praktikum9

November 27, 2025

1 Laporan Praktikum 9

Prediksi Harga Rumah Menggunakan Regresi Linear

1.1 Identitas Praktikum

Mata Kuliah : Data Sceince
Pertemuan : 9
Judul : Regresi Linear untuk Prediksi Variabel Kontinu
Kelas : 29 Informatika

Nama : Peno
NIM : 221220095
Tanggal : 27 November 2025

1.2 Pendahuluan

Regresi linear merupakan salah satu metode dasar dalam machine learning yang digunakan untuk memodelkan hubungan antara satu atau lebih fitur dengan sebuah variabel target kontinu, seperti harga rumah. Dalam konteks data harga rumah, model regresi linear dapat membantu melakukan estimasi harga berdasarkan karakteristik fisik dan fasilitas rumah yang ada.

Tujuan praktikum ini adalah untuk:
- Melakukan eksplorasi dan analisis dataset harga rumah.
- Melakukan preprocessing data (handling outliers, encoding, dan feature scaling).
- Membangun model regresi linear serta variasinya dengan regularisasi Ridge dan Lasso.
- Mengevaluasi performa model menggunakan berbagai metrik dan analisis residual.
- Membandingkan performa beberapa model dan memilih model terbaik

Dataset yang digunakan adalah `Housing.csv` yang berisi 545 data rumah dengan 13 kolom fitur, terdiri dari fitur numerik (misalnya luas area, jumlah kamar) dan fitur kategorikal (misalnya akses jalan utama, keberadaan AC, dan status furnitur).

[25]:
```python
# Data manipulation
import numpy as np
import pandas as pd

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Machine Learning
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Statistics
from scipy import stats

# Settings
import warnings
warnings.filterwarnings('ignore')
sns.set_style('whitegrid')
plt.rcParams['figure.figsize'] = (10, 6)

print("Libraries imported successfully!")
```

```
Libraries imported successfully!
```

[26]:
```
# Load dataset
data = pd.read_csv('Housing.csv')

# Tampilkan 5 baris pertama
print("=" * 50)
print("SAMPLE DATA")
print("=" * 50)
print(data.head())

# Informasi dataset
print("\n" + "=" * 50)
print("DATASET INFO")
print("=" * 50)
print(data.info())

# Statistik deskriptif
print("\n" + "=" * 50)
print("DESCRIPTIVE STATISTICS")
print("=" * 50)
print(data.describe())

# Cek dimensi data
print(f"\nJumlah baris: {data.shape[0]}")
print(f"Jumlah kolom: {data.shape[1]}")
```

```
==================================================
SAMPLE DATA
==================================================
      price  area  bedrooms  bathrooms  stories mainroad guestroom basement  \
0  13300000  7420         4          2        3      yes        no       no
1  12250000  8960         4          4        4      yes        no       no
2  12250000  9960         3          2        2      yes        no      yes
3  12215000  7500         4          2        2      yes        no      yes
4  11410000  7420         4          1        2      yes       yes      yes

  hotwaterheating airconditioning  parking prefarea furnishingstatus
0              no             yes        2      yes        furnished
1              no             yes        3       no        furnished
2              no              no        2      yes   semi-furnished
3              no             yes        3      yes        furnished
4              no             yes        2       no        furnished

==================================================
DATASET INFO
==================================================
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   price             545 non-null    int64
 1   area              545 non-null    int64
 2   bedrooms          545 non-null    int64
 3   bathrooms         545 non-null    int64
 4   stories           545 non-null    int64
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
 7   basement          545 non-null    object
 8   hotwaterheating   545 non-null    object
 9   airconditioning   545 non-null    object
 10  parking           545 non-null    int64
 11  prefarea          545 non-null    object
 12  furnishingstatus  545 non-null    object
dtypes: int64(6), object(7)
memory usage: 55.5+ KB
None

==================================================
DESCRIPTIVE STATISTICS
==================================================
              price         area    bedrooms   bathrooms     stories  \
count  5.450000e+02   545.000000  545.000000  545.000000  545.000000
mean   4.766729e+06  5150.541284    2.965138    1.286239    1.805505
std    1.870440e+06  2170.141023    0.738064    0.502470    0.867492
min    1.750000e+06  1650.000000    1.000000    1.000000    1.000000
25%    3.430000e+06  3600.000000    2.000000    1.000000    1.000000
```

```
50%    4.340000e+06   4600.000000    3.000000    1.000000    2.000000
75%    5.740000e+06   6360.000000    3.000000    2.000000    2.000000
max    1.330000e+07  16200.000000    6.000000    4.000000    4.000000

          parking
count  545.000000
mean     0.693578
std      0.861586
min      0.000000
25%      0.000000
50%      0.000000
75%      1.000000
max      3.000000

Jumlah baris: 545
Jumlah kolom: 13
```

1.3 Eksplorasi Data (EDA)

1.4 Task 1.1 – Eksplorasi Data

Berdasarkan hasil eksplorasi awal:

1. Jumlah total data yang ada adalah **545 baris** dan **13 kolom** sesuai hasil `data.shape`.
2. Rata-rata (mean) harga rumah (`price`) adalah sekitar **4.766.729,25**.
3. Harga rumah termahal dan termurah adalah:
   - Harga maksimum: **13.300.000**.

   - Harga minimum: **1.750.000**.
4. Tipe data fitur:
   - Fitur numerik: `price`, `area`, `bedrooms`, `bathrooms`, `stories`, `parking`.

   - Fitur kategorikal: `mainroad`, `guestroom`, `basement`, `hotwaterheating`, `airconditioning`, `prefarea`, `furnishingstatus`.

[27]:
```python
# Cek missing values
print("=" * 50)
print("MISSING VALUES")
print("=" * 50)
missing = data.isnull().sum()
print(missing[missing > 0])

if missing.sum() == 0:
    print("Tidak ada missing values!")
else:
    print(f"\nTotal missing values: {missing.sum()}")

# Visualisasi missing values (jika ada)
if missing.sum() > 0:
    plt.figure(figsize=(10, 4))
    missing[missing > 0].plot(kind='bar')
    plt.title('Missing Values per Column')
    plt.ylabel('Count')
    plt.xticks(rotation=45)
    plt.show()
```

```
==================================================
MISSING VALUES
==================================================
Series([], dtype: int64)
Tidak ada missing values!
```

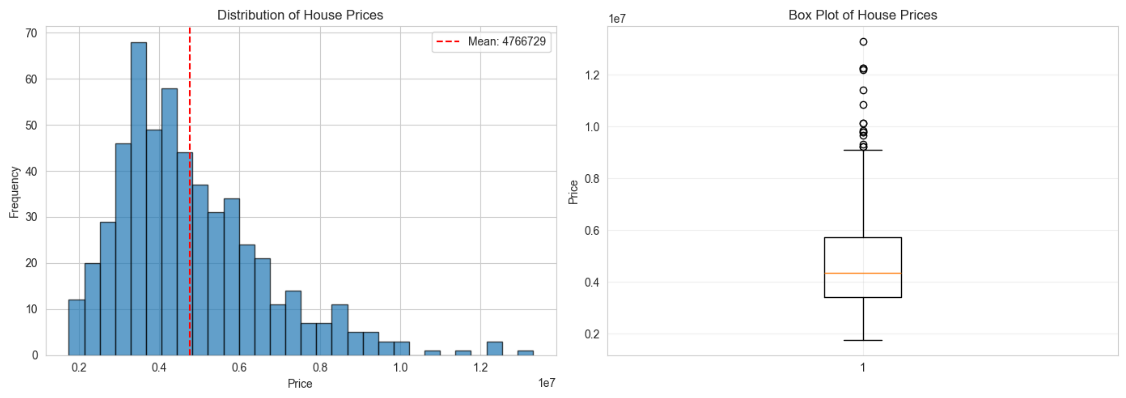
[28]:
```
# Identifikasi kolom kategorikal
categorical_cols = data.select_dtypes(include=['object']).columns
print("=" * 50)
print("CATEGORICAL FEATURES")
print("=" * 50)
print(f"Kolom kategorikal: {list(categorical_cols)}")

# Value counts untuk setiap kolom kategorikal
for col in categorical_cols:
    print(f"\n{col}:")
    print(data[col].value_counts())
    print(f"Unique values: {data[col].nunique()}")
```

```
==================================================
CATEGORICAL FEATURES
==================================================
Kolom kategorikal: ['mainroad', 'guestroom', 'basement', 'hotwaterheating',
'airconditioning', 'prefarea', 'furnishingstatus']

mainroad:
mainroad
yes    468
no      77
Name: count, dtype: int64
Unique values: 2

guestroom:
guestroom
no     448
yes     97
Name: count, dtype: int64
Unique values: 2

basement:
basement
```

```
no      354
yes     191
Name: count, dtype: int64
Unique values: 2

hotwaterheating:
hotwaterheating
no      520
yes      25
Name: count, dtype: int64
Unique values: 2

airconditioning:
airconditioning
no      373
yes     172
Name: count, dtype: int64
Unique values: 2

prefarea:
prefarea
no      417
yes     128
Name: count, dtype: int64
Unique values: 2

furnishingstatus:
furnishingstatus
semi-furnished    227
unfurnished       178
furnished         140
Name: count, dtype: int64
Unique values: 3
```

