est-vs-logisticregression-jonathan

October 25, 2024

[19]:
```
import pandas as pd
import numpy as np

DatasetUTS_Gasal2425_csv = pd.read_csv(r'D:
 ↪\PMDPM\Klasifikasi1\DatasetUTS_Gasal2425.csv')

df_DatasetUTS_Gasal2425 = pd.DataFrame(data = DatasetUTS_Gasal2425_csv, index =␣
 ↪None)
df_DatasetUTS_Gasal2425
```

[19]:

	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	
…	…	…	…	…	…	…	
9995	341	83	no	no	8	1960	
9996	21514	5	no	yes	11	91373	
9997	1726	89	no	yes	5	73133	
9998	44403	29	yes	yes	12	34606	
9999	1440	84	no	no	49	18412	

	citypartrange	numprevowners	made	isnewbuilt	hasstormprotector	\
0	3	8	2005	old	yes	
1	6	8	2021	old	no	
2	3	4	2003	new	no	
3	8	3	2012	new	yes	
4	4	2	2011	new	yes	
…	…	…	…	…	…	
9995	4	4	1993	new	yes	
9996	1	1	1999	old	no	
9997	7	6	2009	old	yes	
9998	9	4	1990	old	yes	
9999	6	10	1994	new	no	

	basement	attic	garage	hasstorageroom	hasguestroom	price	category

```
0         4313   9005     956            no             7  7559081.5   Luxury
1         2937   8852     135           yes             9  5574642.1   Middle
2         6326   4748     654            no            10  8696869.3   Luxury
3          632   5792     807           yes             5  5154055.2   Middle
4         5414   1172     716           yes             9  9652258.1   Luxury
…          …      …       …             …              …       …
9995      2366   4016     229           yes             5    35371.3    Basic
9996      2584   5266     787            no             3  2153602.9    Basic
9997      9311   1698     218            no             4   176425.9    Basic
9998      9061   1742     230            no             0  4448474.0    Basic
9999      8485   2024     278           yes             6   146708.4    Basic

[10000 rows x 18 columns]
```

[20]:
```
print("data null\n", df_DatasetUTS_Gasal2425.isnull().sum())
print("\n")
print("data kosong \n", df_DatasetUTS_Gasal2425.empty)
print("\n")
print("data nan \n", df_DatasetUTS_Gasal2425.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
price                 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
price                 0
category              0
dtype: int64
```

[21]:
```
df_DatasetUTS_Gasal2425.describe()
```

[21]:

	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
```

[22]:
```
import pandas as pd

df = pd.read_csv('D:\\PMDPM\\Klasifikasi1\\DatasetUTS_Gasal2425.csv')

df
```

[22]:

	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	
…	…	…	…	…	…	…	
9995	341	83	no	no	8	1960	
9996	21514	5	no	yes	11	91373	
9997	1726	89	no	yes	5	73133	
9998	44403	29	yes	yes	12	34606	
9999	1440	84	no	no	49	18412	

	citypartrange	numprevowners	made	isnewbuilt	hasstormprotector	\
0	3	8	2005	old	yes	
1	6	8	2021	old	no	
2	3	4	2003	new	no	
3	8	3	2012	new	yes	
4	4	2	2011	new	yes	
…	…	…	…	…	…	
9995	4	4	1993	new	yes	
9996	1	1	1999	old	no	
9997	7	6	2009	old	yes	
9998	9	4	1990	old	yes	
9999	6	10	1994	new	no	

	basement	attic	garage	hasstorageroom	hasguestroom	price	category
0	4313	9005	956	no	7	7559081.5	Luxury
1	2937	8852	135	yes	9	5574642.1	Middle
2	6326	4748	654	no	10	8696869.3	Luxury
3	632	5792	807	yes	5	5154055.2	Middle
4	5414	1172	716	yes	9	9652258.1	Luxury
…	…	…	…	…	…	…	…
9995	2366	4016	229	yes	5	35371.3	Basic
9996	2584	5266	787	no	3	2153602.9	Basic
9997	9311	1698	218	no	4	176425.9	Basic

```
9998       9061    1742      230              no              0  4448474.0     Basic
9999       8485    2024      278             yes              6   146708.4     Basic

[10000 rows x 18 columns]
```

[23]:
```
print(df_DatasetUTS_Gasal2425.head())
```

```
   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 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
```

[24]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

file_path = r'D:\PMDPM\Klasifikasi1\DatasetUTS_Gasal2425.csv'
df_DatasetUTS_Gasal2425 = pd.read_csv(file_path)

categorical_columns = ['hasyard', 'haspool', 'isnewbuilt', 'hasstormprotector',␣
 ↪'hasstorageroom', 'category']

le = LabelEncoder()

for col in categorical_columns:
    if col in df_DatasetUTS_Gasal2425.columns:
        df_DatasetUTS_Gasal2425[col] = le.
 ↪fit_transform(df_DatasetUTS_Gasal2425[col])
    else:
        print(f"Kolom {col} tidak ditemukan dalam DataFrame.")

print(df_DatasetUTS_Gasal2425)
```

	squaremeters	numberofrooms	hasyard	haspool	floors	citycode	\
0	75523	3	0	1	63	9373	
1	55712	58	0	1	19	34457	
2	86929	100	1	0	11	98155	
3	51522	3	0	0	61	9047	
4	96470	74	1	0	21	92029	
…	…	…	…	…	…	…	
9995	341	83	0	0	8	1960	
9996	21514	5	0	1	11	91373	
9997	1726	89	0	1	5	73133	
9998	44403	29	1	1	12	34606	
9999	1440	84	0	0	49	18412	

	citypartrange	numprevowners	made	isnewbuilt	hasstormprotector	\
0	3	8	2005	1	1	
1	6	8	2021	1	0	
2	3	4	2003	0	0	
3	8	3	2012	0	1	
4	4	2	2011	0	1	
…	…	…	…	…	…	
9995	4	4	1993	0	1	
9996	1	1	1999	1	0	
9997	7	6	2009	1	1	
9998	9	4	1990	1	1	
9999	6	10	1994	0	0	

	basement	attic	garage	hasstorageroom	hasguestroom	price	\
0	4313	9005	956	0	7	7559081.5	
1	2937	8852	135	1	9	5574642.1	
2	6326	4748	654	0	10	8696869.3	
3	632	5792	807	1	5	5154055.2	
4	5414	1172	716	1	9	9652258.1	
…	…	…	…	…	…	…	
9995	2366	4016	229	1	5	35371.3	
9996	2584	5266	787	0	3	2153602.9	
9997	9311	1698	218	0	4	176425.9	
9998	9061	1742	230	0	0	4448474.0	
9999	8485	2024	278	1	6	146708.4	

	category
0	1
1	2
2	1
3	2
4	1
…	…
9995	0
9996	0

```
9997         0
9998         0
9999         0

[10000 rows x 18 columns]
```

[25]:
```
print(df_DatasetUTS_Gasal2425.head())
```

```
   squaremeters  numberofrooms  hasyard  haspool  floors  citycode  \
0         75523              3        0        1      63      9373
1         55712             58        0        1      19     34457
2         86929            100        1        0      11     98155
3         51522              3        0        0      61      9047
4         96470             74        1        0      21     92029

   citypartrange  numprevowners  made  isnewbuilt  hasstormprotector  \
0              3              8  2005           1                  1
1              6              8  2021           1                  0
2              3              4  2003           0                  0
3              8              3  2012           0                  1
4              4              2  2011           0                  1

   basement  attic  garage  hasstorageroom  hasguestroom      price  category
0      4313   9005     956               0             7  7559081.5         1
1      2937   8852     135               1             9  5574642.1         2
2      6326   4748     654               0            10  8696869.3         1
3       632   5792     807               1             5  5154055.2         2
4      5414   1172     716               1             9  9652258.1         1
```

[26]:
```
from sklearn.model_selection import train_test_split

X = df_DatasetUTS_Gasal2425.drop(columns=['price'], axis=1)
y = df_DatasetUTS_Gasal2425['category']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=94)

print("bentuk X_train", X_train.shape)
print("bentuk X_test", X_test.shape)
print("bentuk y_train", y_train.shape)
print("bentuk y_test", y_test.shape)
print("y_train \n", y_train)
print("y_test \n", y_test)
```

```
bentuk X_train (7500, 17)
bentuk X_test (2500, 17)
bentuk y_train (7500,)
bentuk y_test (2500,)
```

```
y_train
 8494    0
5661    0
4117    2
929     1
4628    2
       ..
6806    0
1347    2
1381    2
6336    0
6482    0
Name: category, Length: 7500, dtype: int32
y_test
 7497    0
5188    1
5009    1
9434    0
2444    1
       ..
2434    1
3521    1
1955    1
2954    1
703     1
Name: category, Length: 2500, dtype: int32
```

[10]:
```
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

SVM = SVC(C = 1, gamma=0.01, random_state=94)
RF = RandomForestClassifier(max_depth=5, n_estimators=100, random_state=94)

SVM.fit(X_train, y_train)
RF.fit(X_train, y_train)
```

[10]:
```
RandomForestClassifier(max_depth=5, random_state=94)
```

[11]:
```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

data = load_iris()
X, y = data.data, data.target
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=94)

SVM = SVC()
SVM.fit(X_train, y_train)

RF = RandomForestClassifier()
RF.fit(X_train, y_train)

X_new = np.array([[3, 197, 30, 19, 0, 44.8, 0.567, 55]])

# Hapus pembuatan DataFrame dengan nama fitur
X_new_df = pd.DataFrame(X_new)

print("X_new yang akan diprediksi", X_new_df.shape)

n_features_svm = SVM.n_features_in_
n_features_rf = RF.n_features_in_

print(f"Number of features in SVM: {n_features_svm}")
print(f"Number of features in RF: {n_features_rf}")

if X_new_df.shape[1] < n_features_svm:
    missing_features_svm = n_features_svm - X_new_df.shape[1]
    X_new_svm = pd.concat([X_new_df, pd.DataFrame(np.zeros((X_new_df.shape[0],␣
 ↪missing_features_svm)), columns=[f'feature{len(X_new_df.columns) + i + 1}'␣
 ↪for i in range(missing_features_svm)])], axis=1)
    print(f"Menyesuaikan input untuk SVM dengan menambahkan␣
 ↪{missing_features_svm} fitur.")
else:
    X_new_svm = X_new_df.iloc[:, :n_features_svm]

if X_new_df.shape[1] < n_features_rf:
    missing_features_rf = n_features_rf - X_new_df.shape[1]
    X_new_rf = pd.concat([X_new_df, pd.DataFrame(np.zeros((X_new_df.shape[0],␣
 ↪missing_features_rf)), columns=[f'feature{len(X_new_df.columns) + i + 1}'␣
 ↪for i in range(missing_features_rf)])], axis=1)
    print(f"Menyesuaikan input untuk Random Forest dengan menambahkan␣
 ↪{missing_features_rf} fitur.")
else:
    X_new_rf = X_new_df.iloc[:, :n_features_rf]

svm_predict = SVM.predict(X_new_svm)
print("Label Prediksi SVM", svm_predict)

rf_predict = RF.predict(X_new_rf)
print("Label Prediksi RF", rf_predict)
```

```
X_new yang akan diprediksi (1, 8)
Number of features in SVM: 4
Number of features in RF: 4
Label Prediksi SVM [2]
Label Prediksi RF [2]
```

[29]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, SelectPercentile, f_classif
from sklearn.metrics import classification_report, roc_auc_score

# Load dataset
data = load_iris()
X, y = data.data, data.target

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=94)

# Standardize data
scaler_standard = StandardScaler()
X_train_standard = scaler_standard.fit_transform(X_train)
X_test_standard = scaler_standard.transform(X_test)

# Feature selection using SelectKBest
kbest = SelectKBest(score_func=f_classif, k=min(10, X_train_standard.shape[1]))␣
 ↪ # Ensure k does not exceed number of features
X_train_kbest = kbest.fit_transform(X_train_standard, y_train)
X_test_kbest = kbest.transform(X_test_standard)

# Feature selection using SelectPercentile
percentile = SelectPercentile(score_func=f_classif, percentile=50)
X_train_percentile = percentile.fit_transform(X_train_standard, y_train)
X_test_percentile = percentile.transform(X_test_standard)

# Grid search for Random Forest
param_grid_rf = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
```

```
}
rf_grid = GridSearchCV(RandomForestClassifier(random_state=94), param_grid_rf,␣
 ↪cv=5)
rf_grid.fit(X_train_kbest, y_train)

# Grid search for Logistic Regression with corrected parameter grid
param_grid_lr = {
    'C': [0.01, 0.1, 1, 10],
    'penalty': ['l2'],  # Only use l2 penalty for lbfgs
    'solver': ['liblinear', 'saga', 'lbfgs'],  # 'saga' can handle both␣
 ↪penalties
    'max_iter': [1000, 2000]  # Increase max_iter
}
lr_grid = GridSearchCV(LogisticRegression(random_state=94), param_grid_lr, cv=5)
lr_grid.fit(X_train_kbest, y_train)

# Predictions
rf_pred = rf_grid.predict(X_test_kbest)
lr_pred = lr_grid.predict(X_test_kbest)

# Display classification reports
print("Random Forest Report:")
print(classification_report(y_test, rf_pred))
print("Logistic Regression Report:")
print(classification_report(y_test, lr_pred))

# Calculate AUC
rf_auc = roc_auc_score(y_test, rf_grid.predict_proba(X_test_kbest),␣
 ↪multi_class='ovr')
lr_auc = roc_auc_score(y_test, lr_grid.predict_proba(X_test_kbest),␣
 ↪multi_class='ovr')

print("Random Forest AUC:", rf_auc)
print("Logistic Regression AUC:", lr_auc)
```

```
Random Forest Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      0.92      0.96        12
           2       0.89      1.00      0.94         8

    accuracy                           0.97        30
   macro avg       0.96      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30

Logistic Regression Report:
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      0.92      0.96        12
           2       0.89      1.00      0.94         8

    accuracy                           0.97        30
   macro avg       0.96      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30

Random Forest AUC: 0.9965628507295173
Logistic Regression AUC: 0.9965628507295173
```

[30]:
```python
import numpy as np
import pandas as pd

# Define your input array (replace these values with your actual features)
X_new = np.array([[3, 197, 30, 19, 0, 44.8, 0.567, 55, 0, 0, 0, 0, 0, 0, 0, 0,␣
 ↪0]])  # Example input with fewer than 17 features

# Ensure X_new has 17 features
expected_features = 17
current_shape = X_new.shape[1]

if current_shape < expected_features:
    # Add zeros to match the expected number of features
    X_new = np.hstack([X_new, np.zeros((X_new.shape[0], expected_features -␣
 ↪current_shape))])
    print(f"Menyesuaikan input dengan menambahkan {expected_features -␣
 ↪current_shape} fitur.")
elif current_shape > expected_features:
    # Truncate to match the expected number of features
    X_new = X_new[:, :expected_features]
    print(f"Truncating input to keep only the first {expected_features}␣
 ↪features.")

print("X_new yang akan diprediksi", X_new.shape)

# Get the expected number of features from the models
n_features_svm = SVM.n_features_in_
n_features_rf = RF.n_features_in_

print(f"Expected number of features for SVM: {n_features_svm}")
print(f"Expected number of features for RF: {n_features_rf}")

# Adjust input for SVM
```

```
X_new_svm = X_new[:, :n_features_svm]  # Assuming this is correct based on␣
 ↪training features

# Adjust input for Random Forest
X_new_rf = X_new[:, :n_features_rf]  # Assuming this is correct based on␣
 ↪training features

# Make predictions
svm_predict = SVM.predict(X_new_svm)
print("Label Prediksi SVM:", svm_predict)

rf_predict = RF.predict(X_new_rf)
print("Label Prediksi RF:", rf_predict)
```

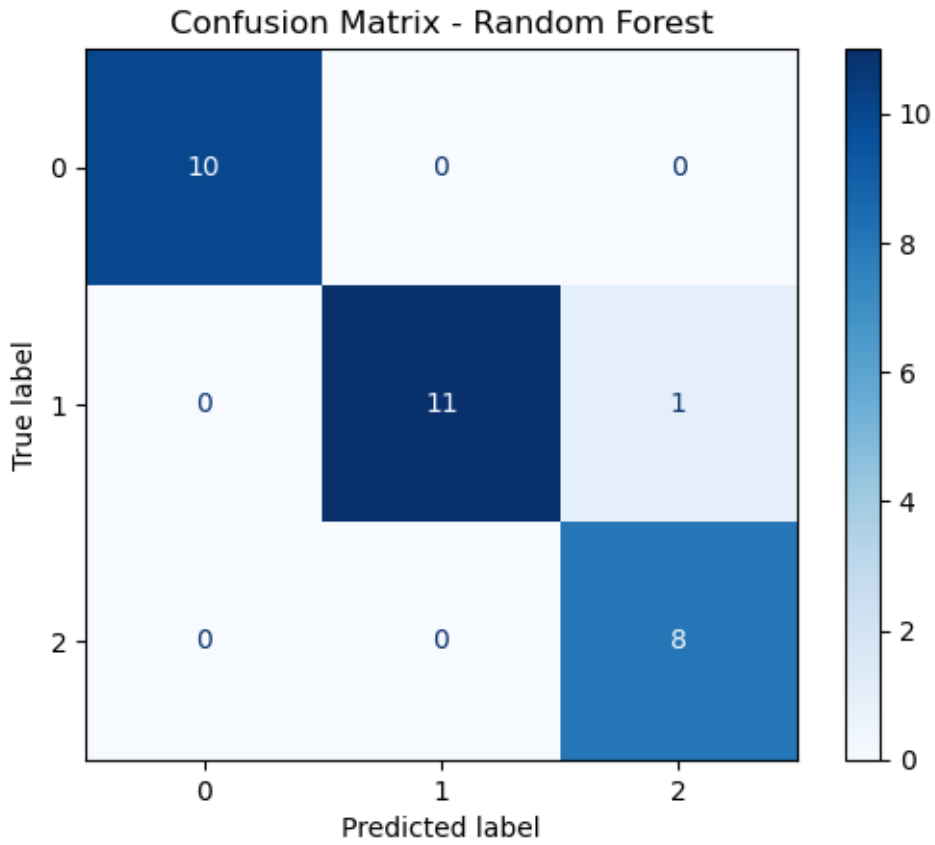
```
X_new yang akan diprediksi (1, 17)
Expected number of features for SVM: 4
Expected number of features for RF: 4
Label Prediksi SVM: [2]
Label Prediksi RF: [2]
```

[31]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, GridSearchCV,␣
 ↪StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪ConfusionMatrixDisplay

# Define your feature names here
feature_names = ['feature1', 'feature2', 'feature3', 'feature4']  # Replace␣
 ↪with your actual feature names

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=94)

# Define k based on the number of features available
k = min(10, X_train.shape[1])  # Ensure k does not exceed the number of features

pipeline_rf = Pipeline([
    ('scaler', StandardScaler()),
    ('feature_selection', SelectKBest(score_func=f_classif, k=k)),
    ('classifier', RandomForestClassifier(random_state=94))
```

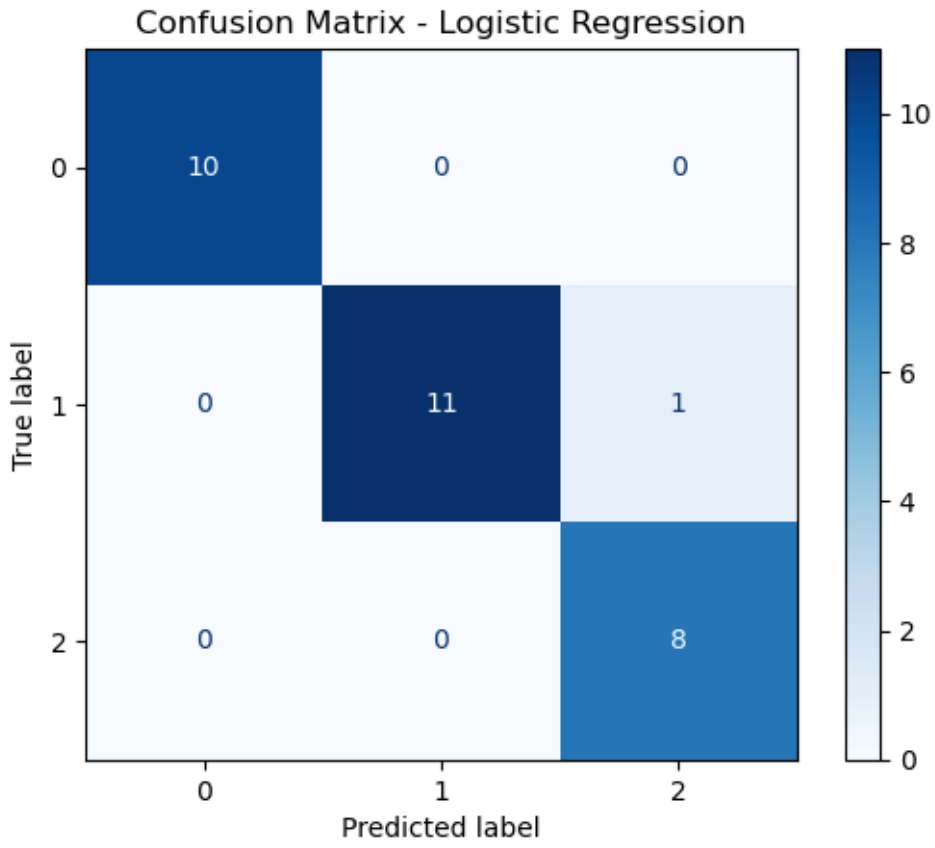
```
])

pipeline_lr = Pipeline([
    ('scaler', StandardScaler()),
    ('feature_selection', SelectKBest(score_func=f_classif, k=k)),
    ('classifier', LogisticRegression(max_iter=2000, solver='saga',␣
 ↪random_state=94))  # Increased max_iter and changed solver
])

param_grid_rf = {
    'classifier__n_estimators': [100, 200],
    'classifier__max_depth': [None, 10, 20],
    'classifier__min_samples_split': [2, 5],
    'classifier__min_samples_leaf': [1, 2]
}

param_grid_lr = {
    'classifier__C': [0.01, 0.1, 1, 10],
    'classifier__penalty': ['l1', 'l2'],
    'classifier__solver': ['liblinear', 'saga']
}

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=94)

grid_search_rf = GridSearchCV(pipeline_rf, param_grid_rf, cv=skf,␣
 ↪scoring='accuracy')
grid_search_rf.fit(X_train, y_train)

grid_search_lr = GridSearchCV(pipeline_lr, param_grid_lr, cv=skf,␣
 ↪scoring='accuracy')
grid_search_lr.fit(X_train, y_train)

rf_pred = grid_search_rf.predict(X_test)
print("Random Forest Classification Report:")
print(classification_report(y_test, rf_pred))

conf_matrix_rf = confusion_matrix(y_test, rf_pred)
ConfusionMatrixDisplay(confusion_matrix=conf_matrix_rf).plot(cmap='Blues')
plt.title('Confusion Matrix - Random Forest')
plt.show()

lr_pred = grid_search_lr.predict(X_test)
print("Logistic Regression Classification Report:")
print(classification_report(y_test, lr_pred))

conf_matrix_lr = confusion_matrix(y_test, lr_pred)
ConfusionMatrixDisplay(confusion_matrix=conf_matrix_lr).plot(cmap='Blues')
```

```
plt.title('Confusion Matrix - Logistic Regression')
plt.show()

# Get relevant features for Random Forest
feature_mask_rf = grid_search_rf.best_estimator_.
 ↪named_steps['feature_selection'].get_support()
relevant_features_rf = np.array(feature_names)[feature_mask_rf]
print("Fitur Relevan yang Dipilih oleh Random Forest:", relevant_features_rf)

# Get relevant features for Logistic Regression
feature_mask_lr = grid_search_lr.best_estimator_.
 ↪named_steps['feature_selection'].get_support()
relevant_features_lr = np.array(feature_names)[feature_mask_lr]
print("Fitur Relevan yang Dipilih oleh Logistic Regression:",␣
 ↪relevant_features_lr)
```

```
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      0.92      0.96        12
           2       0.89      1.00      0.94         8

    accuracy                           0.97        30
   macro avg       0.96      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30
```




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

           0       1.00      1.00      1.00        10
           1       1.00      0.92      0.96        12
           2       0.89      1.00      0.94         8

    accuracy                           0.97        30
   macro avg       0.96      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30
```




```
Fitur Relevan yang Dipilih oleh Random Forest: ['feature1' 'feature2' 'feature3'
'feature4']
Fitur Relevan yang Dipilih oleh Logistic Regression: ['feature1' 'feature2'
'feature3' 'feature4']
```

[15]:
```
rf_best_accuracy = grid_search_rf.best_score_
rf_best_f1 = classification_report(y_test, rf_pred, output_dict=True)['weighted␣
 ↪avg']['f1-score']
rf_best_precision = classification_report(y_test, rf_pred,␣
 ↪output_dict=True)['weighted avg']['precision']
rf_best_recall = classification_report(y_test, rf_pred,␣
 ↪output_dict=True)['weighted avg']['recall']
rf_best_n_features = np.sum(feature_mask_rf)

print("Random Forest Best Metrics:")
print(f"Accuracy: {rf_best_accuracy:.4f}")
print(f"Precision: {rf_best_precision:.4f}")
print(f"Recall: {rf_best_recall:.4f}")
print(f"F1 Score: {rf_best_f1:.4f}")
print(f"Number of Features: {rf_best_n_features}")
```

```
lr_best_accuracy = grid_search_lr.best_score_
lr_best_f1 = classification_report(y_test, lr_pred, output_dict=True)['weighted␣
 ↪avg']['f1-score']
lr_best_precision = classification_report(y_test, lr_pred,␣
 ↪output_dict=True)['weighted avg']['precision']
lr_best_recall = classification_report(y_test, lr_pred,␣
 ↪output_dict=True)['weighted avg']['recall']
lr_best_n_features = np.sum(feature_mask_lr)

print("\nLogistic Regression Best Metrics:")
print(f"Accuracy: {lr_best_accuracy:.4f}")
print(f"Precision: {lr_best_precision:.4f}")
print(f"Recall: {lr_best_recall:.4f}")
print(f"F1 Score: {lr_best_f1:.4f}")
print(f"Number of Features: {lr_best_n_features}")

if rf_best_f1 > lr_best_f1:
    best_model = "Random Forest"
    best_metrics = {
        "Accuracy": rf_best_accuracy,
        "Precision": rf_best_precision,
        "Recall": rf_best_recall,
        "F1 Score": rf_best_f1,
        "Number of Features": rf_best_n_features
    }
else:
    best_model = "Logistic Regression"
    best_metrics = {
        "Accuracy": lr_best_accuracy,
        "Precision": lr_best_precision,
        "Recall": lr_best_recall,
        "F1 Score": lr_best_f1,
        "Number of Features": lr_best_n_features
    }

print(f"\nModel Terbaik: {best_model}")
print("Metrik Terbaik:")
for metric, value in best_metrics.items():
    print(f"{metric}: {value:.4f}")
```

```
Random Forest Best Metrics:
Accuracy: 0.9667
Precision: 0.9704
Recall: 0.9667
F1 Score: 0.9669
Number of Features: 4
```

```
Logistic Regression Best Metrics:
Accuracy: 0.9750
Precision: 0.9704
Recall: 0.9667
F1 Score: 0.9669
Number of Features: 4

Model Terbaik: Logistic Regression
Metrik Terbaik:
Accuracy: 0.9750
Precision: 0.9704
Recall: 0.9667
F1 Score: 0.9669
Number of Features: 4.0000
```

[32]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
import joblib

# Load dataset
df_DatasetUTS_Gasal2425 = pd.read_csv('DatasetUTS_Gasal2425.csv')

# Inisialisasi LabelEncoder
label_encoder = LabelEncoder()

# Mengubah kolom 'category' menjadi numerik
df_DatasetUTS_Gasal2425['category'] = label_encoder.
 ↪fit_transform(df_DatasetUTS_Gasal2425['category'])

# Simpan kelas dari kolom 'category' untuk digunakan di laporan klasifikasi
category_classes = label_encoder.classes_

# Pastikan semua kolom kategorikal lain diubah menjadi numerik
for col in df_DatasetUTS_Gasal2425.columns:
    if df_DatasetUTS_Gasal2425[col].dtype == 'object':
        df_DatasetUTS_Gasal2425[col] = label_encoder.
 ↪fit_transform(df_DatasetUTS_Gasal2425[col])

# Memastikan tidak ada nilai NaN dalam dataset
df_DatasetUTS_Gasal2425 = df_DatasetUTS_Gasal2425.fillna(0)

# Memisahkan fitur dan target
X = df_DatasetUTS_Gasal2425.drop(['category', 'price'], axis=1)
```

```
y = df_DatasetUTS_Gasal2425['category']

# Membagi dataset menjadi training dan testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=94)

# Melatih model RandomForest
clf = RandomForestClassifier(n_estimators=100, random_state=94)
clf.fit(X_train, y_train)

# Melakukan prediksi pada data testing
y_pred = clf.predict(X_test)

# Menampilkan laporan klasifikasi dengan menggunakan category_classes
print(classification_report(y_test, y_pred, target_names=category_classes))

# Menyimpan model
joblib.dump(clf, 'BestModel_CLF_LR_Transformers.pkl')
```

```
              precision    recall  f1-score   support

       Basic       1.00      1.00      1.00       867
      Luxury       1.00      1.00      1.00       602
      Middle       1.00      1.00      1.00       531

    accuracy                           1.00      2000
   macro avg       1.00      1.00      1.00      2000
weighted avg       1.00      1.00      1.00      2000
```

[32]: ['BestModel_CLF_LR_Transformers.pkl']

Ridge vs SVR - Axel

October 25, 2024

[1]:
```
import pandas as pd
import numpy as np

df_price = pd.read_csv(r'C:\Users\ACER\Downloads\UTS MESIN\DatasetUTS_Gasal2425.
 ↪csv')
df_price.head(10)
```

[1]:

	squaremeters	numberofrooms	hasyard	haspool	floors	citycode	citypartrange	numprevowners	made	isnewbuilt	hasstormprotector	basement	attic	garage	hasstorageroom	hasguestroom	price	category
0	75523	3	no	yes	63	9373	3	8	2005	old	yes	4313	9005	956	no	7	7559081.5	Luxury
1	55712	58	no	yes	19	34457	6	8	2021	old	no	2937	8852	135	yes	9	5574642.1	Middle
2	86929	100	yes	no	11	98155	3	4	2003	new	no	6326	4748	654	no	10	8696869.3	Luxury
3	51522	3	no	no	61	9047	10	5	2018	old	yes	8871	5792	807	yes	5	5154055.2	Middle
4	96470	74	yes	no	21	92029	6	9	2009	new	yes	4878	1172	716	yes	9	9652258.1	Luxury
5	79770	3	no	yes	69	54812	3	9	2011	new	yes	3054	7117	240	no	7	7986665.8	Luxury
6	75985	60	yes	no	67	6517	3	7	2000	old	no	7507	281	384	yes	5	7607322.9	Luxury
7	64169	88	no	yes	6	61711	3	4	2003	new	no	6326	129	726	no	9	6420823.1	Middle
8	92383	12	no	no	78	71982	3	7	2000	old	no	7507	9056	892	yes	1	9244344.0	Luxury
9	95121	46	no	yes	3	9382	9	5	2018	old	yes	5414	9515	440.4	Luxury			

