MBTI based on text , second iteration

Made by Megin van Herk

importing the required libraries for the predictions beforehand

In [1]:
```
#importing all neccesary libraries
import pandas as pd
import numpy as np
import sklearn as sk
import matplotlib
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
import plotly.express as px
import warnings #remove warnings
warnings.filterwarnings("ignore")
from collections import Counter
import string
import os
import re
import tqdm
import nltk

#wordcloud
from os import path
from PIL import Image
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

#tokenize

from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.tokenize import RegexpTokenizer, word_tokenize
from tqdm import tqdm
from subprocess import check_output
from collections import Counter
from wordcloud import WordCloud, ImageColorGenerator
#TDIdf
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
import collections
#model
from sklearn.naive_bayes import MultinomialNB
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, roc_curve
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
import seaborn as sns
```

The goal of this document is to prepare data for an eventual personality detector

Domain understanding

A personality is a set of traits or characteristics that determine how an individual thinks, feels, and acts. One of the most utilized psychological instruments for understanding and predicting human behavior is the Myers–Briggs Type Indicator (MBTI), a popular instrument for over 50 years that is now widely discussed on social media. Based on Jung's theory of psychological types (1971), MBTI is a personality measurement model that outlines a person's preferences along four dimensions, where each distinct dimension describes the propensities of the individual

The Myers Briggs Type Indicator (or MBTI for short)

is a personality type system that divides everyone into 16 distinct personality types across 4 axis:

- Introversion (I) – Extroversion (E)
- Intuition (N) – Sensing (S)
- Thinking (T) – Feeling (F)
- Judging (J) – Perceiving (P)

Data collecting

in this research, the dataset was obtained from the Personality Cafe forum. This dataset is available on Kaggle and comprises 8675 rows, with the first column consisting of MBTI type and the second column containing individuals' posts (less than or equal to 50 items), divided by "|||" (the 3-pipe symbol). After the symbol was removed, there were 422,845 posts in the entire row of data.

In [2]:
```
#importing the data into dataframe
train_data=pd.read_csv('mbti_1.csv')
```

Checking the amount of each personality type there is

In [3]:
```
type_count=train_data['type'].value_counts()
print(type_count)
```

```
INFP    1832
INFJ    1470
INTP    1304
INTJ    1091
ENTP     685
ENFP     675
ISTP     337
ISFP     271
ENTJ     231
ISTJ     205
ENFJ     190
ISFJ     166
ESTP      89
ESFP      48
ESFJ      42
ESTJ      39
Name: type, dtype: int64
```

There is a high inbalance within the data, this could later on lead into a bias for here, INFP. since there is more data of it, the chance of it predicting this personality type is allot higher

In [30]:
```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8675 entries, 0 to 8674
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   type         8675 non-null   object
 1   posts        8675 non-null   object
 2   clean_posts  8675 non-null   object
 3   archetype    8675 non-null   object
dtypes: object(4)
memory usage: 271.2+ KB
```

we have 2 columns with a total of 8675 rows

Data cleaning

Text data

To make sure the accuracy is as high as possible, it needs to made sure the data is clean, this is done by first checking if there are any empty rows

In [4]:
```
print("Missing values in test data: ")
print(train_data.isnull().sum())
```

```
Missing values in test data:
type     0
posts    0
dtype: int64
```

no missing values, good. Now since this is text data, we need to clean the text, this is done by removing links and emojis, and repeated words. After this cleaning the remaining text will consists of only key words and clean text.

In [5]:
```
#importing libraries
import re
from bs4 import BeautifulSoup
import string
from nltk.stem.snowball import SnowballStemmer
```

A function is build to remove the special characters, uncluding the ||| that seperates all the different text inputs

In [6]:
```
def text_cleaning(text):
    text=BeautifulSoup(text,'lxml').text
    #removing html and seperators
    text=re.sub(r'\|\|\|', r'  ', text)
    text=re.sub(r'http\S+', r'  ', text)
    #removing puntuations
    text=text.replace('.', '  ')
    translator=str.maketrans('', '', string.punctuation)
    text=text.translate(translator)
    #removing numbers
    text=''.join(i for i in text if not i.isdigit())
    return text
```

Apply the function to the text data

In [7]:
```
train_data['clean_posts']=train_data['posts'].apply(text_cleaning)
```

In [8]:
```
train_data.head()
```

Out[8]:

	type	posts	clean_posts
0	INFJ	'http://www.youtube.com/watch?v=qsXHcwe3krw|||...	enfp and intj moments sportscente...
1	ENTP	'I'm finding the lack of me in these posts ver...	Im finding the lack of me in these posts very ...
2	INTP	'Good one _____ https://www.youtube.com/wat...	Good one Of course to which I say I kn...
3	INTJ	'Dear INTP, I enjoyed our conversation the o...	Dear INTP I enjoyed our conversation the oth...
4	ENTJ	'You're fired.|||That's another silly misconce...	Youre fired Thats another silly misconcepti...

As you can see a new column has been added were it is seen that the links and special characters have been removed from the text data

Since the data is from a forum, people use slang languages and repeat words for emphasis, these words can be removed using STEM, a function is build to do this:

In [9]:
```
def stem_text(text):
    stemmer = SnowballStemmer('english')
    words_list = text.split()
    new_list = []
    for i in words_list:
        word = stemmer.stem(i)
        new_list.append(word)

    words = new_list
    words = ' '.join(words)
    return words
```

Apply the function to the text data

In [10]:
```
train_data['clean_posts'] = train_data['clean_posts'].apply(stem_text)
```

Now the text data as clean as I can get it

In [11]:
```
train_data.head()
```

Out[11]:

	type	posts	clean_posts
0	INFJ	'http://www.youtube.com/watch?v=qsXHcwe3krw|||...	enfp and intj moment sportscent not top ten pl...
1	ENTP	'I'm finding the lack of me in these posts ver...	im find the lack of me in these post veri alar...
2	INTP	'Good one _____ https://www.youtube.com/wat...	good one of cours to which i say i know that m...
3	INTJ	'Dear INTP, I enjoyed our conversation the o...	dear intp i enjoy our convers the other day es...
4	ENTJ	'You're fired.|||That's another silly misconce...	your fire that anoth silli misconcept that app...

Archetypes

For this particular research I require to split the MBTI personality types into 4

- **The Visionary:**
  - MBTI Types: INFP, INFJ, ENTP, ENFP
  - Reasoning: These types are often characterized by their creativity, open-mindedness, and a focus on possibilities. They tend to be imaginative, future-oriented, and driven by their ideals.
- **The Organizer:**
  - MBTI Types: INTP, INTJ, ISTP, ISFP
  - Reasoning: These types are known for their practicality, attention to detail, and organizational skills. They excel at planning, analyzing, and bringing order to complex situations.
- **The Connector :**
  - MBTI Types: ENFJ, ISFJ, ESFJ
  - Reasoning: These types are people-oriented, empathetic, and excel in building strong relationships. They are often supportive, caring, and skilled at connecting with others on an emotional level.
- **The Guide :**
  - MBTI Types: ENTJ, ISTJ, ESTP, ESFP, ESTJ
  - Reasoning: These types are often seen as leaders and guides. They tend to be analytical, logical, and can provide clear direction. They are also action-oriented and focused on achieving practical results.

We group the existing MBTI types to these 4

Just incase, there are any hidden spaces or small letters, the column is converted to string upper and stripped of empty spaces

In [12]:
```
train_data['type'] = train_data['type'].str.upper().str.strip()
```

A function is then built to group them

In [13]:
```
def map_to_group(mbti_type):
    if mbti_type in ['INFP', 'INFJ', 'ENTP', 'ENFP']:
        return 'Visionary'
    elif mbti_type in ['INTP', 'INTJ', 'ISTP', 'ISFP']:
        return 'Organizer'
    elif mbti_type in ['ENFJ', 'ISFJ', 'ESFJ']:
        return 'Connector'
    elif mbti_type in ['ENTJ', 'ISTJ', 'ESTP', 'ESFP', 'ESTJ']:
        return 'Guide'
    else:
        return 'fuck'
```

Apply the function to the data

In [15]:
```
train_data['archetype'] = train_data['type'].apply(map_to_group)
```

In [16]:
```
train_data
```

Out[16]:

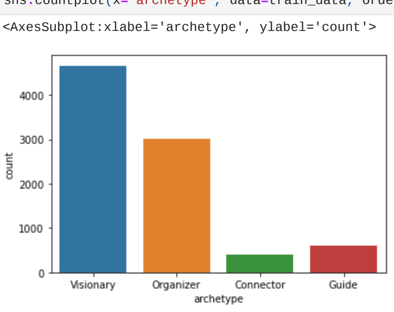
	type	posts	clean_posts	archetype
0	INFJ	'http://www.youtube.com/watch?v=qsXHcwe3krw|||...	enfp and intj moment sportscent not top ten pl...	Visionary
1	ENTP	'I'm finding the lack of me in these posts ver...	im find the lack of me in these post veri alar...	Visionary
2	INTP	'Good one _____ https://www.youtube.com/wat...	good one of cours to which i say i know that m...	Organizer
3	INTJ	'Dear INTP, I enjoyed our conversation the o...	dear intp i enjoy our convers the other day es...	Organizer
4	ENTJ	'You're fired.|||That's another silly misconce...	your fire that anoth silli misconcept that app...	Guide
...	...	...	...	...
8670	ISFP	'https://www.youtube.com/watch?v=t8edHB_h908||...	ixfp just becaus i alway think of cat as fi do...	Organizer
8671	ENFP	'So...if this thread already exists someplace ...	so if this thread alreadi exist someplac els w...	Visionary
8672	INTP	'So many questions when i do these things. I ...	so mani question when i do these thing i would...	Organizer
8673	INFP	'I am very conflicted right now when it comes ...	i am veri conflict right now when it come to w...	Visionary
8674	INFP	'It has been too long since I have been on per...	it has been too long sinc i have been on perso...	Visionary

8675 rows × 4 columns

Now there is a new column defining the archetypes, Lets see how many of each archetype there are:

In [21]:
```
# Create a count plot
sns.countplot(x='archetype', data=train_data, order=['Visionary', 'Organizer', 'Connector', 'Guide'])
```

Out[21]:
```
<AxesSubplot:xlabel='archetype', ylabel='count'>
```



There is still a high inbalance, but since there is such a large difference in data count for each, balancing could make the model very inaccurate. We will see how each performs

Data preparation

To prepare the data for predictions, It is needed to split into test and train. This is done because

Generalization: To check if a model works well not just on training data but can generalize to new, unseen data.

Evaluation: Separating data helps assess a model's performance objectively, avoiding bias towards the training set.

Overfitting Prevention: Ensures that the model doesn't memorize the training data but learns patterns applicable to various data.

Model Tuning: Helps fine-tune the model based on its performance on the test set, improving its robustness.

Real-world Performance: Reflects how well the model is expected to perform when applied to new, real-world scenarios.

In [23]:
```
X= train_data['clean_posts']
y= train_data['archetype']
```

In [24]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

To be able to use the text data in a model, its needs to be converted into numbers, for this TFIDF is used. this transforms all the data into numbers

In [25]:
```
# Feature extraction using TfidfVectorizer
tfidf = TfidfVectorizer(stop_words='english')
X_train = tfidf.fit_transform(X_train)
X_test = tfidf.transform(X_test)
```

When an input from a user is put into the model for predictions, their text also needs to be transformed into numbers. We use the same TFIDS for this, so lets save it into a pickle file to be able to use later on

In [26]:
```
import pickle
```

In [27]:
```
# Save the TfidfVectorizer object as a pickle file
with open('tfidf_vectorizer.pkl', 'wb') as f:
    pickle.dump(tfidf, f)
```

When predicting we can call this file again

Modelling

To see how accurate a model performs a classification report is made,

- Precision
  - Precision is the ratio of true positive predictions to the total number of positive predictions made by the model. It measures the accuracy of the positive predictions.
  - High precision indicates that when the model predicts a positive class, it is likely correct.
- Recall
  - Recall is the ratio of true positive predictions to the total number of actual positive instances in the dataset. It measures the model's ability to capture all positive instances
  - High recall indicates that the model is effective at identifying all instances of the positive class.
- f1 score
  - The F1 score is the harmonic mean of precision and recall. It provides a balance between precision and recall, especially in situations where there is an imbalance between the number of positive and negative instances.
  - F1 score ranges from 0 to 1, where a higher value indicates a better balance between precision and recall.
- Support
  - Support is the number of actual occurrences of the class in the specified dataset. It provides context about the distribution of classes in the dataset.
  - In the context of a classification report, you will see support values for each class, indicating how many instances of each class are present in the dataset.

Multinomial Naive Bayes:

Well-suited for text classification tasks. Efficient and often performs well with text data.

In [31]:
```
# Convert text data to TF-IDF features
vectorizer = TfidfVectorizer(max_features=8000)  # You can adjust max_features based on your data size
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

# Train a Multinomial Naive Bayes classifier
classifier = MultinomialNB()
classifier.fit(X_train_tfidf, y_train)

# Make predictions on the test set
y_pred = classifier.predict(X_test_tfidf)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Print classification report
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.55
              precision    recall  f1-score   support

   Connector       0.00      0.00      0.00        93
       Guide       0.00      0.00      0.00       118
   Organizer       0.73      0.08      0.15       606
   Visionary       0.54      0.99      0.70       918

    accuracy                           0.55      1735
   macro avg       0.32      0.27      0.21      1735
weighted avg       0.54      0.55      0.42      1735
```

Here we see the imbalance of the dataset is affecting the perfomance of a model, the lower quantity archetypes have a 0.00, very shit. lets try another model

Support Vector Machines (SVM):

Effective for text classification tasks. Can handle high-dimensional data like text features.

In [33]:
```
from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline

# Use an SVM classifier with a TF-IDF vectorizer
classifier = make_pipeline(TfidfVectorizer(max_features=5000), SVC())
classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Print classification report
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.77
              precision    recall  f1-score   support

   Connector       0.80      0.17      0.28        93
       Guide       0.81      0.22      0.35       118
   Organizer       0.77      0.76      0.77       606
   Visionary       0.77      0.91      0.83       918

    accuracy                           0.77      1735
   macro avg       0.79      0.52      0.56      1735
weighted avg       0.77      0.77      0.75      1735
```

0.77 accuracy is not bad at all, here the imbalance is surprisingly performing better

Random Forest:

Ensemble methods like Random Forest can work well for a variety of tasks, including text classification. Robust and less prone to overfitting.

In [35]:
```
from sklearn.ensemble import RandomForestClassifier
# Use a Random Forest classifier with a TF-IDF vectorizer
classifier = make_pipeline(TfidfVectorizer(max_features=5000), RandomForestClassifier())
classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Print classification report
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.65
              precision    recall  f1-score   support

   Connector       0.00      0.00      0.00        93
       Guide       0.00      0.00      0.00       118
   Organizer       0.76      0.46      0.57       606
   Visionary       0.62      0.93      0.75       918

    accuracy                           0.65      1735
   macro avg       0.35      0.35      0.33      1735
weighted avg       0.59      0.65      0.60      1735
```

Again bad

Logistic Regression:

Simple and interpretable. Can be a good baseline model for text classification tasks.

In [37]:
```
from sklearn.linear_model import LogisticRegression
# Use a Logistic Regression classifier with a TF-IDF vectorizer
classifier = make_pipeline(TfidfVectorizer(max_features=5000), LogisticRegression())
classifier.fit(X_train, y_train)
# Save the trained classifier to a pickle file
with open('logistic_regression_classifier.pkl', 'wb') as model_file:
    pickle.dump(classifier, model_file)

# Make predictions on the test set
y_pred = classifier.predict(X_test)


# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Print classification report
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.77
              precision    recall  f1-score   support

   Connector       0.79      0.12      0.21        93
       Guide       0.75      0.15      0.25       118
   Organizer       0.77      0.75      0.76       606
   Visionary       0.76      0.92      0.83       918

    accuracy                           0.77      1735
   macro avg       0.77      0.49      0.51      1735
weighted avg       0.77      0.77      0.74      1735
```

Logistic regressions performs similary to SVM, but a little bit worse, when it comes to precisions, I will use SVM for further modelling

Testing

First we grab the model we made earlier, the best performing one, SVM

In [38]:
```
with open('logistic_regression_classifier.pkl', 'rb') as model_file:
    loaded_classifier = pickle.load(model_file)
```

A function is built to get the input from the user, clean their text, translate it to english if needed and then give them the predicted archetype

In [50]:
```
import pickle
from googletrans import Translator
from sklearn.metrics import accuracy_score

def predict_archetype_with_translation(input_text, true_group, model_filename='logistic_regression_classifier.pkl'):
    # Load the trained classifier from the pickle file
    with open(model_filename, 'rb') as model_file:
        loaded_classifier = pickle.load(model_file)

    # Translate the input text to English using googletrans
    translator = Translator()
    english_text = translator.translate(input_text, src='auto', dest='en').text

    # Make predictions for the translated text
    prediction = loaded_classifier.predict([english_text])

    # Calculate the accuracy of the model on the provided true group
    accuracy = accuracy_score([true_group], prediction)

    return prediction[0], accuracy  # Return the predicted group and accuracy

# Example of usage:
user_input_text = "In de rustige straten van het kleine dorpje hoorde je het zachte geluid van vogels in de vroege ochtend. De kleurrijke bloemen in de voortuinen leken te dan
true_group = "The actual group corresponding to the input text."

predicted_group, accuracy = predict_archetype_with_translation(user_input_text, [true_group])

# print(f'Translated Text: {translator.translate(user_input_text, src="auto", dest="en").text}')
print(f'Your archetype is: {predicted_group}')
# print(f'Model Accuracy: {accuracy:.2%}')
```

```
Your archetype is: Organizer
```

Some additional information for each archetype could be handy:

In [55]:
```
def get_archetype_description(archetype):
    descriptions = {
        'Visionary': 'Visionaries are creative and open-minded individuals who focus on possibilities and are driven by their ideals.',
        'Organizer': 'Organizers are practical and detail-oriented, excelling in planning, analyzing, and bringing order to complex situations.',
        'Connector': 'Connectors are people-oriented and empathetic, skilled at building strong relationships and providing support.',
        'Guide': 'Guides are analytical and logical leaders, providing clear direction and focusing on achieving practical results.'
    }

    return descriptions.get(archetype, 'Unknown archetype.')
```

This function will return a description of your archetype

Now We can call the full function

In [57]:
```
# Example of usage:
user_input_text = "My name is megin, I dont enjoy coding, i just want to be appreciated, In my free time i like to play video games and boost my ego"
true_group = "The actual group corresponding to the input text."

predicted_group, accuracy = predict_archetype_with_translation(user_input_text, [true_group])

description = get_archetype_description(predicted_group)

print(f'Your archetype is: {predicted_group}')

print(f'Description for the archetype "{predicted_group}": {description}')
```

```
Your archetype is: Organizer
Description for the archetype "Organizer": Organizers are practical and detail-oriented, excelling in planning, analyzing, and bringing order to complex situations.
```

Nice! I filled it in knowing what my MBTI is and what that MBTI consists with which archetype, and it got it correct! now it does need a quite a bit of text to be able to predict what archetype you are, but the questions dont have to be related to anything with social work.

Now I can transcribe this function into javascript and make a functioing website around it, lesgooo