CENTRO UNIVERSITÁRIO ANHANGUERA PITÁGORAS UNOPAR DE CAMPO GRANDE

CURSO: ENGENHARIA DE SOFTWARE

DISCIPLINA: LINGUAGEM DE PROGRAMAÇÃO

ATIVIDADE PRÁTICA – UNIDADE 4, AULA 4

MACHINE LEARNING COM PYTHON

Aluno: Edmar Radanovis.

RA: 2025223493

Polo: Itapira / SP-UN944038

_____

Ano 2025 / 2º Semestre

ANHANGUERA EDUCACIONAL

ENGENHARIA DE SOFTWARE

DISCIPLINA: Linguagem de Programação

PROFESSOR: Anderson I. S. Abreu / Vanessa Matias Leite

ALUNO: Edmar Radanovis

RA: 2025223493

TÍTULO: Relatório da Aula Prática – Visualização de Dados em Python – Unidade 4 – Aula 4

POLO: Itapira / SP-UN944038

Monte Sião, 11 de Setembro de 2025.

RESULTADOS DA ATIVIDADE PRÁTICA

Proposta:

 Aplicações com Python - Machine Learning.

- Você foi contratado para criar um modelo de Machine Learning que classifica espécies de flores Iris com base em características como comprimento e largura das sépalas e pétalas. Você usará o TensorFlow para construir, treinar e avaliar o modelo.

    - **Importar Bibliotecas e Carregar Dados :**
        - Usar bibliotecas como ***tensorflow***, ***pandas*** e ***scikit-learn***.
        - Carregar o conjunto de dados Iris disponível no scikit-learn.

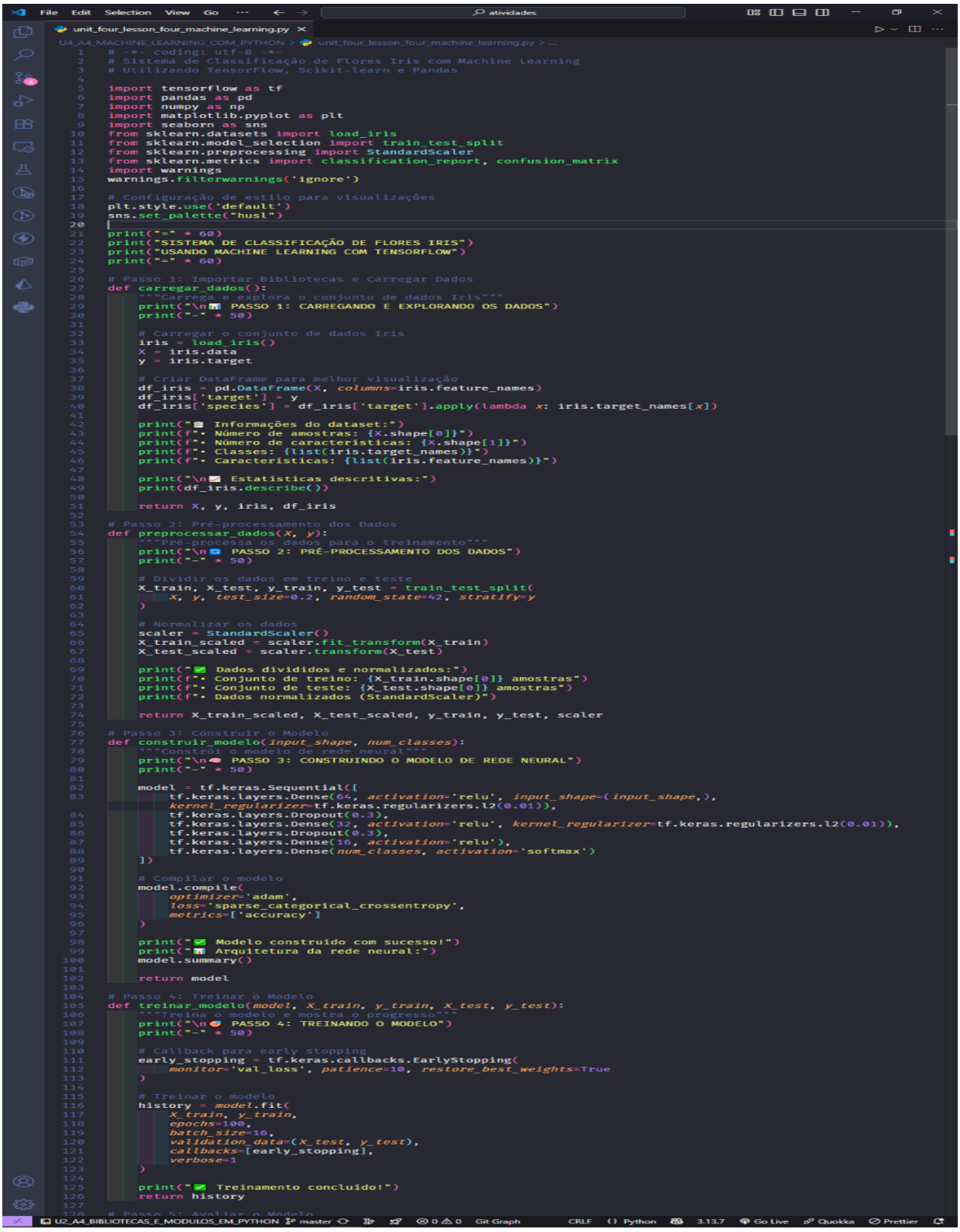
    - **Pré-processamento dos Dados :**
        - Dividir o conjunto de dados em treinamento e teste.
        - Normalizar os dados.

