CODE UTS (Skicit-Learn) PMDPM 2024

1. Notebook Klasifikasi

- **Random Forest & Logistic Regression (Oktavio)**

```
import pandas as pd
import numpy as np

df_2425 = pd.read_csv(r'C:\Kuliah\ML\UTS\Model Suprevised Learning (Praktek)\Tugas3_X_YYYYY\Dataset_UTS_Gasal_2425.csv')
df_2425.head(20)


df_2425_a=df_2425.drop(['price'], axis=1)
df_2425_a.head()


df_2425_a.info()


df_2425_a.describe()


print("data null \n", df_2425_a.isnull().sum())
print("data kosong \n", df_2425_a.empty)
print("data nan \n", df_2425_a.isna().sum())


print("Sebelum Pengecekan data duplikat, ", df_2425_a.shape)
df_2425_b=df_2425_a.drop_duplicates (keep='last')
print("Setelah Pengecekan data duplikat, ", df_2425_b.shape)


from sklearn.model_selection import train_test_split
x = df_2425_b.drop(columns=['category'], axis=1)
y=y=df_2425_b ['category']

x_train, x_test,y_train,y_test = train_test_split(x, y, test_size=0.25, random_state=86)

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_names_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=86)

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=86,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_RF_Scikit-learn.pkl', 'wb') as r:
    pickle.dump((GSCV_RF), r)

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

- **Gradient Boosting Classifier & Support Vector Machine (Arya)**

```
import pandas as pd

import numpy as np

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

df_category.head(20)


df_category2=df_category.drop('price' ,axis=1)

df_category2.head(50)


df_category2.info()

df_category2.describe()


print("data null \n",df_category2.isnull().sum())

print("\ndata kosong \n",df_category2.empty)

print("\ndata nan \n",df_category2.isna().sum())


print("Sebelum pengecekan data duplikat, ",df_category2.shape)

df_category3=df_category2.drop_duplicates(keep='last')

print("Setelah pengecekan data duplikat, ",df_category3.shape)


from sklearn.model_selection import train_test_split

x = df_category3.drop(columns=['category'],axis=1)

y = df_category3['category']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25,
random_state=86)

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_names_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=68)

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=68))])


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))


import pickle


with open('BestModel_CLF_GBT_Scikit-Learn.pkl','wb') as r:

    pickle.dump((GSCV_GBT),r)

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

2. Notebook Regresi

- **Ridge Regression & Support Vector Regressor (Alexa)**

```
import pandas as pd
import  numpy as np

df_UTS  =  pd.read_csv(r'D:\SEMESTER  5\ML\Tubezzz\Regresi  1\Dataset  UTS_Gasal
2425.csv')
df_UTS.head(10)
```

```
df_UTS2 = df_UTS.drop(['category'], axis=1)
df_UTS2.head()

df_UTS2.info()

df_UTS2.describe()

print("Sebelum Pengecekan data duplikat, ", df_UTS2.shape)
df_UTS2=df_UTS2.drop_duplicates(keep='last')
print("Setelah Pengecekan data duplikat, ", df_UTS2.shape)

print("data null \n", df_UTS2.isnull().sum())
print("data kosong \n", df_UTS2.empty)
print("data nan \n", df_UTS2.isna().sum())

print("Sebelum drop missing value", df_UTS2.shape)
df_UTS2 = df_UTS2.dropna(how="any", inplace=False)
print("Sesudah drop missing value", df_UTS2.shape)

median_chole = df_UTS2['price'].median()
print(median_chole)
df_UTS2['price'] = df_UTS2['price'].fillna(median_chole)

import matplotlib.pyplot as plt
df_UTS2.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_UTS_clean = remove_outlier(df_UTS2)
print("Jumlah baris DataFrame sebelum dibuang outlier",  df_UTS2.shape[0])
print("Jumlah baris DataFrame setelah dibuang outlier",  df_UTS_clean.shape[0])
df_UTS_clean.price.plot(kind='box', vert=True)

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

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

from sklearn.model_selection import train_test_split

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

X_train_UTS, X_test_UTS, y_train_UTS, y_test_UTS = train_test_split(X_regress,
y_regress, test_size=0.25, random_state=86)

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

```
cat_cols=X_train_UTS.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_UTS)
X_test_enc = transformer.transform(X_test_UTS)

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=86)

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

```
GSCV_RR.fit(X_train_enc, y_train_UTS)


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_UTS, Ridge_predict)

mae_Ridge = mean_absolute_error(y_test_UTS, 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_UTS, columns=['price'])

df_results = pd.DataFrame(y_test_UTS)

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=86)

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

GSCV_SVR.fit(X_train_enc, y_train_UTS)

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_UTS, SVR_predict)
mae_SVR = mean_absolute_error(y_test_UTS, 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_UTS)
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_UTS})


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()


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

data_len = range(len(y_test_UTS))

plt.scatter(data_len, df_results.price, label="actual", color="blue")

plt.plot(data_len, df_results['Ridge Prediction'], label="Ridge Prediction", color="green", linewidth=4, linestyle="dashed")

plt.plot(data_len, df_results['SVR Prediction'], label="SVR Prediction", color="yellow", linewidth=2, linestyle="-.")

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}")


import pickle

best_model = GSCV_SVR.best_estimator_

with open('BestModel_REG_SVR_Scikit-Learn.pkl', 'wb') as f:

    pickle.dump(best_model, f)

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

- **Lasso Regression & Support Vector Regressor (Christopher)**

```
import pandas as pd
import numpy as np

df_uts = pd.read_csv(r"C:\Users\Lenovo\OneDrive\Documents\K\S 5\PMdPM\Projek-UTS-PMDPM_B_Scikit-Learn\Dataset UTS_Gasal 2425.csv")
df_uts.head(10)

df_uts.info()

df_uts.describe()

print("Data Null \n", df_uts.isnull().sum())
print("Data Kosong \n", df_uts.empty)
print("Data NaN \n", df_uts.isna().sum())

df_uts2 = df_uts.drop(['category'], axis=1)
df_uts2.head()
```

```
df_uts2['price'].value_counts()

print("Sebelum drop missing value", df_uts2.shape)
df_uts2 = df_uts2.dropna(how="any", inplace=False)
print("Sesudah drop missing value", df_uts2.shape)


median_chole = df_uts2['price'].median()

print(median_chole)

df_uts2['price'] = df_uts2['price'].fillna(median_chole)

print("Sebelum Pengecekan data duplikat, ", df_uts2.shape)
df_uts3 = df_uts2.drop_duplicates(keep='last')
print("Sebelum Pengecekan data duplikat, ", df_uts3.shape)

import matplotlib.pyplot as plt

df_uts3.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_uts_clean = remove_outlier(df_uts3)
print("Jumlah baris DataFrame sebelum dibuang outlier", df_uts3.shape[0])
print("Jumlah baris DataFrame sesudah dibuang outlier", df_uts_clean.shape[0])
df_uts_clean.price.plot(kind='box', vert=True)

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

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

from sklearn.model_selection import train_test_split
```

```
x_regress = df_uts_clean.drop(columns=['price'], axis=1)
y_regress = df_uts_clean['price']

x_train_uts, x_test_uts, y_train_uts, y_test_uts = train_test_split(x_regress,
y_regress, test_size=0.25, random_state=86)

print(x_train_uts.shape)
print(x_test_uts.shape)

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

cat_cols = x_train_uts.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_uts)
x_test_enc = transformer.transform(x_test_uts)

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=86)

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

GSCV_Lasso.fit(x_train_enc, y_train_uts)

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_uts, Lasso_predict)
mae_Lasso = mean_absolute_error(y_test_uts, 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_uts, columns=['price'])
df_results = pd.DataFrame(y_test_uts)
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=86))
            ])

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=86)

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

GSCV_RF.fit(x_train_enc, y_train_uts)

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_uts, RF_predict)
mae_RF = mean_absolute_error(y_test_uts, 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_uts)
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_uts})

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_uts))

plt.scatter(data_len, df_results.price, label="actual", color="blue")

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

plt.plot(data_len,   df_results["RF   Prediction"],   label="RF   Prediction",
color="red", 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}")
```

3. Python

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

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

if selected == 'Klasifikasi':
    model = r'BestModel_CLF_GBT_Scikit-Learn.pkl'

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

        GBT_model = loaded_model

        st.title("Prediksi Jenis Rumah")
        st.write("Aplikasi ini membantu user untuk mengecek jenis rumah yang
ingin dibeli")

        squaremeters = st.number_input("Luas", min_value=0)
        numberofrooms = st.number_input("Jumlah Kamar", min_value=0)
        hasyard = st.selectbox("Apakah Memiliki Halaman?", ["yes", "no"])
        haspool = st.selectbox("Apakah Memiliki Kolam Renang?", ["yes", "no"])
        floors = st.number_input("Jumlah Lantai", min_value=0)
        citycode = st.number_input("Kode Kota", min_value=0)
        citypartrange = st.number_input("Rentang Partisi Kota", min_value=0)
```

```
        numprevowners = st.number_input("Jumlah Pemilik Sebelumnya",
min_value=0)
        made = st.number_input("Tahun Dibuat", min_value=1900, max_value=2024)
        isnewbuilt = st.selectbox("Apakah Baru Dibangun?", ["yes", "no"])
        hasstormprotector = st.selectbox("Apakah Memiliki Pelindung Badai?",
["yes", "no"])
        basement = st.number_input("luas Basement", min_value=0)
        attic = st.number_input("attic", min_value=0)
        garage = st.number_input("Luas Garasi", min_value=0)
        hasstorageroom = st.selectbox("Apakah Memiliki Ruang Penyimpanan?",
["yes", "no"])
        hasguestroom = st.number_input("Jumlah Kamar Tamu", min_value=0)


        if hasyard == "yes":
            input_hasyard = 1
        else:
            input_hasyard =0

        if haspool == "yes":
            input_haspool = 1
        else:
            input_haspool =0

        if isnewbuilt == "yes":
            input_isnewbuilt = 1
        else:
            input_isnewbuilt =0

        if hasstormprotector == "yes":
            input_hasstormprotector = 1
        else:
            input_hasstormprotector =0

        if hasstorageroom == "yes":
            input_hasstorageroom = 1
        else:
            input_hasstorageroom =0

        input_data = [[squaremeters, numberofrooms, input_hasyard,
input_haspool, floors, citycode, citypartrange,
                      numprevowners, made, input_isnewbuilt,
input_hasstormprotector, basement, attic, garage,
                      input_hasstorageroom, hasguestroom]]

        if st.button("Prediksi"):
            model_prediction = GBT_model.predict(input_data)
            outcome = {'Basic':'Basic', 'Luxury':'Luxury', 'Middle':'Middle'}
            st.write(f"Property tersebut merupakan kelas :
**{outcome[model_prediction[0]]}**")
    else:
        st.error("Model tidak ditemukan")
```

```
if selected == 'Regresi':
    model = r'BestModel_REG_SVR_Scikit-Learn.pkl'

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

        scaler = loaded_model[0]
        feature_selector = loaded_model[1]
        SVR_model = loaded_model[2]

        st.title("Prediksi Harga Rumah")
        st.write("Aplikasi ini membantu user untuk mengecek estimasi harga
rumah")

        squaremeters = st.number_input("Luas", min_value=0)
        numberofrooms = st.number_input("Jumlah Kamar", min_value=0)
        hasyard = st.selectbox("Apakah Memiliki Halaman?", ["yes", "no"])
        haspool = st.selectbox("Apakah Memiliki Kolam Renang?", ["yes", "no"])
        floors = st.number_input("Jumlah Lantai", min_value=0)
        citycode = st.number_input("Kode Kota", min_value=0)
        citypartrange = st.number_input("Rentang Partisi Kota", min_value=0)
        numprevowners = st.number_input("Jumlah Pemilik Sebelumnya",
min_value=0)
        made = st.number_input("Tahun Dibuat", min_value=1900, max_value=2024)
        isnewbuilt = st.selectbox("Apakah Baru Dibangun?", ["yes", "no"])
        hasstormprotector = st.selectbox("Apakah Memiliki Pelindung Badai?",
["yes", "no"])
        basement = st.number_input("luas Basement", min_value=0)
        attic = st.number_input("attic", min_value=0)
        garage = st.number_input("Luas Garasi", min_value=0)
        hasstorageroom = st.selectbox("Apakah Memiliki Ruang Penyimpanan?",
["yes", "no"])
        hasguestroom = st.number_input("Jumlah Kamar Tamu", min_value=0)


        if hasyard == "yes":
            input_hasyard = 1
        else:
            input_hasyard =0

        if haspool == "yes":
            input_haspool = 1
        else:
            input_haspool =0

        if isnewbuilt == "yes":
            input_isnewbuilt = 1
        else:
            input_isnewbuilt =0

        if hasstormprotector == "yes":
```

```
            input_hasstormprotector = 1
        else:
            input_hasstormprotector =0

        if hasstorageroom == "yes":
            input_hasstorageroom = 1
        else:
            input_hasstorageroom =0

        input_data    =    [squaremeters,    numberofrooms,    input_hasyard,
input_haspool, floors, citycode, citypartrange,
                         numprevowners,         made,         input_isnewbuilt,
input_hasstormprotector, basement, attic, garage,
                         input_hasstorageroom, hasguestroom]

        input_data_scaled = scaler.transform([input_data])
        input_data_selected = feature_selector.transform(input_data_scaled)

        if st.button("Prediksi"):
            model_prediction = SVR_model.predict(input_data_selected)
            formatted_price = "${:,.2f}".format(model_prediction[0])
            st.write(f"Hasil prediksi model: {formatted_price}")
    else:
        st.error("Model tidak ditemukan")

if selected == 'Catatan':
    st.title('Catatan')
    st.write('''1. Untuk memunculkan sidebar agar tidak error ketika di run,
silahkan install library streamlit option menu
            di  terminal  dengan  perintah  "pip  install  streamlit-option-
menu".''')
    st.write('2. Menu yang dibuat ada 2 yaitu Klasifikasi dan Regresi.')
    st.write('3. Inputan nya apa aja, sesuaikan dengan arsitektur code anda
pada notebook.')
    st.write('4.  Referensi  desain  streamlit  dapat  di  akses  pada
https://streamlit.io/.')
    st.write('5. Link streamlit desain ini dapat di akses pada https://apputs-
6qzfvr4ufiyzhj84mrfkt7.streamlit.app/.')
    st.write('''6. Library pada file requirements yang dibutuhkan untuk deploy
online di github ada 5 yaitu streamlit, scikit-learn, pandas, numpy, streamlit-
option-menu.''')
```