Random Forest dan Logistic Regression

IMPORT LIBRARY

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pathlib import Path

from sklearn.model_selection import train_test_split, GridSearchCV,
StratifiedKFold

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, MinMaxScaler,
OneHotEncoder
from sklearn.feature_selection import SelectKBest, chi2, f_classif,
SelectPercentile
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix,
ConfusionMatrixDisplay
import pickle
```

Data Loading

membaca data dari file eksternal & menampilkan contoh isi data

```
df_gempa = pd.read_csv('earthquake_dataset.csv',header=0)

df_gempa.head()
```

	magnitude	depth	cdi	mmi	sig	alert
0	7.0	14.0	8.0	7.0	0.0	green
1	6.9	25.0	4.0	4.0	-33.0	green
2	7.0	579.0	3.0	3.0	-13.0	green
3	7.3	37.0	5.0	5.0	65.0	green
4	6.6	624.0	0.0	2.0	-98.0	green

menampilkan informasi mengenai masing-masing kolom (info) & menampilkan deskripsi terkait gambaran isi data (describe)

```
print("Jumlah baris, kolom:", df_gempa.shape)
print("\nTipe data:")
print(df_gempa.dtypes)

df_gempa.describe()
```

```
Jumlah baris, kolom: (1300, 6)

Tipe data:
magnitude    float64
depth        float64
cdi          float64
mmi          float64
sig          float64
alert         object
dtype: object
```

	magnitude	depth	cdi	mmi	sig
count	1300.000000	1300.000000	1300.000000	1300.000000	1300.000000
mean	7.061008	50.321538	6.976154	6.939231	-9.749231
std	0.423738	104.348286	2.244875	1.354774	62.245214
min	6.500000	2.000000	0.000000	1.000000	-128.000000
25%	6.700000	12.000000	6.000000	6.000000	-54.000000
50%	7.000000	19.000000	8.000000	7.000000	-7.000000
75%	7.300000	38.000000	8.000000	8.000000	31.000000
max	8.600000	670.000000	9.000000	9.000000	127.000000

Data checking meliputi proses pengecekan apakah ada data yang:

null & NaN

```
print("Jumlah nilai kosong per kolom:\n", df_gempa.isnull().sum())
```

```
Jumlah nilai kosong per kolom:
 magnitude    0
depth        0
cdi          0
mmi          0
```

```
sig          0
alert        0
dtype: int64
```

duplikat

```
print("Jumlah data duplikat:", df_gempa.duplicated().sum())
```

```
Jumlah data duplikat: 44
```

kosong

```
for c in df_gempa.select_dtypes(include=['object']).columns:
    n_empty = (df_gempa[c].astype(str).str.strip() == '').sum()
    print(f"data kosong di {c}: {n_empty}")
```

```
data kosong di alert: 0
```

outlier

```
# Quick outlier checks using IQR for numeric columns
num_cols =
df_gempa.select_dtypes(include=[np.number]).columns.tolist()
print('\nKolom Numerik:', num_cols)

outlier_summary = {}
for col in num_cols:
    Q1 = df_gempa[col].quantile(0.25)
    Q3 = df_gempa[col].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    outliers = df_gempa[(df_gempa[col] < lower) | (df_gempa[col] >
upper)].shape[0]
    outlier_summary[col] = outliers
print('\nTotal Outlier per kolom yang numerik:')
print(outlier_summary)
```

```
Kolom Numerik: ['magnitude', 'depth', 'cdi', 'mmi', 'sig']

Total Outlier per kolom yang numerik:
{'magnitude': 13, 'depth': 168, 'cdi': 88, 'mmi': 5, 'sig': 0}
```

Data Preparation

penanganan data yang null dan NaN

```
Karena tidak ada yang Null, maka tahap ini dilewati
```

penanganan data yang kosong

```
Karena tidak ada yang kosong, maka tahap ini dilewati
```

penanganan data yang duplikat

```
# Remove duplicate rows
df_gempa2 = df_gempa.copy()
df_gempa2 = df_gempa2.drop_duplicates()

print("Jumlah data duplikat:", df_gempa2.duplicated().sum())
```

```
Jumlah data duplikat: 0
```

penanganan data yang outlier

```
FEATURES = ['magnitude', 'depth', 'cdi', 'mmi', 'sig']
TARGET = 'alert'

for c in FEATURES:
    low = df_gempa2[c].quantile(0.01)
    high = df_gempa2[c].quantile(0.99)
    df_gempa2[c] = df_gempa2[c].clip(lower=low, upper=high)
```

Mengubah Alert(Object) Menjadi Numerik / Data Encoding

```
label_mapping = {
    'green': 0,
    'yellow': 1,
    'orange': 2,
    'red': 3
}
```

```
df_gempa2['alert'] = df_gempa2['alert'].map(label_mapping)
print('\nLabel mapping:', label_mapping)


Label mapping: {'green': 0, 'yellow': 1, 'orange': 2, 'red': 3}
```

Pembagian Data Train Train/TestSplit

FEATURES = ['magnitude', 'depth', 'cdi', 'mmi', 'sig']

TARGET = 'alert'

```
X = df_gempa2[FEATURES]
y = df_gempa2[TARGET]


X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.4, random_state=86, stratify=y
)


print('\nDistribus Pembagian Train class:')
print(y_train.value_counts(normalize=True))
print('\nDistribus Pembagian Test class:')
print(y_test.value_counts(normalize=True))

print("\nUkuran X_train, X_test:", X_train.shape, X_test.shape)


Distribus Pembagian Train class:
alert
0    0.256308
1    0.253652
2    0.250996
3    0.239044
Name: proportion, dtype: float64

Distribus Pembagian Test class:
alert
0    0.256461
1    0.252485
2    0.252485
3    0.238569
Name: proportion, dtype: float64

