Import packages

```
import pandas as pd
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import BernoulliNB
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
```

Read CSV file. Display relevant data.

```
df = pd.read_csv('./federalist.csv')
# df[['author']] = df[['author']].astype('category')

print(df.head()[:5])
print()
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...

author
HAMILTON               49
MADISON                15
HAMILTON OR MADISON    11
JAY                     5
HAMILTON AND MADISON    3
dtype: int64
```

Create training and testing datasets.

```
# Split data into 80% train and 20% test sets
# Use random state 1234 to ensure reproducibility
train, test = train_test_split(df, test_size=0.2, random_state=1234)
print('Training data shape:', train.shape)
print('Testing data shape: ', test.shape)
```

```
Training data shape: (66, 2)
Testing data shape:  (17, 2)
```

Perform text preprocessing.

```
# Remove stop words from the datasets using tfifdfVectorizer
stop_words = stopwords.words('english')
vectorizer = TfidfVectorizer(stop_words=stop_words)

# Fit and transform the vectorizer on the training data
X_train = vectorizer.fit_transform(train['text'])
y_train = train['author']

# Transform the vectorizer on the test data
X_test = vectorizer.transform(test['text'])
y_test = test['author']

print('Training set shape:', X_train.shape)
print('Testing set shape: ', X_test.shape)
```

```
Training set shape: (66, 7876)
Testing set shape:  (17, 7876)
```

Classify the data using Naive Bayes.

```
# Fit the data to Naive Bayes model
nb = BernoulliNB()
nb.fit(X_train, y_train)

# Get the accuracy score for the Naive Bayes model
acc = nb.score(X_test, y_test)

print('Accuracy:', str(round(acc * 100, 2)) + '%')
```

```
Accuracy: 58.82%
```

Why the low accuracy? The data is imbalanced and this causes the model to be biased towards the majority class.

```
pred = nb.predict(X_train)
pred2 = nb.predict(X_test)

print('Train set predictions: \n', pred)
print()
print('Test set predictions: \n', pred2)
```

```
Train set predictions:
 ['HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON'
 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON'
 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON'
 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON'
```

```
 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON'
 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON'
 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON'
 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON'
 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON'
 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON'
 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON']

Test set predictions:
 ['HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON'
 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON'
 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON' 'HAMILTON']
```

The accuracy of the Naive Bayes model is lower than expected so well try to improve it by changing the vectorizer parameters.

```
# Limit the number of features to 1000
# Utilize unigrams and bigrams
vectorizer2 = TfidfVectorizer(stop_words=stop_words, max_features=1000, ngram_range=(1, 2))

X_train2 = vectorizer2.fit_transform(train['text'])
y_train2 = train['author']

X_test2 = vectorizer2.transform(test['text'])
y_test2 = test['author']

nb2 = BernoulliNB()
nb2.fit(X_train2, y_train2)
acc2 = nb2.score(X_test2, y_test2)
print('Accuracy:', str(round(acc2 * 100, 2)) + '%')
```

```
Accuracy: 94.12%
```

Classify the data using Logistic Regression.

- Changing the maximum iterations seemed to improve the results. (though the results are still lackluster)

```
lr = LogisticRegression(class_weight='balanced', multi_class='multinomial', max_iter=1000)
lr.fit(X_train2, y_train2)

acc2 = lr.score(X_test2, y_test2)
print('Accuracy:', str(round(acc2 * 100, 2)) + '%')
```

```
Accuracy: 76.47%
```

Classify the data using a Neural Network.

```
# Scale the data for faster convergence
scaler = StandardScaler(with_mean=False)
scaler.fit(X_train2)
X_train2_scaled = scaler.transform(X_train2)
X_test2_scaled = scaler.transform(X_test2)

# Fit the data to a neural network
regr = MLPClassifier(random_state=1234, max_iter=1000, hidden_layer_sizes=(100))
regr.fit(X_train2_scaled, y_train2)

# Get the accuracy score for the neural network
acc = regr.score(X_test2_scaled, y_test2)
print('Accuracy:', str(round(acc * 100, 2)) + '%')
```

```
Accuracy: 88.24%
```