Klasifikasi Tsunami dan Gempa - Machine Learning

In [ ]:
```
# Data Manipulation
import pandas as pd
import numpy as np

# Visualization
import matplotlib.pyplot as plt

# Machine Learning, Klasifikasi, dan Evaluasi
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import SelectKBest, SelectPercentile, f_classif
from sklearn.model_selection import StratifiedKFold, GridSearchCV
import time
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc
from sklearn.metrics import (
    confusion_matrix,
    ConfusionMatrixDisplay,
    classification_report
)
```

Load Dataset

In [17]:
```
file_path = 'earthquake_data_tsunami.csv'
df_earthquakes_tsunami = pd.read_csv(file_path)
df_earthquakes_tsunami.head()
```

Out[17]:

	magnitude	cdi	mmi	sig	nst	dmin	gap	depth	latitude	longitude	Year	M
0	7.0	8	7	768	117	0.509	17.0	14.000	-9.7963	159.596	2022	
1	6.9	4	4	735	99	2.229	34.0	25.000	-4.9559	100.738	2022	
2	7.0	3	3	755	147	3.125	18.0	579.000	-20.0508	-178.346	2022	
3	7.3	5	5	833	149	1.865	21.0	37.000	-19.2918	-172.129	2022	
4	6.6	0	2	670	131	4.998	27.0	624.464	-25.5948	178.278	2022	

Pembersihan Data

In [18]:
```
#Cek Kolom
print("Jumlah nilai kosong per kolom:\n", df_earthquakes_tsunami.isnull().sum())

#Validasi Ulang
print("\nSetelah imputasi, nilai kosong per kolom:\n", df_earthquakes_tsunami.is
```

```
Jumlah nilai kosong per kolom:
 magnitude    0
cdi          0
mmi          0
sig          0
nst          0
dmin         0
gap          0
depth        0
latitude     0
longitude    0
Year         0
Month        0
tsunami      0
dtype: int64

Setelah imputasi, nilai kosong per kolom:
 magnitude    0
cdi          0
mmi          0
sig          0
nst          0
dmin         0
gap          0
depth        0
latitude     0
longitude    0
Year         0
Month        0
tsunami      0
dtype: int64
```

In [19]:
```
before = df_earthquakes_tsunami.shape
dupes = df_earthquakes_tsunami[df_earthquakes_tsunami.duplicated(keep=False)]
print(f"Jumlah baris duplikat (terhitung ganda): {dupes.shape[0]}")
df_earthquakes_tsunami2 = df_earthquakes_tsunami.drop_duplicates(keep='first')
print("Bentuk data sebelum/ setelah hapus duplikat:", before, "->", df_earthquak
```

```
Jumlah baris duplikat (terhitung ganda): 0
Bentuk data sebelum/ setelah hapus duplikat: (782, 13) -> (782, 13)
```

Pemisahan Fitur-Target

In [20]:
```
y = df_earthquakes_tsunami['tsunami']
not_needed_columns = ['tsunami', 'Year', 'Month']
X = df_earthquakes_tsunami.drop(columns=not_needed_columns)
```

Train Test

In [21]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.30, random_state = 28, stratify=y)
```

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

```
Ukuran X_train, X_test: (547, 10) (235, 10)
```

Random Forest

In [22]:
```
# ============================================================
#  PIPELINE: Scaling → Feature Selection → Random Forest
# ============================================================


# Rancang pipeline: gabungkan scaling, seleksi fitur, dan model Random Forest
pipe_rf = Pipeline(steps=[
    ('feat_select', SelectKBest()),
    ('clf', RandomForestClassifier(
        class_weight='balanced',
        random_state=28,
        n_estimators=-1
    ))
])

# GridSearch: dua jenis seleksi fitur (KBest dan Percentile) dengan kombinasi pa
params_grid_rf = [
    # Kandidat 1: pakai SelectKBest
    {
        'feat_select__k': np.arange(5, 15),        # jumlah fitur terbaik yang a
        'clf__n_estimators': [100, 300, 500],      # jumlah pohon
        'clf__max_depth': [None, 5, 10],           # batas kedalaman tiap pohon
        'clf__min_samples_split': [2, 5, 10]       # jumlah minimal sampel untuk
    },
    # Kandidat 2: pakai SelectPercentile
    {
        'feat_select': [SelectPercentile()],
        'feat_select__percentile': np.arange(30, 80, 10),
        'clf__n_estimators': [100, 300, 500],
        'clf__max_depth': [None, 5, 10],
        'clf__min_samples_split': [2, 5, 10]
    }
]

# StratifiedKFold: memastikan proporsi kelas tetap sama di setiap fold CV
SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=28)

# Jalankan GridSearchCV: mencari kombinasi parameter terbaik dengan metrik F1
gscv_rf = GridSearchCV(
    pipe_rf,
    params_grid_rf,
    cv=SKF,
    scoring='f1',
    verbose=1,
    n_jobs=-1
)

print("Menjalankan GridSearch untuk Random Forest...")
start = time.time()

gscv_rf.fit(X_train, y_train)
```

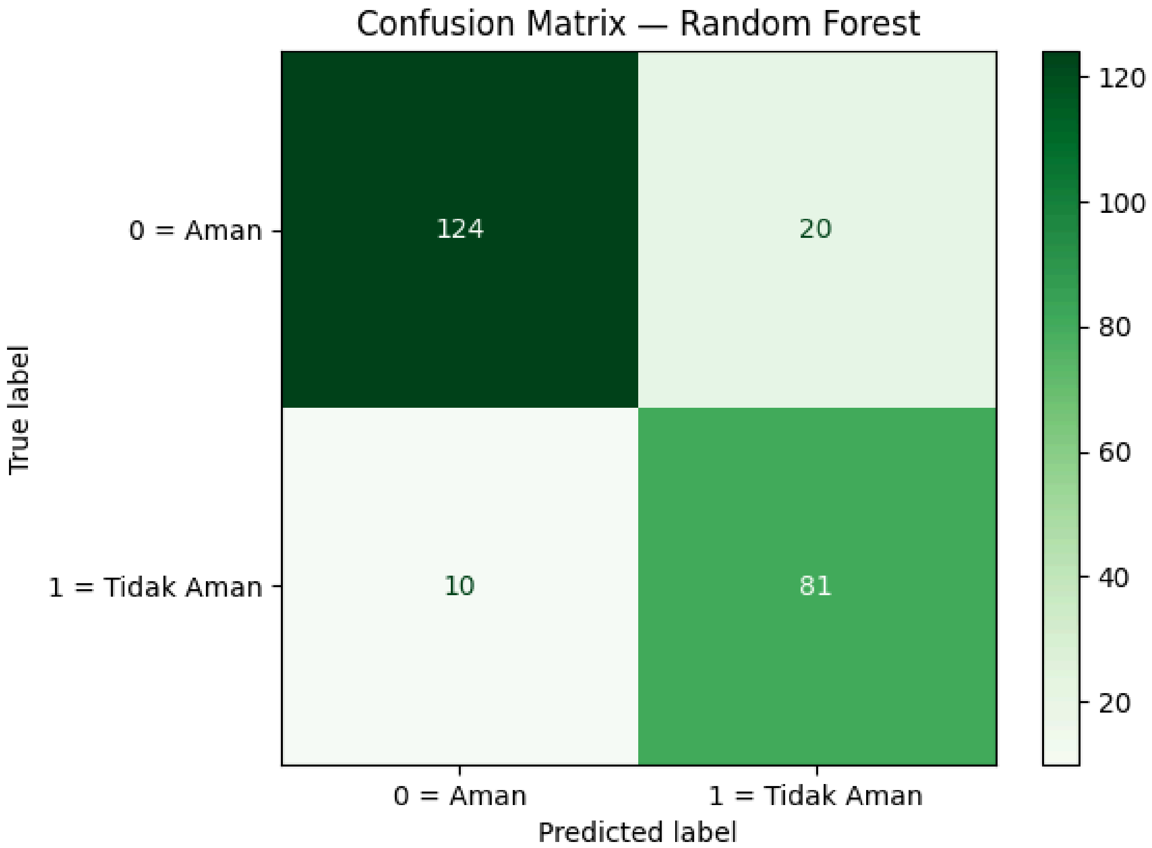
```
print(f"GridSearch Random Forest selesai dalam {time.time() - start:.2f} detik")
```

```
Menjalankan GridSearch untuk Random Forest...
Fitting 5 folds for each of 405 candidates, totalling 2025 fits
GridSearch Random Forest selesai dalam 307.83 detik
```

Evaluasi Random Forest

In [23]:
```
#  Evaluasi hasil GridSearch
print("CV Score (F1) terbaik:", gscv_rf.best_score_)
print("Kombinasi model terbaik:", gscv_rf.best_estimator_)

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

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

#  Confusion Matrix & Classification Report
rf_pred = gscv_rf.predict(X_test)
cm_rf = confusion_matrix(y_test, rf_pred)
disp_rf = ConfusionMatrixDisplay(confusion_matrix=cm_rf, display_labels=['0 = Am
disp_rf.plot(cmap=plt.cm.Greens)
plt.title("Confusion Matrix — Random Forest")
plt.show()

# Silakan diisi bagian ini dengan kode yang tepat ()
print("\nClassification Report — Random Forest:\n", classification_report(y_test
print(f"Accuracy: {accuracy_score(y_test, rf_pred):.4f}")
print(f"Precision: {precision_score(y_test, rf_pred):.4f}")
print(f"Recall: {recall_score(y_test, rf_pred):.4f}")
print(f"F1-Score: {f1_score(y_test, rf_pred):.4f}")
```

```
CV Score (F1) terbaik: 0.8663305322128851
Kombinasi model terbaik: Pipeline(steps=[('feat_select', SelectKBest(k=np.int64
(7))),
                ('clf',
                 RandomForestClassifier(class_weight='balanced',
                                        min_samples_split=10, n_estimators=300,
                                        random_state=28))])

Skor Test (akurasi) Random Forest: 0.8723404255319149

Fitur terbaik (terpilih): ['cdi' 'mmi' 'nst' 'dmin' 'gap' 'latitude' 'longitude']
```




```
Classification Report — Random Forest:
              precision    recall  f1-score   support

           0       0.93      0.86      0.89       144
           1       0.80      0.89      0.84        91

    accuracy                           0.87       235
   macro avg       0.86      0.88      0.87       235
weighted avg       0.88      0.87      0.87       235

Accuracy: 0.8723
Precision: 0.8020
Recall: 0.8901
F1-Score: 0.8438
```

Logistic Regression

In [24]:

```
# ============================================================
#  PIPELINE: Scaling → Feature Selection → Logistic Regression
# ============================================================

pipe_lr = Pipeline(steps=[
    ('scaler', StandardScaler()),  # placeholder, nanti diganti lewat GridSearch
    ('feat_select', SelectKBest()),
    ('clf', LogisticRegression(
        class_weight='balanced',
        solver='liblinear',
        max_iter=500
    ))
])

# GridSearch: tambahkan pilihan dua scaler (Standard & MinMax)
params_grid_lr = [
```

```
    {
        'scaler': [StandardScaler(), MinMaxScaler()],     # <--- Tambahan pentin
        'feat_select': [SelectKBest()],
        'feat_select__k': np.arange(2, 10),
        'clf__penalty': ['l1', 'l2'],
        'clf__C': [0.01, 0.1, 1, 10],
    },
    {
        'scaler': [StandardScaler(), MinMaxScaler()],     # <--- Tambahan pentin
        'feat_select': [SelectPercentile()],
        'feat_select__percentile': np.arange(20, 80, 10),
        'clf__penalty': ['l1', 'l2'],
        'clf__C': [0.01, 0.1, 1, 10],
    }
]

# Stratified K-Fold Cross Validation
SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=28)

# Jalankan GridSearchCV untuk mencari kombinasi terbaik
gscv_lr = GridSearchCV(
    pipe_lr,
    params_grid_lr,
    cv=SKF,
    scoring='f1',
    verbose=1,
    n_jobs=-1
)

print("Menjalankan GridSearch untuk Logistic Regression...")
start = time.time()
gscv_lr.fit(X_train, y_train)
print(f"GridSearch Logistic Regression selesai dalam {time.time() - start:.2f} d
```

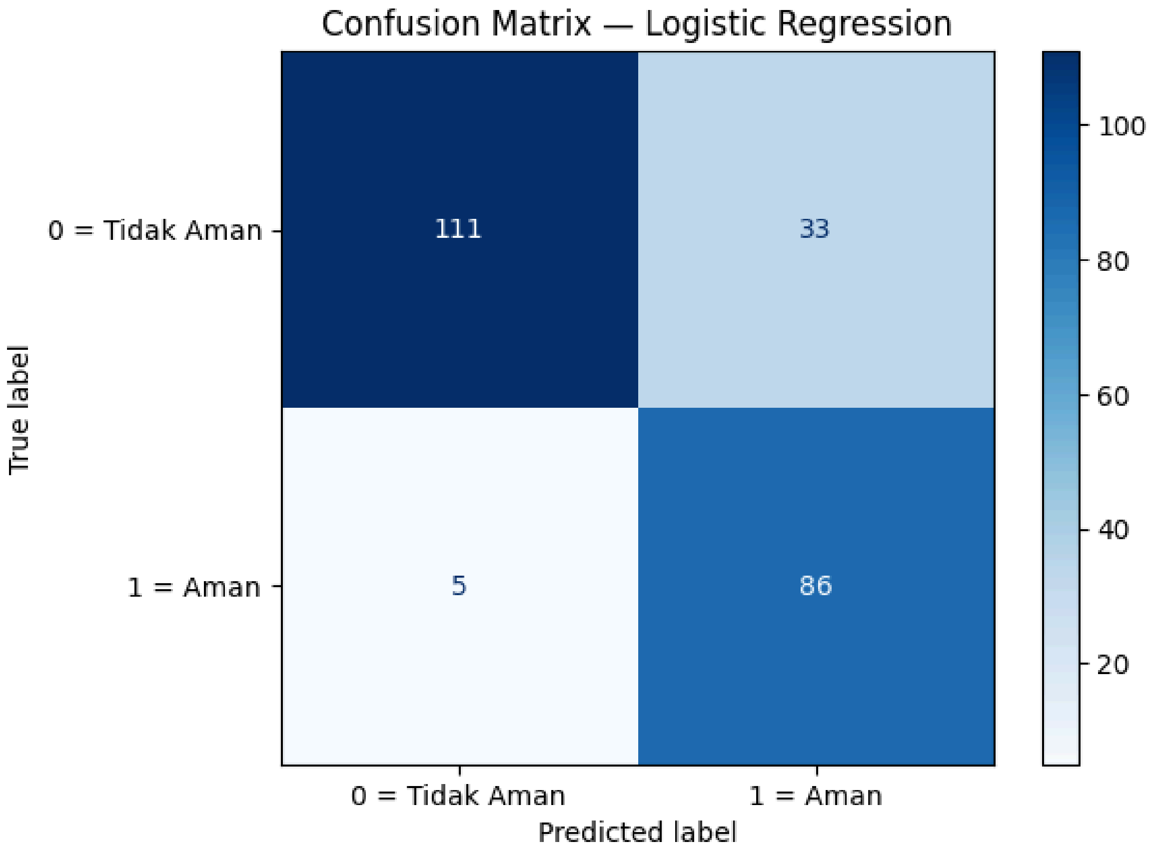
```
Menjalankan GridSearch untuk Logistic Regression...
Fitting 5 folds for each of 224 candidates, totalling 1120 fits
GridSearch Logistic Regression selesai dalam 1.87 detik
```

Evaluasi Logistic Regression

In [25]:
```
# Evaluasi hasil terbaik
print("CV Score (F1) terbaik:", gscv_lr.best_score_)
print("Kombinasi model terbaik:", gscv_lr.best_estimator_)

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

# Fitur terbaik (jika feature selector mendukung get_support)
selector = gscv_lr.best_estimator_.named_steps['feat_select']
if hasattr(selector, 'get_support'):
    mask = selector.get_support()
    selected = np.array(X.columns)[mask]
    print("\nFitur terbaik (terpilih):", selected)

# Confusion Matrix & Classification Report
lr_pred = gscv_lr.predict(X_test)
cm_lr = confusion_matrix(y_test, lr_pred)
disp_lr = ConfusionMatrixDisplay(confusion_matrix=cm_lr, display_labels=['0 = Ti
disp_lr.plot(cmap=plt.cm.Blues)
```

```
plt.title("Confusion Matrix — Logistic Regression")
plt.show()

print("\nClassification Report — Logistic Regression:\n", classification_report(
print(f"Accuracy: {accuracy_score(y_test, lr_pred):.4f}")
print(f"Precision: {precision_score(y_test, lr_pred):.4f}")
print(f"Recall: {recall_score(y_test, lr_pred):.4f}")
print(f"F1-Score: {f1_score(y_test, lr_pred):.4f}")
```

```
CV Score (F1) terbaik: 0.7963007289739963
Kombinasi model terbaik: Pipeline(steps=[('scaler', StandardScaler()),
                ('feat_select', SelectKBest(k=np.int64(4))),
                ('clf',
                 LogisticRegression(C=0.01, class_weight='balanced',
                                    max_iter=500, solver='liblinear'))])

Skor Test (akurasi) Logistic Regression: 0.8382978723404255

Fitur terbaik (terpilih): ['cdi' 'nst' 'dmin' 'longitude']
```

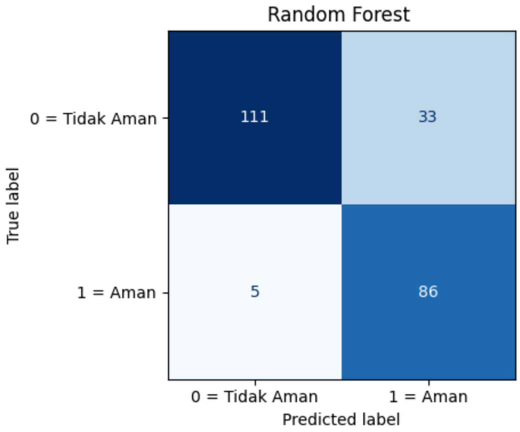


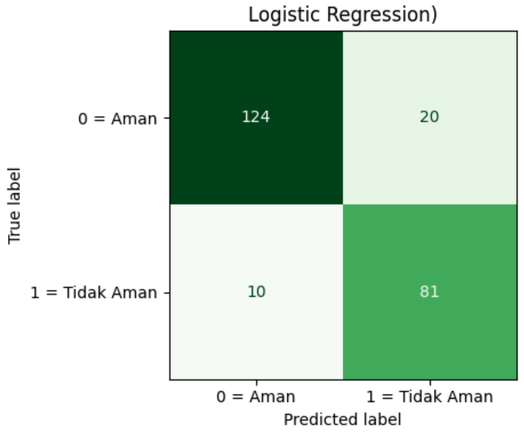

```
Classification Report — Logistic Regression:
               precision    recall  f1-score   support

           0       0.96      0.77      0.85       144
           1       0.72      0.95      0.82        91

    accuracy                           0.84       235
   macro avg       0.84      0.86      0.84       235
weighted avg       0.87      0.84      0.84       235

Accuracy: 0.8383
Precision: 0.7227
Recall: 0.9451
F1-Score: 0.8190
```

Perbandingan Random Forest dan Logistic Regression

In [26]:

```
# Buat figure dengan 2 subplot berdampingan (1 baris, 2 kolom)
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 4))

# Plot Confusion Matrix untuk Random Forest
disp_lr.plot(ax=ax1, cmap=plt.cm.Blues, colorbar=False)
ax1.set_title("Random Forest ")

# Plot Confusion Matrix untuk Logistic Regression
disp_rf.plot(ax=ax2, cmap=plt.cm.Greens, colorbar=False)
ax2.set_title("Logistic Regression)")

# Rapikan tata letak agar subplot tidak tumpang tindih
plt.tight_layout()
plt.show()

# Hitung metrik untuk kedua model
metrics_comparison = {
    'Model': ['Random Forest', 'Logistic Regression'],
    'Accuracy': [
        accuracy_score(y_test, rf_pred),
        accuracy_score(y_test, lr_pred)
    ],
    'Precision': [
        precision_score(y_test, rf_pred),
        precision_score(y_test, lr_pred)
    ],
    'Recall': [
        recall_score(y_test, rf_pred),
        recall_score(y_test, lr_pred)
    ],
    'F1-Score': [
        f1_score(y_test, rf_pred),
        f1_score(y_test, lr_pred)
    ]
}

# Buat DataFrame untuk perbandingan
df_comparison = pd.DataFrame(metrics_comparison)

print("PERBANDINGAN METRIK EVALUASI\n")
print(df_comparison.to_string(index=False))

# Model terbaik berdasarkan F1-Score
best_idx_f1 = df_comparison['F1-Score'].idxmax()
best_model = df_comparison.loc[best_idx_f1, 'Model']
print(f"\nModel terbaik berdasarkan F1-Score adalah: {best_model}")
```






```
PERBANDINGAN METRIK EVALUASI

              Model  Accuracy  Precision   Recall  F1-Score
      Random Forest  0.872340   0.801980 0.890110  0.843750
Logistic Regression  0.838298   0.722689 0.945055  0.819048

Model terbaik berdasarkan F1-Score adalah: Random Forest
```

Klasifikasi Tsunami dan Gempa - Machine Learning

In [14]:
```
# Data Manipulation
import pandas as pd
import numpy as np

# Visualization
import matplotlib.pyplot as plt

# Machine Learning, Klasifikasi, dan Evaluasi
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import SelectKBest, SelectPercentile, f_classif
from sklearn.model_selection import StratifiedKFold, GridSearchCV
import time
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc
from sklearn.metrics import (
    confusion_matrix,
    ConfusionMatrixDisplay,
    classification_report
)
```

Load Dataset

In [3]:
```
file_path = 'earthquake_data_tsunami.csv'
df_earthquakes_tsunami = pd.read_csv(file_path)
df_earthquakes_tsunami.head()
```

Out[3]:

	magnitude	cdi	mmi	sig	nst	dmin	gap	depth	latitude	longitude	Year	M
0	7.0	8	7	768	117	0.509	17.0	14.000	-9.7963	159.596	2022	
1	6.9	4	4	735	99	2.229	34.0	25.000	-4.9559	100.738	2022	
2	7.0	3	3	755	147	3.125	18.0	579.000	-20.0508	-178.346	2022	
3	7.3	5	5	833	149	1.865	21.0	37.000	-19.2918	-172.129	2022	
4	6.6	0	2	670	131	4.998	27.0	624.464	-25.5948	178.278	2022	

Pembersihan Data

In [4]:
```
#Cek Kolom
print("Jumlah nilai kosong per kolom:\n", df_earthquakes_tsunami.isnull().sum())

#Validasi Ulang
print("\nSetelah imputasi, nilai kosong per kolom:\n", df_earthquakes_tsunami.is
```

```
Jumlah nilai kosong per kolom:
 magnitude    0
cdi          0
mmi          0
sig          0
nst          0
dmin         0
gap          0
depth        0
latitude     0
longitude    0
Year         0
Month        0
tsunami      0
dtype: int64

Setelah imputasi, nilai kosong per kolom:
 magnitude    0
cdi          0
mmi          0
sig          0
nst          0
dmin         0
gap          0
depth        0
latitude     0
longitude    0
Year         0
Month        0
tsunami      0
dtype: int64
```

In [5]:
```
before = df_earthquakes_tsunami.shape
dupes = df_earthquakes_tsunami[df_earthquakes_tsunami.duplicated(keep=False)]
print(f"Jumlah baris duplikat (terhitung ganda): {dupes.shape[0]}")
df_earthquakes_tsunami2 = df_earthquakes_tsunami.drop_duplicates(keep='first')
print("Bentuk data sebelum/ setelah hapus duplikat:", before, "->", df_earthquak
```

```
Jumlah baris duplikat (terhitung ganda): 0
Bentuk data sebelum/ setelah hapus duplikat: (782, 13) -> (782, 13)
```

Pemisahan Fitur-Target

In [6]:
```
y = df_earthquakes_tsunami['tsunami']
not_needed_columns = ['tsunami', 'Year', 'Month']
X = df_earthquakes_tsunami.drop(columns=not_needed_columns)
```

Train Test

In [7]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.30, random_state = 28, stratify=y)
```

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

```
Ukuran X_train, X_test: (547, 10) (235, 10)
```

Gradient Boosting Classifier

In [8]:
```
# ============================================================
#  PIPELINE: Feature Selection → Gradient Boosting Classifier
# ============================================================
pipe_gb = Pipeline(steps=[
    ('feat_select', SelectKBest(score_func=f_classif)),
    ('clf', GradientBoostingClassifier(random_state=28))
])

# GridSearch: dua metode seleksi fitur + parameter model
params_grid_gb = [
    {
        'feat_select': [SelectKBest(score_func=f_classif)],
        'feat_select__k': np.arange(5, 15),
        'clf__n_estimators': [100, 200],
        'clf__learning_rate': [0.01, 0.1, 0.2],
        'clf__max_depth': [2, 3, 5]
    },
    {
        'feat_select': [SelectPercentile(score_func=f_classif)],
        'feat_select__percentile': np.arange(30, 80, 10),
        'clf__n_estimators': [100, 200],
        'clf__learning_rate': [0.01, 0.1, 0.2],
        'clf__max_depth': [2, 3, 5]
    }
]

SKF = StratifiedKFold(n_splits=5, shuffle=True, random_state=28)

# Jalankan GridSearchCV
print("Menjalankan GridSearch untuk Gradient Boosting...")
start = time.time()
gscv_gb = GridSearchCV(
    pipe_gb,
    params_grid_gb,
    cv=SKF,
    scoring='f1',
    verbose=1,
    n_jobs=-1
)
gscv_gb.fit(X_train, y_train)
print(f"GridSearch Gradient Boosting selesai dalam {time.time() - start:.2f} det
```

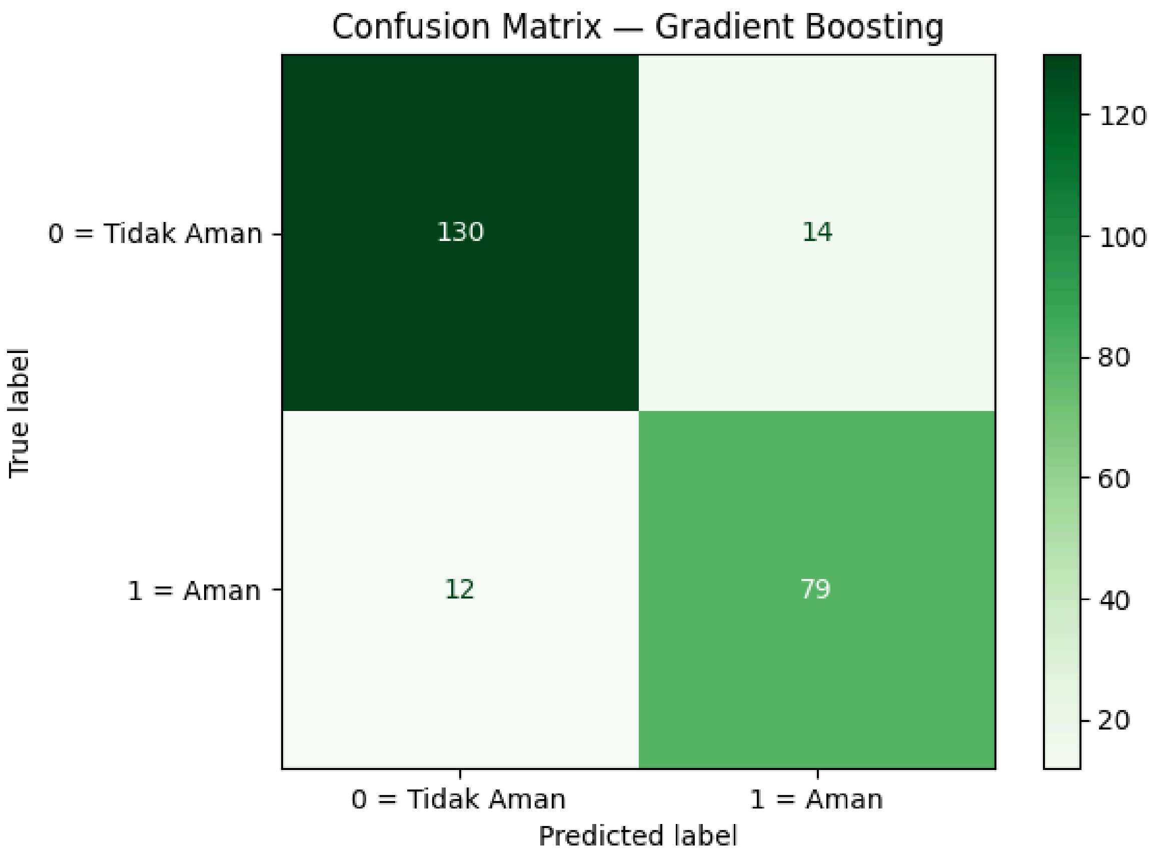
```
Menjalankan GridSearch untuk Gradient Boosting...
Fitting 5 folds for each of 270 candidates, totalling 1350 fits
GridSearch Gradient Boosting selesai dalam 120.15 detik
```

