In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, f1_score, precision_score, recall_score
from sklearn.pipeline import Pipeline
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
import re
import string
import warnings
warnings.filterwarnings('ignore')

nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')
```

```
[nltk_data] Downloading package stopwords to C:\Users\SAMUEL
[nltk_data]     KAUNANG\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to C:\Users\SAMUEL
[nltk_data]     KAUNANG\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to C:\Users\SAMUEL
[nltk_data]     KAUNANG\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

Out[1]:

```
True
```

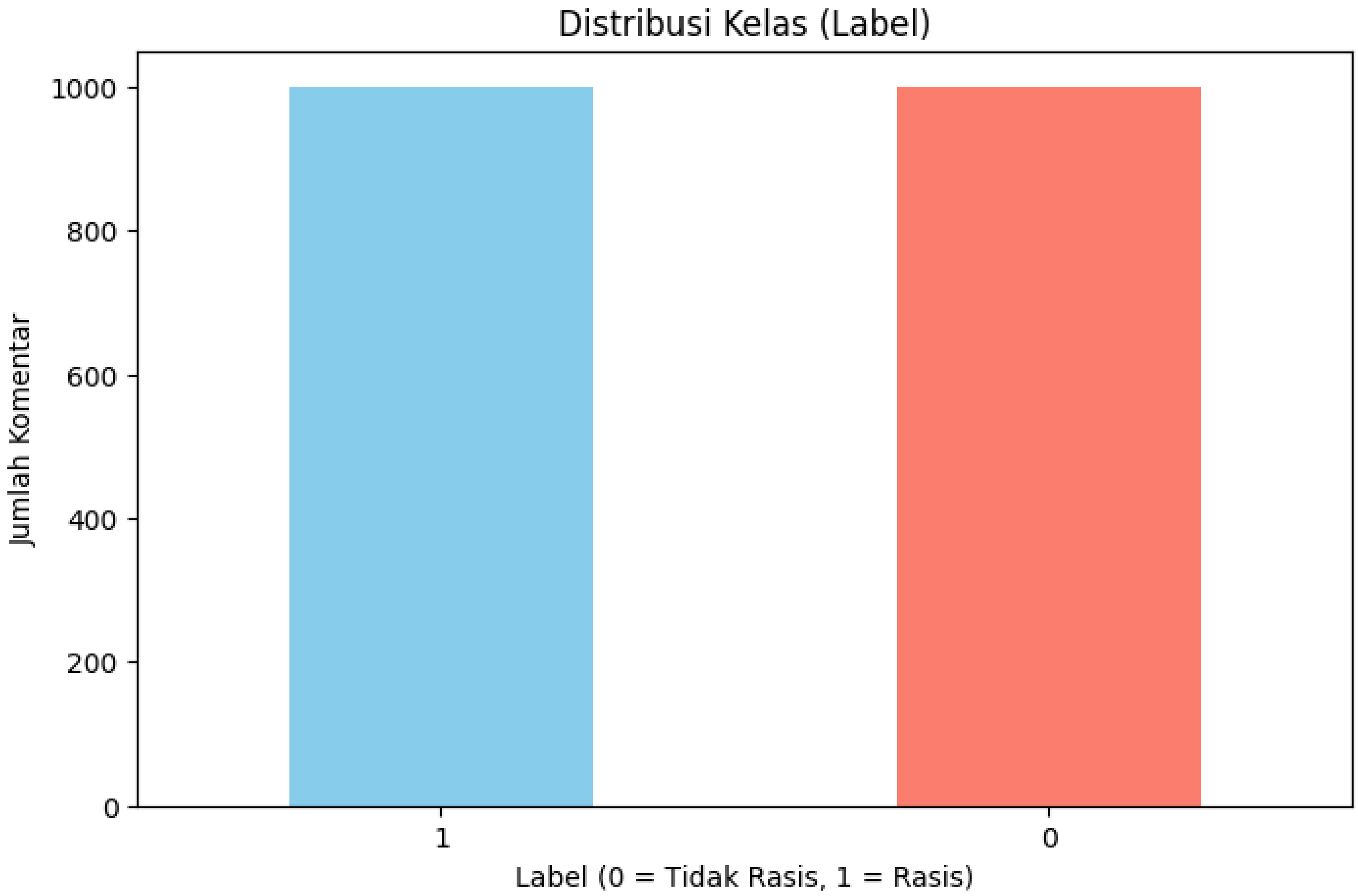
In [2]:

```
data = pd.read_csv('RacismDetectionDataSet.csv')
print("=== LOADING DATASET ===")
df = pd.read_csv('RacismDetectionDataSet.csv')
print(f"Dataset shape: {df.shape}")
print(f"\nFirst few rows:")
print(df.head())
print(f"\nLabel distribution:")
print(df['Label'].value_counts())
print(f"Rasis (1): {df['Label'].value_counts()[1]}")
print(f"Tidak Rasis (0): {df['Label'].value_counts()[0]}")
```

```
=== LOADING DATASET ===
Dataset shape: (1999, 2)

First few rows:
                                             Comment  Label
0  i was born a racist and I will die a racist I ...      1
1                      bitch nigga miss me with that      1
2  if you aint bout that murder game pussy nigga ...      1
3  gay niggas couldnt wait to act like bitches to...      1
4  why deos a gorilla always have a frown because...      1

Label distribution:
Label
1    1000
0     999
Name: count, dtype: int64
Rasis (1): 1000
Tidak Rasis (0): 999
```

In [3]:

```
plt.figure(figsize=(8, 5))
df['Label'].value_counts().plot(kind='bar', color=['skyblue', 'salmon'])
plt.title('Distribusi Kelas (Label)')
plt.xlabel('Label (0 = Tidak Rasis, 1 = Rasis)')
plt.ylabel('Jumlah Komentar')
plt.xticks(rotation=0)
plt.show()
```



In [4]:
```
print("\n=== REMOVING DUPLICATES ===")
initial_count = len(df)
df = df.drop_duplicates(subset=['Comment'], keep='first')
final_count = len(df)
print(f"Duplikasi dihapus: {initial_count - final_count} baris")
print(f"Sisa data: {final_count} baris")
```

```
=== REMOVING DUPLICATES ===
Duplikasi dihapus: 12 baris
Sisa data: 1987 baris
```

In [5]:

```
print("\n=== HANDLING MISSING VALUES ===")
print(f"Missing values sebelum cleaning:")
print(df.isnull().sum())
df = df.dropna(subset=['Comment'])
print(f"Missing values setelah cleaning:")
print(df.isnull().sum())
```

```
=== HANDLING MISSING VALUES ===
Missing values sebelum cleaning:
Comment    0
Label      0
dtype: int64
Missing values setelah cleaning:
Comment    0
Label      0
dtype: int64
```

In [6]:

```
# --- Step 3: Comprehensive Text Cleaning Function ---
def clean_text(text):
    """
    Fungsi untuk membersihkan teks:
    1. Mengubah ke lowercase
    2. Menghapus karakter aneh, simbol, emoji
    3. Menghapus angka
    4. Menghapus URL
    5. Menghapus HTML tags
    6. Menghapus extra spaces
    7. Menghapus punctuation kecuali untuk konteks tertentu
    """
    if not isinstance(text, str):
        return ""

    # 1. Lowercase
    text = text.lower()

    # 2. Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)

    # 3. Remove HTML tags
    text = re.sub(r'<.*?>', '', text)
    # 4. Remove @mentions and #hashtags (keep only text)
    text = re.sub(r'@\w+|#\w+', '', text)

    # 5. Remove special characters, symbols, and emojis
    # Hanya pertahankan huruf, angka, dan spasi
    text = re.sub(r'[^\w\s]', ' ', text)

    # 6. Remove numbers (jika tidak relevan)
    text = re.sub(r'\d+', '', text)

    # 7. Remove extra whitespace
    text = re.sub(r'\s+', ' ', text).strip()

    return text
```

In [7]:

```
print("\n=== CLEANING TEXT ===")
print("Contoh sebelum cleaning:")
print(df['Comment'].iloc[0])
df['Cleaned_Comment'] = df['Comment'].apply(clean_text)
print("\nContoh setelah cleaning:")
print(df['Cleaned_Comment'].iloc[0])
```

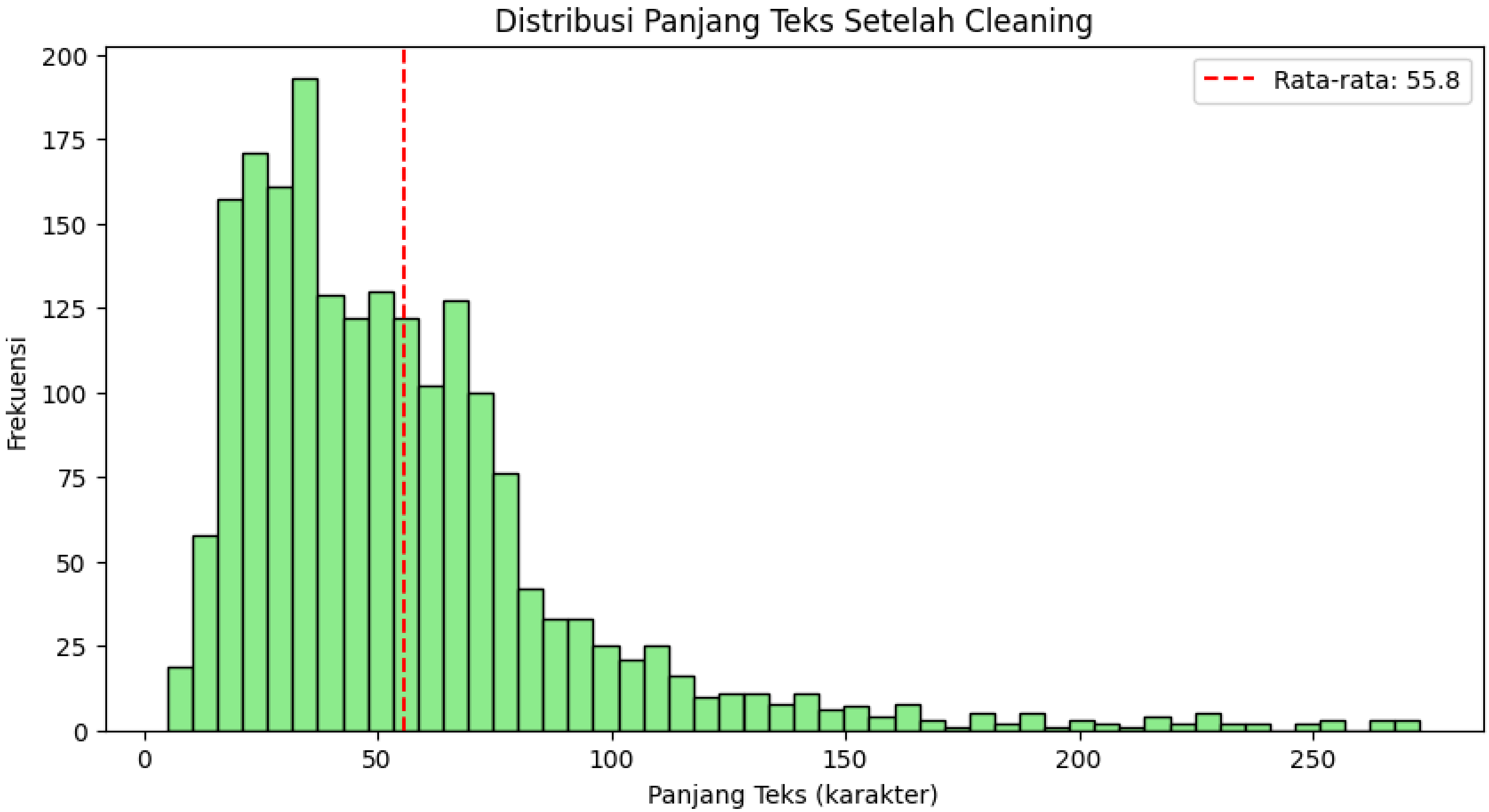
```
=== CLEANING TEXT ===
Contoh sebelum cleaning:
i was born a racist and I will die a racist I will not rest untill every worthless nigger is rounded up and hung niggers are th
e scum of the earth white america

Contoh setelah cleaning:
i was born a racist and i will die a racist i will not rest untill every worthless nigger is rounded up and hung niggers are th
e scum of the earth white america
```

