

```
#Loading the Libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
#loading the data
data=pd.read_csv('survey (1).csv')
```

```
# types of data in dataset
```

```
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1259 entries, 0 to 1258
Data columns (total 27 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Timestamp              1259 non-null   object
1   Age                    1259 non-null   int64
2   Gender                 1259 non-null   object
3   Country                1259 non-null   object
4   state                  744 non-null    object
5   self_employed          1241 non-null   object
6   family_history          1259 non-null   object
7   treatment              1259 non-null   object
8   work_interfere         995 non-null    object
9   no_employees           1259 non-null   object
10  remote_work            1259 non-null   object
11  tech_company           1259 non-null   object
12  benefits               1259 non-null   object
13  care_options           1259 non-null   object
14  wellness_program       1259 non-null   object
15  seek_help              1259 non-null   object
16  anonymity              1259 non-null   object
17  leave                  1259 non-null   object
18  mental_health_consequence 1259 non-null   object
19  phys_health_consequence 1259 non-null   object
20  coworkers              1259 non-null   object
21  supervisor             1259 non-null   object
22  mental_health_interview 1259 non-null   object
23  phys_health_interview   1259 non-null   object
24  mental_vs_physical      1259 non-null   object
25  obs_consequence        1259 non-null   object
26  comments               164 non-null    object
dtypes: int64(1), object(26)
memory usage: 265.7+ KB
```

```
# for finding the rows and columns count in the dataset
data.shape

(1259, 27)
```

```
# for finding the details about data
data.describe()
```

	Age	Gender	self_employed	family_history	treatment	no_employees	remote_work
count	1259.000000	1259.000000	1259.000000	1259.000000	1259.000000	1259.000000	1259.000000
mean	20.013503	22.887212	0.144559	0.390786	0.505957	2.783161	0.298650
std	7.360940	9.745733	0.390355	0.488121	0.500163	1.740247	0.457848
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	15.000000	20.000000	0.000000	0.000000	0.000000	1.000000	0.000000
50%	19.000000	20.000000	0.000000	0.000000	1.000000	3.000000	0.000000
75%	24.000000	33.000000	0.000000	1.000000	1.000000	4.000000	1.000000
max	52.000000	48.000000	2.000000	1.000000	1.000000	5.000000	1.000000



```
#for viewing all the columns in the dataset
data.columns
```

```
Index(['Timestamp', 'Age', 'Gender', 'Country', 'state', 'self_employed',
       'family_history', 'treatment', 'work_interfere', 'no_employees',
       'remote_work', 'tech_company', 'benefits', 'care_options',
       'wellness_program', 'seek_help', 'anonymity', 'leave',
       'mental_health_consequence', 'phys_health_consequence', 'coworkers',
       'supervisor', 'mental_health_interview', 'phys_health_interview',
       'mental_vs_physical', 'obs_consequence', 'comments'],
      dtype='object')
```

```
#For viewing first 5 records
data.head()
```

	Timestamp	Age	Gender	Country	state	self_employed	family_history	treatment	work_interfere	no
0	2014-08-27 11:29:31	37	Female	United States	IL	NaN	No	Yes	Often	
1	2014-08-27 11:29:37	44	M	United States	IN	NaN	No	No	Rarely	
2	2014-08-27 11:29:44	32	Male	Canada	NaN	NaN	No	No	Rarely	
3	2014-08-27 11:29:46	31	Male	United Kingdom	NaN	NaN	Yes	Yes	Often	
4	2014-08-27 11:30:22	31	Male	United States	TX	NaN	No	No	Never	

5 rows × 27 columns

```
#For viewing last 5 records
data.tail()
```

	Timestamp	Age	Gender	Country	state	self_employed	family_history	treatment	work_interfere	no
1254	2015-09-12 11:17:21	26	male	United Kingdom	NaN	No	No	Yes	NaN	
1255	2015-09-26 01:07:35	32	Male	United States	IL	No	Yes	Yes	Often	
1256	2015-11-07 12:36:58	34	male	United States	CA	No	Yes	Yes	Sometimes	
1257	2015-11-30 21:25:06	46	f	United States	NC	No	No	No	NaN	
1258	2016-02-01 23:04:31	25	Male	United States	IL	No	Yes	Yes	Sometimes	

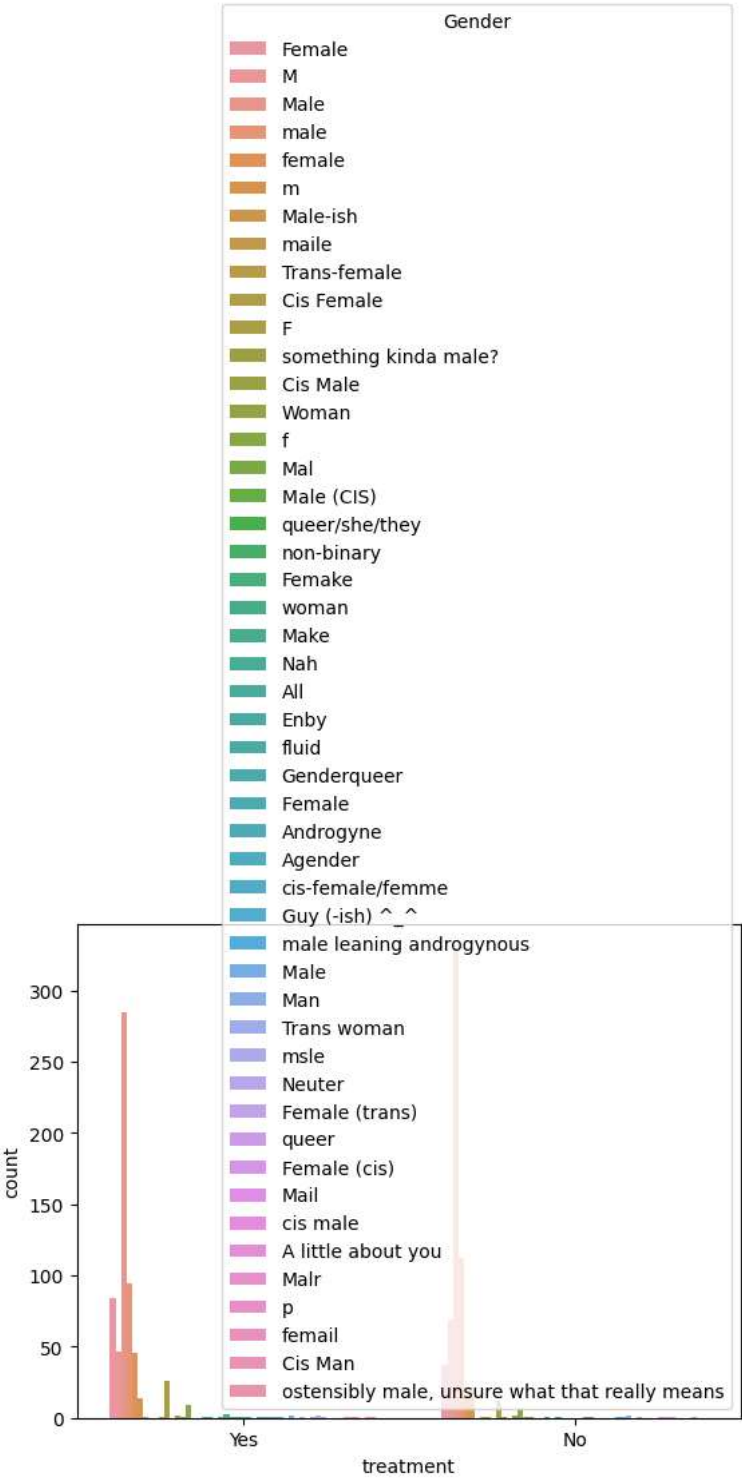
5 rows × 27 columns

```
#for view random samples of dataset
data.sample(4)
```

	Timestamp	Age	Gender	Country	state	self_employed	family_history	treatment	work_interfere
970	2014-08-29 07:12:43	43	Male	United States	MI	No	No	No	Sometimes
1103	2014-08-29 22:08:51	35	Female	United States	WA	No	Yes	No	Sometimes
412	2014-08-27 15:20:22	21	male	United States	MA	No	Yes	No	Never

```
# countplot of treatment and gender
sns.countplot(x='treatment',data=data,hue='Gender')
```

<Axes: xlabel='treatment', ylabel='count'>

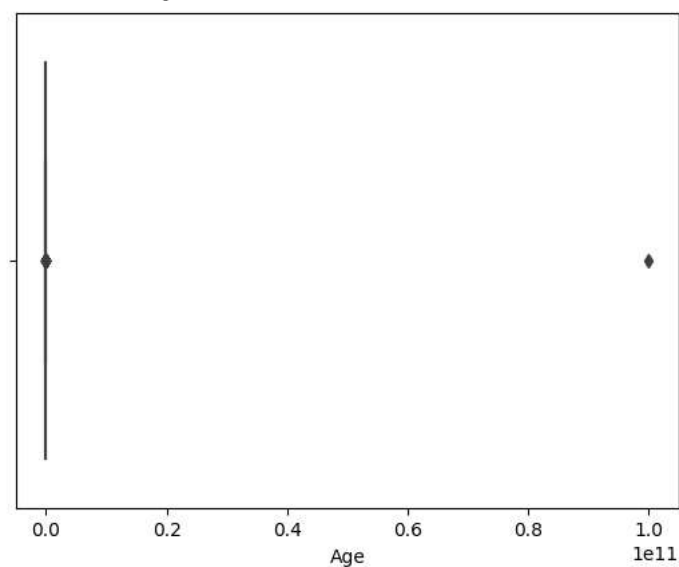


```
#checking null values in a dataset if present
data.isnull().sum()
```

```
Timestamp      0
Age            0
Gender         0
Country        0
state         515
self_employed  18
family_history  0
treatment      0
work_interfere 264
no_employees   0
remote_work    0
tech_company   0
benefits       0
care_options   0
wellness_program 0
seek_help      0
anonymity      0
leave          0
mental_health_consequence 0
phys_health_consequence 0
coworkers      0
supervisor     0
mental_health_interview 0
phys_health_interview 0
mental_vs_physical 0
obs_consequence 0
comments       1095
dtype: int64
```

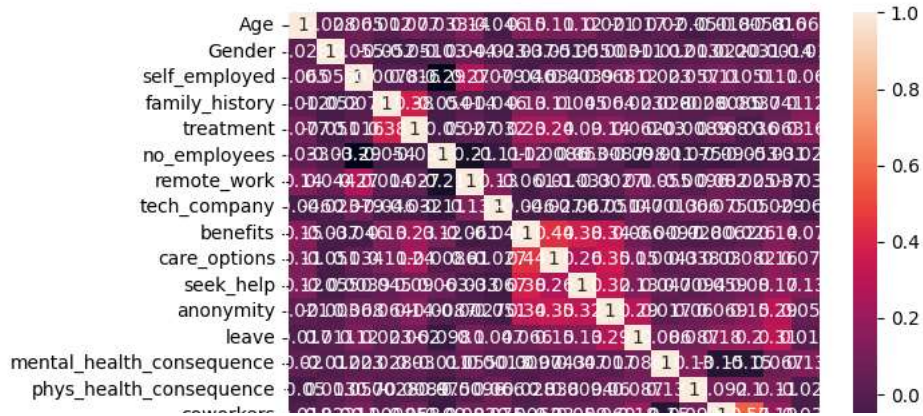
```
# boxplot of Age column:
sns.boxplot(x=data['Age'])
```

```
<Axes: xlabel='Age'>
```



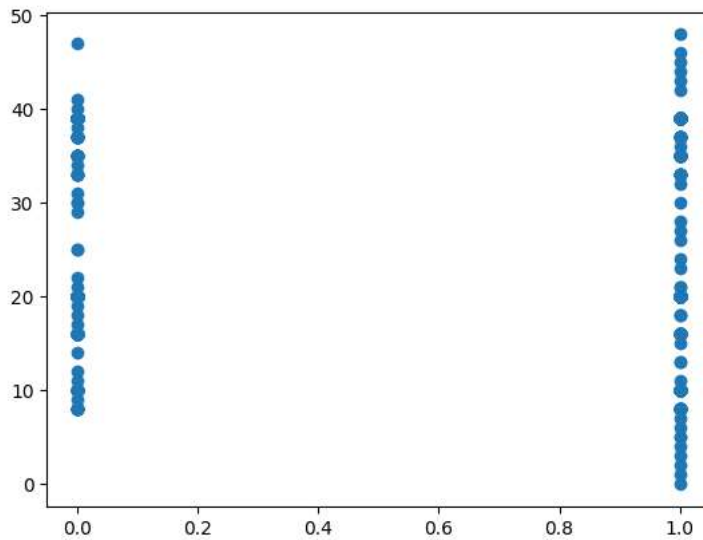
```
#Heatmap of dataset for finding the relationship.
sns.heatmap(data.corr(),annot=True)
```

<Axes: >



```
# plot scatter graph of treatment and gender
plt.scatter(data["treatment"],data["Gender"])
```

<matplotlib.collections.PathCollection at 0x7a1a78370520>



#Encoding (Converting All the Categorical Data into Numerical Data)

```
from sklearn.preprocessing import LabelEncoder
label_encoder=LabelEncoder()
```

```
# df.iloc[:, 1:]=label_encoder.fit_transform(df.iloc[:, 1:])
# Identify columns to label encode (excluding the first column)
columns_to_encode = data.iloc[:, 1:]
# Initialize the label encoder
label_encoder = LabelEncoder()
# Apply label encoding to each column in the DataFrame
for col in columns_to_encode:
    data[col] = label_encoder.fit_transform(data[col])
data.head()
```

```
# to check all columns should be of numerical data types.
data.dtypes
```

```
Timestamp          int64
Age                int64
Gender             int64
Country            int64
state             int64
self_employed      int64
family_history     int64
treatment         int64
work_interfere     int64
no_employees       int64
remote_work        int64
tech_company       int64
benefits           int64
care_options       int64
wellness_program   int64
seek_help          int64
anonymity          int64
leave             int64
mental_health_consequence int64
phys_health_consequence int64
coworkers          int64
supervisor         int64
mental_health_interview int64
phys_health_interview int64
mental_vs_physical int64
obs_consequence    int64
comments           int64
dtype: object
```

```
#Removing the necessary columns which are not necessary.
```

```
data=data.drop(['Timestamp','Country','state','work_interfere','comments','mental_health_interview','phys_health_interview','wellness_program'])
```

```
# checking any missing data for testing purpose :
```

```
data.isnull().sum()
```

```
Age                0
Gender             0
self_employed      0
family_history     0
treatment         0
no_employees       0
remote_work        0
tech_company       0
benefits           0
care_options       0
seek_help          0
anonymity          0
leave             0
mental_health_consequence 0
phys_health_consequence 0
coworkers          0
supervisor         0
mental_vs_physical 0
obs_consequence    0
dtype: int64
```

```
# Training and Testing of dataset
```

```
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score
```

```
#Splitting the data.
```

```
x=data.drop('treatment',axis=1)
y=data['treatment']
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=1)
```

```
# MODELLING
```

```

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

