**JAYPEE INSTITUTE OF INFORMATION AND TECHNOLOGY**



OPEN SOURCE SOFTWARE

# PROJECT REPORT

TOPIC:-Mental Health Data Analysis Using PYTHON

GROUP DETAILS:

|  |  |  |
| --- | --- | --- |
| **NAME** | ENROLLMENT NO. | BATCH |
| PRAKHAR JAIN | 20103303 | B10 |
| CHIRAG SHARMA | 20103302 | B10 |
| SANSKAR GOEL | 20103298 | B10 |
| Yashika Kushwaha | 20103280 | B10 |

FACULTY- JANARDAN KUMAR VERMA

# Abstract

The increase of mental health problems and the need for effective medical health care have led to an investigation of machine learning that can be applied in mental health problems. This paper presents a recent systematic review of machine learning approaches in predicting mental health problems. Furthermore, we will discuss the challenges, limitations, and future directions for the application of machine learning in the mental health field. We collect research articles and studies that are related to the machine learning approaches in predicting mental health problems by searching reliable databases. Moreover, we adhere to the PRISMA methodology in conducting this systematic review. We include a total of 30 research articles in this review after the screening and identification processes. Then, we categorize the collected research articles based on the mental health problems such as schizophrenia, bipolar disorder, anxiety and depression, posttraumatic stress disorder, and mental health problems among children. Discussing the findings, we reflect on the challenges and limitations faced by the researchers on machine learning in mental health problems. Additionally, we provide concrete recommendations on the potential future research and development of applying machine learning in the mental health field.

# Introduction

Mental illness is a health problem that undoubtedly impacts emotions, reasoning, and social interaction of a person. These issues have shown that mental illness gives serious consequences across societies and demands new strategies for prevention and intervention. To accomplish these strategies, early detection of mental health is an essential procedure. Medical predictive analytics will reform the healthcare field broadly as discussed by Miner et al. [1]. Mental illness is usually diagnosed based on the individual self-report that requires questionnaires designed for the detection of the specific patterns of feeling or social interactions [2]. With proper care and treatment, many individuals will hopefully be able to recover from mental illness or emotional disorder

Machine learning is a technique that aims to construct systems that can improve through experience by using advanced statistical and probabilistic techniques. It is believed to be a significantly useful tool to help in predicting mental health. It is allowing many researchers to acquire important information from the data, provide personalized experiences, and develop automated intelligent systems [4]. The widely used algorithms in the field of machine learning such as support vector machine, random forest, and artificial neural networks have been utilized to forecast and categorize the future events

# METHODOLOGY

1. About the dataset
2. Load essential Python Libraries
3. Load Training/Test datasets
4. Data Preprocessing
5. Exploratory data analysis (EDA).
6. Feature Engineering.
7. Build Machine Learning Model
8. Make predictions on the test dataset

# OBJECTIVE

This review aimed to provide a concise snapshot of the research to date investigating ML applications to mental health. Previous reviews have demonstrated ML techniques to be robust and scalable for mental health application, but no review has comprehensively mapped the clinical applications within mental health research and practice. Such a review would equip both data scientists and practitioners in the methods and applications of big data. It would also highlight the challenges of using ML techniques in this context, as well as identify gaps in the field and potential opportunities for further research. First, we outline the search strategies used to find relevant literature. Next, we conduct a synthesis of the literature, describing both the ML techniques and mental health applications of each article. Finally, the paper summarises the extant research and the implications for future work.

CONCLUSION :

Above is the report of the project with data preprocessing , training and test data , feature engineering etc .We have used logistic regression,k neighbour,dicission tree classification-based algorithm .accuracy of all algos comes out to be 79.63%,80.42%,80.69%

CODE AND OUTPUT:

In [182… **import** numpy **as** np **import** pandas **as** pd **import** matplotlib.pyplot **as** plt **import** seaborn **as** sns

In [183… **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 |

In [184…

|  |
| --- |
| *# models*  **from** sklearn.linear\_model **import** LogisticRegression **from** sklearn.tree **import** DecisionTreeClassifier  **from** sklearn.ensemble **import** RandomForestClassifier, ExtraTreesClassifier **from** sklearn.model\_selection **import** RandomizedSearchCV |

In [185…

|  |
| --- |
| *# Validation libraries* **from** sklearn **import** metrics  **from** sklearn.metrics **import** accuracy\_score, mean\_squared\_error, precision\_recall\_cu **from** sklearn.model\_selection **import** cross\_val\_score |

In [186…

|  |  |  |
| --- | --- | --- |
| *#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 |  |  |
|  |  |  |
| train\_df **=** pd**.**read\_csv('survey.csv') train\_df**.**head() |  |  |
| **Timestamp Age Gender Country state self\_employed** | **family\_history** | **treatment work\_inte** |

In [187…

In [188…

In [189…

In [190…

Out[190]:

2014-08-27 United

**0** 37 Female IL NaN No Yes

11:29:31 States

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | 2014-08-27 11:29:37 | 44 | M | United  States | IN | NaN | No | No |
| **2** | 2014-08-27 11:29:44 | 32 | Male | Canada | NaN | NaN | No | No |
| **3** | 2014-08-27 11:29:46 | 31 | Male | United  Kingdom | NaN | NaN | Yes | Yes |
| **4** | 2014-08-27 11:30:22 | 31 | Male | United  States | TX | NaN | No | No |

5 rows × 27 columns

|  |
| --- |
| *#missing data* |

In [191…

total **=** train\_df**.**isnull()**.**sum()**.**sort\_values(ascending**=False**)

percent **=** (train\_df**.**isnull()**.**sum()**/**train\_df**.**isnull()**.**count())**.**sort\_values(ascending 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

In [192… *#dealing with missing data* train\_df**.**drop(['comments'], axis**=** 1, inplace**=True**) train\_df**.**drop(['state'], axis**=** 1, inplace**=True**) train\_df**.**drop(['Timestamp'], axis**=** 1, inplace**=True**) train\_df**.**drop(['Country'], axis**=** 1, inplace**=True**)

|  |
| --- |
| *# Assign default values for each data type* defaultInt **=** 0 defaultString **=** 'NaN' defaultFloat **=** 0.0    *# Create lists by data tpe* intFeatures **=** ['Age']  stringFeatures **=** ['Gender', 'self\_employed', 'family\_history', 'treatment', 'work\_i 'no\_employees', 'remote\_work', 'tech\_company', 'anonymity', 'leave  'phys\_health\_consequence', 'coworkers', 'supervisor', 'mental\_heal 'mental\_vs\_physical', 'obs\_consequence', 'benefits', 'care\_options  'seek\_help'] 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**: |

In [193…

|  |  |  |
| --- | --- | --- |
| print('Error: Feature %s not recognized.' **%** feature) train\_df**.**head() |  |  |
| **Age Gender self\_employed family\_history treatment work\_interfere** | **no\_employees** | **remote** |

Out[193]:

**0** 37 Female NaN No Yes Often 6-25

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | 44 | M |  | NaN |  | No | No | Rarely | More than  1000 |
| **2** | 32 | Male |  | NaN |  | No | No | Rarely | 6-25 |
| **3** | 31 | Male |  | NaN |  | Yes | Yes | Often | 26-100 |
| **4** | 31 | Male |  | NaN |  | No | No | Never | 100-500 |

5 rows × 23 columns

In [194… *#Clean 'Gender'* gender **=** train\_df['Gender']**.**unique() print(gender)

['Female' 'M' 'Male' 'male' 'female' 'm' 'Male-ish' 'maile' 'Trans-female'

'Cis Female' 'F' 'something kinda male?' 'Cis Male' 'Woman' 'f' 'Mal'

'Male (CIS)' 'queer/she/they' 'non-binary' 'Femake' 'woman' 'Make' 'Nah'

'All' 'Enby' 'fluid' 'Genderqueer' 'Female ' 'Androgyne' 'Agender'

'cis-female/femme' 'Guy (-ish) ^\_^' 'male leaning androgynous' 'Male '

'Man' 'Trans woman' 'msle' 'Neuter' 'Female (trans)' 'queer'

'Female (cis)' 'Mail' 'cis male' 'A little about you' 'Malr' 'p' 'femail'

'Cis Man' 'ostensibly male, unsure what that really means']

In [195…

,

|  |
| --- |
| *#Made gender groups*  male\_str **=** ["male", "m", "male-ish", "maile", "mal", "male (cis)", "make", "male " trans\_str **=** ["trans-female", "something kinda male?", "queer/she/they", "non-binary female\_str **=** ["cis female", "f", "female", "woman", "femake", "female ","cis-femal  **for** (row, col) **in** train\_df**.**iterrows():  **if** str**.**lower(col**.**Gender) **in** male\_str:  train\_df['Gender']**.**replace(to\_replace**=**col**.**Gender, value**=**'male', inplace**=**  **if** str**.**lower(col**.**Gender) **in** female\_str:  train\_df['Gender']**.**replace(to\_replace**=**col**.**Gender, value**=**'female', inplace**=**  **if** str**.**lower(col**.**Gender) **in** trans\_str:  train\_df['Gender']**.**replace(to\_replace**=**col**.**Gender, value**=**'trans', inplace**=**    *#Get rid of bullshit*  stk\_list **=** ['A little about you', 'p'] train\_df **=** train\_df[**~**train\_df['Gender']**.**isin(stk\_list)]    print(train\_df['Gender']**.**unique()) |

**Tru**

**T**

**Tr**

['female' 'male' 'trans']

In [196… *#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", |

"

In [197… *#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']

In [198… *#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], 'D print(train\_df['work\_interfere']**.**unique())

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

|  |
| --- |
| *#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) |

In [199…

label\_Age [18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,

36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 53, 54, 55, 56, 5

7, 58, 60, 61, 62, 65, 72]

label\_Gender ['female', 'male', 'trans'] label\_self\_employed ['No', 'Yes'] label\_family\_history ['No', 'Yes'] label\_treatment ['No', 'Yes']

label\_work\_interfere ["Don't know", 'Never', 'Often', 'Rarely', 'Sometimes'] label\_no\_employees ['1-5', '100-500', '26-100', '500-1000', '6-25', 'More than 100

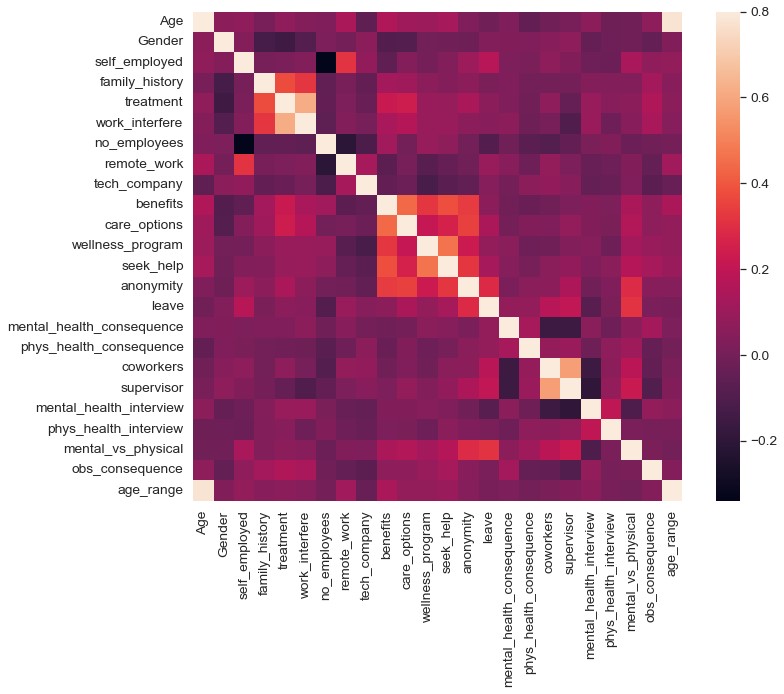
0']

label\_remote\_work ['No', 'Yes'] label\_tech\_company ['No', 'Yes'] label\_benefits ["Don't know", 'No', 'Yes'] label\_care\_options ['No', 'Not sure', 'Yes'] label\_wellness\_program ["Don't know", 'No', 'Yes'] label\_seek\_help ["Don't know", 'No', 'Yes'] label\_anonymity ["Don't know", 'No', 'Yes']

label\_leave ["Don't know", 'Somewhat difficult', 'Somewhat easy', 'Very difficul t', 'Very easy']

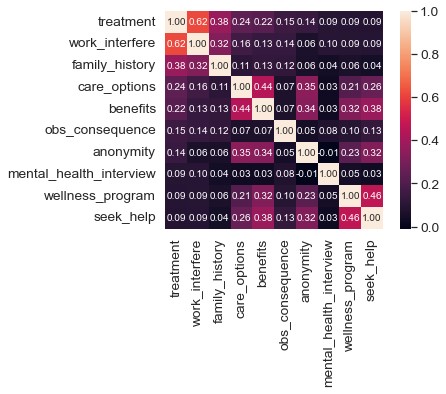
label\_mental\_health\_consequence ['Maybe', 'No', 'Yes'] label\_phys\_health\_consequence ['Maybe', 'No', 'Yes'] label\_coworkers ['No', 'Some of them', 'Yes'] label\_supervisor ['No', 'Some of them', 'Yes'] label\_mental\_health\_interview ['Maybe', 'No', 'Yes'] label\_phys\_health\_interview ['Maybe', 'No', 'Yes'] label\_mental\_vs\_physical ["Don't know", 'No', 'Yes'] label\_obs\_consequence ['No', 'Yes'] label\_age\_range ['0-20', '21-30', '31-65', '66-100']

In [200… *#correlation matrix* 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**=**{'siz plt**.**show() |

In [201…



|  |
| --- |
| g **=** sns**.**FacetGrid(train\_df, col**=**'treatment', size**=**5) g **=** g**.**map(sns**.**distplot, "Age") |

In [202…

C:\ProgramData\Anaconda3\lib\site-packages\seaborn\axisgrid.py:337: UserWarning: T he `size` parameter has been renamed to `height`; please update your code.

warnings.warn(msg, UserWarning)

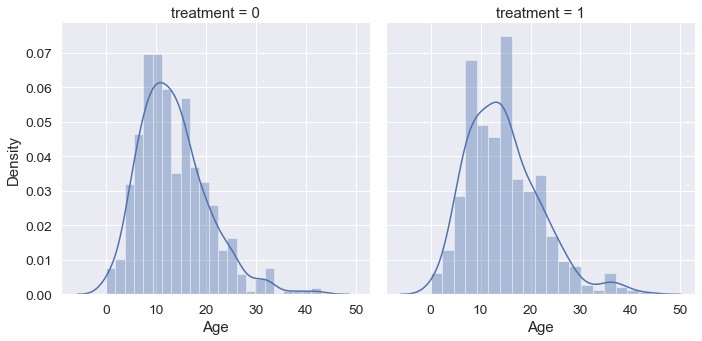
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWa rning: `distplot` is a deprecated function and will be removed in a future versio

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

warnings.warn(msg, FutureWarning)

C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWa rning: `distplot` is a deprecated function and will be removed in a future versio

n. Please adapt your code to use either `displot` (a figure-level function with si milar flexibility) or `histplot` (an axes-level function for histograms). warnings.warn(msg, FutureWarning)



|  |
| --- |
| o **=** labelDict['label\_age\_range']    g **=** sns**.**factorplot(x**=**"age\_range", y**=**"treatment", hue**=**"Gender", data**=**train\_df, kind |

In [203…

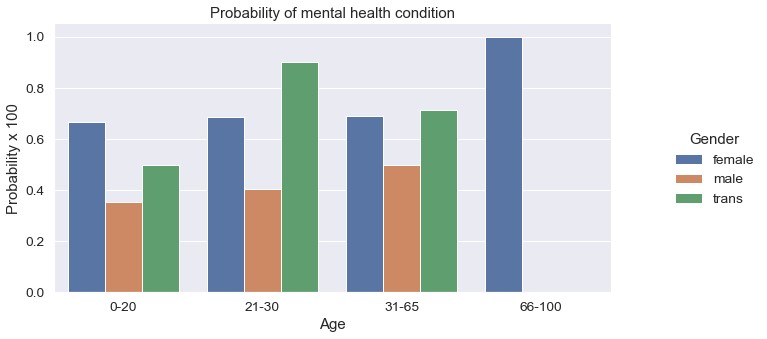
**=**

|  |
| --- |
| 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:\ProgramData\Anaconda3\lib\site-packages\seaborn\categorical.py:3717: UserWarnin g: The `factorplot` function has been renamed to `catplot`. The original name will be removed in a future release. Please update your code. Note that the default `ki nd` in `factorplot` (`'point'`) has changed `'strip'` in `catplot`.

warnings.warn(msg)

C:\ProgramData\Anaconda3\lib\site-packages\seaborn\categorical.py:3723: UserWarnin g: The `size` parameter has been renamed to `height`; please update your code. warnings.warn(msg, UserWarning)



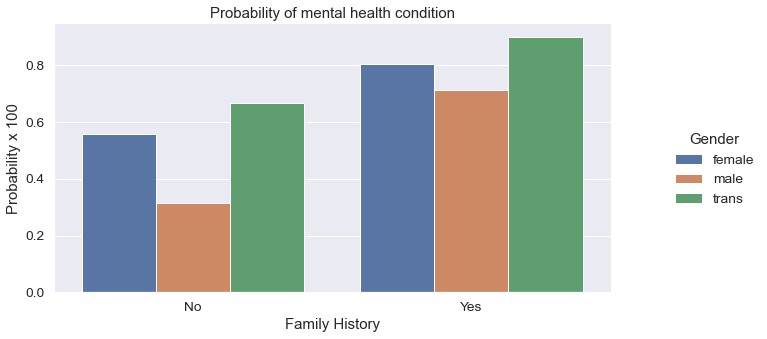
|  |
| --- |
| o **=** labelDict['label\_family\_history']  g **=** sns**.**factorplot(x**=**"family\_history", y**=**"treatment", hue**=**"Gender", data**=**train\_df, g**.**set\_xticklabels(o) plt**.**title('Probability of mental health condition') plt**.**ylabel('Probability x 100') plt**.**xlabel('Family 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() |

In [204…

C:\ProgramData\Anaconda3\lib\site-packages\seaborn\categorical.py:3717: UserWarnin g: The `factorplot` function has been renamed to `catplot`. The original name will be removed in a future release. Please update your code. Note that the default `ki nd` in `factorplot` (`'point'`) has changed `'strip'` in `catplot`.

warnings.warn(msg)

C:\ProgramData\Anaconda3\lib\site-packages\seaborn\categorical.py:3723: UserWarnin g: The `size` parameter has been renamed to `height`; please update your code. warnings.warn(msg, UserWarning)



|  |
| --- |
| *#Barplot to show probabilities for work interfere*    o **=** labelDict['label\_work\_interfere']  g **=** sns**.**factorplot(x**=**"work\_interfere", y**=**"treatment", hue**=**"Gender", data**=**train\_df, 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() |

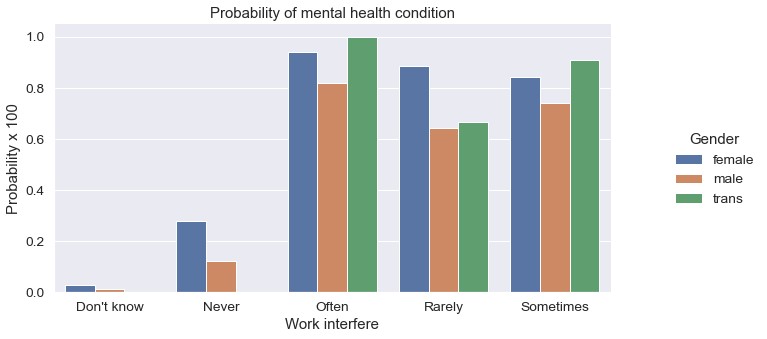
In [205…

C:\ProgramData\Anaconda3\lib\site-packages\seaborn\categorical.py:3717: UserWarnin g: The `factorplot` function has been renamed to `catplot`. The original name will be removed in a future release. Please update your code. Note that the default `ki nd` in `factorplot` (`'point'`) has changed `'strip'` in `catplot`.

warnings.warn(msg)

C:\ProgramData\Anaconda3\lib\site-packages\seaborn\categorical.py:3723: UserWarnin g: The `size` parameter has been renamed to `height`; please update your code.

warnings.warn(msg, UserWarning)



|  |
| --- |
| *#Features Scaling We're going to scale age, because is extremely different from the*    *# Scaling Age* scaler **=** MinMaxScaler()  train\_df['Age'] **=** scaler**.**fit\_transform(train\_df[['Age']]) train\_df**.**head() |

In [206…

Out[206]: **Age Gender self\_employed family\_history treatment work\_interfere no\_employees rem**

**0** 0.431818 0 0 0 1 2 4

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | 0.590909 | 1 |  | 0 |  | 0 |  | 0 |  | 3 | 5 |
| **2** | 0.318182 | 1 |  | 0 |  | 0 |  | 0 |  | 3 | 4 |
| **3** | 0.295455 | 1 |  | 0 |  | 1 |  | 1 |  | 2 | 2 |
| **4** | 0.295455 | 1 |  | 0 |  | 0 |  | 0 |  | 1 | 1 |

5 rows × 24 columns

|  |
| --- |
| *# define X and y*  feature\_cols **=** ['Age', 'Gender', 'family\_history', 'benefits', 'care\_options', 'ano 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\_st  *# Create dictionaries for final graph*  *# Use: methodDict['Stacking'] = accuracy\_score* methodDict **=** {} rmseDict **=** () |

In [207…

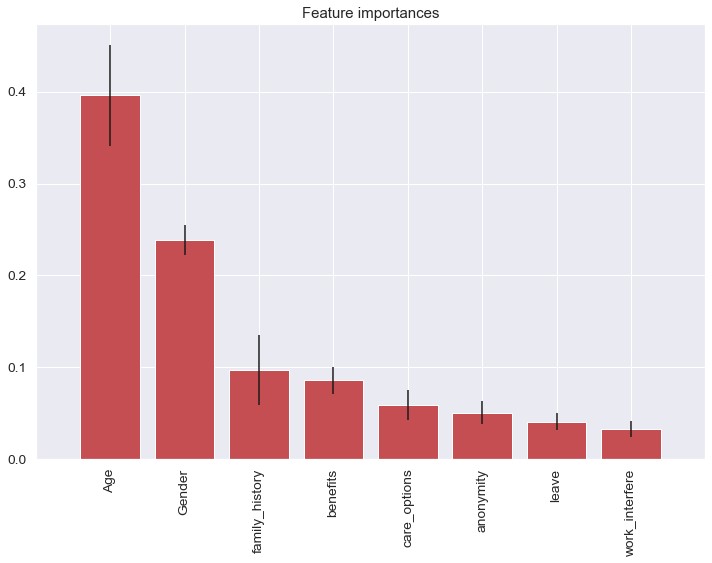
|  |
| --- |
| *# Build a forest and compute the feature importances* forest **=** ExtraTreesClassifier(n\_estimators**=**250, random\_state**=**0)    forest**.**fit(X, y) importances **=** forest**.**feature\_importances\_  std **=** np**.**std([tree**.**feature\_importances\_ **for** tree **in** forest**.**estimators\_], axis**=**0)  indices **=** np**.**argsort(importances)[::**-**1]    labels **=** [] |

In [208…

**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()



|  |
| --- |
| *#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 f*  *# examine the class distribution of the testing set (using a Pandas Series meth*  print('Null accuracy:\n', y\_test**.**value\_counts())    *# calculate the percentage of ones*  print('Percentage of ones:', y\_test**.**mean())  *# calculate the percentage of zeros*  print('Percentage of zeros:',1 **-** y\_test**.**mean())    *#Comparing the true and predicted response values* print('True:', y\_test**.**values[0:25]) print('Pred:', y\_pred\_class[0:25])    *#Confusion matrix*  *# save confusion matrix and slice into four pieces* |

In [209…

|  |
| --- |
| 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 predi*  false\_positive\_rate **=** FP **/** float(TN **+** FP)  print('False Positive Rate:', false\_positive\_rate)  *#Precision: When a positive value is predicted, how often is the prediction cor*  print('Precision:', metrics**.**precision\_score(y\_test, y\_pred\_class))      *# IMPORTANT: first argument is true values, second argument is predicted probab*  print('AUC Score:', metrics**.**roc\_auc\_score(y\_test, y\_pred\_class))  *# calculate cross-validated AUC*  print('Cross-validated AUC:', cross\_val\_score(model, X, y, cv**=**10, scoring**=**'roc\_    *##########################################* *#Adjusting the classification threshold*  *##########################################*  *# print the first 10 predicted responses*  print('First 10 predicted responses:\n', model**.**predict(X\_test)[0:10]) *# print the first 10 predicted probabilities of class membership* print('First 10 predicted probabilities of class members:\n', model**.**predict\_pro    *# print the first 10 predicted probabilities for class 1* model**.**predict\_proba(X\_test)[0:10, 1]    *# store the predicted probabilities for class 1* y\_pred\_prob **=** model**.**predict\_proba(X\_test)[:, 1]  **if** plot **==** **True**:  *# histogram of predicted probabilities* plt**.**rcParams['font.size'] **=** 12 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]    *# 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 probab*  *# 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)' **%** 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 specifi*  **def** evaluate\_threshold(threshold):  *#Sensitivity: When the actual value is positive, how often is the predictio*  *#Specificity: When the actual value is negative, how often is the predictio*  print('Specificity for ' **+** str(threshold) **+** ' :', 1 **-** fpr[thresholds **>** thre  *# 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 |

r

|  |
| --- |
| *#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-a* |

In [210…

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* k\_range **=** list(range(1, 31)) print(k\_range)    *# create a parameter grid: map the parameter names to the values that should be*  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\_) |

In [211…

In [212…

**=**

|  |
| --- |
| *#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 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', rand**.**fit(X, y) |

n\_i

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*  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\_) |

In [213…

|  |
| --- |
| *#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)    accuracy\_score **=** evalClassModel(logreg, y\_test, y\_pred\_class, **True**)  *#Data for final graph*  methodDict['Log. Regression'] **=** accuracy\_score **\*** 100 |

In [214…

In [215… logisticRegression()

Accuracy: 0.7962962962962963 Null accuracy:

1. 191
2. 187

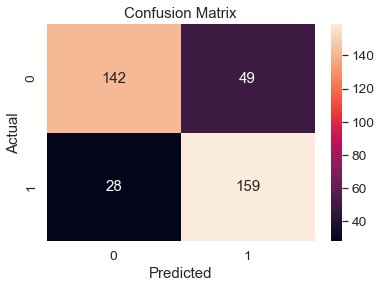
Name: treatment, dtype: int64

Percentage of ones: 0.4947089947089947

Percentage of zeros: 0.5052910052910053

True: [0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1 0 0]

Pred: [1 0 0 0 1 1 0 1 0 1 0 1 1 0 1 1 1 1 0 0 0 0 1 0 0]



Classification Accuracy: 0.7962962962962963 Classification Error: 0.20370370370370372

False Positive Rate: 0.25654450261780104

Precision: 0.7644230769230769

AUC Score: 0.7968614385306716

Cross-validated AUC: 0.8753623882722146 First 10 predicted responses:

[1 0 0 0 1 1 0 1 0 1]

First 10 predicted probabilities of class members:

[[0.09193053 0.90806947]

[0.95991564 0.04008436]

[0.96547467 0.03452533]

[0.78757121 0.21242879]

[0.38959922 0.61040078] [0.05264207 0.94735793]

[0.75035574 0.24964426]

[0.19065116 0.80934884]

[0.61612081 0.38387919]

[0.47699963 0.52300037]] First 10 predicted probabilities:

[[0.90806947]

[0.04008436] [0.03452533]

[0.21242879]

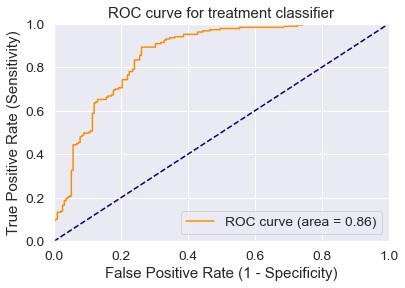
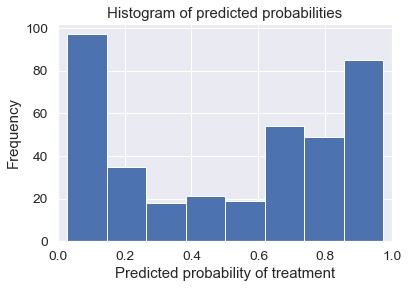
[0.61040078]

[0.94735793] [0.24964426]

[0.80934884]

[0.38387919]

[0.52300037]]



[[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)    *# make class predictions for the testing set* y\_pred\_class **=** knn**.**predict(X\_test)    accuracy\_score **=** evalClassModel(knn, y\_test, y\_pred\_class, **True**)    *#Data for final graph*  methodDict['K-Neighbors'] **=** accuracy\_score **\*** 100 |

In [216…

In [217… Knn()

Rand. Best Score: 0.8217650793650794

Rand. Best Params: {'weights': 'uniform', 'n\_neighbors': 27}

[0.819, 0.817, 0.819, 0.822, 0.822, 0.816, 0.814, 0.822, 0.815, 0.816, 0.822, 0.81

9, 0.819, 0.816, 0.822, 0.817, 0.819, 0.814, 0.822, 0.819]

Accuracy: 0.8042328042328042 Null accuracy:

1. 191
2. 187

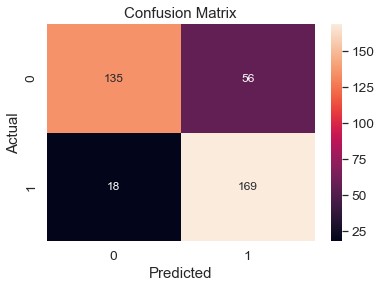
Name: treatment, dtype: int64

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]



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. ]

1. ]

[0.33333333] [0.62962963]

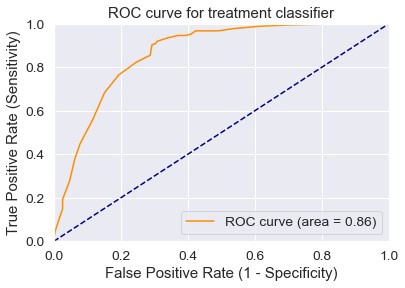
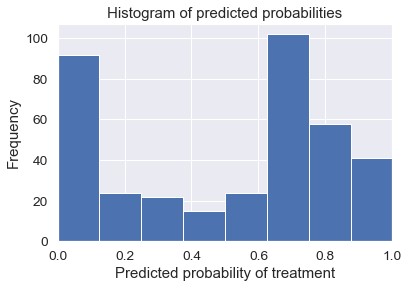
[0.96296296]

[0.40740741]

[0.62962963]

[0.66666667]

[0.66666667]]



[[135 56]

[ 18 169]]

|  |
| --- |
| **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 tree**.**fit(X\_train, y\_train)    *# make class predictions for the testing set* y\_pred\_class **=** tree**.**predict(X\_test)    accuracy\_score **=** evalClassModel(tree, y\_test, y\_pred\_class, **True**)  *#Data for final graph*  methodDict['Decision Tree Classifier'] **=** accuracy\_score **\*** 100 |

In [218…

,

|  |
| --- |
| treeClassifier() |

In [219…

Rand. Best Score: 0.8305206349206349

Rand. Best Params: {'criterion': 'entropy', 'max\_depth': 3, 'max\_features': 6, 'm in\_samples\_leaf': 4, 'min\_samples\_split': 3}

[0.829, 0.831, 0.824, 0.817, 0.831, 0.829, 0.831, 0.831, 0.807, 0.831, 0.831, 0.83

1, 0.83, 0.826, 0.826, 0.831, 0.83, 0.831, 0.811, 0.831] Accuracy: 0.8068783068783069 Null accuracy:

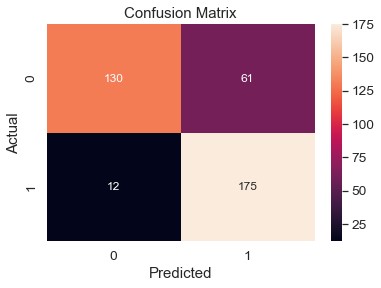
1. 191
2. 187

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]



Classification Accuracy: 0.8068783068783069

Classification Error: 0.19312169312169314

False Positive Rate: 0.3193717277486911

Precision: 0.7415254237288136

AUC Score: 0.8082285746283282

Cross-validated AUC: 0.8880490224390234 First 10 predicted responses:

[1 0 0 0 1 1 0 1 1 1]

First 10 predicted probabilities of class members:

[[0.18 0.82 ]

[0.97959184 0.02040816]

[1. 0. ]

[0.8778626 0.1221374 ]

[0.36097561 0.63902439] [0.18 0.82 ]

[0.8778626 0.1221374 ]

[0.11320755 0.88679245]

[0.36097561 0.63902439]

[0.36097561 0.63902439]]

First 10 predicted probabilities:

[[0.82 ]

[0.02040816] [0. ]

[0.1221374 ]

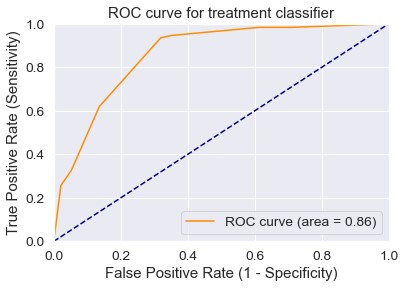
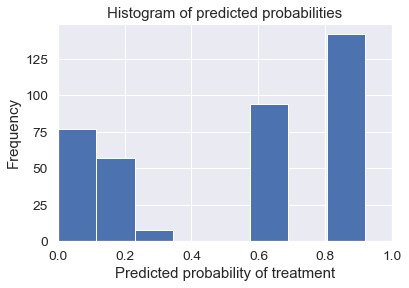
[0.63902439]

[0.82 ] [0.1221374 ]

[0.88679245]

[0.63902439]

[0.63902439]]



[[130 61]

[ 12 175]]

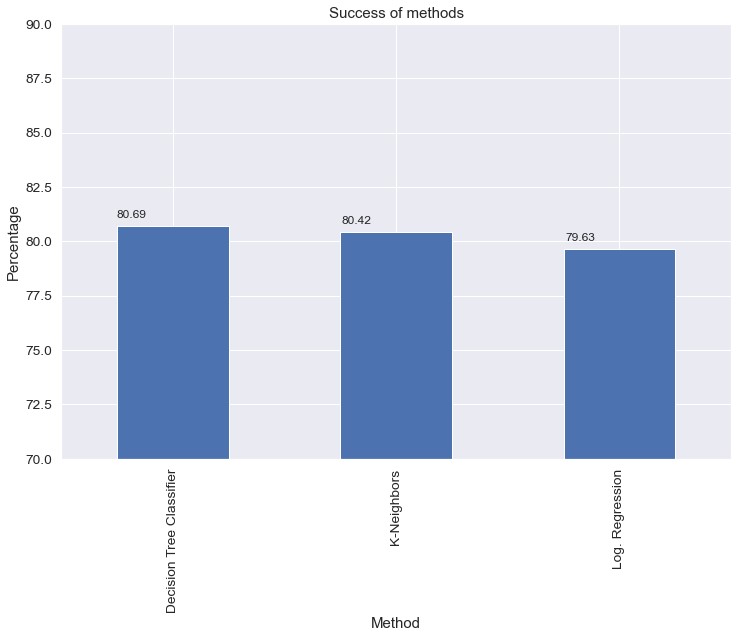
|  |
| --- |
| **def** plotSuccess():  s **=** pd**.**Series(methodDict) s **=** s**.**sort\_values(ascending**=False**) plt**.**figure(figsize**=**(12,8))  *#Colors*  ax **=** s**.**plot(kind**=**'bar') **for** p **in** ax**.**patches: ax**.**annotate(str(round(p**.**get\_height(),2)), (p**.**get\_x() **\*** 1.005, p**.**get\_height plt**.**ylim([70.0, 90.0]) plt**.**xlabel('Method') plt**.**ylabel('Percentage') plt**.**title('Success of methods')  plt**.**show() |

In [220…

(

|  |
| --- |
| plotSuccess() |

In [221…



|  |
| --- |
| *# 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(50) |

In [222…

Out[222]: **Index Treatment**

**0** 5 1

|  |  |  |
| --- | --- | --- |
| **1** | 494 | 0 |
| **2** | 52 | 0 |
| **3** | 984 | 0 |
| **4** | 186 | 0 |
| **5** | 18 | 1 |
| **6** | 317 | 0 |
| **7** | 511 | 1 |
| **8** | 364 | 1 |
| **9** | 571 | 1 |
| **10** | 609 | 0 |
| **11** | 1147 | 1 |
| **12** | 922 | 1 |
| **13** | 461 | 0 |
| **14** | 740 | 1 |
| **15** | 955 | 1 |
| **16** | 814 | 1 |
| **17** | 1160 | 1 |
| **18** | 85 | 0 |
| **19** | 733 | 0 |
| **20** | 1112 | 0 |
| **21** | 124 | 0 |
| **22** | 1040 | 1 |
| **23** | 492 | 0 |
| **24** | 1159 | 0 |
| **25** | 211 | 1 |
| **26** | 1020 | 0 |
| **27** | 892 | 0 |
| **28** | 453 | 0 |
| **29** | 646 | 1 |
| **30** | 161 | 1 |
| **31** | 811 | 1 |
| **32** | 1104 | 0 |
| **33** | 45 | 1 |
| **34** | 1241 | 1 |
| **35** | 874 | 0 |

**Index Treatment**

**36** 921 1

|  |  |  |
| --- | --- | --- |
| **37** | 1191 | 1 |
| **38** | 481 | 1 |
| **39** | 308 | 0 |
| **40** | 269 | 0 |
| **41** | 731 | 0 |
| **42** | 1017 | 1 |
| **43** | 1193 | 0 |
| **44** | 875 | 1 |
| **45** | 1 | 1 |
| **46** | 796 | 1 |
| **47** | 1092 | 1 |
| **48** | 141 | 1 |
| **49** | 1231 | 0 |

|  |
| --- |
|  |