```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
```

Step 1

```
# Read in the csv file using pandas
df = pd.read_csv('federalist.csv')

# Convert the author column to categorical data
df['author'] = pd.Categorical(df.author)
print("Types of Data Frame Fields")
print(df.dtypes)

# Display the first few rows
print("Display First Few Rows")
print(df.head())

#  Display the counts by author
print("Display the counts by author")
print(df["author"].value_counts())
```

```
Types of Data Frame Fields
author    category
text        object
dtype: object
Display First Few Rows
     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
HAMILTON                49
MADISON                 15
HAMILTON OR MADISON     11
JAY                      5
HAMILTON AND MADISON     3
Name: author, dtype: int64
```

Step 2 + Step 3

```
# Step 2
# Divide into train and test, with 80% in train. Use random state 1234.
```

```
X= df['text']
y= df['author']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

# Display the shape of train and test
train_np = np.array([y_train, X_train])
test_np = np.array([X_test, y_test])
print('Train Shape\n',train_np.shape)
print('Test Shape\n',test_np.shape)

# Step 3
# remove stop words
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stopwords = set(stopwords.words('english'))

# tf-idf vectorization
vectorizer = TfidfVectorizer(stop_words=stopwords)
X_train = vectorizer.fit_transform(X_train)  # fit and transform the train data
X_test = vectorizer.transform(X_test)        # transform only the test data

# Output the training set shape and the test set shape.
print('Train Shape after tf-idf\n',X_train.shape)
print('Test Shape after tf-idf\n',X_test.shape)
```

```
Train Shape
 (2, 66)
Test Shape
 (2, 17)
Train Shape after tf-idf
 (66, 7876)
Test Shape after tf-idf
 (17, 7876)
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Step 4

```
# Try a Bernoulli Naïve Bayes model.
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, c

naive_bayes2 = BernoulliNB()
naive_bayes2.fit(X_train, y_train)

# make predictions on the test data
pred = naive_bayes2.predict(X_test)

# print confusion matrix
from sklearn.metrics import confusion_matrix
```

```
confusion_matrix(y_test, pred)

print("NB Bernoulli Statistics")
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred, average='micro'))
print('recall score: ', recall_score(y_test, pred, average='micro'))
print('f1 score: ', f1_score(y_test, pred, average='micro'))
```

```
NB Bernoulli Statistics
accuracy score:  0.5882352941176471
precision score:  0.5882352941176471
recall score:  0.5882352941176471
f1 score:  0.5882352941176471
```

Step 5

```
# 1000 most frequent words and add bigrams as feature
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
vectorizer = TfidfVectorizer(stop_words=stopwords, max_features=1000, ngram_range = (
X_train = vectorizer.fit_transform(X_train)  # fit and transform the train data
X_test = vectorizer.transform(X_test)        # transform only the test data

# Try a Bernoulli Naïve Bayes model again.
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, 

naive_bayes2 = BernoulliNB()
naive_bayes2.fit(X_train, y_train)

# make predictions on the test data
pred = naive_bayes2.predict(X_test)

# print confusion matrix
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, pred)
print("New Metrics")
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred, average='micro'))
print('recall score: ', recall_score(y_test, pred, average='micro'))
print('f1 score: ', f1_score(y_test, pred, average='micro'))
```

```
New Metrics
accuracy score:  0.9411764705882353
precision score:  0.9411764705882353
recall score:  0.9411764705882353
f1 score:  0.9411764705882353
```

Step 6

```
# logistic regression
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression

pipe1 = Pipeline([
        ('tfidf', TfidfVectorizer(stop_words=stopwords, max_features=1000, ngram_range
        ('logreg', LogisticRegression(multi_class='multinomial', solver='saga', class_
])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

pipe1.fit(X_train, y_train)

# make predictions on the test data
pred = pipe1.predict(X_test)

import numpy as np
print("\nOverall accuracy: ", np.mean(pred==y_test))
```

```
     Overall accuracy:  0.8235294117647058
     /usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_sag.py:354: Converg
       ConvergenceWarning,
```

When I have no parameters, my accuracy is 0.588. This value does not change until I add the class weight as balanced, then the accuracy jumps to 0.76. The accuracy jumps to 0.823 when I make the solver saga instead of lbfgs.

Step 7

```
from sklearn.neural_network import MLPClassifier
pipe1 = Pipeline([
        ('tfidf', TfidfVectorizer()),
        ('neuralnet', MLPClassifier(solver='lbfgs', alpha=1e-5,
                   hidden_layer_sizes=(15, 9), random_state=1)),
         ])

pipe1.fit(X_train, y_train)

# make predictions on the test data
pred = pipe1.predict(X_test)

print("\nOverall accuracy: ", np.mean(pred==y_test))
```

```
     Overall accuracy:  0.8823529411764706
```

My final accuracy is 0.882. Changing the hidden layer sizes gave me various results. I started with 15 and 8 and I changed this to 15 and 9. Higher values for either number started decreasing the accuracy.