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

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

Out[1]:

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

In [2]:
```
data = data.drop(['price'], axis=1)
```

In [3]:
```
data.describe()
#kalo dari mean sama max nya harusnya ada outlier
```

Out[3]:

	squaremeters	numberofrooms	floors	citycode	citypartrange	numprev
count	10000.00000	10000.000000	10000.000000	10000.000000	10000.000000	10000
mean	49870.13120	50.358400	50.276300	50225.486100	5.510100	5
std	28774.37535	28.816696	28.889171	29006.675799	2.872024	2
min	89.00000	1.000000	1.000000	3.000000	1.000000	1
25%	25098.50000	25.000000	25.000000	24693.750000	3.000000	3
50%	50105.50000	50.000000	50.000000	50693.000000	5.000000	5
75%	74609.75000	75.000000	76.000000	75683.250000	8.000000	8
max	99999.00000	100.000000	100.000000	99953.000000	10.000000	10

In [6]:
```
data.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
```

In [7]:
```
data['category'].value_counts()
```

Out[7]:
```
category
Basic     4344
Luxury    3065
Middle    2591
Name: count, dtype: int64
```

In [8]:
```
data.describe(include='all')
```

Out[8]:

	squaremeters	numberofrooms	hasyard	haspool	floors	citycode	city
count	10000.00000	10000.000000	10000	10000	10000.000000	10000.000000	10(
unique	NaN	NaN	2	2	NaN	NaN	
top	NaN	NaN	yes	no	NaN	NaN	
freq	NaN	NaN	5087	5032	NaN	NaN	
mean	49870.13120	50.358400	NaN	NaN	50.276300	50225.486100	
std	28774.37535	28.816696	NaN	NaN	28.889171	29006.675799	
min	89.00000	1.000000	NaN	NaN	1.000000	3.000000	
25%	25098.50000	25.000000	NaN	NaN	25.000000	24693.750000	
50%	50105.50000	50.000000	NaN	NaN	50.000000	50693.000000	
75%	74609.75000	75.000000	NaN	NaN	76.000000	75683.250000	
max	99999.00000	100.000000	NaN	NaN	100.000000	99953.000000	

In [9]:

```
print("Data Null \n",data.isnull().sum())
print("\nData Kosong \n",data.empty)
print("\nData Nan \n",data.isna().sum())
```

```
Data Null
 squaremeters         0
numberofrooms        0
hasyard              0
haspool              0
floors               0
citycode             0
citypartrange        0
numprevowners        0
made                 0
isnewbuilt           0
hasstormprotector    0
basement             0
attic                0
garage               0
hasstorageroom       0
hasguestroom         0
category             0
dtype: int64

Data Kosong
 False

Data Nan
 squaremeters         0
numberofrooms        0
hasyard              0
haspool              0
floors               0
citycode             0
citypartrange        0
numprevowners        0
made                 0
isnewbuilt           0
hasstormprotector    0
basement             0
attic                0
garage               0
hasstorageroom       0
hasguestroom         0
category             0
dtype: int64
```

In [10]:

```
import matplotlib.pyplot as plt

data['squaremeters'].plot(kind='box')
plt.gca().invert_yaxis()
plt.show()

#no outlier
```

In [11]:
```
from sklearn.model_selection import train_test_split
print("Sebelum drop data duplicate ",data.shape)
data2= data.drop_duplicates(keep='last')
print("Sesudah drop data duplicate ",data2.shape)

#klasifikasi
X = data2.drop(['category'],axis=1)
y = data2['category']
#regresi
#X = data2.drop(['price'],axis=1)
#y = data2['price']

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25,random_state

print(X_train.shape)
print(X_test.shape)
```

```
Sebelum drop data duplicate  (10000, 17)
Sesudah drop data duplicate  (10000, 17)
(7500, 16)
(2500, 16)
```

In [13]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer

kolom_kategori=['hasyard','haspool','isnewbuilt','hasstormprotector','hasstorageroo

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

```
X_train_encoded = transform.fit_transform(X_train)
X_test_encoded = transform.transform(X_test)


df_train_enc = pd.DataFrame(X_train_encoded, columns=transform.get_feature_names_ou
df_test_enc = pd.DataFrame(X_test_encoded, columns=transform.get_feature_names_out(



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

Out[13]:

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

10 rows × 21 columns

In [15]:

```
#ini bakal pake Random Forest dan Logistic Regression
# untuk skala bakal pake StandardScaler dan MinMaxScaler

from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.feature_selection import SelectKBest, SelectPercentile
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import classification_report,confusion_matrix,ConfusionMatrixD

pipe_RFC = Pipeline(steps=[
    ('scale', MinMaxScaler()),
    ('feat_select', SelectKBest()),
    ('clf', RandomForestClassifier())
])


param_RFC = [
   {
```

```
        'scale': [MinMaxScaler()],
        'feat_select__k': np.arange(5, 20),
        'clf__max_depth': [None, 10, 15, 20], #menentukan max depth dari random for
        'clf__n_estimators': [100, 200, 300] # ini buat jumlah pohon kalo ga salah
    },
    {
        'scale': [StandardScaler()],
        'feat_select': [SelectPercentile()],
        'feat_select__percentile': np.arange(10, 100, 10),
        'clf__max_depth': [None, 10, 15, 20],
        'clf__n_estimators': [100, 200, 300]
    },
    {
        'scale': [StandardScaler()],
        'feat_select__k': np.arange(5, 20),
        'clf__max_depth': [None, 10, 15, 20],
        'clf__n_estimators': [100, 200, 300]
    },
    {
        'scale': [MinMaxScaler()],
        'feat_select': [SelectPercentile()],
        'feat_select__percentile': np.arange(10, 100, 10),
        'clf__max_depth': [None, 10, 15, 20],
        'clf__n_estimators': [100, 200, 300]
    }
]

estimator_RFC = Pipeline(pipe_RFC)

STRA_K_FOLD = StratifiedKFold(n_splits=5, shuffle=True, random_state=91)
GSCV_RFC = GridSearchCV(pipe_RFC, param_RFC, cv=STRA_K_FOLD, n_jobs=-1, verbose=1)

#Dalam GridSearchCV, parameter yang ingin diuji, termasuk max_depth dan n_estimator

GSCV_RFC.fit(X_train_encoded, y_train)

print('FINISH')
```

```
Fitting 5 folds for each of 576 candidates, totalling 2880 fits
FINISH
```

In [16]:

```
print('CV Score : {}'.format(GSCV_RFC.best_score_))
print('Best Param : {}'.format(GSCV_RFC.best_params_))


print('Test Score : {}'.format(GSCV_RFC.best_estimator_.score(X_test_encoded, y_tes

print('Best Model : {}'.format(GSCV_RFC.best_estimator_))

# print('Best Feature : {}'.format(GSCV_RFC.best_estimator_.named_steps['feat_selec
mask = GSCV_RFC.best_estimator_.named_steps['feat_select'].get_support()
print('Best Feature: ', df_train_enc.columns[mask] )

RFC_predict = GSCV_RFC.predict(X_test_encoded)

cm = confusion_matrix(y_test, RFC_predict, labels=GSCV_RFC.classes_)
```

```
display = ConfusionMatrixDisplay(cm, display_labels=GSCV_RFC.classes_)
display.plot()
plt.title('Confusion Matrix Random Forest Classifier')
plt.show()


print("classification_report Random Forest Classifier:\n" , classification_report(y
```

