In [1]:
```python
import pandas as pd
```

In [2]:
```python
# Load the dataset
file_path = r'C:\Users\zahus\Desktop\DATA science\Module 11-Dissertaion\Dataset\6-Sentiment Analysis Dataset (Customer Feedback)
df = pd.read_csv(file_path, delimiter=';')
```

In [3]:
```python
# Display basic information and the first few rows
df_info = df.info()
df_head = df.head()

df_info, df_head
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23486 entries, 0 to 23485
Data columns (total 11 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   Unnamed: 0               23486 non-null  int64 
 1   Clothing.ID              23486 non-null  int64 
 2   Age                      23486 non-null  int64 
 3   Title                    19676 non-null  object
 4   Review.Text              22641 non-null  object
 5   Rating                   23486 non-null  int64 
 6   Recommended.IND          23486 non-null  int64 
 7   Positive.Feedback.Count  23486 non-null  int64 
 8   Division.Name            23472 non-null  object
 9   Department.Name          23472 non-null  object
 10  Class.Name               23472 non-null  object
dtypes: int64(6), object(5)
memory usage: 2.0+ MB
```

Out[3]:
```
(None,
    Unnamed: 0  Clothing.ID  Age                    Title  \
 0           1          767   33                      NaN
 1           2         1080   34                      NaN
 2           3         1077   60  Some major design flaws
 3           4         1049   50         My favorite buy!
 4           5          847   47         Flattering shirt

                                          Review.Text  Rating  Recommended.IND  \
 0  Absolutely wonderful - silky and sexy and comf...       4                1
 1  Love this dress!  it's sooo pretty.  i happene...       5                1
 2  I had such high hopes for this dress and reall...       3                0
 3  I love, love, love this jumpsuit. it's fun, fl...       5                1
 4  This shirt is very flattering to all due to th...       5                1

    Positive.Feedback.Count   Division.Name Department.Name Class.Name
 0                        0       Initmates        Intimate  Intimates
 1                        4         General         Dresses    Dresses
 2                        0         General         Dresses    Dresses
 3                        0  General Petite         Bottoms      Pants
 4                        6         General            Tops    Blouses  )
```

✅ Step 1- Clean and Preprocesse the Dataset

*Removed missing values. *Created sentiment labels from ratings. *Cleaned text by removing punctuation, numbers, and converting to lowercase

In [4]:
```python
# Keep only the necessary columns for sentiment analysis
df_clean = df[['Review.Text', 'Rating']].dropna()
```

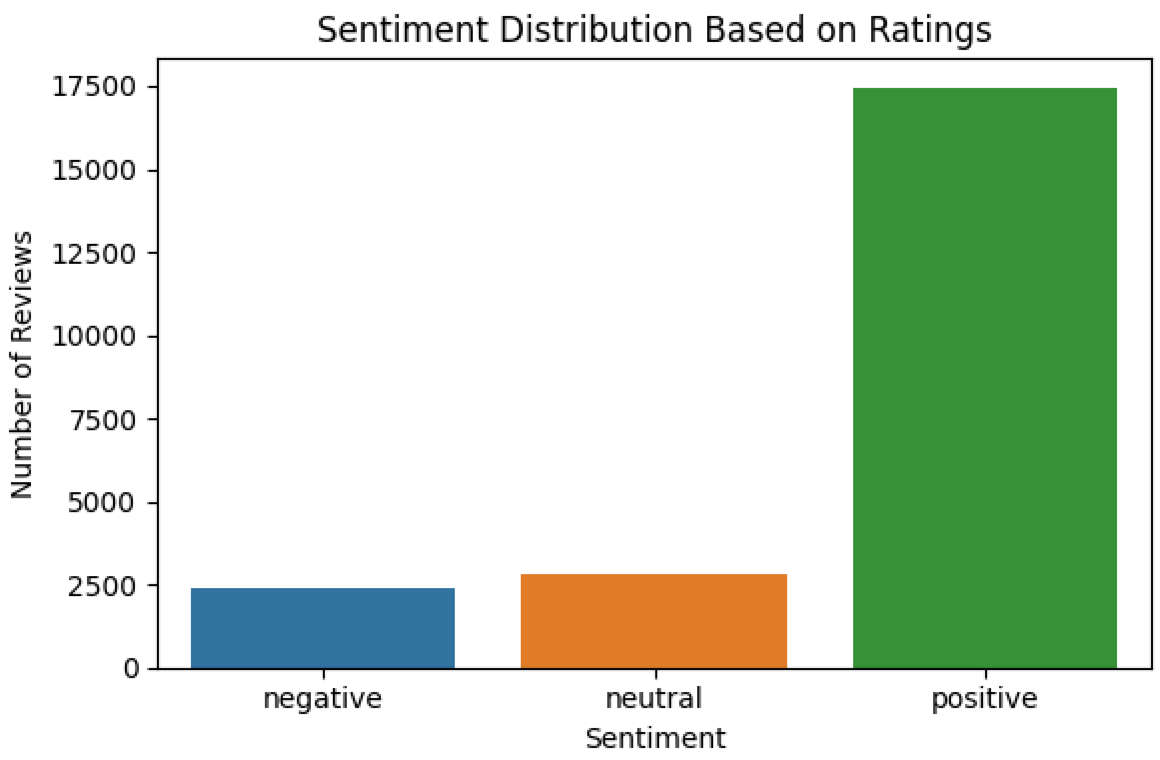
In [5]:
```python
# Define sentiment from rating
def sentiment_from_rating(rating):
    if rating <= 2:
        return 'negative'
    elif rating == 3:
        return 'neutral'
    else:
        return 'positive'
```

