dac-phase-3

October 26, 2023

Date:26/10/2023

Team ID:716

Project Name:Public Health Awareness Campaign Analysis using Data Analytics

IMPORT DEPENDENCIES

[6]:
```
# suppress display of warnings
import warnings
warnings.filterwarnings("ignore")

# 'Pandas' is used for data manipulation and analysis
import pandas as pd

# 'Numpy' is used for mathematical operations on large, multi-dimensional␣
↪arrays and matrices
import numpy as np

# 'Matplotlib' is a data visualization library for 2D and 3D plots, built on␣
↪numpy
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

# 'Seaborn' is based on matplotlib; used for plotting statistical graphics
import seaborn as sns

# import 'is_string_dtype' to check if the type of input is string
from pandas.api.types import is_string_dtype

# import various functions to perform classification
from sklearn.naive_bayes import GaussianNB
from sklearn.multiclass import OneVsRestClassifier
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
```

```python
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn import tree
from sklearn.tree import export_graphviz
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.feature_selection import SelectFromModel
from sklearn.svm import SVC
# display all columns of the dataframe
pd.options.display.max_columns = None
```

[31]:
```python
#setting the plot size using rcParams
plt.rcParams['figure.figsize'] = [10,8]
```

[13]:
```python
#importing datasets for training and testing the models
df=pd.read_csv("C:/Users/sjana/Downloads/public health survey.csv")

#rwading the first 5 records from the training data set
df.head()
```

[13]:
```
             Timestamp  Age  Gender         Country state self_employed  \
0  2014-08-27 11:29:31   37  Female   United States    IL           NaN
1  2014-08-27 11:29:37   44       M   United States    IN           NaN
2  2014-08-27 11:29:44   32    Male          Canada   NaN           NaN
3  2014-08-27 11:29:46   31    Male  United Kingdom   NaN           NaN
4  2014-08-27 11:30:22   31    Male   United States    TX           NaN

  family_history treatment work_interfere    no_employees remote_work  \
0             No       Yes          Often            6-25          No
1             No        No         Rarely  More than 1000          No
2             No        No         Rarely            6-25          No
3            Yes       Yes          Often          26-100          No
4             No        No          Never         100-500         Yes

  tech_company    benefits care_options wellness_program   seek_help  \
0          Yes         Yes     Not sure               No         Yes
1           No  Don't know           No       Don't know  Don't know
2          Yes          No           No               No          No
3          Yes          No          Yes               No          No
4          Yes         Yes           No       Don't know  Don't know

    anonymity               leave mental_health_consequence  \
0         Yes       Somewhat easy                        No
1  Don't know          Don't know                     Maybe
2  Don't know  Somewhat difficult                        No
```

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

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

  phys_health_interview mental_vs_physical obs_consequence comments
0                 Maybe                Yes              No      NaN
1                    No         Don't know              No      NaN
2                   Yes                 No              No      NaN
3                 Maybe                 No             Yes      NaN
4                   Yes         Don't know              No      NaN
```

UNDERSTANDING DATA

[14]:
```python
#checking the number of rows and columns in the training data set
df.shape
```

[14]:
```
(1259, 27)
```

[15]:
```python
# 'dtypes' gives the data type for each column
df.dtypes
```

[15]:
```
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
```

[16]:
```
#Splitting the timestamp feature as it includes both date and time
df[['Date','Time']]=df['Timestamp'].str.split(" ",n=1,expand=True)
```

[17]:
```
#dropping the timestamp column as we have already created two columns which␣
↪have date and time
#axis=1 deletes the entire column
df.drop('Timestamp',axis=1,inplace=True)
```

[18]:
```
#converting the datatype of date and time columns
df['Time']=pd.to_datetime(df['Time'],format='%H:%M:%S')
df['Date']=pd.to_datetime(df['Date'])
```

[19]:
```
#finally checking the columns and their datatypes after alteration
df.dtypes
```

[19]:
```
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
Date                           datetime64[ns]
Time                           datetime64[ns]
dtype: object
```

[20]:
```python
# the describe() returns the statistical summary of the numeric variables
df.describe()
```

[20]:
```
                Age                           Date  \
count  1.259000e+03                           1259
mean   7.942815e+07  2014-09-09 10:54:14.011120128
min   -1.726000e+03            2014-08-27 00:00:00
25%    2.700000e+01            2014-08-27 00:00:00
50%    3.100000e+01            2014-08-28 00:00:00
75%    3.600000e+01            2014-08-28 00:00:00
max    1.000000e+11            2016-02-01 00:00:00
std    2.818299e+09                            NaN

                                Time
count                           1259
mean   1900-01-01 13:21:38.166004992
min              1900-01-01 00:02:36
25%       1900-01-01 11:19:59.500000
50%              1900-01-01 13:18:44
75%              1900-01-01 16:13:40
max              1900-01-01 23:59:59
std                              NaN
```

[21]:
```python
#it gives the statistical summary of all the categorical variables
df.describe(include=object)
```

[21]:
```
       Gender        Country state self_employed family_history treatment  \
count    1259           1259   744          1241           1259      1259
unique     49             48    45             2              2         2
top      Male  United States    CA            No             No       Yes
freq      615            751   138          1095            767       637

       work_interfere no_employees remote_work tech_company benefits  \
count             995         1259        1259         1259     1259
unique              4            6           2            2        3
top         Sometimes         6-25          No          Yes      Yes
freq              465          290         883         1031      477

       care_options wellness_program seek_help   anonymity       leave  \
count          1259             1259      1259        1259        1259
```

```
unique             3                3         3           3           5
top               No               No        No  Don't know  Don't know
freq             501              842       646         819         563

       mental_health_consequence phys_health_consequence     coworkers  \
count                       1259                    1259          1259
unique                         3                       3             3
top                           No                      No  Some of them
freq                         490                     925           774

       supervisor mental_health_interview phys_health_interview  \
count        1259                    1259                  1259
unique          3                       3                     3
top           Yes                      No                 Maybe
freq          516                    1008                   557

       mental_vs_physical obs_consequence                        comments
count                1259            1259                             164
unique                  3               2                             160
top            Don't know              No  * Small family business - YMMV.
freq                  576            1075                               5
```

DATA PREPRATION

[22]:
```
#finding the unique values in the column gender
df['Gender'].unique()
```

[22]:
```
array(['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'], dtype=object)
```

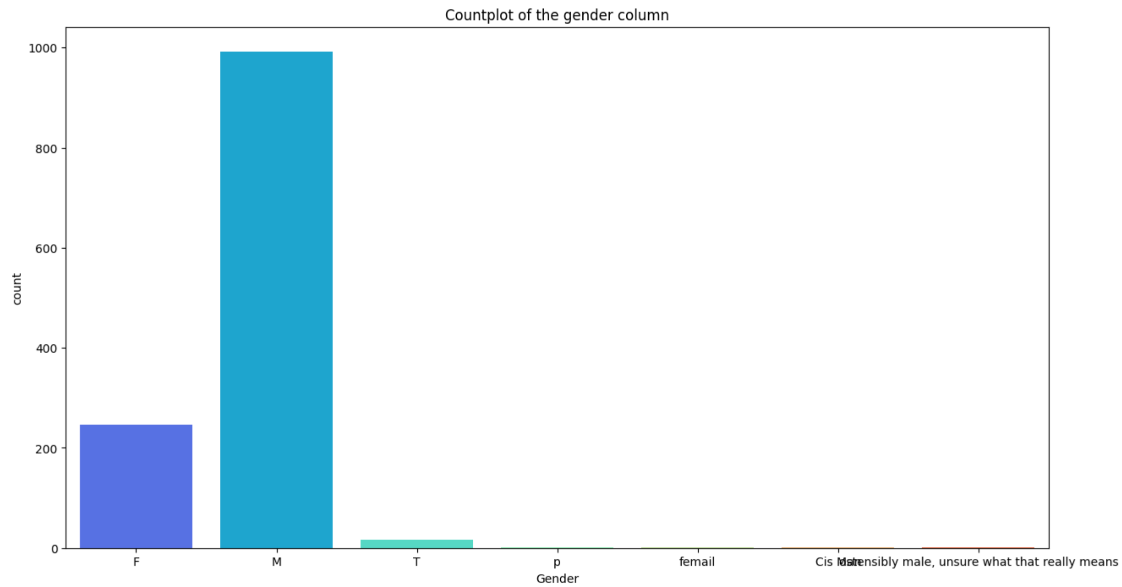
[23]:
```
error={'Female':'F',
      'Male':'M',
      'male':'M',
      'female':'F',
      'm':'M',
      'Male-ish':'M',
      'maile':'M',
      'Trans-female':'T',
      'Cis Female':'F',
```

```
       'something kinda male?':'M',
       'Cis Male':'M',
       'Woman':'F',
       'f':'F',
       'Mal':'M',
        'Male (CIS)':'M',
       'queer/she/they':'F',
       'non-binary':'T',
       'Enby':'T',
       'Femake':'F',
       'woman':'F',
       'Make':'M',
       'fluid':'T',
       'Malr':'M',
       'cis male':'M',
       'Female (cis)':'F',
       'Guy (-ish) ^_^':'M',
       'queer':'T',
       'Female (trans)':'T',
       'male leaning androgynous':'T',
        'Neuter':'T',
        'cis-female/femme':'F',
        'msle':'M',
        'Agender':'T',
        'Genderqueer':'T',
        'Female':'F',
        'Androgyne':'T',
        'Nah':'T',
        'All':'T',
       'Female ':'F',
        'Male ':'M',
        'Man':'M',
        'Trans woman':'T',
        'Mail':'M',
        'A little about you':'T'}
df['Gender']=df['Gender'].map(error).fillna(df['Gender'])
```

[24]:
```
df['Gender'].unique()
```

[24]:
```
array(['F', 'M', 'T', 'p', 'femail', 'Cis Man',
       'ostensibly male, unsure what that really means'], dtype=object)
```

[25]:
```
#plotting the countplot for the gender column
sns.countplot(df['Gender'],palette='rainbow')
plt.title("Countplot of the gender column")
plt.show()
```



[26]:
```
#Checking the number of male,female and transgender in the gender column
df['Gender'].value_counts()
```

[26]:
```
Gender
M                                                 992
F                                                 247
T                                                  16
p                                                   1
femail                                              1
Cis Man                                             1
ostensibly male, unsure what that really means      1
Name: count, dtype: int64
```

[27]:
```
#finding the unique values of age column
df['Age'].unique()
```

[27]:
```
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], dtype=int64)
```

[28]:
```
#removing the redundant values from the age column
df=df[df['Age']!=99999999999]
df=df[df['Age']!=-29]
df=df[df['Age']!=329]
df=df[df['Age']!=-1726]
df=df[df['Age']!=5]
df=df[df['Age']!=8]
```

LABEL ENCODING OF TREATMENT COLUMN

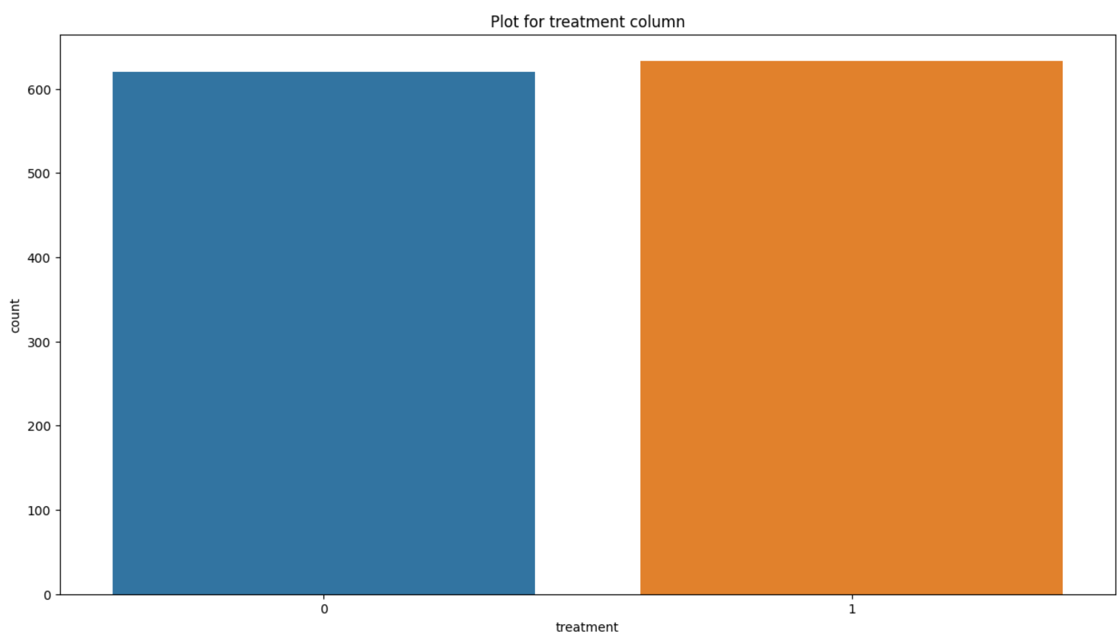
[29]:
```
# replace 'no' with zero
df['treatment'] = df['treatment'].replace('No', 0)
# replace 'yes' with one
df['treatment'] = df['treatment'].replace('Yes', 1)

#displaying the first 5 records to check the treatment column after label␣
↪enconding
df.head()
```

[29]:
```
   Age Gender         Country state self_employed family_history  treatment  \
0   37      F   United States    IL           NaN             No          1
1   44      M   United States    IN           NaN             No          0
2   32      M          Canada   NaN           NaN             No          0
3   31      M  United Kingdom   NaN           NaN            Yes          1
4   31      M   United States    TX           NaN             No          0

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

  care_options wellness_program   seek_help   anonymity               leave  \
0     Not sure               No         Yes         Yes       Somewhat easy
1           No       Don't know  Don't know  Don't know          Don't know
2           No               No          No  Don't know  Somewhat difficult
3          Yes               No          No          No  Somewhat difficult
4           No       Don't know  Don't know  Don't know          Don't know

  mental_health_consequence phys_health_consequence     coworkers supervisor  \
0                        No                      No  Some of them        Yes
1                     Maybe                      No            No         No
2                        No                      No           Yes        Yes
3                       Yes                     Yes  Some of them         No
4                        No                      No  Some of them        Yes
```

```
  mental_health_interview phys_health_interview mental_vs_physical  \
0                      No                 Maybe                Yes
1                      No                    No         Don't know
2                     Yes                   Yes                 No
3                   Maybe                 Maybe                 No
4                     Yes                   Yes         Don't know

  obs_consequence comments       Date                Time
0              No      NaN 2014-08-27 1900-01-01 11:29:31
1              No      NaN 2014-08-27 1900-01-01 11:29:37
2              No      NaN 2014-08-27 1900-01-01 11:29:44
3             Yes      NaN 2014-08-27 1900-01-01 11:29:46
4              No      NaN 2014-08-27 1900-01-01 11:30:22
```

[30]:
```
#plotting the countplot for treatment column
sns.countplot(df['treatment'])
plt.title("Plot for treatment column")

#checking the count of each class
df['treatment'].value_counts()
```

[30]:
```
treatment
1    633
0    620
Name: count, dtype: int64
```

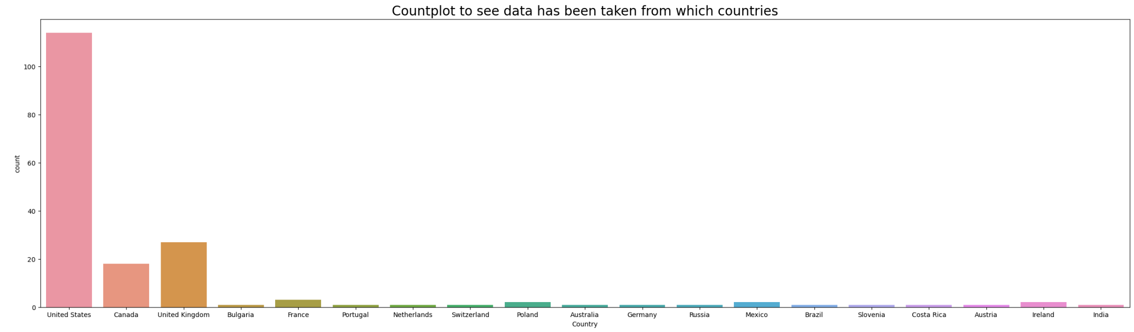


EXPLORATORY DATA ANALYSIS

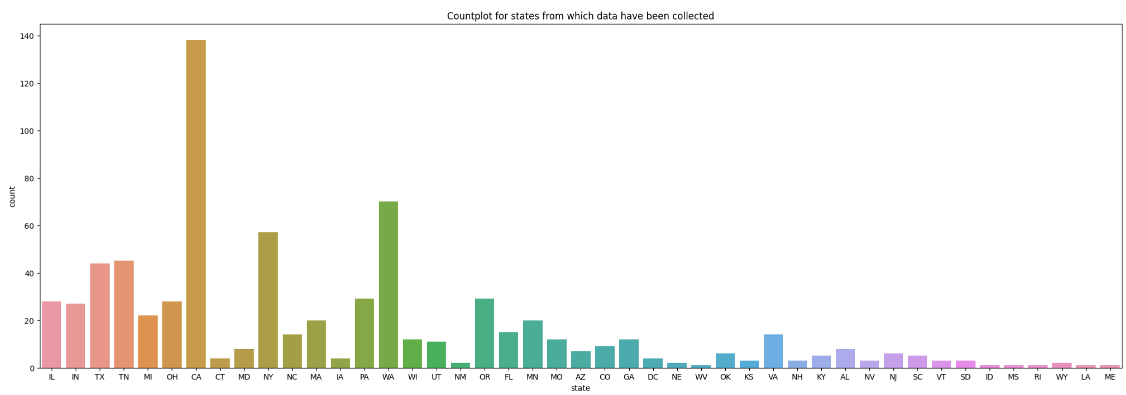
[32]:

```
# create a list of all categorical variables
# initiate an empty list to store the categorical variables
categorical=[]
z=['Country','state']
# use for loop to check the data type of each variable
for column in df:

    # use 'if' statement with condition to check the categorical type
    if is_string_dtype(df[column]):
        if column!=z:

        # append the variables with 'categoric' data type in the list␣
↪'categorical'
            categorical.append(column)


# plot the count plot for each categorical variable
fig, ax = plt.subplots(nrows = 5, ncols = 4, figsize=(25, 30))

# use for loop to plot the count plot for each variable
for variable, subplot in zip(categorical, ax.flatten()):

    # use countplot() to plot the graph
    sns.countplot(df[variable], ax = subplot)

# display the plot
plt.show()
```

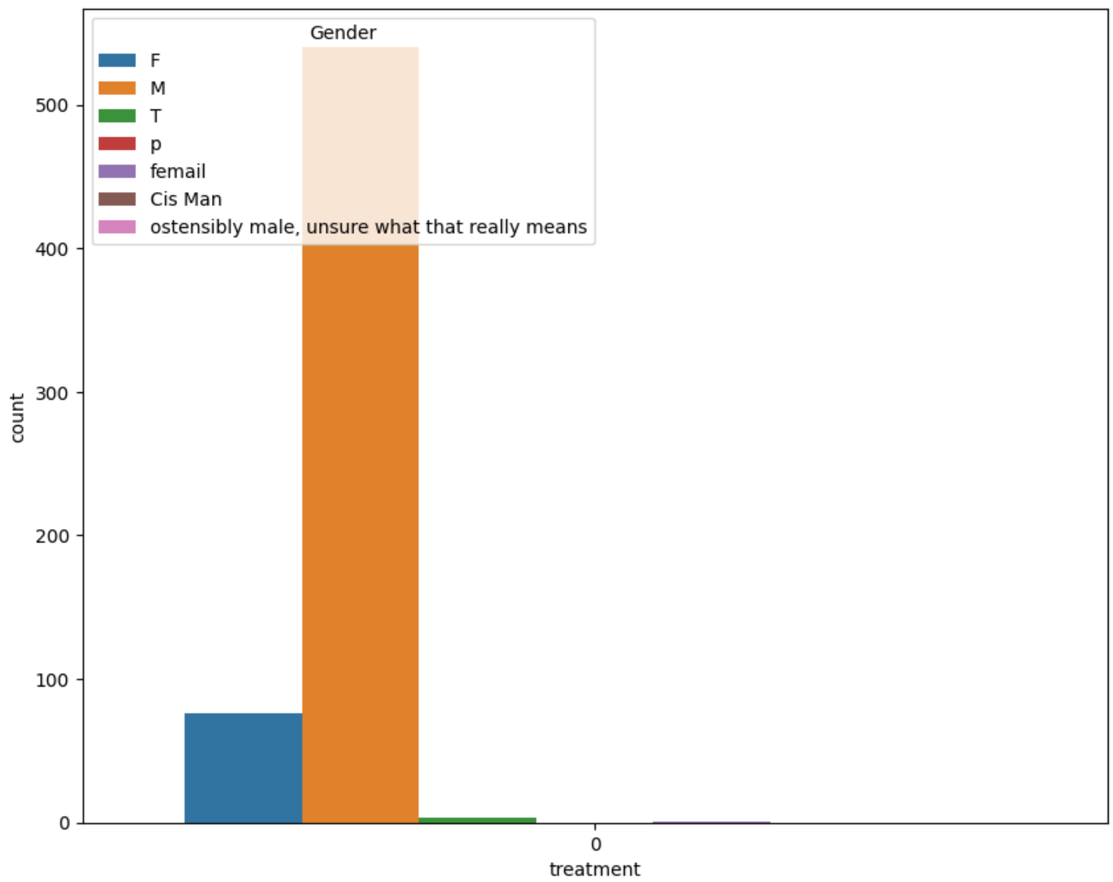

```
[33]: # plotting the counterplot for the country column
      #to see
      plt.figure(figsize=(30,8))
      sns.countplot(df['Country'][:180])
      plt.title("Countplot to see data has been taken from which␣
        ↪countries",fontsize=20)
      plt.show()
```



[34]:
```
plt.figure(figsize=(25,8))
sns.countplot(df['state'])
plt.title("Countplot for states from which data have been collected")
plt.show()
```

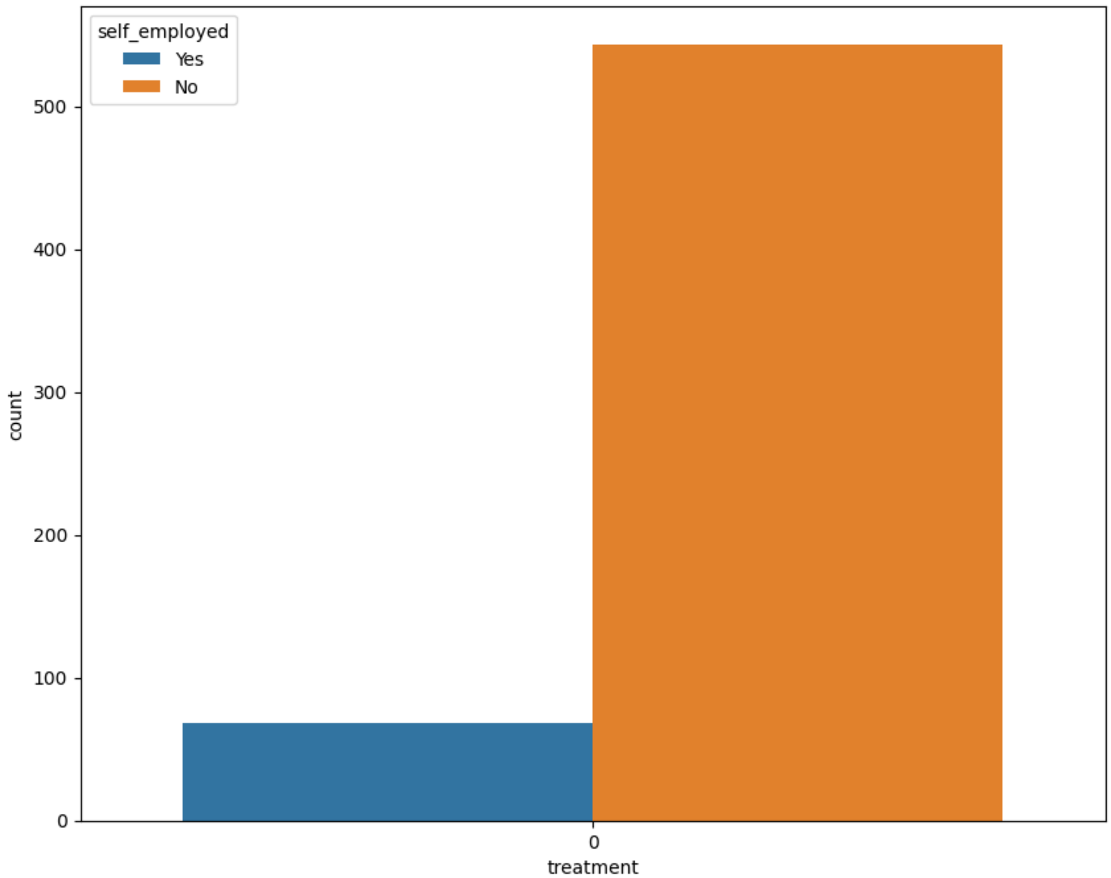


[35]:
```
#Checking employees who require treatment are from which gender
sns.countplot(df.treatment,hue=df.Gender,order=df['treatment'].value_counts().
 ↪iloc[1:2].index)
plt.show()
```



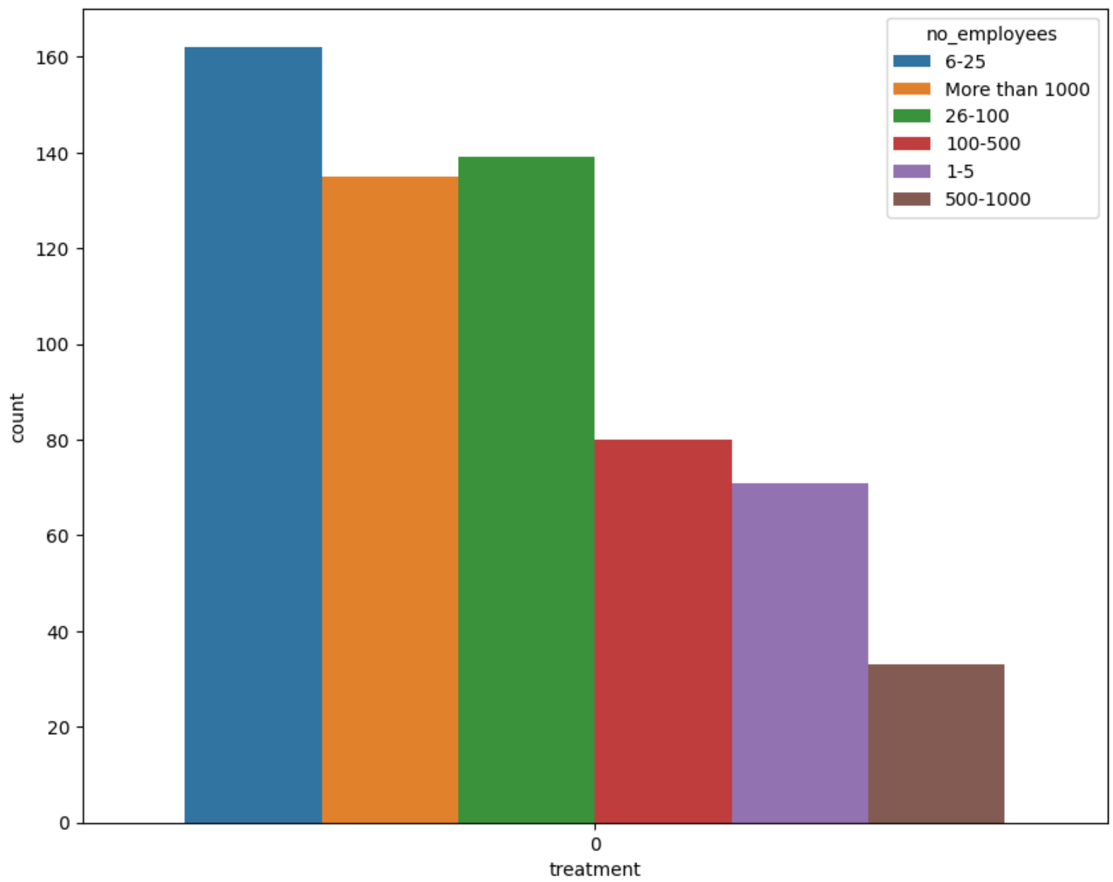
[36]:
```
#plotting countplot to see how many self-employed people requires treatment
sns.countplot(df.treatment,hue=df['self_employed'],order=df['treatment'].
    ↪value_counts().iloc[1:2].index)
```

[36]: <Axes: xlabel='treatment', ylabel='count'>

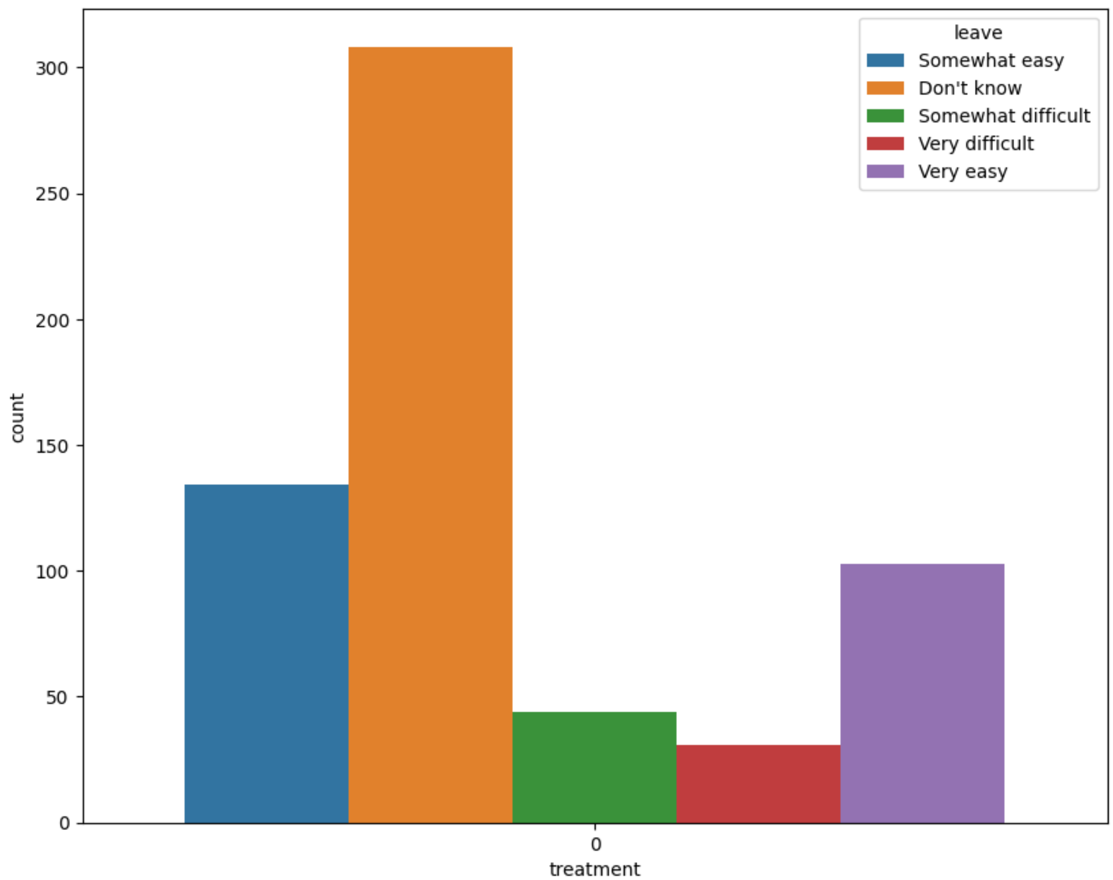


[37]:
```
#checking does number of employees in an organisation affects the treatment rate
sns.countplot(df.treatment,hue=df['no_employees'],order=df['treatment'].
 ↪value_counts().iloc[1:2].index)
```

[37]: <Axes: xlabel='treatment', ylabel='count'>

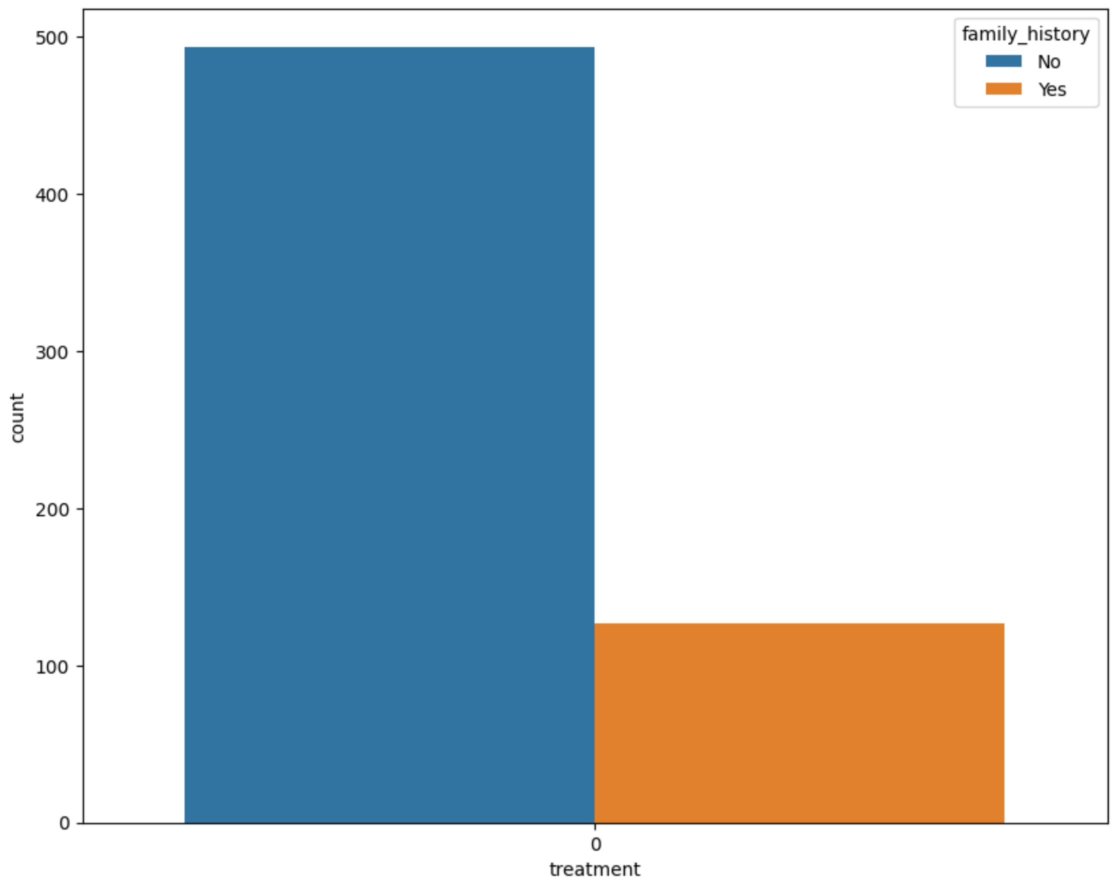


[38]:
```
#plot to see does how easy it to take medical leave and its affect on treatment␣
↪requirement
sns.countplot(df['treatment'],hue=df['leave'],order=df['treatment'].
↪value_counts().iloc[1:2].index)
plt.show()
```

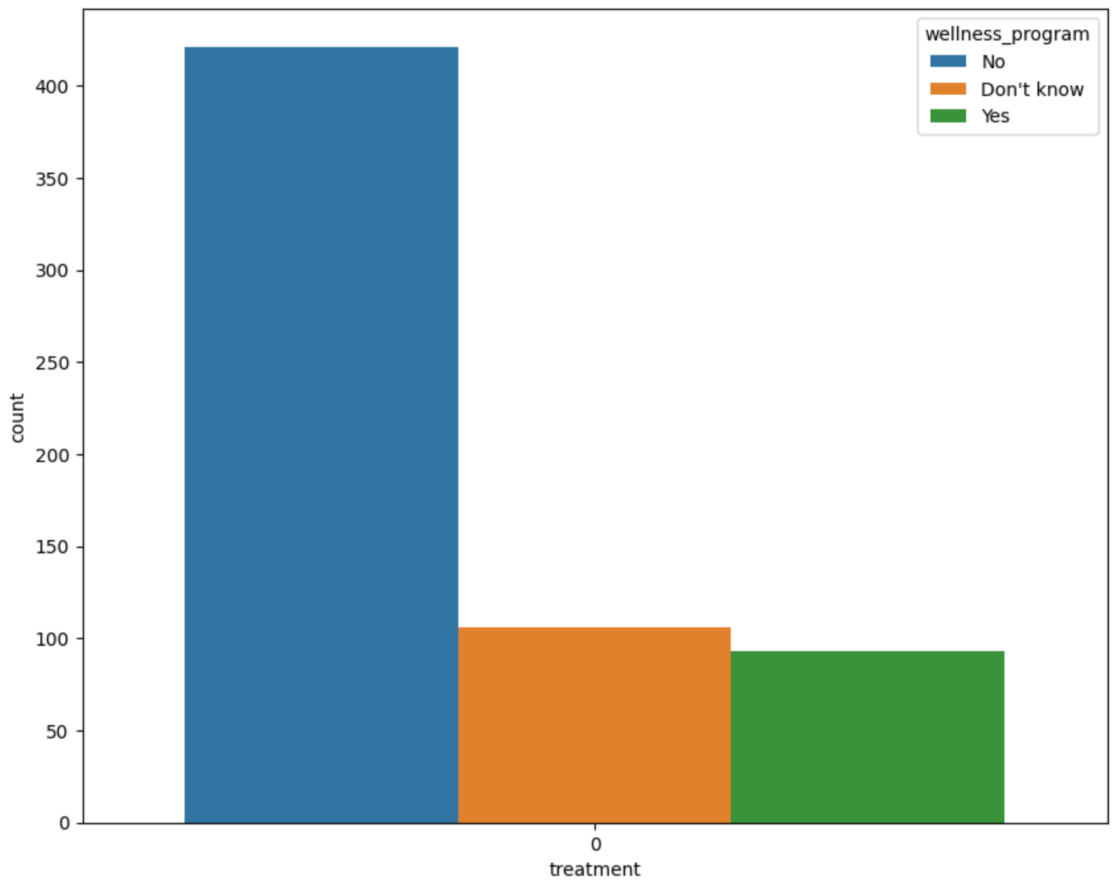


[39]:

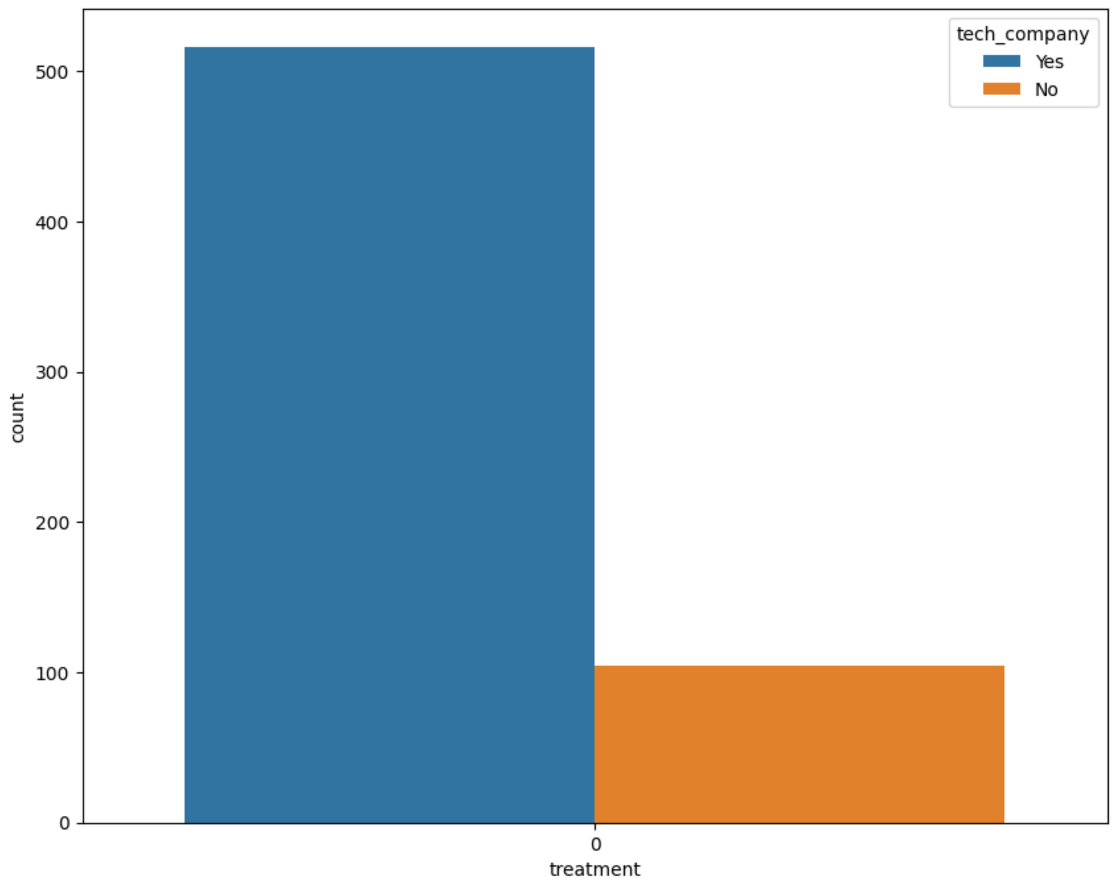
```
sns.countplot(df.treatment,hue=df['family_history'],order=df['treatment'].
  ↪value_counts().iloc[1:2].index)
plt.show()
```



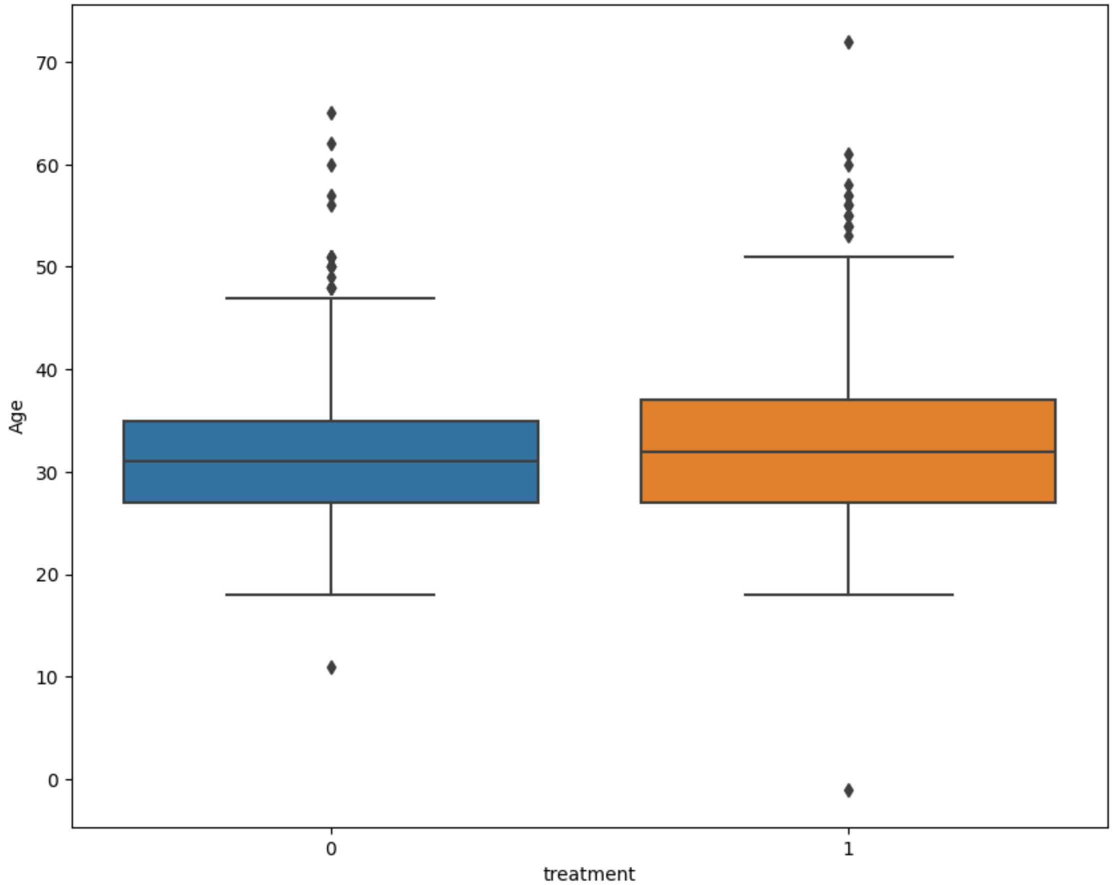
[40]:
```
sns.countplot(df['treatment'],hue=df['wellness_program'],order=df['treatment'].
    ↪value_counts().iloc[1:2].index)
plt.show()
```



[42]:
```
sns.countplot(df['treatment'],hue=df['tech_company'],order=df['treatment'].
  ↪value_counts().iloc[1:2].index)
plt.show()
```



[43]:
```
#plotting the bar plot for age to see if there is any outlier
sns.boxplot(x=df.treatment,y=df['Age'])
plt.show()
```



FINDING THE MISSING VALUES

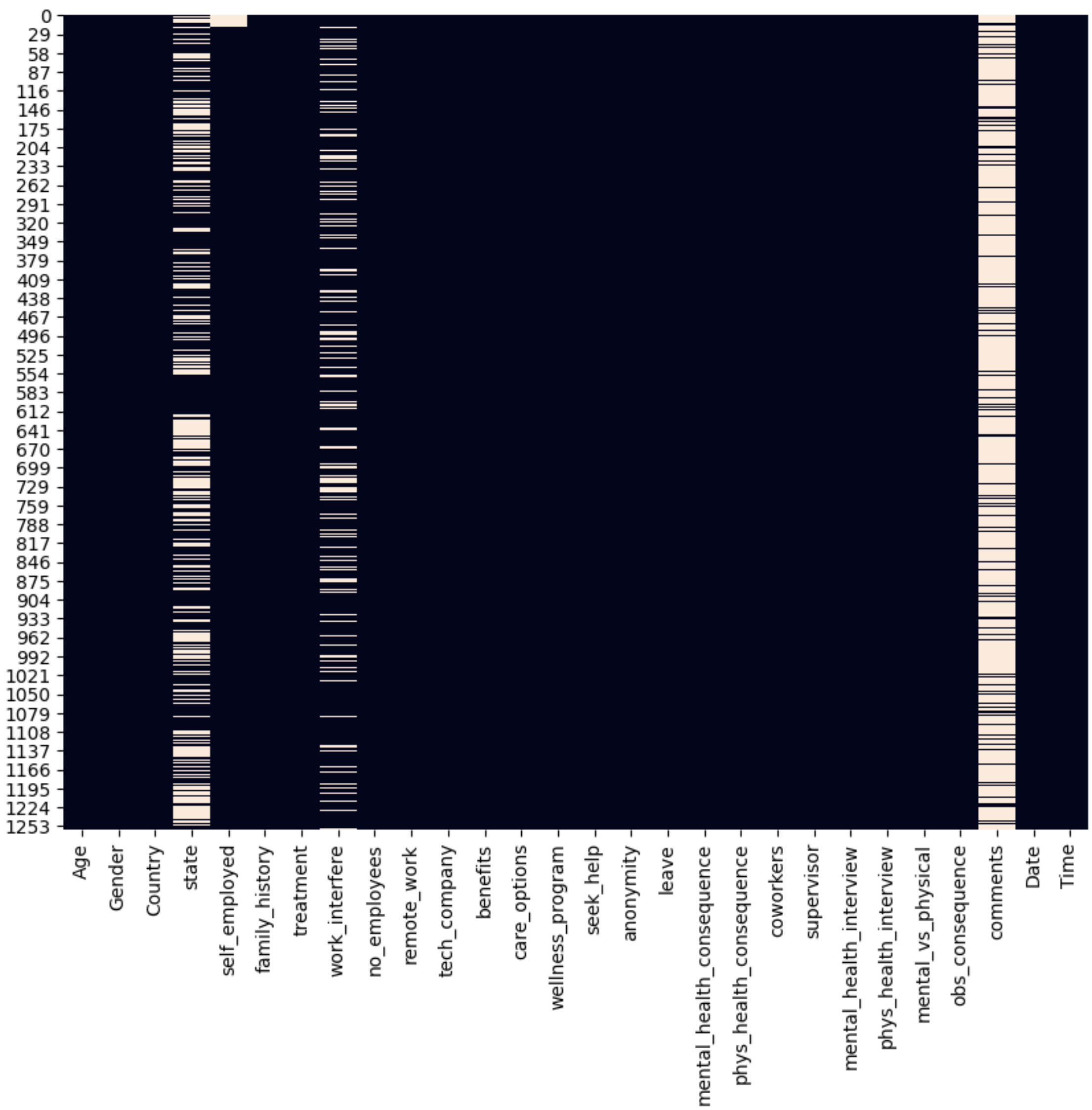
[44]:
```
# sort the variables on the basis of total null values in the variable
# 'isnull().sum()' returns the number of missing values in each variable
Total = df.isnull().sum().sort_values(ascending = False)

# calculate the percentage of missing values
Percent = ((Total*100)/df.isnull().count()).sort_values(ascending = False)

# concat the 'Total' and 'Percent' columns using 'concat' function
missing_data = pd.concat([Total, Percent], axis = 1, keys = ['Total',␣
↪'Percentage of Missing Values'])
missing_data
```

[44]:

	Total	Percentage of Missing Values
comments	1091	87.071030
state	513	40.941740
work_interfere	262	20.909816
self_employed	18	1.436552

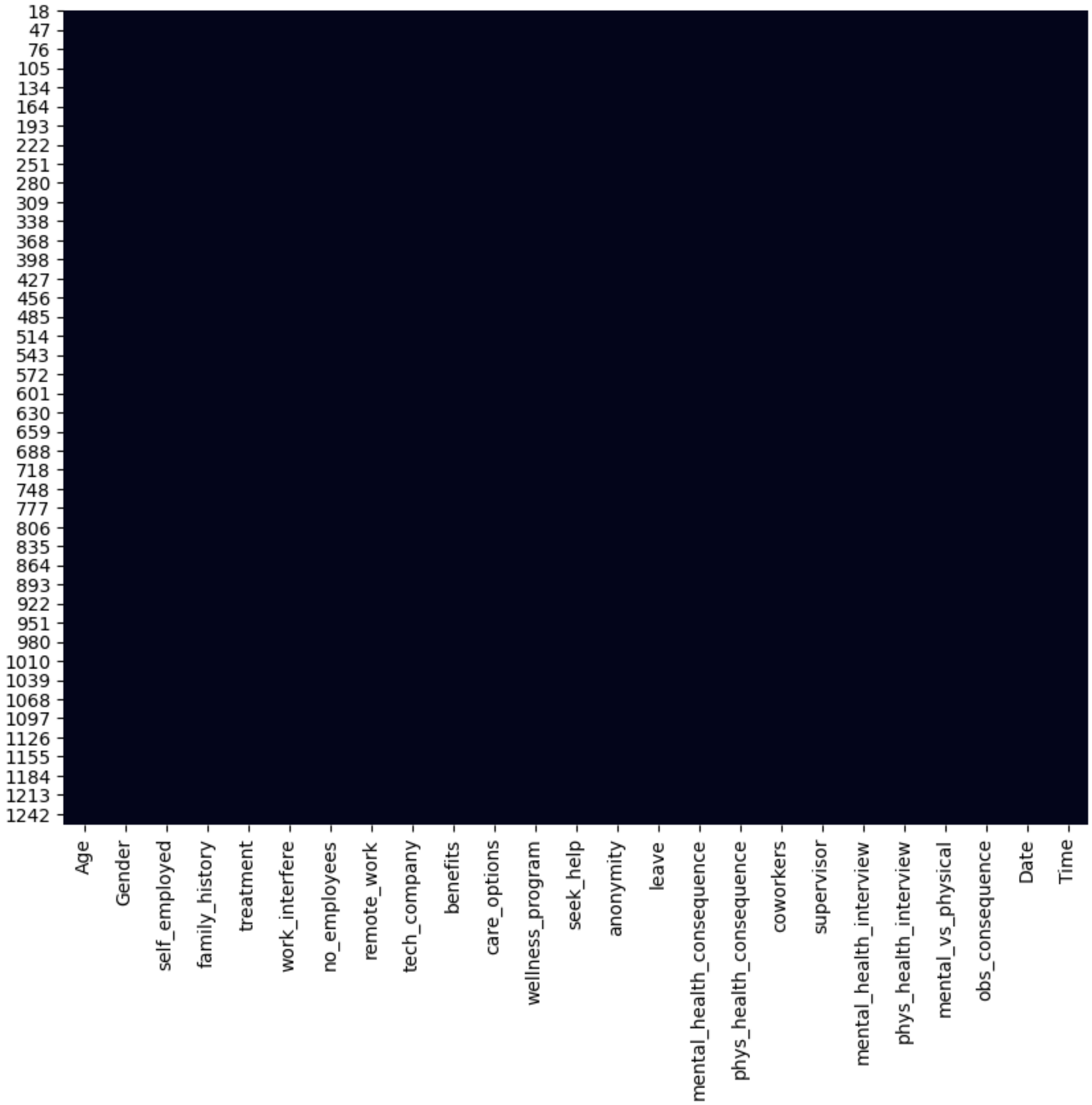
```
Age                          0               0.000000
leave                        0               0.000000
Date                         0               0.000000
obs_consequence              0               0.000000
mental_vs_physical           0               0.000000
phys_health_interview        0               0.000000
mental_health_interview      0               0.000000
supervisor                   0               0.000000
coworkers                    0               0.000000
phys_health_consequence      0               0.000000
mental_health_consequence    0               0.000000
seek_help                    0               0.000000
anonymity                    0               0.000000
Gender                       0               0.000000
wellness_program             0               0.000000
care_options                 0               0.000000
benefits                     0               0.000000
tech_company                 0               0.000000
remote_work                  0               0.000000
no_employees                 0               0.000000
treatment                    0               0.000000
family_history               0               0.000000
Country                      0               0.000000
Time                         0               0.000000
```

[45]:
```python
# plot heatmap to check null values
# 'cbar = False' does not show the color axis
sns.heatmap(df.isnull(), cbar=False)

# display the plot
plt.show()
```



HANDLING THE MISSING VALUES

[46]:
```
df.drop('comments',axis=1,inplace=True)
df['work_interfere']=df['work_interfere'].fillna('Not mentioned')
df.drop(['state','Country'],axis=1,inplace=True)
df.dropna(axis=0, inplace=True)
#Checking if all the null values have been handled or not
sns.heatmap(df.isnull(),cbar=False,color='black')
plt.show()
```



PREPARING THE DATA FOR BUILDING MODEL

[47]:

```
#Creating two dataframes df_features and df_target,df_features contains all the important features which we will dummy encode
#df_target which contains the target variable
df_features=df.drop(['treatment', 'Age','Date', 'Time'],axis=1)
df_target=df['treatment']
#dummy encoding the feature(categorical) variables
df_dummy=pd.get_dummies(df_features,drop_first=True)
#storing the features in X and the target in y variable
X=df_dummy
y=pd.DataFrame(df_target)
```

CREATING GENERALISED FUNCTIONS

```
[48]: # create a generalized function to calculate the metrics values for test set
def get_test_report(model):

    # return the performace measures on test set
    return(classification_report(y_test, y_pred))
# create a generalized function to calculate the metrics values for test set
def kappa_score(model):

    # return the kappa score on test set
    return(cohen_kappa_score(y_test, y_pred))
# define a to plot a confusion matrix for the model
def plot_confusion_matrix(model):

    # create a confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    conf_matrix = pd.DataFrame(data = cm,columns = ['Predicted:0','Predicted:
 ↪1'], index = ['Actual:0','Actual:1'])

    # plot a heatmap to visualize the confusion matrix
    sns.heatmap(conf_matrix, annot = True, fmt = 'd', cmap =␣
 ↪ListedColormap(['lightskyblue']), cbar = False,
                linewidths = 0.1, annot_kws = {'size':25})

    # set the font size of x-axis ticks using 'fontsize'
    plt.xticks(fontsize = 20)

    # set the font size of y-axis ticks using 'fontsize'
    plt.yticks(fontsize = 20)

    # display the plot
    plt.show()
```

```
[51]: # define a function to plot the ROC curve and print the ROC-AUC score
def plot_roc(model):

    # the roc_curve() returns the values for false positive rate, true positive␣
 ↪rate and threshold
    fpr, tpr, thresholds = roc_curve(y_test, y_pred)

    # plot the ROC curve
    plt.plot(fpr, tpr)

    # set limits for x and y axes
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.0])
```

```
    # plot the straight line showing worst prediction for the model
    plt.plot([0, 1], [0, 1],'r--')

    # add plot and axes labels
    # set text size using 'fontsize'
    plt.title('ROC Curve', fontsize = 15)
    plt.xlabel('False positive rate (1-Specificity)', fontsize = 15)
    plt.ylabel('True positive rate (Sensitivity)', fontsize = 15)

    # add the AUC score to the plot
    plt.text(x = 0.02, y = 0.9, s = ('AUC Score:',round(roc_auc_score(y_test,␣
 ↪y_pred),4)))

    # plot the grid
    plt.grid(True)
```

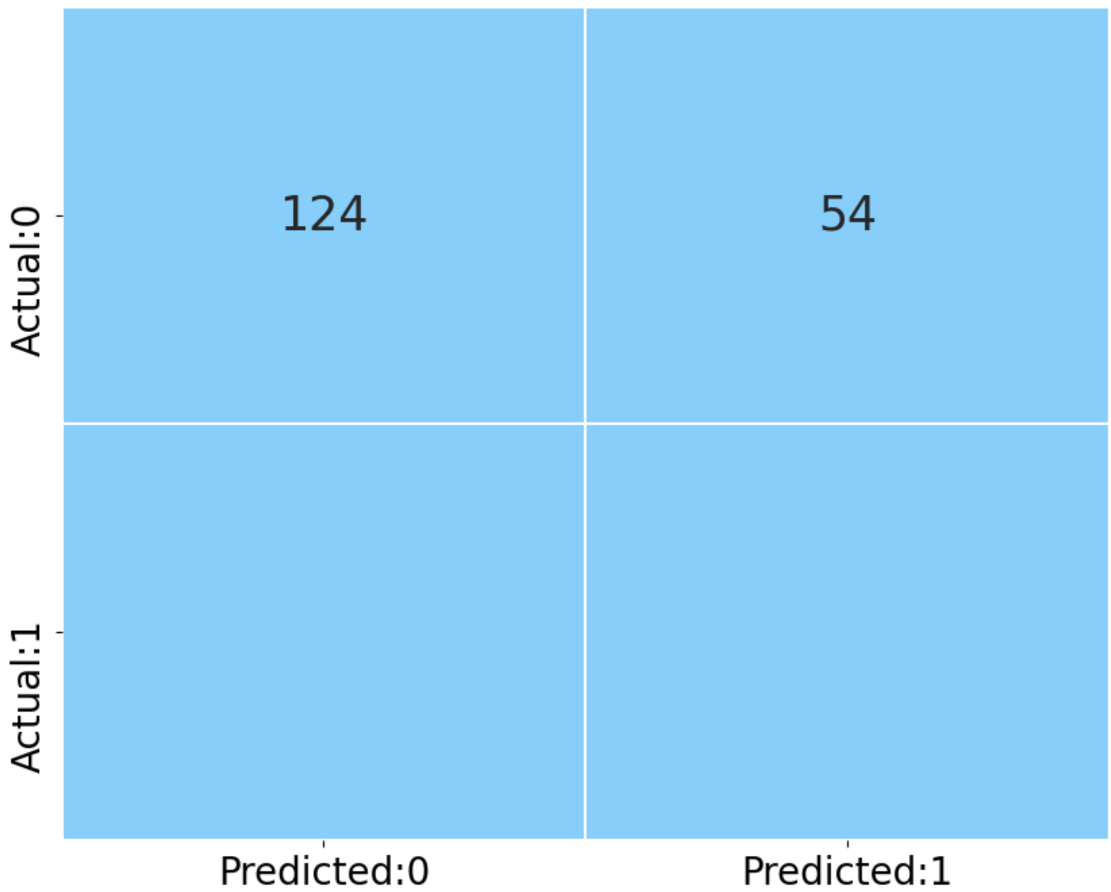
[52]:
```
# create an empty dataframe to store the scores for various classification␣
 ↪algorithms
score_card = pd.DataFrame(columns=['Model', 'AUC Score', 'Precision Score',␣
 ↪'Recall Score', 'Accuracy Score',
                                   'Kappa Score', 'f1-score'])

def update_score_card(model_name):

    # assign 'score_card' as global variable
    global score_card

    # append the results to the dataframe 'score_card'
    # 'ignore_index = True' do not consider the index labels
    score_card = score_card.append({'Model': model_name,
                                    'AUC Score' : roc_auc_score(y_test, y_pred),
                                    'Precision Score': metrics.
 ↪precision_score(y_test, y_pred),
                                    'Recall Score': metrics.
 ↪recall_score(y_test, y_pred),
                                    'Accuracy Score': metrics.
 ↪accuracy_score(y_test, y_pred),
                                    'Kappa Score': cohen_kappa_score(y_test,␣
 ↪y_pred),
                                    'f1-score': metrics.f1_score(y_test,␣
 ↪y_pred)},
                                    ignore_index = True)
    return(score_card)
```

[53]:
```
# split data into train subset and test subset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30,␣
↪random_state = 10)

# check the dimensions of the train & test subset using 'shape'
# print dimension of train set
print("X_train",X_train.shape)
print("y_train",y_train.shape)

# print dimension of test set
print("X_test",X_test.shape)
print("y_test",y_test.shape)
```

```
X_train (864, 48)
y_train (864, 1)
X_test (371, 48)
y_test (371, 1)
```

SUPPORT VECTOR MACHINE

[55]:
```
# build the model
svclassifier = SVC(kernel = 'linear')

# fit the model
svc_model=svclassifier.fit(X_train, y_train)
# predict the values
y_pred = svclassifier.predict(X_test)
# call the function to plot the confusion matrix
plot_confusion_matrix(svc_model)
```



[56]:

```python
# compute the performance measures on test data
test_report = get_test_report(svc_model)

# print the performace measures
print(test_report)
```

```
              precision    recall  f1-score   support

           0       0.97      0.70      0.81       178
           1       0.78      0.98      0.87       193

    accuracy                           0.84       371
   macro avg       0.87      0.84      0.84       371
weighted avg       0.87      0.84      0.84       371
```

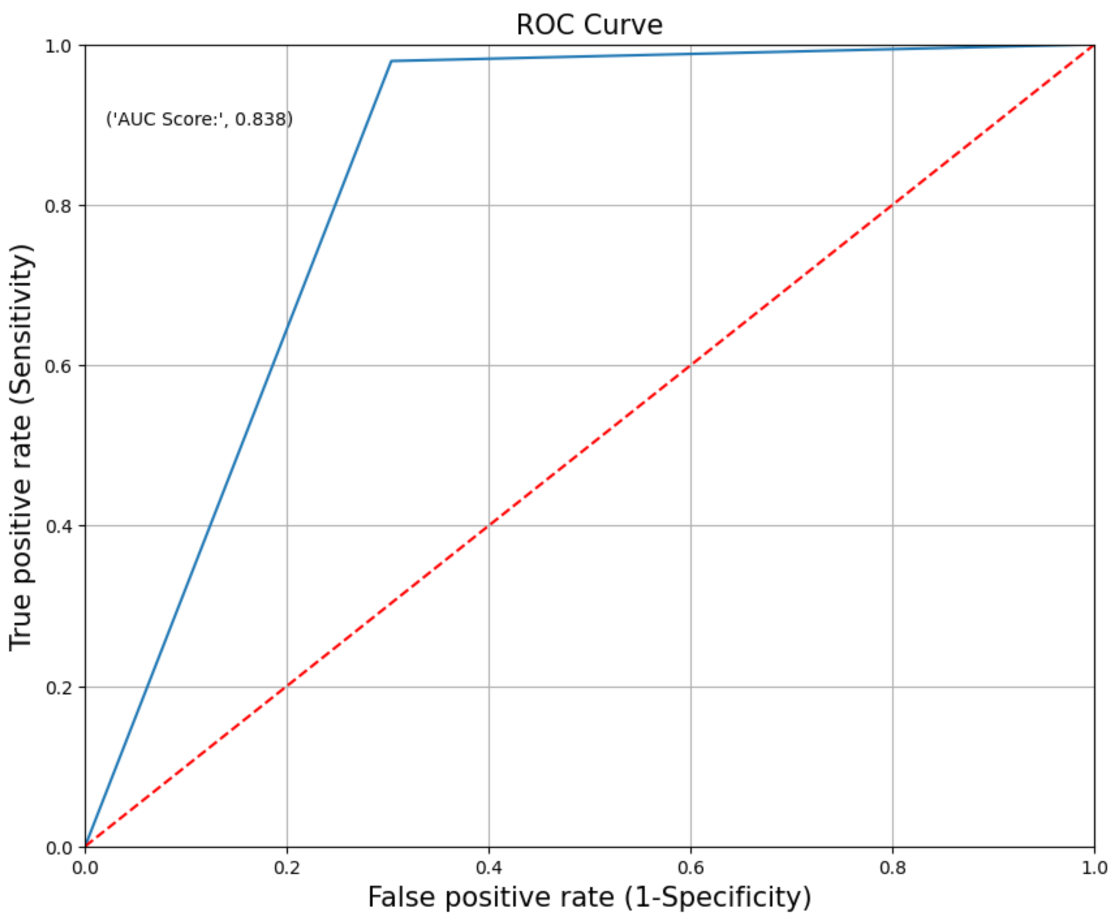
[57]:

```python
# compute kappa score on test set
kappa_value = kappa_score(svc_model)
```

```python
# print the kappa value
print(kappa_value)
```

```
0.6833632537743901
```

[71]:
```python
plot_roc(svc_model)
```



ACCURACY =O.84

[ ]: