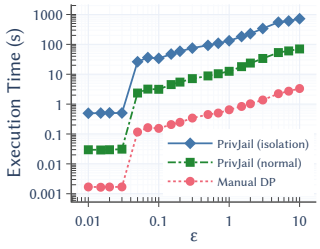

図 6　決定木学習の実行時間

8　評価

PrivJail の評価には 3.3 節の図 2 に示した決定木学習アルゴリズム DiffPID3 [13] のプログラムを用いる．PrivJail の実行方式としては，6 節のプロセス分離を行う実行（“PrivJail (isolation)”）と行わない実行（“PrivJail (normal)”）を比較する．プロセス分離を行う場合，ユーザプロセスと秘密プロセスは同一のサーバ上で起動して実験を行った．加えて，PrivJail を用いずに Numpy [18] と Pandas [31] を直接用いた DiffPID3 の実装（“Manual DP”）と性能比較を行う．PrivJail を用いない実装ではユーザが手動で正しい感度を与える必要があるが，PrivJail を用いれば同様の Pandas 記法を用いつつ自動的に正しい感度を導出することが可能である．

評価では，DiffPID3 の評価 [13] と同様に，Adult データセット [3] を用いて年収が 5 万ドル以下かどうかの 2 クラス分類を行う．連続値は全て 20 個の区間に分割し，離散的なカテゴリ値として扱う．最大の木の深さは 5 とし，pruning は行わない．性能評価は，2 ソケットの Intel Xeon Gold 6126 CPU 上で Ubuntu 22.04 (Linux kernel: `5.15.0-107-generic`) を起動し，CPython 3.11.11 に Numpy 2.2.1，Pandas 2.2.3 のパッケージをインストールして行った．図中のプロットはいずれも 10 回実行した平均を示す．

図 5 に消費したプライバシー予算（ϵ）に応じたタスクの正解率を示す．正解率の参考値として，カテゴリ値の one-hot encoding を行った上で Scikit-learn [32] 1.6.0 の決定木学習（`DecisionTreeClassifier`）を行い，その正解率を破線で示した．ϵ が十分大きい場合，DiffPID3 の実装はいずれも Scikit-learn に近い正答率を示している．$\epsilon \leq 0.03$ では木の深さは 0（ノード数が 1）であった．

次に，図 6 に決定木学習の実行時間の結果を示す．PrivJail (normal) は Manual DP に比べて 20–34 倍の性能悪化が見られる．これは主に動的な感度追跡のオーバーヘッドに起因する．特に，決定木学習では `groupby()` や `value_counts()` において距離変数が頻繁に出現するため（4.3 節），それらを処理するオーバーヘッドが大きいと考えられる．加えて，プロセス分離によって 10 倍程度のオーバーヘッドが生じている．今回は同一のサーバ上での実験を行ったが，ユーザプロセスと秘密プロセスの実行サーバを分離する場合，通信オーバーヘッドが追加で発生する．PrivJail の性能最適化は今後の課題とする．

9　結論

本稿では，Python 上で差分プライバシーの適用を強制するライブラリ，PrivJail を提案した．PrivJail の特徴の 1 つは，データ解析者に親しみのある Numpy や Pandas の記法を用いることが可能な点である．これを達成するため，Pandas に代表されるデータフレーム演算に適用可能な形で秘匿値に対する演算の感度追跡の規則を整理した．結果として，一般的な Pandas 記法を用いつつ，決定木学習のように動的で複雑なアルゴリズムを簡潔に記述可能であることを示した．

PrivJail のもう 1 つの特徴は，個人データのセキュリティ向上のため，データ解析者に対して秘匿値に直接アクセスすることを禁止可能な点である．これは言語処理系と秘匿値のプロセス分離によって実現され，安全な個人データ利活用の促進が期待される．

PrivJail は `https://github.com/privjail/privjail` に OSS として公開している．

謝辞

本研究の一部は，JST，CREST，JPMJCR21M2 の支援を受けたものである．

参考文献

[1] Chiké Abuah, David Darais, and Joseph P. Near. Solo: a lightweight static analysis for differential privacy. *Proceedings of the ACM on Programming Languages*, 6(OOPSLA2):699–728, October 2022.
[2] Chike Abuah, Alex Silence, David Darais, and Joseph P. Near. DDUO: General-purpose dynamic analysis for differential privacy. In *Proceedings of the 2021 IEEE 34th Computer Security Foundations Symposium*, CSF ’21, pages 1–15, 2021.
[3] Barry Becker and Ronny Kohavi. Adult – UC Irvine machine learning repository. `https://doi.org/10.24432/C5XW20`, 1996.
[4] Skye Berghel, Philip Bohannon, Damien Desfontaines, Charles Estes, Sam Haney, Luke Hartman, Michael Hay, Ashwin Machanavajjhala, Tom Magerlein, Gerome Miklau, Amritha Pai, William Sexton, and Ruchit Shrestha. Tumult Analytics: a robust, easy-to-use, scalable, and expressive framework for differential privacy. `https://arxiv.org/abs/2212.04133`, 2022.
[5] Microsoft Corporation and Harvard University. SmartNoise. `https://smartnoise.org`, 2020. Last accessed on 2024-12-19.
[6] Cynthia Dwork. Differential privacy. In *Proceedings of the 33rd International Colloquium on Automata, Languages, and Programming*, ICALP ’06, pages 1–12, 2006.
[7] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Advances in Cryptology: Proceedings of the 25th International Conference on the Theory and Applications of Cryptographic Techniques*, EUROCRYPT ’06, pages 486–503, 2006.
[8] Cynthia Dwork, Moni Naor, Toniann Pitassi, Guy N. Rothblum, and Sergey Yekhanin. Pan-private streaming algorithms. In *Proceedings of the 1st Symposium on Innovations in Computer Science*, ICS ’10, pages 66–80, 2010.
[9] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3–4):211–407, 2014.
[10] Johannes Fischer and Daniel H. Huson. New common ancestor problems in trees and directed acyclic graphs. *Information Processing Letters*, 110(8–9):331–335, April 2010.
[11] OpenMined Foundation. PyDP. `https://github.com/OpenMined/PyDP`, 2020. Last accessed on 2024-12-19.
[12] Zope Foundation. RestrictedPython. `https://github.com/zopefoundation/RestrictedPython`, 2024. Last accessed on 2024-12-27.
[13] Arik Friedman and Assaf Schuster. Data mining with differential

privacy. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 493–502, 2010.
[14] Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C. Pierce. Linear dependent types for differential privacy. In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '13, pages 357–370, 2013.
[15] Marco Gaboardi, Michael Hay, and Salil Vadhan. A programming framework for OpenDP. https://projects.iq.harvard.edu/files/opendp/files/opendp_programming_framework_11may2020_1_01.pdf, 2020. Last accessed on 2024-12-18.
[16] gRPC Authors. gRPC. https://grpc.io, 2025. Last accessed on 2025-02-10.
[17] Andreas Haeberlen, Benjamin C. Pierce, and Arjun Narayan. Differential privacy under fire. In *Proceedings of the 20th USENIX Security Symposium*, USENIX Security '11, pages 1–15, 2011.
[18] Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández Del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.
[19] Naoise Holohan, Stefano Braghin, Pól Mac Aonghusa, and Killian Levacher. Diffprivlib: The IBM differential privacy library. https://arxiv.org/abs/1907.02444, 2019.
[20] Noah Johnson, Joseph P. Near, and Dawn Song. Towards practical differential privacy for SQL queries. *Proceedings of the VLDB Endowment*, 11(5):526–539, January 2018.
[21] Daniel Kifer and Ashwin Machanavajjhala. No free lunch in data privacy. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 193–204, 2011.
[22] Ios Kotsogiannis, Yuchao Tao, Xi He, Maryam Fanaeepour, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. PrivateSQL: a differentially private SQL query engine. *Proceedings of the VLDB Endowment*, 12(11):1371–1384, July 2019.
[23] Ninghui Li, Min Lyu, Dong Su, and Weining Yang. *Differential Privacy: From Theory to Practice*. Springer Cham, October 2016.
[24] Elisabet Lobo-Vesga, Alejandro Russo, Marco Gaboardi, and Carlos Tomé Cortiñas. Sensitivity by parametricity. *Proceedings of the ACM on Programming Languages*, 8(OOPSLA2):415–441, October 2024.
[25] Oblivious Software Ltd. Antigranular. https://www.antigranular.com, 2023. Last accessed on 2024-12-09.
[26] Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '07, pages 94–103, 2007.
[27] Frank D. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 19–30, 2009.
[28] Ilya Mironov. On significance of the least significant bits for differential privacy. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 650–661, 2012.
[29] Joseph P. Near, David Darais, Chike Abuah, Tim Stevens, Pranav Gaddamadugu, Lun Wang, Neel Somani, Mu Zhang, Nikhil Sharma, Alex Shan, and Dawn Song. Duet: an expressive higher-order language and linear type system for statically enforcing differential privacy. *Proceedings of the ACM on Programming Languages*, 3(OOPSLA):172:1–172:30, October 2019.
[30] Ivoline C. Ngong, Brad Stenger, Joseph P. Near, and Yuanyuan Feng. Evaluating the usability of differential privacy tools with data practitioners. In *Proceedings of the Twentieth Symposium on Usable Privacy and Security*, SOUPS '24, pages 21–40, 2024.
[31] The pandas development team. pandas-dev/pandas: Pandas (v2.2.3). https://doi.org/10.5281/zenodo.13819579, 2024.
[32] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, November 2011.
[33] Devin Petersohn, Stephen Macke, Doris Xin, William Ma, Doris Lee, Xiangxi Mo, Joseph E. Gonzalez, Joseph M. Hellerstein, Anthony D. Joseph, and Aditya Parameswaran. Towards scalable dataframe systems. *Proceedings of the VLDB Endowment*, 13(12):2033–2046, July 2020.
[34] Jason Reed and Benjamin C. Pierce. Distance makes the types grow stronger: a calculus for differential privacy. In *Proceedings of the 15th ACM SIGPLAN International Conference on Functional Programming*, ICFP '10, pages 157–168, 2010.
[35] Ryan Rogers, Aaron Roth, Jonathan Ullman, and Salil Vadhan. Privacy odometers and filters: pay-as-you-go composition. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 1929–1937, 2016.
[36] SymPy Development Team. SymPy. https://www.sympy.org, 2024. Last accessed on 2025-02-10.
[37] Royce J. Wilson, Celia Yuxin Zhang, William Lam, Damien Desfontaines, Daniel Simmons-Marengo, and Bryant Gipson. Differentially private SQL with bounded user contribution. In *Proceedings on Privacy Enhancing Technologies*, PETS '20, pages 230–250, 2020.

付録 A　データフレーム間の行単位演算規則の正当性

4.2.2 節にデータフレーム間の「行が保存」した関係を定義し，行が保存したデータフレーム間であれば行単位の演算の結果の距離をバウンド可能であることを述べた．本付録では，その理由について述べる．

複数の秘匿データフレーム（シリーズでも同様）を入力とする演算 f を

$$\langle df_1\rangle_{d_1} : \langle \mathcal{DF}\rangle, \ldots, \langle df_n\rangle_{d_n} : \langle \mathcal{DF}\rangle \xrightarrow{f} \langle v_o\rangle_{d_o} : \langle \tau_o\rangle$$

と表す．この演算規則において式 (6) の条件を考えると，任意の df_i，df_i' の全ての組み合わせに対して距離がバウンドできる必要がある．しかし，行単位の演算においては，4.2.2 節で述べたような行の対応関係のずれにより，出力の距離 d_o をバウンドすることは困難である．

そこでまず，$df_1, \ldots, df_n$ にある共通祖先 df が存在することを仮定する．すなわち，df_i は全て df から派生した値である．その共通祖先 df について距離 d の任意のペア df，df' を考えれば，各 df_i，df_i' は自動的に決まる．共通祖先 df から各 df_i を出力する演算を f_i とし，その演算規則を $\langle df\rangle_d : \langle \mathcal{DF}\rangle \xrightarrow{f_i} \langle df_i\rangle_{d_i} : \langle \mathcal{DF}\rangle$ とすると，

$$\begin{aligned}\forall df, df'.\big(&df_1 = f_1(df) \wedge \cdots \wedge df_n = f_n(df) \wedge \\ &df_1' = f_1(df') \wedge \cdots \wedge df_n' = f_n(df') \wedge \\ &d_{\mathcal{DF}}(df, df') \le d \\ \Longrightarrow\ &d_{\tau_o}(f(df_1, \ldots, df_n), f(df_1', \ldots, df_n')) \le d_o\big)\end{aligned} \tag{7}$$

が出力の距離 d_o に関する条件である．これは，$F(df) = f(f_1(df), \ldots, f_n(df))$ と定義した場合の演算規則 $\langle df\rangle_d : \langle \mathcal{DF}\rangle \xrightarrow{F} \langle v_o\rangle_{d_o} : \langle \tau_o\rangle$ の条件と一致する．この条件では，df_i および df_i' は連動して変化するため，f_i が何らかの性質を保存する演算であった場合，df_i および df_i' は df，df' の性質をそれぞれ受け継いでいると考えることができる．

全ての f_i が「行が保存」した関係にある場合，各 df_i の行数は同じであり，行の対応関係も df から変化していない．df_i' についても df' に対して同様である．すなわち，例えばある df に対し

