MACHINE LEARNING

MYERS BRIGG PERSONALITY PREDICTION

PHASE 2

Members

Sethulakshmi Santhosh - AM.EN.U4CSE20362

Meenakshi M - AM.EN.U4CSE20342

Aishwarya K - AM.EN.U4CSE20303

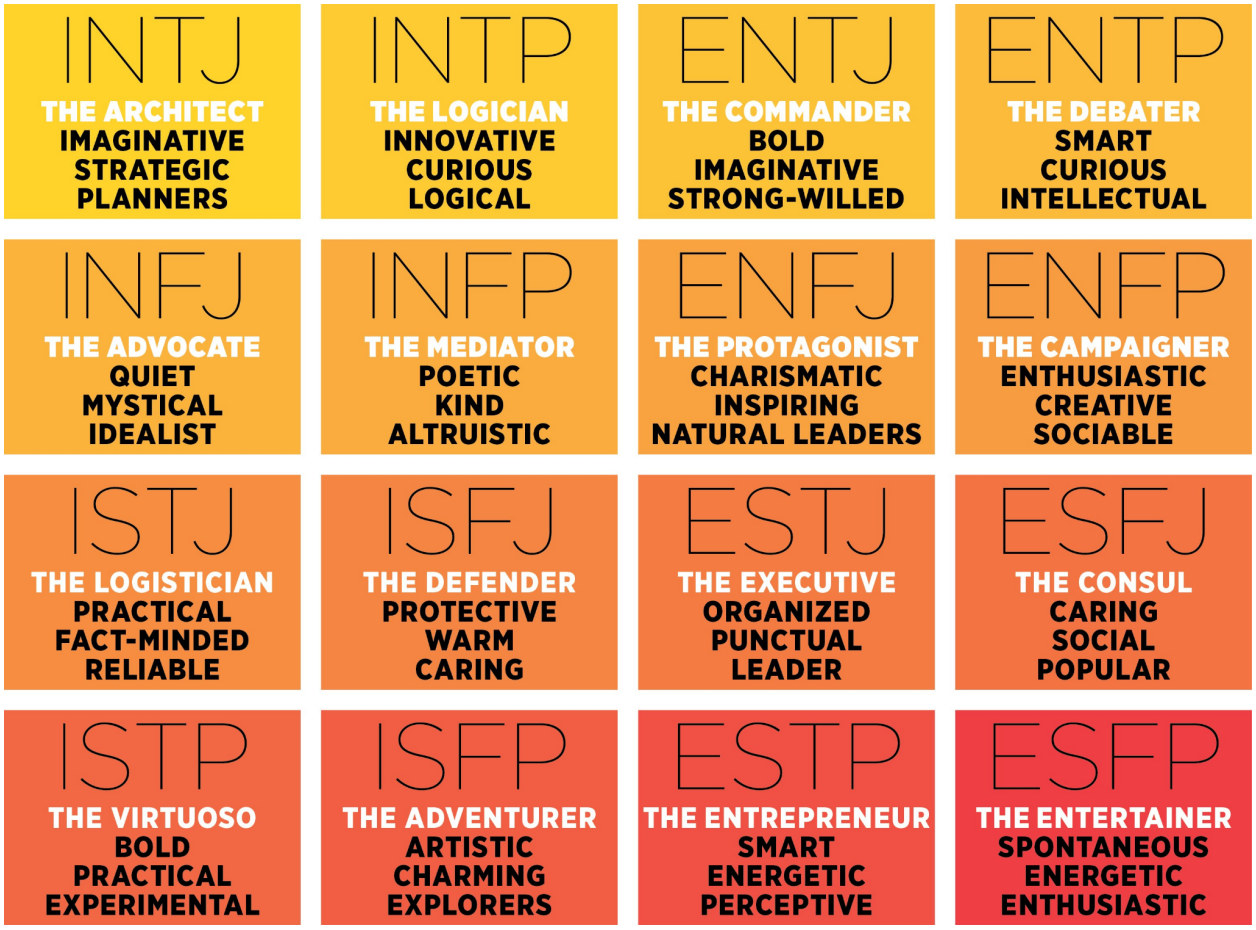
Nandini S Kumar - AM.EN.U4CSE20346

Avani Farida - AM.EN.U4CSE20314

I. Problem Definition

Based on the relationship between people's writing styles and their psychological personalities, the aim of this study is to forecast personality types as one of the sixteen categories of Myers Briggs personality types (MBTI). We think that because social media allows people a place to freely and publicly express themselves, the postings people make there can be a good indication of their personality.

We recognise the equality of all personality types.



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)

II. Datasets

Dataset 1:
Link to dataset- https://www.kaggle.com/datasets/datasnaek/mbti-type

- This dataset contains over 8600 rows of data, on each row is a person's:
- Type (This persons 4 letter MBTI code/type)
- A section of each of the last 50 things they have posted online in social media platforms (Each entry separated by "|||" (3 pipe characters))

Dataset 2:
Link to dataset-
https://www.kaggle.com/datasets/zeyadkhalid/mbti-personality-types-500-dataset

106K records of preprocessed posts and their authors' personality types,
Posts are equal-sized: 500 words per sample.

- Type (This persons 4 letter MBTI code/type)
- Entries posted by users on Reddit are pre collected using Google big query and records of posts collected from PersonalityCafe forum.

Dataset 3:
Link to dataset- https://www.kaggle.com/code/rkuo2000/mbti-lstm/data

This dataset contains unclean data comprising of urls, stopwords, entry separated by "|||" etc. The data consists of two columns-

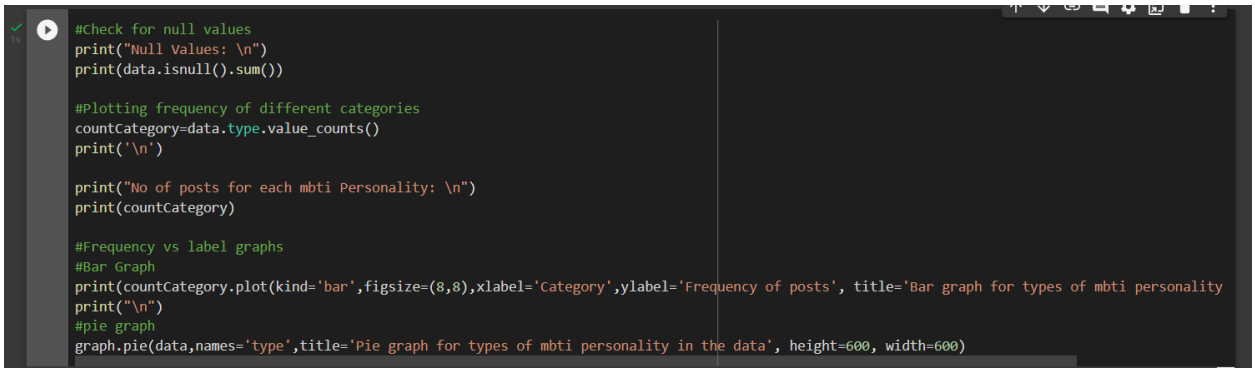
- Type (This persons 4 letter MBTI code/type)
- Entries posted by users on Twitter containing various unwanted signs nd symbols.

