1. Library and data loading

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from scipy import stats
from scipy.stats import randint

# prep
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.datasets import make_classification
from sklearn.preprocessing import Binarizer, LabelEncoder,
MinMaxScaler

# models
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,
ExtraTreesClassifier

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

# Neural Network
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV

#Bagging
from sklearn.ensemble import BaggingClassifier, AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier

#Naive bayes
from sklearn.naive_bayes import GaussianNB

#Stacking
from mlxtend.classifier import StackingClassifier



#reading in CSV's from a file path
train_df = pd.read_csv('input\\survey.csv')
```

```python
#Pandas: whats the data row count?
print("\n" , train_df.shape)

#Pandas: whats the distribution of the data?
print("\n" , train_df.describe())

#Pandas: What types of data do i have?
print("\n" , train_df.info())
```

```
 (1259, 27)

                 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
<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
```

2. Data cleaning

```
#missing data
total = train_df.isnull().sum().sort_values(ascending=False)
percent =
(train_df.isnull().sum()/train_df.isnull().count()).sort_values(ascend
ing=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total',
'Percent'])
missing_data.head(20)
print(missing_data)

                           Total   Percent
comments                    1095  0.869738
state                        515  0.409055
work_interfere               264  0.209690
self_employed                 18  0.014297
seek_help                      0  0.000000
obs_consequence                0  0.000000
mental_vs_physical             0  0.000000
phys_health_interview          0  0.000000
mental_health_interview        0  0.000000
supervisor                     0  0.000000
coworkers                      0  0.000000
phys_health_consequence        0  0.000000
mental_health_consequence      0  0.000000
leave                          0  0.000000
anonymity                      0  0.000000
Timestamp                      0  0.000000
wellness_program               0  0.000000
Age                            0  0.000000
benefits                       0  0.000000
tech_company                   0  0.000000
remote_work                    0  0.000000
no_employees                   0  0.000000
treatment                      0  0.000000
family_history                 0  0.000000
Country                        0  0.000000
```

```
Gender                              0  0.000000
care_options                        0  0.000000

#dealing with missing data
#Removing the variables "Timestamp",“comments”, “state” just to make
our it easier to analyze.
train_df = train_df.drop(['comments'], axis= 1)
train_df = train_df.drop(['state'], axis= 1)
train_df = train_df.drop(['Timestamp'], axis= 1)

train_df.isnull().sum().max()
train_df.head(5)

   Age  Gender         Country self_employed family_history treatment
\
0   37  Female   United States           NaN             No       Yes

1   44       M   United States           NaN             No        No

2   32    Male          Canada           NaN             No        No

3   31    Male  United Kingdom           NaN            Yes       Yes

4   31    Male   United States           NaN             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]
```

Cleaning NaN

```
# Assign default values for each data type
defaultInt = 0
defaultString = 'NaN'
defaultFloat = 0.0

# Create lists by data tpe
intFeatures = ['Age']
stringFeatures = ['Gender', 'Country', 'self_employed', 
'family_history', 'treatment', 'work_interfere',
                 'no_employees', 'remote_work', 'tech_company', 
'anonymity', 'leave', 'mental_health_consequence',
                 'phys_health_consequence', 'coworkers', 'supervisor', 
'mental_health_interview', 'phys_health_interview',
                 'mental_vs_physical', 'obs_consequence', 'benefits', 
'care_options', 'wellness_program',
                 'seek_help']
floatFeatures = []

# Clean the NaN's
for feature in train_df:
    if feature in intFeatures:
        train_df[feature] = train_df[feature].fillna(defaultInt)
    elif feature in stringFeatures:
```

```
        train_df[feature] = train_df[feature].fillna(defaultString)
    elif feature in floatFeatures:
        train_df[feature] = train_df[feature].fillna(defaultFloat)
    else:
        print('Error: Feature %s not recognized.' % feature)
train_df.head(5)
```

```
   Age  Gender         Country self_employed family_history treatment  \
0   37  Female   United States           NaN             No       Yes

1   44       M   United States           NaN             No        No

2   32    Male          Canada           NaN             No        No

3   31    Male  United Kingdom           NaN            Yes       Yes

4   31    Male   United States           NaN             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]

# clean 'Gender'
# lower case all columm's elements
gender = train_df['Gender'].str.lower()


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

# Made gender groups
male_str = ["male", "m", "male-ish", "maile", "mal", "male (cis)",
"make", "male ", "man","msle", "mail", "malr","cis man", "Cis Male",
"cis male"]
trans_str = ["trans-female", "something kinda male?",
"queer/she/they", "non-binary","nah", "all", "enby", "fluid",
"genderqueer", "androgyne", "agender", "male leaning androgynous",
"guy (-ish) ^_^", "trans woman", "neuter", "female (trans)", "queer",
"ostensibly male, unsure what that really means"]
female_str = ["cis female", "f", "female", "woman",  "femake", "female
","cis-female/femme", "female (cis)", "femail"]

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

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

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

    if str.lower(col.Gender) in trans_str:
        train_df['Gender'].replace(to_replace=col.Gender,
```

```
value='trans', inplace=True)

stk_list = ['A little about you', 'p']
train_df = train_df[~train_df['Gender'].isin(stk_list)]

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

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

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

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

# Ranges of Age
train_df['age_range'] = pd.cut(train_df['Age'], [0,20,30,65,100],
labels=["0-20", "21-30", "31-65", "66-100"], include_lowest=True)


#There are only 0.014% of self employed so let's change NaN to NOT
self_employed
#Replace "NaN" string from defaultString
train_df['self_employed'] =
train_df['self_employed'].replace([defaultString], 'No')
print(train_df['self_employed'].unique())
```

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

```
#There are only 0.20% of self work_interfere so let's change NaN to
"Don't know
#Replace "NaN" string from defaultString

train_df['work_interfere'] =
train_df['work_interfere'].replace([defaultString], 'Don\'t know' )
print(train_df['work_interfere'].unique())
```

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

3. Encoding data

```
#Encoding data
labelDict = {}
for feature in train_df:
```

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

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

#Get rid of 'Country'
train_df = train_df.drop(['Country'], axis= 1)
train_df.head()
```

```
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, 57, 58, 60, 61, 62, 65, 72]
label_Gender ['female', 'male', 'trans']
label_Country ['Australia', 'Austria', 'Belgium', 'Bosnia and
Herzegovina', 'Brazil', 'Bulgaria', 'Canada', 'China', 'Colombia',
'Costa Rica', 'Croatia', 'Czech Republic', 'Denmark', 'Finland',
'France', 'Georgia', 'Germany', 'Greece', 'Hungary', 'India',
'Ireland', 'Israel', 'Italy', 'Japan', 'Latvia', 'Mexico', 'Moldova',
'Netherlands', 'New Zealand', 'Nigeria', 'Norway', 'Philippines',
'Poland', 'Portugal', 'Romania', 'Russia', 'Singapore', 'Slovenia',
'South Africa', 'Spain', 'Sweden', 'Switzerland', 'Thailand', 'United
Kingdom', 'United States', 'Uruguay', 'Zimbabwe']
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 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']
label_age_range ['0-20', '21-30', '31-65', '66-100']
```

```
   Age  Gender  self_employed  family_history  treatment
work_interfere  \
0   19       0              0               0          1
2
1   26       1              0               0          0
3
2   14       1              0               0          0
3
3   13       1              0               1          1
2
4   13       1              0               0          0
1

   no_employees  remote_work  tech_company  benefits  ...  leave  \
0             4            0             1         2  ...      2
1             5            0             0         0  ...      0
2             4            0             1         1  ...      1
3             2            0             1         1  ...      1
4             1            1             1         2  ...      0

   mental_health_consequence  phys_health_consequence  coworkers
supervisor  \
0                          1                        1          1
2
1                          0                        1          0
0
2                          1                        1          2
2
3                          2                        2          1
0
4                          1                        1          1
2

   mental_health_interview  phys_health_interview  mental_vs_physical
\
0                        1                      0                   2

1                        1                      1                   0

2                        2                      2                   1

3                        0                      0                   1
```

```
4                          2                       2                   0


   obs_consequence  age_range
0                0          2
1                0          2
2                0          2
3                1          2
4                0          2

[5 rows x 24 columns]
```

Testing there aren't any missing data

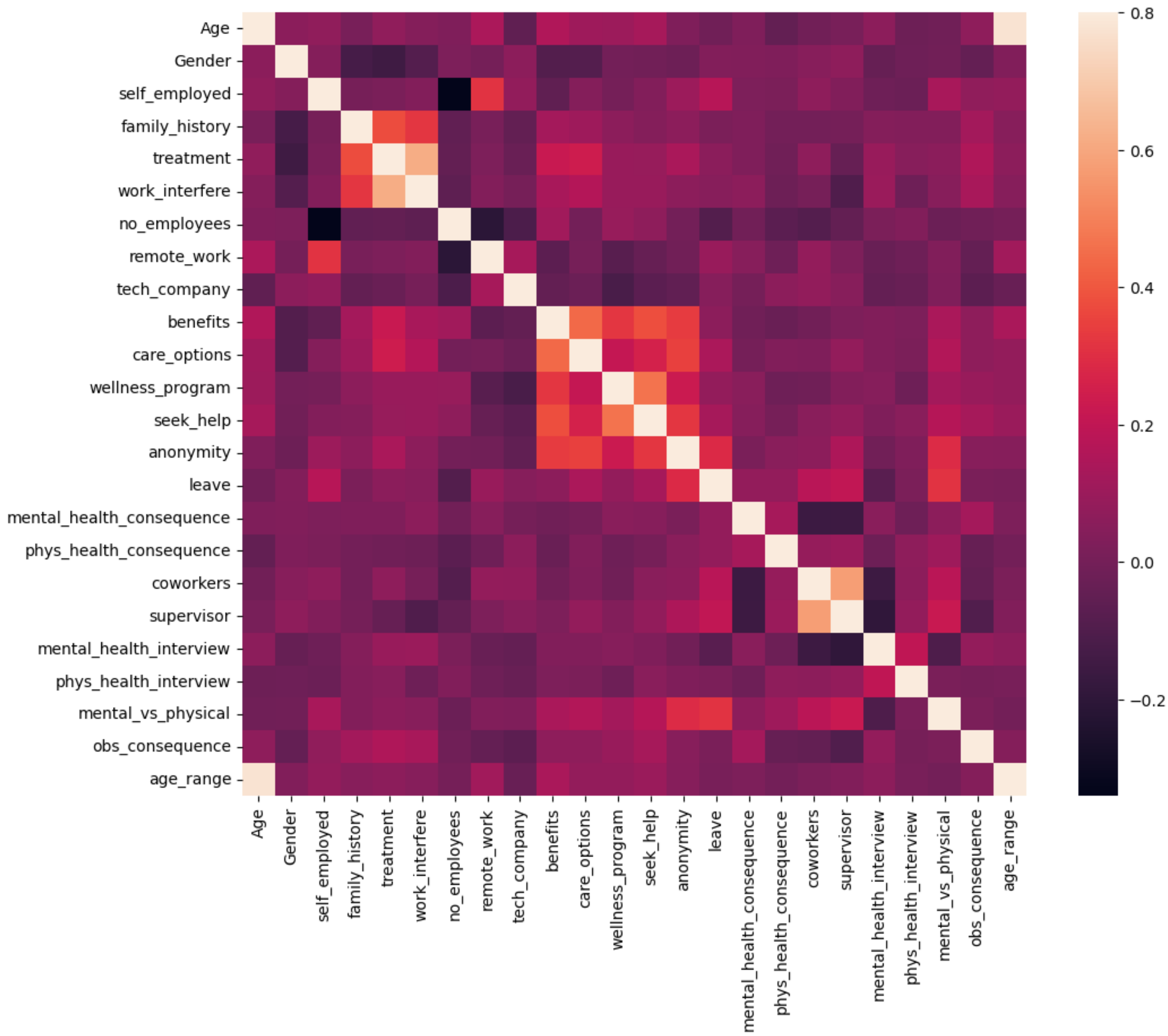
```
#missing data
total = train_df.isnull().sum().sort_values(ascending=False)
percent =
(train_df.isnull().sum()/train_df.isnull().count()).sort_values(ascend
ing=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total',
'Percent'])
missing_data.head(20)
print(missing_data)
```

```
                           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
```

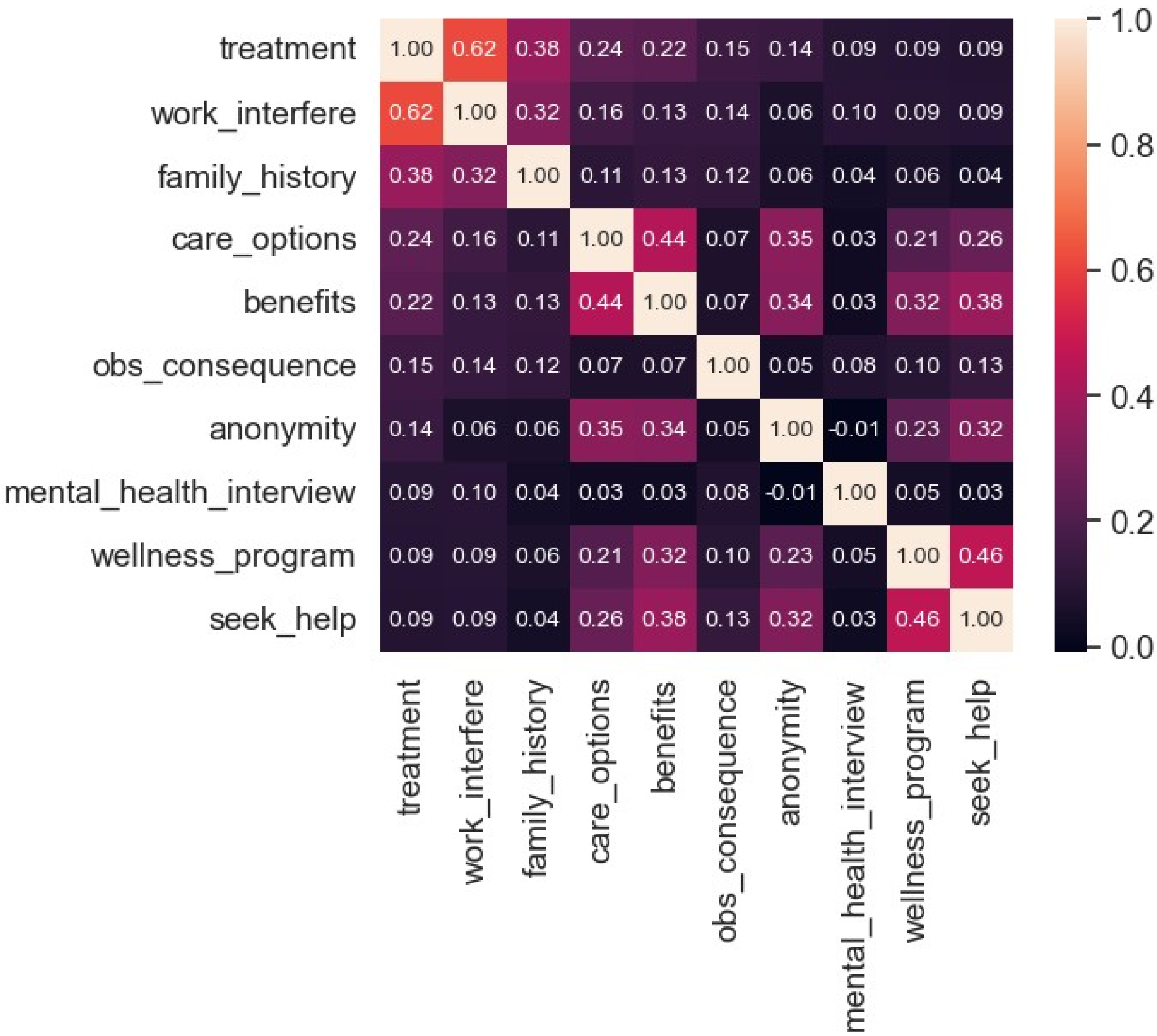
Features Scaling We're going to scale age, because is extremely different from the othere ones.

4. Covariance Matrix. Variability comparison between categories of variables

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

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



Age
Gender
self_employed
family_history
treatment
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
age_range
0.8
0.6
0.4
0.2
0.0
−0.2



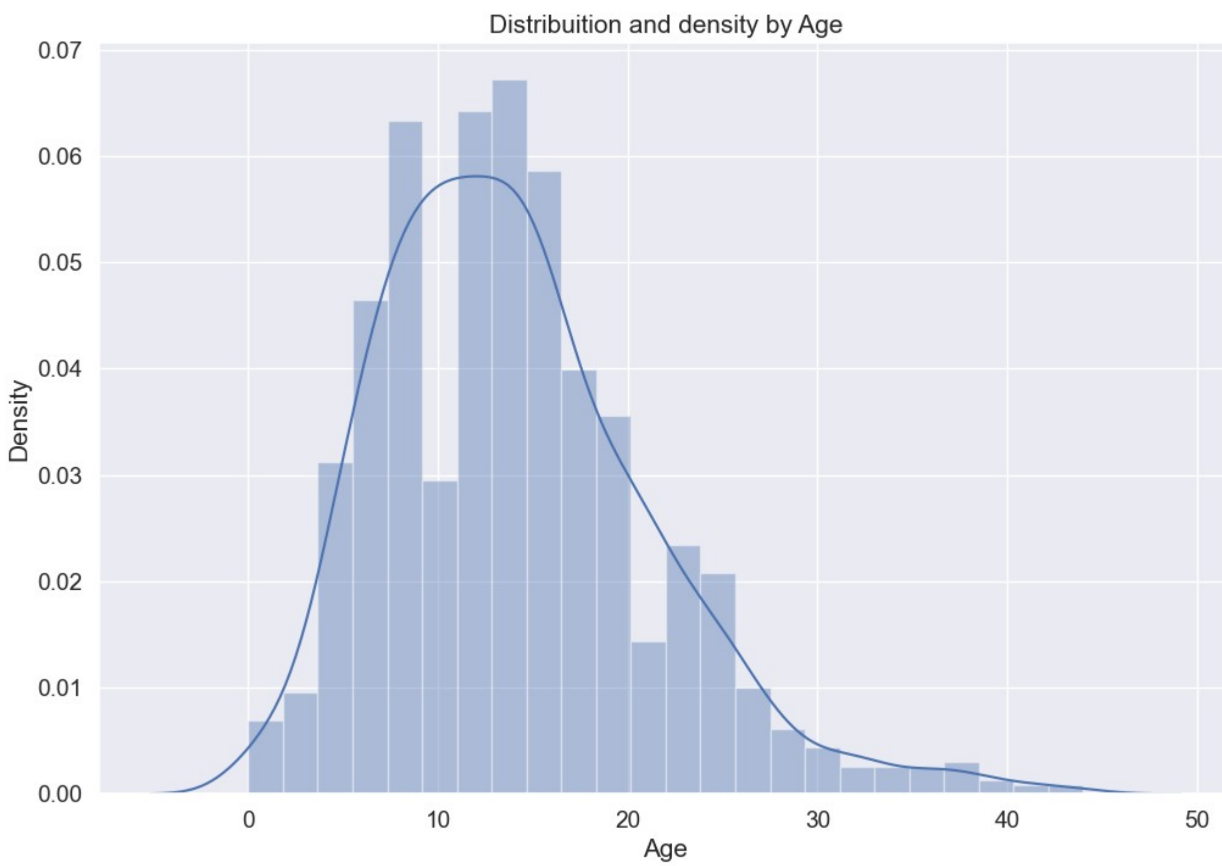
5. Some charts to see data relationship

Distribiution and density by Age

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

```
C:\Users\athar\AppData\Local\Temp\ipykernel_10060\1394260443.py:3:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
```

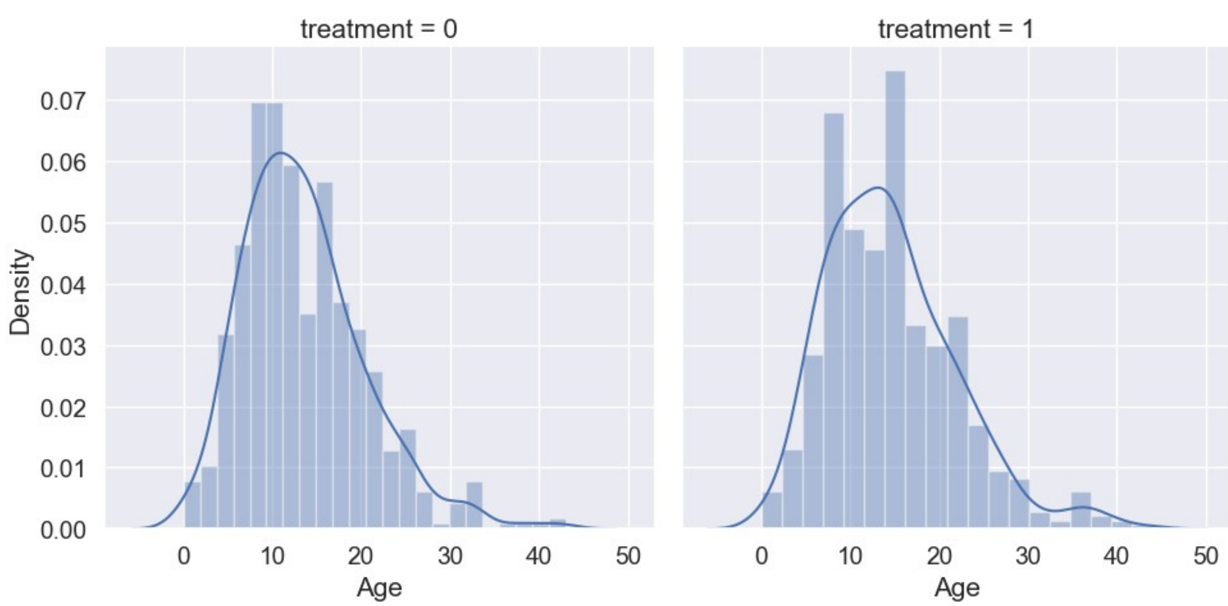
```
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(train_df["Age"], bins=24)

Text(0.5, 0, 'Age')
```



Separate by treatment

```
# Separate by treatment or not

g = sns.FacetGrid(train_df, col='treatment', height=5)
g = g.map(sns.distplot, "Age")

b:\Anaconda3\envs\_AIMLDataScience_env\Lib\site-packages\seaborn\
axisgrid.py:848: UserWarning:
```

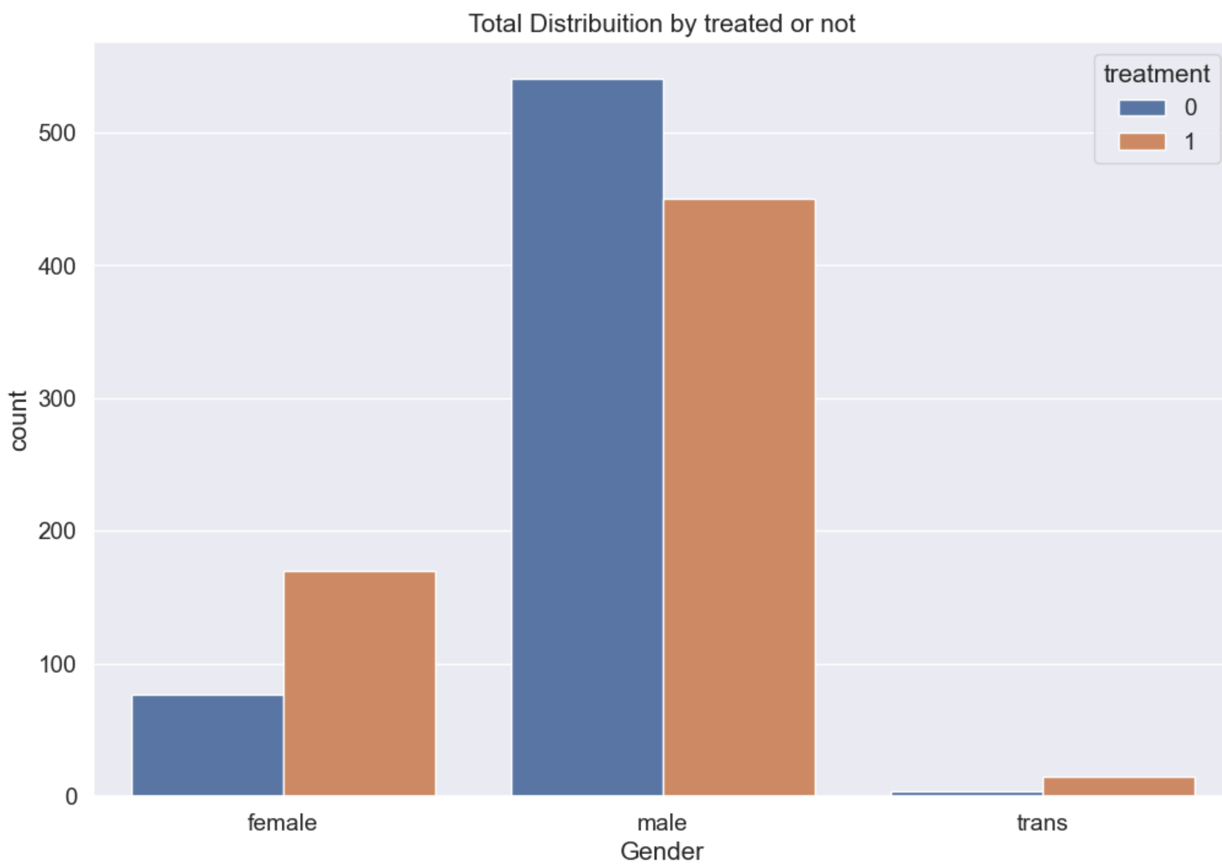
```
`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  func(*plot_args, **plot_kwargs)
b:\Anaconda3\envs\_AIMLDataScience_env\Lib\site-packages\seaborn\
axisgrid.py:848: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  func(*plot_args, **plot_kwargs)
```



How many people has been treated?

```
# how many people has been treated
plt.figure(figsize=(12,8))
labels = labelDict['label_Gender']
g = sns.countplot(x="Gender", hue="treatment", data=train_df)
g.set_xticklabels(labels)

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

```
Text(0.5, 1.0, 'Total Distribuition by treated or not')
```

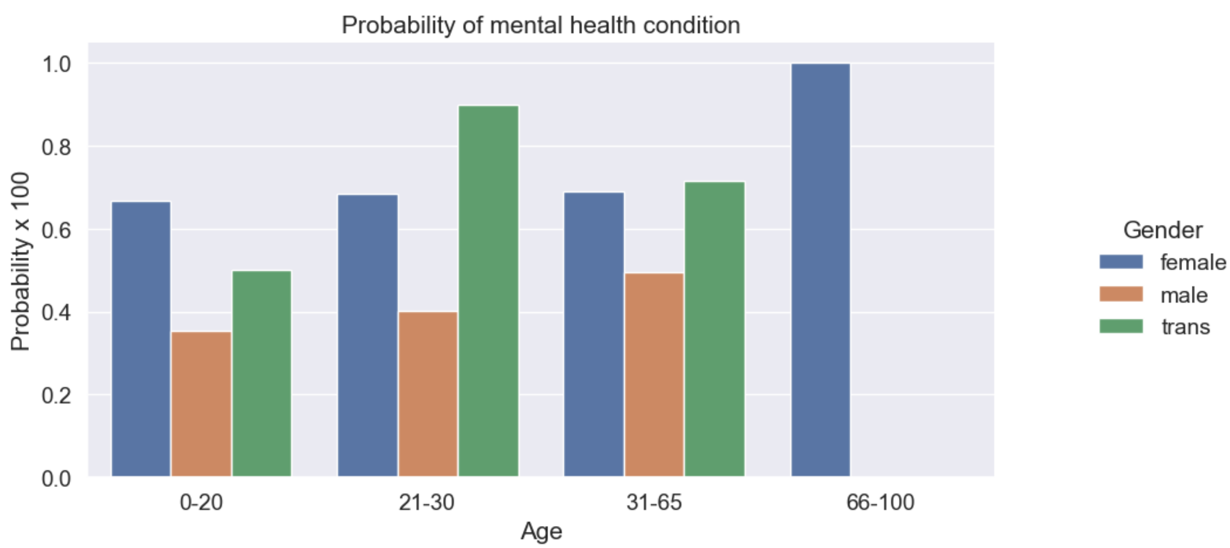


Draw a nested barplot to show probabilities for class and sex

```
o = labelDict['label_age_range']

g = sns.catplot(x="age_range", y="treatment", hue="Gender",
data=train_df, kind="bar",  ci=None, height=5, aspect=2, legend_out =
True)
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()
```

```
C:\Users\athar\AppData\Local\Temp\ipykernel_10060\827670349.py:3:
FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same
effect.

  g = sns.catplot(x="age_range", y="treatment", hue="Gender",
data=train_df, kind="bar",  ci=None, height=5, aspect=2, legend_out =
True)
```

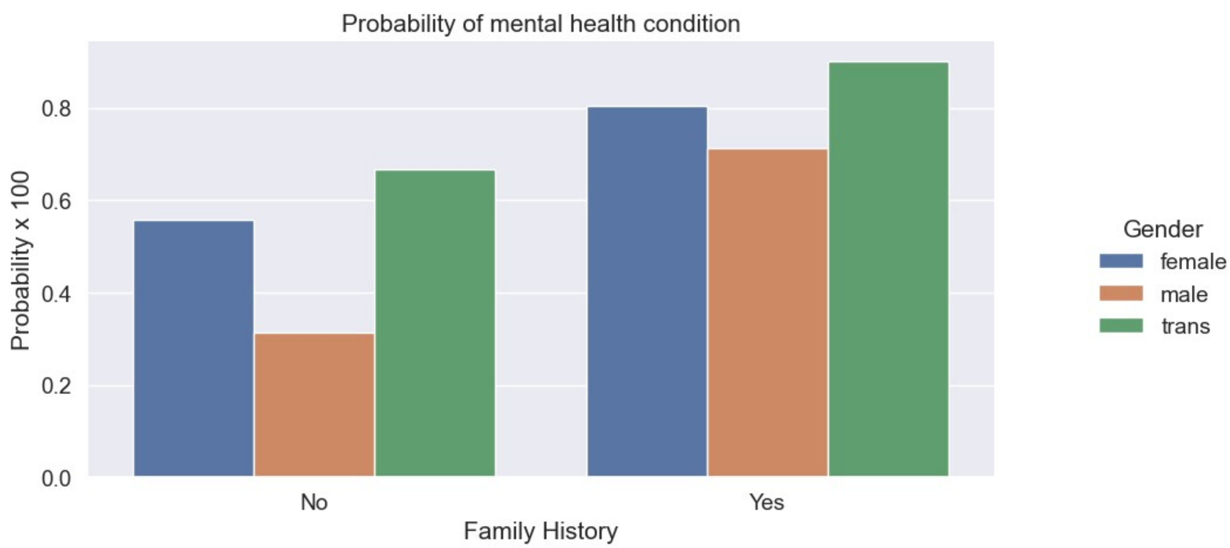


Barplot to show probabilities for family history

```
o = labelDict['label_family_history']
g = sns.catplot(x="family_history", y="treatment", hue="Gender",
data=train_df, kind="bar", ci=None, height=5, aspect=2, legend_out =
True)
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()
```

```
C:\Users\athar\AppData\Local\Temp\ipykernel_10060\2317941586.py:2:
FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same
effect.

  g = sns.catplot(x="family_history", y="treatment", hue="Gender",
data=train_df, kind="bar", ci=None, height=5, aspect=2, legend_out =
True)
```

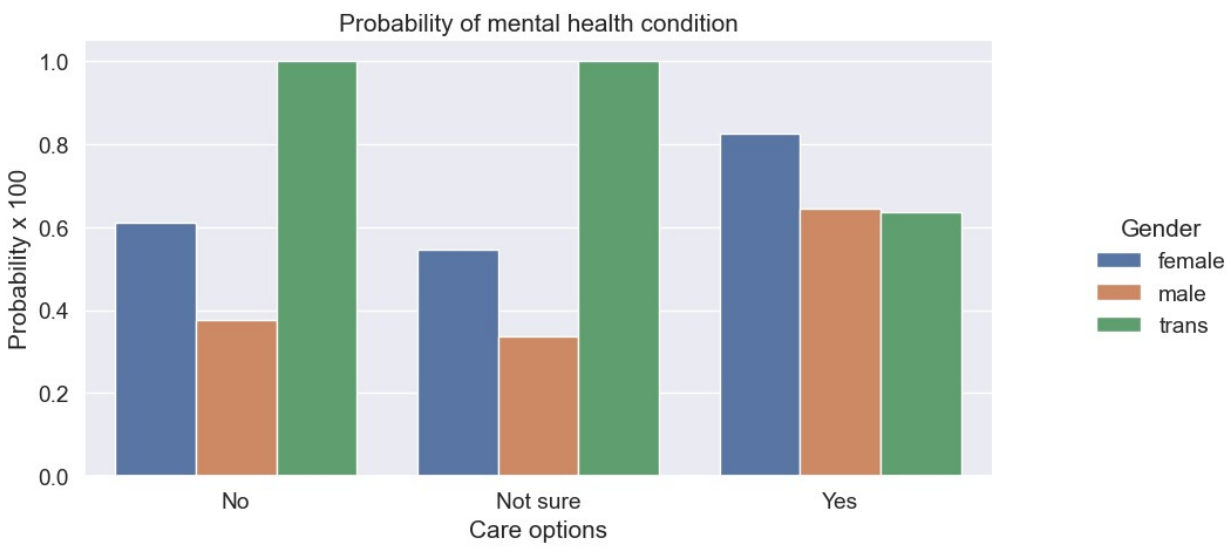


Barplot to show probabilities for care options

```
o = labelDict['label_care_options']
g = sns.catplot(x="care_options", y="treatment", hue="Gender",
data=train_df, kind="bar", ci=None, height=5, aspect=2, legend_out =
True)
g.set_xticklabels(o)
plt.title('Probability of mental health condition')
plt.ylabel('Probability x 100')
plt.xlabel('Care options')

# 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()
```

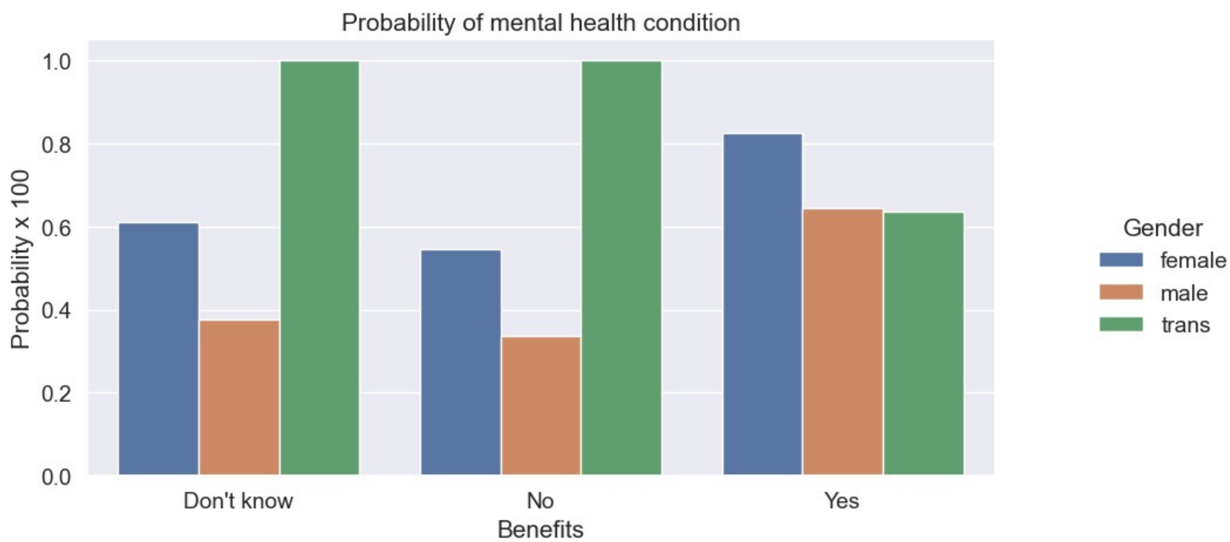
```
C:\Users\athar\AppData\Local\Temp\ipykernel_10060\1780520276.py:2:
FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same
effect.

  g = sns.catplot(x="care_options", y="treatment", hue="Gender",
data=train_df, kind="bar", ci=None, height=5, aspect=2, legend_out =
True)
```



Barplot to show probabilities for benefits

```
o = labelDict['label_benefits']
g = sns.catplot(x="care_options", y="treatment", hue="Gender",
data=train_df, kind="bar", ci=None, height=5, aspect=2, legend_out =
True)
g.set_xticklabels(o)
plt.title('Probability of mental health condition')
plt.ylabel('Probability x 100')
plt.xlabel('Benefits')

# 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()
```

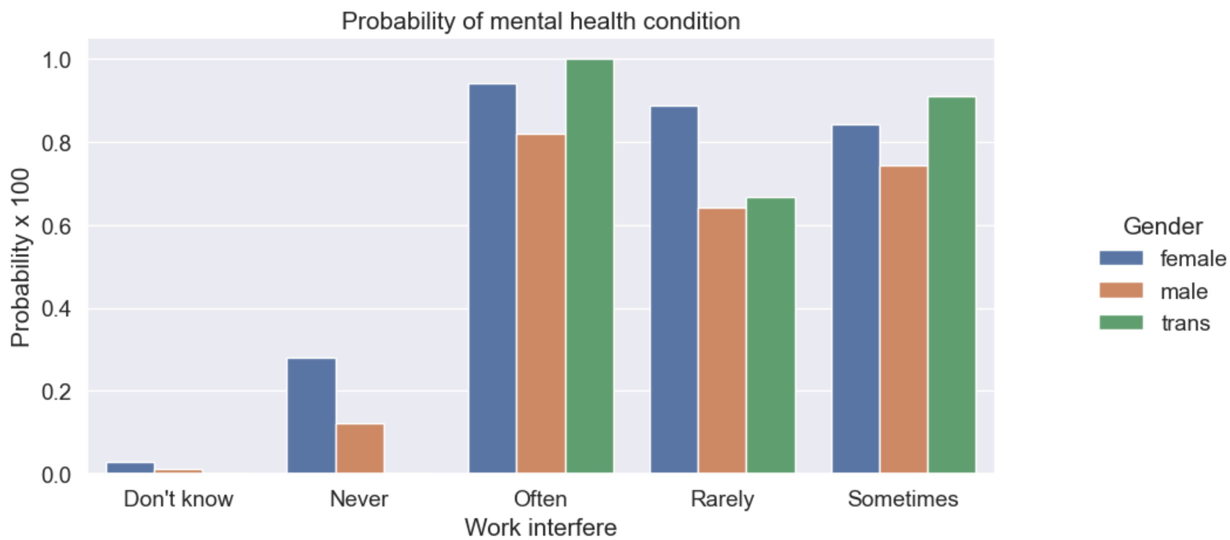
```
C:\Users\athar\AppData\Local\Temp\ipykernel_10060\1084406037.py:2:
FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same
effect.

  g = sns.catplot(x="care_options", y="treatment", hue="Gender",
data=train_df, kind="bar", ci=None, height=5, aspect=2, legend_out =
True)
```



Barplot to show probabilities for work interfere

```
o = labelDict['label_work_interfere']
g = sns.catplot(x="work_interfere", y="treatment", hue="Gender",
data=train_df, kind="bar", ci=None, height=5, aspect=2, legend_out =
True)
g.set_xticklabels(o)
plt.title('Probability of mental health condition')
plt.ylabel('Probability x 100')
plt.xlabel('Work interfere')

# 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()

C:\Users\athar\AppData\Local\Temp\ipykernel_10060\2520480164.py:2:
FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same
```

```
effect.

  g = sns.catplot(x="work_interfere", y="treatment", hue="Gender",
data=train_df, kind="bar", ci=None, height=5, aspect=2, legend_out =
True)
```



6. Scaling and fitting

Features Scaling We're going to scale age, because is extremely different from the othere ones.

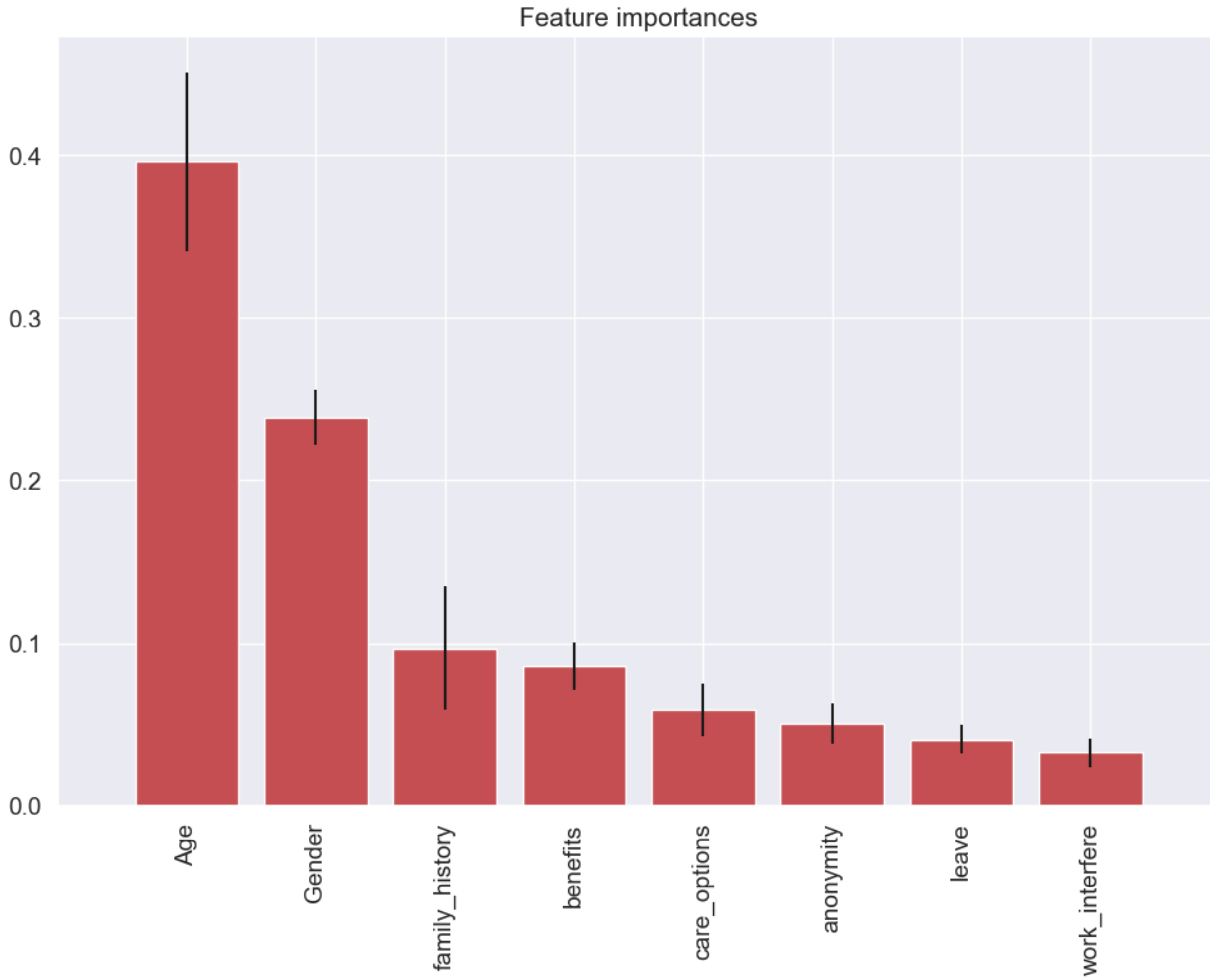
```
# Scaling Age
scaler = MinMaxScaler()
train_df['Age'] = scaler.fit_transform(train_df[['Age']])
train_df.head()

        Age  Gender  self_employed  family_history  treatment  
work_interfere  \
0  0.431818       0              0               0          1  
2
1  0.590909       1              0               0          0  
3
2  0.318182       1              0               0          0  
3
3  0.295455       1              0               1          1  
2
4  0.295455       1              0               0          0  
1

   no_employees  remote_work  tech_company  benefits  ...  leave  \
0             4            0             1         2  ...      2
```

```
1               5             0              0          0  ...       0
2               4             0              1          1  ...       1
3               2             0              1          1  ...       1
4               1             1              1          2  ...       0

   mental_health_consequence  phys_health_consequence  coworkers
supervisor  \
0                          1                        1          1
2
1                          0                        1          0
0
2                          1                        1          2
2
3                          2                        2          1
0
4                          1                        1          1
2

   mental_health_interview  phys_health_interview  mental_vs_physical
\
0                        1                      0                   2

1                        1                      1                   0

2                        2                      2                   1

3                        0                      0                   1

4                        2                      2                   0


   obs_consequence  age_range
0                0          2
1                0          2
2                0          2
3                1          2
4                0          2

[5 rows x 24 columns]
```

Spliltting the dataset

```
# define X and y
feature_cols = ['Age', 'Gender', 'family_history', 'benefits',
'care_options', 'anonymity', 'leave', 'work_interfere']
X = train_df[feature_cols]
y = train_df.treatment

# split X and y into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
```

```
test_size=0.30, random_state=0)

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

# Build a forest and compute the feature importances
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")
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()
```



7. Tuning

Evaluating a Classification Model

This function will evaluate:

- **Classification accuracy:** Percentage of correct predictions
- **Null accuracy:** Accuracy that could be achieved by always predicting the most frequent class
- **Percentage of ones**
- **Percentage of zeros**
- **Confusion matrix:** Table that describes the performance of a classification model
- **Confusion matrix elements:**

  - True Positives (TP): Correctly predicted they have diabetes
  - True Negatives (TN): Correctly predicted they don't have diabetes
  - False Positives (FP): Incorrectly predicted they have diabetes (Type I error)
  - False Negatives (FN): Incorrectly predicted they don't have diabetes (Type II error)
- **False Positive Rate**
- **Precision of Positive value**
- **AUC (Area Under the Curve):** 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)

And some other values for the tuning process.

```
def evalClassModel(model, y_test, y_pred_class, plot=False):
    #Classification accuracy: 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 frequent class
    # examine the class distribution of the testing set (using a 
Pandas Series method)
    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 response 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 you the underlying distribution of response 
values
    #And, it does not tell you what "types" of errors your classifier 
is making

    #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()

    #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 prediction incorrect?
    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_test, y_pred_class))


    # IMPORTANT: first argument is true values, second argument is
predicted probabilities
    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_auc').mean())

    ##########################################
    #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_proba(X_test)[0:10])

    # 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)
    threshold = 0.3
    binarizer = Binarizer(threshold=threshold)
    y_pred_class = binarizer.transform(y_pred_prob)

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



    #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)



    # IMPORTANT: 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 generating an error
    # 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)' % 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()

    # define a function that accepts a threshold and prints
sensitivity and specificity
    def evaluate_threshold(threshold):
        #Sensitivity: When the actual value is positive, how often is
the prediction correct?
        #Specificity: When the actual value is negative, how often is
the prediction correct?print('Sensitivity for ' + str(threshold) +
' :', tpr[thresholds > threshold][-1])
        print('Specificity for ' + str(threshold) + ' :', 1 -
fpr[thresholds > threshold][-1])

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



    return accuracy
```

Tuning with cross validation score

```
##########################################
# Tuning with cross validation score
##########################################
def tuningCV(knn):

```

```
    # search for an optimal value of K for KNN
    k_range = list(range(1, 31))
    k_scores = []
    for k in k_range:
        knn = KNeighborsClassifier(n_neighbors=k)
        scores = cross_val_score(knn, X, y, cv=10, scoring='accuracy')
        k_scores.append(scores.mean())
    print(k_scores)
    # plot the value of K for KNN (x-axis) versus the cross-validated
accuracy (y-axis)
    plt.plot(k_range, k_scores)
    plt.xlabel('Value of K for KNN')
    plt.ylabel('Cross-Validated Accuracy')
    plt.show()
```

Tuning with GridSearchCV

```
def tuningGridSerach(knn):
    #More efficient parameter tuning using GridSearchCV
    # define the parameter values that should be searched
    k_range = list(range(1, 31))
    print(k_range)

    # create a parameter grid: map the parameter names to the values
that should be searched
    param_grid = dict(n_neighbors=k_range)
    print(param_grid)

    # instantiate the grid
    grid = GridSearchCV(knn, param_grid, cv=10, scoring='accuracy')

    # fit the grid with data
    grid.fit(X, y)

    # view the complete results (list of named tuples)
    grid.grid_scores_

    # examine the first tuple
    print(grid.grid_scores_[0].parameters)
    print(grid.grid_scores_[0].cv_validation_scores)
    print(grid.grid_scores_[0].mean_validation_score)

    # create a list of the mean scores only
    grid_mean_scores = [result.mean_validation_score for result in
grid.grid_scores_]
    print(grid_mean_scores)

    # plot the results
    plt.plot(k_range, grid_mean_scores)
```

```
    plt.xlabel('Value of K for KNN')
    plt.ylabel('Cross-Validated Accuracy')
    plt.show()

    # examine the best model
    print('GridSearch best score', grid.best_score_)
    print('GridSearch best params', grid.best_params_)
    print('GridSearch best estimator', grid.best_estimator_)
```

Tuning with RandomizedSearchCV

```
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.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_iter=10)
        rand.fit(X, y)
        best_scores.append(round(rand.best_score_, 3))
    print(best_scores)
```

Tuning with searching multiple parameters simultaneously

```
def tuningMultParam(knn):

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

    # create a parameter grid: map the parameter names to the values
that should be searched
    param_grid = dict(n_neighbors=k_range, weights=weight_options)
    print(param_grid)

    # instantiate and fit the grid
    grid = GridSearchCV(knn, param_grid, cv=10, scoring='accuracy')
```

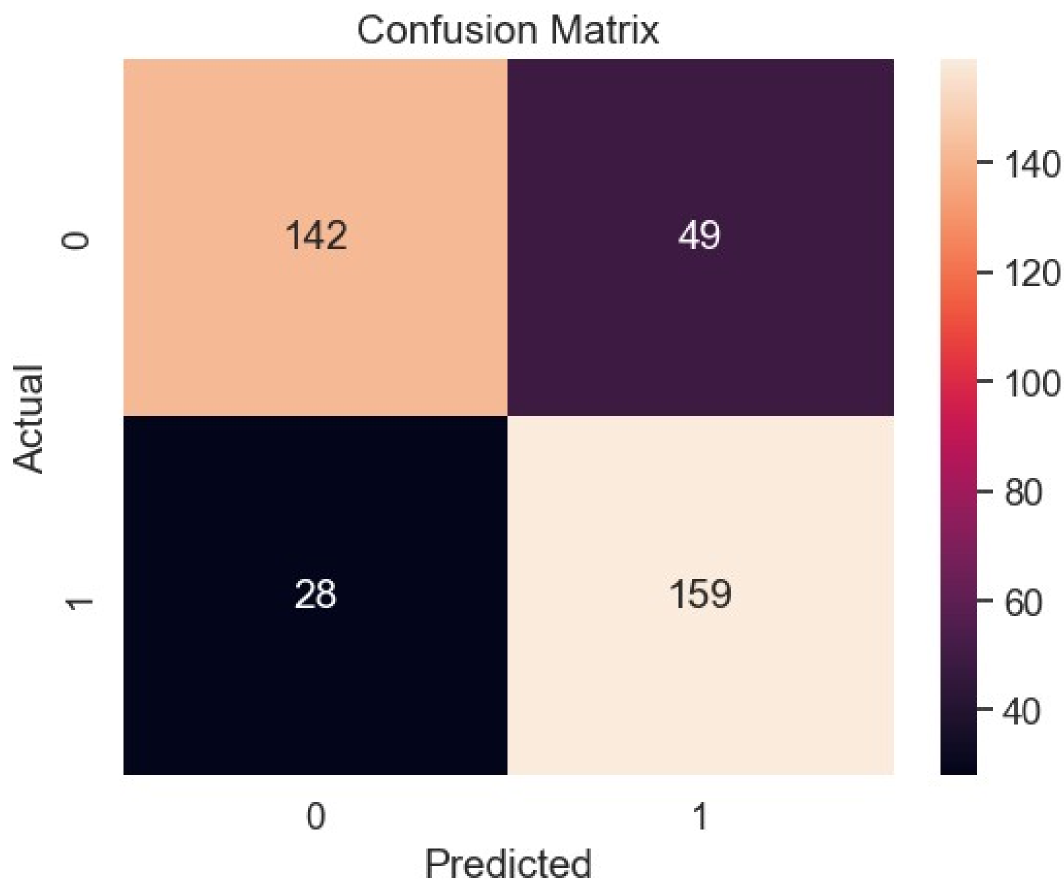
```
    grid.fit(X, y)

    # view the complete results
    print(grid.grid_scores_)

    # examine the best model
    print('Multiparam. Best Score: ', grid.best_score_)
    print('Multiparam. Best Params: ', grid.best_params_)
```

8. Evaluating models

Logistic Regression

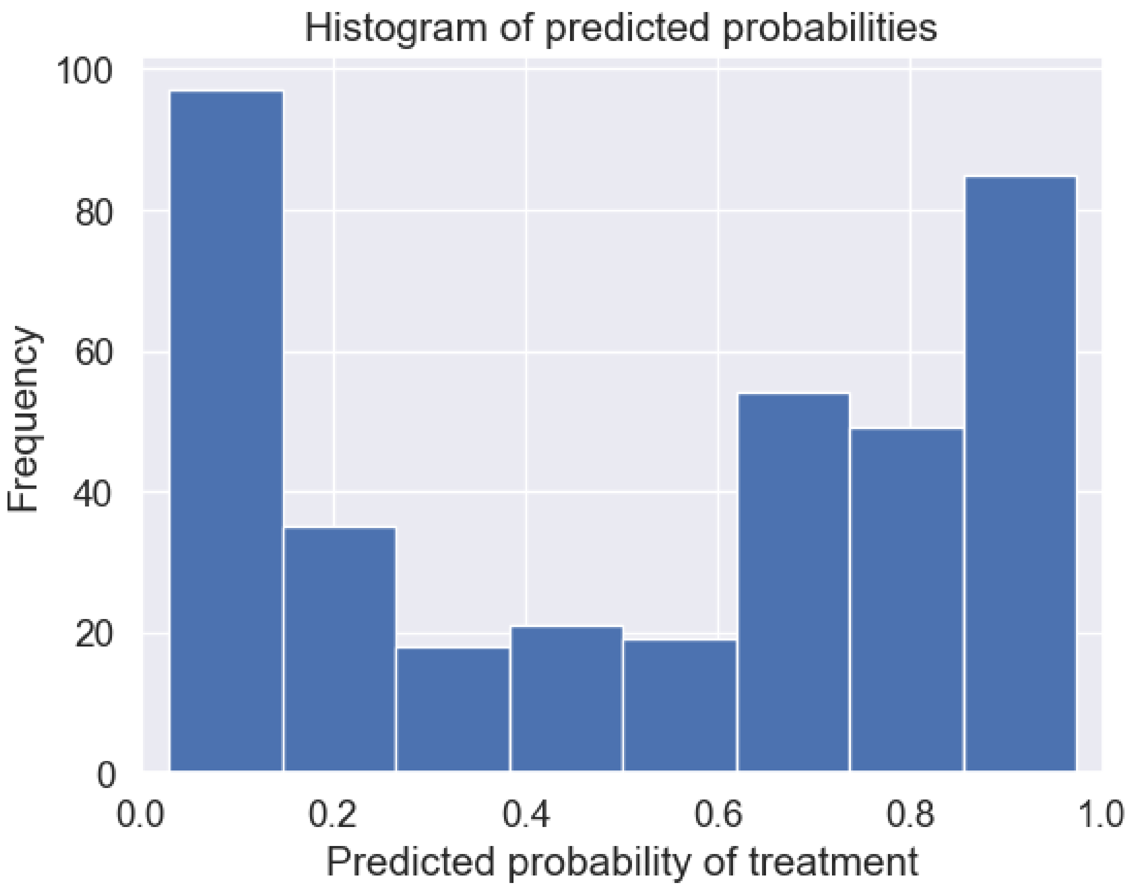
```
def logisticRegression():
    # train a logistic regression model on the training set
    logreg = LogisticRegression()
    logreg.fit(X_train, y_train)

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

logisticRegression()
```

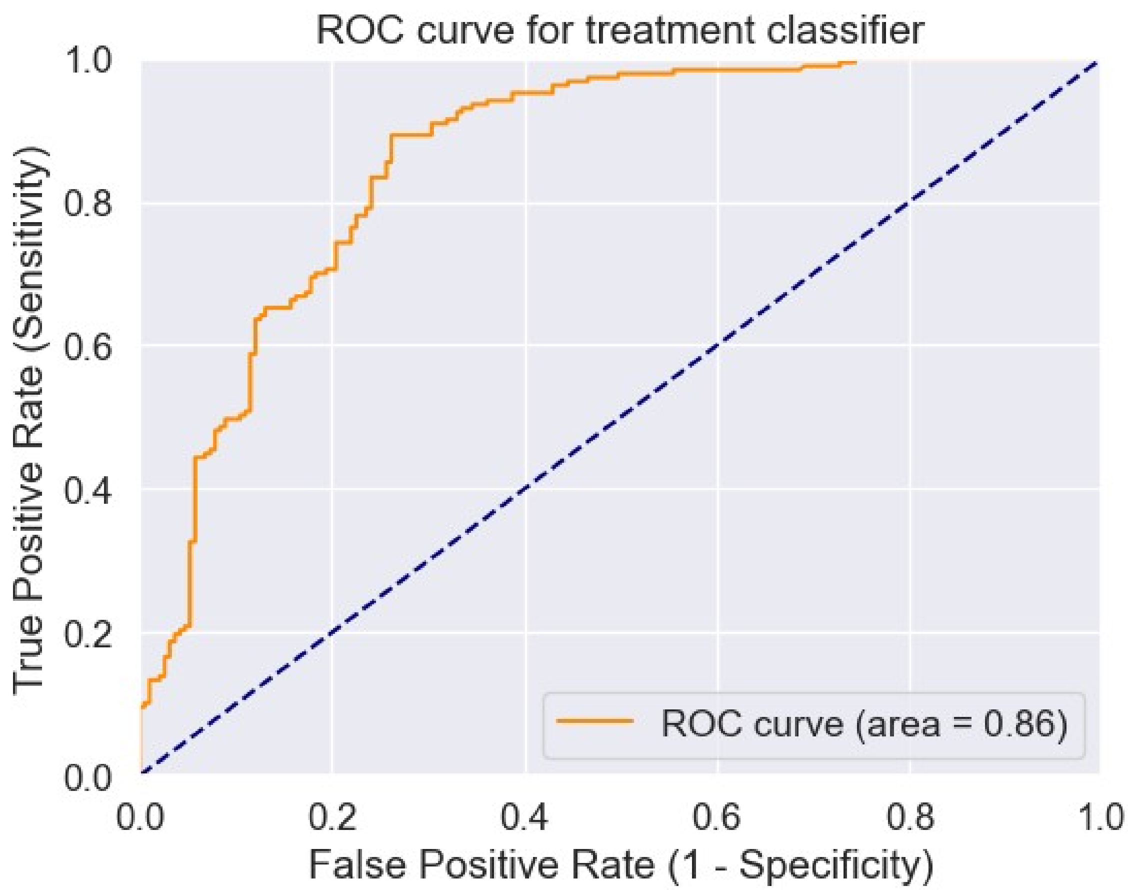
```
########### Logistic Regression ###############
Accuracy: 0.7962962962962963
Null accuracy:
 treatment
0    191
1    187
Name: count, dtype: int64
Percentage of ones: 0.4947089947089947
Percentage of zeros: 0.5052910052910053
True: [0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1 0 0]
Pred: [1 0 0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






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

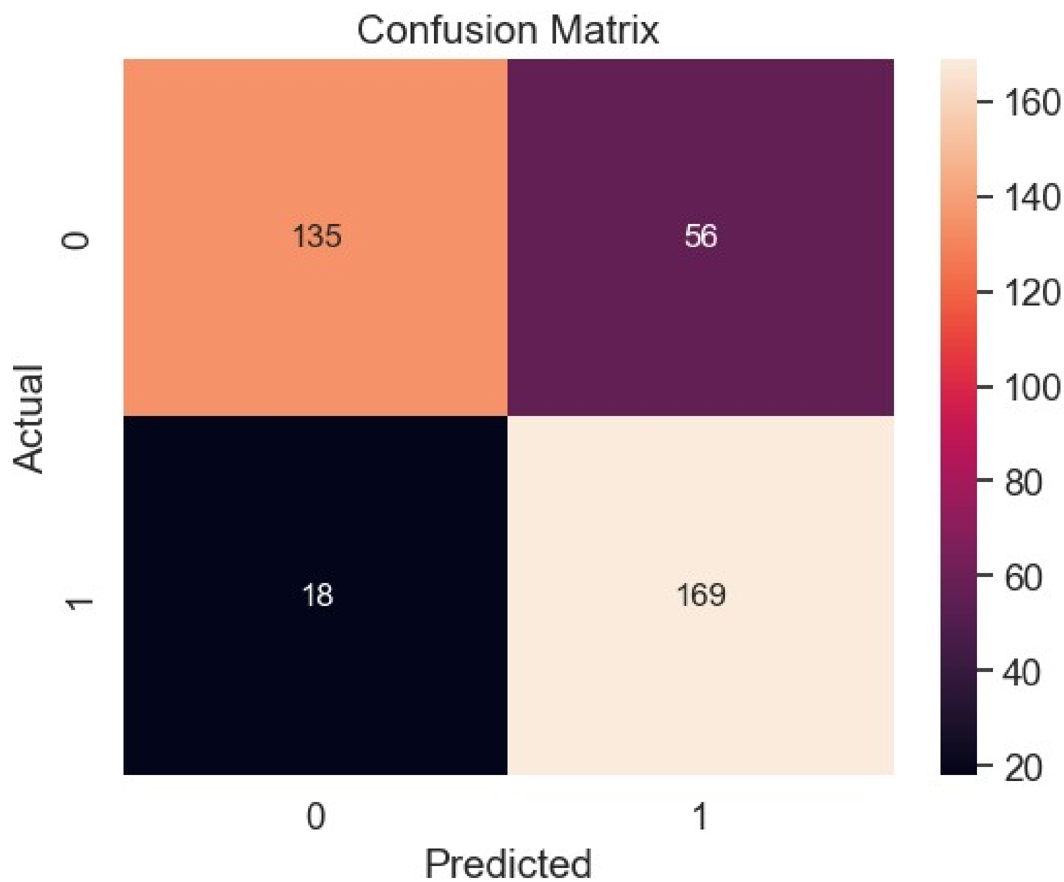
KNeighbors Classifier

```
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)
```

```python
# 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
```

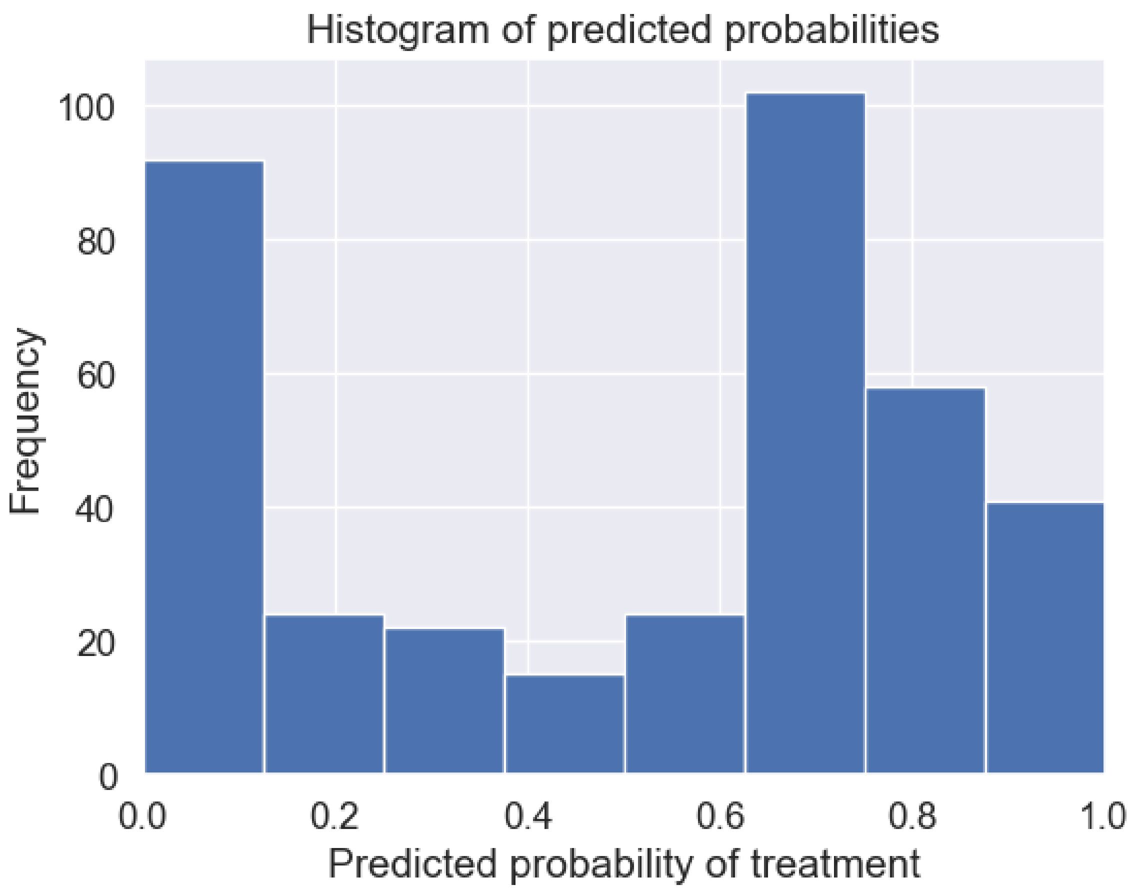
KNEIGHBORSCLASSIFIER

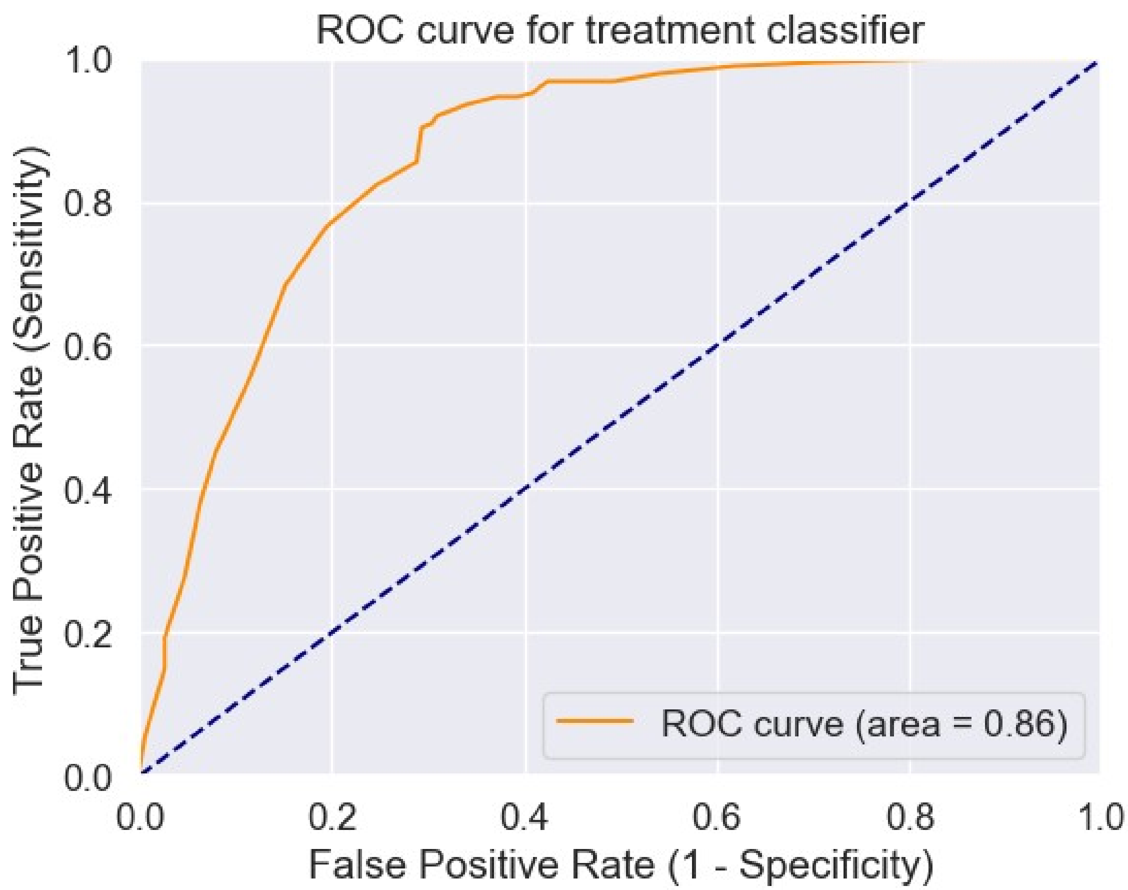
```
Knn()

Rand. Best Score:  0.8217650793650794
Rand. Best Params:  {'weights': 'uniform', 'n_neighbors': 27}
[0.814, 0.819, 0.822, 0.811, 0.814, 0.814, 0.819, 0.819, 0.819, 0.822,
0.822, 0.819, 0.822, 0.817, 0.822, 0.819, 0.817, 0.816, 0.816, 0.813]
########### KNeighborsClassifier ###############
Accuracy: 0.8042328042328042
Null accuracy:
 treatment
0    191
1    187
Name: count, 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]
```




```
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]]
```





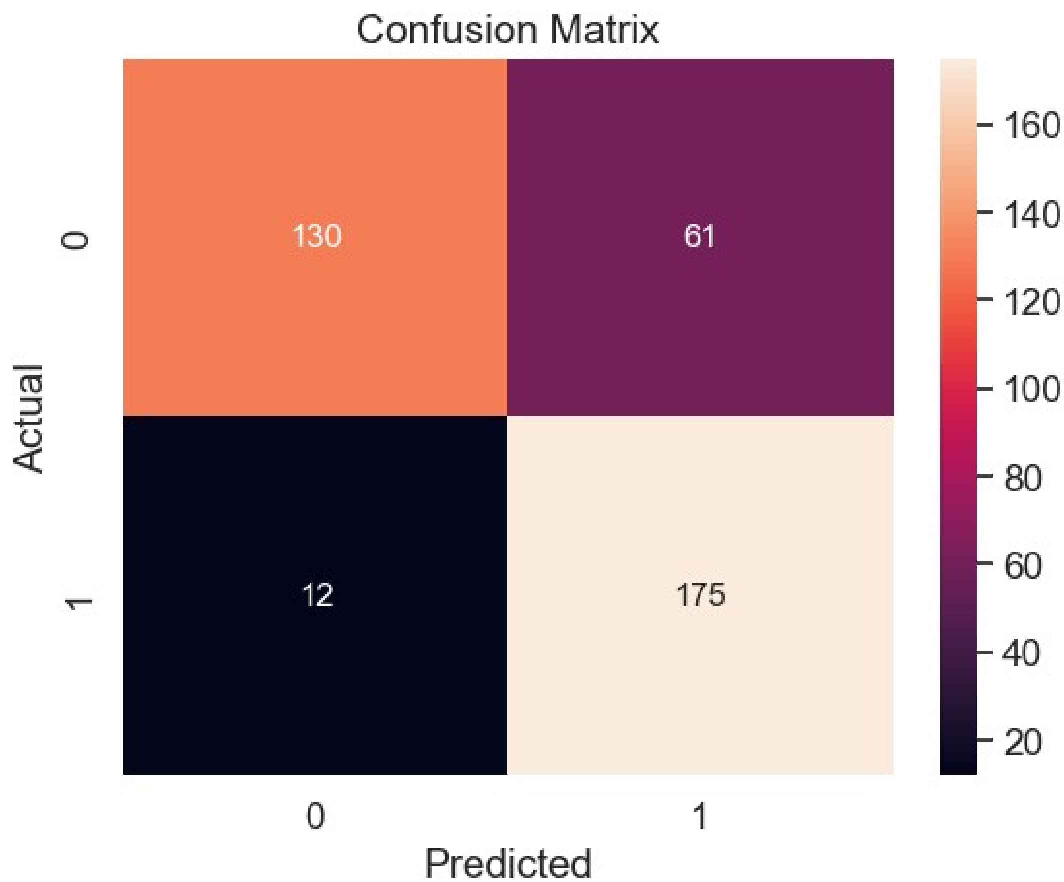

```
[[135  56]
 [ 18 169]]
```

Decision Tree classifier

```
def treeClassifier():
    # 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)
```

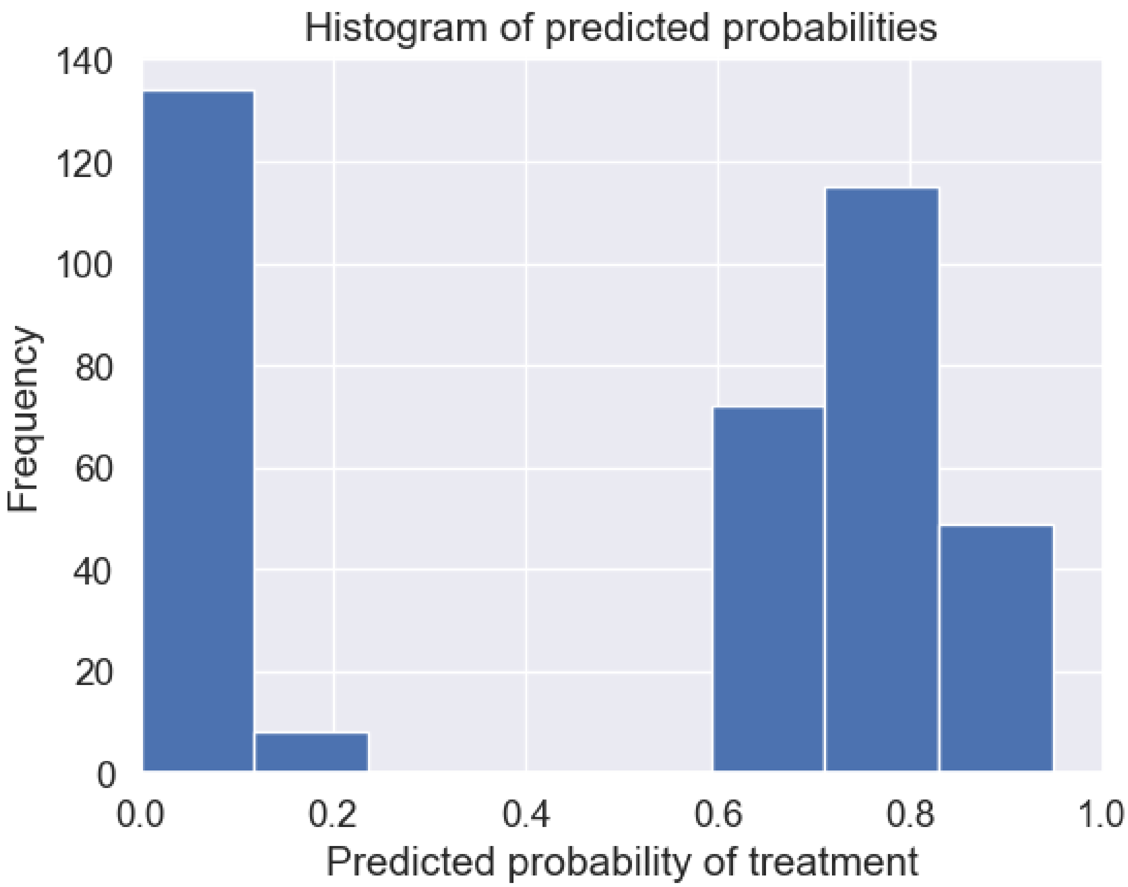
```python
    # make class predictions for the testing set
    y_pred_class = tree.predict(X_test)

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

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

    #Data for final graph
    methodDict['Tree clas.'] = accuracy_score * 100

treeClassifier()
```

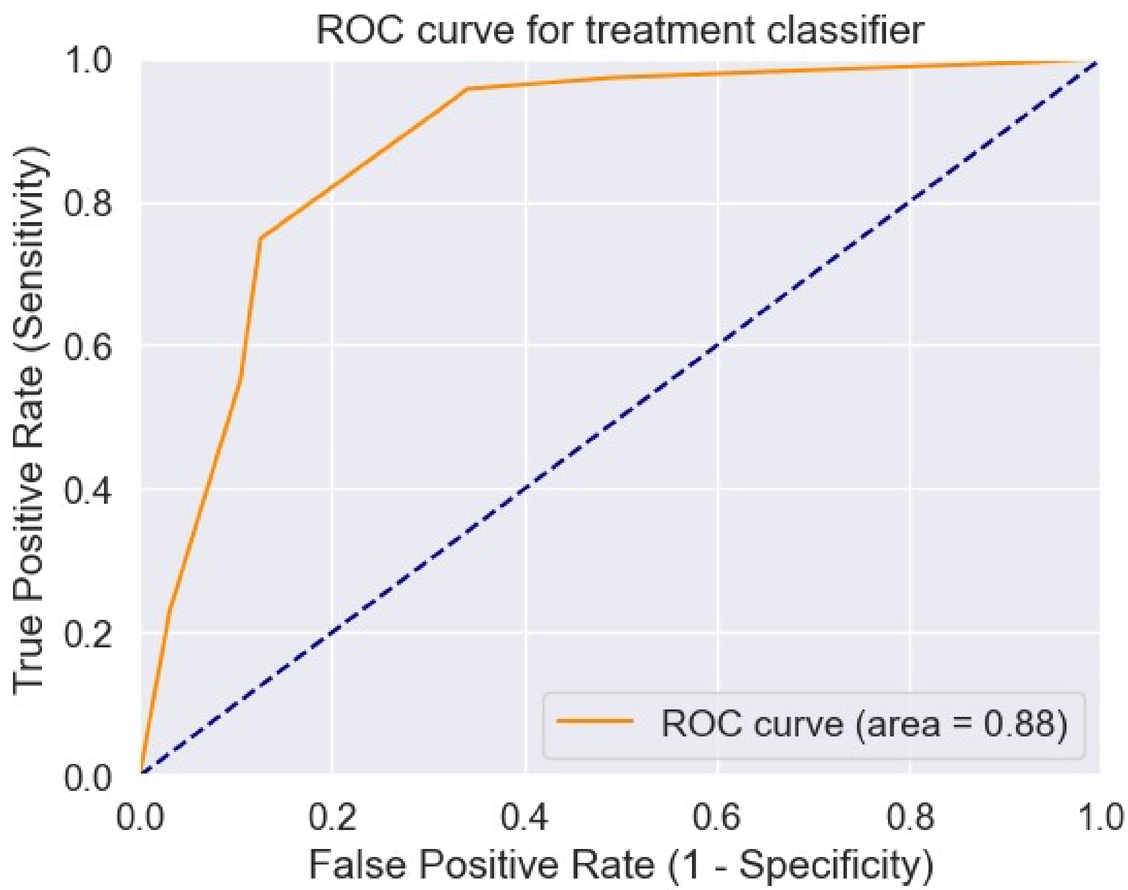
```
Rand. Best Score:  0.8305206349206349
Rand. Best Params:  {'criterion': 'entropy', 'max_depth': 3,
'max_features': 6, 'min_samples_leaf': 7, 'min_samples_split': 8}
[0.831, 0.829, 0.831, 0.819, 0.831, 0.831, 0.821, 0.83, 0.831, 0.831,
0.831, 0.831, 0.831, 0.831, 0.816, 0.811, 0.831, 0.81, 0.831, 0.831]
########### Tree classifier ###############
Accuracy: 0.8068783068783069
Null accuracy:
 treatment
0    191
1    187
Name: count, 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]
```




```
Classification Accuracy: 0.8068783068783069
Classification Error: 0.19312169312169314
False Positive Rate: 0.3193717277486911
Precision: 0.7415254237288136
AUC Score: 0.8082285746283282
Cross-validated AUC: 0.8845857468211298
First 10 predicted responses:
 [1 0 0 0 1 1 0 1 1 1]
First 10 predicted probabilities of class members:
 [[0.18823529 0.81176471]
 [0.95575221 0.04424779]
 [0.95575221 0.04424779]
 [0.95575221 0.04424779]
 [0.37583893 0.62416107]
 [0.05172414 0.94827586]
 [0.90384615 0.09615385]
 [0.37583893 0.62416107]
 [0.22018349 0.77981651]
 [0.22018349 0.77981651]]
First 10 predicted probabilities:
 [[0.81176471]
 [0.04424779]
```

```
 [0.04424779]
 [0.04424779]
 [0.62416107]
 [0.94827586]
 [0.09615385]
 [0.62416107]
 [0.77981651]
 [0.77981651]]
```





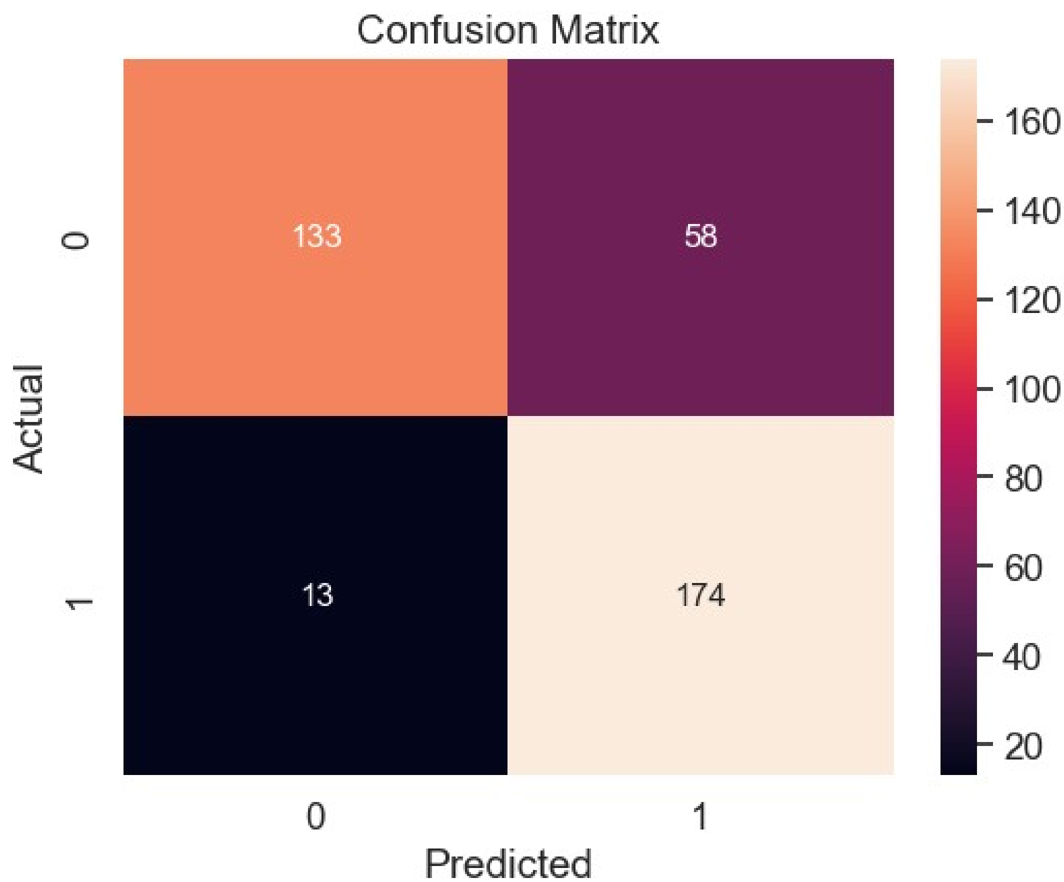

```
[[130  61]
 [ 12 175]]
```

Random Forests

```
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_samples_split=2, n_estimators = 20,
```

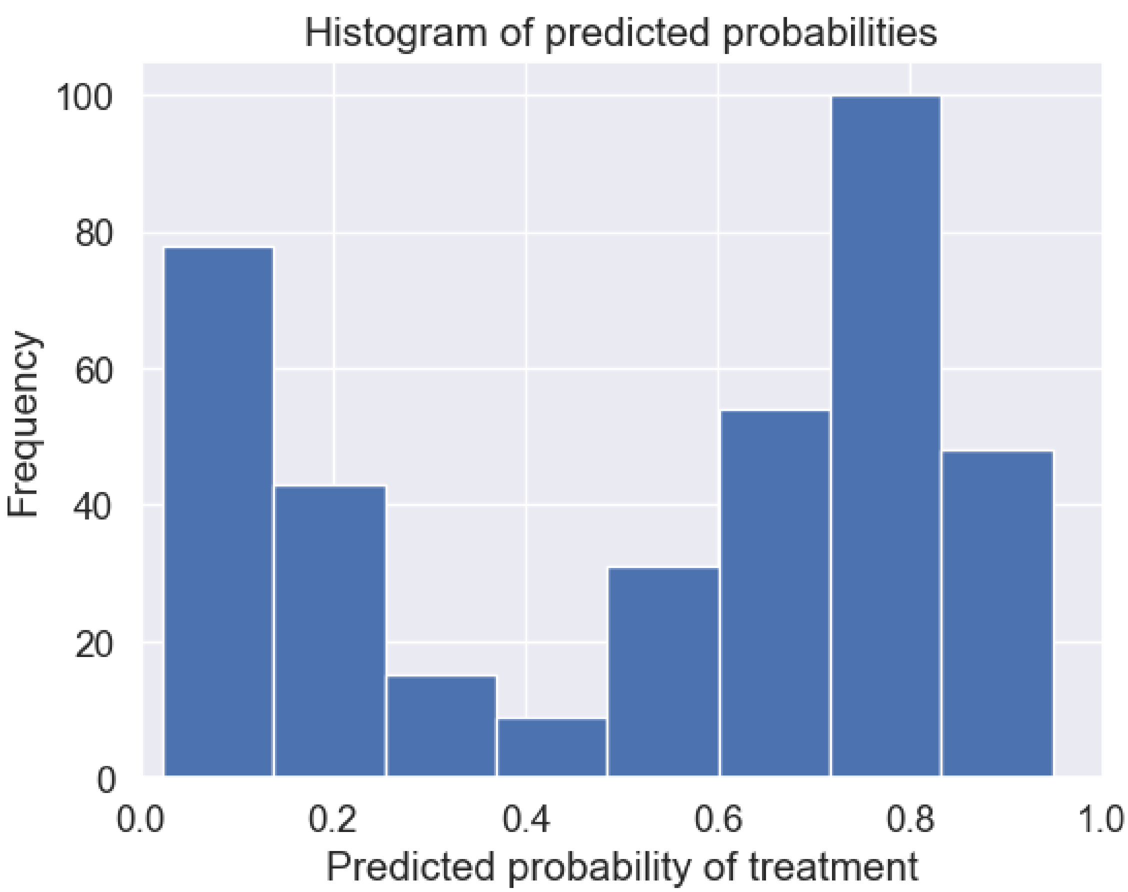
```
random_state = 1)
    my_forest = forest.fit(X_train, y_train)

    # 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

randomForest()
```

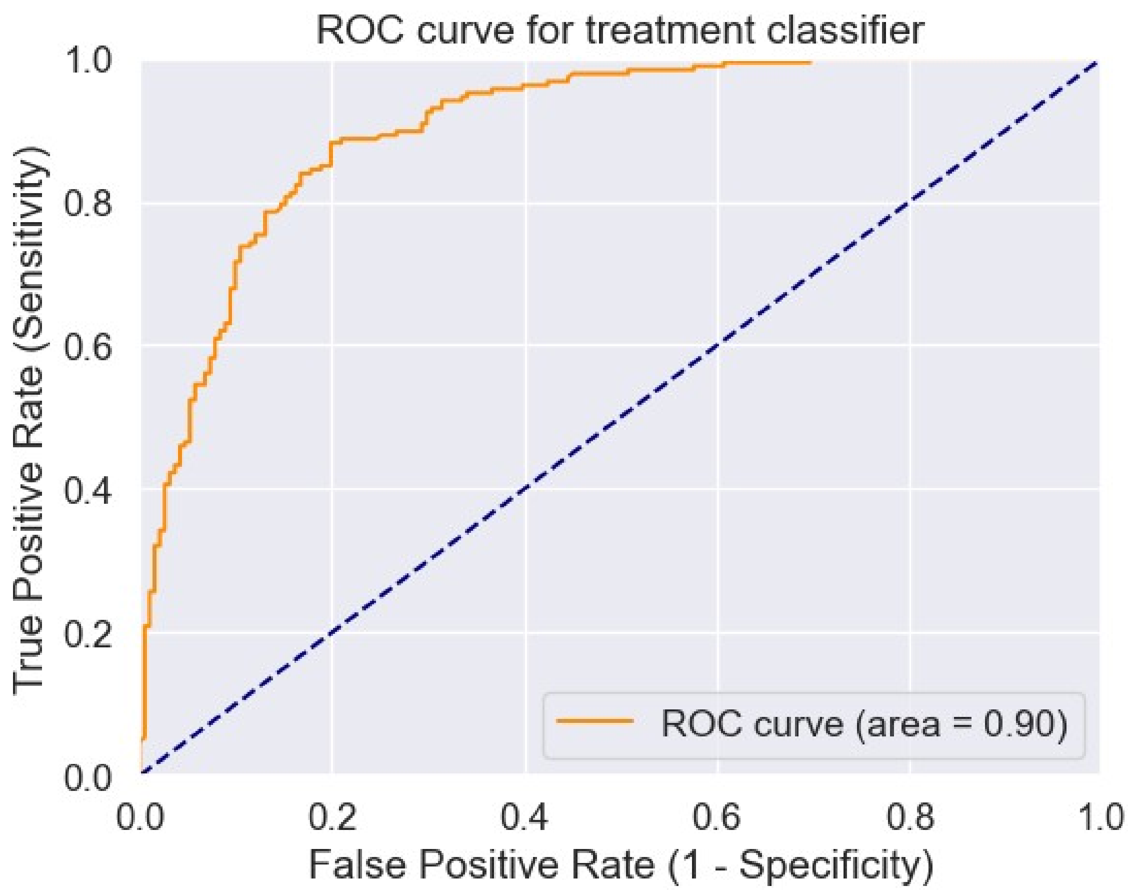
```
Rand. Best Score:  0.8305206349206349
Rand. Best Params:  {'criterion': 'entropy', 'max_depth': 3,
'max_features': 6, 'min_samples_leaf': 7, 'min_samples_split': 8}
[0.832, 0.831, 0.831, 0.833, 0.831, 0.833, 0.831, 0.831, 0.831, 0.831,
0.832, 0.831, 0.831, 0.831, 0.832, 0.831, 0.831, 0.831, 0.831, 0.831]
########### Random Forests ###############
Accuracy: 0.8121693121693122
Null accuracy:
 treatment
0    191
1    187
Name: count, 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]
```




```
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]]
```





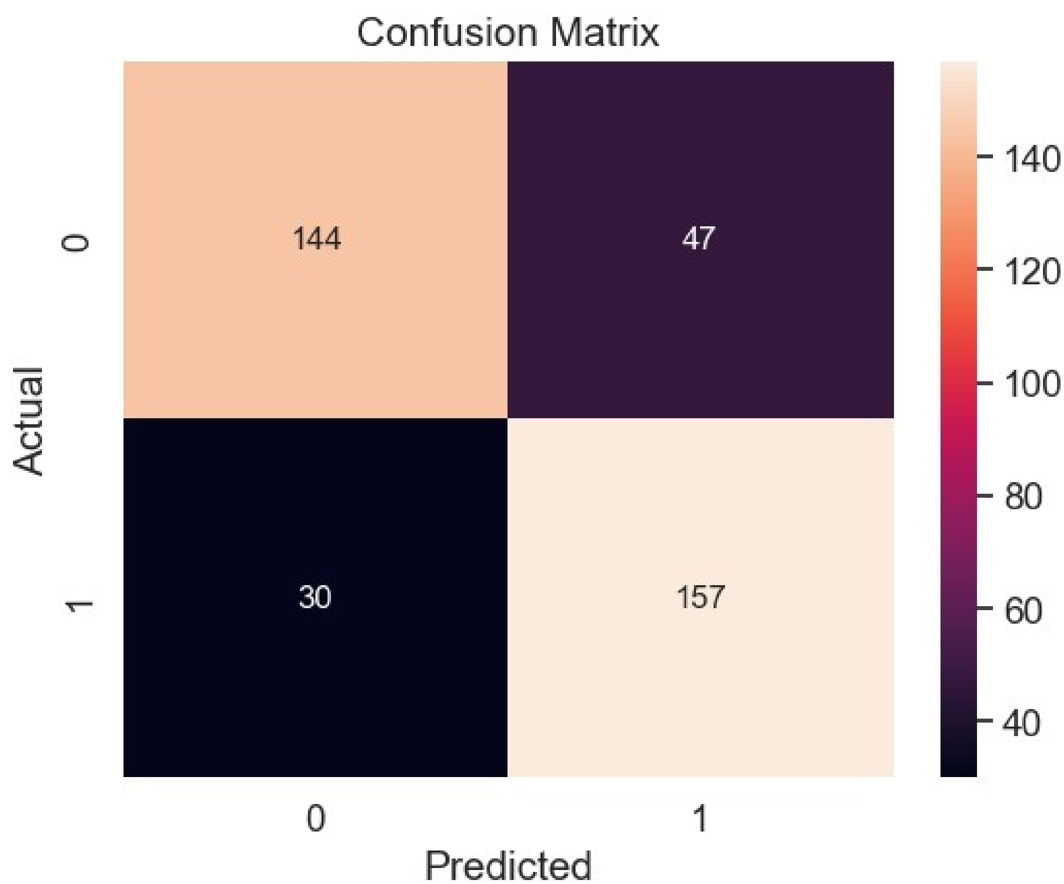

```
[[133  58]
 [ 13 174]]
```

Bagging

```
def bagging():
    # Building and fitting
    bag = BaggingClassifier(DecisionTreeClassifier(), max_samples=1.0,
max_features=1.0, bootstrap_features=False)
    bag.fit(X_train, y_train)

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

    print('########### Bagging ###############')

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

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

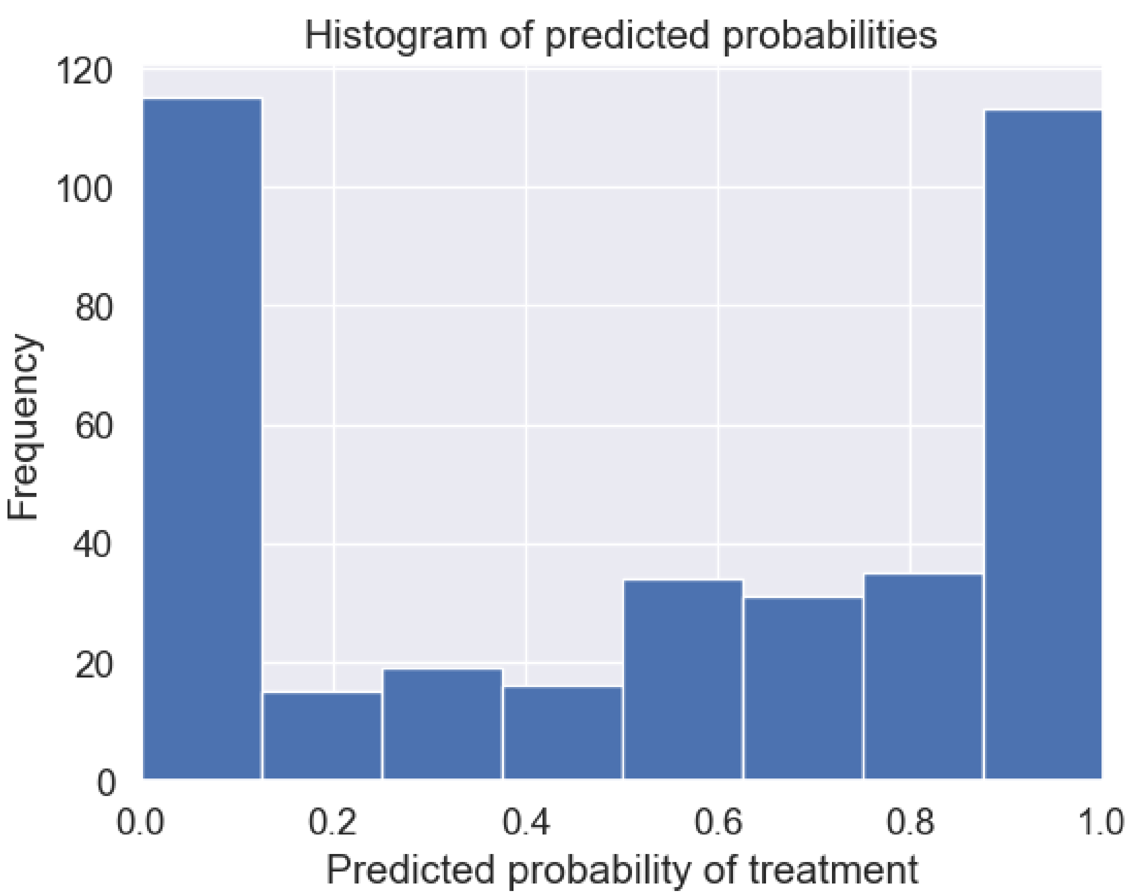
```
bagging()
```

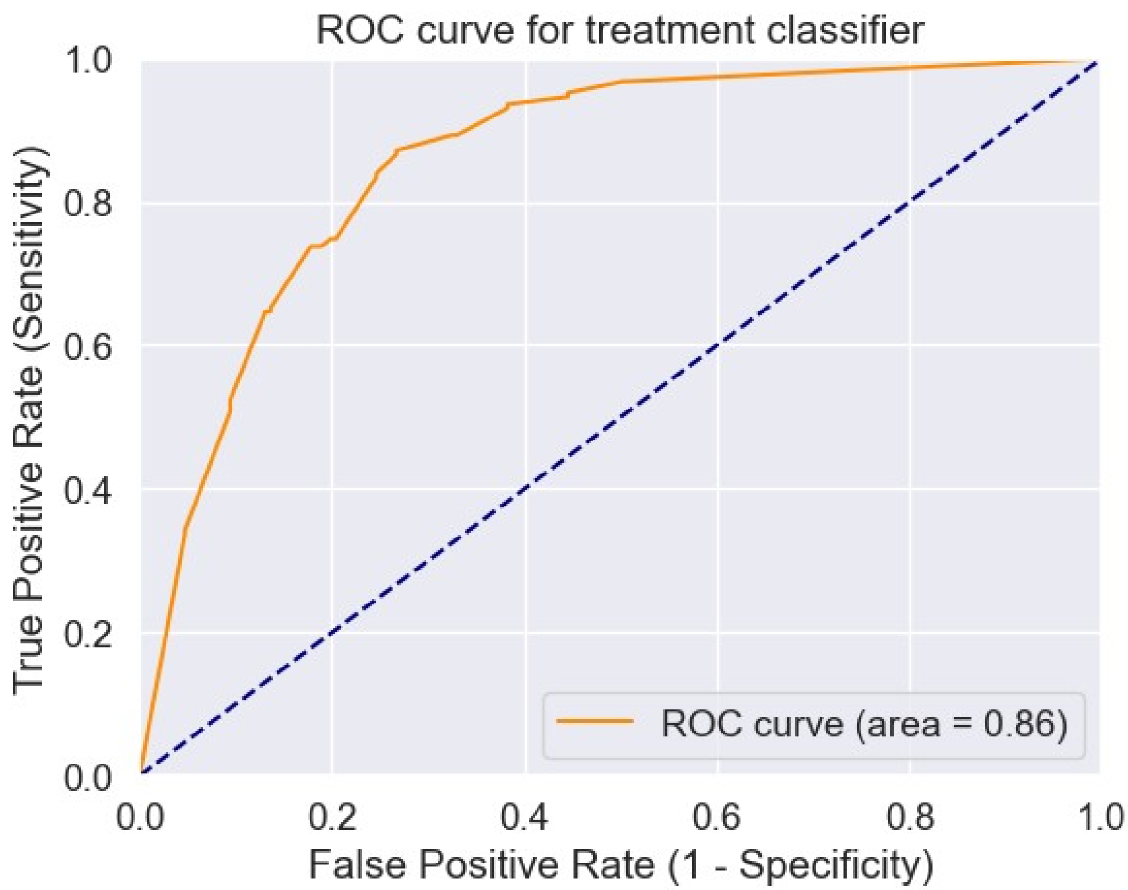
```
########### Bagging ###############
Accuracy: 0.7962962962962963
Null accuracy:
 treatment
0    191
1    187
Name: count, dtype: int64
Percentage of ones: 0.4947089947089947
Percentage of zeros: 0.5052910052910053
True: [0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1 0 0]
Pred: [1 0 0 0 0 1 0 0 1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




```
Classification Accuracy: 0.7962962962962963
Classification Error: 0.20370370370370372
False Positive Rate: 0.24607329842931938
Precision: 0.7696078431372549
AUC Score: 0.7967494470420248
Cross-validated AUC: 0.8516097220698315
First 10 predicted responses:
 [1 0 0 0 0 1 0 0 1 1]
```

```
First 10 predicted probabilities of class members:
 [[0.33333333 0.66666667]
 [1.         0.        ]
 [1.         0.        ]
 [0.8        0.2       ]
 [0.8        0.2       ]
 [0.1        0.9       ]
 [1.         0.        ]
 [0.9        0.1       ]
 [0.         1.        ]
 [0.1        0.9       ]]
First 10 predicted probabilities:
 [[0.66666667]
 [0.        ]
 [0.        ]
 [0.2       ]
 [0.2       ]
 [0.9       ]
 [0.        ]
 [0.1       ]
 [1.        ]
 [0.9       ]]
```






```
[[144  47]
 [ 30 157]]
```

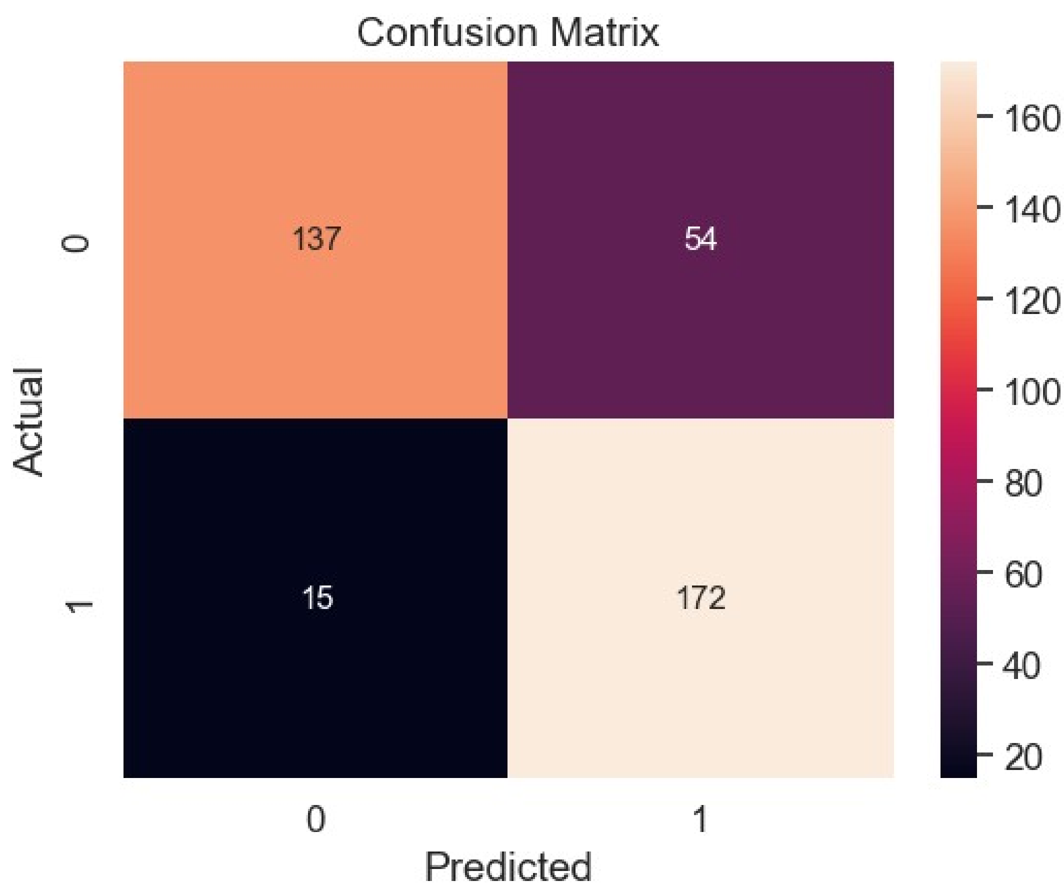
Boosting

```
def boosting():
    # Building and fitting
    clf = DecisionTreeClassifier(criterion='entropy', max_depth=1)
    boost = AdaBoostClassifier(base_estimator=clf, n_estimators=500)
    boost.fit(X_train, y_train)

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

    print('########### Boosting ###############')

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

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

```
boosting()
```

```
b:\Anaconda3\envs\_AIMLDataScience_env\Lib\site-packages\sklearn\
ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to
`estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
```

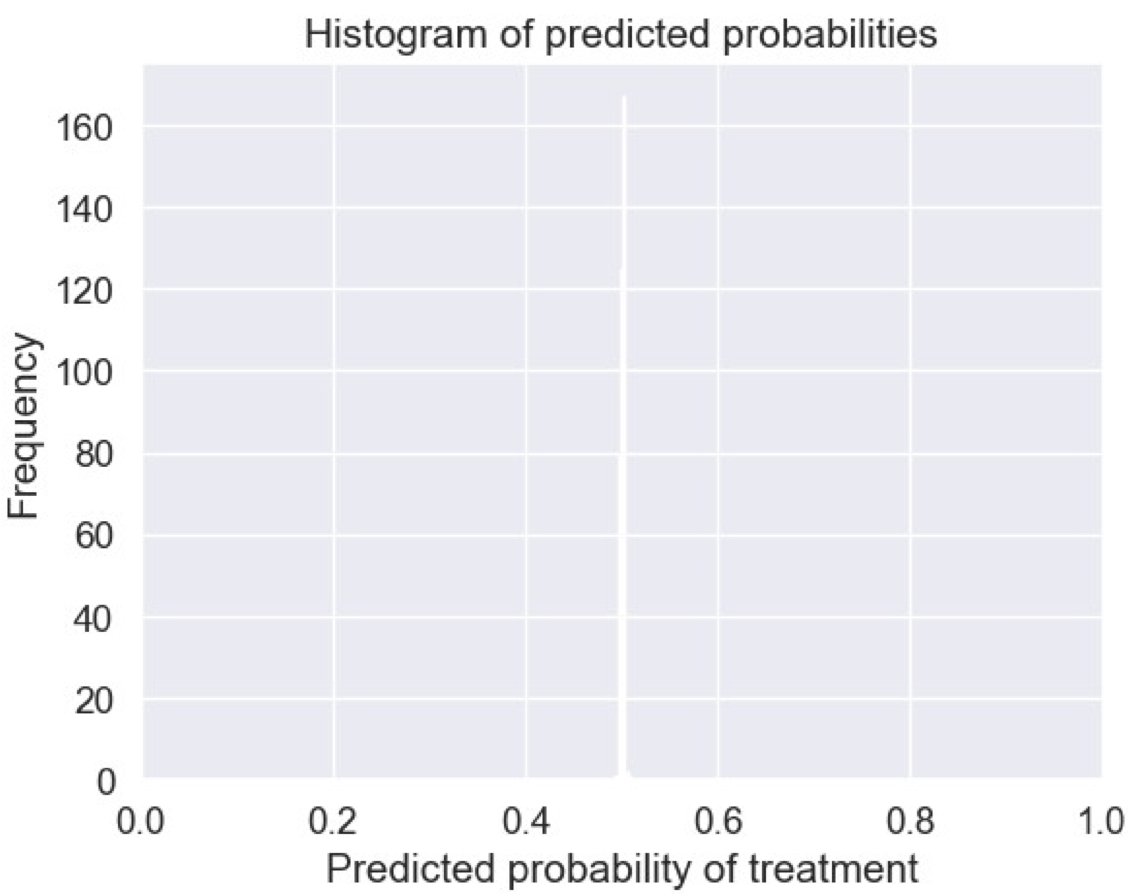
```
########### Boosting ###############
Accuracy: 0.8174603174603174
Null accuracy:
 treatment
0    191
1    187
Name: count, dtype: int64
Percentage of ones: 0.4947089947089947
Percentage of zeros: 0.5052910052910053
True: [0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1 0 0]
Pred: [1 0 0 0 0 1 0 1 1 1 0 1 1 0 1 1 1 1 0 0 0 0 1 0 0]
```

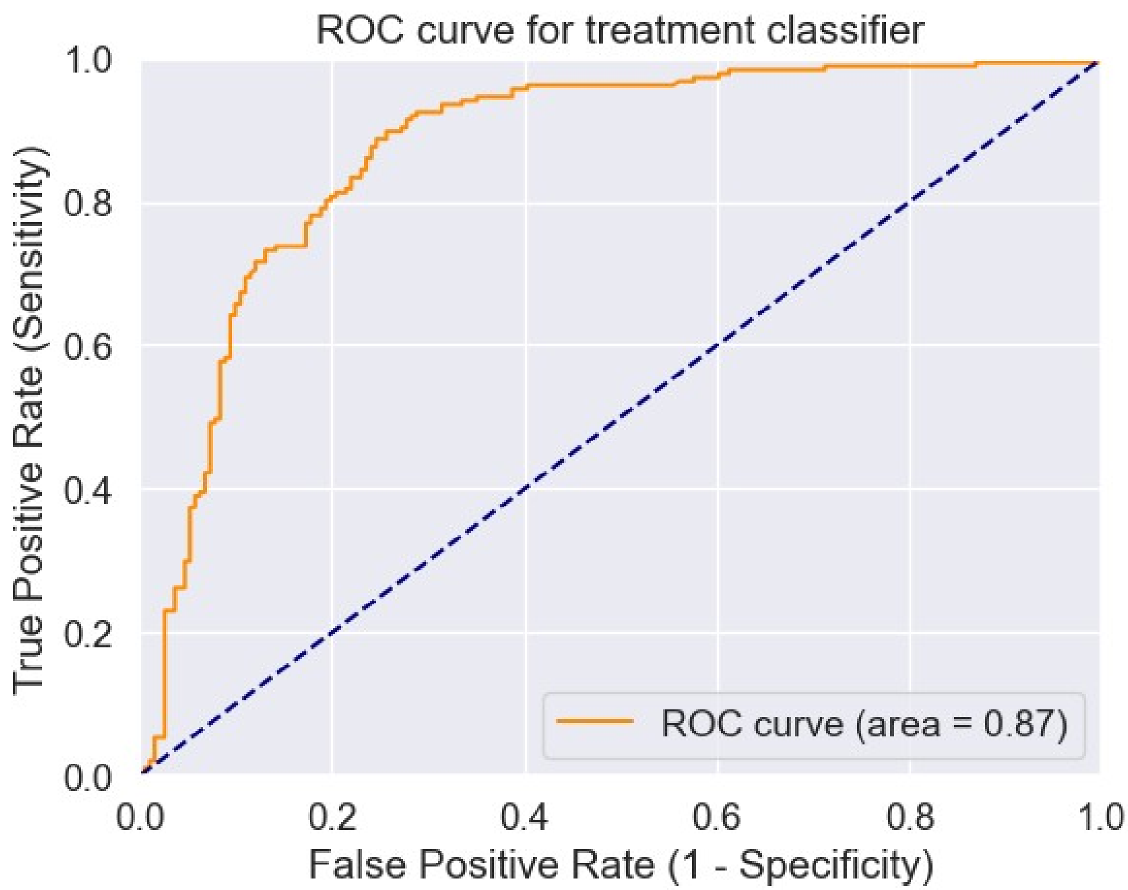



```
Classification Accuracy: 0.8174603174603174
Classification Error: 0.18253968253968256
False Positive Rate: 0.28272251308900526
```

```
Precision: 0.7610619469026548
AUC Score: 0.8185317915838397

b:\Anaconda3\envs\_AIMLDataScience_env\Lib\site-packages\sklearn\
ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to
`estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
b:\Anaconda3\envs\_AIMLDataScience_env\Lib\site-packages\sklearn\
ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to
`estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
b:\Anaconda3\envs\_AIMLDataScience_env\Lib\site-packages\sklearn\
ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to
`estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
b:\Anaconda3\envs\_AIMLDataScience_env\Lib\site-packages\sklearn\
ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to
`estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
b:\Anaconda3\envs\_AIMLDataScience_env\Lib\site-packages\sklearn\
ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to
`estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
b:\Anaconda3\envs\_AIMLDataScience_env\Lib\site-packages\sklearn\
ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to
`estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
b:\Anaconda3\envs\_AIMLDataScience_env\Lib\site-packages\sklearn\
ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to
`estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
b:\Anaconda3\envs\_AIMLDataScience_env\Lib\site-packages\sklearn\
ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to
`estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
b:\Anaconda3\envs\_AIMLDataScience_env\Lib\site-packages\sklearn\
ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to
`estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(
b:\Anaconda3\envs\_AIMLDataScience_env\Lib\site-packages\sklearn\
ensemble\_base.py:166: FutureWarning: `base_estimator` was renamed to
`estimator` in version 1.2 and will be removed in 1.4.
  warnings.warn(

Cross-validated AUC: 0.8746279095195426
First 10 predicted responses:
 [1 0 0 0 0 1 0 1 1 1]
First 10 predicted probabilities of class members:
 [[0.49924555 0.50075445]
 [0.50285507 0.49714493]
```

```
 [0.50291786 0.49708214]
 [0.50127788 0.49872212]
 [0.50013552 0.49986448]
 [0.49796157 0.50203843]
 [0.50046371 0.49953629]
 [0.49939483 0.50060517]
 [0.49921757 0.50078243]
 [0.49897133 0.50102867]]
First 10 predicted probabilities:
 [[0.50075445]
 [0.49714493]
 [0.49708214]
 [0.49872212]
 [0.49986448]
 [0.50203843]
 [0.49953629]
 [0.50060517]
 [0.50078243]
 [0.50102867]]
```





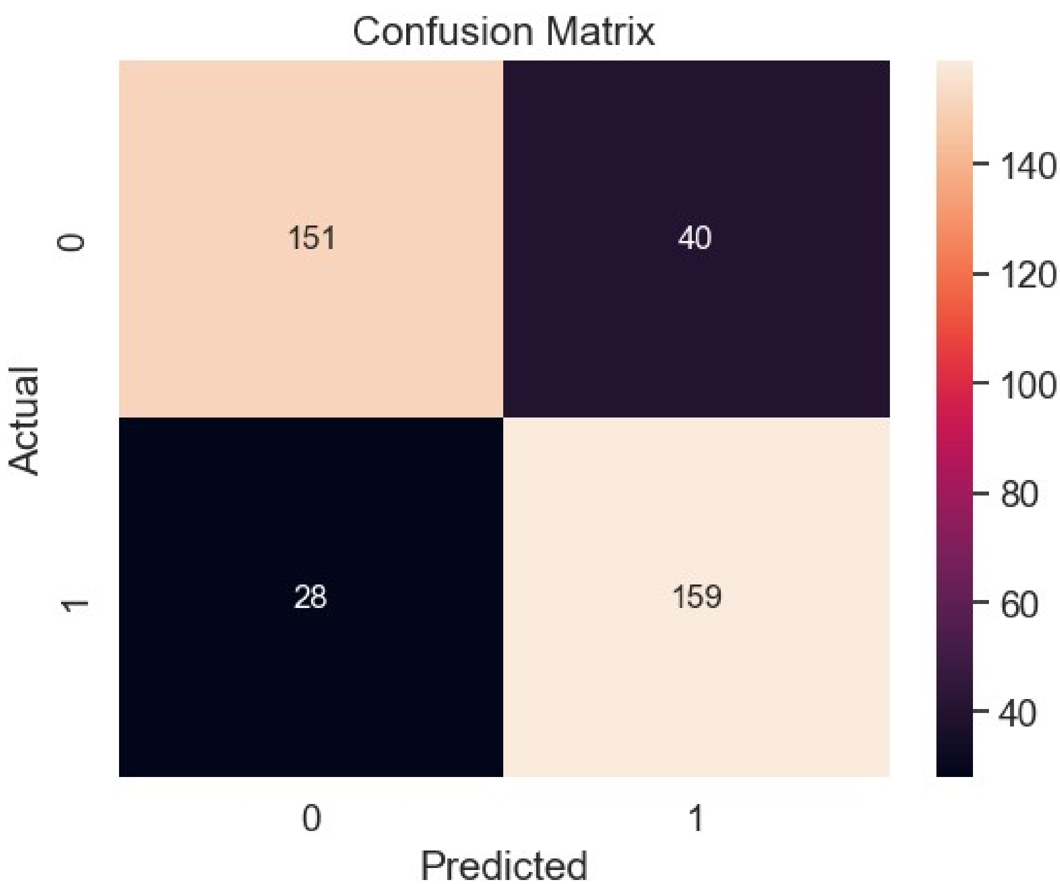

```
[[137  54]
 [ 15 172]]
```

Stacking

```
def stacking():
    # Building and fitting
    clf1 = KNeighborsClassifier(n_neighbors=1)
    clf2 = RandomForestClassifier(random_state=1)
    clf3 = GaussianNB()
    lr = LogisticRegression()
    stack = StackingClassifier(classifiers=[clf1, clf2, clf3],
meta_classifier=lr)
    stack.fit(X_train, y_train)

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

    print('########### Stacking ###############')
```

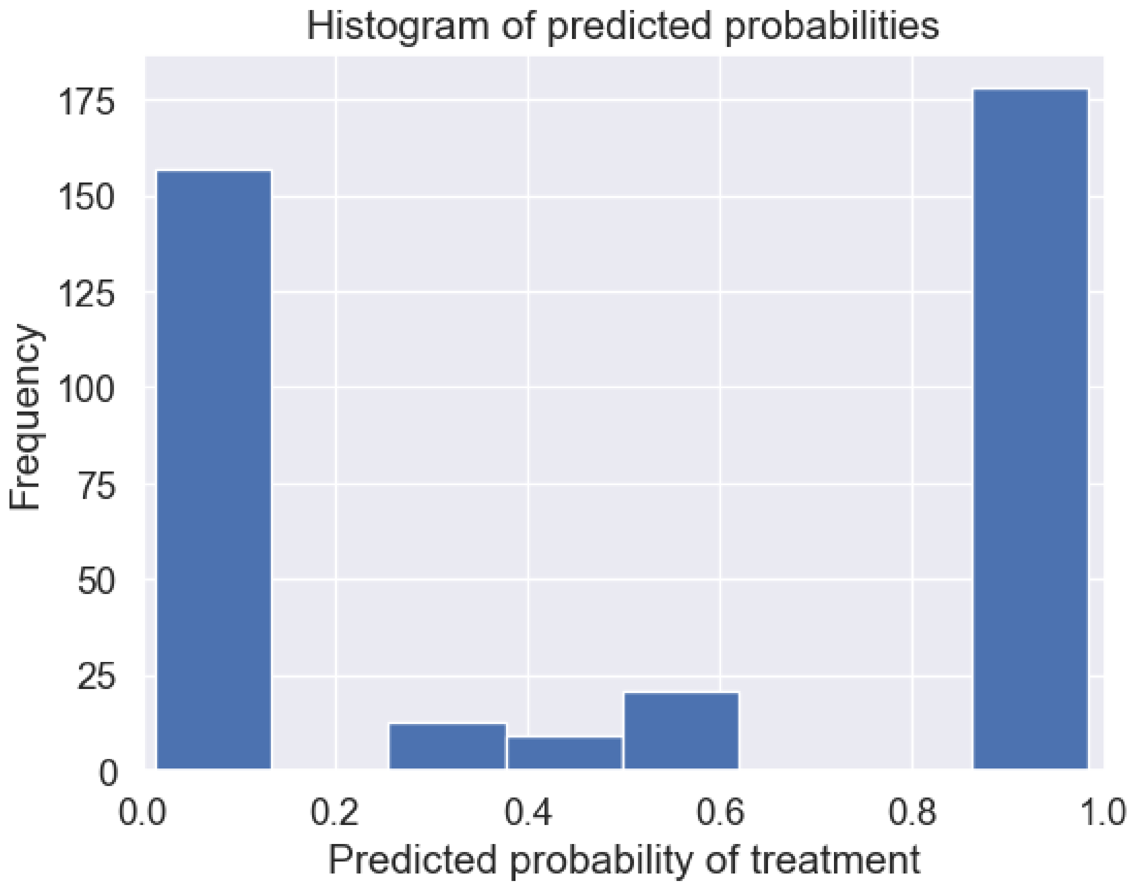
```python
    accuracy_score = evalClassModel(stack, y_test, y_pred_class, True)

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


stacking()
```

```
########### Stacking ###############
Accuracy: 0.8201058201058201
Null accuracy:
 treatment
0    191
1    187
Name: count, 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 0 1 0 0 1 1 0 1 1 1 1 1 0 1 0 0 0 0 1 0 0]
```

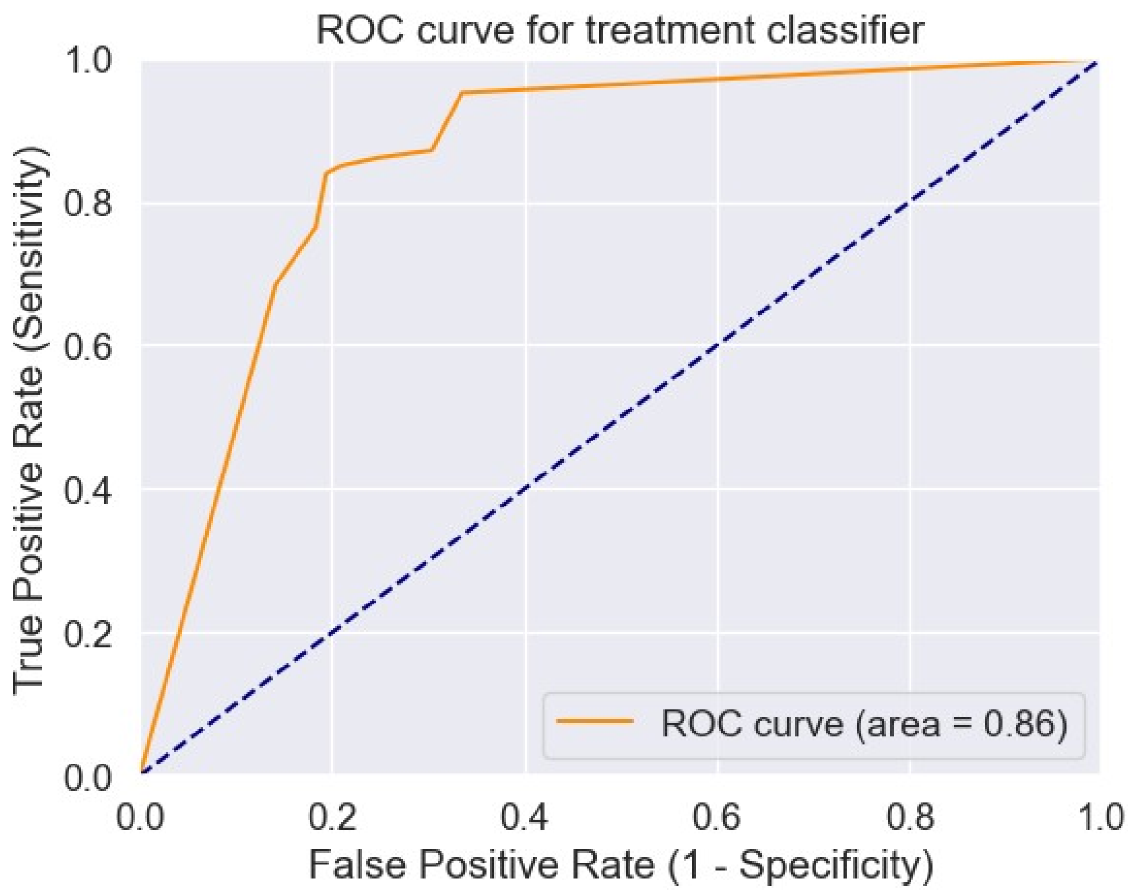



```
Classification Accuracy: 0.8201058201058201
Classification Error: 0.17989417989417988
```

```
False Positive Rate: 0.2094240837696335
Precision: 0.7989949748743719
AUC Score: 0.8204216479547554
Cross-validated AUC: 0.8431811731188892
First 10 predicted responses:
 [1 0 0 0 0 1 0 0 1 1]
First 10 predicted probabilities of class members:
 [[0.01710346 0.98289654]
 [0.98675465 0.01324535]
 [0.98675465 0.01324535]
 [0.98675465 0.01324535]
 [0.98675465 0.01324535]
 [0.01710346 0.98289654]
 [0.98675465 0.01324535]
 [0.97307936 0.02692064]
 [0.03462234 0.96537766]
 [0.01710346 0.98289654]]
First 10 predicted probabilities:
 [[0.98289654]
 [0.01324535]
 [0.01324535]
 [0.01324535]
 [0.01324535]
 [0.98289654]
 [0.01324535]
 [0.02692064]
 [0.96537766]
 [0.98289654]]
```



Histogram of predicted probabilities
Frequency
175
150
125
100
75
50
25
0
0.0
0.2
0.4
0.6
0.8
1.0
Predicted probability of treatment




```
[[151  40]
 [ 28 159]]
```

9. Predicting with Neural Network

Create input functions

```
import tensorflow as tf
import argparse


batch_size = 100
train_steps = 1000

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.30, random_state=0)

def train_input_fn(features, labels, batch_size):
    """An input function for training"""
    # Convert the inputs to a Dataset.
```

```
    dataset = tf.data.Dataset.from_tensor_slices((dict(features),
labels))

    # Shuffle, repeat, and batch the examples.
    return dataset.shuffle(1000).repeat().batch(batch_size)

def eval_input_fn(features, labels, batch_size):
    """An input function for evaluation or prediction"""
    features=dict(features)
    if labels is None:
        # No labels, use only features.
        inputs = features
    else:
        inputs = (features, labels)

    # Convert the inputs to a Dataset.
    dataset = tf.data.Dataset.from_tensor_slices(inputs)

    # Batch the examples
    assert batch_size is not None, "batch_size must not be None"
    dataset = dataset.batch(batch_size)

    # Return the dataset.
    return dataset
```

Define the feature columns

A feature column is an object describing how the model should use raw input data from the features dictionary.

```
# Define Tensorflow feature columns
age = tf.feature_column.numeric_column("Age")
gender = tf.feature_column.numeric_column("Gender")
family_history = tf.feature_column.numeric_column("family_history")
benefits = tf.feature_column.numeric_column("benefits")
care_options = tf.feature_column.numeric_column("care_options")
anonymity = tf.feature_column.numeric_column("anonymity")
leave = tf.feature_column.numeric_column("leave")
work_interfere = tf.feature_column.numeric_column("work_interfere")
feature_columns = [age, gender, family_history, benefits,
care_options, anonymity, leave, work_interfere]
```

```
WARNING:tensorflow:From C:\Users\athar\AppData\Local\Temp\
ipykernel_10060\3225071575.py:2: numeric_column (from
tensorflow.python.feature_column.feature_column_v2) is deprecated and
will be removed in a future version.
Instructions for updating:
Use Keras preprocessing layers instead, either directly or via the
`tf.keras.utils.FeatureSpace` utility. Each of `tf.feature_column.*`
```

```
has a functional equivalent in `tf.keras.layers` for feature
preprocessing when training a Keras model.
```

Instantiate an Estimator

Our problem is a classic classification problem. We want to predict whether a patient has to be treated or not. We'll use tf.estimator.DNNClassifier for deep models that perform multi-class classification.

```
# Build a DNN with 2 hidden layers and 10 nodes in each hidden layer.
model = tf.estimator.DNNClassifier(feature_columns=feature_columns,
                                    hidden_units=[10, 10],

optimizer=tf.keras.optimizers.legacy.Adam(
                                      learning_rate=0.1
                                    ))
```

```
WARNING:tensorflow:From C:\Users\athar\AppData\Local\Temp\
ipykernel_10060\3956841515.py:2: DNNClassifierV2.__init__ (from
tensorflow_estimator.python.estimator.canned.dnn) is deprecated and
will be removed in a future version.
Instructions for updating:
Use tf.keras instead.
WARNING:tensorflow:From b:\Anaconda3\envs\_AIMLDataScience_env\Lib\
site-packages\tensorflow_estimator\python\estimator\head\
head_utils.py:54: BinaryClassHead.__init__ (from
tensorflow_estimator.python.estimator.head.binary_class_head) is
deprecated and will be removed in a future version.
Instructions for updating:
Use tf.keras instead.
WARNING:tensorflow:From b:\Anaconda3\envs\_AIMLDataScience_env\Lib\
site-packages\tensorflow_estimator\python\estimator\canned\dnn.py:759:
Estimator.__init__ (from
tensorflow_estimator.python.estimator.estimator) is deprecated and
will be removed in a future version.
Instructions for updating:
Use tf.keras instead.
WARNING:tensorflow:From b:\Anaconda3\envs\_AIMLDataScience_env\Lib\
site-packages\tensorflow_estimator\python\estimator\estimator.py:1844:
RunConfig.__init__ (from
tensorflow_estimator.python.estimator.run_config) is deprecated and
will be removed in a future version.
Instructions for updating:
Use tf.keras instead.
INFO:tensorflow:Using default config.
WARNING:tensorflow:Using temporary folder as model directory: C:\
Users\athar\AppData\Local\Temp\tmpk7sjx813
INFO:tensorflow:Using config: {'_model_dir': 'C:\\Users\\athar\\
AppData\\Local\\Temp\\tmpk7sjx813', '_tf_random_seed': None,
```

```
'_save_summary_steps': 100, '_save_checkpoints_steps': None,
'_save_checkpoints_secs': 600, '_session_config':
allow_soft_placement: true
graph_options {
  rewrite_options {
    meta_optimizer_iterations: ONE
  }
}
, '_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000,
'_log_step_count_steps': 100, '_train_distribute': None, '_device_fn':
None, '_protocol': None, '_eval_distribute': None,
'_experimental_distribute': None,
'_experimental_max_worker_delay_secs': None,
'_session_creation_timeout_secs': 7200, '_checkpoint_save_graph_def':
True, '_service': None, '_cluster_spec': ClusterSpec({}),
'_task_type': 'worker', '_task_id': 0, '_global_id_in_cluster': 0,
'_master': '', '_evaluation_master': '', '_is_chief': True,
'_num_ps_replicas': 0, '_num_worker_replicas': 1}
```

Train, Evaluate, and Predict

Now that we have an Estimator object, we can call methods to do the following:

- Train the model.
- Evaluate the trained model.
- Use the trained model to make predictions.

Train the model

The steps argument tells the method to stop training after a number of training steps.

```
model.train(input_fn=lambda:train_input_fn(X_train, y_train,
batch_size), steps=train_steps)
```

```
WARNING:tensorflow:From b:\Anaconda3\envs\_AIMLDataScience_env\Lib\
site-packages\tensorflow_estimator\python\estimator\estimator.py:385:
StopAtStepHook.__init__ (from
tensorflow.python.training.basic_session_run_hooks) is deprecated and
will be removed in a future version.
Instructions for updating:
Use tf.keras instead.
INFO:tensorflow:Calling model_fn.
WARNING:tensorflow:From b:\Anaconda3\envs\_AIMLDataScience_env\Lib\
site-packages\tensorflow_estimator\python\estimator\model_fn.py:250:
EstimatorSpec.__new__ (from
tensorflow_estimator.python.estimator.model_fn) is deprecated and will
be removed in a future version.
Instructions for updating:
Use tf.keras instead.
```

```
INFO:tensorflow:Done calling model_fn.
WARNING:tensorflow:From b:\Anaconda3\envs\_AIMLDataScience_env\Lib\
site-packages\tensorflow_estimator\python\estimator\estimator.py:1416:
NanTensorHook.__init__ (from
tensorflow.python.training.basic_session_run_hooks) is deprecated and
will be removed in a future version.
Instructions for updating:
Use tf.keras instead.
WARNING:tensorflow:From b:\Anaconda3\envs\_AIMLDataScience_env\Lib\
site-packages\tensorflow_estimator\python\estimator\estimator.py:1419:
LoggingTensorHook.__init__ (from
tensorflow.python.training.basic_session_run_hooks) is deprecated and
will be removed in a future version.
Instructions for updating:
Use tf.keras instead.
WARNING:tensorflow:From b:\Anaconda3\envs\_AIMLDataScience_env\Lib\
site-packages\tensorflow\python\training\
basic_session_run_hooks.py:232: SecondOrStepTimer.__init__ (from
tensorflow.python.training.basic_session_run_hooks) is deprecated and
will be removed in a future version.
Instructions for updating:
Use tf.keras instead.
WARNING:tensorflow:From b:\Anaconda3\envs\_AIMLDataScience_env\Lib\
site-packages\tensorflow_estimator\python\estimator\estimator.py:1456:
CheckpointSaverHook.__init__ (from
tensorflow.python.training.basic_session_run_hooks) is deprecated and
will be removed in a future version.
Instructions for updating:
Use tf.keras instead.
INFO:tensorflow:Create CheckpointSaverHook.
WARNING:tensorflow:From b:\Anaconda3\envs\_AIMLDataScience_env\Lib\
site-packages\tensorflow\python\training\monitored_session.py:579:
StepCounterHook.__init__ (from
tensorflow.python.training.basic_session_run_hooks) is deprecated and
will be removed in a future version.
Instructions for updating:
Use tf.keras instead.
WARNING:tensorflow:From b:\Anaconda3\envs\_AIMLDataScience_env\Lib\
site-packages\tensorflow\python\training\monitored_session.py:586:
SummarySaverHook.__init__ (from
tensorflow.python.training.basic_session_run_hooks) is deprecated and
will be removed in a future version.
Instructions for updating:
Use tf.keras instead.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Calling checkpoint listeners before saving checkpoint
0...
```

```
INFO:tensorflow:Saving checkpoints for 0 into C:\Users\athar\AppData\
Local\Temp\tmpk7sjx813\model.ckpt.
INFO:tensorflow:Calling checkpoint listeners after saving checkpoint
0...
WARNING:tensorflow:From b:\Anaconda3\envs\_AIMLDataScience_env\Lib\
site-packages\tensorflow\python\training\monitored_session.py:1455:
SessionRunArgs.__new__ (from
tensorflow.python.training.session_run_hook) is deprecated and will be
removed in a future version.
Instructions for updating:
Use tf.keras instead.
WARNING:tensorflow:From b:\Anaconda3\envs\_AIMLDataScience_env\Lib\
site-packages\tensorflow\python\training\monitored_session.py:1454:
SessionRunContext.__init__ (from
tensorflow.python.training.session_run_hook) is deprecated and will be
removed in a future version.
Instructions for updating:
Use tf.keras instead.
WARNING:tensorflow:From b:\Anaconda3\envs\_AIMLDataScience_env\Lib\
site-packages\tensorflow\python\training\monitored_session.py:1474:
SessionRunValues.__new__ (from
tensorflow.python.training.session_run_hook) is deprecated and will be
removed in a future version.
Instructions for updating:
Use tf.keras instead.
INFO:tensorflow:loss = 0.6146792, step = 0
INFO:tensorflow:global_step/sec: 802.572
INFO:tensorflow:loss = 0.36849788, step = 100 (0.127 sec)
INFO:tensorflow:global_step/sec: 1349.14
INFO:tensorflow:loss = 0.31399375, step = 200 (0.073 sec)
INFO:tensorflow:global_step/sec: 1471.31
INFO:tensorflow:loss = 0.32438728, step = 300 (0.068 sec)
INFO:tensorflow:global_step/sec: 1467.63
INFO:tensorflow:loss = 0.43570128, step = 400 (0.068 sec)
INFO:tensorflow:global_step/sec: 1499.49
INFO:tensorflow:loss = 0.3671186, step = 500 (0.067 sec)
INFO:tensorflow:global_step/sec: 1485.53
INFO:tensorflow:loss = 0.38827735, step = 600 (0.068 sec)
INFO:tensorflow:global_step/sec: 1513.57
INFO:tensorflow:loss = 0.30714363, step = 700 (0.065 sec)
INFO:tensorflow:global_step/sec: 1418.35
INFO:tensorflow:loss = 0.31148607, step = 800 (0.071 sec)
INFO:tensorflow:global_step/sec: 1876.76
INFO:tensorflow:loss = 0.41515914, step = 900 (0.053 sec)
INFO:tensorflow:Calling checkpoint listeners before saving checkpoint
1000...
INFO:tensorflow:Saving checkpoints for 1000 into C:\Users\athar\
AppData\Local\Temp\tmpk7sjx813\model.ckpt.
INFO:tensorflow:Calling checkpoint listeners after saving checkpoint
```

```
1000...
INFO:tensorflow:Loss for final step: 0.455875.

<tensorflow_estimator.python.estimator.canned.dnn.DNNClassifierV2 at
0x1ad02a78890>
```

Evaluate the trained model

Now that the model has been trained, we can get some statistics on its performance. The following code block evaluates the accuracy of the trained model on the test data.

```
# Evaluate the model.
eval_result = model.evaluate(
    input_fn=lambda:eval_input_fn(X_test, y_test, batch_size))

print('\nTest set accuracy: {accuracy:0.2f}\n'.format(**eval_result))

#Data for final graph
accuracy = eval_result['accuracy'] * 100
methodDict['NN DNNClasif.'] = accuracy
```

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2023-11-07T09:15:55
WARNING:tensorflow:From b:\Anaconda3\envs\_AIMLDataScience_env\Lib\
site-packages\tensorflow\python\training\evaluation.py:260:
FinalOpsHook.__init__ (from
tensorflow.python.training.basic_session_run_hooks) is deprecated and
will be removed in a future version.
Instructions for updating:
Use tf.keras instead.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from C:\Users\athar\AppData\
Local\Temp\tmpk7sjx813\model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Inference Time : 0.38039s
INFO:tensorflow:Finished evaluation at 2023-11-07-09:15:55
INFO:tensorflow:Saving dict for global step 1000: accuracy =
0.7962963, accuracy_baseline = 0.505291, auc = 0.87921715,
auc_precision_recall = 0.849898, average_loss = 0.49389374,
global_step = 1000, label/mean = 0.49470899, loss = 0.49318892,
precision = 0.72540987, prediction/mean = 0.5608679, recall =
0.9465241
INFO:tensorflow:Saving 'checkpoint_path' summary for global step 1000:
C:\Users\athar\AppData\Local\Temp\tmpk7sjx813\model.ckpt-1000

Test set accuracy: 0.80
```

Making predictions (inferring) from the trained model

We now have a trained model that produces good evaluation results. We can now use the trained model to predict whether a patient needs treatment or not.

```
predictions =
list(model.predict(input_fn=lambda:eval_input_fn(X_train, y_train,
batch_size=batch_size)))
```

```
INFO:tensorflow:Calling model_fn.
WARNING:tensorflow:From b:\Anaconda3\envs\_AIMLDataScience_env\Lib\
site-packages\tensorflow_estimator\python\estimator\head\
base_head.py:786: ClassificationOutput.__init__ (from
tensorflow.python.saved_model.model_utils.export_output) is deprecated
and will be removed in a future version.
Instructions for updating:
Use tf.keras instead.
WARNING:tensorflow:From b:\Anaconda3\envs\_AIMLDataScience_env\Lib\
site-packages\tensorflow_estimator\python\estimator\head\
binary_class_head.py:561: RegressionOutput.__init__ (from
tensorflow.python.saved_model.model_utils.export_output) is deprecated
and will be removed in a future version.
Instructions for updating:
Use tf.keras instead.
WARNING:tensorflow:From b:\Anaconda3\envs\_AIMLDataScience_env\Lib\
site-packages\tensorflow_estimator\python\estimator\head\
binary_class_head.py:563: PredictOutput.__init__ (from
tensorflow.python.saved_model.model_utils.export_output) is deprecated
and will be removed in a future version.
Instructions for updating:
Use tf.keras instead.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from C:\Users\athar\AppData\
Local\Temp\tmpk7sjx813\model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
```

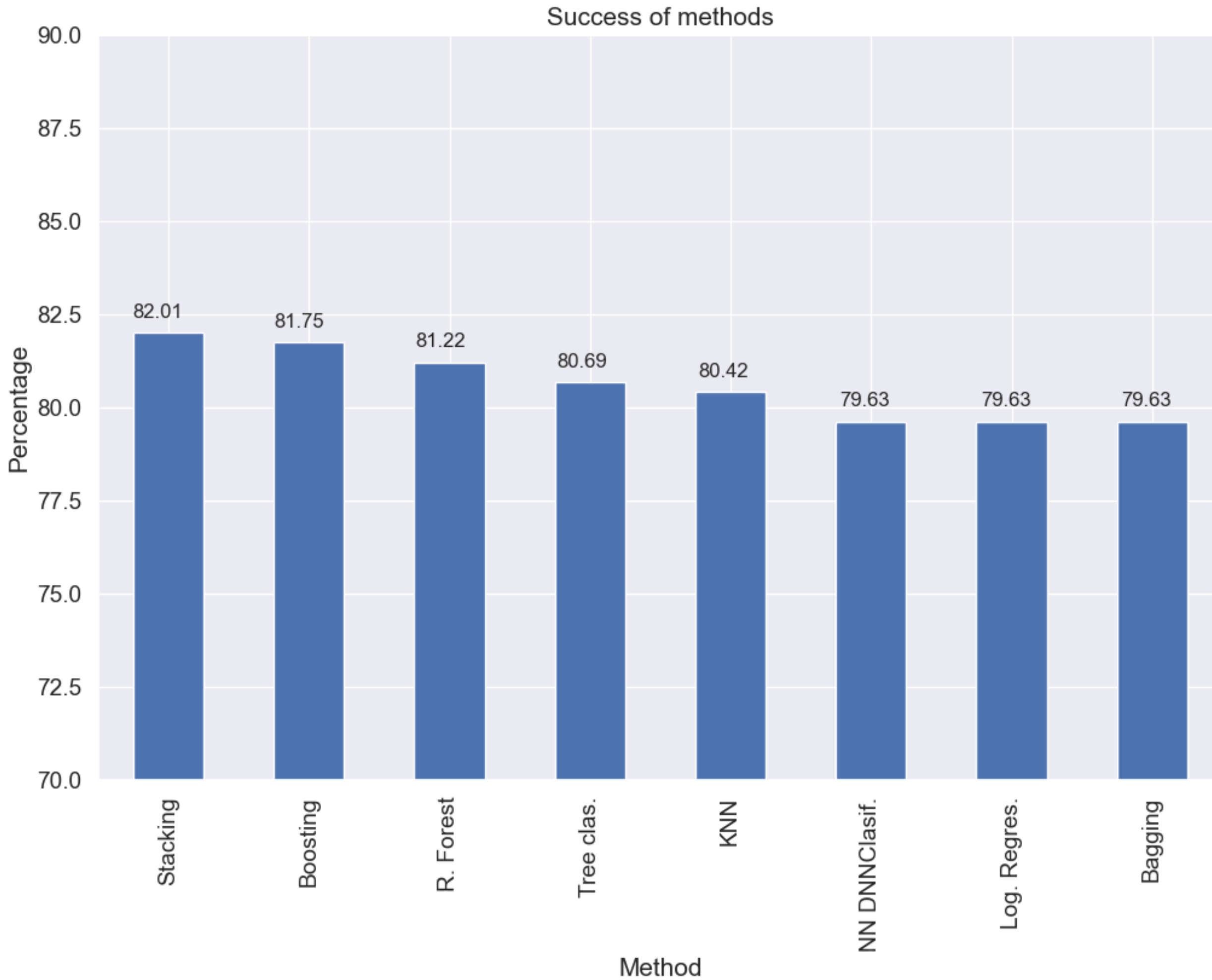
```
# Generate predictions from the model
template = ('\nIndex: "{}", Prediction is "{}" ({:.1f}%), expected
"{}"')

# Dictionary for predictions
col1 = []
col2 = []
col3 = []


for idx, input, p in zip(X_train.index, y_train, predictions):
    v  = p["class_ids"][0]
```

```python
    class_id = p['class_ids'][0]
    probability = p['probabilities'][class_id] # Probability

    # Adding to dataframe
    col1.append(idx) # Index
    col2.append(v) # Prediction
    col3.append(input) # Expecter


    #print(template.format(idx, v, 100 * probability, input))


results = pd.DataFrame({'index':col1, 'prediction':col2,
'expected':col3})
results.head()
```

	index	prediction	expected
0	929	0	0
1	901	1	1
2	579	1	1
3	367	1	1
4	615	1	1

10. Success method plot

```python
def plotSuccess():
    s = pd.Series(methodDict)
    s = s.sort_values(ascending=False)
    plt.figure(figsize=(12,8))
    #Colors
    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() * 1.005))
    plt.ylim([70.0, 90.0])
    plt.xlabel('Method')
    plt.ylabel('Percentage')
    plt.title('Success of methods')

    plt.show()

plotSuccess()
```



11. Creating predictions on test set

```
# Generate predictions with the best method
clf = AdaBoostClassifier()
clf.fit(X, y)
dfTestPredictions = clf.predict(X_test)

# Write 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})
# Save to file
# This file will be visible after publishing in the output section
results.to_csv('results.csv', index=False)
print(results)

     Index  Treatment
0        5          1
```

```
1      494           0
2       52           0
3      984           0
4      186           0
..     ...         ...
373   1084           1
374    506           0
375   1142           0
376   1124           0
377    689           1

[378 rows x 2 columns]
```