Sentiment Analysis

We will be using the Reviews.csv file from Kaggle's Amazon Fine Food Reviews dataset to perform the analysis.

I use a Jupyter Notebook for all analysis and visualization.

Step 1: Read the Dataframe

```
import pandas as pd

df = pd.read_csv('Reviews.csv')

df.head()
```

Checking the head of the dataframe:

We can see that the dataframe contains some product, user and review information.

The data that we will be using most for this analysis is "Summary", "Text", and "Score."

Text — This variable contains the complete product review information.

Summary — This is a summary of the entire review.

Score — The product rating provided by the customer.

Step 2: Data Analysis

Now, we will take a look at the variable "Score" to see if majority of the customer ratings are positive or negative: install the Plotly library first.

```
# Importsimport matplotlib.pyplot as plt

import seaborn as sns

color = sns.color_palette()

%matplotlib inline
```

```
import plotly.offline as py

py.init_notebook_mode(connected=True)

import plotly.graph_objs as go

import plotly.tools as tls

import plotly.express as px# Product Scoresfig = px.histogram(df, x="Score")

fig.update_traces(marker_color="turquoise",marker_line_color='rgb(8,48,107)',

                  marker_line_width=1.5)

fig.update_layout(title_text='Product Score')

fig.show()
```

We can see that most of the customer rating is positive. This leads me to believe that most reviews will be pretty positive too, which will be analyzed in a while.

Now, we can create some wordclouds to see the most frequently used words in the reviews.

```
import nltk

from nltk.corpus import stopwords# Create stopword list:

stopwords = set(STOPWORDS)

stopwords.update(["br", "href"])

textt = " ".join(review for review in df.Text)

wordcloud = WordCloud(stopwords=stopwords).generate(textt)plt.imshow(wordcloud,
interpolation='bilinear')

plt.axis("off")

plt.savefig('wordcloud11.png')

plt.show()
```

Running the code above generates a word cloud that looks like this:

Some popular words that can be observed here include “taste,” “product,” “love,” and “Amazon.” These words are mostly positive, also indicating that most reviews in the dataset express a positive sentiment.

Step 3: Classifying Tweets

In this step, we will classify reviews into “positive” and “negative,” so we can use this as training data for our sentiment classification model.

Positive reviews will be classified as +1, and negative reviews will be classified as -1.

We will classify all reviews with ‘Score’ > 3 as +1, indicating that they are positive.

All reviews with ‘Score’ < 3 will be classified as -1. Reviews with ‘Score’ = 3 will be dropped, because they are neutral. Our model will only classify positive and negative reviews.

assign reviews with score > 3 as positive sentiment

score < 3 negative sentiment

remove score = 3df = df[df['Score'] != 3]

df['sentiment'] = df['Score'].apply(lambda rating : +1 if rating > 3 else -1)

Looking at the head of the data frame now, we can see a new column called ‘sentiment:’

Step 4: More Data Analysis

Now that we have classified tweets into positive and negative, let’s build wordclouds for each:

First, we will create two data frames — one with all the positive reviews, and another with all the negative reviews.

split df - positive and negative sentiment:positive = df[df['sentiment'] == 1]

negative = df[df['sentiment'] == -1]

Wordcloud — Positive Sentiment

stopwords = set(STOPWORDS)

stopwords.update(["br", "href","good","great"]) ## good and great removed because they were included in negative sentimentpos = " ".join(review for review in positive.Summary)

wordcloud2 = WordCloud(stopwords=stopwords).generate(pos)plt.imshow(wordcloud2, interpolation='bilinear')

plt.axis("off")

plt.show()

Wordcloud — Negative Sentiment

```
neg = " ".join(review for review in negative.Summary)

wordcloud3 = WordCloud(stopwords=stopwords).generate(neg)plt.imshow(wordcloud3, interpolation='bilinear')

plt.axis("off")

plt.savefig('wordcloud33.png')

plt.show()
```

As seen above, the positive sentiment word cloud was full of positive words, such as “love,” “best,” and “delicious.”

The negative sentiment word cloud was filled with mostly negative words, such as “disappointed,” and “yuck.”

The words “good” and “great” initially appeared in the negative sentiment word cloud, despite being positive words. This is probably because they were used in a negative context, such as “not good.” Due to this, I removed those two words from the word cloud.

Finaly, we can take a look at the distribution of reviews with sentiment across the dataset:

```
df['sentimentt'] = df['sentiment'].replace({-1 : 'negative'})

df['sentimentt'] = df['sentimentt'].replace({1 : 'positive'})

fig = px.histogram(df, x="sentimentt")

fig.update_traces(marker_color="indianred",marker_line_color='rgb(8,48,107)',

                  marker_line_width=1.5)

fig.update_layout(title_text='Product Sentiment')

fig.show()
```

Step 5: Building the Model

Finally, we can build the sentiment analysis model!

This model will take reviews in as input. It will then come up with a prediction on whether the review is positive or negative.

This is a classification task, so we will train a simple logistic regression model to do it.

For reference, take a look at the data frame again:

There are a few steps we need to take:

Data Cleaning

We will be using the summary data to come up with predictions. First, we need to remove all punctuation from the data.

```
def remove_punctuation(text):
    final = "".join(u for u in text if u not in ("?", ".", ";", ":",  "!",'"'))
    return finaldf['Text'] = df['Text'].apply(remove_punctuation)
df = df.dropna(subset=['Summary'])
df['Summary'] = df['Summary'].apply(remove_punctuation)
```

Split the Dataframe

The new data frame should only have two columns — “Summary” (the review text data), and “sentiment” (the target variable).

```
dfNew = df[['Summary','sentiment']]
dfNew.head()
```

Taking a look at the head of the new data frame, this is the data it will now contain:

We will now split the data frame into train and test sets. 80% of the data will be used for training, and 20% will be used for testing.

```
# random split train and test dataindex = df.index
df['random_number'] = np.random.randn(len(index))train = df[df['random_number'] <= 0.8]
```

```
test = df[df['random_number'] > 0.8]
```

Create a bag of words

Next, we will use a count vectorizer from the Scikit-learn library.

This will transform the text in our data frame into a bag of words model, which will contain a sparse matrix of integers. The number of occurrences of each word will be counted and printed.

We will need to convert the text into a bag-of-words model since the logistic regression algorithm cannot understand text.

```
# count vectorizer:from sklearn.feature_extraction.text import CountVectorizervectorizer = CountVectorizer(token_pattern=r'\b\w+\b')train_matrix = vectorizer.fit_transform(train['Summary'])

test_matrix = vectorizer.transform(test['Summary'])
```

Import Logistic Regression

```
# Logistic Regressionfrom sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
```

Split target and independent variables

```
X_train = train_matrix

X_test = test_matrix

y_train = train['sentiment']

y_test = test['sentiment']
```

Fit model on data

```
lr.fit(X_train,y_train)
```

Make predictions

```
predictions = lr.predict(X_test)
```

We have successfully built a simple logistic regression model, and trained the data on it. We also made predictions using the model.

Step 6: Testing

Now, we can test the accuracy of our model.

```
# find accuracy, precision, recall:from sklearn.metrics import confusion_matrix,classification_report

new = np.asarray(y_test)

confusion_matrix(predictions,y_test)
```

You will get a confusion matrix:

```
print(classification_report(predictions,y_test))
```

The classification report:

The overall accuracy of the model on the test data is around 93%, which is pretty good considering we didn't do any feature extraction or much preprocessing.