```
CV Score : 0.9994666666666667
Best Param : {'clf__max_depth': 20, 'clf__n_estimators': 100, 'feat_select__k': 15,
'scale': MinMaxScaler()}
Test Score : 0.9988
Best Model : Pipeline(steps=[('scale', MinMaxScaler()), ('feat_select', SelectKBest
(k=15)),
                ('clf', RandomForestClassifier(max_depth=20))])
Best Feature:  Index(['onehotencoder__hasyard_no', 'onehotencoder__hasyard_yes',
       'onehotencoder__haspool_no', 'onehotencoder__haspool_yes',
       'onehotencoder__isnewbuilt_new', 'onehotencoder__isnewbuilt_old',
       'onehotencoder__hasstorageroom_no', 'onehotencoder__hasstorageroom_yes',
       'remainder__squaremeters', 'remainder__numberofrooms',
       'remainder__citypartrange', 'remainder__made', 'remainder__basement',
       'remainder__attic', 'remainder__garage'],
      dtype='object')
```

```
classification_report Random Forest Classifier:
              precision    recall  f1-score   support

       Basic       1.00      1.00      1.00      1036
      Luxury       1.00      1.00      1.00       788
      Middle       1.00      1.00      1.00       676

    accuracy                           1.00      2500
   macro avg       1.00      1.00      1.00      2500
weighted avg       1.00      1.00      1.00      2500
```

In [17]:

```
pipe_LR = Pipeline(steps=[
    ('scale', MinMaxScaler()),
    ('feat_select', SelectKBest()),
    ('clf', LogisticRegression())
])


param_LR = [
   {
        'scale': [MinMaxScaler()],
        'feat_select__k': np.arange(5, 20),
        'clf__C': [0.01, 0.1, 1, 10],
        'clf__penalty': ['l1', 'l2'],
        'clf__solver': ['saga'],
        'clf__max_iter': [10000]
        #pake saga sama max_iter biar ga error yang kovergensi belum terjadi tapi i
    },
    {
        'scale': [StandardScaler()],
        'feat_select': [SelectPercentile()],
        'feat_select__percentile': np.arange(10, 100, 10),
        'clf__C': [0.01, 0.1, 1, 10],
        'clf__penalty': ['l1', 'l2'],
        'clf__solver': ['saga'] ,
        'clf__max_iter': [10000]
    },
    {
        'scale': [StandardScaler()],
        'feat_select__k': np.arange(5, 20),
        'clf__C': [0.01, 0.1, 1, 10],
        'clf__penalty': ['l1', 'l2'],
        'clf__solver': ['saga'],
        'clf__max_iter': [10000]
    },
    {
        'scale': [MinMaxScaler()],
        'feat_select': [SelectPercentile()],
        'feat_select__percentile': np.arange(10, 100, 10),
        'clf__C': [0.01, 0.1, 1, 10],
        'clf__penalty': ['l1', 'l2'],
        'clf__solver': ['saga'],
        'clf__max_iter': [10000]
    }
]
```

```
estimator_LR = Pipeline(pipe_LR)

STRA_K_FOLD = StratifiedKFold(n_splits=10, shuffle=True, random_state=91)


GSCV_LR = GridSearchCV(pipe_LR, param_LR, cv=STRA_K_FOLD, n_jobs=-1, verbose=1)

GSCV_LR.fit(X_train_encoded, y_train)



print('finish')
```

```
Fitting 10 folds for each of 384 candidates, totalling 3840 fits
finish
```

In [18]:

```
print('CV Score : {}'.format(GSCV_LR.best_score_))
print('Best Param : {}'.format(GSCV_LR.best_params_))


print('Test Score : {}'.format(GSCV_LR.best_estimator_.score(X_test_encoded, y_test


print('Best Model : {}'.format(GSCV_LR.best_estimator_))


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


LR_predict = GSCV_LR.predict(X_test_encoded)

cm = confusion_matrix(y_test, LR_predict, labels=GSCV_LR.classes_)


display = ConfusionMatrixDisplay(cm, display_labels=GSCV_LR.classes_)
display.plot()
plt.title('Confusion Matrix Logistic Regression')
plt.show()

print("classification_report Logistic Regression:\n", classification_report(y_test,
```

```
CV Score : 0.8946666666666667
Best Param : {'clf__C': 10, 'clf__max_iter': 10000, 'clf__penalty': 'l1', 'clf__solv
er': 'saga', 'feat_select': SelectPercentile(), 'feat_select__percentile': 50, 'scal
e': StandardScaler()}
Test Score : 0.9028
Best Model : Pipeline(steps=[('scale', StandardScaler()),
                ('feat_select', SelectPercentile(percentile=50)),
                ('clf',
                 LogisticRegression(C=10, max_iter=10000, penalty='l1',
                                    solver='saga'))])
Best Feature:  Index(['onehotencoder__hasyard_no', 'onehotencoder__hasyard_yes',
       'onehotencoder__haspool_no', 'onehotencoder__haspool_yes',
       'onehotencoder__isnewbuilt_new', 'onehotencoder__isnewbuilt_old',
       'remainder__squaremeters', 'remainder__numberofrooms',
       'remainder__basement', 'remainder__garage'],
      dtype='object')
```

```
classification_report Logistic Regression:
              precision    recall  f1-score   support

       Basic       0.87      0.90      0.89      1036
      Luxury       1.00      1.00      1.00       788
      Middle       0.84      0.79      0.81       676

    accuracy                           0.90      2500
   macro avg       0.90      0.90      0.90      2500
weighted avg       0.90      0.90      0.90      2500
```

In [19]:
```
# harusnya sih yg lebih bagus itu Random Forest Classifier
if GSCV_RFC.best_estimator_.score(X_test_encoded, y_test) > GSCV_LR.best_estimator_
    print('Best model is Random Forest Classifier')
    name = 'Random_Forest_Classifier'
    model = GSCV_RFC
else:
    print('Best model is Logistic Regression')
    name = 'Logistic_Regression'
    model = GSCV_LR
import pickle
with open('Model_' + name + '.pkl', 'wb') as file:
    pickle.dump(model, file)
```

```
Best model is Random Forest Classifier
```

```
import pandas as pd
import numpy as np

data = pd.read_csv('Dataset UTS_Dset 2025.csv')
data.shape
```

```
(4)
```

```
data.head()
```

	nipannumber	numberofhouses	hoped	hoped	floors	cityside	citycentrage	numpossesses	male	bonedoft	bonetempratucher	bonetest	ahlc	garage	hasotongmasen	hasgantoman	ptlu	category
0	75321	57	no	yes	19	yes	43	3	2020	-ve	6111	9201	no	2	yes	9	5174411	Middle
1	53712	58	no	yes	19	34457	6	3	2021	-ve	2937.8852	135	no	2	yes	9	5174411	Middle
2	80655	60	no	yes	19	34457	6	3	2020	-ve	1535.4198	9201	no	2	yes	9	5174411	Middle
3	57432	3	no	no	21	9247	8	1	2010	+ve	295	630.6782	807	no	5	9	5154953.5	Middle
4	96470	74	yes	no	21	92029	6	2	2015	+ve	1414.772	770	yes	9	9012238.5			
5	75321	57	no	yes	19	34457	6	3	2020	-ve	2871.7717	240	yes	9	7988668.5			
6	75081	60	yes	no	67	6517	5	3	2005	+ve	4878	281	384	5	7507.5214			
7	75321	57	no	yes	19	6777	7	3	2014	-ve	2854.126	726	no	5	9429.8213			
8	92383	52	no	no	78	7382	1	2	2000	-ve	7507	9056	892	yes	5	9240.0443		
9	75321	57	no	yes	19	5982	1	3	2020	+ve	875	5201	1	yes	5	9174.0443		
10	76455	47	yes	no	9	90254	2	2	2008	+ve	2860	2125	982	no	1	7933.038.8		
11	75321	57	no	yes	19	6777	7	3	2014	-ve	3429	4125	1	yes	2	8774.0443		
12	66083	73	yes	no	6	5903	6	2	2021	-ve	1473	756	237	yes	1	6273.6411		
13	75321	57	no	yes	19	5903	6	3	2020	-ve	2173	9556	916	yes	2	6400.6411		
14	76201	39	yes	no	32	5463	5	1	2016	+ve	8732	4207	533	no	7	7574.9763		
15	75321	57	no	yes	19	5463	5	3	2021	-ve	1326	3171	1	yes	7	9273.9763		
16	59800	47	no	yes	27	44015	6	3	2021	-ve	5271.134	364	no	6	5994.6217			
17	75321	57	no	yes	52	44015	6	3	2020	-ve	5276	5036	213	2	5453.6217			
18	75321	52	yes	no	28	95678	4	1	1992	-ve	6490	6919	680	1	7203.5712			
19	75321	57	no	yes	28	5192	4	3	2011	-ve	2871	7408	223	1	5460.5812			
20	63053	5	yes	no	28	4572	1	1	1997	-ve	8414	6270	939	1	6273.0712			
21	75321	57	no	yes	51	95255	1	3	1990	-ve	3651	2400	1	1	9917.0712			
22	93876	60	no	yes	55	5484	2	1	1999	-ve	4096	1891	494	2	5298.0913			
23	75321	57	no	yes	55	6395	2	3	2006	-ve	3429	4125	1	2	5124.0913			
24	89760	40	yes	no	17	7380	5	1	1993	-ve	2495	139	864	1	8980.511			
25	97030	60	no	yes	21	8217	1	2	2007	-ve	768	4798	132	1	10027.511			
26	68671	49	no	no	89	5743	10	1	1999	-ve	5092	8356	740	1	7270.664.4			
27	75314	43	no	no	38	4995	10	1	2018	-ve	1492	5607	625	1	5124.711			
28	75321	57	no	yes	19	5380	1	3	2014	-ve	8364	7226	226	1	7144.6711			
29	71591	20	yes	no	58	46834	7	1	1996	-ve	1476	3715	340	1	7150.9818			
30	75321	57	no	yes	19	45626	7	1	1990	-ve	4325	7808	734	1	6152.9818			
31	81534	73	yes	no	87	2240	5	1	2001	-ve	5091	8974	715	1	6198.571			
32	75321	57	no	yes	72	2274	5	3	1993	-ve	2668	4689	786	1	6414.541			
33	51404	64	no	yes	23	76754	10	1	2012	+ve	2080	9575	753	1	7148.2262			
34	78992	52	no	yes	76	21458	8	1	2015	+ve	7135	9212	974	1	7909.964.1			
35	81870	60	no	yes	105	58048	1	3	2020	-ve	3632	1960	723	1	8198.0818			
36	97030	60	no	yes	21	8217	1	2	2007	-ve	768	4798	132	1	10027.511			
37	72096	9	no	yes	67	9158	2	2	2014	+ve	5092	8356	740	1	7270.664.4			
38	75321	57	no	yes	8	849	1	3	1999	-ve	1492	5607	625	1	5124.711			
39	53715	49	no	yes	52	2423	4	2	2006	-ve	8034	6588	290	1	5260.0344			
40	75321	57	no	yes	91	68198	1	3	1999	-ve	3477	3130	342	1	7144.6711			
41	65151	87	yes	yes	1	60197	7	1	1993	-ve	1276	8119	840	1	6152.9711			
42	97490	60	no	yes	1	83915	8	1	2015	+ve	5480	3041	766	1	6198.571			
43	57160	15	yes	yes	43	45796	8	3	2002	-ve	4074	4871	836	2	5154.0212			
44	75321	57	no	yes	8	5903	8	3	2020	-ve	7300	4238	986	2	5100.9211			
45	87836	68	no	no	92	9143	3	1	2011	-ve	6293	9382	724	2	8198.0727			
46	75321	57	no	yes	71	5595	1	3	2010	-ve	4034	2877	707	2	8198.0727			
47	92867	45	no	yes	7	97173	4	1	1993	-ve	292	744	475	1	6297.7713			
48	75321	57	no	yes	19	46755	4	3	2016	-ve	9821	7176	538	1	6400.7713			
49	84364	76	no	no	19	5723	5	1	1998	-ve	4508	4857	480	1	8423.5543			

```
data = data.drop('price', axis=1)
```

```
data.describe()
```

	nipannumber	numberofhouses	floors	cityside	citycentrage	numpossesses	male	bonedoft	ahlc	garage	hasotongmasen
count	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000
mean	69670.7232	50.20480	52.37426	32223.48570	5.51938	2.52700	2005.48000	5021.50968	5280.57260	5.51120	4.99460
std	19774.37525	20.81606	28.68077	29006.67799	2.87324	1.20444	9.5800	2876.70545	2894.3221	262.05017	3.17947
min	49.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1990.00000	0.00000	1.00000	100.00000	0.00000
25%	23296.50000	25.00000	25.00000	24091.75000	3.00000	1.00000	1997.00000	2534.75000	2512.00000	317.75000	2.00000
50%	75321.00000	50.00000	50.00000	50000.00000	5.00000	3.00000	2005.50000	5001.50000	5240.00000	534.00000	5.00000
75%	94609.75000	75.00000	75.00000	75681.25000	8.00000	4.00000	2014.00000	7511.25000	7540.00000	771.25000	8.00000
max	99999.00000	100.00000	100.00000	99983.00000	10.00000	5.00000	2021.00000	10000.00000	10000.00000	1000.00000	10.00000

```
data.info()
```

```
<class> pandas.core.frame.DataFrame>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 17 columns):
 #  Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0  nipannumber  10000 non-null  int64
 1  numberofhouses  10000 non-null  int64
 2  hoped  10000 non-null  object
 3  hoped  10000 non-null  object
 4  floors  10000 non-null  int64
 5  cityside  10000 non-null  int64
 6  citycentrage  10000 non-null  int64
 7  numpossesses  10000 non-null  int64
 8  male  10000 non-null  int64
 9  bonedoft  10000 non-null  object
 10  bonetempratucher  10000 non-null  object
 11  bonetest  10000 non-null  object
 12  ahlc  10000 non-null  int64
 13  garage  10000 non-null  int64
 14  hasotongmasen  10000 non-null  int64
 15  hasgantoman  10000 non-null  int64
 16  ptlu  10000 non-null  int64
dtype: int64, object: 10, int64: 7
memory usage: 1.3 MB
```

```
data.head(5)
```

	nipannumber	numberofhouses	hoped	hoped	floors	cityside	citycentrage	numpossesses	male	bonedoft	bonetempratucher	bonetest	ahlc	garage	hasotongmasen	hasgantoman	ptlu	category
count	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000
unique	NaN	NaN	2	2	NaN	NaN	NaN	NaN	NaN	NaN	2	2	NaN	NaN	2	2	NaN	NaN
freq	NaN	NaN	5997	5012	NaN	NaN	NaN	NaN	NaN	NaN	no	yes	NaN	NaN	yes	yes	NaN	NaN
mean	69670.7232	50.20480	NaN	NaN	52.37426	32223.48570	5.51938	2.52700	2005.48000	5021.50968	NaN	NaN	5280.57260	5.51120	NaN	NaN	5030	NaN
std	19774.37525	20.81606	NaN	NaN	28.68077	29006.67799	2.87324	1.20444	9.5800	2876.70545	NaN	NaN	2894.3221	262.05017	NaN	NaN	3.17947	NaN
min	49.00000	1.00000	NaN	NaN	1.00000	1.00000	1.00000	1.00000	1990.00000	0.00000	NaN	NaN	1.00000	100.00000	NaN	NaN	0.00000	NaN
25%	23296.50000	25.00000	NaN	NaN	25.00000	24091.75000	3.00000	1.00000	1997.00000	2534.75000	NaN	NaN	2512.00000	317.75000	NaN	NaN	2.00000	NaN
50%	75321.00000	50.00000	NaN	NaN	50.00000	50000.00000	5.00000	3.00000	2005.50000	5001.50000	NaN	NaN	5240.00000	534.00000	NaN	NaN	5.00000	NaN
75%	94609.75000	75.00000	NaN	NaN	75.00000	75681.25000	8.00000	4.00000	2014.00000	7511.25000	NaN	NaN	7540.00000	771.25000	NaN	NaN	8.00000	NaN
max	99999.00000	100.00000	NaN	NaN	100.00000	99983.00000	10.00000	5.00000	2021.00000	10000.00000	NaN	NaN	10000.00000	1000.00000	NaN	NaN	10.00000	NaN

```
print("Data Null 1\n",data.isnull().sum())
print("Data Null 2\n",data.isna().sum())
print("Data Dup\n",data.duplicated.sum())
```

```
Data Null 1
 nipannumber  0
numberofhouses  0
hoped  0
hoped  0
floors  0
cityside  0
citycentrage  0
numpossesses  0
male  0
bonedoft  0
bonetempratucher  0
bonetest  0
ahlc  0
garage  0
hasotongmasen  0
hasgantoman  0
ptlu  0
category  0
dtype: int64

Data Kosong
False

Data Nan
nipannumber  0
numberofhouses  0
hoped  0
hoped  0
floors  0
cityside  0
citycentrage  0
numpossesses  0
male  0
bonedoft  0
bonetempratucher  0
bonetest  0
ahlc  0
garage  0
hasotongmasen  0
hasgantoman  0
ptlu  0
dtype: int64

Data Dup
False
```

```
data = data.drop('nipannumber', axis=1)
plt.pie(y_count,labels=y_count)
plt.show()
```

```
from sklearn.model_selection import train_test_split
print("Sebelum data dang duplicate",data.shape)
data = data.drop_duplicates()
print("Setelah drop data duplicate",data.shape)

#Splitting
X = data.drop('category',axis=1)
y = data['category']
#X = data[['floors']]
# X = data[['floors','cityside']]
#y = data['price']

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25,random_state=0)

print(X_train.shape)
print(X_test.shape)
```

```
Sebelum drop data duplicate (10000, 17)
Setelah drop data duplicate (10000, 17)

(7500, 16)
(2500, 16)
```

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer
#https://stackoverflow.com/questions/54926615/get-encoding-across-multiple-columns-in-scikit-learn

from sklearn.preprocessing import LabelEncoder, StandardScaler, RobustScaler

transformer = make_column_transformer(
    (OneHotEncoder(),['ahlc','category','possesses']),
    )

X_train_encoded = transform.fit_transform(X_train)
X_test_encoded = transform.transform(X_test)

pd_train_enc = pd.DataFrame(X_train_encoded, columns=transform.get_feature_names_out())
pd_test_enc = pd.DataFrame(X_test_encoded, columns=transform.get_feature_names_out())

pd_train_enc.head()
pd_test_enc.head()
```

```
[nipannumber_hoped_no  nipannumber_hoped_yes  nipannumber_hoped_no  nipannumber_hoped_yes  nipannumber_bonedoft_-ve  nipannumber_bonedoft_+ve  nipannumber_bonetest_-ve  nipannumber_bonetest_+ve  nipannumber_hasotongmasen_no  nipannumber_hasotongmasen_yes  nipannumber_hasgantoman_no  nipannumber_hasgantoman_yes
0                      1.0                    0.0                    0.0                    1.0                       0.0                       1.0                       0.0                       1.0                           0.0                            0.0                         0.0                         0.0
1                      1.0                    0.0                    1.0                    0.0                       1.0                       0.0                       1.0                       0.0                           0.0                            0.0                         0.0                         0.0
2                      1.0                    0.0                    1.0                    0.0                       1.0                       0.0                       1.0                       0.0                           0.0                            0.0                         0.0                         0.0
3                      0.0                    1.0                    1.0                    0.0                       0.0                       1.0                       1.0                       0.0                           0.0                            0.0                         0.0                         0.0
4                      0.0                    1.0                    0.0                    1.0                       0.0                       1.0                       0.0                       1.0                           0.0                            0.0                         0.0                         0.0
5                      1.0                    0.0                    1.0                    0.0                       1.0                       0.0                       1.0                       0.0                           0.0                            0.0                         0.0                         0.0
6                      1.0                    0.0                    0.0                    1.0                       0.0                       1.0                       0.0                       1.0                           0.0                            0.0                         0.0                         0.0
7                      1.0                    0.0                    1.0                    0.0                       1.0                       0.0                       1.0                       0.0                           0.0                            0.0                         0.0                         0.0
8                      1.0                    0.0                    1.0                    0.0                       1.0                       0.0                       1.0                       0.0                           0.0                            0.0                         0.0                         0.0
9                      1.0                    0.0                    0.0                    1.0                       0.0                       1.0                       0.0                       1.0                           0.0                            0.0                         0.0                         0.0

[10 rows x 21 columns]
```

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.feature_selection import SelectKBest, SelectPercentile
from sklearn.feature_selection import GenericUnivariateSelect
from sklearn.svm import SVC
from sklearn.pipelines import Pipeline
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import classification_report,confusion_matrix,ConfusionMatrixDisp

y_pipe_svm = Pipeline(steps=[
    ('scaler', MinMaxScaler()),
    ('feat_select', SelectKBest()),
    ('clf', SVC(class_weight='balanced'))
])

#hyp0 parameter grid which step feature selection the classifier
params_grid_pipe = {
    'scaler': [MinMaxScaler(),
    StandardScaler(), RobustScaler()],
    'feat_select__score_func': [f_classif],
    'feat_select__k': [5,10],
    'clf__C': [0.1,1],
    'clf__kernel': ['poly', 'sigmoid', 'rbf'],
    # 'clf__gamma': [0.01,0.1,1],
    # 'clf__degree': [2,3,4],
    # 'clf__gamma': [scale, auto],
    # 'clf__coef0': [0.0,0.1,0.5],
    # 'clf__shrinking': [True, False],
    # 'clf__tol': [1e-3,1e-4,1e-5],
    # 'clf__class_weight': [None, balanced],
    # 'clf__max_iter': [-1,100,1000],
    # 'clf__decision_function_shape': [ovr, ovo],
    # 'clf__random_state': [0,42,2025],
    # 'clf__verbose': [True, False],
    # 'clf__probability': [True, False],
    # 'clf__cache_size': [200,500,1000
```

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

data = pd.read_csv('Dataset UTS_Gasal 2425.csv')
data.head(55)
```

Out[1]:

	squaremeters	numberofrooms	hasyard	haspool	floors	citycode	citypartrange	num
0	75523	3	no	yes	63	9373	3	
1	55712	58	no	yes	19	34457	6	
2	86929	100	yes	no	11	98155	3	
3	51522	3	no	no	61	9047	8	
4	96470	74	yes	no	21	92029	4	
5	79770	3	no	yes	69	54812	10	
6	75985	60	yes	no	67	6517	6	
7	64169	88	no	yes	6	61711	3	
8	92383	12	no	no	78	71982	3	
9	95121	46	no	yes	3	9382	7	
10	76485	47	yes	no	9	90254	2	
11	87060	27	no	yes	91	51803	8	
12	66683	19	yes	yes	6	50801	6	
13	84559	29	no	yes	69	53057	7	
14	76091	38	yes	no	32	59451	5	
15	92696	49	yes	no	38	74381	9	
16	59800	47	no	yes	27	44815	6	
17	54836	25	no	yes	53	64601	10	
18	70021	52	yes	no	28	95678	4	
19	54368	11	yes	yes	20	55761	3	
20	63053	6	yes	yes	28	45312	3	
21	64393	8	no	no	51	95335	4	
22	93876	60	no	yes	70	5484	2	
23	84016	15	yes	no	55	63595	1	
24	89768	48	yes	yes	17	71000	6	
25	58478	5	no	yes	35	5898	6	
26	66621	48	no	no	89	52165	10	
27	73314	43	no	yes	38	49895	10	
28	59972	28	no	yes	18	32083	9	
29	71591	20	yes	no	58	46834	7	

	squaremeters	numberofrooms	hasyard	haspool	floors	citycode	citypartrange	num
30	67311	67	no	no	10	45626	3	
31	61534	73	yes	no	97	22943	9	
32	84091	50	no	yes	72	22718	7	
33	51434	64	no	no	23	79754	10	
34	78960	55	no	yes	76	23408	8	
35	81870	60	no	yes	100	58048	3	
36	91559	36	no	yes	21	82521	6	
37	72098	9	no	yes	67	91168	2	
38	55232	23	no	no	8	849	8	
39	53735	49	no	yes	92	2423	4	
40	71397	71	no	no	93	68199	3	
41	65151	81	yes	yes	3	66191	7	
42	61484	91	yes	no	94	87015	8	
43	57160	15	yes	yes	43	40786	8	
44	55933	15	no	no	97	5800	9	
45	81936	68	no	no	92	6143	3	
46	99683	12	yes	no	77	18300	5	
47	62887	45	no	yes	7	91125	4	
48	73062	34	yes	yes	38	44770	4	
49	84284	76	no	no	39	55723	5	
50	77046	69	yes	no	85	43517	4	
51	58855	1	no	no	66	20127	3	
52	69165	66	yes	no	74	51371	4	
53	57915	25	no	yes	84	87517	10	
54	66279	47	yes	no	9	21934	3	

In [2]:
```
data = data.drop(['category'], axis=1)
```

In [3]:
```
data.describe()
```

Out[3]:

	squaremeters	numberofrooms	floors	citycode	citypartrange	numprev
count	10000.00000	10000.000000	10000.000000	10000.000000	10000.000000	10000
mean	49870.13120	50.358400	50.276300	50225.486100	5.510100	5
std	28774.37535	28.816696	28.889171	29006.675799	2.872024	2
min	89.00000	1.000000	1.000000	3.000000	1.000000	1
25%	25098.50000	25.000000	25.000000	24693.750000	3.000000	3
50%	50105.50000	50.000000	50.000000	50693.000000	5.000000	5
75%	74609.75000	75.000000	76.000000	75683.250000	8.000000	8
max	99999.00000	100.000000	100.000000	99953.000000	10.000000	10

In [4]:
```
data.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
```

In [5]:
```
data['garage'].value_counts()
```

Out[5]:
```
garage
253    24
955    21
866    20
745    20
968    20
       ..
193     4
887     3
483     3
589     2
282     2
Name: count, Length: 901, dtype: int64
```

In [6]:
```
data['price'].describe()
```

Out[6]:
```
count    1.000000e+04
mean     4.993448e+06
std      2.877424e+06
min      1.031350e+04
25%      2.516402e+06
50%      5.016180e+06
75%      7.469092e+06
max      1.000677e+07
Name: price, dtype: float64
```

In [7]:
```
data.describe(include='all')
```

Out[7]:

	squaremeters	numberofrooms	hasyard	haspool	floors	citycode	city
count	10000.00000	10000.000000	10000	10000	10000.000000	10000.000000	10(
unique	NaN	NaN	2	2	NaN	NaN	
top	NaN	NaN	yes	no	NaN	NaN	
freq	NaN	NaN	5087	5032	NaN	NaN	
mean	49870.13120	50.358400	NaN	NaN	50.276300	50225.486100	
std	28774.37535	28.816696	NaN	NaN	28.889171	29006.675799	
min	89.00000	1.000000	NaN	NaN	1.000000	3.000000	
25%	25098.50000	25.000000	NaN	NaN	25.000000	24693.750000	
50%	50105.50000	50.000000	NaN	NaN	50.000000	50693.000000	
75%	74609.75000	75.000000	NaN	NaN	76.000000	75683.250000	
max	99999.00000	100.000000	NaN	NaN	100.000000	99953.000000	

In [8]:
```
print("Data Null \n",data.isnull().sum())
print("\nData Kosong \n",data.empty)
print("\nData Nan \n",data.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
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
```

In [9]:

```
import matplotlib.pyplot as plt

data['price'].plot(kind='box')
plt.gca().invert_yaxis()
plt.show()
```

In [10]:
```
from sklearn.model_selection import train_test_split
print("Sebelum drop data duplicate ",data.shape)
data2= data.drop_duplicates(keep='last')
print("Sesudah drop data duplicate ",data2.shape)

#klasifikasi
# X = data2.drop(['category'],axis=1)
# y = data2['category']
#regresi
X = data2.drop(['price'],axis=1)
y = data2['price']

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25,random_state

print(X_train.shape)
print(X_test.shape)
```

```
Sebelum drop data duplicate  (10000, 17)
Sesudah drop data duplicate  (10000, 17)
(7500, 16)
(2500, 16)
```

In [11]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer
#https://stackoverflow.com/questions/24458645/label-encoding-across-multiple-column

kolom_kategori=['hasyard','haspool','isnewbuilt','hasstormprotector','hasstorageroo

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

```
)

X_train_encoded = transform.fit_transform(X_train)
X_test_encoded = transform.transform(X_test)


df_train_enc = pd.DataFrame(X_train_encoded, columns=transform.get_feature_names_ou
df_test_enc = pd.DataFrame(X_test_encoded, columns=transform.get_feature_names_out(

pd.set_option('display.max_columns', None)

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

Out[11]:

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

In [12]:

```
from sklearn.linear_model import Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.feature_selection import SelectKBest, f_regression , SelectPercentile
from sklearn.metrics import mean_absolute_error, mean_squared_error



pipe_Lasso = Pipeline([
    ('scale',StandardScaler()),
    ('feat_select',SelectKBest(score_func= f_regression)),
    ('reg',Lasso(max_iter=10000))
])


param_grid_lasso = [
    {
        'scale': [MinMaxScaler()],
```

```
        'reg__alpha': [0.01, 0.1, 1, 10, 100],
        'feat_select__k': np.arange(7, 22),
    },
    {
        'scale': [MinMaxScaler()],
        'reg__alpha': [0.01, 0.1, 1, 10, 100],
        'feat_select':[SelectPercentile(score_func=f_regression)],
        'feat_select__percentile': np.arange(10, 100, 5),
    },
    {
        'scale': [StandardScaler()],
        'reg__alpha': [0.01, 0.1, 1, 10, 100],
        'feat_select__k': np.arange(7, 22),
    },
    {
        'scale': [StandardScaler()],
        'reg__alpha': [0.01, 0.1, 1, 10, 100],
        'feat_select':[SelectPercentile(score_func=f_regression)],
        'feat_select__percentile': np.arange(10, 100, 5),
    },
]

kf = KFold(n_splits=7, shuffle=True, random_state=91)
GSCV_LS = GridSearchCV(pipe_Lasso, param_grid=param_grid_lasso, cv=kf, n_jobs=-1, s

GSCV_LS.fit(X_train_encoded, y_train)

print("Best Model:{}".format(GSCV_LS.best_estimator_))
print("Lasso Best parameters:{}".format(GSCV_LS.best_params_))
print("Koeffisien Ridge:{}".format(GSCV_LS.best_estimator_.named_steps['reg'].coef_
print("Intercept/bias:{}".format(GSCV_LS.best_estimator_.named_steps['reg'].interce


Lasso_predict = GSCV_LS.predict(X_test_encoded)

mse_Lasso = mean_squared_error(y_test, Lasso_predict)
mae_Lasso = mean_absolute_error(y_test, 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)))
```

```
Fitting 7 folds for each of 330 candidates, totalling 2310 fits
Best Model:Pipeline(steps=[('scale', StandardScaler()),
                ('feat_select',
                 SelectKBest(k=20,
                             score_func=<function f_regression at 0x00000174E9DE9300
>)),
                ('reg', Lasso(alpha=10, max_iter=10000))])
Lasso Best parameters:{'feat_select__k': 20, 'reg__alpha': 10, 'scale': StandardScal
er()}
Koeffisien Ridge:[-1.49684690e+03  2.52233197e-13 -1.46341272e+03  1.22236088e-13
  5.75209472e+01 -0.00000000e+00 -5.36145983e+01  4.58482342e-11
 -4.60002311e+00  0.00000000e+00  2.86870271e+06  0.00000000e+00
  1.57083608e+03  1.17784067e+02  3.86789293e+00 -2.12464477e+01
 -4.30845902e+00 -2.19999718e+01  3.24660860e+01 -1.85097035e+01]
Intercept/bias:4967834.411573334
Lasso Mean Squared Error (MSE): 3546474.6122352486
Lasso Mean Absolute Error (MAE): 1468.1163288008136
Lasso Root Mean Squared Error: 1883.2085949876207
```

In [13]:

```
import pandas as pd


df_results = pd.DataFrame(y_test, columns=['price'])


df_results['Lasso Prediction'] = Lasso_predict


df_results['Selisih_Pricew_LS'] = df_results['Lasso Prediction'] - df_results['pric


pd.set_option('display.float_format', '{:,.2f}'.format)

# Menampilkan 5 baris pertama dari DataFrame
df_results.head()
```

Out[13]:

	price	Lasso Prediction	Selisih_Pricew_LS
9600	2,327,859.80	2,327,235.93	-623.87
2889	8,694,297.10	8,694,579.69	282.59
2694	8,221,240.90	8,224,089.91	2,849.01
4775	3,252,838.40	3,253,656.58	818.18
5567	463,922.20	468,324.28	4,402.08

In [14]:

```
from sklearn.metrics import r2_score
r2 = r2_score(y_test, Lasso_predict)
print("R² score:", r2)
```

```
R² score: 0.9999995787468514
```

In [15]:

```
import matplotlib.pyplot as plt
import numpy as np
```

```
plt.figure(figsize=(12, 10))
#intinya titik biru di ambil dari sumbu x(y_test) dan sumbu y(Lasso_predict) jadi k
plt.scatter(y_test, Lasso_predict, color='blue', alpha=0.6, edgecolor='k', label='P


plt.xlabel("Harga Aktual (Actual Price)", fontsize=14)
plt.ylabel("Harga Prediksi (Predicted Price)", fontsize=14)
plt.title("Perbandingan Harga Aktual vs Harga Prediksi Lasso", fontsize=16)

#harga actual jadi x dan y sama sama dari y_test buat garis diagonal -- merah
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2, l


plt.grid(True, linestyle='--', alpha=0.6)
plt.legend(loc='upper left')


plt.show()
```

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

Out[16]:

	price	Lasso Prediction	Selisih_Pricew_LS
count	2,500.00	2,500.00	2,500.00
mean	5,070,286.87	5,070,279.87	-7.00
std	2,902,109.03	2,902,130.26	1,883.57
min	10,313.50	13,434.79	-6,912.16
25%	2,572,316.52	2,572,174.93	-1,221.14
50%	5,150,484.40	5,151,235.68	8.07
75%	7,620,090.12	7,620,599.20	1,202.71
max	10,002,945.00	10,001,302.69	5,969.00

In [18]:

```
pipe_RFR = Pipeline(steps=[
    ('scale', MinMaxScaler()),
    ('feat_select', SelectKBest(score_func=f_regression)),
    ('reg', RandomForestRegressor())
])


param_RFR = [
   {
        'scale': [MinMaxScaler()],
        'feat_select__k': np.arange(5, 20),
        'reg__max_depth': [15, 20], #menentukan max depth dari random forest dengan
        'reg__n_estimators': [100, 200, 300] # ini buat jumlah pohon kalo ga salah
    },
    {
        'scale': [StandardScaler()],
        'feat_select': [SelectPercentile()],
        'feat_select__percentile': np.arange(10, 100, 10),
        'reg__max_depth': [15, 20],
        'reg__n_estimators': [100, 200, 300]
    },
    {
        'scale': [StandardScaler()],
        'feat_select__k': np.arange(5, 20),
        'reg__max_depth': [None, 10, 15, 20],
        'reg__n_estimators': [100, 200, 300]
    },
    {
        'scale': [MinMaxScaler()],
        'feat_select': [SelectPercentile()],
        'feat_select__percentile': np.arange(10, 100, 10),
        'reg__max_depth': [15, 20],
        'reg__n_estimators': [100, 200, 300]
    }
]
```

```
kf = KFold(n_splits=7, shuffle=True, random_state=91)
GSCV_RFR = GridSearchCV(pipe_RFR, param_grid=param_RFR, cv=kf, n_jobs=-1, scoring='

GSCV_RFR.fit(X_train_encoded, y_train)

print("Best Model:{}".format(GSCV_RFR.best_estimator_))
print("RFR Best parameters:{}".format(GSCV_RFR.best_params_))
print("Feature Importances: {}".format(GSCV_RFR.best_estimator_.named_steps['reg'].


RFR_predict = GSCV_RFR.predict(X_test_encoded)

mse_RFR = mean_squared_error(y_test, RFR_predict)
mae_RFR = mean_absolute_error(y_test, RFR_predict)

print("RFR Mean Squared Error (MSE): {}".format (mse_RFR))
print("RFR Mean Absolute Error (MAE): {}".format (mae_RFR ))
print("RFR Root Mean Squared Error: {}".format(np.sqrt(mse_RFR)))
```

```
Fitting 7 folds for each of 378 candidates, totalling 2646 fits
```

```
c:\Users\Acer\anaconda3\Lib\site-packages\sklearn\feature_selection\_univariate_sele
ction.py:109: RuntimeWarning: invalid value encountered in divide
  msw = sswn / float(dfwn)
```

```
Best Model:Pipeline(steps=[('scale', StandardScaler()),
                ('feat_select', SelectPercentile(percentile=70)),
                ('reg', RandomForestRegressor(max_depth=20, n_estimators=300))])
RFR Best parameters:{'feat_select': SelectPercentile(), 'feat_select__percentile': 7
0, 'reg__max_depth': 20, 'reg__n_estimators': 300, 'scale': StandardScaler()}
Feature Importances: [3.59480864e-08 3.45672741e-08 3.58128119e-08 3.48684665e-08
 2.90102344e-08 2.95008943e-08 2.52089037e-08 2.55855063e-08
 2.75057344e-08 2.73446743e-08 9.99999067e-01 1.90758152e-07
 2.33659896e-07 2.02951680e-07]
RFR Mean Squared Error (MSE): 13870793.803145923
RFR Mean Absolute Error (MAE): 3001.714381066518
RFR Root Mean Squared Error: 3724.3514607439943
```

In [20]:

```
import pandas as pd


df_results = pd.DataFrame(y_test, columns=['price'])


df_results['RFR Prediction'] = RFR_predict


df_results['Selisih_Pricew_RFR'] = df_results['RFR Prediction'] - df_results['price


pd.set_option('display.float_format', '{:,.2f}'.format)

# Menampilkan 5 baris pertama dari DataFrame
df_results.head()
```

Out[20]:

	price	RFR Prediction	Selisih_Pricew_RFR
9600	2,327,859.80	2,330,119.59	2,259.79
2889	8,694,297.10	8,696,287.21	1,990.11
2694	8,221,240.90	8,227,565.44	6,324.54
4775	3,252,838.40	3,253,046.88	208.48
5567	463,922.20	460,581.68	-3,340.52

In [22]:
```
from sklearn.metrics import r2_score
r2 = r2_score(y_test, RFR_predict)
print("R² score:", r2)
```

```
R² score: 0.9999983524157926
```

In [24]:
```
import matplotlib.pyplot as plt
import numpy as np


plt.figure(figsize=(12, 10))
plt.scatter(y_test, RFR_predict, color='blue', alpha=0.6, edgecolor='k', label='Pre


plt.xlabel("Harga Aktual (Actual Price)", fontsize=14)
plt.ylabel("Harga Prediksi (Predicted Price)", fontsize=14)
plt.title("Perbandingan Harga Aktual vs Harga Prediksi RFR", fontsize=16)


plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2, l


plt.grid(True, linestyle='--', alpha=0.6)
plt.legend(loc='upper left')


plt.show()
```

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

Out[25]:

	price	RFR Prediction	Selisih_Pricew_RFR
count	2,500.00	2,500.00	2,500.00
mean	5,070,286.87	5,070,271.29	-15.58
std	2,902,109.03	2,902,242.82	3,725.06
min	10,313.50	18,414.73	-13,557.23
25%	2,572,316.52	2,572,850.42	-2,494.15
50%	5,150,484.40	5,152,768.07	109.15
75%	7,620,090.12	7,620,276.02	2,706.96
max	10,002,945.00	9,999,950.22	10,113.05

In [29]:
```
df_results = pd.DataFrame(y_test, columns=['price'])

df_results['Lasso Prediction'] = Lasso_predict
df_results['Selisih_IPK_LR'] = df_results['price'] - df_results['Lasso Prediction']
df_results['RFR Prediction'] = RFR_predict
```

```
df_results['Selisih_IPK_RFR'] = df_results['price'] - df_results['RFR Prediction']

df_results.head()
```

Out[29]:

	price	Lasso Prediction	Selisih_IPK_LR	RFR Prediction	Selisih_IPK_RFR
9600	2,327,859.80	2,327,235.93	623.87	2,330,119.59	-2,259.79
2889	8,694,297.10	8,694,579.69	-282.59	8,696,287.21	-1,990.11
2694	8,221,240.90	8,224,089.91	-2,849.01	8,227,565.44	-6,324.54
4775	3,252,838.40	3,253,656.58	-818.18	3,253,046.88	-208.48
5567	463,922.20	468,324.28	-4,402.08	460,581.68	3,340.52

In [31]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(50, 20))

data_len = range(len(y_test))

plt.scatter(data_len, df_results['price'], label='Actual', color='purple')

plt.plot(data_len, df_results['Lasso Prediction'], label='Lasso Prediction', color=

plt.plot(data_len, df_results['RFR Prediction'], label='SVR Prediction', color='red

plt.legend(loc='upper left', prop={'size': 14})
plt.show
```

Out[31]: <function matplotlib.pyplot.show(close=None, block=None)>

In [36]:

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



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 Pred
```

```
lasso_feature_count = GSCV_LS.best_params_['feat_select__k']

mae_RFR= mean_absolute_error(df_results['price'], df_results['RFR Prediction'])

rmse_RFR = np.sqrt(mean_squared_error(df_results['price'], df_results['RFR Predicti

RFR_feature_count = GSCV_RFR.best_params_['feat_select__percentile']




print(f"Lasso MAE: {mae_Lasso}, Lasso RMSE: {rmse_Lasso}, Lasso Feature Count(perce
print(f"RFR MAE: {mae_RFR}, RFR RMSE: {rmse_RFR}, RFR Feature Count(percentile): {R


r2 = r2_score(y_test, Lasso_predict)
print("R² Lasso score:", r2)

r2 = r2_score(y_test, RFR_predict)
print("R² RFRC score:", r2)
```

```
Lasso MAE: 1468.1163288008136, Lasso RMSE: 1883.2085949876207, Lasso Feature Count(p
ercentile): 20
RFR MAE: 3001.714381066518, RFR RMSE: 3724.3514607439943, RFR Feature Count(percenti
le): 70
R² Lasso score: 0.9999995787468514
R² RFRC score: 0.9999983524157926
```

In [38]:

```
#KAYANE PILIH BERDASARKAN YANG PALING BAGUS DENGAN PRIOIRITAS MAE DAN RMSE YANG PAL


print('Best model is Lassso with MAE: ',mae_Lasso, 'and RMSE: ',rmse_Lasso)
import pickle
with open('Model_Lasso.pkl', 'wb') as file:
    pickle.dump(GSCV_LS, file)
```

```
Best model is Lassso with MAE:  1468.1163288008136 and RMSE:  1883.2085949876207
```

Frans Daniel Rajagukguk – 220711826

Angello Khara Sitanggang – 220711833

Davin Gilbert Natanael – 220711841

Emanuel Enrico Anindya Wibawa – 220711890

Archipera Rari Dyaksa - 220711891

```
import streamlit as st

import pandas as pd

import pickle

import os

import streamlit_option_menu

from streamlit_option_menu import option_menu


# Navigasi sidebar dengan warna dan style custom

with st.sidebar:

    st.markdown("<h2 style='color:#4C9A2A;'>🧠 Streamlit UTS ML 24/25</h2>",
unsafe_allow_html=True)

    selected = option_menu('Pilih Menu',

                           ['Klasifikasi', 'Regresi', 'Catatan'],

                           icons=['archive', 'journal-bookmark', 'exclamation-circle'],

                           menu_icon="cast", default_index=0,

                           styles={

                               "container": {"background-color": "#2c2c2c"},

                               "icon": {"color": "#ffffff", "font-size": "20px"},

                               "nav-link": {"font-size": "18px", "text-align": "left", "margin":"0px", "--hover-color":
"grey;"},

                               "nav-link-selected": {"background-color": "#4C9A2A"},

                           })


# Halaman Klasifikasi

if selected == 'Klasifikasi':

    st.title('🏷️ Klasifikasi Properti')
```

```
st.write('Untuk Inputan File dataset (csv) bisa menggunakan st.file_uploader')
file = st.file_uploader('Masukkan File Dataset Anda', type=["csv", "txt"])


def validate_file(data):
    keywords = ["luxury", "basic", "middle"]
    data_str = data.astype(str)  # Convert all data to string for searching
    if not data_str.apply(lambda x: x.str.contains('|'.join(keywords), case=False)).any().any():
        raise ValueError("The dataset does not contain the required categories: 'luxury', 'basic', 'middle'.")
if file is not None:
    try:
        input_data = pd.read_csv(file)




        validate_file(input_data)
        st.markdown("<h3 style='text-align: center; color: #0073e6;'>📊 Data yang Diupload:</h3>", unsafe_allow_html=True)
        st.dataframe(input_data)
    except ValueError as e:
        st.markdown("<h3 style='text-align : center' font-color: red> !! ⚠️ An error occurred while reading the file ⚠️ !! </h3>", unsafe_allow_html=True)
# Lokasi model


model_path = 'BestModel_CLF_GBC_Transformer.pkl'


if os.path.exists(model_path):
    with open(model_path, 'rb') as f:
       loaded_model = pickle.load(f)


    rf_model = loaded_model
```

```
# Bagian Input
st.markdown("### Masukkan Fitur Properti:")
col1, col2, col3 = st.columns(3)
with col1:
    squaremeters = st.number_input('🏠 Luas Tanah (meter persegi)', 0.0)
    numberofrooms = st.number_input('🛏️ Jumlah Kamar', 0)
    floors = st.number_input('🏢 Jumlah Lantai', 0)
    hasguestroom = st.number_input("🛋️ Apakah memiliki ruang tamu?", 0)
    isnewbuilt = st.radio("🆕 Apakah bangunan baru?", ('New', 'Old'))
with col2:
    numprevowners = st.number_input('👥 Jumlah Pemilik Sebelumnya', 0)
    citypartrange = st.number_input('🌆 Eksklusivitas Kawasan', 0)
    made = st.number_input('🗓️ Tahun Pembuatan', 0)
    hasyard = st.radio("🏡 Apakah memiliki halaman?", ('Ya', 'Tidak'))
    haspool = st.radio("🏊 Apakah memiliki kolam renang?", ('Ya', 'Tidak'))
with col3:
    basement = st.number_input('📦 Luas Basement (meter persegi)', 0.0)
    attic = st.number_input('🏚 Luas Loteng (meter persegi)', 0.0)
    garage = st.number_input('🚚 Luas Garasi (meter persegi)', 0.0)
    hasstormprotector = st.radio("🌪️ Apakah memiliki pelindung badai?", ('Ya', 'Tidak'))
    hasstorageroom = st.radio("📦 Apakah memiliki gudang?", ('Ya', 'Tidak'))

col1, col2 = st.columns(2)
with col1:
    citycode = st.text_input('🗺️ Kode Lokasi (City Code)')


if hasyard == 'Ya':
        onehotencoder__hasyard_no = 0
        onehotencoder__hasyard_yes = 1
```

```
elif hasyard == 'Tidak':
    onehotencoder__hasyard_no = 1
    onehotencoder__hasyard_yes = 0
if haspool == 'Ya':
    onehotencoder__haspool_no = 0
    onehotencoder__haspool_yes = 1
elif haspool == 'Tidak':
    onehotencoder__haspool_no = 1
    onehotencoder__haspool_yes = 0
if isnewbuilt == 'New':
    onehotencoder__isnewbuilt_new = 1
    onehotencoder__isnewbuilt_old = 0
elif isnewbuilt == 'Old':
    onehotencoder__isnewbuilt_new = 0
    onehotencoder__isnewbuilt_old = 1
if hasstormprotector == 'Ya':
    onehotencoder__hasstormprotector_no = 0
    onehotencoder__hasstormprotector_yes = 1
elif hasstormprotector == 'Tidak':
    onehotencoder__hasstormprotector_no = 1
    onehotencoder__hasstormprotector_yes = 0
if hasstorageroom == 'Ya':
    onehotencoder__hasstorageroom_no = 0
    onehotencoder__hasstorageroom_yes = 1
elif hasstorageroom == 'Tidak':
    onehotencoder__hasstorageroom_no = 1
    onehotencoder__hasstorageroom_yes = 0


input_data = [[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, squaremeters,
numberofrooms,
            floors, citycode, citypartrange, numprevowners, made, basement, attic, garage,
hasguestroom]]


        if st.button("🔍 Prediksi"):
            rf_model_predict = rf_model.predict(input_data)
            st.success(f"Prediksi Kategori Properti: **{rf_model_predict[0]}**")
    else:
        st.error("⚠️ Model tidak ditemukan, silakan cek file model di direktori.")


# Halaman Regresi
if selected == 'Regresi':
    st.title('📈 Prediksi Harga Properti')


    model_path = 'BestModel_REG_Lasso_Transformer.pkl'


    st.write('Untuk Inputan File dataset (csv) bisa menggunakan st.file_uploader')
    file = st.file_uploader('Masukkan File Dataset Anda', type=["csv", "txt"])


    def validate_file(data):
        keywords = ["luxury", "basic", "middle"]
        data_str = data.astype(str)  # Convert all data to string for searching
        if not data_str.apply(lambda x: x.str.contains('|'.join(keywords), case=False)).any().any():
            raise ValueError("The dataset does not contain the required categories: 'luxury', 'basic',
'middle'.")
    if file is not None:
        try:
            input_data = pd.read_csv(file)
```

```
        validate_file(input_data)
        st.markdown("<h3 style='text-align: center; color: #0073e6;'>📊 Data yang Diupload:</h3>", unsafe_allow_html=True)
        st.dataframe(input_data)
    except ValueError as e:
        st.markdown("<h3 style='text-align : center' font-color: red> !! ⚠️ An error occurred while reading the file ⚠️ !! </h3>", unsafe_allow_html=True)


  if os.path.exists(model_path):
    with open(model_path, 'rb') as f:
      loaded_model = pickle.load(f)
    Lasso_model = loaded_model.best_estimator_

    # Bagian Input
    st.markdown("### Masukkan Fitur Properti:")
    col1, col2, col3 = st.columns(3)
    with col1:
      squaremeters = st.number_input('🏠 Luas Tanah (meter persegi)', 0.0)
      numberofrooms = st.number_input('🛏️ Jumlah Kamar', 0)
      floors = st.number_input('🏢 Jumlah Lantai', 0)
      hasguestroom = st.number_input("🛋️ Apakah memiliki ruang tamu?", 0)
      isnewbuilt = st.radio("🆕 Apakah bangunan baru?", ('New', 'Old'))
    with col2:
      numprevowners = st.number_input('👥 Jumlah Pemilik Sebelumnya', 0)
      citypartrange = st.number_input('🌆 Eksklusivitas Kawasan', 0)
      made = st.number_input('📅 Tahun Pembuatan', 0)
      hasyard = st.radio("🏡 Apakah memiliki halaman?", ('Ya', 'Tidak'))
      haspool = st.radio("🏊 Apakah memiliki kolam renang?", ('Ya', 'Tidak'))

    with col3:
```

```
    basement = st.number_input('📦 Luas Basement (meter persegi)', 0.0)
    attic = st.number_input('🏠 Luas Loteng (meter persegi)', 0.0)
    garage = st.number_input('🚚 Luas Garasi (meter persegi)', 0.0)
    hasstormprotector = st.radio("🌪️ Apakah memiliki pelindung badai?", ('Ya', 'Tidak'))
    hasstorageroom = st.radio("📦 Apakah memiliki gudang?", ('Ya', 'Tidak'))

col1, col2 = st.columns(2)
with col1:
    citycode = st.text_input('🗺️ Kode Lokasi (City Code)')


if hasyard == 'Ya':
        onehotencoder__hasyard_no = 0
        onehotencoder__hasyard_yes = 1
elif hasyard == 'Tidak':
        onehotencoder__hasyard_no = 1
        onehotencoder__hasyard_yes = 0
if haspool == 'Ya':
        onehotencoder__haspool_no = 0
        onehotencoder__haspool_yes = 1
elif haspool == 'Tidak':
        onehotencoder__haspool_no = 1
        onehotencoder__haspool_yes = 0
if isnewbuilt == 'New':
        onehotencoder__isnewbuilt_new = 1
        onehotencoder__isnewbuilt_old = 0
elif isnewbuilt == 'Old':
        onehotencoder__isnewbuilt_new = 0
        onehotencoder__isnewbuilt_old = 1
```

```
    if hasstormprotector == 'Ya':
        onehotencoder__hasstormprotector_no = 0
        onehotencoder__hasstormprotector_yes = 1
    elif hasstormprotector == 'Tidak':
        onehotencoder__hasstormprotector_no = 1
        onehotencoder__hasstormprotector_yes = 0
    if hasstorageroom == 'Ya':
        onehotencoder__hasstorageroom_no = 0
        onehotencoder__hasstorageroom_yes = 1
    elif hasstorageroom == 'Tidak':
        onehotencoder__hasstorageroom_no = 1
        onehotencoder__hasstorageroom_yes = 0


    input_data = [[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,squaremeters,
numberofrooms,
        floors, citycode, citypartrange, numprevowners, made, basement, attic, garage,
hasguestroom]]  # Contoh untuk disederhanakan


    if st.button(" 💸 Prediksi Harga"):
      Lasso_model_predict = Lasso_model.predict(input_data)
      formatted_price = f"${Lasso_model_predict[0]:,.2f}"
      st.markdown(f"<h3 style='color: #4C9A2A;'> 💵 Prediksi Harga Properti:
{formatted_price}</h3>", unsafe_allow_html=True)


  else:
    st.error(" ⚠️ Model tidak ditemukan, silakan cek file model di direktori.")


# Halaman Catatan
```

```
if selected == 'Catatan':
    st.title('📚 Catatan')
    st.write('''1. Bagian sidebar akan menampilkan menu dari fungsi streamlit yang kita buat".''')
    st.write('2. Menu yang dibuat ada 2 yaitu Klasifikasi dan Regresi.')
    st.write('''3. Klasifikasi akan menghasilkan output kategori properti sedangkan regresi akan menghasilkan output harga properti.''')
    st.write('Frans Daniel Rajagukguk - 220711826')
    st.write('Angello Khara Sitanggang - 220711833')
    st.write('Davin Gilbert Natanael - 220711841')
    st.write('Emanuel Enrico Anindya Wibawa - 220711890')
    st.write('Archipera Rari Dyaksa - 220711891')
```