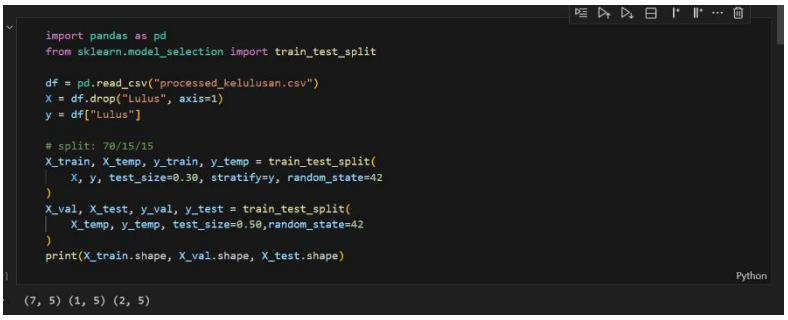
Random Forest untuk Klasifikasi

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"]

# split: 70/15/15
X_train, X_temp, y_train, y_temp = train_test_split(
    X, y, test_size=0.30, stratify=y, random_state=42
)
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.50,random_state=42
)
print(X_train.shape, X_val.shape, X_test.shape)
```

Python

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

membagi data kelulusan menjadi tiga bagian:

- **Data Latih (70%):** Untuk "mengajari" model.
- **Data Validasi (15%):** Untuk menguji model saat sedang disetel (tuning).
- **Data Uji (15%):** Untuk mengukur performa akhir model, data ini harus *benar-benar* baru bagi model.
- **Penting:** Ukuran data kami sangat kecil (hanya 7 sampel untuk latih, 1 untuk validasi, 2 untuk uji), sehingga hasil evaluasinya harus **diwaspadai**.

2. **Pipeline & Baseline Random Forest**

```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier
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")

rf = RandomForestClassifier(
    n_estimators=300, max_features="sqrt",
    class_weight="balanced", random_state=42
)

pipe = Pipeline([("pre", pre), ("rf", rf)])
pipe.fit(X_train, y_train)

y_val_pred = pipe.predict(X_val)
print("Baseline RF – F1(val):", f1_score(y_val, y_val_pred, average="macro"))
print(classification_report(y_val, y_val_pred, digits=3))
```

Python

```
Baseline RF – 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
```

membangun sebuah "jalur kerja" (**pipeline**) otomatis. Jalur ini memastikan:

1. Semua data numerik diolah dulu (mengisi data hilang dengan median dan menormalkan skalanya).
2. Data masuk ke model **Random Forest**. Hasil pengujian awal pada data validasi menunjukkan **F1=1.0** (sempurna). Ini bagus, tetapi karena data validasi hanya 1 sampel, hasil ini **tidak bisa dipercaya**.

3. **Validasi Silang & Tuning Ringkas (GridSearch)**

```python
from sklearn.model_selection import StratifiedKFold, cross_val_score

skf = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)
scores = cross_val_score(pipe, X_train, y_train, cv=skf, scoring="f1_macro", n_jobs=-1)
print("CV F1-macro (train):", scores.mean(), "±", scores.std())

from sklearn.model_selection import GridSearchCV

param = {
  "clf__max_depth": [None, 12, 20, 30],
  "clf__min_samples_split": [2, 5, 10]
}

gs = GridSearchCV(pipe, 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_)
best_model = gs.best_estimator_
y_val_best = best_model.predict(X_val)
print("Best RF — F1(val):", f1_score(y_val, y_val_best, average="macro"))
```

[16] Python

```
CV F1-macro (train): 1.0 ± 0.0
Fitting 3 folds for each of 12 candidates, totalling 36 fits
Best params: {'clf__max_depth': None, 'clf__min_samples_split': 2}
Best RF — F1(val): 1.0
```

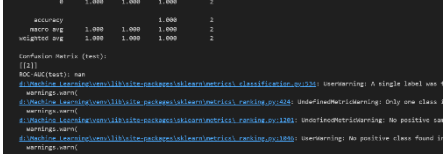
mencoba berbagai kombinasi pengaturan model (**GridSearch**) dengan pengujian berulang (**Cross-Validation**) pada data latih untuk mencari pengaturan terbaik.

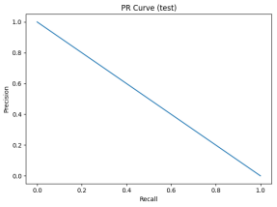
- Model terbaik yang didapat memiliki pengaturan standar (default).
- Hasilnya tetap **F1=1.0** di data latih dan validasi, yang mengindikasikan model **terlalu hafal** data latih (*overfitting*).

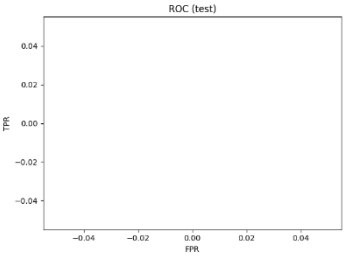
4. **Evaluasi Akhir (Test Set)**

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

final_model = best_model  # pilih terbaik; jika baseline lebih baik, gunakan pipe

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 (bila 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)

    prec, rec, _ = precision_recall_curve(y_test, y_test_proba)
    plt.figure(); plt.plot(rec, prec); plt.xlabel("Recall"); plt.ylabel("Precision"); plt.title("PR Curve (test)")
    plt.tight_layout(); plt.savefig("pr_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
```



