Detecting given news is fake or not

importing packages

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

importing the dataset

```
data = pd.read_csv("True.csv")
```

```
data.head()
```

| | title | text | subject | date | class |
|---|---|---|---|---|---|
| 0 | As U.S. budget fight looms, Republicans flip t... | WASHINGTON (Reuters) - The head of a conservat... | politicsNews | December 31, 2017 | 0 |
| 1 | U.S. military to accept transgender recruits o... | WASHINGTON (Reuters) - Transgender people will... | politicsNews | December 29, 2017 | 0 |
| 2 | Senior U.S. Republican senator: 'Let Mr. Muell... | WASHINGTON (Reuters) - The special counsel inv... | politicsNews | December 31, 2017 | 0 |

```
data.shape
```

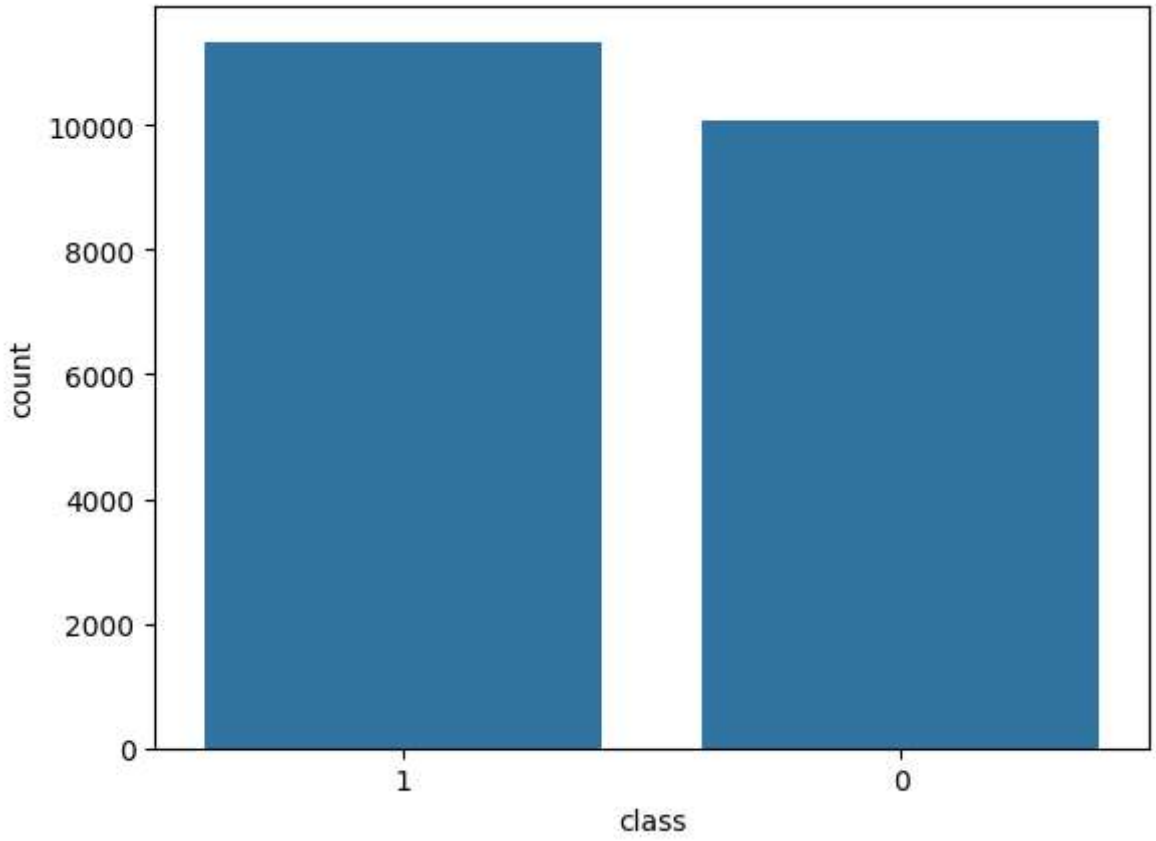
```
(21417, 5)
```

```
data = data.drop(["title", "subject","date"], axis = 1)
data.isnull().sum()
```

```
text     0
class    0
dtype: int64
```

```
data = data.sample(frac=1)
data.reset_index(inplace=True)
data.drop(["index"], axis=1, inplace=True)
sns.countplot(data=data,x='class',order=data['class'].value_counts().index)
```

```
<Axes: xlabel='class', ylabel='count'>
```




```
from tqdm import tqdm
import re
import nltk
nltk.download('punkt')
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.porter import PorterStemmer
from wordcloud import WordCloud
```

