COMP6940 Assignment 1 Warren O'Connell 811000293 To start, we will import some packages that we would use to read and manipulate our data set.

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math
import dateutil
from geopy.geocoders import Nominatim
import folium
from statsmodels.graphics.mosaicplot import mosaic
```

Next, we import the data that we want to explore which is stored in "survey.csv". It contains the responses to questions from a mental health survey.

In [2]:

```
df = pd.read_csv('survey.csv')
```

It is important for us to know the number of participants(rows) and features(columns) captured in the data set. Following this, we must know the data types of the features. This information could potentially guide the decisions we will make when cleaning the data.

In [3]:

```
df.shape
```

Out[3]:

```
(1259, 27)
```

In [4]:

```
df.dtypes
```

Out[4]:

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

Given that all columns, with the exception of age are object type, we need to have a closer look at what data is actually in the file. To do this, we will peek at the first few rows of the file.

In [5]:

```
df.head()
```

Out[5]:

	Timestamp	Age	Gender	Country	state	self_employed	family_history	treatment
0	2014-08-27 11:29:31	37	Female	United States	IL	NaN	No	Yes
1	2014-08-27 11:29:37	44	M	United States	IN	NaN	No	No
2	2014-08-27 11:29:44	32	Male	Canada	NaN	NaN	No	No
3	2014-08-27 11:29:46	31	Male	United Kingdom	NaN	NaN	Yes	Yes
4	2014-08-27 11:30:22	31	Male	United States	TX	NaN	No	No

5 rows × 27 columns

We notice that the other columns are textual and that there are missing values. The self_employed column head does not give any idea of what we might expect here. We will probe into it later. For the others, the object data type may not affect our analysis. We also need to look closer at the timestamp column. Due to the closeness of the values in the head, we might assume it is simply the date and time the form was submitted.

In [6]:

```
set(val[0:10] for val in df['Timestamp'])
```

Out[6]:

```
{'2014-08-27',
 '2014-08-28',
 '2014-08-29',
 '2014-08-30',
 '2014-08-31',
 '2014-09-01',
 '2014-09-02',
 '2014-09-03',
 '2014-09-04',
 '2014-09-05',
 '2014-09-08',
 '2014-09-09',
 '2014-09-11',
 '2014-09-12',
 '2014-09-13',
 '2014-09-14',
 '2014-09-20',
 '2014-09-23',
 '2014-09-26',
 '2014-09-30',
 '2014-10-02',
 '2014-10-05',
 '2014-10-09',
 '2014-11-05',
 '2014-11-06',
 '2014-11-16',
 '2014-12-01',
 '2014-12-15',
 '2015-01-03',
 '2015-02-21',
 '2015-02-22',
 '2015-02-24',
 '2015-02-26',
 '2015-04-02',
 '2015-04-04',
 '2015-04-06',
 '2015-04-11',
 '2015-04-23',
 '2015-05-05',
 '2015-05-06',
 '2015-05-07',
 '2015-06-25',
 '2015-07-22',
 '2015-07-27',
 '2015-08-17',
 '2015-08-20',
 '2015-08-25',
 '2015-09-12',
 '2015-09-26',
 '2015-11-07',
 '2015-11-30',
 '2016-02-01'}
```

This indicates that the data was collected between late August 2014 and early February 2016. This may not hold any relevance depending on what we are looking for. Let's change it's data type to datetime, just in case.

In [7]:

```
df['Timestamp'] = df['Timestamp'].apply(dateutil.parser.parse)
```

In [8]:

```
df.dtypes
```

Out[8]:

```
Timestamp                    datetime64[ns]
Age                                   int64
Gender                               object
Country                              object
state                                object
self_employed                        object
family_history                       object
treatment                            object
work_interfere                       object
no_employees                         object
remote_work                          object
tech_company                         object
benefits                             object
care_options                         object
wellness_program                     object
seek_help                            object
anonymity                            object
leave                                object
mental_health_consequence            object
phys_health_consequence              object
coworkers                            object
supervisor                           object
mental_health_interview              object
phys_health_interview                object
mental_vs_physical                   object
obs_consequence                      object
comments                             object
dtype: object
```

Now, let's see if any columns are totally empty.

In [9]:

```
all_nan = df.columns[df.isnull().all()]
list(all_nan)
```

Out[9]:

```
[]
```

There are no empty columns. Let's get a definitive list of columns with ANY missing values as well.

In [10]:

```
any_nan = df.columns[df.isnull().any()]
list(any_nan)
```

Out[10]:

