```
import pandas as pd
# Read in the csv file using pandas
df = pd.read_csv('federalist.csv')
# Convert the author column to categorical data
df['author'] = df.author.astype('category')
#  Display the first few rows
print(df.head())
# Display the counts by author
df['author'].value_counts()
```

```
         author                                               text
     0  HAMILTON  FEDERALIST. No. 1 General Introduction For the...
     1       JAY  FEDERALIST No. 2 Concerning Dangers from Forei...
     2       JAY  FEDERALIST No. 3 The Same Subject Continued (C...
     3       JAY  FEDERALIST No. 4 The Same Subject Continued (C...
     4       JAY  FEDERALIST No. 5 The Same Subject Continued (C...
     HAMILTON                49
     MADISON                 15
     HAMILTON OR MADISON     11
     JAY                      5
     HAMILTON AND MADISON     3
     Name: author, dtype: int64
```

```
from sklearn.model_selection import train_test_split
X = df.text   # features
y = df.author # targets
# Divide into train and test, with 80% in train
# Use random state 1234
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, random_state=1234)
# Display the shape of train and test
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
     (66,) (17,) (66,) (17,)
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
# Process the text by removing stop words and performing tf-idf vectorization
vectorizer = TfidfVectorizer(stop_words = 'english')
```

```
# fit to the training data only, applied to train and test
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)
# Output the training set shape and the test set shape
print(X_train.shape, X_test.shape)
```

```
     (66, 7727) (17, 7727)
```

```
# Try a Bernoulli Naïve Bayes model
from sklearn.naive_bayes import MultinomialNB
naive_bayes = MultinomialNB()
naive_bayes.fit(X_train, y_train)

# accuracy on the test set
from sklearn.metrics import accuracy_score
pred = naive_bayes.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
```

```
     accuracy score:  0.5882352941176471
```

```
from sklearn.model_selection import train_test_split
X = df.text   # features
y = df.author # targets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, random_

# Redo the vectorization with max_features option set to use only the 1000 most frequent words
# Add bigrams as a feature
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(stop_words = 'english', max_features=1000, ngram_range = (1, 2))
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)
# Try Naïve Bayes again on the new train/test vectors
from sklearn.naive_bayes import MultinomialNB
naive_bayes = MultinomialNB()
naive_bayes.fit(X_train, y_train)

# accuracy on the test set
from sklearn.metrics import accuracy_score
from sklearn.metrics import accuracy_score
pred = naive_bayes.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
# Compare results
print('After adding bigrams as a feature and only using the top 1000 most used words, the accurra
```

```
    accuracy score:  0.8235294117647058
    After adding bigrams as a feature and only using the top 1000 most used words, the accurracy increased from 0
```

```
# Try logistic regression
# Adjust at least one parameter in the LogisticRegression() model
# to see if you can improve results over having no parameters
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, log_loss

# set up X and y
X = df.text
y = df.author

# divide into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, random_state=1234)

# vectorizer
vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(X_train)  # fit and transform the train data
X_test = vectorizer.transform(X_test)        # transform only the test data

# train
classifier = LogisticRegression(solver='lbfgs', class_weight='balanced')
classifier.fit(X_train, y_train)

# results
pred = classifier.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
```

```
    accuracy score:  0.9411764705882353
```

```
# Try a neural network
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split

# set up X and y
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(df.text)
y = df.author

# divide into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, random_s

# train and test
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(solver='lbfgs', alpha=1e-5,
                   hidden_layer_sizes=(5, 2), random_state=1)
classifier.fit(X_train, y_train)

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score
pred = classifier.predict(X_test)
# Accuracy
print('accuracy score: ', accuracy_score(y_test, pred))
```

```
accuracy score:  0.7647058823529411
```