PREDICTING WHETHER A PERSON REQUIRES A TREATMENT FOR HIS/HER MENTAL ILLNESS USING MACHINE LEARNING

A Dissertation

Submitted by

Ashish Bhagavatula	180250125013
Karupakulu Supraja	180250125030
Kumathi Nikhitha Sowjanya	180250125039
Rishabh Bhargava	180250125058

Under the guidance of

MS. SUKESHINI RAMADASU

in partial fulfillment of the course of

Post-Graduate Diploma

in

Big Data Analytics

CDAC ACTS

Knowledge Park,Bengaluru

C-DAC,BENGALURU

CENTER FOR DEVELOPMENT OF ADVANCED COMPUTING

30/7/2018

CERTIFICATE

This is to certify that

Ashish Bhagavatula	180250125013
Karupakulu Supraja	180250125030
Kumathi Nikhitha Sowjanya	180250125039
Rishabh Bhargava	180250125058

have completed the full course of project work titled "**PREDICTING WHETHER PERSON REQUIRES A TREATMENT FOR HIS/HER MENTAL ILLNESS USING MACHINE LEARNING**" satisfactorily in partial fulfillment of Post-Graduate Diploma in Big Data Analytics at C-DAC ACTS,Knowledge Park,Bengaluru in the year 2018.

Ms. Sukeshini Ramadasu

(Project Guide)

Ms. Savithri Murali

(Course Co-ordinator)

TABLE OF CONTENTS

ACKNOWLEDGEMENT

We would like to thank the Big Data Analytics team at CDAC-ACTS for their continuous collaboration throughout the course.They played an important role in helping us to complete this course.

We are very glad to **Ms.Sukeshini Ramadasu** for her valuable guidance to work on this project .Her guidance and support helped us to overcome various obstacles and intricacies during the course of the project work.

We are thankful to **Ms.Savithri Murali**(Course Co-ordinator) who gave all the required support and kind coordination to provide all necessities like hardware,internet facility and extra lab hours to complete the project and throughout the course.

ABSTRACT

This project identifies the risk factors for mental illness and formulated a predictive model based on the identified variables. The study simulated the formulated model and validated the model with a view to developing a model for predicting the risk of mental illness. Following the review of literature in order to understand the body of knowledge surrounding mental illness and their corresponding risk factors, interview with mental experts was conducted in order to validate the identified variables. Naïve Bayes' and the Decision Trees' Classifiers were used to formulate the predictive model for the risk of mental illness based on the identified and validated variables using the machine learning algorithms. Data was collected from 1433 patients with an almost equal distribution of no, low, moderate and high risk of mental illness cases. The results showed that there were three classes of risk factors associated with mental illness, namely: biological factors, psychological factors and environmental factors. The results further showed that the formulation with Decision Trees Classifiers revealed the most relevant variables for the risks of mental illness such as losing anyone close. The study concluded that the variables identified by the Decision Trees algorithm can assist mental health experts to apply the rules deduced by the algorithm for the early detection of mental illness.

1.INTRODUCTION

The World Health Organization (WHO) defines mental health as 'a state of well-being in which the individual realizes his or her own abilities to cope with the normal stresses of life and work productively and fruitfully, and be able to make contribution to his or her community. Mental illness refers to all of the diagnosable mental disorders which are characterized by abnormalities in thinking, feelings or behaviours. Mental illness is closely related to vulnerability, both in its causes and in its effects. Globally, 14% of the global burden of disease is attributed to mental illness – with 75% of those affected being found in low-income countries – which includes a broad spectrum of diagnoses, from common mental illnesses such as anxiety and substance abuse, to severe illnesses like psychosis.

A study conducted in Uganda revealed that the term depression is not culturally acceptable amongst the population while another study conducted in Nigeria found that people responded with fear, avoidance and anger to those who were observed to have a mental illness. The stigma linked to mental illness can be attributed to lack of education, fear, religious reasoning and general prejudice.According to a study by the Grand Challenges in Global Mental Health Initiative, the biggest barrier to global mental health care is the lack of an evidence-based set of primary prevention intervention methods.

Data mining involves the identification of unseen patterns in information stored in database using machine learning algorithms. Data mining has a great potential to enable healthcare systems to use data more efficiently and effectively thereby reducing the likely costs associated with making decisions. Data mining techniques are very useful in healthcare domain. They provide better medical services to the patients and helps to the healthcare organizations in various medical management decisions.

Classification is one of the most popularly used methods of Data Mining in Healthcare sector. It divides data samples into target classes. The classification technique predicts the target class for each data points. With the help of classification approach a risk factor can be associated to patients by analyzing their patterns of diseases.

Machine learning algorithms provide means of obtaining objective unseen patterns from evidence-based information especially in the public health care sector. Therefore, there is a need for the development of a predictive model for the classification of the risks of mental illness based on information regarding the associated risk factors.

A comparative analysis is also done on some supervised machine learning algorithm to the prediction of the mental illness. The machine learning algorithms used were Naïve Bayes' Decision Trees ,Logistic Regression, Boosting, Bagging, Stacking, Random Forest with the use of feature selection algorithms to identify relevant features.

2. LITERATURE SURVEY AND OVERVIEW

Machine learning approach to predicting treatment outcome in depression, using clinical (rather than mechanistic) predictors. Since there are potentially a very large number of predictors, examining all possible predictors in an unbiased manner (sometimes called “data mining”) is most likely to produce a powerful prediction algorithm.

Machine learning approaches are well suited to this approach, because they can identify patterns of information in data, rather than focusing on individual predictors. They can therefore identify the combination of variables that most strongly predict the outcome. However, prediction algorithms generated in this way need to be independently validated. By definition, they will predict the outcome in the data set used to generate the algorithm (the discovery sample). The real test is whether they also predict similar outcomes in independent data sets (the replication sample). This avoids circularity, and increases the likelihood the algorithm will be clinically useful.

Machine learning is a subset of artificial intelligence in the field of computer science that often uses statistical techniques to give computers the ability to "learn" (i.e., progressively improve performance on a specific task) with data, without being explicitly programmed.

Machine learning is closely related to (and often overlaps with) computational statistics, which also focuses on prediction-making through the use of computers. It has strong ties to mathematical optimization, which delivers methods, theory and application domains to the field. Machine learning is sometimes conflated with data mining, where the latter subfield focuses more on exploratory data analysis and is known as unsupervised learning.Machine learning can also be unsupervised and be used to learn and establish baseline behavioral profiles for various entities and then used to find meaningful anomalies.

In classification, inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one or more (multi-label classification) of these classes. This is typically tackled in a supervised manner.

A. DECISION TREE CLASSIFIER:

A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal, but are also a popular tool in machine learning.

B. LOGISTIC REGRESSION:

In regression analysis, logistic regression or logit regression is estimating the parameters of a logistic model. More formally, a logistic model is one where the log-odds of the probability of an event is a linear combination of independent or predictor variables. The two possible dependent variable values are often labelled as "0" and "1", which represent outcomes.

C. K NEAREST NEIGHBOUR CLASSIFIER:

In pattern recognition, the k-nearest neighbours algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression.

D. RANDOM FOREST:

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

E. BAGGING:

Bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid overfitting. Although it is usually applied to decision tree methods, it can be used with any type of method. Bagging is a special case of the model averaging approach.

F.BOOSTING:

Boosting is a machine learning ensemble meta-algorithm for primarily reducing bias, and also variance in supervised learning, and a family of machine learning algorithms that convert weak learners to strong ones.

G. STACKING:

Stacking (sometimes called stacked generalization) involves training a learning algorithm to combine the predictions of several other learning algorithms. First, all of the other algorithms are trained using the available data, then a combiner algorithm is trained to make a final prediction using all the predictions of the other algorithms as additional inputs.

H. PYTHON DOCUMENTATION:

Python language is one of the most flexible languages and can be used for various purposes. Python has gained huge popularity base of this. Python does contain special libraries for machine learning namely scipy and numpy which great for linear algebra and getting to know kernel methods of machine learning. The language is great to use when working with machine learning algorithms and has easy syntax relatively.

3. SOFTWARE REQUIREMENTS

A software requirements specification (SRS) are the complete description of the behavior of the system to be developed. The functional requirement includes what the software should do and the non functional requirement includes the constraints on the design and implementation. Requirements must be measurable, testable, related to identified needs and opportunities and defined to the level of detailed sufficient for the design.

The functionality of the software has to be directly perceived by the users. The common understanding between the user and developers is captured in the requirements. Writing of software requirement specification reduces development effort, careful review of the document can be reveal omissions, misunderstandings, and the inconsistencies early in the development life cycle, so that these problems can be rectified easily at the earliest.

I. FUNCTION:

Here our main objective is to predict whether a patient should be treated of his/her mental illness or not according to the values obtained in the dataset. The data is a survey conducted among various IT companies with 63 questions and about 1430 entries.

Our aim is to find out which features contributes more to predicting the mental health of the employees: like age, family background etc. To visualize the correlation between each of these features, and to fit this data in different models - random forest, KNN, Regression, tree classification etc to predict such cases on test data set.

In the end, we will also figure out which model gives out the best result of these by comparing the accuracy.

J. LIMITATIONS:

As is typical, this analysis is unfortunately is limited by many features of the study design and the data available. Our model does not fit the inference set very well, which suggests that our model is not a strong representation of the data-generating process. This lack of fit also suggests that our model is not well-specified for our model of error to be accurate. In simpler terms, it would suggest that the p-values that measure the significance of our variables may be incorrect.

We also have a limited number of individuals who identify as non-binary. This group is especially interesting in our dataset because they seem to present high rates of diagnosis. It is very possible that working as a non-binary individual in a society that pressures one to fit the binary can cause issues for the mental health of non-binary individuals.

K. HARDWARE REQUIREMENTS:

Processor	:	Dual Core Processor
Hard Disk	:	1TB
Ram	:	2GB
Speed	:	1.3GHz

L. SOFTWARE REQUIREMENTS:

Programming Language	:	Python, R
Operating System	:	Windows, Linux
Python Version	:	Standard Versions of Python

4. EXPERIMENTAL ENVIRONMENT

M. DATA SET DESCRIPTION:

OSMI is a non-profit corporation dedicated to "raising awareness, educating, and providing resources to support mental wellness in the tech and open source communities." What they do in support of this goal includes providing e-books on mental wellness in the workplace, hosting a forum on conversations on mental health, and holding talks at developer conferences about mental health in the community.

In one of their efforts, OSMI provides a survey on mental health in tech industry. This survey contains a variety of questions pertaining to the mental health of the respondents, the demographics of the respondents, and how employer views on mental health in the workplace. This survey was conducted in 2014 and 2016. For today, we will be using the later year's dataset.

In 2016, the survey was distributed via twitter and through talks given at conferences. Since it was an opt-in survey, there is likely some selection bias in the dataset. In particular, it is likely that those who are more interested in mental health would be more likely to participate in this survey. While this dataset is still analyzable, it is important to note this when considering our results later on.

OSMI representatives listed some of their questions on the survey in the data.world forums. Mosts notably, OSMI is interested in how certain demographic and work-life components of respondents impact the rate of mental health conditions in the industry. Some of these variables include:

Age of the respondent

Gender the respondent identifies with

Location and region of work

Type of work (e.g. Front-end development, Design, Marketing)

If the respondent is self-employed

We will consider these variables and a few more in our analysis.

5. IMPLEMENTATION

The proccess is the following:

1.Library and data loading

2.Data cleaning

3.Encoding data

4.Covariance Matrix. Variability comparison between categories of variables

5.Some charts to see data relationship

6.Scaling and fitting

7.Tuning

8.Evaluating models

- A. Logistic Regression
- B. K Neighbors Classifier
- C. Decision Tree Classifier
- D. Random Forests
- E. Bagging
- F. Boosting
- G. Stacking

9.Predicting with Neural Network

10.Success method plot

11.Creating predictions on test set

1. Library and data loading:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
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 binarize, 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.grid_search import RandomizedSearchCV
#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
raw_df = pd.read_csv('D:\survey.csv')
raw_df.head()
```

	Timestamp	Age	Gender	Country	state	self_employed	family_history	treatment	work_interfere	no_employees	...	leave	mental_health_consequenc
0	8/27/2014 11:29	37	Female	United States	IL	NaN	No	Yes	Often	25-Jun	...	Somewhat easy	N
1	8/27/2014 11:29	44	M	United States	IN	NaN	No	No	Rarely	More than 1000	...	Don't know	Mayb
2	8/27/2014 11:29	32	Male	Canada	NaN	NaN	No	No	Rarely	25-Jun	...	Somewhat difficult	N
3	8/27/2014 11:29	31	Male	United Kingdom	NaN	NaN	Yes	Yes	Often	26-100	...	Somewhat difficult	Ye
4	8/27/2014 11:30	31	Male	United States	TX	NaN	No	No	Never	100-500	...	Don't know	N

5 rows × 27 columns

print(raw_df.shape)

#Pandas: whats the distribution of the data?

print(raw_df.describe())

#Pandas: What types of data do i have?

print(raw_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):
Timestamp                  1259 non-null object
Age                        1259 non-null int64
Gender                     1259 non-null object
Country                    1259 non-null object
state                      744 non-null object
self_employed              1241 non-null object
family_history             1259 non-null object
treatment                  1259 non-null object
work_interfere             995 non-null object
no_employees               1259 non-null object
remote_work                1259 non-null object
tech_company               1259 non-null object
benefits                   1259 non-null object
care_options               1259 non-null object
wellness_program           1259 non-null object
seek_help                  1259 non-null object
anonymity                  1259 non-null object
leave                      1259 non-null object
```

2. DATA CLEANING:

#dealing with missing data

#Let's get rid of the variables "Timestamp",“comments”, “state” just to make our lives easier.

raw_df2 = raw_df.drop(['comments'], axis=1)

raw_df2 = raw_df2.drop(['state'], axis=1)

raw_df2 = raw_df2.drop(['Timestamp'], axis=1)

raw_df2.isnull().sum().max() #just checking that there's no missing data missing...

raw_df2.head(5)

	Age	Gender	Country	self_employed	family_history	treatment	work_interfere	no_employees	remote_work	tech_company	...	anonymity	leave	me
0	37	Female	United States	NaN	No	Yes	Often	25-Jun	No	Yes	...	Yes	Somewhat easy	
1	44	M	United States	NaN	No	No	Rarely	More than 1000	No	No	...	Don't know	Don't know	
2	32	Male	Canada	NaN	No	No	Rarely	25-Jun	No	Yes	...	Don't know	Somewhat difficult	
3	31	Male	United Kingdom	NaN	Yes	Yes	Often	26-100	No	Yes	...	No	Somewhat difficult	
4	31	Male	United States	NaN	No	No	Never	100-500	Yes	Yes	...	Don't know	Don't know	

5 rows × 24 columns

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 raw_df2:

    if feature in intFeatures:

        raw_df2[feature] = raw_df2[feature].fillna(defaultInt)

    elif feature in stringFeatures:

        raw_df2[feature] = raw_df2[feature].fillna(defaultString)

    elif feature in floatFeatures:

        raw_df2[feature] = raw_df2[feature].fillna(defaultFloat)

    else:

        print('Error: Feature %s not recognized.' % feature)

raw_df2.head(5)
```

	Age	Gender	Country	self_employed	family_history	treatment	work_interfere	no_employees	remote_work	tech_company	...	anonymity	leave	me
0	37	Female	United States	NaN	No	Yes	Often	25-Jun	No	Yes	...	Yes	Somewhat easy	
1	44	M	United States	NaN	No	No	Rarely	More than 1000	No	No	...	Don't know	Don't know	
2	32	Male	Canada	NaN	No	No	Rarely	25-Jun	No	Yes	...	Don't know	Somewhat difficult	
3	31	Male	United Kingdom	NaN	Yes	Yes	Often	26-100	No	Yes	...	No	Somewhat difficult	
4	31	Male	United States	NaN	No	No	Never	100-500	Yes	Yes	...	Don't know	Don't know	

5 rows × 24 columns

```
#clean 'Gender'

#lower case all columm's elements

gender = raw_df2['Gender'].str.lower()

#print(gender)

#Select unique elements

gender = raw_df2['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 raw_df2.iterrows():

    if str.lower(col.Gender) in male_str:

        raw_df2['Gender'].replace(to_replace=col.Gender, value='male', inplace=True)

    if str.lower(col.Gender) in female_str:

        raw_df2['Gender'].replace(to_replace=col.Gender, value='female', inplace=True)

    if str.lower(col.Gender) in trans_str:

        raw_df2['Gender'].replace(to_replace=col.Gender, value='trans', inplace=True)

#Get rid of bullshit

stk_list = ['A little about you', 'p']

raw_df2 = raw_df2[~raw_df2['Gender'].isin(stk_list)]

print(raw_df2['Gender'].unique())

output: ['female' 'male' 'trans']
```

```
#complete missing age with mean

raw_df2['Age'].fillna(raw_df2['Age'].median(), inplace = True)