[29]:
```
# Distribusi harga rumah
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Histogram
axes[0].hist(data['price'], bins=30, edgecolor='black', alpha=0.7)
axes[0].set_xlabel('Price')
axes[0].set_ylabel('Frequency')
axes[0].set_title('Distribution of House Prices')
axes[0].axvline(data['price'].mean(), color='red',
                linestyle='--', label=f"Mean: {data['price'].mean():.0f}")
axes[0].legend()

# Box plot
```

```
axes[1].boxplot(data['price'])
axes[1].set_ylabel('Price')
axes[1].set_title('Box Plot of House Prices')
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Statistik
print(f"Mean: {data['price'].mean():.2f}")
print(f"Median: {data['price'].median():.2f}")
print(f"Std Dev: {data['price'].std():.2f}")
print(f"Skewness: {data['price'].skew():.2f}")
```

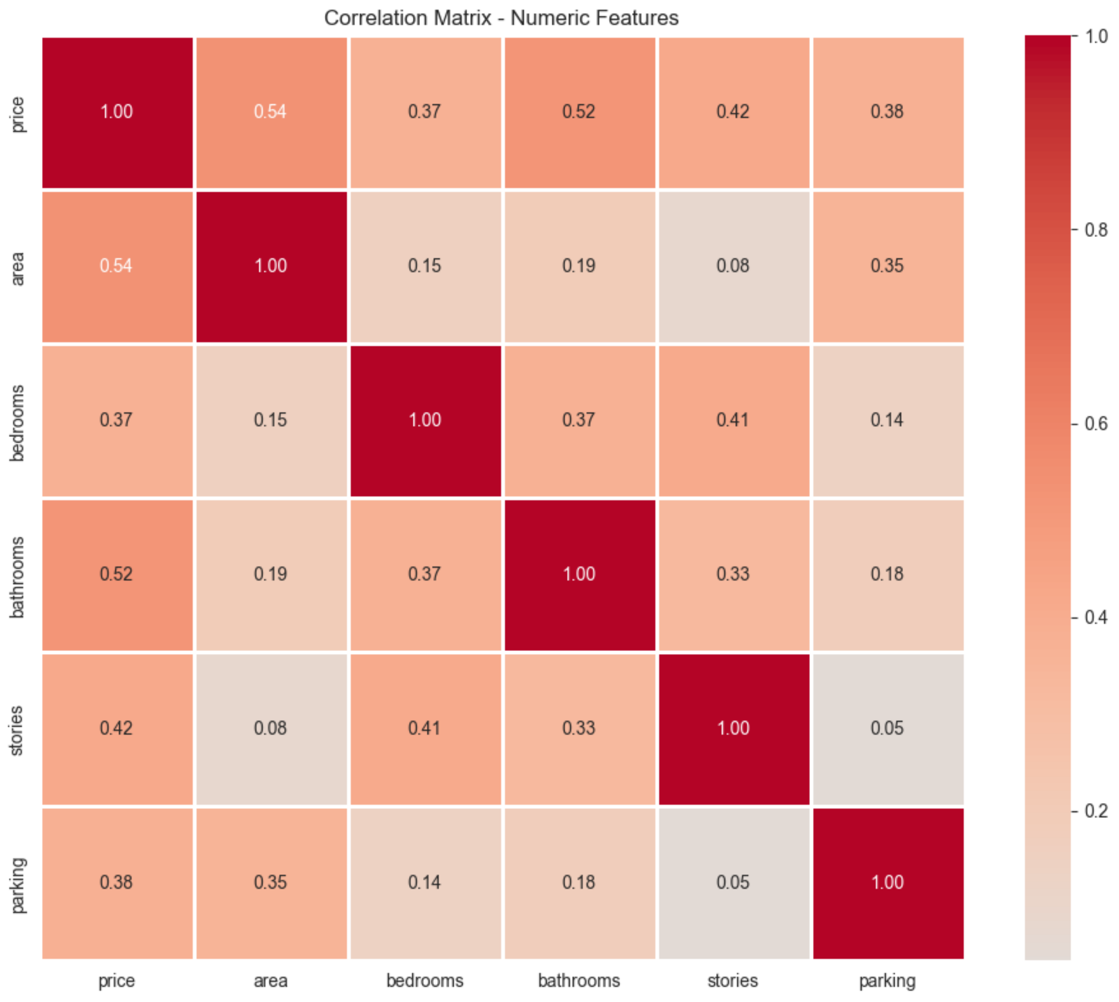


```
Mean: 4766729.25
Median: 4340000.00
Std Dev: 1870439.62
Skewness: 1.21
```

[30]:
```
# Korelasi dengan target (hanya fitur numerik)
numeric_cols = data.select_dtypes(include=[np.number]).columns
correlation = data[numeric_cols].corr()['price'].sort_values(ascending=False)

print("=" * 50)
print("CORRELATION WITH PRICE")
print("=" * 50)
print(correlation)

# Visualisasi correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(data[numeric_cols].corr(), annot=True,
            cmap='coolwarm', center=0, fmt='.2f',
            square=True, linewidths=1)
```

```
plt.title('Correlation Matrix - Numeric Features')
plt.tight_layout()
plt.show()
```

```
==================================================
CORRELATION WITH PRICE
==================================================
price        1.000000
area         0.535997
bathrooms    0.517545
stories      0.420712
parking      0.384394
bedrooms     0.366494
Name: price, dtype: float64
```



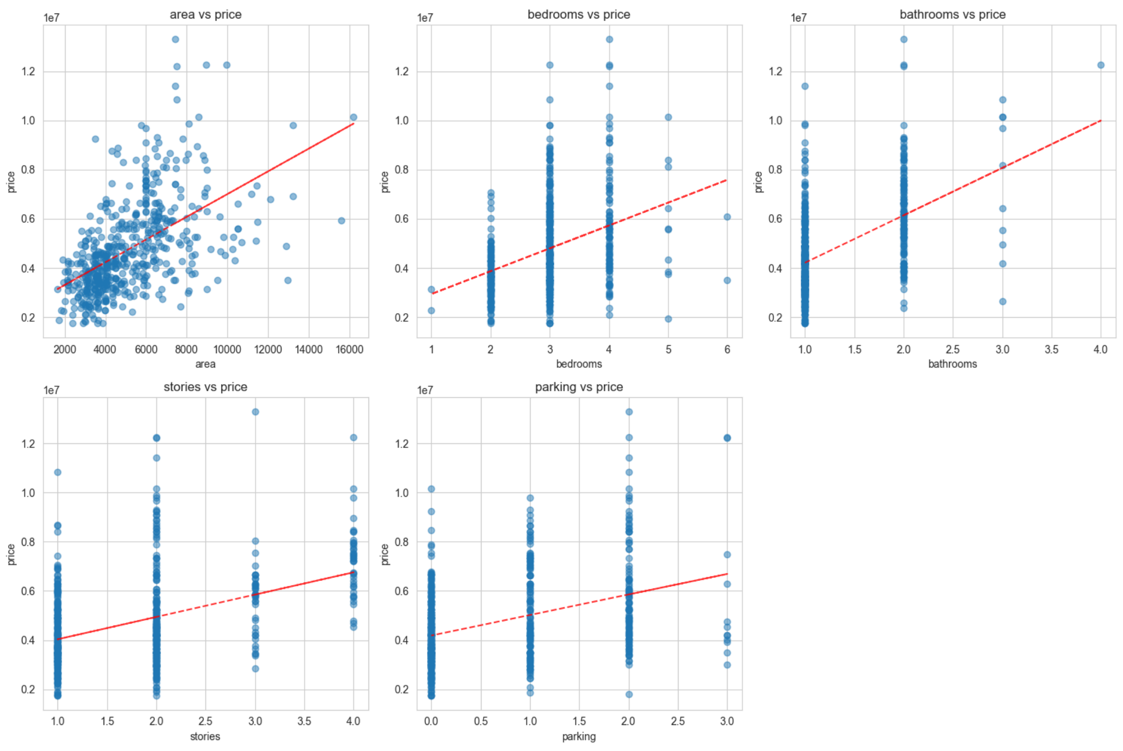
1.5 Task 1.2 – Analisis Korelasi

Dari matriks korelasi fitur numerik terhadap `price`, tiga fitur dengan korelasi tertinggi adalah:
- `area` dengan korelasi sekitar **0,535997**.
- `bathrooms` dengan korelasi sekitar **0,517545**.
- `stories` dengan korelasi sekitar **0,420712**.

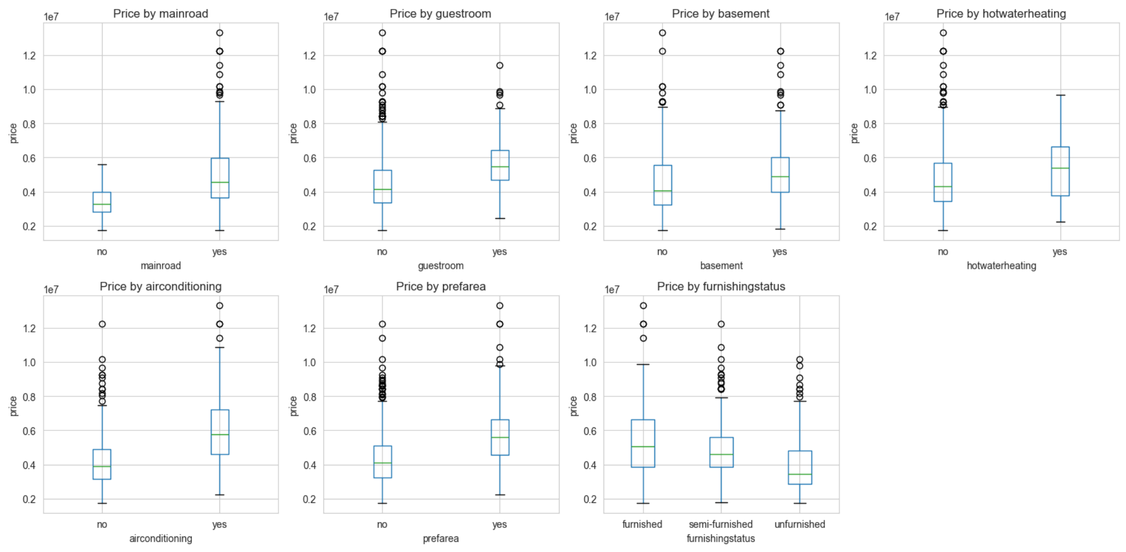
Tidak terdapat pasangan fitur numerik lain yang memiliki korelasi sangat tinggi mendekati 1 atau −1, sehingga indikasi multikolinearitas kuat antar fitur numerik relatif kecil.

Interpretasinya, korelasi positif pada `area`, `bathrooms`, dan `stories` menunjukkan bahwa rumah dengan luas yang lebih besar, lebih banyak kamar mandi, dan jumlah lantai lebih tinggi cenderung memiliki harga yang lebih mahal. Hal ini konsisten dengan intuisi bahwa ukuran dan fasilitas rumah berkontribusi signifikan terhadap harga.

[31]:
```python
# Scatter plots untuk fitur numerik vs price
numeric_features = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking']

fig, axes = plt.subplots(2, 3, figsize=(15, 10))
axes = axes.ravel()

for idx, col in enumerate(numeric_features):
    axes[idx].scatter(data[col], data['price'], alpha=0.5)
    axes[idx].set_xlabel(col)
    axes[idx].set_ylabel('price')
    axes[idx].set_title(f'{col} vs price')

    # Add trend line
    z = np.polyfit(data[col], data['price'], 1)
    p = np.poly1d(z)
    axes[idx].plot(data[col], p(data[col]), "r--", alpha=0.8)

# Remove extra subplot
fig.delaxes(axes[5])

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



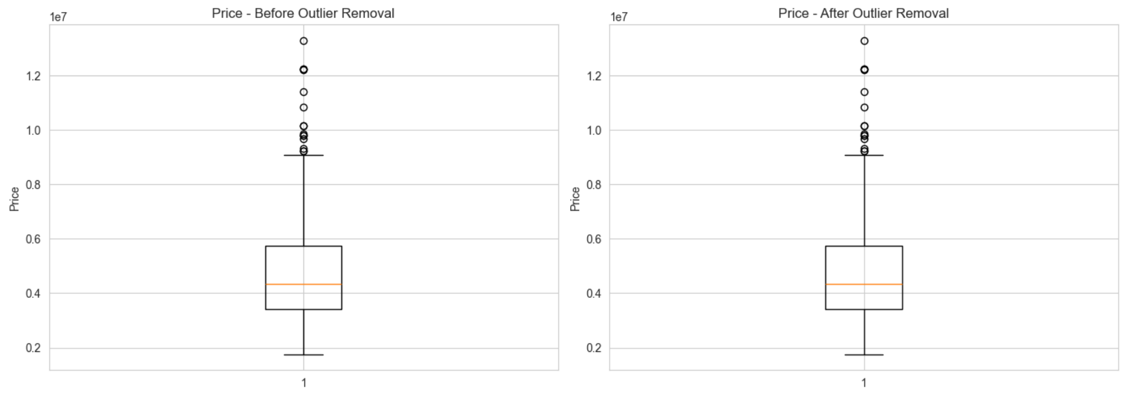
[32]:
```
# Box plots untuk fitur kategorikal vs price
categorical_features = ['mainroad', 'guestroom', 'basement',
                        'hotwaterheating', 'airconditioning',
                        'prefarea', 'furnishingstatus']

fig, axes = plt.subplots(2, 4, figsize=(16, 8))
axes = axes.ravel()

for idx, col in enumerate(categorical_features):
    data.boxplot(column='price', by=col, ax=axes[idx])
    axes[idx].set_title(f'Price by {col}')
    axes[idx].set_xlabel(col)
    axes[idx].set_ylabel('price')

# Remove extra subplot
fig.delaxes(axes[7])

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



[33]:
```python
# Deteksi outliers menggunakan IQR method
def detect_outliers_iqr(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = df[(df[column] < lower_bound) | (df[column] > upper_bound)]
    return outliers, lower_bound, upper_bound

# Cek outliers di price
outliers, lower, upper = detect_outliers_iqr(data, 'price')
print(f"Jumlah outliers di 'price': {len(outliers)}")
print(f"Lower bound: {lower:.0f}")
print(f"Upper bound: {upper:.0f}")

# Visualisasi sebelum dan sesudah handling outliers
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Before
axes[0].boxplot(data['price'])
axes[0].set_title('Price - Before Outlier Removal')
axes[0].set_ylabel('Price')

# Tidak remove outliers (sesuai modul)
data_clean = data.copy()

# After
axes[1].boxplot(data_clean['price'])
```

```
axes[1].set_title('Price - After Outlier Removal')
axes[1].set_ylabel('Price')

plt.tight_layout()
plt.show()

print(f"Data sebelum: {len(data)} baris")
print(f"Data sesudah: {len(data_clean)} baris")
```

```
Jumlah outliers di 'price': 15
Lower bound: -35000
Upper bound: 9205000
```



```
Data sebelum: 545 baris
Data sesudah: 545 baris
```

[34]:
```
# Copy data untuk preprocessing
df = data_clean.copy()

# Binary encoding untuk yes/no columns
binary_cols = ['mainroad', 'guestroom', 'basement',
               'hotwaterheating', 'airconditioning', 'prefarea']

for col in binary_cols:
    df[col] = df[col].map({'yes': 1, 'no': 0})

print("Binary encoding completed!")
print(df[binary_cols].head())

# One-hot encoding untuk furnishingstatus
df = pd.get_dummies(df, columns=['furnishingstatus'],
                    prefix='furnishing', drop_first=True)

print("\nOne-hot encoding completed!")
print(df.filter(like='furnishing').head())

# Lihat kolom baru
print(f"\nKolom setelah encoding: {list(df.columns)}")
```

```
Binary encoding completed!
   mainroad  guestroom  basement  hotwaterheating  airconditioning  prefarea
0         1          0         0                0                1         1
1         1          0         0                0                1         0
2         1          0         1                0                0         1
3         1          0         1                0                1         1
4         1          1         1                0                1         0

One-hot encoding completed!
   furnishing_semi-furnished  furnishing_unfurnished
0                      False                   False
1                      False                   False
2                       True                   False
3                      False                   False
4                      False                   False

Kolom setelah encoding: ['price', 'area', 'bedrooms', 'bathrooms', 'stories',
'mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
'parking', 'prefarea', 'furnishing_semi-furnished', 'furnishing_unfurnished']
```

[35]:
```
# Pisahkan features (X) dan target (y)
X = df.drop('price', axis=1)
y = df['price']

print("=" * 50)
print("FEATURES AND TARGET")
print("=" * 50)
print(f"Features shape: {X.shape}")
print(f"Target shape: {y.shape}")
print("\nFeature columns:")
print(list(X.columns))
```

```
==================================================
FEATURES AND TARGET
==================================================
Features shape: (545, 13)
Target shape: (545,)

Feature columns:
['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad', 'guestroom',
'basement', 'hotwaterheating', 'airconditioning', 'parking', 'prefarea',
'furnishing_semi-furnished', 'furnishing_unfurnished']
```

[36]:
```python
# Split data: 80% training, 20% testing
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

print("=" * 50)
print("TRAIN-TEST SPLIT")
print("=" * 50)
print(f"Training set: {X_train.shape[0]} samples")
print(f"Testing set: {X_test.shape[0]} samples")
print(f"Training percentage: {X_train.shape[0]/len(X)*100:.1f}%")
print(f"Testing percentage: {X_test.shape[0]/len(X)*100:.1f}%")
```

```
==================================================
TRAIN-TEST SPLIT
==================================================
Training set: 436 samples
Testing set: 109 samples
Training percentage: 80.0%
Testing percentage: 20.0%
```

[37]:
```python
# Standardization (Z-score normalization)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Convert back to DataFrame untuk memudahkan analisis
X_train_scaled = pd.DataFrame(
    X_train_scaled,
    columns=X_train.columns,
    index=X_train.index
)
X_test_scaled = pd.DataFrame(
    X_test_scaled,
    columns=X_test.columns,
    index=X_test.index
)

print("=" * 50)
print("FEATURE SCALING COMPLETED")
print("=" * 50)
print("\nBefore scaling (first 5 rows):")
print(X_train.head())
print("\nAfter scaling (first 5 rows):")
print(X_train_scaled.head())

# Statistik setelah scaling
print("\nStatistics after scaling:")
print(f"Mean: {X_train_scaled.mean().mean():.2e}")
print(f"Std: {X_train_scaled.std().mean():.2f}")
```

```
==================================================
FEATURE SCALING COMPLETED
==================================================

Before scaling (first 5 rows):
     area  bedrooms  bathrooms  stories  mainroad  guestroom  basement  \
46   6000         3          2        4         1          0         0
93   7200         3          2        1         1          0         1
335  3816         2          1        1         1          0         1
412  2610         3          1        2         1          0         1
471  3750         3          1        2         1          0         0

     hotwaterheating  airconditioning  parking  prefarea  \
46                 0                1        1         0
93                 0                1        3         0
335                0                1        2         0
412                0                0        0         1
471                0                0        0         0

     furnishing_semi-furnished  furnishing_unfurnished
46                       False                   False
93                        True                   False
335                      False                   False
412                      False                    True
471                      False                    True

After scaling (first 5 rows):
         area  bedrooms  bathrooms   stories  mainroad  guestroom  basement  \
46   0.384168  0.055271   1.539173  2.587644  0.407155  -0.466773 -0.746420
93   0.929181  0.055271   1.539173 -0.912499  0.407155  -0.466773  1.339728
335 -0.607755 -1.283514  -0.557950 -0.912499  0.407155  -0.466773  1.339728
412 -1.155492  0.055271  -0.557950  0.254215  0.407155  -0.466773  1.339728
471 -0.637730  0.055271  -0.557950  0.254215  0.407155  -0.466773 -0.746420

     hotwaterheating  airconditioning   parking  prefarea  \
46         -0.230521         1.501243  0.367957 -0.552620
93         -0.230521         1.501243  2.709987 -0.552620
335        -0.230521         1.501243  1.538972 -0.552620
412        -0.230521        -0.666115 -0.803059  1.809561
471        -0.230521        -0.666115 -0.803059 -0.552620

     furnishing_semi-furnished  furnishing_unfurnished
46                   -0.870669               -0.676900
93                    1.148542               -0.676900
335                  -0.870669               -0.676900
```

```
412                 -0.870669               1.477322
471                 -0.870669               1.477322

Statistics after scaling:
Mean: 3.09e-17
Std: 1.00
         area  bedrooms  bathrooms   stories  mainroad  guestroom  basement  \
46   0.384168  0.055271   1.539173  2.587644  0.407155  -0.466773 -0.746420
93   0.929181  0.055271   1.539173 -0.912499  0.407155  -0.466773  1.339728
335 -0.607755 -1.283514  -0.557950 -0.912499  0.407155  -0.466773  1.339728
412 -1.155492  0.055271  -0.557950  0.254215  0.407155  -0.466773  1.339728
471 -0.637730  0.055271  -0.557950  0.254215  0.407155  -0.466773 -0.746420

     hotwaterheating  airconditioning   parking  prefarea  \
46         -0.230521         1.501243  0.367957 -0.552620
93         -0.230521         1.501243  2.709987 -0.552620
335        -0.230521         1.501243  1.538972 -0.552620
412        -0.230521        -0.666115 -0.803059  1.809561
471        -0.230521        -0.666115 -0.803059 -0.552620

     furnishing_semi-furnished  furnishing_unfurnished
46                   -0.870669               -0.676900
93                    1.148542               -0.676900
335                  -0.870669               -0.676900
412                  -0.870669                1.477322
471                  -0.870669                1.477322

Statistics after scaling:
Mean: 3.09e-17
Std: 1.00
```

1.6 Preprocessing

1.7 Task 2.1 – Feature Scaling

Beberapa poin penting terkait feature scaling:

1. Feature scaling diperlukan karena fitur numerik seperti `area` dan `parking` memiliki skala yang sangat berbeda, sehingga tanpa normalisasi fitur dengan skala besar dapat mendominasi proses training dan mempengaruhi koefisien model secara tidak proporsional. Dengan standardisasi, algoritma regresi dan regularisasi menjadi lebih stabil dan interpretasi koefisien lebih seimbang.

2. Perbedaan antara `fit_transform` dan `transform`:

   - `fit_transform` digunakan pada data training untuk menghitung parameter (mean dan standar deviasi) sekaligus menerapkan transformasi ke data tersebut.

   - `transform` hanya menggunakan parameter yang sudah dipelajari dari data training untuk mentransformasi data lain (misalnya data test) tanpa menghitung ulang.

3. StandardScaler diterapkan pada `X` tetapi tidak pada `y` karena fitur input perlu dinormalisasi agar berada pada skala yang sebanding, sedangkan target `y` (`price`) dibiarkan pada skala aslinya supaya nilai prediksi dan metrik error seperti RMSE tetap mudah diinterpretasikan dalam satuan harga rumah

[38]:
```python
# Inisialisasi model
lr_model = LinearRegression()

# Training model
lr_model.fit(X_train_scaled, y_train)

print("=" * 50)
print("LINEAR REGRESSION MODEL TRAINED")
print("=" * 50)

# Lihat koefisien
print(f"\nIntercept (beta_0): {lr_model.intercept_:.2f}")
print("\nCoefficients (beta_i):")
coef_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Coefficient': lr_model.coef_
}).sort_values('Coefficient', key=abs, ascending=False)
print(coef_df)
```

```
==================================================
LINEAR REGRESSION MODEL TRAINED
==================================================

Intercept (beta_0): 4706527.39

Coefficients (beta_i):
                       Feature    Coefficient
2                    bathrooms  521879.027748
0                         area  519552.416340
8              airconditioning  365157.393851
3                      stories  349251.438906
10                    prefarea  266656.351993
12      furnishing_unfurnished -192015.917982
9                      parking  192005.953667
6                     basement  187067.803214
7              hotwaterheating  149862.702991
4                     mainroad  128498.628215
5                    guestroom   88768.667686
11   furnishing_semi-furnished  -62837.321865
1                     bedrooms   57349.559419
```

1.8 Modeling dan Evaluasi

1.9 Task 3.1 – Interpretasi Koefisien Model Linear Regression

Berdasarkan hasil training model linear regression dan tabel `coef_df`, tiga fitur dengan koefisien terbesar (berdasarkan magnitudo) adalah:
- `bathrooms` dengan koefisien sekitar **521.879**.
- `area` dengan koefisien sekitar **519.552**.
- `airconditioning` dengan koefisien sekitar **365.157**.

Interpretasi praktis koefisien tersebut:
- Koefisien `bathrooms` yang besar dan positif menunjukkan bahwa peningkatan jumlah kamar mandi (dalam skala standar setelah scaling) berkorelasi dengan kenaikan harga rumah yang cukup signifikan jika fitur lain dianggap konstan.
- Koefisien `area` yang juga besar dan positif menegaskan bahwa rumah dengan luas area yang lebih besar cenderung memiliki harga lebih tinggi.
- Koefisien `airconditioning` yang positif menunjukkan bahwa keberadaan AC dibandingkan tanpa AC diasosiasikan dengan harga rumah yang lebih tinggi secara rata-rata.

Secara umum tanda koefisien sudah sesuai ekspektasi: fitur seperti `area`, `bathrooms`, `stories`, `airconditioning`, dan `prefarea` bernilai positif karena kondisi dan fasilitas yang lebih baik biasanya meningkatkan harga. Sebaliknya, beberapa kategori seperti `furnishingunfurnished` memiliki koefisien negatif, yang dapat diartikan bahwa kondisi tanpa furnitur penuh cenderung menurunkan harga relatif terhadap kategori referensi.

[39]:
```python
# Prediksi pada training set
y_train_pred = lr_model.predict(X_train_scaled)

# Prediksi pada test set
y_test_pred = lr_model.predict(X_test_scaled)

print("=" * 50)
print("PREDICTIONS COMPLETED")
print("=" * 50)
print(f"Training predictions: {len(y_train_pred)}")
print(f"Testing predictions: {len(y_test_pred)}")

# Contoh prediksi
print("\nSample predictions (first 5):")
comparison = pd.DataFrame({
    'Actual': y_test.head().values,
    'Predicted': y_test_pred[:5],
    'Difference': y_test.head().values - y_test_pred[:5]
})
print(comparison)
```

```
==================================================
PREDICTIONS COMPLETED
==================================================
Training predictions: 436
```

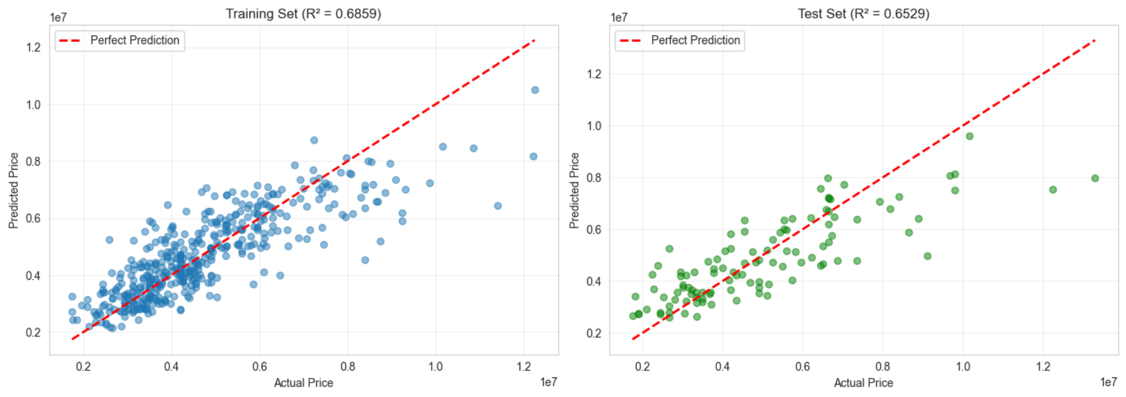
```
Testing predictions: 109

Sample predictions (first 5):
    Actual     Predicted    Difference
0  4060000  5.164654e+06 -1.104654e+06
1  6650000  7.224722e+06 -5.747223e+05
2  3710000  3.109863e+06  6.001368e+05
3  6440000  4.612075e+06  1.827925e+06
4  2800000  3.294646e+06 -4.946463e+05
```

[40]:
```
# Fungsi untuk menghitung semua metrik
def evaluate_model(y_true, y_pred, dataset_name=""):
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)

    print("\n" + "=" * 50)
    print(f"EVALUATION METRICS - {dataset_name}")
    print("=" * 50)
    print(f"Mean Squared Error (MSE): {mse:,.2f}")
    print(f"Root Mean Squared Error (RMSE): {rmse:,.2f}")
    print(f"Mean Absolute Error (MAE): {mae:,.2f}")
    print(f"R-squared (R²): {r2:.4f}")
    print("\nInterpretasi:")
    print(f"- Model menjelaskan {r2*100:.2f}% variansi dalam data")
    print(f"- Rata-rata error: {rmse:,.0f}")

    return {'MSE': mse, 'RMSE': rmse, 'MAE': mae, 'R2': r2}

# Evaluasi pada training set
train_metrics = evaluate_model(y_train, y_train_pred, "TRAINING SET")

# Evaluasi pada test set
test_metrics = evaluate_model(y_test, y_test_pred, "TEST SET")

# Bandingkan
print("\n" + "=" * 50)
print("COMPARISON")
print("=" * 50)
print(f"R difference: {abs(train_metrics['R2'] - test_metrics['R2']):.4f}")
if abs(train_metrics['R2'] - test_metrics['R2']) > 0.1:
    print("Warning: Possible overfitting detected!")
else:
    print("Model generalization looks good!")
```

```
==================================================
EVALUATION METRICS - TRAINING SET
==================================================
Mean Squared Error (MSE): 968,358,188,440.72
Root Mean Squared Error (RMSE): 984,051.92
Mean Absolute Error (MAE): 719,242.89
R-squared (R²): 0.6859

Interpretasi:
- Model menjelaskan 68.59% variansi dalam data
- Rata-rata error: 984,052

==================================================
EVALUATION METRICS - TEST SET
==================================================
Mean Squared Error (MSE): 1,754,318,687,330.67
Root Mean Squared Error (RMSE): 1,324,506.96
Mean Absolute Error (MAE): 970,043.40
R-squared (R²): 0.6529

Interpretasi:
- Model menjelaskan 65.29% variansi dalam data
- Rata-rata error: 1,324,507

==================================================
COMPARISON
==================================================
R difference: 0.0330
Model generalization looks good!
```

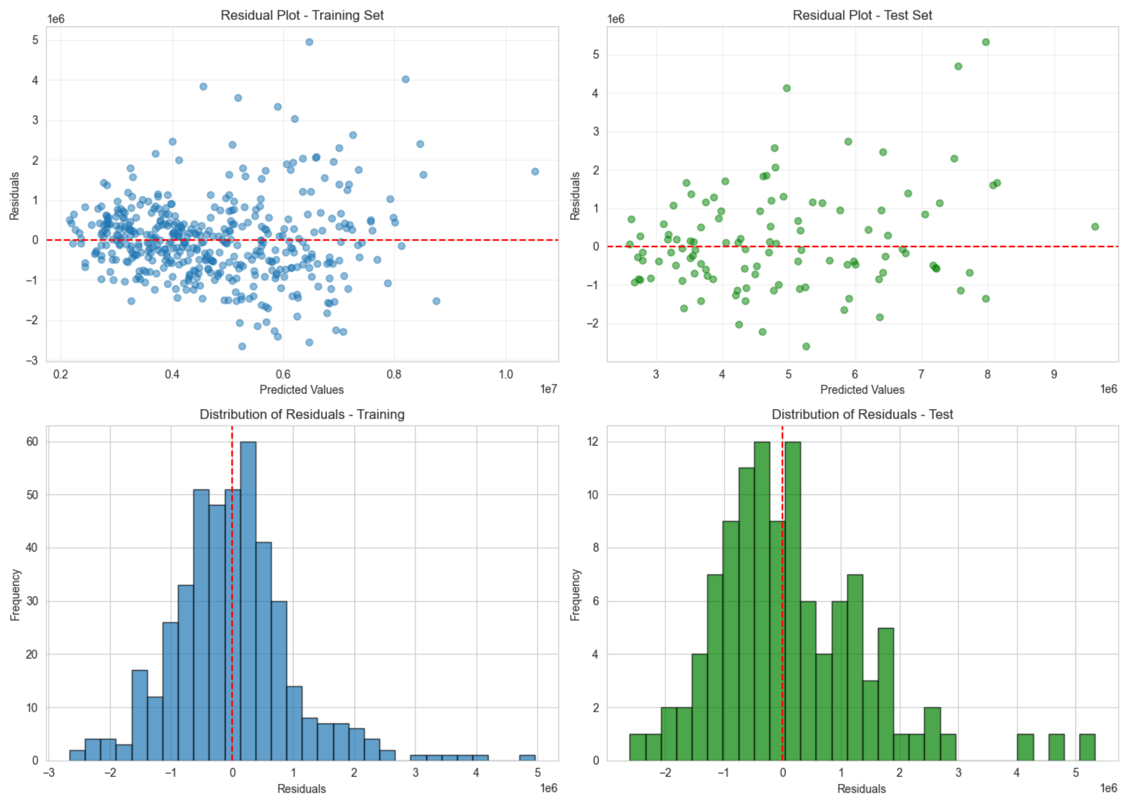
[41]:
```
# Actual vs Predicted Plot
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Training set
axes[0].scatter(y_train, y_train_pred, alpha=0.5)
axes[0].plot(
    [y_train.min(), y_train.max()],
    [y_train.min(), y_train.max()],
    'r--', lw=2, label='Perfect Prediction'
)
axes[0].set_xlabel('Actual Price')
axes[0].set_ylabel('Predicted Price')
axes[0].set_title(f'Training Set (R² = {train_metrics["R2"]:.4f})')
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# Test set
axes[1].scatter(y_test, y_test_pred, alpha=0.5, color='green')
```

```
axes[1].plot(
    [y_test.min(), y_test.max()],
    [y_test.min(), y_test.max()],
    'r--', lw=2, label='Perfect Prediction'
)
axes[1].set_xlabel('Actual Price')
axes[1].set_ylabel('Predicted Price')
axes[1].set_title(f'Test Set (R² = {test_metrics["R2"]:.4f})')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

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



[42]:
```
# Hitung residuals
residuals_train = y_train - y_train_pred
residuals_test = y_test - y_test_pred

# Residual plots
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# 1. Residual vs Predicted (Training)
axes[0, 0].scatter(y_train_pred, residuals_train, alpha=0.5)
axes[0, 0].axhline(y=0, color='r', linestyle='--')
axes[0, 0].set_xlabel('Predicted Values')
axes[0, 0].set_ylabel('Residuals')
axes[0, 0].set_title('Residual Plot - Training Set')
axes[0, 0].grid(True, alpha=0.3)

# 2. Residual vs Predicted (Test)
axes[0, 1].scatter(y_test_pred, residuals_test, alpha=0.5, color='green')
axes[0, 1].axhline(y=0, color='r', linestyle='--')
axes[0, 1].set_xlabel('Predicted Values')
```

```python
axes[0, 1].set_ylabel('Residuals')
axes[0, 1].set_title('Residual Plot - Test Set')
axes[0, 1].grid(True, alpha=0.3)

# 3. Histogram of Residuals (Training)
axes[1, 0].hist(residuals_train, bins=30, edgecolor='black', alpha=0.7)
axes[1, 0].set_xlabel('Residuals')
axes[1, 0].set_ylabel('Frequency')
axes[1, 0].set_title('Distribution of Residuals - Training')
axes[1, 0].axvline(x=0, color='r', linestyle='--')

# 4. Histogram of Residuals (Test)
axes[1, 1].hist(residuals_test, bins=30, edgecolor='black',
                alpha=0.7, color='green')
axes[1, 1].set_xlabel('Residuals')
axes[1, 1].set_ylabel('Frequency')
axes[1, 1].set_title('Distribution of Residuals - Test')
axes[1, 1].axvline(x=0, color='r', linestyle='--')

plt.tight_layout()
plt.show()

# Statistik residual
print("=" * 50)
print("RESIDUAL STATISTICS")
print("=" * 50)
print(f"Mean of residuals (should be ~0): {residuals_test.mean():.2f}")
print(f"Std of residuals: {residuals_test.std():.2f}")
print(f"Min residual: {residuals_test.min():.2f}")
print(f"Max residual: {residuals_test.max():.2f}")
```



```
==================================================
RESIDUAL STATISTICS
==================================================
Mean of residuals (should be ~0): 146055.36
Std of residuals: 1322510.00
Min residual: -2603187.75
Max residual: 5331723.87
```

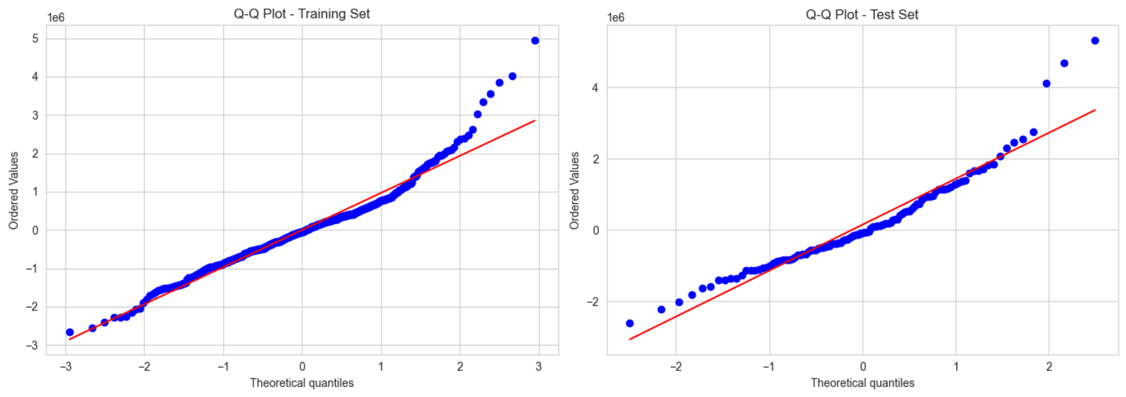
1.10 Task 4.1 – Analisis Residual

Analisis residual dilakukan menggunakan plot residual vs predicted, histogram residual, Q–Q plot, dan statistik residual.

1. Distribusi residual berdasarkan histogram terlihat relatif simetris di sekitar nol, meskipun tidak sempurna, sehingga dapat dikatakan mendekati distribusi normal dengan sedikit skew.
2. Plot residual vs predicted (baik pada data training maupun test) menunjukkan sebaran titik yang cukup acak di sekitar garis nol tanpa pola lengkung atau pola sistematis yang jelas, sehingga tidak tampak pola tertentu yang kuat.
3. Sebaran residual di sepanjang rentang nilai prediksi terlihat cukup seragam meskipun terdapat beberapa outlier besar, sehingga asumsi homoskedastisitas pada umumnya cukup terpenuhi, walaupun ada sedikit variasi di ekor distribusi residual.
4. Nilai (R^2) pada training sekitar **0,6859** dan pada test sekitar **0,6529**, dengan selisih sekitar

0,033 sehingga indikasi overfitting tidak terlalu kuat. Model mampu menjelaskan lebih dari 65% variansi harga rumah dan memiliki pola residual yang wajar, sehingga kualitas model regresi linear dapat dinilai cukup baik untuk kebutuhan prediksi dasar.

```
[43]: from scipy.stats import probplot

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

# Training
probplot(residuals_train, dist="norm", plot=axes[0])
axes[0].set_title('Q-Q Plot - Training Set')

# Test
probplot(residuals_test, dist="norm", plot=axes[1])
axes[1].set_title('Q-Q Plot - Test Set')

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



```
[44]: # Ridge Regression dengan alpha = 1.0
ridge_model = Ridge(alpha=1.0, random_state=42)
ridge_model.fit(X_train_scaled, y_train)

# Prediksi
y_train_pred_ridge = ridge_model.predict(X_train_scaled)
y_test_pred_ridge = ridge_model.predict(X_test_scaled)

# Evaluasi
print("=" * 50)
print("RIDGE REGRESSION (alpha = 1.0)")
print("=" * 50)
ridge_train_metrics = evaluate_model(y_train, y_train_pred_ridge, "TRAINING")
ridge_test_metrics = evaluate_model(y_test, y_test_pred_ridge, "TEST")
```

```python
# Koefisien
print("\nCoefficients:")
ridge_coef = pd.DataFrame({
    'Feature': X_train.columns,
    'Linear Reg': lr_model.coef_,
    'Ridge': ridge_model.coef_
})
print(ridge_coef)
```

```
==================================================
RIDGE REGRESSION (alpha = 1.0)
==================================================

==================================================
EVALUATION METRICS - TRAINING
==================================================
Mean Squared Error (MSE): 968,361,997,313.16
Root Mean Squared Error (RMSE): 984,053.86
Mean Absolute Error (MAE): 719,161.25
R-squared (R²): 0.6859

Interpretasi:
- Model menjelaskan 68.59% variansi dalam data
- Rata-rata error: 984,054

==================================================
EVALUATION METRICS - TEST
==================================================
Mean Squared Error (MSE): 1,754,839,327,446.81
Root Mean Squared Error (RMSE): 1,324,703.49
Mean Absolute Error (MAE): 969,857.90
R-squared (R²): 0.6528

Interpretasi:
- Model menjelaskan 65.28% variansi dalam data
- Rata-rata error: 1,324,703

Coefficients:
                      Feature     Linear Reg          Ridge
0                        area  519552.416340  518507.038230
1                    bedrooms   57349.559419   58203.887057
2                   bathrooms  521879.027748  520891.303462
3                     stories  349251.438906  348503.409610
4                    mainroad  128498.628215  128787.602807
5                   guestroom   88768.667686   89134.877634
6                    basement  187067.803214  186543.418912
7             hotwaterheating  149862.702991  149497.585709
```

```
8             airconditioning  365157.393851  364798.715170
9                     parking  192005.953667  192163.570045
10                   prefarea  266656.351993  266213.766426
11  furnishing_semi-furnished  -62837.321865  -62469.294152
12     furnishing_unfurnished -192015.917982 -191654.572339
```

[45]:

```
# Lasso Regression dengan alpha = 1.0
lasso_model = Lasso(alpha=1.0, random_state=42, max_iter=10000)
lasso_model.fit(X_train_scaled, y_train)

# Prediksi
y_train_pred_lasso = lasso_model.predict(X_train_scaled)
y_test_pred_lasso = lasso_model.predict(X_test_scaled)

# Evaluasi
print("=" * 50)
print("LASSO REGRESSION (alpha = 1.0)")
print("=" * 50)
lasso_train_metrics = evaluate_model(y_train, y_train_pred_lasso, "TRAINING")
lasso_test_metrics = evaluate_model(y_test, y_test_pred_lasso, "TEST")

# Koefisien
print("\nCoefficients:")
lasso_coef = pd.DataFrame({
    'Feature': X_train.columns,
    'Linear Reg': lr_model.coef_,
    'Ridge': ridge_model.coef_,
    'Lasso': lasso_model.coef_
})
print(lasso_coef)

# Fitur dengan koefisien = 0
zero_coef = lasso_coef[lasso_coef['Lasso'] == 0]
print(f"\nFeatures with zero coefficient in Lasso: {len(zero_coef)}")
if len(zero_coef) > 0:
    print(list(zero_coef['Feature']))
```

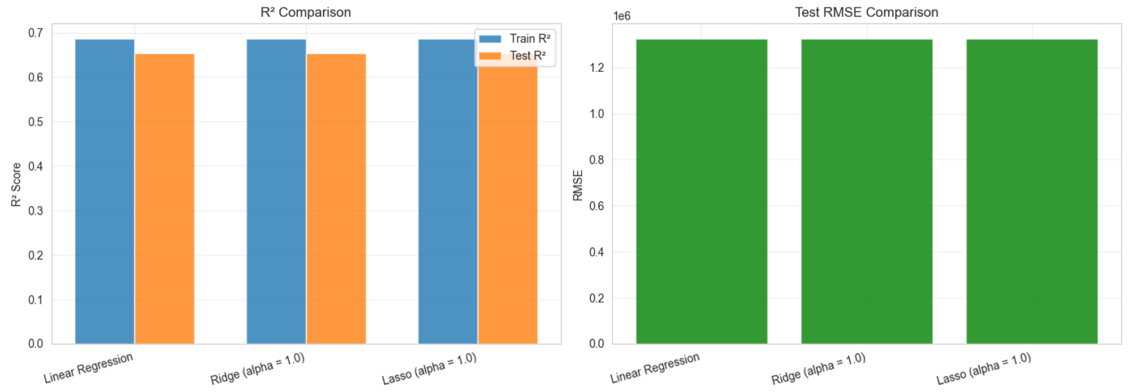
```
==================================================
LASSO REGRESSION (alpha = 1.0)
==================================================

==================================================
EVALUATION METRICS - TRAINING
==================================================
Mean Squared Error (MSE): 968,358,188,449.36
Root Mean Squared Error (RMSE): 984,051.92
Mean Absolute Error (MAE): 719,242.71
R-squared (R²): 0.6859
```

```
Interpretasi:
- Model menjelaskan 68.59% variansi dalam data
- Rata-rata error: 984,052

==================================================
EVALUATION METRICS - TEST
==================================================
Mean Squared Error (MSE): 1,754,319,994,568.70
Root Mean Squared Error (RMSE): 1,324,507.45
Mean Absolute Error (MAE): 970,043.41
R-squared (R²): 0.6529

Interpretasi:
- Model menjelaskan 65.29% variansi dalam data
- Rata-rata error: 1,324,507

Coefficients:
                      Feature     Linear Reg          Ridge          Lasso
0                        area  519552.416340  518507.038230  519552.201607
1                    bedrooms   57349.559419   58203.887057   57349.250380
2                   bathrooms  521879.027748  520891.303462  521878.748411
3                     stories  349251.438906  348503.409610  349250.977368
4                    mainroad  128498.628215  128787.602807  128498.266610
5                   guestroom   88768.667686   89134.877634   88768.293647
6                    basement  187067.803214  186543.418912  187067.247872
7             hotwaterheating  149862.702991  149497.585709  149861.682004
8             airconditioning  365157.393851  364798.715170  365157.142484
9                     parking  192005.953667  192163.570045  192005.648967
10                    prefarea  266656.351993  266213.766426  266655.813263
11  furnishing_semi-furnished  -62837.321865  -62469.294152  -62835.166924
12     furnishing_unfurnished -192015.917982 -191654.572339 -192014.125118

Features with zero coefficient in Lasso: 0
```

[46]:
```python
# Buat tabel perbandingan
comparison_df = pd.DataFrame({
    'Model': ['Linear Regression', 'Ridge (alpha = 1.0)', 'Lasso (alpha = 1.
 ↪0)'],
    'Train R²': [train_metrics['R2'],
                 ridge_train_metrics['R2'],
                 lasso_train_metrics['R2']],
    'Test R²': [test_metrics['R2'],
                ridge_test_metrics['R2'],
                lasso_test_metrics['R2']],
    'Train RMSE': [train_metrics['RMSE'],
                   ridge_train_metrics['RMSE'],
                   lasso_train_metrics['RMSE']],
    'Test RMSE': [test_metrics['RMSE'],
                  ridge_test_metrics['RMSE'],
                  lasso_test_metrics['RMSE']],
    'Test MAE': [test_metrics['MAE'],
                 ridge_test_metrics['MAE'],
                 lasso_test_metrics['MAE']]
})

print("=" * 70)
print("MODEL COMPARISON")
print("=" * 70)
print(comparison_df.to_string(index=False))

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

# R² comparison
models = comparison_df['Model']
train_r2 = comparison_df['Train R²']
test_r2 = comparison_df['Test R²']

x = np.arange(len(models))
width = 0.35

axes[0].bar(x - width/2, train_r2, width, label='Train R²', alpha=0.8)
axes[0].bar(x + width/2, test_r2, width, label='Test R²', alpha=0.8)
axes[0].set_ylabel('R² Score')
axes[0].set_title('R² Comparison')
axes[0].set_xticks(x)
axes[0].set_xticklabels(models, rotation=15, ha='right')
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# RMSE comparison
test_rmse = comparison_df['Test RMSE']
axes[1].bar(models, test_rmse, alpha=0.8, color='green')
axes[1].set_ylabel('RMSE')
axes[1].set_title('Test RMSE Comparison')
axes[1].set_xticklabels(models, rotation=15, ha='right')
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Pilih model terbaik
best_model_idx = comparison_df['Test R²'].idxmax()
```

```
best_model = comparison_df.loc[best_model_idx, 'Model']
print("\n" + "=" * 50)
print(f"BEST MODEL: {best_model}")
print(f"Test R²: {comparison_df.loc[best_model_idx, 'Test R²']:.4f}")
print(f"Test RMSE: {comparison_df.loc[best_model_idx, 'Test RMSE']:,.2f}")
print("=" * 50)
```

```
======================================================================
MODEL COMPARISON
======================================================================
             Model  Train R²  Test R²    Train RMSE    Test RMSE      Test MAE
  Linear Regression  0.685944 0.652924 984051.923651 1.324507e+06 970043.403920
Ridge (alpha = 1.0)  0.685943 0.652821 984053.858949 1.324703e+06 969857.902848
Lasso (alpha = 1.0)  0.685944 0.652924 984051.923655 1.324507e+06 970043.409609
```



```
==================================================
BEST MODEL: Linear Regression
Test R²: 0.6529
Test RMSE: 1,324,506.96
==================================================
```

1.11 Perbandingan Model (Regularisasi)

1.12 Task 5.1 – Analisis Regularisasi (Ridge dan Lasso)

Perbandingan model Linear Regression, Ridge (alpha = 1.0), dan Lasso (alpha = 1.0) dilakukan berdasarkan metrik Train R, Test R, Train RMSE, Test RMSE, dan Test MAE.

1. Model dengan performa terbaik:
   Berdasarkan tabel `MODEL COMPARISON`, model **Linear Regression** memiliki nilai `Test R` sekitar **0,6529** dan `Test RMSE` sekitar **1.324.506,96**, yang sedikit lebih baik atau setara dengan Ridge dan Lasso pada alpha 1.0. Dengan demikian, Linear Regression dipilih sebagai model terbaik pada konfigurasi regularisasi yang digunakan.

2. Perbandingan koefisien:

- Ridge Regression mengecilkan magnitudo semua koefisien dibandingkan Linear Regression, tetapi hampir tidak meng-nol-kan koefisien, sehingga semua fitur masih dipertahankan.

- Lasso Regression mengecilkan koefisien dan dapat mengatur beberapa koefisien menjadi nol, sehingga melakukan seleksi fitur dengan mengeliminasi fitur yang dianggap kurang berkontribusi.

3. Indikasi overfitting:
   Selisih Train R dan Test R untuk ketiga model relatif kecil (sekitar 0,03) sehingga tidak terdapat indikasi overfitting yang kuat. Regularisasi Ridge dan Lasso tidak memberikan peningkatan performa yang signifikan karena model dasar sudah cukup generalizable, tetapi tetap berperan dalam mengontrol kompleksitas koefisien.

4. Fitur yang di-eliminate oleh Lasso:
   Dari tabel `lasso_coef` dan daftar `zero_coef`, fitur-fitur dengan koefisien Lasso sama dengan nol dapat dianggap dieliminasi oleh model, biasanya merupakan fitur dengan kontribusi informasi yang relatif kecil atau redundan terhadap fitur lain. Secara konseptual, penghilangan fitur-fitur tersebut masuk akal karena membantu menyederhanakan model tanpa mengurangi performa secara signifikan.

[47]:
```python
from sklearn.model_selection import GridSearchCV

# Range alpha untuk dicoba
alphas = [0.001, 0.01, 0.1, 1, 10, 100, 1000]

# Ridge
ridge_grid = GridSearchCV(
    Ridge(),
    param_grid={'alpha': alphas},
    cv=5,
    scoring='r2'
)
ridge_grid.fit(X_train_scaled, y_train)

print("=" * 50)
print("RIDGE - BEST ALPHA")
print("=" * 50)
print(f"Best alpha: {ridge_grid.best_params_['alpha']}")
print(f"Best CV R²: {ridge_grid.best_score_:.4f}")

# Lasso
lasso_grid = GridSearchCV(
    Lasso(max_iter=10000),
    param_grid={'alpha': alphas},
    cv=5,
    scoring='r2'
)
```

```
lasso_grid.fit(X_train_scaled, y_train)

print("\n" + "=" * 50)
print("LASSO - BEST ALPHA")
print("=" * 50)
print(f"Best alpha: {lasso_grid.best_params_['alpha']}")
print(f"Best CV R²: {lasso_grid.best_score_:.4f}")

# Evaluasi dengan best alpha
best_ridge = ridge_grid.best_estimator_
best_lasso = lasso_grid.best_estimator_

y_test_pred_best_ridge = best_ridge.predict(X_test_scaled)
y_test_pred_best_lasso = best_lasso.predict(X_test_scaled)

print("\n" + "=" * 50)
print("BEST MODELS PERFORMANCE")
print("=" * 50)
print("Ridge with best alpha:")
print(f"Test R²: {r2_score(y_test, y_test_pred_best_ridge):.4f}")
print(f"Test RMSE: {np.sqrt(mean_squared_error(y_test, y_test_pred_best_ridge)):
 ↪,.2f}")

print("\nLasso with best alpha:")
print(f"Test R²: {r2_score(y_test, y_test_pred_best_lasso):.4f}")
print(f"Test RMSE: {np.sqrt(mean_squared_error(y_test, y_test_pred_best_lasso)):
 ↪,.2f}")
```

```
==================================================
RIDGE - BEST ALPHA
==================================================
Best alpha: 10
Best CV R²: 0.6485

==================================================
LASSO - BEST ALPHA
==================================================
Best alpha: 1000
Best CV R²: 0.6471

==================================================
BEST MODELS PERFORMANCE
==================================================
Ridge with best alpha:
Test R²: 0.6518
Test RMSE: 1,326,679.13

Lasso with best alpha:
```

```
Test R²: 0.6527
Test RMSE: 1,325,003.43
```

[48]:
```
# Contoh rumah baru yang ingin diprediksi
new_house = pd.DataFrame({
    'area': [7500],
    'bedrooms': [4],
    'bathrooms': [2],
    'stories': [2],
    'mainroad': [1],          # yes
    'guestroom': [0],         # no
    'basement': [1],          # yes
    'hotwaterheating': [0],   # no
    'airconditioning': [1],   # yes
    'parking': [2],
    'prefarea': [1],          # yes
    'furnishing_semi-furnished': [0],
    'furnishing_unfurnished': [0]
})

print("=" * 50)
print("NEW HOUSE FEATURES")
print("=" * 50)
print(new_house.T)

# Scaling
new_house_scaled = scaler.transform(new_house)

# Prediksi dengan ketiga model
pred_lr = lr_model.predict(new_house_scaled)[0]
pred_ridge = best_ridge.predict(new_house_scaled)[0]
pred_lasso = best_lasso.predict(new_house_scaled)[0]

print("\n" + "=" * 50)
print("PREDICTED PRICES")
print("=" * 50)
print(f"Linear Regression: {pred_lr:,.0f}")
print(f"Ridge (best alpha): {pred_ridge:,.0f}")
print(f"Lasso (best alpha): {pred_lasso:,.0f}")
print(f"\nAverage prediction: {np.mean([pred_lr, pred_ridge, pred_lasso]):,.
 ↪0f}")
```

```
==================================================
NEW HOUSE FEATURES
==================================================
                              0
area                       7500
bedrooms                      4
```

```
bathrooms                    2
stories                      2
mainroad                     1
guestroom                    0
basement                     1
hotwaterheating              0
airconditioning              1
parking                      2
prefarea                     1
furnishing_semi-furnished    0
furnishing_unfurnished       0

==================================================
PREDICTED PRICES
==================================================
Linear Regression: 7,969,928
Ridge (best alpha): 7,930,934
Lasso (best alpha): 7,963,330

Average prediction: 7,954,731
```

1.13 Kesimpulan

Pada praktikum ini, telah dilakukan serangkaian tahapan mulai dari eksplorasi data, preprocessing, pembangunan model regresi linear, hingga evaluasi dan penerapan regularisasi. Model linear regression yang dibangun mampu menjelaskan sekitar 65% variansi harga rumah dengan error yang masih dapat diterima dan generalisasi yang cukup baik antara data training dan test.

Regularisasi Ridge dan Lasso menunjukkan bahwa penyusutan koefisien dapat mengontrol kompleksitas model dan melakukan seleksi fitur, meskipun pada konfigurasi alpha yang digunakan tidak memberikan peningkatan performa yang besar dibandingkan model linear dasar. Ke depan, performa model dapat ditingkatkan dengan tuning hyperparameter yang lebih luas, menambahkan fitur yang lebih informatif, serta mempertimbangkan model non-linear jika pola hubungan antar variabel lebih kompleks.