PHASE 3

1. **Assessing Reach:** Measure the extent of the public health awareness campaigns' penetration into the target audience by analyzing the distribution of campaign materials and messages across various channels.
2. **Evaluating Awareness Levels:** Determine the effectiveness of the campaigns in conveying the intended health messages by assessing the changes in public awareness levels before and after the campaigns.
3. **Measuring Impact:** Evaluate the impact of the public health campaigns on the audience's perception and behavior concerning the specific health issues targeted by the campaigns, aiming to identify any positive shifts in attitudes or actions.
4. **Identifying Target Audience Segments:** Segment the audience based on demographic data to identify specific groups that responded positively to the campaign, enabling the customization of future strategies to effectively target these segments.
5. **Analyzing Engagement Metrics:** Assess the level of engagement generated by the campaigns across different platforms, such as social media, websites, and community events, to understand the audience's active participation and interaction with the campaign materials.
6. **Determining Effectiveness of Communication Channels:** Analyze the effectiveness of different communication channels used in the campaigns, including social media, traditional media, and direct outreach, to identify the most impactful channels for disseminating public health messages.
7. **Assessing Long-Term Impact:** Evaluate the long-term impact of the campaigns on public health outcomes, such as changes in behavior, increased health-seeking actions, and a reduction in negative health indicators within the targeted communities.
8. **Comparative Analysis with Similar Campaigns:** Compare the outcomes of the current public health campaigns with those of similar past campaigns to identify best practices and areas for improvement, enabling the formulation of more effective future strategies.

```
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt
```

```
data = pd.read_csv('/content/survey.csv')
data.head()
```

	Timestamp	Age	Gender	Country	state	self_employed	family_history	treatment	work_interfere	no_employees	...	leave	mer
0	27-08-2014 11:29	37	Female	United States	IL	NaN	No	Yes	Often	Jun-25	...	Somewhat easy	
1	27-08-2014 11:29	44	M	United States	IN	NaN	No	No	Rarely	More than 1000	...	Don't know	
2	27-08-2014 11:29	32	Male	Canada	NaN	NaN	No	No	Rarely	Jun-25	...	Somewhat difficult	
3	27-08-2014 11:29	31	Male	United Kingdom	NaN	NaN	Yes	Yes	Often	26-100	...	Somewhat difficult	
4	27-08-2014 11:30	31	Male	United States	TX	NaN	No	No	Never	100-500	...	Don't know	

5 rows × 27 columns

```
#check missing data
if data.isnull().sum().sum() == 0 :
    print ('There is no missing data in our dataset')
else:
    print('There is {} missing data in our dataset '.format(data.isnull().sum().sum()))
```

```
There is 1892 missing data in our dataset
```

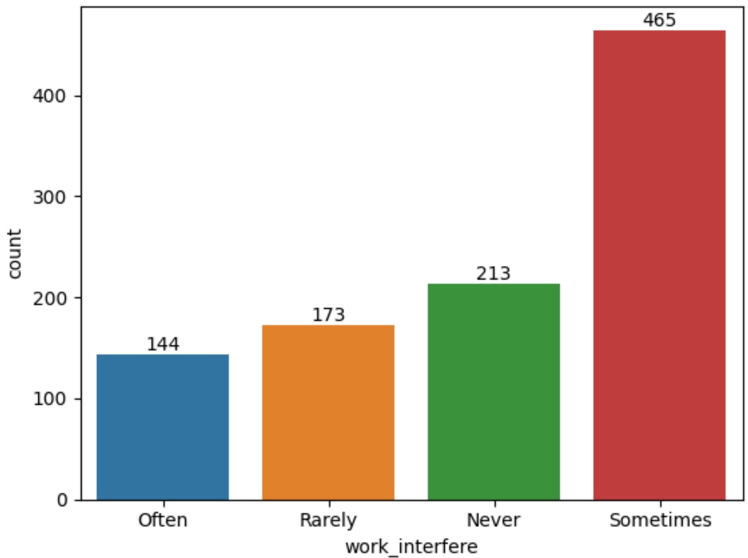
```
#Check our missing data from which columns and how many unique features they have.
frame = pd.concat([data.isnull().sum(), data.nunique(), data.dtypes], axis = 1, sort= False)
frame
```

	0	1	2
Timestamp	0	884	object
Age	0	53	int64
Gender	0	49	object
Country	0	48	object
state	515	45	object
self_employed	18	2	object
family_history	0	2	object
treatment	0	2	object
work_interfere	264	4	object
no_employees	0	6	object
remote_work	0	2	object
tech_company	0	2	object
benefits	0	3	object
care_options	0	3	object
wellness_program	0	3	object
seek_help	0	3	object
anonymity	0	3	object
leave	0	5	object
mental_health_consequence	0	3	object
phys_health_consequence	0	3	object
coworkers	0	3	object
supervisor	0	3	object

```
#Look at what is in the 'Work_interfere' column to choose a suitable method to fill nan values.
data['work_interfere'].unique()
```

```
array(['Often', 'Rarely', 'Never', 'Sometimes', nan], dtype=object)
```

```
#Plot **work_interfere**
ax = sns.countplot(data = data , x = 'work_interfere');
#Add the value of each parametr on the Plot
ax.bar_label(ax.containers[0]);
```




```
from sklearn.impute import SimpleImputer
import numpy as np
columns_to_drop = ['state', 'comments', 'Timestamp']
for column in columns_to_drop:
    if column in data.columns:
        data = data.drop(columns=[column])
```

```
# Fill in missing values in work_interfere column
```

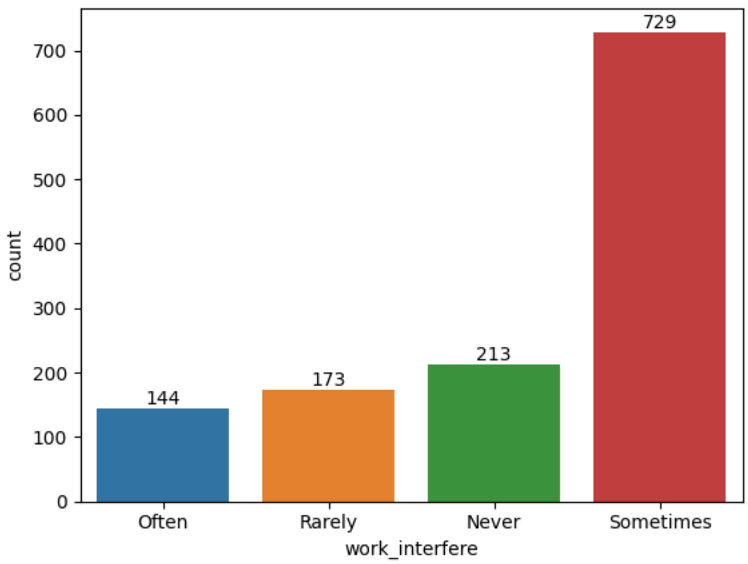
```
data['work_interfere'] = np.ravel(SimpleImputer(strategy = 'most_frequent').fit_transform(data['work_interfere'].values.reshape(-1,1)))
data['self_employed'] = np.ravel(SimpleImputer(strategy = 'most_frequent').fit_transform(data['self_employed'].values.reshape(-1,1)))

data.head()
```

	Age	Gender	Country	self_employed	family_history	treatment	work_interfere	no_employees	remote_work	tech_company	...	anon
0	37	Female	United States	No	No	Yes	Often	Jun-25	No	Yes	...	
1	44	M	United States	No	No	No	Rarely	More than 1000	No	No	...	Don'
2	32	Male	Canada	No	No	No	Rarely	Jun-25	No	Yes	...	Don'
3	31	Male	United Kingdom	No	Yes	Yes	Often	26-100	No	Yes	...	
4	31	Male	United States	No	No	No	Never	100-500	Yes	Yes	...	Don'

5 rows × 24 columns

```
ax = sns.countplot(data=data, x='work_interfere');
ax.bar_label(ax.containers[0]);
```




```
#Check unique data in gender columns
print(data['Gender'].unique())
print('')
print('-'*75)
print('')
#Check number of unique data too.
print('number of unique Gender in our dataset is :', data['Gender'].nunique())
```

```
['Female' 'M' 'Male' 'male' 'female' 'm' 'Male-ish' 'maile' 'Trans-female'
 'Cis Female' 'F' 'something kinda male?' 'Cis Male' 'Woman' 'f' 'Mal'
 'Male (CIS)' 'queer/she/they' 'non-binary' 'Femake' 'woman' 'Make' 'Nah'
 'All' 'Enby' 'fluid' 'Genderqueer' 'Female ' 'Androgyne' 'Agender'
 'cis-female/femme' 'Guy (-ish) ^_^' 'male leaning androgynous' 'Male '
 'Man' 'Trans woman' 'msle' 'Neuter' 'Female (trans)' 'queer'
 'Female (cis)' 'Mail' 'cis male' 'A little about you' 'Malr' 'p' 'femail'
 'Cis Man' 'ostensibly male, unsure what that really means']

---------------------------------------------------------------------------

number of unique Gender in our dataset is : 49
```

```
#Gender data contains dictation problems, nonsense answers, and too unique Genders.
#_So Let's clean it and organize it into Male, Female, and other categories

data['Gender'].replace(['Male ', 'male', 'M', 'm', 'Male', 'Cis Male',
                     'Man', 'cis male', 'Mail', 'Male-ish', 'Male (CIS)',
                      'Cis Man', 'msle', 'Malr', 'Mal', 'maile', 'Make',], 'Male', inplace = True)

data['Gender'].replace(['Female ', 'female', 'F', 'f', 'Woman', 'Female',
                     'femail', 'Cis Female', 'cis-female/femme', 'Femake', 'Female (cis)',
```

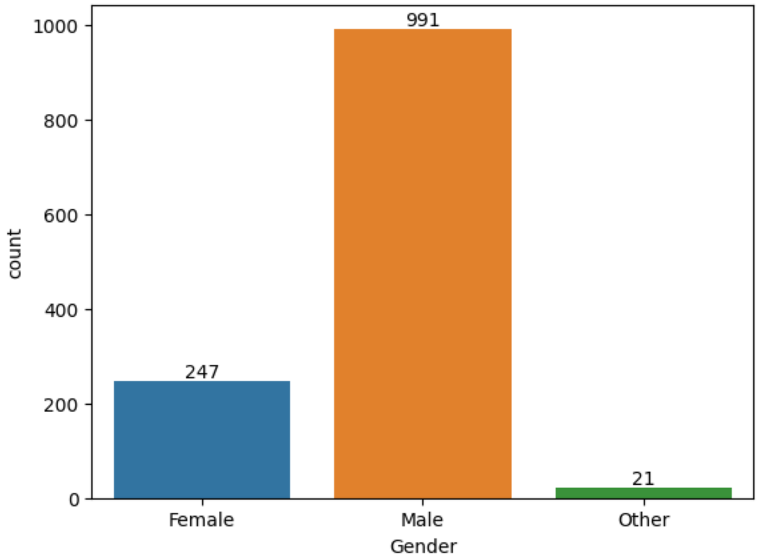
```
                     'woman',], 'Female', inplace = True)

data["Gender"].replace(['Female (trans)', 'queer/she/they', 'non-binary',
                     'fluid', 'queer', 'Androgyne', 'Trans-female', 'male leaning androgynous',
                      'Agender', 'A little about you', 'Nah', 'All',
                      'ostensibly male, unsure what that really means',
                      'Genderqueer', 'Enby', 'p', 'Neuter', 'something kinda male?',
                      'Guy (-ish) ^_^', 'Trans woman',], 'Other', inplace = True)

print(data['Gender'].unique())
```

```
['Female' 'Male' 'Other']
```

```
#Plot Genders column after cleaning and new categorizing
ax = sns.countplot(data=data, x='Gender');
ax.bar_label(ax.containers[0]);
```




```
#Our data is clean now ? let's see.
if data.isnull().sum().sum() == 0:
    print('There is no missing data')
else:
    print('There is {} missing data'.format(data.isnull().sum().sum()))
```

```
There is no missing data
```

```
#Let's check duplicated data.
if data.duplicated().sum() == 0:
    print('There is no duplicated data:')
else:
    print('Tehre is {} duplicated data:'.format(data.duplicated().sum()))
    #If there is duplicated data drop it.
    data.drop_duplicates(inplace=True)

print('-'*50)
print(data.duplicated().sum())
```

```
Tehre is 4 duplicated data:
--------------------------------------------------
0
```

```
#Look unique data in Age column
data['Age'].unique()
```

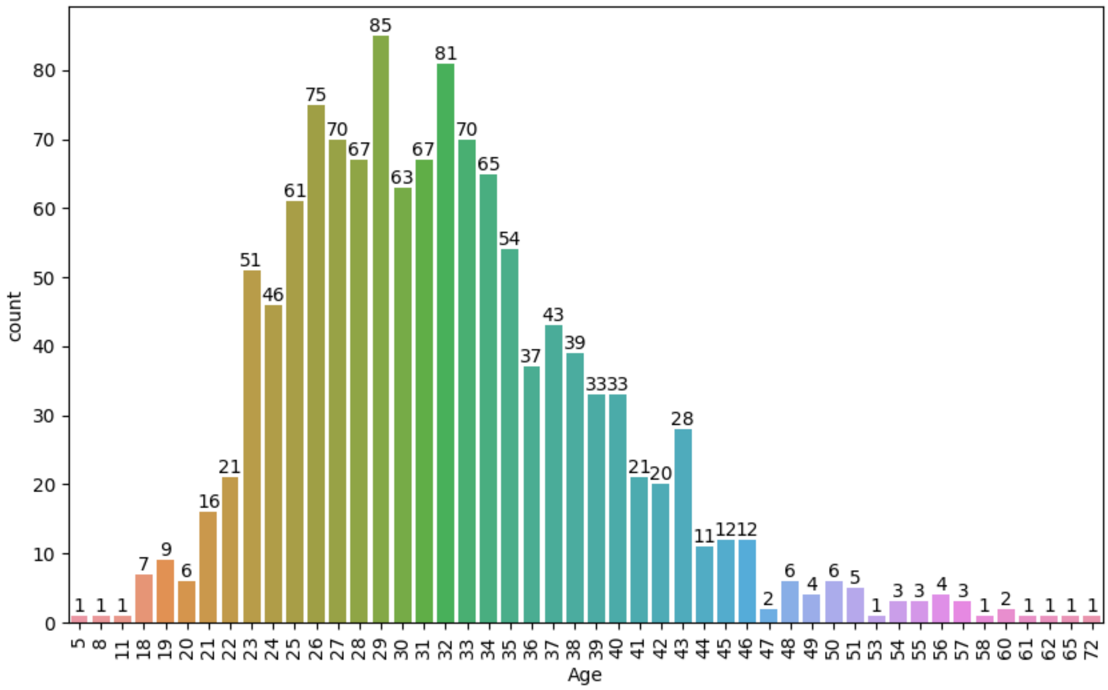
```
array([         37,          44,          32,          31,          33,
                35,          39,          42,          23,          29,
                36,          27,          46,          41,          34,
                30,          40,          38,          50,          24,
                18,          28,          26,          22,          19,
                25,          45,          21,         -29,          43,
                56,          60,          54,         329,          55,
       99999999999,          48,          20,          57,          58,
                47,          62,          51,          65,          49,
             -1726,           5,          53,          61,           8,
                11,          -1,          72])
```

```
#We had a lot of nonsense answers in the Age column too
#This filtering will drop entries exceeding 100 years and those indicating negative values.
data.drop(data[data['Age']<0].index, inplace = True)
data.drop(data[data['Age']>99].index, inplace = True)

print(data['Age'].unique())
```

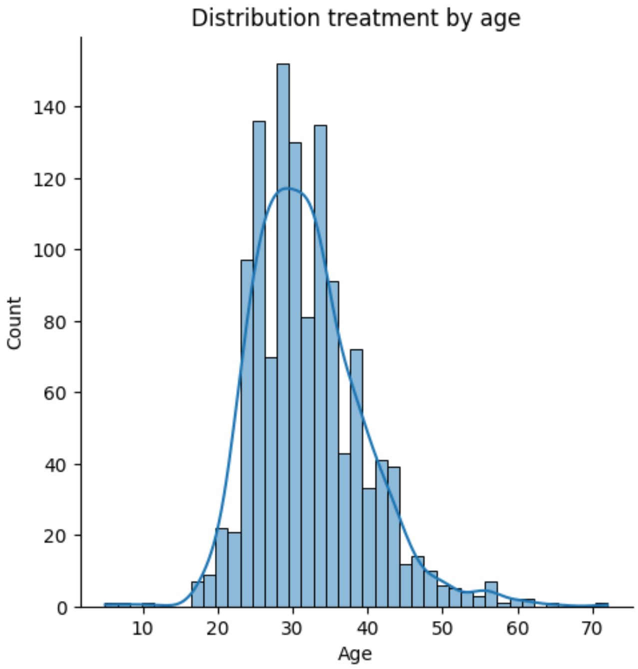
```
[37 44 32 31 33 35 39 42 23 29 36 27 46 41 34 30 40 38 50 24 18 28 26 22
 19 25 45 21 43 56 60 54 55 48 20 57 58 47 62 51 65 49  5 53 61  8 11 72]
```

```
#Let's see the Age distribution in this dataset.
plt.figure(figsize = (10,6))
age_range_plot = sns.countplot(data = data, x = 'Age');
age_range_plot.bar_label(age_range_plot.containers[0]);
plt.xticks(rotation=90);
```



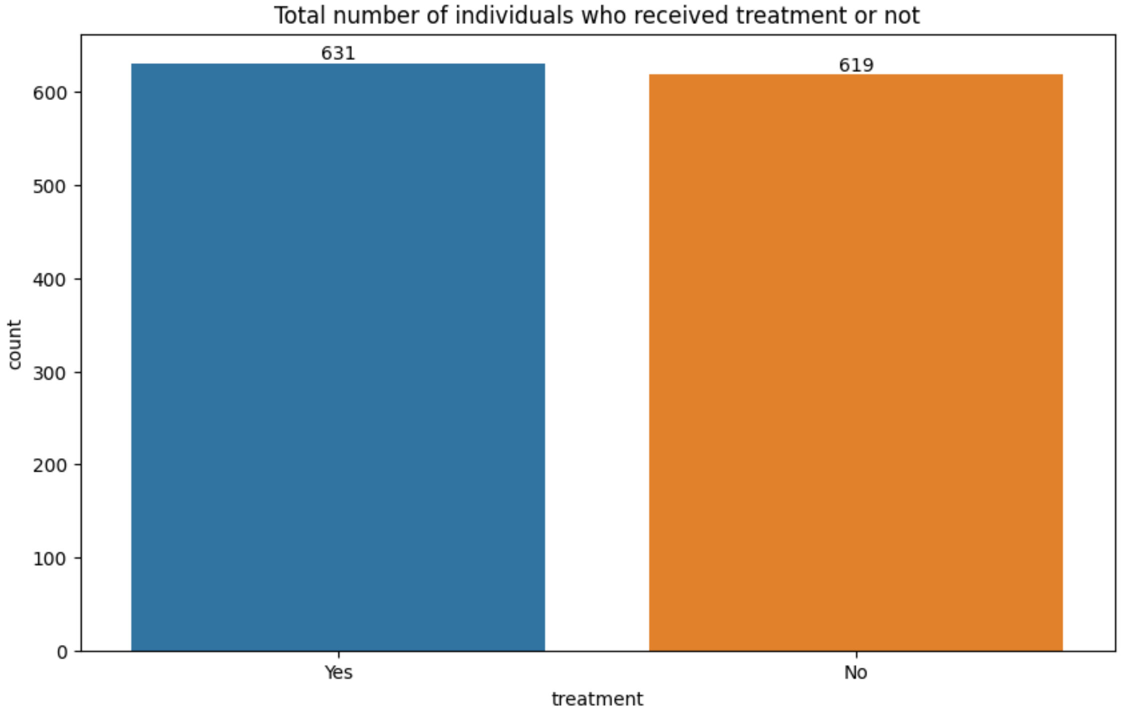

```
#In this plot moreover on Age distribution we can see treatment distribution by age
plt.figure(figsize=(10, 6));
sns.displot(data['Age'], kde = 'treatment');
plt.title('Distribution treatment by age');
```

```
<Figure size 1000x600 with 0 Axes>
```




```
#In this plot We can see Total number of individuals who received treatment or not.
plt.figure(figsize = (10,6));
```

```
treat = sns.countplot(data = data,  x = 'treatment');
treat.bar_label(treat.containers[0]);
plt.title('Total number of individuals who received treatment or not');
```




```
#Check Dtypes
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1250 entries, 0 to 1258
Data columns (total 24 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Age                        1250 non-null   int64
 1   Gender                     1250 non-null   object
 2   Country                    1250 non-null   object
 3   self_employed              1250 non-null   object
 4   family_history             1250 non-null   object
 5   treatment                  1250 non-null   object
 6   work_interfere             1250 non-null   object
 7   no_employees               1250 non-null   object
 8   remote_work                1250 non-null   object
 9   tech_company               1250 non-null   object
 10  benefits                   1250 non-null   object
 11  care_options               1250 non-null   object
 12  wellness_program           1250 non-null   object
 13  seek_help                  1250 non-null   object
 14  anonymity                  1250 non-null   object
 15  leave                      1250 non-null   object
 16  mental_health_consequence  1250 non-null   object
 17  phys_health_consequence    1250 non-null   object
 18  coworkers                  1250 non-null   object
 19  supervisor                 1250 non-null   object
 20  mental_health_interview    1250 non-null   object
 21  phys_health_interview      1250 non-null   object
 22  mental_vs_physical         1250 non-null   object
 23  obs_consequence            1250 non-null   object
dtypes: int64(1), object(23)
memory usage: 244.1+ KB
```

```
#Use LabelEncoder to change the Dtypes to 'int'
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
#Make the dataset include all the columns we need to change their dtypes
columns_to_encode = ['Gender', 'Country', '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']
#Write a Loop for fitting LabelEncoder on columns_to_encode
for columns in columns_to_encode:
    data[columns] = le.fit_transform(data[columns])

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1250 entries, 0 to 1258
Data columns (total 24 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Age                        1250 non-null   int64
 1   Gender                     1250 non-null   int64
 2   Country                    1250 non-null   int64
 3   self_employed              1250 non-null   int64
 4   family_history             1250 non-null   int64
 5   treatment                  1250 non-null   int64
 6   work_interfere             1250 non-null   int64
 7   no_employees               1250 non-null   int64
 8   remote_work                1250 non-null   int64
 9   tech_company               1250 non-null   int64
 10  benefits                   1250 non-null   int64
 11  care_options               1250 non-null   int64
 12  wellness_program           1250 non-null   int64
 13  seek_help                  1250 non-null   int64
 14  anonymity                  1250 non-null   int64
 15  leave                      1250 non-null   int64
 16  mental_health_consequence  1250 non-null   int64
 17  phys_health_consequence    1250 non-null   int64
 18  coworkers                  1250 non-null   int64
 19  supervisor                 1250 non-null   int64
 20  mental_health_interview    1250 non-null   int64
 21  phys_health_interview      1250 non-null   int64
 22  mental_vs_physical         1250 non-null   int64
 23  obs_consequence            1250 non-null   int64
dtypes: int64(24)
memory usage: 244.1 KB
```

```
#Let's check Standard deviation
data.describe()
```

	Age	Gender	Country	self_employed	family_history	treatment	work_interfere	no_employees	remote_work	tecl
count	1250.00000	1250.00000	1250.000000	1250.000000	1250.000000	1250.000000	1250.000000	1250.000000	1250.000000	12
mean	32.02400	0.81760	37.792800	0.114400	0.390400	0.504800	2.128000	2.786400	0.298400	
std	7.38408	0.42388	13.334981	0.318424	0.488035	0.500177	1.165806	1.738733	0.457739	
min	5.00000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	27.00000	1.00000	42.000000	0.000000	0.000000	0.000000	1.000000	1.000000	0.000000	
50%	31.00000	1.00000	45.000000	0.000000	0.000000	1.000000	3.000000	3.000000	0.000000	
75%	36.00000	1.00000	45.000000	0.000000	1.000000	1.000000	3.000000	4.000000	1.000000	
max	72.00000	2.00000	46.000000	1.000000	1.000000	1.000000	3.000000	5.000000	1.000000	

8 rows × 24 columns

```
from sklearn.preprocessing import MaxAbsScaler, StandardScaler

data['Age'] = MaxAbsScaler().fit_transform(data[['Age']])
data['Country'] = StandardScaler().fit_transform(data[['Country']])
data['work_interfere'] = StandardScaler().fit_transform(data[['work_interfere']])
data['no_employees'] = StandardScaler().fit_transform(data[['no_employees']])
data['leave'] = StandardScaler().fit_transform(data[['leave']])

data.describe()
```

	Age	Gender	Country	self_employed	family_history	treatment	work_interfere	no_employees	remote_work
mean	0.444778	0.81760	3.979039e-17	0.114400	0.390400	0.504800	-1.193712e-16	-1.705303e-17	0.298400
std	0.102557	0.42388	1.000400e+00	0.318424	0.488035	0.500177	1.000400e+00	1.000400e+00	0.457739
min	0.069444	0.00000	-2.835244e+00	0.000000	0.000000	0.000000	-1.826077e+00	-1.603187e+00	0.000000
25%	0.375000	1.00000	3.156273e-01	0.000000	0.000000	0.000000	-9.679583e-01	-1.027826e+00	0.000000
50%	0.430556	1.00000	5.406895e-01	0.000000	0.000000	1.000000	7.482798e-01	1.228972e-01	0.000000
75%	0.500000	1.00000	5.406895e-01	0.000000	1.000000	1.000000	7.482798e-01	6.982587e-01	1.000000
max	1.000000	2.00000	6.157103e-01	1.000000	1.000000	1.000000	7.482798e-01	1.273620e+00	1.000000

8 rows × 24 columns