Introduction

Today we will be looking over the BBC news classification dataset. This dataset it comprised of 2225 articles, each in 1 of 5 different categories: business, entertainment, politics, sports or tech.

Goal

The goal will to build a model that can accurately classify unseen news articles into the right category.

Imports

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import NMF
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from collections import Counter
```

Load Data

In [2]:
```
train = pd.read_csv("BBC News Train.csv")
test = pd.read_csv("BBC News Test.csv")
```

Data Quick Glance

In [3]:
```
print("Train:", train.shape)
print("Test:", test.shape)
print(train.head())
print(test.head())
```

```
Train: (1490, 3)
Test: (735, 2)
   ArticleId                                               Text  Category
0       1833  worldcom ex-boss launches defence lawyers defe...  business
1        154  german business confidence slides german busin...  business
2       1101  bbc poll indicates economic gloom citizens in ...  business
3       1976  lifestyle  governs mobile choice  faster  bett...      tech
4        917  enron bosses in $168m payout eighteen former e...  business
   ArticleId                                               Text
0       1018  qpr keeper day heads for preston queens park r...
1       1319  software watching while you work software that...
2       1138  d arcy injury adds to ireland woe gordon d arc...
3        459  india s reliance family feud heats up the ongo...
4       1020  boro suffer morrison injury blow middlesbrough...
```

Exploratory Data Analysis

Now we want to look over the data and see what we can find in it. Patterns and what not.

Data Cleaning

In [4]:
```
print("Missing Values in the training set?")
print(train.isnull().sum())

print("Missing Values in the test set?")
print(test.isnull().sum())
```

```
Missing Values in the training set?
ArticleId    0
Text         0
Category     0
dtype: int64
Missing Values in the test set?
ArticleId    0
Text         0
dtype: int64
```

Perfect! Nothing is missing. No worries of how to deal with it.

Data Inspection/Visualization

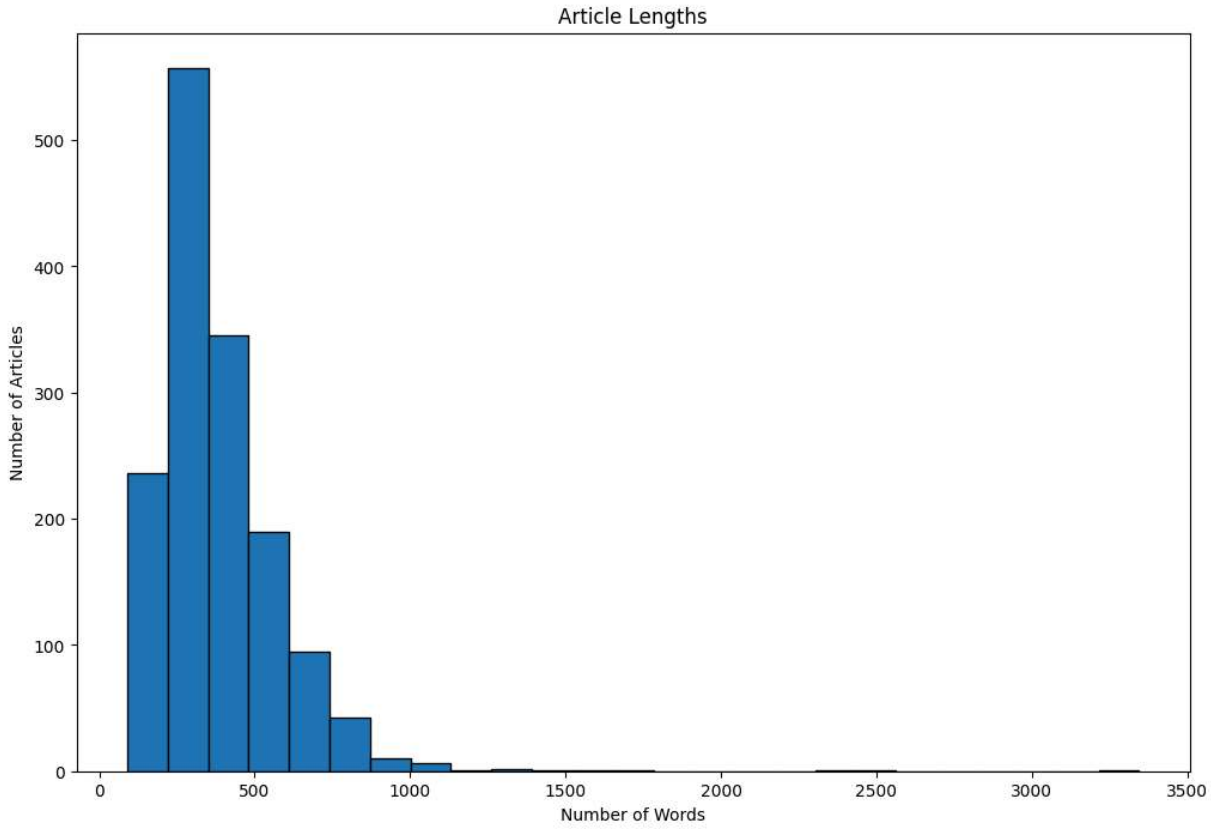
First lets look at the normal article title lengths.

In [5]:
```
train['Text Length'] = train['Text'].apply(lambda x: len(str(x).split()))


text_lengths = train['Text Length']
print("Some Statistics About Text Length:")
print("Minimum number of words:", text_lengths.min())
print("Maximum number of words:", text_lengths.max())
print("Average number of words:", round(text_lengths.mean(), 2))
print("Median number of words:", text_lengths.median())
print("Standard Deviation:", round(text_lengths.std(), 2))
```

```
Some Statistics About Text Length:
Minimum number of words: 90
Maximum number of words: 3345
Average number of words: 385.01
Median number of words: 337.0
Standard Deviation: 210.9
```

Lets see the article length distribution visually.

In [6]:
```
plt.figure(figsize=(12,8))
plt.hist(train['Text Length'], bins=25, edgecolor='black')
plt.title('Article Lengths')
plt.xlabel('Number of Words')
plt.ylabel('Number of Articles')

plt.show()
```

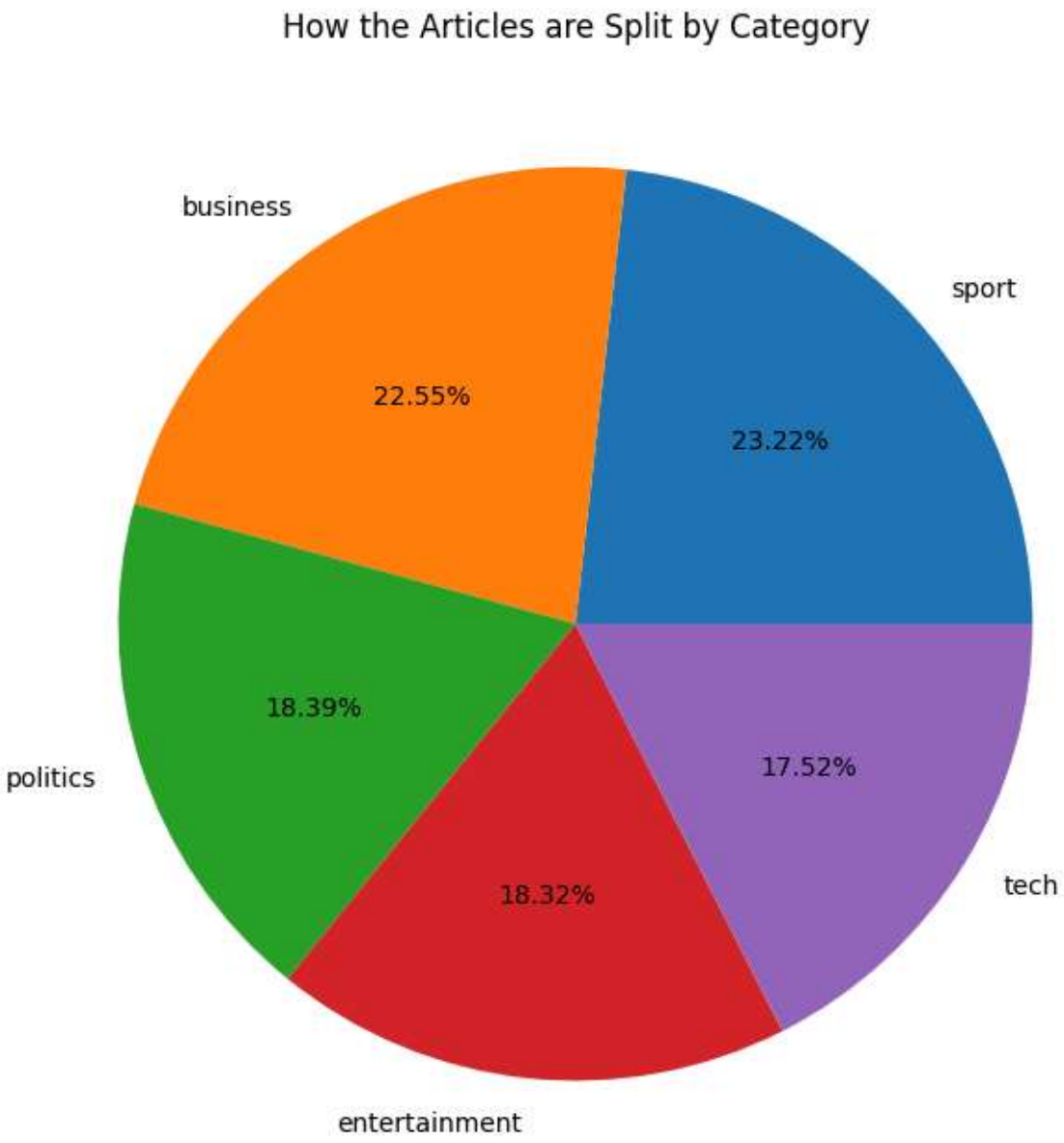


Lets see the distribution of different categories. First the total number and then a pie chart of the distribution.

In [7]:
```
category_counts = train['Category'].value_counts()
print("Number of Articles per Category:")
print(category_counts)
```

```
Number of Articles per Category:
Category
sport            346
business         336
politics         274
entertainment    273
tech             261
Name: count, dtype: int64
```

In [8]:
```
plt.figure(figsize=(8,8))
category_counts.plot.pie(autopct='%1.2f%%')
plt.title('How the Articles are Split by Category')
plt.ylabel('')
plt.show()
```



Now lets look at the most common words in each category.

In [9]:
```
print("Top 5 Words in Each Category:")
for category in train['Category'].unique():
    text = " ".join(train[train['Category'] == category]['Text']).lower().split()
    top_words = Counter(text).most_common(5)
    print("Category:", category)
    for word, count in top_words:
        print({word},":", {count})
```

```
Top 5 Words in Each Category:
Category: business
{'the'} : {7129}
{'to'} : {3293}
{'of'} : {2861}
{'in'} : {2814}
{'a'} : {2271}
Category: tech
{'the'} : {7491}
{'to'} : {4094}
{'of'} : {3410}
{'and'} : {3013}
{'a'} : {2769}
Category: politics
{'the'} : {7950}
{'to'} : {3906}
{'of'} : {2837}
{'and'} : {2552}
{'a'} : {2522}
Category: sport
{'the'} : {6610}
{'to'} : {3173}
{'a'} : {2647}
{'and'} : {2524}
{'in'} : {2485}
Category: entertainment
{'the'} : {5812}
{'and'} : {2120}
{'to'} : {2102}
{'of'} : {2046}
{'in'} : {1977}
```

Welp this wasn't useful. We need to delete common words in the english language. To do this we need to import something I found online called ENGLISH_STOP_WORDS. This library from sklearn will help filter out common words in the english language so we can get key words we care about. Also after reviewing the results I added some extras to the list to be deleting since they were caught by ENGLISH_STOP_WORDS.

In [10]:
```
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS


extra_stopwords = {'s', 'mr', '-', '.', 'said', 'new', 't', 'just', 'one', 'get', '

all_stopwords = ENGLISH_STOP_WORDS.union(extra_stopwords)

print("Top 5 Important Words in Each Category:")
```

```
for category in train['Category'].unique():
    all_text = " ".join(train[train['Category'] == category]['Text']).lower()
    words = all_text.split()

    filtered_words = []
    for word in words:
        if word not in all_stopwords:
            filtered_words.append(word)

    top_words = Counter(filtered_words).most_common(5)


    print("Category:", {category})
    for word, count in top_words:
        print(f"{word}: {count}")
```

```
Top 5 Important Words in Each Category:
Category: {'business'}
year: 302
firm: 242
company: 240
market: 235
growth: 231
Category: {'tech'}
people: 624
mobile: 311
technology: 263
users: 249
digital: 238
Category: {'politics'}
labour: 469
government: 430
blair: 372
people: 361
party: 334
Category: {'sport'}
england: 313
game: 285
win: 261
world: 248
cup: 192
Category: {'entertainment'}
film: 506
best: 404
music: 232
year: 213
number: 165
```

Now these words look so much better! After looking over them I think these make sense for each category. Not many cross over between them now which is a huge plus.

Building and Training Models

Now we need to actually build the model that will predict the correct category. We will first create an unsupervised learning model and then we will also create a supervised learning model and compare the results.

Unsupervised Learning Model

So the first thing we need to do is turn the text into numbers that the models can then use to perform analysis on. We do this with the function TfidVectorizer. This function will give scores to data based on the rarity of the word and then puts all the words into a matrix the model can learn on.

In [11]:
```
transformer = TfidfVectorizer(stop_words='english', max_features=5000)
X_train_tfidf = transformer.fit_transform(train['Text'])
X_test_tfidf = transformer.transform(test['Text'])
```

We now are going to use matrix factorization through a Non-Negative Matrix Factorization method (NMF). This method is ideal because it can break the articles down into parts of most important words used, so it can find trends.

In [12]:
```
num_topics = train['Category'].nunique()
nmf = NMF(n_components=num_topics)
W = nmf.fit_transform(X_train_tfidf)
H = nmf.components_
```

The code below will find the topic most relatvent in each article and match it to the category that relates the most. We are taking our known training data and removing its category. Then we run our prediction on all the entries and compare our result to its actual category.

In [13]:
```
train['NMF_Topic'] = np.argmax(W, axis=1)

topic_to_category = train.groupby('NMF_Topic')['Category'].agg(lambda x: x.value_co

test_W = nmf.transform(X_test_tfidf)

test_topic_assignments = np.argmax(test_W, axis=1)

predicted_nmf_categories = topic_to_category[test_topic_assignments].values
```

Below are the results of this test.

In [14]:
```
print("[Matrix Factorization (NMF) Train Results]")
print(classification_report(train['Category'], topic_to_category[train['NMF_Topic']
```

```
[Matrix Factorization (NMF) Train Results]
               precision    recall  f1-score   support

     business       0.91      0.93      0.92       336
entertainment       0.97      0.81      0.88       273
     politics       0.92      0.89      0.91       274
        sport       0.97      0.99      0.98       346
         tech       0.82      0.95      0.88       261

     accuracy                           0.92      1490
    macro avg       0.92      0.91      0.91      1490
 weighted avg       0.92      0.92      0.92      1490
```

The initial results are pretty good. The overally accuracy across all the categories is 92% which is solid! Entertainment and tech were the worst while sports and business were the highest accuracy.

Optimizing the Model

So our first iteration of the model had k=5. K is the number of topics in each article the model looks for essentially. So we originally chose 5 because we have 5 different categories to choose between, but we can expand on this and make it lets say 32 instead. You can then take 32 topics and put them inside 1 of the 5 different categories we have. Below we will adjust the value of k and see which model performs the best.

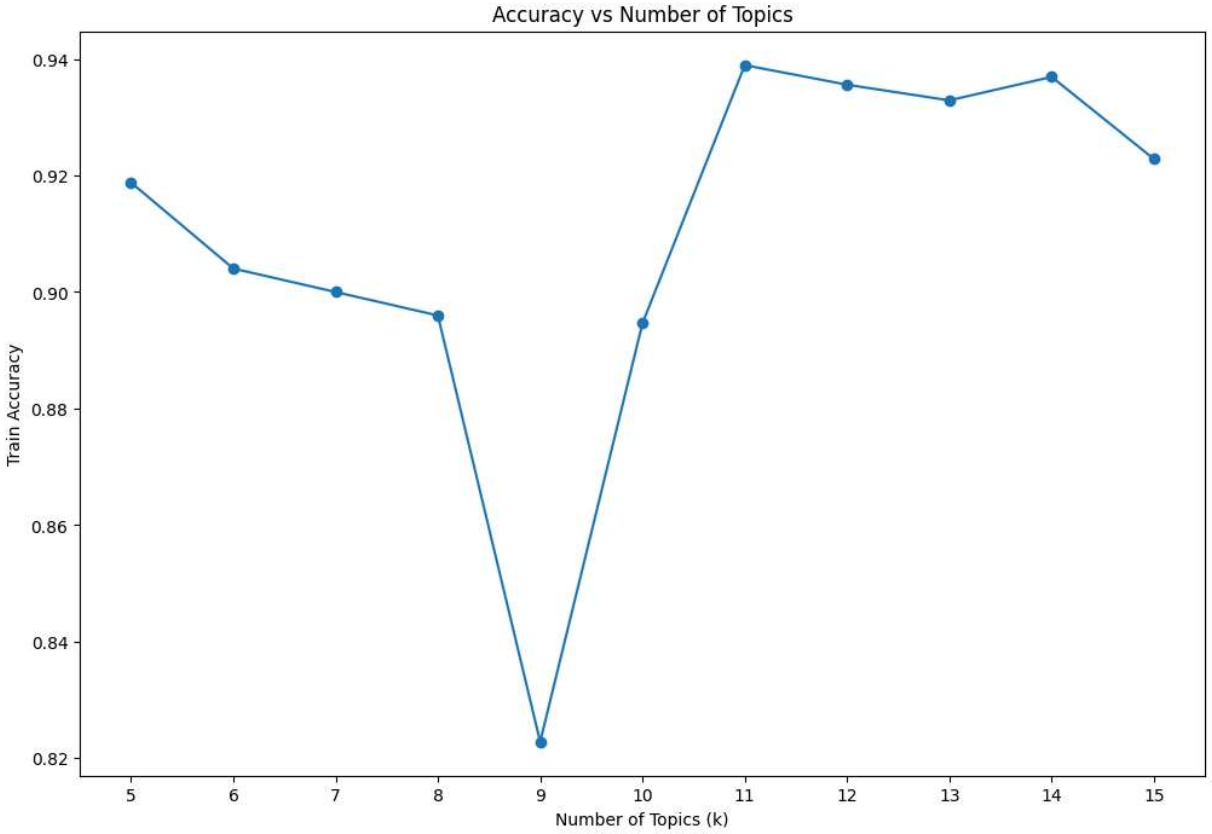
In [15]:
```
print("Trying Different Numbers of Topics:")
results = []

for k in [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]:
    nmf_k = NMF(n_components=k, random_state=42)
    W_k = nmf_k.fit_transform(X_train_tfidf)
    train['Topic_k'] = np.argmax(W_k, axis=1)
    topic_to_cat = train.groupby('Topic_k')['Category'].agg(lambda x: x.value_count
    preds = topic_to_cat[np.argmax(W_k, axis=1)].values
    acc = accuracy_score(train['Category'], preds)
    print("Tried", k, "topics; Accuracy:", round(acc, 3))
    results.append((k, acc))

best_k, best_acc = max(results, key=lambda x: x[1])
print("Best Accuracy:", round(best_acc, 3))
```

```
Trying Different Numbers of Topics:
Tried 5 topics; Accuracy: 0.919
Tried 6 topics; Accuracy: 0.904
Tried 7 topics; Accuracy: 0.9
Tried 8 topics; Accuracy: 0.896
Tried 9 topics; Accuracy: 0.823
Tried 10 topics; Accuracy: 0.895
Tried 11 topics; Accuracy: 0.939
Tried 12 topics; Accuracy: 0.936
Tried 13 topics; Accuracy: 0.933
Tried 14 topics; Accuracy: 0.937
Tried 15 topics; Accuracy: 0.923
Best Accuracy: 0.939
```

I did a few initial test of different k values ranging from 5-50 and found that 5-15 gave the best results. From this we get the best k value of 11 with an accuracy of ~94%.

Lets plot the results

In [16]:
```python
k_vals = []
acc_vals = []

for k, acc in results:
    k_vals.append(k)
    acc_vals.append(acc)


plt.figure(figsize=(12, 8))
plt.plot(k_vals, acc_vals, marker='o')
plt.xticks(k_vals)
plt.xlabel('Number of Topics (k)')
plt.ylabel('Train Accuracy')
plt.title('Accuracy vs Number of Topics')
plt.show()
```



Supervised Learning Model

We are going to use logistic regression for our supervised learning model. Logistic regression predicts categories, so its great for modeling what we need to do here.

In [17]:
```
log_model = LogisticRegression(max_iter=1000)
log_model.fit(X_train_tfidf, train['Category'])


train_lr_predictions = log_model.predict(X_train_tfidf)
test_lr_predictions = log_model.predict(X_test_tfidf)

print("Supervised Learning Train Results")
print(classification_report(train['Category'], train_lr_predictions))
```

```
Supervised Learning Train Results
               precision    recall  f1-score   support

     business       1.00      1.00      1.00       336
entertainment       1.00      1.00      1.00       273
     politics       1.00      1.00      1.00       274
        sport       1.00      1.00      1.00       346
         tech       0.99      1.00      0.99       261

     accuracy                           1.00      1490
    macro avg       1.00      1.00      1.00      1490
 weighted avg       1.00      1.00      1.00      1490
```

Updating the Supervised Model

Our model may be overfitting the data. So we are going to adjust the sampling sizes the supervised model is using to create its model.

In [18]:
```
from sklearn.model_selection import train_test_split

subset_results = []


for size in [0.1, 0.2, 0.3, 0.4, 0.5]:
    X_train_subset, _, y_train_subset, _ = train_test_split(X_train_tfidf, train['C

    # Train logistic regression on subset
    model_subset = LogisticRegression(max_iter=1000, random_state=42)
    model_subset.fit(X_train_subset, y_train_subset)

    # Predict on full train set
    y_train_preds = model_subset.predict(X_train_tfidf)
    accuracy = accuracy_score(train['Category'], y_train_preds)

    print("Training with", int(size*100), "% of data:", "Accuracy:", round(accuracy
    subset_results.append((size, accuracy))

# Find and print the best accuracy
best_size, best_accuracy = max(subset_results, key=lambda x: x[1])

print("Best Accuracy:")
print("Training with", int(best_size*100), "% of data:", "Accuracy:", round(best_ac
```

```
Training with 10 % of data: Accuracy: 0.864
Training with 20 % of data: Accuracy: 0.951
Training with 30 % of data: Accuracy: 0.97
Training with 40 % of data: Accuracy: 0.979
Training with 50 % of data: Accuracy: 0.983
Best Accuracy:
Training with 50 % of data: Accuracy: 0.983
```

In [19]:
```
print("Supervised Model Accuracy:", round(best_accuracy, 3))

print("Unsupervised Model Accuracy (k=11):", round(best_acc, 3))
```

```
Supervised Model Accuracy: 0.983
Unsupervised Model Accuracy (k=11): 0.939
```

As you can see from the results the supervised learning model had better results. This is expected when working with machine learning models. In the supervised learning model it is given the answer key from the beginning and adjust its model to the answer key, while in unsupervised learning there is no answer key for the model to go off of and the model tries to figure out the results without a guide. So overal these are great results.

In [20]:
```
submission = pd.DataFrame({
    'Id': test.index,
```

```
    'Category': test_lr_predictions
})
```

Conclusion

In this lab we created a unsupervised machine learning model that guessed the category articles fall under. The original model had an accuracy of 92%, but after more optimization it was 94%. We then compared these results to a supervised learning model which got an accuracy of 99%. The results of this comparison made sense.

In [ ]: