# STEP-1 Business Problem Understanding

## Predict the Personality types using Machine learning models

# STEP-2 Data Understanding

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.simplefilter("ignore")
```

In [2]:

```
df=pd.read_csv("Extrovert vs Introvert.csv")
df
```

Out[2]:

| | Time_spent_Alone | Stage_fear | Social_event_attendance | Going_outside | Drained_after_socializing | Friends_circle_size | Post_fr |
|---|---|---|---|---|---|---|---|
| **0** | 4.0 | No | 4.0 | 6.0 | No | 13.0 | |
| **1** | 9.0 | Yes | 0.0 | 0.0 | Yes | 0.0 | |
| **2** | 9.0 | Yes | 1.0 | 2.0 | Yes | 5.0 | |
| **3** | 0.0 | No | 6.0 | 7.0 | No | 14.0 | |
| **4** | 3.0 | No | 9.0 | 4.0 | No | 8.0 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **2895** | 3.0 | No | 7.0 | 6.0 | No | 6.0 | |
| **2896** | 3.0 | No | 8.0 | 3.0 | No | 14.0 | |
| **2897** | 4.0 | Yes | 1.0 | 1.0 | Yes | 4.0 | |
| **2898** | 11.0 | Yes | 1.0 | NaN | Yes | 2.0 | |
| **2899** | 3.0 | No | 6.0 | 6.0 | No | 6.0 | |

2900 rows × 8 columns

In [3]:

```
df.shape
```

Out[3]:

```
(2900, 8)
```

In [4]:

```
df["Time_spent_Alone"]
```

Out[4]:

```
0        4.0
1        9.0
2        9.0
3        0.0
4        3.0
        ... 
2895     3.0
2896     3.0
2897     4.0
2898    11.0
2899     3.0
Name: Time_spent_Alone, Length: 2900, dtype: float64
```

In [5]:

```
df["Time_spent_Alone"].unique()
```

Out[5]:

```
array([ 4.,  9.,  0.,  3.,  1.,  2., 10.,  6.,  5.,  8., nan,  7., 11.])
```

In [6]:

```
df["Time_spent_Alone"].value_counts()
```

Out[6]:

```
Time_spent_Alone
0.0     369
2.0     357
3.0     353
1.0     326
9.0     206
10.0    196
4.0     190
7.0     190
5.0     180
8.0     180
6.0     150
11.0    140
Name: count, dtype: int64
```

In [7]:

```
df["Stage_fear"]
```

Out[7]:

```
0        No
1       Yes
2       Yes
3        No
4        No
       ... 
2895     No
2896     No
2897    Yes
2898    Yes
2899     No
Name: Stage_fear, Length: 2900, dtype: object
```

In [8]:

```
df["Stage_fear"].unique()
```

Out[8]:

```
array(['No', 'Yes', nan], dtype=object)
```

In [9]:

```
df["Stage_fear"].value_counts()
```

Out[9]:

```
Stage_fear
No     1417
Yes    1410
Name: count, dtype: int64
```

In [10]:

```
df["Social_event_attendance"]
```

Out[10]:

```
0       4.0
1       0.0
2       1.0
3       6.0
4       9.0
       ... 
2895    7.0
2896    8.0
2897    1.0
2898    1.0
2899    6.0
Name: Social_event_attendance, Length: 2900, dtype: float64
```

In [11]:

```
df["Social_event_attendance"].unique()
```

Out[11]:

```
array([ 4.,  0.,  1.,  6.,  9.,  7.,  8.,  3.,  5.,  2., 10., nan])
```

In [12]:

```
df["Social_event_attendance"].value_counts()
```

Out[12]:

```
Social_event_attendance
2.0     408
0.0     378
1.0     322
3.0     317
4.0     255
6.0     239
7.0     239
9.0     236
5.0     224
8.0     206
10.0     14
Name: count, dtype: int64
```

In [13]:

```
df["Going_outside"]
```

```
Out[13]: 0       6.0
         1       0.0
         2       2.0
         3       7.0
         4       4.0
                ...
         2895    6.0
         2896    3.0
         2897    1.0
         2898    NaN
         2899    6.0
         Name: Going_outside, Length: 2900, dtype: float64
```

```
In [14]: df["Going_outside"].unique()
```

```
Out[14]: array([ 6.,  0.,  2.,  7.,  4.,  5., nan,  3.,  1.])
```

```
In [15]: df["Going_outside"].value_counts()
```

```
Out[15]: Going_outside
         0.0    498
         2.0    456
         1.0    429
         5.0    374
         4.0    359
         6.0    335
         3.0    209
         7.0    174
         Name: count, dtype: int64
```

```
In [16]: df["Drained_after_socializing"]
```

```
Out[16]: 0        No
         1       Yes
         2       Yes
         3        No
         4        No
                ...
         2895     No
         2896     No
         2897    Yes
         2898    Yes
         2899     No
         Name: Drained_after_socializing, Length: 2900, dtype: object
```

```
In [17]: df["Drained_after_socializing"].unique()
```

```
Out[17]: array(['No', 'Yes', nan], dtype=object)
```

```
In [18]: df["Drained_after_socializing"].value_counts()
```

```
Out[18]: Drained_after_socializing
         No     1441
         Yes    1407
         Name: count, dtype: int64
```

```
In [19]: df["Friends_circle_size"]
```

```
Out[19]: 0       13.0
         1        0.0
         2        5.0
         3       14.0
         4        8.0
                 ...
         2895     6.0
         2896    14.0
         2897     4.0
         2898     2.0
         2899     6.0
         Name: Friends_circle_size, Length: 2900, dtype: float64
```

```
In [20]: df["Friends_circle_size"].unique()
```

```
Out[20]: array([13.,  0.,  5., 14.,  8.,  6.,  7., 15.,  4., 10.,  1., 12.,  2.,
                11.,  9.,  3., nan])
```

```
In [21]: df["Friends_circle_size"].value_counts()
```

```
Out[21]: Friends_circle_size
         5.0     301
         3.0     283
         1.0     281
         2.0     274
         4.0     254
         8.0     165
         12.0    148
         10.0    146
         14.0    144
         6.0     137
         9.0     135
         11.0    134
         7.0     133
         13.0    123
         0.0     106
         15.0     59
         Name: count, dtype: int64
```

```
In [22]: df["Post_frequency"]
```

```
Out[22]: 0       5.0
         1       3.0
         2       2.0
         3       8.0
         4       5.0
                ...
         2895    6.0
         2896    9.0
         2897    0.0
         2898    0.0
         2899    9.0
         Name: Post_frequency, Length: 2900, dtype: float64
```

```
In [23]: df["Post_frequency"].unique()
```

```
Out[23]: array([ 5.,  3.,  2.,  8.,  6.,  7.,  0., 10.,  4.,  1.,  9., nan])
```

```
In [24]: df["Post_frequency"].value_counts()
```

```
Out[24]: Post_frequency
         2.0     481
         1.0     455
         0.0     451
         7.0     236
         5.0     212
         6.0     210
         3.0     208
         4.0     195
         8.0     193
         9.0     171
         10.0     23
         Name: count, dtype: int64
```

```
In [25]: df["Personality"]
```

```
Out[25]: 0       Extrovert
         1       Introvert
         2       Introvert
         3       Extrovert
         4       Extrovert
                   ...
         2895    Extrovert
         2896    Extrovert
         2897    Introvert
         2898    Introvert
         2899    Extrovert
         Name: Personality, Length: 2900, dtype: object
```

```
In [26]: df["Personality"].unique()
```

```
Out[26]: array(['Extrovert', 'Introvert'], dtype=object)
```

```
In [27]: df["Personality"].value_counts()
```

```
Out[27]: Personality
         Extrovert    1491
         Introvert    1409
         Name: count, dtype: int64
```

```
In [28]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2900 entries, 0 to 2899
Data columns (total 8 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Time_spent_Alone           2837 non-null   float64
 1   Stage_fear                 2827 non-null   object 
 2   Social_event_attendance    2838 non-null   float64
 3   Going_outside              2834 non-null   float64
 4   Drained_after_socializing  2848 non-null   object 
 5   Friends_circle_size        2823 non-null   float64
 6   Post_frequency             2835 non-null   float64
 7   Personality                2900 non-null   object 
dtypes: float64(5), object(3)
memory usage: 181.4+ KB
```

In [29]:

```
continuous = ["Time_spent_Alone","Social_event_attendance","Going_outside","Friends_circle_size","Post_frequency

discrete_categorical = ["Stage_fear","Drained_after_socializing","Personality"]
```

# Exploratory Data Analysis

## For Continuous Variables

In [30]:

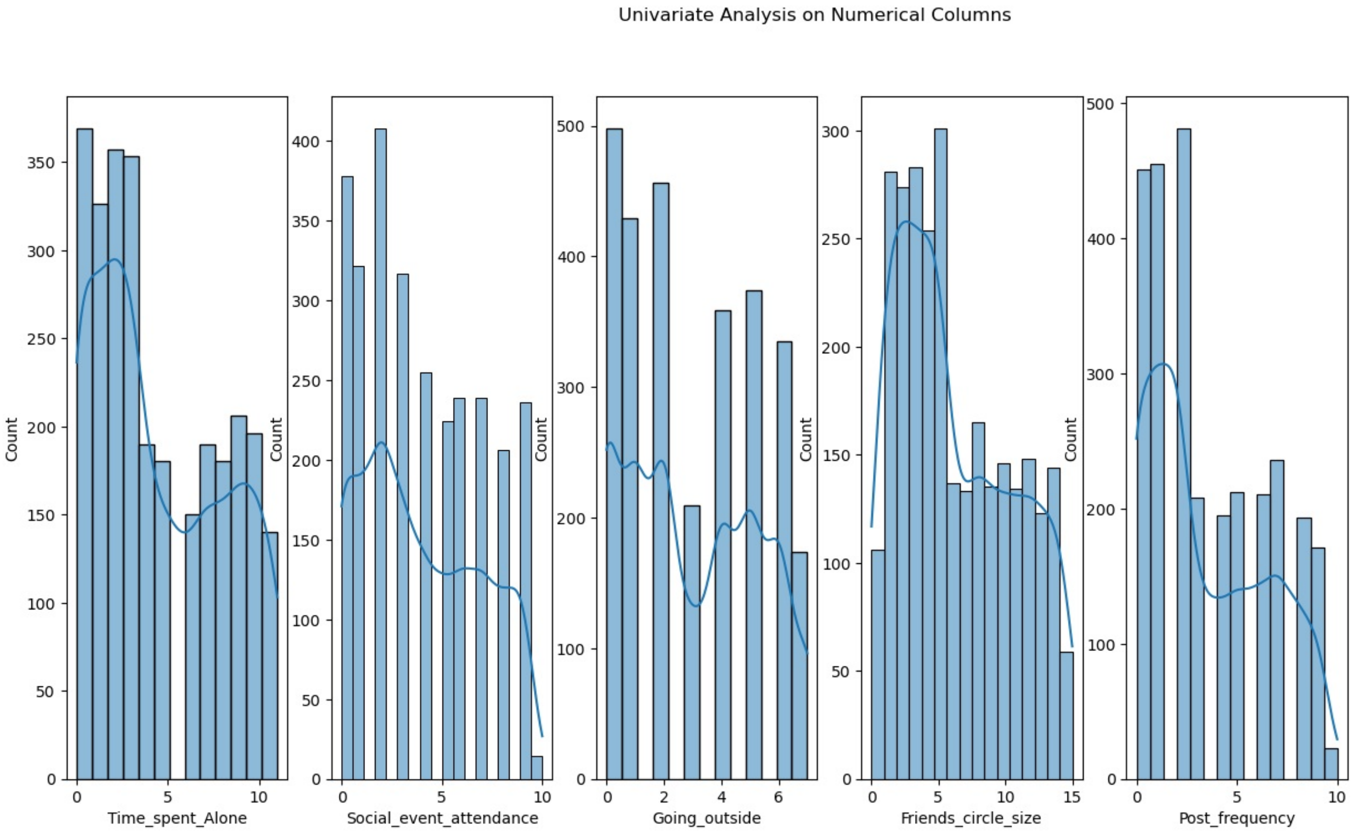
```
df[continuous].describe()
```

Out[30]:

| | Time_spent_Alone | Social_event_attendance | Going_outside | Friends_circle_size | Post_frequency |
|---|---|---|---|---|---|
| **count** | 2837.000000 | 2838.000000 | 2834.000000 | 2823.000000 | 2835.000000 |
| **mean** | 4.505816 | 3.963354 | 3.000000 | 6.268863 | 3.564727 |
| **std** | 3.479192 | 2.903827 | 2.247327 | 4.289693 | 2.926582 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 2.000000 | 2.000000 | 1.000000 | 3.000000 | 1.000000 |
| **50%** | 4.000000 | 3.000000 | 3.000000 | 5.000000 | 3.000000 |
| **75%** | 8.000000 | 6.000000 | 5.000000 | 10.000000 | 6.000000 |
| **max** | 11.000000 | 10.000000 | 7.000000 | 15.000000 | 10.000000 |

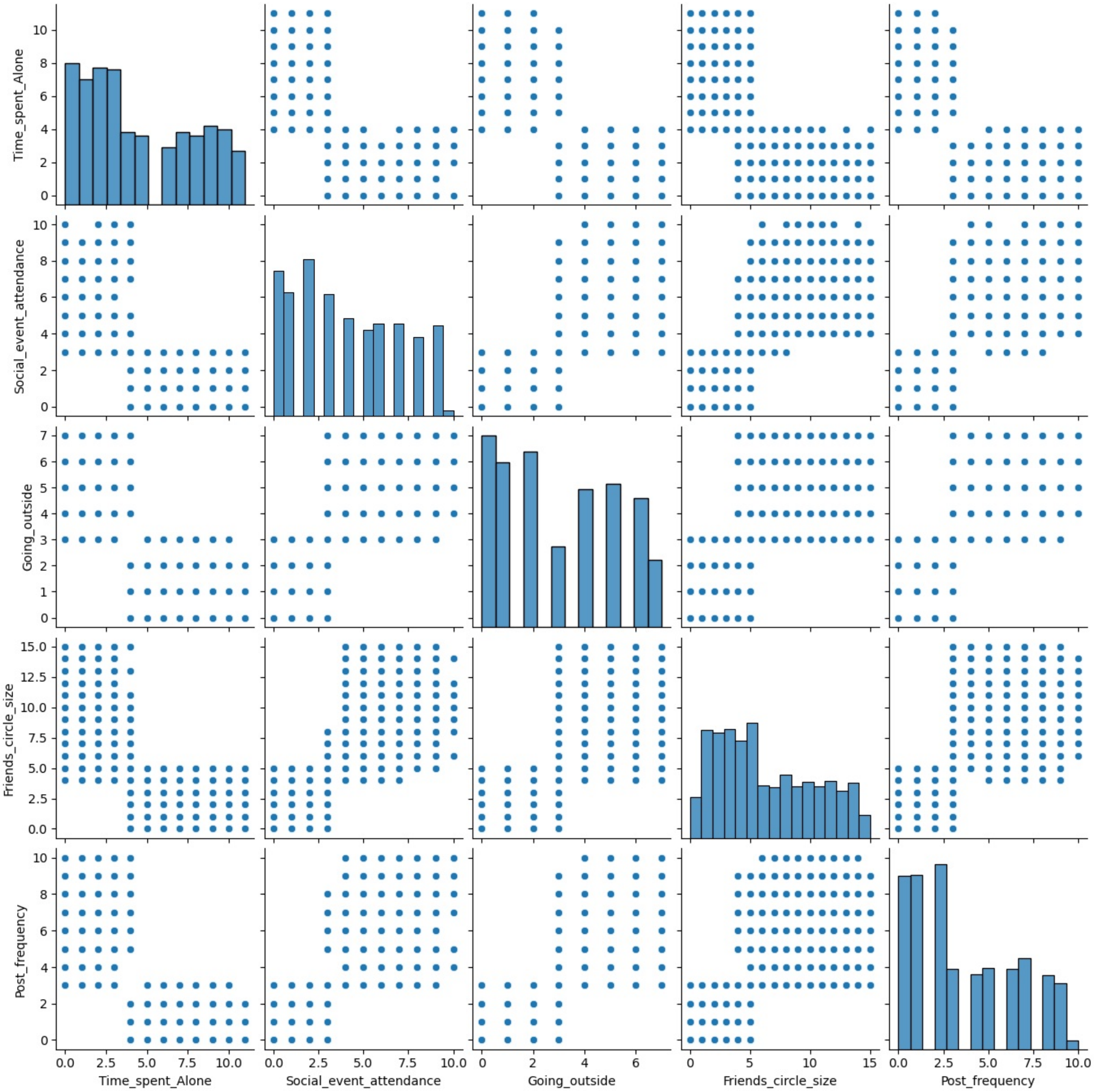
In [31]:

```
plt.rcParams["figure.figsize"] = (18,8)
plt.subplot(1,6, 1)
sns.histplot(df["Time_spent_Alone"],kde=True)
plt.subplot(1,6, 2)
sns.histplot(df["Social_event_attendance"],kde=True)
plt.subplot(1,6, 3)
sns.histplot(df["Going_outside"],kde=True)
plt.subplot(1,6, 4)
sns.histplot(df["Friends_circle_size"],kde=True)
plt.subplot(1,6, 5)
sns.histplot(df["Post_frequency"],kde=True)
plt.suptitle("Univariate Analysis on Numerical Columns")
plt.show()
```



In [32]:

```
sns.pairplot(df[continuous])
plt.show()
```



In [33]:

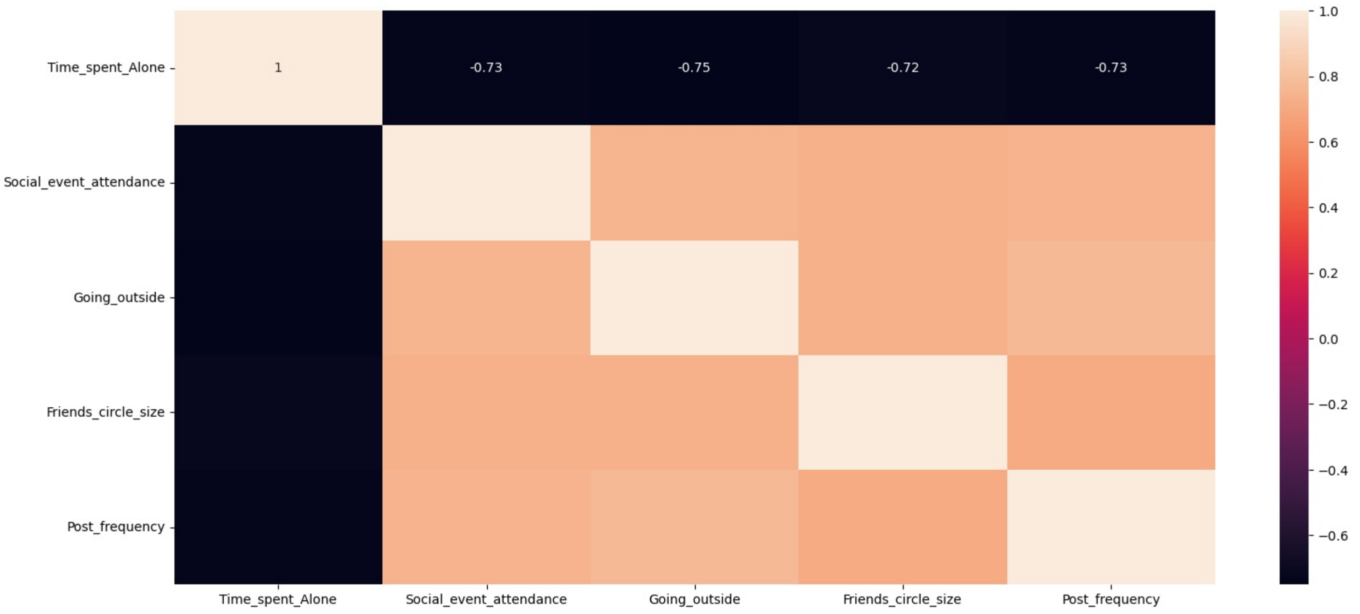
```
df[continuous].corr()
```

Out[33]:

| | Time_spent_Alone | Social_event_attendance | Going_outside | Friends_circle_size | Post_frequency |
|---|---|---|---|---|---|
| **Time_spent_Alone** | 1.000000 | -0.733011 | -0.750760 | -0.717185 | -0.732649 |
| **Social_event_attendance** | -0.733011 | 1.000000 | 0.747756 | 0.734795 | 0.744615 |
| **Going_outside** | -0.750760 | 0.747756 | 1.000000 | 0.736390 | 0.770819 |
| **Friends_circle_size** | -0.717185 | 0.734795 | 0.736390 | 1.000000 | 0.707888 |
| **Post_frequency** | -0.732649 | 0.744615 | 0.770819 | 0.707888 | 1.000000 |

In [34]:

```
sns.heatmap(df[continuous].corr(),annot=True)
plt.show()
```



### For Discrete variables

In [35]:

```
df[discrete_categorical].describe()
```

Out[35]:

| | Stage_fear | Drained_after_socializing | Personality |
|---|---|---|---|
| **count** | 2827 | 2848 | 2900 |
| **unique** | 2 | 2 | 2 |
| **top** | No | No | Extrovert |
| **freq** | 1417 | 1441 | 1491 |

# Steps to be followed for data cleaning

### Check for Wrong Data

### Check for Wrong Data type

In [36]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2900 entries, 0 to 2899
Data columns (total 8 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Time_spent_Alone           2837 non-null   float64
 1   Stage_fear                 2827 non-null   object 
 2   Social_event_attendance    2838 non-null   float64
 3   Going_outside              2834 non-null   float64
 4   Drained_after_socializing  2848 non-null   object 
 5   Friends_circle_size        2823 non-null   float64
 6   Post_frequency             2835 non-null   float64
 7   Personality                2900 non-null   object 
dtypes: float64(5), object(3)
memory usage: 181.4+ KB
```

### Check for duplicates

In [37]:

```
df.duplicated().sum()
```

Out[37]:

```
388
```

In [38]:

```
df[df.duplicated()]
```

Out[38]:

| | Time_spent_Alone | Stage_fear | Social_event_attendance | Going_outside | Drained_after_socializing | Friends_circle_size | Post_fr |
|---|---|---|---|---|---|---|---|
| **47** | 10.0 | Yes | 1.0 | 2.0 | Yes | 2.0 | |
| **217** | 5.0 | Yes | 2.0 | 0.0 | Yes | 2.0 | |
| **246** | 9.0 | Yes | 0.0 | 1.0 | Yes | 2.0 | |
| **248** | 9.0 | Yes | 0.0 | 2.0 | Yes | 3.0 | |
| **254** | 7.0 | Yes | 0.0 | 0.0 | Yes | 3.0 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **2884** | 11.0 | Yes | 0.0 | 2.0 | Yes | 3.0 | |
| **2890** | 8.0 | Yes | 2.0 | 0.0 | Yes | 1.0 | |
| **2891** | 6.0 | Yes | 3.0 | 1.0 | Yes | 5.0 | |
| **2892** | 9.0 | Yes | 2.0 | 0.0 | Yes | 1.0 | |
| **2895** | 3.0 | No | 7.0 | 6.0 | No | 6.0 | |

388 rows × 8 columns

In [39]:

```
df[~df.duplicated()]
```

Out[39]:

| | Time_spent_Alone | Stage_fear | Social_event_attendance | Going_outside | Drained_after_socializing | Friends_circle_size | Post_fr |
|---|---|---|---|---|---|---|---|
| **0** | 4.0 | No | 4.0 | 6.0 | No | 13.0 | |
| **1** | 9.0 | Yes | 0.0 | 0.0 | Yes | 0.0 | |
| **2** | 9.0 | Yes | 1.0 | 2.0 | Yes | 5.0 | |
| **3** | 0.0 | No | 6.0 | 7.0 | No | 14.0 | |
| **4** | 3.0 | No | 9.0 | 4.0 | No | 8.0 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **2894** | 0.0 | No | 9.0 | 3.0 | No | 12.0 | |
| **2896** | 3.0 | No | 8.0 | 3.0 | No | 14.0 | |
| **2897** | 4.0 | Yes | 1.0 | 1.0 | Yes | 4.0 | |
| **2898** | 11.0 | Yes | 1.0 | NaN | Yes | 2.0 | |
| **2899** | 3.0 | No | 6.0 | 6.0 | No | 6.0 | |

2512 rows × 8 columns

## Check for Missing values

In [40]:

```
df.isnull().sum()
```

Out[40]:

```
Time_spent_Alone             63
Stage_fear                   73
Social_event_attendance      62
Going_outside                66
Drained_after_socializing    52
Friends_circle_size          77
Post_frequency               65
Personality                   0
dtype: int64
```

In [41]:

```
df.isnull().sum()/len(df)*100
```

Out[41]:

```
Time_spent_Alone             2.172414
Stage_fear                   2.517241
Social_event_attendance      2.137931
Going_outside                2.275862
Drained_after_socializing    1.793103
Friends_circle_size          2.655172
Post_frequency               2.241379
Personality                  0.000000
dtype: float64
```

## Check for skewness

In [42]:

```
df[["Time_spent_Alone","Social_event_attendance","Going_outside","Friends_circle_size","Post_frequency"]].skew(
```

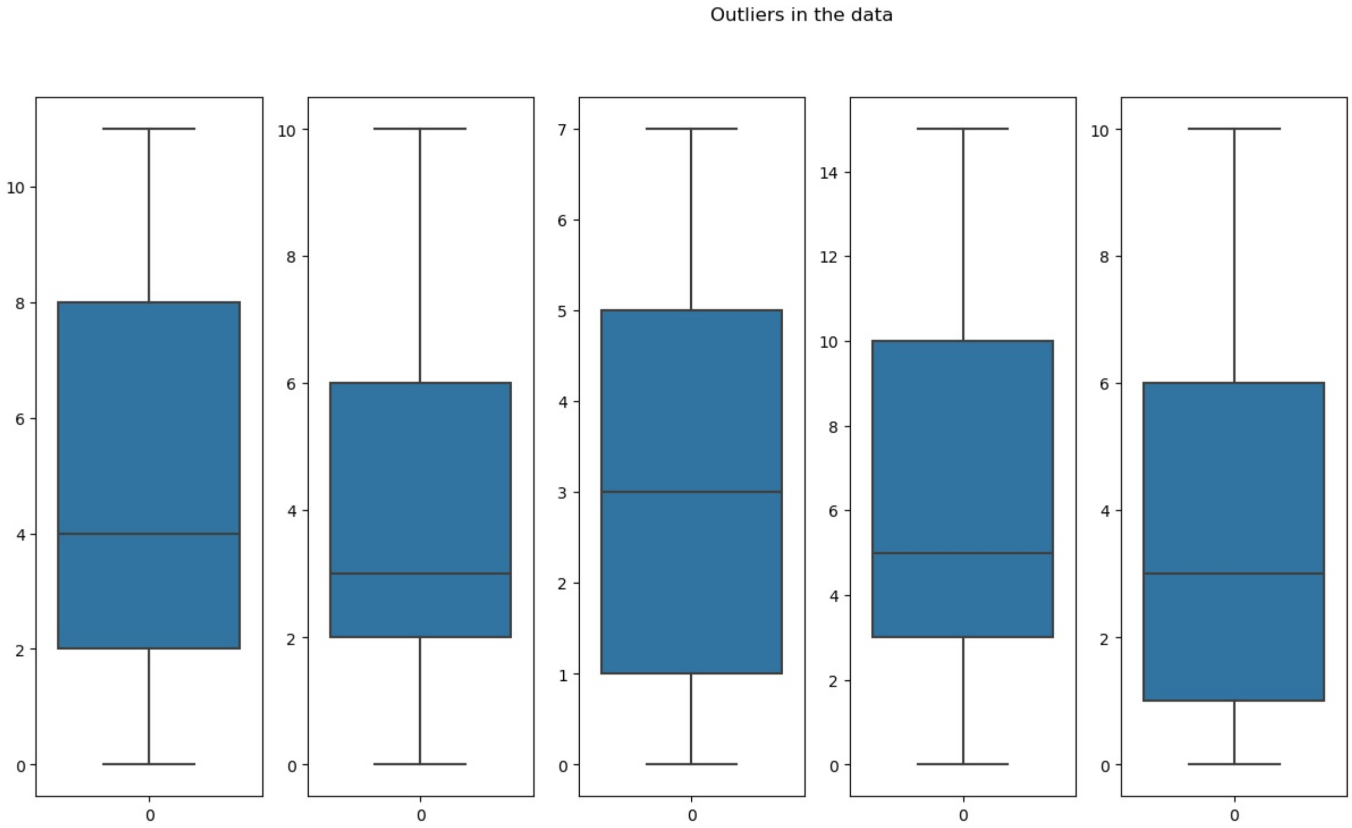
Out[42]:

```
Time_spent_Alone           0.385821
Social_event_attendance    0.294742
Going_outside              0.192891
Friends_circle_size        0.425051
Post_frequency             0.474510
dtype: float64
```

### Check for outliers

In [43]:

```
# Lets visualize the outliers using Boxplot
plt.subplot(1,6, 1)
sns.boxplot(df["Time_spent_Alone"])
plt.subplot(1,6, 2)
sns.boxplot(df["Social_event_attendance"])
plt.subplot(1,6, 3)
sns.boxplot(df["Going_outside"])
plt.subplot(1,6, 4)
sns.boxplot(df["Friends_circle_size"])
plt.subplot(1,6, 5)
sns.boxplot(df["Post_frequency"])
plt.suptitle("Outliers in the data")
plt.show()
```



# STEP-3 Data Preprocessing

## I.Data Cleaning

### No Treatment for wrong data

### No Treatment for wrong datatype

### Treating Duplicates

In [44]:

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

Out[44]:

| | Time_spent_Alone | Stage_fear | Social_event_attendance | Going_outside | Drained_after_socializing | Friends_circle_size | Post_fr |
|---|---|---|---|---|---|---|---|
| **0** | 4.0 | No | 4.0 | 6.0 | No | 13.0 | |
| **1** | 9.0 | Yes | 0.0 | 0.0 | Yes | 0.0 | |
| **2** | 9.0 | Yes | 1.0 | 2.0 | Yes | 5.0 | |
| **3** | 0.0 | No | 6.0 | 7.0 | No | 14.0 | |
| **4** | 3.0 | No | 9.0 | 4.0 | No | 8.0 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **2507** | 0.0 | No | 9.0 | 3.0 | No | 12.0 | |
| **2508** | 3.0 | No | 8.0 | 3.0 | No | 14.0 | |
| **2509** | 4.0 | Yes | 1.0 | 1.0 | Yes | 4.0 | |
| **2510** | 11.0 | Yes | 1.0 | NaN | Yes | 2.0 | |
| **2511** | 3.0 | No | 6.0 | 6.0 | No | 6.0 | |

2512 rows × 8 columns

### Treating Missing values

In [45]:

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

Out[45]:

| | Time_spent_Alone | Stage_fear | Social_event_attendance | Going_outside | Drained_after_socializing | Friends_circle_size | Post_fr |
|---|---|---|---|---|---|---|---|
| **0** | 4.0 | No | 4.0 | 6.0 | No | 13.0 | |
| **1** | 9.0 | Yes | 0.0 | 0.0 | Yes | 0.0 | |
| **2** | 9.0 | Yes | 1.0 | 2.0 | Yes | 5.0 | |
| **3** | 0.0 | No | 6.0 | 7.0 | No | 14.0 | |
| **4** | 3.0 | No | 9.0 | 4.0 | No | 8.0 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **2504** | 5.0 | Yes | 0.0 | 1.0 | Yes | 1.0 | |
| **2505** | 6.0 | Yes | 0.0 | 0.0 | Yes | 3.0 | |
| **2508** | 3.0 | No | 8.0 | 3.0 | No | 14.0 | |
| **2509** | 4.0 | Yes | 1.0 | 1.0 | Yes | 4.0 | |
| **2511** | 3.0 | No | 6.0 | 6.0 | No | 6.0 | |

2098 rows × 8 columns

In [46]:

```
df.isnull().sum()
```

Out[46]:

```
Time_spent_Alone             0
Stage_fear                   0
Social_event_attendance      0
Going_outside                0
Drained_after_socializing    0
Friends_circle_size          0
Post_frequency               0
Personality                  0
dtype: int64
```

### No Treatment for Outliers

## II.Data Wrangling

### Feature Encoding

In [47]:

```
df["Stage_fear"] = df["Stage_fear"].replace({"No":0,"Yes":1})
df["Drained_after_socializing"] = df["Drained_after_socializing"].replace({"No":0,"Yes":1})
df["Personality"] = df["Personality"].replace({"Extrovert":0,"Introvert":1})
```

### No Feature Transformations

## No Feature Scaling

In [48]:
```
df
```

Out[48]:

| | Time_spent_Alone | Stage_fear | Social_event_attendance | Going_outside | Drained_after_socializing | Friends_circle_size | Post_fr |
|---|---|---|---|---|---|---|---|
| **0** | 4.0 | 0 | 4.0 | 6.0 | 0 | 13.0 | |
| **1** | 9.0 | 1 | 0.0 | 0.0 | 1 | 0.0 | |
| **2** | 9.0 | 1 | 1.0 | 2.0 | 1 | 5.0 | |
| **3** | 0.0 | 0 | 6.0 | 7.0 | 0 | 14.0 | |
| **4** | 3.0 | 0 | 9.0 | 4.0 | 0 | 8.0 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **2504** | 5.0 | 1 | 0.0 | 1.0 | 1 | 1.0 | |
| **2505** | 6.0 | 1 | 0.0 | 0.0 | 1 | 3.0 | |
| **2508** | 3.0 | 0 | 8.0 | 3.0 | 0 | 14.0 | |
| **2509** | 4.0 | 1 | 1.0 | 1.0 | 1 | 4.0 | |
| **2511** | 3.0 | 0 | 6.0 | 6.0 | 0 | 6.0 | |

2098 rows × 8 columns

## X&y

In [49]:
```
X = df.drop("Personality",axis=1)
y = df["Personality"]
```

## Identify the best random number

In [50]:
```
Train = []
Test = []
CV = []
for i in range(0,101):
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=i)

    from sklearn.linear_model import LogisticRegression
    log_default = LogisticRegression()
    log_default.fit(X_train,y_train)

    ypred_train = log_default.predict(X_train)
    ypred_test = log_default.predict(X_test)
    from sklearn.metrics import accuracy_score
    Train.append(accuracy_score(y_train, ypred_train))
    Test.append(accuracy_score(y_test, ypred_test))

    from sklearn.model_selection import cross_val_score
    CV.append(cross_val_score(log_default, X_train, y_train, cv=5, scoring="accuracy").mean())
#Storing all results
em = pd.DataFrame({"Train":Train, "Test":Test, "CV":CV})
#Find the best random state
gm = em[(abs(em["Train"]-em["Test"])<=0.05) & (abs(em["Test"]-em["CV"])<=0.05)]
#pick the highest CV
rs = gm[gm["CV"]==gm["CV"].max()].index.to_list()[0]
print("best random_state number:",rs)
```

```
best random_state number: 57
```

## III.train_test_split

In [51]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=57)
```

# Step-4 ML Modelling

In [52]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier

from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
```

## 1.Logistic Regression Algorithm

In [53]:

```
#Modelling
log_model = LogisticRegression()
log_model.fit(X_train,y_train)

#Evaluation
ypred_train = log_model.predict(X_train)
ypred_test  = log_model.predict(X_test)

print("Train Accuracy :",accuracy_score(y_train,ypred_train))
print("cross validation score:",cross_val_score(log_model,X_train,y_train,cv=5,scoring="accuracy").mean())
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
```

```
Train Accuracy : 0.9249106078665077
cross validation score: 0.9249129353233831
Test Accuracy : 0.8761904761904762
```

## 2.KNN Classifier Algorithm

In [54]:

```
#Hyper Parameter Tuning
estimator = KNeighborsClassifier()
param_grid = {"n_neighbors": list(range(1,50))}
knn_grid = GridSearchCV(estimator, param_grid,scoring="accuracy",cv=5)
knn_grid.fit(X_train,y_train)
knn_model = knn_grid.best_estimator_
knn_model
```

Out[54]:

```
▾          KNeighborsClassifier
KNeighborsClassifier(n_neighbors=13)
```

In [55]:

```
#Modelling
knn_model = KNeighborsClassifier(n_neighbors=13)
knn_model.fit(X_train,y_train)

#Evaluation
ypred_train = knn_model.predict(X_train)
ypred_test  = knn_model.predict(X_test)

print("Train Accuracy :",accuracy_score(y_train,ypred_train))
print("cross validation score :",cross_val_score(knn_model,X_train,y_train,cv=5,scoring="accuracy").mean())
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
```

```
Train Accuracy : 0.9249106078665077
cross validation score : 0.9249129353233831
Test Accuracy : 0.8761904761904762
```

## 3.Support Vector Machine Algorithm

In [56]:

```
#Hyper Parameter Tuning
estimator = SVC()
param_grid = {"C":[0.01,0.1,1],"kernel":["linear","rbf","sigmoid","poly"]}
svm_grid = GridSearchCV(estimator,param_grid,scoring="accuracy",cv=5)
svm_grid.fit(X_train,y_train)
svm_model = svm_grid.best_estimator_
svm_model
```

Out[56]:

```
▾          SVC
SVC(C=0.01, kernel='linear')
```

In [57]:

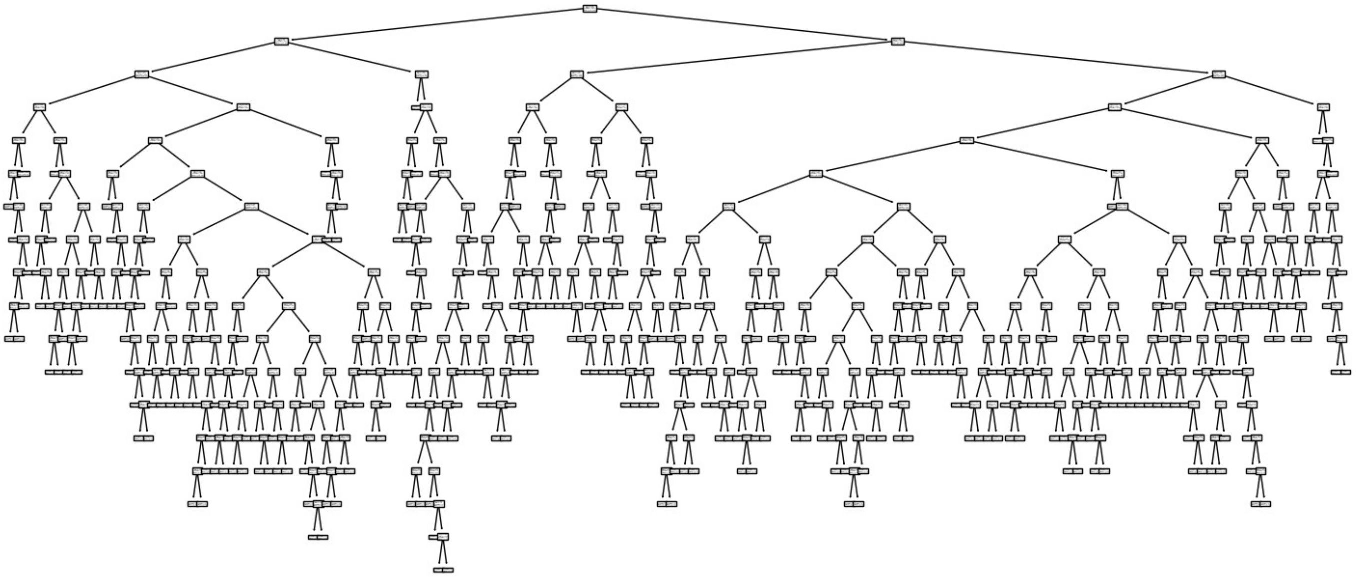
```
#Modelling
svm_model = SVC(C=0.01, kernel="linear")
svm_model.fit(X_train,y_train)

#Evaluation
ypred_train = svm_model.predict(X_train)
ypred_test  = svm_model.predict(X_test)
```

```
print("Train Accuracy :",accuracy_score(y_train,ypred_train))
print("cross validation score :",cross_val_score(svm_model,X_train,y_train,cv=5,scoring="accuracy").mean())
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
```

```
Train Accuracy : 0.9249106078665077
cross validation score : 0.9249129353233831
Test Accuracy : 0.8761904761904762
```

## 4.Decision Tree Classifier Algorithm

In [58]:
```
model = DecisionTreeClassifier(random_state=True)
model.fit(X_train,y_train)
from sklearn.tree import plot_tree
plot_tree(model)
plt.show()
```



In [59]:
```
#Hyper Parameter Tuning
estimator = DecisionTreeClassifier(random_state=True)
param_grid = {"criterion":["gini", "entropy"],
              "max_depth":list(range(1,16))}
dt_grid = GridSearchCV(estimator,param_grid,scoring="accuracy",cv=5)
dt_grid.fit(X_train,y_train)
dt = dt_grid.best_estimator_
dt
```

Out[59]:
```
▾                DecisionTreeClassifier
DecisionTreeClassifier(max_depth=1, random_state=True)
```

In [60]:
```
dt.feature_importances_
```

Out[60]:
```
array([0., 0., 0., 0., 1., 0., 0.])
```

In [61]:
```
#Important features
feats_dt = pd.DataFrame(data=dt.feature_importances_,
                        index=X.columns,
                        columns=["Importance"])
important_features_dt = feats_dt[feats_dt["Importance"]>0].index.tolist()
important_features_dt
```

Out[61]:
```
['Drained_after_socializing']
```

## Creating Decision Tree model with important parameters and important features

In [62]:
```
#Selecting train & test data
X_train_dt = X_train[important_features_dt]
X_test_dt = X_test[important_features_dt]

#Modelling
dt.fit(X_train_dt,y_train)

#Evaluation
ypred_train = dt.predict(X_train_dt)
ypred_test  = dt.predict(X_test_dt)

print("Train Accuracy :",accuracy_score(y_train,ypred_train))
print("cross validation score :",cross_val_score(dt,X_train_dt,y_train,cv=5,scoring="accuracy").mean())
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
```

```
Train Accuracy : 0.9249106078665077
cross validation score : 0.9249129353233831
Test Accuracy : 0.8761904761904762
```

## 5.Random Forest Classifier Algorithm

In [63]:

```
#Hyper Parameter Tuning
estimator = RandomForestClassifier(random_state=True)
param_grid = {"n_estimators":list(range(1,51))}
rf_grid = GridSearchCV(estimator,param_grid,scoring="accuracy",cv=5)
rf_grid.fit(X_train,y_train)
rf = rf_grid.best_estimator_
rf
```

Out[63]:

```
▾                    RandomForestClassifier
RandomForestClassifier(n_estimators=43, random_state=True)
```

In [64]:

```
#Important features
feats_rf = pd.DataFrame(data=rf.feature_importances_,
                        index=X.columns,
                        columns=["Importance"])
important_features_rf = feats_rf[feats_rf["Importance"]>0].index.tolist()
important_features_rf
```

Out[64]:

```
['Time_spent_Alone',
 'Stage_fear',
 'Social_event_attendance',
 'Going_outside',
 'Drained_after_socializing',
 'Friends_circle_size',
 'Post_frequency']
```

In [65]:

```
#Selecting train & test data
X_train_rf = X_train[important_features_rf]
X_test_rf = X_test[important_features_rf]

#Modelling
rf.fit(X_train_rf,y_train)

#Evaluation
ypred_train = rf.predict(X_train_rf)
ypred_test  = rf.predict(X_test_rf)

print("Train Accuracy :",accuracy_score(y_train,ypred_train))
print("cross validation score :",cross_val_score(rf,X_train_rf,y_train,cv=5,scoring="accuracy").mean())
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
```

```
Train Accuracy : 0.9791418355184743
cross validation score : 0.9010820895522389
Test Accuracy : 0.8571428571428571
```

## 6.Ada Boost Classifier Algorithm

In [66]:

```
#Hyper Parameter Tuning
estimator = AdaBoostClassifier(random_state=True)
param_grid = {"n_estimators":list(range(1,51))}
ab_grid = GridSearchCV(estimator,param_grid,scoring="accuracy",cv=5)
ab_grid.fit(X_train,y_train)
ab = ab_grid.best_estimator_
ab
```

Out[66]:

```
▾                 AdaBoostClassifier
AdaBoostClassifier(n_estimators=1, random_state=True)
```

In [67]:

```
#Important features
feats_ab = pd.DataFrame(data=ab.feature_importances_,
                        index=X.columns,
                        columns=["Importance"])
important_features_ab = feats_ab[feats_ab["Importance"]>0].index.tolist()
important_features_ab
```

Out[67]:

```
['Drained_after_socializing']
```

In [68]:

```
#Selecting train & test data
```

```
X_train_ab = X_train[important_features_ab]
X_test_ab = X_test[important_features_ab]

#Modelling
ab.fit(X_train_ab,y_train)

#Evaluation
ypred_train = ab.predict(X_train_ab)
ypred_test  = ab.predict(X_test_ab)

print("Train Accuracy :",accuracy_score(y_train,ypred_train))
print("cross validation score :",cross_val_score(ab,X_train_ab,y_train,cv=5,scoring="accuracy").mean())
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
```

```
Train Accuracy : 0.9249106078665077
cross validation score : 0.9249129353233831
Test Accuracy : 0.8761904761904762
```

## 7.Gradient Boosting Classifier Algorithm

In [69]:

```
#Hyper Parameter Tuning
estimator = GradientBoostingClassifier(random_state=True)
param_grid = {"n_estimators":list(range(1,10)),
              "learning_rate":[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]}

gb_grid = GridSearchCV(estimator,param_grid,scoring="accuracy",cv=5)
gb_grid.fit(X_train,y_train)

gb = gb_grid.best_estimator_
gb
```

Out[69]:

```
▾                         GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=0.8, n_estimators=8, random_state=True)
```

In [70]:

```
#Important features
feats_gb = pd.DataFrame(data=gb.feature_importances_,
                        index=X.columns,
                        columns=["Importance"])
important_features_gb = feats_gb[feats_gb["Importance"]>0].index.tolist()
important_features_gb
```

Out[70]:

```
['Time_spent_Alone',
 'Stage_fear',
 'Social_event_attendance',
 'Going_outside',
 'Drained_after_socializing',
 'Friends_circle_size',
 'Post_frequency']
```

In [71]:

```
#Selecting train & test data
X_train_gb = X_train[important_features_gb]
X_test_gb = X_test[important_features_gb]

#Modelling
gb.fit(X_train_gb,y_train)

#Evaluation
ypred_train = gb.predict(X_train_gb)
ypred_test  = gb.predict(X_test_gb)

print("Train Accuracy :",accuracy_score(y_train,ypred_train))
print("cross validation score :",cross_val_score(gb,X_train_gb,y_train,cv=5,scoring="accuracy").mean())
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
```

```
Train Accuracy : 0.9249106078665077
cross validation score : 0.9255099502487563
Test Accuracy : 0.8761904761904762
```

## 8.XG Boost Classifier Algorithm

In [72]:

```
#Hyper Parameter Tuning
estimator = XGBClassifier()
param_grid = {"n_estimators":[10,20,40,100],
              "max_depth":[3,4,5],
              "gamma":[0,0.15,0.3,0.5,1]}

xgb_grid = GridSearchCV(estimator,param_grid,scoring="accuracy",cv=5)
xgb_grid.fit(X_train,y_train)

xgb = xgb_grid.best_estimator_
```

```
xgb
```

Out[72]:

```
▾                         XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=Non
e,
              enable_categorical=False, eval_metric=None, feature_types=Non
e,
              gamma=0, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=Non
e,
```

In [73]:

```
#Important features
feats_xgb = pd.DataFrame(data=xgb.feature_importances_,
                         index=X.columns,
                         columns=["Importance"])
important_features_xgb = feats_gb[feats_xgb["Importance"]>0].index.tolist()
important_features_xgb
```

Out[73]:

```
['Time_spent_Alone',
 'Stage_fear',
 'Social_event_attendance',
 'Going_outside',
 'Friends_circle_size',
 'Post_frequency']
```

In [74]:

```
#Selecting train & test data
X_train_xgb = X_train[important_features_xgb]
X_test_xgb = X_test[important_features_xgb]

#Modelling
xgb.fit(X_train_xgb,y_train)

#Evaluation
ypred_train = xgb.predict(X_train_xgb)
ypred_test  = xgb.predict(X_test_xgb)

print("Train Accuracy :",accuracy_score(y_train,ypred_train))
print("cross validation score :",cross_val_score(xgb,X_train_xgb,y_train,cv=5,scoring="accuracy").mean())
print("Test Accuracy :",accuracy_score(y_test,ypred_test))
```

```
Train Accuracy : 0.9249106078665077
cross validation score : 0.9249129353233831
Test Accuracy : 0.8761904761904762
```

# Step-5 Save the Best model

In [75]:

```
from joblib import dump

dump(xgb,"Extrovert vs Introvert.joblib")
```

Out[75]:

```
['Extrovert vs Introvert.joblib']
```

# Step-6 Predict on new data

## Person-1:-

In [76]:

```
input_data ={"Time_spent_Alone":5,
             "Stage_fear":"Yes",
             "Social_event_attendance":0,
             "Going_outside":2,
             "Friends_circle_size":1,
             "Post_frequency":1,
             }
```

In [77]:

```
df = pd.DataFrame(input_data,index=[0])
df
```

Out[77]:

| | Time_spent_Alone | Stage_fear | Social_event_attendance | Going_outside | Friends_circle_size | Post_frequency |
|---|---|---|---|---|---|---|
| **0** | 5 | Yes | 0 | 2 | 1 | 1 |

## Apply Data Preprocessing on Unknown data

```
In [78]: df.drop_duplicates(inplace=True,ignore_index=True)
         df.dropna(inplace=True)
         df["Stage_fear"] = df["Stage_fear"].replace({"No":0,"Yes":1})

         X_new = df
```

```
In [79]: X_new
```

Out[79]:

|   | Time_spent_Alone | Stage_fear | Social_event_attendance | Going_outside | Friends_circle_size | Post_frequency |
|---|---|---|---|---|---|---|
| **0** | 5 | 1 | 0 | 2 | 1 | 1 |

```
In [80]: X_new = X_new[["Time_spent_Alone","Stage_fear","Social_event_attendance","Going_outside","Friends_circle_size",
         xgb.predict(X_new)
```

```
Out[80]: array([1])
```

### This Person is an Introvert

## Person-2:-

```
In [81]: input_data ={"Time_spent_Alone":1,
                     "Stage_fear":"No",
                     "Social_event_attendance":9,
                     "Going_outside":6,
                     "Friends_circle_size":11,
                     "Post_frequency":9,
                     }
```

```
In [82]: df = pd.DataFrame(input_data,index=[0])
         df
```

Out[82]:

|   | Time_spent_Alone | Stage_fear | Social_event_attendance | Going_outside | Friends_circle_size | Post_frequency |
|---|---|---|---|---|---|---|
| **0** | 1 | No | 9 | 6 | 11 | 9 |

## Apply Data Preprocessing on unknown data

```
In [83]: df.drop_duplicates(inplace=True,ignore_index=True)
         df.dropna(inplace=True)
         df["Stage_fear"] = df["Stage_fear"].replace({"No":0,"Yes":1})

         X_new = df
```

```
In [84]: X_new
```

Out[84]:

|   | Time_spent_Alone | Stage_fear | Social_event_attendance | Going_outside | Friends_circle_size | Post_frequency |
|---|---|---|---|---|---|---|
| **0** | 1 | 0 | 9 | 6 | 11 | 9 |

```
In [85]: X_new = X_new[["Time_spent_Alone","Stage_fear","Social_event_attendance","Going_outside","Friends_circle_size",
         xgb.predict(X_new)
```

```
Out[85]: array([0])
```

### This Person is an Extrovert

```
In [ ]:
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js