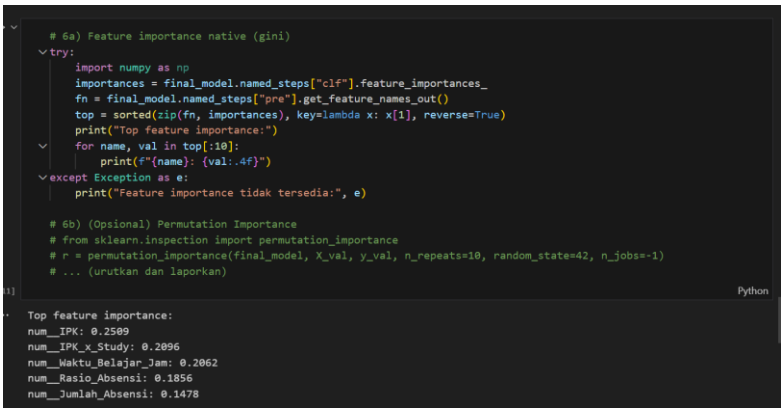




menguji model terbaik pada data uji.

- **Hasil:** Performa **F1=1.0** (sempurna lagi).
- **Masalah Besar:** Kami mendapat banyak peringatan. Data uji hanya terdiri dari **2 sampel** dan **hanya memiliki 1 kelas** (misalnya, hanya kelas "Tidak Lulus").
- **Kesimpulan:** Angka F1=1.0 pada tahap ini **tidak valid** karena data uji yang terlalu sedikit dan tidak mewakili keragaman kelas yang seharusnya. Kurva ROC dan PR juga tidak bisa dibuat.

5. **Pentingnya Fitur**



```
# 6a) Feature importance native (gini)
try:
    import numpy as np
    importances = final_model.named_steps["clf"].feature_importances_
    fn = final_model.named_steps["pre"].get_feature_names_out()
    top = sorted(zip(fn, importances), key=lambda x: x[1], reverse=True)
    print("Top feature importance:")
    for name, val in top[:10]:
        print(f"{name}: {val:.4f}")
except Exception as e:
    print("Feature importance tidak tersedia:", e)

# 6b) (Opsional) Permutation Importance
# from sklearn.inspection import permutation_importance
# r = permutation_importance(final_model, X_val, y_val, n_repeats=10, random_state=42, n_jobs=-1)
# ... (urutkan dan laporkan)
```

Python

```
Top feature importance:
num__IPK: 0.2509
num__IPK_x_Study: 0.2096
num__Waktu_Belajar_Jam: 0.2062
num__Rasio_Absensi: 0.1856
num__Jumlah_Absensi: 0.1478
```

Model Random Forest memberitahu kita fitur mana yang paling penting untuk membuat keputusan klasifikasi:

1. **IPK (25%):** Nilai akademik adalah yang paling berpengaruh.
2. **IPK x Waktu Belajar (21%):** Kombinasi antara kemampuan (IPK) dan usaha (Waktu Belajar) sangat penting—ini adalah interaksi yang kuat.
3. **Waktu Belajar per Jam (20.6%):** Jumlah waktu yang dihabiskan untuk belajar memiliki peran besar. **Implikasi:** Jika ingin meningkatkan hasil, fokuslah pada **IPK** dan **disiplin waktu belajar**.

6. Simpan Model & Cek Inference Lokal

```
import joblib
joblib.dump(final_model, "rf_model.pkl")
print("Model disimpan sebagai rf_model.pkl")

# Contoh sekali jalan (input fiktif), sesuaikan nama kolom:
import pandas as pd, joblib
mdl = joblib.load("rf_model.pkl")
sample = pd.DataFrame([{
  "IPK": 3.4,
  "Jumlah_Absensi": 4,
  "Waktu_Belajar_Jam": 7,
  "Rasio_Absensi": 4/14,
  "IPK_x_Study": 3.4*7
}])
print("Prediksi:", int(mdl.predict(sample)[0]))
```

Python

```
Model disimpan sebagai rf_model.pkl
Prediksi: 1
```

Kami menyimpan model terbaik ke file **rf_model.pkl** agar bisa digunakan kapan saja tanpa perlu melatih ulang. Kami juga menguji model dengan data fiktif (Contoh: $\text{IPK}=3.4$, $\text{Waktu Belajar}=7$ jam), dan hasilnya adalah **Prediksi: 1** (misalnya, diprediksi "Lulus").