In [6]:
```
df_clean['Sentiment'] = df_clean['Rating'].apply(sentiment_from_rating)
```

✅ Step 2 - Exploratory Data Analysis (EDA)

In [7]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [8]:
```
# Plot sentiment distribution
plt.figure(figsize=(6, 4))
sns.countplot(data=df_clean, x='Sentiment', order=['negative', 'neutral', 'positive'])
plt.title('Sentiment Distribution Based on Ratings')
plt.xlabel('Sentiment')
plt.ylabel('Number of Reviews')
plt.tight_layout()
plt.show()
```

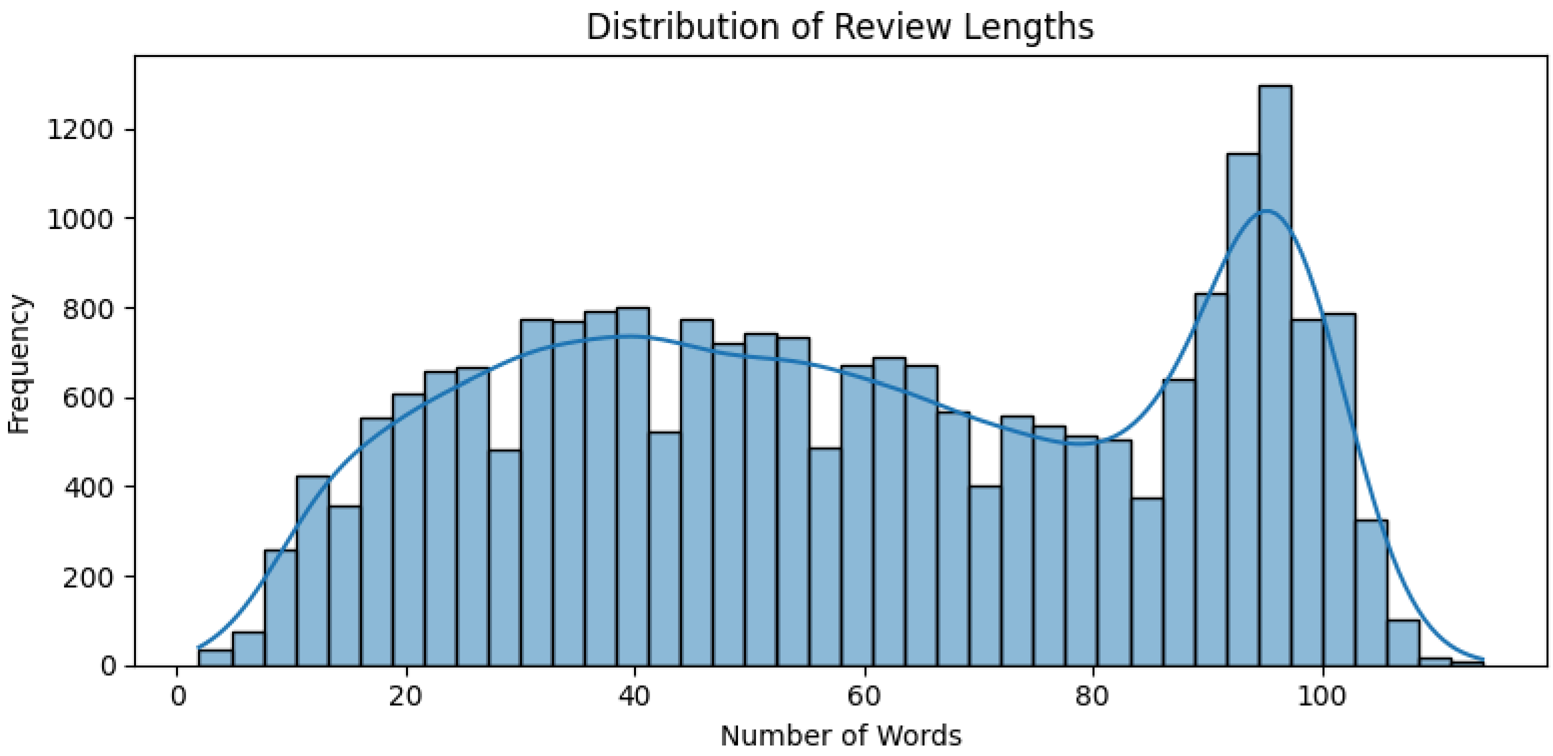


In [9]:
```
import re
```

In [10]:
```
# Clean review text for analysis
def clean_text(text):
    text = str(text).lower()
    text = re.sub(r"[^a-zA-Z\s]", "", text)
    text = re.sub(r"\s+", " ", text).strip()
    return text

df_clean['Clean_Review'] = df_clean['Review.Text'].apply(clean_text)
```

In [11]:
```
# Plot distribution of review lengths
df_clean['Review_Length'] = df_clean['Clean_Review'].apply(lambda x: len(x.split()))

plt.figure(figsize=(8, 4))
sns.histplot(data=df_clean, x='Review_Length', bins=40, kde=True)
```

```
plt.title('Distribution of Review Lengths')
plt.xlabel('Number of Words')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```



✅ Step 3 - Sentiment Encoding & Splitting

Sentiments are encoded as:

0 = Negative

1 = Neutral

2 = Positive

Data split into:

Training set: 18,112 reviews

Test set: 4,529 reviews

In [12]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
```

In [13]:
```
# Step 3: Encode sentiment labels
label_encoder = LabelEncoder()
df_clean['Sentiment_Label'] = label_encoder.fit_transform(df_clean['Sentiment'])
```

In [14]:
```
# Split data into train and test sets
X = df_clean['Clean_Review']
y = df_clean['Sentiment_Label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

In [15]:
```
# Show label encoding and shapes
label_mapping = dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_)))
X_train.shape, X_test.shape, y_train.shape, y_test.shape, label_mapping
```

Out[15]:
```
((18112,),
 (4529,),
 (18112,),
 (4529,),
 {'negative': 0, 'neutral': 1, 'positive': 2})
```

✅ # 4: Train multiple NLP models (Logistic Regression, Naive Bayes, XGBoost).

Logistic Regression

In [16]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
```

In [17]:
```
# Vectorise text using TF-IDF
tfidf = TfidfVectorizer(max_features=10000)
X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)
```

In [18]:
```
# Train Logistic Regression
logreg = LogisticRegression(max_iter=1000)
logreg.fit(X_train_tfidf, y_train)
y_pred_logreg = logreg.predict(X_test_tfidf)
```

In [19]:
```
# Classification report
logreg_report = classification_report(y_test, y_pred_logreg, target_names=label_mapping.keys(), output_dict=True)
logreg_report
```

Out[19]:
```
{'negative': {'precision': 0.6056338028169014,
  'recall': 0.45358649789029537,
  'f1-score': 0.5186972255729795,
  'support': 474},
 'neutral': {'precision': 0.4477124183006536,
  'recall': 0.2424778761061947,
  'f1-score': 0.31458094144661314,
  'support': 565},
 'positive': {'precision': 0.8733195449844882,
  'recall': 0.9679083094555874,
  'f1-score': 0.9181842892090242,
  'support': 3490},
 'accuracy': 0.8235813645396335,
 'macro avg': {'precision': 0.6422219220340144,
  'recall': 0.5546575611506924,
  'f1-score': 0.5838208187428723,
  'support': 4529},
 'weighted avg': {'precision': 0.7922086886445008,
  'recall': 0.8235813645396335,
  'f1-score': 0.8010739426315794,
  'support': 4529}}
```

Naive Bayes model

In [20]:
```
from sklearn.naive_bayes import MultinomialNB
```

In [21]:
```
# Train Naive Bayes
nb = MultinomialNB()
nb.fit(X_train_tfidf, y_train)
y_pred_nb = nb.predict(X_test_tfidf)
```

In [22]:
```
# Classification report
nb_report = classification_report(y_test, y_pred_nb, target_names=label_mapping.keys(), output_dict=True)
nb_report
```

Out[22]:
```
{'negative': {'precision': 1.0,
  'recall': 0.008438818565400843,
  'f1-score': 0.016736401673640167364017,
  'support': 474},
 'neutral': {'precision': 0.0, 'recall': 0.0, 'f1-score': 0.0, 'support': 565},
 'positive': {'precision': 0.7714412024756853,
  'recall': 1.0,
  'f1-score': 0.8709757923633641,
  'support': 3490},
 'accuracy': 0.7714727312872599,
 'macro avg': {'precision': 0.5904804008252285,
  'recall': 0.3361462728551336,
  'f1-score': 0.29590406467900143,
  'support': 4529},
 'weighted avg': {'precision': 0.6991233819033211,
  'recall': 0.7714727312872599,
  'f1-score': 0.6729164428662942,
  'support': 4529}}
```

XGBoost Model

In [23]:
```
import xgboost as xgb
```

In [24]:
```
# Train XGBoost Classifier
xgb_clf = xgb.XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')
xgb_clf.fit(X_train_tfidf, y_train)
y_pred_xgb = xgb_clf.predict(X_test_tfidf)
```

In [25]:
```
# Classification report
xgb_report = classification_report(y_test, y_pred_xgb, target_names=label_mapping.keys(), output_dict=True)
xgb_report
```

Out[25]:
```
{'negative': {'precision': 0.5901060070671378,
  'recall': 0.35232067510548526,
  'f1-score': 0.441215323645971,
  'support': 474},
 'neutral': {'precision': 0.41544117647058826,
  'recall': 0.2,
  'f1-score': 0.2700119474313023,
  'support': 565},
 'positive': {'precision': 0.8558127830900856,
  'recall': 0.9744985673352435,
  'f1-score': 0.9113076098606645,
  'support': 3490},
 'accuracy': 0.8127621991609627,
 'macro avg': {'precision': 0.6204533222092706,
  'recall': 0.508939747480243,
  'f1-score': 0.540844960312646,
  'support': 4529},
 'weighted avg': {'precision': 0.7730671505939731,
  'recall': 0.8127621991609627,
  'f1-score': 0.7821056242262299,
  'support': 4529}}
```

✅ Visual comparison of Logistic Regression and Naive Bayes models

from sklearn.linear_model import LogisticRegression

from sklearn.naive_bayes import MultinomialNB from sklearn.metrics import classification_report

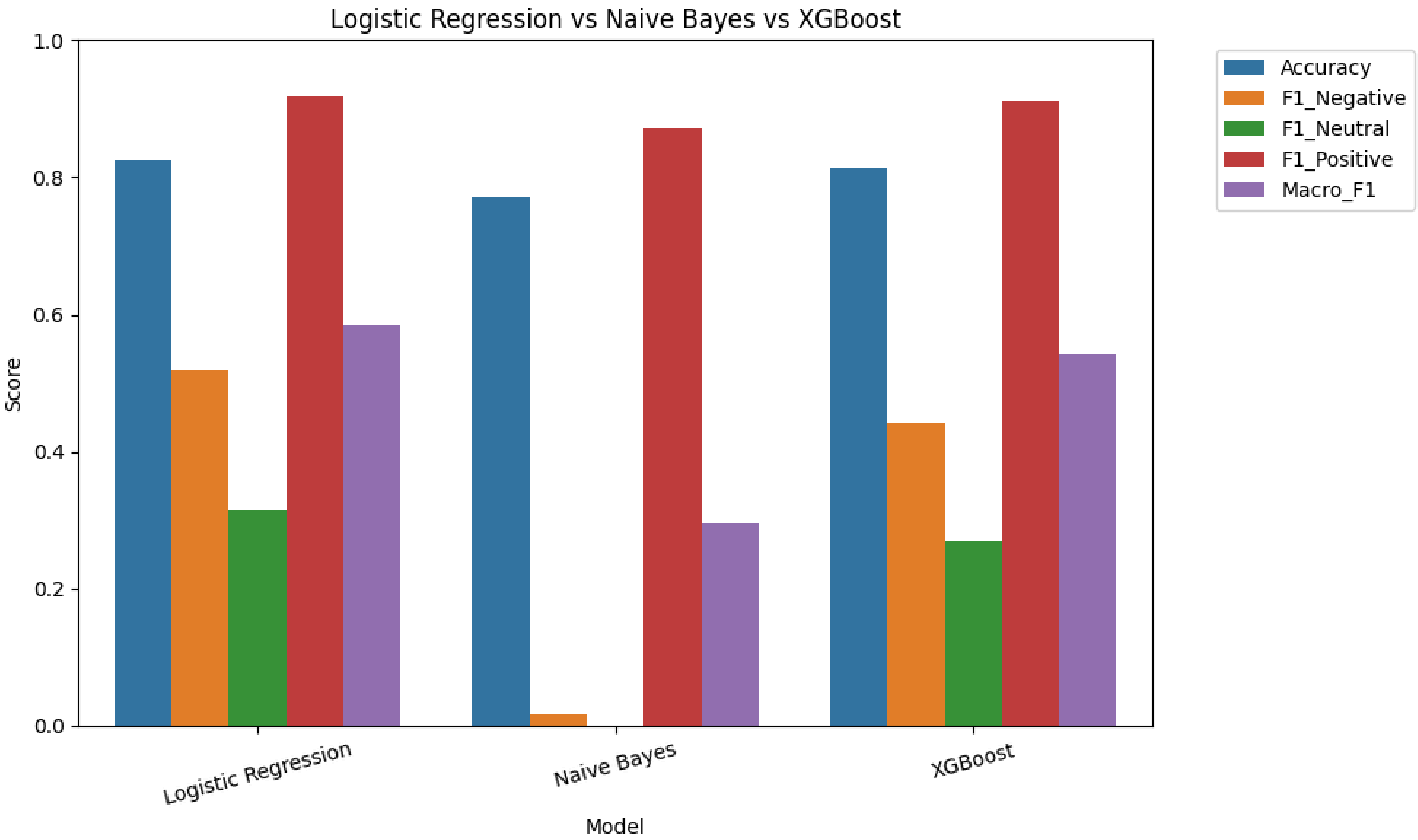
In [26]:
```
# Train and evaluate models
logreg = LogisticRegression(max_iter=1000)
logreg.fit(X_train_tfidf, y_train)
y_pred_logreg = logreg.predict(X_test_tfidf)
logreg_report = classification_report(y_test, y_pred_logreg, output_dict=True, zero_division=0)

nb = MultinomialNB()
nb.fit(X_train_tfidf, y_train)
y_pred_nb = nb.predict(X_test_tfidf)
nb_report = classification_report(y_test, y_pred_nb, output_dict=True, zero_division=0)
```

