In [1]:
```
from google.colab import files
uploaded = files.upload()
```

Upload widget is only available when the cell has been executed in the current browser session. P
rerun this cell to enable.

```
Saving zepto_reviews_filtered.csv to zepto_reviews_filtered.csv
```

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
import re
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confus
from sklearn.preprocessing import LabelEncoder
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
df = pd.read_csv("zepto_reviews_filtered.csv", encoding='ISO-8859-1', on_l
df.dropna(subset=['content', 'score'], inplace=True)  # drop missing valu
df.head()
```

Out[3]:

	score	content
0	1	biggest scammer of the 21st century
1	5	nice product quality is good
2	4	price mismatch on every order
3	5	super ð
4	5	good

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 329414 entries, 0 to 329414
Data columns (total 2 columns):
 #   Column   Non-Null Count   Dtype
---  ------   --------------   -----
 0   score    329414 non-null  int64
 1   content  329414 non-null  object
dtypes: int64(1), object(1)
memory usage: 7.5+ MB
```

In [5]:
```
df.describe()
```

Out[5]:

	score
count	329414.000000
mean	3.621440
std	1.816529
min	1.000000
25%	1.000000
50%	5.000000
75%	5.000000
max	5.000000

In [6]:
```
df.isnull().sum() # 1 missing value in content
```

Out[6]:

	0
score	0
content	0

dtype: int64

In [7]:
```
def map_sentiment(score):
    if score >= 4:
        return 'positive'
    elif score == 3:
        return 'neutral'
    else:
        return 'negative'

df['sentiment'] = df['score'].apply(map_sentiment)
df['sentiment'].value_counts()
```

Out[7]:

sentiment	count
positive	215018
negative	107242
neutral	7154

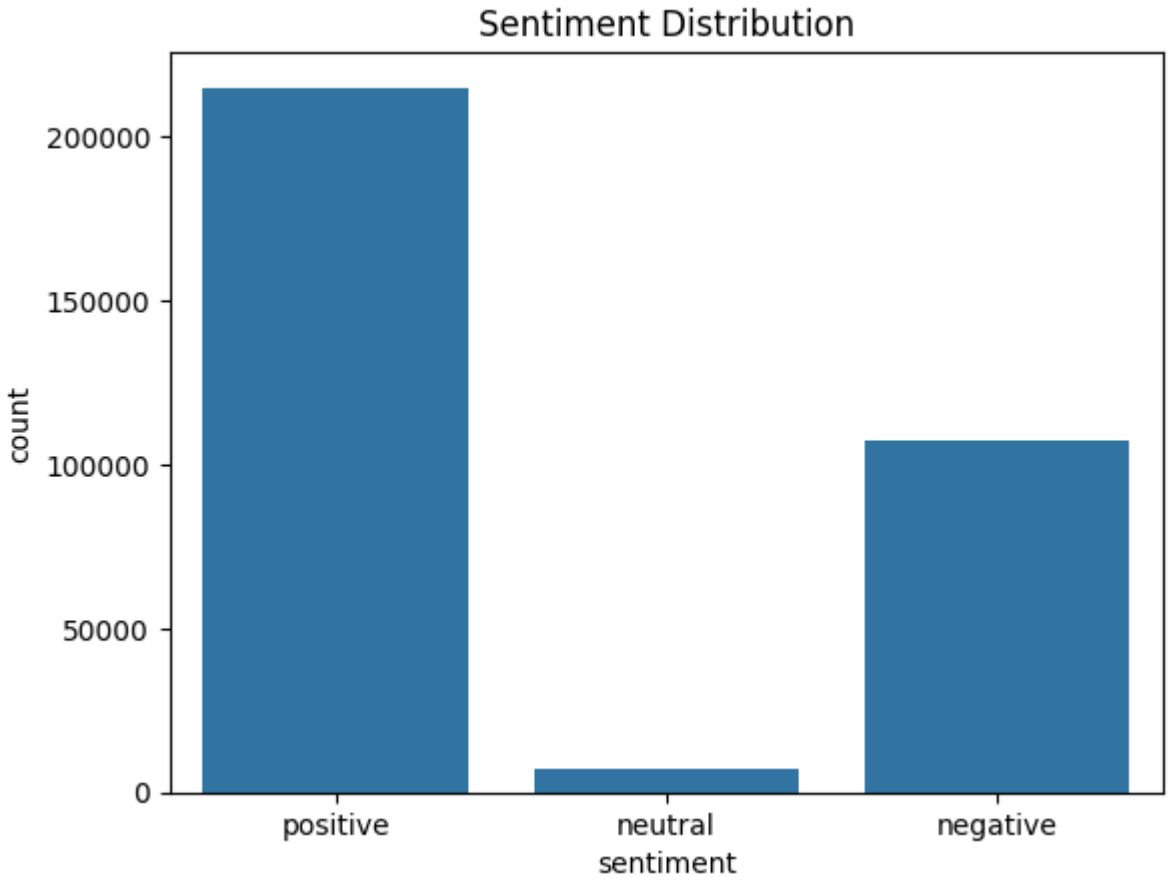
dtype: int64

In [8]:
```
def clean_text(text):
    text = text.lower()
    text = re.sub(r"http\S+|[^a-z\s]", "", text)
    text = re.sub(r"\s+", " ", text).strip()
    return text

df['cleaned_content'] = df['content'].apply(clean_text)
df[['cleaned_content', 'sentiment']].head()
```

Out[8]:

	cleaned_content	sentiment
0	biggest scammer of the st century	negative
1	nice product quality is good	positive
2	price mismatch on every order	positive
3	super	positive
4	good	positive

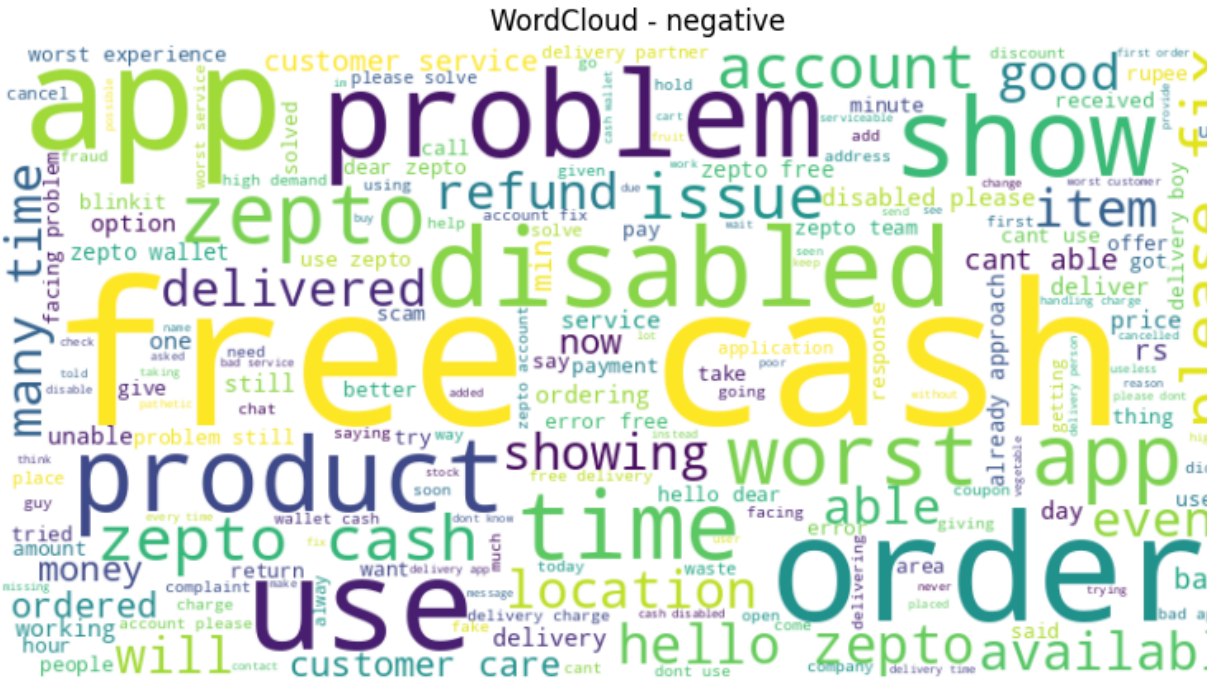
In [9]:
```
sns.countplot(x='sentiment', data=df, order=['positive', 'neutral', 'nega
plt.title("Sentiment Distribution")
plt.show()
```



In [10]:
```
for sentiment in ['positive', 'neutral', 'negative']:
    wc = WordCloud(width=800, height=400, background_color='white').genera
        ' '.join(df[df['sentiment'] == sentiment]['cleaned_content']))
    plt.figure(figsize=(10, 5))
    plt.imshow(wc, interpolation='bilinear')
    plt.axis('off')
    plt.title(f"WordCloud - {sentiment}")
    plt.show()
```

WordCloud - positive

WordCloud - neutral



Out[10]: ()

In [11]:
```
#  TF-IDF VECTORIZATION
tfidf = TfidfVectorizer(max_features=1000)
X = tfidf.fit_transform(df['cleaned_content'])  # Sparse format
```

