IMPORTING LIBRARIES FROM PYTHON

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn import feature_extraction, linear_model, model_selection, preprocessing
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
```

READING DATA

In [ ]:

```
fake = pd.read_csv("Fake.csv")
true = pd.read_csv("True.csv")
```

In [ ]:

```
print(fake)
print(true)
```

In [ ]:

```
fake.shape
true.shape
```

In [ ]:

```
fake.ndim
true.ndim
```

DATA CLEANING AND PREPARATION

In [ ]:

```
# Add flag to track fake and real

fake['target'] = 'fake'
true['target'] = 'true'
```

In [ ]:

```
# Concatenate dataframes

data = pd.concat([fake, true]).reset_index(drop = True)
data.shape
```

In [ ]:

```
print(data)
```

In [ ]:

```
# Shuffle the data

from sklearn.utils import shuffle
data = shuffle(data)
data = data.reset_index(drop=True)
```

In [ ]:

```
# Check the data

data.head()
```

In [ ]:

```
# Removing the date (we won't use it for the analysis)

data.drop(["date"],axis=1,inplace=True)
data.head()
```

In [ ]:

```
# Removing the title (we will only use the text)

data.drop(["title"],axis=1,inplace=True)
data.head()
```

In [ ]:

```
# Convert to lowercase

data['text'] = data['text'].apply(lambda x: x.lower())
data.head()
```

In [ ]:

```
# Remove punctuation

import string

def punctuation_removal(text):
    all_list = [char for char in text if char not in string.punctuation]
    clean_str = ''.join(all_list)
    return clean_str

data['text'] = data['text'].apply(punctuation_removal)
```

In [ ]:

```
# Check

data.head()
```

In [ ]:

```
# Removing stopwords
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stop = stopwords.words('english')

data['text'] = data['text'].apply(lambda x: ' '.join([word for word in x.split() if wor
```

In [ ]:

```
data.head()
```

BASIC DATA EXPLORATION

In [ ]:

```
# How many articles per subject?

print(data.groupby(['subject'])['text'].count())
data.groupby(['subject'])['text'].count().plot(kind="bar")
plt.show()
```

In [ ]:

```
# How many fake and real articles?

print(data.groupby(['target'])['text'].count())
data.groupby(['target'])['text'].count().plot(kind="bar")
plt.show()
```

In [ ]:

```
# Word cloud for fake news
from wordcloud import WordCloud

fake_data = data[data["target"] == "fake"]
all_words = ' '.join([text for text in fake_data.text])

wordcloud = WordCloud(width= 800, height= 500,
                          max_font_size = 110,
                          collocations = False).generate(all_words)

plt.figure(figsize=(10,7))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In [ ]:

```
# Word cloud for real news

from wordcloud import WordCloud

real_data = data[data["target"] == "true"]
all_words = ' '.join([text for text in fake_data.text])

wordcloud = WordCloud(width= 800, height= 500,
                          max_font_size = 110,
                          collocations = False).generate(all_words)

plt.figure(figsize=(10,7))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In [ ]:

```
# Most frequent words counter (Code adapted from https://www.kaggle.com/rodolfoluna/fake

from nltk import tokenize

token_space = tokenize.WhitespaceTokenizer()

def counter(text, column_text, quantity):
    all_words = ' '.join([text for text in text[column_text]])
    token_phrase = token_space.tokenize(all_words)
    frequency = nltk.FreqDist(token_phrase)
    df_frequency = pd.DataFrame({"Word": list(frequency.keys()),
                                   "Frequency": list(frequency.values())})
    df_frequency = df_frequency.nlargest(columns = "Frequency", n = quantity)
    plt.figure(figsize=(12,8))
    ax = sns.barplot(data = df_frequency, x = "Word", y = "Frequency", color = 'blue')
    ax.set(ylabel = "Count")
    plt.xticks(rotation='vertical')
    plt.show()
```

In [ ]:

```
# Most frequent words in fake news

counter(data[data["target"] == "fake"], "text", 20)
```

In [ ]:

```
# Most frequent words in real news

counter(data[data["target"] == "true"], "text", 20)
```

MODELLING

In [ ]:

```
# Function to plot the confusion matrix (code from https://scikit-learn.org/stable/auto_

from sklearn import metrics
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

PREPARING THE DATA

In [ ]:

```
# Split the data

X_train,X_test,y_train,y_test = train_test_split(data['text'], data.target, test_size=0
```

LOGISTIC REGRESSION

In [ ]:

```
# Vectorizing and applying TF-IDF

from sklearn.linear_model import LogisticRegression

pipe = Pipeline([('vect', CountVectorizer()),
                 ('tfidf', TfidfTransformer()),
                 ('model', LogisticRegression())])

# Fitting the model
model = pipe.fit(X_train, y_train)

# Accuracy
prediction = model.predict(X_test)
print("accuracy: {}%".format(round(accuracy_score(y_test, prediction)*100,2)))
```

In [ ]:

```
cm = metrics.confusion_matrix(y_test, prediction)
plot_confusion_matrix(cm, classes=['Fake', 'Real'])
```

DECESION TREE CLASSIFIER

In [ ]:

```
from sklearn.tree import DecisionTreeClassifier

# Vectorizing and applying TF-IDF
pipe = Pipeline([('vect', CountVectorizer()),
                 ('tfidf', TfidfTransformer()),
                 ('model', DecisionTreeClassifier(criterion= 'entropy',
                                                  max_depth = 20,
                                                  splitter='best',
                                                  random_state=42))])
# Fitting the model
model = pipe.fit(X_train, y_train)

# Accuracy
prediction = model.predict(X_test)
print("accuracy: {}%".format(round(accuracy_score(y_test, prediction)*100,2)))
```

In [ ]:

```
cm = metrics.confusion_matrix(y_test, prediction)
plot_confusion_matrix(cm, classes=['Fake', 'Real'])
```

RANDOM FOREST CLASSIFIER

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier

pipe = Pipeline([('vect', CountVectorizer()),
                 ('tfidf', TfidfTransformer()),
                 ('model', RandomForestClassifier(n_estimators=50, criterion="entropy"))

model = pipe.fit(X_train, y_train)
prediction = model.predict(X_test)
print("accuracy: {}%".format(round(accuracy_score(y_test, prediction)*100,2)))
```

In [ ]:

```
cm = metrics.confusion_matrix(y_test, prediction)
plot_confusion_matrix(cm, classes=['Fake', 'Real'])
```