In [27]:
```
xgb_report = classification_report(y_test, y_pred_xgb, target_names=["negative", "neutral", "positive"], output_dict=True)
logreg_report = classification_report(y_test, y_pred_logreg, target_names=["negative", "neutral", "positive"], output_dict=True)
nb_report = classification_report(y_test, y_pred_nb, target_names=["negative", "neutral", "positive"], output_dict=True)
```

In [28]:
```
# Create DataFrame
results = pd.DataFrame([
    {
        "Model": "Logistic Regression",
        "Accuracy": logreg_report["accuracy"],
        "F1_Negative": logreg_report["negative"]["f1-score"],
        "F1_Neutral": logreg_report["neutral"]["f1-score"],
        "F1_Positive": logreg_report["positive"]["f1-score"],
        "Macro_F1": logreg_report["macro avg"]["f1-score"]
    },
    {
        "Model": "Naive Bayes",
        "Accuracy": nb_report["accuracy"],
        "F1_Negative": nb_report["negative"]["f1-score"],
        "F1_Neutral": nb_report["neutral"]["f1-score"],
        "F1_Positive": nb_report["positive"]["f1-score"],
        "Macro_F1": nb_report["macro avg"]["f1-score"]
    },
    {
        "Model": "XGBoost",
        "Accuracy": xgb_report["accuracy"],
        "F1_Negative": xgb_report["negative"]["f1-score"],
```

```
        "F1_Neutral": xgb_report["neutral"]["f1-score"],
        "F1_Positive": xgb_report["positive"]["f1-score"],
        "Macro_F1": xgb_report["macro avg"]["f1-score"]
    }
])
```

