**Nama : Hero Kartiko**

**NIM : 1103210205**

**Kelas : TK-45-G04**

## TUGAS WEEK 11 DUMMY DATASET

```
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import StandardScaler
import pandas as pd
```

```
# Generate dummy data
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

**Analisis:**

kode digunakan untuk mempersiapkan data klasifikasi dengan menggunakan dummy dataset menggunaakn make_classificaton, membuat standar fitur dengan StandardScaler, dan membagi data set menjadi train dan test set menggunakan train_test_split. Kode juga mengimpor library yang akan mendukung pipeline machine learning dan deep learning termasuk PyTorch untuk membangun mode neural network serta fungsi seperti scikit-learn untuk preprocesssing, evaluasi, dan manipulasi data.

```python
# Convert data to PyTorch tensors
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32).unsqueeze(1)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test, dtype=torch.float32).unsqueeze(1)
```

**Analisis:**
kode ini akan mengkonversi data pelatihan dan pengujian ke dalam format tensor PyTorch yang diperlukan untuk melatih model deep learning. Fitur pelatihan (X_train) dan pengujian (X_test) dikonversi menjadi tensor dengan tipe data float32, yang sering digunakan dalam komputasi berbasis PyTorch, untuk label (y_train dan y_test) juga dikonversi menjadi tensor dengan operasi unsqueeze(1) untuk menambah dimensi baru, mengubah bentuk vektor 1D menjadi 2D dengan dimensi [n_sample, 1].

```python
# Define MLP model class
class MLPModel(nn.Module):
    def __init__(self, input_size, hidden_layers, activation_fn):
        super(MLPModel, self).__init__()
        layers = []
        for neurons in hidden_layers:
            layers.append(nn.Linear(input_size, neurons))
            layers.append(activation_fn())
            input_size = neurons
        layers.append(nn.Linear(input_size, 1))  # Output layer
        layers.append(nn.Sigmoid())  # Sigmoid for binary classification
        self.model = nn.Sequential(*layers)

    def forward(self, x):
        return self.model(x)

# Define hyperparameters
hidden_layer_options = [
    [4], [8], [16], [32], [64],
    [4, 16], [16, 32], [32, 64],
    [4, 16, 32], [16, 32, 64]]
activation_functions = [nn.Identity, nn.Sigmoid, nn.ReLU, nn.Tanh]
epoch_options = [1, 10, 25, 50, 100, 250]
learning_rate_options = [10, 1, 0.1, 0.01, 0.001, 0.0001]
batch_size_options = [16, 32, 64, 128, 256, 512]
```

**Analisis:**
Kode ini mendefinisikan arsitektur model Multilayer Perceptron (MLP) menggunakan PyTorch dengan kelas MLPModel, yang dapat dikustomisasi berdasarkan jumlah lapisan tersembunyi, ukuran input, dan fungsi aktivasi. Kelas ini membangun jaringan dengan menambahkan lapisan linear dan fungsi aktivasi untuk setiap neuron di lapisan tersembunyi, serta menggunakan fungsi aktivasi Sigmoid di lapisan output untuk klasifikasi biner. Selain itu, kode ini juga menyediakan berbagai opsi hiperparameter seperti ukuran lapisan tersembunyi, fungsi aktivasi, jumlah epoch, laju pembelajaran, dan ukuran batch, memungkinkan eksperimen yang fleksibel untuk menemukan konfigurasi terbaik.

```python
# Train and evaluate model for each combination
results = []

for hidden_layers in hidden_layer_options:
    for activation_fn in activation_functions:
        for epochs in epoch_options:
            for lr in learning_rate_options:
                for batch_size in batch_size_options:
                    # Initialize model, loss function, and optimizer
                    model = MLPModel(X_train.shape[1], hidden_layers, activation_fn)
                    criterion = nn.BCELoss()
                    optimizer = optim.Adam(model.parameters(), lr=lr)

                    # Training loop
                    model.train()
                    for epoch in range(epochs):
                        for i in range(0, len(X_train_tensor), batch_size):
                            X_batch = X_train_tensor[i:i+batch_size]
                            y_batch = y_train_tensor[i:i+batch_size]

                            optimizer.zero_grad()
                            outputs = model(X_batch)
                            loss = criterion(outputs, y_batch)
                            loss.backward()
                            optimizer.step()

                    # Evaluate the model
                    model.eval()
                    with torch.no_grad():
                        predictions = model(X_test_tensor).round()
                        accuracy = accuracy_score(y_test, predictions.numpy())
                        mae = mean_absolute_error(y_test, predictions.numpy())
                        mse = mean_squared_error(y_test, predictions.numpy())
                        r2 = r2_score(y_test, predictions.numpy())
```

**Analisis:**

Kode ini merupakan implementasi pipeline pelatihan dan evaluasi model Multilayer Perceptron (MLP) dengan eksplorasi kombinasi hiperparameter. Untuk setiap kombinasi dari jumlah lapisan tersembunyi, fungsi aktivasi, jumlah epoch, laju pembelajaran, dan ukuran batch, model dilatih menggunakan fungsi loss BCELoss (Binary Cross-Entropy Loss) untuk klasifikasi biner dan optimizer Adam untuk pembaruan bobot. Proses pelatihan dilakukan secara iteratif menggunakan mini-batch gradient descent, di mana setiap batch diproses dengan forward pass, kalkulasi loss, backward pass, dan pembaruan bobot. Setelah pelatihan, model dievaluasi pada data pengujian (X_test_tensor) dengan metrik seperti akurasi, Mean Absolute Error (MAE), Mean Squared Error (MSE), dan skor $R^2$.

```python
            # Evaluate the model
            model.eval()
            with torch.no_grad():
                predictions = model(X_test_tensor).round()
                accuracy = accuracy_score(y_test, predictions.numpy())
                mae = mean_absolute_error(y_test, predictions.numpy())
                mse = mean_squared_error(y_test, predictions.numpy())
                r2 = r2_score(y_test, predictions.numpy())

            # Store the results
            results.append({
                'Hidden_Layers': hidden_layers,
                'Activation': activation_fn.__name__,
                'Epochs': epochs,
                'Learning_Rate': lr,
                'Batch_Size': batch_size,
                'Loss': loss.item(),
                'Accuracy': accuracy,
                'MAE': mae,
                'MSE': mse,
                'R2': r2
            })

# Convert results to DataFrame and save
results_df = pd.DataFrame(results)
results_df = results_df.sort_values(by='Accuracy', ascending=False)
results_df.to_csv('MLP_Dummy_Classification_Results.csv', index=False)

# Print top 20 results in tabulated format
print(tabulate(results_df.head(20), headers='keys', tablefmt='grid'))
```

**Output :**

| | Hidden_Layers | Activation | Epochs | Learning_Rate | Batch_Size | Loss | Accuracy | MAE | MSE | R |
|---|---|---|---|---|---|---|---|---|---|---|
| 4917 | [4, 16] | ReLU | 100 | 0.01 | 128 | 0.0425412 | 0.9 | 0.1 | 0.1 | 0.59803 |
| 8126 | [16, 32, 64] | Sigmoid | 50 | 0.001 | 64 | 0.174808 | 0.895 | 0.105 | 0.105 | 0.57793 |
| 5102 | [4, 16] | Tanh | 50 | 0.001 | 64 | 0.175496 | 0.895 | 0.105 | 0.105 | 0.57793 |
| 5966 | [16, 32] | Tanh | 50 | 0.001 | 64 | 0.157296 | 0.89 | 0.11 | 0.11 | 0.55783 |
| 4886 | [4, 16] | ReLU | 50 | 0.001 | 64 | 0.168844 | 0.89 | 0.11 | 0.11 | 0.55783 |
| 4706 | [4, 16] | Sigmoid | 100 | 0.001 | 64 | 0.184997 | 0.89 | 0.11 | 0.11 | 0.55783 |
| 4592 | [4, 16] | Sigmoid | 10 | 0.01 | 64 | 0.177109 | 0.885 | 0.115 | 0.115 | 0.53773 |
| 3720 | [64] | Sigmoid | 10 | 0.1 | 16 | 0.066058 | 0.885 | 0.115 | 0.115 | 0.53773 |
| 7393 | [4, 16, 32] | ReLU | 10 | 0.1 | 32 | 0.175292 | 0.885 | 0.115 | 0.115 | 0.53773 |
| 5714 | [16, 32] | ReLU | 25 | 0.001 | 64 | 0.160902 | 0.885 | 0.115 | 0.115 | 0.53773 |
| 4705 | [4, 16] | Sigmoid | 100 | 0.001 | 32 | 0.19004 | 0.885 | 0.115 | 0.115 | 0.53773 |
| 7731 | [4, 16, 32] | Tanh | 100 | 0.001 | 128 | 0.151736 | 0.885 | 0.115 | 0.115 | 0.53773 |
| 4628 | [4, 16] | Sigmoid | 25 | 0.01 | 64 | 0.158222 | 0.885 | 0.115 | 0.115 | 0.53773 |

**Analisis:**

Kode ini mengevaluasi performa model MLP pada data pengujian untuk setiap kombinasi hiperparameter, mencatat metrik evaluasi seperti akurasi, loss, Mean Absolute Error (MAE), Mean Squared Error (MSE), dan skor $R^2$. Setelah proses evaluasi, hasil disimpan dalam daftar results sebagai dictionary yang mencakup detail konfigurasi hiperparameter seperti jumlah lapisan tersembunyi, fungsi aktivasi, jumlah epoch, laju pembelajaran, dan ukuran batch. Data hasil ini kemudian dikonversi menjadi DataFrame menggunakan Pandas, diurutkan berdasarkan akurasi dalam urutan menurun, dan disimpan dalam file CSV. Hasil terbaik terdapat pada kombinasi hidden layers [4, 16], fungsi aktivasi ReLU, 100 epoch, learning rate 0.01, dan batch size 128.

**TUGAS WEEK 11 HEART DISEASE DATASET**

```
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, LabelEncoder


df = pd.read_csv('HeartDiseaseTrain-Test.csv')
df.head()
```

**Output:**

| | age | sex | chest_pain_type | resting_blood_pressure | cholestoral | fasting_blood_sugar | rest_ecg | Max_heart_rate | exercise_induced_angina | oldpeak | slope | vessels_colored_by_flourosopy | thalassemia | targ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 52 | Male | Typical angina | 125 | 212 | Lower than 120 mg/ml | ST-T wave abnormality | 168 | No | 1.0 | Downsloping | Two | Reversable Defect | |
| 1 | 53 | Male | Typical angina | 140 | 203 | Greater than 120 mg/ml | Normal | 155 | Yes | 3.1 | Upsloping | Zero | Reversable Defect | |
| 2 | 70 | Male | Typical angina | 145 | 174 | Lower than 120 mg/ml | ST-T wave abnormality | 125 | Yes | 2.6 | Upsloping | Zero | Reversable Defect | |
| 3 | 61 | Male | Typical angina | 148 | 203 | Lower than 120 mg/ml | ST-T wave abnormality | 161 | No | 0.0 | Downsloping | One | Reversable Defect | |
| 4 | 62 | Female | Typical angina | 138 | 294 | Greater than 120 mg/ml | ST-T wave abnormality | 106 | No | 1.9 | Flat | Three | Fixed Defect | |

**Analisis:**

Dataset ini dibaca menggunakan pd.read_csv() dan ditampilkan dengan df.head(), yang menampilkan lima baris pertama dari dataset. Dataset ini berisi fitur seperti umur (age), jenis kelamin (sex), tipe nyeri dada (chest_pain_type), tekanan darah (resting_blood_pressure), kadar kolesterol (cholestoral), kadar gula darah (fasting_blood_sugar), serta variabel target (target) yang mungkin menunjukkan keberadaan penyakit jantung (1) atau tidak (0).

```
df.info()
```

```
df.describe
```

**Output:**

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   age                           1025 non-null   int64
 1   sex                           1025 non-null   object
 2   chest_pain_type               1025 non-null   object
 3   resting_blood_pressure        1025 non-null   int64
 4   cholestoral                   1025 non-null   int64
 5   fasting_blood_sugar           1025 non-null   object
 6   rest_ecg                      1025 non-null   object
 7   Max_heart_rate                1025 non-null   int64
 8   exercise_induced_angina       1025 non-null   object
 9   oldpeak                       1025 non-null   float64
 10  slope                         1025 non-null   object
 11  vessels_colored_by_flourosopy 1025 non-null   object
 12  thalassemia                   1025 non-null   object
 13  target                        1025 non-null   int64
dtypes: float64(1), int64(5), object(8)
memory usage: 112.2+ KB
```

```
<bound method NDFrame.describe of       age     sex  chest_pain_type  resting_blood_pressure  cholestoral  \
0      52    Male   Typical angina                   125          212
1      53    Male   Typical angina                   140          203
2      70    Male   Typical angina                   145          174
3      61    Male   Typical angina                   148          203
4      62  Female   Typical angina                   138          294
...   ...     ...              ...                   ...          ...
1020   59    Male   Atypical angina                  140          221
1021   60    Male   Typical angina                   125          258
1022   47    Male   Typical angina                   110          275
1023   50  Female   Typical angina                   110          254
1024   54    Male   Typical angina                   120          188

          fasting_blood_sugar              rest_ecg  Max_heart_rate  \
0      Lower than 120 mg/ml  ST-T wave abnormality             168
1     Greater than 120 mg/ml                 Normal             155
2      Lower than 120 mg/ml  ST-T wave abnormality             125
3      Lower than 120 mg/ml  ST-T wave abnormality             161
4     Greater than 120 mg/ml  ST-T wave abnormality             106
...                      ...                   ...             ...
1020   Lower than 120 mg/ml  ST-T wave abnormality             164
1021   Lower than 120 mg/ml                 Normal             141
1022   Lower than 120 mg/ml                 Normal             118
1023   Lower than 120 mg/ml                 Normal             159
1024   Lower than 120 mg/ml  ST-T wave abnormality             113

                        ...                   ...  ...
1022                    One       Fixed Defect    0
1023                   Zero       Fixed Defect    1
1024                    One  Reversable Defect    0

[1025 rows x 14 columns]>
```

**Analisis:**

Kode ini menggunakan fungsi df.info() dan df.describe() untuk melakukan eksplorasi data awal pada dataset *HeartDiseaseTrain-Test.csv*. Fungsi df.info() memberikan informasi mengenai dataset, termasuk jumlah total entri (1.025), jumlah nilai non-null pada setiap kolom, tipe data dari masing-masing kolom, serta penggunaan memori. Dari hasil ini, terlihat bahwa dataset memiliki 14 kolom dengan berbagai tipe data, seperti object untuk data kategorikal (contoh: sex, chest_pain_type) dan int64 atau float64 untuk data numerik (contoh: age, resting_blood_pressure). Sementara itu, fungsi df.describe() memberikan statistik deskriptif untuk kolom numerik, seperti rata-rata, nilai maksimum, dan minimum. Statistik ini membantu mengidentifikasi rentang nilai, distribusi data, dan kemungkinan adanya outlier.

```
# Step 1: Check for missing values
df.isnull().sum()
```

**Output:**

```
age                             0
sex                             0
chest_pain_type                 0
resting_blood_pressure          0
cholestoral                     0
fasting_blood_sugar             0
rest_ecg                        0
Max_heart_rate                  0
exercise_induced_angina         0
oldpeak                         0
slope                           0
vessels_colored_by_flourosopy   0
thalassemia                     0
target                          0
dtype: int64
```

**Analisis:**

df.isnull.sum() digunakan untuk memerika apakah terdapat missing value pada dataset. Hasil menunjukkan semua kolom dalam dataset memilki nilai untuk jumlah nilai yang hilang.

```python
numerical_columns = df.select_dtypes(include=['int64', 'float64']).columns
categorical_columns = df.select_dtypes(include=['object']).columns
label_encoders = {}

for col in categorical_columns:
    label_encoders[col] = LabelEncoder()
    df[col] = label_encoders[col].fit_transform(df[col])

scaler = StandardScaler()
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])

# Step 3: Check cleaned data
df.head()
```

**Output:**

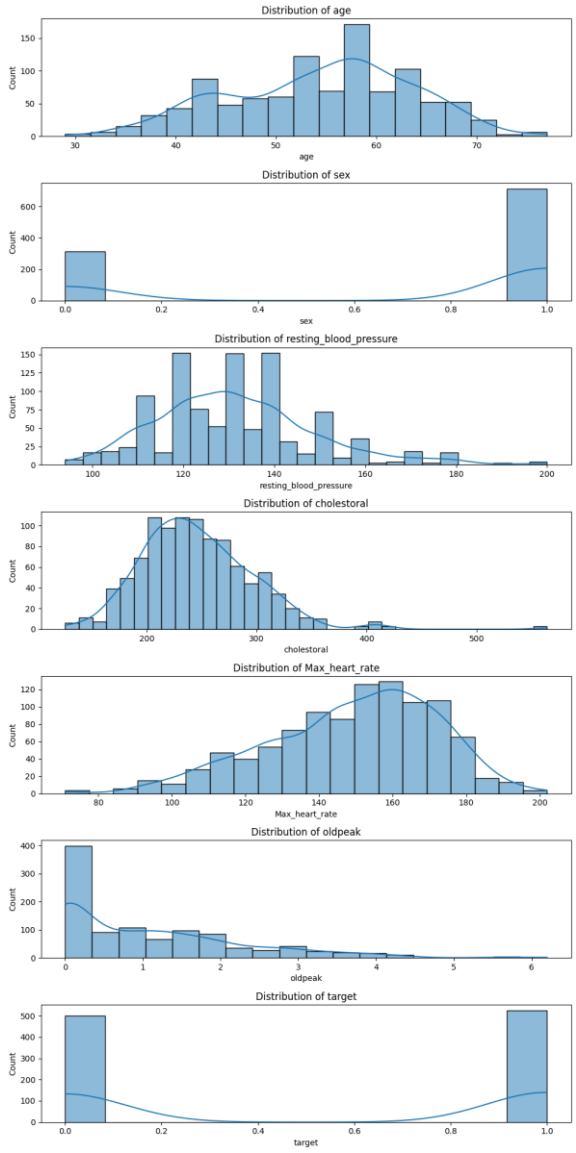
| | age | sex | chest_pain_type | resting_blood_pressure | cholestoral | fasting_blood_sugar | rest_ecg | Max_heart_rate | exercise_induced_angina | oldpeak | slope | vessels_colored_by_flourosopy | thalassemia | targ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | -0.268437 | 0.661504 | 3 | -0.377636 | -0.659332 | 1 | 2 | 0.821321 | 0 | -0.060888 | 0 | 3 | 3 | -1.02669 |
| 1 | -0.158157 | 0.661504 | 3 | 0.479107 | -0.833861 | 0 | 1 | 0.255968 | 1 | 1.727137 | 2 | 4 | 3 | -1.02669 |
| 2 | 1.716595 | 0.661504 | 3 | 0.764688 | -1.396233 | 1 | 2 | -1.048692 | 1 | 1.301417 | 2 | 4 | 3 | -1.02669 |
| 3 | 0.724079 | 0.661504 | 3 | 0.936037 | -0.833861 | 1 | 2 | 0.516900 | 0 | -0.912329 | 0 | 1 | 3 | -1.02669 |
| 4 | 0.834359 | -1.511706 | 3 | 0.364875 | 0.930822 | 0 | 2 | -1.874977 | 0 | 0.705408 | 1 | 2 | 0 | -1.02669 |

**Analisis:**

Kode ini melakukan preprocessing pada dataset dengan dua langkah utama: encoding data kategorikal dan standarisasi data numerik. Kolom dengan tipe data kategorikal diidentifikasi menggunakan df.select_dtypes dan diubah menjadi nilai numerik dengan LabelEncoder, sehingga dapat digunakan oleh algoritma machine learning. Kolom dengan nilai numerik diidentifikasi secara terpisah dan distandarisasi menggunakan StandardScaler untuk memastikan distribusi fitur memiliki rata-rata 0 dan standar deviasi 1. Proses ini membantu mengurangi skala yang tidak konsisten antara fitur dan meningkatkan performa model, terutama pada algoritma yang sensitif terhadap skala fitur seperti gradient descent.

```python
# Plotting distributions for numerical features
fig, axes = plt.subplots(len(numerical_columns), 1, figsize=(10, 20))
for i, col in enumerate(numerical_columns):
    sns.histplot(df[col], ax=axes[i], kde=True)
    axes[i].set_title(f'Distribution of {col}')

plt.tight_layout()
plt.show()
```

**Output :**

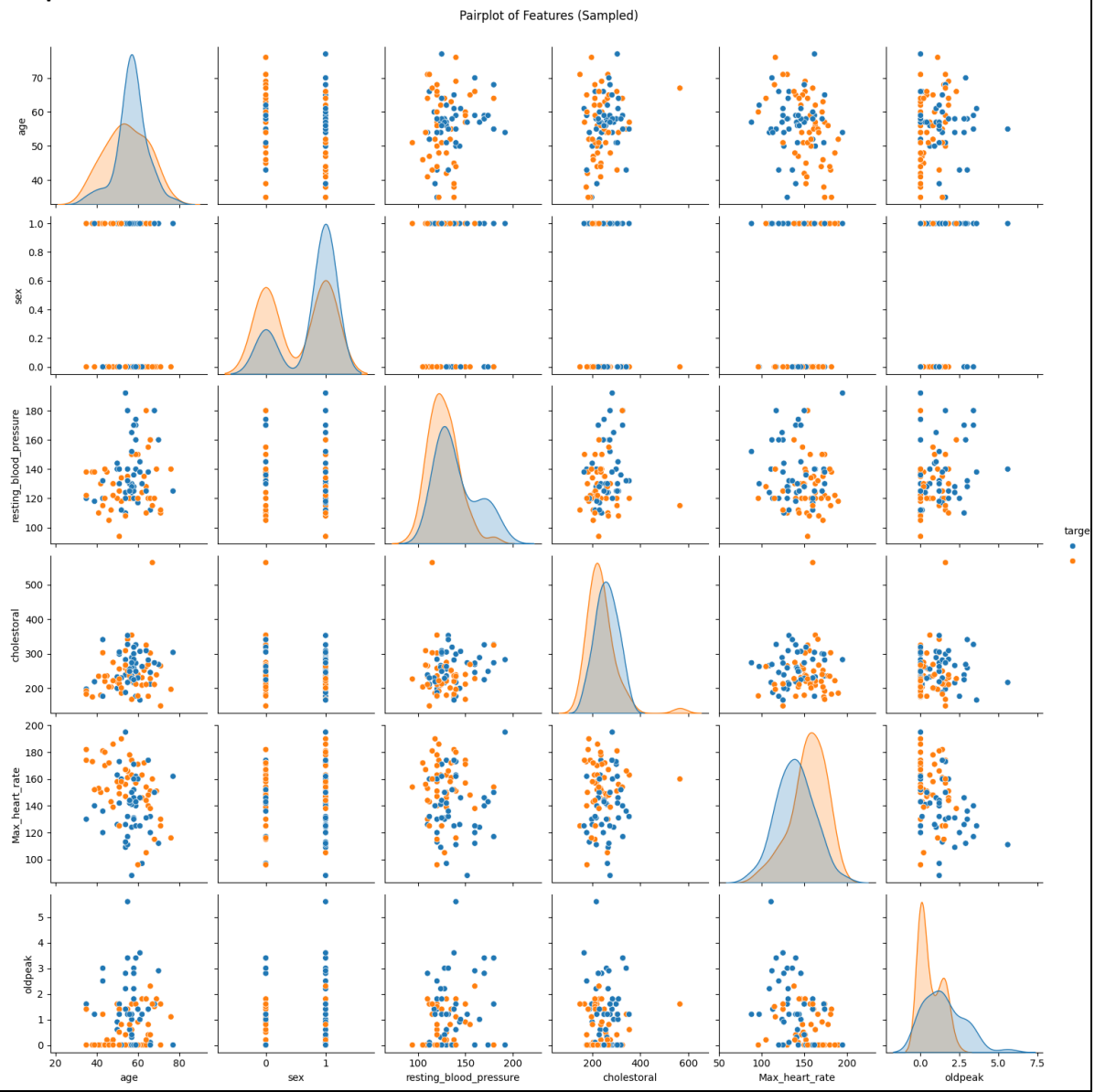



**Analisis:**

Kode ini bertujuan untuk memvisualisasikan distribusi fitur numerik dalam dataset menggunakan histogram dengan fungsi sns.histplot dari Seaborn. Setiap kolom numerik diplotkan dalam subplot terpisah menggunakan plt.subplots, dengan parameter kde=True untuk menampilkan kurva estimasi kepadatan kernel (KDE) yang memberikan gambaran halus distribusi data. Ukuran gambar diatur dengan figsize=(10, 20) untuk memastikan subplot dapat terlihat dengan jelas. Dari hasil visualisasi, distribusi data seperti age dan cholestoral cenderung mendekati distribusi normal, sementara fitur seperti oldpeak dan target menunjukkan distribusi yang tidak simetris.

```
# 3. Pairplot for relationships (sampled for efficiency)
sns.pairplot(df.sample(100), diag_kind='kde', hue='target')
plt.suptitle("Pairplot of Features (Sampled)", y=1.02)
plt.show()
```

**Output:**

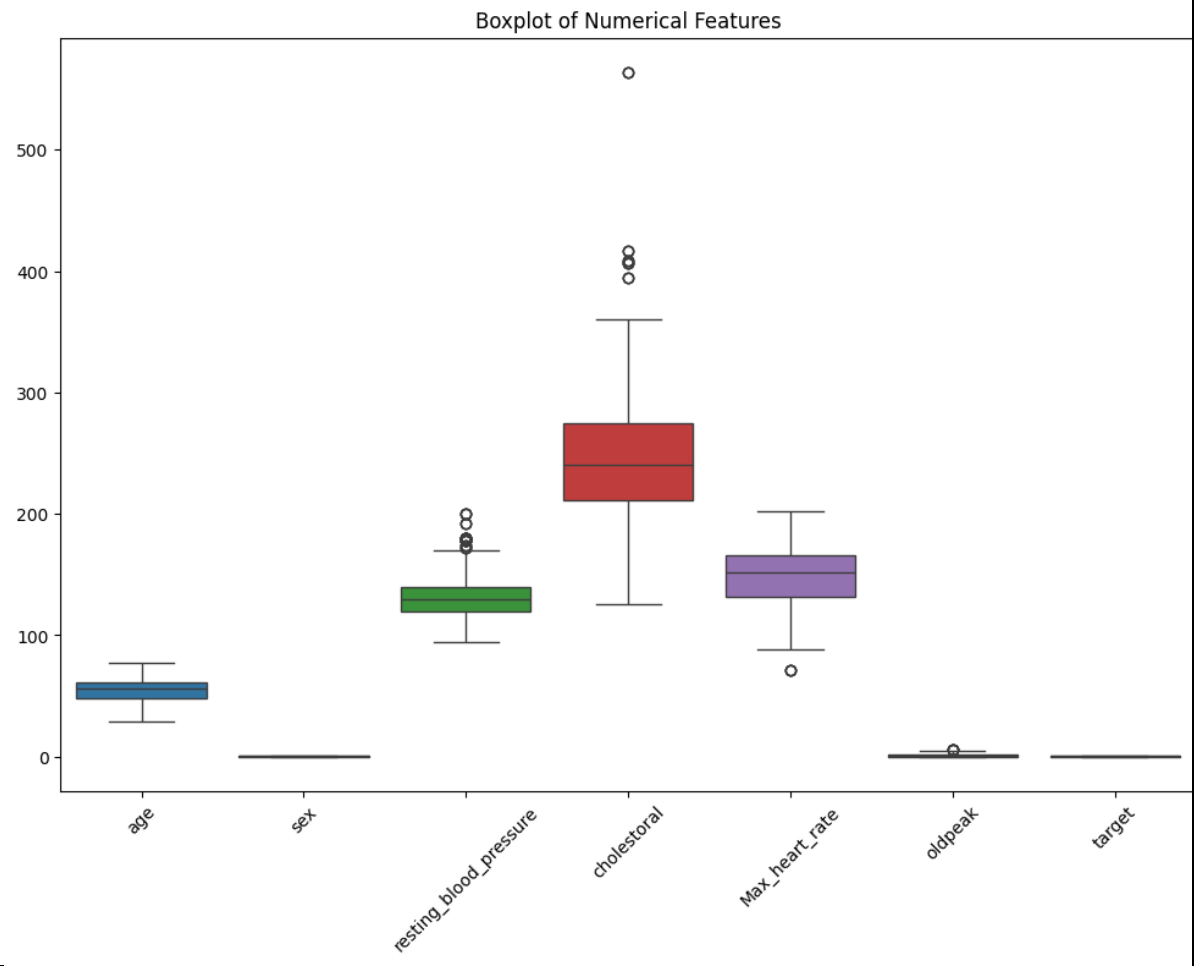



**Analisis:**

airplot ini menunjukkan distribusi dan hubungan antar fitur dalam dataset serta pola pemisahan target (0 dan 1). Fitur seperti Max_heart_rate dan oldpeak menunjukkan pemisahan kelas yang jelas, menjadikannya kandidat kuat untuk prediksi, sementara fitur seperti cholestoral dan resting_blood_pressure memiliki distribusi yang tumpang tindih, menunjukkan prediktabilitas yang rendah secara individual. Hubungan antar fitur seperti age dan resting_blood_pressure menunjukkan korelasi sedang, yang dapat mengindikasikan redundansi dan memerlukan pengurangan dimensi. Scatterplot juga mengungkap adanya hubungan non-linear, misalnya antara oldpeak dan Max_heart_rate, yang menyarankan bahwa algoritma non-linear seperti Random Forest atau Gradient Boosting mungkin lebih efektif daripada model linear.

```python
# 4. Boxplot for outlier detection in numerical features
plt.figure(figsize=(12, 8))
sns.boxplot(data=df[numerical_columns])
plt.title("Boxplot of Numerical Features")
plt.xticks(rotation=45)
plt.show()
```

**Output:**

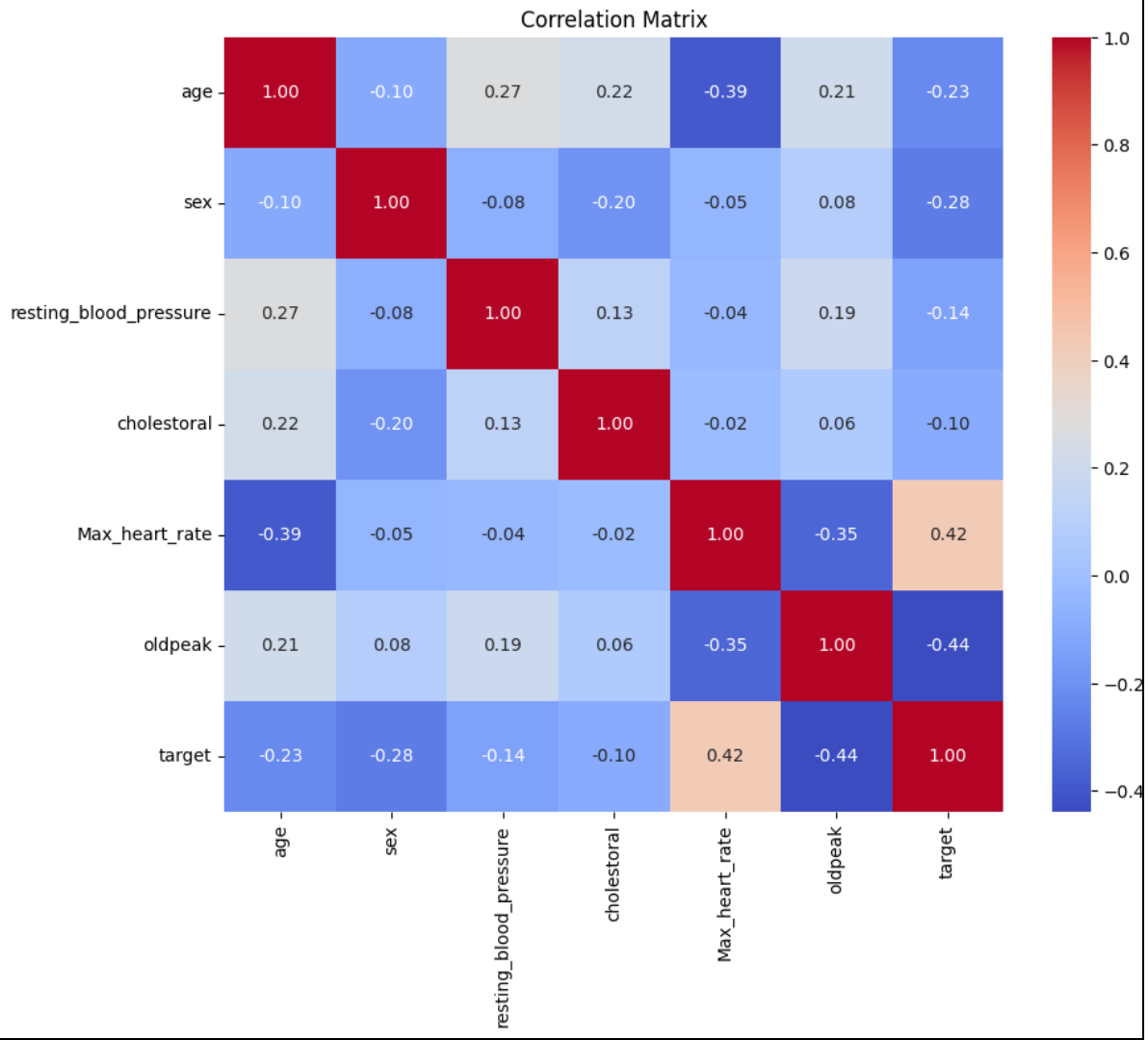



**Analisis:**

Boxplot ini menunjukkan distribusi statistik fitur numerik dalam dataset, mengidentifikasi keberadaan outlier yang signifikan pada fitur seperti cholestoral, resting_blood_pressure, dan oldpeak. Outlier ini dapat memengaruhi performa model machine learning, terutama algoritma yang sensitif seperti Linear Regression dan Support Vector Machines, sehingga memerlukan penanganan seperti winsorization, penghapusan, atau penggunaan **RobustScaler**. Fitur dengan rentang besar seperti Max_heart_rate dan cholestoral memerlukan normalisasi atau standarisasi untuk mencegah dominasi pada algoritma berbasis gradien. Selain itu, fitur dengan distribusi miring, seperti oldpeak, mungkin memerlukan transformasi logaritmik untuk memperbaiki distribusinya.

```python
# Step 3: Display the correlation matrix
correlation_matrix = df[numerical_columns].corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Matrix")
plt.show()
```

**Output :**

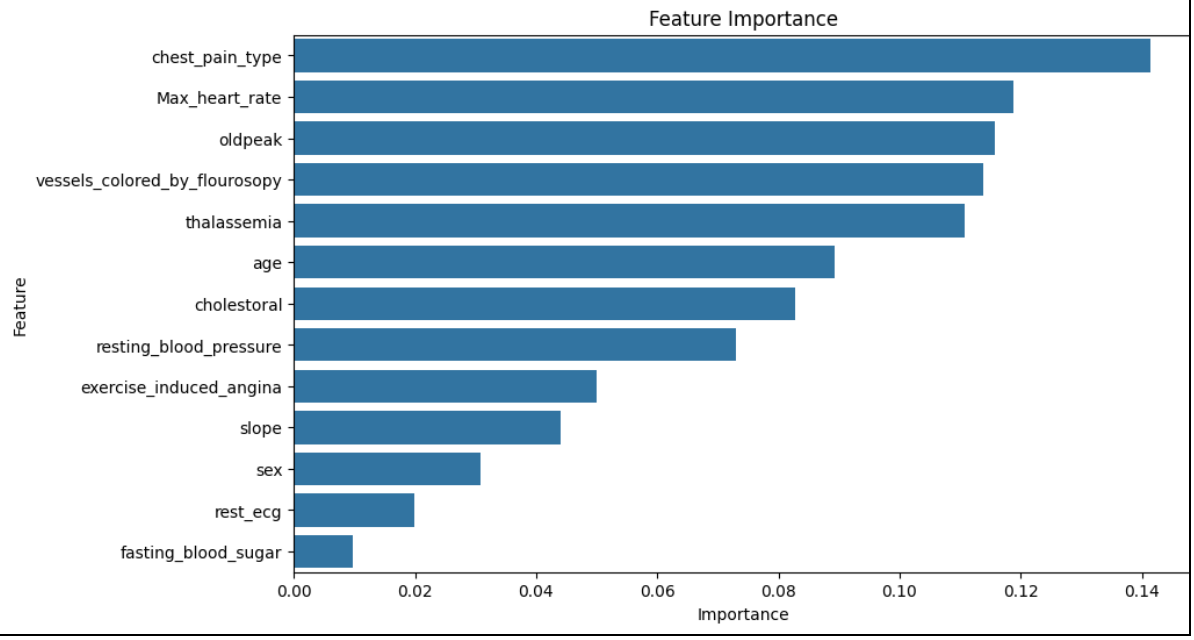



**Analisis:**

Matriks korelasi ini menunjukkan hubungan linier antara fitur numerik dalam dataset, dengan nilai berkisar dari -1 (korelasi negatif sempurna) hingga 1 (korelasi positif sempurna). Fitur Max_heart_rate memiliki korelasi positif moderat dengan target (0.42), menunjukkan bahwa semakin tinggi nilai Max_heart_rate, semakin besar kemungkinan target bernilai 1. Sebaliknya, oldpeak memiliki korelasi negatif signifikan dengan target (-0.44), menandakan bahwa nilai oldpeak yang lebih tinggi cenderung berkaitan dengan target 0. Korelasi antar fitur relatif rendah, seperti antara age dan resting_blood_pressure (0.27), sehingga risiko multikolinearitas rendah.

```python
from sklearn.ensemble import RandomForestClassifier

# Restore target column to binary classification values (0 and 1)
df['target'] = (df['target'] > 0).astype(int)

# Re-separate features and target
X = df.drop('target', axis=1)
y = df['target']

# Fit a Random Forest model to identify feature importance
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X, y)

# Retry fitting Random Forest for feature importance
rf_model.fit(X, y)

# Calculate feature importances
feature_importances = rf_model.feature_importances_
importance_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)

# Visualize feature importance
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=importance_df)
plt.title('Feature Importance')
plt.show()

importance_df
```

**Output:**



**Analisis:**
Kode ini menggunakan algoritma Random Forest untuk mengidentifikasi tingkat kepentingan masing-masing fitur dalam memprediksi target variabel. Hasil visualisasi menunjukkan bahwa fitur chest_pain_type memiliki pengaruh paling signifikan terhadap prediksi, diikuti oleh Max_heart_rate dan oldpeak. Fitur seperti fasting_blood_sugar dan rest_ecg memiliki kontribusi yang paling kecil terhadap model. Informasi ini dapat digunakan untuk seleksi fitur, di mana fitur dengan kepentingan rendah dapat dihapus untuk mengurangi kompleksitas model tanpa signifikan mempengaruhi akurasi. Random Forest menggunakan mekanisme pembagian keputusan secara hierarkis, sehingga memberikan wawasan intuitif tentang fitur mana yang paling relevan untuk membedakan kelas target.

```
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, mean_absolute_error, mean_squared_error, r2_score

# Step 1: Data Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 2: Split the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

**Analisis:**
Kode ini menunjukkan proses awal dalam pipeline pembelajaran mesin dengan dua langkah utama: standarisasi data dan pembagian dataset menjadi set pelatihan dan pengujian. Data fitur (X) distandarisasi menggunakan StandardScaler untuk memastikan setiap fitur memiliki distribusi dengan mean 0 dan standar deviasi 1, sehingga meningkatkan kinerja algoritma yang sensitif terhadap skala fitur seperti neural networks. Setelah itu, dataset dibagi menjadi 80% data pelatihan dan 20% data pengujian menggunakan train_test_split dengan parameter random_state=42 untuk memastikan replikasi pembagian yang sama pada setiap eksekusi.

```python
# Convert data to PyTorch tensors
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train.values, dtype=torch.float32).unsqueeze(1)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test.values, dtype=torch.float32).unsqueeze(1)

# Placeholder for results
results = []

# Step 3: Define MLP model
class MLPModel(nn.Module):
    def __init__(self, input_size, hidden_layers, activation_fn):
        super(MLPModel, self).__init__()
        layers = []
        for neurons in hidden_layers:
            layers.append(nn.Linear(input_size, neurons))
            layers.append(activation_fn())
            input_size = neurons
        layers.append(nn.Linear(input_size, 1))  # Output layer
        layers.append(nn.Sigmoid())  # Sigmoid for binary classification
        self.model = nn.Sequential(*layers)

    def forward(self, x):
        return self.model(x)
```

**Analisis:**
Kode ini menunjukkan proses konversi data menjadi tensor PyTorch dan definisi model Multilayer Perceptron (MLP) untuk klasifikasi biner. Data pelatihan dan pengujian (X_train, y_train, X_test, y_test) dikonversi menjadi tensor dengan tipe data float32, dan label (y_train, y_test) diberi dimensi tambahan menggunakan .unsqueeze(1) agar sesuai dengan bentuk input model. Model MLP didefinisikan dalam kelas MLPModel, yang memungkinkan fleksibilitas dalam menambahkan jumlah lapisan tersembunyi (hidden_layers) dan memilih fungsi aktivasi. Lapisan keluaran menggunakan fungsi aktivasi Sigmoid untuk menghasilkan output antara 0 dan 1, yang sesuai untuk klasifikasi biner. Model ini dibangun menggunakan nn.Sequential untuk menyusun lapisan secara berurutan.

```python
# Step 4: Hyperparameter combinations
hidden_layer_options = [
    [4], [16], [32], [64],              # 1 hidden layer
    [4, 16], [16, 32], [32, 64],       # 2 hidden layers
    [4, 16, 32], [16, 32, 64]          # 3 hidden layers
]
activation_functions = [nn.Identity, nn.Sigmoid, nn.ReLU, nn.Tanh]
epoch_options = [1, 10, 25, 50, 100, 250]
learning_rate_options = [10, 1, 0.1, 0.01, 0.001, 0.0001]
batch_size_options = [16, 32, 64, 128, 256, 512]
```

**Analisis:**
Kode ini mendefinisikan kombinasi hiperparameter yang akan diuji untuk model Multilayer Perceptron (MLP). hidden_layer_options menentukan konfigurasi lapisan tersembunyi dengan 1 hingga 3 lapisan, memungkinkan eksplorasi kedalaman jaringan. Fungsi aktivasi seperti nn.Identity, nn.Sigmoid, nn.ReLU, dan nn.Tanh memungkinkan eksperimen dengan berbagai non-linearitas

dalam model. Pilihan jumlah epoch (1 hingga 250), laju pembelajaran (10 hingga 0.0001), dan ukuran batch (16 hingga 512) mencakup rentang parameter penting untuk memengaruhi stabilitas dan efisiensi pelatihan. Kombinasi ini memungkinkan pencarian hiperparameter yang optimal melalui pendekatan grid search atau eksperimen manual, dengan tujuan menemukan konfigurasi terbaik yang menghasilkan akurasi tinggi dan generalisasi yang baik pada dataset.

```python
# Step 5: Train and evaluate model for each combination
for hidden_layers in hidden_layer_options:
    for activation_fn in activation_functions:
        for epochs in epoch_options:
            for lr in learning_rate_options:
                for batch_size in batch_size_options:
                    # Initialize model, loss, and optimizer
                    model = MLPModel(X_train.shape[1], hidden_layers, activation_fn)
                    criterion = nn.BCELoss()
                    optimizer = optim.Adam(model.parameters(), lr=lr)

                    # Training loop
                    model.train()
                    for epoch in range(epochs):
                        for i in range(0, len(X_train_tensor), batch_size):
                            X_batch = X_train_tensor[i:i+batch_size]
                            y_batch = y_train_tensor[i:i+batch_size]

                            optimizer.zero_grad()
                            outputs = model(X_batch)
                            loss = criterion(outputs, y_batch)
                            loss.backward()
                            optimizer.step()

                    # Evaluate the model
                    model.eval()
                    with torch.no_grad():
                        predictions = model(X_test_tensor).round()
                        acc = (predictions.eq(y_test_tensor).sum() / len(y_test_tensor)).item()
                        mae = mean_absolute_error(y_test, predictions.numpy())
                        mse = mean_squared_error(y_test, predictions.numpy())
                        r2 = r2_score(y_test, predictions.numpy())
```

**Analisis:**

Kode ini mengimplementasikan proses pelatihan dan evaluasi model Multilayer Perceptron (MLP) untuk setiap kombinasi hiperparameter seperti jumlah lapisan tersembunyi, fungsi aktivasi, epoch, laju pembelajaran, dan ukuran batch. Model dilatih menggunakan fungsi loss BCELoss (Binary Cross-Entropy Loss) untuk klasifikasi biner, dengan optimasi dilakukan menggunakan algoritma Adam. Loop pelatihan memproses data dalam mini-batches, menghitung output, loss, dan pembaruan bobot melalui backpropagation. Setelah pelatihan, model dievaluasi pada data pengujian (X_test_tensor) dengan metrik seperti akurasi, Mean Absolute Error (MAE), Mean Squared Error (MSE), dan $R^2$.

```
                # Store the results
                results.append({
                    'Hidden_Layers': hidden_layers,
                    'Activation': activation_fn.__name__,
                    'Epochs': epochs,
                    'Learning_Rate': lr,
                    'Batch_Size': batch_size,
                    'Loss': loss.item(),
                    'Accuracy': acc,
                    'MAE': mae,
                    'MSE': mse,
                    'R2': r2
                })

# Convert results to DataFrame and save to CSV
results_df = pd.DataFrame(results)
results_df = results_df.sort_values(by='Accuracy', ascending=False)
```

**Analisis:**

Kode ini bertujuan untuk menyimpan hasil evaluasi model berdasarkan berbagai kombinasi hiperparameter ke dalam sebuah daftar results, yang kemudian dikonversi menjadi DataFrame menggunakan Pandas untuk analisis lebih lanjut. Setiap kombinasi mencakup informasi tentang struktur model (Hidden_Layers), fungsi aktivasi (Activation), jumlah epoch, laju pembelajaran (Learning_Rate), ukuran batch (Batch_Size), serta metrik performa seperti Loss, akurasi (Accuracy), Mean Absolute Error (MAE), Mean Squared Error (MSE), dan $R^2$. DataFrame hasil evaluasi diurutkan berdasarkan akurasi dalam urutan menurun untuk memprioritaskan kombinasi terbaik.

```
from tabulate import tabulate

# Display the top 10 results in a tabular format using tabulate
top_results = results_df.head(10)
tabulated_results = tabulate(top_results, headers='keys', tablefmt='grid')

print(tabulated_results)
```

**Output:**

| | Hidden_Layers | Activation | Epochs | Learning_Rate | Batch_Size | Loss | Accuracy | MAE | MSE | R2 |
|---|---|---|---|---|---|---|---|---|---|---|
| 4732 | [16, 32] | Sigmoid | 250 | 0.1 | 256 | 0.000206575 | 1 | 0 | 0 | 1 |
| 1274 | [16] | Sigmoid | 250 | 0.1 | 64 | 0.000191023 | 1 | 0 | 0 | 1 |
| 5601 | [32, 64] | Sigmoid | 250 | 0.01 | 128 | 8.53166e-05 | 1 | 0 | 0 | 1 |
| 3433 | [64] | Tanh | 250 | 0.1 | 32 | 1.8788e-06 | 1 | 0 | 0 | 1 |
| 5918 | [32, 64] | Tanh | 25 | 0.1 | 64 | 0.000383622 | 1 | 0 | 0 | 1 |
| 5919 | [32, 64] | Tanh | 25 | 0.1 | 128 | 0.00100707 | 1 | 0 | 0 | 1 |
| 1276 | [16] | Sigmoid | 250 | 0.1 | 256 | 0.00051338 | 1 | 0 | 0 | 1 |
| 5602 | [32, 64] | Sigmoid | 250 | 0.01 | 256 | 0.000150835 | 1 | 0 | 0 | 1 |
| 3152 | [64] | ReLU | 50 | 0.01 | 64 | 0.00627696 | 1 | 0 | 0 | 1 |
| 1241 | [16] | Sigmoid | 100 | 0.1 | 512 | 0.0114415 | 1 | 0 | 0 | 1 |

**Analisis :**

Kode ini menggunakan library tabulate untuk menyajikan 10 hasil teratas dari eksperimen hiperparameter dalam format tabel yang terstruktur. Hasilnya menunjukkan bahwa kombinasi

konfigurasi seperti hidden layers [16, 32], fungsi aktivasi Sigmoid, 250 epoch, learning rate 0.1, dan batch size 256 menghasilkan performa terbaik dengan akurasi 100%, loss sangat kecil (0.000206575), dan metrik MAE, MSE, serta $R^2$ menunjukkan hasil yang sempurna. Secara umum, konfigurasi dengan epoch tinggi (250), learning rate sedang hingga rendah (0.1 atau 0.01), dan batch size besar menghasilkan kinerja yang konsisten baik. Kombinasi seperti fungsi aktivasi Sigmoid dan Tanh mendominasi hasil terbaik, sementara fungsi ReLU hanya muncul di peringkat lebih rendah dengan sedikit perbedaan performa.

```python
model.eval()

# Generate predictions on the test set
with torch.no_grad():
    y_pred = model(X_test_tensor).round().numpy()
    y_true = y_test.values

# Create the classification report
report = classification_report(y_true, y_pred, target_names=['Class 0', 'Class 1'])

# Print the classification report
print("Classification Report:\n")
print(report)
```

**Output:**

```
Classification Report:

              precision    recall  f1-score   support

     Class 0       0.84      0.73      0.78       102
     Class 1       0.76      0.86      0.81       103

    accuracy                           0.80       205
   macro avg       0.80      0.79      0.79       205
weighted avg       0.80      0.80      0.79       205
```

**Analisis :**

Kode ini menghasilkan laporan klasifikasi dari model yang telah dievaluasi pada dataset pengujian. Laporan menunjukkan metrik seperti precision, recall, dan f1-score untuk masing-masing kelas (Class 0 dan Class 1). Model memiliki akurasi keseluruhan sebesar 80%, dengan precision lebih tinggi untuk Class 0 (0.84) dibandingkan Class 1 (0.76), sementara recall lebih tinggi untuk Class 1 (0.86) dibandingkan Class 0 (0.73). Hal ini menunjukkan bahwa model lebih baik dalam mengidentifikasi Class 1 (kemampuan menangkap positif benar) dibandingkan Class 0. F1-score, yang merupakan rata-rata harmonik antara precision dan recall, menunjukkan performa model yang seimbang, masing-masing 0.78 untuk Class 0 dan 0.81 untuk Class 1. Dengan nilai macro dan weighted average F1-score di 0.79, model menunjukkan performa konsisten di kedua kelas.