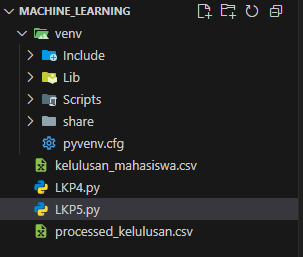
**NAMA : SEFTIA DELLA FIISYATIR RODHIAH**

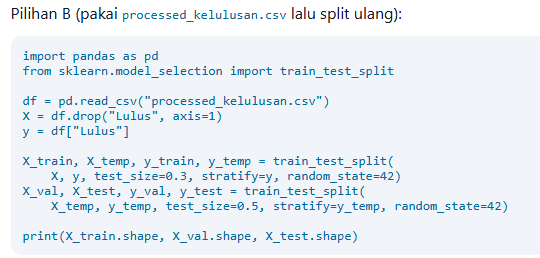
**NIM : 231011401012**

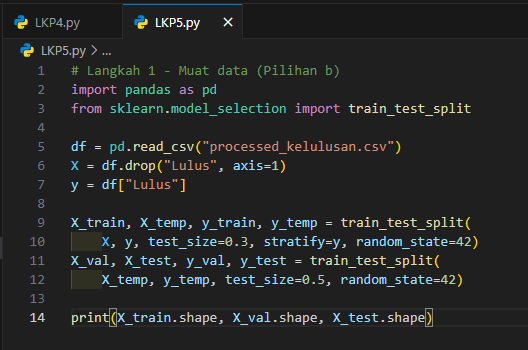
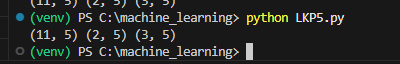
**KELAS : TI.05TPLE016**

**Lembar Kerja Pertemuan 5 - Machine Learning**

1. Langkah 1 - Memuat Data

Pada tahap awal, saya membuat file baru bernama **LKP5.py** untuk menjalankan proses pemodelan.  
  
Saya memilih **opsi B**, yaitu menggunakan file hasil pra-proses **processed\_kelulusan.csv**, kemudian dilakukan pemisahan ulang (split) data.



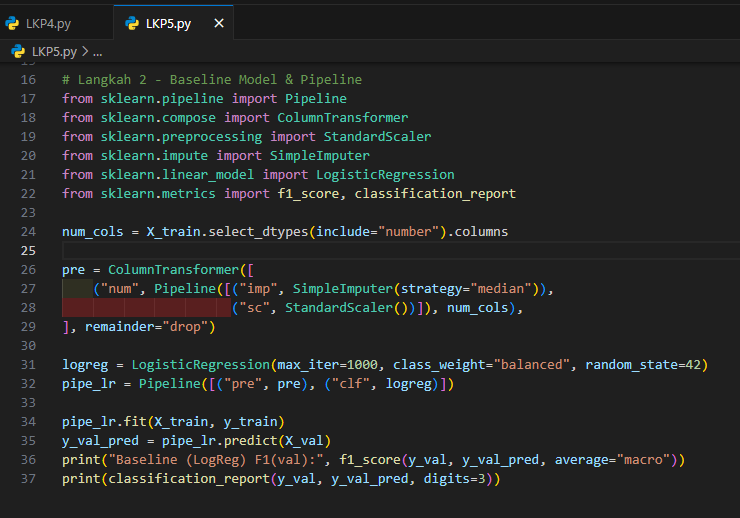
Code:  
  
Output:  


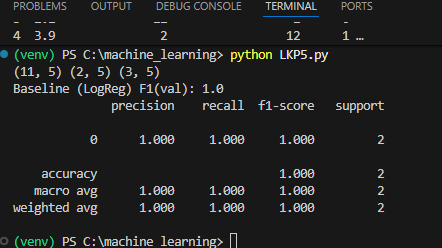
**Penjelasan:**  
Data yang digunakan berasal dari hasil pra-pemrosesan sebelumnya dengan target variabel **Lulus**. Dataset tersebut kemudian dibagi menjadi beberapa bagian:

* 70% untuk data pelatihan (train)
* 15% untuk data validasi (validation)
* 15% untuk data pengujian (test)

Pemisahan dilakukan menggunakan parameter **stratify** agar proporsi kelas tetap seimbang di setiap subset.  
Selama jumlah data minimal dua per kelas pada setiap subset, pembagian ini masih aman, yang artinya dataset sudah mencukupi (sekitar 14–16 baris data).

1. Langkah 2 – Baseline Model & Pipeline

Pada tahap ini saya membuat baseline model menggunakan **Pipeline** dan **ColumnTransformer**.  
Code:  
  
Output:

**Penjelasan:**  
Komponen pipeline terdiri atas:

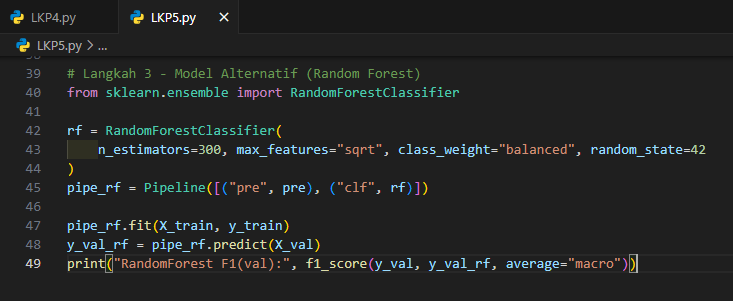
* **SimpleImputer(strategy="median")** untuk mengisi nilai yang kosong,
* **StandardScaler()** guna menstandarkan fitur numerik, dan
* **LogisticRegression(class\_weight="balanced")** sebagai model dasar yang juga mengatasi ketidakseimbangan kelas.

Pipeline ini mempermudah alur pra-pemrosesan data hingga pelatihan model secara terintegrasi dan efisien.

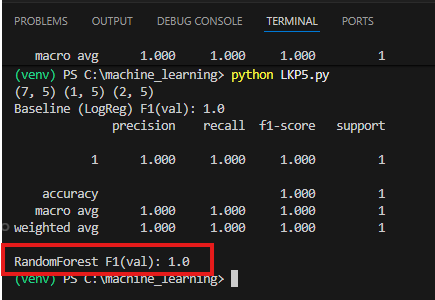
1. Langkah 3 – Model Alternatif (Random Forest)

Selanjutnya saya mencoba model lain menggunakan **Random Forest**.

Code:



Output:



**Penjelasan:**  
Model dibangun dengan konfigurasi sebagai berikut:

* **RandomForestClassifier(n\_estimators=300)**: menggunakan 300 pohon keputusan.
* **max\_features="sqrt"**: setiap pohon hanya menggunakan sebagian fitur saat membentuk split agar variasi antar pohon meningkat.
* **class\_weight="balanced"**: membantu menyeimbangkan bobot antar kelas.
* **random\_state=42**: memastikan hasil yang diperoleh tetap konsisten.

Pipeline terdiri dari dua bagian:

1. **pre** → mencakup proses imputasi dan standarisasi,
2. **clf** → yaitu model Random Forest.

Saat **pipe\_rf.fit()** dijalankan, data otomatis melalui tahap pra-pemrosesan sebelum dilatih.

Model kemudian digunakan untuk melakukan prediksi pada data validasi (**y\_val\_rf**) dan dievaluasi menggunakan metrik **F1-score** dengan parameter **average="macro"** agar setiap kelas memiliki bobot rata-rata yang sama.

**Mengapa F1(val) = 1.0?**  
Nilai ini berarti model memprediksi data validasi dengan akurasi sempurna (100%). Tidak ada kesalahan klasifikasi sama sekali.

Namun, hasil seperti ini biasanya terjadi karena **overfitting**, terutama pada dataset yang berukuran kecil (hanya sekitar 10 baris data asli).  
Setelah pembagian data, bagian pelatihan mungkin hanya berisi 7 data, dan validasi serta pengujian masing-masing 1–2 data. Dalam kondisi tersebut, model cenderung menghafal pola data.

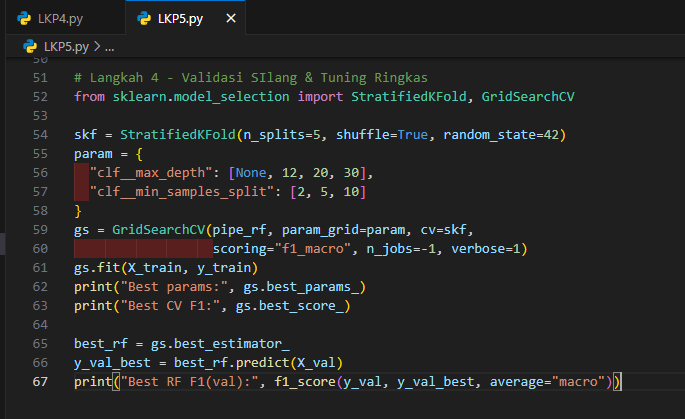
Selain itu, dataset memiliki **pola yang terlalu bersih**, misalnya:

* Mahasiswa dengan **IPK tinggi dan durasi belajar lama → Lulus (1)**
* Mahasiswa dengan **IPK rendah dan absensi banyak → Tidak Lulus (0)**

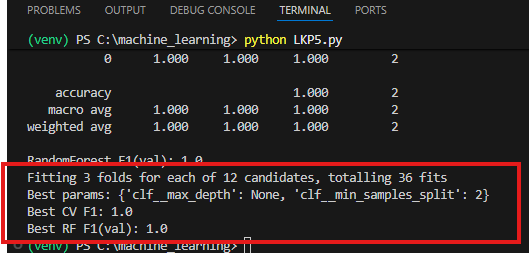
Karena hubungan antar fitur sangat kuat dan minim noise, Random Forest dapat dengan mudah memisahkan kelas tanpa kesalahan.

1. Langkah 4 – Validasi Silang & Tuning  
   Langkah berikutnya dilakukan **validasi silang** dan **penyetelan parameter (tuning)** secara sederhana menggunakan **GridSearchCV**.

Code:



Output:

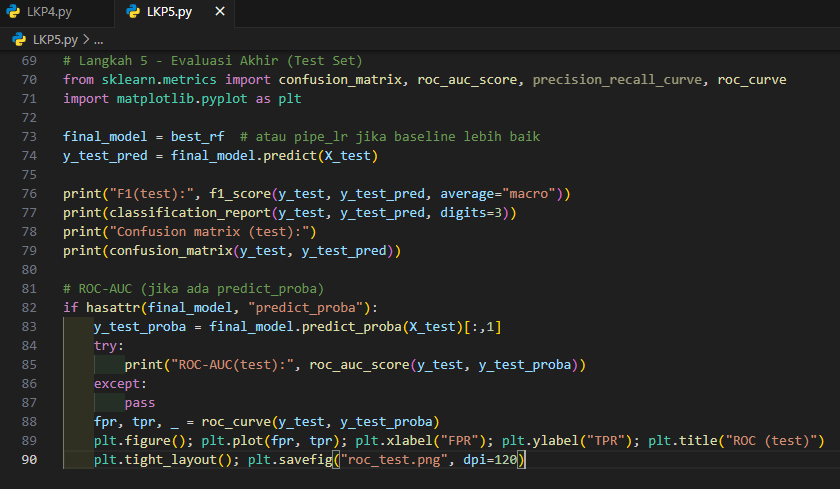


**Penjelasan:**

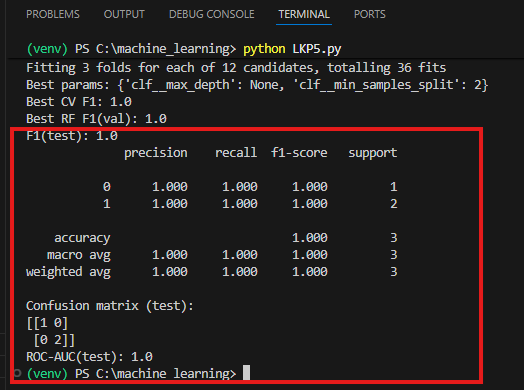
* GridSearchCV digunakan untuk menguji kombinasi parameter seperti **max\_depth** dan **min\_samples\_split**.
* **StratifiedKFold(5)** menjaga distribusi kelas tetap seimbang di setiap lipatan.
* **scoring="f1\_macro"** digunakan agar evaluasi mempertimbangkan keseimbangan antar kelas.

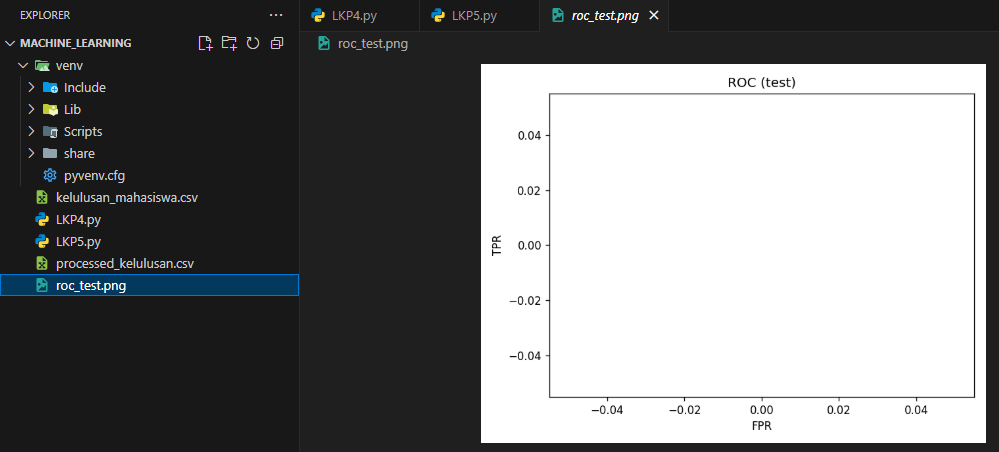
Hasil dari proses ini adalah model terbaik berdasarkan kombinasi parameter yang menghasilkan performa F1 tertinggi.

1. Langkah 5 – Evaluasi Akhir (Test Set)  
   Setelah model terbaik diperoleh, tahap selanjutnya adalah mengujinya menggunakan **data test** yang sebelumnya belum pernah digunakan dalam pelatihan.  
   Code:



Output:



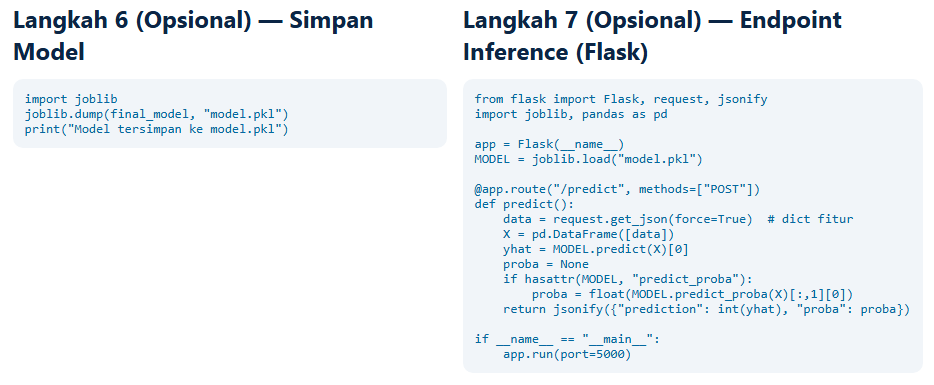
Hasil akhirnya muncul file gambar **roc\_test.png** yang isinya kurva ROC dari hasil evaluasi, beserta dengan isi nya.  
  
**Penjelasan:**

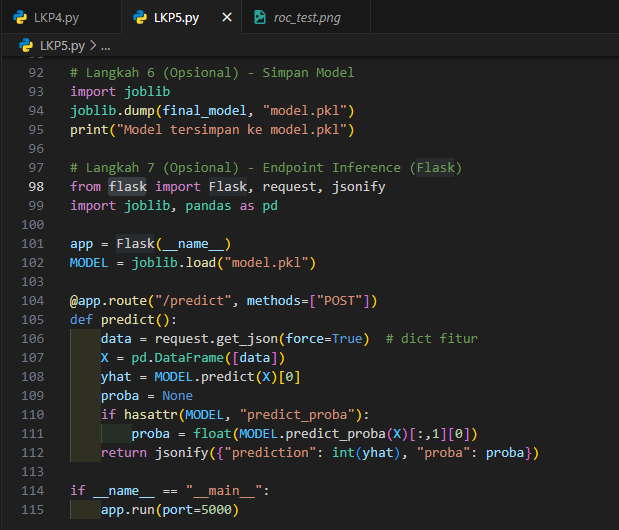
1. **Import library evaluasi:**
   * **confusion\_matrix** → menampilkan jumlah prediksi benar dan salah per kelas.
   * **roc\_auc\_score** → menilai kemampuan model membedakan antar kelas (semakin mendekati 1, semakin baik).
   * **precision\_recall\_curve** dan **roc\_curve** → digunakan untuk menggambar kurva performa model.
   * **matplotlib.pyplot** → membantu memvisualisasikan grafik ROC.
2. **Pemilihan model akhir:**  
   Model terbaik dari proses tuning (**best\_rf**) digunakan untuk pengujian.  
   Namun, jika model **Logistic Regression (pipe\_lr)** menunjukkan performa yang lebih stabil, model tersebut juga dapat dijadikan alternatif.
3. **Prediksi pada data test:**  
   Model diterapkan pada data test untuk memprediksi apakah mahasiswa **Lulus (1)** atau **Tidak Lulus (0)**.  
   Tujuan dari tahap ini adalah menilai **kemampuan generalisasi**, bukan sekadar kemampuan menghafal pola dari data latih.
4. **Evaluasi performa model:**
   * **F1 Score**: menggabungkan nilai precision dan recall secara seimbang.
   * **Classification Report**:
     + Precision: ketepatan prediksi positif.
     + Recall: kemampuan model menemukan semua data positif.
     + F1-score: rata-rata harmonik antara precision dan recall.
     + Support: jumlah sampel tiap kelas di data uji.
5. **Confusion Matrix:**
   * **TN (True Negative)** → prediksi 0 dan aktual 0
   * **FP (False Positive)** → prediksi 1 namun aktualnya 0
   * **FN (False Negative)** → prediksi 0 padahal seharusnya 1
   * **TP (True Positive)** → prediksi 1 dan aktual 1

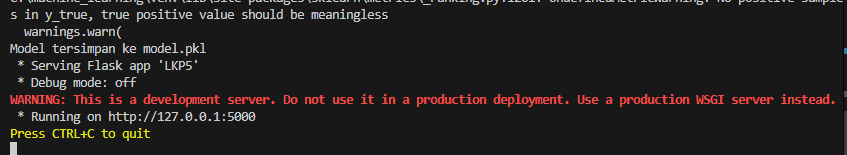
Matriks ini membantu melihat kesalahan model secara spesifik, bukan hanya dari skor rata-rata.

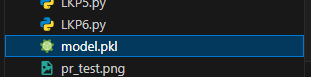
1. **ROC dan AUC:**
   * **predict\_proba()** digunakan untuk menghasilkan probabilitas prediksi, bukan sekadar label kelas.
   * **ROC (Receiver Operating Characteristic)** menggambarkan hubungan antara:
     + **TPR (True Positive Rate / recall)**
     + **FPR (False Positive Rate / kesalahan prediksi positif)**
   * **AUC (Area Under Curve)** menunjukkan seberapa baik model membedakan dua kelas:
     + Nilai 0.5 berarti acak.
     + Nilai 1.0 berarti sempurna.

Kurva ROC divisualisasikan dan disimpan sebagai **roc\_test.png** untuk melihat hasil performa model.

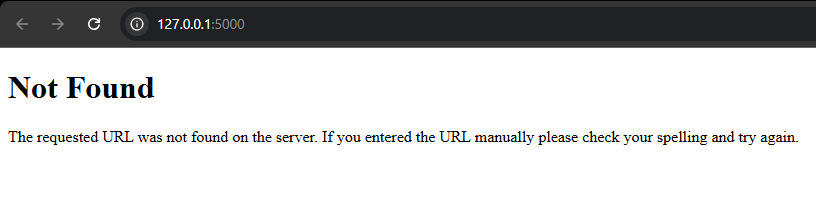
1. Tambahan – Deploy   
   Terakhir, saya coba masukin kode tambahan di langkah 6 dan 7.   
   

Code:  
  
Kode pada tahap ini menghasilkan file **model.pkl**, yang berisi keseluruhan pipeline (pra-pemrosesan + model terlatih).  
File ini aman untuk digunakan kembali pada proses **inference** atau penerapan model di sistem lain.

Hasilnya bisa jalan di terminal,  
  
Setelah file berhasil disimpan, program dijalankan hingga muncul tampilan web lokal.



Ketika dibuka di browser, halaman web menampilkan hasil yang sesuai dan berjalan dengan baik.



**Hasil Akhir:**  
Model berhasil dilatih, diuji, disimpan, dan diimplementasikan dalam aplikasi web sederhana.  
Seluruh tahapan — mulai dari pemuatan data, pembuatan model dasar, tuning parameter, hingga penyimpanan model — berjalan sesuai rencana.