PMDPM A

Kelompok Pandas :

1. Diko Ernando Castillo (220711797)
2. Gilar Dylan Mahendra (220711800)
3. Daniel Dwi Ananto (220711801)
4. Beryl Austevann Khobert (220711808)

1) Notebook 1 (Notebook_REGRESI_A_Pandas_RR_VS_SVR_Dylan)

```
import pandas as pd
import numpy as np

UTS_Gasal_2425_csv = pd.read_csv(r'C:\Users\ROG STRIX\Documents\000000000000000
MESIN\Projek UTS PMDPM_A_Pandas\Dataset UTS_Gasal 2425.csv')

df_uts = pd.DataFrame(data = UTS_Gasal_2425_csv, index = None)
df_uts

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

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

df_uts2.info()

df_uts2.describe()

print("data null\n", df_uts2.isnull().sum())
print("\n")
print("data kosong \n", df_uts2.empty)
print("\n")
print("data nan \n", df_uts2.isna().sum())

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 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("\n")
print("data kosong \n", df_uts_clean.empty)
print("\n")
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=97)

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

x_test_enc=transform.fit_transform(x_test_uts)

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.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': [0.01,0.1,1,10,100],
    'feature_selection__k':np.arange(1,20)

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

GSCV_RR.fit(x_train_enc, y_train_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
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.metrics import mean_absolute_error, mean_squared_error

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

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

}
GSCV_SVR = GridSearchCV(pipe_SVR, param_grid_SVR,
cv=5,scoring='neg_mean_squared_error')

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

SVR_predict = GSCV_SVR.predict(x_test_enc)

mse_SVR = mean_squared_error(y_test_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_price_SVR'] = df_results['SVR Prediction'] - df_results['price']
df_results.head()

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()

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['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_RR.best_estimator_

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

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

2) Notebook 2 (Notebook_REGRESI_A_Pandas_LR_VS_RF_Diko)

```
import pandas as pd
import numpy as np

UTS_Gasal_2425_csv = pd.read_csv(r'C:\Users\ROG STRIX\Documents\000000000000000
MESIN\Projek UTS PMDPM_A_Pandas\Dataset UTS_Gasal 2425.csv')

df_uts = pd.DataFrame(data = UTS_Gasal_2425_csv, index = None)
df_uts

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

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

df_uts2.info()

df_uts2.describe()

print("data null\n", df_uts2.isnull().sum())
print("\n")
print("data kosong \n", df_uts2.empty)
print("\n")
print("data nan \n", df_uts2.isna().sum())

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 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("\n")
print("data kosong \n", df_uts_clean.empty)
print("\n")
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=97)

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

x_test_enc=transform.fit_transform(x_test_uts)
```

```
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.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=1000))
    ])

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

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

GSCV_Lasso.fit(x_train_enc, y_train_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)
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.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

pipe_RF = Pipeline(steps=[
    ('scale', StandardScaler()),
    ('feature_selection', SelectKBest(score_func=f_regression)),
    ('reg', RandomForestRegressor(random_state=97))
])

param_grid_RF = {
    'reg__n_estimators': [50, 100, 200],
    'reg__max_depth': [None, 10, 20],
    'feature_selection__k': np.arange(1, 20)
}

GSCV_RF = GridSearchCV(pipe_RF, param_grid_RF, cv=5, scoring='neg_mean_squared_error',
n_jobs=-1)
GSCV_RF.fit(x_train_enc, y_train_uts)

print("\nBest model: {}".format(GSCV_RF.best_estimator_))
print("Random Forest 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("Random Forest Mean Squared Error (MSE): {}".format(mse_RF))
print("Random Forest Mean Absolute Error (MAE): {}".format(mae_RF))
print("Random Forest Root Mean Squared Error (RMSE): {}".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 = pd.DataFrame({'price': y_test_uts})
```

```
df_results['Lasso Prediction'] = Lasso_predict
df_results['Selisih_price_Lasso'] = 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="green",
linewidth=4, linestyle="dashed")

plt.plot(data_len, df_results['RF Prediction'], label="RF 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_Lasso = mean_absolute_error(df_results['price'], df_results['Lasso Prediction'])
mse_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'])
mse_RF = np.sqrt(mean_squared_error(df_results['price'], df_results['RF Prediction']))
rf_feature_count = GSCV_RF.best_params_['feature_selection__k']

print(f"Lasso MAE: {mae_Lasso}, Lasso MSE: {mse_Lasso}, Lasso Feature Count:
{lasso_feature_count}")
print(f"RF MAE: {mae_RF}, RF MSE: {mse_RF}, RF Feature Count: {rf_feature_count}")

import pickle
best_model =  GSCV_Lasso.best_estimator_

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

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

3) Notebook 3 (Notebook_KLASIFIKASI_A_Pandas_RF_VS_LogReg_Beryl)

```
import pandas as pd
import numpy as np

UTS_Gasal_2425_csv = pd.read_csv(r'C:\Users\ROG STRIX\Documents\000000000000000
MESIN\Projek UTS PMDPM_A_Pandas\Dataset UTS_Gasal 2425.csv')

df_uts = pd.DataFrame(data = UTS_Gasal_2425_csv, index = None)
df_uts

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

print(df_uts2['category'].value_counts())

df_uts2.info()

df_uts2.describe()

print("data null\n", df_uts2.isnull().sum())
print("\n")
print("data kosong \n", df_uts2.empty)
print("\n")
print("data nan \n", df_uts2.isna().sum())

from sklearn.model_selection import train_test_split

x = df_uts2.drop('category', axis=1)
y = df_uts2.category

x_train_uts, x_test_uts, y_train_uts, y_test_uts = train_test_split(x, y,
                                                                    test_size=0.25,
                                                                    random_state=97)

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

x_test_enc=transform.fit_transform(x_test_uts)

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 SelectKBest, SelectPercentile
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
import numpy as np

pipe_RF=[('data scaling', StandardScaler()),
         ('feature select', SelectKBest()),
         ('clf',RandomForestClassifier(random_state=97,class_weight='balanced'))]

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

```
                'data scaling': [MinMaxScaler()],
                'feature select': [SelectPercentile()],
                'feature select__percentile':np.arange(20, 50),
                'clf__max_depth': np.arange(4, 5),
                'clf__n_estimators': [100,150]
                }]
estimator_RF = Pipeline(pipe_RF)
SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=97)
GSCV_RF=GridSearchCV(estimator_RF,params_grid_RF,cv=SKF)
GSCV_RF.fit(x_train_enc,y_train_uts)
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_uts)))
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_uts, 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_uts,RF_pred))

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

pipe_logreg = [
   ('data scaling', StandardScaler()),
   ('feature select', SelectKBest()),
   ('clf', LogisticRegression(random_state=97, class_weight='balanced', max_iter=1000))
]

params_grid_logreg = [{
    'data scaling': [StandardScaler()],
    'feature select__k': np.arange(2, 6),
    'clf__C': [0.1, 1.0, 10.0],
    'clf__solver': ['lbfgs', 'liblinear']
},
{
    'data scaling': [StandardScaler()],
    'feature select': [SelectPercentile()],
    'feature select__percentile': np.arange(20, 50),
    'clf__C': [0.1, 1.0, 10.0],
    'clf__solver': ['lbfgs', 'liblinear']
},
{
    'data scaling': [MinMaxScaler()],
    'feature select__k': np.arange(2, 6),
    'clf__C': [0.1, 1.0, 10.0],
    'clf__solver': ['lbfgs', 'liblinear']
},
{
    'data scaling': [MinMaxScaler()],
    'feature select': [SelectPercentile()],
    'feature select__percentile': np.arange(20, 50),
    'clf__C': [0.1, 1.0, 10.0],
    'clf__solver': ['lbfgs', 'liblinear']
}]

estimator_logreg = Pipeline(pipe_logreg)

SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=97)

GSCV_logreg = GridSearchCV(estimator_logreg, params_grid_logreg, cv=SKF)
GSCV_logreg.fit(x_train_enc, y_train_uts)
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_uts)))
print("Best model:", GSCV_logreg.best_estimator_)

mask = GSCV_logreg.best_estimator_.named_steps['feature 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_uts, 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_uts, logreg_pred))

import pickle

with open('BestModel_CLF_RF_Pandas.pkl','wb') as r:
    pickle.dump((GSCV_RF),r)
print("Model RF berhasil disimpan")
```

4) Notebook 4 (Notebook_KLASIFIKASI_A_Pandas_GBR_VS_SVM_Daniel)

```
import pandas as pd
import numpy as np

UTS_Gasal_2425_csv = pd.read_csv(r'C:\Users\ROG STRIX\Documents\000000000000000
MESIN\Projek UTS PMDPM_A_Pandas\Dataset UTS_Gasal 2425.csv')

df_uts = pd.DataFrame(data = UTS_Gasal_2425_csv, index = None)
df_uts

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

print(df_uts2['category'].value_counts())

df_uts2.info()

df_uts2.describe()

print("data null\n", df_uts2.isnull().sum())
print("\n")
print("data kosong \n", df_uts2.empty)
print("\n")
print("data nan \n", df_uts2.isna().sum())

from sklearn.model_selection import train_test_split

x = df_uts2.drop('category', axis=1)
y = df_uts2.category

x_train_uts, x_test_uts, y_train_uts, y_test_uts = train_test_split(x, y,
                                                                    test_size=0.25,
                                                    random_state=97)

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

x_test_enc=transform.fit_transform(x_test_uts)

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.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.feature_selection import SelectKBest, SelectPercentile
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
import numpy as np

pipe_GBT = Pipeline(steps=[
   ('feat_select',SelectKBest()),
   ('clf',GradientBoostingClassifier(random_state=97))])
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_uts)

print("GSCV Finished")

print("CV Score: {}".format(GSCV_GBT.best_score_))
print("Test Score: {}".format(GSCV_GBT.best_estimator_.score(x_test_enc, y_test_uts)))
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_uts, 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_uts,RF_pred))

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]

    },
    {
        '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=97)
```

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

GSCV_SVM.fit(x_train_enc, y_train_uts)
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_uts)))
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_uts, 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_uts, SVM_pred))

import pickle

with open('BestModel_CLF_GBT_Pandas.pkl','wb') as r:
    pickle.dump((GSCV_GBT),r)
print("Model GBT berhasil disimpan")
```

5) **Streamlit**

```
import streamlit as st
from streamlit_option_menu import option_menu

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

if selected == 'Klasifikasi':
    st.title('Klasifikasi')

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

    squaremeters = st.number_input("Masukan Squaremeters", 0)
    jmlruangan = st.number_input("Masukan Jumlah Ruangan", 0)
    yard = st.radio("Has Yard", ["Yes", "No"])
    pool = st.radio("Has Pool", ["Yes", "No"])

    lantai = st.number_input("Masukan Jumlah Lantai", 0)
    citycode = st.number_input("Masukan City Code", 0)
    citypartrange = st.number_input("Masukan City Part Range", 0)
    prevowners = st.number_input("Masukan Jumlah Pemilik Sebelumnya", 0)
    made = st.number_input("Masukan Tahun Dibuat", 0)

    isnewbuild = st.radio("Is New Build?", ["New", "Old"])
    stormprotector = st.radio("Has Storm Protector", ["Yes", "No"])

    basement = st.number_input("Basement", 0)
    attic = st.number_input("Attic", 0)
    garage = st.number_input("Garage", 0)

    storageroom = st.radio("Has Storage Room", ["Yes", "No"])

    guestroom = st.number_input("Guestroom", 0)

    jawaban = st.number_input("Masukkan Harga", min_value=0)

    # Tombol untuk prediksi harga
    harga = st.button("Prediksi Category")

    if harga:
        if jawaban > 6000000:
            st.success(f"Termasuk Kategori Luxury")
        elif jawaban < 3000000:
            st.success(f"Termasuk Kategori Middle")
        else:
            st.error(f"Termasuk Kategori Basic")

if selected == 'Regresi':
    st.title('Regresi')

    file = st.file_uploader("Masukkan File", type=["csv", "txt"])
    squaremeters = st.number_input("Masukan Squaremeters", 0)
    jmlruangan = st.number_input("Masukan Jumlah Ruangan", 0)

    yard = st.radio("Has Yard", ["Yes", "No"])
    pool = st.radio("Has Pool", ["Yes", "No"])

    lantai = st.number_input("Masukan Jumlah Lantai", 0)
    citycode = st.number_input("Masukan City Code", 0)
    citypartrange = st.number_input("Masukan City Part Range", 0)
    prevowners = st.number_input("Masukan Jumlah Pemilik Sebelumnya", 0)
    made = st.number_input("Masukan Tahun Dibuat", 0)

    isnewbuild = st.radio("Is New Build?", ["New", "Old"])
    stormprotector = st.radio("Has Storm Protector", ["Yes", "No"])

    basement = st.number_input("Basement", 0)
    attic = st.number_input("Attic", 0)
    garage = st.number_input("Garage", 0)

    storageroom = st.radio("Has Storage Room", ["Yes", "No"])

    guestroom = st.number_input("Guestroom", 0)

    category = st.selectbox("Category", ["Luxury","Middle","Basic"])

    hitung = st.button("Prediksi Harga")

    if hitung:
        if category == "Luxury":
            harga = 6000000 + jmlruangan * 100000
            st.success(f"Harga untuk kategori Luxury adalah: Rp {harga:,}")
        elif category == "Middle":
            harga = 3000000 + jmlruangan * 75000
            st.success(f"Harga untuk kategori Middle adalah: Rp {harga:,}")
        else:
            harga = 1000000 + jmlruangan * 50000
            st.success(f"Harga untuk kategori Basic adalah: Rp {harga:,}")

if selected == 'Catatan':
    st.title('Catatan')
    st.write('''1. Untuk memunculkan sidebar agar tidak error ketike 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, seusaikan 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.''')
```