Modeling

1. **Muat Data**

```
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_csv("processed_kelulusan.csv")
X = df.drop("Lulus", axis=1)
y = df["Lulus"]

X_train, X_temp, y_train, y_temp = train_test_split(
    X, y, test_size=0.3, stratify=y, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, random_state=42)

print(X_train.shape, X_val.shape, X_test.shape)
```

[3] Python

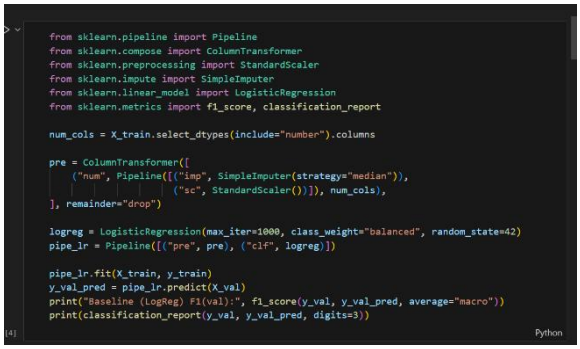
```
(7, 5) (1, 5) (2, 5)
```

- df = pd.read_csv("processed_kelulusan.csv")
- X = df tanpa kolom target "Lulus", y = df["Lulus"]
- Dua panggilan train_test_split:
    - Pertama: test_size=0.3, stratify=y, random_state=42 → menghasilkan X_train, X_temp dengan shapes yang tercetak: **(7,5)** untuk train, **(1,5)** untuk val (ternyata hasil akhir), **(2,5)** untuk test — printed shapes (7,5) (1,5) (2,5).

Interpretasi & catatan penting

- Dataset sangat kecil: total sampel = 7+1+2 = 10 (atau total 10). Ini **sangat rentan overfitting** dan membuat estimasi performa tidak stabil.
- Kamu memakai stratify=y — bagus untuk menjaga proporsi kelas, tapi dengan sampel sedikit, setiap split bisa berisi sangat sedikit (0/1/2) contoh per kelas sehingga beberapa fold/partition jadi hanya berisi satu kelas.
- Akibatnya metrik seperti ROC-AUC atau metrik berbasis ranking bisa tidak terdefinisi jika test/val hanya mengandung satu kelas.

2. **Baseline Model & Pipeline**



```python
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, classification_report

num_cols = X_train.select_dtypes(include="number").columns

pre = ColumnTransformer([
    ("num", Pipeline([("imp", SimpleImputer(strategy="median")),
                      ("sc", StandardScaler())]), num_cols),
], remainder="drop")

logreg = LogisticRegression(max_iter=1000, class_weight="balanced", random_state=42)
pipe_lr = Pipeline([("pre", pre), ("clf", logreg)])

pipe_lr.fit(X_train, y_train)
y_val_pred = pipe_lr.predict(X_val)
print("Baseline (LogReg) F1(val):", f1_score(y_val, y_val_pred, average="macro"))
print(classification_report(y_val, y_val_pred, digits=3))
```

```
Baseline (LogReg) F1(val): 1.0
              precision    recall  f1-score   support

           1      1.000     1.000     1.000         1

    accuracy                         1.000         1
   macro avg      1.000     1.000     1.000         1
weighted avg      1.000     1.000     1.000         1
```

- Membuat ColumnTransformer bernama pre untuk kolom numerik:
    - SimpleImputer(strategy="median") diikuti StandardScaler()
    - remainder="drop"
- Pipeline pipe_lr = Pipeline([("pre", pre), ("clf", logreg)])
- logreg = LogisticRegression(max_iter=1000, class_weight="balanced", random_state=42)
- Fit di X_train, prediksi ke X_val, lalu print F1 (macro) dan classification report.

Interpretasi

- Kamu menerapkan preprocessing di dalam pipeline — ini benar/baik: mencegah data leakage.
- class_weight="balanced" berguna kalau kelas tidak seimbang.
- Ini adalah **baseline** yang sahih: simple, interpretable, dan diperlakukan adil dengan pipeline dan stratify.
- Baseline (LogReg) F1(val): 1.0
- classification_report menunjukkan precision/recall/f1 = 1.000 untuk kelas yang ada di val; support = 1 (hanya 1 sample di val untuk kelas tersebut).

Interpretasi & peringatan

- F1(val)=1.0 terdengar sempurna — namun support di val hanya 1 contoh. Itu **tidak meyakinkan**: kinerja sempurna pada 1 contoh tidak berarti model generalizes.
- Karena dataset kecil, skor sempurna sangat mungkin tanda **overfitting** atau kebetulan split yang “mudah”.

3. **Model Alternatif (Random Forest)**

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(
    n_estimators=300, max_features="sqrt", class_weight="balanced", random_state=42
)
pipe_rf = Pipeline([("pre", pre), ("clf", rf)])

pipe_rf.fit(X_train, y_train)
y_val_rf = pipe_rf.predict(X_val)
print("RandomForest F1(val):", f1_score(y_val, y_val_rf, average="macro"))
```

[5] Python

```
RandomForest F1(val): 1.0
```

- RandomForestClassifier(n_estimators=300, max_features="sqrt", class_weight="balanced", random_state=42)
- Pipeline pipe_rf = Pipeline([("pre", pre), ("clf", rf)])
- Fit dan prediksi pada X_val, lalu print F1(val)

Hasil

- RandomForest F1(val): 1.0

Interpretasi

- RandomForest juga mencapai F1=1.0 pada validation set yang sama — konsisten dengan baseline, tetapi masih terpengaruh masalah sample kecil.
- Karena kedua model sempurna di val, selanjutnya dilakukan tuning/CV agar memilih model final secara lebih andal.

4. Validasi Silang & Tuning Ringkas

```python
from sklearn.metrics import confusion_matrix, roc_auc_score, precision_recall_curve, roc_curve
import matplotlib.pyplot as plt

final_model = best_rf  # atau pipe_lr jika baseline lebih baik
y_test_pred = final_model.predict(X_test)

print("F1(test):", f1_score(y_test, y_test_pred, average="macro"))
print(classification_report(y_test, y_test_pred, digits=3))
print("Confusion matrix (test):")
print(confusion_matrix(y_test, y_test_pred))

# ROC-AUC (jika ada predict_proba)
if hasattr(final_model, "predict_proba"):
    y_test_proba = final_model.predict_proba(X_test)[:,1]
    try:
        print("ROC-AUC(test):", roc_auc_score(y_test, y_test_proba))
    except:
        pass
    fpr, tpr, _ = roc_curve(y_test, y_test_proba)
    plt.figure(); plt.plot(fpr, tpr); plt.xlabel("FPR"); plt.ylabel("TPR"); plt.title("ROC (test)")
    plt.tight_layout(); plt.savefig("roc_test.png", dpi=120)
```

```
F1(test): 1.0
              precision    recall  f1-score   support

           0      1.000     1.000     1.000         2

    accuracy                          1.000         2
   macro avg      1.000     1.000     1.000         2
weighted avg      1.000     1.000     1.000         2

Confusion matrix (test):
[[2]]
ROC-AUC(test): nan
d:\Machine Learning\venv\lib\site-packages\sklearn\metrics\_classification.py:534: UserWarning: A single la
  warnings.warn(
d:\Machine Learning\venv\lib\site-packages\sklearn\metrics\_ranking.py:424: UndefinedMetricWarning: Only on
  warnings.warn(
d:\Machine Learning\venv\lib\site-packages\sklearn\metrics\_ranking.py:1201: UndefinedMetricWarning: No pos
  warnings.warn(
```

- final_model = best_rf (atau pipe_lr jika dipilih)
- y_test_pred = final_model.predict(X_test)
- Print f1_score(y_test, y_test_pred, average="macro"), classification_report, confusion_matrix
- ROC-AUC: jika ada predict_proba, coba y_test_proba = final_model.predict_proba(X_test)[:,1] lalu roc_auc_score dan roc_curve & savefig.

Hasil (7.7):

- F1(test): 1.0
- classification_report shows support=2 (test set only had 2 samples) and perfect metrics
- Confusion matrix printed as [[2]] (1x1 matrix -> only one class present in y_test)
- ROC-AUC(test): nan and sklearn warnings:
    - UserWarning: A single label…
    - UndefinedMetricWarning: Only one class present in y_true. ROC AUC is not defined in that case.

Penjelasan

- Test set sangat kecil dan **mungkin hanya berisi satu kelas** — itulah kenapa confusion matrix adalah [[2]] dan ROC-AUC menjadi nan (tidak bisa dihitung ketika y_test memiliki satu kelas saja).
- Skor F1=1.0 valid untuk test set tersebut, tapi karena test sangat kecil dan hanya satu kelas, nilai ini **tidak valid** sebagai indikator generalisasi.

5. **Evaluasi Akhir (Test Set)**

```
        # Prediksi label
        yhat = MODEL.predict(X)[0]

        # Prediksi probabilitas (jika ada)
        proba = None
        if hasattr(MODEL, "predict_proba"):
            proba = float(MODEL.predict_proba(X)[0, 1])

        # Kembalikan hasil dalam format JSON
        return jsonify({"prediction": int(yhat), "proba": proba})

    except Exception as e:
        # Jika ada error, kirimkan pesan error dalam JSON
        return jsonify({"error": str(e)}), 400


if __name__ == "__main__":
    app.run(port=5000)
```



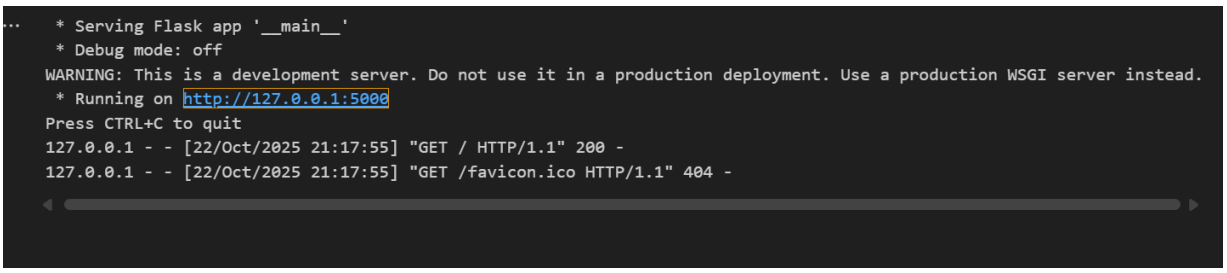
```
        # Prediksi label
        yhat = MODEL.predict(X)[0]

        # Prediksi probabilitas (jika ada)
        proba = None
        if hasattr(MODEL, "predict_proba"):
            proba = float(MODEL.predict_proba(X)[0, 1])

        # Kembalikan hasil dalam format JSON
        return jsonify({"prediction": int(yhat), "proba": proba})

    except Exception as e:
        # Jika ada error, kirimkan pesan error dalam JSON
        return jsonify({"error": str(e)}), 400


if __name__ == "__main__":
    app.run(port=5000)
```



```
 * Serving Flask app '__main__'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [22/Oct/2025 21:17:55] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [22/Oct/2025 21:17:55] "GET /favicon.ico HTTP/1.1" 404 -
```

joblib.dump(final_model, "model.pkl")

Flask app:

- MODEL = joblib.load("model.pkl")
- Route / GET → message "Flask API is running!"
- Route /predict POST → ambil JSON body data = request.get_json(force=True) lalu X = pd.DataFrame([data])
- yhat = MODEL.predict(X)[0]
- proba = None ; jika ada predict_proba, ambil float(MODEL.predict_proba(X)[0,1])
- return jsonify({"prediction": int(yhat), "proba": proba})