て 1 行追加/削除した df' を考えたとき，各 df'_i についても同様に df_i に対して同じ位置に 1 行追加/削除したデータフレームになっている．4.2.2 節のフィルタリングの例では，bool 値のシリーズに 1 行追加されて要素が 1 行ずつずれたとしても，対応するデータフレームも同じ位置に 1 行追加されているはずであり，行の対応関係が変化せず，出力の距離をバウンド可能になる（感度は 1）．

付録 B　並列合成定理の一般化

5 節で述べた並列合成定理を考慮したプライバシー予算管理において以下の一般化された並列合成定理を用いている．ここではその定理とその証明を議論する．

定理 1. あるデータベース D の背反な分割 $\Pi_D = \{D_1, D_2, \ldots, D_n\}$ が存在し，逐次的な全 m ステップの各ステップ $i \in [m] = \{1, 2, \ldots, m\}$ では Π_D の部分集合 $\mathcal{D}_i \subseteq \Pi_D$ に対し ϵ_i-差分プライバシーを満たすメカニズムを適用したとする．このとき，ある $D_k \in \Pi_D$ を含むデータベースに対してメカニズムを適用したステップの集合を $I^+_{D_k} = \{i \in [m] \mid D_k \in \mathcal{D}_i\}$ として，全体では $(\max_{D_k \in \Pi_D} \sum_{i \in I^+_{D_k}} \epsilon_i)$-差分プライバシーを満たす．

証明. 各ステップ $i \in [m]$ で $\mathcal{D}_i \subseteq \Pi_D$ に対し ϵ_i-差分プライバシーを満たすメカニズム M_i を適用したとする．$\mathcal{D}_i$ の要素の和集合を $\cup\mathcal{D}_i = \bigcup_{D_j \in \mathcal{D}_i} D_j$ と表記する．このとき，M_i の任意の出力集合 S_i に対し $P\left[M_i(\cup\mathcal{D}_i) \in S_i\right] \leq \exp(\epsilon_i) P\left[M_i(\cup\mathcal{D}'_i) \in S_i\right]$ が成り立つ．ここで，D_j に対応する隣接データベースを D'_j として，$\mathcal{D}'_i = \{D'_j \mid D_j \in \mathcal{D}_i\}$ である．

D の隣接データベース D' について同様の分割 $\Pi_{D'} = \{D'_1, D'_2, \ldots, D'_n\}$ を考えると，Unbounded DP の定義より，ある $D_k \in \Pi_{D'}$ について $D_k \neq D'_k$ であった場合，他の $D_j \in \Pi_{D'}$ $(j \neq k)$ については $D_j = D'_j$ である．そのような D_k に対して，D_k にメカニズムを適用するステップの集合 $I^+_{D_k} = \{i \in [m] \mid D_k \in \mathcal{D}_i\}$，それ以外のステップの集合 $I^-_{D_k} = [m] \setminus I^+_{D_k}$ を定義して，

$$
\begin{aligned}
&P\left[\bigwedge_{i \in [m]} M_i(\cup\mathcal{D}_i) \in S_i\right] \\
&= \prod_{i \in [m]} P\left[M_i(\cup\mathcal{D}_i) \in S_i\right] \\
&= \prod_{i \in I^+_{D_k}} P\left[M_i(\cup\mathcal{D}_i) \in S_i\right] \prod_{i \in I^-_{D_k}} P\left[M_i(\cup\mathcal{D}_i) \in S_i\right] \\
&\leq \prod_{i \in I^+_{D_k}} \exp(\epsilon_i) P\left[M_i(\cup\mathcal{D}'_i) \in S_i\right] \prod_{i \in I^-_{D_k}} P\left[M_i(\cup\mathcal{D}'_i) \in S_i\right] \\
&= \prod_{i \in I^+_{D_k}} \exp(\epsilon_i) \prod_{i \in [m]} P\left[M_i(\cup\mathcal{D}'_i) \in S_i\right] \\
&= \exp\left(\sum_{i \in I^+_{D_k}} \epsilon_i\right) P\left[\bigwedge_{i \in [m]} M_i(\cup\mathcal{D}'_i) \in S_i\right] \\
&\leq \exp\left(\max_{D_k \in \Pi_D} \sum_{i \in I^+_{D_k}} \epsilon_i\right) P\left[\bigwedge_{i \in [m]} M_i(\cup\mathcal{D}'_i) \in S_i\right]
\end{aligned}
$$

□