European Restaurant

REVIEWS

Start Slide →

Team Members

Sahab Al-Amri

Fatima Al-Amri

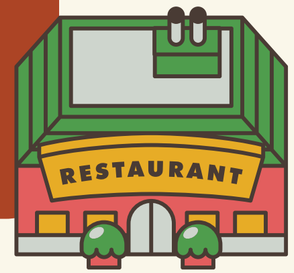
Ashgan Al Shamali

Table Of Content

Problem Definition & Objective

- **Problem**: Manual analysis of restaurant reviews is inefficient due to their large volume.
- **Type**: Binary classification (Positive / Negative).
- **Objective**: Build a machine learning model to classify review sentiment.
- **Method**: Natural Language Processing (NLP) techniques were used with TF-IDF for text representation.

  Three machine learning models were implemented and compared:
  - Naive Bayes
  - Logistic Regression with class_weight = 'balanced'
  - Gradient Boosting
- **Evaluation**: Assess performance before and after optimization.



Dataset Collection & Description

Old dataset

Restaurant reviews

- Number of records: 10,000
- Columns:Restaurant,Reviewer,Review,Rating,Metadata,Time,Pictures,7514

New dataset

European Restaurant Reviews

- Number of records: 1,502
- Columns: Country,Restaurant Name,Sentiment,Review Title,Review Date, Review

Data Cleaning & Preprocessing


```
----------------------------------------
the shape of the data is :  (1502, 6)
----------------------------------------
check if there any null value:  Country
Restaurant Name    0
Sentiment          0
Review Title       0
Review Date        0
Review             0
dtype: int64
```

```
df = df.dropna(subset=['Review', 'Sentiment'])

text_col = "Review"
target_col = "Sentiment"
df
```

- The dataset was inspected for missing and null values.
- No missing values were found in the main columns used for analysis.
- Therefore, no extensive data cleaning was required.
- Basic preprocessing steps were applied, such as selecting relevant columns and ensuring text data was in the correct format for analysis.

```
# 2) Feature Engineering (Numeric Features)
# ================================================
df['review_length'] = df['Review'].apply(len)
df['word_count'] = df['Review'].apply(lambda x: len(x.split()))
df['exclamation_count'] = df['Review'].apply(lambda x: x.count('!'))
```

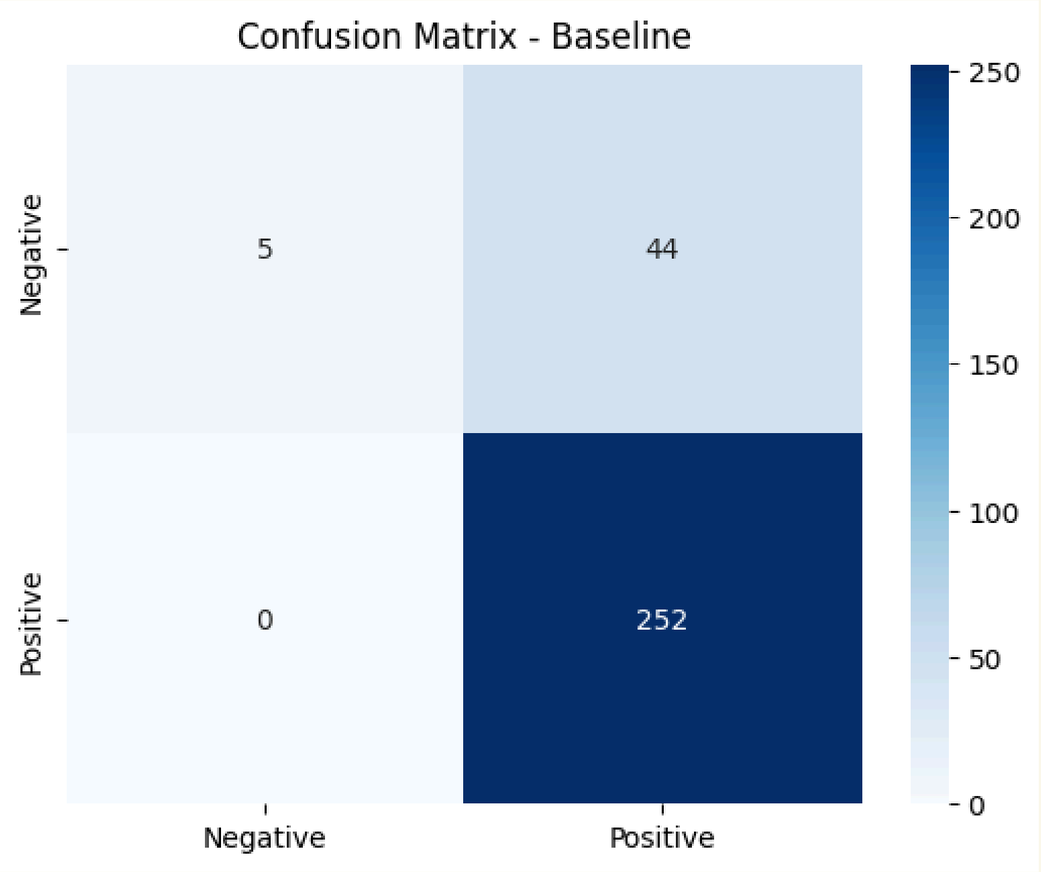
```
X_text = df[text_col]
X_num = df[['review_length','word_count','exclamation_count']].values
y = df[target_col]

X_train_text, X_test_text, X_train_num, X_test_num, y_train, y_test = train_test_split(
    X_text, X_num, y, test_size=0.2, random_state=42
)
```

```
# ================================================
# 4) Text Vectorization (TF-IDF)
# ================================================
tfidf = TfidfVectorizer(stop_words='english', max_features=5000, ngram_range=(1,2))
X_train_tfidf = tfidf.fit_transform(X_train_text)
X_test_tfidf = tfidf.transform(X_test_text)

# Combine TF-IDF with numeric features
X_train_combined = hstack([X_train_tfidf, X_train_num])
X_test_combined = hstack([X_test_tfidf, X_test_num])
```

```
# 5) Model Training (Naive Bayes)
# ================================================
nb_model = MultinomialNB(alpha=1.0)
nb_model.fit(X_train_combined, y_train)
```

```
# ================================================
# 6) Model Evaluation
# ================================================
y_pred = nb_model.predict(X_test_combined)
print("Naive Bayes Test Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred, labels=nb_model.classes_)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=nb_model.classes_, yticklabels=nb_model.classes_)
plt.title("Confusion Matrix - Baseline")
plt.show()
```

```
Naive Bayes Test Accuracy: 0.8538205980066446
              precision    recall  f1-score   support

    Negative       1.00      0.10      0.19        49
    Positive       0.85      1.00      0.92       252

    accuracy                           0.85       301
   macro avg       0.93      0.55      0.55       301
weighted avg       0.88      0.85      0.80       301
```

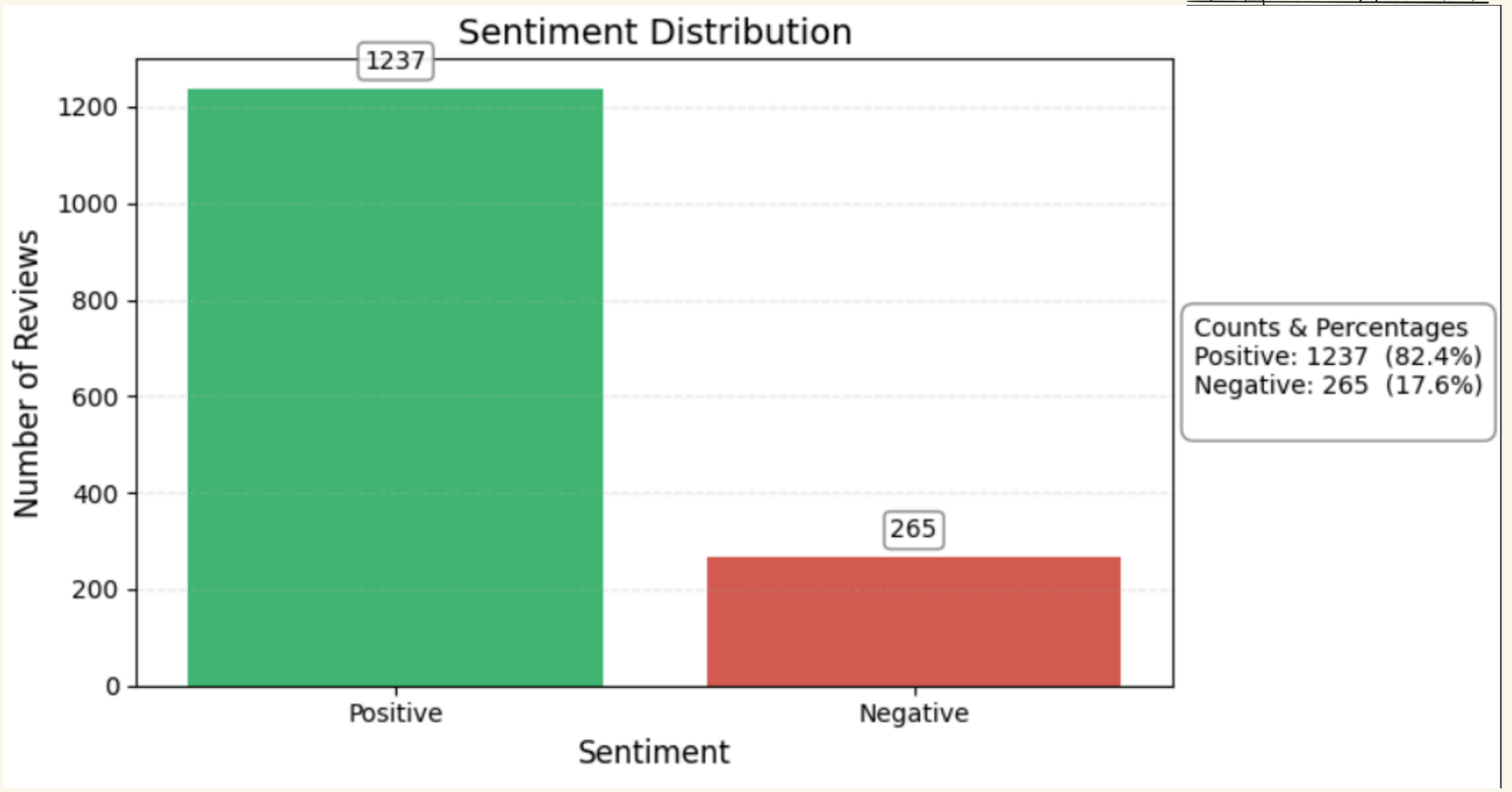


- **The baseline model achieved an accuracy of 85%.**
- **It performed very well on positive reviews.**
- **It struggled to detect negative reviews.**
- **The performance was imbalanced, requiring model improvement.**

Exploratory Data Analysis (EDA)


```python
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8.5, 4.5))  # مساحة أكبر

# Order + custom colors for classes
order = df["Sentiment"].value_counts().index
palette = {"Positive": "#2ecc71", "Negative": "#e74c3c"}  # أخضر/أحمر

ax = sns.countplot(x="Sentiment", data=df, order=order, palette=palette)

# Add counts on top of bars (inside small boxes)
for p in ax.patches:
    ax.annotate(
        f"{int(p.get_height())}",
        (p.get_x() + p.get_width() / 2., p.get_height()),
        ha="center", va="bottom", fontsize=10,
        xytext=(0, 6), textcoords="offset points",
        bbox=dict(boxstyle="round,pad=0.25", fc="white", ec="gray", alpha=0.9)
    )

# Title and labels
plt.title("Sentiment Distribution", fontsize=14)
plt.xlabel("Sentiment", fontsize=12)
plt.ylabel("Number of Reviews", fontsize=12)

# Light grid
plt.grid(axis="y", linestyle="--", alpha=0.25)

        # ---- Side box with counts + percentages ----
        counts = df["Sentiment"].value_counts()
        total = len(df)

        info_text = "Counts & Percentages\n"
        for cls in order:
            c = counts[cls]
            pct = (c / total) * 100
            info_text += f"{cls}: {c}  ({pct:.1f}%)\n"

        # Make space on the right for the box
        plt.subplots_adjust(right=0.78)

        # Add the side box
        plt.gca().text(
            1.02, 0.5, info_text,
            transform=plt.gca().transAxes,
            fontsize=10, va="center",
            bbox=dict(boxstyle="round,pad=0.5", fc="white", ec="gray", alpha=0.9)
        )

        plt.tight_layout()
        plt.show()
```

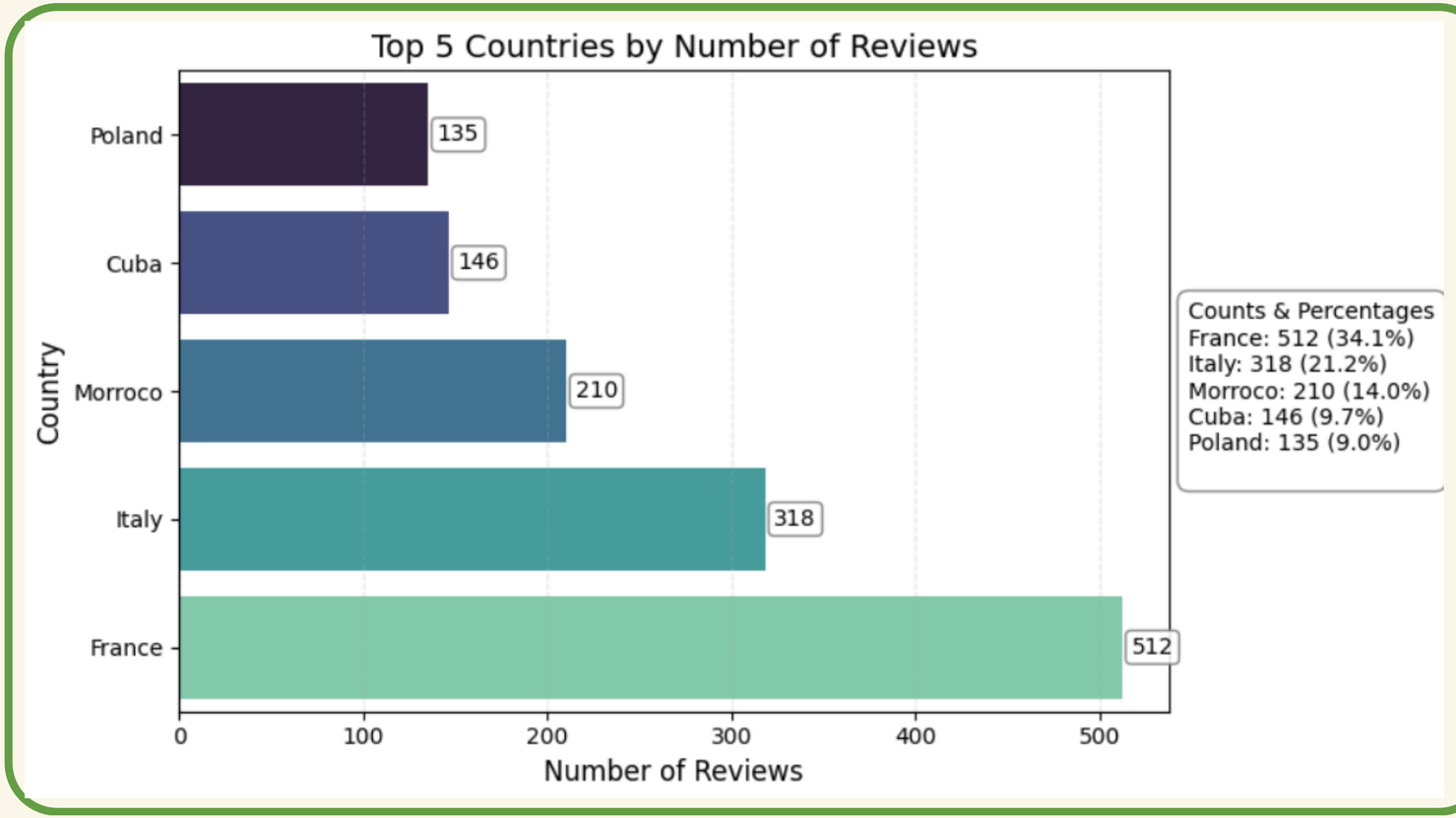
This plot visualizes the distribution of positive and negative reviews, highlighting the class imbalance in the dataset



This chart shows the top 5 countries by number of reviews.

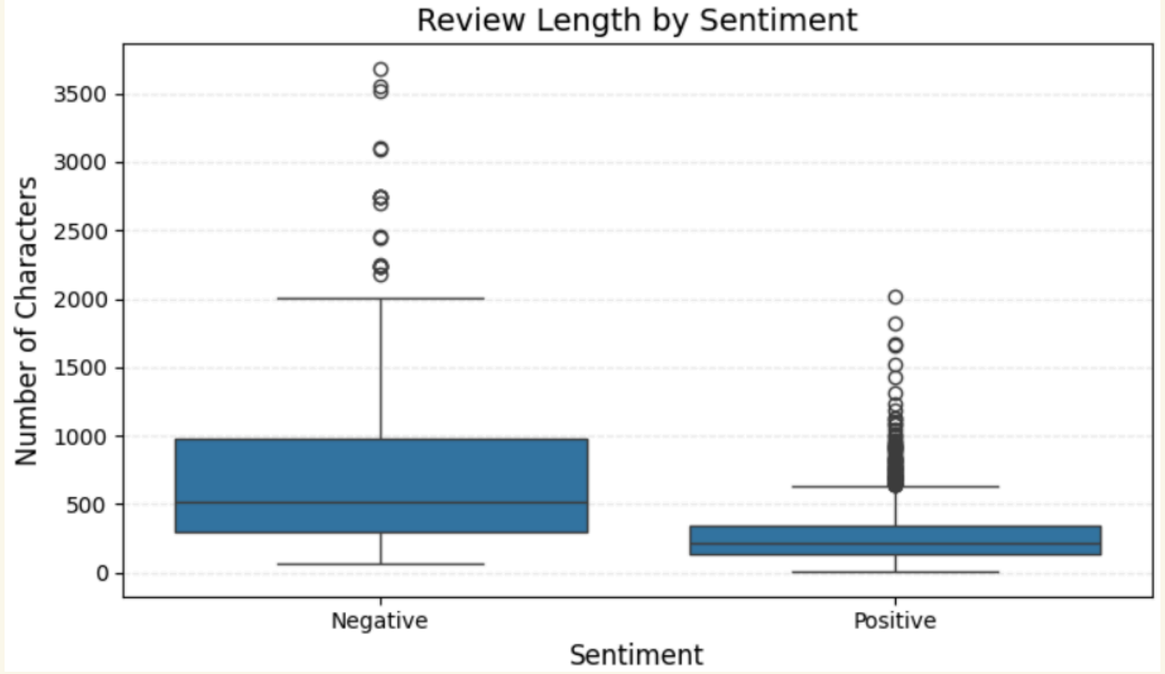

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
top = df["Country"].value_counts().head(5).sort_values()  # sort for nice horizontal order
total = len(df)

plt.figure(figsize=(9, 5))
ax = sns.barplot(x=top.values, y=top.index, palette="mako")  # clean palette

# numbers on bars (boxed)
for i, v in enumerate(top.values):
    ax.text(
        v + (top.values.max() * 0.01), i, f"{v}",
        va="center", fontsize=10,
        bbox=dict(boxstyle="round,pad=0.25", fc="white", ec="gray", alpha=0.9)
    )

plt.title("Top 5 Countries by Number of Reviews", fontsize=14)
plt.xlabel("Number of Reviews", fontsize=12)
plt.ylabel("Country", fontsize=12)
plt.grid(axis="x", linestyle="--", alpha=0.25)

# side box with percentages
info_text = "Counts & Percentages\n"
for c, v in top.sort_values(ascending=False).items():
    pct = (v / total) * 100
    info_text += f"{c}: {v} ({pct:.1f}%)\n"

plt.subplots_adjust(right=0.78)
plt.gca().text(
    1.02, 0.5, info_text,
    transform=plt.gca().transAxes,
    fontsize=10, va="center",
    bbox=dict(boxstyle="round,pad=0.5", fc="white", ec="gray", alpha=0.9)
)
plt.tight_layout()
plt.show()
```



This boxplot compares review lengths between Positive and Negative sentiments


```
plt.figure(figsize=(7.5,4.5))
ax = sns.boxplot(x="Sentiment", y="review_length", data=df)

plt.title("Review Length by Sentiment", fontsize=14)
plt.xlabel("Sentiment", fontsize=12)
plt.ylabel("Number of Characters", fontsize=12)
plt.grid(axis="y", linestyle="--", alpha=0.25)
plt.tight_layout()
plt.show()
```

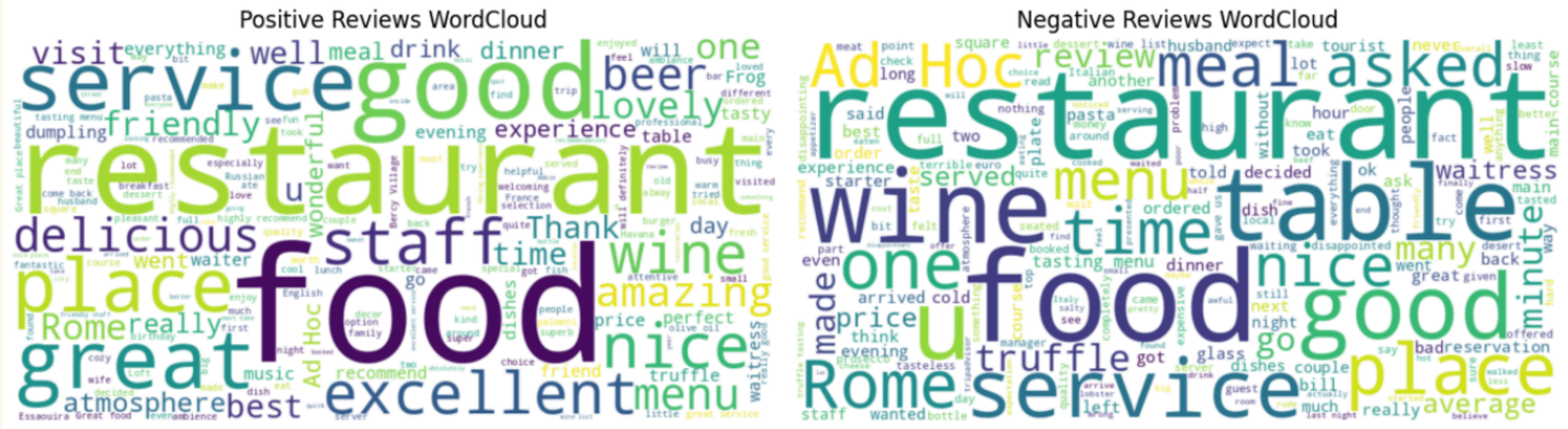


Word Cloud Analysis


```python
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt

# Make sure Review is string
df["Review"] = df["Review"].astype(str)

# Split text by sentiment
pos_text = " ".join(df[df["Sentiment"] == "Positive"]["Review"])
neg_text = " ".join(df[df["Sentiment"] == "Negative"]["Review"])

# Create wordclouds
wc_pos = WordCloud(
    width=900, height=450,
    background_color="white",
    stopwords=STOPWORDS
).generate(pos_text)

wc_neg = WordCloud(
    width=900, height=450,
    background_color="white",
    stopwords=STOPWORDS
).generate(neg_text)

# Display side by side
plt.figure(figsize=(12,5))

plt.subplot(1,2,1)
plt.imshow(wc_pos, interpolation="bilinear")
plt.axis("off")
plt.title("Positive Reviews WordCloud")
```

```python
plt.subplot(1,2,2)
plt.imshow(wc_neg, interpolation="bilinear")
plt.axis("off")
plt.title("Negative Reviews WordCloud")

plt.tight_layout()
plt.show()
```

This slide shows the most frequent words in positive and negative reviews. Word size represents frequency and helps identify key sentiment patterns.



Positive and negative word counts were computed based on the presence of predefined sentiment words within each review.


```python
# ========================
# Feature Engineering (Enhanced)
# ========================
df['review_length'] = df['Review'].apply(len)  # length of the review
df['word_count'] = df['Review'].apply(lambda x: len(x.split()))  # number of words
df['exclamation_count'] = df['Review'].apply(lambda x: x.count('!'))  # number of '!'
df['question_count'] = df['Review'].apply(lambda x: x.count('?'))  # number of '?'

# Simple positive and negative word lists
positive_words = ['amazing', 'excellent', 'good', 'nice', 'delicious', 'perfect', 'best']
negative_words = ['bad', 'terrible', 'horrible', 'awful', 'disappointing', 'worst']

df['pos_word_count'] = df['Review'].apply(lambda x: sum(word in x.lower() for word in positive_words))  # count of positive words
df['neg_word_count'] = df['Review'].apply(lambda x: sum(word in x.lower() for word in negative_words))  # count of negative words

num_features = ['review_length','word_count','exclamation_count','question_count','pos_word_count','neg_word_count']

#  Print output to see results
print("New feature columns added:", num_features)
print("\n Sample rows (Review + features):")
print(df[['Review'] + num_features].head(5))

print("\n Quick stats for features:")
print(df[num_features].describe())
```

```
New feature columns added: ['review_length', 'word_count', 'exclamation_count', 'question_count',
'pos_word_count', 'neg_word_count']

Sample rows (Review + features):
                                              Review  review_length  \
0  The manager became agressive when I said the c...            146
1  I ordered a beef fillet ask to be done medium,...            281
2  This is an attractive venue with welcoming, al...            260
3  Sadly I  used the high TripAdvisor rating too ...           1513
4  From the start this meal was bad- especially g...           1351

   word_count  exclamation_count  question_count  pos_word_count  \
0          28                 0               0               1
1          58                 0               0               0
2          40                 0               0               0
3         279                 0               0               4
4         243                 0               0               2

   neg_word_count
0               2
1               0
2               0
3               0
4               2

Quick stats for features:
       review_length   word_count  exclamation_count  question_count  \
count    1502.000000  1502.000000        1502.000000     1502.000000
mean      366.438083    66.131158           0.855526        0.069907
std       399.792336    74.008747           1.731406        0.496262
min        10.000000     2.000000           0.000000        0.000000
25%       147.000000    26.000000           0.000000        0.000000
50%       236.000000    42.500000           0.000000        0.000000
75%       416.750000    74.000000           1.000000        0.000000
max      3679.000000   646.000000          26.000000       12.000000

       pos_word_count  neg_word_count
count     1502.000000     1502.000000
mean         1.153795        0.091877
std          0.955880        0.366230
min          0.000000        0.000000
25%          0.000000        0.000000
50%          1.000000        0.000000
75%          2.000000        0.000000
max          6.000000        3.000000
```

```
## Split the dataset into training and testing sets (text + numeric features)
# =========================
#  Split Dataset
# =========================
X_text = df['Review']  # text feature
X_num = df[num_features].values  # numerical features
y = df['Sentiment']  # target

X_train_text, X_test_text, X_train_num, X_test_num, y_train, y_test = train_test_split(
    X_text, X_num, y, test_size=0.2, random_state=42
)
```

Convert text reviews into TF-IDF features, then combine with numeric features

```
# =========================
# TF-IDF Vectorization (Enhanced)
# =========================

# Ensure text is string (safe)
X_train_text = X_train_text.astype(str)
X_test_text  = X_test_text.astype(str)

# TF-IDF settings (enhanced)
tfidf = TfidfVectorizer(
    stop_words="english",
    max_features=8000,
    ngram_range=(1, 3),
    min_df=5,
    max_df=0.8
)

# Fit on training text, transform both train and test
X_train_tfidf = tfidf.fit_transform(X_train_text)
X_test_tfidf  = tfidf.transform(X_test_text)

# Combine TF-IDF features with numerical features
X_train_combined = hstack([X_train_tfidf, X_train_num])
X_test_combined  = hstack([X_test_tfidf, X_test_num])
```

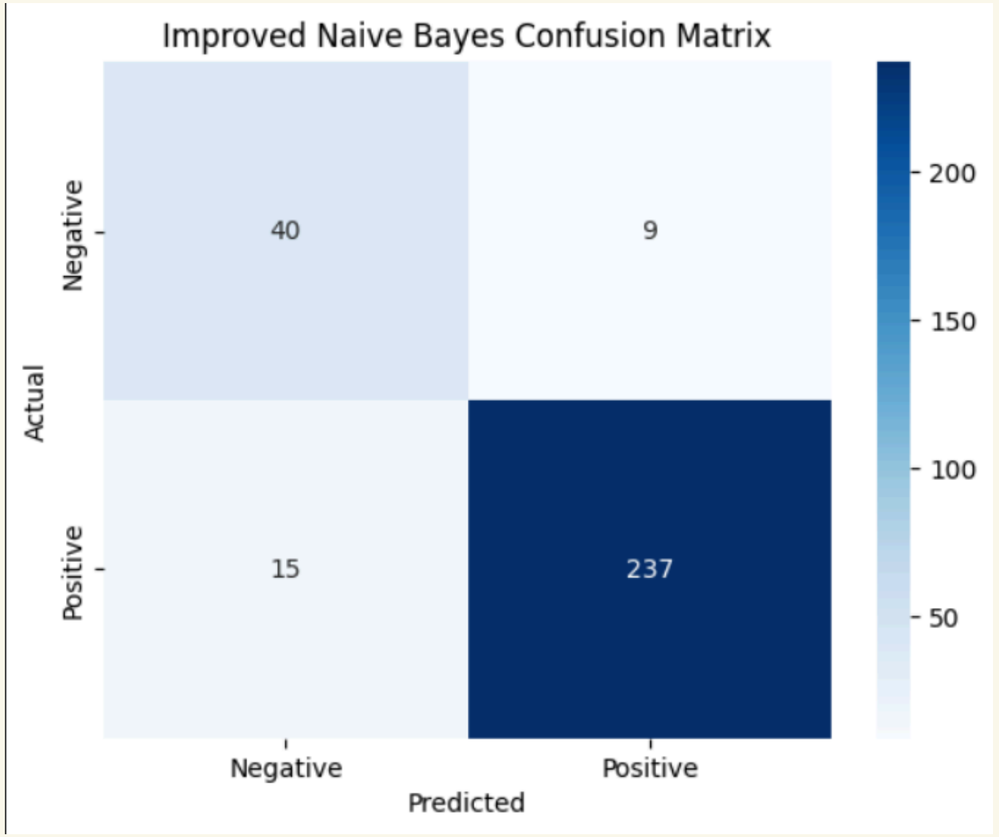
```
# ========================
#  Train Naive Bayes Model
# ========================

# Train the classifier on the combined TF-IDF + numeric features
nb_model = MultinomialNB(alpha=0.5)  # smoothing parameter
nb_model.fit(X_train_combined, y_train)
```

MultinomialNB

MultinomialNB(alpha=0.5)

```
#  Evaluation
# ========================
y_pred = nb_model.predict(X_test_combined)

print("Improved Naive Bayes Test Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Confusion Matrix (keep class order consistent)
cm = confusion_matrix(y_test, y_pred, labels=nb_model.classes_)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=nb_model.classes_, yticklabels=nb_model.classes_)
plt.title("Improved Naive Bayes Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Improved Naive Bayes Test Accuracy: 0.920265780730897
              precision    recall  f1-score   support

    Negative       0.73      0.82      0.77        49
    Positive       0.96      0.94      0.95       252

    accuracy                           0.92       301
   macro avg       0.85      0.88      0.86       301
weighted avg       0.92      0.92      0.92       301
```



Prepare data for GridSearch (DataFrame format)

Model Improvement


```python
# Create a DataFrame that contains the text column + numeric feature columns
X_train_df = pd.concat(
    [X_train_text.reset_index(drop=True),
     pd.DataFrame(X_train_num, columns=num_features)],
    axis=1
)
X_train_df.columns = ["Review"] + num_features

X_test_df = pd.concat(
    [X_test_text.reset_index(drop=True),
     pd.DataFrame(X_test_num, columns=num_features)],
    axis=1
)
X_test_df.columns = ["Review"] + num_features


# ========================
#  Custom model wrapper (TF-IDF + numeric features + Naive Bayes)
# ========================
# This custom estimator allows GridSearchCV to tune TF-IDF and NB parameters together
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.model_selection import GridSearchCV

class TfidfHstackNB(BaseEstimator, ClassifierMixin):
    def __init__(self, ngram_range=(1,3), max_features=8000, min_df=5, max_df=0.8,
                 sublinear_tf=False, alpha=0.5, fit_prior=True):
        self.ngram_range = ngram_range
        self.max_features = max_features
        self.min_df = min_df
        self.max_df = max_df
        self.sublinear_tf = sublinear_tf
        self.alpha = alpha
        self.fit_prior = fit_prior
```

```python
    def fit(self, X, y):
        # Split input into text and numeric features
        X_text = X["Review"].astype(str)
        X_num  = X[num_features].values

        # Convert text to TF-IDF features
        self.tfidf_ = TfidfVectorizer(
            stop_words="english",
            max_features=self.max_features,
            ngram_range=self.ngram_range,
            min_df=self.min_df,
            max_df=self.max_df,
            sublinear_tf=self.sublinear_tf
        )

        # Combine TF-IDF with numeric features
        X_tfidf = self.tfidf_.fit_transform(X_text)
        X_combined = hstack([X_tfidf, X_num])

        # Train Naive Bayes classifier
        self.nb_ = MultinomialNB(alpha=self.alpha, fit_prior=self.fit_prior)
        self.nb_.fit(X_combined, y)
        return self

    def predict(self, X):
        # Transform input using the fitted TF-IDF, then predict with NB
        X_text = X["Review"].astype(str)
        X_num  = X[num_features].values

        X_tfidf = self.tfidf_.transform(X_text)
        X_combined = hstack([X_tfidf, X_num])
        return self.nb_.predict(X_combined)
```

Define GridSearch parameters

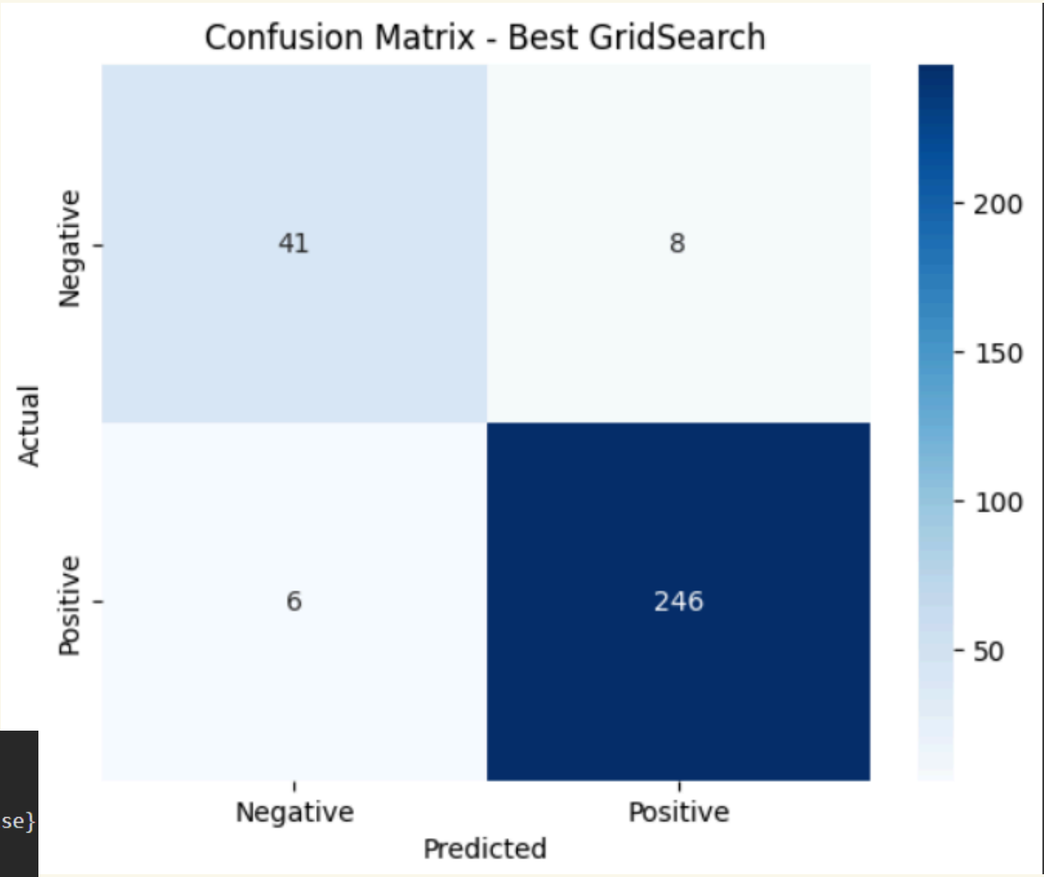
```
# Search over different TF-IDF and Naive Bayes settings
param_grid = {
    "ngram_range": [(1,1), (1,2), (1,3)],
    "max_features": [5000, 8000, 12000],
    "min_df": [2, 5],
    "max_df": [0.8, 0.9],
    "sublinear_tf": [True, False],
    "alpha": [0.1, 0.3, 0.5, 1.0],
    "fit_prior": [True, False],
}
```

Run GridSearchCV

```
# Use cross-validation to find the best combination based on F1-macro
grid = GridSearchCV(
    estimator=TfidfHstackNB(),
    param_grid=param_grid,
    scoring="f1_macro",
    cv=5,
    n_jobs=-1,
    verbose=1
)

grid.fit(X_train_df, y_train)

print("\n===== BEST PARAMS =====")
print(grid.best_params_)
print("Best CV F1-macro:", grid.best_score_)
```

Evaluate the best model on the test set


```
best_model = grid.best_estimator_
y_pred_gs = best_model.predict(X_test_df)

print("\n===== TEST RESULTS (GridSearch Best) =====")
print("Accuracy:", accuracy_score(y_test, y_pred_gs))
print(classification_report(y_test, y_pred_gs))

cm = confusion_matrix(y_test, y_pred_gs, labels=best_model.nb_.classes_)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=best_model.nb_.classes_, yticklabels=best_model.nb_.classes_)
plt.title("Confusion Matrix - Best GridSearch")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



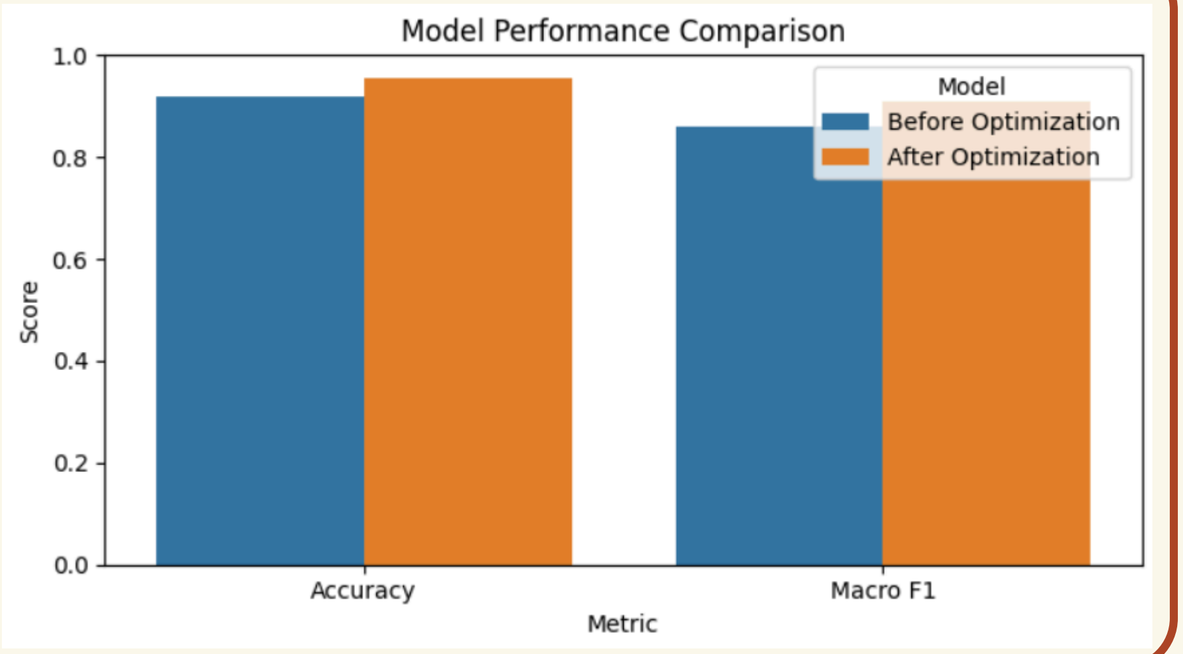
===== TEST RESULTS (GridSearch Best) =====
Accuracy: 0.9534883720930233

```
Fitting 5 folds for each of 576 candidates, totalling 2880 fits

===== BEST PARAMS =====
{'alpha': 0.1, 'fit_prior': False, 'max_df': 0.8, 'max_features': 8000, 'min_df': 2, 'ngram_range': (1, 3), 'sublinear_tf': False}
Best CV F1-macro: 0.9117063443663312

===== TEST RESULTS (GridSearch Best) =====
Accuracy: 0.9534883720930233
              precision    recall  f1-score   support

    Negative       0.87      0.84      0.85        49
    Positive       0.97      0.98      0.97       252

    accuracy                           0.95       301
   macro avg       0.92      0.91      0.91       301
weighted avg       0.95      0.95      0.95       301
```

Model Performance Comparison

This chart compares the model's performance before and after optimization using Accuracy and Macro F1 score. The results show a clear improvement in overall performance after optimization.

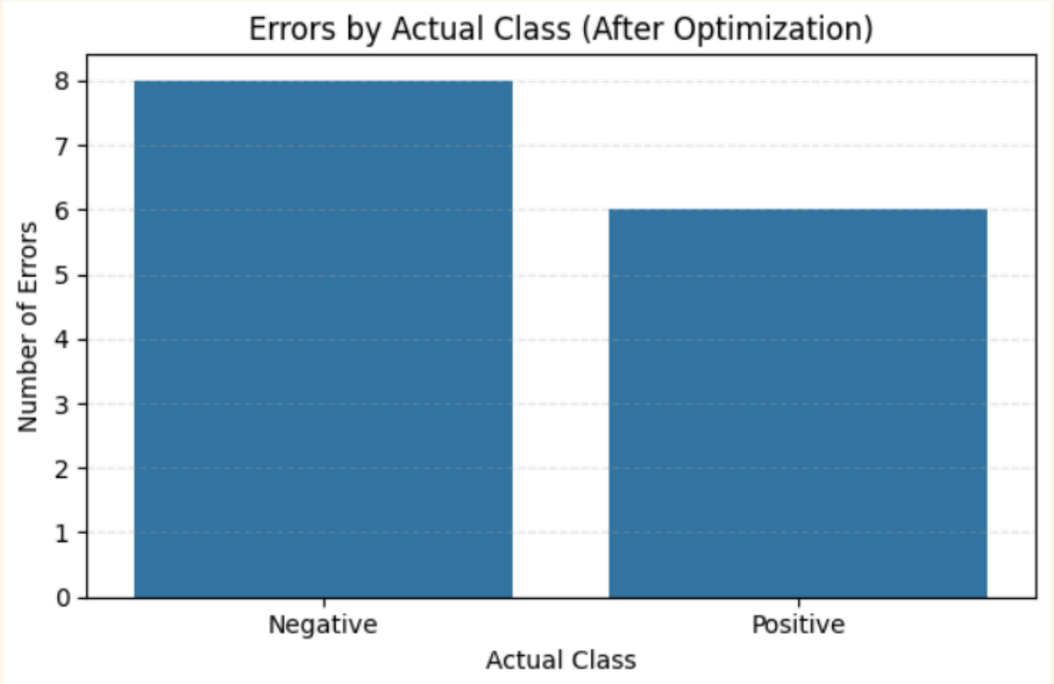
```python
comparison_df = pd.DataFrame({
    "Model": ["Before Optimization", "After Optimization"],
    "Accuracy": [0.9203, 0.9535],
    "Macro F1": [0.86, 0.91]
})

comparison_df_melt = comparison_df.melt(id_vars="Model",
                                        value_vars=["Accuracy", "Macro F1"],
                                        var_name="Metric",
                                        value_name="Score")

plt.figure(figsize=(7,4))
sns.barplot(x="Metric", y="Score", hue="Model", data=comparison_df_melt)
plt.ylim(0,1)
plt.title("Model Performance Comparison")
plt.tight_layout()
plt.show()
```



Error Distribution by Actual Class (After Optimization)

This chart shows the distribution of errors based on the actual class after model optimization. It highlights the number of misclassifications per class.

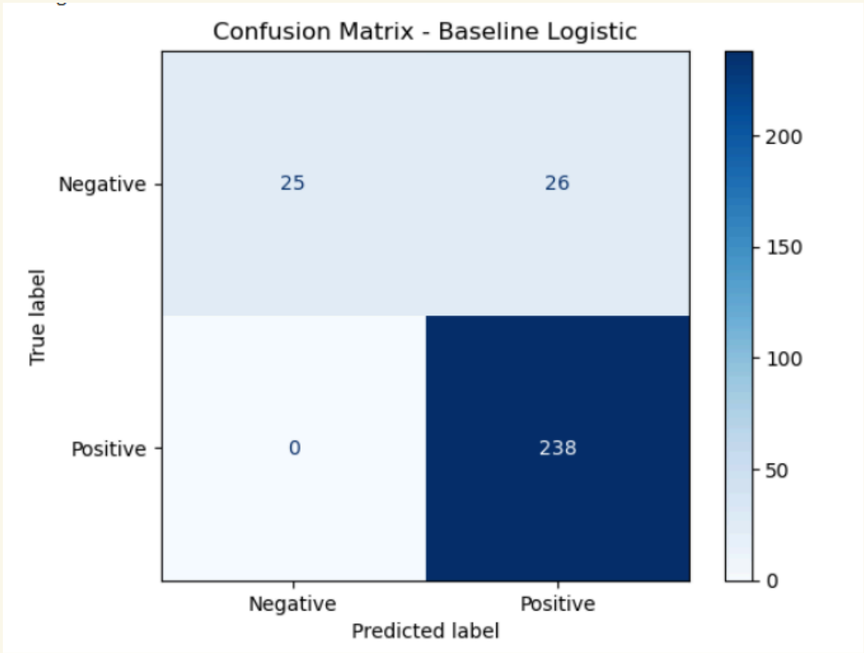
```
errors = (y_test != y_pred_gs)

err_df = pd.DataFrame({
    "Actual": y_test.values,
    "Predicted": y_pred_gs,
    "Error": errors
})

plt.figure(figsize=(6,4))
sns.countplot(data=err_df[err_df["Error"] == True], x="Actual")
plt.title("Errors by Actual Class (After Optimization)")
plt.xlabel("Actual Class")
plt.ylabel("Number of Errors")
plt.grid(axis="y", linestyle="--", alpha=0.3)
plt.tight_layout()
plt.show()
```



LOGISTIC REGRESSION


```python
# ======================================================
# 5) TF-IDF + BASELINE LOGISTIC REGRESSION (NO CLASS WEIGHT)
# ======================================================

tfidf = TfidfVectorizer(
    stop_words="english",
    max_features=5000,
    ngram_range=(1, 2)
)

# Transform text into TF-IDF features
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

# Baseline Logistic Regression (no class_weight)
logit_baseline = LogisticRegression(max_iter=500)
logit_baseline.fit(X_train_tfidf, y_train)

# Predictions
y_pred_base = logit_baseline.predict(X_test_tfidf)

print("\n===== BASELINE MODEL (No Class Weight) =====")
base_acc = accuracy_score(y_test, y_pred_base)
print("Accuracy:", base_acc)

print("\nClassification Report (Baseline):")
print(classification_report(y_test, y_pred_base))
cm_base = confusion_matrix(y_test, y_pred_base)
print("\nConfusion Matrix (Baseline):")
print(cm_base)

plt.figure(figsize=(4, 4))
ConfusionMatrixDisplay(cm_base, display_labels=logit_baseline.classes_).plot(
    cmap="Blues", values_format="d"
)
plt.title("Confusion Matrix - Baseline Logistic")
plt.show()
```

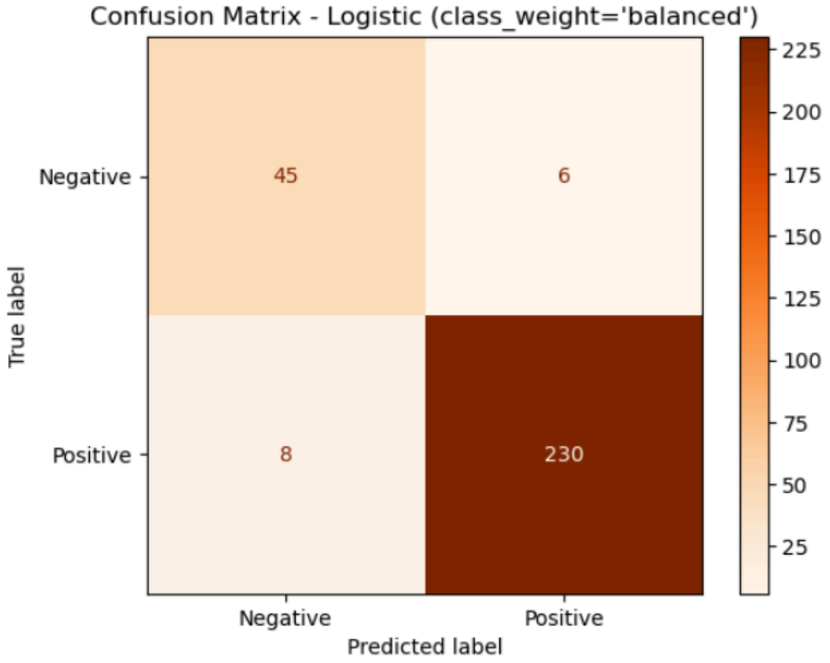
```
===== BASELINE MODEL (No Class Weight) =====
Accuracy: 0.9100346020761245

Classification Report (Baseline):
              precision    recall  f1-score   support

    Negative       1.00      0.49      0.66        51
    Positive       0.90      1.00      0.95       238

    accuracy                           0.91       289
   macro avg       0.95      0.75      0.80       289
weighted avg       0.92      0.91      0.90       289


Confusion Matrix (Baseline):
[[ 25  26]
 [  0 238]]
<Figure size 400x400 with 0 Axes>
```



LOGISTIC REGRESSION


```python
# ======================================================
# 6) LOGISTIC REGRESSION WITH CLASS WEIGHT = 'BALANCED'
# ======================================================

logit_balanced = LogisticRegression(
    max_iter=500,
    class_weight="balanced"
)

logit_balanced.fit(X_train_tfidf, y_train)
y_pred_balanced = logit_balanced.predict(X_test_tfidf)

print("\n===== MODEL WITH CLASS WEIGHT (Balanced) =====")
bal_acc = accuracy_score(y_test, y_pred_balanced)
print("Accuracy:", bal_acc)

print("\nClassification Report (Balanced):")
print(classification_report(y_test, y_pred_balanced))

cm_bal = confusion_matrix(y_test, y_pred_balanced)
print("\nConfusion Matrix (Balanced):")
print(cm_bal)
plt.figure(figsize=(4, 4))
ConfusionMatrixDisplay(cm_bal, display_labels=logit_balanced.classes_).plot(
    cmap="Oranges", values_format="d"
)
plt.title("Confusion Matrix - Logistic (class_weight='balanced')")
plt.show()
```

```
===== MODEL WITH CLASS WEIGHT (Balanced) =====
Accuracy: 0.9515570934256056

Classification Report (Balanced):
              precision    recall  f1-score   support

    Negative       0.85      0.88      0.87        51
    Positive       0.97      0.97      0.97       238

    accuracy                           0.95       289
   macro avg       0.91      0.92      0.92       289
weighted avg       0.95      0.95      0.95       289


Confusion Matrix (Balanced):
[[ 45   6]
 [  8 230]]
```



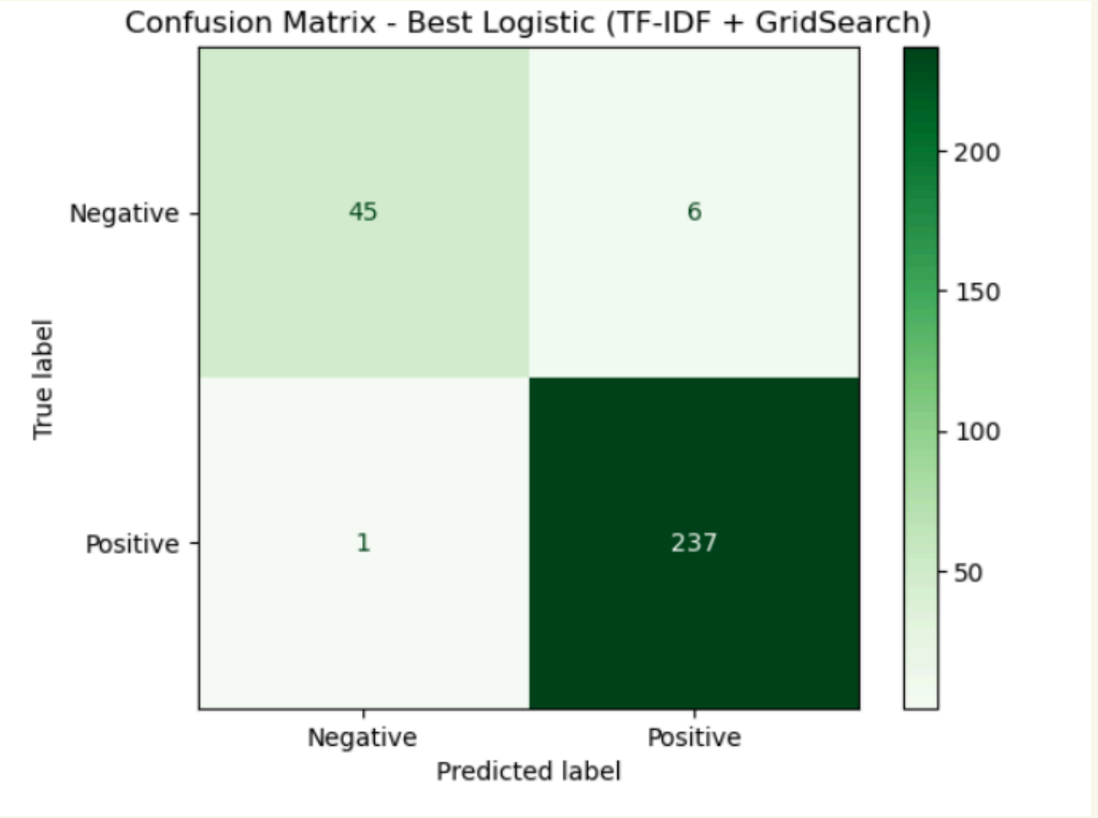
LOGISTIC REGRESSION


```python
# ====================================================
# 7) TUNED LOGISTIC (TF-IDF + GRIDSEARCHCV)
# ====================================================

# Build pipeline: TF-IDF + Logistic Regression
pipe = Pipeline([
    ("tfidf", TfidfVectorizer()),
    ("clf", LogisticRegression(max_iter=500))
])

# Grid of hyperparameters to try
param_grid = {
    # TF-IDF settings
    "tfidf__stop_words": ["english"],
    "tfidf__ngram_range": [(1, 1), (1, 2)],   # unigrams vs unigrams+bigrams
    "tfidf__max_features": [3000, 5000, 8000],

    # Logistic Regression settings
    "clf__C": [0.1, 1, 10],                   # regularization strength
    "clf__class_weight": ["balanced"],
    "clf__solver": ["liblinear"],             # good for small/medium data
}
grid = GridSearchCV(
    pipe,
    param_grid,
    cv=5,
    scoring="f1_macro",   # balance performance between classes
    n_jobs=-1,
    verbose=2
)

grid.fit(X_train, y_train)
```

```python
print("\n===== BEST PARAMS FROM GRIDSEARCH =====")
print(grid.best_params_)
print("Best CV Score (F1-macro):", grid.best_score_)

# Evaluate best model
best_model = grid.best_estimator_
y_pred_best = best_model.predict(X_test)
print("\n===== TEST PERFORMANCE (BEST LOGISTIC MODEL) =====")
best_acc = accuracy_score(y_test, y_pred_best)
print("Accuracy:", best_acc)

print("\nClassification Report (Best Logistic):")
print(classification_report(y_test, y_pred_best))

cm_best = confusion_matrix(y_test, y_pred_best)
print("\nConfusion Matrix (Best Logistic):")
print(cm_best)

plt.figure(figsize=(4, 4))
ConfusionMatrixDisplay(cm_best, display_labels=best_model.classes_).plot(
    cmap="Greens", values_format="d"
)
plt.title("Confusion Matrix - Best Logistic (TF-IDF + GridSearch)")
plt.show()
```

LOGISTIC REGRESSION

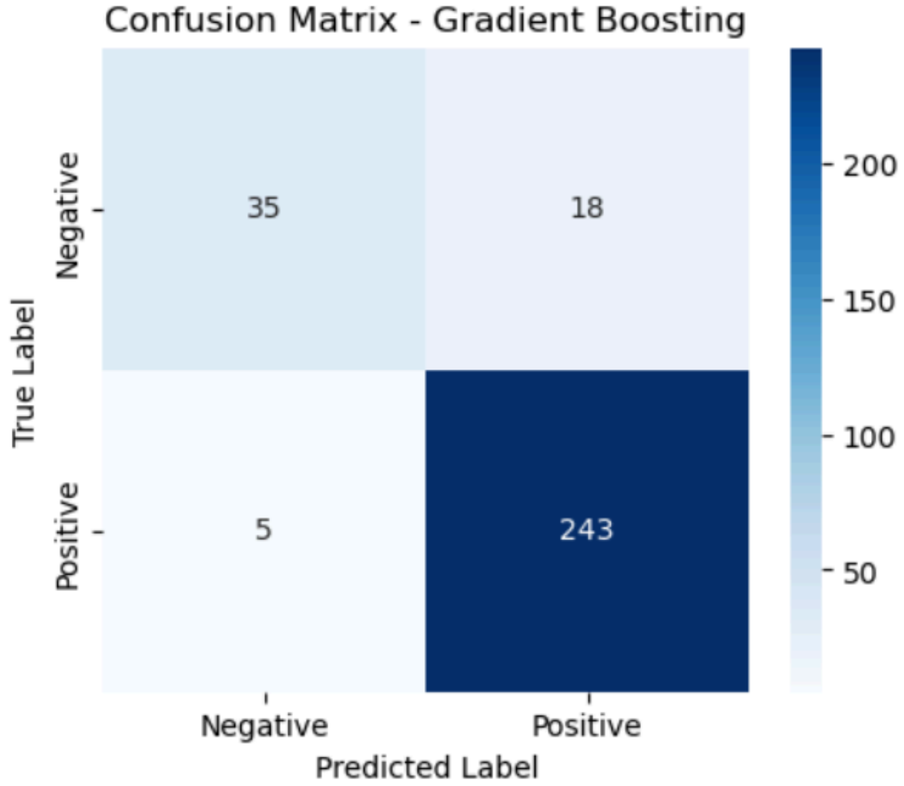

```
===== TEST PERFORMANCE (BEST LOGISTIC MODEL) =====
Accuracy: 0.9757785467128027

Classification Report (Best Logistic):
              precision    recall  f1-score   support

    Negative       0.98      0.88      0.93        51
    Positive       0.98      1.00      0.99       238

    accuracy                           0.98       289
   macro avg       0.98      0.94      0.96       289
weighted avg       0.98      0.98      0.98       289
```



Gradient Boosting


```python
# 1. Import libraries
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report, accuracy_score

# 2. Define features and target
X = df["Review"]
y = df["Sentiment"]

# 3. Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# 4. Convert text to numerical features using TF-IDF
tfidf = TfidfVectorizer(max_features=5000, stop_words='english')
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)

# 5. Create Gradient Boosting model
gb_model = GradientBoostingClassifier(
    n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42
)

# 6. Train the model
gb_model.fit(X_train_tfidf, y_train)

# 7. Predict on test data
y_pred = gb_model.predict(X_test_tfidf)

# 8. Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.9235880398671097
              precision    recall  f1-score   support

    Negative       0.88      0.66      0.75        53
    Positive       0.93      0.98      0.95       248

    accuracy                           0.92       301
   macro avg       0.90      0.82      0.85       301
weighted avg       0.92      0.92      0.92       301
```



Gradient Boosting


```python
#Text cleaning
def clean_text(text):
    text = text.lower()
    text = re.sub(r'[^a-zA-Z\s]', ' ', text)  # remove symbols
    text = re.sub(r'\s+', ' ', text)          # remove extra spaces
    return text

df["Review"] = df["Review"].apply(clean_text)
```

```python
text_col = "Review"
target_col = "Sentiment"

X = df[text_col]
y = df[target_col]
```

```python
# 3. Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

```python
#Converting text to numbers using TF-IDF
tfidf = TfidfVectorizer(
    max_features=15000,
    ngram_range=(1, 3),
    stop_words='english',
    min_df=2
)

X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)
```

```python
#  Feature selection
selector = SelectKBest(chi2, k=8000)
X_train_sel = selector.fit_transform(X_train_tfidf, y_train)
X_test_sel = selector.transform(X_test_tfidf)
```

```python
#Class imbalance handling
class_weights = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(y_train),
    y=y_train
)
weights_dict = {c: w for c, w in zip(np.unique(y_train), class_weights)}
```

```python
# RandomizedSearchCV for Gradient Boosting
from scipy.stats import randint

param_dist = {
    'n_estimators': randint(150, 300),
    'learning_rate': [0.05, 0.1],
    'max_depth': randint(3, 5),
    'subsample': [0.8, 1.0],
    'min_samples_leaf': [1, 3]
}

rand_search = RandomizedSearchCV(
    GradientBoostingClassifier(random_state=42),
    param_distributions=param_dist,
    n_iter=20,
    cv=3,
    scoring='accuracy',
    n_jobs=-1,
    random_state=42
)
```

Gradient Boosting


```python
#Train the model with class weights
rand_search.fit(
    X_train_sel,
    y_train,
    sample_weight=[weights_dict[i] for i in y_train]
)

print("Best Params:", rand_search.best_params_)
print("Best CV Accuracy:", rand_search.best_score_)
```

```
Best Params: {'learning_rate': 0.1, 'max_depth': 4, 'min_samples_leaf': 3, 'n_estimators': 260, 'subsample': 1.0}
Best CV Accuracy: 0.9358790523690773
```

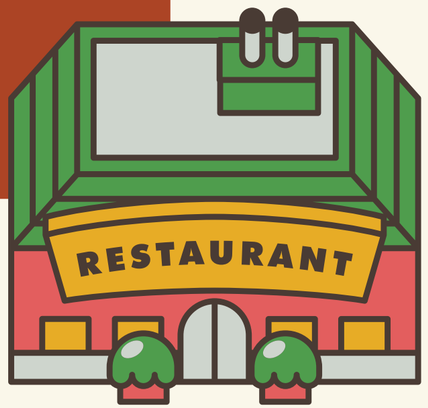
```python
# Final model with increased n_estimators
best_model = rand_search.best_estimator_
best_model.set_params(n_estimators=best_model.n_estimators * 2)
best_model.fit(X_train_sel, y_train)

# Evaluate
y_pred = best_model.predict(X_test_sel)

print("Test Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Test Accuracy: 0.9401993355481728
              precision    recall  f1-score   support

    Negative       0.89      0.75      0.82        53
    Positive       0.95      0.98      0.96       248

    accuracy                           0.94       301
   macro avg       0.92      0.87      0.89       301
weighted avg       0.94      0.94      0.94       301
```

Insight

- Model Performance Comparison (Accuracy)

	Accuracy	Performance
Logistic Regression	97%	Achieved the highest and most stable performance in text classification
Naive Bayes	95%	Fast and simple model with good baseline performance.
Gradient Boosting	94%	Powerful model but did not outperform the others in this task.



THANK YOU