SQLAlchemy

Nama-nama anggota kelompok :

1. Antonius Rosarianto Wisnu Putro	// 200710813
2. Ignatius Joti Argapoda Panggabean	// 200710862
3. Sebastian Willys Lambang	// 200710639
4. Jhonatan Emanuel Wangge	// 210711294
5. Yoga Jesay Tarigan	// 180709976

Notebook_KLASIFIKASI_B_SQLAlchemy_GradientBoostingClassifier_VS_SupportVectorMachine_Anton

```
import pandas as pd

import numpy as np


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


df_SQLAlchemy.head(10)


df_SQLAlchemy2=df_SQLAlchemy.drop('price' ,axis=1)

df_SQLAlchemy2.head(10)


df_SQLAlchemy2.info()


df_SQLAlchemy2.describe()


print("data null \n",df_SQLAlchemy2.isnull().sum())

print("\ndata kosong \n",df_SQLAlchemy2.empty)

print("\ndata nan \n",df_SQLAlchemy2.isna().sum())


print("Sebelum pengecekan data duplikat, ",df_SQLAlchemy2.shape)

df_SQLAlchemy3=df_SQLAlchemy2.drop_duplicates(keep='last')

print("Setelah pengecekan data duplikat, ",df_SQLAlchemy3.shape)


from sklearn.model_selection import train_test_split

x = df_SQLAlchemy3.drop(columns=['category'],axis=1)
```

```
y = df_SQLAlchemy3['category']

x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.25, random_state=94)

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

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

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

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

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

df_train_enc=pd.DataFrame(x_train_enc,columns=transform.get_feature_
names_out())
df_test_enc=pd.DataFrame(x_test_enc,columns=transform.get_feature_na
mes_out())

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

from sklearn.feature_selection import SelectPercentile, SelectKBest
from sklearn.preprocessing import MinMaxScaler, StandardScaler
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]
    },
    {
        '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=4)

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

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


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))



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=94))])


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': [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))
GSCV_GBT.fit(x_train_enc, y_train)
print("GSCV training finished")



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])

RF_pred = GSCV_GBT.predict(x_test_enc)

import matplotlib.pyplot as plt
cm = confusion_matrix(y_test, RF_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,
RF_pred))
```

Notebook_KLASIFIKASI_B_SQLAlchemy_RandomForest_VS_LogisticRegression_Sebastian

```
import pandas as pd
import numpy as np

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

df_SQLAlchemy2=df_SQLAlchemy.drop(['price'], axis=1)
df_SQLAlchemy2.head(10)

df_SQLAlchemy2.info()

df_SQLAlchemy2.describe()

print("data null \n",df_SQLAlchemy2.isnull().sum())
print("\ndata kosong \n",df_SQLAlchemy2.empty)
print("\ndata nan \n",df_SQLAlchemy2.isna().sum())

print("Sebelum pengecekan data duplikat, ",df_SQLAlchemy2.shape)
df_SQLAlchemy3=df_SQLAlchemy2.drop_duplicates(keep='last')
print("Setelah pengecekan data duplikat, ",df_SQLAlchemy3.shape)

from sklearn.model_selection import train_test_split
x = df_SQLAlchemy3.drop(columns=['category'],axis=1)
y = df_SQLAlchemy3['category']
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.25, random_state=94)


print(x_train.shape)

print(x_test.shape)


from sklearn.preprocessing import OneHotEncoder

from sklearn.compose import make_column_transformer


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


transform = make_column_transformer(

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

)


x_train_enc=transform.fit_transform(x_train)

x_test_enc=transform.fit_transform(x_test)


df_train_enc=pd.DataFrame(x_train_enc,columns=transform.get_feature_
names_out())

df_test_enc=pd.DataFrame(x_test_enc,columns=transform.get_feature_na
mes_out())


df_train_enc.head(10)

df_test_enc.head(10)


from sklearn.preprocessing import MinMaxScaler, StandardScaler

from sklearn.feature_selection import SelectPercentile, SelectKBest

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import GridSearchCV, StratifiedKFold

from sklearn.pipeline import Pipeline

from sklearn.metrics import classification_report, confusion_matrix,
ConfusionMatrixDisplay
```

```
import numpy as np


pipe_logreg = Pipeline(steps=[
    ('scale', MinMaxScaler()),
    ('feat_select', SelectKBest()),
    ('clf', LogisticRegression(class_weight='balanced',
max_iter=1000))
])


params_grid_logreg = [
    {
    'scale': [MinMaxScaler()],
    'feat_select__k':np.arange(2,6),
    'clf__penalty': ['l2'],
    'clf__C':[0.1, 1, 10],
    'clf__solver': ['lbfgs', 'saga']
    },
    {
    'scale': [MinMaxScaler()],
    'feat_select': [SelectPercentile()],
    'feat_select__percentile':np.arange(20,50),
    'clf__penalty': ['l2'],
    'clf__C':[0.1, 1, 10],
    'clf__solver': ['lbfgs', 'saga']
    },
    {
    'scale': [StandardScaler()],
    'feat_select__k':np.arange(2,6),
    'clf__penalty': ['l2'],
    'clf__C':[0.1, 1, 10],
    'clf__solver': ['lbfgs', 'saga']
    },
    {
```

```
    'scale': [StandardScaler()],
    'feat_select':[SelectPercentile()],
    'feat_select__percentile': np.arange(20,50),
    'clf__penalty': ['l2'],
    'clf__C':[0.1, 1, 10],
    'clf__solver': ['lbfgs', 'saga']
    }
]

SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=4)

GSCV_LogReg = GridSearchCV(pipe_logreg, params_grid_logreg, cv=SKF)
GSCV_LogReg.fit(x_train_enc, y_train)
print("GSCV training finished")

print("CV Score : {}".format(GSCV_LogReg.best_score_))
print("Test Score:
{}".format(GSCV_LogReg.best_estimator_.score(x_test_enc, y_test)))
print("Best model:", GSCV_LogReg.best_estimator_)
mask =
GSCV_LogReg.best_estimator_.named_steps['feat_select'].get_support()
print("Best features:", df_train_enc.columns[mask])

LogReg_pred = GSCV_LogReg.predict(x_test_enc)

import matplotlib.pyplot as plt
cm = confusion_matrix(y_test, LogReg_pred,
labels=GSCV_LogReg.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=GSCV_LogReg.classes_)
disp.plot()
plt.title("Logistic Regression Confusion Matrix")
plt.show()
```

```
print("Classification report Logistic Regression:\n",
classification_report(y_test, LogReg_pred))


from sklearn.preprocessing import MinMaxScaler, StandardScaler

from sklearn.feature_selection import SelectKBest, SelectPercentile

from sklearn.ensemble import RandomForestClassifier

from sklearn.pipeline import Pipeline

from sklearn.model_selection import GridSearchCV, StratifiedKFold

import numpy as np


pipe_RF=[('data scaling', StandardScaler()),

         ('feature select', SelectKBest()),

         ('clf',
RandomForestClassifier(random_state=94,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)

GSCV_RF = GridSearchCV(estimator_RF, params_grid_RF, cv=SKF)

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

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

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

RF_pred = GSCV_RF.predict(x_test_enc)

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 Random Forest: \n",
classification_report(y_test, RF_pred))

import pickle

with open('BestModel_CLF_RandomForest_SQLAlchemy.pkl', 'wb') as r:
    pickle.dump((GSCV_RF), r)

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

Notebook_REGRESI_B_SQLAlchemy_LassoRegression_VS_RandomForest_Joti

```
import pandas as pd
import numpy as np

df_SQLAlchemy = pd.read_csv("Dataset UTS_Gasal 2425.csv")
df_SQLAlchemy.head(10)

df_SQLAlchemy2=df_SQLAlchemy.drop(['category'], axis=1)
df_SQLAlchemy2.head(10)

df_SQLAlchemy2.info()

print("data null \n",df_SQLAlchemy2.isnull().sum())
print("\ndata kosong \n",df_SQLAlchemy2.empty)
print("\ndata nan \n",df_SQLAlchemy2.isna().sum())

df_SQLAlchemy2.describe()
```

```
print("Sebelum drop missing value", df_SQLAlchemy2.shape)

df_SQLAlchemy2 = df_SQLAlchemy2.dropna(how="any", inplace=False)

print("Sesudah drop missing value", df_SQLAlchemy2.shape)


df_SQLAlchemy2['price'].value_counts()


median_chole = df_SQLAlchemy2['price'].median()


print(median_chole)


df_SQLAlchemy2['price'] =
df_SQLAlchemy2['price'].fillna(median_chole)


print("Sebelum pengecekan data duplikat, ",df_SQLAlchemy2.shape)

df_SQLAlchemy3=df_SQLAlchemy2.drop_duplicates(keep='last')

print("Setelah pengecekan data duplikat, ",df_SQLAlchemy3.shape)


import matplotlib.pyplot as plt


df_SQLAlchemy2.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_sqlalchemy_clean = remove_outlier(df_SQLAlchemy3)

print("Jumlah baris DataFrame sebelum dibuang outlier",
df_SQLAlchemy3.shape[0])

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

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


plt.gca().invert_yaxis()

plt.show()


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

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

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


from sklearn.model_selection import train_test_split


x_regress = df_sqlalchemy_clean.drop(columns=['price'], axis=1)

y_regress = df_sqlalchemy_clean['price']


x_train_sqlalchemy, x_test_sqlalchemy, y_train_sqlalchemy,
y_test_sqlalchemy = train_test_split(x_regress, y_regress,
test_size=0.25, random_state=94)


print(x_train_sqlalchemy.shape)

print(x_test_sqlalchemy.shape)


from sklearn.preprocessing import OneHotEncoder

from sklearn.compose import make_column_transformer
```

```
cat_cols =
x_train_sqlalchemy.select_dtypes(include=['object']).columns.tolist(
)
print("Kolom Kategorik:", cat_cols)

transformer = make_column_transformer(
    (OneHotEncoder(), cat_cols),
    remainder = 'passthrough'
)

x_train_enc = transformer.fit_transform(x_train_sqlalchemy)
x_test_enc = transformer.transform(x_test_sqlalchemy)

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

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

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

pipe_Lasso = Pipeline(steps=[
            ('scale', StandardScaler()),
            ('feature_selection',
SelectKBest(score_func=f_regression)),
            ('reg', Lasso(max_iter=1000))
            ])
```

```
param_grid_Lasso = [
    {
        'scale': [StandardScaler()],
        'feature_selection': [SelectKBest(f_regression)],
        'feature_selection__k': np.arange(1, 20),
        'reg__alpha': [0.01, 0.1, 1, 10, 100],
    },
    {
        'scale': [StandardScaler()],
        'feature_selection': [SelectPercentile(f_regression)],
        'feature_selection__percentile': np.arange(10, 100, 10),
        'reg__alpha': [0.01, 0.1, 1, 10, 100],
    },
    {
        'scale': [MinMaxScaler()],
        'feature_selection': [SelectKBest(f_regression)],
        'feature_selection__k': np.arange(1, 20),
        'reg__alpha': [0.01, 0.1, 1, 10, 100],
    },
    {
        'scale': [MinMaxScaler()],
        'feature_selection': [SelectPercentile(f_regression)],
        'feature_selection__percentile': np.arange(10, 100, 10),
        'reg__alpha': [0.01, 0.1, 1, 10, 100],
    }
]

KF = KFold(n_splits=5, shuffle=True, random_state=94)

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

```
GSCV_Lasso.fit(x_train_enc, y_train_sqlalchemy)


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

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

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

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


Lasso_predict = GSCV_Lasso.predict(x_test_enc)


mse_Lasso = mean_squared_error(y_test_sqlalchemy, Lasso_predict)

mae_Lasso = mean_absolute_error(y_test_sqlalchemy, 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 = pd.DataFrame(y_test_sqlalchemy, columns=['price'])

df_results = pd.DataFrame(y_test_sqlalchemy)

df_results['Lasso Prediction'] = Lasso_predict


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


df_results.head()


df_results.describe()


from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import GridSearchCV, KFold
from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler, MinMaxScaler

from sklearn.feature_selection import SelectKBest, SelectPercentile,
f_regression

from sklearn.metrics import mean_absolute_error, mean_squared_error


pipe_RF = Pipeline(steps=[

            ('scale', StandardScaler()),

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

            ('reg', RandomForestRegressor(random_state=94))

            ])


param_grid_RF = [

    {

        'scale': [StandardScaler()],

        'feature_selection': [SelectKBest(f_regression)],

        'feature_selection__k': np.arange(2, 6),

        'reg__n_estimators': [100, 150],

        'reg__max_depth': [4, 5],

    },

    {

        'scale': [StandardScaler()],

        'feature_selection': [SelectPercentile(f_regression)],

        'feature_selection__percentile': np.arange(20, 50),

        'reg__n_estimators': [100, 150],

        'reg__max_depth': [4, 5],

    },

    {

        'scale': [MinMaxScaler()],

        'feature_selection': [SelectKBest(f_regression)],

        'feature_selection__k': np.arange(2, 6),

        'reg__n_estimators': [100, 150],
        'reg__max_depth': [4, 5],
    },
    {
        'scale': [MinMaxScaler()],
        'feature_selection': [SelectPercentile(f_regression)],
        'feature_selection__percentile': np.arange(20, 50),
        'reg__n_estimators': [100, 150],
        'reg__max_depth': [4, 5],
    }
]

KF = KFold(n_splits=5, shuffle=True, random_state=94)

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

GSCV_RF.fit(x_train_enc, y_train_sqlalchemy)

print("Best model: {}".format(GSCV_RF.best_estimator_))
print("RF best parameters: {}".format(GSCV_RF.best_params_))

RF_predict = GSCV_RF.predict(x_test_enc)

mse_RF = mean_squared_error(y_test_sqlalchemy, RF_predict)
mae_RF = mean_absolute_error(y_test_sqlalchemy, RF_predict)

print("RF Mean Squared Error (MSE): {}".format(mse_RF))
print("RF Mean Absolute Error (MAE): {}".format(mae_RF))
print("RF Root Mean Squared Error: {}".format(np.sqrt(mse_RF)))

df_results['RF Prediction'] = RF_predict
df_results = pd.DataFrame(y_test_sqlalchemy)
```

```
df_results['RF Prediction'] = RF_predict


df_results['Selisih_price_RF'] = df_results['RF Prediction'] -
df_results['price']


df_results.head()


df_results.describe()


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


df_results['Lasso Prediction'] = Lasso_predict

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


df_results['RF Prediction'] = RF_predict

df_results['Selisih_price_RF'] = df_results['price'] -
df_results['RF Prediction']


df_results.head()


df_results.describe()


import matplotlib.pyplot as plt


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


data_len = range(len(y_test_sqlalchemy))


plt.scatter(data_len, df_results.price, label="Actual",
color="navy")

plt.plot(data_len, df_results["Lasso Prediction"], label="Lasso
Prediction", color="limegreen", linewidth=3, linestyle="--")

plt.plot(data_len, df_results["RF Prediction"], label="RF
Prediction", color="crimson", linewidth=1, linestyle=":")
```

```
plt.legend()
plt.show()


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


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


mae_RF = mean_absolute_error(df_results['price'], df_results['RF
Prediction'])
rmse_RF = np.sqrt(mean_squared_error(df_results['price'],
df_results['RF Prediction']))
RF_feature_count =
GSCV_RF.best_params_['feature_selection__percentile']


print(f"Lasso MAE: {mae_lasso}, Lasso RMSE: {rmse_lasso}, Lasso
Feature Count: {lasso_feature_count}")
print(f"RF MAE: {mae_RF}, RF RMSE: {rmse_RF}, RF Feature Count:
{RF_feature_count}")



import pickle


with open('BestModel_REG_RandomForest_SQLAlchemy.pkl', 'wb') as r:
    pickle.dump((GSCV_RF), r)


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

Notebook_REGRESI_B_SQLAlchemy_RidgeRegression_VS_SupportVectorRegressor_Jhonatan-Yoga

```
import pandas as pd
import numpy as np

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

df_SQLAlchemy.head(10)

df_SQLAlchemy2 = df_SQLAlchemy.drop(['category'], axis=1)
df_SQLAlchemy2.head()

df_SQLAlchemy2.info()

print("data null \n",df_SQLAlchemy2.isnull().sum())
print("\ndata kosong \n",df_SQLAlchemy2.empty)
print("\ndata nan \n",df_SQLAlchemy2.isna().sum())

df_SQLAlchemy2.describe()

print("Sebelum drop missing value", df_SQLAlchemy2.shape)
df_SQLAlchemy2 = df_SQLAlchemy2.dropna(how="any", inplace=False)
print("Sesudah drop missing value", df_SQLAlchemy2.shape)

df_SQLAlchemy2['price'].value_counts()

print("data null \n", df_SQLAlchemy2.isnull().sum())
print("data kosong \n", df_SQLAlchemy2.empty)
print("data nan \n", df_SQLAlchemy2.isna().sum())

import matplotlib.pyplot as plt
```

```
df_SQLAlchemy2.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_sqlalchemy_clean = remove_outlier(df_SQLAlchemy2)
print("Jumlah baris DataFrame sebelum dibuang
outlier",  df_SQLAlchemy2.shape[0])
print("Jumlah baris DataFrame setelah dibuang
outlier",  df_sqlalchemy_clean.shape[0])
df_sqlalchemy_clean.price.plot(kind='box', vert=True)

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

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

from sklearn.model_selection import train_test_split
```

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

X_train_sqlalchemy, X_test_sqlalchemy, y_train_sqlalchemy,
y_test_sqlalchemy = train_test_split(X_regress, y_regress,
test_size=0.25, random_state=94)

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

cat_cols=X_train_sqlalchemy.select_dtypes(include=['object']).columns.tolist()
print("Kolom Kategorik:", cat_cols)

transformer = make_column_transformer(
    (OneHotEncoder(), cat_cols),
    remainder='passthrough'
)

X_train_enc = transformer.fit_transform(X_train_sqlalchemy)
X_test_enc = transformer.transform(X_test_sqlalchemy)

df_train_enc = pd.DataFrame(X_train_enc,
columns=transformer.get_feature_names_out())
df_test_enc = pd.DataFrame(X_test_enc, columns =
transformer.get_feature_names_out())

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

from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, MinMaxScaler

from sklearn.feature_selection import SelectKBest, SelectPercentile,
f_regression

from sklearn.metrics import mean_absolute_error, mean_squared_error


pipe_Ridge = Pipeline(steps=[

            ('scale', StandardScaler()),

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

            ('reg', Ridge())

            ])

param_grid_Ridge = [

    {

        'scale': [StandardScaler(), MinMaxScaler()],

        'feature_selection': [SelectKBest(f_regression)],

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

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

    },

    {

        'scale': [StandardScaler()],

        'feature_selection': [SelectPercentile(f_regression)],

        'feature_selection__percentile': np.arange(10, 100, 10),

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

    },

    {

        'scale': [MinMaxScaler()],

        'feature_selection': [SelectKBest(f_regression)],

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

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

    },

    {

        'scale': [MinMaxScaler()],

        'feature_selection': [SelectPercentile(f_regression)],
        'feature_selection__percentile': np.arange(10, 100, 10),
        'reg__alpha': [0.01, 0.1, 1, 10, 100],
    }
]

KF = KFold(n_splits=5, shuffle=True, random_state=94)

GSCV_RR = GridSearchCV(pipe_Ridge, param_grid_Ridge, cv=KF,
                        scoring='neg_mean_squared_error')

GSCV_RR.fit(X_train_enc, y_train_sqlalchemy)

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_sqlalchemy, Ridge_predict)
mae_Ridge = mean_absolute_error(y_test_sqlalchemy, 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)))

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

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


df_results.head()


df_results.describe()


from sklearn.svm import SVR

from sklearn.model_selection import GridSearchCV, KFold

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler, MinMaxScaler

from sklearn.feature_selection import SelectKBest, SelectPercentile,
f_regression

from sklearn.metrics import mean_absolute_error, mean_squared_error

import numpy as np


pipe_SVR = Pipeline(steps=[

            ('scale', StandardScaler()),

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

            ('reg', SVR(kernel='linear'))

            ])


param_grid_SVR = [

    {

        'scale': [StandardScaler()],

        'feature_selection': [SelectKBest(f_regression)],

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

        'reg__C': [0.1, 1, 10, 100],

        'reg__epsilon': [0.01, 0.1, 1],

    },

    {

        'scale': [StandardScaler()],

        'feature_selection': [SelectPercentile(f_regression)],
        'feature_selection__percentile': np.arange(10, 100, 10),
        'reg__C': [0.1, 1, 10, 100],
        'reg__epsilon': [0.01, 0.1, 1],
    },
    {
        'scale': [MinMaxScaler()],
        'feature_selection': [SelectKBest(f_regression)],
        'feature_selection__k': np.arange(1, 20),
        'reg__C': [0.1, 1, 10, 100],
        'reg__epsilon': [0.01, 0.1, 1],
    },
    {
        'scale': [MinMaxScaler()],
        'feature_selection': [SelectPercentile(f_regression)],
        'feature_selection__percentile': np.arange(10, 100, 10),
        'reg__C': [0.1, 1, 10, 100],
        'reg__epsilon': [0.01, 0.1, 1],
    }
]

KF = KFold(n_splits=5, shuffle=True, random_state=94)

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

GSCV_SVR.fit(X_train_enc, y_train_sqlalchemy)

print("Best model: {}".format(GSCV_SVR.best_estimator_))
print("SVR best parameters: {}".format(GSCV_SVR.best_params_))
print("Support Vector Regressor koefisien tidak tersedia untuk
kernel non-linear.")

SVR_predict = GSCV_SVR.predict(X_test_enc)
```

```
mse_SVR = mean_squared_error(y_test_sqlalchemy, SVR_predict)
mae_SVR = mean_absolute_error(y_test_sqlalchemy, 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_sqlalchemy)
df_results['SVR Prediction'] = SVR_predict

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

df_results.head()

df_results.describe()

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

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

df_results['SVR Prediction'] = SVR_predict
df_results['Selisih_price_SVR'] = df_results['price'] -
df_results['SVR Prediction']

df_results.head()

df_results.describe()
import matplotlib.pyplot as plt
```

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


data_len = range(len(y_test_sqlalchemy))


plt.scatter(data_len, df_results.price, label="Actual",
color="midnightblue")
plt.plot(data_len, df_results['Ridge Prediction'], label="Ridge
Prediction", color="darkorange", linewidth=4, linestyle="dashed")  #
Oranye tua untuk prediksi Ridge
plt.plot(data_len, df_results['SVR Prediction'], label="SVR
Prediction", color="mediumvioletred", linewidth=2, linestyle="-
.")  # Merah muda keunguan untuk prediksi SVR


plt.legend()
plt.show()
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np


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


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


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

Streamlit.

```
import os
import pickle
import streamlit as st
from streamlit_option_menu import option_menu

model_path = 'BestModel_CLF_RandomForest_SQLAlchemy.pkl'
model_path2 = 'BestModel_REG_RandomForest_SQLAlchemy.pkl'

with open(model_path, 'rb') as f:
    lr_model = pickle.load(f)

with open(model_path2, 'rb') as f:
    svr_model = pickle.load(f)

with st.sidebar:
    selected = option_menu('SQLAlchemy UTS ML 24/25',
                           ['Klasifikasi', 'Regresi', 'Catatan'],
                           default_index=0)

if selected == 'Klasifikasi':
    st.title('Klasifikasi')
    st.write('Untuk Inputan File dataset (csv) bisa menggunakan
st.file_uploader')
    file = st.file_uploader('Masukkan File', type=['csv', 'txt'])

    LuasTanah = st.number_input('Input luas tanah dalam meter
persegi (squaremeters): ', 0)
    JumlahKamar = st.slider('Input Jumlah Kamar (numberofrooms): ',
0, 100)
    Halaman = st.radio('Apakah memiliki halaman (hasyard)?', ['Yes',
'No'])
    KolamRenang = st.radio('Apakah memiliki kolam renang
(haspool)?', ['Yes', 'No'])
    JumlahLantai = st.slider('Input Jumlah Lantai (floors): ', 0,
100)
```

```
    KodeLokasi = st.number_input('Input Kode Lokasi (citycode): ',
0)
    CityPartRange = st.slider('Input Ekslusivitas Kawasan
(citypartrange): ', 0, 10)
    JumlahPemilik = st.slider('Jumlah Pemilik Sebelumnya
(numprevowners): ', 0, 10)
    TahunPembuatan = st.number_input('Input Tahun Pembuatan (made):
', 0)
    GedungBaru = st.radio('Apakah gedung baru atau bukan
(isnewbuilt)?', ['Old', 'New'])
    PelindungBadai = st.radio('Apakah memiliki pelindung badai
(hasstormprotector)?', ['Yes', 'No'])
    Basement = st.number_input('Input luas basement (basement): ',
0)
    Loteng = st.number_input('Input luas loteng (attic): ', 0)
    Garasi = st.number_input('Input luas garasi (garage): ', 0)
    Gudang = st.radio('Apakah memiliki Gudang (hasstorageroom)?',
['Yes', 'No'])
    RuangTamu = st.slider('Input Ruang Tamu (hasguestroom): ', 0,
10)

    input_halamanY = 1 if Halaman == "Yes" else 0
    input_kolamRenangY = 1 if KolamRenang == "Yes" else 0
    input_gedungBaruY = 1 if GedungBaru == "New" else 0
    input_pelindungBadaiY = 1 if PelindungBadai == "Yes" else 0
    input_gudangY = 1 if Gudang == "Yes" else 0

    input_data = [[
        LuasTanah, JumlahKamar, input_halamanY, 1 - input_halamanY,
input_kolamRenangY, 1 - input_kolamRenangY,
        JumlahLantai, KodeLokasi, CityPartRange, JumlahPemilik,
TahunPembuatan,
        input_gedungBaruY, 1 - input_gedungBaruY,
input_pelindungBadaiY, 1 - input_pelindungBadaiY,
        Basement, Loteng, Garasi, input_gudangY, 1 - input_gudangY,
RuangTamu
    ]]
    if st.button("Cek Kategori"):
        lr_model_prediction = lr_model.predict(input_data)
        st.write(f"Prediksi Model : {lr_model_prediction}")


if selected == 'Regresi':
    st.title('Regresi')
    st.write('Untuk Inputan File dataset (csv) bisa menggunakan
st.file_uploader')
    file = st.file_uploader('Masukkan File', type=['csv', 'txt'])


    LuasTanah = st.number_input('Input luas tanah dalam meter
persegi (squaremeters): ', 0)
    JumlahKamar = st.slider('Input Jumlah Kamar (numberofrooms): ',
0, 100)
    Halaman = st.radio('Apakah memiliki halaman (hasyard)?', ['Yes',
'No'])
    KolamRenang = st.radio('Apakah memiliki kolam renang
(haspool)?', ['Yes', 'No'])
    JumlahLantai = st.slider('Input Jumlah Lantai (floors): ', 0,
100)
    KodeLokasi = st.number_input('Input Kode Lokasi (citycode): ',
0)
    CityPartRange = st.slider('Input Ekslusivitas Kawasan
(citypartrange): ', 0, 10)
    JumlahPemilik = st.slider('Jumlah Pemilik Sebelumnya
(numprevowners): ', 0, 10)
    TahunPembuatan = st.number_input('Input Tahun Pembuatan (made):
', 0)
    GedungBaru = st.radio('Apakah gedung baru atau bukan
(isnewbuilt)?', ['Old', 'New'])
    PelindungBadai = st.radio('Apakah memiliki pelindung badai
(hasstormprotector)?', ['Yes', 'No'])
    Basement = st.number_input('Input luas basement (basement): ',
0)
    Loteng = st.number_input('Input luas loteng (attic): ', 0)
    Garasi = st.number_input('Input luas garasi (garage): ', 0)
    Gudang = st.radio('Apakah memiliki Gudang (hasstorageroom)?',
['Yes', 'No'])
    RuangTamu = st.slider('Input Ruang Tamu (hasguestroom): ', 0,
10)

    input_halamanY = 1 if Halaman == "Yes" else 0
    input_kolamRenangY = 1 if KolamRenang == "Yes" else 0
    input_gedungBaruY = 1 if GedungBaru == "New" else 0
    input_pelindungBadaiY = 1 if PelindungBadai == "Yes" else 0
    input_gudangY = 1 if Gudang == "Yes" else 0

    input_data = [[
        LuasTanah, JumlahKamar, input_halamanY, 1 - input_halamanY,
input_kolamRenangY, 1 - input_kolamRenangY,
        JumlahLantai, KodeLokasi, CityPartRange, JumlahPemilik,
TahunPembuatan,
        input_gedungBaruY, 1 - input_gedungBaruY,
input_pelindungBadaiY, 1 - input_pelindungBadaiY,
        Basement, Loteng, Garasi, input_gudangY, 1 - input_gudangY,
RuangTamu
    ]]

    if st.button("Prediksi Price"):
        svr_model_prediction = svr_model.predict(input_data)
        st.markdown(f"Prediksi Harga properti : $
{svr_model_prediction[0]:.2f}")

if selected == 'Catatan':
    st.title('Catatan')
    st.write('Untuk memunculkan sidebar agar tidak error ketika di
run, silahkan install library streamlit option menu dengan perintah
"pip install streamlit-option-menu".')
    st.write('Pada contoh di atas ada 2 yaitu Klasifikasi dan
Regresi.')
    st.write('Silahkan sesuaikan dengan arsitektur code anda pada
notebook.')
```

```
    st.write('Untuk lebih lanjut bisa di akses pada
https://streamlit.io/')
    st.write('Link desain streamlit dapat diakses pada
https://aputsc-6jzfv4fiuzj84mfc7k7.streamlit.app/')
    st.write('Untuk requirements yang dibutuhkan untuk deploy online
di github ada 5 yaitu streamlit, scikit-learn, pandas, numpy,
streamlit-option-menu.')

import os
import pickle
import streamlit as st
from streamlit_option_menu import option_menu

model_path = 'BestModel_CLF_RandomForest_SQLAlchemy.pkl'
model_path2 = 'BestModel_REG_RandomForest_SQLAlchemy.pkl'

with open(model_path, 'rb') as f:
    lr_model = pickle.load(f)

with open(model_path2, 'rb') as f:
    svr_model = pickle.load(f)

with st.sidebar:
    selected = option_menu('SQLAlchemy UTS ML 24/25',
                           ['Klasifikasi', 'Regresi', 'Catatan'],
                           default_index=0)

if selected == 'Klasifikasi':
    st.title('Klasifikasi')
    st.write('Untuk Inputan File dataset (csv) bisa menggunakan
st.file_uploader')
    file = st.file_uploader('Masukkan File', type=['csv', 'txt'])

    LuasTanah = st.number_input('Input luas tanah dalam meter
persegi (squaremeters): ', 0)
```

```
    JumlahKamar = st.slider('Input Jumlah Kamar (numberofrooms): ',
0, 100)
    Halaman = st.radio('Apakah memiliki halaman (hasyard)?', ['Yes',
'No'])
    KolamRenang = st.radio('Apakah memiliki kolam renang
(haspool)?', ['Yes', 'No'])
    JumlahLantai = st.slider('Input Jumlah Lantai (floors): ', 0,
100)
    KodeLokasi = st.number_input('Input Kode Lokasi (citycode): ',
0)
    CityPartRange = st.slider('Input Ekslusivitas Kawasan
(citypartrange): ', 0, 10)
    JumlahPemilik = st.slider('Jumlah Pemilik Sebelumnya
(numprevowners): ', 0, 10)
    TahunPembuatan = st.number_input('Input Tahun Pembuatan (made):
', 0)
    GedungBaru = st.radio('Apakah gedung baru atau bukan
(isnewbuilt)?', ['Old', 'New'])
    PelindungBadai = st.radio('Apakah memiliki pelindung badai
(hasstormprotector)?', ['Yes', 'No'])
    Basement = st.number_input('Input luas basement (basement): ',
0)
    Loteng = st.number_input('Input luas loteng (attic): ', 0)
    Garasi = st.number_input('Input luas garasi (garage): ', 0)
    Gudang = st.radio('Apakah memiliki Gudang (hasstorageroom)?',
['Yes', 'No'])
    RuangTamu = st.slider('Input Ruang Tamu (hasguestroom): ', 0,
10)

    input_halamanY = 1 if Halaman == "Yes" else 0
    input_kolamRenangY = 1 if KolamRenang == "Yes" else 0
    input_gedungBaruY = 1 if GedungBaru == "New" else 0
    input_pelindungBadaiY = 1 if PelindungBadai == "Yes" else 0
    input_gudangY = 1 if Gudang == "Yes" else 0

    input_data = [[
        LuasTanah, JumlahKamar, input_halamanY, 1 - input_halamanY,
input_kolamRenangY, 1 - input_kolamRenangY,
        JumlahLantai, KodeLokasi, CityPartRange, JumlahPemilik, 
TahunPembuatan,
        input_gedungBaruY, 1 - input_gedungBaruY, 
input_pelindungBadaiY, 1 - input_pelindungBadaiY,
        Basement, Loteng, Garasi, input_gudangY, 1 - input_gudangY, 
RuangTamu
    ]]

    if st.button("Cek Kategori"):
        lr_model_prediction = lr_model.predict(input_data)
        st.write(f"Prediksi Model : {lr_model_prediction}")

if selected == 'Regresi':
    st.title('Regresi')
    st.write('Untuk Inputan File dataset (csv) bisa menggunakan 
st.file_uploader')
    file = st.file_uploader('Masukkan File', type=['csv', 'txt'])

    LuasTanah = st.number_input('Input luas tanah dalam meter 
persegi (squaremeters): ', 0)
    JumlahKamar = st.slider('Input Jumlah Kamar (numberofrooms): ', 
0, 100)
    Halaman = st.radio('Apakah memiliki halaman (hasyard)?', ['Yes', 
'No'])
    KolamRenang = st.radio('Apakah memiliki kolam renang 
(haspool)?', ['Yes', 'No'])
    JumlahLantai = st.slider('Input Jumlah Lantai (floors): ', 0, 
100)
    KodeLokasi = st.number_input('Input Kode Lokasi (citycode): ', 
0)
    CityPartRange = st.slider('Input Ekslusivitas Kawasan 
(citypartrange): ', 0, 10)
    JumlahPemilik = st.slider('Jumlah Pemilik Sebelumnya 
(numprevowners): ', 0, 10)
    TahunPembuatan = st.number_input('Input Tahun Pembuatan (made): 
', 0)
    GedungBaru = st.radio('Apakah gedung baru atau bukan 
(isnewbuilt)?', ['Old', 'New'])
```

```
    PelindungBadai = st.radio('Apakah memiliki pelindung badai
(hasstormprotector)?', ['Yes', 'No'])
    Basement = st.number_input('Input luas basement (basement): ',
0)
    Loteng = st.number_input('Input luas loteng (attic): ', 0)
    Garasi = st.number_input('Input luas garasi (garage): ', 0)
    Gudang = st.radio('Apakah memiliki Gudang (hasstorageroom)?',
['Yes', 'No'])
    RuangTamu = st.slider('Input Ruang Tamu (hasguestroom): ', 0,
10)

    input_halamanY = 1 if Halaman == "Yes" else 0
    input_kolamRenangY = 1 if KolamRenang == "Yes" else 0
    input_gedungBaruY = 1 if GedungBaru == "New" else 0
    input_pelindungBadaiY = 1 if PelindungBadai == "Yes" else 0
    input_gudangY = 1 if Gudang == "Yes" else 0

    input_data = [[
        LuasTanah, JumlahKamar, input_halamanY, 1 - input_halamanY,
input_kolamRenangY, 1 - input_kolamRenangY,
        JumlahLantai, KodeLokasi, CityPartRange, JumlahPemilik,
TahunPembuatan,
        input_gedungBaruY, 1 - input_gedungBaruY,
input_pelindungBadaiY, 1 - input_pelindungBadaiY,
        Basement, Loteng, Garasi, input_gudangY, 1 - input_gudangY,
RuangTamu
    ]]

    if st.button("Prediksi Price"):
        svr_model_prediction = svr_model.predict(input_data)
        st.markdown(f"Prediksi Harga properti : $
{svr_model_prediction[0]:.2f}")

if selected == 'Catatan':
    st.title('Catatan')
    st.write('Untuk memunculkan sidebar agar tidak error ketika di
run, silahkan install library streamlit option menu dengan perintah
"pip install streamlit-option-menu".')

    st.write('Pada contoh di atas ada 2 yaitu Klasifikasi dan
Regresi.')

    st.write('Silahkan sesuaikan dengan arsitektur code anda pada
notebook.')

    st.write('Untuk lebih lanjut bisa di akses pada
https://streamlit.io/')

    st.write('Link desain streamlit dapat diakses pada
https://aputsc-6jzfv4fiuzj84mfc7k7.streamlit.app/')

    st.write('Untuk requirements yang dibutuhkan untuk deploy online
di github ada 5 yaitu streamlit, scikit-learn, pandas, numpy,
streamlit-option-menu.')
```