```

#checking Score by using Logistic Regression and Generate the classification Report:

```

model1=LogisticRegression()
model1.fit(x_train,y_train)
model1.score(x_train,y_train)
print("Score of train is :",model1.score(x_train,y_train))

model1.score(x_test,y_test)
print("Score of test is :",model1.score(x_test,y_test))

print(classification_report(y_test,model1.predict(x_test)))

```

```

Score of train is : 0.730883813306852
Score of test is : 0.6904761904761905

```

	precision	recall	f1-score	support
0	0.68	0.67	0.68	121
1	0.70	0.71	0.70	131
accuracy			0.69	252
macro avg	0.69	0.69	0.69	252
weighted avg	0.69	0.69	0.69	252

```

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

```

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
```

Checking the Training and Testing Score by DecisionTreeClassifier and classification Report

```

model2=DecisionTreeClassifier()
model2.fit(x_train,y_train)
model2.score(x_train,y_train)
print("Score of trained model is :",model2.score(x_train,y_train))
model2.score(x_test,y_test)
print("Score of tested model is :",model2.score(x_test,y_test))
print(classification_report(y_test,model2.predict(x_test)))

```

```

Score of trained model is : 1.0
Score of tested model is : 0.623015873015873

```

	precision	recall	f1-score	support
0	0.61	0.58	0.60	121
1	0.63	0.66	0.65	131
accuracy			0.62	252
macro avg	0.62	0.62	0.62	252
weighted avg	0.62	0.62	0.62	252

Checking the Training and Testing Score by RandomForestClassifier and classification Report

```

model3=RandomForestClassifier()
model3.fit(x_train,y_train)
model3.score(x_train,y_train)
print("Score of trained model is :",model3.score(x_train,y_train))

model3.score(x_test,y_test)
print("Score of tested model is :",model3.score(x_test,y_test))

print(classification_report(y_test,model3.predict(x_test)))

```

```

Score of trained model is : 1.0
Score of tested model is : 0.6706349206349206
      precision    recall  f1-score   support

     0       0.66       0.64       0.65       121
     1       0.68       0.69       0.69       131

 accuracy         0.67         0.67         0.67         252
 macro avg         0.67         0.67         0.67         252
 weighted avg         0.67         0.67         0.67         252

```

Pickling and Unpickling:

```

'''pickle:object to binary-dump() is used
unpickle: binary to object load() is used'''

```

```
import pickle
```

```

# syntax: dump(object,open(filename,mode))
# serialize process :
filename="file.pkl"
pickle.dump(data,open(filename,'wb'))

```

```

import pickle
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

```

```

# Save the trained model to a pickle file
with open("model1.pkl", "wb") as f:
    pickle.dump(model1, f)

```

PREDICTION FOR A GIVEN INPUT WHETHER THEY REQUIRED A TREATMENT OR NOT:

```
import pickle
```

```

# Load the trained model using pickle
with open("model1.pkl", "rb") as f:
    model1 = pickle.load(f)

```

```

# Function to take user input and make predictions
def predict_output(user_input):
    output = model1.predict(user_input)
    return output

```

Example usage:

```

print("***** PLEASE FILL THE BELOW DETAILS FOR OUTPUT PREDICTION *****")
Age=int(input("Enter Age"))
print("Enter 0 for Female\nEnter 1 for Male\nEnter 2 for others ")
Gender=int(input("Enter Gender"))
print("Enter 0 for No\nEnter 1 for Yes")
self_employed=int(input("Enter self_employed"))
print("Enter 0 for No\nEnter 1 for Yes")
family_history=int(input("Enter family_history"))
print("Enter 4 for no. of employee between 6-25\nEnter 5 for no. of employee more than 1000\nEnter 2 for no. of employee between 26-100\nEnter 1 for no. of employee between 101-25")
no_employees=int(input("Enter no_employees"))
print("Enter 0 for no and 1 for yes")
remote_work=int(input("remote_work"))
print("Enter 0 for no and 1 for yes")
tech_company=int(input("Enter tech_company"))
print("enter 0 for Don't know\n Enter 1 for No\nEnter 2 for yes")
benefits=int(input("enter benefits"))
print("enter 0 for no\n Enter 1 for Not sure\nEnter 2 for yes")
care_options=int(input("enter care_options"))
print("enter 0 for Don't know\n Enter 1 for No\nEnter 2 for yes")
seek_help=int(input("enter seek_help"))
print("enter 0 for Don't know\n Enter 1 for No\nEnter 2 for yes")
anonymity=int(input("anonymity"))
print("Enter 0 for Don't know\nEnter 1 for Somewhat difficult\nEnter 2 for Somewhat easy\nenter 3 for Very difficult\nEnter 4 for very easy")
leave=int(input("leave"))
print("enter 0 for may be\n Enter 1 for No\nEnter 2 for yes")
mental_health_consequence=int(input("mental_health_consequence"))
print("enter 0 for may be\n Enter 1 for No\nEnter 2 for yes")
phys_health_consequence=int(input("phys_health_consequence"))

```



```

print("enter 0 for no\n Enter 1 for some of them\nEnter 2 for yes")
coworkers=int(input("Enter no of coworkers"))
print("enter 0 for no\n Enter 1 for some of them\nEnter 2 for yes")
supervisor=int(input("Enter supervisor"))
print("enter 0 for Don't know\n Enter 1 for No\nEnter 2 for yes")
mental_vs_physical=int(input("Enter mental_vs_physical"))
print("Enter 0 for No\nEnter 1 for Yes")
obs_consequence=int(input("Enter obs_consequence"))

user_input_data = [[Age,Gender,self_employed,family_history,no_employees,remote_work,tech_company,benefits,care_options,seek_help,anonymity,1
result = predict_output(user_input_data)
if result[0] == 0:
    print("There is no treatment reaquired")
else:
    print("Yes The treatment required")

```

```

Enter Gender1
Enter 0 for No
Enter 1 for Yes
Enter self_employed1
Enter 0 for No
Enter 1 for Yes
Enter family_history1
Enter 4 for no. of employee between 6-25
Enter 5 for no. of employee more than 1000
Enter 2 for no. of employee between 26-100
Enter 1 for no. of employee between 100-500
Enter 0 for no. of employee between 1-5
Enter 3 for no. of employee between 500-1000

Enter no_employees0
Enter 0 for no and 1 for yes
remote_work0
Enter 0 for no and 1 for yes
Enter tech_company0
enter 0 for Don't know
    Enter 1 for No\nEnter 2 for yes
enter benefits1
enter 0 for no
    Enter 1 for Not sure\nEnter 2 for yes
enter care_options1
enter 0 for Don't know
    Enter 1 for No\nEnter 2 for yes
enter seek_help2
enter 0 for Don't know
    Enter 1 for No\nEnter 2 for yes
anonymity1
Enter 0 for Don't know
Enter 1 for Somewhat difficult
Enter 2 for Somewhat easy\enter 3 for Very difficult
Enter 4 for very easy
leave4
enter 0 for may be
    Enter 1 for No\nEnter 2 for yes
mental_health_consequence1
enter 0 for may be
    Enter 1 for No\nEnter 2 for yes
phys_health_consequence2
enter 0 for no
    Enter 1 for some of them\nEnter 2 for yes
Enter no of coworkers1
enter 0 for no
    Enter 1 for some of them\nEnter 2 for yes
Enter supervisor1
enter 0 for Don't know
    Enter 1 for No\nEnter 2 for yes
Enter mental_vs_physical1
Enter 0 for No
Enter 1 for Yes
Enter obs_consequence0
Yes The treatment required
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but LogisticRegression
warnings.warn(

```

✓ 0s completed at 8:02 AM

● ✕