PUBLIC HEALTH AWARENESS

CODING PART USING PYTHON :

```
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
```

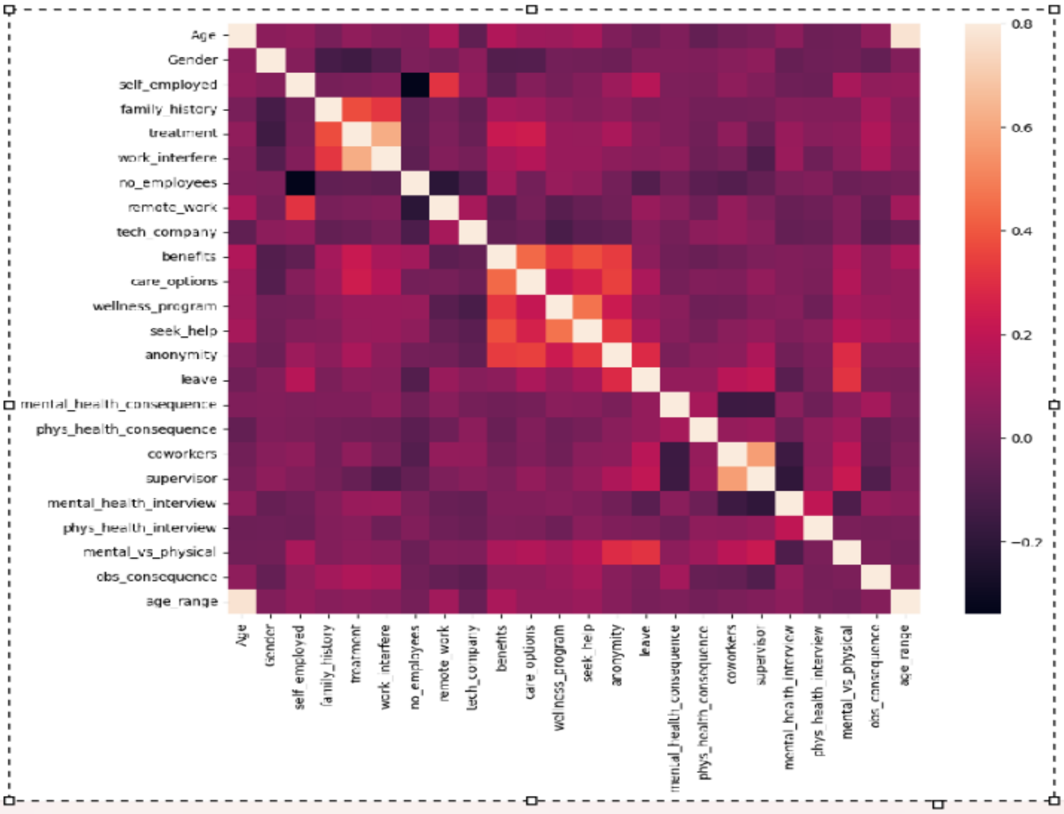
```
#Bagging
from sklearn.ensemble import BaggingClassifier, AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier

#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\harsh\OneDrive\Desktop\Trainings 2023-2024\Firebird\DataSets\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())
pip install mlxtend
#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)
# 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)
#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)
```

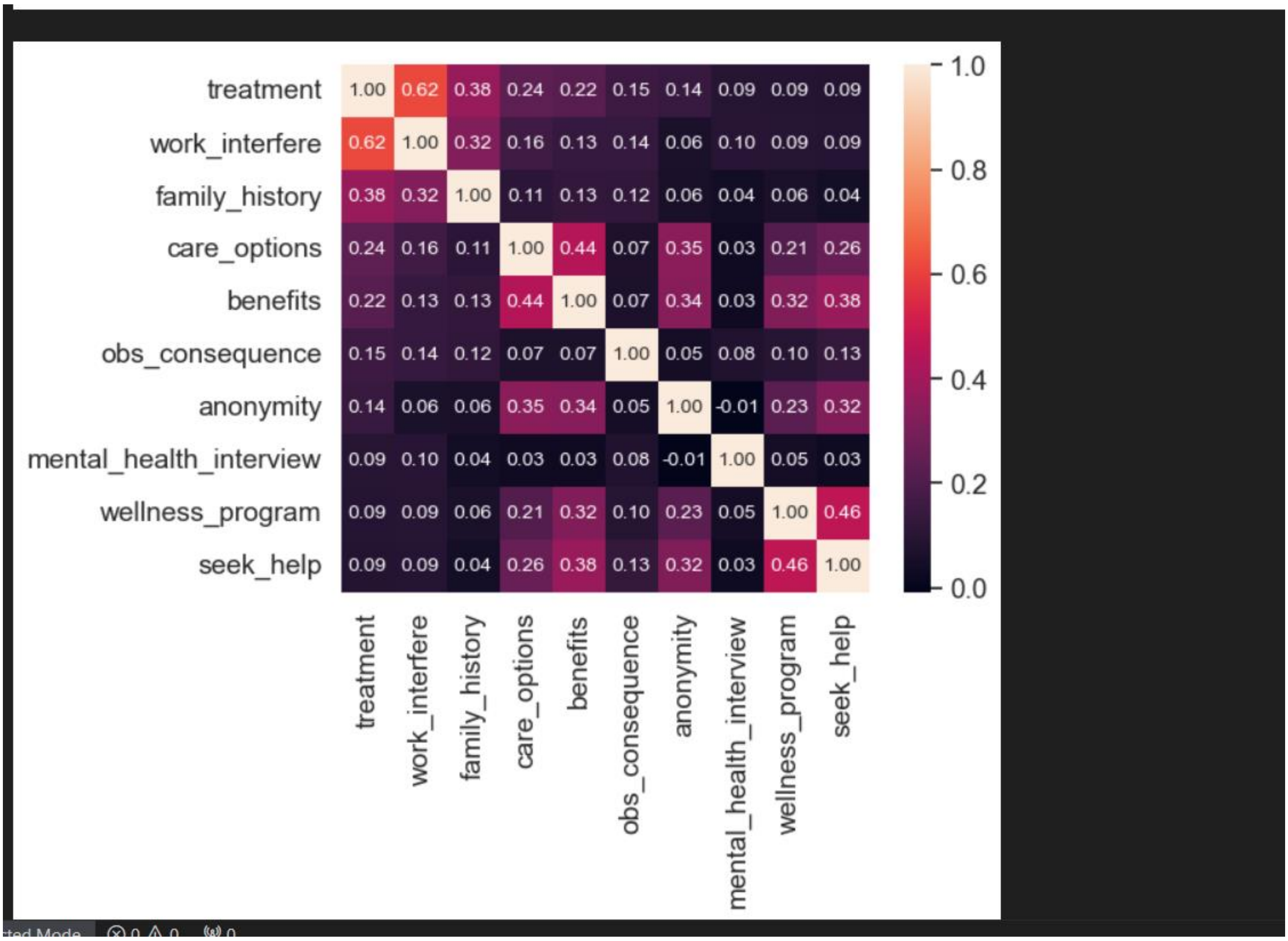
```
#Get rid of bullshit
stk_list = ['A little about you', 'p']
train_df = train_df[~train_df['Gender'].isin(stk_list)]

print(train_df['Gender'].unique())
#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())
#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())
#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()
#missing data
total = train_df.isnull().sum().sort_values(ascending=False)
percent = (train_df.isnull().sum()/train_df.isnull().count()).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data.head(20)
print(missing_data)
#correlation matrix
corrmat = train_df.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat, vmax=.8, square=True);
plt.show()
```

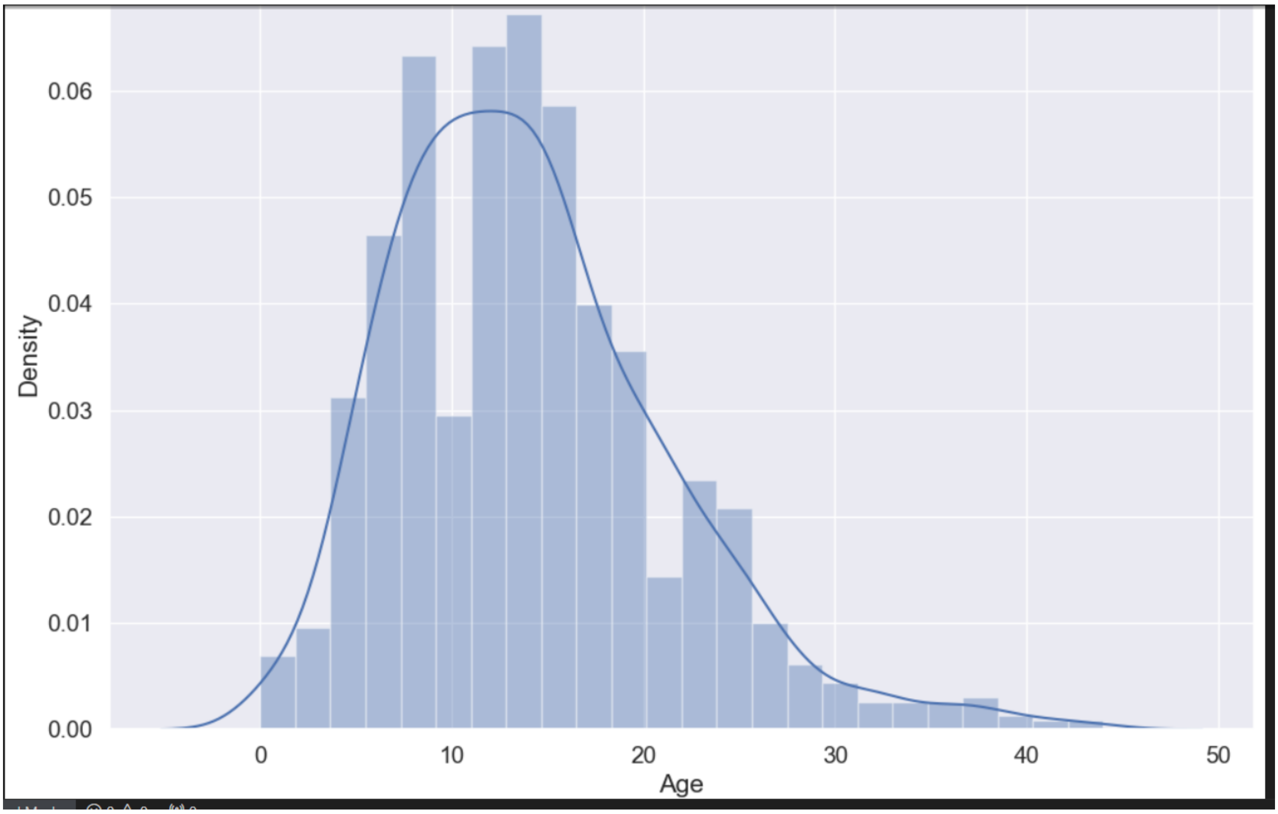
```
#treatment correlation matrix
k = 10 #number of variables for heatmap
cols = corrmat.nlargest(k, 'treatment')['treatment'].index
cm = np.corrcoef(train_df[cols].values.T)
sns.set(font_scale=1.25)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size': 10}, yticklabels=cols.values, xticklabels=cols.values)
plt.show()
# 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")
# Let see how many people has been treated
plt.figure(figsize=(12,8))
labels = labelDict['label_Gender']
g = sns.countplot(x="treatment", data=train_df)
g.set_xticklabels(labels)

plt.title('Total Distribuition by treated or not')
```

OUTPUT :

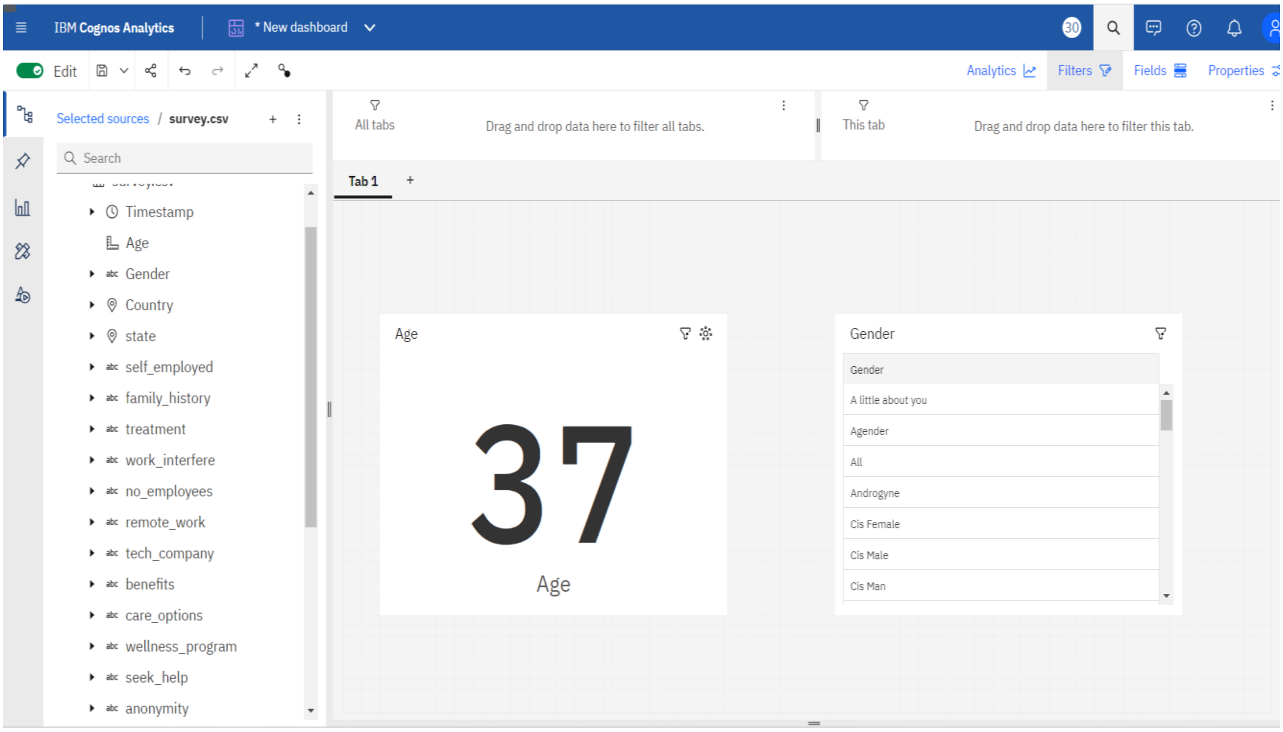


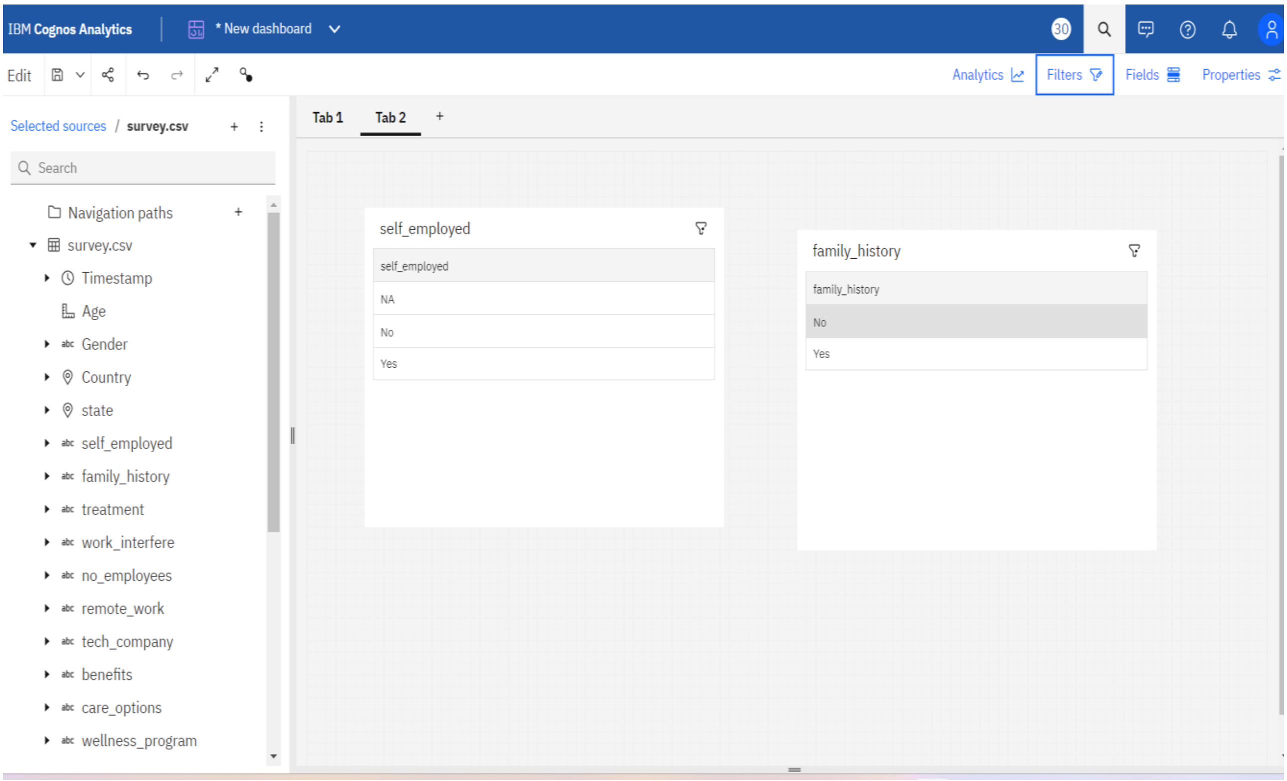


	treatment	work_interfere	family_history	care_options	benefits	obs_consequence	anonymity	mental_health_interview	wellness_program	seek_help
treatment	1.00	0.62	0.38	0.24	0.22	0.15	0.14	0.09	0.09	0.09
work_interfere	0.62	1.00	0.32	0.16	0.13	0.14	0.06	0.10	0.09	0.09
family_history	0.38	0.32	1.00	0.11	0.13	0.12	0.06	0.04	0.06	0.04
care_options	0.24	0.16	0.11	1.00	0.44	0.07	0.35	0.03	0.21	0.26
benefits	0.22	0.13	0.13	0.44	1.00	0.07	0.34	0.03	0.32	0.38
obs_consequence	0.15	0.14	0.12	0.07	0.07	1.00	0.05	0.08	0.10	0.13
anonymity	0.14	0.06	0.06	0.35	0.34	0.05	1.00	-0.01	0.23	0.32
mental_health_interview	0.09	0.10	0.04	0.03	0.03	0.08	-0.01	1.00	0.05	0.03
wellness_program	0.09	0.09	0.06	0.21	0.32	0.10	0.23	0.05	1.00	0.46
seek_help	0.09	0.09	0.04	0.26	0.38	0.13	0.32	0.03	0.46	1.00



CREATING DASHBORD USING IBM COGNOS :





IBM Cognos Analytics | * New dashboard

Edit

Analytics | Filters | Fields | Properties

Selected sources / survey.csv

Search

- Navigation paths
- survey.csv
  - Timestamp
  - Age
  - Gender
  - Country
  - state
  - self_employed
  - family_history
  - treatment
  - work_interfere
  - no_employees
  - remote_work
  - tech_company
  - benefits
  - care_options
  - wellness_program

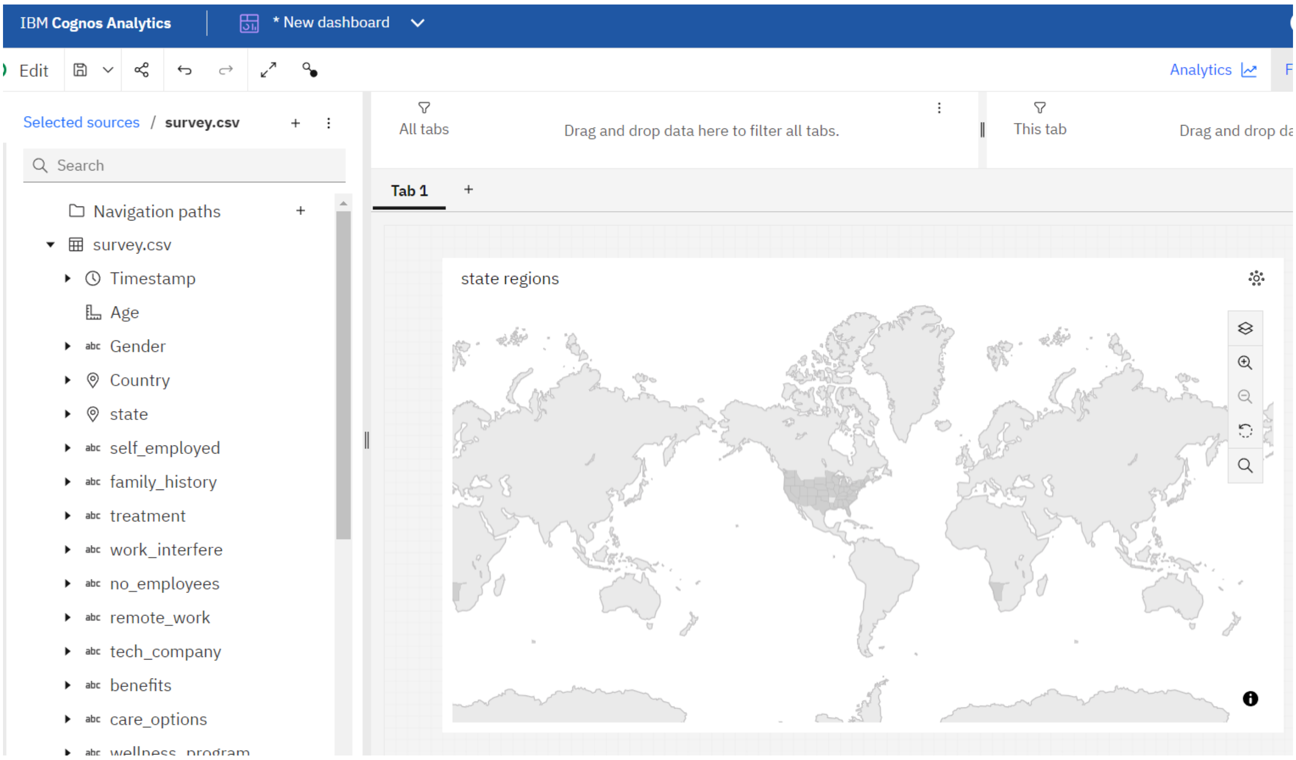
Tab 1 | Tab 2

self_employed

self_employed
NA
No
Yes

family_history

family_history
No
Yes



IBM Cognos Analytics | * New dashboard

Edit

Analytics

Selected sources / survey.csv

All tabs — Drag and drop data here to filter all tabs.

This tab — Drag and drop da

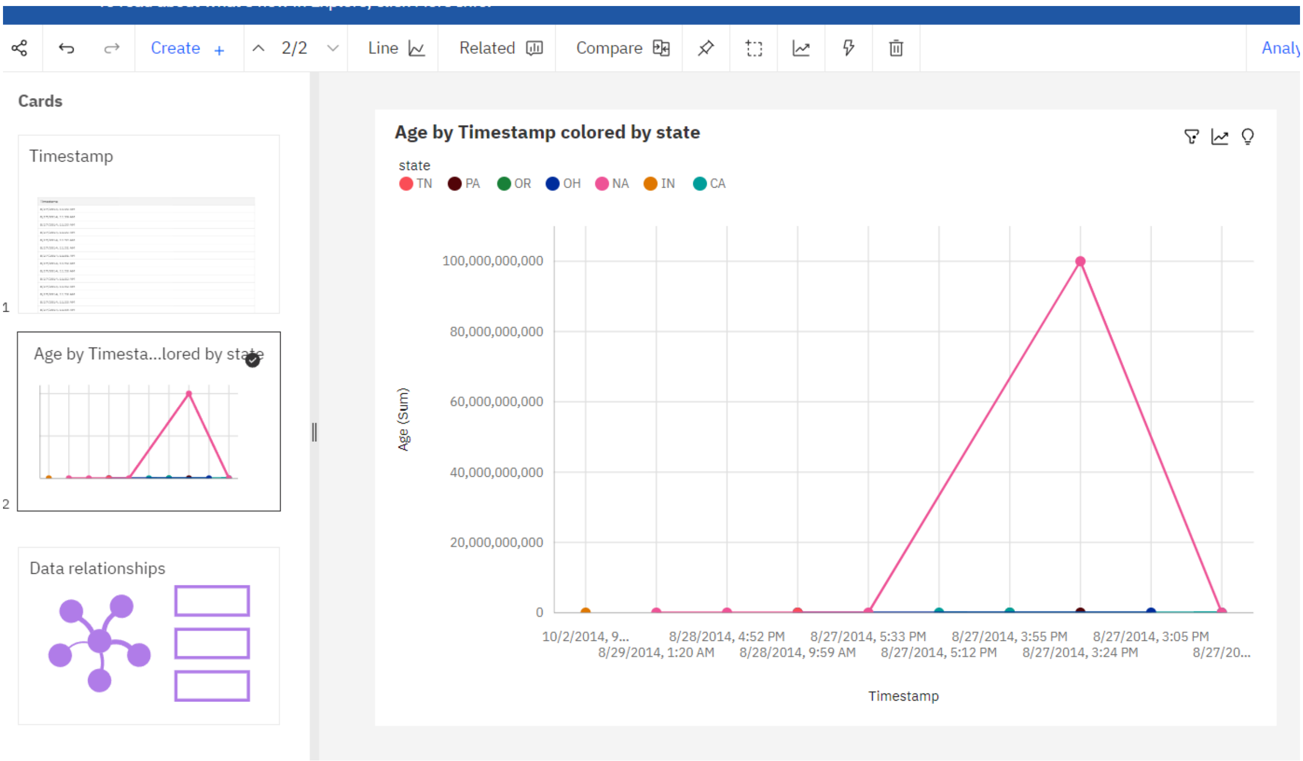
Search

- Navigation paths
- survey.csv
  - Timestamp
  - Age
  - Gender
  - Country
  - state
  - self_employed
  - family_history
  - treatment
  - work_interfere
  - no_employees
  - remote_work
  - tech_company
  - benefits
  - care_options
  - wellness_program

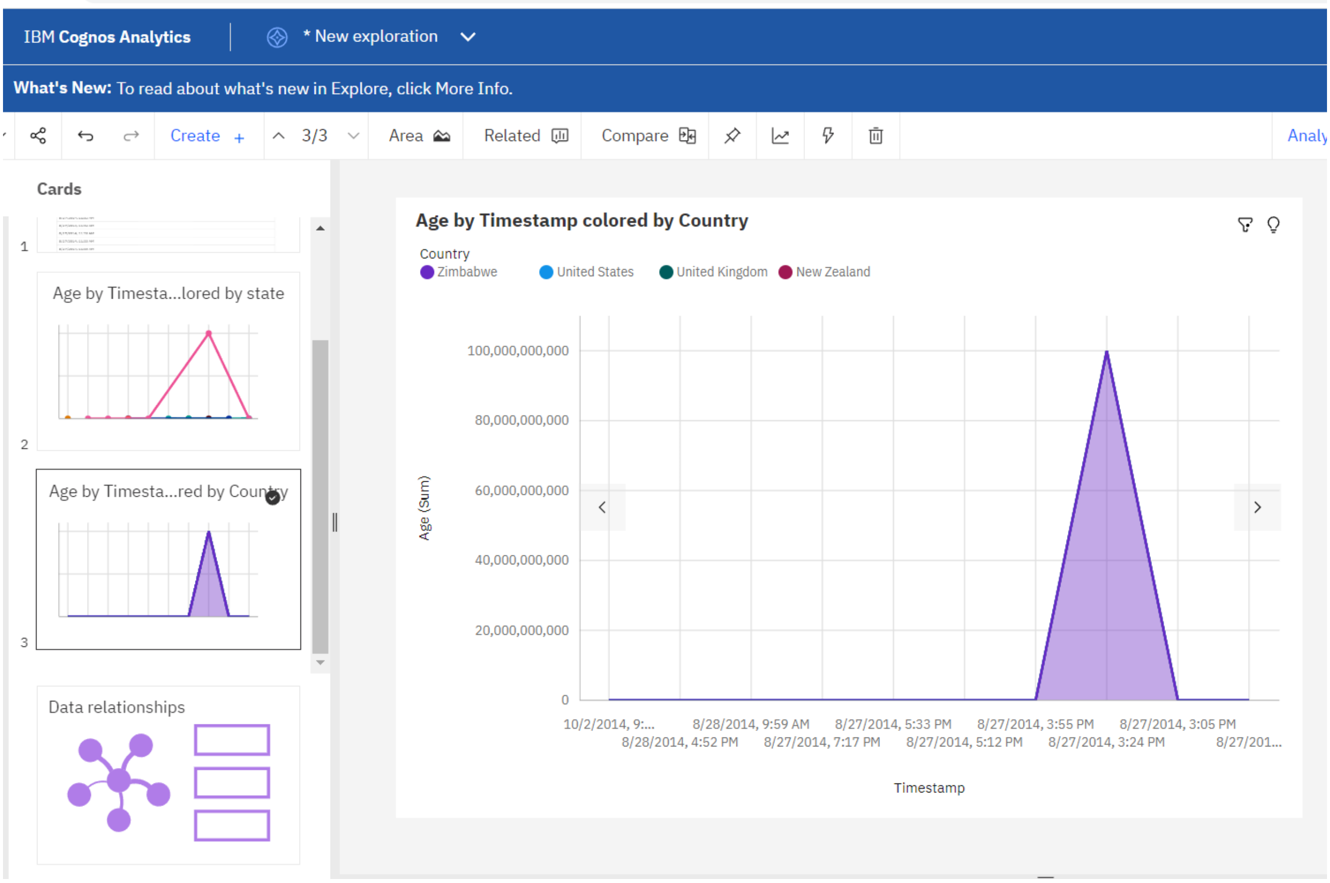
Tab 1

state regions

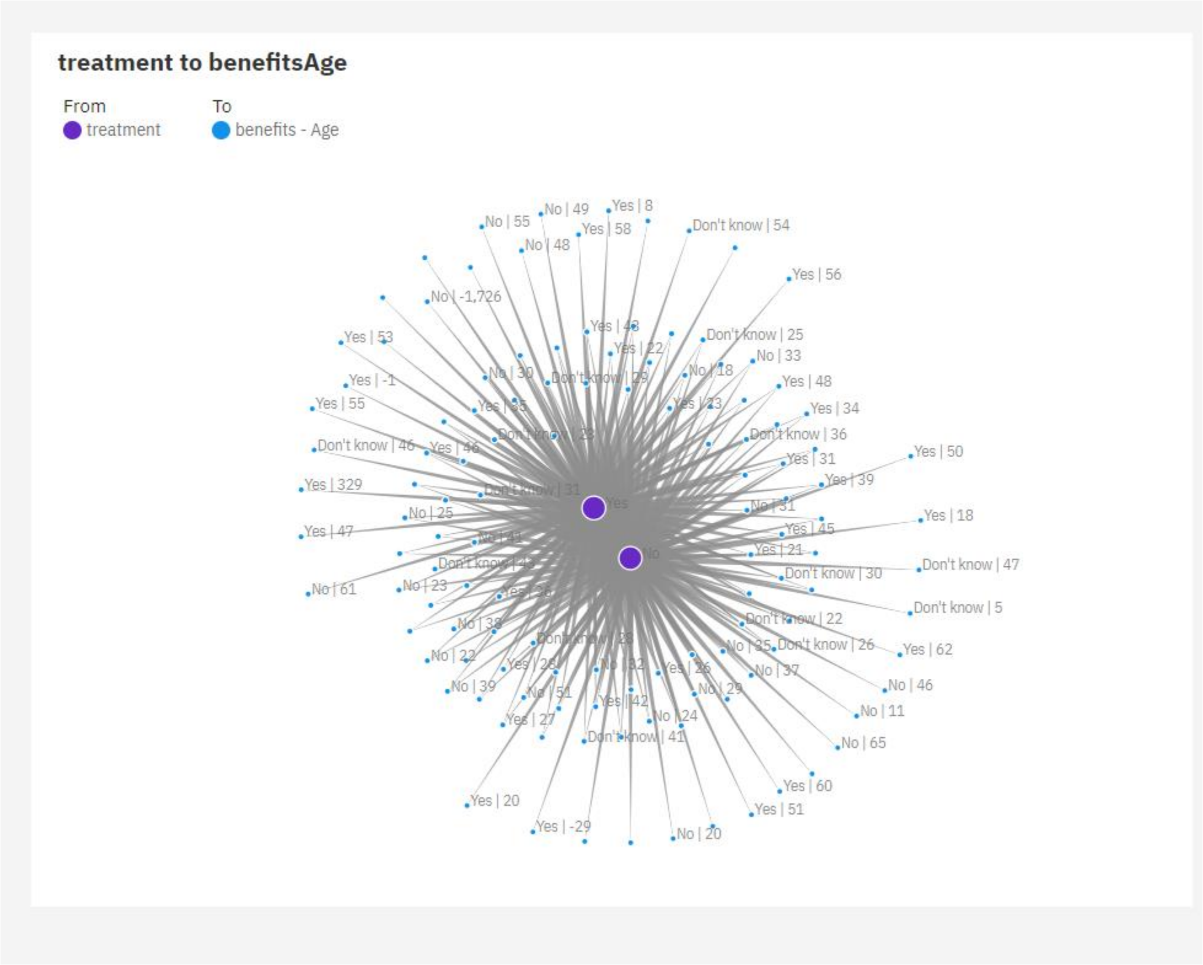
CREATING REPORT USING IBM COGNOS :



Create +
2/2
Line
Related
Compare
Analy
Cards
1
Timestamp
2
Age by Timesta...lored by state
Data relationships
Age by Timestamp colored by state
state
● TN ● PA ● OR ● OH ● NA ● IN ● CA
100,000,000,000
80,000,000,000
60,000,000,000
40,000,000,000
20,000,000,000
0
Age (Sum)
10/2/2014, 9...
8/29/2014, 1:20 AM
8/28/2014, 4:52 PM
8/28/2014, 9:59 AM
8/27/2014, 5:33 PM
8/27/2014, 5:12 PM
8/27/2014, 3:55 PM
8/27/2014, 3:24 PM
8/27/2014, 3:05 PM
8/27/20...
Timestamp



IBM Cognos Analytics
* New exploration
What's New: To read about what's new in Explore, click More Info.
Create +
3/3
Area
Related
Compare
Analy
Cards
1
2
Age by Timesta...lored by state
3
Age by Timesta...red by Country
Data relationships
Age by Timestamp colored by Country
Country
● Zimbabwe ● United States ● United Kingdom ● New Zealand
100,000,000,000
80,000,000,000
60,000,000,000
40,000,000,000
20,000,000,000
0
Age (Sum)
10/2/2014, 9:...
8/28/2014, 4:52 PM
8/28/2014, 9:59 AM
8/27/2014, 7:17 PM
8/27/2014, 5:33 PM
8/27/2014, 5:12 PM
8/27/2014, 3:55 PM
8/27/2014, 3:24 PM
8/27/2014, 3:05 PM
8/27/201...
Timestamp



treatment to benefitsAge