Import Data dan Library

import pandas as pd import numpy as np

import matplotlib.pyplot as plt import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, MinMaxScaler from sklearn.linear\_model import Ridge, Lasso

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error, r2\_score from sklearn.pipeline import Pipeline

from sklearn.feature\_selection import SelectKBest, SelectPercentile, f\_regression from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score import time

Load Data

file\_path = 'track\_data\_final.csv'

df\_spotify = pd.read\_csv(file\_path) df\_spotify.head()

**track\_id track\_name track\_number track\_popularity track\_duration\_ms explicit artist\_name artist\_popul**

**0** 6pymOcrCnMuCWdgGVTvUgP

3

57

61

213173

False

Britney Spears

**1** 2lWc1iJlz2NVcStV5fbtPG Clouds 1 67 158760 False BUNT.

**2** 1msEuwSBneBKpVCZQcFTsU

Forever & Always (Taylor’s Version)

11

63

225328

False Taylor Swift

**3**

7bcy34fBT2ap1L4bfPsl9q

I Didn't Change My

Number

2

72

158463

True Billie Eilish

**4** 0GLfodYacy3BJE7AI3A8en Man Down 7 57 267013 False Rihanna

print("Jumlah baris, kolom:", df\_spotify.shape) print("\nTipe data:")

print(df\_spotify.dtypes)

Jumlah baris, kolom: (8778, 15) Tipe data:

track\_id object

track\_name object

track\_number int64

track\_popularity int64

track\_duration\_ms int64

explicit bool

artist\_name object artist\_popularity float64 artist\_followers float64

artist\_genres object

album\_id object

album\_name object album\_release\_date object album\_total\_tracks int64

album\_type object dtype: object

Pembersihan Data

Cek dan Menangani Missing Value

|  |  |
| --- | --- |
| track\_id | 0 |
| track\_name | 2 |
| track\_number | 0 |
| track\_popularity | 0 |
| track\_duration\_ms | 0 |
| explicit | 0 |
| artist\_name | 4 |
| artist\_popularity | 4 |
| artist\_followers | 4 |
| artist\_genres | 4 |
| album\_id | 0 |
| album\_name | 2 |
| album\_release\_date | 0 |
| album\_total\_tracks | 0 |
| album\_type  dtype: int64 | 0 |

Cek dan Hapus Duplikat

# Cek ukuran awal

print("Dimensi awal:", df\_spotify.shape)

# Cek missing value

print("\nMissing value per kolom:") print(df\_spotify.isnull().sum())

Dimensi awal: (8778, 15) Missing value per kolom:

# Hapus kolom tidak relevan

cols\_to\_drop = [

'track\_id', 'track\_name', 'artist\_name', 'artist\_genres', 'album\_id', 'album\_name'

]

df\_spotify = df\_spotify.drop(columns=cols\_to\_drop) # Hapus baris dengan missing value

df\_spotify = df\_spotify.dropna()

print("\nDimensi setelah cleaning:", df\_spotify.shape)

Dimensi setelah cleaning: (8774, 9)

Ubah ke Numerik

# Ubah Boolean (True/False) ke Angka (1/0)

df\_spotify['explicit'] = df\_spotify['explicit'].astype(int)

# Ambil tahun rilis dari tanggal

df\_spotify['release\_year'] = pd.to\_datetime(

df\_spotify['album\_release\_date'], errors='coerce'

).dt.year

# Hapus kolom tanggal asli

df\_spotify = df\_spotify.drop(columns=['album\_release\_date'])

# Drop NA hasil konversi tanggal df\_spotify = df\_spotify.dropna()

# One-Hot Encoding kolom kategorikal df\_spotify = pd.get\_dummies(

df\_spotify,

columns=['album\_type'], drop\_first=True

)

print("\n=== DATA SIAP TRAINING ===")

print("Dimensi akhir:", df\_spotify.shape) df\_spotify.head()

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **track\_number** | **track\_popularity** | **track\_duration\_ms** | **explicit** | **artist\_popularity** | **artist\_followers** | **album\_total\_tracks** | **release** |
| **0** 57 | 61 | 213173 | 0 | 80.0 | 17755451.0 | 58 | 2 |
| **1** 1 | 67 | 158760 | 0 | 69.0 | 293734.0 | 1 | 2 |
| **2** 11 | 63 | 225328 | 0 | 100.0 | 145396321.0 | 26 | 2 |
| **3** 2 | 72 | 158463 | 1 | 90.0 | 118692183.0 | 16 | 2 |
| **4** 7 | 57 | 267013 | 0 | 90.0 | 68997177.0 | 13 | 2 |

Pemisahan Fitur Target dan Train Test Split

=== DATA SIAP TRAINING ===

Dimensi akhir: (8573, 10)

# Tentukan Fitur (X) dan Target (y) target = 'track\_popularity'

X = df\_spotify.drop(columns=[target]) y = df\_spotify[target]

# Split Data (80:20) dengan Random State 28 # (Sesuai instruksi: 2 digit terakhir NPM)

X\_train, X\_test, y\_train, y\_test = train\_test\_split( X, y, test\_size=0.2, random\_state=28

)

print(f"Jumlah Data Training: {X\_train.shape[0]}") print(f"Jumlah Data Testing : {X\_test.shape[0]}")

Jumlah Data Training: 6858 Jumlah Data Testing : 1715

Standard Scaler

print("=== HASIL DENGAN STANDARD SCALER ===")

# 1. Scaling (Standardisasi) scaler\_std = StandardScaler()

X\_train\_std = scaler\_std.fit\_transform(X\_train) X\_test\_std = scaler\_std.transform(X\_test)

# 2. Model Ridge (L2 Regularization) ridge = Ridge(alpha=1.0)

ridge.fit(X\_train\_std, y\_train)

y\_pred\_ridge = ridge.predict(X\_test\_std)

# 3. Model Lasso (L1 Regularization) lasso = Lasso(alpha=0.1)

lasso.fit(X\_train\_std, y\_train)

y\_pred\_lasso = lasso.predict(X\_test\_std)

# Evaluasi

print(f"[Ridge] RMSE: {np.sqrt(mean\_squared\_error(y\_test, y\_pred\_ridge)):.4f}, R2: {r2\_score(y\_test, y\_pred\_ridge):.4f}") print(f"[Lasso] RMSE: {np.sqrt(mean\_squared\_error(y\_test, y\_pred\_lasso)):.4f}, R2: {r2\_score(y\_test, y\_pred\_lasso):.4f}")

=== HASIL DENGAN STANDARD SCALER === [Ridge] RMSE: 20.9049, R2: 0.2607

[Lasso] RMSE: 20.9038, R2: 0.2608

Minmax Scaler

print("=== HASIL DENGAN MINMAX SCALER ===")

# 1. Scaling (Normalisasi 0-1) scaler\_mm = MinMaxScaler()

X\_train\_mm = scaler\_mm.fit\_transform(X\_train) X\_test\_mm = scaler\_mm.transform(X\_test)

# 2. Model Ridge

ridge\_mm = Ridge(alpha=1.0)

ridge\_mm.fit(X\_train\_mm, y\_train)

y\_pred\_ridge\_mm = ridge\_mm.predict(X\_test\_mm)

# 3. Model Lasso

lasso\_mm = Lasso(alpha=0.1)

lasso\_mm.fit(X\_train\_mm, y\_train)

y\_pred\_lasso\_mm = lasso\_mm.predict(X\_test\_mm)

# Evaluasi

print(f"[Ridge] RMSE: {np.sqrt(mean\_squared\_error(y\_test, y\_pred\_ridge\_mm)):.4f}, R2: {r2\_score(y\_test, y\_pred\_ridge\_mm):.4f}") print(f"[Lasso] RMSE: {np.sqrt(mean\_squared\_error(y\_test, y\_pred\_lasso\_mm)):.4f}, R2: {r2\_score(y\_test, y\_pred\_lasso\_mm):.4f}")

=== HASIL DENGAN MINMAX SCALER === [Ridge] RMSE: 20.9040, R2: 0.2608

[Lasso] RMSE: 21.0106, R2: 0.2532

Pipeline Ridge Regression

print("=== PIPELINE RIDGE REGRESSION ===")

n\_features = X\_train.shape[1] # 1. Pipeline

pipe\_ridge = Pipeline([

('scaler', MinMaxScaler()),

('feature\_selection', SelectKBest(f\_regression)), ('model', Ridge())

])

# 2. Parameter Grid param\_grid\_ridge = [

{

'feature\_selection': [SelectKBest(f\_regression)],

'feature\_selection k': [5, n\_features],

'model alpha': [0.1, 1.0]

},

{

'feature\_selection': [SelectPercentile(f\_regression)],

'feature\_selection percentile': [25, 50],

'model alpha': [0.1, 1.0]

}

]

# 3. GridSearch

grid\_ridge = GridSearchCV( pipe\_ridge,

param\_grid\_ridge, cv=5,

scoring='r2'

)

grid\_ridge.fit(X\_train, y\_train) # 4. Ambil model terbaik

best\_ridge = grid\_ridge.best\_estimator\_

# 5. Evaluasi

y\_pred = best\_ridge.predict(X\_test)

print("Best Params:", grid\_ridge.best\_params\_)

print("RMSE:", np.sqrt(mean\_squared\_error(y\_test, y\_pred))) print("R2:", r2\_score(y\_test, y\_pred))

=== PIPELINE RIDGE REGRESSION ===

Best Params: {'feature\_selection': SelectKBest(score\_func=<function f\_regression at 0x0000015655281B20>), 'feature\_selection k' RMSE: 20.904765432473148

R2: 0.26074810535801785

Pipeline Lasso Regression

print("=== PIPELINE LASSO REGRESSION ===")

n\_features = X\_train.shape[1] # 1. Pipeline

pipe\_lasso = Pipeline([

('scaler', MinMaxScaler()),

('feature\_selection', SelectKBest(f\_regression)), ('model', Lasso(max\_iter=5000))

])

# 2. Parameter Grid param\_grid\_lasso = [

{

'feature\_selection': [SelectKBest(f\_regression)], 'feature\_selection k': [5, n\_features],

'model alpha': [0.01, 0.1, 1.0]

},

{

'feature\_selection': [SelectPercentile(f\_regression)], 'feature\_selection percentile': [25, 50],

'model alpha': [0.01, 0.1, 1.0]

}

]

# 3. GridSearch

grid\_lasso = GridSearchCV( pipe\_lasso,

param\_grid\_lasso, cv=5,

scoring='r2'

)

grid\_lasso.fit(X\_train, y\_train) # 4. Ambil model terbaik

best\_lasso = grid\_lasso.best\_estimator\_

# 5. Evaluasi

y\_pred = best\_lasso.predict(X\_test)

print("Best Params:", grid\_lasso.best\_params\_)

print("RMSE:", np.sqrt(mean\_squared\_error(y\_test, y\_pred))) print("R2:", r2\_score(y\_test, y\_pred))

=== PIPELINE LASSO REGRESSION ===

Best Params: {'feature\_selection': SelectKBest(score\_func=<function f\_regression at 0x0000015655281B20>), 'feature\_selection k' RMSE: 20.903138907940157

R2: 0.2608631379393621

GridSearchCV

pipe\_lasso = Pipeline([

('feature\_selection', SelectKBest(score\_func=f\_regression)), ('model', Lasso(random\_state=28, max\_iter=5000))

])

from sklearn.feature\_selection import SelectKBest, SelectPercentile, f\_regression param\_grid\_lasso = [

# Lasso

{

'feature\_selection': [None],

'model alpha': [0.001, 0.01, 0.1, 1.0, 10.0, 100.0]

},

# SelectKBest

{

'feature\_selection': [SelectKBest(score\_func=f\_regression)], 'feature\_selection k': [5, 10, 15, 20],

'model alpha': [0.01, 0.1, 1.0]

},

# SelectPercentile

{

'feature\_selection': [SelectPercentile(score\_func=f\_regression)], 'feature\_selection percentile': [25, 50, 75],

'model alpha': [0.01, 0.1, 1.0]

}

]

# Inisialisasi GridSearch

grid\_lasso = GridSearchCV( estimator=pipe\_lasso,

param\_grid=param\_grid\_lasso,

cv=5,

scoring='r2', n\_jobs=-1

# 5-fold cross-validation

# Gunakan R-squared untuk membandingkan model

# Menggunakan semua core CPU untuk komputasi paralel

)

print("Memulai proses fitting GridSearchCV...") start\_time = time.time()

# Latih Grid Search pada data latih grid\_lasso.fit(X\_train, y\_train)

end\_time = time.time()

print(f"Proses fitting selesai dalam {end\_time - start\_time:.2f} detik.")

# --- HASIL DAN EVALUASI ---

best\_lasso = grid\_lasso.best\_estimator\_ y\_pred = best\_lasso.predict(X\_test)

print("\n--- Hasil GridSearchCV dan Evaluasi Model Terbaik ---")

print(f"Best R2 Score (Cross-Validation): {grid\_lasso.best\_score\_:.4f}") print("Best Params:", grid\_lasso.best\_params\_)

# Hitung Metrik Evaluasi pada Test Set

mse = mean\_squared\_error(y\_test, y\_pred) rmse = np.sqrt(mse)

mae = mean\_absolute\_error(y\_test, y\_pred) r2 = r2\_score(y\_test, y\_pred)

print(f"\nMean Squared Error (MSE) pada Test Set: {mse:.4f}")

print(f"Root Mean Squared Error (RMSE) pada Test Set: {rmse:.4f}") print(f"Mean Absolute Error (MAE) pada Test Set: {mae:.4f}")

print(f"R-squared (R2 Score) pada Test Set: {r2:.4f}")

Memulai proses fitting GridSearchCV...

Proses fitting selesai dalam 0.38 detik.

--- Hasil GridSearchCV dan Evaluasi Model Terbaik --- Best R2 Score (Cross-Validation): 0.2477

Best Params: {'feature\_selection': None, 'model alpha': 0.001}

Mean Squared Error (MSE) pada Test Set: 437.0145

Root Mean Squared Error (RMSE) pada Test Set: 20.9049 Mean Absolute Error (MAE) pada Test Set: 15.8236

R-squared (R2 Score) pada Test Set: 0.2607

Scaflerplot Matrix

# Konversi X\_train ke DataFrame dengan nama kolom asli X\_train\_df = pd.DataFrame(X\_train, columns=X.columns)

# Pilih beberapa fitur penting untuk visualisasi (agar tidak terlalu padat) if X\_train\_df.shape[1] > 6:

# Gunakan korelasi untuk pilih fitur terpenting

corr\_matrix = X\_train\_df.corrwith(pd.Series(y\_train.values, index=X\_train\_df.index)).abs() top\_features = corr\_matrix.nlargest(6).index.tolist()

X\_vis = X\_train\_df[top\_features[:6]] else:

X\_vis = X\_train\_df

# Tambahkan target untuk scatter matrix df\_vis = X\_vis.copy()

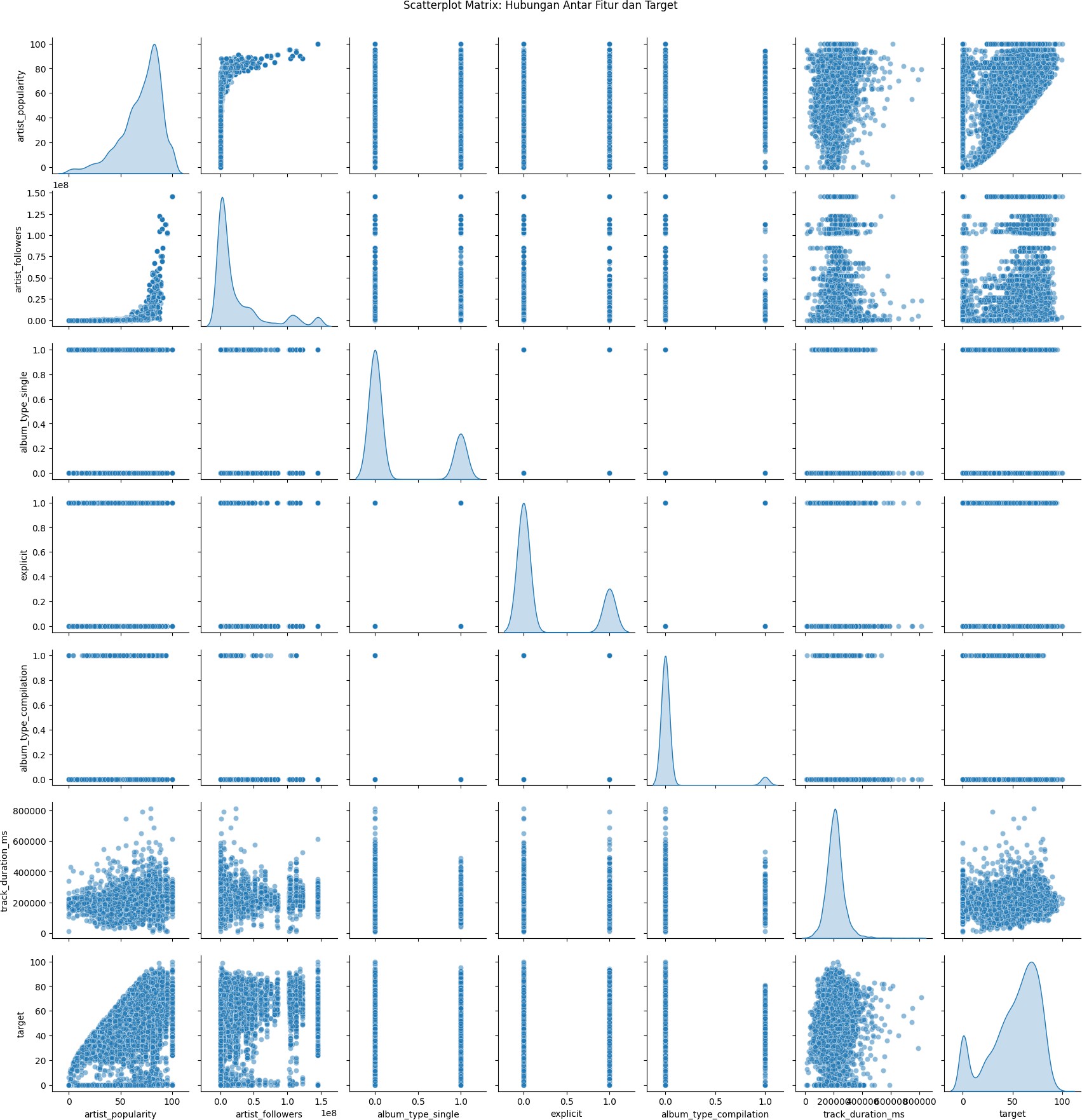
df\_vis['target'] = y\_train.values

print(f"Fitur: {list(X\_vis.columns)}") # Buat scatter matrix

sns.pairplot(df\_vis, diag\_kind='kde', plot\_kws={'alpha': 0.5})

plt.suptitle('Scatterplot Matrix: Hubungan Antar Fitur dan Target', y=1.02) plt.show()

Fitur: ['artist\_popularity', 'artist\_followers', 'album\_type\_single', 'explicit', 'album\_type\_compilation', 'track\_duration\_ms



Plot Residual Training dan Test Data Untuk Setiap Model

models = {

'Ridge': best\_ridge, 'Lasso': best\_lasso

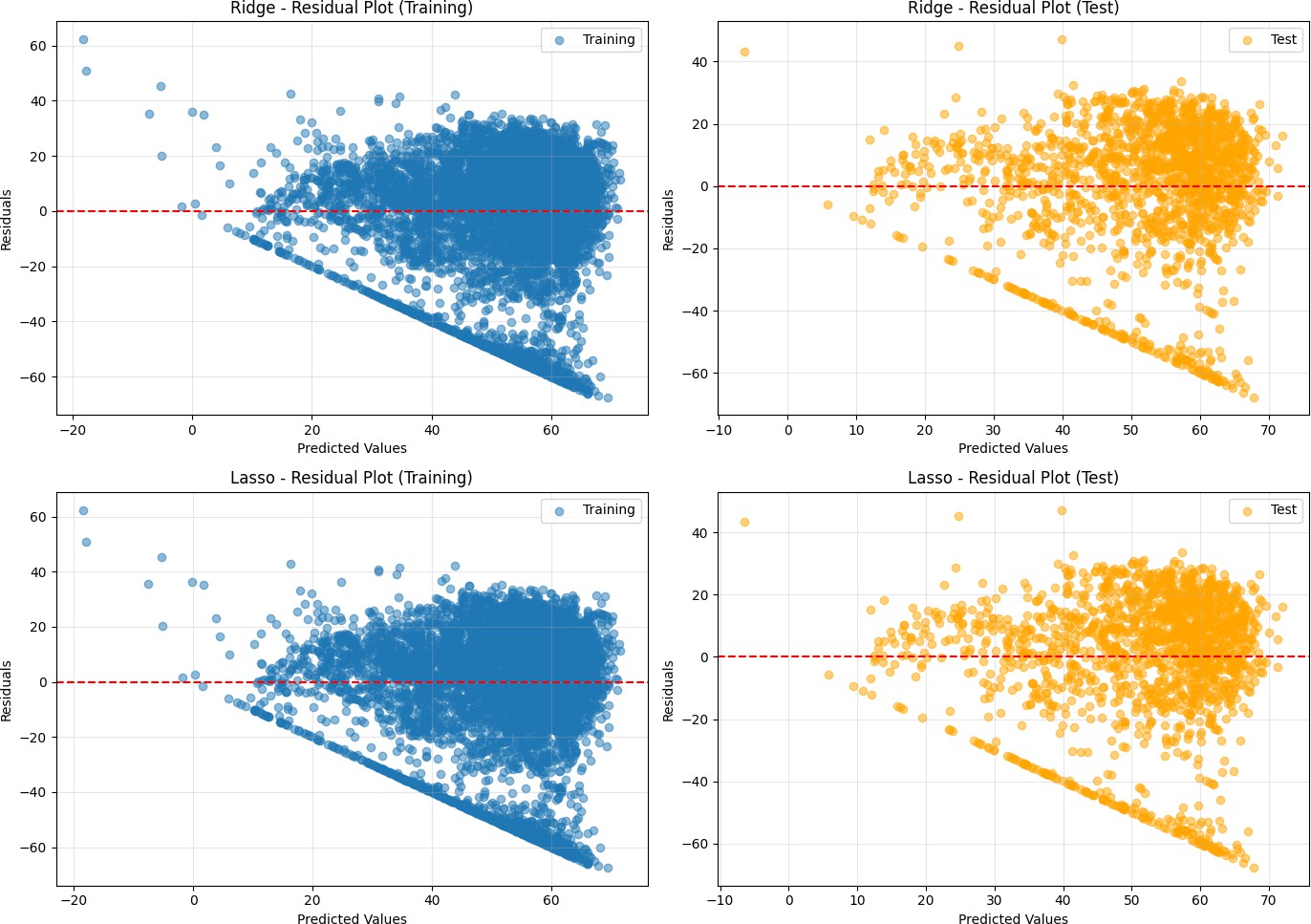
}

fig, axes = plt.subplots(2, 2, figsize=(14, 10)) axes = axes.flatten()

for idx, (name, model) in enumerate(models.items()): # Prediksi untuk training dan test

y\_pred\_train = model.predict(X\_train) y\_pred\_test = model.predict(X\_test)

# Residual



residuals\_train = y\_train - y\_pred\_train residuals\_test = y\_test - y\_pred\_test

# Plot residual training

axes[idx\*2].scatter(y\_pred\_train, residuals\_train, alpha=0.5, label='Training') axes[idx\*2].axhline(y=0, color='r', linestyle='--')

axes[idx\*2].set\_xlabel('Predicted Values') axes[idx\*2].set\_ylabel('Residuals')

axes[idx\*2].set\_title(f'{name} - Residual Plot (Training)') axes[idx\*2].legend()

axes[idx\*2].grid(True, alpha=0.3)

# Plot residual test

axes[idx\*2+1].scatter(y\_pred\_test, residuals\_test, alpha=0.5, label='Test', color='orange') axes[idx\*2+1].axhline(y=0, color='r', linestyle='--')

axes[idx\*2+1].set\_xlabel('Predicted Values') axes[idx\*2+1].set\_ylabel('Residuals')

axes[idx\*2+1].set\_title(f'{name} - Residual Plot (Test)') axes[idx\*2+1].legend()

axes[idx\*2+1].grid(True, alpha=0.3)

plt.tight\_layout() plt.show()

Plot antara Fitur dan Target Untuk Setiap Model Pada Training Data

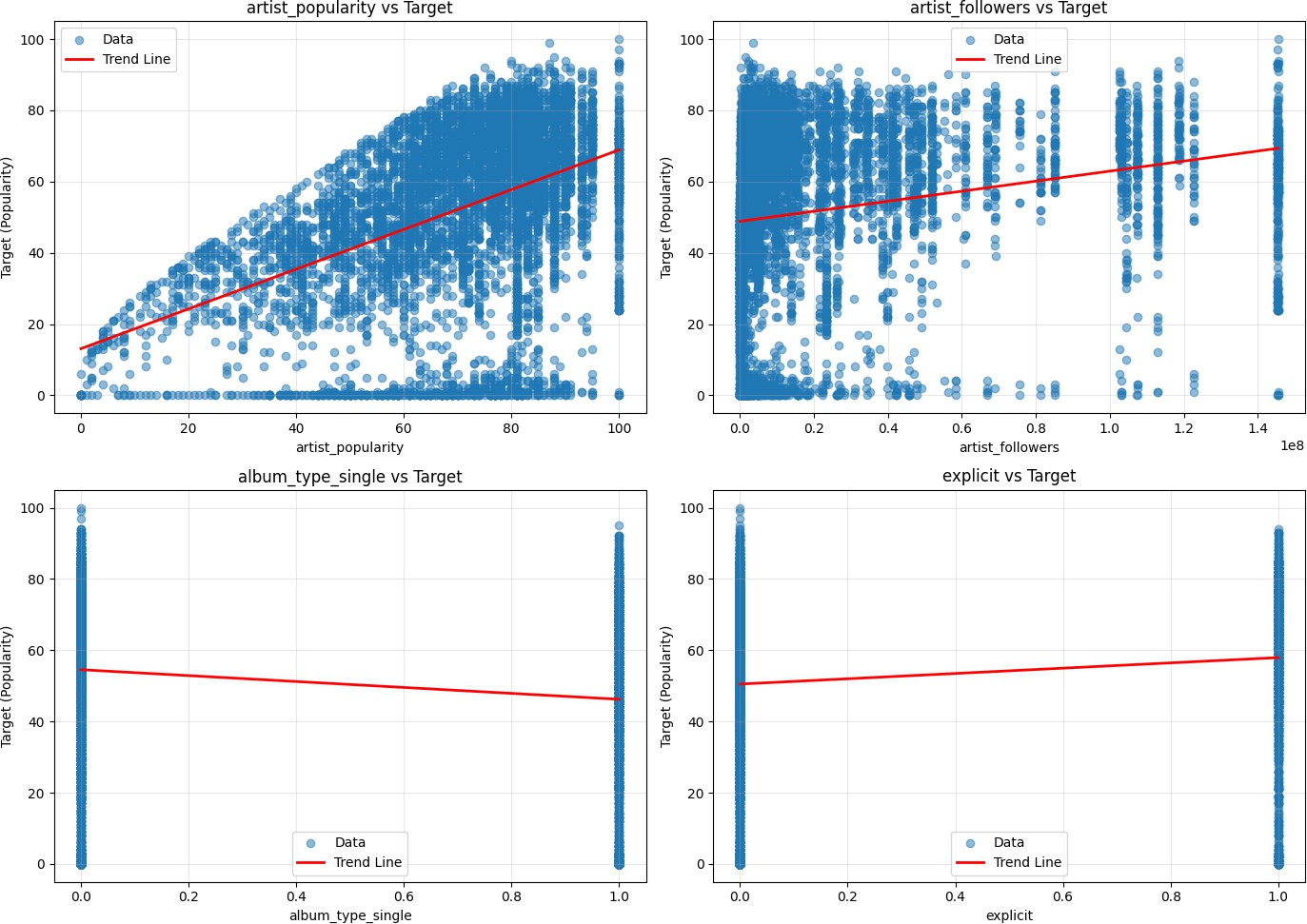
# Konversi X\_train ke DataFrame

X\_train\_df = pd.DataFrame(X\_train, columns=X.columns)

# Hitung korelasi dengan target

correlations = X\_train\_df.corrwith(pd.Series(y\_train.values, index=X\_train\_df.index)).abs() top\_4\_features = correlations.nlargest(4).index.tolist()

print(f"4 fitur dengan korelasi tertinggi: {top\_4\_features}")



fig, axes = plt.subplots(2, 2, figsize=(14, 10)) axes = axes.flatten()

for i, feat\_name in enumerate(top\_4\_features):

# Scatter plot - AKSES dengan NAMA kolom, bukan indeks

axes[i].scatter(X\_train\_df[feat\_name], y\_train, alpha=0.5, label='Data')

# Tambahkan garis regresi

x\_vals = X\_train\_df[feat\_name]

# Regresi linear sederhana

coeffs = np.polyfit(x\_vals, y\_train, 1) poly = np.poly1d(coeffs)

x\_range = np.linspace(x\_vals.min(), x\_vals.max(), 100)

axes[i].plot(x\_range, poly(x\_range), color='red', linewidth=2, label='Trend Line')

axes[i].set\_xlabel(feat\_name)

axes[i].set\_ylabel('Target (Popularity)') axes[i].set\_title(f'{feat\_name} vs Target') axes[i].legend()

axes[i].grid(True, alpha=0.3)

plt.tight\_layout() plt.show()

4 fitur dengan korelasi tertinggi: ['artist\_popularity', 'artist\_followers', 'album\_type\_single', 'explicit']

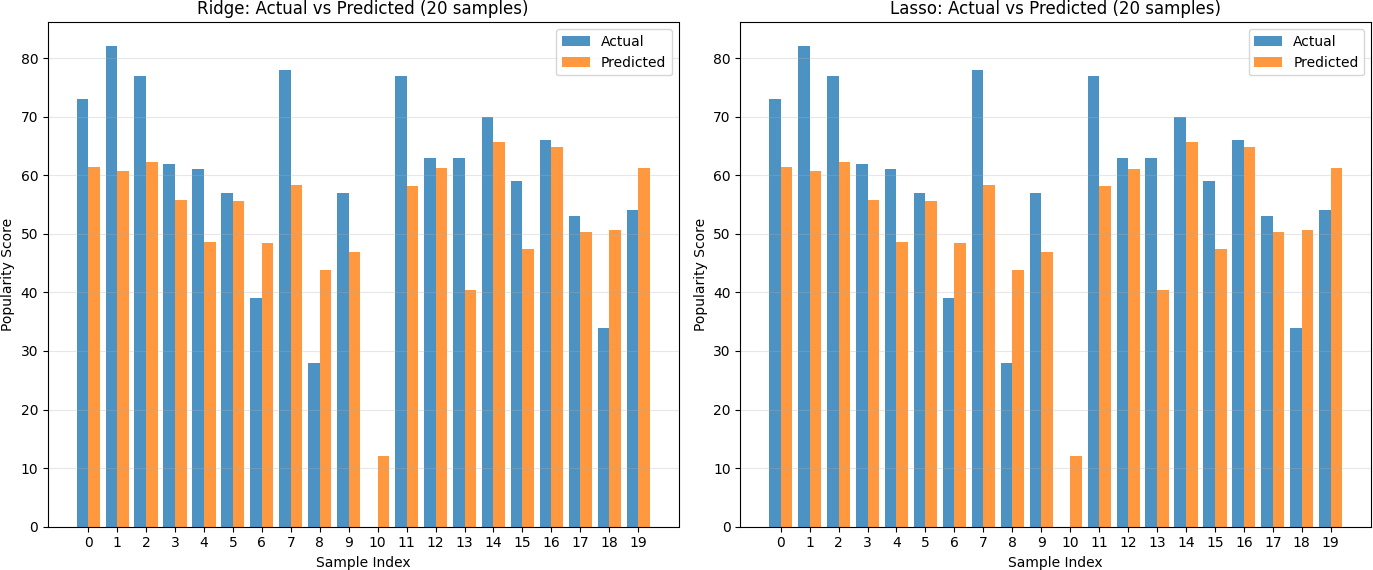
Prediksi vs Aktual

fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Ambil 20 sampel pertama

sample\_indices = range(min(20, len(X\_test)))

for idx, (name, model) in enumerate(models.items()): # Prediksi untuk 20 sampel pertama



X\_sample = X\_test.iloc[sample\_indices] if hasattr(X\_test, 'iloc') else X\_test[sample\_indices]

y\_true\_sample = y\_test.iloc[sample\_indices] if hasattr(y\_test, 'iloc') else y\_test[sample\_indices] y\_pred\_sample = model.predict(X\_sample)

# Buat dataframe untuk plotting df\_compare = pd.DataFrame({

'Actual': y\_true\_sample,

'Predicted': y\_pred\_sample,

'Sample': list(sample\_indices)

})

# Plot bar chart

x\_pos = np.arange(len(df\_compare))

axes[idx].bar(x\_pos - 0.2, df\_compare['Actual'], width=0.4, label='Actual', alpha=0.8)

axes[idx].bar(x\_pos + 0.2, df\_compare['Predicted'], width=0.4, label='Predicted', alpha=0.8)

axes[idx].set\_xlabel('Sample Index')

axes[idx].set\_ylabel('Popularity Score')

axes[idx].set\_title(f'{name}: Actual vs Predicted (20 samples)') axes[idx].set\_xticks(x\_pos)

axes[idx].legend()

axes[idx].grid(True, alpha=0.3, axis='y')

plt.tight\_layout() plt.show()

Mengambil Model Terbaik

best\_lasso\_pipeline = grid\_lasso.best\_estimator\_ best\_params = grid\_lasso.best\_params\_

best\_cv\_score = grid\_lasso.best\_score\_

# Prediksi pada Test Set

y\_pred\_best = best\_lasso\_pipeline.predict(X\_test)

# Hitung Metrik Evaluasi Akhir

mse = mean\_squared\_error(y\_test, y\_pred\_best) rmse = np.sqrt(mse)

mae = mean\_absolute\_error(y\_test, y\_pred\_best) r2 = r2\_score(y\_test, y\_pred\_best)

print(f"Algoritma: Lasso Regression (dalam Pipeline)") print(f"Parameter Terbaik: {best\_params}")

print(f"R2 Score (Cross-Validation Terbaik): {best\_cv\_score:.4f}") print("-" \* 50)

print("Metrik Evaluasi pada TEST SET:") print(f"R-squared (R2): {r2:.4f}")

print(f"Root Mean Squared Error (RMSE): {rmse:.4f}") print(f"Mean Squared Error (MSE): {mse:.4f}")

print(f"Mean Absolute Error (MAE): {mae:.4f}")

Algoritma: Lasso Regression (dalam Pipeline)

Parameter Terbaik: {'feature\_selection': None, 'model alpha': 0.001} R2 Score (Cross-Validation Terbaik): 0.2477

Metrik Evaluasi pada TEST SET: R-squared (R2): 0.2607

Root Mean Squared Error (RMSE): 20.9049

Mean Squared Error (MSE): 437.0145 Mean Absolute Error (MAE): 15.8236

Export Pickle

import pickle

# Simpan model Lasso Regression yang sudah dilatih ke file .pkl (format biner) with open('notebook1\_best\_model.pkl', 'wb') as f:

pickle.dump(best\_lasso\_pipeline, f)

# Konfirmasi bahwa model berhasil disimpan

print(" Model Lasso Regression berhasil disimpan ke notebook1\_best\_model.pkl")

Model Lasso Regression berhasil disimpan ke notebook1\_best\_model.pkl

Import Data

import pandas as pd import numpy as np

import matplotlib.pyplot as plt import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, MinMaxScaler from sklearn.linear\_model import Ridge, Lasso

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score from sklearn.pipeline import Pipeline

from sklearn.feature\_selection import SelectKBest, SelectPercentile, f\_regression from sklearn.model\_selection import GridSearchCV

import time

Load Data

file\_path = 'track\_data\_final.csv'

df\_spotify = pd.read\_csv(file\_path) df\_spotify.head()

**track\_id track\_name track\_number track\_popularity track\_duration\_ms explicit artist\_name artist\_popul**

**0** 6pymOcrCnMuCWdgGVTvUgP

3

57

61

213173

False

Britney Spears

**1** 2lWc1iJlz2NVcStV5fbtPG Clouds 1 67 158760 False BUNT.

**2** 1msEuwSBneBKpVCZQcFTsU

Forever & Always (Taylor’s Version)

11

63

225328

False Taylor Swift

**3**

7bcy34fBT2ap1L4bfPsl9q

I Didn't Change My

Number

2

72

158463

True Billie Eilish

**4** 0GLfodYacy3BJE7AI3A8en Man Down 7 57 267013 False Rihanna

print("Jumlah baris, kolom:", df\_spotify.shape) print("\nTipe data:")

print(df\_spotify.dtypes)

Jumlah baris, kolom: (8778, 15) Tipe data:

track\_id object

track\_name object

track\_number int64

track\_popularity int64

track\_duration\_ms int64

explicit bool

artist\_name object artist\_popularity float64 artist\_followers float64

artist\_genres object

album\_id object

album\_name object album\_release\_date object album\_total\_tracks int64

album\_type object dtype: object

Pembersihan Data

Cek dan Menangani Missing Value

|  |  |
| --- | --- |
| track\_id | 0 |
| track\_name | 2 |
| track\_number | 0 |
| track\_popularity | 0 |
| track\_duration\_ms | 0 |
| explicit | 0 |
| artist\_name | 4 |
| artist\_popularity | 4 |
| artist\_followers | 4 |
| artist\_genres | 4 |
| album\_id | 0 |
| album\_name | 2 |
| album\_release\_date | 0 |
| album\_total\_tracks | 0 |
| album\_type  dtype: int64 | 0 |

Cek dan Hapus Duplikat

# Cek ukuran awal

print("Dimensi awal:", df\_spotify.shape)

# Cek missing value

print("\nMissing value per kolom:") print(df\_spotify.isnull().sum())

Dimensi awal: (8778, 15) Missing value per kolom:

# Hapus kolom tidak relevan

cols\_to\_drop = [

'track\_id', 'track\_name', 'artist\_name', 'artist\_genres', 'album\_id', 'album\_name'

]

df\_spotify = df\_spotify.drop(columns=cols\_to\_drop) # Hapus baris dengan missing value

df\_spotify = df\_spotify.dropna()

print("\nDimensi setelah cleaning:", df\_spotify.shape)

Dimensi setelah cleaning: (8774, 9)

Ubah ke Numerik

# Ubah Boolean (True/False) ke Angka (1/0)

df\_spotify['explicit'] = df\_spotify['explicit'].astype(int)

# Ambil tahun rilis dari tanggal

df\_spotify['release\_year'] = pd.to\_datetime(

df\_spotify['album\_release\_date'], errors='coerce'

).dt.year

# Hapus kolom tanggal asli

df\_spotify = df\_spotify.drop(columns=['album\_release\_date'])

# Drop NA hasil konversi tanggal df\_spotify = df\_spotify.dropna()

# One-Hot Encoding kolom kategorikal df\_spotify = pd.get\_dummies(

df\_spotify,

columns=['album\_type'], drop\_first=True

)

print("\n=== DATA SIAP TRAINING ===")

print("Dimensi akhir:", df\_spotify.shape) df\_spotify.head()

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **track\_number** | **track\_popularity** | **track\_duration\_ms** | **explicit** | **artist\_popularity** | **artist\_followers** | **album\_total\_tracks** | **release** |
| **0** 57 | 61 | 213173 | 0 | 80.0 | 17755451.0 | 58 | 2 |
| **1** 1 | 67 | 158760 | 0 | 69.0 | 293734.0 | 1 | 2 |
| **2** 11 | 63 | 225328 | 0 | 100.0 | 145396321.0 | 26 | 2 |
| **3** 2 | 72 | 158463 | 1 | 90.0 | 118692183.0 | 16 | 2 |
| **4** 7 | 57 | 267013 | 0 | 90.0 | 68997177.0 | 13 | 2 |

Pemisahan Fitur Target dan Train Test Split

=== DATA SIAP TRAINING ===

Dimensi akhir: (8573, 10)

# Tentukan Fitur (X) dan Target (y) target = 'track\_popularity'

X = df\_spotify.drop(columns=[target]) y = df\_spotify[target]

# Split Data (80:20) dengan Random State 28 # (Sesuai instruksi: 2 digit terakhir NPM)

X\_train, X\_test, y\_train, y\_test = train\_test\_split( X, y, test\_size=0.2, random\_state=28

)

print(f"Jumlah Data Training: {X\_train.shape[0]}") print(f"Jumlah Data Testing : {X\_test.shape[0]}")

Jumlah Data Training: 6858 Jumlah Data Testing : 1715

Standard Scaler

print("=== HASIL DENGAN STANDARD SCALER ===")

# Scaling

scaler\_std = StandardScaler()

X\_train\_std = scaler\_std.fit\_transform(X\_train) X\_test\_std = scaler\_std.transform(X\_test)

# 1. Decision Tree (Rawan Overfitting)

dt = DecisionTreeRegressor(random\_state=28) dt.fit(X\_train\_std, y\_train)

y\_pred\_dt = dt.predict(X\_test\_std)

# 2. Random Forest (Ensemble - Lebih Stabil)

rf = RandomForestRegressor(n\_estimators=100, random\_state=28) rf.fit(X\_train\_std, y\_train)

y\_pred\_rf = rf.predict(X\_test\_std)

# Evaluasi

print(f"[Decision Tree] RMSE: {np.sqrt(mean\_squared\_error(y\_test, y\_pred\_dt)):.4f}, R2: {r2\_score(y\_test, y\_pred\_dt):.4f}") print(f"[Random Forest] RMSE: {np.sqrt(mean\_squared\_error(y\_test, y\_pred\_rf)):.4f}, R2: {r2\_score(y\_test, y\_pred\_rf):.4f}")

=== HASIL DENGAN STANDARD SCALER ===

[Decision Tree] RMSE: 26.6669, R2: -0.2029

[Random Forest] RMSE: 20.0679, R2: 0.3188

Minmax Scaler

print("=== HASIL DENGAN MINMAX SCALER ===")

# Scaling

scaler\_mm = MinMaxScaler()

X\_train\_mm = scaler\_mm.fit\_transform(X\_train) X\_test\_mm = scaler\_mm.transform(X\_test)

# Decision Tree

dt\_mm = DecisionTreeRegressor(random\_state=28) dt\_mm.fit(X\_train\_mm, y\_train)

y\_pred\_dt\_mm = dt\_mm.predict(X\_test\_mm)

# Random Forest

rf\_mm = RandomForestRegressor(n\_estimators=100, random\_state=28) rf\_mm.fit(X\_train\_mm, y\_train)

y\_pred\_rf\_mm = rf\_mm.predict(X\_test\_mm)

# Evaluasi

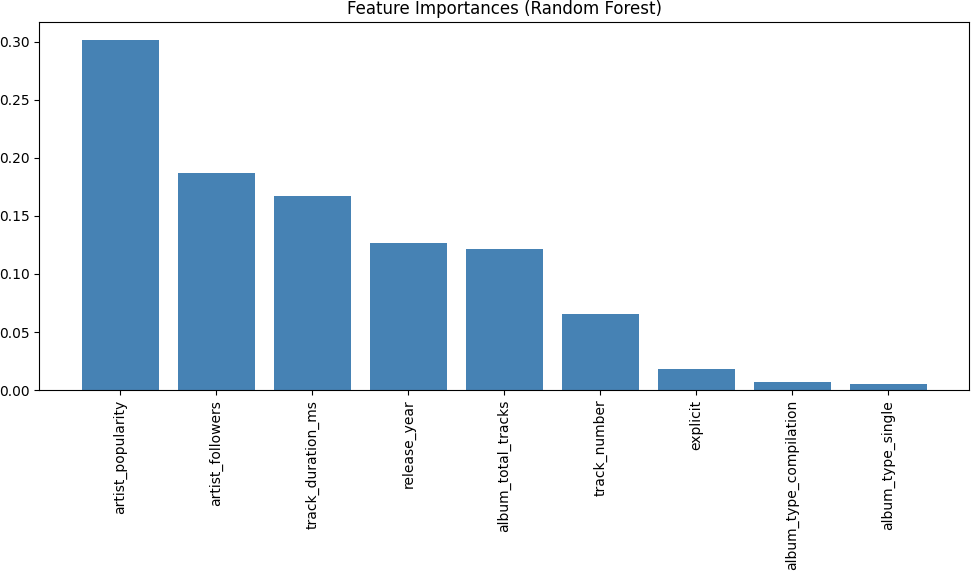
print(f"[Decision Tree] RMSE: {np.sqrt(mean\_squared\_error(y\_test, y\_pred\_dt\_mm)):.4f}, R2: {r2\_score(y\_test, y\_pred\_dt\_mm):.4f} print(f"[Random Forest] RMSE: {np.sqrt(mean\_squared\_error(y\_test, y\_pred\_rf\_mm)):.4f}, R2: {r2\_score(y\_test, y\_pred\_rf\_mm):.4f}

=== HASIL DENGAN MINMAX SCALER ===

[Decision Tree] RMSE: 26.8662, R2: -0.2210

[Random Forest] RMSE: 20.0647, R2: 0.3190

Visualisasi Fitur Penting



# Visualisasi Fitur Penting (Random Forest) importances = rf.feature\_importances\_

feature\_names = X.columns

indices = np.argsort(importances)[::-1]

plt.figure(figsize=(10, 6))

plt.title("Feature Importances (Random Forest)")

plt.bar(range(X.shape[1]), importances[indices], align="center", color='steelblue') plt.xticks(range(X.shape[1]), feature\_names[indices], rotation=90)

plt.tight\_layout() plt.show()

Pipeline Decision Tree Regression

print("=== PIPELINE DECISION TREE REGRESSION ===")

n\_features = X\_train.shape[1] # 1. Pipeline

pipe\_dt = Pipeline([

('scaler', MinMaxScaler()),

('feature\_selection', SelectKBest(f\_regression)), ('model', DecisionTreeRegressor(random\_state=28))

])

# 2. Parameter Grid param\_grid\_dt = [

{

'feature\_selection': [SelectKBest(f\_regression)], 'feature\_selection k': [5, n\_features],

'model max\_depth': [None, 5, 10]

},

{

'feature\_selection': [SelectPercentile(f\_regression)], 'feature\_selection percentile': [25, 50],

'model max\_depth': [None, 5, 10]

}

]

# 3. GridSearch

grid\_dt = GridSearchCV( pipe\_dt,

param\_grid\_dt, cv=5,

scoring='r2'

)

grid\_dt.fit(X\_train, y\_train) # 4. Evaluasi

y\_pred = grid\_dt.best\_estimator\_.predict(X\_test)

print("Best Params:", grid\_dt.best\_params\_)

print("RMSE:", np.sqrt(mean\_squared\_error(y\_test, y\_pred))) print("R2:", r2\_score(y\_test, y\_pred))

=== PIPELINE DECISION TREE REGRESSION ===

Best Params: {'feature\_selection': SelectKBest(score\_func=<function f\_regression at 0x000001F4B6F61BC0>), 'feature\_selection k' RMSE: 20.803718291242554

R2: 0.2678774610317086

Pipeline Random Forest Regression

print("=== PIPELINE RANDOM FOREST REGRESSION ===")

n\_features = X\_train.shape[1] # 1. Pipeline

pipe\_rf = Pipeline([

('scaler', MinMaxScaler()),

('feature\_selection', SelectKBest(f\_regression)), ('model', RandomForestRegressor(random\_state=28))

])

# 2. Parameter Grid param\_grid\_rf = [

{

'feature\_selection': [SelectKBest(f\_regression)], 'feature\_selection k': [5, n\_features],

'model n\_estimators': [100], 'model max\_depth': [None, 10]

},

{

'feature\_selection': [SelectPercentile(f\_regression)], 'feature\_selection percentile': [25, 50],

'model n\_estimators': [100], 'model max\_depth': [None, 10]

}

]

# 3. GridSearch

grid\_rf = GridSearchCV( pipe\_rf,

param\_grid\_rf, cv=5,

scoring='r2', n\_jobs=-1

)

grid\_rf.fit(X\_train, y\_train)

# 4. Evaluasi

y\_pred = grid\_rf.best\_estimator\_.predict(X\_test)

print("Best Params:", grid\_rf.best\_params\_)

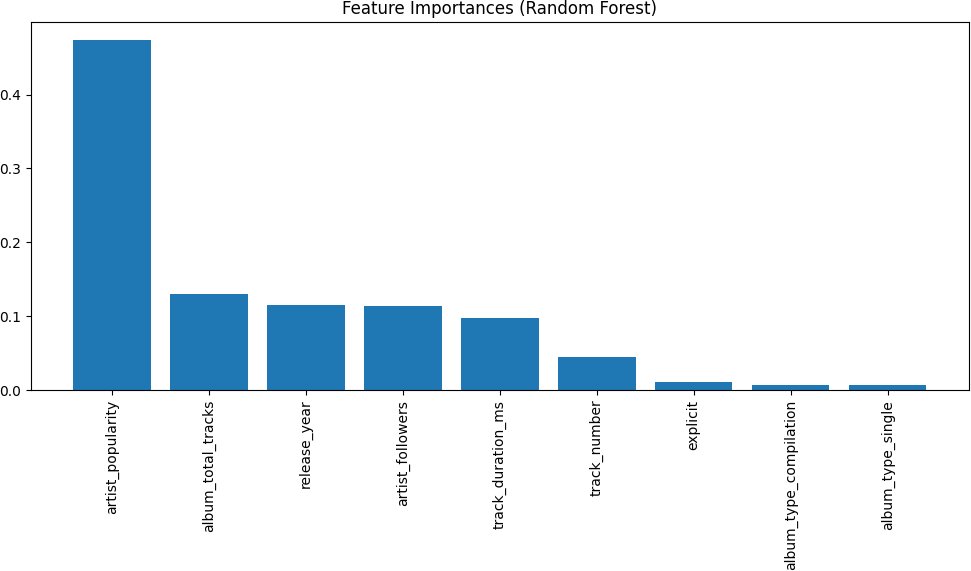
print("RMSE:", np.sqrt(mean\_squared\_error(y\_test, y\_pred))) print("R2:", r2\_score(y\_test, y\_pred))

=== PIPELINE RANDOM FOREST REGRESSION ===

Best Params: {'feature\_selection': SelectKBest(score\_func=<function f\_regression at 0x000001F4B6F61BC0>), 'feature\_selection k' RMSE: 19.67199226438753

R2: 0.3453660288160846

Model Terbaik



best\_rf = grid\_rf.best\_estimator\_.named\_steps['model'] importances = best\_rf.feature\_importances\_

feature\_names = X.columns

indices = np.argsort(importances)[::-1] plt.figure(figsize=(10, 6))

plt.title("Feature Importances (Random Forest)")

plt.bar(range(len(importances)), importances[indices])

plt.xticks(range(len(importances)), feature\_names[indices], rotation=90) plt.tight\_layout()

plt.show()

GridSearchCV Random Forest

pipe\_rf = Pipeline([

# 'feature\_selection' adalah placeholder

('feature\_selection', SelectKBest(score\_func=f\_regression)), # 'model' adalah Random Forest Regressor

('model', RandomForestRegressor(random\_state=28))

])

param\_grid\_rf = [

# Kumpulan 1: Tanpa Feature Selection (Random Forest Saja)

{

'feature\_selection': [None],

'model n\_estimators': [50, 100, 200], # Coba jumlah estimator

'model max\_depth': [5, 10, None], # Coba kedalaman maksimum

'model min\_samples\_split': [2, 5] # Coba min samples split

},

# Kumpulan 2: Dengan SelectKBest

{

'feature\_selection': [SelectKBest(score\_func=f\_regression)],

'feature\_selection k': [10, 20, 30], # Coba jumlah fitur terbaik (k) 'model n\_estimators': [100],

'model max\_depth': [10, 20],

}

]

grid\_rf = GridSearchCV(

estimator=pipe\_rf,

param\_grid=param\_grid\_rf,

cv=5,

scoring='r2', n\_jobs=-1

# Gunakan 5-fold cross-validation

# Gunakan R-squared untuk regresi # Menggunakan semua inti CPU

)

print("Memulai proses fitting GridSearchCV untuk Random Forest...") start\_time = time.time()

# Latih Grid Search pada data latih grid\_rf.fit(X\_train, y\_train)

end\_time = time.time()

print(f"Proses fitting selesai dalam {end\_time - start\_time:.2f} detik.")

best\_rf\_pipeline = grid\_rf.best\_estimator\_

y\_pred\_rf = best\_rf\_pipeline.predict(X\_test)

print("\n--- Hasil Terbaik Random Forest ---")

print(f"Best R2 Score (Cross-Validation): {grid\_rf.best\_score\_:.4f}") print("Best Params:", grid\_rf.best\_params\_)

# Evaluasi Metrik pada Test Set

mse = mean\_squared\_error(y\_test, y\_pred\_rf) rmse = np.sqrt(mse)

mae = mean\_absolute\_error(y\_test, y\_pred\_rf) r2 = r2\_score(y\_test, y\_pred\_rf)

print(f"\nMean Squared Error (MSE) pada Test Set: {mse:.4f}")

print(f"Root Mean Squared Error (RMSE) pada Test Set: {rmse:.4f}") print(f"Mean Absolute Error (MAE) pada Test Set: {mae:.4f}")

print(f"R-squared (R2 Score) pada Test Set: {r2:.4f}")

Memulai proses fitting GridSearchCV untuk Random Forest... Proses fitting selesai dalam 70.05 detik.

--- Hasil Terbaik Random Forest ---

Best R2 Score (Cross-Validation): 0.3209

Best Params: {'feature\_selection': None, 'model max\_depth': 10, 'model min\_samples\_split': 5, 'model n\_estimators': 200}

Mean Squared Error (MSE) pada Test Set: 385.4812

Root Mean Squared Error (RMSE) pada Test Set: 19.6337 Mean Absolute Error (MAE) pada Test Set: 14.4914

R-squared (R2 Score) pada Test Set: 0.3479

GridSearchCV DecisionTree

pipe\_dt = Pipeline([

# 'feature\_selection' adalah placeholder

('feature\_selection', SelectKBest(score\_func=f\_regression)), # 'model' adalah Decision Tree Regressor

('model', DecisionTreeRegressor(random\_state=28))

])

param\_grid\_dt = [

# Kumpulan 1: Tanpa Feature Selection (Decision Tree Saja)

{

'feature\_selection': [None],

'model max\_depth': [3, 5, 8, 12, None], # Kedalaman pohon

'model min\_samples\_split': [2, 10, 20] # Minimum samples untuk split

},

# Kumpulan 2: Dengan SelectKBest

{

'feature\_selection': [SelectKBest(score\_func=f\_regression)],

'feature\_selection k': [10, 20, 30], # Coba jumlah fitur terbaik (k) 'model max\_depth': [10, 20],

'model min\_samples\_split': [2, 10]

}

]

grid\_dt = GridSearchCV( estimator=pipe\_dt,

param\_grid=param\_grid\_dt, cv=5,

scoring='r2', n\_jobs=-1

)

print("Memulai proses fitting GridSearchCV untuk Decision Tree...") start\_time = time.time()

# Latih Grid Search

# Pastikan X\_train dan y\_train sudah tersedia grid\_dt.fit(X\_train, y\_train)

end\_time = time.time()

print(f"Proses fitting selesai dalam {end\_time - start\_time:.2f} detik.")

# --- Evaluasi Hasil Terbaik DT ---

best\_dt\_pipeline = grid\_dt.best\_estimator\_

y\_pred\_dt = best\_dt\_pipeline.predict(X\_test)

print("\n--- Hasil Terbaik Decision Tree ---")

print(f"Best R2 Score (Cross-Validation): {grid\_dt.best\_score\_:.4f}") print("Best Params:", grid\_dt.best\_params\_)

mse = mean\_squared\_error(y\_test, y\_pred\_dt) rmse = np.sqrt(mse)

mae = mean\_absolute\_error(y\_test, y\_pred\_dt) r2 = r2\_score(y\_test, y\_pred\_dt)

print(f"\nMean Squared Error (MSE) pada Test Set: {mse:.4f}")

print(f"Root Mean Squared Error (RMSE) pada Test Set: {rmse:.4f}") print(f"Mean Absolute Error (MAE) pada Test Set: {mae:.4f}")

print(f"R-squared (R2 Score) pada Test Set: {r2:.4f}")

Memulai proses fitting GridSearchCV untuk Decision Tree... Proses fitting selesai dalam 1.12 detik.

--- Hasil Terbaik Decision Tree ---

Best R2 Score (Cross-Validation): 0.2528

Best Params: {'feature\_selection': None, 'model max\_depth': 5, 'model min\_samples\_split': 20}

Mean Squared Error (MSE) pada Test Set: 433.4301

Root Mean Squared Error (RMSE) pada Test Set: 20.8190 Mean Absolute Error (MAE) pada Test Set: 15.7799

R-squared (R2 Score) pada Test Set: 0.2668

Scaflerplot Matrix

# Konversi X\_train ke DataFrame dengan nama kolom asli X\_train\_df = pd.DataFrame(X\_train, columns=X.columns)

# Pilih beberapa fitur penting untuk visualisasi (agar tidak terlalu padat) if X\_train\_df.shape[1] > 6:

# Gunakan korelasi untuk pilih fitur terpenting

corr\_matrix = X\_train\_df.corrwith(pd.Series(y\_train.values, index=X\_train\_df.index)).abs() top\_features = corr\_matrix.nlargest(6).index.tolist()

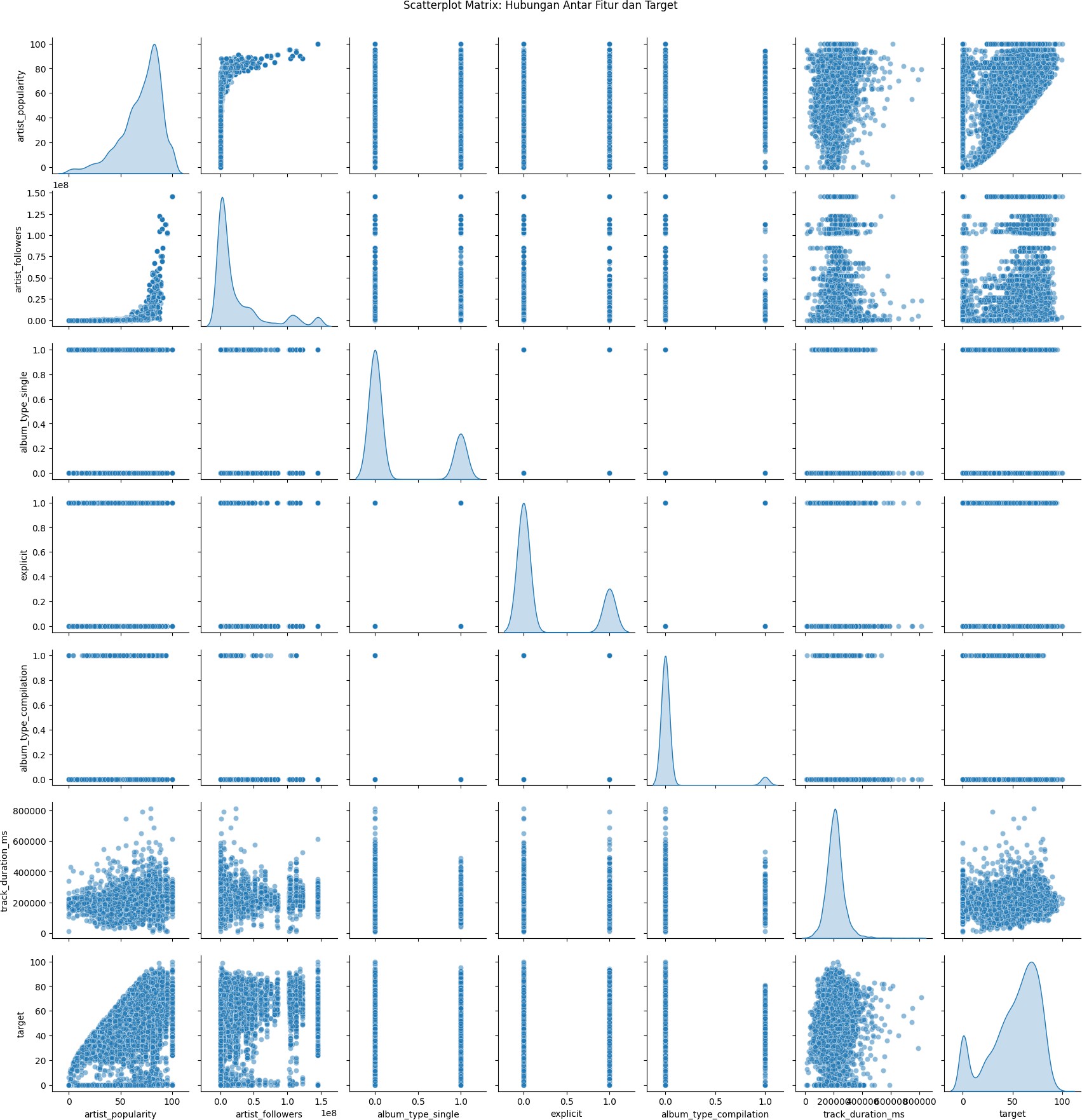
X\_vis = X\_train\_df[top\_features[:6]] else:

X\_vis = X\_train\_df

# Tambahkan target untuk scatter matrix df\_vis = X\_vis.copy()

df\_vis['target'] = y\_train.values

print(f"Fitur: {list(X\_vis.columns)}")



# Buat scatter matrix

sns.pairplot(df\_vis, diag\_kind='kde', plot\_kws={'alpha': 0.5})

plt.suptitle('Scatterplot Matrix: Hubungan Antar Fitur dan Target', y=1.02) plt.show()

Fitur: ['artist\_popularity', 'artist\_followers', 'album\_type\_single', 'explicit', 'album\_type\_compilation', 'track\_duration\_ms

Plot Residual Training dan Test Data Untuk Setiap Model

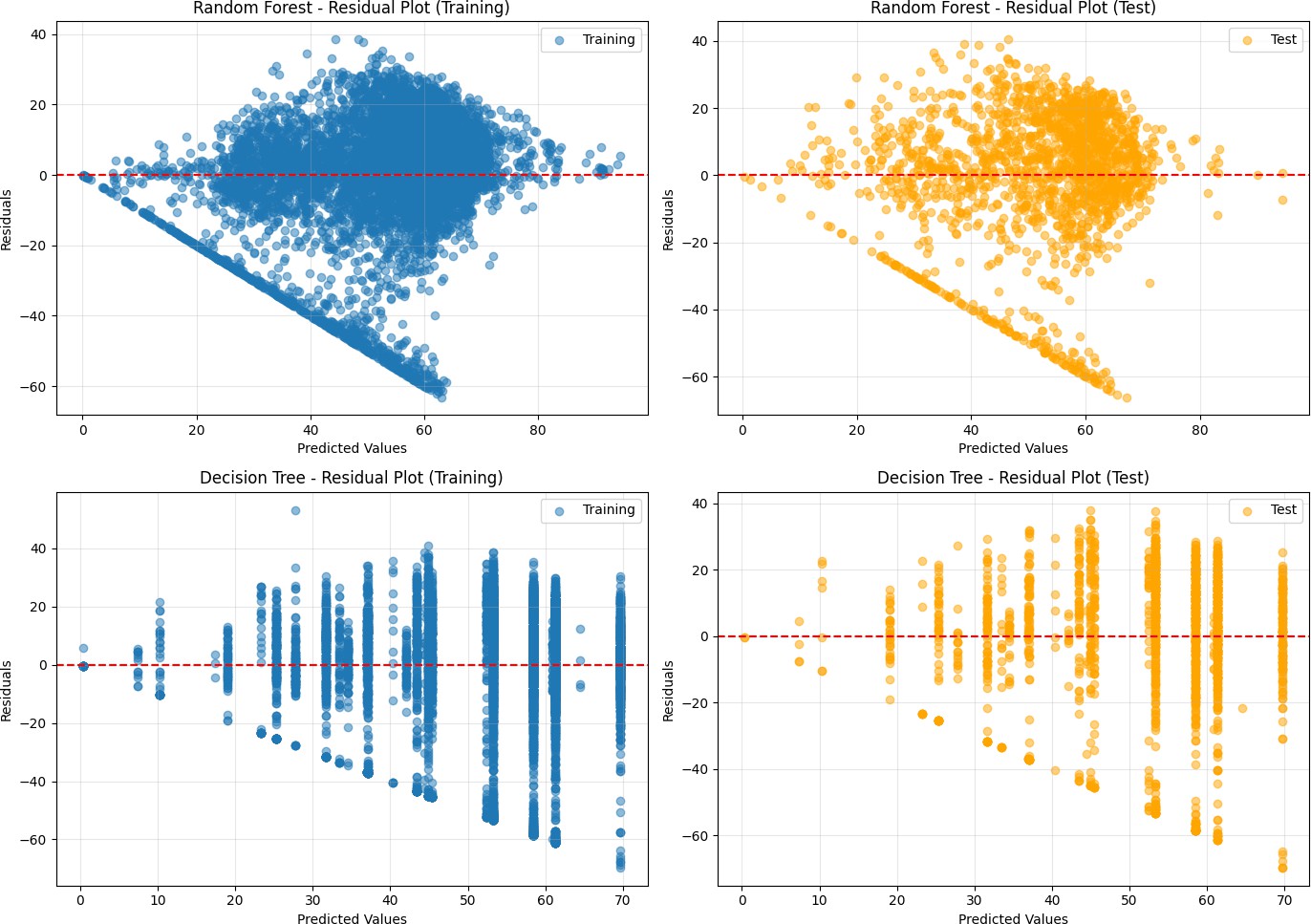
models = {

'Random Forest': best\_rf\_pipeline, 'Decision Tree': best\_dt\_pipeline

}

fig, axes = plt.subplots(2, 2, figsize=(14, 10))

axes = axes.flatten()



for idx, (name, model) in enumerate(models.items()): # Prediksi untuk training dan test

y\_pred\_train = model.predict(X\_train) y\_pred\_test = model.predict(X\_test)

# Residual

residuals\_train = y\_train - y\_pred\_train residuals\_test = y\_test - y\_pred\_test

# Plot residual training

axes[idx\*2].scatter(y\_pred\_train, residuals\_train, alpha=0.5, label='Training') axes[idx\*2].axhline(y=0, color='r', linestyle='--')

axes[idx\*2].set\_xlabel('Predicted Values') axes[idx\*2].set\_ylabel('Residuals')

axes[idx\*2].set\_title(f'{name} - Residual Plot (Training)') axes[idx\*2].legend()

axes[idx\*2].grid(True, alpha=0.3)

# Plot residual test

axes[idx\*2+1].scatter(y\_pred\_test, residuals\_test, alpha=0.5, label='Test', color='orange') axes[idx\*2+1].axhline(y=0, color='r', linestyle='--')

axes[idx\*2+1].set\_xlabel('Predicted Values') axes[idx\*2+1].set\_ylabel('Residuals')

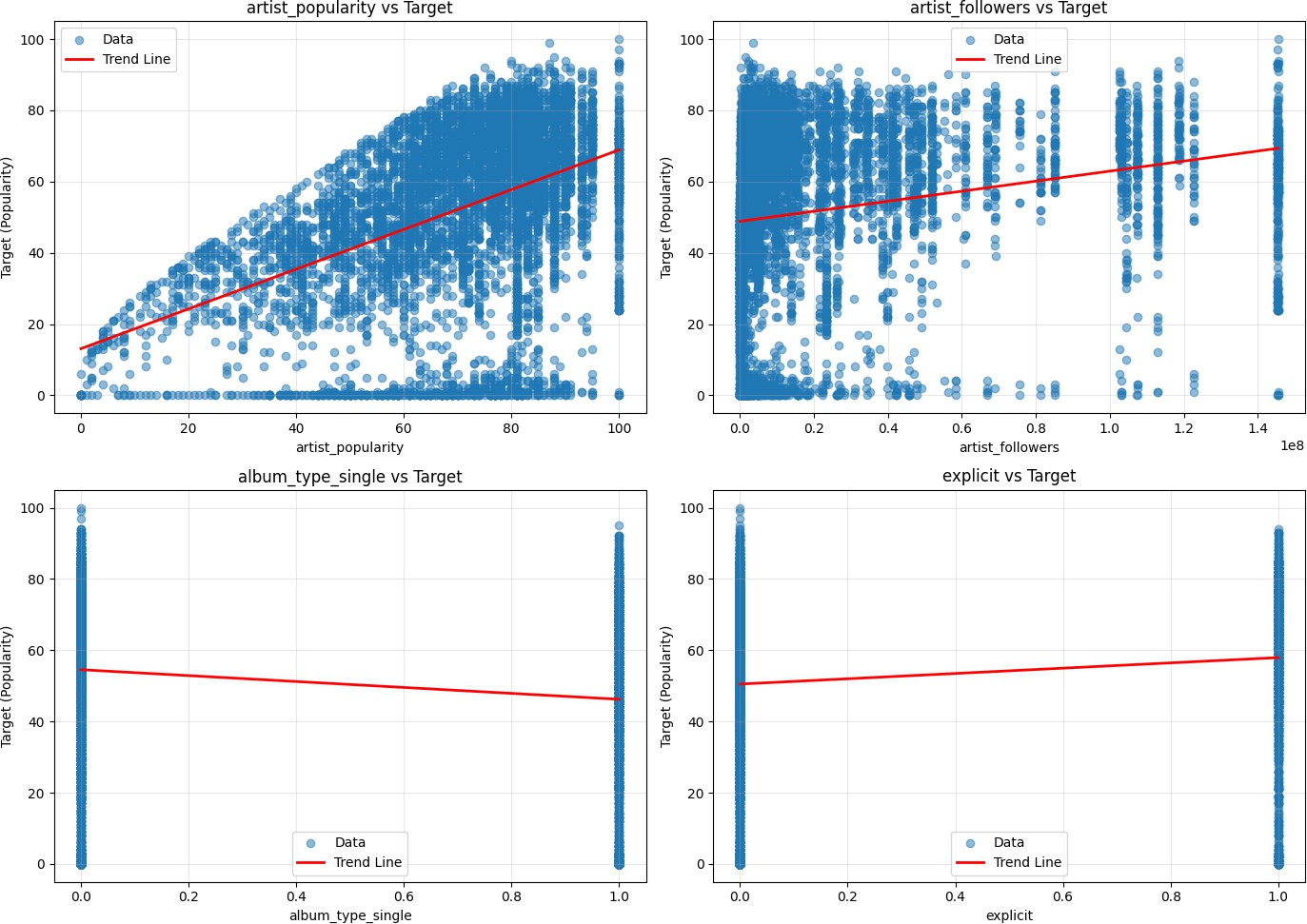
axes[idx\*2+1].set\_title(f'{name} - Residual Plot (Test)') axes[idx\*2+1].legend()

axes[idx\*2+1].grid(True, alpha=0.3)

plt.tight\_layout() plt.show()

Plot antara Fitur dan Target Untuk Setiap Model Pada Training Data

# Konversi X\_train ke DataFrame



X\_train\_df = pd.DataFrame(X\_train, columns=X.columns)

# Hitung korelasi dengan target

correlations = X\_train\_df.corrwith(pd.Series(y\_train.values, index=X\_train\_df.index)).abs() top\_4\_features = correlations.nlargest(4).index.tolist()

print(f"4 fitur dengan korelasi tertinggi: {top\_4\_features}") fig, axes = plt.subplots(2, 2, figsize=(14, 10))

axes = axes.flatten()

for i, feat\_name in enumerate(top\_4\_features):

# Scatter plot - AKSES dengan NAMA kolom, bukan indeks

axes[i].scatter(X\_train\_df[feat\_name], y\_train, alpha=0.5, label='Data')

# Tambahkan garis regresi

x\_vals = X\_train\_df[feat\_name]

# Regresi linear sederhana

coeffs = np.polyfit(x\_vals, y\_train, 1) poly = np.poly1d(coeffs)

x\_range = np.linspace(x\_vals.min(), x\_vals.max(), 100)

axes[i].plot(x\_range, poly(x\_range), color='red', linewidth=2, label='Trend Line')

axes[i].set\_xlabel(feat\_name)

axes[i].set\_ylabel('Target (Popularity)') axes[i].set\_title(f'{feat\_name} vs Target') axes[i].legend()

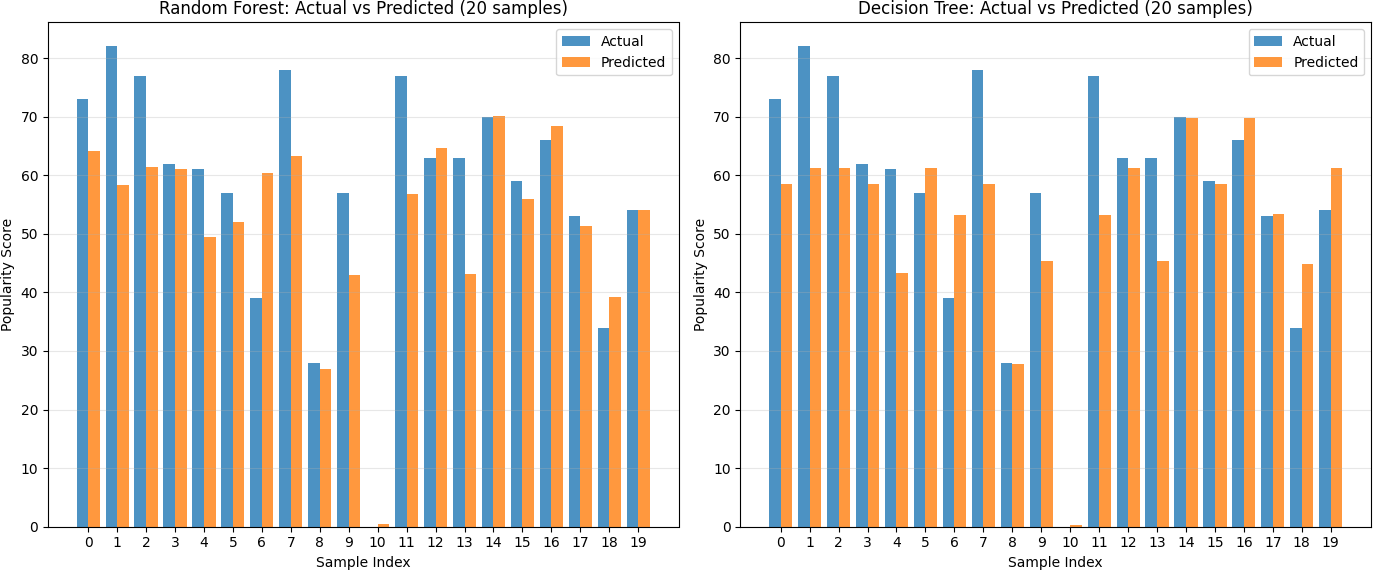
axes[i].grid(True, alpha=0.3)

plt.tight\_layout() plt.show()

4 fitur dengan korelasi tertinggi: ['artist\_popularity', 'artist\_followers', 'album\_type\_single', 'explicit']

Prediksi vs Aktual

fig, axes = plt.subplots(1, 2, figsize=(14, 6)) # Ambil 20 sampel pertama



sample\_indices = range(min(20, len(X\_test)))

for idx, (name, model) in enumerate(models.items()): # Prediksi untuk 20 sampel pertama

X\_sample = X\_test.iloc[sample\_indices] if hasattr(X\_test, 'iloc') else X\_test[sample\_indices]

y\_true\_sample = y\_test.iloc[sample\_indices] if hasattr(y\_test, 'iloc') else y\_test[sample\_indices] y\_pred\_sample = model.predict(X\_sample)

# Buat dataframe untuk plotting df\_compare = pd.DataFrame({

'Actual': y\_true\_sample,

'Predicted': y\_pred\_sample,

'Sample': list(sample\_indices)

})

# Plot bar chart

x\_pos = np.arange(len(df\_compare))

axes[idx].bar(x\_pos - 0.2, df\_compare['Actual'], width=0.4, label='Actual', alpha=0.8)

axes[idx].bar(x\_pos + 0.2, df\_compare['Predicted'], width=0.4, label='Predicted', alpha=0.8)

axes[idx].set\_xlabel('Sample Index')

axes[idx].set\_ylabel('Popularity Score')

axes[idx].set\_title(f'{name}: Actual vs Predicted (20 samples)') axes[idx].set\_xticks(x\_pos)

axes[idx].legend()

axes[idx].grid(True, alpha=0.3, axis='y')

plt.tight\_layout() plt.show()

Mengambil Model Terbaik

best\_rf\_pipeline = grid\_rf.best\_estimator\_ best\_params = grid\_rf.best\_params\_

best\_cv\_score = grid\_rf.best\_score\_

# Prediksi pada Test Set

y\_pred\_best = best\_rf\_pipeline.predict(X\_test) # Hitung Metrik Evaluasi Akhir

mse = mean\_squared\_error(y\_test, y\_pred\_best) rmse = np.sqrt(mse)

mae = mean\_absolute\_error(y\_test, y\_pred\_best) r2 = r2\_score(y\_test, y\_pred\_best)

print(f"Algoritma: Random Forest (dalam Pipeline)") print(f"Parameter Terbaik: {best\_params}")

print(f"R2 Score (Cross-Validation Terbaik): {best\_cv\_score:.4f}") print("-" \* 50)

print("Metrik Evaluasi pada TEST SET:") print(f"R-squared (R2): {r2:.4f}")

print(f"Root Mean Squared Error (RMSE): {rmse:.4f}") print(f"Mean Squared Error (MSE): {mse:.4f}")

print(f"Mean Absolute Error (MAE): {mae:.4f}")

Algoritma: Random Forest (dalam Pipeline)

Parameter Terbaik: {'feature\_selection': None, 'model max\_depth': 10, 'model min\_samples\_split': 5, 'model n\_estimators': 200 R2 Score (Cross-Validation Terbaik): 0.3209

Metrik Evaluasi pada TEST SET: R-squared (R2): 0.3479

Root Mean Squared Error (RMSE): 19.6337

Mean Squared Error (MSE): 385.4812 Mean Absolute Error (MAE): 14.4914