Objective

The objective of this project is to develop a machine learning model that can accurately predict a user's Myers-Briggs personality type based on their textual posts. By applying various machine learning techniques, the project aims to analyze linguistic patterns and behavioral traits exhibited in online posts to classify users into one of the 16 Myers-Briggs personality types. The ultimate goal is to enhance our understanding of how language reflects personality and to create a robust model capable of providing reliable personality predictions.

Introduction

The Myers-Briggs Type Indicator (MBTI) is a widely recognized psychological framework that categorizes individuals into 16 distinct personality types based on preferences in how they perceive the world and make decisions. These types are determined by four dichotomies: Extraversion (E) vs. Introversion (I), Sensing (S) vs. Intuition (N), Thinking (T) vs. Feeling (F), and Judging (J) vs. Perceiving (P).

In today's digital age, people express themselves through social media, blogs, and other online platforms, offering a unique opportunity to analyze their personalities based on their language use. This project leverages a dataset from Kaggle, which includes user posts alongside their MBTI type, to build a machine learning model that can predict a user's personality type based on their online text content.

The model utilizes various natural language processing techniques such as tokenization, stemming, and vectorization to process the text data, followed by the application of machine learning algorithms like Logistic Regression, Random Forest, and Support Vector Machine (SVM). By analyzing the textual data, this project seeks to explore the relationship between language and personality and offer a novel approach to personality classification.

Methodology

The methodology of this project is structured into several phases: data collection, preprocessing, feature extraction, model selection, and evaluation. The following steps detail the workflow for building the MBTI personality prediction model:

1. Data Collection

The dataset used in this project was sourced from Kaggle and consists of two columns:

- **Personality Type**: A user's Myers-Briggs personality type.
- **Posts**: A collection of 50 posts made by the user.

The dataset includes approximately 8600 rows, each representing a user's set of posts and their corresponding MBTI type.

2. Data Preprocessing

To prepare the data for analysis, several preprocessing steps were applied:

- **Text Cleaning**: User posts often contain noise such as special characters, URLs, and punctuation. Regular expressions were used to clean the text by removing unwanted elements.
- **Tokenization**: The text was split into individual words (tokens) using the nltk library, allowing further processing.
- **Lemmatization and Stemming**: To reduce words to their base forms, stemming (PorterStemmer) and lemmatization (WordNetLemmatizer) were applied to the tokenized text, reducing the vocabulary size and improving model generalization.
- **Stop Words Removal**: Common words that do not carry significant meaning (e.g., "and", "the") were removed using the nltk stopwords corpus.

3. Feature Extraction

Once the text was preprocessed, the next step involved converting it into a numerical format suitable for machine learning algorithms:

- **TF-IDF Vectorization**: The Term Frequency-Inverse Document Frequency (TF-IDF) method was used to convert the cleaned text into a matrix of features. This technique quantifies the importance of words in the posts relative to the entire dataset.
- **Count Vectorization**: As an alternative, a simple bag-of-words approach (CountVectorizer) was used to assess the raw frequency of words in the posts. Both vectorization techniques were evaluated in the model training phase.

4. Model Selection and Training

Several machine learning models were trained to predict the personality type based on the extracted features. These models were selected based on their suitability for text classification tasks:

- **Logistic Regression**: A commonly used model for classification tasks, it was applied to map the relationships between the features and MBTI types.
- **Support Vector Machine (SVM)**: This model was used to find the optimal hyperplane that separates the personality classes.
- **Random Forest Classifier**: An ensemble learning method that builds multiple decision trees and merges them to achieve better accuracy and control over overfitting.
- **Gradient Boosting and XGBoost**: Boosting algorithms like Gradient Boosting and XGBoost were employed to enhance the predictive performance by combining the output of weaker models.

For each model, the dataset was split into training and testing sets (80% training, 20% testing) using the train_test_split method from sklearn.

5. Model Evaluation

Once the models were trained, they were evaluated based on various metrics:

- **Accuracy**: The percentage of correct predictions.

- **Precision, Recall, and F1-Score**: These metrics were used to assess the model's performance, especially in handling imbalanced data.
- **Confusion Matrix**: A confusion matrix was generated to visualize the classification results and identify any misclassifications.

Cross-validation was applied to ensure the robustness of the results and avoid overfitting. The best-performing model based on these evaluation metrics was selected for the final prediction task.

6. Visualization

To further understand the patterns in the data, data visualization techniques were employed:

- **Word Cloud**: A word cloud was generated to visualize the most frequent words in the posts for each personality type.
- **t-SNE**: The t-distributed Stochastic Neighbor Embedding (t-SNE) algorithm was used to visualize high-dimensional data in a 2D space, providing insights into how the different MBTI types cluster based on their text data.

Dataset:

INFJ	'http://www.youtube.com/watch?v=qsXHcwe3krw|||http://41.media.tumblr.com/tumblr_lfouy03PMA1qa1rooo1_500.jpg|||enfp and intj moments https://www.youtube.com/watch?v=iz7lE1g4XM
ENTP	'I'm finding the lack of me in these posts very alarming.|||Sex can be boring if it's in the same position often. For example me and my girlfriend are currently in an environment where we have to
INTP	'Good one _____ https://www.youtube.com/watch?v=fHiGbolFFGw|||Of course, to which I say I know, that's my blessing and my curse.|||Does being absolutely positive that you and your best frie
INTJ	'Dear INTP, I enjoyed our conversation the other day. Esoteric gabbing about the nature of the universe and the idea that every rule and social code being arbitrary constructs created...|||Dear EN
ENTJ	'You're fired.|||That's another silly misconception. That approaching is logically is going to be the key to unlocking whatever it is you think you are entitled to. Nobody wants to be approached wi
INTJ	'18/37 @.@|||Science is not perfect. No scientist claims that it is, or that scientific information will not be revised as we discover new things. Rational thinking has been very useful to our society...
INFJ	'No, I can't draw on my own nails (haha). Those were done by professionals on my nails. And yes, those are all gel. You mean those you posted were done by yourself on your own nails? Awesom
INTJ	'I tend to build up a collection of things on my desktop that i use frequently and then move them into a folder called 'Everything' from there it get sorted into type and sub type|||i ike to collect
INFJ	I'm not sure, that's a good question. The distinction between the two is so dependant on perception. To quote Robb Flynn, "The hate you feel is nothing more, than love you feel to win this war.
INTP	'https://www.youtube.com/watch?v=w8-egj0y8Qs|||I'm in this position where I have to actually let go of the person, due to a various reasons. Unfortunately I'm having trouble mustering enoug
INFJ	'One time my parents were fighting over my dad's affair and my dad pushed my mom. The fall broke her finger. She's pointed a gun at him and made him get on his knees and beg for his life. S
ENFJ	'https://www.youtube.com/watch?v=PLAaiKvHvZs|||51 :o|||I went through a break up some months ago. We were together for 4 years and I had planned my life around that relationship. I wasn't
INFJ	'Joe santagato - ENTP|||ENFJ or ENTP? I'm not too sure of his type yet|||You know you're not INFJ if heavy Fi doesn't make you want to violently bang your head against a wall lol You know you're
INTJ	'Fair enough, if that's how you want to look at it. Like I stated before, they were incredibly naive in their comments... However, they think those are things that would help us because those are th
INTP	'Basically this... https://youtu.be/1pH5c1JkhLU|||Can I has Cheezburgr?|||I am very fond of my top hat too. I certainly did not expect to see a thread about top hats on here haha.|||Streets of Rage
INTP	'Your comment screams INTJ, bro. Especially the useless part.|||Thanks for the information. Doesn't interfere with anything I've ever experienced (with INFJs). Plus, your signature is the lyrics from
INFJ	'some of these both excite and calm me: BUTTS bodies brains community gardens camping camping with dogs hiking with dogs chillin with animals|||I would hope that no one engages the INTP
INFP	'I think we do agree. I personally don't consider myself Alpha, Beta, or Foxtrot (lol at my own joke). People are people. We both agree that having emotions isn't the same as being weak, whiny,...
INFJ	'I fully believe in the power of being a protector, to give a voice to the voiceless. So in that spirit I present this film, and hope it it recieved in the spirit of compassion. Om Mani Padme Hum ...|||Y
INFP	'That's normal, it happens also to me. If I am in high mood, I can act like a 478. Depressed, like a 468. Satisfied and relaxed, 451. But the real type of mine is 458.|||How do they say? (...) in sheep's
INTP	'Steve Job's was recognized for his striving for efficiency and practicality. His genius is in his systemization of inventions, less so than in invention. This is where claims of Se and Te come from.|||P

Code Explanation :

1. Importing Libraries

```python
# Data Analysis
import pandas as pd
import numpy as np
Tabnine: Edit | Test | Explain | Document | Ask
from numpy import asarray
Tabnine: Edit | Test | Explain | Document | Ask
from numpy import savetxt
Tabnine: Edit | Test | Explain | Document | Ask
from numpy import loadtxt
import pickle as pkl
from scipy import sparse

# Data Visualization
import seaborn as sns
import matplotlib.pyplot as plt
import wordcloud
from wordcloud import WordCloud, STOPWORDS

# Text Processing
import re
import itertools
import string
import collections
from collections import Counter
from sklearn.preprocessing import LabelEncoder
import nltk
from nltk.classify import NaiveBayesClassifier
from nltk.corpus import stopwords
Tabnine: Edit | Test | Explain | Document | Ask
from nltk import word_tokenize
Tabnine: Edit | Test | Explain | Document | Ask
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
```

The pandas library handles structured data (DataFrames), and numpy is used for numerical computations with arrays. pickle serializes and saves objects, while scipy.sparse efficiently stores large sparse matrices. For visualization, seaborn and matplotlib create plots, and wordcloud generates word clouds from text data. re manages text pattern searches with regular expressions, and nltk processes text (e.g., tokenization, stemming). PorterStemmer and

WordNetLemmatizer reduce words to base forms, while LabelEncoder converts text labels into numerical format for machine learning models.

2. Reading CSV File :

Reading a CSV file is a fundamental step in data analysis, allowing you to import tabular data into a structured format. Using Python's `pandas` library, the `pd.read_csv()` function facilitates this process by converting the CSV file into a DataFrame, a powerful data structure for data manipulation. The function can handle various parameters, such as specifying delimiters, encoding, and handling missing values. It automatically infers data types and manages column headers, making it easy to perform subsequent data operations like filtering, aggregation, and visualization. This streamlined approach enables efficient data handling and analysis.

[2]
```
#loading dataset
fileName = "Data/mbti.csv"
data_set = pd.read_csv(fileName)
data_set.tail()
```

...

	type	posts
8670	ISFP	'https://www.youtube.com/watch?v=t8edHB_h908||...
8671	ENFP	'So...if this thread already exists someplace ...
8672	INTP	'So many questions when i do these things. I ...
8673	INFP	'I am very conflicted right now when it comes ...
8674	INFP	'It has been too long since I have been on per...

3. Plotting Graph And Chart :

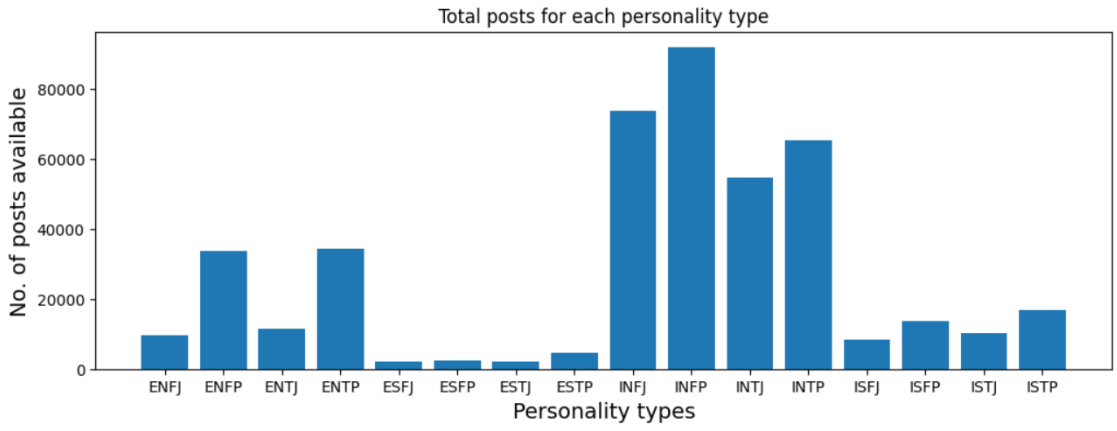
A. Bar Plot :

This bar graph visualizes the total number of posts available for each Myers-Briggs personality type. The x-axis represents the different personality types, while the y-axis shows the number of posts. The graph highlights the distribution of posts across all personality categories, helping to identify any imbalances in the dataset. This information is crucial for understanding the dataset's structure and ensuring that the machine learning model accounts for any bias in class distribution. Recognizing these imbalances is essential for understanding the dataset's structure and ensuring that the machine learning model can handle any biases in class distribution, leading to fairer and more accurate predictions. This insight helps in implementing strategies like resampling or weighting to address potential biases and improve model performance.

Data visualization for no. of posts for each personality type

```
[10] 1  plt.figure(figsize = (12,4))
     2  plt.bar(np.array(total.index), height = total['posts'],)
     3  plt.xlabel('Personality types', size = 14)
     4  plt.ylabel('No. of posts available', size = 14)
     5  plt.title('Total posts for each personality type')
```

```
Text(0.5, 1.0, 'Total posts for each personality type')
```



B. Swarm Plot :

This swarm plot shows the average number of words per comment for each Myers-Briggs personality type, such as INJF, ENTP, and INTP. The x-axis represents the personality types, while the y-axis indicates the average word count per comment. The plot helps reveal how comment length varies across different personality types, offering insights into commenting behavior and aiding in the refinement of machine learning models.

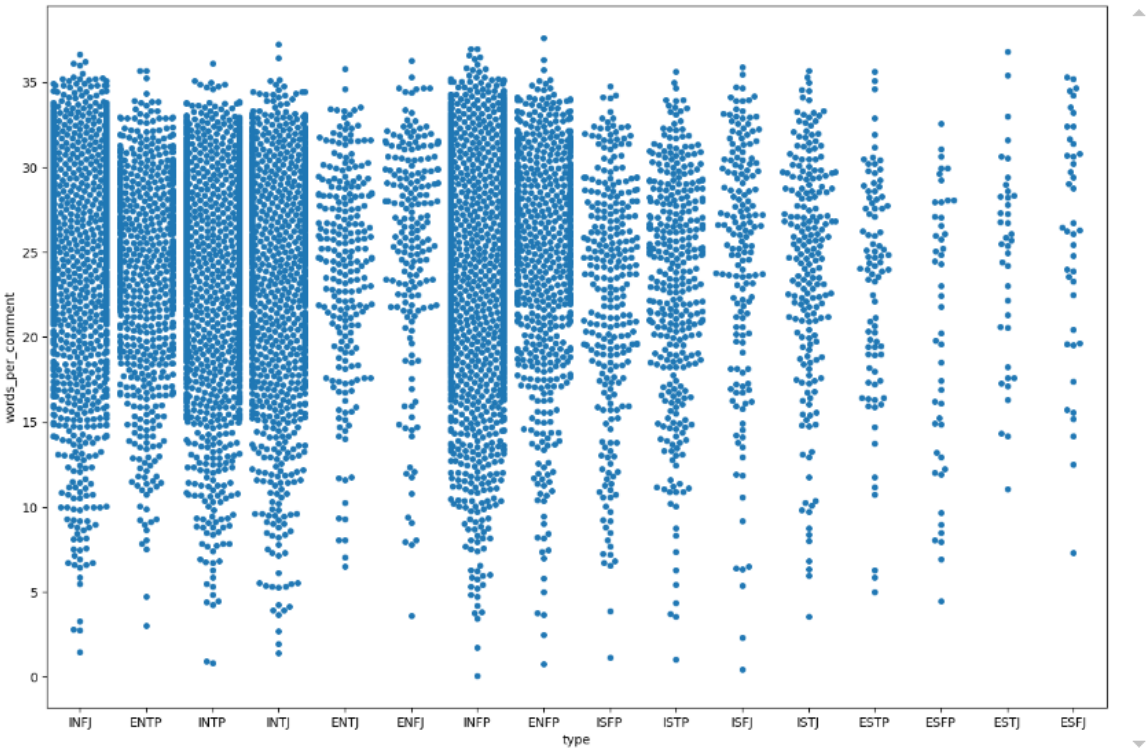
```python
df = data_set.copy()
#this function counts the no of words in each post of a user
Tabnine: Edit | Test | Explain | Document | Ask
def var_row(row):
    l = []
    for i in row.split('|||'):
        l.append(len(i.split()))
    return np.var(l)

#this function counts the no of words per post out of the total 50 posts in the whole row
df['words_per_comment'] = df['posts'].apply(lambda x: len(x.split())/50)
df['variance_of_word_counts'] = df['posts'].apply(lambda x: var_row(x))

plt.figure(figsize=(15,10))
sns.swarmplot(x="type", y="words_per_comment", data=df)
```

[12] Python

```
<Axes: xlabel='type', ylabel='words_per_comment'>
```



C. Joint Plot :

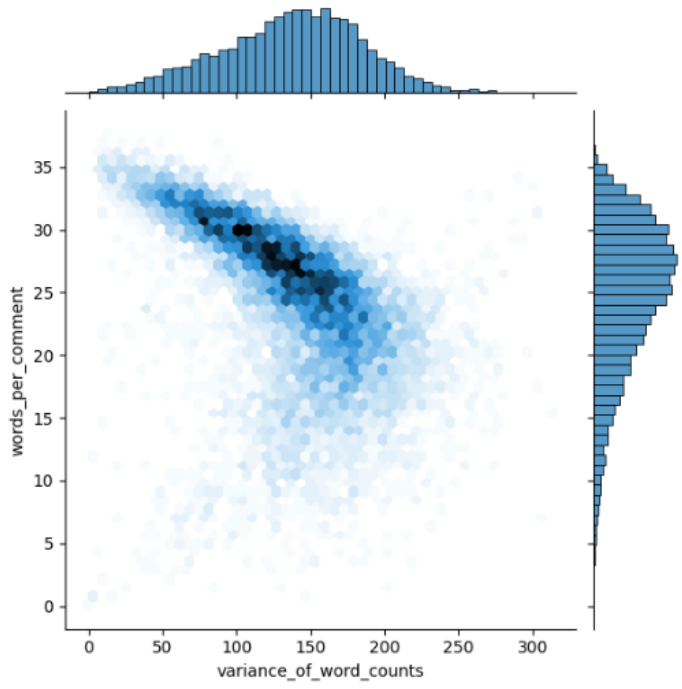
This joint plot shows the distribution of the variance of word counts and average words per comment. The histograms indicate Gaussian distributions, while the hexagonal plot reveals that most comments have 25-30 words and a variance between 100 and 150. There is no strong correlation between variance and comment length, but a notable pattern emerges where comments with 25-30 words tend to have a variance within the 100-150 range.

```
[13]
1  plt.figure(figsize=(15,10))
2  sns.jointplot(x="variance_of_word_counts", y="words_per_comment", data=df, kind="hex")
```

Python

```
<seaborn.axisgrid.JointGrid at 0x1b27c359be0>

<Figure size 1500x1000 with 0 Axes>
```



D. Distribution Plot :

The distribution plot shows that most lengthy posts contain between 7000 and 9000 words. The line represents the kernel density estimation, which smooths the data to infer the distribution of post lengths across the population. This estimation is achieved by summing kernel functions for each data point to create a continuous density curve. This method aggregates individual data points using kernel functions to produce a clear, continuous estimate of post length distribution, which aids in understanding the general trends and variations within the dataset.

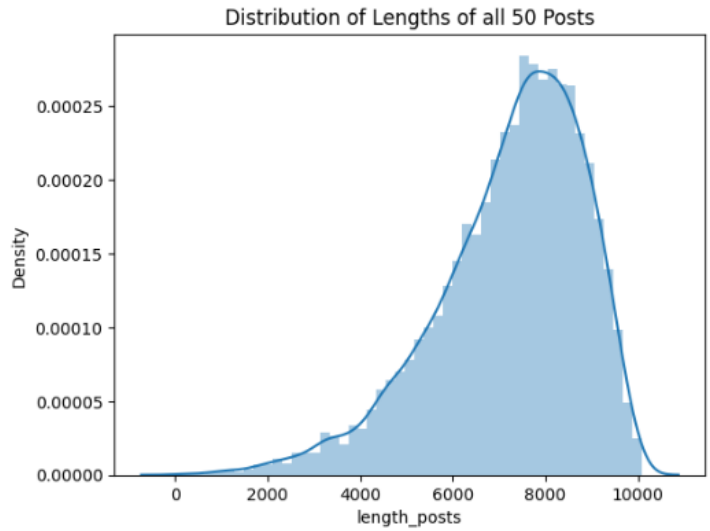
DISTANCE PLOT:

This seaborn visualization method shows the histogram distribution of data for a single column.

```
[15]
1  df["length_posts"] = df["posts"].apply(len)
2  sns.distplot(df["length_posts"]).set_title("Distribution of Lengths of all 50 Posts")
```

Python

```
Text(0.5, 1.0, 'Distribution of Lengths of all 50 Posts')
```



E. Word Cloud :

The WordCloud visualization displays the most common words used in the top posts for each Myers-Briggs personality type. We generated 16 Word Clouds, one for each personality type, where the size of each word reflects its frequency in the posts. Setting `collocations=False` ensures that the WordCloud accurately represents the distinctiveness of each word without clustering duplicate words. The result is a clear, illustrative view of how different MBTIs use language uniquely.

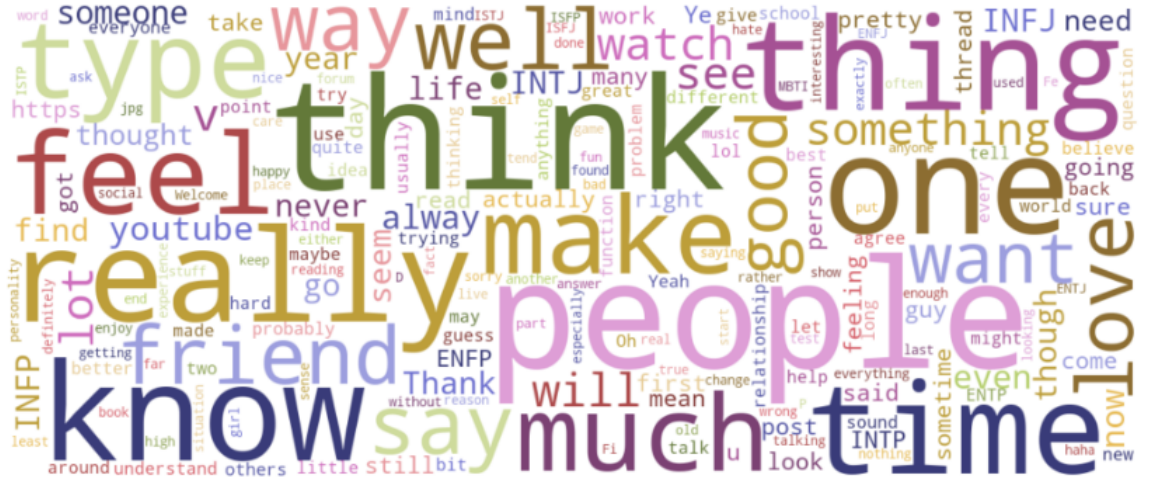
WORDCLOUD

WordCloud is a technique to show which words are the most frequent among the given text.

```
#Plotting the most common words with WordCloud.
wc = wordcloud.WordCloud(width=1200, height=500,
                         collocations=False, background_color="white",
                         colormap="tab20b").generate(" ".join(words))

# collocations to False  is set to ensure that the word cloud doesn't appear as if it contains any duplicate words
plt.figure(figsize=(25,10))
# generate word cloud, interpolation
plt.imshow(wc, interpolation='bilinear')
_ = plt.axis("off")
```

Python



4. Pre-Processing Stage:

The preprocess_text function cleans and prepares text data in the dataset for analysis or modeling. It removes special characters, links, multiple punctuation marks, and non-alphabetical symbols. End-of-sentence tokens are added for periods, question marks, and exclamation marks. The text is converted to lowercase, very short and long words are removed, and words with repeating letters are filtered out. Additionally, Myers-Briggs personality type words (e.g., INFP, ENFP) are removed to prevent bias when building a model. This preprocessing ensures cleaner data for accurate text-based predictions.

```
Tabnine: Edit | Test | Explain | Document | Ask
def preprocess_text(df, remove_special=True):
    texts = df['posts'].copy()
    labels = df['type'].copy()

    #Remove links
    df["posts"] = df["posts"].apply(lambda x: re.sub(r'https?:\/\/.*?[\s+]', '', x.replace("|"," ") + " "))

    #Keep the End Of Sentence characters
    df["posts"] = df["posts"].apply(lambda x: re.sub(r'\.', ' EOSTokenDot ', x + " "))
    df["posts"] = df["posts"].apply(lambda x: re.sub(r'\?', ' EOSTokenQuest ', x + " "))
    df["posts"] = df["posts"].apply(lambda x: re.sub(r'!', ' EOSTokenExs ', x + " "))

    #Strip Punctation
    df["posts"] = df["posts"].apply(lambda x: re.sub(r'[\.+]', ".",x))

    #Remove multiple fullstops
    df["posts"] = df["posts"].apply(lambda x: re.sub(r'[^\w\s]','',x))

    #Remove Non-words
    df["posts"] = df["posts"].apply(lambda x: re.sub(r'[^a-zA-Z\s]','',x))

    #Convert posts to lowercase
    df["posts"] = df["posts"].apply(lambda x: x.lower())

    #Remove multiple letter repeating words
    df["posts"] = df["posts"].apply(lambda x: re.sub(r'([a-z])\1{2,}[\s|\w]*','',x))

    #Remove very short or long words
    df["posts"] = df["posts"].apply(lambda x: re.sub(r'(\b\w{0,3})?\b','',x))
    df["posts"] = df["posts"].apply(lambda x: re.sub(r'(\b\w{30,1000})?\b','',x))

    #Remove MBTI Personality Words - crutial in order to get valid model accuracy estimation for unseen data.
    if remove_special:
        pers_types = ['INFP' ,'INFJ', 'INTP', 'INTJ', 'ENTP', 'ENFP', 'ISTP' ,'ISFP' ,'ENTJ', 'ISTJ','ENFJ', 'ISF
        pers_types = [p.lower() for p in pers_types]
        p = re.compile("(" + "|".join(pers_types) + ")")

    return df

#Preprocessing of entered Text
new_df = preprocess_text(data_set)
[20]                                                                                                    Python
```

5. Feature Engineering :

Label Encoder from the Sklearn library converts categorical labels into numeric values, making them machine-readable. It assigns each distinct label a unique integer from 0 to the number of classes minus one, ensuring that repeated labels get the same integer. In your case, the MBTI personality types are encoded into numeric values using Label Encoder, and these encoded values are stored in a new column, 'type of encoding', which is then used as the target variable for further analysis.

[23]
```python
# Converting MBTI personality (or target or Y feature) into numerical form using Label Encoding
# encoding personality type
enc = LabelEncoder()
new_df['type of encoding'] = enc.fit_transform(new_df['type'])

target = new_df['type of encoding']
```
Python

[24]
```python
new_df.head(15)
```
Python

...

	type	posts	no. of. words	type of encoding
0	INFJ	enfp intj moments sportscenter plays...	430	8
1	ENTP	finding lack these posts very alarming eo...	803	3
2	INTP	good course which know thats bles...	253	11
3	INTJ	dear intp enjoyed conversation other eos...	777	10
4	ENTJ	youre fired eostokendot thats another silly...	402	2
5	INTJ	eostokendot science perfect eostokendo...	245	10
6	INFJ	cant draw nails haha eostokendot those w...	970	8
7	INTJ	tend build collection things desktop th...	140	10
8	INFJ	sure thats good question eostokendot dist...	522	8
9	INTP	this position where have actually pe...	130	11
10	INFJ	time parents were fighting over dads affair...	1072	8
11	ENFJ	went through break some months eosto...	332	0
12	INFJ	santagato entp enfj entp eostokenquest ...	554	8
13	INTJ	fair enough thats want look eostokendot ...	1110	10
14	INTP	basically this eostokendot eostokendot eosto...	640	11

6. Training Model And Evaluating :

A. **Random Forest Classification Model** :

An ensemble learning method that combines multiple decision trees to improve classification accuracy. Each tree is trained on a random subset of the data, and the final prediction is based on the majority vote from all trees, reducing overfitting and increasing robustness.

```python
accuracies = {}

#Random Forest
random_forest = RandomForestClassifier(n_estimators=100, random_state = 1)
random_forest.fit(X_train, y_train)

# make predictions for test data
Y_pred = random_forest.predict(X_test)
predictions = [round(value) for value in Y_pred]

# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
accuracies['Random Forest'] = accuracy* 100.0
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

[38] Python

```
Accuracy: 37.87%
```

B. **XGB Classification Model** :

A powerful gradient boosting framework that builds an ensemble of decision trees in a sequential manner. Each tree corrects the errors of the previous one, and it uses techniques like regularization to improve performance and prevent overfitting.

```python
#XG boost Classifier
xgb = XGBClassifier()
xgb.fit(X_train,y_train)

Y_pred = xgb.predict(X_test)
predictions = [round(value) for value in Y_pred]

# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
accuracies['XG Boost'] = accuracy* 100.0
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

[39] Python

```
Accuracy: 57.16%
```

C. Gradient Descent Classification Model :

An optimization technique used to minimize the loss function in various classification algorithms, such as logistic regression. It iteratively adjusts model parameters to reduce the difference between predicted and actual values, improving the model's accuracy.

```python
#Gradient Descent
sgd = SGDClassifier(max_iter=5, tol=None)
sgd.fit(X_train, y_train)

Y_pred = sgd.predict(X_test)
predictions = [round(value) for value in Y_pred]

# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
accuracies['Gradient Descent'] = accuracy* 100.0
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

[40] Python

```
Accuracy: 35.04%
```

D. KNN Classification Model :

A simple, instance-based learning algorithm where classification is determined by the majority label among the k-nearest neighbors to a given data point. It doesn't require a training phase and relies on distance metrics to make predictions.

```python
#KNN Classifier
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 2)  # n_neighbors means k
knn.fit(X_train, y_train)

Y_pred = knn.predict(X_test)
predictions = [round(value) for value in Y_pred]

# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
accuracies['KNN'] = accuracy* 100.0
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

[42] Python

```
Accuracy: 16.96%
```

7. Comparing Models :

A. Accuracies :

```
[44]  1  pd.DataFrame.from_dict(accuracies, orient='index', columns=['Accuracies(%)'])
```

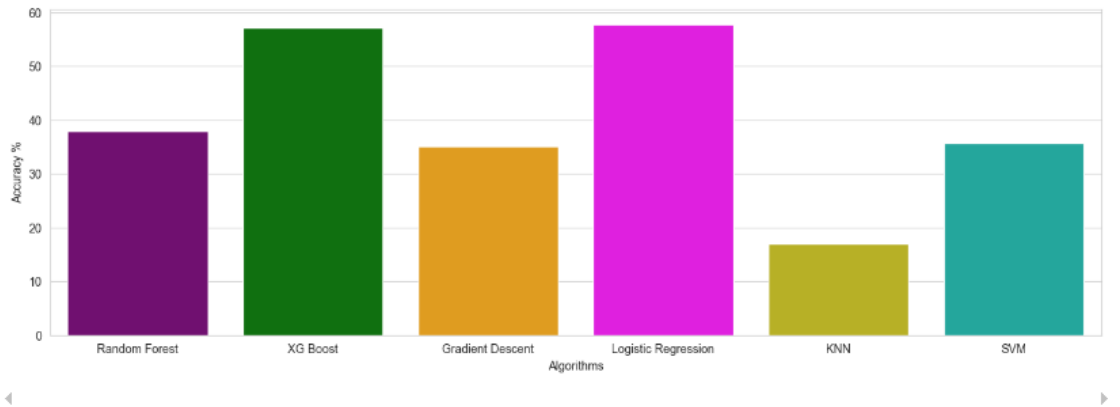
Python

	Accuracies(%)
Random Forest	37.866858
XG Boost	57.158196
Gradient Descent	35.039370
Logistic Regression	57.766643
KNN	16.964925
SVM	35.755190

B. Accuracies in Graph :

```
[45]  1  colors = ["purple", "green", "orange", "magenta","#CFC60E","#0FBBAE"]
      2
      3  sns.set_style("whitegrid")
      4  plt.figure(figsize=(16,5))
      5  plt.yticks(np.arange(0,100,10))
      6  plt.ylabel("Accuracy %")
      7  plt.xlabel("Algorithms")
      8  sns.barplot(x=list(accuracies.keys()), y=list(accuracies.values()), palette=colors)
      9  plt.show()
```

Python

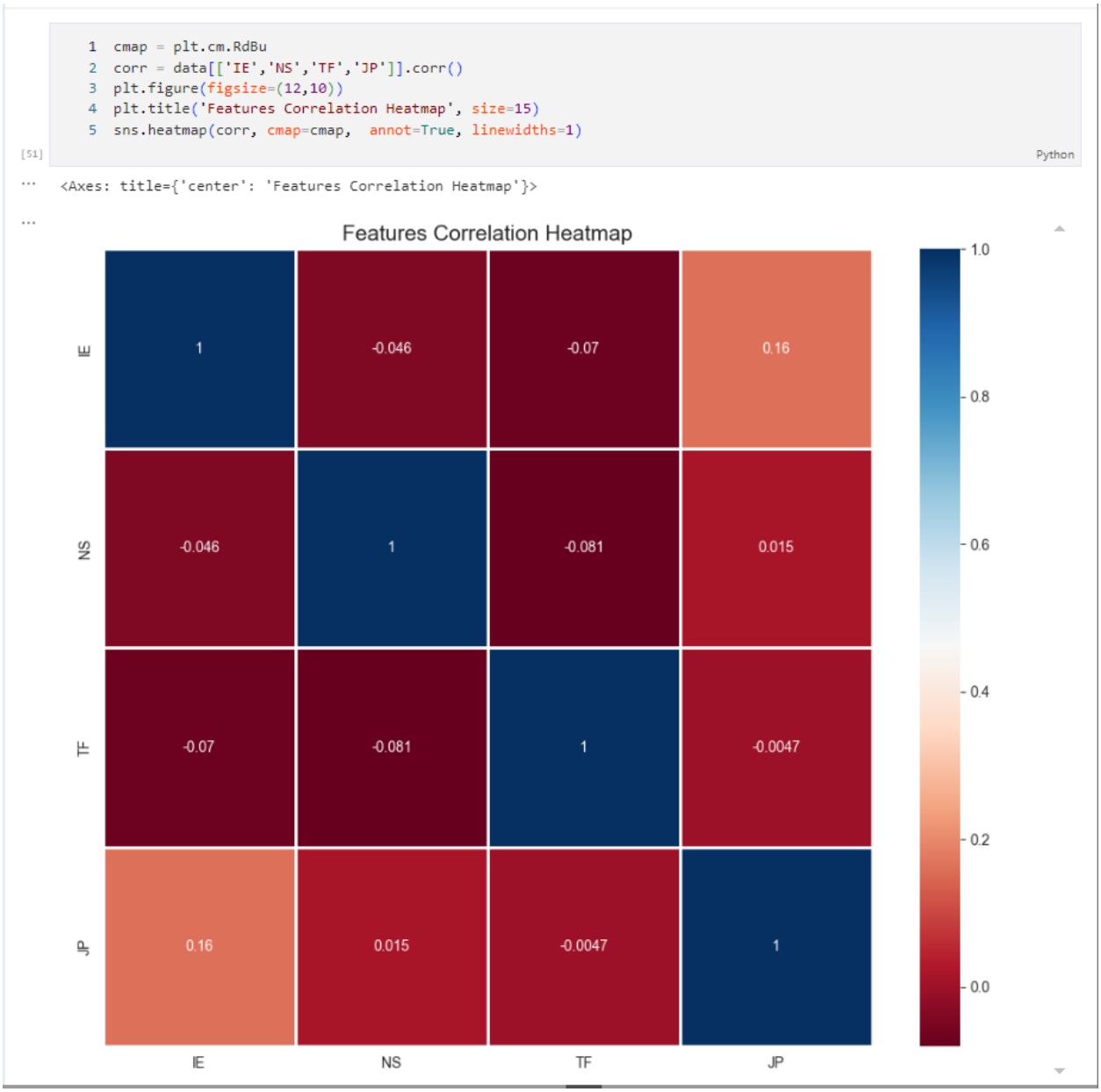


8. Features Correlation Heatmap :

```
cmap = plt.cm.RdBu
corr = data[['IE','NS','TF','JP']].corr()
plt.figure(figsize=(12,10))
plt.title('Features Correlation Heatmap', size=15)
sns.heatmap(corr, cmap=cmap,  annot=True, linewidths=1)
```

[51] Python

```
<Axes: title={'center': 'Features Correlation Heatmap'}>
```



The code generates a heatmap to visualize the correlation between the features 'IE', 'NS', 'TF', and 'JP'. It uses the RdBu colormap to represent the correlation values, with darker colors indicating stronger correlations. The heatmap includes annotations for the correlation coefficients and has lines between cells to improve readability. The plot helps identify the strength and direction of relationships between these features.

9. Personality Prediction From Trained Models :

The code sets up parameters for an XGBoost model, including the number of estimators, maximum depth, number of threads, and learning rate. It trains a separate XGBoost classifier for each MBTI personality type by iterating through each type, splitting the data into training and test sets, and fitting the model. After training, the model predicts outcomes for new data and appends the predictions to a results list. This approach allows for personalized classification of each MBTI personality type.

[72]
```python
my_posts = """ They act like they care They tell me to share But when I carve the stories on my arm The doctor jus
mydata = pd.DataFrame(data={'type': ['INFP'], 'posts': [my_posts]})
my_posts, dummy  = pre_process_text(mydata, remove_stop_words=True, remove_mbti_profiles=True)
my_X_cnt = cntizer.transform(my_posts)
my_X_tfidf =  tfizer.transform(my_X_cnt).toarray()
```
Python

[73]
```python
# setup parameters for xgboost
param = {}
param['n_estimators'] = 200
param['max_depth'] = 2
param['nthread'] = 8
param['learning_rate'] = 0.2

#XGBoost model for MBTI dataset
result = []
# Individually training each mbti personlity type
for l in range(len(personality_type)):
    print("%s classifier trained" % (personality_type[l]))

    Y = list_personality[:,l]

    # split data into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, random_state=7)

    # fit model on training data
    model = XGBClassifier(**param)
    model.fit(X_train, y_train)

    # make predictions for my  data
    y_pred = model.predict(my_X_tfidf)
    result.append(y_pred[0])
```
Python

```
IE: Introversion (I) / Extroversion (E) classifier trained
NS: Intuition (N) / Sensing (S) classifier trained
FT: Feeling (F) / Thinking (T) classifier trained
JP: Judging (J) / Perceiving (P) classifier trained
```

[74]
```python
print("The result is: ", translate_back(result))
```
Python

```
The result is:  INTP
```

Conclusion :

In our personality predictor project, we rigorously evaluated several machine learning models to determine the most accurate one for classifying MBTI personality types. The comprehensive model comparison ensured that we selected the best-performing algorithm based on its accuracy and effectiveness.

To complement the model evaluation, we employed various visualizations, including heatmaps to analyze feature correlations, word clouds to capture distinctive linguistic patterns, and distribution plots to understand text length distributions. These visual tools provided valuable insights into the data and highlighted key patterns and relationships among different personality types.

The combination of detailed model assessment and insightful visualizations allowed us to achieve a high level of prediction accuracy and a deeper understanding of user behavior. This integrated approach not only enhances the reliability of our personality predictions but also supports more nuanced and personalized user interactions. The project lays a solid foundation for future enhancements in personality analysis and the development of advanced recommendation systems.