Kode Notebook Random Forest vs Logistic Regression

Raditya Alifka

```
import pandas as pd
import numpy as np
df_classification = pd.read_csv('Dataset UTS_Gasal 2425.csv')
print(df_classification)


df_classification.info()


df_classification.describe()


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


df_classification2 = df_classification.drop('price', axis = 1)
df_classification2.head()



from sklearn.model_selection import train_test_split
x = df_classification2.drop(columns=['category'], axis=1)
y = y =df_classification2['category']
x_train, x_test, y_train, y_test = train_test_split(x,y,
test_size=0.25, random_state=93)


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(drop='first'), 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(20)

df_train_enc.head(20)


from sklearn.preprocessing import MinMaxScaler, StandardScaler

from sklearn.feature_selection import SelectPercentile,
SelectKBest

from sklearn.model_selection import StratifiedKFold,
GridSearchCV

from sklearn.ensemble import RandomForestClassifier

from sklearn.pipeline import Pipeline

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=93,
class_weight='balanced'))]


params_grid_RF = [{
                'data scaling' : [StandardScaler()],
                'feature select__k' : np.arange(2,17),
                '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(2,17),
                '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=93)

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 feature: ", 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("Classificaation report RF: \n",
classification_report(y_test, RF_pred))
```

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

from sklearn.feature_selection import SelectPercentile,
SelectKBest

from sklearn.model_selection import StratifiedKFold,
GridSearchCV

from sklearn.linear_model import LogisticRegression

from sklearn.pipeline import Pipeline

from sklearn.metrics import classification_report,
confusion_matrix, ConfusionMatrixDisplay

import numpy as np


pipe_LR = [

        ('data scaling', StandardScaler()),

        ('feature select', SelectKBest()),

        ('clf', LogisticRegression(random_state=93,
class_weight='balanced', solver='liblinear'))]


params_grid_LR = [{

                'data scaling': [StandardScaler()],

                'feature select__k': np.arange(2,17),

                'clf__C': [0.1, 1, 10],

                'clf__max_iter': [100, 200]

            },

            {

                'data scaling': [StandardScaler()],

                'feature select': [SelectPercentile()],

                'feature select__percentile': np.arange(20,
50),

                'clf__C': [0.1, 1, 10],

                'clf__max_iter': [100, 200]

            },

            {
```

```
                'data scaling': [MinMaxScaler()],
                'feature select__k': np.arange(2,17),
                'clf__C': [0.1, 1, 10],
                'clf__max_iter': [100, 200]
            },
            {
                'data scaling': [MinMaxScaler()],
                'feature select': [SelectPercentile()],
                'feature select__percentile': np.arange(20,
50),
                'clf__C': [0.1, 1, 10],
                'clf__max_iter': [100, 200]}]

estimator_LR = Pipeline(pipe_LR)
GSCV_LR = GridSearchCV(estimator_LR, params_grid_LR, cv=SKF)

GSCV_LR.fit(x_train_enc, y_train)
print("GSCV Training Finished for Logistic Regression")

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 feature: ", df_train_enc.columns[mask])

LR_pred = GSCV_LR.predict(x_test_enc)

import matplotlib.pyplot as plt
```

```
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('RF_Classification_model.pkl','wb') as r:

    pickle.dump((GSCV_RF),r)


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

Kode Notebook SVM vs GBT

Gabriel Mario Binsar

```
import pandas as pd

import numpy as np


df_klasifikasi=pd.read_csv(r'C:\Users\Lenovo\Documents\ML\uts\
Dataset UTS_Gasal 2425.csv')

df_klasifikasi.head(20)

df_klasifikasi2=df_klasifikasi.drop('price', axis=1)

df_klasifikasi.head(50)
```

```
df_klasifikasi2['category'].value_counts()

print("data null \n",df_klasifikasi2.isnull().sum())

print("\ndata kosong \n",df_klasifikasi2.empty)

print("\ndata nan \n",df_klasifikasi2.isna().sum())

print("Sebelum drop missing value ",df_klasifikasi2.shape)

df_klasifikasi2 =
df_klasifikasi2.dropna(how="any",inplace=False)

print("Sesudah drop missing value ",df_klasifikasi2.shape)

print("Sebelum pengecekan data duplikat,
",df_klasifikasi2.shape)

df_klasifikasi3 = df_klasifikasi2.drop_duplicates(keep =
'last')

print("Setelah Pemgecekan data duplikat,
",df_klasifikasi3.shape)

from sklearn.model_selection import train_test_split

x = df_klasifikasi3.drop(columns=['category'],axis=1)

y= y=df_klasifikasi3['category']


x_train, x_test,y_train,y_test=train_test_split(x,y,test_size
= 0.25, random_state=93)


print(x_train.shape)

print(x_test.shape)

df_klasifikasi3.info()

from sklearn.preprocessing import OneHotEncoder

from sklearn.compose import make_column_transformer


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



transform = make_column_transformer(
```

```
    (OneHotEncoder(drop='first'),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_ou
t())


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
import numpy as np


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=15)


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_suppor
t()

print("Best Features: ", df_test_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=15))
])

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_spl
its=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_s
upport()
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('klasifikasi_model1.pkl', 'wb') as r:
    pickle.dump(GSCV_GBT, r)


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

Kode Notebook Ridge vs Support Vector Regressor

Mukti Laksono

