Kelompok : Pandas

Anggota :

- **Kevin Stevano Rianto / 220711655**
- **Averina Harseno Subianto / 220711757**
- **Christofer Nicholas Japri / 220711803**
- **Ivona Atha Nur Afiah / 220711805**

Algoritma Regressor :

Lasso & Random Forest Regression

```
# %%

import pandas as pd

import numpy as np


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

df_harga.head(10)


# %%

df_harga2 = df_harga.drop(['category'], axis=1)

df_harga2.head()


# %%

df_harga2.info()


# %%

df_harga2.describe()


# %%

print(df_harga2['price'].value_counts())


# %%
```

```
print("data null \n", df_harga2.isnull().sum())

print("data kosong \n", df_harga2.empty)

print("data nan \n", df_harga2.isna().sum())


# %%

import matplotlib.pyplot as plt


df_harga2.price.plot(kind='box')

plt.gca().invert_yaxis()

plt.show()


# %%

from pandas.api.types import is_numeric_dtype

def remove_outlier(df_in):

    for col_name in list(df_in.columns):

        if is_numeric_dtype (df_in[col_name]):

            q1 = df_in[col_name].quantile(0.25)

            q3 = df_in[col_name].quantile(0.75)


            iqr = q3-q1

            batas_atas = q3 + (1.5 * iqr)

            batas_bawah = q1 - (1.5 * iqr)


            df_out = df_in.loc[(df_in[col_name] >= batas_bawah) &
(df_in[col_name] <= batas_atas)]

    return df_out


df_harga_clean = remove_outlier(df_harga2)

print("Jumlah baris DataFram sebelum dibuang outlier",
df_harga2.shape[0])
```

```
print("Jumlah baris DataFrame sesudah dibuang outlier",
df_harga_clean.shape[0])

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


plt.gca().invert_yaxis()

plt.show()


# %%

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

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

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


# %%

print("Sebelum pengecekan data duplikan, ", df_harga_clean.shape)

df_harga_clean=df_harga_clean.drop_duplicates(keep='last')

print("Setelah pengecekan data duplikat, ", df_harga_clean.shape)


# %%

from sklearn.preprocessing import OneHotEncoder

from sklearn.compose import make_column_transformer

import pandas as pd



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


transform = make_column_transformer(

    (OneHotEncoder(), kolom_kategori),

    remainder='passthrough'

)
```

```
df_encoded = transform.fit_transform(df_harga_clean)


ohe_categories =
transform.named_transformers_['onehotencoder'].get_feature_names_out(ko
lom_kategori)


remaining_columns =
df_harga_clean.columns.difference(kolom_kategori).tolist()


all_columns = list(ohe_categories) + remaining_columns


df_harga_clean = pd.DataFrame(df_encoded, columns=all_columns)




# %%
from sklearn.model_selection import train_test_split


x_regress = df_harga_clean.drop('price' ,axis=1)
y_regress = df_harga_clean.price


x_train_harga, x_test_harga, y_train_harga, y_test_harga =
train_test_split(x_regress, y_regress,

test_size=0.3,

random_state=57)




# %%
```

```
from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import GridSearchCV

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler

from sklearn.feature_selection import SelectKBest, f_regression

from sklearn.metrics import mean_absolute_error, mean_squared_error


pipe_RFR = Pipeline(steps=[

    ('scale', StandardScaler()),

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

    ('reg', RandomForestRegressor())

    ])


param_grid_RFR = {

    'reg__n_estimators': [200, 250, 300],

    'reg__max_depth': [6,8,10],

    'feature_selection__k': np.arange(1, 21)

}



GSCV_RFR = GridSearchCV(pipe_RFR, param_grid_RFR, cv=5,
scoring='neg_mean_squared_error', n_jobs=-1, verbose=1)


GSCV_RFR.fit(x_train_harga, y_train_harga)


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

print("Random Forest best parameter:{}".format(GSCV_RFR.best_params_))

print("Feature
Importance:{}".format(GSCV_RFR.best_estimator_.named_steps['reg'].featu
re_importances_))
```

```
RFR_predict = GSCV_RFR.predict(x_test_harga)


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


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




# %%
df_results = pd.DataFrame(y_test_harga, columns=['price'])
df_results = pd.DataFrame(y_test_harga)
df_results['RFR Prediction'] = RFR_predict


df_results['Selisih_Harga_RFR'] = df_results['RFR Prediction'] -
df_results['price']
df_results.head()


# %%
df_results.describe()


# %%
from sklearn.linear_model import Lasso
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_Lasso = Pipeline(steps=[
    ('scale', StandardScaler()),
    ('feature_selection', SelectKBest(score_func=f_regression)),
    ('reg', Lasso(max_iter=2000))
    ])


param_grid_Lasso = {
    'reg__alpha': [0.0001, 0.001, 0.01, 0.05, 0.1, 0.5, 1.0, 5.0,
10.0],
    'feature_selection__k': np.arange(1,21,2),
    'reg__tol': [1e-4, 1e-3],
    'reg__fit_intercept': [True]
}


GSCV_Lasso = GridSearchCV(pipe_Lasso, param_grid_Lasso, cv=5,
scoring='neg_mean_squared_error', verbose=1)


GSCV_Lasso.fit(x_train_harga, y_train_harga)


print("Best model:{}".format(GSCV_Lasso.best_estimator_))
print("Lasso best parameter:{}".format(GSCV_Lasso.best_params_))
print("Koefisien/bobot:{}".format(GSCV_Lasso.best_estimator_.named_step
s['reg'].coef_))
print("Intercept/bias:{}".format(GSCV_Lasso.best_estimator_.named_steps
['reg'].intercept_))
print("Best Alpha:{}".format(GSCV_Lasso.best_params_['reg__alpha']))
```

```
Lasso_predict = GSCV_Lasso.predict(x_test_harga)


mse_Lasso = mean_squared_error(y_test_harga, Lasso_predict)

mae_Lasso = mean_absolute_error(y_test_harga, 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)))




# %%

df_results['Lasso Prediction'] = Lasso_predict

df_results = pd.DataFrame(y_test_harga)

df_results['Lasso Prediction'] = Lasso_predict


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

df_results.head()


# %%

df_results.describe()


# %%

from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import GridSearchCV, StratifiedKFold

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import MinMaxScaler

from sklearn.feature_selection import SelectPercentile, f_regression

from sklearn.metrics import mean_absolute_error, mean_squared_error
```

```
y_strata = pd.qcut(y_train_harga, q=5, labels=False, duplicates='drop')


pipe_RFR2 = Pipeline(steps=[
    ('scale', MinMaxScaler()),
    ('feature_selection', SelectPercentile(score_func=f_regression)),
    ('reg', RandomForestRegressor())
])


param_grid_RFR2 = {
    'reg__n_estimators': [200, 350, 300],
    'reg__max_depth': [6, 8, 10],
    'feature_selection__percentile': np.arange(10, 100, 10)
}


GSCV_RFR2 = GridSearchCV(pipe_RFR2, param_grid_RFR2,
cv=StratifiedKFold(n_splits=5, shuffle=True,
random_state=57).split(x_train_harga, y_strata),
scoring='neg_mean_squared_error', n_jobs=-1, verbose=1)
GSCV_RFR2.fit(x_train_harga, y_train_harga)


print("Best model:{}".format(GSCV_RFR2.best_estimator_))
print("Random Forest best parameter:{}".format(GSCV_RFR2.best_params_))
print("Feature
Importance:{}".format(GSCV_RFR2.best_estimator_.named_steps['reg'].feat
ure_importances_))


RFR_predict2 = GSCV_RFR2.predict(x_test_harga)
mse_RFR2 = mean_squared_error(y_test_harga, RFR_predict2)
mae_RFR2 = mean_absolute_error(y_test_harga, RFR_predict2)
```

```
print("Random Forest Mean Squared Error (MSE): {}".format(mse_RFR2))

print("Random Forest Mean Absolute Error (MAE): {}".format(mae_RFR2))

print("Random Forest Root Mean Squared Error:
{}".format(np.sqrt(mse_RFR2)))



# %%

df_results['RFR Prediction 2'] = RFR_predict2

df_results = pd.DataFrame(y_test_harga)

df_results['RFR Prediction 2'] = RFR_predict2


df_results['Selisih_Harga_RFR_2'] = df_results['RFR Prediction 2'] -
df_results['price']

df_results.head()


# %%

df_results.describe()


# %%

from sklearn.linear_model import Lasso

from sklearn.model_selection import GridSearchCV, StratifiedKFold

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import MinMaxScaler

from sklearn.feature_selection import SelectPercentile, f_regression

from sklearn.metrics import mean_absolute_error, mean_squared_error


y_strata = pd.qcut(y_train_harga, q=5, labels=False, duplicates='drop')


pipe_Lasso2 = Pipeline(steps=[

    ('scale', MinMaxScaler()),
```

```
    ('feature_selection', SelectPercentile(score_func=f_regression)),
    ('reg', Lasso(max_iter=2000))
])


param_grid_Lasso2 = {
    'reg__alpha': [0.0001, 0.001, 0.01, 0.05, 0.1, 0.5, 1.0, 5.0,
10.0],
    'feature_selection__percentile': np.arange(10, 100, 10),
    'reg__tol': [1e-4, 1e-3],
    'reg__fit_intercept': [True]
}


GSCV_Lasso2 = GridSearchCV(pipe_Lasso2, param_grid_Lasso2,
cv=StratifiedKFold(n_splits=5, shuffle=True,
random_state=57).split(x_train_harga, y_strata), n_jobs=-1,
verbose=1,scoring='neg_mean_squared_error')
GSCV_Lasso2.fit(x_train_harga, y_train_harga)


print("Best model:{}".format(GSCV_Lasso2.best_estimator_))
print("Lasso best parameter:{}".format(GSCV_Lasso2.best_params_))
print("Koefisien/bobot:{}".format(GSCV_Lasso2.best_estimator_.named_ste
ps['reg'].coef_))
print("Intercept/bias:{}".format(GSCV_Lasso2.best_estimator_.named_step
s['reg'].intercept_))


Lasso_predict2 = GSCV_Lasso2.predict(x_test_harga)
mse_Lasso2 = mean_squared_error(y_test_harga, Lasso_predict2)
mae_Lasso2 = mean_absolute_error(y_test_harga, Lasso_predict2)
```

```
print("Lasso Mean Squared Error 2 (MSE2): {}".format(mse_Lasso2))

print("Lasso Mean Absolute Error 2 (MAE2): {}".format(mae_Lasso2))

print("Lasso Root Mean Squared Error 2:
{}".format(np.sqrt(mse_Lasso2)))


# %%

df_results['Lasso Prediction 2'] = Lasso_predict2

df_results = pd.DataFrame(y_test_harga)

df_results['Lasso Prediction 2'] = Lasso_predict2


df_results['Selisih_Harga_LR2'] = df_results['Lasso Prediction 2'] -
df_results['price']

df_results.head()


# %%

df_results.describe()


# %%


df_results['RFR Prediction'] = RFR_predict


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


df_results['Lasso Prediction'] = Lasso_predict


df_results['Selisih_Harga_LR'] = df_results['price'] -
df_results['Lasso Prediction']
```

```
df_results['RFR Prediction 2'] = RFR_predict2


df_results['Selisih_Harga_RFR_2'] = df_results['price'] -
df_results['RFR Prediction 2']


df_results['Lasso Prediction 2'] = Lasso_predict2


df_results['Selisih_Harga_LR2'] = df_results['price'] -
df_results['Lasso Prediction 2']


df_results.head()




# %%
df_results.describe()


# %%
import matplotlib.pyplot as plt


plt.figure(figsize=(20,20))
data_len = range(len(y_test_harga))
plt.scatter(data_len, df_results.price, label="actual", color="blue")
plt.plot(data_len, df_results['RFR Prediction'], label="RFR
Prediction", color="yellow", linewidth=4, linestyle="dashed")
plt.plot(data_len, df_results['Lasso Prediction'], label="Lasso
Prediction", color="green", linewidth=3, linestyle="-.")
plt.plot(data_len, df_results['RFR Prediction 2'], label="RFR
Prediction 2", color="black", linewidth=2, linestyle="--")
plt.plot(data_len, df_results['Lasso Prediction 2'], label="Lasso
Prediction 2", color="red", linewidth=1, linestyle=":")
plt.legend()
```

```
plt.show


# %%

from sklearn.metrics import  mean_absolute_error, mean_squared_error

import numpy as np



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 Prediction']))

rfr_feature_count = GSCV_RFR.best_params_['feature_selection__k']


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

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

lasso_feature_count = GSCV_Lasso.best_params_['feature_selection__k']


mae_lasso2 = mean_absolute_error(df_results['price'], df_results['Lasso
Prediction 2'])

rmse_lasso2 = np.sqrt(mean_squared_error(df_results['price'],
df_results['Lasso Prediction 2']))

lasso2_feature_count =
GSCV_Lasso2.best_params_['feature_selection__percentile']


mae_rfr2 = mean_absolute_error(df_results['price'], df_results['RFR
Prediction 2'])

rmse_rfr2 = np.sqrt(mean_squared_error(df_results['price'],
df_results['RFR Prediction 2']))

rfr2_feature_count =
GSCV_RFR2.best_params_['feature_selection__percentile']
```

```
print(f"RFR MAE: {mae_rfr}, RFR RMSE: {rmse_rfr}, RFR Feature Count:
{rfr_feature_count}")

print(f"LASSO MAE: {mae_lasso}, LASSO RMSE: {rmse_lasso}, LASSO Feature
Count: {lasso_feature_count}")

print(f"RFR2 MAE: {mae_rfr2}, RFR2 RMSE: {rmse_rfr2}, RFR2 Feature
Count: {rfr2_feature_count}")

print(f"LASSO 2 MAE: {mae_lasso2}, LASSO2 RMSE: {rmse_lasso2}, LASSO2
Feature Count: {lasso2_feature_count}")


# %%

import pickle


best_model = GSCV_Lasso2.best_estimator_


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

    pickle.dump(best_model, f)


    print("Model Berhasil disimpan ke Lasso_properti_model.pkl")
```

