IE 7275 Project

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import SimpleImputer, IterativeImputer
from sklearn.preprocessing import MinMaxScaler
#from category_encoders import BinaryEncoder
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, recall_score, plot_roc_curve
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoosting(
from xgboost.sklearn import XGBClassifier
from sklearn.model_selection import GridSearchCV
from sklearn import svm
import seaborn as sns
```

In [2]:

```
df_mhealth = pd.read_csv('survey.csv')
df_mhealth.head()
```

Out[2]:

	Timestamp	Age	Gender	Country	state	self_employed	family_history	treatment	work_interfere	no_employee
0	2014-08-27 11:29:31	37	Female	United States	IL	NaN	No	Yes	Often	6-2
1	2014-08-27 11:29:37	44	M	United States	IN	NaN	No	No	Rarely	More tha 100
2	2014-08-27 11:29:44	32	Male	Canada	NaN	NaN	No	No	Rarely	6-2
3	2014-08-27 11:29:46	31	Male	United Kingdom	NaN	NaN	Yes	Yes	Often	26-10
4	2014-08-27 11:30:22	31	Male	United States	TX	NaN	No	No	Never	100-50

5 rows × 27 columns

In [3]:

```
df_mhealth.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1259 entries, 0 to 1258
Data columns (total 27 columns):
 #   Column                     Non-Null Count  Dtype 
---  ------                     --------------  ----- 
 0   Timestamp                  1259 non-null   object
 1   Age                        1259 non-null   int64 
 2   Gender                     1259 non-null   object
 3   Country                    1259 non-null   object
 4   state                      744 non-null    object
 5   self_employed              1241 non-null   object
 6   family_history             1259 non-null   object
 7   treatment                  1259 non-null   object
 8   work_interfere             995 non-null    object
 9   no_employees               1259 non-null   object
 10  remote_work                1259 non-null   object
 11  tech_company               1259 non-null   object
 12  benefits                   1259 non-null   object
 13  care_options               1259 non-null   object
 14  wellness_program           1259 non-null   object
 15  seek_help                  1259 non-null   object
 16  anonymity                  1259 non-null   object
 17  leave                      1259 non-null   object
 18  mental_health_consequence  1259 non-null   object
 19  phys_health_consequence    1259 non-null   object
 20  coworkers                  1259 non-null   object
 21  supervisor                 1259 non-null   object
 22  mental_health_interview    1259 non-null   object
 23  phys_health_interview      1259 non-null   object
 24  mental_vs_physical         1259 non-null   object
 25  obs_consequence            1259 non-null   object
 26  comments                   164 non-null    object
dtypes: int64(1), object(26)
memory usage: 265.7+ KB
```

Data Cleaning

In [4]:

```
df_mhealth.isna().sum()
```

Out[4]:

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

In [5]:

```
df_mhealth.drop(columns=['Timestamp', 'state', 'comments'], inplace = True)
```

In [6]:

```
df_mhealth['Age'].unique()
```

Out[6]:

```
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)
```

In [7]:

```
df_mhealth['Age'].replace([df_mhealth['Age'][df_mhealth['Age'] < 18]], np.nan, inplace = T
df_mhealth['Age'].replace([df_mhealth['Age'][df_mhealth['Age'] > 90]], np.nan, inplace = T
```

In [8]:

```
df_mhealth['Gender'].unique()
```

Out[8]:

```
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)
```

In [9]:

```
df_mhealth['Gender'].replace(['Male ', 'male', 'm', 'M', 'Male', 'Man', 'Cis Male', 'cis m
'Cis Man', 'msle', 'Malr', 'Mal', 'maile', 'Make',], 'Male', inplace = True)

df_mhealth['Gender'].replace(['Female ', 'female', 'F', 'f', 'Woman', 'woman','Female', 'f
'cis-female/femme', 'Femake', ], 'Female', inplace = True)

df_mhealth["Gender"].replace(['Female (trans)', 'queer/she/they', 'non-binary','fluid', 'q
'male leaning androgynous', 'Agender', 'A little about you', 'Nah', 'All', 'ostensibly mal
'Genderqueer', 'Enby', 'p', 'Neuter', 'something kinda male?','Guy (-ish) ^_^', 'Trans wor
```

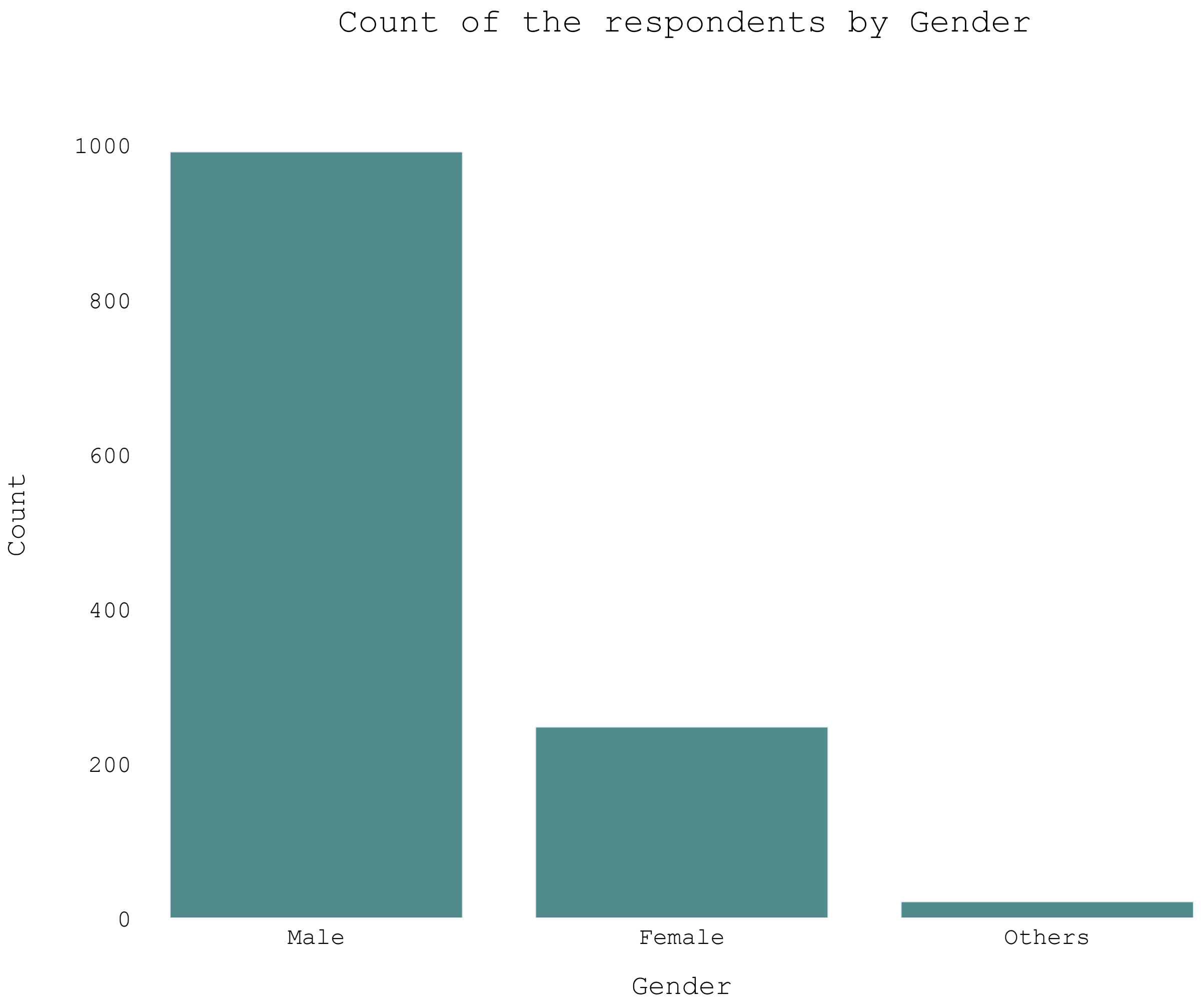
In [10]:

```
male_country   = df_mhealth[df_mhealth['Gender'] == 'Male'][['Country', 'Gender']]
female_country = df_mhealth[df_mhealth['Gender'] == 'Female'][['Country', 'Gender']]
male_country   = male_country.value_counts()
female_country = female_country.value_counts()

male_country   = pd.DataFrame(male_country).reset_index().rename(columns={0:'count'}).head
female_country = pd.DataFrame(female_country).reset_index().rename(columns={0:'count'}).he
male_country['count'] = male_country['count'] * -1
```

In [11]:

```
import plotly.graph_objs as go
import plotly
fig = plotly.tools.make_subplots(
                    shared_yaxes=True,
                    horizontal_spacing=0)

fig.append_trace(go.Bar(
                 y = df_mhealth["Gender"].value_counts(),
                 x = ["Male", "Female", "Others"],
                 textfont = dict(size = 10, color = '#6aa87b'),
                 textposition = 'outside',
                 name = 'Count of Employees by Gender',
                 marker_color='#528B8B',
                 orientation = 'v'),
                 row=1, col=1)

fig.update_layout(
                  font_family   = 'monospace',
                  title         = dict(text = 'Count of the respondents by Gender', x = 0.
                  margin        = dict(t=80, b=0, l=70, r=40),
                  hovermode     = "y unified",
                  plot_bgcolor  = '#edf2c7',
                  paper_bgcolor = '#edf2c7',
                  xaxis_title = "Gender",
                  yaxis_title = "Count",
                  font          = dict(color='black'),
                  legend        = dict(orientation="h",
                                       yanchor="bottom", y=1,
                                       xanchor="center", x=0.5),
                  hoverlabel    = dict(bgcolor="#edf2c7", font_size=13,
                                      font_family="Monospace"))

fig.show()
```

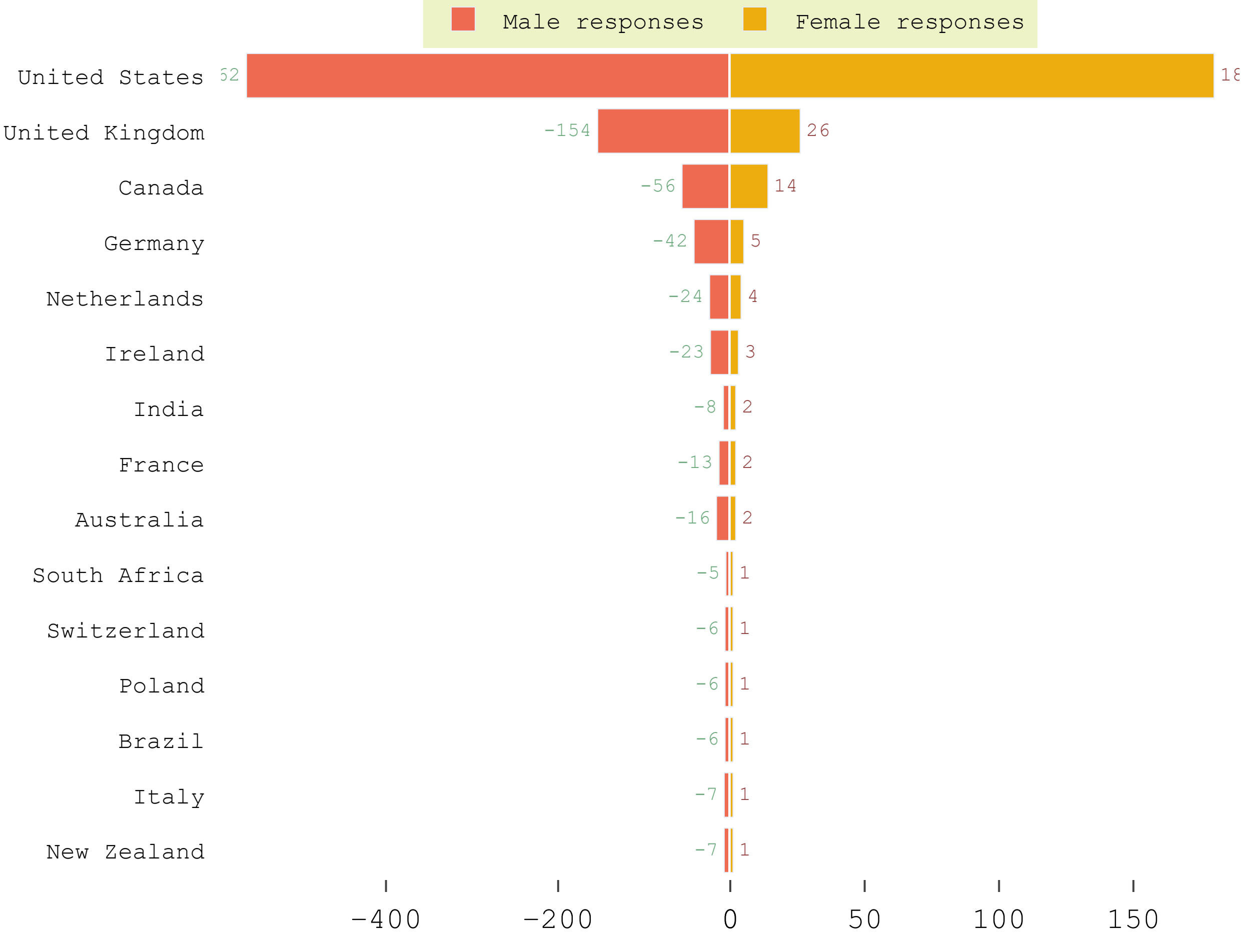
```
c:\users\amogha shettar\appdata\local\programs\python\python39\lib\site-packages\plotly\to
ols.py:461: DeprecationWarning:

plotly.tools.make_subplots is deprecated, please use plotly.subplots.make_subplots instead
```



In [12]:

```
import plotly.graph_objs as go
import plotly
fig = plotly.tools.make_subplots(rows=1, cols=2,
                    specs=[[{}, {}]],
                    shared_yaxes=True,
                    horizontal_spacing=0)

fig.append_trace(go.Bar(
                 y = male_country.Country,
                 x = male_country['count'],
                 text = male_country['count'],
                 textfont = dict(size = 10, color = '#6aa87b'),
                 textposition = 'outside',
                 name = 'Male responses',
                 marker_color='#EE6A50',
                 orientation = 'h'),
                 row=1, col=1)

fig.append_trace(go.Bar(
                 y = male_country.Country,
                 x = female_country['count'],
                 text = female_country['count'],
                 textfont = dict(size = 10, color = '#913f3f'),
                 textposition = 'outside',
                 name = 'Female responses',
                 marker_color='#EEAD0E',
                 orientation = 'h'),
                 row=1, col=2)


fig.update_xaxes(
        tickfont = dict(size=15),
        tickmode = 'array',
        ticklen = 6,
        showline = False,
        showgrid = False,
        ticks = 'outside')

fig.update_yaxes(showgrid=False,
                 categoryorder='total ascending',
                 ticksuffix=' ',
                 showline=False)

fig.update_layout(
                  font_family   = 'monospace',
                  title         = dict(text = 'Gender of the respondents across Countries'
                  margin        = dict(t=80, b=0, l=70, r=40),
                  hovermode     = "y unified",
                  plot_bgcolor  = '#edf2c7',
                  paper_bgcolor = '#edf2c7',
                  font          = dict(color='black'),
                  legend        = dict(orientation="h",
                                       yanchor="bottom", y=1,
                                       xanchor="center", x=0.5),
                  hoverlabel    = dict(bgcolor="#edf2c7", font_size=13,
                                      font_family="Monospace"))


fig.show()
```

```
c:\users\amogha shettar\appdata\local\programs\python\python39\lib\site-packages\plotly\to
ols.py:461: DeprecationWarning:

plotly.tools.make_subplots is deprecated, please use plotly.subplots.make_subplots instead
```



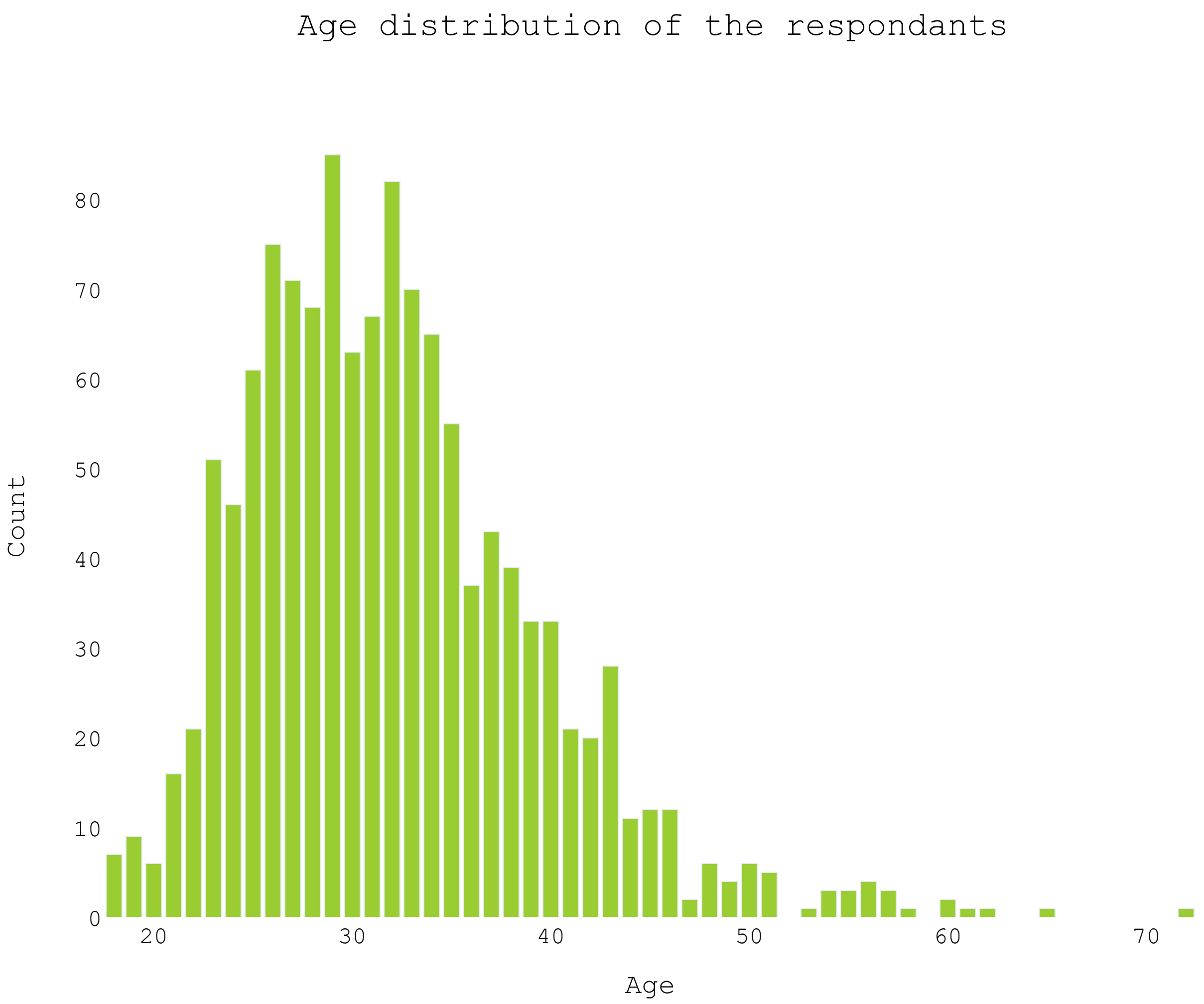
In [13]:

```
print('Maximum Age: ', df_mhealth.Age.max())
df_mhealth = df_mhealth[(df_mhealth.Age > 11) & (df_mhealth.Age <= 100)]
print(df_mhealth.Age.describe().astype(int))
```

```
Maximum Age:  72.0
count    1251
mean       32
std         7
min        18
25%        27
50%        31
75%        36
max        72
Name: Age, dtype: int32
```

In [14]:

```
fig1 = plotly.subplots.make_subplots(
                    shared_yaxes=True,
                    horizontal_spacing=0)

fig1.append_trace(go.Bar(
                 y = df_mhealth["Age"].value_counts(),
                 x = [29.0, 32.0,26.0,27.0,33.0,28.0,31.0,34.0,30.0,25.0,35.0, 23.0,24.0,
                 textfont = dict(size = 10, color = '#6aa87b'),
                 textposition = 'outside',
                 marker_color='#9ACD32',
                 orientation = 'v'),
                 row=1, col=1)

fig1.update_layout(
                  font_family   = 'monospace',
                  title         = dict(text = 'Age distribution of the respondants', x = (
                  margin        = dict(t=80, b=0, l=70, r=40),
                  hovermode     = "y unified",
                  xaxis_title = "Age",
                  yaxis_title = "Count",
                  plot_bgcolor  = '#edf2c7',
                  paper_bgcolor = '#edf2c7',
                  font          = dict(color='black'),
                  legend        = dict(orientation="h",
                                       yanchor="bottom", y=1,
                                       xanchor="center", x=0.5),
                  hoverlabel    = dict(bgcolor="#edf2c7", font_size=13,
                                      font_family="Monospace"))

fig1.show()
```

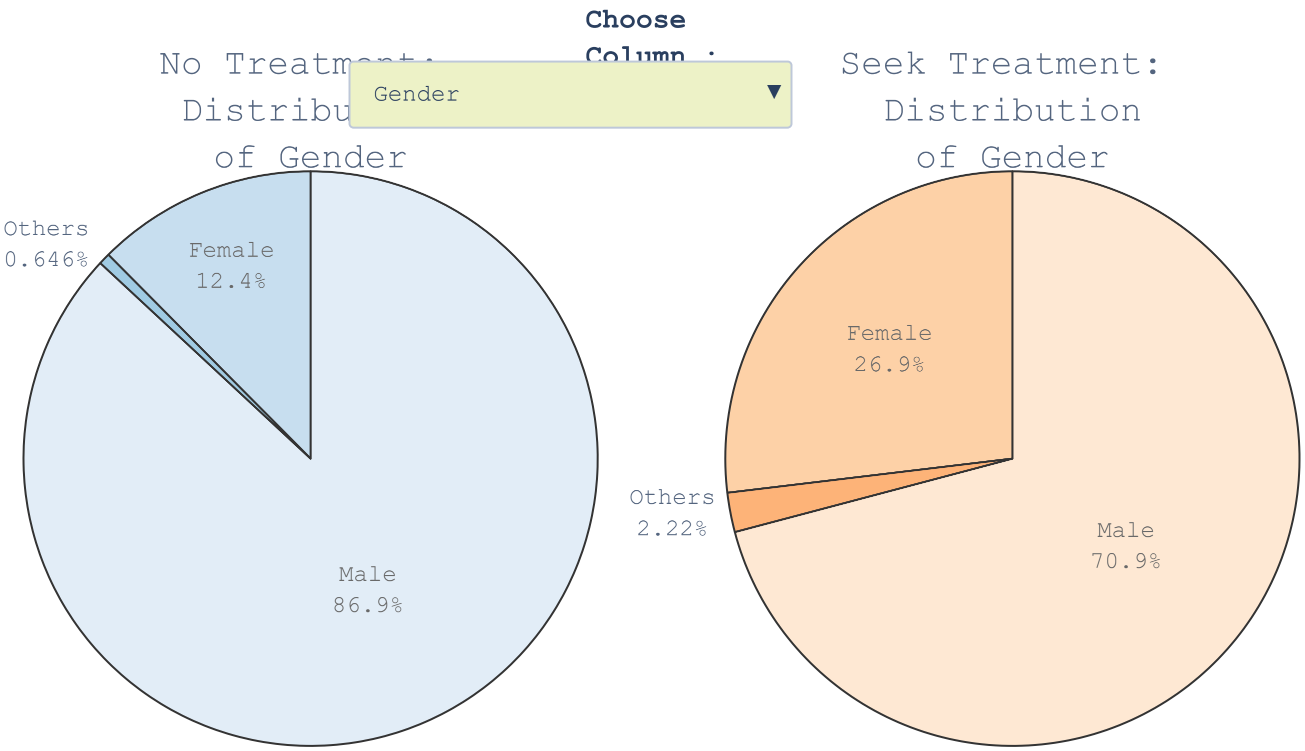


In [15]:

```
seek = df_mhealth[df_mhealth.treatment == 'Yes'].drop(['treatment', 'Country', 'Age'], axi
dont = df_mhealth[df_mhealth.treatment == 'No'].drop(['treatment', 'Country', 'Age'], axis
```

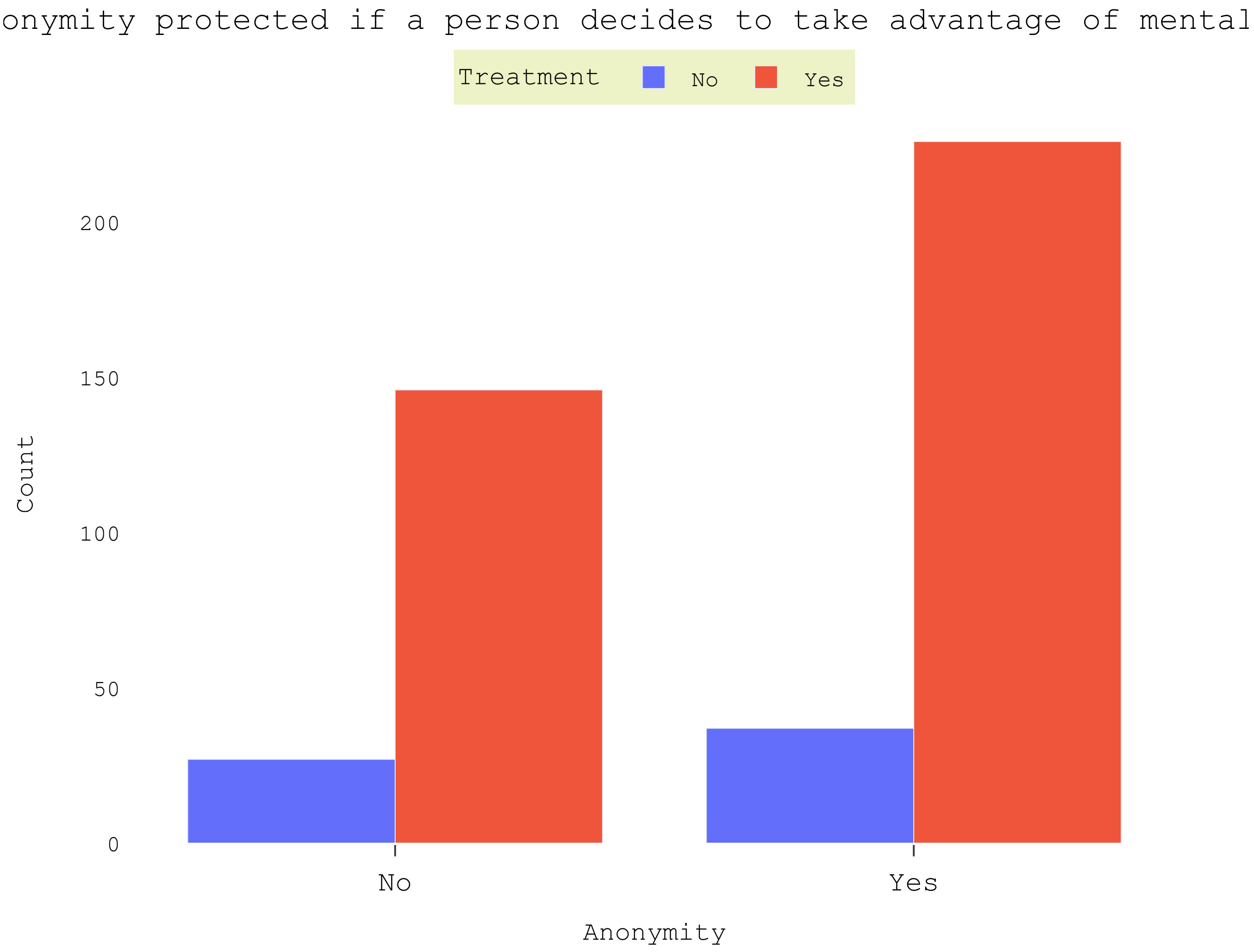
In [16]:

```
buttons = []
i = 0
vis = [False] * 21

for col in seek.columns:
    vis[i] = True
    buttons.append({'label' : col,
             'method' : 'update',
             'args'   : [{'visible' : vis},
             {'title'  : col}] })
    i+=1
    vis = [False] * 21

fig = plotly.tools.make_subplots(rows=1, cols=2,
                    specs=[[{'type':'domain'}, {'type':'domain'}]])

for col in dont.columns:
    fig.add_trace(go.Pie(
             values = dont[col].value_counts(),
             labels = dont[col].value_counts().index,
             title = dict(text = 'No Treatment: <br>Distribution<br>of {}'.format(col),
                          font = dict(size=18, family = 'monospace'),
                          ),
             hoverinfo='label+percent',),1,1)


for col in seek.columns:
    fig.add_trace(go.Pie(
             values = seek[col].value_counts(),
             labels = seek[col].value_counts().index,
             title = dict(text = 'Seek Treatment: <br>Distribution<br>of {}'.format(col),
                          font = dict(size=18, family = 'monospace'),
                          ),
             hoverinfo='label+percent',),1,2)


fig.update_traces(hoverinfo='label+percent',
                  textinfo='label+percent',
                  textfont_size=12,
                  opacity = 0.8,
                  showlegend = False,
                  marker = dict(colors = sns.color_palette('Blues').as_hex(),
                              line=dict(color='#000000', width=1)))

fig.update_traces(row=1, col=2, hoverinfo='label+percent',
                  textinfo='label+percent',
                  textfont_size=12,
                  opacity = 0.8,
                  showlegend = False,
                  marker = dict(colors = sns.color_palette('Oranges').as_hex(),
                              line=dict(color='#000000', width=1)))


fig.update_layout(margin=dict(t=0, b=0, l=0, r=0),
                  font_family   = 'monospace',
                  hovermode     = "y unified",
                  plot_bgcolor  = '#edf2c7',
                  paper_bgcolor = '#edf2c7',
                  updatemenus = [dict(
                        type = 'dropdown',
                        x = 0.60,
                        y = 0.95,
                        showactive = True,
                        active = 0,
                        buttons = buttons)],
                 annotations=[
                             dict(text = "<b>Choose<br>Column<b> : ",
                                  font = dict(size = 14),
                             showarrow=False,
                             x = 0.5, y = 1.03, yref = "paper", align = "left")])

for i in range(1,42):
    fig.data[i].visible = False
fig.data[21].visible = True

fig.show()
```



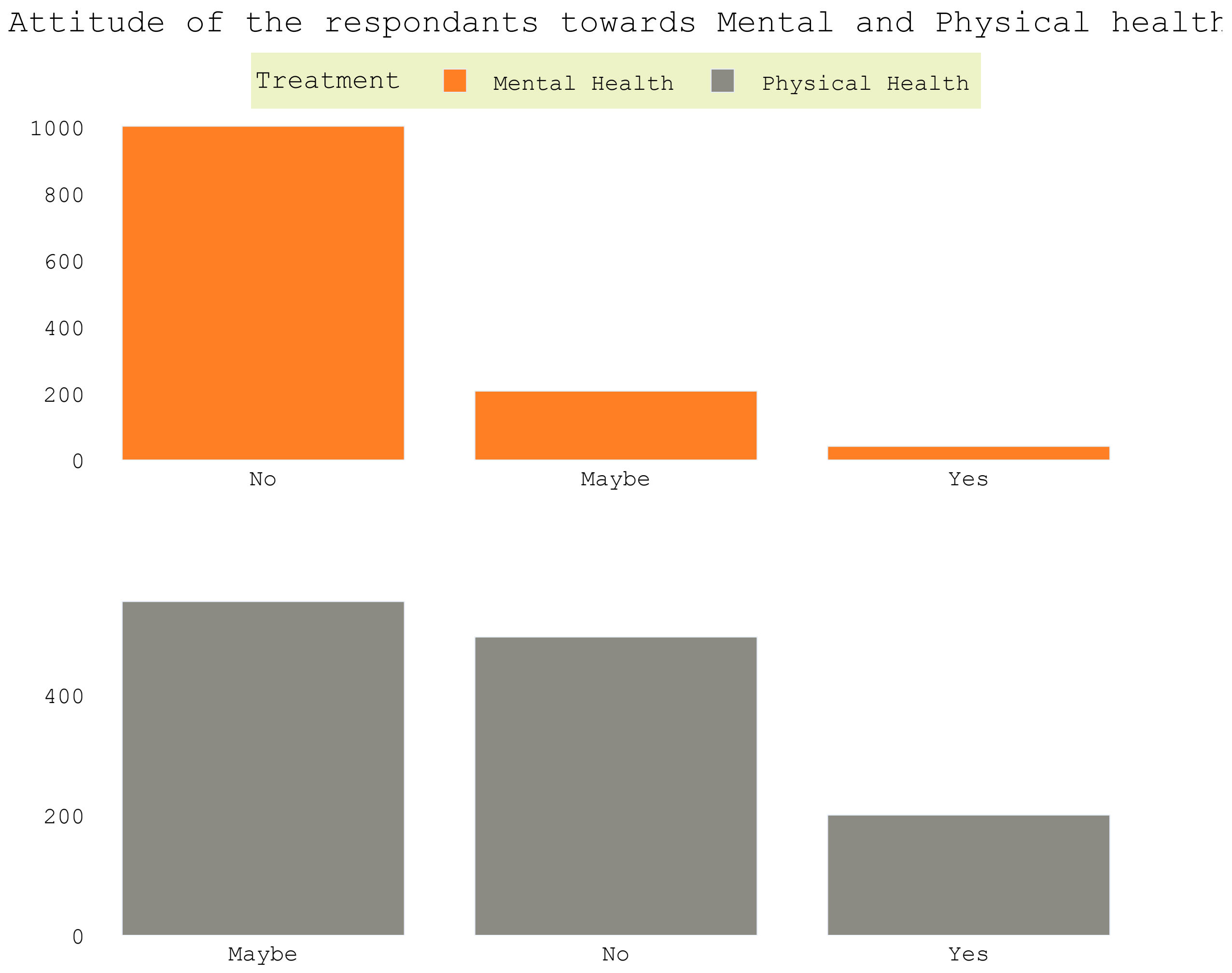
In [17]:

```
adf = df_mhealth[(df_mhealth.anonymity == 'Yes') | (df_mhealth.anonymity == 'No')]
df_cross = pd.crosstab(adf.treatment,adf.anonymity)
# initiate data list for figure
data = []
#use for loop on every zoo name to create bar data
for x in df_cross.columns:
    data.append(go.Bar(name=str(x), x=df_cross.index, y=df_cross[x]))

figure = go.Figure(data)
figure.update_layout(barmode = 'group')

figure.update_xaxes(
        tickfont = dict(size=15),
        tickmode = 'array',
        ticklen = 6,
        showline = False,
        showgrid = False,
        ticks = 'outside')

figure.update_yaxes(showgrid=False,
                 categoryorder='total ascending',
                 ticksuffix=' ',
                 showline=False)

figure.update_layout(
                  font_family   = 'monospace',
                  title         = dict(text = 'Is anonymity protected if a person decides
                  margin        = dict(t=80, b=0, l=70, r=40),
                  hovermode     = "y unified",
                  plot_bgcolor  = '#edf2c7',
                  paper_bgcolor = '#edf2c7',
                  xaxis_title = "Anonymity",
                  yaxis_title = "Count",
                  legend_title = "Treatment",
                  font          = dict(color='black'),
                  legend        = dict(orientation="h",
                                       yanchor="bottom", y=1,
                                       xanchor="center", x=0.5),
                  hoverlabel    = dict(bgcolor="#edf2c7", font_size=13,
                                      font_family="Monospace"))

figure.show()
```



In [18]:

```
fig = plotly.tools.make_subplots(rows=2, cols=1,
                    shared_yaxes=True,
                    horizontal_spacing=0)

fig.append_trace(go.Bar(
                 y = df_mhealth.mental_health_interview.value_counts(),
                 x = ["No", "Maybe", "Yes"],
                 textfont = dict(size = 10, color = '#6aa87b'),
                 textposition = 'outside',
                 name = 'Mental Health',
                 marker_color='#FF7F24',
                 orientation = 'v'),
                 row=1, col=1)

fig.append_trace(go.Bar(
                 y = df_mhealth.phys_health_interview.value_counts(),
                 x = ["Maybe", "No", "Yes"],
                 textfont = dict(size = 10, color = '#913f3f'),
                 textposition = 'outside',
                 name = 'Physical Health',
                 marker_color='#8B8B83',
                 orientation = 'v'),
                 row=2, col=1)

fig.update_layout(
                  font_family   = 'monospace',
                  title         = dict(text = 'Attitude of the respondants towards Mental
                  margin        = dict(t=80, b=0, l=70, r=40),
                  hovermode     = "y unified",
                  plot_bgcolor  = '#edf2c7',
                  paper_bgcolor = '#edf2c7',
                  #xaxis_title = "Response",
                  #yaxis_title = "Count",
                  legend_title = "Treatment",
                  font          = dict(color='black'),
                  legend        = dict(orientation="h",
                                       yanchor="bottom", y=1,
                                       xanchor="center", x=0.5),
                  hoverlabel    = dict(bgcolor="#edf2c7", font_size=13,
                                      font_family="Monospace"))

fig.show()
```



In [19]:
```
df_mhealth.drop(columns=['Country'], inplace = True)
```

Preprocessing

In [20]:
```
mode_onehot_pipe = Pipeline([
    ('encoder', SimpleImputer(strategy = 'most_frequent')),
    ('one hot encoder', OneHotEncoder(handle_unknown = 'ignore'))])
```

```
transformer = ColumnTransformer([
    ('one hot', OneHotEncoder(handle_unknown = 'ignore'), ['Gender', 'family_history', 'no
                                                           'remote_work', 'tech_company',
                                                           'wellness_program', 'anonymity'
                                                           'mental_health_interview', 'phy
                    'mental_vs_physical', 'obs_consequence', 'mental_health_consequence',
                               'seek_help']),
    ('mode_onehot_pipe', mode_onehot_pipe, ['self_employed', 'work_interfere']),
    ('iterative', IterativeImputer(max_iter = 10, random_state = 0), ['Age'])])
```

In [21]:

```
df_mhealth['treatment'] = np.where(df_mhealth['treatment'] == 'Yes', 1, 0)
```

In [22]:

```
X = df_mhealth.drop(df_mhealth[['treatment']] , axis = 1)
y = df_mhealth['treatment']
```

In [23]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify = y, test_size = 0.3, r
```

In [24]:

```
def model_evaluation(model, metric):
    model_cv = cross_val_score(model, X_train, y_train, cv = StratifiedKFold(n_splits = 5)
    return model_cv
```

In [25]:

```
log_model = LogisticRegression()
```

In [26]:

```
log_pipe = Pipeline([('transformer', transformer), ('log_model', log_model)])

log_pipe_cv = model_evaluation(log_pipe, 'recall')

for model in [log_pipe]:
    model.fit(X_train, y_train)

log_score_cv = [log_pipe_cv.round(2)]
log_score_recall_score = [recall_score(y_test, log_pipe.predict(X_test))
                     ]
method_name = ['Logistic Regression']
log_cv = pd.DataFrame({
    'method': method_name,
    'cv score': log_score_cv,
    'recall score': log_score_recall_score
})
log_cv
```

Out[26]:

	method	cv score	recall score
0	Logistic Regression	[0.69, 0.77, 0.74, 0.76, 0.72]	0.742105

In [27]:

```
decision_model = DecisionTreeClassifier(random_state = 1111)
```

In [28]:

```
decision_pipe = Pipeline([('transformer', transformer), ('decision_model', decision_model)

decision_pipe_cv = model_evaluation(decision_pipe, 'recall')

for model in [decision_pipe]:
    model.fit(X_train, y_train)
```

```
decision_score_cv = [decision_pipe_cv.round(2)]
decision_score_recall_score = [recall_score(y_test, decision_pipe.predict(X_test))
                              ]
method_name = ['Decision Tree Classifier']
decision_cv = pd.DataFrame({
    'method': method_name,
    'cv score': decision_score_cv,
    'recall score': decision_score_recall_score
})
decision_cv
```

Out[28]:

	method	cv score	recall score
0	Decision Tree Classifier	[0.63, 0.65, 0.67, 0.61, 0.66]	0.678947

In [29]:

```
svm_model = svm.SVC()
```

In [30]:

```
svm_pipe = Pipeline([('transformer', transformer), ('svm_model', svm_model)])

svm_pipe_cv = model_evaluation(svm_pipe, 'recall')

for model in [svm_pipe]:
    model.fit(X_train, y_train)

svm_score_cv = [svm_pipe_cv.round(2)]
svm_score_recall_score = [recall_score(y_test, svm_pipe.predict(X_test))
                              ]
method_name = ['Support Vector Machine']
svm_cv = pd.DataFrame({
    'method': method_name,
    'cv score': svm_score_cv,
    'recall score': svm_score_recall_score
})
svm_cv
```

Out[30]:

	method	cv score	recall score
0	Support Vector Machine	[0.65, 0.67, 0.7, 0.66, 0.65]	0.563158

In [31]:

```
random_model = RandomForestClassifier(random_state = 1111)
```

In [32]:

```
random_pipe = Pipeline([('transformer', transformer), ('random_model', random_model)])

random_pipe_cv = model_evaluation(random_pipe, 'recall')

for model in [random_pipe]:
    model.fit(X_train, y_train)

random_score_cv = [random_pipe_cv.round(2)]
random_score_recall_score = [recall_score(y_test, random_pipe.predict(X_test))
                              ]
method_name = ['Random Forest Classifier']
random_cv = pd.DataFrame({
    'method': method_name,
    'cv score': random_score_cv,
    'recall score': random_score_recall_score
})
random_cv
```

Out[32]:

	method	cv score	recall score
0	Random Forest Classifier	[0.74, 0.76, 0.76, 0.7, 0.73]	0.715789

In [33]:

```
#Encoding data
from sklearn import preprocessing
from sklearn.preprocessing import binarize, LabelEncoder, MinMaxScaler

labelDict = {}
for feature in df_mhealth:
    le = preprocessing.LabelEncoder()
    le.fit(df_mhealth[feature])
    le_name_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
    df_mhealth[feature] = le.transform(df_mhealth[feature])
    labelKey = 'label_' + feature
    labelValue = [*le_name_mapping]
    labelDict[labelKey] =labelValue
```

In [34]:

```
scaler = MinMaxScaler()
df_mhealth['Age'] = scaler.fit_transform(df_mhealth[['Age']])
```

In [35]:

```
feature_all = ['Age', 'Gender', 'family_history', 'benefits', 'care_options', 'anonymity',
feature_cols = ['Age', 'Gender', 'family_history', 'benefits', 'care_options', 'anonymity'
                 'no_employees', 'remote_work', 'tech_company', 'wellness_program', 'anonym
                 'mental_health_interview', 'phys_health_interview', 'mental_vs_physical',
                 'mental_health_consequence', 'phys_health_consequence', 'leave', 'self_emp
X2 = df_mhealth[feature_cols]
X1 = df_mhealth[feature_all]
y1 = df_mhealth.treatment

X_train1, X_test1, y_train1, y_test1 = train_test_split(X1, y1, test_size=0.30, random_sta

X_train2, X_test2, y_train2, y_test2 = train_test_split(X2, y1, test_size=0.30, random_sta

# Create dictionaries for final graph//

# Use: methodDict['Stacking'] = accuracy_score
methodDict = {}
rmseDict = ()
```

In [36]:

```
def evalClassModel(model, y_test2, y_pred_class2, plot=False):
    #Classification accuracy: percentage of correct predictions
    # calculate accuracy
    print('Accuracy:', metrics.accuracy_score(y_test2, y_pred_class2))

    confusion = metrics.confusion_matrix(y_test2, y_pred_class2)
    #[row, column]
    TP = confusion[1, 1]
    TN = confusion[0, 0]
    FP = confusion[0, 1]
    FN = confusion[1, 0]

    sns.heatmap(confusion,annot=True,fmt="d")
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

    accuracy = metrics.accuracy_score(y_test2, y_pred_class2)
    print('Error:', 1 - metrics.accuracy_score(y_test2, y_pred_class2))

    false_positive_rate = FP / float(TN + FP)

    #print('False Positive Rate:', false_positive_rate)

    #Precision: When a positive value is predicted, how often is the prediction correct?
    print('Precision:', metrics.precision_score(y_test2, y_pred_class2))


    # IMPORTANT: first argument is true values, second argument is predicted probabilitie
    print('AUC Score:', metrics.roc_auc_score(y_test2, y_pred_class2))

    # calculate cross-validated AUC
    print('Cross-validated AUC:', cross_val_score(model, X2, y1, cv=10, scoring='roc_auc')

    model.predict_proba(X_test2)[0:10, 1]

    y_pred_prob2 = model.predict_proba(X_test2)[:, 1]


    y_pred_prob2 = y_pred_prob2.reshape(-1,1)
    y_pred_class2 = binarize(y_pred_prob2)[0]



    roc_auc = metrics.roc_auc_score(y_test2, y_pred_prob2)


    predict_mine2 = np.where(y_pred_prob2 > 0.50, 1, 0)
    confusion = metrics.confusion_matrix(y_test2, predict_mine2)

    return accuracy
```

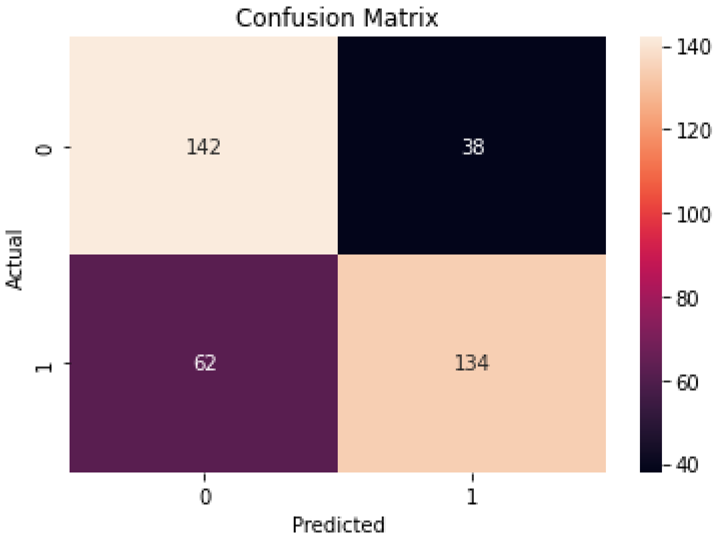
In [37]:

```
import sklearn.metrics as metrics

def logisticRegression2():
    logreg2 = LogisticRegression(C=60)
    logreg2.fit(X_train2, y_train2)

    # make class predictions for the testing set
    y_pred_class2 = logreg2.predict(X_test2)

    print('Logistic Regression before feature selection -')

    accuracy_score = evalClassModel(logreg2, y_test2, y_pred_class2)

    #Data for final graph
    methodDict['Log. Regres.'] = accuracy_score * 100
```

In [38]:

```
logisticRegression2()
```

```
Logistic Regression before feature selection -
Accuracy: 0.7340425531914894
```



```
Error: 0.26595744680851063
Precision: 0.7790697674418605
AUC Score: 0.736281179138322
Cross-validated AUC: 0.7635268722772028
```

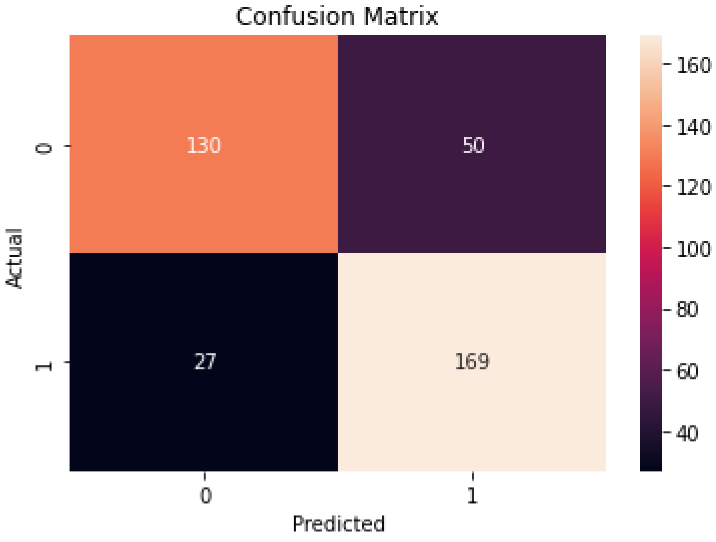
In [39]:

```python
import sklearn.metrics as metrics

def SVM_C2():
    svm_c2 = svm.SVC(probability = True)
    svm_c2.fit(X_train2, y_train2)

    # make class predictions for the testing set
    y_pred_class2 = svm_c2.predict(X_test2)

    print('Support Vector Machine before feature selection -')

    accuracy_score = evalClassModel(svm_c2, y_test2, y_pred_class2)

    #Data for final graph
    #methodDict['Log. Regres.'] = accuracy_score * 100
```

In [40]:

```python
SVM_C2()
```

```
Support Vector Machine before feature selection -
Accuracy: 0.7952127659574468
```



```
Error: 0.20478723404255317
Precision: 0.771689497716895
```

```
AUC Score: 0.7922335600907029
Cross-validated AUC: 0.8614626090168132
```

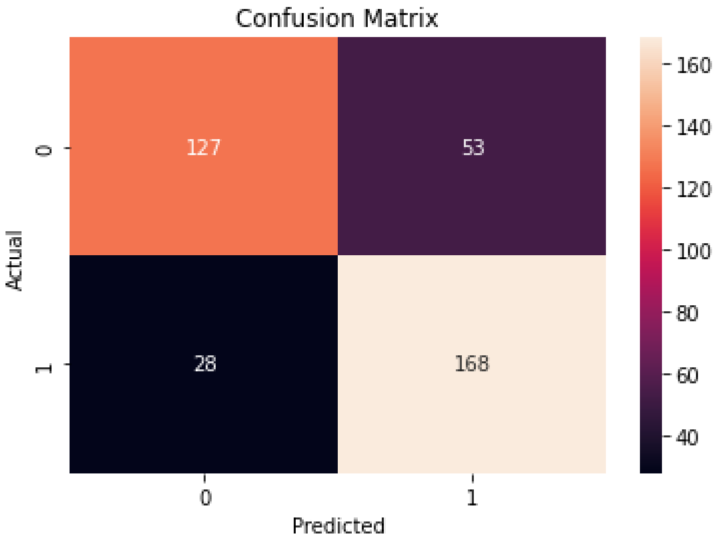
In [41]:

```
def randomForest2():
    # Calculating the best parameters
    forest2 = RandomForestClassifier(n_estimators = 100)

    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(forest2, param_dist)

    forest2 = RandomForestClassifier(max_depth = None, min_samples_leaf=8, min_samples_spl
    my_forest2 = forest2.fit(X_train2, y_train2)

    y_pred_class2 = my_forest2.predict(X_test2)

    print('Random Forest before feature selection-')
    accuracy_score = evalClassModel(my_forest2, y_test2, y_pred_class2, True)
```

In [42]:

```
randomForest2()
```

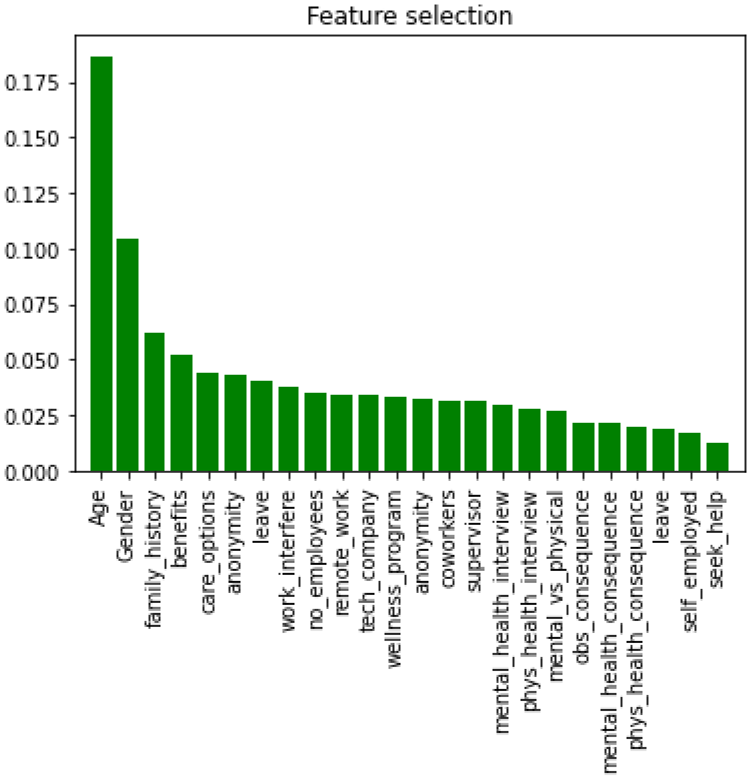
```
Random Forest before feature selection-
Accuracy: 0.7845744680851063
```



```
Error: 0.21542553191489366
Precision: 0.7601809954751131
AUC Score: 0.7813492063492063
Cross-validated AUC: 0.8726061287594538
```

In [43]:

```
from sklearn.ensemble import ExtraTreesClassifier

forest = ExtraTreesClassifier(n_estimators=10, random_state=1111)

forest.fit(X2, y1)
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(X2.shape[1]):
    labels.append(feature_cols[f])

#plt.figure(figsize=(12,24))
plt.title("Feature selection")
plt.bar(range(X2.shape[1]), importances[indices], color="g", align="center")
plt.xticks(range(X2.shape[1]), labels, rotation='vertical')
plt.xlim([-1, X2.shape[1]])
plt.show()
```



In [44]:

```
def evalClassModel(model, y_test1, y_pred_class, plot=False):
    #Classification accuracy: percentage of correct predictions
    # calculate accuracy
    print('Accuracy:', metrics.accuracy_score(y_test1, y_pred_class))

    confusion = metrics.confusion_matrix(y_test1, y_pred_class)
    #[row, column]
    TP = confusion[1, 1]
    TN = confusion[0, 0]
    FP = confusion[0, 1]
    FN = confusion[1, 0]

    sns.heatmap(confusion,annot=True,fmt="d")
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

    accuracy = metrics.accuracy_score(y_test1, y_pred_class)

    print('Error:', 1 - metrics.accuracy_score(y_test1, y_pred_class))

    false_positive_rate = FP / float(TN + FP)

    #print('False Positive Rate:', false_positive_rate)

    #Precision: When a positive value is predicted, how often is the prediction correct?
    print('Precision:', metrics.precision_score(y_test1, y_pred_class))
    # IMPORTANT: first argument is true values, second argument is predicted probabilitie
    print('AUC Score:', metrics.roc_auc_score(y_test1, y_pred_class))

    # calculate cross-validated AUC
    print('Cross-validated AUC:', cross_val_score(model, X1, y1, cv=10, scoring='roc_auc')

    #model.predict_proba(X_test1)[0:10, 1]

    y_pred_prob = model.predict_proba(X_test1)[:, 1]


    y_pred_prob = y_pred_prob.reshape(-1,1)


    roc_auc = metrics.roc_auc_score(y_test1, y_pred_prob)


    predict_mine = np.where(y_pred_prob > 0.50, 1, 0)
    confusion = metrics.confusion_matrix(y_test1, predict_mine)

    return accuracy
```

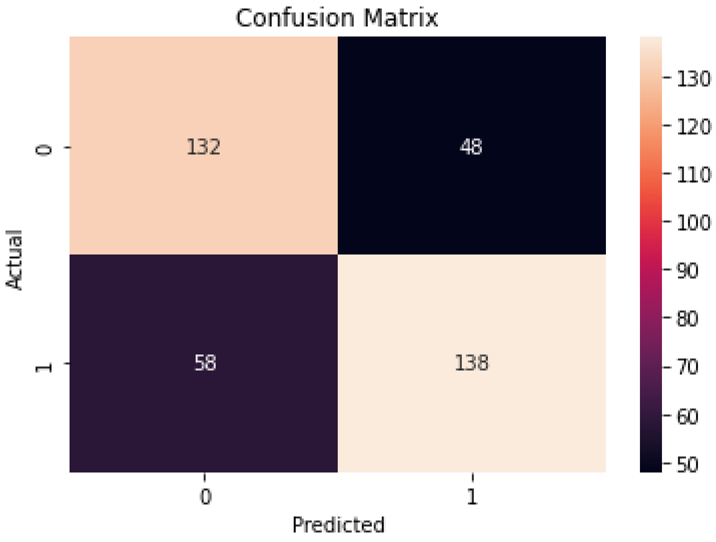
In [45]:

```
import sklearn.metrics as metrics

def logisticRegression():
    logreg = LogisticRegression(C=60)
    logreg.fit(X_train1, y_train1)

    # make class predictions for the testing set
    y_pred_class = logreg.predict(X_test1)

    print('Logistic Regression after feature selection -')

    accuracy_score = evalClassModel(logreg, y_test1, y_pred_class)

    #Data for final graph
    #methodDict['Log. Regres.'] = accuracy_score * 100
```

In [46]:

```
logisticRegression()
```

```
Logistic Regression after feature selection -
Accuracy: 0.7180851063829787
```



```
Error: 0.28191489361702127
Precision: 0.7419354838709677
```

```
AUC Score: 0.7187074829931973
Cross-validated AUC: 0.7547748891994661
```

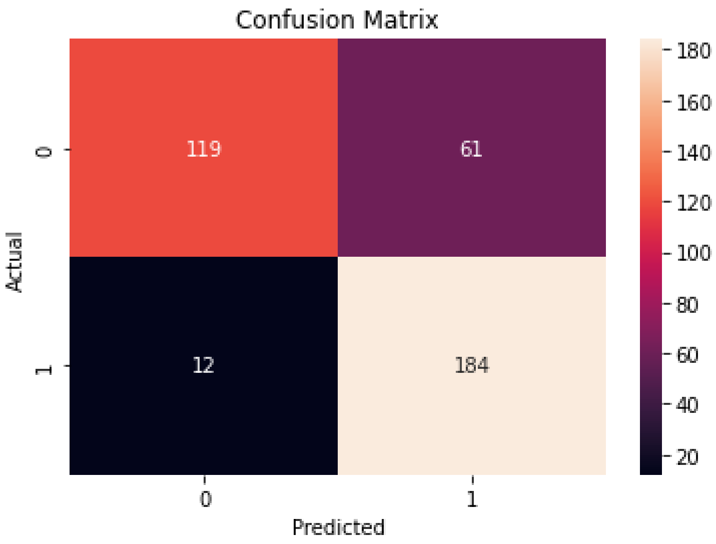
In [47]:

```python
import sklearn.metrics as metrics

def SVM_C():
    svm_c = svm.SVC(probability = True)
    svm_c.fit(X_train1, y_train1)

    # make class predictions for the testing set
    y_pred_class = svm_c.predict(X_test1)

    print('Support Vector Machine after feature selection -')

    accuracy_score = evalClassModel(svm_c, y_test1, y_pred_class)

    #Data for final graph
    #methodDict['Log. Regres.'] = accuracy_score * 100
```

In [48]:

```python
SVM_C()
```

```
Support Vector Machine after feature selection -
Accuracy: 0.8058510638297872
```


Confusion Matrix

```
Error: 0.19414893617021278
Precision: 0.7510204081632653
AUC Score: 0.7999433106575964
Cross-validated AUC: 0.8495489612974575
```

In [49]:

```python
from sklearn.model_selection import RandomizedSearchCV

def tuningRandomizedSearchCV(model, param_dist):
    rand = RandomizedSearchCV(model, param_dist, cv=10, scoring='accuracy', n_iter=10, ran
    rand.fit(X1, y1)
    #rand.grid_scores_
```

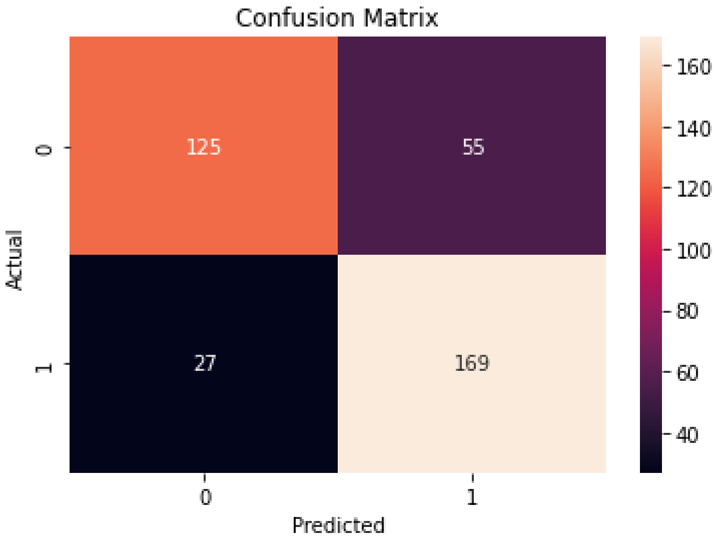
In [50]:

```python
def randomForest():
    # Calculating the best parameters
    forest = RandomForestClassifier(n_estimators = 100)

    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(forest, param_dist)

    forest = RandomForestClassifier(max_depth = None, min_samples_leaf=8, min_samples_spli
    my_forest = forest.fit(X_train1, y_train1)

    y_pred_class = my_forest.predict(X_test1)

    print('Random Forest after feature selection-')
    accuracy_score = evalClassModel(my_forest, y_test1, y_pred_class, True)
```

In [51]:

```
randomForest()
```

```
Random Forest after feature selection-
Accuracy: 0.7819148936170213
```



```
Error: 0.21808510638297873
Precision: 0.7544642857142857
AUC Score: 0.778344671201814
Cross-validated AUC: 0.8903591567565241
```

In [ ]: