Topik : 4.4. Finalisasi Prototipe

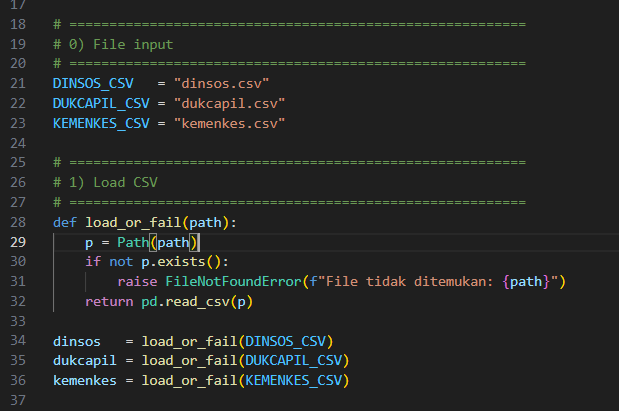
Objective : Siapkan pipeline end-to-end: dataset → FL → DP → hasil

Task : Uji ulang semua modul, pastikan reproducible

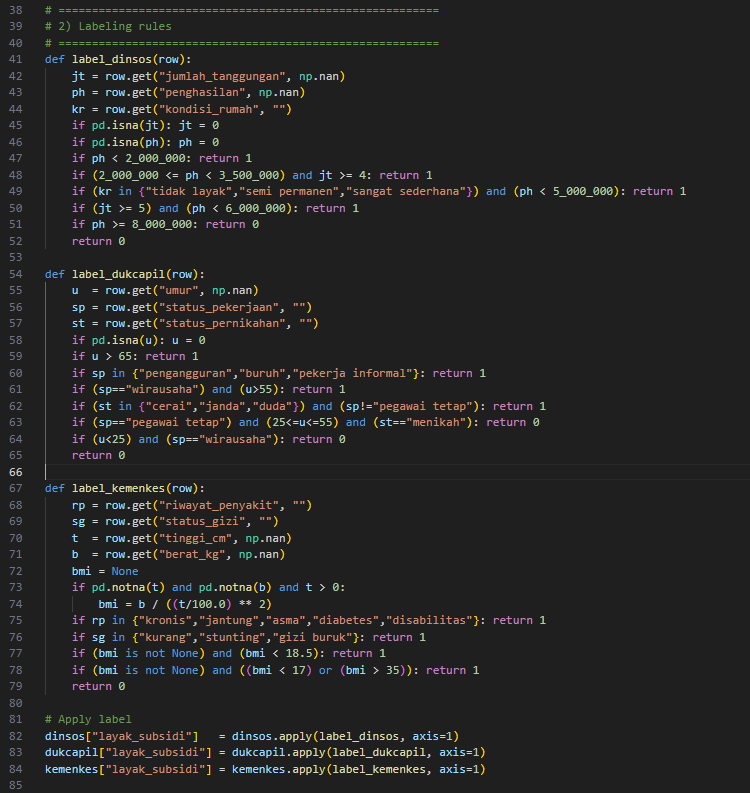


**LOCAL (manual)**

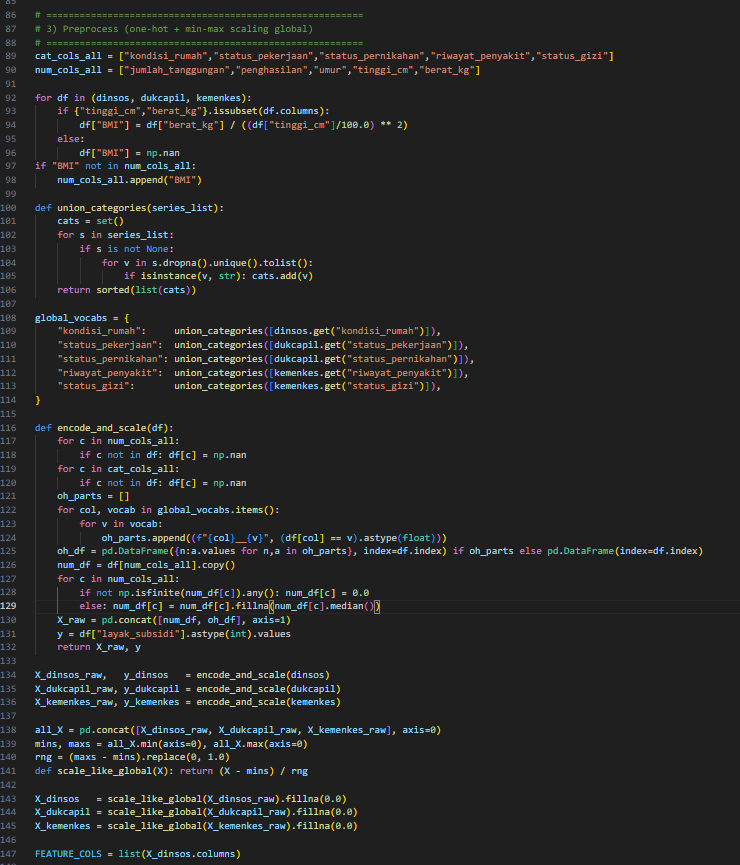
1. Dataset (Load Data)



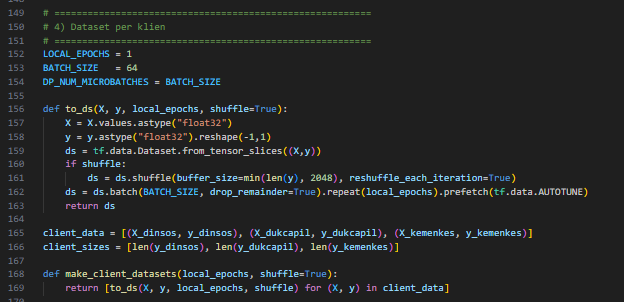
1. Labeling



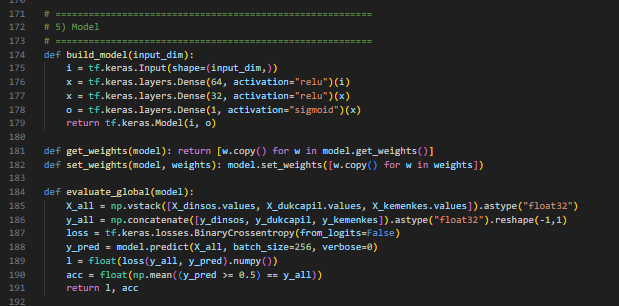
1. Preprocessing



1. Dataset per klien



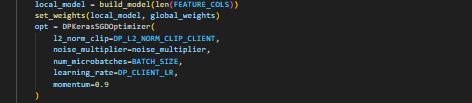
1. Model

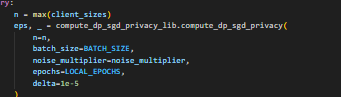


1. Federated learning

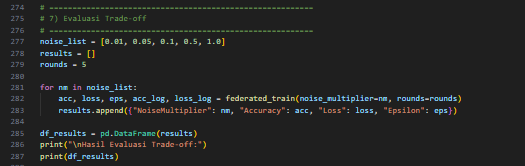


1. Differential Privacy

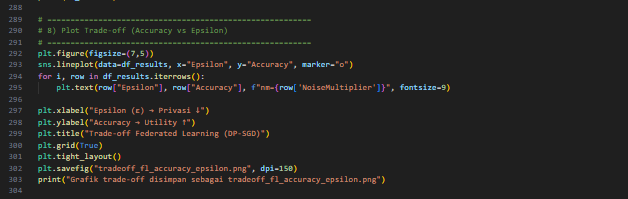




1. Evaluasi



1. Plot Trade-off



**Pipeline end-to-end :**

1. Dataset (load data)

Tahap pertama Adalah mengambil data mentah dari tiga client :

* Dinsos 🡪 berisi informasi jumlah\_tanggungan, penghasilan, kondisi rumah
* Dukcapil 🡪 berisi informasi umur, status\_pekerjaan, status\_pernikahan
* Kemenkes 🡪 berisi informasi Riwayat\_penyakit, status\_gizi, tinggi, berat

Data ini di-*load* dari file CSV menggunakan pandas.read\_csv(). Jika file tidak ada, program berhenti agar reproducibility terjaga.

1. Labeling

Data mentah tidak memiliki target langsung. Maka dibuat aturan untuk menentukan label layak\_subsidi (1 = layak, 0 = tidak layak).

* Output tahap ini : kolom baru layak\_subsidi pada tiap dataset

1. Pre-processing

Agar model dapat dilatih :

* Numerik : nilai yang hilang diisi mendian, lalu di-*scale* pakai min-max normalisasi global
* Kategori : diubah menjadi one-hot encoding berdasarkan vocab gabungan (supaya konsisten antar klien)
* Fitur Tambahan : BMI dihitung dari tinggi & berat badan

Output : matriks fitur numerik dan vector label yang sudah bersih

1. Dataset per Klien

Setiap sumber data dianggap sebagai client dalam Federated Learning.

* Data tiap klien dikonversi ke tf.data.Dataset kemudian dilakukan batching, shuffling, repeat sesuai dengan local epochs
* Tiga dataset klien siap dipakai dalam loop training

Hal ini mensimulasikan kondisi nyata Dimana data tidak dikumpullkan disatu tempat, melainkan dilatih secara terdistribusi

1. Model

Menggunakan arsitektur :

* Input layer sesuai jumlah fitur
* Hidden layer : 64 neuron (ReLU) 🡪 32 neuron (ReLU)
* Output layer : 1 neuron sigmoid (binary classification)

Penerapan model yang cukup ringan sehingga cocok untuk melakukan federated learning.

1. Federated Learning

Implementasi manual Federated Averaging (FedAvg) :

* Tiap klien dimulai dari bobot global
* Klien melatih model local pada datasetnya dengan optimizer DP-SGD
* Delta bobot hasil training di-clip dengan L2 norm agar tidak terlalu besar
* Server melakukan aggregasi delta bobot dari semua klien (weighted average berdasarkan ukuran dataset
* Update bobot global 🡪 looping untuk beberapa round

1. Differential privacy

Untuk menjamin privasi data tiap klien, maka digunakan :

* DP-SGD Optimizer 🡪 menambahkan *noise Gaussian* ke gradien, plus clipping gradien
* Menggunakan parameter penting
  + L2\_norm\_clip 🡪 batas clipping gradien
  + Noise\_multiplier 🡪 besar noise yang ditambahkan
* Estimasi privasi dihitung dengan menggunakan compute\_dp\_sgd\_privacy 🡪 menghasilkan nilai ε (epsilon), semakin kecil artinya privasi lebih kuat

1. Evaluasi

Setelah setiap federated round :

* Model global dievaluasi pada gabungan semua data klien
* Metrics : accuracy dan loss
* Disimpan log per round agar bisa dianalisis konsistensinya
* Dicatat pula nilai rata – rata akurasi/loss untuk tiap noise multiplier

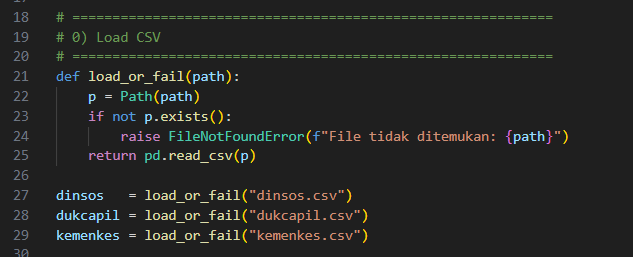
1. Plot Trade-off

Hasil evaluasi divisualisasikan :

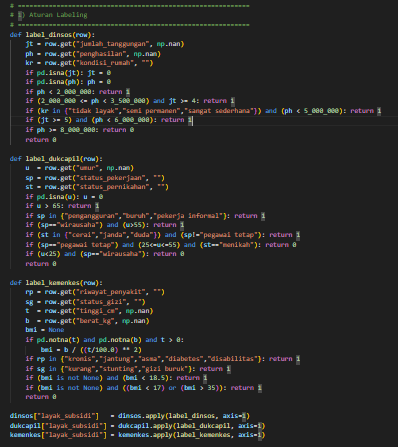
* Grafik Accuracy vs Epsilon (ε) 🡪 menunjukkan trade-off antara utility(akurasi) dan privacy **(ε)**.
* Tiap titik diberi label nm = noise\_multiplier
* Grafik disimpan dalam file PNG agar reproducible

**MENGGUNAKAN TFF**

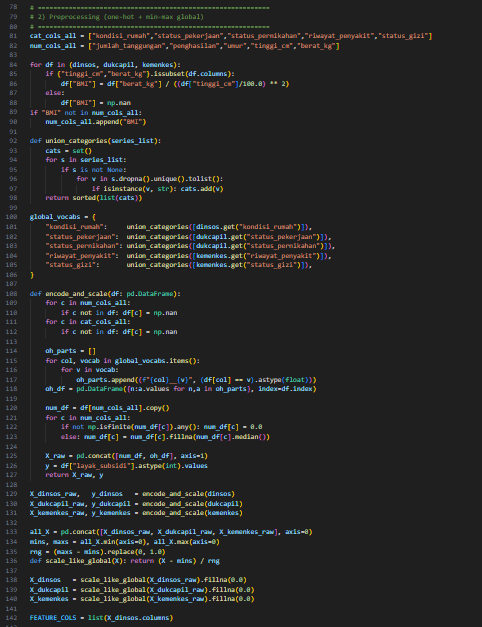
1. Load data



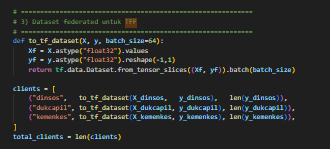
1. Aturan Labbeling



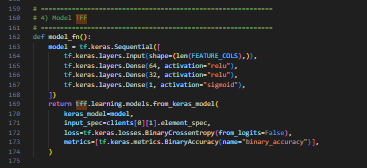
1. Pre-processing



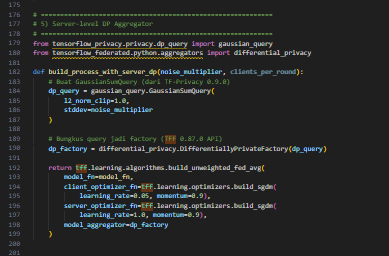
1. Datasets



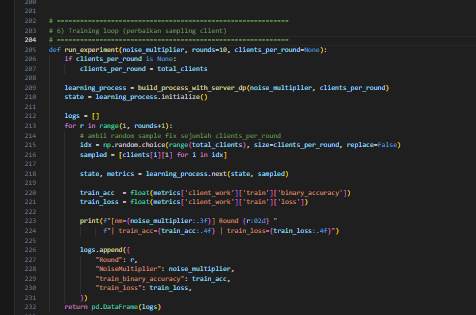
1. Model TFF



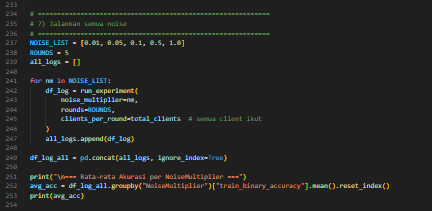
1. DP Aggregator



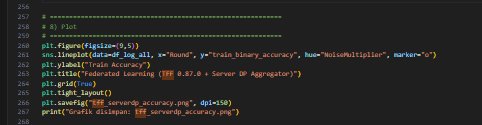
1. Training



1. Hasil



1. Plot



1. Load Dataset

* Membaca tiga dataset dari CSV (dinsos.csv, dukcapil.csv, kemenkes.csv).
* Fungsi load\_or\_fail dipakai agar jika file tidak ada, langsung error (FileNotFoundError).
* Dataset ini jadi sumber data mentah untuk setiap *client* federated learning.

1. Aturan Labeling

* Menentukan label target (layak\_subsidi) berdasarkan logika aturan sosial/ekonomi/kesehatan.
* Ada tiga fungsi khusus:
  + label\_dinsos: berdasar penghasilan, jumlah tanggungan, kondisi rumah.
  + label\_dukcapil: berdasar umur, status pekerjaan, status pernikahan.
  + label\_kemenkes: berdasar riwayat penyakit, status gizi, serta perhitungan BMI.
* tiap dataset punya kolom biner layak\_subsidi (0 = tidak, 1 = ya).

1. Preprocessing (Encoding + Scaling)

* Menyatukan fitur numerik & kategorikal agar bisa dipakai model.
* Langkah:
  1. Numerik: (jumlah\_tanggungan, penghasilan, umur, tinggi\_cm, berat\_kg, BMI) → normalisasi dengan min-max global (pakai gabungan semua dataset supaya skala seragam).
  2. Kategorikal: (kondisi\_rumah, status\_pekerjaan, dll.) → di-*one-hot encoding* berdasarkan vocab global (union dari semua kategori di semua dataset).
* Output: X (fitur terstandarisasi) dan y (label biner).

1. Dataset Federated

* Ubah data tiap institusi jadi format tf.data.Dataset agar kompatibel dengan TFF.
* Fungsi to\_tf\_dataset → menghasilkan dataset batched (X, y).
* Dikelompokkan sebagai *client*:
  + Client 1: dinsos
  + Client 2: dukcapil
  + Client 3: kemenkes

1. Model TFF

* Definisi model ML yang akan dilatih secara federated.
* Model = Keras Sequential:
  + Dense(64, relu)
  + Dense(32, relu)
  + Dense(1, sigmoid)
* Dibungkus dengan tff.learning.models.from\_keras\_model agar bisa dipakai di federated averaging.

1. Server-level DP Aggregator

* Menjamin privasi pada tahap agregasi model di server.
* Pakai GaussianSumQuery dari tensorflow-privacy → tambahkan *noise* Gaussian pada update agregasi.
* Dimasukkan ke TFF dengan DifferentiallyPrivateFactory.
* Hasilnya dipakai dalam tff.learning.algorithms.build\_unweighted\_fed\_avg.

1. Training Loop

* Melatih model federated dengan sampling client dan logging metrik.
* Tahapan tiap ronde:
  1. Pilih subset client (random sampling).
  2. Panggil learning\_process.next(state, sampled) untuk update global.
  3. Catat train accuracy dan loss tiap ronde.
* Bisa diulang dengan berbagai nilai noise multiplier (nm) untuk membandingkan trade-off.

1. Evaluasi & Plot

* Analisis hasil percobaan.
* Setelah training untuk semua nm selesai:
  + Hitung rata-rata akurasi tiap noise multiplier.
  + Plot kurva akurasi per ronde (dengan seaborn lineplot).

**PERBANDINGAN MANUAL FEDERATED DAN TFF**

1. Framework yang dipakai

* CODE 1  
  → Manual Federated Learning (pure NumPy + Keras), loop federasi dibuat manual, tidak pakai TFF.  
  → DP dilakukan di client-side dengan DPKerasSGDOptimizer (DP-SGD di level optimizer tiap klien).
* CODE 2  
  → TensorFlow Federated (TFF 0.87.0) dipakai untuk membangun algoritme federated (build\_unweighted\_fed\_avg).  
  → DP dilakukan di server-side dengan DifferentiallyPrivateFactory(GaussianSumQuery) (DP Aggregator).

2. Posisi Differential Privacy

* CODE 1 → Client-level DP-SGD
  + Noise ditambahkan saat update gradien di setiap klien.
  + Gunakan DPKerasSGDOptimizer, l2\_norm\_clip, noise\_multiplier.
  + Ada perhitungan epsilon dengan compute\_dp\_sgd\_privacy\_lib.
* CODE 2 → Server-level DP Aggregator
  + Noise ditambahkan saat agregasi model update di server.
  + Gunakan GaussianSumQuery + DifferentiallyPrivateFactory.
  + Tidak ada perhitungan epsilon eksplisit, hanya log akurasi/loss.

3. Training Loop

* CODE 1 (manual)
  + Loop round federasi ditulis sendiri (for round\_idx in range(rounds)).
  + Model global diupdate manual (delta, clipping, averaging).
  + Harus handle federated averaging secara eksplisit.
* CODE 2 (TFF)
  + Training loop lebih ringkas, karena learning\_process = build\_unweighted\_fed\_avg(...) sudah mengatur federated averaging.
  + Hanya perlu memanggil state, metrics = learning\_process.next(state, sampled).

4. Evaluasi Trade-off

* CODE 1
  + Setelah training, dihitung ε (epsilon) via compute\_dp\_sgd\_privacy\_lib.
  + Plot: Accuracy vs Epsilon (langsung lihat trade-off privasi vs utilitas).
* CODE 2
  + Hanya menghitung rata-rata akurasi per noise multiplier.
  + Plot: Accuracy vs Round untuk tiap NoiseMultiplier.
  + Tidak ada epsilon.

5. Output

* CODE 1 menghasilkan:
  + DataFrame dengan kolom: NoiseMultiplier, Accuracy, Loss, Epsilon.
  + Grafik: *tradeoff\_fl\_accuracy\_epsilon.png* (Accuracy vs Epsilon).
* CODE 2 menghasilkan:
  + Rata-rata akurasi per NoiseMultiplier.
  + Grafik: *tff\_serverdp\_accuracy.png* (Accuracy vs Round, warna berdasarkan NoiseMultiplier).

Ringkasnya

* CODE 1 = manual FedAvg + DP-SGD di klien, ada perhitungan ε.
* CODE 2 = TFF FedAvg + DP Aggregator di server, hanya akurasi/loss.

**TABEL PERBANDINGAN :**

|  |  |  |
| --- | --- | --- |
| **Aspek** | **Code 1** | **Code 2** |
| Framework | Manual federated learning (NumPy + Keras) | TensorFlow Federated |
| Posisi Differential Privacy | Client-side DP-SGD → noise ditambahkan saat training klien | Server-side DP Aggregator → noise ditambahkan saat agregasi |
| DP Tools | DPKerasSGDOptimizer (optimizer dengan clipping + noise) + compute\_dp\_sgd\_privacy\_lib | GaussianSumQuery + DifferentiallyPrivateFactory |
| Clipping & Noise | L2 clipping di level gradien tiap klien | L2 clipping di level update global sebelum agregasi |
| Federated Averaging | Ditulis manual: delta weights dihitung & dirata-rata sendiri | Ditangani otomatis oleh tff.learning.algorithms.build\_unweighted\_fed\_avg |
| Training Loop | Manual loop, update global model dilakukan sendiri | Gunakan learning\_process.next(state, sampled) |
| Evaluasi | Hitung: Accuracy, Loss, dan ε (epsilon) | Hitung: Accuracy, Loss (tanpa epsilon) |
| Plot Trade-off | Accuracy vs Epsilon → menilai utility vs privacy | Accuracy vs Round (per NoiseMultiplier) |