

## LAMPIRAN

### Lampiran 1. Kode Program Pengumpulan dan Penggabungan Data

```
import pandas as pd

KOLOM_FINAL = ['harga', 'kamar_tidur', 'kamar_mandi', 'garasi',
               'luas_tanah', 'luas_bangunan', 'lokasi']

# Dataset 1 - Cleaned_Final_Rumah.csv

# Load dataset 1
df1 = pd.read_csv('Dataset_Depok.csv')
print(f'Shape: {df1.shape}')

# Cek kondisi kolom dataset 1
print('Kolom df1:')
print(df1.columns.tolist())
print()
print('Tipe data:')
print(df1.dtypes)
print()
print('Sample data:')
print(df1.head(3))

# Penyesuaian kolom dataset 1
df1 = df1.rename(columns={
    'Harga' : 'harga',
    'Kamar Tidur' : 'kamar_tidur',
    'Kamar Mandi' : 'kamar_mandi',
    'Garasi' : 'garasi',
    'Luas Tanah' : 'luas_tanah',
    'Luas Bangunan' : 'luas_bangunan',
    'Lokasi' : 'lokasi'
})

# Eliminasi kolom yang tidak diperlukan
df1 = df1[KOLOM_FINAL]

print('Kolom df1 setelah penyesuaian:')
print(df1.columns.tolist())

# Dataset 2 - results_cleaned.csv

# Load dataset 2
df2 = pd.read_csv('Dataset_Bandung.csv')
print(f'Shape: {df2.shape}')

# Cek kondisi kolom dataset 2
print('Kolom df2:')
print(df2.columns.tolist())
print()
print('Tipe data:')
print(df2.dtypes)
print()
print('Sample data:')
```

```

print(df2.head(3))

# Penyesuaian kolom dataset 2
# Pisah kecamatan dari format "Kecamatan, Kota" pada kolom
location
df2['location'] = df2['location'].apply(
    lambda x: x.split(',')[0].strip() if ',' in str(x) else x
)

df2 = df2.rename(columns={
    'price'           : 'harga',
    'bedroom_count'   : 'kamar_tidur',
    'bathroom_count'  : 'kamar_mandi',
    'carport_count'    : 'garasi',
    'land_area'        : 'luas_tanah',
    'building_area (m2)': 'luas_bangunan',
    'location'         : 'lokasi'
})

# Eliminasi kolom yang tidak diperlukan
df2 = df2[KOLOM_FINAL]

print('Kolom df2 setelah penyesuaian:')
print(df2.columns.tolist())

# Dataset 3 - jaksel_cleaned.csv

# Load dataset 3 (hasil preprocessing jaksel)
df3 = pd.read_csv('Dataset Jaksel.csv')
print(f'Shape: {df3.shape}')

# Cek kondisi kolom dataset 3
print('Kolom df3:')
print(df3.columns.tolist())
print()
print('Tipe data:')
print(df3.dtypes)
print()
print('Sample data:')
print(df3.head(3))

# Penyesuaian kolom dataset 3
df3 = df3.rename(columns={
    'HARGA': 'harga',
    'KT'    : 'kamar_tidur',
    'KM'    : 'kamar_mandi',
    'GRS'   : 'garasi',
    'LT'    : 'luas_tanah',
    'LB'    : 'luas_bangunan',
})

# Eliminasi kolom yang tidak diperlukan (NO, NAMA RUMAH)
df3 = df3[KOLOM_FINAL]

print('Kolom df3 setelah penyesuaian:')
print(df3.columns.tolist())

# Penggabungan

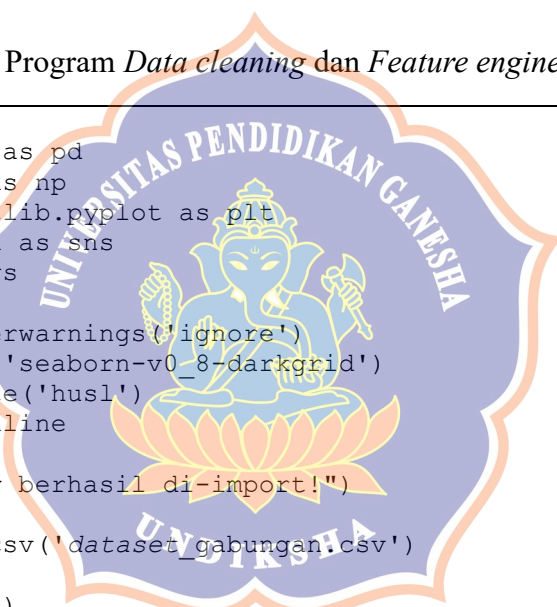
```

```
# Gabungkan ketiga dataset
df_gabungan = pd.concat([
    df1,
    df2,
    df3
], ignore_index=True)

print('=== HASIL GABUNGAN ===')
print(f'Total baris : {len(df_gabungan):,}')
print(f'Total kolom : {len(df_gabungan.columns)}')
print(f'\nNull per kolom:')
print(df_gabungan.isnull().sum())
print(f'\nDistribusi lokasi:')
print(df_gabungan['lokasi'].value_counts())

df_gabungan.to_csv('dataset_gabungan.csv', index=False)
print('File dataset_gabungan.csv berhasil disimpan!')
```

## Lampiran 2. Kode Program *Data cleaning* dan *Feature engineering*



```
#
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

warnings.filterwarnings('ignore')
plt.style.use('seaborn-v0_8-darkgrid')
sns.set_palette('husl')
%matplotlib inline

print("Library berhasil di-import!")

df = pd.read_csv('dataset_gabungan.csv')

print("=" * 60)
print("DATA OVERVIEW")
print("=" * 60)
print(f"Jumlah baris : {len(df):,}")
print(f"Jumlah kolom : {len(df.columns)}")
print(f"\nKolom      : {df.columns.tolist()}")

print("\n" + "=" * 60)
print("TIPE DATA")
print("=" * 60)
print(df.dtypes)

print("\n" + "=" * 60)
print("MISSING VALUES")
print("=" * 60)
missing = pd.DataFrame({
    'Kolom'      : df.columns,
    'Missing'    : df.isnull().sum(),
    'Missing (%)': (df.isnull().sum() / len(df) * 100).round(2)
})
```

```

print(missing)

print("\n" + "=" * 60)
print("STATISTIK DESKRIPTIF")
print("=" * 60)
print(df.describe())

print("\n" + "=" * 60)
print("SAMPLE DATA")
print("=" * 60)
print(df.head(5))

print("=" * 60)
print("TAHAP 1: PENGHAPUSAN DUPLIKAT")
print("=" * 60)
print(f"Jumlah data SEBELUM : {len(df):,}")

before = len(df)
df = df.drop_duplicates()
removed = before - len(df)

print(f"Duplikat ditemukan : {removed:,} baris")
print(f"Jumlah data SETELAH : {len(df):,}")

print("=" * 60)
print("TAHAP 2: PENANGANAN MISSING VALUES")
print("=" * 60)

print("\nMissing values sebelum handling:")
print(df.isnull().sum())
print(f"\nJumlah data SEBELUM : {len(df):,}")

kolom_penting = ['harga', 'kamar_tidur', 'kamar_mandi',
                  'garasi', 'luas_tanah', 'luas_bangunan',
                  'lokasi']

df = df.dropna(subset=kolom_penting)

print(f"\nMissing values setelah handling:")
print(df.isnull().sum())
print(f"\nJumlah data SETELAH : {len(df):,}")

print("=" * 60)
print("TAHAP 3: VALIDASI LOGIS NILAI ATRIBUT")
print("=" * 60)
print(f"Jumlah data SEBELUM : {len(df):,}")

total_removed = 0

invalid = df[df['harga'] <= 0]
print(f"\n1. Harga <= 0 : {len(invalid)} baris")
df = df[df['harga'] > 0]
total_removed += len(invalid)

invalid = df[df['luas_tanah'] <= 0]
print(f"\n2. Luas Tanah <= 0 : {len(invalid)} baris")
df = df[df['luas_tanah'] > 0]
total_removed += len(invalid)

```

```

invalid = df[df['luas_bangunan'] <= 0]
print(f"3. Luas Bangunan <= 0      : {len(invalid)} baris")
df = df[df['luas_bangunan'] > 0]
total_removed += len(invalid)

invalid = df[df['kamar_tidur'] < 1]
print(f"4. Kamar Tidur < 1        : {len(invalid)} baris")
df = df[df['kamar_tidur'] >= 1]
total_removed += len(invalid)

invalid = df[df['kamar_mandi'] < 1]
print(f"5. Kamar Mandi < 1         : {len(invalid)} baris")
df = df[df['kamar_mandi'] >= 1]
total_removed += len(invalid)

print(f"\nTotal dihapus              : {total_removed:,} baris")
print(f"Jumlah data SETELAH          : {len(df):,}")

from scipy.stats import skew

print("=" * 60)
print("TRANSFORMASI LOG PADA 'harga' (SEBELUM OUTLIER HANDLING)")
print("=" * 60)

skew_before = skew(df['harga'])
print(f"Skewness 'harga' SEBELUM log : {skew_before:.4f}")
print(f"Range SEBELUM log             : [{df['harga'].min():, .0f} - {df['harga'].max():, .0f}]")

df['harga'] = np.log(df['harga'])

skew_after = skew(df['harga'])
print(f"\nSkewness 'harga' SETELAH log : {skew_after:.4f}")
print(f"Range SETELAH log             : [{df['harga'].min():, .4f} - {df['harga'].max():, .4f}]")
print("\nCatatan: mulai baris ini dan seterusnya, kolom 'harga' PERMANEN\n")
print("dalam skala log tidak akan dikembalikan (inverse-transform) ke\n")
print("satuan Rupiah di notebook ini.")

print("=" * 60)
print("TAHAP 4: OUTLIER DETECTION - HYBRID APPROACH")
print("=" * 60)

print("\n[4a] IQR METHOD - VARIABEL KONTINU")
print("-" * 60)

def remove_outliers_iqr(df, columns):
    df_out = df.copy()
    for col in columns:
        if col in df_out.columns:
            Q1 = df_out[col].quantile(0.25)
            Q3 = df_out[col].quantile(0.75)
            IQR = Q3 - Q1
            lower = max(0, Q1 - 1.5 * IQR) # clamp ke 0, tidak
            boleh negatif
            upper = Q3 + 1.5 * IQR

```

```

        outliers = df_out[(df_out[col] < lower) | (df_out[col]
> upper)]
        print(f"    {col}: {len(outliers)} outlier | batas
[{lower:,.0f} - {upper:,.0f}]" )
        df_out = df_out[(df_out[col] >= lower) & (df_out[col]
<= upper)]
        return df_out

kolom_kontinu = ['harga', 'luas_tanah', 'luas_bangunan']

before_IQR = len(df)
df = remove_outliers_iqr(df, kolom_kontinu)
total_removed_IQR = before_IQR - len(df)
print(f"\n  Total dihapus IQR : {total_removed_iqr:,} baris")

# --- 4b. Domain Filter untuk variabel DISKRIT ---
print("\n[4b] DOMAIN FILTER - VARIABEL DISKRIT")
print("-" * 60)

domain_rules = {
    'kamar_tidur': {'min': 1, 'max': 10, 'justifikasi': 'Rumah
tinggal; Kumar & Singh (2021)'},
    'kamar_mandi': {'min': 1, 'max': 10, 'justifikasi': 'Logika
domain properti hunian'},
    'garasi'      : {'min': 0, 'max': 5, 'justifikasi': 'Empiris:
>5 cenderung properti komersial'},
}

total_removed_domain = 0
before_domain = len(df)

for col, rules in domain_rules.items():
    if col in df.columns:
        invalid = df[(df[col] < rules['min']) | (df[col] >
rules['max'])]
        print(f"    {col}: {len(invalid)} outlier | batas
[{rules['min']} - {rules['max']}]" )
        print(f"    Justifikasi: {rules['justifikasi']}")
        df = df[(df[col] >= rules['min']) & (df[col] <=
rules['max'])]
        total_removed_domain += len(invalid)

print(f"\n      Total      dihapus      Domain      Filter      :
{total_removed_domain:,} baris")

# Ringkasan
print("\n" + "=" * 60)
print("RINGKASAN OUTLIER REMOVAL")
print("=" * 60)
print(f"  IQR (Kontinu)      : {total_removed_iqr:,} baris")
print(f"  Domain (Diskrit) : {total_removed_domain:,} baris")
print(f"  TOTAL      dihapus      : {total_removed_IQR +
total_removed_domain:,} baris")
print(f"  Data tersisa      : {len(df):,} baris")

print("=" * 60)
print("TAHAP 5: VISUALISASI DISTRIBUSI SETELAH CLEANING")
print("=" * 60)

```

```

kolom_numerik = ['harga', 'luas_tanah', 'luas_bangunan',
                  'kamar_tidur', 'kamar_mandi', 'garasi']

fig, axes = plt.subplots(2, 3, figsize=(18, 10))
axes = axes.flatten()

for idx, col in enumerate(kolom_numerik):
    axes[idx].boxplot(df[col].dropna())
    label = f'{col} (skala log)' if col == 'harga' else col
    axes[idx].set_title(label, fontweight='bold')
    axes[idx].set_ylabel('Value')
    print(f"{col}:      Min={df[col].min():.0f}      |
Max={df[col].max():.0f} | Median={df[col].median():.0f}")

plt.suptitle('Distribusi Setelah Cleaning', fontsize=14,
fontweight='bold')
plt.tight_layout()
plt.show()

df.head()

print("=" * 60)
print("FINAL DATA SUMMARY")
print("=" * 60)
print(f"Rows      : {len(df):,}")
print(f"Columns   : {len(df.columns):,}")

print("\n" + "=" * 60)
print("KOLOM FINAL")
print("=" * 60)
for i, col in enumerate(df.columns, 1):
    print(f" {i}. {col}")

print("\n" + "=" * 60)
print("MISSING VALUES CHECK")
print("=" * 60)
missing = df.isnull().sum()
if missing.sum() == 0:
    print("Tidak ada missing values.")
else:
    print(missing[missing > 0])

print("\n" + "=" * 60)
print("DATA QUALITY METRICS")
print("=" * 60)
print(f"Duplikat           : {df.duplicated().sum():,}")
print(f"Total missing      : {df.isnull().sum().sum():,}")
print(f"Harga (log) min    : {df['harga'].min():.4f}")
print(f"Harga (log) max    : {df['harga'].max():.4f}")
print(f"Luas Tanah <= 0    : {len(df[df['luas_tanah'] <= 0])}")
print(f"Luas Bangunan <= 0: {len(df[df['luas_bangunan'] <= 0])}")

print("\n" + "=" * 60)
print("PERINGATAN: KOLOM 'harga' SUDAH DALAM SKALA LOG")
print("=" * 60)

output = 'dataset_cleaned_log.csv'
df.to_csv(output, index=False)
print(f"\nData berhasil disimpan: {output}")

```

```
print(f"Shape final: {df.shape}")
```

### Lampiran 3. Kode Program *Feature engineering*

```
import pandas as pd
import numpy as np

print("Library berhasil di-import!")

df = pd.read_csv('dataset_cleaned_log.csv')

print("=" * 60)
print("DATA OVERVIEW (Hasil Cleaning)")
print("=" * 60)
print(f"Jumlah baris : {len(df):,}")
print(f"Jumlah kolom : {len(df.columns)}")
print(f"\nKolom      : {df.columns.tolist()}")
print("\nCatatan: kolom 'harga' pada file ini sudah dalam skala log\n"
      "(lihat data_cleaning_log_experiment.ipynb), belum di-inverse-transform.")

depok_kec = [
    'Beji', 'Bojong Sari', 'Cibubur', 'Cimanggis', 'Cilodong',
    'Cinangka',
    'Cinere', 'Cipayung', 'Citayam', 'Depok I', 'Depok II', 'Grand
    Depok City',
    'Harjamukti', 'Kelapa Dua', 'Kukusan', 'Limo', 'Margonda',
    'Pancoran Mas',
    'Pangkalan Jati', 'Rangkapanjaya', 'Sawangan', 'Sukmajaya',
    'Tanah Baru',
    'Tapos', 'Tirtajaya', 'Babadan', 'Ciangsana', 'Cisalak',
    'Gandul',
    'Kalimanggis', 'Krukut', 'Mekarsari', 'Pasir Putih', 'Studio
    Alam',
    'Sukatani', 'Tugu'
]

bandung_kec = [
    'Andir', 'Antapani', 'Arcamanik', 'Astanaanyar',
    'Babakanciparay',
    'Bandung Kidul', 'Bandung Kulon', 'Bandung Wetan',
    'Batununggal',
    'Bojongloa Kidul', 'Buah Batu', 'Cibeunying Kidul', 'Cibiru',
    'Cicendo',
    'Cidadap', 'Coblong', 'Gede Bage', 'Kiaracondong',
    'Lengkong',
    'Mandalajati', 'Panyileukan', 'Rancasari', 'Regol',
    'Sukajadi',
    'Sukasari', 'Sumurbandung', 'Ujungberung', 'Cilangkap'
]

jaksel_kec = ['Jakarta Selatan', 'Tebet']

def assign_kota(lokalasi):
    if lokalasi in depok_kec:
```

```

        return 'Depok'
    elif lokasi in bandung_kec:
        return 'Bandung'
    elif lokasi in jaksel_kec:
        return 'Jakarta Selatan'
    else:
        return 'Tidak Diketahui'

df['sumber_kota'] = df['lokasi'].apply(assign_kota)

print("=" * 60)
print("FITUR BARU: sumber_kota")
print("=" * 60)
print("\nDistribusi sumber_kota:")
dist = df['sumber_kota'].value_counts()
for kota, jumlah in dist.items():
    print(f"    {kota:<20} : {jumlah:,} data")

if (df['sumber_kota'] == 'Tidak Diketahui').any():
    n_unknown = (df['sumber_kota'] == 'Tidak Diketahui').sum()
    print(f"\nPeringatan: {n_unknown} baris memiliki lokasi yang belum terpetakan ke kota manapun.")
    print("Nilai lokasi tersebut:")
    print(df.loc[df['sumber_kota'] == 'Tidak Diketahui', 'lokasi'].unique())

print("=" * 60)
print("FITUR TURUNAN")
print("=" * 60)

harga_rupiah_rekonstruksi = np.exp(df['harga'])

df['harga_per_m2_tanah'] = harga_rupiah_rekonstruksi / df['luas_tanah']
print("Dibuat: harga_per_m2_tanah (dari harga hasil rekonstruksi exp)")

df['harga_per_m2_bangunan'] = harga_rupiah_rekonstruksi / df['luas_bangunan']
print("Dibuat: harga_per_m2_bangunan (dari harga hasil rekonstruksi exp)")

df['building_land_ratio'] = df['luas_bangunan'] / df['luas_tanah']
print("Dibuat: building_land_ratio")

print(f"\nTotal kolom sekarang : {len(df.columns)}")
print(f"Kolom : {df.columns.tolist()}")

df.head()

print("=" * 60)
print("RINGKASAN FEATURE ENGINEERING")
print("=" * 60)
print(f"Rows : {len(df):,}")
print(f"Columns : {len(df.columns)}")

print("\n" + "=" * 60)
print("KOLOM FINAL")

```

```

print("=" * 60)
for i, col in enumerate(df.columns, 1):
    print(f"  {i}. {col}")

print("\n" + "=" * 60)
print("MISSING VALUES CHECK (fitur baru)")
print("=" * 60)
fitur_baru = ['sumber_kota', 'harga_per_m2_tanah',
              'harga_per_m2_bangunan',
              'building_land_ratio']
missing_baru = df[fitur_baru].isnull().sum()
if missing_baru.sum() == 0:
    print("Tidak ada missing values pada fitur baru.")
else:
    print(missing_baru[missing_baru > 0])

print("\n" + "=" * 60)
print("PERINGATAN: KOLOM 'harga' MASIH DALAM SKALA LOG")
print("=" * 60)

output = 'dataset_clean_log.csv'
df.to_csv(output, index=False)
print(f"\nData berhasil disimpan: {output}")
print(f"Shape final: {df.shape}")

```

#### Lampiran 4. Kode Program *Exploratory Data Analysis* dan Feature Selection

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

warnings.filterwarnings('ignore')
plt.style.use('seaborn-v0_8-darkgrid')
sns.set_palette('husl')
%matplotlib inline

print("Library berhasil di-import!")

df = pd.read_csv('dataset_clean_log.csv')

print("Dataset hasil feature engineering (log-experiment)
berhasil dimuat!")
print(f"  Jumlah baris : {len(df):,}")
print(f"  Jumlah kolom : {len(df.columns)}")
print(f"  Kolom       : {df.columns.tolist()}")

df['harga_rupiah_rekonstruksi'] = np.exp(df['harga'])

print("=" * 60)
print("STATISTIK DESKRIPTIF (skala asli sesuai kolom masing-
masing)")
print("=" * 60)

```

```

df.describe().round(4)

dist = df['sumber_kota'].value_counts()
print("=" * 40)
print("JUMLAH DATA PER KOTA")
print("=" * 40)
for kota, jumlah in dist.items():
    print(f"{kota:<20} : {jumlah:,} data")
print("-" * 40)
print(f"{'Total':<20} : {len(df):,} data")

fig, axes = plt.subplots(1, 2, figsize=(16, 6))

df.boxplot(column='harga_rupiah_rekonstruksi', by='sumber_kota',
ax=axes[0])
axes[0].set_title('Distribusi Harga per Kota (skala Rupiah)')
axes[0].set_xlabel('Kota')
axes[0].set_ylabel('Harga (Rp)')
plt.sca(axes[0])
plt.xticks(rotation=30)

rata_harga = df.groupby('sumber_kota')['harga_rupiah_rekonstruksi'].mean().sort_values(ascending=False)
rata_harga.plot(kind='bar', ax=axes[1], color='steelblue', edgecolor='black')
axes[1].set_title('Rata-rata Harga per Kota (skala Rupiah)')
axes[1].set_xlabel('Kota')
axes[1].set_ylabel('Rata-rata Harga (Rp)')
axes[1].tick_params(axis='x', rotation=30)

plt.suptitle('')
plt.tight_layout()
plt.show()

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

axes[0].hist(df['harga_rupiah_rekonstruksi'], bins=40, color='indianred', edgecolor='black')
axes[0].set_title('Distribusi Harga – Skala Rupiah (rekonstruksi)', fontweight='bold')
axes[0].set_xlabel('Harga (Rp)')
axes[0].set_ylabel('Frekuensi')

axes[1].hist(df['harga'], bins=40, color='steelblue', edgecolor='black')
axes[1].set_title('Distribusi Harga – Skala log (dipakai model)', fontweight='bold')
axes[1].set_xlabel('log(Harga)')
axes[1].set_ylabel('Frekuensi')

plt.suptitle('Efek Transformasi Log terhadap Distribusi Harga', fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()

print(f"Skewness harga (Rupiah) : {df['harga_rupiah_rekonstruksi'].skew():.4f}")
print(f"Skewness harga (log) : {df['harga'].skew():.4f}")

```

```

kolom_numerik = ['luas_tanah', 'luas_bangunan', 'kamar_tidur',
'kamar_mandi', 'garasi']

fig, axes = plt.subplots(2, 3, figsize=(16, 10))
axes = axes.flatten()

for i, col in enumerate(kolom_numerik):
    axes[i].hist(df[col], bins=40, color='steelblue',
edgecolor='black')
    axes[i].set_title(f'Distribusi {col}', fontweight='bold')
    axes[i].set_xlabel(col)
    axes[i].set_ylabel('Frekuensi')

axes[-1].axis('off')
plt.suptitle('Distribusi Fitur Numerik Dasar (selain harga)',
fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()

for col in ['kamar_tidur', 'kamar_mandi', 'garasi']:
    print("=" * 50)
    print(f"VALUE COUNTS: {col}")
    print("=" * 50)
    vc = df[col].value_counts().sort_index()
    for nilai, jumlah in vc.items():
        print(f"    {col} = {nilai:<5}: {jumlah:,} data")
    print()

kolom_turunan = ['harga_per_m2_tanah', 'harga_per_m2_bangunan',
'building_land_ratio']

fig, axes = plt.subplots(1, 3, figsize=(16, 5))

for i, col in enumerate(kolom_turunan):
    axes[i].hist(df[col], bins=40, color='darkorange',
edgecolor='black')
    axes[i].set_title(f'Distribusi {col}', fontweight='bold')
    axes[i].set_xlabel(col)
    axes[i].set_ylabel('Frekuensi')

plt.suptitle('Distribusi Fitur Hasil Feature engineering',
fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()

kolom_corr = ['harga', 'kamar_tidur', 'kamar_mandi', 'garasi',
'luas_tanah', 'luas_bangunan', 'harga_per_m2_tanah',
'harga_per_m2_bangunan', 'building_land_ratio']

corr = df[kolom_corr].corr().round(2)

plt.figure(figsize=(12, 8))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f',
linewidths=0.5, square=True)
plt.title('Heatmap Korelasi - harga dalam skala log',
fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()

```

```

print("=" * 70)
print("Matriks Korelasi Pearson Lengkap (semua pasangan fitur)")
print("=" * 70)
print(corr.to_string())

print("=" * 60)
print("Rata-rata Fitur per Kota")
print("=" * 60)

ringkasan = df.groupby('sumber_kota')[['harga_rupiah_rekonstruksi',
'luas_tanah',
'luas_bangunan',
'kamar_tidur',
'kamar_mandi',
'garasi']].mean().round(0)
ringkasan = ringkasan.rename(columns={'harga_rupiah_rekonstruksi': 'harga (Rp)'})
print(ringkasan.to_string())

kolom_kandidat = ['kamar_tidur', 'kamar_mandi', 'garasi',
'luas_tanah', 'luas_bangunan',
'harga_per_m2_tanah', 'harga_per_m2_bangunan',
'building_land_ratio']

korelasi_harga = df[['harga'] +
kolom_kandidat].corr()[['harga']].drop('harga')
korelasi_harga = korelasi_harga.sort_values(ascending=False)

tabel_korelasi = pd.DataFrame({
'Fitur': korelasi_harga.index,
'Korelasi Pearson': korelasi_harga.values.round(4)
}).reset_index(drop=True)

print("=" * 45)
print("Tabel Korelasi Pearson terhadap Harga (log)")
print("=" * 45)
print(tabel_korelasi.to_string(index=False))
print("=" * 45)
print(f"Total fitur dievaluasi : {len(tabel_korelasi)}")
print("=" * 45)

THRESHOLD_KORELASI = 0.50

korelasi_abs_sorted = korelasi_harga.reindex(korelasi_harga.abs().sort_values(ascending=False).index)

tabel_threshold = pd.DataFrame({
'Fitur': korelasi_abs_sorted.index,
'Korelasi Pearson': korelasi_abs_sorted.values.round(4),
'Abs(Korelasi)': korelasi_abs_sorted.abs().values.round(4),
'Lolos Threshold (>= 0.50)': (korelasi_abs_sorted.abs() >=
THRESHOLD_KORELASI).values
}).reset_index(drop=True)

print("=" * 70)

```

```

print(f"TABEL KORELASI PEARSON TERHADAP HARGA (threshold =
{THRESHOLD_KORELASI})")
print("=" * 70)
print(tabel_threshold.to_string(index=False))

lolos_korelasi = tabel_threshold.loc[tabel_threshold['Lolos
Threshold (>= 0.50)'], 'Fitur'].tolist()
gagal_korelasi = tabel_threshold.loc[~tabel_threshold['Lolos
Threshold (>= 0.50)'], 'Fitur'].tolist()

print(f"\nFitur LOLOS threshold korelasi : {lolos_korelasi}")
print(f"Fitur TIDAK lolos : {gagal_korelasi}")

FITUR_LEAKAGE = ['harga_per_m2_tanah', 'harga_per_m2_bangunan']

print("=" * 60)
print("FITUR DIBUANG KARENA TARGET LEAKAGE")
print("=" * 60)
for f in FITUR_LEAKAGE:
    status = "lolos threshold korelasi, TAPI tetap dibuang
(leakage)" \
        if f in lolos_korelasi else "tidak lolos threshold
korelasi, DAN mengandung leakage"
    print(f" - {f}: {status}")

lolos_setelah_leakage_check = [f for f in lolos_korelasi if f not
in FITUR_LEAKAGE]

print(f"\nFitur tersisa setelah eksklusi leakage:
{lolos_setelah_leakage_check}")

fitur_final = lolos_setelah_leakage_check

print("=" * 60)
print("FITUR FINAL SETELAH ELIMINASI LEAKAGE")
print("=" * 60)
print(f"\nFitur final: {fitur_final}")

ringkasan_fs = []
semua_kolom = kolom_kandidat

for f in semua_kolom:
    if f in FITUR_LEAKAGE:
        alasan = "Dibuang - target leakage (mengandung informasi
harga dalam rumus)"
    elif f in fitur_final:
        alasan = f"Dipertahankan - korelasi
{korelasi_harga[f]:.2f} (lolos threshold 0.50)"
    else:
        alasan = f"Dibuang - korelasi {korelasi_harga[f]:.2f} (di
bawah threshold 0.50)"
    ringkasan_fs.append({'Fitur': f, 'Keputusan': alasan})

df_ringkasan = pd.DataFrame(ringkasan_fs)

print("=" * 90)
print("RINGKASAN KEPUTUSAN FEATURE SELECTION")
print("=" * 90)
print(df_ringkasan.to_string(index=False))

```

```

print("\n" + "=" * 60)
print(f"FITUR NUMERIK FINAL ({len(fitur_final)} fitur):")
print("=" * 60)
for f in fitur_final:
    print(f" - {f}")

```

## Lampiran 5. Kode Program Data Preprocessing, Pengembangan dan Validasi Model

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import copy
import warnings
warnings.filterwarnings('ignore')

from sklearn.linear_model import LinearRegression, Ridge
from sklearn.ensemble import RandomForestRegressor
from XGBoost import XGBRegressor

from sklearn.model_selection import train_test_split, KFold,
RepeatedKFold, GridSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import r2_score, mean_squared_error,
mean_absolute_error
from scipy.stats import wilcoxon

print('Library berhasil diimport.')
print('Versi eksperimen : K1 v14 (Log-Transform Konsisten: Single
Split + Repeated K-Fold)')

df = pd.read_csv('dataset_clean_log.csv')

print('=' * 60)
print('INFORMASI DATASET')
print('=' * 60)
print(f'Jumlah baris : {df.shape[0],}')
print(f'Jumlah kolom : {df.shape[1],}')
print(f'\nKolom yang tersedia:')
print(df.columns.tolist())
print(f'\nContoh 3 baris pertama:')
print(df.head(3).to_string())

depok_kec = [
    'Beji', 'Bojong Sari', 'Cibubur', 'Cimanggis', 'Cilodong',
    'Cinangka',
    'Cinere', 'Cipayung', 'Citayam', 'Depok I', 'Depok II', 'Grand
Depok City',
    'Harjamukti', 'Kelapa Dua', 'Kukusan', 'Limo', 'Margonda',
    'Pancoran Mas',
    'Pangkalan Jati', 'Rangkapanjaya', 'Sawangan', 'Sukmajaya',
    'Tanah Baru',
    'Tapos', 'Tirtajaya', 'Babadan', 'Ciangsana', 'Cisalak',
    'Gandul',
    'Kalimanggis', 'Krukut', 'Mekarsari', 'Pasir Putih', 'Studio
Alam',

```

```

    'Sukatani', 'Tugu'
]

bandung_kec = [
    'Andir', 'Antapani', 'Arcamanik', 'Astanaanyar',
    'Babakanciparay',
    'Bandung Kidul', 'Bandung Kulon', 'Bandung Wetan',
    'Batununggal',
    'Bojongloa Kidul', 'Buah Batu', 'Cibeunying Kidul', 'Cibiru',
    'Cicendo',
    'Cidadap', 'Coblong', 'Gede Bage', 'Kiaracandong',
    'Lengkong',
    'Mandalajati', 'Panyileukan', 'Rancasari', 'Regol',
    'Sukajadi',
    'Sukasari', 'Sumurbandung', 'Ujungberung', 'Cilangkap'
]

jaksel_kec = ['Jakarta Selatan', 'Tebet']

def assign_kota(lokalasi):
    if lokasi in depok_kec:
        return 'Depok'
    elif lokasi in bandung_kec:
        return 'Bandung'
    elif lokasi in jaksel_kec:
        return 'Jakarta Selatan'
    else:
        return 'Tidak Diketahui'

df['sumber_kota'] = df['lokasi'].apply(assign_kota)

print('\n' + '=' * 60)
print('DISTRIBUSI SUMBER KOTA')
print('=' * 60)
print(df['sumber_kota'].value_counts())

print('\n' + '=' * 60)
print('PREPROCESSING v14 (log-transform konsisten end-to-end)')
print('=' * 60)

# -- 3a. Cek missing values --
print('\nMissing values per kolom:')
print(df.isnull().sum())

# -- 3b. Label Encoding untuk sumber_kota --
le_kota = LabelEncoder()
df['sumber_kota_enc'] = le_kota.fit_transform(df['sumber_kota'])
print(f'\nLabel Encoding selesai:')
print(f'    sumber_kota : {df["sumber_kota"].unique()}    nilai unik')
print(f'    Mapping : {dict(zip(le_kota.classes_, le_kota.transform(le_kota.classes_)))}')

# -- 3c. Definisi fitur (X) dan target (y) --
FITUR_BASE = [
    'kamar_tidur', 'kamar_mandi',
    'luas_tanah', 'luas_bangunan',
    'sumber_kota_enc'
]

```

```

FITUR = FITUR_BASE + ['lokasi_enc']

X_raw = df[FITUR_BASE + ['lokasi']]
y = df['harga'] # SUDAH log(harga_rupiah) sejak
data_cleaning.ipynb

print(f'\nJumlah fitur yang akan digunakan : {len(FITUR)}
(termasuk lokasi_enc)')
print(f'Daftar fitur: {FITUR}')

print(f'\nShape X_raw : {X_raw.shape}')
print(f'Shape y : {y.shape}')

stratify_kota = df['sumber_kota']

print('\n' + '=' * 60)
print('TAHAP 0: STRATIFIED HOLD-OUT SPLIT (70:30)')
print('=' * 60)

X_raw_train, X_raw_test, y_train, y_test = train_test_split(
    X_raw, y, test_size=0.30, random_state=42,
    stratify=stratify_kota
)

print(f'\nData training (70%) : {X_raw_train.shape[0],} baris')
print(f'Data test (30%) : {X_raw_test.shape[0],} baris')

print('\n' + '-' * 60)
print('TARGET SUDAH DALAM SKALA log SEJAK data_cleaning.ipynb')
print('-' * 60)
print('Tidak ada transformasi log tambahan di sini (menghindari
double log-transform).')

y_train_log = y_train.copy()
y_test_log = y_test.copy()

print(f'Skewness harga log (train) : {y_train_log.skew():.3f}')

print('\n' + '-' * 60)
print('[1x outlier] OUTLIER REMOVAL DI-SKIP -- sudah ditangani di
data_cleaning.ipynb')
print('-' * 60)
print(f'Data 70% (train) : {X_raw_train.shape[0],} baris (tidak
ada filtering tambahan)')
print(f'Data 30% (test) : {X_raw_test.shape[0],} baris (tidak
ada filtering tambahan)')

y_train = np.exp(y_train_log)
y_test = np.exp(y_test_log)

print('\n' + '-' * 60)
print('TAHAP 1: SMOOTHED TARGET ENCODING (dari 70% training, skala
LOG)')
print('-' * 60)

SMOOTHING_K = 10

target_enc_df = X_raw_train.copy()
target_enc_df['harga_log'] = y_train_log.values

```

```

global_mean = y_train_log.mean()
lokasi_stats = target_enc_df.groupby('lokasi')['harga_log'].agg(['mean', 'count'])
lokasi_stats.columns = ['mean_lokasi', 'count']

lokasi_smoothed_enc = (
    (lokasi_stats['count'] * lokasi_stats['mean_lokasi'] +
     SMOOTHING_K * global_mean)
    / (lokasi_stats['count'] + SMOOTHING_K)
)

print(f'Smoothing factor k = {SMOOTHING_K}')
print(f'Jumlah kecamatan unik: {len(lokasi_stats)}')

X_raw_train = X_raw_train.copy()
X_raw_test = X_raw_test.copy()

X_raw_train['lokasi_enc'] = lokasi_smoothed_enc
X_raw_train['lokasi'].map(lokasi_smoothed_enc).fillna(global_mean)
X_raw_test['lokasi_enc'] = lokasi_smoothed_enc
X_raw_test['lokasi'].map(lokasi_smoothed_enc).fillna(global_mean)

X_train = X_raw_train[FITUR].reset_index(drop=True)
X_test = X_raw_test[FITUR].reset_index(drop=True)
y_train_log = y_train_log.reset_index(drop=True)
y_test_log = y_test_log.reset_index(drop=True)
y_train = y_train.reset_index(drop=True)
y_test = y_test.reset_index(drop=True)

print(f'\nShape X_train (70%) : {X_train.shape}')
print(f'Shape X_test (30%) : {X_test.shape}')

print('\n' + '-' * 60)
print('TAHAP 2: SCALING (StandardScaler di-fit dari 70% training)')
print('-' * 60)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print('Scaling selesai.')
print('\n' + '=' * 60)
print('RINGKASAN VARIABEL TARGET SETELAH PREPROCESSING')
print('=' * 60)
print('y_train_log, y_test_log -> dipakai untuk melatih & memantau model (Bagian 4-6)')
print('y_train, y_test -> skala asli (Rp), dipakai untuk evaluasi akhir (Bagian 7)')

print('\n' + '=' * 60)
print('BAGIAN 4: TAHAP 2 -- HYPERPARAMETER TUNING RF DAN XGBOOST')
print('=' * 60)
print('Tuning RF dan XGBoost via GridSearchCV 5-Fold pada X_train (70%),')

```

```

print('terhadap y_train_log (skala log, konsisten dengan v14).')
print('Ridge (meta learner) tetap dituning di Bagian 5, setelah
OOE selesai.\n')

N_FOLDS = 5

# -- 4a. Tuning Random Forest --
print('[1/2] Grid Search CV untuk Random Forest...')
param_grid_rf = {
    'n_estimators'      : [100, 200, 300, 400],
    'max_depth'         : [10, 20, 30, None],
    'min_samples_split' : [2, 5, 10],
    'min_samples_leaf'  : [1, 3, 5],
    'max_features'      : ['sqrt', 0.5],
}

gs_rf = GridSearchCV(
    estimator = RandomForestRegressor(random_state=42, n_jobs=-
1),
    param_grid = param_grid_rf,
    cv          = N_FOLDS,
    scoring     = 'r2',
    n_jobs      = -1,
    verbose     = 1
)
gs_rf.fit(X_train, y_train_log)

print(f'\nParameter terbaik Random Forest      :
{gs_rf.best_params_}')
print(f'R2 CV terbaik (5-Fold, skala log) :
{gs_rf.best_score_:.4f}')

print('\nTop 5 kombinasi parameter RF:')
cv_res_rf = pd.DataFrame(gs_rf.cv_results_)
top5_rf   = cv_res_rf.nlargest(5, 'mean_test_score')[['params',
'mean_test_score', 'std_test_score']]
for _, row in top5_rf.iterrows():
    print(f'R2={row["mean_test_score"]:.4f} (+/-
{row["std_test_score"]:.4f}) | {row["params"]}')

# -- 4b. Tuning XGBoost --
print('\n[2/2] Grid Search CV untuk XGBoost...')
param_grid_xgb = {
    'n_estimators'      : [100, 200, 300],
    'max_depth'         : [3, 6, 9],
    'learning_rate'     : [0.01, 0.05, 0.1],
    'subsample'         : [0.8, 1.0],
    'colsample_bytree'  : [0.7, 0.9],
    'min_child_weight'  : [1, 5],
    'reg_lambda'        : [1, 10],
    'reg_alpha'         : [0, 0.1, 1.0],
}

gs_xgb = GridSearchCV(
    estimator = XGBRegressor(random_state=42, verbosity=0,
n_jobs=-1),
    param_grid = param_grid_xgb,
    cv          = N_FOLDS,
    scoring     = 'r2',

```

```

        n_jobs      = -1,
        verbose     = 1
    )
    gs_xgb.fit(X_train, y_train_log)

    print(f'\nParameter terbaik XGBoost : {gs_xgb.best_params_}')
    print(f'R2      CV      terbaik      (5-Fold,      skala      log) :
    {gs_xgb.best_score_:.4f}')
```

print('\nTop 5 kombinasi parameter XGBoost:')  
cv\_res\_xgb = pd.DataFrame(gs\_xgb.cv\_results\_)  
top5\_xgb = cv\_res\_xgb.nlargest(5, 'mean\_test\_score')[['params',  
'mean\_test\_score', 'std\_test\_score']]  
for \_, row in top5\_xgb.iterrows():  
 print(f' R2={row["mean\_test\_score"]:.4f} (+/-  
{row["std\_test\_score"]:.4f}) | {row["params"]}')  
  
print('\n' + '=' \* 60)  
print('BAGIAN 5: TAHAP 3 -- OOF + TUNING RIDGE PADA MATRIKS OOF')  
print('=' \* 60)

```

base_learners = {
    'Linear Regression': LinearRegression(),
    'Random Forest' : RandomForestRegressor(**gs_rf.best_params_, random_state=42,
n_jobs=-1),
    'XGBoost' : XGBRegressor(**gs_xgb.best_params_,
random_state=42, verbosity=0, n_jobs=-1)
}

kf_oof = KFold(n_splits=N_FOLDS, shuffle=True, random_state=42)
oof_predictions = np.zeros((X_train.shape[0],
len(base_learners)))

for idx, (nama, model) in enumerate(base_learners.items()):
    for tr_idx, val_idx in kf_oof.split(X_train):
        if nama == 'Linear Regression':
            Xtr, Xval = X_train_scaled[tr_idx],
X_train_scaled[val_idx]
        else:
            Xtr, Xval = X_train.iloc[tr_idx],
X_train.iloc[val_idx]
            ytr = y_train_log.iloc[tr_idx]
            m_tmp = copy.deepcopy(model)
            m_tmp.fit(Xtr, ytr)
            oof_predictions[val_idx, idx] = m_tmp.predict(Xval)

    r2_oof = r2_score(y_train_log, oof_predictions[:, idx])
    print(f'{nama:<20} OOF R2 (skala log) : {r2_oof:.4f}')
```

```

gs_ridge = GridSearchCV(
    Ridge(), {'alpha': [0.01, 0.1, 1.0, 10.0, 50.0, 100.0, 500.0,
1000.0]},
    cv=N_FOLDS, scoring='r2', n_jobs=-1
)
gs_ridge.fit(oof_predictions, y_train_log)
ridge_alpha = gs_ridge.best_params_['alpha']
print(f'\nRidge best alpha (dituning pada matriks OOF) :
{ridge_alpha}')
```

```

df_grid_ridge =
pd.DataFrame(gs_ridge.cv_results_)[['param_alpha',
'mean_test_score', 'std_test_score']]
df_grid_ridge.columns = ['Alpha', 'Mean R2 (CV)', 'Std R2 (CV)']
df_grid_ridge =
df_grid_ridge.sort_values('Alpha').reset_index(drop=True)
print('\n' + '-' * 60)
print('DETAIL GRIDSEARCHCV RIDGE (per kandidat alpha, matriks
OOF)')
print('-' * 60)
print(df_grid_ridge.to_string(index=False))
print(f'\n>> Alpha terpilih: {ridge_alpha} (Mean R2 CV
tertinggi)')

meta_learner = Ridge(alpha=ridge_alpha)
meta_learner.fit(oof_predictions, y_train_log)

nama_base = list(base_learners.keys())
df_pengaruh = pd.DataFrame({
    'Base learner': nama_base,
    'Koefisien Ridge': meta_learner.coef_,
    'Abs(Koefisien)': np.abs(meta_learner.coef_)
}).sort_values('Abs(Koefisien)',
ascending=False).reset_index(drop=True)
df_pengaruh['Kontribusi Relatif (%)'] =
(df_pengaruh['Abs(Koefisien)']
df_pengaruh['Abs(Koefisien)'].sum() * 100).round(2)

print('\n' + '=' * 60)
print('BESAR PENGARUH TIAP BASE LEARNER TERHADAP STACKING
ENSEMBLE')
print('=' * 60)
print(df_pengaruh.to_string(index=False))
print(f'\nIntercept meta learner :
{meta_learner.intercept_:.6f}')

# Visualisasi kontribusi relatif tiap base learner
plt.figure(figsize=(8, 4.5))
bars = plt.barh(df_pengaruh['Base learner'],
df_pengaruh['Kontribusi Relatif (%)'],
color='mediumseagreen', edgecolor='black')
plt.xlabel('Kontribusi Relatif terhadap Stacking Ensemble (%)')
plt.title('Besar Pengaruh Tiap Base learner pada Meta learner
(Ridge)', fontweight='bold')
for bar, val in zip(bars, df_pengaruh['Kontribusi Relatif (%)']):
    plt.text(bar.get_width() + 0.5, bar.get_y() +
bar.get_height()/2, f'{val:.2f}%',
va='center', fontsize=9)
plt.tight_layout()
plt.show()

print('\n' + '=' * 60)
print('BAGIAN 6: TAHAP 4 -- RETRAIN BASE LEARNER PADA SELURUH 70%
TRAINING')
print('=' * 60)

trained_base_learners = {}
r2_train_per_model = {}

```

```

for nama, model in base_learners.items():
    m_full = copy.deepcopy(model)
    if nama == 'Linear Regression':
        m_full.fit(X_train_scaled, y_train_log)
        pred_train_log = m_full.predict(X_train_scaled)
    else:
        m_full.fit(X_train, y_train_log)
        pred_train_log = m_full.predict(X_train)
    trained_base_learners[nama] = m_full
    r2_train = r2_score(y_train_log, pred_train_log)
    r2_train_per_model[nama] = r2_train
    print(f'{nama:<20} R2 pada train (skala log) : {r2_train:.4f}')

pred_stack_train_log = meta_learner.predict(oof_predictions)
r2_stack_train = r2_score(y_train_log, pred_stack_train_log)
r2_train_per_model['Stacking Ensemble'] = r2_stack_train

print('\n' + '=' * 60)
print('PERBANDINGAN R2 TRAIN: STACKING ENSEMBLE vs BASE LEARNER')
print('=' * 60)

df_train_compare = pd.DataFrame({
    'Model': list(r2_train_per_model.keys()),
    'R2 Train (skala log)': [round(v, 4) for v in
r2_train_per_model.values()]
}).sort_values('R2 Train (skala log)',
ascending=False).reset_index(drop=True)
print(df_train_compare.to_string(index=False))

selisih = r2_stack_train - max(v for k, v in
r2_train_per_model.items() if k != 'Stacking Ensemble')
print(f"\nSelisih R2 Stacking Ensemble terhadap base learner
terbaik : {selisih:+.4f}")

pred_stack_train_asli = np.exp(pred_stack_train_log)
y_train_asli = np.exp(y_train_log)

r2_train_rp = r2_score(y_train_asli, pred_stack_train_asli)
rmse_train_rp = np.sqrt(mean_squared_error(y_train_asli,
pred_stack_train_asli))
mae_train_rp = mean_absolute_error(y_train_asli,
pred_stack_train_asli)

print(f'R2 Train (skala Rupiah) : {r2_train_rp:.4f}')
print(f'RMSE Train (skala Rupiah) : Rp {rmse_train_rp:,.0f}')
print(f'MAE Train (skala Rupiah) : Rp {mae_train_rp:,.0f}')

print('\n' + '=' * 60)
print('BAGIAN 7: TAHAP 5 -- EVALUASI AKHIR PADA DATA 30% TEST')
print('=' * 60)
print('Data 30% test baru disentuh di sini untuk pertama
kalinya.\n')

test_meta_features = np.zeros((X_test.shape[0],
len(trained_base_learners)))

prediksi_individual_log = {}

```

```

for idx, (nama, model) in enumerate(trained_base_learners.items()):
    if nama == 'Linear Regression':
        pred_log = model.predict(X_test_scaled)
    else:
        pred_log = model.predict(X_test)
    test_meta_features[:, idx] = pred_log
    prediksi_individual_log[nama] = pred_log

y_pred_stack_log = meta_learner.predict(test_meta_features)
prediksi_individual_log['Stacking Ensemble'] = y_pred_stack_log

print('Balik semua prediksi dari skala log ke skala asli (Rp) via
np.exp()...')
prediksi_individual = {nama: np.exp(pred_log) for nama, pred_log
in prediksi_individual_log.items()}
print('Selesai. Seluruh metrik di bawah ini dihitung pada skala
Rupiah (y_test asli).\n')

print('\n' + '=' * 60)
print('TABEL PERBANDINGAN PERFORMA (Data 30% Test, skala LOG,
TANPA np.exp())')
print('=' * 60)

hasil_evaluasi = []
urutan_model = ['Linear Regression', 'Random Forest', 'XGBoost',
'Stacking Ensemble']

for nama in urutan_model:
    pred = prediksi_individual[nama]
    r2 = r2_score(y_test, pred)
    rmse = np.sqrt(mean_squared_error(y_test, pred))
    mae = mean_absolute_error(y_test, pred)
    hasil_evaluasi.append({'Model': nama, 'R2': round(r2, 4),
'RMSE': round(rmse, 2), 'MAE': round(mae,
2)})

df_hasil = pd.DataFrame(hasil_evaluasi)

hasil_evaluasi_log = []
for nama in urutan_model:
    pred_log = prediksi_individual_log[nama]
    r2_log = r2_score(y_test_log, pred_log)
    rmse_log = np.sqrt(mean_squared_error(y_test_log, pred_log))
    mae_log = mean_absolute_error(y_test_log, pred_log)
    hasil_evaluasi_log.append({'Model': nama, 'R2 (log)':
round(r2_log, 4),
'RMSE (log)': round(rmse_log, 4),
'MAE (log)': round(mae_log, 4)})

df_hasil_log = pd.DataFrame(hasil_evaluasi_log)
print(df_hasil_log.to_string(index=False))

print('=' * 60)
print('TABEL PERBANDINGAN PERFORMA (Data 30% Test, skala
Rupiah)')
print('=' * 60)
print(df_hasil.to_string(index=False))

```

```

r2_stack      = df_hasil.loc[df_hasil['Model'] == 'Stacking Ensemble', 'R2'].values[0]
rmse_stack    = df_hasil.loc[df_hasil['Model'] == 'Stacking Ensemble', 'RMSE'].values[0]
mae_stack     = df_hasil.loc[df_hasil['Model'] == 'Stacking Ensemble', 'MAE'].values[0]

df_tampil = df_hasil.copy()
df_tampil['RMSE (Rp)'] = df_tampil['RMSE'].apply(lambda x: f'Rp {x:,.0f}')
df_tampil['MAE (Rp)'] = df_tampil['MAE'].apply(lambda x: f'Rp {x:,.0f}')
df_tampil_show = df_tampil[['Model', 'R2', 'RMSE (Rp)', 'MAE (Rp)']].copy()

print('\n' + '-' * 60)
best_r2      = df_hasil.loc[df_hasil['R2'].idxmax(), 'Model']
best_rmse    = df_hasil.loc[df_hasil['RMSE'].idxmin(), 'Model']
best_mae     = df_hasil.loc[df_hasil['MAE'].idxmin(), 'Model']
print(f'Model dengan R2 tertinggi      : {best_r2}')
print(f'Model dengan RMSE terendah    : {best_rmse}')
print(f'Model dengan MAE terendah     : {best_mae}')

print('\n' + '=' * 60)
print('BAGIAN 8: UJI SIGNIFIKANSI STATISTIK (Wilcoxon Signed-Rank Test)')
print('=' * 60)

errors = {}
for nama in urutan_model:
    errors[nama] = np.abs(y_test.values - prediksi_individual[nama])

pairs = [
    ('Stacking Ensemble', 'Random Forest'),
    ('Stacking Ensemble', 'XGBoost'),
    ('Stacking Ensemble', 'Linear Regression'),
    ('Random Forest', 'XGBoost'),
]

hasil_uji = []
for m1, m2 in pairs:
    try:
        stat, pval = wilcoxon(errors[m1], errors[m2])
    except ValueError:
        stat, pval = 0, 1.0
    if pval < 0.05:
        kesimpulan = f'{m1} LEBIH BAIK' if np.mean(errors[m1]) < np.mean(errors[m2]) else f'{m2} LEBIH BAIK'
    else:
        kesimpulan = 'TIDAK BERBEDA SIGNIFIKAN'
    print(f'{m1} vs {m2}<20 : stat={stat:,.0f}, p={pval:.6f} -> {kesimpulan}')
    hasil_uji.append({'Pasangan': f'{m1} vs {m2}', 'W': stat, 'p': pval, 'Kesimpulan': kesimpulan})

fig, axes = plt.subplots(1, 3, figsize=(16, 5))
fig.suptitle('Perbandingan Performa (Log-Transform Konsisten - Data 30% Test)',

```

```

        fontsize=11, fontweight='bold', y=1.02)

model_names = df_hasil['Model'].tolist()
colors = ['#4C72B0', '#DD8452', '#55A868', '#C44E52']

bars = axes[0].bar(model_names, df_hasil['R2'],
color=colors[:len(model_names)], edgecolor='white')
axes[0].set_title('R2 Score (lebih tinggi lebih baik)',
fontsize=11)
axes[0].set_ylim(0, 1.05)
axes[0].set_ylabel('R2')
axes[0].tick_params(axis='x', rotation=25)
for bar, val in zip(bars, df_hasil['R2']):
    axes[0].text(bar.get_x() + bar.get_width()/2,
bar.get_height() + 0.01,
f'{val:.4f}', ha='center', va='bottom',
fontsize=9)

rmse_miliar = df_hasil['RMSE'] / 1e9
bars2 = axes[1].bar(model_names, rmse_miliar,
color=colors[:len(model_names)], edgecolor='white')
axes[1].set_title('RMSE (lebih rendah lebih baik)', fontsize=11)
axes[1].set_ylabel('RMSE (Miliar Rupiah)')
axes[1].tick_params(axis='x', rotation=25)
for bar, val in zip(bars2, rmse_miliar):
    axes[1].text(bar.get_x() + bar.get_width()/2,
bar.get_height() + 0.005,
f'{val:.3f}', ha='center', va='bottom',
fontsize=9)

mae_miliar = df_hasil['MAE'] / 1e9
bars3 = axes[2].bar(model_names, mae_miliar,
color=colors[:len(model_names)], edgecolor='white')
axes[2].set_title('MAE (lebih rendah lebih baik)', fontsize=11)
axes[2].set_ylabel('MAE (Miliar Rupiah)')
axes[2].tick_params(axis='x', rotation=25)
for bar, val in zip(bars3, mae_miliar):
    axes[2].text(bar.get_x() + bar.get_width()/2,
bar.get_height() + 0.005,
f'{val:.3f}', ha='center', va='bottom',
fontsize=9)

plt.tight_layout()
plt.savefig('hasil_perbandingan_model_v14_log.png', dpi=150,
bbox_inches='tight')
plt.show()
print('Visualisasi disimpan sebagai
hasil_perbandingan_model_v14_log.png')

fig, axes = plt.subplots(2, 2, figsize=(14, 12))
fig.suptitle('Aktual vs Prediksi v14 (Log-Transform Konsisten -
Data 30% Test)',
fontsize=13, fontweight='bold')

all_models_plot = {
    'Stacking Ensemble': prediksi_individual['Stacking
Ensemble'],
    'Linear Regression': prediksi_individual['Linear
Regression'],

```

```

    'Random Forest'      : prediksi_individual['Random Forest'],
    'XGBoost'            : prediksi_individual['XGBoost']
}

colors = ['#C44E52', '#4C72B0', '#DD8452', '#55A868']

for i, (nama, pred) in enumerate(all_models_plot.items()):
    ax = axes[i // 2][i % 2]
    ax.scatter(y_test, pred, alpha=0.3, s=8, color=colors[i])

    min_val = min(y_test.min(), pred.min())
    max_val = max(y_test.max(), pred.max())
    ax.plot([min_val, max_val], [min_val, max_val], 'r--',
            linewidth=1.5, label='Ideal (y=x)')

    r2_val = r2_score(y_test, pred)
    ax.set_title(f'{nama}\nR2 = {r2_val:.4f}', fontsize=11)
    ax.set_xlabel('Harga Aktual (Rp)')
    ax.set_ylabel('Harga Prediksi (Rp)')
    ax.legend(fontsize=8)
    ax.grid(alpha=0.3)

plt.tight_layout()
plt.savefig('scatter_aktual_vs_prediksi_v14_log.png', dpi=150,
bbox_inches='tight')
plt.show()
print('Scatter plot disimpan sebagai scatter_aktual_vs_prediksi_v14_log.png')

print('\n' + '=' * 60)
print('RINGKASAN AKHIR -- STACKING ENSEMBLE K1 v14 (Log-Transform Konsisten)')
print('=' * 60)

print(f'\n--- Informasi Dataset ---')
print(f'Total data asli : {df.shape[0]:,} baris')
print(f'Data training 70% (setelah IQR filter) : {X_train.shape[0]:,} baris')
print(f'Data test 30% (setelah IQR filter) : {X_test.shape[0]:,} baris')

print(f'\n--- Konfigurasi Eksperimen v14 ---')
print(f'Pembagian data : 70% training / 30% test (stratified sumber_kota)')
print(f'Transformasi target : log(harga), dibalik exp() sebelum evaluasi')
print(f'OOB Folds : {N_FOLDS}-Fold (KFold, shuffle=True, random_state=42)')
print(f'Base learner : Linear Regression, Random Forest, XGBoost')
print(f'Meta Learner : Ridge (alpha={ridge_alpha})')
print(f'Input Meta Learner : hanya OOF/test predictions (skala log)')
print(f'Tuning RF + XGB : GridSearchCV 5-Fold pada X_train (70%), target skala log')
print(f'Tuning Ridge : GridSearchCV 5-Fold pada matriks OOF (skala log)')
print(f'Smoothing TE factor : k = {SMOOTHING_K}')

```

```

print(f'random_state          : 42 (semua komponen)')

print(f'\n--- Hasil Evaluasi Akhir (Data 30% Test, skala Rupiah)
---')
print(df_tampil_show.to_string(index=False))

print(f'\n--- Hyperparameter Terpilih ---')
print(f'Random Forest   : {gs_rf.best_params_}')
print(f'XGBoost           : {gs_xgb.best_params_}')
print(f'Ridge (meta)      : alpha = {ridge_alpha} (dituning pada
matriks OOF, skala log)')

print(f'\n--- Koefisien Ridge (bobot base learner, skala log) --
-')
for nama, koef in zip(['Linear Regression', 'Random Forest',
'XGBoost'], meta_learner.coef_):
    print(f' {nama:<25} : {koef:.6f}')
print(f' Intercept                : {meta_learner.intercept_:.6f}')

print('\n' + '=' * 60)
print('SELESAI -- Stacking Ensemble K1 v14 (Log-Transform
Konsisten)')
print('=' * 60)

print('\n' + '=' * 60)
print('BAGIAN 12: REPEATED K-FOLD (10 x 5) -- LOG-TRANSFORM
KONSISTEN')
print('=' * 60)
print('Total fold evaluasi : 10 ulangan x 5 fold = 50 evaluasi
per model')
print('Catatan : Hyperparameter RF/XGB memakai hasil
tuning Bagian 4 (efisiensi).')
print('Tuning Ridge dilakukan fresh tiap
fold pada matriks OOF (skala log).\n')

N_REPEATS = 10
N_SPLITS = 5

rkf = RepeatedKfold(n_splits=N_SPLITS, n_repeats=N_REPEATS,
random_state=42)

best_params_rf = gs_rf.best_params_
best_params_xgb = gs_xgb.best_params_

rkf_results = {nama: {'r2': [], 'rmse': [], 'mae': []} for nama
in urutan_model}

X_full = pd.concat([X_train, X_test], ignore_index=True)
lokasi_train_series =
X_raw_train['lokasi'].reset_index(drop=True)
lokasi_test_series =
X_raw_test['lokasi'].reset_index(drop=True)
lokasi_full = pd.concat([lokasi_train_series,
lokasi_test_series], ignore_index=True)
y_full_log = pd.concat([y_train_log, y_test_log],
ignore_index=True)

FITUR_NUM = FITUR_BASE

```

```

print(f'Shape X_full (train+test bersih) : {X_full.shape}')
print(f'Shape y_full_log : {y_full_log.shape}\n')

for fold_idx, (tr_idx, val_idx) in enumerate(rkf.split(X_full)):

    ulangan = fold_idx // N_SPLITS + 1
    fold = fold_idx % N_SPLITS + 1

    if fold == 1:
        print(f'--- Ulangan {ulangan}/{N_REPEATS} ---')

        X_tr_num =
X_full.iloc[tr_idx][FITUR_NUM].reset_index(drop=True)
        X_val_num =
X_full.iloc[val_idx][FITUR_NUM].reset_index(drop=True)
        lok_tr = lokasi_full.iloc[tr_idx].reset_index(drop=True)
        lok_val = lokasi_full.iloc[val_idx].reset_index(drop=True)
        y_tr_log = y_full_log.iloc[tr_idx].reset_index(drop=True)
        y_val_log = y_full_log.iloc[val_idx].reset_index(drop=True)
        y_val_asli = np.exp(y_val_log)

        X_tr_num_clean = X_tr_num.copy()
        lok_tr_clean = lok_tr.copy()
        y_tr_clean_log = y_tr_log.copy()

        gm_fold = y_tr_clean_log.mean()
        te_tmp = X_tr_num_clean.copy()
        te_tmp['harga_log'] = y_tr_clean_log.values
        te_tmp['lokasi'] = lok_tr_clean.values
        lok_stats_fold =
te_tmp.groupby('lokasi')['harga_log'].agg(['mean', 'count'])
        lok_enc_fold = ((lok_stats_fold['count'] *
lok_stats_fold['mean'] + SMOOTHING_K * gm_fold)
/ (lok_stats_fold['count'] + SMOOTHING_K))
        X_tr_num_clean['lokasi_enc'] =
lok_tr_clean.map(lok_enc_fold).fillna(gm_fold).values
        X_val_enc = X_val_num.copy()
        X_val_enc['lokasi_enc'] =
lok_val.map(lok_enc_fold).fillna(gm_fold).values

        FITUR_FOLD = FITUR_NUM + ['lokasi_enc']
        X_tr_final =
X_tr_num_clean[FITUR_FOLD].reset_index(drop=True)
        X_val_final = X_val_enc[FITUR_FOLD].reset_index(drop=True)

        sc_fold = StandardScaler()
        X_tr_scaled = sc_fold.fit_transform(X_tr_final)
        X_val_scaled = sc_fold.transform(X_val_final)

        bl_fold = {
            'Linear Regression': LinearRegression(),
            'Random Forest':
RandomForestRegressor(**best_params_rf, random_state=42,
n_jobs=-1),
            'XGBoost': XGBRegressor(**best_params_xgb,
random_state=42, verbosity=0, n_jobs=-1)
        }

        kf_inner = KFold(n_splits=5, shuffle=True, random_state=42)

```

```

oof_fold = np.zeros((X_tr_final.shape[0], 3))

for bidx, (bname, bmodel) in enumerate(bl_fold.items()):
    for t_idx, v_idx in kf_inner.split(X_tr_final):
        if bname == 'Linear Regression':
            Xbt = X_tr_scaled[t_idx]; Xbv = X_tr_scaled[v_idx]
        else:
            Xbt = X_tr_final.iloc[t_idx]; Xbv =
X_tr_final.iloc[v_idx]
            ybt = y_tr_clean_log.iloc[t_idx]
            m_tmp = copy.deepcopy(bmodel)
            m_tmp.fit(Xbt, ybt)
            oof_fold[v_idx, bidx] = m_tmp.predict(Xbv)

gs_r_fold = GridSearchCV(
    Ridge(), {'alpha': [0.01, 0.1, 1.0, 10.0, 50.0, 100.0,
500.0, 1000.0]},
    cv=5, scoring='r2', n_jobs=-1
)
gs_r_fold.fit(oof_fold, y_tr_clean_log)
meta_fold = Ridge(alpha=gs_r_fold.best_params_['alpha'])
meta_fold.fit(oof_fold, y_tr_clean_log)

trained_fold = {}
for bname, bmodel in bl_fold.items():
    m_full = copy.deepcopy(bmodel)
    if bname == 'Linear Regression':
        m_full.fit(X_tr_scaled, y_tr_clean_log)
    else:
        m_full.fit(X_tr_final, y_tr_clean_log)
    trained_fold[bname] = m_full

val_meta = np.zeros((X_val_final.shape[0], 3))
pred_fold_ind_log = {}
for bidx, (bname, bmodel) in enumerate(trained_fold.items()):
    if bname == 'Linear Regression':
        p_log = bmodel.predict(X_val_scaled)
    else:
        p_log = bmodel.predict(X_val_final)
    val_meta[:, bidx] = p_log
    pred_fold_ind_log[bname] = p_log

    pred_fold_ind_log['Stacking Ensemble'] =
meta_fold.predict(val_meta)

for nama in urutan_model:
    p_log = pred_fold_ind_log[nama]
    p_asli = np.exp(p_log)
    rkf_results[nama]['r2'].append(r2_score(y_val_asli,
p_asli))
    rkf_results[nama]['rmse'].append(np.sqrt(mean_squared_er
ror(y_val_asli, p_asli)))
    rkf_results[nama]['mae'].append(mean_absolute_error(y_va
l_asli, p_asli))

print('\n' + '=' * 70)
print('HASIL REPEATED K-FOLD (10 x 5 = 50 fold) -- SKALA RUPIAH
ASLI')
print('=' * 70)

```

```

print(f'{"Model":<25} {"R2 Mean":>10} {"R2 Std":>10} {"RMSE Mean":>18} {"MAE Mean":>18}')
print('-' * 85)

rkf_summary = []
for nama in urutan_model:
    r2_mean = np.mean(rkf_results[nama]['r2'])
    r2_std = np.std(rkf_results[nama]['r2'])
    rmse_mean = np.mean(rkf_results[nama]['rmse'])
    mae_mean = np.mean(rkf_results[nama]['mae'])
    print(f'{nama:<25} {r2_mean:>10.4f} {r2_std:>10.4f} {rmse_mean:>15,.0f} Rp {mae_mean:>15,.0f} Rp')
    rkf_summary.append({'Model': nama, 'R2_mean': r2_mean, 'R2_std': r2_std, 'RMSE_mean': rmse_mean, 'MAE_mean': mae_mean})

df_rkf = pd.DataFrame(rkf_summary)

import json
with open('rkf_Stacking_log_e2e_1x.json', 'w') as f:
    json.dump(rkf_results['Stacking Ensemble']['r2'], f)
print(f"\n[Tersimpan] {len(rkf_results['Stacking Ensemble']['r2'])} skor R2 Stacking "
      f"-> rkf_Stacking_log_e2e_1x.json")

print('\n' + '=' * 60)
print('BAGIAN 12A: TABEL DETAIL PERFORMA PER FOLD')
print('=' * 60)

detail_fold = []
for i in range(N_REPEATS * N_SPLITS):
    ulangan = i // N_SPLITS + 1
    fold = i % N_SPLITS + 1
    baris = {'Ulangan': ulangan, 'Fold': fold}
    for nama in urutan_model:
        baris[f'{nama} - R2'] = rkf_results[nama]['r2'][i]
        baris[f'{nama} - RMSE'] = rkf_results[nama]['rmse'][i]
        baris[f'{nama} - MAE'] = rkf_results[nama]['mae'][i]
    detail_fold.append(baris)

df_fold_detail = pd.DataFrame(detail_fold)

kolom_r2 = ['Ulangan', 'Fold'] + [f'{nama} - R2' for nama in urutan_model]
df_fold_r2 = df_fold_detail[kolom_r2].copy()

pd.set_option('display.max_rows', None)
pd.set_option('display.width', 160)

print('\n--- Tabel R2 per Fold (semua model) ---')
print(df_fold_r2.to_string(index=False, float_format=lambda x: f'{x:.4f}'))

ringkasan = {'Ulangan': 'Mean', 'Fold': '-'}
ringkasan_std = {'Ulangan': 'Std', 'Fold': '-'}
for nama in urutan_model:
    ringkasan[f'{nama} - R2'] = df_fold_r2[f'{nama} - R2'].mean()

```

```

        ringkasan_std[f'{nama} - R2'] = df_fold_r2[f'{nama} -
R2'].std()
print('\n--- Ringkasan ---')
print(pd.DataFrame([ringkasan,
ringkasan_std]).to_string(index=False, float_format=lambda x:
f'{x:.4f}' if isinstance(x, float) else str(x)))

df_fold_detail.to_csv('rkf_detail_per_fold.csv', index=False)
print(f"\n[Tersimpan] Tabel detail 50 fold (R2, RMSE, MAE, semua
model) -> rkf_detail_per_fold.csv")

print('\n' + '=' * 60)
print('BAGIAN 12B: UJI SIGNIFIKANSI PADA REPEATED K-FOLD
(Wilcoxon)')
print('=' * 60)

pairs_rkf = [
    ('Stacking Ensemble', 'Random Forest'),
    ('Stacking Ensemble', 'XGBoost'),
    ('Stacking Ensemble', 'Linear Regression'),
    ('Random Forest', 'XGBoost'),
]

ALPHA = 0.05
ALPHA_BONFERRONI = ALPHA / len(pairs_rkf)

hasil_uji_rkf = []
for m1, m2 in pairs_rkf:
    r2_m1 = np.array(rkf_results[m1]['r2'])
    r2_m2 = np.array(rkf_results[m2]['r2'])
    try:
        stat, pval = wilcoxon(r2_m1, r2_m2)
    except ValueError:
        stat, pval = 0.0, 1.0
    sig_alpha = 'Ya' if pval < ALPHA else 'Tidak'
    sig_bonf = 'Ya' if pval < ALPHA_BONFERRONI else 'Tidak'
    if pval < ALPHA:
        lebih_baik = m1 if r2_m1.mean() > r2_m2.mean() else m2
        kesimpulan = f'{lebih_baik} lebih unggul (p<0.05)'
    else:
        kesimpulan = 'Tidak ada perbedaan signifikan'
    print(f'{m1} vs {m2}<20> : W={stat:.2f}, p={pval:.4f},
sig_a05={sig_alpha}, sig_bonf={sig_bonf}')
    hasil_uji_rkf.append({'Pasangan': f'{m1} vs {m2}', 'W_stat':
stat, 'p_value': pval,
                        'Kesimpulan': kesimpulan})

fig, axes = plt.subplots(1, 2, figsize=(14, 5))
fig.suptitle('Repeated K-Fold (10 x 5) - Distribusi R2 per Model
v14 (Log-Transform Konsisten)',
             fontsize=12, fontweight='bold')

colors_rkf = ['#4C72B0', '#DD8452', '#55A868', '#C44E52']

r2_data = [rkf_results[nama]['r2'] for nama in urutan_model]
bp = axes[0].boxplot(r2_data, labels=urutan_model,
patch_artist=True, notch=False)
for patch, color in zip(bp['boxes'], colors_rkf):
    patch.set_facecolor(color)

```

```

    patch.set_alpha(0.7)
    axes[0].set_title('Distribusi R2 (50 fold)', fontsize=11)
    axes[0].set_ylabel('R2')
    axes[0].tick_params(axis='x', rotation=20)
    axes[0].grid(axis='y', alpha=0.3)

    r2_means = [np.mean(rkf_results[n]['r2']) for n in urutan_model]
    r2_stds = [np.std(rkf_results[n]['r2']) for n in urutan_model]
    bars_rkf = axes[1].bar(urutan_model, r2_means, yerr=r2_stds,
    color=colors_rkf,
    capsize=5, edgecolor='white', alpha=0.85)
    axes[1].set_title('R2 Mean ± Std (50 fold)', fontsize=11)
    axes[1].set_ylabel('R2 Mean')
    axes[1].set_ylim(0, 1.1)
    axes[1].tick_params(axis='x', rotation=20)
    for bar, mean, std in zip(bars_rkf, r2_means, r2_stds):
        axes[1].text(bar.get_x() + bar.get_width()/2,
            bar.get_height() + std + 0.01,
            f'{mean:.4f} ± {std:.4f}',
            ha='center', va='bottom', fontsize=8)

    plt.tight_layout()
    plt.savefig('rkf_r2_distribusi_v14_log.png', dpi=150,
    bbox_inches='tight')
    plt.show()
    print('Visualisasi Repeated K-Fold disimpan sebagai
    rkf_r2_distribusi_v14_log.png')

    print('\n' + '=' * 60)
    print('PERBANDINGAN: SINGLE SPLIT vs REPEATED K-FOLD (v14, log
    konsisten)')
    print('=' * 60)
    print(f'{"Model":<25} {"Single Split R2":>16} {"RKf R2 Mean":>14}
    {"RKf R2 Std":>12}')
    print('-' * 70)
    for nama in urutan_model:
        r2_single = df_hasil.loc[df_hasil['Model'] == nama,
        'R2'].values[0]
        r2_rkf_m = df_rkf.loc[df_rkf['Model'] == nama,
        'R2_mean'].values[0]
        r2_rkf_s = df_rkf.loc[df_rkf['Model'] == nama,
        'R2_std'].values[0]
        print(f'{nama:<25} {r2_single:>16.4f} {r2_rkf_m:>14.4f}
        {r2_rkf_s:>12.4f}')
```

## Lampiran 6. Kode Program Antarmuka Pengujian Model

```

import streamlit as st
import numpy as np
import pandas as pd
import joblib
import json

# — PAGE CONFIG

st.set_page_config(
    page_title="House Predictor",
```

```

page_icon="🏠",
layout="wide",
)

# — CUSTOM CSS

st.markdown("""
<style>
                                @import
url('https://fonts.googleapis.com/css2?family=DM+Sans:wght@300;4
00;500;600;700&family=DM+Mono:wght@400;500&display=swap');

/* — Base — */
html, body, .stApp {
    background-color: #FFFFFF6 !important;
    font-family: 'DM Sans', sans-serif !important;
    color: #1A1A1A;
}
.main .block-container { padding: 0 !important; max-width:
100% !important; }
    section[data-testid="stMain"] > div { padding-top: 0
!important; }
    * { box-sizing: border-box; }

/* — Sidebar — */
[data-testid="stSidebar"] {
    background-color: #1A1A1A !important;
    border-right: none !important;
}
[data-testid="stSidebar"] > div { padding: 0 !important; }
[data-testid="stSidebar"] section { padding: 0 !important; }
    [data-testid="stSidebarUserContent"] { padding: 0 0 40px 0
!important; }

/* Sidebar text overrides */
[data-testid="stSidebar"] h1,
[data-testid="stSidebar"] h2,
[data-testid="stSidebar"] h3,
[data-testid="stSidebar"] p,
[data-testid="stSidebar"] span,
[data-testid="stSidebar"] div,
[data-testid="stSidebar"] label { color: #E8E8E2 !important;
}

/* Sidebar brand */
.sb-brand {
    padding: 28px 24px 22px;
    border-bottom: 1px solid #2E2E2E;
}
.sb-brand-label {
    font-size: 0.6rem;
    letter-spacing: 0.18em;
    color: #888880 !important;
    font-weight: 500;
    text-transform: uppercase;
    margin-bottom: 6px;
}
.sb-brand-name {

```

```

        font-size: 1.3rem;
        font-weight: 700;
        color: #F5F5F0 !important;
        letter-spacing: -0.02em;
        line-height: 1;
    }
    .sb-brand-name em {
        color: #C8B560 !important;
        font-style: normal;
    }

    /* Sidebar section */
    .sb-sec {
        font-size: 0.6rem;
        letter-spacing: 0.16em;
        color: #555550 !important;
        font-weight: 600;
        text-transform: uppercase;
        padding: 22px 24px 8px;
    }

    /* Sidebar inputs */
    [data-testid="stSidebar"] label {
        color: #A8A8A0 !important;
        font-size: 0.75rem !important;
        font-weight: 500 !important;
        letter-spacing: 0.01em !important;
    }
    [data-testid="stSidebar"] input[type="number"] {
        background-color: #252525 !important;
        color: #F5F5F0 !important;
        border: 1px solid #2E2E2E !important;
        border-radius: 6px !important;
        font-family: 'DM Mono', monospace !important;
        font-size: 0.88rem !important;
    }
    [data-testid="stSidebar"] input[type="number"]:focus {
        border-color: #C8B560 !important;
        box-shadow: 0 0 0 2px rgba(200,181,96,0.15) !important;
    }
    [data-testid="stSidebar"] [data-baseweb="select"] > div {
        background-color: #252525 !important;
        border: 1px solid #2E2E2E !important;
        border-radius: 6px !important;
        color: #F5F5F0 !important;
    }
    [data-testid="stSidebar"] [data-baseweb="select"] * {
        color: #F5F5F0 !important;
        background-color: #252525 !important;
    }
    [data-testid="stSidebar"] [role="option"]:hover {
        background-color: #2E2E2E !important;
    }

    /* Predict button */
    .stButton > button {
        width: 100% !important;
        background: #C8B560 !important;
        color: #1A1A1A !important;
    }

```

```

    font-weight: 700 !important;
    font-size: 0.8rem !important;
    letter-spacing: 0.1em !important;
    text-transform: uppercase !important;
    border: none !important;
    border-radius: 6px !important;
    padding: 13px !important;
    font-family: 'DM Sans', sans-serif !important;
    transition: opacity 0.15s !important;
}
.stButton > button:hover { opacity: 0.85 !important; }

/* — Main area — */
.main-wrap { padding: 40px 52px; }

/* Hero */
.hero { padding: 72px 0 0; max-width: 520px; }
.hero-eyebrow {
    font-size: 0.68rem;
    letter-spacing: 0.16em;
    color: #1A1A1A;
    font-weight: 500;
    text-transform: uppercase;
    margin-bottom: 18px;
}
.hero-title {
    font-size: clamp(2.2rem, 4.5vw, 3.6rem);
    font-weight: 700;
    line-height: 1.08;
    color: #1A1A1A;
    letter-spacing: -0.03em;
    margin-bottom: 24px;
}
.hero-title strong { color: #C8B560; font-weight: 700; }
.hero-body {
    font-size: 0.92rem;
    color: #1A1A1A;
    line-height: 1.7;
    max-width: 400px;
}

/* Section label */
.sec-label {
    font-size: 0.62rem;
    letter-spacing: 0.14em;
    color: #888880;
    font-weight: 600;
    text-transform: uppercase;
    margin: 32px 0 14px;
}

/* Result header */
.res-header {
    padding-bottom: 20px;
    border-bottom: 1px solid #E0E0D8;
    margin-bottom: 28px;
}
.res-model {
    font-size: 0.7rem;

```

```

        letter-spacing: 0.14em;
        color: #888880;
        font-weight: 600;
        text-transform: uppercase;
        margin-bottom: 6px;
    }
    .res-title {
        font-size: 1.6rem;
        font-weight: 700;
        color: #1A1A1A;
        letter-spacing: -0.02em;
    }
}

/* Price box (hero result -- dibuat lebih besar/menonjol) */
.price-box {
    background: #1A1A1A;
    border-radius: 14px;
    padding: 48px 46px;
    margin: 0 0 32px;
    position: relative;
    overflow: hidden;
}
.price-box::after {
    content: '';
    position: absolute;
    top: 0; left: 0;
    width: 5px; height: 100%;
    background: #C8B560;
    border-radius: 14px 0 0 14px;
}
.pb-eyebrow {
    font-size: 0.66rem;
    letter-spacing: 0.16em;
    color: #C8B560 !important;
    font-weight: 600;
    text-transform: uppercase;
    margin-bottom: 16px;
}
.pb-price {
    font-size: clamp(2.4rem, 5vw, 3.6rem);
    font-weight: 700;
    color: #F5F5F0 !important;
    font-family: 'DM Mono', monospace;
    letter-spacing: -0.03em;
    line-height: 1;
    margin-bottom: 16px;
}
.pb-note {
    font-size: 0.86rem;
    color: #888880 !important;
    line-height: 1.6;
}
.pb-note strong { color: #E8E8E2 !important; }

/* Summary table */
.sum-wrap {
    background: #FFFFFF;
    border: 1px solid #E0E0D8;
    border-radius: 10px;

```

```

        overflow: hidden;
        margin-bottom: 20px;
    }
    .sum-table { width: 100%; border-collapse: collapse; }
    .sum-table tr { border-bottom: 1px solid #F0F0E8; }
    .sum-table tr:last-child { border-bottom: none; }
    .sum-table td {
        padding: 10px 18px;
        font-size: 0.84rem;
        color: #888880;
    }
    .sum-table td:last-child {
        text-align: right;
        font-family: 'DM Mono', monospace;
        font-size: 0.8rem;
        font-weight: 500;
        color: #1A1A1A;
    }
}

/* Expander (dropdown) styling */
[data-testid="stExpander"] {
    background: #FFFFFF !important;
    border: 1px solid #E0E0D8 !important;
    border-radius: 10px !important;
    margin-bottom: 14px !important;
}
[data-testid="stExpander"] summary {
    color: #1A1A1A !important;
    font-size: 0.85rem !important;
    font-weight: 600 !important;
}
[data-testid="stExpander"] summary:hover { color: #C8B560 !important; }
.cmp-row {
    display: flex;
    align-items: center;
    padding: 11px 0;
    border-bottom: 1px solid #F5F5F0;
    gap: 16px;
}
.cmp-row:last-child { border-bottom: none; }
.cmp-name {
    font-size: 0.84rem;
    font-weight: 500;
    color: #888880;
    width: 170px;
    flex-shrink: 0;
}
.cmp-price {
    font-size: 0.9rem;
    font-weight: 600;
    color: #1A1A1A;
    font-family: 'DM Mono', monospace;
    flex: 1;
}
.cmp-metrics {
    font-size: 0.73rem;
    color: #AAAAAA;
    font-family: 'DM Mono', monospace;

```

```

        text-align: right;
        white-space: nowrap;
    }
    .cmp-row.cmp-selected .cmp-name { color: #1A1A1A; font-weight: 700; }
    .cmp-row.cmp-selected { background: #F5F5F0; margin: 0 -8px; padding: 11px 8px; border-radius: 6px; }

    /* Streamlit tweaks */
    [data-testid="stNumberInput"] > div { background: transparent !important; }
</style>
"""', unsafe_allow_html=True)

```

| # | — | KONFIGURASI | MODEL |
|---|---|-------------|-------|
|---|---|-------------|-------|

```

# PENTING: sesuaikan flag ini dengan versi .pkl yang sedang dipakai di
# folder model_artifacts/.
#
# False -> pkl LAMA (model dilatih & prediksi langsung dalam skala Rupiah,
#           TANPA log-transform). Gunakan ini selama kamu belum export
#           pkl dari notebook v14 (log end-to-end).
# True  -> pkl BARU dari notebook v14 (model dilatih pada loglp(harga),
#           perlu np.expml() untuk balik ke skala Rupiah).
#
# Jangan lupa ganti ke True begitu kamu sudah export & copy pkl terbaru,
# atau prediksi akan salah (masih dalam skala log, bukan Rupiah).
MODEL_IS_LOG_SCALE = True

```

| # | — | LOAD | ARTIFACTS |
|---|---|------|-----------|
|---|---|------|-----------|

```

@st.cache_resource
def load_artifacts():
    models = {
        "Linear Regression":
joblib.load("model_artifacts/model_lr.pkl"),
        "Random Forest":
joblib.load("model_artifacts/model_rf.pkl"),
        "XGBoost":
joblib.load("model_artifacts/model_xgb.pkl"),
    }
    meta_learner = joblib.load("model_artifacts/model_meta.pkl")
    scaler = joblib.load("model_artifacts/scaler.pkl")

    with open("model_artifacts/encoding_data.json") as f:
        enc_data = json.load(f)
    with open("model_artifacts/kota_mapping.json") as f:
        kota_mapping = json.load(f)
    with open("model_artifacts/metrics.json") as f:
        metrics = json.load(f)
    with open("model_artifacts/kecamatan_per_kota.json", encoding="utf-8") as f:
        kec_per_kota = json.load(f)
    with open("model_artifacts/fitur_order.json") as f:

```

```

        fitur_order = json.load(f)

        return models, meta_learner, scaler, enc_data, kota_mapping,
metrics, kec_per_kota, fitur_order

try:
    models, meta_learner, scaler, enc_data, kota_mapping,
metrics, kec_per_kota, fitur_order = load_artifacts()
    artifacts_ok = True
except FileNotFoundError as e:
    artifacts_ok = False
    missing = str(e)

# _____ HELPERS
_____

def fmt_rp(v: float) -> str:
    return f"Rp {v:,.0f},-".replace(",", ".")

def fmt_rp_short(v: float) -> str:
    if v >= 1_000_000_000:
        return f"{v/1_000_000_000:.2f}M"
    return f"{v/1_000_000:.1f}Jt"

def build_X(lb, lt, kt, km, kec, kab):
    lokasi_enc = enc_data["lokasi_enc_map"].get(kec,
enc_data["global_mean"])
    kota_enc = kota_mapping.get(kab, 0)
    raw = {
        "kamar_tidur": kt, "kamar_mandi": km,
        "luas_tanah": lt, "luas_bangunan": lb,
        "sumber_kota_enc": kota_enc,
        "lokasi_enc": lokasi_enc,
    }
    return np.array([[raw[f] for f in fitur_order]])

def predict_all(X):
    """
    Menghitung prediksi harga dari base learner + Stacking meta
learner.

    Perilaku tergantung MODEL_IS_LOG_SCALE (lihat blok
konfigurasi di atas):
    - True : base learner & meta learner beroperasi dalam skala
LOG
(mengikuti Bagian 6-7 notebook v14). np.expm1()
diterapkan
di akhir, SETELAH Stacking selesai, bukan sebelum.
- False : base learner & meta learner beroperasi langsung
dalam
skala Rupiah (kompatibel dengan pk1 lama, tanpa
log-transform).
    """
    X_sc = scaler.transform(X)

    pred_raw = {
        "Linear Regression": float(models["Linear
Regression"].predict(X_sc)[0]),

```

```

        "Random Forest": float(models["Random
Forest"].predict(X)[0]),
        "XGBoost": float(models["XGBoost"].predict(X)
[0]),
    }

    meta_feat = np.array([[pred_raw["Linear Regression"],
                             pred_raw["Random Forest"],
                             pred_raw["XGBoost"]]])
    pred_raw["Stacking Ensemble"] =
float(meta_learner.predict(meta_feat)[0])

    if MODEL_IS_LOG_SCALE:
        # Balik ke skala Rupiah SETELAH proses Stacking selesai
        preds = {nama: float(np.expml(v)) for nama, v in
pred_raw.items()}
    else:
        preds = pred_raw

    return preds

# _____ SIDEBAR
_____
with st.sidebar:
    st.markdown("""
<div class="sb-brand">
    <div class="sb-brand-label">ML · Research</div>
    <div class="sb-brand-name">House <em>Predictor</em></div>
</div>
""", unsafe_allow_html=True)

    if not artifacts_ok:
        st.error(f"Model tidak ditemukan:\n`{missing}`")
        st.stop()

        st.markdown('<div class="sb-sec">Model</div>',
unsafe_allow_html=True)
        selected_model = st.selectbox(
            "Pilih Model", label_visibility="collapsed",
            options=["Stacking Ensemble", "Random Forest", "XGBoost",
"Linear Regression"],
            index=0
        )

        st.markdown('<div class="sb-sec">Parameter</div>',
unsafe_allow_html=True)
        luas_bangunan = st.number_input("Luas Bangunan (m²)",
min_value=10, max_value=5000, value=100, step=5)
        luas_tanah = st.number_input("Luas Tanah
(m²)", min_value=10, max_value=5000, value=120, step=5)
        kamar_tidur = st.number_input("Kamar
Tidur", min_value=1, max_value=20, value=3, step=1)
        kamar_mandi = st.number_input("Kamar
Mandi", min_value=1, max_value=20, value=2, step=1)

        st.markdown('<div class="sb-sec">Lokasi</div>',
unsafe_allow_html=True)

```

```

        kabupaten = st.selectbox("Kabupaten/Kota",
list(kec_per_kota.keys()), label_visibility="collapsed")
        kecamatan = st.selectbox("Kecamatan",
sorted(kec_per_kota.get(kabupaten, [])),
label_visibility="collapsed")
        st.markdown(
            f"<div style='padding:4px 0 16px 2px;font-
size:0.7rem;color:#555550;'> {kabupaten} · {kecamatan}</div>",
            unsafe_allow_html=True
        )

        st.markdown("<div style='height:8px'></div>",
unsafe_allow_html=True)
        predict_btn = st.button("Prediksi Harga",
use_container_width=True)

# — MAIN

st.markdown('<div class="main-wrap">', unsafe_allow_html=True)

if not predict_btn:
    st.markdown("""
    <div class="hero">
        <div class="hero-eyebrow">Machine Learning ·
Regression</div>
        <div class="hero-title">
            Estimasi Harga<br>
            Rumah <strong>Tinggal</strong>
        </div>
        <p class="hero-body">
            Masukkan spesifikasi properti dan lokasi pada panel
            kiri,
            lalu tekan <em>Prediksi Harga</em> untuk melihat
            estimasi
            dari empat model regresi sekaligus.
        </p>
    </div>
    """, unsafe_allow_html=True)

else:
    X_input = build_X(luas_bangunan, luas_tanah, kamar_tidur,
kamar_mandi,
                        kecamatan, kabupaten)
    all_preds = predict_all(X_input)
    harga = all_preds[selected_model]

    # Result header
    st.markdown(f"""
    <div class="res-header">
        <div class="res-model">{selected_model}</div>
        <div class="res-title">Hasil Prediksi</div>
    </div>
    """, unsafe_allow_html=True)

    # Price box -- fokus utama halaman
    st.markdown(f"""
    <div class="price-box">

```

```

<div class="pb-eyebrow">Estimasi Harga Properti</div>
<div class="pb-price">{fmt_rp(harga)}</div>
<div class="pb-note">
    Berdasarkan model <strong>{selected_model}</strong>
    · {kecamatan}, {kabupaten}
</div>
</div>
""", unsafe_allow_html=True)

# Dropdown 1 – Model Lain (perbandingan harga prediksi semua
model)
urutan_tampil = ["Stacking Ensemble", "Random Forest",
"XGBoost", "Linear Regression"]
with st.expander("Model Lain"):
    rows = ""
    for nama in urutan_tampil:
        selected_cls = " cmp-selected" if nama ==
selected_model else ""
        tanda = " (dipilih)" if nama == selected_model else
""
        rows += f"""
        <div class="cmp-row{selected_cls}">
            <div class="cmp-name">{nama}{tanda}</div>
            <div class="cmp-
price">{fmt_rp(all_preds[nama])}</div>
            </div>"""
    st.markdown(rows, unsafe_allow_html=True)

# Dropdown 2 Hasil Validasi Model (MAE/RMSE/R2, data test
keseluruhan)
with st.expander("Hasil Validasi Model"):
    st.caption(
        "Angka MAE/RMSE/R2 di bawah ini dihitung sekali dari
seluruh "
        "data test saat training, tidak berubah walau kamu
ganti "
        "parameter input."
    )
    rows = ""
    for nama in urutan_tampil:
        selected_cls = " cmp-selected" if nama ==
selected_model else ""
        tanda = " (dipilih)" if nama == selected_model else
""
        mm = metrics[nama]
        rows += f"""
        <div class="cmp-row{selected_cls}">
            <div class="cmp-name">{nama}{tanda}</div>
            <div class="cmp-metrics">
                MAE {fmt_rp_short(mm["MAE"])} &nbsp; ·&nbsp; &nbsp;
                RMSE {fmt_rp_short(mm["RMSE"])} &nbsp; ·&nbsp; &nbsp;
                R2 {mm["R2"]:.4f}
            </div>
        </div>"""
    st.markdown(rows, unsafe_allow_html=True)

# Dropdown 3 – Ringkasan Input (spesifikasi yang diinput user)
with st.expander("Ringkasan Input"):
    col_l, col_r = st.columns(2, gap="medium")

```

```

with col_l:
    st.markdown(f"""
    <div class="sum-wrap">
    <table class="sum-table">
        <tr><td>Luas Bangunan</td><td>{luas_bangunan}
m²</td></tr>
        <tr><td>Luas Tanah</td><td>{luas_tanah}
m²</td></tr>
        <tr><td>Kamar
Tidur</td><td>{kamar_tidur}</td></tr>
        <tr><td>Kamar
Mandi</td><td>{kamar_mandi}</td></tr>
    </table>
    </div>
    """, unsafe_allow_html=True)
with col_r:
    st.markdown(f"""
    <div class="sum-wrap">
    <table class="sum-table">
        <tr><td>Kabupaten</td><td>{kabupaten}</td></tr>
        <tr><td>Kecamatan</td><td>{kecamatan}</td></tr>
    </table>
    </div>
    """, unsafe_allow_html=True)
st.markdown('</div>', unsafe_allow_html=True)

```