In [29]:
```
# Melt and plot
results_melted = results.melt(id_vars='Model', var_name='Metric', value_name='Score')
plt.figure(figsize=(10, 6))
sns.barplot(data=results_melted, x='Model', y='Score', hue='Metric')
plt.title('Logistic Regression vs Naive Bayes vs XGBoost')
plt.ylabel('Score')
plt.ylim(0, 1)
plt.xticks(rotation=15)
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```



✅ Forecasting for 5 years

In [30]:
```
from prophet import Prophet
import matplotlib.pyplot as plt
```

In [31]:
```
#  Create Sentiment column
def sentiment_from_rating(r):
    if r <= 2:
        return 'negative'
    elif r == 3:
        return 'neutral'
    else:
        return 'positive'
df['Sentiment'] = df['Rating'].apply(sentiment_from_rating)
```

In [32]:
```
# Simulate a date column for each review (e.g., starting from Jan 2015)
start_date = pd.to_datetime("2015-01-01")
df['Date'] = pd.date_range(start=start_date, periods=len(df), freq='D')
```

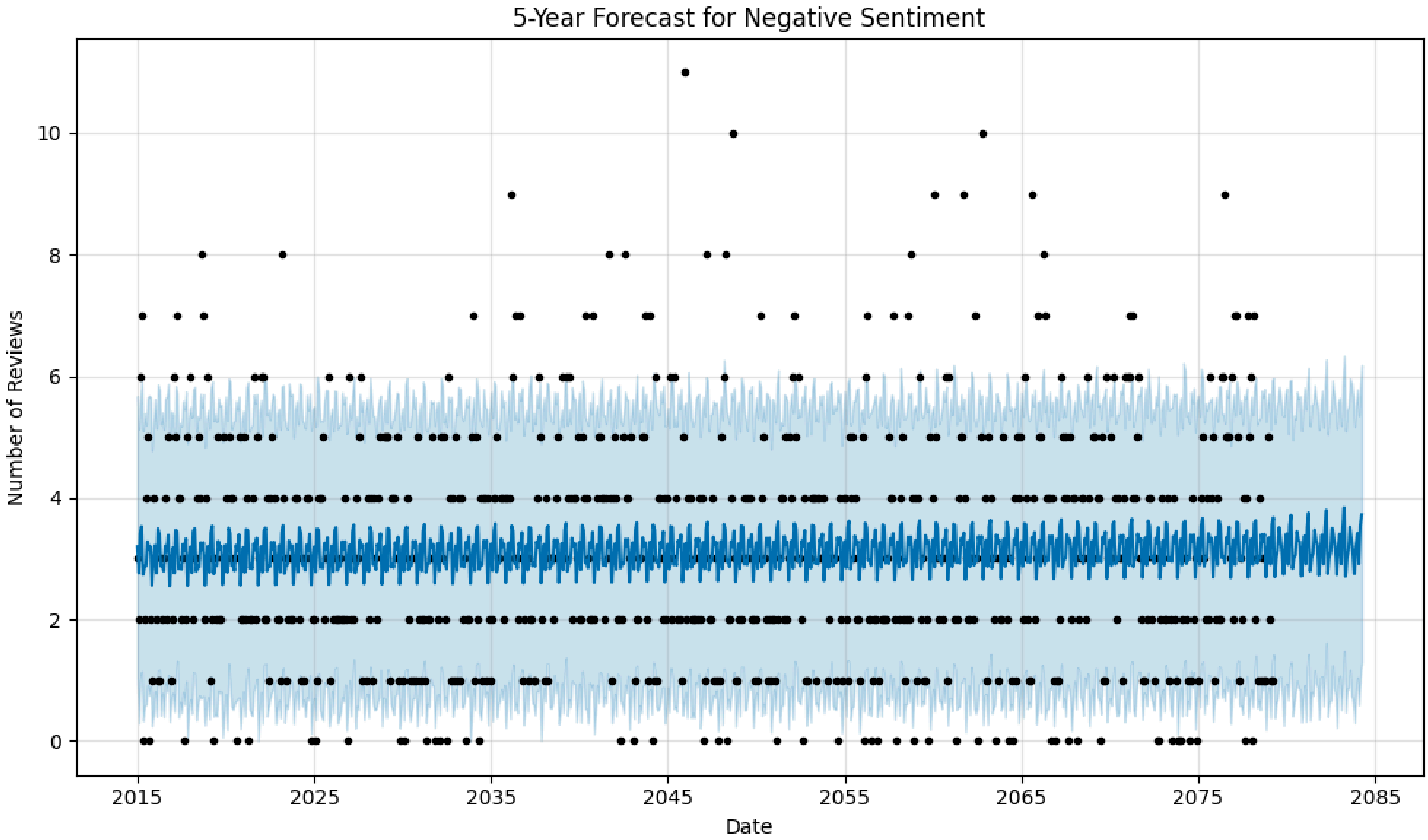
In [33]:
```
# Group by month and sentiment
df['Month'] = df['Date'].dt.to_period('M').dt.to_timestamp()
monthly_sentiment = df.groupby(['Month', 'Sentiment']).size().unstack().fillna(0)
```

In [34]:
```
# Forecasting for each sentiment
forecast_years = 5
forecast_horizon = forecast_years * 12  # 5 years in months

for sentiment in ['positive', 'neutral', 'negative']:
    sentiment_df = monthly_sentiment[[sentiment]].reset_index()
    sentiment_df.columns = ['ds', 'y']  # Prophet requires 'ds' and 'y' columns

    model = Prophet()
    model.fit(sentiment_df)

    future = model.make_future_dataframe(periods=forecast_horizon, freq='M')
    forecast = model.predict(future)
```

```
19:54:58 - cmdstanpy - INFO - Chain [1] start processing
19:54:58 - cmdstanpy - INFO - Chain [1] done processing
19:54:58 - cmdstanpy - INFO - Chain [1] start processing
19:54:58 - cmdstanpy - INFO - Chain [1] done processing
19:54:59 - cmdstanpy - INFO - Chain [1] start processing
19:54:59 - cmdstanpy - INFO - Chain [1] done processing
```

In [36]:

```
# Plot forecast
fig = model.plot(forecast)
plt.title(f'5-Year Forecast for {sentiment.capitalize()} Sentiment')
plt.xlabel("Date")
plt.ylabel("Number of Reviews")
plt.tight_layout()
plt.show()
```


5-Year Forecast for Negative Sentiment
Number of Reviews
0
2
4
6
8
10
2015
2025
2035
2045
2055
2065
2075
2085
Date

In [ ]: