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

```
# 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:
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             0
dtype: int64
```

Cek dan Hapus Duplikat

```
# 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()
```

```
=== DATA SIAP TRAINING ===
Dimensi akhir: (8573, 10)
```

| | 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

```
# 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]
    }
]
```

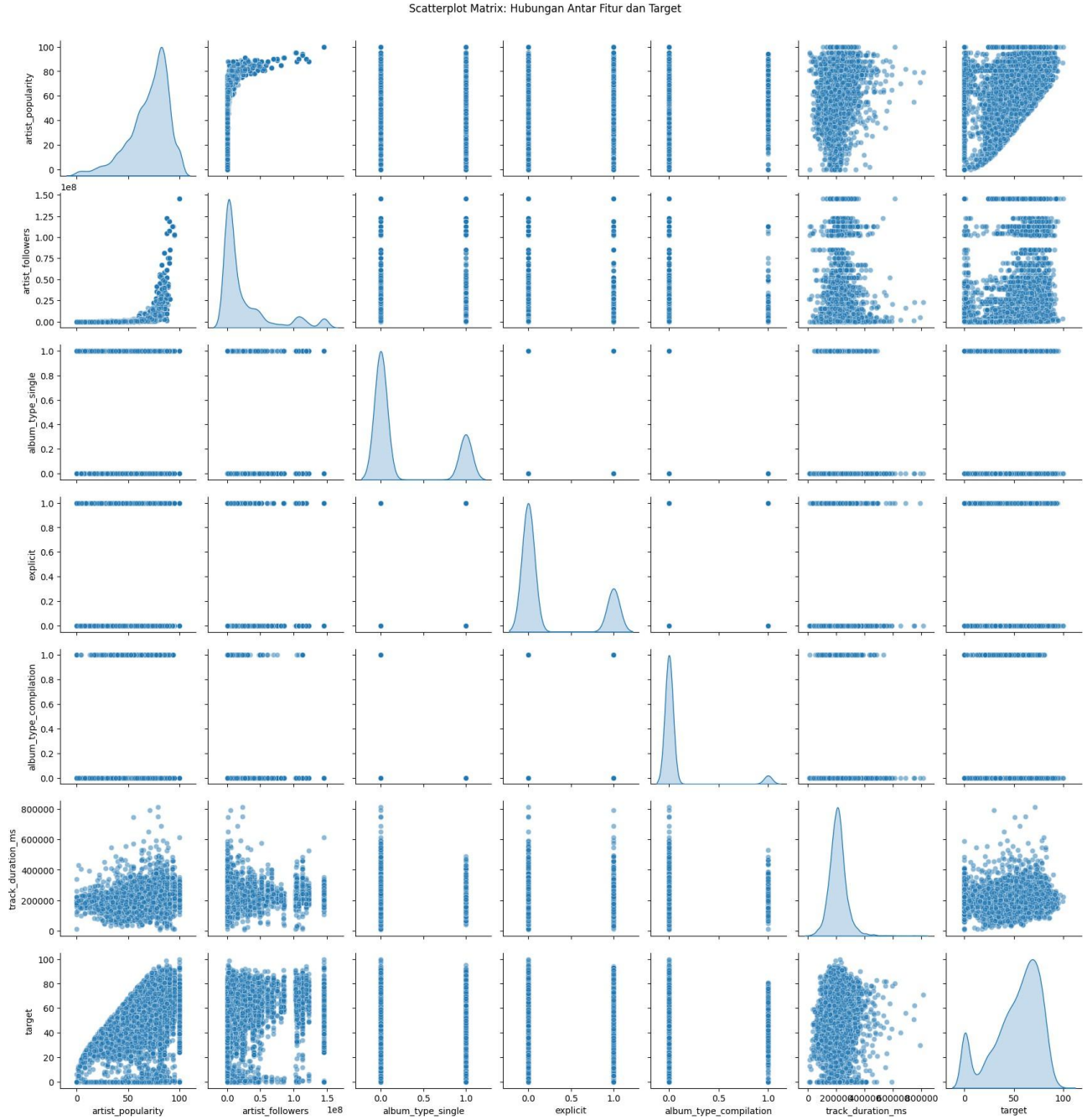
```
# Inisialisasi GridSearch
grid_lasso = GridSearchCV(
    estimator=pipe_lasso,
    param_grid=param_grid_lasso,
    cv=5,                  # 5-fold cross-validation
    scoring='r2',          # Gunakan R-squared untuk membandingkan model
    n_jobs=-1              # Menggunakan semua core CPU untuk 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()
```

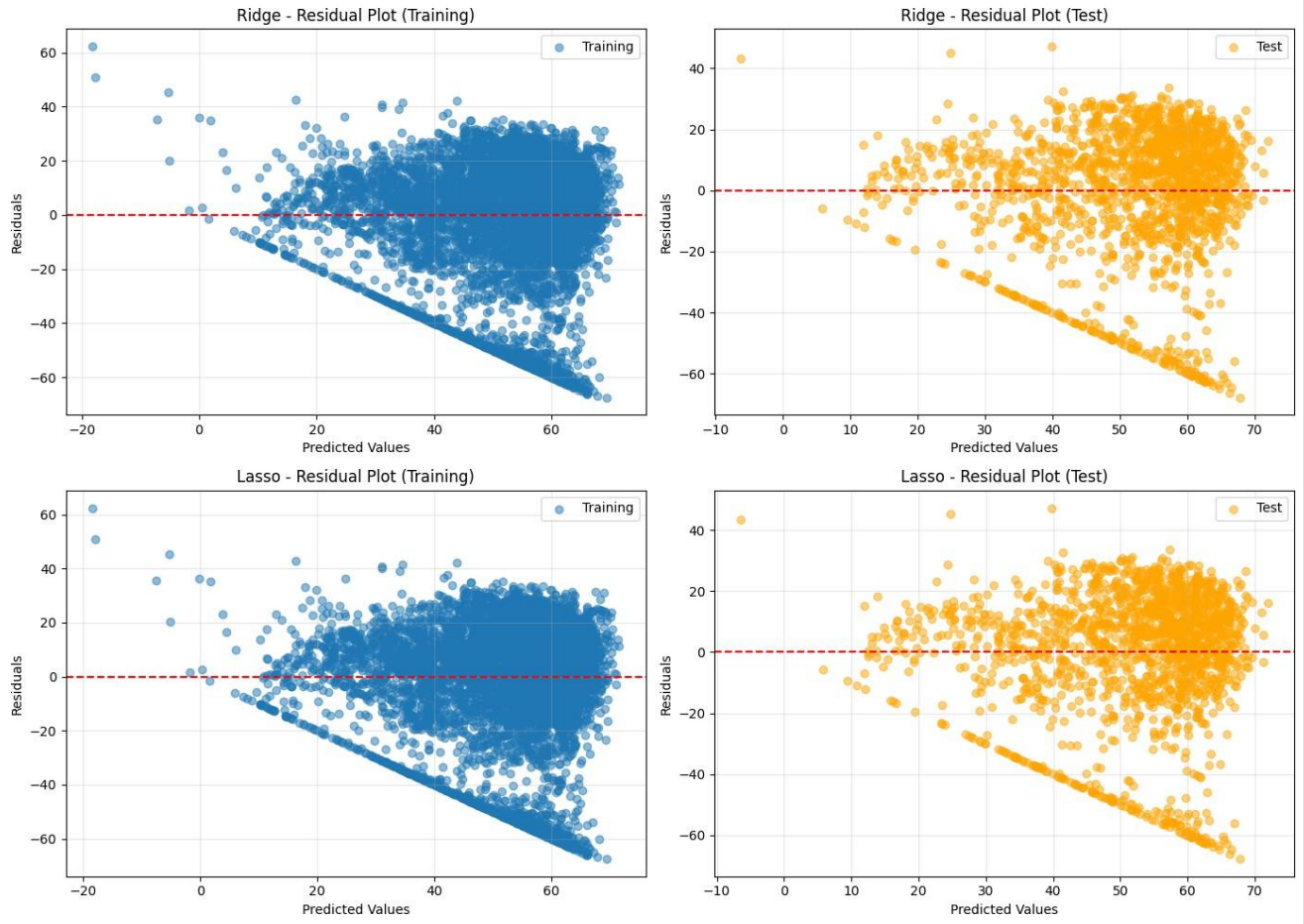
```
Fitur: ['artist_popularity', 'artist_followers', 'album_type_single', 'explicit', 'album_type_compilation', 'track_duration_ms
```




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()
```

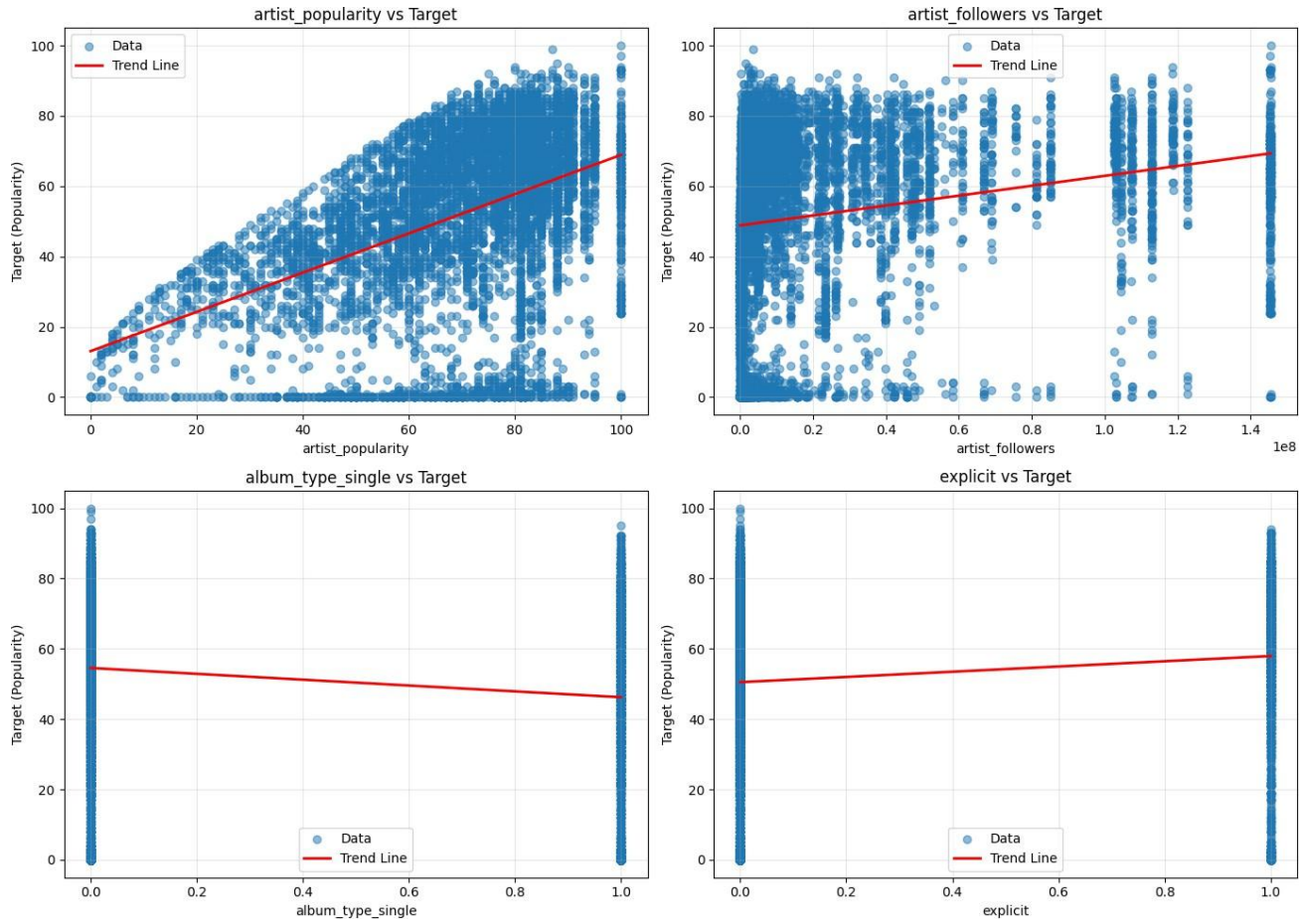



Plot antara Fitur dan Target Untuk Setiap Model Pada Training Data

```
# 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()
```

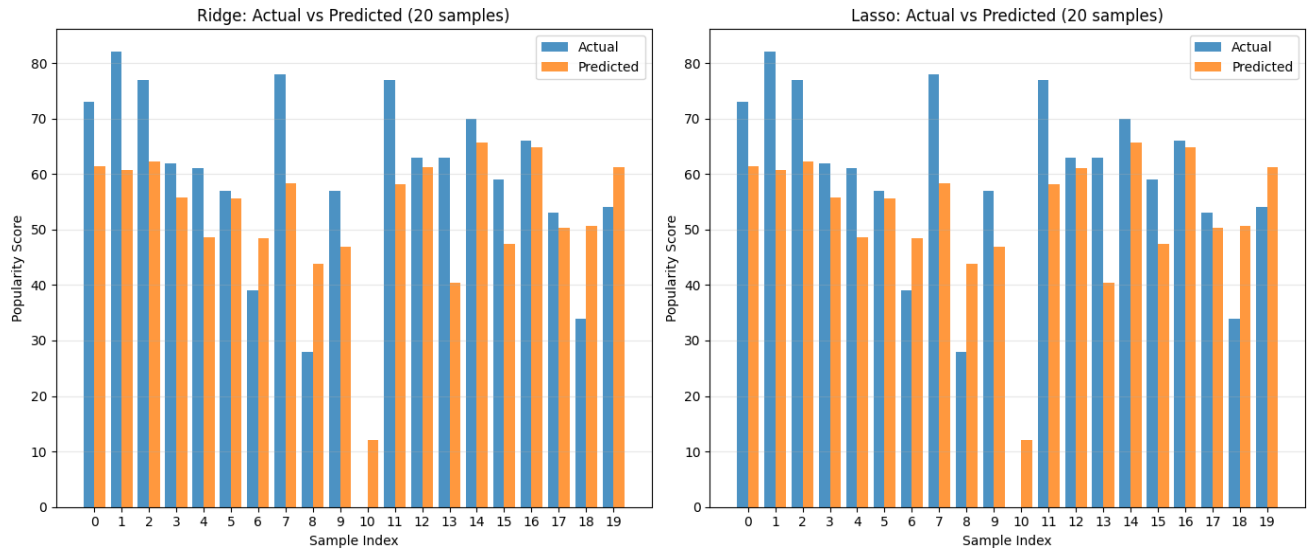
```
4 fitur dengan korelasi tertinggi: ['artist_popularity', 'artist_followers', 'album_type_single', 'explicit']
```




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

```
# 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:
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             0
dtype: int64
```

Cek dan Hapus Duplikat

```
# 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()
```

```
=== DATA SIAP TRAINING ===
Dimensi akhir: (8573, 10)
```

| | 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

```
# 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

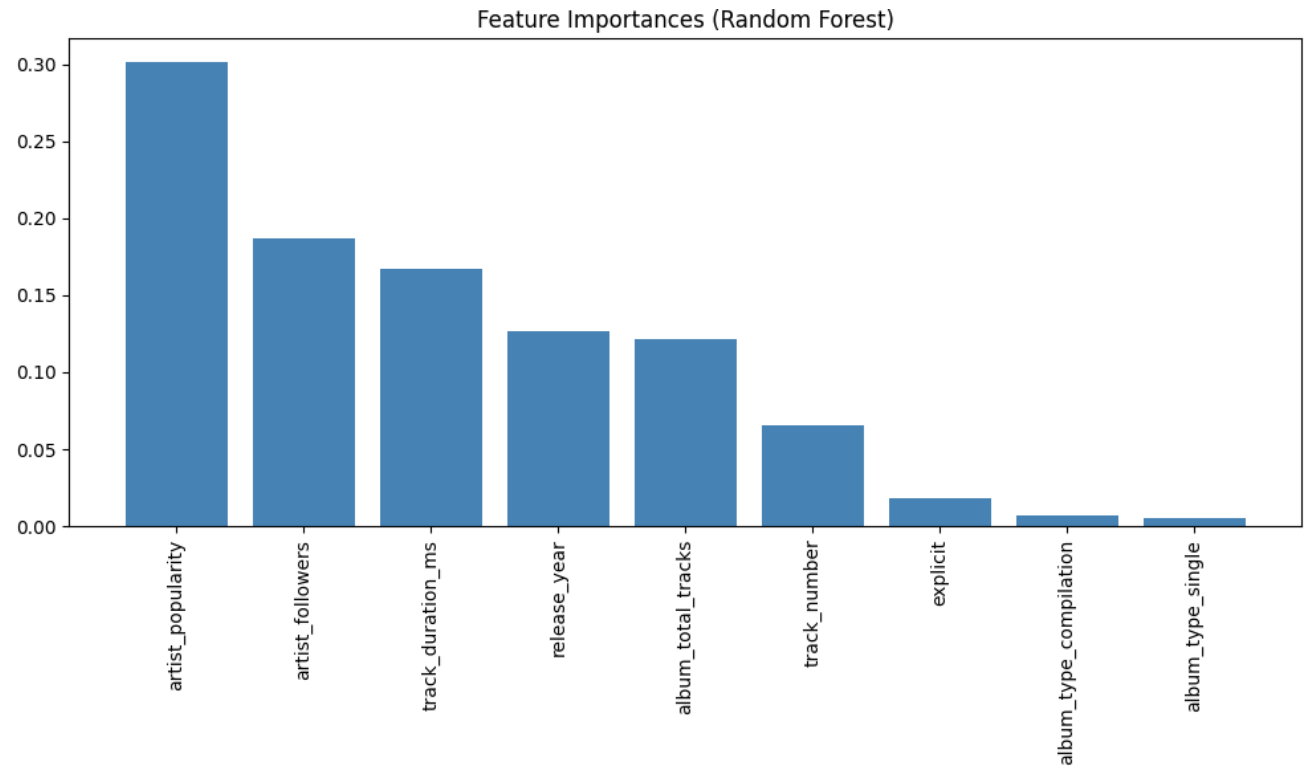
```
print("=== HASIL DENGAN MINMAX SCALER ===")

# 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

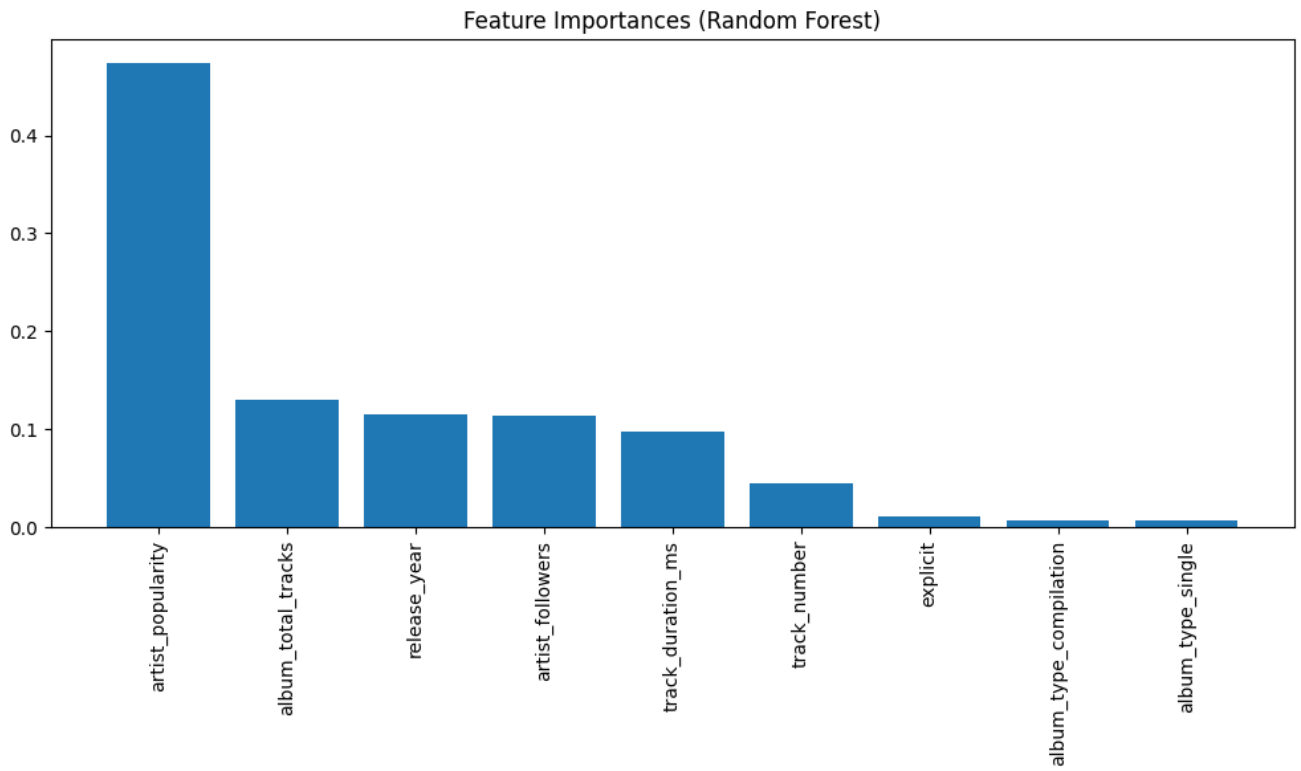
```
print("=== PIPELINE RANDOM FOREST REGRESSION ===")

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,                 # Gunakan 5-fold cross-validation
    scoring='r2',         # Gunakan R-squared untuk regresi
    n_jobs=-1             # 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}")
```

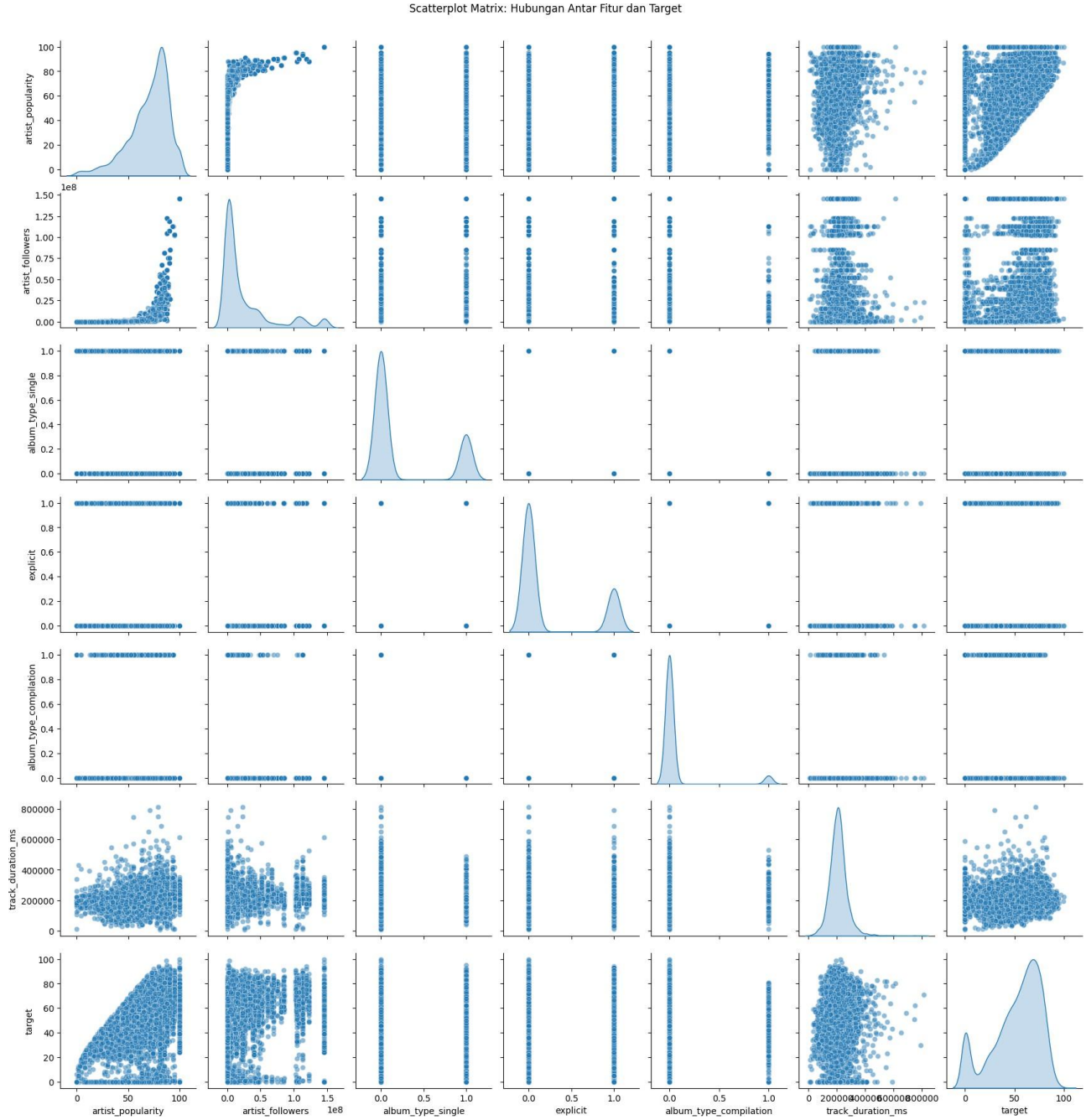
```
Memulai proses fitting GridSearchCV untuk Decision Tree...
Proses fitting selesai dalam 1.12 detik.

--- 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()
```

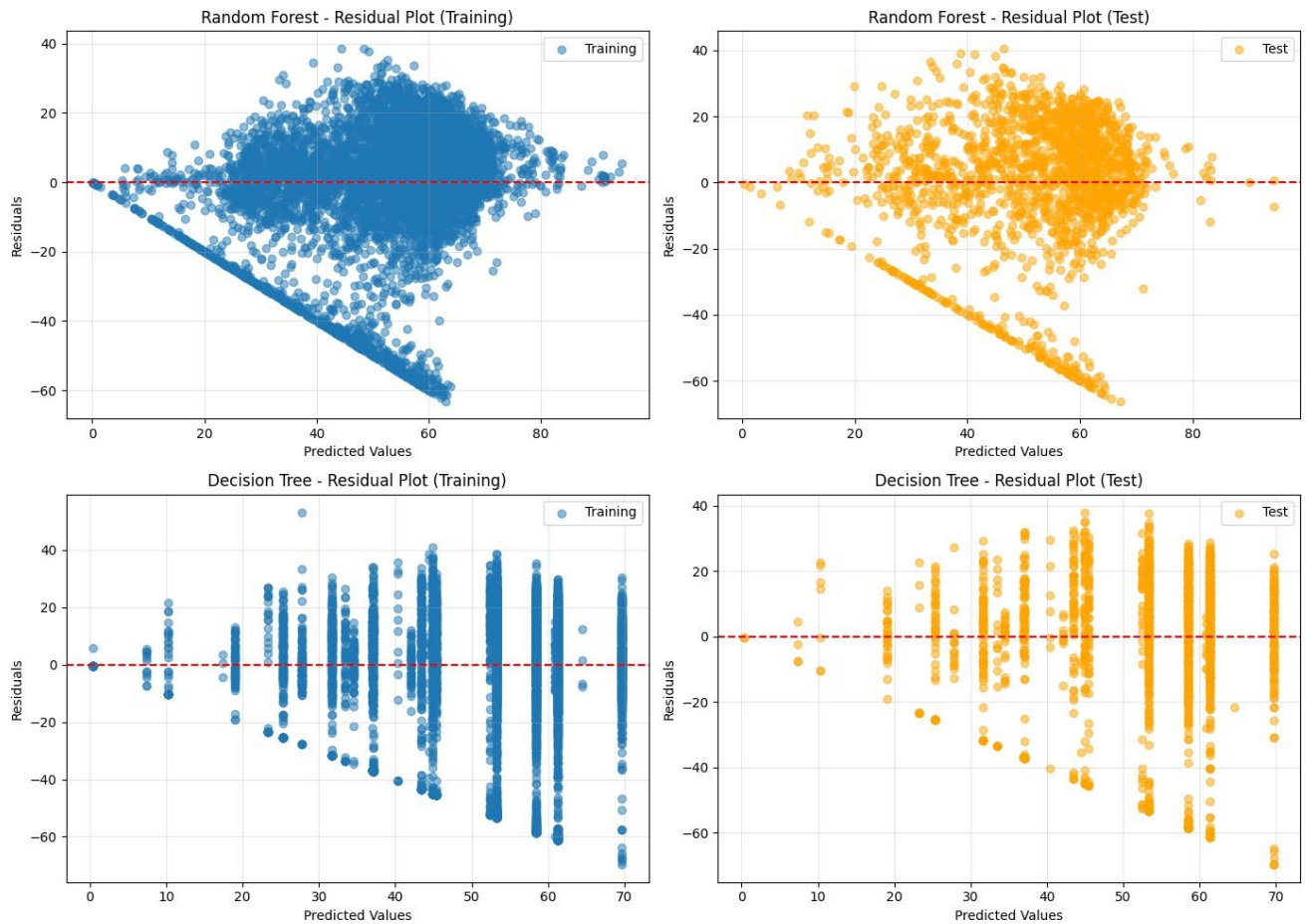
```
Fitur: ['artist_popularity', 'artist_followers', 'album_type_single', 'explicit', 'album_type_compilation', 'track_duration_ms
```




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))
```

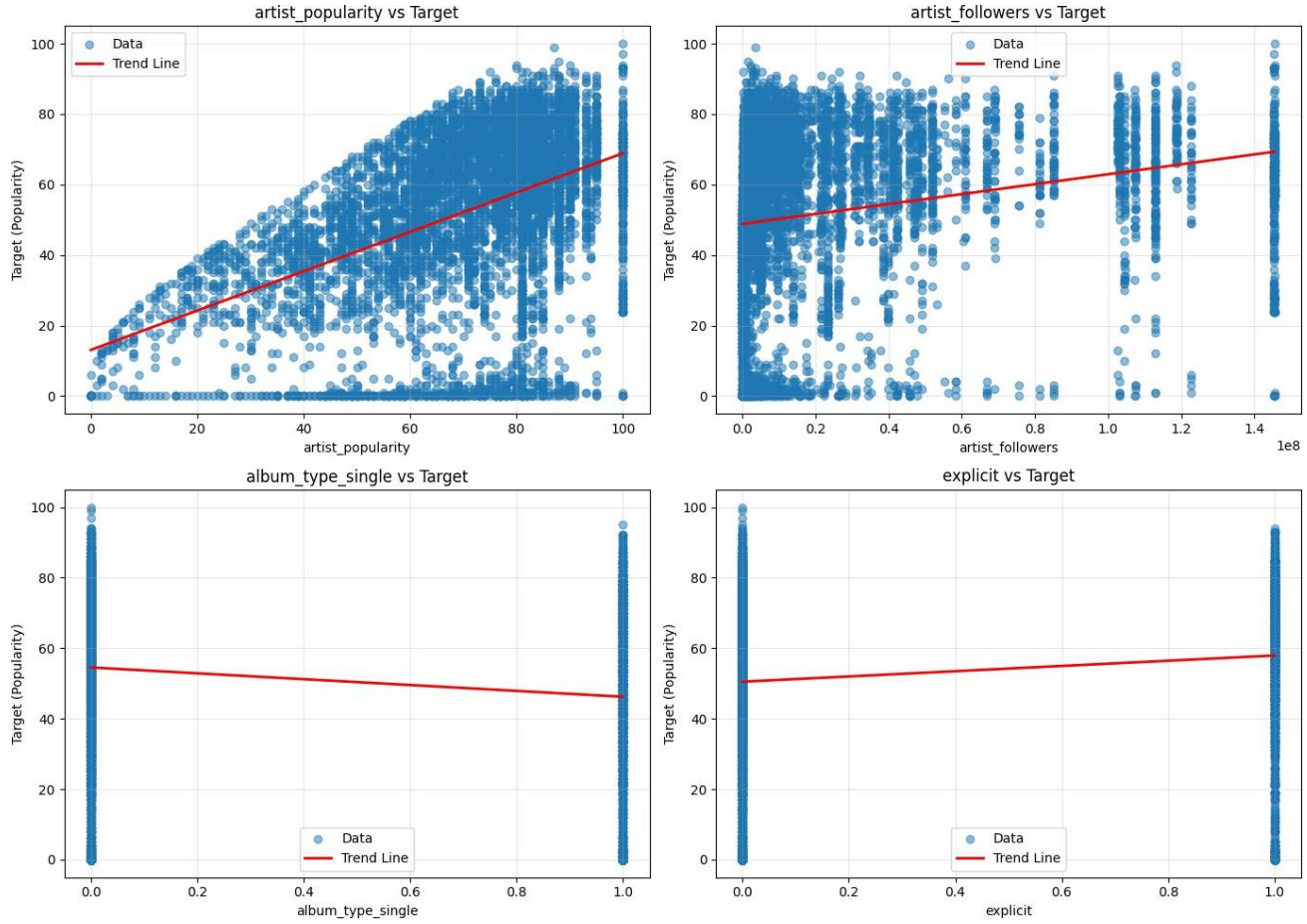
```
axes = axes.flatten()

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

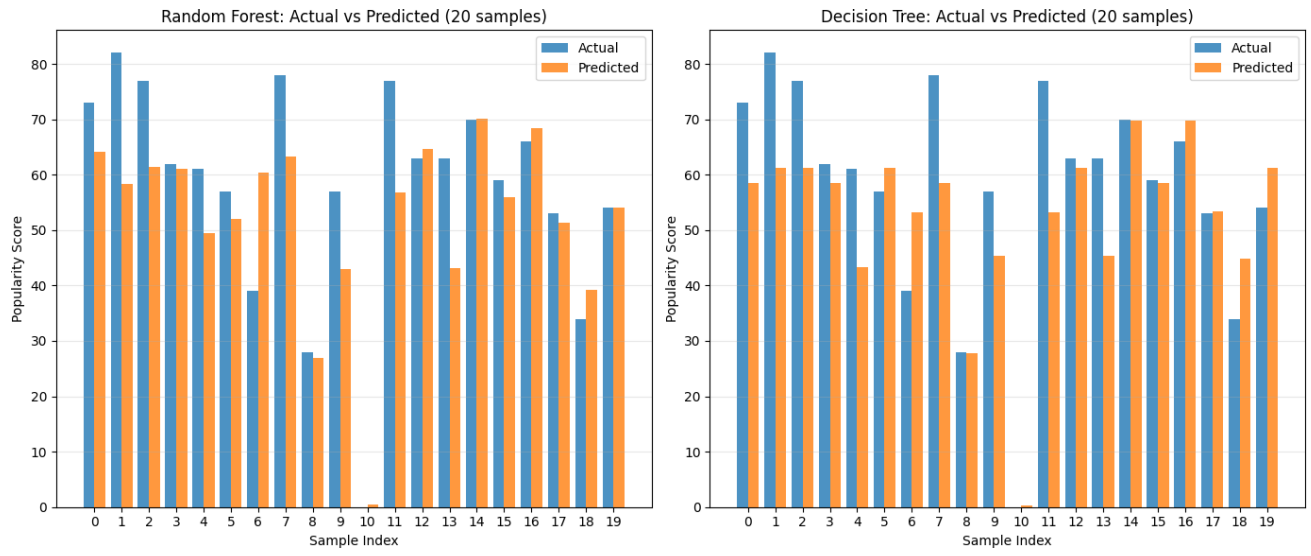
```
# Konversi X_train ke DataFrame
X_train_df = pd.DataFrame(X_train, columns=X.columns)

# 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
```

```
import streamlit as st
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.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, f_regression
from sklearn.model_selection import GridSearchCV
import time

# Konfigurasi Halaman
st.set_page_config(
    page_title="Spotify Track Popularity Predictor",
    page_icon="🎵",
    layout="wide"
)

# CSS Custom
st.markdown("""
<style>
    .main-header {
        font-size: 2.5rem;
        color: #1DB954;
        text-align: center;
        font-weight: bold;
        margin-bottom: 1rem;
    }
    .sub-header {
        font-size: 1.2rem;
        color: #666;
        text-align: center;
        margin-bottom: 2rem;
    }
    .metric-card {
        background-color: #f0f2f6;
        padding: 1rem;
        border-radius: 0.5rem;
        border-left: 4px solid #1DB954;
    }
</style>
""", unsafe_allow_html=True)

# Header
st.markdown('<p class="main-header">🎵 Spotify Track Popularity Predictor</p>',
unsafe_allow_html=True)
st.markdown('<p class="sub-header">Prediksi popularitas lagu menggunakan Machine
Learning</p>', unsafe_allow_html=True)
```

```
# Sidebar
st.sidebar.title("⚙️ Pengaturan")
st.sidebar.markdown("---")

# Fungsi untuk memuat dan memproses data
@st.cache_data
def load_and_process_data(file):
    df = pd.read_csv(file)

    # Hapus kolom tidak relevan
    cols_to_drop = ['track_id', 'track_name', 'artist_name',
                    'artist_genres', 'album_id', 'album_name']
    df = df.drop(columns=[col for col in cols_to_drop if col in df.columns])

    # Hapus missing values
    df = df.dropna()

    # Ubah Boolean ke int
    if 'explicit' in df.columns:
        df['explicit'] = df['explicit'].astype(int)

    # Extract tahun rilis
    if 'album_release_date' in df.columns:
        df['release_year'] = pd.to_datetime(df['album_release_date'],
errors='coerce').dt.year
        df = df.drop(columns=['album_release_date'])

    # Drop NA hasil konversi
    df = df.dropna()

    # One-Hot Encoding
    if 'album_type' in df.columns:
        df = pd.get_dummies(df, columns=['album_type'], drop_first=True)

    return df

# Fungsi untuk melatih model
@st.cache_resource
def train_models(X_train, y_train, X_test, y_test):
    results = {}

    # Random Forest Pipeline
    pipe_rf = Pipeline([
        ('feature_selection', SelectKBest(score_func=f_regression)),
        ('model', RandomForestRegressor(random_state=28))
    ])

    param_grid_rf = [
        {
            'feature_selection': [None],
            'model__n_estimators': [50, 100, 200],
```

```
            'model__max_depth': [5, 10, None],
            'model__min_samples_split': [2, 5]
        },
        {
            'feature_selection': [SelectKBest(score_func=f_regression)],
            'feature_selection__k': [10, 20, 30],
            'model__n_estimators': [100],
            'model__max_depth': [10, 20],
        }
    ]

    with st.spinner('🔄 Melatih Random Forest Model...'):
        grid_rf = GridSearchCV(pipe_rf, param_grid_rf, cv=5, scoring='r2', n_jobs=-1)
        grid_rf.fit(X_train, y_train)
        y_pred_rf = grid_rf.best_estimator_.predict(X_test)

        results['Random Forest'] = {
            'model': grid_rf.best_estimator_,
            'best_params': grid_rf.best_params_,
            'cv_score': grid_rf.best_score_,
            'predictions': y_pred_rf,
            'mse': mean_squared_error(y_test, y_pred_rf),
            'rmse': np.sqrt(mean_squared_error(y_test, y_pred_rf)),
            'mae': mean_absolute_error(y_test, y_pred_rf),
            'r2': r2_score(y_test, y_pred_rf)
        }

    # Decision Tree Pipeline
    pipe_dt = Pipeline([
        ('feature_selection', SelectKBest(score_func=f_regression)),
        ('model', DecisionTreeRegressor(random_state=28))
    ])

    param_grid_dt = [
        {
            'feature_selection': [None],
            'model__max_depth': [3, 5, 8, 12, None],
            'model__min_samples_split': [2, 10, 20]
        },
        {
            'feature_selection': [SelectKBest(score_func=f_regression)],
            'feature_selection__k': [10, 20, 30],
            'model__max_depth': [10, 20],
            'model__min_samples_split': [2, 10]
        }
    ]

    with st.spinner('🔄 Melatih Decision Tree Model...'):
        grid_dt = GridSearchCV(pipe_dt, param_grid_dt, cv=5, scoring='r2', n_jobs=-1)
        grid_dt.fit(X_train, y_train)
        y_pred_dt = grid_dt.best_estimator_.predict(X_test)
```

```
        results['Decision Tree'] = {
            'model': grid_dt.best_estimator_,
            'best_params': grid_dt.best_params_,
            'cv_score': grid_dt.best_score_,
            'predictions': y_pred_dt,
            'mse': mean_squared_error(y_test, y_pred_dt),
            'rmse': np.sqrt(mean_squared_error(y_test, y_pred_dt)),
            'mae': mean_absolute_error(y_test, y_pred_dt),
            'r2': r2_score(y_test, y_pred_dt)
        }

    return results

# Upload File
uploaded_file = st.sidebar.file_uploader("📁 Upload CSV Dataset", type=['csv'])

if uploaded_file is not None:
    # Load dan Process Data
    with st.spinner('🔄 Memproses data...'):
        df_spotify = load_and_process_data(uploaded_file)

    st.success(f"✅ Data berhasil dimuat! Shape: {df_spotify.shape}")

    # Tab Layout
    tab1, tab2, tab3, tab4, tab5 = st.tabs([
        "📊 Eksplorasi Data",
        "🤖 Model Training",
        "📈 Evaluasi Model",
        "🔮 Prediksi",
        "📉 Visualisasi"
    ])

    # TAB 1: Eksplorasi Data
    with tab1:
        st.header("📊 Eksplorasi Data")

        col1, col2, col3 = st.columns(3)
        with col1:
            st.metric("Total Baris", df_spotify.shape[0])
        with col2:
            st.metric("Total Kolom", df_spotify.shape[1])
        with col3:
            st.metric("Missing Values", df_spotify.isnull().sum().sum())

        st.subheader("Preview Data")
        st.dataframe(df_spotify.head(10), use_container_width=True)

        st.subheader("Statistik Deskriptif")
        st.dataframe(df_spotify.describe(), use_container_width=True)

        st.subheader("Distribusi Target (Track Popularity)")
        fig, ax = plt.subplots(figsize=(10, 4))
```

```
        ax.hist(df_spotify['track_popularity'], bins=50, color='#1DB954',
edgecolor='black')
        ax.set_xlabel('Popularity Score')
        ax.set_ylabel('Frequency')
        ax.set_title('Distribusi Track Popularity')
        st.pyplot(fig)

    # TAB 2: Model Training
    with tab2:
        st.header("🤖 Model Training")

        # Split Data
        target = 'track_popularity'
        X = df_spotify.drop(columns=[target])
        y = df_spotify[target]

        test_size = st.sidebar.slider("Test Size (%)", 10, 40, 20) / 100

        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=test_size, random_state=28
        )

        col1, col2 = st.columns(2)
        with col1:
            st.info(f"📂 Training Set: {X_train.shape[0]} samples")
        with col2:
            st.info(f"🩹 Test Set: {X_test.shape[0]} samples")

        if st.button("🚀 Mulai Training", type="primary"):
            start_time = time.time()
            results = train_models(X_train, y_train, X_test, y_test)
            end_time = time.time()

            st.success(f"☑️ Training selesai dalam {end_time - start_time:.2f} detik!")

            # Simpan hasil ke session state
            st.session_state['results'] = results
            st.session_state['X_test'] = X_test
            st.session_state['y_test'] = y_test
            st.session_state['X_train'] = X_train
            st.session_state['y_train'] = y_train

    # TAB 3: Evaluasi Model
    with tab3:
        st.header("📈 Evaluasi Model")

        if 'results' in st.session_state:
            results = st.session_state['results']

            # Perbandingan Model
            st.subheader("Perbandingan Performa Model")
```

```
            comparison_data = []
            for model_name, metrics in results.items():
                comparison_data.append({
                    'Model': model_name,
                    'R² Score': f"{metrics['r2']:.4f}",
                    'RMSE': f"{metrics['rmse']:.4f}",
                    'MAE': f"{metrics['mae']:.4f}",
                    'CV Score': f"{metrics['cv_score']:.4f}"
                })

            df_comparison = pd.DataFrame(comparison_data)
            st.dataframe(df_comparison, use_container_width=True)

            # Detail per Model
            for model_name, metrics in results.items():
                with st.expander(f"📊 Detail {model_name}"):
                    col1, col2, col3, col4 = st.columns(4)

                    with col1:
                        st.metric("R² Score", f"{metrics['r2']:.4f}")
                    with col2:
                        st.metric("RMSE", f"{metrics['rmse']:.4f}")
                    with col3:
                        st.metric("MAE", f"{metrics['mae']:.4f}")
                    with col4:
                        st.metric("CV Score", f"{metrics['cv_score']:.4f}")

                    st.write("**Best Parameters:**")
                    st.json(metrics['best_params'])
        else:
            st.warning("⚠️ Silakan lakukan training terlebih dahulu di tab 'Model Training'")

    # TAB 4: Prediksi
    with tab4:
        st.header("🔮 Prediksi Popularitas Lagu")

        if 'results' in st.session_state:
            results = st.session_state['results']
            X_test = st.session_state['X_test']
            y_test = st.session_state['y_test']

            model_choice = st.selectbox("Pilih Model", list(results.keys()))

            st.subheader("Prediksi vs Aktual (20 Sampel Pertama)")

            model = results[model_choice]['model']
            y_pred_sample = model.predict(X_test[:20])
            y_true_sample = y_test[:20].values if hasattr(y_test[:20], 'values') else y_test[:20]

            # Plot
```

```
            fig, ax = plt.subplots(figsize=(12, 6))
            x_pos = np.arange(20)
            ax.bar(x_pos - 0.2, y_true_sample, width=0.4, label='Actual', alpha=0.8,
color='#1DB954')
            ax.bar(x_pos + 0.2, y_pred_sample, width=0.4, label='Predicted', alpha=0.8,
color='#FF6B6B')
            ax.set_xlabel('Sample Index')
            ax.set_ylabel('Popularity Score')
            ax.set_title(f'{model_choice}: Actual vs Predicted')
            ax.legend()
            ax.grid(True, alpha=0.3, axis='y')
            st.pyplot(fig)

            # Tabel Perbandingan
            df_pred = pd.DataFrame({
                'Sample': range(20),
                'Actual': y_true_sample,
                'Predicted': y_pred_sample,
                'Error': np.abs(y_true_sample - y_pred_sample)
            })
            st.dataframe(df_pred, use_container_width=True)
        else:
            st.warning("⚠ Silakan lakukan training terlebih dahulu di tab 'Model
Training'")

    # TAB 5: Visualisasi
    with tab5:
        st.header("📉 Visualisasi Analisis")

        if 'results' in st.session_state:
            results = st.session_state['results']
            X_test = st.session_state['X_test']
            y_test = st.session_state['y_test']

            # Residual Plots
            st.subheader("Residual Plots")

            fig, axes = plt.subplots(1, 2, figsize=(14, 5))

            for idx, (name, metrics) in enumerate(results.items()):
                model = metrics['model']
                y_pred = model.predict(X_test)
                residuals = y_test.values - y_pred if hasattr(y_test, 'values') else
y_test - y_pred

                axes[idx].scatter(y_pred, residuals, alpha=0.5, color='#1DB954')
                axes[idx].axhline(y=0, color='red', linestyle='--', linewidth=2)
                axes[idx].set_xlabel('Predicted Values')
                axes[idx].set_ylabel('Residuals')
                axes[idx].set_title(f'{name} - Residual Plot')
                axes[idx].grid(True, alpha=0.3)
```

```
            plt.tight_layout()
            st.pyplot(fig)

            # Feature Importance (Random Forest)
            if 'Random Forest' in results:
                st.subheader("Feature Importance (Random Forest)")

                rf_model = results['Random Forest']['model']
                # Cek apakah ada feature selection
                if hasattr(rf_model.named_steps['feature_selection'], 'get_support'):
                    # Ada feature selection
                    selected_features_mask =
rf_model.named_steps['feature_selection'].get_support()
                    selected_features = X_test.columns[selected_features_mask]
                    importances = rf_model.named_steps['model'].feature_importances_
                else:
                    # Tidak ada feature selection
                    selected_features = X_test.columns
                    importances = rf_model.named_steps['model'].feature_importances_

                # Sort by importance
                indices = np.argsort(importances)[::-1][:10]  # Top 10

                fig, ax = plt.subplots(figsize=(10, 6))
                ax.barh(range(len(indices)), importances[indices], color='#1DB954')
                ax.set_yticks(range(len(indices)))
                ax.set_yticklabels([selected_features[i] for i in indices])
                ax.set_xlabel('Feature Importance')
                ax.set_title('Top 10 Feature Importance')
                ax.invert_yaxis()
                plt.tight_layout()
                st.pyplot(fig)
        else:
            st.warning("⚠ Silakan lakukan training terlebih dahulu di tab 'Model
Training'")

else:
    st.info("👆 Silakan upload file CSV dataset Spotify di sidebar untuk memulai")

    st.markdown("""
    ### 📝 Panduan Penggunaan:
    1. **Upload Dataset**: Upload file CSV dataset Spotify di sidebar
    2. **Eksplorasi Data**: Lihat statistik dan distribusi data
    3. **Model Training**: Klik tombol 'Mulai Training' untuk melatih model
    4. **Evaluasi Model**: Lihat perbandingan performa model
    5. **Prediksi**: Lihat hasil prediksi vs nilai aktual
    6. **Visualisasi**: Analisis residual dan feature importance

    ### 📊 Dataset yang Dibutuhkan:
    File CSV dengan kolom-kolom seperti:
    - `track_popularity` (target)
    - Audio features (danceability, energy, loudness, dll.)
```

```
    - Metadata (explicit, album_release_date, album_type, dll.)
    """)

# Footer
st.sidebar.markdown("---")
st.sidebar.markdown("### 📌 Info")
st.sidebar.info("""
**Spotify Popularity Predictor**
Developed with 💚 using Streamlit
Models: Random Forest & Decision Tree
""")
```