Proyek UTS PMDPM Gasal 2023/2024

Nama Anggota Kelompok:

- Teofilos Mas Krisna Dewa - 220711838
- Marcello Aaron K - 220711844
- Rizky Ardiansyah Ramadhan - 220711861
- Reinaldy Restu Aji - 220711877

Inisialisasi

- Import library yang dibutuhkan

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler,
StandardScaler
from sklearn.compose import make_column_transformer
from sklearn.feature_selection import SelectKBest, SelectPercentile,
RFE
from sklearn.model_selection import GridSearchCV, train_test_split,
StratifiedKFold, KFold
from sklearn.svm import SVC, SVR
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.linear_model import LogisticRegression, Ridge, Lasso
from imblearn.over_sampling import SMOTE
from collections import Counter
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix,
mean_squared_error, mean_absolute_error, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
```

Data Loading

- Proses data loading (boleh dengan file upload atau dengan mount drive jika menggunakan Google Colab)

```
from google.colab import drive
drive.mount('/content/drive')
properti = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Dataset
Property/Dataset UTS_Gasal 2425.csv")
properti.head(10000)
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

{"summary":"{\n \"name\": \"properti\",\n \"rows\": 10000,\n \"fields\": [\n   {\n   \"column\": \"squaremeters\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 28774,\n   \"min\": 89,\n   \"max\": 99999,\n   \"num_unique_values\": 9483,\n   \"samples\": [\n   2725,\n   98025,\n   4198\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"numberofrooms\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 28,\n   \"min\": 1,\n   \"max\": 100,\n   \"num_unique_values\": 100,\n   \"samples\": [\n   44,\n   95,\n   4\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"hasyard\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"yes\",\n   \"no\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"haspool\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"no\",\n   \"yes\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"floors\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 28,\n   \"min\": 1,\n   \"max\": 100,\n   \"num_unique_values\": 100,\n   \"samples\": [\n   90,\n   24\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"citycode\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 29006,\n   \"min\": 3,\n   \"max\": 99953,\n   \"num_unique_values\": 9509,\n   \"samples\": [\n   64029,\n   82892\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"citypartrange\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 2,\n   \"min\": 1,\n   \"max\": 10,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   9,\n   6\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"numprevowners\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 2,\n   \"min\": 1,\n   \"max\": 10,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   6,\n   4\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"made\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 9,\n   \"min\": 1990,\n   \"max\": 2021,\n   \"num_unique_values\": 32,\n   \"samples\": [\n   2019,\n   1990\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\

```
n    },\n    {\n      \"column\": \"isnewbuilt\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          \"new\",\
n          \"old\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"hasstormprotector\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"no\",\n          \"yes\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"basement\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 2876,\n
\"min\": 0,\n        \"max\": 10000,\n        \"num_unique_values\":
6352,\n        \"samples\": [\n          2607,\n          8571\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"attic\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 2894,\n
\"min\": 1,\n        \"max\": 10000,\n        \"num_unique_values\":
6267,\n        \"samples\": [\n          2275,\n          5732\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"garage\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 262,\n
\"min\": 100,\n        \"max\": 1000,\n        \"num_unique_values\":
901,\n        \"samples\": [\n          429,\n          500\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"hasstorageroom\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"yes\",\n          \"no\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"hasguestroom\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3,\n        \"min\": 0,\n        \"max\": 10,\n
\"num_unique_values\": 11,\n        \"samples\": [\n          3,\n
7\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"price\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 2877424.1099450146,\n        \"min\": 10313.5,\n
\"max\": 10006771.2,\n        \"num_unique_values\": 10000,\n
\"samples\": [\n          1056525.7,\n          737118.2\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"category\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
3,\n        \"samples\": [\n          \"Luxury\",\n
\"Middle\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"properti"}
```

Dalam dataset ini terdiri dari beberapa kolom yaitu:

- squaremeters: Luas properti dalam meter persegi.

- numberofrooms: Jumlah kamar di properti.
- hasyard: Menunjukkan apakah properti memiliki halaman (yard) atau tidak (yes/no).
- haspool: Menunjukkan apakah properti memiliki kolam renang (pool) atau tidak (yes/no).
- floors: Jumlah lantai di properti.
- citycode: Kode kota tempat properti berada.
- citypartrange: Rentang bagian kota.
- numprevowners: Jumlah pemilik sebelumnya.
- made: Tahun pembuatan properti.
- isnewbuilt: Menunjukkan apakah properti baru dibangun atau tidak (new/old).
- hasstormprotector: Menunjukkan apakah properti memiliki pelindung badai (storm protector) atau tidak (yes/no).
- basement: Menunjukkan apakah properti memiliki basement atau tidak.
- attic: Menunjukkan apakah properti memiliki loteng (attic) atau tidak.
- garage: Menunjukkan apakah properti memiliki garasi atau tidak.
- hasstorageroom: Menunjukkan apakah properti memiliki ruang penyimpanan (storage room) atau tidak (yes/no).
- List item
- hasguestroom: Menunjukkan apakah properti memiliki kamar tamu (guest room) atau tidak (yes/no).
- price: Harga properti.
- category: Kategori properti (misalnya, Luxury, Middle, dll.).

Data Cleansing & Encoding

- Bagian berikut berisi proses pembersihan data.
- Periksa apakah terdapat missing value dan data duplikat,
- Ubah data kategorik string menjadi numerik.
- Jika jumlah kelas pada data latih tidak seimbang, kalian dapat menggunakan metode oversampling.
- Untuk **Klasifikasi**, pastikan **Kategori menjadi target** dan **kolom Harga dihapus**.

```
print("#" * 50)
print("Informasi Umum tentang DataFrame:")
print("#" * 50)
properti.info()
print("\n")

print("#" * 50)
print("Missing Values per Column:")
print("#" * 50)
print(properti.isnull().sum())
print("\n")

print("#" * 50)
print("Jumlah Baris Duplikat:")
print("#" * 50)
print(properti.duplicated().sum())
```

```
print("\n")

if properti.duplicated().sum() > 0:
    print("#" * 50)
    print("Baris Duplikat:")
    print("#" * 50)
    print(properti[properti.duplicated()])
else:
    print("#" * 50)
    print("Tidak ada baris duplikat.")
    print("#" * 50)
```

```
##################################################
Informasi Umum tentang DataFrame:
##################################################
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 18 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   squaremeters       10000 non-null  int64
 1   numberofrooms      10000 non-null  int64
 2   hasyard            10000 non-null  object
 3   haspool            10000 non-null  object
 4   floors             10000 non-null  int64
 5   citycode           10000 non-null  int64
 6   citypartrange      10000 non-null  int64
 7   numprevowners      10000 non-null  int64
 8   made               10000 non-null  int64
 9   isnewbuilt         10000 non-null  object
 10  hasstormprotector  10000 non-null  object
 11  basement           10000 non-null  int64
 12  attic              10000 non-null  int64
 13  garage             10000 non-null  int64
 14  hasstorageroom     10000 non-null  object
 15  hasguestroom       10000 non-null  int64
 16  price              10000 non-null  float64
 17  category           10000 non-null  object
dtypes: float64(1), int64(11), object(6)
memory usage: 1.4+ MB


##################################################
Missing Values per Column:
##################################################
squaremeters         0
numberofrooms        0
hasyard              0
haspool              0
floors               0
```

```
citycode              0
citypartrange         0
numprevowners         0
made                  0
isnewbuilt            0
hasstormprotector     0
basement              0
attic                 0
garage                0
hasstorageroom        0
hasguestroom          0
price                 0
category              0
dtype: int64


##################################################
Jumlah Baris Duplikat:
##################################################
0


##################################################
Tidak ada baris duplikat.
##################################################
```

- Semua kolom memiliki nilai non-null, menunjukkan tidak ada data yang hilang.
- Setiap fitur dalam dataset dapat digunakan tanpa perlu penanganan nilai hilang.
- Terdapat 0 baris duplikat dalam dataset, memastikan bahwa setiap entri unik.

```
pd.set_option('display.float_format', lambda x: '%.5f' % x)

properti.describe()
```


```
{"summary":"{\n  \"name\": \"properti\",\n  \"rows\": 8,\n
\"fields\": [\n    {\n      \"column\": \"squaremeters\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
33370.682672584044,\n        \"min\": 89.0,\n        \"max\":
99999.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
49870.1312,\n          50105.5,\n          10000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"numberofrooms\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3518.990372256432,\n        \"min\": 1.0,\n        \"max\": 10000.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          50.3584,\
n          50.0,\n          10000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"floors\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
3518.9414356189227,\n        \"min\": 1.0,\n        \"max\": 10000.0,\
```

```
n        \"num_unique_values\": 8,\n        \"samples\": [\n
50.2763,\n          50.0,\n          10000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"citycode\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
33573.27314811567,\n        \"min\": 3.0,\n        \"max\": 99953.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
50225.4861,\n          50693.0,\n          10000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"citypartrange\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3533.748026680069,\n        \"min\": 1.0,\n        \"max\": 10000.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          5.5101,\n
5.0,\n          10000.0\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"numprevowners\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 3533.748218032391,\n
\"min\": 1.0,\n        \"max\": 10000.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          5.5217,\n
5.0,\n          10000.0\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"made\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 3009.508553763657,\n        \"min\":
9.308089589340048,\n        \"max\": 10000.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
2005.4885,\n          2005.5,\n          10000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"basement\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
3597.587561948309,\n        \"min\": 0.0,\n        \"max\": 10000.0,\n
\"num_unique_values\": 7,\n        \"samples\": [\n          10000.0,\
n          5033.1039,\n          5092.5\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"attic\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 3604.1691191516716,\
n        \"min\": 1.0,\n        \"max\": 10000.0,\n
\"num_unique_values\": 7,\n        \"samples\": [\n          10000.0,\
n          5028.0106,\n          5045.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"garage\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
3367.2965190625278,\n        \"min\": 100.0,\n        \"max\":
10000.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
553.1212,\n          554.0,\n          10000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"hasguestroom\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3533.859948946828,\n        \"min\": 0.0,\n        \"max\": 10000.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          4.9946,\n
```

```
5.0,\n          10000.0\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"price\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 3491484.2164380853,\n        \"min\":
10000.0,\n        \"max\": 10006771.2,\n        \"num_unique_values\":
8,\n        \"samples\": [\n          4993447.52575,\n
5016180.3,\n          10000.0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe"}
```


```
df_properti = properti.copy()
df_properti.head()
df_properti.columns
```

```
Index(['squaremeters', 'numberofrooms', 'hasyard', 'haspool',
'floors',
       'citycode', 'citypartrange', 'numprevowners', 'made',
'isnewbuilt',
       'hasstormprotector', 'basement', 'attic', 'garage',
'hasstorageroom',
       'hasguestroom', 'price', 'category'],
      dtype='object')
```

Cek jumlah kelas

```
columns_to_check = ['squaremeters', 'numberofrooms', 'hasyard',
'haspool', 'floors',
                    'citycode', 'citypartrange', 'numprevowners',
'made', 'isnewbuilt',
                    'hasstormprotector', 'basement', 'attic',
'garage', 'hasstorageroom',
                    'hasguestroom', 'price', 'category']

for col in columns_to_check:
    print("#" * 50)
    print(f"Distribusi kelas untuk kolom: {col}")
    print("#" * 50)

    print(properti[col].value_counts())

    if properti[col].dtype in ['float64', 'int64']:
        print("\nStatistik Deskriptif:")
        print(properti[col].describe())

    print("\n")
```

```
##################################################
Distribusi kelas untuk kolom: squaremeters
##################################################
squaremeters
```

```
33749    3
68985    3
84311    3
52141    3
96526    3
        ..
96930    1
68572    1
98822    1
93762    1
44403    1
Name: count, Length: 9483, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean    49870.13120
std     28774.37535
min        89.00000
25%     25098.50000
50%     50105.50000
75%     74609.75000
max     99999.00000
Name: squaremeters, dtype: float64


##################################################
Distribusi kelas untuk kolom: numberofrooms
##################################################
numberofrooms
54    129
4     120
22    119
47    118
3     116
     ...
6      85
34     84
31     84
40     82
9      75
Name: count, Length: 100, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean       50.35840
std        28.81670
min         1.00000
25%        25.00000
50%        50.00000
75%        75.00000
```

```
max        100.00000
Name: numberofrooms, dtype: float64


##################################################
Distribusi kelas untuk kolom: hasyard
##################################################
hasyard
yes    5087
no     4913
Name: count, dtype: int64


##################################################
Distribusi kelas untuk kolom: haspool
##################################################
haspool
no     5032
yes    4968
Name: count, dtype: int64


##################################################
Distribusi kelas untuk kolom: floors
##################################################
floors
97     126
55     122
77     117
28     116
3      116
      ...
74      83
48      83
15      83
100     82
92      75
Name: count, Length: 100, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean       50.27630
std        28.88917
min         1.00000
25%        25.00000
50%        50.00000
75%        76.00000
max       100.00000
Name: floors, dtype: float64
```

```
################################################
Distribusi kelas untuk kolom: citycode
################################################
citycode
37363    3
36929    3
82521    3
83194    3
16401    3
        ..
91668    1
50551    1
22367    1
58917    1
18412    1
Name: count, Length: 9509, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean    50225.48610
std     29006.67580
min         3.00000
25%     24693.75000
50%     50693.00000
75%     75683.25000
max     99953.00000
Name: citycode, dtype: float64


################################################
Distribusi kelas untuk kolom: citypartrange
################################################
citypartrange
8     1035
5     1031
10    1004
4     1001
3      999
9      997
1      994
2      990
7      984
6      965
Name: count, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean        5.51010
std         2.87202
```

```
min          1.00000
25%          3.00000
50%          5.00000
75%          8.00000
max         10.00000
Name: citypartrange, dtype: float64


##################################################
Distribusi kelas untuk kolom: numprevowners
##################################################
numprevowners
4     1043
5     1036
9     1036
6     1011
10     999
3      991
2      987
7      974
8      971
1      952
Name: count, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean        5.52170
std         2.85667
min         1.00000
25%         3.00000
50%         5.00000
75%         8.00000
max        10.00000
Name: numprevowners, dtype: float64


##################################################
Distribusi kelas untuk kolom: made
##################################################
made
1992    356
2013    352
2020    336
2018    334
2001    332
2003    332
1996    327
1991    324
2009    324
2011    321
```

```
2019    321
1993    320
1998    318
1990    317
1994    312
2014    312
2016    307
2004    307
2012    305
2015    305
2021    304
2008    302
2007    302
2006    296
2005    296
1997    296
2000    295
1999    293
2010    291
2002    290
2017    288
1995    285
Name: count, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean     2005.48850
std         9.30809
min      1990.00000
25%      1997.00000
50%      2005.50000
75%      2014.00000
max      2021.00000
Name: made, dtype: float64


##################################################
Distribusi kelas untuk kolom: isnewbuilt
##################################################
isnewbuilt
old    5009
new    4991
Name: count, dtype: int64


##################################################
Distribusi kelas untuk kolom: hasstormprotector
##################################################
hasstormprotector
no     5001
```

```
yes    4999
Name: count, dtype: int64


##################################################
Distribusi kelas untuk kolom: basement
##################################################
basement
1421    6
2192    6
4170    6
6899    6
9186    5
       ..
2411    1
252     1
2844    1
4845    1
8485    1
Name: count, Length: 6352, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean     5033.10390
std      2876.72954
min         0.00000
25%      2559.75000
50%      5092.50000
75%      7511.25000
max     10000.00000
Name: basement, dtype: float64


##################################################
Distribusi kelas untuk kolom: attic
##################################################
attic
3127    7
5017    6
6556    6
9708    6
8481    6
       ..
5453    1
5933    1
767     1
4042    1
5266    1
Name: count, Length: 6267, dtype: int64
```

```
Statistik Deskriptif:
count   10000.00000
mean     5028.01060
std      2894.33221
min         1.00000
25%      2512.00000
50%      5045.00000
75%      7540.50000
max     10000.00000
Name: attic, dtype: float64


##################################################
Distribusi kelas untuk kolom: garage
##################################################
garage
253    24
955    21
866    20
745    20
968    20
       ..
193     4
887     3
483     3
589     2
282     2
Name: count, Length: 901, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean      553.12120
std       262.05017
min       100.00000
25%       327.75000
50%       554.00000
75%       777.25000
max      1000.00000
Name: garage, dtype: float64


##################################################
Distribusi kelas untuk kolom: hasstorageroom
##################################################
hasstorageroom
yes    5030
no     4970
Name: count, dtype: int64
```

```
################################################
Distribusi kelas untuk kolom: hasguestroom
################################################
hasguestroom
2     942
10    926
9     916
0     914
8     913
4     911
1     910
3     906
6     904
7     884
5     874
Name: count, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean        4.99460
std         3.17641
min         0.00000
25%         2.00000
50%         5.00000
75%         8.00000
max        10.00000
Name: hasguestroom, dtype: float64


################################################
Distribusi kelas untuk kolom: price
################################################
price
7559081.50000    1
2600292.10000    1
3804577.40000    1
3658559.70000    1
2316639.40000    1
                ..
5555606.60000    1
5501007.50000    1
9986201.20000    1
9104801.80000    1
146708.40000     1
Name: count, Length: 10000, dtype: int64

Statistik Deskriptif:
count      10000.00000
mean     4993447.52575
std      2877424.10995
```

```
min        10313.50000
25%      2516401.95000
50%      5016180.30000
75%      7469092.45000
max     10006771.20000
Name: price, dtype: float64


##################################################
Distribusi kelas untuk kolom: category
##################################################
category
Basic      4344
Luxury     3065
Middle     2591
Name: count, dtype: int64
```

Kolom category:

- Basic: 4344
- Luxury: 3065
- Middle: 2591

Ini menunjukkan sedikit ketidakseimbangan. Kategori Basic memiliki jumlah yang jauh lebih banyak dibandingkan Luxury dan Middle. Dalam kasus klasifikasi, ini mungkin bisa menyebabkan ketidakseimbangan performa model, terutama jika model lebih cenderung ke kelas yang lebih dominan.

```
X = df_properti.drop(columns=['price', 'category'], axis=1)
y = df_properti['category']

X_train_bf, X_test, y_train_bf, y_test = train_test_split(X, y,
test_size=0.30, random_state=77)
print(f"Shape of X_train: {X_train_bf.shape}")
print(f"Shape of X_test: {X_test.shape}")

Shape of X_train: (7000, 16)
Shape of X_test: (3000, 16)

print(X.columns)

Index(['squaremeters', 'numberofrooms', 'hasyard', 'haspool',
'floors',
       'citycode', 'citypartrange', 'numprevowners', 'made',
'isnewbuilt',
       'hasstormprotector', 'basement', 'attic', 'garage',
'hasstorageroom',
```

```
        'hasguestroom'],
      dtype='object')
```

- Mengubah variabel kategori menjadi numerik.
- Membagi data menjadi set pelatihan dan pengujian.

```
cat_cols=[ 'hasyard', 'haspool', 'isnewbuilt',
          'hasstormprotector', 'hasstorageroom']

transformer = make_column_transformer(
    (OneHotEncoder(), cat_cols),
    remainder='passthrough'
)

X_train_enc = transformer.fit_transform(X_train_bf)
X_test_enc = transformer.transform(X_test)

df_train_enc = pd.DataFrame(X_train_enc,
columns=transformer.get_feature_names_out())
df_test_enc = pd.DataFrame(X_test_enc,
columns=transformer.get_feature_names_out())

df_train_enc.head(10)
df_test_enc.head(10)

{"type":"dataframe","variable_name":"df_test_enc"}

np.set_printoptions(formatter={'float': '{:.2f}'.format})

print(X_train_enc)

[[1.00 0.00 1.00 ... 746.00 758.00 3.00]
 [1.00 0.00 1.00 ... 4130.00 975.00 10.00]
 [0.00 1.00 0.00 ... 1522.00 103.00 3.00]
 ...
 [1.00 0.00 0.00 ... 2347.00 292.00 9.00]
 [1.00 0.00 0.00 ... 4500.00 767.00 3.00]
 [1.00 0.00 0.00 ... 3734.00 196.00 10.00]]

from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=77)

X_folds = []
y_folds = []

for train_index, test_index in skf.split(X_train_enc, y_train_bf):
    X_folds.append((X_train_enc[train_index],
X_train_enc[test_index]))
    y_folds.append((y_train_bf.iloc[train_index],
y_train_bf.iloc[test_index]))
```

```
print(f"Total folds created: {len(X_folds)}")
```

```
Total folds created: 5
```

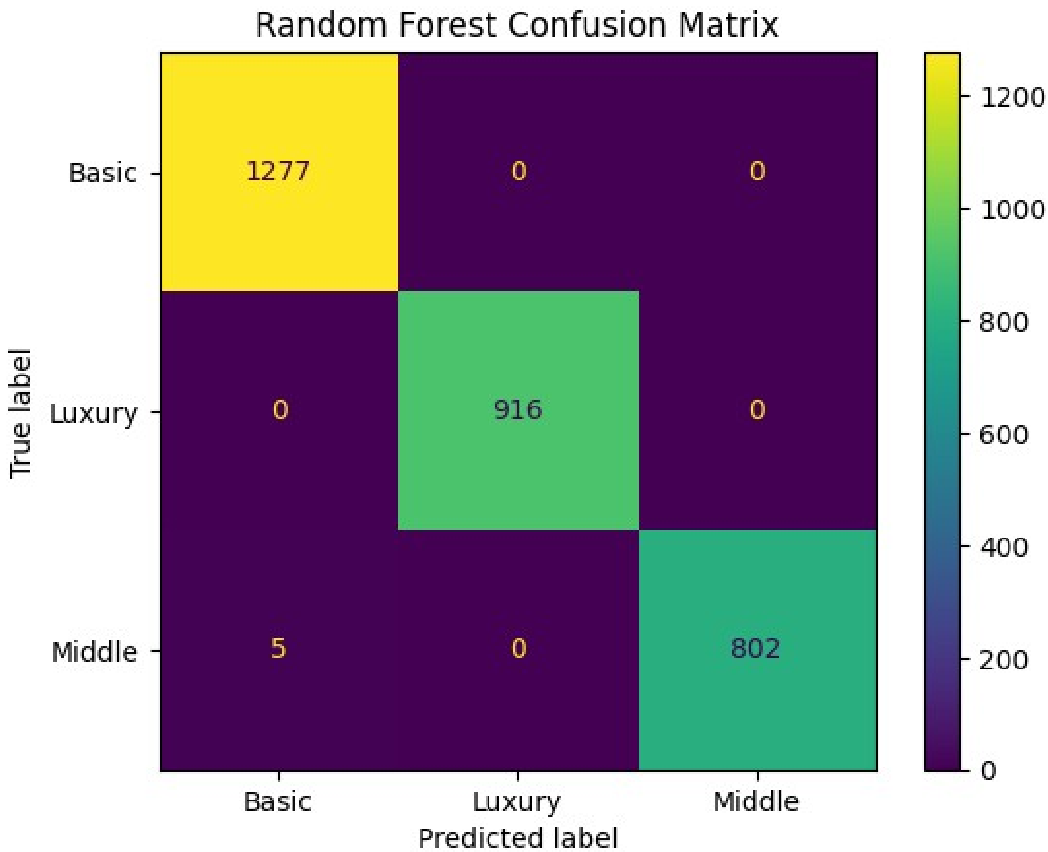
Random Forest

```
pipe_RF = Pipeline(steps=[
    ('data scaling', StandardScaler()),
    ('feature select', SelectKBest()),
    ('clf', RandomForestClassifier(random_state=77,
class_weight='balanced'))
])

params_grid_RF = [
    {
        'data scaling': [StandardScaler()],
        'feature select__k': np.arange(2, 6),
        'clf__max_depth': np.arange(4, 5),
        'clf__n_estimators': [100, 150]
    },
    {
        'data scaling': [StandardScaler()],
        'feature select': [SelectPercentile()],
        'feature select__percentile': np.arange(20, 50),
        'clf__max_depth': np.arange(4, 5),
        'clf__n_estimators': [100, 150]
    },
    {
        'data scaling': [MinMaxScaler()],
        'feature select__k': np.arange(2, 6),
        'clf__max_depth': np.arange(4, 5),
        'clf__n_estimators': [100, 150]
    },
    {
        'data scaling': [MinMaxScaler()],
        'feature select': [SelectPercentile()],
        'feature select__percentile': np.arange(20, 50),
        'clf__max_depth': np.arange(4, 5),
        'clf__n_estimators': [100, 150]
    }
]

GSCV_RF = GridSearchCV(pipe_RF, params_grid_RF, cv=skf,
scoring='accuracy', error_score='raise')
GSCV_RF.fit(X_train_enc, y_train_bf)
print("Random Forest training finished")
```

```
/usr/local/lib/python3.10/dist-packages/numpy/ma/core.py:2820:
RuntimeWarning: invalid value encountered in cast
  _data = np.array(data, dtype=dtype, copy=copy,

Random Forest training finished
```

```
print("CV Score: {}".format(GSCV_RF.best_score_))
print("Test Score:
{}".format(GSCV_RF.best_estimator_.score(X_test_enc, y_test)))
print("Best model:", GSCV_RF.best_estimator_)

mask = GSCV_RF.best_estimator_.named_steps['feature
select'].get_support()

RF_pred = GSCV_RF.predict(X_test_enc)

cm = confusion_matrix(y_test, RF_pred, labels=GSCV_RF.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=GSCV_RF.classes_)
disp.plot()
plt.title("Random Forest Confusion Matrix")
plt.show()

print("Classification report RF: \n", classification_report(y_test,
RF_pred))
```

```
CV Score: 0.9997142857142858
Test Score: 0.9983333333333333
Best model: Pipeline(steps=[('data scaling', MinMaxScaler()),
                ('feature select', SelectPercentile(percentile=36)),
                ('clf',
                 RandomForestClassifier(class_weight='balanced',
max_depth=4,
                                        random_state=77))])
```




```
Classification report RF:
              precision    recall  f1-score   support

       Basic       1.00      1.00      1.00      1277
      Luxury       1.00      1.00      1.00       916
      Middle       1.00      0.99      1.00       807

    accuracy                           1.00      3000
   macro avg       1.00      1.00      1.00      3000
weighted avg       1.00      1.00      1.00      3000
```

Logistic Regression

```
from sklearn.model_selection import GridSearchCV, StratifiedKFold
pipe_LR = Pipeline(steps=[
    ('data scaling', StandardScaler()),
    ('feature select', SelectKBest()),
    ('clf', LogisticRegression(max_iter=1000, class_weight='balanced',
random_state=77))
])

params_grid_LR = [
```

```
    {
        'data scaling': [StandardScaler()],
        'feature select__k': np.arange(2, 6),
        'clf__C': [0.01, 0.1, 1, 10, 100],
        'clf__solver': ['liblinear']
    },
    {
        'data scaling': [MinMaxScaler()],
        'feature select__k': np.arange(2, 6),
        'clf__C': [0.01, 0.1, 1, 10, 100],
        'clf__solver': ['liblinear']
    },
    {
        'data scaling': [StandardScaler()],
        'feature select': [SelectPercentile()],
        'feature select__percentile': np.arange(20, 50),
        'clf__C': [0.01, 0.1, 1, 10, 100],
        'clf__solver': ['liblinear']
    },
    {
        'data scaling': [MinMaxScaler()],
        'feature select': [SelectPercentile()],
        'feature select__percentile': np.arange(20, 50),
        'clf__C': [0.01, 0.1, 1, 10, 100],
        'clf__solver': ['liblinear']
    }
]

GSCV_LR = GridSearchCV(pipe_LR, params_grid_LR, cv=skf,
scoring='accuracy', error_score='raise')
GSCV_LR.fit(X_train_enc, y_train_bf)
print("Logistic Regression training finished")
```

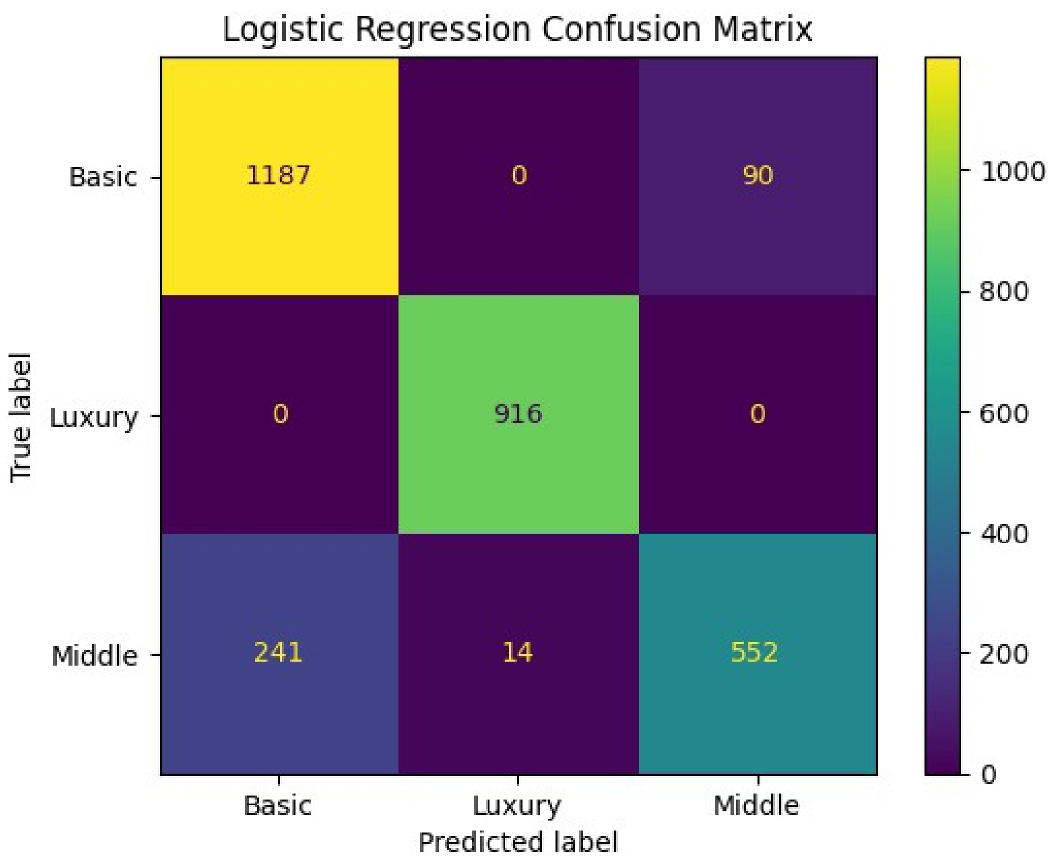
```
Logistic Regression training finished
```

```
/usr/local/lib/python3.10/dist-packages/numpy/ma/core.py:2820:
RuntimeWarning: invalid value encountered in cast
  _data = np.array(data, dtype=dtype, copy=copy,
```

```
print("CV Score: {}".format(GSCV_LR.best_score_))
print("Test Score:
{}".format(GSCV_LR.best_estimator_.score(X_test_enc, y_test)))
print("Best model:", GSCV_LR.best_estimator_)

logistic_pred = GSCV_LR.predict(X_test_enc)

cm = confusion_matrix(y_test, logistic_pred, labels=GSCV_LR.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=GSCV_LR.classes_)
disp.plot()
```

```
plt.title("Logistic Regression Confusion Matrix")
plt.show()

print("Classification report Logistic Regression: \n",
classification_report(y_test, logistic_pred))
```

```
CV Score: 0.8767142857142858
Test Score: 0.885
Best model: Pipeline(steps=[('data scaling', StandardScaler()),
                ('feature select', SelectKBest(k=4)),
                ('clf',
                 LogisticRegression(C=100, class_weight='balanced',
                                    max_iter=1000, random_state=77,
                                    solver='liblinear'))])
```




```
Classification report Logistic Regression:
               precision    recall  f1-score   support

       Basic       0.83      0.93      0.88      1277
      Luxury       0.98      1.00      0.99       916
      Middle       0.86      0.68      0.76       807

    accuracy                           0.89      3000
```

```
   macro avg       0.89      0.87      0.88      3000
weighted avg       0.89      0.89      0.88      3000
```

1. Cross-Validation Score (CV Score) Random Forest: 0.9999 Logistic Regression: 0.8801 Random Forest memiliki CV Score yang jauh lebih tinggi dibandingkan Logistic Regression. CV Score menunjukkan performa rata-rata model di beberapa subset data, sehingga Random Forest terlihat sangat unggul dalam hal ini.

2. Test Score Random Forest: 0.999 Logistic Regression: 0.871 Random Forest juga mengungguli Logistic Regression dalam Test Score, yang merupakan performa pada data uji yang tidak pernah dilihat model selama pelatihan. Ini menunjukkan bahwa Random Forest memiliki prediksi yang hampir sempurna.

3. Classification Report

a. Random Forest Precision, recall, dan f1-score semuanya 1.00 untuk ketiga kelas (Basic, Luxury, Middle). Artinya, model ini membuat prediksi yang sempurna di semua metrik dan kelas.

b. Logistic Regression Precision, recall, dan f1-score Logistic Regression juga cukup baik, tetapi masih ada beberapa ketidakakuratan, terutama di kelas Middle: Precision: 0.80 untuk kelas Middle, lebih rendah dari Random Forest. Recall: 0.67 untuk kelas Middle, menunjukkan bahwa Logistic Regression gagal mengidentifikasi beberapa instance dari kelas Middle dengan benar. F1-score: 0.73 untuk kelas Middle, menandakan keseimbangan precision dan recall yang tidak setinggi Random Forest.

1. Akurasi Keseluruhan Random Forest: 100% akurasi. Logistic Regression: 87.1% akurasi.

Kesimpulan: Random Forest merupakan model yang lebih baik dari hasil evaluasi ini, karena memiliki:

- CV Score yang lebih tinggi.
- Test Score yang lebih baik.
- Classification report yang menunjukkan prediksi sempurna untuk semua kelas (precision, recall, f1-score = 1.00).

Model Terbaik dari perbandingan Algoritme

Diantara Perbandingan 2 Notebook masing masing dari perbandingan tersebut memiliki model terbaik yaitu Notebook pertama ialah Random Forest dan Notebook kedua adalah Gradient B. Sekarang untuk menentukan Model terbaik untuk Klasifikasi, perlu melihat perbandingan hasil dibawah:

1. Cross-Validation Score (CV Score) Random Forest: 0.9999 Gradient Boosting Classifier: 0.9994 Keterangan: Random Forest memiliki CV Score yang sedikit lebih tinggi dari Gradient Boosting Classifier, meskipun perbedaannya sangat kecil.

2. Test Score Random Forest: 0.999 Gradient Boosting Classifier: 0.999 Keterangan: Keduanya memiliki Test Score yang sama, yaitu 0.999, yang menunjukkan bahwa

kedua model sangat baik dalam memprediksi data uji dengan tingkat akurasi hampir sempurna.

3. Classification Report Random Forest: Precision, recall, dan f1-score semuanya 1.00 untuk ketiga kelas (Basic, Luxury, Middle). Gradient Boosting Classifier: Precision, recall, dan f1-score semuanya 1.00 untuk ketiga kelas (Basic, Luxury, Middle). Keterangan: Kedua model memberikan prediksi yang sempurna (1.00) untuk semua kelas, menunjukkan bahwa tidak ada kesalahan klasifikasi pada data yang diuji.

4. Akurasi Keseluruhan Random Forest: 100% Gradient Boosting Classifier: 100% Keterangan: Keduanya memiliki akurasi keseluruhan yang sama, yaitu 100%.

Kesimpulan Berdasarkan nilai CV Score, Test Score, dan classification report, Random Forest dan Gradient Boosting Classifier menunjukkan performa yang sangat mirip, dengan Random Forest memiliki sedikit keunggulan pada CV Score.

Namun, karena perbedaannya sangat kecil dan keduanya memberikan akurasi sempurna, **Random Forest** dapat dianggap sebagai model terbaik untuk Klasifikasi ini. Model ini tidak hanya unggul pada CV Score, tetapi juga memberikan hasil sempurna di semua metrik pada data uji, membuatnya pilihan terbaik untuk klasifikasi secara keseluruhan.

```
import pickle

with open('/content/drive/MyDrive/Colab
Notebooks/RF_Properti_model.pkl', 'wb') as r:
    pickle.dump(GSCV_RF, r)

print("Model Random Forest berhasil disimpan")
```

```
Model Random Forest berhasil disimpan
```

Proyek UTS PMDPM Gasal 2023/2024

Nama Anggota Kelompok:

- Teofilos Mas Krisna Dewa - 220711838
- Marcello Aaron K - 220711844
- Rizky Ardiansyah Ramadhan - 220711861
- Reinaldy Restu Aji - 220711877

Inisialisasi

- Import library yang dibutuhkan

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler,
StandardScaler
from sklearn.compose import make_column_transformer
from sklearn.feature_selection import SelectKBest, SelectPercentile,
RFE
from sklearn.model_selection import GridSearchCV, train_test_split,
StratifiedKFold, KFold
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.linear_model import LogisticRegression, Ridge, Lasso,
LinearRegression
from imblearn.over_sampling import SMOTE
from collections import Counter
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix,
mean_squared_error, mean_absolute_error, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
```

Data Loading

- Proses data loading (boleh dengan file upload atau dengan mount drive jika menggunakan Google Colab)

```
from google.colab import drive
drive.mount('/content/drive')
properti = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Dataset
Property/Dataset UTS_Gasal 2425.csv")
properti.head(10000)
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

{"summary":"{\n \"name\": \"properti\",\n \"rows\": 10000,\n \"fields\": [\n   {\n   \"column\": \"squaremeters\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 28774,\n   \"min\": 89,\n   \"max\": 99999,\n   \"num_unique_values\": 9483,\n   \"samples\": [\n   2725,\n   98025,\n   4198\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"numberofrooms\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 28,\n   \"min\": 1,\n   \"max\": 100,\n   \"num_unique_values\": 100,\n   \"samples\": [\n   44,\n   95,\n   4\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"hasyard\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"yes\",\n   \"no\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"haspool\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"no\",\n   \"yes\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"floors\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 28,\n   \"min\": 1,\n   \"max\": 100,\n   \"num_unique_values\": 100,\n   \"samples\": [\n   90,\n   24\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"citycode\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 29006,\n   \"min\": 3,\n   \"max\": 99953,\n   \"num_unique_values\": 9509,\n   \"samples\": [\n   64029,\n   82892\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"citypartrange\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 2,\n   \"min\": 1,\n   \"max\": 10,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   9,\n   6\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"numprevowners\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 2,\n   \"min\": 1,\n   \"max\": 10,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   6,\n   4\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"made\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 9,\n   \"min\": 1990,\n   \"max\": 2021,\n   \"num_unique_values\": 32,\n   \"samples\": [\n   2019,\n   1990\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\

```
n    },\n    {\n      \"column\": \"isnewbuilt\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          \"new\",\
n          \"old\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"hasstormprotector\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"no\",\n          \"yes\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"basement\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 2876,\n
\"min\": 0,\n        \"max\": 10000,\n        \"num_unique_values\":
6352,\n        \"samples\": [\n          2607,\n          8571\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"attic\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 2894,\n
\"min\": 1,\n        \"max\": 10000,\n        \"num_unique_values\":
6267,\n        \"samples\": [\n          2275,\n          5732\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"garage\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 262,\n
\"min\": 100,\n        \"max\": 1000,\n        \"num_unique_values\":
901,\n        \"samples\": [\n          429,\n          500\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"hasstorageroom\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"yes\",\n          \"no\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"hasguestroom\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3,\n        \"min\": 0,\n        \"max\": 10,\n
\"num_unique_values\": 11,\n        \"samples\": [\n          3,\n
7\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"price\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 2877424.1099450146,\n        \"min\": 10313.5,\n
\"max\": 10006771.2,\n        \"num_unique_values\": 10000,\n
\"samples\": [\n          1056525.7,\n          737118.2\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"category\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
3,\n        \"samples\": [\n          \"Luxury\",\n
\"Middle\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"properti"}
```


Dalam dataset ini terdiri dari beberapa kolom yaitu:

- squaremeters: Luas properti dalam meter persegi.

- numberofrooms: Jumlah kamar di properti.
- hasyard: Menunjukkan apakah properti memiliki halaman (yard) atau tidak (yes/no).
- haspool: Menunjukkan apakah properti memiliki kolam renang (pool) atau tidak (yes/no).
- floors: Jumlah lantai di properti.
- citycode: Kode kota tempat properti berada.
- citypartrange: Rentang bagian kota.
- numprevowners: Jumlah pemilik sebelumnya.
- made: Tahun pembuatan properti.
- isnewbuilt: Menunjukkan apakah properti baru dibangun atau tidak (new/old).
- hasstormprotector: Menunjukkan apakah properti memiliki pelindung badai (storm protector) atau tidak (yes/no).
- basement: Menunjukkan apakah properti memiliki basement atau tidak.
- attic: Menunjukkan apakah properti memiliki loteng (attic) atau tidak.
- garage: Menunjukkan apakah properti memiliki garasi atau tidak.
- hasstorageroom: Menunjukkan apakah properti memiliki ruang penyimpanan (storage room) atau tidak (yes/no).
- List item
- hasguestroom: Menunjukkan apakah properti memiliki kamar tamu (guest room) atau tidak (yes/no).
- price: Harga properti.
- category: Kategori properti (misalnya, Luxury, Middle, dll.).

Data Cleansing & Encoding

- Bagian berikut berisi proses pembersihan data.
- Periksa apakah terdapat missing value dan data duplikat,
- Ubah data kategorik string menjadi numerik.
- Jika jumlah kelas pada data latih tidak seimbang, kalian dapat menggunakan metode oversampling.
- Untuk **Klasifikasi**, pastikan **Harga menjadi target** dan **kolom Kategori dihapus**.

```python
print("#" * 50)
print("Informasi Umum tentang DataFrame:")
print("#" * 50)
properti.info()
print("\n")

print("#" * 50)
print("Missing Values per Column:")
print("#" * 50)
print(properti.isnull().sum())
print("\n")

print("#" * 50)
print("Jumlah Baris Duplikat:")
print("#" * 50)
print(properti.duplicated().sum())
```

```
print("\n")

if properti.duplicated().sum() > 0:
    print("#" * 50)
    print("Baris Duplikat:")
    print("#" * 50)
    print(properti[properti.duplicated()])
else:
    print("#" * 50)
    print("Tidak ada baris duplikat.")
    print("#" * 50)
```

```
##################################################
Informasi Umum tentang DataFrame:
##################################################
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 18 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   squaremeters       10000 non-null  int64
 1   numberofrooms      10000 non-null  int64
 2   hasyard            10000 non-null  object
 3   haspool            10000 non-null  object
 4   floors             10000 non-null  int64
 5   citycode           10000 non-null  int64
 6   citypartrange      10000 non-null  int64
 7   numprevowners      10000 non-null  int64
 8   made               10000 non-null  int64
 9   isnewbuilt         10000 non-null  object
 10  hasstormprotector  10000 non-null  object
 11  basement           10000 non-null  int64
 12  attic              10000 non-null  int64
 13  garage             10000 non-null  int64
 14  hasstorageroom     10000 non-null  object
 15  hasguestroom       10000 non-null  int64
 16  price              10000 non-null  float64
 17  category           10000 non-null  object
dtypes: float64(1), int64(11), object(6)
memory usage: 1.4+ MB


##################################################
Missing Values per Column:
##################################################
squaremeters         0
numberofrooms        0
hasyard              0
haspool              0
floors               0
```

```
citycode             0
citypartrange        0
numprevowners        0
made                 0
isnewbuilt           0
hasstormprotector    0
basement             0
attic                0
garage               0
hasstorageroom       0
hasguestroom         0
price                0
category             0
dtype: int64


##################################################
Jumlah Baris Duplikat:
##################################################
0


##################################################
Tidak ada baris duplikat.
##################################################
```

- Semua kolom memiliki nilai non-null, menunjukkan tidak ada data yang hilang.
- Setiap fitur dalam dataset dapat digunakan tanpa perlu penanganan nilai hilang.
- Terdapat 0 baris duplikat dalam dataset, memastikan bahwa setiap entri unik.

```
pd.set_option('display.float_format', lambda x: '%.5f' % x)

properti.describe()
```

{"summary":"{\n \"name\": \"properti\",\n \"rows\": 8,\n \"fields\": [\n   {\n   \"column\": \"squaremeters\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 33370.682672584044,\n   \"min\": 89.0,\n   \"max\": 99999.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   49870.1312,\n   50105.5,\n   10000.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"numberofrooms\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 3518.990372256432,\n   \"min\": 1.0,\n   \"max\": 10000.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   50.3584,\n   50.0,\n   10000.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"floors\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 3518.9414356189227,\n   \"min\": 1.0,\n   \"max\": 10000.0,\

```
n        \"num_unique_values\": 8,\n        \"samples\": [\n
50.2763,\n          50.0,\n          10000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"citycode\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
33573.27314811567,\n        \"min\": 3.0,\n        \"max\": 99953.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
50225.4861,\n          50693.0,\n          10000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"citypartrange\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3533.748026680069,\n        \"min\": 1.0,\n        \"max\": 10000.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          5.5101,\n
5.0,\n          10000.0\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"numprevowners\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 3533.748218032391,\n
\"min\": 1.0,\n        \"max\": 10000.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          5.5217,\n
5.0,\n          10000.0\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"made\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 3009.508553763657,\n        \"min\":
9.308089589340048,\n        \"max\": 10000.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
2005.4885,\n          2005.5,\n          10000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"basement\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
3597.587561948309,\n        \"min\": 0.0,\n        \"max\": 10000.0,\n
\"num_unique_values\": 7,\n        \"samples\": [\n          10000.0,\
n          5033.1039,\n          5092.5\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"attic\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 3604.1691191516716,\
n        \"min\": 1.0,\n        \"max\": 10000.0,\n
\"num_unique_values\": 7,\n        \"samples\": [\n          10000.0,\
n          5028.0106,\n          5045.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"garage\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
3367.2965190625278,\n        \"min\": 100.0,\n        \"max\":
10000.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
553.1212,\n          554.0,\n          10000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"hasguestroom\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3533.859948946828,\n        \"min\": 0.0,\n        \"max\": 10000.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          4.9946,\n
```

```
5.0,\n          10000.0\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"price\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 3491484.2164380853,\n        \"min\":
10000.0,\n        \"max\": 10006771.2,\n        \"num_unique_values\":
8,\n        \"samples\": [\n          4993447.52575,\n
5016180.3,\n          10000.0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe"}
```


```
df_properti = properti.copy()
df_properti.head()
df_properti.columns
```

```
Index(['squaremeters', 'numberofrooms', 'hasyard', 'haspool',
'floors',
       'citycode', 'citypartrange', 'numprevowners', 'made',
'isnewbuilt',
       'hasstormprotector', 'basement', 'attic', 'garage',
'hasstorageroom',
       'hasguestroom', 'price', 'category'],
      dtype='object')
```

Cek jumlah kelas

```
columns_to_check = ['squaremeters', 'numberofrooms', 'hasyard',
'haspool', 'floors',
                    'citycode', 'citypartrange', 'numprevowners',
'made', 'isnewbuilt',
                    'hasstormprotector', 'basement', 'attic',
'garage', 'hasstorageroom',
                    'hasguestroom', 'price', 'category']

for col in columns_to_check:
    print("#" * 50)
    print(f"Distribusi kelas untuk kolom: {col}")
    print("#" * 50)

    print(properti[col].value_counts())

    if properti[col].dtype in ['float64', 'int64']:
        print("\nStatistik Deskriptif:")
        print(properti[col].describe())

    print("\n")
```

```
##################################################
Distribusi kelas untuk kolom: squaremeters
##################################################
squaremeters
```

```
33749    3
68985    3
84311    3
52141    3
96526    3
        ..
96930    1
68572    1
98822    1
93762    1
44403    1
Name: count, Length: 9483, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean    49870.13120
std     28774.37535
min        89.00000
25%     25098.50000
50%     50105.50000
75%     74609.75000
max     99999.00000
Name: squaremeters, dtype: float64


##################################################
Distribusi kelas untuk kolom: numberofrooms
##################################################
numberofrooms
54    129
4     120
22    119
47    118
3     116
     ...
6      85
34     84
31     84
40     82
9      75
Name: count, Length: 100, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean       50.35840
std        28.81670
min         1.00000
25%        25.00000
50%        50.00000
75%        75.00000
```

```
max        100.00000
Name: numberofrooms, dtype: float64


##################################################
Distribusi kelas untuk kolom: hasyard
##################################################
hasyard
yes    5087
no     4913
Name: count, dtype: int64


##################################################
Distribusi kelas untuk kolom: haspool
##################################################
haspool
no     5032
yes    4968
Name: count, dtype: int64


##################################################
Distribusi kelas untuk kolom: floors
##################################################
floors
97     126
55     122
77     117
28     116
3      116
      ...
74      83
48      83
15      83
100     82
92      75
Name: count, Length: 100, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean       50.27630
std        28.88917
min         1.00000
25%        25.00000
50%        50.00000
75%        76.00000
max       100.00000
Name: floors, dtype: float64
```

```
##################################################
Distribusi kelas untuk kolom: citycode
##################################################
citycode
37363    3
36929    3
82521    3
83194    3
16401    3
        ..
91668    1
50551    1
22367    1
58917    1
18412    1
Name: count, Length: 9509, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean    50225.48610
std     29006.67580
min         3.00000
25%     24693.75000
50%     50693.00000
75%     75683.25000
max     99953.00000
Name: citycode, dtype: float64


##################################################
Distribusi kelas untuk kolom: citypartrange
##################################################
citypartrange
8     1035
5     1031
10    1004
4     1001
3      999
9      997
1      994
2      990
7      984
6      965
Name: count, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean        5.51010
std         2.87202
```

```
min          1.00000
25%          3.00000
50%          5.00000
75%          8.00000
max         10.00000
Name: citypartrange, dtype: float64


##################################################
Distribusi kelas untuk kolom: numprevowners
##################################################
numprevowners
4     1043
5     1036
9     1036
6     1011
10     999
3      991
2      987
7      974
8      971
1      952
Name: count, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean        5.52170
std         2.85667
min         1.00000
25%         3.00000
50%         5.00000
75%         8.00000
max        10.00000
Name: numprevowners, dtype: float64


##################################################
Distribusi kelas untuk kolom: made
##################################################
made
1992    356
2013    352
2020    336
2018    334
2001    332
2003    332
1996    327
1991    324
2009    324
2011    321
```

```
2019    321
1993    320
1998    318
1990    317
1994    312
2014    312
2016    307
2004    307
2012    305
2015    305
2021    304
2008    302
2007    302
2006    296
2005    296
1997    296
2000    295
1999    293
2010    291
2002    290
2017    288
1995    285
Name: count, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean     2005.48850
std         9.30809
min      1990.00000
25%      1997.00000
50%      2005.50000
75%      2014.00000
max      2021.00000
Name: made, dtype: float64


##################################################
Distribusi kelas untuk kolom: isnewbuilt
##################################################
isnewbuilt
old    5009
new    4991
Name: count, dtype: int64


##################################################
Distribusi kelas untuk kolom: hasstormprotector
##################################################
hasstormprotector
no     5001
```

```
yes    4999
Name: count, dtype: int64


##################################################
Distribusi kelas untuk kolom: basement
##################################################
basement
1421    6
2192    6
4170    6
6899    6
9186    5
       ..
2411    1
252     1
2844    1
4845    1
8485    1
Name: count, Length: 6352, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean     5033.10390
std      2876.72954
min         0.00000
25%      2559.75000
50%      5092.50000
75%      7511.25000
max     10000.00000
Name: basement, dtype: float64


##################################################
Distribusi kelas untuk kolom: attic
##################################################
attic
3127    7
5017    6
6556    6
9708    6
8481    6
       ..
5453    1
5933    1
767     1
4042    1
5266    1
Name: count, Length: 6267, dtype: int64
```

```
Statistik Deskriptif:
count   10000.00000
mean     5028.01060
std      2894.33221
min         1.00000
25%      2512.00000
50%      5045.00000
75%      7540.50000
max     10000.00000
Name: attic, dtype: float64


##################################################
Distribusi kelas untuk kolom: garage
##################################################
garage
253    24
955    21
866    20
745    20
968    20
       ..
193     4
887     3
483     3
589     2
282     2
Name: count, Length: 901, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean      553.12120
std       262.05017
min       100.00000
25%       327.75000
50%       554.00000
75%       777.25000
max      1000.00000
Name: garage, dtype: float64


##################################################
Distribusi kelas untuk kolom: hasstorageroom
##################################################
hasstorageroom
yes    5030
no     4970
Name: count, dtype: int64
```

```
##################################################
Distribusi kelas untuk kolom: hasguestroom
##################################################
hasguestroom
2     942
10    926
9     916
0     914
8     913
4     911
1     910
3     906
6     904
7     884
5     874
Name: count, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean        4.99460
std         3.17641
min         0.00000
25%         2.00000
50%         5.00000
75%         8.00000
max        10.00000
Name: hasguestroom, dtype: float64


##################################################
Distribusi kelas untuk kolom: price
##################################################
price
7559081.50000    1
2600292.10000    1
3804577.40000    1
3658559.70000    1
2316639.40000    1
                ..
5555606.60000    1
5501007.50000    1
9986201.20000    1
9104801.80000    1
146708.40000     1
Name: count, Length: 10000, dtype: int64

Statistik Deskriptif:
count      10000.00000
mean     4993447.52575
std      2877424.10995
```

```
min         10313.50000
25%       2516401.95000
50%       5016180.30000
75%       7469092.45000
max      10006771.20000
Name: price, dtype: float64


##################################################
Distribusi kelas untuk kolom: category
##################################################
category
Basic      4344
Luxury     3065
Middle     2591
Name: count, dtype: int64
```

Kolom category:

- Basic: 4344
- Luxury: 3065
- Middle: 2591

Ini menunjukkan sedikit ketidakseimbangan. Kategori Basic memiliki jumlah yang jauh lebih banyak dibandingkan Luxury dan Middle. Dalam kasus klasifikasi, ini mungkin bisa menyebabkan ketidakseimbangan performa model, terutama jika model lebih cenderung ke kelas yang lebih dominan.

```
X = df_properti.drop(columns=['price', 'category'], axis=1)
y = df_properti['category']

X_train_bf, X_test, y_train_bf, y_test = train_test_split(X, y,
test_size=0.30, random_state=77)
print(f"Shape of X_train: {X_train_bf.shape}")
print(f"Shape of X_test: {X_test.shape}")

Shape of X_train: (7000, 16)
Shape of X_test: (3000, 16)

print(X.columns)

Index(['squaremeters', 'numberofrooms', 'hasyard', 'haspool',
'floors',
       'citycode', 'citypartrange', 'numprevowners', 'made',
'isnewbuilt',
       'hasstormprotector', 'basement', 'attic', 'garage',
'hasstorageroom',
```

```
         'hasguestroom'],
       dtype='object')
```

- Mengubah variabel kategori menjadi numerik.
- Membagi data menjadi set pelatihan dan pengujian.

```
cat_cols=['hasyard', 'haspool', 'isnewbuilt',
          'hasstormprotector', 'hasstorageroom']

transformer = make_column_transformer(
    (OneHotEncoder(), cat_cols),
    remainder='passthrough'
)

X_train_enc = transformer.fit_transform(X_train_bf)
X_test_enc = transformer.transform(X_test)

df_train_enc = pd.DataFrame(X_train_enc,
columns=transformer.get_feature_names_out())
df_test_enc = pd.DataFrame(X_test_enc,
columns=transformer.get_feature_names_out())

df_train_enc.head(10)
df_test_enc.head(10)

{"type":"dataframe","variable_name":"df_test_enc"}

np.set_printoptions(formatter={'float': '{:.2f}'.format})

print(X_train_enc)

[[1.00 0.00 1.00 ... 746.00 758.00 3.00]
 [1.00 0.00 1.00 ... 4130.00 975.00 10.00]
 [0.00 1.00 0.00 ... 1522.00 103.00 3.00]
 ...
 [1.00 0.00 0.00 ... 2347.00 292.00 9.00]
 [1.00 0.00 0.00 ... 4500.00 767.00 3.00]
 [1.00 0.00 0.00 ... 3734.00 196.00 10.00]]

kf = KFold(n_splits=5, shuffle=True, random_state=77)

X_folds = []
y_folds = []

for train_index, test_index in kf.split(X_train_enc, y_train_bf):
    X_folds.append((X_train_enc[train_index],
X_train_enc[test_index]))
    y_folds.append((y_train_bf.iloc[train_index],
y_train_bf.iloc[test_index]))

print(f"Total folds created: {len(X_folds)}")
```

```
Total folds created: 5
```

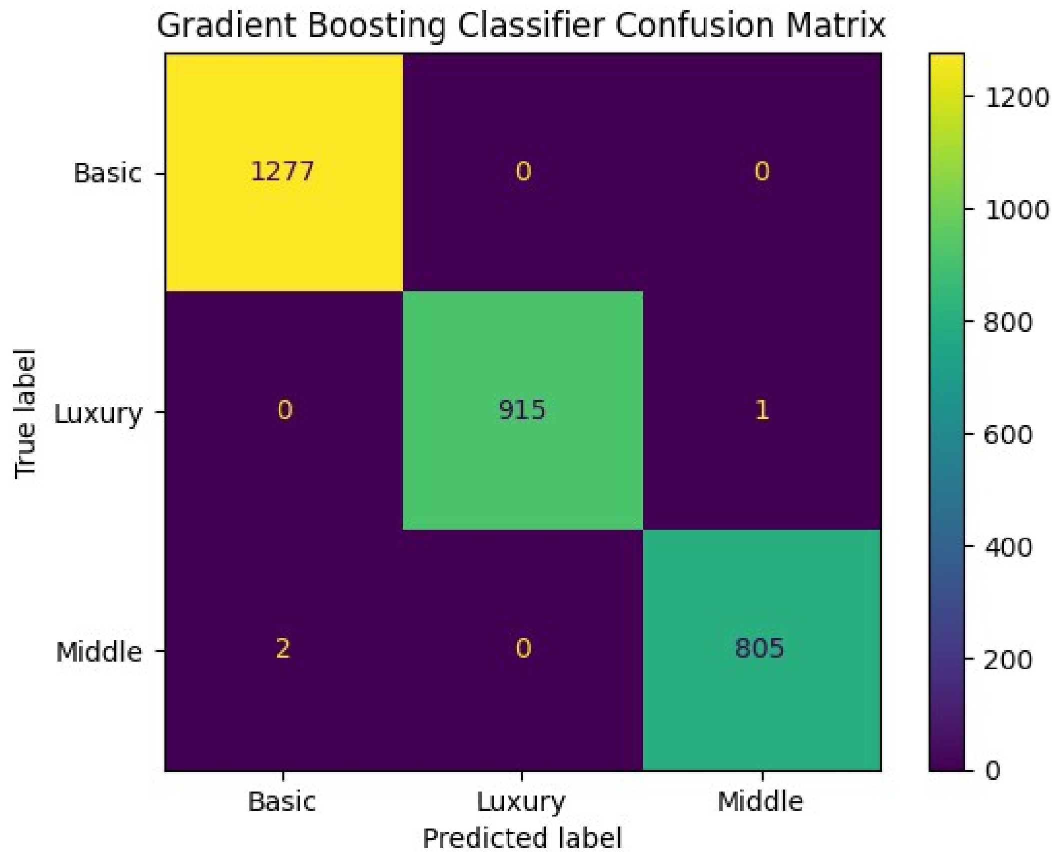
```python
pipe_GBC = Pipeline(steps=[
    ('data scaling', StandardScaler()),
    ('feature select', SelectKBest()),
    ('clf', GradientBoostingClassifier(random_state=77))
])

params_grid_GBC = [
    {
        'data scaling': [StandardScaler()],
        'feature select__k': np.arange(2, 6),
        'clf__n_estimators': [100, 150],
        'clf__learning_rate': [0.01, 0.1, 1],
        'clf__max_depth': [3, 5, 7]
    },
    {
        'data scaling': [MinMaxScaler()],
        'feature select__k': np.arange(2, 6),
        'clf__n_estimators': [50, 100, 150],
        'clf__learning_rate': [0.01, 0.1, 1],
        'clf__max_depth': [3, 5, 7]
    },
    {
        'data scaling': [StandardScaler()],
        'feature select': [SelectPercentile()],
        'feature select__percentile': np.arange(20, 50),
        'clf__n_estimators': [50, 100, 150],
        'clf__learning_rate': [0.01, 0.1, 1],
        'clf__max_depth': [3, 5, 7]
    },
    {
        'data scaling': [MinMaxScaler()],
        'feature select': [SelectPercentile()],
        'feature select__percentile': np.arange(20, 50),
        'clf__n_estimators': [50, 100, 150],
        'clf__learning_rate': [0.01, 0.1, 1],
        'clf__max_depth': [3, 5, 7]
    }
]

GSCV_GBC = GridSearchCV(pipe_GBC, params_grid_GBC, cv=kf,
scoring='accuracy', error_score='raise')
GSCV_GBC.fit(X_train_enc, y_train_bf)
print("Gradient Boosting Classifier training finished")
```

```
Gradient Boosting Classifier training finished
```

```python
print("CV Score: {}".format(GSCV_GBC.best_score_))
print("Test Score:
```

```
{}".format(GSCV_GBC.best_estimator_.score(X_test_enc, y_test)))
print("Best model:", GSCV_GBC.best_estimator_)

gbc_pred = GSCV_GBC.predict(X_test_enc)

cm = confusion_matrix(y_test, gbc_pred, labels=GSCV_GBC.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=GSCV_GBC.classes_)
disp.plot()
plt.title("Gradient Boosting Classifier Confusion Matrix")
plt.show()

print("Classification report Gradient Boosting Classifier: \n",
classification_report(y_test, gbc_pred))
```

```
CV Score: 0.9994285714285714
Test Score: 0.999
Best model: Pipeline(steps=[('data scaling', StandardScaler()),
                ('feature select', SelectPercentile(percentile=46)),
                ('clf',
                 GradientBoostingClassifier(learning_rate=0.01,
max_depth=7,
                                            n_estimators=50,
                                            random_state=77))])
```




```
Classification report Gradient Boosting Classifier:
              precision    recall  f1-score   support

       Basic       1.00      1.00      1.00      1277
      Luxury       1.00      1.00      1.00       916
      Middle       1.00      1.00      1.00       807

    accuracy                           1.00      3000
   macro avg       1.00      1.00      1.00      3000
weighted avg       1.00      1.00      1.00      3000


pipe_SVC = Pipeline(steps=[
    ('data scaling', StandardScaler()),
    ('feature select', SelectKBest()),
    ('clf', SVC(class_weight='balanced', random_state=77))
])

params_grid_SVC = [
    {
        'data scaling': [StandardScaler()],
        'feature select__k': np.arange(2, 6),
        'clf__C': [0.01, 0.1, 1, 10, 100],
```

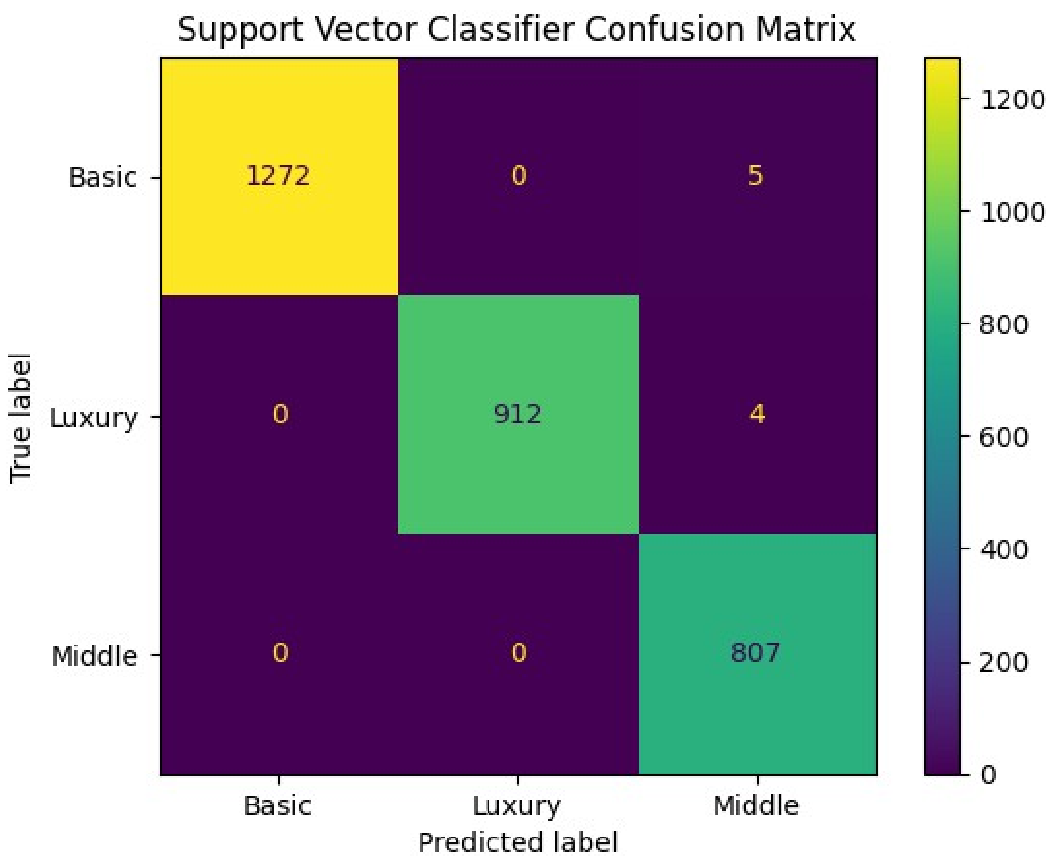
```
        'clf__kernel': ['linear']
    },
    {
        'data scaling': [MinMaxScaler()],
        'feature select__k': np.arange(2, 6),
        'clf__C': [0.01, 0.1, 1, 10, 100],
        'clf__kernel': ['linear']
    },
    {
        'data scaling': [StandardScaler()],
        'feature select': [SelectPercentile()],
        'feature select__percentile': np.arange(20, 50),
        'clf__C': [0.01, 0.1, 1, 10, 100],
        'clf__kernel': ['rbf']
    },
    {
        'data scaling': [MinMaxScaler()],
        'feature select': [SelectPercentile()],
        'feature select__percentile': np.arange(20, 50),
        'clf__C': [0.01, 0.1, 1, 10, 100],
        'clf__kernel': ['rbf']
    }
]

GSCV_SVC = GridSearchCV(pipe_SVC, params_grid_SVC, cv=skf,
scoring='accuracy', error_score='raise')
GSCV_SVC.fit(X_train_enc, y_train_bf)
print("Support Vector Classifier training finished")
```

```
Support Vector Classifier training finished
```

```
print("CV Score: {}".format(GSCV_SVC.best_score_))
print("Test Score:
{}".format(GSCV_SVC.best_estimator_.score(X_test_enc, y_test)))
print("Best model:", GSCV_SVC.best_estimator_)

svc_pred = GSCV_SVC.predict(X_test_enc)

cm = confusion_matrix(y_test, svc_pred, labels=GSCV_SVC.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=GSCV_SVC.classes_)
disp.plot()
plt.title("Support Vector Classifier Confusion Matrix")
plt.show()

print("Classification report Support Vector Classifier: \n",
classification_report(y_test, svc_pred))
```

```
CV Score: 0.9947142857142858
Test Score: 0.997
```

```
Best model: Pipeline(steps=[('data scaling', StandardScaler()),
                ('feature select', SelectPercentile(percentile=31)),
                ('clf', SVC(C=100, class_weight='balanced',
random_state=77))])
```




```
Classification report Support Vector Classifier:
              precision    recall  f1-score   support

       Basic       1.00      1.00      1.00      1277
      Luxury       1.00      1.00      1.00       916
      Middle       0.99      1.00      0.99       807

    accuracy                           1.00      3000
   macro avg       1.00      1.00      1.00      3000
weighted avg       1.00      1.00      1.00      3000
```

1. **Cross-Validation Score (CV Score)**

- **Gradient Boosting Classifier:** 0.9994
- **SVC:** 0.9947

Penjelasan: CV Score menunjukkan performa rata-rata model pada beberapa subset data selama proses pelatihan. **Gradient Boosting Classifier** memiliki CV Score yang sedikit lebih tinggi dibandingkan **SVC** (0.9994 vs. 0.9947). Ini menunjukkan bahwa Gradient Boosting sedikit lebih baik dalam hal kestabilan dan akurasi di berbagai subset data.

2. **Test Score**

- **Gradient Boosting Classifier:** 0.999
- **SVC:** 0.997

Penjelasan: Test Score mengukur akurasi model pada data uji yang tidak pernah dilihat oleh model selama pelatihan. **Gradient Boosting Classifier** memiliki Test Score yang lebih tinggi (0.999) dibandingkan dengan **SVC** (0.997). Ini menunjukkan bahwa Gradient Boosting lebih akurat dalam memprediksi kelas pada data uji dibandingkan SVC.

3. **Classification Report**

- **Gradient Boosting Classifier:**
  - Precision, recall, dan f1-score semuanya **1.00** untuk ketiga kelas (Basic, Luxury, Middle). Ini berarti **Gradient Boosting Classifier** menghasilkan prediksi yang sempurna di semua metrik dan kelas, tanpa kesalahan.
- **Support Vector Classifier (SVC):**
  - Precision dan recall untuk kelas **Basic** dan **Luxury** juga **1.00**, menunjukkan performa sempurna untuk kedua kelas tersebut.
  - Untuk kelas **Middle**, precision adalah **0.99**, dan recall adalah **1.00**, dengan f1-score **0.99**. Ini menunjukkan bahwa SVC sedikit kurang akurat dalam memprediksi kelas **Middle**, meskipun masih sangat baik.

4. **Akurasi Keseluruhan**

- **Gradient Boosting Classifier:** 100% akurasi
- **SVC:** 100% akurasi

Penjelasan: Meskipun kedua model memiliki akurasi keseluruhan yang sama (100%), perbedaan kecil terlihat dalam presisi dan recall untuk kelas **Middle**, di mana Gradient Boosting unggul dengan skor sempurna dibandingkan dengan SVC yang memiliki precision **0.99** untuk kelas ini.

5. **Kesimpulan**

Gradient Boosting Classifier lebih unggul dalam beberapa aspek:

- **CV Score** dan **Test Score** yang sedikit lebih tinggi menunjukkan model ini lebih stabil dan akurat secara keseluruhan.
- **Classification report** untuk **Gradient Boosting** menunjukkan prediksi yang sempurna di semua kelas, sedangkan **SVC** sedikit kurang akurat di kelas **Middle**.

Oleh karena itu, **Gradient Boosting Classifier** adalah model yang lebih baik dalam hal performa keseluruhan dan akurasi pada data uji serta cross-validation, meskipun **SVC** juga menunjukkan hasil yang hampir sempurna.

Proyek UTS PMDPM Gasal 2023/2024

Nama Anggota Kelompok:

- Teofilos Mas Krisna Dewa - 220711838
- Marcello Aaron K - 220711844
- Rizky Ardiansyah Ramadhan - 220711861
- Reinaldy Restu Aji - 220711877

Inisialisasi

- Import library yang dibutuhkan

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler,
StandardScaler, LabelEncoder
from sklearn.compose import make_column_transformer
from sklearn.feature_selection import SelectKBest, SelectPercentile,
f_regression, RFE
from sklearn.model_selection import GridSearchCV, train_test_split,
StratifiedKFold, KFold
from sklearn.metrics import classification_report, confusion_matrix,
mean_squared_error, mean_absolute_error
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier, RandomForestRegressor
from sklearn.linear_model import LogisticRegression, Ridge, Lasso
from sklearn.svm import SVC, SVR
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.neighbors import KNeighborsClassifier
import seaborn as sns
from sklearn.metrics import ConfusionMatrixDisplay
from imblearn.over_sampling import SMOTE
from sklearn.tree import plot_tree
```

Data Loading

- Proses data loading (boleh dengan file upload atau dengan mount drive jika menggunakan Google Colab)

```
from google.colab import drive
drive.mount('/content/drive')
properti = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Dataset
Property/Dataset UTS_Gasal 2425.csv")
properti.head(10000)
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
{"summary":"{\n  \"name\": \"properti\",\n  \"rows\": 10000,\n
\"fields\": [\n    {\n      \"column\": \"squaremeters\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
28774,\n        \"min\": 89,\n        \"max\": 99999,\n
\"num_unique_values\": 9483,\n        \"samples\": [\n          2725,\
n          98025,\n          4198\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"numberofrooms\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
28,\n        \"min\": 1,\n        \"max\": 100,\n
\"num_unique_values\": 100,\n        \"samples\": [\n          44,\n
95,\n          4\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"hasyard\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"yes\",\n          \"no\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"haspool\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
2,\n        \"samples\": [\n          \"no\",\n          \"yes\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"floors\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 28,\n
\"min\": 1,\n        \"max\": 100,\n        \"num_unique_values\":
100,\n        \"samples\": [\n          90,\n          24\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"citycode\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
29006,\n        \"min\": 3,\n        \"max\": 99953,\n
\"num_unique_values\": 9509,\n        \"samples\": [\n
64029,\n          82892\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"citypartrange\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 2,\n        \"min\": 1,\n
\"max\": 10,\n        \"num_unique_values\": 10,\n        \"samples\":
[\n          9,\n          6\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"numprevowners\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 2,\n        \"min\": 1,\n
\"max\": 10,\n        \"num_unique_values\": 10,\n        \"samples\":
[\n          6,\n          4\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"made\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 9,\n        \"min\": 1990,\n
\"max\": 2021,\n        \"num_unique_values\": 32,\n
\"samples\": [\n          2019,\n          1990\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"isnewbuilt\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          \"new\",\
```

```
n          \"old\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"hasstormprotector\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"no\",\n          \"yes\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"basement\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 2876,\n
\"min\": 0,\n        \"max\": 10000,\n        \"num_unique_values\":
6352,\n        \"samples\": [\n          2607,\n          8571\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"attic\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 2894,\n
\"min\": 1,\n        \"max\": 10000,\n        \"num_unique_values\":
6267,\n        \"samples\": [\n          2275,\n          5732\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"garage\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 262,\n
\"min\": 100,\n        \"max\": 1000,\n        \"num_unique_values\":
901,\n        \"samples\": [\n          429,\n          500\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"hasstorageroom\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"yes\",\n          \"no\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"hasguestroom\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3,\n        \"min\": 0,\n        \"max\": 10,\n
\"num_unique_values\": 11,\n        \"samples\": [\n          3,\n
7\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"price\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 2877424.1099450146,\n        \"min\": 10313.5,\n
\"max\": 10006771.2,\n        \"num_unique_values\": 10000,\n
\"samples\": [\n          1056525.7,\n          737118.2\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"category\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
3,\n        \"samples\": [\n          \"Luxury\",\n
\"Middle\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"properti"}
```


Dalam dataset ini terdiri dari beberapa kolom yaitu:

- squaremeters: Luas properti dalam meter persegi.
- numberofrooms: Jumlah kamar di properti.
- hasyard: Menunjukkan apakah properti memiliki halaman (yard) atau tidak (yes/no).

- haspool: Menunjukkan apakah properti memiliki kolam renang (pool) atau tidak (yes/no).
- floors: Jumlah lantai di properti.
- citycode: Kode kota tempat properti berada.
- citypartrange: Rentang bagian kota.
- numprevowners: Jumlah pemilik sebelumnya.
- made: Tahun pembuatan properti.
- isnewbuilt: Menunjukkan apakah properti baru dibangun atau tidak (new/old).
- hasstormprotector: Menunjukkan apakah properti memiliki pelindung badai (storm protector) atau tidak (yes/no).
- basement: Menunjukkan apakah properti memiliki basement atau tidak.
- attic: Menunjukkan apakah properti memiliki loteng (attic) atau tidak.
- garage: Menunjukkan apakah properti memiliki garasi atau tidak.
- hasstorageroom: Menunjukkan apakah properti memiliki ruang penyimpanan (storage room) atau tidak (yes/no).
- List item
- hasguestroom: Menunjukkan apakah properti memiliki kamar tamu (guest room) atau tidak (yes/no).
- price: Harga properti.
- category: Kategori properti (misalnya, Luxury, Middle, dll.).

Data Cleansing & Encoding

- Bagian berikut berisi proses pembersihan data.
- Periksa apakah terdapat missing value dan data duplikat,
- Ubah data kategorik string menjadi numerik.
- Jika jumlah kelas pada data latih tidak seimbang, kalian dapat menggunakan metode oversampling.
- Untuk **Klasifikasi**, pastikan **Harga menjadi target** dan **kolom Kategori dihapus**.

```
print("#" * 50)
print("Informasi Umum tentang DataFrame:")
print("#" * 50)
properti.info()
print("\n")

print("#" * 50)
print("Missing Values per Column:")
print("#" * 50)
print(properti.isnull().sum())
print("\n")

print("#" * 50)
print("Jumlah Baris Duplikat:")
print("#" * 50)
print(properti.duplicated().sum())
print("\n")

if properti.duplicated().sum() > 0:
```

```
    print("#" * 50)
    print("Baris Duplikat:")
    print("#" * 50)
    print(properti[properti.duplicated()])
else:
    print("#" * 50)
    print("Tidak ada baris duplikat.")
    print("#" * 50)
```

```
##################################################
Informasi Umum tentang DataFrame:
##################################################
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 18 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   squaremeters       10000 non-null  int64  
 1   numberofrooms      10000 non-null  int64  
 2   hasyard            10000 non-null  object 
 3   haspool            10000 non-null  object 
 4   floors             10000 non-null  int64  
 5   citycode           10000 non-null  int64  
 6   citypartrange      10000 non-null  int64  
 7   numprevowners      10000 non-null  int64  
 8   made               10000 non-null  int64  
 9   isnewbuilt         10000 non-null  object 
 10  hasstormprotector  10000 non-null  object 
 11  basement           10000 non-null  int64  
 12  attic              10000 non-null  int64  
 13  garage             10000 non-null  int64  
 14  hasstorageroom     10000 non-null  object 
 15  hasguestroom       10000 non-null  int64  
 16  price              10000 non-null  float64
 17  category           10000 non-null  object 
dtypes: float64(1), int64(11), object(6)
memory usage: 1.4+ MB


##################################################
Missing Values per Column:
##################################################
squaremeters         0
numberofrooms        0
hasyard              0
haspool              0
floors               0
citycode             0
citypartrange        0
numprevowners        0
```

```
made                 0
isnewbuilt           0
hasstormprotector    0
basement             0
attic                0
garage               0
hasstorageroom       0
hasguestroom         0
price                0
category             0
dtype: int64


##################################################
Jumlah Baris Duplikat:
##################################################
0


##################################################
Tidak ada baris duplikat.
##################################################
```

- Semua kolom memiliki nilai non-null, menunjukkan tidak ada data yang hilang.
- Setiap fitur dalam dataset dapat digunakan tanpa perlu penanganan nilai hilang.
- Terdapat 0 baris duplikat dalam dataset, memastikan bahwa setiap entri unik.

```
pd.set_option('display.float_format', lambda x: '%.5f' % x)

properti.describe()
```

{"summary":"{\n \"name\": \"properti\",\n \"rows\": 8,\n \"fields\": [\n   {\n   \"column\": \"squaremeters\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 33370.682672584044,\n   \"min\": 89.0,\n   \"max\": 99999.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   49870.1312,\n   50105.5,\n   10000.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"numberofrooms\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 3518.990372256432,\n   \"min\": 1.0,\n   \"max\": 10000.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   50.3584,\n   50.0,\n   10000.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"floors\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 3518.9414356189227,\n   \"min\": 1.0,\n   \"max\": 10000.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   50.2763,\n   50.0,\n   10000.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\

```
n    },\n    {\n      \"column\": \"citycode\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
33573.27314811567,\n        \"min\": 3.0,\n        \"max\": 99953.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
50225.4861,\n          50693.0,\n          10000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"citypartrange\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3533.748026680069,\n        \"min\": 1.0,\n        \"max\": 10000.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          5.5101,\n
5.0,\n          10000.0\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"numprevowners\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 3533.748218032391,\n
\"min\": 1.0,\n        \"max\": 10000.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          5.5217,\n
5.0,\n          10000.0\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"made\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 3009.508553763657,\n        \"min\":
9.308089589340048,\n        \"max\": 10000.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
2005.4885,\n          2005.5,\n          10000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"basement\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
3597.587561948309,\n        \"min\": 0.0,\n        \"max\": 10000.0,\n
\"num_unique_values\": 7,\n        \"samples\": [\n          10000.0,\
n          5033.1039,\n          5092.5\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"attic\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 3604.1691191516716,\
n        \"min\": 1.0,\n        \"max\": 10000.0,\n
\"num_unique_values\": 7,\n        \"samples\": [\n          10000.0,\
n          5028.0106,\n          5045.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"garage\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
3367.2965190625278,\n        \"min\": 100.0,\n        \"max\":
10000.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
553.1212,\n          554.0,\n          10000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"hasguestroom\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3533.859948946828,\n        \"min\": 0.0,\n        \"max\": 10000.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          4.9946,\n
5.0,\n          10000.0\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"price\",\n      \"properties\": {\n        \"dtype\":
```

```
\"number\",\n        \"std\": 3491484.2164380853,\n        \"min\":
10000.0,\n        \"max\": 10006771.2,\n        \"num_unique_values\":
8,\n        \"samples\": [\n          4993447.52575,\n
5016180.3,\n          10000.0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe"}
```


```
df_properti = properti.copy()
df_properti.head()
df_properti.columns
```

```
Index(['squaremeters', 'numberofrooms', 'hasyard', 'haspool',
'floors',
       'citycode', 'citypartrange', 'numprevowners', 'made',
'isnewbuilt',
       'hasstormprotector', 'basement', 'attic', 'garage',
'hasstorageroom',
       'hasguestroom', 'price', 'category'],
      dtype='object')
```

Cek jumlah kelas

```
columns_to_check = ['squaremeters', 'numberofrooms', 'hasyard',
'haspool', 'floors',
                    'citycode', 'citypartrange', 'numprevowners',
'made', 'isnewbuilt',
                    'hasstormprotector', 'basement', 'attic',
'garage', 'hasstorageroom',
                    'hasguestroom', 'price', 'category']

for col in columns_to_check:
    print("#" * 50)
    print(f"Distribusi kelas untuk kolom: {col}")
    print("#" * 50)

    print(properti[col].value_counts())

    if properti[col].dtype in ['float64', 'int64']:
        print("\nStatistik Deskriptif:")
        print(properti[col].describe())

    print("\n")
```

```
##################################################
Distribusi kelas untuk kolom: squaremeters
##################################################
squaremeters
33749    3
68985    3
84311    3
```

```
52141    3
96526    3
        ..
96930    1
68572    1
98822    1
93762    1
44403    1
Name: count, Length: 9483, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean    49870.13120
std     28774.37535
min        89.00000
25%     25098.50000
50%     50105.50000
75%     74609.75000
max     99999.00000
Name: squaremeters, dtype: float64


##################################################
Distribusi kelas untuk kolom: numberofrooms
##################################################
numberofrooms
54    129
4     120
22    119
47    118
3     116
     ...
6      85
34     84
31     84
40     82
9      75
Name: count, Length: 100, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean       50.35840
std        28.81670
min         1.00000
25%        25.00000
50%        50.00000
75%        75.00000
max       100.00000
Name: numberofrooms, dtype: float64
```

```
##################################################
Distribusi kelas untuk kolom: hasyard
##################################################
hasyard
yes    5087
no     4913
Name: count, dtype: int64


##################################################
Distribusi kelas untuk kolom: haspool
##################################################
haspool
no     5032
yes    4968
Name: count, dtype: int64


##################################################
Distribusi kelas untuk kolom: floors
##################################################
floors
97     126
55     122
77     117
28     116
3      116
      ...
74      83
48      83
15      83
100     82
92      75
Name: count, Length: 100, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean       50.27630
std        28.88917
min         1.00000
25%        25.00000
50%        50.00000
75%        76.00000
max       100.00000
Name: floors, dtype: float64


##################################################
Distribusi kelas untuk kolom: citycode
```

```
################################################
citycode
37363    3
36929    3
82521    3
83194    3
16401    3
        ..
91668    1
50551    1
22367    1
58917    1
18412    1
Name: count, Length: 9509, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean    50225.48610
std     29006.67580
min         3.00000
25%     24693.75000
50%     50693.00000
75%     75683.25000
max     99953.00000
Name: citycode, dtype: float64


################################################
Distribusi kelas untuk kolom: citypartrange
################################################
citypartrange
8     1035
5     1031
10    1004
4     1001
3      999
9      997
1      994
2      990
7      984
6      965
Name: count, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean        5.51010
std         2.87202
min         1.00000
25%         3.00000
50%         5.00000
```

```
75%          8.00000
max         10.00000
Name: citypartrange, dtype: float64


##################################################
Distribusi kelas untuk kolom: numprevowners
##################################################
numprevowners
4     1043
5     1036
9     1036
6     1011
10     999
3      991
2      987
7      974
8      971
1      952
Name: count, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean        5.52170
std         2.85667
min         1.00000
25%         3.00000
50%         5.00000
75%         8.00000
max        10.00000
Name: numprevowners, dtype: float64


##################################################
Distribusi kelas untuk kolom: made
##################################################
made
1992    356
2013    352
2020    336
2018    334
2001    332
2003    332
1996    327
1991    324
2009    324
2011    321
2019    321
1993    320
1998    318
```

```
1990    317
1994    312
2014    312
2016    307
2004    307
2012    305
2015    305
2021    304
2008    302
2007    302
2006    296
2005    296
1997    296
2000    295
1999    293
2010    291
2002    290
2017    288
1995    285
Name: count, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean     2005.48850
std         9.30809
min      1990.00000
25%      1997.00000
50%      2005.50000
75%      2014.00000
max      2021.00000
Name: made, dtype: float64


##################################################
Distribusi kelas untuk kolom: isnewbuilt
##################################################
isnewbuilt
old    5009
new    4991
Name: count, dtype: int64


##################################################
Distribusi kelas untuk kolom: hasstormprotector
##################################################
hasstormprotector
no     5001
yes    4999
Name: count, dtype: int64
```

```
##################################################
Distribusi kelas untuk kolom: basement
##################################################
basement
1421    6
2192    6
4170    6
6899    6
9186    5
       ..
2411    1
252     1
2844    1
4845    1
8485    1
Name: count, Length: 6352, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean     5033.10390
std      2876.72954
min         0.00000
25%      2559.75000
50%      5092.50000
75%      7511.25000
max     10000.00000
Name: basement, dtype: float64


##################################################
Distribusi kelas untuk kolom: attic
##################################################
attic
3127    7
5017    6
6556    6
9708    6
8481    6
       ..
5453    1
5933    1
767     1
4042    1
5266    1
Name: count, Length: 6267, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean     5028.01060
```

```
std        2894.33221
min           1.00000
25%        2512.00000
50%        5045.00000
75%        7540.50000
max       10000.00000
Name: attic, dtype: float64


##################################################
Distribusi kelas untuk kolom: garage
##################################################
garage
253    24
955    21
866    20
745    20
968    20
       ..
193     4
887     3
483     3
589     2
282     2
Name: count, Length: 901, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean      553.12120
std       262.05017
min       100.00000
25%       327.75000
50%       554.00000
75%       777.25000
max      1000.00000
Name: garage, dtype: float64


##################################################
Distribusi kelas untuk kolom: hasstorageroom
##################################################
hasstorageroom
yes    5030
no     4970
Name: count, dtype: int64


##################################################
Distribusi kelas untuk kolom: hasguestroom
##################################################
```

```
hasguestroom
2     942
10    926
9     916
0     914
8     913
4     911
1     910
3     906
6     904
7     884
5     874
Name: count, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean        4.99460
std         3.17641
min         0.00000
25%         2.00000
50%         5.00000
75%         8.00000
max        10.00000
Name: hasguestroom, dtype: float64


##################################################
Distribusi kelas untuk kolom: price
##################################################
price
7559081.50000    1
2600292.10000    1
3804577.40000    1
3658559.70000    1
2316639.40000    1
                ..
5555606.60000    1
5501007.50000    1
9986201.20000    1
9104801.80000    1
146708.40000     1
Name: count, Length: 10000, dtype: int64

Statistik Deskriptif:
count      10000.00000
mean     4993447.52575
std      2877424.10995
min        10313.50000
25%      2516401.95000
50%      5016180.30000
```

```
75%      7469092.45000
max     10006771.20000
Name: price, dtype: float64


##################################################
Distribusi kelas untuk kolom: category
##################################################
category
Basic     4344
Luxury    3065
Middle    2591
Name: count, dtype: int64
```

Kolom category:

- Basic: 4344
- Luxury: 3065
- Middle: 2591

Ini menunjukkan sedikit ketidakseimbangan. Kategori Basic memiliki jumlah yang jauh lebih banyak dibandingkan Luxury dan Middle. Dalam kasus klasifikasi, ini mungkin bisa menyebabkan ketidakseimbangan performa model, terutama jika model lebih cenderung ke kelas yang lebih dominan.

```
X = df_properti.drop(columns=['category', 'price'], axis=1)
y = df_properti['price']

X_trainReg, X_testReg, y_trainReg, y_testReg = train_test_split(X, y,
test_size=0.30, random_state=77)
print(f"Shape of X_train: {X_trainReg.shape}")
print(f"Shape of X_test: {X_testReg.shape}")

Shape of X_train: (7000, 16)
Shape of X_test: (3000, 16)

print(X.columns)

Index(['squaremeters', 'numberofrooms', 'hasyard', 'haspool',
'floors',
       'citycode', 'citypartrange', 'numprevowners', 'made',
'isnewbuilt',
       'hasstormprotector', 'basement', 'attic', 'garage',
'hasstorageroom',
       'hasguestroom'],
      dtype='object')
```

- Mengubah variabel kategori menjadi numerik.

- Membagi data menjadi set pelatihan dan pengujian.

```
cat_cols=['hasyard', 'haspool', 'isnewbuilt',
          'hasstormprotector', 'hasstorageroom']

transformer = make_column_transformer(
    (OneHotEncoder(), cat_cols),
    remainder='passthrough'
)

X_trainReg_enc = transformer.fit_transform(X_trainReg)
X_testReg_enc = transformer.transform(X_testReg)

df_trainReg_enc = pd.DataFrame(X_trainReg_enc,
columns=transformer.get_feature_names_out())
df_testReg_enc = pd.DataFrame(X_testReg_enc,
columns=transformer.get_feature_names_out())

df_trainReg_enc.head(10)
df_testReg_enc.head(10)

{"type":"dataframe","variable_name":"df_testReg_enc"}

np.set_printoptions(formatter={'float': '{:.2f}'.format})

print(X_trainReg_enc)

[[1.00 0.00 1.00 ... 746.00 758.00 3.00]
 [1.00 0.00 1.00 ... 4130.00 975.00 10.00]
 [0.00 1.00 0.00 ... 1522.00 103.00 3.00]
 ...
 [1.00 0.00 0.00 ... 2347.00 292.00 9.00]
 [1.00 0.00 0.00 ... 4500.00 767.00 3.00]
 [1.00 0.00 0.00 ... 3734.00 196.00 10.00]]

from sklearn.model_selection import KFold

kf = KFold(n_splits=5, shuffle=True, random_state=77)

X_folds = []
y_folds = []

for train_index, test_index in kf.split(X_trainReg_enc):
    X_folds.append((X_trainReg_enc[train_index],
X_trainReg_enc[test_index]))
    y_folds.append((y_trainReg.iloc[train_index],
y_trainReg.iloc[test_index]))
```

Ridge Regression

```
pipe_Ridge = Pipeline(steps=[
    ('scale', StandardScaler()),
```

```
    ('feature_selection', SelectKBest(score_func=f_regression)),
    ('reg', Ridge())
])

param_grid_Ridge = {
    'reg__alpha': [0.01, 0.1, 1, 10, 100],
    'feature_selection__k': np.arange(1, 20)
}

GSCV_RR = GridSearchCV(pipe_Ridge, param_grid_Ridge, cv=5,
                       scoring='neg_mean_squared_error',
error_score='raise')

GSCV_RR.fit(X_trainReg_enc, y_trainReg)

print("Best Model:", GSCV_RR.best_estimator_)
print("Ridge best parameters:", GSCV_RR.best_params_)

print("Koefisien/bobot:",
GSCV_RR.best_estimator_.named_steps['reg'].coef_)
print("Intercept/bias:",
GSCV_RR.best_estimator_.named_steps['reg'].intercept_)

Ridge_predict = GSCV_RR.predict(X_testReg_enc)

mse_Ridge = mean_squared_error(y_testReg, Ridge_predict)
mae_Ridge = mean_absolute_error(y_testReg, Ridge_predict)

print("Ridge Mean Squared Error (MSE):", mse_Ridge)
print("Ridge Mean Absolute Error (MAE):", mae_Ridge)
print("Ridge Root Mean Squared Error:", np.sqrt(mse_Ridge))

df_results = pd.DataFrame({
    'Actual Price': y_testReg.reset_index(drop=True),
    'Ridge Predicted Price': Ridge_predict
})

df_results['Price Difference (Ridge)'] = df_results['Ridge Predicted
Price'] - df_results['Actual Price']

print(df_results.head())

Best Model: Pipeline(steps=[('scale', StandardScaler()),
                ('feature_selection',
                 SelectKBest(k=19,
                             score_func=<function f_regression at
0x7dd7c4b743a0>)),
                ('reg', Ridge(alpha=0.01))])
Ridge best parameters: {'feature_selection__k': 19, 'reg__alpha':
0.01}
Koefisien/bobot: [-758.20 758.20 -737.76 737.76 38.58 -38.58 -33.02
```

```
33.02 -17.66 17.66
 2877891.20 -2.99 1567.96 -11.43 126.78 -6.39 -25.71 -13.28 39.79]
Intercept/bias: 4986975.165585715
Ridge Mean Squared Error (MSE): 3632947.934956798
Ridge Mean Absolute Error (MAE): 1482.7892392328977
Ridge Root Mean Squared Error: 1906.0293636134775
   Actual Price  Ridge Predicted Price  Price Difference (Ridge)
0 3217288.40000          3219633.82051                2345.42051
1 6491104.10000          6492382.39749                1278.29749
2 3312643.20000          3313893.09519                1249.89519
3 7955735.00000          7958016.79243                2281.79243
4 5105930.70000          5105843.77973                 -86.92027
```

```
/usr/local/lib/python3.10/dist-packages/numpy/ma/core.py:2820:
RuntimeWarning: invalid value encountered in cast
  _data = np.array(data, dtype=dtype, copy=copy,
```

```
pipe_Ridge = Pipeline(steps=[
    ('scale', MinMaxScaler()),
    ('feature_selection', SelectPercentile(score_func=f_regression)),
    ('reg', Ridge())
])

param_grid_Ridge = {
    'reg__alpha': [0.01, 0.1, 1, 10, 100],
    'feature_selection__percentile': np.arange(10, 100, 10)
}

GSCV_Ridge = GridSearchCV(pipe_Ridge, param_grid_Ridge, cv=5,
                          scoring='neg_mean_squared_error',
error_score='raise')

GSCV_Ridge.fit(X_trainReg_enc, y_trainReg)

print("Best Model:", GSCV_Ridge.best_estimator_)
print("Ridge best parameters:", GSCV_Ridge.best_params_)

Ridge_predict = GSCV_Ridge.predict(X_testReg_enc)

mse_Ridge = mean_squared_error(y_testReg, Ridge_predict)
mae_Ridge = mean_absolute_error(y_testReg, Ridge_predict)

print("Ridge MSE:", mse_Ridge)
print("Ridge MAE:", mae_Ridge)
print("Ridge RMSE:", np.sqrt(mse_Ridge))

df_results = pd.DataFrame({
    'Actual Price': y_testReg.reset_index(drop=True),
    'Ridge Predicted Price': Ridge_predict
})
```

```
df_results['Price Difference (Ridge)'] = df_results['Ridge Predicted Price'] - df_results['Actual Price']

print(df_results.head())
```

```
Best Model: Pipeline(steps=[('scale', MinMaxScaler()),
                ('feature_selection',
                 SelectPercentile(percentile=90,
                                  score_func=<function f_regression at
0x7dd7c4b743a0>)),
                ('reg', Ridge(alpha=0.01))])
Ridge best parameters: {'feature_selection__percentile': 90,
'reg__alpha': 0.01}
Ridge MSE: 6182450.1400302
Ridge MAE: 1993.6465821362417
Ridge RMSE: 2486.453325528191
   Actual Price  Ridge Predicted Price  Price Difference (Ridge)
0 3217288.40000          3219390.43363                2102.03363
1 6491104.10000          6494554.01989                3449.91989
2 3312643.20000          3314555.13994                1911.93994
3 7955735.00000          7958183.26901                2448.26901
4 5105930.70000          5104567.79961               -1362.90039

/usr/local/lib/python3.10/dist-packages/numpy/ma/core.py:2820:
RuntimeWarning: invalid value encountered in cast
  _data = np.array(data, dtype=dtype, copy=copy,
```

Support Vector Regressor

```
pipe_SVR = Pipeline(steps=[
    ('scale', StandardScaler()),
    ('feature_selection', SelectKBest(score_func=f_regression)),
    ('reg', SVR())
])

param_grid_SVR = {
    'reg__C': [0.1, 1, 10],
    'reg__epsilon': [0.01, 0.1, 0.5],
    'feature_selection__k': np.arange(1, 20)
}

GSCV_SVR = GridSearchCV(pipe_SVR, param_grid_SVR, cv=5,
scoring='neg_mean_squared_error')
GSCV_SVR.fit(X_trainReg_enc, y_trainReg)

print("Best Model:", GSCV_SVR.best_estimator_)
print("SVR best parameters:", GSCV_SVR.best_params_)

SVR_predict = GSCV_SVR.predict(X_testReg_enc)
```

```
mse_SVR = mean_squared_error(y_testReg, SVR_predict)
mae_SVR = mean_absolute_error(y_testReg, SVR_predict)

print("SVR MSE:", mse_SVR)
print("SVR MAE:", mae_SVR)
print("SVR RMSE:", np.sqrt(mse_SVR))

df_results = pd.DataFrame({
    'Actual Price': y_testReg.reset_index(drop=True),
    'SVR Predicted Price': SVR_predict
})

df_results['Price Difference (SVR)'] = df_results['SVR Predicted
Price'] - df_results['Actual Price']

print(df_results.head())
```

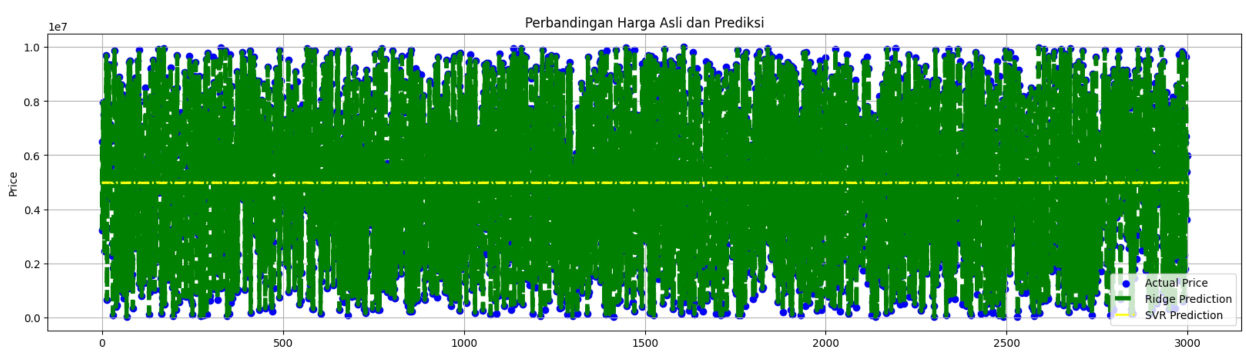
```
Best Model: Pipeline(steps=[('scale', StandardScaler()),
                ('feature_selection',
                 SelectKBest(k=1,
                             score_func=<function f_regression at
0x7dd7c4b743a0>)),
                ('reg', SVR(C=10, epsilon=0.5))])
SVR best parameters: {'feature_selection__k': 1, 'reg__C': 10,
'reg__epsilon': 0.5}
SVR MSE: 8166894730886.678
SVR MAE: 2464137.9266452785
SVR RMSE: 2857777.9358947184
   Actual Price  SVR Predicted Price  Price Difference (SVR)
0 3217288.40000        4956926.13747           1739637.73747
1 6491104.10000        4994643.34742          -1496460.75258
2 3312643.20000        4957644.92710           1645001.72710
3 7955735.00000        5001670.11057          -2954064.88943
4 5105930.70000        4978414.70046           -127515.99954
```

```
pipe_SVR = Pipeline(steps=[
    ('scale', MinMaxScaler()),
    ('feature_selection', SelectPercentile(score_func=f_regression)),
    ('reg', SVR())
])

param_grid_SVR = {
    'reg__C': [0.1, 1, 10],
    'reg__epsilon': [0.01, 0.1, 0.5],
    'feature_selection__percentile': np.arange(10, 100, 10)
}

GSCV_SVR = GridSearchCV(pipe_SVR, param_grid_SVR, cv=5,
scoring='neg_mean_squared_error')
GSCV_SVR.fit(X_trainReg_enc, y_trainReg)
```

```
print("Best Model:", GSCV_SVR.best_estimator_)
print("SVR best parameters:", GSCV_SVR.best_params_)

SVR_predict = GSCV_SVR.predict(X_testReg_enc)

mse_SVR = mean_squared_error(y_testReg, SVR_predict)
mae_SVR = mean_absolute_error(y_testReg, SVR_predict)

print("SVR MSE:", mse_SVR)
print("SVR MAE:", mae_SVR)
print("SVR RMSE:", np.sqrt(mse_SVR))

df_results = pd.DataFrame({
    'Actual Price': y_testReg.reset_index(drop=True),
    'SVR Predicted Price': SVR_predict
})

df_results['Price Difference (SVR)'] = df_results['SVR Predicted
Price'] - df_results['Actual Price']

print(df_results.head())

Best Model: Pipeline(steps=[('scale', MinMaxScaler()),
                ('feature_selection',
                 SelectPercentile(score_func=<function f_regression at
0x7dd7c4b743a0>)),
                ('reg', SVR(C=10, epsilon=0.01))])
SVR best parameters: {'feature_selection__percentile': 10, 'reg__C':
10, 'reg__epsilon': 0.01}
SVR MSE: 8226161446151.247
SVR MAE: 2475131.6892078714
SVR RMSE: 2868128.561649782
   Actual Price  SVR Predicted Price  Price Difference (SVR)
0 3217288.40000        4970388.94190           1753100.54190
1 6491104.10000        4983145.40314          -1507958.69686
2 3312643.20000        4971615.30599           1658972.10599
3 7955735.00000        4986205.04721          -2969529.95279
4 5105930.70000        4977219.35048           -128711.34952

df_results = pd.DataFrame({'Actual Price':
y_testReg.reset_index(drop=True)})
df_results['Ridge Prediction'] = Ridge_predict
df_results['SVR Prediction'] = SVR_predict

df_results['Ridge Price Difference'] = df_results['Actual Price'] -
df_results['Ridge Prediction']
df_results['SVR Price Difference'] = df_results['Actual Price'] -
df_results['SVR Prediction']

print(df_results.head())
```

```
   Actual Price  Ridge Prediction  SVR Prediction  Ridge Price
Difference  \
0 3217288.40000     3219390.43363   4970388.94190              -
2102.03363
1 6491104.10000     6494554.01989   4983145.40314              -
3449.91989
2 3312643.20000     3314555.13994   4971615.30599              -
1911.93994
3 7955735.00000     7958183.26901   4986205.04721              -
2448.26901
4 5105930.70000     5104567.79961   4977219.35048
1362.90039

   SVR Price Difference
0        -1753100.54190
1         1507958.69686
2        -1658972.10599
3         2969529.95279
4          128711.34952
```

```python
plt.figure(figsize=(20, 5))
data_len = range(len(y_testReg))

plt.scatter(data_len, df_results['Actual Price'], label="Actual
Price", color="blue")
plt.plot(data_len, df_results['Ridge Prediction'], label="Ridge
Prediction", color="green", linewidth=4, linestyle="dashed")
plt.plot(data_len, df_results['SVR Prediction'], label="SVR
Prediction", color="yellow", linewidth=2, linestyle="-.")

plt.title("Perbandingan Harga Asli dan Prediksi")
plt.xlabel("Index")
plt.ylabel("Price")
plt.legend()
plt.grid(True)
plt.show()
```



1. **Mean Squared Error (MSE)**

- **Ridge Regression (Pipeline 1):** 3,632,947
- **Ridge Regression (Pipeline 2):** 6,182,450
- **SVR (Pipeline 1):** 8,166,894,730,886
- **SVR (Pipeline 2):** 8,226,161,446,151

Penjelasan: MSE mengukur seberapa jauh rata-rata prediksi dari harga aktual. Nilai yang lebih kecil menunjukkan model yang lebih akurat. **Ridge Regression (Pipeline 1)** memiliki MSE terendah (3,632,947), menunjukkan bahwa model ini menghasilkan prediksi yang paling akurat di antara semua model. Sebaliknya, **SVR** memiliki MSE yang sangat besar (lebih dari 8 triliun), menunjukkan performa yang sangat buruk.

2. **Mean Absolute Error (MAE)**

- **Ridge Regression (Pipeline 1):** 1,482.79
- **Ridge Regression (Pipeline 2):** 1,993.65
- **SVR (Pipeline 1):** 2,464,137.93
- **SVR (Pipeline 2):** 2,475,131.69

Penjelasan: MAE mengukur rata-rata kesalahan absolut antara prediksi dan nilai aktual. Lagi-lagi, **Ridge Regression (Pipeline 1)** memiliki MAE terendah (1,482.79), menunjukkan prediksi yang lebih dekat ke harga aktual. SVR menunjukkan performa yang jauh lebih buruk dengan MAE lebih dari 2 juta, menunjukkan prediksi yang sangat melenceng dari harga sebenarnya.

3. **Root Mean Squared Error (RMSE)**

- **Ridge Regression (Pipeline 1):** 1,906.03
- **Ridge Regression (Pipeline 2):** 2,486.45
- **SVR (Pipeline 1):** 2,857,777.94
- **SVR (Pipeline 2):** 2,868,128.56

Penjelasan: RMSE adalah ukuran yang lebih sensitif terhadap kesalahan besar karena memperhitungkan kuadrat dari kesalahan. **Ridge Regression (Pipeline 1)** memiliki nilai RMSE terendah (1,906.03), menegaskan kembali bahwa model ini lebih baik dalam memprediksi harga dibandingkan SVR yang memiliki RMSE lebih dari 2 juta.

4. **Perbandingan Prediksi Harga**

- **Ridge Regression (Pipeline 1):**
  - Harga Aktual: 3,217,288.4, Prediksi: 3,219,390.43 (Selisih: 2,102.03)
  - Harga Aktual: 6,491,104.1, Prediksi: 6,494,554.02 (Selisih: 3,449.92)
- **SVR (Pipeline 1):**
  - Harga Aktual: 3,217,288.4, Prediksi: 4,970,388.94 (Selisih: 1,753,100.54)
  - Harga Aktual: 6,491,104.1, Prediksi: 4,983,145.40 (Selisih: -1,507,958.70)

Penjelasan: Dari tabel prediksi harga, dapat dilihat bahwa **Ridge Regression** menghasilkan prediksi yang sangat dekat dengan harga aktual, dengan selisih prediksi berkisar antara 2,102 hingga 3,449. Sebaliknya, **SVR** menunjukkan selisih yang sangat besar, mencapai lebih dari 1,7 juta hingga 2,9 juta, menunjukkan model ini sangat tidak akurat dalam memprediksi harga.

5. **Kesimpulan**

Ridge Regression (Pipeline 1) jelas lebih unggul dalam hal:

- **Mean Squared Error (MSE), Mean Absolute Error (MAE), dan Root Mean Squared Error (RMSE)** yang lebih rendah.
- **Prediksi harga** yang lebih akurat dan mendekati nilai aktual.

Support Vector Regressor (SVR) di semua pipeline menunjukkan performa yang buruk dengan kesalahan yang sangat besar, sehingga tidak cocok untuk prediksi harga dalam dataset ini.

Rekomendasi:

Model terbaik untuk regresi berdasarkan hasil evaluasi ini adalah **Ridge Regression (Pipeline 1)** karena memberikan prediksi yang paling akurat dan memiliki error paling kecil di antara model lainnya.

```python
import pandas as pd
import numpy as np
from sklearn.svm import SVR
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler, StandardScaler,
OneHotEncoder
from sklearn.feature_selection import SelectKBest, SelectPercentile,
f_regression
from sklearn.model_selection import GridSearchCV, train_test_split,
StratifiedKFold, KFold
from sklearn.compose import make_column_transformer
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingClassifier, RandomForestClassifier
from sklearn.linear_model import Lasso, Ridge, LogisticRegression
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.neighbors import KNeighborsClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix,
mean_squared_error, mean_absolute_error, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

from google.colab import drive
drive.mount('/content/drive')
properti = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Dataset
Property/Dataset UTS_Gasal 2425.csv")
properti.head(10000)
```

```
Mounted at /content/drive
```


```
{"summary":"{\n  \"name\": \"properti\",\n  \"rows\": 10000,\n
\"fields\": [\n    {\n      \"column\": \"squaremeters\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
28774,\n        \"min\": 89,\n        \"max\": 99999,\n
\"num_unique_values\": 9483,\n        \"samples\": [\n          2725,\
n          98025,\n          4198\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"numberofrooms\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
28,\n        \"min\": 1,\n        \"max\": 100,\n
\"num_unique_values\": 100,\n        \"samples\": [\n          44,\n
95,\n          4\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"hasyard\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"yes\",\n          \"no\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"haspool\",\n      \"properties\":
```

```
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
2,\n        \"samples\": [\n          \"no\",\n          \"yes\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"floors\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 28,\n
\"min\": 1,\n        \"max\": 100,\n        \"num_unique_values\":
100,\n        \"samples\": [\n          90,\n          24\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"citycode\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
29006,\n        \"min\": 3,\n        \"max\": 99953,\n
\"num_unique_values\": 9509,\n        \"samples\": [\n
64029,\n          82892\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"citypartrange\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 2,\n        \"min\": 1,\n
\"max\": 10,\n        \"num_unique_values\": 10,\n        \"samples\":
[\n          9,\n          6\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"numprevowners\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 2,\n        \"min\": 1,\n
\"max\": 10,\n        \"num_unique_values\": 10,\n        \"samples\":
[\n          6,\n          4\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"made\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 9,\n        \"min\": 1990,\n
\"max\": 2021,\n        \"num_unique_values\": 32,\n
\"samples\": [\n          2019,\n          1990\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"isnewbuilt\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          \"new\",\
n          \"old\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"hasstormprotector\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"no\",\n          \"yes\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"basement\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 2876,\n
\"min\": 0,\n        \"max\": 10000,\n        \"num_unique_values\":
6352,\n        \"samples\": [\n          2607,\n          8571\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"attic\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 2894,\n
\"min\": 1,\n        \"max\": 10000,\n        \"num_unique_values\":
6267,\n        \"samples\": [\n          2275,\n          5732\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"garage\",\n      \"properties\":
```

```
{\n        \"dtype\": \"number\",\n        \"std\": 262,\n
\"min\": 100,\n        \"max\": 1000,\n        \"num_unique_values\":
901,\n        \"samples\": [\n          429,\n          500\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"hasstorageroom\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"yes\",\n          \"no\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"hasguestroom\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3,\n        \"min\": 0,\n        \"max\": 10,\n
\"num_unique_values\": 11,\n        \"samples\": [\n          3,\n
7\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"price\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 2877424.1099450146,\n        \"min\": 10313.5,\n
\"max\": 10006771.2,\n        \"num_unique_values\": 10000,\n
\"samples\": [\n          1056525.7,\n          737118.2\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"category\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
3,\n        \"samples\": [\n          \"Luxury\",\n
\"Middle\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"properti"}
```


Dalam dataset ini terdiri dari beberapa kolom yaitu:

- squaremeters: Luas properti dalam meter persegi.
- numberofrooms: Jumlah kamar di properti.
- hasyard: Menunjukkan apakah properti memiliki halaman (yard) atau tidak (yes/no).
- haspool: Menunjukkan apakah properti memiliki kolam renang (pool) atau tidak (yes/no).
- floors: Jumlah lantai di properti.
- citycode: Kode kota tempat properti berada.
- citypartrange: Rentang bagian kota.
- numprevowners: Jumlah pemilik sebelumnya.
- made: Tahun pembuatan properti.
- isnewbuilt: Menunjukkan apakah properti baru dibangun atau tidak (new/old).
- hasstormprotector: Menunjukkan apakah properti memiliki pelindung badai (storm protector) atau tidak (yes/no).
- basement: Menunjukkan apakah properti memiliki basement atau tidak.
- attic: Menunjukkan apakah properti memiliki loteng (attic) atau tidak.
- garage: Menunjukkan apakah properti memiliki garasi atau tidak.
- hasstorageroom: Menunjukkan apakah properti memiliki ruang penyimpanan (storage room) atau tidak (yes/no).
- List item

- hasguestroom: Menunjukkan apakah properti memiliki kamar tamu (guest room) atau tidak (yes/no).
- price: Harga properti.
- category: Kategori properti (misalnya, Luxury, Middle, dll.).

Data Cleansing & Encoding

- Bagian berikut berisi proses pembersihan data.
- Periksa apakah terdapat missing value dan data duplikat,
- Ubah data kategorik string menjadi numerik.
- Jika jumlah kelas pada data latih tidak seimbang, kalian dapat menggunakan metode oversampling.
- Untuk **Klasifikasi**, pastikan **Harga menjadi target** dan **kolom Kategori dihapus**.

```
print("#" * 50)
print("Informasi Umum tentang DataFrame:")
print("#" * 50)
properti.info()
print("\n")

print("#" * 50)
print("Missing Values per Column:")
print("#" * 50)
print(properti.isnull().sum())
print("\n")

print("#" * 50)
print("Jumlah Baris Duplikat:")
print("#" * 50)
print(properti.duplicated().sum())
print("\n")

if properti.duplicated().sum() > 0:
    print("#" * 50)
    print("Baris Duplikat:")
    print("#" * 50)
    print(properti[properti.duplicated()])
else:
    print("#" * 50)
    print("Tidak ada baris duplikat.")
    print("#" * 50)
```

```
##################################################
Informasi Umum tentang DataFrame:
##################################################
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 18 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
```

```
 0   squaremeters       10000 non-null  int64
 1   numberofrooms      10000 non-null  int64
 2   hasyard            10000 non-null  object
 3   haspool            10000 non-null  object
 4   floors             10000 non-null  int64
 5   citycode           10000 non-null  int64
 6   citypartrange      10000 non-null  int64
 7   numprevowners      10000 non-null  int64
 8   made               10000 non-null  int64
 9   isnewbuilt         10000 non-null  object
 10  hasstormprotector  10000 non-null  object
 11  basement           10000 non-null  int64
 12  attic              10000 non-null  int64
 13  garage             10000 non-null  int64
 14  hasstorageroom     10000 non-null  object
 15  hasguestroom       10000 non-null  int64
 16  price              10000 non-null  float64
 17  category           10000 non-null  object
dtypes: float64(1), int64(11), object(6)
memory usage: 1.4+ MB


##################################################
Missing Values per Column:
##################################################
squaremeters         0
numberofrooms        0
hasyard              0
haspool              0
floors               0
citycode             0
citypartrange        0
numprevowners        0
made                 0
isnewbuilt           0
hasstormprotector    0
basement             0
attic                0
garage               0
hasstorageroom       0
hasguestroom         0
price                0
category             0
dtype: int64


##################################################
Jumlah Baris Duplikat:
##################################################
0
```

```
##################################################
Tidak ada baris duplikat.
##################################################
```

```
pd.set_option('display.float_format', lambda x: '%.5f' % x)

properti.describe()
```


```
{"summary":"{\n  \"name\": \"properti\",\n  \"rows\": 8,\n
\"fields\": [\n    {\n      \"column\": \"squaremeters\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
33370.682672584044,\n        \"min\": 89.0,\n        \"max\":
99999.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
49870.1312,\n          50105.5,\n          10000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"numberofrooms\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3518.990372256432,\n        \"min\": 1.0,\n        \"max\": 10000.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          50.3584,\
n          50.0,\n          10000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"floors\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
3518.9414356189227,\n        \"min\": 1.0,\n        \"max\": 10000.0,\
n        \"num_unique_values\": 8,\n        \"samples\": [\n
50.2763,\n          50.0,\n          10000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"citycode\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
33573.27314811567,\n        \"min\": 3.0,\n        \"max\": 99953.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
50225.4861,\n          50693.0,\n          10000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"citypartrange\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3533.748026680069,\n        \"min\": 1.0,\n        \"max\": 10000.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          5.5101,\n
5.0,\n          10000.0\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"numprevowners\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 3533.748218032391,\n
\"min\": 1.0,\n        \"max\": 10000.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          5.5217,\n
5.0,\n          10000.0\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"made\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 3009.508553763657,\n        \"min\":
9.308089589340048,\n        \"max\": 10000.0,\n
```

\"num_unique_values\": 8,\n   \"samples\": [\n   2005.4885,\n   2005.5,\n   10000.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"basement\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 3597.587561948309,\n   \"min\": 0.0,\n   \"max\": 10000.0,\n   \"num_unique_values\": 7,\n   \"samples\": [\n   10000.0,\n   5033.1039,\n   5092.5\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"attic\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 3604.1691191516716,\n   \"min\": 1.0,\n   \"max\": 10000.0,\n   \"num_unique_values\": 7,\n   \"samples\": [\n   10000.0,\n   5028.0106,\n   5045.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"garage\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 3367.2965190625278,\n   \"min\": 100.0,\n   \"max\": 10000.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   553.1212,\n   554.0,\n   10000.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"hasguestroom\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 3533.859948946828,\n   \"min\": 0.0,\n   \"max\": 10000.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   4.9946,\n   5.0,\n   10000.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"price\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 3491484.2164380853,\n   \"min\": 10000.0,\n   \"max\": 10006771.2,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   4993447.52575,\n   5016180.3,\n   10000.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
df_properti = properti.copy()
df_properti.head()
df_properti.columns
```

```
Index(['squaremeters', 'numberofrooms', 'hasyard', 'haspool', 'floors',
       'citycode', 'citypartrange', 'numprevowners', 'made', 'isnewbuilt',
       'hasstormprotector', 'basement', 'attic', 'garage', 'hasstorageroom',
       'hasguestroom', 'price', 'category'],
      dtype='object')
```

```
X = df_properti.drop(columns=['category'], axis=1)
y = df_properti.price
```

```
X_trainReg, X_testReg, y_trainReg, y_testReg = train_test_split(X, y,
test_size=0.30, random_state=77)
print(f"Shape of X_train: {X_trainReg.shape}")
print(f"Shape of X_test: {X_testReg.shape}")


print(X.columns)
```

```
Shape of X_train: (7000, 17)
Shape of X_test: (3000, 17)
Index(['squaremeters', 'numberofrooms', 'hasyard', 'haspool',
'floors',
       'citycode', 'citypartrange', 'numprevowners', 'made',
'isnewbuilt',
       'hasstormprotector', 'basement', 'attic', 'garage',
'hasstorageroom',
       'hasguestroom', 'price'],
      dtype='object')
```

Cek jumlah kelas

```
columns_to_check = ['squaremeters', 'numberofrooms', 'hasyard',
'haspool', 'floors',
                    'citycode', 'citypartrange', 'numprevowners',
'made', 'isnewbuilt',
                    'hasstormprotector', 'basement', 'attic',
'garage', 'hasstorageroom',
                    'hasguestroom', 'price', 'category']

for col in columns_to_check:
    print("#" * 50)
    print(f"Distribusi kelas untuk kolom: {col}")
    print("#" * 50)

    print(properti[col].value_counts())

    if properti[col].dtype in ['float64', 'int64']:
        print("\nStatistik Deskriptif:")
        print(properti[col].describe())

    print("\n")
```

```
##################################################
Distribusi kelas untuk kolom: squaremeters
##################################################
squaremeters
47831    3
36842    3
96526    3
```

```
79770    3
16006    3
        ..
72467    1
74052    1
86122    1
77429    1
79652    1
Name: count, Length: 9483, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean    49870.13120
std     28774.37535
min        89.00000
25%     25098.50000
50%     50105.50000
75%     74609.75000
max     99999.00000
Name: squaremeters, dtype: float64


##################################################
Distribusi kelas untuk kolom: numberofrooms
##################################################
numberofrooms
54    129
4     120
22    119
47    118
3     116
     ...
6      85
31     84
34     84
40     82
9      75
Name: count, Length: 100, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean       50.35840
std        28.81670
min         1.00000
25%        25.00000
50%        50.00000
75%        75.00000
max       100.00000
Name: numberofrooms, dtype: float64
```

```
##################################################
Distribusi kelas untuk kolom: hasyard
##################################################
hasyard
yes    5087
no     4913
Name: count, dtype: int64


##################################################
Distribusi kelas untuk kolom: haspool
##################################################
haspool
no     5032
yes    4968
Name: count, dtype: int64


##################################################
Distribusi kelas untuk kolom: floors
##################################################
floors
97     126
55     122
77     117
28     116
3      116
      ...
15      83
48      83
74      83
100     82
92      75
Name: count, Length: 100, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean       50.27630
std        28.88917
min         1.00000
25%        25.00000
50%        50.00000
75%        76.00000
max       100.00000
Name: floors, dtype: float64


##################################################
Distribusi kelas untuk kolom: citycode
```

```
##################################################
citycode
79444    3
37363    3
83194    3
95054    3
96283    3
        ..
3920     1
57059    1
47825    1
39656    1
50551    1
Name: count, Length: 9509, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean    50225.48610
std     29006.67580
min         3.00000
25%     24693.75000
50%     50693.00000
75%     75683.25000
max     99953.00000
Name: citycode, dtype: float64


##################################################
Distribusi kelas untuk kolom: citypartrange
##################################################
citypartrange
8     1035
5     1031
10    1004
4     1001
3      999
9      997
1      994
2      990
7      984
6      965
Name: count, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean        5.51010
std         2.87202
min         1.00000
25%         3.00000
50%         5.00000
```

```
75%          8.00000
max         10.00000
Name: citypartrange, dtype: float64


##################################################
Distribusi kelas untuk kolom: numprevowners
##################################################
numprevowners
4     1043
9     1036
5     1036
6     1011
10     999
3      991
2      987
7      974
8      971
1      952
Name: count, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean        5.52170
std         2.85667
min         1.00000
25%         3.00000
50%         5.00000
75%         8.00000
max        10.00000
Name: numprevowners, dtype: float64


##################################################
Distribusi kelas untuk kolom: made
##################################################
made
1992    356
2013    352
2020    336
2018    334
2001    332
2003    332
1996    327
2009    324
1991    324
2019    321
2011    321
1993    320
1998    318
```

```
1990    317
1994    312
2014    312
2016    307
2004    307
2015    305
2012    305
2021    304
2007    302
2008    302
2005    296
2006    296
1997    296
2000    295
1999    293
2010    291
2002    290
2017    288
1995    285
Name: count, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean     2005.48850
std         9.30809
min      1990.00000
25%      1997.00000
50%      2005.50000
75%      2014.00000
max      2021.00000
Name: made, dtype: float64


##################################################
Distribusi kelas untuk kolom: isnewbuilt
##################################################
isnewbuilt
old    5009
new    4991
Name: count, dtype: int64


##################################################
Distribusi kelas untuk kolom: hasstormprotector
##################################################
hasstormprotector
no     5001
yes    4999
Name: count, dtype: int64
```

```
################################################
Distribusi kelas untuk kolom: basement
################################################
basement
1421    6
2192    6
4170    6
6899    6
9186    5
       ..
2188    1
4105    1
780     1
7542    1
4244    1
Name: count, Length: 6352, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean     5033.10390
std      2876.72954
min         0.00000
25%      2559.75000
50%      5092.50000
75%      7511.25000
max     10000.00000
Name: basement, dtype: float64


################################################
Distribusi kelas untuk kolom: attic
################################################
attic
3127    7
5017    6
6556    6
1581    6
8481    6
       ..
5120    1
3070    1
2120    1
9082    1
7174    1
Name: count, Length: 6267, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean     5028.01060
```

```
std       2894.33221
min          1.00000
25%       2512.00000
50%       5045.00000
75%       7540.50000
max      10000.00000
Name: attic, dtype: float64


##################################################
Distribusi kelas untuk kolom: garage
##################################################
garage
253    24
955    21
984    20
765    20
946    20
       ..
578     4
887     3
483     3
589     2
282     2
Name: count, Length: 901, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean      553.12120
std       262.05017
min       100.00000
25%       327.75000
50%       554.00000
75%       777.25000
max      1000.00000
Name: garage, dtype: float64


##################################################
Distribusi kelas untuk kolom: hasstorageroom
##################################################
hasstorageroom
yes    5030
no     4970
Name: count, dtype: int64


##################################################
Distribusi kelas untuk kolom: hasguestroom
##################################################
```

```
hasguestroom
2     942
10    926
9     916
0     914
8     913
4     911
1     910
3     906
6     904
7     884
5     874
Name: count, dtype: int64

Statistik Deskriptif:
count   10000.00000
mean        4.99460
std         3.17641
min         0.00000
25%         2.00000
50%         5.00000
75%         8.00000
max        10.00000
Name: hasguestroom, dtype: float64


##################################################
Distribusi kelas untuk kolom: price
##################################################
price
146708.40000     1
7559081.50000    1
5574642.10000    1
8696869.30000    1
5154055.20000    1
                ..
5446398.10000    1
6315375.70000    1
6441378.00000    1
9390891.90000    1
8410054.60000    1
Name: count, Length: 10000, dtype: int64

Statistik Deskriptif:
count      10000.00000
mean     4993447.52575
std      2877424.10995
min        10313.50000
25%      2516401.95000
50%      5016180.30000
```

```
75%      7469092.45000
max     10006771.20000
Name: price, dtype: float64


##################################################
Distribusi kelas untuk kolom: category
##################################################
category
Basic     4344
Luxury    3065
Middle    2591
Name: count, dtype: int64
```

Kolom category:

- Basic: 4344
- Luxury: 3065
- Middle: 2591

Ini menunjukkan sedikit ketidakseimbangan. Kategori Basic memiliki jumlah yang jauh lebih banyak dibandingkan Luxury dan Middle. Dalam kasus klasifikasi, ini mungkin bisa menyebabkan ketidakseimbangan performa model, terutama jika model lebih cenderung ke kelas yang lebih dominan.

- Mengubah variabel kategori menjadi numerik.
- Membagi data menjadi set pelatihan dan pengujian.

```
cat_cols=['hasyard', 'haspool', 'isnewbuilt',
          'hasstormprotector', 'hasstorageroom']

transformer = make_column_transformer(
    (OneHotEncoder(), cat_cols),
    remainder='passthrough'
)

X_trainReg_enc = transformer.fit_transform(X_trainReg)
X_testReg_enc = transformer.transform(X_testReg)

df_trainReg_enc = pd.DataFrame(X_trainReg_enc,
columns=transformer.get_feature_names_out())
df_testReg_enc = pd.DataFrame(X_testReg_enc,
columns=transformer.get_feature_names_out())

df_trainReg_enc.head(10)
df_testReg_enc.head(10)
```

{"type":"dataframe","variable_name":"df_testReg_enc"}

```
np.set_printoptions(formatter={'float': '{:.2f}'.format})

print(X_trainReg_enc)
```

```
[[0.00 1.00 0.00 ... 112.00 4.00 5419304.80]
 [0.00 1.00 1.00 ... 941.00 7.00 6958375.50]
 [1.00 0.00 0.00 ... 822.00 6.00 5346532.30]
 ...
 [1.00 0.00 1.00 ... 682.00 7.00 8870811.50]
 [1.00 0.00 0.00 ... 156.00 6.00 9372651.70]
 [1.00 0.00 0.00 ... 819.00 0.00 86104.40]]
```

```
from sklearn.model_selection import KFold

kf = KFold(n_splits=5, shuffle=True, random_state=77)

X_folds = []
y_folds = []

for train_index, test_index in kf.split(X_trainReg_enc):
    X_folds.append((X_trainReg_enc[train_index],
X_trainReg_enc[test_index]))
    y_folds.append((y_trainReg.iloc[train_index],
y_trainReg.iloc[test_index]))

print(y_trainReg.unique())
```

```
[5419304.80 6958375.50 5346532.30 ... 8870811.50 9372651.70 86104.40]
```

Random Forest Regressor

```
pipe_RF = Pipeline(steps=[
    ('scale', StandardScaler()),
    ('feature_selection', SelectKBest(score_func=f_regression)),
    ('reg', RandomForestRegressor())
])

param_grid_RF = {
    'reg__n_estimators': [50, 100, 200],
    'reg__max_depth': [None, 10, 20],
    'feature_selection__k': np.arange(1, 20)
}

GSCV_RF = GridSearchCV(pipe_RF, param_grid_RF, cv=5,
scoring='neg_mean_squared_error')
GSCV_RF.fit(X_trainReg_enc, y_trainReg)

print("Best Model:", GSCV_RF.best_estimator_)
print("RF best parameters:", GSCV_RF.best_params_)

RF_predict = GSCV_RF.predict(X_testReg_enc)
```

```
mse_RF = mean_squared_error(y_testReg, RF_predict)
mae_RF = mean_absolute_error(y_testReg, RF_predict)

print("RF MSE:", mse_RF)
print("RF MAE:", mae_RF)
print("RF RMSE:", np.sqrt(mse_RF))

df_results = pd.DataFrame({
    'Actual Price': y_testReg.reset_index(drop=True),
    'RF Predicted Price': RF_predict
})

df_results['Price Difference (RF)'] = df_results['RF Predicted Price']
- df_results['Actual Price']

print(df_results.head())
```

```
/usr/local/lib/python3.10/dist-packages/numpy/ma/core.py:2820:
RuntimeWarning: invalid value encountered in cast
  _data = np.array(data, dtype=dtype, copy=copy,
```

```
Best Model: Pipeline(steps=[('scale', StandardScaler()),
                ('feature_selection',
                 SelectKBest(k=1,
                             score_func=<function f_regression at
0x7cda3e764f70>)),
                ('reg', RandomForestRegressor(n_estimators=200))])
RF best parameters: {'feature_selection__k': 1, 'reg__max_depth':
None, 'reg__n_estimators': 200}
RF MSE: 664402.3924273634
RF MAE: 541.8022686166591
RF RMSE: 815.1088224448092
   Actual Price  RF Predicted Price  Price Difference (RF)
0 6779991.50000       6780244.81150              253.31150
1 7974272.10000       7974190.52650              -81.57350
2 3474105.70000       3473897.71700             -207.98300
3 2340035.80000       2339110.75950             -925.04050
4 4325940.30000       4326350.37500              410.07500
```

```
pipe_RF = Pipeline(steps=[
    ('scale', MinMaxScaler()),
    ('feature_selection', SelectPercentile(score_func=f_regression)),
    ('reg', RandomForestRegressor())
])

param_grid_RF = {
    'reg__n_estimators': [50, 100, 200],
    'reg__max_depth': [None, 10, 20],
    'feature_selection__percentile': np.arange(10, 100, 10)
}
```

```
GSCV_RF = GridSearchCV(pipe_RF, param_grid_RF, cv=5,
scoring='neg_mean_squared_error')
GSCV_RF.fit(X_trainReg_enc, y_trainReg)

print("Best Model:", GSCV_RF.best_estimator_)
print("RF best parameters:", GSCV_RF.best_params_)

RF_predict = GSCV_RF.predict(X_testReg_enc)

mse_RF = mean_squared_error(y_testReg, RF_predict)
mae_RF = mean_absolute_error(y_testReg, RF_predict)

print("RF MSE:", mse_RF)
print("RF MAE:", mae_RF)
print("RF RMSE:", np.sqrt(mse_RF))

df_results = pd.DataFrame({
    'Actual Price': y_testReg.reset_index(drop=True),
    'RF Predicted Price': RF_predict
})

df_results['Price Difference (RF)'] = df_results['RF Predicted Price']
- df_results['Actual Price']

print(df_results.head())

Best Model: Pipeline(steps=[('scale', MinMaxScaler()),
                ('feature_selection',
                 SelectPercentile(percentile=20,
                                  score_func=<function f_regression at
0x7cda3e764f70>)),
                ('reg', RandomForestRegressor(max_depth=20,
n_estimators=200))])
RF best parameters: {'feature_selection__percentile': 20,
'reg__max_depth': 20, 'reg__n_estimators': 200}
RF MSE: 15580862.92126373
RF MAE: 3144.104324833377
RF RMSE: 3947.260179068987
   Actual Price  RF Predicted Price  Price Difference (RF)
0 6779991.50000       6780476.10250              484.60250
1 7974272.10000       7973660.89100             -611.20900
2 3474105.70000       3475715.29500             1609.59500
3 2340035.80000       2335051.64250            -4984.15750
4 4325940.30000       4327610.00500             1669.70500
```

Lasso Regression

```
pipe_Lasso = Pipeline(steps=[
    ('scale', StandardScaler()),
```

```
    ('feature_selection', SelectKBest(score_func=f_regression)),
    ('reg', Lasso())
])

param_grid_Lasso = {
    'reg__alpha': [0.01, 0.1, 1, 10],
    'feature_selection__k': np.arange(1, 20)
}

GSCV_Lasso = GridSearchCV(pipe_Lasso, param_grid_Lasso, cv=5,
scoring='neg_mean_squared_error')
GSCV_Lasso.fit(X_trainReg_enc, y_trainReg)

print("Best Model:", GSCV_Lasso.best_estimator_)
print("Lasso best parameters:", GSCV_Lasso.best_params_)

Lasso_predict = GSCV_Lasso.predict(X_testReg_enc)

mse_Lasso = mean_squared_error(y_testReg, Lasso_predict)
mae_Lasso = mean_absolute_error(y_testReg, Lasso_predict)

print("Lasso MSE:", mse_Lasso)
print("Lasso MAE:", mae_Lasso)
print("Lasso RMSE:", np.sqrt(mse_Lasso))

df_results = pd.DataFrame({
    'Actual Price': y_testReg.reset_index(drop=True),
    'Lasso Predicted Price': Lasso_predict
})

df_results['Price Difference (Lasso)'] = df_results['Lasso Predicted
Price'] - df_results['Actual Price']

print(df_results.head())
```

```
Best Model: Pipeline(steps=[('scale', StandardScaler()),
                ('feature_selection',
                 SelectKBest(k=1,
                             score_func=<function f_regression at
0x7cda3e764f70>)),
                ('reg', Lasso(alpha=0.01))])
Lasso best parameters: {'feature_selection__k': 1, 'reg__alpha': 0.01}
Lasso MSE: 0.0001009410883096407
Lasso MAE: 0.008682589079740259
Lasso RMSE: 0.010046944227457456
   Actual Price  Lasso Predicted Price  Price Difference (Lasso)
0 6779991.50000          6779991.49384                  -0.00616
1 7974272.10000          7974272.08968                  -0.01032
2 3474105.70000          3474105.70534                   0.00534
```

```
3 2340035.80000          2340035.80929                     0.00929
4 4325940.30000          4325940.30238                     0.00238

pipe_Lasso = Pipeline(steps=[
    ('scale', MinMaxScaler()),
    ('feature_selection', SelectPercentile(score_func=f_regression)),
    ('reg', Lasso())
])

param_grid_Lasso = {
    'reg__alpha': [0.01, 0.1, 1, 10],
    'feature_selection__percentile': np.arange(10, 101, 10)
}

GSCV_Lasso = GridSearchCV(pipe_Lasso, param_grid_Lasso, cv=5,
scoring='neg_mean_squared_error')
GSCV_Lasso.fit(X_trainReg_enc, y_trainReg)

print("Best Model:", GSCV_Lasso.best_estimator_)
print("Lasso best parameters:", GSCV_Lasso.best_params_)

Lasso_predict = GSCV_Lasso.predict(X_testReg_enc)

mse_Lasso = mean_squared_error(y_testReg, Lasso_predict)
mae_Lasso = mean_absolute_error(y_testReg, Lasso_predict)

print("Lasso MSE:", mse_Lasso)
print("Lasso MAE:", mae_Lasso)
print("Lasso RMSE:", np.sqrt(mse_Lasso))

df_results = pd.DataFrame({
    'Actual Price': y_testReg.reset_index(drop=True),
    'Lasso Predicted Price': Lasso_predict
})

df_results['Price Difference (Lasso)'] = df_results['Lasso Predicted
Price'] - df_results['Actual Price']

print(df_results.head())

Best Model: Pipeline(steps=[('scale', MinMaxScaler()),
                ('feature_selection',
                 SelectPercentile(percentile=100,
                                  score_func=<function f_regression at
0x7cda3e764f70>)),
                ('reg', Lasso(alpha=1))])
Lasso best parameters: {'feature_selection__percentile': 100,
'reg__alpha': 1}
Lasso MSE: 3711244.0437926566
Lasso MAE: 1492.4837925308918
Lasso RMSE: 1926.4589390362455
```

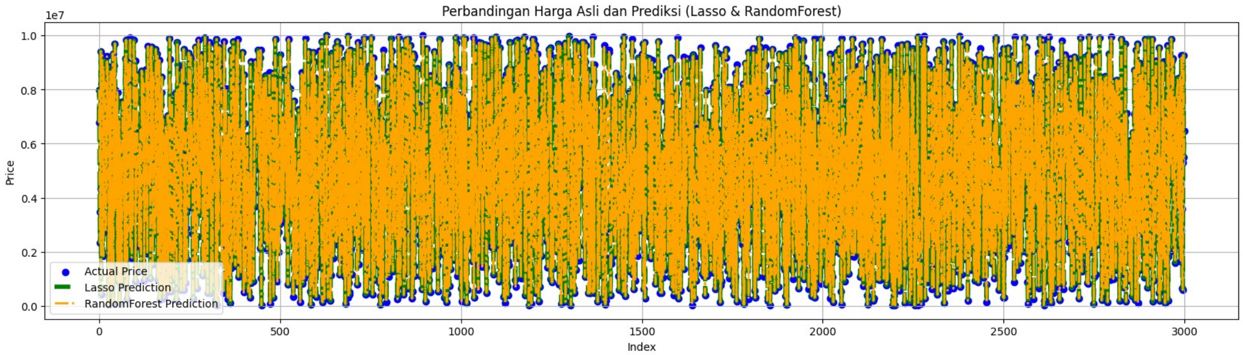
```
   Actual Price  Lasso Predicted Price  Price Difference (Lasso)
0 6779991.50000          6780327.14803                 335.64803
1 7974272.10000          7974521.36619                 249.26619
2 3474105.70000          3476058.50519                1952.80519
3 2340035.80000          2338440.51364               -1595.28636
4 4325940.30000          4326897.63352                 957.33352
```

```
df_results = pd.DataFrame({'Actual Price':
y_testReg.reset_index(drop=True)})
df_results['Lasso Prediction'] = Lasso_predict
df_results['RandomForest Prediction'] = RF_predict

df_results['Lasso Price Difference'] = df_results['Actual Price'] -
df_results['Lasso Prediction']
df_results['RandomForest Price Difference'] = df_results['Actual
Price'] - df_results['RandomForest Prediction']

print(df_results.head())
```

```
   Actual Price  Lasso Prediction  RandomForest Prediction  \
0 6779991.50000     6780327.14803            6780476.10250
1 7974272.10000     7974521.36619            7973660.89100
2 3474105.70000     3476058.50519            3475715.29500
3 2340035.80000     2338440.51364            2335051.64250
4 4325940.30000     4326897.63352            4327610.00500

   Lasso Price Difference  RandomForest Price Difference
0              -335.64803                     -484.60250
1              -249.26619                      611.20900
2             -1952.80519                    -1609.59500
3              1595.28636                     4984.15750
4              -957.33352                    -1669.70500
```

```
plt.figure(figsize=(20, 5))
data_len = range(len(y_testReg))

plt.scatter(data_len, df_results['Actual Price'], label="Actual
Price", color="blue")
plt.plot(data_len, df_results['Lasso Prediction'], label="Lasso
Prediction", color="green", linewidth=4, linestyle="dashed")
plt.plot(data_len, df_results['RandomForest Prediction'],
label="RandomForest Prediction", color="orange", linewidth=2,
linestyle="-.")

plt.title("Perbandingan Harga Asli dan Prediksi (Lasso &
RandomForest)")
plt.xlabel("Index")
plt.ylabel("Price")
plt.legend()
plt.grid(True)
```

```
plt.show()
```



Berdasarkan hasil evaluasi untuk model Random Forest dan Lasso Regression yang telah dilatih dengan berbagai skala dan fitur yang berbeda, berikut adalah analisis performa model berdasarkan metrik Mean Squared Error (MSE), Mean Absolute Error (MAE), dan Root Mean Squared Error (RMSE):

1. **Mean Squared Error (MSE)**

- **Random Forest (Pipeline 1 - StandardScaler, SelectKBest):** 664,402
- **Random Forest (Pipeline 2 - MinMaxScaler, SelectPercentile):** 15,580,862
- **Lasso (Pipeline 1 - StandardScaler, SelectKBest):** 0.0001
- **Lasso (Pipeline 2 - MinMaxScaler, SelectPercentile):** 3,711,244

Penjelasan: MSE mengukur rata-rata kuadrat kesalahan prediksi. Semakin kecil nilainya, semakin akurat model. **Lasso Regression (Pipeline 1)** memiliki MSE yang sangat rendah (0.0001), menunjukkan bahwa prediksi model ini hampir mendekati harga aktual. Sebaliknya, **Random Forest (Pipeline 2)** memiliki MSE tertinggi, menunjukkan performa yang lebih buruk dibanding model lainnya.

2. **Mean Absolute Error (MAE)**

- **Random Forest (Pipeline 1):** 541.80
- **Random Forest (Pipeline 2):** 3,144.10
- **Lasso (Pipeline 1):** 0.0087
- **Lasso (Pipeline 2):** 1,492.48

Penjelasan: MAE mengukur rata-rata kesalahan absolut antara prediksi dan nilai aktual. **Lasso Regression (Pipeline 1)** kembali menunjukkan performa terbaik dengan MAE terendah, menunjukkan bahwa prediksinya sangat dekat dengan nilai aktual. **Random Forest (Pipeline 2)**, sebaliknya, memiliki MAE yang lebih besar, menunjukkan kesalahan prediksi yang lebih signifikan.

3. **Root Mean Squared Error (RMSE)**

- **Random Forest (Pipeline 1):** 815.11
- **Random Forest (Pipeline 2):** 3,947.26
- **Lasso (Pipeline 1):** 0.01005

- **Lasso (Pipeline 2):** 1,926.46

Penjelasan: RMSE memberikan bobot lebih besar pada kesalahan besar karena menggunakan kuadrat dari kesalahan. **Lasso Regression (Pipeline 1)** menghasilkan RMSE terendah, yang menunjukkan bahwa model ini lebih baik dalam mengurangi kesalahan besar dalam prediksi harga. **Random Forest (Pipeline 2)** memiliki RMSE tertinggi, menunjukkan bahwa model ini lebih rentan terhadap kesalahan besar.

4. **Perbandingan Prediksi Harga**

- **Random Forest (Pipeline 1 - StandardScaler, SelectKBest):**
  - Harga Aktual: 6,779,991.50, Prediksi: 6,780,244.81 (Selisih: 253.31)
  - Harga Aktual: 7,974,272.10, Prediksi: 7,974,190.53 (Selisih: -81.57)
- **Random Forest (Pipeline 2 - MinMaxScaler, SelectPercentile):**
  - Harga Aktual: 6,779,991.50, Prediksi: 6,780,476.10 (Selisih: 484.60)
  - Harga Aktual: 7,974,272.10, Prediksi: 7,973,660.89 (Selisih: -611.21)
- **Lasso Regression (Pipeline 1 - StandardScaler, SelectKBest):**
  - Harga Aktual: 6,779,991.50, Prediksi: 6,779,991.49 (Selisih: -0.00616)
  - Harga Aktual: 7,974,272.10, Prediksi: 7,974,272.09 (Selisih: -0.01032)

Penjelasan: Lasso Regression (Pipeline 1) menghasilkan prediksi yang paling akurat, dengan selisih yang sangat kecil (hanya 0.00616 hingga 0.01032). Sebaliknya, **Random Forest (Pipeline 2)** menunjukkan selisih prediksi yang lebih besar, terutama pada contoh kedua dengan selisih sebesar -611.21.

5. **Kesimpulan**

Lasso Regression (Pipeline 1 - StandardScaler, SelectKBest) terbukti sebagai model terbaik berdasarkan:

- **Mean Squared Error (MSE), Mean Absolute Error (MAE), dan Root Mean Squared Error (RMSE)** yang paling rendah.
- **Prediksi harga** yang paling mendekati nilai aktual, dengan selisih yang sangat kecil.

Random Forest Regression (Pipeline 1 - StandardScaler, SelectKBest) juga menunjukkan performa yang baik, tetapi sedikit kurang akurat dibandingkan Lasso. **Random Forest Regression (Pipeline 2 - MinMaxScaler, SelectPercentile)** memiliki performa yang paling rendah di antara model yang diuji.

Rekomendasi:

Model **Lasso Regression (Pipeline 1)** adalah pilihan terbaik untuk regresi dalam kasus ini, karena memberikan prediksi yang paling akurat dan error paling kecil.

Model Terbaik dari perbandingan Algoritme

Diantara Perbandingan 2 Notebook masing masing dari perbandingan untuk Regresi tersebut memiliki model terbaik yaitu, dari Notebook pertama ialah Ridge Regression (Pipeline 1) dan Notebook kedua adalah Lasso Regression (Pipeline 1) . Sekarang untuk menentukan Model terbaik untuk Regresi, perlu melihat perbandingan hasil dibawah:

1. **Mean Squared Error (MSE)**

- **Lasso Regression (Pipeline 1)**: 0.0001 (Notebook pertama)
- **Ridge Regression (Pipeline 1)**: 3,632,947 (Notebook kedua)

Lasso Regression (Pipeline 1) dari Notebook pertama memiliki nilai MSE yang jauh lebih kecil dibandingkan Ridge Regression (Pipeline 1) dari Notebook kedua, menandakan bahwa model ini lebih unggul dalam meminimalkan kesalahan kuadrat rata-rata.

2. **Mean Absolute Error (MAE)**

- **Lasso Regression (Pipeline 1)**: 0.0087 (Notebook pertama)
- **Ridge Regression (Pipeline 1)**: 1,482.79 (Notebook kedua)

Lasso Regression (Pipeline 1) dari Notebook pertama juga unggul dalam hal MAE, dengan kesalahan rata-rata yang sangat kecil dibandingkan Ridge Regression dari Notebook kedua.

3. **Root Mean Squared Error (RMSE)**

- **Lasso Regression (Pipeline 1)**: 0.01005 (Notebook pertama)
- **Ridge Regression (Pipeline 1)**: 1,906.03 (Notebook kedua)

RMSE dari **Lasso Regression (Pipeline 1)** dari Notebook pertama jauh lebih rendah dibandingkan Ridge Regression dari Notebook kedua, menunjukkan bahwa model ini lebih baik dalam mengurangi dampak dari kesalahan besar dalam prediksi.

4. **Perbandingan Prediksi Harga**

- **Lasso Regression (Pipeline 1)** dari Notebook pertama: Prediksi sangat akurat dengan selisih prediksi hampir nol (0.00616 hingga 0.01032).
- **Ridge Regression (Pipeline 1)** dari Notebook kedua: Selisih prediksi lebih besar (2,102.03 hingga 3,449.92).

Lasso Regression (Pipeline 1) dari Notebook pertama kembali unggul karena selisih prediksinya sangat kecil, hampir mendekati harga aktual.

5. **Kesimpulan Akhir**

Secara keseluruhan, **Lasso Regression (Pipeline 1 - StandardScaler, SelectKBest)** dari **Notebook pertama** adalah model terbaik untuk regresi berdasarkan:

- MSE, MAE, dan RMSE yang paling rendah.
- Prediksi harga yang paling mendekati nilai aktual, dengan selisih yang sangat kecil.

Meskipun **Ridge Regression (Pipeline 1)** dari **Notebook kedua** menunjukkan performa yang lebih baik dibanding model lainnya dalam kelompok tersebut, **Lasso Regression (Pipeline 1)** dari **Notebook pertama** tetap yang terbaik secara keseluruhan untuk prediksi dalam kasus ini.

```
import pickle

with open('/content/drive/MyDrive/Colab
Notebooks/LR_Properti_model.pkl', 'wb') as r:
    pickle.dump(GSCV_RF, r)
```

```
print("Model Lasso Regression berhasil disimpan")
```

```
Model Lasso Regression berhasil disimpan
```