TASK GABUNGAN 3.1 – 3.3

1. Load data per client

```python
# ============================================================
# 0) Konfigurasi input file
# ============================================================
DINSOS_CSV   = "dinsos.csv"
DUKCAPIL_CSV = "dukcapil.csv"
KEMENKES_CSV = "kemenkes.csv"

# ============================================================
# 1) Load CSV per client (tanpa label)
# ============================================================
def load_or_fail(path):
    p = Path(path)
    if not p.exists():
        raise FileNotFoundError(f"File tidak ditemukan: {path}")
    return pd.read_csv(p)

dinsos   = load_or_fail(DINSOS_CSV)
dukcapil = load_or_fail(DUKCAPIL_CSV)
kemenkes = load_or_fail(KEMENKES_CSV)
```

Baca tiga CSV (Dinsos, Dukcapil, Kemenkes) jadi DataFrame.

2. Buat Label biner local

Dinsos :

```python
# ============================================================
# 2) Aturan kompleks untuk label 'layak_subsidi'
# ============================================================
def label_dinsos(row):
    jt = row.get("jumlah_tanggungan", np.nan)
    ph = row.get("penghasilan", np.nan)
    kr = row.get("kondisi_rumah", "")

    if pd.isna(jt): jt = 0
    if pd.isna(ph): ph = 0

    # Layak (1)
    if ph < 2_000_000:
        return 1
    if (2_000_000 <= ph < 3_500_000) and jt >= 4:
        return 1
    if (kr in {"tidak layak", "semi permanen", "sangat sederhana"}) and (ph < 5_000_000):
        return 1
    if (jt >= 5) and (ph < 6_000_000):
        return 1

    # Tidak layak (0)
    if ph >= 8_000_000:
        return 0
    return 0
```

Dukcapil :

```python
def label_dukcapil(row):
    u  = row.get("umur", np.nan)
    sp = row.get("status_pekerjaan", "")
    st = row.get("status_pernikahan", "")

    if pd.isna(u): u = 0

    # Layak (1)
    if u > 65:
        return 1
    if sp in {"pengangguran", "buruh", "pekerja informal"}:
        return 1
    if (sp == "wirausaha") and (u > 55):
        return 1
    if (st in {"cerai", "janda", "duda"}) and (sp != "pegawai tetap"):
        return 1

    # Tidak layak (0)
    if (sp in {"pegawai tetap"}) and (25 <= u <= 55) and (st == "menikah"):
        return 0
    if (u < 25) and (sp == "wirausaha"):
        return 0
    return 0
```

Kemenkes :

```python
def label_kemenkes(row):
    rp = row.get("riwayat_penyakit", "")
    sg = row.get("status_gizi", "")
    t  = row.get("tinggi_cm", np.nan)
    b  = row.get("berat_kg", np.nan)

    bmi = None
    if pd.notna(t) and pd.notna(b) and t > 0:
        bmi = b / ((t/100.0) ** 2)

    # Layak (1)
    if rp in {"kronis", "jantung", "asma", "diabetes", "disabilitas"}:
        return 1
    if sg in {"kurang", "stunting", "gizi buruk"}:
        return 1
    if (bmi is not None) and (bmi < 18.5):
        return 1
    if (bmi is not None) and ((bmi < 17) or (bmi > 35)):
        return 1

    # Tidak layak (0)
    return 0
```

Terapkan aturan domain per sumber untuk menghasilkan layak_subsidi (0/1) tanpa mengubah file asli.

```python
# Terapkan aturan ke masing-masing client (tanpa mengubah file asal)
dinsos_lab    = dinsos.copy()
dukcapil_lab = dukcapil.copy()
kemenkes_lab = kemenkes.copy()

dinsos_lab["layak_subsidi"]    = dinsos_lab.apply(label_dinsos, axis=1)
dukcapil_lab["layak_subsidi"] = dukcapil_lab.apply(label_dukcapil, axis=1)
kemenkes_lab["layak_subsidi"] = kemenkes_lab.apply(label_kemenkes, axis=1)
```

3. Samakan skema fitur lintas-klien

- Tambah fitur turunan **BMI** bila ada tinggi/berat.
- Bentuk **vocabulary global** untuk kolom kategorikal, lalu **one-hot** agar urutan & jumlah kolom konsisten.
- Imputasi numerik (median) untuk NaN.

```python
# ============================================================
# 3) Samakan skema fitur (union fitur, one-hot, scaling global)
# ============================================================
# Kelompok kolom kategorikal & numerik yang mungkin ada
cat_cols_all = [
    "kondisi_rumah", "status_pekerjaan", "status_pernikahan",
    "riwayat_penyakit", "status_gizi"
]
num_cols_all = [
    "jumlah_tanggungan", "penghasilan",
    "umur",
    "tinggi_cm", "berat_kg"
]
```

```python
# Tambahkan BMI bila memungkinkan
for df in (dinsos_lab, dukcapil_lab, kemenkes_lab):
    if set(["tinggi_cm","berat_kg"]).issubset(df.columns):
        df["BMI"] = df["berat_kg"] / ((df["tinggi_cm"]/100.0) ** 2)
    else:
        df["BMI"] = np.nan
if "BMI" not in num_cols_all:
    num_cols_all.append("BMI")
```

```python
def union_categories(series_list):
    cats = set()
    for s in series_list:
        if s is not None:
            vals = s.dropna().unique().tolist()
            for v in vals:
                if isinstance(v, str):
                    cats.add(v)
    return sorted(list(cats))

global_vocabs = {
    "kondisi_rumah":     union_categories([dinsos_lab.get("kondisi_rumah")]),
    "status_pekerjaan":  union_categories([dukcapil_lab.get("status_pekerjaan")]),
    "status_pernikahan": union_categories([dukcapil_lab.get("status_pernikahan")]),
    "riwayat_penyakit":  union_categories([kemenkes_lab.get("riwayat_penyakit")]),
    "status_gizi":       union_categories([kemenkes_lab.get("status_gizi")]),
}
```

```python
def encode_and_scale(df):
    # Pastikan semua kolom ada (yang tidak ada → NaN)
    for c in num_cols_all:
        if c not in df.columns:
            df[c] = np.nan
    for c in cat_cols_all:
        if c not in df.columns:
            df[c] = np.nan

    # One-hot kategori dengan vocabulary global
    oh_parts = []
    for col, vocab in global_vocabs.items():
        col_series = df[col]
        for v in vocab:
            name = f"{col}__{v}"
            oh_parts.append((name, (col_series == v).astype(float)))

    oh_df = pd.DataFrame({name: arr.values for name, arr in oh_parts}, index=df.index) if oh_parts else pd.DataFrame(index=df.index)

    # Numerik: isi NaN pakai median kolom
    num_df = df[num_cols_all].copy()
    for c in num_cols_all:
        if not np.isfinite(num_df[c]).any():
            num_df[c] = 0.0
        else:
            num_df[c] = num_df[c].fillna(num_df[c].median())

    X_raw = pd.concat([num_df, oh_df], axis=1)
    y = df["layak_subsidi"].astype(int).values
    return X_raw, y

X_dinsos_raw,   y_dinsos   = encode_and_scale(dinsos_lab)
X_dukcapil_raw, y_dukcapil = encode_and_scale(dukcapil_lab)
X_kemenkes_raw, y_kemenkes = encode_and_scale(kemenkes_lab)
```

4. Scaling global yang konsisten
   Gabungkan semua fitur → hitung min/max global → terapkan min-max scaling ke tiap klien.

```python
# Min-max scaling GLOBAL (gabungkan semua agar konsisten)
all_X = pd.concat([X_dinsos_raw, X_dukcapil_raw, X_kemenkes_raw], axis=0)
mins  = all_X.min(axis=0)
maxs  = all_X.max(axis=0)
rng   = (maxs - mins).replace(0, 1.0)

def scale_like_global(X):
    return (X - mins) / rng

X_dinsos   = scale_like_global(X_dinsos_raw).fillna(0.0)
X_dukcapil = scale_like_global(X_dukcapil_raw).fillna(0.0)
X_kemenkes = scale_like_global(X_kemenkes_raw).fillna(0.0)

FEATURE_COLS = list(X_dinsos.columns)  # sama untuk semua client
```

5. Siapkan dataset Tensors
   Konversi tiap (X, y) klien ke tf.data.Dataset (shuffle + batch).

```python
# ============================================================
def df_to_tf_dataset(features_df, y_array, batch_size=32, shuffle=True):
    X = features_df.values.astype("float32")
    y = y_array.astype("float32").reshape(-1, 1)
    ds = tf.data.Dataset.from_tensor_slices((X, y))
    if shuffle:
        ds = ds.shuffle(buffer_size=len(y), reshuffle_each_iteration=True)
    ds = ds.batch(batch_size)
    return ds

client_ds = [
    df_to_tf_dataset(X_dinsos,   y_dinsos,   batch_size=32, shuffle=True),
    df_to_tf_dataset(X_dukcapil, y_dukcapil, batch_size=32, shuffle=True),
    df_to_tf_dataset(X_kemenkes, y_kemenkes, batch_size=32, shuffle=True),
]
```

6. Bangun model Keras sederhana
   MLP: Dense(64) → Dense(32) → Dense(1, sigmoid) untuk klasifikasi biner.

```python
def create_keras_model(input_dim):
    inputs = tf.keras.Input(shape=(input_dim,))
    x = tf.keras.layers.Dense(64, activation="relu")(inputs)
    x = tf.keras.layers.Dense(32, activation="relu")(x)
    outputs = tf.keras.layers.Dense(1, activation="sigmoid")(x)
    model = tf.keras.Model(inputs, outputs)
    return model
```

7. Wrap ke TFF
   Gunakan tff.learning.models.from_keras_model dengan input_spec, BinaryCrossentropy, dan metrik BinaryAccuracy.

```python
input_spec = client_ds[0].element_spec  # (TensorSpec for X, TensorSpec for y)

def model_fn():
    keras_model = create_keras_model(input_dim=len(FEATURE_COLS))
    return tff.learning.models.from_keras_model(
        keras_model=keras_model,
        input_spec=input_spec,
        loss=tf.keras.losses.BinaryCrossentropy(from_logits=False),
        metrics=[tf.keras.metrics.BinaryAccuracy(name="binary_accuracy")]
    )
```

8. **Konfigurasi Federated Averaging**
   Build FedAvg dengan optimizer klien (SGD lr=0.001) dan server (SGD lr=1.0).

```python
federated_averaging = tff.learning.algorithms.build_weighted_fed_avg(
    model_fn,
    client_optimizer_fn=tff.learning.optimizers.build_sgdm(learning_rate=0.001),
    server_optimizer_fn=tff.learning.optimizers.build_sgdm(learning_rate=1.0)
)

state = federated_averaging.initialize()
```

9. Training federated
   Inisialisasi state → loop beberapa ronde → tiap ronde: latih di klien, agregasi ke server, update model global, dan log loss & binary_accuracy.

   Menggunakan 5 round :

```python
NUM_ROUNDS = 5
for round_num in range(1, NUM_ROUNDS + 1):
    result = federated_averaging.next(state, client_ds)
    state = result.state
    train_metrics = result.metrics["client_work"]["train"]
    acc = float(train_metrics.get("binary_accuracy", 0.0))
    loss = float(train_metrics.get("loss", 0.0))
    print(f"Round {round_num:02d} | loss={loss:.4f} | bin_acc={acc:.4f}")

print("Training selesai.")
```

```
Skipping registering GPU devices...
Round 01 | loss=0.7373 | bin_acc=0.2724
Round 02 | loss=0.7282 | bin_acc=0.2933
Round 03 | loss=0.7196 | bin_acc=0.3313
Round 04 | loss=0.7115 | bin_acc=0.4058
Round 05 | loss=0.7037 | bin_acc=0.5049
Training selesai.
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/Gabungan Materi 3.1 - 3.3$
```

Menggunakan 10 round :

```python
NUM_ROUNDS = 10
for round_num in range(1, NUM_ROUNDS + 1):
    result = federated_averaging.next(state, client_ds)
    state = result.state
    train_metrics = result.metrics["client_work"]["train"]
    acc = float(train_metrics.get("binary_accuracy", 0.0))
    loss = float(train_metrics.get("loss", 0.0))
    print(f"Round {round_num:02d} | loss={loss:.4f} | bin_acc={acc:.4f}")

print("Training selesai.")
```

```
Skipping registering GPU devices...
Round 01 | loss=0.7176 | bin_acc=0.3789
Round 02 | loss=0.7076 | bin_acc=0.4562
Round 03 | loss=0.6983 | bin_acc=0.5231
Round 04 | loss=0.6895 | bin_acc=0.5876
Round 05 | loss=0.6812 | bin_acc=0.6109
Round 06 | loss=0.6734 | bin_acc=0.6080
Round 07 | loss=0.6660 | bin_acc=0.6658
Round 08 | loss=0.6590 | bin_acc=0.7147
Round 09 | loss=0.6524 | bin_acc=0.7180
Round 10 | loss=0.6461 | bin_acc=0.7282
Training selesai.
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/Gabungan Materi 3.1 - 3.3$
```