Random Forest vs logistic Regression

```
import pandas as pd
import numpy as np
df_dataset=pd.read_csv('Dataset UTS_Gasal 2425.csv')
df_dataset.head(20)


df_dataset2=df_dataset.drop('price', axis=1)
df_dataset2.head(50)


print("data null \n",df_dataset2.isnull().sum())
print("\ndata kosong \n",df_dataset2.empty)
print("\ndata nan \n",df_dataset2.isna().sum())


print("Sebelum drop missing value",df_dataset2.shape)
df_dataset2 = df_dataset2.dropna(how="any",inplace=False)
print("Sesudah drop missing value", df_dataset2.shape)


print("Sebelum Pengecekan data duplikat, ", df_dataset2.shape)
df_dataset3=df_dataset2.drop_duplicates(keep='last')
print("Setelah Pengecekan data duplikat, ", df_dataset3.shape)


from sklearn.model_selection import train_test_split
x = df_dataset3.drop(columns=['category'],axis=1)
y = df_dataset3['category']
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.25,random_state=99
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 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=99,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
GSCV_RF.fit(x_train_enc,y_train)
print("GSCV training finished")

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classification_report
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 pl
cm = confusion_matrix(y_test, RF_pred, labels=GSCV_RF.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=GSCV_RF.classes_)
disp.plot()
plt.title("Random Forest Confusion Matrix")
plt.show()
print("Classification report RF: \n", classification_report(y_test, RF_pred))



import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.feature_selection import SelectKBest, SelectPercentile
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sn
pipe_LR = [
   ('data_scaling', StandardScaler()),
   ('feature_select', SelectKBest()),
   ('clf', LogisticRegression(random_state=99, class_weight='balanced'))
]

params_grid_LR = [{
```

```
    '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_LR = Pipeline(pipe_LR)
SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=92)
GSCV_LR = GridSearchCV(
    estimator_LR,
```

```
    params_grid_LR,
    cv=SKF,
    scoring='accuracy',
    n_jobs=-1,
    verbose=1
)
GSCV_LR.fit(x_train_enc, y_train)
print("GSCV training finished")



print("CV Score: {}".format(GSCV_LR.best_score_))
print("Test Score: {}".format(GSCV_LR.best_estimator_.score(x_test_enc, y_test))
print("Best model:", GSCV_LR.best_estimator_)
mask = GSCV_LR.best_estimator_.named_steps['feature_select'].get_support()
print("Best features:", df_train_enc.columns[mask])
LR_pred = GSCV_LR.predict(x_test_enc
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classification_report
cm = confusion_matrix(y_test, LR_pred, labels=GSCV_LR.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=GSCV_LR.classes_)
disp.plot()
plt.title("Logistic Regression Confusion Matrix")
plt.show()
print("Classification report LR: \n", classification_report(y_test, LR_pred))

import pickle
with open('BestModel_CLF_Random Forest_VS_Logistic Regression_numpy.pkl','wb') as r:
    pickle.dump((GSCV_RF),r)
print("Model Random Forest berhasil disimpan")
```

Gradient Boosting Classifier VS Support Vector Machine

```
import pandas as pd
import numpy as np
df_dataset=pd.read_csv('Dataset UTS_Gasal 2425.csv')
df_dataset.head(20)

df_dataset2=df_dataset.drop('price', axis=1)
df_dataset2.head(50)

df_dataset2['isnewbuilt'].value_counts()

print("data null \n",df_dataset2.isnull().sum())
print("\ndata kosong \n",df_dataset2.empty)
print("\ndata nan \n",df_dataset2.isna().sum())

print("Sebelum drop missing value",df_dataset2.shape)
df_dataset2 = df_dataset2.dropna(how="any",inplace=False)
print("Sesudah drop missing value" ,df_dataset2.shape)

print("Sebelum Pengecekan data duplikat, ", df_dataset2.shape)
df_dataset3=df_dataset2.drop_duplicates(keep='last')
print("Setelah Pengecekan data duplikat, ", df_dataset3.shape)

from sklearn.model_selection import train_test_split
x = df_dataset3.drop(columns=['category'],axis=1)
y = df_dataset3['category']
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.25,random_state=99)
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.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.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=99)
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.feature_selection import SelectKBest, SelectPercentile
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.pipeline import Pipeline
pipe_GBT = Pipeline(steps=[
   ('feat_select', SelectKBest()),
   ('clf', GradientBoostingClassifier(random_state=99))
])
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 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))
```

Lasso Regression dan Random Forest Regressor

```
import pandas as pd

import numpy as np

df_dataset = pd.read_csv(r'C:\KULIAH\Semester 5\Machine Learning\Tugas_UTS\Dataset UTS_Gasal 2425.csv')

df_dataset.head(10)


df_dataset2 = df_dataset.drop(['category'], axis=1)

df_dataset2.head()


df_dataset2.info()


df_dataset2.describe()


print(df_dataset2.columns)


print("data null \n", df_dataset2.isnull().sum())

print("data kosong \n", df_dataset2.empty)

print("data nan \n", df_dataset2.isna().sum())


import matplotlib.pyplot as plt

df_dataset2.floors.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_dataset_clean = remove_outlier(df_dataset2)
print("Jumlah Baris DataFrame sebelum dibuang outlier", df_dataset2.shape[0])
print("Jumlah Baris DataFrame sesudah dibuang outlier", df_dataset_clean.shape[0])
df_dataset_clean.floors.plot(kind='box', vert=True)
plt.gca().invert_yaxis()
plt.show()

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

from sklearn.model_selection import train_test_split

X_regress = df_dataset_clean.drop('floors', axis=1)
y_regress = df_dataset_clean['floors']

X_train_dataset, X_test_dataset, y_train_dataset, y_test_dataset = train_test_split(
  X_regress, y_regress,
  test_size=0.25,
  random_state=85
)
```

```
print(X_train_dataset.shape)
print(X_test_dataset.shape)

import pandas as pd
from sklearn.preprocessing import OneHotEncoder
transform = OneHotEncoder(sparse=False, handle_unknown='ignore')
X_train_enc = transform.fit_transform(X_train_dataset)
X_test_enc = transform.transform(X_test_dataset)
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()
print(df_train_enc.head(10))
print(df_test_enc.head(10))

import numpy as np
import pandas as pd
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.model_selection import train_test_split
df = pd.read_csv('C:/KULIAH/Semester 5/Machine Learning/Tugas_UTS/Dataset UTS_Gasal
2425.csv')
print(df.head())
print(df.dtypes)
df = pd.get_dummies(df, drop_first=True)
X = df.drop('price', axis=1)
y = df['price']
```

```
print(X.dtypes)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
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, min(20, X_train.shape[1]))
}
GSCV_Lasso = GridSearchCV(pipe_Lasso, param_grid_Lasso, cv=5,
scoring='neg_mean_squared_error')
try:
  GSCV_Lasso.fit(X_train, y_train)
except ValueError as e:
  print("ValueError: ", e
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)
mse_Lasso = mean_squared_error(y_test, Lasso_predict)
mae_Lasso = mean_absolute_error(y_test, 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_dataset, columns=['dataset'])
df_results['Lasso Prediction'] = Lasso_predict
df_results['Selisih_dataset_LR'] = df_results['dataset'] - df_results['Lasso Prediction']
df_results.head()

df_results.describe()

from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt
import pandas as pd

print("Columns in DataFrame X:", X.columns.tolist())

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

categorical_features = ['citycode', 'made']

for col in categorical_features:
  if col not in X.columns:
    raise ValueError(f"Column '{col}' not found in DataFrame.")

preprocessor = ColumnTransformer(
  transformers=[
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
```

```
    ],
    remainder='passthrough'
)

x_train_enc = preprocessor.fit_transform(X_train)
x_test_enc = preprocessor.transform(X_test)

rf_model = RandomForestRegressor()