```
['state', 'self_employed', 'work_interfere', 'comments']
```

State is expected to be missing some values as this only applies to responses from USA. Comments are also expected to be missing values as it is most likely optional free text. Let's see how many rows are missing values for self_employed and work_interfere. We would also like seeing the unique responses to these questions.

In [11]:

```
set(df['self_employed'])
```

Out[11]:

```
{nan, 'Yes', 'No'}
```

In [12]:

```
set(df['work_interfere'])
```

Out[12]:

```
{'Often', nan, 'Sometimes', 'Rarely', 'Never'}
```

In [13]:

```
miss = df[(df['work_interfere'].isnull())
          | (df['self_employed'].isnull())]
```

In [14]:

```
miss.shape
```

Out[14]:

```
(282, 27)
```

282 rows are missing values for the 2 fields. We should check if they overlap to see if this is related somehow.

In [15]:

```
overlap = miss[(miss['work_interfere'].isnull())
          & (miss['self_employed'].isnull())]
overlap.shape
```

Out[15]:

```
(0, 27)
```

Given that there is no overlap, let's examine the different occurences.

In [16]:

```
miss_set = miss[['work_interfere','self_employed']].drop_duplicates()
miss_set
```

Out[16]:

	work_interfere	self_employed
0	Often	NaN
1	Rarely	NaN
4	Never	NaN
5	Sometimes	NaN
19	NaN	Yes
26	NaN	No

Seems there is no obvious relationship between the 2. For now, we will not attempt to fill these but we can't delete the rows as they make up a large portion of our data and can be useful as the other fields are filled.As age is the only numerical column, let us examine some descriptive statistics about it.

In [17]:

```
df['Age'].describe()
```

Out[17]:

```
count    1.259000e+03
mean     7.942815e+07
std      2.818299e+09
min     -1.726000e+03
25%      2.700000e+01
50%      3.100000e+01
75%      3.600000e+01
max      1.000000e+11
Name: Age, dtype: float64
```

Immediately, we notice the minimum age is below 0 and the maximum is well over 1000. These values are most likely erroneous and could skew our findings. Let's check how many rows have these outliers in them.

In [18]:

```
err_ages = df.query('Age<0 or Age>150')
err_ages.shape
```

Out[18]:

```
(5, 27)
```

As it is only 5 rows out of 1259 (< 1% of records), we can remove them.

In [19]:

```
df = df.query('Age>=0 and Age<150')
df.shape
```

Out[19]:

```
(1254, 27)
```

Now we may re-examine the age stats.

In [20]:

```
df['Age'].describe()
```

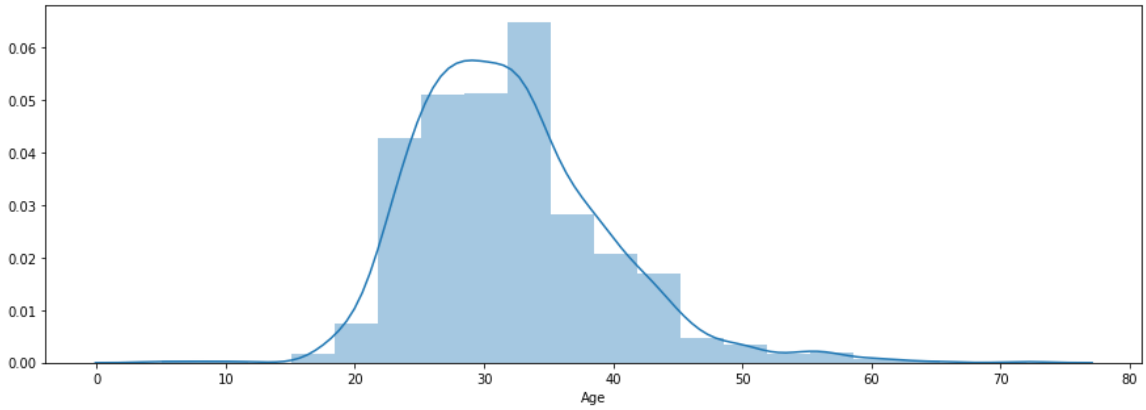
Out[20]:

```
count    1254.000000
mean       32.019139
std         7.375005
min         5.000000
25%        27.000000
50%        31.000000
75%        36.000000
max        72.000000
Name: Age, dtype: float64
```

Question 2. Now that all ages lie between 5 and 75, we can plot the age distribution.

In [21]:

```
plt.subplots(figsize=(15,5))
sns.distplot(df['Age'],bins=20)
plt.show()
```



This shows that age is normally distributed with the majority of participants being around the mean age, 32.

In [22]:

```
plt.subplots(figsize=(15,8))
sns.violinplot(x="tech_company", y="Age", data=df,
               bw=.1, scale="count", palette='pastel');
sns.swarmplot(x="tech_company", y="Age", hue="remote_work", data=df, alpha=1);
plt.show()
```

It seems that working in a tech company was related to whether or not employees work remotely. It seems like tech companies have a higher percentage of employees who work from home than non-tech companies.Question3. We should get an idea of how this data looks per country. First, we'll count how many entries there are for all countries.

In [23]:

```
country_agg = (df.groupby('Country')[['Age']].count())
country_agg['Country'] = country_agg.index
country_agg.rename(columns={'Age':'count'},inplace=True)
country_agg.describe()
```

Out[23]:

	count
count	47.000000
mean	26.680851
std	111.294577
min	1.000000
25%	1.000000
50%	3.000000
75%	7.000000
max	748.000000

The standard deviation is large and the mmedian value is much different to the mean value. This, coupled with a range of 747 seems to indicate a wide spread of values that are negatively skewed. Let's see how this looks on a map. We will plot the highest 10 on the map(this is because geopy throttles requests).

In [26]:

```
#please wait a minute and re-run this step if it fails.
#geopy sometimes thros an error for what seems to be too many requests.
geolocator = Nominatim()

country_top10 = country_agg.nlargest(10,'count')
countries  = pd.DataFrame({'Country':list(country_top10['Country'])})

names = []
lats = []
longs = []

print('Fetching countries, please wait...')
for c in list(countries['Country']):
   geo = geolocator.geocode(c)
   names.append(c)
   lats.append(geo.latitude)
   longs.append(geo.longitude)
   print('...')
print('Done!')

geo_df = pd.DataFrame({'Country':names,'longitude':longs,'latitude':lats})
df_coords = pd.merge(country_agg,geo_df,how='inner',on='Country')

map = folium.Map(location=[20,0], tiles='Mapbox Bright', zoom_start=1.5)
for i in range(0,10):
    row = df_coords.iloc[i]
    folium.Circle(
        location=[row['latitude'],row['longitude']],
        popup=row['Country'],
        radius=str(row['count']*1000),
        color='crimson',
        fill=True,
        fill_color='crimson'
    ).add_to(map)
map
```

```
Fetching countries, please wait...
...
...
...
...
...
...
...
...
...
...
Done!
```

Out[26]:

We can see based on the radius of the red circles that the most responses were recorded in the United States, United Kingdom and Canada respectively. If this data set is a representative sample of the world, it would seem to indicate that these countries, in the order listed, have the highest prevalence of mental health issues. We can clearly see that the radius of the United States is significantly larger than the others.Question 4 and 5 Now we will look at the role that family history has to play in individuals with mental health issues. First, we get an idea of the values of this column.

In [27]:

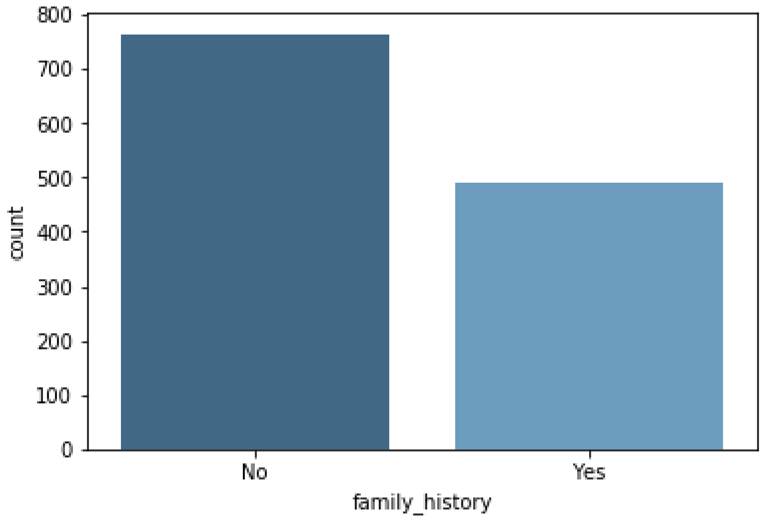
```
set(df['family_history'])
```

Out[27]:

```
{'No', 'Yes'}
```

In [28]:

```
sns.countplot(x='family_history', data=df, palette="Blues_d")
plt.show()
```



From this, we can see, that the majority of people with mental issues actually did not have history of it in their family.Questions 6 Let's look at the values of the columns of interest to see if they require massaging.

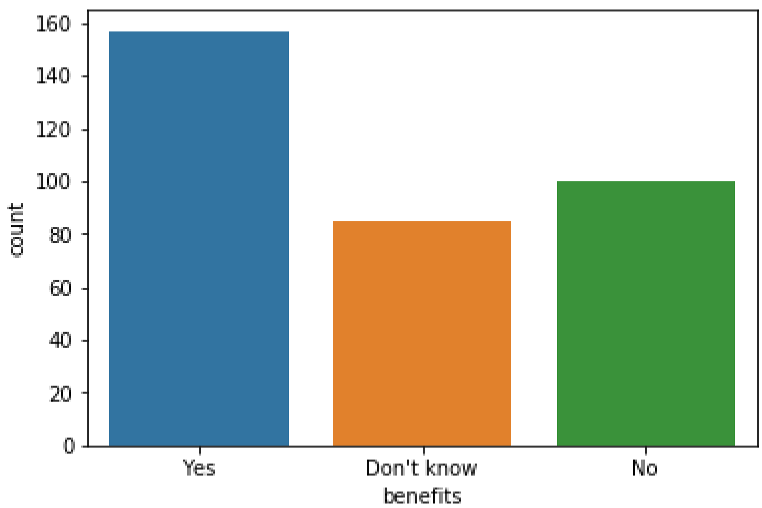
In [29]:

```
print(set(df['mental_vs_physical']))
print(set(df['benefits']))
print(set(df['no_employees']))
```

```
{'Yes', 'No', "Don't know"}
{'Yes', 'No', "Don't know"}
{'1-5', '26-100', 'More than 1000', '100-500', '6-25', '500-1000'}
```

In [30]:

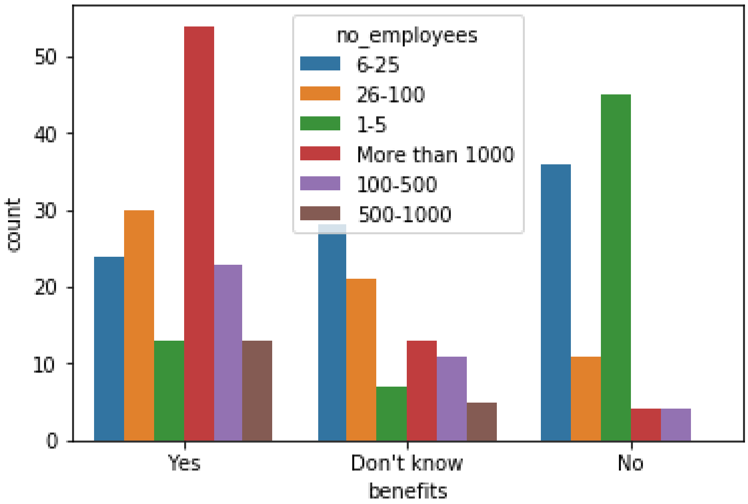
```
sns.countplot(x='benefits', data=df[df['mental_vs_physical']=='Yes'])
plt.show()
```



Most people with mental health issues say that their employer provides benefits to cater to them. We can probe further into this and see if there is any relation to the size of the companies offering these benefits.

In [31]:

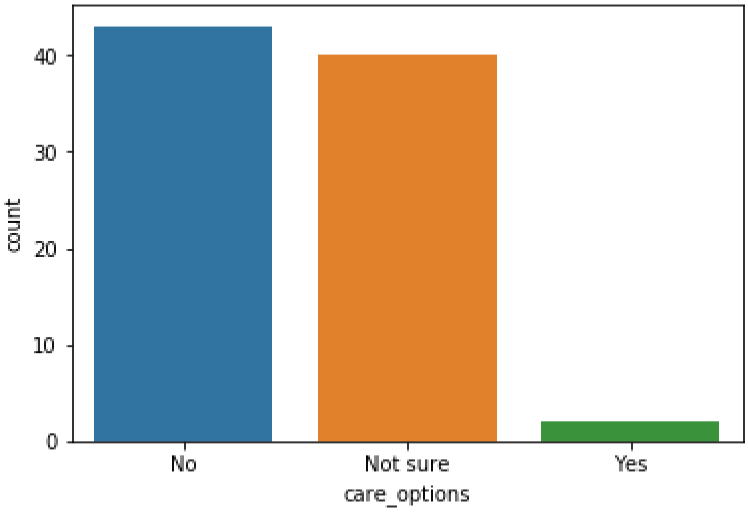
```
sns.countplot(x='benefits', data=df[df['mental_vs_physical']=='Yes'],hue='no_emp
loyees')
plt.show()
```



It seems like the larger companies with over 25 employees, have the highest presence and awareness of mental health issues benefits.

In [32]:

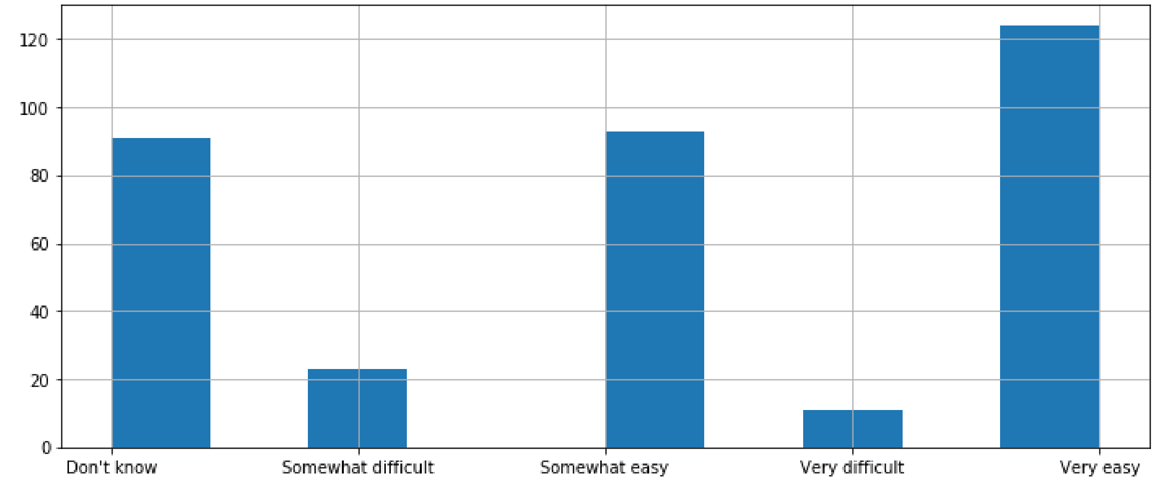
```
sns.countplot(x='care_options', data=df[
    (df['mental_vs_physical']=='Yes') &
    (df['benefits'] == "Don't know")] )
plt.show()
```



Those who don't know if their employer has benefits, neither know about care options. This could be an area to investigate for these companies as their employees may benefit from an education drive.Question 7 The histogram below shows that most people think it is either somewhat easy or very easy to take leave for mental health issues in companies that take it seriously. This could suggest that the companies support their employees through the challenges presented by their mental health issues.

In [33]:

```
(df[df['mental_vs_physical'] == "Yes"])['leave'].hist(figsize=(12,5))
plt.show()
```



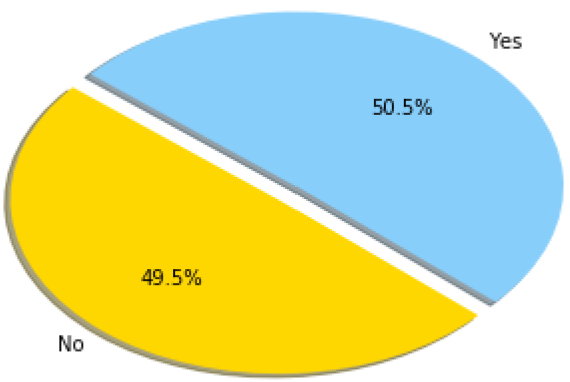
Question 8 and 9 There seems to be an almost even number of people receiving treatment vs those who are not.

In [34]:

```
treat_agg = (df.groupby('treatment')[['Age']].count())
treat_agg['treatment'] = treat_agg.index
treat_agg.rename(columns={'Age':'count'},inplace=True)

plt.pie(treat_agg['count'],explode=(0.1,0),labels=treat_agg['treatment'],
        colors=['gold','lightskyblue'],autopct='%1.1f%%', shadow=True, startangl
e=140)

plt.show()
```



Question 10

In [35]:

```
mosaic(df,['work_interfere'])
plt.show()
```

Most people in the survey say that their mental health affects their work atleast sometimes. The minority, what looks like around 20% of the people claim they are never affected.