DSC550 - Term Project - sidbhaumik

June 3, 2023

0.0.1 Mental Health in Tech Industry

Tech industry is considered as one of the most sought after place for students and professionals due to various reasons like great salary and perks, big offices, sometimes fancy work environment, possible to climb up the management ladder in fairly short time, etc.

Tech is one of the fastest growing industry where people want answers even before they could formulate their problems. Companies are trying to stay one step ahead of their competitors to stay profitable in the Fiercely competitive or inovative market. With the fast-paced environment comes a lot of pressure to deliver quality production ready deliverables. In addition with the advent of new methodologies such as Agile, Peer programming and XP (Extreme Programming) workers are now under more stress than ever before to perform and deliver.

According to OSMI data, 51% of tech professionals have been diagnosed with a mental health condition. By comparison, 19.1% of U.S. adults experience mental illness, according to the National Alliance on Mental Illness.

The terrifying problem with mental illness is that it is invisible; it's a private battle that people have, and it's hard to know when people need help.

The dataset I have selected is from a 2014 survey posted in Kaggle that measures attitudes towards mental health and frequency of mental health disorders in the tech workplace.

Content of this dataset:

1. Timestamp
2. Age
3. Gender
4. Country
5. state: If you live in the United States, which state or territory do you live in?
6. self_employed: Are you self-employed?
7. family_history: Do you have a family history of mental illness?
8. treatment: Have you sought treatment for a mental health condition?
9. work_interfere: If you have a mental health condition, do you feel that it interferes with your work?
10. no_employees: How many employees does your company or organization have?
11. remote_work: Do you work remotely (outside of an office) at least 50% of the time?
12. tech_company: Is your employer primarily a tech company/organization?
13. benefits: Does your employer provide mental health benefits?
14. care_options: Do you know the options for mental health care your employer provides?
15. wellness_program: Has your employer ever discussed mental health as part of an employee wellness program?

16. seek_help: Does your employer provide resources to learn more about mental health issues and how to seek help?
17. anonymity: Is your anonymity protected if you choose to take advantage of mental health or substance abuse treatment resources?
18. leave: How easy is it for you to take medical leave for a mental health condition?
19. mental_health_consequence: Do you think that discussing a mental health issue with your employer would have negative consequences?
20. phys_health_consequence: Do you think that discussing a physical health issue with your employer would have negative consequences?
21. coworkers: Would you be willing to discuss a mental health issue with your coworkers?
22. supervisor: Would you be willing to discuss a mental health issue with your direct supervisor(s)?
23. mental_health_interview: Would you bring up a mental health issue with a potential employer in an interview?
24. phys_health_interview: Would you bring up a physical health issue with a potential employer in an interview?
25. mental_vs_physical: Do you feel that your employer takes mental health as seriously as physical health?
26. obs_consequence: Have you heard of or observed negative consequences for coworkers with mental health conditions in your workplace?
27. comments: Any additional notes or comments

My target variable here is 'treatment'. Workplaces which promote mental health and support their employees through different benefits and wellness programs will see more people opting for treatments and other kind of help if needed. Also, such companies will benefit as well if the employees feel cared for, feel happy as that will improve their work quality and their likelihood to stay with the employer. This will help companies retain employees specially the good one's as well as make them preferred choice when a job applicant decides on a specific company on factors other than position and salary offered.Also, this will improve their ratings in job sites like Glassdoor, Indeed, etc.

[1]:
```python
# Import necessary libraries
import numpy as np # linear algebra
import pandas as pd # data processing
# Libraries for Data Visualization
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import seaborn as sns
from scipy import stats
from scipy.stats import randint
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```python
# Reading Mental Health Survey CSV file and writing to the Dataframe
src_df= pd.read_csv("survey.csv")
```

[3]:
```
# Read Sample data from Dataframe
src_df.head()
```

[3]:
```
             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  …  \
0             No       Yes          Often            6-25  …
1             No        No         Rarely  More than 1000  …
2             No        No         Rarely            6-25  …
3            Yes       Yes          Often          26-100  …
4             No        No          Never         100-500  …

               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                     Yes
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                   Yes
3  Some of them         No                   Maybe                 Maybe
4  Some of them        Yes                     Yes                   Yes

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

[5 rows x 27 columns]
```

[4]:
```
# Checking the number of rows and columns in the dataset
print("Number of rows:",src_df.shape[0])
print("Number of columns:",src_df.shape[1])
```

```
Number of rows: 1259
Number of columns: 27
```

[5]:
```
# Checking for columns with missing data
src_df.isnull().sum().sort_values(ascending=False)
```

[5]:
```
comments                     1095
state                         515
work_interfere                264
self_employed                  18
seek_help                       0
obs_consequence                 0
mental_vs_physical              0
phys_health_interview           0
mental_health_interview         0
supervisor                      0
coworkers                       0
phys_health_consequence         0
mental_health_consequence       0
leave                           0
anonymity                       0
Timestamp                       0
wellness_program                0
Age                             0
benefits                        0
tech_company                    0
remote_work                     0
no_employees                    0
treatment                       0
family_history                  0
Country                         0
Gender                          0
care_options                    0
dtype: int64
```

Data Cleansing

[6]:
```
## Dropping column either due to missing data or I dont find it useful
src_df.drop(columns=["comments","Timestamp","self_employed"],inplace=True)

### COmments has mostly missing values.So, need to remove it.
### Survey timstamp doesnt have any value since I want to look at 2014 stats at␣
↪a high level.
### Same is applicable for self_employed column. I am Specifically looking at␣
↪mental health issues/challenges for employees with a Tech organization.
```

[7]:
```
## Putting default values for other columns with missing values
src_df["state"].fillna("Others",inplace=True)
```

[8]:
```
src_df["work_interfere"].fillna(src_df["work_interfere"].mode()[0],inplace=True)
```

[9]:
```python
# Checking Unique data for Age column
src_df["Age"].unique()
```

[9]:
```
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])
```

[10]:
```python
# I see multiple invalid Age's which I will drop and store cleansed data in a␣
↪new dataframe
src_clean_df = src_df[(src_df['Age'] > 0) & (src_df['Age'] <= 100)]
```

[11]:
```python
# Checking cleaned Age data
src_clean_df["Age"].unique()
```

[11]:
```
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, 43, 56, 60, 54, 55, 48,
       20, 57, 58, 47, 62, 51, 65, 49,  5, 53, 61,  8, 11, 72])
```

[12]:
```python
# Checking Unique data for Gender column
src_clean_df["Gender"].unique()
```

[12]:
```
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', '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', 'femail', 'Cis Man',
       'ostensibly male, unsure what that really means'], dtype=object)
```

[13]:
```python
# Gender data is messy and need to be cleansed and standardized
src_clean_df["Gender"]=src_clean_df["Gender"].replace("f","female")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("m","male")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("Female","female")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("Male","male")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("F","female")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("M","male")
```

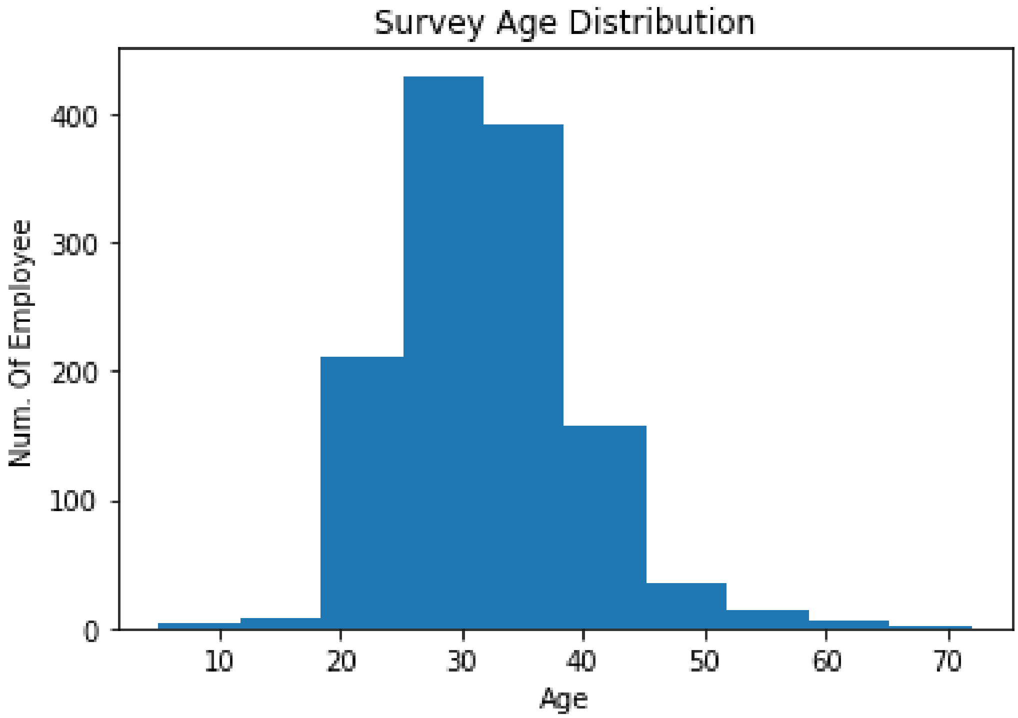
```
src_clean_df["Gender"]=src_clean_df["Gender"].replace("maile","male")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("Male-ish","male")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("women","female")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("Women","female")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("women","female")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("Mail","male")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("Man","male")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("Make","male")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("Cis Female","female")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("Cis Male","male")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("Male (CIS)","male")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("Female (cis)","female")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("Mal","male")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("Femake","female")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("woman","female")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("cis male","male")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("Cis Man","male")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("femail","female")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("Female ","female")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("Male ","male")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("msle","male")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("Malr","male")
src_clean_df["Gender"]=src_clean_df["Gender"].replace("Woman","female")
```

[14]:
```
src_clean_df['Gender'] = np.where((src_clean_df['Gender'] != 'female') &␣
↪(src_clean_df['Gender'] != 'male'), 'Other', src_clean_df['Gender'])
```

[15]:
```
src_clean_df["Gender"].unique()
```

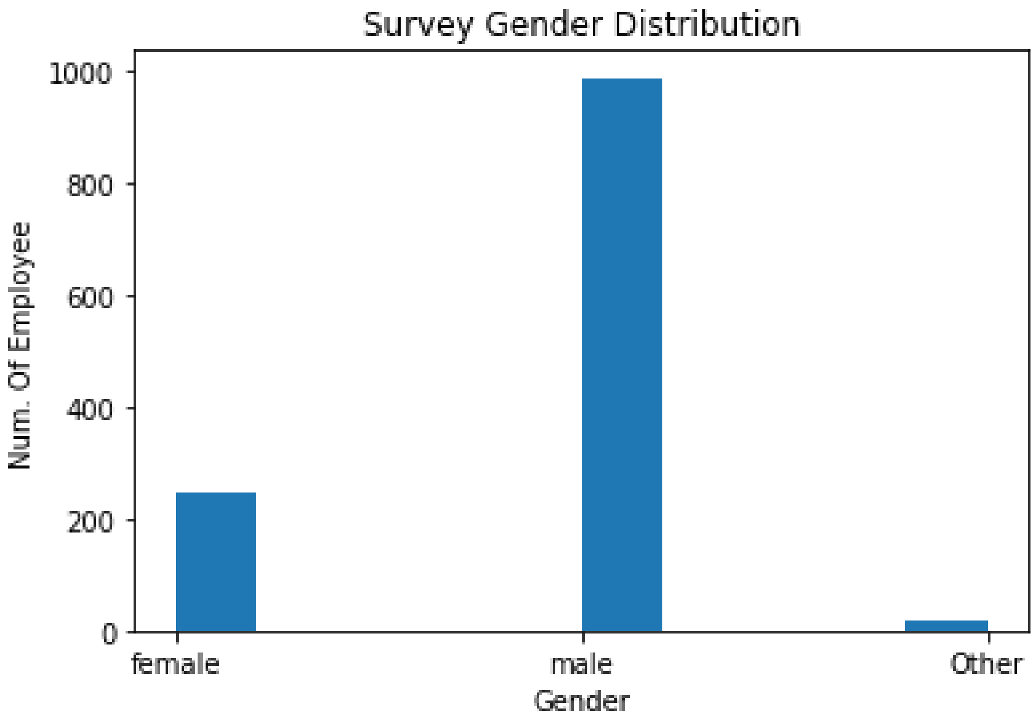
[15]:
```
array(['female', 'male', 'Other'], dtype=object)
```

[16]:
```
# Looking at the Age range for people surveyed using Histogram
plt.hist(src_clean_df["Age"])
plt.xlabel("Age")
plt.ylabel("Num. Of Employee")
plt.title('Survey Age Distribution')
plt.show()
```



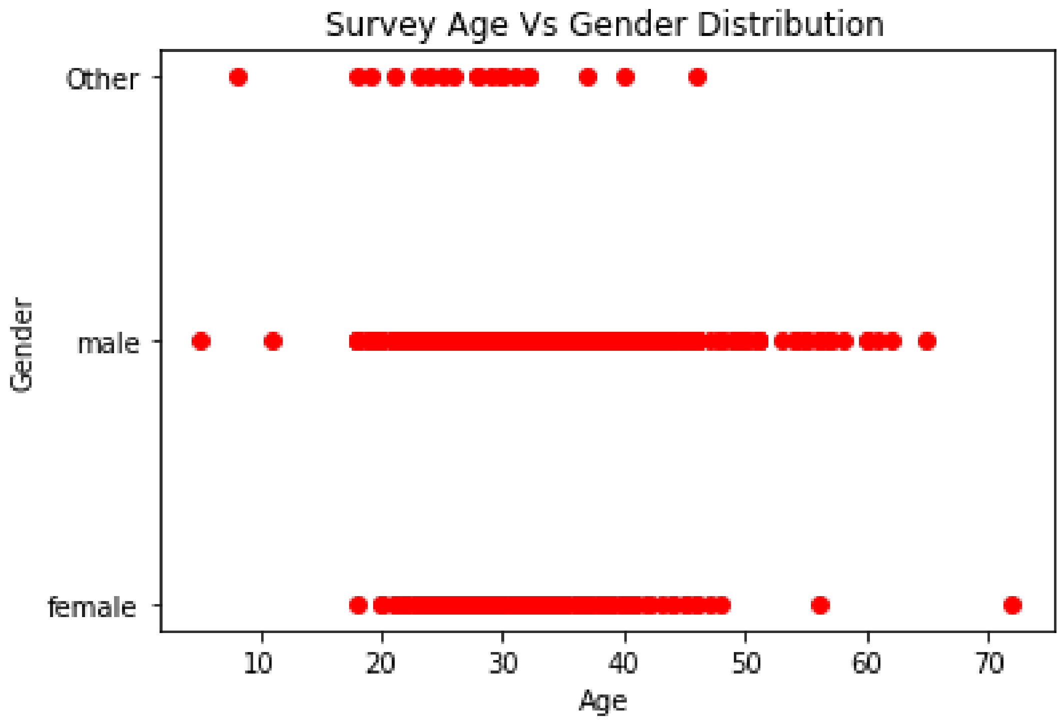
The above plot shows that most of the people participated in the survey are between 25 to 35 Age range. I am interested to see which Age group are most affected with Mental health issues.

[17]:
```
# Looking at the Gender distribution for people surveyed using Histogram
plt.hist(src_clean_df["Gender"])
plt.xlabel("Gender")
plt.ylabel("Num. Of Employee")
plt.title('Survey Gender Distribution')
plt.show()
```



The above plot shows that most of the people participated in the survey are Males. I am interested to see which Gender is most affected with Mental health issues.

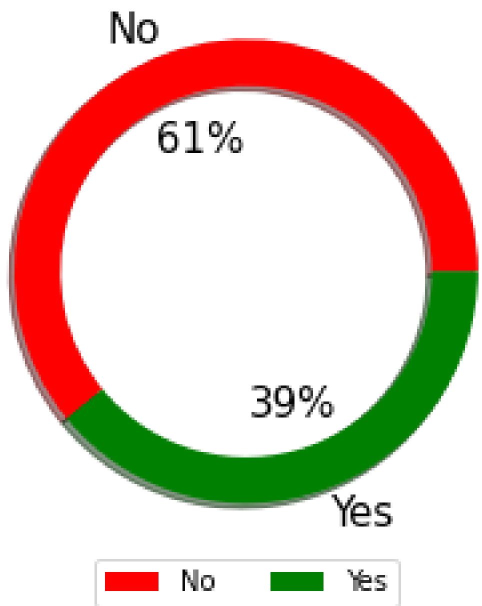
[18]:
```
# Looking at the Age Vs Gender Distribution for people surveyed using Scatter␣
↪Plot
plt.scatter(src_clean_df["Age"], src_clean_df["Gender"],c="red")
plt.xlabel("Age")
plt.ylabel("Gender")
plt.title('Survey Age Vs Gender Distribution')
plt.show()
```



The above plot shows the same as previous plots. That more number of Male employees within 25 - 35 age range participated in the survey.

[19]:
```
# Looking for employee with family history of mental illness
fh=src_clean_df["family_history"].value_counts()
print(fh)

plt.pie(fh,labels=fh.index,autopct="%0.0f%%",textprops={"fontsize":
 ↪15},wedgeprops={'width': 0.20},shadow=True,colors="rg",explode=[0,0]);
plt.legend(loc='lower center', bbox_to_anchor=(0.5, -0.1),ncol=3);
plt.show()
```

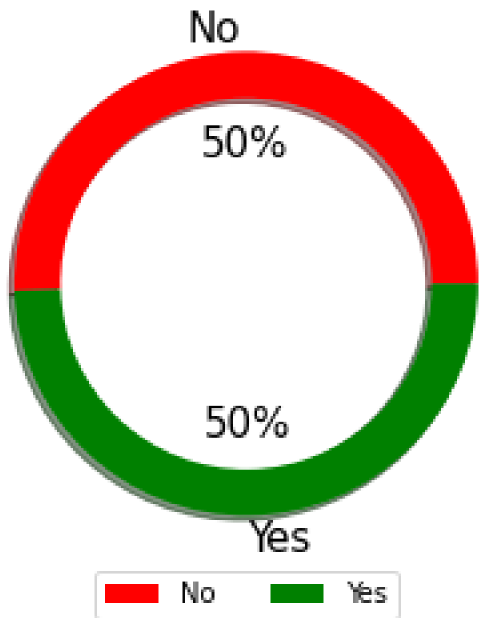
```
No      764
Yes     490
Name: family_history, dtype: int64
```



I believe people with family history of mental illness will be more aware of this issue and will be eager to seek help if need arise.From the survey, I see 39% of employee has some family history of mental illness.

[20]:
```
# Looking for employee who have taken some treatment for Mental Illness
tr=src_clean_df["treatment"].value_counts()
print(tr)

plt.pie(tr,labels=fh.index,autopct="%0.0f%%",textprops={"fontsize":
 ↪15},wedgeprops={'width': 0.20},shadow=True,colors="rg",explode=[0,0]);
plt.legend(loc='lower center', bbox_to_anchor=(0.5, -0.1),ncol=3);
plt.show()
```

```
Yes    633
No     621
Name: treatment, dtype: int64
```



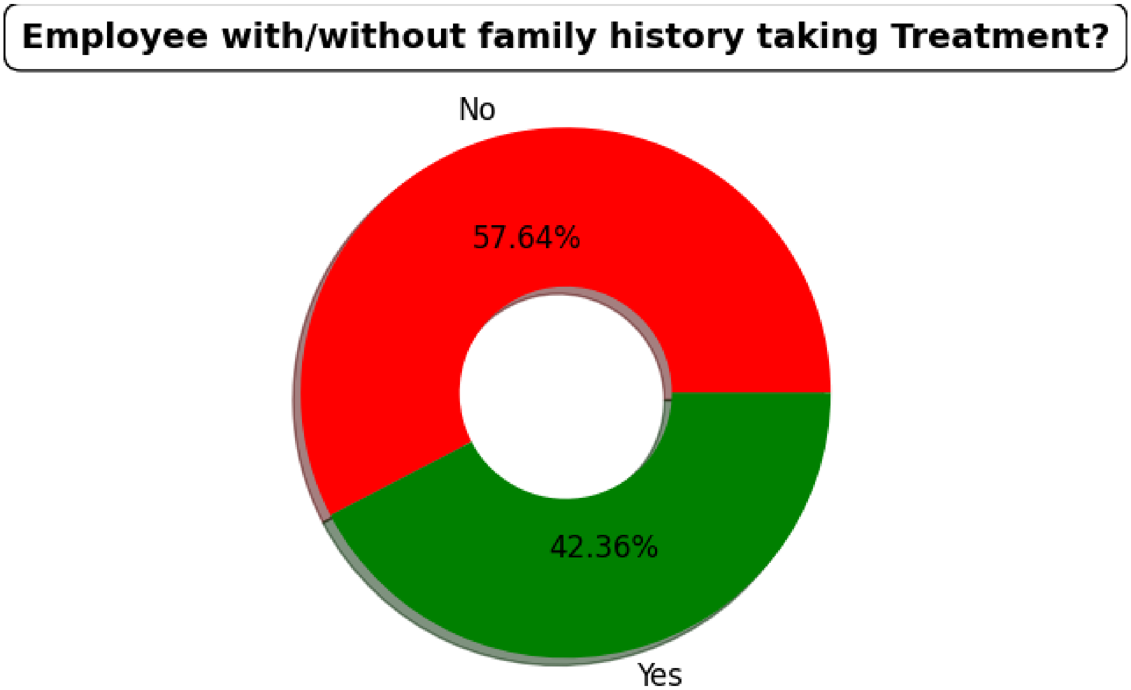
From the survey population, 50% of the employee have taken some form of treatment for Mental health.

[21]:
```
## People who have family history and undergoing any treament
df=src_clean_df[src_clean_df["family_history"]==src_clean_df["treatment"]]
df_new=df[["family_history","treatment"]].value_counts().reset_index()
df_new
```

[21]:

	family_history	treatment	0
0	No	No	494
1	Yes	Yes	363

[22]:
```
plt.figure(figsize=(5,5),dpi=100)
plt.pie(df_new[0],labels=df_new["treatment"],autopct="%0.
↪02f%%",textprops={"fontsize":12},wedgeprops={'width': 0.
↪6},shadow=True,colors="rg")
plt.title("Employee with/without family history taking Treatment?",fontsize=14,␣
↪fontweight='bold', bbox=dict(facecolor='white', edgecolor='black',␣
↪boxstyle='round,pad=0.5'));
```

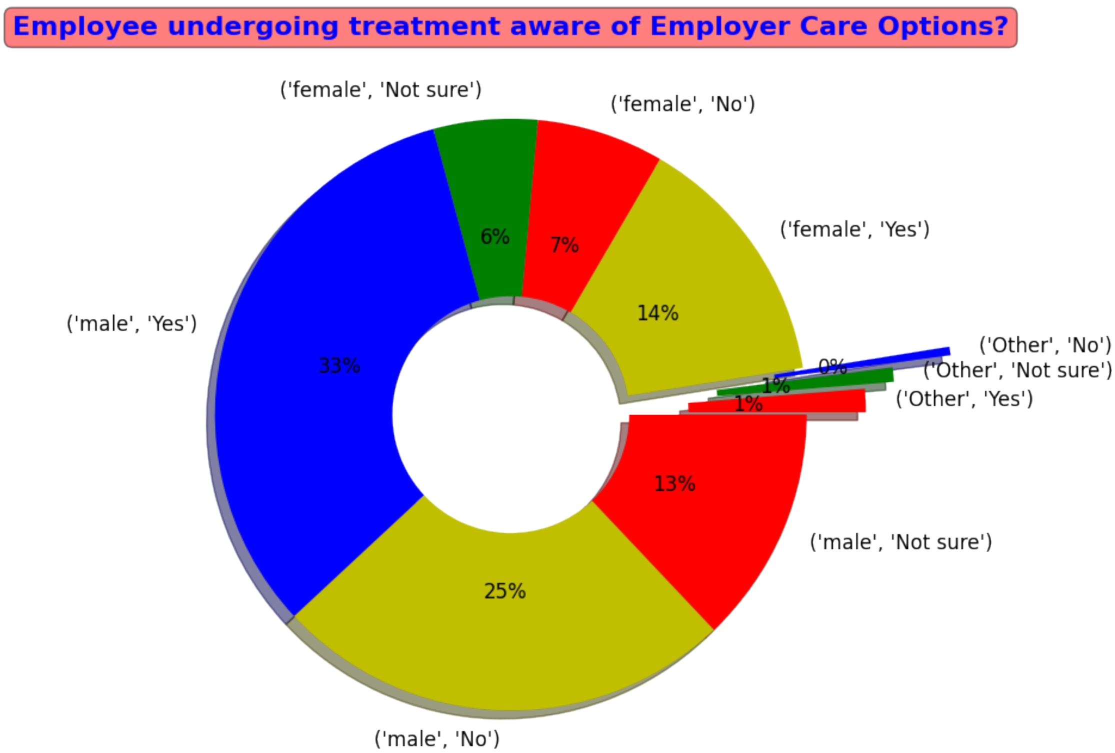


Around 42% employee with some family history of Mental health issue are undergoing some form of treatment.

[23]:
```
## The people who are undergoing treatment Do options for mental health care␣
↪your employer provides?
emp_trt=src_clean_df[src_clean_df["treatment"]=="Yes"]

co=emp_trt.groupby("Gender")["care_options"].value_counts()
print(co)
plt.figure(figsize=(8,20),dpi=100)
plt.pie(co,labels=emp_trt.groupby("Gender")["care_options"].value_counts().
↪index,autopct="%0.0f%%",textprops={"fontsize":12},wedgeprops={'width': 0.
↪6},shadow=True,colors="rgby",explode=[0.20,0.30,0.50,0,0,0,0,0,0])
plt.title('Employee undergoing treatment aware of Employer Care Options?',␣
↪fontdict={'fontsize': 16, 'color': 'blue', 'fontweight': 'bold'},␣
↪bbox=dict(boxstyle='round,pad=0.3', facecolor='red', alpha=0.
↪5,edgecolor='black'));
```

```
Gender  care_options
Other   Yes             8
        Not sure        5
        No              3
female  Yes            89
        No             44
```

```
        Not sure          36
male    Yes              207
        No               159
        Not sure          82
Name: care_options, dtype: int64
```



33% of the male who were undergoing treatment say they are aware of care options being provided to them by the Employer.

By initial exploration of the dataset, I selected below listed variables which I thought will be useful and influence the target variable 'Treatment': Age, Gender, family_history, benefits, care_options.

My assumption that 'Age' and 'Gender' may play a significant part was not correct. But I see family_history and care_options have good influence over employees opting for treatment. I will continue my exploration for other variables to see if any other variable has influence over people opting for treatment.

[24]:
```
src_clean_df.head()
```

[24]:

```
   Age  Gender         Country   state family_history treatment  \
0   37  female   United States      IL             No       Yes
1   44    male   United States      IN             No        No
2   32    male          Canada  Others             No        No
3   31    male  United Kingdom  Others            Yes       Yes
```

```
4   31    male   United States      TX             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                     Yes
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                   Yes
3  Some of them         No                   Maybe                 Maybe
4  Some of them        Yes                     Yes                   Yes

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

[5 rows x 24 columns]
```

[25]:
```python
# Making sure there are no columns with missing data
src_clean_df.isnull().sum().sort_values(ascending=False)
```

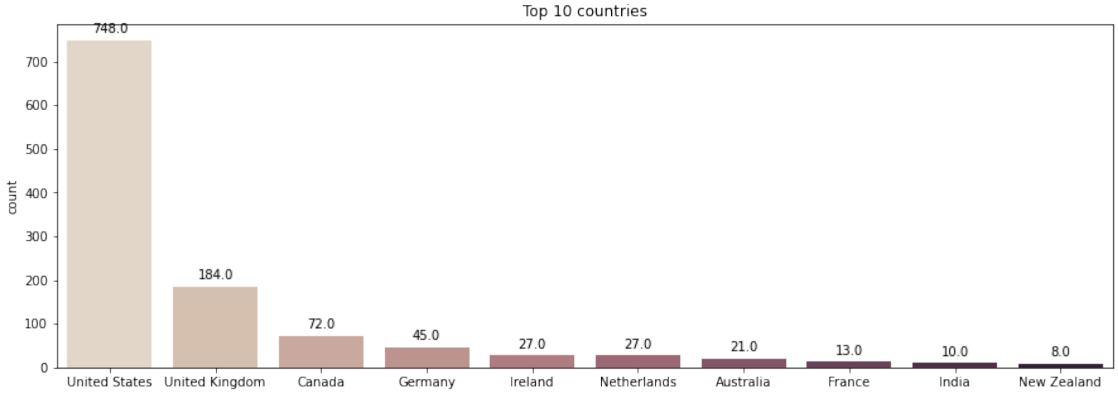
[25]:
```
Age                          0
Gender                       0
mental_vs_physical           0
phys_health_interview        0
mental_health_interview      0
supervisor                   0
coworkers                    0
phys_health_consequence      0
mental_health_consequence    0
leave                        0
anonymity                    0
seek_help                    0
```

```
wellness_program          0
care_options              0
benefits                  0
tech_company              0
remote_work               0
no_employees              0
work_interfere            0
treatment                 0
family_history            0
state                     0
Country                   0
obs_consequence           0
dtype: int64
```

[26]:
```
# Checking Unique data for Country column
src_clean_df["Country"].unique()
```

[26]:
```
array(['United States', 'Canada', 'United Kingdom', 'Bulgaria', 'France',
       'Portugal', 'Netherlands', 'Switzerland', 'Poland', 'Australia',
       'Germany', 'Russia', 'Mexico', 'Brazil', 'Slovenia', 'Costa Rica',
       'Austria', 'Ireland', 'India', 'South Africa', 'Italy', 'Sweden',
       'Colombia', 'Latvia', 'Romania', 'Belgium', 'New Zealand', 'Spain',
       'Finland', 'Uruguay', 'Israel', 'Bosnia and Herzegovina',
       'Hungary', 'Singapore', 'Japan', 'Nigeria', 'Croatia', 'Norway',
       'Thailand', 'Denmark', 'Bahamas, The', 'Greece', 'Moldova',
       'Georgia', 'China', 'Czech Republic', 'Philippines'], dtype=object)
```

[27]:
```
country_count = src_clean_df.Country.value_counts().
 ↪sort_values(ascending=False).to_frame()[:10]
country_count = country_count.rename(columns={'Country': 'count'})
plt.figure(figsize=(15,5))
ax = sns.barplot(x=country_count.index, y='count', data=country_count, ␣
 ↪palette="ch:.25")
for p in ax.patches:
    ax.annotate(format(p.get_height(), '.1f'),
                   (p.get_x() + p.get_width() / 2., p.get_height()),
                   ha = 'center', va = 'center',
                   xytext = (0, 9),
                   textcoords = 'offset points')
ax = ax.set_title('Top 10 countries')
```



[28]:
```
## Since United States is at the top of list based on the survey and above␣
↪result, I am interested in only exploring United States data. So, removing␣
↪rest of the countries.
src_clean_df = src_clean_df[(src_clean_df['Country'] == 'United States')]
```

[29]:
```
# Making sure I have only United States data now
src_clean_df["Country"].unique()
```

[29]:
```
array(['United States'], dtype=object)
```

[30]:
```
## Dropping state column as I want to see the overall US stats
src_clean_df.drop(columns=["state"],inplace=True)
```

[31]:
```
# Checking the number of rows and columns in the dataset after cleanup
print("Number of rows:",src_clean_df.shape[0])
print("Number of columns:",src_clean_df.shape[1])
```

```
Number of rows: 748
Number of columns: 23
```

[79]:
```
from sklearn.model_selection import␣
↪train_test_split,RandomizedSearchCV,GridSearchCV
from sklearn import preprocessing
from sklearn.datasets import make_classification
from sklearn.preprocessing import binarize, LabelEncoder, MinMaxScaler
from sklearn.svm import SVC
# Models
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier

# Validation libraries
from sklearn import metrics
```

```python
from sklearn.metrics import accuracy_score, mean_squared_error,␣
 ↪precision_recall_curve
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_curve,auc
from sklearn.metrics import f1_score
```

[33]:
```python
## Encoding data to feed the model
labelDict = {}
for feature in src_clean_df:
    le = preprocessing.LabelEncoder()
    le.fit(src_clean_df[feature])
    le_name_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
    src_clean_df[feature] = le.transform(src_clean_df[feature])
    # Get labels
    labelKey = 'label_' + feature
    labelValue = [*le_name_mapping]
    labelDict[labelKey] =labelValue

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

```
label_Age [5, 11, 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, 62, 65, 72]
label_Gender ['Other', 'female', 'male']
label_Country ['United States']
label_family_history ['No', 'Yes']
label_treatment ['No', 'Yes']
label_work_interfere ['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']
```

[34]:
```
src_clean_df.head()
```

[34]:
```
   Age  Gender  Country  family_history  treatment  work_interfere  \
0   21       1        0               0          1               1
1   28       2        0               0          0               2
4   15       2        0               0          0               0
5   17       2        0               1          0               3
6   19       1        0               1          1               3

   no_employees  remote_work  tech_company  benefits  …  anonymity  leave  \
0             4            0             1         2  …          2      2
1             5            0             0         0  …          0      0
4             1            1             1         2  …          0      0
5             4            0             1         2  …          0      0
6             0            1             1         1  …          1      1

   mental_health_consequence  phys_health_consequence  coworkers  supervisor  \
0                          1                        1          1           2
1                          0                        1          0           0
4                          1                        1          1           2
5                          1                        1          2           2
6                          0                        0          1           0

   mental_health_interview  phys_health_interview  mental_vs_physical  \
0                        1                      0                   2
1                        1                      1                   0
4                        2                      2                   0
5                        1                      0                   0
6                        1                      1                   0

   obs_consequence
0                0
1                0
4                0
5                0
6                0

[5 rows x 23 columns]
```

Scaling & Model Fitting

[35]:
```
# Scaling Age because it's completely different from others.
scaler = MinMaxScaler()
src_clean_df['Age'] = scaler.fit_transform(src_clean_df[['Age']])
src_clean_df.head()
```

[35]:
```
        Age  Gender  Country  family_history  treatment  work_interfere  \
0  0.466667       1        0               0          1               1
1  0.622222       2        0               0          0               2
4  0.333333       2        0               0          0               0
5  0.377778       2        0               1          0               3
6  0.422222       1        0               1          1               3

   no_employees  remote_work  tech_company  benefits  …  anonymity  leave  \
0             4            0             1         2  …          2      2
1             5            0             0         0  …          0      0
4             1            1             1         2  …          0      0
5             4            0             1         2  …          0      0
6             0            1             1         1  …          1      1

   mental_health_consequence  phys_health_consequence  coworkers  supervisor  \
0                          1                        1          1           2
1                          0                        1          0           0
4                          1                        1          1           2
5                          1                        1          2           2
6                          0                        0          1           0

   mental_health_interview  phys_health_interview  mental_vs_physical  \
0                        1                      0                   2
1                        1                      1                   0
4                        2                      2                   0
5                        1                      0                   0
6                        1                      1                   0

   obs_consequence
0                0
1                0
4                0
5                0
6                0

[5 rows x 23 columns]
```

[36]:
```
# Creating X containing all the features except target treatment
X = src_clean_df.drop('treatment',axis=1)
```

[37]:
```
# Creating y containing only target class
y = src_clean_df['treatment']
```

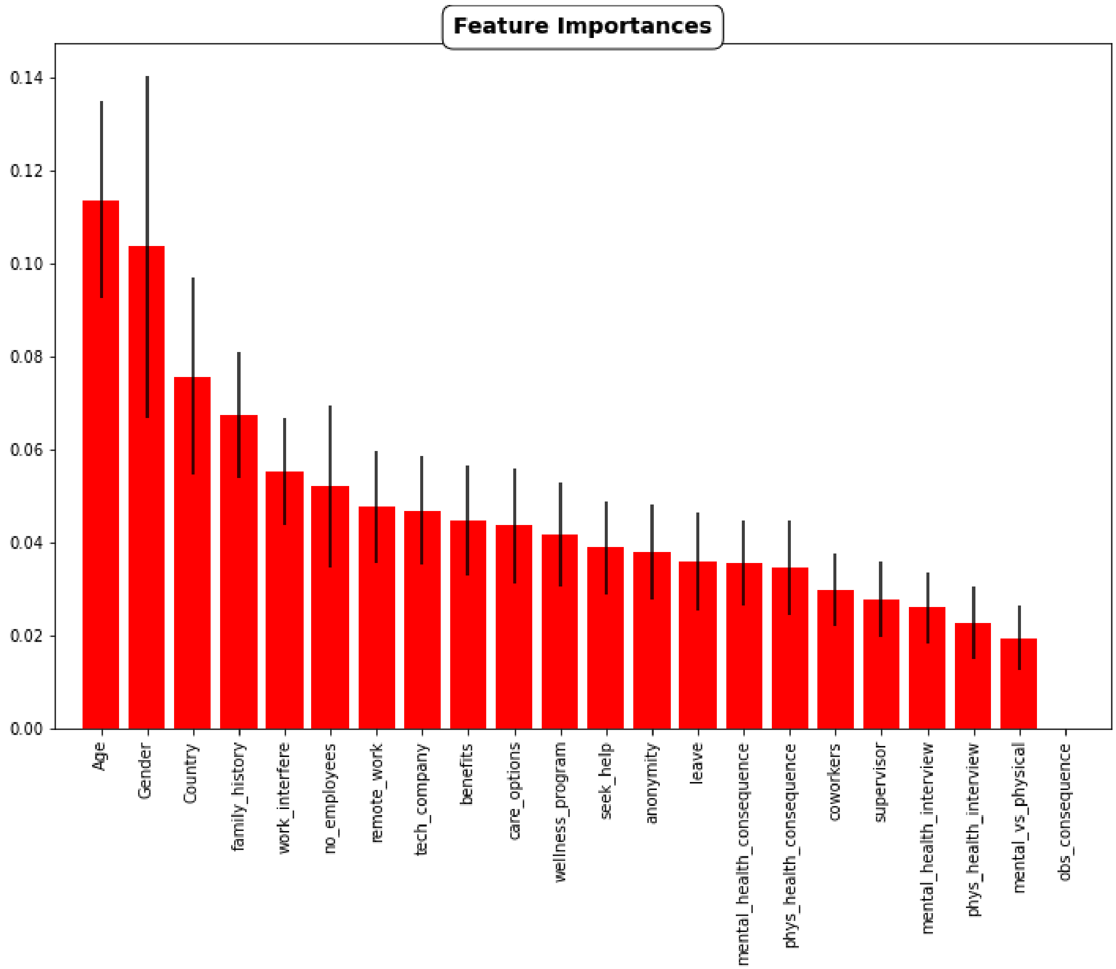
80/20 Train - Test Datasplit

[38]:
```
# Data split between Train & Test
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,␣
↪random_state=42)
```

[39]:
```
# Create dictionaries for final graph
# Use: methodDict['Stacking'] = accuracy_score
methodDict = {}
rmseDict = ()
```

[40]:
```
# Build a forest and compute the feature importances
feature_cols = ['Age', 'Gender', 'Country', 'family_history', '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']

forest = ExtraTreesClassifier(n_estimators=250,
                              random_state=0)

forest.fit(X, y)
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

labels = []
for f in range(X.shape[1]):
    labels.append(feature_cols[f])

# Plot the feature importances of the forest
plt.figure(figsize=(12,8))
plt.title("Feature Importances",fontsize=14, fontweight='bold',␣
↪bbox=dict(facecolor='white', edgecolor='black', boxstyle='round,pad=0.5'))
plt.bar(range(X.shape[1]), importances[indices],
       color="r", yerr=std[indices], align="center")
plt.xticks(range(X.shape[1]), labels, rotation='vertical')
plt.xlim([-1, X.shape[1]])
plt.show()
```



Evaluating a Classification Model. This function will evaluate:

1. Classification accuracy: percentage of correct predictions
2. Null accuracy: accuracy that could be achieved by always predicting the most frequent class
3. Percentage of ones
4. Percentage of zeros
5. Confusion matrix True Positives (TP) True Negatives (TN) False Positives (FP) False Negatives (FN) Falsely predict negative
6. False Positive Rate
7. Precision of Positive value
8. AUC: Is the percentage of the ROC plot that is underneath the curve .90-1 = excellent (A) .80-.90 = good (B) .70-.80 = fair (C) .60-.70 = poor (D) .50-.60 = fail (F)

Logistic Regression

[41]:
```
# instantiate model
logreg = LogisticRegression()
```

```
# fit model
logreg.fit(X_train, y_train)
```

[41]:
```
LogisticRegression()
```

[42]:
```
# make class predictions for the testing set
y_pred_class = logreg.predict(X_test)
```

Classification accuracy: percentage of correct predictions

[43]:
```
# calculate accuracy
print("Classification accuracy is",metrics.accuracy_score(y_test,␣
↪y_pred_class)*100,"%")
```

```
Classification accuracy is 80.0 %
```

Null accuracy: accuracy that could be achieved by always predicting the most frequent class

[44]:
```
# examine the class distribution of the testing set
y_test.value_counts()
```

[44]:
```
0    76
1    74
Name: treatment, dtype: int64
```

[45]:
```
# calculate the percentage of ones
# because y_test only contains ones and zeros, we can simply calculate the mean␣
↪= percentage of ones
y_test.mean()
```

[45]:
```
0.49333333333333335
```

[46]:
```
# calculate the percentage of zeros
1 - y_test.mean()
```

[46]:
```
0.5066666666666666
```

This means thats the model will be right 51% of time. This shows that the classification accuracy is not that good.

[47]:
```
##Comparing the true and predicted response values

# print the first 25 true and predicted responses
print('True:', y_test.values[0:25])
print('False:', y_pred_class[0:25])
```

```
True: [1 0 0 1 0 1 0 0 0 1 0 1 0 1 0 0 0 1 1 0 0 1 0 0 1]
False: [1 0 1 1 0 1 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0 1]
```

Classification accuracy is the easiest classification metric to understand, But, it does not tell you the underlying distribution of response values.

We examine by calculating the null accuracy.

And, it does not tell you what "types" of errors your classifier is making.

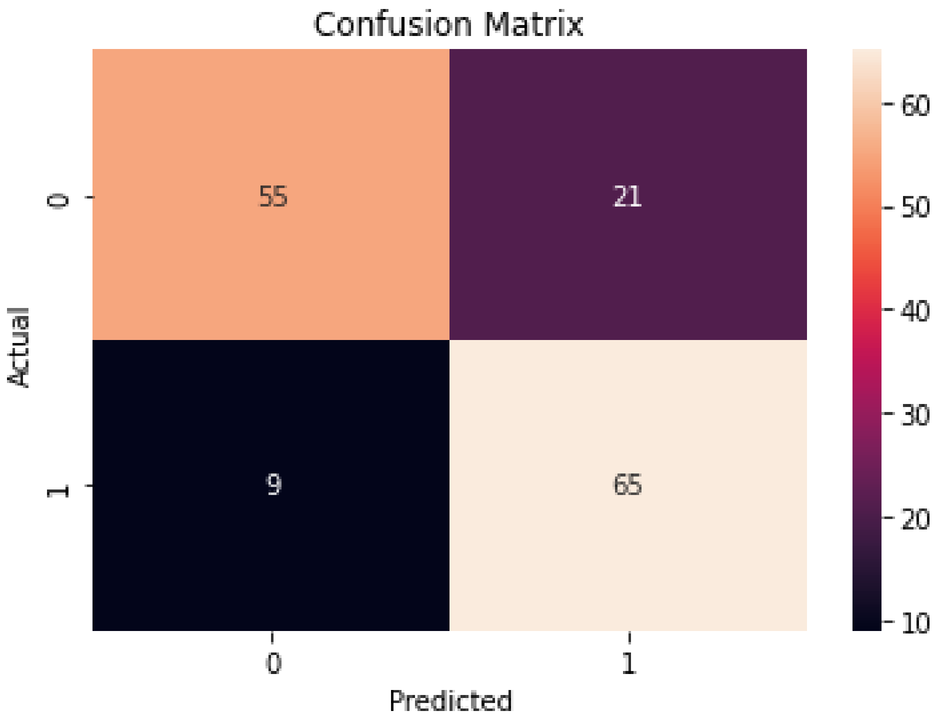
Confusion Matrix

```
Table that describes the performance of a classification model
```

[48]:
```
# IMPORTANT: first argument is true values, second argument is predicted values
# this produces a 2x2 numpy array (matrix)
confusion = metrics.confusion_matrix(y_test, y_pred_class)
print(confusion)
#[row, column]
TP = confusion[1, 1]
TN = confusion[0, 0]
FP = confusion[0, 1]
FN = confusion[1, 0]
```

```
[[55 21]
 [ 9 65]]
```

[49]:
```
  # visualize Confusion Matrix
sns.heatmap(confusion,annot=True,fmt="d")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



Basic terminology

```
. True Positives (TP): we correctly predicted that they do have diabetes
    65
. True Negatives (TN): we correctly predicted that they don't have diabetes
    55
. False Positives (FP): we incorrectly predicted that they do have diabetes (a "Type I error")
    21
    Falsely predict positive
    Type I error
. False Negatives (FN): we incorrectly predicted that they don't have diabetes (a "Type II erro
    9
    Falsely predict negative
    Type II error

. 0: negative class
. 1: positive class
```

Classification Accuracy: Overall, how often is the classifier correct?

[50]:
```python
# use float to perform true division, not integer division
print((TP + TN) / float(TP + TN + FP + FN))
print(metrics.accuracy_score(y_test, y_pred_class))
```

```
0.8
0.8
```

Sensitivity: When the actual value is positive, how often is the prediction correct?

1. Something we want to maximize
2. How "sensitive" is the classifier to detecting positive instances?
3. Also known as "True Positive Rate" or "Recall"
4. TP / all positive. all positive = TP + FN

[51]:
```
sensitivity = TP / float(FN + TP)

print(sensitivity)
print(metrics.recall_score(y_test, y_pred_class))
```

```
0.8783783783783784
0.8783783783783784
```

Specificity: When the actual value is negative, how often is the prediction correct?

1. Something we want to maximize
2. How "specific" (or "selective") is the classifier in predicting positive instances?
3. TN / all negative all negative = TN + FP

[52]:
```
specificity = TN / (TN + FP)

print(specificity)
```

```
0.7236842105263158
```

Our classifier is

1. Highly specific
2. Not sensitive

Confusion matrix gives you a more complete picture of how your classifier is performing.

Also allows you to compute various classification metrics, and these metrics can guide your model selection.

Adjusting the classification threshold

[53]:
```
# print the first 10 predicted responses
# 1D array (vector) of binary values (0, 1)
logreg.predict(X_test)[0:10]
```

[53]:
```
array([1, 0, 1, 1, 0, 1, 0, 0, 0, 1])
```

[54]:
```
# print the first 10 predicted probabilities of class membership
logreg.predict_proba(X_test)[0:10]
```

[54]:
```
array([[0.16226662, 0.83773338],
       [0.65401382, 0.34598618],
       [0.47555222, 0.52444778],
       [0.28630639, 0.71369361],
       [0.63929965, 0.36070035],
       [0.30902833, 0.69097167],
       [0.67066338, 0.32933662],
       [0.83277338, 0.16722662],
       [0.58600923, 0.41399077],
       [0.2643511 , 0.7356489 ]])
```

```
. Row: observation
       Each row, numbers sum to 1
. Column: class
       2 response classes there 2 columns
           1. column 0: predicted probability that each observation is a member of class 0
           2. column 1: predicted probability that each observation is a member of class 1
. Importance of predicted probabilities
     . We can rank observations by probability of diabetes
         . Prioritize contacting those with a higher probability
. predict_proba process
    1. Predicts the probabilities
    2. Choose the class with the highest probability
. There is a 0.5 classification threshold
    1. Class 1 is predicted if probability > 0.5
    2. Class 0 is predicted if probability < 0.5
```

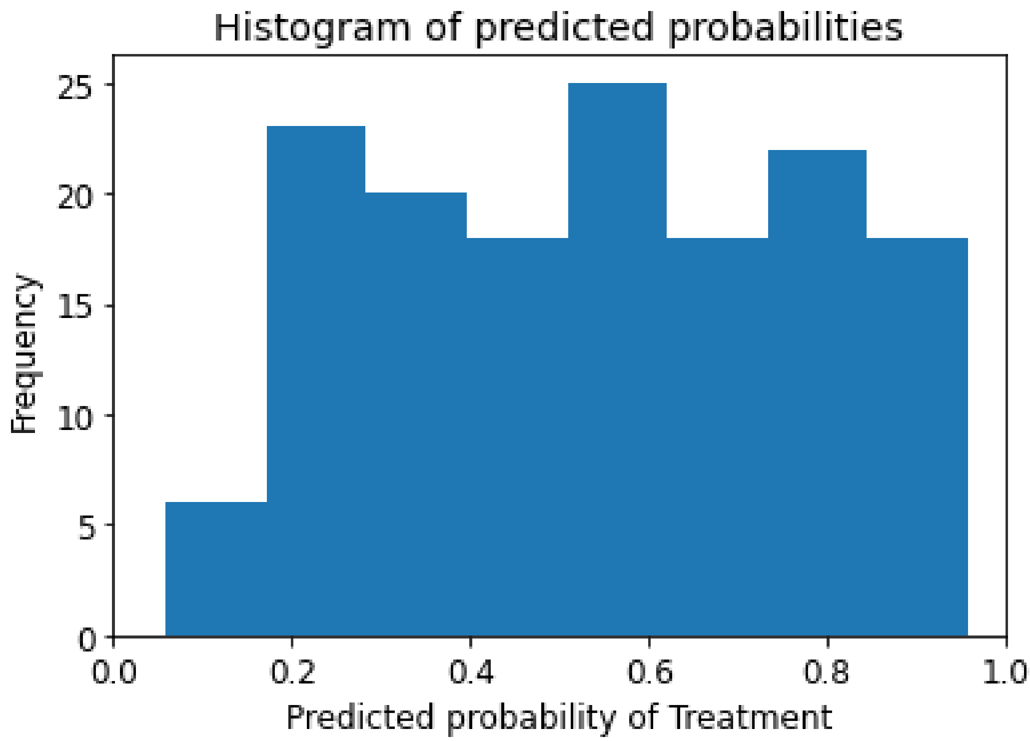
[55]:
```python
# print the first 10 predicted probabilities for class 1
logreg.predict_proba(X_test)[0:10, 1]

# store the predicted probabilities for class 1
y_pred_prob = logreg.predict_proba(X_test)[:, 1]
```

[56]:
```python
# allow plots to appear in the notebook
%matplotlib inline
import matplotlib.pyplot as plt

# adjust the font size
plt.rcParams['font.size'] = 12
```

[57]:
```python
# histogram of predicted probabilities

# 8 bins
plt.hist(y_pred_prob, bins=8)

# x-axis limit from 0 to 1
plt.xlim(0,1)
```

```
plt.title('Histogram of predicted probabilities')
plt.xlabel('Predicted probability of Treatment')
plt.ylabel('Frequency')
```

[57]: Text(0, 0.5, 'Frequency')

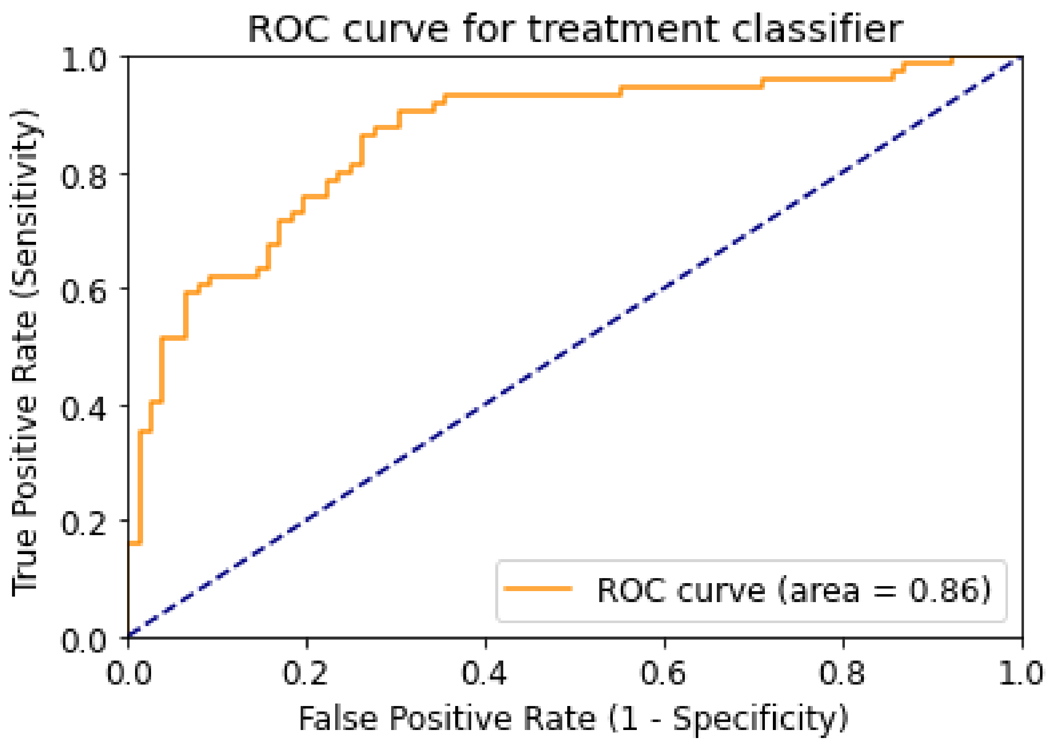


Question: Wouldn't it be nice if we could see how sensitivity and specificity are affected by various thresholds, without actually changing the threshold?

Answer: Plot the ROC curve.

[58]:
```
roc_auc = metrics.roc_auc_score(y_test, y_pred_prob)

fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_prob)
plt.figure()

plt.plot(fpr, tpr, color='darkorange', label='ROC curve (area = %0.2f)' %␣
 ↪roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.rcParams['font.size'] = 12
plt.title('ROC curve for treatment classifier')
plt.xlabel('False Positive Rate (1 - Specificity)')
```

```
plt.ylabel('True Positive Rate (Sensitivity)')
plt.legend(loc="lower right")
plt.show()
```



ROC curve can help you to choose a threshold that balances sensitivity and specificity in a way that makes sense for your particular context.

[59]:
```
# define a function that accepts a threshold and prints sensitivity and␣
↪specificity
def evaluate_threshold(threshold):
    print('Sensitivity:', tpr[thresholds > threshold][-1])
    print('Specificity:', 1 - fpr[thresholds > threshold][-1])
```

[60]:
```
evaluate_threshold(0.5)
```

```
Sensitivity: 0.8783783783783784
Specificity: 0.7236842105263157
```

[61]:
```
evaluate_threshold(0.3)
```

```
Sensitivity: 0.9459459459459459
Specificity: 0.4473684210526315
```

[62]:
```
def tuningRandomizedSearchCV(model, param_dist):
    #Searching multiple parameters simultaneously
    # n_iter controls the number of searches
    rand = RandomizedSearchCV(model, param_dist, cv=10, scoring='accuracy',␣
 ↪n_iter=10, random_state=5)
    rand.fit(X, y)
    #rand.grid_scores_

    # examine the best model
    print('Rand. Best Score: ', rand.best_score_)
    print('Rand. Best Params: ', rand.best_params_)

    # run RandomizedSearchCV 20 times (with n_iter=10) and record the best score
    best_scores = []
    for _ in range(20):
        rand = RandomizedSearchCV(model, param_dist, cv=10, scoring='accuracy',␣
 ↪n_iter=10)
        rand.fit(X, y)
        best_scores.append(round(rand.best_score_, 3))
    print(best_scores)
```

[63]:
```
    # Calculating the best parameters
tree = DecisionTreeClassifier()
featuresSize = feature_cols.__len__()
param_dist = {"max_depth": [3, None],
              "max_features": randint(1, featuresSize),
              "min_samples_split": randint(2, 9),
              "min_samples_leaf": randint(1, 9),
              "criterion": ["gini", "entropy"]}
tuningRandomizedSearchCV(tree, param_dist)

    # train a decision tree model on the training set
tree = DecisionTreeClassifier(max_depth=3, min_samples_split=8, max_features=6,␣
 ↪criterion='entropy', min_samples_leaf=7)
tree.fit(X_train, y_train)

    # make class predictions for the testing set
y_pred_class = tree.predict(X_test)

print('########### Tree classifier ###############')

 # calculate accuracy
print("Classification accuracy is",metrics.accuracy_score(y_test,␣
 ↪y_pred_class)*100,"%")
```

```
Rand. Best Score:  0.7687927927927927
Rand. Best Params:  {'criterion': 'entropy', 'max_depth': 3, 'max_features': 17,
```

```
'min_samples_leaf': 7, 'min_samples_split': 2}
[0.761, 0.746, 0.762, 0.762, 0.762, 0.77, 0.757, 0.755, 0.762, 0.766, 0.769,
0.727, 0.774, 0.763, 0.761, 0.756, 0.766, 0.765, 0.759, 0.767]
########### Tree classifier ###############
Classification accuracy is 64.0 %
```

[64]:
```
#Null accuracy: accuracy that could be achieved by always predicting the most␣
↪frequent class
    # examine the class distribution of the testing set (using a Pandas Series␣
↪method)
print('Null accuracy:\n', y_test.value_counts())
```

```
Null accuracy:
 0    76
1    74
Name: treatment, dtype: int64
```

[65]:
```
    # calculate the percentage of ones
print('Percentage of ones:', y_test.mean())
```

```
Percentage of ones: 0.49333333333333335
```

[66]:
```
    # calculate the percentage of zeros
print('Percentage of zeros:',1 - y_test.mean())
```

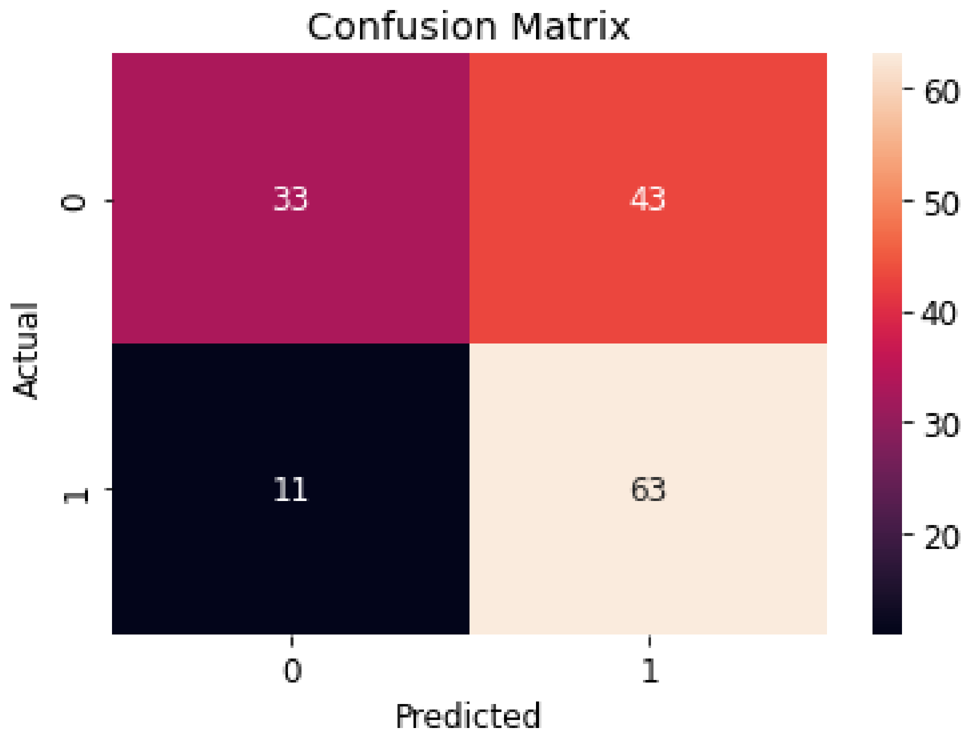
```
Percentage of zeros: 0.5066666666666666
```

[67]:
```
    #Comparing the true and predicted response values
print('True:', y_test.values[0:25])
print('Pred:', y_pred_class[0:25])
```

```
True: [1 0 0 1 0 1 0 0 0 1 0 1 0 1 0 0 0 1 1 0 0 1 0 0 1]
Pred: [1 1 1 1 1 0 0 0 1 1 0 1 1 1 1 0 1 1 1 0 0 1 0 1 1]
```

[68]:
```
    #Confusion matrix
    # save confusion matrix and slice into four pieces
confusion = metrics.confusion_matrix(y_test, y_pred_class)
    #[row, column]
TP = confusion[1, 1]
TN = confusion[0, 0]
FP = confusion[0, 1]
FN = confusion[1, 0]

    # visualize Confusion Matrix
sns.heatmap(confusion,annot=True,fmt="d")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
```

```
plt.ylabel('Actual')
plt.show()
```



[69]:
```
#False Positive Rate: When the actual value is negative, how often is the␣
↪prediction incorrect?
false_positive_rate = FP / float(TN + FP)
print('False Positive Rate:', false_positive_rate)
```

```
False Positive Rate: 0.5657894736842105
```

[70]:
```
#Precision: When a positive value is predicted, how often is the prediction␣
↪correct?
print('Precision:', metrics.precision_score(y_test, y_pred_class))
```

```
Precision: 0.5943396226415094
```

[71]:
```
print('AUC Score:', metrics.roc_auc_score(y_test, y_pred_class))
```

```
AUC Score: 0.6427809388335703
```

Adjusting the classification threshold

[72]:
```
# print the first 10 predicted responses
# 1D array (vector) of binary values (0, 1)
```

```
tree.predict(X_test)[0:10]
```

[72]:
```
array([1, 1, 1, 1, 1, 0, 0, 0, 1, 1])
```

[73]:
```
# print the first 10 predicted probabilities of class membership
tree.predict_proba(X_test)[0:10]
```

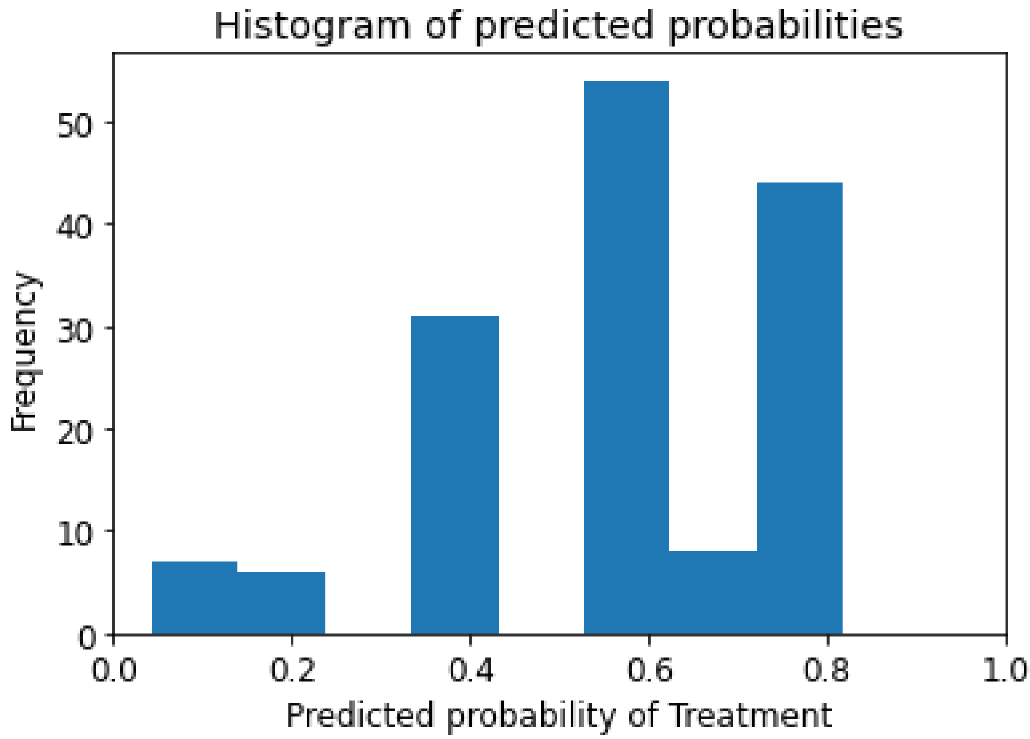
[73]:
```
array([[0.18518519, 0.81481481],
       [0.44578313, 0.55421687],
       [0.44578313, 0.55421687],
       [0.18518519, 0.81481481],
       [0.44578313, 0.55421687],
       [0.62773723, 0.37226277],
       [0.62773723, 0.37226277],
       [0.62773723, 0.37226277],
       [0.44578313, 0.55421687],
       [0.18518519, 0.81481481]])
```

[74]:
```
# print the first 10 predicted probabilities for class 1
tree.predict_proba(X_test)[0:10, 1]

# store the predicted probabilities for class 1
y_pred_prob = tree.predict_proba(X_test)[:, 1]
```

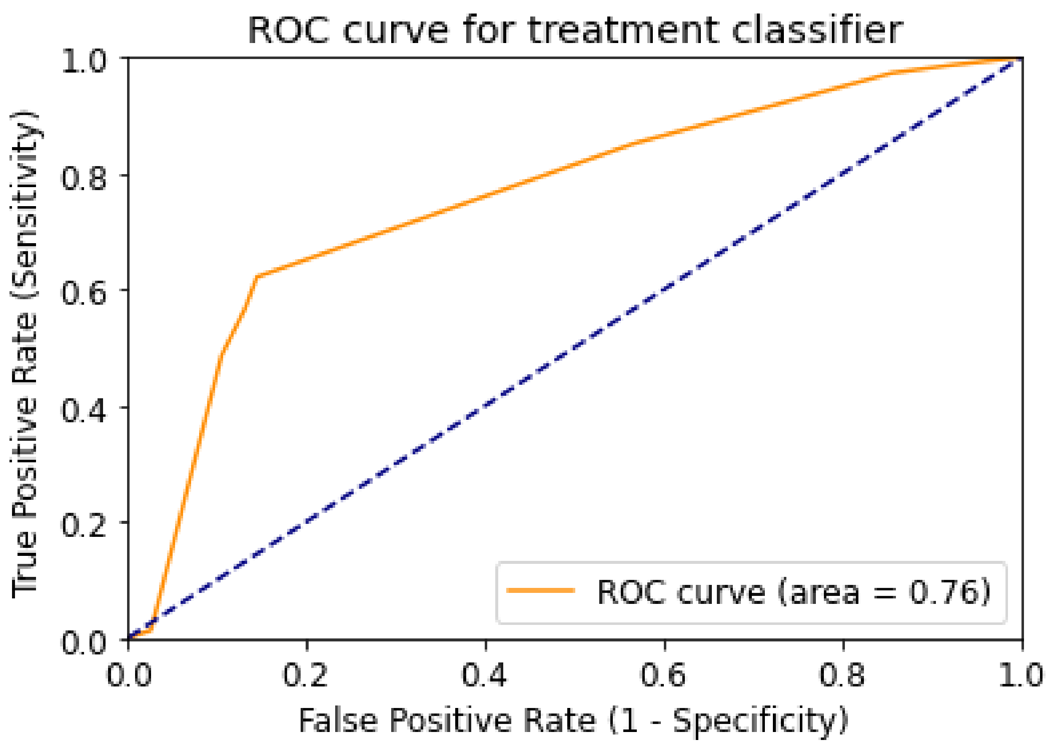
[75]:
```
# allow plots to appear in the notebook
%matplotlib inline
import matplotlib.pyplot as plt

# adjust the font size
plt.rcParams['font.size'] = 12
```

[76]:
```
# histogram of predicted probabilities

# 8 bins
plt.hist(y_pred_prob, bins=8)

# x-axis limit from 0 to 1
plt.xlim(0,1)
plt.title('Histogram of predicted probabilities')
plt.xlabel('Predicted probability of Treatment')
plt.ylabel('Frequency')
```

[76]:
```
Text(0, 0.5, 'Frequency')
```



[77]:
```
#### Plot the ROC Curve

roc_auc = metrics.roc_auc_score(y_test, y_pred_prob)

fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_prob)
plt.figure()

plt.plot(fpr, tpr, color='darkorange', label='ROC curve (area = %0.2f)' %␣
 ↪roc_auc)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.rcParams['font.size'] = 12
plt.title('ROC curve for treatment classifier')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.legend(loc="lower right")
plt.show()
```



0.0.2 Trying Other Models

[80]:
```
def train_evaluate(model,X_train,y_train,name):
    model.fit(X_train,y_train)
    y_pred = model.predict(X_train)
    f1_train = f1_score(y_train,y_pred)

    #Cross validation
    f1_val = cross_val_score(model,X_train,y_train,scoring='f1',cv=10)

    # returning the scores
    score = pd.DataFrame({'Name' : name ,'F1_score_trainset' : [f1_train],␣
 ↪'F1_score_validationset' : [f1_val.mean()]})
    return score
```

Gradient Boost

[82]:
```
from sklearn.ensemble import GradientBoostingClassifier
gdb_clf = GradientBoostingClassifier(random_state=42,subsample=0.8)

train_evaluate(gdb_clf,X_train,y_train,"GradientBoosting CLASSIFIER")
```

[82]:
```
                          Name  F1_score_trainset  F1_score_validationset
0  GradientBoosting CLASSIFIER           0.925547                0.774363
```

XGBoost

[84]:
```python
from xgboost import XGBClassifier
xgb_clf = XGBClassifier(verbosity=0)

train_evaluate(xgb_clf,X_train,y_train,"XG Boost CLASSIFIER")
```

[84]:
```
                  Name  F1_score_trainset  F1_score_validationset
0  XG Boost CLASSIFIER                1.0                0.754313
```

Finetuning GradientBoost

[85]:
```python
param_grid = [
    {'n_estimators':[3,10,30,50,100],
    'max_features':[2,4,6,8,10],
    'max_depth' : [1,2,3,4],
    'subsample': [0.25,0.5,0.75]}
]

gdb_clf2 = GradientBoostingClassifier(random_state=42)
grid_search2 = GridSearchCV(gdb_clf2, param_grid, cv=5,
                           scoring='f1',
                           return_train_score=True)
grid_search2.fit(X_train, y_train)
```

[85]:
```
GridSearchCV(cv=5, estimator=GradientBoostingClassifier(random_state=42),
             param_grid=[{'max_depth': [1, 2, 3, 4],
                          'max_features': [2, 4, 6, 8, 10],
                          'n_estimators': [3, 10, 30, 50, 100],
                          'subsample': [0.25, 0.5, 0.75]}],
             return_train_score=True, scoring='f1')
```

[86]:
```python
grid_search2.best_estimator_
```

[86]:
```
GradientBoostingClassifier(max_depth=2, max_features=8, n_estimators=10,
                           random_state=42, subsample=0.25)
```

Re-evaluating Model

[87]:
```python
train_evaluate(grid_search2.best_estimator_,X_train,y_train,"GradientBoosting␣
 ↪Tuned")
```

[87]:
```
                     Name  F1_score_trainset  F1_score_validationset
0  GradientBoosting Tuned           0.796143                0.790374
```

Finetuning XGBoost

[88]:
```
param_grid = [
    {'n_estimators':[3,10,30,50,100],
    'eta' : [0.01,0.025, 0.05, 0.1],
    'max_features':[2,4,6,8],
    'max_depth' : [1,2,3,4],
    'subsample': [0.5,0.75],
    'booster':['gblinear','gbtree']}
]

xgb_clf = XGBClassifier(verbosity = 0)
grid_search3 = GridSearchCV(xgb_clf, param_grid, cv=5,
                           scoring='f1',
                           return_train_score=True)
grid_search3.fit(X_train, y_train)
```

[88]:
```
GridSearchCV(cv=5,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     callbacks=None, colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None,
                                     early_stopping_rounds=None,
                                     enable_categorical=False, eval_metric=None,
                                     feature_types=None, gamma=None,
                                     gpu_id=None, grow_policy=None,
                                     importance_type=None,
                                     interaction_constraints=None,
                                     learning_rate=None,…
                                     max_leaves=None, min_child_weight=None,
                                     missing=nan, monotone_constraints=None,
                                     n_estimators=100, n_jobs=None,
                                     num_parallel_tree=None, predictor=None,
                                     random_state=None, …),
             param_grid=[{'booster': ['gblinear', 'gbtree'],
                          'eta': [0.01, 0.025, 0.05, 0.1],
                          'max_depth': [1, 2, 3, 4],
                          'max_features': [2, 4, 6, 8],
                          'n_estimators': [3, 10, 30, 50, 100],
                          'subsample': [0.5, 0.75]}],
             return_train_score=True, scoring='f1')
```

[89]:
```
grid_search3.best_estimator_
```

[89]:
```
XGBClassifier(base_score=None, booster='gbtree', callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eta=0.025, eval_metric=None,
```

```
                   feature_types=None, gamma=None, gpu_id=None, grow_policy=None,
                   importance_type=None, interaction_constraints=None,
                   learning_rate=None, max_bin=None, max_cat_threshold=None,
                   max_cat_to_onehot=None, max_delta_step=None, max_depth=3,
                   max_features=2, max_leaves=None, min_child_weight=None,
                   missing=nan, monotone_constraints=None, n_estimators=30,
                   n_jobs=None, num_parallel_tree=None, …)
```

[90]:
```
train_evaluate(grid_search3.best_estimator_,X_train,y_train,"XGBoost Finetuned")
```

[90]:

	Name	F1_score_trainset	F1_score_validationset
0	XGBoost Finetuned	0.813953	0.79559

After finetuning, XGboost is generalizing well on our dataset. So XGBoost will be my selected model.

So, We have successfully developed a model which can predict whether a employee seeks mental health treatment or not.