In [8]:

```
# --- Step 5: Remove Noise - Very Short Comments ---
print("\n=== REMOVING NOISE (Very Short Comments) ===")
initial_len = len(df)
df['text_length'] = df['Cleaned_Comment'].apply(len)
df = df[df['text_length'] > 3]  # Hapus komentar dengan kurang dari 3 karakter
final_len = len(df)
print(f"Komentar terlalu pendek dihapus: {initial_len - final_len}")
print(f"Sisa data: {final_len}")
```

```
# Visualize text length distribution
plt.figure(figsize=(10, 5))
plt.hist(df['text_length'], bins=50, color='lightgreen', edgecolor='black')
plt.title('Distribusi Panjang Teks Setelah Cleaning')
plt.xlabel('Panjang Teks (karakter)')
plt.ylabel('Frekuensi')
plt.axvline(df['text_length'].mean(), color='red', linestyle='--', label=f'Rata-rata: {df["text_length"].mean():.1f}')
plt.legend()
plt.show()
```

```
=== REMOVING NOISE (Very Short Comments) ===
Komentar terlalu pendek dihapus: 0
Sisa data: 1987
```



In [9]:

```
# --- Step 6: Tokenization and Stopword Removal (NLTK-safe version) ---
print("\n=== TOKENIZATION & STOPWORD REMOVAL ===")

stop_words = set(stopwords.words('english'))

def tokenize_and_remove_stopwords(text):
    if not isinstance(text, str):
        return []

    # Tokenization sederhana (tanpa punkt)
    tokens = text.split()

    # Hapus stopwords dan token terlalu pendek
    tokens = [
        word for word in tokens
        if word not in stop_words and len(word) > 1
    ]

    return tokens

df['Tokens'] = df['Cleaned_Comment'].apply(tokenize_and_remove_stopwords)

print("Contoh tokens (3 data pertama):")
print(df['Tokens'].iloc[:3])
```

```
=== TOKENIZATION & STOPWORD REMOVAL ===
Contoh tokens (3 data pertama):
0    [born, racist, die, racist, rest, untill, ever...
1                                [bitch, nigga, miss]
2       [aint, bout, murder, game, pussy, nigga, shut]
Name: Tokens, dtype: object
```

In [10]:

```
# --- Step 7: Stemming/Lemmatization (Opsional) ---
print("\n=== STEMMING/LEMMATIZATION ===")
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

def stem_tokens(tokens):
    return [stemmer.stem(token) for token in tokens]
```

```python
def lemmatize_tokens(tokens):
    return [lemmatizer.lemmatize(token) for token in tokens]

# Pilih salah satu: stemming atau lemmatization
# df['Processed_Tokens'] = df['Tokens'].apply(stem_tokens)
df['Processed_Tokens'] = df['Tokens'].apply(lemmatize_tokens)

# Gabungkan kembali tokens menjadi teks
df['Final_Text'] = df['Processed_Tokens'].apply(lambda x: ' '.join(x))

print(f"Contoh hasil preprocessing lengkap:")
print(f"Original: {df['Comment'].iloc[0][:100]}...")
print(f"Cleaned: {df['Cleaned_Comment'].iloc[0][:100]}...")
print(f"Final: {df['Final_Text'].iloc[0][:100]}...")
```

```
=== STEMMING/LEMMATIZATION ===
Contoh hasil preprocessing lengkap:
Original: i was born a racist and I will die a racist I will not rest untill every worthless nigger is rounded...
Cleaned: i was born a racist and i will die a racist i will not rest untill every worthless nigger is rounded...
Final: born racist die racist rest untill every worthless nigger rounded hung nigger scum earth white ameri...
```

In [11]:

```python
# --- Step 8: Final Check for Empty Texts ---
print("\n=== FINAL CHECK ===")
df = df[df['Final_Text'].str.strip() != '']
print(f"Data final shape: {df.shape}")
print(f"\nDistribusi kelas akhir:")
print(df['Label'].value_counts())
```

```
=== FINAL CHECK ===
Data final shape: (1987, 7)

Distribusi kelas akhir:
Label
1    996
0    991
Name: count, dtype: int64
```

