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

data = pd.read_csv('Dataset UTS_Gasal 2425.csv')
data.head(50)
```

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	

In [2]:
```
data.describe()
```

Out[2]:

	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 [3]:
```
data.info()
```

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

In [4]:
```
data['price'].value_counts()
```

Out[4]:
```
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
```

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

Out[5]:
```
category
Basic     4344
Luxury    3065
Middle    2591
Name: count, dtype: int64
```

In [6]:
```
data.describe(include='all')
```

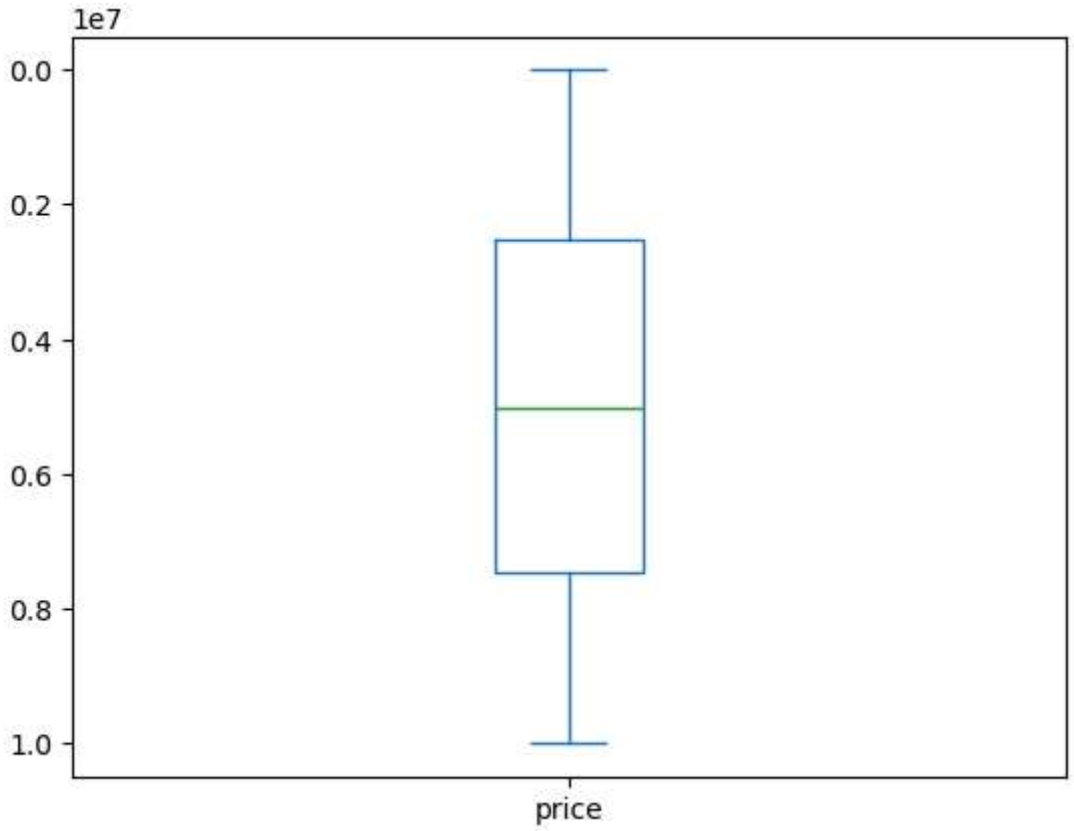
Out[6]:

	squaremeters	numberofrooms	hasyard	haspool	floors	citycode	city
count	10000.00000	10000.000000	10000	10000	10000.000000	10000.000000	100
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 [7]:
```
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
category             0
dtype: int64

Data Kosong
 False

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

In [2]:
```
import matplotlib.pyplot as plt

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



In [9]:
```
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, 18)
Sesudah drop data duplicate  (10000, 18)
(7500, 17)
(2500, 17)
```

In [10]:
```
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(



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

Out[10]:

	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 × 22 columns

In [11]:

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.feature_selection import SelectKBest, SelectPercentile
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import classification_report,confusion_matrix,ConfusionMatrixD


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

#buat parameter grid untuk step feature selection dan classifier
params_grid_svm = [
    {
```

```
        'scale': [MinMaxScaler()],
        'feat_select__k': np.arange(5, 20),
        'clf__kernel': ['poly', 'rbf'],
        'clf__C': [0.1,1],
        'clf__gamma': [0.1, 1]
        #'clf__kernel': Memilih kernel terbaik (polinomial atau RBF) yang cocok unt
        # 'clf__C': Mengontrol keseimbangan antara generalisasi dan akurasi pada mo
        # 'clf__gamma': Mengatur jangkauan pengaruh dari setiap contoh training unt
    },
    {
        'scale': [MinMaxScaler()],
        'feat_select': [SelectPercentile()],
        #0% hingga 90%, dengan kenaikan 10%. Jadi, pada setiap iterasi dari GridSea
        'feat_select__percentile': np.arange(10, 100, 10),
        #'poly': Kernel polinomial, yang digunakan untuk memodelkan hubungan non-li
        'clf__kernel': ['poly', 'rbf'],
        'clf__C': [ 0.1, 1],
        'clf__gamma': [0.1, 1]
    },
    {
        'scale': [StandardScaler()],
        'feat_select__k': np.arange(5, 20),
        'clf__kernel': ['poly', 'rbf'],
        'clf__C': [0.1, 1],
        'clf__gamma': [0.1, 1]
    },
    {
        'scale': [StandardScaler()],
        'feat_select': [SelectPercentile()],
        'feat_select__percentile': np.arange(10, 100, 10),
        'clf__kernel': ['poly', 'rbf'],
        'clf__C': [0.1, 1],
        'clf__gamma': [0.1, 1]
    }
]

#muat rancangan pipeline ke dalam objek pipeline
estimator_svm = Pipeline(pipe_svm)

#muat pipeline dan parameter gri ke dalam objek GSCV dengan Stratified 5-fold CV
SKF = StratifiedKFold(n_splits=10, shuffle=True, random_state=91)
GSCV_SVM = GridSearchCV(pipe_svm, params_grid_svm, cv=SKF ,n_jobs=-1, verbose=1)

#jalankan objek GSC untuk melatih model dengan train set menggunakan fungsi fit
GSCV_SVM.fit(X_train_encoded, y_train)
print("GSCV training finished")
```

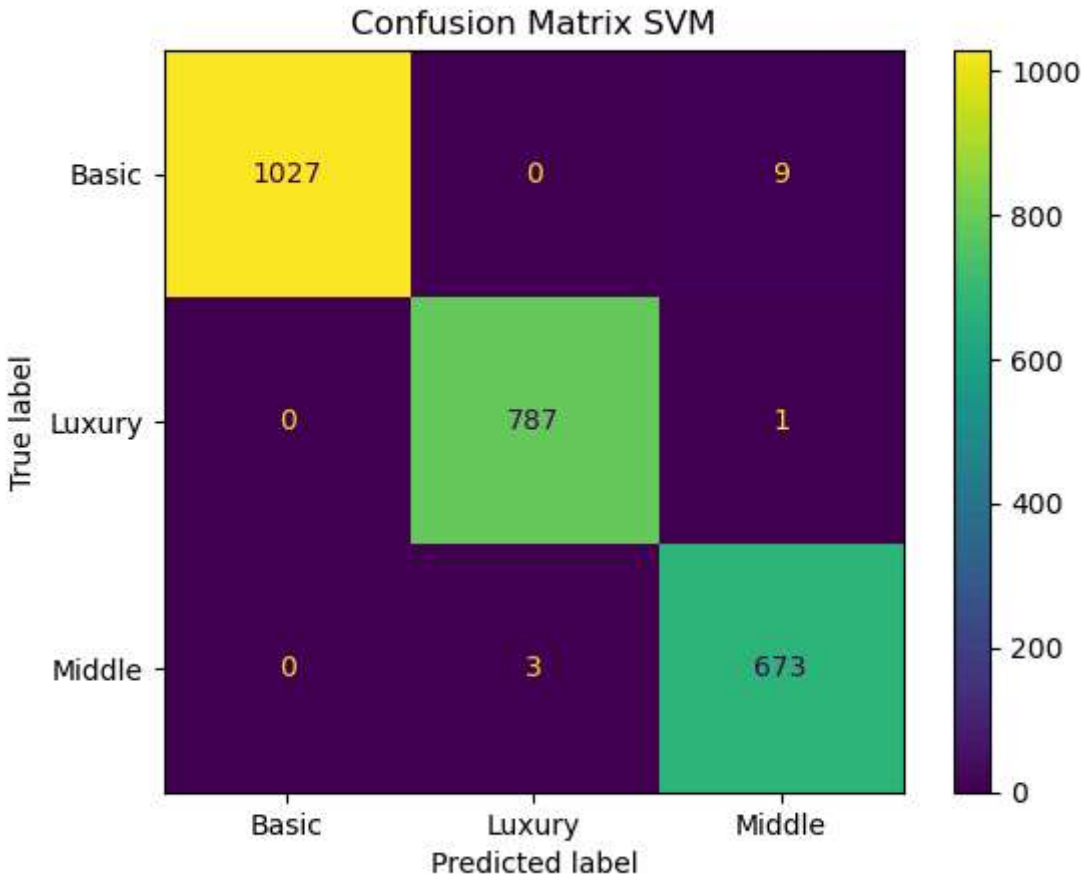
```
Fitting 10 folds for each of 384 candidates, totalling 3840 fits
GSCV training finished
```

In [12]:

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


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

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

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

SVM_predict = GSCV_SVM.predict(X_test_encoded)

cm = confusion_matrix(y_test, SVM_predict, labels=GSCV_SVM.classes_)

display = ConfusionMatrixDisplay(cm, display_labels=GSCV_SVM.classes_)
display.plot()
plt.title('Confusion Matrix SVM')
plt.show()


print("classification_report SVM:\n" , classification_report(y_test, SVM_predict))
```

```
CV Score : 0.9951999999999999
Best Param : {'clf__C': 1, 'clf__gamma': 1, 'clf__kernel': 'poly', 'feat_select__k':
8, 'scale': StandardScaler()}
Test Score : 0.9948
Best Model : Pipeline(steps=[('scale', StandardScaler()), ('feat_select', SelectKBes
t(k=8)),
                ('clf',
                 SVC(C=1, class_weight='balanced', gamma=1, kernel='poly'))])
Best Feature:  Index(['onehotencoder__hasyard_no', 'onehotencoder__hasyard_yes',
       'onehotencoder__haspool_no', 'onehotencoder__haspool_yes',
       'onehotencoder__isnewbuilt_new', 'onehotencoder__isnewbuilt_old',
       'remainder__squaremeters', 'remainder__price'],
      dtype='object')
```




```
classification_report SVM:
              precision    recall  f1-score   support

       Basic       1.00      0.99      1.00      1036
      Luxury       1.00      1.00      1.00       788
      Middle       0.99      1.00      0.99       676

    accuracy                           0.99      2500
   macro avg       0.99      1.00      0.99      2500
weighted avg       0.99      0.99      0.99      2500
```

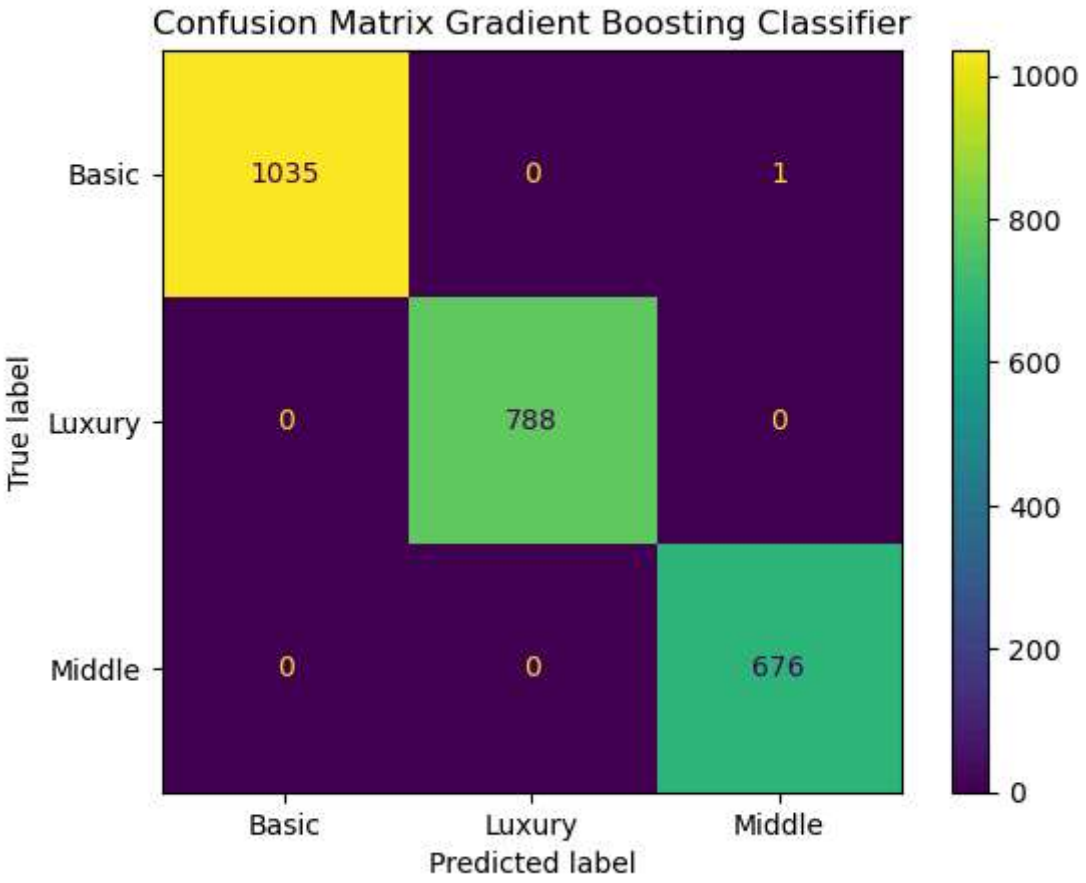
In [18]:
```
pipe_gbc = Pipeline(steps=[
    ('scale', MinMaxScaler()),
    ('feat_select', SelectKBest()),
    ('clf', GradientBoostingClassifier())
])

#buat parameter grid untuk step feature selection dan classifier

# n_estimators: [50, 100, 200]
# n_estimators menentukan jumlah pohon keputusan yang akan dibangun.
# - 50: Jumlah pohon yang lebih sedikit, lebih cepat, tetapi mungkin kurang komplek
# - 100: Jumlah pohon yang sedang, keseimbangan antara performa dan waktu komputasi
# - 200: Jumlah pohon yang lebih banyak, meningkatkan kompleksitas dan akurasi, tet

# learning_rate: [0.01, 0.1, 0.2]
# learning_rate menentukan seberapa besar setiap pohon baru berkontribusi terhadap
# - 0.01: Pembelajaran lebih lambat, namun bisa lebih stabil, cocok jika menggunaka
```

```
# - 0.1: Nilai standar yang sering digunakan, memberikan keseimbangan antara kecepa
# - 0.2: Pembelajaran lebih cepat, tetapi risiko overfitting lebih tinggi jika tida

# max_depth: [3, 4, 5]
# max_depth menentukan kedalaman maksimum dari setiap pohon keputusan.
# - 3: Pohon lebih dangkal, membuat model lebih sederhana, cocok untuk dataset yang
# - 4: Kedalaman menengah, seimbang dalam menangkap pola dan menghindari overfittin
# - 5: Pohon lebih dalam, lebih kompleks, cocok untuk dataset yang kompleks, tetapi
params_grid_gbc = [
    {
        'scale': [MinMaxScaler()],
        'feat_select__k': np.arange(5, 20),
        'clf__n_estimators': [50, 100, 200],
        'clf__learning_rate': [0.01, 0.1, 0.2],
        'clf__max_depth': [3, 4, 5]

    },
    {
        'scale': [MinMaxScaler()],
        'feat_select': [SelectPercentile()],
        #0% hingga 90%, dengan kenaikan 10%. Jadi, pada setiap iterasi dari GridSea
        'feat_select__percentile': np.arange(10, 100, 10),
        'clf__n_estimators': [50, 100, 200],
        'clf__learning_rate': [0.01, 0.1, 0.2],
        'clf__max_depth': [3, 4, 5]
    },
    {
        'scale': [StandardScaler()],
        'feat_select__k': np.arange(5, 20),
        'clf__n_estimators': [50, 100, 200],
        'clf__learning_rate': [0.01, 0.1, 0.2],
        'clf__max_depth': [3, 4, 5]
    },
    {
        'scale': [StandardScaler()],
        'feat_select': [SelectPercentile()],
        'feat_select__percentile': np.arange(10, 100, 10),
        'clf__n_estimators': [50, 100, 200],
        'clf__learning_rate': [0.01, 0.1, 0.2],
        'clf__max_depth': [3, 4, 5]
    }
]


estimator_svm = Pipeline(pipe_gbc)


SKF = StratifiedKFold(n_splits=10, shuffle=True, random_state=91)
GSCV_GBC = GridSearchCV(pipe_gbc, params_grid_gbc, cv=SKF ,n_jobs=-1, verbose=1)


GSCV_GBC.fit(X_train_encoded, y_train)
print("GSCV training finished")
```

```
Fitting 10 folds for each of 1296 candidates, totalling 12960 fits
GSCV training finished
```

In [26]:
```
print('CV Score : {}'.format(GSCV_GBC.best_score_))
print('Best Param : {}'.format(GSCV_GBC.best_params_))


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

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

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

GBC_predict = GSCV_GBC.predict(X_test_encoded)

cm = confusion_matrix(y_test, GBC_predict, labels=GSCV_GBC.classes_)

display = ConfusionMatrixDisplay(cm, display_labels=GSCV_GBC.classes_)
display.plot()
plt.title('Confusion Matrix Gradient Boosting Classifier')
plt.show()


print("classification_report Gradient Boosting Classifier:\n" , classification_repo
```

```
CV Score : 0.9998666666666667
Best Param : {'clf__learning_rate': 0.01, 'clf__max_depth': 3, 'clf__n_estimators':
200, 'feat_select__k': 7, 'scale': MinMaxScaler()}
Test Score : 0.9996
Best Model : Pipeline(steps=[('scale', MinMaxScaler()), ('feat_select', SelectKBest
(k=7)),
                ('clf',
                 GradientBoostingClassifier(learning_rate=0.01,
                                            n_estimators=200))])
Best Feature:  Index(['onehotencoder__hasyard_yes', 'onehotencoder__haspool_no',
       'onehotencoder__haspool_yes', 'onehotencoder__isnewbuilt_new',
       'onehotencoder__isnewbuilt_old', 'remainder__squaremeters',
       'remainder__price'],
      dtype='object')
```




```
classification_report Gradient Boosting 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 [21]:
```
if GSCV_SVM.best_estimator_.score(X_test_encoded, y_test) > GSCV_GBC.best_estimator
    print('Best model is SVM')
    name = 'SVM'
    model = GSCV_SVM
else:
    print('Best model is Gradient Boosting Classifier')
    name = 'Gradient Boosting Classifier'
    model = GSCV_GBC
import pickle
with open('Model_' + name + '.pkl', 'wb') as file:
    pickle.dump(model, file)
```

```
Best model is Gradient Boosting Classifier
```

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

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

Out[2]:

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

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

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

Out[5]:
```
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
```

In [6]:
```
data['category'].value_counts()
```

Out[6]:
```
category
Basic     4344
Luxury    3065
Middle    2591
Name: count, dtype: int64
```

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

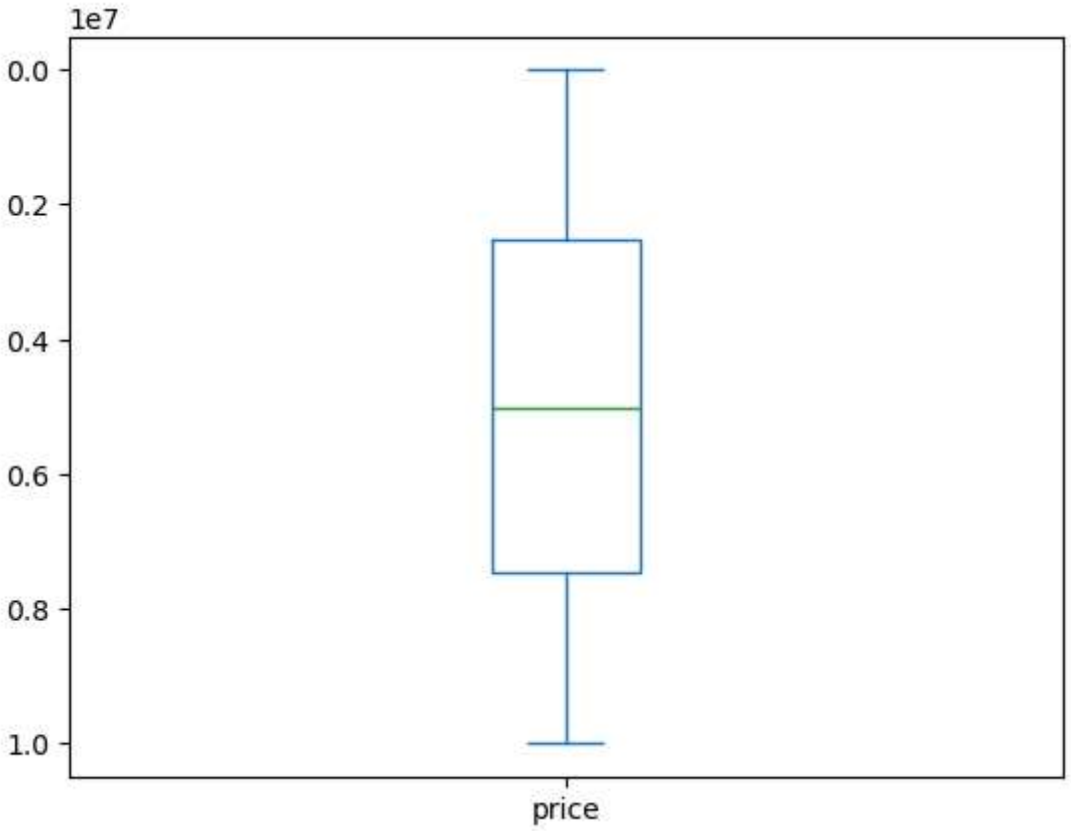
Out[7]:

	squaremeters	numberofrooms	hasyard	haspool	floors	citycode	city
count	10000.00000	10000.000000	10000	10000	10000.000000	10000.000000	100
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
category             0
dtype: int64

Data Kosong
 False

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

In [9]:
```
import matplotlib.pyplot as plt

data['price'].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, 18)
Sesudah drop data duplicate  (10000, 18)
(7500, 17)
(2500, 17)
```

In [12]:
```
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[12]:

	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 × 22 columns

In [13]:

```
#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=10, 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')
```

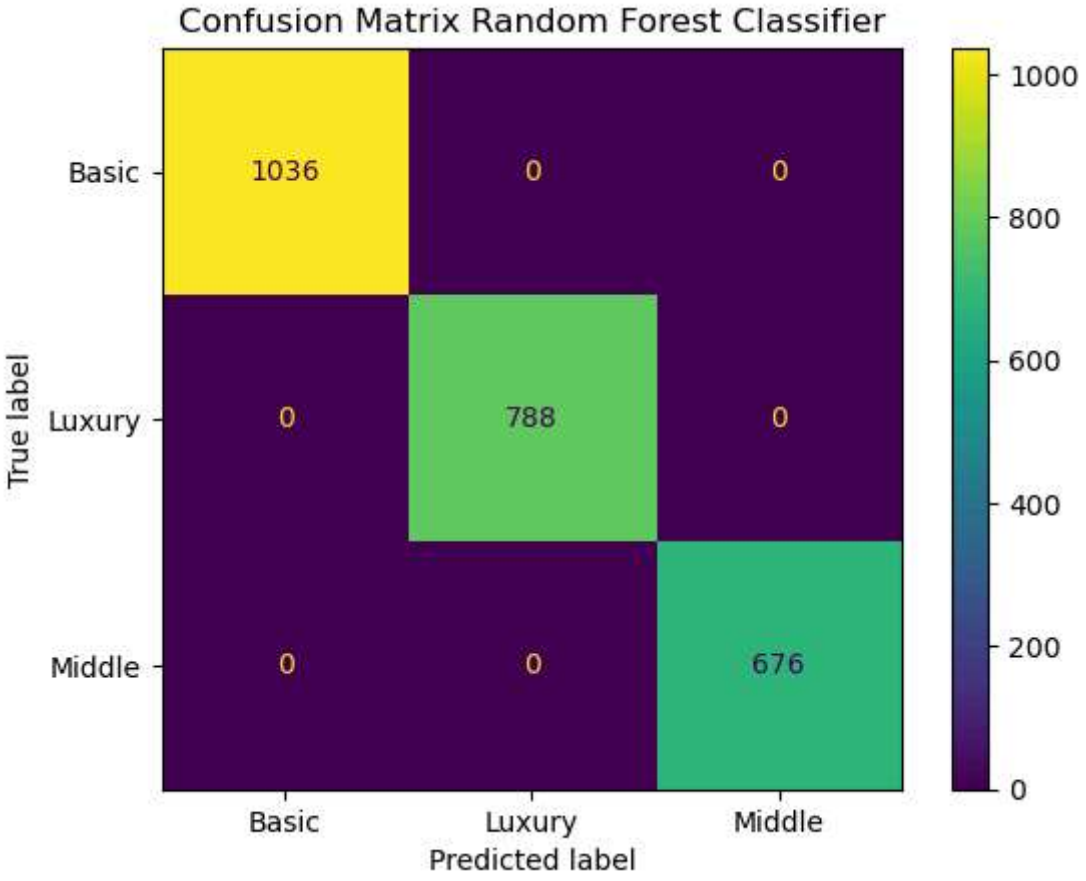
```
Fitting 10 folds for each of 576 candidates, totalling 5760 fits
FINISH
```

In [14]:

```
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 : 0.9998666666666667
Best Param : {'clf__max_depth': None, 'clf__n_estimators': 200, 'feat_select__k': 9,
'scale': StandardScaler()}
Test Score : 1.0
Best Model : Pipeline(steps=[('scale', StandardScaler()), ('feat_select', SelectKBes
t(k=9)),
                ('clf', RandomForestClassifier(n_estimators=200))])
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__price'],
      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 [15]:

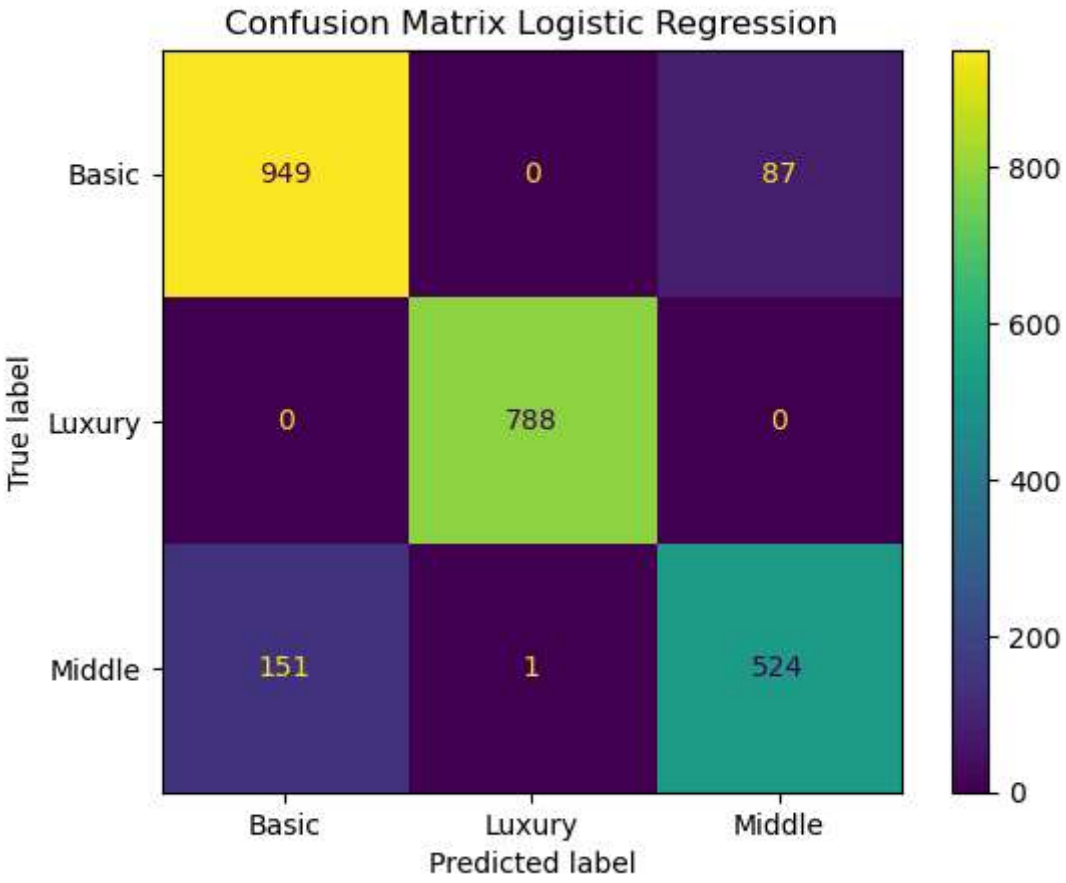
```
pipe_LR = Pipeline(steps=[
    ('scale', MinMaxScaler()),
    ('feat_select', SelectKBest()),
    ('clf', LogisticRegression())
])


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 [16]:
```
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 : 0.8984
Best Param : {'clf__C': 10, 'clf__max_iter': 10000, 'clf__penalty': 'l1', 'clf__solv
er': 'saga', 'feat_select': SelectPercentile(), 'feat_select__percentile': 10, 'scal
e': StandardScaler()}
Test Score : 0.9044
Best Model : Pipeline(steps=[('scale', StandardScaler()),
                ('feat_select', SelectPercentile()),
                ('clf',
                 LogisticRegression(C=10, max_iter=10000, penalty='l1',
                                    solver='saga'))])
Best Feature:  Index(['remainder__squaremeters', 'remainder__price'], dtype='objec
t')
```




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

       Basic       0.86      0.92      0.89      1036
      Luxury       1.00      1.00      1.00       788
      Middle       0.86      0.78      0.81       676

    accuracy                           0.90      2500
   macro avg       0.91      0.90      0.90      2500
weighted avg       0.90      0.90      0.90      2500
```

In [31]:
```
# 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
```

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.describe()
```

Out[2]:

	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 [3]:
```
data.info()
```

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

In [4]:
```
data['garage'].value_counts()
```

Out[4]:
```
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 [5]:
```
data['price'].describe()
```

Out[5]:
```
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 [6]:
```
data.describe(include='all')
```

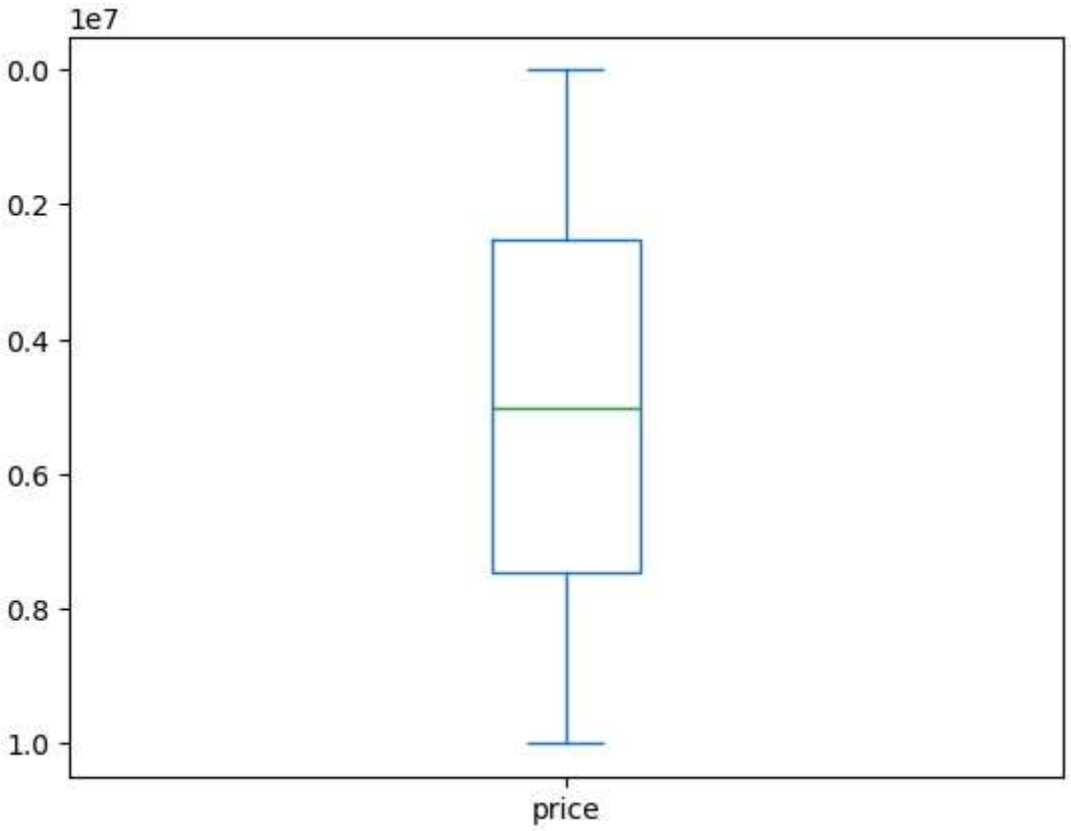
Out[6]:

	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 [7]:
```
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
category             0
dtype: int64

Data Kosong
 False

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

In [8]:
```
import matplotlib.pyplot as plt

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



In [9]:
```
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, 18)
Sesudah drop data duplicate  (10000, 18)
(7500, 17)
(2500, 17)
```

In [10]:
```
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[10]:

	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 [11]:

```
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', MinMaxScaler()),
                ('feat_select',
                 SelectPercentile(percentile=95,
                                  score_func=<function f_regression at 0x000001CADE3
1BE20>)),
                ('reg', Lasso(alpha=10, max_iter=10000))])
Lasso Best parameters:{'feat_select': SelectPercentile(score_func=<function f_regres
sion at 0x000001CADE31BE20>), 'feat_select__percentile': 95, 'reg__alpha': 10, 'scal
e': MinMaxScaler()}
Koeffisien Ridge:[-2.80186269e+03  3.49489546e-13 -2.76812176e+03  0.00000000e+00
 -0.00000000e+00  0.00000000e+00 -8.11218816e+01 -0.00000000e+00
  0.00000000e+00 -5.85424925e+02  1.36385399e+03  3.01723964e+02
  9.98618838e+06 -0.00000000e+00  5.31760328e+03  3.06151878e+02
  0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00
  3.16786171e+01 -0.00000000e+00]
Intercept/bias:18588.186509666964
Lasso Mean Squared Error (MSE): 3756546.8885637573
Lasso Mean Absolute Error (MAE): 1523.171391273097
Lasso Root Mean Squared Error: 1938.1813353150828
```

In [12]:
```
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()
```

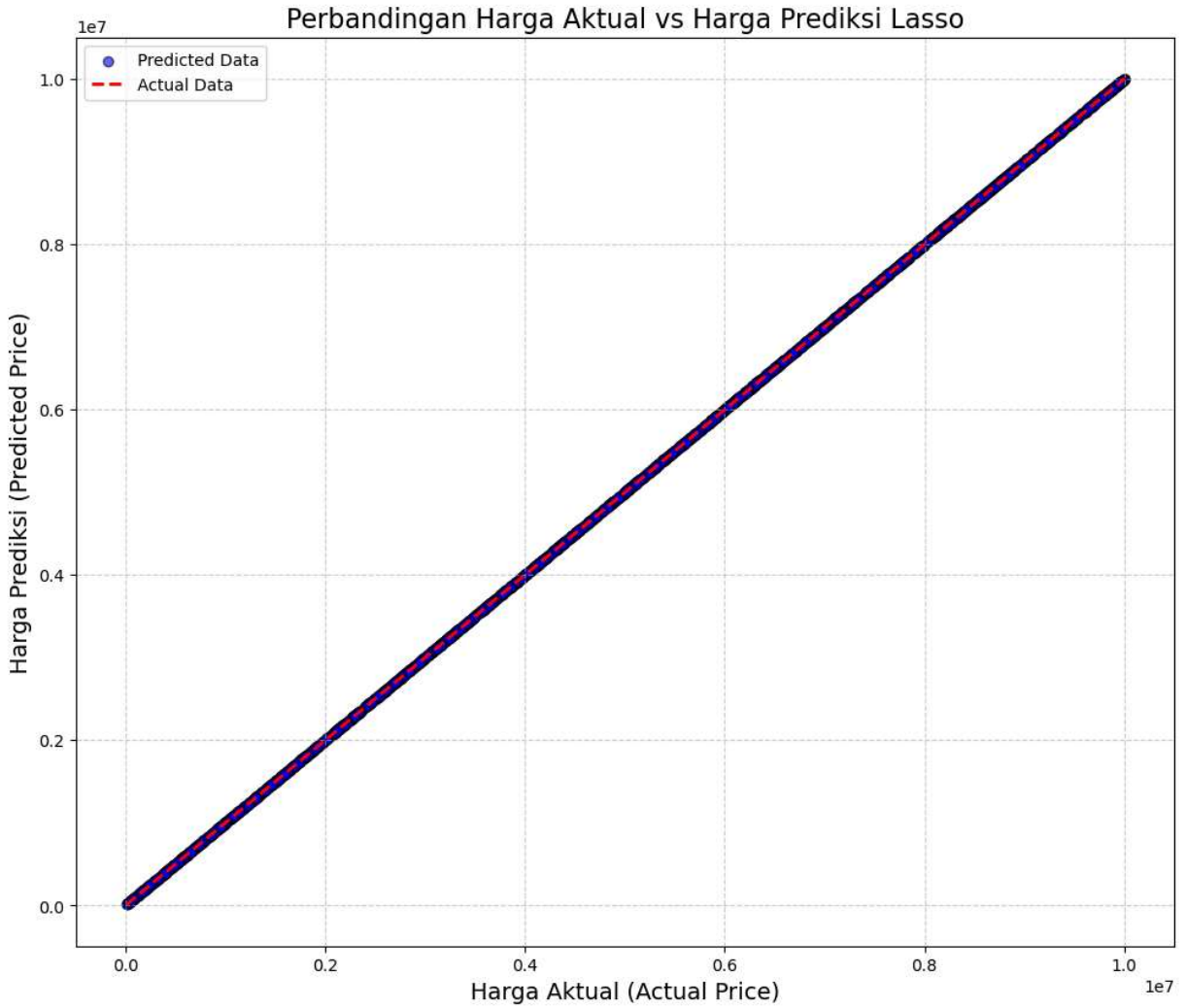
Out[12]:

	price	Lasso Prediction	Selisih_Pricew_LS
9600	2,327,859.80	2,327,064.79	-795.01
2889	8,694,297.10	8,694,822.37	525.27
2694	8,221,240.90	8,224,526.34	3,285.44
4775	3,252,838.40	3,253,972.98	1,134.58
5567	463,922.20	469,326.71	5,404.51

In [13]:
```
from sklearn.metrics import r2_score
r2 = r2_score(y_test, Lasso_predict)
print("R² score:", r2)
```

```
R² score: 0.999999553794295
```

In [14]:
```
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 [15]:
```
df_results.describe()
```

Out[15]:

	price	Lasso Prediction	Selisih_Pricew_LS
count	2,500.00	2,500.00	2,500.00
mean	5,070,286.87	5,070,290.09	3.23
std	2,902,109.03	2,902,101.96	1,938.57
min	10,313.50	14,120.79	-6,919.34
25%	2,572,316.52	2,571,959.05	-1,237.03
50%	5,150,484.40	5,151,121.00	63.04
75%	7,620,090.12	7,621,021.20	1,245.66
max	10,002,945.00	10,001,025.01	6,077.50

In [16]:
```
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'].


RFC_predict = GSCV_RFR.predict(X_test_encoded)

mse_RFR = mean_squared_error(y_test, RFC_predict)
mae_RFR = mean_absolute_error(y_test, RFC_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:\ProgramData\anaconda3\Lib\site-packages\sklearn\feature_selection\_univariate_sel
ection.py:109: RuntimeWarning: invalid value encountered in divide
  msw = sswn / float(dfwn)
Best Model:Pipeline(steps=[('scale', StandardScaler()),
                ('feat_select', SelectPercentile(percentile=60)),
                ('reg', RandomForestRegressor(max_depth=15, n_estimators=200))])
RFR Best parameters:{'feat_select': SelectPercentile(), 'feat_select__percentile': 6
0, 'reg__max_depth': 15, 'reg__n_estimators': 200, 'scale': StandardScaler()}
Feature Importances: [6.32437403e-08 6.58633831e-08 6.39659543e-08 6.23173307e-08
 4.96111477e-08 5.07259572e-08 4.77723224e-08 4.75201602e-08
 5.01696216e-08 4.90497386e-08 1.43204044e-08 1.50566422e-05
 1.49362720e-05 9.99969443e-01]
RFR Mean Squared Error (MSE): 13273976.63758383
RFR Mean Absolute Error (MAE): 2919.7930005813105
RFR Root Mean Squared Error: 3643.346900527567
```

In [17]:
```
import pandas as pd


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


df_results['RFR Prediction'] = RFC_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()
```

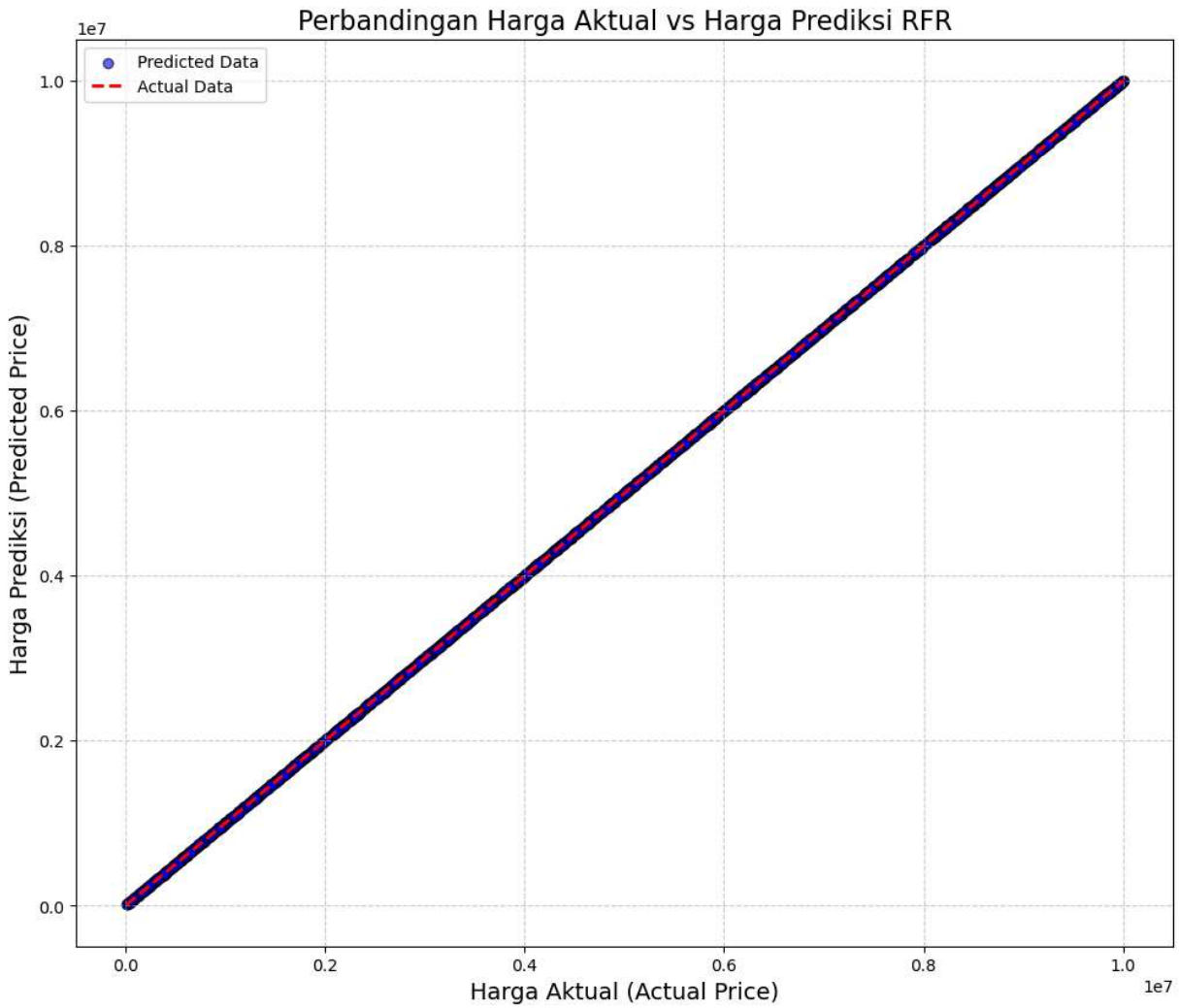
Out[17]:

	price	RFR Prediction	Selisih_Pricew_RFR
9600	2,327,859.80	2,329,745.11	1,885.31
2889	8,694,297.10	8,696,669.85	2,372.75
2694	8,221,240.90	8,227,981.57	6,740.67
4775	3,252,838.40	3,253,064.19	225.79
5567	463,922.20	461,206.06	-2,716.14

In [25]:
```
from sklearn.metrics import r2_score
r2 = r2_score(y_test, RFC_predict)
print("R² score:", r2)
```

```
R² score: 0.9999984233062226
```

In [19]:
```
import matplotlib.pyplot as plt
import numpy as np


plt.figure(figsize=(12, 10))
plt.scatter(y_test, RFC_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 [20]:
```
df_results.describe()
```

Out[20]:

	price	RFR Prediction	Selisih_Pricew_RFR
count	2,500.00	2,500.00	2,500.00
mean	5,070,286.87	5,070,315.75	28.88
std	2,902,109.03	2,902,233.24	3,643.96
min	10,313.50	14,696.53	-13,179.55
25%	2,572,316.52	2,572,644.92	-2,357.41
50%	5,150,484.40	5,152,858.19	259.36
75%	7,620,090.12	7,620,406.79	2,609.90
max	10,002,945.00	9,998,812.96	10,717.25

In [21]:
```
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'] = RFC_predict
```

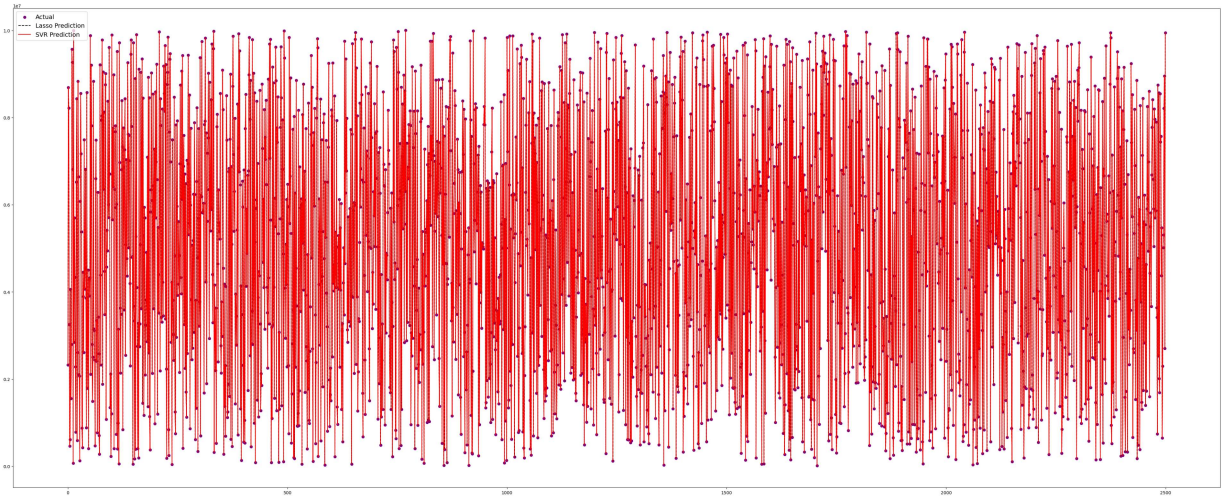
```
df_results['Selisih_IPK_RFR'] = df_results['price'] - df_results['RFR Prediction']

df_results.head()
```

Out[21]:

	price	Lasso Prediction	Selisih_IPK_LR	RFR Prediction	Selisih_IPK_RFR
9600	2,327,859.80	2,327,064.79	795.01	2,329,745.11	-1,885.31
2889	8,694,297.10	8,694,822.37	-525.27	8,696,669.85	-2,372.75
2694	8,221,240.90	8,224,526.34	-3,285.44	8,227,981.57	-6,740.67
4775	3,252,838.40	3,253,972.98	-1,134.58	3,253,064.19	-225.79
5567	463,922.20	469,326.71	-5,404.51	461,206.06	2,716.14

In [22]:
```
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[22]: <function matplotlib.pyplot.show(close=None, block=None)>



In [26]:
```
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__percentile']

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

svr_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): {s


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

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

```
Lasso MAE: 1523.171391273097, Lasso RMSE: 1938.1813353150828, Lasso Feature Count(pe
rcentile): 95
RFR MAE: 2919.7930005813105, RFR RMSE: 3643.346900527567, RFR Feature Count(percenti
le): 60
R² Lasso score: 0.999999553794295
R² RFRC score: 0.9999984233062226
```

In [24]:
```
#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:  1523.171391273097 and RMSE:  1938.1813353150828
```

In [3]:
```
import pandas as pd
import numpy as np

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

Out[3]:

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

Out[4]:

	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 [5]:
```
data.info()
```

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

In [6]:
```
data['garage'].value_counts()
```

Out[6]:
```
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 [7]:
```
data['price'].describe()
```

Out[7]:
```
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 [8]:
```
data.describe(include='all')
```

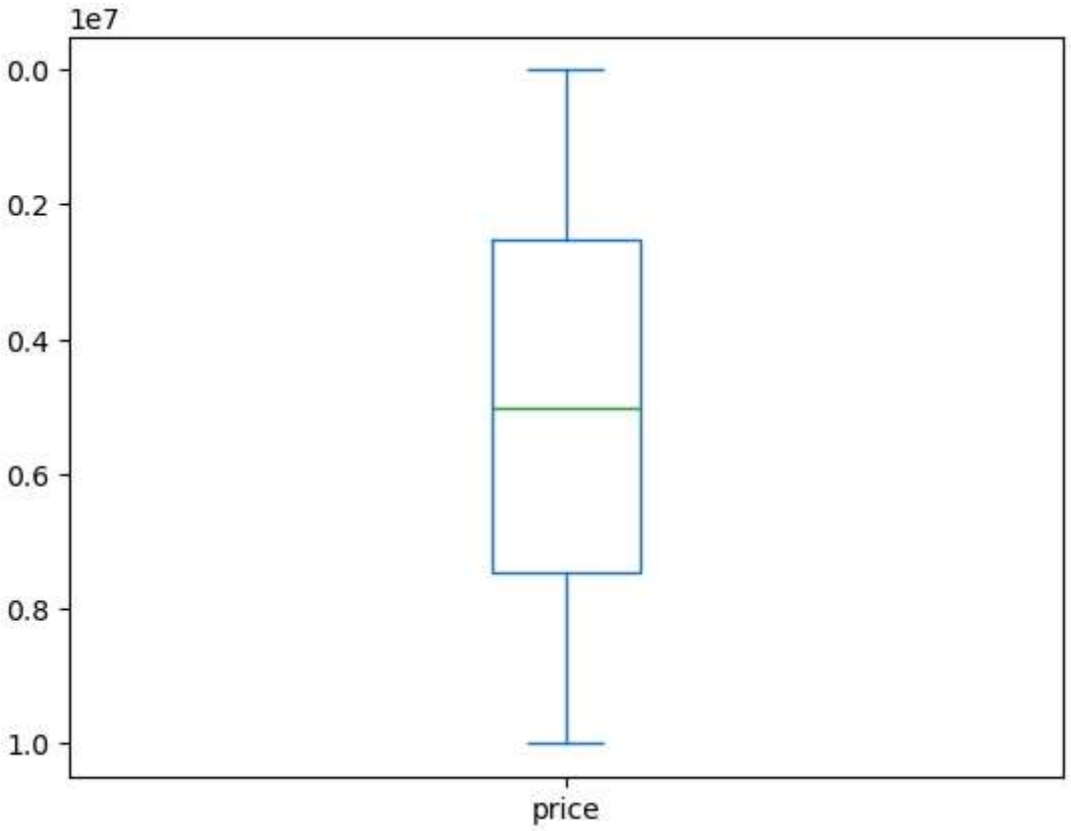
Out[8]:

	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
price                0
category             0
dtype: int64

Data Kosong
 False

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

In [10]:
```
import matplotlib.pyplot as plt

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



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, 18)
Sesudah drop data duplicate  (10000, 18)
(7500, 17)
(2500, 17)
```

In [12]:
```
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[12]:

	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 [13]:

```
from sklearn.linear_model import Ridge
from sklearn.svm import SVR
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_Ridge = Pipeline([
    ('scale', StandardScaler()),
    ('feat_select', SelectKBest(score_func=f_regression)),
    ('reg', Ridge())
])

param_grid_Ridge = [
    {
        'scale': [MinMaxScaler()],
        'reg__alpha': [0.01, 0.1, 1, 10, 100],
```

```
        'feat_select__k': np.arange(7, 22),
        'reg__solver': ['saga'],
        'reg__max_iter': [10000]
    },
    {
        '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),
        'reg__solver': ['saga'],
        'reg__max_iter': [10000]
    },
    {
        'scale': [StandardScaler()],
        'reg__alpha': [0.01, 0.1, 1, 10, 100],
        'feat_select__k': np.arange(7, 22),
        'reg__solver': ['saga'],
        'reg__max_iter': [10000]
    },
    {
        '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),
        'reg__solver': ['saga'],
        'reg__max_iter': [10000]
    },
]



kf = KFold(n_splits=7, shuffle=True, random_state=91)
GSCV_RR = GridSearchCV(pipe_Ridge, param_grid=param_grid_Ridge, cv=kf, n_jobs=-1, s

GSCV_RR.fit(X_train_encoded, y_train)

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

Ridge_predict = GSCV_RR.predict(X_test_encoded)

mse_Ridge = mean_squared_error(y_test, Ridge_predict)
mae_Ridge = mean_absolute_error(y_test, 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)))
```

```
Fitting 7 folds for each of 330 candidates, totalling 2310 fits
Best Model:Pipeline(steps=[('scale', StandardScaler()),
                ('feat_select',
                 SelectPercentile(percentile=90,
                                  score_func=<function f_regression at 0x00000258A1F
3E160>)),
                ('reg', Ridge(alpha=0.1, max_iter=10000, solver='saga'))])
Ridge Best parameters:{'feat_select': SelectPercentile(score_func=<function f_regres
sion at 0x00000258A1F3E160>), 'feat_select__percentile': 90, 'reg__alpha': 0.1, 'reg
__max_iter': 10000, 'reg__solver': 'saga', 'scale': StandardScaler()}
Koeffisien Ridge:[-7.16763873e+02  7.16763873e+02 -7.39487541e+02  7.39487541e+02
  8.83715916e+00 -8.83715916e+00  6.04493390e+00 -6.04493390e+00
 -1.10500000e+02  1.47673862e+02 -2.98988174e+01  2.86852456e+06
 -2.65802922e+01  1.64385098e+03  5.92483893e+01  1.95150500e+01
 -2.13125739e+01 -3.29796959e+01 -1.12190684e+02  7.08771690e+01
 -1.21777285e+01]
Intercept/bias:4967834.411573334
Ridge Mean Squared Error (MSE): 3636206.345466554
Ridge Mean Absolute Error (MAE): 1491.0326240901168
Ridge Root Mean Squared Error: 1906.883936024045
```

In [14]:
```python
import pandas as pd

# Membuat DataFrame dari y_test dengan kolom 'price'
df_results = pd.DataFrame(y_test, columns=['price'])

# Menambahkan kolom untuk prediksi Ridge
df_results['Ridge Prediction'] = Ridge_predict

# Menambahkan kolom yang berisi selisih antara prediksi dan harga aktual
df_results['Selisih_Pricew_RR'] = df_results['Ridge Prediction'] - df_results['pric

# Mengatur agar angka ditampilkan dalam format biasa
pd.set_option('display.float_format', '{:,.2f}'.format)

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

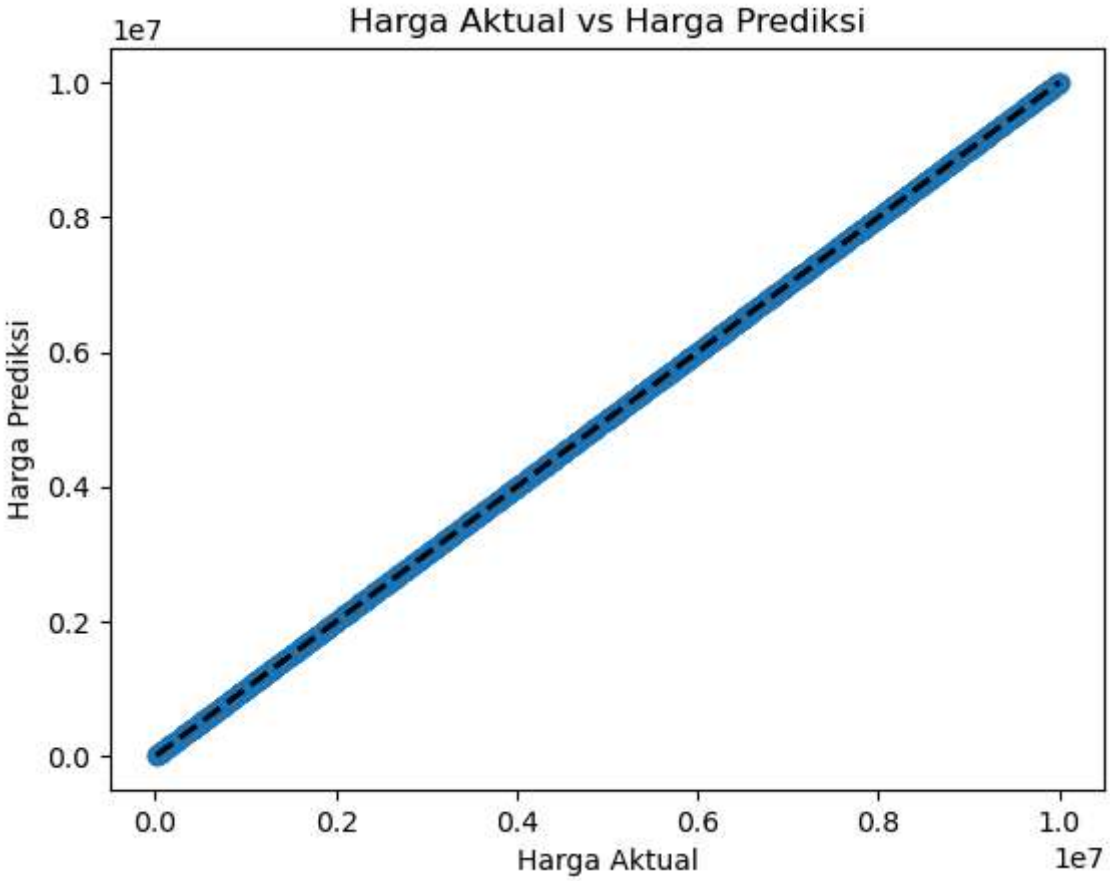
Out[14]:

	price	Ridge Prediction	Selisih_Pricew_RR
9600	2,327,859.80	2,327,383.49	-476.31
2889	8,694,297.10	8,694,486.75	189.65
2694	8,221,240.90	8,224,367.84	3,126.94
4775	3,252,838.40	3,253,471.46	633.06
5567	463,922.20	468,628.20	4,706.00

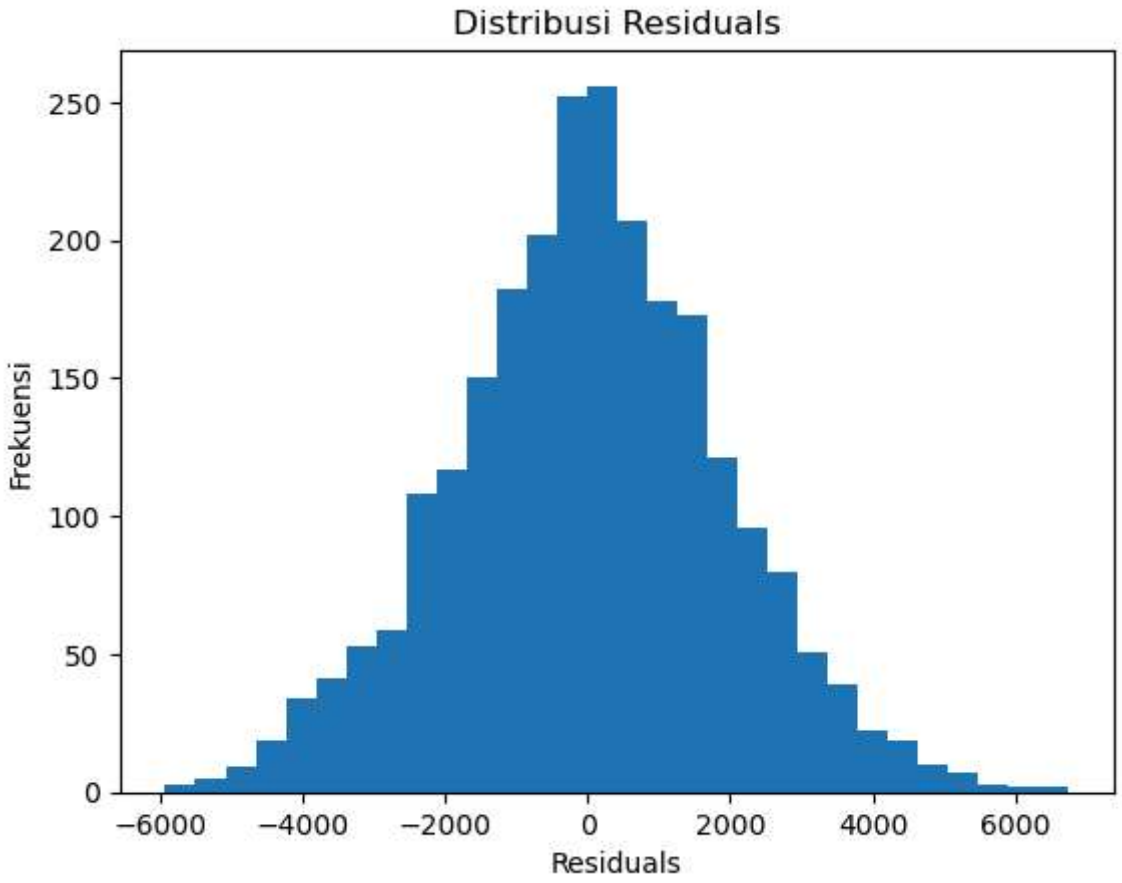
In [15]:
```python
from sklearn.metrics import r2_score
r2 = r2_score(y_test, Ridge_predict)
print("R² score:", r2)
```

```
R² score: 0.9999995680884429
```

In [16]:
```
import matplotlib.pyplot as plt

plt.scatter(y_test, Ridge_predict)
plt.xlabel("Harga Aktual")
plt.ylabel("Harga Prediksi")
plt.title("Harga Aktual vs Harga Prediksi")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2)
plt.show()
```



In [17]:
```
residuals = y_test - Ridge_predict
plt.hist(residuals, bins=30)
plt.title("Distribusi Residuals")
plt.xlabel("Residuals")
plt.ylabel("Frekuensi")
plt.show()
```



In [18]:
```
df_results.describe()
```

Out[18]:

	price	Ridge Prediction	Selisih_Pricew_RR
count	2,500.00	2,500.00	2,500.00
mean	5,070,286.87	5,070,277.08	-9.79
std	2,902,109.03	2,902,158.51	1,907.24
min	10,313.50	13,770.87	-6,742.12
25%	2,572,316.52	2,571,848.52	-1,255.05
50%	5,150,484.40	5,151,110.87	-8.71
75%	7,620,090.12	7,620,720.51	1,193.04
max	10,002,945.00	10,001,511.35	5,919.54

In [19]:
```
from sklearn.linear_model import Ridge
from sklearn.svm import SVR
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_SVR = Pipeline(steps=[
    ('scale', StandardScaler()),
```

```
    ('feat_select', SelectKBest(score_func=f_regression)),
    ('reg', SVR(kernel='linear'))  # Removed the incorrect positional argument
])

param_grid_SVR = [
    {
        'scale': [StandardScaler()],
        'reg__C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],  # Menambah range C
        'reg__epsilon': [0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5],  # Range epsilon yan
        'feat_select__k': np.arange(5, 25, 2),  # Menambah range dan step size
    },
    {
        'scale': [StandardScaler()],
        'reg__C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
        'reg__epsilon': [0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5],
        'feat_select': [SelectPercentile(score_func=f_regression)],
        'feat_select__percentile': np.arange(5, 100, 5),  # Range persentil yang le
    },
    {
        'scale': [MinMaxScaler()],
        'reg__C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
        'reg__epsilon': [0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5],
        'feat_select__k': np.arange(5, 25, 2),
    },
    {
        'scale': [MinMaxScaler()],
        'reg__C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
        'reg__epsilon': [0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5],
        'feat_select': [SelectPercentile(score_func=f_regression)],
        'feat_select__percentile': np.arange(5, 100, 5),
    }
]
# Mengimplementasikan GridSearchCV untuk SVR
GSCV_SVR = GridSearchCV(pipe_SVR, param_grid_SVR, cv=5, scoring='neg_mean_squared_e
GSCV_SVR.fit(X_train_encoded, y_train)

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

```
Fitting 5 folds for each of 2842 candidates, totalling 14210 fits
Best model:Pipeline(steps=[('scale', StandardScaler()),
                ('feat_select',
                 SelectPercentile(percentile=5,
                                  score_func=<function f_regression at 0x00000258A1F
3E160>)),
                ('reg', SVR(C=1000, epsilon=0.01, kernel='linear'))])
SVR best parameters:{'feat_select': SelectPercentile(score_func=<function f_regressi
on at 0x00000258A1F3E160>), 'feat_select__percentile': 5, 'reg__C': 1000, 'reg__epsi
lon': 0.01, 'scale': StandardScaler()}
```

In [ ]:

In [20]:
```
SVR_predict = GSCV_SVR.predict(X_test_encoded)

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

```
SVR Mean Squared Error (MSE): 25195566.632881764
SVR Mean Absolute Error (MAE): 4088.202582634997
SVR Root Mean Squared Error: 5019.5185658469045
```

In [21]:
```
svr_feature_count = GSCV_SVR.best_params_['feat_select__percentile']
print(svr_feature_count)
```

```
5
```

In [22]:
```
import pandas as pd

# Membuat DataFrame dari y_test dengan kolom 'price'
df_results = pd.DataFrame(y_test, columns=['price'])

# Menambahkan kolom untuk prediksi Ridge
df_results['SVR Prediction'] = SVR_predict

# Menambahkan kolom yang berisi selisih antara prediksi dan harga aktual
df_results['Selisih_Pricew_SVR'] = df_results['SVR Prediction'] - df_results['price

# Mengatur agar angka ditampilkan dalam format biasa
pd.set_option('display.float_format', '{:,.2f}'.format)

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

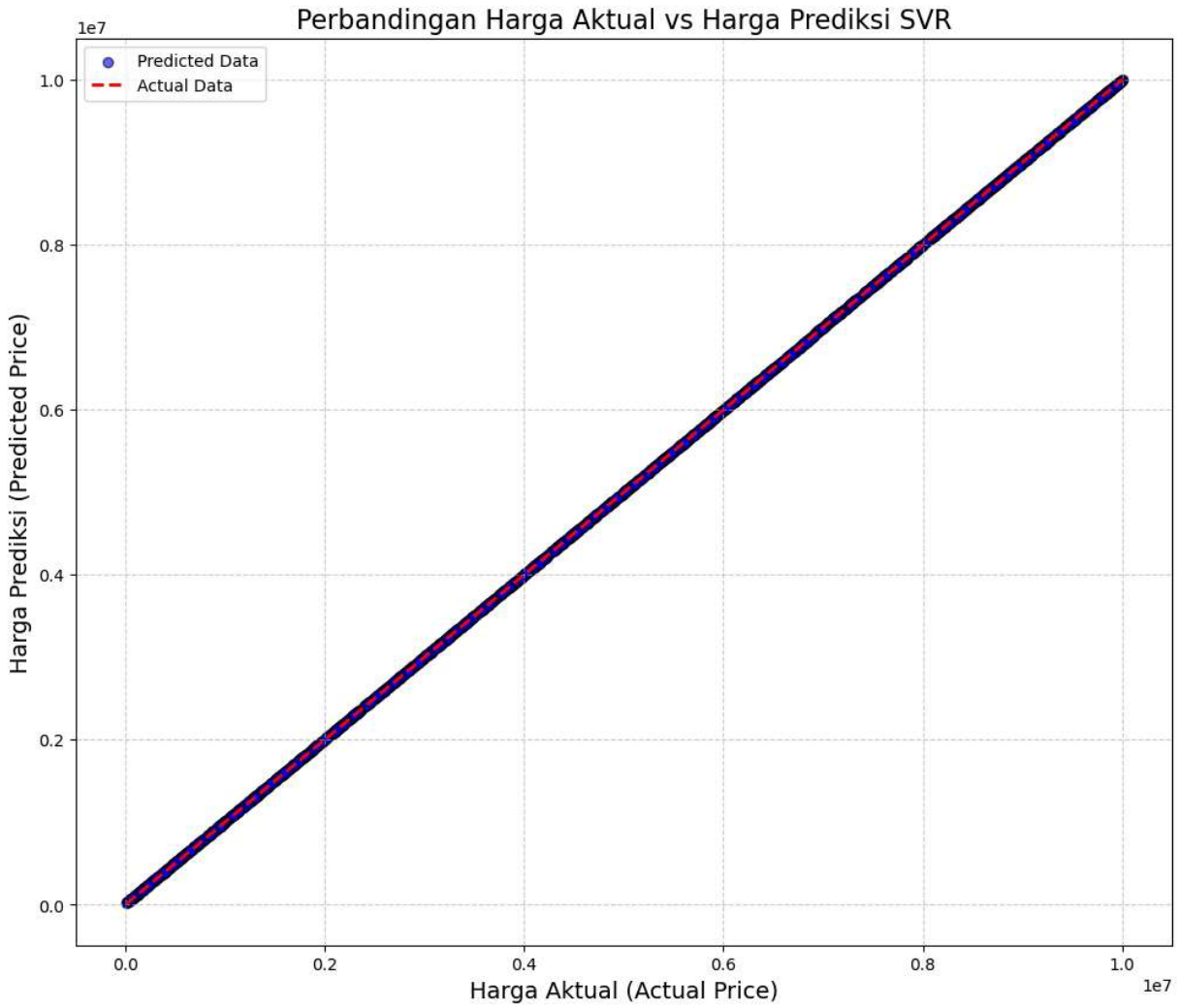
Out[22]:

	price	SVR Prediction	Selisih_Pricew_SVR
9600	2,327,859.80	2,330,072.97	2,213.17
2889	8,694,297.10	8,699,045.59	4,748.49
2694	8,221,240.90	8,225,771.14	4,530.24
4775	3,252,838.40	3,250,977.44	-1,860.96
5567	463,922.20	470,203.16	6,280.96

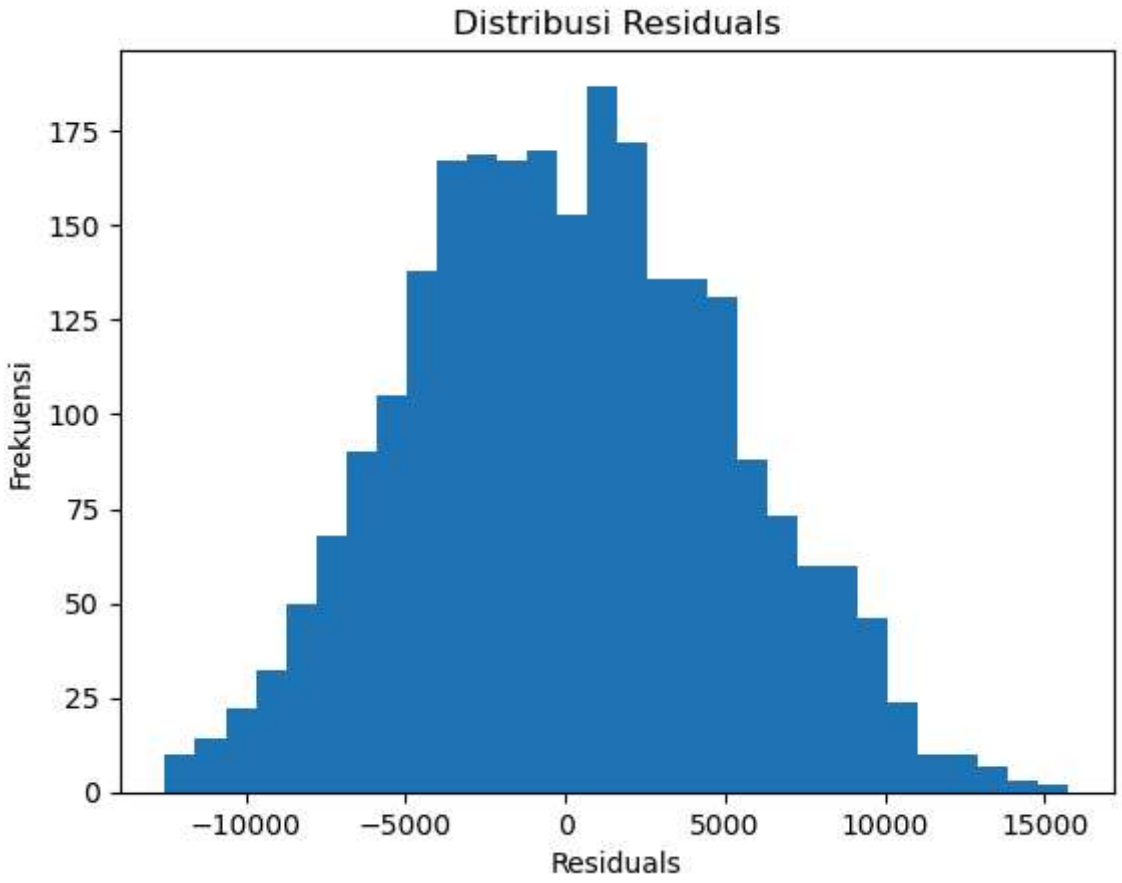
In [23]:
```
import matplotlib.pyplot as plt
import numpy as np


plt.figure(figsize=(12, 10))
plt.scatter(y_test, SVR_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 SVR", 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')
```

Out[23]: <matplotlib.legend.Legend at 0x258a20d5010>



In [30]:
```
residuals = y_test - SVR_predict
plt.hist(residuals, bins=30)
plt.title("Distribusi Residuals")
plt.xlabel("Residuals")
plt.ylabel("Frekuensi")
plt.show()
```



In [31]:
```
from sklearn.metrics import r2_score
r2 = r2_score(y_test, SVR_predict)
print("R² score:", r2)
```

```
R² score: 0.9999970072500342
```

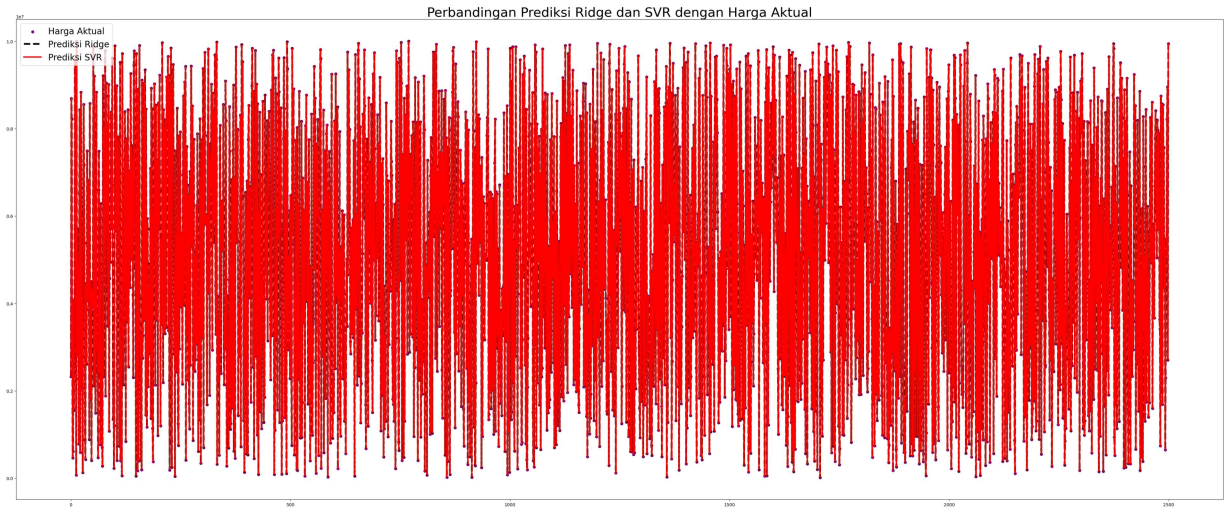
In [26]:
```
df_results = pd.DataFrame(y_test, columns=['price'])

df_results['Ridge Prediction'] = Ridge_predict
df_results['Selisih_IPK_RR'] = df_results['price'] - df_results['Ridge Prediction']
df_results['SVR Prediction'] = SVR_predict
df_results['Selisih_IPK_SVR'] = df_results['price'] - df_results['SVR Prediction']

df_results.head()
```

Out[26]:

	price	Ridge Prediction	Selisih_IPK_RR	SVR Prediction	Selisih_IPK_SVR
9600	2,327,859.80	2,327,383.49	476.31	2,330,072.97	-2,213.17
2889	8,694,297.10	8,694,486.75	-189.65	8,699,045.59	-4,748.49
2694	8,221,240.90	8,224,367.84	-3,126.94	8,225,771.14	-4,530.24
4775	3,252,838.40	3,253,471.46	-633.06	3,250,977.44	1,860.96
5567	463,922.20	468,628.20	-4,706.00	470,203.16	-6,280.96

In [32]:
```
import matplotlib.pyplot as plt
```

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

data_lens = range(len(y_test))

plt.scatter(data_lens, df_results['price'],label='Harga Aktual', color='purple')

plt.plot(data_lens, df_results['Ridge Prediction'], label='Prediksi Ridge', linesty

plt.plot(data_lens, df_results['SVR Prediction'], label='Prediksi SVR', linestyle='

plt.legend(loc='upper left',prop={'size': 20})

plt.title('Perbandingan Prediksi Ridge dan SVR dengan Harga Aktual', fontsize=30)

plt.show()
```



In [33]:
```
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 Pred

ridge_feature_count = GSCV_RR.best_params_['feat_select__percentile']

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 Predicti

svr_feature_count = GSCV_SVR.best_params_['feat_select__percentile']



print(f"Ridge MAE: {mae_Ridge}, Ridge RMSE: {rmse_Ridge}, Ridge Feature Count: {rid
print(f"SVR MAE: {mae_SVR}, SVR RMSE: {rmse_SVR}, SVR Feature Count(percentile): {s
```

```
r2 = r2_score(y_test, Ridge_predict)
print("R² Ridge score:", r2)
r2 = r2_score(y_test, SVR_predict)
print("R² SVR score:", r2)
```

```
Ridge MAE: 1491.0326240901168, Ridge RMSE: 1906.883936024045, Ridge Feature Count: 9
0
SVR MAE: 4088.202582634997, SVR RMSE: 5019.5185658469045, SVR Feature Count(percenti
le): 5
R² Ridge score: 0.9999995680884429
R² SVR score: 0.9999970072500342
```

In [35]:
```
print('Best model is Ridge with MAE: ',mae_Ridge, 'and RMSE: ',rmse_Ridge)
import pickle
with open('Model_Ridge.pkl', 'wb') as file:
    pickle.dump(GSCV_RR, file)
```

```
Best model is Ridge with MAE:  1491.0326240901168 and RMSE:  1906.883936024045
```

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

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_directory = r'E:\clone\Project'
    model_path = os.path.join(model_directory, r'BestModel_CLF_RFC_NamaSB.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)')
with col2:
    price = st.number_input('💰 Harga (dalam mata uang yang sesuai)', 0.0)

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, price]]


    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_directory = r'E:\clone\Project'
  model_path = os.path.join(model_directory, r'BestModel_REG_Ridge_NamaSB.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)')
with col2:
    category = st.selectbox('Kategori Properti', ['Basic', 'Middle', 'Luxury'])




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
    if category == 'Basic':
        onehotencoder__category_Basic = 1
        onehotencoder__category_Luxury = 0
        onehotencoder__category_Middle = 0
    elif category == 'Middle':
        onehotencoder__category_Basic = 0
        onehotencoder__category_Luxury = 0
        onehotencoder__category_Middle = 1
    elif category == 'Luxury':
        onehotencoder__category_Basic = 0
        onehotencoder__category_Luxury = 1
        onehotencoder__category_Middle = 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,
onehotencoder__category_Basic, onehotencoder__category_Luxury,
onehotencoder__category_Middle,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.'')
```