DSC550 Week 12 Final Project

Name: Madhuri Basava

Date: 11/16/2023

Milestone 1

Import the necessary libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import binarize, LabelEncoder, MinMaxScaler
from sklearn.model_selection import RandomizedSearchCV

# Validation libraries
from sklearn import metrics
from sklearn.metrics import accuracy_score, mean_squared_error, precision_recall_cu
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score

from scipy.stats import randint

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

Read the csv file into a data frame

In [2]:
```
df = pd.read_csv("survey.csv")
df
```

Out[2]:

	Timestamp	Age	Gender	Country	state	self_employed	family_history	treatment	work_i
0	2014-08-27 11:29:31	37	Female	United States	IL	NaN	No	Yes	
1	2014-08-27 11:29:37	44	M	United States	IN	NaN	No	No	
2	2014-08-27 11:29:44	32	Male	Canada	NaN	NaN	No	No	
3	2014-08-27 11:29:46	31	Male	United Kingdom	NaN	NaN	Yes	Yes	
4	2014-08-27 11:30:22	31	Male	United States	TX	NaN	No	No	
...	...	...	...	...	...	...	...	...	
1254	2015-09-12 11:17:21	26	male	United Kingdom	NaN	No	No	Yes	
1255	2015-09-26 01:07:35	32	Male	United States	IL	No	Yes	Yes	
1256	2015-11-07 12:36:58	34	male	United States	CA	No	Yes	Yes	So
1257	2015-11-30 21:25:06	46	f	United States	NC	No	No	No	
1258	2016-02-01 23:04:31	25	Male	United States	IL	No	Yes	Yes	So

1259 rows × 27 columns

In [3]:

```
print(df.shape)
```

```
(1259, 27)
```

In [4]:

```
# Describe the data
print(df.describe())
```

```
                Age
count  1.259000e+03
mean   7.942815e+07
std    2.818299e+09
min   -1.726000e+03
25%    2.700000e+01
50%    3.100000e+01
75%    3.600000e+01
max    1.000000e+11
```

In [5]:

```
# Print the info
print(df.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
None
```

In [6]:
```
# copy the data frame into a new data frame to perform cleaning in the next Milesto
df1=df
df1
```

Out[6]:

	Timestamp	Age	Gender	Country	state	self_employed	family_history	treatment	work_i
0	2014-08-27 11:29:31	37	Female	United States	IL	NaN	No	Yes	
1	2014-08-27 11:29:37	44	M	United States	IN	NaN	No	No	
2	2014-08-27 11:29:44	32	Male	Canada	NaN	NaN	No	No	
3	2014-08-27 11:29:46	31	Male	United Kingdom	NaN	NaN	Yes	Yes	
4	2014-08-27 11:30:22	31	Male	United States	TX	NaN	No	No	
...	...	...	...	...	...	...	...	...	
1254	2015-09-12 11:17:21	26	male	United Kingdom	NaN	No	No	Yes	
1255	2015-09-26 01:07:35	32	Male	United States	IL	No	Yes	Yes	
1256	2015-11-07 12:36:58	34	male	United States	CA	No	Yes	Yes	So
1257	2015-11-30 21:25:06	46	f	United States	NC	No	No	No	
1258	2016-02-01 23:04:31	25	Male	United States	IL	No	Yes	Yes	So

1259 rows × 27 columns

Clean 'Gender' column

In [7]:
```
#lower case all columm's elements
gender = df['Gender'].str.lower()
#print(gender)

#Select unique elements
gender = df['Gender'].unique()

#Made gender groups
male_str = ["male", "m", "male-ish", "maile", "mal", "male (cis)", "make", "male ",
trans_str = ["trans-female", "something kinda male?", "queer/she/they", "non-binary
female_str = ["cis female", "f", "female", "woman",  "femake", "female ","cis-femal

for (row, col) in df.iterrows():

    if str.lower(col.Gender) in male_str:
        df['Gender'].replace(to_replace=col.Gender, value='male', inplace=True)

    if str.lower(col.Gender) in female_str:
        df['Gender'].replace(to_replace=col.Gender, value='female', inplace=True)
```

```
    if str.lower(col.Gender) in trans_str:
        df['Gender'].replace(to_replace=col.Gender, value='trans', inplace=True)

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

print(df['Gender'].unique())
```

```
['female' 'male' 'trans']
```

Replace missing age with mean value

In [8]:

```
#complete missing age with mean
df['Age'].fillna(df['Age'].median(), inplace = True)

# Fill with median() values < 18 and > 120
s = pd.Series(df['Age'])
s[s<18] = df['Age'].median()
df['Age'] = s
s = pd.Series(df['Age'])
s[s>120] = df['Age'].median()
df['Age'] = s

#Ranges of Age
df['age_range'] = (pd.cut(df['Age'], [0,20,30,65,100], labels=["0-20", "21-30", "31
df
```

Out[8]:

	Timestamp	Age	Gender	Country	state	self_employed	family_history	treatment	work_i
0	2014-08-27 11:29:31	37	female	United States	IL	NaN	No	Yes	
1	2014-08-27 11:29:37	44	male	United States	IN	NaN	No	No	
2	2014-08-27 11:29:44	32	male	Canada	NaN	NaN	No	No	
3	2014-08-27 11:29:46	31	male	United Kingdom	NaN	NaN	Yes	Yes	
4	2014-08-27 11:30:22	31	male	United States	TX	NaN	No	No	
...	...	...	...	...	...	...	...	...	
1254	2015-09-12 11:17:21	26	male	United Kingdom	NaN	No	No	Yes	
1255	2015-09-26 01:07:35	32	male	United States	IL	No	Yes	Yes	
1256	2015-11-07 12:36:58	34	male	United States	CA	No	Yes	Yes	So
1257	2015-11-30 21:25:06	46	female	United States	NC	No	No	No	
1258	2016-02-01 23:04:31	25	male	United States	IL	No	Yes	Yes	So

1257 rows × 28 columns

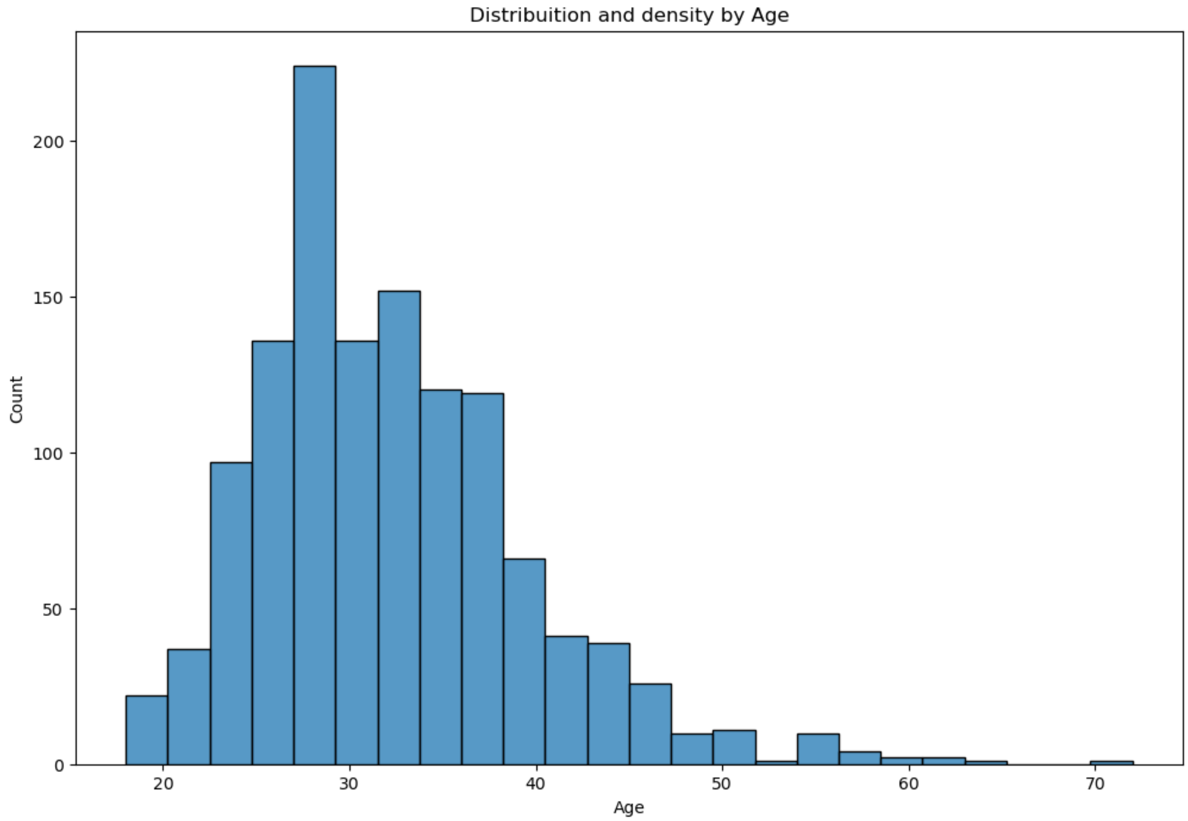
Graphical analysis creating a minimum of four graphs.

1) Distribiution and density by Age

In [9]:

```
plt.figure(figsize=(12,8))
sns.histplot(df["Age"], bins=24)
plt.title("Distribiution and density by Age")
plt.xlabel("Age")
```

Out[9]: Text(0.5, 0, 'Age')



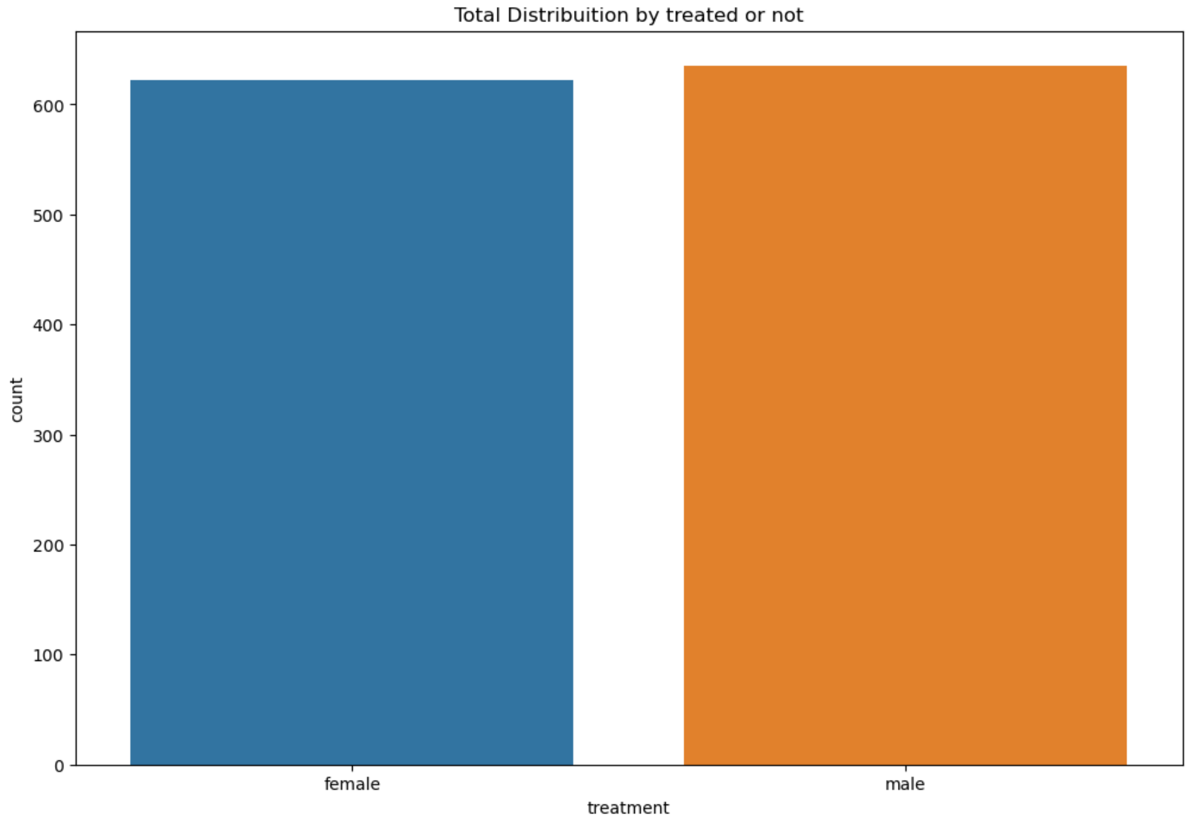
Looks like teenage individuals have more mental health problems

How many people has been treated?

In [10]:
```
from sklearn import preprocessing
labelDict = {}
for feature in df:
    le = preprocessing.LabelEncoder()
    le.fit(df[feature])
    le_name_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
    df[feature] = le.transform(df[feature])
    # Get labels
    labelKey = 'label_' + feature
    labelValue = [*le_name_mapping]
    labelDict[labelKey] =labelValue
```

In [11]:
```
plt.figure(figsize=(12,8))
labels = labelDict['label_Gender'][0:2]
g = sns.countplot(x="treatment", data=df)
g.set_xticklabels(labels)

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

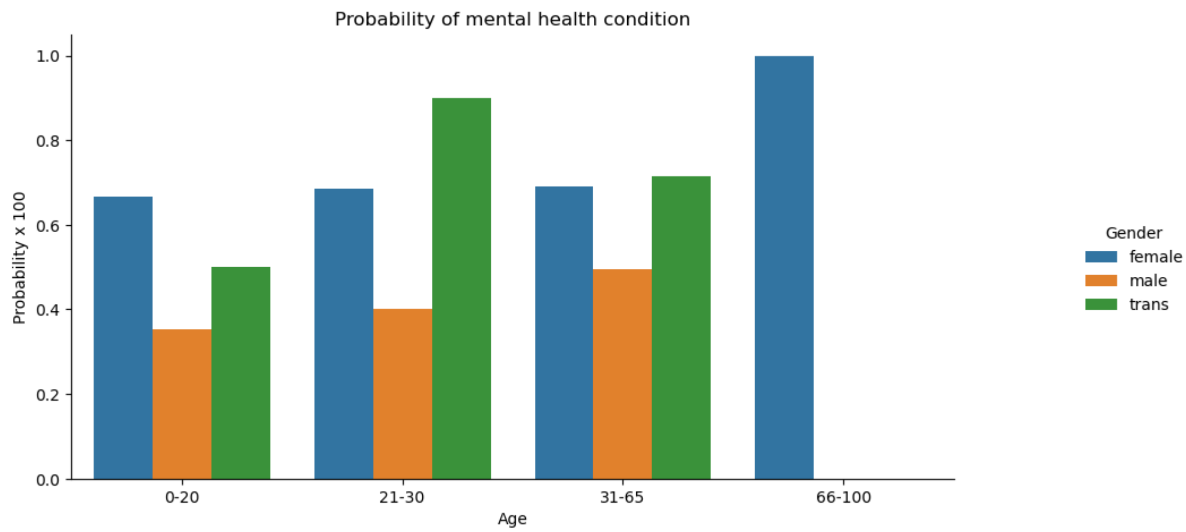
Out[11]:
```
Text(0.5, 1.0, 'Total Distribuition by treated or not')
```



From the above graph we can see that number of Males who took treatment is little more than the number number females who took treatment

Nested barplot to show probabilities for Age and gender

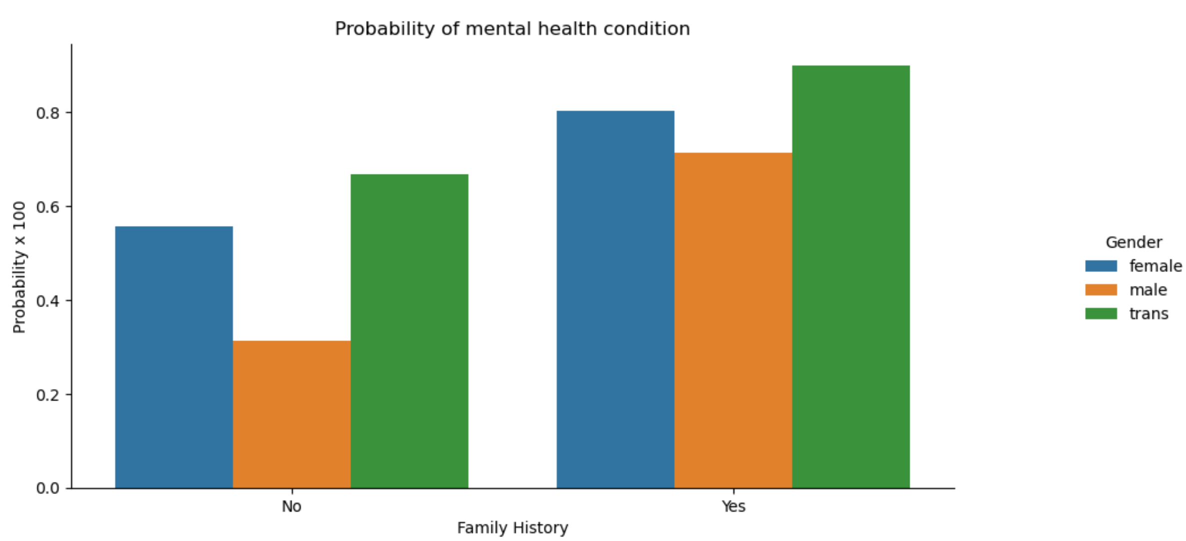
In [12]:
```
o = labelDict['label_age_range']

g = sns.factorplot(x="age_range", y="treatment", hue="Gender", data=df, kind="bar",
g.set_xticklabels(o)

plt.title('Probability of mental health condition')
plt.ylabel('Probability x 100')
plt.xlabel('Age')
# replace legend labels

new_labels = labelDict['label_Gender']
for t, l in zip(g._legend.texts, new_labels): t.set_text(l)

# Positioning the legend
g.fig.subplots_adjust(top=0.9,right=0.8)

plt.show()
```



From the above graph, we can conclude that males from ages 66 and above does not have metal health issues.

Barplot to show probabilities for family history

In [13]:
```
o = labelDict['label_family_history']
g = sns.factorplot(x="family_history", y="treatment", hue="Gender", data=df, kind="
g.set_xticklabels(o)
plt.title('Probability of mental health condition')
plt.ylabel('Probability x 100')
plt.xlabel('Family History')

# replace legend labels
new_labels = labelDict['label_Gender']
for t, l in zip(g._legend.texts, new_labels): t.set_text(l)

# Positioning the legend
g.fig.subplots_adjust(top=0.9,right=0.8)

plt.show()
```

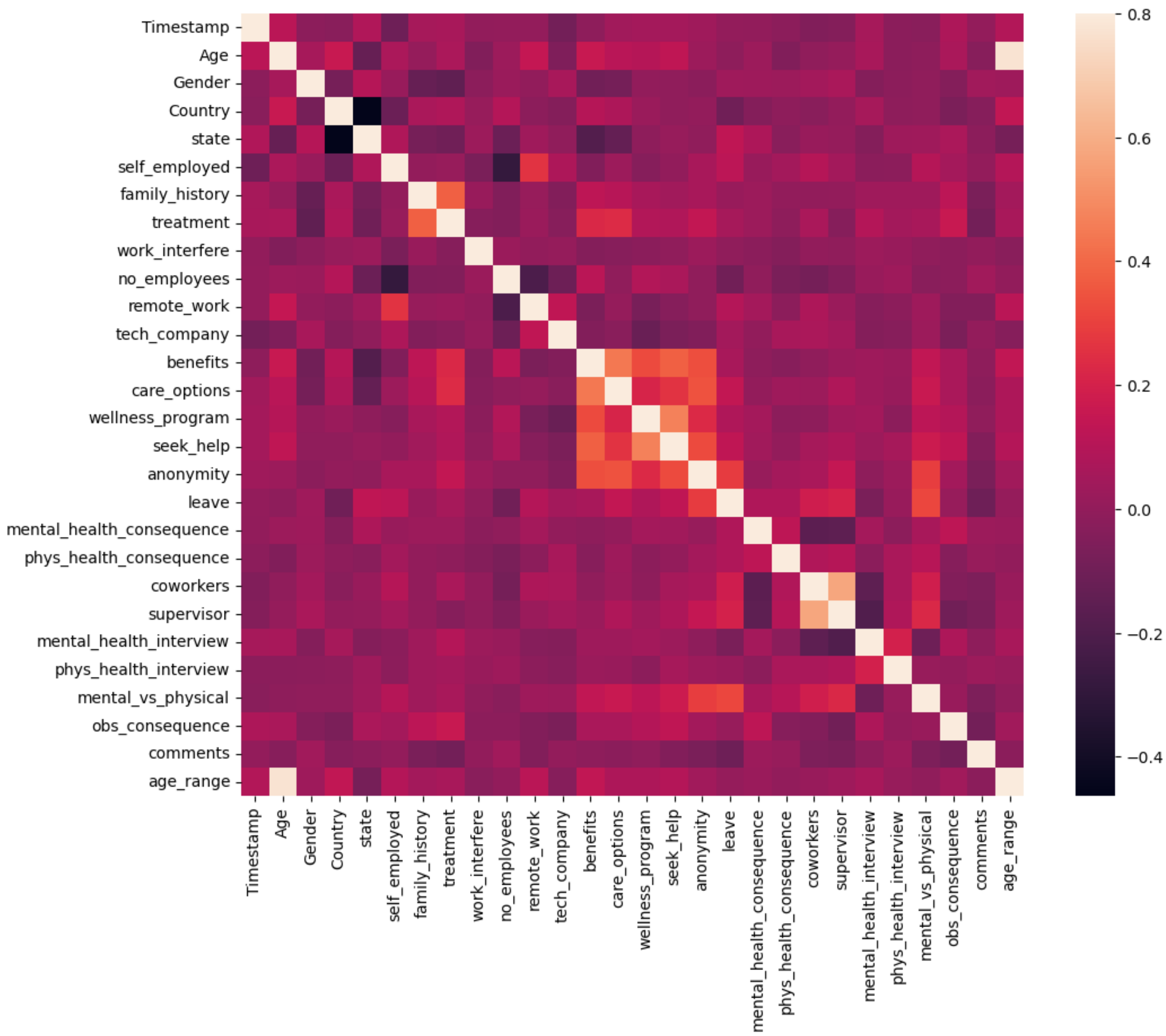


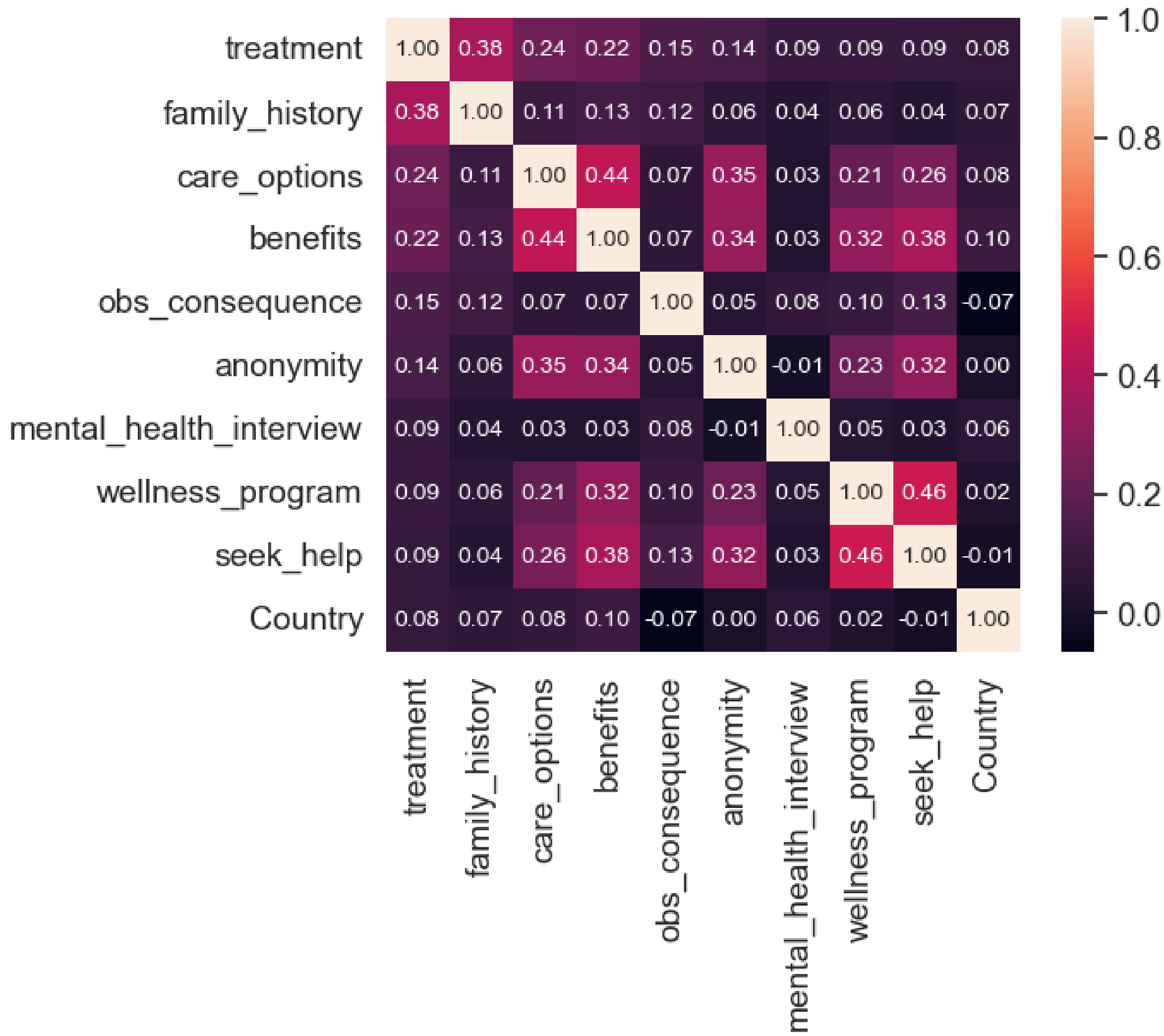
From the above graph we can conclude that the individuals with a family history of mental health issues are more susceptible to the mental health condition.

Correlation Matrix

In [14]:

```
corrmat = df.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat, vmax=.8, square=True);
plt.show()

#treatment correlation matrix
k = 10 #number of variables for heatmap
cols = corrmat.nlargest(k, 'treatment')['treatment'].index
cm = np.corrcoef(df[cols].values.T)
sns.set(font_scale=1.25)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'siz
plt.show()
```





Milestone 2

Drop any features that are not useful for your model building and explain why they are not useful.

In [15]:
```
# Dropping the unneccessary variables "Timestamp",“comments”, country and “state” a
df1 = df1.drop(['comments'], axis= 1)
df1 = df1.drop(['state'], axis= 1)
df1 = df1.drop(['Country'], axis= 1)
df1 = df1.drop(['Timestamp'], axis= 1)
df1
```

Out[15]:

	Age	Gender	self_employed	family_history	treatment	work_interfere	no_employees	remo
0	37	female	NaN	No	Yes	Often	6-25	
1	44	male	NaN	No	No	Rarely	More than 1000	
2	32	male	NaN	No	No	Rarely	6-25	
3	31	male	NaN	Yes	Yes	Often	26-100	
4	31	male	NaN	No	No	Never	100-500	
...	...	...	...	...	...	...	...	
1254	26	male	No	No	Yes	NaN	26-100	
1255	32	male	No	Yes	Yes	Often	26-100	
1256	34	male	No	Yes	Yes	Sometimes	More than 1000	
1257	46	female	No	No	No	NaN	100-500	
1258	25	male	No	Yes	Yes	Sometimes	26-100	

1259 rows × 23 columns

Treatment is the target variable that we will drop later from the dataset and split the data into training and testing data and build a model

In [16]:
```
# Checking the missing data
df1.isnull().sum().max()
```

Out[16]: 264

Data Cleaning

Check the missing data

In [17]:
```
total = df1.isnull().sum().sort_values(ascending=False)
percent = (df1.isnull().sum()/df1.isnull().count()).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data.head(20)
print(missing_data)
```

```
                           Total   Percent
work_interfere               264  0.209690
self_employed                 18  0.014297
Age                            0  0.000000
anonymity                      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
leave                          0  0.000000
wellness_program               0  0.000000
seek_help                      0  0.000000
Gender                         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
obs_consequence                0  0.000000
```

Replace missing age with mean value

In [18]:
```
#complete missing age with mean
df1['Age'].fillna(df1['Age'].median(), inplace = True)

# Fill with median() values < 18 and > 120
s = pd.Series(df1['Age'])
s[s<18] = df1['Age'].median()
df['Age'] = s
s = pd.Series(df1['Age'])
s[s>120] = df1['Age'].median()
df1['Age'] = s

#Ranges of Age
df1['age_range'] = (pd.cut(df1['Age'], [0,20,30,65,100], labels=["0-20", "21-30", "
df1
```

Out[18]:

	Age	Gender	self_employed	family_history	treatment	work_interfere	no_employees	remo
0	37	female	NaN	No	Yes	Often	6-25	
1	44	male	NaN	No	No	Rarely	More than 1000	
2	32	male	NaN	No	No	Rarely	6-25	
3	31	male	NaN	Yes	Yes	Often	26-100	
4	31	male	NaN	No	No	Never	100-500	
...	...	...	...	...	...	...	...	
1254	26	male	No	No	Yes	NaN	26-100	
1255	32	male	No	Yes	Yes	Often	26-100	
1256	34	male	No	Yes	Yes	Sometimes	More than 1000	
1257	46	female	No	No	No	NaN	100-500	
1258	25	male	No	Yes	Yes	Sometimes	26-100	

1259 rows × 24 columns

Clean 'Gender' column

In [19]:

```
#lower case all columm's elements
gender = df1['Gender'].str.lower()
#print(gender)

#Select unique elements
gender = df1['Gender'].unique()

#Made gender groups
male_str = ["male", "m", "male-ish", "maile", "mal", "male (cis)", "make", "male ",
trans_str = ["trans-female", "something kinda male?", "queer/she/they", "non-binary
female_str = ["cis female", "f", "female", "woman",  "femake", "female ","cis-femal

for (row, col) in df1.iterrows():

    if str.lower(col.Gender) in male_str:
        df1['Gender'].replace(to_replace=col.Gender, value='male', inplace=True)

    if str.lower(col.Gender) in female_str:
        df1['Gender'].replace(to_replace=col.Gender, value='female', inplace=True)
```

```
    if str.lower(col.Gender) in trans_str:
        df1['Gender'].replace(to_replace=col.Gender, value='trans', inplace=True)

#Get rid of bullshit
stk_list = ['A little about you', 'p']
df1 = df1[~df1['Gender'].isin(stk_list)]

print(df1['Gender'].unique())
```

```
['female' 'male' 'trans']
```

Replace "NaN" string to 'NO' for self_employed column

In [20]:
```
defaultString = 'NaN'  # Assign default values for each data type
stringFeatures = ['Gender', 'Country', 'self_employed', 'family_history', 'treatmen
                 'no_employees', 'remote_work', 'tech_company', 'anonymity', 'leave
                 'phys_health_consequence', 'coworkers', 'supervisor', 'mental_heal
                 'mental_vs_physical', 'obs_consequence', 'benefits', 'care_options
                 'seek_help']
for feature in df1:
    if feature in stringFeatures:
        df1[feature] = df1[feature].fillna(defaultString)

df1['self_employed'] = df1['self_employed'].replace([defaultString], 'No')
print(df1['self_employed'].unique())
```

```
['No' 'Yes']
```

Replace "NaN" string to "Don't know" for work_interfere column

In [21]:
```
df1['work_interfere'] = df1['work_interfere'].replace([defaultString], 'Don\'t know
print(df1['work_interfere'].unique())
```

```
['Often' 'Rarely' 'Never' 'Sometimes' "Don't know"]
```

Recheck the missing data

In [22]:
```
total = df1.isnull().sum().sort_values(ascending=False)
percent = (df1.isnull().sum()/df1.isnull().count()).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data.head(20)
print(missing_data)
```

```
                           Total  Percent
Age                            0      0.0
Gender                         0      0.0
obs_consequence                0      0.0
mental_vs_physical             0      0.0
phys_health_interview          0      0.0
mental_health_interview        0      0.0
supervisor                     0      0.0
coworkers                      0      0.0
phys_health_consequence        0      0.0
mental_health_consequence      0      0.0
leave                          0      0.0
anonymity                      0      0.0
seek_help                      0      0.0
wellness_program               0      0.0
care_options                   0      0.0
benefits                       0      0.0
tech_company                   0      0.0
remote_work                    0      0.0
no_employees                   0      0.0
work_interfere                 0      0.0
treatment                      0      0.0
family_history                 0      0.0
self_employed                  0      0.0
age_range                      0      0.0
```

In [23]:

```
print(df1.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1257 entries, 0 to 1258
Data columns (total 24 columns):
 #   Column                     Non-Null Count  Dtype 
---  ------                     --------------  ----- 
 0   Age                        1257 non-null   int64 
 1   Gender                     1257 non-null   object
 2   self_employed              1257 non-null   object
 3   family_history             1257 non-null   object
 4   treatment                  1257 non-null   object
 5   work_interfere             1257 non-null   object
 6   no_employees               1257 non-null   object
 7   remote_work                1257 non-null   object
 8   tech_company               1257 non-null   object
 9   benefits                   1257 non-null   object
 10  care_options               1257 non-null   object
 11  wellness_program           1257 non-null   object
 12  seek_help                  1257 non-null   object
 13  anonymity                  1257 non-null   object
 14  leave                      1257 non-null   object
 15  mental_health_consequence  1257 non-null   object
 16  phys_health_consequence    1257 non-null   object
 17  coworkers                  1257 non-null   object
 18  supervisor                 1257 non-null   object
 19  mental_health_interview    1257 non-null   object
 20  phys_health_interview      1257 non-null   object
 21  mental_vs_physical         1257 non-null   object
 22  obs_consequence            1257 non-null   object
 23  age_range                  1257 non-null   object
dtypes: int64(1), object(23)
memory usage: 245.5+ KB
None
```

Encoding Data

In [24]:

```
labelDict = {}
for feature in df1:
    le = preprocessing.LabelEncoder()
    le.fit(df1[feature])
    le_name_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
    df1[feature] = le.transform(df1[feature])
    # Get labels
    labelKey = 'label_' + feature
    labelValue = [*le_name_mapping]
    labelDict[labelKey] =labelValue

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

```
label_Age [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, 5
7, 58, 60, 61, 62, 65, 72]
label_Gender ['female', 'male', 'trans']
label_self_employed ['No', 'Yes']
label_family_history ['No', 'Yes']
label_treatment ['No', 'Yes']
label_work_interfere ["Don't know", 'Never', 'Often', 'Rarely', 'Sometimes']
label_no_employees ['1-5', '100-500', '26-100', '500-1000', '6-25', 'More than 100
0']
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 difficul
t', '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']
label_age_range ['0-20', '21-30', '31-65', '66-100']
```

In [25]:
```
df1
```

Out[25]:

	Age	Gender	self_employed	family_history	treatment	work_interfere	no_employees	remo
0	19	0	0	0	1	2	4	
1	26	1	0	0	0	3	5	
2	14	1	0	0	0	3	4	
3	13	1	0	1	1	2	2	
4	13	1	0	0	0	1	1	
...	...	...	...	...	...	...	...	
1254	8	1	0	0	1	0	2	
1255	14	1	0	1	1	2	2	
1256	16	1	0	1	1	4	5	
1257	28	0	0	0	0	0	1	
1258	7	1	0	1	1	4	2	

1257 rows × 24 columns

Milestone 3

Scaling and fitting

In [26]:
```
# Scaling the Age feature since it is different from the other features
scaler = MinMaxScaler()
df1['Age'] = scaler.fit_transform(df1[['Age']])
df1.head()
```

Out[26]:

	Age	Gender	self_employed	family_history	treatment	work_interfere	no_employees	rem
0	0.431818	0	0	0	1	2	4	
1	0.590909	1	0	0	0	3	5	
2	0.318182	1	0	0	0	3	4	
3	0.295455	1	0	1	1	2	2	
4	0.295455	1	0	0	0	1	1	

5 rows × 24 columns

Splitting the data into training and test tests

In [27]:
```
# select few important columns by analysing the correlation chart above.
feature_cols = ['Age', 'Gender', 'family_history', 'benefits', 'care_options', 'ano
X = df1[feature_cols]
y = df1.treatment   # Since treatment is the target variable

# split X and y into training and testing sets in 70/30 ratio
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_st

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

Evaluate Classification model

In [28]:
```
def evalClassModel(model, y_test, y_pred_class, plot=False):
    # Classification accuracy is the percentage of correct predictions
    # calculate accuracy
    print('Accuracy:', metrics.accuracy_score(y_test, y_pred_class))

    # Null accuracy: accuracy that could be achieved by always predicting the most
    # examine the class distribution of the testing set (using a Pandas Series meth
    print('Null accuracy:\n', y_test.value_counts())

    # Calculate the percentage of ones
    print('Percentage of ones:', y_test.mean())
```

```
# Calculate the percentage of zeros
print('Percentage of zeros:',1 - y_test.mean())

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

# Conclusion:
# Classification accuracy is the easiest classification metric to understand
# But, it does not tell about the underlying distribution of response values
# And, it does not tell about what "types" of errors the classifier is making

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

# Metrics computed from a confusion matrix
# Classification Accuracy: Overall, how often is the classifier correct?
accuracy = metrics.accuracy_score(y_test, y_pred_class)
print('Classification Accuracy:', accuracy)

#Classification Error: Overall, how often is the classifier incorrect?
print('Classification Error:', 1 - metrics.accuracy_score(y_test, y_pred_class)

#False Positive Rate: When the actual value is negative, how often is the predi
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 cor
print('Precision:', metrics.precision_score(y_test, y_pred_class))


# IMPORTANT: first argument is true values, second argument is predicted probab
print('AUC Score:', metrics.roc_auc_score(y_test, y_pred_class))

# calculate cross-validated AUC
print('Cross-validated AUC:', cross_val_score(model, X, y, cv=10, scoring='roc_

########################################
# Adjusting the classification threshold
########################################

# print the first 10 predicted responses
```

```
    # 1D array (vector) of binary values (0, 1)
    print('First 10 predicted responses:\n', model.predict(X_test)[0:10])

    # print the first 10 predicted probabilities of class membership
    print('First 10 predicted probabilities of class members:\n', model.predict_pro

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

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

    if plot == True:
        # histogram of predicted probabilities
        # adjust the font size
        plt.rcParams['font.size'] = 12
        # 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')


    # predict treatment if the predicted probability is greater than 0.3
    # it will return 1 for all values above 0.3 and 0 otherwise
    # results are 2D so we slice out the first column
    y_pred_prob = y_pred_prob.reshape(-1,1)
    # y_pred_class = binarize(y_pred_prob, 0.3)[0]
    y_pred_class = binarize(y_pred_prob)

    # print the first 10 predicted probabilities
    print('First 10 predicted probabilities:\n', y_pred_prob[0:10])

    #########################################
    # ROC Curves and Area Under the Curve (AUC)
    #########################################

    # AUC is the percentage of the ROC plot that is underneath the curve
    # Higher value = better classifier
    roc_auc = metrics.roc_auc_score(y_test, y_pred_prob)


    # first argument is true values, second argument is predicted probabilities
    # we pass y_test and y_pred_prob
    # we do not use y_pred_class, because it will give incorrect results without ge
    # roc_curve returns 3 objects fpr, tpr, thresholds
    # fpr: false positive rate
    # tpr: true positive rate
    fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_prob)
    if plot == True:
        plt.figure()

        plt.plot(fpr, tpr, color='darkorange', label='ROC curve (area = %0.2f)' % r
```

```
        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()

    # Define evaluate_threshold function that accepts a threshold and prints sensit
    def evaluate_threshold(threshold):
        # Sensitivity: When the actual value is positive, how often is the predicti
        # Specificity: When the actual value is negative, how often is the predicti
        print('Sensitivity for ' + str(threshold) + ' :', tpr[thresholds > threshol
        print('Specificity for ' + str(threshold) + ' :', 1 - fpr[thresholds > thre

    # One way of setting the threshold
    predict_mine = np.where(y_pred_prob > 0.50, 1, 0)
    confusion = metrics.confusion_matrix(y_test, predict_mine)
    print(confusion)

    return accuracy
```

We want to compare different algorithms to select the best-performing model.

So, evaluating three models (Logistic Regression, KNeighbors Classifier and Random Forests) here.

Tuning with RandomizedSearchCV

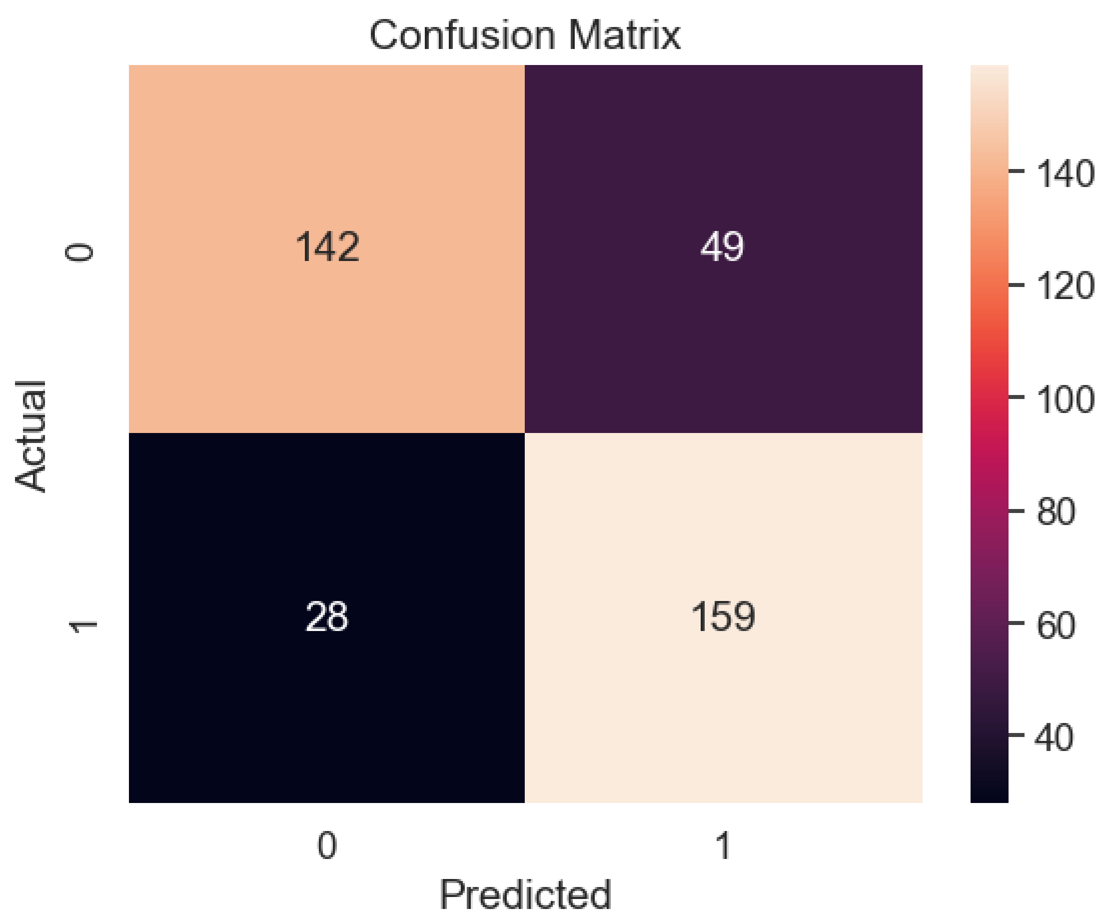
In [29]:
```
def tuningRandomizedSearchCV(model, param_dist):
    # Searching multiple parameters simultaneously
    # n_iter controls the number of searches
    rand = RandomizedSearchCV(model, param_dist, cv=10, scoring='accuracy', n_iter=
    rand.fit(X, y)
    rand.cv_results_

    # 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_i
        rand.fit(X, y)
        best_scores.append(round(rand.best_score_, 3))
    print(best_scores)
```

1) Evaluating Logistic Regression

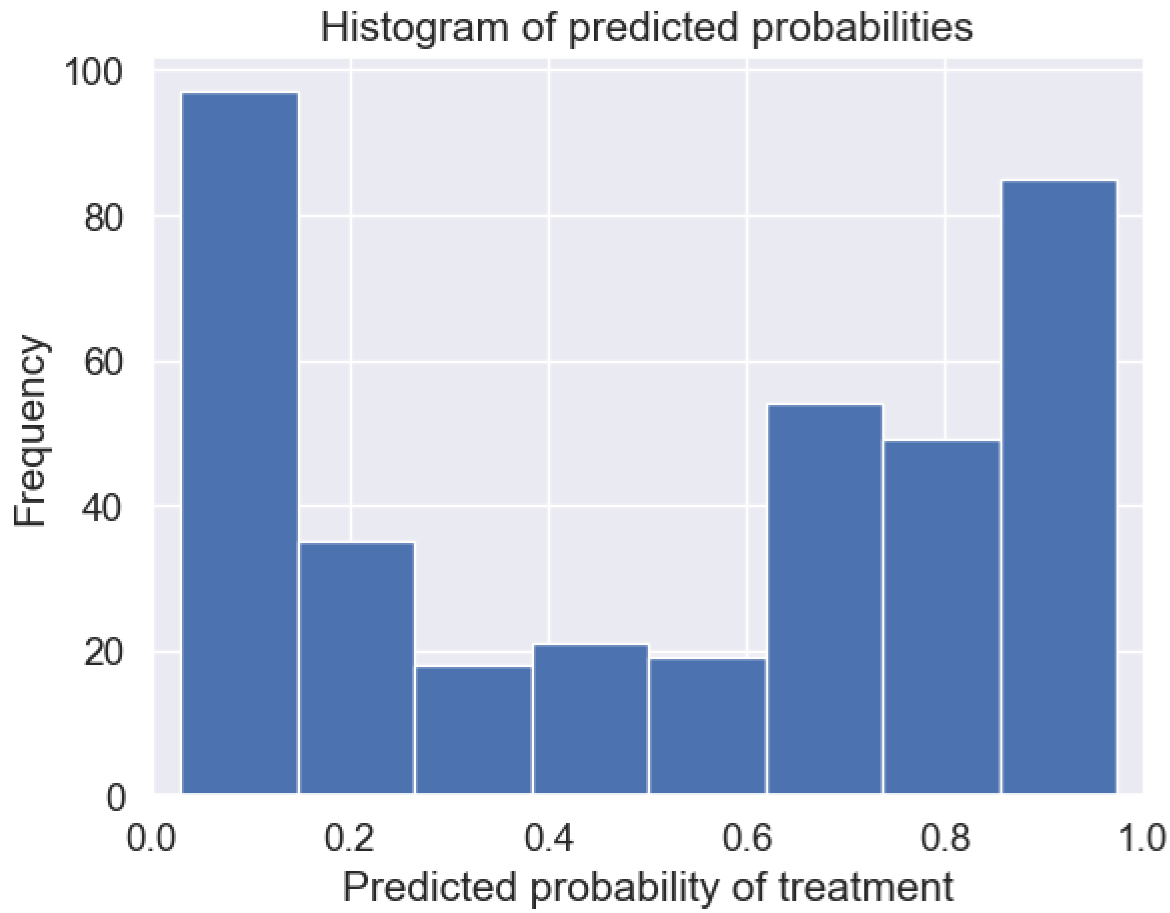
In [30]:
```
def logisticRegression():
    # Train a logistic regression model on the training set
    logreg = LogisticRegression()
    logreg.fit(X_train, y_train)

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

    print('########## Logistic Regression ##############')

    accuracy_score = evalClassModel(logreg, y_test, y_pred_class, True)

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

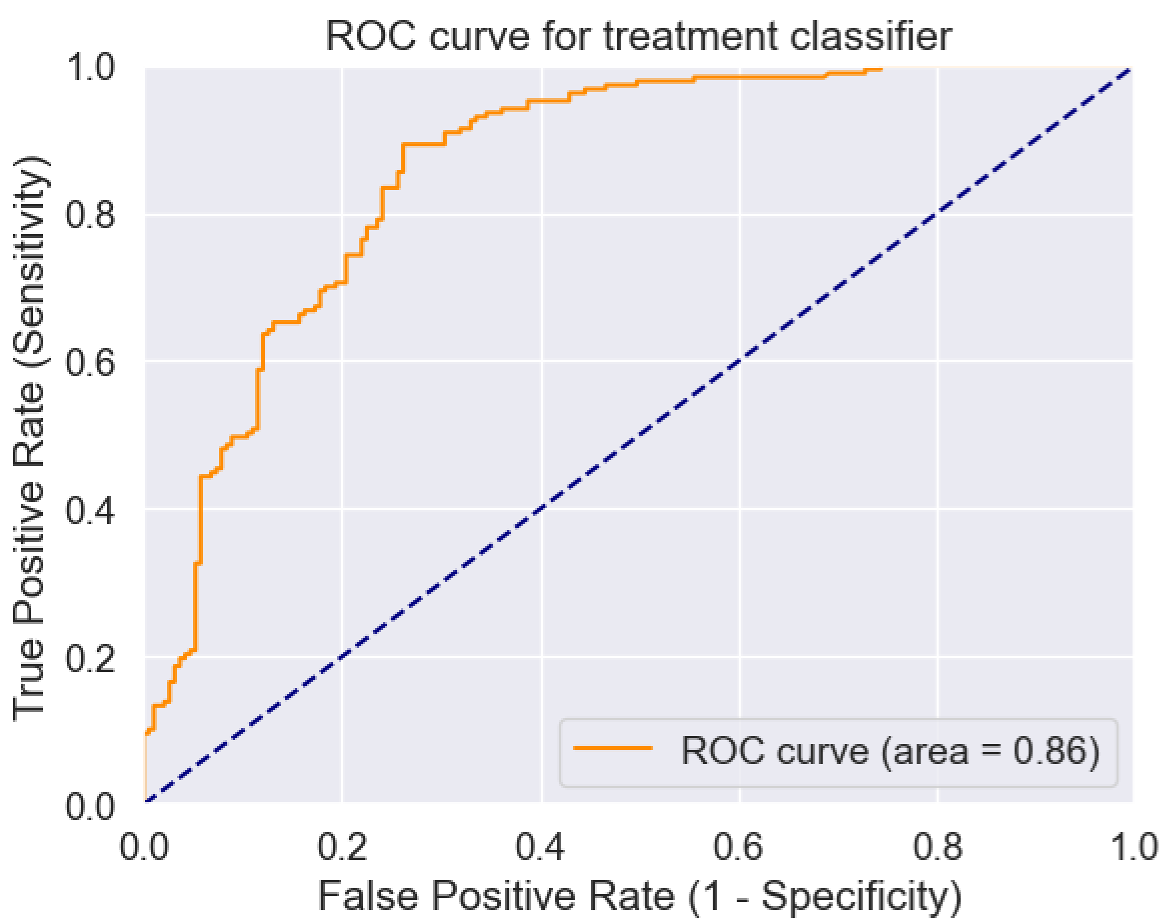
In [31]:
```
logisticRegression()
```

```
########## Logistic Regression ##############
Accuracy: 0.7962962962962963
Null accuracy:
 0    191
1    187
Name: treatment, dtype: int64
Percentage of ones: 0.4947089947089947
Percentage of zeros: 0.5052910052910053
True: [0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1 0 0]
Pred: [1 0 0 0 1 1 0 1 0 1 0 1 1 0 1 1 1 1 0 0 0 0 1 0 0]
```



```
Classification Accuracy: 0.7962962962962963
Classification Error: 0.20370370370370372
False Positive Rate: 0.25654450261780104
Precision: 0.7644230769230769
AUC Score: 0.7968614385306716
Cross-validated AUC: 0.8753623882722146
First 10 predicted responses:
 [1 0 0 0 1 1 0 1 0 1]
First 10 predicted probabilities of class members:
 [[0.09193053 0.90806947]
 [0.95991564 0.04008436]
 [0.96547467 0.03452533]
 [0.78757121 0.21242879]
 [0.38959922 0.61040078]
 [0.05264207 0.94735793]
 [0.75035574 0.24964426]
 [0.19065116 0.80934884]
 [0.61612081 0.38387919]
 [0.47699963 0.52300037]]
First 10 predicted probabilities:
 [[0.90806947]
 [0.04008436]
 [0.03452533]
 [0.21242879]
 [0.61040078]
 [0.94735793]
 [0.24964426]
 [0.80934884]
 [0.38387919]
 [0.52300037]]
```


Histogram of predicted probabilities
Frequency
100
80
60
40
20
0
0.0
0.2
0.4
0.6
0.8
1.0
Predicted probability of treatment


ROC curve for treatment classifier
True Positive Rate (Sensitivity)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate (1 - Specificity)
ROC curve (area = 0.86)

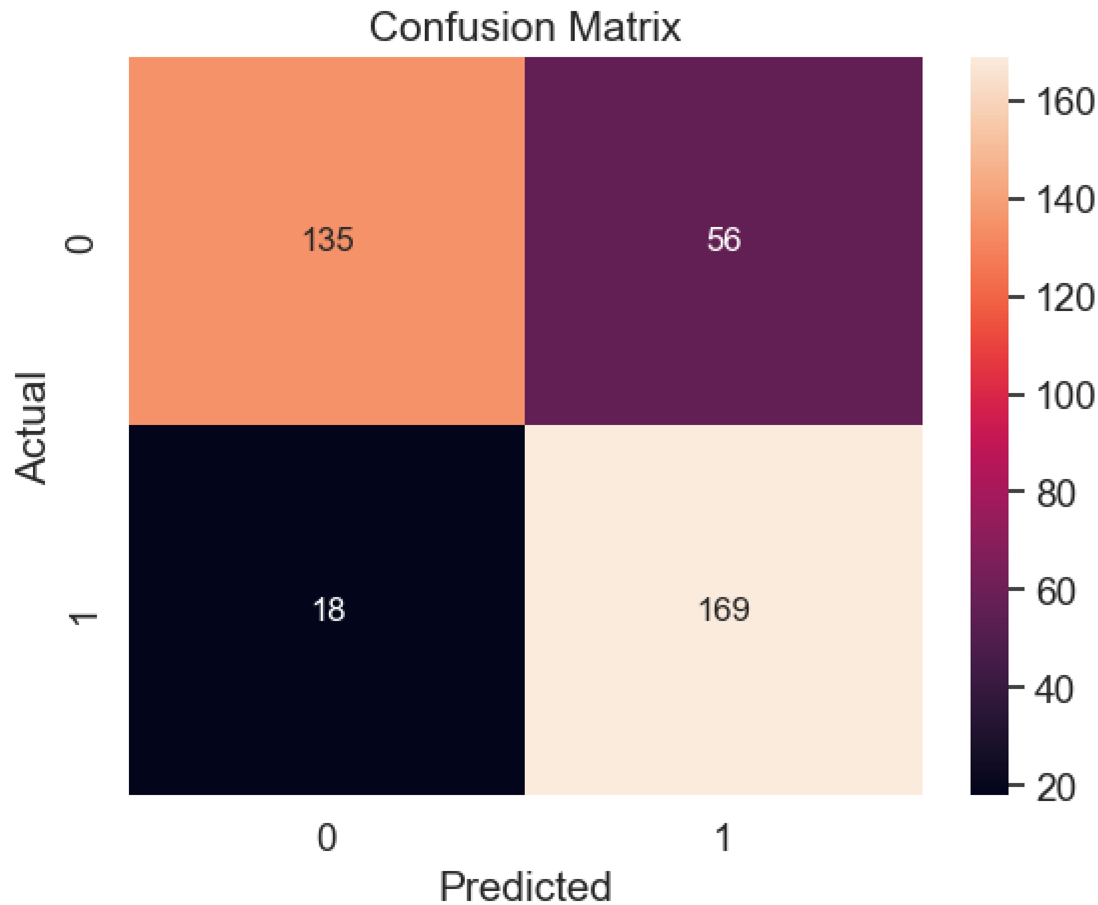
```
[[142  49]
 [ 28 159]]
```

2) Evaluating KNeighbors Classifier

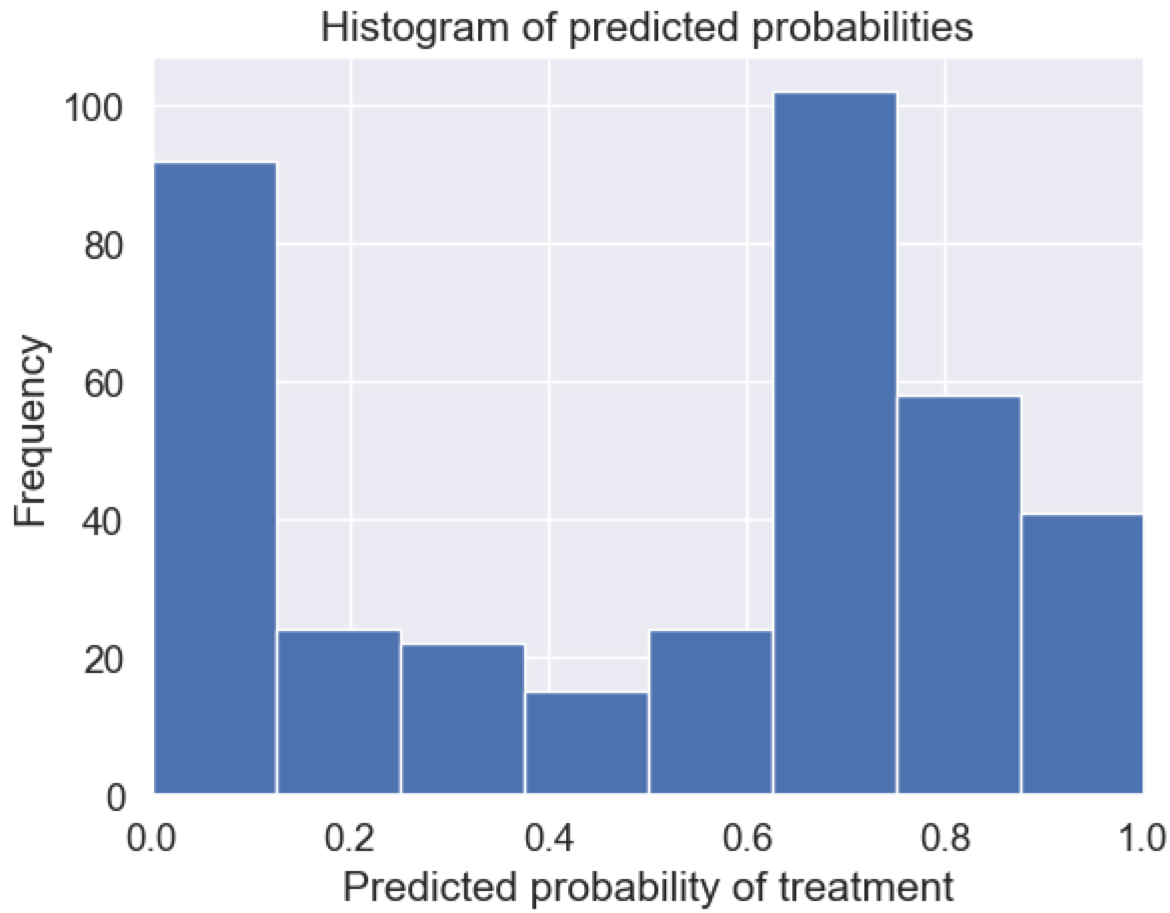
In [32]:
```
def Knn():
    # Calculating the best parameters
    knn = KNeighborsClassifier(n_neighbors=5)

    # define the parameter values that should be searched
    k_range = list(range(1, 31))
    weight_options = ['uniform', 'distance']

    # specify "parameter distributions" rather than a "parameter grid"
    param_dist = dict(n_neighbors=k_range, weights=weight_options)
    tuningRandomizedSearchCV(knn, param_dist)

    # train a KNeighborsClassifier model on the training set
    knn = KNeighborsClassifier(n_neighbors=27, weights='uniform')
    knn.fit(X_train, y_train)

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

    print('########### KNeighborsClassifier ###############')

    accuracy_score = evalClassModel(knn, y_test, y_pred_class, True)

    #Data for final graph
    methodDict['KNN'] = accuracy_score * 100
```

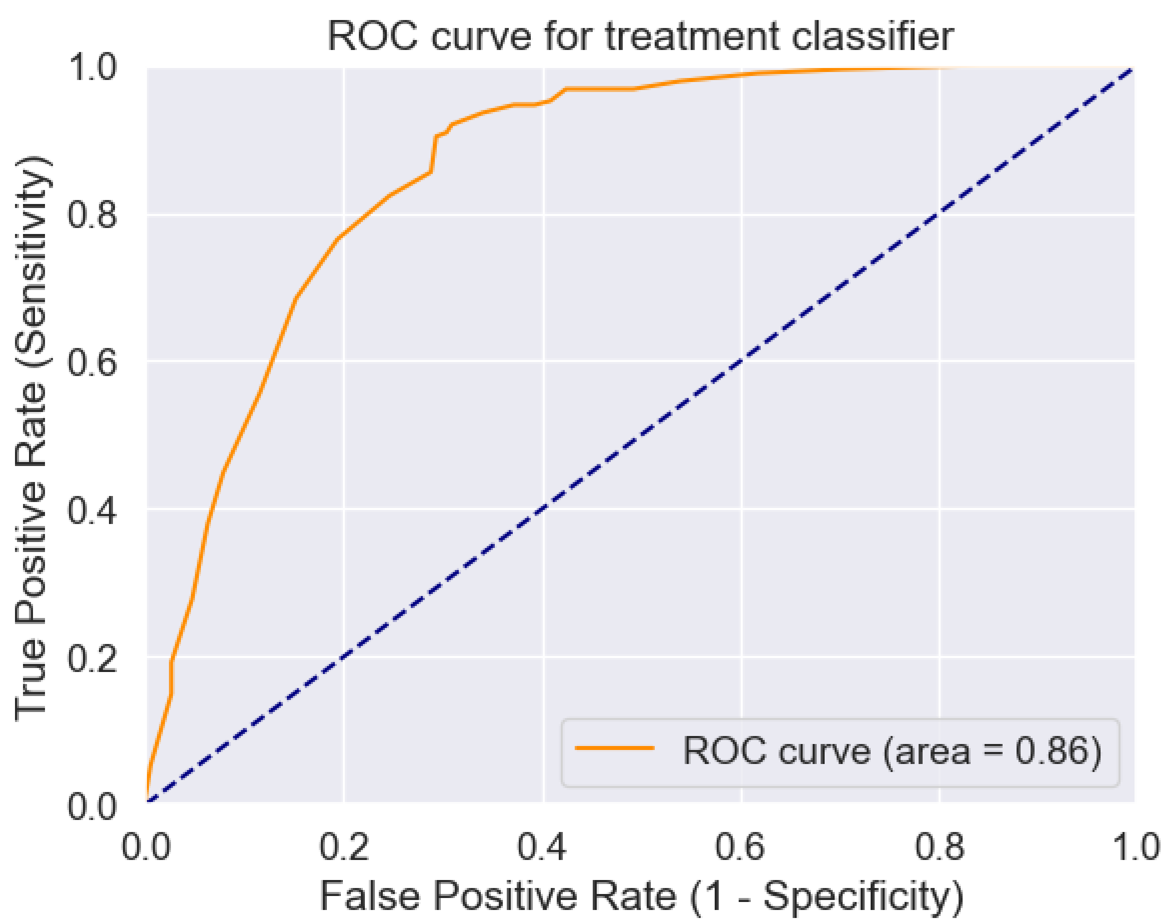
In [33]:
```
Knn()
```

```
Rand. Best Score:  0.8217650793650794
Rand. Best Params:  {'weights': 'uniform', 'n_neighbors': 27}
[0.822, 0.817, 0.814, 0.822, 0.805, 0.814, 0.819, 0.814, 0.816, 0.817, 0.819, 0.81
7, 0.816, 0.819, 0.81, 0.817, 0.814, 0.822, 0.819, 0.816]
########### KNeighborsClassifier ###############
Accuracy: 0.8042328042328042
Null accuracy:
 0    191
1    187
Name: treatment, dtype: int64
Percentage of ones: 0.4947089947089947
Percentage of zeros: 0.5052910052910053
True: [0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1 0 0]
Pred: [1 0 0 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 0 0 0 0 1 0 0]
```


Confusion Matrix
135
56
18
169
Actual
0
1
Predicted
0
1
160
140
120
100
80
60
40
20

```
Classification Accuracy: 0.8042328042328042
Classification Error: 0.1957671957671958
False Positive Rate: 0.2931937172774869
Precision: 0.7511111111111111
AUC Score: 0.8052747991152673
Cross-validated AUC: 0.8784644661702792
First 10 predicted responses:
 [1 0 0 0 1 1 0 1 1 1]
First 10 predicted probabilities of class members:
 [[0.33333333 0.66666667]
 [1.         0.        ]
 [1.         0.        ]
 [0.66666667 0.33333333]
 [0.37037037 0.62962963]
 [0.03703704 0.96296296]
 [0.59259259 0.40740741]
 [0.37037037 0.62962963]
 [0.33333333 0.66666667]
 [0.33333333 0.66666667]]
First 10 predicted probabilities:
 [[0.66666667]
 [0.        ]
 [0.        ]
 [0.33333333]
 [0.62962963]
 [0.96296296]
 [0.40740741]
 [0.62962963]
 [0.66666667]
 [0.66666667]]
```


Histogram of predicted probabilities
Frequency
100
80
60
40
20
0
0.0
0.2
0.4
0.6
0.8
1.0
Predicted probability of treatment


ROC curve for treatment classifier
True Positive Rate (Sensitivity)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate (1 - Specificity)
ROC curve (area = 0.86)

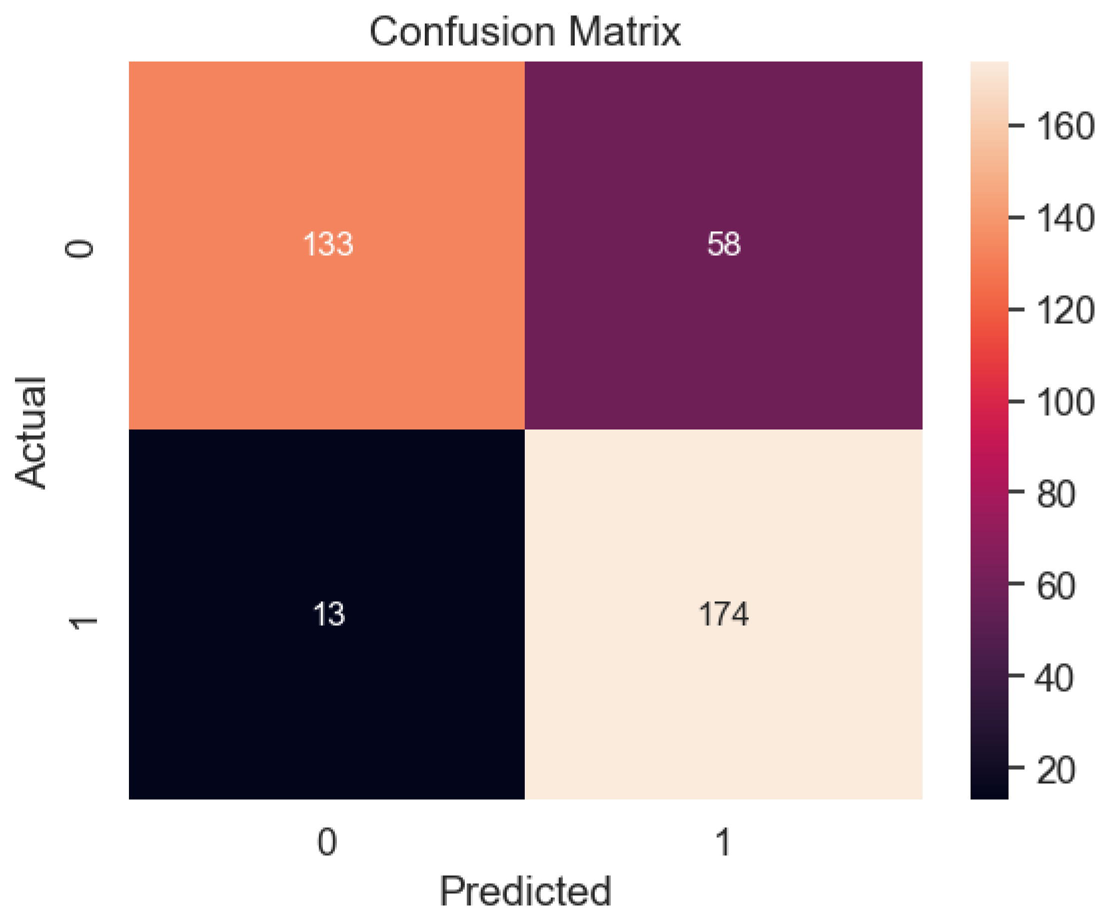
```
[[135  56]
 [ 18 169]]
```

3) Evaluating Random Forests

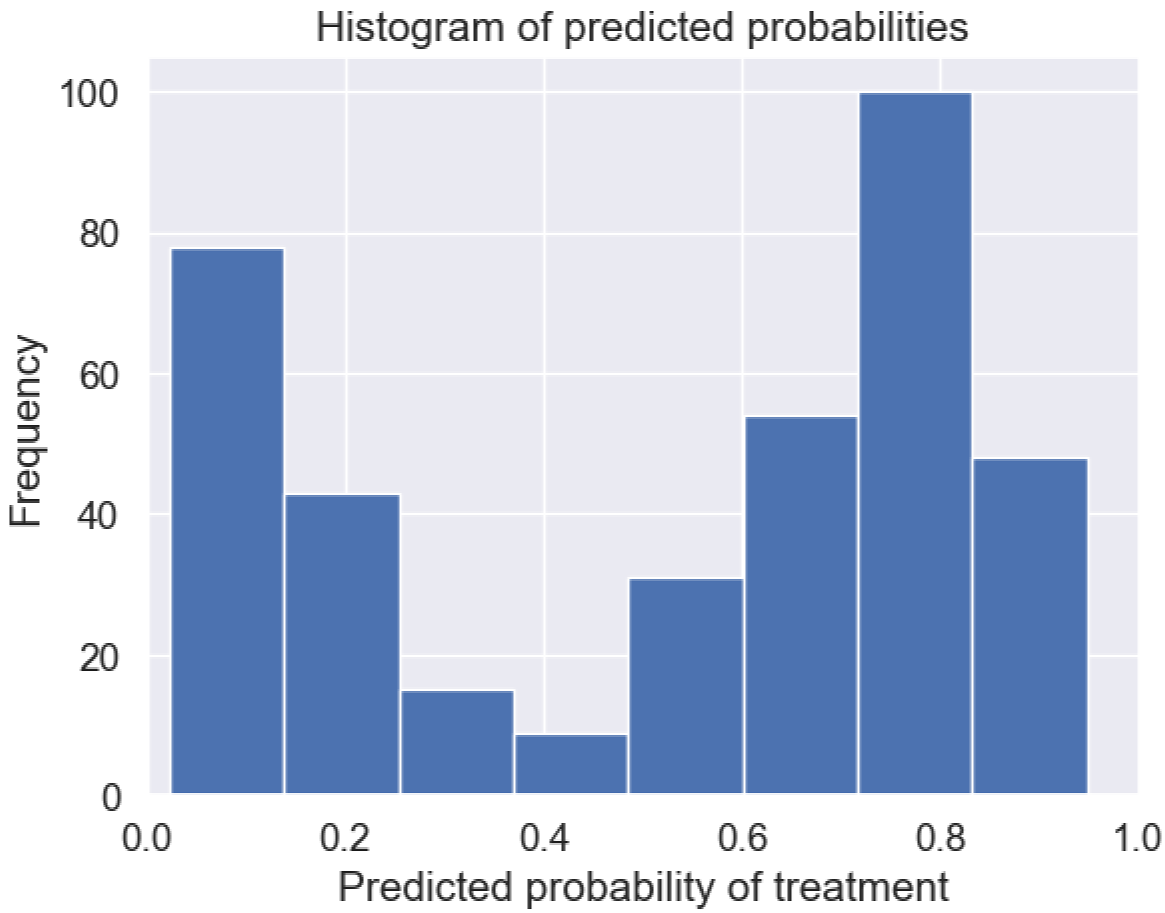
In [34]:
```
def randomForest():
    # Calculating the best parameters
    forest = RandomForestClassifier(n_estimators = 20)

    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)

    # Building and fitting my_forest
    forest = RandomForestClassifier(max_depth = None, min_samples_leaf=8, min_sampl
    my_forest = forest.fit(X_train, y_train)

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

    print('########### Random Forests ###############')

    accuracy_score = evalClassModel(my_forest, y_test, y_pred_class, True)

    #Data for final graph
    methodDict['R. Forest'] = accuracy_score * 100
```

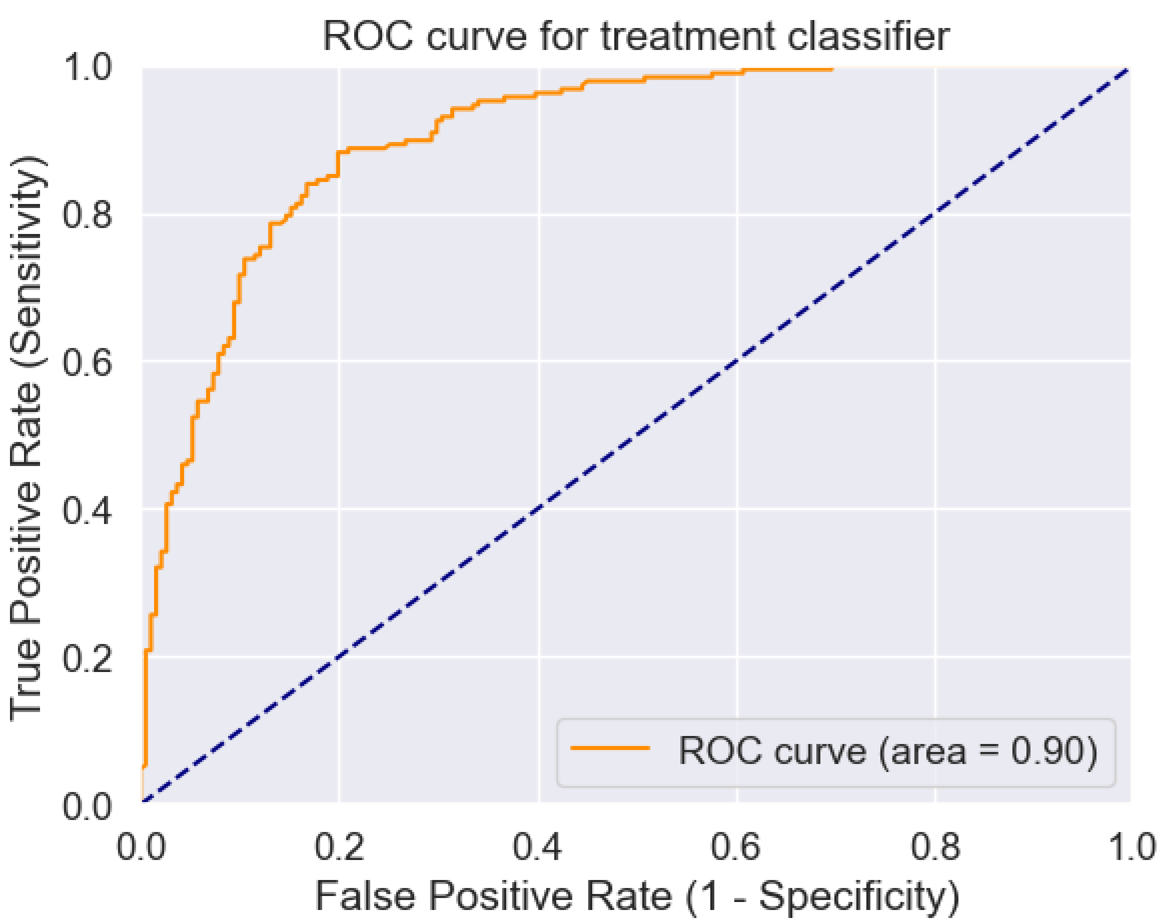
In [35]:
```
randomForest()
```

```
Rand. Best Score:  0.8305206349206349
Rand. Best Params:  {'criterion': 'entropy', 'max_depth': 3, 'max_features': 6, 'm
in_samples_leaf': 7, 'min_samples_split': 8}
[0.831, 0.831, 0.831, 0.831, 0.831, 0.831, 0.831, 0.832, 0.831, 0.831, 0.831, 0.83
1, 0.831, 0.831, 0.831, 0.831, 0.834, 0.831, 0.834, 0.831]
########### Random Forests ###############
Accuracy: 0.8121693121693122
Null accuracy:
 0    191
1    187
Name: treatment, dtype: int64
Percentage of ones: 0.4947089947089947
Percentage of zeros: 0.5052910052910053
True: [0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1 0 0]
Pred: [1 0 0 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 0 0 0 0 1 0 0]
```


Confusion Matrix
Actual
0
1
133
58
13
174
0
1
Predicted
160
140
120
100
80
60
40
20

```
Classification Accuracy: 0.8121693121693122
Classification Error: 0.1878306878306878
False Positive Rate: 0.3036649214659686
Precision: 0.75
AUC Score: 0.8134081809782457
Cross-validated AUC: 0.8934280651104528
First 10 predicted responses:
 [1 0 0 0 1 1 0 1 1 1]
First 10 predicted probabilities of class members:
 [[0.2555794  0.7444206 ]
 [0.95069083 0.04930917]
 [0.93851009 0.06148991]
 [0.87096597 0.12903403]
 [0.40653554 0.59346446]
 [0.17282958 0.82717042]
 [0.89450448 0.10549552]
 [0.4065912  0.5934088 ]
 [0.20540631 0.79459369]
 [0.19337644 0.80662356]]
First 10 predicted probabilities:
 [[0.7444206 ]
 [0.04930917]
 [0.06148991]
 [0.12903403]
 [0.59346446]
 [0.82717042]
 [0.10549552]
 [0.5934088 ]
 [0.79459369]
 [0.80662356]]
```


Histogram of predicted probabilities
Frequency
Predicted probability of treatment
0
20
40
60
80
100
0.0
0.2
0.4
0.6
0.8
1.0

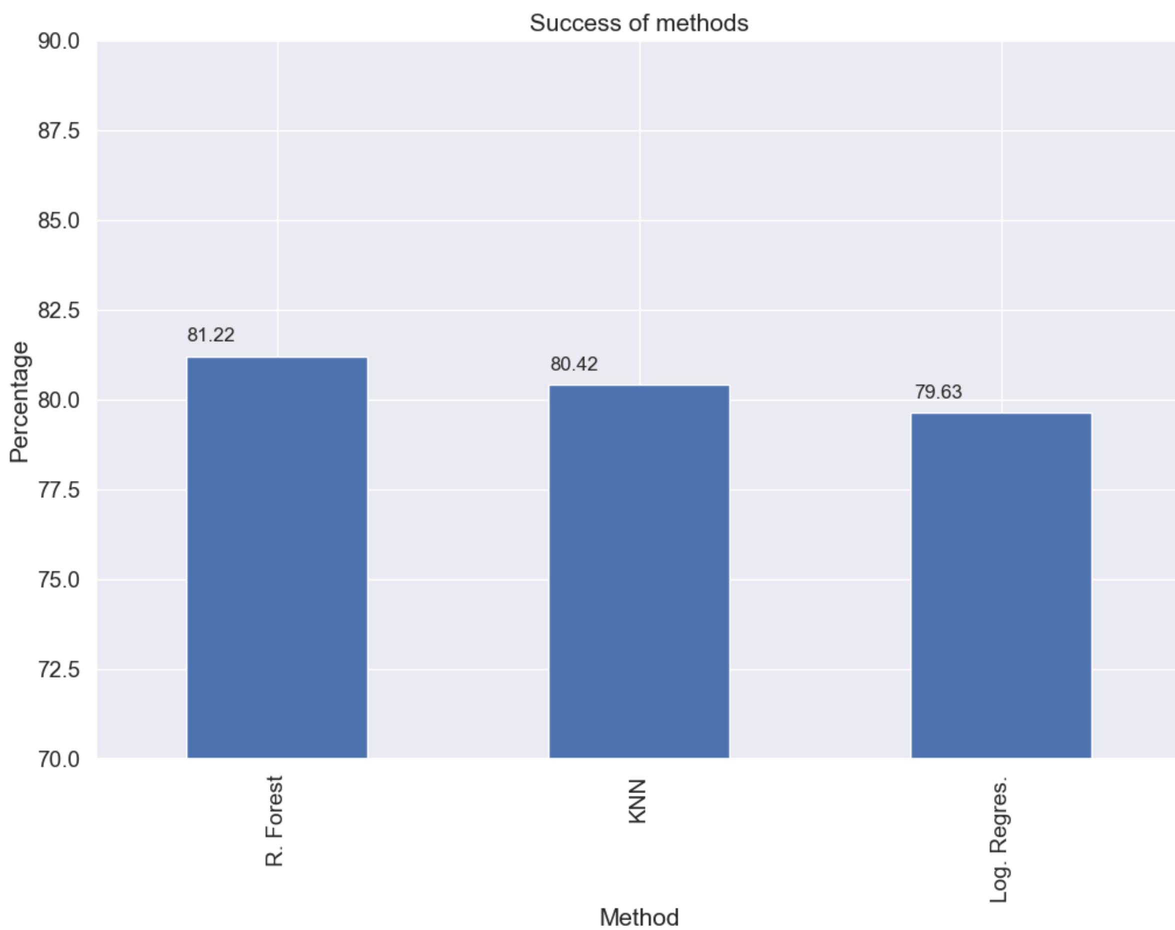

ROC curve for treatment classifier
True Positive Rate (Sensitivity)
False Positive Rate (1 - Specificity)
ROC curve (area = 0.90)
0.0
0.2
0.4
0.6
0.8
1.0

```
[[133  58]
 [ 13 174]]
```

Success of methods plot

In [36]:
```
def plotSuccess():
    s = pd.Series(methodDict)
    s = s.sort_values(ascending=False)
    plt.figure(figsize=(12,8))

    ax = s.plot(kind='bar')
    for p in ax.patches:
        ax.annotate(str(round(p.get_height(),2)), (p.get_x() * 1.005, p.get_height(
    plt.ylim([70.0, 90.0])
    plt.xlabel('Method')
    plt.ylabel('Percentage')
    plt.title('Success of methods')

    plt.show()
```

In [37]:
```
plotSuccess()
```



From the above graph, we can infer that the machine learning algorithm that is best-suited for this problem is Random Forest

Creating predictions on the test set with the Random Forest algorithm

In [38]:
```
clf = RandomForestClassifier(max_depth = None, min_samples_leaf=8, min_samples_spli

clf.fit(X, y)
dfTestPredictions = clf.predict(X_test)

# Writing predictions to csv file
# We don't have any significative field so we save the index
results = pd.DataFrame({'Index': X_test.index, 'Treatment': dfTestPredictions})
# Saving to file
# This file will be visible after publishing in the output section
results.to_csv('results.csv', index=False)
print (results.shape)
results.head(10)
```

```
(378, 2)
```

Out[38]:

	Index	Treatment
0	5	1
1	494	0
2	52	0
3	984	0
4	186	0
5	18	1
6	317	0
7	511	1
8	364	1
9	571	1

From the above graph, we can infer that the machine learning algorithm that is best-suited for this problem is Random Forest.

From the above charts we can predict (with 81.22% accuracy) the individuals who needs treatment for the mental illness.

This project is valuable for treating individuals suffering from mental illness and saving thier lives.

Conclusion

The analysis and model-building process have provided important insights into the predictive capabilities of our machine-learning model.

The model exhibits promising accuracy in identifying individuals in our community who may require mental health treatment.

The features incorporated, ranging from age, gender, and family history data, contribute to a comprehensive understanding of the complex dynamics associated with mental health.