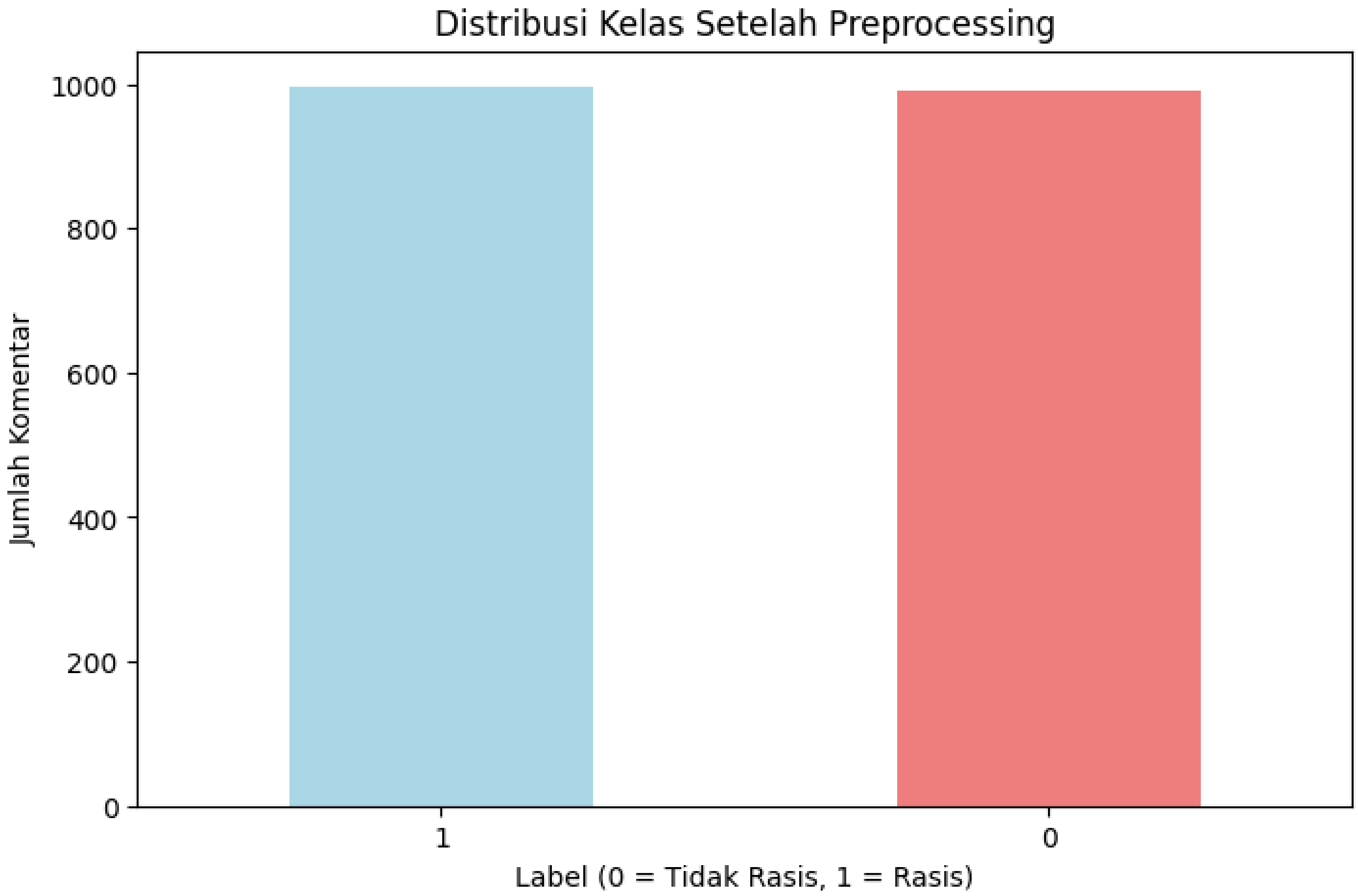
In [12]:

```python
# --- Step 8.5: Save Preprocessed Dataset ---
print("\n=== SAVING PREPROCESSED DATASET ===")

processed_df = df[[
    'Comment',              # teks asli
    'Cleaned_Comment',      # hasil cleaning
    'Final_Text',           # teks final untuk ML
    'Label'                 # ground truth
]]

processed_df.to_csv(
    'racism_dataset_preprocessed.csv',
    index=False,
    encoding='utf-8'
)

print(f"Preprocessed dataset saved!")
print(f"Total rows: {processed_df.shape[0]}")
print(f"Columns: {processed_df.columns.tolist()}")
```

```
=== SAVING PREPROCESSED DATASET ===
Preprocessed dataset saved!
Total rows: 1987
Columns: ['Comment', 'Cleaned_Comment', 'Final_Text', 'Label']
```

In [13]:

```
# Visualize final class distribution
plt.figure(figsize=(8, 5))
df['Label'].value_counts().plot(kind='bar', color=['lightblue', 'lightcoral'])
plt.title('Distribusi Kelas Setelah Preprocessing')
plt.xlabel('Label (0 = Tidak Rasis, 1 = Rasis)')
plt.ylabel('Jumlah Komentar')
plt.xticks(rotation=0)
plt.show()
```



In [14]:

```
#--- Step 9: Feature Extraction ---
print("\n=== FEATURE EXTRACTION ===")
# Pilih metode feature extraction
vectorizer = CountVectorizer(
    max_features=3000,  # Batasi jumlah fitur
    min_df=2,           # Term harus muncul minimal di 2 dokumen
    max_df=0.95,        # Term tidak boleh muncul di lebih dari 95% dokumen
    ngram_range=(1, 2)  # Gunakan unigram dan bigram
)

# Alternatif: TF-IDF
# vectorizer = TfidfVectorizer(max_features=3000, min_df=2, max_df=0.95, ngram_range=(1, 2))
```

```
X = vectorizer.fit_transform(df['Final_Text']).toarray()
y = df['Label'].values

print(f"Shape feature matrix: {X.shape}")
print(f"Jumlah fitur (vocabulary size): {len(vectorizer.get_feature_names_out())}")
```

```
=== FEATURE EXTRACTION ===
Shape feature matrix: (1987, 1968)
Jumlah fitur (vocabulary size): 1968
```

In [15]:

```
# --- Step 10: Split Data (DATAFRAME-BASED, FIXED) ---
print("\n=== SPLITTING DATA ===")

# 70% train, 15% validation, 15% test (split dataframe dulu)
df_temp, df_test = train_test_split(
    df,
    test_size=0.15,
    stratify=df['Label'],
    random_state=42
)

df_train, df_val = train_test_split(
    df_temp,
    test_size=0.1765,   # ≈15% dari total
    stratify=df_temp['Label'],
    random_state=42
)

# Buat X dan y dari masing-masing dataframe
X_train = vectorizer.fit_transform(df_train['Final_Text'])
y_train = df_train['Label']

X_val = vectorizer.transform(df_val['Final_Text'])
y_val = df_val['Label']

X_test = vectorizer.transform(df_test['Final_Text'])
y_test = df_test['Label']

print(f"Training set: {X_train.shape[0]} samples")
print(f"Validation set: {X_val.shape[0]} samples")
print(f"Test set: {X_test.shape[0]} samples")

print(f"\nDistribusi kelas training:\n{y_train.value_counts()}")
print(f"Distribusi kelas validation:\n{y_val.value_counts()}")
print(f"Distribusi kelas test:\n{y_test.value_counts()}")
```

```
=== SPLITTING DATA ===
Training set: 1390 samples
Validation set: 298 samples
Test set: 299 samples

Distribusi kelas training:
Label
1    697
0    693
Name: count, dtype: int64
Distribusi kelas validation:
Label
1    149
0    149
Name: count, dtype: int64
Distribusi kelas test:
Label
1    150
0    149
Name: count, dtype: int64
```

In [16]:

```
# --- Step 11: Naive Bayes Model Training ---
print("\n=== TRAINING NAIVE BAYES MODEL ===")
# Model baseline
baseline_model = MultinomialNB()
baseline_model.fit(X_train, y_train)

# Predict on validation set
y_pred_val_baseline = baseline_model.predict(X_val)

print("Baseline Model Performance (Validation):")
print(classification_report(y_val, y_pred_val_baseline))
print(f"Accuracy: {accuracy_score(y_val, y_pred_val_baseline):.4f}")
```

```
=== TRAINING NAIVE BAYES MODEL ===
Baseline Model Performance (Validation):
              precision    recall  f1-score   support

           0       0.85      0.81      0.83       149
           1       0.82      0.86      0.84       149

    accuracy                           0.83       298
   macro avg       0.83      0.83      0.83       298
weighted avg       0.83      0.83      0.83       298

Accuracy: 0.8322
```

In [17]:

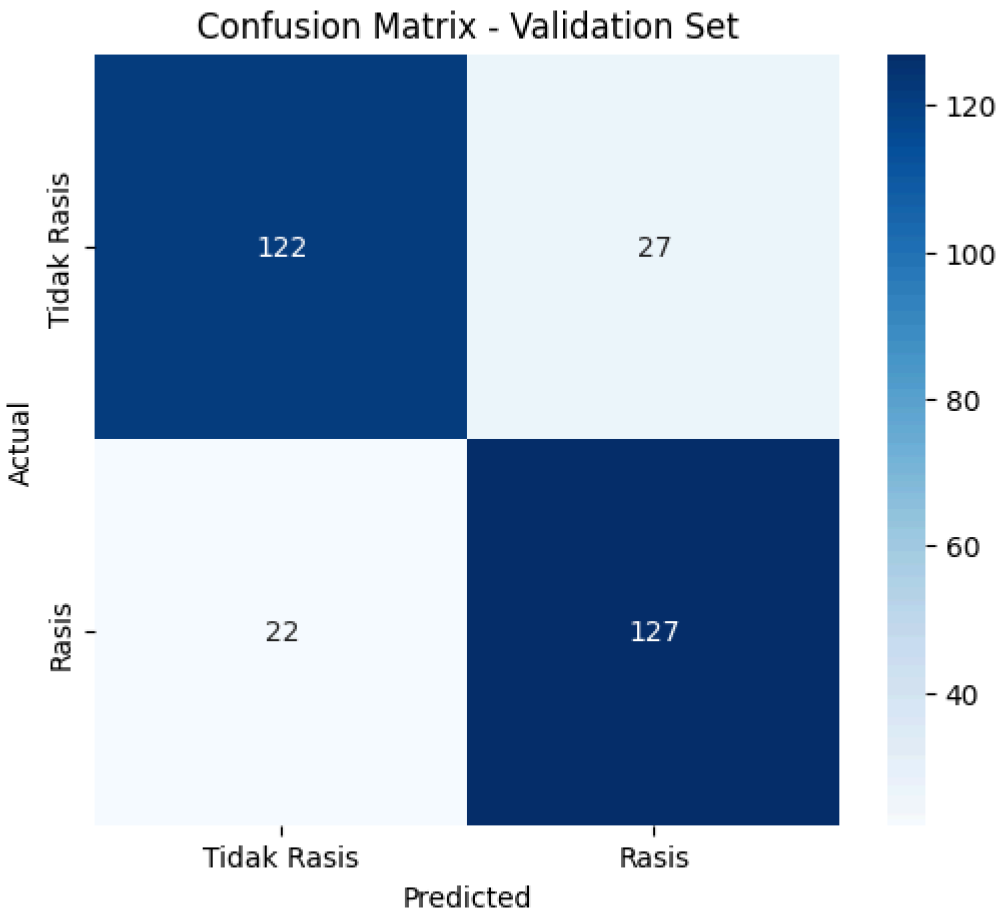
```python
# --- Step 12: Hyperparameter Tuning ---
print("\n=== HYPERPARAMETER TUNING ===")
# Definisikan parameter grid
param_grid = {
    'alpha': [0.01, 0.1, 0.5, 1.0, 2.0, 5.0, 10.0],
    'fit_prior': [True, False]
}

# Grid Search dengan 5-fold cross validation
grid_search = GridSearchCV(
    MultinomialNB(),
    param_grid,
    cv=5,
    scoring='f1',
    n_jobs=-1,
    verbose=1
)

grid_search.fit(X_train, y_train)

print(f"\nBest parameters: {grid_search.best_params_}")
print(f"Best cross-validation F1 score: {grid_search.best_score_:.4f}")

# Get best model
best_model = grid_search.best_estimator_
```

```
=== HYPERPARAMETER TUNING ===
Fitting 5 folds for each of 14 candidates, totalling 70 fits

Best parameters: {'alpha': 2.0, 'fit_prior': False}
Best cross-validation F1 score: 0.8421
```

In [18]:

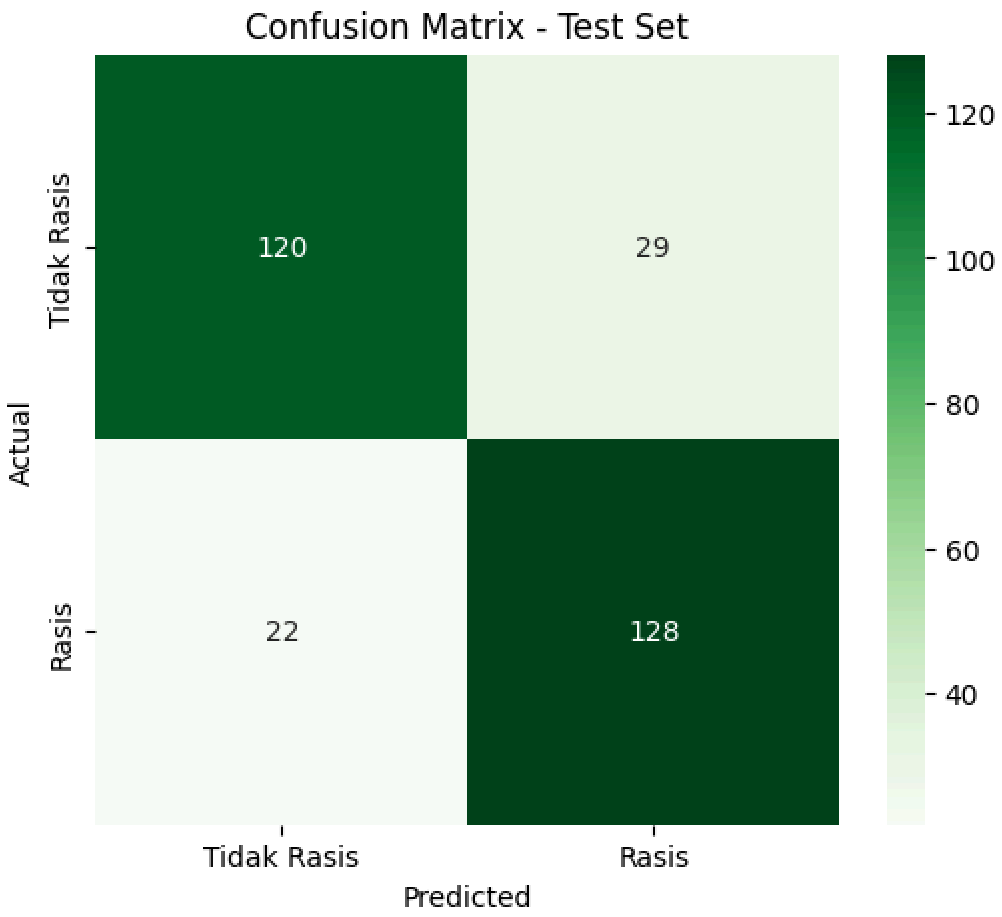
```
# --- Step 13: Evaluation on Validation Set ---
print("\n=== EVALUATION ON VALIDATION SET ===")
y_pred_val = best_model.predict(X_val)

print("Best Model Performance (Validation):")
print(classification_report(y_val, y_pred_val))

# Confusion Matrix for Validation
cm_val = confusion_matrix(y_val, y_pred_val)
plt.figure(figsize=(6, 5))
sns.heatmap(cm_val, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Tidak Rasis', 'Rasis'],
            yticklabels=['Tidak Rasis', 'Rasis'])
plt.title('Confusion Matrix - Validation Set')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

```
=== EVALUATION ON VALIDATION SET ===
Best Model Performance (Validation):
              precision    recall  f1-score   support

           0       0.85      0.82      0.83       149
           1       0.82      0.85      0.84       149

    accuracy                           0.84       298
   macro avg       0.84      0.84      0.84       298
weighted avg       0.84      0.84      0.84       298
```



```
In [19]: #--- Step 14: Final Evaluation on Test Set ---
print("\n=== FINAL EVALUATION ON TEST SET ===")
y_pred_test = best_model.predict(X_test)

print("Best Model Performance (Test Set):")
print(classification_report(y_test, y_pred_test))

# Confusion Matrix for Test
cm_test = confusion_matrix(y_test, y_pred_test)
plt.figure(figsize=(6, 5))
sns.heatmap(cm_test, annot=True, fmt='d', cmap='Greens',
            xticklabels=['Tidak Rasis', 'Rasis'],
            yticklabels=['Tidak Rasis', 'Rasis'])
plt.title('Confusion Matrix - Test Set')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

```
=== FINAL EVALUATION ON TEST SET ===
Best Model Performance (Test Set):
              precision    recall  f1-score   support

           0       0.85      0.81      0.82       149
           1       0.82      0.85      0.83       150

    accuracy                           0.83       299
   macro avg       0.83      0.83      0.83       299
weighted avg       0.83      0.83      0.83       299
```



In [20]:

```
# --- Step 15: Feature Importance Analysis ---
print("\n=== FEATURE IMPORTANCE ANALYSIS ===")
# Get feature names
feature_names = vectorizer.get_feature_names_out()

# Get log probabilities for each class
log_prob = best_model.feature_log_prob_

# Calculate difference between classes
feature_importance = log_prob[1] - log_prob[0]  # Class 1 vs Class 0

# Get top 20 features for each class
```

```
top_20_class0 = np.argsort(log_prob[0])[-20:]
top_20_class1 = np.argsort(log_prob[1])[-20:]

print("\nTop 20 fitur untuk kelas 'Tidak Rasis' (0):")
for idx in top_20_class0[::-1]:
    print(f"  {feature_names[idx]}: {log_prob[0][idx]:.4f}")

print("\nTop 20 fitur untuk kelas 'Rasis' (1):")
for idx in top_20_class1[::-1]:
    print(f"  {feature_names[idx]}: {log_prob[1][idx]:.4f}")#
```

```
=== FEATURE IMPORTANCE ANALYSIS ===

Top 20 fitur untuk kelas 'Tidak Rasis' (0):
  people: -4.3126
  woman: -4.4304
  hate: -4.6380
  think: -4.9622
  fucking: -5.0988
  muslim: -5.1488
  would: -5.1488
  shit: -5.2570
  country: -5.2860
  need: -5.3784
  society: -5.4451
  gay: -5.4802
  bitch: -5.5166
  would agree: -5.5166
  agree: -5.5166
  black: -5.5935
  fuck: -5.5935
  real: -5.6343
  like: -5.6769
  come: -5.6769

Top 20 fitur untuk kelas 'Rasis' (1):
  black: -3.6175
  people: -3.9974
  white: -4.3693
  like: -4.7213
  black people: -4.8197
  nigger: -4.9883
  zionist: -5.3538
  think: -5.3538
  dont: -5.4462
  society: -5.4462
  get: -5.5129
  country: -5.5129
  real: -5.5843
  one: -5.5843
  dark: -5.6221
  nigga: -5.6221
```

```
asian: -5.6613
good: -5.6613
white people: -5.6613
want: -5.6613
```

In [21]:

```
# --- Step 16: Sample Predictions (FIXED & CLEAN) ---
print("\n=== SAMPLE PREDICTIONS ===")

# Ambil 10 sampel acak dari test set
sample_indices = np.random.choice(X_test.shape[0], 10, replace=False)

for i, idx in enumerate(sample_indices):
    actual_label = "Rasis" if y_test.iloc[idx] == 1 else "Tidak Rasis"
    predicted_label = "Rasis" if y_pred_test[idx] == 1 else "Tidak Rasis"

    original_text = df_test.iloc[idx]['Comment']
    processed_text = df_test.iloc[idx]['Final_Text']

    print(f"\nSample {i+1}:")
    print(f"Original Text : {original_text}")
    print(f"Processed Text: {processed_text}")
    print(f"Actual Label  : {actual_label}")
    print(f"Predicted     : {predicted_label}")
    print(f"Correct       : {'✓' if y_test.iloc[idx] == y_pred_test[idx] else '✗'}")
```

```
=== SAMPLE PREDICTIONS ===

Sample 1:
Original Text : they even got the nigga
Processed Text: even got nigga
Actual Label  : Rasis
Predicted     : Rasis
Correct       : ✓

Sample 2:
Original Text : any normal person with a brain would agree that whatever comes out of rishi sunak's mouth is complete bullshit
Processed Text: normal person brain would agree whatever come rishi sunak mouth complete bullshit
Actual Label  : Tidak Rasis
Predicted     : Tidak Rasis
Correct       : ✓

Sample 3:
Original Text : government cuts are fucking up this country
Processed Text: government cut fucking country
Actual Label  : Tidak Rasis
Predicted     : Tidak Rasis
Correct       : ✓

Sample 4:
Original Text : I hate the democrats
Processed Text: hate democrat
Actual Label  : Tidak Rasis
Predicted     : Tidak Rasis
Correct       : ✓

Sample 5:
Original Text : When a Democrat and not a Republican bombs another country
Processed Text: democrat republican bomb another country
Actual Label  : Tidak Rasis
Predicted     : Tidak Rasis
Correct       : ✓

Sample 6:
Original Text : i want to kick trump in his fat head
Processed Text: want kick trump fat head
Actual Label  : Tidak Rasis
```

```
Predicted     : Tidak Rasis
Correct       : ✓

Sample 7:
Original Text : Mind, while Long ago we conquered our passions looking Of cursed nation. You must remember these tricks if I fe
ll in love
Processed Text: mind long ago conquered passion looking cursed nation must remember trick fell love
Actual Label  : Tidak Rasis
Predicted     : Tidak Rasis
Correct       : ✓

Sample 8:
Original Text : It was the largest slave revolt in modern times and a defining moment in the story of the Atlantic World. A bru
tal war that lasted over 12 years led to independence and the abolition of slavery in the former colony.
Processed Text: largest slave revolt modern time defining moment story atlantic world brutal war lasted year led independence a
bolition slavery former colony
Actual Label  : Tidak Rasis
Predicted     : Tidak Rasis
Correct       : ✓

Sample 9:
Original Text : @user shit i just realized your account isn't a month old. it's only a few weeks old. that's more like 110 twee
ts a day. just
Processed Text: shit realized account month old week old like tweet day
Actual Label  : Tidak Rasis
Predicted     : Tidak Rasis
Correct       : ✓

Sample 10:
Original Text : women shouldn't be allowed to exist
Processed Text: woman allowed exist
Actual Label  : Tidak Rasis
Predicted     : Tidak Rasis
Correct       : ✓
```

In [22]:

```python
# --- Step 17: Summary Report ---
print("\n" + "="*50)
print("SUMMARY REPORT")
print("="*50)

# Calculate metrics
```

```
val_accuracy = accuracy_score(y_val, y_pred_val)
test_accuracy = accuracy_score(y_test, y_pred_test)
val_f1 = f1_score(y_val, y_pred_val)
test_f1 = f1_score(y_test, y_pred_test)
val_precision = precision_score(y_val, y_pred_val)
test_precision = precision_score(y_test, y_pred_test)
val_recall = recall_score(y_val, y_pred_val)
test_recall = recall_score(y_test, y_pred_test)

print(f"\nData Statistics:")
print(f"  Total data after preprocessing: {len(df)}")
print(f"  Rasis comments: {df['Label'].value_counts()[1]}")
print(f"  Non-rasis comments: {df['Label'].value_counts()[0]}")
print(f"  Vocabulary size: {len(feature_names)}")

print(f"\nModel Performance:")
print(f"  Best hyperparameters: {grid_search.best_params_}")
print(f"\n  Validation Set:")
print(f"    Accuracy:  {val_accuracy:.4f}")
print(f"    F1-Score:  {val_f1:.4f}")
print(f"    Precision: {val_precision:.4f}")
print(f"    Recall:    {val_recall:.4f}")

print(f"\n  Test Set:")
print(f"    Accuracy:  {test_accuracy:.4f}")
print(f"    F1-Score:  {test_f1:.4f}")
print(f"    Precision: {test_precision:.4f}")
print(f"    Recall:    {test_recall:.4f}")

print(f"\n  Difference (Test - Validation):")
print(f"    Accuracy:  {test_accuracy - val_accuracy:+.4f}")
print(f"    F1-Score:  {test_f1 - val_f1:+.4f}")
```

```
==================================================
SUMMARY REPORT
==================================================

Data Statistics:
  Total data after preprocessing: 1987
  Rasis comments: 996
  Non-rasis comments: 991
  Vocabulary size: 1406

Model Performance:
  Best hyperparameters: {'alpha': 2.0, 'fit_prior': False}

  Validation Set:
    Accuracy:  0.8356
    F1-Score:  0.8383
    Precision: 0.8247
    Recall:    0.8523

  Test Set:
    Accuracy:  0.8294
    F1-Score:  0.8339
    Precision: 0.8153
    Recall:    0.8533

  Difference (Test - Validation):
    Accuracy:  -0.0061
    F1-Score:  -0.0044
```

In [23]:

```python
# --- Step 18: Save Results (FIXED VERSION) ---
print("\n=== SAVING RESULTS ===")

results_df = pd.DataFrame({
    'Original_Text': df_test['Comment'].values,
    'Processed_Text': df_test['Final_Text'].values,
    'Actual_Label': y_test.values,
    'Predicted_Label': y_pred_test
})

results_df['Correct'] = (
    results_df['Actual_Label'] == results_df['Predicted_Label']
)

results_df.to_csv('naive_bayes_predictions.csv', index=False)
print("Predictions saved to 'naive_bayes_predictions.csv'")
```

```
=== SAVING RESULTS ===
Predictions saved to 'naive_bayes_predictions.csv'
```

In [24]:

```
# --- Step 19: User Input Prediction ---
print("\n=== INTERACTIVE PREDICTION MODE ===")

def preprocess_user_input(text):
    # 1. cleaning
    text = clean_text(text)

    # 2. tokenization + stopword removal (split version)
    tokens = text.split()
    tokens = [
        word for word in tokens
        if word not in stop_words and len(word) > 1
    ]

    # 3. lemmatization
    tokens = [lemmatizer.lemmatize(word) for word in tokens]

    # 4. gabung lagi
    final_text = ' '.join(tokens)

    return final_text
```

```
=== INTERACTIVE PREDICTION MODE ===
```

In [ ]:

```
while True:
    user_input = input("\nMasukkan komentar (ketik 'exit' untuk keluar): ")

    if user_input.lower() == 'exit':
        print("Keluar dari mode eksperimen.")
        break

    # preprocess
    processed_input = preprocess_user_input(user_input)
    if processed_input.strip() == "":
        print("⚠️ Input tidak valid setelah preprocessing.")
        continue

    # vectorize (PAKAI VECTOR DARI TRAINING)
    input_vector = vectorizer.transform([processed_input])

    # prediction
    prediction = best_model.predict(input_vector)[0]
    prob = best_model.predict_proba(input_vector)[0]

    label = "RASIS 🚨" if prediction == 1 else "TIDAK RASIS ✅"

    print("\n--- HASIL PREDIKSI ---")
    print(f"Teks Asli     : {user_input}")
    print(f"Teks Diproses : {processed_input}")
    print(f"Prediksi      : {label}")
    print(f"Confidence    : Rasis={prob[1]:.2f}, Tidak Rasis={prob[0]:.2f}")
```

```
--- HASIL PREDIKSI ---
Teks Asli     : youre a cute little patootie
Teks Diproses : youre cute little patootie
Prediksi      : RASIS 🚨
Confidence    : Rasis=0.59, Tidak Rasis=0.41

--- HASIL PREDIKSI ---
Teks Asli     : clanker is bad race
Teks Diproses : clanker bad race
Prediksi      : RASIS 🚨
Confidence    : Rasis=0.81, Tidak Rasis=0.19

--- HASIL PREDIKSI ---
Teks Asli     : clanker are thw worst
Teks Diproses : clanker thw worst
Prediksi      : TIDAK RASIS ✅
Confidence    : Rasis=0.19, Tidak Rasis=0.81
```

In [ ]: