Step 1 : Read in Data and NLTK Basics

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

plt.style.use('ggplot') # Set the style for plots

import nltk # Download the NLTK data files
import re
import spacy
from nltk.corpus import stopwords

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC
from sklearn.metrics import classification_report, confusion_matrix

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

This dataset consists of reviews of fine foods from amazon. The data span a period of more than 10 years, including all ~500,000 reviews up to October 2012. Reviews include product and user information, ratings, and a plain text review. It also includes reviews from all other Amazon categories.

```
df.head()
```

```
   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...
```

```
df['Text'].values[0]
```

```
'I have bought several of the Vitality canned dog food products and have found them all to be of good quality. The product looks more like a stew than a processed meat and it smells better. My Labrador is finicky and she appreciates this product better than  most.'
```

```
print(df.shape)
```

```
(568454, 10)
```

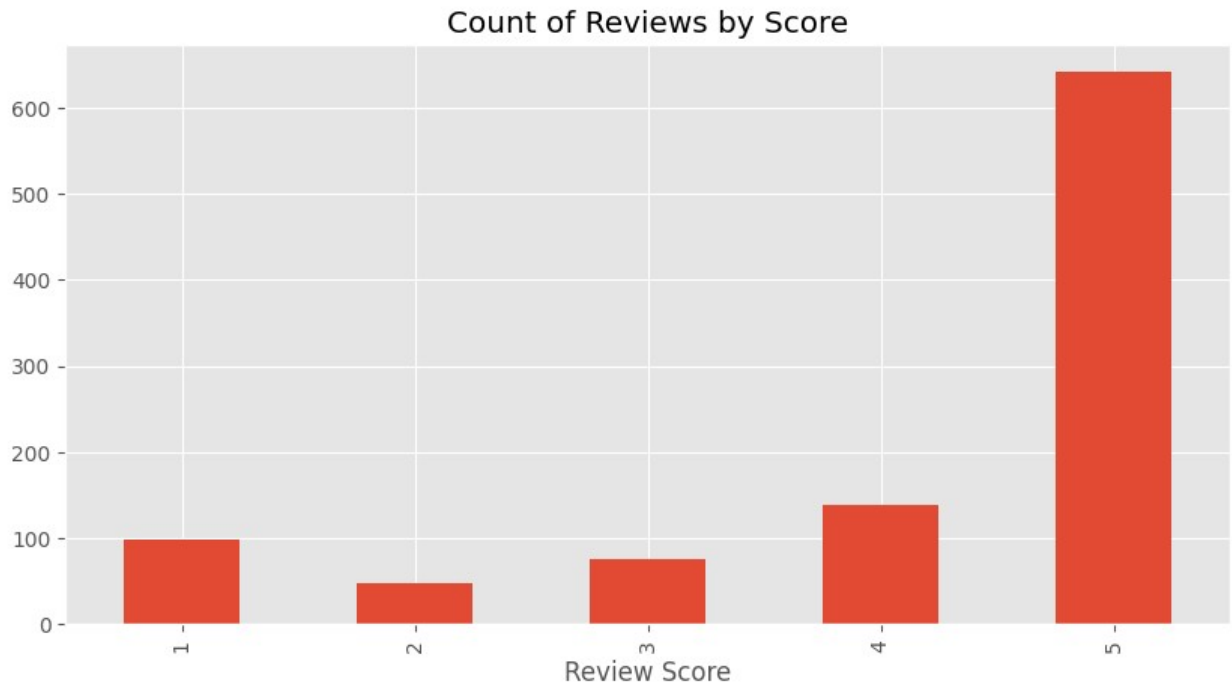
We can see here the that dataset is very large (half a million reviews)

```
df = df.head(1000) # Limit to 1000 rows for faster processing
df.shape
```

```
(1000, 10)
```

Quick EDA

```
ax = df['Score'].value_counts().sort_index()\
    .plot(kind='bar',
          title='Count of Reviews by Score',
          figsize=(10, 5))
ax.set_xlabel('Review Score')
plt.show()
```



We can see that most reviews tend to be 5 stars in our dataset

Basic NLTK

```
example = df['Text'].values[50]
print(example)
```

```
This oatmeal is not good. Its mushy, soft, I don't like it. Quaker
Oats is the way to go.
```

```
tokens = nltk.word_tokenize(example)
tokens[:10] # Show the first 10
```

```
['This', 'oatmeal', 'is', 'not', 'good', '.', 'Its', 'mushy', ',',
'soft']
```

```
#nltk.download('averaged_perceptron_tagger_eng')
tagged = nltk.pos_tag(tokens)
tagged[:10] # Show the first 10 tagged tokens
```

```
[('This', 'DT'),
 ('oatmeal', 'NN'),
 ('is', 'VBZ'),
 ('not', 'RB'),
 ('good', 'JJ'),
 ('.', '.'),
 ('Its', 'PRP$'),
 ('mushy', 'NN'),
 (',', ','),
 ('soft', 'JJ')]
```

```
#nltk.download('maxent_ne_chunker_tab')
#nltk.download('words')
entitites = nltk.chunk.ne_chunk(tagged)
entitites.pprint()
```

```
(S
  This/DT
  oatmeal/NN
  is/VBZ
  not/RB
  good/JJ
  ./.
  Its/PRP$
  mushy/NN
  ,/,
  soft/JJ
  ,/,
  I/PRP
  do/VBP
  n't/RB
  like/VB
  it/PRP
  ./.
  (ORGANIZATION Quaker/NNP Oats/NNPS)
  is/VBZ
  the/DT
  way/NN
  to/TO
  go/VB
  ./.)
```

Step 2 : VADER Sentiment Scoring

The VADER (Valence Aware Dictionary and Sentiment Reasoner) approach just basicaly takes all the words in our sentence and has a value of either positive, negative or neutral for each of those words and it will add up the words to say if our sentence is positive, negative or neutral based on all those words. However, this approach does not account for relationship between words which is crucial in human speech. But it is a good start.

We are also removing stop words when using VADER like "and", "the", or words that doesnt have a positive or negative feeling ot them, they are just for the structure of the sentence.

```
from nltk.sentiment import SentimentIntensityAnalyzer
from tqdm.notebook import tqdm

#nltk.download('vader_lexicon')
sia = SentimentIntensityAnalyzer()
```

```
sia.polarity_scores('I am very happy with the product!')
```

```
{'neg': 0.0, 'neu': 0.539, 'pos': 0.461, 'compound': 0.6468}
```

So in this example, we see that the sentence is mostly positive and the compound scores (goes from -1 to 1) represent how negative to positive the sentece is.

```
sia.polarity_scores('This is the worst product I have ever bought!')
```

```
{'neg': 0.386, 'neu': 0.614, 'pos': 0.0, 'compound': -0.6588}
```

Here the sia shows the sentece is mostly negative

```
sia.polarity_scores(example)
```

```
{'neg': 0.22, 'neu': 0.78, 'pos': 0.0, 'compound': -0.5448}
```

In our previous example with the oatmeals, it is pretty high neutral but also with some negatives and the compound score is negative.

```
# Run the polarity scores on the entire dataset
res = {}
for i, row in tqdm(df.iterrows(), total=len(df)):
    text = row['Text']
    myid = row['Id']
    res[myid] = sia.polarity_scores(text)
```

```
{"model_id":"756b8f51caf3455b9e88e508e1b0e531","version_major":2,"vers
ion_minor":0}
```

```
vaders = pd.DataFrame(res).T
vaders = vaders.reset_index().rename(columns={'index': 'Id'}) #so we
can merge with the original dataframe
vaders = vaders.merge(df, how='left')

vaders.head()
```

```
   Id    neg    neu    pos  compound   ProductId          UserId  \
0   1  0.000  0.695  0.305    0.9441  B001E4KFG0  A3SGXH7AUHU8GW
1   2  0.138  0.862  0.000   -0.5664  B00813GRG4  A1D87F6ZCVE5NK
2   3  0.091  0.754  0.155    0.8265  B000LQOCH0   ABXLMWJIXXAIN
3   4  0.000  1.000  0.000    0.0000  B000UA0QIQ  A395BORC6FGVXV
4   5  0.000  0.552  0.448    0.9468  B006K2ZZ7K  A1UQRSCLF8GW1T

                     ProfileName  HelpfulnessNumerator  \
0                       delmartian                     1
1                           dll pa                     0
2  Natalia Corres "Natalia Corres"                     1
3                             Karl                     3
4    Michael D. Bigham "M. Wassir"                     0
```

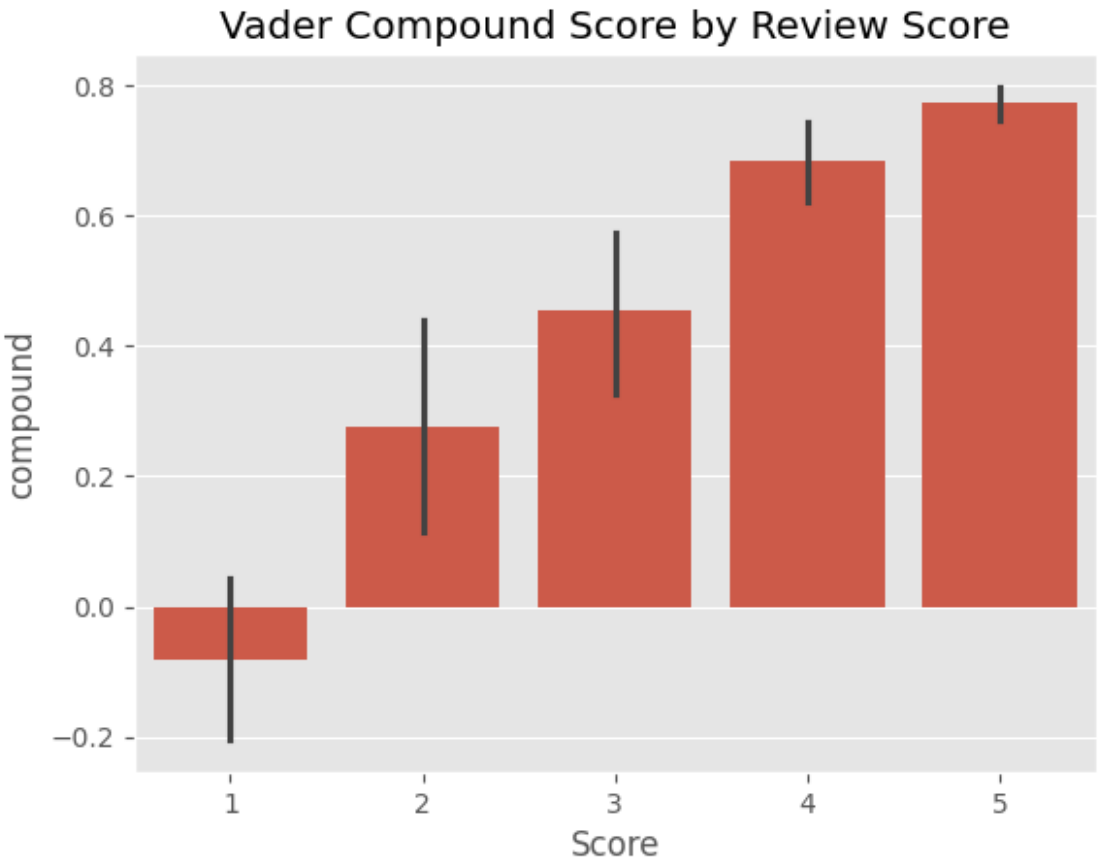
```
   HelpfulnessDenominator  Score        Time                Summary  \
0                       1      5  1303862400  Good Quality Dog Food
1                       0      1  1346976000      Not as Advertised
2                       1      4  1219017600  "Delight" says it all
3                       3      2  1307923200         Cough Medicine
4                       0      5  1350777600            Great taffy

                                                Text
0  I have bought several of the Vitality canned d...
1  Product arrived labeled as Jumbo Salted Peanut...
2  This is a confection that has been around a fe...
3  If you are looking for the secret ingredient i...
4  Great taffy at a great price.  There was a wid...
```

Now we want to look if the given compound score matches the score (from 1 to 5) of each review.

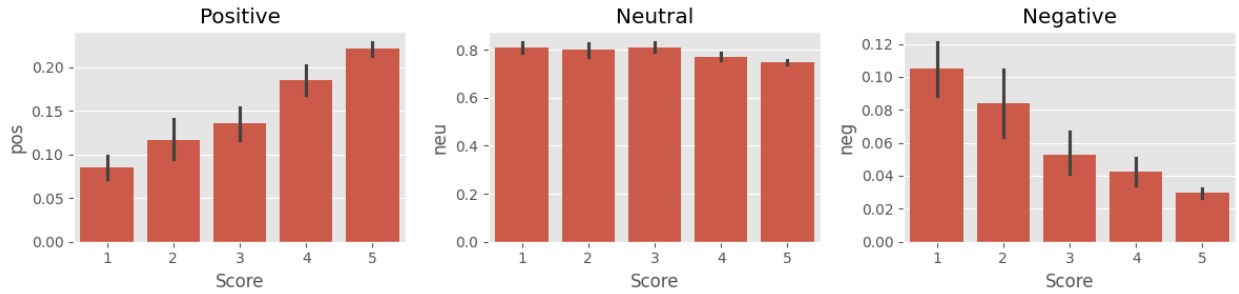
Plot VADER results

```
ax = sns.barplot(data=vaders, x='Score', y='compound')
ax.set_title('Vader Compound Score by Review Score')
plt.show()
```



So this is what we would expect. 1 star reviews have a lower compound score and 5 star reviews is high rate. So the more positive the compound becomes the more positive the text is respectively.

```
fig, axs = plt.subplots(1, 3, figsize=(12, 3))
sns.barplot(data=vaders, x='Score', y='pos', ax=axs[0])
sns.barplot(data=vaders, x='Score', y='neu', ax=axs[1])
sns.barplot(data=vaders, x='Score', y='neg', ax=axs[2])
axs[0].set_title('Positive')
axs[1].set_title('Neutral')
axs[2].set_title('Negative')
plt.tight_layout()
plt.show()
```



Positivity is higher as the score is higher in terms of stars, the neutral is flat and the negative goes down, it becomes less negative of a comment as the star review becoems higher.

This confirms what we want to see and it shows that VADER is valuable in having this realtion between the score of the text and sentiment score and that it does relate to the actual rating review of the reviewers.

Step 3 : Roberta Pretrained Model

Our previous method just took each word of the sentence and score the individually. However, human language depends a lot on context. A sentence that has negative words but could be sarcastic or related to other words in which way it makes it a positive sentence. The VADER model wouldnt pick up on that sort of relationship between words. We are going to use from (hugging face) the 'transformers' library.

```
from transformers import AutoTokenizer,
AutoModelForSequenceClassification
from scipy.special import softmax
```

Now we will take a model from that has been prettrained and we can import it so that we dont have to train the model ourselves.

```
MODEL = f"cardiffnlp/twitter-roberta-base-sentiment"
tokenizer = AutoTokenizer.from_pretrained(MODEL)
model = AutoModelForSequenceClassification.from_pretrained(MODEL)

#VADER results on example
print(example)
sia.polarity_scores(example)
```

```
This oatmeal is not good. Its mushy, soft, I don't like it. Quaker
Oats is the way to go.

{'neg': 0.22, 'neu': 0.78, 'pos': 0.0, 'compound': -0.5448}
```

```
# Run for Roberta model
encoded_txt = tokenizer(example, return_tensors='pt') # Tokenize the
text so it can be processed by the model
output = model(**encoded_txt)
scores = output[0][0].detach().numpy()
scores = softmax(scores)
scores
```

```
array([0.97635514, 0.02068746, 0.00295737], dtype=float32)
```

So basicaly this is the negative, neutral and positive score for the example text.

```
scores_dict = {
    'roberta_neg': scores[0],
    'roberta_neu': scores[1],
    'roberta_pos': scores[2]
}
print(scores_dict)
```

```
{'roberta_neg': np.float32(0.97635514), 'roberta_neu':
np.float32(0.020687463), 'roberta_pos': np.float32(0.0029573694)}
```

This makes sense as the example is a negative review of the product. We can already see how much Roberta is more powerful than the VADER model.

```
def polarity_scores_roberta(example):
    encoded_txt = tokenizer(example, return_tensors='pt') # Tokenize
the text so it can be processed by the model
    output = model(**encoded_txt)
    scores = output[0][0].detach().numpy()
    scores = softmax(scores)
    scores_dict = {
        'roberta_neg': scores[0],
        'roberta_neu': scores[1],
        'roberta_pos': scores[2]
    }
    return scores_dict
```

```
res = {}
for i, row in tqdm(df.iterrows(), total=len(df)):
    try:
        text = row['Text']
        myid = row['Id']
        vader_results = sia.polarity_scores(text)
        vader_results_rename ={}
        for key, value in vader_results.items():
            vader_results_rename[f"vader_{key}"] = value
        roberta_results = polarity_scores_roberta(text)
        both_results = {**vader_results_rename, **roberta_results} #
Merge the two dictionaries
        res[myid] = both_results
    except RuntimeError:
        print(f"Broke for id {myid}")
```


```
{"model_id":"f794df7dba4c416488cc474b23d0c3a6","version_major":2,"vers
ion_minor":0}
```


```
Broke for id 83
Broke for id 187
Broke for id 529
Broke for id 540
Broke for id 746
Broke for id 863
```

After running our previous loop, we notice that the loops breaks with an error on one of the reviews. Indeed some reviews are too long or the text is not handled by the tokenization and so we implemented a try/except structure to skip those rather than going for a lond debuging session.

```
results_df = pd.DataFrame(res).T
results_df = results_df.reset_index().rename(columns={'index': 'Id'})
#so we can merge with the original dataframe
results_df = results_df.merge(df, how='left')

results_df.head()
```

	Id	vader_neg	vader_neu	vader_pos	vader_compound	roberta_neg	\
0	1	0.000	0.695	0.305	0.9441	0.009624	
1	2	0.138	0.862	0.000	-0.5664	0.508986	
2	3	0.091	0.754	0.155	0.8265	0.003229	
3	4	0.000	1.000	0.000	0.0000	0.002295	
4	5	0.000	0.552	0.448	0.9468	0.001635	

	roberta_neu	roberta_pos	ProductId	UserId	\
0	0.049980	0.940395	B001E4KFG0	A3SGXH7AUHU8GW	
1	0.452414	0.038600	B00813GRG4	A1D87F6ZCVE5NK	
2	0.098068	0.898704	B000LQOCH0	ABXLMWJIXXAIN	
3	0.090219	0.907486	B000UA0QIQ	A395BORC6FGVXV	

```
4     0.010302     0.988063  B006K2ZZ7K  A1UQRSCLF8GW1T

                      ProfileName  HelpfulnessNumerator  \
0                        delmartian                     1
1                            dll pa                     0
2  Natalia Corres "Natalia Corres"                      1
3                              Karl                     3
4    Michael D. Bigham "M. Wassir"                      0

   HelpfulnessDenominator  Score        Time                Summary  \
0                       1      5  1303862400  Good Quality Dog Food
1                       0      1  1346976000      Not as Advertised
2                       1      4  1219017600  "Delight" says it all
3                       3      2  1307923200         Cough Medicine
4                       0      5  1350777600            Great taffy

                                                Text
0  I have bought several of the Vitality canned d...
1  Product arrived labeled as Jumbo Salted Peanut...
2  This is a confection that has been around a fe...
3  If you are looking for the secret ingredient i...
4  Great taffy at a great price.  There was a wid...
```

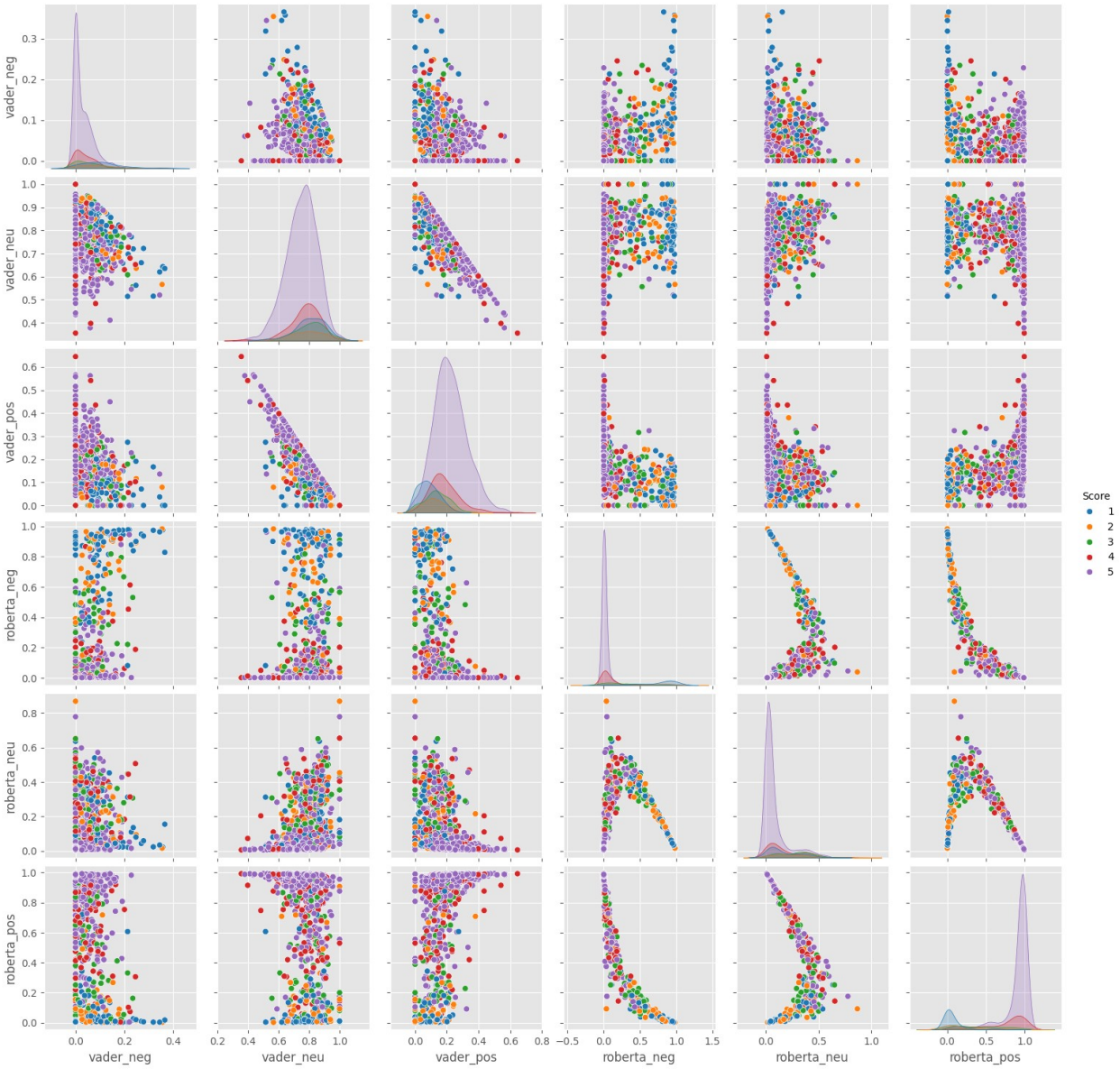
Now we have our vader scores and roberta scores for each row in the dataset.

Compare scores between models

```
results_df.columns
```

```
Index(['Id', 'vader_neg', 'vader_neu', 'vader_pos', 'vader_compound',
       'roberta_neg', 'roberta_neu', 'roberta_pos', 'ProductId',
'UserId',
       'ProfileName', 'HelpfulnessNumerator',
'HelpfulnessDenominator',
       'Score', 'Time', 'Summary', 'Text'],
      dtype='object')
```

```
sns.pairplot(data=results_df,
             vars=['vader_neg', 'vader_neu', 'vader_pos',
                 'roberta_neg', 'roberta_neu', 'roberta_pos'],
             hue='Score',
             palette='tab10')
plt.show()
```



One thing that we notice here is the 5 star reviews are this purple color and if we look at vader, the positive reviews are more or so to the right for these 5 star reviews. For the roberta model, we can see its way over to the right. And we can see that there are some correlations between the roberta model and the vader model it is a litle hard to see exactly if there are correlations. But one thing that becomes very clear is the vader model is a litle bit less confident in its predictions compared to the roberta model which realy seperates the positivity and neutral and negative scores for each of these predicted values. Also if we look at the correlation between roberta_pos and roberta_neu we can see that the roberta model has very high score for the 5 stars and most of the 1 star are very low in positivity sentiment scory which is consistent.

Step 4 : Review examples

Now we can look and see where the model does the opposite of what it should be doing.

```
results_df.query('Score == 1').sort_values('roberta_pos',
ascending=False)['Text'].values[0]
```

```
"I just wanted to post here that I found small bits of plastic in this
food as I was feeding my 9 month old.  Plastic!!! in food!!!! baby
food!!!  So please be careful if you buy this or are considering
it.<br /><br />My daughter LOVES this food-- it's actually her
favorite.  This is the first time we have noticed plastic in it in
over 2 months."
```

So we can understand here why the model did not recognize it was a negative sentence as there are bits of this sentence saying "My daughter loves this food", "its her favorite". So the model is getting confused and see this as a more positive sentence when its not.

```
results_df.query('Score == 1').sort_values('vader_pos',
ascending=False)['Text'].values[0]
```

```
'So we cancelled the order.  It was cancelled without any problem.
That is a positive note...'
```

In the sentence we actualy use the word "positive" and because the sentence is sarcastic, the vader model doesnt recognize it as it is only looking at the score of each word individually.

```
results_df.query('Score == 5').sort_values('roberta_neg',
ascending=False)['Text'].values[0]
```

```
'this was sooooo deliscious but too bad i ate em too fast and gained 2
pds! my fault'
```

So it is sort of a negative sentiment but it is a positive review

```
results_df.query('Score == 5').sort_values('vader_neg',
ascending=False)['Text'].values[0]
```

```
'this was sooooo deliscious but too bad i ate em too fast and gained 2
pds! my fault'
```

It happens to be the same sentence so both model seemed to have been confused with this review but maybe it is actualy a negative sentiment for a positive review so maybe it did a better job than we would expect to see.

The transformers pipeline

The extra piece than we can look at is to use the hugging face transformers pipeline which makes everything a lot simpler. We import from the transformers library pipeline and we can make a pipeline which will automaticaly set up a default model for sentiment analysis and we

can run it with just a few code lines without having to set anything up. Then we can just run text on it.

```
from transformers import pipeline

sent_pipeline = pipeline("sentiment-analysis")
```

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

```
sent_pipeline('I love this product!')
```

```
[{'label': 'POSITIVE', 'score': 0.9998855590820312}]
```

```
sent_pipeline('boooo')
```

```
[{'label': 'NEGATIVE', 'score': 0.7565664649009705}]
```

```
sent_pipeline(example)
```

```
[{'label': 'NEGATIVE', 'score': 0.9994776844978333}]
```

Step 5 : Train a classical ML model (SVM)

Now that we have been working with pretrained data about sentiment analysis, we can try training our own model using SVM.

Load and Clean the data

```
# Convert star ratings to sentiment labels
def label_sentiment(score):
    if score >= 4:
        return "positive"
    elif score == 3:
        return "neutral"
    else:
        return "negative"

df = pd.read_csv('Reviews.csv')
df = df.head(1000)
df = df[['Text', 'Score']].dropna() # Drop rows with missing values
df['Sentiment'] = df['Score'].apply(label_sentiment)
df = df[['Text', 'Sentiment']]  # Keep only needed columns
df.head()
```

```
                                                Text Sentiment
0  I have bought several of the Vitality canned d...  positive
1  Product arrived labeled as Jumbo Salted Peanut...  negative
2  This is a confection that has been around a fe...  positive
3  If you are looking for the secret ingredient i...  negative
4  Great taffy at a great price.  There was a wid...  positive
```

Preprocess the code

We will do the following : lowercase the text, remove punctuation and numbers, tokenize, remove stop words and lemmatize. We will use NLTK for stop word removal and spaCy for lemmatization.

```
nlp = spacy.load('en_core_web_sm')
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    # Lowercase and remove non-alphabetic characters
    text = text.lower()
    text = re.sub(r'[^a-z\s]', '', text)

    # Tokenize and remove stop words
    doc = nlp(text)
    tokens = [token.lemma_ for token in doc if token.text not in
stop_words and token.is_alpha]

    return ' '.join(tokens)

df['Clean_Text'] = df['Text'].apply(preprocess_text)
df[['Text', 'Clean_Text', 'Sentiment']].head()
```

```
                                                Text  \
0  I have bought several of the Vitality canned d...
1  Product arrived labeled as Jumbo Salted Peanut...
2  This is a confection that has been around a fe...
3  If you are looking for the secret ingredient i...
4  Great taffy at a great price.  There was a wid...

                                          Clean_Text Sentiment
0  buy several vitality can dog food product find...  positive
1  product arrived label jumbo salt peanutsthe pe...  negative
2  confection around century light pillowy citrus...  positive
3  look secret ingredient robitussin believe find...  negative
4  great taffy great price wide assortment yummy ...  positive
```

Convert text into TF-IDF Vectors

```
tfidf = TfidfVectorizer(max_features=5000)
X = tfidf.fit_transform(df['Clean_Text'])
y = df['Sentiment']
```

Train/Test Split

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

Train the SVM model

```
svm = LinearSVC()
svm.fit(X_train, y_train)
```

```
LinearSVC()
```

Evaluation of the model

```
y_pred = svm.predict(X_test)

print("Classification Report:")
print(classification_report(y_test, y_pred))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

    negative       0.82      0.31      0.45        29
     neutral       0.00      0.00      0.00        15
    positive       0.82      0.99      0.90       156

    accuracy                           0.82       200
   macro avg       0.55      0.43      0.45       200
weighted avg       0.76      0.82      0.77       200

Confusion Matrix:
[[  9   0  20]
 [  1   0  14]
 [  1   0 155]]

/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
```

```
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/
site-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

We can see that we have a high accuracy overall (82%). That sounds good at first glance but it is mostly because the model is doing well on the "positive" class, which is the majority class. Our model classifies almost everything as positive, even when its actualy neutral or negative. Indeed, precision and recall for positive are very high meaning the model got most of them right. However, negative and neutral performances are very poor. The neutral F1 score is 0.00 meaning the model isnt predicting this class at all. Negative F1 score is 0.45 but recall is only 31% so we are missing most negatives. Those inconsistencies are likely caused because of class imbalance. As we saw in our dataset analysis, tha data is dominated by positive reviews. From the confusion matrix: Actual: Negative → Predicted Positive: 20 Actual: Neutral → Predicted Positive: 14 So the model rarely predicts anything but positives.

What we can do to try and get better results is to use the class weights in SVM

```
svm_balanced = LinearSVC(class_weight='balanced')
svm_balanced.fit(X_train, y_train)
y_pred_balanced = svm_balanced.predict(X_test)

print("Classification Report (Balanced):")
print(classification_report(y_test, y_pred_balanced))

print("Confusion Matrix (Balanced):")
print(confusion_matrix(y_test, y_pred_balanced))
```

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

    negative       0.69      0.31      0.43        29
     neutral       0.00      0.00      0.00        15
    positive       0.82      0.97      0.89       156

    accuracy                           0.80       200
   macro avg       0.50      0.43      0.44       200
weighted avg       0.74      0.80      0.75       200

Confusion Matrix (Balanced):
[[  9   0  20]
 [  1   0  14]
 [  3   2 151]]
```

We notice again that even if the precision for negative has improved, overall the neutral sentiment is still ignored and accuracy is skewed by the dominant positive class.

Here we could think of using a Resampling method like SMOTE but for raw text data it will probably not be working.

The pretrained models (especially RoBERTa) produced highly consistent and contextually accurate results, significantly reducing the effort required for data preprocessing and model tuning. VADER, while lightweight and fast, offered surprisingly effective results for binary sentiment but struggled with neutral or nuanced cases.

On the other hand, training models from scratch required careful handling of issues like class imbalance, feature engineering, and model selection. Despite achieving high accuracy, these models were less robust to complex or subtle linguistic patterns in the text compared to the transformer-based approach.