III. Prepare Data

Preparing raw data to be acceptable for a machine learning model is known as data preparation. In order to build a machine learning model, it is the first and most important stage.

It is not always the case that we come across clean and prepared data when developing a machine learning project. Additionally, any time you work with data, you must clean it up and format it. Therefore, we use a data pretreatment activity for this.

Real-world data typically includes noise, missing values, and may be in an undesirable format, making it impossible to build machine learning models on it directly. Data preprocessing is necessary to clean the data and prepare it for a machine learning model, which also improves the model's accuracy and effectiveness.



```
#Check for null values
print("Null Values: \n")
print(data.isnull().sum())

#Plotting frequency of different categories
countCategory=data.type.value_counts()
print('\n')

print("No of posts for each mbti Personality: \n")
print(countCategory)

#Frequency vs label graphs
#Bar Graph
print(countCategory.plot(kind='bar',figsize=(8,8),xlabel='Category',ylabel='Frequency of posts', title='Bar graph for types of mbti personality
print("\n")
#pie graph
graph.pie(data,names='type',title='Pie graph for types of mbti personality in the data', height=600, width=600)
```

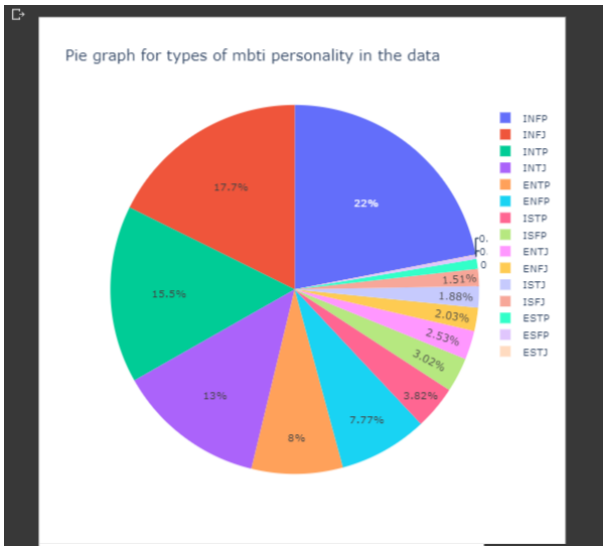
Summarization :

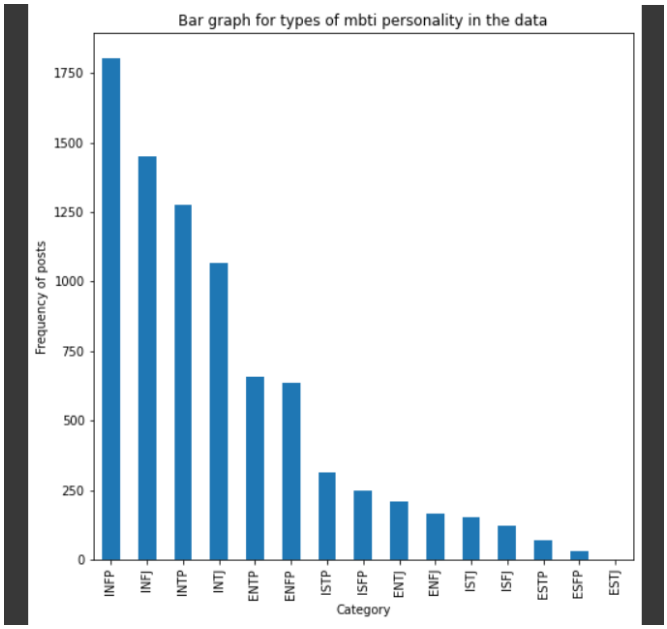
Summarizing the null values found and the number of records/posts for each MBTI personality type.

```
Null Values:

type     0
posts    0
dtype: int64


No of posts for each mbti Personality:

INFP    1804
INFJ    1451
INTP    1276
INTJ    1068
ENTP     657
ENFP     638
ISTP     314
ISFP     248
ENTJ     208
ENFJ     167
ISTJ     154
ISFJ     124
ESTP      70
ESFP      29
ESTJ       2
Name: type, dtype: int64
AxesSubplot(0.125,0.125;0.775x0.755)
```

Visualization:

Visualizing the amount of different MBTI personality types in the dataset using pie chart and bar graph.





Getting the dataset information and description along with the dimensions it posses-

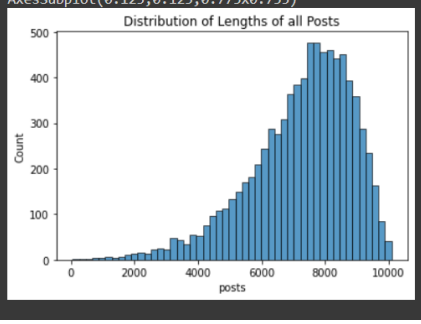
```
[6] #Description about data
    print("Dataset Description: \n", data.describe())
    print("\n")
    #Info about data
    print("Datset Info: \n", data.info())
    print("\n")

    #Shape data
    print("Dimensions of Dataset: ",data.shape)
    print("\n\n")
```

```
Dataset Description:
          type                                               posts
count   8210                                               8210
unique    15                                               8210
top     INFP  'More like calling close to 10 times from 1 am...
freq    1804                                                  1


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8210 entries, 0 to 8209
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   type    8210 non-null   object
 1   posts   8210 non-null   object
dtypes: object(2)
memory usage: 128.4+ KB
Datset Info:
 None


Dimensions of Dataset:  (8210, 2)
```