```
import pandas as pd
import numpy as np


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


df_price2 = df_price.drop(['category'], axis=1)
df_price2.head()


df_price2.info()


df_price2.describe()


print(df_price2['price'].value_counts())


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

```
import matplotlib.pyplot as plt
df_price2.price.plot(kind='box')
plt.gca().invert_yaxis()
plt.show()


from sklearn.preprocessing import LabelEncoder


label_encoder = LabelEncoder()


df_price2['hasyard'] =
label_encoder.fit_transform(df_price2['hasyard'])
df_price2['haspool'] =
label_encoder.fit_transform(df_price2['haspool'])
df_price2['isnewbuilt'] =
label_encoder.fit_transform(df_price2['isnewbuilt'])
df_price2['hasstormprotector'] =
label_encoder.fit_transform(df_price2['hasstormprotector'])
df_price2['hasstorageroom'] =
label_encoder.fit_transform(df_price2['hasstorageroom'])


import pandas as pd
from sklearn.model_selection import train_test_split


X_regress = df_price2.drop('price', axis=1)
y_regress = df_price2.price


X_train_price, X_test_price, y_train_price, y_test_price =
train_test_split(X_regress, y_regress, test_size=0.3,
random_state=93)


print("Jumlah data train:", len(X_train_price))
print("Jumlah data test:", len(X_test_price))
```

```
print(df_price2.columns)


print(X_regress.head())
print(df_price2.head())


from sklearn.model_selection import train_test_split,
GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest,
f_regression
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_absolute_error,
mean_squared_error


num_features = df_price2.shape[1]


k = min(16, num_features)


selector = SelectKBest(score_func=f_regression, k=k)
X_new = selector.fit_transform(X_regress, y_regress)


pipe_Ridge = Pipeline(steps=[
    ('scale', StandardScaler()),
    ('feature_selection',
SelectKBest(score_func=f_regression)),
    ('reg', Ridge())
])


param_grid_Ridge = {
    'feature_selection__k': np.arange(1, k+1),
```

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

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

GSCV_RR.fit(X_train_price, y_train_price)

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'].intercep
t_))

Ridge_predict = GSCV_RR.predict(X_test_price)

mse_Ridge = mean_squared_error(y_test_price, Ridge_predict)
mae_Ridge = mean_absolute_error(y_test_price, 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_price, columns=['price'])
df_results = pd.DataFrame(y_test_price)
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

import numpy as np


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, k+1)

}


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

```
GSCV_SVR.fit(X_train_price, y_train_price)


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'].interce
pt_))


SVR_predict = GSCV_SVR.predict(X_test_price)


mse_SVR = mean_squared_error(y_test_price, SVR_predict)
mae_SVR = mean_absolute_error(y_test_price, 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_price)
df_results["SVR Prediction"] = SVR_predict


df_results['Selisih_price_SVR'] = df_results["SVR Prediction"]
- df_results['price']
df_results.head()


df_results.describe()


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

```
df_results['Rigde Prediction'] = Ridge_predict

df_results['Selisih_price_RR'] = df_results['Rigde
Prediction'] - df_results['price']


df_results["SVR Prediction"] = SVR_predict

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


df_results.head()


df_results.describe()

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


data_len = range(len(y_test_price))


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


plt.plot(data_len, df_results["Rigde 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


print(df_results.columns)


from sklearn.metrics import mean_absolute_error,
mean_squared_error

import numpy as np
```

```
mae_ridge = mean_absolute_error(df_results['price'],
df_results['Rigde Prediction'])

rmse_ridge = np.sqrt(mean_squared_error(df_results['price'],
df_results['Rigde 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"RidgeMAE: {mae_ridge}, Ridge RMSE: {rmse_ridge}, Ridge
Feature Count: {ridge_feature_count}")

print(f"SVRMAE: {mae_SVR}, SVR RMSE: {rmse_SVR}, SVR Feature
Count: {SVR_feature_count}")


X_regress.head()


import pickle


best_model = GSCV_RR.best_estimator_


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

    pickle.dump(best_model, f)


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

Kode Notebook Lasso vs Random Forest Regressor

Eirine Mamesah

```
import pandas as pd

import numpy as np

df_price =
pd.read_csv(r'C:\Users\ASUS\Downloads\BestModel_Lasso_RF_CatBo
ost\Dataset

UTS_Gasal 2425.csv')

df_price.head(10)

df_price2 = df_price.drop(['squaremeters'], axis=1)

df_price2.head()

df_price2.info()

df_price2.describe()

print(df_price2['price'].value_counts())

print("data null \n", df_price2.isnull().sum())

print("data kosong \n", df_price2.empty)

print("data nan\n", df_price2.isna().sum())

import matplotlib.pyplot as plt

df_price2.price.plot(kind='box')

plt.gca().invert_yaxis()

plt.show()

from pandas.api.types import is_numeric_dtype

def remove_outlier(df_in):

df_out = df_in.copy()

for col_name in 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_out.loc[(df_out[col_name] >= batas_bawah) &
(df_out[col_name]

<= batas_atas)]
```

```
return df_out

df_price_clean = remove_outlier(df_price2)

print("Jumlah baris DataFrame sebelum dibuang outlier",
df_price2.shape[0])

print("Jumlah baris DataFrame setelah dibuang outlier",
df_price_clean.shape[0])

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

plt.gca().invert_yaxis()

plt.show()

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

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

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

from sklearn.model_selection import train_test_split

X_regress = df_price_clean.drop('price', axis=1)

y_regress = df_price_clean.price

X_train_price, X_test_price, y_train_price, y_test_price

= train_test_split(X_regress, y_regress, test_size=0.25,
random_state=93)

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

import numpy as np

import pandas as pd

X_train_price = pd.get_dummies(X_train_price, drop_first=True)

X_test_price = pd.get_dummies(X_test_price, drop_first=True)

X_test_price =
X_test_price.reindex(columns=X_train_price.columns,
fill_value=0)
```

```
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_price, y_train_price)
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'].inter
cept_))
Lasso_predict = GSCV_Lasso.predict(X_test_price)
mse_Lasso = mean_squared_error(y_test_price, Lasso_predict)
mae_Lasso = mean_absolute_error(y_test_price, 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({'price': y_test_price})
df_results['Lasso Prediction'] = Lasso_predict
df_results['Selisih_price_LR'] = df_results['Lasso
Prediction'] - df_results['price']
```

```
print(df_results.head())

df_results.describe()

from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import GridSearchCV

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean_absolute_error,
mean_squared_error

import numpy as np

import pandas as pd

X_train_price = pd.get_dummies(X_train_price, drop_first=True)

X_test_price = pd.get_dummies(X_test_price, drop_first=True)

X_test_price =
X_test_price.reindex(columns=X_train_price.columns,
fill_value=0)

pipe_RF = Pipeline(steps=[

('scale', StandardScaler()),

('reg', RandomForestRegressor())

])

param_grid_RF = {

'reg__n_estimators': [100, 200, 300],

'reg__max_features': ['auto', 'sqrt'],

'reg__max_depth': [None, 10, 20, 30],

'reg__min_samples_split': [2, 5, 10],

}

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

GSCV_RF.fit(X_train_price, y_train_price)

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

print("Random Forest best parameters:
{}".format(GSCV_RF.best_params_))

RF_predict = GSCV_RF.predict(X_test_price)

mse_RF = mean_squared_error(y_test_price, RF_predict)
```

```
mae_RF = mean_absolute_error(y_test_price, 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:
{}".format(np.sqrt(mse_RF)))

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

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_price})

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_price))

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=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'])

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']))

print(f"Lasso MAE: {mae_Lasso}, Lasso RMSE: {rmse_Lasso},
Lasso Feature Count:

{lasso_feature_count}")

print(f"Random Forest MAE: {mae_RF}, Random Forest RMSE:
{rmse_RF}")

X_regress.head()

import pickle

best_model = GSCV_Lasso.best_estimator_

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

pickle.dump(best_model, f)

print("Model terbaik berhasil disimpan

ke 'Notebook_REGRESI_A_CatBoost_Lasso_VS_RFR_EIRINE.pkl'")
```