Klasifikasi model Random Forest VS Logistic Regression

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler, StandardScaler, LabelEncoder
from sklearn.compose import make_column_transformer
from sklearn.feature_selection import SelectKBest, SelectPercentile, SelectFromModel, RFE
from sklearn.model_selection import GridSearchCV, train_test_split, StratifiedKFold, KFold
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.linear_model import LogisticRegression, Ridge, Lasso, LinearRegression
from collections import Counter
from sklearn.metrics import classification_report, confusion_matrix, mean_squared_error, mean_absolute_error, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
```

[3] ✓ 0.8s Python

```
uts = pd.read_csv(r'D:\PMDPM\Projek UTS Gasal 20242025-20241023\Dataset UTS_Gasal 2425.csv')
uts.head(10000)
```

[4] ✓ 0.0s Python

```
print("=" * 50)
print("Informasi DataFrame:")
print("=" * 50)
uts.info()
print("\n")

print("=" * 50)
print("Jumlah Nilai Kosong per Kolom:")
print("=" * 50)
print(uts.isnull().sum())
print("\n")

print("=" * 50)
print("Jumlah Total Baris Duplikat:")
print("=" * 50)
print(uts.duplicated().sum())
print("\n")

if uts.duplicated().sum() > 0:
    print("=" * 50)
    print("Detail Baris Duplikat:")
    print("=" * 50)
    print(uts[uts.duplicated()])
else:
    print("=" * 50)
    print("Tidak Ditemukan Baris Duplikat dalam DataFrame.")
    print("=" * 50)
```

[5] ✓ 0.0s Python

```
pd.set_option('display.float_format', lambda x: '%.5f' % x)

uts.describe()
```

[6] ✓ 0.1s Python

	squaremeters	numberofrooms	floors	citycode	citypartrange	numprevowners	made	basement	attic	garage	hasguestroom	price
count	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000
mean	49870.13120	50.35840	50.27630	50225.48610	5.51010	5.52170	2005.48850	5033.10390	5028.01060	553.12120	4.99460	4993447.52575
std	28774.37535	28.81670	28.88917	29006.67580	2.87202	2.85667	9.30809	2876.72954	2894.33221	262.05017	3.17641	2877424.10995
min	89.00000	1.00000	1.00000	3.00000	1.00000	1.00000	1990.00000	0.00000	1.00000	100.00000	0.00000	10313.50000
25%	25098.50000	25.00000	25.00000	24693.75000	3.00000	3.00000	1997.00000	2559.75000	2512.00000	327.75000	2.00000	2516401.95000
50%	50105.50000	50.00000	50.00000	50693.00000	5.00000	5.00000	2005.50000	5092.50000	5045.00000	554.00000	5.00000	5016180.30000
75%	74609.75000	75.00000	76.00000	75683.25000	8.00000	8.00000	2014.00000	7511.25000	7540.50000	777.25000	8.00000	7469092.45000
max	99999.00000	100.00000	100.00000	99953.00000	10.00000	10.00000	2021.00000	10000.00000	10000.00000	1000.00000	10.00000	10006771.20000

```
df_properti = uts.copy()
df_properti.head()
df_properti.columns
```

[7] ✓ 0.0s Python

```
Index(['squaremeters', 'numberofrooms', 'hasyard', 'haspool', 'floors',
       'citycode', 'citypartrange', 'numprevowners', 'made', 'isnewbuilt',
       'hasstormprotector', 'basement', 'attic', 'garage', 'hasstorageroom',
       'hasguestroom', 'price', 'category'],
      dtype='object')
```

```python
X = df_properti.drop(columns=['price', 'category'], axis=1)
y = df_properti['category']

X_train_bf, X_test, y_train_bf, y_test = train_test_split(X, y, test_size=0.30, random_state=95)
print(f"Shape of X_train: {X_train_bf.shape}")
print(f"Shape of X_test: {X_test.shape}")
```

[8] ✓ 0.0s Python

```
Shape of X_train: (7000, 16)
Shape of X_test: (3000, 16)
```

```python
print(X.columns)
```

[9] ✓ 0.0s Python

```
Index(['squaremeters', 'numberofrooms', 'hasyard', 'haspool', 'floors',
       'citycode', 'citypartrange', 'numprevowners', 'made', 'isnewbuilt',
       'hasstormprotector', 'basement', 'attic', 'garage', 'hasstorageroom',
       'hasguestroom'],
      dtype='object')
```

```python
cat_cols=['hasyard', 'haspool', 'isnewbuilt',
          'hasstormprotector', 'hasstorageroom']

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

X_train_enc = transformer.fit_transform(X_train_bf)
X_test_enc = transformer.transform(X_test)

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())
np.set_printoptions(formatter={'float': '{:.2f}'.format})

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

[10] ✓ 0.0s Python

```python
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=95)

X_folds = []
y_folds = []

for train_index, test_index in skf.split(X_train_enc, y_train_bf):
    X_folds.append((X_train_enc[train_index], X_train_enc[test_index]))
    y_folds.append((y_train_bf.iloc[train_index], y_train_bf.iloc[test_index]))
```

[11] ✓ 0.0s Python

Random Forest

```python
pipe_RF = Pipeline(steps=[
    ('scale', MinMaxScaler()),
    ('feat_select', SelectKBest()),
    ('clf', RandomForestClassifier(random_state=95))
])

param_grid_RF = {
    'feat_select__k': np.arange(2, 5),
    'clf__n_estimators': [100, 150],
    'clf__max_depth': [10, 20]
}

GSCV_RF = GridSearchCV(pipe_RF, param_grid_RF, cv=StratifiedKFold(n_splits=5))

GSCV_RF.fit(X_train_enc, y_train_bf)

mask = GSCV_RF.best_estimator_.named_steps['feat_select'].get_support()
selected_features = df_train_enc.columns[mask]

print("Best model: {}".format(GSCV_RF.best_estimator_))
print("Selected features: {}".format(selected_features))
print("Best CV score: {:.2f}".format(GSCV_RF.best_score_))
print("Test set score: {:.2f}".format(GSCV_RF.score(X_test_enc, y_test)))
```

[12] ✓ 1m 25.3s Python

```
Best model: Pipeline(steps=[('scale', MinMaxScaler()), ('feat_select', SelectKBest(k=4)),
                ('clf', RandomForestClassifier(max_depth=20, random_state=95))])
Selected features: Index(['onehotencoder__hasyard_yes', 'onehotencoder__haspool_no',
       'onehotencoder__haspool_yes', 'remainder__squaremeters'],
      dtype='object')
Best CV score: 0.94
Test set score: 0.94
```

```python
from sklearn.feature_selection import SelectPercentile

pipe_RF = Pipeline(steps=[
    ('scale', MinMaxScaler()),
    ('feat_select', SelectPercentile()),
    ('clf', RandomForestClassifier(random_state=95))
])

param_grid_RF = {
    'feat_select__percentile': [10, 20, 30],
    'clf__n_estimators': [100, 150],
    'clf__max_depth': [10, 20]
}

GSCV_RF2 = GridSearchCV(pipe_RF, param_grid_RF, cv=StratifiedKFold(n_splits=5))
GSCV_RF2.fit(X_train_enc, y_train_bf)

mask = GSCV_RF2.best_estimator_.named_steps['feat_select'].get_support()
selected_features = df_train_enc.columns[mask]

print("Best model: {}".format(GSCV_RF2.best_estimator_))
print("Selected features: {}".format(selected_features))
print("Best CV score: {:.2f}".format(GSCV_RF2.best_score_))
print("Test set score: {:.2f}".format(GSCV_RF2.score(X_test_enc, y_test)))
```

[13] ✓ 1m 17.5s Python

```
Best model: Pipeline(steps=[('scale', MinMaxScaler()),
                ('feat_select', SelectPercentile(percentile=30)),
                ('clf', RandomForestClassifier(max_depth=10, random_state=95))])
Selected features: Index(['onehotencoder__hasyard_no', 'onehotencoder__hasyard_yes',
       'onehotencoder__haspool_no', 'onehotencoder__haspool_yes',
       'onehotencoder__isnewbuilt_new', 'remainder__squaremeters'],
      dtype='object')
Best CV score: 1.00
Test set score: 1.00
```

Logistic Regression

```python
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

pipe_LogReg = Pipeline(steps=[
    ('scale', StandardScaler()),
    ('feat_select', SelectKBest()),
    ('clf', LogisticRegression(solver='liblinear', max_iter=10000))
])

param_grid_LogReg = {
    'feat_select__k': np.arange(2, 5),
    'clf__C': [0.001, 0.01, 0.1, 1, 10],
    'clf__penalty': ['l1', 'l2']
}

GSCV_LogReg = GridSearchCV(pipe_LogReg, param_grid_LogReg, cv=StratifiedKFold(n_splits=5))
GSCV_LogReg.fit(X_train_enc, y_train_bf)

mask = GSCV_LogReg.best_estimator_.named_steps['feat_select'].get_support()
selected_features = df_train_enc.columns[mask]

print("Best model: {}".format(GSCV_LogReg.best_estimator_))
print("Selected features: {}".format(selected_features))
print("Best CV score: {:.2f}".format(GSCV_LogReg.best_score_))
print("Test set score: {:.2f}".format(GSCV_LogReg.score(X_test_enc, y_test)))
```

[14] ✓ 17.2s Python

```
Best model: Pipeline(steps=[('scale', StandardScaler()), ('feat_select', SelectKBest(k=4)),
                ('clf',
                 LogisticRegression(C=10, max_iter=10000, penalty='l1',
                                    solver='liblinear'))])
Selected features: Index(['onehotencoder__hasyard_yes', 'onehotencoder__haspool_no',
       'onehotencoder__haspool_yes', 'remainder__squaremeters'],
```

```python
pipe_LogReg = Pipeline(steps=[
    ('scale', MinMaxScaler()),
    ('feat_select', SelectPercentile()),
    ('clf', LogisticRegression(solver='liblinear', max_iter=10000))
])

param_grid_LogReg = {
    'feat_select__percentile': [10, 20, 30],
    'clf__C': [0.001, 0.01, 0.1, 1, 10],
    'clf__penalty': ['l1', 'l2']
}

GSCV_LogReg2 = GridSearchCV(pipe_LogReg, param_grid_LogReg, cv=StratifiedKFold(n_splits=5))
GSCV_LogReg2.fit(X_train_enc, y_train_bf)

mask = GSCV_LogReg2.best_estimator_.named_steps['feat_select'].get_support()
selected_features = df_train_enc.columns[mask]

print("Best model: {}".format(GSCV_LogReg2.best_estimator_))
print("Selected features: {}".format(selected_features))
print("Best CV score: {:.2f}".format(GSCV_LogReg2.best_score_))
print("Test set score: {:.2f}".format(GSCV_LogReg2.score(X_test_enc, y_test)))
```

[15] ✓ 58.2s Python

```
Best model: Pipeline(steps=[('scale', MinMaxScaler()),
                ('feat_select', SelectPercentile(percentile=30)),
                ('clf',
                 LogisticRegression(C=10, max_iter=10000, penalty='l1',
                                    solver='liblinear'))])
Selected features: Index(['onehotencoder__hasyard_no', 'onehotencoder__hasyard_yes',
       'onehotencoder__haspool_no', 'onehotencoder__haspool_yes',
       'onehotencoder__isnewbuilt_new', 'remainder__squaremeters'],
      dtype='object')
Best CV score: 0.87
Test set score: 0.88
```

Evaluasi

```python
y_pred_RF = GSCV_RF.predict(X_test_enc)

print("Classification Report RandomForest:")
print(classification_report(y_test, y_pred_RF))

cm_RF = confusion_matrix(y_test, y_pred_RF)
ConfusionMatrixDisplay(cm_RF, display_labels=GSCV_RF.classes_).plot(cmap='Blues')
plt.title("Confusion Matrix RandomForest")
plt.show()
```

[16] ✓ 0.4s Python

```
Classification Report RandomForest:
              precision    recall  f1-score   support

       Basic       0.93      0.93      0.93      1304
      Luxury       1.00      1.00      1.00       921
      Middle       0.89      0.88      0.89       775

    accuracy                           0.94      3000
   macro avg       0.94      0.94      0.94      3000
weighted avg       0.94      0.94      0.94      3000
```

```python
y_pred_RF2 = GSCV_RF2.predict(X_test_enc)

print("Classification Report RandomForest SelectPercentile:")
print(classification_report(y_test, y_pred_RF2))

cm_RF2 = confusion_matrix(y_test, y_pred_RF2)
ConfusionMatrixDisplay(cm_RF2, display_labels=GSCV_RF2.classes_).plot(cmap='Blues')
plt.title("Confusion Matrix RandomForest SelectPercentile")
plt.show()
```

[17] ✓ 0.4s Python

```
Classification Report RandomForest SelectPercentile:
              precision    recall  f1-score   support

       Basic       1.00      1.00      1.00      1304
      Luxury       1.00      1.00      1.00       921
      Middle       1.00      1.00      1.00       775

    accuracy                           1.00      3000
   macro avg       1.00      1.00      1.00      3000
weighted avg       1.00      1.00      1.00      3000
```

```python
y_pred_LogReg = GSCV_LogReg.predict(X_test_enc)

print("Classification Report LogisticRegression SelectKBest:")
print(classification_report(y_test, y_pred_LogReg))

cm_LogReg = confusion_matrix(y_test, y_pred_LogReg)
ConfusionMatrixDisplay(cm_LogReg, display_labels=GSCV_LogReg.classes_).plot(cmap='Blues')
plt.title("Confusion Matrix LogisticRegression SelectKBest")
plt.show()
```

[18] ✓ 0.7s Python

```
Classification Report LogisticRegression SelectKBest:
              precision    recall  f1-score   support

       Basic       0.80      0.95      0.87      1304
      Luxury       0.99      1.00      1.00       921
      Middle       0.88      0.59      0.70       775

    accuracy                           0.87      3000
   macro avg       0.89      0.85      0.86      3000
weighted avg       0.88      0.87      0.87      3000
```

```python
y_pred_LogReg2 = GSCV_LogReg2.predict(X_test_enc)

print("Classification Report LogisticRegression SelectPercentile:")
print(classification_report(y_test, y_pred_LogReg2))

cm_LogReg2 = confusion_matrix(y_test, y_pred_LogReg2)
ConfusionMatrixDisplay(cm_LogReg2, display_labels=GSCV_LogReg2.classes_).plot(cmap='Blues')
plt.title("Confusion Matrix LogisticRegression SelectPercentile")
plt.show()
```

[19] ✓ 0.5s Python

```
Classification Report LogisticRegression SelectPercentile:
              precision    recall  f1-score   support

       Basic       0.82      0.94      0.87      1304
      Luxury       0.99      1.00      0.99       921
      Middle       0.86      0.63      0.73       775

    accuracy                           0.88      3000
   macro avg       0.89      0.86      0.87      3000
weighted avg       0.88      0.88      0.87      3000
```

```python
import pickle

with open('RF_Properti_model.pkl', 'wb') as r:
    pickle.dump(GSCV_RF2, r)

print("Model Random Forest dengan Select Percentile berhasil disimpan")
```

[20] ✓ 0.0s Python

Regresi Model Ridge Regression vs SVM

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler, StandardScaler, LabelEncoder
from sklearn.compose import make_column_transformer
from sklearn.feature_selection import SelectKBest, SelectPercentile, SelectFromModel, RFE, f_regression
from sklearn.model_selection import GridSearchCV, train_test_split, StratifiedKFold, KFold
from sklearn.svm import SVC, SVR
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, RandomForestRegressor
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.linear_model import LogisticRegression, Ridge, Lasso, LinearRegression
from imblearn.over_sampling import SMOTE
from collections import Counter
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix, mean_squared_error, mean_absolute_error, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
```

✓ 13.6s

```python
uts = pd.read_csv(r"D:\Tugas\Sem 5\Mesin Learning\Tugas_UTS\Dataset UTS_Gasal 2425.csv")
uts.head(10000)
```

✓ 0.0s Python

	squaremeters	numberofrooms	hasyard	haspool	floors	citycode	citypartrange	numprevowners	made	isnewbuilt	hasstormprotector	basement	attic
0	75523	3	no	yes	63	9373	3	8	2005	old	yes	4313	9005
1	55712	58	no	yes	19	34457	6	8	2021	old	no	2937	8852
2	86929	100	yes	no	11	98155	3	4	2003	new	no	6326	4748
3	51522	3	no	no	61	9047	8	3	2012	new	yes	632	5792
4	96470	74	yes	no	21	92029	4	2	2011	new	yes	5414	1172
...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	341	83	no	no	8	1960	4	4	1993	new	yes	2366	4016
9996	21514	5	no	yes	11	91373	1	1	1999	old	no	2584	5266
9997	1726	89	no	yes	5	73133	7	6	2009	old	yes	9311	1698
9998	44403	29	yes	yes	12	34606	9	4	1990	old	yes	9061	1742
9999	1440	84	no	no	49	18412	6	10	1994	new	no	8485	2024

10000 rows × 18 columns

```python
print("=" * 50)
print("Informasi DataFrame:")
print("=" * 50)
uts.info()
print("\n")

print("=" * 50)
print("Jumlah Nilai Kosong per Kolom:")
print("=" * 50)
print(uts.isnull().sum())
print("\n")

print("=" * 50)
print("Jumlah Total Baris Duplikat:")
print("=" * 50)
print(uts.duplicated().sum())
print("\n")

if uts.duplicated().sum() > 0:
    print("=" * 50)
    print("Detail Baris Duplikat:")
    print("=" * 50)
    print(uts[uts.duplicated()])
else:
    print("=" * 50)
    print("Tidak Ditemukan Baris Duplikat dalam DataFrame.")
    print("=" * 50)
```

✓ 0.0s

```
==================================================
Informasi DataFrame:
==================================================
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 18 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   squaremeters       10000 non-null  int64
 1   numberofrooms      10000 non-null  int64
 2   hasyard            10000 non-null  object
 3   haspool            10000 non-null  object
 4   floors             10000 non-null  int64
 5   citycode           10000 non-null  int64
 6   citypartrange      10000 non-null  int64
 7   numprevowners      10000 non-null  int64
 8   made               10000 non-null  int64
 9   isnewbuilt         10000 non-null  object
 10  hasstormprotector  10000 non-null  object
 11  basement           10000 non-null  int64
 12  attic              10000 non-null  int64
 13  garage             10000 non-null  int64
 14  hasstorageroom     10000 non-null  object
 15  hasguestroom       10000 non-null  int64
 16  price              10000 non-null  float64
...
```

```
pd.set_option('display.float_format', lambda x: '%.5f' % x)

uts.describe()
```

✓ 0.0s Python

	squaremeters	numberofrooms	floors	citycode	citypartrange	numprevowners	made	basement	attic	garage	hasguestroo
count	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.000(
mean	49870.13120	50.35840	50.27630	50225.48610	5.51010	5.52170	2005.48850	5033.10390	5028.01060	553.12120	4.994(
std	28774.37535	28.81670	28.88917	29006.67580	2.87202	2.85667	9.30809	2876.72954	2894.33221	262.05017	3.176
min	89.00000	1.00000	1.00000	3.00000	1.00000	1.00000	1990.00000	0.00000	1.00000	100.00000	0.000(
25%	25098.50000	25.00000	25.00000	24693.75000	3.00000	3.00000	1997.00000	2559.75000	2512.00000	327.75000	2.000(
50%	50105.50000	50.00000	50.00000	50693.00000	5.00000	5.00000	2005.50000	5092.50000	5045.00000	554.00000	5.000(
75%	74609.75000	75.00000	76.00000	75683.25000	8.00000	8.00000	2014.00000	7511.25000	7540.50000	777.25000	8.000(
max	99999.00000	100.00000	100.00000	99953.00000	10.00000	10.00000	2021.00000	10000.00000	10000.00000	1000.00000	10.000(

```
df_properti = uts.copy()
df_properti.head()
df_properti.columns
```

✓ 0.0s

```
Index(['squaremeters', 'numberofrooms', 'hasyard', 'haspool', 'floors',
       'citycode', 'citypartrange', 'numprevowners', 'made', 'isnewbuilt',
       'hasstormprotector', 'basement', 'attic', 'garage', 'hasstorageroom',
       'hasguestroom', 'price', 'category'],
      dtype='object')
```

```
X = df_properti.drop(columns=['category', 'price'], axis=1)
y = df_properti['price']

X_trainReg, X_testReg, y_trainReg, y_testReg = train_test_split(X, y, test_size=0.30, random_state=95)
print(f"Shape of X_train: {X_trainReg.shape}")
print(f"Shape of X_test: {X_testReg.shape}")
```

9] ✓ 0.0s

```
Shape of X_train: (7000, 16)
Shape of X_test: (3000, 16)
```

```
print(X.columns)
```

✓ 0.0s

```
Index(['squaremeters', 'numberofrooms', 'hasyard', 'haspool', 'floors',
       'citycode', 'citypartrange', 'numprevowners', 'made', 'isnewbuilt',
       'hasstormprotector', 'basement', 'attic', 'garage', 'hasstorageroom',
       'hasguestroom'],
      dtype='object')
```

```
cat_cols=['hasyard', 'haspool', 'isnewbuilt',
          'hasstormprotector', 'hasstorageroom']

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

X_trainReg_enc = transformer.fit_transform(X_trainReg)
X_testReg_enc = transformer.transform(X_testReg)

df_trainReg_enc = pd.DataFrame(X_trainReg_enc, columns=transformer.get_feature_names_out())
df_testReg_enc = pd.DataFrame(X_testReg_enc, columns=transformer.get_feature_names_out())

df_trainReg_enc.head(10)
df_testReg_enc.head(10)
```

✓ 0.0s

	onehotencoder__hasyard_no	onehotencoder__hasyard_yes	onehotencoder__haspool_no	onehotencoder__haspool_yes	onehotencoder__isnewbuilt_new	onehoten
0	0.00000	1.00000	0.00000	1.00000	0.00000	
1	0.00000	1.00000	1.00000	0.00000	1.00000	
2	1.00000	0.00000	0.00000	1.00000	0.00000	
3	1.00000	0.00000	1.00000	0.00000	0.00000	
4	0.00000	1.00000	1.00000	0.00000	1.00000	

```
np.set_printoptions(formatter={'float': '{:.2f}'.format})

print(X_trainReg_enc)
```

✓ 0.0s

```
[[1.00 0.00 1.00 ... 8149.00 816.00 3.00]
 [1.00 0.00 0.00 ... 331.00 814.00 2.00]
 [1.00 0.00 0.00 ... 8197.00 173.00 9.00]
 ...
 [1.00 0.00 0.00 ... 7733.00 111.00 10.00]
 [0.00 1.00 1.00 ... 6453.00 130.00 1.00]
 [1.00 0.00 0.00 ... 3497.00 107.00 10.00]]
```

```python
from sklearn.model_selection import KFold

kf = KFold(n_splits=5, shuffle=True, random_state=95)

X_folds = []
y_folds = []

for train_index, test_index in kf.split(X_trainReg_enc):
    X_folds.append((X_trainReg_enc[train_index], X_trainReg_enc[test_index]))
    y_folds.append((y_trainReg.iloc[train_index], y_trainReg.iloc[test_index]))
```

✓ 0.0s

```python
    st_scaler = StandardScaler()
    X_trainReg_stscaled = st_scaler.fit_transform(X_trainReg_enc)
    X_testReg_stscaled = st_scaler.transform(X_testReg_enc)

    selector = SelectKBest(score_func=f_regression, k='all')
    X_trainReg_kbest = selector.fit_transform(X_trainReg_stscaled, y_trainReg)
    X_testReg_kbest = selector.transform(X_testReg_stscaled)

    ridge_model = Ridge(alpha=1.0)
    ridge_model.fit(X_trainReg_kbest, y_trainReg)
    y_pred_ridge_kbest = ridge_model.predict(X_testReg_kbest)
    mse_ridge_kbest = mean_squared_error(y_testReg, y_pred_ridge_kbest)
    print("Berhasil")
```

✓ 0.0s

```python
minmax_scaler = MinMaxScaler()
X_trainReg_mmscaled = minmax_scaler.fit_transform(X_trainReg_enc)
X_testReg_mmscaled = minmax_scaler.transform(X_testReg_enc)

selector = SelectPercentile(score_func=f_regression, percentile=50)
X_trainReg_percentile = selector.fit_transform(X_trainReg_mmscaled, y_trainReg)
X_testReg_percentile = selector.transform(X_testReg_mmscaled)

ridge_model.fit(X_trainReg_percentile, y_trainReg)
y_pred_ridge_percentile = ridge_model.predict(X_testReg_percentile)
mse_ridge_percentile = mean_squared_error(y_testReg, y_pred_ridge_percentile)
print("Berhasil")
```

✓ 0.0s

```
    svr_kbest_model = SVR()
    svr_kbest_model.fit(X_trainReg_kbest, y_trainReg)
    y_pred_svr_kbest = svr_kbest_model.predict(X_testReg_kbest)
    mse_svr_kbest = mean_squared_error(y_testReg, y_pred_svr_kbest)
    print("Berhasil")
```

6] ✓ 6.1s

```
Berhasil
```

```
    svr_percentile_model = SVR()
    svr_percentile_model.fit(X_trainReg_percentile, y_trainReg)
    y_pred_svr_percentile = svr_percentile_model.predict(X_testReg_percentile)
    mse_svr_percentile = mean_squared_error(y_testReg, y_pred_svr_percentile)
    print("Berhasil")
```

7] ✓ 5.3s

```
Berhasil
```

```
    results_df = pd.DataFrame(columns=['Model', 'MAE', 'MSE', 'RMSE'])
```

8] ✓ 0.0s

```
def evaluate_model_with_grid_search(model, params, X_train, y_train, X_test, y_test, model_name):
    kf = KFold(n_splits=5, shuffle=True, random_state=95)

    grid_search = GridSearchCV(model, params, cv=kf, scoring='neg_mean_squared_error')
    grid_search.fit(X_train, y_train)

    best_model = grid_search.best_estimator_
    y_pred = best_model.predict(X_test)

    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)

    results_df.loc[len(results_df)] = [model_name, mae, mse, rmse]

    return y_pred, best_model
```

✓ 0.0s

```
ridge_params = {'alpha': [0.1, 1.0, 10.0]}
ridge_preds, ridge_best_model = evaluate_model_with_grid_search(
    Ridge(), ridge_params, X_trainReg_kbest, y_trainReg, X_testReg_kbest, y_testReg, 'Ridge Regression'
)
```

✓ 0.0s

```
svr_params = {'C': [0.1, 1.0, 10.0], 'epsilon': [0.1, 0.2, 0.5]}
svr_preds, svr_best_model = evaluate_model_with_grid_search(
    SVR(), svr_params, X_trainReg_kbest, y_trainReg, X_testReg_kbest, y_testReg, 'Support Vector Regressor'
)
```

✓ 2m 4.1s

```
from sklearn.model_selection import KFold

def evaluate_model_with_grid_search(model, params, X_train, y_train, X_test, y_test, model_name):
    kf = KFold(n_splits=5, shuffle=True, random_state=95)

    grid_search = GridSearchCV(model, params, cv=kf, scoring='neg_mean_squared_error')
    grid_search.fit(X_train, y_train)

    best_model = grid_search.best_estimator_
    y_pred = best_model.predict(X_test)

    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)

    results_df.loc[len(results_df)] = [model_name, mae, mse, rmse]

    return y_pred, best_model
```

```
Click to add a breakpoint pha': [0.1, 1.0, 10.0]}
ridge_preds, ridge_best_model = evaluate_model_with_grid_search(
    Ridge(), ridge_params, X_trainReg_kbest, y_trainReg, X_testReg_kbest, y_testReg, 'Ridge Regression'
)

svr_params = {'C': [0.1, 1.0, 10.0], 'epsilon': [0.1, 0.2, 0.5]}
svr_preds, svr_best_model = evaluate_model_with_grid_search(
    SVR(), svr_params, X_trainReg_kbest, y_trainReg, X_testReg_kbest, y_testReg, 'Support Vector Regressor'
)
```

✓ 2m 6.0s

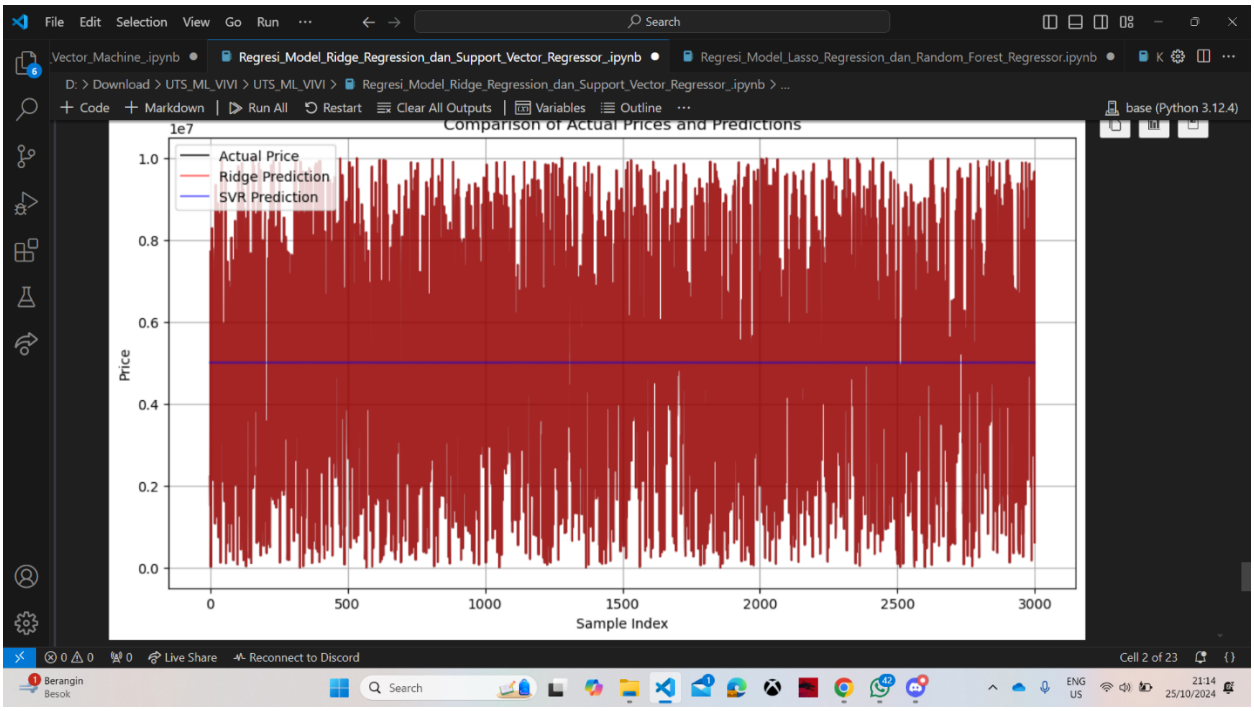
```
print(results_df)
```

✓ 0.0s

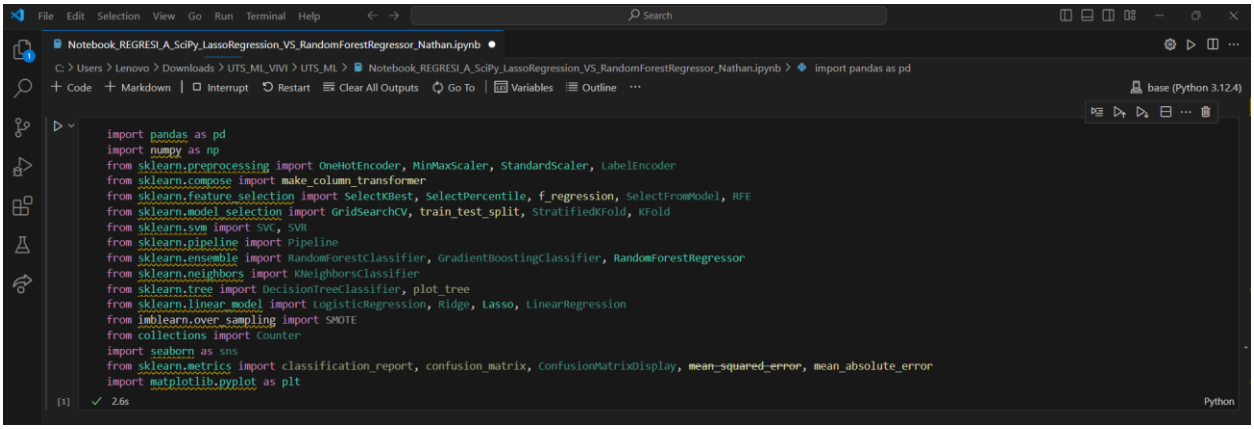
```
                    Model           MAE                 MSE          RMSE
0         Ridge Regression    1466.48557       3525683.20394    1877.68027
1  Support Vector Regressor 2500856.81968 8371273461984.10352 2893315.30635
2         Ridge Regression    1466.48557       3525683.20394    1877.68027
3  Support Vector Regressor 2500856.81968 8371273461984.10352 2893315.30635
```

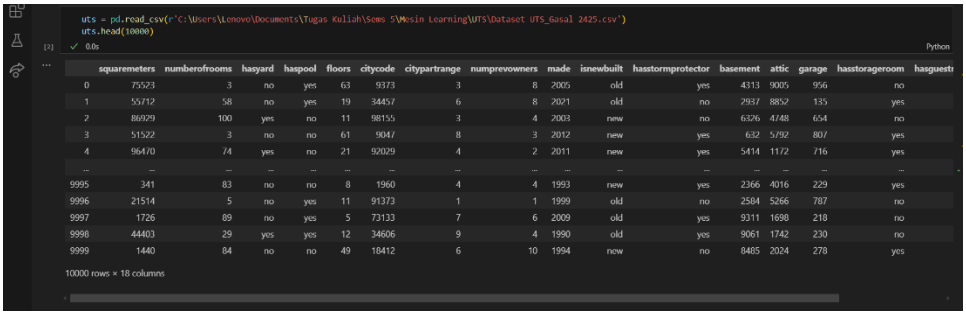
```
comparison_df = pd.DataFrame({
    'Actual Price': y_testReg,
    'Ridge Prediction': ridge_preds,
    'SVR Prediction': svr_preds,
})


comparison_df['Ridge Difference'] = comparison_df['Actual Price'] - comparison_df['Ridge Prediction']
comparison_df['SVR Difference'] = comparison_df['Actual Price'] - comparison_df['SVR Prediction']

print(comparison_df.head())


plt.figure(figsize=(12, 6))
plt.plot(comparison_df['Actual Price'].values, label='Actual Price', color='black', alpha=0.7)
plt.plot(comparison_df['Ridge Prediction'].values, label='Ridge Prediction', color='red', alpha=0.5)
plt.plot(comparison_df['SVR Prediction'].values, label='SVR Prediction', color='blue', alpha=0.5)
plt.title('Comparison of Actual Prices and Predictions')
plt.xlabel('Sample Index')
plt.ylabel('Price')
plt.legend()
plt.grid()
plt.show()
```

✓ 0.5s



Regresi Model Lasso Regression Vs Random Forest Regressor





```
uts = pd.read_csv(r'C:\Users\Lenovo\Documents\Tugas Kuliah\Sems 5\Mesin Learning\UTS\Dataset UTS_Gasal 2425.csv')
uts.head(10000)
```

	squaremeters	numberofrooms	hasyard	haspool	floors	citycode	citypartrange	numprevowners	made	isnewbuilt	hasstormprotector	basement	attic	garage	hasstorageroom	hasguestr
0	75523	3	no	yes	63	9373	3	8	2005	old	yes	4313	9005	956	no	
1	55712	58	no	yes	19	34457	6	8	2021	old	no	2937	8852	135	yes	
2	86929	100	yes	no	11	98155	3	4	2003	new	no	6326	4748	654	no	
3	51522	3	no	no	61	9047	8	3	2012	new	yes	632	5792	807	yes	
4	96470	74	yes	no	21	92029	4	2	2011	new	yes	5414	1172	716	yes	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
9995	341	83	no	no	8	1960	4	4	1993	new	yes	2366	4016	229	yes	
9996	21514	5	no	yes	11	91373	1	1	1999	old	no	2584	5266	787	no	
9997	1726	89	no	yes	5	73133	7	6	2009	old	yes	9311	1698	218	no	
9998	44403	29	yes	yes	12	34606	9	4	1990	old	yes	9061	1742	230	no	
9999	1440	84	no	no	49	18412	6	10	1994	new	no	8485	2024	278	yes	

10000 rows × 18 columns

```python
print("=" * 50)
print("Informasi DataFrame:")
print("=" * 50)
uts.info()
print("\n")

print("=" * 50)
print("Jumlah Nilai Kosong per Kolom:")
print("=" * 50)
print(uts.isnull().sum())
print("\n")

print("=" * 50)
print("Jumlah Total Baris Duplikat:")
print("=" * 50)
print(uts.duplicated().sum())
print("\n")

if uts.duplicated().sum() > 0:
    print("=" * 50)
    print("Detail Baris Duplikat:")
    print("=" * 50)
    print(uts[uts.duplicated()])
else:
    print("=" * 50)
    print("Tidak Ditemukan Baris Duplikat dalam DataFrame.")
    print("=" * 50)
```

[3] ✓ 0.0s Python

```
==================================================
Informasi DataFrame:
```

⊗ 0 ⚠ 18 0 Live Share

```
==================================================
Informasi DataFrame:
==================================================
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   squaremeters      10000 non-null  int64
 1   numberofrooms     10000 non-null  int64
 2   hasyard           10000 non-null  object
 3   haspool           10000 non-null  object
 4   floors            10000 non-null  int64
 5   citycode          10000 non-null  int64
 6   citypartrange     10000 non-null  int64
 7   numprevowners     10000 non-null  int64
 8   made              10000 non-null  int64
 9   isnewbuilt        10000 non-null  object
 10  hasstormprotector 10000 non-null  object
 11  basement          10000 non-null  int64
 12  attic             10000 non-null  int64
 13  garage            10000 non-null  int64
 14  hasstorageroom    10000 non-null  object
 15  hasguestroom      10000 non-null  int64
 16  price             10000 non-null  float64
...

==================================================
Tidak Ditemukan Baris Duplikat dalam DataFrame.
==================================================
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```python
pd.set_option('display.float_format', lambda x: '%.5f' % x)

uts.describe()
```

[4] ✓ 0.0s Python

	squaremeters	numberofrooms	floors	citycode	citypartrange	numprevowners	made	basement	attic	garage	hasguestroom	price
count	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000
mean	49870.13120	50.35840	50.27630	50225.48610	5.51010	5.52170	2005.48850	5033.10390	5028.01060	553.12120	4.99460	4993447.52575
std	28774.37535	28.81670	28.88917	29006.67580	2.87202	2.85667	9.30809	2876.72954	2894.33221	262.05017	3.17641	2877424.10995
min	89.00000	1.00000	1.00000	3.00000	1.00000	1.00000	1990.00000	0.00000	1.00000	100.00000	0.00000	10313.50000
25%	25098.50000	25.00000	25.00000	24693.75000	3.00000	3.00000	1997.00000	2559.75000	2512.00000	327.75000	2.00000	2516401.95000
50%	50105.50000	50.00000	50.00000	50693.00000	5.00000	5.00000	2005.50000	5092.50000	5045.00000	554.00000	5.00000	5016180.30000
75%	74609.75000	75.00000	76.00000	75683.25000	8.00000	8.00000	2014.00000	7511.25000	7540.50000	777.25000	8.00000	7469092.45000
max	99999.00000	100.00000	100.00000	99953.00000	10.00000	10.00000	2021.00000	10000.00000	10000.00000	1000.00000	10.00000	10006771.20000

```python
df_properti = uts.copy()
df_properti.head()
df_properti.columns
```

[5] ✓ 0.0s Python

```
Index(['squaremeters', 'numberofrooms', 'hasyard', 'haspool', 'floors',
       'citycode', 'citypartrange', 'numprevowners', 'made', 'isnewbuilt',
       'hasstormprotector', 'basement', 'attic', 'garage', 'hasstorageroom',
       'hasguestroom', 'price', 'category'],
      dtype='object')
```

```python
X = df_properti.drop(columns=['category', 'price'], axis=1)
y = df_properti['price']

X_trainReg, X_testReg, y_trainReg, y_testReg = train_test_split(X, y, test_size=0.30, random_state=95)
print(f"Shape of X_train: {X_trainReg.shape}")
print(f"Shape of X_test: {X_testReg.shape}")
```

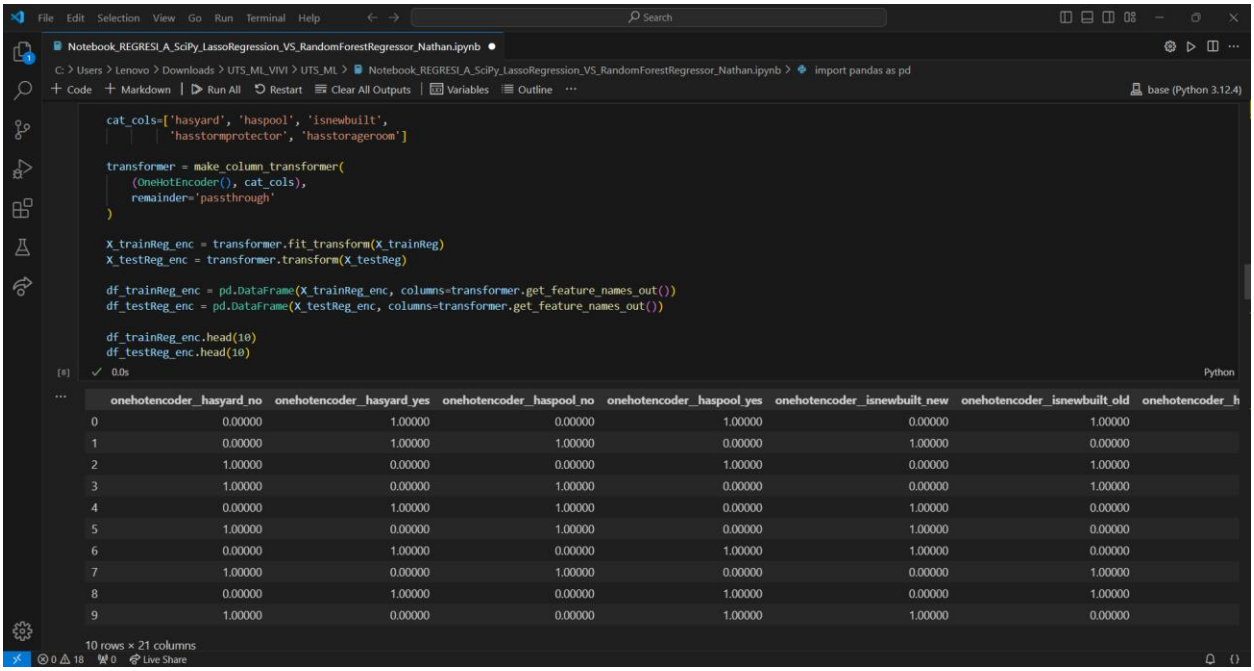
[6] ✓ 0.0s Python

```
Shape of X_train: (7000, 16)
Shape of X_test: (3000, 16)
```

```python
print(X.columns)
```

[7] ✓ 0.0s Python

```
Index(['squaremeters', 'numberofrooms', 'hasyard', 'haspool', 'floors',
       'citycode', 'citypartrange', 'numprevowners', 'made', 'isnewbuilt',
       'hasstormprotector', 'basement', 'attic', 'garage', 'hasstorageroom',
       'hasguestroom'],
      dtype='object')
```



File Edit Selection View Go Run Terminal Help — Search

Notebook_REGRESI_A_SciPy_LassoRegression_VS_RandomForestRegressor_Nathan.ipynb

C: > Users > Lenovo > Downloads > UTS_ML_VIVI > UTS_ML > Notebook_REGRESI_A_SciPy_LassoRegression_VS_RandomForestRegressor_Nathan.ipynb > import pandas as pd

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· base (Python 3.12.4)

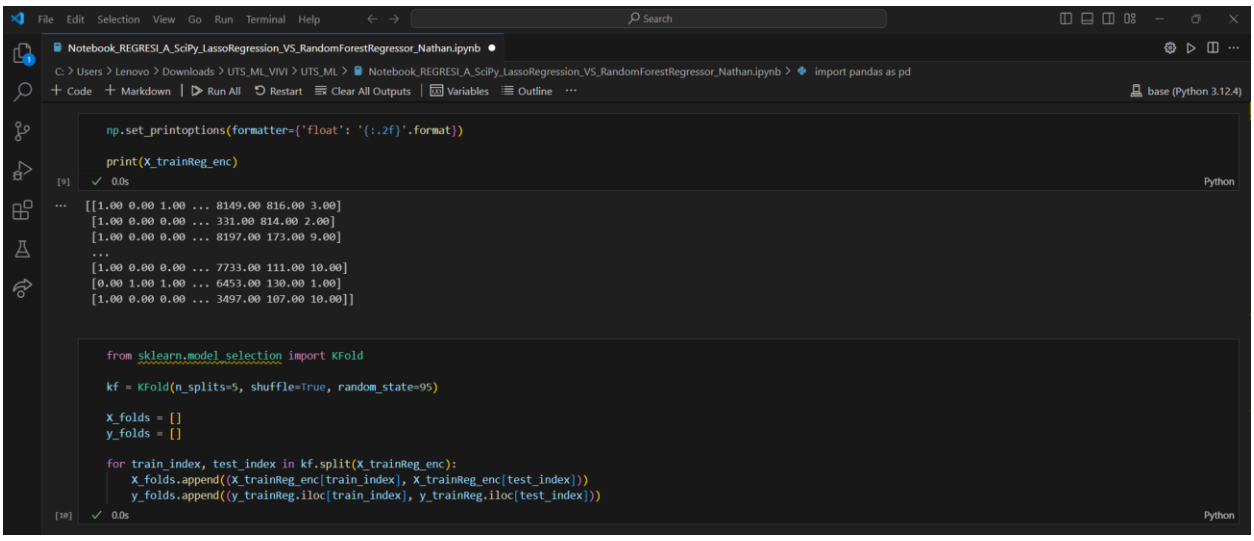
```python
cat_cols=['hasyard', 'haspool', 'isnewbuilt',
          'hasstormprotector', 'hasstorageroom']

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

X_trainReg_enc = transformer.fit_transform(X_trainReg)
X_testReg_enc = transformer.transform(X_testReg)

df_trainReg_enc = pd.DataFrame(X_trainReg_enc, columns=transformer.get_feature_names_out())
df_testReg_enc = pd.DataFrame(X_testReg_enc, columns=transformer.get_feature_names_out())

df_trainReg_enc.head(10)
df_testReg_enc.head(10)
```

[8] ✓ 0.0s Python

	onehotencoder__hasyard_no	onehotencoder__hasyard_yes	onehotencoder__haspool_no	onehotencoder__haspool_yes	onehotencoder__isnewbuilt_new	onehotencoder__isnewbuilt_old	onehotencoder_h
0	0.00000	1.00000	0.00000	1.00000	0.00000	1.00000	
1	0.00000	1.00000	1.00000	0.00000	1.00000	0.00000	
2	1.00000	0.00000	0.00000	1.00000	0.00000	1.00000	
3	1.00000	0.00000	1.00000	0.00000	0.00000	1.00000	
4	0.00000	1.00000	1.00000	0.00000	1.00000	0.00000	
5	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000	
6	0.00000	1.00000	0.00000	1.00000	1.00000	0.00000	
7	1.00000	0.00000	1.00000	0.00000	0.00000	1.00000	
8	0.00000	1.00000	0.00000	1.00000	0.00000	1.00000	
9	1.00000	0.00000	0.00000	1.00000	1.00000	0.00000	

10 rows × 21 columns

⊗ 0 ⚠ 18 0 Live Share



File Edit Selection View Go Run Terminal Help — Search

Notebook_REGRESI_A_SciPy_LassoRegression_VS_RandomForestRegressor_Nathan.ipynb

C: > Users > Lenovo > Downloads > UTS_ML_VIVI > UTS_ML > Notebook_REGRESI_A_SciPy_LassoRegression_VS_RandomForestRegressor_Nathan.ipynb > import pandas as pd

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· base (Python 3.12.4)

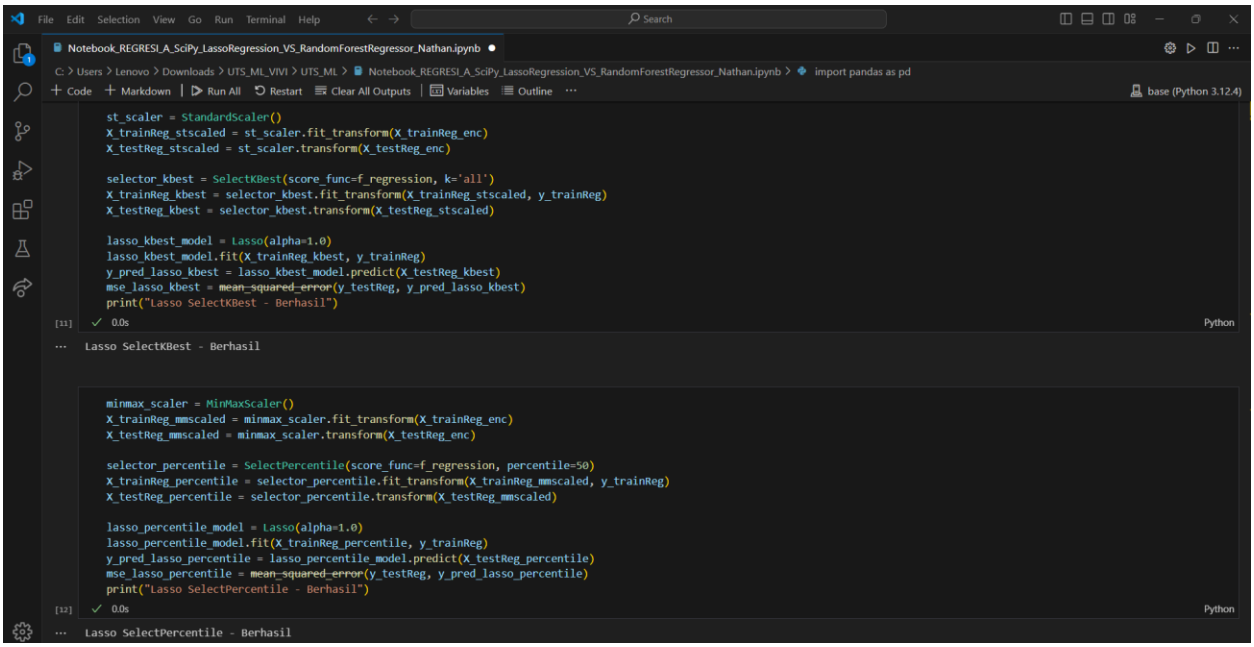
```python
np.set_printoptions(formatter={'float': '{:.2f}'.format})

print(X_trainReg_enc)
```

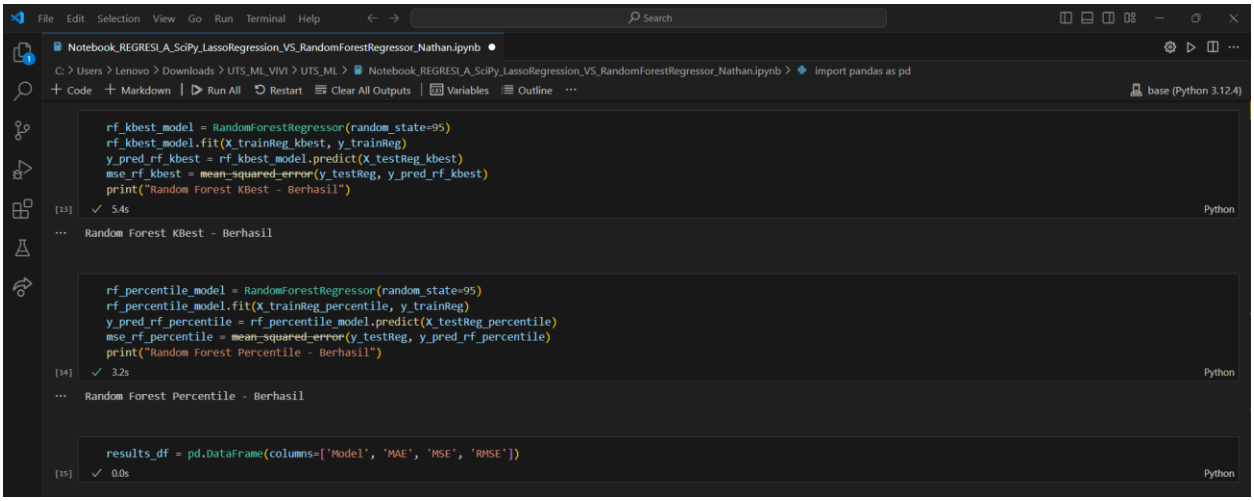
[9] ✓ 0.0s Python

```
[[1.00 0.00 1.00 ... 8149.00 816.00 3.00]
 [1.00 0.00 0.00 ... 331.00 814.00 2.00]
 [1.00 0.00 0.00 ... 8197.00 173.00 9.00]
 ...
 [1.00 0.00 0.00 ... 7733.00 111.00 10.00]
 [0.00 1.00 1.00 ... 6453.00 130.00 1.00]
 [1.00 0.00 0.00 ... 3497.00 107.00 10.00]]
```

```python
from sklearn.model_selection import KFold

kf = KFold(n_splits=5, shuffle=True, random_state=95)

X_folds = []
y_folds = []

for train_index, test_index in kf.split(X_trainReg_enc):
    X_folds.append((X_trainReg_enc[train_index], X_trainReg_enc[test_index]))
    y_folds.append((y_trainReg.iloc[train_index], y_trainReg.iloc[test_index]))
```

[10] ✓ 0.0s Python


Notebook_REGRESI_A_SciPy_LassoRegression_VS_RandomForestRegressor_Nathan.ipynb
C: > Users > Lenovo > Downloads > UTS_ML_VIVI > UTS_ML > Notebook_REGRESI_A_SciPy_LassoRegression_VS_RandomForestRegressor_Nathan.ipynb > import pandas as pd
+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ...
base (Python 3.12.4)
st_scaler = StandardScaler()
X_trainReg_stscaled = st_scaler.fit_transform(X_trainReg_enc)
X_testReg_stscaled = st_scaler.transform(X_testReg_enc)
selector_kbest = SelectKBest(score_func=f_regression, k='all')
X_trainReg_kbest = selector_kbest.fit_transform(X_trainReg_stscaled, y_trainReg)
X_testReg_kbest = selector_kbest.transform(X_testReg_stscaled)
lasso_kbest_model = Lasso(alpha=1.0)
lasso_kbest_model.fit(X_trainReg_kbest, y_trainReg)
y_pred_lasso_kbest = lasso_kbest_model.predict(X_testReg_kbest)
mse_lasso_kbest = mean_squared_error(y_testReg, y_pred_lasso_kbest)
print("Lasso SelectKBest - Berhasil")
[11] 0.0s
Python
Lasso SelectKBest - Berhasil
minmax_scaler = MinMaxScaler()
X_trainReg_mmscaled = minmax_scaler.fit_transform(X_trainReg_enc)
X_testReg_mmscaled = minmax_scaler.transform(X_testReg_enc)
selector_percentile = SelectPercentile(score_func=f_regression, percentile=50)
X_trainReg_percentile = selector_percentile.fit_transform(X_trainReg_mmscaled, y_trainReg)
X_testReg_percentile = selector_percentile.transform(X_testReg_mmscaled)
lasso_percentile_model = Lasso(alpha=1.0)
lasso_percentile_model.fit(X_trainReg_percentile, y_trainReg)
y_pred_lasso_percentile = lasso_percentile_model.predict(X_testReg_percentile)
mse_lasso_percentile = mean_squared_error(y_testReg, y_pred_lasso_percentile)
print("Lasso SelectPercentile - Berhasil")
[12] 0.0s
Python
Lasso SelectPercentile - Berhasil


Notebook_REGRESI_A_SciPy_LassoRegression_VS_RandomForestRegressor_Nathan.ipynb
C: > Users > Lenovo > Downloads > UTS_ML_VIVI > UTS_ML > Notebook_REGRESI_A_SciPy_LassoRegression_VS_RandomForestRegressor_Nathan.ipynb > import pandas as pd
+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ...
base (Python 3.12.4)
rf_kbest_model = RandomForestRegressor(random_state=95)
rf_kbest_model.fit(X_trainReg_kbest, y_trainReg)
y_pred_rf_kbest = rf_kbest_model.predict(X_testReg_kbest)
mse_rf_kbest = mean_squared_error(y_testReg, y_pred_rf_kbest)
print("Random Forest KBest - Berhasil")
[13] 5.4s
Python
Random Forest KBest - Berhasil
rf_percentile_model = RandomForestRegressor(random_state=95)
rf_percentile_model.fit(X_trainReg_percentile, y_trainReg)
y_pred_rf_percentile = rf_percentile_model.predict(X_testReg_percentile)
mse_rf_percentile = mean_squared_error(y_testReg, y_pred_rf_percentile)
print("Random Forest Percentile - Berhasil")
[14] 3.2s
Python
Random Forest Percentile - Berhasil
results_df = pd.DataFrame(columns=['Model', 'MAE', 'MSE', 'RMSE'])
[15] 0.0s
Python


def evaluate_model_with_grid_search(model, params, X_train, y_train, X_test, y_test, model_name):
kf = KFold(n_splits=5, shuffle=True, random_state=95)
grid_search = GridSearchCV(model, params, cv=kf, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
results_df.loc[len(results_df)] = [model_name, mae, mse, rmse]
return y_pred, best_model
[16] 0.0s
Python

```python
from sklearn.model_selection import KFold

def evaluate_model_with_grid_search(model, params, X_train, y_train, X_test, y_test, model_name):
    kf = KFold(n_splits=5, shuffle=True, random_state=95)

    grid_search = GridSearchCV(model, params, cv=kf, scoring='neg_mean_squared_error')
    grid_search.fit(X_train, y_train)

    best_model = grid_search.best_estimator_
    y_pred = best_model.predict(X_test)

    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)

    results_df.loc[len(results_df)] = [model_name, mae, mse, rmse]

    return y_pred, best_model
```

[17] ✓ 0.0s Python

Evaluasi

```python
lasso_params = {'alpha': [0.1, 1.0, 10.0]}
lasso_preds, lasso_best_model = evaluate_model_with_grid_search(
    Lasso(), lasso_params, X_trainReg_kbest, y_trainReg, X_testReg_kbest, y_testReg, 'Lasso Regression'
)
```

[18] ✓ 0.0s Python

Evaluasi

```python
lasso_params = {'alpha': [0.1, 1.0, 10.0]}
lasso_preds, lasso_best_model = evaluate_model_with_grid_search(
    Lasso(), lasso_params, X_trainReg_kbest, y_trainReg, X_testReg_kbest, y_testReg, 'Lasso Regression'
)
```

[18] ✓ 0.0s Python

```python
rf_params = {'n_estimators': [100, 200], 'max_depth': [None, 10, 20]}
rf_preds, rf_best_model = evaluate_model_with_grid_search(
    RandomForestRegressor(random_state=95), rf_params, X_trainReg_kbest, y_trainReg, X_testReg_kbest, y_testReg, 'Random Forest Regressor'
)
```

[19] ✓ 3m 8.3s Python

```python
print(results_df)
```

[20] ✓ 0.0s Python

```
                        Model        MAE            MSE       RMSE
0         Lasso Regression 1466.16979  3522128.18136 1876.73338
1  Random Forest Regressor 3145.35292 15358471.79534 3918.98862
```

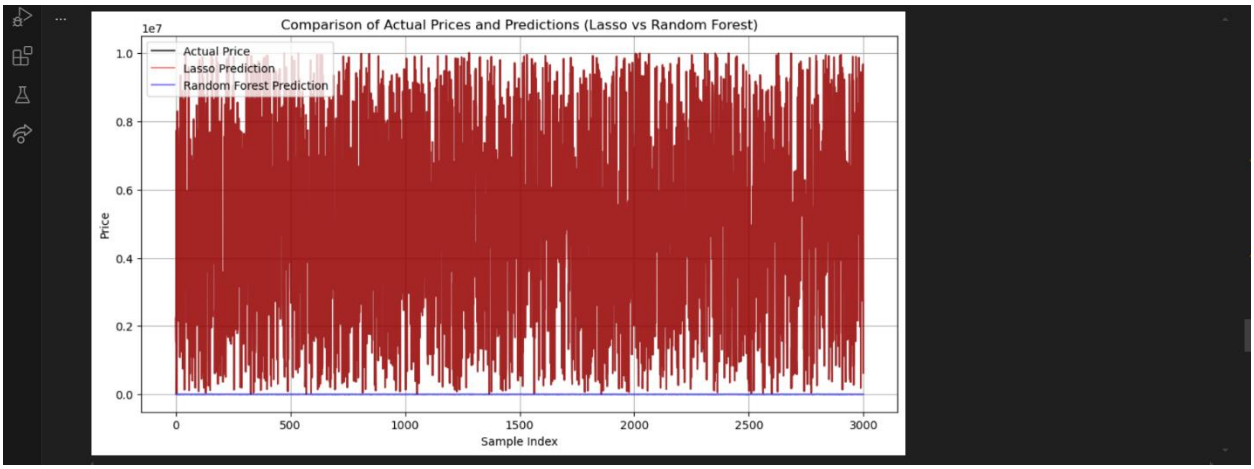
```python
comparison_df = pd.DataFrame({
    'Actual Price': y_testReg,
    'Lasso Prediction': lasso_preds,
    'Random Forest Prediction': rf_preds, \
})

comparison_df['Lasso Difference'] = comparison_df['Actual Price'] - comparison_df['Lasso Prediction']
comparison_df['Random Forest Prediction'] = comparison_df['Actual Price'] - comparison_df['Random Forest Prediction']

print(comparison_df.head())
```

[21] ✓ 0.0s Python

```
        Actual Price  Lasso Prediction  Random Forest Prediction  \
7592 2232476.20000     2231696.29454               7570.64200
8738 1523752.90000     1523193.05537               2870.85650
212  7725844.60000     7729395.21174              -4444.37750
9378   28656.50000       27961.25891                 33.03300
9136  755909.20000      757601.85636                713.29650

      Lasso Difference
7592         779.90546
8738         559.84463
212        -3550.61174
9378         695.24109
9136       -1692.65636
```

```python
plt.figure(figsize=(12, 6))
plt.plot(comparison_df['Actual Price'].values, label='Actual Price', color='black', alpha=0.7)
plt.plot(comparison_df['Lasso Prediction'].values, label='Lasso Prediction', color='red', alpha=0.5)
plt.plot(comparison_df['Random Forest Prediction'].values, label='Random Forest Prediction', color='blue', alpha=0.5)  # Pastikan nama kolom ini benar
plt.title('Comparison of Actual Prices and Predictions (Lasso vs Random Forest)')
plt.xlabel('Sample Index')
plt.ylabel('Price')
plt.legend()
plt.grid()
plt.show()
```

[22] ✓ 0.4s Python




```
import pickle

with open('BestModel_REG_Lasso_SciPy.pkl', 'wb') as r:
    pickle.dump(rf_kbest_model, r)

print("Model Lasso Regression berhasil disimpan")
```

```
[23] ✓ 0.0s                                                    Python
Model Lasso Regression berhasil disimpan
```

Klasifikasi Gradient Boosting Classifier VS Support Vector Machine

```
+ Code + Markdown | Interrupt Restart Clear All Outputs Go To | Variables Outline ···   base (Python 3.12.4)

import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler, StandardScaler, LabelEncoder
from sklearn.compose import make_column_transformer
from sklearn.feature_selection import SelectKBest, SelectPercentile, SelectFromModel, RFE
from sklearn.model_selection import GridSearchCV, train_test_split, StratifiedKFold, KFold
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.linear_model import LogisticRegression, Ridge, Lasso, LinearRegression
from imblearn.over_sampling import SMOTE
from collections import Counter
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix, mean_squared_error, mean_absolute_error, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

[1] ✓ 2.7s                                                     Python
```

```
uts = pd.read_csv(r'C:\Users\Lenovo\Documents\Tugas Kuliah\Sems 5\Mesin Learning\UTS\Dataset UTS_Gasal 2425.csv')
uts.head(10000)

[2] ✓ 0.0s                                                     Python
```

	squaremeters	numberofrooms	hasyard	haspool	floors	citycode	citypartrange	numprevowners	made	isnewbuilt	hasstormprotector	basement	attic	garage	hasstorageroom	hasguestr
0	75523	3	no	yes	63	9373	3	8	2005	old	yes	4313	9005	956	no	
1	55712	58	no	yes	19	34457	6	8	2021	old	no	2937	8852	135	yes	
2	86929	100	yes	no	11	98155	3	4	2003	new	no	6326	4748	654	no	
3	51522	3	no	no	61	9047	8	3	2012	new	yes	632	5792	807	yes	
4	96470	74	yes	no	21	92029	4	2	2011	new	yes	5414	1172	716	yes	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
9995	341	83	no	no	8	1960	4	4	1993	new	yes	2366	4016	229	yes	
9996	21514	5	no	yes	11	91373	1	1	1999	old	no	2584	5266	787	no	
9997	1726	89	no	yes	5	73133	7	6	2009	old	yes	9311	1698	218	no	
9998	44403	29	yes	yes	12	34606	9	4	1990	old	yes	9061	1742	230	no	
9999	1440	84	no	no	49	18412	6	10	1994	new	no	8485	2024	278	yes	

10000 rows × 18 columns

```python
print("=" * 50)
print("Informasi DataFrame:")
print("=" * 50)
uts.info()
print("\n")

print("=" * 50)
print("Jumlah Nilai Kosong per Kolom:")
print("=" * 50)
print(uts.isnull().sum())
print("\n")

print("=" * 50)
print("Jumlah Total Baris Duplikat:")
print("=" * 50)
print(uts.duplicated().sum())
print("\n")

if uts.duplicated().sum() > 0:
    print("=" * 50)
    print("Detail Baris Duplikat:")
    print("=" * 50)
    print(uts[uts.duplicated()])
else:
    print("=" * 50)
    print("Tidak Ditemukan Baris Duplikat dalam DataFrame.")
    print("=" * 50)
```

[3] ✓ 0.0s Python

```
==================================================
Informasi DataFrame:
==================================================
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 18 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   squaremeters       10000 non-null  int64
 1   numberofrooms      10000 non-null  int64
 2   hasyard            10000 non-null  object
 3   haspool            10000 non-null  object
 4   floors             10000 non-null  int64
 5   citycode           10000 non-null  int64
 6   citypartrange      10000 non-null  int64
 7   numprevowners      10000 non-null  int64
 8   made               10000 non-null  int64
 9   isnewbuilt         10000 non-null  object
 10  hasstormprotector  10000 non-null  object
 11  basement           10000 non-null  int64
 12  attic              10000 non-null  int64
 13  garage             10000 non-null  int64
 14  hasstorageroom     10000 non-null  object
 15  hasguestroom       10000 non-null  int64
 16  price              10000 non-null  float64
...

==================================================
Tidak Ditemukan Baris Duplikat dalam DataFrame.
==================================================
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```python
pd.set_option('display.float_format', lambda x: '%.5f' % x)

uts.describe()
```

[4] ✓ 0.0s Python

	squaremeters	numberofrooms	floors	citycode	citypartrange	numprevowners	made	basement	attic	garage	hasguestroom	price
count	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000	10000.00000
mean	49870.13120	50.35840	50.27630	50225.48610	5.51010	5.52170	2005.48850	5033.10390	5028.01060	553.12120	4.99460	4993447.52575
std	28774.37535	28.81670	28.88917	29006.67580	2.87202	2.85667	9.30809	2876.72954	2894.33221	262.05017	3.17641	2877424.10995
min	89.00000	1.00000	1.00000	3.00000	1.00000	1.00000	1990.00000	0.00000	1.00000	100.00000	0.00000	10313.50000
25%	25098.50000	25.00000	25.00000	24693.75000	3.00000	3.00000	1997.00000	2559.75000	2512.00000	327.75000	2.00000	2516401.95000
50%	50105.50000	50.00000	50.00000	50693.00000	5.00000	5.00000	2005.50000	5092.50000	5045.00000	554.00000	5.00000	5016180.30000
75%	74609.75000	75.00000	76.00000	75683.25000	8.00000	8.00000	2014.00000	7511.25000	7540.50000	777.25000	8.00000	7469092.45000
max	99999.00000	100.00000	100.00000	99953.00000	10.00000	10.00000	2021.00000	10000.00000	10000.00000	1000.00000	10.00000	10006771.20000

```python
df_properti = uts.copy()
df_properti.head()
df_properti.columns
```

[5] ✓ 0.0s Python

```
Index(['squaremeters', 'numberofrooms', 'hasyard', 'haspool', 'floors',
       'citycode', 'citypartrange', 'numprevowners', 'made', 'isnewbuilt',
       'hasstormprotector', 'basement', 'attic', 'garage', 'hasstorageroom',
       'hasguestroom', 'price', 'category'],
      dtype='object')
```

```python
X = df_properti.drop(columns=['price', 'category'], axis=1)
y = df_properti['category']

X_train_bf, X_test, y_train_bf, y_test = train_test_split(X, y, test_size=0.30, random_state=95)
print(f"Shape of X_train: {X_train_bf.shape}")
print(f"Shape of X_test: {X_test.shape}")
```

[6] ✓ 0.0s Python

```
Shape of X_train: (7000, 16)
Shape of X_test: (3000, 16)
```

```python
print(X.columns)
```

[7] ✓ 0.0s Python

```
Index(['squaremeters', 'numberofrooms', 'hasyard', 'haspool', 'floors',
       'citycode', 'citypartrange', 'numprevowners', 'made', 'isnewbuilt',
       'hasstormprotector', 'basement', 'attic', 'garage', 'hasstorageroom',
       'hasguestroom'],
      dtype='object')
```

```python
cat_cols=['hasyard', 'haspool', 'isnewbuilt',
          'hasstormprotector', 'hasstorageroom']

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

X_train_enc = transformer.fit_transform(X_train_bf)
X_test_enc = transformer.transform(X_test)

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())
np.set_printoptions(formatter={'float': '{:.2f}'.format})

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

[8] ✓ 0.0s Python

	onehotencoder__hasyard_no	onehotencoder__hasyard_yes	onehotencoder__haspool_no	onehotencoder__haspool_yes	onehotencoder__isnewbuilt_new	onehotencoder__isnewbuilt_old	onehotencoder__h
0	0.00000	1.00000	0.00000	1.00000	0.00000	1.00000	
1	0.00000	1.00000	1.00000	0.00000	1.00000	0.00000	
2	1.00000	0.00000	0.00000	1.00000	0.00000	1.00000	
3	1.00000	0.00000	1.00000	0.00000	0.00000	1.00000	
4	0.00000	1.00000	1.00000	0.00000	1.00000	0.00000	
5	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000	
6	0.00000	1.00000	0.00000	1.00000	1.00000	0.00000	
7	1.00000	0.00000	1.00000	0.00000	0.00000	1.00000	
8	0.00000	1.00000	0.00000	1.00000	0.00000	1.00000	
9	1.00000	0.00000	0.00000	1.00000	1.00000	0.00000	

```python
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=95)

X_folds = []
y_folds = []

for train_index, test_index in skf.split(X_train_enc, y_train_bf):
    X_folds.append((X_train_enc[train_index], X_train_enc[test_index]))
    y_folds.append((y_train_bf.iloc[train_index], y_train_bf.iloc[test_index]))
```

[9] ✓ 0.0s Python

Gradient Boosting Classifier

```python
pipe_GBC = Pipeline(steps=[
    ('scale', MinMaxScaler()),
    ('feat_select', SelectKBest()),
    ('clf', GradientBoostingClassifier(random_state=95))
])

param_grid_GBC = {
    'feat_select__k': np.arange(2, 5),
    'clf__n_estimators': [100, 150],
    'clf__learning_rate': [0.01, 0.1, 1]
}

GSCV_GBC = GridSearchCV(pipe_GBC, param_grid_GBC, cv=StratifiedKFold(n_splits=5))
GSCV_GBC.fit(X_train_enc, y_train_bf)

mask = GSCV_GBC.best_estimator_.named_steps['feat_select'].get_support()
selected_features = df_train_enc.columns[mask]

print("Best model: {}".format(GSCV_GBC.best_estimator_))
print("Selected features: {}".format(selected_features))
print("Best CV score: {:.2f}".format(GSCV_GBC.best_score_))
print("Test set score: {:.2f}".format(GSCV_GBC.score(X_test_enc, y_test)))
```

[10] ✓ 2m 3.1s Python

```
Best model: Pipeline(steps=[('scale', MinMaxScaler()), ('feat_select', SelectKBest(k=4)),
                ('clf',
                 GradientBoostingClassifier(learning_rate=1, random_state=95))])
Selected features: Index(['onehotencoder__hasyard_yes', 'onehotencoder__haspool_no',
       'onehotencoder__haspool_yes', 'remainder__squaremeters'],
      dtype='object')
Best CV score: 0.94
Test set score: 0.94
```

```
pipe_GBC = Pipeline(steps=[
    ('scale', MinMaxScaler()),
    ('feat_select', SelectPercentile()),
    ('clf', GradientBoostingClassifier(random_state=95))
])

param_grid_GBC = {
    'feat_select__percentile': [10, 20, 30],
    'clf__n_estimators': [100, 150],
    'clf__learning_rate': [0.01, 0.1, 1]
}

GSCV_GBC2 = GridSearchCV(pipe_GBC, param_grid_GBC, cv=StratifiedKFold(n_splits=5))
GSCV_GBC2.fit(X_train_enc, y_train_bf)

mask = GSCV_GBC2.best_estimator_.named_steps['feat_select'].get_support()
selected_features = df_train_enc.columns[mask]

print("Best model: {}".format(GSCV_GBC2.best_estimator_))
print("Selected features: {}".format(selected_features))
print("Best CV score: {:.2f}".format(GSCV_GBC2.best_score_))
print("Test set score: {:.2f}".format(GSCV_GBC2.score(X_test_enc, y_test)))
```

[11] ✓ 1m 58.4s Python

```
Best model: Pipeline(steps=[('scale', MinMaxScaler()),
                ('feat_select', SelectPercentile(percentile=30)),
                ('clf',
                 GradientBoostingClassifier(learning_rate=0.01,
                                            n_estimators=150,
                                            random_state=95))])
Selected features: Index(['onehotencoder__hasyard_no', 'onehotencoder__hasyard_yes',
       'onehotencoder__haspool_no', 'onehotencoder__haspool_yes',
       'onehotencoder__isnewbuilt_new', 'remainder__squaremeters'],
      dtype='object')
Best CV score: 1.00
```

```
Best model: Pipeline(steps=[('scale', MinMaxScaler()),
                ('feat_select', SelectPercentile(percentile=30)),
                ('clf',
                 GradientBoostingClassifier(learning_rate=0.01,
                                            n_estimators=150,
                                            random_state=95))])
Selected features: Index(['onehotencoder__hasyard_no', 'onehotencoder__hasyard_yes',
       'onehotencoder__haspool_no', 'onehotencoder__haspool_yes',
       'onehotencoder__isnewbuilt_new', 'remainder__squaremeters'],
      dtype='object')
Best CV score: 1.00
Test set score: 1.00
```

```
pipe_SVC = Pipeline(steps=[
    ('scale', StandardScaler()),
    ('feat_select', SelectKBest()),
    ('clf', SVC())
])

param_grid_SVC = {
    'feat_select__k': np.arange(2, 5),
    'clf__C': [0.1, 1, 10],
    'clf__kernel': ['linear', 'rbf']
}

GSCV_SVC = GridSearchCV(pipe_SVC, param_grid_SVC, cv=StratifiedKFold(n_splits=5))
GSCV_SVC.fit(X_train_enc, y_train_bf)

mask = GSCV_SVC.best_estimator_.named_steps['feat_select'].get_support()
selected_features = df_train_enc.columns[mask]

print("Best model: {}".format(GSCV_SVC.best_estimator_))
print("Selected features: {}".format(selected_features))
print("Best CV score: {:.2f}".format(GSCV_SVC.best_score_))
print("Test set score: {:.2f}".format(GSCV_SVC.score(X_test_enc, y_test)))
```

[12] ✓ 39.2s Python

```
Best model: Pipeline(steps=[('scale', StandardScaler()), ('feat_select', SelectKBest(k=2)),
                ('clf', SVC(C=10))])
Selected features: Index(['onehotencoder__haspool_yes', 'remainder__squaremeters'], dtype='object')
Best CV score: 0.93
Test set score: 0.94
```

```python
pipe_SVC = Pipeline(steps=[
    ('scale', MinMaxScaler()),
    ('feat_select', SelectPercentile()),
    ('clf', SVC())
])

param_grid_SVC = {
    'feat_select__percentile': [10, 20, 30],
    'clf__C': [0.1, 1, 10],
    'clf__kernel': ['linear', 'rbf']
}

GSCV_SVC2 = GridSearchCV(pipe_SVC, param_grid_SVC, cv=StratifiedKFold(n_splits=5))
GSCV_SVC2.fit(X_train_enc, y_train_bf)

mask = GSCV_SVC2.best_estimator_.named_steps['feat_select'].get_support()
selected_features = df_train_enc.columns[mask]

print("Best model: {}".format(GSCV_SVC2.best_estimator_))
print("Selected features: {}".format(selected_features))
print("Best CV score: {:.2f}".format(GSCV_SVC2.best_score_))
print("Test set score: {:.2f}".format(GSCV_SVC2.score(X_test_enc, y_test)))
```

[13] ✓ 44.5s Python

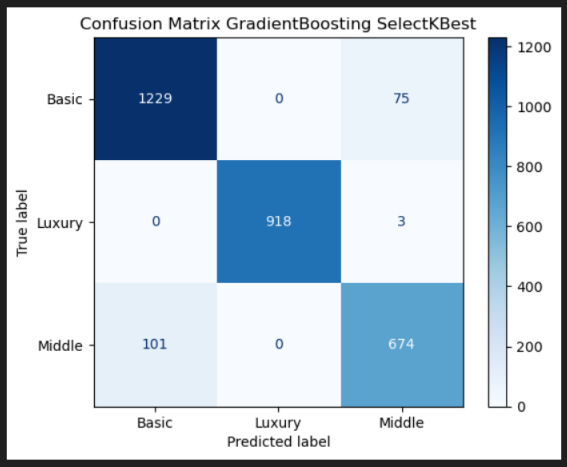
```
Best model: Pipeline(steps=[('scale', MinMaxScaler()),
                ('feat_select', SelectPercentile(percentile=30)),
                ('clf', SVC(C=10))])
Selected features: Index(['onehotencoder__hasyard_no', 'onehotencoder__hasyard_yes',
       'onehotencoder__haspool_no', 'onehotencoder__haspool_yes',
       'onehotencoder__isnewbuilt_new', 'remainder__squaremeters'],
      dtype='object')
Best CV score: 0.99
Test set score: 0.99
```

Evaluasi

```python
y_pred_GBC = GSCV_GBC.predict(X_test_enc)

print("Classification Report GradientBoosting SelectKBest:")
print(classification_report(y_test, y_pred_GBC))

cm_GBC = confusion_matrix(y_test, y_pred_GBC)
ConfusionMatrixDisplay(cm_GBC, display_labels=GSCV_GBC.classes_).plot(cmap='Blues')
plt.title("Confusion Matrix GradientBoosting SelectKBest")
plt.show()
```

[14] ✓ 0.2s Python

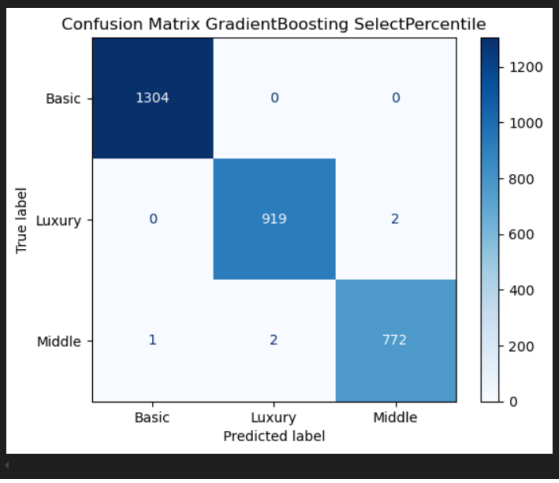
```
Classification Report GradientBoosting SelectKBest:
              precision    recall  f1-score   support

       Basic       0.92      0.94      0.93      1304
      Luxury       1.00      1.00      1.00       921
      Middle       0.90      0.87      0.88       775

    accuracy                           0.94      3000
   macro avg       0.94      0.94      0.94      3000
weighted avg       0.94      0.94      0.94      3000
```



```python
y_pred_GBC2 = GSCV_GBC2.predict(X_test_enc)

print("Classification Report GradientBoosting SelectPercentile:")
print(classification_report(y_test, y_pred_GBC2))

cm_GBC2 = confusion_matrix(y_test, y_pred_GBC2)
ConfusionMatrixDisplay(cm_GBC2, display_labels=GSCV_GBC2.classes_).plot(cmap='Blues')
plt.title("Confusion Matrix GradientBoosting SelectPercentile")
plt.show()
```

[15] ✓ 0.2s Python

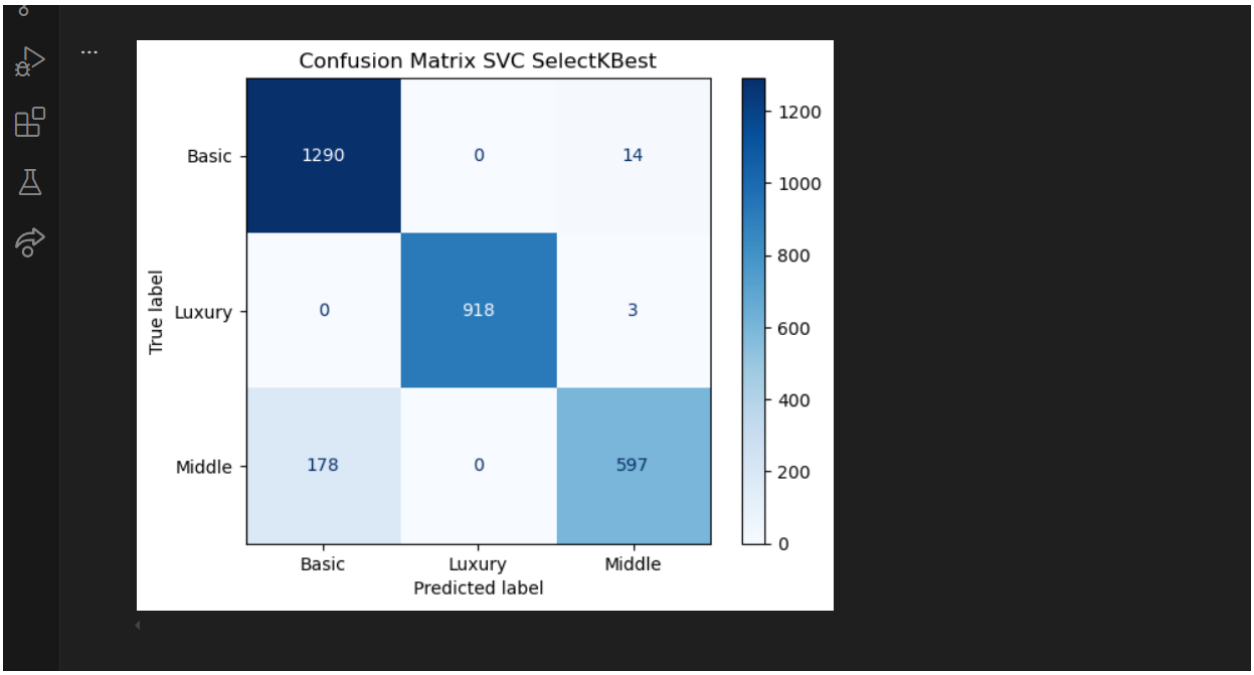
```
Classification Report GradientBoosting SelectPercentile:
              precision    recall  f1-score   support

       Basic       1.00      1.00      1.00      1304
      Luxury       1.00      1.00      1.00       921
      Middle       1.00      1.00      1.00       775

    accuracy                           1.00      3000
   macro avg       1.00      1.00      1.00      3000
weighted avg       1.00      1.00      1.00      3000
```




```python
y_pred_SVC = GSCV_SVC.predict(X_test_enc)

print("Classification Report SVC SelectKBest:")
print(classification_report(y_test, y_pred_SVC))

cm_SVC = confusion_matrix(y_test, y_pred_SVC)
ConfusionMatrixDisplay(cm_SVC, display_labels=GSCV_SVC.classes_).plot(cmap='Blues')
plt.title("Confusion Matrix SVC SelectKBest")
plt.show()
```

[16] ✓ 0.5s Python

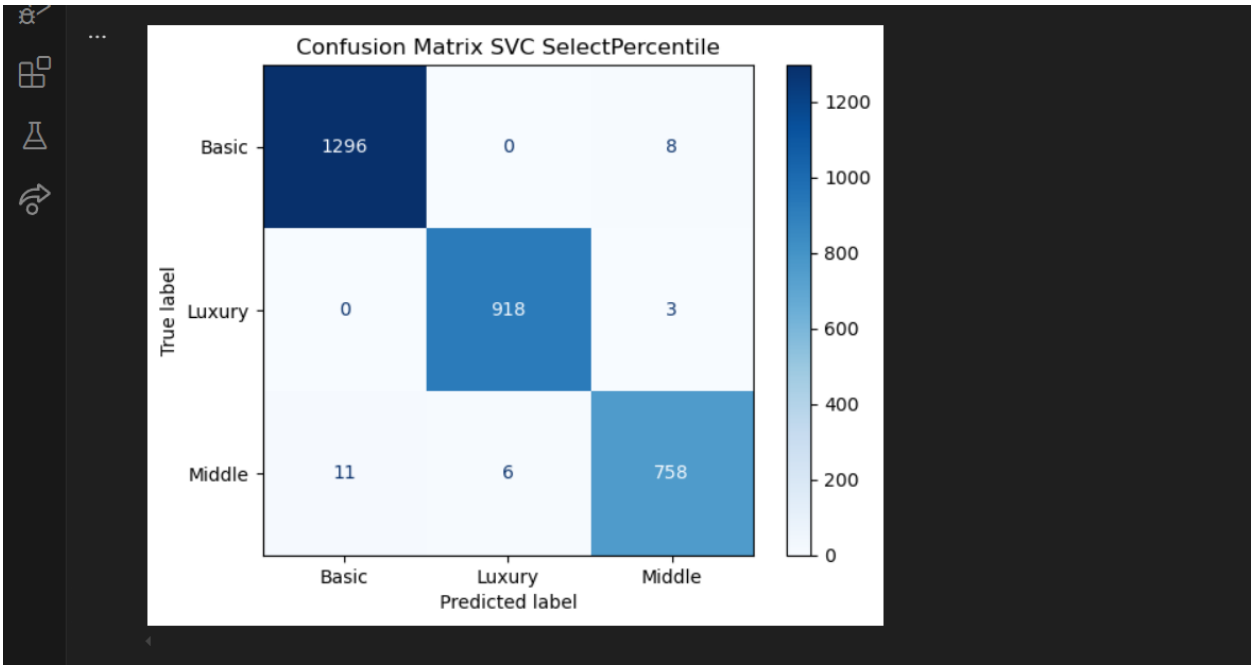
```
Classification Report SVC SelectKBest:
              precision    recall  f1-score   support

       Basic       0.88      0.99      0.93      1304
      Luxury       1.00      1.00      1.00       921
      Middle       0.97      0.77      0.86       775

    accuracy                           0.94      3000
   macro avg       0.95      0.92      0.93      3000
weighted avg       0.94      0.94      0.93      3000
```




```
y_pred_SVC2 = GSCV_SVC2.predict(X_test_enc)

print("Classification Report SVC SelectPercentile:")
print(classification_report(y_test, y_pred_SVC2))

cm_SVC2 = confusion_matrix(y_test, y_pred_SVC2)
ConfusionMatrixDisplay(cm_SVC2, display_labels=GSCV_SVC2.classes_).plot(cmap='Blues')
plt.title("Confusion Matrix SVC SelectPercentile")
plt.show()
```

[17] ✓ 0.4s Python

```
Classification Report SVC SelectPercentile:
              precision    recall  f1-score   support

       Basic       0.99      0.99      0.99      1304
      Luxury       0.99      1.00      1.00       921
      Middle       0.99      0.98      0.98       775

    accuracy                           0.99      3000
   macro avg       0.99      0.99      0.99      3000
weighted avg       0.99      0.99      0.99      3000
```



Kode Steamlit:

```python
import streamlit as st
import pickle
import os
import numpy as np
from streamlit_option_menu import option_menu

def load_model(model_file):
    model_path = os.path.join('BestModel_CLF_RF_SciPy.pkl')
    with open(model_path, 'rb') as f:
        return pickle.load(f)

with st.sidebar:
    selected = option_menu('Pilih Analisis',
                           ['Analisis Kategori Properti', 'Analisis Harga Properti'],
                           default_index=0)

rf_model = load_model('BestModel_CLF_RF_SciPy.pkl')

if selected == 'Analisis Kategori Properti':
    st.title("Klasifikasi Jenis Properti")
    st.write("Masukkan fitur properti untuk mengetahui kategori properti.")

    squaremeters = st.number_input("Luas Properti (m2)", min_value=0)
    numberofrooms = st.number_input("Jumlah Kamar Dalam Properti", min_value=0)
    hasyard = st.selectbox("Apakah Properti Memiliki Halaman?", ["yes", "no"])
    haspool = st.selectbox("Apakah Properti Memiliki Kolam Renang?", ["yes", "no"])
    floors = st.number_input("Jumlah Lantai Pada Properti", min_value=0)
    citycode = st.number_input("Kode Kota Properti", min_value=0)
    citypartrange = st.number_input("Rentang Wilayah Properti", min_value=0)
```

```python

    if selected == 'Analisis Kategori Properti':
        st.title("Klasifikasi Jenis Properti")
        st.write("Masukkan fitur properti untuk mengetahui kategori properti.")

        squaremeters = st.number_input("Luas Properti (m2)", min_value=0)
        numberofrooms = st.number_input("Jumlah Kamar Dalam Properti", min_value=0)
        hasyard = st.selectbox("Apakah Properti Memiliki Halaman?", ["yes", "no"])
        haspool = st.selectbox("Apakah Properti Memiliki Kolam Renang?", ["yes", "no"])
        floors = st.number_input("Jumlah Lantai Pada Properti", min_value=0)
        citycode = st.number_input("Kode Kota Properti", min_value=0)
        citypartrange = st.number_input("Rentang Wilayah Properti", min_value=0)
        numprevowners = st.number_input("Jumlah Pemilik Properti Sebelumnya", min_value=0)
        made = st.number_input("Tahun Pembuatan Properti", min_value=1900, max_value=2024)
        isnewbuilt = st.selectbox("Apakah Properti Baru / Lama?", ["yes", "no"])
        hasstormprotector = st.selectbox("Apakah Properti Memiliki Pelindung Badai?", ["yes", "no"])
        basement = st.number_input("Luas Basement Properti (m2)", min_value=0)
        attic = st.number_input("Luas Loteng Properti (m2)", min_value=0)
        garage = st.number_input("Kapasitas Garasi", min_value=0)
        hasstorageroom = st.selectbox("Apakah Properti Memiliki Ruang Penyimpanan?", ["yes", "no"])
        hasguestroom = st.number_input("Berapa Jumlah Kamar Tamu Di Properti?", min_value=0)

        angkaBinary = {"yes": 1, "no": 0}
        input_hasyard = angkaBinary[hasyard]
        input_haspool = angkaBinary[haspool]
        input_isnewbuilt = angkaBinary[isnewbuilt]
        input_hasstormprotector = angkaBinary[hasstormprotector]
        input_hasstorageroom = angkaBinary[hasstorageroom]

        input_data = np.array([[squaremeters, numberofrooms, input_hasyard, input_haspool, floors, citycode,
```

```
        angkaBinary = {"yes": 1, "no": 0}
        input_hasyard = angkaBinary[hasyard]
        input_haspool = angkaBinary[haspool]
        input_isnewbuilt = angkaBinary[isnewbuilt]
        input_hasstormprotector = angkaBinary[hasstormprotector]
        input_hasstorageroom = angkaBinary[hasstorageroom]

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

        if st.button("Lakukan Prediksi"):
            prediction = rf_model.predict(input_data)[0]
            kategori_map = {0: 'Basic', 1: 'Luxury', 2: 'Middle'}
            kategori_properti = kategori_map.get(prediction, "Tidak Diketahui")
            st.success(f"Kategori properti: *{kategori_properti}*")

    if selected == 'Analisis Harga Properti':
        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"])

```

```
65          Sex = st.radio("Gender", ["Female", "Male"])
66          st.write('Untuk beberapa kolom bisa menggunakan st.selectbox')
67          nama_kolom = st.selectbox("Nama Kolom", ["Under", "Normal", "Over"])
68
69          # Input untuk panjang dan lebar
70          st.write('Untuk inputan manual bisa menggunakan st.number_input')
71          panjang = st.number_input("Masukan Input", 0)
72          lebar = st.number_input("Masukan Nilai Lebar", 0)
73
74          alas = st.slider("Masukkan Nilai Alas", 0, 100)
75          tinggi = st.slider("Masukkan Nilai Tinggi", 0, 100)
76          st.write('Tombol button (Menggunakan st.button)')
77          hitung = st.button("Prediksi")
78
79          if hitung:
80              luas = 0.5 * alas * tinggi
81              st.write("Luas Segitiga Adalah", luas)
82
83      # Halaman ketentuan
84      if selected == 'Catatan':
85          st.title('Catatan')
86          st.write('''1. Untuk memunculkan sidebar agar tidak error ketika di run, silahkan install library streamlit
87          st.write('2. Menu yang dibuat ada 2 yaitu Klasifikasi dan Regresi.''')
88          st.write('3. Inputnya apa saja, sesuaikan dengan arsitektur code anda pada notebook.')
89          st.write('4. Referensi desain streamlit dapat di akses pada https://streamlit.io/')
90          st.write('5. Link streamlit design ini dapat di akses pada https://apputs-6qzfrvr4ufiyzhj84mrfkt7.streamlit.
91          st.write('''6. Library dan file requirements yang dibutuhkan untuk deploy online di github ada 5 yaitu strea
```