In [12]:
```
df.head()
```

Out[12]:

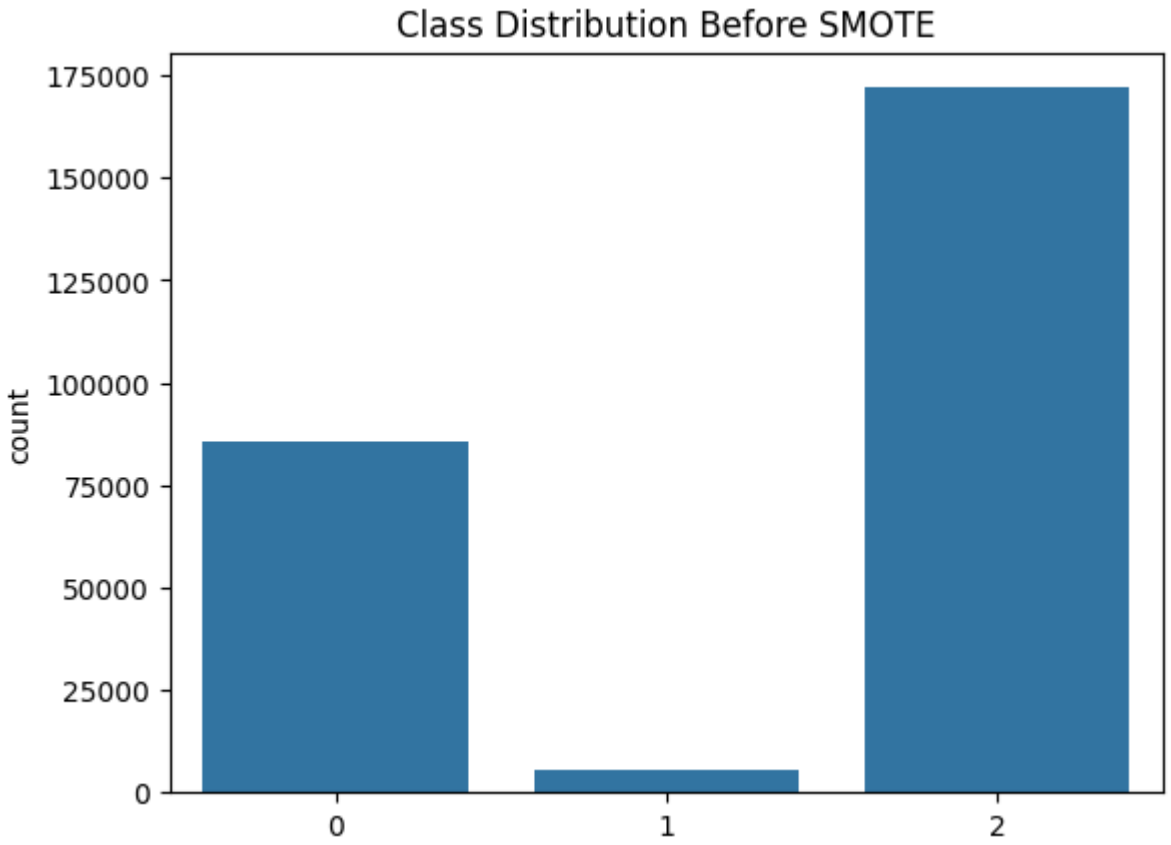
	score	content	sentiment	cleaned_content
0	1	biggest scammer of the 21st century	negative	biggest scammer of the st century
1	5	nice product quality is good	positive	nice product quality is good
2	4	price mismatch on every order	positive	price mismatch on every order
3	5	super ð	positive	super
4	5	good	positive	good

In [13]:
```
#  ENCODE TARGET
le = LabelEncoder()
y = le.fit_transform(df['sentiment'])  # 0 = negative, 1 = neutral, 2 = p
```

In [14]:
```
#  TRAIN-TEST SPLIT
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42)
```

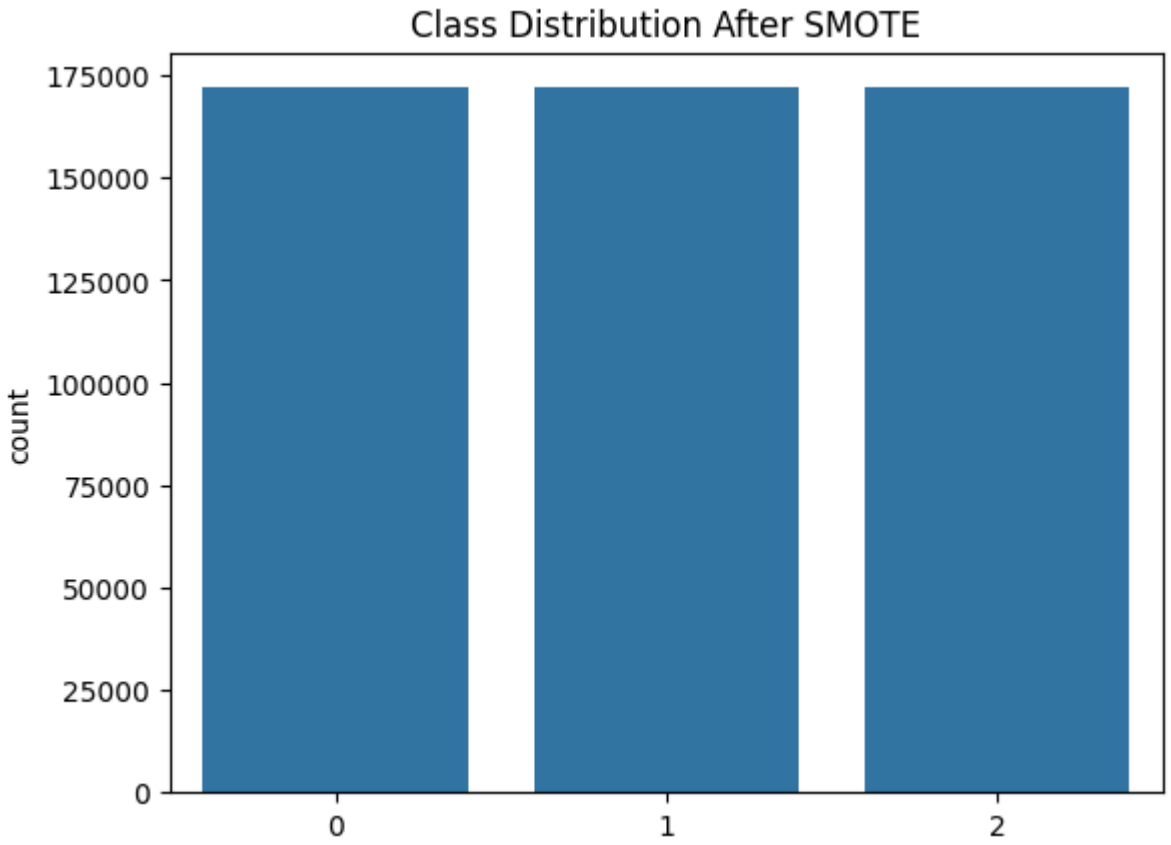
In [15]:
```
#  CHECK CLASS BALANCE
from collections import Counter

print("Before SMOTE:", Counter(y_train))
sns.countplot(x=y_train)
plt.title("Class Distribution Before SMOTE")
plt.show()
```

```
Before SMOTE: Counter({np.int64(2): 172014, np.int64(0): 85794, np.int64(1):
```



In [16]:
```
from imblearn.over_sampling import SMOTE

# Apply SMOTE to balance the training data
smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)

print("After SMOTE:", Counter(y_train_res))
sns.countplot(x=y_train_res)
plt.title("Class Distribution After SMOTE")
plt.show()
# Finally our data is balanced now
```

```
After SMOTE: Counter({np.int64(2): 172014, np.int64(0): 172014, np.int64(1):
172014})
```



In [28]:
```
model = LogisticRegression(class_weight='balanced', max_iter=1000)   #Tra
model.fit(X_train_res, y_train_res)
```

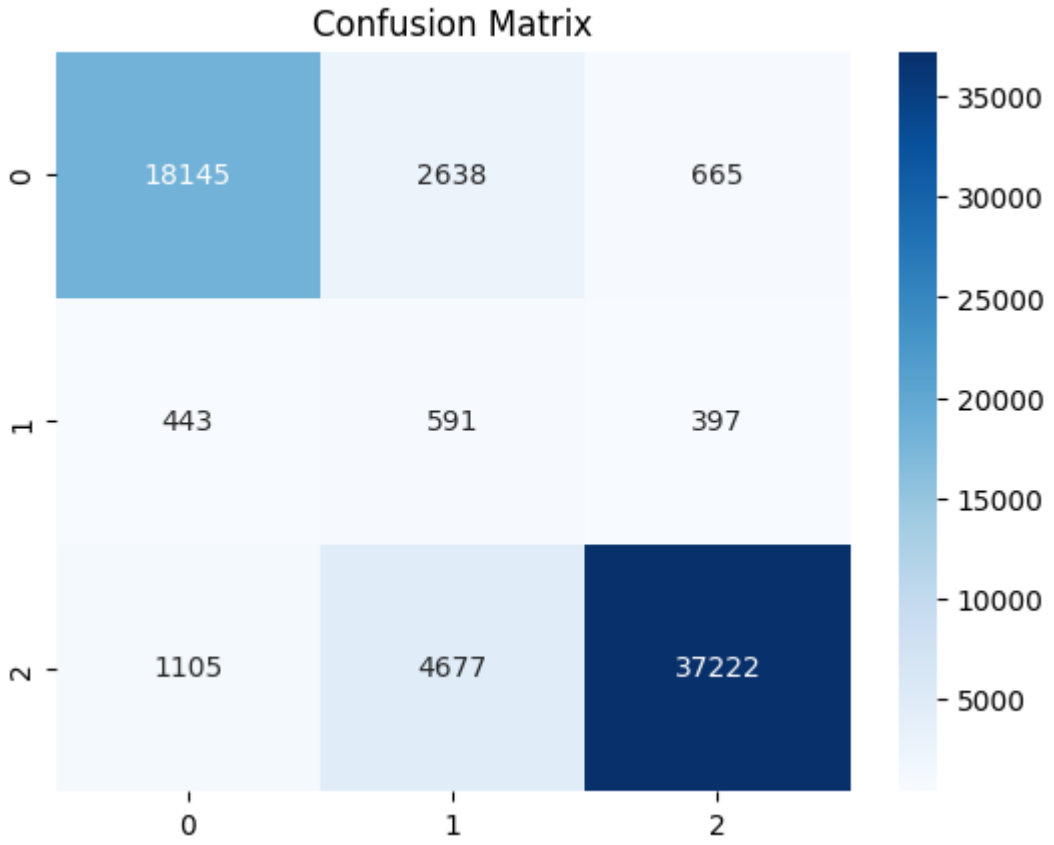
Out[28]:

LogisticRegression

```
LogisticRegression(class_weight='balanced', max_iter=1000)
```

In [39]:
```
from sklearn.metrics import classification_report, confusion_matrix, accu
import seaborn as sns
import matplotlib.pyplot as plt

# Predictions
y_pred = model.predict(X_test)

# Accuracy
print("    Accuracy:",accuracy_score(y_test, y_pred))

# Fix for target names
target_names = list(map(str, le.inverse_transform(np.unique(y_test))))

# Classification Report
print("\n    Classification Report\n", classification_report(y_test, y_pre

# Confusion Matrix
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d',
            xticklabels=target_names, yticklabels=target_names, cmap="Blu
plt.title("Confusion Matrix")
plt.show()
```

```
Accuracy: 0.8493541581288041

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.85      0.88     21448
           1       0.07      0.41      0.13      1431
           2       0.97      0.87      0.92     43004

    accuracy                           0.85     65883
   macro avg       0.66      0.71      0.64     65883
weighted avg       0.94      0.85      0.89     65883
```



In [48]:
```
from sklearn.naive_bayes import MultinomialNB   # naive bayes
from sklearn.metrics import accuracy_score, classification_report
import numpy as np

nb_model = MultinomialNB()
nb_model.fit(X_train_res, y_train_res)
y_pred_nb = nb_model.predict(X_test)

print(" Naive Bayes Accuracy:", accuracy_score(y_test, y_pred_nb))
print(classification_report(y_test, y_pred_nb, target_names=list(map(str,
```

```
Naive Bayes Accuracy: 0.8195133797793057
              precision    recall  f1-score   support

           0       0.81      0.84      0.82     21448
           1       0.08      0.39      0.13      1431
           2       0.98      0.82      0.89     43004

    accuracy                           0.82     65883
   macro avg       0.62      0.68      0.61     65883
weighted avg       0.90      0.82      0.85     65883
```

In [49]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

#     Reduce n_estimators and use n_jobs=-1 for parallel processing
rf_model = RandomForestClassifier(
    n_estimators=50,      # Use fewer trees (faster training)
    max_depth=15,         # Limit tree depth to avoid overfitting + impro
    random_state=42,
    n_jobs=-1             # Use all CPU cores to speed up
)

rf_model.fit(X_train_res, y_train_res)
y_pred_rf = rf_model.predict(X_test)

#     Convert class labels to string (if needed)
target_names = list(map(str, le.classes_))

#     Display metrics
print("    Random Forest Accuracy:"accuracy_score(y_test, y_pred_rf))
print("\n    Classification Report:\n", classification_report(y_test, y_pre
```

```
    Random Forest Accuracy: 0.8618156428820788

    Classification Report:
                  precision    recall  f1-score   support

           0       0.88      0.78      0.83     21448
           1       0.08      0.22      0.12      1431
           2       0.92      0.92      0.92     43004

    accuracy                           0.86     65883
   macro avg       0.63      0.64      0.62     65883
weighted avg       0.89      0.86      0.87     65883
```

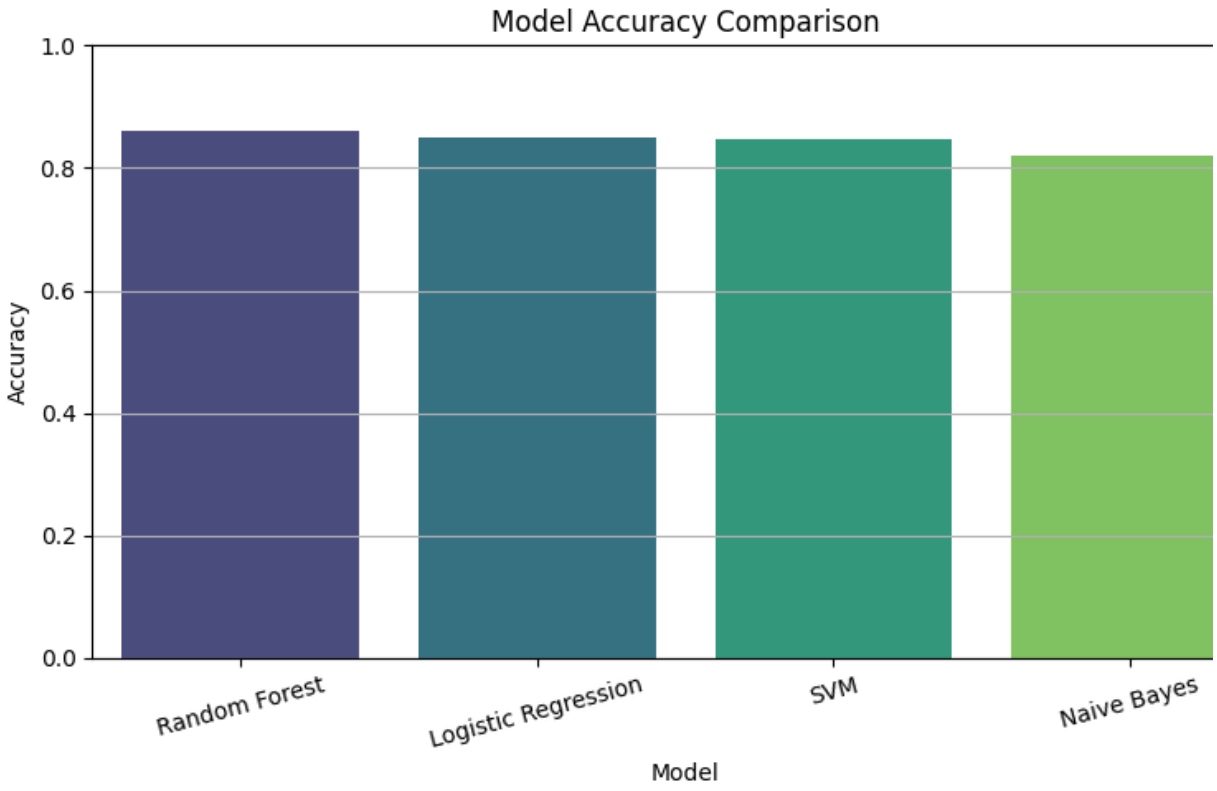
In [50]:
```
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score, classification_report

#     Fast version of SVM
svm_model = LinearSVC(max_iter=3000, dual=False)  # dual=False is faster

# Fit the model
svm_model.fit(X_train_res, y_train_res)

# Predict
y_pred_svm = svm_model.predict(X_test)

#     Convert encoded labels back to original class names (if needed)
target_names = list(map(str, le.classes_))

# Accuracy & Report
print("    SVM Accuracy:",accuracy_score(y_test, y_pred_svm))
print("\n    Classification Report:\n", classification_report(y_test, y_pred
```

```
    SVM Accuracy: 0.8475782827132948

    Classification Report:
                  precision    recall  f1-score   support

               0       0.91      0.85      0.88     21448
               1       0.07      0.40      0.12      1431
               2       0.97      0.86      0.91     43004

        accuracy                           0.85     65883
       macro avg       0.65      0.70      0.64     65883
    weighted avg       0.93      0.85      0.89     65883
```

In [52]:
```
results_df = pd.DataFrame({
    'Model': ['Logistic Regression', 'Naive Bayes', 'Random Forest', 'SVM
    'Accuracy': [
        accuracy_score(y_test, y_pred),  # Use y_pred from logistic regres
        accuracy_score(y_test, y_pred_nb),   # from naive bayes
        accuracy_score(y_test, y_pred_rf),   # from random forest
        accuracy_score(y_test, y_pred_svm)   # from svm
    ]
})

results_df = results_df.sort_values(by='Accuracy', ascending=False)
results_df.reset_index(drop=True, inplace=True)
print(results_df)
```

```
                 Model  Accuracy
0        Random Forest  0.861816
1  Logistic Regression  0.849354
2                  SVM  0.847578
3          Naive Bayes  0.819513
```

In [53]:
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 5))
sns.barplot(data=results_df, x='Model', y='Accuracy', palette='viridis')
plt.title('Model Accuracy Comparison')
plt.ylabel('Accuracy')
plt.ylim(0, 1)
plt.xticks(rotation=15)
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```



In [57]:
```
from sklearn.ensemble import RandomForestClassifier

# Train a fast and reasonably accurate model directly
fast_rf_model = RandomForestClassifier(
    n_estimators=50,        # fewer trees = faster
    max_depth=10,           # limit depth
    min_samples_split=2,
    min_samples_leaf=1,
    random_state=42,
    n_jobs=-1
)

# Fit model
fast_rf_model.fit(X_train_res, y_train_res)

# Predict and evaluate
y_pred = fast_rf_model.predict(X_test)
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
                precision    recall  f1-score   support

           0       0.87      0.74      0.80     21448
           1       0.08      0.22      0.12      1431
           2       0.91      0.92      0.91     43004

    accuracy                           0.85     65883
   macro avg       0.62      0.63      0.61     65883
weighted avg       0.88      0.85      0.86     65883
```

In [58]:
```
import joblib

# Save the model to a file
joblib.dump(fast_rf_model, 'random_forest_model.pkl')
```

Out[58]:
```
['random_forest_model.pkl']
```

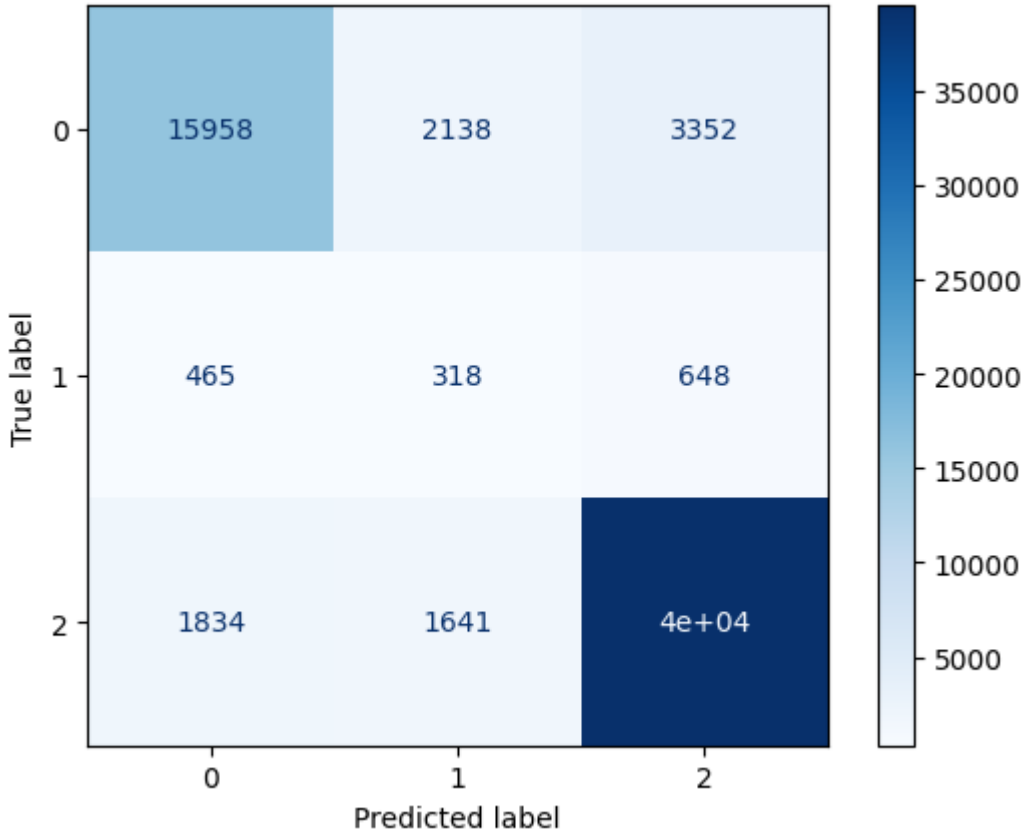
In [59]:
```
loaded_model = joblib.load('random_forest_model.pkl')
```

In [60]:
```
def predict_sentiment(text, vectorizer, model):
    # Convert text to feature vector
    text_vector = vectorizer.transform([text])
    # Predict sentiment
    prediction = model.predict(text_vector)
    return prediction[0]
```

In [62]:
```
import pandas as pd

# Create DataFrame with actual and predicted
results_df = pd.DataFrame({
    'Actual': y_test,
    'Predicted': y_pred
})

# Save to CSV
results_df.to_csv('rf_predictions.csv', index=False)
```

In [63]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=fast_rf_
disp.plot(cmap='Blues')
```

Out[63]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7dbc10ad
```



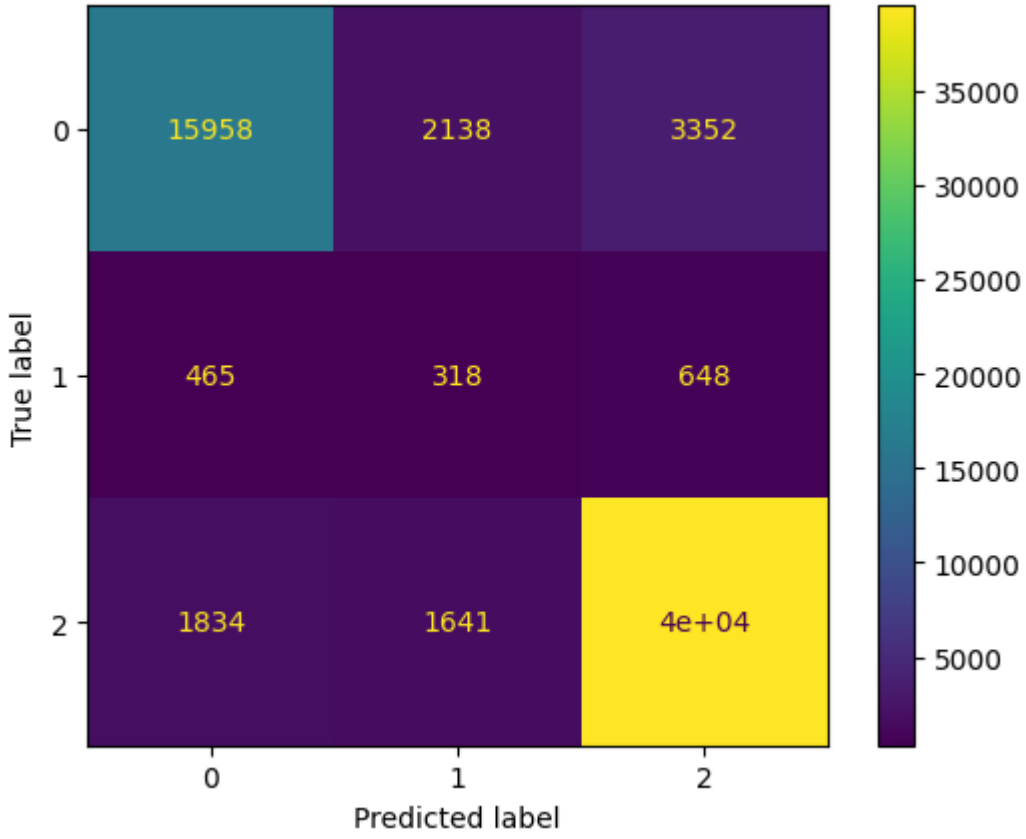
In [64]:
```
from sklearn.metrics import classification_report

report = classification_report(y_test, y_pred, output_dict=True)
report_df = pd.DataFrame(report).transpose()
report_df.to_csv("classification_report_rf.csv")
report_df.head()
```

Out[64]:

	precision	recall	f1-score	support
0	0.874076	0.744032	0.803828	21448.000000
1	0.077618	0.222222	0.115051	1431.000000
2	0.908107	0.919194	0.913617	43004.000000
accuracy	0.847032	0.847032	0.847032	0.847032
macro avg	0.619934	0.628483	0.610832	65883.000000

In [65]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
```

Out[65]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7dbc10a4
```


15958
2138
3352
465
318
648
1834
1641
4e+04
0
1
2
True label
0
1
2
Predicted label
35000
30000
25000
20000
15000
10000
5000

In [ ]: