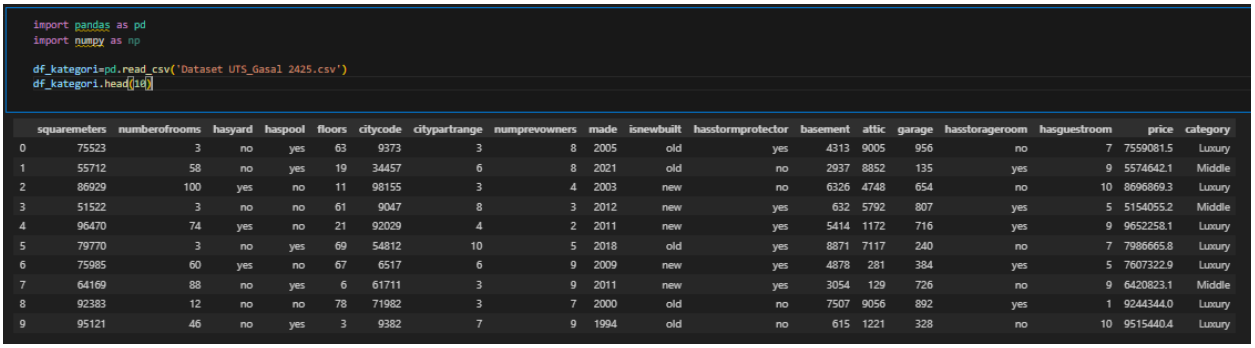
Klasifikasi dengan algoritma berbasis linear

```
import pandas as pd

import numpy as np


df_kategori=pd.read_csv('Dataset UTS_Gasal 2425.csv')

df_kategori.head(10)
```



```
import pandas as pd
import numpy as np

df_kategori=pd.read_csv('Dataset UTS_Gasal 2425.csv')
df_kategori.head(10)
```

	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	8	3	2012	new	yes	632	5792	807	yes	5	5154055.2	Middle
4	96470	74	yes	no	21	92029	4	2	2011	new	yes	5414	1172	716	yes	9	9652258.1	Luxury
5	79770	3	no	yes	69	54812	10	5	2018	old	yes	8871	7117	240	no	7	7986665.8	Luxury
6	75985	60	yes	no	67	6517	6	9	2009	new	yes	4878	281	384	yes	5	7607322.9	Luxury
7	64169	88	no	yes	6	61711	3	9	2011	new	yes	3054	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	7	9	1994	old	no	615	1221	328	no	10	9515440.4	Luxury

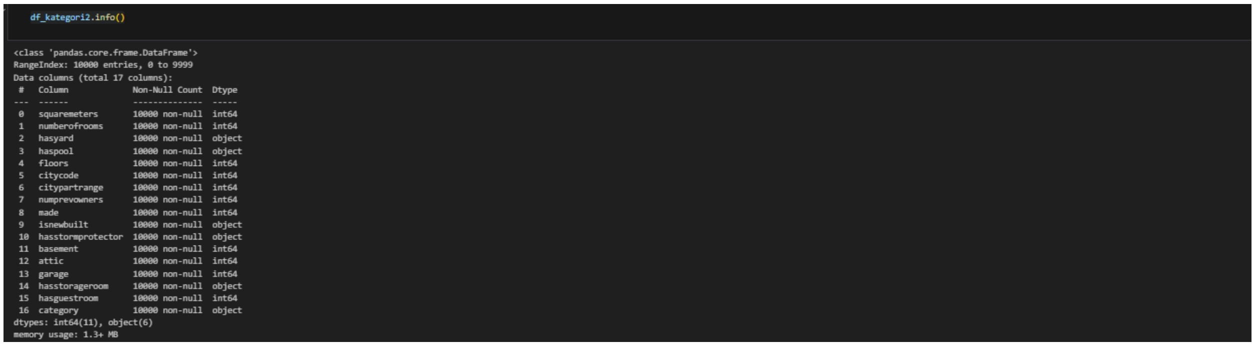
df_kategori2 = df_kategori.drop(['price'], axis=1)

df_kategori2.head()

```
df_kategori2 = df_kategori.drop(['price'], axis=1)
df_kategori2.head()
```

	squaremeters	numberofrooms	hasyard	haspool	floors	citycode	citypartrange	numprevowners	made	isnewbuilt	hasstormprotector	basement	attic	garage	hasstorageroom	hasguestroom	category
0	75523	3	no	yes	63	9373	3	8	2005	old	yes	4313	9005	956	no	7	Luxury
1	55712	58	no	yes	19	34457	6	8	2021	old	no	2937	8852	135	yes	9	Middle
2	86929	100	yes	no	11	98155	3	4	2003	new	no	6326	4748	654	no	10	Luxury
3	51522	3	no	no	61	9047	8	3	2012	new	yes	632	5792	807	yes	5	Middle
4	96470	74	yes	no	21	92029	4	2	2011	new	yes	5414	1172	716	yes	9	Luxury

df_kategori2.info()



```
df_kategori2.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  category           10000 non-null  object
dtypes: int64(11), object(6)
memory usage: 1.3+ MB
```

df_kategori2.describe()

```
df_kategori2.describe()
```

	squaremeters	numberofrooms	floors	citycode	citypartrange	numprevowners	made	basement	attic	garage	hasguestroom
count	10000.00000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.00000	10000.00000	10000.00000
mean	49870.13120	50.358400	50.276300	50225.486100	5.510100	5.521700	2005.48850	5033.103900	5028.01060	553.12120	4.99460
std	28774.37535	28.816696	28.889171	29006.675799	2.872024	2.856667	9.30809	2876.729545	2894.33221	262.05017	3.17641
min	89.00000	1.000000	1.000000	3.000000	1.000000	1.000000	1990.00000	0.000000	1.00000	100.00000	0.00000
25%	25098.50000	25.000000	25.000000	24693.750000	3.000000	3.000000	1997.00000	2559.750000	2512.00000	327.75000	2.00000
50%	50105.50000	50.000000	50.000000	50693.000000	5.000000	5.000000	2005.50000	5092.500000	5045.00000	554.00000	5.00000
75%	74609.75000	75.000000	76.000000	75683.250000	8.000000	8.000000	2014.00000	7511.250000	7540.50000	777.25000	8.00000
max	99999.00000	100.000000	100.000000	99953.000000	10.000000	10.000000	2021.00000	10000.000000	10000.00000	1000.00000	10.00000

print("data NULL \n", df_kategori2.isnull().sum())

print("data KOSONG \n", df_kategori2.empty)

print("data NaN \n", df_kategori2.isna().sum())

```
print("data NULL \n", df_kategori2.isnull().sum())
print("data  KOSONG \n", df_kategori2.empty)
print("data NaN \n", df_kategori2.isna().sum())
```

```
data NULL
 squaremeters        0
numberofrooms       0
hasyard             0
haspool             0
floors              0
citycode            0
citypartrange       0
numprevowners       0
made                0
isnewbuilt          0
hasstormprotector   0
basement            0
attic               0
garage              0
hasstorageroom      0
hasguestroom        0
category            0
dtype: int64
data  KOSONG
 False
data NaN
 squaremeters        0
numberofrooms       0
hasyard             0
...
hasstorageroom      0
hasguestroom        0
category            0
dtype: int64
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

print("Sebelum Pengecekan data duplikat, ", df_kategori2.shape)

df_kategori3=df_kategori2.drop_duplicates(keep='last')

print("Setelah Pengecekan data duplikat, ", df_kategori3.shape)

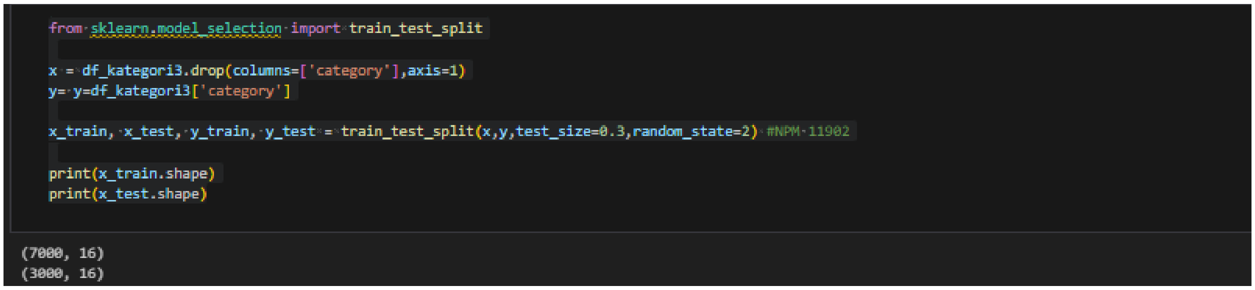
```
#cek duplikat
print("Sebelum Pengecekan data duplikat, ", df_kategori2.shape)
df_kategori3=df_kategori2.drop_duplicates(keep='last')
print("Setelah Pengecekan data duplikat, ", df_kategori3.shape)
```

```
Sebelum Pengecekan data duplikat,  (10000, 17)
Setelah Pengecekan data duplikat,  (10000, 17)
```

from sklearn.model_selection import train_test_split

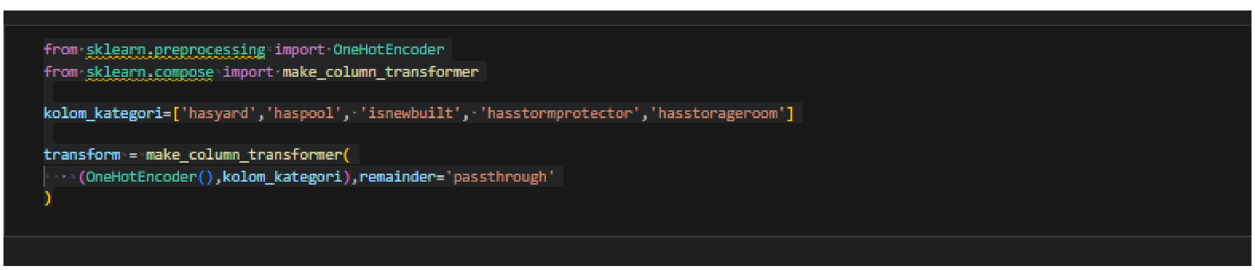
x = df_kategori3.drop(columns=['category'],axis=1)

```
y= y=df_kategori3['category']

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3,random_state=2) #NPM 11902

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



```
from sklearn.model_selection import train_test_split

x = df_kategori3.drop(columns=['category'],axis=1)
y= y=df_kategori3['category']

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3,random_state=2) #NPM 11902

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

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

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer

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

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



```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer

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

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

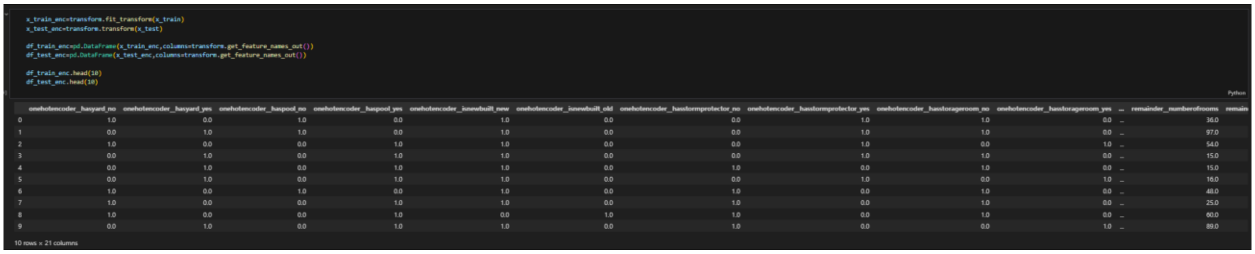
```
x_train_enc=transform.fit_transform(x_train)
x_test_enc=transform.transform(x_test)
```

df_train_enc=pd.DataFrame(x_train_enc,columns=transform.get_feature_names_out())

df_test_enc=pd.DataFrame(x_test_enc,columns=transform.get_feature_names_out())

df_train_enc.head(10)

df_test_enc.head(10)



```
x_train_enc=transform.fit_transform(x_train)
x_test_enc=transform.transform(x_test)

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

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

Python

	onehotencoder__hasyard_no	onehotencoder__hasyard_yes	onehotencoder__haspool_no	onehotencoder__haspool_yes	onehotencoder__isnewbuilt_new	onehotencoder__isnewbuilt_old	onehotencoder__hasstormprotector_no	onehotencoder__hasstormprotector_yes	onehotencoder__hasstorageroom_no	onehotencoder__hasstorageroom_yes	...	remainder__numberofrooms	remain
0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	...	36.0	
1	0.0	1.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	...	97.0	
2	1.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	1.0	...	54.0	
3	0.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0	0.0	...	15.0	
4	0.0	1.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	...	15.0	
5	0.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	1.0	...	16.0	
6	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	...	48.0	
7	1.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	...	25.0	
8	1.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	1.0	0.0	...	60.0	
9	0.0	1.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	1.0	...	89.0	

10 rows × 21 columns

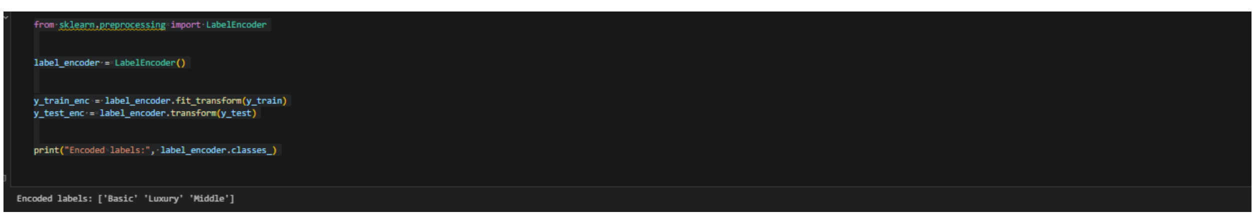
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()

y_train_enc = label_encoder.fit_transform(y_train)

y_test_enc = label_encoder.transform(y_test)

print("Encoded labels:", label_encoder.classes_)



```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()

y_train_enc = label_encoder.fit_transform(y_train)
y_test_enc = label_encoder.transform(y_test)

print("Encoded labels:", label_encoder.classes_)
```

```
Encoded labels: ['Basic' 'Luxury' 'Middle']
```

from sklearn.preprocessing import MinMaxScaler, StandardScaler

from sklearn.feature_selection import SelectPercentile, SelectKBest

from sklearn.svm import SVC

```
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.pipeline import Pipeline
import numpy as np

pipe_svm = Pipeline(steps=[
    ('scale', MinMaxScaler()),
    ('feat_select', SelectKBest(k=16)),
    ('clf', SVC(class_weight='balanced', kernel='linear'))
])

params_grid_svm = [
    {
        'scale': [MinMaxScaler()],
        'clf__kernel': ['linear'],
        'clf__C': [0.1, 1]
    },
    {
        'scale': [StandardScaler()],
        'clf__kernel': ['linear'],
        'clf__C': [0.1, 1]
    }
]

estimator_svm = Pipeline(pipe_svm)

SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=2)
```

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

GSCV_SVM.fit(x_train_enc, y_train_enc)

print("GSCV Training finished")

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.feature_selection import SelectPercentile, SelectKBest
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.pipeline import Pipeline
import numpy as np

pipe_svm = Pipeline(steps=[
    ('scale', MinMaxScaler()),
    ('feat_select', SelectKBest(k=16)),
    ('clf', SVC(class_weight='balanced', kernel='linear'))
])

params_grid_svm = [
    {
        'scale': [MinMaxScaler()],
        'clf__kernel': ['linear'],
        'clf__C': [0.1, 1]
    },
    {
        'scale': [StandardScaler()],
        'clf__kernel': ['linear'],
        'clf__C': [0.1, 1]
    }
]

estimator_svm = Pipeline(pipe_svm)

SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=2)

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

GSCV_SVM.fit(x_train_enc, y_train_enc)
print("GSCV Training finished")
```

```
GSCV Training finished
```

from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay

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

print("Test Score: {}".format(GSCV_SVM.best_estimator_.score(x_test_enc, y_test_enc)))

print("Best model:", GSCV_SVM.best_estimator_)

mask = GSCV_SVM.best_estimator_.named_steps['feat_select'].get_support()

print("Best features:", df_train_enc.columns[mask])

SVM_pred = GSCV_SVM.predict(x_test_enc)

import matplotlib.pyplot as plt

cm = confusion_matrix(y_test_enc, SVM_pred, labels=GSCV_SVM.classes_)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=GSCV_SVM.classes_)

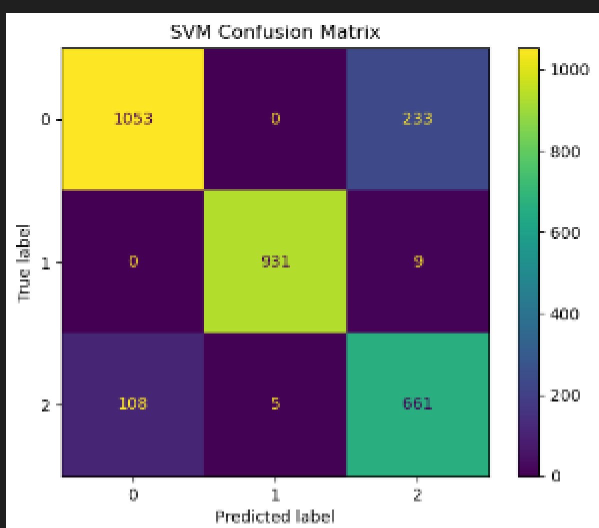
disp.plot()

plt.title("SVM Confusion Matrix")

plt.show()

print("Classification report SVM:\n", classification_report(y_test_enc, SVM_pred))

```
CV Score : 0.8768571428571429
Test Score: 0.8816666666666667
Best model: Pipeline(steps=[('scale', StandardScaler()), ('feat_select', SelectKBest(k=16)),
                ('clf', SVC(C=1, class_weight='balanced', kernel='linear'))])
Best features: Index(['onehotencoder__hasyard_no', 'onehotencoder__hasyard_yes',
       'onehotencoder__haspool_no', 'onehotencoder__haspool_yes',
       'onehotencoder__isnewbuilt_new', 'onehotencoder__isnewbuilt_old',
       'onehotencoder__hasstormprotector_no',
       'onehotencoder__hasstormprotector_yes',
       'onehotencoder__hasstorageroom_no', 'onehotencoder__hasstorageroom_yes',
       'remainder__squaremeters', 'remainder__numberofrooms',
       'remainder__citypartrange', 'remainder__made', 'remainder__basement',
       'remainder__garage'],
      dtype='object')
```



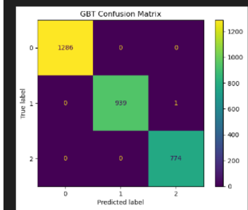

```
Classification report SVM:
              precision    recall  f1-score   support

           0       0.91      0.82      0.86      1286
           1       0.99      0.99      0.99       940
           2       0.73      0.85      0.79       774

    accuracy                           0.88      3000
   macro avg       0.88      0.89      0.88      3000
weighted avg       0.89      0.88      0.88      3000
```

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.tree import DecisionTreeClassifier

pipe_GBT = Pipeline(steps=[
    ('feat_select',SelectKBest(k=16)),
    ('clf',GradientBoostingClassifier(random_state=2))])

params_grid_GBT = [
    {
        'clf__max_depth': [4],
        'clf__n_estimators': [100, 150],
        'clf__learning_rate': [0.01, 0.1, 1]
    },
    {
        'feat_select': [SelectPercentile(percentile=100)],
        'clf__max_depth': [4],
        'clf__n_estimators': [100, 150],
        'clf__learning_rate': [0.01, 0.1, 1]
    }
]

GSCV_GBT = GridSearchCV(pipe_GBT,params_grid_GBT, cv=StratifiedKFold(n_splits=5))
GSCV_GBT.fit(x_train_enc, y_train_enc)
print("GSCV Finished")
```

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.tree import DecisionTreeClassifier

pipe_GBT = Pipeline(steps=[
    ('feat_select',SelectKBest(k=16)),
    ('clf',GradientBoostingClassifier(random_state=2))])

params_grid_GBT = [
    {
        'clf__max_depth': [4],
        'clf__n_estimators': [100, 150],
        'clf__learning_rate': [0.01, 0.1, 1]
    },
    {
        'feat_select': [SelectPercentile(percentile=100)],
        'clf__max_depth': [4],
        'clf__n_estimators': [100, 150],
        'clf__learning_rate': [0.01, 0.1, 1]
    }
]

GSCV_GBT = GridSearchCV(pipe_GBT,params_grid_GBT, cv=StratifiedKFold(n_splits=5))
GSCV_GBT.fit(x_train_enc, y_train_enc)
print("GSCV Finished")

GSCV Finished
```

```
print("CV Score: {}".format(GSCV_GBT.best_score_))
print("Test Score: {}".format(GSCV_GBT.best_estimator_.score(x_test_enc, y_test_enc)))
print("Best model:", GSCV_GBT.best_estimator_)

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

GBT_pred = GSCV_GBT.predict(x_test_enc)

import matplotlib.pyplot as plt

cm = confusion_matrix(y_test_enc, GBT_pred, labels=GSCV_GBT.classes_)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=GSCV_GBT.classes_)
disp.plot()

plt.title("GBT Confusion Matrix")
plt.show()

print("Classification report GBT: \n",  classification_report(y_test_enc, GBT_pred))
```

```
CV Score: 0.9991428571428571
Test Score: 0.9996666666666667
Best model: Pipeline(steps=[('feat_select', SelectKBest(k=16)),
                ('clf',
                 GradientBoostingClassifier(learning_rate=0.01, max_depth=4,
                                            n_estimators=150,
                                            random_state=2))])
Best features: Index(['onehotencoder__hasyard_no', 'onehotencoder__hasyard_yes',
       'onehotencoder__haspool_no', 'onehotencoder__haspool_yes',
       'onehotencoder__isnewbuilt_new', 'onehotencoder__isnewbuilt_old',
       'onehotencoder__hasstormprotector_no',
       'onehotencoder__hasstormprotector_yes',
       'onehotencoder__hasstorageroom_no', 'onehotencoder__hasstorageroom_yes',
       'remainder__squaremeters', 'remainder__numberofrooms',
       'remainder__citypartrange', 'remainder__made', 'remainder__basement',
       'remainder__garage'],
      dtype='object')
```




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

           0       1.00      1.00      1.00      1286
           1       1.00      1.00      1.00       940
           2       1.00      1.00      1.00       774

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

import pickle

with open('BestModel_CLF_GBC_Keras.pkl','wb') as r:

    pickle.dump((GSCV_GBT), r)

print("Model GBT berhasil disimpan")

```
import pickle

with open('BestModel_CLF_GBC_Keras.pkl','wb') as r:
    pickle.dump((GSCV_GBT), r)

print("Model GBT berhasil disimpan")
```

```
Model GBT berhasil disimpan
```

Regresi algortima regressor berbasis model

```
import pandas as pd

import numpy as np


df_price = pd.read_csv(r'D:\Projek UTS Gasal 20242025-20241010\Dataset UTS_Gasal 2425.csv')

df_price.head(10)
```

```
import pandas as pd
import numpy as np

df_price = pd.read_csv(r'D:\Projek UTS Gasal 20242025-20241010\Dataset UTS_Gasal 2425.csv')
df_price.head(10)
```

[1] 3.0s

	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	8	3	2012	new	yes	632	5792	807	yes	5	5154055.2	Middle
4	96470	74	yes	no	21	92029	4	2	2011	new	yes	5414	1172	716	yes	9	9652258.1	Luxury
5	79770	3	no	yes	69	54812	10	5	2018	old	yes	8871	7117	240	no	7	7986665.8	Luxury
6	75985	60	yes	no	67	6517	6	9	2009	new	yes	4878	281	384	yes	5	7607322.9	Luxury
7	64169	88	no	yes	6	61711	3	9	2011	new	yes	3054	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	7	9	1994	old	no	615	1221	328	no	10	9515440.4	Luxury

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

df_price2.head()
```

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

✓ 0.0s

	squaremeters	numberofrooms	hasyard	haspool	floors	citycode	citypartrange	numprevowners	made	isnewbuilt	hasstormprotector	basement	attic	garage	hasstorageroom	hasguestroom	price
0	75523	3	no	yes	63	9373	3	8	2005	old	yes	4313	9005	956	no	7	7559081.5
1	55712	58	no	yes	19	34457	6	8	2021	old	no	2937	8852	135	yes	9	5574642.1
2	86929	100	yes	no	11	98155	3	4	2003	new	no	6326	4748	654	no	10	8696869.3
3	51522	3	no	no	61	9047	8	3	2012	new	yes	632	5792	807	yes	5	5154055.2
4	96470	74	yes	no	21	92029	4	2	2011	new	yes	5414	1172	716	yes	9	9652258.1

```
df_price2.info()
```

```
df_price2.info()
```

✓ 0.0s

```
<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
```

df_price2.describe()

```
df_price2.describe()
✓ 0.0s
```

	squaremeters	numberofrooms	floors	citycode	citypartrange	numprevowners	made	basement	attic	garage	hasguestroom	price
count	10000.00000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.00000	10000.00000	10000.00000	1.000000e+04
mean	49870.13120	50.358400	50.276300	50225.486100	5.510100	5.521700	2005.48850	5033.103900	5028.01060	553.12120	4.99460	4.993448e+06
std	28774.37535	28.816696	28.889171	29006.675799	2.872024	2.856667	9.30809	2876.729545	2894.33221	262.05017	3.17641	2.877424e+06
min	89.00000	1.000000	1.000000	3.000000	1.000000	1.000000	1990.00000	0.000000	1.00000	100.00000	0.00000	1.031350e+04
25%	25098.50000	25.000000	25.000000	24693.750000	3.000000	3.000000	1997.00000	2559.750000	2512.00000	327.75000	2.00000	2.516402e+06
50%	50105.50000	50.000000	50.000000	50693.000000	5.000000	5.000000	2005.50000	5092.500000	5045.00000	554.00000	5.00000	5.016180e+06
75%	74609.75000	75.000000	76.000000	75683.250000	8.000000	8.000000	2014.00000	7511.250000	7540.50000	777.25000	8.00000	7.469092e+06
max	99999.00000	100.000000	100.000000	99953.000000	10.000000	10.000000	2021.00000	10000.000000	10000.00000	1000.00000	10.00000	1.000677e+07

print(df_price2['price'].value_counts())

```
print(df_price2['price'].value_counts())
✓ 0.0s

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
```

print("data null \n", df_price2.isnull().sum())

print("data kosong \n", df_price2.empty)

print("data nan \n", df_price2.isna().sum())

```
print("data null \n", df_price2.isnull().sum())
print("data kosong \n", df_price2.empty)
print("data nan \n", df_price2.isna().sum())
✓ 0.0s

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
dtype: int64
data kosong
 False
data nan
 squaremeters       0
numberofrooms      0
hasyard            0
...
hasstorageroom     0
hasguestroom       0
price              0
dtype: int64
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

print("Sebelum drop missing value", df_price2.shape)

df_price2 = df_price2.dropna(how="any", inplace=False)

print("Sesudah drop missing value", df_price2.shape)

```
print("Sebelum drop missing value", df_price2.shape)
df_price2 = df_price2.dropna(how="any", inplace=False)
print("Sesudah drop missing value", df_price2.shape)
```

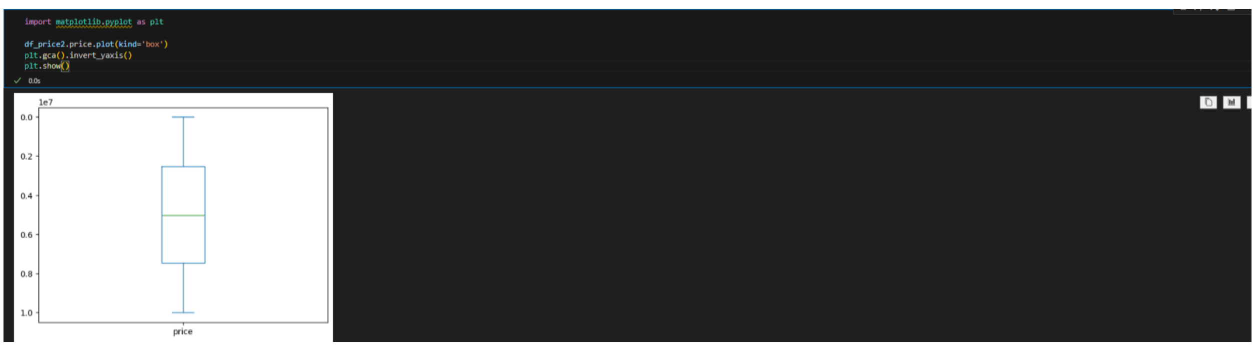
```
Sebelum drop missing value (10000, 17)
Sesudah drop missing value (10000, 17)
```

```
print("Sebelum pengecekan data duplikat, ", df_price2.shape)

df_price3 = df_price2.drop_duplicates(keep='last')

print("Sesudah pengecekan data duplikat, ", df_price3.shape)
```

```
print("Sebelum pengecekan data duplikat, ", df_price2.shape)
df_price3 = df_price2.drop_duplicates(keep='last')
print("Sesudah pengecekan data duplikat, ", df_price3.shape)
```

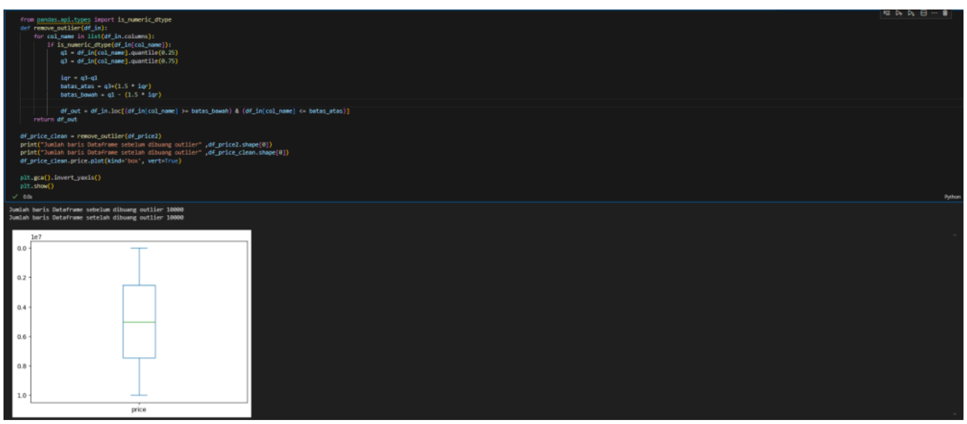
```
Sebelum pengecekan data duplikat,  (10000, 17)
Sesudah pengecekan data duplikat,  (10000, 17)
```

```
import matplotlib.pyplot as plt

df_price2.price.plot(kind='box')

plt.gca().invert_yaxis()

plt.show()
```

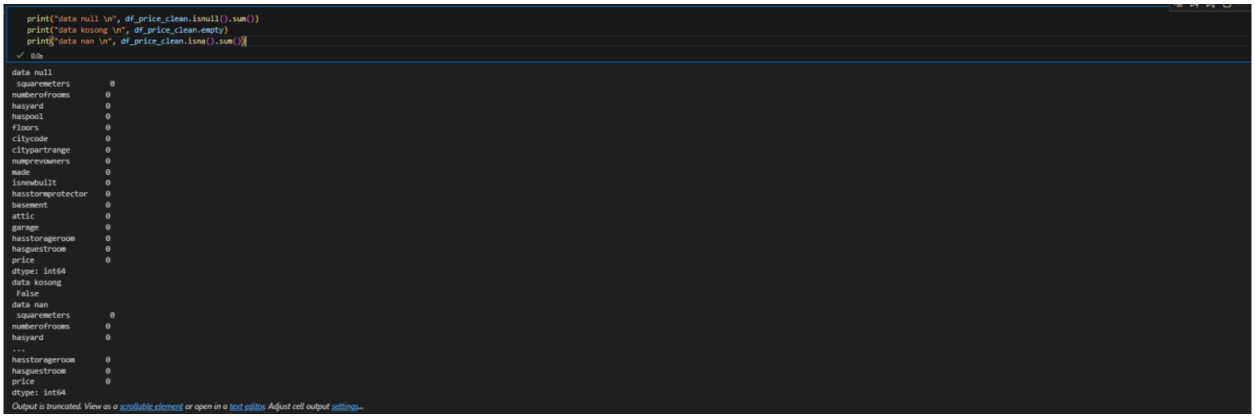



```
from pandas.api.types import is_numeric_dtype

def remove_outlier(df_in):

    for col_name in list(df_in.columns):

        if is_numeric_dtype(df_in[col_name]):

            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 outlier" ,df_price2.shape[0])

print("Jumlah baris Dataframe setelah dibuang
outlier" ,df_price_clean.shape[0])

df_price_clean.price.plot(kind='box', vert=True)


plt.gca().invert_yaxis()

plt.show()
```

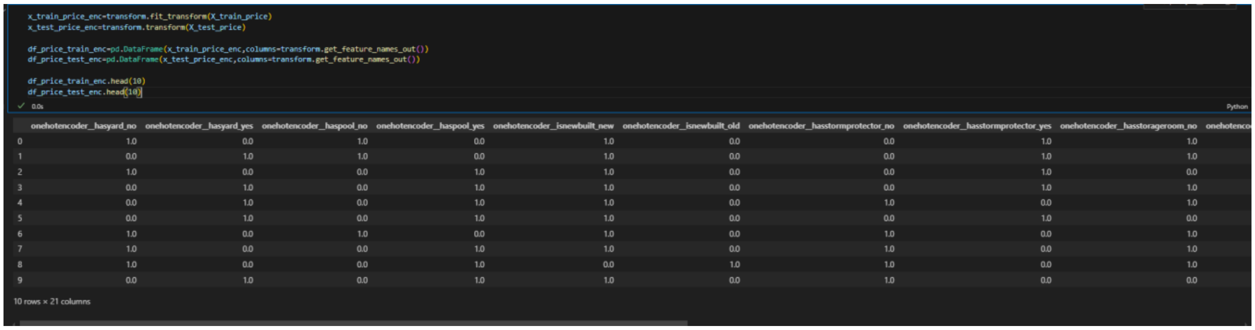



```
print("data null \n", df_price_clean.isnull().sum())

print("data kosong \n", df_price_clean.empty)

print("data nan \n", df_price_clean.isna().sum())
```




```
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_price, X_test_price, y_train_price, y_test_price =
train_test_split(X_regress, y_regress, test_size=0.25, random_state=2)



from sklearn.preprocessing import OneHotEncoder

from sklearn.compose import make_column_transformer


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


transform = make_column_transformer(

    (OneHotEncoder(), kolom_price), remainder='passthrough'

)


x_train_price_enc=transform.fit_transform(X_train_price)

x_test_price_enc=transform.transform(X_test_price)
```

```
df_price_train_enc=pd.DataFrame(x_train_price_enc,columns=transform.get_featu
re_names_out())

df_price_test_enc=pd.DataFrame(x_test_price_enc,columns=transform.get_feature
_names_out())


df_price_train_enc.head(10)

df_price_test_enc.head(10)
```



```
x_train_price_enc=transform.fit_transform(X_train_price)
x_test_price_enc=transform.transform(X_test_price)

df_price_train_enc=pd.DataFrame(x_train_price_enc,columns=transform.get_feature_names_out())
df_price_test_enc=pd.DataFrame(x_test_price_enc,columns=transform.get_feature_names_out())

df_price_train_enc.head(10)
df_price_test_enc.head(10)
```

0.0s

Python

	onehotencoder__hasyard_no	onehotencoder__hasyard_yes	onehotencoder__haspool_no	onehotencoder__haspool_yes	onehotencoder__isnewbuilt_new	onehotencoder__isnewbuilt_old	onehotencoder__hasstormprotector_no	onehotencoder__hasstormprotector_yes	onehotencoder__hasstorageroom_no
0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0
1	0.0	1.0	1.0	0.0	1.0	0.0	0.0	1.0	1.0
2	1.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0
3	0.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0	1.0
4	0.0	1.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0
5	0.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0
6	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0
7	1.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0
8	1.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	1.0
9	0.0	1.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0

10 rows × 21 columns

```
from sklearn.linear_model import Lasso

from sklearn.ensemble import RandomForestRegressor

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_Lasso = Pipeline(steps=[

            ('scale', StandardScaler()),

            ('feature_selection', SelectKBest(score_func=f_regression)),

            ('reg', Lasso(max_iter=1000))

            ])


param_grid_Lasso = {

    'reg__alpha': [0.01,0.1,1,10,100],

    'feature_selection__k': np.arange(1,20)
```

```
}

GSCV_Lasso = GridSearchCV(pipe_Lasso, param_grid_Lasso, cv=5,
scoring='neg_mean_squared_error')

GSCV_Lasso.fit(x_train_price_enc, y_train_price)

print("Best model:{}".format(GSCV_Lasso.best_estimator_))
print("Lasso best parameters:{}".format(GSCV_Lasso.best_params_))

print("Koefisien/bobot:{}".format(GSCV_Lasso.best_estimator_.named_steps['reg
'].coef_))
print("Intercept/bias:{}".format(GSCV_Lasso.best_estimator_.named_steps['reg'
].intercept_))

Lasso_predict = GSCV_Lasso.predict(x_test_price_enc)

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

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

```python
from sklearn.linear_model import Lasso
from sklearn.ensemble import RandomForestRegressor
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_Lasso = Pipeline(steps=[
            ('scale', StandardScaler()),
            ('feature_selection', SelectKBest(score_func=f_regression)),
            ('reg', Lasso(max_iter=1000))
            ])

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

GSCV_Lasso = GridSearchCV(pipe_Lasso, param_grid_Lasso, cv=5, scoring='neg_mean_squared_error')

GSCV_Lasso.fit(x_train_price_enc, y_train_price)

print("Best model:{}".format(GSCV_Lasso.best_estimator_))
print("Lasso best parameters:{}".format(GSCV_Lasso.best_params_))

print("Koefisien/bobot:{}".format(GSCV_Lasso.best_estimator_.named_steps['reg'].coef_))
print("Intercept/bias:{}".format(GSCV_Lasso.best_estimator_.named_steps['reg'].intercept_))

Lasso_predict = GSCV_Lasso.predict(x_test_price_enc)

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

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

3.3s

```
Best model:Pipeline(steps=[('scale', StandardScaler()),
                ('feature_selection',
                 SelectKBest(k=19,
                             score_func=<function f_regression at 0x0000022C88A18EA0>)),
                ('reg', Lasso(alpha=10))])
Lasso best parameters:{'feature_selection__k': 19, 'reg__alpha': 10}
Koefisien/bobot:[-1.48790569e+03  8.20728019e-13 -1.48419052e+03  4.88944352e-13
  5.30504539e+01 -0.00000000e+00 -6.09793957e+01  2.79280357e-11
 -0.00000000e+00  0.00000000e+00  2.87693034e+06  0.00000000e+00
  1.56659678e+03 -1.72180843e+01  1.21928155e+02  1.10254383e+01
 -0.00000000e+00  0.00000000e+00  2.26690934e+01]
Intercept/bias:4974345.527720001
Lasso Mean Squared Error (MSE): 3495420.15157818
Lasso Mean Absolute Error (MAE): 1441.6110198833521
Lasso Root Mean Squared Error: 1869.6042767329616
```

```python
df_results = pd.DataFrame(y_test_price, columns=['price'])

df_results = pd.DataFrame(y_test_price)

df_results['Lasso Prediction'] = Lasso_predict


df_results['Selisih_Price_LR'] = df_results['Lasso Prediction'] -
df_results['price']


df_results.head()
```

```python
df_results = pd.DataFrame(y_test_price, columns=['price'])
df_results = pd.DataFrame(y_test_price)
df_results['Lasso Prediction'] = Lasso_predict

df_results['Selisih_Price_LR'] = df_results['Lasso Prediction'] - df_results['price']

df_results.head()
```

0.0s

	price	Lasso Prediction	Selisih_Price_LR
7878	4963749.8	4.963847e+06	96.884528
3224	8498552.3	8.497207e+06	-1345.525166
1919	9887158.4	9.885706e+06	-1452.633203
4432	7075926.0	7.073797e+06	-2128.546109
4835	1399630.5	1.401528e+06	1897.182357

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

```
df_results.describe()
```

0.0s

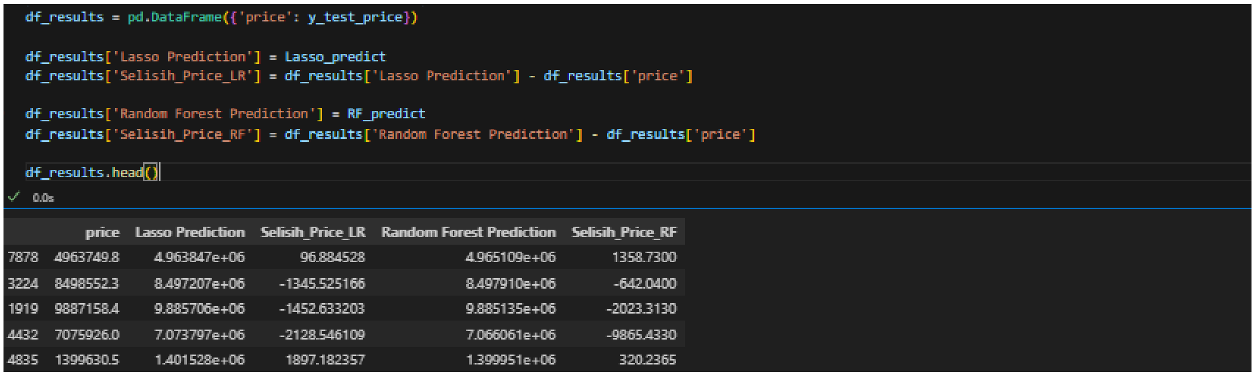
	price	Lasso Prediction	Selisih_Price_LR
count	2.500000e+03	2.500000e+03	2500.000000
mean	5.050754e+06	5.050712e+06	-41.880343
std	2.878183e+06	2.878143e+06	1869.509084
min	1.031350e+04	1.340584e+04	-6453.804795
25%	2.587625e+06	2.586494e+06	-1210.040834
50%	5.056844e+06	5.057061e+06	-28.369071
75%	7.519084e+06	7.517498e+06	1079.030986
max	9.999687e+06	9.997085e+06	6202.080918