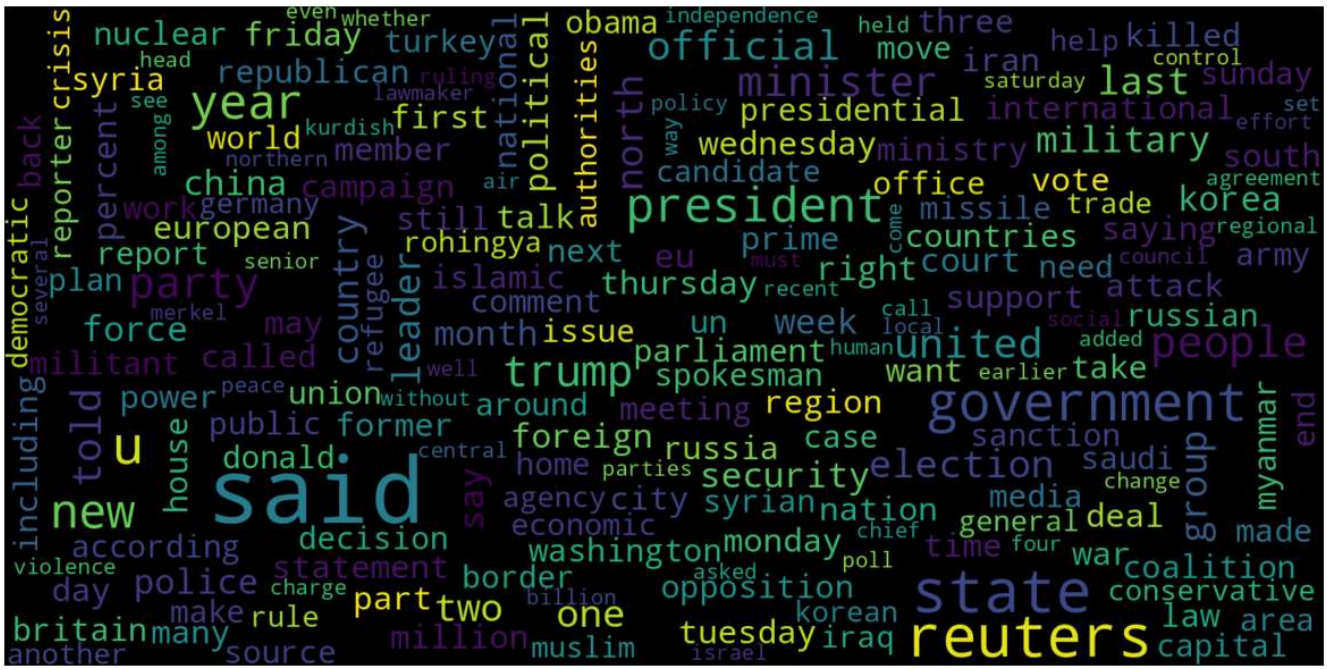
```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
def preprocess_text(text_data):
    preprocessed_text = []
    for sentence in tqdm(text_data):
        sentence = re.sub(r'[^\w\s]', '', sentence)
        preprocessed_text.append(' '.join(token.lower()
                                 for token in str(sentence).split()
                                 if token not in stopwords.words('english')))
    return preprocessed_text
```

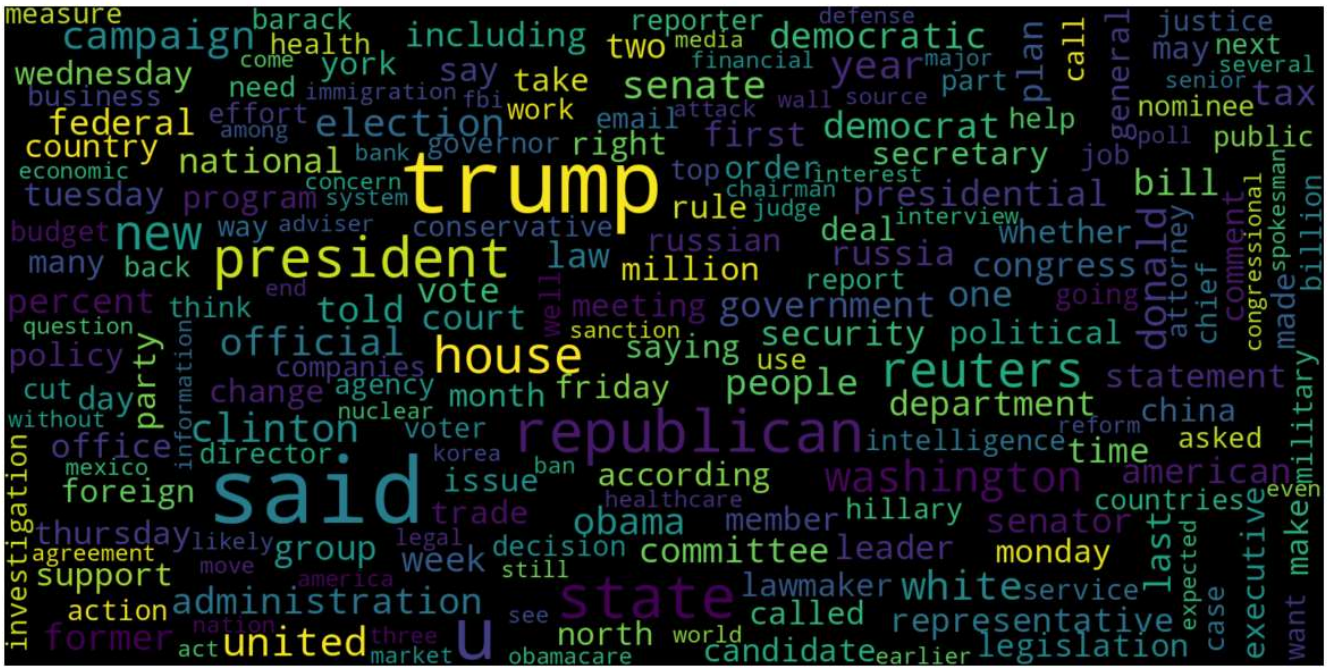
```
preprocessed_review = preprocess_text(data['text'].values)
data['text'] = preprocessed_review
```

```
 100%|██████████| 21417/21417 [17:25<00:00, 20.48it/s]
```

```
# Real
consolidated = ' '.join(
    word for word in data['text'][data['class'] == 1].astype(str))
wordCloud = WordCloud(width=1600,
                      height=800,
                      random_state=21,
                      max_font_size=110,
                      collocations=False)
plt.figure(figsize=(15, 10))
plt.imshow(wordCloud.generate(consolidated), interpolation='bilinear')
plt.axis('off')
plt.show()
```




```
# Fake
consolidated = ' '.join(word for word in data['text'][data['class'] == 0].astype(str))
wordCloud = WordCloud(width=1600,height=800,random_state=21,max_font_size=110,collocations=
plt.figure(figsize=(15, 10))
plt.imshow(wordCloud.generate(consolidated), interpolation='bilinear')
plt.axis('off')
plt.show()
```

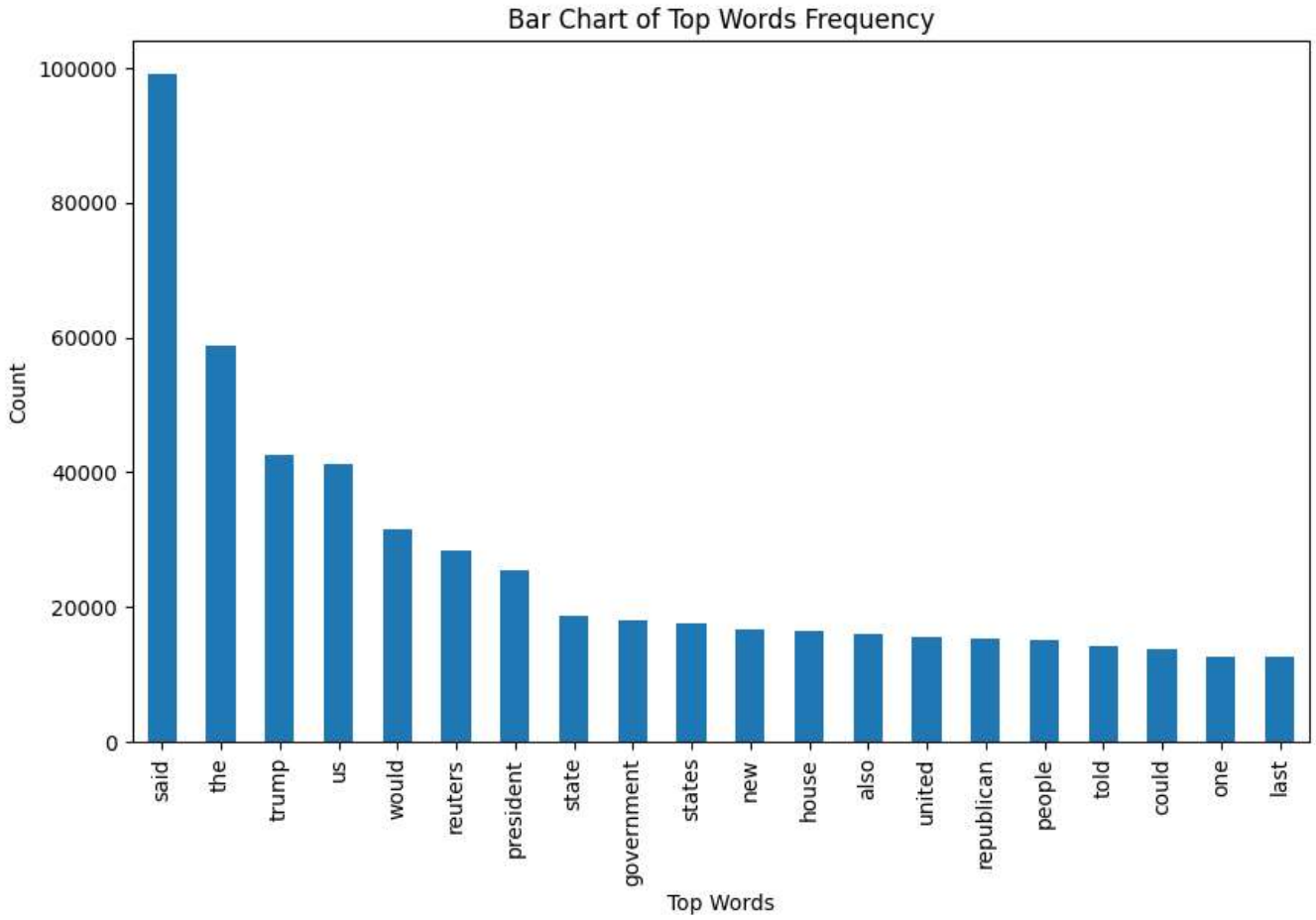



```
from sklearn.feature_extraction.text import CountVectorizer

def get_top_n_words(corpus, n=None):
    vec = CountVectorizer().fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq = sorted(words_freq, key=lambda x: x[1],reverse=True)
    return words_freq[:n]


common_words = get_top_n_words(data['text'], 20)
df1 = pd.DataFrame(common_words, columns=['Review', 'count'])
df1.groupby('Review').sum()['count'].sort_values(ascending=False).plot(kind='bar',figsize=(
```

```
<Axes: title={'center': 'Bar Chart of Top Words Frequency'}, xlabel='Top Words',
ylabel='Count'>
```




```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
x_train, x_test, y_train, y_test = train_test_split(data['text'],data['class'],test_size=0.
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorization = TfidfVectorizer()
x_train = vectorization.fit_transform(x_train)
x_test = vectorization.transform(x_test)
```

```
from sklearn.linear_model import LogisticRegression
```

```
model = LogisticRegression()
```

```
model.fit(x_train, y_train)
```

```
LogisticRegression
LogisticRegression()
```

```
# testing the model
print(accuracy_score(y_train, model.predict(x_train)))
print(accuracy_score(y_test, model.predict(x_test)))
```

```
0.9401693437928029
0.9176470588235294
```

```
from sklearn.tree import DecisionTreeClassifier
```

```
model = DecisionTreeClassifier()
```

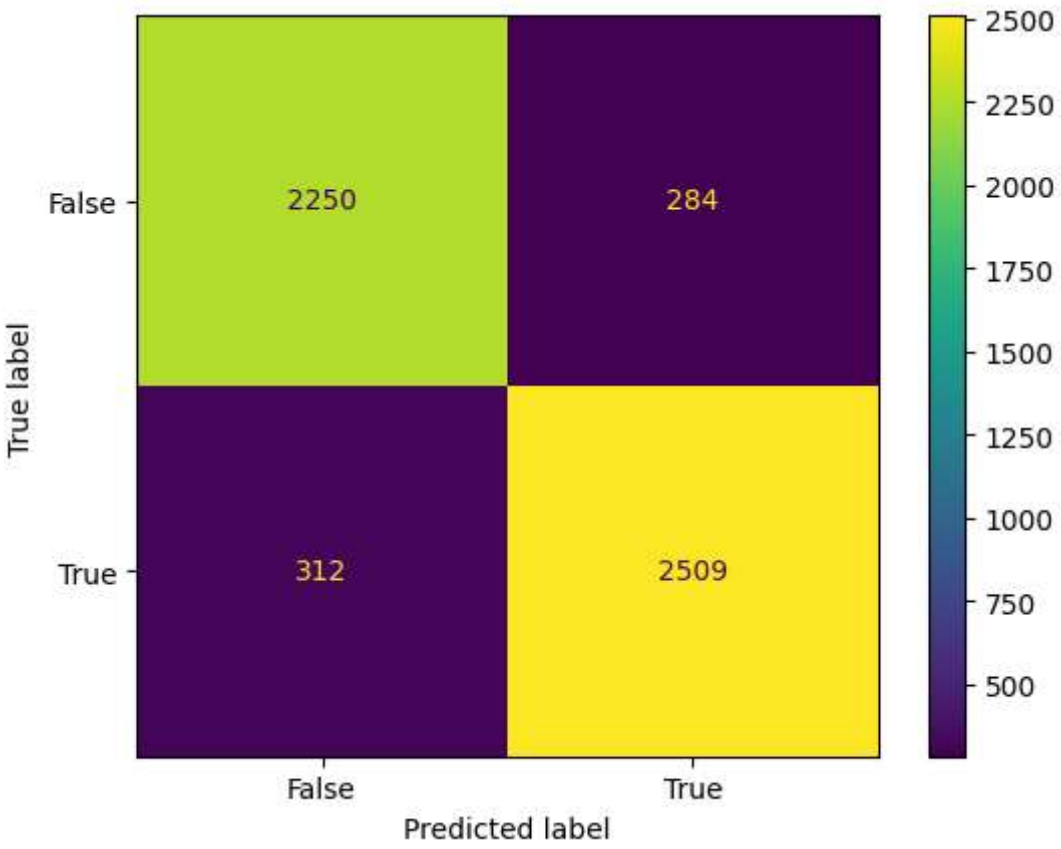
```
model.fit(x_train, y_train) # testing the model
```

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

```
print(accuracy_score(y_train, model.predict(x_train)))
print(accuracy_score(y_test, model.predict(x_test)))
```

```
0.9988793425476279
0.8887021475256769
```

```
# Confusion matrix of Results from Decision Tree classification
from sklearn import metrics
cm = metrics.confusion_matrix(y_test, model.predict(x_test))
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=[False, True
cm_display.plot()
plt.show()
```




```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score


df = pd.read_csv('True.csv')
# Concatenate 'title', 'subject', and 'date' columns into a single text column
df['text'] = df['title'] + ' ' + df['subject'] + ' ' + df['date']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df['text'], df['class'], test_size=0.2,

# Identify missing values
print(X_train.isnull().sum())
print(X_test.isnull().sum())
```

```
0
0
```

```
# Fill missing values with an empty string
X_train = X_train.fillna('')
X_test = X_test.fillna('')
X_train = X_train.astype(str)
X_test = X_test.astype(str)


# Create a TF-IDF vectorizer to convert text into numerical features
vectorizer = TfidfVectorizer()
X_train_vectors = vectorizer.fit_transform(X_train)
X_test_vectors = vectorizer.transform(X_test)



# K-Nearest Neighbors (KNN)
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_vectors, y_train)
knn_y_pred = knn.predict(X_test_vectors)
knn_accuracy = accuracy_score(y_test, knn_y_pred)
print('KNN Accuracy:', knn_accuracy)
```

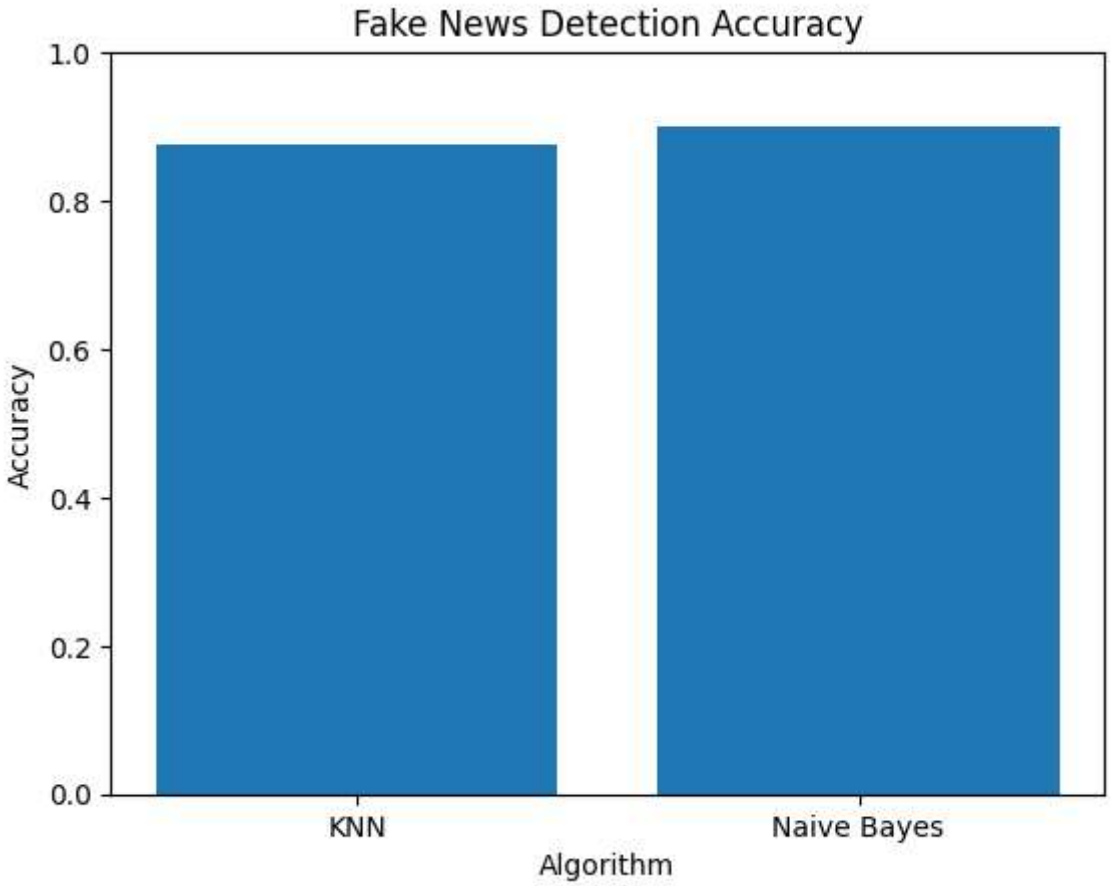
```
KNN Accuracy: 0.8751167133520075
```

```
# Naive Bayes
nb = MultinomialNB()
nb.fit(X_train_vectors, y_train)
nb_y_pred = nb.predict(X_test_vectors)
nb_accuracy = accuracy_score(y_test, nb_y_pred)
print('Naive Bayes Accuracy:', nb_accuracy)
```

```
Naive Bayes Accuracy: 0.9010270774976658
```

```
# Plot the accuracy comparison
import matplotlib.pyplot as plt

labels = ['KNN', 'Naive Bayes']
accuracy_scores = [knn_accuracy, nb_accuracy]
plt.bar(labels, accuracy_scores)
plt.ylim(0, 1)
plt.xlabel('Algorithm')
plt.ylabel('Accuracy')
plt.title('Fake News Detection Accuracy')
plt.show()
```




```
# Identify missing values
print(X_train.isnull().sum())
print(X_test.isnull().sum())
```

```
0
0
```

```
# Fill missing values with an empty string
X_train = X_train.fillna('')
X_test = X_test.fillna('')
```

```
# Identify missing values
print(X_train.isnull().sum())
print(X_test.isnull().sum())
```

```
0
0
```

```
# Fill missing values with an empty string
```

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
# Load the existing dataset (assuming it's a CSV file with 'text' and 'class' columns)
df = pd.read_csv('True.csv')


# Check if the DataFrame contains the expected columns
if 'text' in df.columns and 'class' in df.columns:
  # Create an instance of TfidfVectorizer vectorizer = TfidfVectorizer()

  # Transform the text data using TfidfVectorizer
  X = vectorizer.fit_transform(df['text'])
  y = df['class']
  # Print the shape of X and the unique classes in y
  print('X shape:', X.shape)
  print('Unique classes in y:', y.unique())
else:
  print('The DataFrame does not contain the required columns.')
```

```
X shape: (21417, 66663)
Unique classes in y: [0 1]
```

```
# Vectorize the text data
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(df['text'])
y = df['class']


# Train the models
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X, y)
nb = MultinomialNB()
nb.fit(X, y)


# Example new data
new_data = [{'title': 'How McConnell kept Republicans in line to win Senate tax bill', 'sub
'title': 'Study Finds Link Between Vaccines and Autism', 'subject': 'Health', 'date': '2023


# Preprocess the new data and convert it to TF-IDF vectors
new_data_text = [data['title'] + ' ' + data['subject'] + ' ' + data['date'] for data in new
new_data_vectors = vectorizer.transform(new_data_text)


# Predict the labels using KNN
knn_predictions = knn.predict(new_data_vectors)
# Print the predictions
for i in range(len(new_data)):
```