param_grid_RF = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5]
}

GSCV_RF = GridSearchCV(rf_model, param_grid_RF, cv=5, scoring='neg_mean_squared_error')

GSCV_RF.fit(x_train_enc, y_train)

print(f"Best Cross-Validation Score (neg MSE): {GSCV_RF.best_score_:.4f}")

test_score = GSCV_RF.score(x_test_enc, y_test)
print(f"Test Score (R^2): {test_score:.4f}")

print("Best model parameters:", GSCV_RF.best_estimator_)

RF_pred = GSCV_RF.predict(x_test_enc)

mse = mean_squared_error(y_test, RF_pred)
```

```
mae = mean_absolute_error(y_test, RF_pred)
r2 = r2_score(y_test, RF_pred)

print(f"Mean Squared Error: {mse:.4f}")
print(f"Mean Absolute Error: {mae:.4f}")
print(f"R^2 Score: {r2:.4f}")

plt.figure(figsize=(10, 6))
plt.scatter(y_test, RF_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], '--r')
plt.title("Predicted vs Actual Prices")
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.grid()
plt.savefig("predicted_vs_actual_prices_rf.png")
plt.show()

df_results.describe()

import pandas as pd

y_test_dataset = [100, 200, 300, 400, 500]
Lasso_predict = [90, 210, 290, 390, 480]
RandomForest_predict = [95, 205, 295, 395, 490]

def validate_lengths(y_test, lasso_pred, rf_pred):
   if len(y_test) != len(lasso_pred):
      raise ValueError(f"Length of y_test_dataset ({len(y_test)}) does not match length of
Lasso_predict ({len(lasso_pred)})")
```

```
    if len(y_test) != len(rf_pred):
        raise ValueError(f"Length of y_test_dataset ({len(y_test)}) does not match length of RandomForest_predict ({len(rf_pred)})")

validate_lengths(y_test_dataset, Lasso_predict, RandomForest_predict)

df_results = pd.DataFrame({'dataset': y_test_dataset})

df_results['Lasso Prediction'] = Lasso_predict

df_results['Selisih_dataset_LR'] = df_results['dataset'] - df_results['Lasso Prediction']

df_results['Random Forest Prediction'] = RandomForest_predict

df_results['Selisih_dataset_RFR'] = df_results['dataset'] - df_results['Random Forest Prediction']

print(df_results.head())

print(f"Jumlah Baris DataFrame sebelum dibuang outlier: {len(df_results)}")
print(f"Jumlah Baris DataFrame sesudah dibuang outlier: {len(df_results)}")

print(df_results.info())

print("Data null:")
print(df_results.isnull().sum())

print("Data kosong:")
print((df_results == 0).sum())
```

```
print("Data NaN:")
print(df_results.isna().sum())

df_results.describe()

import matplotlib.pyplot as plt

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

data_len = range(len(df_results['dataset']))

plt.scatter(data_len, df_results['dataset'], 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['Random Forest Prediction'], label='Random Forest 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['dataset'], df_results['Lasso Prediction'])
rmse_lasso = np.sqrt(mean_squared_error(df_results['dataset'], df_results['Lasso Prediction']))
lasso_feature_count = GSCV_Lasso.best_params_['feature_selection__k']
```

```
mae_rfr = mean_absolute_error(df_results['dataset'], df_results['Random Forest Prediction'])

rmse_rfr = np.sqrt(mean_squared_error(df_results['dataset'], df_results['Random Forest Prediction']))

rfr_feature_count = len(df_results.columns) - 1


print(f"Lasso MAE: {mae_lasso:.4f}, Lasso RMSE: {rmse_lasso:.4f}, Lasso Feature Count: {lasso_feature_count}")

print(f"Random Forest Regressor MAE: {mae_rfr:.4f}, RFR RMSE: {rmse_rfr:.4f}, RFR Feature Count: {rfr_feature_count}")
```

Ridge Regression VS Support Vector Regressor

```
import pandas as pd

import numpy as np


df_dataset = pd.read_csv(r'C:\Users\yudhika wira utama\Downloads\Prject UTS Mesin\Dataset UTS_Gasal 2425.csv')

df_dataset.head(10)

df_dataset2 = df_dataset.drop(['category'], axis=1)

df_dataset2.head()

df_dataset2.info()

df_dataset2.describe()

print(df_dataset2['price'].value_counts())

print("data null \n",df_dataset2.isnull().sum())

print("data kosong \n",df_dataset2.empty)

print("data nan \n", df_dataset2.isna().sum())


import matplotlib.pyplot as plt
```

```
df_dataset2.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_dataset = remove_outlier(df_dataset2)
print("jumlah baris DataFrame sebelum dibuang outlier", df_dataset2.shape[0])
print("jumlah baris DataFrame sesudah dibuang outlier", df_dataset.shape[0])
df_dataset.price.plot(kind= 'box', vert=True)

plt.gca().invert_yaxis()
plt.show()
print("data null \n", df_dataset2.isnull().sum())
print("data kosong \n", df_dataset.empty)
print("data nan \n", df_dataset2.isna().sum())
from sklearn.model_selection import train_test_split
```

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

X_train_dataset, X_test_dataset, y_train_dataset, y_test_dataset = train_test_split(X_regress, y_regress, test_size=0.25, random_state= 99)
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'
)
import numpy as np
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.metrics import mean_absolute_error, mean_squared_error

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

le = LabelEncoder()
for col in categorical_columns:
   X_train_dataset[col] = le.fit_transform(X_train_dataset[col])
   X_test_dataset[col] = le.transform(X_test_dataset[col])

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_dataset, y_train_dataset)

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_dataset)

mse_Lasso = mean_squared_error(y_test_dataset, Lasso_predict)
mae_Lasso = mean_absolute_error(y_test_dataset, 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)))
```

```
import pandas as pd

df_results = pd.DataFrame(y_test_dataset, columns=['price'])

df_results['Ridge Prediction'] = Lasso_predict

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

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_dataset, y_train_dataset)

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_dataset)

mse_SVR = mean_squared_error(y_test_dataset, SVR_predict)
mae_SVR = mean_absolute_error(y_test_dataset, 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_dataset)
df_results['SVR Prediction'] = SVR_predict

df_results['Selisih_dataset_SVR'] = df_results['SVR Prediction'] - df_results['price']
df_results.head()
df_results.describe()
import pickle

best_model = GSCV_SVR.best_estimator_
```

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

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

Streamlit Python

```
import streamlit as st
import pickle
import os

from streamlit_option_menu import option_menu

model_path = r'C:\Users\LENOVO\Documents\Semester 5\Pembelajaran mesin dan mendalam\Projek UTS Gasal 20242025-20241020\Project UTS'
model=os.path.join(model_path,'BestModel_CLF_Random Forest_numpy')
model=os.path.join(model_path,'BestModel_REG_Support Vector Regressor_numpy')

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')

    st.write('Untuk Inputan File dataset (csv) bisa menggunakan st.file_uploader')
    file = st.file_uploader("Masukkan File", type=["csv", "txt"])
```

```
    st.write('Untuk usia bisa menggunakan st.slider')
    Age = st.slider("Age", 0, 100)
    st.write('Untuk jenis kelamin bisa menggunakan st.radio')
    Sex = st.radio("Gender", ["Female", "Male"])
    st.write('Untuk beberapa kolom bisa menggunakan st.selectbox')
    nama_kolom = st.selectbox("Nama Kolom", ["Under", "Normal", "Over"])

    st.write('Untuk inputan manual bisa menggunakan st.number_input')
    panjang = st.number_input("Masukan Input", 0)
    lebar = st.number_input("Masukan Nilai Lebar", 0)

    jawaban = st.number_input("Masukkan Jawaban Anda", min_value=0)
    st.write('Tombol button(Menggunakan st.button)')
    hitung = st.button("Prediksi")

    if hitung:
        luas_benar = panjang * lebar
        st.write(f"Panjang: {panjang}, Lebar: {lebar}")

        if jawaban == luas_benar:
            st.success(f"Benar! Luas Persegi Panjang adalah {luas_benar}.")
        else:
            st.error(f"Salah! Luas Persegi Panjang yang benar adalah {luas_benar}.")

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"])
```

```
    st.write('Untuk usia bisa menggunakan st.slider')
    Age = st.slider("Age", 0, 100)
    st.write('Untuk jenis kelamin bisa menggunakan st.radio')
    Sex = st.radio("Gender", ["Female", "Male"])
    st.write('Untuk beberapa kolom bisa menggunakan st.selectbox')
    nama_kolom = st.selectbox("Nama Kolom", ["Under", "Normal", "Over"])

    st.write('Untuk inputan manual bisa menggunakan st.number_input')
    panjang = st.number_input("Masukan Input", 0)
    lebar = st.number_input("Masukan Nilai Lebar", 0)
    alas = st.slider("Masukkan Nilai Alas", 0, 100)
    tinggi = st.slider("Masukkan Nilai Tinggi", 0, 100)
    st.write('Tombol button(Menggunakan st.button)')
    hitung = st.button("Prediksi")

    if hitung:
        luas = 0.5 * alas * tinggi
        st.write("Luas Segitiga Adalah", luas)

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 saja, 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-6qxfvt4ufiyzhj84mrfkt7.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.''')