

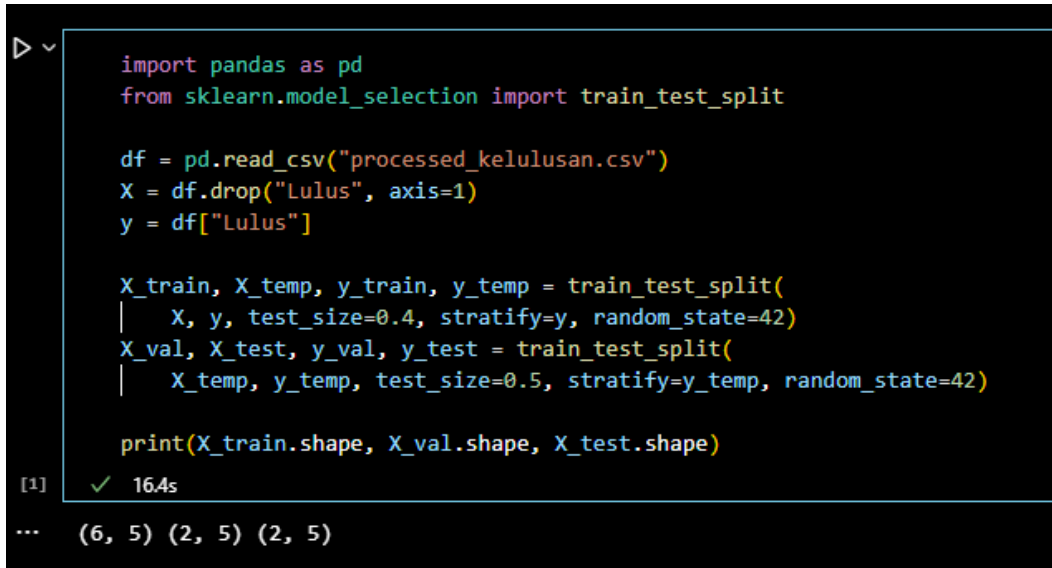
LAPORAN PERTEMUAN 5

Nama : Muhammad Salman Imamwan Abdillah

Nim : 231011401032

Kelas : 05TPLE017

1. Pada tahap ini dilakukan proses pembacaan dataset hasil pra-pemrosesan dan pembagian data untuk keperluan pelatihan model. Dataset `processed_kelulusan.csv` dibaca menggunakan library Pandas, kemudian kolom *Lulus* dipisahkan sebagai variabel target (label), sedangkan kolom lainnya digunakan sebagai fitur. Selanjutnya data dibagi menjadi tiga bagian menggunakan fungsi `train_test_split`, yaitu data latih (60%), data validasi (20%), dan data uji (20%). Pembagian dilakukan secara *stratified* agar proporsi kelas tetap seimbang dan diatur dengan *random_state* agar hasil konsisten.



```
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=0.4, stratify=y, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, stratify=y_temp, random_state=42)

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

[1] ✓ 16.4s

... (6, 5) (2, 5) (2, 5)

2. Pada tahap ini dilakukan pelatihan model klasifikasi menggunakan algoritma *Logistic Regression*. Sebelum model dilatih, data numerik melalui proses pra-pemrosesan yang meliputi pengisian nilai kosong dengan median dan standarisasi menggunakan *StandardScaler*. Tahapan tersebut diintegrasikan ke dalam pipeline bersama model *Logistic Regression* agar alur pelatihan lebih efisien dan terstruktur. Hasil evaluasi pada data validasi menunjukkan nilai *F1-score* sebesar 1.0 atau 100%, yang berarti model mampu memprediksi seluruh data validasi dengan benar. Namun, hasil ini masih bersifat sementara karena ukuran dataset kecil sehingga perlu pengujian lebih lanjut pada data yang lebih besar untuk menilai keakuratan model secara menyeluruh.

```

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))

```

[2] ✓ 1.8s

```

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

      0       1.000      1.000      1.000         1
      1       1.000      1.000      1.000         1

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

```

- Model kedua yang digunakan adalah *Random Forest Classifier*, yaitu algoritma berbasis *ensemble* yang menggabungkan beberapa pohon keputusan untuk meningkatkan akurasi dan mengurangi overfitting. Model ini dilatih dengan parameter utama `n_estimators=300`, `max_features="sqrt"`, dan `class_weight="balanced"`. Sama seperti sebelumnya, proses pelatihan dilakukan melalui pipeline yang mencakup tahap imputasi dan standarisasi data numerik. Berdasarkan hasil evaluasi menggunakan data validasi, model Random Forest menghasilkan nilai *F1-score* sebesar 1.0 (100%), yang menunjukkan performa sempurna pada data tersebut. Namun, hasil ini masih perlu divalidasi lebih lanjut dengan data uji yang lebih besar agar keandalan model dapat dipastikan.

```

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"))

```

RandomForest F1(val): 1.0

4. Proses tuning parameter dilakukan menggunakan *GridSearchCV* dengan *Stratified 3-Fold Cross Validation* untuk model Random Forest. Parameter terbaik yang diperoleh adalah `max_depth=None` dan `min_samples_split=2`, dengan nilai F1-macro sebesar 1.0 pada data pelatihan maupun validasi. Hasil ini menunjukkan model memiliki performa klasifikasi yang sangat baik tanpa indikasi kesalahan prediksi.

```

from sklearn.model_selection import StratifiedKFold, GridSearchCV

skf = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)
param = {
    "clf_max_depth": [None, 12, 20, 30],
    "clf_min_samples_split": [2, 5, 10]
}
gs = GridSearchCV(pipe_rf, param_grid=param, cv=skf,
    | | | | | | | scoring="f1_macro", n_jobs=-1, verbose=1)
gs.fit(X_train, y_train)
print("Best params:", gs.best_params_)
print("Best CV F1:", gs.best_score_)

best_rf = gs.best_estimator_
y_val_best = best_rf.predict(X_val)
print("Best RF F1(val):", f1_score(y_val, y_val_best, average="macro"))

```

[8]

```

... Fitting 3 folds for each of 12 candidates, totalling 36 fits
Best params: {'clf_max_depth': None, 'clf_min_samples_split': 2}
Best CV F1: 1.0
Best RF F1(val): 1.0

```

5. Evaluasi model dilakukan menggunakan metrik F1-score, Confusion Matrix, dan ROC-AUC. Hasil menunjukkan nilai F1-score, precision, recall, dan akurasi sebesar 1.0, dengan ROC-AUC juga mencapai 1.0. Confusion matrix memperlihatkan tidak adanya kesalahan klasifikasi pada data uji. Hal ini menunjukkan performa model sangat baik, namun perlu diperhatikan kemungkinan overfitting karena jumlah data uji yang kecil.

```

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)

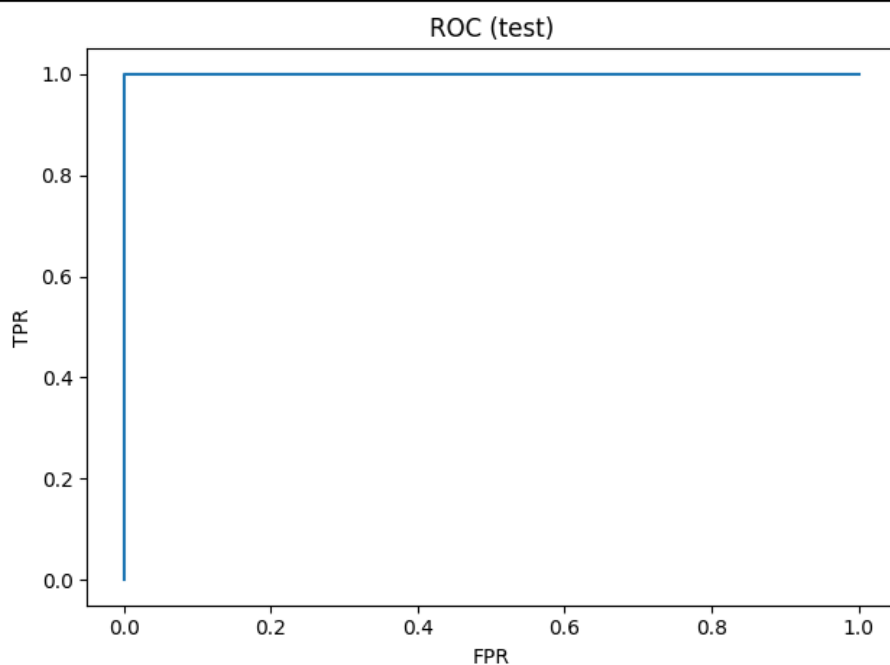
```

F1(test): 1.0

	precision	recall	f1-score	support
0	1.000	1.000	1.000	1
1	1.000	1.000	1.000	1
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):
[[1 0]
[0 1]]

ROC-AUC(test): 1.0



6. Model hasil pelatihan disimpan menggunakan library *joblib* dalam format file *.pkl*. Proses ini bertujuan agar model dapat digunakan kembali tanpa perlu dilakukan pelatihan ulang, sehingga mempermudah proses implementasi dan deployment. File yang dihasilkan bernama *model.pkl*, yang berisi seluruh parameter dan struktur model akhir.

```
import joblib
joblib.dump(final_model, "model.pkl")
print("Model tersimpan ke model.pkl")
```

Model tersimpan ke model.pkl

7. Aplikasi *Flask* dibuat untuk menyediakan layanan prediksi berbasis web API. Model hasil pelatihan yang telah disimpan dalam file *model.pkl* dimuat dan digunakan untuk memproses input data yang dikirim melalui metode POST pada endpoint */predict*. Aplikasi ini mengembalikan hasil prediksi dan probabilitas dalam format JSON, sehingga dapat dengan mudah diintegrasikan ke dalam sistem lain atau antarmuka pengguna. Server dijalankan secara lokal pada port 5000 menggunakan *Flask development server*.

```
from flask import Flask, request, jsonify
import joblib, pandas as pd

app = Flask(__name__)
MODEL = joblib.load("model.pkl")

@app.route("/predict", methods=["POST"])
def predict():
    data = request.get_json(force=True) # dict fitur
    X = pd.DataFrame([data])
    yhat = MODEL.predict(X)[0]
    proba = None
    if hasattr(MODEL, "predict_proba"):
        proba = float(MODEL.predict_proba(X[:,1])[0])
    return jsonify({"prediction": int(yhat), "proba": proba})

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

Python

```
* 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 - - [18/Oct/2025 13:04:36] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [18/Oct/2025 13:04:36] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [18/Oct/2025 13:14:08] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [18/Oct/2025 13:14:08] "GET /favicon.ico HTTP/1.1" 404 -
```