***PUBLIC HEALTH AWARENESS***

***NAME : BIPUL PANDEY***

***REG NO : 620821104020***

***DEPT : CSE(A SECTION)***

***COLLEGE CODE : 6208***

***COLLEGE NAME : GNANAMANI COLLEGE OF TECHNOLOGY***

***SKILLUP ID : bipul9162866323@gmail.com***

**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))

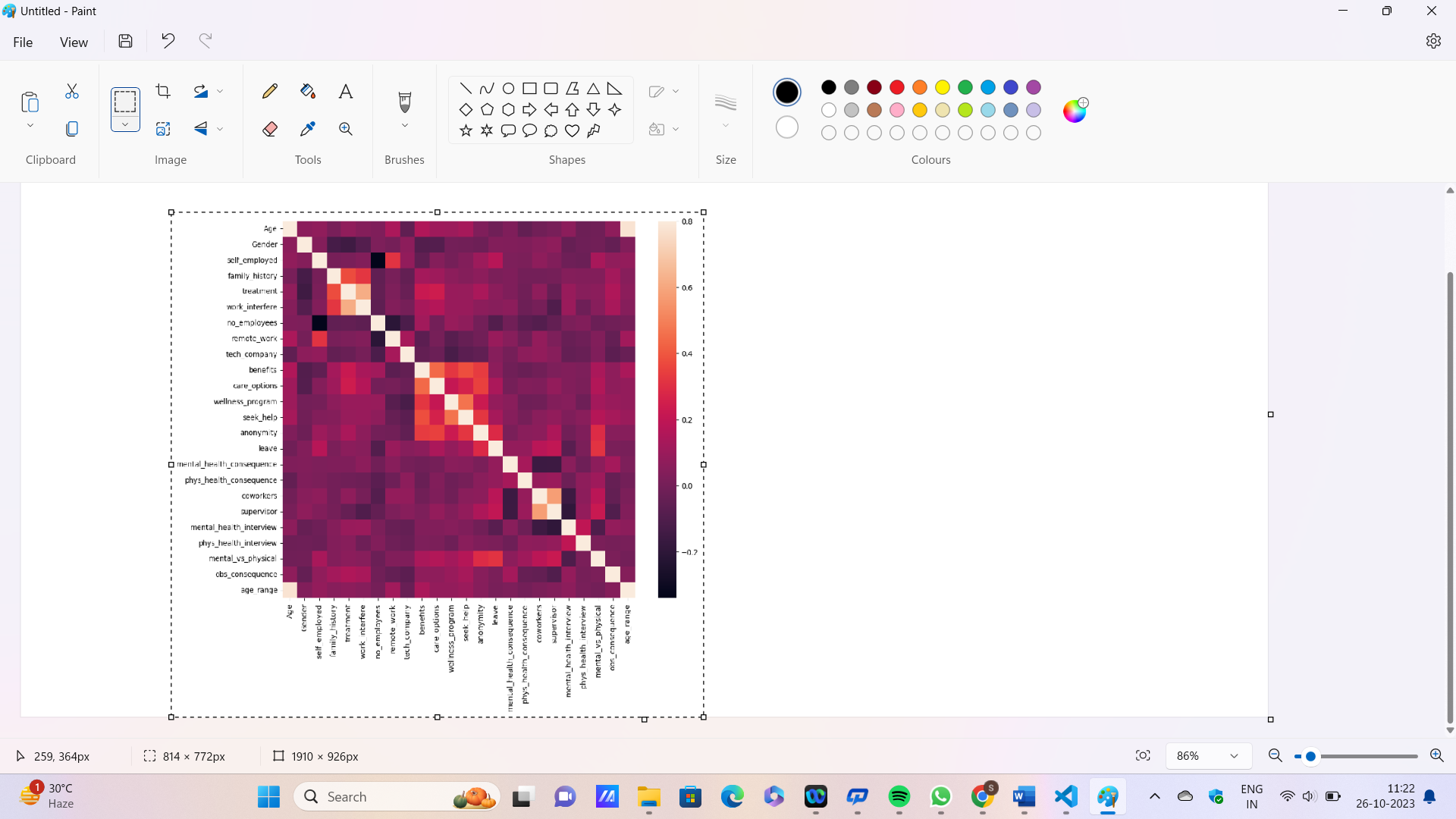
labels = labelDict['label\_Gender']

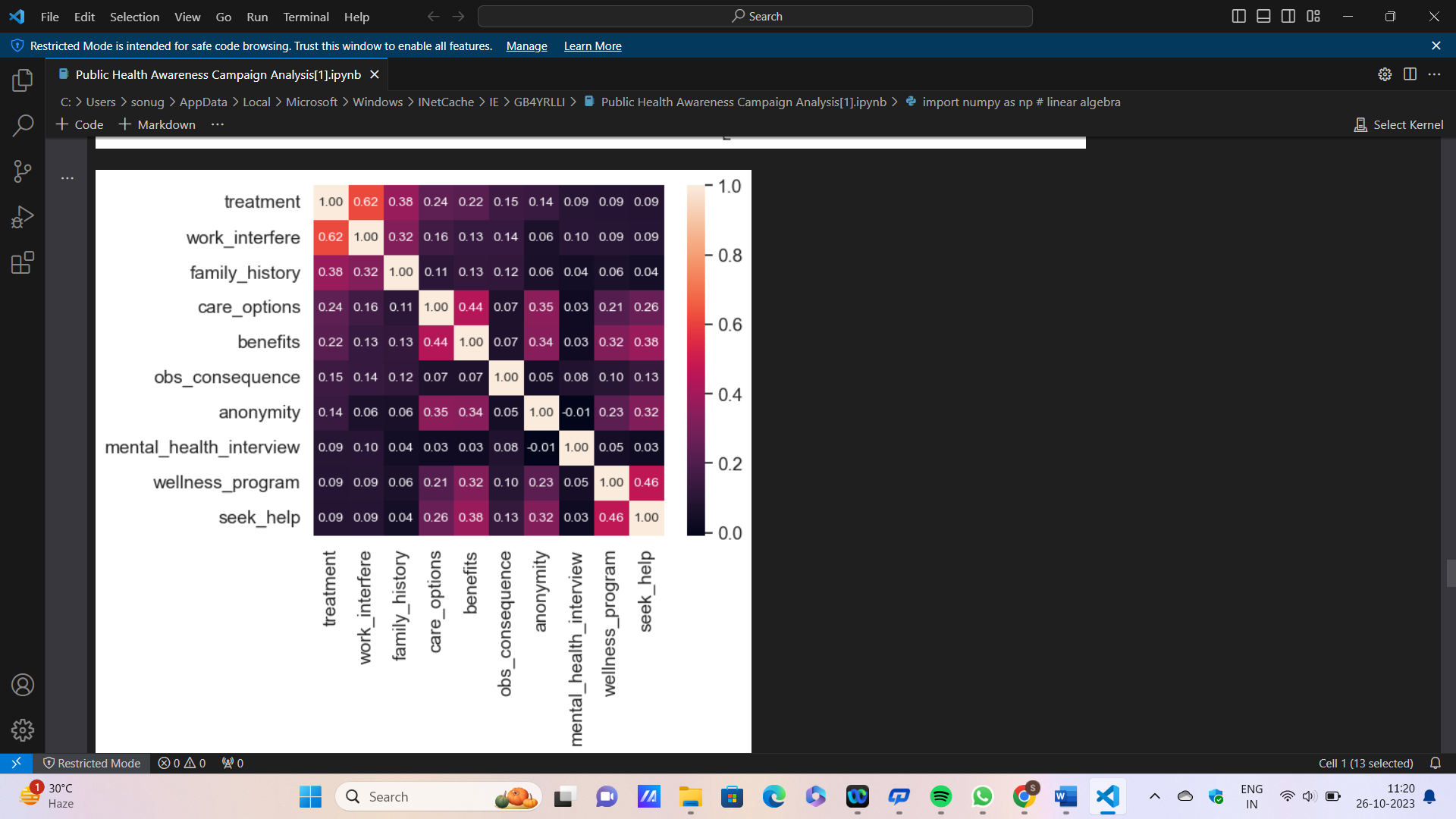
g = sns.countplot(x="treatment", data=train\_df)

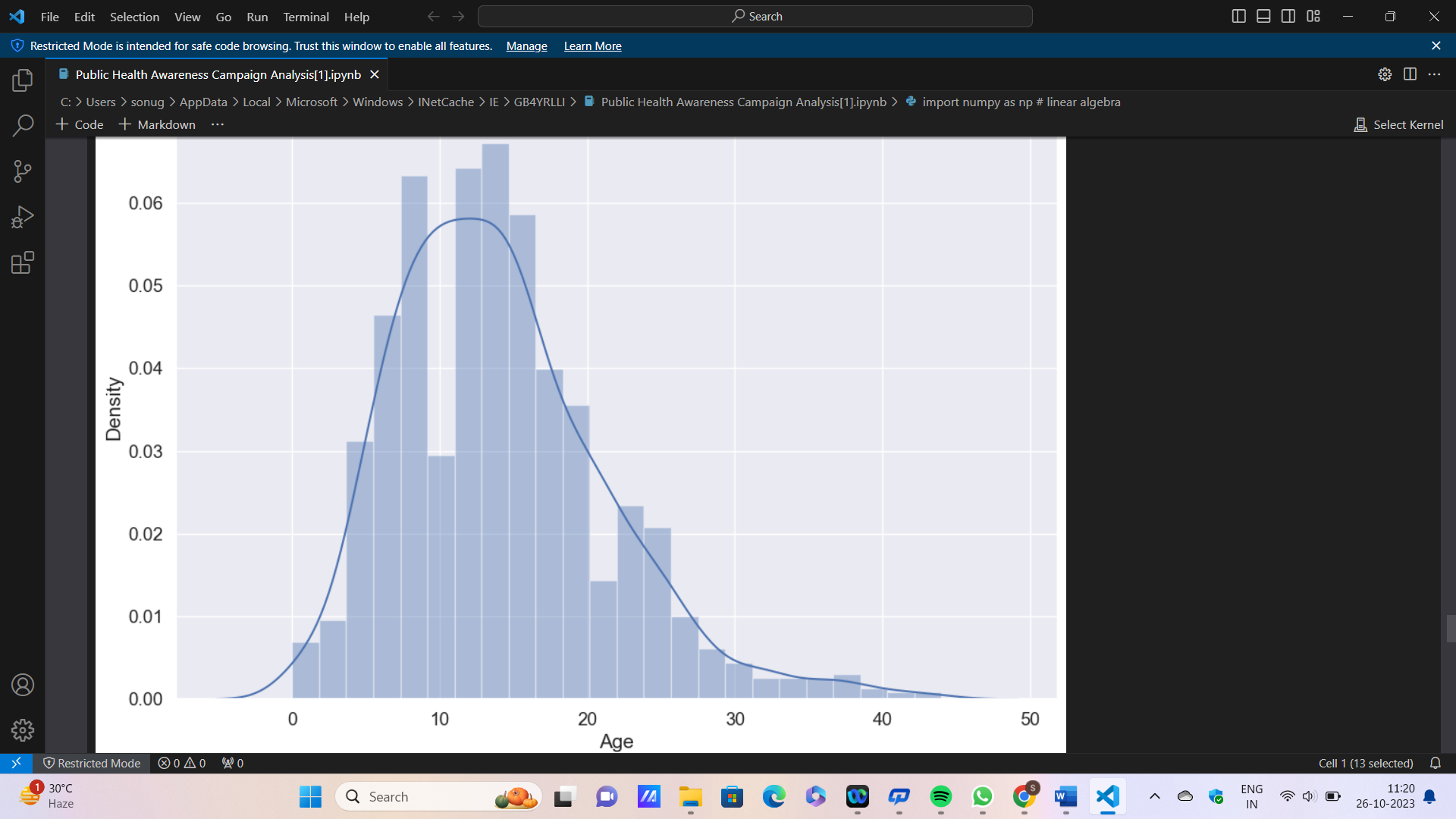
g.set\_xticklabels(labels)

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

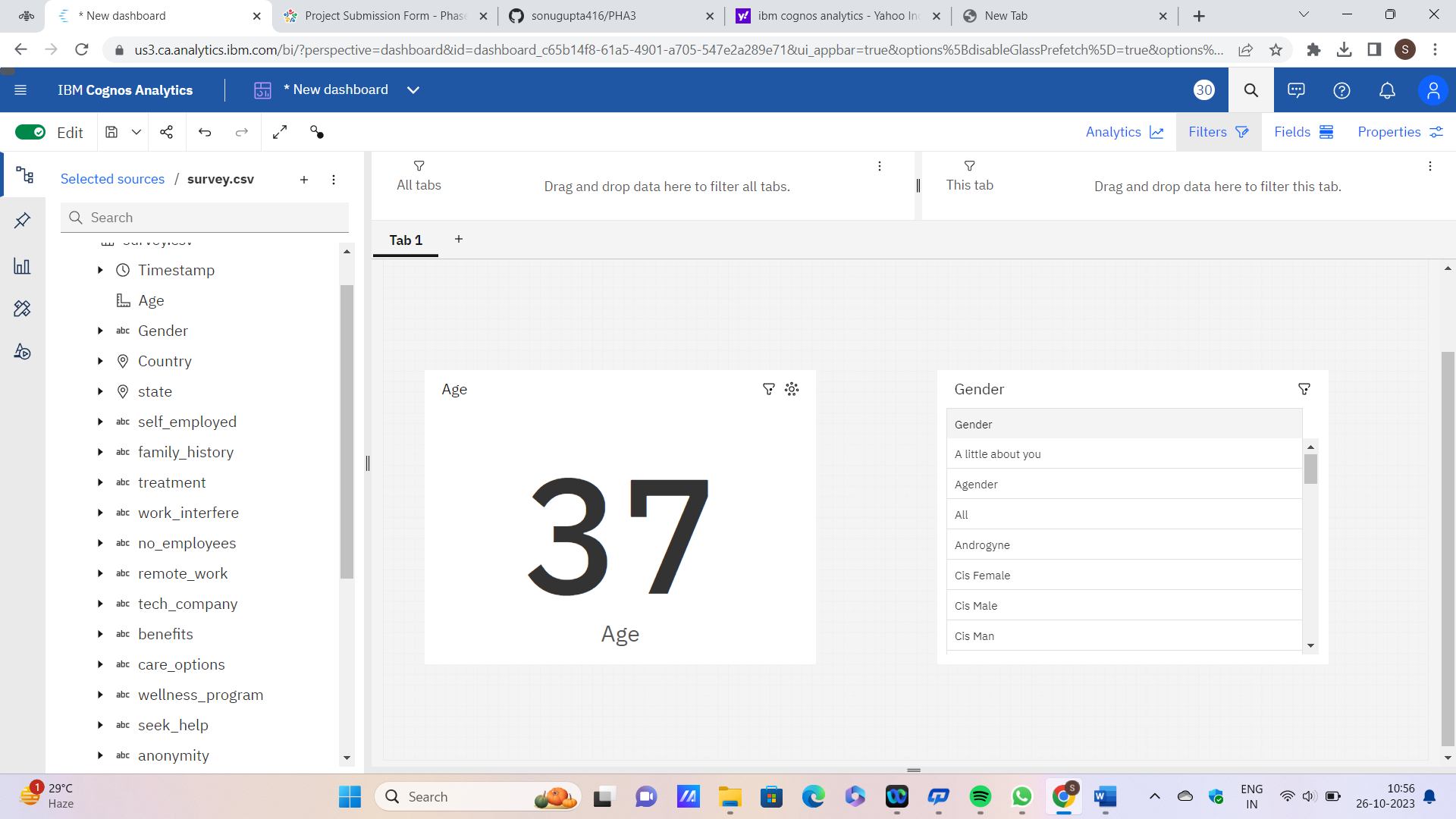
**OUTPUT :**

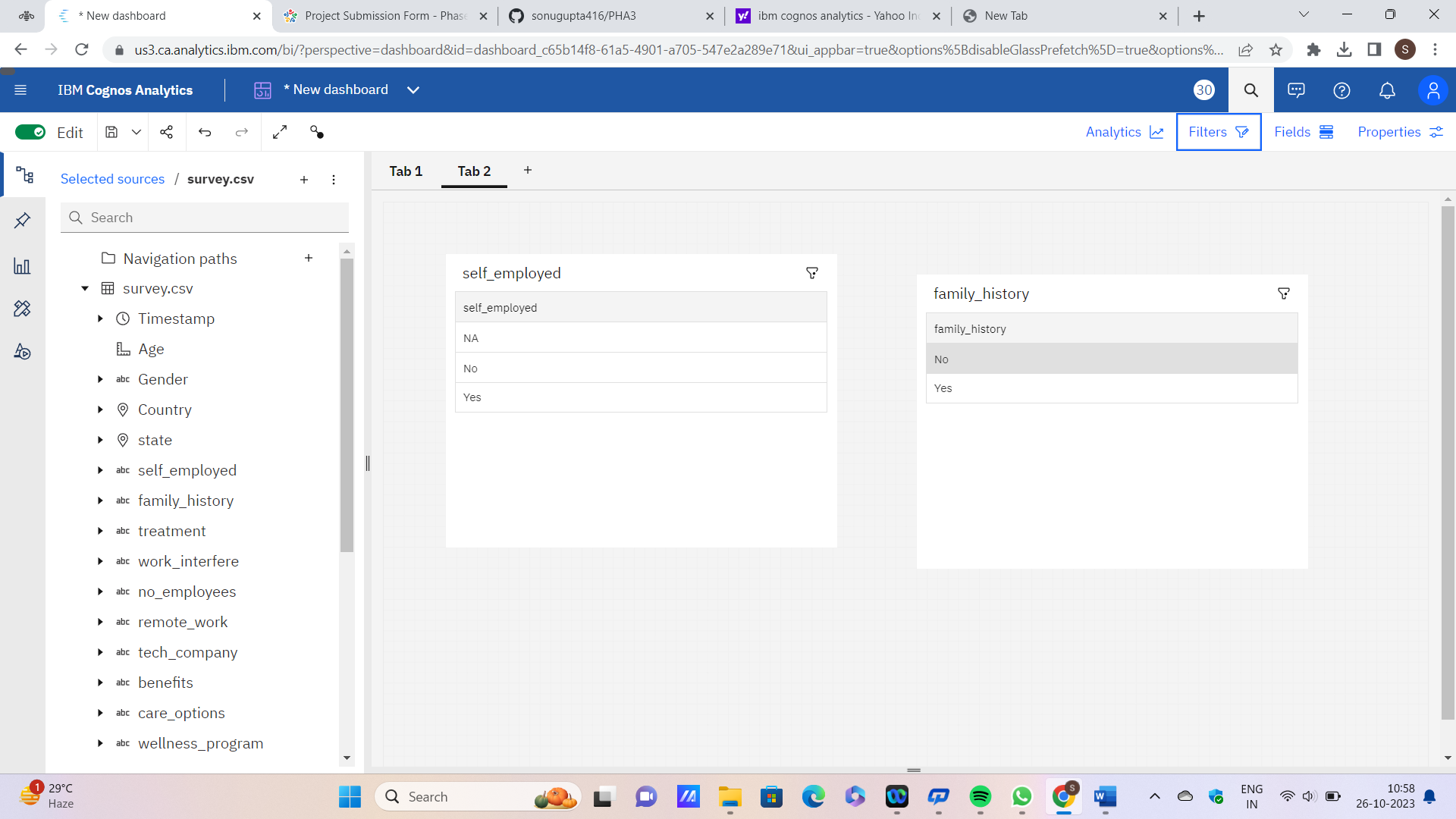
****

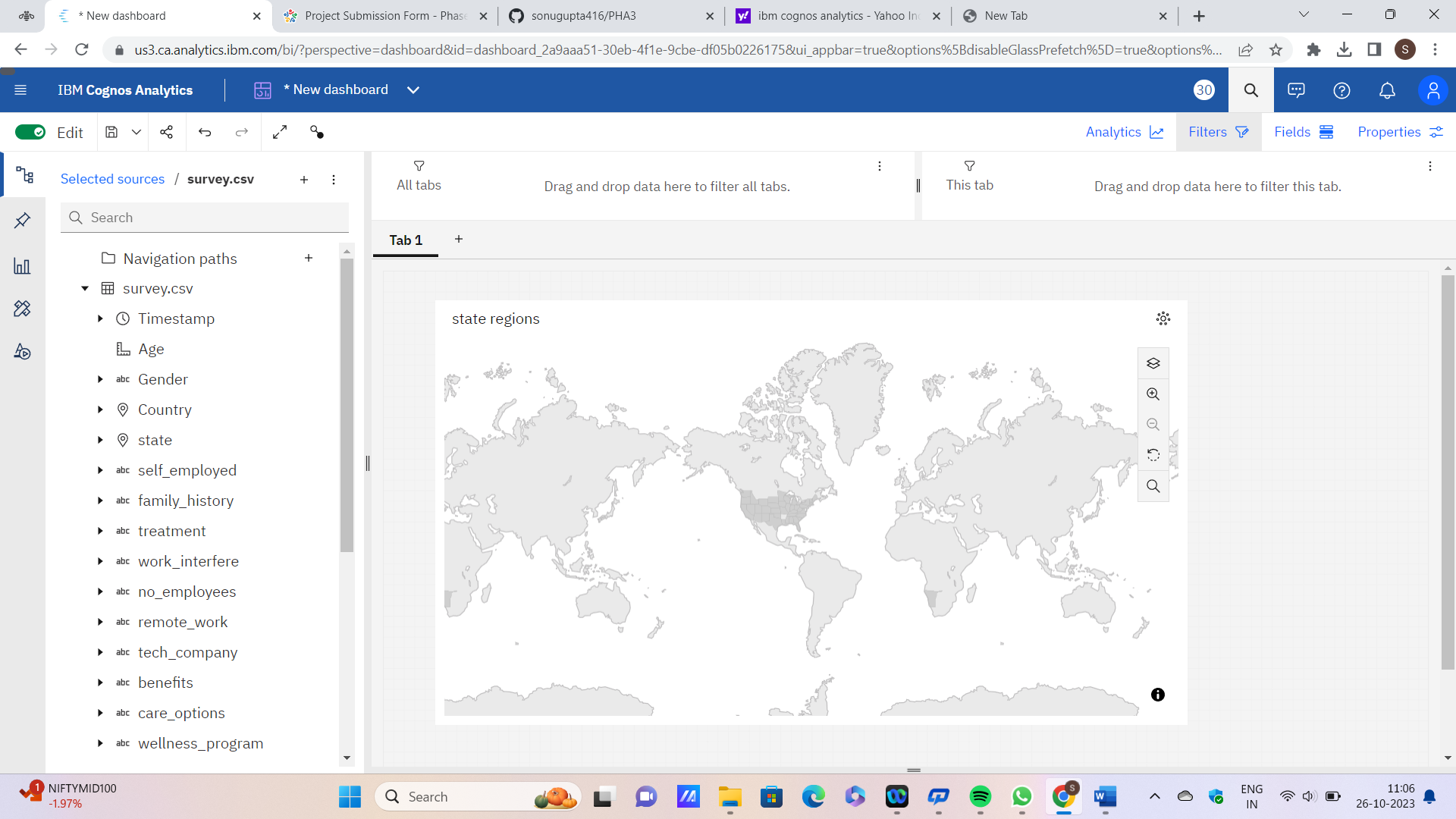
****

****

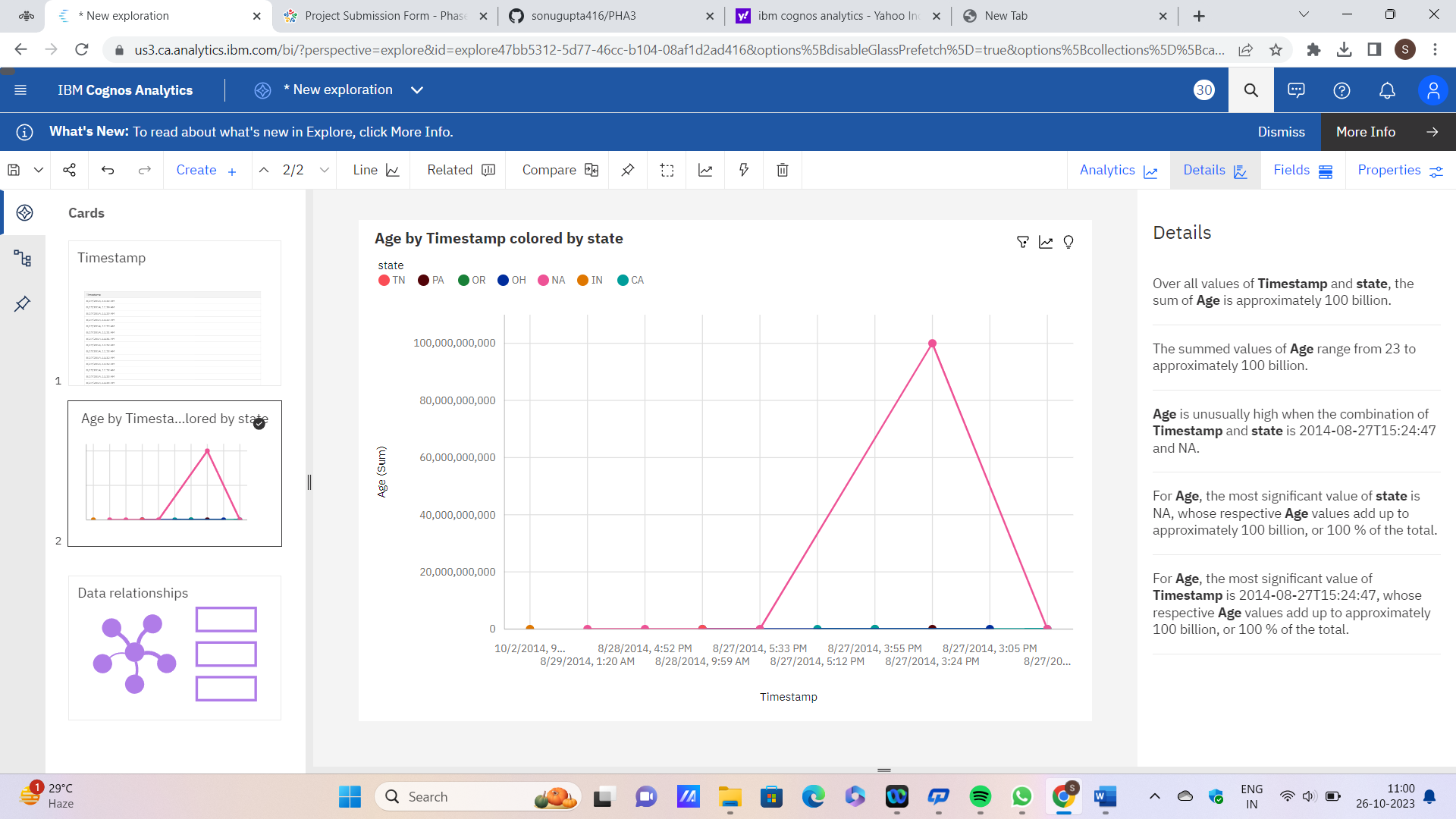
**CREATING DASHBORD USING IBM COGNOS :**

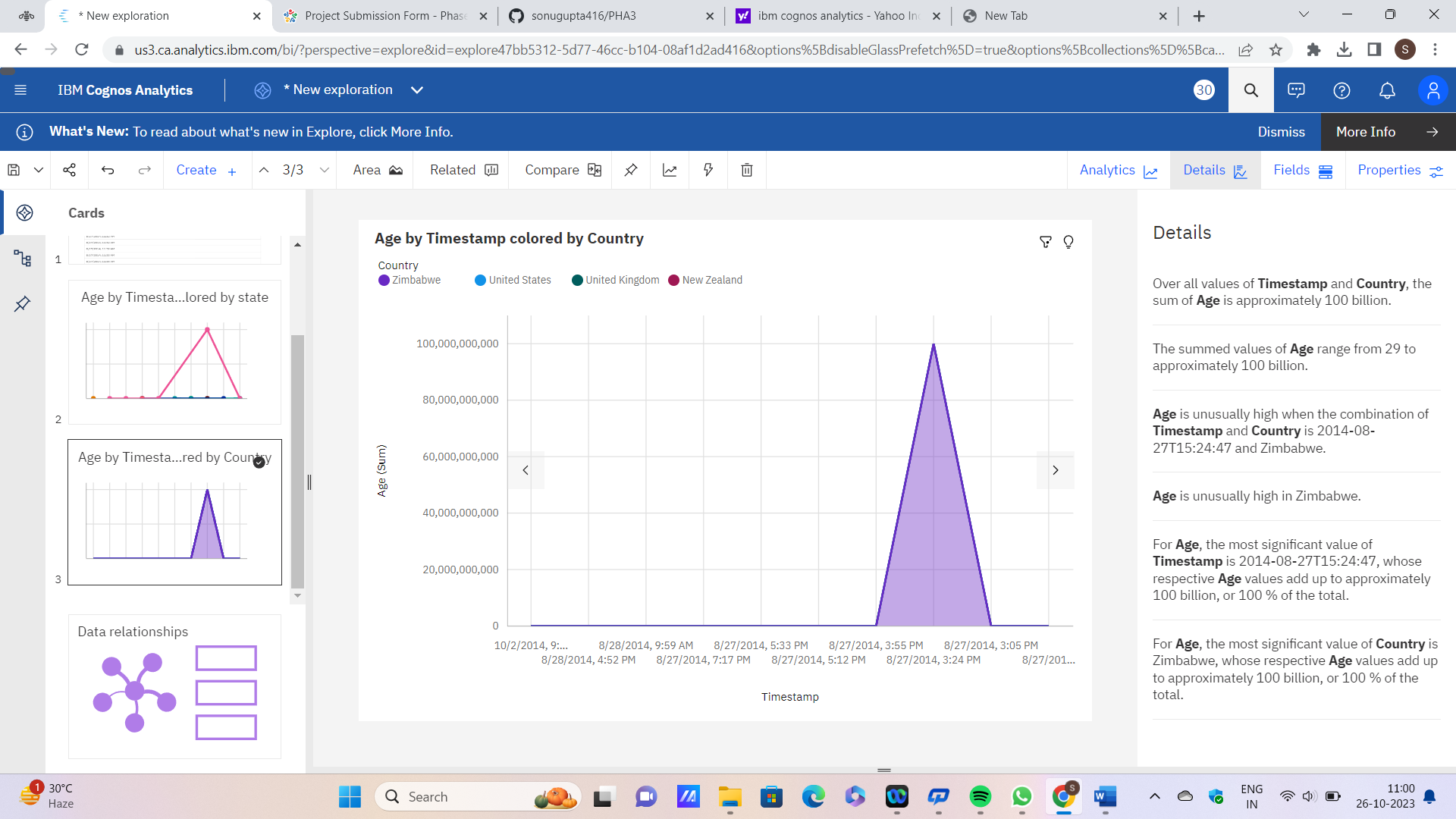
****

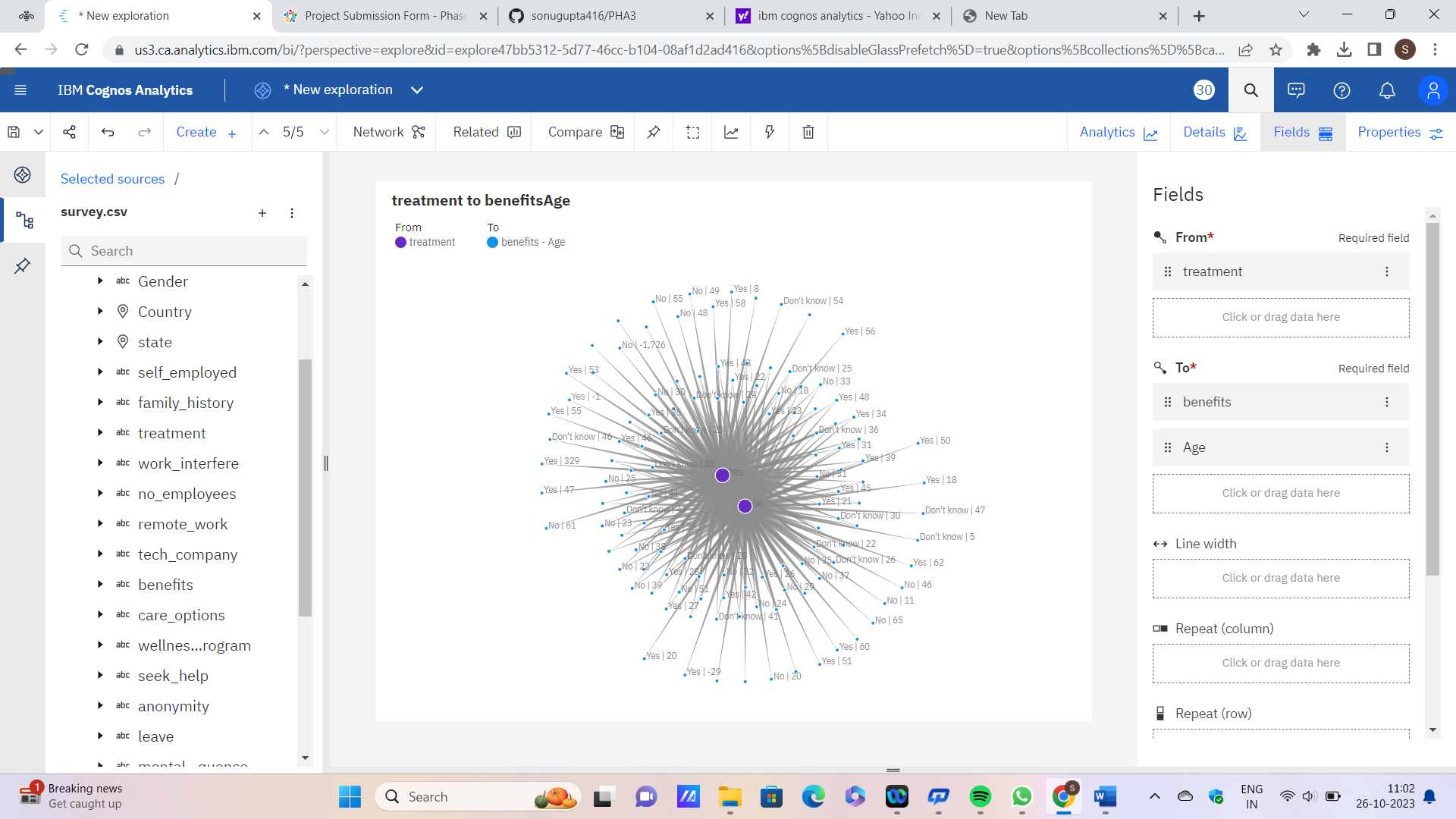
****

****

**CREATING REPORT USING IBM COGNOS :**

****

****

****