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

from streamlit_option_menu import option_menu

with st.sidebar:
    selected = option_menu("Tugas UTS 2024/2025",
                           ['Klasifikasi','Regresi'],default_index=0)

if selected == 'Klasifikasi':
    model_path = r'E:\Rendy\Kampus\Semester 5\MLDL\UTS'
    model = os.path.join(model_path,'BestModel_CLF_GBT_Tensorflow.pkl')

    with open(model, 'rb') as file:
        model = pickle.load(file)

    st.title('Klasifikasi')
    st.write('Masukkan Input-input berikut ini')
    luas = st.slider('Luas', 0, 99999)
    jumlah_ruangan = st.slider('Jumlah Ruangan', 0, 100)
    halaman = st.selectbox('Halaman', ['Ada', 'Tidak Ada'])
    kolam = st.selectbox('Kolam', ['Ada', 'Tidak Ada'])
    lantai = st.slider('Lantai', 0, 100)
    kode_lokasi = st.number_input('Kode Lokasi', 0, 99953)
    ekslusifitas_kawasan = st.number_input('Ekslusifitas Kawasan', 0, 10)
    jumlah_pemilik_sebelumnya = st.number_input('Jumlah Pemilik Sebelumnya', 0, 10)
    tahun_pembangunan = st.slider('Tahun Pembangunan', 1990, 2021)
    baru = st.selectbox('Apakah Baru', ['Baru', 'Tidak Baru'])
    punyaStormProctector = st.selectbox('Punya Storm Protector', ['Punya', 'Tidak Punya'])
    luas_basement = st.slider('Luas Basement', 0, 10000)
    luas_loteng = st.slider('Luas Loteng', 0, 10000)
    luas_garasi = st.slider('Luas Garasi', 0, 10000)
    gudang = st.selectbox('Gudang', ['Ada', 'Tidak Ada'])
    jumlah_kamar_tamu = st.number_input('Jumlah Kamar Tamu', 0, 10)

    if halaman == 'Ada':
        halaman_ada = 1
        halaman_tidak_ada = 0
    elif halaman == 'Tidak Ada':
        halaman_ada = 0
        halaman_tidak_ada = 1

    if kolam == 'Ada':
        kolam_ada = 1
        kolam_tidak_ada = 0
    elif kolam == 'Tidak Ada':
        kolam_ada = 0
        kolam_tidak_ada = 1

    if baru == 'Baru':
        baru_baru = 1
        baru_tidak_baru = 0
    elif baru == 'Tidak Baru':
        baru_baru = 0
        baru_tidak_baru = 1

    if punyaStormProctector == 'Punya':
        punya_storm_protector_punya = 1
        punya_storm_protector_tidak_punya = 0
    elif punyaStormProctector == 'Tidak Punya':
        punya_storm_protector_punya = 0
        punya_storm_protector_tidak_punya = 1

    if gudang == 'Ada':
        gudang_ada = 1
        gudang_tidak_ada = 0
    elif gudang == 'Tidak Ada':
        gudang_ada = 0
        gudang_tidak_ada = 1

    input_data = [[luas, jumlah_ruangan, halaman_ada, halaman_tidak_ada, kolam_ada, kolam_tidak_ada, lantai, kode_lokasi, ekslusifitas_kawasan, jumlah_pemilik_



    if st.button('Prediksi'):
        model = model.predict(input_data)
        st.write('Kategori yang diprediksi adalah', model)

elif selected == 'Regresi':

    model_path = r'E:\Rendy\Kampus\Semester 5\MLDL\UTS'
    model = os.path.join(model_path,'BestModel_REG_Lasso_Tensorflow.pkl')

    with open(model, 'rb') as file:
        model = pickle.load(file)

    st.title('Regresi')
    st.write('Masukkan Input-input berikut ini')
    luas = st.slider('Luas', 0, 99999)
    jumlah_ruangan = st.slider('Jumlah Ruangan', 0, 100)
    halaman = st.selectbox('Halaman', ['Ada', 'Tidak Ada'])
    kolam = st.selectbox('Kolam', ['Ada', 'Tidak Ada'])
    lantai = st.slider('Lantai', 0, 100)
    kode_lokasi = st.number_input('Kode Lokasi', 0, 99953)
    ekslusifitas_kawasan = st.number_input('Ekslusifitas Kawasan', 0, 10)
    jumlah_pemilik_sebelumnya = st.number_input('Jumlah Pemilik Sebelumnya', 0, 10)
    tahun_pembangunan = st.slider('Tahun Pembangunan', 1990, 2021)
    baru = st.selectbox('Apakah Baru', ['Baru', 'Tidak Baru'])
    punyaStormProctector = st.selectbox('Punya Storm Protector', ['Punya', 'Tidak Punya'])
    luas_basement = st.slider('Luas Basement', 0, 10000)
    luas_loteng = st.slider('Luas Loteng', 0, 10000)
    luas_garasi = st.slider('Luas Garasi', 0, 10000)
    gudang = st.selectbox('Gudang', ['Ada', 'Tidak Ada'])
    jumlah_kamar_tamu = st.number_input('Jumlah Kamar Tamu', 0, 10)
```

```
    if halaman == 'Ada':
        halaman_ada = 1
        halaman_tidak_ada = 0
    elif halaman == 'Tidak Ada':
        halaman_ada = 0
        halaman_tidak_ada = 1

    if kolam == 'Ada':
        kolam_ada = 1
        kolam_tidak_ada = 0
    elif kolam == 'Tidak Ada':
        kolam_ada = 0
        kolam_tidak_ada = 1

    if baru == 'Baru':
        baru_baru = 1
        baru_tidak_baru = 0
    elif baru == 'Tidak Baru':
        baru_baru = 0
        baru_tidak_baru = 1

    if punyaStormProctector == 'Punya':
        punya_storm_protector_punya = 1
        punya_storm_protector_tidak_punya = 0
    elif punyaStormProctector == 'Tidak Punya':
        punya_storm_protector_punya = 0
        punya_storm_protector_tidak_punya = 1

    if gudang == 'Ada':
        gudang_ada = 1
        gudang_tidak_ada = 0
    elif gudang == 'Tidak Ada':
        gudang_ada = 0
        gudang_tidak_ada = 1

    input_data = [[luas, jumlah_ruangan, halaman_ada, halaman_tidak_ada, kolam_ada, kolam_tidak_ada, lantai, kode_lokasi, ekslusifitas_kawasan, jumlah_pemilik_



    if st.button('Prediksi'):
        model = model.predict(input_data)
        st.write('Harga yang diprediksi adalah', model)
```

# fikasi-b-tensorflow-lr-vs-rf-rendy

October 25, 2024

[3]:
```
#semoga kali ini jadi

import pandas as pd
import numpy as np

#load data
df_prop = pd.read_csv('Dataset UTS_Gasal 2425.csv')
df_prop.head(10)
```

[3]:
```
   squaremeters  numberofrooms hasyard haspool  floors  citycode  \
0         75523              3      no     yes      63      9373
1         55712             58      no     yes      19     34457
2         86929            100     yes      no      11     98155
3         51522              3      no      no      61      9047
4         96470             74     yes      no      21     92029
5         79770              3      no     yes      69     54812
6         75985             60     yes      no      67      6517
7         64169             88      no     yes       6     61711
8         92383             12      no      no      78     71982
9         95121             46      no     yes       3      9382

   citypartrange  numprevowners  made isnewbuilt hasstormprotector  basement  \
0              3              8  2005        old               yes      4313
1              6              8  2021        old                no      2937
2              3              4  2003        new                no      6326
3              8              3  2012        new               yes       632
4              4              2  2011        new               yes      5414
5             10              5  2018        old               yes      8871
6              6              9  2009        new               yes      4878
7              3              9  2011        new               yes      3054
8              3              7  2000        old                no      7507
9              7              9  1994        old                no       615

   attic  garage hasstorageroom  hasguestroom      price category
0   9005     956             no             7  7559081.5   Luxury
1   8852     135            yes             9  5574642.1   Middle
2   4748     654             no            10  8696869.3   Luxury
```

```
3   5792     807             yes             5  5154055.2   Middle
4   1172     716             yes             9  9652258.1   Luxury
5   7117     240              no             7  7986665.8   Luxury
6    281     384             yes             5  7607322.9   Luxury
7    129     726              no             9  6420823.1   Middle
8   9056     892             yes             1  9244344.0   Luxury
9   1221     328              no            10  9515440.4   Luxury
```

[4]:
```
df_prop.describe()
```

[4]:
```
       squaremeters  numberofrooms        floors      citycode  citypartrange  \
count   10000.00000   10000.000000  10000.000000  10000.000000   10000.000000
mean    49870.13120      50.358400     50.276300  50225.486100       5.510100
std     28774.37535      28.816696     28.889171  29006.675799       2.872024
min        89.00000       1.000000      1.000000      3.000000       1.000000
25%     25098.50000      25.000000     25.000000  24693.750000       3.000000
50%     50105.50000      50.000000     50.000000  50693.000000       5.000000
75%     74609.75000      75.000000     76.000000  75683.250000       8.000000
max     99999.00000     100.000000    100.000000  99953.000000      10.000000

       numprevowners         made      basement        attic       garage  \
count   10000.000000  10000.00000  10000.000000  10000.00000  10000.00000
mean        5.521700   2005.48850   5033.103900   5028.01060    553.12120
std         2.856667      9.30809   2876.729545   2894.33221    262.05017
min         1.000000   1990.00000      0.000000      1.00000    100.00000
25%         3.000000   1997.00000   2559.750000   2512.00000    327.75000
50%         5.000000   2005.50000   5092.500000   5045.00000    554.00000
75%         8.000000   2014.00000   7511.250000   7540.50000    777.25000
max        10.000000   2021.00000  10000.000000  10000.00000   1000.00000

       hasguestroom         price
count   10000.00000  1.000000e+04
mean        4.99460  4.993448e+06
std         3.17641  2.877424e+06
min         0.00000  1.031350e+04
25%         2.00000  2.516402e+06
50%         5.00000  5.016180e+06
75%         8.00000  7.469092e+06
max        10.00000  1.000677e+07
```

[5]:
```
df_prop2 = df_prop.drop('price', axis=1)
df_prop2['category'].value_counts()
```

[5]:
```
category
Basic     4344
Luxury    3065
Middle    2591
```

```
Name: count, dtype: int64
```

[6]:
```
print("data null\n", df_prop2.isnull().sum())
print("data kosong\n", df_prop2.empty)
print("data nan\n", df_prop2.isna().sum())
```

```
data null
 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
category             0
dtype: int64
data kosong
 False
data nan
 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
category             0
dtype: int64
```

[7]:
```
print("Sebelum drop missing value \n",df_prop2.shape)
df_prop2= df_prop2.dropna(how='any',inplace=False)
print("Sesudah drop missing value \n",df_prop2.shape)
```

```
Sebelum drop missing value
 (10000, 17)
Sesudah drop missing value
 (10000, 17)
```

[8]:
```
print("sebelum cek duplikat \n",df_prop2.shape)
df_prop3=df_prop2.drop_duplicates(keep='last')
print("sesudah cek duplikat \n",df_prop3.shape)
```

```
sebelum cek duplikat
 (10000, 17)
sesudah cek duplikat
 (10000, 17)
```

[9]:
```
from sklearn.model_selection import train_test_split
x = df_prop3.drop(columns='category', axis=1)
y = df_prop3['category']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,␣
 ↪random_state=83)

print(x_train.shape)
print(x_test.shape)
```

```
(7000, 16)
(3000, 16)
```

[10]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer

kolom_kategori=['hasyard','haspool','isnewbuilt','hasstormprotector','hasstorageroom']

transform = make_column_transformer(
    (OneHotEncoder(), kolom_kategori),remainder='passthrough'
)
```

[11]:
```
x_train_enc = transform.fit_transform(x_train)

x_test_enc = transform.fit_transform(x_test)

df_train_enc = pd.DataFrame(x_train_enc,columns=transform.
 ↪get_feature_names_out())
df_test_enc = pd.DataFrame(x_test_enc,columns=transform.get_feature_names_out())
```

```python
df_train_enc.head(10)
df_test_enc.head(10)
```

[11]:

```
   onehotencoder__hasyard_no  onehotencoder__hasyard_yes  \
0                        1.0                         0.0
1                        0.0                         1.0
2                        1.0                         0.0
3                        0.0                         1.0
4                        1.0                         0.0
5                        1.0                         0.0
6                        1.0                         0.0
7                        1.0                         0.0
8                        1.0                         0.0
9                        1.0                         0.0

   onehotencoder__haspool_no  onehotencoder__haspool_yes  \
0                        1.0                         0.0
1                        0.0                         1.0
2                        1.0                         0.0
3                        0.0                         1.0
4                        1.0                         0.0
5                        1.0                         0.0
6                        0.0                         1.0
7                        0.0                         1.0
8                        0.0                         1.0
9                        1.0                         0.0

   onehotencoder__isnewbuilt_new  onehotencoder__isnewbuilt_old  \
0                            0.0                            1.0
1                            0.0                            1.0
2                            0.0                            1.0
3                            0.0                            1.0
4                            1.0                            0.0
5                            1.0                            0.0
6                            1.0                            0.0
7                            0.0                            1.0
8                            1.0                            0.0
9                            0.0                            1.0

   onehotencoder__hasstormprotector_no  onehotencoder__hasstormprotector_yes  \
0                                  1.0                                   0.0
1                                  1.0                                   0.0
2                                  1.0                                   0.0
3                                  0.0                                   1.0
4                                  0.0                                   1.0
5                                  0.0                                   1.0
```

```
6                                1.0                                     0.0
7                                1.0                                     0.0
8                                1.0                                     0.0
9                                1.0                                     0.0

   onehotencoder__hasstorageroom_no  onehotencoder__hasstorageroom_yes  …  \
0                               1.0                                0.0  …
1                               1.0                                0.0  …
2                               1.0                                0.0  …
3                               1.0                                0.0  …
4                               0.0                                1.0  …
5                               0.0                                1.0  …
6                               0.0                                1.0  …
7                               0.0                                1.0  …
8                               0.0                                1.0  …
9                               0.0                                1.0  …

   remainder__numberofrooms  remainder__floors  remainder__citycode  \
0                      81.0               46.0              49263.0
1                      51.0               43.0              50903.0
2                      51.0               26.0              86507.0
3                      21.0               55.0               5727.0
4                      33.0               56.0               3014.0
5                      88.0               35.0              32911.0
6                      17.0               32.0              64941.0
7                      12.0                6.0              11203.0
8                      69.0               87.0              38738.0
9                      11.0               81.0              68456.0

   remainder__citypartrange  remainder__numprevowners  remainder__made  \
0                       9.0                       2.0           2004.0
1                       2.0                       2.0           1992.0
2                       8.0                       6.0           2006.0
3                       5.0                       1.0           2000.0
4                      10.0                       6.0           2001.0
5                       7.0                       8.0           2006.0
6                       7.0                       2.0           2012.0
7                       2.0                       1.0           2008.0
8                       4.0                       1.0           2006.0
9                       1.0                       4.0           2008.0

   remainder__basement  remainder__attic  remainder__garage  \
0               5221.0            7101.0              289.0
1               8005.0            7138.0              113.0
2                730.0            5692.0              935.0
3               2872.0             851.0              565.0
4               8835.0            9875.0              559.0
```

```
5              6209.0          8195.0              134.0
6              9907.0          1932.0              728.0
7              8781.0          1584.0              599.0
8              7388.0          3537.0              791.0
9              7269.0          1015.0              715.0

   remainder__hasguestroom
0                      4.0
1                     10.0
2                      3.0
3                      5.0
4                      0.0
5                      6.0
6                      4.0
7                     10.0
8                      4.0
9                      4.0

[10 rows x 21 columns]
```

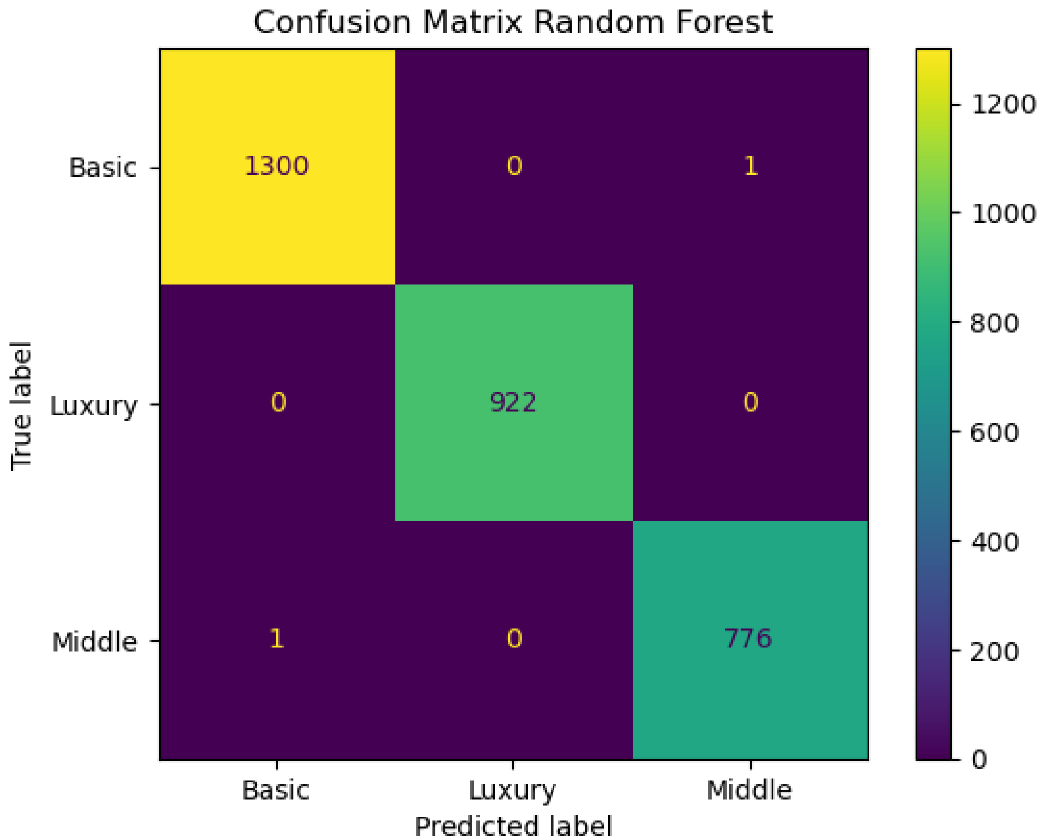
[12]:
```python
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.feature_selection import SelectKBest, SelectPercentile
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV, StratifiedKFold
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪ConfusionMatrixDisplay

pipe_RF = [('data scaling', StandardScaler()),
           ('feature select', SelectPercentile()),
           ('clf', RandomForestClassifier(random_state=83,␣
 ↪class_weight='balanced'))]

params_grid_RF = [
                {
                    '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': [SelectPercentile()],
                    'feature select__percentile': np.arange(20,50),
                    'clf__max_depth': np.arange(4, 5),
```

```
                    'clf__n_estimators': [100,150]
                    }]

estimator_RF = Pipeline(pipe_RF)
SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=83)
GSCV_RF_SP = GridSearchCV(estimator_RF, params_grid_RF, cv=SKF)
GSCV_RF_SP.fit(x_train_enc, y_train)
print("Finished")
```

```
Finished
```

[13]:
```
print("CV SCORE : {}", format(GSCV_RF_SP.best_score_))

print("Test Score : {}", format(GSCV_RF_SP.best_estimator_.score(x_test_enc,␣
 ↪y_test)))

print("Best Model : ",GSCV_RF_SP.best_estimator_)

mask = GSCV_RF_SP.best_estimator_.named_steps['feature select'].get_support()
print("Selected Feature : ",df_train_enc.columns[mask])

RF_pred = GSCV_RF_SP.predict(x_test_enc)

import matplotlib.pyplot as plt
cm = confusion_matrix(y_test, RF_pred, labels=GSCV_RF_SP.classes_)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=GSCV_RF_SP.
 ↪classes_)
disp.plot()
plt.title("Confusion Matrix Random Forest")
plt.show()
print(classification_report(y_test, RF_pred))
```

```
CV SCORE : {} 0.9994285714285714
Test Score : {} 0.9993333333333333
Best Model :  Pipeline(steps=[('data scaling', StandardScaler()),
                ('feature select', SelectPercentile(percentile=41)),
                ('clf',
                 RandomForestClassifier(class_weight='balanced', max_depth=4,
                                        random_state=83))])
Selected Feature :  Index(['onehotencoder__hasyard_no',
'onehotencoder__hasyard_yes',
       'onehotencoder__haspool_no', 'onehotencoder__haspool_yes',
       'onehotencoder__isnewbuilt_new', 'onehotencoder__isnewbuilt_old',
       'onehotencoder__hasstormprotector_no', 'remainder__squaremeters',
       'remainder__numberofrooms'],
      dtype='object')
```




```
              precision    recall  f1-score   support

       Basic       1.00      1.00      1.00      1301
      Luxury       1.00      1.00      1.00       922
      Middle       1.00      1.00      1.00       777

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

[14]:
```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.feature_selection import SelectKBest
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV, StratifiedKFold
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪ConfusionMatrixDisplay
```

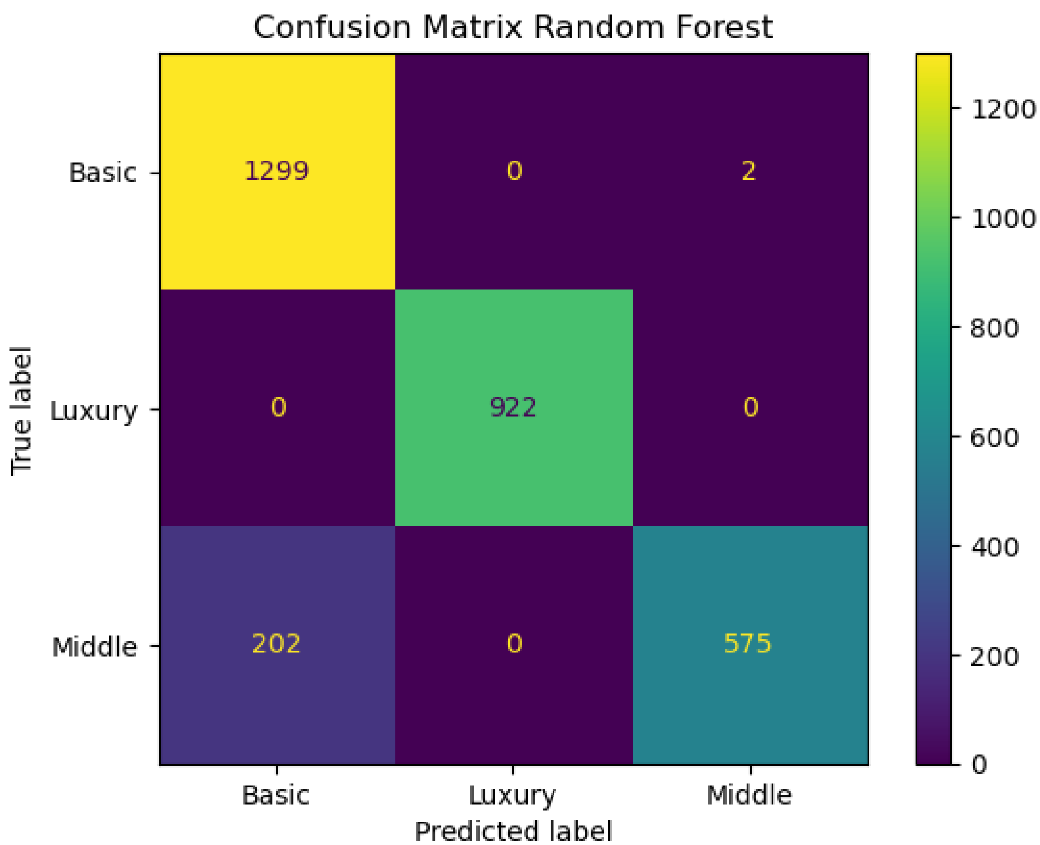
```
pipe_RF = [('data scaling', StandardScaler()),
           ('feature select', SelectKBest()),
           ('clf', RandomForestClassifier(random_state=83,␣
 ↪class_weight='balanced'))]

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': [MinMaxScaler()],
                'feature select__k': np.arange(2, 6),
                'clf__max_depth': np.arange(4, 5),
                'clf__n_estimators': [100, 150]
                }]

estimator_RF = Pipeline(pipe_RF)
SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=83)
GSCV_RF_SKB = GridSearchCV(estimator_RF, params_grid_RF, cv=SKF)
GSCV_RF_SKB.fit(x_train_enc, y_train)
print("Finished")
```

```
Finished
```

[16]:

```
print("CV SCORE : {}", format(GSCV_RF_SKB.best_score_))

print("Test Score : {}", format(GSCV_RF_SKB.best_estimator_.score(x_test_enc,␣
 ↪y_test)))

print("Best Model : ",GSCV_RF_SKB.best_estimator_)

mask = GSCV_RF_SKB.best_estimator_.named_steps['feature select'].get_support()
print("Selected Feature : ",df_train_enc.columns[mask])

RF_pred = GSCV_RF_SKB.predict(x_test_enc)

import matplotlib.pyplot as plt
cm = confusion_matrix(y_test, RF_pred, labels=GSCV_RF_SKB.classes_)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=GSCV_RF_SKB.
 ↪classes_)
disp.plot()
plt.title("Confusion Matrix Random Forest")
plt.show()
print(classification_report(y_test, RF_pred))
```

```
CV SCORE : {} 0.9365714285714286
Test Score : {} 0.932
Best Model :  Pipeline(steps=[('data scaling', StandardScaler()),
                ('feature select', SelectKBest(k=2)),
                ('clf',
                 RandomForestClassifier(class_weight='balanced', max_depth=4,
                                        random_state=83))])
Selected Feature :  Index(['onehotencoder__haspool_no',
'remainder__squaremeters'], dtype='object')
```




```
              precision    recall  f1-score   support

       Basic       0.87      1.00      0.93      1301
      Luxury       1.00      1.00      1.00       922
      Middle       1.00      0.74      0.85       777

    accuracy                           0.93      3000
   macro avg       0.95      0.91      0.93      3000
weighted avg       0.94      0.93      0.93      3000
```

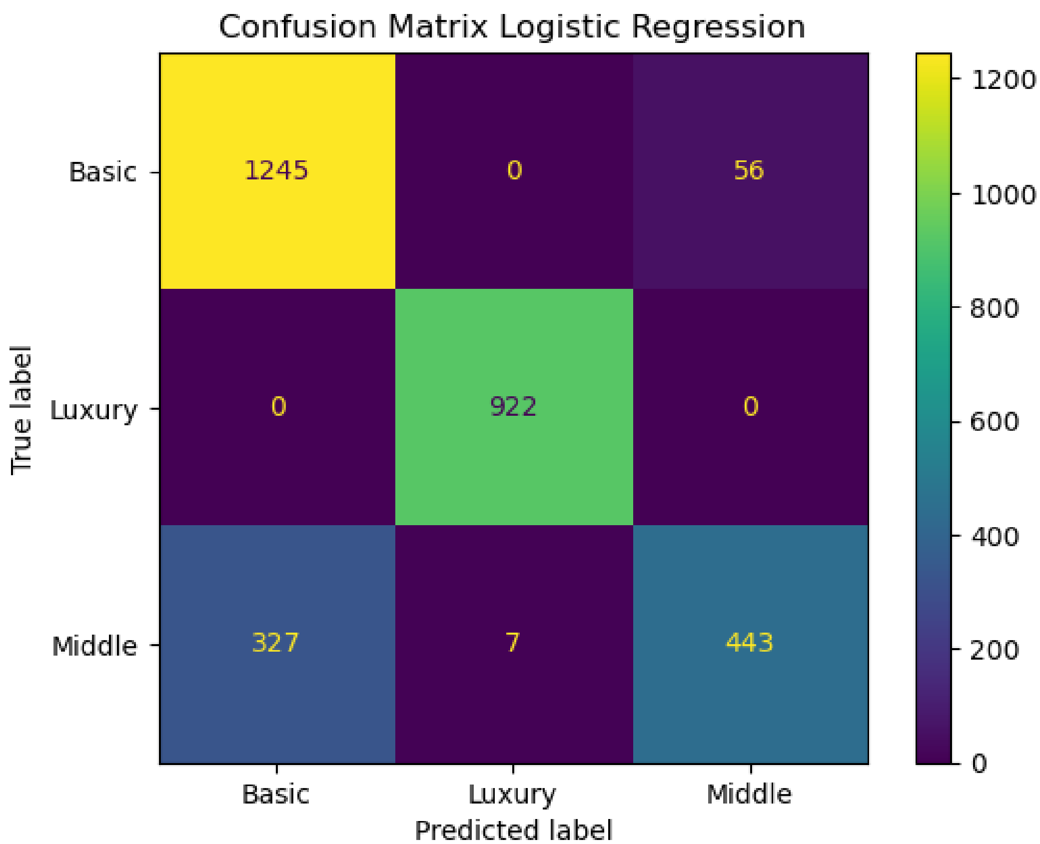
[17]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.feature_selection import SelectKBest
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV, StratifiedKFold
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix,␣
↪ConfusionMatrixDisplay

pipe_LR = Pipeline(steps=[
    ('scale', MinMaxScaler()),
    ('feat_select', SelectKBest()),
    ('clf', LogisticRegression(solver='liblinear', max_iter=10000))
])

params_grid_LR = [
    {
        'scale': [MinMaxScaler()],
        'feat_select__k': np.arange(2, 6),
        'clf__C': [0.0001, 0.01, 0.1, 1, 10],
        'clf__penalty': ['l1', 'l2']
    },
    {
        'scale': [StandardScaler()],
        'feat_select__k': np.arange(2, 6),
        'clf__C': [0.0001, 0.01, 0.1, 1, 10],
        'clf__penalty': ['l1', 'l2']
    }
]

GSCV_LR_SKB = GridSearchCV(pipe_LR, params_grid_LR,␣
↪cv=StratifiedKFold(n_splits=5))
GSCV_LR_SKB.fit(x_train_enc, y_train)
print("Finished")
```

```
Finished
```

[19]:
```
mask = GSCV_LR_SKB.best_estimator_.named_steps['feat_select'].get_support()
print("CV SCORE : {}",format(GSCV_LR_SKB.best_score_))
print("Test Score : {}",format(GSCV_LR_SKB.best_estimator_.score(x_test_enc,␣
↪y_test)))

print("Best Model : {}",format(GSCV_LR_SKB.best_estimator_))
print("Selected Feature : {}",format(df_train_enc.columns[mask]))

LR_pred = GSCV_LR_SKB.predict(x_test_enc)
```

```python
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, LR_pred, labels=GSCV_LR_SKB.classes_)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=GSCV_LR_SKB.
 ↪classes_)
disp.plot()
plt.title("Confusion Matrix Logistic Regression")
plt.show()

print(classification_report(y_test, LR_pred))
```

```
CV SCORE : {} 0.8700000000000001
Test Score : {} 0.87
Best Model : {} Pipeline(steps=[('scale', StandardScaler()), ('feat_select',
SelectKBest(k=4)),
                ('clf',
                 LogisticRegression(C=10, max_iter=10000, penalty='l1',
                                    solver='liblinear'))])
Selected Feature : {} Index(['onehotencoder__haspool_no',
'onehotencoder__haspool_yes',
       'onehotencoder__isnewbuilt_new', 'remainder__squaremeters'],
      dtype='object')
```




```
              precision    recall  f1-score   support

       Basic       0.79      0.96      0.87      1301
      Luxury       0.99      1.00      1.00       922
      Middle       0.89      0.57      0.69       777

    accuracy                           0.87      3000
   macro avg       0.89      0.84      0.85      3000
weighted avg       0.88      0.87      0.86      3000
```

[20]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.feature_selection import SelectPercentile
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV, StratifiedKFold
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix,␣
↪ConfusionMatrixDisplay
```

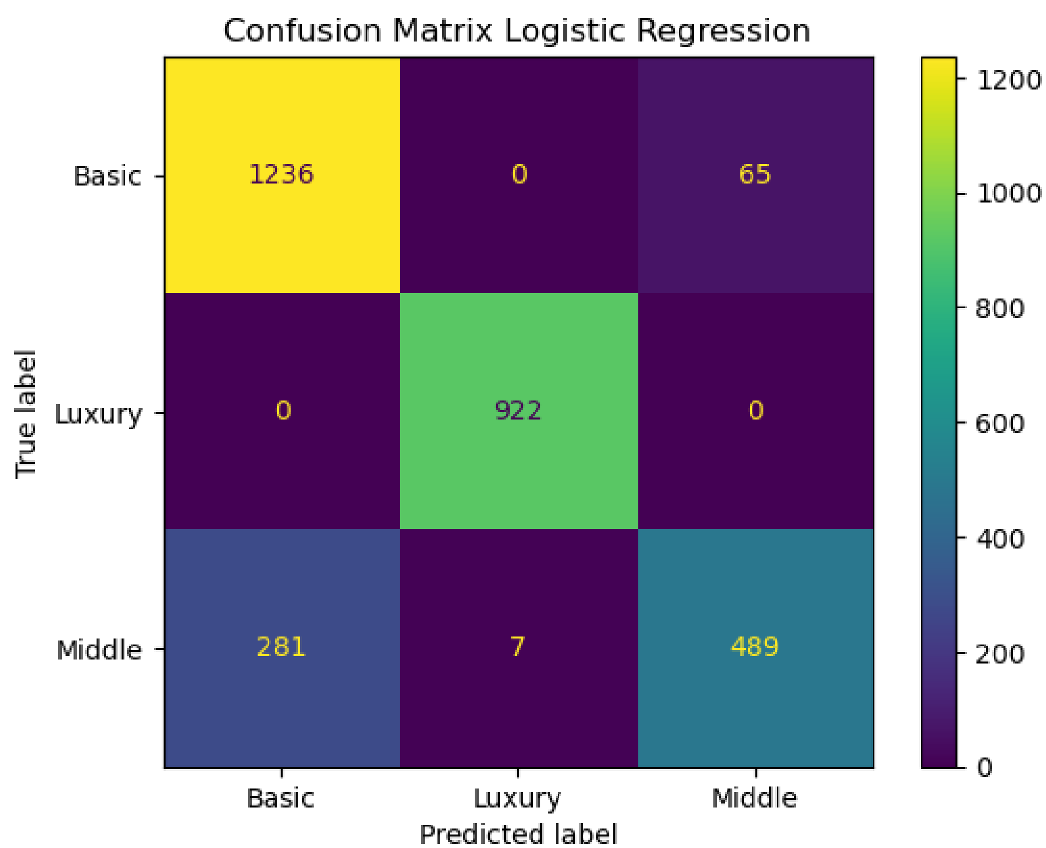
```
pipe_LR = Pipeline(steps=[
    ('scale', MinMaxScaler()),
    ('feat_select', SelectPercentile()),
    ('clf', LogisticRegression(solver='liblinear', max_iter=10000))
])

params_grid_LR = [
    {
        'scale': [MinMaxScaler()],
        'feat_select__percentile': np.arange(10, 100, 10),
        'clf__C': [0.0001, 0.01, 0.1, 1, 10],
        'clf__penalty': ['l1', 'l2']
    },
    {
        'scale': [StandardScaler()],
        'feat_select__percentile': np.arange(10, 100, 10),
        'clf__C': [0.0001, 0.01, 0.1, 1, 10],
        'clf__penalty': ['l1', 'l2']
    }
]

GSCV_LR_SP = GridSearchCV(pipe_LR, params_grid_LR,␣
 ↪cv=StratifiedKFold(n_splits=5))
GSCV_LR_SP.fit(x_train_enc, y_train)
print("Finished")
```

```
Finished
```

[21]:
```
mask = GSCV_LR_SP.best_estimator_.named_steps['feat_select'].get_support()
print("CV SCORE : {}",format(GSCV_LR_SP.best_score_))
print("Test Score : {}",format(GSCV_LR_SP.best_estimator_.score(x_test_enc,␣
 ↪y_test)))

print("Best Model : {}",format(GSCV_LR_SP.best_estimator_))
print("Selected Feature : {}",format(df_train_enc.columns[mask]))

LR_pred = GSCV_LR_SP.predict(x_test_enc)

import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, LR_pred, labels=GSCV_LR_SP.classes_)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=GSCV_LR_SP.
 ↪classes_)
disp.plot()
plt.title("Confusion Matrix Logistic Regression")
plt.show()
```

```python
print(classification_report(y_test, LR_pred))
```

```
CV SCORE : {} 0.8734285714285714
Test Score : {} 0.8823333333333333
Best Model : {} Pipeline(steps=[('scale', StandardScaler()),
                ('feat_select', SelectPercentile(percentile=30)),
                ('clf',
                 LogisticRegression(C=10, max_iter=10000, penalty='l1',
                                    solver='liblinear'))])
Selected Feature : {} Index(['onehotencoder__hasyard_no',
'onehotencoder__haspool_no',
       'onehotencoder__haspool_yes', 'onehotencoder__isnewbuilt_new',
       'onehotencoder__isnewbuilt_old', 'remainder__squaremeters'],
      dtype='object')
```




```
              precision    recall  f1-score   support

       Basic       0.81      0.95      0.88      1301
      Luxury       0.99      1.00      1.00       922
```

```
      Middle       0.88      0.63      0.73       777

    accuracy                           0.88      3000
   macro avg       0.90      0.86      0.87      3000
weighted avg       0.89      0.88      0.88      3000
```

[23]:
```
import pickle

with open('model_RF.pkl', 'wb') as file:
    pickle.dump(GSCV_RF_SP, file)
```

[ ]:
```
import sklearn
print(sklearn.__version__)
```

```
1.4.2
```

# kasi-b-tensorflow-svm-vs-gbt-rendy

October 25, 2024

[11]:
```python
#semoga kali ini jadi

import pandas as pd
import numpy as np

#load data
df_prop = pd.read_csv('Dataset UTS_Gasal 2425.csv')
df_prop.head(10)
```

[11]:
```
   squaremeters  numberofrooms hasyard haspool  floors  citycode  \
0         75523              3      no     yes      63      9373
1         55712             58      no     yes      19     34457
2         86929            100     yes      no      11     98155
3         51522              3      no      no      61      9047
4         96470             74     yes      no      21     92029
5         79770              3      no     yes      69     54812
6         75985             60     yes      no      67      6517
7         64169             88      no     yes       6     61711
8         92383             12      no      no      78     71982
9         95121             46      no     yes       3      9382

   citypartrange  numprevowners  made isnewbuilt hasstormprotector  basement  \
0              3              8  2005        old               yes      4313
1              6              8  2021        old                no      2937
2              3              4  2003        new                no      6326
3              8              3  2012        new               yes       632
4              4              2  2011        new               yes      5414
5             10              5  2018        old               yes      8871
6              6              9  2009        new               yes      4878
7              3              9  2011        new               yes      3054
8              3              7  2000        old                no      7507
9              7              9  1994        old                no       615

   attic  garage hasstorageroom  hasguestroom      price category
0   9005     956             no             7  7559081.5   Luxury
1   8852     135            yes             9  5574642.1   Middle
2   4748     654             no            10  8696869.3   Luxury
```

```
3   5792    807        yes        5   5154055.2   Middle
4   1172    716        yes        9   9652258.1   Luxury
5   7117    240         no        7   7986665.8   Luxury
6    281    384        yes        5   7607322.9   Luxury
7    129    726         no        9   6420823.1   Middle
8   9056    892        yes        1   9244344.0   Luxury
9   1221    328         no       10   9515440.4   Luxury
```

[12]:
```python
df_prop.describe()
```

[12]:
```
       squaremeters  numberofrooms        floors       citycode  citypartrange  \
count   10000.00000   10000.000000  10000.000000   10000.000000   10000.000000
mean    49870.13120      50.358400     50.276300   50225.486100       5.510100
std     28774.37535      28.816696     28.889171   29006.675799       2.872024
min        89.00000       1.000000      1.000000       3.000000       1.000000
25%     25098.50000      25.000000     25.000000   24693.750000       3.000000
50%     50105.50000      50.000000     50.000000   50693.000000       5.000000
75%     74609.75000      75.000000     76.000000   75683.250000       8.000000
max     99999.00000     100.000000    100.000000   99953.000000      10.000000

       numprevowners         made      basement        attic       garage  \
count   10000.000000  10000.00000  10000.000000  10000.00000  10000.00000
mean        5.521700   2005.48850   5033.103900   5028.01060    553.12120
std         2.856667      9.30809   2876.729545   2894.33221    262.05017
min         1.000000   1990.00000      0.000000      1.00000    100.00000
25%         3.000000   1997.00000   2559.750000   2512.00000    327.75000
50%         5.000000   2005.50000   5092.500000   5045.00000    554.00000
75%         8.000000   2014.00000   7511.250000   7540.50000    777.25000
max        10.000000   2021.00000  10000.000000  10000.00000   1000.00000

       hasguestroom         price
count   10000.00000  1.000000e+04
mean        4.99460  4.993448e+06
std         3.17641  2.877424e+06
min         0.00000  1.031350e+04
25%         2.00000  2.516402e+06
50%         5.00000  5.016180e+06
75%         8.00000  7.469092e+06
max        10.00000  1.000677e+07
```

[13]:
```python
df_prop2 = df_prop.drop('price', axis=1)
df_prop2['category'].value_counts()
```

[13]:
```
category
Basic     4344
Luxury    3065
Middle    2591
```

```
Name: count, dtype: int64
```

[14]:
```
print("data null\n", df_prop2.isnull().sum())
print("data kosong\n", df_prop2.empty)
print("data nan\n", df_prop2.isna().sum())
```

```
data null
 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
category             0
dtype: int64
data kosong
 False
data nan
 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
category             0
dtype: int64
```

[15]:
```
print("Sebelum drop missing value \n",df_prop2.shape)
df_prop2= df_prop2.dropna(how='any',inplace=False)
print("Sesudah drop missing value \n",df_prop2.shape)
```

```
Sebelum drop missing value
 (10000, 17)
Sesudah drop missing value
 (10000, 17)
```

[16]:
```
print("sebelum cek duplikat \n",df_prop2.shape)
df_prop3=df_prop2.drop_duplicates(keep='last')
print("sesudah cek duplikat \n",df_prop3.shape)
```

```
sebelum cek duplikat
 (10000, 17)
sesudah cek duplikat
 (10000, 17)
```

[17]:
```
from sklearn.model_selection import train_test_split
x = df_prop3.drop(columns='category', axis=1)
y = df_prop3['category']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,␣
↪random_state=83)

print(x_train.shape)
print(x_test.shape)
```

```
(7000, 16)
(3000, 16)
```

[18]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer

kolom_kategori=['hasyard','haspool','isnewbuilt','hasstormprotector','hasstorageroom']

transform = make_column_transformer(
    (OneHotEncoder(), kolom_kategori),remainder='passthrough'
)
```

[19]:
```
x_train_enc = transform.fit_transform(x_train)

x_test_enc = transform.fit_transform(x_test)

df_train_enc = pd.DataFrame(x_train_enc,columns=transform.
↪get_feature_names_out())
df_test_enc = pd.DataFrame(x_test_enc,columns=transform.get_feature_names_out())
```

```python
df_train_enc.head(10)
df_test_enc.head(10)
```

[19]:

```
   onehotencoder__hasyard_no  onehotencoder__hasyard_yes  \
0                        1.0                         0.0
1                        0.0                         1.0
2                        1.0                         0.0
3                        0.0                         1.0
4                        1.0                         0.0
5                        1.0                         0.0
6                        1.0                         0.0
7                        1.0                         0.0
8                        1.0                         0.0
9                        1.0                         0.0

   onehotencoder__haspool_no  onehotencoder__haspool_yes  \
0                        1.0                         0.0
1                        0.0                         1.0
2                        1.0                         0.0
3                        0.0                         1.0
4                        1.0                         0.0
5                        1.0                         0.0
6                        0.0                         1.0
7                        0.0                         1.0
8                        0.0                         1.0
9                        1.0                         0.0

   onehotencoder__isnewbuilt_new  onehotencoder__isnewbuilt_old  \
0                            0.0                            1.0
1                            0.0                            1.0
2                            0.0                            1.0
3                            0.0                            1.0
4                            1.0                            0.0
5                            1.0                            0.0
6                            1.0                            0.0
7                            0.0                            1.0
8                            1.0                            0.0
9                            0.0                            1.0

   onehotencoder__hasstormprotector_no  onehotencoder__hasstormprotector_yes  \
0                                  1.0                                   0.0
1                                  1.0                                   0.0
2                                  1.0                                   0.0
3                                  0.0                                   1.0
4                                  0.0                                   1.0
5                                  0.0                                   1.0
```

```
6                                  1.0                                     0.0
7                                  1.0                                     0.0
8                                  1.0                                     0.0
9                                  1.0                                     0.0

   onehotencoder__hasstorageroom_no  onehotencoder__hasstorageroom_yes  …  \
0                               1.0                                0.0  …
1                               1.0                                0.0  …
2                               1.0                                0.0  …
3                               1.0                                0.0  …
4                               0.0                                1.0  …
5                               0.0                                1.0  …
6                               0.0                                1.0  …
7                               0.0                                1.0  …
8                               0.0                                1.0  …
9                               0.0                                1.0  …

   remainder__numberofrooms  remainder__floors  remainder__citycode  \
0                      81.0               46.0              49263.0
1                      51.0               43.0              50903.0
2                      51.0               26.0              86507.0
3                      21.0               55.0               5727.0
4                      33.0               56.0               3014.0
5                      88.0               35.0              32911.0
6                      17.0               32.0              64941.0
7                      12.0                6.0              11203.0
8                      69.0               87.0              38738.0
9                      11.0               81.0              68456.0

   remainder__citypartrange  remainder__numprevowners  remainder__made  \
0                       9.0                       2.0           2004.0
1                       2.0                       2.0           1992.0
2                       8.0                       6.0           2006.0
3                       5.0                       1.0           2000.0
4                      10.0                       6.0           2001.0
5                       7.0                       8.0           2006.0
6                       7.0                       2.0           2012.0
7                       2.0                       1.0           2008.0
8                       4.0                       1.0           2006.0
9                       1.0                       4.0           2008.0

   remainder__basement  remainder__attic  remainder__garage  \
0               5221.0            7101.0              289.0
1               8005.0            7138.0              113.0
2                730.0            5692.0              935.0
3               2872.0             851.0              565.0
4               8835.0            9875.0              559.0
```

```
5              6209.0           8195.0             134.0
6              9907.0           1932.0             728.0
7              8781.0           1584.0             599.0
8              7388.0           3537.0             791.0
9              7269.0           1015.0             715.0

   remainder__hasguestroom
0                      4.0
1                     10.0
2                      3.0
3                      5.0
4                      0.0
5                      6.0
6                      4.0
7                     10.0
8                      4.0
9                      4.0

[10 rows x 21 columns]
```

[22]:
```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.feature_selection import SelectKBest, SelectPercentile
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪ConfusionMatrixDisplay

pipe_svm = Pipeline([
    ('scale', MinMaxScaler()),
    ('feat_select',SelectPercentile()),
    ('clf', SVC(class_weight='balanced'))
])

params_grid_svm=[
    {
        'scale' : [MinMaxScaler()],
        'feat_select': [SelectPercentile()],
        'feat_select__percentile' : np.arange(20,50),
        'clf__kernel' : ['poly','rbf'],
         'clf__C' : [0.1,1],
    'clf__gamma' : [0.1,1]
    },
    {
        'scale' : [StandardScaler()],
        'feat_select': [SelectPercentile()],
        'feat_select__percentile' : np.arange(20,50),
```

```
        'clf__kernel' : ['poly','rbf'],
         'clf__C' : [0.1,1],
    'clf__gamma' : [0.1,1]
    }
]

estimator_svm = Pipeline(pipe_svm)

SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=83)

GSCV_SVM_SP = GridSearchCV(pipe_svm,params_grid_svm,cv=SKF)

GSCV_SVM_SP.fit(x_train_enc,y_train)
print("GSCV Finished")
#35 menit
```

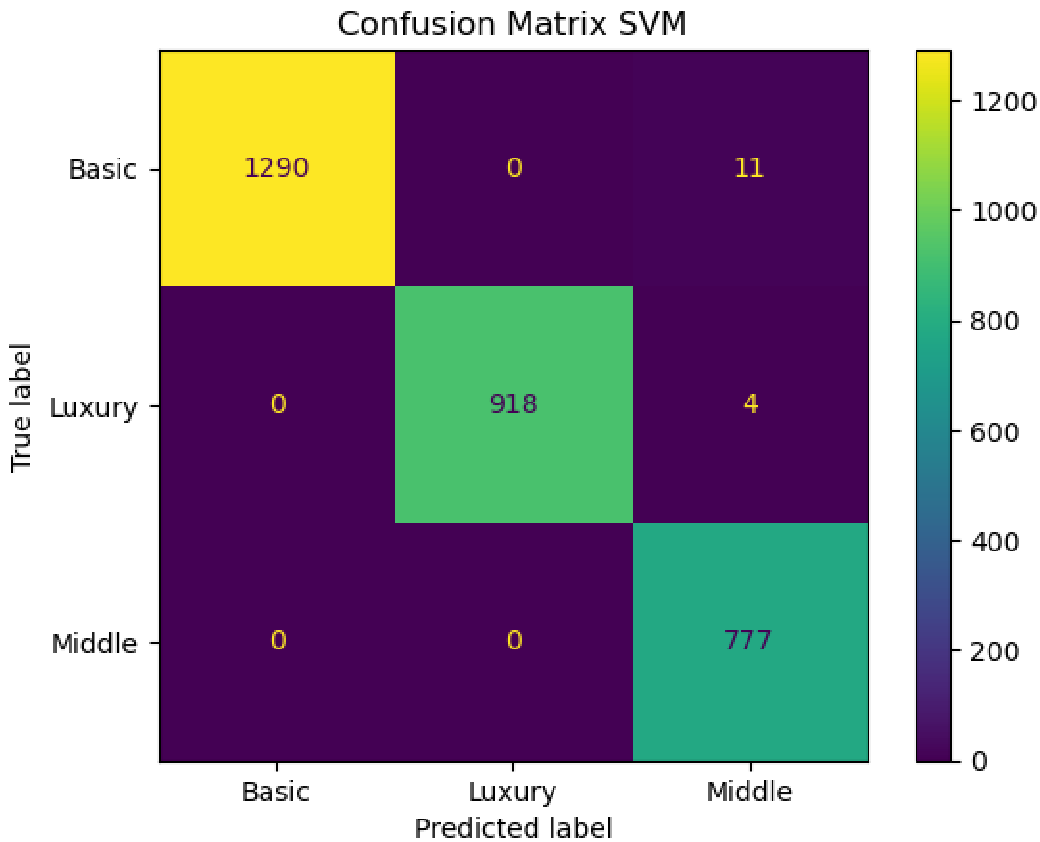
```
GSCV Finished
```

[24]:

```
print("CV Score : ",format(GSCV_SVM_SP.best_score_))

print("Test Score : ",format(GSCV_SVM_SP.best_estimator_.
 ↪score(x_test_enc,y_test)))

print("Best Model : ",GSCV_SVM_SP.best_estimator_)
mask = GSCV_SVM_SP.best_estimator_.named_steps['feat_select'].get_support()
print("Best Feature : ",df_train_enc.columns[mask])

SVM_pred = GSCV_SVM_SP.predict(x_test_enc)

import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, SVM_pred, labels=GSCV_SVM_SP.classes_)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=GSCV_SVM_SP.
 ↪classes_)
disp.plot()
plt.title('Confusion Matrix SVM')
plt.show()

print("Classification Report SVM : \n",classification_report(y_test, SVM_pred))
```

```
CV Score :  0.9951428571428572
Test Score :  0.995
Best Model :  Pipeline(steps=[('scale', StandardScaler()),
                ('feat_select', SelectPercentile(percentile=31)),
                ('clf',
                 SVC(C=1, class_weight='balanced', gamma=1, kernel='poly'))])
```

```
Best Feature :  Index(['onehotencoder__hasyard_no',
'onehotencoder__hasyard_yes',
       'onehotencoder__haspool_no', 'onehotencoder__haspool_yes',
       'onehotencoder__isnewbuilt_new', 'onehotencoder__isnewbuilt_old',
       'remainder__squaremeters'],
      dtype='object')
```




```
Classification Report SVM :
               precision    recall  f1-score   support

       Basic       1.00      0.99      1.00      1301
      Luxury       1.00      1.00      1.00       922
      Middle       0.98      1.00      0.99       777

    accuracy                           0.99      3000
   macro avg       0.99      1.00      0.99      3000
weighted avg       1.00      0.99      1.00      3000
```

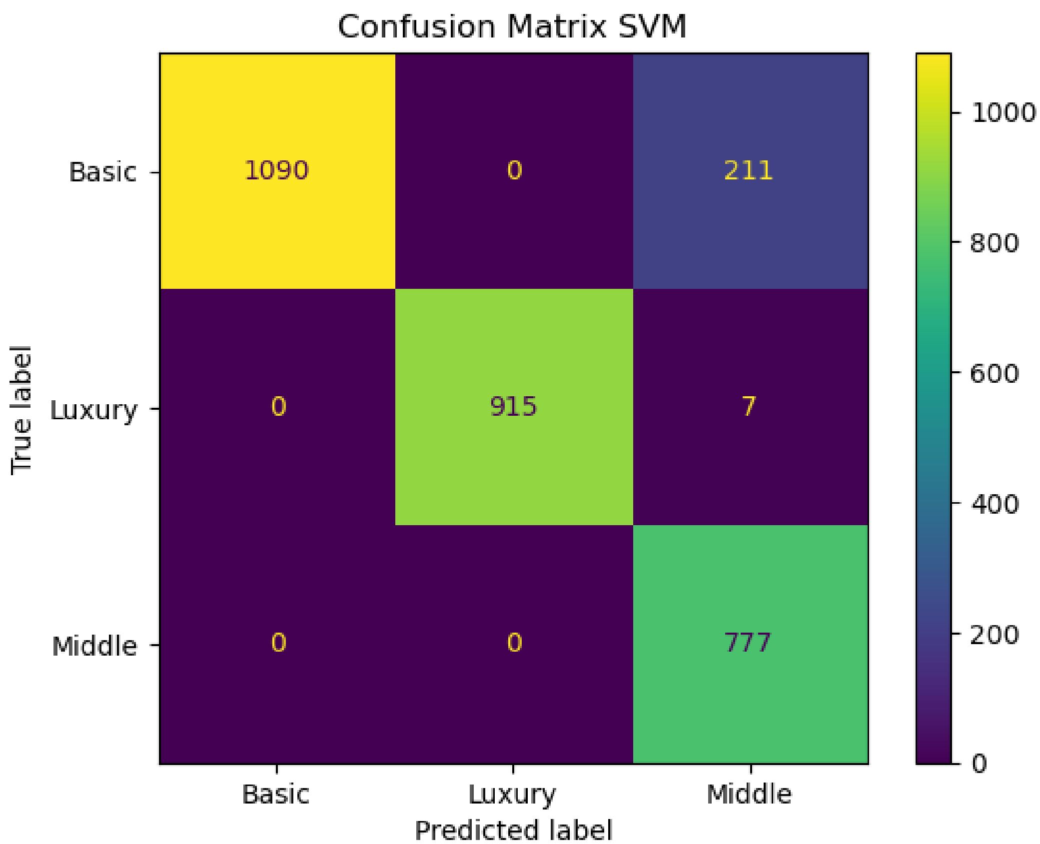
[26]:
```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.feature_selection import SelectKBest, SelectPercentile
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, confusion_matrix,␣
↪ConfusionMatrixDisplay

pipe_svm = Pipeline([
    ('scale', MinMaxScaler()),
    ('feat_select',SelectKBest()),
    ('clf', SVC(class_weight='balanced'))
])

params_grid_svm=[
    {
    'scale' : [MinMaxScaler()],
    'feat_select__k' : np.arange(2,6),
    'clf__kernel' : ['poly','rbf'],
    'clf__C' : [0.1,1],
    'clf__gamma' : [0.1,1]
    },
    {
    'scale' : [StandardScaler()],
    'feat_select__k' : np.arange(2,6),
    'clf__kernel' : ['poly','rbf'],
     'clf__C' : [0.1,1],
    'clf__gamma' : [0.1,1]
    }
]

estimator_svm = Pipeline(pipe_svm)

SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=83)

GSCV_SVM_SKB = GridSearchCV(pipe_svm,params_grid_svm,cv=SKF)

GSCV_SVM_SKB.fit(x_train_enc,y_train)
print("GSCV Finished")
#35 menit
```

```
GSCV Finished
```

[27]:
```
print("CV Score : ",format(GSCV_SVM_SKB.best_score_))

print("Test Score : ",format(GSCV_SVM_SKB.best_estimator_.
↪score(x_test_enc,y_test)))
```

```python
print("Best Model : ",GSCV_SVM_SKB.best_estimator_)
mask = GSCV_SVM_SKB.best_estimator_.named_steps['feat_select'].get_support()
print("Best Feature : ",df_train_enc.columns[mask])

SVM_pred = GSCV_SVM_SKB.predict(x_test_enc)

import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, SVM_pred, labels=GSCV_SVM_SKB.classes_)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=GSCV_SVM_SKB.
 ↪classes_)
disp.plot()
plt.title('Confusion Matrix SVM')
plt.show()

print("Classification Report SVM : \n",classification_report(y_test, SVM_pred))
```

```
CV Score :  0.9275714285714287
Test Score :  0.9273333333333333
Best Model :  Pipeline(steps=[('scale', StandardScaler()), ('feat_select',
SelectKBest(k=5)),
                ('clf',
                 SVC(C=1, class_weight='balanced', gamma=1, kernel='poly'))])
Best Feature :  Index(['onehotencoder__haspool_no',
'onehotencoder__haspool_yes',
       'onehotencoder__isnewbuilt_new', 'onehotencoder__isnewbuilt_old',
       'remainder__squaremeters'],
      dtype='object')
```




```
Classification Report SVM :
              precision    recall  f1-score   support

       Basic       1.00      0.84      0.91      1301
      Luxury       1.00      0.99      1.00       922
      Middle       0.78      1.00      0.88       777

    accuracy                           0.93      3000
   macro avg       0.93      0.94      0.93      3000
weighted avg       0.94      0.93      0.93      3000
```

[41]:

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest, SelectPercentile

pipe_GBT = Pipeline(steps=[
```

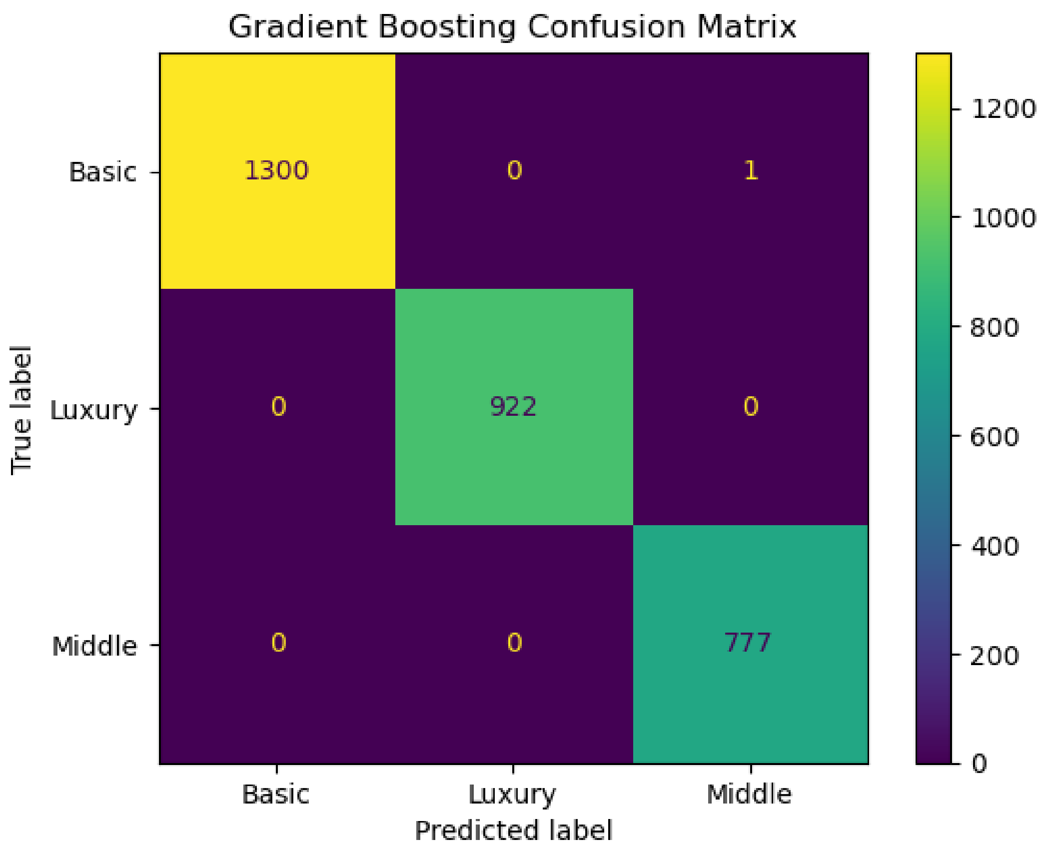
```
    ('feat_select', SelectPercentile()),
    ('clf', GradientBoostingClassifier(random_state=83))])

param_grid_GBT = [
    {
        'feat_select' : [SelectPercentile()],
        'feat_select__percentile' : np.arange(20,50),
        'clf__max_depth': [*np.arange(4,5)],
        'clf__n_estimators': [100,150],
        'clf__learning_rate': [0.01,0.1,1]
    },
    {
        'feat_select' : [SelectPercentile()],
        'feat_select__percentile' : np.arange(20,50),
        'clf__max_depth': [*np.arange(4,5)],
        'clf__n_estimators': [100,150],
        'clf__learning_rate': [0.01,0.1,1]
    }
]

GSCV_GBT_SP = GridSearchCV(pipe_GBT, param_grid_GBT,␣
 ↪cv=StratifiedKFold(n_splits=5))
GSCV_GBT_SP.fit(x_train_enc, y_train)
```

[41]:
```
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=False),
             estimator=Pipeline(steps=[('feat_select', SelectPercentile()),
                                       ('clf',
GradientBoostingClassifier(random_state=83))]),
             param_grid=[{'clf__learning_rate': [0.01, 0.1, 1],
                          'clf__max_depth': [4],
                          'clf__n_estimators': [100, 150],
                          'feat_select': [SelectPercentile()],
                          'feat_select__percentile': array([20, 21, 22, 23, 24,
25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,
       37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])},
                         {'clf__learning_rate': [0.01, 0.1, 1],
                          'clf__max_depth': [4],
                          'clf__n_estimators': [100, 150],
                          'feat_select': [SelectPercentile()],
                          'feat_select__percentile': array([20, 21, 22, 23, 24,
25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,
       37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])}])
```

[42]:
```
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪ConfusionMatrixDisplay

print("CV Score : {}".format(GSCV_GBT_SP.best_score_))
```

```
print("Test Score : {}".format(GSCV_GBT_SP.best_estimator_.score(x_test_enc,␣
↪y_test)))
print("Best Model : ", GSCV_GBT_SP.best_estimator_)

mask = GSCV_GBT_SP.best_estimator_.named_steps['feat_select'].get_support()
print("Best Features:", df_train_enc.columns[mask])

RF_pred = GSCV_GBT_SP.predict(x_test_enc)

import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, RF_pred, labels=GSCV_GBT_SP.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=GSCV_GBT_SP.
↪classes_)
disp.plot()

plt.title("Gradient Boosting Confusion Matrix")
plt.show()
print("Classification report Gradient Boosting:
↪\n",classification_report(y_test,RF_pred))
```

```
CV Score : 0.9991428571428571
Test Score : 0.9996666666666667
Best Model :  Pipeline(steps=[('feat_select', SelectPercentile(percentile=29)),
                ('clf',
                 GradientBoostingClassifier(learning_rate=0.01, max_depth=4,
                                            random_state=83))])
Best Features: Index(['onehotencoder__hasyard_no', 'onehotencoder__haspool_no',
       'onehotencoder__haspool_yes', 'onehotencoder__isnewbuilt_new',
       'onehotencoder__isnewbuilt_old', 'remainder__squaremeters'],
      dtype='object')
```




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

       Basic       1.00      1.00      1.00      1301
      Luxury       1.00      1.00      1.00       922
      Middle       1.00      1.00      1.00       777

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

[43]:
```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest, SelectPercentile

pipe_GBT = Pipeline(steps=[
```

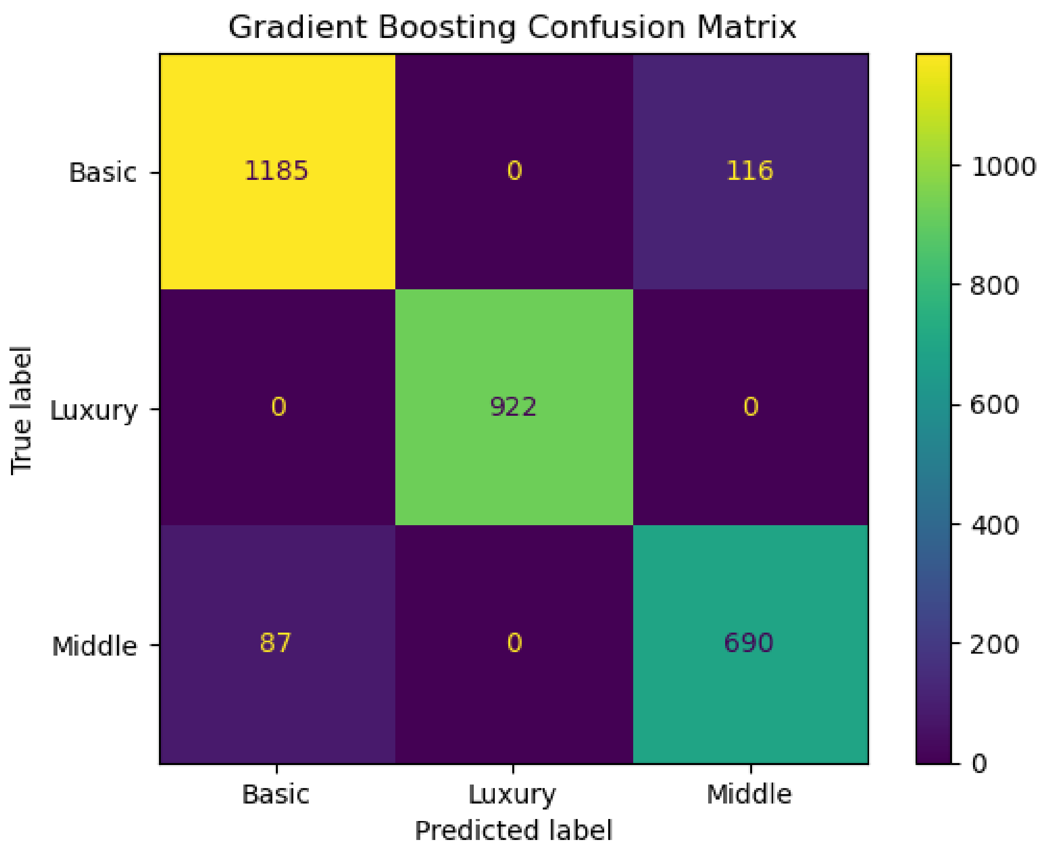
```
    ('feat_select', SelectKBest()),
    ('clf', GradientBoostingClassifier(random_state=83))])

param_grid_GBT = [
    {
        'feat_select__k' : np.arange(2,6),
        'clf__max_depth': [*np.arange(4,5)],
        'clf__n_estimators': [100,150],
        'clf__learning_rate': [0.01,0.1,1]
    },
    {
        'feat_select__k' : np.arange(2,6),
        'clf__max_depth': [*np.arange(4,5)],
        'clf__n_estimators': [100,150],
        'clf__learning_rate': [0.01,0.1,1]
    }
]

GSCV_GBT_SK = GridSearchCV(pipe_GBT, param_grid_GBT,␣
 ↪cv=StratifiedKFold(n_splits=5))
GSCV_GBT_SK.fit(x_train_enc, y_train)
```

[43]:
```
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=False),
             estimator=Pipeline(steps=[('feat_select', SelectKBest()),
                                       ('clf',
GradientBoostingClassifier(random_state=83))]),
             param_grid=[{'clf__learning_rate': [0.01, 0.1, 1],
                          'clf__max_depth': [4],
                          'clf__n_estimators': [100, 150],
                          'feat_select__k': array([2, 3, 4, 5])},
                         {'clf__learning_rate': [0.01, 0.1, 1],
                          'clf__max_depth': [4],
                          'clf__n_estimators': [100, 150],
                          'feat_select__k': array([2, 3, 4, 5])}])
```

[44]:
```
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪ConfusionMatrixDisplay

print("CV Score : {}".format(GSCV_GBT_SK.best_score_))
print("Test Score : {}".format(GSCV_GBT_SK.best_estimator_.score(x_test_enc,␣
 ↪y_test)))
print("Best Model : ", GSCV_GBT_SK.best_estimator_)

mask = GSCV_GBT_SK.best_estimator_.named_steps['feat_select'].get_support()
print("Best Features:", df_train_enc.columns[mask])

RF_pred = GSCV_GBT_SK.predict(x_test_enc)
```

```python
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, RF_pred, labels=GSCV_GBT_SK.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=GSCV_GBT_SK.
 ↪classes_)
disp.plot()

plt.title("Gradient Boosting Confusion Matrix")
plt.show()
print("Classification report Gradient Boosting:
 ↪\n",classification_report(y_test,RF_pred))
```

```
CV Score : 0.9385714285714286
Test Score : 0.9323333333333333
Best Model :  Pipeline(steps=[('feat_select', SelectKBest(k=4)),
                ('clf',
                 GradientBoostingClassifier(max_depth=4, n_estimators=150,
                                            random_state=83))])
Best Features: Index(['onehotencoder__haspool_no', 'onehotencoder__haspool_yes',
       'onehotencoder__isnewbuilt_new', 'remainder__squaremeters'],
      dtype='object')
```



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

       Basic       0.93      0.91      0.92      1301
      Luxury       1.00      1.00      1.00       922
      Middle       0.86      0.89      0.87       777

    accuracy                           0.93      3000
   macro avg       0.93      0.93      0.93      3000
weighted avg       0.93      0.93      0.93      3000
```

[45]:
```python
import pickle

with open('model_GBT.pkl', 'wb') as file:
    pickle.dump(GSCV_GBT_SP, file)
```

[46]:
```python
import sklearn
print(sklearn.__version__)
```

```
1.4.2
```

# lasso-vs-randomforest-albert-kent

October 25, 2024

[90]:
```
import pandas as pd
import numpy as np

df_uts = pd.read_csv(r'Dataset UTS_Gasal 2425.csv')
df_uts.head(20)
```

[90]:
```
    squaremeters  numberofrooms hasyard haspool  floors  citycode  \
0          75523              3      no     yes      63      9373
1          55712             58      no     yes      19     34457
2          86929            100     yes      no      11     98155
3          51522              3      no      no      61      9047
4          96470             74     yes      no      21     92029
5          79770              3      no     yes      69     54812
6          75985             60     yes      no      67      6517
7          64169             88      no     yes       6     61711
8          92383             12      no      no      78     71982
9          95121             46      no     yes       3      9382
10         76485             47     yes      no       9     90254
11         87060             27      no     yes      91     51803
12         66683             19     yes     yes       6     50801
13         84559             29      no     yes      69     53057
14         76091             38     yes      no      32     59451
15         92696             49     yes      no      38     74381
16         59800             47      no     yes      27     44815
17         54836             25      no     yes      53     64601
18         70021             52     yes      no      28     95678
19         54368             11     yes     yes      20     55761

    citypartrange  numprevowners  made isnewbuilt hasstormprotector  basement  \
0               3              8  2005        old               yes      4313
1               6              8  2021        old                no      2937
2               3              4  2003        new                no      6326
3               8              3  2012        new               yes       632
4               4              2  2011        new               yes      5414
5              10              5  2018        old               yes      8871
6               6              9  2009        new               yes      4878
7               3              9  2011        new               yes      3054
```

```
8                3              7  2000        old             no      7507
9                7              9  1994        old             no       615
10               2              9  2008        new             no      2860
11               8             10  2000        old             no      6629
12               6              2  2001        old             no      7473
13               7              7  2000        new             no      3573
14               5              8  2016        new             no      8150
15               9              2  2021        old             no      1559
16               6              9  2021        old             no      5075
17              10              5  2020        new             no      5278
18               4              6  1992        old            yes      4480
19               3              7  2021        old             no       231

    attic  garage hasstorageroom  hasguestroom      price category
0    9005     956             no             7  7559081.5   Luxury
1    8852     135            yes             9  5574642.1   Middle
2    4748     654             no            10  8696869.3   Luxury
3    5792     807            yes             5  5154055.2   Middle
4    1172     716            yes             9  9652258.1   Luxury
5    7117     240             no             7  7986665.8   Luxury
6     281     384            yes             5  7607322.9   Luxury
7     129     726             no             9  6420823.1   Middle
8    9056     892            yes             1  9244344.0   Luxury
9    1221     328             no            10  9515440.4   Luxury
10   3129     982             no             1  7653300.8   Luxury
11    435     512             no             7  8711426.0   Luxury
12    796     237            yes             3  6677649.1   Middle
13   9556     918            yes             8  8460604.0   Luxury
14   6037     930             no             7  7614076.6   Luxury
15   5111     957            yes             2  9272740.1   Luxury
16   3104     864             no             4  5984462.1   Middle
17   1059     313            yes             6  5492532.0   Middle
18   6919     680            yes             1  7005572.2   Luxury
19   1939     223             no             8  5446398.1   Middle
```

[91]:

```python
df_uts2 = df_uts.drop(['category'],axis=1)
df_uts2.head()
```

[91]:

```
   squaremeters  numberofrooms hasyard haspool  floors  citycode  \
0         75523              3      no     yes      63      9373
1         55712             58      no     yes      19     34457
2         86929            100     yes      no      11     98155
3         51522              3      no      no      61      9047
4         96470             74     yes      no      21     92029

   citypartrange  numprevowners  made isnewbuilt hasstormprotector  basement  \
0              3              8  2005        old               yes      4313
```

```
1              6              8  2021       old                no      2937
2              3              4  2003       new                no      6326
3              8              3  2012       new               yes       632
4              4              2  2011       new               yes      5414

   attic  garage hasstorageroom  hasguestroom      price
0   9005     956             no             7  7559081.5
1   8852     135            yes             9  5574642.1
2   4748     654             no            10  8696869.3
3   5792     807            yes             5  5154055.2
4   1172     716            yes             9  9652258.1
```

[92]:
```
df_uts2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 17 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
dtypes: float64(1), int64(11), object(5)
memory usage: 1.3+ MB
```

[93]:
```
df_uts2.describe()
```

[93]:
```
       squaremeters  numberofrooms        floors      citycode  citypartrange  \
count   10000.00000   10000.000000  10000.000000  10000.000000   10000.000000
mean    49870.13120      50.358400     50.276300  50225.486100       5.510100
std     28774.37535      28.816696     28.889171  29006.675799       2.872024
min        89.00000       1.000000      1.000000      3.000000       1.000000
25%     25098.50000      25.000000     25.000000  24693.750000       3.000000
```

```
50%     50105.50000      50.000000      50.000000  50693.000000       5.000000
75%     74609.75000      75.000000      76.000000  75683.250000       8.000000
max     99999.00000     100.000000     100.000000  99953.000000      10.000000

       numprevowners          made      basement         attic        garage  \
count    10000.000000   10000.00000  10000.000000   10000.00000   10000.00000
mean         5.521700    2005.48850   5033.103900    5028.01060     553.12120
std          2.856667       9.30809   2876.729545    2894.33221     262.05017
min          1.000000    1990.00000      0.000000       1.00000     100.00000
25%          3.000000    1997.00000   2559.750000    2512.00000     327.75000
50%          5.000000    2005.50000   5092.500000    5045.00000     554.00000
75%          8.000000    2014.00000   7511.250000    7540.50000     777.25000
max         10.000000    2021.00000  10000.000000   10000.00000    1000.00000

       hasguestroom         price
count   10000.00000  1.000000e+04
mean        4.99460  4.993448e+06
std         3.17641  2.877424e+06
min         0.00000  1.031350e+04
25%         2.00000  2.516402e+06
50%         5.00000  5.016180e+06
75%         8.00000  7.469092e+06
max        10.00000  1.000677e+07
```

[94]:
```python
print(df_uts2['price'].value_counts())
```

```
price
7559081.5    1
2600292.1    1
3804577.4    1
3658559.7    1
2316639.4    1
            ..
5555606.6    1
5501007.5    1
9986201.2    1
9104801.8    1
146708.4     1
Name: count, Length: 10000, dtype: int64
```

[95]:
```python
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer
import pandas as pd


kolom_kategori = ['hasyard', 'haspool', 'isnewbuilt', 'hasstormprotector',␣
 ↪'hasstorageroom']
```

```python
transform = make_column_transformer(
    (OneHotEncoder(), kolom_kategori),
    remainder='passthrough'
)

# Transformasikan df_uts2
df_encoded = transform.fit_transform(df_uts2)

# Ambil nama kolom dari hasil OneHotEncoding
ohe_categories = transform.named_transformers_['onehotencoder'].
 ↪get_feature_names_out(kolom_kategori)

# Ambil kolom lainnya yang tidak diubah
remaining_columns = df_uts2.columns.difference(kolom_kategori).tolist()

# Gabungkan semua nama kolom
all_columns = list(ohe_categories) + remaining_columns

# Konversi hasil ke DataFrame dengan nama kolom yang benar
df_uts2 = pd.DataFrame(df_encoded, columns=all_columns)
```

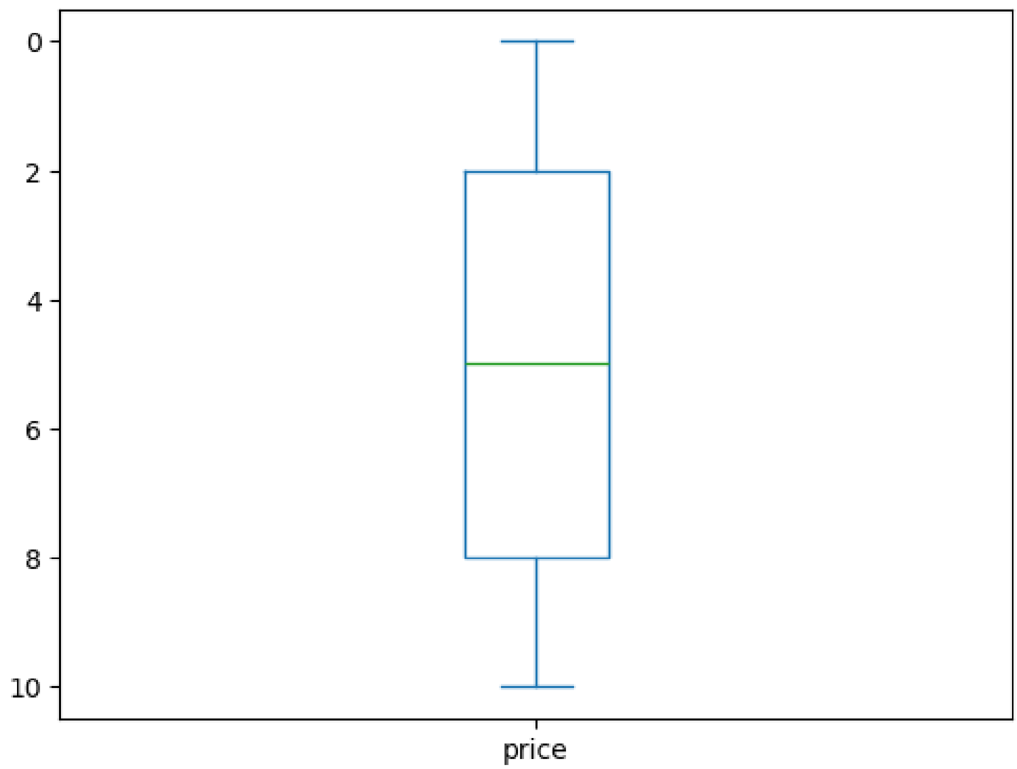
[96]:
```python
print("data null\n", df_uts2.isnull().sum())
print("data kosong\n",df_uts2.empty)
print("data nan \n", df_uts2.isna().sum())
```

```
data null
 hasyard_no              0
hasyard_yes             0
haspool_no              0
haspool_yes             0
isnewbuilt_new          0
isnewbuilt_old          0
hasstormprotector_no    0
hasstormprotector_yes   0
hasstorageroom_no       0
hasstorageroom_yes      0
attic                   0
basement                0
citycode                0
citypartrange           0
floors                  0
garage                  0
hasguestroom            0
made                    0
numberofrooms           0
numprevowners           0
```

```
price                      0
squaremeters               0
dtype: int64
data kosong
 False
data nan
 hasyard_no                 0
hasyard_yes                0
haspool_no                 0
haspool_yes                0
isnewbuilt_new             0
isnewbuilt_old             0
hasstormprotector_no       0
hasstormprotector_yes      0
hasstorageroom_no          0
hasstorageroom_yes         0
attic                      0
basement                   0
citycode                   0
citypartrange              0
floors                     0
garage                     0
hasguestroom               0
made                       0
numberofrooms              0
numprevowners              0
price                      0
squaremeters               0
dtype: int64
```

[97]:

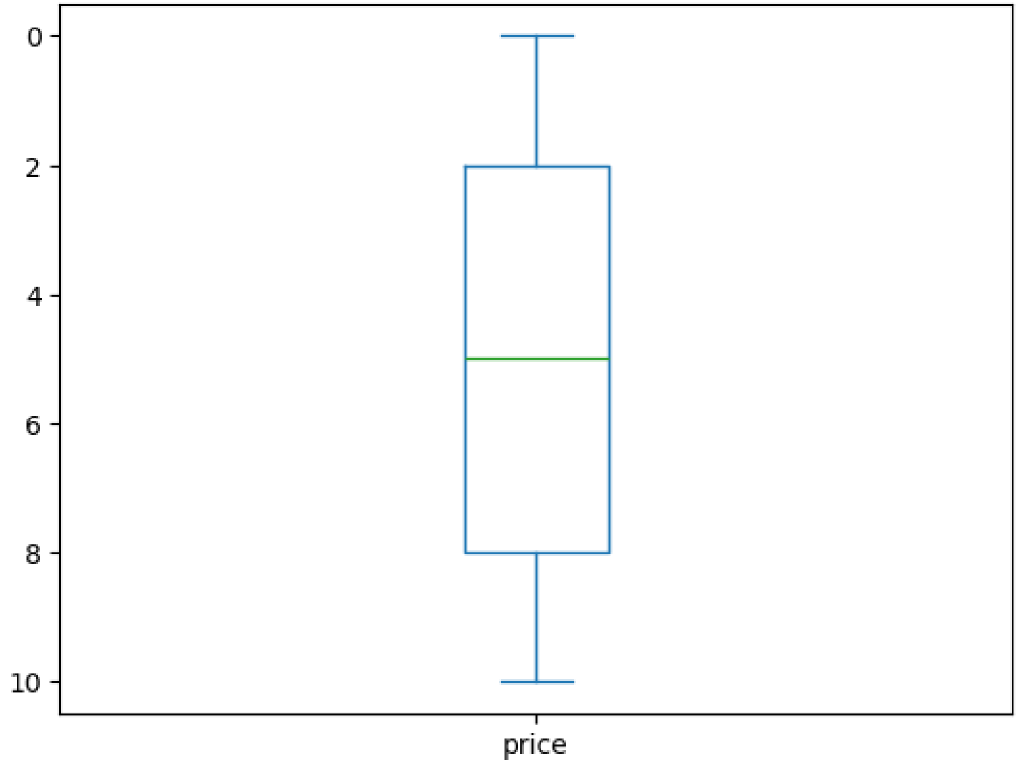
```
import matplotlib.pyplot as plt

df_uts2.price.plot(kind='box')
plt.gca().invert_yaxis()
plt.show()
```




[98]:
```
from pandas.api.types import is_numeric_dtype
def remove_outliner(df_in):
    for col_category in list(df_in.columns):
        q1 = df_in[col_category].quantile(0.25)
        q3 = df_in[col_category].quantile(0.75)

        iqr = q3-q1
        batas_atas = q3 + (1.5 * iqr)
        batas_bawah = q1 - (1.5 * iqr)

        df_out = df_in.loc[(df_in[col_category] >= batas_bawah) &␣
 ↪(df_in[col_category] <= batas_atas)]
    return df_out
df_uts_clean = remove_outliner(df_uts2)
print("Jumlah baris DataFrame sebelum dibuang outlier", df_uts2.shape[0])
print("Jumlah baris DataFrame sesudah dibuang outlier", df_uts_clean.shape[0])
df_uts_clean.price.plot(kind='box', vert = True)

plt.gca().invert_yaxis()
plt.show()
```

```
Jumlah baris DataFrame sebelum dibuang outlier 10000
```

Jumlah baris DataFrame sesudah dibuang outlier 10000




[99]:
```
df_uts2.describe()
```

[99]:

| | hasyard_no | hasyard_yes | haspool_no | haspool_yes | isnewbuilt_new | \ |
|---|---|---|---|---|---|---|
| count | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | |
| mean | 0.491300 | 0.508700 | 0.503200 | 0.496800 | 0.499100 | |
| std | 0.499949 | 0.499949 | 0.500015 | 0.500015 | 0.500024 | |
| min | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | |
| 25% | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | |
| 50% | 0.000000 | 1.000000 | 1.000000 | 0.000000 | 0.000000 | |
| 75% | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | |
| max | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | |

| | isnewbuilt_old | hasstormprotector_no | hasstormprotector_yes | \ |
|---|---|---|---|---|
| count | 10000.000000 | 10000.000000 | 10000.000000 | |
| mean | 0.500900 | 0.500100 | 0.499900 | |
| std | 0.500024 | 0.500025 | 0.500025 | |
| min | 0.000000 | 0.000000 | 0.000000 | |
| 25% | 0.000000 | 0.000000 | 0.000000 | |
| 50% | 1.000000 | 1.000000 | 0.000000 | |
| 75% | 1.000000 | 1.000000 | 1.000000 | |

```
max          1.000000             1.000000               1.000000

       hasstorageroom_no  hasstorageroom_yes  …      citycode  \
count       10000.000000        10000.000000  …  10000.000000
mean            0.497000            0.503000  …     50.276300
std             0.500016            0.500016  …     28.889171
min             0.000000            0.000000  …      1.000000
25%             0.000000            0.000000  …     25.000000
50%             0.000000            1.000000  …     50.000000
75%             1.000000            1.000000  …     76.000000
max             1.000000            1.000000  …    100.000000

       citypartrange        floors        garage  hasguestroom          made  \
count   10000.000000  10000.000000  10000.000000   10000.00000  10000.000000
mean    50225.486100      5.510100      5.521700    2005.48850   5033.103900
std     29006.675799      2.872024      2.856667       9.30809   2876.729545
min         3.000000      1.000000      1.000000    1990.00000      0.000000
25%     24693.750000      3.000000      3.000000    1997.00000   2559.750000
50%     50693.000000      5.000000      5.000000    2005.50000   5092.500000
75%     75683.250000      8.000000      8.000000    2014.00000   7511.250000
max     99953.000000     10.000000     10.000000    2021.00000  10000.000000

       numberofrooms  numprevowners        price  squaremeters
count    10000.00000    10000.00000  10000.00000  1.000000e+04
mean      5028.01060      553.12120      4.99460  4.993448e+06
std       2894.33221      262.05017      3.17641  2.877424e+06
min          1.00000      100.00000      0.00000  1.031350e+04
25%       2512.00000      327.75000      2.00000  2.516402e+06
50%       5045.00000      554.00000      5.00000  5.016180e+06
75%       7540.50000      777.25000      8.00000  7.469092e+06
max      10000.00000     1000.00000     10.00000  1.000677e+07

[8 rows x 22 columns]
```

[100]:
```
print("data null\n", df_uts_clean.isnull().sum())
print("data kosong \n", df_uts_clean.empty)
print("data nan \n", df_uts_clean.isna().sum())
```

```
data null
 hasyard_no                0
hasyard_yes               0
haspool_no                0
haspool_yes               0
isnewbuilt_new            0
isnewbuilt_old            0
hasstormprotector_no      0
hasstormprotector_yes     0
```

```
hasstorageroom_no        0
hasstorageroom_yes       0
attic                    0
basement                 0
citycode                 0
citypartrange            0
floors                   0
garage                   0
hasguestroom             0
made                     0
numberofrooms            0
numprevowners            0
price                    0
squaremeters             0
dtype: int64
data kosong
 False
data nan
 hasyard_no               0
hasyard_yes              0
haspool_no               0
haspool_yes              0
isnewbuilt_new           0
isnewbuilt_old           0
hasstormprotector_no     0
hasstormprotector_yes    0
hasstorageroom_no        0
hasstorageroom_yes       0
attic                    0
basement                 0
citycode                 0
citypartrange            0
floors                   0
garage                   0
hasguestroom             0
made                     0
numberofrooms            0
numprevowners            0
price                    0
squaremeters             0
dtype: int64
```

[101]:
```
from sklearn.model_selection import train_test_split

X_regress = df_uts_clean.drop('price',axis=1)
y_regress = df_uts_clean.price
```

```
X_train_price, X_test_price, y_train_price, y_test_price =␣
↪train_test_split(X_regress, y_regress, test_size=0.25, random_state=83)
```

[102]:
```
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.metrics import mean_absolute_error, mean_squared_error

pipe_Lasso1 = Pipeline(steps=[
    ('scale',  StandardScaler()),
    ('feature_selection', SelectKBest(score_func= f_regression)),
    ('reg',Lasso(max_iter=1000))
    ])
param_grid_Lasso1 = {
    'reg__alpha': [0.01,0.1,1,10,100],
    'feature_selection__k': np.arange(1,20)
}

GSCV_Lasso1 = GridSearchCV(pipe_Lasso1, param_grid_Lasso1, cv=5,
                       scoring= 'neg_mean_squared_error',n_jobs=-1)
GSCV_Lasso1.fit(X_train_price, y_train_price)

print("Best model: {}".format(GSCV_Lasso1.best_estimator_))
print("Lasso best parameters :{}".format(GSCV_Lasso1.best_params_))
print("Koefisien/bobot:{}".format(GSCV_Lasso1.best_estimator_.
↪named_steps['reg'].coef_))
print("Intercept/bias:{}".format(GSCV_Lasso1.best_estimator_.named_steps['reg'].
↪intercept_))

Lasso_predict1 = GSCV_Lasso1.predict(X_test_price)
mse_Lasso1 = mean_squared_error(y_test_price, Lasso_predict1)
mae_Lasso1 = mean_absolute_error(y_test_price, Lasso_predict1)

print("Lasso Mean Squared Error (MSE): {}".format(mse_Lasso1))
print("Lasso Mean Absolute Error (MAE): {}".format(mae_Lasso1))
print("Lasso Root Mean Squared Error: {}".format(np.sqrt(mse_Lasso1)))
```

```
Best model: Pipeline(steps=[('scale', StandardScaler()),
                ('feature_selection',
                 SelectKBest(k=1,
                             score_func=<function f_regression at
0x000001E9660F04A0>)),
                ('reg', Lasso(alpha=1))])
Lasso best parameters :{'feature_selection__k': 1, 'reg__alpha': 1}
Koefisien/bobot:[-0.]
```

```
Intercept/bias:4.966533333333335
Lasso Mean Squared Error (MSE): 10.188394364444445
Lasso Mean Absolute Error (MAE): 2.7679608533333333
Lasso Root Mean Squared Error: 3.191926434685556
```

[103]:
```
df_results = pd.DataFrame(y_test_price, columns=['price'])
df_results = pd.DataFrame(y_test_price)
df_results['Lasso Prediction1'] = Lasso_predict1
df_results['Selisih_price_LR1'] = df_results['Lasso Prediction1'] -␣
↪df_results['price']

df_results.head()
```

[103]:

| | price | Lasso Prediction1 | Selisih_price_LR1 |
|---|---|---|---|
| 2353 | 4.0 | 4.966533 | 0.966533 |
| 2050 | 10.0 | 4.966533 | -5.033467 |
| 3276 | 3.0 | 4.966533 | 1.966533 |
| 4297 | 5.0 | 4.966533 | -0.033467 |
| 9322 | 0.0 | 4.966533 | 4.966533 |

[104]:
```
df_results.describe()
```

[104]:

| | price | Lasso Prediction1 | Selisih_price_LR1 |
|---|---|---|---|
| count | 2500.00000 | 2.500000e+03 | 2500.000000 |
| mean | 5.07880 | 4.966533e+00 | -0.112267 |
| std | 3.19059 | 2.309726e-13 | 3.190590 |
| min | 0.00000 | 4.966533e+00 | -5.033467 |
| 25% | 2.00000 | 4.966533e+00 | -3.033467 |
| 50% | 5.00000 | 4.966533e+00 | -0.033467 |
| 75% | 8.00000 | 4.966533e+00 | 2.966533 |
| max | 10.00000 | 4.966533e+00 | 4.966533 |

[105]:
```
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import SelectPercentile, f_regression
from sklearn.metrics import mean_absolute_error, mean_squared_error

pipe_Lasso2 = Pipeline(steps=[
    ('scale',  MinMaxScaler()),
    ('feature_selection', SelectPercentile(score_func= f_regression)),
    ('reg',Lasso(max_iter=1000))
    ])
param_grid_Lasso2 = {
    'reg__alpha': [0.01,0.1,1,10,100],
    'feature_selection__percentile': np.arange(1,100,5)
```

```
}

GSCV_Lasso2 = GridSearchCV(pipe_Lasso2, param_grid_Lasso2, cv=5,
                       scoring= 'neg_mean_squared_error',n_jobs=-1)
GSCV_Lasso2.fit(X_train_price, y_train_price)

print("Best model: {}".format(GSCV_Lasso2.best_estimator_))
print("Lasso best parameters :{}".format(GSCV_Lasso2.best_params_))
print("Koefisien/bobot:{}".format(GSCV_Lasso2.best_estimator_.
 ↪named_steps['reg'].coef_))
print("Intercept/bias:{}".format(GSCV_Lasso2.best_estimator_.named_steps['reg'].
 ↪intercept_))

Lasso_predict2 = GSCV_Lasso2.predict(X_test_price)
mse_Lasso2 = mean_squared_error(y_test_price, Lasso_predict2)
mae_Lasso2 = mean_absolute_error(y_test_price, Lasso_predict2)

print("Lasso Mean Squared Error (MSE): {}".format(mse_Lasso2))
print("Lasso Mean Absolute Error (MAE): {}".format(mae_Lasso2))
print("Lasso Root Mean Squared Error: {}".format(np.sqrt(mse_Lasso2)))
```

```
Best model: Pipeline(steps=[('scale', MinMaxScaler()),
                ('feature_selection',
                 SelectPercentile(percentile=1,
                                  score_func=<function f_regression at
0x000001E9660F04A0>)),
                ('reg', Lasso(alpha=0.1))])
Lasso best parameters :{'feature_selection__percentile': 1, 'reg__alpha': 0.1}
Koefisien/bobot:[0.]
Intercept/bias:4.9665333333333335
Lasso Mean Squared Error (MSE): 10.188394364444445
Lasso Mean Absolute Error (MAE): 2.7679608533333333
Lasso Root Mean Squared Error: 3.191926434685556
```

[106]:
```
df_results = pd.DataFrame(y_test_price, columns=['price'])
df_results = pd.DataFrame(y_test_price)
df_results['Lasso Prediction2'] = Lasso_predict2
df_results['Selisih_price_LR2'] = df_results['Lasso Prediction2'] -␣
 ↪df_results['price']

df_results.head()
```

[106]:

| | price | Lasso Prediction2 | Selisih_price_LR2 |
|---|---|---|---|
| 2353 | 4.0 | 4.966533 | 0.966533 |
| 2050 | 10.0 | 4.966533 | -5.033467 |
| 3276 | 3.0 | 4.966533 | 1.966533 |
| 4297 | 5.0 | 4.966533 | -0.033467 |

```
9322    0.0          4.966533          4.966533
```

[107]:
```
df_results.describe()
```

[107]:

| | price | Lasso Prediction2 | Selisih_price_LR2 |
|---|---|---|---|
| count | 2500.00000 | 2.500000e+03 | 2500.000000 |
| mean | 5.07880 | 4.966533e+00 | -0.112267 |
| std | 3.19059 | 2.309726e-13 | 3.190590 |
| min | 0.00000 | 4.966533e+00 | -5.033467 |
| 25% | 2.00000 | 4.966533e+00 | -3.033467 |
| 50% | 5.00000 | 4.966533e+00 | -0.033467 |
| 75% | 8.00000 | 4.966533e+00 | 2.966533 |
| max | 10.00000 | 4.966533e+00 | 4.966533 |

[108]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

# Pipeline untuk Random Forest Regressor
pipe_RF1 = Pipeline(steps=[
    ('scale', StandardScaler()),  # Penskalaan data
    ('feature_selection', SelectKBest(score_func=f_regression)),  # Pemilihan␣
 ↪fitur
    ('reg', RandomForestRegressor())
])

# Parameter grid untuk GridSearchCV
param_grid_RF1 = {
    'reg__n_estimators': [50, 100, 200],
    'reg__max_depth': [None, 10, 20, 30],  # Kedalaman maksimum pohon
    'feature_selection__k': np.arange(1, 20)  # Jumlah fitur yang dipilih
}

# GridSearchCV untuk Random Forest
GSCV_RF1 = GridSearchCV(pipe_RF1, param_grid_RF1, cv=5,
                        scoring='neg_mean_squared_error', n_jobs=-1)

# Fit model dengan data pelatihan
GSCV_RF1.fit(X_train_price, y_train_price)

# Menampilkan hasil terbaik
print("Best model: {}".format(GSCV_RF1.best_estimator_))
print("Random Forest best parameters: {}".format(GSCV_RF1.best_params_))
```

```python
# Melakukan prediksi dengan model terbaik
RF_predict1 = GSCV_RF1.predict(X_test_price)
mse_RF1 = mean_squared_error(y_test_price, RF_predict1)
mae_RF1 = mean_absolute_error(y_test_price, RF_predict1)

# Menampilkan metrik performa
print("Random Forest Mean Squared Error (MSE): {}".format(mse_RF1))
print("Random Forest Mean Absolute Error (MAE): {}".format(mae_RF1))
print("Random Forest Root Mean Squared Error: {}".format(np.sqrt(mse_RF1)))
```

```
Best model: Pipeline(steps=[('scale', StandardScaler()),
                ('feature_selection',
                 SelectKBest(k=15,
                             score_func=<function f_regression at
0x000001E9660F04A0>)),
                ('reg', RandomForestRegressor(max_depth=10, n_estimators=200))])
Random Forest best parameters: {'feature_selection__k': 15, 'reg__max_depth':
10, 'reg__n_estimators': 200}
Random Forest Mean Squared Error (MSE): 10.27240052040819
Random Forest Mean Absolute Error (MAE): 2.7849240576479826
Random Forest Root Mean Squared Error: 3.2050585829916107
```

[109]:

```python
df_results = pd.DataFrame({'price': y_test_price})
RFR_predict1 = GSCV_RF1.predict(X_test_price)
df_results['Rfr Prediction1'] = RFR_predict1
df_results['Selisih_price_Rfr1'] = df_results['Rfr Prediction1'] -
  ↪df_results['price']
print(df_results.head())
```

```
      price  Rfr Prediction1  Selisih_price_Rfr1
2353    4.0         4.740753            0.740753
2050   10.0         5.614398           -4.385602
3276    3.0         4.838944            1.838944
4297    5.0         4.922992           -0.077008
9322    0.0         4.916289            4.916289
```

[110]:

```python
df_results.describe()
```

[110]:

| | price | Rfr Prediction1 | Selisih_price_Rfr1 |
|---|---|---|---|
| count | 2500.00000 | 2500.000000 | 2500.000000 |
| mean | 5.07880 | 4.955712 | -0.123088 |
| std | 3.19059 | 0.269223 | 3.203335 |
| min | 0.00000 | 3.713565 | -6.067737 |
| 25% | 2.00000 | 4.797611 | -2.980137 |
| 50% | 5.00000 | 4.959285 | -0.113735 |
| 75% | 8.00000 | 5.108439 | 2.799299 |

```
max      10.00000         6.086543            5.887084
```

[111]:
```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import SelectPercentile, f_regression
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

# Pipeline untuk Random Forest Regressor
pipe_RF2 = Pipeline(steps=[
    ('scale', MinMaxScaler()),  # Penskalaan data
    ('feature_selection', SelectPercentile(score_func=f_regression)),  #
 ↪Pemilihan fitur
    ('reg', RandomForestRegressor())
])

# Parameter grid untuk GridSearchCV
param_grid_RF2 = {
    'reg__n_estimators': [50, 100, 200],
    'reg__max_depth': [None, 10, 20, 30],
    'feature_selection__percentile': np.arange(1, 100, 5)
}

# GridSearchCV untuk Random Forest
GSCV_RF2 = GridSearchCV(pipe_RF2, param_grid_RF2, cv=5,
                        scoring='neg_mean_squared_error', n_jobs=-1)

# Fit model dengan data pelatihan
GSCV_RF2.fit(X_train_price, y_train_price)

# Menampilkan hasil terbaik
print("Best model: {}".format(GSCV_RF2.best_estimator_))
print("Random Forest best parameters: {}".format(GSCV_RF2.best_params_))

# Melakukan prediksi dengan model terbaik
RF_predict2 = GSCV_RF2.predict(X_test_price)
mse_RF2 = mean_squared_error(y_test_price, RF_predict2)
mae_RF2 = mean_absolute_error(y_test_price, RF_predict2)

# Menampilkan metrik performa
print("Random Forest Mean Squared Error (MSE): {}".format(mse_RF2))
print("Random Forest Mean Absolute Error (MAE): {}".format(mae_RF2))
print("Random Forest Root Mean Squared Error: {}".format(np.sqrt(mse_RF2)))
```

```
Best model: Pipeline(steps=[('scale', MinMaxScaler()),
```

```
                ('feature_selection',
                 SelectPercentile(percentile=86,
                                  score_func=<function f_regression at
0x000001E9660F04A0>)),
                ('reg', RandomForestRegressor(max_depth=10, n_estimators=50))])
Random Forest best parameters: {'feature_selection__percentile': 86,
'reg__max_depth': 10, 'reg__n_estimators': 50}
Random Forest Mean Squared Error (MSE): 10.307742902882746
Random Forest Mean Absolute Error (MAE): 2.7917007587756952
Random Forest Root Mean Squared Error: 3.2105673802122183
```

[112]:
```
df_results = pd.DataFrame({'price': y_test_price})
RFR_predict2 = GSCV_RF2.predict(X_test_price)
df_results['Rfr Prediction2'] = RFR_predict2
df_results['Selisih_price_Rfr2'] = df_results['Rfr Prediction2'] -␣
 ↪df_results['price']
print(df_results.head())
```

```
      price  Rfr Prediction2  Selisih_price_Rfr2
2353    4.0         4.719914            0.719914
2050   10.0         5.848616           -4.151384
3276    3.0         5.049430            2.049430
4297    5.0         4.971565           -0.028435
9322    0.0         4.704022            4.704022
```

[113]:
```
df_results.describe()
```

[113]:

| | price | Rfr Prediction2 | Selisih_price_Rfr2 |
|---|---|---|---|
| count | 2500.00000 | 2500.000000 | 2500.000000 |
| mean | 5.07880 | 4.967832 | -0.110968 |
| std | 3.19059 | 0.312743 | 3.209291 |
| min | 0.00000 | 3.457008 | -6.080234 |
| 25% | 2.00000 | 4.780487 | -2.964763 |
| 50% | 5.00000 | 4.963642 | -0.075615 |
| 75% | 8.00000 | 5.150321 | 2.801684 |
| max | 10.00000 | 6.483074 | 6.162311 |

[114]:
```
df_results = pd.DataFrame({'price' : y_test_price})

df_results['Lasso Prediction1'] = Lasso_predict1
df_results['Selisih_price_LR1'] = df_results['price'] - df_results['Lasso␣
 ↪Prediction1']

df_results['Lasso Prediction2'] = Lasso_predict2
df_results['Selisih_price_LR2'] = df_results['price'] - df_results['Lasso␣
 ↪Prediction2']
```

```
df_results['Rfr Prediction1'] = RFR_predict1
df_results['Selisih_price_Rfr1'] = df_results['price'] - df_results['Rfr␣
↪Prediction1']

df_results['Rfr Prediction2'] = RFR_predict2
df_results['Selisih_price_Rfr2'] = df_results['price'] - df_results['Rfr␣
↪Prediction2']

df_results.head()
```

[114]:

```
      price  Lasso Prediction1  Selisih_price_LR1  Lasso Prediction2  \
2353    4.0           4.966533          -0.966533           4.966533
2050   10.0           4.966533           5.033467           4.966533
3276    3.0           4.966533          -1.966533           4.966533
4297    5.0           4.966533           0.033467           4.966533
9322    0.0           4.966533          -4.966533           4.966533

      Selisih_price_LR2  Rfr Prediction1  Selisih_price_Rfr1  Rfr Prediction2  \
2353          -0.966533         4.740753           -0.740753         4.719914
2050           5.033467         5.614398            4.385602         5.848616
3276          -1.966533         4.838944           -1.838944         5.049430
4297           0.033467         4.922992            0.077008         4.971565
9322          -4.966533         4.916289           -4.916289         4.704022

      Selisih_price_Rfr2
2353           -0.719914
2050            4.151384
3276           -2.049430
4297            0.028435
9322           -4.704022
```

[115]:

```
df_results.describe()
```

[115]:

```
            price  Lasso Prediction1  Selisih_price_LR1  Lasso Prediction2  \
count  2500.00000       2.500000e+03        2500.000000       2.500000e+03
mean      5.07880       4.966533e+00           0.112267       4.966533e+00
std       3.19059       2.309726e-13           3.190590       2.309726e-13
min       0.00000       4.966533e+00          -4.966533       4.966533e+00
25%       2.00000       4.966533e+00          -2.966533       4.966533e+00
50%       5.00000       4.966533e+00           0.033467       4.966533e+00
75%       8.00000       4.966533e+00           3.033467       4.966533e+00
max      10.00000       4.966533e+00           5.033467       4.966533e+00

       Selisih_price_LR2  Rfr Prediction1  Selisih_price_Rfr1  \
count        2500.000000      2500.000000         2500.000000
mean            0.112267         4.955712            0.123088
std             3.190590         0.269223            3.203335
```

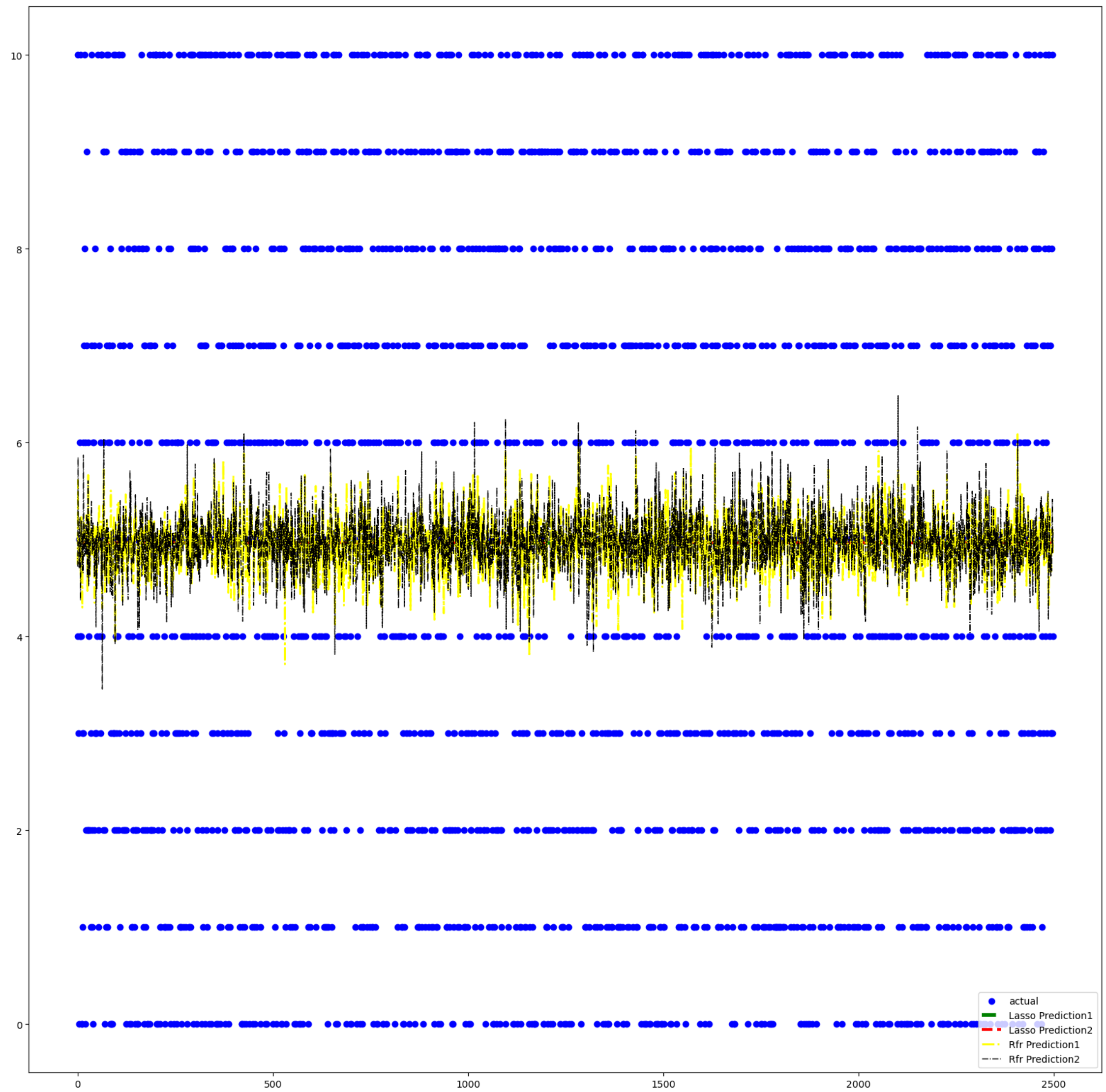
```
min           -4.966533         3.713565           -5.887084
25%           -2.966533         4.797611           -2.799299
50%            0.033467         4.959285            0.113735
75%            3.033467         5.108439            2.980137
max            5.033467         6.086543            6.067737

       Rfr Prediction2  Selisih_price_Rfr2
count      2500.000000         2500.000000
mean          4.967832            0.110968
std           0.312743            3.209291
min           3.457008           -6.162311
25%           4.780487           -2.801684
50%           4.963642            0.075615
75%           5.150321            2.964763
max           6.483074            6.080234
```

[116]:

```
plt.figure(figsize=(20,20))

data_len = range(len(y_test_price))

plt.scatter(data_len, df_results.price, label= "actual", color="blue")
plt.plot(data_len, df_results['Lasso Prediction1'], label="Lasso Prediction1",␣
 ↪color="green", linewidth=4, linestyle="dashed")
plt.plot(data_len, df_results['Lasso Prediction2'], label="Lasso Prediction2",␣
 ↪color="red", linewidth=3, linestyle="dashed")
plt.plot(data_len, df_results['Rfr Prediction1'], label= "Rfr Prediction1",␣
 ↪color="yellow", linewidth= 2, linestyle= "-.")
plt.plot(data_len, df_results['Rfr Prediction2'], label= "Rfr Prediction2",␣
 ↪color="black", linewidth= 1, linestyle= "-.")
plt.legend()
plt.show
```

[116]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```




[117]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

mae_Lasso1 = mean_absolute_error(df_results['price'], df_results['Lasso␣
↪Prediction1'])
rmse_Lasso1 = np.sqrt(mean_absolute_error(df_results['price'],df_results['Lasso␣
↪Prediction1']))
Lasso_feature_count1 = GSCV_Lasso1.best_params_.get('feature_selection__k')

mae_Lasso2 = mean_absolute_error(df_results['price'], df_results['Lasso␣
↪Prediction2'])
rmse_Lasso2 = np.sqrt(mean_absolute_error(df_results['price'],df_results['Lasso␣
↪Prediction2']))
```

```
Lasso_feature_count2 = GSCV_Lasso2.best_params_['feature_selection__percentile']

mae_RF1 = mean_absolute_error(df_results['price'], df_results['Rfr␣
↪Prediction1'])
rmse_RF1 = np.sqrt(mean_squared_error(df_results['price'], df_results['Rfr␣
↪Prediction1']))
RF_feature_count1 = GSCV_RF1.best_params_.get('feature_selection__k')


mae_RF2 = mean_absolute_error(df_results['price'], df_results['Rfr␣
↪Prediction2'])
rmse_RF2 = np.sqrt(mean_squared_error(df_results['price'], df_results['Rfr␣
↪Prediction2']))
RF_feature_count2 = GSCV_RF2.best_params_['feature_selection__percentile']

print(f"Lasso MAE 1: {mae_Lasso1}, Lasso RMSE 1: {rmse_Lasso1}, Lasso Feature␣
↪Count 1: {Lasso_feature_count1}")
print(f"Lasso MAE 2: {mae_Lasso2}, Lasso RMSE 2: {rmse_Lasso2}, Lasso Feature␣
↪Count 2: {Lasso_feature_count2}")
print(f"RF MAE 1: {mae_RF1}, RF RMSE 1: {rmse_RF1}, RF Feature Count 1:␣
↪{RF_feature_count1}")
print(f"RF MAE 2: {mae_RF2}, RF RMSE 2: {rmse_RF2}, RF Feature Count 2:␣
↪{RF_feature_count2}")
```

```
Lasso MAE 1: 2.7679608533333333, Lasso RMSE 1: 1.6637189826810697, Lasso Feature
Count 1: 1
Lasso MAE 2: 2.7679608533333333, Lasso RMSE 2: 1.6637189826810697, Lasso Feature
Count 2: 1
RF MAE 1: 2.7849240576479826, RF RMSE 1: 3.2050585829916107, RF Feature Count 1:
15
RF MAE 2: 2.7917007587756952, RF RMSE 2: 3.2105673802122183, RF Feature Count 2:
86
```

# ensorflow-ridge-vs-svr-albert-kent

October 25, 2024

[2]:
```python
import pandas as pd
import numpy as np

df_uts = pd.read_csv(r'Dataset UTS_Gasal 2425.csv')
df_uts.head(20)
```

[2]:

| | squaremeters | numberofrooms | hasyard | haspool | floors | citycode | \ |
|---|---|---|---|---|---|---|---|
| 0 | 75523 | 3 | no | yes | 63 | 9373 | |
| 1 | 55712 | 58 | no | yes | 19 | 34457 | |
| 2 | 86929 | 100 | yes | no | 11 | 98155 | |
| 3 | 51522 | 3 | no | no | 61 | 9047 | |
| 4 | 96470 | 74 | yes | no | 21 | 92029 | |
| 5 | 79770 | 3 | no | yes | 69 | 54812 | |
| 6 | 75985 | 60 | yes | no | 67 | 6517 | |
| 7 | 64169 | 88 | no | yes | 6 | 61711 | |
| 8 | 92383 | 12 | no | no | 78 | 71982 | |
| 9 | 95121 | 46 | no | yes | 3 | 9382 | |
| 10 | 76485 | 47 | yes | no | 9 | 90254 | |
| 11 | 87060 | 27 | no | yes | 91 | 51803 | |
| 12 | 66683 | 19 | yes | yes | 6 | 50801 | |
| 13 | 84559 | 29 | no | yes | 69 | 53057 | |
| 14 | 76091 | 38 | yes | no | 32 | 59451 | |
| 15 | 92696 | 49 | yes | no | 38 | 74381 | |
| 16 | 59800 | 47 | no | yes | 27 | 44815 | |
| 17 | 54836 | 25 | no | yes | 53 | 64601 | |
| 18 | 70021 | 52 | yes | no | 28 | 95678 | |
| 19 | 54368 | 11 | yes | yes | 20 | 55761 | |

| | citypartrange | numprevowners | made | isnewbuilt | hasstormprotector | basement | \ |
|---|---|---|---|---|---|---|---|
| 0 | 3 | 8 | 2005 | old | yes | 4313 | |
| 1 | 6 | 8 | 2021 | old | no | 2937 | |
| 2 | 3 | 4 | 2003 | new | no | 6326 | |
| 3 | 8 | 3 | 2012 | new | yes | 632 | |
| 4 | 4 | 2 | 2011 | new | yes | 5414 | |
| 5 | 10 | 5 | 2018 | old | yes | 8871 | |
| 6 | 6 | 9 | 2009 | new | yes | 4878 | |
| 7 | 3 | 9 | 2011 | new | yes | 3054 | |

```
8               3              7  2000        old              no      7507
9               7              9  1994        old              no       615
10              2              9  2008        new              no      2860
11              8             10  2000        old              no      6629
12              6              2  2001        old              no      7473
13              7              7  2000        new              no      3573
14              5              8  2016        new              no      8150
15              9              2  2021        old              no      1559
16              6              9  2021        old              no      5075
17             10              5  2020        new              no      5278
18              4              6  1992        old             yes      4480
19              3              7  2021        old              no       231

    attic  garage hasstorageroom  hasguestroom      price category
0    9005     956             no             7  7559081.5   Luxury
1    8852     135            yes             9  5574642.1   Middle
2    4748     654             no            10  8696869.3   Luxury
3    5792     807            yes             5  5154055.2   Middle
4    1172     716            yes             9  9652258.1   Luxury
5    7117     240             no             7  7986665.8   Luxury
6     281     384            yes             5  7607322.9   Luxury
7     129     726             no             9  6420823.1   Middle
8    9056     892            yes             1  9244344.0   Luxury
9    1221     328             no            10  9515440.4   Luxury
10   3129     982             no             1  7653300.8   Luxury
11    435     512             no             7  8711426.0   Luxury
12    796     237            yes             3  6677649.1   Middle
13   9556     918            yes             8  8460604.0   Luxury
14   6037     930             no             7  7614076.6   Luxury
15   5111     957            yes             2  9272740.1   Luxury
16   3104     864             no             4  5984462.1   Middle
17   1059     313            yes             6  5492532.0   Middle
18   6919     680            yes             1  7005572.2   Luxury
19   1939     223             no             8  5446398.1   Middle
```

[3]:
```python
df_uts2 = df_uts.drop(['category'],axis=1)
df_uts2.head()
```

[3]:
```
   squaremeters  numberofrooms hasyard haspool  floors  citycode  \
0         75523              3      no     yes      63      9373
1         55712             58      no     yes      19     34457
2         86929            100     yes      no      11     98155
3         51522              3      no      no      61      9047
4         96470             74     yes      no      21     92029

   citypartrange  numprevowners  made isnewbuilt hasstormprotector  basement  \
0              3              8  2005        old               yes      4313
```

```
1            6           8  2021         old                  no      2937
2            3           4  2003         new                  no      6326
3            8           3  2012         new                 yes       632
4            4           2  2011         new                 yes      5414

   attic  garage hasstorageroom  hasguestroom      price
0   9005     956             no             7  7559081.5
1   8852     135            yes             9  5574642.1
2   4748     654             no            10  8696869.3
3   5792     807            yes             5  5154055.2
4   1172     716            yes             9  9652258.1
```

[4]:
```
df_uts2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 17 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
dtypes: float64(1), int64(11), object(5)
memory usage: 1.3+ MB
```

[5]:
```
df_uts2.describe()
```

[5]:
```
       squaremeters  numberofrooms         floors      citycode  citypartrange  \
count   10000.00000   10000.000000   10000.000000  10000.000000   10000.000000
mean    49870.13120      50.358400      50.276300  50225.486100       5.510100
std     28774.37535      28.816696      28.889171  29006.675799       2.872024
min        89.00000       1.000000       1.000000      3.000000       1.000000
25%     25098.50000      25.000000      25.000000  24693.750000       3.000000
```

```
50%      50105.50000      50.000000     50.000000  50693.000000       5.000000
75%      74609.75000      75.000000     76.000000  75683.250000       8.000000
max      99999.00000     100.000000    100.000000  99953.000000      10.000000

       numprevowners         made      basement        attic       garage  \
count   10000.000000  10000.00000  10000.000000  10000.00000  10000.00000
mean        5.521700   2005.48850   5033.103900   5028.01060    553.12120
std         2.856667      9.30809   2876.729545   2894.33221    262.05017
min         1.000000   1990.00000      0.000000      1.00000    100.00000
25%         3.000000   1997.00000   2559.750000   2512.00000    327.75000
50%         5.000000   2005.50000   5092.500000   5045.00000    554.00000
75%         8.000000   2014.00000   7511.250000   7540.50000    777.25000
max        10.000000   2021.00000  10000.000000  10000.00000   1000.00000

       hasguestroom         price
count   10000.00000  1.000000e+04
mean        4.99460  4.993448e+06
std         3.17641  2.877424e+06
min         0.00000  1.031350e+04
25%         2.00000  2.516402e+06
50%         5.00000  5.016180e+06
75%         8.00000  7.469092e+06
max        10.00000  1.000677e+07
```

[6]:
```
print(df_uts2['price'].value_counts())
```

```
price
7559081.5    1
2600292.1    1
3804577.4    1
3658559.7    1
2316639.4    1
            ..
5555606.6    1
5501007.5    1
9986201.2    1
9104801.8    1
146708.4     1
Name: count, Length: 10000, dtype: int64
```

[7]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer
import pandas as pd


kolom_kategori = ['hasyard', 'haspool', 'isnewbuilt', 'hasstormprotector',␣
 ↪'hasstorageroom']
```

```
transform = make_column_transformer(
    (OneHotEncoder(), kolom_kategori),
    remainder='passthrough'
)

# Transformasikan df_uts2
df_encoded = transform.fit_transform(df_uts2)

# Ambil nama kolom dari hasil OneHotEncoding
ohe_categories = transform.named_transformers_['onehotencoder'].
 ↪get_feature_names_out(kolom_kategori)

# Ambil kolom lainnya yang tidak diubah
remaining_columns = df_uts2.columns.difference(kolom_kategori).tolist()

# Gabungkan semua nama kolom
all_columns = list(ohe_categories) + remaining_columns

# Konversi hasil ke DataFrame dengan nama kolom yang benar
df_uts2 = pd.DataFrame(df_encoded, columns=all_columns)
```

[8]:

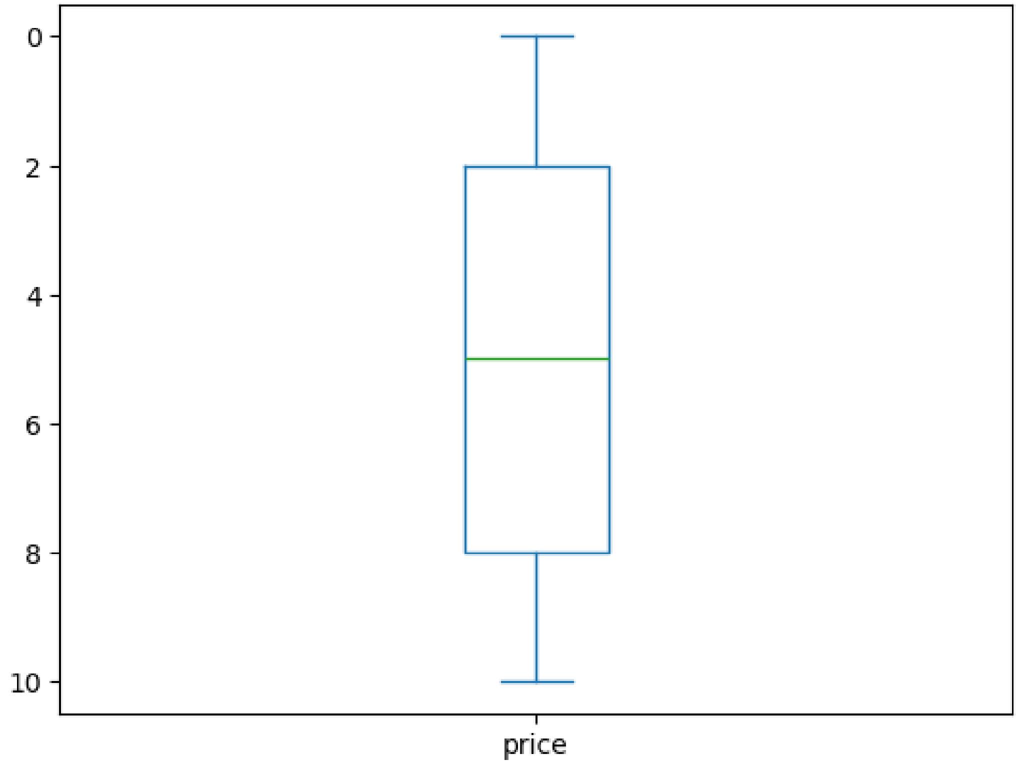
```
print("data null\n", df_uts2.isnull().sum())
print("data kosong\n",df_uts2.empty)
print("data nan \n", df_uts2.isna().sum())
```

```
data null
 hasyard_no              0
hasyard_yes             0
haspool_no              0
haspool_yes             0
isnewbuilt_new          0
isnewbuilt_old          0
hasstormprotector_no    0
hasstormprotector_yes   0
hasstorageroom_no       0
hasstorageroom_yes      0
attic                   0
basement                0
citycode                0
citypartrange           0
floors                  0
garage                  0
hasguestroom            0
made                    0
numberofrooms           0
numprevowners           0
```

```
price                     0
squaremeters              0
dtype: int64
data kosong
 False
data nan
 hasyard_no                0
hasyard_yes               0
haspool_no                0
haspool_yes               0
isnewbuilt_new            0
isnewbuilt_old            0
hasstormprotector_no      0
hasstormprotector_yes     0
hasstorageroom_no         0
hasstorageroom_yes        0
attic                     0
basement                  0
citycode                  0
citypartrange             0
floors                    0
garage                    0
hasguestroom              0
made                      0
numberofrooms             0
numprevowners             0
price                     0
squaremeters              0
dtype: int64
```

[9]:

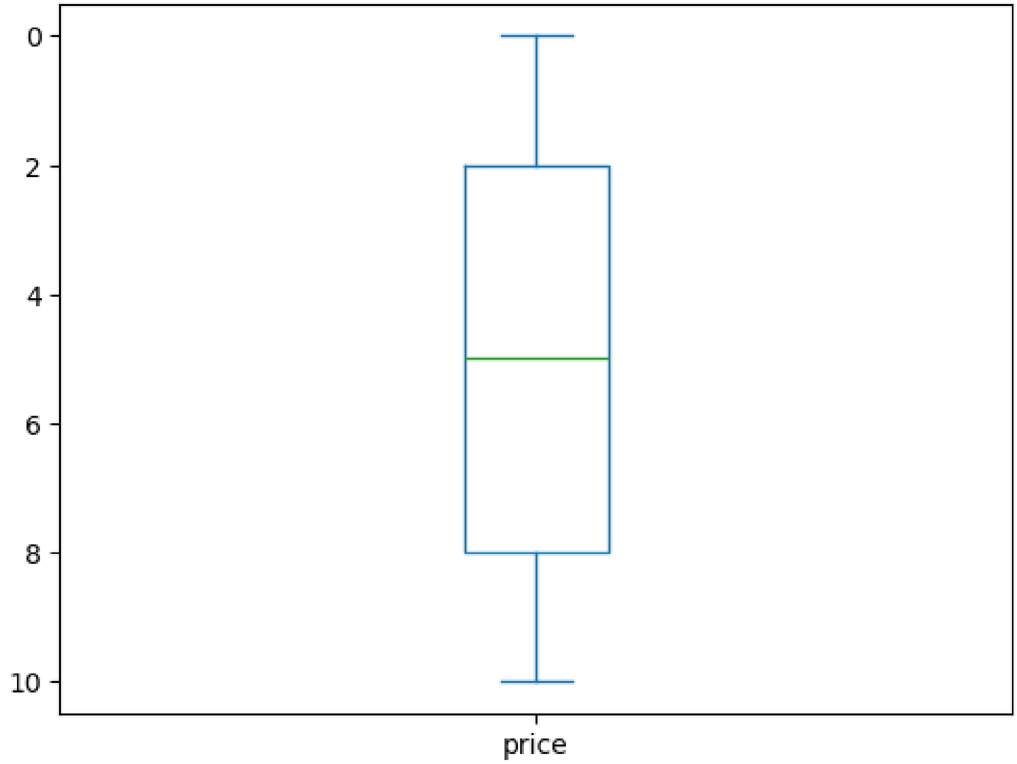
```
import matplotlib.pyplot as plt

df_uts2.price.plot(kind='box')
plt.gca().invert_yaxis()
plt.show()
```




[10]:
```
from pandas.api.types import is_numeric_dtype
def remove_outliner(df_in):
    for col_category in list(df_in.columns):
        q1 = df_in[col_category].quantile(0.25)
        q3 = df_in[col_category].quantile(0.75)

        iqr = q3-q1
        batas_atas = q3 + (1.5 * iqr)
        batas_bawah = q1 - (1.5 * iqr)

        df_out = df_in.loc[(df_in[col_category] >= batas_bawah) &␣
 ↪(df_in[col_category] <= batas_atas)]
    return df_out
df_uts_clean = remove_outliner(df_uts2)
print("Jumlah baris DataFrame sebelum dibuang outlier", df_uts2.shape[0])
print("Jumlah baris DataFrame sesudah dibuang outlier", df_uts_clean.shape[0])
df_uts_clean.price.plot(kind='box', vert = True)

plt.gca().invert_yaxis()
plt.show()
```

```
Jumlah baris DataFrame sebelum dibuang outlier 10000
```

```
Jumlah baris DataFrame sesudah dibuang outlier 10000
```




[11]:
```python
df_uts2.describe()
```

[11]:
```
         hasyard_no   hasyard_yes    haspool_no   haspool_yes  isnewbuilt_new  \
count  10000.000000  10000.000000  10000.000000  10000.000000    10000.000000
mean       0.491300      0.508700      0.503200      0.496800        0.499100
std        0.499949      0.499949      0.500015      0.500015        0.500024
min        0.000000      0.000000      0.000000      0.000000        0.000000
25%        0.000000      0.000000      0.000000      0.000000        0.000000
50%        0.000000      1.000000      1.000000      0.000000        0.000000
75%        1.000000      1.000000      1.000000      1.000000        1.000000
max        1.000000      1.000000      1.000000      1.000000        1.000000

       isnewbuilt_old  hasstormprotector_no  hasstormprotector_yes  \
count    10000.000000          10000.000000           10000.000000
mean         0.500900              0.500100               0.499900
std          0.500024              0.500025               0.500025
min          0.000000              0.000000               0.000000
25%          0.000000              0.000000               0.000000
50%          1.000000              1.000000               0.000000
75%          1.000000              1.000000               1.000000
```

```
max          1.000000             1.000000               1.000000

       hasstorageroom_no  hasstorageroom_yes  …     citycode  \
count       10000.000000        10000.000000  …  10000.000000
mean            0.497000            0.503000  …     50.276300
std             0.500016            0.500016  …     28.889171
min             0.000000            0.000000  …      1.000000
25%             0.000000            0.000000  …     25.000000
50%             0.000000            1.000000  …     50.000000
75%             1.000000            1.000000  …     76.000000
max             1.000000            1.000000  …    100.000000

       citypartrange        floors        garage  hasguestroom          made  \
count   10000.000000  10000.000000  10000.000000   10000.00000  10000.000000
mean    50225.486100      5.510100      5.521700    2005.48850   5033.103900
std     29006.675799      2.872024      2.856667       9.30809   2876.729545
min         3.000000      1.000000      1.000000    1990.00000      0.000000
25%     24693.750000      3.000000      3.000000    1997.00000   2559.750000
50%     50693.000000      5.000000      5.000000    2005.50000   5092.500000
75%     75683.250000      8.000000      8.000000    2014.00000   7511.250000
max     99953.000000     10.000000     10.000000    2021.00000  10000.000000

       numberofrooms  numprevowners        price  squaremeters
count    10000.00000    10000.00000  10000.00000  1.000000e+04
mean      5028.01060      553.12120      4.99460  4.993448e+06
std       2894.33221      262.05017      3.17641  2.877424e+06
min          1.00000      100.00000      0.00000  1.031350e+04
25%       2512.00000      327.75000      2.00000  2.516402e+06
50%       5045.00000      554.00000      5.00000  5.016180e+06
75%       7540.50000      777.25000      8.00000  7.469092e+06
max      10000.00000     1000.00000     10.00000  1.000677e+07

[8 rows x 22 columns]
```

[12]:
```
print("data null\n", df_uts_clean.isnull().sum())
print("data kosong \n", df_uts_clean.empty)
print("data nan \n", df_uts_clean.isna().sum())
```

```
data null
 hasyard_no                0
hasyard_yes               0
haspool_no                0
haspool_yes               0
isnewbuilt_new            0
isnewbuilt_old            0
hasstormprotector_no      0
hasstormprotector_yes     0
```

```
hasstorageroom_no          0
hasstorageroom_yes         0
attic                      0
basement                   0
citycode                   0
citypartrange              0
floors                     0
garage                     0
hasguestroom               0
made                       0
numberofrooms              0
numprevowners              0
price                      0
squaremeters               0
dtype: int64
data kosong
 False
data nan
 hasyard_no                 0
hasyard_yes                0
haspool_no                 0
haspool_yes                0
isnewbuilt_new             0
isnewbuilt_old             0
hasstormprotector_no       0
hasstormprotector_yes      0
hasstorageroom_no          0
hasstorageroom_yes         0
attic                      0
basement                   0
citycode                   0
citypartrange              0
floors                     0
garage                     0
hasguestroom               0
made                       0
numberofrooms              0
numprevowners              0
price                      0
squaremeters               0
dtype: int64
```

[13]:

```
from sklearn.model_selection import train_test_split

X_regress = df_uts_clean.drop('price',axis=1)
y_regress = df_uts_clean.price
```

```
X_train_price, X_test_price, y_train_price, y_test_price =␣
 ↪train_test_split(X_regress, y_regress, test_size=0.25, random_state=83)
```

[14]:
```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.metrics import mean_absolute_error, mean_squared_error


pipe_Ridge1 = Pipeline(steps=[
    ('scale', StandardScaler()),
    ('feature_selection', SelectKBest(score_func=f_regression)),
    ('reg', Ridge())
])


param_grid_Ridge1 = {
    'reg__alpha': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
    'feature_selection__k': np.arange(1, 20),
}


GSCV_RR1 = GridSearchCV(pipe_Ridge1, param_grid_Ridge1, cv=5,
                       scoring='neg_mean_squared_error', error_score='raise',␣
 ↪n_jobs=-1)


GSCV_RR1.fit(X_train_price, y_train_price)


print("Best model: {}".format(GSCV_RR1.best_estimator_))
print("Ridge best parameters: {}".format(GSCV_RR1.best_params_))
print("Koefisien/bobot: {}".format(GSCV_RR1.best_estimator_.named_steps['reg'].
 ↪coef_))
print("Intercept/bias: {}".format(GSCV_RR1.best_estimator_.named_steps['reg'].
 ↪intercept_))


Ridge_predict1 = GSCV_RR1.predict(X_test_price)
mse_Ridge1 = mean_squared_error(y_test_price, Ridge_predict1)
mae_Ridge1 = mean_absolute_error(y_test_price, Ridge_predict1)

print("Ridge Mean Squared Error (MSE): {}".format(mse_Ridge1))
print("Ridge Mean Absolute Error (MAE): {}".format(mae_Ridge1))
print("Ridge Root Mean Squared Error: {}".format(np.sqrt(mse_Ridge1)))
```

```
Best model: Pipeline(steps=[('scale', StandardScaler()),
                ('feature_selection',
                 SelectKBest(k=18,
                             score_func=<function f_regression at
0x000001E3D0CCA480>)),
                ('reg', Ridge(alpha=1000))])
Ridge best parameters: {'feature_selection__k': 18, 'reg__alpha': 1000}
Koefisien/bobot: [ 0.01928019 -0.01928019  0.03766515 -0.03766515 -0.00705871
0.00705871
 -0.02562885  0.02562885  0.00805359 -0.06241694 -0.04543358 -0.03987613
 -0.06613026 -0.02391961 -0.03012721 -0.01514143 -0.03928844  0.0079404 ]
Intercept/bias: 4.9665333333333335
Ridge Mean Squared Error (MSE): 10.220054489118938
Ridge Mean Absolute Error (MAE): 2.777530469141877
Ridge Root Mean Squared Error: 3.19688199486921
```

[15]:
```
df_results = pd.DataFrame(y_test_price, columns=['price'])
df_results = pd.DataFrame(y_test_price)
df_results['Ridge Prediction1'] = Ridge_predict1
df_results['Selisih_price_RR1'] = df_results['Ridge Prediction1'] -␣
 ↪df_results['price']

df_results.head()
```

[15]:

| | price | Ridge Prediction1 | Selisih_price_RR1 |
|---|---|---|---|
| 2353 | 4.0 | 4.769949 | 0.769949 |
| 2050 | 10.0 | 4.957095 | -5.042905 |
| 3276 | 3.0 | 4.802099 | 1.802099 |
| 4297 | 5.0 | 5.009274 | 0.009274 |
| 9322 | 0.0 | 5.078501 | 5.078501 |

[16]:
```
df_results.describe()
```

[16]:

| | price | Ridge Prediction1 | Selisih_price_RR1 |
|---|---|---|---|
| count | 2500.00000 | 2500.000000 | 2500.000000 |
| mean | 5.07880 | 4.966710 | -0.112090 |
| std | 3.19059 | 0.159284 | 3.195556 |
| min | 0.00000 | 4.488009 | -5.477801 |
| 25% | 2.00000 | 4.862105 | -2.994186 |
| 50% | 5.00000 | 4.967807 | -0.087029 |
| 75% | 8.00000 | 5.073668 | 2.863164 |
| max | 10.00000 | 5.486070 | 5.448695 |

[17]:
```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
```

```
from sklearn.feature_selection import SelectPercentile, f_regression
from sklearn.metrics import mean_absolute_error, mean_squared_error


pipe_Ridge2 = Pipeline(steps=[
    ('scale', MinMaxScaler()),
    ('feature_selection', SelectPercentile(score_func=f_regression)),
    ('reg', Ridge())
])


param_grid_Ridge2 = {
    'reg__alpha': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
    'feature_selection__percentile': np.arange(1, 100, 5),
}


GSCV_RR2 = GridSearchCV(pipe_Ridge2, param_grid_Ridge2, cv=5,
                       scoring='neg_mean_squared_error', error_score='raise',␣
 ↪n_jobs=-1)


GSCV_RR2.fit(X_train_price, y_train_price)


print("Best model: {}".format(GSCV_RR2.best_estimator_))
print("Ridge best parameters: {}".format(GSCV_RR2.best_params_))
print("Koefisien/bobot: {}".format(GSCV_RR2.best_estimator_.named_steps['reg'].
 ↪coef_))
print("Intercept/bias: {}".format(GSCV_RR2.best_estimator_.named_steps['reg'].
 ↪intercept_))


Ridge_predict2 = GSCV_RR2.predict(X_test_price)
mse_Ridge2 = mean_squared_error(y_test_price, Ridge_predict2)
mae_Ridge2 = mean_absolute_error(y_test_price, Ridge_predict2)

print("Ridge Mean Squared Error (MSE): {}".format(mse_Ridge2))
print("Ridge Mean Absolute Error (MAE): {}".format(mae_Ridge2))
print("Ridge Root Mean Squared Error: {}".format(np.sqrt(mse_Ridge2)))
```

```
Best model: Pipeline(steps=[('scale', MinMaxScaler()),
                ('feature_selection',
                 SelectPercentile(percentile=96,
                                  score_func=<function f_regression at
0x000001E3D0CCA480>)),
                ('reg', Ridge(alpha=1000))])
```

```
Ridge best parameters: {'feature_selection__percentile': 96, 'reg__alpha': 1000}
Koefisien/bobot: [ 0.0330829  -0.0330829  -0.01184287  0.01184287  0.06360392
-0.06360392
 -0.01216413  0.01216413 -0.04359034  0.04359034  0.01677399 -0.09675684
 -0.07017122 -0.06053596 -0.10268351 -0.03759298 -0.0459055  -0.02422058
 -0.06134852  0.01667829]
Intercept/bias: 5.2003872640447
Ridge Mean Squared Error (MSE): 10.204805785909578
Ridge Mean Absolute Error (MAE): 2.7736714562691303
Ridge Root Mean Squared Error: 3.194496170902319
```

[18]:

```python
df_results = pd.DataFrame(y_test_price, columns=['price'])
df_results = pd.DataFrame(y_test_price)
df_results['Ridge Prediction2'] = Ridge_predict2
df_results['Selisih_price_RR2'] = df_results['Ridge Prediction2'] -
 ↪df_results['price']

df_results.head()
```

[18]:

| | price | Ridge Prediction2 | Selisih_price_RR2 |
|---|---|---|---|
| 2353 | 4.0 | 4.824650 | 0.824650 |
| 2050 | 10.0 | 4.910772 | -5.089228 |
| 3276 | 3.0 | 4.841420 | 1.841420 |
| 4297 | 5.0 | 4.939335 | -0.060665 |
| 9322 | 0.0 | 5.072129 | 5.072129 |

[19]:

```python
df_results.describe()
```

[19]:

| | price | Ridge Prediction2 | Selisih_price_RR2 |
|---|---|---|---|
| count | 2500.00000 | 2500.000000 | 2500.000000 |
| mean | 5.07880 | 4.966649 | -0.112151 |
| std | 3.19059 | 0.102459 | 3.193166 |
| min | 0.00000 | 4.682003 | -5.302744 |
| 25% | 2.00000 | 4.893080 | -3.013918 |
| 50% | 5.00000 | 4.970294 | -0.059937 |
| 75% | 8.00000 | 5.036254 | 2.892151 |
| max | 10.00000 | 5.279053 | 5.259660 |

[20]:

```python
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.metrics import mean_absolute_error, mean_squared_error

pipe_SVR1 = Pipeline(steps=[
    ('scale',  StandardScaler()),
```

```python
    ('feature_selection', SelectKBest(score_func= f_regression)),
    ('reg',SVR(kernel='linear'))
    ])
param_grid_SVR1 = {
    'reg__C': [0.1, 1],
    'reg__epsilon': [0.1, 0.5],
    'feature_selection__k': [5, 10, 15]
}

GSCV_SVR1 = GridSearchCV(pipe_SVR1, param_grid_SVR1, cv=5,
                        scoring= 'neg_mean_squared_error', n_jobs=-1)
GSCV_SVR1.fit(X_train_price, y_train_price)

print("Best model: {}".format(GSCV_SVR1.best_estimator_))
print("SVR best parameters :{}".format(GSCV_SVR1.best_params_))
print("Koefisien/bobot:{}".format(GSCV_SVR1.best_estimator_.named_steps['reg'].
 ↪coef_))
print("Intercept/bias:{}".format(GSCV_SVR1.best_estimator_.named_steps['reg'].
 ↪intercept_))

SVR_predict1 = GSCV_SVR1.predict(X_test_price)

mse_SVR1 = mean_squared_error(y_test_price, SVR_predict1)
mae_SVR1 = mean_absolute_error(y_test_price, SVR_predict1)

print("SVR Mean Squared Error (MSE): {}".format(mse_SVR1))
print("SVR Mean Absolute Error (MAE): {}".format(mae_SVR1))
print("SVR Root Mean Squared Error: {}".format(np.sqrt(mse_SVR1)))
```

```
Best model: Pipeline(steps=[('scale', StandardScaler()),
                ('feature_selection',
                 SelectKBest(k=15,
                             score_func=<function f_regression at
0x000001E3D0CCA480>)),
                ('reg', SVR(C=0.1, kernel='linear'))])
SVR best parameters :{'feature_selection__k': 15, 'reg__C': 0.1, 'reg__epsilon':
0.1}
Koefisien/bobot:[[ 0.01281518 -0.01281518  0.03751563 -0.03751563 -0.01326535
0.01326535
   0.0196735  -0.0485701  -0.03610083 -0.03745557 -0.06183886 -0.00677204
  -0.00059959 -0.01369374 -0.03240035]]
Intercept/bias:[4.969023]
SVR Mean Squared Error (MSE): 10.21817615702048
SVR Mean Absolute Error (MAE): 2.7765882848319805
SVR Root Mean Squared Error: 3.196588205731304
```

[21]:
```
df_results['SVR Prediction1'] = SVR_predict1
df_results = pd.DataFrame(y_test_price)
df_results['SVR Prediction1'] = SVR_predict1

df_results['Selisih_price_SVR1'] = df_results['SVR Prediction1'] -
  ↪df_results['price']
df_results.head()
```

[21]:

| | price | SVR Prediction1 | Selisih_price_SVR1 |
|---|---|---|---|
| 2353 | 4.0 | 4.813654 | 0.813654 |
| 2050 | 10.0 | 5.004715 | -4.995285 |
| 3276 | 3.0 | 4.802901 | 1.802901 |
| 4297 | 5.0 | 4.986395 | -0.013605 |
| 9322 | 0.0 | 5.058320 | 5.058320 |

[22]:
```
df_results.describe()
```

[22]:

| | price | SVR Prediction1 | Selisih_price_SVR1 |
|---|---|---|---|
| count | 2500.00000 | 2500.000000 | 2500.000000 |
| mean | 5.07880 | 4.970395 | -0.108405 |
| std | 3.19059 | 0.133106 | 3.195389 |
| min | 0.00000 | 4.597910 | -5.402090 |
| 25% | 2.00000 | 4.879044 | -2.997998 |
| 50% | 5.00000 | 4.971886 | -0.075344 |
| 75% | 8.00000 | 5.060051 | 2.878130 |
| max | 10.00000 | 5.357458 | 5.332233 |

[23]:
```
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import SelectPercentile, f_regression
from sklearn.metrics import mean_absolute_error, mean_squared_error

pipe_SVR2 = Pipeline(steps=[
    ('scale',  MinMaxScaler()),
    ('feature_selection', SelectPercentile(score_func= f_regression)),
    ('reg',SVR(kernel='linear'))
    ])
param_grid_SVR2 = {
    'reg__C': [0.1, 1],
    'reg__epsilon': [0.1, 0.5],
    'feature_selection__percentile': [5, 10, 15]
}

GSCV_SVR2 = GridSearchCV(pipe_SVR2, param_grid_SVR2, cv=5,
                         scoring= 'neg_mean_squared_error', n_jobs=-1)
```

```python
GSCV_SVR2.fit(X_train_price, y_train_price)

print("Best model: {}".format(GSCV_SVR2.best_estimator_))
print("SVR best parameters :{}".format(GSCV_SVR2.best_params_))
print("Koefisien/bobot:{}".format(GSCV_SVR2.best_estimator_.named_steps['reg'].
 ↪coef_))
print("Intercept/bias:{}".format(GSCV_SVR2.best_estimator_.named_steps['reg'].
 ↪intercept_))

SVR_predict2 = GSCV_SVR2.predict(X_test_price)

mse_SVR2 = mean_squared_error(y_test_price, SVR_predict2)
mae_SVR2 = mean_absolute_error(y_test_price, SVR_predict2)

print("SVR Mean Squared Error (MSE): {}".format(mse_SVR2))
print("SVR Mean Absolute Error (MAE): {}".format(mae_SVR2))
print("SVR Root Mean Squared Error: {}".format(np.sqrt(mse_SVR2)))
```

```
Best model: Pipeline(steps=[('scale', MinMaxScaler()),
                ('feature_selection',
                 SelectPercentile(percentile=15,
                                  score_func=<function f_regression at
0x000001E3D0CCA480>)),
                ('reg', SVR(C=0.1, kernel='linear'))])
SVR best parameters :{'feature_selection__percentile': 15, 'reg__C': 0.1,
'reg__epsilon': 0.1}
Koefisien/bobot:[[ 1.00000000e-01 -1.00000000e-01  2.85903613e-07]]
Intercept/bias:[5.]
SVR Mean Squared Error (MSE): 10.190479952722335
SVR Mean Absolute Error (MAE): 2.773279993904061
SVR Root Mean Squared Error: 3.1922531153908102
```

[24]:

```python
df_results['SVR Prediction2'] = SVR_predict2
df_results = pd.DataFrame(y_test_price)
df_results['SVR Prediction2'] = SVR_predict2

df_results['Selisih_price_SVR2'] = df_results['SVR Prediction2'] -␣
 ↪df_results['price']
df_results.head()
```

[24]:

| | price | SVR Prediction2 | Selisih_price_SVR2 |
|---|---|---|---|
| 2353 | 4.0 | 4.9 | 0.9 |
| 2050 | 10.0 | 4.9 | -5.1 |
| 3276 | 3.0 | 4.9 | 1.9 |
| 4297 | 5.0 | 4.9 | -0.1 |
| 9322 | 0.0 | 5.1 | 5.1 |

[25]:
```
df_results.describe()
```

[25]:

| | price | SVR Prediction2 | Selisih_price_SVR2 |
|---|---|---|---|
| count | 2500.00000 | 2500.000 | 2500.000000 |
| mean | 5.07880 | 5.002 | -0.076800 |
| std | 3.19059 | 0.100 | 3.191968 |
| min | 0.00000 | 4.900 | -5.100000 |
| 25% | 2.00000 | 4.900 | -2.900000 |
| 50% | 5.00000 | 5.100 | -0.100000 |
| 75% | 8.00000 | 5.100 | 2.900000 |
| max | 10.00000 | 5.100 | 5.100000 |

[26]:
```
df_results = pd.DataFrame({'price' : y_test_price})

df_results['Ridge Prediction1'] = Ridge_predict1
df_results['Selisih_price_RR1'] = df_results['price'] - df_results['Ridge␣
↪Prediction1']

df_results['Ridge Prediction2'] = Ridge_predict2
df_results['Selisih_price_RR2'] = df_results['price'] - df_results['Ridge␣
↪Prediction2']

df_results['SVR Prediction1'] = SVR_predict1
df_results['Selisih_price_SVR1'] = df_results['price'] - df_results['SVR␣
↪Prediction1']

df_results['SVR Prediction2'] = SVR_predict2
df_results['Selisih_price_SVR2'] = df_results['price'] - df_results['SVR␣
↪Prediction2']

df_results.head()
```

[26]:

| | price | Ridge Prediction1 | Selisih_price_RR1 | Ridge Prediction2 \ |
|---|---|---|---|---|
| 2353 | 4.0 | 4.769949 | -0.769949 | 4.824650 |
| 2050 | 10.0 | 4.957095 | 5.042905 | 4.910772 |
| 3276 | 3.0 | 4.802099 | -1.802099 | 4.841420 |
| 4297 | 5.0 | 5.009274 | -0.009274 | 4.939335 |
| 9322 | 0.0 | 5.078501 | -5.078501 | 5.072129 |

| | Selisih_price_RR2 | SVR Prediction1 | Selisih_price_SVR1 | SVR Prediction2 \ |
|---|---|---|---|---|
| 2353 | -0.824650 | 4.813654 | -0.813654 | 4.9 |
| 2050 | 5.089228 | 5.004715 | 4.995285 | 4.9 |
| 3276 | -1.841420 | 4.802901 | -1.802901 | 4.9 |
| 4297 | 0.060665 | 4.986395 | 0.013605 | 4.9 |
| 9322 | -5.072129 | 5.058320 | -5.058320 | 5.1 |

Selisih_price_SVR2

```
2353                -0.9
2050                 5.1
3276                -1.9
4297                 0.1
9322                -5.1
```

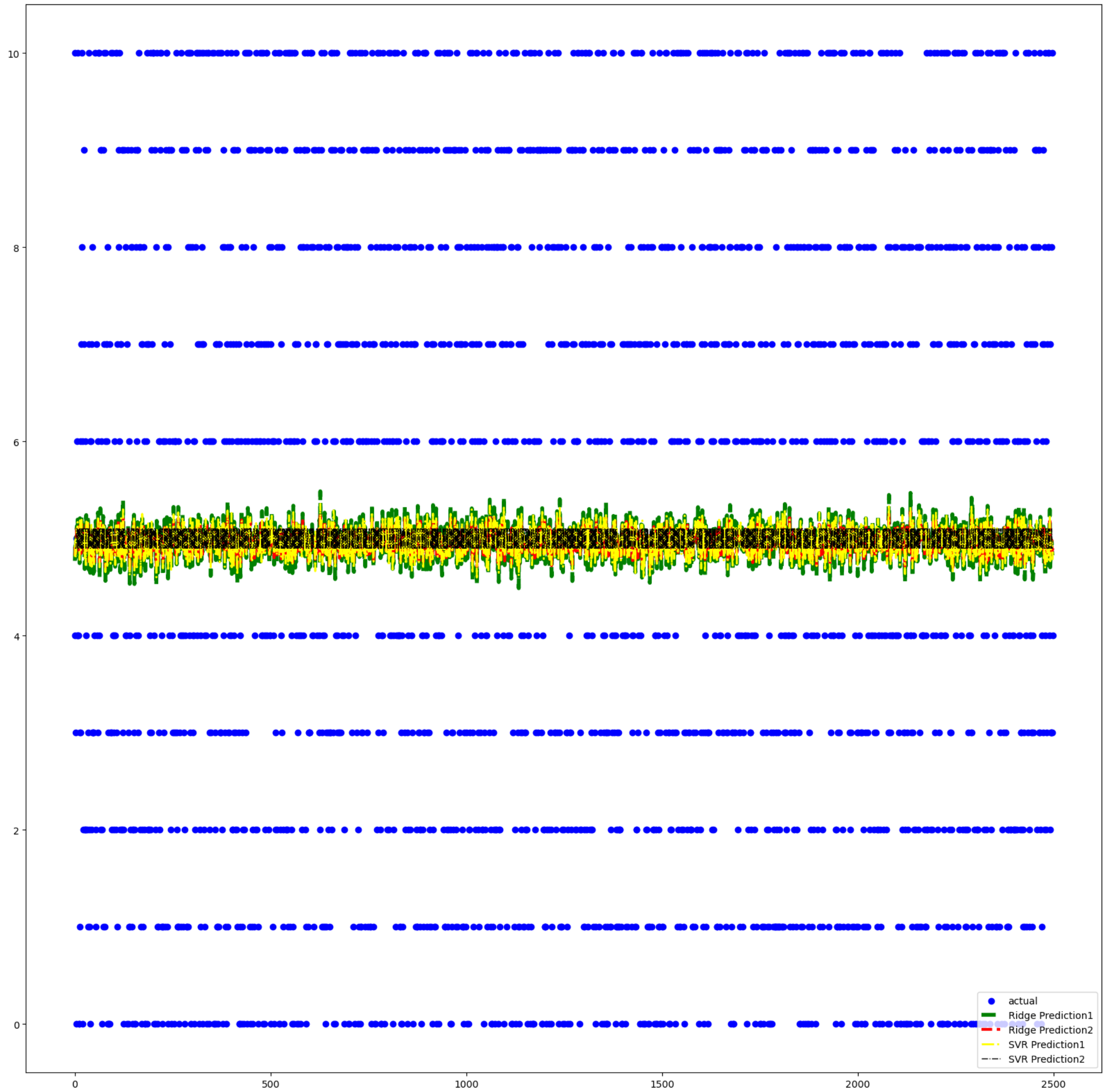
[27]:
```
df_results.describe()
```

[27]:
```
             price  Ridge Prediction1  Selisih_price_RR1  Ridge Prediction2  \
count  2500.00000        2500.000000        2500.000000        2500.000000
mean      5.07880           4.966710           0.112090           4.966649
std       3.19059           0.159284           3.195556           0.102459
min       0.00000           4.488009          -5.448695           4.682003
25%       2.00000           4.862105          -2.863164           4.893080
50%       5.00000           4.967807           0.087029           4.970294
75%       8.00000           5.073668           2.994186           5.036254
max      10.00000           5.486070           5.477801           5.279053

       Selisih_price_RR2  SVR Prediction1  Selisih_price_SVR1  \
count        2500.000000      2500.000000         2500.000000
mean            0.112151         4.970395            0.108405
std             3.193166         0.133106            3.195389
min            -5.259660         4.597910           -5.332233
25%            -2.892151         4.879044           -2.878130
50%             0.059937         4.971886            0.075344
75%             3.013918         5.060051            2.997998
max             5.302744         5.357458            5.402090

       SVR Prediction2  Selisih_price_SVR2
count         2500.000         2500.000000
mean             5.002            0.076800
std              0.100            3.191968
min              4.900           -5.100000
25%              4.900           -2.900000
50%              5.100            0.100000
75%              5.100            2.900000
max              5.100            5.100000
```

[28]:
```
plt.figure(figsize=(20, 20))

data_len = range(len(y_test_price))

plt.scatter(data_len, df_results.price, label= "actual", color="blue")
plt.plot(data_len, df_results['Ridge Prediction1'], label="Ridge Prediction1",␣
 ↪color="green", linewidth=4, linestyle="dashed")
plt.plot(data_len, df_results['Ridge Prediction2'], label="Ridge Prediction2",␣
 ↪color="red", linewidth=3, linestyle="dashed")
```

```
plt.plot(data_len, df_results['SVR Prediction1'], label= "SVR Prediction1",␣
    ↪color="yellow", linewidth= 2, linestyle= "-.")
plt.plot(data_len, df_results['SVR Prediction2'], label= "SVR Prediction2",␣
    ↪color="black", linewidth= 1, linestyle= "-.")
plt.legend()
plt.show
```

[28]: <function matplotlib.pyplot.show(close=None, block=None)>




[29]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np
```

```
mae_ridge1 = mean_absolute_error(df_results['price'], df_results['Ridge␣
 ↪Prediction1'])
rmse_ridge1 = np.sqrt(mean_absolute_error(df_results['price'],df_results['Ridge␣
 ↪Prediction1']))
ridge_feature_count1 = GSCV_RR1.best_params_['feature_selection__k']

mae_svr1 = mean_absolute_error(df_results['price'], df_results['SVR␣
 ↪Prediction1'])
rmse_svr1 = np.sqrt(mean_squared_error(df_results['price'], df_results['SVR␣
 ↪Prediction1']))
svr_feature_count1 = GSCV_SVR1.best_params_['feature_selection__k']


mae_svr2 = mean_absolute_error(df_results['price'], df_results['SVR␣
 ↪Prediction2'])
rmse_svr2 = np.sqrt(mean_squared_error(df_results['price'], df_results['SVR␣
 ↪Prediction2']))
svr_feature_count2 = GSCV_SVR2.best_params_['feature_selection__percentile']

mae_ridge2 = mean_absolute_error(df_results['price'], df_results['Ridge␣
 ↪Prediction2'])
rmse_ridge2 = np.sqrt(mean_absolute_error(df_results['price'],df_results['Ridge␣
 ↪Prediction2']))
ridge_feature_count2 = GSCV_RR2.best_params_['feature_selection__percentile']

print(f"Ridge MAE 1: {mae_ridge1}, Ridge RMSE 1: {rmse_ridge1}, Ridge Feature␣
 ↪Count 1: {ridge_feature_count1}")
print(f"Ridge MAE 2: {mae_ridge2}, Ridge RMSE 2: {rmse_ridge2}, Ridge Feature␣
 ↪Count 2: {ridge_feature_count2}")
print(f"SVR MAE 1: {mae_svr1}, SVR RMSE 1: {rmse_svr1}, SVR Feature Count 1:␣
 ↪{svr_feature_count1}")
print(f"SVR MAE 2: {mae_svr2}, SVR RMSE 2: {rmse_svr2}, SVR Feature Count 2:␣
 ↪{svr_feature_count2}")
```

```
Ridge MAE 1: 2.777530469141877, Ridge RMSE 1: 1.6665924724244607, Ridge Feature
Count 1: 18
Ridge MAE 2: 2.7736714562691303, Ridge RMSE 2: 1.6654343146065924, Ridge Feature
Count 2: 96
SVR MAE 1: 2.7765882848319805, SVR RMSE 1: 3.196588205731304, SVR Feature Count
1: 15
SVR MAE 2: 2.773279993904061, SVR RMSE 2: 3.1922531153908102, SVR Feature Count
2: 15
```

[30]:
```
import pickle

best_model = GSCV_RR1.best_estimator_
```

```python
# simpan model ke file .pkl
with open('BestModel_REG_Ridge_Tensorflow.pkl', 'wb') as f:
    pickle.dump(best_model, f)

print("Model terbaik berhasil disimpan ke 'BestModel_REG_Ridge_Tensorflow.pkl'")
```

```
Model terbaik berhasil disimpan ke 'Ridge_Price_model.pkl'
```