Klasifikasi:

- Klasifikasi Random Forest dengan Logistic Regression
- Sebagai Metode Klasfikasi terbaik digunakan Logistic Regression

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

df_property=pd.read_csv(r'C:\Users\lenovo\Downloads\Dataset UTS_Gasal 2425.csv')
df_property.head(10)
```

Out[1]:

	squaremeters	numberofrooms	hasyard	haspool	floors	citycode	citypartrange	numprevowners	made	isnewbuilt	hasstormp
0	75523	3	no	yes	63	9373	3	8	2005	old	
1	55712	58	no	yes	19	34457	6	8	2021	old	
2	86929	100	yes	no	11	98155	3	4	2003	new	
3	51522	3	no	no	61	9047	8	3	2012	new	
4	96470	74	yes	no	21	92029	4	2	2011	new	
5	79770	3	no	yes	69	54812	10	5	2018	old	
6	75985	60	yes	no	67	6517	6	9	2009	new	
7	64169	88	no	yes	6	61711	3	9	2011	new	
8	92383	12	no	no	78	71982	3	7	2000	old	
9	95121	46	no	yes	3	9382	7	9	1994	old	

In [2]:
```
df_property2=df_property.drop('price', axis=1)
df_property2.head(10)
```

Out[2]:

	squaremeters	numberofrooms	hasyard	haspool	floors	citycode	citypartrange	numprevowners	made	isnewbuilt	hasstormp
0	75523	3	no	yes	63	9373	3	8	2005	old	
1	55712	58	no	yes	19	34457	6	8	2021	old	
2	86929	100	yes	no	11	98155	3	4	2003	new	
3	51522	3	no	no	61	9047	8	3	2012	new	
4	96470	74	yes	no	21	92029	4	2	2011	new	
5	79770	3	no	yes	69	54812	10	5	2018	old	
6	75985	60	yes	no	67	6517	6	9	2009	new	
7	64169	88	no	yes	6	61711	3	9	2011	new	
8	92383	12	no	no	78	71982	3	7	2000	old	
9	95121	46	no	yes	3	9382	7	9	1994	old	

In [3]:
```
df_property2.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 [4]:
```
df_property2.describe()
```

Out[4]:

	squaremeters	numberofrooms	floors	citycode	citypartrange	numprevowners	made	basement	
count	10000.00000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	1000
mean	49870.13120	50.358400	50.276300	50225.486100	5.510100	5.521700	2005.48850	5033.103900	502
std	28774.37535	28.816696	28.889171	29006.675799	2.872024	2.856667	9.30809	2876.729545	289
min	89.00000	1.000000	1.000000	3.000000	1.000000	1.000000	1990.00000	0.000000	
25%	25098.50000	25.000000	25.000000	24693.750000	3.000000	3.000000	1997.00000	2559.750000	25
50%	50105.50000	50.000000	50.000000	50693.000000	5.000000	5.000000	2005.50000	5092.500000	504
75%	74609.75000	75.000000	76.000000	75683.250000	8.000000	8.000000	2014.00000	7511.250000	754
max	99999.00000	100.000000	100.000000	99953.000000	10.000000	10.000000	2021.00000	10000.000000	1000

In [5]:
```
print("data null \n", df_property2.isnull().sum())
print("data kosong \n", df_property2.empty)
print("data nan \n", df_property2.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 [6]:

```
from sklearn.model_selection import train_test_split

x = df_property2.drop(columns=['category'], axis=1)
y = df_property2['category']

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.25, random_state=72)

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

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

In [7]:

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

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

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

In [8]:

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

df_train_enc_x = pd.DataFrame(x_train_enc, columns=transform.get_feature_names_out())
df_test_enc_x = pd.DataFrame(x_test_enc, columns=transform.get_feature_names_out())

df_train_enc_x.head(10)
df_test_enc_x.head(10)
```

Out[8]:

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

10 rows × 21 columns

In [9]:

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

y_train = pd.DataFrame(y_train, columns=['category'])
y_test = pd.DataFrame(y_test, columns=['category'])

y_train = y_train.values.ravel()
y_test = y_test.values.ravel()

encoder = OneHotEncoder()
y_train_enc = encoder.fit_transform(y_train.reshape(-1, 1)).toarray()
y_test_enc = encoder.transform(y_test.reshape(-1, 1)).toarray()

df_train_enc = pd.DataFrame(y_train_enc, columns=encoder.get_feature_names_out())
df_test_enc = pd.DataFrame(y_test_enc, columns=encoder.get_feature_names_out())
```

In [10]:

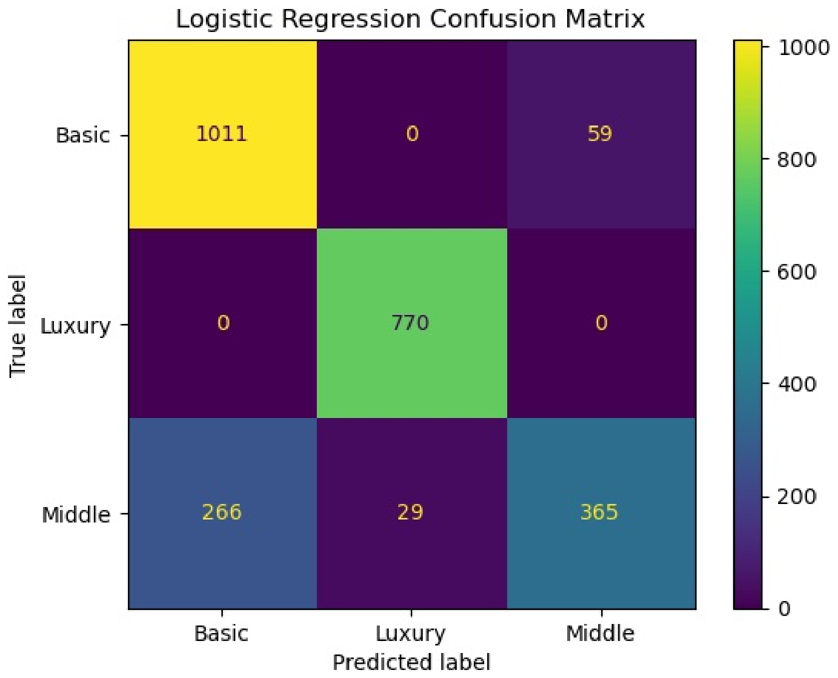
```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.feature_selection import SelectKBest, SelectPercentile
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

LR = LogisticRegression(solver='liblinear', max_iter=500)

pipe_LR = Pipeline(steps=[
    ('data scaling', StandardScaler()),
    ('feature select', SelectKBest()),
    ('clf', LR)
])

params_LR = [
    {
        'data scaling': [StandardScaler()],
        'feature select__k': np.arange(2, 6),
        'clf__C': [0.1, 1, 10],
        'clf__penalty': ['l2']
    },
    {
        'data scaling': [StandardScaler()],
        'feature select': [SelectPercentile()],
        'feature select__percentile': np.arange(20, 50),
        'clf__C': [0.1, 1, 10],
        'clf__penalty': ['l2']
    },
    {
        'data scaling': [MinMaxScaler()],
        'feature select__k': np.arange(2, 6),
        'clf__C': [0.1, 1, 10],
        'clf__penalty': ['l2']
    },
    {
        'data scaling': [MinMaxScaler()],
        'feature select': [SelectPercentile()],
        'feature select__percentile': np.arange(20, 50),
        'clf__C': [0.1, 1, 10],
        'clf__penalty': ['l2']
    }
```

```
]

GSCV_LR = GridSearchCV(pipe_LR, params_LR, cv=5)

GSCV_LR.fit(x_train_enc, y_train)
print("GSCV training finished")
```

```
GSCV training finished
```

