```
# 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.

import pandas as pd # Load the Pandas libraries with alias 'pd'
df = pd.read_csv("federalist.csv")
df['author'] = pd.Categorical(df.author)

# Preview the first 5 lines of the loaded data
print(df.head())
print("\n")
print(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
```

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

# divide into train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df.text, df.author, test_size=0.2, train_

# Display the shape of train and test.
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

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

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

# removing stop words and performing tf-idf vectorization
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
stopwords = set(stopwords.words('english'))
vectorizer = TfidfVectorizer(stop_words=stopwords)
```

```
vectorizer = TfidfVectorizer(stop_words=stopwords)

# vectorize
X_train_fit = vectorizer.fit_transform(X_train) # returns document term matrix
X_test_fit = vectorizer.transform(X_test)

print("Train and test sizes (shapes): ", X_train.shape, X_test.shape)
print("peek the data:\n", X_train_fit.toarray(), '\n\n', X_test_fit.toarray())
# print("peek the data:\n", X_train,'\n\n', X_test)
# print("peek the labels:\n", y_train,'\n\n', y_test)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Train and test sizes (shapes):  (66,) (17,)
peek the data:
 [[0.         0.         0.02956872 ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 ...
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.02275824 0.         0.        ]]

 [[0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.02314673 0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 ...
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]]
```

```
# 4.  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_fit, y_train)

# evaluate on the test data
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusic
# make predictions on the test data
pred = naive_bayes.predict(X_test_fit)

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

```
accuracy score:  0.5882352941176471
```

```
# 5.  Redo the vectorization with max_features option set to use only the 1000 most frequent
# In addition to the words, add bigrams as a feature.
# Try Naïve Bayes again on the new train/test vectors and compare your results.

# new vectorization
vectorizer2 = TfidfVectorizer(stop_words=stopwords, max_features=1000, ngram_range=(1, 2))
X_train_fit2 = vectorizer2.fit_transform(X_train) # returns document term matrix
X_test_fit2 = vectorizer.transform(X_test)
```

```
naive_bayes.fit(X_train_fit2, y_train)

pred = naive_bayes.predict(X_test_fit2)

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

```
accuracy score:  0.5882352941176471
```

```
# 6.  Try logistic regression. Adjust at least one parameter in the LogisticRegression() mode
# to see if you can improve results over having no parameters.
# What are your results?
from sklearn.linear_model.logistic import LogisticRegression

classifier = LogisticRegression()
classifier.fit(X_train_fit2, y_train)

pred = classifier.predict(X_test_fit2)
print('accuracy score without params:\t', accuracy_score(y_test, pred))

# Change parameters
classifier = LogisticRegression(multi_class='multinomial', solver='lbfgs', class_weight='bala
classifier.fit(X_train_fit2, y_train)

pred = classifier.predict(X_test_fit2)
print('accuracy score using params:\t', accuracy_score(y_test, pred))
```

```
accuracy score without params:   0.5882352941176471
accuracy score using params:     0.7647058823529411
```

```
# 7.  Neural Network

# start with straightforward design
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(15, 2), random_sta
classifier.fit(X_train_fit2, y_train)

pred = classifier.predict(X_test_fit2)
print('15, 2 accuracy:\t', accuracy_score(y_test, pred))

# try other topologies
for i in [(44),        # 2/3 the size of the input in a single layer
          (30, 14),    # 2/3 the size of the input in two layers
          (22, 14, 8), # 2/3 the size of the input in two layers
          (33),     # 1/2 the size of the input in a single layer
          (22, 11), # 1/2 the size of the input in two layers
          (22),     # 1/3 the size of the input in a single layer
          (15, 7),  # 1/3 the size of the input in two layers
          (5), # near the output size
          (6),  # near the output size
```

```
            (6), # near the output size
            (7), # near the output size
            (30, 10), # playing with numbers near (30, 14)
            (35, 14), # playing with numbers near (30, 14)
            (35, 12), # playing with numbers near (30, 14)

            ]:
    classifier = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(i), random_state
    classifier.fit(X_train_fit2, y_train)
    pred = classifier.predict(X_test_fit2)
    print(i, 'accuracy:\t', accuracy_score(y_test, pred))

print("\nThe best is to take 6 nodes in a single layer! Who could've guessed.")
```

```
15, 2 accuracy:  0.7058823529411765
44 accuracy:     0.7647058823529411
(30, 14) accuracy:       0.8235294117647058
(22, 14, 8) accuracy:    0.8235294117647058
33 accuracy:     0.7058823529411765
(22, 11) accuracy:       0.7647058823529411
22 accuracy:     0.7058823529411765
(15, 7) accuracy:        0.7647058823529411
5 accuracy:      0.8235294117647058
6 accuracy:      0.8823529411764706
7 accuracy:      0.7647058823529411
(30, 10) accuracy:       0.7058823529411765
(35, 14) accuracy:       0.8235294117647058
(35, 12) accuracy:       0.7647058823529411

The best is to take 6 nodes in a single layer! Who could've guessed.
```