Distribution of the number of posts and their respective lengths with a histogram.

```
[7] #Plot to show the distribution of Lengths of all Posts
    df= data["posts"].apply(len)
    graph=sns.histplot(df)
    graph.set_title("Distribution of Lengths of all Posts")
    print(graph)
```

```
AxesSubplot(0.125,0.125;0.775x0.755)
```



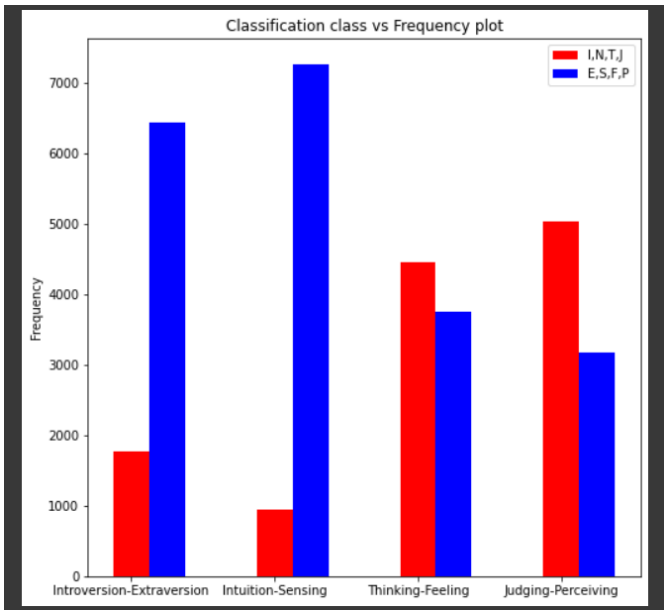
To examine the proportionality of each of the sixteen personality types, Matplotlib was used to plot the value counts of each of these sixteen types. Since the classes were heavily imbalanced, following steps were taken:

- First step was to divide the single “type” feature into four features:
    - Extroversion vs. Introversion
        - I - 0
        - E - 1
    - Sensing vs. Intuition
        - N - 0

        - S - 1
    - Thinking vs. Feeling
        - F - 0
        - T - 1
    - Judging vs. Perceiving
        - P - 0
        - J - 1

Accordingly we observe the 4 type indicators added and visualize as follows.

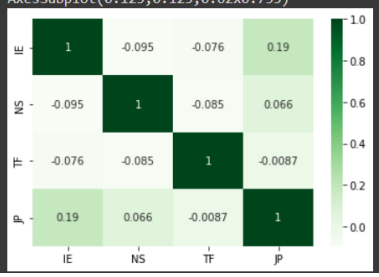
```
[9]  # Plotting Classification class vs Frequency plot for IE, NS, TF, JP

     I,N,T,J=data_copy['IE'].value_counts()[0],data_copy['NS'].value_counts()[0],data_copy['TF'].value_counts()[0],data_copy['JP'].value_counts()[0]
     E,S,F,P=data_copy['IE'].value_counts()[1],data_copy['NS'].value_counts()[1],data_copy['TF'].value_counts()[1],data_copy['JP'].value_counts()[1]

     plt.figure(figsize=(8,8))

     coordinates=np.array(range(0,4))*2
     plt.bar(coordinates-0.25,(I,N,T,J),0.5,label="I,N,T,J",color='red',tick_label=['Introversion-Extraversion','Intuition-Sensing','Thinking-Feeling
     plt.bar(coordinates+0.25,(E,S,F,P),0.5,label="E,S,F,P",color='blue')

     plt.legend()
     plt.title('Classification class vs Frequency plot')
     plt.ylabel('Frequency')
     plt.show()
```



Finding a correlation matrix for the 4 type indicators created and the corresponding heatmap.

```
[10] #Correlation matrix and Heatmap Visuals

     columns=['IE','NS','TF','JP']
     matrix=data_copy[columns].corr()

     print('Correlation Matrix:\n',matrix)
     print('\nHeatMap Visual:\n')
     print(sns.heatmap(matrix,annot=True,cmap='Greens'))
```

```
Correlation Matrix:
          IE        NS        TF        JP
IE  1.000000 -0.094805 -0.076270  0.187097
NS -0.094805  1.000000 -0.084669  0.065791
TF -0.076270 -0.084669  1.000000 -0.008722
JP  0.187097  0.065791 -0.008722  1.000000

HeatMap Visual:

AxesSubplot(0.125,0.125;0.62x0.755)
```



Preprocessing

The following steps were carried out in preprocessing-

- Posts were converted into lower case.
- ||| and punctuations were replaced by spaces.
- Links and Emails were dropped.
- MBTI personality types were dropped. There were quite a few uses of these types in the posts and we didn't find them adding any value.
- Words with one to two character lengths were dropped.
- The cleaned data generated after executing above steps was Lemmitized using NLTK WordNet Lemmatizer. Stop Words were dropped at this stage.
- The result of these steps stored as "clean posts" was a cleaned and lemmatized set of words for each user.

```
#data before preprocessing
print("Data (post 2) before preprocessing :", data.iloc[2,1], "\n")
#data after preprocessing
print("Data (post 2) after preprocessing :", data_copy.iloc[2,1], "\n")
```

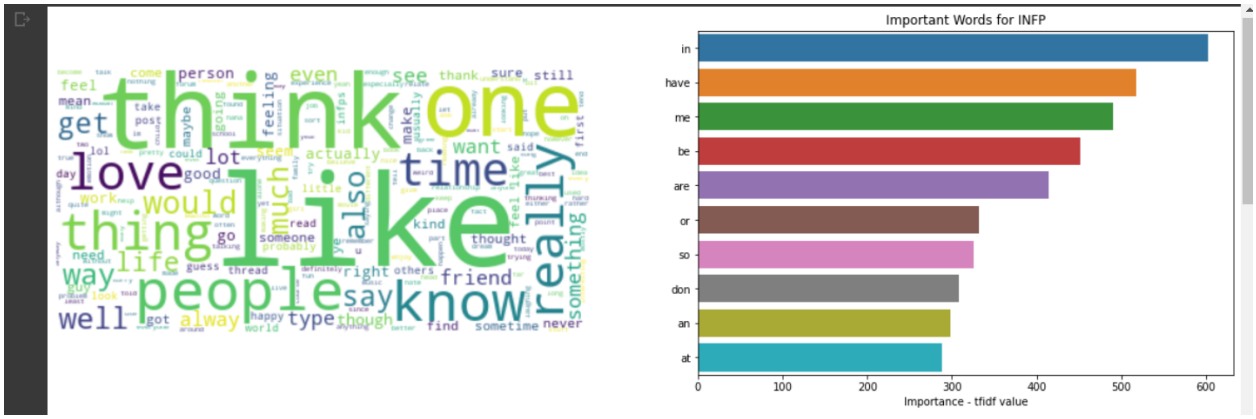
```
Data (post 2) before preprocessing : 'On paper, I know that. Several people have said the same thing, and reinforcement is probably what I'm looking for. Thank

Data (post 2) after preprocessing :  paper know several people said thing reinforcement probably looking thank never felt pain like dating male three year livi
```

Visualizing data after preprocessing

```
[16] from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

     def show_type_properties(types):
       extracted_data=data_copy[data['type']==types]
       texts=" ".join(extracted_data['posts'].tolist())
       worldcount=WordCloud(background_color="white", max_words=200,
                            stopwords=remove_words, contour_width=3)
       worldcount.generate(texts)
       #Plot
       plt.figure(figsize=[20,6])
       plt.subplot(121)
       plt.imshow(worldcount, interpolation='bilinear')
       plt.axis("off")
       plt.subplot(122)
       tuples = imp_words_dict[types]
       words = [x[0] for x in tuples]
       imp = [x[1] for x in tuples]
       sns.barplot(y = words,x = imp)
       plt.title(f'Important Words for {types}')
       plt.xlabel('Importance - tfidf value')
```

```
[17] for i in countCategory.index:
       show_type_properties(i)
```

Sample instance of one ‘i’ output-



The above shows the frequency of certain words.

IV. Python Packages

```
#download
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')

#importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
import time
import plotly.express as graph
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.feature_extraction.text import CountVectorizer,TfidfTransformer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.svm import SVC
```

- ***nltk:*** The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing for English written in the Python programming language.
- ***Nltk.corpus import stopwords:*** A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query. We would not want these words to take up space in our database, or taking up valuable processing time. For this, we can remove them easily, by storing a list of words that you consider to stop words. NLTK(Natural Language Toolkit) in python has a list of stopwords stored in 16 different languages.
- ***Pandas:*** Pandas is mainly used for data analysis and associated manipulation of tabular data in DataFrames. Pandas allows importing data from various file formats such as comma-separated values, JSON, Parquet, SQL database tables or queries, and Microsoft Excel.
- ***NumPy***:NumPy stands for Numerical Python and it is a core scientific computing library in Python. It provides efficient multi-dimensional array objects and various operations to work with these array objects.

- ***Matplotlib:***Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible. Create publication quality plots. Make interactive figures that can zoom, pan, update.
- ***Seaborn:***Seaborn is a library that uses Matplotlib underneath to plot graphs. It will be used to visualize random distributions.
- ***re:***A regular expression (or RE) specifies a set of strings that matches it; the functions in this module let you check if a particular string matches a given regular expression
- ***time:*** python time module provides many ways of representing time in code, such as objects, numbers, and strings. It also provides functionality other than representing time, like waiting during code execution and measuring the efficiency of your code.
- ***plotly.express:***plotly.express module (usually imported as px) contains functions that can create entire figures at once, and is referred to as Plotly Express or PX. Plotly Express is a built-in part of the plotly library, and is the recommended starting point for creating most common figures.
- ***Scikit:***Scikit-learn (sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python.
- ***sklearn. metrics:***The sklearn. metrics module implements several loss, score, and utility functions to measure classification performance. Some metrics might require probability estimates of the positive class, confidence values, or binary decisions values.
- ***roc_curve:*** The ROC curve is used to assess the overall diagnostic performance of a test and to compare the performance of two or more diagnostic tests. It is also used to select an optimal cut-off value for determining the presence or absence of a disease.
- ***roc_auc_score:***roc_auc_score is defined as the area under the ROC curve, which is the curve having False Positive Rate on the x-axis and True Positive Rate on the y-axis at all classification thresholds.
- ***CountVectorizer,Tfidftransformer:*** CountVectorizer performs the task of tokenizing and counting, while TfidfTransformer normalizes the data. TfidfVectorizer, on the other hand,

performs all three operations, thereby streamlining the process of natural language processing

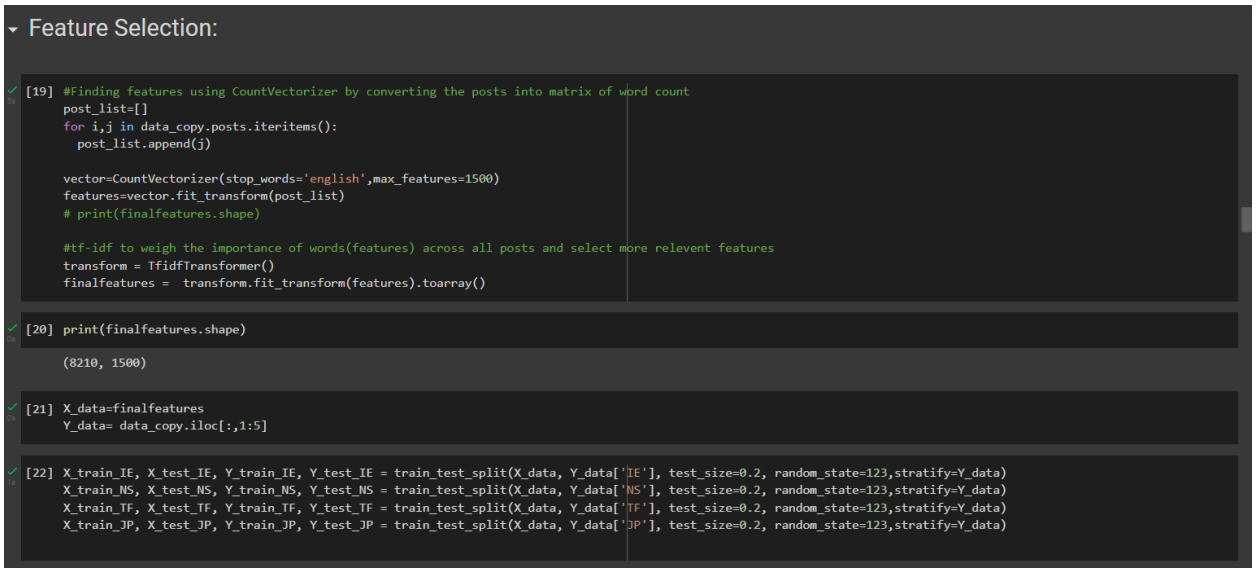
- ***train_test_split:*** train_test_split is a function in Sklearn model selection for splitting data arrays into two subsets: for training data and for testing data. With this function, you don't need to divide the dataset manually. By default, Sklearn train_test_split will make random partitions for the two subsets.
- ***accuracy_score:*** In multilabel classification, this function computes subset accuracy: the set of labels predicted for a sample must exactly match the corresponding set of labels in y_true.
- ***GridSearchCV:*** GridSearchCV tries all the combinations of the values passed in the dictionary and evaluates the model for each combination using the Cross-Validation method. Hence after using this function we get accuracy/loss for every combination of hyperparameters and we can choose the one with the best performance.
- ***classification_report:*** It is a performance evaluation metric in machine learning which is used to show the precision, recall, F1 Score, and support score of your trained classification model.
- ***GuassianNB:*** As the name suggest, Gaussian Naïve Bayes classifier assumes that the data from each label is drawn from a simple Gaussian distribution. The Scikit-learn provides sklearn. naive_bayes. GaussianNB to implement the Gaussian Naïve Bayes algorithm for classification.
- ***RandomForestClassifier:*** From there, the random forest classifier can be used to solve for regression or classification problems.
- ***LogisticRegression:*** Logistic Regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.).
- ***KNeighborsClassifier:*** KNeighborsClassifier implements classification based on voting by nearest k-neighbors of target point, t.
- ***SGDClassifier:*** Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to fitting linear classifiers and regressors under convex loss functions such as (linear) Support Vector Machines and Logistic Regression.

- ***SVC:*** SVC, or Support Vector Classifier, is a supervised machine learning algorithm typically used for classification tasks. SVC works by mapping data points to a high-dimensional space and then finding the optimal hyperplane that divides the data into two classes.

V. Learning Algorithms

Each of the four categories needs to be encoded as follows:

- Mind: I = 0, E = 1
- Energy: S = 0, N = 1
- Nature: F = 0, T = 1
- Tactics: P = 0, J = 1

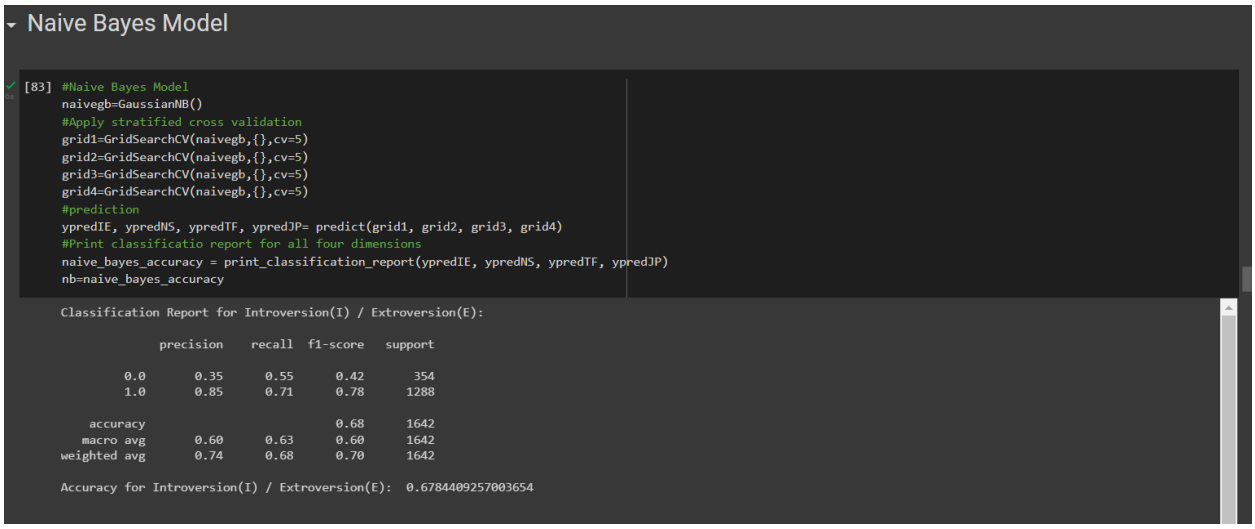


For each we make use of ‘cv’ pertaining to cross validation by specifying the number of folds along with it. Here we set it as 5 for each model.

1. Naive Bayes Model

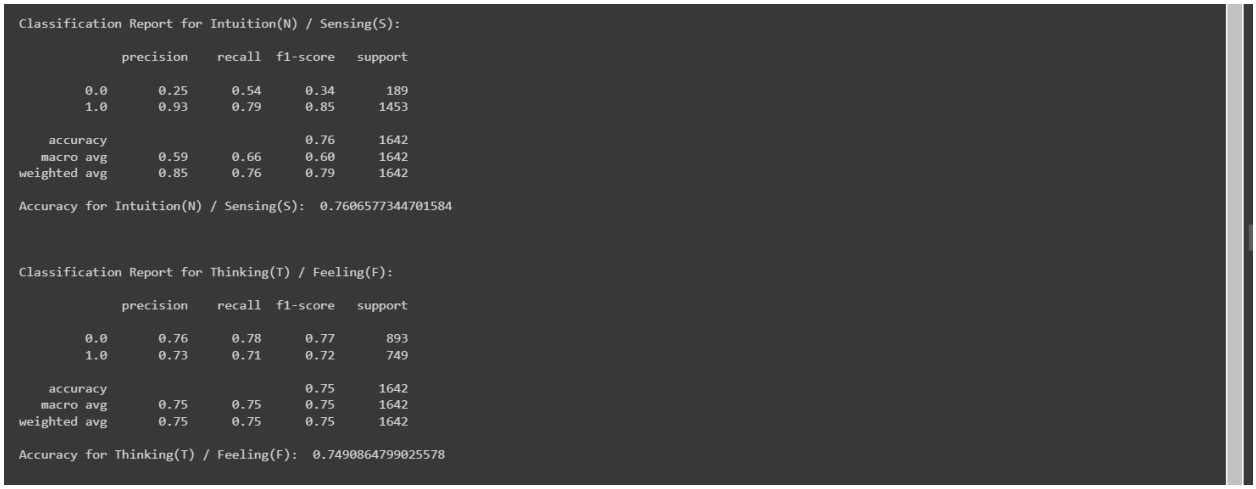
   A group of classification algorithms built on the Bayes' Theorem are known as naive Bayes classifiers. It is a family of algorithms rather than a single method, and they are all based on the idea that every pair of features being classified is independent of the other.

The majority of applications for naive Bayes algorithms include sentiment analysis, spam filtering, recommendation systems, etc. Although they are quick and simple to use, their major drawback is the need for independent predictors. The classifier performs worse when the predictors are dependent, which occurs in the majority of real-world situations.



Naive Bayes Model

```
[83] #Naive Bayes Model
     naivegb=GaussianNB()
     #Apply stratified cross validation
     grid1=GridSearchCV(naivegb,{},cv=5)
     grid2=GridSearchCV(naivegb,{},cv=5)
     grid3=GridSearchCV(naivegb,{},cv=5)
     grid4=GridSearchCV(naivegb,{},cv=5)
     #prediction
     ypredIE, ypredNS, ypredTF, ypredJP= predict(grid1, grid2, grid3, grid4)
     #Print classificatio report for all four dimensions
     naive_bayes_accuracy = print_classification_report(ypredIE, ypredNS, ypredTF, ypredJP)
     nb=naive_bayes_accuracy

     Classification Report for Introversion(I) / Extroversion(E):

                   precision    recall  f1-score   support

              0.0       0.35      0.55      0.42       354
              1.0       0.85      0.71      0.78      1288

         accuracy                           0.68      1642
        macro avg       0.60      0.63      0.60      1642
     weighted avg       0.74      0.68      0.70      1642

     Accuracy for Introversion(I) / Extroversion(E):  0.6784409257003654
```



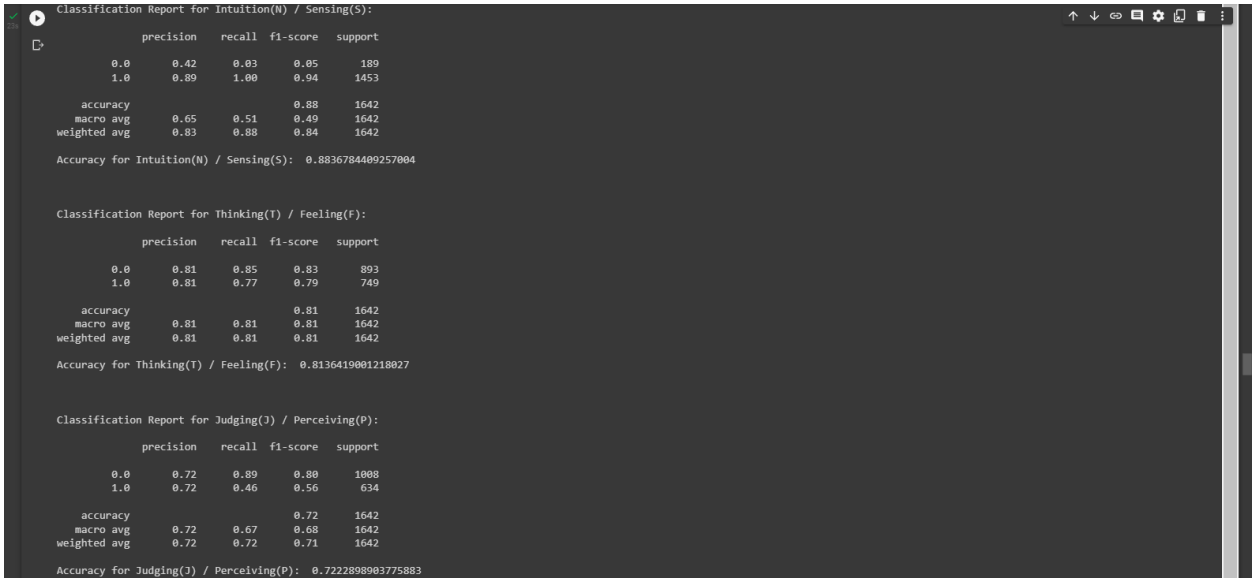
```
Classification Report for Intuition(N) / Sensing(S):

              precision    recall  f1-score   support

         0.0       0.25      0.54      0.34       189
         1.0       0.93      0.79      0.85      1453

    accuracy                           0.76      1642
   macro avg       0.59      0.66      0.60      1642
weighted avg       0.85      0.76      0.79      1642

Accuracy for Intuition(N) / Sensing(S):  0.7606577344701584


Classification Report for Thinking(T) / Feeling(F):

              precision    recall  f1-score   support

         0.0       0.76      0.78      0.77       893
         1.0       0.73      0.71      0.72       749

    accuracy                           0.75      1642
   macro avg       0.75      0.75      0.75      1642
weighted avg       0.75      0.75      0.75      1642

Accuracy for Thinking(T) / Feeling(F):  0.7490864799025578
```

```
Classification Report for Judging(J) / Perceiving(P):

              precision    recall  f1-score   support

         0.0       0.74      0.69      0.71      1008
         1.0       0.56      0.62      0.58       634

    accuracy                           0.66      1642
   macro avg       0.65      0.65      0.65      1642
weighted avg       0.67      0.66      0.66      1642

Accuracy for Judging(J) / Perceiving(P):  0.6613885505481121
```

2. Logistic Regression

Logistic regression is a type of machine learning algorithm that is used to predict the categorical dependent variable (e.g. whether an individual is male or female) based on a set of independent variables (e.g. age, height, etc.). The algorithm produces probabilistic values (between 0 and 1) that represent the likelihood that the individual is male or female. In logistic regression, a logit transformation is applied on the odds—that is, the probability of success divided by the probability of failure. This is also commonly known as the log odds, or the natural logarithm of odds.

```
[84] #Logistic Regression model
     log =LogisticRegression(max_iter=500)
     #Apply stratified cross validation
     grid1= GridSearchCV(log,{},cv=5)
     grid2= GridSearchCV(log,{},cv=5)
     grid3= GridSearchCV(log,{},cv=5)
     grid4= GridSearchCV(log,{},cv=5)
     #prediction
     ypredIE, ypredNS, ypredTF, ypredJP= predict(grid1, grid2, grid3, grid4)
     #Print classificatio report for all four dimensions
     logistic_regression_accuracy = print_classification_report(ypredIE, ypredNS, ypredTF, ypredJP)
     lr=logistic_regression_accuracy

     Classification Report for Introversion(I) / Extroversion(E):

                   precision    recall  f1-score   support

              0.0       0.80      0.19      0.31       354
              1.0       0.82      0.99      0.89      1288

         accuracy                           0.81      1642
        macro avg       0.81      0.59      0.60      1642
     weighted avg       0.81      0.81      0.77      1642

     Accuracy for Introversion(I) / Extroversion(E):  0.8148599269183922
```



```
Classification Report for Intuition(N) / Sensing(S):

              precision    recall  f1-score   support

         0.0       0.42      0.03      0.05       189
         1.0       0.89      1.00      0.94      1453

    accuracy                           0.88      1642
   macro avg       0.65      0.51      0.49      1642
weighted avg       0.83      0.88      0.84      1642

Accuracy for Intuition(N) / Sensing(S):  0.8836784409257004


Classification Report for Thinking(T) / Feeling(F):

              precision    recall  f1-score   support

         0.0       0.81      0.85      0.83       893
         1.0       0.81      0.77      0.79       749

    accuracy                           0.81      1642
   macro avg       0.81      0.81      0.81      1642
weighted avg       0.81      0.81      0.81      1642

Accuracy for Thinking(T) / Feeling(F):  0.8136419001218027


Classification Report for Judging(J) / Perceiving(P):

              precision    recall  f1-score   support

         0.0       0.72      0.89      0.80      1008
         1.0       0.72      0.46      0.56       634

    accuracy                           0.72      1642
   macro avg       0.72      0.67      0.68      1642
weighted avg       0.72      0.72      0.71      1642

Accuracy for Judging(J) / Perceiving(P):  0.7222898903775883
```

3. KNN Classifier

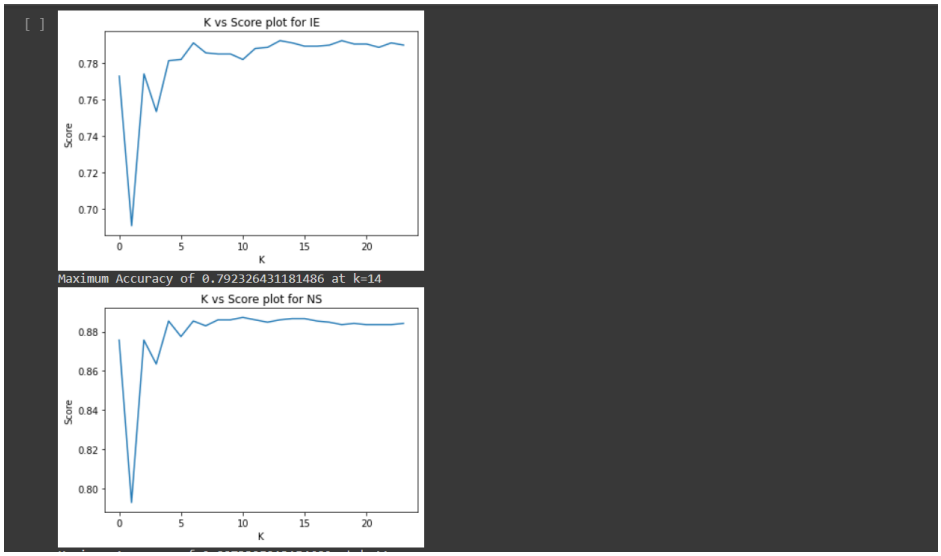
K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm. K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems. K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data. It's also worth noting that the KNN algorithm is also part of a family of “lazy learning” models, meaning that it only stores a training dataset versus undergoing a training stage. This also means that all the computation occurs when a classification or prediction is being made. Since it heavily relies on memory to store all its training data, it

is also referred to as an instance-based or memory-based learning method.

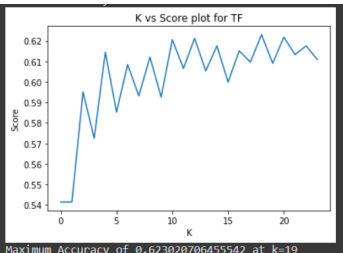
KNN Classifier

```
[ ]  #K-Nearest Neighbor Model

     '''Below we find the best value of K for our model.
        Analysis of accuracy vs K graph gives us the best value of K
        Analysis of accuracy vs K graph are plotted.'''
     def plot_graph(val,xlab,ylab,title):
       plt.figure()
       plt.plot(val)
       plt.ylabel(ylab)
       plt.xlabel(xlab);
       plt.title(title)
       plt.show()

     #IE prediction
     k1_score=[]
     for i in range(1,25):
       k1=KNeighborsClassifier(n_neighbors=i)
       k1.fit(X_train_IE,Y_train_IE)
       k1_score.append(k1.score(X_test_IE,Y_test_IE))

     df1=pd.DataFrame(k1_score,columns=['IE'])
     plot_graph(k1_score,'K','Score','K vs Score plot for IE')
     print(f'Maximum Accuracy of {df1.max()[0]} at k={df1.idxmax()[0]+1}')
```

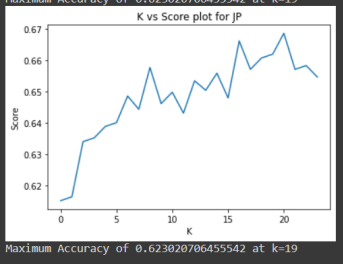
```
     #NS prediction
     k2_score=[]
     for i in range(1,25):
       k2=KNeighborsClassifier(n_neighbors=i)
       k2.fit(X_train_NS,Y_train_NS)
       k2_score.append(k2.score(X_test_NS,Y_test_NS))

     df2=pd.DataFrame(k2_score,columns=['NS'])
     plot_graph(k2_score,'K','Score','K vs Score plot for NS')
     print(f'Maximum Accuracy of {df2.max()[0]} at k={df2.idxmax()[0]+1}')

     #TF prediction
     k3_score=[]
     for i in range(1,25):
       k3=KNeighborsClassifier(n_neighbors=i)
       k3.fit(X_train_TF,Y_train_TF)
       k3_score.append(k3.score(X_test_TF,Y_test_TF))

     df3=pd.DataFrame(k3_score,columns=['TF'])
     plot_graph(k3_score,'K','Score','K vs Score plot for TF')
     print(f'Maximum Accuracy of {df3.max()[0]} at k={df3.idxmax()[0]+1}')

     #JP prediction
     k4_score=[]
     for i in range(1,25):
       k4=KNeighborsClassifier(n_neighbors=i)
       k4.fit(X_train_JP,Y_train_JP)
       k4_score.append(k4.score(X_test_JP,Y_test_JP))

     df4=pd.DataFrame(k3_score,columns=['JP'])
     plot_graph(k4_score,'K','Score','K vs Score plot for JP')
     print(f'Maximum Accuracy of {df4.max()[0]} at k={df4.idxmax()[0]+1}')
```




```
Maximum Accuracy of 0.792326431181486 at k=14
```


```
Maximum Accuracy of 0.623020706455542 at k=19
```




```
Maximum Accuracy of 0.623020706455542 at k=19
```

```
#Print KNN Results
knn_accuracy = print_classification_report(ypredIE, ypredNS, ypredTF, ypredJP)
knn=knn_accuracy
```

```
Classification Report for Introversion(I) / Extroversion(E):

              precision    recall  f1-score   support

         0.0       0.80      0.19      0.31       354
         1.0       0.82      0.99      0.89      1288

    accuracy                           0.81      1642
   macro avg       0.81      0.59      0.60      1642
weighted avg       0.81      0.81      0.77      1642

Accuracy for Introversion(I) / Extroversion(E):  0.8148599269183922


Classification Report for Intuition(N) / Sensing(S):

              precision    recall  f1-score   support

         0.0       0.42      0.03      0.05       189
         1.0       0.89      1.00      0.94      1453

    accuracy                           0.88      1642
   macro avg       0.65      0.51      0.49      1642
weighted avg       0.83      0.88      0.84      1642

Accuracy for Intuition(N) / Sensing(S):  0.8836784409257004
```

```
Classification Report for Thinking(T) / Feeling(F):

              precision    recall  f1-score   support

         0.0       0.81      0.85      0.83       893
         1.0       0.81      0.77      0.79       749

    accuracy                           0.81      1642
   macro avg       0.81      0.81      0.81      1642
weighted avg       0.81      0.81      0.81      1642

Accuracy for Thinking(T) / Feeling(F):  0.8136419001218027


Classification Report for Judging(J) / Perceiving(P):

              precision    recall  f1-score   support

         0.0       0.72      0.89      0.80      1008
         1.0       0.72      0.46      0.56       634

    accuracy                           0.72      1642
   macro avg       0.72      0.67      0.68      1642
weighted avg       0.72      0.72      0.71      1642

Accuracy for Judging(J) / Perceiving(P):  0.7222898903775883
```

4. SGD Classifier

SGD Classifier is a linear classifier (SVM, logistic regression, a.o.) optimized by the SGD. SGD allows minibatch (online/out-of-core) learning. Therefore, it makes sense to use SGD for large scale problems where it's very efficient. The minimum of the cost function of Logistic Regression cannot be calculated directly, so we try to minimize it via Stochastic Gradient Descent, also known as Online Gradient Descent. In this process we descend along the cost function towards its minimum (please have a look at the diagram above) for each training observation we encounter. Another reason to use SGD Classifier is that SVM or logistic regression will not work if you cannot keep the record in RAM. However, SGD Classifier continues to work.

SGD Classifier:

```
#SGDClassifier Model
sgd_model= SGDClassifier(loss='log')
#Apply cross validation
grid1=GridSearchCV(sgd_model,{},cv=5)
grid2=GridSearchCV(sgd_model,{},cv=5)
grid3=GridSearchCV(sgd_model,{},cv=5)
grid4=GridSearchCV(sgd_model,{},cv=5)
#prediction
ypredIE, ypredNS, ypredTF, ypredJP= predict(grid1, grid2, grid3, grid4)
#Print classificatio report for all four axis
sgd_accuracy = print_classification_report(ypredIE, ypredNS, ypredTF, ypredJP)
sgd=sgd_accuracy
```

```
[ ] Classification Report for Introversion(I) / Extroversion(E):

                precision    recall  f1-score   support

           0.0       0.82      0.17      0.29       354
           1.0       0.81      0.99      0.89      1288

      accuracy                           0.81      1642
     macro avg       0.82      0.58      0.59      1642
  weighted avg       0.82      0.81      0.76      1642

  Accuracy for Introversion(I) / Extroversion(E):  0.8136419001218027


  Classification Report for Intuition(N) / Sensing(S):

                precision    recall  f1-score   support

           0.0       0.44      0.02      0.04       189
           1.0       0.89      1.00      0.94      1453

      accuracy                           0.88      1642
     macro avg       0.67      0.51      0.49      1642
  weighted avg       0.84      0.88      0.84      1642

  Accuracy for Intuition(N) / Sensing(S):  0.8842874543239951
```

```
Classification Report for Thinking(T) / Feeling(F):

              precision    recall  f1-score   support

         0.0       0.81      0.85      0.83       893
         1.0       0.81      0.76      0.79       749

    accuracy                           0.81      1642
   macro avg       0.81      0.81      0.81      1642
weighted avg       0.81      0.81      0.81      1642

Accuracy for Thinking(T) / Feeling(F):  0.8099878197320342


Classification Report for Judging(J) / Perceiving(P):

              precision    recall  f1-score   support

         0.0       0.74      0.82      0.78      1008
         1.0       0.66      0.54      0.60       634

    accuracy                           0.71      1642
   macro avg       0.70      0.68      0.69      1642
weighted avg       0.71      0.71      0.71      1642

Accuracy for Judging(J) / Perceiving(P):  0.7149817295980512
```

V. Comparisons

We now compare the various models used in the project with their average accuracy of the 4 different models generated for each algorithm. Hence, from the mean value for each algorithm we plot a pie chart to compare the performance of the same.

We see that Logistic Regression and KNN provides the same amount of accuracy whereas Naive Bayes Model provides the least accuracy.

We have now derived the following percentage of accuracy measures-

- Logistic Regression - 25.79%
- KNN - 29.7%
- SGD - 25.70%
- Naive Bayes - 22.72%

```
[109] labels = ['Naive\nBayes','Logistic\nRegression','KNN','SGD']
      size = [nb,lr,knn,sgd]
      mycolors = ["#deac7a", "#b38150", "#bd732a", "#9c4f02"]
      plt.pie(size, labels = labels, colors = mycolors,wedgeprops=
            {'edgecolor':'black'},autopct='%1.2f%%')

      plt.show()
```