Read in the csv file using pandas.

In [1]:
```
import pandas as pd
df = pd.read_csv('federalist.csv')
df[12:18]     #just showing myself there are rows where authors are not exactly known
```

Out[1]:

	author	text
12	HAMILTON	FEDERALIST No. 13 Advantage of the Union in Re...
13	MADISON	FEDERALIST No. 14 Objections to the Proposed C...
14	HAMILTON	FEDERALIST No. 15 The Insufficiency of the Pre...
15	HAMILTON	FEDERALIST No. 16 The Same Subject Continued (...
16	HAMILTON	FEDERALIST No. 17 The Same Subject Continued (...
17	HAMILTON AND MADISON	FEDERALIST No. 18 The Same Subject Continued (...

Convert the author column to categorical data. Display the first few rows.

In [2]:
```
print(df.dtypes, '\n')
```

```
author    object
text      object
dtype: object
```

In [3]:
```
df['author'] = pd.Categorical(df.author)
print(df.dtypes, '\n')
print(df.head())
```

```
author    category
text        object
dtype: object

     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...
```

Display the counts by author.

In [4]:
```
#display counts by author
s=pd.Series(df['author'])
s.value_counts()
```

Out[4]:
```
HAMILTON               49
MADISON                15
HAMILTON OR MADISON    11
JAY                     5
HAMILTON AND MADISON    3
Name: author, dtype: int64
```

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

In [5]:
```
from sklearn.model_selection import train_test_split
X = df.text
y = df.author
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, train_size=0.8, random_state=1234)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

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

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.

In [6]:
```
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer

stopwords = set(stopwords.words('english'))
vectorizer = TfidfVectorizer(stop_words=stopwords)

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

# take a peek at the data
print('train size:', X_train.shape)
print('\ntest size:', X_test.shape)
```

```
train size: (66, 7876)

test size: (17, 7876)
```

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

In [7]:
```
from sklearn.naive_bayes import BernoulliNB
BNB = BernoulliNB()
BNB.fit(X_train, y_train)

from sklearn.metrics import accuracy_score
pred = BNB.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
```

```
accuracy score:  0.5882352941176471
```

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.

In [8]:
```
# re-apply tfidf vectorizer with max_features and adding bigrams.
X = df.text
y = df.author
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 = TfidfVectorizer(stop_words=stopwords, max_features=1000, ngram_range=(1,2))
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)
BNB = BernoulliNB()
BNB.fit(X_train, y_train)
pred = BNB.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
```

```
accuracy score:  0.9411764705882353
```

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?

In [9]:
```
#Logistic regression without parameters
from sklearn.linear_model import LogisticRegression
X = df.text
y = df.author
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 = TfidfVectorizer(stop_words=stopwords, max_features=1000, ngram_range=(1,2))
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)
LR = LogisticRegression()
LR.fit(X_train, y_train)
pred = LR.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
```

```
accuracy score:  0.5882352941176471
```

In [10]:
```
#Logistic regression with parameters
X = df.text
y = df.author
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 = TfidfVectorizer(stop_words=stopwords, max_features=1000, ngram_range=(1,2))
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)
LR = LogisticRegression(solver='newton-cg', class_weight='balanced', random_state=5, C=.2)
LR.fit(X_train, y_train)
pred = LR.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
```

```
accuracy score:  0.8235294117647058
```

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

In [20]:
```
from sklearn.neural_network import MLPClassifier
X = df.text
y = df.author
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 = TfidfVectorizer(stop_words=stopwords, max_features=1000, ngram_range=(1,2))
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)
Classifier = MLPClassifier(activation='tanh', solver='adam', alpha=.0001, batch_size='auto', max_iter=1000)
Classifier.fit(X_train, y_train)
pred = Classifier.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
```

```
accuracy score:  0.8823529411764706
```

In [24]:
```
Classifier = MLPClassifier(hidden_layer_sizes=(150,5), activation='tanh', solver='adam', alpha=.0001, batch_size='auto', max_iter=1000)
Classifier.fit(X_train, y_train)
pred = Classifier.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
```

```
accuracy score:  0.8823529411764706
```

In [36]:
```
Classifier = MLPClassifier(hidden_layer_sizes=(200,150, 100, 50), activation='tanh', solver='adam', alpha=.0001, batch_size='auto', max_iter=1500)
Classifier.fit(X_train, y_train)
pred = Classifier.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
```

```
accuracy score:  0.8235294117647058
```

In [ ]: