Importing required packages

In [1]:

```
import pandas as pd                                          # to import data in and use
as dataframes
import numpy as np
from sklearn.linear_model import LogisticRegression          # used to implement logistic
regression
from sklearn.feature_extraction.text import TfidfVectorizer  # used in creating tf-idf
from sklearn.decomposition import LatentDirichletAllocation  # implementing lda
import nltk                                                  # used in feature engineeri
ng remove stopwords
from nltk.corpus import stopwords                            # importing stopwords
from nltk.tokenize import word_tokenize                      # used in feature engineeri
ng to tokenize words
from nltk.stem import PorterStemmer                          # used in feature engineeri
ng to convert words to their stem
from nltk.stem import WordNetLemmatizer                      # used in feature engineeri
ng to lemmatize words
```

Functions to call

In [2]:

```
def form1(record):
    record = pd.DataFrame.from_dict([record])
    a = part1(record)
    b = part2(record)
    c = part3(record)
    return (a or b or c)
```

In [3]:

```
def form2(dict1):
    score = 0
    for key,value in dict1.items():
        if value == 1:
            score = score + 1
        elif value == 2:
            score = score + 2
        elif value == 3:
            score = score + 3
    return insight(score)
```

In [4]:

```
def first_run():
    df = initial_data()
    list1 = create_list()
    df = refine_text(df,list1)
    df2 = refine_dataframe(df)
    model = model_development(df2)
    vectorizer,lda_model = tf_idf(df)
    return (df,list1,model,vectorizer,lda_model)
```

Feature Engineering

In [5]:

```
def initial_data():
    df = pd.read_excel('training.xlsx')                                    # re
```

```
ad from excel file
    for i in range(26):                                                    # ch
anging column names for better workability
        df.rename(columns = {df.columns[i] : i},inplace = True)
    df.fillna('',inplace = True)                                           # re
move NaN values
    df.replace('','Not comfortable',inplace = True)
    df['words'] = df[3] + ' ' +  df[7] + ' ' + df[18] + ' '+ df[20] + ' ' + df[24] + '
' + df[25]
    return df
```

Creating a list for stopwords out of library and internet

In [6]:

```
def create_list():
    list1 = ['a', 'about', 'above', 'across', 'after', 'afterwards', 'again', 'against',
'all', 'almost', 'alone',
             'along', 'already', 'also', 'although', 'always', 'am', 'among', 'amongst',
'amoungst', 'amount',
             'an', 'and', 'another', 'any', 'anyhow', 'anyone', 'anything', 'anyway', 'a
nywhere', 'are', 'around',
             'as', 'at', 'back', 'be', 'became', 'because', 'become', 'becomes', 'becomi
ng', 'been', 'before',
             'beforehand', 'behind', 'being', 'below', 'beside', 'besides', 'between', '
beyond', 'bill', 'both',
             'bottom', 'but', 'by', 'call', 'can', 'cannot', 'cant', 'co', 'con', 'could
', 'couldnt', 'cry', 'de',
             'describe', 'detail', 'did', 'do', 'does', 'doing', 'don', 'done', 'down',
'due', 'during', 'each', 'eg',
             'eight', 'either', 'eleven', 'else', 'elsewhere', 'empty', 'enough', 'etc',
'even', 'ever', 'every', 'everyone',
             'everything', 'everywhere', 'except', 'few', 'fifteen', 'fify', 'fill', 'fi
nd', 'fire', 'first', 'five', 'for',
             'former', 'formerly', 'forty', 'found', 'four', 'from', 'front', 'full', 'f
urther', 'get', 'give', 'go', 'had',
             'has', 'hasnt', 'have', 'having', 'he', 'hence', 'her', 'here', 'hereafter'
, 'hereby', 'herein', 'hereupon',
             'hers', 'herself', 'him', 'himself', 'his', 'how', 'however', 'hundred', 'i
', 'ie', 'if', 'in', 'inc', 'indeed',
             'interest', 'into', 'is', 'it', 'its', 'itself', 'just', 'keep', 'last', 'l
atter', 'latterly', 'least', 'less',
             'ltd', 'made', 'many', 'may', 'me', 'meanwhile', 'might', 'mill', 'mine', '
more', 'moreover', 'most', 'mostly',
             'move', 'much', 'must', 'my', 'myself', 'name', 'namely', 'neither', 'never
', 'nevertheless', 'next', 'nine',
             'no', 'nobody', 'none', 'noone', 'nor', 'not', 'nothing', 'now', 'nowhere',
'of', 'off', 'often', 'on', 'once',
             'one', 'only', 'onto', 'or', 'other', 'others', 'otherwise', 'our', 'ours',
'ourselves', 'out', 'over', 'own',
             'part', 'per', 'perhaps', 'please', 'put', 'rather', 're', 's', 'same', 'se
e', 'seem', 'seemed', 'seeming',
             'seems', 'serious', 'several', 'she', 'should', 'show', 'side', 'since', 's
incere', 'six', 'sixty', 'so',
             'some', 'somehow', 'someone', 'something', 'sometime', 'sometimes', 'somewh
ere', 'still', 'such', 'system',
             't', 'take', 'ten', 'than', 'that', 'the', 'their', 'theirs', 'them', 'them
selves', 'then', 'thence', 'there',
             'thereafter', 'thereby', 'therefore', 'therein', 'thereupon', 'these', 'the
y', 'thickv', 'thin', 'third', 'this',
             'those', 'though', 'three', 'through', 'throughout', 'thru', 'thus', 'to',
'together', 'too', 'top', 'toward',
             'towards', 'twelve', 'twenty', 'two', 'un', 'under', 'until', 'up', 'upon',
'us', 'very', 'via', 'was', 'we',
             'well', 'were', 'what', 'whatever', 'when', 'whence', 'whenever', 'where',
'whereafter', 'whereas', 'whereby',
             'wherein', 'whereupon', 'wherever', 'whether', 'which', 'while', 'whither',
'who', 'whoever', 'whole', 'whom',
             'whose', 'why', 'will', 'with', 'within', 'without', 'would', 'yet', 'you',
'your', 'yours', 'yourself',
```

```
            'yourselves',',','.']
    list2 = stopwords.words('English')
    list3 = []
    for i in list1:
        if i not in list2:
            list3.append(i)
    for i in list3:
        list1.append(i)
    return list1
```

Step 1 : Lower the characters of string

Step 2 : Remove sybmols(to remove punctuation) and apostrophe

Step 3 : Word tokenise

Step 4 : Removing single characters, stopwords ,lemmatizing and stemming

In [7]:

```
def refine_text(df1,list1):
    symbols = "!\"#$%&()*+-./:;<=>?@[\]^_`{|}~\n'"
    list4 = []
    list5 = []
    list6 = []
    list7 = []
    for i in range(100):
        temp = np.char.lower(str(df1[13][i]))
        for i in symbols:
            temp = np.char.replace(temp, i, ' ')
        list6.append((np.array2string(temp)).split("'")[1])
    df1['words_new'] = list6
    for i in range(100):
        list4.append(word_tokenize(df1['words_new'][i]))
    df1['tokens'] = list4
    lemmatizer = WordNetLemmatizer()
    ps = PorterStemmer()
    for i  in range(100):
        filtered_tokens = []
        for w in df1['tokens'][i]:
            if w not in list1 and len(w)>1:
                filtered_tokens.append(ps.stem(lemmatizer.lemmatize(w)))
        list5.append(filtered_tokens)
    df1['filtered_tokens'] = list5
    for i in range(100):
        temp = ''
        for j in range(len(df1['filtered_tokens'][i])):
            temp = temp + ' ' + df1['filtered_tokens'][i][j]
        list7.append(temp)
    df1['tf_idf_sentences'] = list7
    return df1
```

Part 1

In [8]:

```
def part1(df):
    df = refine_text(df, list1)
    df = df['filtered_tokens']
    prediction = predict_tag1(df)
    return prediction
```

In [9]:

```
def predict_tag1(df):
    if df[0]:
        for j in df[0]:
```

```
            if j in ['guess','prefer','answer']:
                    return 1
            else:
                    return 0
    else:
        return 1
```

Part 2

In [10]:

```
def part2(df):
    df = refine_dataframe(df)
    prediction = model.predict(df)
    return prediction
```

In [11]:

```
def refine_dataframe(df):
    df2 = df.drop([0,1,2,3,7,13,16,17,18,19,20,21,22,23,24,25,'words','words_new','toke
ns','filtered_tokens','tf_idf_sentences'],axis = 1)
    df2.replace('Yes','1',inplace = True)
    df2.replace('Yes ','1',inplace = True)
    df2.replace('No','-1',inplace = True)
    df2.replace('Maybe','0',inplace = True)
    df2.replace('Extrovert','1',inplace = True)
    df2.replace('Introvert','-1',inplace = True)
    df2.replace('Ambivert','0',inplace = True)
    df2.replace('Positively','1',inplace = True)
    df2.replace('Negatively','-1',inplace = True)
    df2.replace('Not comfortable','0',inplace = True)
    df2.replace('0-25','0.25',inplace = True)
    df2.replace('26-50','0.50',inplace = True)
    df2.replace('51-75','0.75',inplace = True)
    df2.replace('76-100','1.00',inplace = True)
    return df2
```

Model Development - Logistic Regression

In [12]:

```
def model_development(df2):
    x_train = df2.drop(['Flag'],axis = 1)
    y_train = df2['Flag']
    model = LogisticRegression(C = 1).fit(x_train,y_train)
    return model
```

Part 3

In [13]:

```
def part3(df3):
    a = df3['tf_idf_sentences'][0]
    b = vectorizer.fit_transform([a])
    c = lda_model.fit_transform(b)
    prediction = predict_tag2(c)
    return prediction
```

In [14]:

```
def tf_idf(df):
    vectorizer = TfidfVectorizer()
    list1 = []
    for i in range(100):
        list1.append(df['tf_idf_sentences'][i])
    X = vectorizer.fit_transform(list1)
```

```
    idf=vectorizer.idf_
    dd=dict(zip(vectorizer.get_feature_names(), idf))
    lda_model=LatentDirichletAllocation(n_components=2,learning_method='online',random_st
ate=42,max_iter=1)
    lda_top=lda_model.fit_transform(X)
    return vectorizer,lda_model
```

In [15]:

```
def predict_tag2(c):
    if c[1][0]>c[1][1]:
        return(0)
    else:
        return(1)
```

Part 4

In [16]:

```
def insight(score):
    if score < 5:
        return "None"
    elif score < 10:
        return "Mild"
    elif score < 15:
        return "Moderate"
    elif score < 20:
        return "Moderately Severe"
    else :
        return "Severe"
```

Testing

In [17]:

```
df,list1,model,vectorizer,lda_model = first_run()
```

In [18]:

```
dict1 = {'a' : 1, 'b' : 1, 'c' : 1, 'd' : 1, 'e' : 1, 'f' : 1, 'g' : 1, 'h' : 1, 'i' : 1
}
temp = form2(dict1)
```

In [19]:

```
temp
```

Out[19]:

```
'Mild'
```

In [ ]:

```

```