[2]:
```
df_price2 = df_price.drop(['category'], axis=1)
df_price2.head()
```

[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
  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
```

[3]:
```
df_price2.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
```

[4]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

file_path = r'C:\Users\ACER\Downloads\UTS MESIN\DatasetUTS_Gasal2425.csv'
df_price2 = pd.read_csv(file_path)

categorical_columns = ['hasyard', 'haspool', 'isnewbuilt', 'hasstormprotector',␣
 ↪'hasstorageroom', 'category']

le = LabelEncoder()

for col in categorical_columns:
    if col in df_price2.columns:
        df_price2[col] = le.fit_transform(df_price2[col])
    else:
        print(f"Kolom {col} tidak ditemukan dalam DataFrame.")

print(df_price2.head())
```

```
   squaremeters numberofrooms hasyard haspool floors citycode \
0         75523            3       0       1     63     9373
1         55712           58       0       1     19    34457
2         86929          100       1       0     11    98155
3         51522            3       0       0     61     9047
4         96470           74       1       0     21    92029
   citypartrange numprevowners made isnewbuilt hasstormprotector \
```

```
0 3 8 2005 1                                                                    4

2 3 4 2003 0 03 8 3 2012 0                                                      4

4 4 2 2011 0                                                                    4

basement attic garage hasstorageroom hasguestroom price category0 4313
9005 956 0 7 7559081.5                                                          4

1 2937 8852 135 1 9 5574642.1                                                   4


1            6   8 2021     1     0

2            6326 4748      654   0     10 8696869.3     1

3            632 5792  807   1     5 5154055.2      2

4            5414 1172      716   1     9 9652258.1      1
```

[5]:
```
print(df_price2.head())
```

```
   squaremeters numberofrooms hasyard haspool floors citycode \
0         75523   3     0     1     63    9373

1         55712   58    0     1     19    34457

2         86929   100   1     0     11    98155

3         51522   3     0     0     61    9047

4         96470   74    1     0     21    92029

    citypartrange numprevowners made isnewbuilt hasstormprotector \
0             3             8 2005          1                 1

1             6             8 2021          1                 0

2             3             4 2003          0                 0

3             8             3 2012          0                 1

4             4             2 2011          0                 1

    basement attic garage hasstorageroom hasguestroom   price category
0      4313 9005     956              0            7 7559081.5        1

1      2937 8852     135              1            9 5574642.1        2

2       6326 4748 654   0     10 8696869.3     1

3       632 5792  807   1     5 5154055.2      2

4       5414 1172 716   1     9 9652258.1      1
```

[6]:
```
df_price2.describe()
```

[6]:

	squaremeters	numberofrooms	hasyard	haspool	floors
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	49870.13120	50.358400	0.508700	0.496800	50.276300
std	28774.37535	28.816696	0.499949	0.500015	28.889171
min	89.00000	1.000000	0.000000	0.000000	1.000000
25%	25098.50000	25.000000	0.000000	0.000000	25.000000
50%	50105.50000	50.000000	1.000000	0.000000	50.000000
75%	74609.75000	75.000000	1.000000	1.000000	76.000000
max	99999.00000	100.000000	1.000000	1.000000	100.000000

	citycode	citypartrange	numprevowners	made	isnewbuilt
count	10000.000000	10000.000000	10000.000000	10000.00000	
			10000.000000		
mean	50225.486100	5.510100	5.521700	2005.48850	0.500900
std	29006.675799	2.872024	2.856667	9.30809	0.500024
min	3.000000	1.000000	1.000000	1990.00000	0.000000
25%	24693.750000	3.000000	3.000000	1997.00000	0.000000
50%	50693.000000	5.000000	5.000000	2005.50000	1.000000
75%	75683.250000	8.000000	8.000000	2014.00000	1.000000
max	99953.000000	10.000000	10.000000	2021.00000	1.000000

	hasstormprotector	basement	attic	garage
count	10000.000000	10000.000000	10000.00000	10000.00000
mean	0.499900	5033.103900	5028.01060	553.12120
std	0.500025	2876.729545	2894.33221	262.05017
min	0.000000	0.000000	1.00000	100.00000
25%	0.000000	2559.750000	2512.00000	327.75000
50%	0.000000	5092.500000	5045.00000	554.00000
75%	1.000000	7511.250000	7540.50000	777.25000
max	1.000000	10000.000000	10000.00000	1000.00000

	hasstorageroom	hasguestroom	price	category
count	10000.000000	10000.00000	1.000000e+04	10000.000000
mean	0.503000	4.99460	4.993448e+06	0.824700
std	0.500016	3.17641	2.877424e+06	0.814148
min	0.000000	0.00000	1.031350e+04	0.000000
25%	0.000000	2.00000	2.516402e+06	0.000000
50%	1.000000	5.00000	5.016180e+06	1.000000
75%	1.000000	8.00000	7.469092e+06	2.000000
max	1.000000	10.00000	1.000677e+07	2.000000

[7]:
```
print(df_price2['price'].value_counts())
```

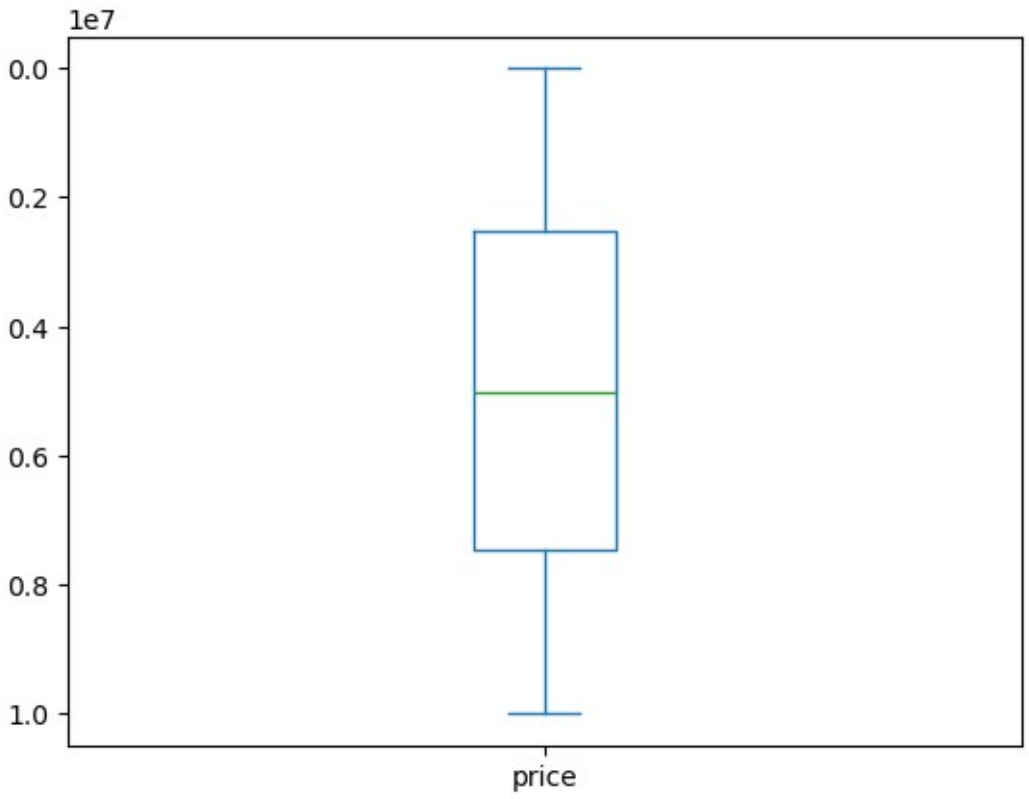
```
price
    7559081.5   1
    2600292.1   1
    3804577.4   1
    3658559.7   1
    2316639.4   1
..
    5555606.6   1
    5501007.5   1
    9986201.2   1
    9104801.8   1
    146708.4    1
    Name: count, Length: 10000, dtype: int64
```

[8]:
```
print("data null \n",
df_price2.isnull().sum()) print("data kosong
\n", df_price2.empty) print("data nan \n",
df_price2.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
price             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
price             0
category          0
dtype: int64
```

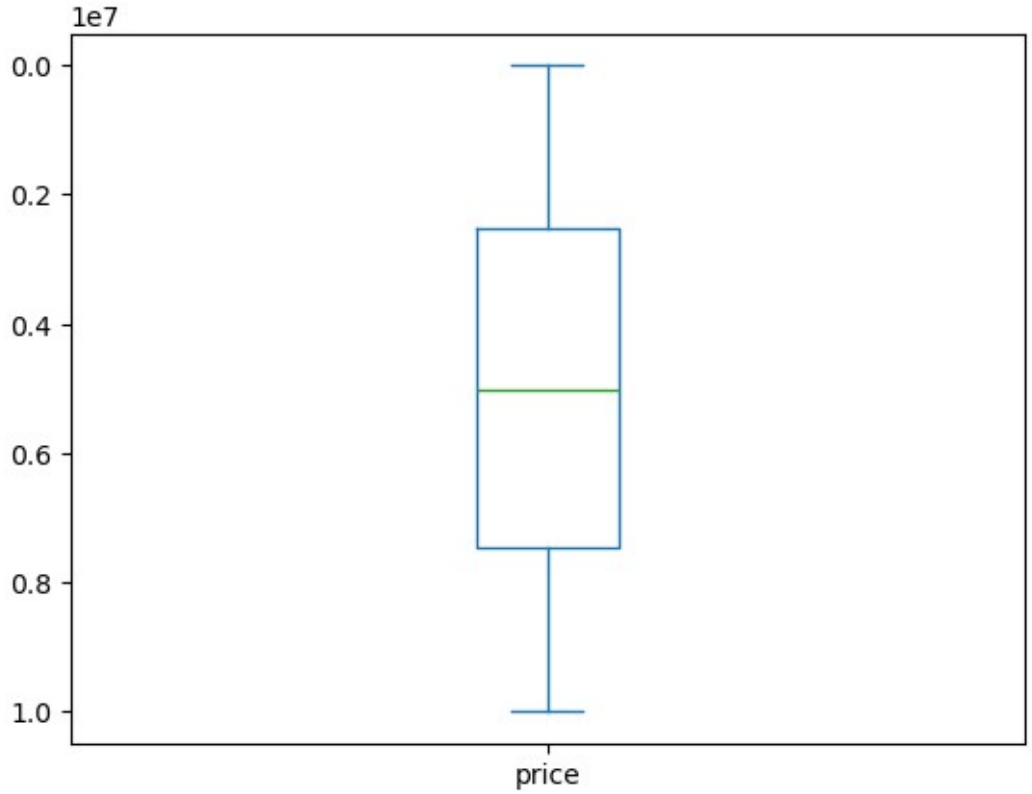
[9]:
```
import matplotlib.pylab as plt

df_price2.price.plot(kind='box')
plt.gca().invert_yaxis()
plt.show()
```



[10]:
```
from pandas.api.types import is_numeric_dtype
def remove_outlier(df_in):
    for col_name in list(df_in.columns):
        q1 = df_in[col_name].quantile(0.25)
        q3 = df_in[col_name].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_name] >= batas_bawah) & (df_in[col_name]
 ↪<= batas_atas)]
    return df_out

df_price_clean = remove_outlier(df_price2)
print("Jumlah baris DataFrame sebelum dibuang outliner ", df_price2.shape[0])
print("Jumlah baris DataFrame sesudah dibuang outliner ", df_price_clean.
 ↪shape[0])
df_price_clean.price.plot(kind='box', vert=True)

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

```
plt.show()
```

```
Jumlah baris DataFrame sebelum dibuang outliner
                                                10000
Jumlah baris DataFrame sesudah dibuang outliner
                                                10000
```



[11]:
```
print("data null \n",
df_price_clean.isnull().sum()) print("data
kosong \n", df_price_clean.empty) print("data
nan \n", df_price_clean.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
price              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
price              0
category           0
dtype: int64
```

[12]:
```
from sklearn.model_selection import train_test_split

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

X_train_price2, X_test_price2, y_train_price2, y_test_price2 =␣
        ↪train_test_split(X_regress, y_regress, test_size=0.25,
                                                random_state=94)
```

[14]:
```
from sklearn.preprocessing import StandardScaler,
MinMaxScaler from sklearn.feature_selection import
SelectKBest, SelectPercentile,␣
 ↪f_regression from sklearn.linear_model import Ridge from
sklearn.model_selection import GridSearchCV,
train_test_split from sklearn.pipeline import Pipeline
```

```
from sklearn.metrics import mean_squared_error,
mean_absolute_error import numpy as np
```

```
# Splitting dataset
X_regress = df_price_clean.drop('price', axis=1)
y_regress = df_price_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=94)

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

param_grid = [
    {
        'scale': [StandardScaler()],
        'feature_selection': [SelectKBest(score_func=f_regression)],
        'feature_selection__k': np.arange(1, X_train_price.shape[1] + 1),
        'reg__alpha': [0.01, 0.1, 1, 10, 100]
    },
    {
        'scale': [MinMaxScaler()],
        'feature_selection': [SelectKBest(score_func=f_regression)],
        'feature_selection__k': np.arange(1, X_train_price.shape[1] + 1),
        'reg__alpha': [0.01, 0.1, 1, 10, 100]
    },
    {
        'scale': [StandardScaler()],
        'feature_selection': [SelectPercentile(score_func=f_regression)],
        'feature_selection__percentile': np.arange(10, 101, 10),
        'reg__alpha': [0.01, 0.1, 1, 10, 100]
    },
    {
        'scale': [MinMaxScaler()],
        'feature_selection': [SelectPercentile(score_func=f_regression)],
        'feature_selection__percentile': np.arange(10, 101, 10),
        'reg__alpha': [0.01, 0.1, 1, 10, 100]
    }
]

GSCV_RR = GridSearchCV(pipe, param_grid, cv=5, scoring='neg_mean_squared_error')
GSCV_RR.fit(X_train_price, y_train_price)

print("Best model:{}".format(GSCV_RR.best_estimator_))
print("Best parameters:{}".format(GSCV_RR.best_params_))
```

```
y_pred = GSCV_RR.predict(X_test_price) mse =
mean_squared_error(y_test_price, y_pred) mae =
mean_absolute_error(y_test_price, y_pred)
print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("Root Mean Squared Error (RMSE):",
np.sqrt(mse))
```

```
Best model:Pipeline(steps=[('scale', StandardScaler()),
                ('feature_selection',
                 SelectKBest(k=17, score_func=<function
                             f_regression at
0x0000019DF7559440>)),
                ('reg', Ridge(alpha=0.01))])
Best parameters:{'feature_selection': SelectKBest(score_func=<function
f_regression at 0x0000019DF7559440>), 'feature_selection__k': 17,
'reg__alpha':
0.01, 'scale': StandardScaler()}
Mean Squared Error (MSE): 3662143.1317835227
Mean Absolute Error (MAE): 1498.5166835810023
Root Mean Squared Error (RMSE): 1913.672681464498
```

[15]:
```
df_result = pd.DataFrame(y_test_price2, columns=['price'])
df_result = pd.DataFrame(y_test_price2)
df_result['Ridge Prediction'] = y_pred

df_result['Selisih_price_RR'] = df_result['Ridge Prediction'] -␣
 ↪df_result['price']

df_result.head()
```

[15]:

	price	Ridge Prediction	Selisih_price_RR
7497	998783.8	9.918541e+05	-6929.703188
5188	7581701.0	7.584702e+06	3000.807025
5009	7674844.1	7.677885e+06	3041.138462
9434	610305.1	6.104511e+05	145.976398
2444	9153666.9	9.152004e+06	-1663.322269

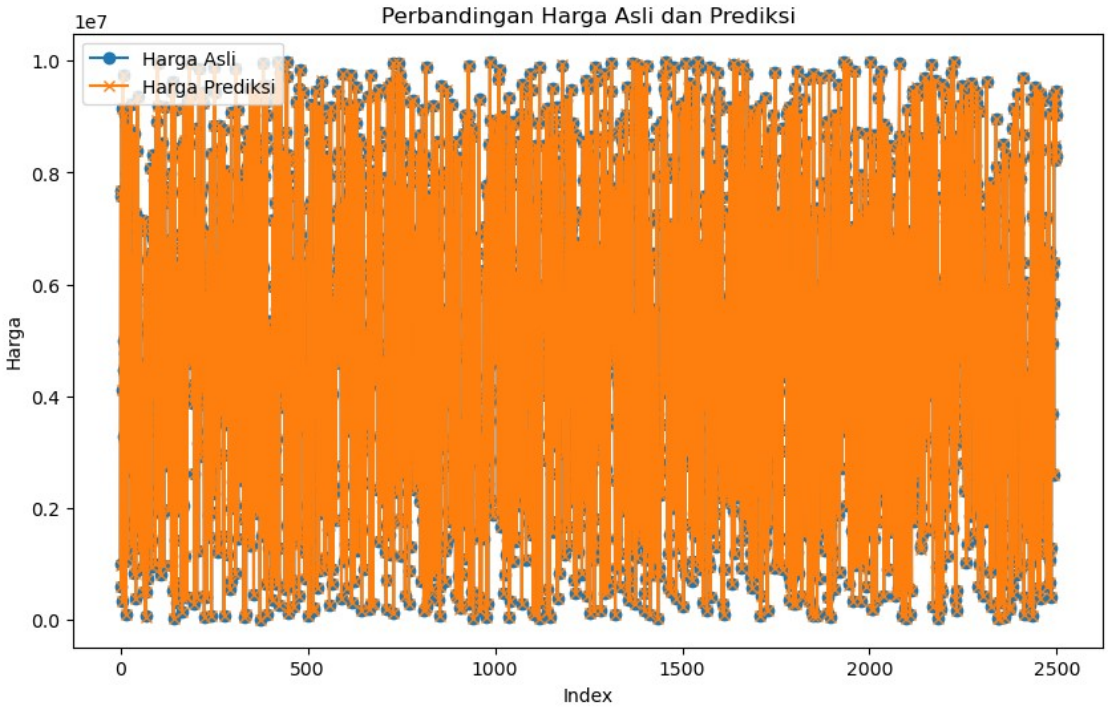
[16]:
```
df_result.describe()
```

[16]:

	price	Ridge Prediction	Selisih_price_RR
count	2.500000e+03	2.500000e+03	2500.000000
mean	4.961097e+06	4.961141e+06	43.764851
std	2.849760e+06	2.849746e+06	1913.554924
min	1.443130e+04	1.645097e+04	-6929.703188

```
25%   2.519592e+06   2.519722e+06   -
                                    1171.621615
50%   4.944934e+06   4.944243e+06      27.285594
75%   7.385645e+06   7.384477e+06    1249.497923
max   9.989599e+06   9.989396e+06    6324.114310
```

[17]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.plot(y_test_price.values, label='Harga Asli', marker='o')
plt.plot(y_pred, label='Harga Prediksi', marker='x')
plt.title('Perbandingan Harga Asli dan Prediksi ')
plt.xlabel('Index')
plt.ylabel('Harga')
plt.legend()
plt.show()
```



[18]:
```
import numpy as np from sklearn.svm
import SVR from sklearn.model_selection
import GridSearchCV from sklearn.pipeline
import Pipeline
from sklearn.preprocessing import StandardScaler,
MinMaxScaler from sklearn.feature_selection import
SelectKBest, SelectPercentile,␣
```

```
 ↪f_regression from sklearn.metrics import
mean_absolute_error, mean_squared_error

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

```
])

n_features = X_train_price.shape[1]

param_grid_SVR = [
    {
        'scale': [StandardScaler()],
        'feature_selection': [SelectKBest(score_func=f_regression)],
        'feature_selection__k': np.arange(1, n_features + 1),
        'reg__C': [0.01, 0.1, 1, 10, 100],
        'reg__epsilon': [0.1, 0.2, 0.5, 1]
    },
    {
        'scale': [MinMaxScaler()],
        'feature_selection': [SelectKBest(score_func=f_regression)],
        'feature_selection__k': np.arange(1, n_features + 1),
        'reg__C': [0.01, 0.1, 1, 10, 100],
        'reg__epsilon': [0.1, 0.2, 0.5, 1]
    },
    {
        'scale': [StandardScaler()],
        'feature_selection': [SelectPercentile(score_func=f_regression)],
        'feature_selection__percentile': np.arange(10, 101, 10),
        'reg__C': [0.01, 0.1, 1, 10, 100],
        'reg__epsilon': [0.1, 0.2, 0.5, 1]
    },
    {
        'scale': [MinMaxScaler()],
        'feature_selection': [SelectPercentile(score_func=f_regression)],
        'feature_selection__percentile': np.arange(10, 101, 10),
        'reg__C': [0.01, 0.1, 1, 10, 100],
        'reg__epsilon': [0.1, 0.2, 0.5, 1]
    }
]

GSCV_SVR = GridSearchCV(pipe_SVR, param_grid_SVR, cv=5,␣
 ↪scoring='neg_mean_squared_error')

GSCV_SVR.fit(X_train_price, y_train_price)

print("Best model: {}".format(GSCV_SVR.best_estimator_))
print("SVR best parameters: {}".format(GSCV_SVR.best_params_))

SVR_predict = GSCV_SVR.predict(X_test_price)

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

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

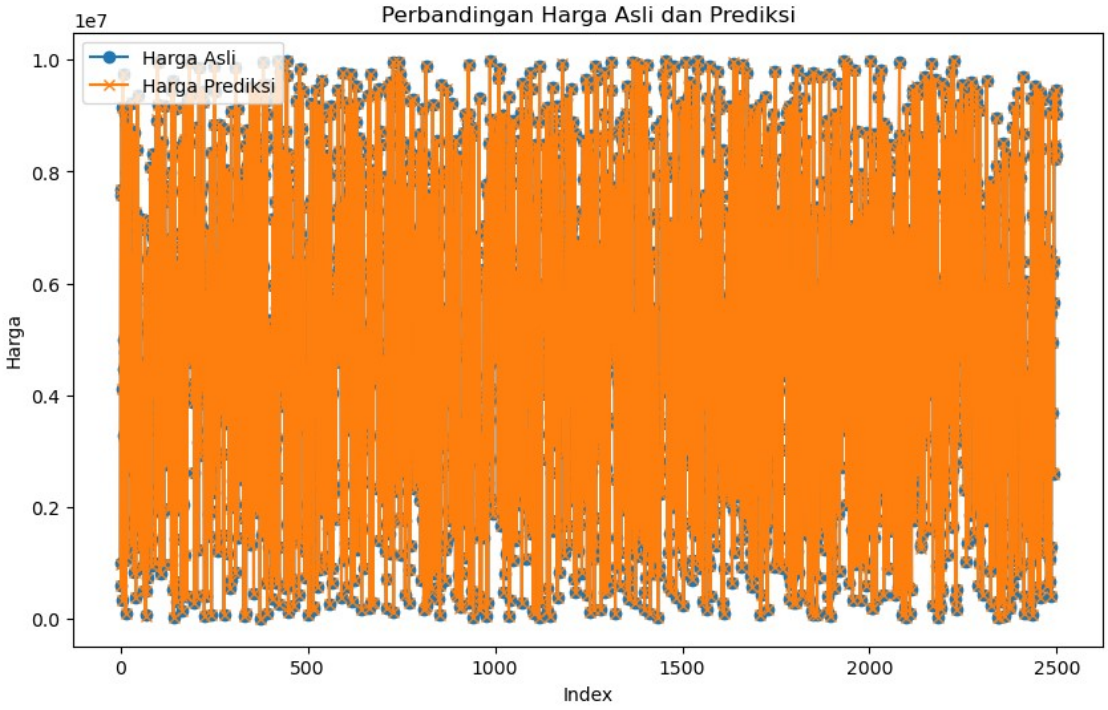
```
Best model: Pipeline(steps=[('scale', StandardScaler()),
                ('feature_selection',
                 SelectKBest(k=15, score_func=<function
f_regression at 0x0000019DF7559440>)),
                ('reg', SVR(C=100, epsilon=1, kernel='linear'))])
SVR best parameters: {'feature_selection':
SelectKBest(score_func=<function f_regression at
0x0000019DF7559440>), 'feature_selection__k': 15, 'reg__C': 100,
'reg__epsilon': 1, 'scale': StandardScaler()}
SVR Mean Squared Error (MSE): 4362982235410.3936
SVR Mean Absolute Error (MAE): 1724904.7501602867
SVR Root Mean Squared Error (RMSE): 2088775.2955764278
```

[19]:
```
df_result['SVR Prediction'] = SVR_predict
df_result = pd.DataFrame(y_test_price2)
df_result['SVR Prediction'] = SVR_predict

df_result['Selisih_price_SVR'] = df_result['SVR Prediction'] -␣
 ↪df_result['price']
df_result.head()
```

[19]:

	price	SVR Prediction	Selisih_price_SVR
7497	998783.8	3.673600e+06	2.674816e+06
5188	7581701.0	5.526782e+06	-2.054919e+06
5009	7674844.1	5.541527e+06	-2.133317e+06
9434	610305.1	3.595238e+06	2.984933e+06
2444	9153666.9	5.903871e+06	-3.249795e+06

[20]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.plot(y_test_price.values, label='Harga Asli', marker='o')
plt.plot(y_pred, label='Harga Prediksi', marker='x')
plt.title('Perbandingan Harga Asli dan Prediksi ')
plt.xlabel('Index')
plt.ylabel('Harga')
plt.legend()
plt.show()
```



[21]:
```
df_result.describe()
```

[21]:

	price	SVR Prediction	Selisih_price_SVR
count	2.500000e+03	2.500000e+03	2.500000e+03
mean	4.961097e+06	4.889043e+06	-7.205365e+04
std	2.849760e+06	8.363125e+05	2.087950e+06
min	1.443130e+04	3.430104e+06	-3.962214e+06
25%	2.519592e+06	4.116152e+06	-1.897693e+06
50%	4.944934e+06	5.263951e+06	5.530077e+03
75%	7.385645e+06	5.649964e+06	1.610137e+06

```
max    9.989599e+06    6.117743e+06       4.231090e+06
```

[22]:
```
df_results= pd.DataFrame({'price': y_test_price2})

df_results['Ridge Prediction'] = y_pred
df_results['Selisih_price_RR'] = df_results['price'] - df_results['Ridge␣
 ↪Prediction']

df_results['SVR Prediction'] = SVR_predict
df_results['Selisih_price_SVR'] = df_results['price'] - df_results['SVR␣
 ↪Prediction']

df_results.head()
```

[22]:
```
       price Ridge Prediction Selisih_price_RR SVR Prediction \
7497  998783.8    9.918541e+05     6929.703188  3.673600e+06
5188 7581701.0    7.584702e+06    -3000.807025  5.526782e+06
5009 7674844.1    7.677885e+06    -3041.138462  5.541527e+06
9434  610305.1    6.104511e+05     -145.976398  3.595238e+06
2444 9153666.9    9.152004e+06     1663.322269  5.903871e+06
     Selisih_price_SVR
7497    -2.674816e+06
5188     2.054919e+06
5009 2.133317e+06 9434
-2.984933e+06
2444     3.249795e+06
```

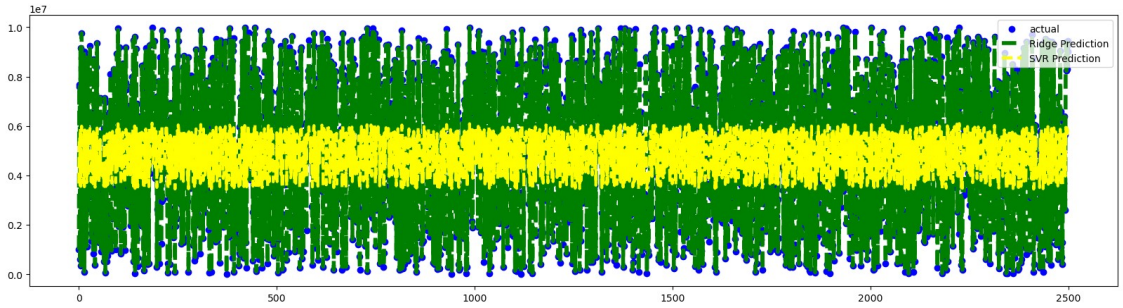
[24]:
```
df_results.describe()
```

[24]:
```
          price Ridge Prediction Selisih_price_RR SVR Prediction \
count 2.500000e+03   2.500000e+03    2500.000000  2.500000e+03
mean  4.961097e+06   4.961141e+06     -43.764851  4.889043e+06
std   2.849760e+06   2.849746e+06    1913.554924  8.363125e+05
min   1.443130e+04   1.645097e+04   -6324.114310  3.430104e+06
25%   2.519592e+06   2.519722e+06   -1249.497923  4.116152e+06
50%   4.944934e+06   4.944243e+06     -27.285594  5.263951e+06
75%   7.385645e+06   7.384477e+06    1171.621615  5.649964e+06
max   9.989599e+06   9.989396e+06    6929.703188  6.117743e+06

      Selisih_price_SVR
count     2.500000e+03

mean      7.205365e+04

std       2.087950e+06

min      -4.231090e+06

25%      -1.610137e+06
```

```
50%       -5.530077e+03
75%        1.897693e+06
max        3.962214e+06
```

[25]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(20, 5))

data_len = range(len(y_test_price2))

plt.scatter(data_len, df_results.price, label="actual", color="blue")

plt.plot(data_len, df_results['Ridge Prediction'], label="Ridge Prediction",␣
 ↪color="green", linewidth=4, linestyle="dashed")

plt.plot(data_len, df_results['SVR Prediction'], label="SVR Prediction",␣
 ↪color="yellow", linewidth=3, linestyle="--")

plt.legend()
plt.show
```

[25]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



[26]:
```
from sklearn.metrics import mean_absolute_error,
mean_squared_error import numpy as np

mae_ridge = mean_absolute_error(df_results['price'],
df_results['Ridge␣
 ↪Prediction']) rmse_ridge =
np.sqrt(mean_squared_error(df_results['price'], df_results['Ridge␣
 ↪Prediction'])) ridge_feature_count =
GSCV.best_params_['feature_selection__k']
```

```
mae_svr = mean_absolute_error(df_results['price'], df_results['SVR
Prediction']) rmse_svr =
np.sqrt(mean_squared_error(df_results['price'], df_results['SVR␣
 ↪Prediction'])) svr_feature_count =
GSCV_SVR.best_params_['feature_selection__k']

print(f"Ridge MAE: {mae_ridge}, Ridge RMSE: {rmse_ridge}, Ridge
Feature Count:␣
 ↪{ridge_feature_count}") print(f"SVR MAE: {mae_svr}, SVR
RMSE: {rmse_svr}, SVR Feature Count:␣
↪{svr_feature_count}")
```

```
Ridge MAE: 1498.5166835810023, Ridge RMSE: 1913.672681464498, Ridge
Feature Count: 17
SVR MAE: 1724904.7501602867, SVR RMSE: 2088775.2955764278, SVR Feature
                                                        Count: 15
```

[27]:
```
import pickle

best_model = GSCV_RR.best_estimator_
with open('BestModel_REG_RidgeRegression_Transformers.pkl', 'wb') as
f:
    pickle.dump(best_model, f)
    print("Model terbaik berhasil disimpan ke␣
 ↪'BestModel_REG_RidgeRegression_Transformers.pkl'")
```

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

sformer-lasso-vs-randomforest-tutu

October 25, 2024

[141]:
```
import pandas as pd
import numpy as np

df_price = pd.read_csv(r'D:\ATMAJAYA\Semester5\PMDPL\ProjectUTS\Dataset␣
↪UTS_Gasal 2425.csv')
df_price.head(10)
```

[141]:

	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
```

[142]:
```
df_price2 = df_price.drop(['category'], axis=1)
df_price2.head()
```

[142]:
```
   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
```

[143]:
```
print(df_price2.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
None
```

[144]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

file_path = r'D:\ATMAJAYA\Semester5\PMDPL\ProjectUTS\Dataset UTS_Gasal 2425.csv'
df_price2 = pd.read_csv(file_path)

categorical_columns = ['hasyard', 'haspool', 'isnewbuilt', 'hasstormprotector',␣
 ↪'hasstorageroom', 'category']

le = LabelEncoder()

for col in categorical_columns:
    if col in df_price2.columns:
        df_price2[col] = le.fit_transform(df_price2[col])
    else:
        print(f"Kolom {col} tidak ditemukan dalam DataFrame.")

print(df_price2.head())
```

```
   squaremeters  numberofrooms  hasyard  haspool  floors  citycode  \
0         75523              3        0        1      63      9373
1         55712             58        0        1      19     34457
2         86929            100        1        0      11     98155
3         51522              3        0        0      61      9047
4         96470             74        1        0      21     92029

   citypartrange  numprevowners  made  isnewbuilt  hasstormprotector  \
0              3              8  2005           1                  1
1              6              8  2021           1                  0
2              3              4  2003           0                  0
3              8              3  2012           0                  1
4              4              2  2011           0                  1

   basement  attic  garage  hasstorageroom  hasguestroom      price  category
0      4313   9005     956               0             7  7559081.5         1
1      2937   8852     135               1             9  5574642.1         2
2      6326   4748     654               0            10  8696869.3         1
```

```
3       632   5792     807                1             5  5154055.2         2
4      5414   1172     716                1             9  9652258.1         1
```

[145]:
```
print(df_price2.head())
```

```
   squaremeters  numberofrooms  hasyard  haspool  floors  citycode  \
0         75523              3        0        1      63      9373
1         55712             58        0        1      19     34457
2         86929            100        1        0      11     98155
3         51522              3        0        0      61      9047
4         96470             74        1        0      21     92029

   citypartrange  numprevowners  made  isnewbuilt  hasstormprotector  \
0              3              8  2005           1                  1
1              6              8  2021           1                  0
2              3              4  2003           0                  0
3              8              3  2012           0                  1
4              4              2  2011           0                  1

   basement  attic  garage  hasstorageroom  hasguestroom      price  category
0      4313   9005     956               0             7  7559081.5         1
1      2937   8852     135               1             9  5574642.1         2
2      6326   4748     654               0            10  8696869.3         1
3       632   5792     807               1             5  5154055.2         2
4      5414   1172     716               1             9  9652258.1         1
```

[146]:
```
df_price2.describe()
```

[146]:
```
       squaremeters  numberofrooms       hasyard       haspool        floors  \
count   10000.00000   10000.000000  10000.000000  10000.000000  10000.000000
mean    49870.13120      50.358400      0.508700      0.496800     50.276300
std     28774.37535      28.816696      0.499949      0.500015     28.889171
min        89.00000       1.000000      0.000000      0.000000      1.000000
25%     25098.50000      25.000000      0.000000      0.000000     25.000000
50%     50105.50000      50.000000      1.000000      0.000000     50.000000
75%     74609.75000      75.000000      1.000000      1.000000     76.000000
max     99999.00000     100.000000      1.000000      1.000000    100.000000

            citycode  citypartrange  numprevowners         made    isnewbuilt  \
count   10000.000000   10000.000000   10000.000000  10000.00000  10000.000000
mean    50225.486100       5.510100       5.521700   2005.48850      0.500900
std     29006.675799       2.872024       2.856667      9.30809      0.500024
min         3.000000       1.000000       1.000000   1990.00000      0.000000
25%     24693.750000       3.000000       3.000000   1997.00000      0.000000
50%     50693.000000       5.000000       5.000000   2005.50000      1.000000
75%     75683.250000       8.000000       8.000000   2014.00000      1.000000
max     99953.000000      10.000000      10.000000   2021.00000      1.000000
```

```
       hasstormprotector      basement        attic       garage  \
count       10000.000000  10000.000000  10000.00000  10000.00000
mean            0.499900   5033.103900   5028.01060    553.12120
std             0.500025   2876.729545   2894.33221    262.05017
min             0.000000      0.000000      1.00000    100.00000
25%             0.000000   2559.750000   2512.00000    327.75000
50%             0.000000   5092.500000   5045.00000    554.00000
75%             1.000000   7511.250000   7540.50000    777.25000
max             1.000000  10000.000000  10000.00000   1000.00000

       hasstorageroom  hasguestroom         price      category
count    10000.000000   10000.00000  1.000000e+04  10000.000000
mean         0.503000       4.99460  4.993448e+06      0.824700
std          0.500016       3.17641  2.877424e+06      0.814148
min          0.000000       0.00000  1.031350e+04      0.000000
25%          0.000000       2.00000  2.516402e+06      0.000000
50%          1.000000       5.00000  5.016180e+06      1.000000
75%          1.000000       8.00000  7.469092e+06      2.000000
max          1.000000      10.00000  1.000677e+07      2.000000
```

[147]:
```
print(df_price2['price'].value_counts())
```

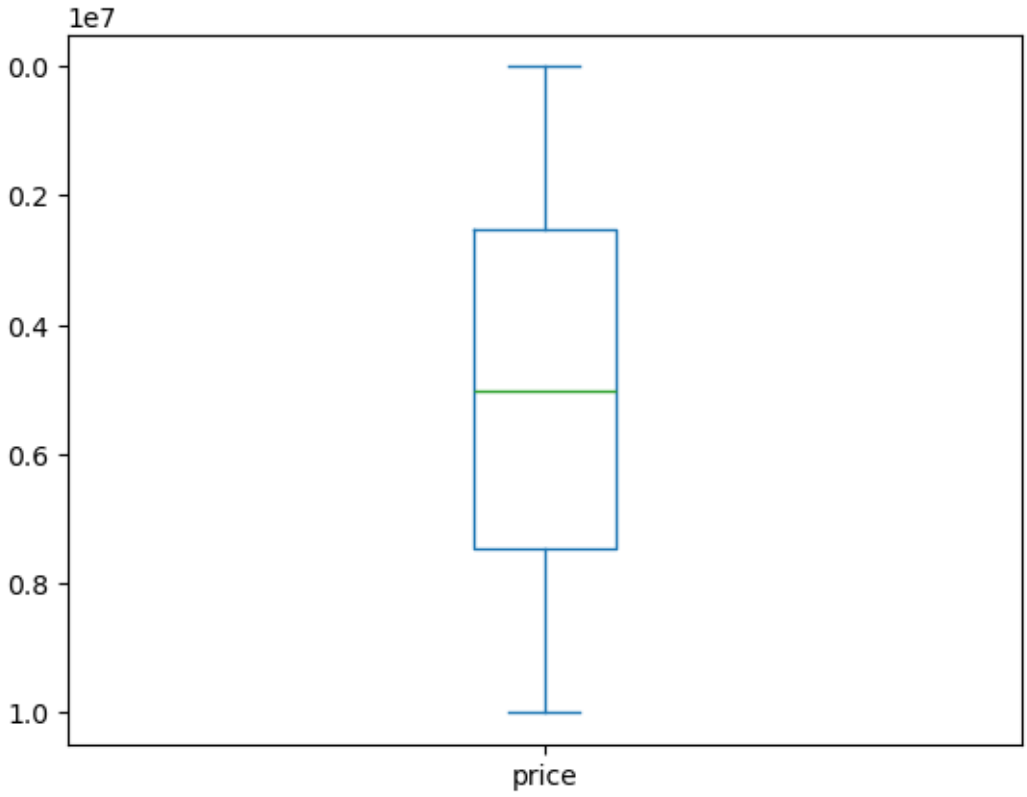
```
price
7559081.5    1
2600292.1    1
3804577.4    1
3658559.7    1
2316639.4    1
            ..
5555606.6    1
5501007.5    1
9986201.2    1
9104801.8    1
146708.4     1
Name: count, Length: 10000, dtype: int64
```

[148]:
```
print("data null \n", df_price2.isnull().sum())
print("data kosong \n", df_price2.empty)
print("data nan \n", df_price2.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
price                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
price                0
category             0
dtype: int64
```

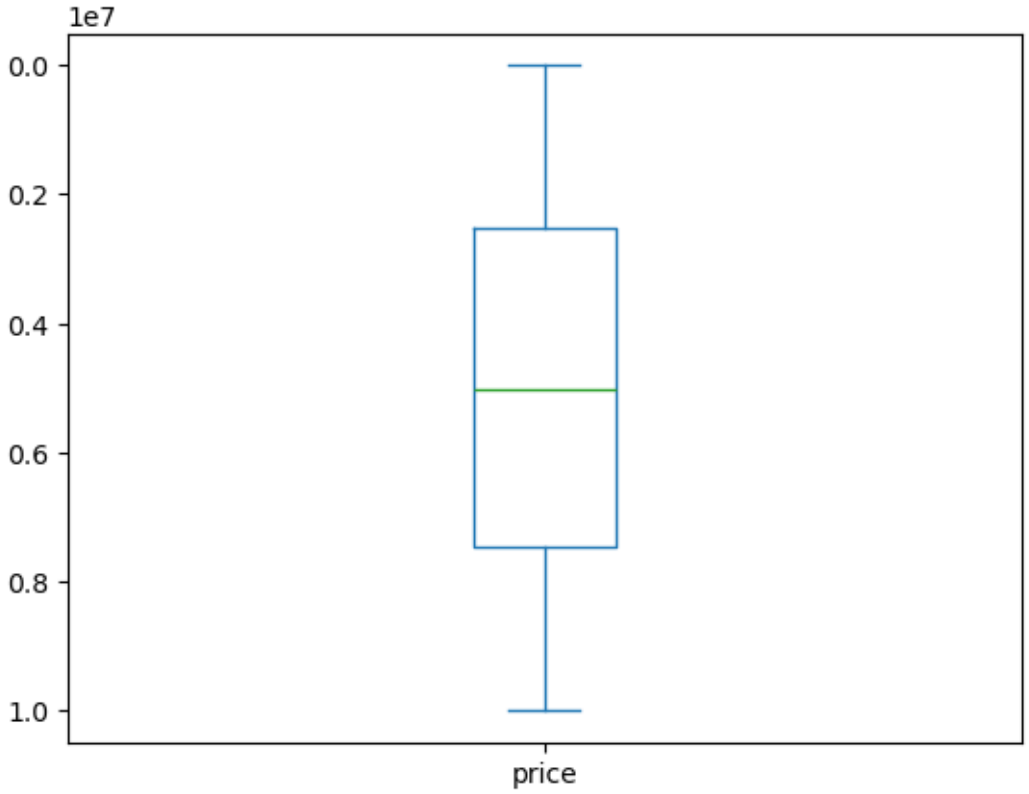
[149]:
```
import matplotlib.pylab as plt

df_price2.price.plot(kind='box')
plt.gca().invert_yaxis()
plt.show()
```



[150]:
```
from pandas.api.types import is_numeric_dtype
def remove_outlier(df_in):
    for col_name in list(df_in.columns):
        q1 = df_in[col_name].quantile(0.25)
        q3 = df_in[col_name].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_name] >= batas_bawah) & (df_in[col_name]␣
 ↪<= batas_atas)]
    return df_out

df_price_clean = remove_outlier(df_price2)
print("Jumlah baris DataFrame sebelum dibuang outliner", df_price2.shape[0])
print("Jumlah baris DataFrame sesudah dibuang outliner", df_price_clean.
 ↪shape[0])
df_price_clean.price.plot(kind='box', vert=True)

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

```
plt.show()
```

```
Jumlah baris DataFrame sebelum dibuang outliner 10000
Jumlah baris DataFrame sesudah dibuang outliner 10000
```



[151]:
```
print("data null \n", df_price_clean.isnull().sum())
print("data kosong \n", df_price_clean.empty)
print("data nan \n", df_price_clean.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
price                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
price                0
category             0
dtype: int64
```

[152]:
```
from sklearn.model_selection import train_test_split

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

X_train_price2, X_test_price2, y_train_price2, y_test_price2 =␣
 ↪train_test_split(X_regress, y_regress, test_size=0.25, random_state=94)
```

[153]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.feature_selection import SelectKBest, SelectPercentile,␣
 ↪f_regression
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, mean_absolute_error
import numpy as np
```

```
# Splitting dataset
X_regress = df_price_clean.drop('price', axis=1)
y_regress = df_price_clean['price']

X_train_price2, X_test_price2, y_train_price2, y_test_price2 =␣
 ↪train_test_split(X_regress, y_regress, test_size=0.25, random_state=94)

# Pipeline configuration for Lasso regression
pipe_lasso = Pipeline(steps=[
    ('scale', StandardScaler()),  # or MinMaxScaler() - we'll test both
    ('feature_selection', SelectKBest(score_func=f_regression)),  # or␣
 ↪SelectPercentile
    ('reg', Lasso())
])

# Parameter grid for both StandardScaler and MinMaxScaler with SelectKBest &␣
 ↪SelectPercentile
param_grid_lasso = [
    {
        'scale': [StandardScaler()],
        'feature_selection': [SelectKBest(score_func=f_regression)],  # k␣
 ↪digunakan untuk SelectKBest
        'feature_selection__k': np.arange(1, X_train_price2.shape[1] + 1),
        'reg__alpha': [0.01, 0.1, 1, 10, 100]
    },
    {
        'scale': [MinMaxScaler()],
        'feature_selection': [SelectKBest(score_func=f_regression)],
        'feature_selection__k': np.arange(1, X_train_price2.shape[1] + 1),
        'reg__alpha': [0.01, 0.1, 1, 10, 100]
    },
    {
        'scale': [StandardScaler()],
        'feature_selection': [SelectPercentile(score_func=f_regression)],  #␣
 ↪percentile digunakan untuk SelectPercentile
        'feature_selection__percentile': np.arange(10, 101, 10),
        'reg__alpha': [0.01, 0.1, 1, 10, 100]
    },
    {
        'scale': [MinMaxScaler()],
        'feature_selection': [SelectPercentile(score_func=f_regression)],
        'feature_selection__percentile': np.arange(10, 101, 10),
        'reg__alpha': [0.01, 0.1, 1, 10, 100]
    }
]

# Perform grid search for Lasso
```

```
GSCV_lasso = GridSearchCV(pipe_lasso, param_grid_lasso, cv=5,␣
 ↪scoring='neg_mean_squared_error')
GSCV_lasso.fit(X_train_price2, y_train_price2)

# Output best model parameters for Lasso
print("Best model (Lasso): {}".format(GSCV_lasso.best_estimator_))
print("Best parameters (Lasso): {}".format(GSCV_lasso.best_params_))

# Predictions and evaluation for Lasso
y_pred_lasso = GSCV_lasso.predict(X_test_price2)
mse_lasso = mean_squared_error(y_test_price2, y_pred_lasso)
mae_lasso = mean_absolute_error(y_test_price2, y_pred_lasso)
print("Mean Squared Error (Lasso):", mse_lasso)
print("Mean Absolute Error (Lasso):", mae_lasso)
print("Root Mean Squared Error (Lasso):", np.sqrt(mse_lasso))
```

```
Best model (Lasso): Pipeline(steps=[('scale', StandardScaler()),
                ('feature_selection',
                 SelectKBest(k=17,
                             score_func=<function f_regression at
0x000001EA8DD6BCE0>)),
                ('reg', Lasso(alpha=10))])
Best parameters (Lasso): {'feature_selection': SelectKBest(score_func=<function
f_regression at 0x000001EA8DD6BCE0>), 'feature_selection__k': 17, 'reg__alpha':
10, 'scale': StandardScaler()}
Mean Squared Error (Lasso): 3659351.6245070854
Mean Absolute Error (Lasso): 1497.9728437973881
Root Mean Squared Error (Lasso): 1912.9431838157361
```

[154]:
```
df_result = pd.DataFrame(y_test_price2, columns=['price'])
df_result = pd.DataFrame(y_test_price2)
df_result['Lasso Prediction'] = y_pred_lasso

df_result['Selisih_price_RR'] = df_result['Lasso Prediction'] -␣
 ↪df_result['price']

df_result.head()
```

[154]:

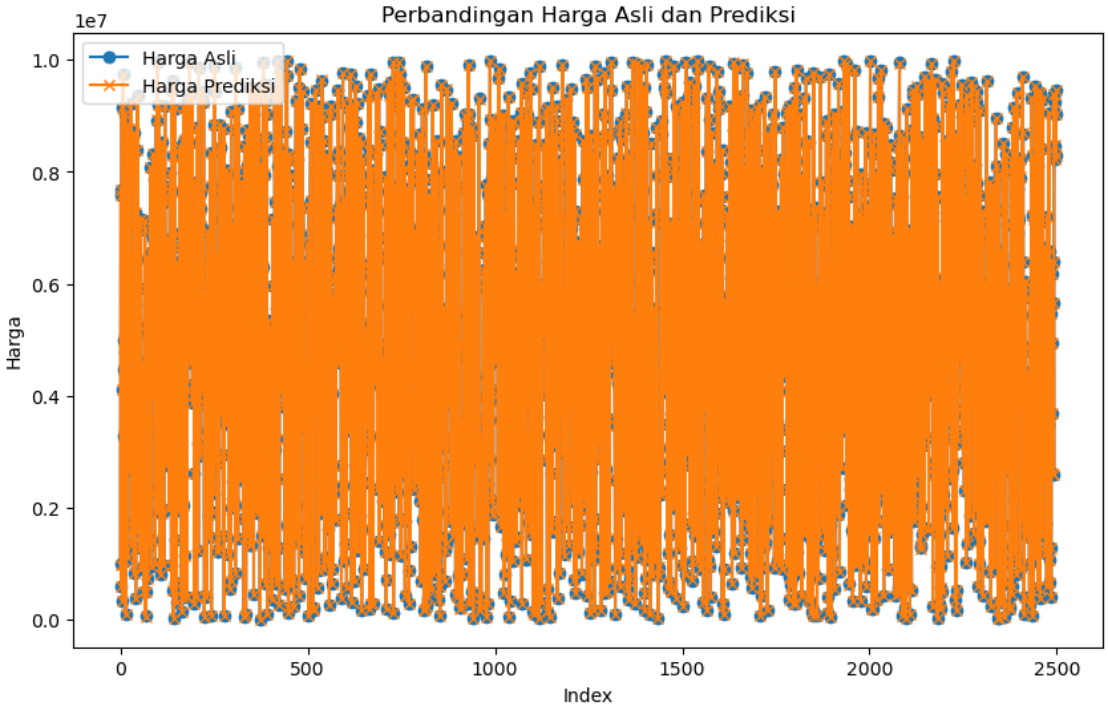
	price	Lasso Prediction	Selisih_price_RR
7497	998783.8	9.918587e+05	-6925.064493
5188	7581701.0	7.584649e+06	2948.137848
5009	7674844.1	7.677844e+06	2999.946262
9434	610305.1	6.105380e+05	232.942118
2444	9153666.9	9.152001e+06	-1665.822762

[155]:
```
df_result.describe()
```

[155]:

	price	Lasso Prediction	Selisih_price_RR
count	2.500000e+03	2.500000e+03	2500.000000
mean	4.961097e+06	4.961143e+06	45.901040
std	2.849760e+06	2.849735e+06	1912.775000
min	1.443130e+04	1.647232e+04	-6925.064493
25%	2.519592e+06	2.519774e+06	-1178.764843
50%	4.944934e+06	4.944266e+06	39.747655
75%	7.385645e+06	7.384420e+06	1257.089650
max	9.989599e+06	9.989413e+06	6294.773757

[156]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.plot(y_test_price2.values, label='Harga Asli', marker='o')
plt.plot(y_pred_lasso, label='Harga Prediksi', marker='x')
plt.title('Perbandingan Harga Asli dan Prediksi')
plt.xlabel('Index')
plt.ylabel('Harga')
plt.legend()
plt.show()
```



[158]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.feature_selection import SelectKBest, SelectPercentile,␣
 ↪f_regression
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, mean_absolute_error
import numpy as np

# Splitting dataset
X_regress = df_price_clean.drop('price', axis=1)
y_regress = df_price_clean['price']

X_train_price2, X_test_price2, y_train_price2, y_test_price2 =␣
 ↪train_test_split(X_regress, y_regress, test_size=0.25, random_state=94)

# Pipeline configuration for Random Forest regression
pipe_rf = Pipeline(steps=[
    ('scale', StandardScaler()),  # or MinMaxScaler() - we'll test both
    ('feature_selection', SelectKBest(score_func=f_regression)),  # or␣
 ↪SelectPercentile
    ('reg', RandomForestRegressor())
])

# Parameter grid for both StandardScaler and MinMaxScaler with SelectKBest &␣
 ↪SelectPercentile
param_grid_rf = [
    {
        'scale': [StandardScaler()],
        'feature_selection': [SelectKBest(score_func=f_regression)],  # k␣
 ↪digunakan untuk SelectKBest
        'feature_selection__k': np.arange(1, X_train_price2.shape[1] + 1),
        'reg__n_estimators': [50],  # Hanya 50 estimators
        'reg__max_depth': [None, 10],  # Hanya 2 pilihan untuk max_depth
        'reg__min_samples_split': [2],  # Hanya 1 pilihan untuk␣
 ↪min_samples_split
        'reg__min_samples_leaf': [1]  # Hanya 1 pilihan untuk min_samples_leaf
    },
    {
        'scale': [MinMaxScaler()],
        'feature_selection': [SelectKBest(score_func=f_regression)],
        'feature_selection__k': np.arange(1, X_train_price2.shape[1] + 1),
        'reg__n_estimators': [50],
        'reg__max_depth': [None, 10],
        'reg__min_samples_split': [2],
        'reg__min_samples_leaf': [1]
    },
    {
        'scale': [StandardScaler()],
```

```
        'feature_selection': [SelectPercentile(score_func=f_regression)],  #␣
 ↪percentile digunakan untuk SelectPercentile
        'feature_selection__percentile': np.arange(10, 101, 10),
        'reg__n_estimators': [50],
        'reg__max_depth': [None, 10],
        'reg__min_samples_split': [2],
        'reg__min_samples_leaf': [1]
    },
    {
        'scale': [MinMaxScaler()],
        'feature_selection': [SelectPercentile(score_func=f_regression)],
        'feature_selection__percentile': np.arange(10, 101, 10),
        'reg__n_estimators': [50],
        'reg__max_depth': [None, 10],
        'reg__min_samples_split': [2],
        'reg__min_samples_leaf': [1]
    }
]

# Perform grid search for Random Forest
GSCV_rf = GridSearchCV(pipe_rf, param_grid_rf, cv=5,␣
 ↪scoring='neg_mean_squared_error')
GSCV_rf.fit(X_train_price2, y_train_price2)

# Output best model parameters for Random Forest
print("Best model (Random Forest): {}".format(GSCV_rf.best_estimator_))
print("Best parameters (Random Forest): {}".format(GSCV_rf.best_params_))

# Predictions and evaluation for Random Forest
y_pred_rf = GSCV_rf.predict(X_test_price2)
mse_rf = mean_squared_error(y_test_price2, y_pred_rf)
mae_rf = mean_absolute_error(y_test_price2, y_pred_rf)
print("Mean Squared Error (Random Forest):", mse_rf)
print("Mean Absolute Error (Random Forest):", mae_rf)
print("Root Mean Squared Error (Random Forest):", np.sqrt(mse_rf))
```

```
Best model (Random Forest): Pipeline(steps=[('scale', MinMaxScaler()),
                ('feature_selection',
                 SelectPercentile(percentile=20,
                                  score_func=<function f_regression at
0x000001EA8DD6BCE0>)),
                ('reg', RandomForestRegressor(n_estimators=50))])
Best parameters (Random Forest): {'feature_selection':
SelectPercentile(score_func=<function f_regression at 0x000001EA8DD6BCE0>),
'feature_selection__percentile': 20, 'reg__max_depth': None,
'reg__min_samples_leaf': 1, 'reg__min_samples_split': 2, 'reg__n_estimators':
50, 'scale': MinMaxScaler()}
```

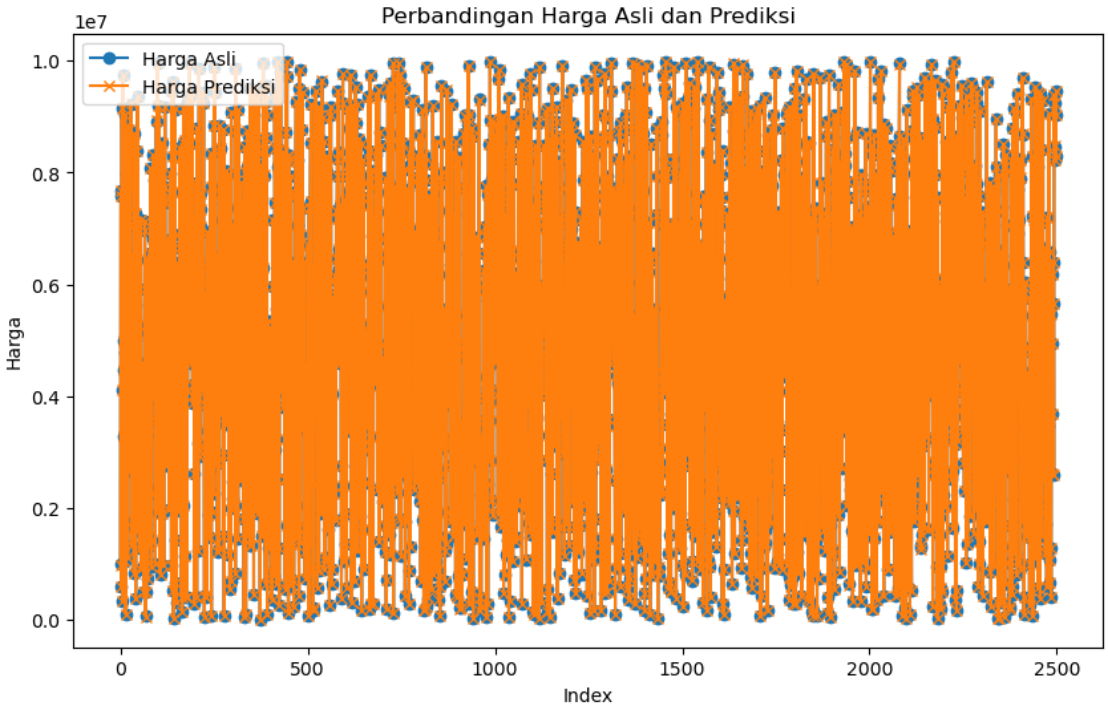
```
Mean Squared Error (Random Forest): 15922644.296667071
Mean Absolute Error (Random Forest): 3184.566835199974
Root Mean Squared Error (Random Forest): 3990.3188214310735
```

[159]:
```
df_result['Random_Forest Prediction'] = y_pred_rf
df_result = pd.DataFrame(y_test_price2)
df_result['Random_Forest Prediction'] = y_pred_rf

df_result['Selisih_price_Random_Forest'] = df_result['Random_Forest␣
 ↪Prediction'] - df_result['price']
df_result.head()
```

[159]:

	price	Random_Forest Prediction	Selisih_price_Random_Forest
7497	998783.8	988767.216	-10016.584
5188	7581701.0	7580327.138	-1373.862
5009	7674844.1	7672067.618	-2776.482
9434	610305.1	611217.066	911.966
2444	9153666.9	9149066.952	-4599.948

[160]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.plot(y_test_price2.values, label='Harga Asli', marker='o')
plt.plot(y_pred_lasso, label='Harga Prediksi', marker='x')
plt.title('Perbandingan Harga Asli dan Prediksi')
plt.xlabel('Index')
plt.ylabel('Harga')
plt.legend()
plt.show()
```



[161]:
```
df_result.describe()
```

[161]:

	price	Random_Forest Prediction	Selisih_price_Random_Forest
count	2.500000e+03	2.500000e+03	2500.000000
mean	4.961097e+06	4.961144e+06	47.203944
std	2.849760e+06	2.849675e+06	3990.837857
min	1.443130e+04	1.327065e+04	-13562.556000
25%	2.519592e+06	2.520742e+06	-2592.344000
50%	4.944934e+06	4.942407e+06	128.550000
75%	7.385645e+06	7.383038e+06	2791.889000
max	9.989599e+06	9.993591e+06	12176.494000

[162]:
```
df_results= pd.DataFrame({'price': y_test_price2})

df_results['Lasso Prediction'] = y_pred_lasso
df_results['Selisih_price_RR'] = df_results['price'] - df_results['Lasso␣
 ↪Prediction']

df_results['RF Prediction'] = y_pred_rf
df_results['Selisih_price_RF'] = df_results['price'] - df_results['RF␣
 ↪Prediction']

df_results.head()
```

[162]:
```
          price  Lasso Prediction  Selisih_price_RR  RF Prediction  \
7497   998783.8      9.918587e+05       6925.064493     988767.216
5188  7581701.0      7.584649e+06      -2948.137848    7580327.138
5009  7674844.1      7.677844e+06      -2999.946262    7672067.618
9434   610305.1      6.105380e+05       -232.942118     611217.066
2444  9153666.9      9.152001e+06       1665.822762    9149066.952

      Selisih_price_RF
7497         10016.584
5188          1373.862
5009          2776.482
9434          -911.966
2444          4599.948
```

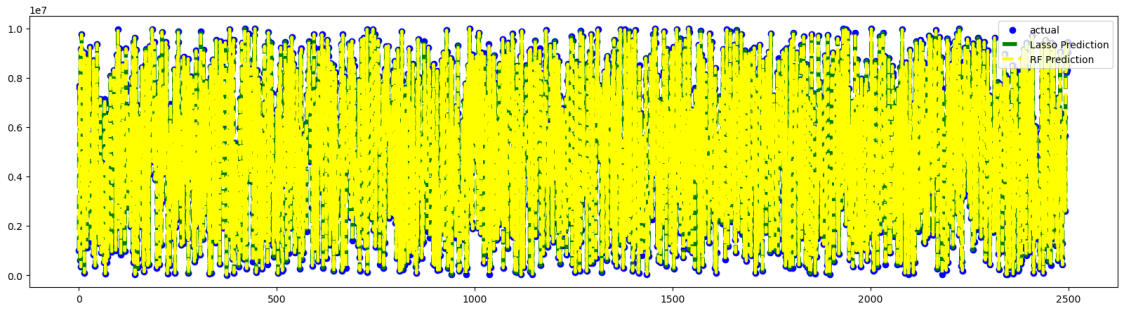
[163]:
```
df_results.describe()
```

[163]:
```
              price  Lasso Prediction  Selisih_price_RR  RF Prediction  \
count  2.500000e+03      2.500000e+03       2500.000000   2.500000e+03
mean   4.961097e+06      4.961143e+06        -45.901040   4.961144e+06
std    2.849760e+06      2.849735e+06       1912.775000   2.849675e+06
min    1.443130e+04      1.647232e+04      -6294.773757   1.327065e+04
25%    2.519592e+06      2.519774e+06      -1257.089650   2.520742e+06
50%    4.944934e+06      4.944266e+06        -39.747655   4.942407e+06
75%    7.385645e+06      7.384420e+06       1178.764843   7.383038e+06
max    9.989599e+06      9.989413e+06       6925.064493   9.993591e+06

       Selisih_price_RF
count       2500.000000
mean         -47.203944
std         3990.837857
min       -12176.494000
25%        -2791.889000
50%         -128.550000
75%         2592.344000
max        13562.556000
```

[164]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(20, 5))

data_len = range(len(y_test_price2))

plt.scatter(data_len, df_results.price, label="actual", color="blue")

plt.plot(data_len, df_results['Lasso Prediction'], label="Lasso Prediction",␣
 ↪color="green", linewidth=4, linestyle="dashed")
```

```
plt.plot(data_len, df_results['RF Prediction'], label="RF Prediction",␣
 ↪color="yellow", linewidth=3, linestyle="--")

plt.legend()
plt.show
```

[164]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



[166]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

# Calculate evaluation metrics for Lasso
mae_Lasso = mean_absolute_error(df_results['price'], df_results['Lasso␣
 ↪Prediction'])
rmse_Lasso = np.sqrt(mean_squared_error(df_results['price'], df_results['Lasso␣
 ↪Prediction']))
Lasso_feature_count = GSCV_lasso.best_params_.get('feature_selection__k',␣
 ↪GSCV_lasso.best_params_.get('feature_selection__percentile'))

# Calculate evaluation metrics for Random Forest
mae_RF = mean_absolute_error(df_results['price'], df_results['RF Prediction'])
rmse_RF = np.sqrt(mean_squared_error(df_results['price'], df_results['RF␣
 ↪Prediction']))
RF_feature_count = GSCV_rf.best_params_.get('feature_selection__k', GSCV_rf.
 ↪best_params_.get('feature_selection__percentile'))

# Display results
print(f"Lasso MAE: {mae_Lasso}, Lasso RMSE: {rmse_Lasso}, Lasso Feature Count:␣
 ↪{Lasso_feature_count}")
print(f"RF MAE: {mae_RF}, RF RMSE: {rmse_RF}, RF Feature Count:␣
 ↪{RF_feature_count}")
```

```
Lasso MAE: 1497.9728437973881, Lasso RMSE: 1912.9431838157361, Lasso Feature
Count: 17
RF MAE: 3184.566835199974, RF RMSE: 3990.3188214310735, RF Feature Count: 20
```

[167]:
```
import pickle

best_model = GSCV_lasso.best_estimator_

with open('BestModel_REG_LassoRegression_Transformers.pkl', 'wb') as f:
    pickle.dump(best_model, f)
    print("Model terbaik berhasil disimpan ke␣
 ↪'BestModel_REG_LassoRegression_Transformers.pkl'")
```

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

si-a-transformers-gbc-vs-svm-dylan dan Yohanes Paulus

October 25, 2024

[56]:
```
import pandas as pd
import numpy as np

DatasetUTS_Gasal2425_csv = pd.read_csv(r'C:
 ↪\Users\TUF\Downloads\MLUTS\MLUTS\DatasetUTS_Gasal2425.csv')

df_DatasetUTS_Gasal2425 = pd.DataFrame(data = DatasetUTS_Gasal2425_csv, index =␣
 ↪None)
df_DatasetUTS_Gasal2425
```

[56]:

	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
…	…	…	…	…	…	…
9995	341	83	no	no	8	1960
9996	21514	5	no	yes	11	91373
9997	1726	89	no	yes	5	73133
9998	44403	29	yes	yes	12	34606
9999	1440	84	no	no	49	18412

	citypartrange	numprevowners	made	isnewbuilt	hasstormprotector
0	3	8	2005	old	yes
1	6	8	2021	old	no
2	3	4	2003	new	no
3	8	3	2012	new	yes
4	4	2	2011	new	yes
…	…	…	…	…	…
9995	4	4	1993	new	yes

```
9996            1            1 1999       old             no
9997            7            6 2009       old            yes
9998            9            4 1990       old            yes
9999            6           10 1994       new             no
```

	basement	attic	garage	hasstorageroom	hasguestroom	price category
0	4313	9005	956	no	7	7559081.5 Luxury
1	2937	8852	135	yes	9	5574642.1 Middle
2	6326	4748	654	no	10	8696869.3 Luxury
3	632	5792	807	yes	5	5154055.2 Middle
4	5414	1172	716	yes	9	9652258.1 Luxury
…	…	…	…	…	…	…
9995	2366	4016	229	yes	5	35371.3 Basic
9996	2584	5266	787	no	3	2153602.9 Basic
9997	9311	1698	218	no	4	176425.9 Basic
9998	9061	1742	230	no	0	4448474.0 Basic
9999	8485	2024	278	yes	6	146708.4 Basic

[10000 rows x 18 columns]

[47]:
```
print("data null\n",
df_DatasetUTS_Gasal2425.isnull().sum()) print("\n")
print("data kosong \n",
df_DatasetUTS_Gasal2425.empty) print("\n")
print("data nan \n", df_DatasetUTS_Gasal2425.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
price             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
price             0
category          0
dtype: int64
```

[48]:
```
df_DatasetUTS_Gasal2425.describe()
```

[48]:

	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 10000.000001000.00000

       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
```

[49]:
```
import pandas as pd

df = pd.read_csv('C:\\Users\\TUF\\Downloads\\MLUTS\\MLUTS\\DatasetUTS_Gasal2425.
 ↪csv')

df
```

[49]:

	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
…	…	…	…	…	…	…
9995	341	83	no	no	8	1960
9996	21514	5	no	yes	11	91373
9997	1726	89	no	yes	5	73133
9998	44403	29	yes	yes	12	34606
9999	1440	84	no	no	49	18412

	citypartrange	numprevowners	made	isnewbuilt	hasstormprotector
0	3	8	2005	old	yes
1	6	8	2021	old	no

```
2                3            4 2003      new            no
3                8            3 2012      new           yes
4                4            2 2011      new           yes
…            …            … …      …            …
9995             4            4 1993      new           yes
9996             1            1 1999      old            no
9997             7            6 2009      old           yes
9998             9            4 1990      old           yes
9999             6           10 1994      new            no

     basement attic garage hasstorageroom hasguestroom        price
     category
0        4313 9005     956           no            7 7559081.5 Luxury
1        2937 8852     135          yes            9 5574642.1 Middle
2        6326 4748     654           no           10 8696869.3 Luxury
3         632  5792    807          yes            5 5154055.2 Middle
4        5414 1172     716          yes            9 9652258.1 Luxury
…       …    …    …         …          …       …     …
9995     2366 4016     229          yes            5   35371.3   Basic
9996     2584 5266     787           no            3 2153602.9   Basic
9997     9311 1698     218           no            4  176425.9   Basic
9998     9061 1742     230           no            0 4448474.0   Basic
9999     8485 2024     278          yes            6  146708.4   Basic
[10000 rows x 18 columns]
```

[50]:
```python
print(df_DatasetUTS_Gasal2425.head())
```

```
  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 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
```

[51]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

file_path = r'C:\Users\TUF\Downloads\MLUTS\MLUTS\DatasetUTS_Gasal2425.csv'
df_DatasetUTS_Gasal2425 = pd.read_csv(file_path)

categorical_columns = ['hasyard', 'haspool', 'isnewbuilt', 'hasstormprotector',␣
 ↪'hasstorageroom', 'category']

le = LabelEncoder()

for col in categorical_columns:
    if col in df_DatasetUTS_Gasal2425.columns:
        df_DatasetUTS_Gasal2425[col] = le.
 ↪fit_transform(df_DatasetUTS_Gasal2425[col])
    else:
        print(f"Kolom {col} tidak ditemukan dalam DataFrame.")
```

```
print(df_DatasetUTS_Gasal2425)
```

```
      squaremeters numberofrooms hasyard haspool floors citycode \
0           755233     0     1     63    9373
1           5571258    0     1     19    34457
2           86929100   1     0     11    98155
3           515223     0     0     61    9047
4           9647074    1     0     21    92029
...             ...          ...      ...      ...      ...       ...
9995          341 83    0     0     8     1960
9996          21514    5     0     1     11    91373
9997          1726     89    0     1     5     73133
9998          44403    29    1     1     12    34606
9999          1440     84    0     0     49    18412

      citypartrange numprevowners made isnewbuilt hasstormprotector \
0                3             8 2005          1                 1
1                6             8 2021          1                 0
2                3             4 2003          0                 0
3                8             3 2012          0                 1
```

```
     4               4             2 2011          0                1
     …               …             … …             …                …
9995             4      4 1993     0     1
9996             1      1 1999     1     0
9997             7      6 2009     1     1
9998             9      4 1990     1     1
9999             6      10 1994    0     0
         basement attic garage hasstorageroom hasguestroom    price \
     0         4313 9005     956             0            7 7559081.5
     1         2937 8852     135             1            9 5574642.1
2          6326 4748    654   0     10 8696869.3
3          632    5792  807   1     5 5154055.2
4          5414 1172    716   1     9 9652258.1
     …        …    …     …             …           …        …
9995     2366 4016     229   1     5     35371.3
9996     2584 5266     787   0     3 2153602.9
9997     9311 1698     218   0     4 176425.9
9998     9061 1742     230   0     0 4448474.0
9999     8485 2024     278   1     6 146708.4
          category
     0           1
     1           2
     2           1
     3           2
     4           1
     …         …
     9995        0
     9996        0
     9997        0
     9998        0
     9999        0

     [10000 rows x 18 columns]
```

[52]:
```
print(df_DatasetUTS_Gasal2425.head())
```

```
   squaremeters numberofrooms hasyard haspool floors citycode \
0         75523            3       0       1     63     9373
1         55712           58       0       1     19    34457
2         86929          100       1       0     11    98155
3         51522            3       0       0     61     9047
4         96470           74       1       0     21    92029
   citypartrange numprevowners made isnewbuilt hasstormprotector \
0 3 8 2005 1                                                    6

2 3 4 2003 0 03 8 3 2012 0                                      6

4 4 2 2011 0                                                    7

0 4313 9005 956 0 7 7559081.5                                   7

1 2937 8852 135 1 9 5574642.1                                   7


1            6   8 2021     1     0

    basement attic garage hasstorageroom hasguestroom    price category
2            6326 4748      654   0     10 8696869.3     1

3            632 5792  807   1     5 5154055.2      2

4      5414   1172     716              1            9  9652258.1         1
```

[53]:
```
from sklearn.model_selection import train_test_split

X = df_DatasetUTS_Gasal2425.drop(columns=['price'], axis=1)
y = df_DatasetUTS_Gasal2425['category']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=94)

print("bentuk X_train", X_train.shape)
print("bentuk X_test", X_test.shape)
print("bentuk y_train", y_train.shape)
print("bentuk y_test", y_test.shape)
print("y_train \n", y_train)
print("y_test \n",y_test)
```

```
bentuk X_train (7500, 17)
bentuk X_test (2500, 17)
bentuk y_train (7500,)
bentuk y_test (2500,)
y_train
 8494    0
5661    0
4117    2
```

```
929    1
4628 2
       ..
6806    0
1347    2
1381    2
6336    0
6482    0
Name: category, Length: 7500, dtype: int32
y_test
 7497    0
5188    1
5009    1
9434    0
2444    1
       ..
2434    1
3521    1
1955    1
2954    1
703     1
Name: category, Length: 2500, dtype: int32
```

[57]:
```
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

GBC = GradientBoostingClassifier(random_state=94)
SVM = SVC(C=1, gamma=0.01, random_state=94)

GBC.fit(X_train, y_train)
SVM.fit(X_train, y_train)
```

[57]:
```
SVC(C=1, gamma=0.01, random_state=94)
```

[58]:
```
import numpy as np
import pandas as pd
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

data = load_iris() X, y =
data.data, data.target
X_train, X_test, y_train, y_test = train_test_split(X, y,
 test_size=0.2, ␣ ↪random_state=94)
```

```
SVM = SVC()
SVM.fit(X_train, y_train)

GBC = GradientBoostingClassifier()
GBC.fit(X_train, y_train)

X_new = np.array([[3, 197, 30, 19, 0, 44.8, 0.567, 55]])

# Hapus pembuatan DataFrame dengan nama
fitur X_new_df = pd.DataFrame(X_new)
print("X_new yang akan diprediksi",
X_new_df.shape)

n_features_svm = SVM.n_features_in_
n_features_gbc = GBC.n_features_in_

print(f"Number of features in SVM: {n_features_svm}")
print(f"Number of features in GBC: {n_features_gbc}")

if X_new_df.shape[1] < n_features_svm:
    missing_features_svm = n_features_svm - X_new_df.shape[1]
    X_new_svm = pd.concat([X_new_df,
    pd.DataFrame(np.zeros((X_new_df.shape[0],␣
 ↪missing_features_svm)), columns=[f'feature{len(X_new_df.columns) +
 i + 1}'␣
 ↪for i in range(missing_features_svm)])], axis=1)
    print(f"Menyesuaikan input untuk SVM dengan
    menambahkan␣
 ↪{missing_features_svm} fitur.")
else:
    X_new_svm = X_new_df.iloc[:, :n_features_svm]

if X_new_df.shape[1] < n_features_gbc:
    missing_features_gbc = n_features_gbc - X_new_df.shape[1]
    X_new_gbc = pd.concat([X_new_df,
    pd.DataFrame(np.zeros((X_new_df.shape[0],␣
 ↪missing_features_gbc)), columns=[f'feature{len(X_new_df.columns) +
 i + 1}'␣
 ↪for i in range(missing_features_gbc)])], axis=1)
    print(f"Menyesuaikan input untuk Gradient Boosting
    Classifier dengan␣
 ↪menambahkan {missing_features_gbc}
fitur.") else:
```

```
    X_new_gbc = X_new_df.iloc[:, :n_features_gbc]

svm_predict = SVM.predict(X_new_svm)
```

```
print("Label Prediksi SVM", svm_predict)

gbc_predict = GBC.predict(X_new_gbc)
print("Label Prediksi GBC", gbc_predict)
```

```
X_new yang akan diprediksi (1, 8)
Number of features in SVM: 4
Number of features in GBC: 4
Label Prediksi SVM [2]
Label Prediksi GBC [2]
```

[59]:
```
import numpy as np import pandas as pd import
matplotlib.pyplot as plt from sklearn.datasets import
load_iris from sklearn.model_selection import
train_test_split, GridSearchCV from sklearn.ensemble
import GradientBoostingClassifier from sklearn.svm import
SVC
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, SelectPercentile,
f_classif from sklearn.metrics import classification_report,
roc_auc_score

# Load dataset
data =
load_iris()
X, y = data.data, data.target

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y,
 test_size=0.2,␣ ↪random_state=94)

# Standardize data
scaler_standard = StandardScaler()
X_train_standard = scaler_standard.fit_transform(X_train)
X_test_standard = scaler_standard.transform(X_test)

# Feature selection using SelectKBest kbest =
SelectKBest(score_func=f_classif, k=min(10,
X_train_standard.shape[1]))␣ ↪# Ensure k does not exceed number of
features
X_train_kbest = kbest.fit_transform(X_train_standard, y_train)
```

```
X_test_kbest = kbest.transform(X_test_standard)