In [11]:
```
print("Best CV Score: {}".format(GSCV_LR.best_score_))
print("Test Score: {}".format(GSCV_LR.score(x_test_enc, y_test)))
print("Best Model: ", GSCV_LR.best_estimator_)

LR_pred = GSCV_LR.predict(x_test_enc)

cm_LR = confusion_matrix(y_test, LR_pred, labels=GSCV_LR.classes_)
disp_LR = ConfusionMatrixDisplay(confusion_matrix=cm_LR, display_labels=GSCV_LR.classes_)
disp_LR.plot()
plt.title("Logistic Regression Confusion Matrix")
plt.show()

print("Classification Report for Logistic Regression: \n", classification_report(y_test, LR_pred))
```

```
Best CV Score: 0.8677333333333334
Test Score: 0.8584
Best Model:  Pipeline(steps=[('data scaling', StandardScaler()),
                ('feature select', SelectKBest(k=4)),
                ('clf',
                 LogisticRegression(C=10, max_iter=500, solver='liblinear'))])
```




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

       Basic       0.79      0.94      0.86      1070
      Luxury       0.96      1.00      0.98       770
      Middle       0.86      0.55      0.67       660

    accuracy                           0.86      2500
   macro avg       0.87      0.83      0.84      2500
weighted avg       0.86      0.86      0.85      2500
```

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

pipe_RF=[
    ('data scaling', StandardScaler()),
    ('feature select', SelectKBest()),
    ('clf', RandomForestClassifier(random_state=5, class_weight='balanced'))
]

params_grid_RF = [
    {
```

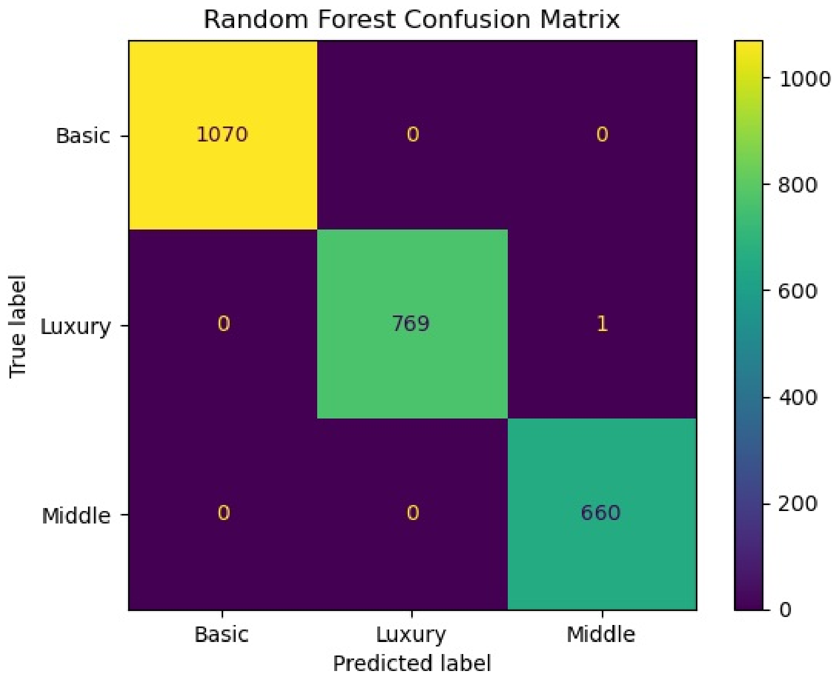
```
        'data scaling' : [StandardScaler()],
        'feature select__k' : np.arange(2,6),
        'clf__max_depth' : np.arange(4,5),
        'clf__n_estimators' : [100, 150]
    },
    {
        'data scaling' : [StandardScaler()],
        'feature select' : [SelectPercentile()],
        'feature select__percentile' : np.arange(20,50),
        'clf__max_depth' : np.arange(4,5),
        'clf__n_estimators' : [100, 150]
    },
    {
        'data scaling' : [MinMaxScaler()],
        'feature select__k' : np.arange(2,6),
        'clf__max_depth' : np.arange(4,5),
        'clf__n_estimators' : [100, 150]
    },
    {
        'data scaling' : [MinMaxScaler()],
        'feature select' : [SelectPercentile()],
        'feature select__percentile' : np.arange(20,50),
        'clf__max_depth' : np.arange(4,5),
        'clf__n_estimators' : [100, 150]
    }
]

estimator_RF = Pipeline(pipe_RF)

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

GSCV_RF = GridSearchCV(estimator_RF, params_grid_RF, cv=SKF, n_jobs=-1)

GSCV_RF.fit(x_train_enc, y_train)
print("GSCV training finished")
```

```
GSCV training finished
```

In [13]:

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

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

print("Best Mode: ", GSCV_RF.best_estimator_)

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

RF_pred = GSCV_RF.predict(x_test_enc)

import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, RF_pred, labels=GSCV_RF.classes_)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=GSCV_RF.classes_)
disp.plot()
plt.title("Random Forest Confusion Matrix")
plt.show()

print("CLassification Report RF: \n", classification_report(y_test, RF_pred))
```

```
CV Score: 0.9997333333333334
Test Score: 0.9996
Best Mode:  Pipeline(steps=[('data scaling', MinMaxScaler()),
                ('feature select', SelectPercentile(percentile=46)),
                ('clf',
                 RandomForestClassifier(class_weight='balanced', max_depth=4,
                                        n_estimators=150, random_state=5))])
Best Features:  Index(['onehotencoder__hasyard_no', 'onehotencoder__hasyard_yes',
       'onehotencoder__haspool_no', 'onehotencoder__haspool_yes',
       'onehotencoder__isnewbuilt_new', 'onehotencoder__isnewbuilt_old',
       'remainder__squaremeters', 'remainder__numberofrooms',
       'remainder__citycode', 'remainder__numprevowners'],
      dtype='object')
```




```
CLassification Report RF:
              precision    recall  f1-score   support

       Basic       1.00      1.00      1.00      1070
      Luxury       1.00      1.00      1.00       770
      Middle       1.00      1.00      1.00       660

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

In [14]:

```
import pickle

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

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

```
Model LR berhasil disimpan
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

KLASIFIKASI : Algoritma berbasis linear

- Support Vector Machine
- Gradient Boosting Classifier

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

df_properti=pd.read_csv(r'D:\Kuliah\SEM 5\ml\UTS\UTS PMDPM_A_H2O\Dataset UTS_Gasal 2425.csv')
df_properti.head(10)
```

Out[3]:

	squaremeters	numberofrooms	hasyard	haspool	floors	citycode	citypartrange	numprevowners	made	isnewbuilt	hasstormp
0	75523	3	no	yes	63	9373	3	8	2005	old	
1	55712	58	no	yes	19	34457	6	8	2021	old	
2	86929	100	yes	no	11	98155	3	4	2003	new	
3	51522	3	no	no	61	9047	8	3	2012	new	
4	96470	74	yes	no	21	92029	4	2	2011	new	
5	79770	3	no	yes	69	54812	10	5	2018	old	
6	75985	60	yes	no	67	6517	6	9	2009	new	
7	64169	88	no	yes	6	61711	3	9	2011	new	
8	92383	12	no	no	78	71982	3	7	2000	old	
9	95121	46	no	yes	3	9382	7	9	1994	old	

In [4]:
```python
df_properti2=df_properti.drop('price',axis=1)
df_properti2.head(10)
```

Out[4]:

	squaremeters	numberofrooms	hasyard	haspool	floors	citycode	citypartrange	numprevowners	made	isnewbuilt	hasstormp
0	75523	3	no	yes	63	9373	3	8	2005	old	
1	55712	58	no	yes	19	34457	6	8	2021	old	
2	86929	100	yes	no	11	98155	3	4	2003	new	
3	51522	3	no	no	61	9047	8	3	2012	new	
4	96470	74	yes	no	21	92029	4	2	2011	new	
5	79770	3	no	yes	69	54812	10	5	2018	old	
6	75985	60	yes	no	67	6517	6	9	2009	new	
7	64169	88	no	yes	6	61711	3	9	2011	new	
8	92383	12	no	no	78	71982	3	7	2000	old	
9	95121	46	no	yes	3	9382	7	9	1994	old	

In [5]:
```python
df_properti2.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 [6]:

```
df_properti2.describe()
```

Out[6]:

	squaremeters	numberofrooms	floors	citycode	citypartrange	numprevowners	made	basement	
count	10000.00000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	1000
mean	49870.13120	50.358400	50.276300	50225.486100	5.510100	5.521700	2005.48850	5033.103900	502
std	28774.37535	28.816696	28.889171	29006.675799	2.872024	2.856667	9.30809	2876.729545	28
min	89.00000	1.000000	1.000000	3.000000	1.000000	1.000000	1990.00000	0.000000	
25%	25098.50000	25.000000	25.000000	24693.750000	3.000000	3.000000	1997.00000	2559.750000	25
50%	50105.50000	50.000000	50.000000	50693.000000	5.000000	5.000000	2005.50000	5092.500000	504
75%	74609.75000	75.000000	76.000000	75683.250000	8.000000	8.000000	2014.00000	7511.250000	754
max	99999.00000	100.000000	100.000000	99953.000000	10.000000	10.000000	2021.00000	10000.000000	1000

In [7]:

```
print("data null \n", df_properti2.isnull().sum())
print("data kosong \n", df_properti2.empty)
print("data nan \n", df_properti2.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 [8]:

```
from sklearn.model_selection import train_test_split
x = df_properti2.drop(columns=['category'],axis=1)
y = y=df_properti2['category']

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.25,random_state=72)

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

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

In [9]:

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

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

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

In [10]:
```
x_train_enc=transform.fit_transform(x_train)
x_test_enc=transform.fit_transform(x_test)

df_x_train_enc=pd.DataFrame(x_train_enc,columns=transform.get_feature_names_out())
df_x_test_enc=pd.DataFrame(x_test_enc,columns=transform.get_feature_names_out())

df_x_train_enc.head(10)
df_x_test_enc.head(10)
```

Out[10]:

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

10 rows × 21 columns

In [11]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer

y_train = pd.DataFrame(y_train, columns=['category'])
y_test = pd.DataFrame(y_test, columns=['category'])

kolom_kategori=['category']

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

In [12]:
```
y_train_enc=transform.fit_transform(y_train)
y_test_enc=transform.fit_transform(y_test)

df_train_enc=pd.DataFrame(y_train_enc, columns=transform.get_feature_names_out())
df_test_enc=pd.DataFrame(y_test_enc, columns=transform.get_feature_names_out())

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

Out[12]:

	onehotencoder__category_Basic	onehotencoder__category_Luxury	onehotencoder__category_Middle
0	1.0	0.0	0.0
1	0.0	0.0	1.0
2	0.0	1.0	0.0
3	1.0	0.0	0.0
4	0.0	1.0	0.0
5	1.0	0.0	0.0
6	1.0	0.0	0.0
7	0.0	0.0	1.0
8	1.0	0.0	0.0
9	1.0	0.0	0.0

In [13]:
```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.feature_selection import SelectPercentile, SelectKBest
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay

pipe_svm = Pipeline(steps=[
    ('scale', MinMaxScaler()),
    ('feat_select', SelectKBest()),
```

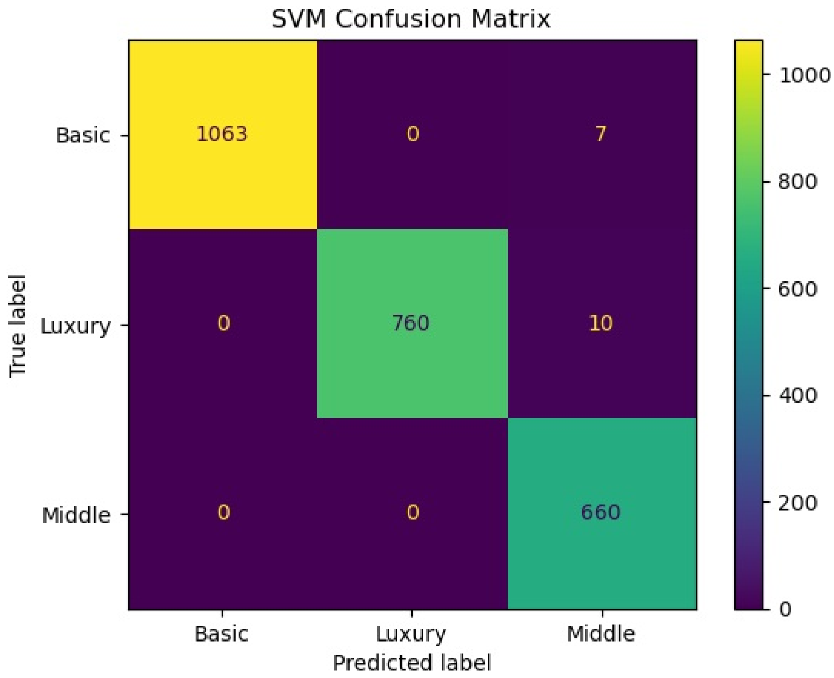
```python
    ('clf', SVC(class_weight='balanced'))
])

params_grid_svm = [
    {
        'scale':[MinMaxScaler()],
        'feat_select': [SelectKBest()],
        'feat_select__k':np.arange(2,6),
        'clf__kernel':['poly', 'rbf'],
        'clf__C':[0.1,1],
        'clf__gamma':[0.1,1]
    },
    {
        'scale':[MinMaxScaler()],
        'feat_select':[SelectPercentile()],
        'feat_select__percentile':np.arange(20,50),
        'clf__kernel':['poly', 'rbf'],
        'clf__C':[0.1,1],
        'clf__gamma':[0.1,1]
    },
    {
        'scale':[StandardScaler()],
        'feat_select': [SelectKBest()],
        'feat_select__k':np.arange(2,6),
        'clf__kernel':['poly', 'rbf'],
        'clf__C':[0.1,1],
        'clf__gamma':[0.1,1]
    },
    {
        'scale':[StandardScaler()],
        'feat_select':[SelectPercentile()],
        'feat_select__percentile':np.arange(20,50),
        'clf__kernel':['poly', 'rbf'],
        'clf__C':[0.1,1],
        'clf__gamma':[0.1,1]
    },
]

estimator_svm = Pipeline(pipe_svm)
SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=72)e

GSCV_SVM = GridSearchCV(pipe_svm, params_grid_svm, cv=SKF, n_jobs=-1)
GSCV_SVM.fit(x_train_enc, y_train.values.ravel())

print("GSCV training finished")
```

```
GSCV training finished
```

In [20]:

```python
print("CV Score: {}".format(GSCV_SVM.best_score_))
print("Test Score: {}".format(GSCV_SVM.best_estimator_.score(x_test_enc, y_test)))
print("Best Model: {}", GSCV_SVM.best_estimator_)
mask = GSCV_SVM.best_estimator_.named_steps['feat_select'].get_support()
print("Best Features: {}", df_x_train_enc.columns[mask])

SVM_pred = GSCV_SVM.predict(x_test_enc)

import matplotlib.pyplot as plt
cm = confusion_matrix(y_test, SVM_pred, labels=GSCV_SVM.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=GSCV_SVM.classes_)
disp.plot()
plt.title("SVM Confusion Matrix")
plt.show()

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

```
CV Score: 0.9930666666666668
Test Score: 0.9932
Best Model: {} Pipeline(steps=[('scale', StandardScaler()),
                ('feat_select', SelectPercentile(percentile=36)),
                ('clf',
                 SVC(C=1, class_weight='balanced', gamma=1, kernel='poly'))])
Best Features: {} Index(['onehotencoder__hasyard_no', 'onehotencoder__hasyard_yes',
       'onehotencoder__haspool_no', 'onehotencoder__haspool_yes',
       'onehotencoder__isnewbuilt_new', 'onehotencoder__isnewbuilt_old',
       'remainder__squaremeters', 'remainder__numberofrooms'],
      dtype='object')
```




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

       Basic       1.00      0.99      1.00      1070
      Luxury       1.00      0.99      0.99       770
      Middle       0.97      1.00      0.99       660

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

In [23]:

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

pipe_GBT = Pipeline(steps=[
    ('scale', MinMaxScaler()),
    ('feat_select', SelectKBest()),
    ('clf',GradientBoostingClassifier(random_state=72))])

params_grid_GBT = [
    {
        'scale': [MinMaxScaler()],
        'feat_select': [SelectKBest()],
        'feat_select__k':np.arange(2,6),
        'clf__max_depth':[*np.arange(3,4)],
        'clf__n_estimators':[100,150],
        'clf__learning_rate':[0.01,0.1,1]
    },
    {
        'scale': [MinMaxScaler()],
        'feat_select': [SelectPercentile()],
        'feat_select__percentile':np.arange(20,50),
        'clf__max_depth':[*np.arange(3,4)],
        'clf__n_estimators':[100,150],
        'clf__learning_rate':[0.01,0.1,1]
    },
    {
        'scale': [StandardScaler()],
        'feat_select': [SelectKBest()],
        'feat_select__k':np.arange(2,6),
        'clf__max_depth':[*np.arange(3,4)],
        'clf__n_estimators':[100,150],
```

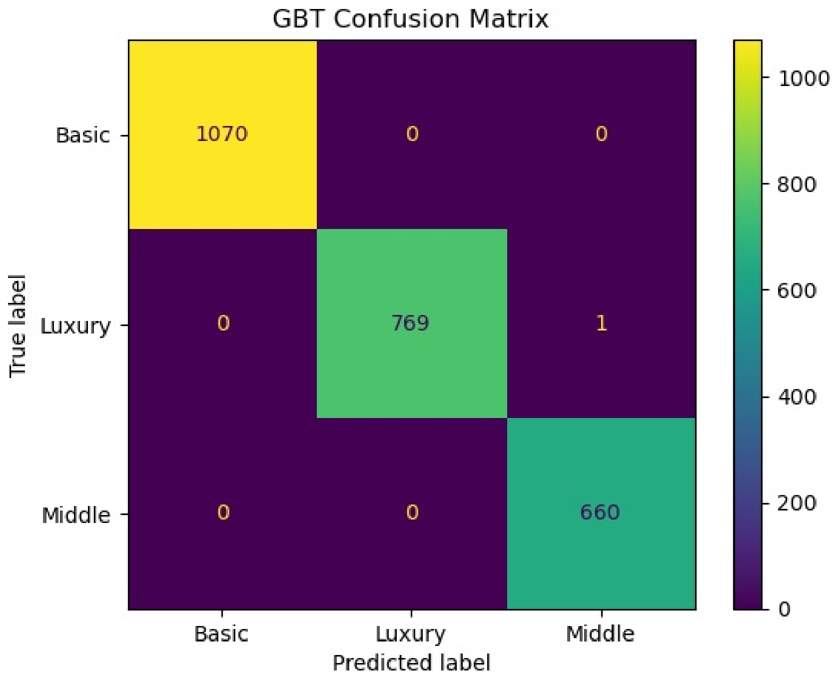
```
        'clf__learning_rate':[0.01,0.1,1]
    },
    {
        'scale': [StandardScaler()],
        'feat_select': [SelectPercentile()],
        'feat_select__percentile':np.arange(20,50),
        'clf__max_depth':[*np.arange(3,4)],
        'clf__n_estimators':[100,150],
        'clf__learning_rate':[0.01,0.1,1]
    },
]

GSCV_GBT = GridSearchCV(pipe_GBT,params_grid_GBT,cv=StratifiedKFold(n_splits=5),n_jobs=-1)
GSCV_GBT.fit(x_train_enc,y_train.values.ravel())

print("GSCV Finished")
```

```
GSCV Finished
```

In [24]:

```
print("CV Score: {}".format(GSCV_GBT.best_score_))
print("Test Score: {}".format(GSCV_GBT.best_estimator_.score(x_test_enc, y_test)))
print("Best Model: {}", GSCV_GBT.best_estimator_)
mask = GSCV_GBT.best_estimator_.named_steps['feat_select'].get_support()
print("Best Features: {}", df_x_train_enc.columns[mask])

GBT_pred = GSCV_GBT.predict(x_test_enc)

import matplotlib.pyplot as plt
cm = confusion_matrix(y_test, GBT_pred, labels=GSCV_GBT.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=GSCV_GBT.classes_)
disp.plot()
plt.title("GBT Confusion Matrix")
plt.show()

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

```
CV Score: 0.9996
Test Score: 0.9996
Best Model: {} Pipeline(steps=[('scale', MinMaxScaler()),
                ('feat_select', SelectPercentile(percentile=41)),
                ('clf', GradientBoostingClassifier(random_state=72))])
Best Features: {} Index(['onehotencoder__hasyard_no', 'onehotencoder__hasyard_yes',
       'onehotencoder__haspool_no', 'onehotencoder__haspool_yes',
       'onehotencoder__isnewbuilt_new', 'onehotencoder__isnewbuilt_old',
       'remainder__squaremeters', 'remainder__numberofrooms',
       'remainder__numprevowners'],
      dtype='object')
```



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

       Basic       1.00      1.00      1.00      1070
      Luxury       1.00      1.00      1.00       770
      Middle       1.00      1.00      1.00       660

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

In [26]:

```
import pickle

with open('SVM_properti_model.pkl', 'wb') as r:
    pickle.dump((GSCV_SVM),r)

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

```
Model SVR berhasil disimpan
```

REGRESI : Algoritma berbasis linear regression

- Ridge Regression
- Support Vector Regressor

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

df_properti = pd.read_csv(r'D:\Kuliah\SEMESTER 5\ML\UTS\Projek UTS PMDPM_A_H2O\Dataset UTS_Gasal 2425.csv')
df_properti.head(10)
```

Out[1]:

	squaremeters	numberofrooms	hasyard	haspool	floors	citycode	citypartrange	numprevowners	made	isnewbuilt	hasstormprotector	basement	attic	garage	hasstorageroom	hasguestroom	price	category
0	75523	3	no	yes	63	9373	3	8	2005	old	yes	4313	9005	956	no	7	7559081.5	Luxury
1	55712	58	no	yes	19	34457	6	8	2021	old	no	2937	8852	135	yes	9	5574642.1	Middle
2	86929	100	yes	no	11	98155	3	4	2003	new	no	6326	4748	654	no	10	8696869.3	Luxury
3	51522	3	no	no	61	9047	8	3	2012	new	yes	632	5792	807	yes	5	5154055.2	Middle
4	96470	74	yes	no	21	92029	4	2	2011	new	yes	5414	1172	716	yes	9	9652258.1	Luxury
5	79770	3	no	yes	69	54812	10	5	2018	old	yes	8871	7117	240	no	7	7986665.8	Luxury
6	75985	60	yes	no	67	6517	6	9	2009	new	yes	4878	281	384	yes	5	7607322.9	Luxury
7	64169	88	no	yes	6	61711	3	9	2011	new	yes	3054	129	726	no	9	6420823.1	Middle
8	92383	12	no	no	78	71982	3	7	2000	old	no	7507	9056	892	yes	1	9244344.0	Luxury
9	95121	46	no	yes	3	9382	7	9	1994	old	no	615	1221	328	no	10	9515440.4	Luxury

In [2]:
```python
df_properti2 = df_properti.drop(['category'], axis=1)
df_properti2.head(10)
```

Out[2]:

	squaremeters	numberofrooms	hasyard	haspool	floors	citycode	citypartrange	numprevowners	made	isnewbuilt	hasstormprotector	basement	attic	garage	hasstorageroom	hasguestroom	price
0	75523	3	no	yes	63	9373	3	8	2005	old	yes	4313	9005	956	no	7	7559081.5
1	55712	58	no	yes	19	34457	6	8	2021	old	no	2937	8852	135	yes	9	5574642.1
2	86929	100	yes	no	11	98155	3	4	2003	new	no	6326	4748	654	no	10	8696869.3
3	51522	3	no	no	61	9047	8	3	2012	new	yes	632	5792	807	yes	5	5154055.2
4	96470	74	yes	no	21	92029	4	2	2011	new	yes	5414	1172	716	yes	9	9652258.1
5	79770	3	no	yes	69	54812	10	5	2018	old	yes	8871	7117	240	no	7	7986665.8
6	75985	60	yes	no	67	6517	6	9	2009	new	yes	4878	281	384	yes	5	7607322.9
7	64169	88	no	yes	6	61711	3	9	2011	new	yes	3054	129	726	no	9	6420823.1
8	92383	12	no	no	78	71982	3	7	2000	old	no	7507	9056	892	yes	1	9244344.0
9	95121	46	no	yes	3	9382	7	9	1994	old	no	615	1221	328	no	10	9515440.4

In [3]:
```python
df_properti2.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 [4]:
```python
df_properti2.describe()
```

Out[4]:

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

In [5]:
```python
print("data null \n", df_properti2.isnull().sum())
print("\ndata kosong \n", df_properti2.empty)
print("\ndata nan \n", df_properti2.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 [6]:
```python
print("Sebelum Pengecekan data duplikat, ", df_properti2.shape)
df_properti3=df_properti2.drop_duplicates(keep='last')
print("Setelah Pengecekan data duplikat, ", df_properti3.shape)
```

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

In [15]:
```python
from sklearn.model_selection import train_test_split

X_regress = df_properti3.drop(columns=['price'], axis=1)
y_regress = df_properti3['price']

X_train_properti, X_test_properti, y_train_properti, y_test_properti = train_test_split(X_regress, y_regress, test_size=0.25, random_state=72)

print(X_train_properti.shape)
print(X_test_properti.shape)
print(y_train_properti.shape)
print(y_test_properti.shape)
```

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

In [8]:
```python
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer

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

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

In [10]:
```python
x_train_enc=transform.fit_transform(X_train_properti)
x_test_enc=transform.transform(X_test_properti)

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

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

Out[10]:

	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__hasstorageroo
0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	1.0	
1	0.0	1.0	0.0	1.0	1.0	0.0	1.0	0.0	
2	0.0	1.0	1.0	0.0	0.0	1.0	0.0	1.0	
3	1.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	
4	1.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	
5	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	
6	0.0	1.0	1.0	0.0	0.0	1.0	0.0	1.0	
7	1.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	
8	1.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	
9	1.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	

10 rows × 21 columns

In [17]:
```python
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.metrics import mean_absolute_error, mean_squared_error

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

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

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

GSCV_RR.fit(x_train_enc, y_train_properti)

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

Ridge_predict = GSCV_RR.predict(x_test_enc)

mse_Ridge = mean_squared_error(y_test_properti, Ridge_predict)
mae_Ridge = mean_absolute_error(y_test_properti, 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 (RMSE):{}".format(np.sqrt(mse_Ridge)))
```

```
Best model:Pipeline(steps=[('scale', StandardScaler()),
                ('feature_selection',
                 SelectKBest(k=18,
                             score_func=<function f_regression at 0x000002842B5A8FE0>)),
                ('reg', Ridge(alpha=0.01))])
Ridge best parameters:{'feature_selection__k': 18, 'reg__alpha': 0.01}
Koefisien/bobot:[-7.58021442e+02  7.58021441e+02 -7.49582654e+02  7.49582654e+02
  4.88960039e+01 -4.88960363e+01 -3.03938405e+01  3.03938405e+01
 -4.34626405e+00  4.34623239e+00  2.87646731e+06  1.56729667e+03
 -1.19875081e+01  1.43252853e+02 -6.69899153e+00 -4.83480756e+00
  2.48777674e+01 -8.35727399e+00]
Intercept/bias:4989238.000666667
Ridge Mean Squared Error (MSE):3667897.5327414596
Ridge Mean Absolute Error (MAE):1483.6207754128613
Ridge Root Mean Squared Error (RMSE):1915.1755879661425
```

In [18]:
```python
df_results = pd.DataFrame(y_test_properti, columns=['price'])
df_results = pd.DataFrame(y_test_properti)
df_results['Ridge Prediction'] = Ridge_predict

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

df_results.head(10)
```

Out[18]:

	price	Ridge Prediction	Selisih_Price_RR
7619	2714670.6	2.716003e+06	1332.150296
5479	5714000.4	5.716484e+06	2483.645076
1356	7274269.9	7.274709e+06	438.618820
7560	4822319.0	4.824374e+06	2055.308907
3551	8780796.5	8.784101e+06	3304.056988
6215	743576.1	7.447270e+05	1150.850540
9718	4807525.4	4.806934e+06	-591.329495
2504	6647572.6	6.646035e+06	-1537.626336
9152	3957780.9	3.956017e+06	-1763.436764
8110	4661805.0	4.658478e+06	-3326.718218

In [19]:
```python
df_results.describe()
```

Out[19]:

	price	Ridge Prediction	Selisih_Price_RR
count	2.500000e+03	2.500000e+03	2500.000000
mean	5.006076e+06	5.006065e+06	-10.629318
std	2.880289e+06	2.880284e+06	1915.529235
min	1.443130e+04	1.646901e+04	-6439.028762
25%	2.512003e+06	2.508618e+06	-1176.090606
50%	5.095722e+06	5.096904e+06	-17.587803
75%	7.396514e+06	7.395103e+06	1164.627837
max	1.000294e+07	1.000122e+07	6232.170018

In [21]:
```python
from sklearn.svm  import SVR
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.metrics import mean_absolute_error, mean_squared_error

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

param_grid_SVR = {
    'reg__C': [0.01, 0.1, 1, 10, 100],
    'reg__epsilon': [0.1, 0.2, 0.5, 1],
    'feature_selection__k': np.arange(1,20)
}

GSCV_SVR = GridSearchCV(pipe_SVR, param_grid_SVR, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)

GSCV_SVR.fit(x_train_enc, y_train_properti)

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

SVR_predict = GSCV_SVR.predict(x_test_enc)

mse_SVR = mean_squared_error(y_test_properti, SVR_predict)
mae_SVR = mean_absolute_error(y_test_properti, SVR_predict)

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

```
Best model:Pipeline(steps=[('scale', StandardScaler()),
                ('feature_selection',
                 SelectKBest(k=2,
                             score_func=<function f_regression at 0x000002842B5A8FE0>)),
                ('reg', SVR(C=100, kernel='linear'))])
Ridge best parameters:{'feature_selection__k': 2, 'reg__C': 100, 'reg__epsilon': 0.1}
Koefisien/bobot:[[647958.92070824  21714.67149088]]
Intercept/bias:[4985563.02448221]
SVR Mean Squared Error (MSE):4980030669523.993
SVR Mean Absolute Error (MAE):1929862.0391984528
SVR Root Mean Squared Error (RMSE):2231598.232102722
```

In [22]:
```python
df_results['SVR Prediction'] = SVR_predict
df_results = pd.DataFrame(y_test_properti)
df_results['SVR Prediction'] = SVR_predict

df_results['Selisih_Price_SVR'] = df_results['SVR Prediction'] - df_results['price']

df_results.head(10)
```

Out[22]:

	price	SVR Prediction	Selisih_Price_SVR
7619	2714670.6	4.454680e+06	1.740010e+06
5479	5714000.4	5.122242e+06	-5.917584e+05
1356	7274269.9	5.534214e+06	-1.740055e+06
7560	4822319.0	4.914256e+06	9.193703e+04
3551	8780796.5	5.843627e+06	-2.937169e+06
6215	743576.1	3.996044e+06	3.252468e+06
9718	4807525.4	4.948707e+06	1.411817e+05
2504	6647572.6	5.362333e+06	-1.285240e+06
9152	3957780.9	4.726705e+06	7.689238e+05
8110	4661805.0	4.930081e+06	2.682763e+05

In [23]:
```python
df_results.describe()
```

Out[23]:

	price	SVR Prediction	Selisih_Price_SVR
count	2.500000e+03	2.500000e+03	2.500000e+03
mean	5.006076e+06	4.989146e+06	-1.692961e+04
std	2.880289e+06	6.487739e+05	2.231980e+06
min	1.443130e+04	3.831828e+06	-3.906800e+06
25%	2.512003e+06	4.425845e+06	-1.873044e+06
50%	5.095722e+06	5.001531e+06	-8.876309e+04
75%	7.396514e+06	5.529154e+06	1.925800e+06
max	1.000294e+07	6.136293e+06	3.871938e+06

In [27]:
```python
df_results = pd.DataFrame({'properti': y_test_properti})

df_results['Ridge Prediction'] = Ridge_predict
df_results['Selisih_Price_RR'] = df_results['properti'] - df_results['Ridge Prediction']

df_results['SVR Prediction'] = SVR_predict
df_results['Selisih_Price_SVR'] = df_results['properti'] - df_results['SVR Prediction']

df_results.head(10)
```

Out[27]:

	properti	Ridge Prediction	Selisih_Price_RR	SVR Prediction	Selisih_Price_SVR
7619	2714670.6	2.716003e+06	-1332.150296	4.454680e+06	-1.740010e+06
5479	5714000.4	5.716484e+06	-2483.645076	5.122242e+06	5.917584e+05
1356	7274269.9	7.274709e+06	-438.618820	5.534214e+06	1.740055e+06
7560	4822319.0	4.824374e+06	-2055.308907	4.914256e+06	-9.193703e+04
3551	8780796.5	8.784101e+06	-3304.056988	5.843627e+06	2.937169e+06
6215	743576.1	7.447270e+05	-1150.850540	3.996044e+06	-3.252468e+06
9718	4807525.4	4.806934e+06	591.329495	4.948707e+06	-1.411817e+05
2504	6647572.6	6.646035e+06	1537.626336	5.362333e+06	1.285240e+06
9152	3957780.9	3.956017e+06	1763.436764	4.726705e+06	-7.689238e+05
8110	4661805.0	4.658478e+06	3326.718218	4.930081e+06	-2.682763e+05

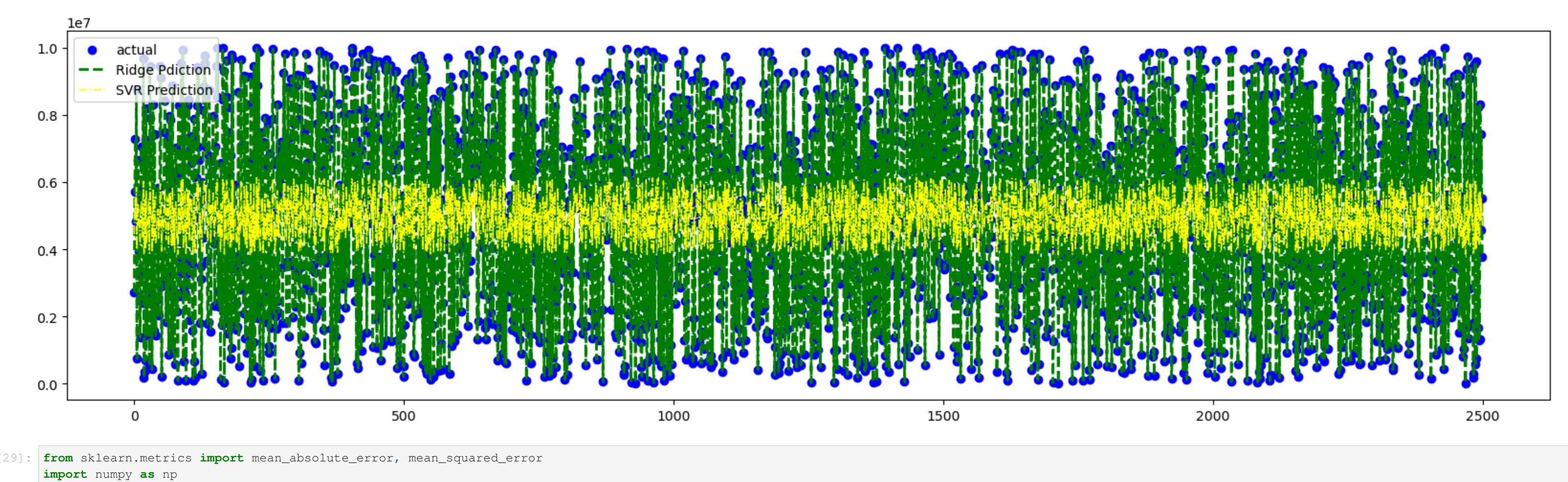
In [25]:
```python
df_results.describe()
```

Out[25]:

	properti	Ridge Prediction	Selisih_Price_RR	SVR Prediction	Selisih_Price_SVR
count	2.500000e+03	2.500000e+03	2500.000000	2.500000e+03	2.500000e+03
mean	5.006076e+06	5.006065e+06	10.629318	4.989146e+06	1.692961e+04
std	2.880289e+06	2.880284e+06	1915.529235	6.487739e+05	2.231980e+06
min	1.443130e+04	1.646901e+04	-6232.170018	3.831828e+06	-3.871938e+06
25%	2.512003e+06	2.508618e+06	-1164.627837	4.425845e+06	-1.925800e+06
50%	5.095722e+06	5.096904e+06	17.587803	5.001531e+06	8.876309e+04
75%	7.396514e+06	7.395103e+06	1176.090606	5.529154e+06	1.873044e+06
max	1.000294e+07	1.000122e+07	6439.028762	6.136293e+06	3.906800e+06

In [28]:
```python
import matplotlib.pyplot as plt

plt.figure(figsize=(20,5))
data_len = range(len(y_test_properti))
plt.scatter(data_len, df_results.properti, label="actual", color="blue")
plt.plot(data_len, df_results['Ridge Prediction'], label="Ridge Pdiction", color="green", linewidth=2, linestyle="dashed")
plt.plot(data_len, df_results['SVR Prediction'], label="SVR Prediction", color="yellow", linewidth=1, linestyle="-.")
plt.legend()
plt.show
```

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



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

mae_ridge = mean_absolute_error(df_results['properti'], df_results['Ridge Prediction'])
rmse_ridge = np.sqrt(mean_squared_error(df_results['properti'], df_results['Ridge Prediction']))
ridge_feature_count = GSCV_RR.best_params_['feature_selection__k']

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

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

```
Ridge MAE: 1483.6207754128613, Ridge RMSE:  1915.1755879661425, Ridge Feature Count: 18
SVR MAE: 1929862.0391984528, SVR RMSE: 2231598.232102722, SVR Feature Count: 2
```

In [30]:
```python
import pickle

best_model = GSCV_SVR.best_estimator_
```

```python
with open('SVR_properti_model.pkl','wb') as f:
    pickle.dump(best_model, f)

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

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

Klasifikasi : Algoritma Regressor berbasis model

- Lasso Regression
- Random Forest Regressor

In [40]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, KFold, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder
from sklearn.compose import make_column_transformer
from sklearn.impute import SimpleImputer
from sklearn.linear_model import Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest, f_regression
```

In [41]:
```python
housing = pd.read_csv(r'C:\Users\pf34h\Downloads\UTS (2)\UTS\Dataset UTS_Gasal 2425 .csv')
print(housing.head(10))
```

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

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

   attic  garage hasstorageroom  hasguestroom      price category
0   9005     956             no             7  7559081.5   Luxury
1   8852     135            yes             9  5574642.1   Middle
2   4748     654             no            10  8696869.3   Luxury
3   5792     807            yes             5  5154055.2   Middle
4   1172     716            yes             9  9652258.1   Luxury
5   7117     240             no             7  7986665.8   Luxury
6    281     384            yes             5  7607322.9   Luxury
7    129     726             no             9  6420823.1   Middle
8   9056     892            yes             1  9244344.0   Luxury
9   1221     328             no            10  9515440.4   Luxury
```

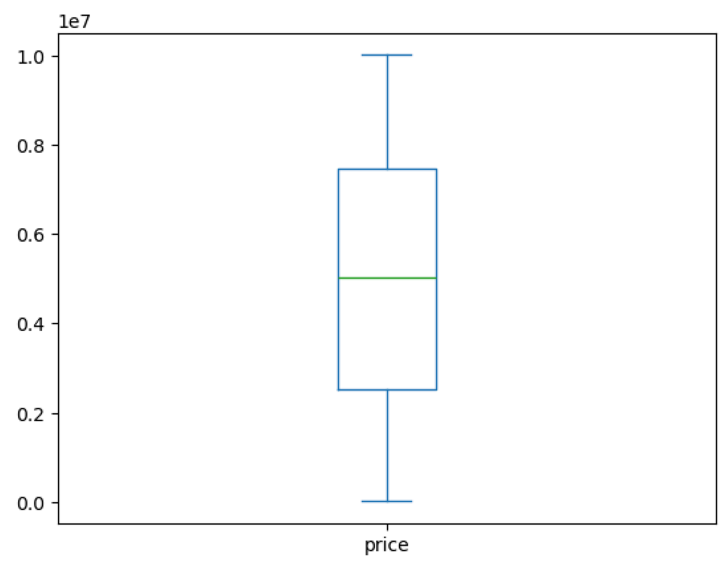
In [42]:
```python
print("Data Null:\n", housing.isnull().sum())
print("Data Kosong:", housing.empty)
```

```
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
```

In [43]:
```python
housing = housing.drop(['citycode', 'category'], axis=1)
print(housing.head(10))
```

```
   squaremeters  numberofrooms hasyard haspool  floors  citypartrange  \
0         75523              3      no     yes      63              3
1         55712             58      no     yes      19              6
2         86929            100     yes      no      11              3
3         51522              3      no      no      61              8
4         96470             74     yes      no      21              4
5         79770              3      no     yes      69             10
6         75985             60     yes      no      67              6
7         64169             88      no     yes       6              3
8         92383             12      no      no      78              3
9         95121             46      no     yes       3              7

   numprevowners  made isnewbuilt hasstormprotector  basement  attic  garage  \
0              8  2005        old               yes      4313   9005     956
1              8  2021        old                no      2937   8852     135
2              4  2003        new                no      6326   4748     654
3              3  2012        new               yes       632   5792     807
4              2  2011        new               yes      5414   1172     716
5              5  2018        old               yes      8871   7117     240
6              9  2009        new               yes      4878    281     384
7              9  2011        new               yes      3054    129     726
8              7  2000        old                no      7507   9056     892
9              9  1994        old                no       615   1221     328

  hasstorageroom  hasguestroom      price
0             no             7  7559081.5
1            yes             9  5574642.1
2             no            10  8696869.3
3            yes             5  5154055.2
4            yes             9  9652258.1
5             no             7  7986665.8
6            yes             5  7607322.9
7             no             9  6420823.1
8            yes             1  9244344.0
9             no            10  9515440.4
```

In [44]:
```python
housing['price'].plot(kind='box')
plt.show()
```



In [45]:
```python
def remove_outlier(df):
    for col in df.columns:
        if pd.api.types.is_numeric_dtype(df[col]):
            Q1 = df[col].quantile(0.25)
            Q3 = df[col].quantile(0.75)
            IQR = Q3 - Q1
            df = df[(df[col] >= Q1 - 1.5 * IQR) & (df[col] <= Q3 + 1.5 * IQR)]
    return df

housing_clean = remove_outlier(housing)
print("Jumlah baris sebelum:", housing.shape[0])
print("Jumlah baris sesudah:", housing_clean.shape[0])
```

```
Jumlah baris sebelum: 10000
Jumlah baris sesudah: 10000
```

In [46]:
```python
X_regress = housing_clean.drop('price', axis=1)
y_regress = housing_clean['price']
```

In [47]:
```python
X_train, X_test, y_train, y_test = train_test_split(X_regress, y_regress, test_size=0.2, random_state=72)
```

In [48]:
```python
cat_cols = X_train.select_dtypes(include=['object']).columns
col_transformer = make_column_transformer((OneHotEncoder(), cat_cols), remainder='passthrough')

X_train_enc = col_transformer.fit_transform(X_train)
X_test_enc = col_transformer.transform(X_test)
```

In [66]:
```python
pipe_lasso = Pipeline([
    ('scale', StandardScaler()),
    ('feature_selection', SelectKBest(score_func=f_regression)),
    ('reg', Lasso(max_iter=1000)),
])

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

GSCV_Lasso = GridSearchCV(pipe_lasso, param_grid_lasso, cv=5, scoring='neg_mean_squared_error')
GSCV_Lasso.fit(X_train_enc, y_train)

print("Best Model Lasso:", GSCV_Lasso.best_estimator_)
print("Koefisien:", GSCV_Lasso.best_estimator_.named_steps['reg'].coef_)
print("Intercept:", GSCV_Lasso.best_estimator_.named_steps['reg'].intercept_)
```

```
Best Model Lasso: Pipeline(steps=[('scale', StandardScaler()),
                ('feature_selection',
                 SelectKBest(k=19,
                             score_func=<function f_regression at 0x000001D58DEBF740>)),
                ('reg', Lasso(alpha=10))])
Koefisien: [-1.50538168e+03  0.00000000e+00 -1.48363148e+03  0.00000000e+00
  8.12763395e+01 -0.00000000e+00 -5.76955953e+01  3.27418093e-14
  0.00000000e+00 -0.00000000e+00  2.87987064e+06  0.00000000e+00
  1.57072110e+03  1.25244258e+02 -0.00000000e+00 -0.00000000e+00
 -0.00000000e+00  9.19615181e+00 -0.00000000e+00]
Intercept: 4979308.578425
```

In [74]:
```python
pipe_rf = Pipeline([
    ('scale', StandardScaler()),
    ('feature_selection', SelectKBest(score_func=f_regression)),
    ('reg', RandomForestRegressor(random_state=72))
])

param_grid_rf = {
    'scale': [MinMaxScaler(), StandardScaler()],
    'reg__n_estimators': [100, 200],
    'reg__max_depth': [10, 20, None],
    'feature_selection__k': np.arange(1, 20),
}

GSCV_RF = GridSearchCV(pipe_rf, param_grid_rf, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
GSCV_RF.fit(X_train_enc, y_train)

print("Best Model RF:", GSCV_RF.best_estimator_)
```

```
Best Model RF: Pipeline(steps=[('scale', MinMaxScaler()),
                ('feature_selection',
                 SelectKBest(k=5,
                             score_func=<function f_regression at 0x000001D58DEBF740>)),
                ('reg',
                 RandomForestRegressor(n_estimators=200, random_state=72))])
```

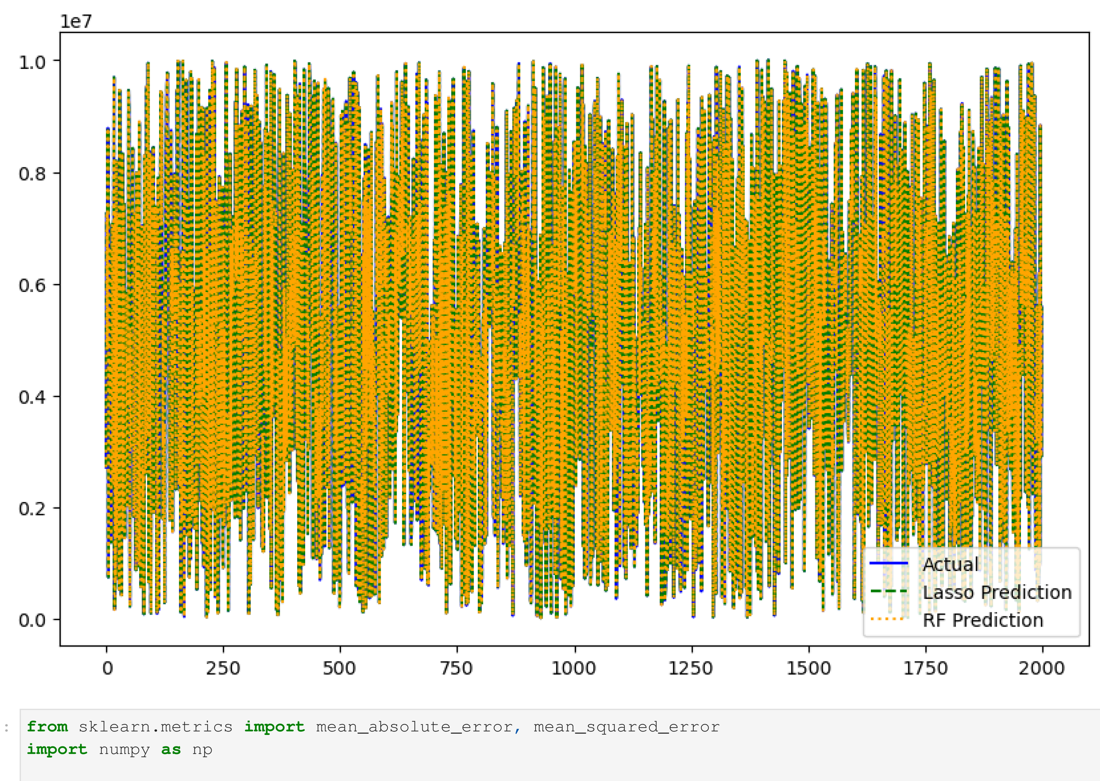
In [75]:
```python
lasso_pred = GSCV_Lasso.predict(X_test_enc)
mse_lasso = mean_squared_error(y_test, lasso_pred)
mae_lasso = mean_absolute_error(y_test, lasso_pred)

print("Lasso MSE:", mse_lasso)
print("Lasso MAE:", mae_lasso)
print("Lasso RMSE:", np.sqrt(mse_lasso))
```

```
Lasso MSE: 3639329.1304730116
Lasso MAE: 1478.7234844071418
Lasso RMSE: 1907.7025791440897
```

In [76]:
```python
rf_pred = GSCV_RF.predict(X_test_enc)
mse_rf = mean_squared_error(y_test, rf_pred)
mae_rf = mean_absolute_error(y_test, rf_pred)

print("RF MSE:", mse_rf)
print("RF MAE:", mae_rf)
print("RF RMSE:", np.sqrt(mse_rf))
```

```
RF MSE: 14905239.418717457
RF MAE: 3104.0755612500902
RF RMSE: 3860.730425543521
```

In [77]:
```python
df_results = pd.DataFrame({'Actual': y_test, 'Lasso Prediction': lasso_pred, 'RF Prediction': rf_pred})

plt.figure(figsize=(10, 6))
plt.plot(df_results['Actual'].values, label='Actual', color='blue')
plt.plot(df_results['Lasso Prediction'].values, label='Lasso Prediction', linestyle='--', color='green')
plt.plot(df_results['RF Prediction'].values, label='RF Prediction', linestyle=':', color='orange')
plt.legend()
plt.show()
```



In [80]:
```python
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

mae_lasso = mean_absolute_error(df_results['Actual'], df_results['Lasso Prediction'])
rmse_lasso = np.sqrt(mean_squared_error(df_results['Actual'], df_results['Lasso Prediction']))
lasso_feature_count = GSCV_Lasso.best_params_['feature_selection__k']

mae_rf = mean_absolute_error(df_results['Actual'], df_results['RF Prediction'])
rmse_rf = np.sqrt(mean_squared_error(df_results['Actual'], df_results['RF Prediction']))
rf_feature_count = GSCV_RF.best_params_['feature_selection__k']

print(f"Lasso MAE: {mae_lasso}, Lasso RMSE:  {rmse_lasso}, Lasso Feature Count: {lasso_feature_count}")
print(f"Random Forest MAE: {mae_rf}, Random Forest RMSE:  {rmse_rf}, Random Forest Feature Count: {rf_feature_count}")
```

```
Lasso MAE: 1478.7234844071418, Lasso RMSE:  1907.7025791440897, Lasso Feature Count: 19
Random Forest MAE: 3104.0755612500902, Random Forest RMSE:  3860.730425543521, Random Forest Feature Count: 5
```

In [81]:
```python
import pickle

best_model = GSCV_RF.best_estimator_

with open('RF_properti_model.pkl','wb') as f:
```

```
    pickle.dump(best_model, f)

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

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