```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import os

#EDA and preprocessing
import re
import nltk.corpus
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from string import digits


# In[2]:


from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.decomposition import NMF
from sklearn.metrics import accuracy_score
import sklearn.metrics as metrics
import itertools
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split

#find the files names
for dirname, _, filenames in os.walk('file.csv'):
    for filename in filenames:
        print(os.path.join(dirname, filename))


# In[3]:


path_dir = '/kaggle/input/learn-ai-bbc/'
train_path = path_dir + 'BBC News Train.csv'
sample_solution_path = path_dir + 'BBC News Sample Solution.csv'
test_path = path_dir + 'BBC News Test.csv


# In[4]:


train = pd.read_csv(train_path)
sample_solution = pd.read_csv(sample_solution_path)
test = pd.read_csv(test_path)


# In[5]:


sample_solution


# In[6]:


train


# In[7]:


train.describe()


# In[8]:


train.info()


# In[9]:


train['ArticleId'].nunique()


# In[10]:


train['Category'].unique()


# In[11]:
```

```
train['Text'][0]


# In[12]:


fig, ax = plt.subplots(figsize=(8, 5))
sns.histplot(
    data = train,
    x = 'Category',
    hue = 'Category',
    palette = 'colorblind',
    legend = False,
    ).set(
        title = 'Category Counts');


# In[13]:


def clean_text(dataframe, text_col):
    '''
    A helper function which takes a dataframe
    and removes punction and stopwords.
    '''
    #remove all punctuation
    dataframe['no_punct'] = dataframe[text_col].apply(lambda row: re.sub(r'[^\w\s]+', '', row))

    #remove numbers
    dataframe['no_punct_num'] = dataframe['no_punct'].apply(lambda row: re.sub(r'[0-9]+', '', row))

    #remove stopwords
    stop_words = stopwords.words('english')
    dataframe['no_stopwords'] = dataframe['no_punct_num'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop_words)]))

    #remove extra spaces
    dataframe['clean_text'] = dataframe['no_stopwords'].apply(lambda x: re.sub(' +', ' ', x))
    return


# In[14]:


clean_text(train, 'Text')


# In[15]:


train['clean_text'][1]


# In[16]:


# tokenize text function
wordnet_lemmatizer = WordNetLemmatizer()
def lemmatizer(text):
    '''
    A helper function to lemmatize an entire sentence/string
    '''
    lem = [wordnet_lemmatizer.lemmatize(word.lower()) for word in text]
    return lem

def tokenize_lemmatize(dataframe, text_col):
    '''
    A helper function to tokenize then lemmatize the string.
    Also, add column which counts the number of words in that string.
    '''
    dataframe['tokenized'] = dataframe.apply(lambda row: nltk.word_tokenize(row[text_col]), axis=1)
    dataframe['lemmatized'] = dataframe['tokenized'].apply(lambda string: lemmatizer(string))
    dataframe['num_words'] = dataframe['lemmatized'].apply(lambda lst: len(lst))
    retur


# In[17]:


tokenize_lemmatize(train, 'clean_text')


# In[18]:


fig, ax = plt.subplots(figsize=(15, 5))
sns.histplot(
    data = train,
    x = 'num_words',
    palette = 'colorblind',
    ).set(
```

```
        title = 'Number of Words per Article');


# In[19]:


train = train[train['num_words'] < 750]
len(train)


# In[20]:


train_df = train.copy()


# In[21]:


def predict(w_matrix):
    sortedW = np.argsort(w_matrix)
    n_predictions, maxValue = sortedW.shape
    predictions = [[sortedW[i][maxValue - 1]] for i in range(n_predictions)]
    topics = np.empty(n_predictions, dtype = np.int64)
    for i in range(n_predictions):
        topics[i] = predictions[i][0]
    return topics


# In[22]:


def label_permute(ytdf,yp,n=5):
    """
    ytdf: labels dataframe object
    yp: clustering label prediction output
    Returns permuted label order and accuracy.
    Example output: (3, 4, 1, 2, 0), 0.74
    """
    perms = list(itertools.permutations([0, 1, 2, 3, 4]))    #create permutation list
    best_labels = []
    best_acc = 0
    current = {}
    labels = ['business', 'tech', 'politics', 'sport', 'entertainment']
    for perm in perms:
        for i in range(n):
            current[labels[i]] = perm[i]
            if len(current) == 5:
                conditions = [
                    (ytdf['Category'] == current['business']),
                    (ytdf['Category'] == current['tech']),
                    (ytdf['Category'] == current['politics']),
                    (ytdf['Category'] == current['sport']),
                    (ytdf['Category'] == current['entertainment'])]
                ytdf['test'] = ytdf['Category'].map(current)
                current_accuracy = accuracy_score(ytdf['test'], yp)
                if current_accuracy > best_acc:
                    best_acc = current_accuracy
                    best_labels = perm
                    ytdf['best'] = ytdf['test']
    return best_labels, best_acc


# In[23]:


#create vectorizer
tfidvec = TfidfVectorizer(min_df = 2,
                          max_df = 0.95,
                          norm = 'l2',
                          stop_words = 'english')
tfidvec_train = tfidvec.fit_transform(train_df['clean_text'])

#create model
nmf_model = NMF(n_components=5,
                init='nndsvda',
                solver = 'mu',
                beta_loss = 'kullback-leibler',
                l1_ratio = 0.5,
                random_state = 101)
nmf_model.fit(tfidvec_train)

#view results
yhat_train = predict(nmf_model.transform(tfidvec_train))
label_order, accuracy = label_permute(train_df, yhat_train )
print('accuracy=', accuracy


# In[24]:


label_dict = {4:'business', 2:'tech', 1:'politics', 0:'sport', 3:'entertainment'}
```

```
for i in range(5):
    print(f'{label_order[i]}:  {label_dict[label_order[i]]}')


# In[25]:


clean_text(test, 'Text')
tfidvec_test = tfidvec.transform(test['clean_text'])
yhat_test = predict(nmf_model.transform(tfidvec_test)


# In[26]:


test_predictions = pd.DataFrame(columns=['ArticleId', 'Category', 'yhat'])
test_predictions['ArticleId'] = test['ArticleId']
test_predictions['yhat'] = yhat_test
test_predictions['Category'] = test_predictions['yhat'].apply(lambda i: label_dict[i])

#delete columns unneeded for submission
test_predictions = test_predictions.drop('yhat', 1)
print(test_predictions.head(15))


# In[27]:


try:
    test_predictions.to_csv('submission.csv', index=False)
except:
    pass


# In[28]:


train = pd.read_csv(train_path)
test = pd.read_csv(test_path)


# In[29]:


#clean data
clean_text(train, 'Text')

#split data into X and y
y_train = train['Category'].values
X_train = train['clean_text'].values

#create new vectorizer for supervised learning model
tfidfvec_supervised = TfidfVectorizer(min_df = 2,
                          max_df = 0.95,
                          norm = 'l2',
                          stop_words = 'english')
tfSuper_train = tfidfvec_supervised.fit_transform(X_train)

#create KMeans Model and train
kmeans = KMeans(n_clusters = 5,
                init = 'k-means++',
                algorithm = 'full',
                random_state = 101)
yhat_train_super = kmeans.fit_predict(tfSuper_train)

#get accuracy
y_train_df = pd.DataFrame(y_train, columns=['Category'])
label_order, accuracy = label_permute(y_train_df, yhat_train_super)
print('accuracy=', accuracy)
print(label_order, '\n')

#show label order
label_dict = {3:'business', 1:'tech', 4:'politics', 2:'sport', 0:'entertainment'}
for i in range(5):
    print(f'{label_order[i]}:  {label_dict[label_order[i]]}')


# In[30]:


#clean data
clean_text(test, 'Text')

#split data
X_test = test['clean_text'].values

#create vectorizer (do not fit it!)
tfSuper_test = tfidfvec_supervised.transform(X_test)
yhat_test = kmeans.predict(tfSuper_test)

#create a submission dataframe
test_predictions = pd.DataFrame(columns=['ArticleId', 'Category', 'yhat'])
```

```
test_predictions['ArticleId'] = test['ArticleId']
test_predictions['yhat'] = yhat_test
test_predictions['Category'] = test_predictions['yhat'].apply(lambda i: label_dict[i])

#delete columns unneeded for submission
test_predictions = test_predictions.drop('yhat', 1)
print(test_predictions.head(2))


# In[ ]:
```