Natural Language Processing Project

Welcome to the NLP Project for this section of the course. In this NLP project you will be attempting to classify Yelp Reviews into 1 star or 5 star categories based off the text content in the reviews. This will be a simpler procedure than the lecture, since we will utilize the pipeline methods for more complex tasks.

We will use the Yelp Review Data Set from Kaggle**.**

Each observation in this dataset is a review of a particular business by a particular user.

The "stars" column is the number of stars (1 through 5) assigned by the reviewer to the business. (Higher stars is better.) In other words, it is the rating of the business by the person who wrote the review.

The "cool" column is the number of "cool" votes this review received from other Yelp users.

All reviews start with 0 "cool" votes, and there is no limit to how many "cool" votes a review can receive. In other words, it is a rating of the review itself, not a rating of the business.

The "useful" and "funny" columns are similar to the "cool" column.

Let's get started! Just follow the directions below!

Imports

**Import the usual suspects. :) **

In [1]:
```
import numpy as np
import pandas as pd
```

The Data

Read the yelp.csv file and set it as a dataframe called yelp.

In [2]:
```
yelp = pd.read_csv('yelp.csv')
```

** Check the head, info , and describe methods on yelp.**

In [3]:
```
yelp.head()
```

Out[3]:

	business_id	date	review_id	stars	text	type
0	9yKzy9PApeiPPOUJEtnvkg	2011-01-26	fWKvX83p0-ka4JS3dc6E5A	5	My wife took me here on my birthday for breakf...	review
1	ZRJwVLyzEJq1VAihDhYiow	2011-07-27	IjZ33sJrzXqU-0X6U8NwyA	5	I have no idea why some people give bad review...	review
2	6oRAC4uyJCsJl1X0WZpVSA	2012-06-14	IESLBzqUCLdSzSqm0eCSxQ	4	love the gyro plate. Rice is so good and I als...	review
3	_1QQZuf4zZOyFCvXc0o6Vg	2010-05-27	G-WvGaISbqqaMHlNnByodA	5	Rosie, Dakota, and I LOVE Chaparral Dog Park!!...	review
4	6ozycU1RpktNG2-1BroVtw	2012-01-05	1uJFq2r5QfJG_6ExMRCaGw	5	General Manager Scott Petello is a good egg!!!...	review

In [4]:
```
yelp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   business_id  10000 non-null  object
 1   date         10000 non-null  object
 2   review_id    10000 non-null  object
 3   stars        10000 non-null  int64 
 4   text         10000 non-null  object
 5   type         10000 non-null  object
 6   user_id      10000 non-null  object
 7   cool         10000 non-null  int64 
 8   useful       10000 non-null  int64 
 9   funny        10000 non-null  int64 
dtypes: int64(4), object(6)
memory usage: 781.4+ KB
```

In [5]:
```
yelp.describe()
```

Out[5]:

	stars	cool	useful	funny
count	10000.000000	10000.000000	10000.000000	10000.000000
mean	3.777500	0.876800	1.409300	0.701300
std	1.214636	2.067861	2.336647	1.907942
min	1.000000	0.000000	0.000000	0.000000
25%	3.000000	0.000000	0.000000	0.000000
50%	4.000000	0.000000	1.000000	0.000000
75%	5.000000	1.000000	2.000000	1.000000
max	5.000000	77.000000	76.000000	57.000000

Create a new column called "text length" which is the number of words in the text column.

In [6]:
```
yelp['text length'] = yelp['text'].apply(len)
```

EDA

Let's explore the data

Imports

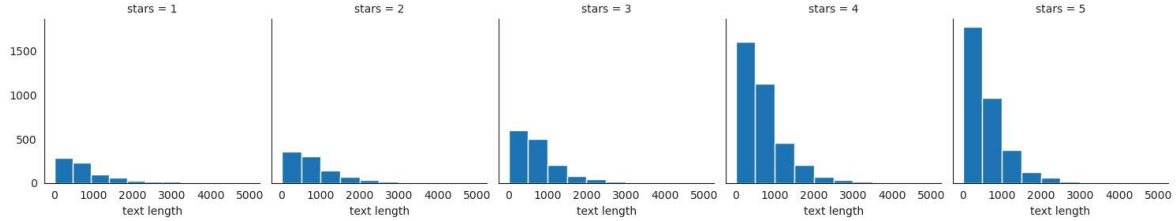
Import the data visualization libraries if you haven't done so already.

In [7]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('white')
%matplotlib inline
```

Use FacetGrid from the seaborn library to create a grid of 5 histograms of text length based off of the star ratings. Reference the seaborn documentation for hints on this

In [8]:
```
g = sns.FacetGrid(yelp,col='stars')
g.map(plt.hist,'text length')
```

Out[8]:
```
<seaborn.axisgrid.FacetGrid at 0x7480240dcd10>
```



Create a boxplot of text length for each star category.

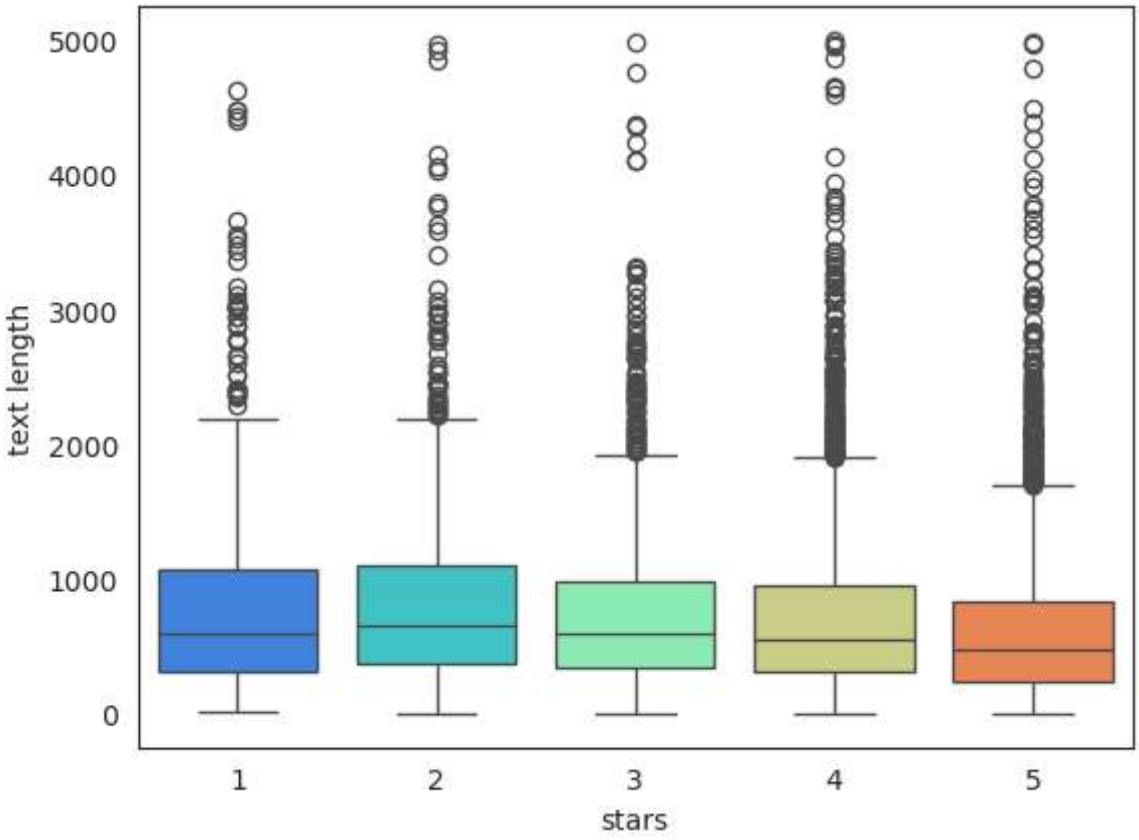
In [9]:
```
sns.boxplot(x='stars',y='text length',data=yelp,palette='rainbow')
```

```
/tmp/ipykernel_46016/468530455.py:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.boxplot(x='stars',y='text length',data=yelp,palette='rainbow')
```

Out[9]: <Axes: xlabel='stars', ylabel='text length'>



Create a countplot of the number of occurrences for each type of star rating.

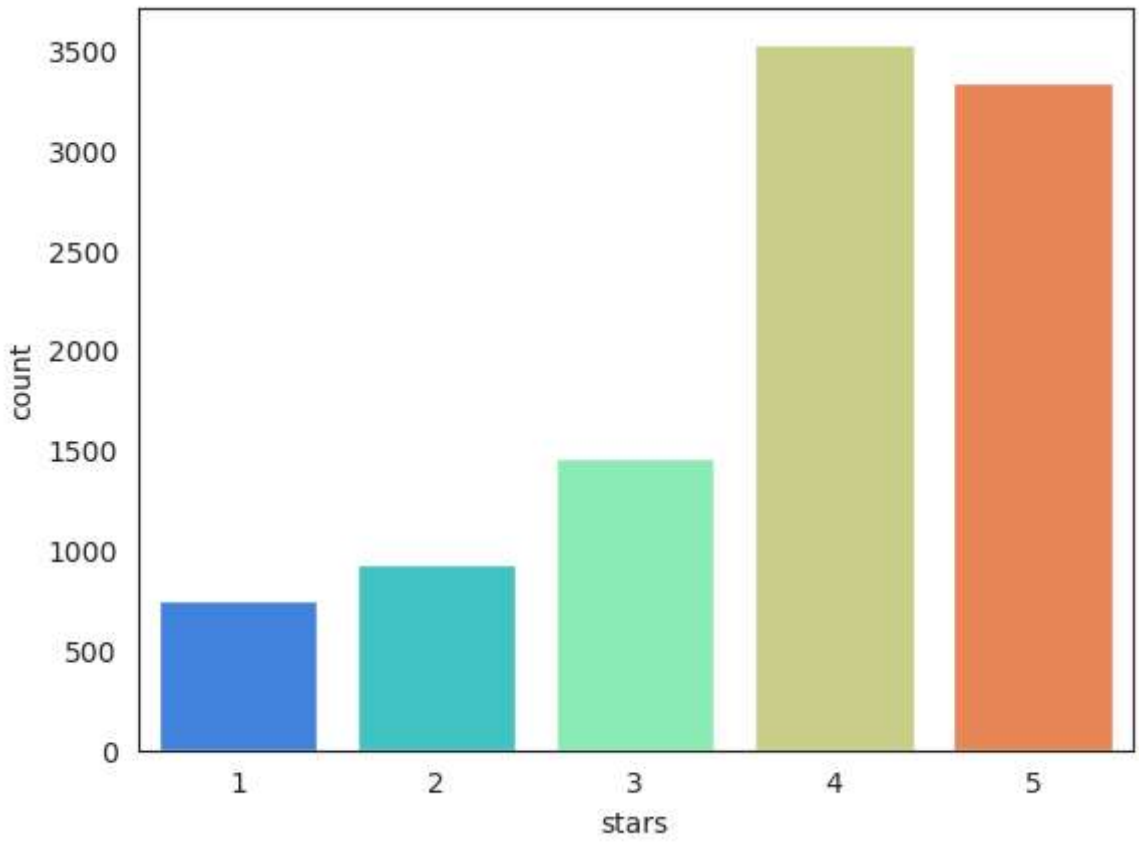
In [10]:
```
sns.countplot(x='stars',data=yelp,palette='rainbow')
```

```
/tmp/ipykernel_46016/2116447000.py:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.countplot(x='stars',data=yelp,palette='rainbow')
```

Out[10]: <Axes: xlabel='stars', ylabel='count'>



** Use groupby to get the mean values of the numerical columns, you should be able to create this dataframe with the operation:**

In [17]:
```
stars = yelp.groupby('stars').mean(numeric_only=True)
```

Use the corr() method on that groupby dataframe to produce this dataframe:

In [18]:
```
stars.corr()
```

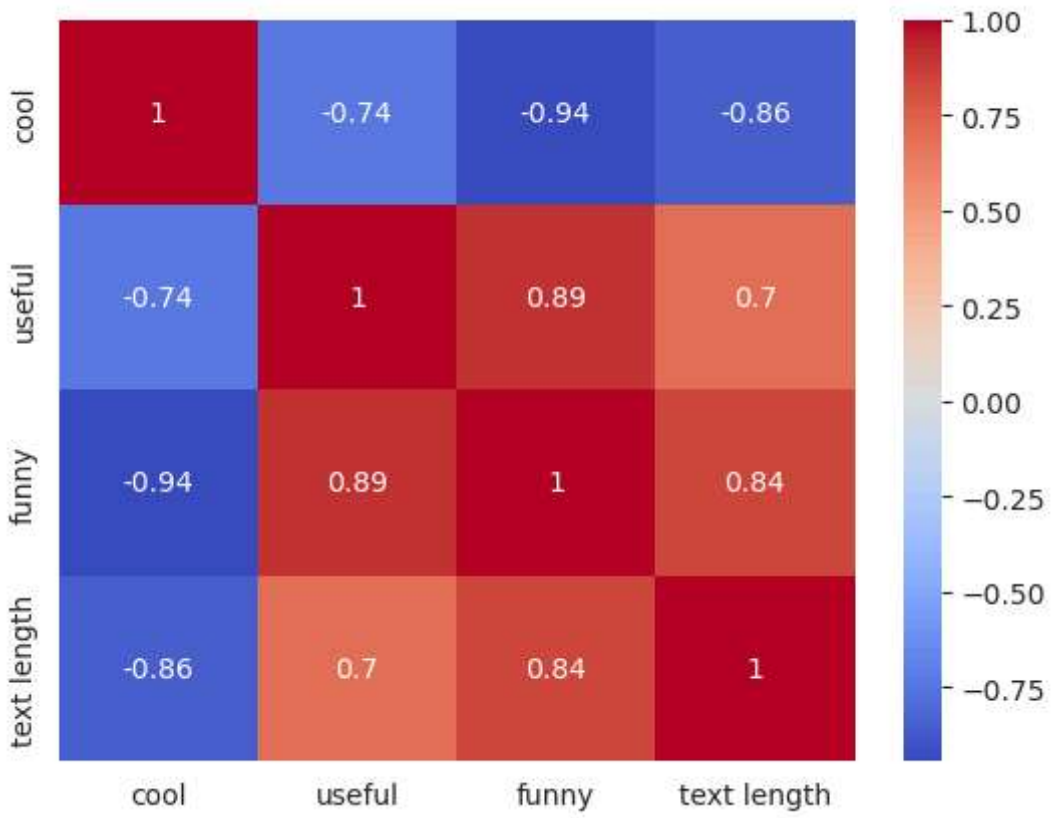
Out[18]:

	cool	useful	funny	text length
cool	1.000000	-0.743329	-0.944939	-0.857664
useful	-0.743329	1.000000	0.894506	0.699881
funny	-0.944939	0.894506	1.000000	0.843461
text length	-0.857664	0.699881	0.843461	1.000000

Then use seaborn to create a heatmap based off that .corr() dataframe:

In [19]:
```
sns.heatmap(stars.corr(),cmap='coolwarm',annot=True)
```

Out[19]:
```
<Axes: >
```



NLP Classification Task

Let's move on to the actual task. To make things a little easier, go ahead and only grab reviews that were either 1 star or 5 stars.

Create a dataframe called yelp_class that contains the columns of yelp dataframe but for only the 1 or 5 star reviews.

In [20]:
```
yelp_class = yelp[(yelp.stars==1) | (yelp.stars==5)]
```

** Create two objects X and y. X will be the 'text' column of yelp_class and y will be the 'stars' column of yelp_class. (Your features and target/labels)**

In [21]:
```
X = yelp_class['text'] #the messages
y = yelp_class['stars'] # the review
```

Import CountVectorizer and create a CountVectorizer object.

In [23]:
```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
```

** Use the fit_transform method on the CountVectorizer object and pass in X (the 'text' column). Save this result by overwriting X.**

In [24]:
```
X = cv.fit_transform(X) # overwrite X to transform the messages into vectorised
                        # so that python can work with it
```

Train Test Split

Let's split our data into training and testing data.

** Use train_test_split to split up the data into X_train, X_test, y_train, y_test. Use test_size=0.3 and random_state=101 **

In [25]:
```
from sklearn.model_selection import train_test_split
```

In [26]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.3,random_st
```

Training a Model

Time to train a model!

** Import MultinomialNB and create an instance of the estimator and call is nb **

In [27]:
```
from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB()
```

Now fit nb using the training data.

In [28]:
```
nb.fit(X_train,y_train)
```

Out[28]:
```
▾ MultinomialNB
▶ Parameters
```

Predictions and Evaluations

Time to see how our model did!

Use the predict method off of nb to predict labels from X_test.

In [29]:
```
predictions = nb.predict(X_test)
```

** Create a confusion matrix and classification report using these predictions and y_test **

In [30]:
```
from sklearn.metrics import confusion_matrix,classification_report
```

In [31]:
```
print(confusion_matrix(y_test,predictions))
print('\n')
print(classification_report(y_test,predictions))
```

```
[[159  69]
 [ 22 976]]


              precision    recall  f1-score   support

           1       0.88      0.70      0.78       228
           5       0.93      0.98      0.96       998

    accuracy                           0.93      1226
   macro avg       0.91      0.84      0.87      1226
weighted avg       0.92      0.93      0.92      1226
```

Great! Let's see what happens if we try to

include TF-IDF

to this process using a pipeline.

Using Text Processing

** Import TfidfTransformer from sklearn. **

In [32]:
```
from sklearn.feature_extraction.text import  TfidfTransformer
```

** Import Pipeline from sklearn. **

In [33]:
```
from sklearn.pipeline import Pipeline
```

** Now create a pipeline with the following steps:CountVectorizer(), TfidfTransformer(),MultinomialNB()**

In [34]:
```
pipeline = Pipeline([
    ('bow', CountVectorizer()),  # strings to token integer counts (turns messag
    ('tfidf', TfidfTransformer()),  # integer counts to weighted TF-IDF scores
    ('classifier', MultinomialNB()),  # train on TF-IDF vectors w/ Naive Bayes c
])
```

Using the Pipeline

Time to use the pipeline! Remember this pipeline has all your pre-process steps in it already, meaning we'll need to re-split the original data (Remember that we overwrote X as the CountVectorized version. What we need is just the text

Train Test Split

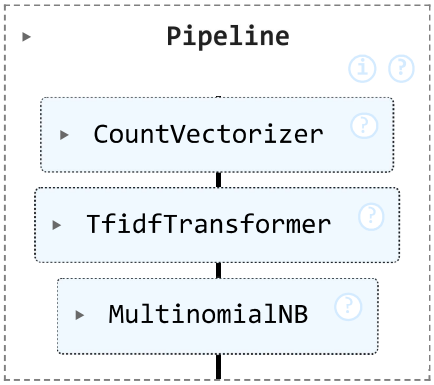
Redo the train test split on the yelp_class object.

In [35]:
```
X = yelp_class['text'] # We  need X as just the messages as the pipeline will tr
y = yelp_class['stars']
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.3,random_st
```

Now fit the pipeline to the training data. Remember you can't use the same training data as last time because that data has already been vectorized. We need to pass in just the text and labels

In [36]:
```
# May take some time
pipeline.fit(X_train,y_train)
```

Out[36]:



Predictions and Evaluation

** Now use the pipeline to predict from the X_test and create a classification report and confusion matrix. You should notice strange results.**

In [37]:
```
predictions = pipeline.predict(X_test)
```

In [38]:
```
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
```

```
[[  0 228]
 [  0 998]]
              precision    recall  f1-score   support

           1       0.00      0.00      0.00       228
           5       0.81      1.00      0.90       998

    accuracy                           0.81      1226
   macro avg       0.41      0.50      0.45      1226
weighted avg       0.66      0.81      0.73      1226
```

```
/home/eduart-cakaj/anaconda3/envs/Data_Science/lib/python3.12/site-packages/sklea
rn/metrics/_classification.py:1706: UndefinedMetricWarning: Precision is ill-defi
ned 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", result.shape[0])
/home/eduart-cakaj/anaconda3/envs/Data_Science/lib/python3.12/site-packages/sklea
rn/metrics/_classification.py:1706: UndefinedMetricWarning: Precision is ill-defi
ned 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", result.shape[0])
/home/eduart-cakaj/anaconda3/envs/Data_Science/lib/python3.12/site-packages/sklea
rn/metrics/_classification.py:1706: UndefinedMetricWarning: Precision is ill-defi
ned 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", result.shape[0])
```

Looks like Tf-Idf actually made things worse! That is it for this project. But there is still a lot more you can play with:

Some other things to try.... Try going back and playing around with the pipeline steps and seeing if creating a custom analyzer like we did in the lecture helps (note: it probably won't). Or recreate the pipeline with just the CountVectorizer() and NaiveBayes. Does changing the ML model at the end to another classifier help at all?

Now we will use a custom analyzer

In [47]:
```
import string
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer


cv = CountVectorizer(analyzer=text_process)  # custom analyzer
bow = cv.fit_transform(yelp_class['text'])



def text_process(msg):
    # Remove punctuation
    nopunc = [char for char in msg if char not in string.punctuation]
    nopunc = ''.join(nopunc)

    # Remove stopwords
    words = [word for word in nopunc.split() if word.lower() not in stopwords.wo

    return words
```

In [48]:
```
# Now we create pipeline to  include the customer analyzer function
pipeline = Pipeline([
    ('bow', CountVectorizer(analyzer=text_process)),  # strings to token integer
    ('tfidf', TfidfTransformer()),  # integer counts to weighted TF-IDF scores
    ('classifier', MultinomialNB()),  # train on TF-IDF vectors w/ Naive Bayes c
])
```

Now we train test split the model and use it to predict

In [49]:
```
X = yelp_class['text']
y = yelp_class['stars']
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.3,random_st
```

In [50]:
```
pipeline.fit(X_train, y_train)
```

Out[50]:
```
▸ Pipeline
  ▸ CountVectorizer
  ▸ TfidfTransformer
  ▸ MultinomialNB
```

In [54]:
```
predictions2 = pipeline.predict(X_test)
```

In [55]:
```
print(confusion_matrix(y_test,predictions2))
print(classification_report(y_test,predictions2))
```

```
[[  0 228]
 [  0 998]]
              precision    recall  f1-score   support

           1       0.00      0.00      0.00       228
           5       0.81      1.00      0.90       998

    accuracy                           0.81      1226
   macro avg       0.41      0.50      0.45      1226
weighted avg       0.66      0.81      0.73      1226
```

```
/home/eduart-cakaj/anaconda3/envs/Data_Science/lib/python3.12/site-packages/sklea
rn/metrics/_classification.py:1706: UndefinedMetricWarning: Precision is ill-defi
ned 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", result.shape[0])
/home/eduart-cakaj/anaconda3/envs/Data_Science/lib/python3.12/site-packages/sklea
rn/metrics/_classification.py:1706: UndefinedMetricWarning: Precision is ill-defi
ned 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", result.shape[0])
/home/eduart-cakaj/anaconda3/envs/Data_Science/lib/python3.12/site-packages/sklea
rn/metrics/_classification.py:1706: UndefinedMetricWarning: Precision is ill-defi
ned 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", result.shape[0])
```

Very little difference was made when we used a custom analyzer...

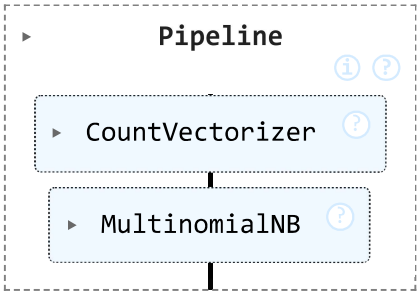
Now we will recreate the pipeline with just the CountVectorizer() and NaiveBayes

In [56]:
```
pipeline = Pipeline([
    ('bow', CountVectorizer(analyzer=text_process)),  # strings to token integer
    ('classifier', MultinomialNB()),  # train on TF-IDF vectors w/ Naive Bayes c
])
```

In [57]:
```
X = yelp_class['text']
y = yelp_class['stars']
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.3,random_st
```

In [58]:
```
pipeline.fit(X_train, y_train)
```

Out[58]:



In [59]:
```
predictions3 = pipeline.predict(X_test)
```

In [60]:
```
print(confusion_matrix(y_test,predictions3))
print(classification_report(y_test,predictions3))
```

```
[[143  85]
 [ 11 987]]
              precision    recall  f1-score   support

           1       0.93      0.63      0.75       228
           5       0.92      0.99      0.95       998

    accuracy                           0.92      1226
   macro avg       0.92      0.81      0.85      1226
weighted avg       0.92      0.92      0.92      1226
```

Based on the evaluation results, the best performing model was the pipeline using only Bag-of-Words features combined with Naive Bayes. This is likely because Raw counts tell Naive Bayes which words appear a lot in spam, and that's often the best clue. However, when using the TF-IDF transformer, it could have potentially made those frequent words less important, which would have confused the NaiveBayes model.