Catatan : #%% pemisah antar bagian

SVR & Ridge

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

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

```python
df_harga2 = df_harga.drop(['category'], axis=1)
df_harga2.head()
```

```python
df_harga2.info()
```

```
df_harga2.describe()
```

```
print(df_harga2['price'].value_counts())
```

```
print("data null \n", df_harga2.isnull().sum())
print("data kosong \n", df_harga2.empty)
print("data nan \n", df_harga2.isna().sum())
```

```python
import matplotlib.pyplot as plt

df_harga2.price.plot(kind='box')
plt.gca().invert_yaxis()
plt.show()
```

```python
from pandas.api.types import is_numeric_dtype
def remove_outlier(df_in):
    for col_name in list(df_in.columns):
        if is_numeric_dtype (df_in[col_name]):
            q1 = df_in[col_name].quantile(0.25)
            q3 = df_in[col_name].quantile(0.75)

            iqr = q3-q1
            batas_atas = q3 + (1.5 * iqr)
            batas_bawah = q1 - (1.5 * iqr)

            df_out = df_in.loc[(df_in[col_name] >= batas_bawah) & (df_in[col_name] <= batas_atas)]
    return df_out

df_harga_clean = remove_outlier(df_harga2)
print("Jumlah baris DataFrame sebelum dibuang outlier", df_harga2.shape[0])
print("Jumlah baris Dataframe sesudah dibuang outlier", df_harga_clean.shape[0])
df_harga_clean.price.plot(kind='box', vert=True)

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

```

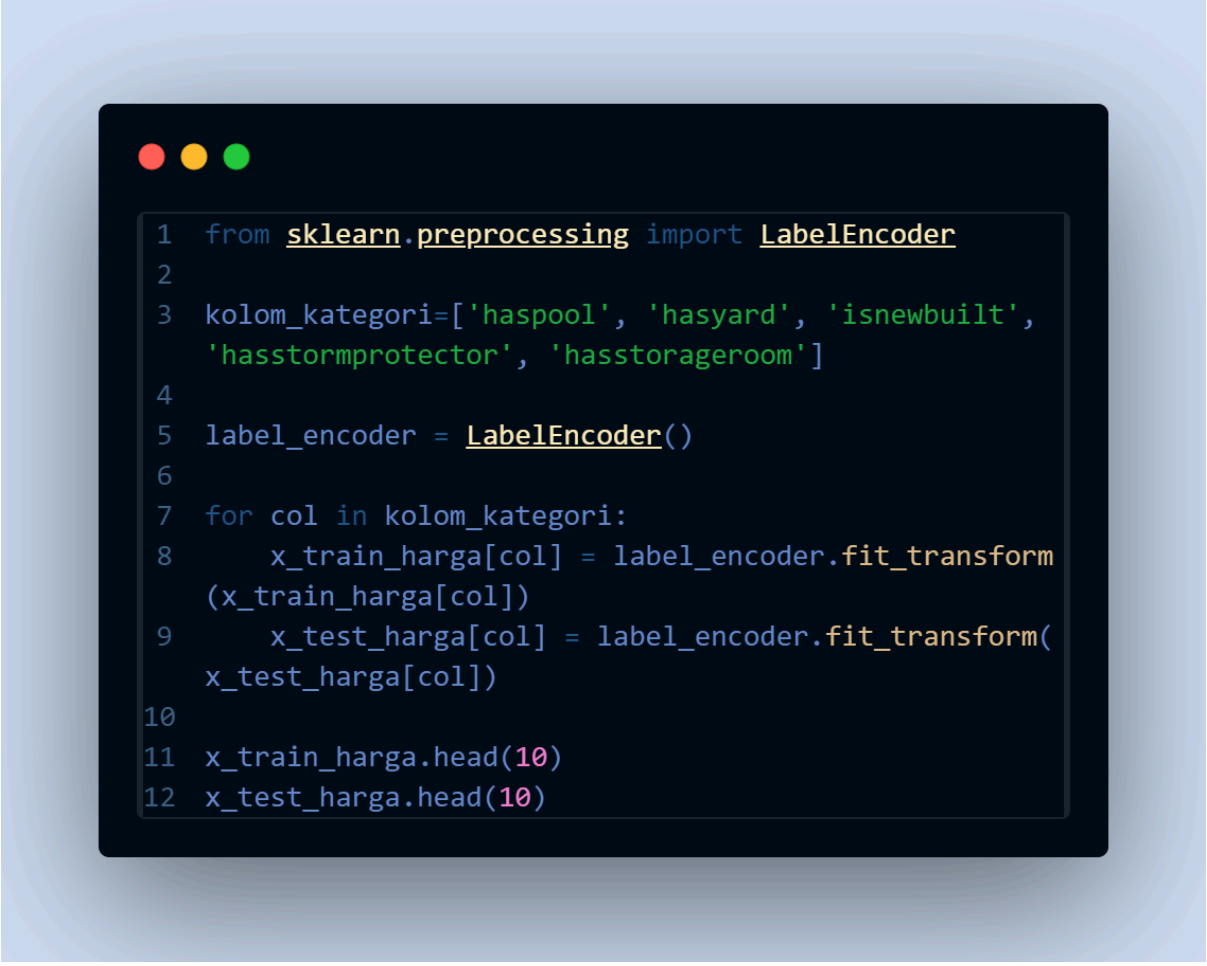
```
print("data null \n", df_harga_clean.isnull().sum())
print("data kosong \n", df_harga_clean.empty)
print("data nan \n", df_harga_clean.isna().sum())
```

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

```
from sklearn.model_selection import train_test_split

x_regress = df_harga_clean.drop('price', axis=1)
y_regress = df_harga_clean.price

x_train_harga, x_test_harga, y_train_harga,
y_test_harga = train_test_split(x_regress, y_regress,
test_size=0.3, random_state=57)
```



```python
from sklearn.preprocessing import LabelEncoder

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

label_encoder = LabelEncoder()

for col in kolom_kategori:
    x_train_harga[col] = label_encoder.fit_transform(x_train_harga[col])
    x_test_harga[col] = label_encoder.fit_transform(x_test_harga[col])

x_train_harga.head(10)
x_test_harga.head(10)
```

```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.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' : [0.001, 0.01, 0.1, 1, 10],
    'feature_selection__k': np.arange(1, 17)
}

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

GSCV_RR.fit(x_train_harga, y_train_harga)

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_harga)

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

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

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

df_results.head()
```

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

```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': [0.01, 0.05, 0.1, 0.2, 0.5],
    'feature_selection__k': np.arange(1,17)
}

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

GSCV_SVR.fit(x_train_harga, y_train_harga)

print("Best model: {}".format(GSCV_SVR.best_estimator_))
print("SVR 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_harga)

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

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

df_results['Selisih_Harga_SVR'] = df_results[
'SVR Prediction'] - df_results['price']
df_results.head
```

```
df_results.describe()
```

```python
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import SelectPercentile, f_regression
from sklearn.metrics import mean_absolute_error, mean_squared_error

y_strata = pd.qcut(y_train_harga, q=5, labels=False, duplicates='drop')
pipe_SVR2 = Pipeline(steps=[
    ('scale', MinMaxScaler()),
    ('feature_selection', SelectPercentile(score_func=f_regression)),
    ('reg', SVR(kernel='linear'))
])

param_grid_SVR2 = {
    'feature_selection__percentile': [10, 20, 30, 50, 70, 90],
    'reg__C': [0.1, 1, 10, 100],
    'reg__epsilon': [0.1, 0.2, 0.3],
}

GSCV_SVR2 = GridSearchCV(pipe_SVR2, param_grid_SVR2,
                         cv=StratifiedKFold(n_splits=5, shuffle=True,
random_state=57).split(x_train_harga, y_strata),
                         scoring='neg_mean_squared_error', n_jobs=-1)

GSCV_SVR2.fit(x_train_harga, y_train_harga)

print("Best model: {}".format(GSCV_SVR2.best_estimator_))
print("SVR best parameters: {}".format(GSCV_SVR2.best_params_))

print("Koefisien/bobot: {}".format(GSCV_SVR2.best_estimator_.named_steps['reg'
].coef_))
print("Intercept/bias: {}".format(GSCV_SVR2.best_estimator_.named_steps['reg'
].intercept_))

SVR_predict2 = GSCV_SVR2.predict(x_test_harga)

mse_SVR2 = mean_squared_error(y_test_harga, SVR_predict2)
mae_SVR2 = mean_absolute_error(y_test_harga, SVR_predict2)

print("SVR Mean Squared Error (MSE): {}".format(mse_SVR2))
print("SVR Mean Absolute Error (MAE): {}".format(mae_SVR2))
print("SVR Root Mean Squared Error: {}".format(np.sqrt(mse_SVR2)))
```

```
df_results['SVR Prediction'] = SVR_predict2
df_results = pd.DataFrame(y_test_harga)
df_results['SVR Prediction'] = SVR_predict2

df_results['Selisih_Harga_SVR'] = df_results['SVR Prediction'] - df_results['price'
]
df_results.head
```

```
df_results.describe()
```

```python
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import SelectPercentile, f_regression
from sklearn.metrics import mean_absolute_error, mean_squared_error

y_strata = pd.qcut(y_train_harga, q=5, labels=False, duplicates='drop')

pipe_Ridge2 = Pipeline(steps=[
    ('scale', MinMaxScaler()),
    ('feature_selection', SelectPercentile(score_func=f_regression)),
    ('reg', Ridge())
])

param_grid_Ridge2 = {
    'feature_selection__percentile': [10, 20, 30, 50, 70, 90],
    'reg__alpha': [0.001, 0.01, 0.1, 1.0, 10.0, 100.0],
    'reg__solver': ['auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg']
}

GSCV_RR2 = GridSearchCV(pipe_Ridge2, param_grid_Ridge2, cv=StratifiedKFold(
n_splits=5, shuffle=True, random_state=57).split(x_train_harga, y_strata),
                        scoring='neg_mean_squared_error', error_score='raise')

GSCV_RR2.fit(x_train_harga, y_train_harga)

print("Best model: {}".format(GSCV_RR2.best_estimator_))
print("Ridge best parameters: {}".format(GSCV_RR2.best_params_))

print("Koefisien/bobot:{}".format(GSCV_RR2.best_estimator_.named_steps['reg'
].coef_))
print("Intercept/bias:{}".format(GSCV_RR2.best_estimator_.named_steps['reg'
].intercept_))

Ridge_predict2 = GSCV_RR2.predict(x_test_harga)

mse_Ridge2 = mean_squared_error(y_test_harga, Ridge_predict)
mae_Ridge2 = mean_absolute_error(y_test_harga, Ridge_predict)

print("Ridge Mean Squared Error (MSE): {}".format(mse_Ridge2))
print("Ridge Mean Absolute Error (MAE): {}".format(mae_Ridge2))
print("Ridge Root Mean Squared Error: {}".format(np.sqrt(mse_Ridge2)))
```

```
df_results = pd.DataFrame(y_test_harga, columns=['price'])
df_results = pd.DataFrame(y_test_harga)
df_results['Ridge Prediction 2'] = Ridge_predict2

df_results['Selisih_Harga_RR2'] = df_results['Ridge Prediction 2'] - df_results[
'price']

df_results.head()
```

```
df_results.describe()
```

```
df_results['SVR Prediction'] = SVR_predict

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

df_results['SVR Prediction 2'] = SVR_predict2

df_results['Selisih_harga_SVR2'] = df_results['price'] - df_results[
'SVR Prediction 2']

df_results['Ridge Prediction'] = Ridge_predict

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

df_results['Ridge Prediction 2'] = Ridge_predict2

df_results['Selisih_harga_Ridge 2'] = df_results['price'] - df_results[
'Ridge Prediction 2']

df_results.head()
```

```
df_results.describe()
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(20,5))
data_len = range(len(y_test_harga))
plt.scatter(data_len, df_results.price, label="actual", color="blue")
plt.plot(data_len, df_results['Ridge Prediction'], label="Ridge Prediction", color="yellow", linewidth=4, linestyle="dashed")
plt.plot(data_len, df_results['Ridge Prediction 2'], label="Ridge Prediction 2", color="green", linewidth=3, linestyle="-.")
plt.plot(data_len, df_results['SVR Prediction'], label="Ridge Prediction", color="black", linewidth=2, linestyle="--")
plt.plot(data_len, df_results['SVR Prediction 2'], label="SVR Prediction 2", color="red", linewidth=1, linestyle=":")
plt.legend()
plt.show
```