# Fill with media() values < 18 and > 120

s = pd.Series(raw_df2['Age'])

s[s<18] = raw_df2['Age'].median()

raw_df2['Age'] = s

s = pd.Series(raw_df2['Age'])

s[s>120] = raw_df2['Age'].median()

raw_df2['Age'] = s

#Ranges of Age

raw_df2['age_range'] = pd.cut(raw_df2['Age'], [0,20,30,65,100], labels=["0-20", "21-30", "31-65", "66-100"], include_lowest=True)

raw_df2
```

Out[23]:

	Age	Gender	Country	self_employed	family_history	treatment	work_interfere	no_employees	remote_work	tech_company	...	leave	mental_
0	37	female	United States	NaN	No	Yes	Often	25-Jun	No	Yes	...	Somewhat easy	
1	44	male	United States	NaN	No	No	Rarely	More than 1000	No	No	...	Don't know	
2	32	male	Canada	NaN	No	No	Rarely	25-Jun	No	Yes	...	Somewhat difficult	
3	31	male	United Kingdom	NaN	Yes	Yes	Often	26-100	No	Yes	...	Somewhat difficult	
4	31	male	United States	NaN	No	No	Never	100-500	Yes	Yes	...	Don't know	
5	33	male	United States	NaN	Yes	No	Sometimes	25-Jun	No	Yes	...	Don't know	

```
#There are only 0.014% of self employed so let's change NaN to NOT self_employed

#Replace "NaN" string from defaultString

raw_df2['self_employed'] = raw_df2['self_employed'].replace([defaultString], 'No')

print(raw_df2['self_employed'].unique())

output: ['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

raw_df2['work_interfere'] = raw_df2['work_interfere'].replace([defaultString], 'Don\'t know' )

print(raw_df2['work_interfere'].unique())

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

3.ENCODING DATA:

```
#Encoding data

labelDict = {}

for feature in raw_df2:

    le = preprocessing.LabelEncoder()

    le.fit(raw_df2[feature])

    le_name_mapping = dict(zip(le.classes_, le.transform(le.classes_)))

    raw_df2[feature] = le.transform(raw_df2[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'
raw_df2 = raw_df2.drop(['Country'],axis=1)
raw_df2.head()
```

```
label_Age [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 3
1, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44]
label_Gender [0, 1, 2]
label_self_employed [0, 1]
label_family_history [0, 1]
label_treatment [0, 1]
label_work_interfere [0, 1, 2, 3, 4]
label_no_employees [0, 1, 2, 3, 4, 5]
label_remote_work [0, 1]
label_tech_company [0, 1]
label_benefits [0, 1, 2]
label_care_options [0, 1, 2]
label_wellness_program [0, 1, 2]
label_seek_help [0, 1, 2]
label_anonymity [0, 1, 2]
label_leave [0, 1, 2, 3, 4]
label_mental_health_consequence [0, 1, 2]
label_phys_health_consequence [0, 1, 2]
label_coworkers [0, 1, 2]
label_supervisor [0, 1, 2]
label_mental_health_interview [0, 1, 2]
label_phys_health_interview [0, 1, 2]
label_mental_vs_physical [0, 1, 2]
label_obs_consequence [0, 1]
label_age_range [0, 1, 2, 3]
```

```
#missing data
total = raw_df2.isnull().sum().sort_values(ascending=False)
percent = (raw_df2.isnull().sum()/raw_df2.isnull().count()).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data.head(20)
print(missing_data)
```

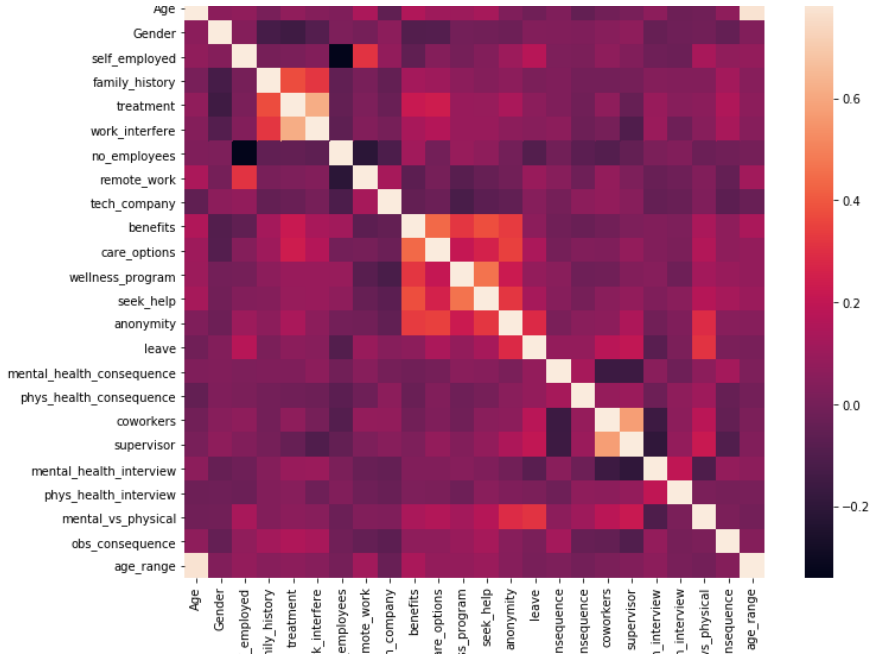
	Total	Percent
age_range	0	0.0
obs_consequence	0	0.0
Gender	0	0.0
self_employed	0	0.0
family_history	0	0.0
treatment	0	0.0
work_interfere	0	0.0
no_employees	0	0.0
remote_work	0	0.0
tech_company	0	0.0
benefits	0	0.0
care_options	0	0.0
wellness_program	0	0.0
seek_help	0	0.0
anonymity	0	0.0
leave	0	0.0
mental_health_consequence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

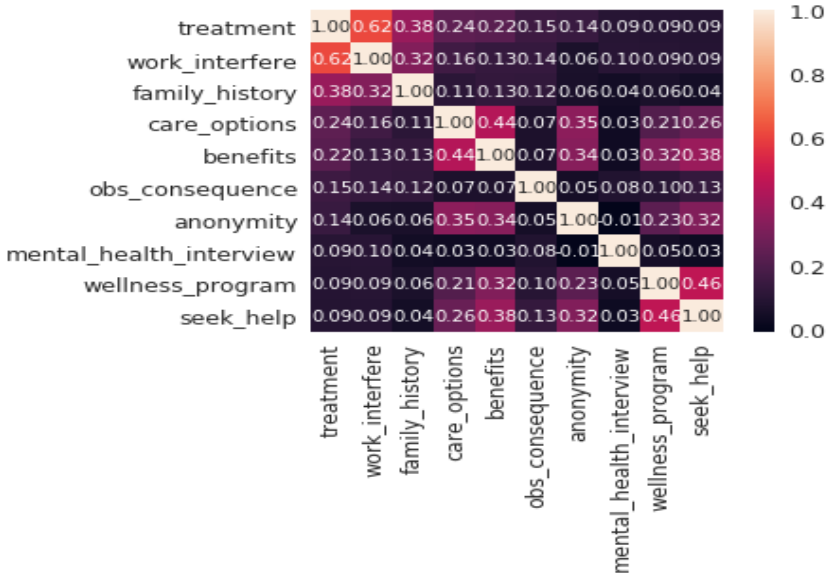
4. Covariance Matrix. Variability comparison between categories of variables:

```
#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()

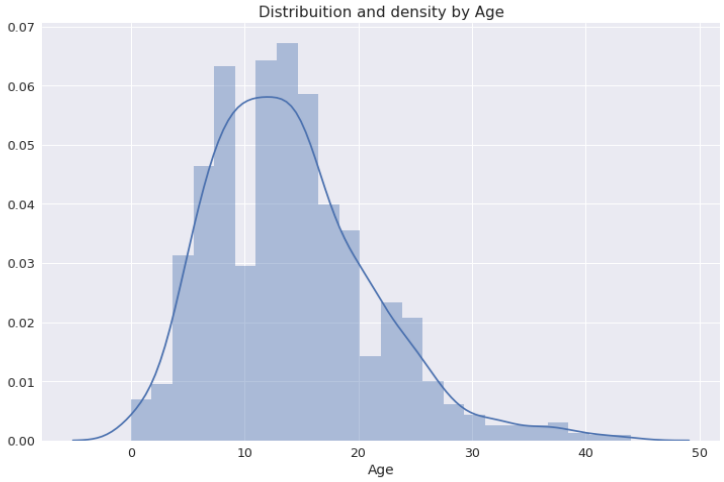




5. Some charts to see data relationship:

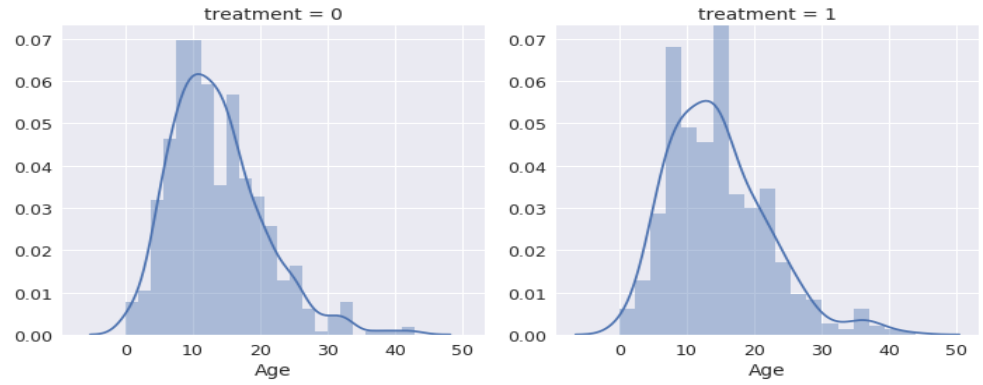
Distribiution and density by Age

```
plt.figure(figsize=(12,8))

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

plt.title("Distribuition and density by Age")

plt.xlabel("Age")
```



Separate by treatment

```
g = sns.FacetGrid(train_df, col='treatment', size=5)

g = g.map(sns.distplot, "Age")
```



Let see how many people has been treated

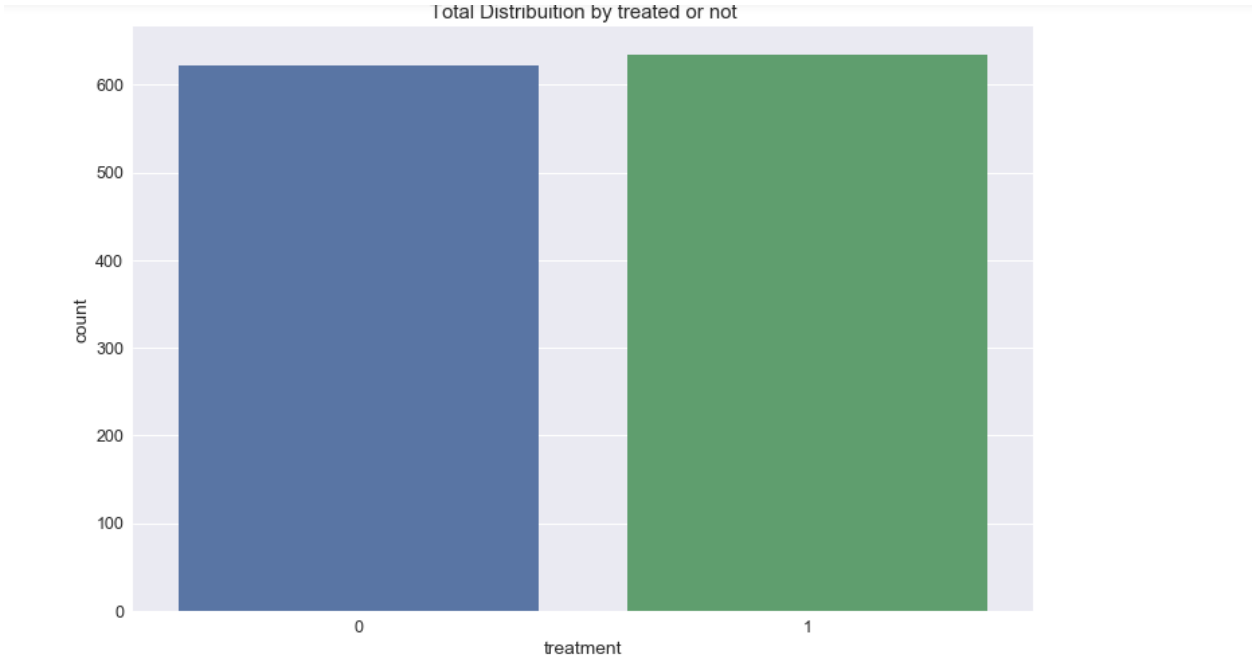
plt.figure(figsize=(12,8))

labels = labelDict['label_Gender']

g = sns.countplot(x="treatment", data=raw_df2)

g.set_xticklabels(labels)

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



o = labelDict['label_age_range']

g = sns.factorplot(x="age_range", y="treatment", hue="Gender", data=raw_df2, kind="bar", ci=None, size=5, aspect=2, legend_out = True)

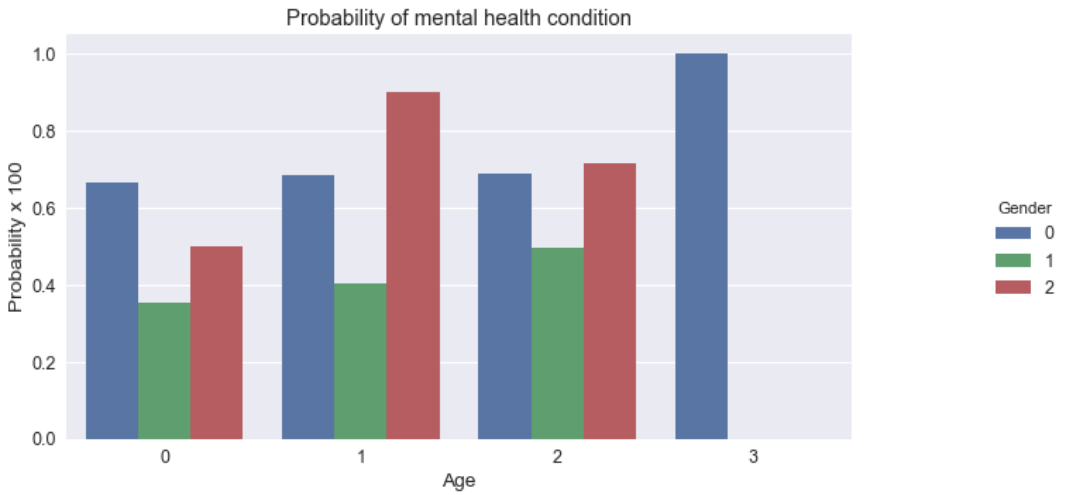
g.set_xticklabels(o)

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()
```

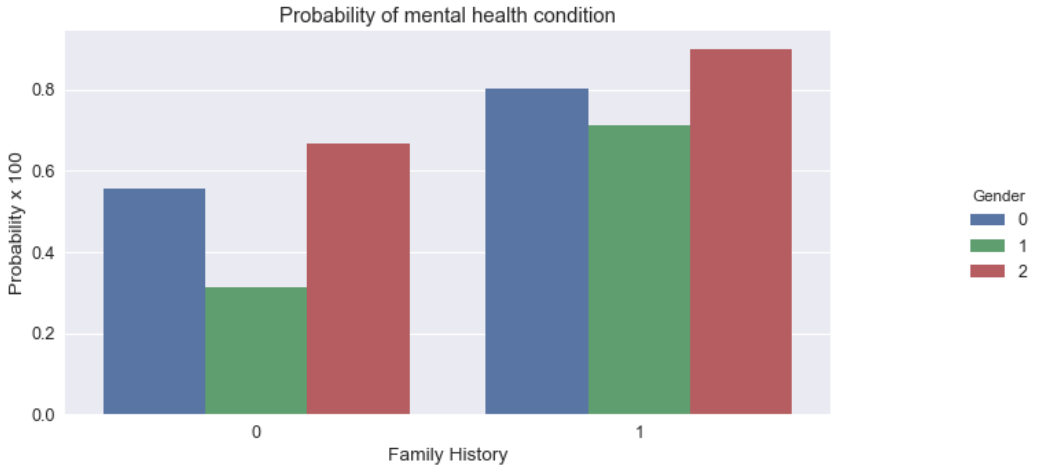



```
o = labelDict['label_family_history']

g = sns.factorplot(x="family_history", y="treatment", hue="Gender", data=raw_df2, kind="bar",
ci=None, size=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()



o = labelDict['label_care_options']

g = sns.factorplot(x="care_options", y="treatment", hue="Gender", data=raw_df2, kind="bar", ci=None, size=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

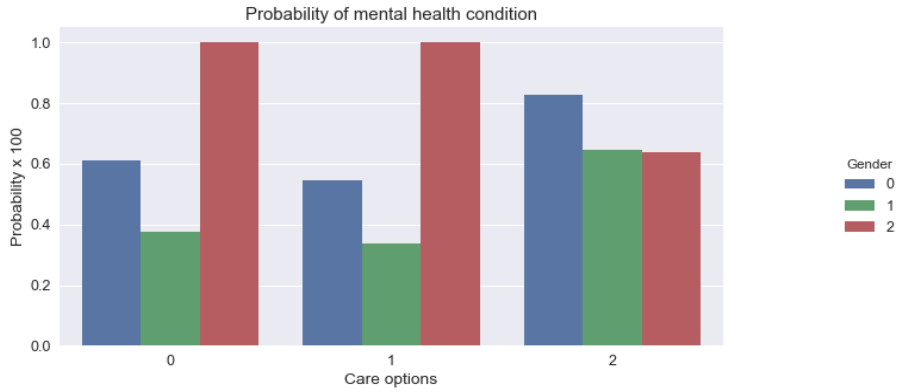
new_labels = labelDict['label_Gender']

for t, l in zip(g._legend.texts, new_labels): t.set_text(l)

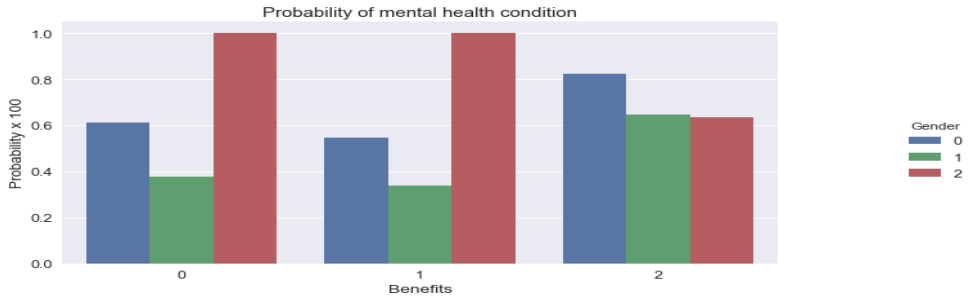
Positioning the legend

g.fig.subplots_adjust(top=0.9,right=0.8)

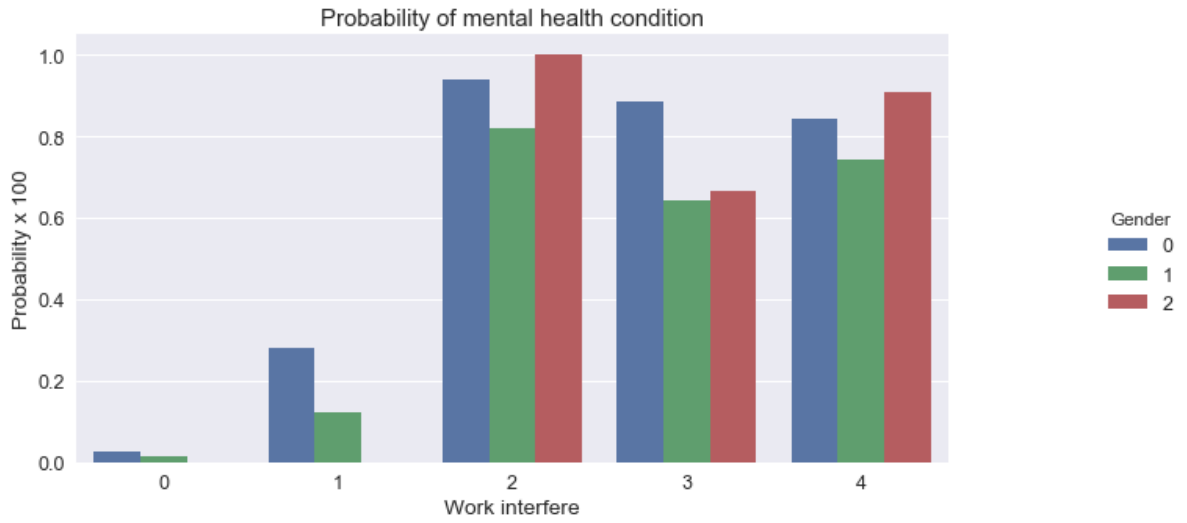
plt.show()




```
o = labelDict['label_benefits']

g = sns.factorplot(x="care_options", y="treatment", hue="Gender", data=raw_df2, kind="bar", ci=None, size=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()
```



```
o = labelDict['label_work_interfere']

g = sns.factorplot(x="work_interfere", y="treatment", hue="Gender", data=raw_df2, kind="bar", ci=None, size=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()
```



6. Scaling and fitting:

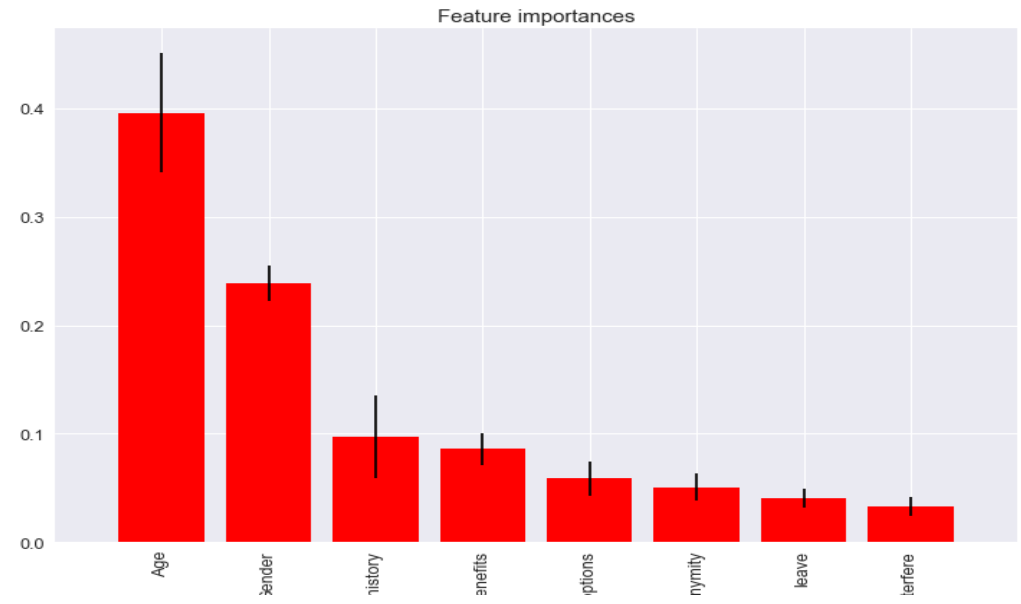
```
# Scaling Age

scaler = MinMaxScaler()

raw_df2['Age'] = scaler.fit_transform(raw_df2[['Age']])

raw_df2.head()
```

	Age	Gender	self_employed	family_history	treatment	work_interfere	no_employees	remote_work	tech_company	benefits	...	leave	mental_health_
0	0.431818	0	0	0	1	2	1	0	1	2	...	2	
1	0.590909	1	0	0	0	3	5	0	0	0	...	0	
2	0.318182	1	0	0	0	3	1	0	1	1	...	1	
3	0.295455	1	0	1	1	2	2	0	1	1	...	1	
4	0.295455	1	0	0	0	1	0	1	1	2	...	0	

5 rows × 24 columns

```
# define X and y

feature_cols = ['Age', 'Gender', 'family_history', 'benefits', 'care_options', 'anonymity', 'leave',
'work_interfere']

X = raw_df2[feature_cols]

y = raw_df2.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:

```
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)
y_pred_class = binarize(y_pred_prob, 0.3)[0]
# 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)
# 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

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()
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_)

def tuningRandomizedSearchCV(model, param_dist):

#Searching multiple parameters simultaneously

# n_iter controls the number of searches

rand = RandomizedSearchCV(model, param_dist, cv=10, scoring='accuracy', n_iter=10,
random_state=5)

rand.fit(X, y)

rand.grid_scores_

# examine the best model

print('Rand. Best Score: ', rand.best_score_)

print('Rand. Best Params: ', rand.best_params_)

# run RandomizedSearchCV 20 times (with n_iter=10) and record the best score

best_scores = []
```

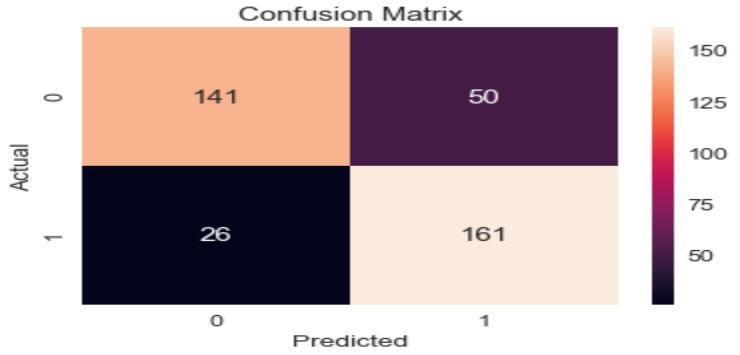
```
for _ in range(20):
rand = RandomizedSearchCV(model, param_dist, cv=10, scoring='accuracy', n_iter=10)
rand.fit(X, y)
best_scores.append(round(rand.best_score_, 3))
print(best_scores)
```

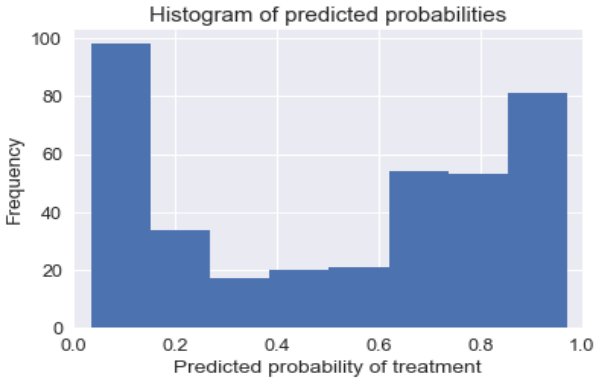
8.EVALUATING MODELS:

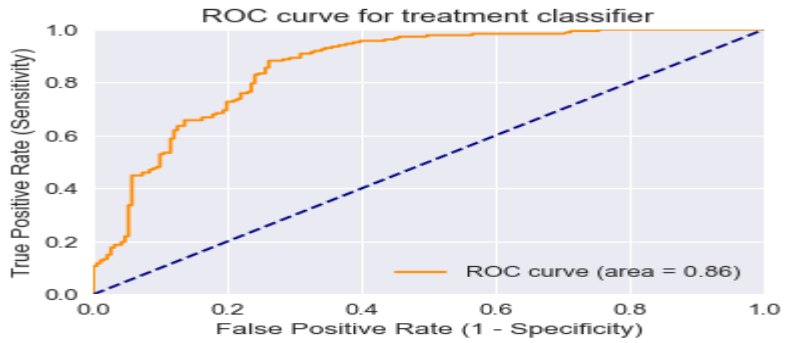
A.LOGISTIC REGRESSION:

```
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: 0.798941798941799
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]
```





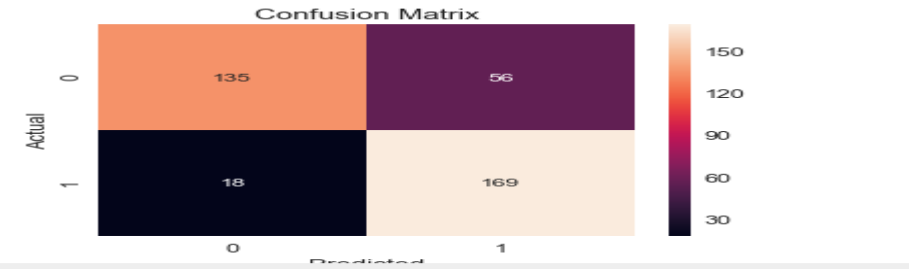


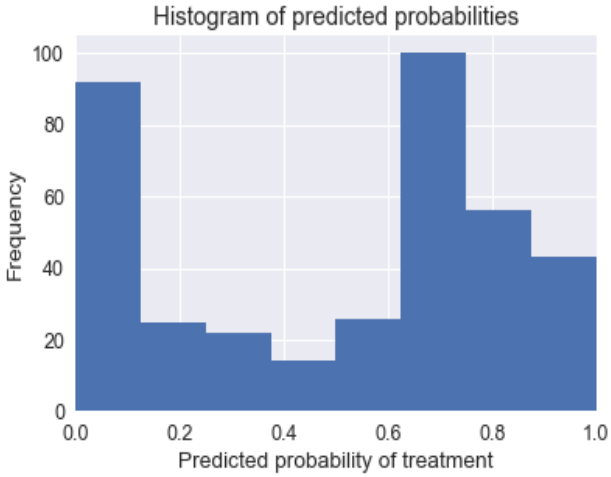

```
[[141  50]
 [ 26 161]]
```

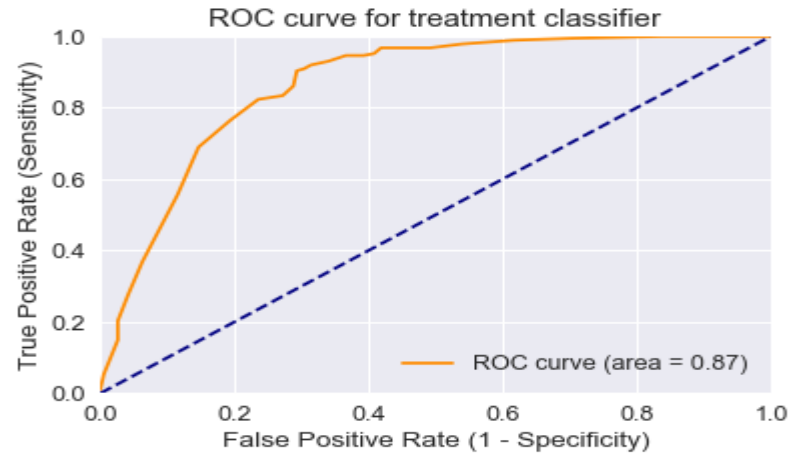
B. K NEAREST NEIGHBOUR:

```
def Knn():
# Calculating the best parameters
knn = KNeighborsClassifier(n_neighbors=5)
#tuningCV(knn)
#tuningGridSerach(knn)
#tuningMultParam(knn)
# 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
knn()
```

```
############ 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]
```





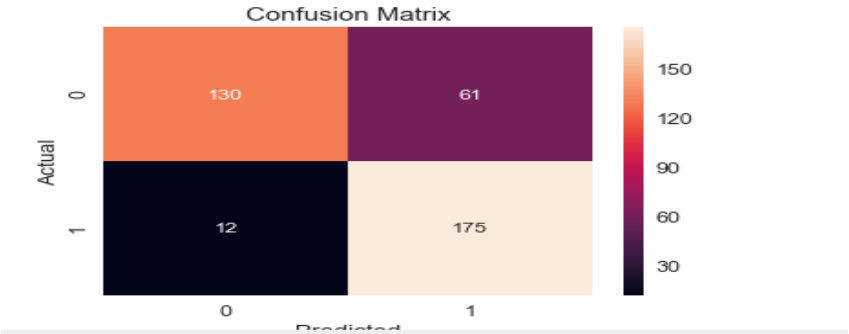


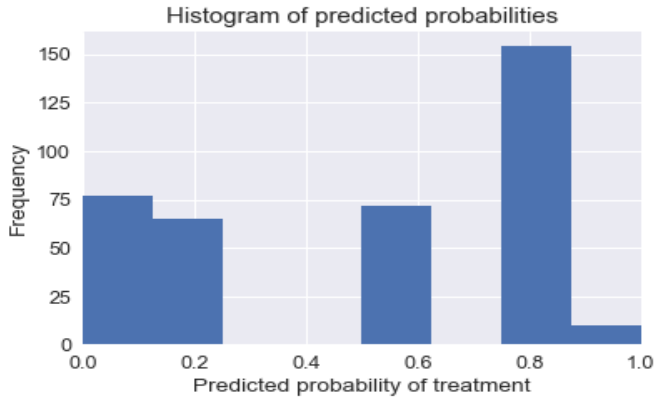

```
[[135  56]
 [ 18 169]]
```

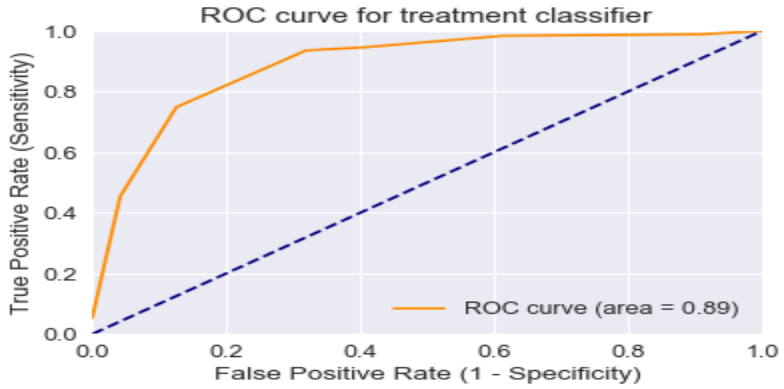
C. DECISION TREE CLASSIFIER:

```
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)

# 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()
```

```
############ Tree classifier ###############
Accuracy: 0.8068783068783069
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]
```





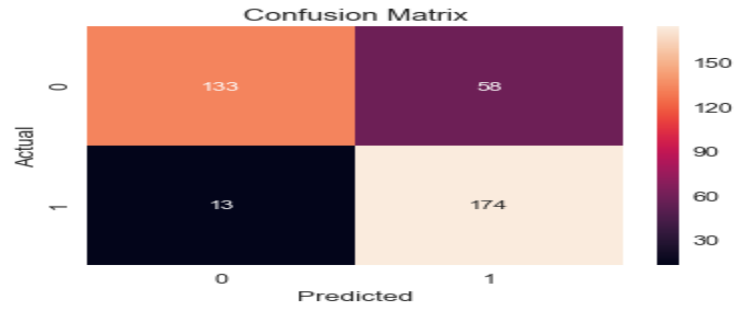


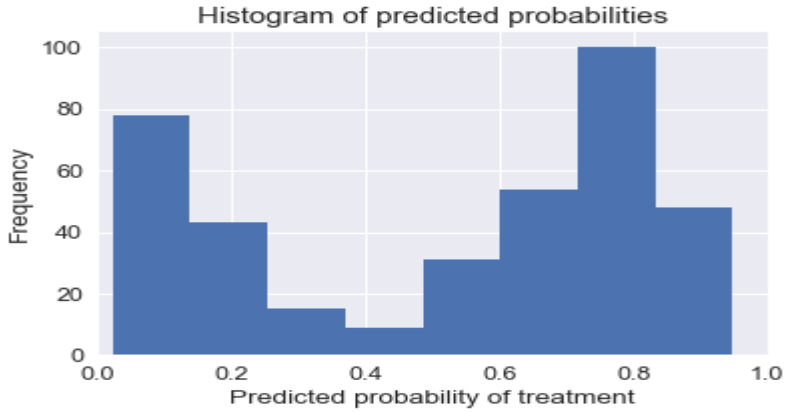

```
[[130  61]
 [ 12 175]]
```

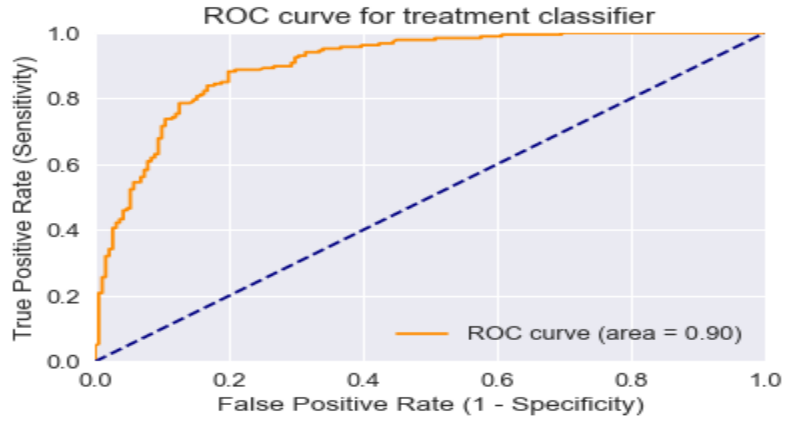
D. RANDOM FOREST:

```
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()
```

```
############ 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]
```





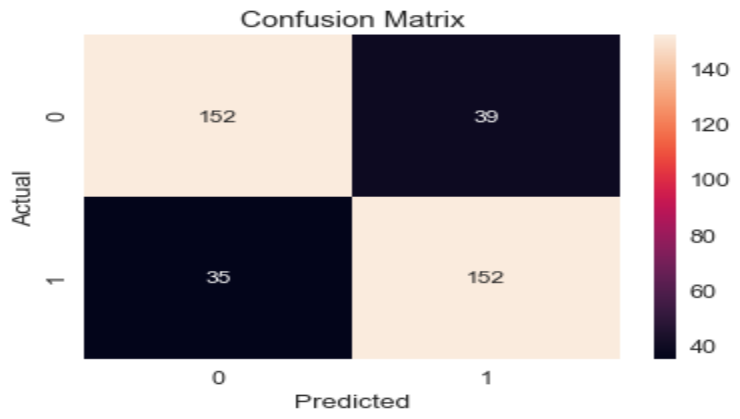


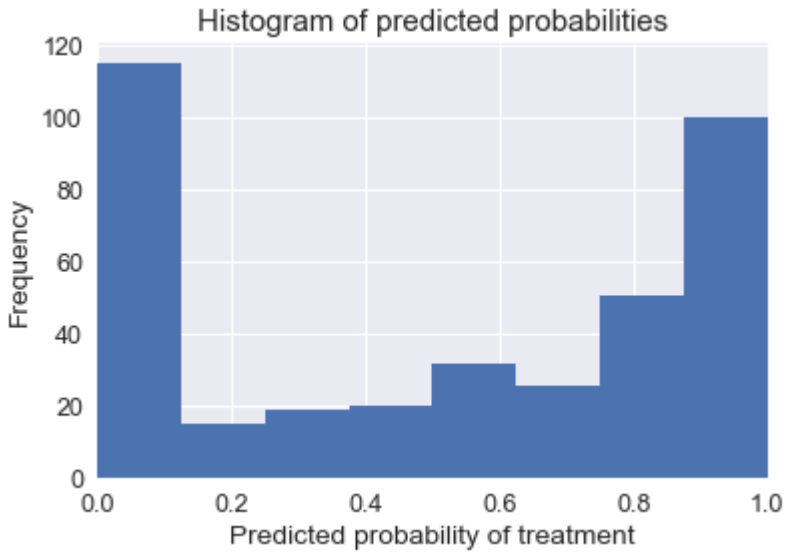

```
[[133  58]
 [ 13 174]]
```

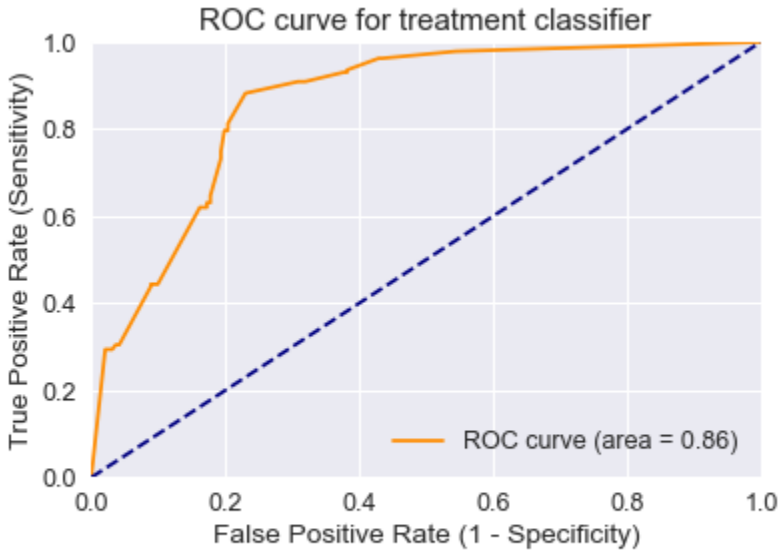
E. BAGGING:

```
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: 0.8042328042328042
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: [0 0 0 0 0 1 0 0 1 1 0 1 1 0 1 0 0 1 0 0 0 0 1 0 0]
```




Histogram of predicted probabilities
Frequency
Predicted probability of treatment
0
20
40
60
80
100
120
0.0
0.2
0.4
0.6
0.8
1.0

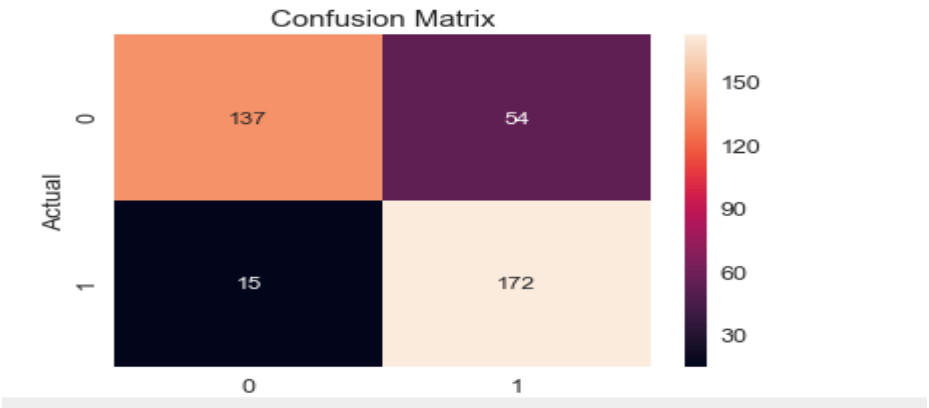

ROC curve for treatment classifier
True Positive Rate (Sensitivity)
False Positive Rate (1 - Specificity)
ROC curve (area = 0.86)
0.0
0.2
0.4
0.6
0.8
1.0

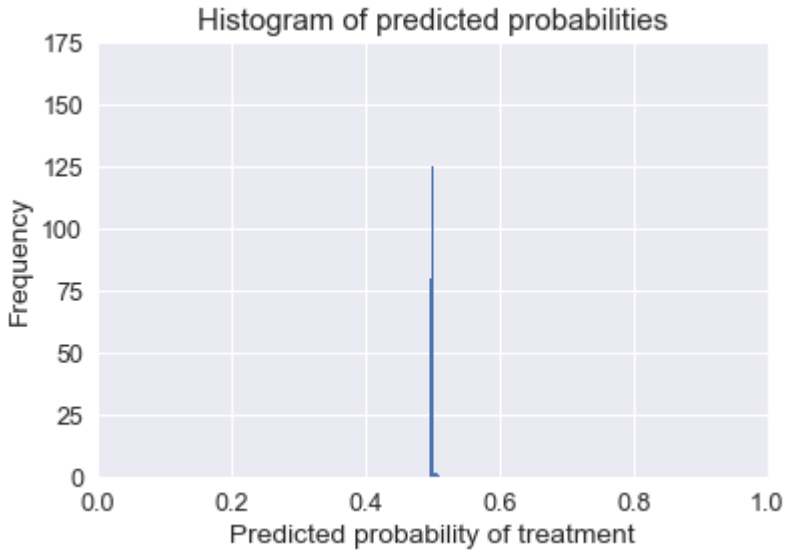
```
[[152  39]
 [ 35 152]]
```

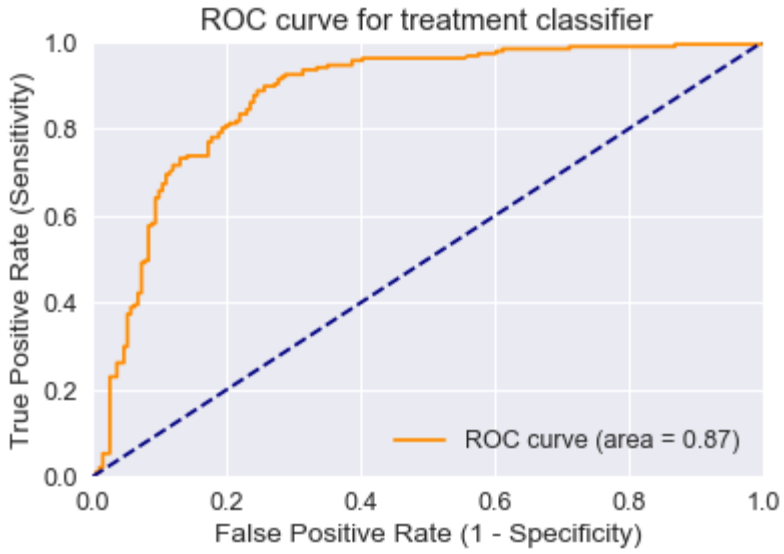
F. BOOSTING:

```
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()
```

```
########### Boosting ################
Accuracy: 0.8174603174603174
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 0 1 0 1 1 1 0 1 1 0 1 1 1 1 0 0 0 0 1 0 0]
```




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


ROC curve for treatment classifier
True Positive Rate (Sensitivity)
1.0
0.8
0.6
0.4
0.2
0.0
ROC curve (area = 0.87)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate (1 - Specificity)

G. 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 ###############')
accuracy_score = evalClassModel(stack, y_test, y_pred_class, True)
#Data for final graph
methodDict['Stacking'] = accuracy_score * 100
```

9. Predicting with Neural Network

```
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
stacking()
# 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]

model = tf.estimator.DNNClassifier(feature_columns=feature_columns,

                                   hidden_units=[10, 10],

                                   optimizer=tf.train.ProximalAdagradOptimizer(

                                     learning_rate=0.1,

                                     l1_regularization_strength=0.001

                                   ))

# 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

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

# Generate predictions from the model

template = ('\nIndex: "{}", Prediction is "{}" ({:.1f}%), expected "{}"')

# Dictionary for predictions

col1 = []
```

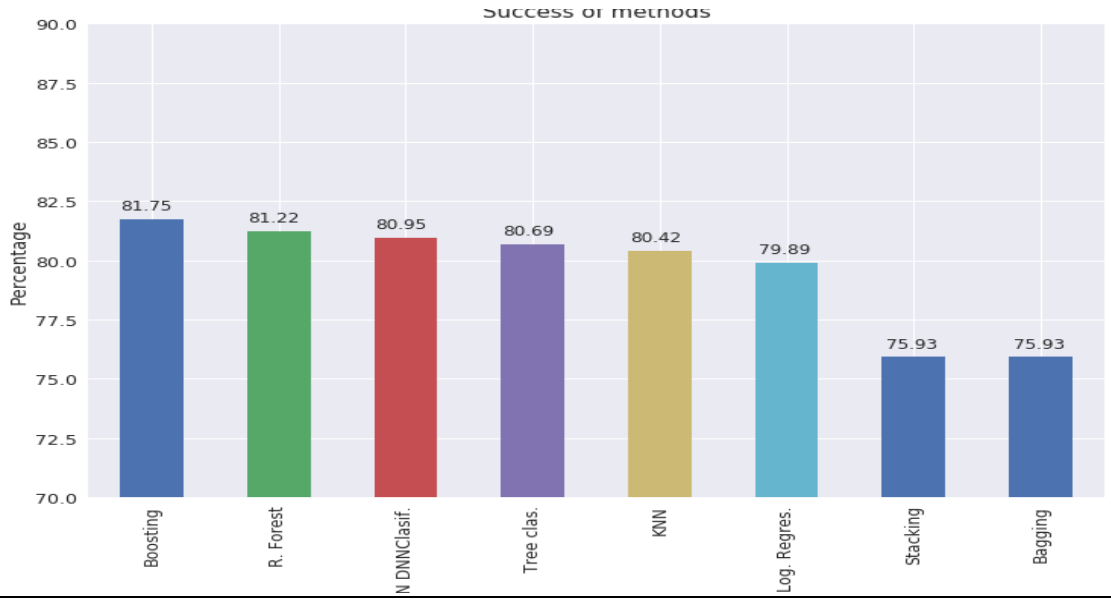
```
col3 = []
for idx, input, p in zip(X_train.index, y_train, predictions):
v  = p["class_ids"][0]
class_id = p['class_ids'][0]
probability = p['probabilities'][class_id] # Probability
# Adding to dataframe
col1.append(idx) # Index
col2.append(v) # Prediction
col3.append(input) # Expecter
#print(template.format(idx, v, 100 * probability, input))
```

Out[48]:

	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 METHODS PLOT:

```
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 DATA:

```
# 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)

results.head()
```

	Index	Treatment
0	5	1
1	494	0
2	52	0
3	984	0
4	186	0

6.CONCLUSION AND FUTURE SCOPE

Thankfully, there are plenty of avenues of future analysis related to this dataset: The rest of the survey primarily deals with questions related to how workplaces, companies, and the industry as a whole deal with mental health. If we perform a cluster analysis on these responses, we may find a profile of different perspectives on how the industry is dealing with mental health.

Because our model is not very accurate, it would be useful to consider measurements of workplace stress and culture in a further survey. These may help to inform our variable of medical diagnosis.

As discussed before, being medically diagnosed with a mental health disorder is not the only possible target variable to consider. The two other measurements could present meaningful analyses if we estimated the effects of demographic and employment variables on these targets.

Since this survey was also taken in 2014, it may be useful to do a yearly analysis and see if our results change over the two time period.

7. REFERENCES

[1] World Health Organization (2011a). Political Declaration of the High-level Meeting of the General Assembly on the Prevention and Control of Non-Communicable Diseases. 66th Session of the Unites Nations General Assembly. New York: WHO.

[2] Gordon, A. (2013). Mental Health Remains an Invisible Problem in Africa. Think Africa Press. Retrieved from http://thinkafricapress.com on May 12, 2017.

[3] Arboleda-Florez, J. (2002). What Causes Stigma. World Psychiatry 1 (1): 25–26.

[4] Corrigan, P. W., Druss, B. G. and Perlick, D. A. 2014. The Impact of Mental Illness Stigma on Seeking and Participating in Mental Health Care. Psychological Science in the Public Interest, 15 (2) 37–70: sagepub.com/journalsPermissions.nav.

[5] Fournier, O. A. (2011). The Status of Mental Health Care in Ghana, West Africa and Signs and Progress in the Greater Accra Region. Berkeley Undergraduate Journal 24 (3)

[6] Naifeh, J. A., Colpe, C. L. J., Aliaga, P. A., Sampson, N. A., Heeringa, S. G., Stein, M. B., Ursano, R. J., Fullerton, C. S., Nock, M. K., Schoenbaum, M. and Zaslavsky, A. M., 2016. Barriers to initiating and continuing mental health treatment among soldiers in the Army Study to Assess Risk and Resilience in Servicemembers (Army STARRS). Military medicine, 181 (9), p.1021.

[7] Hanlon, C., Wondimagegn, D. and Alem, A. (2010). Lessons Learned in Developing Community Mental Health Care in Africa. World Psychiatry 9 (3): 185–189.

[8] World Health Organization (2011b). WHO African Regional Ministerial Consultation on Non-communicable Diseases. Brazzaville, Congo. Brazzaville: WHO Regional Office for Africa.

[9] World Health Organization (2015). Mental health atlas 2014. World Health Organization, Reterived from http://apps.who.int/iris/bitstream/10665/178879/1/978924156 5011_eng.pdf. Accessed on March 20th, 2016.

[10] Njenga, F. (2002). 'Focus on Psychiatry in East Africa'. British Journal of Psychiatry (181): 354-59.