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np
import pandas as pd

pipe_RF = Pipeline(steps=[
    ('scale', StandardScaler()),
    ('feature_selection', SelectKBest(score_func=f_regression)),
    ('reg', RandomForestRegressor(random_state=2))
])

param_grid_RF = {
    'reg__n_estimators': [50, 100, 200],
    'reg__max_depth': [None, 10, 20, 30],
    'feature_selection__k': np.arange(1, 20)
}

GSCV_RF = GridSearchCV(pipe_RF, param_grid_RF, cv=5,
scoring='neg_mean_squared_error')
```

```
GSCV_RF.fit(x_train_price_enc, y_train_price)


print("Best model:{}".format(GSCV_RF.best_estimator_))

print("Random Forest best parameters:{}".format(GSCV_RF.best_params_))


RF_predict = GSCV_RF.predict(x_test_price_enc)

mse_RF = mean_squared_error(y_test_price, RF_predict)

mae_RF = mean_absolute_error(y_test_price, RF_predict)


print("Random Forest Mean Squared Error (MSE): {}".format(mse_RF))

print("Random Forest Mean Absolute Error (MAE): {}".format(mae_RF))

print("Random Forest Root Mean Squared Error: {}".format(np.sqrt(mse_RF)))


df_results['Random Forest Prediction'] = RF_predict

df_results['Selisih_Price_RF'] = df_results['Random Forest Prediction'] -
df_results['price']


df_results.head()


df_results.describe()




df_results = pd.DataFrame({'price': y_test_price})


df_results['Lasso Prediction'] = Lasso_predict

df_results['Selisih_Price_LR'] = df_results['Lasso Prediction'] -
df_results['price']


df_results['Random Forest Prediction'] = RF_predict
```

```
df_results['Selisih_Price_RF'] = df_results['Random Forest Prediction'] - df_results['price']
```

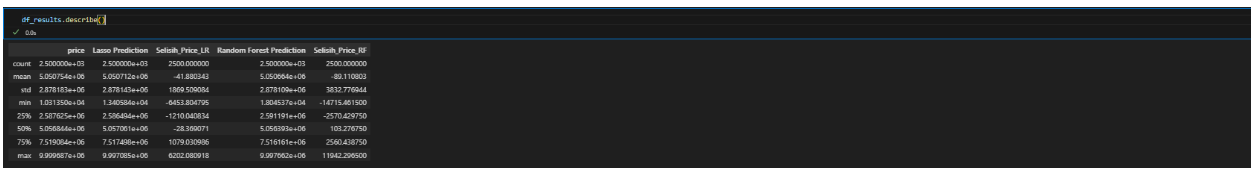
```
df_results.head()
```



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

df_results['Lasso Prediction'] = Lasso_predict
df_results['Selisih_Price_LR'] = df_results['Lasso Prediction'] - df_results['price']

df_results['Random Forest Prediction'] = RF_predict
df_results['Selisih_Price_RF'] = df_results['Random Forest Prediction'] - df_results['price']

df_results.head()
```
0.0s

	price	Lasso Prediction	Selisih_Price_LR	Random Forest Prediction	Selisih_Price_RF
7878	4963749.8	4.963847e+06	96.884528	4.965109e+06	1358.7300
3224	8498552.3	8.497207e+06	-1345.525166	8.497910e+06	-642.0400
1919	9887158.4	9.885706e+06	-1452.633203	9.885135e+06	-2023.3130
4432	7075926.0	7.073797e+06	-2128.546109	7.066061e+06	-9865.4330
4835	1399630.5	1.401528e+06	1897.182357	1.399951e+06	320.2365

```
df_results.describe()
```



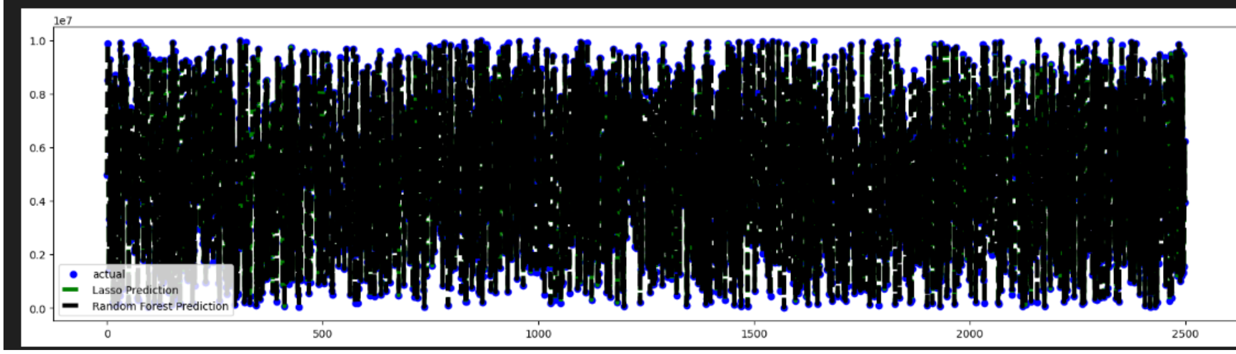
```
df_results.describe()
```
0.0s

	price	Lasso Prediction	Selisih_Price_LR	Random Forest Prediction	Selisih_Price_RF
count	2.500000e+03	2.500000e+03	2500.000000	2.500000e+03	2500.000000
mean	5.050754e+06	5.050712e+06	-41.880343	5.050664e+06	-89.110803
std	2.878183e+06	2.878143e+06	1869.509084	2.878109e+06	3832.776944
min	1.031350e+04	1.340584e+04	-6453.804795	1.804537e+04	-14715.461500
25%	2.587625e+06	2.586494e+06	-1210.040834	2.591191e+06	-2570.429750
50%	5.056844e+06	5.057061e+06	-28.369071	5.056393e+06	103.276750
75%	7.519084e+06	7.517498e+06	1079.030986	7.516161e+06	2560.438750
max	9.999687e+06	9.997085e+06	6202.080918	9.997662e+06	11942.296500

```
import matplotlib.pyplot as plt


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

data_len = range(len(y_test_price))

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

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

plt.plot(data_len, df_results['Random Forest Prediction'], label="Random Forest Prediction", color="black", linewidth=4, linestyle="--")

plt.legend()

plt.show
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(20,5))
data_len = range(len(y_test_price))
plt.scatter(data_len, df_results.price, label="actual", color="blue")
plt.plot(data_len, df_results['Lasso Prediction'], label="Lasso Prediction", color="green", linewidth=4, linestyle="dashed")
plt.plot(data_len, df_results['Random Forest Prediction'], label="Random Forest Prediction", color="black", linewidth=4, linestyle="--")
plt.legend()
plt.show
```
✓ 0.4s

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

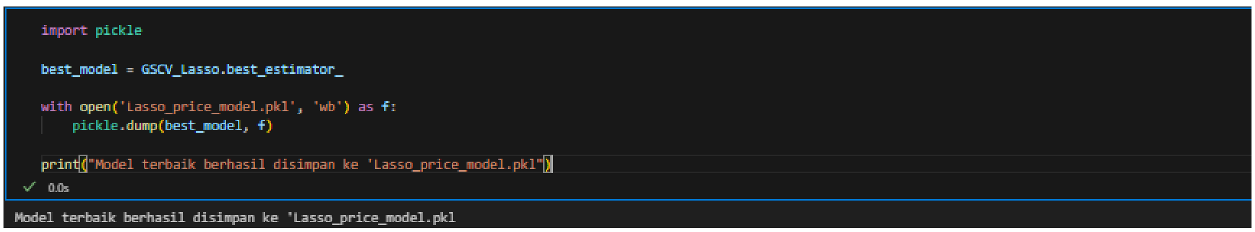



```
mae_Lasso = mean_absolute_error(df_results['price'], df_results['Lasso
Prediction'])

rmse_lasso = np.sqrt(mean_squared_error(df_results['price'],
df_results['Lasso Prediction']))

lasso_feature_count = GSCV_Lasso.best_params_['feature_selection__k']


mae_Random = mean_absolute_error(df_results['price'], df_results['Random
Forest Prediction'])

rmse_Random = np.sqrt(mean_squared_error(df_results['price'],
df_results['Random Forest Prediction']))

Random_feature_count = GSCV_Lasso.best_params_['feature_selection__k']


print(f"Lasso MAE: {mae_Lasso}, Lasso RMSE: {rmse_lasso}, Lasso Feature
Count: {lasso_feature_count}")

print(f"Random Forest MAE: {mae_Random}, Random Forest RMSE: {rmse_Random},
Random Forest Feature Count: {Random_feature_count}")
```

```
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_['feature_selection__k']

mae_Random = mean_absolute_error(df_results['price'], df_results['Random Forest Prediction'])
rmse_Random = np.sqrt(mean_squared_error(df_results['price'], df_results['Random Forest Prediction']))
Random_feature_count = GSCV_Lasso.best_params_['feature_selection__k']

print(f"Lasso MAE: {mae_Lasso}, Lasso RMSE: {rmse_lasso}, Lasso Feature Count: {lasso_feature_count}")
print(f"Random Forest MAE: {mae_Random}, Random Forest RMSE: {rmse_Random}, Random Forest Feature Count: {Random_feature_count}")
```
✓ 0.0s

```
Lasso MAE: 1441.6110198833521, Lasso RMSE: 1869.6042767329616, Lasso Feature Count: 19
Random Forest MAE: 3057.164297400172, Random Forest RMSE: 3833.046277283069, Random Forest Feature Count: 19
```

```
import pickle

best_model = GSCV_Lasso.best_estimator_

with open('Lasso_price_model.pkl', 'wb') as f:
    pickle.dump(best_model, f)

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



LOAD DATASET :

Load datasaet berdasarkan path di mana dataset disimpan

```
import pandas as pd
import numpy as np

df_dataset = pd.read_csv(r'C:\Pembelajaran Mesin dan Pembelajaran Mendalam\UTS\Dataset UTS_Gasal 2425.csv')
df_dataset.head(10)
```

```python
import pandas as pd
import numpy as np

df_dataset = pd.read_csv(r'C:\Pembelajaran Mesin dan Pembelajaran Mendalam\UTS\Dataset UTS_Gasal 2425.csv')
df_dataset.head(10)
```

✓ 0.9s Python

	squaremeters	numberofrooms	hasyard	haspool	floors	citycode	citypartrange	numprevowners	made	isnewbuilt	hasstormprotector	basement	attic	garage
0	75523	3	no	yes	63	9373	3	8	2005	old	yes	4313	9005	956
1	55712	58	no	yes	19	34457	6	8	2021	old	no	2937	8852	135
2	86929	100	yes	no	11	98155	3	4	2003	new	no	6326	4748	654
3	51522	3	no	no	61	9047	8	3	2012	new	yes	632	5792	807
4	96470	74	yes	no	21	92029	4	2	2011	new	yes	5414	1172	716
5	79770	3	no	yes	69	54812	10	5	2018	old	yes	8871	7117	240
6	75985	60	yes	no	67	6517	6	9	2009	new	yes	4878	281	384
7	64169	88	no	yes	6	61711	3	9	2011	new	yes	3054	129	726
8	92383	12	no	no	78	71982	3	7	2000	old	no	7507	9056	892
9	95121	46	no	yes	3	9382	7	9	1994	old	no	615	1221	328

Data Cleansing :

```python
df_dataset2 = df_dataset.drop(['category'], axis=1)
df_dataset2.head()
```

```python
df_dataset2 = df_dataset.drop(['category'], axis=1)
df_dataset2.head()
```

✓ 0.0s Python

	squaremeters	numberofrooms	hasyard	haspool	floors	citycode	cityp
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	

```python
df_dataset2.info()
```

```python
df_dataset2.info()
```

✓ 0.0s

```
<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
```

```python
df_dataset2.describe()
```

```python
df_dataset2.describe()
```

✓ 0.0s Python

	squaremeters	numberofrooms	floors	citycode	citypartra
count	10000.00000	10000.000000	10000.000000	10000.000000	10000.00
mean	49870.13120	50.358400	50.276300	50225.486100	5.51
std	28774.37535	28.816696	28.889171	29006.675799	2.87
min	89.00000	1.000000	1.000000	3.000000	1.00
25%	25098.50000	25.000000	25.000000	24693.750000	3.00
50%	50105.50000	50.000000	50.000000	50693.000000	5.00
75%	74609.75000	75.000000	76.000000	75683.250000	8.00
max	99999.00000	100.000000	100.000000	99953.000000	10.00

```python
print(df_dataset2['price'].value_counts())
```

```python
print(df_dataset2['price'].value_counts())
```

✓ 0.0s Python

```
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
```

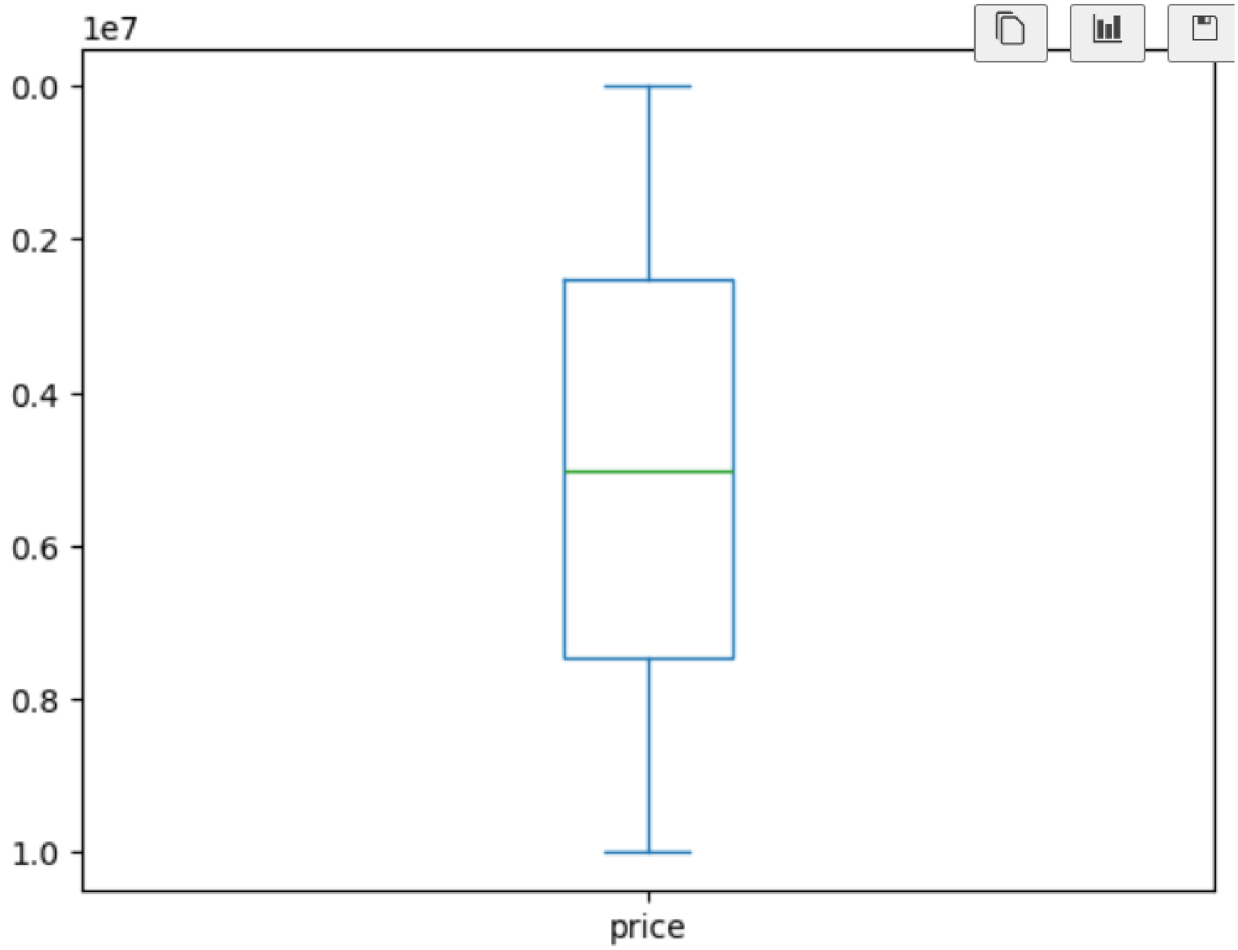
```python
print("data null \n", df_dataset2.isnull().sum())
print("data kosong \n", df_dataset2.empty)
```

```
print("data nan \n", df_dataset2.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
     dtype: int64
     data kosong
      False
     data nan
      squaremeters         0
     numberofrooms        0
     hasyard              0
     ...
     hasstorageroom       0
     hasguestroom         0
     price                0
     dtype: int64
```

DELETE DATA KOSONG :

```
import matplotlib.pyplot as plt

df_dataset2.price.plot(kind='box')
plt.gca().invert_yaxis()
plt.show()
```

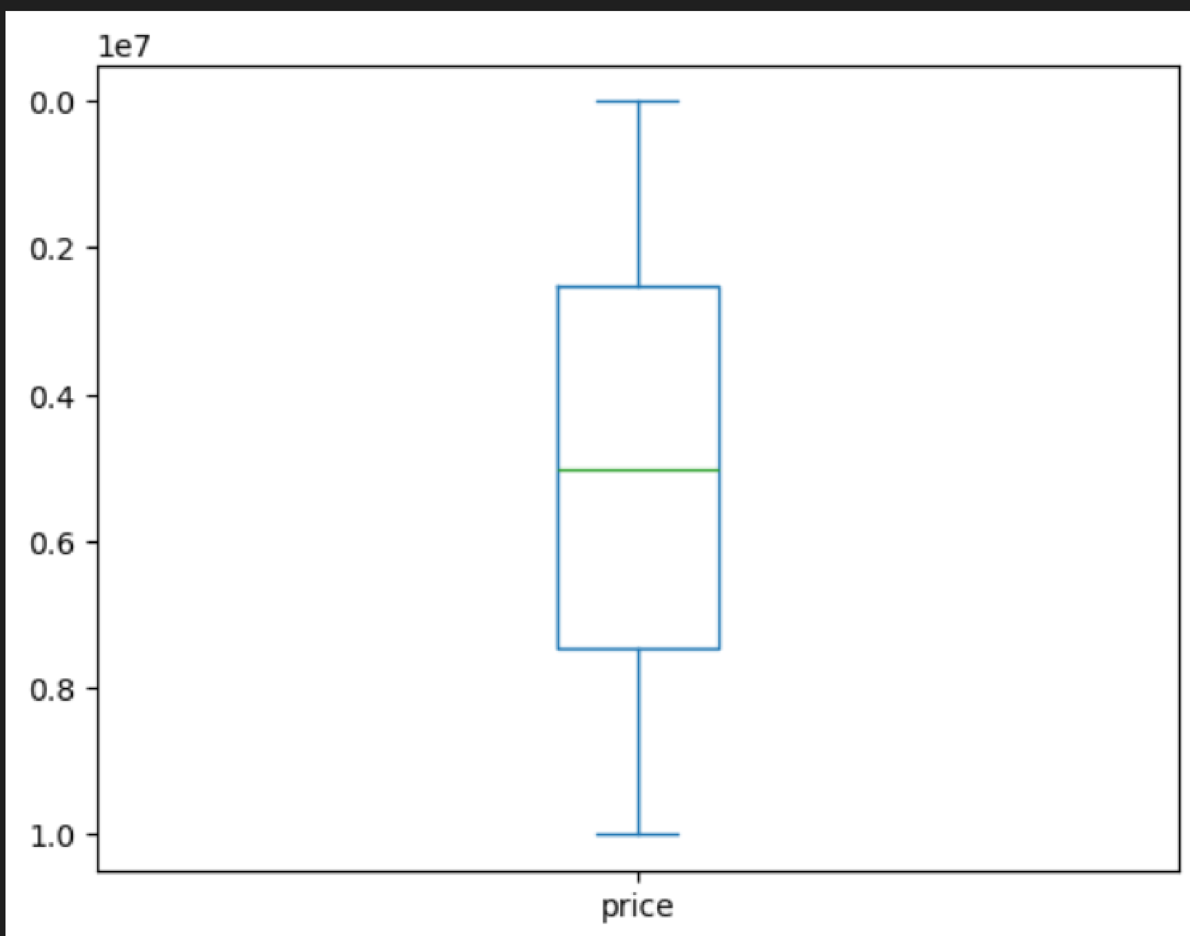


Pembersihan outlier dengan metode inter-quartile range, Menampilkan perbandingan jumlah baris dari Dataframe sebelum dan sesudah pembersihan outl

```
from pandas.api.types import is_numeric_dtype
def remove_outlier(df_in):
    for col_name in list(df_in.columns):
        if is_numeric_dtype(df_in[col_name]):
            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_dataset_clean =  remove_outlier(df_dataset2)
print("Jumlah baris DataFrame sebelum dibuang outlier" ,df_dataset2.shape[0])
print("Jumlah baris DataFrame sesudah dibuang
outlier" ,df_dataset_clean.shape[0])
df_dataset_clean.price.plot(kind='box', vert=True)

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

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



Mengecek ulang apakah masih ada kemungkinan data yang kosong, null, atau NaN

```
print("data null\n", df_dataset_clean.isnull().sum())
```

```
print("data_kosong \n", df_dataset_clean.empty)
print("data nan \n", df_dataset_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
dtype: int64
data_kosong
 False
data nan
 squaremeters         0
numberofrooms        0
hasyard              0
...
hasstorageroom       0
hasguestroom         0
price                0
dtype: int64
```

Train-test split

```
from sklearn.model_selection import train_test_split
```

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

X_train_dataset, X_test_dataset, y_train_dataset, y_test_dataset =
train_test_split(X_regress, y_regress, \

test_size=0.25,

random_state=2)
```

Ridge Regressor :

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer

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

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

```
import numpy as np
import pandas as pd
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.metrics import mean_absolute_error, mean_squared_error



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

le = LabelEncoder()
for col in cols_to_encode:
    X_train_dataset[col] = le.fit_transform(X_train_dataset[col])
```

```
    X_test_dataset[col] = le.transform(X_test_dataset[col])

X_test_dataset = X_test_dataset.reindex(columns=X_train_dataset.columns,
fill_value=0)

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

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

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

GSCV_RR.fit(X_train_dataset, y_train_dataset)

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

Ridge_predict = GSCV_RR.predict(X_test_dataset)

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

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

```
Best model: Pipeline(steps=[('scale', StandardScaler()),
                ('feature_selection',
                 SelectKBest(k=16,
                             score_func=<function f_regression at 0x000(
                ('reg', Ridge(alpha=0.01))])
Ridge best parameters: {'feature_selection__k': 16, 'reg__alpha': 0.01}
Koefisien/bobot: [ 2.87693661e+06  5.39975172e+00  1.49809603e+03  1.49:
  1.57591520e+03 -2.70950125e+01  1.31994833e+02  2.02393175e+01
 -6.44422468e+00 -6.37760424e+01  7.09678679e+01  8.23264867e-01
 -8.27797603e+00  3.17725120e+01  7.54511709e+00 -2.15571536e+01]
Intercept/bias: 4974345.527720001
Ridge Mean Squared Error (MSE): 3493242.884751669
Ridge Mean Absolute Error (MAE): 1441.1902350041391
Ridge Root Mean Squared Error: 1869.0219059047085
```

```python
df_results = pd.DataFrame(y_test_dataset, columns=['price'])
df_results = pd.DataFrame(y_test_dataset)
df_results['Ridge Prediction'] = Ridge_predict

df_results['Selisih_price_RR'] = df_results['Ridge Prediction'] -
df_results['price']

df_results.head()
```

	price	Ridge Prediction	Selisih_price_RR
7878	4963749.8	4.963841e+06	91.176731
3224	8498552.3	8.497250e+06	-1302.275680
1919	9887158.4	9.885726e+06	-1432.017087
4432	7075926.0	7.073877e+06	-2049.381803
4835	1399630.5	1.401514e+06	1883.637919

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

```
df_results.describe()
```

✓ 0.0s Python

	price	Ridge Prediction	Selisih_price_RR
count	2.500000e+03	2.500000e+03	2500.000000
mean	5.050754e+06	5.050712e+06	-41.995920
std	2.878183e+06	2.878150e+06	1868.923856
min	1.031350e+04	1.334784e+04	-6451.396209
25%	2.587625e+06	2.586459e+06	-1215.350130
50%	5.056844e+06	5.057013e+06	-35.509601
75%	7.519084e+06	7.517520e+06	1089.488014
max	9.999687e+06	9.997104e+06	6209.475727

SVR Regressor :

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



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

le = LabelEncoder()
for col in cols_to_encode:
    X_train_dataset[col] = le.fit_transform(X_train_dataset[col])
    X_test_dataset[col] = le.transform(X_test_dataset[col])

X_test_dataset = X_test_dataset.reindex(columns=X_train_dataset.columns,
fill_value=0)
```

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

param_grid_SVR = {
    'reg__C': [0.01, 0.1, 1, 10, 100],
    'reg__epsilon': [0.1, 0.2, 0.5, 1],
    'feature_selection__k': np.arange(1, X_train_dataset.shape[1] + 1)
}

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

GSCV_SVR.fit(X_train_dataset, y_train_dataset)

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

SVR_predict = GSCV_SVR.predict(X_test_dataset)

mse_SVR = mean_squared_error(y_test_dataset, SVR_predict)
mae_SVR = mean_absolute_error(y_test_dataset, 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: {}".format(np.sqrt(mse_SVR)))
```

```
Best model: Pipeline(steps=[('scale', StandardScaler()),
                ('feature_selection',
                 SelectKBest(k=2,
                             score_func=<function f_regression at 0x000(
                ('reg', SVR(C=100, kernel='linear'))])
Ridge best parameters: {'feature_selection__k': 2, 'reg__C': 100, 'reg_
Koefisien/bobot: [[647987.26025222 -12854.84609513]]
Intercept/bias: [4991740.35485764]
SVR Mean Squared Error (MSE): 4972950021790.335
SVR Mean Absolute Error (MAE): 1928006.6061967392
SVR Root Mean Squared Error: 2230011.215619853
```

```
df_results['SVR Prediction'] = SVR_predict
df_results = pd.DataFrame(y_test_dataset)
df_results['SVR Prediction'] = SVR_predict

df_results['Selisih_price_SVR'] = df_results['SVR Prediction'] -
df_results['price']
df_results.head()
```

	price	SVR Prediction	Selisih_price_SVR
7878	4963749.8	5.002022e+06	3.827264e+04
3224	8498552.3	5.768450e+06	-2.730103e+06
1919	9887158.4	6.081885e+06	-3.805274e+06
4432	7075926.0	5.441772e+06	-1.634154e+06
4835	1399630.5	4.201222e+06	2.801591e+06

```
df_results.describe()
```

	price	SVR Prediction	Selisih_price_SVR
count	2.500000e+03	2.500000e+03	2.500000e+03
mean	5.050754e+06	5.008845e+06	-4.190858e+04
std	2.878183e+06	6.482842e+05	2.230063e+06
min	1.031350e+04	3.861014e+06	-3.893315e+06
25%	2.587625e+06	4.447310e+06	-1.945206e+06
50%	5.056844e+06	5.012500e+06	-4.954392e+04
75%	7.519084e+06	5.565493e+06	1.875200e+06
max	9.999687e+06	6.140747e+06	3.878636e+06

Membandingkan dataframe hasil dari setiap model yang sudah di latih sebelumnya :

```
df_results = pd.DataFrame({'price': y_test_dataset})

df_results['Ridge Prediction'] = Ridge_predict

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()
```

	price	Ridge Prediction	Selisih_price_RR	SVR Prediction	Selisih_price_S
7878	4963749.8	4.963841e+06	-91.176731	5.002022e+06	-3.827264e
3224	8498552.3	8.497250e+06	1302.275680	5.768450e+06	2.730103e
1919	9887158.4	9.885726e+06	1432.017087	6.081885e+06	3.805274e
4432	7075926.0	7.073877e+06	2049.381803	5.441772e+06	1.634154e
4835	1399630.5	1.401514e+06	-1883.637919	4.201222e+06	-2.801591e

```
df_results.describe()
```

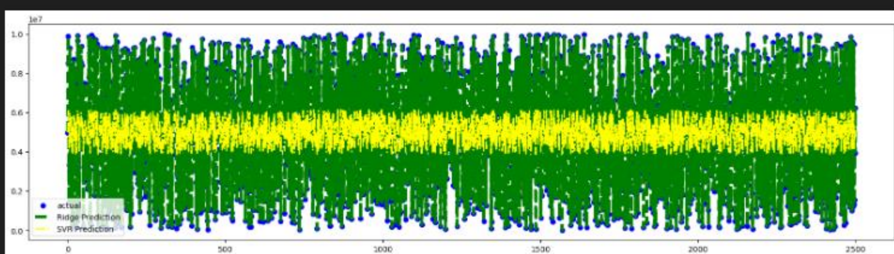
	price	Ridge Prediction	Selisih_price_RR	SVR Prediction	Selisih_pr
count	2.500000e+03	2.500000e+03	2500.000000	2.500000e+03	2.500
mean	5.050754e+06	5.050712e+06	41.995920	5.008845e+06	4.190
std	2.878183e+06	2.878150e+06	1868.923856	6.482842e+05	2.230
min	1.031350e+04	1.334784e+04	-6209.475727	3.861014e+06	-3.878
25%	2.587625e+06	2.586459e+06	-1089.488014	4.447310e+06	-1.875
50%	5.056844e+06	5.057013e+06	35.509601	5.012500e+06	4.954
75%	7.519084e+06	7.517520e+06	1215.350130	5.565493e+06	1.945
max	9.999687e+06	9.997104e+06	6451.396209	6.140747e+06	3.893

Membuat grafik untuk menunjukkan perbandingan antara data asli, hasil prediksi Ridge Regression dan SVR Regression dari dataframe hasil :

```
import matplotlib.pyplot as plt

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

data_len = range(len(y_test_dataset))

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=2, linestyle="-.")

plt.legend()
plt.show
```

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



Memilih model terbaik berdasarkan seberapa mirip prediksinya dengan data asli

```
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_RR.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_feature_count = GSCV_RR.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}")
```

✓ 0.0s

```
Ridge MAE: 1441.1902350041391, Ridge RMSE: 1869.0219059047085, Ridge Feature Count: 16
SVR MAE: 1928006.6061967392, SVR RMSE: 2230011.215619853, SVR Feature Count: 2
```

Dump model dengan MAE dan RMSE terendah

```
import pickle

best_model = GSCV_SVR.best_estimator_

with open('SVR_price_model.pkl', 'wb') as f:
    pickle.dump(best_model, f)

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

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

Load data set

```
import pandas as pd
```

```python
import numpy as np

df_kategori=pd.read_csv('Dataset UTS_Gasal 2425.csv')
df_kategori.head(10)
```

	squaremeters	numberofrooms	hasyard	haspool	floors	citycode	c
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	

```python
df_kategori2 = df_kategori.drop(['price'], axis=1)
df_kategori2.head()
```

	squaremeters	numberofrooms	hasyard	haspool	floors	citycode	c
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	

```python
df_kategori2.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  category           10000 non-null  object
dtypes: int64(11), object(6)
memory usage: 1.3+ MB
```

```
df_kategori2.describe()
```

	squaremeters	numberofrooms	floors	citycode	citypa
count	10000.00000	10000.000000	10000.000000	10000.000000	1000(
mean	49870.13120	50.358400	50.276300	50225.486100	5
std	28774.37535	28.816696	28.889171	29006.675799	2
min	89.00000	1.000000	1.000000	3.000000	1
25%	25098.50000	25.000000	25.000000	24693.750000	3
50%	50105.50000	50.000000	50.000000	50693.000000	5
75%	74609.75000	75.000000	76.000000	75683.250000	8
max	99999.00000	100.000000	100.000000	99953.000000	1(

```
print("data NULL \n", df_kategori2.isnull().sum())
print("data  KOSONG \n", df_kategori2.empty)
print("data NaN \n", df_kategori2.isna().sum())
```

```
data NULL
 squaremeters         0
numberofrooms        0
hasyard              0
haspool              0
floors               0
citycode             0
citypartrange        0
numprevowners        0
made                 0
isnewbuilt           0
hasstormprotector    0
basement             0
attic                0
garage               0
hasstorageroom       0
hasguestroom         0
category             0
dtype: int64
data  KOSONG
 False
data NaN
 squaremeters         0
numberofrooms        0
hasyard              0
...
hasstorageroom       0
hasguestroom         0
category             0
dtype: int64
```

```python
print("Sebelum Pengecekan data duplikat, ", df_kategori2.shape)
df_kategori3=df_kategori2.drop_duplicates(keep='last')
print("Setelah Pengecekan data duplikat, ", df_kategori3.shape)
```

```
Sebelum Pengecekan data duplikat,  (10000, 17)
Setelah Pengecekan data duplikat,  (10000, 17)
```

```python
from sklearn.model_selection import train_test_split

x = df_kategori3.drop(columns=['category'],axis=1)
y= y=df_kategori3['category']

x_train, x_test, y_train, y_test =
train_test_split(x,y,test_size=0.3,random_state=2) #NPM 11902
```

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

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

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer

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

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

```
x_train_enc=transform.fit_transform(x_train)
x_test_enc=transform.fit_transform(x_test)

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

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

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

Random Forest

```python
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.feature_selection import SelectKBest, SelectPercentile
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,
classification_report
import matplotlib.pyplot as plt


pipe_RF=[('data scaling', StandardScaler()),
         ('feature select',SelectKBest()),

('clf',RandomForestClassifier(random_state=2,class_weight='balanced',max_features
='sqrt'))]

params_grid_RF = [{
                'data scaling' : [StandardScaler()],
                'feature select__k': np.arange(2, 6),
                'clf__max_depth': [4, 8, 12, None],
                'clf__n_estimators': [100, 200, 300]
                },
                {
                    'data scaling' : [StandardScaler()],
                    'feature select': [SelectPercentile()],
                    'feature select__percentile': [20, 40, 60, 80],
                    'clf__max_depth': [4, 8, 12, None],
                    'clf__n_estimators': [100, 200, 300]
                },
                {
            'data scaling' : [MinMaxScaler()],
            'feature select__k' : np.arange(2, 6),
            'clf__max_depth' : [4, 8, 12, None],
            'clf__n_estimators' : [100, 200, 300]
                },
                {
                    'data scaling' : [MinMaxScaler()],
                    'feature select' : [SelectPercentile()],
                    'feature select__percentile' : [20, 40, 60, 80],
                    'clf__max_depth' : [4, 8, 12, None],
                    'clf__n_estimators' : [100, 200, 300]
                   }]
```

```python
estimator_RF = Pipeline(pipe_RF)
SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=2)
GSCV_RF=GridSearchCV(estimator_RF,params_grid_RF,cv=SKF)
GSCV_RF.fit(x_train_enc,y_train)
print("RF training finished")
print("CV Score: {}".format(GSCV_RF.best_score_))
print("Test Score: {}".format(GSCV_RF.best_estimator_.score(x_test_enc,y_test)))
print("Best model:",GSCV_RF.best_estimator_)

mask = GSCV_RF.best_estimator_.named_steps['feature select'].get_support()
print("Best features:",df_train_enc.columns[mask])

RF_pred =  GSCV_RF.predict(x_test_enc)

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

plt.title("Random Forest Confusion Matrix")
plt.show()
print("Classification report RF:\n", classification_report(y_test, RF_pred))
```

```
CV Score: 0.9998571428571429
Test Score: 0.9996666666666667
Best model: Pipeline(steps=[('data scaling', MinMaxScaler()),
                ('feature select', SelectPercentile(percentile=80)),
                ('clf',
                 RandomForestClassifier(class_weight='balanced', max_depth=12,
                                        random_state=2))])
Best features: Index(['onehotencoder__hasyard_no', 'onehotencoder__hasyard_yes',
       'onehotencoder__haspool_no', 'onehotencoder__haspool_yes',
       'onehotencoder__isnewbuilt_new', 'onehotencoder__isnewbuilt_old',
       'onehotencoder__hasstormprotector_no',
       'onehotencoder__hasstormprotector_yes',
       'onehotencoder__hasstorageroom_no', 'onehotencoder__hasstorageroom_yes',
       'remainder__squaremeters', 'remainder__numberofrooms',
       'remainder__citypartrange', 'remainder__made', 'remainder__basement',
       'remainder__garage'],
      dtype='object')
```


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

       Basic       1.00      1.00      1.00      1286
      Luxury       1.00      1.00      1.00       940
      Middle       1.00      1.00      1.00       774

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

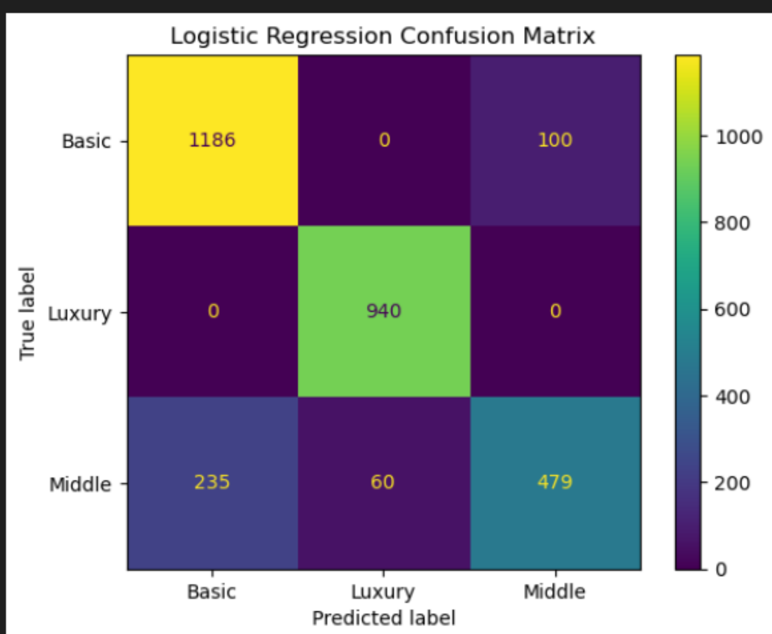
Logistic Regression

```
import numpy as np
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.feature_selection import SelectKBest, SelectPercentile
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV, StratifiedKFold
```

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,
classification_report
import matplotlib.pyplot as plt

pipe_LR = [('data scaling', StandardScaler()),
           ('feature select', SelectKBest()),
           ('clf', LogisticRegression(random_state=2, class_weight='balanced',
penalty ='l2'))]

params_grid_LR = [{
    'data scaling': [StandardScaler()],
    'feature select__k': np.arange(2, 6),
    'clf__solver' : ['liblinear']
},
{
    'data scaling': [StandardScaler()],
    'feature select': [SelectPercentile()],
    'feature select__percentile': [20, 50],
    'clf__solver' : ['liblinear']
},
{
    'data scaling': [MinMaxScaler()],
    'feature select__k': np.arange(2, 6),
    'clf__solver' : ['liblinear']
},
{
    'data scaling': [MinMaxScaler()],
    'feature select': [SelectPercentile()],
    'feature select__percentile': [20, 50],
    'clf__solver' : ['liblinear']
}]

estimator_LR = Pipeline(pipe_LR)
SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=2)
GSCV_LR = GridSearchCV(estimator_LR, params_grid_LR, cv=SKF)
GSCV_LR.fit(x_train_enc, y_train)
print("LR training finished")
print("CV Score: {}".format(GSCV_LR.best_score_))
print("Test Score: {}".format(GSCV_LR.best_estimator_.score(x_test_enc, y_test)))
print("Best model:", GSCV_LR.best_estimator_)

mask = GSCV_LR.best_estimator_.named_steps['feature select'].get_support()
print("Best features:", df_train_enc.columns[mask])
```

```python
LR_pred =  GSCV_LR.predict(x_test_enc)

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

plt.title("Logistic Regression Confusion Matrix")
plt.show()
print("Classification report RF:\n", classification_report(y_test, RF_pred))
```

```
CV Score: 0.8630000000000001
Test Score: 0.8683333333333333
Best model: Pipeline(steps=[('data scaling', StandardScaler()),
                ('feature select', SelectKBest(k=4)),
                ('clf',
                 LogisticRegression(class_weight='balanced', random_state=2,
                                    solver='liblinear'))])
Best features: Index(['onehotencoder__haspool_no', 'onehotencoder__haspool_yes',
       'onehotencoder__isnewbuilt_old', 'remainder__squaremeters'],
      dtype='object')
```




```
Classification report RF:
```

```python
import pickle

with open('BestModel_RF_LR_Keras.pkl','wb') as r:
    pickle.dump((GSCV_LR), r)

print("Model LR berhasil disimpan")
```

Best model menggunakan LR