***Public Health Awareness***

*# I am using Pandas for data manipulation.*

*# Matplotlib, Seaborn and hvPlot for Data Visualization*

In [2]:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import os

In [3]:

*# I will then create the file path and file name, load my dataset, create a dataframe and assign it to the name "df"*

In [4]:

os.listdir("/kaggle/input/")

Out[4]:

['mental-health-in-tech-survey']

In [5]:

df = pd.read\_csv("/kaggle/input/mental-health-in-tech-survey/survey.csv")

In [6]:

*# To see the shape of the dataframe:*

df.shape

Out[6]:

(1259, 27)

Data Cleaning: Missing Values

In [7]:

*# We seek for information about our data using .info()*

*# It will help us see both the missing values & data types for each attribute*

*# The index, columns, and number of missing values*

df.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

In [8]:

*# I can see the exact number of missing values there is using the isna().sum()*

In [9]:

df.isna().sum()

Out[9]:

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

In [10]:

*# From the above analyses, we see that;*

*# "state",*

*# "self\_employed" and*

*# "work\_interfere"*

*# "comments"*

*# all have missing values*

In [11]:

*# In the future, I will drop "state" and "comments' columns because we won't be needing them*

*# Our analysis will be done at "country" level, since our dataset covers different countries*

*# So I will be filling up the missing values for just "self\_employed" and "work\_interfere" columns*

In [12]:

*# I want to see the percentage missing values in "self\_employed" and "work\_interfere" columns*

*# I will use the round() function to round them to 2 decimal places*

self\_employed\_percent = (df["self\_employed"].isnull().sum()/len(df["self\_employed"]))\*100

work\_interfere\_percent = (df["work\_interfere"].isnull().sum()/len(df["work\_interfere"]))\*100

print(f"The percentage of missing values in self\_employed column is **{**round(self\_employed\_percent, 2)**}**%")

print(f"The percentage of missing values in work\_interfere column is **{**round(work\_interfere\_percent, 2)**}**%")

The percentage of missing values in self\_employed column is 1.43%

The percentage of missing values in work\_interfere column is 20.97%

In [13]:

*# Because I have approx. 80% of both, I will fill up both columns with the mode for each.*

In [14]:

df["self\_employed"] = df["self\_employed"].fillna(df["self\_employed"].mode()[0])

df["work\_interfere"] = df["work\_interfere"].fillna(df["work\_interfere"].mode()[0])

In [15]:

*# I can now take a peek at our dataframe to see how it looks like, using the .head() method*

In [16]:

df.head()

Out[16]:

|  | Timestamp | Age | Gender | Country | state | self\_employed | family\_history | treatment | work\_interfere | no\_employees | ... | leave | mental\_health\_consequence | phys\_health\_consequence | coworkers | supervisor | mental\_health\_interview | phys\_health\_interview | mental\_vs\_physical | obs\_consequence | comments |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 2014-08-27 11:29:31 | 37 | Female | United States | IL | No | No | Yes | Often | 6-25 | ... | Somewhat easy | No | No | Some of them | Yes | No | Maybe | Yes | No | NaN |
| 1 | 2014-08-27 11:29:37 | 44 | M | United States | IN | No | No | No | Rarely | More than 1000 | ... | Don't know | Maybe | No | No | No | No | No | Don't know | No | NaN |
| 2 | 2014-08-27 11:29:44 | 32 | Male | Canada | NaN | No | No | No | Rarely | 6-25 | ... | Somewhat difficult | No | No | Yes | Yes | Yes | Yes | No | No | NaN |
| 3 | 2014-08-27 11:29:46 | 31 | Male | United Kingdom | NaN | No | Yes | Yes | Often | 26-100 | ... | Somewhat difficult | Yes | Yes | Some of them | No | Maybe | Maybe | No | Yes | NaN |
| 4 | 2014-08-27 11:30:22 | 31 | Male | United States | TX | No | No | No | Never | 100-500 | ... | Don't know | No | No | Some of them | Yes | Yes | Yes | Don't know | No | NaN |

5 rows × 27 columns

In [17]:

*# Final step is to drop the "state" and "comments" columns*

*# And then, I will confirm that all missing values have been fixed*

In [18]:

df.drop(["state", "comments"], axis=1, inplace=True)

In [19]:

df.isna().sum()

Out[19]:

Timestamp 0

Age 0

Gender 0

Country 0

self\_employed 0

family\_history 0

treatment 0

work\_interfere 0

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

dtype: int64

In [20]:

*# Our dataframe is now free of any missing values.*

Univariate Analyses

In [21]:

*# First, let's see all our columns*

In [22]:

df.columns

Out[22]:

Index(['Timestamp', 'Age', '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'],

dtype='object')

In [23]:

*# First, I will look at some of the interesting attributes and analyse them*

Attribute 1: Country

In [24]:

plt.figure(figsize=(17,5))

ax = sns.countplot(x='Country', data=df)

ax.set\_xticklabels(

ax.get\_xticklabels(),

rotation=45,

horizontalalignment='right'

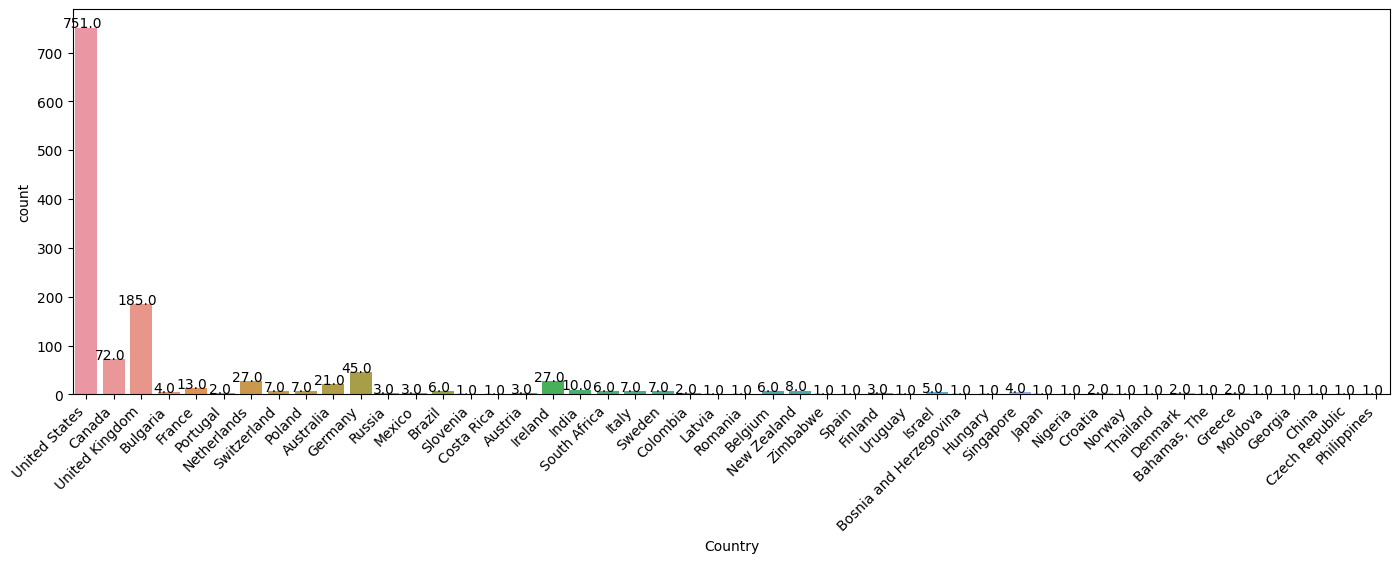
)

None *# So it won't show the label objects*

*# Then we also display the values for each bar above it;*

for p **in** ax.patches:

ax.annotate(p.get\_height(), (p.get\_x()+0.25, p.get\_height()+0.01), ha='center')



In [25]:

*# We clearly see above that the countries with the highest number of mental health issues in Tech are;*

*# United States*

*# United Kingdom*

*# Canada*

In [26]:

*# But why is this so?*

*# I can proceed to look at other factors like "Age", "remote\_work", "family\_history", "no\_employees", etc*

Attribute 2: Age

In [27]:

*# To analyze this, it's best I group them into categories*

*# To help me get a good enough group size, I will see the min, max, median and mean of the distribution for Age*

In [28]:

min\_age = df["Age"].min()

max\_age = df["Age"].max()

mean\_age = df["Age"].mean()

median\_age = df["Age"].median()

print(f"Min: **{**min\_age**}**, **\n**Max: **{**max\_age**}**, **\n**Mean: **{**mean\_age**}**, **\n**Median: **{**median\_age**}**")

Min: -1726,

Max: 99999999999,

Mean: 79428148.31135821,

Median: 31.0

In [29]:

*# We see from above, that even though all Age collumns are filled, some of the values are invalid*

In [30]:

*# By my own personal preference, I will replace all invalid values by the median, 31.*

*# I chose this because I believe for people in tech today, 31 seems appropariate to use.*

*# I will also exclude those above 80, and treat them as invalid.*

*# Then I will see if there are other outliers worth looking at.*

In [31]:

df["Age"].unique()

Out[31]:

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])

In [32]:

*# I would like to see the number of values with negative or above 80.*

negative\_age = (df["Age"]<0).sum()

over\_age = (df["Age"]>80).sum()

print(f"Number of negative age entries: **{**negative\_age**}\n**Number of overage: **{**over\_age**}**")

Number of negative age entries: 3

Number of overage: 2

In [33]:

*# Setting them to the median age*

df.loc[df.Age<0, ["Age"]] = df["Age"].median()

df.loc[df.Age>80, ["Age"]] = df["Age"].median()

In [34]:

df["Age"].unique()

Out[34]:

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, 43, 56, 60, 54, 55, 48,

20, 57, 58, 47, 62, 51, 65, 49, 5, 53, 61, 8, 11, 72])

In [35]:

*# We have outliers of 5years, 8years*

*# With our domain knowledge, we can confidently go further to treat these as invalid.*

*# Only 18 and above is accepted to be a legal tech employee.*

*# I will replace them with the median.*

In [36]:

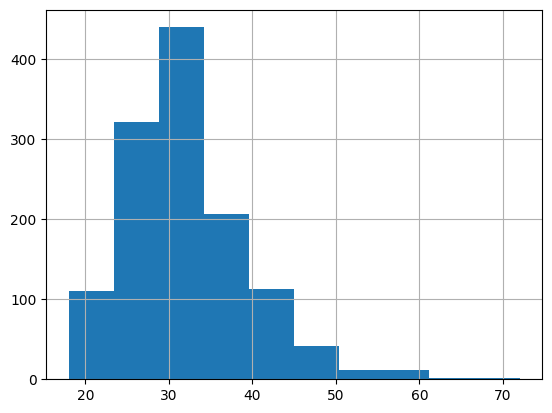
df.loc[df.Age<18, ["Age"]] = df["Age"].median()

In [37]:

df["Age"].hist()

Out[37]:

<Axes: >

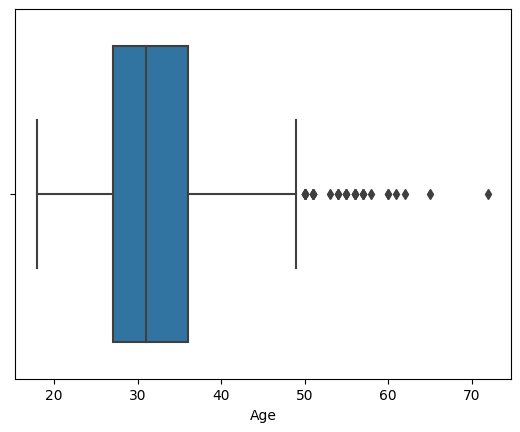


In [38]:

sns.boxplot(x=df["Age"])

Out[38]:

<Axes: xlabel='Age'>



In [39]:

*# Outliers: Age above 50*

*# These can be men and women who entered tech in the 80's*

New Results for Mean. Also calculate for S.Deviation & Variance

In [40]:

import statistics

variance\_age = df["Age"].var()

standard\_dev\_age = statistics.stdev(df["Age"])

print(f"Mean: **{**round(mean\_age, 2)**}**"

f"**\n**Variance: **{**round(variance\_age, 2)**}**"

f"**\n**Standard Deviation: **{**round(standard\_dev\_age, 2)**}**")

Mean: 79428148.31

Variance: 52.79

Standard Deviation: 7.27

Five (5) Other Attributes: No. of Employees, Family History, Remote Work, Self-Employed & Tech Company

In [41]:

plt.figure(figsize=(7,4))

ax = sns.countplot(x='no\_employees', data=df)

ax.set\_xticklabels(ax.get\_xticklabels(),

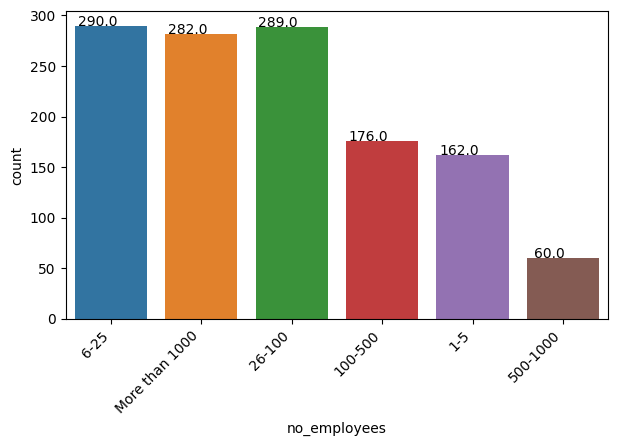
rotation=45,

horizontalalignment='right')

*# Then we also display the values for each bar above it;*

for p **in** ax.patches:

ax.annotate(p.get\_height(), (p.get\_x()+0.25, p.get\_height()+0.01), ha='center')



In [42]:

*# This doesn't help us much, because number of employees isn't directly proportional with cases of mental health.*

*# In fact, there are identical mental health issues with employees working in companies with staff strengths of*

*# 6 - 25*

*# 25 - 100*

*# More than 1000*

In [43]:

plt.figure(figsize=(5,5))

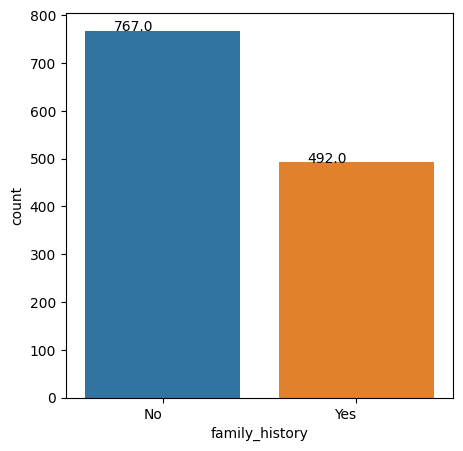
ax = sns.countplot(x='family\_history', data=df)

ax.set\_xticklabels(ax.get\_xticklabels(),

horizontalalignment='right')

for p **in** ax.patches:

ax.annotate(p.get\_height(), (p.get\_x()+0.25, p.get\_height()+0.01), ha='center')



In [44]:

plt.figure(figsize=(5,5))

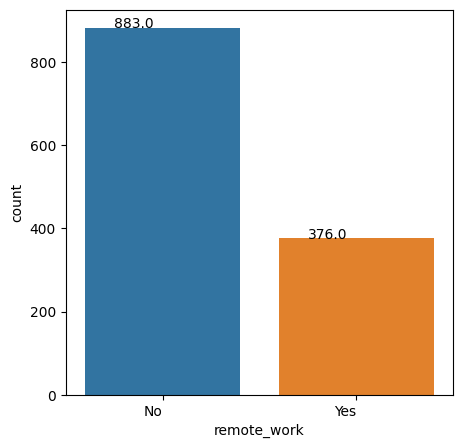
ax = sns.countplot(x='remote\_work', data=df)

ax.set\_xticklabels(ax.get\_xticklabels(),

horizontalalignment='right')

for p **in** ax.patches:

ax.annotate(p.get\_height(), (p.get\_x()+0.25, p.get\_height()+0.01), ha='center')



In [45]:

plt.figure(figsize=(5,5))

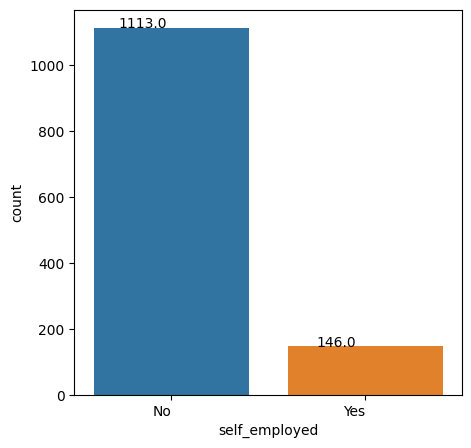
ax = sns.countplot(x='self\_employed', data=df)

ax.set\_xticklabels(ax.get\_xticklabels(),

horizontalalignment='right')

for p **in** ax.patches:

ax.annotate(p.get\_height(), (p.get\_x()+0.25, p.get\_height()+0.01), ha='center')



In [46]:

plt.figure(figsize=(5,5))

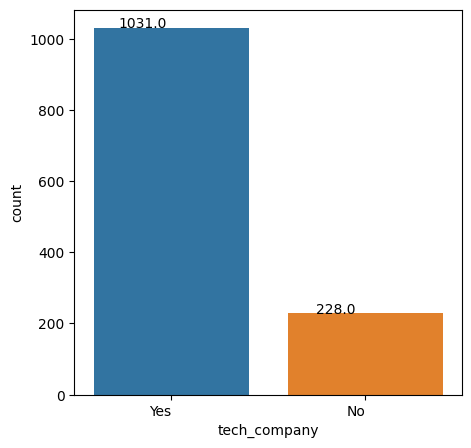
ax = sns.countplot(x='tech\_company', data=df)

ax.set\_xticklabels(ax.get\_xticklabels(),

horizontalalignment='right')

for p **in** ax.patches:

ax.annotate(p.get\_height(), (p.get\_x()+0.25, p.get\_height()+0.01), ha='center')



Attribute 8: Time

In [47]:

df["Timestamp"].head()

Out[47]:

0 2014-08-27 11:29:31

1 2014-08-27 11:29:37

2 2014-08-27 11:29:44

3 2014-08-27 11:29:46

4 2014-08-27 11:30:22

Name: Timestamp, dtype: object

In [48]:

*# For me to work with just the Years, excluding the Time, Months and Days, I will do the following;*

df\_year = pd.to\_datetime(df["Timestamp"]).dt.year

df\_year.head()

Out[48]:

0 2014

1 2014

2 2014

3 2014

4 2014

Name: Timestamp, dtype: int32

In [49]:

*# I will add this year column to my dataframe and plot the histogram*

df["Year"] = df\_year

plt.figure(figsize=(5,5))

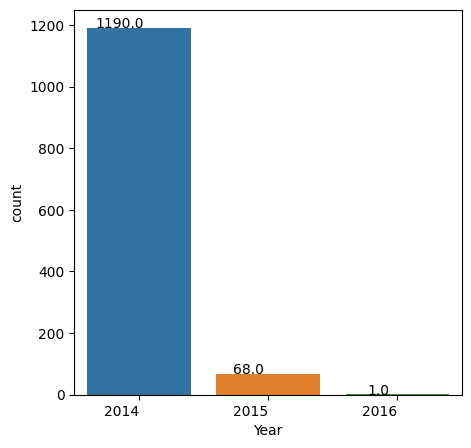
ax = sns.countplot(x='Year', data=df)

ax.set\_xticklabels(ax.get\_xticklabels(),

horizontalalignment='right')

for p **in** ax.patches:

ax.annotate(p.get\_height(), (p.get\_x()+0.25, p.get\_height()+0.01), ha='center')



In [50]:

*# There has to be a reason why only 68 cases of mental health amongst employees was recorded in 2015.*

*# For 2016, we assume it's because the data stopped in Feb 2016.*

Conclusion of Univariate Analysis

In [51]:

*# Nevertheless, we can still draw some very useful insights from the above analyses;*

*# 1.) USA, UK and Canada accounts for the most cases*

*# 2.) Mental Health issues affect more of individuals who are;*

*# Employees*

*# Aged between 25-40*

*# Working in the office, and not remote, and*

*# Working in Tech companies*

Bivariate & Multivariate Analyses

In [52]:

*# I will analyse further to see relationships between attributes and see if there are similarities.*

In [53]:

df.describe()

Out[53]:

|  | Age | Year |
| --- | --- | --- |
| count | 1259.000000 | 1259.00000 |
| mean | 32.069897 | 2014.05560 |
| std | 7.265565 | 0.23268 |
| min | 18.000000 | 2014.00000 |
| 25% | 27.000000 | 2014.00000 |
| 50% | 31.000000 | 2014.00000 |
| 75% | 36.000000 | 2014.00000 |
| max | 72.000000 | 2016.00000 |

In [54]:

*# We see above that only the "Age" column contains numerical values*

*# Therefore, in order to proceed I will have to convert the categorical variables into numerical*

Data Manipulation: Changing from Categorical to Numerical

In [55]:

*# I will assign No: 0 and Yes: 1 where applicable*

*# Then for "Don't Know", "Maybe" I will assign 2*

*# Finally, I will create a new column comprising of the numerical values*

In [56]:

print(f'Family History Unique Entries: **{**df["family\_history"].unique()**}**')

Family History Unique Entries: ['No' 'Yes']

In [57]:

df["family\_history\_num"] = df["family\_history"].map({"No": 0, "Yes": 1})

In [58]:

df[["family\_history", "family\_history\_num"]].head()

Out[58]:

|  | family\_history | family\_history\_num |
| --- | --- | --- |
| 0 | No | 0 |
| 1 | No | 0 |
| 2 | No | 0 |
| 3 | Yes | 1 |
| 4 | No | 0 |

In [59]:

*# I will now proceed to do this for some other key attributes;*

print(f'Self Employed Unique Entries: **{**df["self\_employed"].unique()**}**'

f'**\n**Treatment Unique Entries: **{**df["treatment"].unique()**}**'

f'**\n**Remote Work Unique Entries: **{**df["remote\_work"].unique()**}**'

f'**\n**Benefits Unique Entries: **{**df["benefits"].unique()**}**'

f'**\n**Wellness Program Unique Entries: **{**df["wellness\_program"].unique()**}**'

f'**\n**Seek Help Unique Entries: **{**df["seek\_help"].unique()**}**'

f'**\n**Anonymity Unique Entries: **{**df["anonymity"].unique()**}**'

f'**\n**Mental Health Consequence Unique Entries: **{**df["mental\_health\_consequence"].unique()**}**'

f'**\n**Phy. Health Consequence Unique Entries: **{**df["phys\_health\_consequence"].unique()**}**')

Self Employed Unique Entries: ['No' 'Yes']

Treatment Unique Entries: ['Yes' 'No']

Remote Work Unique Entries: ['No' 'Yes']

Benefits Unique Entries: ['Yes' "Don't know" 'No']

Wellness Program Unique Entries: ['No' "Don't know" 'Yes']

Seek Help Unique Entries: ['Yes' "Don't know" 'No']

Anonymity Unique Entries: ['Yes' "Don't know" 'No']

Mental Health Consequence Unique Entries: ['No' 'Maybe' 'Yes']

Phy. Health Consequence Unique Entries: ['No' 'Yes' 'Maybe']

In [60]:

df["self\_employed\_num"] = df["self\_employed"].map({"No": 0, "Yes": 1})

df["treatment\_num"] = df["treatment"].map({"No": 0, "Yes": 1})

df["remote\_work\_num"] = df["remote\_work"].map({"No": 0, "Yes": 1})

df["benefits\_num"] = df["benefits"].map({"No": 0, "Yes": 1, "Don't know": 2})

df["wellness\_programs\_num"] = df["wellness\_program"].map({"No": 0, "Yes": 1, "Don't know": 2})

df["seek\_help\_num"] = df["seek\_help"].map({"No": 0, "Yes": 1, "Don't know": 2})

df["anonymity\_num"] = df["anonymity"].map({"No": 0, "Yes": 1, "Don't know": 2})

df["mental\_health\_consequence\_num"] = df["mental\_health\_consequence"].map({"No": 0, "Yes": 1, "Maybe": 2})

df["phys\_health\_consequence\_num"] = df["phys\_health\_consequence"].map({"No": 0, "Yes": 1, "Maybe": 2})

In [61]:

*# To plot the heatmap using seaborn, I will extract all numerical attributes and create a new dataframe called df1*

*# I am doing this because the sns.heatmap function can't work with strings or categorical variables*

df1 = pd.DataFrame() *# Empty dataframe*

df1["Year"] = df["Year"]

df1["self\_employed\_num"] = df["self\_employed\_num"]

df1["treatment\_num"] = df["treatment\_num"]

df1["remote\_work\_num"] = df["remote\_work\_num"]

df1["benefits\_num"] = df["benefits\_num"]

df1["wellness\_programs\_num"] = df["wellness\_programs\_num"]

df1["seek\_help\_num"] = df["seek\_help\_num"]

df1["anonymity\_num"] = df["anonymity\_num"]

df1["mental\_health\_consequence\_num"] = df["mental\_health\_consequence\_num"]

df1["phys\_health\_consequence\_num"] = df["phys\_health\_consequence\_num"]

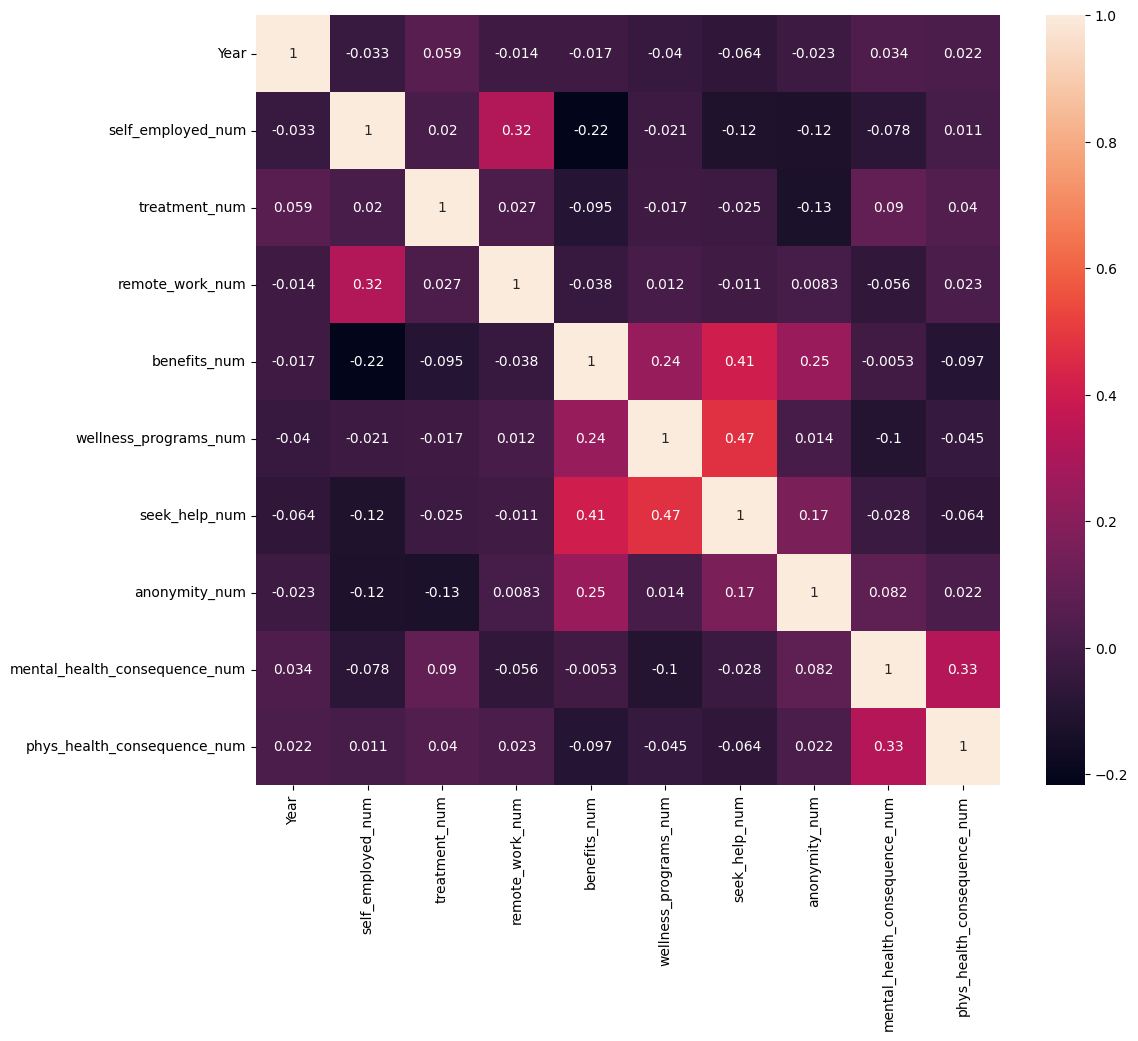
In [62]:

plt.figure(figsize=(12, 10))

sns.heatmap(df1.corr(), annot=True)

Out[62]:

<Axes: >



In [63]:

*# From the diagram above, the attributes that are most related are "seek\_help" and "wellness\_program"*

*# Followed by "seek\_help" and "benefits"*

*# The following is the description of these attributes;*

1.) Wellness\_program: Has your employer ever discussed mental health as part of an employee wellness program?

2.) Seek\_help: Does your employer provide resources to learn more about mental health issues **and** how to seek help?

3.) Benefits: Does your employer provide mental health benefits?

Object `program` not found.

Object `benefits` not found.