Project Report

TO PERFORM PREDICTIVE ANALYSES ON A DATASET AND DEVELOP MODEL USING THE PYTHON PROGRAMMING LANGUAGE

MODULE TITLE: DATA ANALYTICS AND VISUALISATION – PART II
MODULE CODE: B9IS107

GROUP MEMBERS:

SINGUPURAM NAVIN KUMAR | 10612366

MODULE LEADER: DR SHAZIA A AFZAL

TABLE OF CONTENTS

TABLE OF FIGURES

1. CHOOSING AND EXPLORING DATA

The survey on the mental health was selected to know whether a patient needs treatment or not. Also, to understand the predictive analysis which is a series of surveys conducted in more than 20 countries that assesses the prevalence and treatment of mental disorders.

1.1 DATA SOURCE

LINK FOR THE DATASET

It is available on the Kaggle Dataset Repository website.

DATASET INFORMATION

This information comes from a 2014 poll that assesses attitudes about mental health as well as the prevalence of mental health issues in the IT workplace.

REASONS FOR SELECTING DATBASE

TO UNDERSTAND:

- In what ways does the prevalence of mental health disorders and societal attitudes towards mental health differ across various geographical regions?
- What are the most significant factors that can predict the occurrence of mental health disorders or specific attitudes towards mental health within a work environment?

1.2 DATA EXPLORATION

a. Understanding the data as part of the data-set of the mental health.

```
[2] import pandas as pd  # functions or library for python
    from sklearn.model_selection import train_test_split # splitting data arrays into two subsets: for training data and for testing data.
    from sklearn.linear_model import LogisticRegression  # create a Logistic Regression classifier object
    from sklearn import metrics
    from sklearn.preprocessing import StandardScaler
    import seaborn as sn # seaborn uses matplotlib to draw its plots.
    import matplotlib.pyplot as plt # to plot graph
    import warnings


    warnings.filterwarnings('ignore')


    train_df  = pd.read_csv("/content/sample_data/survey.csv")


    #visualizations -uderstanding data

    train_df .head()
```

	Timestamp	Age	Gender	Country	state	self_employed	family_history	treatment	work_interfere	no_employees	...	leave	mental_health_c
0	2014-08-27 11:29:31	37	Female	United States	IL	NaN	No	Yes	Often	6-25	...	Somewhat easy	
1	2014-08-27 11:29:37	44	M	United States	IN	NaN	No	No	Rarely	More than 1000	...	Don't know	

Figure 1: Reading of the Data

b. On reading of the csv file the columns formed and are displayed as below

```
train_df.columns
```

```
Index(['Timestamp', 'Age', 'Gender', 'Country', 'state', 'self_employed',
       'family_history', 'treatment', 'work_interfere', 'no_employees',
       'remote_work', 'tech_company', 'benefits', 'care_options',
       'wellness_program', 'seek_help', 'anonymity', 'leave',
       'mental_health_consequence', 'phys_health_consequence', 'coworkers',
       'supervisor', 'mental_health_interview', 'phys_health_interview',
       'mental_vs_physical', 'obs_consequence', 'comments'],
      dtype='object')
```

Figure 2: Data Set Columns

c. On reading of the csv file the tail data displayed as below

```
[7] train_df.tail()
```

	Timestamp	Age	Gender	Country	state	self_employed	family_history	treatment	work_interfere	no_employees	...	leave	mental_healt
1254	2015-09-12 11:17:21	26	male	United Kingdom	NaN	No	No	Yes	NaN	26-100	...	Somewhat easy	
1255	2015-09-26 01:07:35	32	Male	United States	IL	No	Yes	Yes	Often	26-100	...	Somewhat difficult	
1256	2015-11-07 12:36:58	34	male	United States	CA	No	Yes	Yes	Sometimes	More than 1000	...	Somewhat difficult	
1257	2015-11-30 21:25:06	46	f	United States	NC	No	No	No	NaN	100-500	...	Don't know	
1258	2016-02-01	25	Male	United States	IL	No	Yes	Yes	Sometimes	26-100	...	Don't know	

Figure 3: Tail Data

2. ANALYSING, CLEANING, AND PREPARING DATA

2.1 ANALYSING

- **Timestamp**
- **Age**
- **Gender**
- **Country**
- **state**: What is your current state or territory of residence, if you live in the United States?
- **self_employed**: Are you self-employed?
- **family_history**: Is there a history of mental illness in your family?
- **treatment**: Have you sought treatment for a mental health condition?
- **work**_interfere: If you have a mental health condition, do you feel that it interferes with your work?
- **no_employees**: How many employees does your company or organization have?
- **remote**_work: Do you work remotely (outside of an office) at least 50% of the time?
- **tech_company**: Is your employer primarily a tech company/organization?
- **benefits**: Does your employer provide mental health benefits?
- **care**_options: Do you know the options for mental health care your employer provides?
- **wellness_program**: Has your employer ever discussed mental health as part of an employee wellness program?
- **seek_help**: Does your employer provide resources to learn more about mental health issues and how to seek help?
- **anonymity**: If you decide to use mental health or substance abuse treatment resources, is your anonymity protected?

- **leave**: How easy is it for you to take medical leave for a mental health condition?
- **mental_health_consequence**: Do you think that discussing a mental health issue with your employer would have negative consequences?
- **phys_health_consequence**: Do you think that discussing a physical health issue with your employer would have negative consequences?
- **coworkers**: Would you be willing to discuss a mental health issue with your coworkers?
- **supervisor**: Would you be willing to discuss a mental health issue with your direct supervisor(s)?
- **mental_health_interview**: Would you bring up a mental health issue during a job interview with a potential employer?
- **phys_health_interview**: Would you bring up a physical health issue during a job interview with a potential employer?
- **mental_vs_physical**: Do you believe that your employer takes mental health as seriously as physical health?
- **obs_consequence**: Have you personally observed or heard of negative consequences for coworkers with mental health conditions in your workplace?
- **comments**: Are there any additional notes or comments you would like to provide?

a. Analysing the data in a form of different ways like displayed below as shap and the info of the data- set chosen.

```
train_df.shape
```

```
(1259, 27)
```

```
train_df.info
```

```
1255  2015-09-26 01:07:35   32    Male   United States    IL            No
1256  2015-11-07 12:36:58   34    male   United States    CA            No
1257  2015-11-30 21:25:06   46       f   United States    NC            No
1258  2016-02-01 23:04:31   25    Male   United States    IL            No

     family_history treatment work_interfere    no_employees  ...  \
0                No       Yes          Often            6-25  ...
1                No        No         Rarely  More than 1000  ...
2                No        No         Rarely            6-25  ...
3               Yes       Yes          Often          26-100  ...
4                No        No          Never         100-500  ...
...             ...       ...            ...             ...  ...
1254             No       Yes            NaN          26-100  ...
1255            Yes       Yes          Often          26-100  ...
1256            Yes       Yes      Sometimes  More than 1000  ...
1257             No        No            NaN         100-500  ...
1258            Yes       Yes      Sometimes          26-100  ...

                  leave mental_health_consequence phys_health_consequence  \
0         Somewhat easy                         No                      No
```

Figure 4: Shape and Info of the Dataset

b. Describing the data of the data- set chosen with the age and the count, mean and different parameters.

```
[8]  train_df.describe()
```

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

Figure 5: Describe of the data

c. NUnique the data of the data- set chosen with the age and the count, mean and different parameters.

```
train_df.nunique()
```

```
Timestamp                    1246
Age                            53
Gender                         49
Country                        48
state                          45
self_employed                   2
family_history                  2
treatment                       2
work_interfere                  4
no_employees                    6
remote_work                     2
tech_company                    2
benefits                        3
care_options                    3
wellness_program                3
seek_help                       3
anonymity                       3
leave                           5
mental_health_consequence       3
```

Figure 6: nunique of the Dataset

d. Analysing the missing data and checking the head by sorting it

```
#missing data
total = train_df.isnull().sum().sort_values(ascending=False)
percent = (train_df.isnull().sum()/train_df.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
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
```

Figure 7: Missing Data

e. Working on the missing data by dropping the three columns and checking further if there is any missing data and printing the head.

```
#working on the missing data
#Let's get rid of the variables "Timestamp",“comments”, “state”
train_df = train_df.drop(['comments'], axis= 1)
train_df = train_df.drop(['state'], axis= 1)
train_df = train_df.drop(['Timestamp'], axis= 1)

train_df.isnull().sum().max() #just checking that there's no missing data missing...
train_df.head(5)
```

	Age	Gender	Country	self_employed	family_history	treatment	work_interfere	no_employees	remote_work	tech_company	...	anonymity	
0	37	Female	United States	NaN	No	Yes	Often	6-25	No	Yes	...	Yes	Son
1	44	M	United States	NaN	No	No	Rarely	More than 1000	No	No	...	Don't know	
2	32	Male	Canada	NaN	No	No	Rarely	6-25	No	Yes	...	Don't know	Son
3	31	Male	United Kingdom	NaN	Yes	Yes	Often	26-100	No	Yes	...	No	Son
4	31	Male	United States	NaN	No	No	Never	100-500	Yes	Yes	...	Don't know	

5 rows × 24 columns

Figure 8: Dealing with missing data

2.2 CLEANING DATA

- Eliminating unnecessary columns from a dataframe.

- Analyzing the dataset, computing the total of null values for each column, and eliminating any rows that have null values.

a. Cleaning the NaN values and assigning the default values for each column.

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

# Create lists by data tpe
intFeatures = ['Age']
stringFeatures = ['Gender', 'Country', 'self_employed', 'family_history', 'treatment', 'work_interfere',
                  'no_employees', 'remote_work', 'tech_company', 'anonymity', 'leave', 'mental_health_consequence',
                  'phys_health_consequence', 'coworkers', 'supervisor', 'mental_health_interview', 'phys_health_interview',
                  'mental_vs_physical', 'obs_consequence', 'benefits', 'care_options', 'wellness_program',
                  'seek_help']
floatFeatures = []
# Clean the NaN's
for feature in train_df:
    if feature in intFeatures:
        train_df[feature] = train_df[feature].fillna(defaultInt)
    elif feature in stringFeatures:
        train_df[feature] = train_df[feature].fillna(defaultString)
    elif feature in floatFeatures:
        train_df[feature] = train_df[feature].fillna(defaultFloat)
    else:
        print('Error: Feature %s not recognized.' % feature)
train_df.head(5)
```

	Age	Gender	Country	self_employed	family_history	treatment	work_interfere	no_employees	remote_work	tech_company	...	anonymity
0	37	Female	United States	NaN	No	Yes	Often	6-25	No	Yes	...	Yes
1	44	M	United States	NaN	No	No	Rarely	More than 1000	No	No	...	Don't know

Figure 9: Assigning default values

b. Cleaning the gender values by lowering the case of the column elements, selecting the gender elements and segregating into gender groups and also removing the unwanted string from the values in the dataset.

```
gender = train_df['Gender'].str.lower()
#print(gender)

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

#Made gender groups
male_str = ["male", "m", "male-ish", "maile", "mal", "male (cis)", "make", "male ", "man","msle", "mail", "malr","cis man", "Cis Male", "cis mal
trans_str = ["trans-female", "something kinda male?", "queer/she/they", "non-binary","nah", "all", "enby", "fluid", "genderqueer", "androgyne",
female_str = ["cis female", "f", "female", "woman",  "femake", "female ","cis-female/femme", "female (cis)", "femail"]

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

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

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

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

#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())
```

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

Figure 10: Seggregating the gender groups

c. Updated the missing age with mean and check the age ranges and for the self employed and the self work interfere which updated with No and Don't know respectively.

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

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

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

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

     ['No' 'Yes']
```

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

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

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

✓ 0s completed at 7:03 PM

Figure 11: Updating the default string for self, work and age with mean

2.3 PREPARING DATA

To double-check the column datatypes

We need to have input as numerical values in machine learning, thus we must ensure that all values are in digits format. We made certain that categorical data was transformed to numbers in accordance with the categories.

1. Data cleaning: This involves identifying and correcting errors in the dataset, such as missing values, outliers, or inconsistent formatting. It is important to ensure that the data is complete and accurate before using it for machine learning.

2. Feature selection: This involves choosing which variables to include in the machine learning model. It is important to select variables that are relevant to the research question and have been shown to be associated with mental health outcomes in previous research.

3. Feature engineering: This involves transforming the variables into a format that can be used in the machine learning model. This may include scaling the variables, encoding categorical variables, or creating new variables based on existing ones.

4. Data splitting: This involves dividing the dataset into training, validation, and testing sets. The training set is used to train the machine learning model, the validation set is used to tune the hyperparameters of the model, and the testing set is used to evaluate the performance of the final model.

2.4 WHAT IS FEATURE SCALING?

The technique of standardizing or normalizing the range of features or variables in a dataset in order to make them comparable and understandable for machine learning algorithms is known as feature scaling.

When the characteristics in a dataset have multiple sizes or units, some algorithms may struggle to appropriately understand the data. Certain algorithms, such as K-nearest neighbors or Support Vector Machines, may be sensitive to feature scale, but others, such as gradient descent, may converge quicker when the features are scaled. As a result, it is frequently advised to scale the features before employing them in machine learning models..

2.5 SELECTING THE DEPENDENT AND INDEPENDENT VARIABLES

DEPENDENT VARIABLE:

Mental health status: This could be measured using a standardized assessment tool such as the General Health Questionnaire (GHQ), the Depression Anxiety and Stress Scale (DASS), or the Patient Health Questionnaire (PHQ-9).

Perception of mental health support: This could be measured using a Likert scale or qualitative data.

INDEPENDENT VARIABLES:

Demographic variables: This could include age, gender, ethnicity, and education level.

Employment-related variables: This could include job role, length of time with current employer, number of working hours, and type of contract.

Technology-related variables: This could include frequency of computer use, type of technology used, and social media use.

Workplace-related variables: This could include workplace culture, work-life balance, and job satisfaction.

a. Below shows the encoding of the data and getting rid of the country attribute

```
from sklearn import preprocessing

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

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

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

```
label_Timestamp ['2014-08-27 11:29:31', '2014-08-27 11:29:37', '2014-08-27 11:29:44', '2014-08-27 11:29:46', '2014-08-27 11:30:22', '2014-08-27
label_Age [-1726, -29, 5, 11, 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, 4
label_Gender ['female', 'male', 'trans']
label_Country ['Australia', 'Austria', 'Belgium', 'Bosnia and Herzegovina', 'Brazil', 'Bulgaria', 'Canada', 'China', 'Colombia', 'Costa Rica',
label_state ['AL', 'AZ', 'CA', 'CO', 'CT', 'DC', 'FL', 'GA', 'IA', 'ID', 'IL', 'IN', 'KS', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN', 'MO', 'MS'
label_self_employed ['NaN', 'No', 'Yes']
label_family_history ['No', 'Yes']
label_treatment ['No', 'Yes']
label_work_interfere ['NaN', 'Never', 'Often', 'Rarely', 'Sometimes']
label_no_employees ['1-5', '100-500', '26-100', '500-1000', '6-25', 'More than 1000']
label_remote_work ['No', 'Yes']
```

```
    labelDict[labelKey] =labelValue

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

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

```
label_Age [18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 53, 54, 55, 56, 57, 58, 60, 61, 62, 65, 72]
label_Gender ['female', 'male', 'trans']
label_Country ['Australia', 'Austria', 'Belgium', 'Bosnia and Herzegovina', 'Brazil', 'Bulgaria', 'Canada', 'China', 'Colombia', 'Costa Rica', 'Croatia', 'Czech Republic', 'Denmark', 'Finland', 'France', 'Ge
label_self_employed ['No', 'Yes']
label_family_history ['No', 'Yes']
label_treatment ['No', 'Yes']
label_work_interfere ["Don't know", 'Never', 'Often', 'Rarely', 'Sometimes']
label_no_employees ['1-5', '100-500', '26-100', '500-1000', '6-25', 'More than 1000']
label_remote_work ['No', 'Yes']
label_tech_company ['No', 'Yes']
label_benefits ["Don't know", 'No', 'Yes']
label_care_options ['No', 'Not sure', 'Yes']
label_wellness_program ["Don't know", 'No', 'Yes']
label_seek_help ["Don't know", 'No', 'Yes']
label_anonymity ["Don't know", 'No', 'Yes']
label_leave ["Don't know", 'Somewhat difficult', 'Somewhat easy', 'Very difficult', 'Very easy']
label_mental_health_consequence ['Maybe', 'No', 'Yes']
label_phys_health_consequence ['Maybe', 'No', 'Yes']
label_coworkers ['No', 'Some of them', 'Yes']
label_supervisor ['No', 'Some of them', 'Yes']
label_mental_health_interview ['Maybe', 'No', 'Yes']
label_phys_health_interview ['Maybe', 'No', 'Yes']
label_mental_vs_physical ["Don't know", 'No', 'Yes']
label_obs_consequence ['No', 'Yes']
label_age_range ['0-20', '21-30', '31-65', '66-100']
```

	Age	Gender	self_employed	family_history	treatment	work_interfere	no_employees	remote_work	tech_company	benefits	...	leave	mental_health_consequence	phys_health_consequence	coworkers	supervisc
0	19	0	0	0	1	2	4	0	1	2	...	2	1	1	1	
1	26	1	0	0	0	3	5	0	0	0	...	0	0	1	0	
2	14	1	0	0	0	3	4	0	1	1	...	1	1	1	2	
3	13	1	0	1	1	2	2	0	1	1	...	1	2	2	1	
4	13	1	0	0	0	1	1	1	1	2	...	0	1	1	1	

Figure 12: Encoding of the data

b. Testing that there aren't any missing data after the cleaning and updating the values

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

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

Figure 13: Testing of the data after preparation

3. MODEL BUILDING, TESTING, AND EVALUATION

3.1 FEATURE SELECTION

While developing a predictive model, the feature selection procedure entails reducing the number of input variables. In some cases, limiting the amount of input variables may improve model performance while simultaneously cutting modeling processing costs.

3.2 CORRELATION MATRIX WITH HEATMAP

A correlation heatmap is a type of graph that displays the strength of correlations between numerical variables. Correlation graphs are used to discover which variables are connected to one another and how strongly this relationship exists. Positive values represent a strong association, whilst negative values represent a weak relationship. The cell values show the strength of the link. Correlation heatmaps may be used to find potential correlations between variables and to assess the strength of these associations.

```
[61] #correlation matrix
     corrmat = train_df.corr()
     f, ax = plt.subplots(figsize=(6, 6))
     sns.heatmap(corrmat, vmax=.8, square=True);
     plt.show()
```

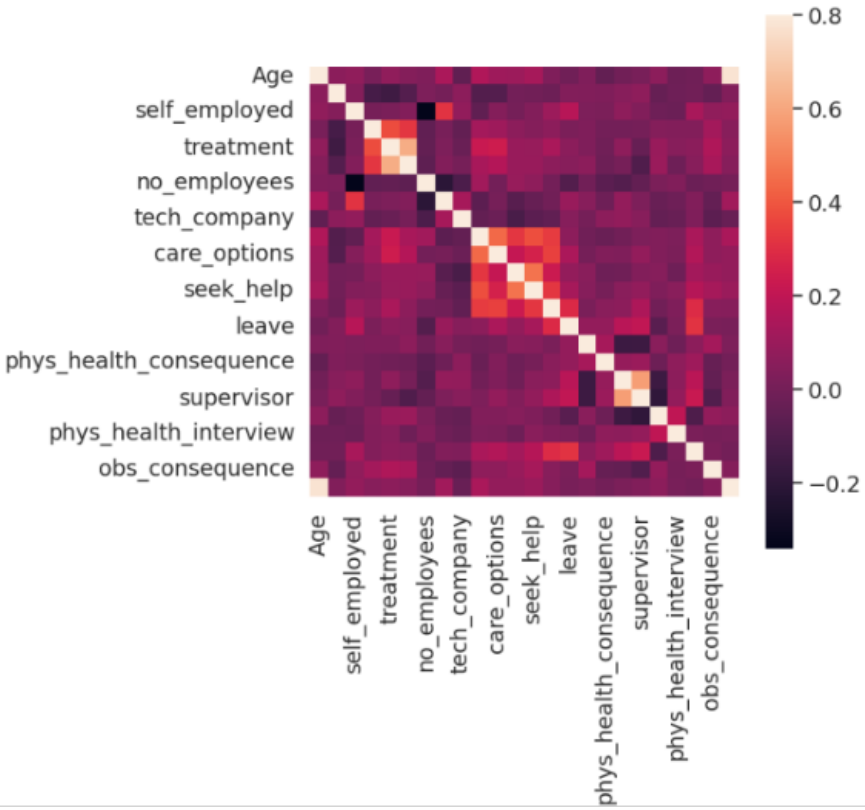

Figure 14: Correlation matrix with heatmap

3.2.1 Treatment correlation matrix – Heap Map

```
[62] #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()
```

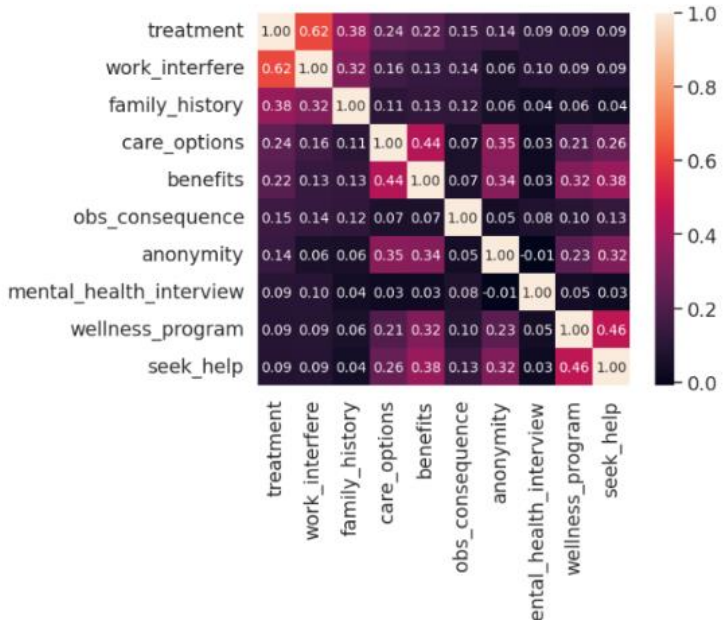

Figure 15: Treatment Correlation Heat Matrix

Correlation is a well-known statistic for measuring how similar two features are. If two characteristics are linearly related, their correlation coefficient is 1. There is no connection between two traits if they are uncorrelated. If the value is greater than the threshold, the feature is enabled.

- A correlation heatmap is an image of a correlation matrix that demonstrates the link between several variables.

- Correlation can range between -1 and 1.

- A correlation between two random variables or bivariate data does not always suggest a causal relationship.

- A scatter plot of these two variables may also be used to determine the correlation between them.

3.3 PAR PLOT

A "pairs plot" scatterplot matrix aids in understanding the pairwise relationship between distinct variables in a dataset.

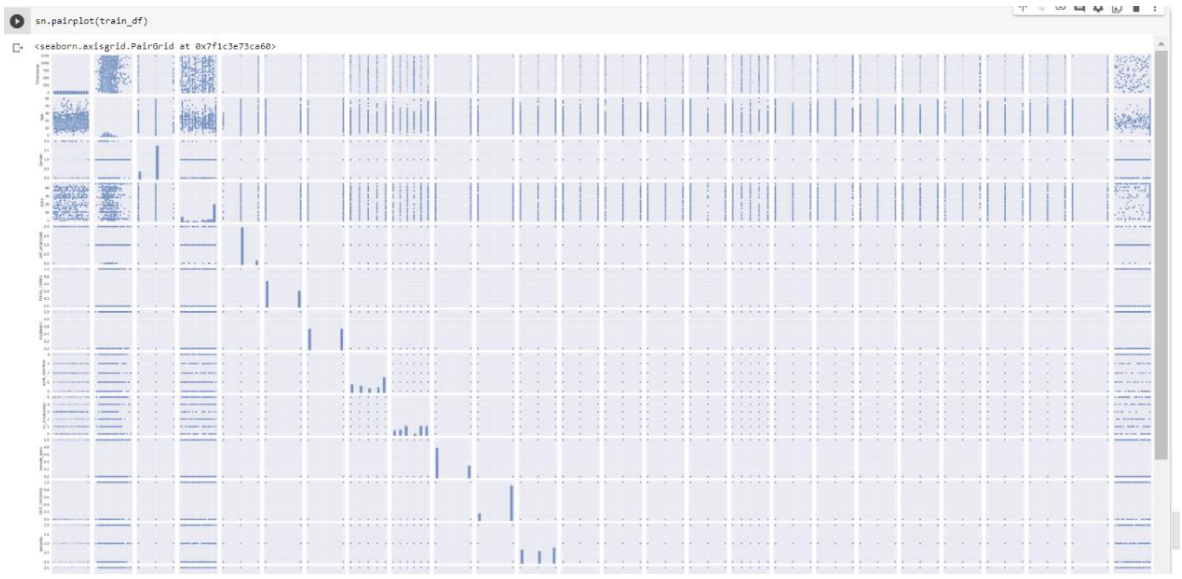

Figure 16: Pair Plot Matrix 1

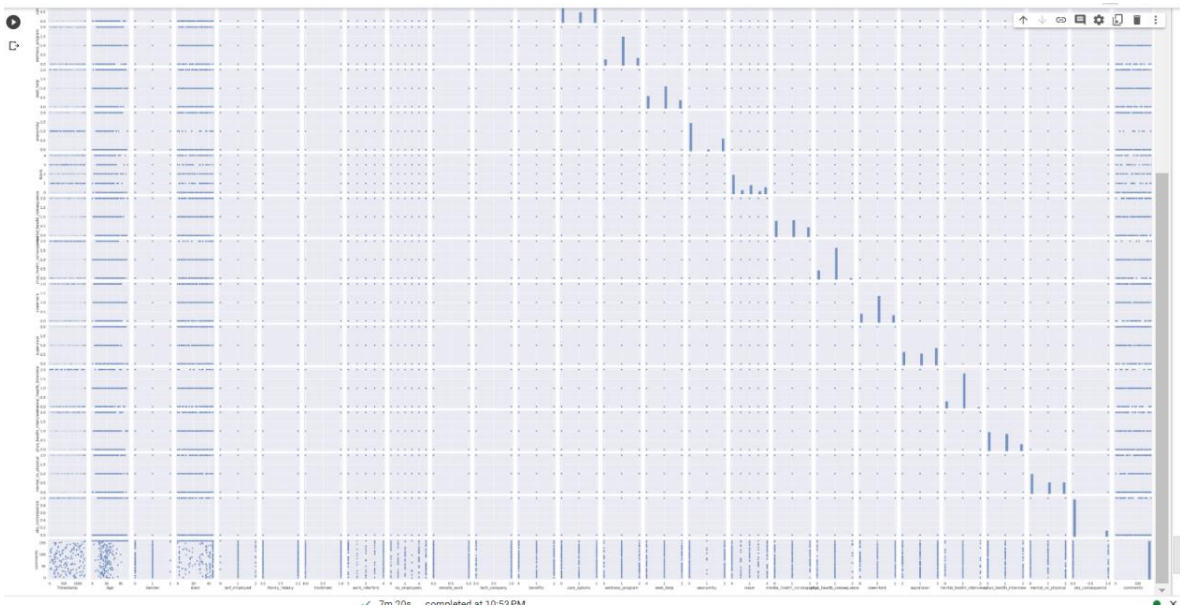

Figure 17: Pair Plot Matrix 2

Users can use the Seaborn Pairplot function to create an axis grid that distributes each numerical variable in the data over the X- and Y-axes in the

form of columns and rows. We may use scatter plots to demonstrate pairwise correlations in addition to the distribution plot, which depicts the data distribution in the column diagonally.

We may plot several variable types on rows and columns, or we can use the pairplot() method to display a subset of variables.

3.4 RELATIONAL PLOTS

Relational graphs are used to represent the statistical relationship between the data elements. Visualization is critical because it allows humans to detect trends and patterns in data. The process of discovering the correlations between and among variables in a dataset is known as statistical analysis.

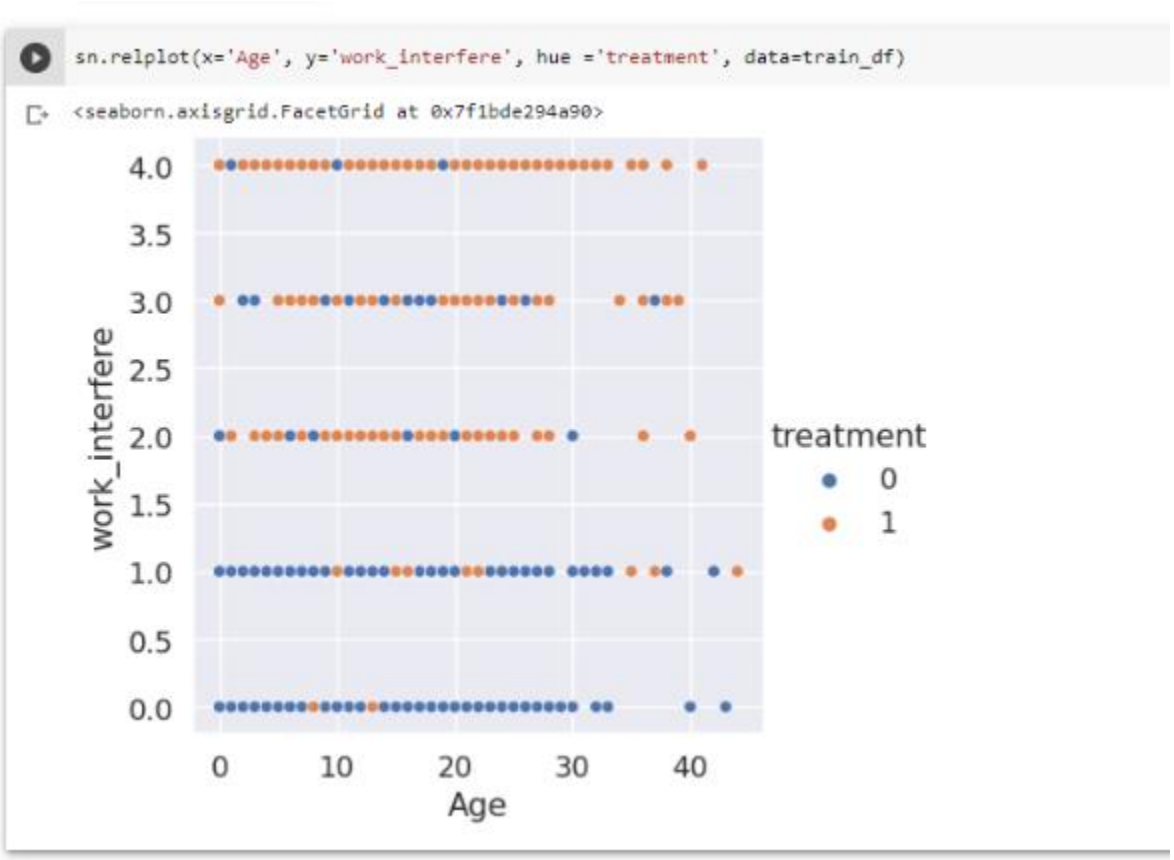

Figure 18: Relational Plot

Here the relational plot uses weight parameter as X and Y parameter as height and we can see that BMI is plotted on the graph with different hues.

3.5 DISTRIBUTION PLOT

A distribution plot, or Distplot, depicts the change in the data distribution. A seaborn plot depicts the overall distribution of continuous data variables.

```
plt.figure(figsize=(6,6))
sn.distplot(train_df["Age"], bins=24)
plt.title("Distribution and density by age")
plt.xlabel("Age")
```

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

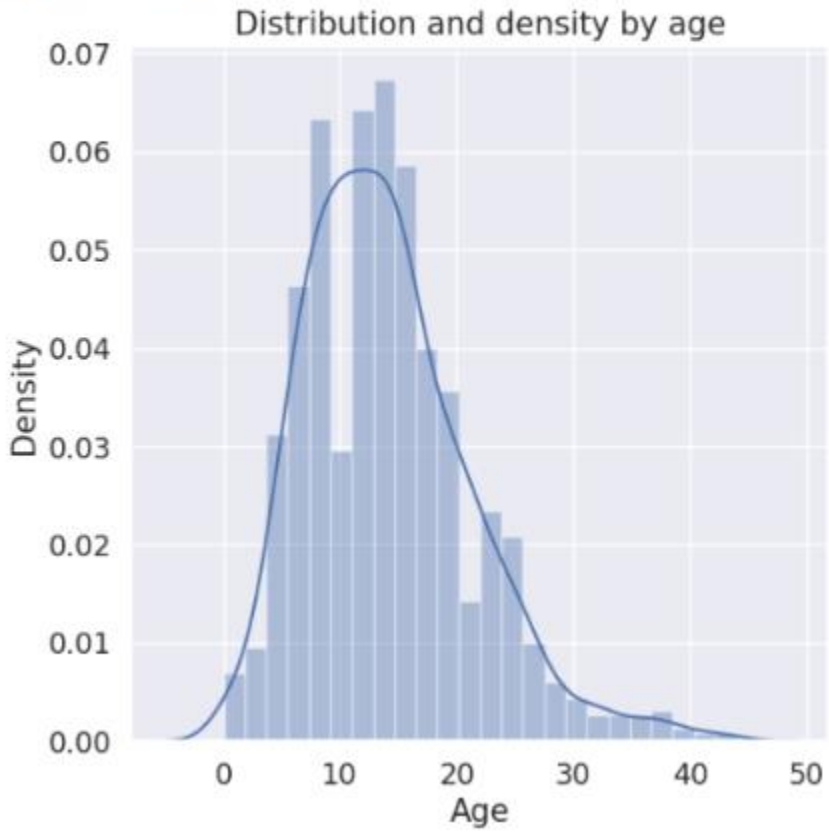

Figure 19: Distribution plot

This scatter plot depicts the treatment values that are dispersed throughout the dataset. We may also see a trend in the distribution of treatment values between ages and treatment group.

```
# Separate by treatment or not
plt.figure(figsize=(8,8))
g = sn.FacetGrid(train_df, col='treatment')
g = g.map(sn.distplot, "Age")
```

```
<Figure size 800x800 with 0 Axes>
```

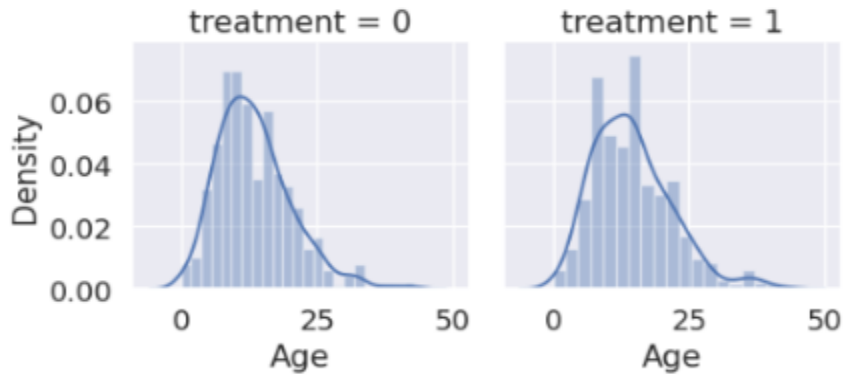

Figure 20: Distribution Plot on seperate treatments

3.6 NESTED BARPLOT

Below is the bar plot which depicts the age on the x axis and the probability on the y axis through the mental health condition probability

```
o = labelDict['label_age_range']

g = sn.catplot(x="age_range", y="treatment", hue="Gender", data=train_df, kind="bar",  ci=None, 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()
```

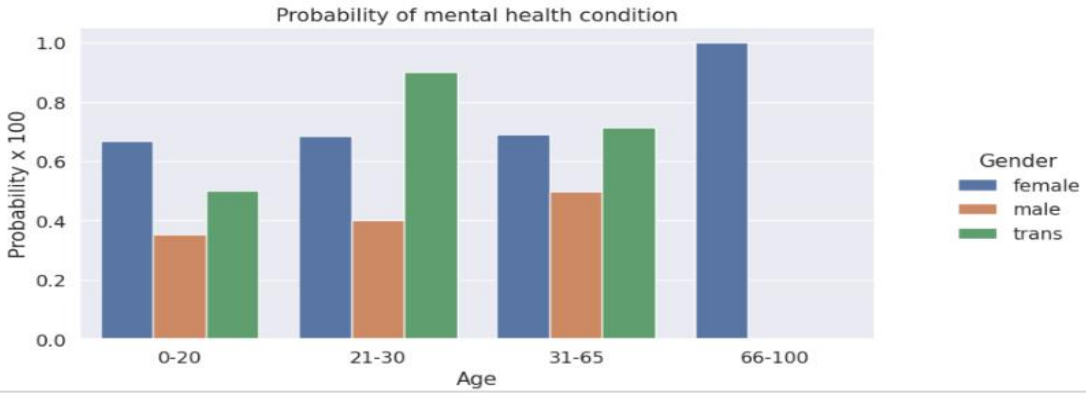

Figure 21: Nested Barplot

3.7 SPLITTING AND TUNING THE DATASET FOR MODEL

Evaluating a Classification Model

This function will evalue:

- **Classification accuracy:** percentage of correct predictions
- **Null accuracy:** accuracy that could be achieved by always predicting the most frequent class
- **Percentage of ones**
- **Percentage of zeros**

3.7.1 REGRESSION TYPES

Regression can be used to determine the statistical relationship between a dependent variable and one or more independent variables. The change in the independent variable is related to the change in the independent variable. This may be divided into two broad categories.

LOGISTIC REGRESSION

When the dependent variable is discrete, logistic regression is employed as a regression analysis approach. For example, true or false, 0 or 1, and so on. As a result, the target variable can only have two values, and the relationship between the target variable and the independent variable is illustrated by a sigmoid curve.

The logit function is used in logistic regression to quantify the relationship between the dependent and independent variables. The logistic regression equation is presented in the figure below.

```
logistic_regression= LogisticRegression(multi_class='multinomial', solver='lbfgs')
logistic_regression.fit(X_train,y_train)
y_pred=logistic_regression.predict(X_test)
#print(X)
print (X_test) #test dataset
print (y_pred) #predicted values
```

```
      Age  Gender  family_history  benefits  care_options  anonymity  leave  \
5      15       1               1         2             1          0      0
494     7       1               0         0             1          0      0
52     13       1               0         0             0          0      0
984     9       1               0         2             1          2      4
186     6       1               0         0             0          0      2
...   ...     ...             ...       ...           ...        ...    ...
1084   14       1               1         2             2          2      2
506    24       1               0         2             0          0      3
1142   10       1               0         1             0          0      2
1124   17       1               0         0             1          2      0
689     9       0               0         0             1          2      4

      work_interfere
5                  4
494                0
52                 0
984                1
186                4
...              ...
1084               4
506                1
1142               4
1124               0
689                4

[378 rows x 8 columns]
[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 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1
 1 1 0 0 0 0 0 1 0 1 1 1 1 1 0 1 1 0 0 0 0 1 0 1 0 1 0 0 1 1 0 1 1 1 1 1 0 0
 0 0 1 1 0 0 1 1 0 0 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1 1 1 1 1 0 0 1 0 1 1 0 0 0
 1 1 0 1 0 1 1 1 1 0 0 1 1 1 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 1 0 0 1 0 1 1 1 1
 0 0 0 1 0 1 1 0 0 1 0 0 1 1 0 1 1 1 1 1 1 1 0 1 0 0 0 1 0 1 0 1 1 1 0 0 1
 1 1 0 0 1 1 0 0 1 0 1 1 1 1 0 0 1 0 1 0 0 0 1 1 0 1 1 1 1 1 1 0 0 0 1 0 1 0
 1 0 1 0 0 1 0 1 1 1 1 0 1 1 1 0 0 1 0 1 1 1 0 1 0 1 1 0 1 0 1 0 1 1 1 0 1
 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 0 1 0 0 0 1 1 0 1 0 0 0 1 1 1 0 1 0 1 0 0 1
 0 0 0 0 0 1 1 1 0 0 0 0 1 0 1 1 1 0 0 1 1 0 1 0 0 1 1 0 1 0 0 1 0 1 1 0 0
 1 0 1 0 1 1 0 1 1 0 0 0 1 1 1 1 0 1 1 1 0 1 1 0 0 1 0 1 1 0 1 0 0 0 1 0 0
```

Figure 22: Logistic Regression predicted values

LINEAR REGRESSION

Linear regression is one of the most fundamental types of regression in machine learning. The linear regression model is made up of a predictor variable and a dependant variable that are linearly connected to one another. When there are several independent variables in the data, linear regression is known as numerous linear regression models.

```
from sklearn.linear_model import LinearRegression
regr=LinearRegression()
regr.fit(X_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
regr.predict(X_test)
```

```
array([ 0.90557221, -0.02974301, -0.05073951,  0.26614095,  0.62402692,
        0.99603419,  0.29665824,  0.77168691,  0.42447605,  0.51097409,
        0.1420698 ,  0.9148211 ,  0.85839355,  0.30475938,  0.98061608,
        0.5612279 ,  0.774113  ,  0.76582779, -0.01546183,  0.13947625,
        0.02809338,  0.20542684,  0.88449862, -0.03138449,  0.16992047,
        0.85054128,  0.09295775,  0.1452791 ,  0.68390302,  0.82197893,
        0.93670255,  0.66560016,  0.21418736,  1.09851635,  0.3238502 ,
        0.31766585,  0.83326833,  0.96906474,  0.85946641, -0.02974301,
        0.15237787, -0.03812685,  0.3747294 ,  0.23536794,  0.83868888,
```

```
        0.35350653,  1.04693664,  0.72155388,  0.72860226,  0.52068119,
       -0.04439538,  0.82534722,  0.66725275,  0.75739389,  0.11279577,
        0.11663484,  0.66564874,  0.76602668,  1.07708274,  0.5987017 ,
        0.35163235,  0.12737447,  0.21976576,  0.12439862,  0.56426905,
        0.65492651,  0.92223358,  0.5785683 ,  0.13788335,  0.97109529,
       -0.05073951,  0.17647736,  0.53846357,  0.233548  ,  0.79666883,
        0.75123584,  1.0063244 ,  0.66718391,  0.19601377,  0.32998098,
        0.2672849 ,  0.90821581,  0.50461524,  0.78125359,  0.71503336,
        0.07023975,  0.15397078,  0.76829544,  0.44738005, -0.02498261,
        0.60495905,  1.08185498,  0.15998553,  0.98657115,  0.62529801,
        0.70161656,  0.9050969 ,  0.99084854,  0.89303516,  0.73032003,
        0.18377639,  0.54322396,  0.09546508,  0.10384892,  0.32147   ,
        0.82253834,  0.05332639,  0.87745953,  0.33077743,  0.70973767,
        1.03547277,  0.8241617 ,  0.2992936 ,  0.03329356,  0.67908488,
        1.00397123,  0.6934238 ,  0.15838153, -0.06264049,  0.80848383,
        0.80844495,  0.1508789 ,  0.28069321,  1.0186236 , -0.02657552,
        0.69009787,  0.68820744,  0.54008284,  0.76387285,  0.03832584,
        0.39322152,  0.74773603,  0.10384892,  0.67710292,  0.13855267,
        0.06693677,  0.05022682,  0.63910455,  0.64751623,  0.08426894,
        0.83368249,  0.73305663,  0.75082047,  0.54208703,  0.75229301,
        0.12030949,  0.28384477,  0.02694942,  0.6044893 ,  0.38972292,
        0.85800381, -0.02022222,  0.68226153,  0.44779421,  0.91157586,
        0.22648517,  0.5077051 ,  0.87550173,  0.13617247,  0.76193946,
        0.6699464 ,  0.50833273,  0.99244145,  0.1195222 ,  0.73039935,
        0.68548676,  0.99322874,  0.50008291,  0.14445   ,  0.5864025 ,
       -0.03098627,  0.64148475,  1.08661537,  0.82364744, -0.08327497,
        0.56466728, -0.04597912,  0.90115999,  0.73336118,  0.10009833,
        0.65020908,  0.13813851,  0.82638967,  0.11517596,  0.73833639,
        0.66440548,  0.88700448,  0.24158722,  0.98132776,  0.87029453,
        0.11212645,  0.8369962 ,  0.55469969,  0.57177522,  0.07964605,
        0.89367123,  0.81650686,  0.68027481,  0.21572815,  0.66798036,
        0.92771335,  0.88150538,  0.92069989,  0.8076485 , -0.01551957,
        0.73137921,  0.16089802,  0.47892163,  0.03639245,  0.67609311,
        1.00116578,  0.49894598,  0.89647668,  0.15296098,  0.04353304,
        0.27883838,  0.99482164,  0.60209317,  0.85654493,  0.3225884 ,
```

```
[26] regr.score(X_test,y_test)
```

```
0.4015221629293929
```

Figure 23: Linear regression of the xtest

4. MODELS EVALUATION AND PREDICTIONS

4.1 CONFUSION MATRIX WITH LOGISTIC CLASSIFIER

Confusion matrices are a popular tool for attempting to solve classification problems. It can perform both binary classification and multiclass classification problems.

```
[37]  # 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: 0.7936507936507936
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]
```

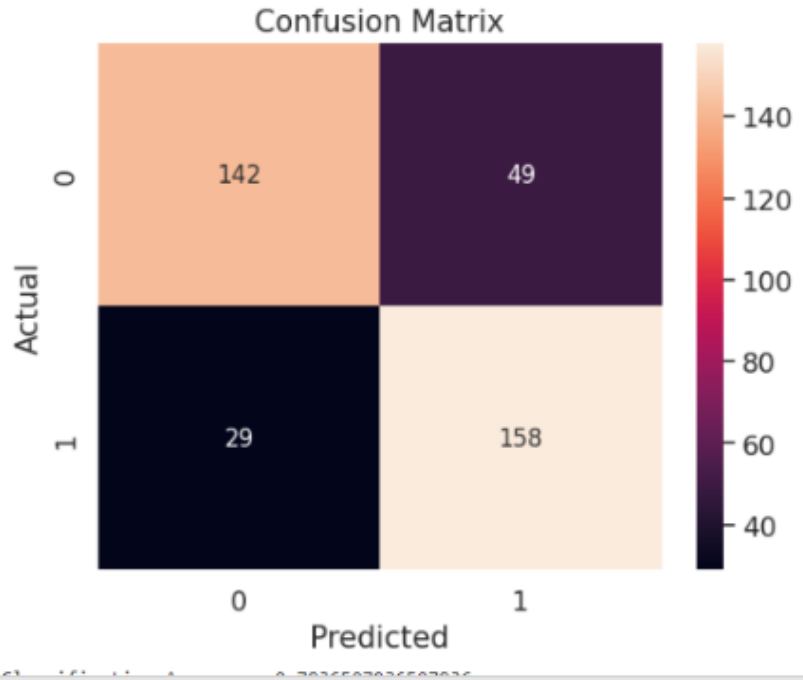

Figure 24: Confusion Accuracy Matrix

```
Classification Accuracy: 0.7936507936507936
Precision: 0.7632850241545893
First 10 predicted responses:
 [1 0 0 0 1 1 0 1 0 1]
First 10 predicted probabilities:
 [[0.90820879]
 [0.03889938]
 [0.03444655]
 [0.20813197]
 [0.60329714]
 [0.95056897]
 [0.24287247]
 [0.80564385]
 [0.38069155]
 [0.53320454]]
```

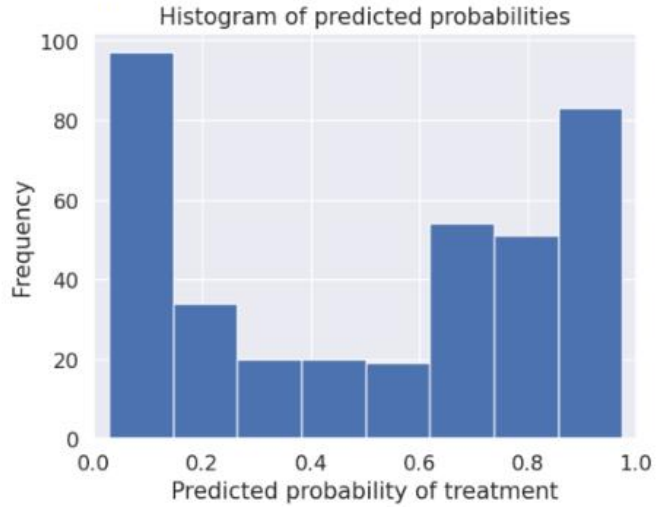

Figure 25: Histogram of the prediction

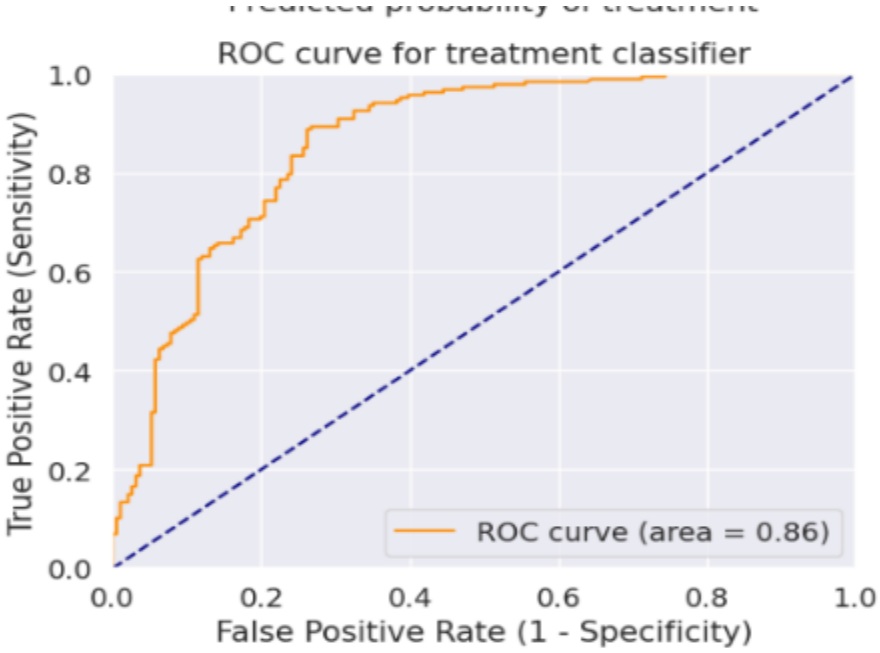

Figure 26: ROC Curve for the classifier

4.2 KNEIGHBORS CLASSIFIER WITH CROSS VALIDATION SCORE TUNING WITH CROSS VALIDATION SCORE

Tuning a model with cross-validation is a common practice in machine learning to improve the model's performance and avoid overfitting. Cross-validation is a technique where the data is split into multiple folds, and the model is trained and validated on each fold. This procedure aids in assessing the effectiveness of the model across various data subsets and delivers a more reliable estimate of its performance

```
[27] #from sklearn.linear_model import KNeighborsClassifier
     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()
```

Figure 27: Tuning the KNN

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

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

```
Knn()
```

```
Rand. Best Score:  0.7954666666666667
Rand. Best Params:  {'weights': 'uniform', 'n_neighbors': 27}
[0.794, 0.799, 0.799, 0.799, 0.799, 0.798, 0.797, 0.795, 0.795, 0.797, 0.797, 0.798, 0.794, 0.797, 0.797, 0.799, 0.798, 0.794, 0.793, 0.794]
########## KNeighborsClassifier ##############
Accuracy: 0.7619047619047619
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 1]
```

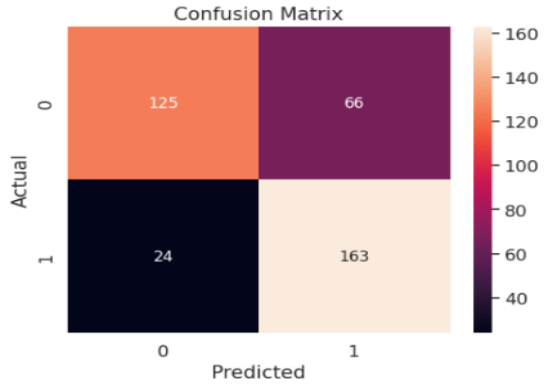

Figure 28: KNN prediction with confusion matrix

```
Classification Accuracy: 0.7619047619047619
Precision: 0.7117903930131004
First 10 predicted responses:
 [1 0 0 0 1 1 0 1 0 1]
First 10 predicted probabilities:
 [[0.77777778]
 [0.07407407]
 [0.03703704]
 [0.2962963 ]
 [0.62962963]
 [0.85185185]
 [0.33333333]
 [0.59259259]
 [0.40740741]
 [0.62962963]]
```

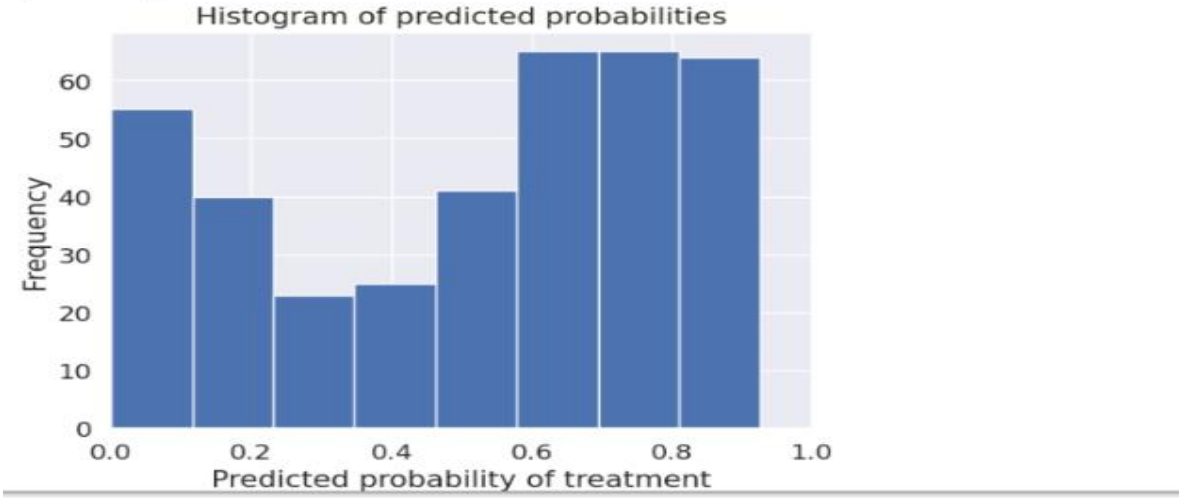

Figure 29: Histogram of the KNN classifier with predicted values

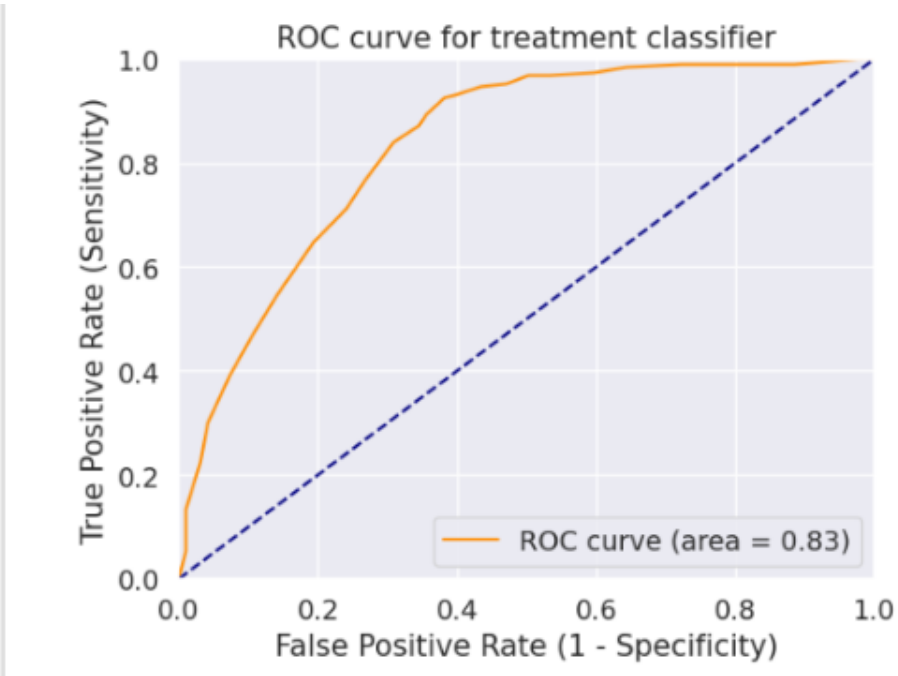

Figure 30: ROC Curve for the KNN classifier

4.3 DECISION TREE CLASSIFIER

TRAINING THE DECISION TREE CLASSIFIER

Tuning a Decision Tree Classifier is an essential step in optimizing the model's performance. The hyperparameters of the decision tree model that can be tuned include:

- Maximum depth of the tree (max_depth): The maximum depth of the decision tree, which limits the number of levels of nodes in the tree.
- Minimum number of samples required to split an internal node (min_samples_split): This refers to the smallest quantity of samples that must be present for a node to be divided in the decision tree.
- Minimum number of samples required to be at a leaf node (min_samples_leaf): The minimum number of samples required to be at a leaf node in the decision tree.
- Maximum number of leaf nodes in the tree (max_leaf_nodes): The maximum number of leaf nodes allowed in the decision tree.

```
[52] 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"]}


        # 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
```

Figure 31: Training of the Tree Classifier

```
treeClassifier()

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

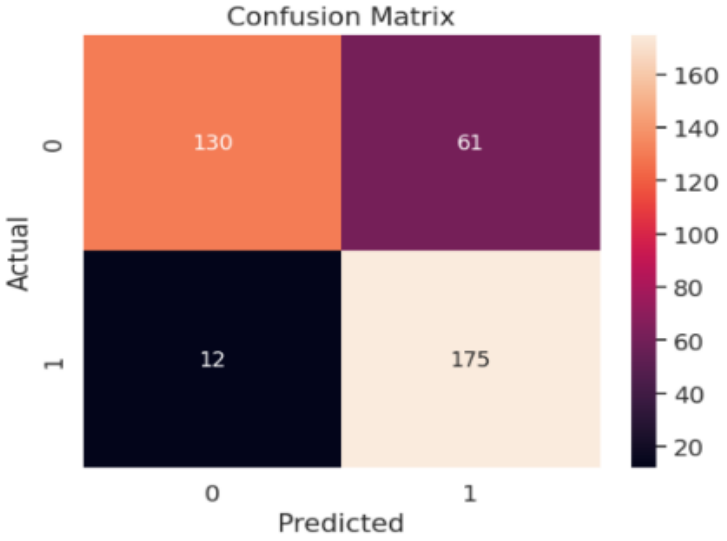

Figure 32: Decision Tree prediction with confusion matrix

```
Classification Accuracy: 0.8068783068783069
Precision: 0.7415254237288136
First 10 predicted responses:
 [1 0 0 0 1 1 0 1 1 1]
First 10 predicted probabilities:
 [[0.82      ]
 [0.         ]
 [0.01030928]
 [0.1221374 ]
 [0.63030303]
 [0.82      ]
 [0.1221374 ]
 [0.63030303]
 [0.79569892]
 [0.79569892]]
```

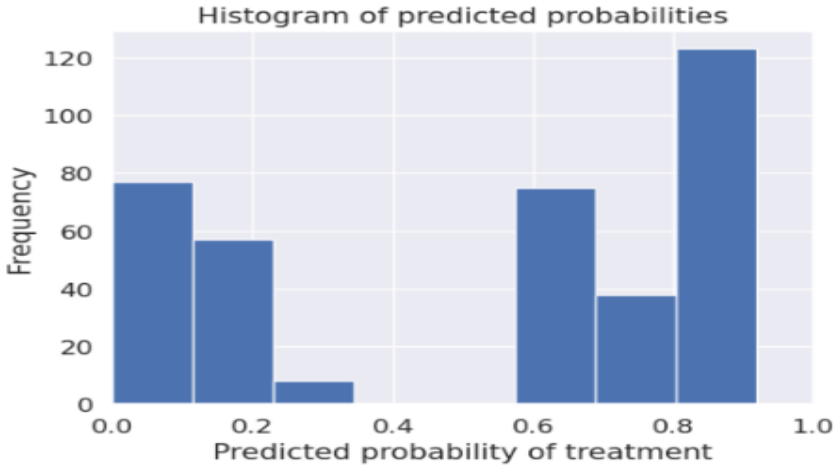

Figure 33: Histogram Predictions of the Tree Classifier

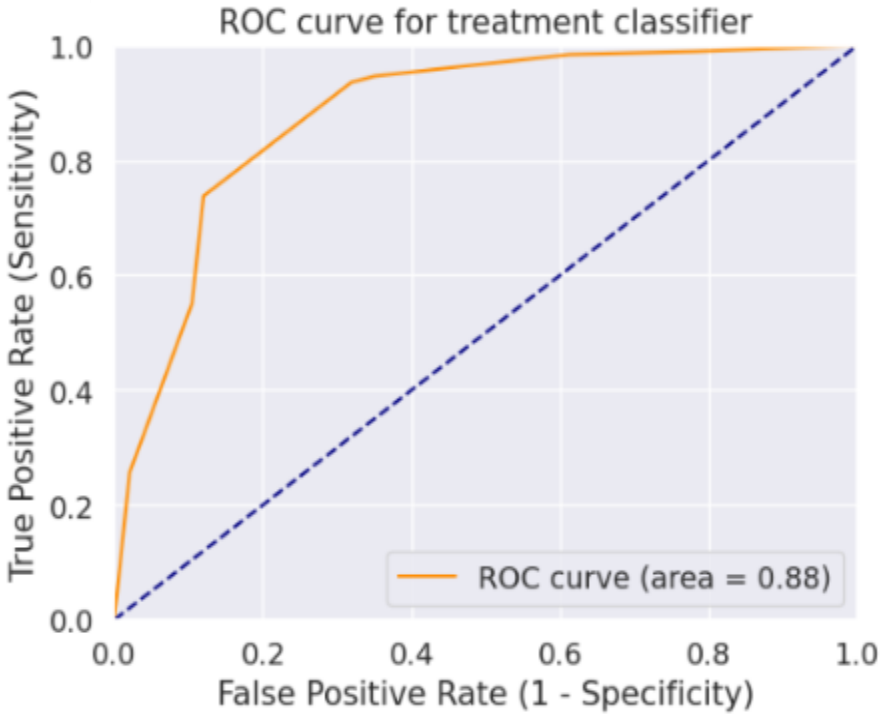

Figure 34: ROC Curve for the Tree Classifier

5. MODEL COMPARISON

5.1 STOCHASTIC GRADIENT DESCENT (SGD)

SGDC (Stochastic Gradient Descent Classifier) is a linear classifier that is part of the scikit-learn library. It is a modification of the classic gradient descent algorithm and uses stochastic gradient descent to optimize a linear model with a hinge loss (linear SVM) or log loss (logistic regression). SGDC Classifier has several hyperparameters that can be tuned to optimize its performance. Some of these hyperparameters include:

- **penalty**: The regularization term used to prevent overfitting. It can be set to 'l1', 'l2', 'elasticnet', or 'none'.
- **alpha**: The regularization strength, which controls the amount of shrinkage applied to the weights. Smaller values of alpha lead to less regularization and a more complex model, while larger values lead to more regularization and a simpler model.
- **loss**: The loss function used to optimize the model. It can be set to 'hinge' for linear SVM or 'log' for logistic regression.

- **max_iter**: The maximum number of iterations allowed for the optimization algorithm.

```
[54] from sklearn.linear_model import SGDClassifier
     sgdc = SGDClassifier(loss="log", max_iter=10000,penalty='l1',random_state=1,alpha=0.0001).fit(X_train,y_train)

[56] y_pred = sgdc.predict(X_test)
     y_pred

     array([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, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
            1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1,
            0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1,
            1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0,
            0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0,
            0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1,
            1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1,
            0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0,
            1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0,
            1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1,
            1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1,
            0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0,
            1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0,
            1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1,
            1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1,
            1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1,
            0, 1, 0, 1])

[57] sgdc.score(X_train, y_train)

     0.8100113765642776

[58] sgdc.score(X_test,y_test)

     0.798941798941799
```

Figure 35: SGDC Classifier score

```
from sklearn.model_selection import GridSearchCV

params = {'max_iter':[1000,10000,20000],'penalty':['l1','l2','elasticnet'],'alpha':[0.0001,0.005,0.001]}

grid_search = GridSearchCV(sgdc,params,cv=5)

grid_search.fit(X_train,y_train)
```

GridSearchCV
estimator: SGDClassifier
SGDClassifier
SGDClassifier(loss='log', max_iter=10000, penalty='l1', random_state=1)

Figure 36: SGDC prediction with Grid Search CV

6. CONCLUSION

In conclusion, data visualization has numerous possible applications in various fields, but we must also consider the practical and ethical challenges it poses. We have discussed essential theoretical and practical guidelines for creating data visualizations and evaluated several visualization examples to identify common

mistakes and helpful tips. Developing a trustworthy and ethical data visualization can be a complex process, and in many cases, automating data mining using Python can be a useful and efficient solution. Popular Python data mining techniques include association rules, clustering, regression, and classification.

Data visualization is a critical aspect of machine learning, helping to explore and analyze datasets, communicate insights and findings, and evaluate model performance. Effective data visualization can enhance the accuracy and interpretability of machine learning models, allowing stakeholders to make informed decisions based on data-driven insights.

However, creating impactful and ethical data visualizations requires careful consideration of various theoretical and practical guidelines, as well as potential biases and ethical implications. Additionally, automating data mining using Python can be a valuable approach to enhance the efficiency and accuracy of data visualization and analysis, particularly for complex datasets.

In summary, data visualization is a vital tool for machine learning, and by following best practices and ethical principles, we can maximize its potential to derive insights and improve decision-making in various domains.

7. REFERENCES

[1] "Kaggle: Your Home for Data Science." https://www.kaggle.com/ (accessed Apr. 27, 2023).

[2] Song, C., Ristenpart, T. and Shmatikov, V. (2017) 'Machine Learning Models that Remember Too Much', in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. New York, NY, USA: Association for Computing Machinery (CCS '17), pp. 587–601. Available at: https://doi.org/10.1145/3133956.3134077.

[3] How to develop machine learning models for healthcare | Nature Materials. Available at: https://www.nature.com/articles/s41563-019-0345-0 (Accessed: 27 April 2023).

[4] Machine Learning Techniques for Prediction of Mental Health | IEEE Conference Publication | IEEE Xplore. Available at: https://ieeexplore.ieee.org/abstract/document/9545061 (Accessed: 27 April 2023).

[5] Srividya, M., Mohanavalli, S. and Bhalaji, N. (2018) 'Behavioral Modeling for Mental Health using Machine Learning Algorithms', Journal of Medical Systems, 42(5), p. 88. Available at: https://doi.org/10.1007/s10916-018-0934-5.

[6] Mohanapriya, M. and Lekha, J. (2018) 'Comparative study between decision tree and knn of data mining classification technique', Journal of Physics: Conference Series, 1142(1), p. 012011. Available at: https://doi.org/10.1088/17426596/1142/1/012011.

[7] A simple method of sample size calculation for linear and logistic regression - Hsieh - 1998 - Statistics in Medicine - Wiley Online Library (no date). Available at: https://onlinelibrary.wiley.com/doi/abs/10.1002/(SICI)10970258(19980730)17:14%3C1623::AID-SIM871%3E3.0.CO;2-S (Accessed: 27 April 2023).

8. APPENDIX

```
# 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

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

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

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

# Any results you write to the current directory are saved as output.

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

```
#Pandas: whats the data row count?
print(train_df.shape)
```

```
[2] train_df.columns
```

```
[ ] train_df.tail()
```

```
[ ] train_df.shape
```

```
[ ] train_df.info
```

```
[ ] train_df.describe()
```

```
[ ] train_df.nunique()
```

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

```
[5]  #dealing with missing data
     #Let's get rid of the variables "Timestamp",“comments”, “state”.
     train_df = train_df.drop(['comments'], axis= 1)
     train_df = train_df.drop(['state'], axis= 1)
     train_df = train_df.drop(['Timestamp'], axis= 1)

     train_df.isnull().sum().max() #just checking that there's no missing data missing...
     train_df.head(5)
```

```
[6]  # 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 = []
```

```
[6]  # Clean the NaN's
     for feature in train_df:
         if feature in intFeatures:
             train_df[feature] = train_df[feature].fillna(defaultInt)
         elif feature in stringFeatures:
             train_df[feature] = train_df[feature].fillna(defaultString)
         elif feature in floatFeatures:
             train_df[feature] = train_df[feature].fillna(defaultFloat)
         else:
             print('Error: Feature %s not recognized.' % feature)
     train_df.head(5)
```

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

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

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

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

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


```
[7] for (row, col) in train_df.iterrows():

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

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

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

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

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

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

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

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

```
[9] #There are only few 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())
```

```
[10] #There are only few of self work_interfere so let's change NaN to "Don't know
     #Replace "NaN" string from defaultString

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

```
[11] from sklearn import preprocessing

     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)
```


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

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

```
[13] #correlation matrix
     corrmat = train_df.corr()
     f, ax = plt.subplots(figsize=(6, 6))
     sns.heatmap(corrmat, vmax=.8, square=True);
     plt.show()
```

```
[14] #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()
```

```
[61] sn.pairplot(train_df)
```

```
[15] sn.relplot(x='Age', y='work_interfere', hue ='treatment', data=train_df)
```

```
[16] plt.figure(figsize=(6,6))
     sn.distplot(train_df["Age"], bins=24)
     plt.title("Distribution and density by age")
     plt.xlabel("Age")
```

```
[17] # Separate by treatment or not
     plt.figure(figsize=(8,8))
     g = sn.FacetGrid(train_df, col='treatment')
     g = g.map(sn.distplot, "Age")
```

```
[ ]  o = labelDict['label_age_range']

     g = sn.catplot(x="age_range", y="treatment", hue="Gender", data=train_df, kind="bar",  ci=None, 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()
```

```
[18] #Splitting of the data set
     # define X and y
     feature_cols = ['Age', 'Gender', 'family_history', 'benefits', 'care_options', 'anonymity', 'leave', 'work_interfere
     X = train_df[feature_cols]
     y = train_df.treatment

     # split X and y into training and testing sets
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=0)

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

```
[40] logistic_regression= LogisticRegression(multi_class='multinomial', solver='lbfgs')
     logistic_regression.fit(X_train,y_train)
     y_pred=logistic_regression.predict(X_test)
     #print(X)
     print (X_test) #test dataset
     print (y_pred) #predicted values
```

```
[19] from sklearn.preprocessing import binarize
     from sklearn.metrics import roc_auc_score
     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))

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


```
accuracy = metrics.accuracy_score(y_test, y_pred_class)
print('Classification Accuracy:', accuracy)


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

########################################
#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
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)
```


```
        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')

        y_pred_prob = y_pred_prob.reshape(-1,1)

    # print the first 10 predicted probabilities
        print('First 10 predicted probabilities:\n', y_pred_prob[0:10])
    roc_auc = metrics.roc_auc_score(y_test, y_pred_prob)
    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()

    return accuracy
```


```
[20] from sklearn.model_selection import cross_val_score
     from sklearn.preprocessing import binarize
     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
```

```
[21] logisticRegression()
```

```
[23] from sklearn.linear_model import LinearRegression
     regr=LinearRegression()
     regr.fit(X_train,y_train)
```

```
[25] regr.predict(y_test)
```


```
[43] regr.score(X_test,y_test)
```

```
[27] #from sklearn.linear_model import KNeighborsClassifier
     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()
```

```
[33] from sklearn.neighbors import KNeighborsClassifier
     from sklearn.model_selection import RandomizedSearchCV
     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']
```


```
[33]
# 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
```

```
[34] Knn()
```

```
[52] 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"]}


        # 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
```

```
[53] treeClassifier()
```


```
[54] from sklearn.linear_model import SGDClassifier
     sgdc = SGDClassifier(loss="log", max_iter=10000,penalty='l1',random_state=1,alpha=0.0001).fit(X_train,y_train)
```

```
[56] y_pred = sgdc.predict(X_test)
     y_pred
```

```
[57] sgdc.score(X_train, y_train)
```

```
[58] sgdc.score(X_test,y_test)
```

```
[59] from sklearn.model_selection import GridSearchCV

     params = {'max_iter':[1000,10000,20000],'penalty':['l1','l2','elasticnet'],'alpha':[0.0001,0.005,0.001]}

     grid_search = GridSearchCV(sgdc,params,cv=5)

     grid_search.fit(X_train,y_train)
```

9. COLAB LINK

https://colab.research.google.com/drive/1Fb579KUbnQj8MQ8EBhcRZhReQEd_06_R?usp=sharing