Evaluasi Gradient Boosting Classifier

In [9]:
```
# === Evaluasi Hasil Terbaik
print("CV Score (F1) terbaik:", gscv_gb.best_score_)
print("Kombinasi model terbaik:", gscv_gb.best_estimator_)
```

```
gb_test_score = gscv_gb.best_estimator_.score(X_test, y_test)
print("\nSkor Test (akurasi) Gradient Boosting:", gb_test_score)

# Fitur terbaik (jika selector mendukung get_support)
selector_gb = gscv_gb.best_estimator_.named_steps['feat_select']
if hasattr(selector_gb, 'get_support'):
    mask = selector_gb.get_support()
    selected = np.array(X.columns)[mask]
    print("\nFitur terbaik (terpilih):", selected)

# Confusion Matrix
gb_pred = gscv_gb.predict(X_test)
cm_gb = confusion_matrix(y_test, gb_pred)
disp_gb = ConfusionMatrixDisplay(confusion_matrix=cm_gb, display_labels=['0 = Ti
disp_gb.plot(cmap=plt.cm.Greens)
plt.title("Confusion Matrix — Gradient Boosting")
plt.show()

print("\nClassification Report — Gradient Boosting:\n", classification_report(y_
print(f"Accuracy: {accuracy_score(y_test, gb_pred):.4f}")
print(f"Precision: {precision_score(y_test, gb_pred):.4f}")
print(f"Recall: {recall_score(y_test, gb_pred):.4f}")
print(f"F1-Score: {f1_score(y_test, gb_pred):.4f}")
```

```
CV Score (F1) terbaik: 0.8741186185088624
Kombinasi model terbaik: Pipeline(steps=[('feat_select', SelectKBest(k=np.int64
(7))),
                ('clf',
                 GradientBoostingClassifier(max_depth=2, n_estimators=200,
                                            random_state=28))])

Skor Test (akurasi) Gradient Boosting: 0.8893617021276595

Fitur terbaik (terpilih): ['cdi' 'mmi' 'nst' 'dmin' 'gap' 'latitude' 'longitude']
```



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

           0       0.92      0.90      0.91       144
           1       0.85      0.87      0.86        91

    accuracy                           0.89       235
   macro avg       0.88      0.89      0.88       235
weighted avg       0.89      0.89      0.89       235

Accuracy: 0.8894
Precision: 0.8495
Recall: 0.8681
F1-Score: 0.8587
```

Support Vector Machine

In [10]:
```
# ============================================================
#  PIPELINE: Scaling → Feature Selection → Support Vector Machine
# ============================================================
pipe_svm = Pipeline(steps=[
    ('scaler', StandardScaler()),  # placeholder (akan diganti di grid)
    ('feat_select', SelectKBest(score_func=f_classif)),
    ('clf', SVC(probability=True, random_state=28))
])

# GridSearch: bandingkan 2 scaler + 2 metode seleksi fitur + parameter SVM
params_grid_svm = [
    {
        'scaler': [StandardScaler(), MinMaxScaler()],
        'feat_select': [SelectKBest(score_func=f_classif)],
        'feat_select__k': np.arange(5, 15),
        'clf__C': [0.1, 1, 10],
        'clf__kernel': ['linear', 'rbf'],
        'clf__gamma': ['scale', 'auto']
    },
    {
        'scaler': [StandardScaler(), MinMaxScaler()],
        'feat_select': [SelectPercentile(score_func=f_classif)],
        'feat_select__percentile': np.arange(30, 80, 10),
        'clf__C': [0.1, 1, 10],
        'clf__kernel': ['linear', 'rbf'],
        'clf__gamma': ['scale', 'auto']
    }
]

print("Menjalankan GridSearch untuk SVM...")
start = time.time()
gscv_svm = GridSearchCV(
    pipe_svm,
    params_grid_svm,
    cv=SKF,
    scoring='f1',
    verbose=1,
    n_jobs=-1
)
gscv_svm.fit(X_train, y_train)
print(f"GridSearch SVM selesai dalam {time.time() - start:.2f} detik")
```

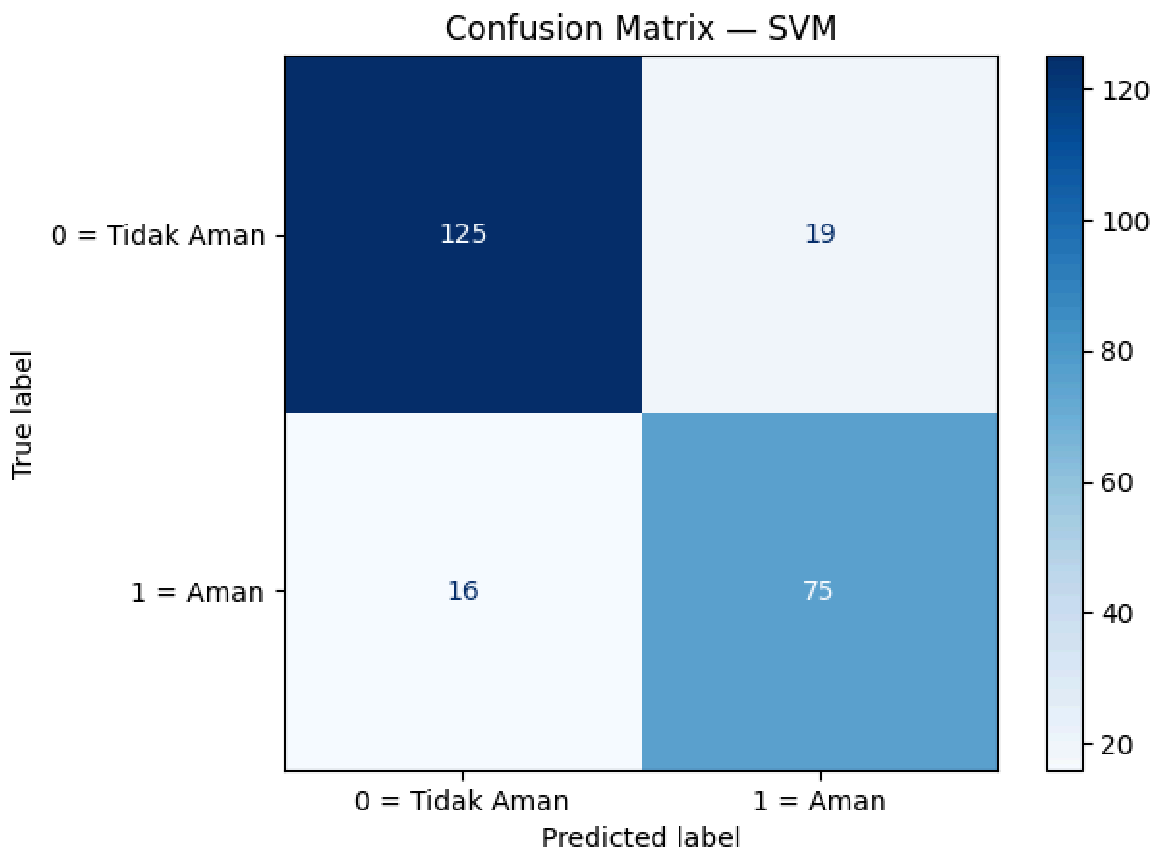
```
Menjalankan GridSearch untuk SVM...
Fitting 5 folds for each of 360 candidates, totalling 1800 fits
GridSearch SVM selesai dalam 23.34 detik
```

Evaluasi Support Vector Machine

In [11]:

```python
# === Evaluasi hasil terbaik
print("CV Score (F1) terbaik:", gscv_svm.best_score_)
print("Kombinasi model terbaik:", gscv_svm.best_estimator_)

svm_test_score = gscv_svm.best_estimator_.score(X_test, y_test)
print("\nSkor Test (akurasi) SVM:", svm_test_score)

# Fitur terbaik (jika selector mendukung get_support)
selector_svm = gscv_svm.best_estimator_.named_steps['feat_select']
if hasattr(selector_svm, 'get_support'):
    mask = selector_svm.get_support()
    selected = np.array(X.columns)[mask]
    print("\nFitur terbaik (terpilih):", selected)

# Confusion Matrix
svm_pred = gscv_svm.predict(X_test)
cm_svm = confusion_matrix(y_test, svm_pred)
disp_svm = ConfusionMatrixDisplay(confusion_matrix=cm_svm, display_labels=['0 =
disp_svm.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix – SVM")
plt.show()

print("\nClassification Report – SVM:\n", classification_report(y_test, svm_pred
print(f"Accuracy: {accuracy_score(y_test, svm_pred):.4f}")
print(f"Precision: {precision_score(y_test, svm_pred):.4f}")
print(f"Recall: {recall_score(y_test, svm_pred):.4f}")
print(f"F1-Score: {f1_score(y_test, svm_pred):.4f}")
```

```
CV Score (F1) terbaik: 0.8346354546792452
Kombinasi model terbaik: Pipeline(steps=[('scaler', MinMaxScaler()),
                ('feat_select', SelectKBest(k=np.int64(6))),
                ('clf', SVC(C=10, probability=True, random_state=28))])

Skor Test (akurasi) SVM: 0.851063829787234

Fitur terbaik (terpilih): ['cdi' 'mmi' 'nst' 'dmin' 'gap' 'longitude']
```

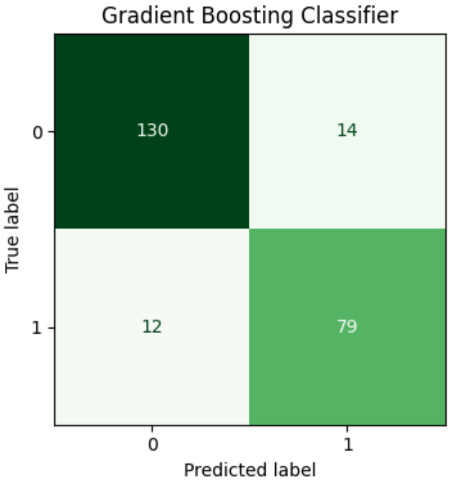


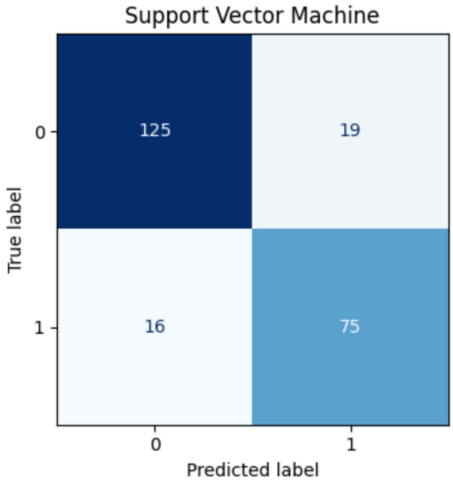

```
Classification Report — SVM:
              precision    recall  f1-score   support

           0       0.89      0.87      0.88       144
           1       0.80      0.82      0.81        91

    accuracy                           0.85       235
   macro avg       0.84      0.85      0.84       235
weighted avg       0.85      0.85      0.85       235

Accuracy: 0.8511
Precision: 0.7979
Recall: 0.8242
F1-Score: 0.8108
```

Perbandingan Gradient Boosting Classifier dan Support Vector Machine

In [12]:
```
# Buat figure dengan 2 subplot berdampingan (1 baris, 2 kolom)
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 4))

# Confusion Matrix untuk Gradient Boosting Classifier
disp_gb = ConfusionMatrixDisplay.from_estimator(gscv_gb.best_estimator_, X_test,
ax1.set_title("Gradient Boosting Classifier")

# Confusion Matrix untuk Support Vector Machine
disp_svm = ConfusionMatrixDisplay.from_estimator(gscv_svm.best_estimator_, X_tes
ax2.set_title("Support Vector Machine")

# Rapikan tata letak agar subplot tidak tumpang tindih
plt.tight_layout()
plt.show()
```

```python
# Hitung metrik untuk kedua model
metrics_comparison = {
    'Model': ['Gradient Boosting', 'Support Vector Machine'],
    'Accuracy': [
        accuracy_score(y_test, gb_pred),
        accuracy_score(y_test, svm_pred)
    ],
    'Precision': [
        precision_score(y_test, gb_pred),
        precision_score(y_test, svm_pred)
    ],
    'Recall': [
        recall_score(y_test, gb_pred),
        recall_score(y_test, svm_pred)
    ],
    'F1-Score': [
        f1_score(y_test, gb_pred),
        f1_score(y_test, svm_pred)
    ]
}

# Buat DataFrame untuk perbandingan
df_comparison = pd.DataFrame(metrics_comparison)

print("PERBANDINGAN METRIK EVALUASI\n")
print(df_comparison.to_string(index=False))

# Model terbaik berdasarkan F1-Score
best_idx_f1 = df_comparison['F1-Score'].idxmax()
best_model = df_comparison.loc[best_idx_f1, 'Model']
print(f"\nModel terbaik berdasarkan F1-Score adalah: {best_model}")
```






```
PERBANDINGAN METRIK EVALUASI

                 Model  Accuracy  Precision   Recall  F1-Score
     Gradient Boosting  0.889362   0.849462 0.868132  0.858696
Support Vector Machine  0.851064   0.797872 0.824176  0.810811

Model terbaik berdasarkan F1-Score adalah: Gradient Boosting
```

Import salah satu model terbaik

In [13]:
```python
import pickle
# Ambil model terbaik dari hasil GridSearchCV
```

```
best_model_ = gscv_gb.best_estimator_.named_steps['clf']

# Simpan model terbaik ke file pickle
with open("BestModel_CLF_GB_Pandas.pkl", "wb") as f:
    pickle.dump(best_model_, f)

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

```
✅ Model terbaik berhasil disimpan ke 'BestModel_CLF_GB_Pandas.pkl'
```

```
import pandas as pd
import pickle
import numpy as np
import streamlit as st
from streamlit_option_menu import option_menu

#Navigasi sidebar
with st.sidebar:
    selected = option_menu ('UTS Machine Learning 24/25',
['Klasifikasi',
'Regresi'],
default_index=0)

#Load model
model = pickle.load(open('BestModel_CLF_GB_Pandas.pkl', 'rb'))

#Halaman Klasifikasi
if selected == 'Klasifikasi':
    st.title("Prediksi Klasifikasi Gempa dan Tsunami")

    #Upload dataset
    file = st.file_uploader("Upload Dataset Gempa (CSV)", type=["csv"])

    #Input manual fitur
    st.write('Input data')
    magnitude = st.number_input("Magnitudo", min_value=0.0, step=0.1)
    depth = st.number_input("Depth", min_value=0.0, step=1.0)
    latitude = st.number_input("Latitude", min_value=-90.0, max_value=90.0, step=0.1)
    longitude = st.number_input("Longitude", min_value=-180.0, max_value=180.0,
step=0.1)

    #Prediksi
    if st.button("Prediksi"):
        input_data = np.array([[magnitude, depth, latitude, longitude]])
        prediction = model.predict(input_data)
```

```
        result = "Berpotensi Tsunami 🌊" if prediction[0] == 1 else "Tidak Berpotensi Tsunami
✅"
        st.success(f"Hasil Prediksi: **{result}**")

#Halaman Regresi
if selected == 'Regresi':
    st.title("📈 Estimasi Dampak Gempa")

    magnitude = st.slider("Magnitude (Skala Richter)", 0.0, 10.0, 5.0, 0.1)
    depth = st.slider("Depth (Kedalaman, km)", 0.0, 700.0, 50.0, 1.0)

    if st.button("Hitung Estimasi Dampak"):
        impact_score = magnitude * (100 - (depth / 10))
        st.write(f"Estimasi Dampak (skor simulasi): **{impact_score:.2f}**")
```