```
# Importing Libraries
import numpy as np
import pandas as pd
from sklearn import svm
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import time
```

```
# Importing MBTI Dataset into Collab Environment
from google.colab import files
uploaded = files.upload()
```

Choose Files mbti_1.csv

- **mbti_1.csv**(text/csv) - 62856486 bytes, last modified: 2/10/2022 - 100% done

```
Saving mbti_1.csv to mbti_1 (1).csv
```

```
# Converting uploaded dataset into a Pandas DataFrame
df = pd.read_csv('mbti_1.csv')
```

```
# Seeing the number of datapoints in each Label.
target_col = df.groupby(['type']).count()
print(target_col)
```

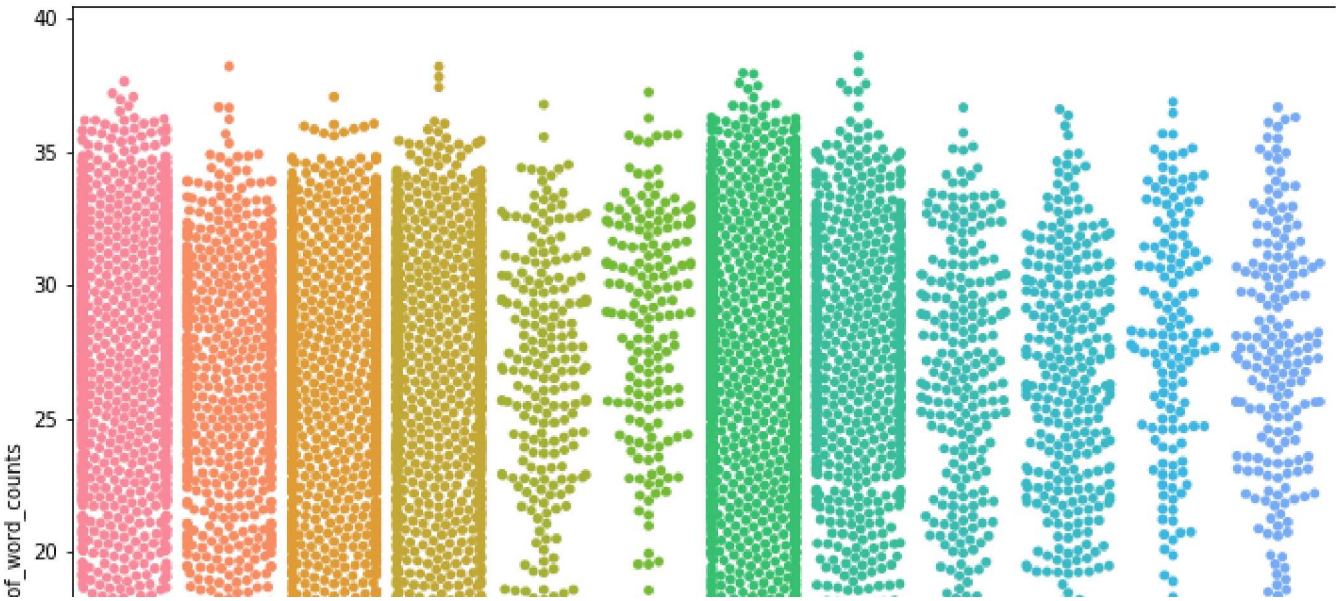
```
      posts
type
ENFJ    190
ENFP    675
ENTJ    231
ENTP    685
ESFJ     42
ESFP     48
ESTJ     39
ESTP     89
INFJ   1470
INFP   1832
INTJ   1091
INTP   1304
ISFJ    166
ISFP    271
ISTJ    205
ISTP    337
```

```
# Visualising the first datapoint
df['posts'][0]
```

```
''http://www.youtube.com/watch?v=qsXHcwe3krw|||http://41.media.tumblr.com/tumblr_lfouy0
3PMA1qa1rooo1_500.jpg|||enfp and intj moments  https://www.youtube.com/watch?v=iz7lE1g4
XM4  sportscenter not top ten plays  https://www.youtube.com/watch?v=uCdfze1etec  prank
s|||What has been the most life-changing experience in your life?|||http://www.youtube.
com/watch?v=vXZeYwwRDw8   http://www.youtube.com/watch?v=u8ejam5DP3E  On repeat for mos
t of today.|||May the PerC Experience immerse you.|||The last thing my INFJ friend post
ed on his facebook before committing suicide the next day. Rest in peace~   http://vime
o.com/22842206|||Hello ENFJ7. Sorry to hear of your distress. It's only natural for a r
elationship to not be perfection all the time in every moment of existence. Try to figu
```

```
# Visualising using a Swarm Plot:
import seaborn as sb
# Swarm Plot
df1 = df.copy()
#this function counts the average number of words in each post of a user
def mean_row(row):
    l = []
    for i in row.split('|||'):
        l.append(len(i.split()))
    return np.mean(l)
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
df1['words_per_datapoint'] = df1['posts'].apply(lambda x: len(x.split()))
df1['avg_words_per_comment'] = df1['posts'].apply(lambda x: len(x.split())/50)
df1['mean_of_word_counts'] = df1['posts'].apply(lambda x: mean_row(x))
df1['var_of_word_counts'] = df1['posts'].apply(lambda x: var_row(x))

plt.figure(figsize=(15,10))
sb.swarmplot("type", "mean_of_word_counts", data=df1)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:1296: UserWarning: 65.5% o
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:1296: UserWarning: 41.6% o
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:1296: UserWarning: 63.3% o
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:1296: UserWarning: 57.2% o
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:1296: UserWarning: 8.2% of
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:1296: UserWarning: 69.8% o
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:1296: UserWarning: 41.8% o
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:1296: UserWarning: 11.3% o
  warnings.warn(msg, UserWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f201a9d3850>
```




```
# Visualising the No of words per data-point, avg words per comment:
df1
```

	type	posts	words_per_datapoint	avg_words_per_comment	m
0	INFJ	'http://www.youtube.com/watch?v=qsXHcwe3krw|||...	556	11.12	
1	ENTP	'I'm finding the lack of me in these posts ver...	1170	23.40	
2	INTP	'Good one _____ https://www.youtube.com/wat...	836	16.72	
3	INTJ	'Dear INTP, I enjoyed our conversation the o...	1064	21.28	
4	ENTJ	'You're fired.|||That's another silly misconce...	967	19.34	
...	...	...	...	...	...
		'https://www.youtube.com/watch?			

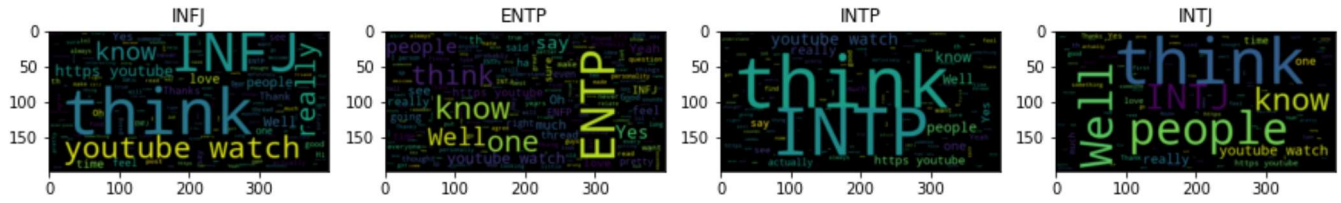
```
# Visulaising the dataset using the WordCloud:
from wordcloud import WordCloud
from collections import Counter

#Plotting WordCloud.

#Finding the most common words in all posts.
words = list(df1["posts"].apply(lambda x: x.split()))
words = [x for y in words for x in y]

most_common40 = Counter(words).most_common(40)
wc = 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 contain
plt.figure(figsize=(25,10))
# generate word cloud, interpolation
plt.imshow(wc, interpolation='bilinear')
_ = plt.axis("off")
```


```
# Seeing the most common top 40 words in the dataset.
Counter(words).most_common(40)
```

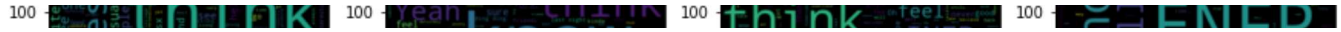
```
[('I', 387957),
 ('to', 290168),
 ('the', 270699),
 ('a', 230918),
 ('and', 219498),
 ('of', 177853),
 ('is', 128804),
 ('you', 128750),
 ('that', 127221),
 ('in', 117263),
 ('my', 104561),
 ('it', 93101),
 ('for', 83057),
 ('have', 79784),
 ('with', 77131),
 ('but', 74729),
 ('be', 69317),
 ('are', 65034),
 ('like', 61390),
 ('not', 59496),
 ('an', 59020),
 ("I'm", 57339),
 ('on', 57062),
 ('was', 56146),
 ('me', 55488),
 ('as', 53310),
 ('this', 52601),
 ('just', 48292),
 ('about', 46305),
 ('think', 46229),
 ('or', 45724),
 ("don't", 44821),
 ('so', 42935),
 ('your', 40918),
 ('do', 40867),
 ('what', 37746),
 ('at', 37566),
 ('can', 37535),
```

```
        ('if', 37042),
        ('people', 35546)]
```

```
fig, ax = plt.subplots(len(df1['type'].unique()), sharex=True, figsize=(15,len(df1['type'].un
k = 0
for i in df1['type'].unique():
    df_4 = df[df['type'] == i]
    wordcloud = WordCloud(max_words=1628,relative_scaling=1,normalize_plurals=False).generate
    plt.subplot(4,4,k+1)
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.title(i)
    ax[k].axis("off")
    k+=1
```




```
import re
import string
import math
```



```
df.drop_duplicates(inplace=True)
```

```
df
```

	type	posts
0	INFJ	'http://www.youtube.com/watch?v=qsXHcwe3krw|||...
1	ENTP	'I'm finding the lack of me in these posts ver...
2	INTP	'Good one _____ https://www.youtube.com/wat...
3	INTJ	'Dear INTP, I enjoyed our conversation the o...
4	ENTJ	'You're fired.|||That's another silly misconce...
...	...	...
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...

8675 rows × 2 columns

```
print(df.info())
print("--------------------------------------------------------------------------------")
print(df['type'].value_counts())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8675 entries, 0 to 8674
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   type    8675 non-null   object
 1   posts   8675 non-null   object
```

```
dtypes: object(2)
memory usage: 203.3+ KB
None
--------------------------------------------------------------------------------
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
```

```
# Expanding Contractions
contractions_dict = {
"ain't": "am not / are not / is not / has not / have not",
"aren't": "are not / am not",
"can't": "cannot",
"can't've": "cannot have",
"'cause": "because",
"could've": "could have",
"couldn't": "could not",
"couldn't've": "could not have",
"didn't": "did not",
"doesn't": "does not",
"don't": "do not",
"hadn't": "had not",
"hadn't've": "had not have",
"hasn't": "has not",
"haven't": "have not",
"he'd": "he had / he would",
"he'd've": "he would have",
"he'll": "he shall / he will",
"he'll've": "he shall have / he will have",
"he's": "he has / he is",
"how'd": "how did",
"how'd'y": "how do you",
"how'll": "how will",
"how's": "how has / how is / how does",
"I'd": "I had / I would",
"I'd've": "I would have",
"I'll": "I shall / I will",
"I'll've": "I shall have / I will have",
```

```
    "I'm": "I am",
    "I've": "I have",
    "isn't": "is not",
    "it'd": "it had / it would",
    "it'd've": "it would have",
    "it'll": "it shall / it will",
    "it'll've": "it shall have / it will have",
    "it's": "it has / it is",
    "let's": "let us",
    "ma'am": "madam",
    "mayn't": "may not",
    "might've": "might have",
    "mightn't": "might not",
    "mightn't've": "might not have",
    "must've": "must have",
    "mustn't": "must not",
    "mustn't've": "must not have",
    "needn't": "need not",
    "needn't've": "need not have",
    "o'clock": "of the clock",
    "oughtn't": "ought not",
    "oughtn't've": "ought not have",
    "shan't": "shall not",
    "sha'n't": "shall not",
    "shan't've": "shall not have",
    "she'd": "she had / she would",
    "she'd've": "she would have",
    "she'll": "she shall / she will",
    "she'll've": "she shall have / she will have",
    "she's": "she has / she is",
    "should've": "should have",
    "shouldn't": "should not",
    "shouldn't've": "should not have",
    "so've": "so have",
    "so's": "so as / so is",
    "that'd": "that would / that had",
    "that'd've": "that would have",
    "that's": "that has / that is",
    "there'd": "there had / there would",
    "there'd've": "there would have",
    "there's": "there has / there is",
    "they'd": "they had / they would",
    "they'd've": "they would have",
    "they'll": "they shall / they will",
    "they'll've": "they shall have / they will have",
    "they're": "they are",
    "they've": "they have",
    "to've": "to have",
    "wasn't": "was not",
    "we'd": "we had / we would",
    "we'd've": "we would have",
```

```
    "we'll": "we will",
    "we'll've": "we will have",
    "we're": "we are",
    "we've": "we have",
    "weren't": "were not",
    "what'll": "what shall / what will",
    "what'll've": "what shall have / what will have",
    "what're": "what are",
    "what's": "what has / what is",
    "what've": "what have",
    "when's": "when has / when is",
    "when've": "when have",
    "where'd": "where did",
    "where's": "where has / where is",
    "where've": "where have",
    "who'll": "who shall / who will",
    "who'll've": "who shall have / who will have",
    "who's": "who has / who is",
    "who've": "who have",
    "why's": "why has / why is",
    "why've": "why have",
    "will've": "will have",
    "won't": "will not",
    "won't've": "will not have",
    "would've": "would have",
    "wouldn't": "would not",
    "wouldn't've": "would not have",
    "y'all": "you all",
    "y'all'd": "you all would",
    "y'all'd've": "you all would have",
    "y'all're": "you all are",
    "y'all've": "you all have",
    "you'd": "you had / you would",
    "you'd've": "you would have",
    "you'll": "you shall / you will",
    "you'll've": "you shall have / you will have",
    "you're": "you are",
    "you've": "you have"
    }

    # Regular expression for finding contractions
    contractions_re=re.compile('(%s)' % '|'.join(contractions_dict.keys()))

    def expand_contractions(text,contractions_dict=contractions_dict):
        def replace(match):
            return contractions_dict[match.group(0)]
        return contractions_re.sub(replace, text)


    # Expanding Contractions in the reviews
    df['posts'] = df['posts'].apply(lambda x:expand_contractions(x))
```

```
# Making the whole data in lower case
df['posts'] = df['posts'].str.lower()
```

```
df['posts'][0]
```

```
''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=iz7le1g4xm4  sportscenter not top ten plays  https://www.youtube.com/watch?v=ucdfze1etec  pranks|||what has been the most life-changing experience in your life?|||http://www.youtube.com/watch?v=vxzeywwrdw8   http://www.youtube.com/watch?v=u8ejam5dp3e  on repeat for most of today.|||may the perc experience immerse you.|||the last thing my infj friend posted on his facebook before committing suicide the next day. rest in peace~   http://vimeo.com/22842206|||hello enfj7. sorry to hear of your distress. it's only natural for a relationship to not be perfection all the time in every moment of existence. try to figu
```

```
#Define the text from which you want to replace the url with "".
def remove_URL(text):
    """Remove URLs from a text string"""
    return re.sub(r"http\S+", "", text)

# Removing URL's in the reviews
df['posts'] = df['posts'].apply(lambda x:remove_URL(x))
```

```
print(df.shape)

# removing Punctuations
punctuation = """!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~)"""
print(punctuation)
#remove punctuation
df['posts'] = df['posts'].apply(lambda x: re.sub('[%s]' % re.escape(string.punctuation), '' ,
```

```
(8675, 2)
!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~)
```

```
df['posts'][0]
```

```
' and intj moments    sportscenter not top ten plays    prankswhat has been the most lifechanging experience in your life     on repeat for most of todaymay the perc experience immerse youthe last thing my infj friend posted on his facebook before committing suicide the next day rest in peace    enfj7 sorry to hear of your distress its only natural for a relationship to not be perfection all the time in every moment of existence try to figure the hard times as times of growth as84389  84390     welcome and stuff  game set matchprozac wellbrutin at least thirty minutes of moving your legs and i do not mean moving them while sitting in your same desk chair weed in moderation maybe try edibles as a healthier alternativebasically come up with three items you have determined th
```

```
#remove words and digits
# df['posts'] = df['posts'].apply(lambda x: re.sub('W*dw*','',x))
```

```
df['posts'][0]
```

```
' and intj moments    sportscenter not top ten plays    prankswhat has been the most li
fechanging experience in your life     on repeat for most of todaymay the perc experien
ce immerse youthe last thing my infj friend posted on his facebook before committing su
icide the next day rest in peace    enfj7 sorry to hear of your distress its only natur
al for a relationship to not be perfection all the time in every moment of existence tr
y to figure the hard times as times of growth as84389  84390     welcome and stuff  gam
e set matchprozac wellbrutin at least thirty minutes of moving your legs and i do not m
ean moving them while sitting in your same desk chair weed in moderation maybe try edib
les as a healthier alternativebasically come up with three items you have determined th
```

```
#remove stopwords
import nltk
nltk.download('stopwords')
```

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

```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
stop_words.add('http')
def remove_stopwords(text):
    return " ".join([word for word in str(text).split() if word not in stop_words])
df['posts'] = df['posts'].apply(lambda x: remove_stopwords(x))
```

```
df['posts'][0]
```

```
'intj moments sportscenter top ten plays prankswhat lifechanging experience life repeat
today may perc experience immerse youthe last thing infj friend posted facebook committi
ng suicide next day rest peace enfj7 sorry hear distress natural relationship perfectio
n time every moment existence try figure hard times times growth as84389 84390 welcome
stuff game set matchprozac wellbrutin least thirty minutes moving legs mean moving sitt
ing desk chair weed moderation maybe try edibles healthier alternativebasically come th
ree items determined type whichever types want would likely use given types cognitive f
unctions whatnot left byall things moderation sims indeed video game good one note good
one somewhat subjective completely promoting death given simdear enfp favorite video ga
```

```
#stemming
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
def stem_words(text):
    return " ".join([stemmer.stem(word) for word in text.split()])
df['posts'] = df['posts'].apply(lambda x: stem_words(x))
```

```
df['posts'][0]
```

```
'intj moment sportscent top ten play prankswhat lifechang experi life repeat todaymay p
erc experi immers youth last thing infj friend post facebook commit suicid next day res
t peac enfj7 sorri hear distress natur relationship perfect time everi moment exist tri
figur hard time time growth as84389 84390 welcom stuff game set matchprozac wellbrutin
least thirti minut move leg mean move sit desk chair weed moder mayb tri edibl healthie
r alternativebas come three item determin type whichev type want would like use given t
ype cognit function whatnot left byall thing moder sim inde video game good one note go
od one somewhat subject complet promot death given simdear enfp favorit video game grow
```

```
nltk.download('wordnet')
```

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

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
def lemmatize_words(text):
    return " ".join([lemmatizer.lemmatize(word) for word in text.split()])
df['posts'] = df['posts'].apply(lambda text: lemmatize_words(text))
```

```
len(df['posts'][0])
```

```
1868
```

```
len(df['posts'].values)
```

```
8675
```

```
# Removal of Frequent words
# In the previos preprocessing step, we removed the stopwords based on language information.

# So this step is to remove the frequent words in the given corpus. If we use something like

# Let us get the most common words adn then remove them in the next step
```

```
from collections import Counter
cnt = Counter()
for text in df['posts'].values:
    for word in text.split():
        cnt[word] += 1

cnt.most_common(10)
```

```
[('like', 73728),
 ('think', 56854),
 ('would', 50127),
 ('peopl', 45535),
 ('know', 38366),
 ('get', 37770),
```

```
        ('one', 37270),
        ('feel', 35671),
        ('realli', 34019),
        ('thing', 32963)]
```

```
target_names = np.unique(df['type'])
print(target_names)
```

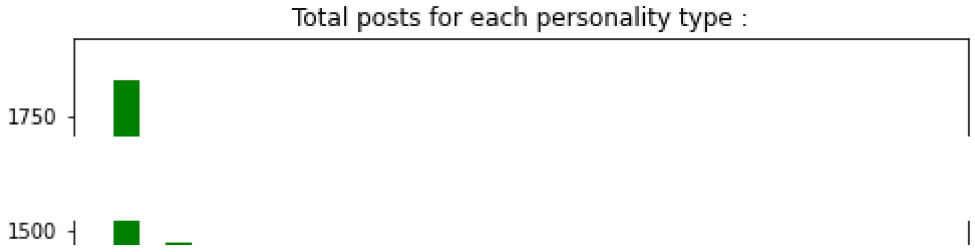
```
['ENFJ' 'ENFP' 'ENTJ' 'ENTP' 'ESFJ' 'ESFP' 'ESTJ' 'ESTP' 'INFJ' 'INFP'
 'INTJ' 'INTP' 'ISFJ' 'ISFP' 'ISTJ' 'ISTP']
```

```
target_dic = dict(df['type'].value_counts())
```

```
target_dic.keys()
```

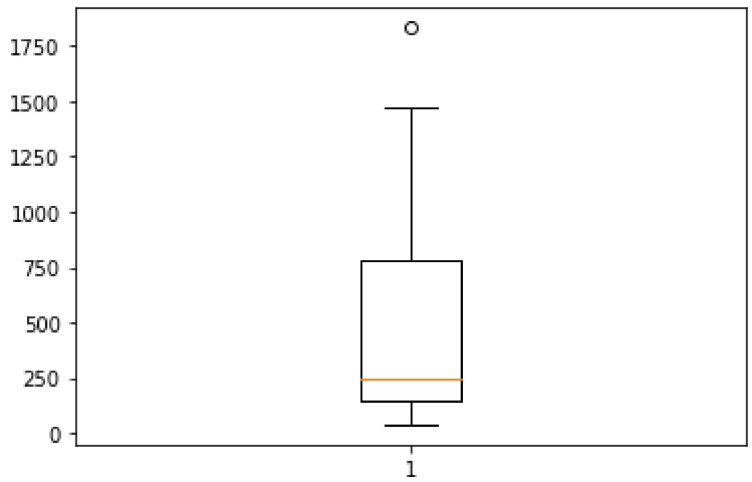
```
dict_keys(['INFP', 'INFJ', 'INTP', 'INTJ', 'ENTP', 'ENFP', 'ISTP', 'ISFP', 'ENTJ', 'ISTJ
```

```
plt.figure(figsize=(8,8))
plt.bar(list(target_dic.keys()), list(target_dic.values()), color='g',width = 0.5)
plt.xlabel('Personality types', size = 14)
plt.ylabel('No. of posts available', size = 14)
plt.title('Total posts for each personality type :')
plt.show()
```




```
# Above we can see that there is great unbalance in Introvert/Extrovert and Intuition/Sensing
# Whereas Feeling/Thinking and Perception/Judgment pairs are quite balanced.
```

```
plt.boxplot(target_dic.values())
plt.show()
```

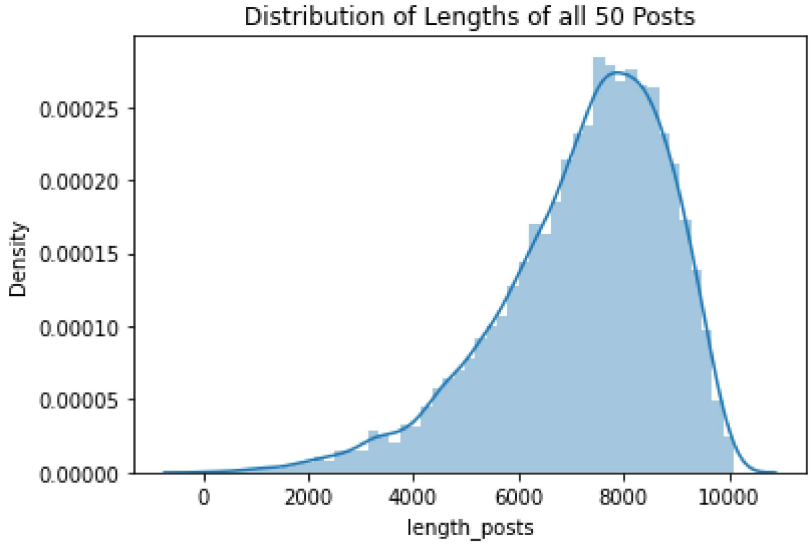



```
plt.figure(figsize=(8,8))
plt.scatter(list(target_dic.keys()), list(target_dic.values()), color='r', alpha=0.5)
```

```
<matplotlib.collections.PathCollection at 0x7f1fe0e9b6d0>
```

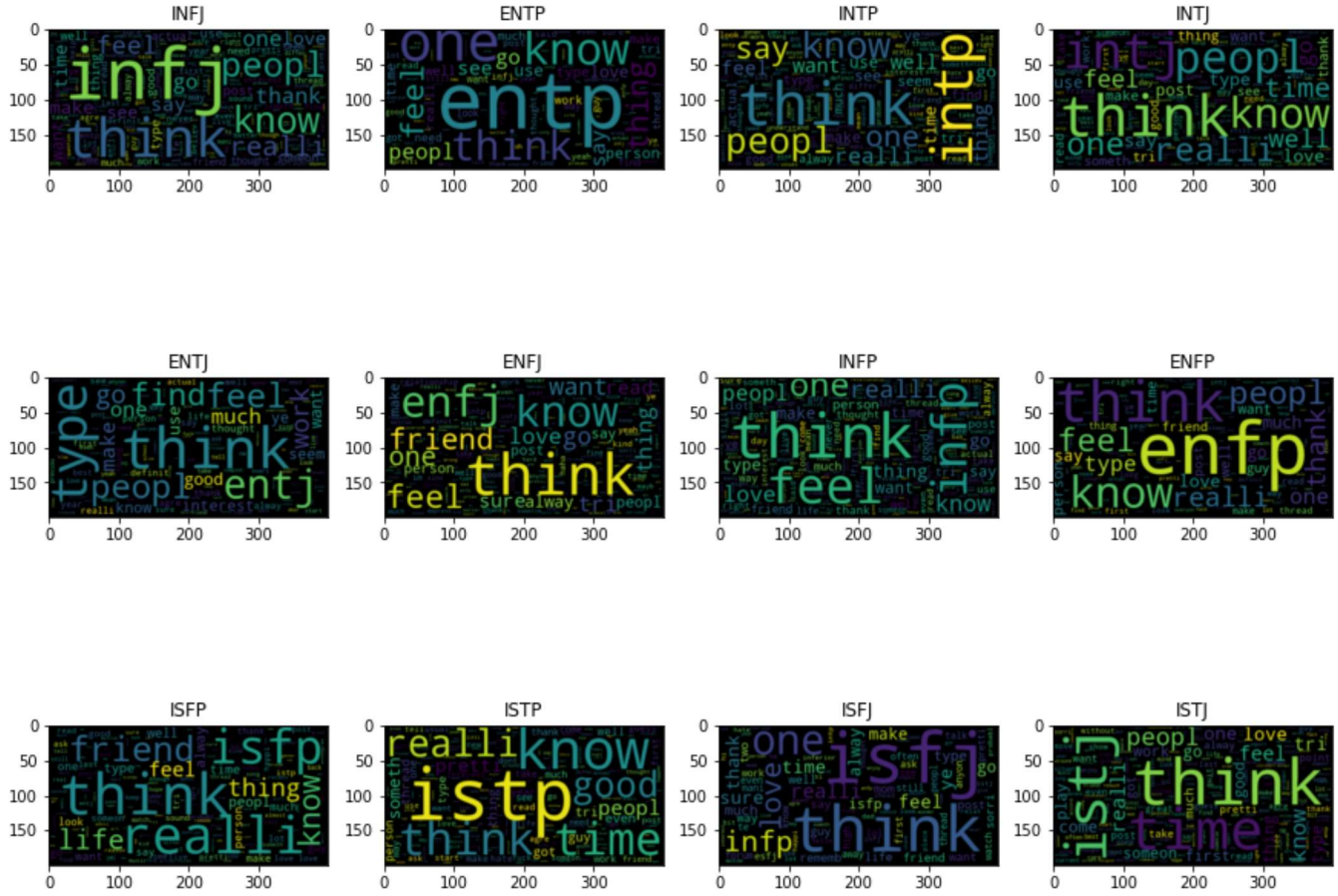

```
df1["length_posts"] = df1["posts"].apply(len)
sb.distplot(df1["length_posts"]).set_title("Distribution of Lengths of all 50 Posts")
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `di
  warnings.warn(msg, FutureWarning)
Text(0.5, 1.0, 'Distribution of Lengths of all 50 Posts')
```




```
from wordcloud import WordCloud
from collections import Counter

#Plotting WordCloud.

#Finding the most common words in all posts.
words = list(df["posts"].apply(lambda x: x.split()))
words = [x for y in words for x in y]

most_common40 = Counter(words).most_common(40)
wc = 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 contain
plt.figure(figsize=(25,10))
# generate word cloud, interpolation
plt.imshow(wc, interpolation='bilinear')
_ = plt.axis("off")
```


```
fig, ax = plt.subplots(len(df['type'].unique()), sharex=True, figsize=(15,len(df['type'].uniq
k = 0
for i in df['type'].unique():
    df_4 = df[df['type'] == i]
    wordcloud = WordCloud(max_words=1628,relative_scaling=1,normalize_plurals=False).generate
    plt.subplot(4,4,k+1)
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.title(i)
    ax[k].axis("off")
    k+=1
```



Converting labels to Numeric Data:

```
df["type"].values
```

```
array(['INFJ', 'ENTP', 'INTP', ..., 'INTP', 'INFP', 'INFP'], dtype=object)
```

```
# Converting MBTI personality (or target or Y feature) into numerical form using Label Encodi
# encoding personality type
from sklearn import preprocessing
enc = preprocessing.LabelEncoder()
df['type of encoding'] = enc.fit_transform(df['type'])

target = df['type of encoding']
```

```
df
```

	type	posts	type of encoding
0	INFJ	intj moment sportscent top ten play prankswhat...	8
1	ENTP	find lack post alarmingsex bore posit often ex...	3
2	INTP	good one cours say know bless cursedo absolut ...	11
3	INTJ	dear intp enjoy convers day esoter gab natur u...	10
4	ENTJ	your firedthat anoth silli misconcept approach...	2
...	...	...	...
8670	ISFP	alway think cat fi dom reason websit becom neo...	13
8671	ENFP	soif thread alreadi exist someplac el heck del...	1
8672	INTP	mani question thing would take purpl pill pick...	11
8673	INFP	conflict right come want child honestli matern...	9
8674	INFP	long sinc personalitycaf although seem chang o...	9

```
from sklearn.feature_extraction.text import CountVectorizer
vect = CountVectorizer()
# Converting posts (or training or X feature) into numerical form by count vectorization
train =  vect.fit_transform(df["posts"])
```

```
train.shape
```

```
(8675, 283192)
```

```
x_train, x_test, y_train, y_test = train_test_split(train, df["type of encoding"], test_size=
print ((x_train.shape),(y_train.shape),(x_test.shape),(y_test.shape))
```

```
(6940, 283192) (6940,) (1735, 283192) (1735,)
```

```
x_train[0]
```

```
<1x283192 sparse matrix of type '<class 'numpy.int64'>'
        with 493 stored elements in Compressed Sparse Row format>
```

```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
```

```
pip install xgboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.7/dist-packages (0.90)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from xgb
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from xgb
```

```
#Metrics
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error, accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, multilabel_confusion_mat
from sklearn.metrics import classification_report


#Models
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier
from xgboost import plot_importance


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))
```

```
Accuracy: 33.49%
```

```
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

print(confusion_matrix(y_test, Y_pred))
```

```
[[ 13   1   1   0   0   0   0   0  10  11   1   0   0   0   0   1]
 [  2  74   1   4   0   0   0   0  11  24  11   3   1   0   1   3]
 [  0   1  11   4   0   0   0   0   4  14   9   2   0   0   0   1]
 [  0   8   0  74   0   0   0   0  13  14  10  16   0   2   0   0]
 [  0   0   0   0   1   0   0   0   6   2   0   0   0   0   0   0]
 [  0   2   1   0   0   0   0   0   0   4   1   2   0   0   0   0]
 [  0   0   0   1   0   0   0   0   1   4   0   1   0   0   0   1]
 [  0   0   0   1   0   0   0   2   4   3   2   5   0   0   0   1]
 [  2   6   0   7   0   0   0   0 197  57   9  14   0   0   0   2]
 [  0   7   1   7   0   0   0   0  24 298   7  19   0   3   0   0]
 [  0   1   1   8   0   0   0   0  21  18 137  29   0   0   2   1]
```

```
       [  0   2   1   7   0   0   0   0   8  21  10 211   0   0   1   0]
       [  0   0   0   1   0   0   0   0   7   7   2   6   8   1   0   1]
       [  1   0   0   3   1   0   0   0   7  21   3   2   0  16   0   0]
       [  0   1   0   1   0   0   0   0   3  13   6   5   0   0  12   0]
       [  0   0   0   2   0   0   0   0   2  13   1  21   0   0   0  28]]
```

```
#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))
```

```
     Accuracy: 65.36%
```

```
from sklearn.svm import SVC
svm = SVC(random_state = 1)
svm.fit(x_train, y_train)

Y_pred = svm.predict(x_test)

predictions = [round(value) for value in Y_pred]
# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
accuracies['SVM'] = accuracy* 100.0
print("Accuracy: %.2f%%" % (accuracy * 100.0))

confusion_matrix(y_test, predictions)
```

```
     Accuracy: 62.36%
```

```
confusion_matrix(y_test, predictions)
```

```
     array([[ 13,   1,   1,   0,   0,   0,   0,   0,  10,  11,   1,   0,   0,
                0,   0,   1],
            [  2,  74,   1,   4,   0,   0,   0,   0,  11,  24,  11,   3,   1,
               0,   1,   3],
            [  0,   1,  11,   4,   0,   0,   0,   0,   4,  14,   9,   2,   0,
               0,   0,   1],
            [  0,   8,   0,  74,   0,   0,   0,   0,  13,  14,  10,  16,   0,
               2,   0,   0],
            [  0,   0,   0,   0,   1,   0,   0,   0,   6,   2,   0,   0,   0,
               0,   0,   0],
            [  0,   2,   1,   0,   0,   0,   0,   0,   0,   4,   1,   2,   0,
               0,   0,   0],
            [  0,   0,   0,   1,   0,   0,   0,   0,   1,   4,   0,   1,   0,
```

```
    0,   0,   1],
[   0,   0,   0,   1,   0,   0,   0,   2,   4,   3,   2,   5,   0,
    0,   0,   1],
[   2,   6,   0,   7,   0,   0,   0,   0, 197,  57,   9,  14,   0,
    0,   0,   2],
[   0,   7,   1,   7,   0,   0,   0,   0,  24, 298,   7,  19,   0,
    3,   0,   0],
[   0,   1,   1,   8,   0,   0,   0,   0,  21,  18, 137,  29,   0,
    0,   2,   1],
[   0,   2,   1,   7,   0,   0,   0,   0,   8,  21,  10, 211,   0,
    0,   1,   0],
[   0,   0,   0,   1,   0,   0,   0,   0,   7,   7,   2,   6,   8,
    1,   0,   1],
[   1,   0,   0,   3,   1,   0,   0,   0,   7,  21,   3,   2,   0,
   16,   0,   0],
[   0,   1,   0,   1,   0,   0,   0,   0,   3,  13,   6,   5,   0,
    0,  12,   0],
[   0,   0,   0,   2,   0,   0,   0,   0,   2,  13,   1,  21,   0,
    0,   0,  28]])
```

✓ 0s completed at 15:02