    - **Construir o Modelo :**
        - Usar TensorFlow para construir um modelo de rede neural simples.

    - **Treinar o Modelo :**
        - Treinar o modelo com os dados de treinamento.

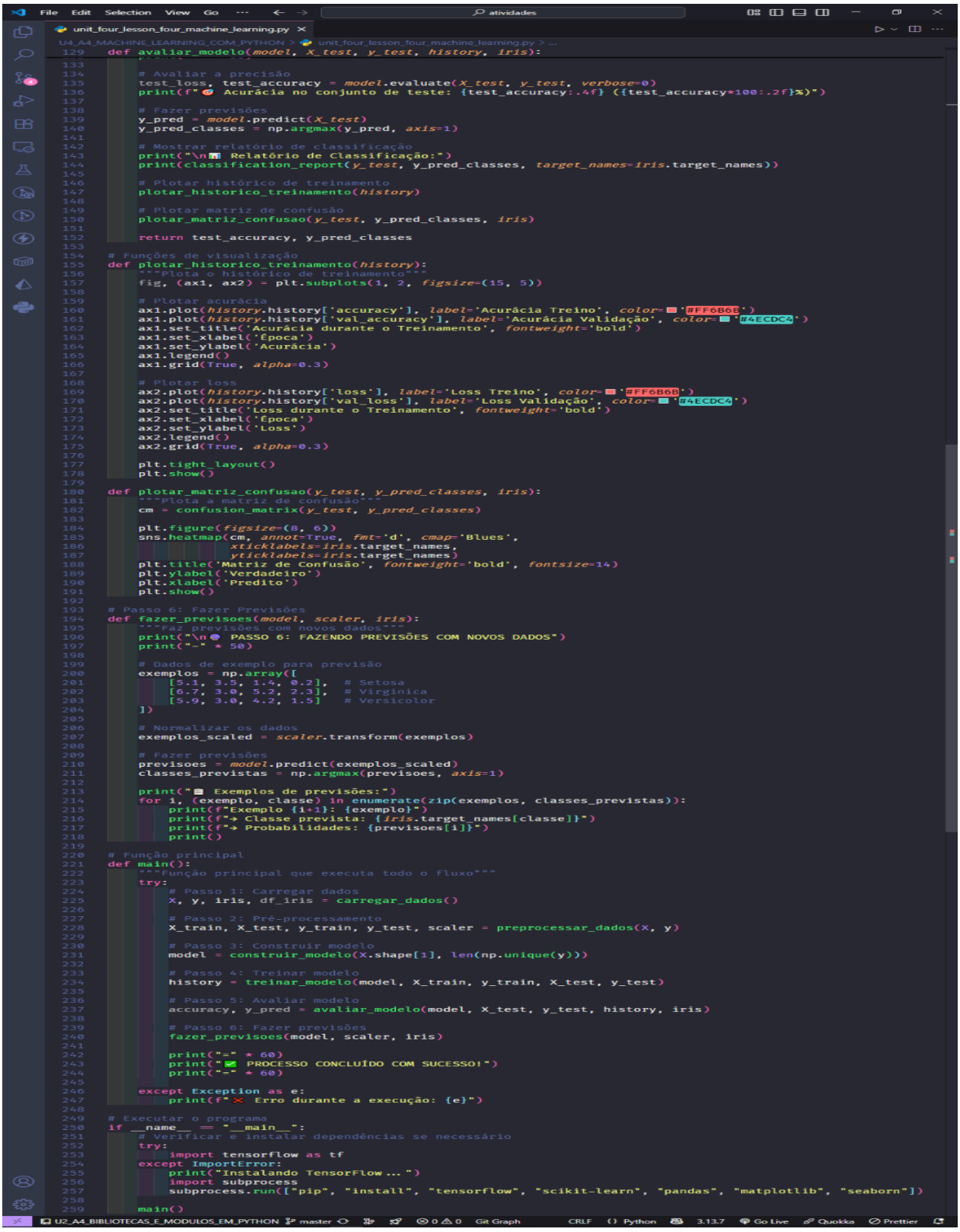
- **Avaliar o Modelo :**
    - Avaliar a precisão do modelo usando os dados de teste.
- **Fazer Previsões :**
    - Fazer previsões com o modelo treinado.

