PUBLIC HEALTH AWARENESS CAMPAIGN

PHASE-5

TEAM MEMBERS:

KEERTHI VARSHINI S - 2021115053

KONETI SASANK - 2021115054

KOWSHIK T - 2021115055

LEKHA S - 2021115056

SANTHOSH - 2021115311

PHASE 1:

1)PROJECT DEFINITION:

This project involves analyzing the data from public health awareness campaigns to measure their effectiveness in reaching the target audience and increasing awareness. The objective is to provide insights that evaluate the impact of the campaigns and inform future strategies. This project includes defining analysis objectives, collecting campaign data, designing relevant visualizations in IBM Cognos and using code for data analysis.

2)DESIGN THINKING:

ANALYSIS OBJECTIVES:

Before stepping into the data analysis, it's very essential to define clear objectives. In this phase we establish some important analysis that provide a clear direction for focusing our requirements of the project clearly.

- Defining analysis objectives
- Collecting campaign data(Measuring Audience reach, Awareness levels)
- Campaign impact

DATA COLLECTION:

To proceed with the analysis, the data must be essentially required. The dataset is accessible via the link,

https://www.kaggle.com/datasets/osmi/mental-health-in-tech-survey

The above data set is from a 2014 survey that measures attitudes towards mental health and frequency of mental disorders in the tech workplace. It also contains the timestamp, age factor, gender, country etc …. Also it compares the gender differences and similarities of the mental health between individuals.

VISUALIZATION STRATEGY:

Effective data visualization is key to conveying insights. In this Phase , We plan how to visualize the mean derivatives of the public health awareness survey using IBM Cognos. The visualization strategies include:

- Line charts and Time series graphs – To show the key metrics such as engagement levels and awareness rates
- Pie charts – Illustrating the distribution of various factors such as composition of the target audience in terms of age groups or gender
- Interactive dashboards – Will offer users the flexibility to explore campaign metrics, demographics and survey results.

INSIGHTS GENERATION:

One of the primary goal is to cover trends and patterns within the campaign data which examines data over time. This process includes,

- Segmenting the audience and campaign data is crucial for generating insights
- Comparing the performance of different campaigns or variations can yield insights
- Sentiment analysis involves analyzing text data from survey responses, to understand public sentiment towards the campaign.

PHASE 2 :

DATA COLLECTION AND PREPARATION :

Data Collection involves the systematic gathering of relevant information from various sources, which can include surveys, databases, sensors, social media, and more. It's essential to ensure that the data collected is accurate, complete, and representative of the phenomenon under study. In public health awareness campaigns, data collection may involve obtaining information about campaign reach, engagement, demographic profiles, health outcomes, and resource allocation. Data Preparation is the process of cleaning, organizing, and transforming raw data into a structured and usable format for analysis. This includes tasks such as

handling missing values, removing duplicates, standardizing formats, and aggregating data as needed. Proper data preparation is crucial for accurate analysis and insightful reporting, as it helps ensure that the data is reliable and ready for further processing.

DATA MODELLING :

Data modeling in this project involves creating a structured representation of the collected data. This representation typically defines how different data elements are related, organized, and hierarchically structured. Data models can help in understanding the underlying patterns and associations within the data, which is essential for making informed decisions and gaining insights. In the context of the project, data modeling is crucial for establishing relationships between campaign data, demographic information, and health outcomes, enabling a deeper understanding of the campaign's effectiveness and its impact on target audiences. It provides a foundation for generating meaningful reports, visualizations, and analysis that guide evidence-based decision-making to enhance the campaign's success.

REPORT DEVELOPMENT:

Report development step is the process of creating informative and visually appealing documents that present the project's findings and insights. These reports are designed to communicate the campaign's performance, outcomes, and recommendations to various stakeholders,

including policymakers, healthcare professionals, and the general public. Utilizing tools like IBM Cognos, report developers can transform data and analysis results into easy to-understand charts, graphs, and narratives. Effective report development is essential for conveying the campaign's impact, supporting data-driven decisions, and ensuring that key stakeholders have access to the information needed for improving public health initiatives

DATA ANALYSIS:

Data analysis is the process of examining and interpreting the collected data to extract meaningful insights and patterns. Through techniques such as statistical analysis, data mining, and visualization, analysts uncover trends and relationships within the data. In this context, data analysis helps evaluate the effectiveness of the campaign, identify target audience behaviours, and measure the impact on public health outcomes. It empowers decision-makers with the knowledge needed to optimize the campaign's strategies, refine messaging, and allocate resources more efficiently, ultimately contributing to the success of the public health initiative.

DERIVE INSIGHTS:

Deriving insights in a public health awareness campaign analysis project involves extracting valuable conclusions and actionable information from the data. By scrutinizing the data through various analytical techniques and tools, one can uncover patterns, trends, correlations, and areas

of significance. These insights shed light on the campaign's strengths and weaknesses, audience responses, and overall impact, enabling informed decision-making and strategies for improving the campaign's effectiveness. Essentially, deriving insights transforms raw data into knowledge that empowers stakeholders to refine and optimize the campaign for better outcomes in public health awareness.

MACHINE LEARNING MODELS:

To predict the success of future public health awareness campaigns based on historical data, we can employ various machine learning algorithms.

1. Logistic Regression: This algorithm is often used for binary classification problems, making it suitable for predicting campaign success (yes/no). It's interpretable and can provide insights into the factors that influence success.

2. Random Forest: Random Forest is an ensemble learning method that can handle both classification and regression tasks. It's robust and can capture complex relationships in our data.

3. Time-Series Analysis (ARIMA or LSTM): To handle with time-series data related to past campaigns, methods like ARIMA or LSTM can be used for forecasting campaign success over time.

4. Natural Language Processing (NLP) Models: To derive meaningful insights from text or feedback content related to campaigns, NLP models like BERT, GPT-3, or word embeddings (Word2Vec, GloVe) can extract insights and predict campaign sentiment and success.

PHASE 3:

Dataset Preprocessing and Cleaning :

this phase, the primary task to complete is to analyze the dataset and make sure the dataset is clean. In the cleaning process, we may have to remove null values and make sure that the dataset finally contains all non-null values

Import necessary libraries

```
#imports necessary libraries to do basic things on the dataset
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt

print('Successfully imported')
```

```
Successfully imported
```

Read Dataset

```
#Reading data
data = pd.read_csv('survey.csv')
data.head()
```

	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

Preprocessing and Cleaning dataset

```
#Check the dataset for 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
```

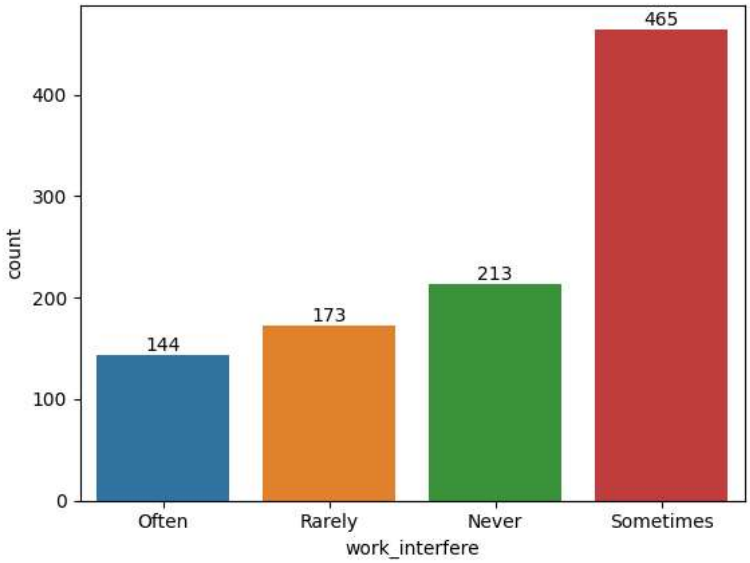
```
#Check our missing data from which columns and how many unique features they have.
frame = pd.concat([data.isnull().sum(), data.nunique(), data.dtypes], axis = 1, sort= False)
frame
```

	0	1	2
Timestamp	0	1246	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
coworkers	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]);
```

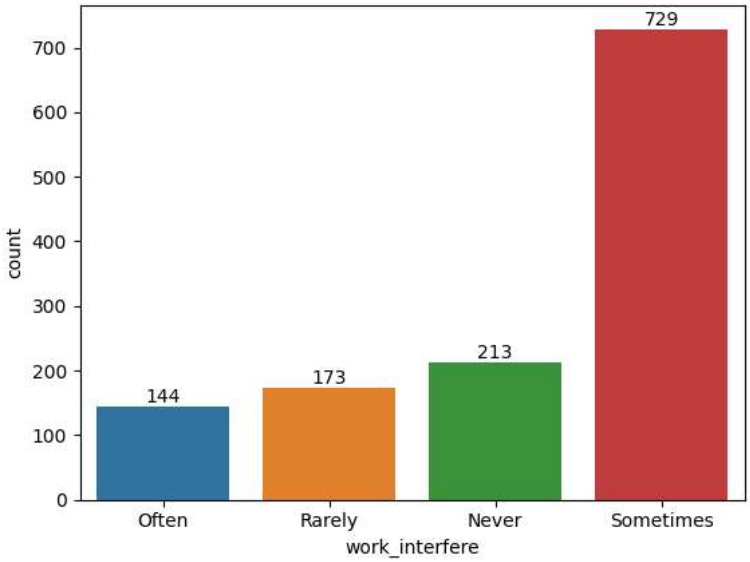



```
from sklearn.impute import SimpleImputer
import numpy as np
columns_to_drop = ['state', 'comments', 'Timestamp']
for column in columns_to_drop:
    if column in data.columns:
        data = data.drop(columns=[column])


# 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	...	anor
0	37	Female	United States	No	No	Yes	Often	6-25	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	6-25	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())
```

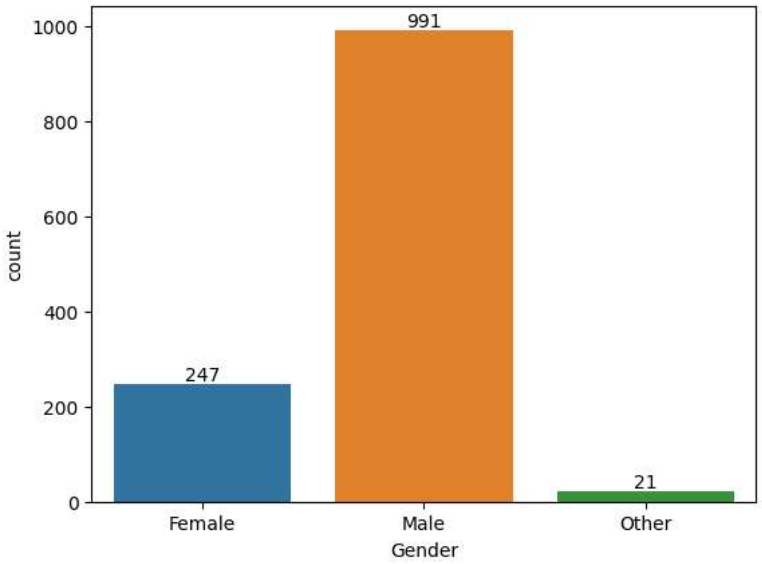
```
['Female' 'M' 'Male' 'male' 'female' 'm' 'Male-ish' 'maile' 'Trans-female'
 'Cis Female' 'F' 'something kinda male?' 'Cis Male' 'Woman' 'f' 'Mal'
 'Male (CIS)' 'queer/she/they' 'non-binary' 'Femake' 'woman' 'Make' 'Nah'
 'All' 'Enby' 'fluid' 'Genderqueer' 'Female ' 'Androgyne' 'Agender'
 'cis-female/femme' '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]);
```



CHECKING AFTER CLEANING THE DATASET

```
#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('There 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())
```

```
There is no duplicated data:
--------------------------------------------------
0
```

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

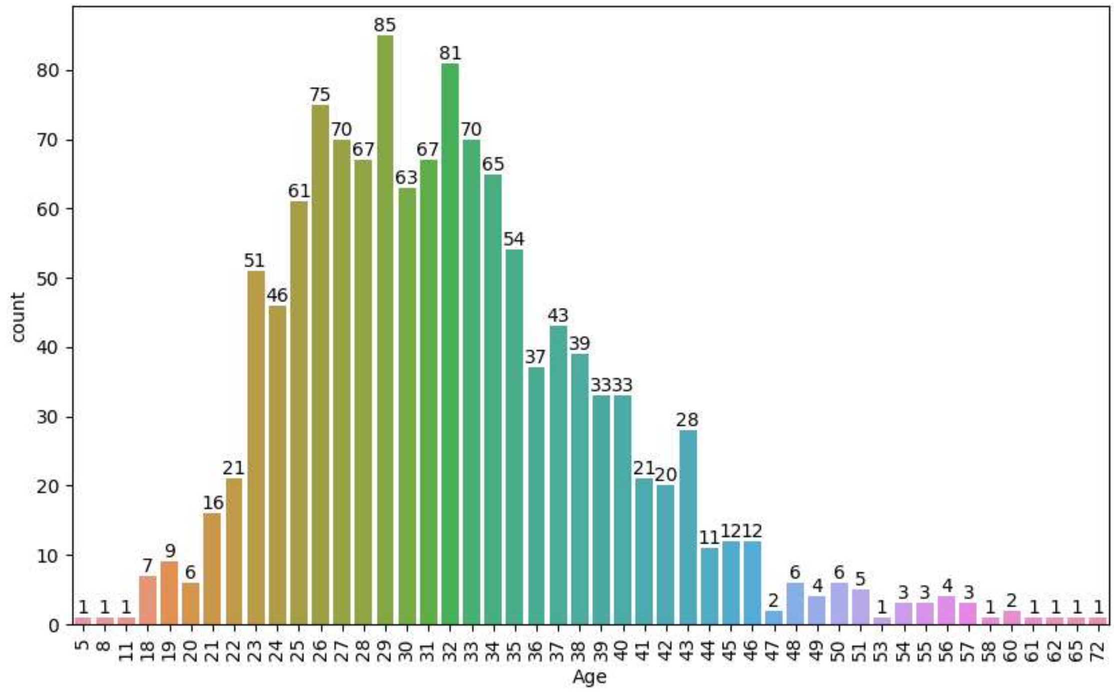
```
array([         37,          44,          32,          31,          33,
                35,          39,          42,          23,          29,
                36,          27,          46,          41,          34,
                30,          40,          38,          50,          24,
                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())
```

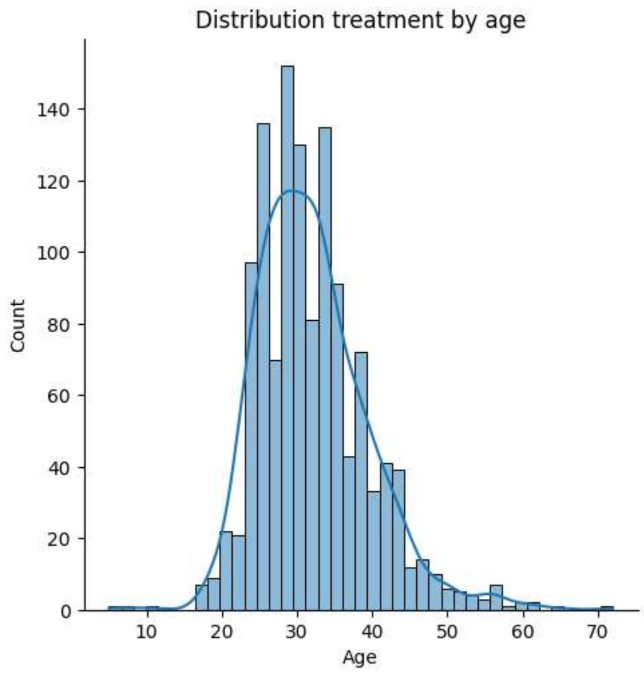
```
[37 44 32 31 33 35 39 42 23 29 36 27 46 41 34 30 40 38 50 24 18 28 26 22
 19 25 45 21 43 56 60 54 55 48 20 57 58 47 62 51 65 49  5 53 61  8 11 72]
```

```
#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);
```

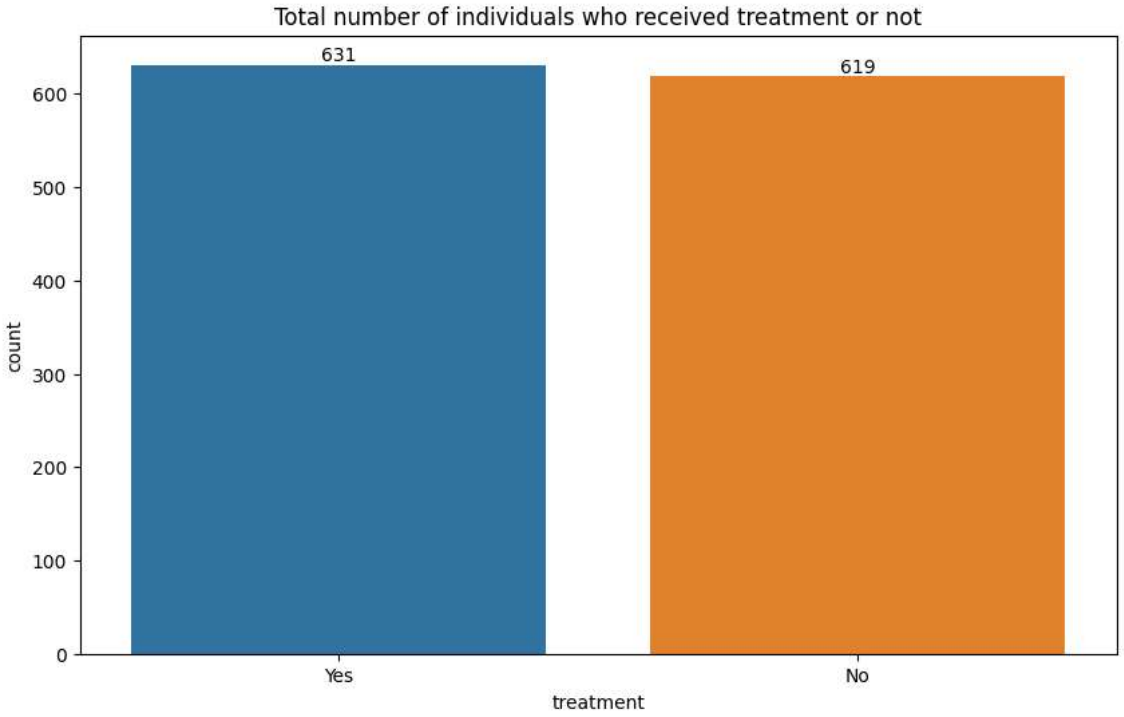



```
#In this plot moreover on Age distribution we can see treatment distribution by age
plt.figure(figsize=(10, 6));
sns.displot(data['Age'], kde = 'treatment');
plt.title('Distribution treatment by age');
```

```
<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
```

PHASE4:

Objectives:

In this phase defines start to building the Project by loading and preprocessing the dataset and perform different analysis and visualization using IBM Cognos.

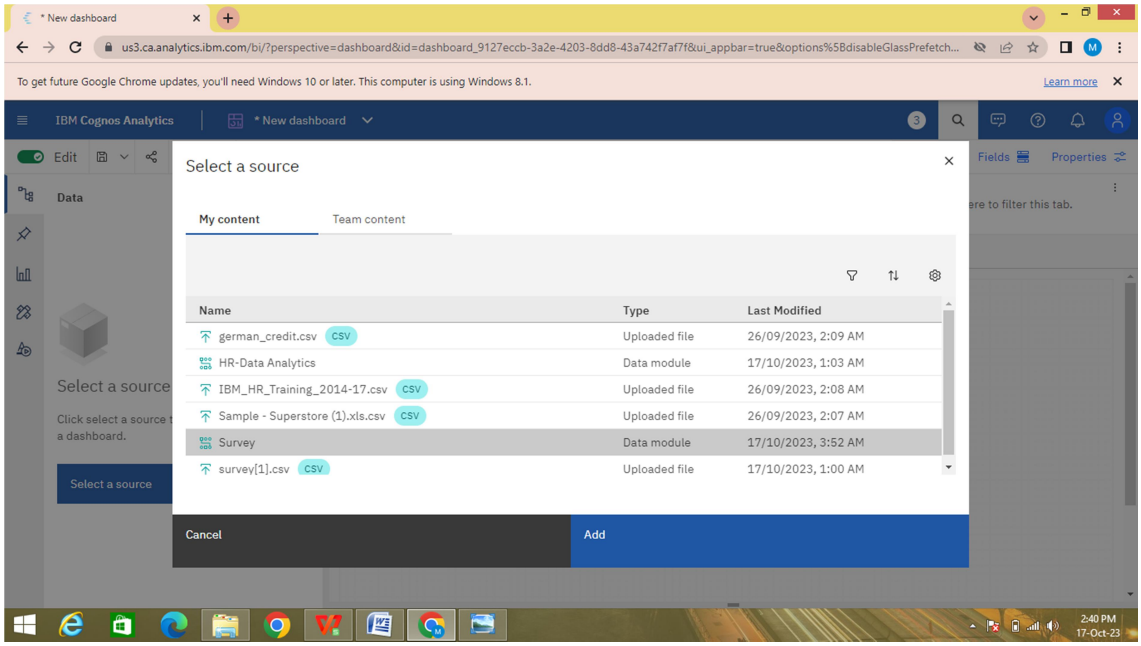
Visualization in IBM Cognos

Steps Involved in data loading on IBM cognos.

Step 1:

1. Login to your IBM cognos
2. Click more menu from the left side
3. Select new tab
4. Click Dashboard tap
5. Select Template for your dashboard
6. Now Dashboard is created and select your data source

7. Select the data source



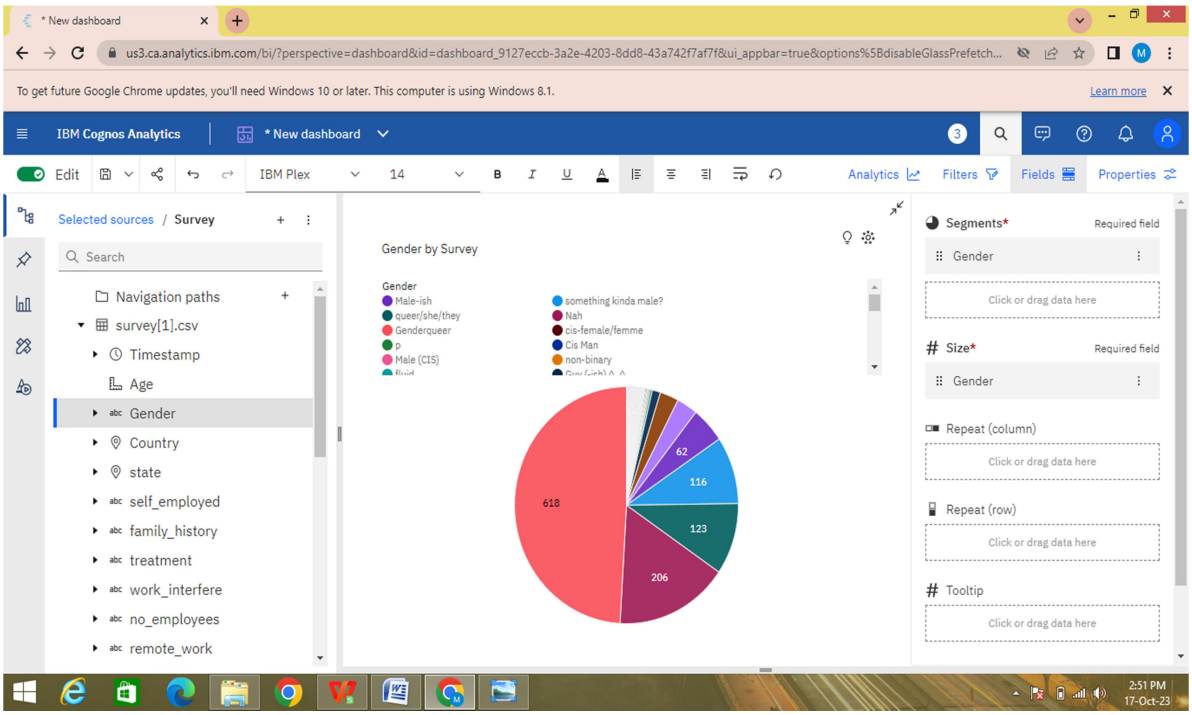
Visualization

After creating the dashboard, the next step is to visualize the data

In IBM Cognos

1. Goes to the Corresponding Dashboard

2. select the visualizations tab in the left side of title bar



In the above screen shot displays the Pie chart in Gender by survey.

After performing these activities a comprehensive document will be created to demonstrate the ability to Communicate and share finding.

```
import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import matplotlib.pyplot as plt
import seaborn as sns

from scipy import stats

from scipy.stats import randint


# prep

from sklearn.model_selection import train_test_split
from sklearn import preprocessing

from sklearn.datasets import make_classification

from sklearn.preprocessing import binarize, LabelEncoder, MinMaxScaler

# models

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier,

ExtraTreesClassifier


# Validation libraries

from sklearn import metrics

from sklearn.metrics import accuracy_score, mean_squared_error,
precision_recall_curve

from sklearn.model_selection import cross_val_score


#Neural Network

from sklearn.neural_network import MLPClassifier


#Bagging

from sklearn.ensemble import BaggingClassifier, AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier

#Naive bayes

from sklearn.naive_bayes import GaussianNB


#Stacking

from mlxtend.classifier import StackingClassifier
```

```
# Any results you write to the current directory are saved as output.

#reading in CSV's from a file path
train_df = pd.read_csv("C:\\Users\Manikandan\Downloads\survey.csv")

#Pandas: whats the data row count?
print(train_df.shape)

#Pandas: whats the distribution of the data?
print(train_df.describe())
```

```
#Pandas: What types of data do i have?

print(train_df.info())

(1259, 27)

                Age
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
<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
None

pip install mlxtend
```

```
Defaulting to user installation because normal site-packages is not
writeable

Collecting mlxtend

  Obtaining dependency information for mlxtend from
https://files.pythonhosted.org/packages/73/da/d5d77a9a7a135c948dbf8d3b
873655b105a152d69e590150c83d23c3d070/mlxtend-0.23.0-py3-none-
any.whl.metadata

  Downloading mlxtend-0.23.0-py3-none-any.whl.metadata (7.3 kB)
Requirement already satisfied: scipy>=1.2.1 in c:\programdata\
anaconda3\lib\site-packages (from mlxtend) (1.11.1)

Requirement already satisfied: numpy>=1.16.2 in c:\programdata\
anaconda3\lib\site-packages (from mlxtend) (1.24.3)

Requirement already satisfied: pandas>=0.24.2 in c:\programdata\
anaconda3\lib\site-packages (from mlxtend) (2.0.3)

Requirement already satisfied: scikit-learn>=1.0.2 in c:\programdata\
anaconda3\lib\site-packages (from mlxtend) (1.3.0)

Requirement already satisfied: matplotlib>=3.0.0 in c:\programdata\
anaconda3\lib\site-packages (from mlxtend) (3.7.2)

Requirement already satisfied: joblib>=0.13.2 in c:\users\harsh\
appdata\roaming\python\python311\site-packages (from mlxtend) (1.1.1)
Requirement already satisfied: contourpy>=1.0.1 in c:\programdata\
anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (1.0.5)
Requirement already satisfied: cycler>=0.10 in c:\programdata\
anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\programdata\
anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\programdata\
anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\programdata\
anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (23.1)
Requirement already satisfied: pillow>=6.2.0 in c:\programdata\
anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (9.4.0)
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in c:\
programdata\anaconda3\lib\site-packages (from matplotlib>=3.0.0-

>mlxtend) (3.0.9)

Requirement already satisfied: python-dateutil>=2.7 in c:\programdata\
anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\programdata\
anaconda3\lib\site-packages (from pandas>=0.24.2->mlxtend)
(2023.3.post1)

Requirement already satisfied: tzdata>=2022.1 in c:\programdata\
anaconda3\lib\site-packages (from pandas>=0.24.2->mlxtend) (2023.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\programdata\
anaconda3\lib\site-packages (from scikit-learn>=1.0.2->mlxtend)   --.
(2.2.0)

Requirement already satisfied: six>=1.5 in c:\programdata\anaconda3\
lib\site-packages (from python-dateutil>=2.7->matplotlib>=3.0.0-
```

```
#dealing with missing data

#Let's get rid of the variables "Timestamp",“comments”, “state” just
to make our lives easier.
train_df = train_df.drop(['comments'], axis= 1)
train_df = train_df.drop(['state'], axis= 1)
train_df = train_df.drop(['Timestamp'], axis= 1)

train_df.isnull().sum().max() #just checking that there's no missing
data missing...

train_df.head(5)
```

```
   Age  Gender          Country self_employed family_history treatment
\
0   37  Female    United States           NaN              No       Yes

1   44       M    United States           NaN              No        No

2   32    Male           Canada           NaN              No        No

3   31    Male   United Kingdom           NaN             Yes       Yes

4   31    Male    United States           NaN              No        No


  work_interfere    no_employees remote_work tech_company ...
anonymity \

0          Often             6-25          No          Yes  ...
Yes
1         Rarely  More than 1000          No           No  ...  Don't
know
2         Rarely             6-25          No          Yes  ...  Don't
know
3          Often           26-100          No          Yes  ...
```

```
No
4              Never           100-500          Yes              Yes  ...  Don't
know


                              leave mental_health_consequence
phys_health_consequence \
0          Somewhat easy                                   No
No
1          Don't know                                Maybe
No
2          Somewhat difficult                              No
No
3          Somewhat difficult                             Yes
No
4          Don't know                                      No
No

          coworkers supervisor mental_health_interview
phys_health_interview \
0 Some of them         Yes                          No
Maybe
1              No          No                       No
No
2              Yes         Yes                      Yes
No
3  Some of them            No                       Maybe
Maybe
4  Some of themYes         Yes                      Yes
No


  mental_vs_physical obs_consequence
0                Yes                 No
1         Don't know                 No
                                     No
2                 No                Yes
                  No                 No

[5 rows x 24 columns]
```

```
# Assign default values for each data type

defaultInt = 0
defaultString = 'NaN'
defaultFloat = 0.0

# Create lists by data tpe

intFeatures = ['Age']

stringFeatures = ['Gender', 'Country', 'self_employed',
'family_history', 'treatment', 'work_interfere',

                  'no_employees', 'remote_work', 'tech_company',
'anonymity', 'leave', 'mental_health_consequence',
```

```
                  'phys_health_consequence', 'coworkers', 'supervisor',
'mental_health_interview', 'phys_health_interview',
                  'mental_vs_physical', 'obs_consequence', 'benefits',
'care_options', 'wellness_program',
                  'seek_help']
floatFeatures = []

# Clean the NaN's

for feature in train_df:
    if feature in intFeatures:
        train_df[feature] = train_df[feature].fillna(defaultInt)
    elif feature in stringFeatures:
        train_df[feature] = train_df[feature].fillna(defaultString)
    elif feature in floatFeatures:
        train_df[feature] = train_df[feature].fillna(defaultFloat)
    else:
        print('Error: Feature %s not recognized.' % feature)
train_df.head(5)
```

```
   Age  Gender         Country self_employed family_history treatment
\
0   37  Female   United States           NaN              No       Yes

1   44       M   United States           NaN              No        No

2   32    Male          Canada           NaN              No        No

3   31    Male  United Kingdom           NaN             Yes       Yes

4   31    Male   United States           NaN              No        No


  work_interfere    no_employees remote_work tech_company ...
anonymity \
0          Often             6-25          No          Yes  ...
Yes
1         Rarely  More than 1000          No           No  ...  Don't
know
2         Rarely             6-25          No          Yes  ...  Don't
know
3          Often           26-100          No          Yes  ...
No
4          Never          100-500         Yes          Yes  ...  Don't
know
```

```
No
2
  Somewhat difficult                          No
No
3
Yes Somewhat difficult                        Yes
4
No
           Don't know                         No


      coworkers supervisor mental_health_interview
phys_health_interview  \
0  Some of them         Yes                       No
Maybe
1            No          No                       No
No
2           Yes         Yes                      Yes
Yes
3  Some of them          No                   Maybe
Maybe
4  Some of them         Yes                      Yes
Yes
```

0 Yes No
Don't know No
1 No Yes
2 No No

```
[5 rows x 24 columns]
```

```
#clean 'Gender'

#Slower case all columm's elements
gender = train_df['Gender'].str.lower()
#print(gender)

#Select unique elements

gender = train_df['Gender'].unique()
```

```
#Made gender groups

male_str = ["male", "m", "male-ish", "maile", "mal", "male (cis)",

"make", "male ", "man","msle", "mail", "malr","cis man", "Cis Male",
"cis male"]

trans_str = ["trans-female", "something kinda male?",
"queer/she/they", "non-binary","nah", "all", "enby", "fluid",
"genderqueer", "androgyne", "agender", "male leaning androgynous",
"guy (-ish) ^_^", "trans woman", "neuter", "female (trans)", "queer",
"ostensibly male, unsure what that really means"]

female_str = ["cis female", "f", "female", "woman",  "femake", "female
","cis-female/femme", "female (cis)", "femail"]
```

```
for (row, col) in train_df.iterrows():


    if str.lower(col.Gender) in male_str:
        train_df['Gender'].replace(to_replace=col.Gender,
value='male', inplace=True)


    if str.lower(col.Gender) in female_str:
        train_df['Gender'].replace(to_replace=col.Gender,
value='female', inplace=True)


    if str.lower(col.Gender) in trans_str:
        train_df['Gender'].replace(to_replace=col.Gender,
value='trans', inplace=True)


#Get rid of bullshit
stk_list = ['A little about you', 'p']
train_df = train_df[~train_df['Gender'].isin(stk_list)]
print(train_df['Gender'].unique())
```

```
['female' 'male' 'trans']
```

```
#complete missing age with mean
train_df['Age'].fillna(train_df['Age'].median(), inplace = True)


# Fill with media() values < 18 and > 120
s = pd.Series(train_df['Age'])
s[s<18] = train_df['Age'].median()
train_df['Age'] = s
s = pd.Series(train_df['Age'])
s[s>120] = train_df['Age'].median()
train_df['Age'] = s
#Ranges of Age
train_df['age_range'] = pd.cut(train_df['Age'], [0,20,30,65,100],
labels=["0-20", "21-30", "31-65", "66-100"], include_lowest=True)
#There are only 0.014% of self employed so let's change NaN to NOT
self_employed
```

```
#Replace "NaN" string from defaultString
train_df['self_employed'] =
train_df['self_employed'].replace([defaultString], 'No')
print(train_df['self_employed'].unique())
```

```
['No' 'Yes']
```

```
#There are only 0.20% of self work_interfere so let's change NaN to
"Don't know
```

```
#Replace "NaN" string from defaultString
```

```
train_df['work_interfere']= train_df['work_interfere'].replace
([defaultString], 'Don\'t know' )
print(train_df['work_interfere'].unique())
```

```
['Often' 'Rarely' 'Never' 'Sometimes' "Don't know"]
```

```
#Encoding data

labelDict = {}

for feature in train_df:

    le = preprocessing.LabelEncoder()
    le.fit(train_df[feature])
    le_name_mapping = dict(zip(le.classes_,

le.transform(le.classes_)))

    train_df[feature] = le.transform(train_df[feature])

    # Get labels

labelKey = 'label_' + feature labelValue = [*le_name_mapping]
labelDict[labelKey] =labelValue
```

```
for key, value in labelDict.items():
    print(key, value)
```

```
#Get rid of 'Country'

train_df = train_df.drop(['Country'], axis= 1)
train_df.head()

label_Age [18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32,

33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49,

50, 51, 53, 54, 55, 56, 57, 58, 60, 61, 62, 65, 72]

label_Gender ['female', 'male', 'trans']

label_Country ['Australia', 'Austria', 'Belgium', 'Bosnia and
Herzegovina', 'Brazil', 'Bulgaria', 'Canada', 'China', 'Colombia',
'Costa Rica', 'Croatia', 'Czech Republic', 'Denmark', 'Finland',
'France', 'Georgia', 'Germany', 'Greece', 'Hungary', 'India',

'Ireland', 'Israel', 'Italy', 'Japan', 'Latvia', 'Mexico', 'Moldova',
'Netherlands', 'New Zealand', 'Nigeria', 'Norway', 'Philippines',
'Poland', 'Portugal', 'Romania', 'Russia', 'Singapore', 'Slovenia',
'South Africa', 'Spain', 'Sweden', 'Switzerland', 'Thailand', 'United
Kingdom', 'United States', 'Uruguay', 'Zimbabwe']

label_self_employed ['No', 'Yes']
label_family_history ['No', 'Yes']
label_treatment ['No', 'Yes']

label_work_interfere ["Don't know", 'Never', 'Often', 'Rarely',
'Sometimes']

label_no_employees ['1-5', '100-500', '26-100', '500-1000', '6-25',

'More than 1000']
label_remote_work ['No', 'Yes']
label_tech_company ['No', 'Yes']

label_benefits ["Don't know", 'No', 'Yes']
label_care_options ['No', 'Not sure', 'Yes']
label_wellness_program ["Don't know", 'No', 'Yes']
```

```
label_seek_help ["Don't know", 'No', 'Yes']
label_anonymity ["Don't know", 'No', 'Yes']

label_leave ["Don't know", 'Somewhat difficult', 'Somewhat easy',
'Very difficult', 'Very easy']

label_mental_health_consequence ['Maybe', 'No', 'Yes']
label_phys_health_consequence ['Maybe', 'No', 'Yes']
label_coworkers ['No', 'Some of them', 'Yes']
label_supervisor ['No', 'Some of them', 'Yes']
label_mental_health_interview ['Maybe', 'No', 'Yes']
label_phys_health_interview ['Maybe', 'No', 'Yes']
label_mental_vs_physical ["Don't know", 'No', 'Yes']
label_obs_consequence ['No', 'Yes']

label_age_range ['0-20', '21-30', '31-65', '66-100']


   Age  Gender  self_employed  family_history  treatment
work_interfere  \
0   19       0              0               0          1
2
1   26       1              0               0          0
3
2   14       1              0               0          0
3
3   13       1              0               1          1
2
4   13       1              0               0          0
1


   no_employees  remote_work  tech_company  benefits  ...  leave  \
0             4            0             1         2  ...      2
1             5            0             0         0  ...      0
2             4            0             1         1  ...      1
3             2            0             1         1  ...      1
4             1            1             1         2  ...      0


   mental_health_consequence phys_health_consequence coworkers
supervisor \
0                          1                         1          1
2
1                          0                         1          0
0
2                          1                         1          2
2
3                          2                         2          1
0
4                          1                         1          1
2
```

mental_health_interview phys_health_interview mental_vs_physical

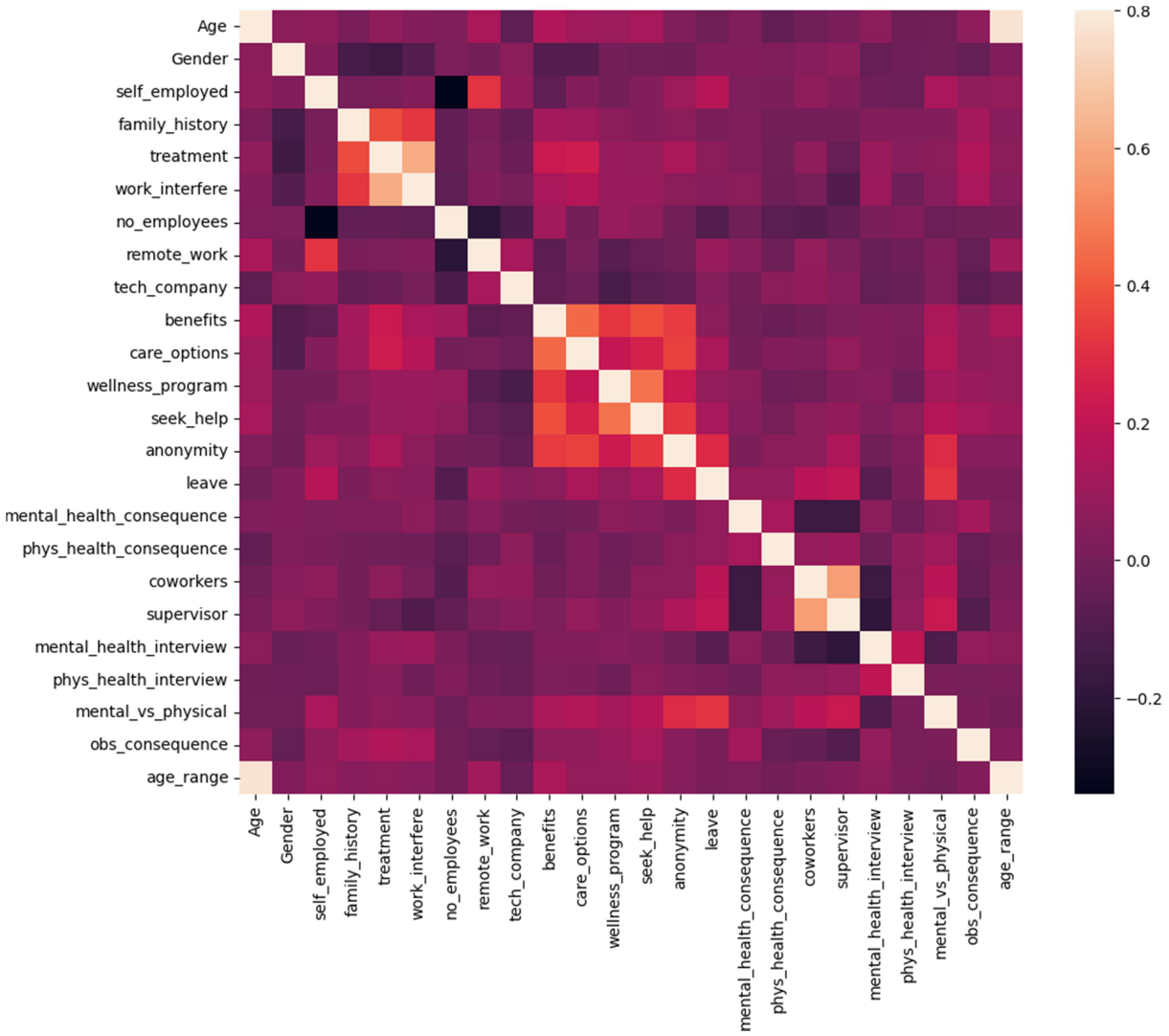
```
1                          1                          1                          0

2                          2                          2                          1

3                          0                          0                          1

4                          2                          2                          0


   obs_consequence  age_range
0                0          2
1                0          2
2                0          2
3                1          2
4                0          2

[5 rows x 24 columns]
```

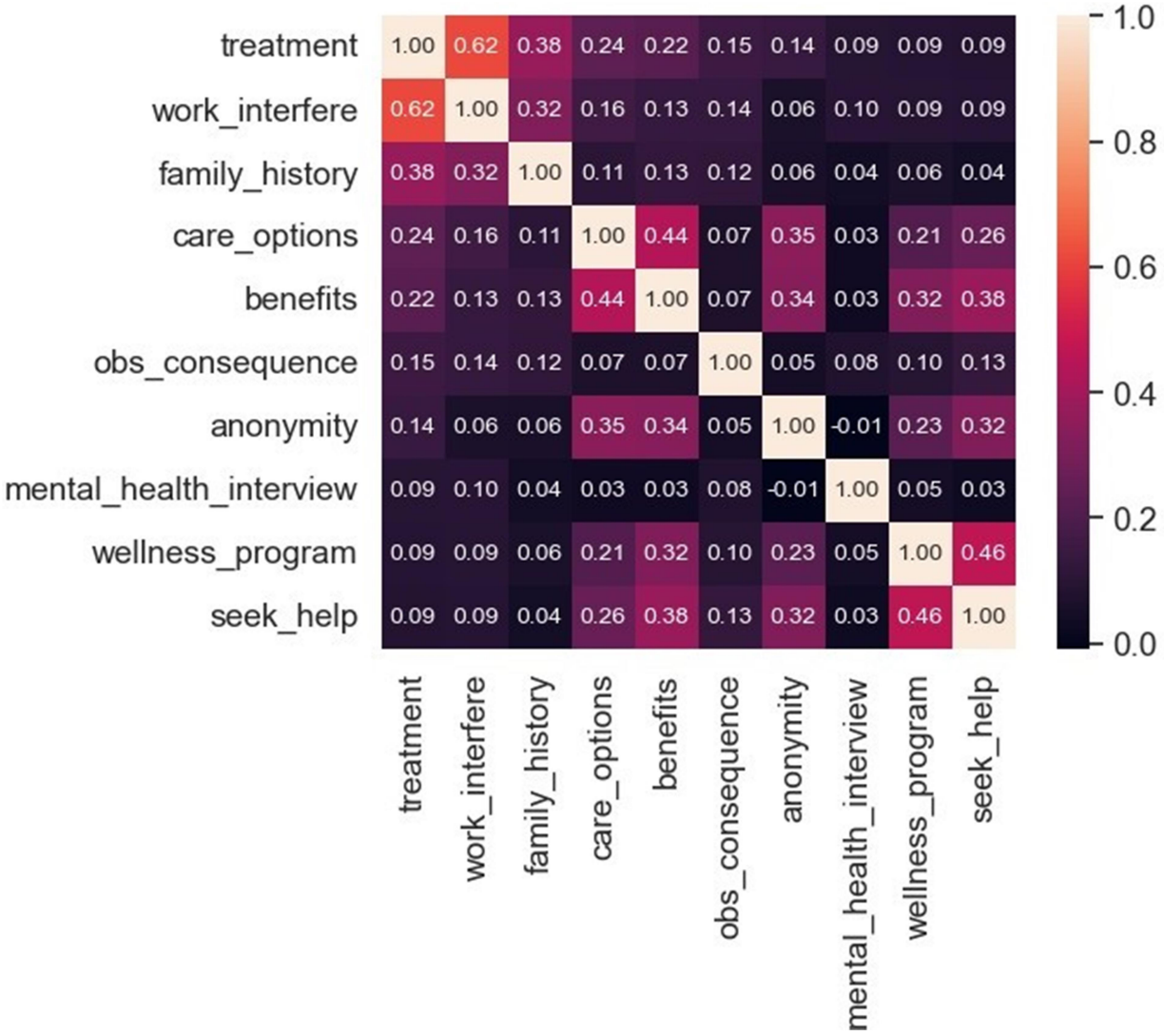
```
#missing data

total = train_df.isnull().sum().sort_values(ascending=False)
percent =

(train_df.isnull().sum()/train_df.isnull().count()).sort_values(ascend
ing=False)

missing_data = pd.concat([total, percent], axis=1, keys=['Total',
'Percent'])

missing_data.head(20)
print(missing_data)
```

```
                           Total  Percent
Age                            0      0.0
Gender                         0      0.0
obs_consequence                0      0.0
mental_vs_physical             0      0.0
phys_health_interview          0      0.0
mental_health_interview        0      0.0
supervisor                     0      0.0
coworkers                      0      0.0
phys_health_consequence        0      0.0
mental_health_consequence      0      0.0
leave                          0      0.0
anonymity                      0      0.0
seek_help                      0      0.0
wellness_program               0      0.0
care_options                   0      0.0
benefits                       0      0.0
tech_company                   0      0.0
remote_work                    0      0.0
no_employees                   0      0.0
work_interfere                 0      0.0
```

```
treatment                       0       0.0
family_history                  0       0.0
self_employed                   0       0.0
age_range                       0       0.0
```

```
#correlation matrix

corrmat = train_df.corr()

f, ax = plt.subplots(figsize=(12, 9)) sns.heatmap(corrmat, vmax=.8,
square=True);plt.show()

#treatment correlation matrix

k = 10 #number of variables for heatmap

cols = corrmat.nlargest(k, 'treatment')['treatment'].indexcm =
np.corrcoef(train_df[cols].values.T) sns.set(font_scale=1.25)
```


Age
Gender
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
age_range
0.8
0.6
0.4
0.2
0.0
−0.2




```
# Distribiution and density by Age plt.figure(figsize=(12,8))
sns.distplot(train_df["Age"], bins=24) plt.title("Distribuition and density by Age")
plt.xlabel("Age")
```

```
C:\Users\harsh\AppData\Local\Temp\ipykernel_21212\2516591640.py:3:UserWarning:
`distplot` is a deprecated function and will be removed in seabornv0.14.0.
Please adapt your code to use either `displot` (a figure-levelfunction with
similar flexibility) or `histplot` (an axes-level function forhistograms).
```

For a guide to updating your code to use the new functions, please see https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

```
    sns.distplot(train_df["Age"], bins=24)Text(0.5, 0,
'Age')
```