Mental Health Prediction Using Machine Learning

Group 2
Krishnaprasad V M
Rohit Pal
Manisha Mahor

INTRODUCTION

Technology is evolving round the clock in recent times. This has resulted in job opportunities for people all around the world. It comes with a hectic schedule that can be detrimental to people’s mental health. So During the Covid-19 pandemic, mental health has been one of the most prominent issues, with stress, loneliness, and depression all on the rise over the last year. Diagnosing mental health is difficult because people aren’t always willing to talk about their problems.

Machine learning is a branch of artificial intelligence that is mostly used nowadays. ML is becoming more capable for disease diagnosis and also provides a platform for doctors to analyze a large number of patient data and create personalized treatment according to the patient’s medical situation. In this article, we are going to predict the mental health of Employees using various machine learning models. We have collected the required data set from Kaggle.

The process is the following:

1. Library and data loading
2. Data cleaning
3. Encoding data
4. Covariance Matrix. Variability comparison between categories of variables
5. Data Visualisation
6. Scaling and fitting
7. Splitting the data set
8. Evaluating models
    A. Logistic Regression
    B. KNN Algorithm
    C. Decision Tree Classifier
    D. Random Forests
    E. AdaBoost Classifier
9. Success method plot
10. Creating predictions on test set
11. Submission
12. Conclusions

Library and data loading

Imported several libraries for data analysis and machine learning tasks in Python.

- Pandas is used for data manipulation and analysis.
- NumPy is used for numerical computing and scientific computing.
- Matplotlib and Seaborn are used for data visualization and plotting.
- Scipy is used for scientific computing and statistical analysis.
- Sklearn (Scikit-learn) is a machine learning library in Python, and it is used for data preparation, modeling, and validation.
- The specific machine learning models included in this code are: Logistic Regression, Decision Tree Classifier, Random Forest Classifier, Extra Trees Classifier, MLP Classifier, K-Nearest Neighbors Classifier, AdaBoost Classifier, and Gaussian Naive Bayes.

In [22]:

```python
#importing the libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from scipy.stats import randint
#preparation
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.datasets import make_classification
from sklearn.preprocessing import binarize, LabelEncoder, MinMaxScaler
#models
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
#validation libraries
from sklearn import metrics
from sklearn.metrics import accuracy_score, mean_squared_error, precision_recall_curve
from sklearn.model_selection import cross_val_score
#neural network
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import learning_curve,GridSearchCV
from sklearn.model_selection import LeaveOneOut
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier
#Naive bayes
from sklearn.naive_bayes import GaussianNB
```

Data Cleaning

Data cleaning is a crucial step in data analysis, and Pandas is a popular Python library that offers powerful tools for data cleaning. Here are some common data cleaning tasks that we are used in our project.

1. Removing unnecessary columns: Use the `drop()` method to remove unnecessary columns, or use indexing to select the desired columns.
2. Handling missing values:Use the `fillna()` method to fill missing values with a specified value or method.
3. Handling string data: Use string methods such as `str.replace()` to clean information from string data.

In [26]:
```python
#cleaning data
#removing columns: Timestamp,state and comments
df=df.drop(['Timestamp'], axis = 1)
df=df.drop(['state'], axis = 1)
df=df.drop(['comments'], axis = 1)
df = df.drop(['Country'], axis= 1)
```

```python
#Assigning default values for each data type

defaultInt=0
defaultFloat=0.0
defaultString='NaN'

#creating list by data type

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 df:
    if feature in intFeatures:
        df[feature] = df[feature].fillna(defaultInt)
    elif feature in stringFeatures:
        df[feature] = df[feature].fillna(defaultString)
    elif feature in floatFeatures:
        df[feature] = df[feature].fillna(defaultFloat)
    else:
        print('Error: Feature %s not recognized.' % feature)
pd.set_option('display.max_columns', None)
df.head()
```

Encoding data

Encoding is a technique used to convert categorical data into numerical data. It is useful when working with machine learning algorithms that can only handle numerical data. LabelEncoder() function in scikit-learn is a convenient way to perform label encoding in Python.

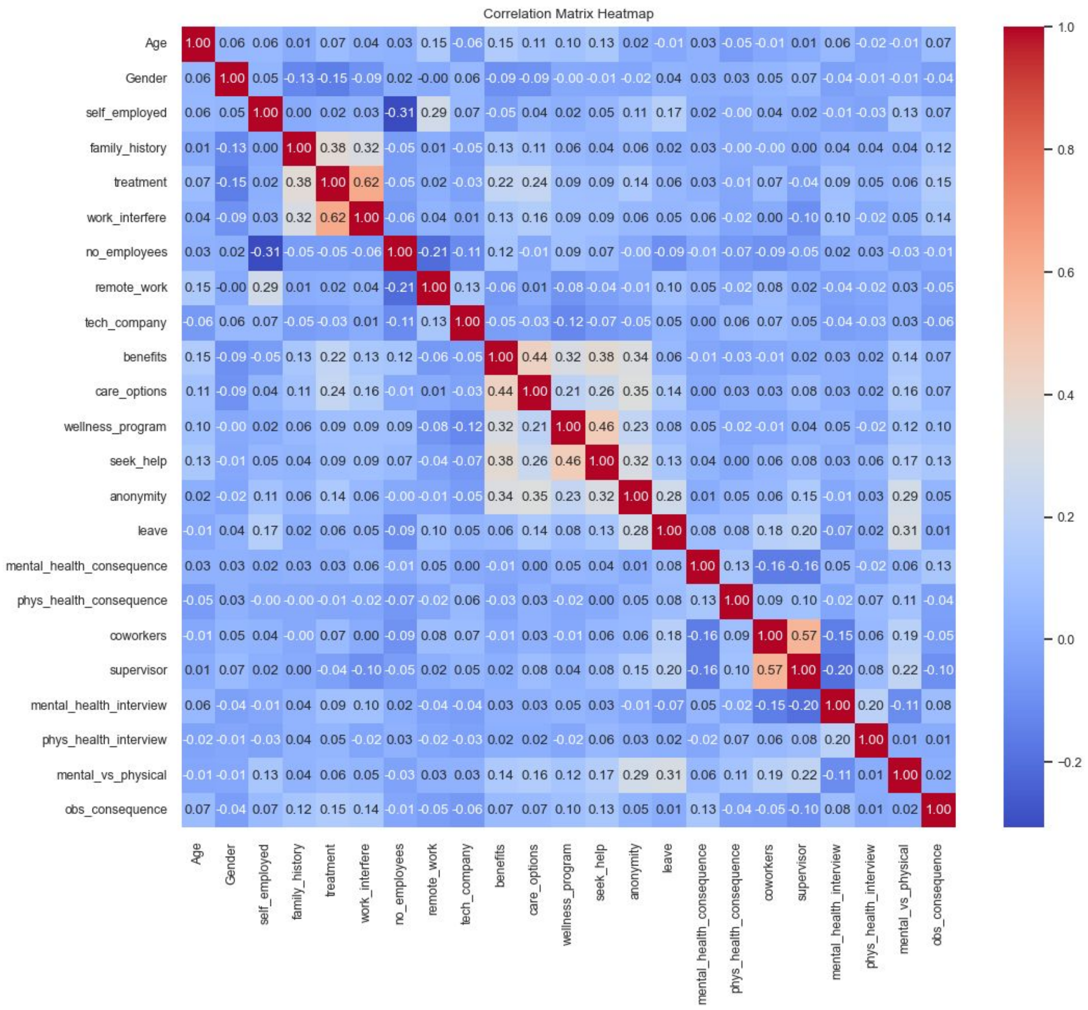
So, in this project we were used Label Encoder function for converting the categorical data into numerical data.

3. Encoding the data ¶

```
le = LabelEncoder()
# Loop over each column in the dataframe and apply the label encoder
for col in df.columns:
    if df[col].dtype == 'object': # check if the column contains categorical data
        df[col] = le.fit_transform(df[col])
```

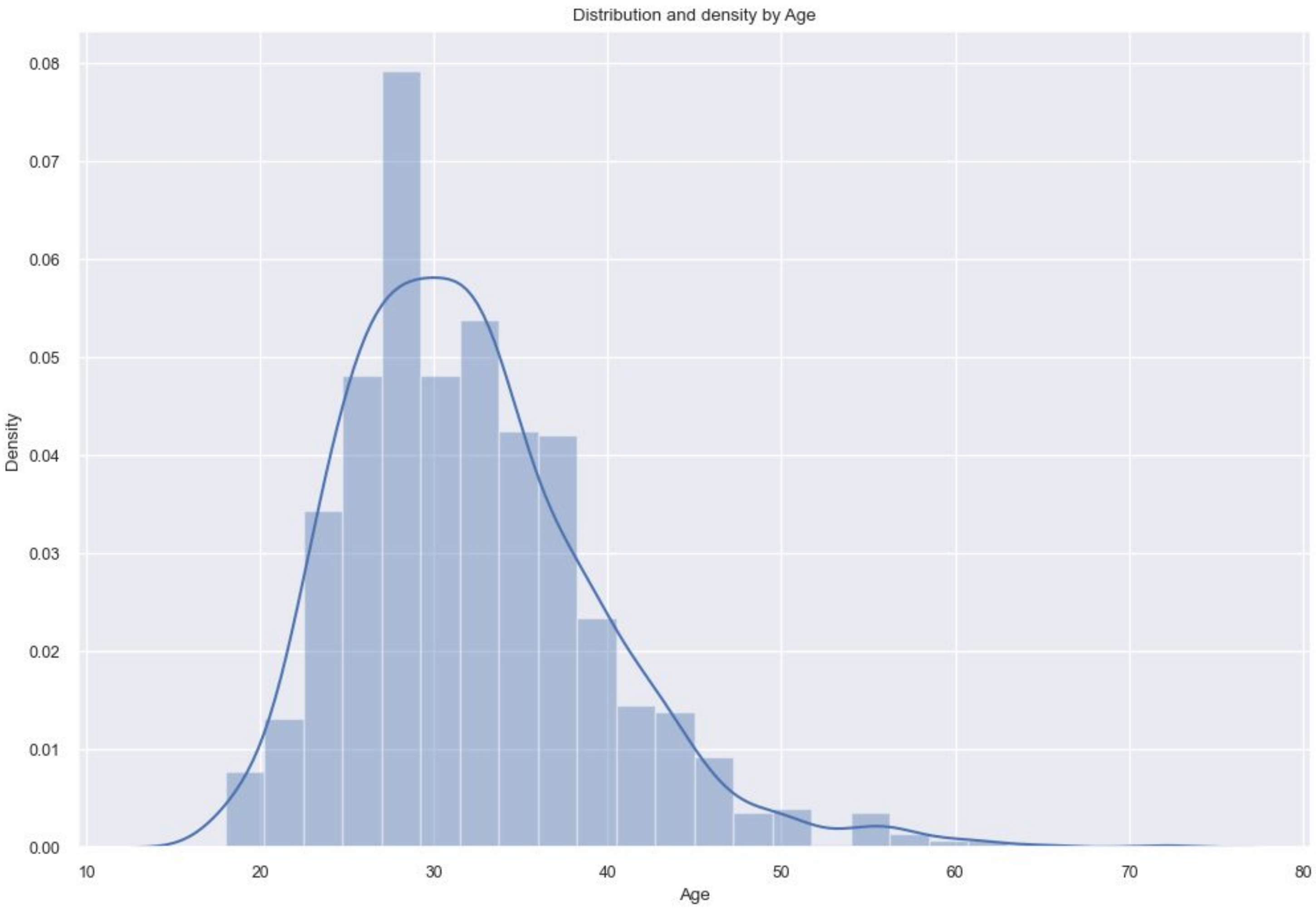
Covariance Matrix. Variability comparison between categories of variables

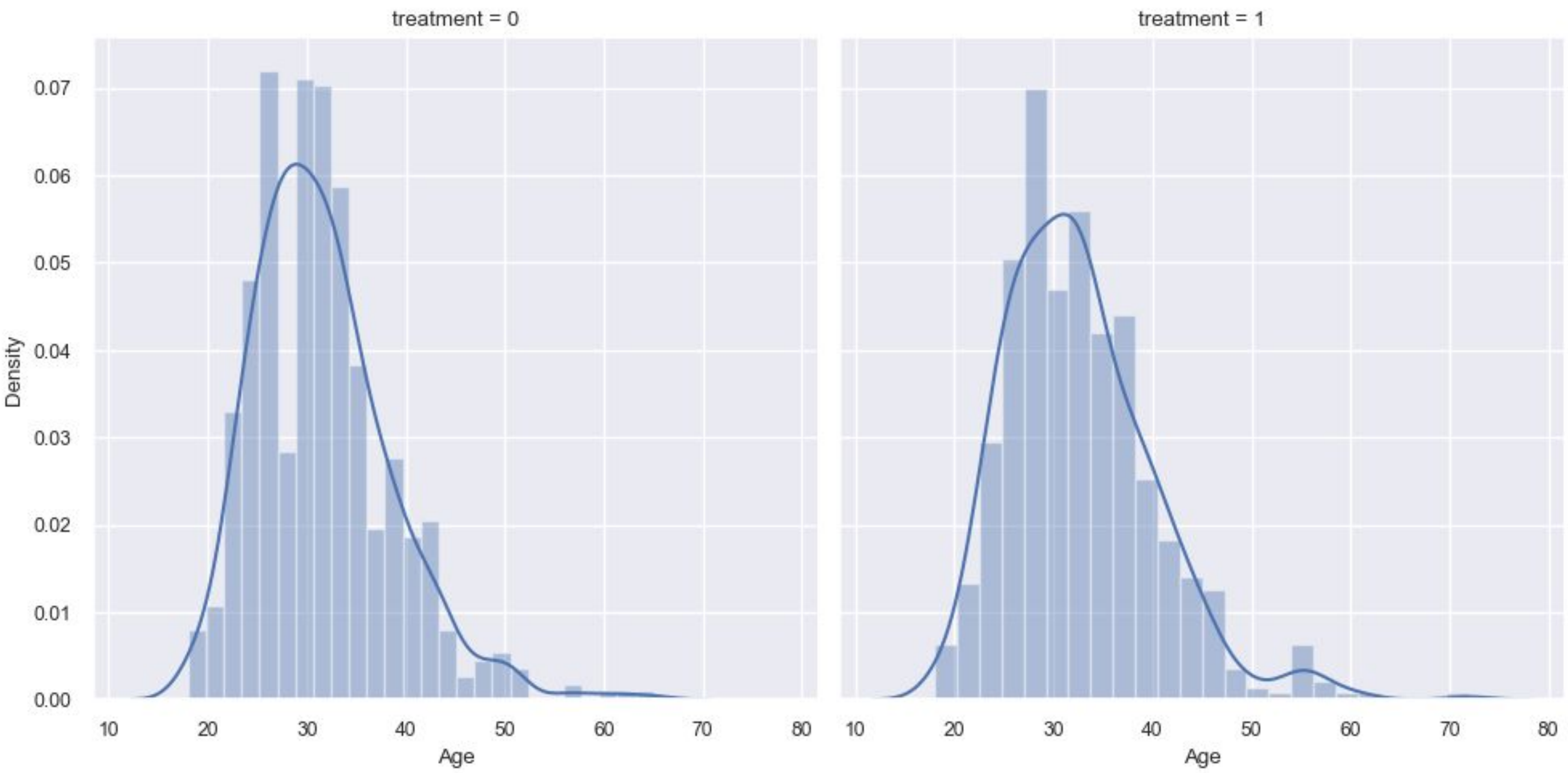
A covariance matrix is a square matrix that shows the covariances between two or more variables. A covariance matrix is useful for examining the relationships between multiple variables. It can be used to compare the variability between different categories of variables.

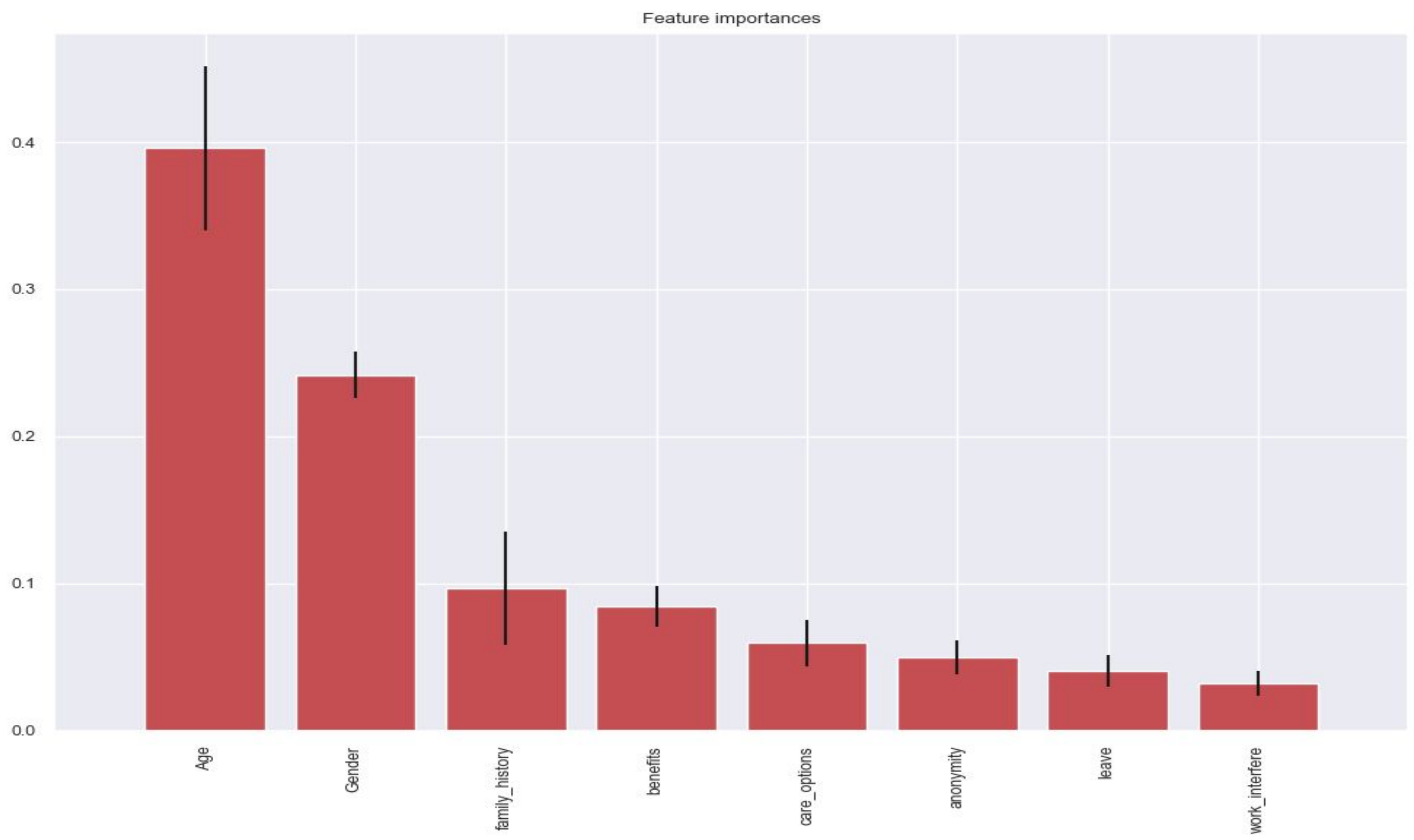

Correlation Matrix Heatmap
Age 1.00 0.06 0.06 0.01 0.07 0.04 0.03 0.15 -0.06 0.15 0.11 0.10 0.13 0.02 -0.01 0.03 -0.05 -0.01 0.01 0.06 -0.02 -0.01 0.07
Gender 0.06 1.00 0.05 -0.13 -0.15 -0.09 0.02 -0.00 0.06 -0.09 -0.09 -0.00 -0.01 -0.02 0.04 0.03 0.03 0.05 0.07 -0.04 -0.01 -0.01 -0.04
self_employed 0.06 0.05 1.00 0.00 0.02 0.03 -0.31 0.29 0.07 -0.05 0.04 0.02 0.05 0.11 0.17 0.02 -0.00 0.04 0.02 -0.01 -0.03 0.13 0.07
family_history 0.01 -0.13 0.00 1.00 0.38 0.32 -0.05 0.01 -0.05 0.13 0.11 0.06 0.04 0.06 0.02 0.03 -0.00 -0.00 0.00 0.04 0.04 0.04 0.12
treatment 0.07 -0.15 0.02 0.38 1.00 0.62 -0.05 0.02 -0.03 0.22 0.24 0.09 0.09 0.14 0.06 0.03 -0.01 0.07 -0.04 0.09 0.05 0.06 0.15
work_interfere 0.04 -0.09 0.03 0.32 0.62 1.00 -0.06 0.04 0.01 0.13 0.16 0.09 0.09 0.06 0.05 0.06 -0.02 0.00 -0.10 0.10 -0.02 0.05 0.14
no_employees 0.03 0.02 -0.31 -0.05 -0.05 -0.06 1.00 -0.21 -0.11 0.12 -0.01 0.09 0.07 -0.00 -0.09 -0.01 -0.07 -0.09 -0.05 0.02 0.03 -0.03 -0.01
remote_work 0.15 -0.00 0.29 0.01 0.02 0.04 -0.21 1.00 0.13 -0.06 0.01 -0.08 -0.04 -0.01 0.10 0.05 -0.02 0.08 0.02 -0.04 -0.02 0.03 -0.05
tech_company -0.06 0.06 0.07 -0.05 -0.03 0.01 -0.11 0.13 1.00 -0.05 -0.03 -0.12 -0.07 -0.05 0.05 0.00 0.06 0.07 0.05 -0.04 -0.03 0.03 -0.06
benefits 0.15 -0.09 -0.05 0.13 0.22 0.13 0.12 -0.06 -0.05 1.00 0.44 0.32 0.38 0.34 0.06 -0.01 -0.03 -0.01 0.02 0.03 0.02 0.14 0.07
care_options 0.11 -0.09 0.04 0.11 0.24 0.16 -0.01 0.01 -0.03 0.44 1.00 0.21 0.26 0.35 0.14 0.00 0.03 0.03 0.08 0.03 0.02 0.16 0.07
wellness_program 0.10 -0.00 0.02 0.06 0.09 0.09 0.09 -0.08 -0.12 0.32 0.21 1.00 0.46 0.23 0.08 0.05 -0.02 -0.01 0.04 0.05 -0.02 0.12 0.10
seek_help 0.13 -0.01 0.05 0.04 0.09 0.09 0.07 -0.04 -0.07 0.38 0.26 0.46 1.00 0.32 0.13 0.04 0.00 0.06 0.08 0.03 0.06 0.17 0.13
anonymity 0.02 -0.02 0.11 0.06 0.14 0.06 -0.00 -0.01 -0.05 0.34 0.35 0.23 0.32 1.00 0.28 0.01 0.05 0.06 0.15 -0.01 0.03 0.29 0.05
leave -0.01 0.04 0.17 0.02 0.06 0.05 -0.09 0.10 0.05 0.06 0.14 0.08 0.13 0.28 1.00 0.08 0.08 0.18 0.20 -0.07 0.02 0.31 0.01
mental_health_consequence 0.03 0.03 0.02 0.03 0.03 0.06 -0.01 0.05 0.00 -0.01 0.00 0.05 0.04 0.01 0.08 1.00 0.13 -0.16 -0.16 0.05 -0.02 0.06 0.13
phys_health_consequence -0.05 0.03 -0.00 -0.00 -0.01 -0.02 -0.07 -0.02 0.06 -0.03 0.03 -0.02 0.00 0.05 0.08 0.13 1.00 0.09 0.10 -0.02 0.07 0.11 -0.04
coworkers -0.01 0.05 0.04 -0.00 0.07 0.00 -0.09 0.08 0.07 -0.01 0.03 -0.01 0.06 0.06 0.18 -0.16 0.09 1.00 0.57 -0.15 0.06 0.19 -0.05
supervisor 0.01 0.07 0.02 0.00 -0.04 -0.10 -0.05 0.02 0.05 0.02 0.08 0.04 0.08 0.15 0.20 -0.16 0.10 0.57 1.00 -0.20 0.08 0.22 -0.10
mental_health_interview 0.06 -0.04 -0.01 0.04 0.09 0.10 0.02 -0.04 -0.04 0.03 0.03 0.05 0.03 -0.01 -0.07 0.05 -0.02 -0.15 -0.20 1.00 0.20 -0.11 0.08
phys_health_interview -0.02 -0.01 -0.03 0.04 0.05 -0.02 0.03 -0.02 -0.03 0.02 0.02 -0.02 0.06 0.03 0.02 -0.02 0.07 0.06 0.08 0.20 1.00 0.01 0.01
mental_vs_physical -0.01 -0.01 0.13 0.04 0.06 0.05 -0.03 0.03 0.03 0.14 0.16 0.12 0.17 0.29 0.31 0.06 0.11 0.19 0.22 -0.11 0.01 1.00 0.02
obs_consequence 0.07 -0.04 0.07 0.12 0.15 0.14 -0.01 -0.05 -0.06 0.07 0.07 0.10 0.13 0.05 0.01 0.13 -0.04 -0.05 -0.10 0.08 0.01 0.02 1.00
Age Gender self_employed family_history treatment work_interfere no_employees remote_work tech_company benefits care_options wellness_program seek_help anonymity leave mental_health_consequence phys_health_consequence coworkers supervisor mental_health_interview phys_health_interview mental_vs_physical obs_consequence
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

Data Visualisation

The resulting plot shows the distribution of ages in the dataset as a histogram with a density curve overlaid on top. The histogram shows the frequency of ages in each bin, while the density curve shows the probability density function of the ages. This plot can be used to gain insights into the age distribution of the dataset, such as identifying any peaks or clusters in the distribution, or whether the distribution is skewed or symmetrical.


Distribution and density by Age
Density
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
10
20
30
40
50
60
70
80
Age


treatment = 0
treatment = 1
Density
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
10
20
30
40
50
60
70
80
Age
Age


Feature importances
0.4
0.3
0.2
0.1
0.0
Age
Gender
family_history
benefits
care_options
anonymity
leave
work_interfere

Scaling and fitting

Scaling:

- Many machine learning models require that input features be on the same scale. If the features are on different scales, then some features may dominate others in terms of their influence on the model. Scaling ensures that all features have the same scale and therefore the same influence on the model.
- Scaling also helps to avoid numerical instabilities that can occur when features are on vastly different scales, which can negatively impact the model's accuracy.

Fitting:

- Fitting is the process of calculating the optimal parameters of a model so that it can accurately predict unseen data. For example, in the case of the `MinMaxScaler` used in the code you provided, fitting is the process of calculating the minimum and maximum values of the 'Age' column so that it can be scaled properly.
- In many machine learning models, the process of fitting involves training the model on a subset of the data and then evaluating its performance on a separate validation set. This helps to ensure that the model is not overfitting to the training data (i.e., memorizing the training data instead of learning to generalize to new data).

6. Scaling and fitting

In [40]:
```
#scaling the age
scaler = MinMaxScaler()
df['Age'] = scaler.fit_transform(df[['Age']])
df.head()
```

Out[40]:

	Age	Gender	self_employed	family_history	treatment	work_interfere	no_employees	remote_work	tech_company	benefits	care_options	wellness_pr
0	0.351852	0	0	0	1	2	4	0	1	2	1	
1	0.481481	1	0	0	0	3	5	0	0	0	0	
2	0.259259	1	0	0	0	3	4	0	1	1	0	
3	0.240741	1	0	1	1	2	2	0	1	1	2	
4	0.240741	1	0	0	0	1	1	1	1	2	0	

Splitting the data set

Splitting the dataset is an important step in machine learning, as it helps to evaluate the performance of the model on unseen data. Here's why:

- In machine learning, it is important to evaluate the performance of a model on data that it has not seen before. This helps to ensure that the model is not simply memorizing the training data and can generalize well to new, unseen data.
- Splitting the dataset into training and testing sets allows us to train the model on one set of data (the training set) and evaluate its performance on another set of data (the testing set). This helps to provide an estimate of how well the model will perform on unseen data in the real world.
- In some cases, such as when the dataset is small, it may be useful to use techniques such as cross-validation to split the dataset into multiple training and testing sets. This can help to improve the robustness of the model's performance estimates.

7.Splitting the data set

In [41]:
```
# define X and y
feature_cols = ['Age', 'Gender', 'family_history', 'benefits', 'care_options', 'anonymity', 'leave', 'work_interfere']
X = df[feature_cols]
y = 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)
```

Evaluating models

Logistic Regression: Logistic regression is a statistical method used for binary classification problems, where the goal is to predict a binary outcome (0 or 1) based on a set of input variables or features.

Accuracy of data set by using logistic regression

“Accuracy: 0.7962962962962963”“

8. Evaluating models

A. Logistic Regression

In [44]:
```
# Create a logistic regression model and fit it to the training data
def log_regression():
    model = LogisticRegression()
    model.fit(X_train, y_train)
    # Use the model to make predictions on the test set
    y_pred = model.predict(X_test)
    # Calculate the accuracy of the model
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy: {accuracy}")
```

In [45]:
```
#call logistic regression
log_regression()
```

```
Accuracy: 0.7962962962962963
```

K-Nearest Neighbors (KNN) algorithm: It is a non-parametric and lazy learning algorithm used for both classification and regression problems. It is a type of instance-based learning where the algorithm predicts the class of a new data point based on the classes of its k nearest neighbors in the training dataset.

“Accuracy: 0.8068783068783069”

In [46]:
```
def knn_algo():
    # Create a KNN model with k=5
    knn = KNeighborsClassifier(n_neighbors=5)
    # Train the model using the training data
    knn.fit(X_train, y_train)

    # Predict the target values for the test set
    y_pred = knn.predict(X_test)
    # Calculate the accuracy of the model
    accuracy = accuracy_score(y_test, y_pred)

    # Print the accuracy of the model
    print("Accuracy:", accuracy)
```

In [47]:
```
#call knn alogrithm
knn_algo()
```

```
Accuracy: 0.8068783068783069
```

Decision Tree Classifier: It is a supervised machine learning algorithm that is used for classification problems. It is a tree-based model that recursively splits the data into subsets based on the most significant features, resulting in a tree-like structure. The goal is to create a model that can predict the target variable by learning simple decision rules derived from the data features.

Accuracy: 0.7513227513227513

```
In [48]: def dec_tree_classifier():
             # Create a Decision Tree Classifier
             clf = DecisionTreeClassifier()

             # Train the model using the training data
             clf.fit(X_train, y_train)

             # Predict the target values for the test set
             y_pred = clf.predict(X_test)

             # Calculate the accuracy of the model
             accuracy = accuracy_score(y_test, y_pred)

             # Print the accuracy of the model
             print("Accuracy:", accuracy)
```

```
In [49]: #calling decision tree
         dec_tree_classifier()
```

```
Accuracy: 0.7513227513227513
```

Random forest Classifier: It is an ensemble learning method for classification, regression and other tasks. In random forest, a large number of decision trees are created, and each tree is trained on a random subset of the training data. The output of the random forest is the majority vote of the decision trees. It reduces overfitting and improves the accuracy of the model.

Accuracy: 0.7962962962962963

In [50]:
```python
def random_forest():
    # Create a Random Forest Classifier with 100 trees
    rf = RandomForestClassifier(n_estimators=100)

    # Train the model using the training data
    rf.fit(X_train, y_train)

    # Predict the target values for the test set
    y_pred = rf.predict(X_test)

    # Calculate the accuracy of the model
    accuracy = accuracy_score(y_test, y_pred)

    # Print the accuracy of the model
    print("Accuracy:", accuracy)
```

In [51]:
```python
#call random forest
random_forest()
```

```
Accuracy: 0.7962962962962963
```

AdaBoost Classifier: It is an ensemble learning method that can improve the classification accuracy of weak learners (classifiers that perform only slightly better than random guessing) by combining their predictions. AdaBoost works by iteratively training a sequence of weak learners, where each subsequent learner is trained on the examples that were misclassified by the previous learner. The final prediction is a weighted sum of the predictions made by each learner.

Accuracy: 0.8201058201058201

“Best prediction”

In [54]:
```
def ada_boost():
    # Create the AdaBoost Classifier model with 50 estimators
    ada = AdaBoostClassifier(n_estimators=50, random_state=42)

    # Fit the model to the training data
    ada.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = ada.predict(X_test)

    # Calculate the accuracy of the model
    accuracy = accuracy_score(y_test, y_pred)

    # Print the accuracy of the model
    print(f"Accuracy: {accuracy}")
```

In [56]:
```
#calling ada boost classifier
ada_boost()
```

```
Accuracy: 0.8201058201058201
```

Creating Prediction on test data set using AdaBoost Classifier

We have predicted the test results of given data by using the best method “Ada Boost Classifier”

The resultant data is then stored on another csv file, name: “submission”

The final prediction consists of 0 and 1. 0 means the person is not needed any mental health treatment and 1 means the person is needed mental health treatment.

In [52]:

```
# 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()
```

Out[52]:

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

jupyter submission.csv 2 hours ago

Logout

File Edit View Language current mode

```
Index,Treatment
5,1
494,0
52,0
984,0
186,0
18,1
317,0
511,1
364,1
571,1
609,0
1147,1
922,1
461,0
740,1
955,1
814,1
1160,1
85,0
733,0
1112,0
124,0
1040,1
492,0
1159,0
211,1
1020,0
892,0
453,0
```

Index	Treatment
5	1
494	0
52	0
984	0
186	0
18	1
317	0
511	1
364	1
571	1
609	0
1147	1
922	1
461	0
740	1
955	1
814	1
1160	1
85	0
733	0
1112	0
124	0
1040	1
492	0
1159	0
211	1
1020	0
892	0

results (1)

Conclusions

After using all these Employee records, we are able to build various machine learning models. From all the models, ADA–Boost achieved 82% accuracy along with that we were able to draw some insights from the data via data analysis and visualization.


Thank you