Notebook1_Regresi_KERAS

December 12, 2025

1 Analisis Regresi pada Student Performance Dataset Menggunakan Lasso dan Ridge Regression

1.1 Tujuan Proyek

Proyek ini bertujuan untuk membangun dan membandingkan performa dua algoritma regresi linear yang dilengkapi dengan regularisasi, yaitu Lasso Regression (L1) dan Ridge Regression (L2), dalam memprediksi nilai indeks performa siswa (Performance Index) berdasarkan karakteristik akademik dan kebiasaan siswa.

Dataset yang digunakan adalah Student Performance Dataset, yang berisi kombinasi data numerik dan kategorik, seperti jam belajar, skor sebelumnya, kegiatan ekstrakurikuler, jam tidur, dan jumlah latihan soal.

1.2 Ruang Lingkup dan Tujuan Analisis

Analisis pada notebook ini difokuskan pada Linear Model dengan Regularisasi, dengan tujuan untuk:

1. Melakukan Exploratory Data Analysis (EDA) dan Data Cleaning (penanganan nilai kosong, duplikasi, dan outlier).
2. Menentukan fitur (X) dan target (y), di mana target yang digunakan adalah Performance Index.
3. Membangun dua pipeline model regresi, yaitu:
    - Pipeline 1: Lasso Regression
    - Pipeline 2: Ridge Regression
4. Membandingkan performa kedua model dengan variasi:
    - Scaling: StandardScaler dan MinMaxScaler.
    - Feature Selection: SelectKBest dan SelectPercentile.
5. Melakukan Hyperparameter Tuning (mencari nilai Alpha terbaik) menggunakan GridSearchCV dengan metode Cross Validation.
6. Mengevaluasi performa model berdasarkan metrik R^2, Mean Squared Error (MSE), Mean Absolute Error (MAE), dan Root Mean Squared Error (RMSE).

Alasan Pemilihan Model Lasso Regression (Least Absolute Shrinkage and Selection Operator): Dipilih karena kemampuannya melakukan feature selection secara otomatis. Lasso dapat membuat koefisien fitur yang kurang penting menjadi tepat nol, sehingga menghasilkan model yang lebih sederhana dan mudah diinterpretasikan.

Ridge Regression: Digunakan untuk menangani masalah multikolinearitas (korelasi tinggi antar fitur) dan mencegah overfitting. Ridge memperkecil nilai koefisien fitur tetapi tidak sampai nol, sehingga menjaga semua fitur tetap berkontribusi namun dengan bobot yang terkontrol.

1.3 Tahapan Eksperimen

Tahapan eksperimen dalam penelitian ini dilakukan sebagai berikut:

1. Data Understanding dan Data Cleaning Membaca dataset dan memeriksa tipe data.

Pengecekan dan penanganan missing value.

Pengecekan dan penghapusan data duplikat.

Deteksi outlier menggunakan metode IQR.

2. Data Encoding Mengubah fitur kategorik Extracurricular Activities (Yes/No) menjadi bentuk numerik (1/0).
3. Penentuan Fitur dan Target Menetapkan variabel independen (X) dan variabel dependen target (y).
4. Train–Test Split Membagi dataset menjadi data latih (training) dan data uji (testing) untuk validasi model yang objektif.
5. Pembangunan Pipeline Model Menyusun pipeline yang mencakup langkah-langkah:

Scaler: Menstandarisasi skala data.

Selector: Memilih fitur terbaik secara statistik.

Regressor: Algoritma Lasso atau Ridge.

6. Hyperparameter Tuning dan Cross Validation Melakukan pencarian parameter terbaik (khususnya nilai alpha) menggunakan GridSearchCV.
7. Evaluasi dan Visualisasi Mengevaluasi model menggunakan metrik regresi.

Membandingkan hasil prediksi dengan data aktual.

Menganalisis fitur-fitur yang paling berpengaruh.

8. Pemilihan Model Terbaik Menentukan satu model terbaik (Lasso atau Ridge) berdasarkan skor evaluasi dan mengekspornya dalam format Pickle (.pkl) untuk implementasi pada aplikasi Streamlit.

1.4 Hasil yang Diharapkan

Melalui rangkaian eksperimen ini, diharapkan:

1. Diperoleh model regresi linear terbaik yang mampu memprediksi performa siswa dengan akurasi tinggi (R^2 yang baik dan RMSE yang rendah).

2. Dapat membandingkan efektivitas regularisasi L1 (Lasso) vs L2 (Ridge) pada dataset ini.
3. Mengetahui faktor-faktor (fitur) apa saja yang paling signifikan dalam mempengaruhi nilai siswa.
4. Tersimpannya Best model yang siap digunakan di tahap deployment.

1.4.1 Loading Data

[208]:
```
import pandas as pd
import numpy as np

#  Visualisasi
import matplotlib.pyplot as plt
import seaborn as sns

#  Pembagian data & pencarian hyperparameter
from sklearn.model_selection import train_test_split, GridSearchCV

#  Pra-pemrosesan & Seleksi Fitur
from sklearn.preprocessing import MinMaxScaler, StandardScaler, LabelEncoder
from sklearn.feature_selection import SelectKBest, SelectPercentile,␣
 ↪f_regression

#  Model Regresi
from sklearn.linear_model import Lasso, Ridge

#  Pipeline & Utilitas
from sklearn.pipeline import Pipeline
import pickle
import warnings

#Kolom yang dipilih dari dataset
columns = ['Hours studied', 'Previous score', 'Extracurricular Activities',␣
 ↪'Sleep Hours'
        ,'Sample Question Papers Practiced','Performance Index']

#Membaca dataset
df_house = pd.read_csv('dataset/Student_Performance.csv')

df_house.head()
```

[208]:

	Hours Studied	Previous Scores	Extracurricular Activities	Sleep Hours	\
0	7	99	Yes	9	
1	4	82	No	4	
2	8	51	Yes	7	
3	5	52	Yes	5	
4	7	75	No	8	

```
   Sample Question Papers Practiced  Performance Index
0                                 1               91.0
1                                 2               65.0
2                                 2               45.0
3                                 2               36.0
4                                 5               66.0
```

[209]:
```
#Melihat jumlah baris dan kolom pada dataset
df_house.shape
```

[209]:
```
(10000, 6)
```

[210]:
```
#Mengubah data kategorikal pada kolom 'Extracurricular Activities' menjadi␣
 ↪numerikal
df_house['Extracurricular Activities'] = df_house['Extracurricular Activities'].
 ↪map({'Yes': 1, 'No': 0})
```

[211]:
```
# Mengecek jumlah missing value di setiap kolom
df_house.isnull().sum()
```

[211]:
```
Hours Studied                     0
Previous Scores                   0
Extracurricular Activities        0
Sleep Hours                       0
Sample Question Papers Practiced  0
Performance Index                 0
dtype: int64
```

[212]:
```
# Menghapus seluruh baris yang punya missing value
df = df_house.dropna(axis=0)

# Memastikan tidak ada missing value lag
df.isnull().sum()
```

[212]:
```
Hours Studied                     0
Previous Scores                   0
Extracurricular Activities        0
Sleep Hours                       0
Sample Question Papers Practiced  0
Performance Index                 0
dtype: int64
```

[213]:
```
#Menghapus duplikat pada dataset
before = df_house.shape
dupes = df_house[df_house.duplicated(keep=False)]
print(f"Jumlah baris duplikat (terhitung ganda): {dupes.shape[0]}")
```

```
df_house2 = df_house.drop_duplicates(keep='first')
print("Bentuk data sebelum/ setelah hapus duplikat:", before, "->", df_house2.
 ↪shape)
```

```
Jumlah baris duplikat (terhitung ganda): 253
Bentuk data sebelum/ setelah hapus duplikat: (10000, 6) -> (9873, 6)
```

[214]:
```
# Data Checking: Outlier (IQR)
numeric_cols = df_house.select_dtypes(include=np.number).columns

print("Pengecekan outlier menggunakan metode IQR:\n")

for col in numeric_cols:
    Q1 = df_house[col].quantile(0.25)
    Q3 = df_house[col].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    outlier_count = ((df_house[col] < lower_bound) |
                     (df_house[col] > upper_bound)).sum()

    print(f"{col}: {outlier_count} outlier")
```

```
Pengecekan outlier menggunakan metode IQR:

Hours Studied: 0 outlier
Previous Scores: 0 outlier
Extracurricular Activities: 0 outlier
Sleep Hours: 0 outlier
Sample Question Papers Practiced: 0 outlier
Performance Index: 0 outlier
```

[215]:
```
# Mengimpor StandardScaler untuk menstandarisasi data (mean = 0, std = 1)
from sklearn.preprocessing import StandardScaler

# Mengambil fitur 'Gr Liv Area' sebagai input (X), diubah ke bentuk array NumPy␣
 ↪2D
X = df_house[['Hours Studied']].values

# Mengambil kolom '____' sebagai target (y), dalam bentuk array 1D
y = df_house['Performance Index'].values                                    ␣
 ↪       # Silakan diisi bagian ini dengan kode yang tepat

# Membuat dua scaler terpisah: satu untuk fitur (X), satu untuk target (y)
sc_x = StandardScaler()
```

```
sc_y = StandardScaler()                                                    #␣
 ↪Silakan diisi bagian ini dengan kode yang tepat

# Menstandarisasi X (menghitung mean dan std pada X, lalu mengubah datanya)
X_std = sc_x.fit_transform(X)

# y perlu dijadikan 2D menggunakan np.newaxis sebelum distandarisasi
# Hasil fit_transform dikembalikan ke 1D dengan .flatten()
y_std = sc_y.fit_transform(y[:, np.newaxis]).flatten()
```

2 Visualisasi Hubungan Antar Fitur

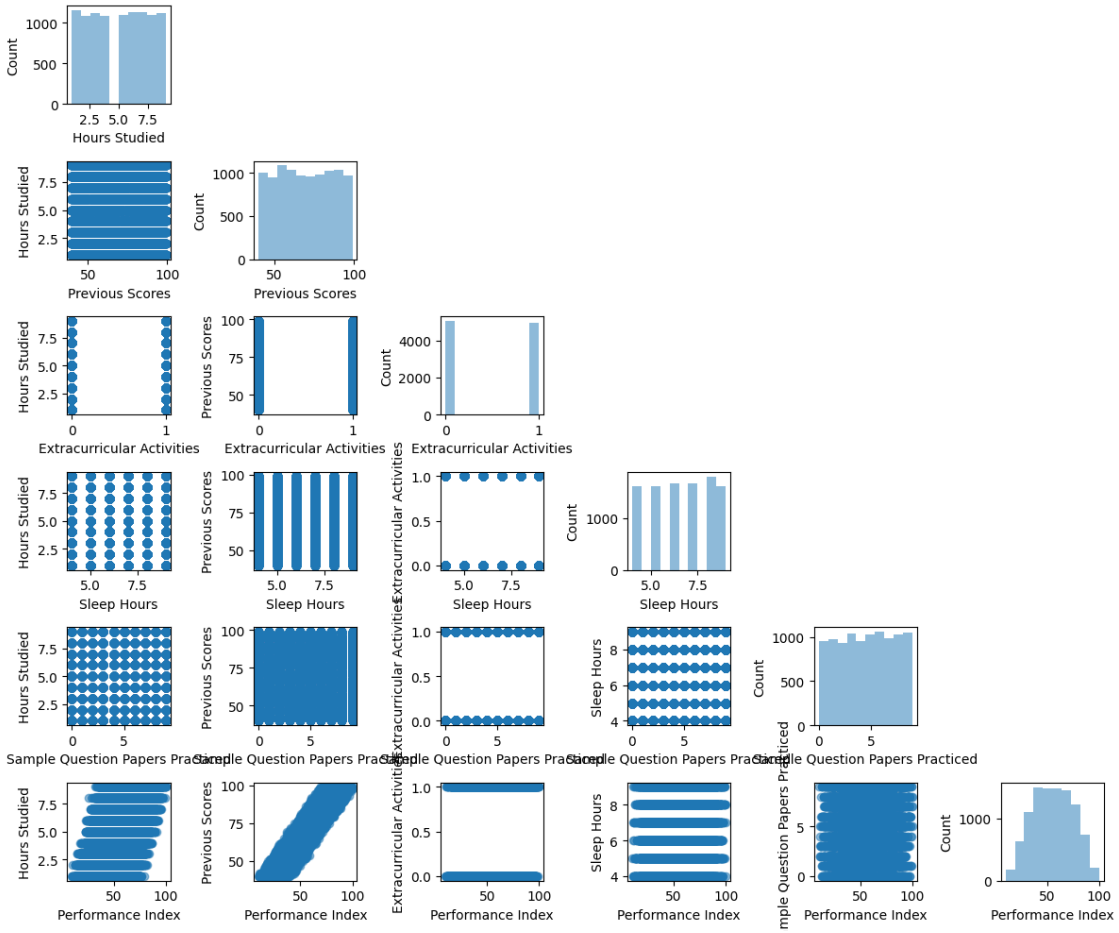
Dua visualisasi digunakan untuk memahami pola awal dataset:

- **Scatterplot Matrix**: melihat hubungan antar fitur dan mendeteksi pola/outlier.

Visualisasi ini membantu memahami struktur data sebelum membangun model regresi.

[216]:
```
import matplotlib.pyplot as plt
from mlxtend.plotting import scatterplotmatrix
```

[217]:
```
# Membuat scatterplot matrix untuk melihat hubungan antar fitur
scatterplotmatrix(df.values, figsize=(12, 10),
                  names=df.columns,              # Membuat scatterplot matrix␣
 ↪untuk melihat hubungan antar fitur
                  alpha=0.5)                     # transparansi titik

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



2.0.1 Pembagian Data: Train/Test Split

[218]:
```
target_col = 'Performance Index'

# 1. Ambil X (Semua kolom KECUALI target)

feature_cols = df.drop(columns=[target_col]).columns.tolist()
X = df[feature_cols].values

# 2. Ambil y (Hanya kolom target)
y = df[target_col].values

# 3. Split Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=98)

print(f"Fitur yang digunakan: {feature_cols}")
print(f"Shape Train: {X_train.shape}")
```

```
print(f"Shape Test : {X_test.shape}")
```

```
Fitur yang digunakan: ['Hours Studied', 'Previous Scores', 'Extracurricular
Activities', 'Sleep Hours', 'Sample Question Papers Practiced']
Shape Train: (7000, 5)
Shape Test : (3000, 5)
```

[219]:
```
# ============================================================
# 1. DEFINISI PIPELINE (Sesuai Rubrik: Scaler -> Selector -> Model)
# ============================================================

# Pipeline 1: LASSO
pipeline_lasso = Pipeline([
    ('scaler', StandardScaler()),       # Placeholder, akan divariasikan di␣
 ↪GridSearch
    ('selector', SelectKBest()),        # Placeholder, akan divariasikan di␣
 ↪GridSearch
    ('regressor', Lasso(random_state=98)) # Model Lasso
])

# Pipeline 2: RIDGE
pipeline_ridge = Pipeline([
    ('scaler', StandardScaler()),
    ('selector', SelectKBest()),
    ('regressor', Ridge(random_state=98)) # Model Ridge
])

# ============================================================
# 2. PARAMETER GRID (Skenario Eksperimen)
# ============================================================
# membuat skenario untuk membandingkan:
# - Scaler: StandardScaler vs MinMaxScaler
# - Selector: SelectKBest vs SelectPercentile
# - Hyperparameter Alpha: 0.01, 0.1, 1, 10
# ============================================================

param_grid = [
    # Skenario A: Menggunakan SelectKBest (Memilih k fitur terbaik)
    {
        'scaler': [StandardScaler(), MinMaxScaler()],
        'selector': [SelectKBest(score_func=f_regression)],
        'selector__k': [3, 4, 5, 'all'],            # Eksperimen jumlah fitur
        'regressor__alpha': [0.01, 0.1, 1, 10]      # Eksperimen alpha
    },
    # Skenario B: Menggunakan SelectPercentile (Memilih % fitur terbaik)
    {
        'scaler': [StandardScaler(), MinMaxScaler()],
```

```
        'selector': [SelectPercentile(score_func=f_regression)],
        'selector__percentile': [50, 75],           # Eksperimen persentase
        'regressor__alpha': [0.01, 0.1, 1, 10]
    }
]

# ============================================================
# 3. EKSEKUSI TRAINING (GridSearchCV)
# ============================================================

print("melatih model LASSO...")
grid_lasso = GridSearchCV(pipeline_lasso, param_grid, cv=5, scoring='r2',␣
 ↪n_jobs=-1)
grid_lasso.fit(X_train, y_train)
print(f"Skor R2 Terbaik Lasso: {grid_lasso.best_score_:.4f}")
print(f"   Parameter Terbaik: {grid_lasso.best_params_}\n")

print("melatih model RIDGE...")
grid_ridge = GridSearchCV(pipeline_ridge, param_grid, cv=5, scoring='r2',␣
 ↪n_jobs=-1)
grid_ridge.fit(X_train, y_train)
print(f"Skor R2 Terbaik Ridge: {grid_ridge.best_score_:.4f}")
print(f"   Parameter Terbaik: {grid_ridge.best_params_}")                #␣
 ↪Standarisasi fitur input
```

```
melatih model LASSO…
Skor R2 Terbaik Lasso: 0.9889
   Parameter Terbaik: {'regressor__alpha': 0.01, 'scaler': StandardScaler(),
'selector': SelectKBest(score_func=<function f_regression at 0x300111a80>),
'selector__k': 5}

melatih model RIDGE…
Skor R2 Terbaik Ridge: 0.9889
   Parameter Terbaik: {'regressor__alpha': 0.01, 'scaler': MinMaxScaler(),
'selector': SelectKBest(score_func=<function f_regression at 0x300111a80>),
'selector__k': 5}
```

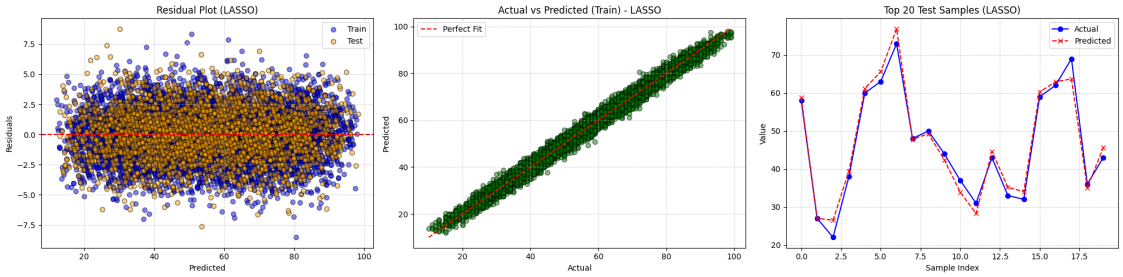
[220]:
```
# ================================================================================
# CELL EVALUASI FINAL (LENGKAP: R2, MSE, MAE, RMSE)
# ================================================================================
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

# ------------------------------------------------------------
# 1. PERSIAPAN DATA (Safety Check)
```

```python
# ------------------------------------------------------------
target_col = 'Performance Index'
feature_names_safe = df.drop(columns=[target_col]).columns.tolist()

print(f"  Fitur yang terdeteksi: {feature_names_safe}")

# ------------------------------------------------------------
# 2. FUNGSI EVALUASI
# ------------------------------------------------------------
def evaluate_and_plot_robust(model_grid, model_name, X_train, y_train, X_test,␣
 ↪y_test, feature_names):
    y_test_np = np.array(y_test).flatten()
    y_train_np = np.array(y_train).flatten()

    # Prediksi menggunakan model terbaik dari GridSearch
    best_model = model_grid.best_estimator_
    y_pred_train = best_model.predict(X_train)
    y_pred_test = best_model.predict(X_test)

    # Hitung Semua Metrik
    r2 = r2_score(y_test_np, y_pred_test)
    mse = mean_squared_error(y_test_np, y_pred_test)
    mae = mean_absolute_error(y_test_np, y_pred_test)
    rmse = np.sqrt(mse)

    print(f"\n{'='*40}")
    print(f"  HASIL EVALUASI: {model_name}")
    print(f"{'='*40}")
    print(f"R2 Score (Test) : {r2:.4f}")
    print(f"MSE             : {mse:.4f}")
    print(f"MAE             : {mae:.4f}")
    print(f"RMSE            : {rmse:.4f}")

    # Cek Fitur Terpilih (Feature Selection)
    try:
        selector = best_model.named_steps['selector']
        if hasattr(selector, 'get_support'):
            mask = selector.get_support()
            selected_feats = np.array(feature_names)[mask]
            print(f"Fitur Terpilih: {list(selected_feats)}")
        else:
            print("Semua fitur digunakan.")
    except:
        print("Info: Tidak dapat menampilkan detail fitur.")

    # --- VISUALISASI ---
    fig, axes = plt.subplots(1, 3, figsize=(20, 5))
```

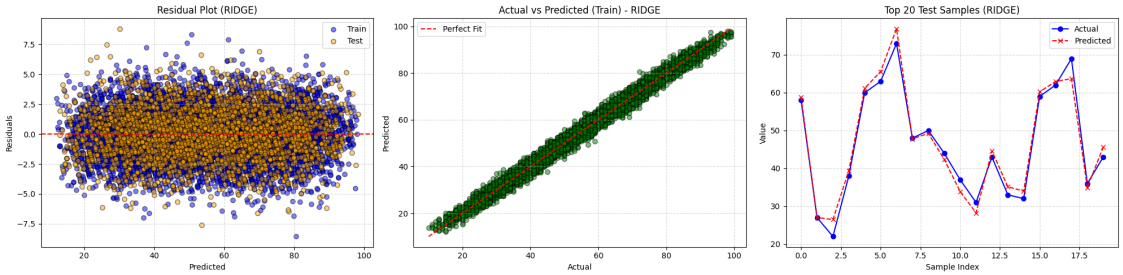
```
    # Plot 1: Residual Plot
    residual_train = y_train_np - y_pred_train
    residual_test = y_test_np - y_pred_test
    axes[0].scatter(y_pred_train, residual_train, c='blue', alpha=0.5,␣
↪label='Train', edgecolor='k')
    axes[0].scatter(y_pred_test, residual_test, c='orange', alpha=0.5,␣
↪label='Test', edgecolor='k')
    axes[0].axhline(0, c='red', linestyle='--')
    axes[0].set_title(f'Residual Plot ({model_name})')
    axes[0].set_xlabel('Predicted')
    axes[0].set_ylabel('Residuals')
    axes[0].legend()
    axes[0].grid(True, linestyle='--', alpha=0.5)

    # Plot 2: Actual vs Predicted (Train)
    axes[1].scatter(y_train_np, y_pred_train, c='green', alpha=0.5,␣
↪edgecolor='k')
    min_val = min(y_train_np.min(), y_pred_train.min())
    max_val = max(y_train_np.max(), y_pred_train.max())
    axes[1].plot([min_val, max_val], [min_val, max_val], 'r--', label='Perfect␣
↪Fit')
    axes[1].set_title(f'Actual vs Predicted (Train) - {model_name}')
    axes[1].set_xlabel('Actual')
    axes[1].set_ylabel('Predicted')
    axes[1].legend()
    axes[1].grid(True, linestyle='--', alpha=0.5)

    # Plot 3: Top 20 Samples Comparison (Test)
    limit = min(20, len(y_test_np))
    indices = np.arange(limit)
    axes[2].plot(indices, y_test_np[:limit], marker='o', c='blue',␣
↪label='Actual')
    axes[2].plot(indices, y_pred_test[:limit], marker='x', linestyle='--',␣
↪c='red', label='Predicted')
    axes[2].set_title(f'Top {limit} Test Samples ({model_name})')
    axes[2].set_xlabel('Sample Index')
    axes[2].set_ylabel('Value')
    axes[2].legend()
    axes[2].grid(True, linestyle='--', alpha=0.5)

    plt.tight_layout()
    plt.show()

# ------------------------------------------------------------
# 3. EKSEKUSI
```

```
# ------------------------------------------------------------
evaluate_and_plot_robust(grid_lasso, "LASSO", X_train, y_train, X_test, y_test,␣
 ↪feature_names_safe)
evaluate_and_plot_robust(grid_ridge, "RIDGE", X_train, y_train, X_test, y_test,␣
 ↪feature_names_safe)
```

```
 Fitur yang terdeteksi: ['Hours Studied', 'Previous Scores', 'Extracurricular
Activities', 'Sleep Hours', 'Sample Question Papers Practiced']

========================================
 HASIL EVALUASI: LASSO
========================================
R2 Score (Test) : 0.9882
MSE             : 4.2708
MAE             : 1.6374
RMSE            : 2.0666
Fitur Terpilih: [np.str_('Hours Studied'), np.str_('Previous Scores'),
np.str_('Extracurricular Activities'), np.str_('Sleep Hours'), np.str_('Sample
Question Papers Practiced')]
```




```
========================================
 HASIL EVALUASI: RIDGE
========================================
R2 Score (Test) : 0.9883
MSE             : 4.2669
MAE             : 1.6366
RMSE            : 2.0656
Fitur Terpilih: [np.str_('Hours Studied'), np.str_('Previous Scores'),
np.str_('Extracurricular Activities'), np.str_('Sleep Hours'), np.str_('Sample
Question Papers Practiced')]
```



[221]:
```python
import pickle

# 1. Bandingkan Skor Terbaik (R2 Score dari Cross Validation)
score_lasso = grid_lasso.best_score_
score_ridge = grid_ridge.best_score_

print(f"Best CV Score - Lasso: {score_lasso:.4f}")
print(f"Best CV Score - Ridge: {score_ridge:.4f}")

best_model = None
best_algo_name = ""

# 2. Logika Pemilihan
if score_lasso > score_ridge:
    best_model = grid_lasso.best_estimator_
    best_algo_name = "Lasso"
    print("\nKESIMPULAN: Model Terbaik adalah LASSO")
else:
    best_model = grid_ridge.best_estimator_
    best_algo_name = "Ridge"
    print("\nKESIMPULAN: Model Terbaik adalah RIDGE")

# Tampilkan parameter akhir model terbaik
print(f"Parameter Terbaik: {best_model}")


# ------------------------------------------------------------
#  MENAMPILKAN FITUR TERBAIK
# ------------------------------------------------------------
print(f"\n--- Fitur yang Dipilih oleh {best_algo_name} ---")

try:
    selector = best_model.named_steps['selector']

    mask = selector.get_support()
```

```
    nama_fitur_asli = df_house.drop(columns=['Performance Index']).columns

    fitur_terpilih = nama_fitur_asli[mask]

    print(f"Jumlah Fitur Terpilih: {len(fitur_terpilih)}")
    for i, nama_fitur in enumerate(fitur_terpilih, 1):
        print(f"{i}. {nama_fitur}")

except Exception as e:
    print(f"  Masih ada error kecil: {str(e)}")
    print("Tips: Pastikan sel tempat Anda mendefinisikan 'df_house' sudah␣
 ↪dijalankan.")

with open(f"model/BestModel_{best_algo_name}_KERAS.pkl", "wb") as f:
    pickle.dump(best_model, f)

print(f"\nModel terbaik: {best_algo_name}")
print(f"File tersimpan sebagai: BestModel_{best_algo_name}_KERAS.pkl")
```

```
Best CV Score - Lasso: 0.9889
Best CV Score - Ridge: 0.9889

KESIMPULAN: Model Terbaik adalah RIDGE
Parameter Terbaik: Pipeline(steps=[('scaler', MinMaxScaler()),
                ('selector',
                 SelectKBest(k=5,
                             score_func=<function f_regression at
0x300111a80>)),
                ('regressor', Ridge(alpha=0.01, random_state=98))])

--- Fitur yang Dipilih oleh Ridge ---
Jumlah Fitur Terpilih: 5
1. Hours Studied
2. Previous Scores
3. Extracurricular Activities
4. Sleep Hours
5. Sample Question Papers Practiced

Model terbaik: Ridge
File tersimpan sebagai: BestModel_Ridge_KERAS.pkl
```

Notebook2_Regresi_KERAS

December 12, 2025

1 Analisis Regresi pada Student Performance Dataset Menggunakan Decision Tree dan Random Forest Regression

1.1 Tujuan Proyek

Proyek ini bertujuan untuk membangun dan membandingkan performa dua algoritma regresi non-linear, yaitu Decision Tree Regressor dan Random Forest Regressor, dalam memprediksi nilai median harga rumah (Median House Value) berdasarkan karakteristik demografis dan geografis wilayah di California.

Dataset yang digunakan adalah Student Performance Dataset, yang berisi kombinasi data numerik dan kategorik, seperti jam belajar, tingkat kehadiran, nilai sebelumnya, jenis kelamin, tingkat pendidikan orang tua, serta faktor pendukung lainnya.

1.2 Ruang Lingkup dan Tujuan Analisis

Analisis pada notebook ini difokuskan pada regresi non-linear, dengan tujuan untuk:

Melakukan exploratory data analysis (EDA) guna memahami karakteristik dataset, termasuk pengecekan nilai kosong, duplikasi data, dan distribusi fitur numerik.

Menentukan fitur (X) dan target (y), di mana target yang digunakan adalah performa siswa.

Membangun dua pipeline model regresi non-linear, yaitu:

Pipeline 1: Decision Tree Regressor

Pipeline 2: Random Forest Regressor

Membandingkan performa kedua model menggunakan:

Dua metode penskalaan data: StandardScaler dan MinMaxScaler

Dua metode seleksi fitur: SelectKBest dan SelectPercentile

Melakukan hyperparameter tuning menggunakan GridSearchCV dengan metode Cross Validation.

Mengevaluasi performa model berdasarkan metrik regresi R^2, Mean Squared Error (MSE), Mean Absolute Error (MAE), dan Root Mean Squared Error (RMSE).

1.3 Alasan Pemilihan Model

Decision Tree Regressor dipilih karena kemampuannya dalam menangkap hubungan non-linear antar fitur tanpa memerlukan asumsi bentuk hubungan tertentu, serta kemudahan dalam inter-

pretasi struktur pohon keputusan. Model ini sesuai untuk menganalisis pengaruh berbagai faktor akademik dan non-akademik terhadap performa siswa.

Random Forest Regressor digunakan sebagai model ensemble yang mengombinasikan banyak decision tree untuk meningkatkan stabilitas dan akurasi prediksi. Dengan mekanisme agregasi beberapa pohon, model ini mampu mengurangi overfitting yang sering terjadi pada decision tree tunggal dan memberikan performa yang lebih baik pada data uji.

1.4 Tahapan Eksperimen

Tahapan eksperimen dalam penelitian ini dilakukan sebagai berikut:

1.4.1 1 Data Understanding dan Data Cleaning

Membaca dataset, menampilkan contoh data, informasi kolom, serta statistik deskriptif. Dilakukan pengecekan missing value, data duplikat, dan outlier.

1.4.2 2 Data Encoding

Mengidentifikasi fitur kategorik dan melakukan encoding (misalnya One-Hot Encoding) untuk mengubah data kategorik menjadi bentuk numerik yang dapat diproses oleh model regresi.

1.4.3 3 Penentuan Fitur dan Target

Menentukan variabel fitur (X) dan variabel target (y) berupa nilai performa siswa.

1.4.4 4 Train–Test Split

Membagi dataset menjadi data latih dan data uji dengan rasio 80:20 / 75:25 / 70:30, serta menggunakan nilai random_state sesuai ketentuan UAS.

1.4.5 5 Pembangunan Pipeline Model

Menyusun pipeline yang mencakup: + Scaling (StandardScaler dan MinMaxScaler) + Feature selection (SelectKBest dan SelectPercentile) + Model regresi (Decision Tree dan Random Forest)

1.4.6 6 Hyperparameter Tuning dan Cross Validation

Melakukan pencarian parameter terbaik menggunakan GridSearchCV dengan metode Cross Validation.

1.4.7 7 Evaluasi dan Visualisasi

Mengevaluasi performa model menggunakan R^2, MSE, MAE, dan RMSE, serta menampilkan visualisasi: * Scatterplot matrix antar fitur * Plot residual data training dan testing * Plot hubungan fitur terhadap target * Perbandingan nilai prediksi dan aktual pada data uji

1.4.8 8 Pemilihan Model Terbaik

Menentukan satu model regresi terbaik dan mengekspornya dalam format Pickle (.pkl) untuk digunakan pada aplikasi Streamlit.

1.5 Hasil yang Diharapkan

Melalui rangkaian eksperimen yang dilakukan, hasil yang diharapkan dari notebook ini adalah sebagai berikut:

Diperoleh model regresi non-linear terbaik dengan nilai R^2 tertinggi serta nilai error (MSE, MAE, dan RMSE) terendah.

Teridentifikasi fitur-fitur paling relevan yang berkontribusi signifikan dalam memprediksi Median House Value pada dataset California Housing.

Diperoleh perbandingan performa yang jelas antara Decision Tree Regressor dan Random Forest Regressor, baik dari sisi akurasi prediksi maupun stabilitas model.

Terlihat bahwa Random Forest Regressor secara umum diharapkan memiliki performa lebih baik dibandingkan Decision Tree Regressor, terutama dalam mengurangi overfitting dan meningkatkan generalisasi model.

Hasil evaluasi dan visualisasi dapat menjadi dasar pemilihan model terbaik yang selanjutnya digunakan dalam aplikasi Streamlit sebagai implementasi akhir projek UAS.

[ ]:
```
# ============================================================
#   IMPORT LIBRARY - Regresi Non-Linear (UAS ML)
# ============================================================

#   Manipulasi Data
import pandas as pd
import numpy as np

#   Visualisasi
import matplotlib.pyplot as plt
import seaborn as sns

#   Pembagian Data & Hyperparameter Tuning
from sklearn.model_selection import train_test_split, KFold, GridSearchCV

#   Pra-pemrosesan & Seleksi Fitur
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.feature_selection import SelectKBest, SelectPercentile,␣
 ↪f_regression

#   Model Regresi Non-Linear
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

#   Pipeline
from sklearn.pipeline import Pipeline

#   Evaluasi Model Regresi
from sklearn.metrics import (
```

```
    r2_score,
    mean_squared_error,
    mean_absolute_error
)

#  Utilitas
import time

np.random.seed(98)
```

1.6 Load Data

[ ]:
```
# Load Dataset
df_house = pd.read_csv("dataset/Student_Performance.csv", header=0)

df_house.head()
```

[ ]:
```
# Info cepat tentang kolom & tipe datanya
print("Jumlah baris, kolom:", df_house.shape)
print("\nTipe data:")
print(df_house.dtypes)
```

1.7 PEMBERSIHAN DATA

[ ]:
```
# 1) Cek jumlah nilai kosong per kolom
print("Jumlah nilai kosong per Kolom:\n", df_house.isnull().sum())
missing_values = df_house.isnull().sum()
```

[ ]:
```
# Penanganan missing value (jika ada)
if missing_values.sum() > 0:
    for col in df_house.select_dtypes(include=[np.number]).columns:
        median_val = df_house[col].median()
        df_house[col].fillna(median_val, inplace=True)
```

[ ]:
```
# 3) Validasi ulang
print("\nSetelah inputasi, nilai kosong per kolom:\n", df_house.isnull().sum())
```

1.7.1 Pembersihan Data (Bagian 3): Cek & Hapus Duplikat

- Data yang **kembar** dapat merusak evaluasi model.
- Kita cek duplikat lalu **drop** agar setiap baris unik.

[ ]:
```
before = df_house.shape
dupes = df_house[df_house.duplicated(keep=False)]
print(f"Jumlah baris duplikat (terhitung ganda): {dupes.shape[0]}")
df_house2 = df_house.drop_duplicates(keep='first')
```

```
print("Bentuk data sebelum/ setelah hapus duplikat:", before, "->", df_house2.
  ↪shape)
```

1.7.2 Pembersihan Data (Bagian 4): Cek Outlier

- Data yang **Outlier** dapat merusak evaluasi model.
- Kenapa tidak dihapus? karena itu adalah fenomena nyata dan **Penting** untuk klasifikasi.

[ ]:
```
# =========================
# Data Checking: Outlier (IQR)
# =========================
numeric_cols = df_house.select_dtypes(include=np.number).columns

print("Pengecekan outlier menggunakan metode IQR:\n")

for col in numeric_cols:
    Q1 = df_house[col].quantile(0.25)
    Q3 = df_house[col].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    outlier_count = ((df_house[col] < lower_bound) |
                     (df_house[col] > upper_bound)).sum()

    print(f"{col}: {outlier_count} outlier")
```

[ ]:
```
# =========================
# Data Preparation: Outlier Handling (Capping / Winsorization)
# =========================

df_house3 = df_house2.copy()  # df_house2 = hasil drop_duplicates sebelumnya

numeric_cols = df_house3.select_dtypes(include=np.number).columns

# (opsional) jika target ada di dataset, jangan ikut dicapping
# ganti 'Performance Index' sesuai nama target kamu
target_col = 'Performance Index'  # <-- UBAH sesuai kolom target
numeric_features = [c for c in numeric_cols if c != target_col]

before_shape = df_house3.shape

for col in numeric_features:
    Q1 = df_house3[col].quantile(0.25)
    Q3 = df_house3[col].quantile(0.75)
    IQR = Q3 - Q1
```

```
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR

    # capping (winsorize)
    df_house3[col] = np.clip(df_house3[col], lower, upper)

print("Bentuk data sebelum:", before_shape)
print("Bentuk data sesudah :", df_house3.shape)

# Validasi ulang jumlah outlier setelah capping
print("\nPengecekan outlier (IQR) setelah capping:\n")
for col in numeric_features:
    Q1 = df_house3[col].quantile(0.25)
    Q3 = df_house3[col].quantile(0.75)
    IQR = Q3 - Q1
    outliers = ((df_house3[col] < (Q1 - 1.5 * IQR)) | (df_house3[col] > (Q3 + 1.
 ↪5 * IQR))).sum()
    print(f"{col}: {outliers} outlier")
```

1.7.3 Pembagian Data: Train/Test Split

1.7.4 Encode Label

- **Tujuan:** Mengubah kolom `Extracurricular Activities` dari huruf menjadi angka agar bisa diproses oleh model ML.
  - `Yes` → 1 (**mengikuti extracurricular**)
  - `No` → 0 (**tidak mengikuti extracurricular**)

[ ]:
```
# Mengubah label Extracurricular Activities dari huruf menjadi angka:
df_house3['Extracurricular Activities'] = df_house3['Extracurricular␣
 ↪Activities'].map({'Yes': 1, 'No':0})
```

[ ]:
```
# Menentukan X sebagai fitur (semua kolom kecuali Performance Index)
X = df_house3.drop(columns=['Performance Index'])

# Menentukan y sebagai target (kolom Performance Index)
y = df_house3['Performance Index']
```

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=98
)

print("Ukuran X_train, X_test :", X_train.shape, X_test.shape)
```

```python
from sklearn.model_selection import KFold

cv = KFold(
    n_splits=5,
    shuffle=True,
    random_state=98
)
```

1.8 Membangun Model Regresi Non-Linear (Decision Tree & Random Forest) dengan Pipeline + GridSearchCV

Pada bagian ini, kita akan membangun dan membandingkan dua model regresi non-linear berbasis pohon, yaitu Decision Tree Regressor (DTR) dan Random Forest Regressor (RFR). Kedua algoritma ini mampu menangkap pola hubungan fitur–target yang kompleks tanpa mengasumsikan hubungan linear.

1.9 Konsep Singkat Model Berbasis Pohon (Tree-Based Regression)

Decision Tree Regressor bekerja dengan membagi data secara berulang berdasarkan aturan (split) pada fitur tertentu untuk meminimalkan error prediksi pada setiap node. Model ini mudah diinterpretasikan, tetapi dapat overfitting jika kedalaman pohon terlalu besar.

Random Forest Regressor adalah model ensemble yang menggabungkan banyak decision tree dengan teknik bootstrap sampling dan pemilihan fitur acak. Pendekatan ini biasanya menghasilkan model yang lebih stabil dan akurat karena mampu mengurangi overfitting yang sering terjadi pada decision tree tunggal.

1.10 Alur Pipeline

1.10.1 1 Scaling (StandardScaler vs MinMaxScaler)

Scaling disertakan untuk memenuhi ketentuan eksperimen UAS dan menjaga konsistensi pipeline, meskipun model berbasis pohon umumnya tidak sensitif terhadap skala fitur.

1.10.2 2 Feature Selection (SelectKBest vs SelectPercentile)

Tahap ini digunakan untuk memilih fitur yang paling relevan terhadap target regresi: - SelectKBest: memilih k fitur terbaik - SelectPercentile: memilih persentase fitur terbaik (p%) - Skor seleksi fitur menggunakan metode f_regression, karena sesuai untuk kasus regresi (target numerik).

1.10.3 3 Model (DecisionTreeRegressor / RandomForestRegressor)

Parameter yang diuji melalui GridSearchCV meliputi: Decision Tree Regressor: - max_depth: kedalaman maksimum pohon - min_samples_split: minimum sampel untuk membagi node - min_samples_leaf: minimum sampel pada leaf - random_state: menjaga reprodusibilitas Random Forest Regressor: - n_estimators: jumlah pohon dalam forest - max_depth, min_samples_split, min_samples_leaf - max_features: jumlah fitur yang dipertimbangkan saat split - random_state: menjaga reprodusibilitas

1.10.4 Tujuan GridSearchCV

Melakukan pencarian otomatis kombinasi parameter terbaik untuk menghasilkan model dengan performa optimal menggunakan K-Fold Cross Validation (mis. 5-fold). Evaluasi utama dilakukan menggunakan R^2, serta dilengkapi metrik error MSE, MAE, dan RMSE untuk melihat seberapa besar kesalahan prediksi model.

Output yang Diharapkan dari Cell Ini - Model Decision Tree Regressor terbaik beserta parameter optimalnya - Model Random Forest Regressor terbaik beserta parameter optimalnya
- Waktu komputasi total untuk masing-masing GridSearchCV - Perbandingan performa model berdasarkan R^2, MSE, MAE, RMSE serta jumlah fitur terbaik yang terpilih

[ ]:
```
# ============================================================
# Pipeline Decision Tree Regressor (Regresi) + GridSearchCV
# ============================================================

from sklearn.pipeline import Pipeline
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.feature_selection import SelectKBest, SelectPercentile,␣
 ↪f_regression
from sklearn.tree import DecisionTreeRegressor
import time

# Pipeline dasar (scaler -> selector -> regressor)
pipe_dt = Pipeline([
    ('scaler', StandardScaler()),  # akan digrid: StandardScaler / MinMaxScaler
    ('selector', SelectKBest(score_func=f_regression)),  # akan digrid: KBest /␣
 ↪Percentile
    ('reg', DecisionTreeRegressor(random_state=98)),
])

param_grid_dt = [
    # ---- SelectKBest ----
    {
        'scaler': [StandardScaler(), MinMaxScaler()],
        'selector': [SelectKBest(score_func=f_regression)],
        'selector__k': [2, 4, 'all'],  # sesuaikan dengan jumlah fitur kamu
        'reg__max_depth': [None, 3, 5, 10],
        'reg__min_samples_split': [2, 5, 10],
        'reg__min_samples_leaf': [1, 2, 4],
    },
    # ---- SelectPercentile ----
    {
        'scaler': [StandardScaler(), MinMaxScaler()],
        'selector': [SelectPercentile(score_func=f_regression)],
        'selector__percentile': [50, 70, 100],
        'reg__max_depth': [None, 3, 5, 10],
```

```
        'reg__min_samples_split': [2, 5, 10],
        'reg__min_samples_leaf': [1, 2, 4],
    }
]

# CV untuk regresi (bukan stratified)
cv = KFold(n_splits=5, shuffle=True, random_state=98)

grid_dt = GridSearchCV(
    estimator=pipe_dt,
    param_grid=param_grid_dt,
    cv=cv,
    scoring='r2',      # evaluasi utama UAS
    n_jobs=-1,
    verbose=1
)

start = time.time()
grid_dt.fit(X_train, y_train)

print("\n Best Parameters (Decision Tree Regressor):", grid_dt.best_params_)
print(" Best CV R²:", grid_dt.best_score_)
print(f"GridSearch Decision Tree selesai dalam {time.time() - start:.2f} detik")
```

1.11 Membangun Model Random Forest Regressor dengan Pipeline + GridSearchCV

Pada bagian ini, kita akan membangun model Random Forest Regressor (RFR) — algoritma regresi non-linear berbasis ensemble yang menggabungkan banyak decision tree untuk menghasilkan prediksi yang lebih stabil dan akurat. Model ini cocok digunakan untuk studi kasus regresi seperti prediksi performa siswa (target numerik) berdasarkan berbagai fitur akademik dan non-akademik.

1.12 Konsep Singkat Random Forest Regression

Random Forest Regressor bekerja dengan membangun banyak decision tree menggunakan teknik bootstrap sampling (mengambil sampel acak dari data latih) dan menggabungkan prediksi setiap pohon (umumnya dengan rata-rata untuk regresi). Selain itu, pada setiap split, Random Forest hanya mempertimbangkan subset fitur secara acak (feature randomness) sehingga model menjadi lebih robust.

Ciri utama:

- Mampu menangkap pola non-linear dan interaksi antar fitur.
- Lebih stabil dibanding Decision Tree tunggal karena hasilnya merupakan gabungan banyak pohon.
- Cenderung lebih tahan overfitting, tetapi tetap perlu tuning parameter agar tidak terlalu kompleks dan lambat.

Alur Pipeline ## 1) Scaling (MinMaxScaler vs StandardScaler)

Scaling tetap disertakan untuk memenuhi ketentuan eksperimen UAS dan menjaga konsistensi pipeline, meskipun model berbasis pohon umumnya tidak sensitif terhadap skala fitur. Dua metode scaling dibandingkan:

- StandardScaler: menstandarkan data (mean = 0, std = 1).
- MinMaxScaler: menormalisasi data ke rentang [0, 1].

1.13 2) Feature Selection (SelectKBest vs SelectPercentile)

- Digunakan untuk memilih fitur yang paling relevan terhadap target regresi:
- SelectKBest: memilih k fitur terbaik.
- SelectPercentile: memilih persentase fitur terbaik (p%). Fungsi skor yang digunakan adalah f_regression, karena sesuai untuk hubungan fitur numerik dengan target numerik.

1.14 3) Model (RandomForestRegressor)

- Parameter utama yang diuji melalui GridSearchCV antara lain:
- n_estimators: jumlah pohon dalam forest.
- max_depth: kedalaman maksimum setiap pohon (kontrol kompleksitas).
- min_samples_split: minimum sampel untuk melakukan split.
- min_samples_leaf: minimum sampel yang harus ada pada leaf.
- max_features: jumlah fitur yang dipertimbangkan saat melakukan split.
- random_state: menjaga konsistensi hasil eksperimen.

1.15 Tujuan GridSearchCV

Melakukan pencarian kombinasi parameter terbaik untuk mendapatkan model Random Forest yang paling optimal dan stabil, dengan evaluasi menggunakan 5-Fold Cross Validation (KFold) karena kasus ini merupakan regresi (target kontinu).

- Metrik evaluasi utama yang digunakan adalah:
- R^2 (R-Squared) sebagai skor utama, serta metrik error tambahan:
- MSE, MAE, dan RMSE untuk mengukur besar kesalahan prediksi.

2 Output dari Tahap Ini

2.1 Output yang diharapkan dari tahap pemodelan ini meliputi:

- Model Random Forest Regressor terbaik dengan parameter hasil optimasi GridSearchCV,
- Skor R^2 rata-rata cross validation terbaik,
- Daftar fitur terpilih dari feature selection pada konfigurasi terbaik,

- Evaluasi pada data uji berupa R^2, MSE, MAE, RMSE,
- Visualisasi yang mendukung analisis model (residual plot, prediksi vs aktual, dan visualisasi lain sesuai ketentuan UAS).

[ ]:
```
# ============================================================
# Pipeline Random Forest Regressor (UAS Regresi)
# ============================================================

pipe_rf = Pipeline([
    ('scaler', StandardScaler()),
    ('selector', SelectKBest(score_func=f_regression)),
    ('reg', RandomForestRegressor(random_state=98))
])

param_grid_rf = [
    # --- SelectKBest ---
    {
        'scaler': [StandardScaler(), MinMaxScaler()],
        'selector': [SelectKBest(score_func=f_regression)],
        'selector__k': [2, 4, 'all'],   # sesuaikan dengan jumlah fitur kamu
        'reg__n_estimators': [100, 200],
        'reg__max_depth': [None, 5, 10],
        'reg__min_samples_split': [2, 5, 10],
        'reg__min_samples_leaf': [1, 2, 4],
        'reg__max_features': ['sqrt', 0.8, 1.0]
    },
    # --- SelectPercentile ---
    {
        'scaler': [StandardScaler(), MinMaxScaler()],
        'selector': [SelectPercentile(score_func=f_regression)],
        'selector__percentile': [50, 70, 100],
        'reg__n_estimators': [100, 200],
        'reg__max_depth': [None, 5, 10],
        'reg__min_samples_split': [2, 5, 10],
        'reg__min_samples_leaf': [1, 2, 4],
        'reg__max_features': ['sqrt', 0.8, 1.0]
    }
]

cv = KFold(n_splits=5, shuffle=True, random_state=98)

grid_rf = GridSearchCV(
    estimator=pipe_rf,
    param_grid=param_grid_rf,
    cv=cv,
    scoring='r2',      # metrik utama regresi
    n_jobs=-1,
```

```
    verbose=1
)

start = time.time()
grid_rf.fit(X_train, y_train)

print("\n Best Parameters (Random Forest Regressor):", grid_rf.best_params_)
print(" Best CV R²:", grid_rf.best_score_)
print(f"GridSearch Random Forest selesai dalam {time.time() - start:.2f} detik")
```

[ ]:
```
# Ambil model terbaik dari GridSearch
best_rf = grid_rf.best_estimator_
y_pred_rf = best_rf.predict(X_test)

# Hitung metrik regresi
metrics_rf = {
    "R2": r2_score(y_test, y_pred_rf),
    "MSE": mean_squared_error(y_test, y_pred_rf),
    "MAE": mean_absolute_error(y_test, y_pred_rf),
    "RMSE": np.sqrt(mean_squared_error(y_test, y_pred_rf))
}

print("Kombinasi model terbaik (Random Forest):", best_rf)
print("CV score terbaik (R²):", grid_rf.best_score_)

print("\n Random Forest Regression Performance (Test Set):")
for k, v in metrics_rf.items():
    print(f"{k}: {v:.4f}")

# Tampilkan fitur terpilih dari feature selection
selector = best_rf.named_steps['selector']
if hasattr(selector, 'get_support'):
    mask = selector.get_support()
    selected = np.array(X.columns)[mask]
    print("\nFitur terbaik (terpilih):", selected)

# ========================
# Visualisasi Wajib (Regresi)
# ========================

# 1) Prediksi vs Aktual (20 data test pertama)
plt.figure(figsize=(8, 5))
plt.plot(np.array(y_test)[:20], label="Aktual", marker='o')
plt.plot(y_pred_rf[:20], label="Prediksi", marker='x')
plt.title("Prediksi vs Aktual - Random Forest (20 Data Test Pertama)")
plt.xlabel("Index Sampel")
plt.ylabel("Nilai Target")
```

```
plt.legend()
plt.grid(True)
plt.show()

# 2) Residual Plot (Test)
residuals = np.array(y_test) - y_pred_rf

plt.figure(figsize=(8, 5))
plt.scatter(y_pred_rf, residuals, alpha=0.7)
plt.axhline(0, color='red', linestyle='--')
plt.title("Residual Plot - Random Forest Regressor")
plt.xlabel("Nilai Prediksi")
plt.ylabel("Residual (Aktual - Prediksi)")
plt.grid(True)
plt.show()
```

2.2 VISUALISASI PERBANDINGAN MODEL

[ ]:
```
# ============================================================
#   Visualisasi Gabungan: Decision Tree vs Random Forest (Regresi)
# ============================================================

# Pastikan sudah punya:
# y_pred_dt  -> prediksi Decision Tree pada X_test
# y_pred_rf  -> prediksi Random Forest pada X_test

metrics_dt = {
    "R2": r2_score(y_test, y_pred_dt),
    "MAE": mean_absolute_error(y_test, y_pred_dt),
    "MSE": mean_squared_error(y_test, y_pred_dt),
    "RMSE": np.sqrt(mean_squared_error(y_test, y_pred_dt))
}

metrics_rf = {
    "R2": r2_score(y_test, y_pred_rf),
    "MAE": mean_absolute_error(y_test, y_pred_rf),
    "MSE": mean_squared_error(y_test, y_pred_rf),
    "RMSE": np.sqrt(mean_squared_error(y_test, y_pred_rf))
}

df_compare = pd.DataFrame([metrics_dt, metrics_rf], index=["Decision Tree",␣
 ↪"Random Forest"])
display(df_compare)

# Buat figure dengan 3 subplot (1 baris, 3 kolom)
fig, (ax1, ax2, ax3) = plt.subplots(ncols=3, figsize=(18, 4))
```

```
# 1) Prediksi vs Aktual (Decision Tree)
ax1.plot(np.array(y_test)[:20], label="Aktual", marker='o')
ax1.plot(np.array(y_pred_dt)[:20], label="Prediksi", marker='x')
ax1.set_title("Decision Tree - Prediksi vs Aktual (20 data)")
ax1.set_xlabel("Index Sampel")
ax1.set_ylabel("Nilai Target")
ax1.grid(True, linestyle="--", alpha=0.5)
ax1.legend()

# 2) Prediksi vs Aktual (Random Forest)
ax2.plot(np.array(y_test)[:20], label="Aktual", marker='o')
ax2.plot(np.array(y_pred_rf)[:20], label="Prediksi", marker='x')
ax2.set_title("Random Forest - Prediksi vs Aktual (20 data)")
ax2.set_xlabel("Index Sampel")
ax2.set_ylabel("Nilai Target")
ax2.grid(True, linestyle="--", alpha=0.5)
ax2.legend()

# 3) Grafik Bar Perbandingan Metrik
cols = df_compare.columns.tolist()
x = np.arange(len(cols))
width = 0.35

ax3.bar(x - width/2, df_compare.loc["Decision Tree"], width, label="Decision␣
 ↪Tree")
ax3.bar(x + width/2, df_compare.loc["Random Forest"], width, label="Random␣
 ↪Forest")

ax3.set_xticks(x)
ax3.set_xticklabels(cols)
ax3.set_ylabel("Score")
ax3.set_title("Perbandingan Kinerja Model (Regresi)")
ax3.legend()
ax3.grid(axis="y", linestyle="--", alpha=0.6)

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

2.3 Feature dengan score tertinggi

[ ]:
```
# ========================
# Feature Importance - Decision Tree
# ========================

selector_dt = grid_dt.best_estimator_.named_steps['selector']
```

```
feature_scores_dt = pd.DataFrame({
    'Feature': X.columns,
    'Score': selector_dt.scores_
}).sort_values(by='Score', ascending=False)

print("\n Feature Importance (Decision Tree - f_regression):")
display(feature_scores_dt)
# ========================
# Feature Importance - Random Forest
# ========================

rf_model = grid_rf.best_estimator_.named_steps['reg']

feature_importance_rf = pd.DataFrame({
    'Feature': X.columns,
    'Importance': rf_model.feature_importances_
}).sort_values(by='Importance', ascending=False)

print("\n Feature Importance (Random Forest):")
display(feature_importance_rf)
```

2.4 Model Terbaik

[ ]:
```
import pickle
from sklearn.metrics import r2_score

# Hitung R² pada data test
r2_dt = r2_score(y_test, y_pred_dt)
r2_rf = r2_score(y_test, y_pred_rf)

# Bandingkan model
if r2_rf >= r2_dt:
    best_model = best_rf
    best_name = "RandomForestRegressor"
else:
    best_model = best_dt
    best_name = "DecisionTreeRegressor"

# Simpan model terbaik
with open(f"model/BestModel_REG_{best_name}.pkl", "wb") as f:
    pickle.dump(best_model, f)

print(f"\n Model terbaik: {best_name}")
print(f" R² Decision Tree : {r2_dt:.4f}")
print(f" R² Random Forest : {r2_rf:.4f}")
print(f" File tersimpan sebagai: BestModel_REG_{best_name}.pkl")
```

Python Code

```python
import streamlit as st
import pandas as pd
import pickle
import os

# ==============================================================================
# KONFIGURASI HALAMAN
# ==============================================================================
st.set_page_config(
    page_title="Prediksi Performance Index (Ridge)",
    page_icon="🎓",
    layout="centered"
)

# ==============================================================================
# LOAD MODEL RIDGE
# ==============================================================================
# Pastikan path folder 'model/' sudah benar dan file ada di dalamnya
FILE_MODEL_RIDGE = 'model/BestModel_Ridge_KERAS.pkl'

@st.cache_resource
def load_model(filename):
    if not os.path.exists(filename):
        return None
    try:
        with open(filename, 'rb') as file:
            model = pickle.load(file)
        return model
    except Exception as e:
        st.error(f"Error memuat model {filename}: {e}")
        return None

model_ridge = load_model(FILE_MODEL_RIDGE)

# ==============================================================================
# HEADER
# ==============================================================================
st.title("🎓 Prediksi Performance Index Siswa")
st.write(f"""
Aplikasi ini menggunakan model **Ridge Regression** untuk memprediksi nilai performa siswa.
""")
st.divider()

# ==============================================================================
# FORM INPUT FITUR
# ==============================================================================
st.subheader("Masukkan Data Siswa")

col1, col2 = st.columns(2)

with col1:
```

```
        hours_studied = st.number_input(
            "Jam Belajar (Hours Studied)",
            min_value=0, max_value=24, value=5, step=1
        )

        previous_scores = st.number_input(
            "Nilai Sebelumnya (Previous Scores)",
            min_value=0, max_value=100, value=75, step=1
        )

        extracurricular = st.selectbox(
            "Ekstrakurikuler",
            options=["Yes", "No"]
        )

    with col2:
        sleep_hours = st.number_input(
            "Jam Tidur (Sleep Hours)",
            min_value=0, max_value=24, value=7, step=1
        )

        sample_papers = st.number_input(
            "Latihan Soal (Sample Papers)",
            min_value=0, max_value=100, value=2, step=1
        )

    # ==============================================================================
    # PRE-PROCESSING
    # ==============================================================================
    # Konversi input sesuai format training (Yes=1, No=0)
    extracurricular_value = 1 if extracurricular == "Yes" else 0

    input_data = {
        'Hours Studied': [hours_studied],
        'Previous Scores': [previous_scores],
        'Extracurricular Activities': [extracurricular_value],
        'Sleep Hours': [sleep_hours],
        'Sample Question Papers Practiced': [sample_papers]
    }

    input_df = pd.DataFrame(input_data)

    # ==============================================================================
    # PREDIKSI DAN HASIL
    # ==============================================================================
    st.markdown("---")

    if st.button("Prediksi", type="primary"):
        # Cek ketersediaan model
        if model_ridge is None:
            st.error(f"File model '{FILE_MODEL_RIDGE}' tidak ditemukan. Pastikan file .p
kl ada di folder 'model/'.")
        else:
            try:
                # Prediksi menggunakan Ridge
```

```
            prediction = model_ridge.predict(input_df)
            hasil_prediksi = prediction[0] if hasattr(prediction, '__iter__') else p
rediction

            # Tampilkan Hasil
            st.success("Hasil Prediksi")
            st.metric(
                label="Performance Index",
                value=f"{hasil_prediksi:.2f}",
                delta=None
            )

            # Tambahan Info Visual
            if hasil_prediksi >= 80:
                st.info("Kategori: Sangat Baik 🌟")
            elif hasil_prediksi >= 60:
                st.info("Kategori: Baik 👍")
            else:
                st.warning("Kategori: Perlu Peningkatan ⚠️")

        except Exception as e:
            st.error(f"Terjadi error saat melakukan prediksi: {e}")
            st.write("Tips: Pastikan nama dan urutan kolom input sama persis dengan
saat model dilatih.")
```