Step 00: Import necessary libraries

```
import numpy as np
import pandas as pd
```

Step 01: Load dataset

```
df = pd.read_csv("/content/Reviews.csv")
df
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not as Advertised
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	"Delight" says it all
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	1307923200	Cough Medicine
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	1350777600	Great taffy
...	...	...	...	...	...	...	...	...	...
47914	47915	B004SRH2B6	A191ACUFKGFO53	Cynthia J. Newsom	0	0	5	1344211200	Tastes Great
47915	47916	B004SRH2B6	ABGMKOWUNRCM5	Jennifer Joann Ellis "Jen"	0	0	5	1343347200	Like a light Yoo-Hoo

```
from google.colab import drive
drive.mount('/content/drive')
```

Step 02: Check last 20 rows

```
df.tail(20)
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary
47899	47900	B004SRH2B6	A3EJZZ8HPFCTSK	cmt018	0	0	5	1347926400	Wonderful
47900	47901	B004SRH2B6	A3J5J16BPO6Q9B	kel	0	0	5	1347840000	So far the best
47901	47902	B004SRH2B6	A186NOZW7YYOX2	Wurk Indad	0	0	2	1347840000	Okay...
47902	47903	B004SRH2B6	A1CT5D2CE2JOH	Rob M	0	0	5	1347840000	ZICO - Great stuff - a bit expensive!
47903	47904	B004SRH2B6	A3MPTMYAYVVM8M	E. Gladden	0	0	5	1347667200	Coconut water " Yoo-Hoo"
47904	47905	B004SRH2B6	A16FINS0ZM8EJF	R. E. Wolff	0	0	3	1347580800	Good but not great
47905	47906	B004SRH2B6	A17ALZ8BNDN17D	LG	0	0	5	1347062400	Love this stuff!!
47906	47907	B004SRH2B6	A2H0XZNQKU7MQ	Lynne "LynneMH"	0	0	5	1346976000	It's great!
47907	47908	B004SRH2B6	AUJ9KXX20JX4Q	Jeremy A. Mazur	0	0	5	1346889600	great drink
47908	47909	B004SRH2B6	A1XFDTSZA5Z4X3	retired	0	0	5	1346544000	Great Stuff
47909	47910	B004SRH2B6	A75AIUX8ZP2UM	Laurie	0	0	5	1346112000	Zico
47910	47911	B004SRH2B6	A285ML7MDHXA7P	That Guy	0	0	4	1345593600	Not the best, but a good price
47911	47912	B004SRH2B6	AI75XZ6BENG2E	kathi	0	0	5	1345075200	the best coconut water!
									Zico

Step 03: Check data types

```
df.dtypes
```

	0
Id	int64
ProductId	object
UserId	object
ProfileName	object
HelpfulnessNumerator	int64
HelpfulnessDenominator	int64
Score	int64
Time	int64
Summary	object
Text	object

dtype: object

Step 04: Basic statistical description

```
df.describe()
```

	Id	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
count	47919.000000	47919.000000	47919.000000	47919.000000	4.791900e+04
mean	23960.000000	1.600806	2.054175	4.149627	1.295042e+09
std	13833.168111	5.636381	6.224214	1.323196	4.728541e+07
min	1.000000	0.000000	0.000000	1.000000	9.617184e+08
25%	11980.500000	0.000000	0.000000	4.000000	1.269302e+09
50%	23960.000000	0.000000	1.000000	5.000000	1.308874e+09
75%	35939.500000	2.000000	2.000000	5.000000	1.330906e+09
max	47919.000000	398.000000	401.000000	5.000000	1.351210e+09

Step 05: Full statistical description

```
df.describe(include="all")
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
count	47919.000000	47919	47919	47916	47919.000000	47919.000000	47919.000000	4.791900e+04
unique	NaN	5970	38443	35536	NaN	NaN	NaN	NaN
top	NaN	B002QWP89S	AY12DBB0U420B	Gary Peterson	NaN	NaN	NaN	NaN
freq	NaN	632	42	42	NaN	NaN	NaN	NaN
mean	23960.000000	NaN	NaN	NaN	1.600806	2.054175	4.149627	1.295042e+09
std	13833.168111	NaN	NaN	NaN	5.636381	6.224214	1.323196	4.728541e+07
min	1.000000	NaN	NaN	NaN	0.000000	0.000000	1.000000	9.617184e+08
25%	11980.500000	NaN	NaN	NaN	0.000000	0.000000	4.000000	1.269302e+09
50%	23960.000000	NaN	NaN	NaN	0.000000	1.000000	5.000000	1.308874e+09

Step 06: Dataset info

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 47919 entries, 0 to 47918
Data columns (total 10 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Id                      47919 non-null  int64
 1   ProductId               47919 non-null  object
 2   UserId                  47919 non-null  object
 3   ProfileName             47916 non-null  object
 4   HelpfulnessNumerator    47919 non-null  int64
 5   HelpfulnessDenominator  47919 non-null  int64
 6   Score                   47919 non-null  int64
 7   Time                    47919 non-null  int64
 8   Summary                 47917 non-null  object
 9   Text                    47918 non-null  object
dtypes: int64(5), object(5)
memory usage: 3.7+ MB
```

Step 07: Check for null values

```
df.isnull().sum()
```

	0
Id	0
ProductId	0
UserId	0
ProfileName	3
HelpfulnessNumerator	0
HelpfulnessDenominator	0
Score	0
Time	0
Summary	2
Text	1

dtype: int64

Double-click (or enter) to edit

Step 08: Detailed missing data analysis

```
missing_data = df.isnull()
for column in missing_data.columns.values.tolist():
    print(column)
    print (missing_data[column].value_counts())
    print("")
```

```
Id
Id
False    47919
Name: count, dtype: int64

ProductId
ProductId
False    47919
Name: count, dtype: int64

UserId
UserId
False    47919
Name: count, dtype: int64

ProfileName
ProfileName
False    47916
True         3
Name: count, dtype: int64

HelpfulnessNumerator
HelpfulnessNumerator
False    47919
Name: count, dtype: int64

HelpfulnessDenominator
HelpfulnessDenominator
False    47919
Name: count, dtype: int64

Score
Score
False    47919
Name: count, dtype: int64

Time
Time
False    47919
Name: count, dtype: int64

Summary
Summary
False    47917
True         2
Name: count, dtype: int64
```

```
Text
Text
False    47918
True         1
Name: count, dtype: int64
```

Step 09: Import visualization libraries

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

Step 10: Sample the dataset

```
# Step 1: Load the dataset
# df = pd.read_csv('/content/Reviews.csv')
df = df.sample(n=10000, random_state=42)
```

Step 11: Calculate mean, median, mode for numerical columns

```
# Step 2: Calculate Mean, Median, and Mode
# Numerical columns
numerical_cols = ['Score', 'HelpfulnessNumerator', 'HelpfulnessDenominator', 'Time']
print("\nMean, Median, Mode for Numerical Columns:")
for col in numerical_cols:
    mean_val = df[col].mean()
    median_val = df[col].median()
    mode_val = df[col].mode()[0]
    print(f"{col}: Mean = {mean_val:.2f}, Median = {median_val:.2f}, Mode = {mode_val}")
```

```
Mean, Median, Mode for Numerical Columns:
Score: Mean = 4.16, Median = 5.00, Mode = 5
HelpfulnessNumerator: Mean = 1.61, Median = 0.00, Mode = 0
HelpfulnessDenominator: Mean = 2.06, Median = 1.00, Mode = 0
Time: Mean = 1294732615.68, Median = 1308873600.00, Mode = 1350345600
```

Step 12: Mode for text columns

```
# Categorical/text columns
text_cols = ['ProfileName', 'Summary']
print("\nMode for Text Columns:")
for col in text_cols:
    mode_val = df[col].mode()[0]
    print(f"{col}: Mode = {mode_val}")
```

```
Mode for Text Columns:
ProfileName: Mode = Gary Peterson
Summary: Mode = Delicious
```

Step 13: Handle missing values - numerical columns

```
# Step 3: Handle Missing Values
# 3.1: Binning for numerical columns (if there were missing values)
# Note: No missing values in numerical columns, but we'll demonstrate the technique
for col in numerical_cols:
    if df[col].isnull().sum() > 0:
        # Create bins based on quantiles
        df[f'{col}_bin'] = pd.qcut(df[col], q=4, duplicates='drop', labels=False)
        # Replace missing values with the mean of the respective bin
        df[col] = df.groupby(f'{col}_bin')[col].transform(lambda x: x.fillna(x.mean()))
        df.drop(columns=[f'{col}_bin'], inplace=True)
```

