In [ ]:
```
import pandas as pd
from pandas.api.types import CategoricalDtype
df = pd.read_csv('federalist.csv')
cat_type = CategoricalDtype(categories=["HAMILTON", "JAY", "HAMILTON AND MADISON", "MA
df['author'] = df['author'].astype(cat_type)
print(df.head())
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
```

In [ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

from nltk.corpus import stopwords

stops = set(stopwords.words('english'))
vectorization = TfidfVectorizer(stop_words=stops)
```

In [ ]:
```
import numpy as np
import seaborn as sb
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

print(df.text)
print(df.author)

Xdata = df.text
Ytarget = df.author

X_train, X_test, y_train, y_test = train_test_split(Xdata, Ytarget, test_size=.2, trai
print("Training shape: ", X_train.shape)
print("Testing shape: ", X_test.shape)

#become "Vector!!!! a mathematical term, a quantity represented by an arrow with Direc
X_train = vectorization.fit_transform(X_train)
X_test = vectorization.transform(X_test)
```

```
0     FEDERALIST. No. 1 General Introduction For the...
1     FEDERALIST No. 2 Concerning Dangers from Forei...
2     FEDERALIST No. 3 The Same Subject Continued (C...
3     FEDERALIST No. 4 The Same Subject Continued (C...
4     FEDERALIST No. 5 The Same Subject Continued (C...
                          ...
78    FEDERALIST No. 79 The Judiciary Continued From...
79    FEDERALIST No. 80 The Powers of the Judiciary ...
80    FEDERALIST. No. 81 The Judiciary Continued, an...
81    FEDERALIST No. 82 The Judiciary Continued From...
82    FEDERALIST No. 83 The Judiciary Continued in R...
Name: text, Length: 83, dtype: object
0     HAMILTON
1          JAY
2          JAY
3          JAY
4          JAY
        ...
78    HAMILTON
79    HAMILTON
80    HAMILTON
81    HAMILTON
82    HAMILTON
Name: author, Length: 83, dtype: category
Categories (5, object): ['HAMILTON' < 'JAY' < 'HAMILTON AND MADISON' < 'MADISON' < 'H
AMILTON OR MADISON']
Training shape:  (66,)
Testing shape:  (17,)
```

In [ ]:
```
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score

#maksure to run all code before this statement(even the other blocks) it might get pis
nb = BernoulliNB()
nb.fit(X_train, y_train)

prediction = nb.predict(X_test)

print("Accuracy: ", accuracy_score(y_test, prediction))
```

```
Accuracy:  0.5882352941176471
```

In [ ]:
```
#i require more accuracy
vectorization = TfidfVectorizer(stop_words=stops, max_features=1000, ngram_range=(1, 2

X_train, X_test, y_train, y_test = train_test_split(Xdata, Ytarget, test_size=.2, trai

#become "Vector!!!! a mathematical term, a quantity represented by an arrow with Direc
X_train = vectorization.fit_transform(X_train)
X_test = vectorization.transform(X_test)

nb.fit(X_train, y_train)
prediction = nb.predict(X_test)

#oh took at that increase Poggies
print("Accuracy: ", accuracy_score(y_test, prediction))
```

```
Accuracy:  0.9411764705882353
```

In [ ]:
```
from sklearn.linear_model import LogisticRegression

#now without bigrams
vectorization = TfidfVectorizer(stop_words=stops, max_features=1000)
Xtf = vectorization.fit_transform(Xdata)
X_train, X_test, y_train, y_test = train_test_split(Xtf, Ytarget, test_size=.2, train_

#out of the box performance
LRDefaults = LogisticRegression()

#hope for more accuracy by double iteration, and balancing class weight
LRAcc = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=200, cl

LRDefaults.fit(X_train, y_train)
LRAcc.fit(X_train, y_train)

PDefaults = LRDefaults.predict(X_test)
PAcc = LRAcc.predict(X_test)

#success Pog
print("Default Accuracy: ", accuracy_score(y_test, PDefaults))
print("(Hopefully) Better Accuracy: ", accuracy_score(y_test, PAcc))
```

```
Default Accuracy:  0.5882352941176471
(Hopefully) Better Accuracy:  0.7647058823529411
```

In [ ]:
```
from sklearn.neural_network import MLPClassifier

#run previous to get the vectorization object and Xtf, as well as the x and y trains a
classics = MLPClassifier()
classics.fit(X_train, y_train)

#i was gonna abv it to CP but on second thought maybe thats not a goood idea
classicParams = MLPClassifier(max_iter=1000, solver='lbfgs', hidden_layer_sizes=(50, 4
classicParams.fit(X_train, y_train)

prediction = classics.predict(X_test)
#ha
PP = classicParams.predict(X_test)

print("Accuracy: ", accuracy_score(y_test, prediction))
print("Param Accuracy: ", accuracy_score(y_test, PP))
```

```
C:\Users\antho\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra
8p0\LocalCache\local-packages\Python310\site-packages\sklearn\neural_network\_multila
yer_perceptron.py:702: ConvergenceWarning: Stochastic Optimizer: Maximum iterations
(200) reached and the optimization hasn't converged yet.
  warnings.warn(
Accuracy:  0.7647058823529411
Param Accuracy:  0.8235294117647058
```