```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, c
```

```
import nltk
nltk.download('stopwords')
nltk.download('punkt')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
```

1. Read in the csv file using pandas. Convert the author column to categorical data. Display the first few rows. Display the counts by author.

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

```
df
```

	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...

```
author_map = {'HAMILTON': 1,
              'JAY': 2,
              'MADISON': 3,
              'HAMILTON AND MADISON': 4,
              'HAMILTON OR MADISON': 5}
df['author'] = df['author'].map(author_map).astype(int)
```

```
  80  HAMILTON     FEDERALIST No. 81 The Judiciary Continued, an
```

```
df.head()
```

	author	text
0	1	FEDERALIST. No. 1 General Introduction For the...
1	2	FEDERALIST No. 2 Concerning Dangers from Forei...
2	2	FEDERALIST No. 3 The Same Subject Continued (C...
3	2	FEDERALIST No. 4 The Same Subject Continued (C...
4	2	FEDERALIST No. 5 The Same Subject Continued (C...

```
plt.figure(figsize = (12, 8))
sns.countplot(df['author'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning:
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f84f0d59210>
```

2. Divide into train and test, with 80% in train. Use random state 1234. Display the shape of train and test.

```
X = df['text']
y = df['author']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_stat
```

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

```
X_train shape:  (66,)
X_test shape:  (17,)
y_train shape:  (66,)
y_test shape:  (17,)
```

3. Process the text by removing stop words and performing tf-idf vectorization, fit to the training data only, and applied to train and test. Output the training set shape and the test set shape.

```
stop_words = stopwords.words('english')

def remove_stopwords (data):
    filtered_data = data.apply(lambda x: ' '.join([w for w in word_tokenize(x) if w.lo
    return filtered_data
```

```
X_train = remove_stopwords(X_train)
X_test = remove_stopwords(X_test)
```

```
X_train
```

```
66    FEDERALIST . 67 Executive Department New York ...
54    FEDERALIST . 55 Total Number House Representat...
68    FEDERALIST . 69 Real Character Executive New Y...
18    FEDERALIST . 19 Subject Continued ( Insufficie...
45    FEDERALIST . 46 Influence State Federal Govern...
                              ...
24    FEDERALIST . 25 Subject Continued ( Powers Nec...
76    FEDERALIST . 77 Appointing Power Continued Pow...
53    FEDERALIST . 54 Apportionment Members Among St...
38    FEDERALIST . 39 Conformity Plan Republican Pri...
47    FEDERALIST . 48 Departments Far Separated Cons...
Name: text, Length: 66, dtype: object
```

```
X_test
```

```
70    FEDERALIST . 71 Duration Office Executive New ...
51    FEDERALIST . 52 House Representatives New York...
67    FEDERALIST . 68 Mode Electing President New Yo...
59    FEDERALIST . 60 Subject Continued ( Concerning...
35    FEDERALIST . 36 Subject Continued ( Concerning...
71    FEDERALIST . 72 Subject Continued , Re-Eligibi...
33    FEDERALIST . 34 Subject Continued ( Concerning...
63    FEDERALIST . 64 Powers Senate New York Packet ...
10    FEDERALIST . 11 Utility Union Respect Commerci...
61    FEDERALIST . 62 Senate Independent Journal . P...
65    FEDERALIST . 66 Objections Power Senate Set Co...
21    FEDERALIST . 22 Subject Continued ( Defects Pr...
55    FEDERALIST . 56 Subject Continued ( Total Numb...
4     FEDERALIST . 5 Subject Continued ( Concerning ...
46    FEDERALIST . 47 Particular Structure New Gover...
27    FEDERALIST . 28 Subject Continued ( Idea Restr...
40    FEDERALIST . 41 General View Powers Conferred ...
Name: text, dtype: object
```

```
vectorizer = TfidfVectorizer()
vectorizer.fit(X_train)
```

```
TfidfVectorizer()
```

```
X_train_vec = vectorizer.transform(X_train)
X_test_vec = vectorizer.transform(X_test)
```

```
print('X_train_vec shape', X_train_vec.shape)
print('X_test_vec shape', X_test_vec.shape)
```

```
    X_train_vec shape (66, 7883)
    X_test_vec shape (17, 7883)
```

4. Try a Bernoulli Naïve Bayes model. What is your accuracy on the test set?

```
from sklearn.naive_bayes import MultinomialNB
```

```
naive_bayes = MultinomialNB()
naive_bayes.fit(X_train_vec, y_train)

    MultinomialNB()
```

```
pred = naive_bayes.predict(X_test_vec)
```

```
print('accuracy score: ', accuracy_score(y_test, pred))

    accuracy score:  0.5882352941176471
```

5. The results from step 4 will be disappointing. The classifier just guessed the predominant class, Hamilton, every time. Looking at the train data shape above, there are 7876 unique words in the vocabulary. This may be too much, and many of those words may not be helpful. Redo the vectorization with max_features option set to use only the 1000 most frequent words. In addition to the words, add bigrams as a feature. Try Naïve Bayes again on the new train/test vectors and compare your results.

```
vectorizer = TfidfVectorizer(max_features = 1000, ngram_range = (2, 2))
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)
```

```
print('X_train_vec shape', X_train_vec.shape)
print('X_test_vec shape', X_test_vec.shape)

    X_train_vec shape (66, 1000)
    X_test_vec shape (17, 1000)
```

```
naive_bayes = MultinomialNB()
naive_bayes.fit(X_train_vec, y_train)

    MultinomialNB()
```

```
pred = naive_bayes.predict(X_test_vec)
```

```
print('accuracy score: ', accuracy_score(y_test, pred))

    accuracy score:  0.5882352941176471
```

THe results are the same as the preivous experiment

6. Try logistic regression. Adjust at least one parameter in the LogisticRegression() model to see if you can improve results over having no parameters. What are your results?

```
from sklearn.linear_model import LogisticRegression
```

```
lgm = LogisticRegression()
lgm.fit(X_train_vec, y_train)

    LogisticRegression()
```

```
pred = lgm.predict(X_test_vec)
```

```
print('accuracy score: ', accuracy_score(y_test, pred))

    accuracy score:  0.5882352941176471
```

```
lgm = LogisticRegression(multi_class = 'multinomial')
lgm.fit(X_train_vec, y_train)
```

```
     LogisticRegression(multi_class='multinomial')
```

```
pred = lgm.predict(X_test_vec)
print('accuracy score: ', accuracy_score(y_test, pred))
```

```
     accuracy score:  0.5882352941176471
```

Adding a parameter did not change the accuracy

7. Try a neural network. Try different topologies until you get good results. What is your final accuracy?

```
from sklearn.neural_network import MLPClassifier
```

```
def neural_network_predictions (topology):
    nn = MLPClassifier(solver = 'lbfgs', alpha = 1e-5,
                  hidden_layer_sizes = topology, random_state = 1)
    nn.fit(X_train_vec, y_train)
    pred = nn.predict(X_test_vec)
    print('accuracy score: ', accuracy_score(y_test, pred))
```

```
neural_network_predictions(topology = (5))
```

```
     accuracy score:  0.7058823529411765
```

```
neural_network_predictions(topology = (5, 5))
```

```
     accuracy score:  0.5294117647058824
     /usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_percep
     STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

     Increase the number of iterations (max_iter) or scale the data as shown in:
         https://scikit-learn.org/stable/modules/preprocessing.html
       self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

```
neural_network_predictions(topology = (5, 2))
```

```
     accuracy score:  0.6470588235294118
```

```
neural_network_predictions(topology = (4, 4))
```

```
     accuracy score:  0.6470588235294118
```

```
neural_network_predictions(topology = (2, 2, 2, 2))

     accuracy score:  0.5882352941176471


neural_network_predictions(topology = (6, 2))

     accuracy score:  0.5882352941176471


neural_network_predictions(topology = (8))

     accuracy score:  0.7058823529411765
```

The highest accuracy we were able to get is around 70%, achieved with one layer. Both 8 and 5 nodes achieve this accuracy