Ukuran X_train, X_test: (753, 5) (503, 5)
```

Feature selection

```
n_features = len(FEATURES)
k_options = list(range(1, n_features + 1)) # [1..n_features]
percentile_options = [10, 25, 50, 75, 100]


print('\nFeature selection param choices:')
print('k_options =', k_options)
print('percentile_options =', percentile_options)
```

```
Feature selection param choices:
k_options = [1, 2, 3, 4, 5]
percentile_options = [10, 25, 50, 75, 100]
```

Pipeline

```
selector_k = SelectKBest(score_func=f_classif)
selector_pct = SelectPercentile(score_func=f_classif)
```

Random Forest Pipeline

```
from sklearn.impute import SimpleImputer

num_imputer = SimpleImputer(strategy='median')
selector_k = SelectKBest(score_func=f_classif, k='all')

# Pipeline
pipe_rf = Pipeline([
    ('imputer', num_imputer),
    ('selector', selector_k),
    ('clf', RandomForestClassifier(random_state=86))
])
```

Logistic Regression Pipeline

```
pipe_lr = Pipeline([
    ('imputer', num_imputer),
    ('scaler', StandardScaler()),
    ('selector', selector_k),
    ('clf', LogisticRegression(max_iter=500, random_state=86))
])
```

Paramater Grids

```
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Parameter grids (random forest)
param_grid_rf = [
    {
        'selector': [SelectKBest(score_func=f_classif)],
        'selector__k': k_options,
        'clf__n_estimators': [100, 200],
        'clf__max_depth': [None, 5, 10],
        'clf__class_weight': [None, 'balanced']
    },
    {
        'selector': [SelectPercentile(score_func=f_classif)],
        'selector__percentile': [10, 25, 50, 75, 100],
        'clf__n_estimators': [100, 200],
        'clf__max_depth': [None, 5, 10],
        'clf__class_weight': [None, 'balanced']
    }
]

# Parameter grids (logistic regression)
param_grid_lr = [
    {
        'scaler': [StandardScaler(), MinMaxScaler()],
        'selector': [SelectKBest(score_func=f_classif)],
        'selector__k': k_options,
        'clf__C': [0.01, 0.1, 1, 10],
        'clf__penalty': ['l2'],
        'clf__solver': ['lbfgs']
    },
    {
        'scaler': [StandardScaler(), MinMaxScaler()],
        'selector': [SelectPercentile(score_func=f_classif)],
        'selector__percentile': [10, 25, 50, 75, 100],
        'clf__C': [0.01, 0.1, 1, 10],
        'clf__penalty': ['l2'],
        'clf__solver': ['lbfgs']
    }
]
```

GridSearchCV

```
#Random Forest
grid_rf = GridSearchCV(
    estimator=pipe_rf,
    param_grid=param_grid_rf,
```

```
    scoring='f1_macro',
    n_jobs=-1,
    cv=cv,
    verbose=1,
    return_train_score=False
)

#Logistic Regression
grid_lr = GridSearchCV(
    estimator=pipe_lr,
    param_grid=param_grid_lr,
    scoring='f1_macro',
    n_jobs=-1,
    cv=cv,
    verbose=1,
    return_train_score=False
)

import time
print("Menjalankan Grid Search untuk Random Forest...")
start = time.time()
grid_rf.fit(X_train, y_train)
print(f"GridSearch Random Forest selesai dalam {time.time() -
start:.2f} detik")

print("\nMenjalankan Grid Search untuk Logistic Regression...")
grid_lr.fit(X_train, y_train)
print(f"GridSearch Logistic Regression selesai dalam {time.time() -
start:.2f} detik")
```

```
Menjalankan Grid Search untuk Random Forest...
Fitting 5 folds for each of 120 candidates, totalling 600 fits
GridSearch Random Forest selesai dalam 5.86 detik

Menjalankan Grid Search untuk Logistic Regression...
Fitting 5 folds for each of 80 candidates, totalling 400 fits
GridSearch Logistic Regression selesai dalam 6.30 detik
```

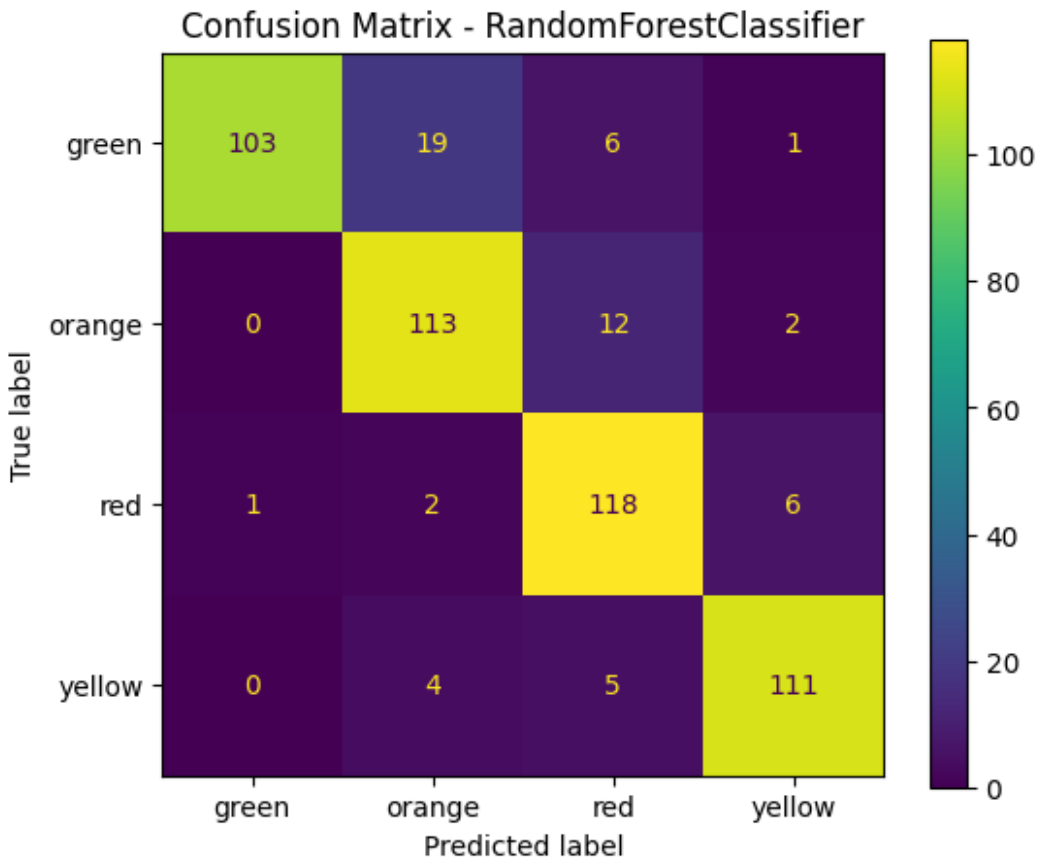
pengecekan fitur yang signifikan + Visualisasi hasil + confusion matrix

```
# #RF
# print("CV Score (F1) terbaik:", grid_rf.best_score_)
# print("Kombinasi model terbaik:", grid_rf.best_estimator_)

# rf_test_score = grid_rf.best_estimator_.score(X_test, y_test)
# print("\nSkor Test (akurasi) Random Forest:", rf_test_score)
```

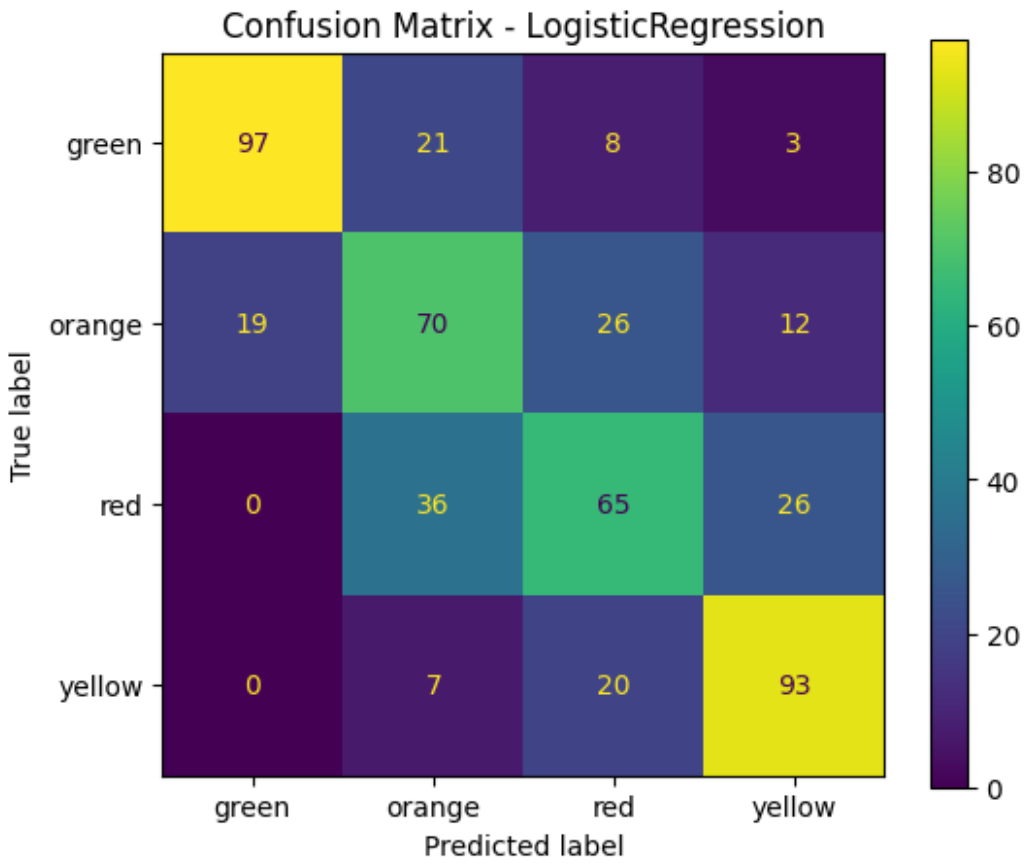
```
# selector = grid_rf.best_estimator_.named_steps['selector']
# if hasattr(selector, 'get_support'):
#     mask = selector.get_support()
#     selected = np.array(X.columns)[mask]
#     print("\nFitur terbaik (terpilih):", selected)

# rf_pred = grid_rf.predict(X_test)
# cm_rf = confusion_matrix(y_test, rf_pred)
# disp_rf = ConfusionMatrixDisplay(confusion_matrix=cm_rf,
display_labels=['0 = green','1 = yellow', '2 = orange', '3 = red'])
# disp_rf.plot(cmap=plt.cm.Greens)
# plt.title("Confusion Matrix – Random Forest")
# plt.show()

# print("\nClassification Report – Random Forest:\n",
classification_report(y_test, rf_pred))

# # Logistic Regression

# print("CV Score (F1) terbaik:", grid_lr.best_score_)
# print("Kombinasi model terbaik:", grid_lr.best_estimator_)

# lr_test_score = grid_lr.best_estimator_.score(X_test, y_test)
# print("\nSkor Test (akurasi) Logistic Regression:", lr_test_score)

# selector_lr = grid_lr.best_estimator_.named_steps['selector']
# if hasattr(selector_lr, 'get_support'):
#     mask_lr = selector_lr.get_support()
#     selected_lr = np.array(X.columns)[mask_lr]
#     print("\nFitur terbaik (terpilih):", selected_lr)

# lr_pred = grid_lr.predict(X_test)
# cm_lr = confusion_matrix(y_test, lr_pred)
# disp_lr = ConfusionMatrixDisplay(confusion_matrix=cm_lr,
display_labels=['0 = green','1 = yellow', '2 = orange', '3 = red'])
# disp_lr.plot(cmap=plt.cm.Blues)
# plt.title("Confusion Matrix – Logistic Regression")
# plt.show()

# print("\nClassification Report – Logistic Regression:\n",
classification_report(y_test, lr_pred))

def evaluate_model(grid_search, X_test, y_test, feature_names):
    best = grid_search.best_estimator_
    y_pred = best.predict(X_test)
    print('\n--- Model evaluation:',
type(best.named_steps['clf']).__name__)
    print('\nClassification Report:')
    print(classification_report(y_test, y_pred, target_names=[str(k)
for k in sorted(label_mapping.keys())]))
```

```
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=[str(k) for k in sorted(label_mapping.keys())])
    fig, ax = plt.subplots(figsize=(6,5))
    disp.plot(ax=ax)
    ax.set_title(f'Confusion Matrix -
{type(best.named_steps["clf"]).__name__}')
    plt.show()


    # Extract selected features
    selector = best.named_steps.get('selector')
    if selector is not None:
        # selector may be SelectKBest or SelectPercentile
        mask = selector.get_support()
        selected = [f for f, m in zip(feature_names, mask) if m]
        print('\nSelected features:', selected)
    else:
        print('\nNo selector found in pipeline.')


# Evaluasi RF terbaik
evaluate_model(grid_rf, X_test, y_test, FEATURES)


# Evaluasi LR terbaik
evaluate_model(grid_lr, X_test, y_test, FEATURES)
```

```
--- Model evaluation: RandomForestClassifier

Classification Report:
              precision    recall  f1-score   support

       green       0.99      0.80      0.88       129
      orange       0.82      0.89      0.85       127
         red       0.84      0.93      0.88       127
      yellow       0.93      0.93      0.93       120

    accuracy                           0.88       503
   macro avg       0.89      0.89      0.89       503
weighted avg       0.89      0.88      0.89       503
```




```
Selected features: ['magnitude', 'depth', 'cdi', 'mmi', 'sig']

--- Model evaluation: LogisticRegression

Classification Report:
              precision    recall  f1-score   support

       green       0.84      0.75      0.79       129
      orange       0.52      0.55      0.54       127
         red       0.55      0.51      0.53       127
      yellow       0.69      0.78      0.73       120

    accuracy                           0.65       503
   macro avg       0.65      0.65      0.65       503
weighted avg       0.65      0.65      0.65       503
```




```
Selected features: ['magnitude', 'depth', 'cdi', 'mmi', 'sig']
```

pengecekan model terbaik

```
#RF
print('RF terbaik:')
print(grid_rf.best_params_)
print('Best RF cross-val score (f1_macro):', grid_rf.best_score_)

#LR
print('\nLR terbaik:')
print(grid_lr.best_params_)
print('Best LR cross-val score (f1_macro):', grid_lr.best_score_)
```

```
RF terbaik:
{'clf__class_weight': 'balanced', 'clf__max_depth': None,
'clf__n_estimators': 100, 'selector': SelectKBest(), 'selector__k': 5}
Best RF cross-val score (f1_macro): 0.8869651336938297

LR terbaik:
```

```
{'clf__C': 10, 'clf__penalty': 'l2', 'clf__solver': 'lbfgs', 'scaler': StandardScaler(), 'selector': SelectKBest(), 'selector__k': 5}
Best LR cross-val score (f1_macro): 0.6535978846771364
```

```
import pickle


best_rf_model = grid_rf.best_estimator_.named_steps['clf']

with open("best_random_forest.pkl", "wb") as f:
        pickle.dump(best_rf_model, f)

print("□ Model Random Forest yang terbaik berhasil disimpan ke 'best_random_forest.pkl'")
```

```
□ Model Random Forest yang terbaik berhasil disimpan ke 'best_random_forest.pkl'
```

Gradient Boosting Classifier dan Support Vector Machine

IMPORT LIBRARY

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold, GridSearchCV

from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import SelectKBest, SelectPercentile

from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest
from sklearn.ensemble import GradientBoostingClassifier

import time

from sklearn.metrics import (
    confusion_matrix,
    ConfusionMatrixDisplay,
    classification_report
)
```

Loading Data...

```
df_gempa = pd.read_csv('earthquake_dataset.csv',header=0) #membaca
data dan simpan pada df_gempa

df_gempa.head()
```

	magnitude	depth	cdi	mmi	sig	alert
0	7.0	14.0	8.0	7.0	0.0	green
1	6.9	25.0	4.0	4.0	-33.0	green
2	7.0	579.0	3.0	3.0	-13.0	green
3	7.3	37.0	5.0	5.0	65.0	green
4	6.6	624.0	0.0	2.0	-98.0	green

Membaca tipedata dan deskripsi data

```
print("Jumlah baris, kolom:", df_gempa.shape)
print("\nTipe data:")
print(df_gempa.dtypes)

df_gempa.describe()
```

```
Jumlah baris, kolom: (1300, 6)

Tipe data:
magnitude    float64
depth        float64
cdi          float64
mmi          float64
sig          float64
alert         object
dtype: object
```

	magnitude	depth	cdi	mmi	sig
count	1300.000000	1300.000000	1300.000000	1300.000000	1300.000000
mean	7.061008	50.321538	6.976154	6.939231	-9.749231
std	0.423738	104.348286	2.244875	1.354774	62.245214
min	6.500000	2.000000	0.000000	1.000000	-128.000000
25%	6.700000	12.000000	6.000000	6.000000	-54.000000
50%	7.000000	19.000000	8.000000	7.000000	-7.000000
75%	7.300000	38.000000	8.000000	8.000000	31.000000
max	8.600000	670.000000	9.000000	9.000000	127.000000

Mengubah Alert(Object) Menjadi Numerik

```
df_gempa['alert'].unique() #melakukan pengecekan isi alert ada apa
saja
```

```
array(['green', 'yellow', 'orange', 'red'], dtype=object)
```

```
mapping = {
    'green': 0,
    'yellow': 1,
    'orange': 2,
    'red': 3
}  #melakukan mapping isi dari array menjadi numerik
df_gempa2 = df_gempa
df_gempa2['alert'] = df_gempa2['alert'].map(mapping)

df_gempa2.head(20) #hasil dari konversi
```

	magnitude	depth	cdi	mmi	sig	alert
0	7.0	14.0	8.0	7.0	0.0	0

```
1          6.9   25.0  4.0  4.0  -33.0      0
2          7.0  579.0  3.0  3.0  -13.0      0
3          7.3   37.0  5.0  5.0   65.0      0
4          6.6  624.0  0.0  2.0  -98.0      0
5          7.0  660.0  4.0  3.0  -13.0      0
6          6.8  630.0  1.0  3.0  -57.0      0
7          6.7   20.0  7.0  6.0   29.0      0
8          6.8   20.0  8.0  7.0 -101.0      1
9          7.6   26.0  9.0  8.0    7.0      1
10         6.9   10.0  9.0  9.0  119.0      1
11         6.5   10.0  7.0  7.0  -12.0      0
12         7.0  137.0  7.0  5.0   -7.0      0
13         7.6  116.0  8.0  8.0  -59.0      1
14         6.6   12.0  9.0  8.0   19.0      2
15         6.6   30.0  7.0  6.0  -96.0      0
16         7.0   33.0  9.0  8.0   71.0      1
17         6.5  622.0  3.0  2.0 -115.0      0
18         7.2  236.0  7.0  5.0  108.0      0
19         6.9   10.0  2.0  5.0  -35.0      0
```

Pembersihan Data : pengecekan Data NULL & Pembersihan Data Duplikat

```
print("Jumlah nilai kosong per kolom:\n", df_gempa2.isnull().sum())
print("Jumlah data duplikat:", df_gempa2.duplicated().sum())
```

```
Jumlah nilai kosong per kolom:
 magnitude    0
depth        0
cdi          0
mmi          0
sig          0
alert        0
dtype: int64
Jumlah data duplikat: 44
```

```
    magnitude  depth  cdi  mmi    sig  alert
0          7.0   14.0  8.0  7.0    0.0      0
1          6.9   25.0  4.0  4.0  -33.0      0
2          7.0  579.0  3.0  3.0  -13.0      0
3          7.3   37.0  5.0  5.0   65.0      0
4          6.6  624.0  0.0  2.0  -98.0      0
5          7.0  660.0  4.0  3.0  -13.0      0
6          6.8  630.0  1.0  3.0  -57.0      0
7          6.7   20.0  7.0  6.0   29.0      0
8          6.8   20.0  8.0  7.0 -101.0      1
9          7.6   26.0  9.0  8.0    7.0      1
10         6.9   10.0  9.0  9.0  119.0      1
11         6.5   10.0  7.0  7.0  -12.0      0
```

```
12         7.0  137.0  7.0  5.0   -7.0      0
13         7.6  116.0  8.0  8.0  -59.0      1
14         6.6   12.0  9.0  8.0   19.0      2
15         6.6   30.0  7.0  6.0  -96.0      0
16         7.0   33.0  9.0  8.0   71.0      1
17         6.5  622.0  3.0  2.0 -115.0      0
18         7.2  236.0  7.0  5.0  108.0      0
19         6.9   10.0  2.0  5.0  -35.0      0
```

```
df_gempa3 = df_gempa2 #alasan bikin sampe gempa3 itu buat mencegah ada korup data sehingga perlu run dari awal jika korup
df_gempa3 = df_gempa3.drop_duplicates() #penghapusan data duplikat

print("Jumlah data duplikat:", df_gempa3.duplicated().sum()) #cek data duplikat

X = df_gempa3.drop(columns=['alert'])
y = df_gempa3['alert']
```

```
Jumlah data duplikat: 0
```

Pembagian Data Train Train/TestSplit

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.4,
    random_state=86,   # NPM terbesar
    stratify=y
)

print("Ukuran X_train, X_test:", X_train.shape, X_test.shape)
```

```
Ukuran X_train, X_test: (753, 5) (503, 5)
```

Membangun Model Gradient Boosting Classifier dengan Pipeline

```
pipe_GBS = Pipeline(steps=[
    ('feat_select', SelectKBest()),    # variabel default hanya buat pengisi, sudah diurus di param
    ('clf', GradientBoostingClassifier(
        n_estimators=100,
        learning_rate=0.2,
        max_depth=3,
        random_state=86
    ))
])
```

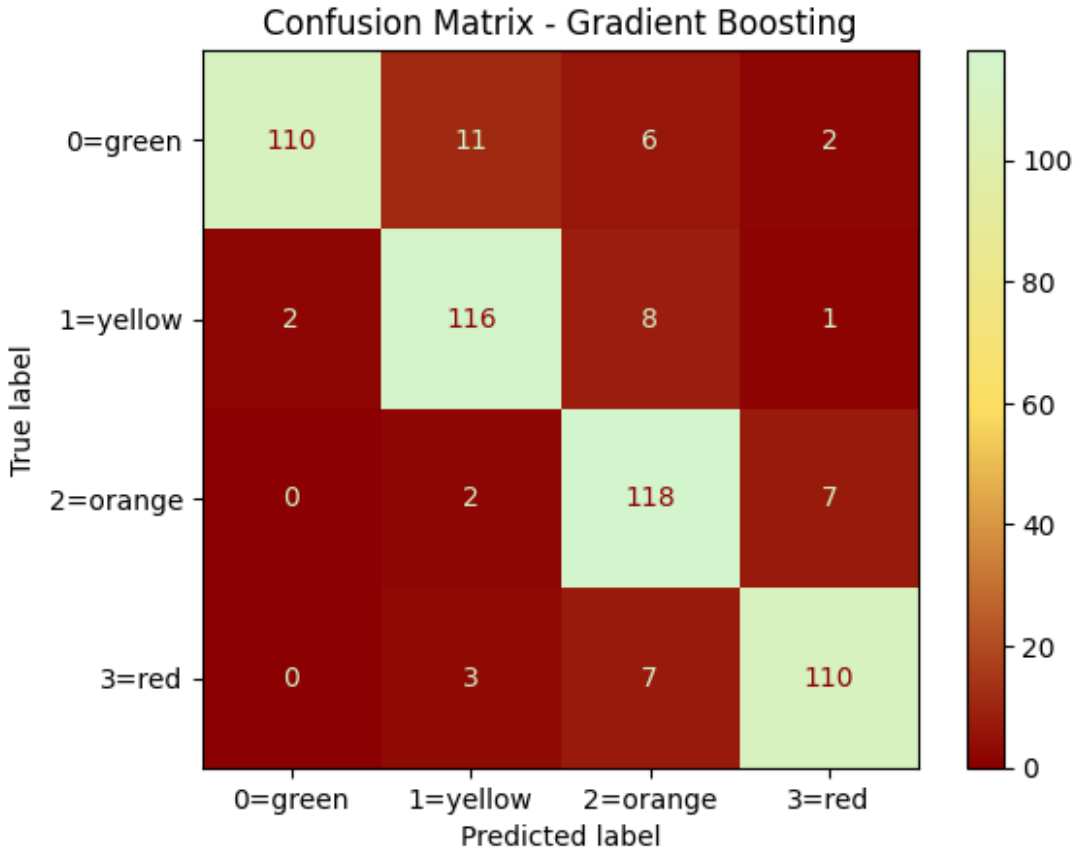
Pembuatan Params dan juga GridSearch

```
SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

param_grid_gb = [
    { #bandingkan dengan selectKBest
        'feat_select__k': np.arange(1, 6),
        'clf__n_estimators': [100],
        'clf__max_depth': [3, 5, 10],
        'clf__min_samples_split': [2, 5]
    },
    { #bandingkan dengan select percentil
        'feat_select': [SelectPercentile()],
        'feat_select__percentile': [50, 70, 90],
        'clf__n_estimators': [100],
        'clf__max_depth': [3, 5, 10],
        'clf__min_samples_split': [2, 5]
    }
]

gscv_gb = GridSearchCV(
    pipe_GBS,
    param_grid_gb,
    cv=SKF,
    scoring='f1_macro',  # untuk bisa menghitung data dengan lebih
dari satu kelas karena kalau tidak begini maka ada error hasilnya
infinite
    verbose=1,
    n_jobs=-1
)

gscv_gb.fit(X_train, y_train)

print("Best params (Gradient Boosting):", gscv_gb.best_params_) #
Hasil dari model dengan Params terbagus
print("Best score (Gradient Boosting):", gscv_gb.best_score_) # Hasil
dari model dengan Score terbagius
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits
Best params (Gradient Boosting): {'clf__max_depth': 5,
'clf__min_samples_split': 2, 'clf__n_estimators': 100,
'feat_select__k': np.int64(5)}
Best score (Gradient Boosting): 0.8741100360485264
```

```
from matplotlib.colors import LinearSegmentedColormap

colors = ["#8B0000", "#fcde62", "#d2f5ce"] # merah tua → kuning →
hijau muda
```

```
custom_cmap = LinearSegmentedColormap.from_list("red_to_green",
colors, N=256)

y_pred_gb = gscv_gb.best_estimator_.predict(X_test)

# Confusion Matrix dan Classification Report
cm_gb = confusion_matrix(y_test, y_pred_gb)
disp_gb = ConfusionMatrixDisplay(confusion_matrix=cm_gb,

display_labels=['0=green','1=yellow','2=orange','3=red'])
disp_gb.plot(cmap=custom_cmap)
plt.title("Confusion Matrix - Gradient Boosting")
plt.show()

print("\nGradient Boosting Report:")
print(classification_report(y_test, y_pred_gb))
```




```
Gradient Boosting Report:
              precision    recall  f1-score   support

           0       0.98      0.85      0.91       129
           1       0.88      0.91      0.90       127
```

```
           2       0.85      0.93      0.89       127
           3       0.92      0.92      0.92       120

    accuracy                           0.90       503
   macro avg       0.91      0.90      0.90       503
weighted avg       0.91      0.90      0.90       503
```

Membangun Support Vector Machine Dengan Pipeline

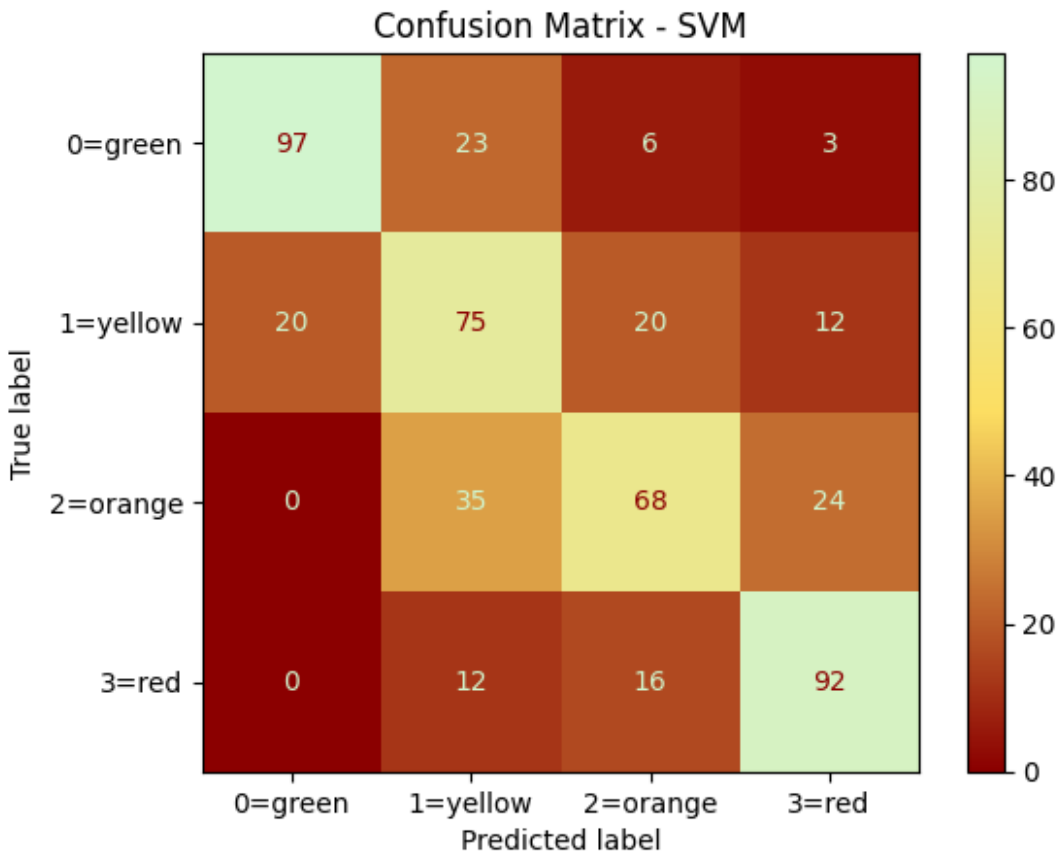
```
pipe_svm = Pipeline(steps=[
    ('scaler', StandardScaler()), # variabel default hanya buat
pengisi, sudah diurus di param
    ('feat_select', SelectKBest()), # variabel default hanya buat
pengisi, sudah diurus di param
    ('clf', SVC(
        C=1.0,
        kernel='linear',
        random_state=86,
    ))
])

params_grid_svm = [
    {
        'scaler': [StandardScaler(), MinMaxScaler()], # langsung
melakukan dengan 2 scaler yang lalu nanti dibandingkan secara
bersamaan
        'feat_select__k': np.arange(2, 10), # mencari k terbaik
        'clf__C': [0.001, 0.01, 0.1, 1, 10, 100], # mencari parameter
C terbaik
    },
    {
        'scaler': [StandardScaler(), MinMaxScaler()], # langsung
melakukan dengan 2 scaler yang lalu nanti dibandingkan secara
bersamaan
        'feat_select': [SelectPercentile()],
        'feat_select__percentile': np.arange(20, 80, 10), # mencari
persentil terbaik
        'clf__C': [0.001, 0.01, 0.1, 1, 10, 100], # mencari parameter
C terbaik
    }
]

SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

gscv_svm = GridSearchCV(
    pipe_svm,
    params_grid_svm,
    cv=SKF,
```

```
    scoring='f1_macro', # untuk bisa menghitung data dengan lebih dari satu kelas karena kalau tidak begini maka ada error hasilnya infinite
    verbose=1,
    n_jobs=-1
)

print("Menjalankan GridSearch untuk SVM...")
start = time.time()
gscv_svm.fit(X_train, y_train)
print(f"GridSearch SVM selesai dalam {time.time() - start:.2f} detik")
```

```
Menjalankan GridSearch untuk SVM...
Fitting 5 folds for each of 168 candidates, totalling 840 fits
GridSearch SVM selesai dalam 4.68 detik
```

Evaluasi model Support Vector Machine

```
colors = ["#8B0000", "#fcde62", "#d2f5ce"] # merah tua → kuning → hijau muda
custom_cmap = LinearSegmentedColormap.from_list("red_to_green", colors, N=256)

y_pred_svm = gscv_svm.best_estimator_.predict(X_test)

# Confusion Matrix dan Classification Report
cm_svm = confusion_matrix(y_test, y_pred_svm)
disp_svm = ConfusionMatrixDisplay(confusion_matrix=cm_svm,

display_labels=['0=green','1=yellow','2=orange','3=red'])
disp_svm.plot(cmap=custom_cmap)
plt.title("Confusion Matrix - SVM")
plt.show()

print("\nSVM Report:")
print(classification_report(y_test, y_pred_svm))
```

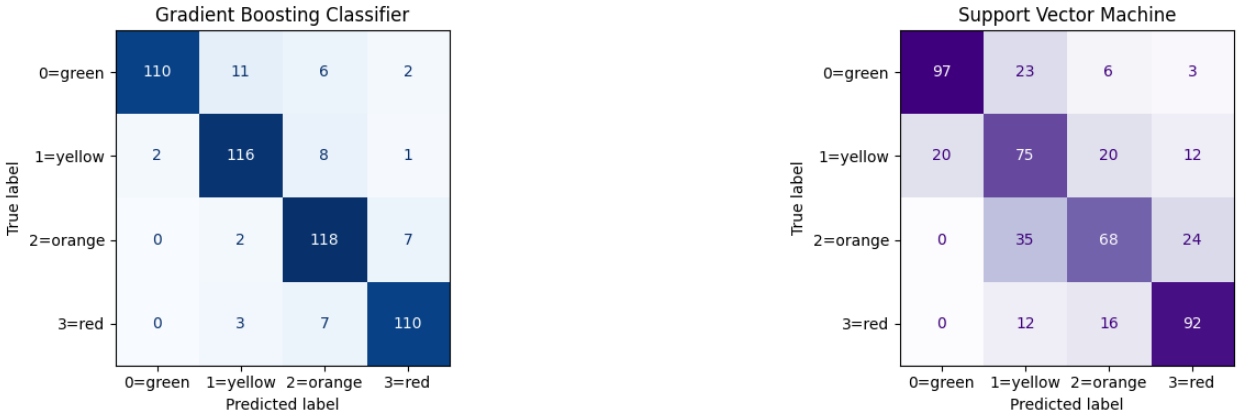



```
SVM Report:
              precision    recall  f1-score   support

           0       0.83      0.75      0.79       129
           1       0.52      0.59      0.55       127
           2       0.62      0.54      0.57       127
           3       0.70      0.77      0.73       120

    accuracy                           0.66       503
   macro avg       0.67      0.66      0.66       503
weighted avg       0.67      0.66      0.66       503
```

Visualisasi Perbandingan Dua Model Terbaik

```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(16, 4))

disp_gb.plot(ax=ax1, cmap=plt.cm.Blues, colorbar=False)
ax1.set_title("Gradient Boosting Classifier")


disp_svm.plot(ax=ax2, cmap=plt.cm.Purples, colorbar=False)
```

```
ax2.set_title("Support Vector Machine")


plt.tight_layout()
plt.show()  # tampilkan semua plot
```



Dapat disimpulkan dari grafik perbandingan diatas antara Algoritma Gradient Boosting Classifier dengan Support Vector Machine, Gradient Boosting Classifier menjadi algoritma yang memiliki akurasi lebih tinggi dibandingkan dengan SVM Linear.

Karena Gradient Boosting menjadi algoritma dengan accuracy terbaik. Kami menjadikan Gradient Boosting sebagai algoritma model kami. Berikut adalah cell untuk export model Gradient Boosting dengan parameter terbaik.

```
import pickle


best_gb_model = gscv_gb.best_estimator_.named_steps['clf']

with open("best_gradient_boosting.pkl", "wb") as f:
        pickle.dump(best_gb_model, f)


print("□ Model gradient boosting terbaik berhasil disimpan ke
'best_gradient_boosting.pkl'")
```

```
□ Model gradient boosting terbaik berhasil disimpan ke
'best_gradient_boosting.pkl'
```

```python
#STREAMLIT

import streamlit as st
import pandas as pd
import pickle

@st.cache_resource
def load_model():
    with open("BestModel_CLF_RandomForest_Kelompokan.pkl", "rb") as f:
        model = pickle.load(f)
    return model

rf_model = load_model()

st.set_page_config(page_title="Earthquake Alert Prediction", layout="centered")
st.title("🌋 Earthquake Alert Prediction")

st.markdown("""
Aplikasi ini memprediksi **warna peringatan (alert)** dari suatu gempa bumi
menggunakan model Machine Learning terbaik: **Random Forest Classifier**.
""")

st.markdown("""
### Penjelasan Fitur Input
| Nama Fitur | Arti | Penjelasan Sederhana |
|-------------|------|----------------------|
| **Magnitude** | Kekuatan gempa (skala Richter) | Semakin besar nilainya, semakin kuat gempa. |
| **Depth (km)** | Kedalaman pusat gempa | Gempa dangkal (<70 km) biasanya lebih terasa di permukaan. |
| **CDI** | Community Determined Intensity | Seberapa kuat guncangan dirasakan oleh masyarakat (1-10). |
| **MMI** | Modified Mercalli Intensity | Skala intensitas fisik gempa terhadap bangunan dan lingkungan (1-10). |
| **SIG** | Significance | Nilai positif = signifikan, negatif = ringan. |
""")

st.markdown("---")

alert_labels = {
    0: "🟢 Green - Dampak sangat kecil atau tidak signifikan",
    1: "🟡 Yellow - Dampak sedang, potensi kerusakan kecil",
    2: "🟠 Orange - Dampak signifikan, kemungkinan kerusakan sedang-besar",
    3: "🔴 Red - Dampak parah, kerusakan besar dan potensi korban tinggi"
}

alert_colors = {
    0: "#00C853",  # green
    1: "#FFD600",  # yellow
    2: "#FF6D00",  # orange
    3: "#D50000",  # red
}

def highlight_alert(model_name, value):
    label = alert_labels.get(value, "Tidak diketahui")
    color = alert_colors.get(value, "#FFFFFF")
    html = f"""
    <div style="background-color:{color}; padding:15px; border-radius:10px; text-align:center; color:white; font-weight:bold; font-size:18px;">
        {model_name} Prediction: {label}
    </div>
    """
    st.markdown(html, unsafe_allow_html=True)

st.subheader("                Masukkan Data Gempa")

col1, col2 = st.columns(2)
with col1:
    magnitude = st.number_input("Magnitude", 0.0, 10.0, 6.5)
    cdi = st.number_input("CDI", 0.0, 10.0, 5.0)
    sig = st.number_input("SIG", -1000.0, 1000.0, 0.0)
with col2:
    depth = st.number_input("Depth (km)", 0.0, 700.0, 20.0)
    mmi = st.number_input("MMI", 0.0, 10.0, 5.0)

input_data = pd.DataFrame([[magnitude, depth, cdi, mmi, sig]],
                          columns=["magnitude", "depth", "cdi", "mmi", "sig"])

if st.button("🔍 Prediksi Alert"):
    pred = rf_model.predict(input_data)[0]
    st.success("✅ Prediksi berhasil dilakukan!")

    st.markdown("### 🎯 Hasil Prediksi")
    highlight_alert("Random Forest", pred)

    st.markdown("---")
    st.markdown("###                  Arti Warna Alert")
    alert_df = pd.DataFrame({
        "Kode": list(alert_labels.keys()),
        "Arti": list(alert_labels.values())
    })
    st.table(alert_df)
```