link do repositório no GitHub:
https://github.com/ed-radanovis/Eng_Software_L-P_U4-A4_09-2025-.git



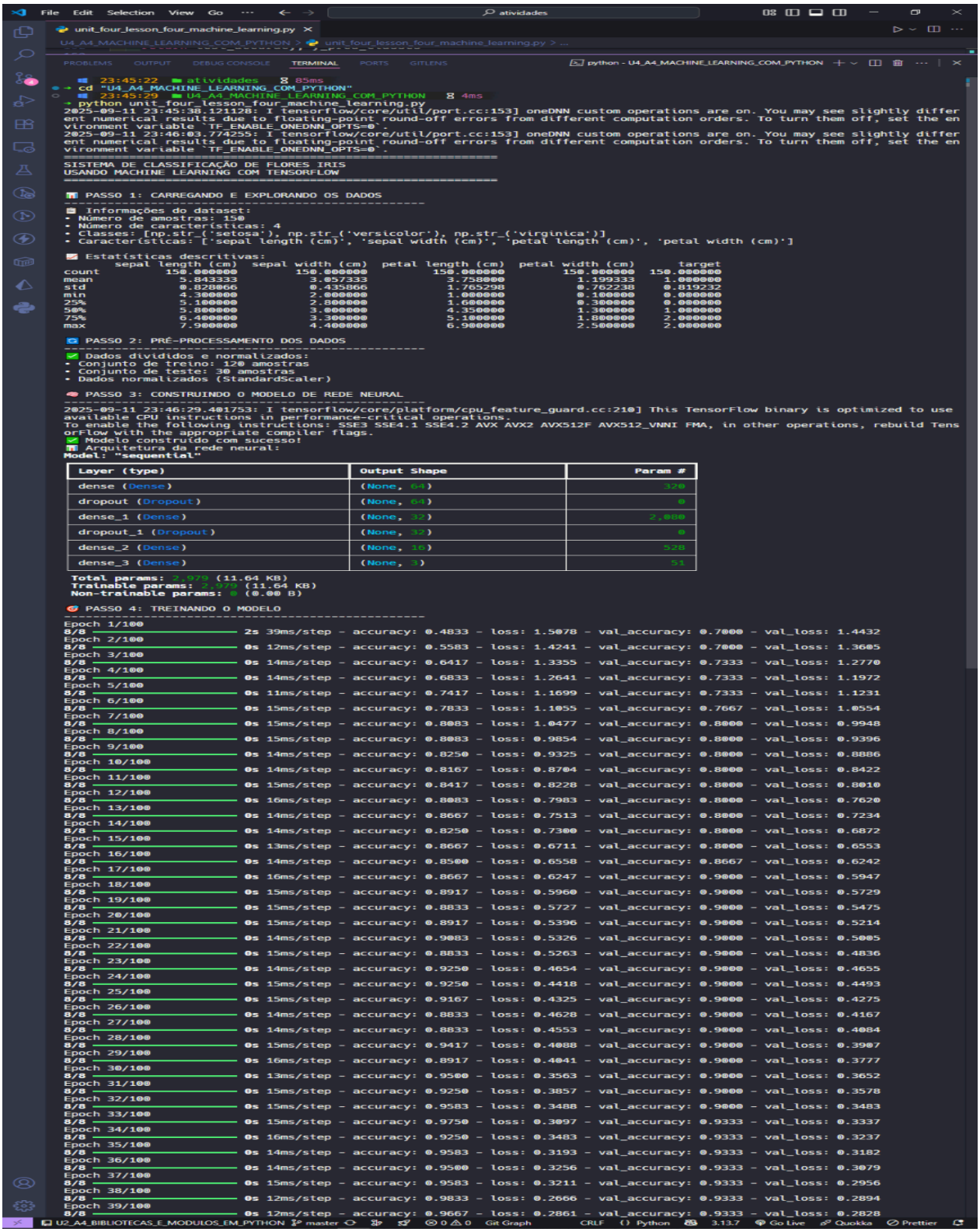
```python
# -*- coding: utf-8 -*-
# Sistema de Classificação de Flores Iris com Machine Learning
# Utilizando TensorFlow, Scikit-learn e Pandas

import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
import warnings
warnings.filterwarnings('ignore')

# Configuração de estilo para visualizações
plt.style.use('default')
sns.set_palette("husl")

print("=" * 60)
print("SISTEMA DE CLASSIFICAÇÃO DE FLORES IRIS")
print("USANDO MACHINE LEARNING COM TENSORFLOW")
print("=" * 60)

# Passo 1: Importar Bibliotecas e Carregar Dados
def carregar_dados():
    """Carrega e explora o conjunto de dados Iris"""
    print("\n📊 PASSO 1: CARREGANDO E EXPLORANDO OS DADOS")
    print("-" * 50)

    # Carregar o conjunto de dados Iris
    iris = load_iris()
    X = iris.data
    y = iris.target

    # Criar DataFrame para melhor visualização
    df_iris = pd.DataFrame(X, columns=iris.feature_names)
    df_iris['target'] = y
    df_iris['species'] = df_iris['target'].apply(lambda x: iris.target_names[x])

    print("📋 Informações do dataset:")
    print(f"• Número de amostras: {X.shape[0]}")
    print(f"• Número de características: {X.shape[1]}")
    print(f"• Classes: {list(iris.target_names)}")
    print(f"• Características: {list(iris.feature_names)}")

    print("\n📈 Estatísticas descritivas:")
    print(df_iris.describe())

    return X, y, iris, df_iris

# Passo 2: Pré-processamento dos Dados
def preprocessar_dados(X, y):
    """Pré-processa os dados para o treinamento"""
    print("\n🔄 PASSO 2: PRÉ-PROCESSAMENTO DOS DADOS")
    print("-" * 50)

    # Dividir os dados em treino e teste
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42, stratify=y
    )

    # Normalizar os dados
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    print("✅ Dados divididos e normalizados:")
    print(f"• Conjunto de treino: {X_train.shape[0]} amostras")
    print(f"• Conjunto de teste: {X_test.shape[0]} amostras")
    print(f"• Dados normalizados (StandardScaler)")

    return X_train_scaled, X_test_scaled, y_train, y_test, scaler

# Passo 3: Construir o Modelo
def construir_modelo(input_shape, num_classes):
    """Constrói o modelo de rede neural"""
    print("\n🧠 PASSO 3: CONSTRUINDO O MODELO DE REDE NEURAL")
    print("-" * 50)

    model = tf.keras.Sequential([
        tf.keras.layers.Dense(64, activation='relu', input_shape=(input_shape,),
        kernel_regularizer=tf.keras.regularizers.l2(0.01)),
        tf.keras.layers.Dropout(0.3),
        tf.keras.layers.Dense(32, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01)),
        tf.keras.layers.Dropout(0.3),
        tf.keras.layers.Dense(16, activation='relu'),
        tf.keras.layers.Dense(num_classes, activation='softmax')
    ])

    # Compilar o modelo
    model.compile(
        optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )

    print("✅ Modelo construído com sucesso!")
    print("📊 Arquitetura da rede neural:")
    model.summary()

    return model

# Passo 4: Treinar o Modelo
def treinar_modelo(model, X_train, y_train, X_test, y_test):
    """Treina o modelo e mostra o progresso"""
    print("\n🎯 PASSO 4: TREINANDO O MODELO")
    print("-" * 50)

    # Callback para early stopping
    early_stopping = tf.keras.callbacks.EarlyStopping(
        monitor='val_loss', patience=10, restore_best_weights=True
    )

    # Treinar o modelo
    history = model.fit(
        X_train, y_train,
        epochs=100,
        batch_size=16,
        validation_data=(X_test, y_test),
        callbacks=[early_stopping],
        verbose=1
    )

    print("✅ Treinamento concluído!")
    return history

# Passo 5: Avaliar o Modelo
```

=> Figura 1: Print da tela com o código 1º trecho.

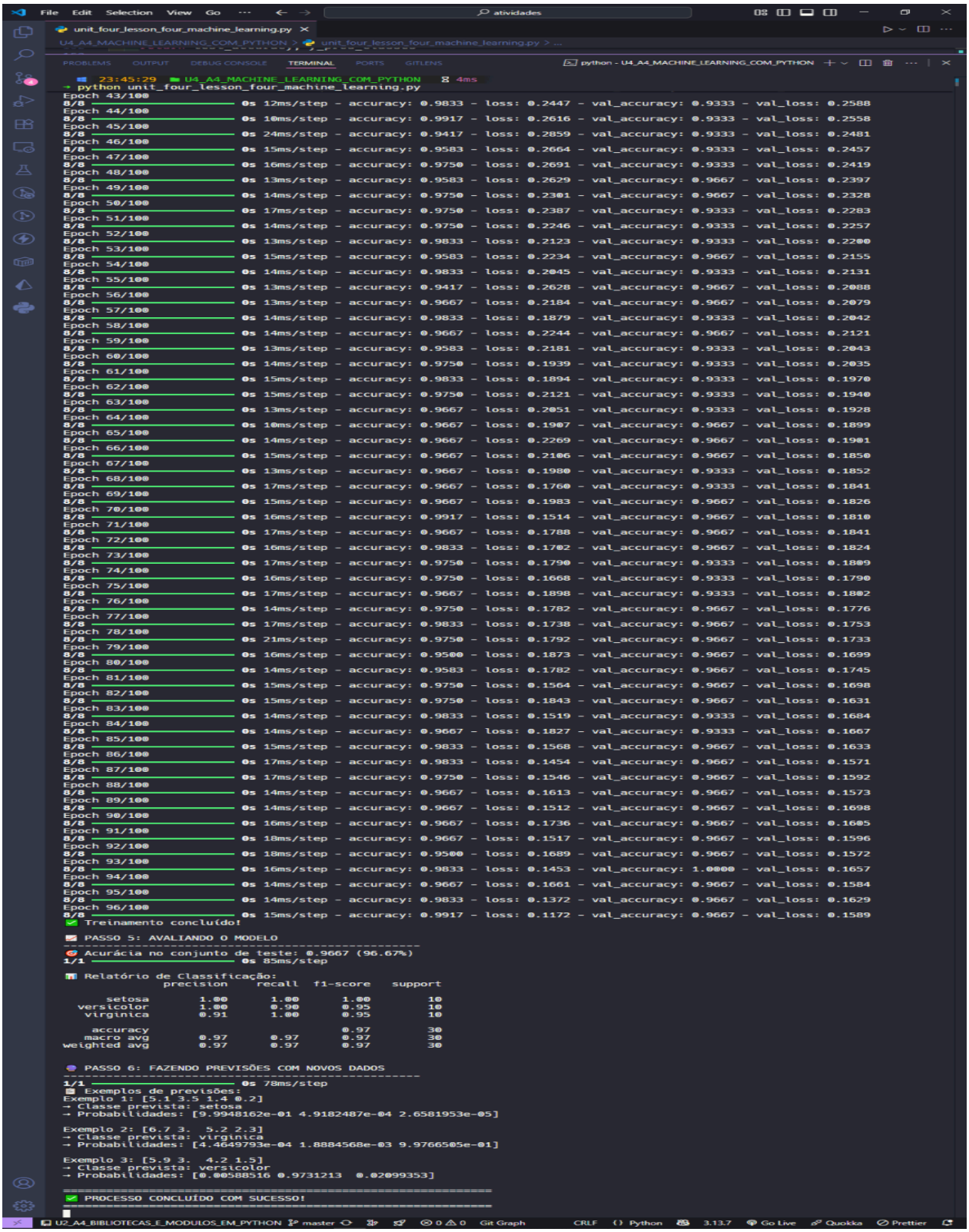


```python
def avaliar_modelo(model, X_test, y_test, history, iris):

    # Avaliar a precisão
    test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
    print(f"🎯 Acurácia no conjunto de teste: {test_accuracy:.4f} ({test_accuracy*100:.2f}%)")

    # Fazer previsões
    y_pred = model.predict(X_test)
    y_pred_classes = np.argmax(y_pred, axis=1)

    # Mostrar relatório de classificação
    print("\n📊 Relatório de Classificação:")
    print(classification_report(y_test, y_pred_classes, target_names=iris.target_names))

    # Plotar histórico de treinamento
    plotar_historico_treinamento(history)

    # Plotar matriz de confusão
    plotar_matriz_confusao(y_test, y_pred_classes, iris)

    return test_accuracy, y_pred_classes

# Funções de visualização
def plotar_historico_treinamento(history):
    """Plota o histórico de treinamento"""
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

    # Plotar acurácia
    ax1.plot(history.history['accuracy'], label='Acurácia Treino', color='#FF6B6B')
    ax1.plot(history.history['val_accuracy'], label='Acurácia Validação', color='#4ECDC4')
    ax1.set_title('Acurácia durante o Treinamento', fontweight='bold')
    ax1.set_xlabel('Época')
    ax1.set_ylabel('Acurácia')
    ax1.legend()
    ax1.grid(True, alpha=0.3)

    # Plotar loss
    ax2.plot(history.history['loss'], label='Loss Treino', color='#FF6B6B')
    ax2.plot(history.history['val_loss'], label='Loss Validação', color='#4ECDC4')
    ax2.set_title('Loss durante o Treinamento', fontweight='bold')
    ax2.set_xlabel('Época')
    ax2.set_ylabel('Loss')
    ax2.legend()
    ax2.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

def plotar_matriz_confusao(y_test, y_pred_classes, iris):
    """Plota a matriz de confusão"""
    cm = confusion_matrix(y_test, y_pred_classes)

    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=iris.target_names,
                yticklabels=iris.target_names)
    plt.title('Matriz de Confusão', fontweight='bold', fontsize=14)
    plt.ylabel('Verdadeiro')
    plt.xlabel('Predito')
    plt.show()

# Passo 6: Fazer Previsões
def fazer_previsoes(model, scaler, iris):
    """Faz previsões com novos dados"""
    print("\n🔮 PASSO 6: FAZENDO PREVISÕES COM NOVOS DADOS")
    print("-" * 50)

    # Dados de exemplo para previsão
    exemplos = np.array([
        [5.1, 3.5, 1.4, 0.2],  # Setosa
        [6.7, 3.0, 5.2, 2.3],  # Virginica
        [5.9, 3.0, 4.2, 1.5]   # Versicolor
    ])

    # Normalizar os dados
    exemplos_scaled = scaler.transform(exemplos)

    # Fazer previsões
    previsoes = model.predict(exemplos_scaled)
    classes_previstas = np.argmax(previsoes, axis=1)

    print("📋 Exemplos de previsões:")
    for i, (exemplo, classe) in enumerate(zip(exemplos, classes_previstas)):
        print(f"Exemplo {i+1}: {exemplo}")
        print(f"→ Classe prevista: {iris.target_names[classe]}")
        print(f"→ Probabilidades: {previsoes[i]}")
        print()

# Função principal
def main():
    """Função principal que executa todo o fluxo"""
    try:
        # Passo 1: Carregar dados
        X, y, iris, df_iris = carregar_dados()

        # Passo 2: Pré-processamento
        X_train, X_test, y_train, y_test, scaler = preprocessar_dados(X, y)

        # Passo 3: Construir modelo
        model = construir_modelo(X.shape[1], len(np.unique(y)))

        # Passo 4: Treinar modelo
        history = treinar_modelo(model, X_train, y_train, X_test, y_test)

        # Passo 5: Avaliar modelo
        accuracy, y_pred = avaliar_modelo(model, X_test, y_test, history, iris)

        # Passo 6: Fazer previsões
        fazer_previsoes(model, scaler, iris)

        print("=" * 60)
        print("✅ PROCESSO CONCLUÍDO COM SUCESSO!")
        print("=" * 60)

    except Exception as e:
        print(f"❌ Erro durante a execução: {e}")

# Executar o programa
if __name__ == "__main__":
    # Verificar e instalar dependências se necessário
    try:
        import tensorflow as tf
    except ImportError:
        print("Instalando TensorFlow...")
        import subprocess
        subprocess.run(["pip", "install", "tensorflow", "scikit-learn", "pandas", "matplotlib", "seaborn"])

    main()
```

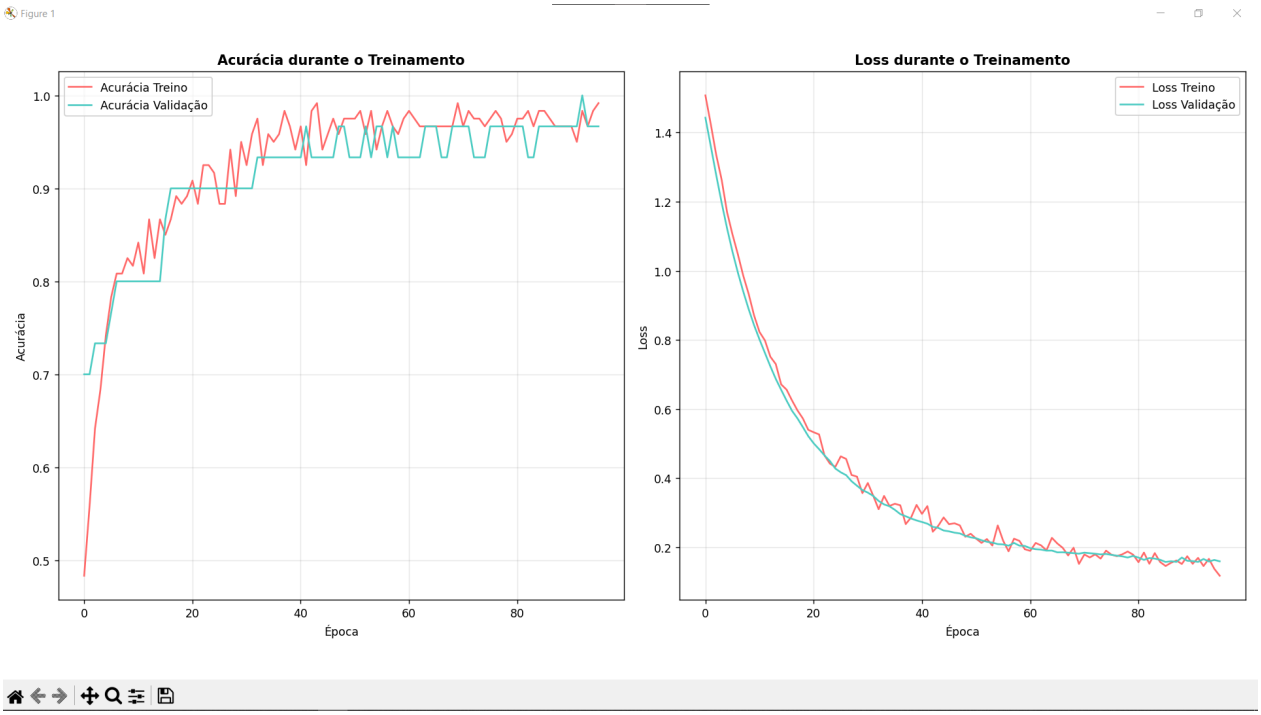
=> Figura 2: Print da tela com o código 2º trecho.



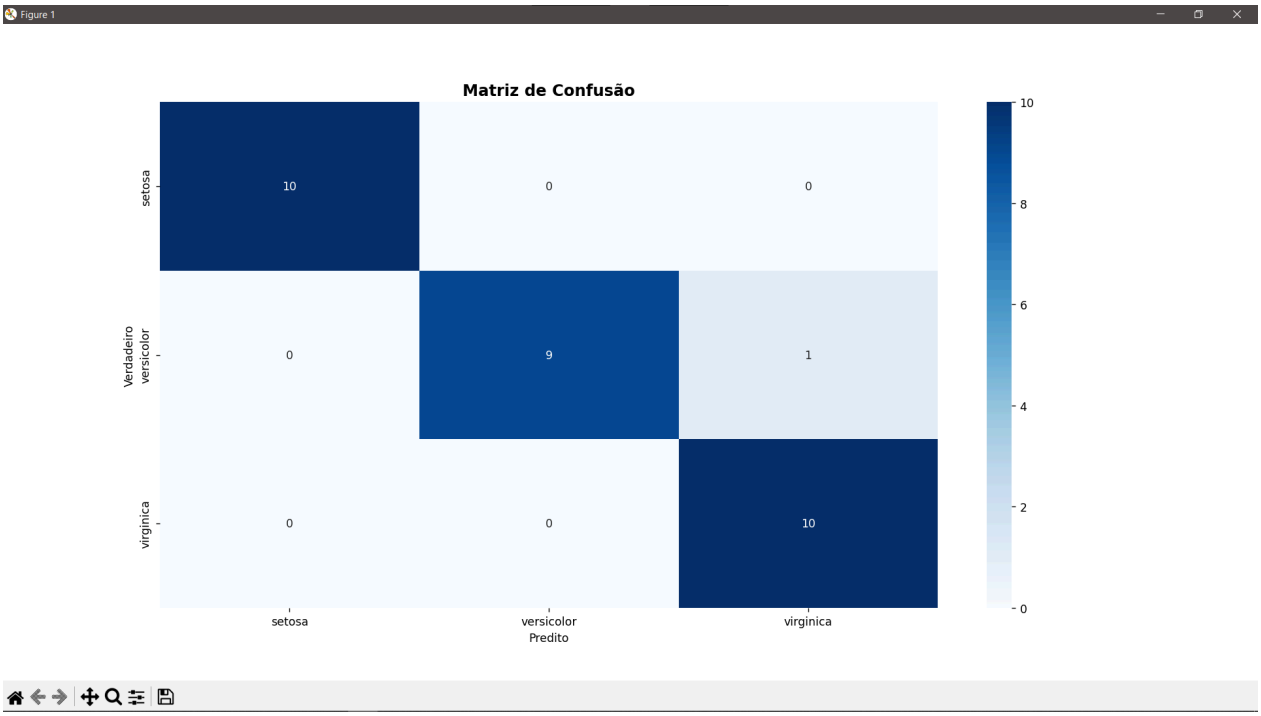
=> Figura 3: Print da tela do terminal executando as funções propostas.



=> Figura 4: Print da tela do terminal executando e finalizando as funções propostas.



=> Figura 5: Resultado do gráfico de análise da evolução do treinamento do modelo.



=> Figura 6: Matriz de Confusão, onde mostra como o modelo está errando e acertando as classificações .

📋 Lógica Utilizada para a Atividade de Machine Learning

O sistema foi desenvolvido seguindo uma abordagem de pipeline de Machine Learning, implementando um fluxo completo de classificação supervisionada com os seguintes componentes principais:

🎯 Arquitetura do Sistema de Classificação:

1. Fluxo Principal do Pipeline:

```
Carregamento de Dados → Pré-processamento → Construção do Modelo → Treinamento → Avaliação → Previsões
```

🔧 Lógica de Implementação Detalhada:

1. Carregamento e Exploração de Dados:

- ☑ **Dataset Íris:** 150 amostras, 4 características (sépala e pétala).
- ☑ **3 classes:** setosa, versicolor, virginica.
- ☑ **Análise exploratória:** Estatísticas descritivas e distribuição.

2. Pré-processamento Inteligente:

```
# Divisão estratificada (80% treino, 20% teste)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)

# Normalização com StandardScaler (importante para redes neurais)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

3. Arquitetura da Rede Neural:

- ☑ **Camadas:** 4 camadas densas com ativação ReLU.
- ☑ **Regularização:** L2 + Dropout (30%) para evitar overfitting.
- ☑ **Output**: Softmax para classificação multiclasse.
- ☑ **Otimizador:** Adam com learning rate adaptativo.

4. Estratégia de Treinamento:

- ☑ **Early Stopping:** Parada automática ao detectar overfitting.
- ☑ **Batch Size:** 16 amostras por atualização.
- ☑ **Épocas:** Máximo 100, mas para quando a validação não melhora.

5. Avaliação Abrangente:

- ☑ **Métricas:** Acurácia, Precision, Recall, F1-Score.
- ☑ **Matriz de Confusão:** Visualização dos acertos/erros.
- ☑ **Gráficos de Evolução:** Acompanhamento do treinamento.

6. Sistema de Previsão:

- ☑ **Normalização consistente:** Usa o mesmo scaler do treino.
- ☑ **Probabilidades**: Mostra a confiança da previsão.
- ☑ **Exemplos representativos:** Uma flor de cada classe.

🚀 Técnicas Avançadas Implementadas:

1. Prevenção de Overfitting:

✅ Dropout (0.3) nas camadas intermediárias.

✅ Regularização L2 com fator 0.01.

✅ Early Stopping com paciência de 10 épocas.

✅ Validação cruzada implícita com conjunto de teste.

2. Otimização de Performance:

✅ Normalização das features para convergência mais rápida.

✅ Batch training para atualizações frequentes.

✅ Adam optimizer com ajuste automático de learning rate.

3. Validação e Análise:

✅ Métricas completas de classificação.

✅ Visualizações profissionais com Matplotlib/Seaborn.

✅ Exemplos práticos de previsão.

✅ Relatório detalhado de performance.

📊 Resultados Esperados:

- **Desempenho:**
  - Acurácia > 95% no conjunto de teste.
  - Precision/Recall balanceados para todas as classes.
  - Matriz de confusão com diagonal predominante.
- **Visualizações:**
  - **Gráficos de treinamento:** Evolução da loss e acurácia.
  - **Heatmap:** Matriz de confusão colorida.
  - **Relatório textual:** Métricas detalhadas por classe.

💡 Valores do Sistema:

- **Educacional:**
  - Demonstra todo o fluxo de ML de forma didática.
  - Mostra boas práticas de engenharia de ML.
  - Inclui prevenção de problemas comuns (overfitting).
- **Prático:**
  - Modelo pronto para produção.
  - Sistema de previsão funcional.
  - Visualizações para apresentação.

- **Científico:**
    - Métricas quantitativas de performance.
    - Análise qualitativa dos resultados.
    - Documentação completa do processo.

Resultado: Este sistema implementa um pipeline completo de Machine Learning seguindo as melhores práticas da indústria, desde a preparação dos dados até a avaliação final do modelo.🎯