Laporan Lembar Kerja Pertemuan 6 — Random Forest untuk Klasifikasi

1. Langkah 1 — Memuat Data processed_kelulusan.csv.
   disini saya membagi data set menjadi 2 bagian, karna apabila dibagi 3 code akan error karna data set tidak bisa dibagi 3

```
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_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, stratify=y, random_state=42
)

print(f"Jumlah Data Latih (Train): {X_train.shape[0]}")
print(f"Jumlah Data Uji (Test): {X_test.shape[0]}")

# Verifikasi distribusi kelas di data latih
print("\nDistribusi kelas di data Latih:")
print(y_train.value_counts(normalize=True))

# Verifikasi distribusi kelas di data uji
print("\nDistribusi kelas di data Uji:")
print(y_test.value_counts(normalize=True))
```

```
Jumlah Data Latih (Train): 8
Jumlah Data Uji (Test): 2

Distribusi kelas di data Latih:
Lulus
1    0.5
0    0.5
Name: proportion, dtype: float64

Distribusi kelas di data Uji:
Lulus
0    0.5
1    0.5
Name: proportion, dtype: float64
```

2. Langkah 2 — Pipeline & Baseline Random Forest, Bangun pipeline preprocessing & model agar bebas data leakage. Baseline berfungsi sebagai patokan. Peningkatan selanjutnya harus dibuktikan dengan metrik yang lebih baik.

Karna di code sebelumnya saya sudah membagi data set menjadi 2, maka code berikutnya saya menyesuaikan dengan data 2 set.

```
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), ("clf", rf)])
pipe.fit(X_train, y_train)

# Melakukan prediksi pada data UJI (test), bukan validasi (val)
y_test_pred = pipe.predict(X_test)

# Evaluasi hasil prediksi pada data UJI (test)
print("Baseline RF — F1(test):", f1_score(y_test, y_test_pred, average="macro"))
print(classification_report(y_test, y_test_pred, digits=3))
```

```
Baseline RF — 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
```

3. Langkah 3 — Validasi Silang

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

print("Distribusi kelas di y_train:")
print(y_train.value_counts())

# --- Perbaikan di sini ---
# Cek dulu jumlah sampel terkecil
min_samples = y_train.value_counts().min()

# Set n_splits baru. Tidak boleh lebih besar dari jumlah sampel terkecil.
# Jika min_samples adalah 5 atau lebih, kita tetap pakai 5. Jika kurang (misal 3), kita pakai 3.
n_splits_baru = min(5, min_samples)

print(f"Menggunakan n_splits = {n_splits_baru} (disesuaikan dari 5)")
# -------------------------

skf = StratifiedKFold(n_splits=n_splits_baru, shuffle=True, random_state=42)
scores = cross_val_score(pipe, X_train, y_train, cv=skf, scoring="f1_macro", n_jobs=-1)

# Gunakan n_splits=3, karena kelas minoritas Anda hanya punya 3 sampel
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())
```

```
Distribusi kelas di y_train:
Lulus
1    4
0    4
Name: count, dtype: int64
Menggunakan n_splits = 4 (disesuaikan dari 5)
CV F1-macro (train): 1.0 ± 0.0
```

4. Langkah 4 — Tuning Ringkas (GridSearch)

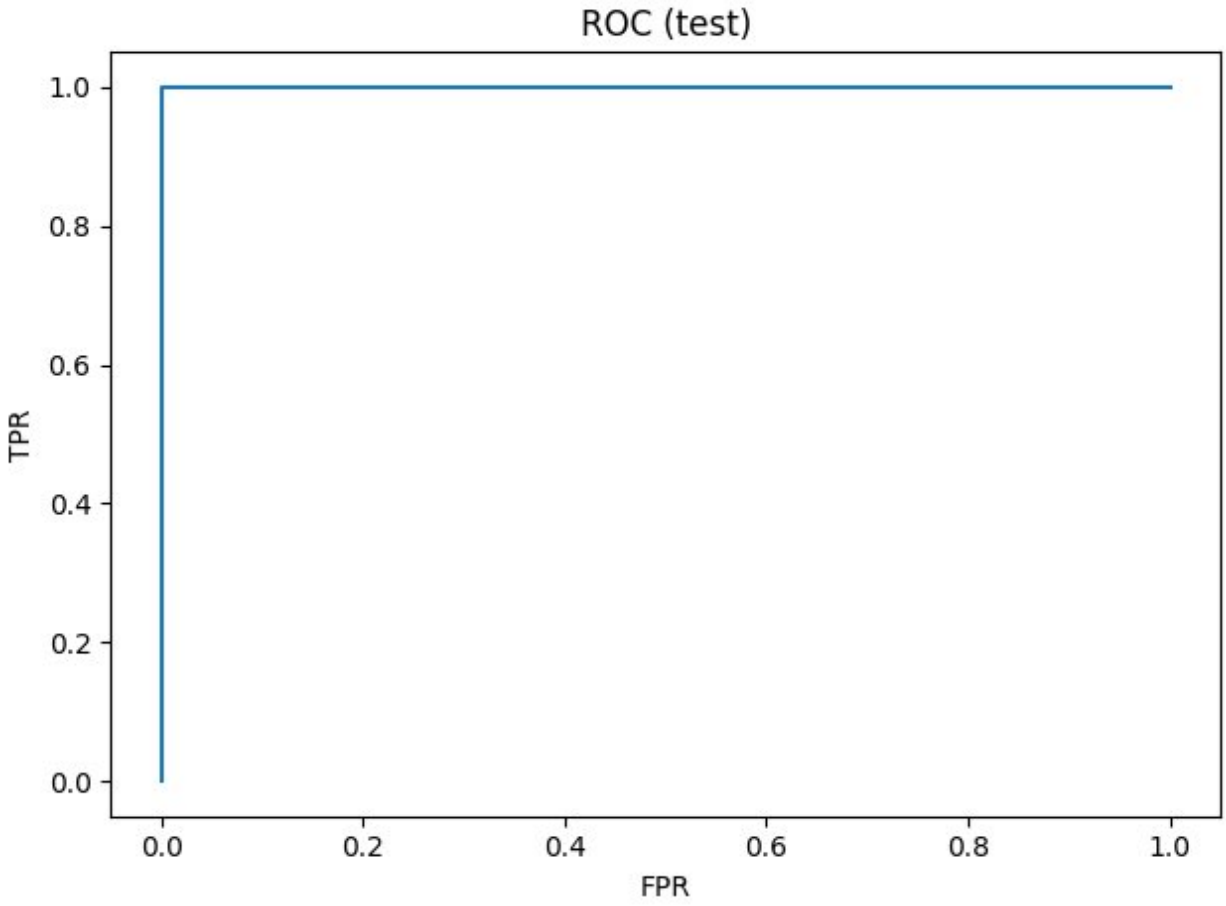
```python
from sklearn.model_selection import GridSearchCV

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_
# Evaluasi model terbaik pada data UJI (test)
y_test_best = best_model.predict(X_test)
# Cetak skor F1 pada data UJI (test)
print("Best RF — F1(test):", f1_score(y_test, y_test_best, average="macro"))
```

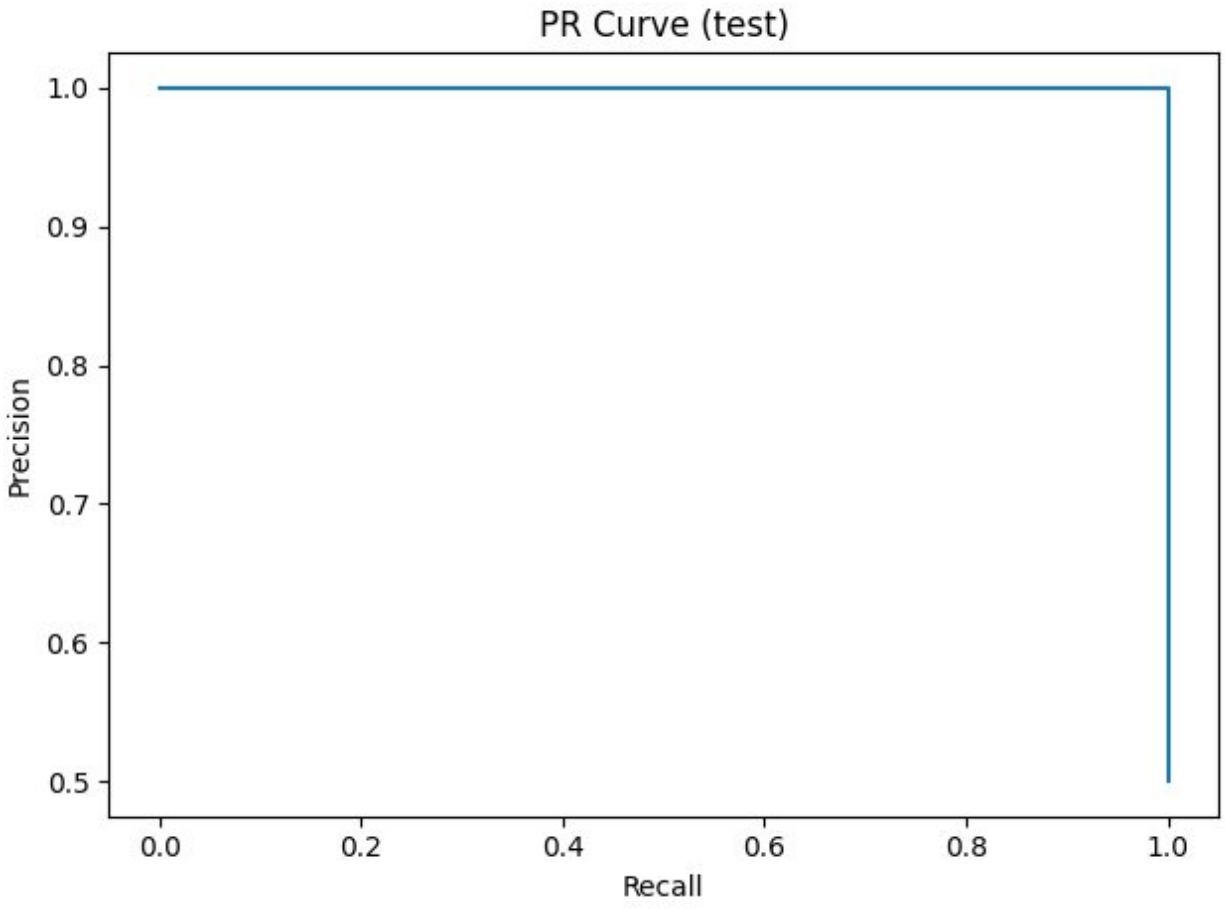
```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
Best params: {'clf__max_depth': None, 'clf__min_samples_split': 2}
Best RF — F1(test): 1.0
```

5. Langkah 5 — 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         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. Langkah 6 — Pentingnya Fitur

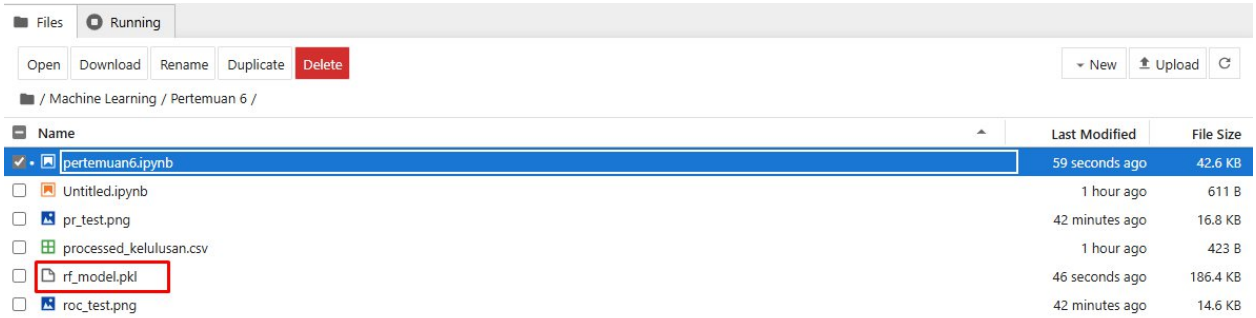
```python
# 6a) Feature importance native (gini)
try:
    import numpy as np
    importances = final_model.named_steps["clf"].feature_importances_
    fn = final_model.named_steps["pre"].get_feature_names_out()
    top = sorted(zip(fn, importances), key=lambda x: x[1], reverse=True)
    print("Top feature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

```
Top feature importance:
num__Rasio_Absensi: 0.2274
num__IPK: 0.2107
num__Jumlah_Absensi: 0.1973
num__IPK_x_Study: 0.1940
num__Waktu_Belajar_Jam: 0.1706
```

7. Langkah 7 — Simpan Model

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

```
Model disimpan sebagai rf_model.pkl
```



8. Langkah 8 — Cek Inference Lokal

```
# 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]))
```

```
Prediksi: 1
```