Instructions: To complete the following task using Python, please download an Integrated Development Environment (IDE) of your choice. Ensure that your solution includes both the written code (input) and its corresponding output. Once completed, upload your solution in PDF format or any other format you prefer. The questions are worth 50 points each.

Question 1: Financial Sentiment Analysis Background: The utilization of statistical techniques in financial sentiment analysis is often limited due to complexities in practical applications and a shortage of high-quality training data. In finance and economics texts, where annotated datasets are scarce and largely proprietary, this challenge becomes more pronounced. To address this, a collection of approximately 5000 sentences has been compiled to establish benchmarks for various modeling techniques. Dataset: The original Financial Phrase Bank data contains 4840 sentences, annotated by 16 individuals with robust backgrounds in financial markets, including researchers from Aalto University School of Business. However, this assignment focuses on approximately 47% of the dataset, which corresponds to sentences with 100% agreement among the annotators. The analysis will center on sentences with unanimous annotator agreement, labeled as 'positive', 'negative', and 'neutral'.

All of the imports

In [ ]:
```
import pandas as pd
import numpy as np
import datetime as dt
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC # "Support vector classifier"
```

1. Data Preparation: • Load and convert the text file into a DataFrame with columns: 'Text' and 'Sentiment'.

In [ ]:
```
pd.options.display.float_format = '{:,.2f}'.format
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 2000)

HW6F1 = pd.read_csv('./Q1_FinancialDataset_Sentences_AllAgree.txt',sep='.@',names=[
HW6F1
```

Out[ ]:

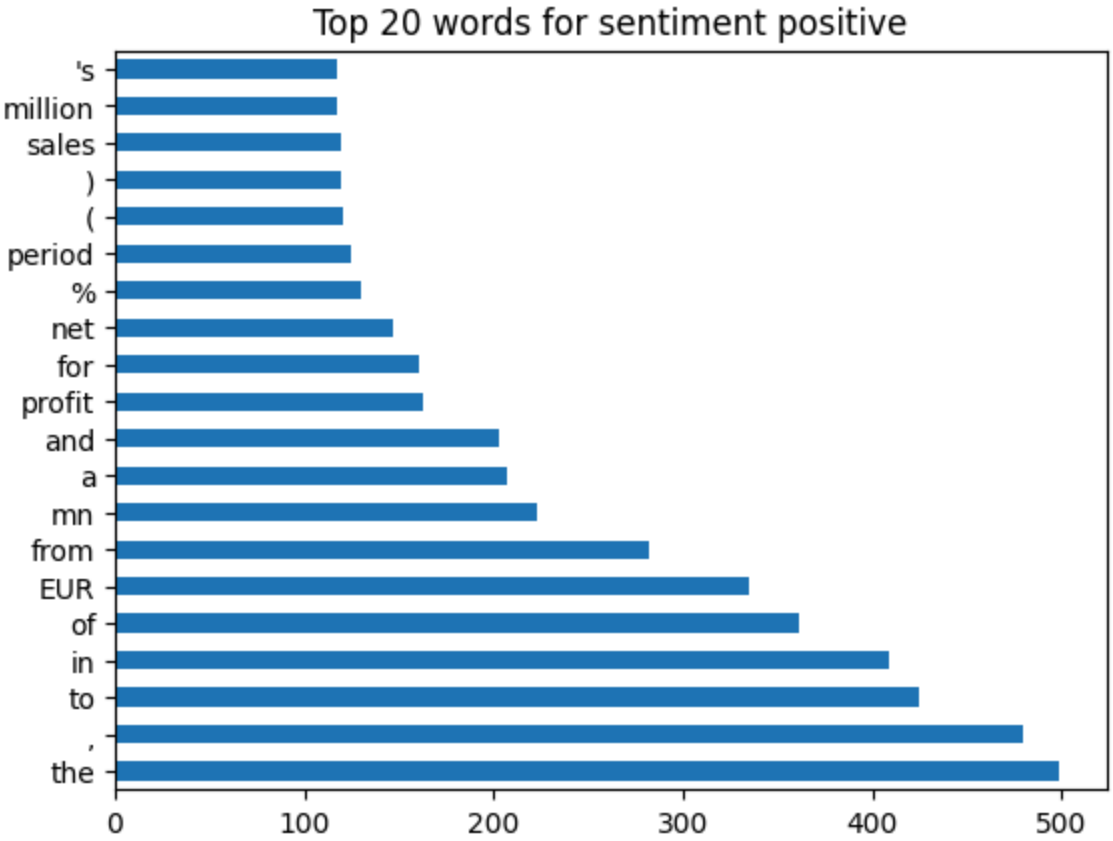
	Text	Sentiment
0	According to Gran , the company has no plans t...	neutral
1	For the last quarter of 2010 , Componenta 's n...	positive
2	In the third quarter of 2010 , net sales incre...	positive
3	Operating profit rose to EUR 13.1 mn from EUR ...	positive
4	Operating profit totalled EUR 21.1 mn , up fro...	positive
...	...	...
2259	Operating result for the 12-month period decre...	negative
2260	HELSINKI Thomson Financial - Shares in Cargote...	negative
2261	LONDON MarketWatch -- Share prices ended lower...	negative
2262	Operating profit fell to EUR 35.4 mn from EUR ...	negative
2263	Sales in Finland decreased by 10.5 % in Januar...	negative

2264 rows × 2 columns

Exploratory Data Analysis (EDA): • Perform EDA and plot bar charts for the frequency of the top 20 words in each sentiment category.

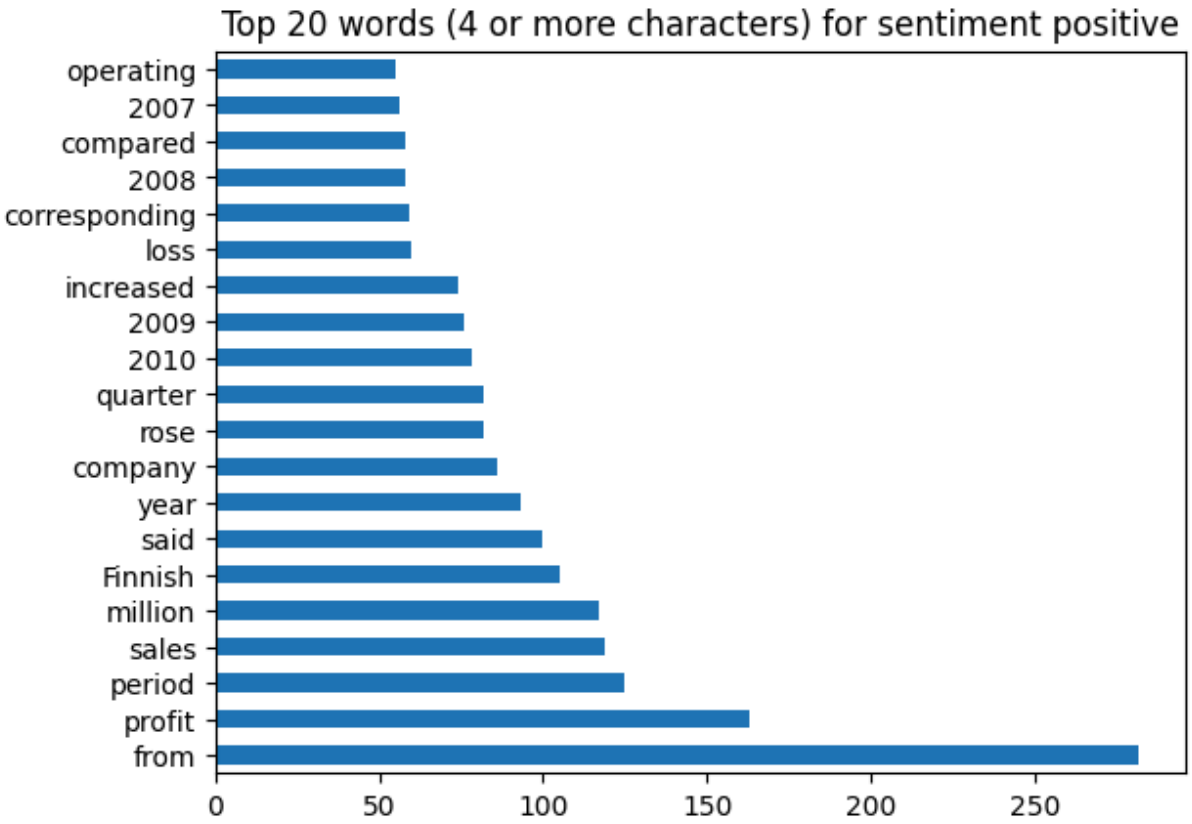
In [ ]:
```
sel = 'positive'
plt.title(f"Top 20 words for sentiment {sel}")
HW6F1.loc[HW6F1['Sentiment'] == sel]['Text'].str.split(expand=True).stack().value_c
# Too many small words and punctuation. I wil get rid of words that are three chara
# to do this in one line.
```

Out[ ]:
```
<Axes: title={'center': 'Top 20 words for sentiment positive'}>
```



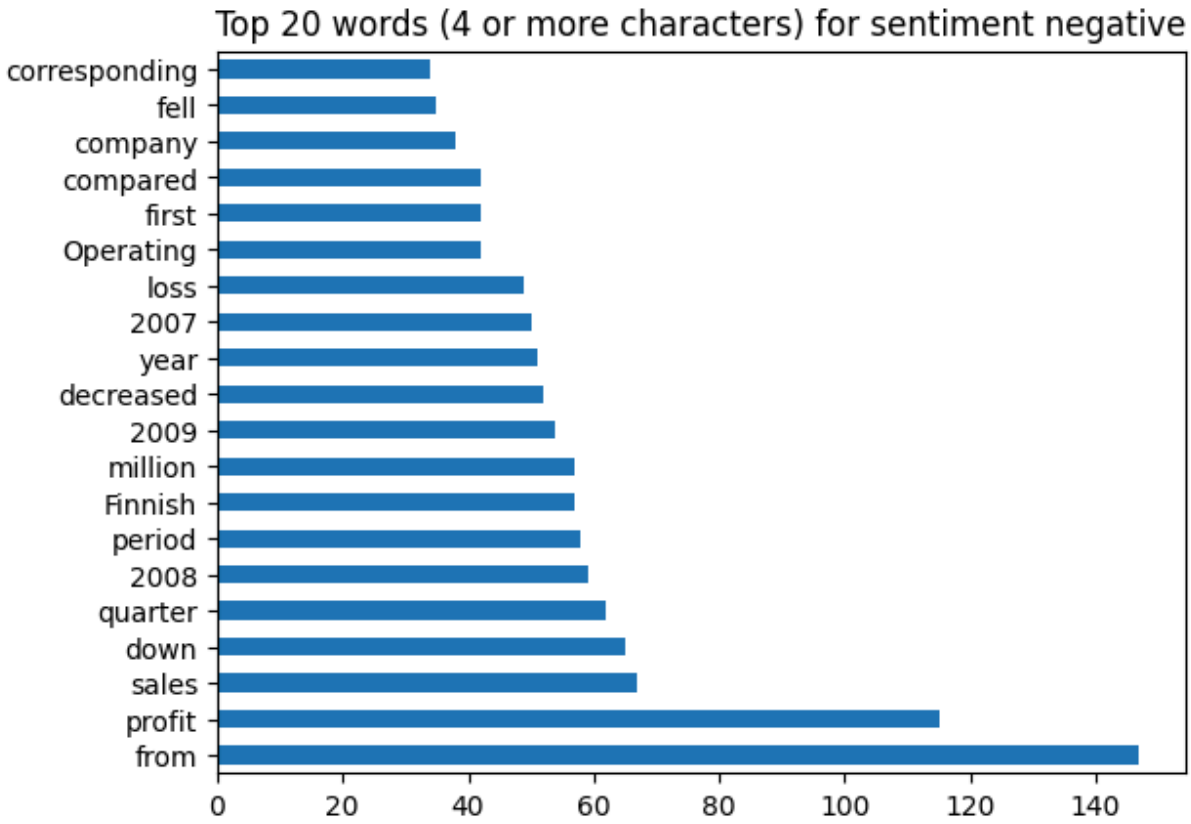
In [ ]:
```
sel = 'positive'
HW6F1Words = HW6F1.loc[HW6F1['Sentiment'] == sel]['Text'].str.split(expand=True).st
plt.title(f"Top 20 words (4 or more characters) for sentiment {sel}")
HW6F1Words[HW6F1Words.index.str.get(3).notnull()][:20].plot(kind='barh')
# This is much better. I will use this approach. I could remove the years, but that
```

Out[ ]:
```
<Axes: title={'center': 'Top 20 words (4 or more characters) for sentiment positiv
e'}>
```



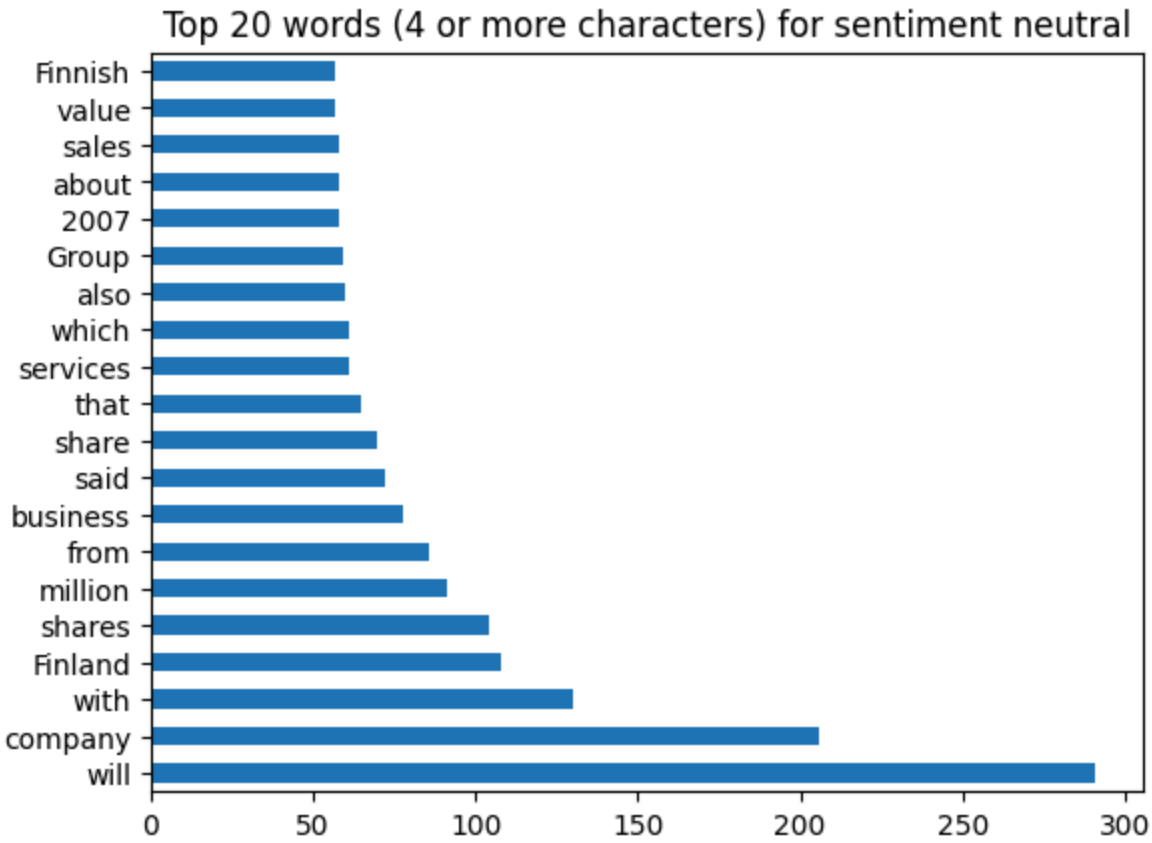
In [ ]:
```
sel = 'negative'
HW6F1Words = HW6F1.loc[HW6F1['Sentiment'] == sel]['Text'].str.split(expand=True).st
plt.title(f"Top 20 words (4 or more characters) for sentiment {sel}")
HW6F1Words[HW6F1Words.index.str.get(3).notnull()][:20].plot(kind='barh')
```

Out[ ]:
```
<Axes: title={'center': 'Top 20 words (4 or more characters) for sentiment negativ
e'}>
```



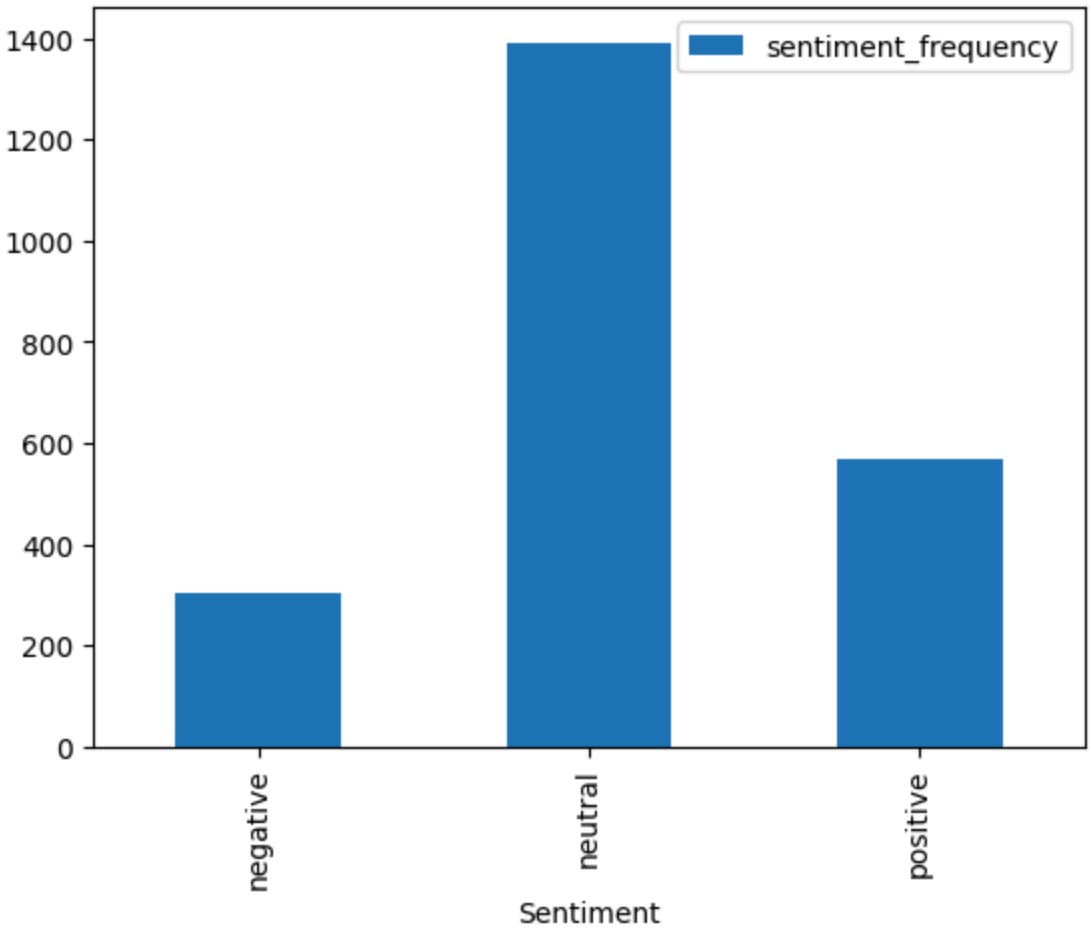
In [ ]:
```
sel = 'neutral'
HW6F1Words = HW6F1.loc[HW6F1['Sentiment'] == sel]['Text'].str.split(expand=True).st
plt.title(f"Top 20 words (4 or more characters) for sentiment {sel}")
HW6F1Words[HW6F1Words.index.str.get(3).notnull()][:20].plot(kind='barh')
```

Out[ ]:
```
<Axes: title={'center': 'Top 20 words (4 or more characters) for sentiment neutra
l'}>
```



Class Imbalance Analysis: • Compute and visualize the frequency of each sentiment label with a bar graph. Discuss class imbalance.

In [ ]:
```
HW6F1Group = HW6F1.groupby("Sentiment").agg(
    sentiment_frequency=("Text", "count")
).plot(kind='bar')
```

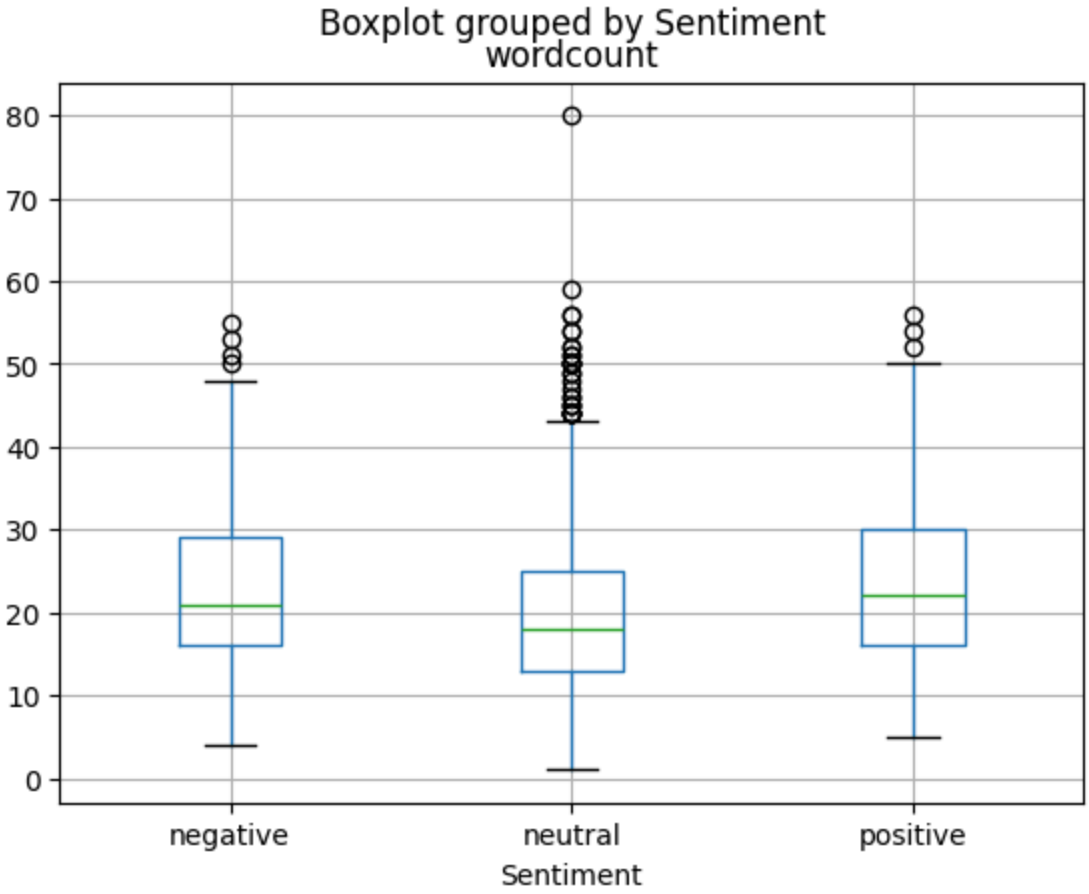


I would have expected the sentiment of 'negative' to be the most frequent. This is because people generally do not speak up unless they have a complaint. But it may have to do with the source. The 'neutral' sentiment may be the most common since it is not a customer service site.

Word Count Analysis: • Create box plots for word/token counts per sentiment label. Discuss discrepancies.

In [ ]:
```
HW6F1['wordcount'] = HW6F1['Text'].apply(lambda x: len(str(x).split()))
HW6F1.boxplot(by = 'Sentiment', column=['wordcount'])
```

Out[ ]:
```
<Axes: title={'center': 'wordcount'}, xlabel='Sentiment'>
```



My expectation would have been that negative would have the longest sentences followed by neutral and then positive. They are all about the same but neutral has far more outliers. All the outliers are in the direction of longer sentences. The neutral sentences must be considering more topics and that is what makes them longer. Maybe the reviewers did not have time to fully read and understand long sentences, so they defaulted to marking them as neutral.

Data Splitting: • Split data into training (80%) and testing (20%) sets using stratified splitting with a random seed of 64.

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(HW6F1.drop('Sentiment', axis=1)

print('X_train shape:', X_train.shape)
print('y_train shape:', y_train.shape)
print('X_test shape:', X_test.shape)
print('y_test shape:', y_test.shape)
```

```
X_train shape: (1811, 2)
y_train shape: (1811,)
X_test shape: (453, 2)
y_test shape: (453,)
```

Model Development and Evaluation: • Vectorization: Use CountVectorizer and Tf-Idf.

In [ ]:
```
# Vectorization: Use CountVectorizer and Tf-Idf.

vec = CountVectorizer()
X = vec.fit_transform(X_train['Text'])
pd.DataFrame(X.toarray(), columns=vec.get_feature_names_out())
vec = TfidfVectorizer()
X = vec.fit_transform(X_train['Text'])
pd.DataFrame(X.toarray(), columns=vec.get_feature_names_out())
```

Out[ ]:

	00	000	002	0025	003	0030	008	01	0101138	012	02	023	027	03
0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1	0.19	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1806	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1807	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1808	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1809	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1810	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

1811 rows × 5328 columns

Model Development and Evaluation: • Model Training: Employ Naive Bayes

In [ ]:
```
# Model Training: Employ Naive Bayes


model = make_pipeline(TfidfVectorizer(), MultinomialNB())
model.fit(X_train['Text'], y_train)
labels = model.predict(X_test['Text'])
```

Model Development and Evaluation: • Metrics Tracking: Present Accuracy, Precision, Recall, F1-Score for Naive Bayes

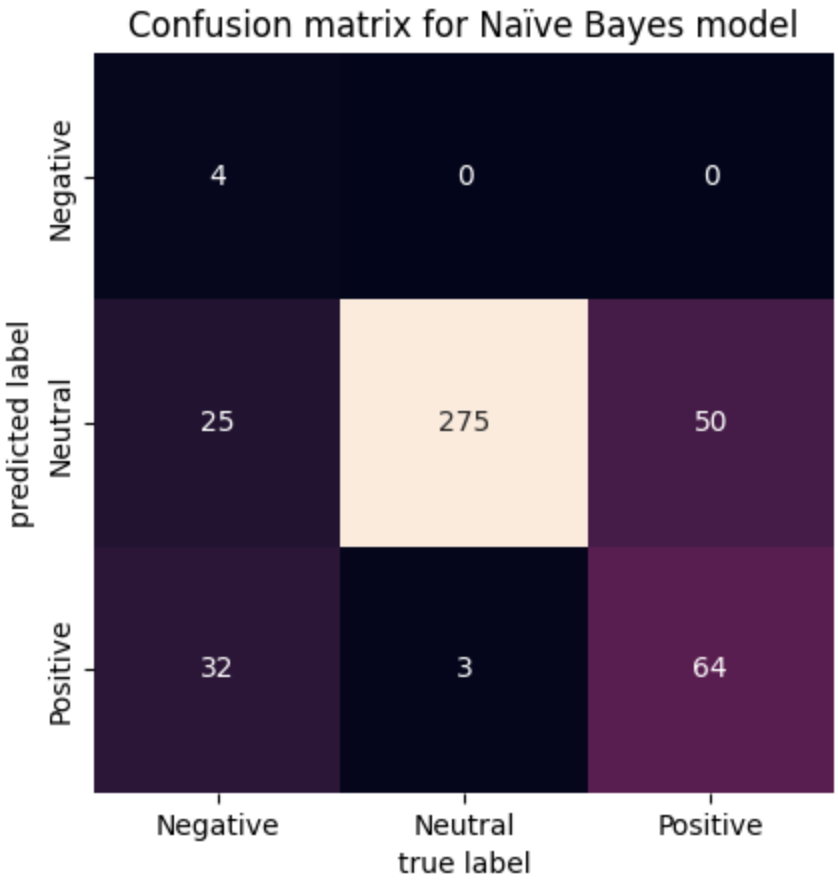
In [ ]:
```
# Metrics Tracking: Present Accuracy, Precision, Recall, F1-Score for each class
acc = accuracy_score(y_test, labels)
prec = precision_score(y_test, labels, average='macro')
recall = recall_score(y_test, labels, average='macro')
f1score = f1_score(y_test, labels, average='macro')

print(f"For the Naïve Bayes, the accuracy is {acc} the precision is {prec} the reca
```

```
For the Naïve Bayes, the accuracy is 0.7571743929359823 the precision is 0.810726310
7263107 the recall is 0.5387286374524194 and th F1 Score is 0.5332706895190075.
```

Model Development and Evaluation: • an overall Confusion Matrix

In [ ]:
```
# An overall Confusion Matrix.
classes = ['Negative', 'Neutral', 'Positive']
title = 'Confusion matrix for Naïve Bayes model'
mat = confusion_matrix(y_test, labels)
#sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False, xticklabels=train
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False, xticklabels=classe
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.title(title)
```

Out[ ]:
```
Text(0.5, 1.0, 'Confusion matrix for Naïve Bayes model')
```



Model Development and Evaluation: • Analysis: Discuss the effectiveness of each model based on the tracked metrics.

Looking at the Accuracy, Precision, Recall, and F1 Score, this model is good but not great. I would prefer to see scores in the 90s. These are all far lower. An F1 Score of 0.53 is particularly concerning. Moving to the Confusion Matrix, this is born out again. While it seems to get the Neutral right a lot, it is very poor at getting the Negative right.

Based on this, I would search for a better model.

Model Development and Evaluation: • Model Training - Bonus Question (5pts): Employ Random Forest

In [ ]:
```
model = make_pipeline(TfidfVectorizer(), RandomForestClassifier(n_estimators=100, r
model.fit(X_train['Text'], y_train)
labels = model.predict(X_test['Text'])
```

Model Development and Evaluation: • Metrics Tracking: Present Accuracy, Precision, Recall, F1-Score for Random Forest

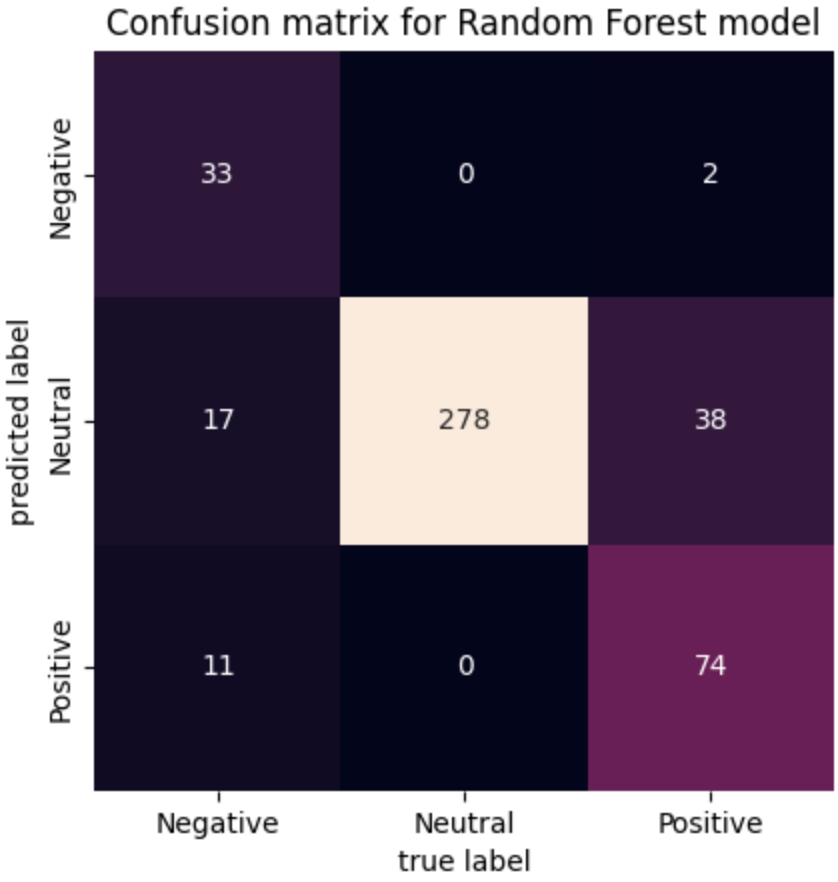
In [ ]:
```
acc = accuracy_score(y_test, labels)
prec = precision_score(y_test, labels, average='macro')
recall = recall_score(y_test, labels, average='macro')
f1score = f1_score(y_test, labels, average='macro')

print(f"For the Random Forest, the accuracy is {acc} the precision is {prec} the re
```

```
For the Random Forest, the accuracy is 0.8498896247240618 the precision is 0.8827600
70995365 the recall is 0.7300354711916404 and th F1 Score is 0.780400742117571.
```

Model Development and Evaluation: • Metrics Tracking: Present an overall Confusion Matrix.

In [ ]:
```
classes = ['Negative', 'Neutral', 'Positive']
title = 'Confusion matrix for Random Forest model'
mat = confusion_matrix(y_test, labels)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False, xticklabels=classe
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.title(title)
```

Out[ ]:
```
Text(0.5, 1.0, 'Confusion matrix for Random Forest model')
```



Model Development and Evaluation: • Analysis: Discuss the effectiveness of each model based on the tracked metrics.

This model seems t be better than the Naive Bayes. Accuracy, Precisions, Recall, and the F1 Score are all much higher. They are all in the 80s or 70s. An F1 score of 0.78 is better but could still be higher. Looking at the Confusion Matrix, this model did much better on the Negative comments. It was perfect on the Neutral comments.

This is certainly better than the Naive Bayes model but I would hope that we can do better.

Model Development and Evaluation: • Model Training - Bonus Question (5pts): Employ Support Vector Machines.

In [ ]:
```
model = make_pipeline(TfidfVectorizer(), SVC(kernel='linear', C=1E10))
model.fit(X_train['Text'], y_train)
labels = model.predict(X_test['Text'])
```

Model Development and Evaluation: • Metrics Tracking: Present Accuracy, Precision, Recall, F1-Score for each class

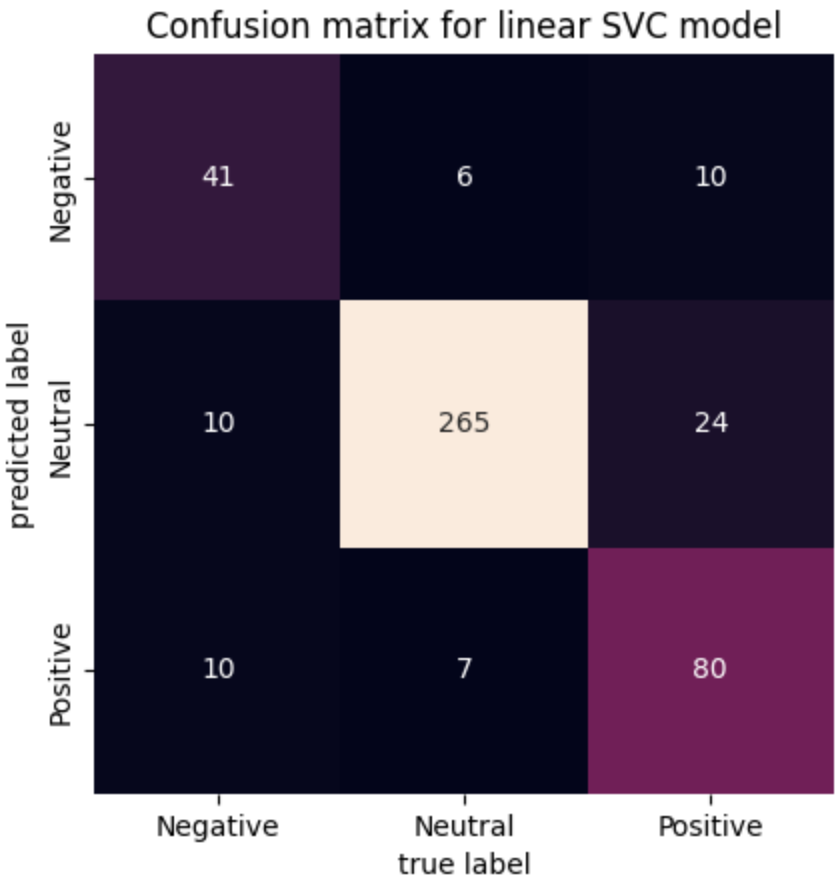
In [ ]:
```
acc = accuracy_score(y_test, labels)
prec = precision_score(y_test, labels, average='macro')
recall = recall_score(y_test, labels, average='macro')
f1score = f1_score(y_test, labels, average='macro')
```

```
print(f"For the linear Support Vector Machines, the accuracy is {acc} the precision
```

```
For the linear Support Vector Machines, the accuracy is 0.8520971302428256 the preci
sion is 0.8101093796911109 the recall is 0.7757076478592794 and th F1 Score is 0.790
5844290690999.
```

In [ ]:
```
classes = ['Negative', 'Neutral', 'Positive']
title = 'Confusion matrix for linear SVC model'
mat = confusion_matrix(y_test, labels)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False, xticklabels=classe
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.title(title)
```

Out[ ]:
```
Text(0.5, 1.0, 'Confusion matrix for linear SVC model')
```



Model Development and Evaluation: • Analysis: Discuss the effectiveness of each model based on the tracked metrics.

I was hoping that the SVC model would provide a much better outcome than the Random Forest Model. But, looking at the measures and the Confusions Matrix,it is roughly the same. The F1 Score is 0.79 which is only slightly better than the F1 Score for the Random Forest. I also note that this model did far worse on the Neutral predictions. I would be hard pressed to select this model over the Random Forest. So, the search continues.

Model Development and Evaluation: • Model Training - Bonus Question (5pts): Employ Support Vector Machines.

In [ ]:
```
model = make_pipeline(TfidfVectorizer(), SVC(kernel='rbf', C=1E6))
model.fit(X_train['Text'], y_train)
labels = model.predict(X_test['Text'])
```

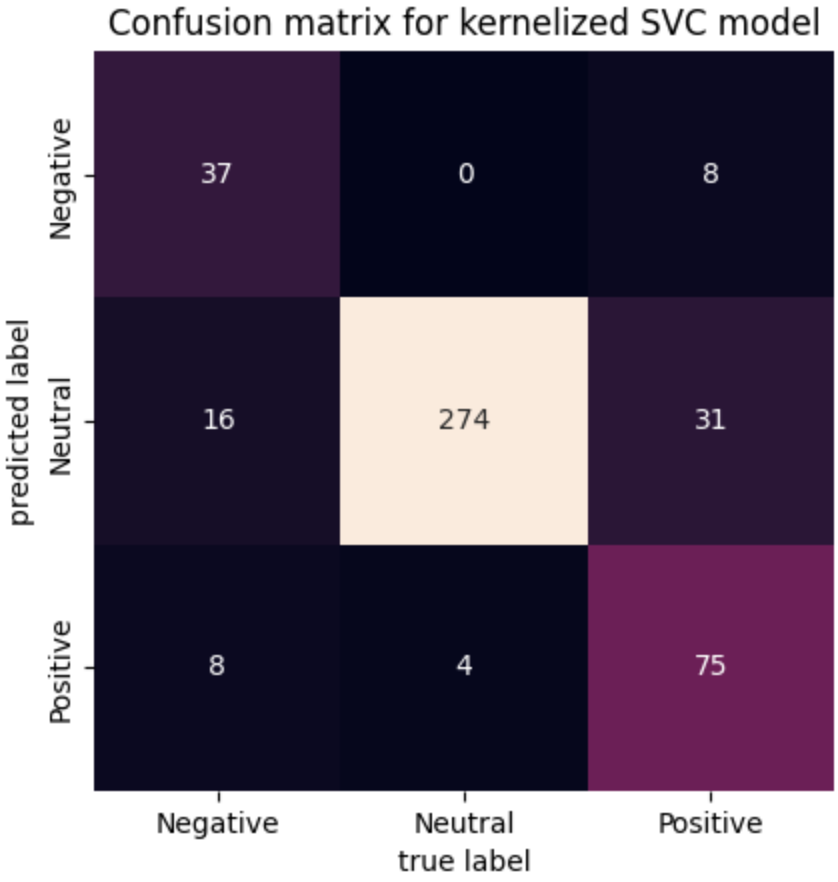
Model Development and Evaluation: • Metrics Tracking: Present Accuracy, Precision, Recall, F1-Score for each class

In [ ]:
```
acc = accuracy_score(y_test, labels)
prec = precision_score(y_test, labels, average='macro')
recall = recall_score(y_test, labels, average='macro')
f1score = f1_score(y_test, labels, average='macro')

print(f"For the kernelized Support Vector Machines, the accuracy is {acc} the preci
```

```
For the kernelized Support Vector Machines, the accuracy is 0.8520971302428256 the p
recision is 0.8459579140855326 the recall is 0.7500212082275509 and th F1 Score is
0.7864133203638781.
```

In [ ]:
```
classes = ['Negative', 'Neutral', 'Positive']
title = 'Confusion matrix for kernelized SVC model'
mat = confusion_matrix(y_test, labels)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False, xticklabels=classe
plt.xlabel('true label')
plt.ylabel('predicted label')
plt.title(title)
```

Out[ ]:
```
Text(0.5, 1.0, 'Confusion matrix for kernelized SVC model')
```



Model Development and Evaluation: • Analysis: Discuss the effectiveness of each model based on the tracked metrics.

The next and last model is the kernalized version of the SVC model. I had some hope that this would be significantly better. It may be that I eed to tune the Hyperparameters. But with the initial set I selected, it is roughly similar to the linear SVC model. The F1 score is slightly lower, not a significant difference. The Confusion Matrix is similar. Although this did do better on the Neutral comments.

All in all, based on these models, I would stick with the Rand Forest Model for now.