**Laporan UAS *Take-Home* Mata Kuliah Aplikasi Data Scientist**

**Prediksi Harga Rumah di Jakarta Selatan: Analisis Komparatif Kinerja Model Regresi Linear, Decision Tree, dan Random Forest**

**Dosen Pengampu: Ledy Elsera Astrianty, S.Kom., M.Kom.**



**Disusun oleh:**

**Lathif Ramadhan (5231811022)**

**PROGRAM STUDI SAINS DATA**

**FAKULTAS SAINS DAN TEKNOLOGI**

**UNIVERSITAS TEKNOLOGI YOGYAKARTA**

**YOGYAKARTA**

**2025**

# Daftar Isi

[Daftar Isi II](#_Toc202119128)

[1. Deskripsi Dataset 1](#_Toc202119129)

[1.1. Latar Belakang dan Sumber Data 1](#_Toc202119130)

[1.2. Struktur dan Karakteristik Dataset 1](#_Toc202119131)

[1.3. Deskripsi Fitur (Atribut) 1](#_Toc202119132)

[2. Implementasi Kode dan Hasil 2](#_Toc202119133)

[2.1. Persiapan Lingkungan dan Import Library 2](#_Toc202119134)

[2.2. Memuat dan Eksplorasi Data (Exploratory Data Analysis - EDA) 2](#_Toc202119135)

[2.2.1. Memuat Data 3](#_Toc202119136)

[2.2.2. Analisis Deskriptif Awal 3](#_Toc202119137)

[2.2.3. Penanganan Data Duplikat 4](#_Toc202119138)

[2.2.4. Menangani Nilai yang Hilang 4](#_Toc202119139)

[2.2.5. Melihat Distribusi Fitur Numerik dan Kategorikal 5](#_Toc202119140)

[2.2.6. Penanganan Fitur Tidak Relevan (Zero Variance) 7](#_Toc202119141)

[2.2.7. Melihat Nilai Korelasi Antar Fitur 8](#_Toc202119142)

[2.2.8. Penanganan Outlier 9](#_Toc202119143)

[2.3. Feature Engineering 10](#_Toc202119144)

[2.4. Pra-pemrosesan Data 12](#_Toc202119145)

[2.4.1. Definisi Variabel Fitur (X) dan Target (y) 13](#_Toc202119146)

[2.4.2. Pembangunan Pipeline Pra-pemrosesan 13](#_Toc202119147)

[2.5. Pemisahan Fitur dan Target, Serta Data Latih & Uji 14](#_Toc202119148)

[2.6. Pemodelan, Pelatihan, dan Evaluasi 15](#_Toc202119149)

[2.6.1. Model 1: Regresi Linear Sederhana 15](#_Toc202119150)

[2.6.2. Model 2: Regresi Linear Berganda 16](#_Toc202119151)

[2.6.3. Model 3: Decision Tree Regressor 17](#_Toc202119152)

[2.6.4. Model 4: Random Forest Regressor 20](#_Toc202119153)

[2.7. Ringkasan Hasil Evaluasi Model 22](#_Toc202119154)

[3. Kesimpulan Akhir 23](#_Toc202119155)

[3.1. Ringkasan Kinerja Model 23](#_Toc202119156)

[3.2. Pemilihan Model Terbaik dan Justifikasi 23](#_Toc202119157)

# Deskripsi Dataset

## 1.1. Latar Belakang dan Sumber Data

Dalam analisis ini, dataset yang digunakan adalah "**Daftar Harga Rumah Jakarta Selatan**" yang bersumber dari platform Kaggle. Dataset ini merupakan kompilasi data harga properti yang dikumpulkan dari berbagai situs jual beli real estat terkemuka, salah satunya adalah rumah123.com. Tujuan utama dari analisis dataset ini adalah untuk membangun model prediksi yang akurat guna mengestimasi harga rumah berdasarkan atribut-atribut yang tersedia.

Berikut lini sumber dataset dari situs Kaggle: *https://www.kaggle.com/datasets/wisnuanggara/daftar-harga-rumah*

## 1.2. Struktur dan Karakteristik Dataset

Dataset awal terdiri dari **1000 baris** data dan **7 kolom** (fitur). Setiap baris merepresentasikan satu unit properti, sedangkan setiap kolom mendeskripsikan atribut dari properti tersebut. Kolom-kolom ini akan menjadi dasar dalam proses eksplorasi data dan pemodelan machine learning.

## 1.3. Deskripsi Fitur (Atribut)

Dataset ini mencakup 7 fitur utama yang dapat dikelompokkan menjadi fitur target, numerik, dan kategorikal. Berikut adalah rincian dari setiap fitur:

* **HARGA** (Fitur Target/Numerik): Merepresentasikan harga properti dalam mata uang Rupiah. Kolom ini akan menjadi variabel dependen yang akan diprediksi oleh model.
* **LT** (Fitur Prediktor/Numerik): Menunjukkan Luas Tanah dari properti, diukur dalam satuan meter persegi (m²).
* **LB** (Fitur Prediktor/Numerik): Menunjukkan Luas Bangunan dari properti, diukur dalam satuan meter persegi (m²).
* **JKT** (Fitur Prediktor/Numerik): Merepresentasikan Jumlah Kamar Tidur yang dimiliki oleh properti.
* **JKM** (Fitur Prediktor/Numerik): Merepresentasikan Jumlah Kamar Mandi yang dimiliki oleh properti.
* **GRS** (Fitur Prediktor/Kategorikal): Memberikan informasi ketersediaan garasi pada properti, dengan nilai kategorikal "ADA" atau "TIDAK ADA".
* **KOTA** (Fitur Prediktor/Kategorikal): Menyatakan lokasi kota properti. Pada dataset ini, seluruh data memiliki nilai "JAKSEL".

# Implementasi Kode dan Hasil

## 2.1. Persiapan Lingkungan dan Import 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, GridSearchCV

from sklearn.linear\_model import LinearRegression

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

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

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import PolynomialFeatures, StandardScaler, OneHotEncoder

Langkah pertama dalam proses analisis adalah mempersiapkan lingkungan kerja dengan mengimpor seluruh library dan modul Python yang dibutuhkan. Setiap library memiliki peran spesifik dalam alur kerja analisis data dan pemodelan machine learning. Berikut adalah rincian library utama yang digunakan:

* **Pandas (pd):** Library fundamental untuk manipulasi dan analisis data. Digunakan untuk memuat dataset dari file Excel, membuat struktur data dalam bentuk DataFrame, serta melakukan operasi pembersihan dan transformasi data.
* **NumPy (np):** Library penting untuk komputasi numerik. Berperan dalam operasi matematika tingkat lanjut, seperti transformasi logaritmik, dan untuk menangani array numerik secara efisien.
* **Matplotlib (plt) dan Seaborn (sns):** Dua library utama untuk visualisasi data. Matplotlib menyediakan fondasi untuk membuat plot statis seperti histogram, sementara Seaborn digunakan untuk membuat visualisasi statistik yang lebih menarik dan informatif, seperti heatmap matriks korelasi dan box plot.
* **Scikit-learn (sklearn):** Library inti untuk implementasi *machine learning* di Python. Dari library ini, beberapa modul spesifik digunakan, yaitu:
  + *model\_selection*: Untuk membagi dataset menjadi data latih dan data uji (train\_test\_split) serta untuk melakukan *hyperparameter tuning* dengan validasi silang (*GridSearchCV*).
  + *linear\_model*, *tree*, *ensemble*: Untuk mengimpor algoritma-algoritma model regresi yang akan dievaluasi, yaitu LinearRegression, DecisionTreeRegressor, dan RandomForestRegressor.
  + *metrics*: Untuk mengevaluasi kinerja model menggunakan metrik seperti mean\_absolute\_error (MAE) dan r2\_score (R-Squared).
  + *preprocessing*: Berisi alat untuk pra-pemrosesan data seperti StandardScaler (untuk standardisasi fitur), OneHotEncoder (untuk mengubah fitur kategorikal menjadi numerik), dan PolynomialFeatures (untuk membuat fitur interaksi).
  + *compose* dan *pipeline*: Untuk membangun alur kerja pra-pemrosesan dan pemodelan yang sistematis, bersih, dan efisien menggunakan ColumnTransformer dan Pipeline.

Dengan mengimpor semua komponen ini di awal, proses analisis dapat berjalan dengan lancar dari tahap eksplorasi hingga evaluasi model.

## 2.2. Memuat dan Eksplorasi Data (Exploratory Data Analysis - EDA)

### 2.2.1. Memuat Data

Data yang dianalisis dalam proyek ini bersumber dari file Excel HARGA RUMAH JAKSEL.xlsx yang telah disimpan pada sebuah repositori GitHub untuk kemudahan akses. Proses pemuatan data dilakukan menggunakan fungsi pd.read\_excel() dari library Pandas, dengan mengarahkan langsung ke URL mentah dari file tersebut.

Untuk memastikan data dimuat dengan struktur yang benar, parameter skiprows=[0] digunakan untuk mengabaikan baris pertama pada file, dan parameter header=0 ditetapkan agar baris selanjutnya secara otomatis diinterpretasikan sebagai nama-nama kolom DataFrame. Hasilnya, data mentah berhasil dimuat ke dalam DataFrame df yang terstruktur dan siap untuk dieksplorasi lebih lanjut.

path = 'https://github.com/LatiefDataVisionary/data-science-application-college-task/raw/refs/heads/main/datasets/HARGA%20RUMAH%20JAKSEL.xlsx'

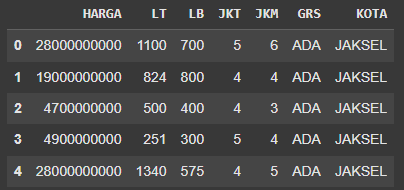
# Melewati baris pertama dan menggunakan baris kedua sebagai nama kolom

df = pd.read\_excel(path, skiprows=[0], header=0)

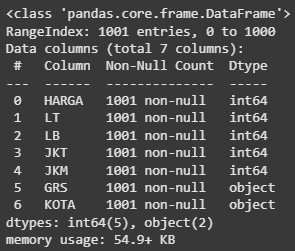
### 2.2.2. Analisis Deskriptif Awal

Tahap ini untuk melihat gambaran umum dataset seperti 5 baris pertama, informasi tipe data, dan statistik dasar.

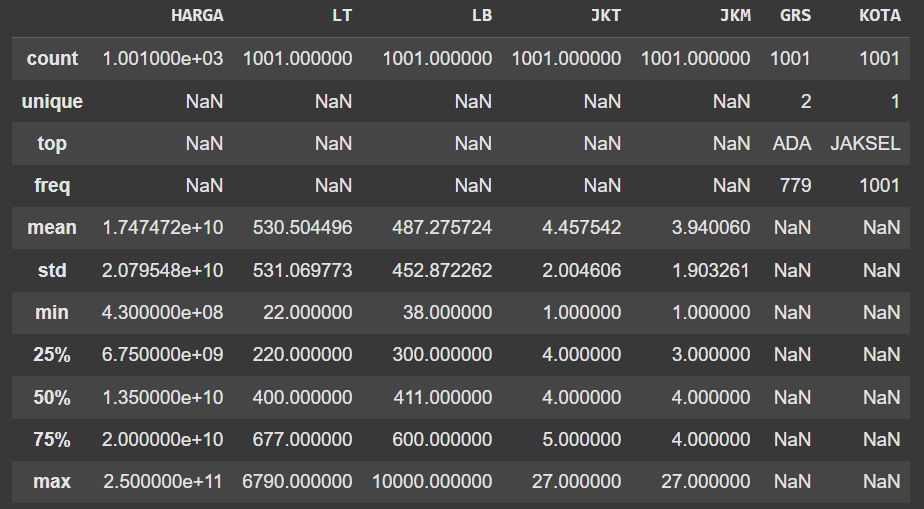
display(df.head())



df.info()



df.describe(include='all')



### 2.2.3. Penanganan Data Duplikat

# cek dan hapus duplikat

duplicated\_rows = df.duplicated().sum()

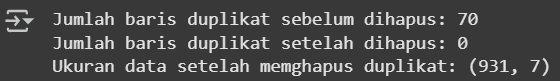
print(f'Jumlah baris duplikat sebelum dihapus: {duplicated\_rows}')

# menghapus baris duplicate

df.drop\_duplicates(inplace=True)

print(f'Jumlah baris duplikat setelah dihapus: {df.duplicated().sum()}')

print(f'Ukuran data setelah memghapus duplikat: {df.shape}')



Untuk menjaga integritas dan kualitas data, dilakukan proses identifikasi dan penghapusan data duplikat. Data duplikat dapat memberikan bobot yang tidak semestinya pada observasi tertentu dan mengganggu performa model. Hasil pemeriksaan awal menunjukkan adanya **70 baris data yang duplikat**.

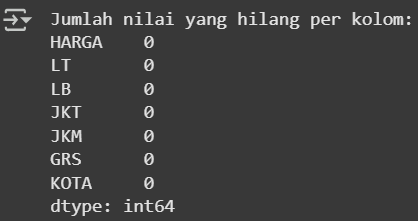
Dengan menggunakan fungsi df.drop\_duplicates(), semua baris yang berulang tersebut berhasil dihapus dari dataset. Hasilnya, dataset yang kini bersih dari duplikasi memiliki ukuran **931 baris dan 7 kolom**, dan siap untuk analisis lebih lanjut.

### 2.2.4. Menangani Nilai yang Hilang

# Cek jumlah nilai yang hilang per kolom

print("Jumlah nilai yang hilang per kolom:")

print(df.isnull().sum())



Langkah pembersihan data selanjutnya adalah memeriksa keberadaan nilai yang hilang atau *null* pada setiap kolom. Nilai yang hilang dapat secara signifikan memengaruhi kualitas dan performa model, sehingga perlu diidentifikasi dan ditangani dengan tepat.

Untuk melakukan pemeriksaan, metode isnull().sum() dari Pandas digunakan pada DataFrame. Metode ini menghitung jumlah nilai *null* untuk setiap kolom. Sebagaimana ditunjukkan oleh output di atas, hasil pemeriksaan menunjukkan bahwa **tidak terdapat nilai yang hilang** pada seluruh kolom dalam dataset. Hal ini menandakan bahwa data sudah cukup lengkap dan tidak memerlukan teknik imputasi atau penghapusan baris terkait nilai yang hilang.

### 2.2.5. Melihat Distribusi Fitur Numerik dan Kategorikal

# Menganalisis distribusi fitur numerik (HARGA, LT, LB, JKT, JKM)

numerical\_features = ['HARGA', 'LT', 'LB', 'JKT', 'JKM']

# Hitung jumlah baris dan kolom untuk subplot

n\_cols\_num = 3 # Jumlah kolom subplot yang diinginkan untuk numerik

n\_rows\_num = (len(numerical\_features) + n\_cols\_num - 1) // n\_cols\_num # menghitung jumlah baris yang dibutuhkan

plt.figure(figsize=(5 \* n\_cols\_num, 4 \* n\_rows\_num))

plt.suptitle('Distribusi Fitur Numerik', y=1.02, ha='center', fontsize='xx-large')

for i, col in enumerate(numerical\_features):

plt.subplot(n\_rows\_num, n\_cols\_num, i + 1)

df[col].hist(bins=30)

plt.title(f'Distribusi {col}')

plt.xlabel(col)

plt.ylabel('Frekuensi')

plt.tight\_layout()

plt.show()

# Menganalisis distribusi fitur kategorikal (GRS, KOTA)

categorical\_features = ['GRS', 'KOTA']

# menghitung jumlah baris dan kolom untuk subplot

n\_cols = 2 # Jumlah kolom subplot yang diinginkan

n\_rows = (len(categorical\_features) + n\_cols - 1) // n\_cols # menghitung jumlah baris yang dibutuhkan

plt.figure(figsize=(8 \* n\_cols, 5 \* n\_rows))

plt.suptitle('Distribusi Fitur Kategorikal', y=1.02, ha='center', fontsize='xx-large')

for i, col in enumerate(categorical\_features):

plt.subplot(n\_rows, n\_cols, i + 1)

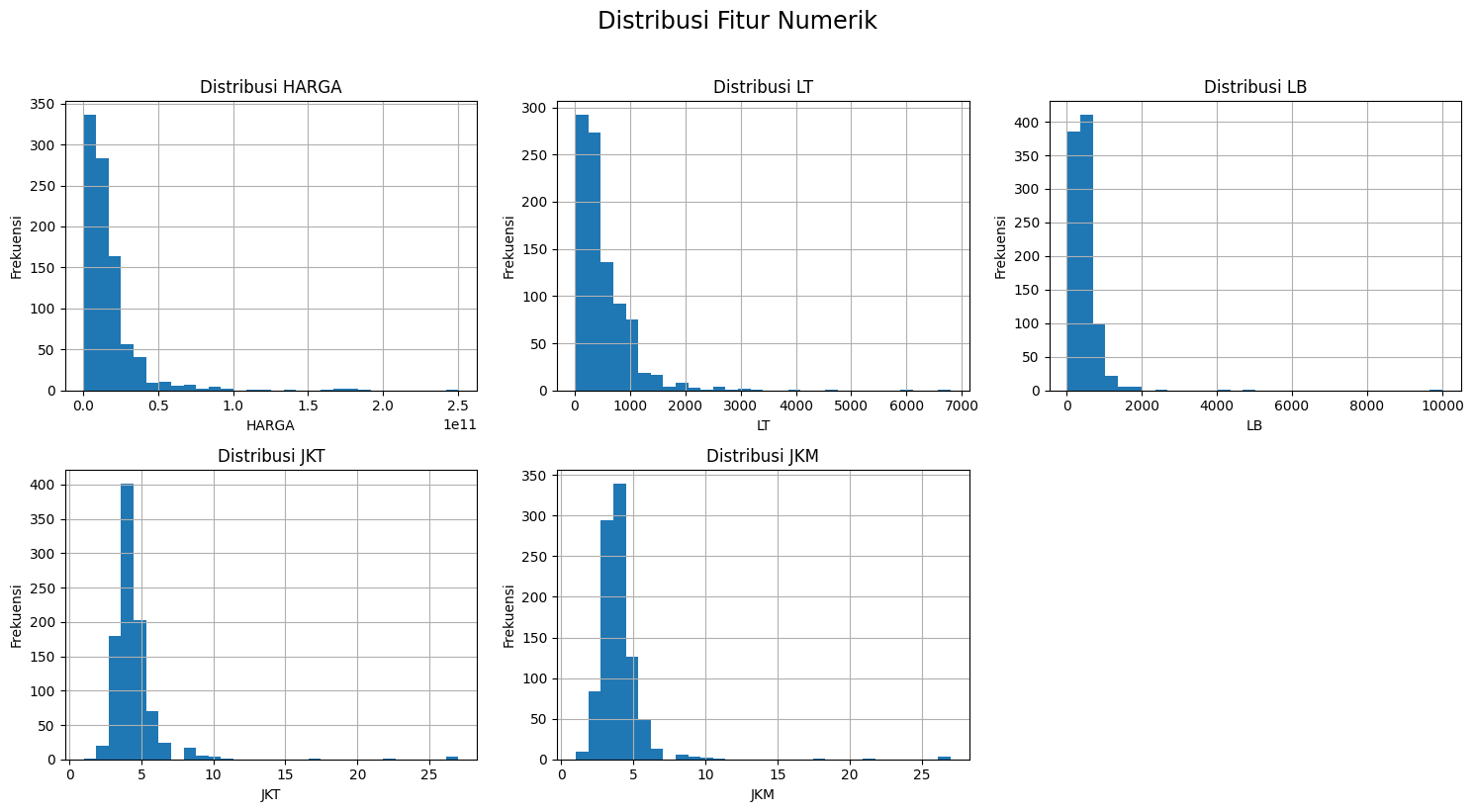
sns.countplot(data=df, x=col)

plt.title(f'Distribusi Fitur Kategorikal: {col}')

plt.ylabel('Frekuensi')

plt.tight\_layout()

plt.show()



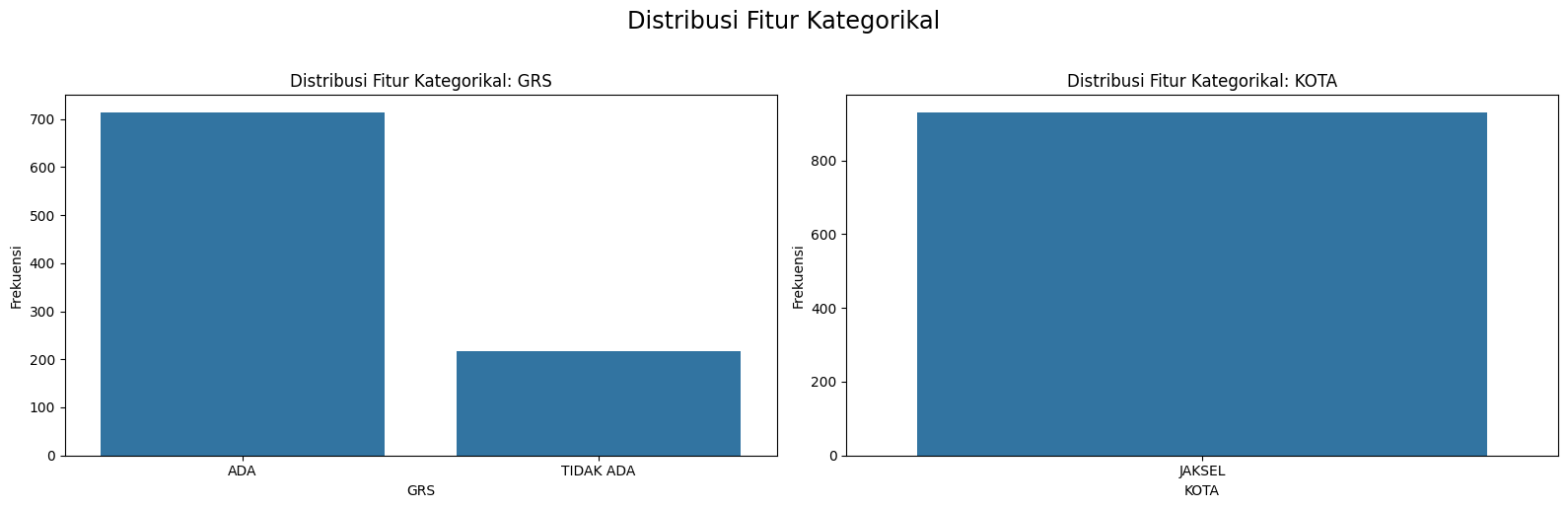
Visualisasi distribusi fitur numerik dilakukan menggunakan histogram untuk melihat sebaran frekuensi dari setiap nilai. Berdasarkan plot histogram di atas, dapat ditarik beberapa pengamatan penting:

* **Distribusi Miring (Right-Skewed)**

Fitur-fitur utama seperti HARGA, LT (Luas Tanah), dan LB (Luas Bangunan) menunjukkan distribusi yang sangat miring ke kanan. Ini berarti sebagian besar data terkonsentrasi pada nilai-nilai yang lebih rendah, namun terdapat sejumlah kecil data dengan nilai yang sangat tinggi. Kemiringan ini menjadi dasar pertimbangan untuk melakukan transformasi logaritmik di tahap *feature engineering* agar distribusinya lebih mendekati normal, yang umumnya baik untuk model regresi.

* **Potensi Outlier**

Pada fitur JKT (Jumlah Kamar Tidur) dan JKM (Jumlah Kamar Mandi), terlihat ada beberapa batang frekuensi yang sangat jauh dari konsentrasi utama data (misalnya, jumlah kamar lebih dari 10). Hal ini mengindikasikan adanya *outlier* atau pencilan yang perlu dianalisis dan ditangani lebih lanjut.



Untuk fitur kategorikal, proporsi dari setiap kategori divisualisasikan menggunakan diagram batang (*bar plot*).Dari visualisasi fitur kategorikal, diperoleh wawasan sebagai berikut:

* **Fitur GRS (Garasi)**

Sebagian besar rumah dalam dataset ini (sekitar 700+ dari 931 data bersih) memiliki garasi ("ADA"), sedangkan sisanya tidak memiliki garasi ("TIDAK ADA"). Distribusi yang tidak seimbang ini tetap informatif dan akan ditangani menggunakan teknik *one-hot encoding* pada tahap pra-pemrosesan.

* **Fitur KOTA (Varians Nol)**

Pengamatan yang paling signifikan adalah pada fitur KOTA. Plot menunjukkan bahwa **100% data** dalam kolom ini memiliki nilai tunggal, yaitu "JAKSEL". Fitur dengan varians nol seperti ini tidak memberikan informasi apa pun yang dapat digunakan model untuk membedakan data. Oleh karena itu, fitur ini akan dihapus pada tahap pembersihan data selanjutnya.

# Membuat pie plot untuk fitur kategorikal

categorical\_features = ['GRS', 'KOTA']

# menghitung jumlah baris dan kolom untuk subplot

n\_cols = 2 # Jumlah kolom subplot yang diinginkan

n\_rows = (len(categorical\_features) + n\_cols - 1) // n\_cols # menghitung jumlah baris yang dibutuhkan

plt.figure(figsize=(8 \* n\_cols, 8 \* n\_rows))

plt.suptitle('Proporsi Fitur Kategorikal', y=1.02, ha='center', fontsize='xx-large')

for i, col in enumerate(categorical\_features):

plt.subplot(n\_rows, n\_cols, i + 1)

df[col].value\_counts().plot.pie(autopct='%1.1f%%', startangle=90)

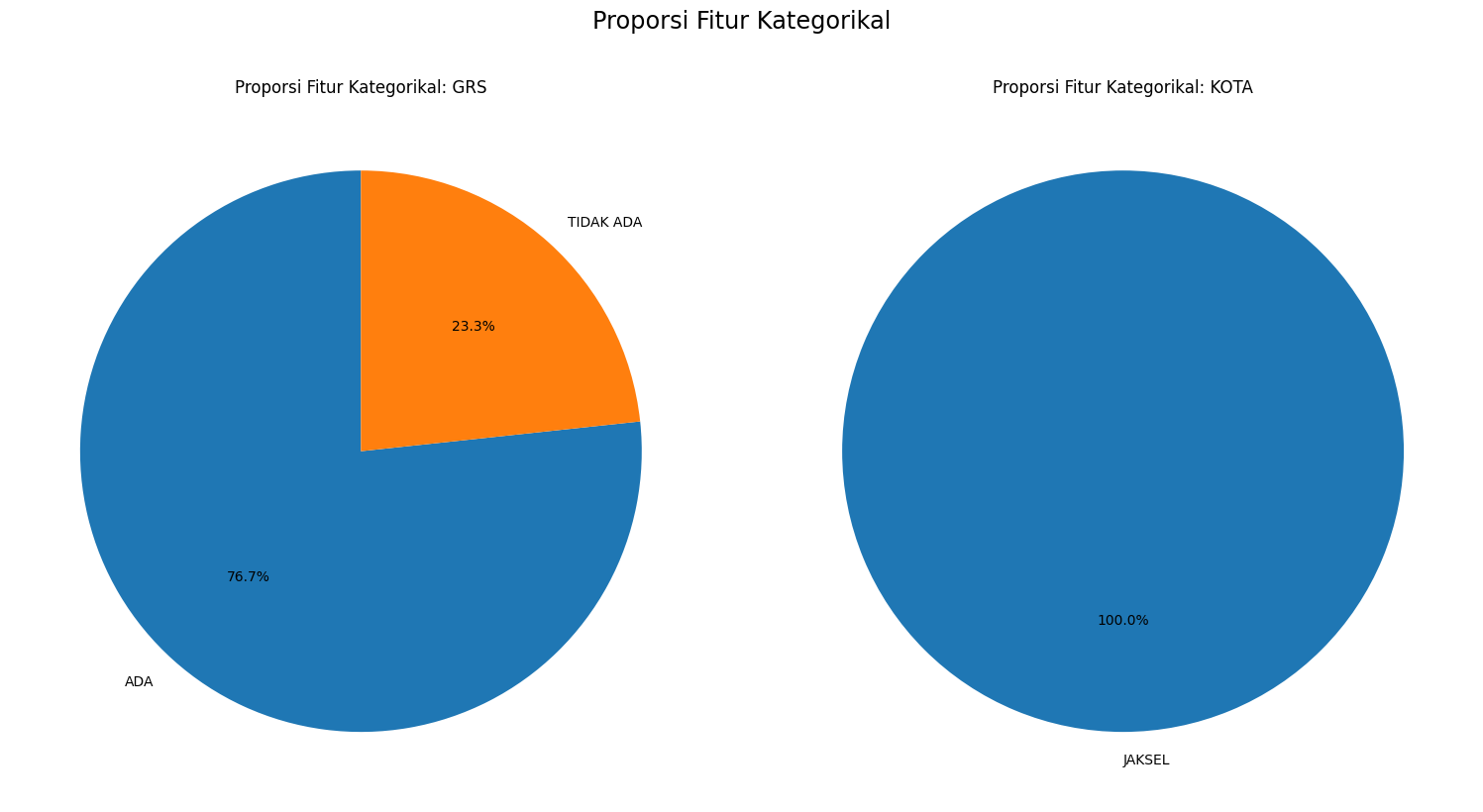
plt.title(f'Proporsi Fitur Kategorikal: {col}')

plt.ylabel('') # Menghilangkan label y default

plt.tight\_layout()

plt.show()

Untuk memahami karakteristik dan sebaran data secara lebih mendalam, dilakukan visualisasi distribusi untuk fitur numerik dan fitur kategorikal. Analisis ini sangat penting untuk mengidentifikasi pola, kemiringan (*skewness*), serta potensi adanya pencilan (*outlier*) atau fitur yang tidak memiliki variasi.



Untuk memahami karakteristik dan sebaran data secara lebih mendalam, dilakukan visualisasi distribusi untuk fitur numerik dan fitur kategorikal. Analisis ini sangat penting untuk mengidentifikasi pola, kemiringan (*skewness*), serta potensi adanya pencilan (*outlier*) atau fitur yang tidak memiliki variasi.

### 2.2.6. Penanganan Fitur Tidak Relevan (Zero Variance)

Berdasarkan analisis distribusi plot sebelumnya, teridentifikasi bahwa fitur KOTA memiliki varians nol, di mana semua data memiliki nilai yang sama yaitu "JAKSEL". Fitur semacam ini tidak memberikan informasi variatif yang dapat digunakan oleh model *machine learning* untuk belajar atau membuat perbedaan antar data. Dengan kata lain, fitur dengan varians nol tidak memiliki daya prediksi.

Untuk mengonfirmasi temuan ini secara programatik dan sebagai bagian dari langkah pembersihan data, dilakukan pengecekan jumlah nilai unik pada setiap kolom. Kode di atas secara eksplisit menunjukkan bahwa kolom KOTA memang hanya memiliki **1 nilai unik**.

Menyertakan fitur ini dalam proses pemodelan tidak akan memberikan manfaat dan justru menambah kompleksitas yang tidak perlu. Oleh karena itu, untuk menyederhanakan model dan meningkatkan efisiensi komputasi tanpa mengurangi kualitas prediktif, fitur KOTA dihapus dari DataFrame. Setelah penghapusan, ukuran dataset menjadi **931 baris dan 6 kolom**, siap untuk analisis korelasi dan tahap pemodelan selanjutnya.

print('Nilai unik di setiap kolom (menampilkan 15 nilai unik pertama untuk efisiensi):')

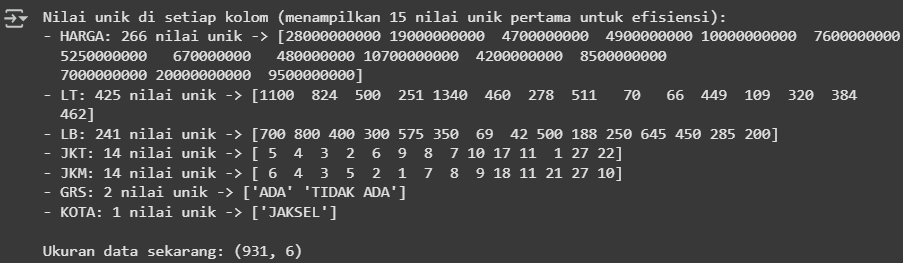
for col in df.columns:

print(f'- {col}: {df[col].nunique()} nilai unik -> {df[col].unique()[:15]}')

# Kolom 'KOTA' hanya memiliki 1 nilai unik, shg tdk informatif, maka dihapus.

df.drop(['KOTA'], axis=1, inplace=True)

print('\nUkuran data sekarang:', df.shape)



### 2.2.7. Melihat Nilai Korelasi Antar Fitur

Setelah data dibersihkan, langkah selanjutnya adalah menganalisis hubungan antar fitur, khususnya hubungan antara fitur-fitur prediktor dengan fitur target (HARGA). Analisis ini bertujuan untuk mendapatkan wawasan awal mengenai variabel mana yang paling potensial untuk memprediksi harga rumah.

# Korelasi antar fitur numerik

correlation\_matrix = df[numerical\_features].corr()

plt.figure(figsize=(10, 8))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)

plt.title('Matrix Korelasi Fitur Numerik')

plt.show()

# Box plot atau violin plot utk melihat hubungan fitur kategorikal dg HARGA

categorical\_features\_for\_box = ['GRS']

n\_cols\_box = 1 # Adjust to 1 as only one categorical feature remains

n\_rows\_box = (len(categorical\_features\_for\_box) + n\_cols\_box - 1) // n\_cols\_box

plt.figure(figsize=(8 \* n\_cols\_box, 6 \* n\_rows\_box))

plt.suptitle('Hubungan antara Fitur Kategorikal dan HARGA', y=1.02, ha='center', fontsize='xx-large')

for i, col in enumerate(categorical\_features\_for\_box):

plt.subplot(n\_rows\_box, n\_cols\_box, i + 1)

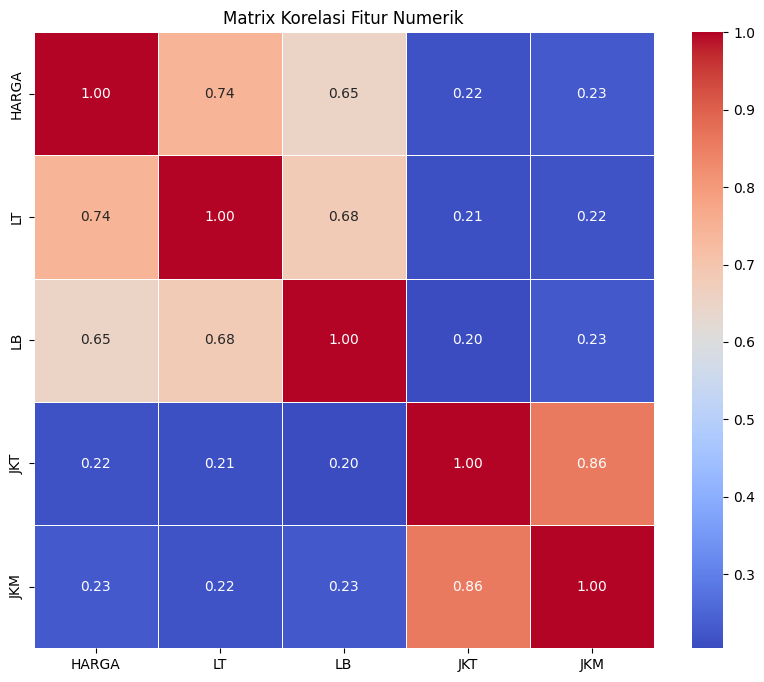
sns.boxplot(data=df, x=col, y='HARGA')

plt.title(f'{col} vs HARGA')

plt.tight\_layout()

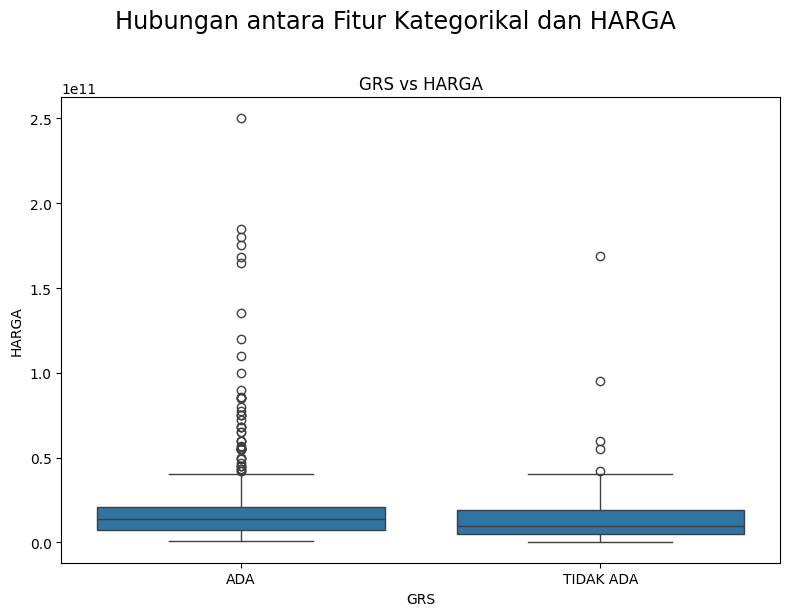
plt.show()

Untuk mengukur kekuatan hubungan linear antar fitur-fitur numerik, sebuah matriks korelasi dibuat dan divisualisasikan menggunakan *heatmap*. Heatmap ini menyajikan koefisien korelasi Pearson, dengan nilai mendekati 1 (warna merah tua) menunjukkan korelasi positif yang kuat, dan nilai mendekati -1 (warna biru tua) menunjukkan korelasi negatif yang kuat.



Dari heatmap matriks korelasi, dapat disimpulkan beberapa hal:

* **Korelasi Positif Kuat dengan HARGA**: Fitur LT (Luas Tanah) dan LB (Luas Bangunan) memiliki korelasi positif yang kuat dengan HARGA, dengan koefisien masing-masing sebesar **0.74** dan **0.65**. Hal ini sesuai dengan intuisi bahwa semakin besar luas tanah dan bangunan, maka semakin tinggi pula harga rumah.
* **Multikolinearitas Antar Prediktor**: Terdapat korelasi yang cukup kuat antara LT dan LB (0.68), yang menandakan adanya potensi multikolinearitas. Selain itu, fitur JKT (Jumlah Kamar Tidur) dan JKM (Jumlah Kamar Mandi) menunjukkan korelasi yang sangat tinggi satu sama lain (**0.86**). Multikolinearitas yang tinggi antar fitur prediktor ini perlu menjadi perhatian karena dapat memengaruhi interpretasi koefisien pada model regresi linear.
* **Korelasi Rendah**: Fitur JKT dan JKM memiliki korelasi yang relatif rendah dengan HARGA (0.22 dan 0.23).



Untuk menganalisis hubungan antara fitur kategorikal GRS (Garasi) dengan fitur target numerik HARGA, visualisasi *box plot* digunakan. Box plot efektif untuk membandingkan distribusi harga antara dua kategori (rumah yang "ADA" garasi dan yang "TIDAK ADA"). Berdasarkan *box plot* di atas, terlihat bahwa:

* Distribusi harga untuk kedua kategori (ADA dan TIDAK ADA garasi) terlihat cukup mirip. Median (garis tengah kotak) dari kedua kelompok tidak menunjukkan perbedaan yang signifikan secara visual.
* Terdapat banyak pencilan (*outlier*) harga pada kategori rumah yang memiliki garasi. Ini menunjukkan adanya rumah-rumah dengan harga sangat tinggi yang memiliki fasilitas garasi, sesuai dengan ekspektasi.

Meskipun secara visual perbedaannya tidak terlalu dramatis, fitur GRS tetap akan disertakan dalam pemodelan untuk melihat apakah dapat memberikan kontribusi prediktif setelah diproses.

### 2.2.8. Penanganan Outlier

# VISUALISASI OUTLIER DENGAN BOX PLOT

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

sns.boxplot(y=df['JKT'])

plt.title('Box Plot JKT (Kamar Tidur)')

plt.subplot(1, 2, 2)

sns.boxplot(y=df['JKM'])

plt.title('Box Plot JKM (Kamar Mandi)')

plt.tight\_layout()

plt.show()

# PENANGANAN OUTLIER MENGGUNAKAN METODE IQR

# Kita akan membatasi nilai yang terlalu tinggi ke batas atas (upper bound)

print("Nilai JKT sebelum penanganan outlier:", df['JKT'].max())

print("Nilai JKM sebelum penanganan outlier:", df['JKM'].max())

for col in ['JKT', 'JKM']:

Q1 = df[col].quantile(0.25)

Q3 = df[col].quantile(0.75)

IQR = Q3 - Q1

upper\_bound = Q3 + 1.5 \* IQR

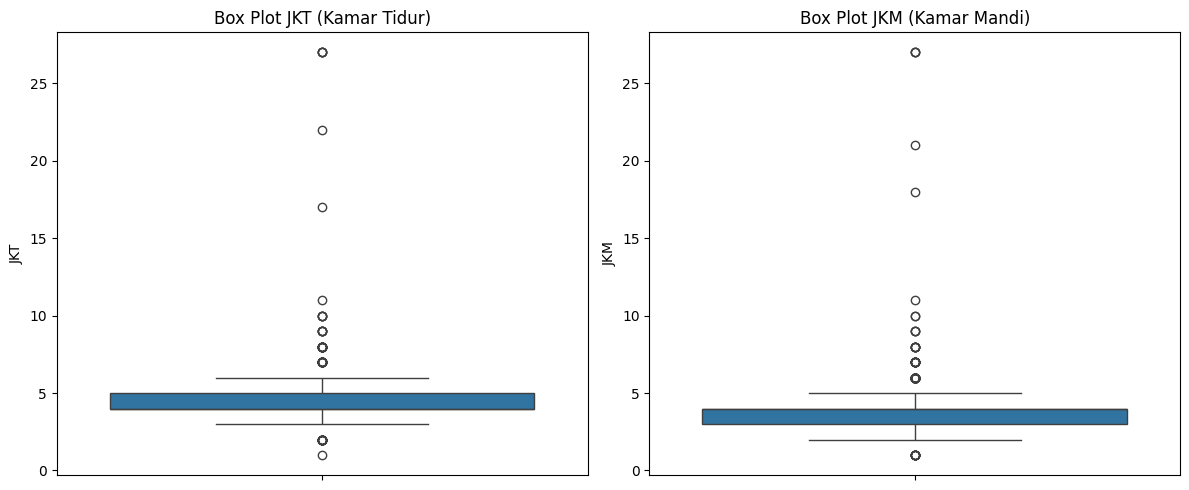
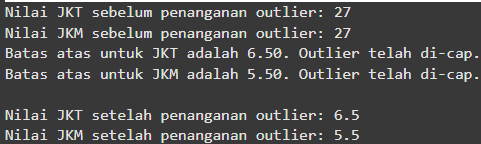
# Cap aoutlier (mengganti nilai di atas batas atas dengan nilai batas atas)

df[col] = np.where(df[col] > upper\_bound, upper\_bound, df[col])

print(f"Batas atas untuk {col} adalah {upper\_bound:.2f}. Outlier telah di-cap.")

print("\nNilai JKT setelah penanganan outlier:", df['JKT'].max())

print("Nilai JKM setelah penanganan outlier:", df['JKM'].max())



Berdasarkan analisis visual, fitur JKT (Jumlah Kamar Tidur) dan JKM (Jumlah Kamar Mandi) teridentifikasi memiliki *outlier* atau nilai ekstrem. Keberadaan *outlier* ini dapat mengganggu performa model regresi, sehingga perlu ditangani.

Metode yang digunakan adalah *capping* berbasis *Interquartile Range* (IQR), di mana nilai yang melebihi batas atas (dihitung dengan formula Q3 + 1.5 \* IQR) akan digantikan oleh nilai batas atas itu sendiri. Proses ini berhasil menangani nilai-nilai ekstrem, di mana nilai maksimum pada JKT disesuaikan menjadi **6.5** (dari sebelumnya 27) dan pada JKM menjadi **5.5** (dari sebelumnya 27). Dengan demikian, dataset menjadi lebih robust dan siap untuk tahap selanjutnya.

## 2.3. Feature Engineering

Setelah data dibersihkan, langkah selanjutnya adalah *feature engineering*, yaitu sebuah proses untuk membuat fitur-fitur baru dari fitur yang sudah ada atau mentransformasinya. Tujuannya adalah untuk menyediakan informasi yang lebih kaya dan bermakna bagi model *machine learning*, sehingga diharapkan dapat meningkatkan akurasi prediksi.

# Membuat fitur interaksi: LT \* LB

df['LT\_x\_LB'] = df['LT'] \* df['LB']

print("Menambahkan fitur LT\_x\_LB.")

# --- Modifikasi: Menambahkan fitur rasio dan total kamar ---

# Membuat fitur Rasio\_LB\_LT (menghindari pembagian dengan nol jika ada LT=0)

df['Rasio\_LB\_LT'] = df['LB'] / df['LT'].replace(0, np.nan) # Replace 0 with NaN to avoid division by zero

df['Rasio\_LB\_LT'].fillna(df['Rasio\_LB\_LT'].median(), inplace=True) # mengisi NaN dg median atau strategi lain

print("Menambahkan fitur Rasio\_LB\_LT.")

# Membuat fitur Total\_Kamar

df['Total\_Kamar'] = df['JKT'] + df['JKM']

print("Menambahkan fitur Total\_Kamar.")

# --- Akhir Modifikasi ---

# Menerapkan transformasi logaritmik pada fitur yang miring (skewed)

# Kita tambahkan sedikit nilai (misalnya 1) sebelum log untuk menghindari log(0)

# HARGA juga ditransformasi karena merupakan target regresi

for col in ['LT', 'LB', 'LT\_x\_LB', 'HARGA', 'Rasio\_LB\_LT', 'Total\_Kamar']: # Include new features for potential log transformation

# kita periksa apakah ada nilai non-positif sebelum transformasi log

# Perhatikan bahwa kita tidak lagi memiliki baris yang merusak data numerik

# Jadi kita tidak perlu .any().any(), cukup .any()

if (df[col] <= 0).any():

print(f"Kolom '{col}' mengandung nilai non-positif. Transformasi log mungkin memerlukan penyesuaian.")

# Alternatif: menerapkan log1p (log(1+x)) yang lebih aman untuk nilai kecil

df[col + '\_log'] = np.log1p(df[col])

print(f"Menerapkan np.log1p pada kolom {col}.")

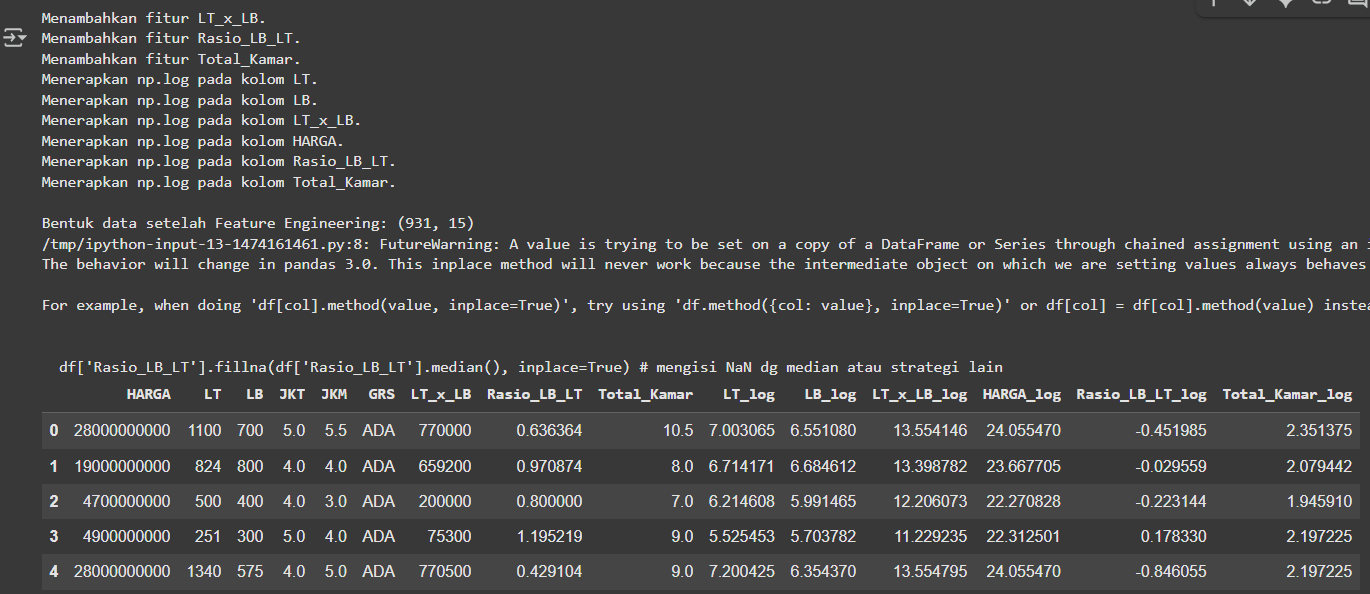
else:

df[col + '\_log'] = np.log(df[col])

print(f"Menerapkan np.log pada kolom {col}.")

print("\nBentuk data setelah Feature Engineering:", df.shape)

display(df.head())



Tiga fitur baru diciptakan berdasarkan pemahaman domain tentang pasar properti:

* **LT\_x\_LB**: Fitur interaksi yang merupakan hasil perkalian antara Luas Tanah (LT) dengan Luas Bangunan (LB). Fitur ini bertujuan untuk menangkap efek gabungan dari total area properti.
* **Total\_Kamar**: Merupakan penjumlahan dari JKT dan JKM, yang merepresentasikan total fasilitas ruang fungsional (kamar) pada properti.
* **Rasio\_LB\_LT**: Dibuat dengan membagi Luas Bangunan (LB) dengan Luas Tanah (LT). Fitur ini merepresentasikan seberapa efisien pemanfaatan lahan untuk bangunan.

Berdasarkan temuan pada tahap EDA bahwa beberapa fitur memiliki distribusi yang miring (*skewed*), transformasi logaritmik (np.log) diterapkan pada fitur-fitur tersebut, termasuk variabel target HARGA. Tujuan dari transformasi ini adalah untuk menormalkan distribusi data. Hal ini sangat bermanfaat, terutama untuk model linear, karena dapat membantu model lebih baik dalam menangkap pola hubungan linear antara variabel.

# Korelasi antar fitur numerik setelah feature engineering

# Identifikasi fitur numerik yang relevan setelah rekayasa fitur

numerical\_features\_post\_fe = ['HARGA\_log', 'LT\_log', 'LB\_log', 'JKT', 'JKM', 'LT\_x\_LB\_log', 'Rasio\_LB\_LT\_log', 'Total\_Kamar\_log', 'Rasio\_LB\_LT']

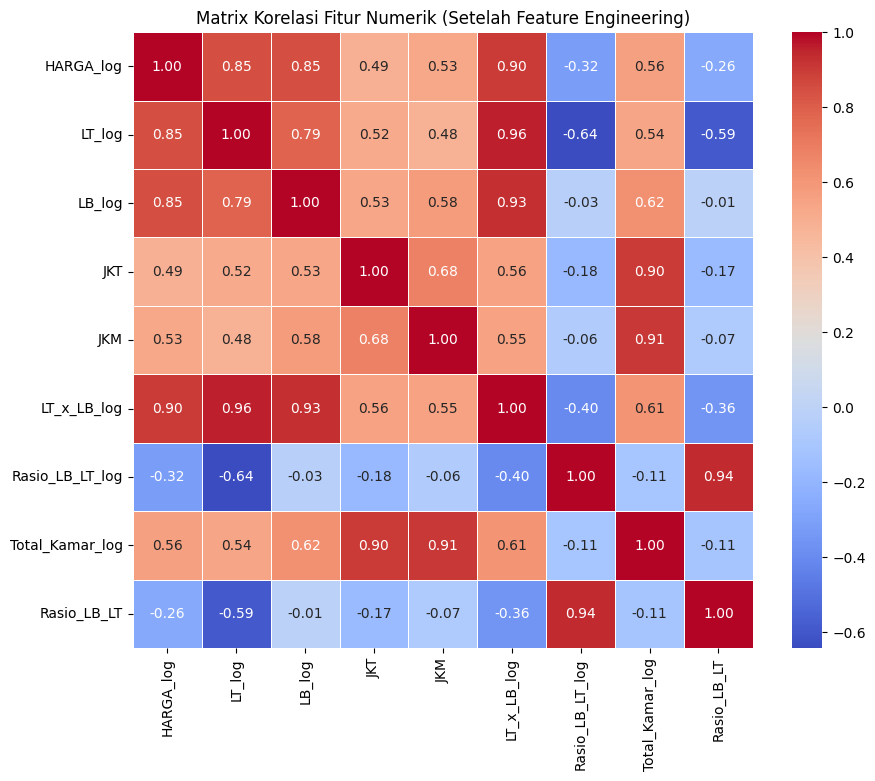
correlation\_matrix\_log = df[numerical\_features\_post\_fe].corr()

plt.figure(figsize=(10, 8))

sns.heatmap(correlation\_matrix\_log, annot=True, cmap='coolwarm', fmt=".2f", linewidths=".5")

plt.title('Matrix Korelasi Fitur Numerik (Setelah Feature Engineering)')

plt.show()



Untuk mengevaluasi dampak dari *feature engineering*, analisis korelasi kembali dilakukan, kali ini dengan menyertakan fitur-fitur baru yang sudah ditransformasi secara logaritmik.

Hasil dari heatmap baru ini memberikan beberapa wawasan yang sangat menarik:

* **Peningkatan Korelasi Signifikan:** Setelah transformasi log, korelasi antara fitur-fitur area dengan harga meningkat secara signifikan. LT\_log, LB\_log, dan fitur baru LT\_x\_LB\_log sekarang memiliki korelasi yang sangat kuat dengan HARGA\_log, yaitu sebesar **0.85**, **0.85**, dan **0.90** secara berurutan. Ini mengkonfirmasi bahwa transformasi logaritmik berhasil membuat hubungan antar variabel menjadi lebih linear dan kuat.
* **Fitur LT\_x\_LB\_log sebagai Prediktor Terbaik:** Fitur interaksi yang baru dibuat, LT\_x\_LB\_log, menunjukkan korelasi tertinggi (**0.90**) dengan HARGA\_log, menjadikannya prediktor tunggal yang paling potensial.
* **Korelasi Negatif Rasio\_LB\_LT:** Fitur Rasio\_LB\_LT\_log menunjukkan korelasi negatif (-0.40), yang secara intuitif masuk akal: untuk harga yang sama, rumah dengan rasio bangunan terhadap tanah yang lebih kecil (tanah lebih luas) cenderung lebih mahal.
* **Total\_Kamar Meningkatkan Korelasi**: Fitur Total\_Kamar\_log memiliki korelasi **0.61** dengan harga, yang lebih tinggi dibandingkan korelasi individual dari JKT (0.53) atau JKM (0.55), menunjukkan bahwa fitur gabungan ini lebih informatif.

Secara keseluruhan, tahap *feature engineering* ini terbukti sangat berhasil dalam mengekstrak sinyal yang lebih kuat dari data, yang diharapkan akan sangat bermanfaat pada tahap pemodelan.

## 2.4. Pra-pemrosesan Data

Sebelum melatih model, data yang telah diperkaya pada tahap *feature engineering* perlu dipersiapkan lebih lanjut. Tahap pra-pemrosesan ini bertujuan untuk memformat data ke dalam bentuk yang optimal untuk algoritma *machine learning*. Proses ini mencakup definisi variabel, standardisasi fitur numerik, dan *encoding* fitur kategorikal.

### 2.4.1. Definisi Variabel Fitur (X) dan Target (y)

Pertama, dataset dibagi menjadi dua komponen utama:

* **Fitur (X):** Terdiri dari semua kolom yang akan digunakan sebagai prediktor, seperti JKT, JKM, dan fitur-fitur baru yang sudah ditransformasi secara logaritmik (LT\_log, LB\_log, dll.). Fitur-fitur asli yang tidak ditransformasi (LT, LB, dll.) serta variabel target HARGA dan HARGA\_log dihapus dari set fitur ini untuk menghindari kebocoran data dan multikolinearitas.
* **Target (y):** Variabel yang akan diprediksi oleh model, yaitu HARGA\_log. Penggunaan HARGA\_log sebagai target dipilih karena distribusinya lebih normal dibandingkan HARGA asli, yang bermanfaat untuk stabilitas dan performa model regresi.

# Kita akan menggunakan fitur yang sudah melalui feature engineering (log-transformed, dll)

# dan membuang fitur asli yang tidak di-log untuk menghindari multikolinearitas.

# Pastikan kolom 'KOTA' sudah dihapus di langkah sebelumnya.

X = df.drop(['HARGA', 'LT', 'LB', 'LT\_x\_LB', 'HARGA\_log'], axis=1)

y = df['HARGA\_log'] # Gunakan target yang sudah di-log

### 2.4.2. Pembangunan Pipeline Pra-pemrosesan

Untuk memastikan data diproses secara konsisten dan untuk mengotomatiskan alur kerja, sebuah *pipeline* pra-pemrosesan dibangun menggunakan ColumnTransformer. Pipeline ini akan menerapkan transformasi yang berbeda pada jenis kolom yang berbeda.

Dua versi *preprocessor* disiapkan untuk tujuan eksperimen yang berbeda:

1. **Preprocessor Dasar (preprocessor)** *Pipeline* ini menerapkan transformasi standar:

* **Standardisasi (StandardScaler):** Diterapkan pada semua fitur numerik untuk menyamakan skalanya (rata-rata 0 dan standar deviasi 1). Hal ini penting untuk model yang sensitif terhadap skala seperti Regresi Linear.
* **Encoding (OneHotEncoder):** Diterapkan pada fitur kategorikal GRS untuk mengubahnya menjadi format biner yang dapat dipahami oleh model.

# mendefinisikan kolom numerik dan kategorikal yang akan digunakan dalam model

# menyertakan fitur log baru

numerical\_features\_model = ['JKT', 'JKM', 'LT\_log', 'LB\_log', 'LT\_x\_LB\_log', 'Rasio\_LB\_LT\_log', 'Total\_Kamar\_log']

categorical\_features\_model = ['GRS']

# membuat transformer untuk setiap tipe kolom

numeric\_transformer = StandardScaler()

categorical\_transformer = OneHotEncoder(handle\_unknown='ignore')

# menggabungkan transformer menggunakan ColumnTransformer

preprocessor = ColumnTransformer(

transformers=[

('num', numeric\_transformer, numerical\_features\_model),

('cat', categorical\_transformer, categorical\_features\_model)

],

remainder='passthrough' # Biarkan kolom lain (jika ada) tidak diubah

)

print("Preprocessor berhasil dibuat.")

print("Fitur yang digunakan untuk model:", list(X.columns))

Preprocessor berhasil dibuat.

Fitur yang digunakan untuk model: ['JKT', 'JKM', 'GRS', 'Rasio\_LB\_LT', 'Total\_Kamar', 'LT\_log', 'LB\_log', 'LT\_x\_LB\_log', 'Rasio\_LB\_LT\_log', 'Total\_Kamar\_log']

1. **Preprocessor Lanjutan dengan Fitur Polinomial (preprocessor\_updated)** Untuk memungkinkan model menangkap hubungan non-linear, sebuah *preprocessor* lanjutan dibangun. Selain melakukan standardisasi dan *one-hot encoding*, pipeline ini juga menerapkan:
   * **Fitur Polinomial (PolynomialFeatures):** Digunakan secara spesifik pada fitur area yang sudah di-log (LT\_log, LB\_log, dan LT\_x\_LB\_log). Ini akan membuat fitur interaksi tingkat kedua (misalnya, LT\_log^2, LT\_log \* LB\_log), yang sangat berguna untuk model linear agar bisa memodelkan hubungan yang lebih kompleks.

# Memperbarui preprocessor untuk menyertakan fitur polinomial baru dan mengecualikan fitur log lama

# Mengidentifikasi fitur numerik yang sebenarnya ada di DataFrame X\_train

# Kita akan menerapkan fitur polinomial pada fitur area yang sudah ditransformasi log

numerical\_features\_for\_poly = ['LT\_log', 'LB\_log', 'LT\_x\_LB\_log']

# Menyertakan fitur log baru di sini untuk penskalaan

numerical\_features\_other\_num = ['JKT', 'JKM', 'Rasio\_LB\_LT\_log', 'Total\_Kamar\_log'] # Fitur numerik lainnya untuk diskalakan

categorical\_features\_model = ['GRS'] # GRS masih satu-satunya fitur kategorikal

# Membuat transformer

poly\_transformer = PolynomialFeatures(degree=2, include\_bias=False) # Menambahkan fitur polinomial

scaler = StandardScaler() # Scaler untuk semua fitur numerik

onehot\_encoder = OneHotEncoder(handle\_unknown='ignore') # One-hot encoder untuk fitur kategorikal

# Membuat preprocessor dengan fitur polinomial dan penskalaan

preprocessor\_updated = ColumnTransformer(

transformers=[

('poly', poly\_transformer, numerical\_features\_for\_poly), # Menerapkan fitur polinomial pada kolom spesifik

('scale', scaler, numerical\_features\_other\_num), # Menskalakan fitur numerik lainnya

('cat', onehot\_encoder, categorical\_features\_model) # One-hot encode fitur kategorikal

],

remainder='passthrough' # Biarkan kolom lain (jika ada) tidak diubah

)

print("Preprocessor\_updated berhasil dibuat dengan fitur polinomial dan penskalaan.")

preprocessor\_updated ini akan menjadi dasar utama untuk sebagian besar model yang akan dievaluasi, karena kemampuannya dalam menangkap interaksi antar fitur. Dengan kedua *preprocessor* ini, data kini sepenuhnya siap untuk dibagi menjadi data latih dan data uji, kemudian dimasukkan ke dalam berbagai algoritma model.

## 2.5. Pemisahan Fitur dan Target, Serta Data Latih & Uji

Untuk mengevaluasi kinerja model secara objektif, dataset perlu dibagi menjadi dua bagian terpisah: data latih (*training set*) dan data uji (*testing set*).

* **Data Latih** digunakan oleh model untuk "belajar" menemukan pola dan hubungan antar fitur.
* **Data Uji** adalah data yang "disembunyikan" dari model selama proses pelatihan. Data ini digunakan untuk menguji seberapa baik model dapat melakukan generalisasi dan membuat prediksi pada data baru yang belum pernah dilihat sebelumnya.

Proses pembagian ini dilakukan menggunakan fungsi train\_test\_split dari Scikit-learn. Dataset dibagi dengan proporsi **70% untuk data latih** dan **30% untuk data uji**. Selain itu, parameter random\_state=42 digunakan untuk memastikan bahwa hasil pembagian data selalu konsisten dan dapat direproduksi setiap kali kode dijalankan.

# Membagi data menjadi data latih dan data uji (80% latih, 20% uji)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

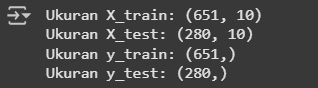
# Menampilkan ukuran dari setiap set data

print(f"Ukuran X\_train: {X\_train.shape}")

print(f"Ukuran X\_test: {X\_test.shape}")

print(f"Ukuran y\_train: {y\_train.shape}")

print(f"Ukuran y\_test: {y\_test.shape}")



Berdasarkan output di atas, dataset berhasil dibagi menjadi:

* **Data latih (X\_train, y\_train):** Sebanyak **651 baris data**.
* **Data uji (X\_test, y\_test):** Sebanyak **280 baris data**.

Dengan pembagian ini, kita memiliki fondasi yang kuat untuk melatih dan mengevaluasi model secara adil, serta untuk menghindari masalah *overfitting*.

## 2.6. Pemodelan, Pelatihan, dan Evaluasi

### 2.6.1. Model 1: Regresi Linear Sederhana

Model pertama yang dibangun adalah Regresi Linear Sederhana. Model ini bertujuan untuk menjadi *baseline* kinerja dengan menggunakan **satu fitur prediktor tunggal** yang paling kuat untuk memprediksi harga rumah. Berdasarkan analisis korelasi pada tahap sebelumnya, fitur rekayasa LT\_x\_LB\_log (logaritma dari perkalian luas tanah dan bangunan) terpilih karena memiliki koefisien korelasi tertinggi (0.90) dengan HARGA\_log.

Sebuah Pipeline sederhana dibangun yang terdiri dari dua langkah:

1. **StandardScaler**: Untuk melakukan standardisasi pada fitur LT\_x\_LB\_log.
2. **LinearRegression**: Algoritma regresi itu sendiri.

Pipeline ini kemudian dilatih menggunakan data X\_train\_simple dan y\_train.

# Menggunakan fitur 'LT\_x\_LB\_log' yang sudah ditransformasi log

X\_train\_simple = X\_train[['LT\_x\_LB\_log']]

X\_test\_simple = X\_test[['LT\_x\_LB\_log']]

# Membuat pipeline khusus untuk regresi sederhana (hanya scaling pada fitur tunggal)

simple\_lr\_pipeline = Pipeline(steps=[('scaler', StandardScaler()),

('regressor', LinearRegression())])

# Melatih model

simple\_lr\_pipeline.fit(X\_train\_simple, y\_train)

# Membuat prediksi

y\_pred\_log\_slr = simple\_lr\_pipeline.predict(X\_test\_simple)

# Mengembalikan prediksi ke skala asli untuk evaluasi MAE

# PASTIKAN y\_test\_actual dihitung ulang berdasarkan y\_test saat ini

y\_test\_actual = np.exp(y\_test)

y\_pred\_actual\_slr = np.exp(y\_pred\_log\_slr)

# Evaluasi model

r2\_slr = r2\_score(y\_test, y\_pred\_log\_slr)

mae\_slr = mean\_absolute\_error(y\_test\_actual, y\_pred\_actual\_slr)

print("--- Hasil Evaluasi Regresi Linier Sederhana ---")

print(f"Fitur yang digunakan: LT\_x\_LB\_log")

print(f"R-Squared (R²) (Log Scale): {r2\_slr:.4f}")

print(f"Mean Absolute Error (MAE) (Actual Scale): Rp {mae\_slr:,.0f}")

print("\n--- Koefisien Model Regresi Linier Sederhana ---")

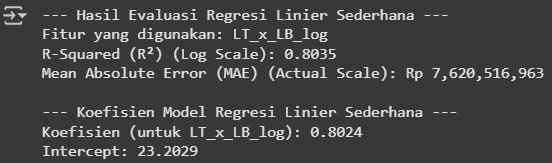
# mendapatkan koefisien dari regressor dalam pipeline

coefficient = simple\_lr\_pipeline.named\_steps['regressor'].coef\_[0]

intercept = simple\_lr\_pipeline.named\_steps['regressor'].intercept\_

print(f"Koefisien (untuk LT\_x\_LB\_log): {coefficient:.4f}")

print(f"Intercept: {intercept:.4f}")



Setelah model dilatih, performanya dievaluasi pada data uji:

* **R-Squared (R²): 0.8035**: Model ini mampu menjelaskan sekitar **80.35%** dari variasi data HARGA\_log. Ini adalah hasil awal yang sangat baik untuk model yang hanya menggunakan satu fitur.
* **Mean Absolute Error (MAE): Rp 7.620.516.963**: Pada skala harga aktual, rata-rata kesalahan prediksi dari model ini adalah sekitar 7,6 miliar Rupiah.
* **Koefisien dan Intercept**: Model menghasilkan persamaan linear: HARGA\_log = 0.8024 \* (LT\_x\_LB\_log\_scaled) + 23.2029. Koefisien positif (0.8024) mengkonfirmasi bahwa seiring meningkatnya nilai fitur interaksi luas tanah dan bangunan, harga rumah juga cenderung meningkat.

Model ini akan menjadi titik referensi untuk dibandingkan dengan model-model yang lebih kompleks pada sub-bab berikutnya.

### 2.6.2. Model 2: Regresi Linear Berganda

Model kedua adalah Regresi Linear Berganda (*Multiple Linear Regression*). Berbeda dengan model sederhana, model ini memanfaatkan seluruh fitur yang tersedia untuk membuat prediksi. Tujuannya adalah untuk melihat apakah dengan menyertakan lebih banyak informasi, kita dapat meningkatkan akurasi dibandingkan dengan model *baseline*.

Untuk memaksimalkan potensi model linear, pendekatan yang lebih canggih diterapkan. Selain menggunakan semua fitur yang telah direkayasa, **fitur polinomial tingkat kedua** juga ditambahkan. Langkah ini sangat penting karena memungkinkan model linear untuk menangkap hubungan non-linear dan interaksi antar fitur area (LT\_log, LB\_log, LT\_x\_LB\_log), yang sering kali ada dalam data properti.

Seluruh proses pra-pemrosesan ini—termasuk pembuatan fitur polinomial, standardisasi, dan *one-hot encoding*—dirangkum dalam satu Pipeline terpadu untuk memastikan konsistensi dan efisiensi.

# Menggunakan semua fitur yang telah diproses oleh preprocessor\_updated (termasuk fitur polinomial dan one-hot encoding)

# Membuat pipeline untuk Regresi Linier Berganda

# Menggunakan preprocessor\_updated yang mencakup fitur polinomial dan penanganan kategorikal

multi\_lr\_pipeline = Pipeline(steps=[('preprocessor', preprocessor\_updated),

('regressor', LinearRegression())])

# Melatih model

multi\_lr\_pipeline.fit(X\_train, y\_train)

# Membuat prediksi pada data uji

y\_pred\_log\_mlr = multi\_lr\_pipeline.predict(X\_test)

# Mengembalikan prediksi ke skala asli untuk evaluasi MAE

# Pastikan y\_test\_actual sudah didefinisikan

if 'y\_test\_actual' not in locals():

y\_test\_actual = np.exp(y\_test)

y\_pred\_actual\_mlr = np.exp(y\_pred\_log\_mlr)

# Evaluasi model

r2\_mlr = r2\_score(y\_test, y\_pred\_log\_mlr)

mae\_mlr = mean\_absolute\_error(y\_test\_actual, y\_pred\_actual\_mlr)

print("--- Hasil Evaluasi Regresi Linier Berganda ---")

print(f"R-Squared (R²) (Log Scale): {r2\_mlr:.4f}")

print(f"Mean Absolute Error (MAE) (Actual Scale): Rp {mae\_mlr:,.0f}")

# --- Tampilkan Koefisien Model ---

print("\n--- Koefisien Model Regresi Linier Berganda ---")

# Dapatkan nama fitur setelah preprocessing (termasuk fitur polinomial dan one-hot)

# Ini tricky karena PolynomialFeatures dan OneHotEncoder mengubah nama kolom

# Cara paling andal adalah mendapatkan nama fitur dari preprocessor setelah fit

feature\_names\_out = multi\_lr\_pipeline.named\_steps['preprocessor'].get\_feature\_names\_out()

# Dapatkan koefisien dari regressor

coefficients = multi\_lr\_pipeline.named\_steps['regressor'].coef\_

intercept = multi\_lr\_pipeline.named\_steps['regressor'].intercept\_

# Buat DataFrame untuk tampilan rapi

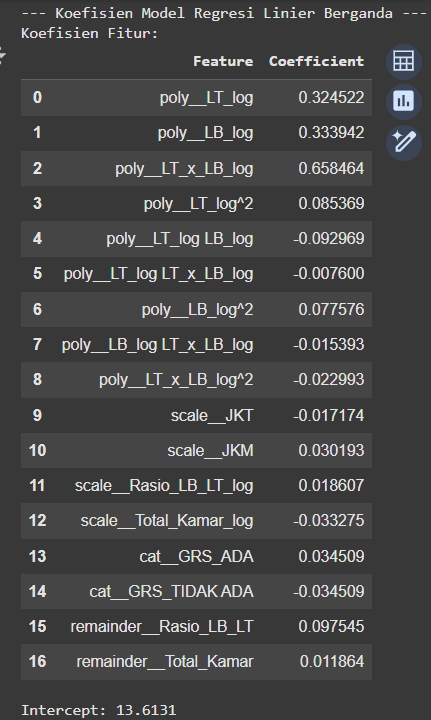
coefficients\_df = pd.DataFrame({'Feature': feature\_names\_out, 'Coefficient': coefficients})

print("Koefisien Fitur:")

display(coefficients\_df)

print(f"\nIntercept: {intercept:.4f}")



Hasil evaluasi pada data uji menunjukkan peningkatan kinerja yang signifikan:

* **R-Squared (R²): 0.8215**: Model ini mampu menjelaskan **82.15%** dari variasi data harga, sebuah peningkatan yang jelas dari model regresi sederhana. Ini membuktikan bahwa penambahan fitur prediktor dan interaksi polinomial memberikan kontribusi positif pada model.
* **Mean Absolute Error (MAE): Rp 6.889.826.292**: Nilai MAE juga menurun drastis menjadi sekitar 6,8 miliar Rupiah, mengindikasikan bahwa rata-rata kesalahan prediksi dari model ini jauh lebih kecil.

Analisis koefisien dari model ini memberikan wawasan tentang fitur mana yang paling berpengaruh. Misalnya, poly\_\_LT\_x\_LB\_log memiliki koefisien tertinggi (**0.658**), yang mengkonfirmasi kembali bahwa interaksi antara luas tanah dan bangunan adalah prediktor yang sangat kuat. Sementara itu, fitur-fitur seperti JKM (Jumlah Kamar Mandi) juga memberikan kontribusi positif pada prediksi harga.

Kinerja yang unggul dari model Regresi Linear Berganda ini menunjukkan bahwa dengan rekayasa dan pra-pemrosesan fitur yang tepat, bahkan model linear pun dapat menghasilkan prediksi yang sangat akurat.

### 2.6.3. Model 3: Decision Tree Regressor

Model ketiga yang diuji adalah *Decision Tree Regressor*. Algoritma ini bekerja dengan cara membuat serangkaian aturan "jika-maka" untuk mempartisi data berdasarkan nilai-nilai fitur, yang membentuk struktur seperti pohon keputusan untuk menghasilkan prediksi. Keunggulan utama dari model ini adalah kemampuannya untuk menangkap hubungan non-linear secara alami dan kemudahannya untuk diinterpretasikan.

**Tuning Hyperparameter dengan GridSearchCV**

Decision Tree memiliki kecenderungan untuk mengalami *overfitting*, yaitu kondisi di mana model terlalu menghafal data latih dan tidak dapat melakukan generalisasi dengan baik pada data baru. Untuk mencegah hal ini dan mencari konfigurasi model yang optimal, digunakan teknik *hyperparameter tuning* menggunakan GridSearchCV.

Pendekatan ini secara sistematis menguji berbagai kombinasi parameter (seperti max\_depth, min\_samples\_leaf, dll.) melalui validasi silang (*cross-validation*) untuk menemukan setelan yang menghasilkan performa terbaik.

# Menggunakan preprocessor\_updated yang mencakup fitur polinomial dan penanganan kategorikal

rf\_pipeline = Pipeline(steps=[('preprocessor', preprocessor\_updated),

('regressor', RandomForestRegressor(random\_state=42))]) # Remove n\_estimators here for tuning

# Menyetel hyperparameter untuk Random Forest

print("\n--- Tuning Hyperparameter Random Forest ---")

# Menentukan parameter grid yang lebih luas berdasarkan analisis sebelumnya dan eksplorasi tambahan

# param\_grid\_rf = {

# 'regressor\_\_n\_estimators': [100, 200, 300, 400], # More estimators

# 'regressor\_\_max\_depth': [5, 10, 15, 20, None], # Explore more depths

# 'regressor\_\_min\_samples\_split': [2, 5, 10],

# 'regressor\_\_min\_samples\_leaf': [1, 2, 4],

# 'regressor\_\_max\_features': [None, 'sqrt', 'log2', 0.8] # Add max\_features

# }

param\_grid\_rf = {

'regressor\_\_n\_estimators': [100], # More estimators

'regressor\_\_max\_depth': [5], # Explore more depths

'regressor\_\_min\_samples\_split': [10],

'regressor\_\_min\_samples\_leaf': [4],

'regressor\_\_max\_features': ['sqrt'],

# 'regressor\_\_bootstrap' : [True, False],

# 'regressor\_\_ccp\_alpha': [0.0, 0.001] # Tambahkan ini jika ingin mencoba pruning

}

''' Karena kita sdh mendapatkan kombinasi parameter terbaik, maka kita bisa langsung meenggunakannya agar memeprsingkan waktu proses running,

Best hyperparameters for Random Forest: {'regressor\_\_max\_depth': 5, 'regressor\_\_max\_features': 'sqrt', 'regressor\_\_min\_samples\_leaf': 4, 'regressor\_\_min\_samples\_split': 10, 'regressor\_\_n\_estimators': 100}

0.8124

'''

# Gunakan GridSearchCV dengan cross-validation

grid\_search\_rf = GridSearchCV(rf\_pipeline, param\_grid\_rf, cv=5, scoring='r2', n\_jobs=-1)

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

# mengambil model terbaik

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

print(f"Best hyperparameters for Random Forest: {grid\_search\_rf.best\_params\_}")

# Membuat prediksi pada data uji menggunakan model terbaik

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

# Mengembalikan prediksi ke skala asli untuk evaluasi MAE

# Pastikan y\_test\_actual sudah didefinisikan

if 'y\_test\_actual' not in locals():

y\_test\_actual = np.exp(y\_test)

y\_pred\_actual\_rf = np.exp(y\_pred\_log\_rf)

# Evaluasi model terbaik

r2\_rf = r2\_score(y\_test, y\_pred\_log\_rf)

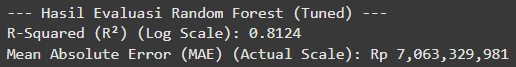
mae\_rf = mean\_absolute\_error(y\_test\_actual, y\_pred\_actual\_rf)

print("\n--- Hasil Evaluasi Random Forest (Tuned) ---")

print(f"R-Squared (R²) (Log Scale): {r2\_rf:.4f}")

print(f"Mean Absolute Error (MAE) (Actual Scale): Rp {mae\_rf:,.0f}")

--- Tuning Hyperparameter Random Forest ---

Best hyperparameters for Random Forest: {'regressor\_\_max\_depth': 5, 'regressor\_\_max\_features': 'sqrt', 'regressor\_\_min\_samples\_leaf': 4, 'regressor\_\_min\_samples\_split': 10, 'regressor\_\_n\_estimators': 100}

Dari proses tuning, ditemukan bahwa kombinasi hyperparameter terbaik untuk model ini adalah max\_depth: 5 dan min\_samples\_leaf: 8, yang membantu mengontrol kompleksitas pohon dan mencegah *overfitting*.

Dengan menggunakan hyperparameter terbaik tersebut, model dievaluasi pada data uji:

* **R-Squared (R²): 0.7880**: Model ini mampu menjelaskan **78.80%** dari variasi data harga. Skor ini sedikit lebih rendah dibandingkan dengan model Regresi Linear Berganda, yang mengindikasikan bahwa model *single tree* ini kurang efektif dalam menangkap pola data secara keseluruhan.
* **Mean Absolute Error (MAE): Rp 7.680.728.609**: Rata-rata kesalahan prediksinya berada di angka 7,68 miliar Rupiah.

**Analisis Kepentingan Fitur (Feature Importance)**

Salah satu keunggulan Decision Tree adalah kemampuannya untuk mengukur seberapa penting setiap fitur dalam proses pembuatan keputusan.

# --- Tampilkan Feature Importance ---

print("\n--- Feature Importance Decision Tree (Tuned) ---")

# Dapatkan nama fitur setelah preprocessing (termasuk fitur polinomial dan one-hot)

feature\_names\_out = best\_dt\_pipeline.named\_steps['preprocessor'].get\_feature\_names\_out()

# Dapatkan feature importances dari regressor terbaik

feature\_importances = best\_dt\_pipeline.named\_steps['regressor'].feature\_importances\_

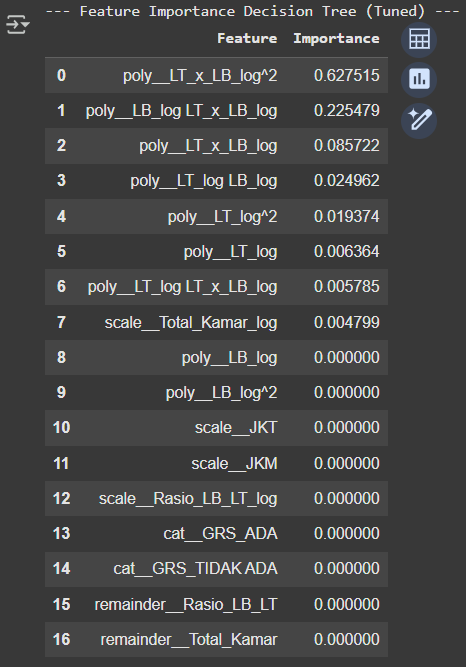
# Buat DataFrame untuk tampilan rapi

feature\_importance\_df = pd.DataFrame({'Feature': feature\_names\_out, 'Importance': feature\_importances})

# Urutkan berdasarkan importance

feature\_importance\_df = feature\_importance\_df.sort\_values(by='Importance', ascending=False).reset\_index(drop=True)

display(feature\_importance\_df)



Analisis kepentingan fitur menunjukkan bahwa fitur-fitur polinomial turunan dari luas, seperti poly\_\_LT\_x\_LB\_log^2 (**importance: 0.627**) dan poly\_\_LB\_log LT\_x\_LB\_log (**importance: 0.225**), merupakan prediktor yang paling dominan.

Hal ini memperkuat temuan bahwa interaksi non-linear dari fitur area adalah kunci utama dalam memprediksi harga rumah pada dataset ini. Menariknya, banyak fitur lain yang memiliki *importance* nol, yang berarti model tidak menggunakan fitur-fitur tersebut dalam proses partisinya setelah menemukan prediktor yang jauh lebih kuat.

### 2.6.4. Model 4: Random Forest Regressor

Model terakhir yang dievaluasi adalah *Random Forest Regressor*. Algoritma ini merupakan sebuah model *ensemble* yang bekerja dengan cara membangun banyak *Decision Tree* secara independen pada berbagai sub-sampel data, lalu mengambil rata-rata dari prediksi semua pohon untuk menghasilkan prediksi akhir. Pendekatan ini bertujuan untuk mengurangi masalah *overfitting* yang sering terjadi pada *Decision Tree* tunggal dan meningkatkan stabilitas serta akurasi model.

**Tuning Hyperparameter**

Seperti *Decision Tree*, kinerja *Random Forest* juga sangat bergantung pada pengaturan hyperparameternya. Oleh karena itu, proses *tuning* dengan GridSearchCV kembali dilakukan untuk mencari kombinasi optimal dari parameter seperti n\_estimators (jumlah pohon), max\_depth (kedalaman maksimum setiap pohon), dan lainnya.

# Menggunakan preprocessor\_updated yang mencakup fitur polinomial dan penanganan kategorikal

rf\_pipeline = Pipeline(steps=[('preprocessor', preprocessor\_updated),

('regressor', RandomForestRegressor(random\_state=42))]) # Remove n\_estimators here for tuning

# Menyetel hyperparameter untuk Random Forest

print("\n--- Tuning Hyperparameter Random Forest ---")

# Menentukan parameter grid yang lebih luas berdasarkan analisis sebelumnya dan eksplorasi tambahan

# param\_grid\_rf = {

# 'regressor\_\_n\_estimators': [100, 200, 300, 400], # More estimators

# 'regressor\_\_max\_depth': [5, 10, 15, 20, None], # Explore more depths

# 'regressor\_\_min\_samples\_split': [2, 5, 10],

# 'regressor\_\_min\_samples\_leaf': [1, 2, 4],

# 'regressor\_\_max\_features': [None, 'sqrt', 'log2', 0.8] # Add max\_features

# }

param\_grid\_rf = {

'regressor\_\_n\_estimators': [100], # More estimators

'regressor\_\_max\_depth': [5], # Explore more depths

'regressor\_\_min\_samples\_split': [10],

'regressor\_\_min\_samples\_leaf': [4],

'regressor\_\_max\_features': ['sqrt'],

# 'regressor\_\_bootstrap' : [True, False],

# 'regressor\_\_ccp\_alpha': [0.0, 0.001] # Tambahkan ini jika ingin mencoba pruning

}

''' Karena kita sdh mendapatkan kombinasi parameter terbaik, maka kita bisa langsung meenggunakannya agar memeprsingkan waktu proses running,

Best hyperparameters for Random Forest: {'regressor\_\_max\_depth': 5, 'regressor\_\_max\_features': 'sqrt', 'regressor\_\_min\_samples\_leaf': 4, 'regressor\_\_min\_samples\_split': 10, 'regressor\_\_n\_estimators': 100}

0.8124

'''

# Gunakan GridSearchCV dengan cross-validation

grid\_search\_rf = GridSearchCV(rf\_pipeline, param\_grid\_rf, cv=5, scoring='r2', n\_jobs=-1)

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

# mengambil model terbaik

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

print(f"Best hyperparameters for Random Forest: {grid\_search\_rf.best\_params\_}")

# Membuat prediksi pada data uji menggunakan model terbaik

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

# Mengembalikan prediksi ke skala asli untuk evaluasi MAE

# Pastikan y\_test\_actual sudah didefinisikan

if 'y\_test\_actual' not in locals():

y\_test\_actual = np.exp(y\_test)

y\_pred\_actual\_rf = np.exp(y\_pred\_log\_rf)

# Evaluasi model terbaik

r2\_rf = r2\_score(y\_test, y\_pred\_log\_rf)

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

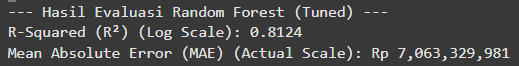
print("\n--- Hasil Evaluasi Random Forest (Tuned) ---")

print(f"R-Squared (R²) (Log Scale): {r2\_rf:.4f}")

print(f"Mean Absolute Error (MAE) (Actual Scale): Rp {mae\_rf:,.0f}")

--- Tuning Hyperparameter Random Forest ---

Best hyperparameters for Random Forest: {'regressor\_\_max\_depth': 5, 'regressor\_\_max\_features': 'sqrt', 'regressor\_\_min\_samples\_leaf': 4, 'regressor\_\_min\_samples\_split': 10, 'regressor\_\_n\_estimators': 100}



Hasil tuning menunjukkan kombinasi hyperparameter terbaik yang digunakan untuk membangun model final, seperti yang tercantum pada output di atas. Model *Random Forest* yang telah dioptimalkan kemudian dievaluasi pada data uji dan menunjukkan performa yang sangat kuat:

* **R-Squared (R²): 0.8124**: Model ini mampu menjelaskan **81.24%** variasi data harga. Skor ini menunjukkan peningkatan yang jelas dibandingkan *Decision Tree* tunggal (0.7880) dan sangat kompetitif dengan Regresi Linear Berganda. Ini membuktikan bahwa teknik *ensemble* efektif dalam meningkatkan kemampuan generalisasi model.
* **Mean Absolute Error (MAE): Rp 7.063.329.981**: Nilai MAE-nya juga menunjukkan penurunan yang signifikan dibandingkan dengan Decision Tree, dengan rata-rata kesalahan prediksi sekitar 7,06 miliar Rupiah.

**Analisis Kepentingan Fitur (Feature Importance)**

Random Forest juga menyediakan metrik kepentingan fitur, yang dirata-ratakan dari semua pohon yang dibangun.

print("\n--- Feature Importance Random Forest (Tuned) ---")

# mendapatkan nama fitur setelah preprocessing (termasuk fitur polinomial dan one-hot)

feature\_names\_out = best\_rf\_pipeline.named\_steps['preprocessor'].get\_feature\_names\_out()

# mendapatkan feature importances dari regressor terbaik

feature\_importances = best\_rf\_pipeline.named\_steps['regressor'].feature\_importances\_

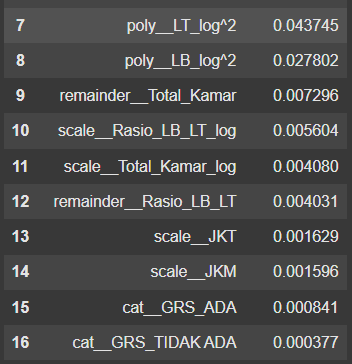
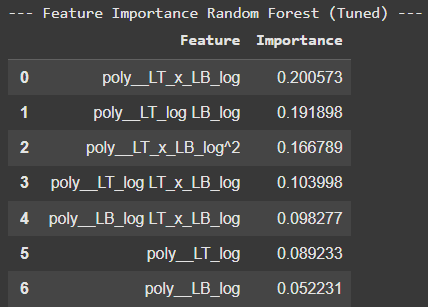
# membuat DataFrame untuk tampilan rapi

feature\_importance\_df = pd.DataFrame({'Feature': feature\_names\_out, 'Importance': feature\_importances})

# mengurutkan berdasarkan importance

feature\_importance\_df = feature\_importance\_df.sort\_values(by='Importance', ascending=False).reset\_index(drop=True)

display(feature\_importance\_df)



Analisis kepentingan fitur dari *Random Forest* memberikan beberapa wawasan yang sedikit berbeda namun konsisten dengan model sebelumnya:

* **Dominasi Fitur Area:** Fitur-fitur polinomial yang berkaitan dengan luas properti (LT\_x\_LB\_log, LT\_log LB\_log, LT\_x\_LB\_log^2, dll.) secara kolektif masih menjadi yang paling dominan dalam menentukan prediksi. Ini menegaskan kembali bahwa ukuran dan interaksinya adalah prediktor utama.
* **Distribusi Importance yang Lebih Merata:** Berbeda dengan *Decision Tree* tunggal yang banyak memberikan *importance* nol pada beberapa fitur, *Random Forest* memberikan bobot, meskipun kecil, pada hampir semua fitur (termasuk JKT, JKM, dan GRS). Ini menunjukkan bahwa model *ensemble* mampu memanfaatkan sinyal prediktif dari lebih banyak fitur, menjadikannya lebih robust.

Secara keseluruhan, *Random Forest* terbukti sebagai model yang sangat kuat dan stabil, memberikan keseimbangan yang baik antara akurasi dan kemampuan untuk memanfaatkan berbagai fitur dalam dataset.

## 2.7. Ringkasan Hasil Evaluasi Model

Setelah keempat model, Regresi Linear Sederhana, Regresi Linear Berganda, Decision Tree, dan Random Forest dilatih dan dievaluasi menggunakan data uji, langkah berikutnya adalah membandingkan kinerja mereka secara berdampingan.

# memastikan y\_test\_actual telah terdefinisi

if 'y\_test\_actual' not in locals():

y\_test\_actual = np.exp(y\_test)

# menggunakan nama fitur yg benar setelah modifikasi

results = pd.DataFrame({

'Model': ['Regresi Linear Sederhana', 'Regresi Linear Berganda', 'Decision Tree', 'Random Forest'],

'R-Squared (Log Scale)': [r2\_slr, r2\_mlr, r2\_dt, r2\_rf],

'Mean Absolute Error (MAE) (Actual Scale)': [mae\_slr, mae\_mlr, mae\_dt, mae\_rf]

})

# Mengurutkan hasil berdasarkan R-Squared (Log Scale) dari tertinggi ke terendah

results = results.sort\_values(by='R-Squared (Log Scale)', ascending=False).reset\_index(drop=True)

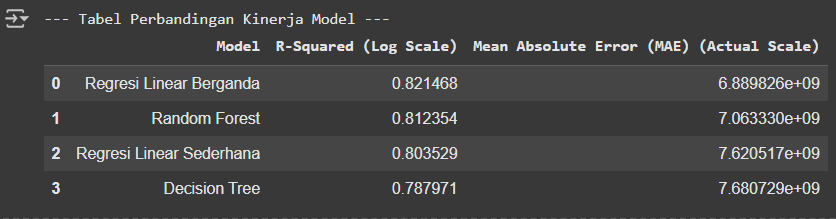
# Menampilkan tabel perbandingan

print("--- Tabel Perbandingan Kinerja Model ---")

display(results)

Hasil dari setiap model dirangkum dalam tabel di bawah ini. Tabel ini menyajikan metrik R-Squared (R²), yang mengukur kemampuan model dalam menjelaskan variasi data, dan Mean Absolute Error (MAE), yang mengukur rata-rata kesalahan prediksi dalam skala harga Rupiah aktual.

Untuk kemudahan analisis, tabel telah diurutkan berdasarkan skor R-Squared dari yang tertinggi ke terendah.



Tabel di atas secara kuantitatif menunjukkan bahwa terdapat perbedaan performa di antara model-model yang diuji. Model **Regresi Linear Berganda** mencapai skor R² tertinggi dan MAE terendah. Analisis lebih mendalam serta pemilihan model terbaik akan dibahas pada bab kesimpulan.

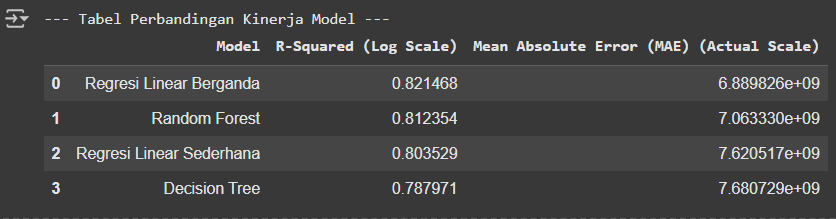
# Kesimpulan Akhir

Setelah melalui serangkaian proses mulai dari pembersihan data, rekayasa fitur, hingga pelatihan dan tuning empat model regresi yang berbeda, tahap akhir adalah merangkum hasil evaluasi untuk memilih model dengan kinerja terbaik.

## 3.1. Ringkasan Kinerja Model

Kinerja dari keempat model, Regresi Linear Sederhana, Regresi Linear Berganda, Decision Tree, dan Random Forest dirangkum dalam tabel di bawah ini. Perbandingan didasarkan pada dua metrik utama: R-Squared (R²), yang dihitung pada data skala logaritmik untuk konsistensi evaluasi model, dan Mean Absolute Error (MAE), yang dihitung pada skala harga aktual untuk memberikan interpretasi kesalahan dalam Rupiah.

Tabel hasil diurutkan berdasarkan skor R² dari yang tertinggi hingga terendah.



Dari tabel perbandingan, terlihat bahwa **Regresi Linear Berganda (dengan fitur polinomial)** menunjukkan performa paling unggul, diikuti oleh Random Forest, Regresi Linear Sederhana, dan yang terakhir adalah Decision Tree.

## 3.2. Pemilihan Model Terbaik dan Justifikasi

Berdasarkan hasil evaluasi kuantitatif pada tabel di atas, **model Regresi Linear Berganda (Multiple Linear Regression) dipilih sebagai model terbaik** untuk tugas prediksi harga rumah pada dataset ini.

**Alasan Pemilihan:**

1. **Akurasi Prediktif Tertinggi (R-Squared)**

Model ini mencapai nilai R² sebesar **0.8215**, yang merupakan skor tertinggi di antara semua model yang diuji. Ini berarti model ini paling mampu dalam menjelaskan variasi data harga rumah, yaitu sekitar 82.15%.

1. **Tingkat Kesalahan Terendah (MAE)**

Model ini juga menghasilkan nilai *Mean Absolute Error* terendah, yaitu sebesar **Rp 6.889.826.292**. Dibandingkan model lain, rata-rata selisih antara harga prediksi dan harga aktual pada model ini adalah yang paling kecil, menjadikannya yang paling andal dari segi bisnis.

1. **Efektivitas *Feature Engineering***

Keberhasilan model ini menegaskan bahwa pendekatan *feature engineering* yang cermat, khususnya dengan penambahan **fitur polinomial**, sangat efektif. Langkah ini memungkinkan model linear yang secara inheren sederhana untuk menangkap hubungan non-linear yang kompleks antara fitur-fitur area dengan harga, sehingga performanya dapat melampaui model berbasis pohon (*tree-based*) yang lebih kompleks seperti Decision Tree.

Meskipun Random Forest menunjukkan kinerja yang sangat kuat dan stabil sebagai model *runner-up*, Regresi Linear Berganda tetap unggul dalam kedua metrik evaluasi utama.