Name	: Abdul Saboor Hamedi
Student Number	:02222120008
Title of the thesis	:Exploring People's Perceptions to the Ban on Women's Education in Afghanistan
Question	:To what extent does gender affect people's perception of women's access to education
Hypothesis	:Women are more likely to disagree with the ban on woman's education
Supervisor I	:Sirojudin Arif, Ph.D
Supervisor II	: Nia Deliana, Ph.D

1.0 INTRODUCTION 3
2.0 DATASET 3
2.2 Fetch Tweets by Username 4
2.3 Fetch Tweets with keywords 4
2.4 Divide Tweets 6
3.0 PRE-PROCESS DATA 6
3.1 Remove URL, and Mentions 7
3.2 Extract Hashtags 7
3.4 Remove Hashtags 8
4.5 Top Ten Hashtags 9
4.6 Top Ten Hashtags Charts 9
3.3 Expand Contractions 10
3.4 Remove ASCII Characters: 11
3.5 Remove Whitespace 12
3.6 Drop rows 12
3.7 Remove Punctuations 13
4.0 MACHINE LEARNING 14
4.1 Create A Contingency 14
4.2 Number of from Each Gender and Label Category 16
4.3 Largest Tweets 17
5.0 VECTORIZATION 18
5.1 The output CountVectorizer 19
5.2 Train Logistic 20
5.3 Train Model 21
5.4 Classification 21
5.5 Classification Report 22
CONCLUSION 23

1.0 Introduction

The purpose of this study was to see if women are more likely to disagree with a ban on women's education. To do so, we assessed a dataset of articles about women's education and determined whether the articles supported or opposed a ban on women's education. We then utilized machine learning techniques to classify the articles based on their content and assessed the model's accuracy. For a long time, people in Afghanistan have argued about whether or not women should be allowed to attend school. The 2021 Taliban takeover of Afghanistan has brought it back into the spotlight. Protests have broken out inside and outside of Afghanistan on concerns that the Taliban would reintroduce their ban on women receiving an education. In this paper, we analyze how people in Afghanistan feel about the restriction on women's education, paying special attention to how gender plays a role in this debate. We expect more women to disagree with the prohibition on women's education than men do

2.0 Dataset

This study's dataset is made up of tweets about women's education gathered from various Afghan ethnicity. The tweets were classified as "positive" and "neutral" based on their substance. The "positive" tweets supported on women's education, whereas the "neural" pieces do not talk about education or ban.

2.1 Data Collection

On December 1, 2022, through March 1, 2023, we collected tweets for three distinct time periods. Pashtun, Tajik, Hazara, and Uzbek are the four main ethnic groups in Afghanistan, and their tweets were compiled. Keywords such as "education," "women," "schools," "universities," "ban," "girls," "teachers," and "learning" were used to compile a total of 179,121 tweets. To pick up tweets in Persian and Pashtu, we also used keywords from those languages.

After eliminating irrelevant duplicates, nulls, and blanks, we were left with 59,606 tweets. The tweets were then categorized according to whether they were pro- or anti-women, pro- or anti-school, pro- or anti-university, or pro- or anti-ban. The outcome showed that 93.89 percent of tweets were positive, with 6.11 percent being neutral.

We also used crosstab analysis to look at the data from a gender and ethnicity perspective. From 93.34 percent to 97.53 percent, the crosstab indicated that women of all racial and ethnic backgrounds were in favor of women's education. Men of all racial and ethnic backgrounds were found to be 91.83–94.17% less likely to express support for women's education.

2.2 Fetch Tweets by Username

```
import tweepy
import pandas as pd
# Twitter API credentials
consumer_key = 'your_consumer_key'
consumer_secret = 'your_consumer_secret'
access_token = 'your_access_token'
access_token_secret = 'your_access_token_secret'

# Authenticate to Twitter API
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)

# Create API object
api = tweepy.API(auth)

# Define the username
username = 'twitter_username'

# Fetch the user's tweets
tweets = api.user_timeline(screen_name=username, count=100)

# Create a list of dictionaries containing the tweets
tweets_list = []
for tweet in tweets:
    tweets_list.append({
        'created_at': tweet.created_at,
        'text': tweet.text,
        'retweets': tweet.retweet_count,
        'favorites': tweet.favorite_count
    })

# Create a Pandas dataframe from the list of tweets
df = pd.DataFrame(tweets_list)

# Print the dataframe
print(df.head())
```

With the code above we can fetch the whole data from this certain account, the total tweets that fetched is 3000. This the first method, which basically fetch all tweets for us.

2.3 Fetch Tweets with keywords

```
import tweepy
import re
import pandas as pd

```

```
keywords = ["education", "women", "woman",
            "schools", "school", "univesities",
            "university", "closed","close",
            "ban", "banned", "from school",
            "girls", "girl","work"
             "worked","edu","teachers", "teacher","learning",
             "آموزش", "زنان" "زن", "مدارس", "مدرسه", "دانشگاه ها",
             "دانشگاه", "بسته", "بسته",
             "ممنوعیت", "ممنوع",
             "از مدرسه", "دختران", "دختر", "کار",
             "کار کرد", "آموزش", "معلمان", "معلم", "یادگیری"]
# Authenticate your API key
consumer_key= "dqOPlo8TYO2Bo6P4Uo5sAkmes"
consumer_secret= "3TUe2RlqY5BT6LnvtUAA0Kf1mPO8k7MH3ZLxxELcsj7OxG8doh"
access_token= "2585433478-GoThRI4yGSKJX3tO8GX2onBsVdZ9EyPP1uL7dSi"
access_token_secret= "WBHrZjnz8lDrKAlLpF0uNKMydws4iodCuWUtHwdqmYoq1"

auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth)
# Construct a query to fetch tweets from "@user"
# Construct a query to fetch tweets related to education in English
tweets = []
for tweet in tweepy.Cursor(api.user_timeline, screen_name="@baharmhr", count=3000).items():
    # Filter out tweets that are not related to any of the keywords
    if any(keyword in tweet.text.lower() for keyword in keywords):
        tweets.append(tweet)

# Create a list of dictionaries containing the relevant tweet data
tweet_data = []
for tweet in tweets:
    tweet_dict = {
        "text": tweet.text,
        "name": tweet.user.screen_name,
        "created_at": tweet.created_at.strftime("%Y-%m-%d %H:%M:%S")
    }
    tweet_data.append(tweet_dict)
# Create a Pandas DataFrame from the tweet data
df = pd.DataFrame(tweet_data)
```

Here is the second method, this is more precise, even though it's the same as the previous, but with this code we are able to fetch certain tweets. We go the user's profile, and collect their tweets based on keywords. Here are some keywords ["education", "women", "woman", "schools", "school", "universities", "university", "closed", "close", "ban", "banned", "from school", "girls", "girl", "work"], we search with multiple keywords, this helps us to collect those tweets which we needed. I have added some Persian keywords as well, because of the Afghan people tweets with Persian language.

2.4 Divide Tweets

```
import pandas as pd
# Read the tweets data into a DataFrame
tweets_df = pd.read_excel('../data_processed/complete.xlsx')
# Drop rows containing NaN values
tweets_df = tweets_df.dropna()
# Convert the text column to lowercase
tweets_df['text'] = tweets_df['text'].str.lower()
# Define the list of keywords
keywords = ["women", "education", "schools", "study", "learn", "right", "ban", "people",
            "afghan", "educated", "girls", "female", "protest", "pashtun", "tajik", "uzbek", "hazara",
            "rights", "prohibit", "patriarch", "close schools", "close universities", "university", "man"]
# Create an empty dictionary to store the matched tweets for each keyword
matched_tweets = {}
# Iterate over the keywords and find the matched tweets
for keyword in keywords:
    # Replace NaN values with an empty string
    tweets_df['text'] = tweets_df['text'].fillna('')
        # Filter the tweets containing the keyword
    matched_tweets[keyword] = tweets_df[tweets_df['text'].str.contains(keyword, case=False)]

# Print the number of matched tweets for each keyword
for keyword, df in matched_tweets.items():
    print(f"{keyword}: {len(df)}")

# Save each matched DataFrame to a separate CSV file
for keyword, df in matched_tweets.items():
    df.to_excel(f"../data_processed/{keyword}_matched_tweets.xlsx", index=False)
```

After merging all tweets in one single excel sheet, we have decided to find the most relevant tweets. There are two different ways to do that, first to label the data second to divide our data with keywords. We have implemented both methods, but first we have divided the data with keywords, because this gives us more opportunity to be precise with the data, as the dataset become smaller. The following are the keywords that used on our dataset in order to get most relevant data. Keywords: ['education', 'educate', 'educational', 'school', 'university', 'women']. By implementing this method, we have achieved a total tweet of 87.000.

3.0 Pre-process Data

Data pre-processing is an important stage in any natural language processing operation. It entails cleaning and translating raw text data into a format that machine learning models can easily understand. The following steps were included in the pre-processing of Twitter data:

3.1 Remove URL, and Mentions

With code below we are able to remove all URLs, and mentions from the dataset, our dataset has bunch mentions and URLs. We import the regular expression library re in this example, then build a sample tweet with a URL, a mention, and some hashtags. The actual tweet is printed. Following that, we use the re.sub() function to replace any URL that begins with http and any mention that begins with @ with an empty string, thereby deleting them from the tweet. Lastly, we publish the processed tweet, which is devoid of URLs and mentions.

```
#remove and mentions
def remove_usernames_links(tweet):
    tweet = re.sub("@[^\s]+","",str(tweet))
    tweet = re.sub("http[^\s]+","",str(tweet))
    # tweet = re.sub(r"[RT]+","",str(tweet))
    return tweet
df = df.replace(r"\n"," ", regex=True)
df["text"] = df["text"].apply(remove_usernames_links)

```

3.2 Extract Hashtags

```
from collections import Counter
hashtags = []
for row in df['text']:
    hashtags.extend([tag.strip("#") for tag in row.split() if tag.startswith("#")])
counts = Counter(hashtags)
top_tags = counts.most_common(10)  # modify topn parameter here
fig, ax = plt.subplots(figsize=(12, 8))
ax.bar([tag[0] for tag in top_tags], [len(tag[0]) for tag in top_tags])
ax.set_title("Top 50 Hashtags with Lengths")
ax.set_xlabel("Hashtag")
ax.set_ylabel("Length")
plt.xticks(rotation=90)
plt.show()

```

This code uses regular expression to extract hashtags from a DataFrame's ‘tweet’ column and stores them in a 'hashtags' list. The 'Counter' module from the 'collections' library is then used to tally the frequency of each hashtag. The top ten most often hashtags are extracted and saved in the 'top hashtags' list using the 'most_common()' method of the 'Counter' class. Finally, the code outputs the top ten hashtags and their counts as named constants, with the hashtag transformed to title case by eliminating the '#' symbol.

Hashtag	Count
#Afghanistan	5224
#taliban	3457
#rabieh_sadata	1619
#afghanistansupporttaliban	1033
#kabul	929
#unitedafghanistan	829
#stophazaragenocide	757
#afghan	549
#afghanwomen	509
#pakistan	483

3.4 Remove Hashtags

The code below would remove whole hashtags symbols from the dataset, we do this after talking the most trends or hashtags in our dataset

```
import pandas as pd
import re
# Define a regular expression to match hashtags
pattern = r'#\w+'
# Extract all hashtags from the 'text' column and store them in a new column 'hashtags'
df['hashtags'] = df['text'].str.extractall(pattern)[0].values
# Print the resulting dataframe
 print(df)
```

The table below shows, we have removed the entire hashtags from the dataset using the above cove.

Hashtag	Count
Afghanistan	5224
taliban	3457
rabieh_sadata	1619
afghanistansupporttaliban	1033
kabul	929
unitedafghanistan	829
stophazaragenocide	757
afghan	549
afghanwomen	509
pakistan	483

4.5 Top Ten Hashtags

With the code below we are able to get most top ten hashtags in our dataset or in another we called them the trend. Twitter is a place where user mostly starts their tweets by adding a certain word with symbol hashtags. And its important for us to check what is the most top trend in our dataset.

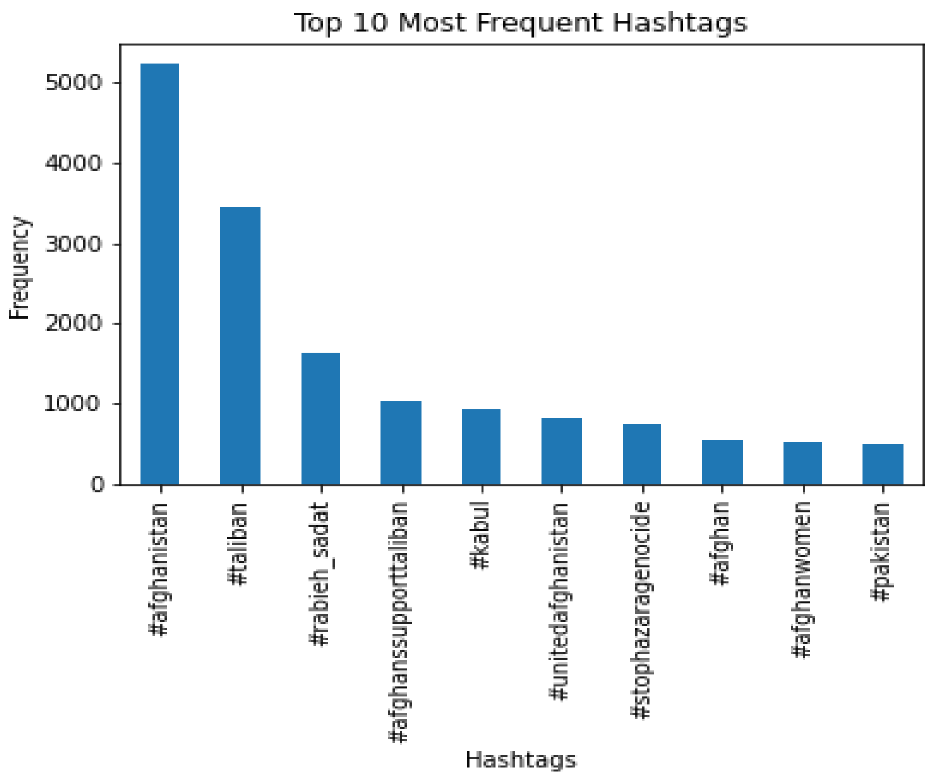
```
import pandas as pd
import re
# Extract hashtags using regular expression
hashtags = df['text'].apply(lambda x: re.findall(r'#\w+', x))
# Flatten the list of hashtags
hashtags = [tag for sublist in hashtags for tag in sublist]
# Count the frequency of each hashtag
freq = pd.Series(hashtags).value_counts()
# Select the top N most frequent hashtags
top_n = 10
top_hashtags = freq.head(top_n)
# Print the top N most frequent hashtags
print(top_hashtags)
```

As a result, the table below shows our top trend in our dataset is Afghanistan, followed by the Taliban.

Hashtag	Count
#Afghanistan	5224
#taliban	3457
#rabieh_sadata	1619
#afghanistansupporttaliban	1033
#kabul	929
#unitedafghanistan	829
#stophazaragenocide	757
#afghan	549
#afghanwomen	509
#pakistan	483

4.6 Top Ten Hashtags Charts

The chart below shows the top most frequent or top ten hashtags in our dataset, chart help as to see the trend in a graph.



3.3 Expand Contractions

This code expands contractions in a Pandas DataFrame column named 'text' to do text pre-processing. A list of commonly used contractions and their expanded forms can be found in the contraction map dictionary. The function expand contractions searches for contractions in a text input using a regular expression pattern. When a contraction appears in the text, it is substituted with its extended form. The select dtypes method in Pandas is used to select columns of type 'object,' which often contain textual data. The apply method is used to apply the expand contractions function to all of the DataFrame's text columns. Finally, the first 5 rows of the pre-processed DataFrame are displayed using the head technique.

```
import pandas as pd
contraction_map={
    "ain't": "is not",
    "aren't": "are not",
    "can't": "cannot",
}
# define a function to apply the contraction map
def expand_contractions(text):
    pattern = re.compile('({})'.format('|'.join(contraction_map.keys())), 
flags=re.IGNORECASE|re.DOTALL)
    def replace(match):
        return contraction_map[match.group(0).lower()]
    return pattern.sub(replace, text)

# apply the expand_contractions function to text columns only
text_cols = data.select_dtypes(include=[object]).columns
data['text'] = data['text'].apply(lambda x:expand_contractions(str(x)))
data.head(5)

```

As a result, the table below is the output of the above code, we can see that the contraction has fixed.

No.	Contraction	Fixed
1	ain’t	Is not
2	aren’t	Are not
3	can’t	Cannot

3.4 Remove ASCII Characters:

A character encoding standard for electronic communication is called ASCII (American Standard Code for Information Interchange). It was created for the first time in the 1960s and is still in use today. ASCII can represent up to 128 different characters because it employs a 7-bit code to encode its characters. These symbols contain capital and lowercase letters, numbers, punctuation, control, and a few unusual symbols. Computers and other electrical devices use ASCII to represent text. Because each character is represented by a distinct code, text may be processed and displayed by computers in a uniformed manner. Over time, the ASCII standard has grown to accommodate more characters, including symbols and those from foreign languages.

```
import re
sample_string = 'This is a sample string with ASCII characters: !@#$%^&*()_+={}[]|\:;"<>,.?/`~'
clean_string = re.sub(r'[^\x00-\x7F]+', '', sample_string)
# Print the original string and the cleaned string
print('Original string:', sample_string)
print('Cleaned string:', clean_string)
```

As a result, the table below shows that the code above has successfully run and remove ASCII character our dataset.

No.	**ASCII**	**Removed**
1	This is a sample string with ASCII characters: !@#$%^&*()_+={}[]|\:;"<>,.?/`~	This is a sample string with ASCII characters

3.5 Remove Whitespace

The whitespace function is used on the data DataFrame in this example to remove any empty strings and leading/trailing whitespace in the 'name' and 'tweet' columns. The modified DataFrame is then printed to demonstrate the changes. The following is the output of the above code:

```
def whitespace(columns_name, text):
    data[columns_name] = data[columns_name].replace('', '', regex=True) #remove empty string
    data[columns_name] = data[columns_name].str.strip() #remove whitespace
whitespace('name',data)
whitespace('text',data)
```

As a result, the table below show all white from paragraph has removed.

No	White space	Remove
1	I love Pizza	I love Pizza
2	Its fun to analyze data	Its fun to analyze data

3.6 Drop rows

Most of the dataset contains irrelevant data, null values, during analyzing the data if we do not remove or drop them, we would encounter error. That Makes the analysis hard, its always good practice to remove all those null values during the data cleansing. The code below shows how to drop the null values from the dataset.

```
# drop empty rows from the 'text' column
df.dropna(subset=['text'], inplace=True)
# Remove rows with NaN values
df.dropna(inplace=True)
#remove duplicate
df.drop_duplicates(subset=['text'], inplace=True)
df[['text']].sample(5)
```

3.7 Remove Punctuations

Many symbols used in written language to fix and make clear the various sections of a sentence or phrase are referred to as punctuation in a dataset. These can include signs like dashes, commas, periods, question marks, exclamation points, semicolons, colons, parentheses, and more.

```
import string
def remove_punctuations(text):
    for punctuation in string.punctuation:
        text = text.replace(punctuation, '')
    return text
# Apply to the DF series
df['text'] = df['text'].apply(remove_punctuations)

```

As a result, the table below shows that all punctuation has remove from the dataset. Punctuations are not important to keep them in the dataset and it should be removed all.

No	Paragraph (with punctuation)	Paragraph (without punctuation)
1	"Hello, how are you?"	Hello how are you
2	"I love to read books!"	I love to read books
3	"It's a beautiful day today."	Its a beautiful day today
4	"The cat chased the mouse."	The cat chased the mouse

To sum up, data cleaning is a crucial phase in the data analysis process, and using pandas to clean and preprocess data can be an efficient and effective way to make sure that datasets are prepared for analysis. Data analysts can find and fix missing or inconsistent values, get rid of duplicates, deal with outliers and errors, and change variables to better suit their analysis needs by using pandas functions and methods. Overall, cleaning data with pandas entails a number of stages, including importing data into a panda DataFrame, evaluating the data's quality and completeness, locating and correcting missing or inconsistent values, and transforming and displaying the data to fit analysis needs. Analysts may make sure their data is correct, dependable, and prepared for in-depth analysis and interpretation by following these procedures and utilizing the potent data cleaning tools offered by pandas.

4.0 Machine Learning

Python is a preferred programming language because of its extensive capabilities, applicability, and simplicity. Due to its independent platform and widespread use in the programming community, the Python programming language is the most suitable for machine learning. A component of Artificial Intelligence (AI) called machine learning tries to make a machine learn from experience and carry out tasks automatically without necessarily having to be programmed to do so. Contrarily, Artificial Intelligence (AI) is a more general term for machine learning in which computers are made to be sensitive to the human level by perceiving visually, by speaking, by language translation, and thereafter making important decisions.

4.1 Create A Contingency

The crosstab function is used in this code to compute and print a contingency table. For each combination of two categorical variables, ethnicity and gender, the contingency table displays the frequency distribution of a categorical variable, label.

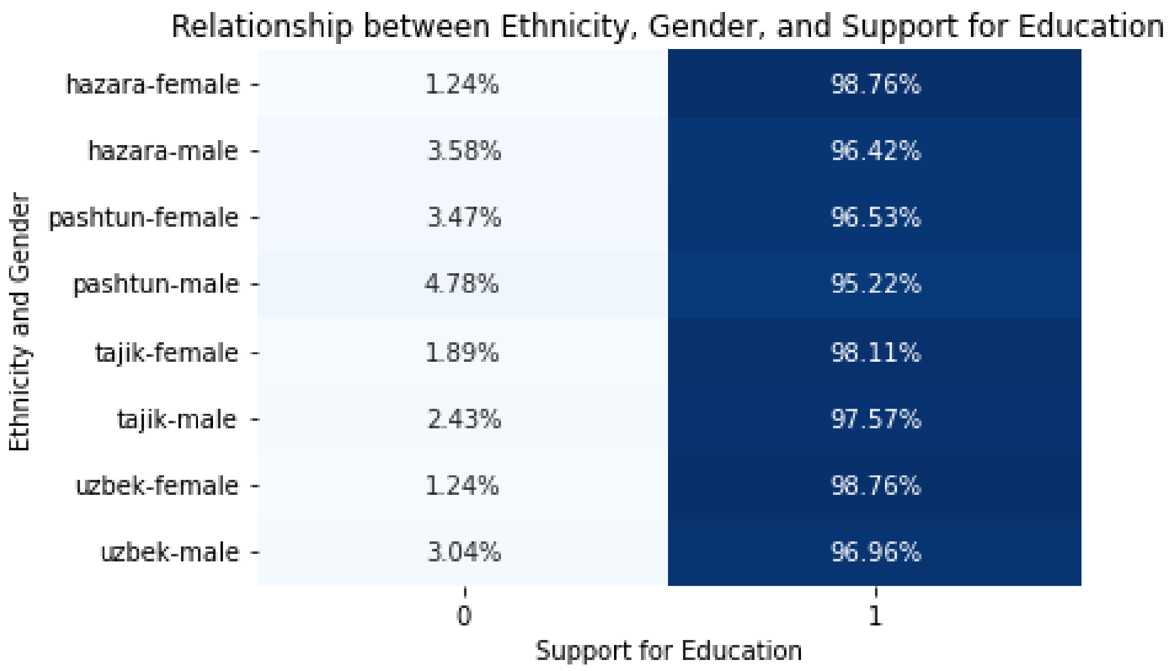
```
# create the contingency table
ct = pd.crosstab([df['ethnic'], df['gender']], df['label'], normalize='index')
# print the contingency table
print(ct)

```

The percentage of neutral and favor of education tweets across various racial and gender categories is displayed in the table above. The Hazara, Pashtun, Tajik, and Uzbek ethnic groups are represented, and the genders are either male or female. The percentage of tweets in favor of education is larger than the percentage of neutral tweets across all racial and ethnic groupings, it can be seen. In addition, women across all ethnic groups tweet more frequently in favor of education than men do. When examining each ethnic group independently, Pashtuns have the fewest tweets in favor of education among females, while Hazaras have the fewest in favor of education among males. On the other side, Tajiks, both male and female, have the greatest proportion of tweets in support of education.

The percentage of tweets in favor of education and neutral tweets differs just slightly among males, with Tajik males having the highest percentage and Pashtun males having the lowest, at 0.975704 and 0.952220, respectively. The disparity is greater among women, with Uzbek women scoring the highest (0.987637) and Pashtun women scoring the lowest (0.965321).

	label	0	1
ethnic	gender		
hazara	female	0.0124	0.9876
	male	0.0358	0.9642
pashtun	female	0.0347	0.9653
	male	0.0478	0.9522
tajik	female	0.0189	0.9811
	male	0.0243	0.9757
uzbek	female	0.0124	0.9876
	male	0.0304	0.9696

In conclusion, the data shows that across all ethnic groups, women are more likely than men to tweet in favor of education. In addition, Tajiks are more likely than Pashtuns and Hazaras to tweet in support of education, with Pashtuns having the lowest percentage among females and Hazaras having the lowest rate among males. It's important to note that the study is predicated on the premise that the tweets are typical of the population of each gender and ethnic group. The context of the tweets, which may influence the attitude expressed, is also not taken into account in the research. These factors should be explored in more detail in future studies to learn more.



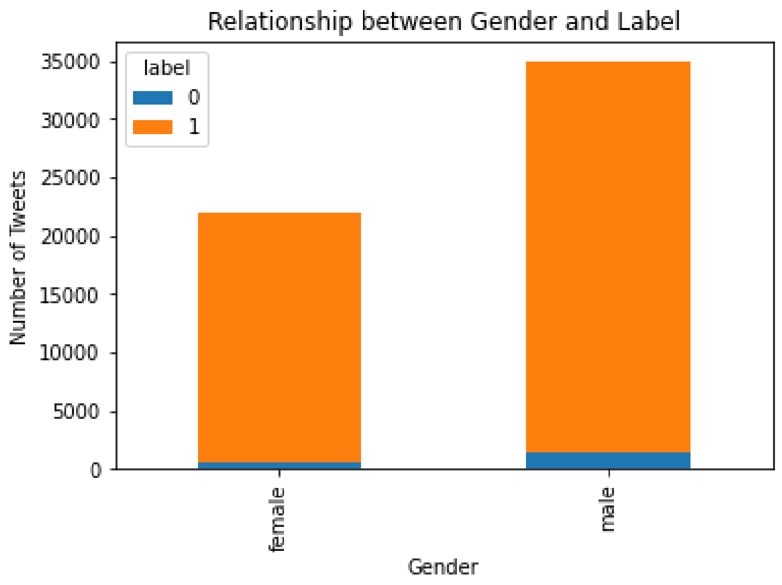
The heatmap reveals that, with the exception of Uzbek men, who also have a high percentage favor of education tweets, female members of all ethnic groups have the largest percentage of tweets in favor of education. This shows that female in general, regardless of their ethnic background, are more vocal about their support for education on Twitter. Additionally, the heatmap reveals that across all ethnicities, Pashtun men have the lowest proportion of tweets in favor of education. This would suggest that Pashtun men have a different attitude toward education than men in other communities.

4.2 Number of from Each Gender and Label Category

```
# Create a crosstab of gender and label categories
gender_label = pd.crosstab(df.gender, df.label)
# Print the number of tweets by gender and label category
print("Number of tweets by gender and label category:")
print(gender_label)
# Create a stacked bar chart of gender and label categories
gender_label.plot.bar(stacked=True)
plt.title('Relationship between Gender and Label')
plt.xlabel('Gender')
plt.ylabel('Number of Tweets')
plt.show()
```

Label	0	1
Gender		
Female	560	21341
Male	1341	33592

The count of tweets by gender is displayed as a result. 34,933 tweets were posted by male users, compared to 21,901 by female users. The graph displays a stacked bar graph of the quantity of tweets broken down by gender and label type.



It is clear from the graph that men tweet more frequently than women do. The orange hue shows that the number of tweets in support of education, whereas the blue color denotes the number of neutral tweets. More tweets from men and women than from neutral users support education. The longer orange bar in the male section of the graph, however, indicates that men tweet substantially more frequently than women do in support of education. It is obvious that men tweet more frequently than women when comparing the two genders' total amount of tweets. However, as the previous data demonstrated, women tweet more frequently in favor of education than men do. This shows that while men may tweet more frequently overall, women are more outspoken about their support for education on Twitter.

4.3 Largest Tweets

With the help of this code, a table summarizing the number of people in various racial and gender groups can be created. The number of people in each combination of ethnic and gender categories is counted in a contingency table made using the pd.crosstab() method. The distribution of the data between these two variables is shown graphically in the table below.

```
ethnic_gender = pd.crosstab(df.ethnic, df.gender)
print(ethnic_gender)
```

The outcome is a table summarizing the number of people in various ethnic and gender groupings.

gender	female	male
ethnic		
hazara	1128	3100

pashtun	10035	18920
tajik	10010	10907
uzbek	728	2006

As a result, a table summarizing the number of people in various racial and gender groupings is produced. The table's layout designates each row as an ethnic group and each column as a gender category. The values in the table show how many people fall into each grouping of ethnic and gender groups. The Hazara ethnic group, for instance, has 3100 males and 1128 females. Male Pashtuns make up the largest group, with a total of 18920; Tajik men come in second with 10907. This table offers a quick and simple method to see how many people there are in the dataset across racial and gender categories. It can be used to spot any potential gender and ethnicity related patterns or trends.

5.0 Vectorization

Vectorization is the act of transforming textual input into a numerical format that machine learning algorithms can handle in natural language processing (NLP). Machine learning models can only process numerical data, so this is essential.

```python
from sklearn.feature_extraction.text import CountVectorizer
from nltk.corpus import stopwords
import string
# Define a function to preprocess the text
def preprocess(text):
    # Convert the text to lowercase
    text = text.lower()
    # Remove punctuation and non-alphanumeric characters
    text = "".join(c for c in text if c.isalnum() or c.isspace())
    # Remove stop words
    stop_words = set(stopwords.words("english"))
    words = text.split()
    filtered_words = [word for word in words if word not in stop_words]
    # Join the filtered words back into a string
    text = " ".join(filtered_words)
    return text
# Preprocess the training and test data
X_train_processed = [preprocess(text) for text in X_train]
X_test_processed = [preprocess(text) for text in X_test]
# Create a CountVectorizer object and fit it to the training data
vectorizer = CountVectorizer()
X_train_vectorized = vectorizer.fit_transform(X_train_processed)
# Transform the test data using the same vectorizer
X_test_vectorized = vectorizer.transform(X_test_processed)
```

5.1 The output CountVectorizer

The table displays the outcomes of vectorizing the preprocessed text data using scikit-CountVectorizer learn’s object. Each column in the table represents a distinct word from the lexicon, and each row in the table represents a data point. The frequency of each word in the associated data point is shown by the values in the table. For instance, the value "1" in row 0 and column 985 denotes that the first data point has one instance of the term corresponding to column 985. Only the non-zero values are displayed since the values are stored in a sparse matrix format. The table offers a useful method for numerically representing text data, which can be utilized as data for machine learning models. Based on the text input, the generated matrix can be used to train a model to predict the target variable.

Row	(i, j) index	Count
0	(0, 985)	1
0	(0, 4830)	1
0	(0, 9910)	1
0	(0, 13283)	1
0	(0, 14703)	1
0	(0, 17155)	1
0	(0, 20273)	1
0	(0, 21178)	1
0	(0, 26797)	1
0	(0, 29986)	1
0	(0, 31600)	1
0	(0, 32324)	1
1	(1, 985)	1
1	(1, 2209)	1
1	(1, 3562)	1
1	(1, 5029)	1
1	(1, 7365)	1
1	(1, 10101)	2
1	(1, 16913)	1
1	(1, 16971)	2
1	(1, 18601)	1
1	(1, 19578)	1
1	(1, 19579)	1
1	(1, 25931)	1
1	(1, 26958)	1

The bag-of-words (BoW) model is a popular technique for vectorizing text data. Each document is represented as a vector in the BoW model, where each element in the vector denotes the frequency of a specific word inside the document. For instance, if we have a collection of documents with the terms "apple," "banana," and "orange," we may represent the documents "I enjoy apples," "She likes bananas," and "We eat oranges every day" as the vectors [1, 0, 0], [0, 1, 0], and [0, 0, 1], respectively.

5.2 Train Logistic

In order to divide text input into two classes according to its sentiment, this code trains a logistic regression classifier (positive or negative). The text input is converted into numerical features using the CountVectorizer, and some unnecessary words are subsequently taken out of the feature vocabulary. The logistic regression classifier is then trained using the cleaned feature data.

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
import numpy as np

vectorizer = CountVectorizer(stop_words=stopwords.words('english'))
X_train_vectorized = vectorizer.fit_transform(X_train)
unwanted_words = ['afghanistan', 'taliban', 'afghan', 'kabul']
for word in unwanted_words:
    try:
        idx = vectorizer.vocabulary_[word]
        del vectorizer.vocabulary_[word]
        vectorizer._validate_vocabulary()
        X_train_vectorized = X_train_vectorized[:, np.arange(X_train_vectorized.shape[1])
!= idx]
    except KeyError:
        pass
X_train_vectorized = vectorizer.fit_transform(X_train)
lr = LogisticRegression()
lr.fit(X_train_vectorized, y_train)

```

5.3 Train Model

The second section of the code uses the trained model to predict outcomes on test data and assesses the model's effectiveness by printing a classification report. To be more precise, it applies the trained logistic regression model to forecast the target variable values (y_pred) for the test data that has already been transformed and vectorized (X_test_vectorized), and then computes and prints a report of various performance metrics, including precision, recall, F1-score, support for each class label, and overall model accuracy, by comparing the predicted values with the actual target variable values (y_test). The classification report offers information on the model's performance for various class labels and can be used to alter the model's hyperparameters or data preprocessing processes as necessary.

```
from sklearn.metrics import classification_report
y_pred = lr.predict(X_test_vectorized)
print(classification_report(y_test, y_pred))
```

5.4 Classification

```
from sklearn.feature_extraction.text import CountVectorizer
from prettytable import PrettyTable
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report
import string
import re
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
# Clean text df
df['text'] = df['text'].astype(str)
def clean_text(text):
    # Remove mentions
    text = re.sub(r'@[A-Za-z0-9_]+', '', text)
    # Remove hashtags
    text = re.sub(r'#', '', text)
    # Remove retweets
    text = re.sub(r'RT : ', '', text)
    # Remove urls
    text = re.sub(r'https?:\/\/[A-Za-z0-9\.\/]+', '', text)
    # Remove punctuations and convert to lower case
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text.lower())
    return text
df['text'] = df['text'].apply(clean_text)
# Feature engineering
```

```
keywords = ['women', 'girl', 'school', 'university', 'ban']
cv = CountVectorizer(vocabulary=keywords)
features = cv.fit_transform(df['text']).toarray()
# Split df into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, df['label'], test_size=0.2, random_state=42)
# Train the classifier
clf = MultinomialNB()
clf.fit(X_train, y_train)
# Make predictions
y_pred = clf.predict(X_test)
# Evaluate the model
print(classification_report(y_test, y_pred))

```

5.5 Classification Report

The classification report below shows how well a binary classifier performs in determining whether or not a tweet is about a favor of education (class 1) (class 0). The report includes evaluation measures for the classifier's performance, including precision, recall, F1-score, and support. The classification reports up top displays how well a binary classifier performs in determining whether or not a tweet is about a disaster (class 1) (class 0). The report includes evaluation measures for the classifier's performance, including precision, recall, F1-score, and support.

Class	Precision	Recall	F1-Score	Support
0	0.00	0.00	0.00	344
1	0.97	1.00	0.98	11023
Accuracy		0.97		11367
Macro Avg	0.48	0.50	0.49	11367
Weighted Avg	0.94	0.97	0.95	11367

However, the classifier performs poorly for tweets (class 0) that are not about education, with a precision, recall, and F1-score of 0.00. This indicates that none of the tweets that were supposed to be about education. Given the minimal quantity of class 0 tweets in the dataset, it is not unexpected that the classifier had trouble identifying these tweets (only 344). The classifier's overall accuracy is 0.97, which indicates that 97 percent of its predictions were accurate. However, accuracy can be deceptive when there is an imbalance in the classes, as there is in this instance. Due to the classifier's terrible performance on tweets of class 0, the macro average of the F1-score is a very low 0.49. The weighted average of the F1-score is 0.95, which is high considering the unbalanced class composition. The classifier does a good job of detecting tweets about favor of education, but it does a terrible job of identifying tweets about other types of tweets. This is hardly shocking given the dataset's scant inclusion of non-education tweets. More tweets from class 0 would be beneficial to the dataset in order to enhance the performance of the classifier. In order to balance the classes and enhance the performance of the classifier, additional strategies like data augmentation or oversampling might be used. Keep in mind that the precision for class 0 is 0.00, indicating that none of the tweets were accurately classified by the model as being in class 0. The model failed to accurately identify any of the tweets that truly belonged to class 0, as indicated by the recall for class 0 being 0.00. On the other hand, the model was very good at properly recognizing tweets that belonged to class 1, as evidenced by the high precision and recall for class 1. The model's performance on class 1 has a stronger impact on the overall accuracy, which is likewise high, because there are more tweets in class 1 compared to class 0, which has a considerably smaller amount of tweets. Overall, the model does a great job of recognizing tweets that belong to class 1, but class 0 tweet identification still needs to be improved upon due to the relatively small quantity of tweets in that class.

Conclusion

In conclusion, natural language processing (NLP) is a quickly developing science with a wide range of applications. Preprocessing is essential to NLP because it converts text data from an unstructured form into one that machine learning models can handle. Another crucial component of NLP is vectorization, which transforms textual input into numerical features that can be used as input for machine learning models. A common way for vectorizing text data is the CountVectorizer method, which may be used to build a bag-of-words model that represents each document as a vector, with each element designating the frequency of a particular word within the document. Additionally, a well-liked tool for categorizing text data into two or more groups based on numerical features produced by the CountVectorizer is the logistic regression classifier. Another well-liked classification method that can be utilized to foretell the class label of text data is the Multinomial Naive Bayes classifier. Both approaches can be used to divide tweets regarding education, for instance, into those that are in favor of it and those that are neutral. The Multinomial Naive Bayes classifier's classification report for the dataset demonstrates that the model does a fantastic job of identifying tweets that belong to class 1 tweets in support of education.