# Feature selection using SelectPercentile
percentile = SelectPercentile(score_func=f_classif, percentile=50)
X_train_percentile = percentile.fit_transform(X_train_standard,
y_train)
X_test_percentile = percentile.transform(X_test_standard)

# Grid search for Gradient Boosting Classifier
```

```
param_grid_gbc = {
    'n_estimators':  100, 200],
    'learning_rate':  0.01, 0.1, 1],
    'max_depth':  3, 5, 7]
}
gbc_grid = GridSearchCV(GradientBoostingClassifier(random_state=94),␣
 ↪param_grid_gbc, cv=5)
gbc_grid.fit(X_train_kbest, y_train)

# Grid search for Support Vector Machine
param_grid_svm = {
    'C':  0.01, 0.1, 1, 10],
    'kernel':  'linear', 'rbf', 'poly'],
    'gamma':  'scale', 'auto']
}
svm_grid = GridSearchCV(SVC(probability=True, random_state=94), param_grid_svm,␣
 ↪cv=5)
svm_grid.fit(X_train_kbest, y_train)

# Predictions
gbc_pred = gbc_grid.predict(X_test_kbest)
svm_pred = svm_grid.predict(X_test_kbest)

# Display classification reports
print("Gradient Boosting Classifier Report:")
print(classification_report(y_test, gbc_pred))
print("Support Vector Machine Report:")
print(classification_report(y_test, svm_pred))

# Calculate AUC
gbc_auc = roc_auc_score(y_test, gbc_grid.predict_proba(X_test_kbest),␣
 ↪multi_class='ovr')
svm_auc = roc_auc_score(y_test, svm_grid.predict_proba(X_test_kbest),␣
 ↪multi_class='ovr')

print("Gradient Boosting Classifier AUC:", gbc_auc)
print("Support Vector Machine AUC:", svm_auc)
```

```
Gradient Boosting Classifier Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      0.92      0.96        12
           2       0.89      1.00      0.94         8

    accuracy                           0.97        30
```

```
       macro avg      0.96      0.97      0.97        30
    weighted avg      0.97      0.97      0.97        30

     Support Vector Machine Report:
                  precision    recall  f1-score   support

0       1.00      1.00      1.00        10

1       1.00      0.92      0.96        12

2       0.89      1.00      0.94         8

        accuracy                           0.97        30
       macro avg      0.96      0.97      0.97        30
    weighted avg      0.97      0.97      0.97        30

     Gradient Boosting Classifier AUC: 0.9965628507295173
     Support Vector Machine AUC: 0.9931257014590348
```

[60]:
```
import numpy as np
import pandas as pd

# Define your input array (replace these values with your actual
features)
X_new = np.array([[3, 197, 30, 19, 0, 44.8, 0.567, 55, 0, 0, 0, 0, 0,
0, 0, 0,␣
 ↪0]]) # Example input with fewer than 17 features

# Ensure X_new has 17 features
expected_features = 17
current_shape = X_new.shape[1]

if current_shape < expected_features:
    # Add zeros to match the expected number of features
    X_new = np.hstack([X_new, np.zeros((X_new.shape[0],
    expected_features -␣
 ↪current_shape))]) print(f"Menyesuaikan input dengan
    menambahkan {expected_features -␣
 ↪current_shape} fitur.") elif
current_shape >
expected_features:
    # Truncate to match the expected number of features
    X_new = X_new[:, :expected_features]
```

```
    print(f"Truncating input to keep only the first
{expected_features}␣ ↪features.") print("X_new yang akan
diprediksi", X_new.shape)

# Get the expected number of features from the models
n_features_svm = SVM.n_features_in_
n_features_gbc = GBC.n_features_in_

print(f"Expected number of features for SVM:
{n_features_svm}") print(f"Expected number of features for
GBC: {n_features_gbc}")
```

```
# Adjust input for SVM
X_new_svm = X_new[:, :n_features_svm]  # Assuming this is correct based on ␣
 ↪training features

# Adjust input for Gradient Boosting Classifier
X_new_gbc = X_new[:, :n_features_gbc]  # Assuming this is correct based on ␣
 ↪training features

# Make predictions
svm_predict = SVM.predict(X_new_svm)
print("Label Prediksi SVM:", svm_predict)

gbc_predict = GBC.predict(X_new_gbc)
print("Label Prediksi GBC:", gbc_predict)
```

```
X_new yang akan diprediksi (1, 17)
Expected number of features for SVM: 4
Expected number of features for GBC: 4
Label Prediksi SVM: [2]
Label Prediksi GBC: [2]
```

[67]:
```
import numpy as np import pandas as pd import
matplotlib.pyplot as plt from sklearn.pipeline import
Pipeline from sklearn.model_selection import
train_test_split, GridSearchCV,␣
 ↪StratifiedKFold from sklearn.ensemble import
GradientBoostingClassifier from sklearn.svm
import SVC
from sklearn.preprocessing import StandardScaler from
sklearn.feature_selection import SelectKBest, f_classif
from sklearn.metrics import classification_report,
confusion_matrix,␣ ↪ConfusionMatrixDisplay
```

```
# Define your feature names here feature_names = ['feature1',
'feature2', 'feature3', 'feature4'] # Replace␣ ↪with your actual
feature names

# Assuming X and y are defined elsewhere in your code
# X_train, X_test, y_train, y_test = train_test_split(X, y,
 test_size=0.2,␣ ↪random_state=94)

# Define k based on the number of features available
k = min(10, X_train.shape[1]) # Ensure k does not exceed the number
of features
```

```
pipeline_gbc = Pipeline([
    ('scaler', StandardScaler()),
    ('feature_selection', SelectKBest(score_func=f_classif, k=k)),
    ('classifier', GradientBoostingClassifier(random_state=94))
])

pipeline_svm = Pipeline([
    ('scaler', StandardScaler()),
    ('feature_selection', SelectKBest(score_func=f_classif, k=k)),
    ('classifier', SVC(probability=True, random_state=94))  # Added␣
 ↪probability=True for AUC calculation
])

param_grid_gbc = {
    'classifier__n_estimators': [100, 200],
    'classifier__max_depth': [3, 5],
    'classifier__learning_rate': [0.01, 0.1]
}

param_grid_svm = {
    'classifier__C': [0.01, 0.1, 1, 10],
    'classifier__kernel': ['linear', 'rbf'],
    'classifier__gamma': ['scale', 'auto']
}

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=94)

grid_search_gbc = GridSearchCV(pipeline_gbc, param_grid_gbc, cv=skf,␣
 ↪scoring='accuracy')
grid_search_gbc.fit(X_train, y_train)

grid_search_svm = GridSearchCV(pipeline_svm, param_grid_svm, cv=skf,␣
 ↪scoring='accuracy')
grid_search_svm.fit(X_train, y_train)

gbc_pred = grid_search_gbc.predict(X_test)
print("Gradient Boosting Classifier Classification Report:")
print(classification_report(y_test, gbc_pred))

conf_matrix_gbc = confusion_matrix(y_test, gbc_pred)
ConfusionMatrixDisplay(confusion_matrix=conf_matrix_gbc).plot(cmap='Blues')
plt.title('Confusion Matrix - Gradient Boosting Classifier')
plt.savefig('confusion_matrix_gbc.png')

svm_pred = grid_search_svm.predict(X_test)
print("Support Vector Machine Classification Report:")
```

```
print(classification_report(y_test, svm_pred))

conf_matrix_svm = confusion_matrix(y_test, svm_pred)
ConfusionMatrixDisplay(confusion_matrix=conf_matrix_svm).plot(cmap='B
lues')
plt.title('Confusion Matrix - Support Vector Machine')
plt.savefig('confusion_matrix_svm.png')

# Get relevant features for Gradient Boosting Classifier
feature_mask_gbc = grid_search_gbc.best_estimator_.
 ↪named_steps['feature_selection'].get_support()
relevant_features_gbc =
np.array(feature_names)[feature_mask_gbc] print("Fitur
Relevan yang Dipilih oleh Gradient Boosting Classifier:", ␣
↪relevant_features_gbc)

# Get relevant features for Support Vector Machine
feature_mask_svm = grid_search_svm.best_estimator_.
 ↪named_steps['feature_selection'].get_support()
relevant_features_svm =
np.array(feature_names)[feature_mask_svm] print("Fitur
Relevan yang Dipilih oleh Support Vector Machine:", ␣
↪relevant_features_svm)
```

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

           0       1.00      1.00      1.00        10
           1       1.00      0.92      0.96        12
           2       0.89      1.00      0.94         8

    accuracy                           0.97        30
   macro avg       0.96      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30

Support Vector Machine Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      0.92      0.96        12
           2       0.89      1.00      0.94         8

    accuracy                           0.97        30
   macro avg       0.96      0.97      0.97        30
```

```
weighted avg     0.97      0.97      0.97       30

Fitur Relevan yang Dipilih oleh Gradient Boosting Classifier:
['feature1'
'feature2' 'feature3' 'feature4']
Fitur Relevan yang Dipilih oleh Support Vector Machine: ['feature1'
'feature2' 'feature3' 'feature4']
```

```
[68]: # Assuming you have already performed grid search and predictions
for GBC and␣
    ↪SVM gbc_best_accuracy =
grid_search_gbc.best_score_ gbc_best_f1 =
classification_report(y_test, gbc_pred,␣
    ↪output_dict=True)['weighted avg']['f1-score']
gbc_best_precision = classification_report(y_test,
gbc_pred,␣
    ↪output_dict=True)['weighted avg']['precision']
gbc_best_recall = classification_report(y_test,
gbc_pred,␣
    ↪output_dict=True)['weighted avg']['recall']
gbc_best_n_features = np.sum(feature_mask_gbc)

print("Gradient Boosting Classifier Best
Metrics:") print(f"Accuracy:
{gbc_best_accuracy:.4f}") print(f"Precision:
{gbc_best_precision:.4f}") print(f"Recall:
{gbc_best_recall:.4f}") print(f"F1 Score:
{gbc_best_f1:.4f}") print(f"Number of
Features: {gbc_best_n_features}")

svm_best_accuracy = grid_search_svm.best_score_
svm_best_f1 = classification_report(y_test,
svm_pred,␣
    ↪output_dict=True)['weighted avg']['f1-score']
svm_best_precision = classification_report(y_test,
svm_pred,␣
    ↪output_dict=True)['weighted avg']['precision']
svm_best_recall = classification_report(y_test,
svm_pred,␣
    ↪output_dict=True)['weighted avg']['recall']
svm_best_n_features = np.sum(feature_mask_svm)

print("\nSupport Vector Machine Best
Metrics:") print(f"Accuracy:
{svm_best_accuracy:.4f}") print(f"Precision:
{svm_best_precision:.4f}") print(f"Recall:
```

```
{svm_best_recall:.4f}") print(f"F1 Score:
{svm_best_f1:.4f}") print(f"Number of
Features: {svm_best_n_features}")

if gbc_best_f1 > svm_best_f1:
    best_model = "Gradient Boosting Classifier"
    best_metrics = {
        "Accuracy": gbc_best_accuracy,
        "Precision": gbc_best_precision,
        "Recall": gbc_best_recall,
        "F1 Score": gbc_best_f1,
        "Number of Features": gbc_best_n_features
    }
else:
    best_model = "Support Vector Machine"
    best_metrics = {
```

```
        "Accuracy": svm_best_accuracy,
        "Precision": svm_best_precision,
        "Recall": svm_best_recall,
        "F1 Score": svm_best_f1,
        "Number of Features": svm_best_n_features
    }

print(f"\nModel Terbaik: {best_model}")
print("Metrik Terbaik:")
for metric, value in best_metrics.items():
    print(f"{metric}: {value:.4f}")
```

```
Gradient Boosting Classifier Best Metrics:
Accuracy: 0.9500
Precision: 0.9704
Recall: 0.9667
F1 Score: 0.9669
Number of Features: 4

Support Vector Machine Best Metrics:
Accuracy: 0.9667
Precision: 0.9704
Recall: 0.9667
F1 Score: 0.9669
Number of Features: 4

Model Terbaik: Support Vector Machine
Metrik Terbaik:
Accuracy: 0.9667
```

```
Precision: 0.9704
Recall: 0.9667
F1 Score: 0.9669
Number of Features: 4.0000
```

[70]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report
import joblib

# Load dataset
df_DatasetUTS_Gasal2425 = pd.read_csv('DatasetUTS_Gasal2425.csv')

# Inisialisasi LabelEncoder
label_encoder = LabelEncoder()
# Mengubah kolom 'category' menjadi numerik
df_DatasetUTS_Gasal2425['category'] = label_encoder.
 ↪fit_transform(df_DatasetUTS_Gasal2425['category'])

# Simpan kelas dari kolom 'category' untuk digunakan di laporan
klasifikasi category_classes = label_encoder.classes_

# Pastikan semua kolom kategorikal lain diubah menjadi
numerik for col in df_DatasetUTS_Gasal2425.columns:
    if df_DatasetUTS_Gasal2425[col].dtype == 'object':
        df_DatasetUTS_Gasal2425[col] = label_encoder.
 ↪fit_transform(df_DatasetUTS_Gasal2425[col])

# Memastikan tidak ada nilai NaN dalam dataset
df_DatasetUTS_Gasal2425 = df_DatasetUTS_Gasal2425.fillna(0)

# Memisahkan fitur dan target
X = df_DatasetUTS_Gasal2425.drop(['category', 'price'],
axis=1) y = df_DatasetUTS_Gasal2425['category']

# Membagi dataset menjadi training dan testing
X_train, X_test, y_train, y_test = train_test_split(X, y,
 test_size=0.2, ␣ ↪random_state=94)

# Melatih model Gradient Boosting
Classifier gbc =
GradientBoostingClassifier(random_state=94
) gbc.fit(X_train, y_train)
```

```
# Melakukan prediksi pada data testing dengan GBC
gbc_pred = gbc.predict(X_test)

# Menampilkan laporan klasifikasi untuk GBC
print("Gradient Boosting Classifier Best
Metrics:")
print(classification_report(y_test, gbc_pred,
target_names=category_classes))

# Melatih model Support Vector Machine
svm = SVC(random_state=94)
svm.fit(X_train, y_train)

# Melakukan prediksi pada data testing dengan SVM
svm_pred = svm.predict(X_test)

# Menampilkan laporan klasifikasi untuk SVM
print("\nSupport Vector Machine Best Metrics:")
print(classification_report(y_test, svm_pred,
target_names=category_classes)) # Menyimpan model yang terbaik
if classification_report(y_test, gbc_pred,
 output_dict=True)['weighted␣ ↪avg']['f1-score'] >
 classification_report(y_test, svm_pred,␣
 ↪output_dict=True)['weighted avg']['f1-score']: best_model =
 gbc
    best_model_name = "Gradient Boosting Classifier"
else:
    best_model = svm
    best_model_name = "Support Vector Machine"

joblib.dump(best_model,
f'BestModel_GBC_SVM_Transformers.pkl') print(f"\nModel
Terbaik: BestModel_GBC_SVM_Transformers.pkl") Gradient
Boosting Classifier Best Metrics:
              precision    recall  f1-score   support

       Basic       1.00      1.00      1.00       867
      Luxury       1.00      1.00      1.00       602
      Middle       1.00      1.00      1.00       531

    accuracy                           1.00      2000
   macro avg       1.00      1.00      1.00      2000
weighted avg       1.00      1.00      1.00      2000
Support Vector Machine Best Metrics:
              precision    recall  f1-score   support
```

```
      Basic      0.86      0.97      0.91       867
     Luxury      1.00      1.00      1.00       602
     Middle      0.94      0.73      0.82       531

   accuracy                           0.92      2000
  macro avg      0.93      0.90      0.91      2000
weighted avg     0.92      0.92      0.91      2000
Model Terbaik: BestModel_GBC_SVM_Transformers.pkl
```