

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	2IWc1iJlz2NVcStV5fbtPG	Clouds	1	67	158760	False	BUNT.	
2	1msEuwSBneBKpVCZQcFTsU	Forever & Always (Taylor's Version)	11	63	225328	False	Taylor Swift	
3	7bcy34fBT2ap1L4bfPsI9q	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

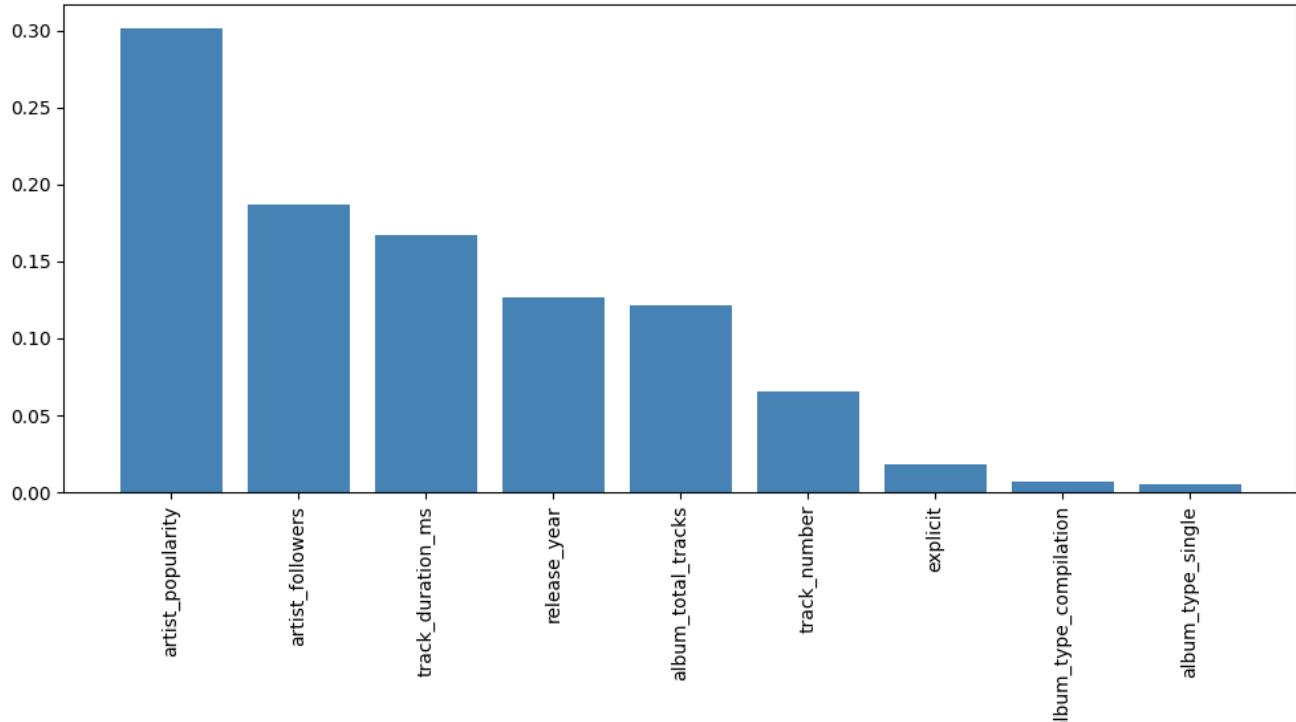
```

# 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()

```

Feature Importances (Random Forest)



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': 5}
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': 10}
RMSE: 19.67199226438753
R2: 0.3453660288160846
```

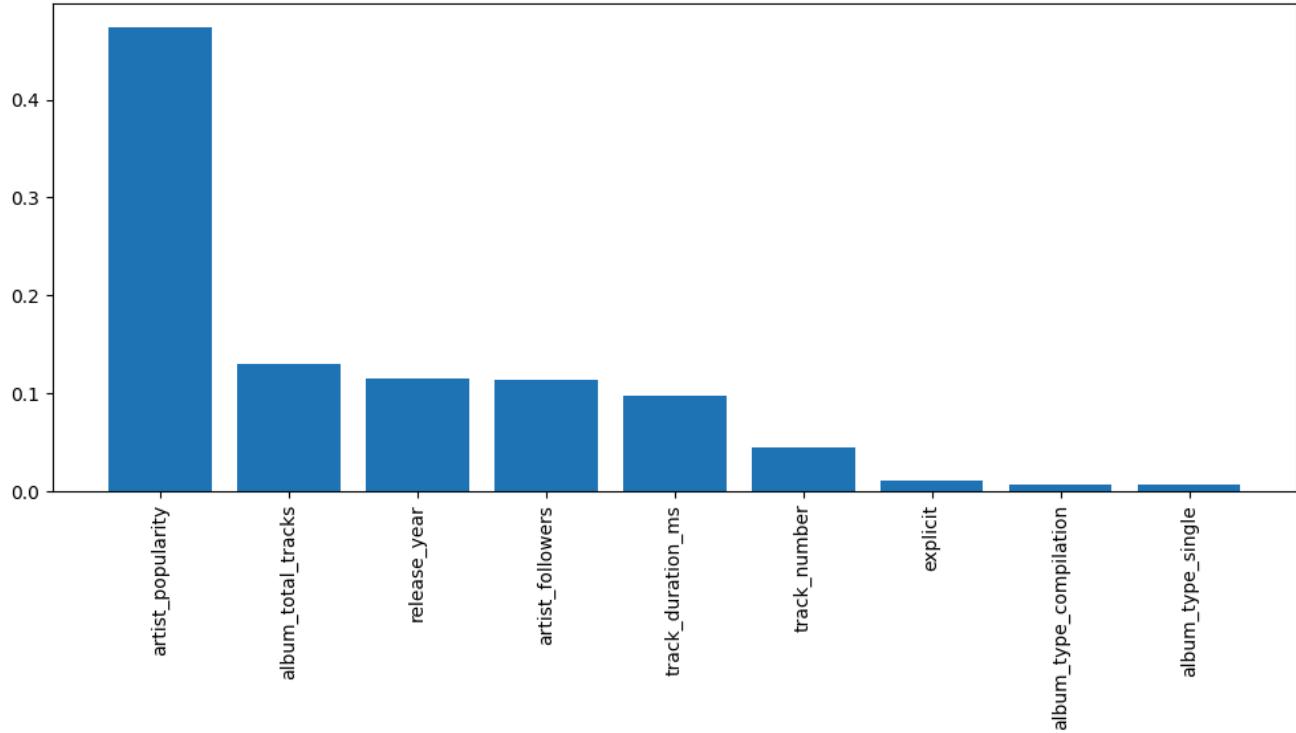
Model Terbaik

```
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()
```

Feature Importances (Random Forest)



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
```

Scatterplot Matrix

```
# Konversi X_train ke DataFrame dengan nama kolom asli
X_train_df = pd.DataFrame(X_train, columns=X.columns)

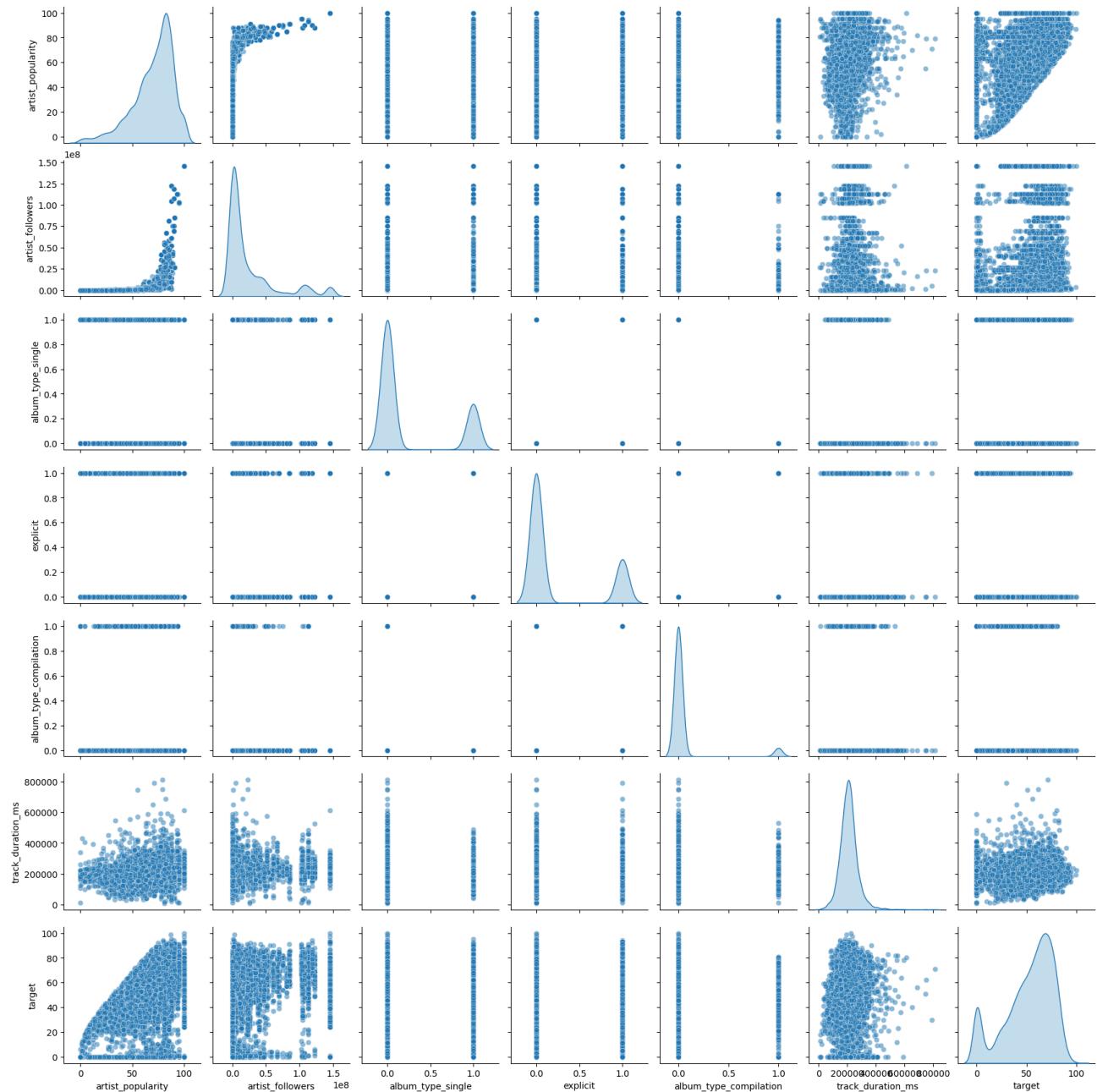
# Pilih beberapa fitur penting untuk visualisasi (agar tidak terlalu padat)
if X_train_df.shape[1] > 6:
    # Gunakan korelasi untuk pilih fitur terpenting
    corr_matrix = X_train_df.corrwith(pd.Series(y_train.values, index=X_train_df.index)).abs()
    top_features = corr_matrix.nlargest(6).index.tolist()
    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']
Scatterplot Matrix: Hubungan Antar Fitur dan Target



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)

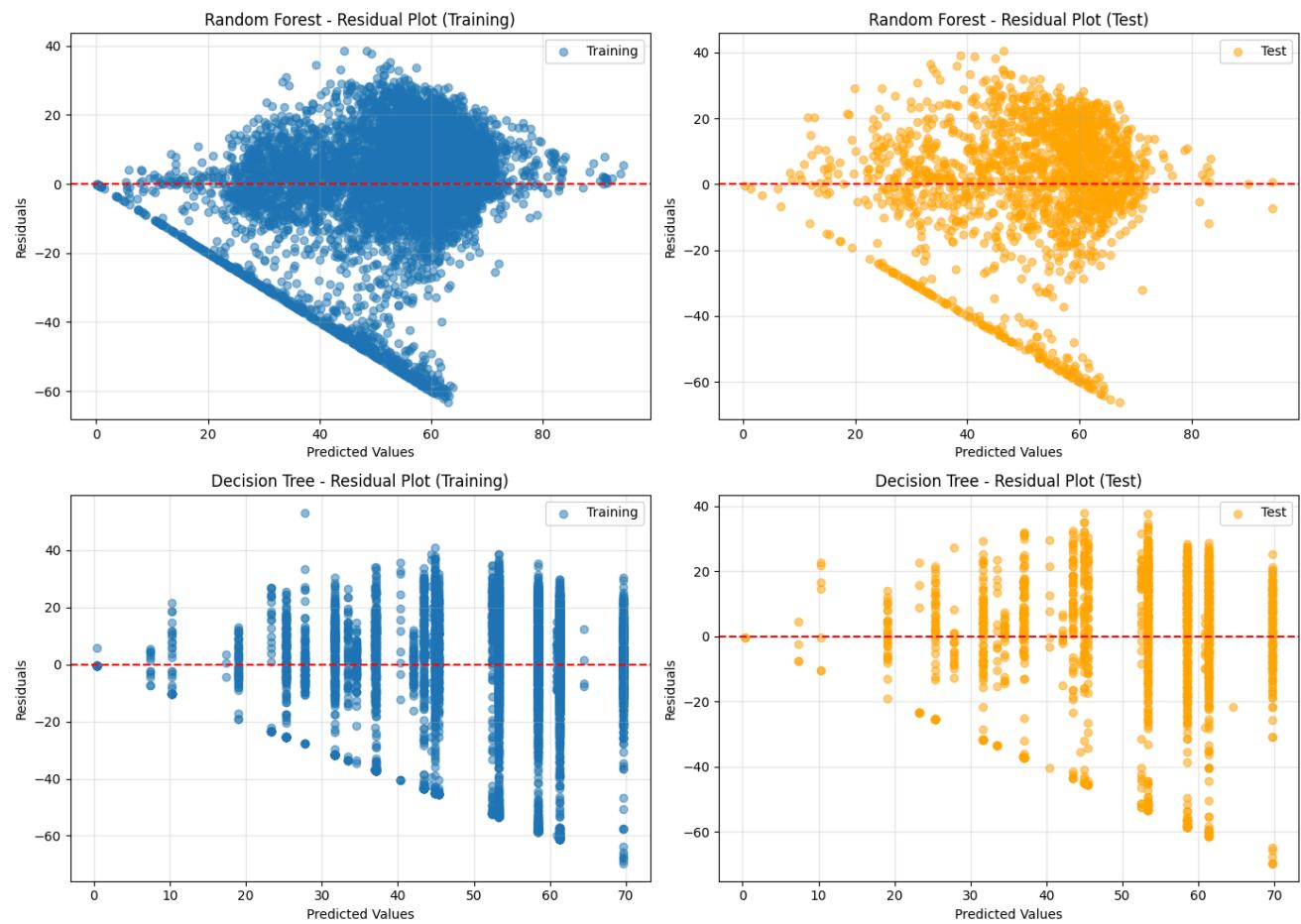
    # Residual
    residuals_train = y_train - y_pred_train
    residuals_test = y_test - y_pred_test

    # 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()

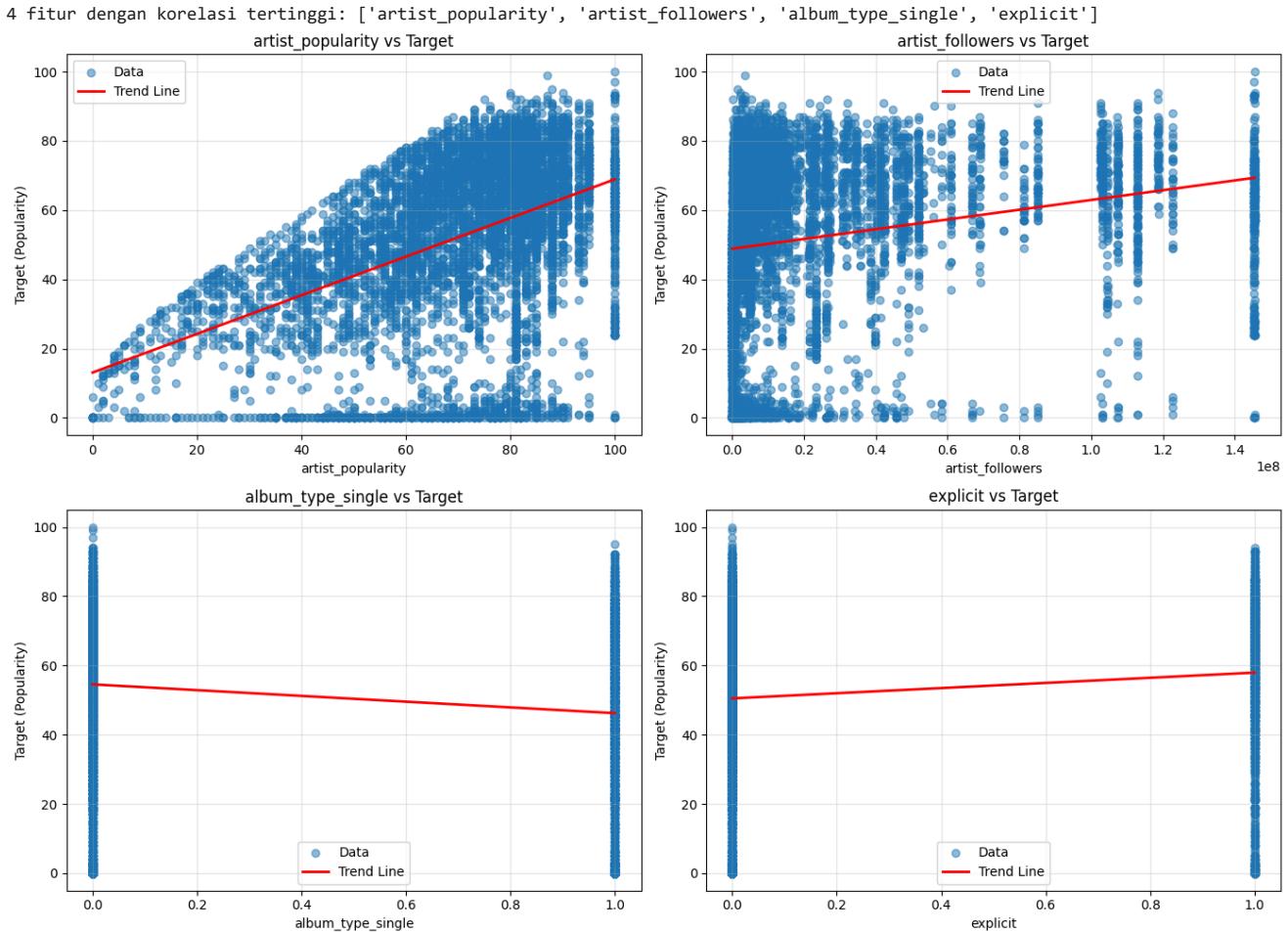
for i, feat_name in enumerate(top_4_features):
    # Scatter plot - AKSES dengan NAMA kolom, bukan indeks
    axes[i].scatter(X_train_df[feat_name], y_train, alpha=0.5, label='Data')

    # 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()
```



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)

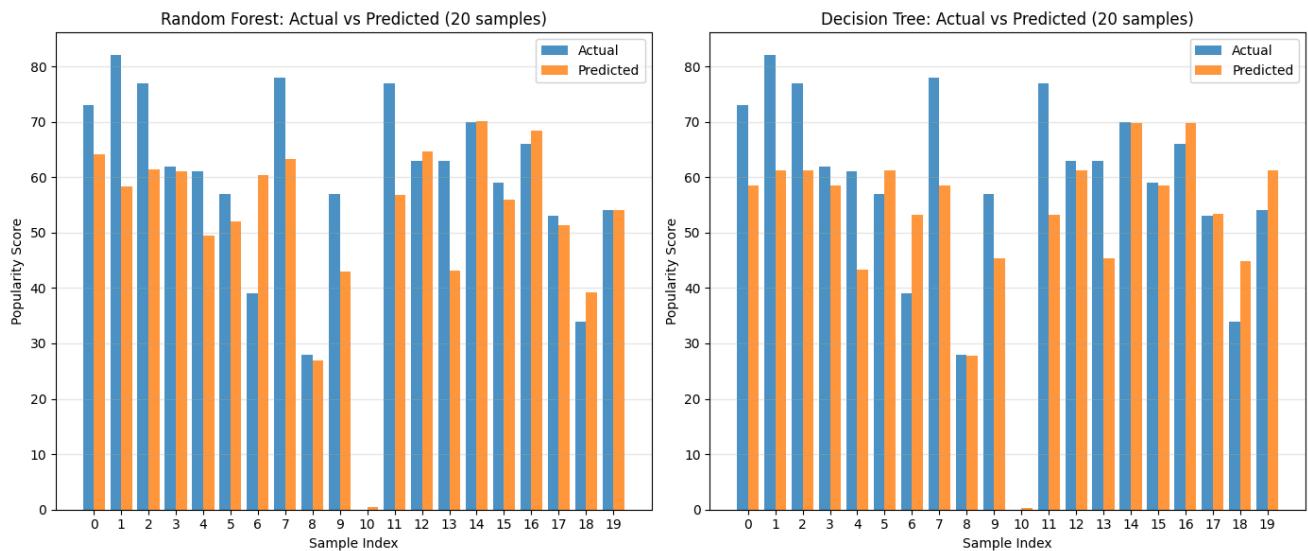
    # Buat dataframe untuk plotting
    df_compare = pd.DataFrame({
        'Actual': y_true_sample,
        'Predicted': y_pred_sample,
        'Sample': list(sample_indices)
    })

    # 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
```