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

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

mae_ridge2 = mean_absolute_error(df_results['price'], df_results['Ridge Prediction 2'])
rmse_ridge2 = np.sqrt(mean_squared_error(df_results['price'], df_results['Ridge Prediction 2']))
ridge_feature_count2 = GSCV_RR2.best_params_['feature_selection__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 Prediction']))
svr_feature_count = GSCV_SVR.best_params_['feature_selection__k']

mae_svr2= mean_absolute_error(df_results['price'], df_results['SVR Prediction 2'])
rmse_svr2 = np.sqrt(mean_squared_error(df_results['price'], df_results['SVR Prediction 2']))
svr_feature_count2 = GSCV_SVR2.best_params_['feature_selection__percentile']


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}")

print(f"Ridge MAE 2: {mae_ridge2}, Ridge RMSE 2: {rmse_ridge2}, Ridge Feature Count 2: {ridge_feature_count2}")
print(f"SVR MAE 2: {mae_svr2}, SVR RMSE 2: {rmse_svr2}, SVR Feature Count 2: {svr_feature_count2}")
```

```python
import pickle

best_model = GSCV_RR.best_estimator_

with open('MAE2_Harga_model.pkl', 'wb') as f:
    pickle.dump(best_model, f)

print("Model terbaik berhasil disimpan ke 'MAE2_Harga_model.pkl'")
```

Algoritma Klasifikasi:

Random Forest dan Logistic Regression

```
import pandas as pd
import numpy as np


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

```
df_properti2=df_properti.drop('price',axis=1)
df_properti2.head(10)
```

```
df_properti2.info()
```

```
df_properti2.describe()
```

```
df_properti2['category'].value_counts()
```

```
print("data null \n",df_properti2.isnull().sum())
print("\ndata kosong \n",df_properti2.empty)
print("\ndata nan \n",df_properti2.isna().sum())
```

```
from pandas.api.types import is_numeric_dtype
def remove_outlier(df_in):
    for col_name in list(df_in.columns):
        if is_numeric_dtype(df_in[col_name]):
            q1 = df_in[col_name].quantile(0.25)
            q3 = df_in[col_name].quantile(0.75)

            iqr = q3 - q1
            batas_atas = q3 + (1.5 * iqr)
            batas_bawah = q1 - (1.5 * iqr)

            df_out = df_in.loc[(df_in[col_name] >= batas_bawah) &
(df_in[col_name] <= batas_atas)]
    return df_out

df_properti_clean = remove_outlier(df_properti2)
print("jumlah baris dataframe sebelum dibuang outlier",
df_properti2.shape[0])
print("jumlah baris dataframe sesudah dibuang outlier",
```

```
df_properti_clean.shape[0])
```

```
print("sebelum drop missing value",df_properti2.shape)
df_properti2 = df_properti2.dropna(how='any',inplace=False)
print("setelah drop missing value",df_properti2.shape)
```

```
print("sebelum pengecekan data duplikat, ", df_properti2.shape)
df_properti3=df_properti2.drop_duplicates(keep='last')
print("setelah pengecekan data duplikat, ", df_properti3.shape)
```

```
from sklearn.model_selection import train_test_split

x = df_properti3.drop(columns=['category'],axis=1)
y = df_properti3['category']

x_train, x_test, y_train, y_test =
train_test_split(x,y,test_size=0.3,random_state=57)

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

```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()

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

for col in kolom_kategori:
    x_train[col] = label_encoder.fit_transform(x_train[col])
    x_test[col] = label_encoder.fit_transform(x_test[col])

df_train_enc = pd.DataFrame(x_train)
df_test_enc = pd.DataFrame(x_test)

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

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

df_y_train_enc = pd.DataFrame(y_train_enc)
```

```
df_y_test_enc = pd.DataFrame(y_test_enc)

df_y_train_enc.head(10)
df_y_test_enc.head(10)
```

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

pipe_RF = [
    ('data scaling', StandardScaler()),
    ('feature select',SelectKBest()),
    ('clf',RandomForestClassifier(random_state=57,
class_weight='balanced'))
]

params_grid_RF = [
    {
        'data scaling':[StandardScaler()],
        'feature select__k': np.arange(2,6),
        'clf__max_depth': np.arange(2,4),
        'clf__n_estimators':[200,300]
    },
    {
        'data scaling':[StandardScaler()],
        'feature select':[SelectPercentile()],
        'feature select__percentile': np.arange(20,50),
        'clf__max_depth': np.arange(2,4),
        'clf__n_estimators':[200,300]
    },
    {
        'data scaling':[MinMaxScaler()],
        'feature select__k': np.arange(2,6),
        'clf__max_depth': np.arange(2,4),
        'clf__n_estimators':[200,300]
    },
    {
        'data scaling':[MinMaxScaler()],
        'feature select':[SelectPercentile()],
```

```
        'feature select__percentile': np.arange(20,50),
        'clf__max_depth': np.arange(2,4),
        'clf__n_estimators':[200,300]
    }
]

estimator_RF = Pipeline(pipe_RF)

SKF = StratifiedKFold(n_splits=5,shuffle=True,random_state=57)

GSCV_RF = GridSearchCV(estimator_RF,params_grid_RF,cv=SKF,n_jobs=-1)
GSCV_RF.fit(x_train,y_train_enc)
print("GSCV finished")
```

```
print("CV Score: {}".format(GSCV_RF.best_score_))
print("Test Score: {}".format(GSCV_RF.best_estimator_.score(x_test,
y_test_enc)))
print("Best Model: ",GSCV_RF.best_estimator_)

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

RF_pred = GSCV_RF.predict(x_test)

import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,
classification_report
cm = confusion_matrix(y_test_enc, 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_enc, RF_pred))
```

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import SelectKBest, SelectPercentile
from sklearn.pipeline import Pipeline
```

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

pipe_LR = [
    ('data scaling', StandardScaler()),
    ('feature select', SelectKBest()),
    ('clf', LogisticRegression(random_state=57,
class_weight='balanced', max_iter=1000))
]

params_grid_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']
    }
]

estimator_LR = Pipeline(pipe_LR)

SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=57)
```

```
GSCV_LR = GridSearchCV(estimator_LR, params_grid_LR, cv=SKF)
GSCV_LR.fit(x_train, y_train_enc)
print("GSCV finished")
```

```
print("CV Score: {}".format(GSCV_LR.best_score_))
print("Test Score: {}".format(GSCV_LR.best_estimator_.score(x_test,
y_test_enc)))
print("Best Model: ",GSCV_LR.best_estimator_)

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

LR_pred = GSCV_LR.predict(x_test)

import matplotlib.pyplot as plt

cm = confusion_matrix(y_test_enc, LR_pred, labels=GSCV_LR.classes_)

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

plt.title("LR Confusion Matrix")
plt.show()
print("Classification report LR: \n",
classification_report(y_test_enc, LR_pred))
```

```
import pickle

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

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

Gradient Boosting Classifier dan Support Vector Machine

```
import pandas as pd
import numpy as np

df_properti=pd.read_csv(r"C:\Users\H P\OneDrive\Documents\mldl\Projek UTS Gasal 20242025-20241016\Dataset UTS_Gasal 2425.csv")
df_properti.head(20)
```

✓ 0.0s

```
df_properti2=df_properti.drop('price',axis=1)
df_properti2.head(20)
```
0.0s Python

```
df_properti2['category'].value_counts()
```
0.0s

```
df_properti2.info()
```
0.0s

```
df_properti2.describe()
```
0.0s

```
print("data null \n", df_properti2.isnull().sum())
print("\ndata kosong \n", df_properti2.empty)
print("\ndata nan \n", df_properti2.isna().sum())
```
0.0s

```
print("Sebelum pengecekan data duplikat, ", df_properti2.shape)
df_properti3=df_properti2.drop_duplicates(keep='last')
print("Setelah pengecekan data duplikat, ", df_properti3.shape)
```
0.0s

```
from sklearn.model_selection import train_test_split
x = df_properti3.drop(columns=['category'],axis=1)
y = df_properti3['category']

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3,random_state=57)

print(x_train.shape)
print(x_test.shape)
```
0.0s

```
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'
)
```
0.0s

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

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

df_train_enc.head(10)
df_test_enc.head(10)
```
0.0s

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

params_grid_svm = [
    {
    'scale': [MinMaxScaler()],
    'feat_select__k':np.arange(2,6),
    'clf__kernel':['poly','rbf'],
    'clf__C':[0.1,1],
    'clf__gamma':[0.1, 1]

    },
```

```python
    {
        '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__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=57)
GSCV_SVM = GridSearchCV(pipe_svm, params_grid_svm, cv=SKF, n_jobs=-1)
GSCV_SVM.fit(x_train_enc, y_train)
print("GSCV_SVM training finished")
```

✓ 3m 14.3s

```
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_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))
```

0.1s

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

pipe_GBT = Pipeline(steps=[
    ('feat_select', SelectKBest()),
    ('clf', GradientBoostingClassifier(random_state=57))])

params_grid_GBT = [
    {
        'feat_select__k': np.arange(2,6),
    'clf__max_depth':[*np.arange(4,5)],
    'clf__n_estimators':[100,150],
    'clf__learning_rate': [0.01,0.1,1]
    },
    {
        'feat_select':[SelectPercentile()],
        'feat_select__percentile': np.arange(20,50),
        'clf__max_depth':[*np.arange(4,5)],
        'clf__n_estimators':[100,150],
        'clf__learning_rate': [0.01,0.1,1]
    },
    {
        'feat_select__k': np.arange(2,6),
    'clf__max_depth':[*np.arange(4,5)],
    'clf__n_estimators':[100,150],
    'clf__learning_rate': [0.01,0.1,1]
    },
```

```
    {
        'feat_select__k': np.arange(2,6),
    'clf__max_depth':[*np.arange(4,5)],
    'clf__n_estimators':[100,150],
    'clf__learning_rate': [0.01,0.1,1]
    },
    {
        'feat_select':[SelectPercentile()],
        'feat_select__percentile': np.arange(20,50),
        'clf__max_depth':[*np.arange(4,5)],
        '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)
print("GSCV_GBT training finished")
```

✓ 6m 12.4s

```
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_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))
```

✓ 0.1s

```
import pickle
with open('Model_CLF_SVM_Pandas.pkl','wb') as r:
    pickle.dump((GSCV_SVM),r)

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

✓ 0.0s

File Streamlit

```
import streamlit as st

from streamlit_option_menu import option_menu

import pickle
```

```
import os


model_path = r'D:\atma\Semester 5\ML\UTS 1.1'

model = os.path.join(model_path,'BestModel_CLF_LR_Pandas.pkl')


model2 = 'Lasso_properti_model.pkl'


with open(model, 'rb') as f:

    loaded_model = pickle.load(f)


lr_model = loaded_model


with open(model2, 'rb') as f:

    loaded_model2 = pickle.load(f)


lasso_model = loaded_model2


#sidebar

with st.sidebar:

    selected = option_menu('UTS ML
24/25',['Klasifikasi','Regresi'],default_index=0)


#halaman klasifikasi

if selected == 'Klasifikasi':

    st.title('Klasifikasi')


    file = st.file_uploader("Masukan File", type=["csv","txt"])


    squaremeters = st.number_input("Masukan Luas Square Meter",0)
```

```
numberofrooms = st.slider("Jumlah Ruangan",0,100)

hasyard = st.radio("Punya Yard",["Yes","No"])

haspool = st.radio("Punya Kolam Renang",["Yes","No"])

floors = st.slider("Jumlah Lantai",0,100)

citycode = st.number_input("City Code",0)

citypartrange = st.slider("City Part Range",0,100)

numprevowners = st.slider("Jumlah Pemilik Sebelumnya",0,10)

made = st.number_input("Dibuat Tahun",0)

isnewbuilt = st.radio("Baru dibangun",["Yes","No"])

hasstormprotector = st.radio("Punya Storm Protector",["Yes","No"])

basement = st.number_input("Luas Basement",0)

attic = st.number_input("Luas Attic",0)

garage = st.number_input("Luas Garage",0)

hasstorageroom = st.radio("Punya Storage Room",["Yes","No"])
```

```
    hasguestroom = st.slider("Jumlah Guest Room",0,10)


#halaman regresi
if selected == 'Regresi':
    st.title('Regresi')


    file = st.file_uploader("Masukan File", type=["csv","txt"])


    squaremeters = st.number_input("Masukan Luas Square Meter",0)


    numberofrooms = st.slider("Jumlah Ruangan",0,100)


    hasyard = st.radio("Punya Yard",["Yes","No"])


    haspool = st.radio("Punya Kolam Renang",["Yes","No"])


    floors = st.slider("Jumlah Lantai",0,100)


    citycode = st.number_input("City Code",0)


    citypartrange = st.slider("City Part Range",0,100)


    numprevowners = st.slider("Jumlah Pemilik Sebelumnya",0,10)


    made = st.number_input("Dibuat Tahun",0)


    isnewbuilt = st.radio("Baru dibangun",["Yes","No"])


    hasstormprotector = st.radio("Punya Storm Protector",["Yes","No"])
```

```
    basement = st.number_input("Luas Basement",0)

    attic = st.number_input("Luas Attic",0)

    garage = st.number_input("Luas Garage",0)

    hasstorageroom = st.radio("Punya Storage Room",["Yes","No"])

    hasguestroom = st.slider("Jumlah Guest Room",0,10)



if hasyard == "Y":
    hasyard = 1
else:
    hasyard = 0

if haspool == "Y":
    haspool = 1
else:
    haspool = 0

if isnewbuilt == "Y":
    isnewbuilt = 1
else:
    isnewbuilt = 0

if hasstormprotector == "Y":
    hasstormprotector = 1
```

```
else:
    hasstormprotector = 0

if hasstorageroom == "Y":
    hasstorageroom = 1
else:
    hasstorageroom = 0

input_data = [[squaremeters, numberofrooms, hasyard, haspool,
               floors, citycode, citypartrange, numprevowners,
               made, isnewbuilt, hasstormprotector,  basement,
               attic, garage, hasstorageroom, hasguestroom]]

st.write("Data properti yang akan diinputkan ke model")
st.write(input_data)

if selected == 'Klasifikasi':
    if st.button("Klasifikasi Category"):
        lr_model_prediction = lr_model.predict(input_data)
        outcome = {0:'Basic', 1:'Luxury', 2:'Middle'}
        st.write(f"Properti tersebut kategori
**{outcome[lr_model_prediction[0]]}**")

if selected == 'Regresi':
    if st.button("Prediksi Price"):
        lasso_model_prediction = lasso_model.predict(input_data)
        st.markdown(f"Prediksi Harga properti adalah : $
{lasso_model_prediction[0]:.2f}")
```