Step 14: Impute missing ProfileName with mode

```
# 3.2: Impute missing values in ProfileName and Summary
# ProfileName: Replace with mode
profile_mode = df['ProfileName'].mode()[0]
df['ProfileName'] = df['ProfileName'].fillna(profile_mode)
```

Step 15: Print ProfileName mode

```
# Calculate and print the mode of ProfileName
profile_mode = df['ProfileName'].mode()[0]
print(f"Mode of ProfileName: {profile_mode}")
```

```
Mode of ProfileName: Gary Peterson
```

Step 16: ProfileName frequency count

```
# Count the frequency of each unique value in the ProfileName column
profile_counts = df['ProfileName'].value_counts()

# Display the results
print("Frequency of each ProfileName:")
print(profile_counts)
```

```
Frequency of each ProfileName:
ProfileName
Gary Peterson                              12
Rebecca of Amazon "The Rebecca Review"      9
Jessica                                     8
Jennifer                                    8
Lynrie "Oh HELL no"                         8
                                           ..
Joan Hough "Jo Cook"                        1
Johnny "jb-in-nc"                           1
truth seeker "truth seeker"                 1
BeanCounter                                 1
Pat Shand "Pat Shand"                       1
Name: count, Length: 8953, dtype: int64
```

Step 17: Impute missing Summary with mode

```
# Summary: Replace with mode (or empty string if preferred)
summary_mode = df['Summary'].mode()[0]
df['Summary'] = df['Summary'].fillna(summary_mode)
```

Step 18: Print Summary mode

```
summary_mode = df['Summary'].mode()[0]
print(f"Mode of Summary: {summary_mode}")
```

```
Mode of Summary: Delicious
```

Step 19: Summary frequency count

```
# Count the frequency of each unique value in the ProfileName column
Summary_counts = df['Summary'].value_counts()

# Display the results
print("Frequency of each Summary:")
print(Summary_counts)
```

```
Frequency of each Summary:
Summary
Delicious              42
Delicious!             40
Yummy!                 29
Great product          22
Great Product          20
                       ..
great deal              1
Just Pretty Good        1
Easy and Yummy Snack    1
great catnip            1
Tasty Chips             1
Name: count, Length: 8959, dtype: int64
```

No Missing or NUll Values Now

Step 20: Verify no missing values

```
# Verify missing values after imputation
print("\nMissing Values After Imputation:")
print(df.isnull().sum())
```

```

Missing Values After Imputation:
Id                        0
ProductId                 0
UserId                    0
ProfileName               0
HelpfulnessNumerator      0
HelpfulnessDenominator    0
Score                     0
Time                      0
Summary                   0
Text                      0
dtype: int64
```

Step 21: Install required libraries

```
# Install required libraries
!pip install pandas numpy matplotlib seaborn

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

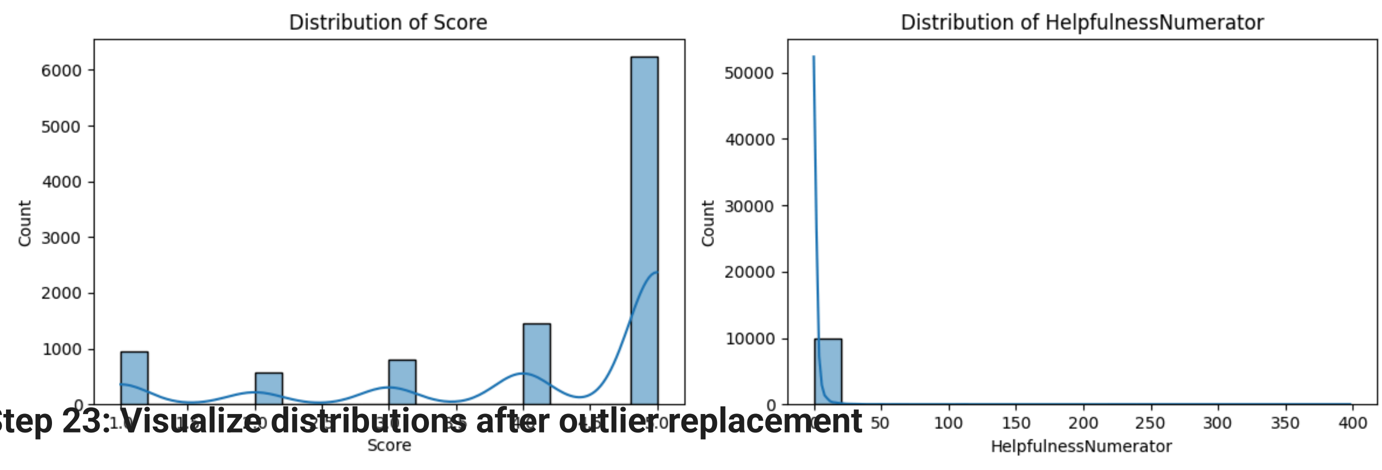
```
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (2.0.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.11/dist-packages (0.13.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
```

Step 22: Load and analyze dataset

```
import csv

# Step 1: Load and sample the dataset
```

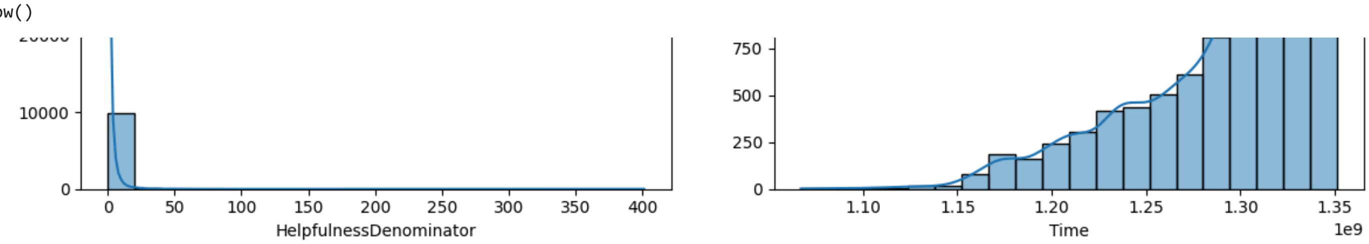
```
# Load dataset from /content/ with robust parsing
dataset_path = '/content/Reviews.csv'
df = pd.read_csv(
    dataset_path,
    quoting=csv.QUOTE_ALL,
    engine='python',
    on_bad_lines='skip',
    encoding='utf-8'
)
print("Dataset loaded successfully. Shape:", df.shape)
print(df.head())
df = df.sample(n=10000, random_state=42)

# Step 2: Define numerical columns
numerical_cols = ['Score', 'HelpfulnessNumerator', 'HelpfulnessDenominator', 'Time']

# Step 3: Summary statistics to understand distribution
print("Summary Statistics:")
print(df[numerical_cols].describe())

# Step 4: Visualize distributions with histograms
plt.figure(figsize=(12, 8))
for i, col in enumerate(numerical_cols, 1):
    plt.subplot(2, 2, i)
    sns.histplot(df[col], bins=20, kde=True)
    plt.title(f'Distribution of {col}')
plt.tight_layout()
plt.show()

# Step 5: Identify outliers with adjusted IQR (multiplier = 3)
outliers_dict = {}
for col in numerical_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 3 * IQR  # Increased multiplier to 3
    upper_bound = Q3 + 3 * IQR  # Increased multiplier to 3

    # Identify outliers
    outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)][col]
    outliers_dict[col] = outliers.tolist()

    # Print column name and outlier values
    print(f"\nOutliers in {col} (IQR multiplier = 3):")
    if len(outliers) > 0:
        print(f"Number of outliers: {len(outliers)}")
        print(f"Outlier values: {outliers.tolist()}")
    else:
        print("No outliers found.")

# Step 6: Visualize outliers with boxplots
plt.figure(figsize=(12, 8))
for i, col in enumerate(numerical_cols, 1):
    plt.subplot(2, 2, i)
    sns.boxplot(x=df[col])
    plt.title(f'Boxplot of {col}')
plt.tight_layout()
plt.show()
```

```
Dataset loaded successfully. Shape: (58151, 10)
   Id   ProductId          UserId                      ProfileName  \
0   1  B001E4KFG0  A3SGXH7AUHU8GW                       delmartian
1   2  B00813GRG4  A1D87F6ZCVE5NK                           dll pa
2   3  B000LQOCH0   ABXLMWJIXXAIN  Natalia Corres "Natalia Corres"
3   4  B000UA0QIQ  A395BORC6FGVXV                             Karl
4   5  B006K2ZZ7K  A1UQRSCLF8GW1T    Michael D. Bigham "M. Wassir"

   HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
0                     1                       1      5  1303862400
1                     0                       0      1  1346976000
2                     1                       1      4  1219017600
3                     3                       3      2  1307923200
4                     0                       0      5  1350777600

                 Summary                                               Text
0  Good Quality Dog Food  I have bought several of the Vitality canned d...
1      Not as Advertised  Product arrived labeled as Jumbo Salted Peanut...
2  "Delight" says it all  This is a confection that has been around a fe...
3         Cough Medicine  If you are looking for the secret ingredient i...
4            Great taffy  Great taffy at a great price.  There was a wid...
Summary Statistics:
              Score  HelpfulnessNumerator  HelpfulnessDenominator  \
count  10000.000000          10000.000000            10000.000000
mean       4.148100              1.658600                2.133200
std        1.324742              6.482349                7.039049
min        1.000000              0.000000                0.000000
25%        4.000000              0.000000                0.000000
50%        5.000000              0.000000                1.000000
75%        5.000000              2.000000                2.000000
max        5.000000            398.000000              401.000000

               Time
count  1.000000e+04
mean   1.294707e+09
std    4.794754e+07
min    1.067040e+09
25%    1.268438e+09
50%    1.308787e+09
75%    1.331510e+09
max    1.351210e+09
```

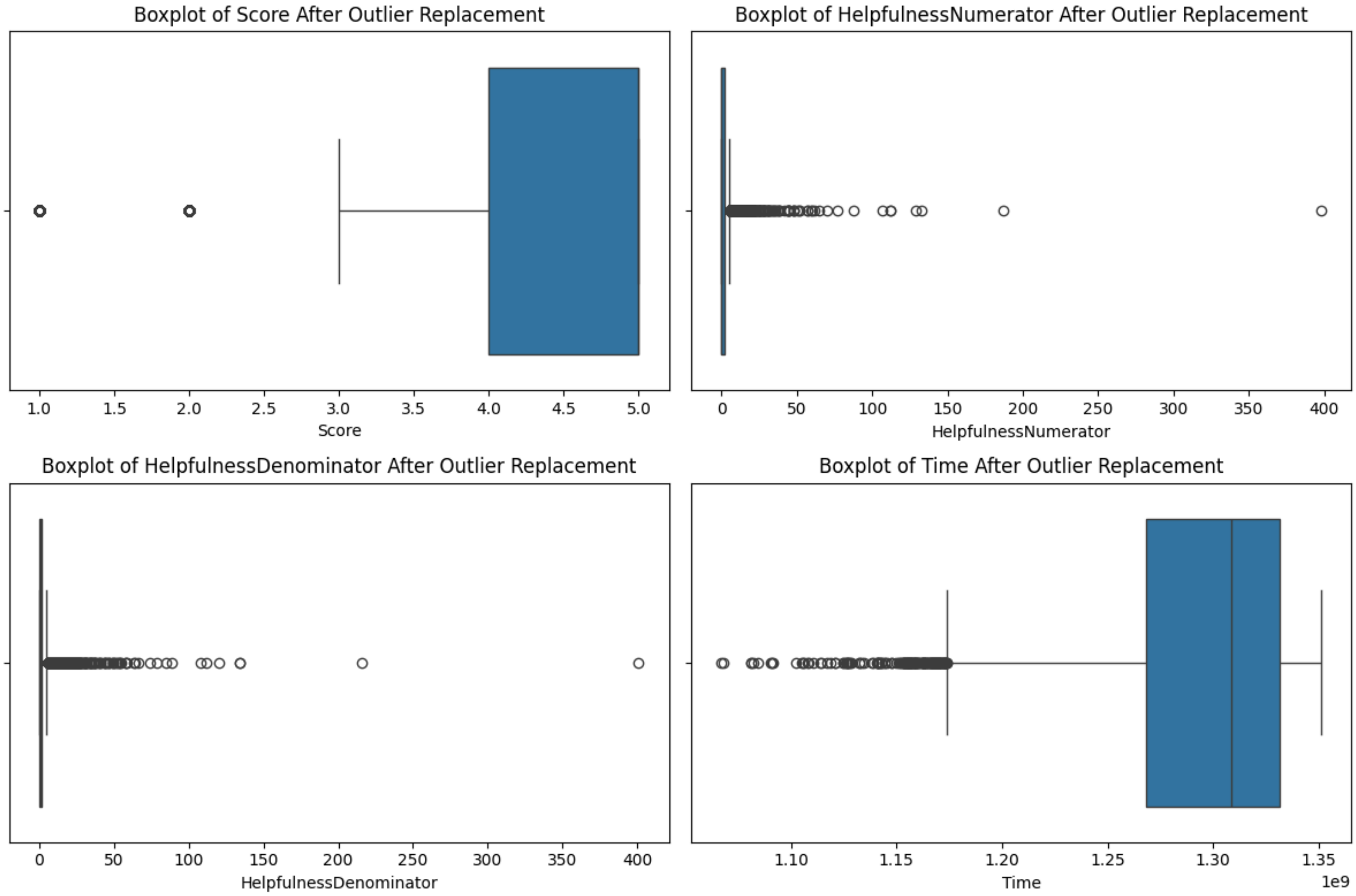


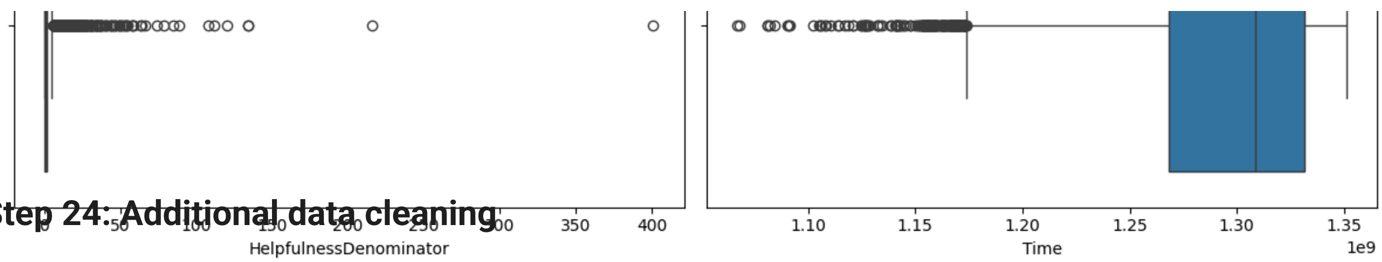
Step 23: Visualize distributions after outlier replacement

```
# 4.3: Visualize after outlier replacement
plt.figure(figsize=(12, 8))
for i, col in enumerate(numerical_cols, 1):
    plt.subplot(2, 2, i)
    sns.boxplot(x=df[col])
    plt.title(f'Boxplot of {col} After Outlier Replacement')
plt.tight_layout()
plt.show()
```




```
Outliers in Score (IQR multiplier = 3):
No outliers found.
```





Step 24: Additional data cleaning

```
# Step 5: Other Necessary Data Cleaning
# 5.1: Remove duplicates
df = df.drop_duplicates()
print(f"\nNumber of rows after removing duplicates: {len(df)}")

# 5.2: Standardize text columns
df['ProfileName'] = df['ProfileName'].str.lower().str.strip()
df['Summary'] = df['Summary'].str.lower().str.strip()
df['Text'] = df['Text'].str.lower().str.strip()

# 5.3: Ensure correct data types
df['Id'] = df['Id'].astype(int)
df['HelpfulnessNumerator'] = df['HelpfulnessNumerator'].astype(int)
df['HelpfulnessDenominator'] = df['HelpfulnessDenominator'].astype(int)
df['Score'] = df['Score'].astype(int)
df['Time'] = df['Time'].astype(int)
```

```
Number of rows after removing duplicates: 10000
```

Step 25: Show cleaned dataset

```
# Step 6: Show the Cleaned Dataset
print("\nCleaned Dataset Info:")
print(df.info())
```

```
print("\nFirst 5 rows of Cleaned Dataset:")
print(df.head())
```

```
Cleaned Dataset Info:
<class 'pandas.core.frame.DataFrame'>
Index: 10000 entries, 44711 to 54288
Data columns (total 10 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Id                      10000 non-null  int64
 1   ProductId               10000 non-null  object
 2   UserId                  10000 non-null  object
 3   ProfileName             9999 non-null   object
 4   HelpfulnessNumerator    10000 non-null  int64
 5   HelpfulnessDenominator  10000 non-null  int64
 6   Score                   10000 non-null  int64
 7   Time                    10000 non-null  int64
 8   Summary                 9999 non-null   object
 9   Text                    10000 non-null  object
dtypes: int64(5), object(5)
memory usage: 859.4+ KB
None

First 5 rows of Cleaned Dataset:
          Id   ProductId          UserId  \
44711  44712  B001EQ55RW   AQ8DU6XVA3USJ
29840  29841  B000F9XBJ8  A33DKGANHQJSXC
42810  42811  B002NHYQAS   A22PUBSSNP54L
32660  32661  B0083QJU72  A2XJKCHDT5NNAA
47393  47394  B000OQ2JBS  A261H8DCNDAQ3J

                                   ProfileName  HelpfulnessNumerator  \
44711  alejandra vernon "artist & illustrator"                     2
29840                        anastasia "stasia"                     0
42810                  g. little "value seeker"                     0
32660                           monica kim "mk"                     0
47393         d. m. pheneger "hungry for truth"                     1

       HelpfulnessDenominator  Score        Time  \
44711                       4      5  1211241600
29840                       0      5  1245715200
42810                       0      5  1298419200
32660                       0      5  1305763200
47393                       1      5  1209945600

                                          Summary  \
44711  the subtle and sophisticated chocolate almond
29840               very different than the original
42810                             smooth and rich ...
32660                             great tasting syrup
47393                        best all around condiment

                                                 Text
44711  this most unusual dry roasted almond from emer...
29840  but very delicious just the same. if you're lo...
42810  smooth and rich and so yummy!  what more can y...
32660  wow, reviewers weren't kidding.  this was the ...
47393  as a chef in the food service field i have use...
```

Step 26: Verify no missing values

```
# Verify missing values after imputation
print("\nMissing Values After Imputation:")
print(df.isnull().sum())
```

```
Missing Values After Imputation:
Id                        0
ProductId                 0
UserId                    0
ProfileName               1
HelpfulnessNumerator      0
HelpfulnessDenominator    0
Score                     0
Time                      0
Summary                   1
Text                      0
dtype: int64
```

Step 27: Handle empty strings in Summary

```
# Step 4: Check for empty strings or whitespace in Summary and convert to NaN
df['Summary'] = df['Summary'].replace(r'^\s*$', np.nan, regex=True)  # Replace empty or whitespace-only strings with NaN
print("\nMissing Values After Converting Empty Strings to NaN:")
print(df.isnull().sum())

# Step 5: Impute Summary with mode
summary_mode = df['Summary'].mode()[0]
df['Summary'] = df['Summary'].fillna(summary_mode)

# Step 6: Verify missing values after imputation
print("\nMissing Values After Imputation:")
print(df.isnull().sum())

# Step 7: Additional check for empty strings (just to be sure)
empty_summary_count = len(df[df['Summary'].str.strip() == ''])
print(f"\nNumber of empty string values in Summary after imputation: {empty_summary_count}")
```

```
Missing Values After Converting Empty Strings to NaN:
Id                        0
ProductId                 0
UserId                    0
ProfileName               1
HelpfulnessNumerator      0
HelpfulnessDenominator    0
Score                     0
Time                      0
Summary                   1
Text                      0
dtype: int64

Missing Values After Imputation:
Id                        0
ProductId                 0
UserId                    0
ProfileName               1
HelpfulnessNumerator      0
HelpfulnessDenominator    0
Score                     0
Time                      0
Summary                   0
Text                      0
dtype: int64

Number of empty string values in Summary after imputation: 0
```

Step 28: Save cleaned dataset

```
# Optional: Save the cleaned dataset
df.to_csv('/content/Reviews_cleaned.csv', index=False)
print("\nCleaned dataset saved as 'Reviews_cleaned.csv'")
```

```
Cleaned dataset saved as 'Reviews_cleaned.csv'
```

Step 29: Final null check

```
df.isnull().sum()
```

	0
Id	0
ProductId	0
UserId	0
ProfileName	1
HelpfulnessNumerator	0
HelpfulnessDenominator	0
Score	0
Time	0
Summary	0
Text	0

dtype: int64

Double-click (or enter) to edit

Step 30: Initialize NLP tools

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
import warnings
warnings.filterwarnings('ignore')
```

Step 31: Load cleaned data and prepare for analysis

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from wordcloud import WordCloud
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import nltk

# Download NLTK data
nltk.download('vader_lexicon')

# Load the cleaned dataset
df = pd.read_csv('/content/Reviews_cleaned.csv')
print("Dataset Loaded Successfully:")
print(df.head())
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
Dataset Loaded Successfully:
      Id   ProductId          UserId                            ProfileName  \
0  44712  B001EQ55RW   AQ8DU6XVA3USJ  alejandra vernon "artist & illustrator"
1  29841  B000F9XBJ8  A33DKGANHQJSXC                     anastasia "stasia"
2  42811  B002NHYQAS   A22PUBSSNP54L                 g. little "value seeker"
3  32661  B0083QJU72  A2XJKCHDT5NNAA                         monica kim "mk"
4  47394  B000OQ2JBS  A261H8DCNDAQ3J        d. m. pheneger "hungry for truth"

   HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
0                     2                       4      5  1211241600
1                     0                       0      5  1245715200
2                     0                       0      5  1298419200
3                     0                       0      5  1305763200
4                     1                       1      5  1209945600

                                          Summary  \
```

```
0  the subtle and sophisticated chocolate almond
1              very different than the original
2                           smooth and rich ...
3                           great tasting syrup
4                      best all around condiment

                                            Text
0  this most unusual dry roasted almond from emer...
1  but very delicious just the same. if you're lo...
2  smooth and rich and so yummy!  what more can y...
3  wow, reviewers weren't kidding.  this was the ...
4  as a chef in the food service field i have use...
```

Double-click (or enter) to edit

Step 32: Text cleaning function

```
import re
from nltk.corpus import stopwords
nltk.download('stopwords')

# Function to clean text
def clean_text(text):
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    stop_words = set(stopwords.words('english'))
    text = ' '.join(word for word in text.split() if word.lower() not in stop_words)
    return text

# Apply cleaning to Text column and create Text_Cleaned
df['Text_Cleaned'] = df['Text'].apply(clean_text)
print("\nSample of Text_Cleaned:")
print(df['Text_Cleaned'].head())
```

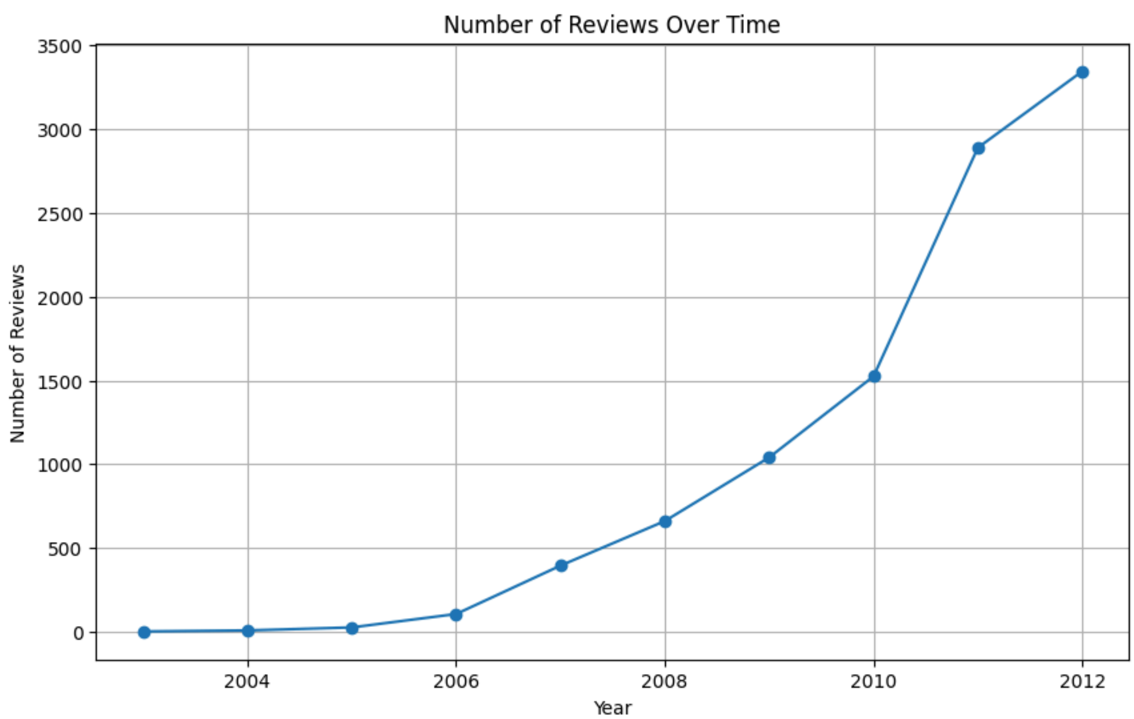
```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.

Sample of Text_Cleaned:
0    unusual dry roasted almond emerald lightly coa...
1    delicious youre looking wee bit different flav...
2    smooth rich yummy say good chocolate well also...
3    wow reviewers werent kidding best tasting mapl...
4    chef food service field used sauce many differ...
Name: Text_Cleaned, dtype: object
```

Step 33: Word cloud visualization

```
# Step 3: Word Cloud
text = ' '.join(df['Text_Cleaned'].dropna())
wordcloud = WordCloud(width=800, height=400, background_color='white', max_words=100).generate(text)
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud of Review Text')
plt.show()
```

Step 34: Time-series analysis

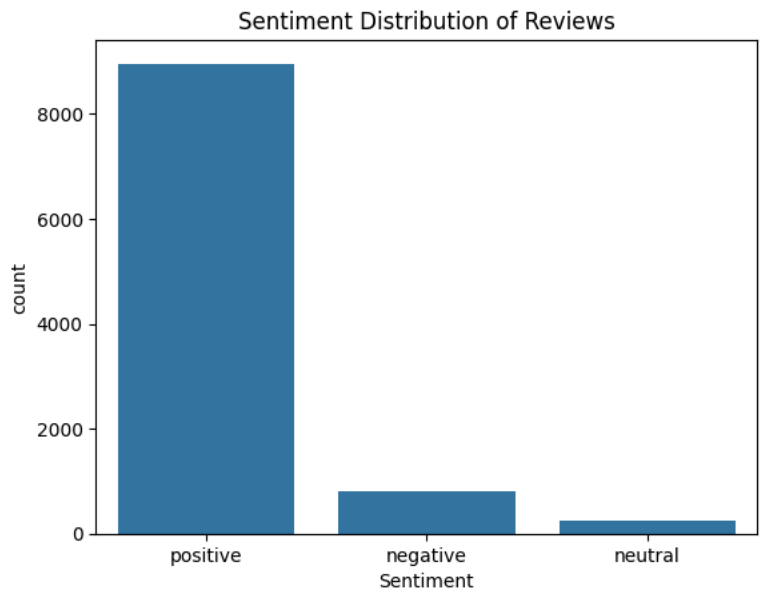
```
# Convert Time to Year
df['Year'] = pd.to_datetime(df['Time'], unit='s').dt.year

# Step 4: Time-Series Plot
reviews_by_year = df.groupby('Year').size()
plt.figure(figsize=(10, 6))
reviews_by_year.plot(kind='line', marker='o')
plt.title('Number of Reviews Over Time')
plt.xlabel('Year')
plt.ylabel('Number of Reviews')
plt.grid(True)
plt.show()
```



Step 35: Sentiment analysis with VADER(Valence Aware Dictionary and sEntiment Reasoner)

```
# Initialize VADER
sid = SentimentIntensityAnalyzer()

# Function to get sentiment
def get_sentiment(text):
    score = sid.polarity_scores(text)['compound']
    return 'positive' if score >= 0.05 else 'negative' if score <= -0.05 else 'neutral'

# Apply sentiment analysis
df['Sentiment'] = df['Text_Cleaned'].apply(get_sentiment)
print("\nSentiment Distribution:")
print(df['Sentiment'].value_counts())

# Visualize sentiment distribution
sns.countplot(x='Sentiment', data=df)
plt.title('Sentiment Distribution of Reviews')
plt.show()
```

```
Sentiment Distribution:
Sentiment
positive    8956
negative     805
neutral      239
Name: count, dtype: int64
```



Step 36: Statistical summary

```
# Step 7: Statistical Summary
print("\nStatistical Summary of Numerical Columns:")
print(df[['HelpfulnessNumerator', 'HelpfulnessDenominator', 'Score', 'Time']].describe())
```

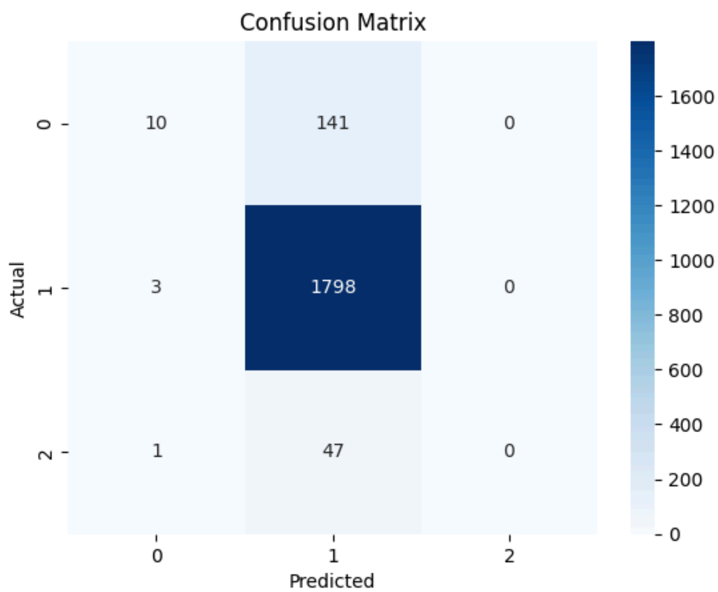
```
Statistical Summary of Numerical Columns:
       HelpfulnessNumerator  HelpfulnessDenominator         Score  \
count          10000.000000            10000.000000  10000.000000   
mean               1.658600                2.133200      4.148100   
std                6.482349                7.039049      1.324742   
min                0.000000                0.000000      1.000000   
25%                0.000000                0.000000      4.000000   
50%                0.000000                1.000000      5.000000   
75%                2.000000                2.000000      5.000000   
max              398.000000              401.000000      5.000000   

               Time  
count  1.000000e+04  
mean   1.294707e+09  
std    4.794754e+07  
min    1.067040e+09  
25%    1.268438e+09  
50%    1.308787e+09  
```

```
75%     1.331510e+09
max     1.351210e+09
```

Step 37: TF-IDF and model training(Term Frequency-Inverse Document Frequency)

```
# TF-IDF Vectorization
tfidf = TfidfVectorizer(max_features=5000)
X = tfidf.fit_transform(df['Text_Cleaned'])
y = df['Sentiment'].map({'positive': 1, 'negative': 0, 'neutral': 2})  # Encode sentiment

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predict and evaluate
y_pred = rf_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred, average='weighted')

print(f"\nModel Accuracy: {accuracy:.2f}")
print(f"Model F1 Score: {f1:.2f}")

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Model Accuracy: 0.90
Model F1 Score: 0.86
```



Step 38: Save Model and Vectorizer

```
import joblib


joblib.dump(rf_model, 'rf_sentiment_model.pkl')
joblib.dump(tfidf, 'tfidf_vectorizer.pkl')
print("\nModel and Vectorizer saved successfully.")
```

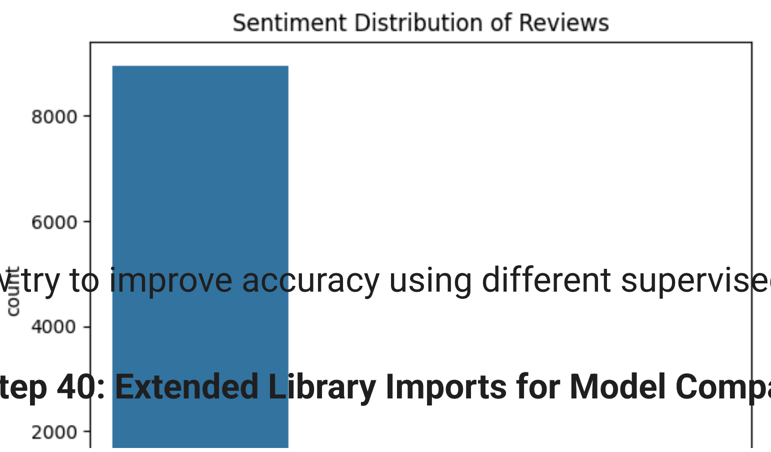
```
Model and Vectorizer saved successfully.
```

Step 39: Full Pipeline Recap & Initial Evaluation

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from wordcloud import WordCloud
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, classification_report, roc_auc_score, roc_curve
from sklearn.preprocessing import label_binarize
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.corpus import stopwords
import nltk
import re
import joblib

# Download NLTK data
nltk.download('vader_lexicon')
nltk.download('stopwords')

# Load the cleaned dataset
df = pd.read_csv('/content/Reviews_cleaned.csv')
print("Dataset Loaded Successfully:")
print(df.head())

# Step 1: Create Text_Cleaned (if not already present)
def clean_text(text):
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    stop_words = set(stopwords.words('english'))
    text = ' '.join(word for word in text.split() if word.lower() not in stop_words)
    return text

df['Text_Cleaned'] = df['Text'].apply(clean_text)
print("\nSample of Text_Cleaned:")
print(df['Text_Cleaned'].head())

# Step 2: Sentiment Analysis to Create Sentiment Column
sid = SentimentIntensityAnalyzer()

def get_sentiment(text):
    score = sid.polarity_scores(text)['compound']
    return 'positive' if score >= 0.05 else 'negative' if score <= -0.05 else 'neutral'

df['Sentiment'] = df['Text_Cleaned'].apply(get_sentiment)
print("\nSentiment Distribution:")
print(df['Sentiment'].value_counts())

# Visualize sentiment distribution
sns.countplot(x='Sentiment', data=df)
plt.title('Sentiment Distribution of Reviews')
plt.show()

# Step 3: TF-IDF Vectorization and Model Training
tfidf = TfidfVectorizer(max_features=5000)
X = tfidf.fit_transform(df['Text_Cleaned'])
y = df['Sentiment'].map({'positive': 1, 'negative': 0, 'neutral': 2})

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predictions
y_pred = rf_model.predict(X_test)

# Step 4: Model Evaluation
# 4.1: Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=['negative', 'positive', 'neutral']))

# 4.2: Confusion Matrix
```

```
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['negative', 'positive', 'neutral'], yticklabels=['negative', 'positive', 'ne
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# 4.3: Cross-Validation
cv_scores = cross_val_score(rf_model, X, y, cv=5, scoring='accuracy')
print("\nCross-Validation Accuracy Scores:", cv_scores)
print(f"Mean CV Accuracy: {cv_scores.mean():.2f}")
print(f"Standard Deviation: {cv_scores.std():.2f}")

# 4.4: ROC-AUC (One-vs-Rest)
y_test_bin = label_binarize(y_test, classes=[0, 1, 2])
y_pred_prob = rf_model.predict_proba(X_test)

roc_auc = {}
for i, label in enumerate(['negative', 'positive', 'neutral']):
    roc_auc[label] = roc_auc_score(y_test_bin[:, i], y_pred_prob[:, i])
    fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_pred_prob[:, i])
    plt.plot(fpr, tpr, label=f'ROC {label} (AUC = {roc_auc[label]:.2f})')

plt.plot([0, 1], [0, 1], 'k--')
plt.title('ROC Curve (One-vs-Rest)')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()

print("\nROC-AUC Scores:")
for label, score in roc_auc.items():
    print(f"{label}: {score:.2f}")

# 4.5: Feature Importance
feature_names = tfidf.get_feature_names_out()
importances = rf_model.feature_importances_
top_n = 20
indices = np.argsort(importances)[::-1][:top_n]

plt.figure(figsize=(10, 6))
plt.barh(range(top_n), importances[indices], align='center')
plt.yticks(range(top_n), [feature_names[i] for i in indices])
plt.xlabel('Feature Importance')
plt.title('Top 20 Important Features (Words)')
plt.gca().invert_yaxis()
plt.show()

# 4.6: Error Analysis
error_df = pd.DataFrame({'Text': df.loc[y_test.index, 'Text'], 'Actual': y_test, 'Predicted': y_pred})
error_df['Actual'] = error_df['Actual'].map({0: 'negative', 1: 'positive', 2: 'neutral'})
error_df['Predicted'] = error_df['Predicted'].map({0: 'negative', 1: 'positive', 2: 'neutral'})

misclassified = error_df[error_df['Actual'] != error_df['Predicted']]
print("\nSample of Misclassified Reviews (First 5):")
print(misclassified.head())

misclassified.to_csv('/content/misclassified_reviews.csv', index=False)
print("\nMisclassified reviews saved as 'misclassified_reviews.csv'")
```

```
Dataset Loaded Successfully:
      Id   ProductId          UserId                              ProfileName  \
0  44712  B001EQ55RW   AQ8DU6XVA3USJ  alejandra vernon "artist & illustrator"
1  29841  B000F9XBJ8  A33DKGANHQJSXC                       anastasia "stasia"
2  42811  B002NHYQAS   A22PUBSSNP54L                 g. little "value seeker"
3  32661  B0083QJU72  A2XJKCHDT5NNAA                           monica kim "mk"
4  47394  B000OQ2JBS  A261H8DCNDAQ3J        d. m. pheneger "hungry for truth"

   HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
0                     2                       4      5  1211241600
1                     0                       0      5  1245715200
2                     0                       0      5  1298419200
3                     0                       0      5  1305763200
4                     1                       1      5  1209945600

                                      Summary  \
0  the subtle and sophisticated chocolate almond
1               very different than the original
2                              smooth and rich ...
3                              great tasting syrup
4                        best all around condiment

                                                Text
0  this most unusual dry roasted almond from emer...
1  but very delicious just the same. if you're lo...
2  smooth and rich and so yummy!  what more can y...
3  wow, reviewers weren't kidding.  this was the ...
4  as a chef in the food service field i have use...
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!

Sample of Text_Cleaned:
0    unusual dry roasted almond emerald lightly coa...
1    delicious youre looking wee bit different flav...
2    smooth rich yummy say good chocolate well also...
3    wow reviewers werent kidding best tasting mapl...
4    chef food service field used sauce many differ...
Name: Text_Cleaned, dtype: object

Sentiment Distribution:
Sentiment
positive    8956
negative     805
neutral      239
Name: count, dtype: int64
```



Now try to improve accuracy using different supervised learning algorithms

Step 40: Extended Library Imports for Model Comparison

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from wordcloud import WordCloud
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, f1_score, classification_report, confusion_matrix
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_auc_score, roc_curve
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.corpus import stopwords
import nltk
import re
import joblib
```

```
from sklearn.cluster import KMeans, DBSCAN
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
```

Step 41: Reload Dataset for Model Comparison Phase

```
# Load the cleaned dataset
df = pd.read_csv('/content/Reviews_cleaned.csv')
```

Step 42: Text Cleaning (Re-applied for Safety or Pipeline Reuse)

```
# Clean text
def clean_text(text):
    text = re.sub(r'[^\w\s]', '', text)
    stop_words = set(stopwords.words('english'))
    text = ' '.join(word for word in text.split() if word.lower() not in stop_words)
    return text

df['Text_Cleaned'] = df['Text'].apply(clean_text)
```

Step 43: Sentiment Analysis + TF-IDF + Train-Test Split (Reused Setup)

```
Cross-Validation Accuracy Scores: [0.9    0.9    0.8995 0.901  0.9015]
Mean CV Accuracy: 0.90
```

```
# Sentiment Analysis
sid = SentimentIntensityAnalyzer()
def get_sentiment(text):
    score = sid.polarity_scores(text)['compound']
    return 'positive' if score >= 0.05 else 'negative' if score <= -0.05 else 'neutral'
df['Sentiment'] = df['Text_Cleaned'].apply(get_sentiment)

# TF-IDF Vectorization
tfidf = TfidfVectorizer(max_features=5000)
X = tfidf.fit_transform(df['Text_Cleaned'])
y = df['Sentiment'].map({'positive': 1, 'negative': 0, 'neutral': 2})

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

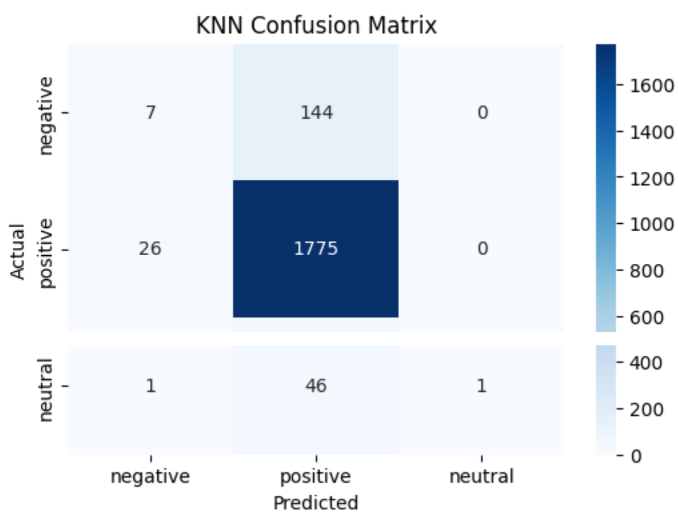
Step 44: Train & Evaluate Multiple Models

```
# Initialize models
models = {
    'KNN': KNeighborsClassifier(n_neighbors=5),
    'SVM': SVC(kernel='linear', probability=True, random_state=42),
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42),
    'Gradient Boosting': GradientBoostingClassifier(random_state=42)
}

# Evaluate each model
results = {}
for name, model in models.items():
    # Train
    model.fit(X_train, y_train)
    # Predict
    y_pred = model.predict(X_test)
    # Metrics
    accuracy = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='weighted')
    results[name] = {'Accuracy': accuracy, 'F1 Score': f1}
    print(f"\n{name} Results:")
    print(classification_report(y_test, y_pred, target_names=['negative', 'positive', 'neutral']))
    # Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
```

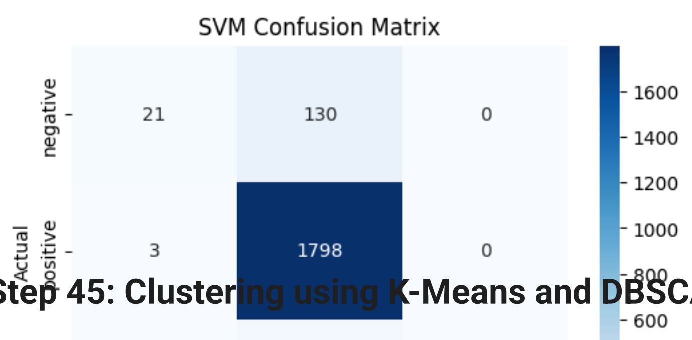
```
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['negative', 'positive', 'neutral'], yticklabels=['negative', 'positive',
    plt.title(f'{name} Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

# Summary of results
print("\nModel Performance Summary:")
for name, metrics in results.items():
    print(f"{name}: Accuracy = {metrics['Accuracy']:.2f}, F1 Score = {metrics['F1 Score']:.2f}")

# Select best model based on F1 Score (more balanced metric for multi-class)
best_model_name = max(results, key=lambda x: results[x]['F1 Score'])
best_model = models[best_model_name]
print(f"\nBest Model: {best_model_name} with F1 Score = {results[best_model_name]['F1 Score']:.2f}")
```

```
2750  of course they don't taste quite as good as "r...  negative  positive
7487  i ordered this after having much success with ...  negative  positive
3999  wow---2 pounds of hard candy is a lot-lol.  th...   neutral  positive
4640  crackers are okay, had been broken into small ...  negative  positive

Misclassified reviews saved as 'misclassified_reviews.csv'
```

```
KNN Results:
              precision    recall  f1-score   support

    negative       0.21      0.05      0.08       151
    positive       0.90      0.99      0.94      1801
     neutral       1.00      0.02      0.04        48

    accuracy                           0.89      2000
   macro avg       0.70      0.35      0.35      2000
weighted avg       0.85      0.89      0.86      2000
```




```
SVM Results:
              precision    recall  f1-score   support

    negative       0.84      0.14      0.24       151
    positive       0.91      1.00      0.95      1801
     neutral       0.00      0.00      0.00        48

    accuracy                           0.91      2000
   macro avg       0.58      0.38      0.40      2000
weighted avg       0.88      0.91      0.88      2000
```

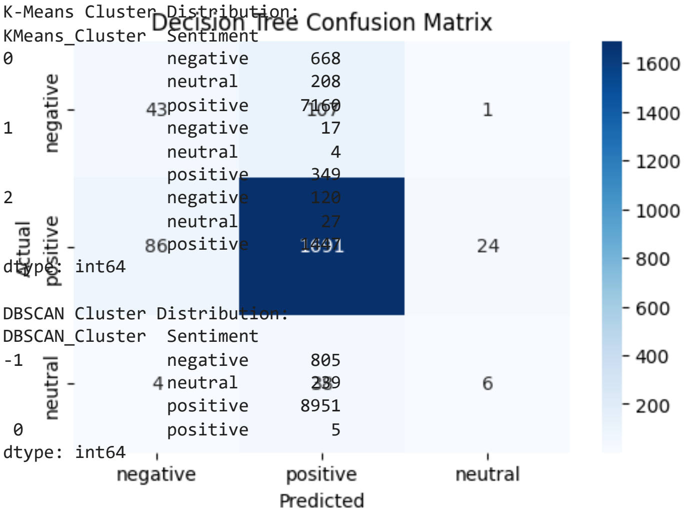


Step 45: Clustering using K-Means and DBSCAN

```
# K-Means Clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans_labels = kmeans.fit_predict(X)
```

```
df['KMeans_Cluster'] = kmeans_labels
print("\nK-Means Cluster Distribution:")
print(df.groupby(['KMeans_Cluster', 'Sentiment']).size())

# DBSCAN Clustering
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan_labels = dbscan.fit_predict(X.toarray())  # Convert to dense array for DBSCAN
df['DBSCAN_Cluster'] = dbscan_labels
print("\nDBSCAN Cluster Distribution:")
print(df.groupby(['DBSCAN_Cluster', 'Sentiment']).size().head())  # Limited output due to noise (-1)

# Note: Clustering may not align well with sentiment due to high-dimensional TF-IDF space
```

```
K-Means Cluster Distribution:
KMeans_Cluster  Sentiment
0               negative      668
                neutral       208
                positive     7160
1               negative       17
                neutral         4
                positive      349
2               negative      120
                neutral        27
                positive     1447
dtype: int64

DBSCAN Cluster Distribution:
DBSCAN_Cluster  Sentiment
-1              negative      805
                neutral       239
                positive     8951
 0              positive        5
dtype: int64
```



Step 46: Predicting sentiment for a new review using the best ML model

```
Random Forest Results:
              precision    recall  f1-score   support
```

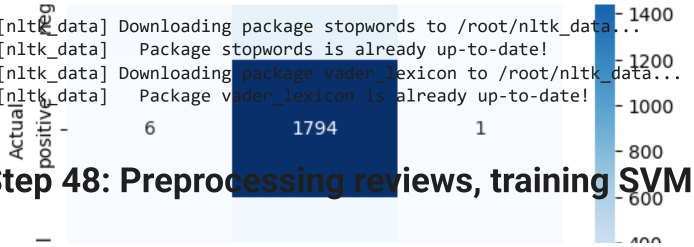
```
# New review
new_review = "This product is absolutely amazing and works perfectly!"
new_review_cleaned = clean_text(new_review)
new_review_tfidf = tfidf.transform([new_review_cleaned])
new_prediction = best_model.predict(new_review_tfidf)
sentiment_map = {0: 'negative', 1: 'positive', 2: 'neutral'}
print(f"\nPrediction for new review: {sentiment_map[new_prediction[0]]}")
print(f"New review text: {new_review}")

# Probability (if available)
if hasattr(best_model, 'predict_proba'):
    new_prob = best_model.predict_proba(new_review_tfidf)
    print(f"Prediction probabilities: {dict(zip(['negative', 'positive', 'neutral'], new_prob[0]))}")
```

```
Prediction for new review: positive
New review text: This product is absolutely amazing and works perfectly!
Prediction probabilities: {'negative': np.float64(0.021551453613345345), 'positive': np.float64(0.9427319621438653), 'neutral': np.float
```

Step 47: Importing essential libraries and downloading NLTK resources

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from nltk.corpus import stopwords
import re
import joblib
from sklearn.model_selection import train_test_split

# Download NLTK data (if not already downloaded)
try:
    import nltk
```

```
    nltk.download('stopwords')
    nltk.download('vader_lexicon')
except:
    pass
```




```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

Step 48: Preprocessing reviews, training SVM model, and saving it

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from nltk.corpus import stopwords
import re
import joblib

# Load dataset
df = pd.read_csv('/content/Reviews_cleaned.csv')  # Use Reviews.csv since Reviews_cleaned.csv may not exist

# Create Sentiment column from Score
df['Sentiment'] = df['Score'].apply(lambda x: 'positive' if x >= 4 else 'negative' if x <= 2 else 'neutral')

# Clean text function
def clean_text(text):
    # Handle non-string inputs (e.g., if Text is missing or not a string)
    if not isinstance(text, str):
        text = str(text)
    text = re.sub(r'[^\w\s]', '', text)
    stop_words = set(stopwords.words('english')) - {'not'}
    text = ' '.join(word for word in text.split() if word.lower() not in stop_words)
    return text

# Apply text cleaning
df['Text_Cleaned'] = df['Text'].apply(clean_text)

# Initialize and fit the TF-IDF vectorizer
tfidf = TfidfVectorizer(max_features=5000, ngram_range=(1, 2))
X = tfidf.fit_transform(df['Text_Cleaned'])  # Fit and transform in one step
y = df['Sentiment'].map({'positive': 1, 'negative': 0, 'neutral': 2})

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train SVM model
svm = SVC(kernel='linear', probability=True, random_state=42)
svm.fit(X_train, y_train)

# Save the vectorizer and model for future use
joblib.dump(tfidf, 'tfidf_vectorizer.pkl')
joblib.dump(svm, 'svm_model.pkl')

# Function to predict sentiment
def predict_sentiment(review):
    review_cleaned = clean_text(review)
    review_tfidf = tfidf.transform([review_cleaned])  # Transform the new review
    sentiment_map = {0: 'negative', 1: 'positive', 2: 'neutral'}
    prediction = svm.predict(review_tfidf)
    probs = svm.predict_proba(review_tfidf)[0]
    probs_dict = dict(zip(['negative', 'positive', 'neutral'], probs))
    print(f"\nReview: {review}")
    print(f"SVM Prediction: {sentiment_map[prediction[0]]}")
    print(f"SVM Probabilities: {probs_dict}")

# Test the review
predict_sentiment("The taste is not so good")
```

```
Review: The taste is not so good
SVM Prediction: negative
SVM Probabilities: {'negative': np.float64(0.4658113807951335), 'positive': np.float64(0.1887401983855602), 'neutral': np.float64(0.3454
```

Step 49: Displaying sentiment distribution

```
print(df['Sentiment'].value_counts())
```

```
Sentiment
positive    7693
negative    1508
neutral      799
Name: count, dtype: int64
```

Step 50: Testing the text cleaning function

```
print(clean_text("The taste is not so good"))  # Should output: "taste not good"
```

```
taste not good
```

Step 51: Sentiment prediction using Hugging Face’s transformer pipeline

```
from transformers import pipeline
sentiment_analyzer = pipeline("sentiment-analysis")
print(sentiment_analyzer("The taste is not so good"))
```

```
No model was supplied, defaulted to distilbert/distilbert-base-uncased-finetuned-sst-2-english and revision 714eb0f (https://huggingface
Using a pipeline without specifying a model name and revision in production is not recommended.
```

config.json: 100% 629/629 [00:00<00:00, 69.2kB/s]

model.safetensors: 100% 268M/268M [00:03<00:00, 98.9MB/s]

tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 5.04kB/s]

vocab.txt: 100% 232k/232k [00:00<00:00, 3.59MB/s]

```
Device set to use cuda:0
[{'label': 'NEGATIVE', 'score': 0.9997603297233582}]
```

Step 52: Interactive sentiment analysis using Hugging Face pipeline

Double-click (or enter) to edit

```
from transformers import pipeline

# Initialize the sentiment analysis pipeline
sentiment_analyzer = pipeline("sentiment-analysis")

# Function to predict sentiment
def predict_sentiment(review):
    result = sentiment_analyzer(review)
    print(f"\nReview: {review}")
    print(f"Sentiment: {result[0]['label']}")
    print(f"Confidence Score: {result[0]['score']:.4f}")

# Main loop to take user input
while True:
    user_review = input("\nPlease enter a review (or type 'exit' to quit): ")
    if user_review.lower() == 'exit':
        print("Exiting the program. Goodbye!")
        break
    if not user_review.strip():
        print("Please enter a valid review.")
        continue
    predict_sentiment(user_review)
```

```
No model was supplied, defaulted to distilbert/distilbert-base-uncased-finetuned-sst-2-english and revision 714eb0f (https://huggingface
Using a pipeline without specifying a model name and revision in production is not recommended.
Device set to use cuda:0

Please enter a review (or type 'exit' to quit): The taste was good

Review: The taste was good
Sentiment: POSITIVE
Confidence Score: 0.9999

Please enter a review (or type 'exit' to quit): exit
Exiting the program. Goodbye!
```

Step 53: Final pipeline with SVM + BERT comparison, using SMOTE for balancing

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from imblearn.over_sampling import SMOTE
from nltk.corpus import stopwords
import re
import joblib
from transformers import pipeline
import csv

# Initialize BERT-based sentiment analyzer
sentiment_analyzer = pipeline("sentiment-analysis", model="nlptown/bert-base-multilingual-uncased-sentiment"

# Load dataset
df = pd.read_csv('/content/Reviews_cleaned.csv')

# Create Sentiment column from Score
df['Sentiment'] = df['Score'].apply(lambda x: 'positive' if x >= 4 else 'negative' if x <= 2 else 'neutral')

# Check sentiment distribution before SMOTE
print("Sentiment Distribution Before SMOTE:")
print(df['Sentiment'].value_counts())

# Clean text function
def clean_text(text):
    if not isinstance(text, str):
        text = str(text)
    text = re.sub(r'[^\w\s]', '', text)
    stop_words = set(stopwords.words('english')) - {'not', 'no', 'never'}
    text = ' '.join(word for word in text.split() if word.lower() not in stop_words)
    return text

# Apply text cleaning
df['Text_Cleaned'] = df['Text'].apply(clean_text)

# Check cleaned text for the review
test_review = "The taste is not so good"
print(f"\nCleaned review: {clean_text(test_review)}")

# Initialize and fit the TF-IDF vectorizer
tfidf = TfidfVectorizer(max_features=5000, ngram_range=(1, 2))
X = tfidf.fit_transform(df['Text_Cleaned'])
y = df['Sentiment'].map({'positive': 1, 'negative': 0, 'neutral': 2})

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply SMOTE to balance the training set
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train, y_train)

# Check sentiment distribution after SMOTE
print("\nSentiment Distribution After SMOTE (Training Set):")
print(pd.Series(y_train_balanced).value_counts())

# Train SVM model on balanced data
```

```
svm = SVC(kernel='linear', probability=True, random_state=42)
svm.fit(X_train_balanced, y_train_balanced)

# Save the vectorizer and model
joblib.dump(tfidf, 'tfidf_vectorizer.pkl')
joblib.dump(svm, 'svm_model.pkl')

# Evaluate SVM model on test set
y_pred = svm.predict(X_test)
print("\nSVM Model Performance on Test Set:")
print(classification_report(y_test, y_pred, target_names=['negative', 'positive', 'neutral']))

# Function to predict sentiment with SVM
def predict_sentiment_svm(review):
    review_cleaned = clean_text(review)
    review_tfidf = tfidf.transform([review_cleaned])
    sentiment_map = {0: 'negative', 1: 'positive', 2: 'neutral'}
    prediction = svm.predict(review_tfidf)
    probs = svm.predict_proba(review_tfidf)[0]
    probs_dict = dict(zip(['negative', 'positive', 'neutral'], probs))
    print(f"\nSVM Prediction for Review: {review}")
    print(f"SVM Sentiment: {sentiment_map[prediction[0]]}")
    print(f"SVM Probabilities: {probs_dict}")

# Function to predict sentiment with BERT
def predict_sentiment_bert(review):
    result = sentiment_analyzer(review)
    star_rating = int(result[0]['label'].split()[0])
    sentiment = 'positive' if star_rating >= 4 else 'negative' if star_rating <= 2 else 'neutral'
    print(f"\nBERT Prediction for Review: {review}")
    print(f"BERT Sentiment: {sentiment}")
    print(f"BERT Confidence Score: {result[0]['score']:.4f} (Star Rating: {star_rating})")

# Test the review with both models
predict_sentiment_svm(test_review)
predict_sentiment_bert(test_review)

# Main loop for user input
while True:
    user_review = input("\nPlease enter a review (or type 'exit' to quit): ")
    if user_review.lower() == 'exit':
        print("Exiting the program. Goodbye!")
        break
    if not user_review.strip():
        print("Please enter a valid review.")
```