**NAMA : SEFTIA DELLA FIISYATIR RODHIAH**

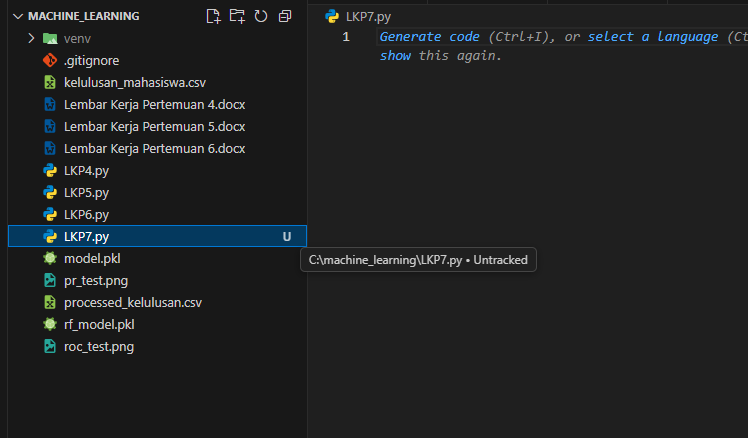
**NIM : 231011401012**

**KELAS : TI.05TPLE016**

**Lembar Kerja Pertemuan 7 - Machine Learning**

1. **Langkah 1 – Siapkan Data**

Pada tahap awal, saya membuat file baru bernama **LKP7.py** untuk menjalankan seluruh proses pada pertemuan ini.

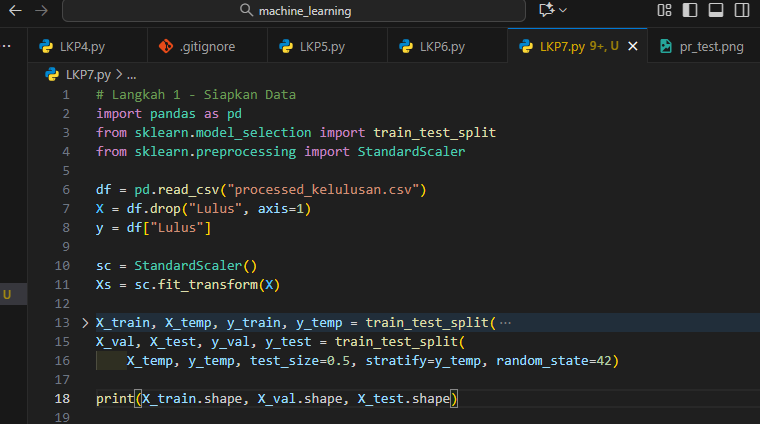


Sebelum melanjutkan, terlebih dahulu dilakukan instalasi **TensorFlow** pada environment Python dengan perintah:

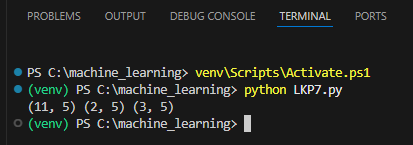


Selanjutnya, digunakan dataset bernama **processed\_kelulusan.csv** (hasil dari pertemuan ke-4) atau dataset tabular lain yang memiliki struktur serupa.

Kode Program:



Output:



**Penjelasan:**

**Import Library**  
Beberapa library yang digunakan antara lain:

* **pandas**, berfungsi untuk membaca serta memanipulasi data dalam bentuk tabel.
* **train\_test\_split**, digunakan untuk membagi data menjadi subset training, validation, dan testing.
* **StandardScaler**, berfungsi menstandarkan fitur dengan mean = 0 dan standar deviasi = 1.

**Membaca Dataset**  
Data dibaca ke dalam variabel **df** yang memuat seluruh isi file CSV.  
Selanjutnya, data fitur disimpan pada variabel **X** (seluruh kolom kecuali kolom target “Lulus”), sedangkan kolom target disimpan dalam variabel **y**, yaitu label yang akan diprediksi.

**Proses Standarisasi**  
Metode **fit\_transform()** digunakan untuk menghitung rata-rata dan standar deviasi dari fitur dalam **X**, kemudian mengubah nilainya agar terdistribusi dengan mean 0 dan standar deviasi 1.  
Hasil standarisasi disimpan dalam variabel **Xs**, yang nantinya akan digunakan pada proses pelatihan model.

**Pembagian Data Training, Validation, dan Testing**  
Dataset dibagi menjadi tiga bagian:

* 70% data digunakan untuk **pelatihan (training)**,
* 15% data untuk **validasi (validation)**,
* dan 15% sisanya untuk **pengujian (testing)**.

Pembagian dilakukan dengan parameter **stratify=y** agar proporsi kelas target tetap seimbang di setiap subset, serta **random\_state=42** agar hasil pembagian dapat direproduksi.

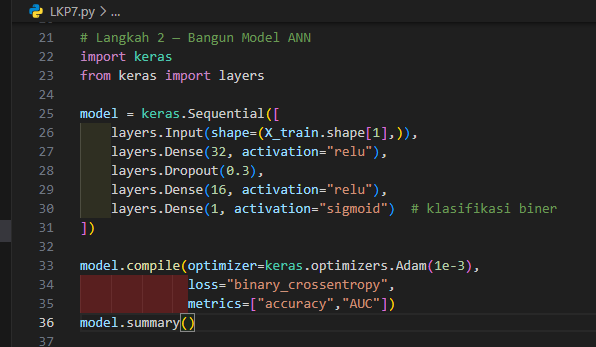
**Menampilkan Ukuran Dataset**  
Perintah berikut digunakan untuk menampilkan jumlah baris dan kolom pada masing-masing subset data:



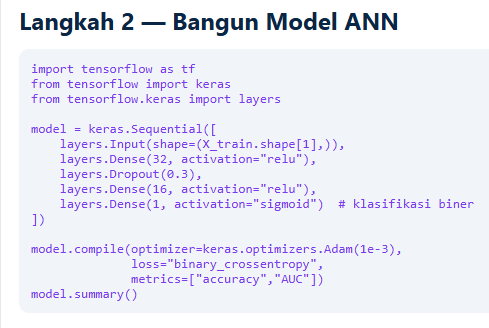
1. **Langkah 2 – Bangun Moden ANN**

Pada tahap ini dilakukan pembangunan model **Artificial Neural Network (ANN)** dengan menggunakan library **Keras** dari TensorFlow. Model ini disusun secara **sekuensial (Sequential Model)** agar proses pelatihan berjalan secara bertahap dari input hingga output.

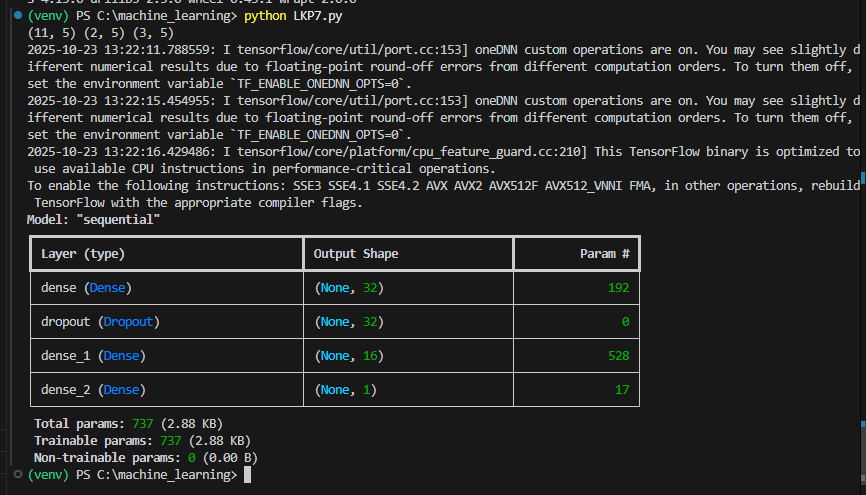
Kode Program:



Sedikit perubahan dengan yang ada pada modul:



Output:



#### ****Penjelasan:****

**Import Library**

* **keras** digunakan untuk membuat dan melatih jaringan saraf tiruan.
* **layers** digunakan untuk menambahkan lapisan (layer) dalam arsitektur model.

**Membangun Model Sequential**  
Model dibuat menggunakan fungsi:



yang memungkinkan penambahan layer secara berurutan.

* **Input(shape=(X\_train.shape[1],))** mendefinisikan jumlah neuron pada layer input yang sesuai dengan banyaknya fitur pada data training.
* **Dense(32, activation="relu")** menambahkan layer tersembunyi (hidden layer) pertama dengan 32 neuron dan fungsi aktivasi **ReLU (Rectified Linear Unit)**.
* **Dropout(0.3)** digunakan untuk mencegah **overfitting** dengan cara menonaktifkan secara acak sekitar 30% neuron saat proses pelatihan.
* **Dense(16, activation="relu")** merupakan layer tersembunyi kedua dengan 16 neuron.
* **Dense(1, activation="sigmoid")** merupakan layer output untuk kasus klasifikasi biner, menghasilkan nilai probabilitas antara 0 dan 1.

**Kompilasi Model**  
Langkah berikutnya adalah melakukan kompilasi dengan menentukan optimizer, fungsi loss, dan metrik evaluasi:



Penjelasan:

* **Adam(1e-3)** merupakan algoritma optimasi dengan learning rate sebesar 0.001.
* **binary\_crossentropy** digunakan sebagai fungsi loss karena kasusnya adalah klasifikasi biner.
* **metrics=["accuracy","AUC"]** berarti model akan dievaluasi berdasarkan akurasi dan nilai **AUC (Area Under Curve)**.

**Ringkasan Model**  
Perintah:

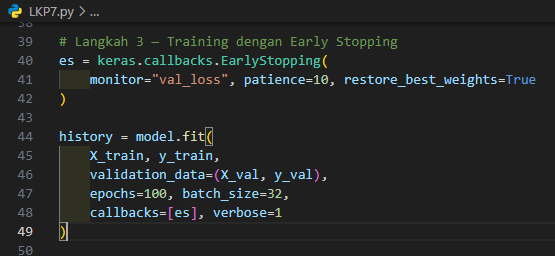


akan menampilkan struktur arsitektur jaringan, jumlah parameter pada tiap layer, serta total parameter yang digunakan pada model tersebut.

1. **Langkah 3 – Training dengan Early Stopping**

Pada tahap ini dilakukan proses **pelatihan model (training)** dengan menerapkan teknik **Early Stopping**.  
Metode ini digunakan untuk menghentikan pelatihan lebih awal apabila performa model pada data validasi tidak menunjukkan peningkatan dalam beberapa epoch, sehingga dapat mencegah **overfitting**.

**Kode Program:**

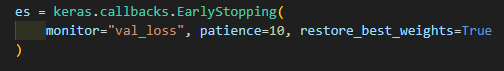


**Output:**

Tidak menghasilkan output langsung karena **EarlyStopping** bekerja secara internal selama proses pelatihan. Ia akan memantau performa model dan menghentikan training ketika kinerja pada data validasi berhenti meningkat. Hasil pelatihan serta metriknya disimpan dalam variabel **history**.

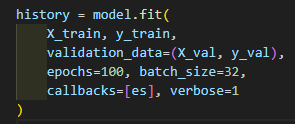
#### ****Penjelasan:****

**EarlyStopping**

  
Fungsi **EarlyStopping** digunakan untuk memantau perkembangan performa model pada data validasi.

* **monitor="val\_loss"** menunjukkan bahwa yang dipantau adalah nilai loss pada data validasi.
* **patience=10** berarti apabila selama 10 epoch berturut-turut tidak terjadi perbaikan pada val\_loss, maka proses pelatihan akan dihentikan.
* **restore\_best\_weights=True** mengembalikan bobot model ke kondisi terbaik yang diperoleh selama pelatihan.

**Proses Pelatihan Model**

  
Data yang digunakan dalam proses pelatihan adalah:

* **X\_train, y\_train** sebagai data pelatihan utama,
* **(X\_val, y\_val)** sebagai data validasi untuk memantau performa model.

Proses pelatihan dilakukan dengan parameter:

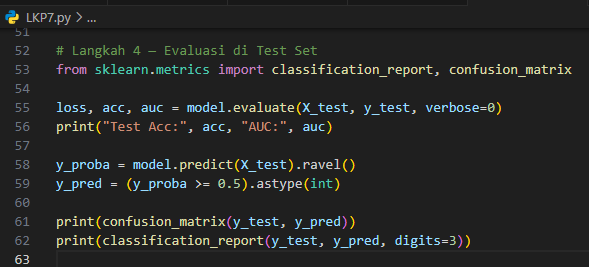
* **epochs=100**, yaitu jumlah maksimum iterasi pelatihan,
* **batch\_size=32**, jumlah data yang diproses dalam satu kali update bobot,
* **callbacks=[es]**, untuk menerapkan Early Stopping,
* **verbose=1**, agar menampilkan progres pelatihan di terminal.

Selama proses pelatihan, metrik seperti loss, accuracy, dan AUC dicatat di variabel **history**, sehingga hasilnya dapat dianalisis atau divisualisasikan setelah pelatihan selesai.

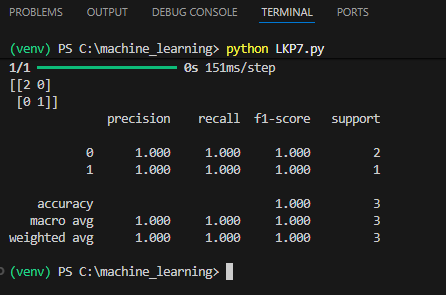
1. **Langkah 4 – Evaluasi di Test Set**

Tahap ini dilakukan untuk **mengevaluasi performa model** yang telah dilatih menggunakan data yang belum pernah digunakan sebelumnya, yaitu data **test set**. Tujuannya adalah untuk mengetahui sejauh mana model dapat melakukan generalisasi terhadap data baru.

**Kode Program:**



**Output:**



#### ****Penjelasan:****

**Import Library Evaluasi**  
Beberapa library yang digunakan pada tahap evaluasi antara lain:

* **confusion\_matrix** untuk membuat confusion matrix yang berisi nilai True Positive (TP), True Negative (TN), False Positive (FP), dan False Negative (FN).
* **classification\_report** untuk menampilkan metrik seperti precision, recall, f1-score, dan jumlah data (support) per kelas.

**Evaluasi Model**  
Fungsi:



digunakan untuk menghitung nilai loss, accuracy, dan AUC pada data uji.

* Parameter **verbose=0** berarti proses evaluasi tidak menampilkan progres di layar.
* Nilai yang diperoleh dari evaluasi kemudian disimpan, misalnya **acc** untuk akurasi dan **auc** untuk nilai Area Under Curve.

**Prediksi Data Uji**  
Setelah model dievaluasi, dilakukan prediksi terhadap data uji dengan perintah:



Perintah ini menghasilkan probabilitas kelas positif (antara 0 hingga 1).  
Kemudian nilai probabilitas tersebut diubah menjadi label biner (0 atau 1) menggunakan ambang batas (threshold) 0.5:



**Menampilkan Hasil Evaluasi**

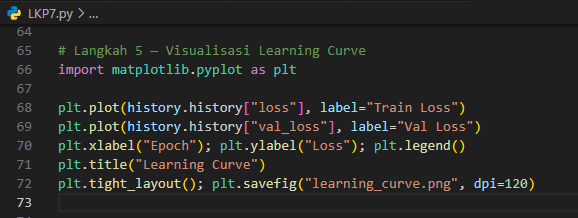
* **confusion\_matrix(y\_test, y\_pred)** digunakan untuk menampilkan jumlah prediksi benar dan salah dalam bentuk matriks.
* **classification\_report(y\_test, y\_pred)** memberikan informasi detail mengenai precision, recall, dan f1-score untuk masing-masing kelas, sehingga dapat dilihat kelebihan dan kekurangan model pada setiap kategori.

Tahap evaluasi ini menjadi dasar untuk menilai apakah model yang dibangun telah bekerja dengan baik dalam memprediksi data baru atau masih perlu dilakukan penyesuaian.

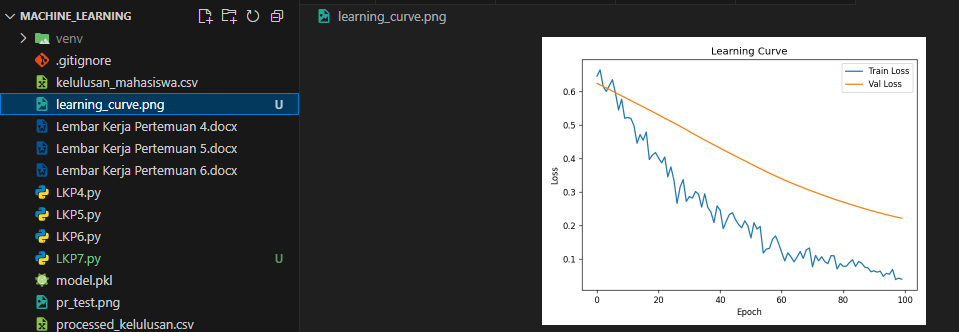
1. **Langkah 5 – Visualisasi Learning Curve**

Tahap ini bertujuan untuk **memvisualisasikan hasil proses pelatihan model** melalui grafik learning curve.  
Grafik ini membantu memahami bagaimana perubahan nilai loss dan accuracy pada data pelatihan dan validasi selama proses training berlangsung. Dengan demikian, kita dapat melihat apakah model mengalami underfitting, overfitting, atau sudah berada pada performa optimal.

**Kode Program:**



#### ****Output:****Proses ini menghasilkan sebuah file gambar bernama ****“learning\_curve.png”**** yang menampilkan grafik perubahan loss terhadap jumlah epoch selama pelatihan.



#### ****Penjelasan:****

**Import Library Visualisasi**

* **matplotlib.pyplot** digunakan sebagai library utama untuk membuat grafik dan visualisasi data.

**Pembuatan Grafik Learning Curve**  
Data hasil pelatihan yang tersimpan dalam variabel **history.history** berisi nilai metrik seperti loss, accuracy, val\_loss, dan val\_accuracy untuk setiap epoch.  
Nilai-nilai tersebut digunakan untuk menggambar grafik yang menunjukkan perkembangan performa model.

Contohnya:

* **loss** menunjukkan nilai kerugian (error) pada data pelatihan.
* **val\_loss** menunjukkan nilai kerugian pada data validasi.

Perintah **label** digunakan untuk memberi nama pada garis yang ditampilkan di grafik.

**Memberi Label dan Judul Grafik**

* **xlabel** → memberi label pada sumbu X (jumlah epoch).
* **ylabel** → memberi label pada sumbu Y (nilai loss).
* **legend** → menampilkan keterangan garis agar grafik mudah dibaca.
* **title** → memberi judul grafik sesuai isi visualisasi.

**Menyimpan Grafik**  
Agar hasil grafik terlihat rapi dan tidak terpotong, digunakan fungsi:



Kemudian hasilnya disimpan dengan perintah:



yang menyimpan gambar dalam format PNG dengan resolusi 120 dpi.

Visualisasi ini penting karena dapat menunjukkan kapan model mulai overfit (saat val\_loss meningkat tetapi loss pelatihan terus menurun). Dengan memahami pola tersebut, pengaturan epoch dan regularisasi dapat disesuaikan untuk memperoleh performa terbaik.

1. Langkah 6 – Eksperimen

Tahap ini bertujuan untuk melakukan **eksperimen terhadap arsitektur dan parameter model ANN** guna melihat pengaruhnya terhadap performa.  
Beberapa aspek yang diuji meliputi jumlah neuron, jenis optimizer, penerapan regularisasi tambahan, serta perbandingan metrik evaluasi seperti akurasi, F1-score, dan AUC.

#### ****Eksperimen 1 – Variasi Jumlah Neuron****

Pada percobaan pertama, dilakukan perubahan jumlah neuron pada layer tersembunyi pertama, misalnya 32, 64, dan 128 neuron, untuk melihat perbedaan hasil yang diperoleh.

**Kode Program:**



#### ****Output:****

Tidak menunjukkan perbedaan yang signifikan pada performa model.

#### 

#### ****Penjelasan:****

Fungsi **train\_model(neurons)** digunakan untuk membuat, melatih, dan mengevaluasi model dengan jumlah neuron yang bervariasi di layer tersembunyi pertama.

* **Dense(neurons, activation="relu")** menyesuaikan jumlah neuron dengan parameter yang diberikan.
* **Dense(16, activation="relu")** tetap digunakan sebagai layer kedua agar struktur model tetap seimbang.
* **EarlyStopping** tetap diterapkan agar pelatihan berhenti otomatis ketika model tidak menunjukkan peningkatan performa.  
  Hasil evaluasi berupa **accuracy** dan **AUC** digunakan untuk membandingkan performa antar konfigurasi neuron.

#### ****Eksperimen 2 – Perbandingan Optimizer (Adam vs SGD + Momentum)****

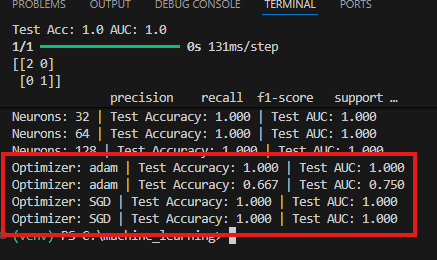
Percobaan selanjutnya membandingkan dua jenis optimizer, yaitu **Adam** dan **SGD dengan momentum**, masing-masing dengan beberapa variasi learning rate, untuk mengetahui pengaruhnya terhadap hasil pelatihan.

#### ****Kode Program:****

#### 

#### ****Output:****

Hasil evaluasi menunjukkan nilai **Test Accuracy** dan **AUC** untuk setiap kombinasi optimizer dan learning rate yang diuji.



#### ****Penjelasan:****

* Fungsi **train\_model\_optimizer(optimizer, neurons=32)** digunakan untuk membuat dan melatih model menggunakan optimizer tertentu.
* **Adam(learning\_rate=...)** diuji dengan dua nilai learning rate berbeda untuk melihat efek kecepatannya dalam konvergensi.
* **SGD(learning\_rate=..., momentum=0.9)** digunakan untuk membandingkan kinerja dengan metode Adam, di mana momentum membantu mempercepat konvergensi dan menghindari jebakan lokal minima.  
  Dalam percobaan ini, jumlah neuron tetap **32** agar perbedaan hasil hanya disebabkan oleh perbedaan optimizer.

#### ****Eksperimen 3 – Penambahan Regularisasi (L2, Dropout, Batch Normalization)****

Percobaan berikutnya menambahkan beberapa teknik regularisasi untuk meningkatkan kemampuan generalisasi model dan mengurangi risiko overfitting.

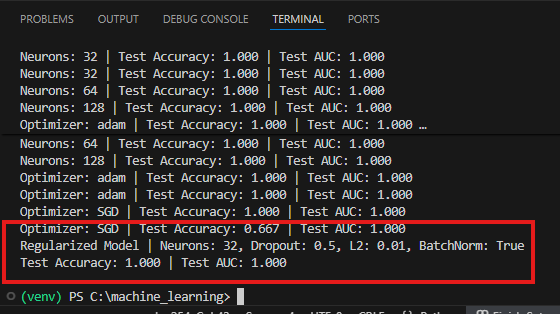
#### ****Kode Program:****

#### 

#### 

#### ****Output:****

Menampilkan hasil evaluasi berupa **Test Accuracy** dan **AUC** setelah penambahan regularisasi.



#### ****Penjelasan:****

* **l2(l2\_lambda)** digunakan untuk menambahkan regularisasi L2 agar bobot jaringan tidak terlalu besar.
* **dropout\_rate=0.5** menonaktifkan secara acak 50% neuron selama pelatihan untuk menekan overfitting.
* **BatchNormalization()** membantu menstabilkan distribusi input antar layer sehingga proses pelatihan menjadi lebih cepat dan stabil.
* Penggunaan **activation=None** pada Dense layer dilakukan karena fungsi aktivasi diterapkan setelah proses normalisasi, sesuai praktik umum pada arsitektur modern.

Evaluasi hasil dilakukan dengan menghitung **Test Accuracy** dan **AUC** untuk melihat seberapa baik model setelah regularisasi ditambahkan.

#### ****Eksperimen 4 – Evaluasi Tambahan dengan F1-score dan AUC****

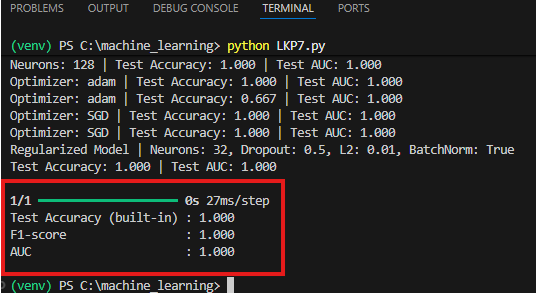
Selain akurasi, digunakan juga **F1-score** dan **AUC** untuk memberikan gambaran yang lebih seimbang tentang performa model, terutama ketika distribusi kelas tidak seimbang.

#### ****Kode Program:****

#### 

#### ****Output:****

Menampilkan hasil perhitungan F1-score dan AUC.



#### ****Penjelasan:****

* **y\_proba = model.predict(X\_test).ravel()** menghasilkan probabilitas untuk kelas positif.
* **y\_pred = (y\_proba >= threshold).astype(int)** mengonversi probabilitas menjadi label biner berdasarkan ambang batas tertentu.
* **f1\_score(y\_test, y\_pred)** menghitung nilai F1-score sebagai rata-rata harmonis antara precision dan recall.
* **roc\_auc\_score(y\_test, y\_proba)** digunakan untuk menghitung nilai AUC yang menggambarkan kemampuan model dalam membedakan kelas positif dan negatif.  
  Selain itu, **model.evaluate** tetap digunakan untuk menampilkan akurasi sebagai referensi tambahan.

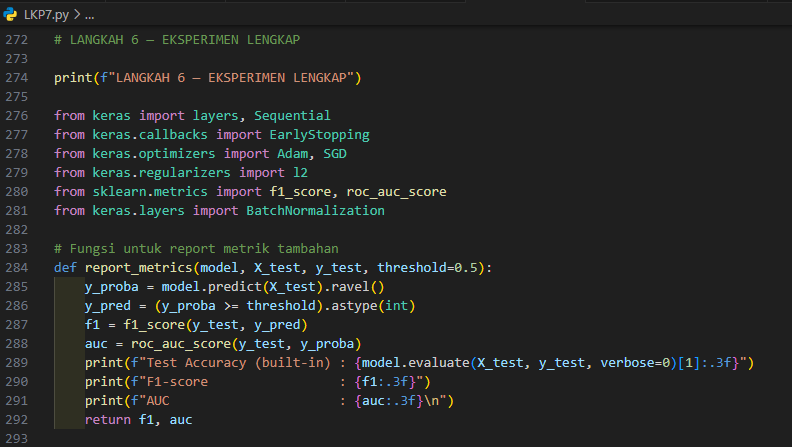
#### ****Rangkuman Keseluruhan Langkah 1–4****

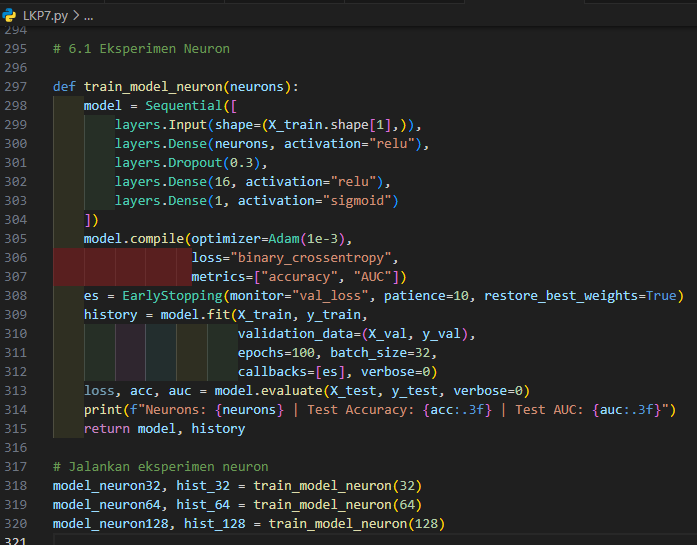
Secara umum, dari langkah 1 hingga 4, proses yang dilakukan meliputi:

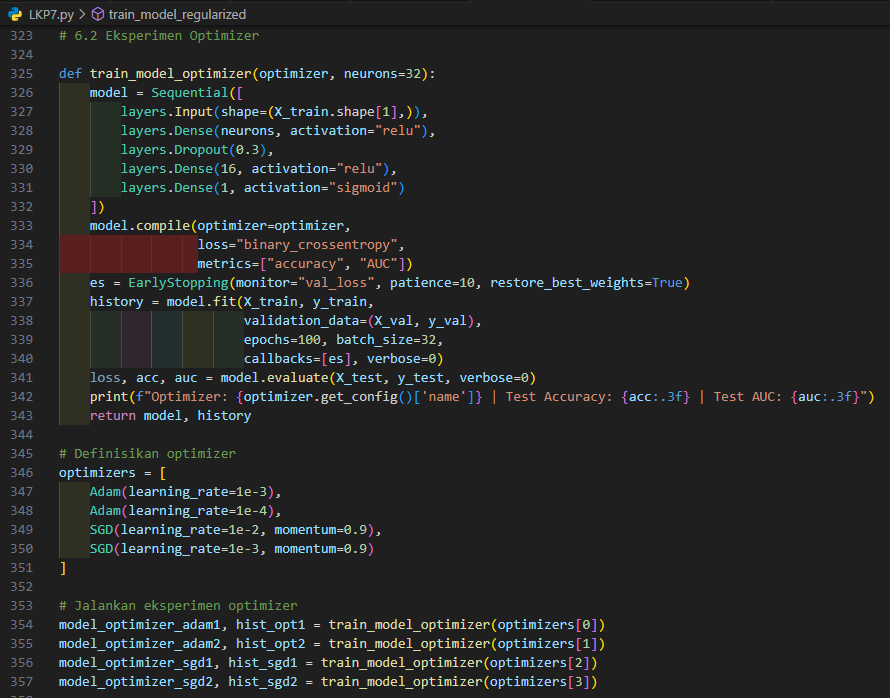
1. **Persiapan data** (pembacaan, standarisasi, dan pembagian dataset),
2. **Pembangunan model ANN** dengan arsitektur dasar,
3. **Pelatihan model** menggunakan Early Stopping, dan
4. **Evaluasi performa** dengan berbagai metrik seperti akurasi, F1-score, dan AUC.

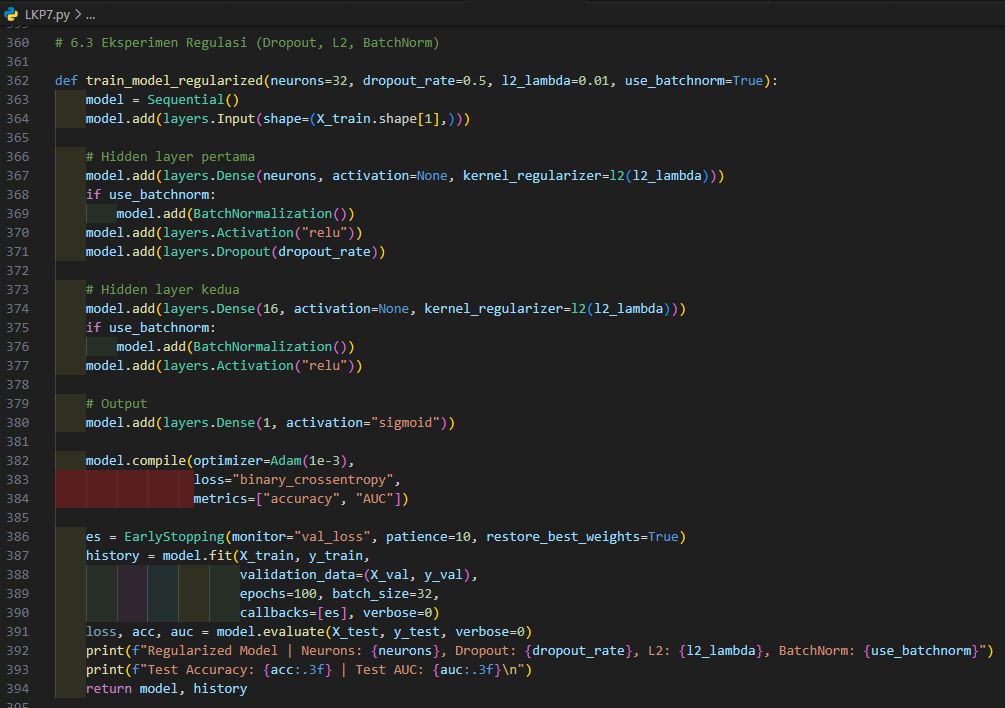
Langkah-langkah selanjutnya memperluas eksperimen untuk memperoleh konfigurasi model terbaik dengan mempertimbangkan keseimbangan antara akurasi dan kemampuan generalisasi.

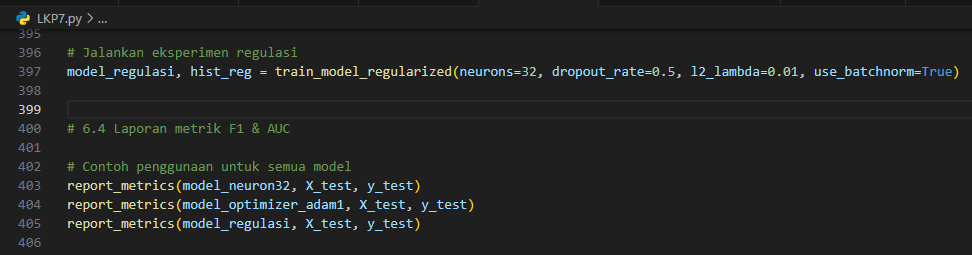
**Kode Program:**











**Output:**

