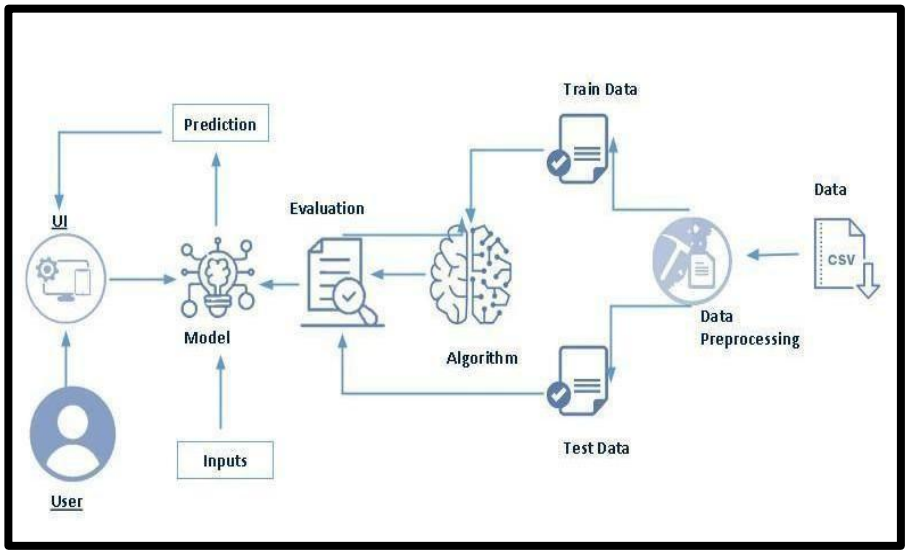
MENTAL HEALTH PREDICTION USING MACHINE LEARNING

Insurance Fraud Detection Using Machine Learning

Mental health issues are reported in order to seek proper care and medical attention for emotional or psychological struggles. Mental health care is a means of support for improving well-being, but now-a-days many cases remain unnoticed or are misclassified due to lack of awareness or proper tools. This can lead to worsening conditions. It becomes a concern when a person's condition is not properly identified, or when help is delayed or denied despite visible symptoms.

These types of overlooked or misjudged mental health cases can cause serious impact to individuals. So, it is necessary to detect the people who may need mental health support. The number of people suffering silently is much higher than the number of cases that are actually identified. So the main purpose of the Mental Health Prediction system is to predict whether a person is likely to have mental health issues or not based on different inputs and behavioural parameters.

Technical Architecture:



Project Flow:

- User interacts with the UI to enter the input.
- Entered input is analysed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

- Define Problem / Problem Understanding
    - Specify the business problem
    - Business requirements ○ Literature Survey
    - Social or Business Impact.
- Data Collection & Preparation
    - Collect the dataset
    - Data Preparation
- Exploratory Data Analysis
    - Descriptive statistical
    - Visual Analysis
- Model Building
    - Training the model in multiple algorithms
    - Testing the model
- Performance Testing & Hyperparameter Tuning
    - Testing model with multiple evaluation metrics
    - Comparing model accuracy before & after applying hyperparameter tuning
- Model Deployment
    - Save the best model
    - Integrate with Web Framework
- Project Demonstration & Documentation
    - Record explanation Video for project end to end solution
    - Project Documentation-Step by step project development procedure

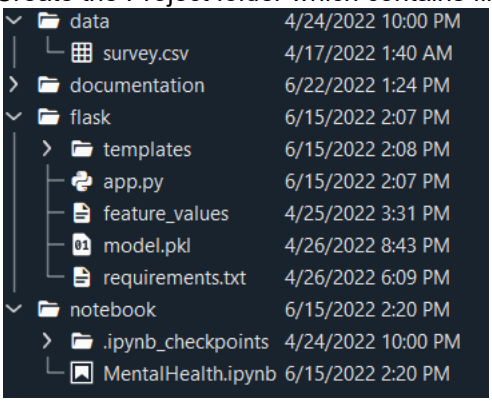
Prior Knowledge:

You must have prior knowledge of following topics to complete this project.

- ML Concepts
  - Supervised learning: https://www.youtube.com/watch?v=QeKshry8pWQ
  - Unsupervised learning: https://www.youtube.com/watch?v=D6gtZrsYi6c
    - Decision tree: https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm
    - Random forest: https://www.javatpoint.com/machine-learning-random-forest-algorithm
    - KNN: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
    - Xgboost: https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/
    - Evaluation metrics: https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/
    - Performance matrix for problem: https://www.youtube.com/watch?v=D6gtZrsYi6c
- Flask Basics : https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Structure:

Create the Project folder which contains files as shown below



- the data folder contains the .csv file used for our project
- We are building a flask application that needs the HTML pages to be stored in the templates folder and a python script app.py for scripting.
- Notebook folder contains model training file MentalHealth.ipynb
- model.pkl is our saved model. We will use this model for flask integration.

ap

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem

Mental health disorders are frequently underdiagnosed due to limitations in early screening, real-time assessment, and targeted public health strategies. Healthcare providers lack efficient tools for timely detection during routine visits, support services face difficulties in delivering personalized care, and public health programs struggle to identify and address high-risk populations. This project proposes a machine learning-based mental health prediction model designed to enhance clinical screening, support individualized interventions, and inform population-level strategies—ultimately improving early detection, access to care, and overall mental well-being.

Activity 2: Business requirements

- Accuracy and Reliability: The system must leverage recent, high-quality, and validated mental health datasets to ensure precise and trustworthy predictions aligned with current psychological standards and diagnostic criteria.
- Scalability and Flexibility: The prediction model should be adaptable to new data inputs, behavioral indicators, and evolving mental health trends, allowing for continuous improvement and updates without system overhaul.
- Regulatory Compliance: The solution must adhere to all applicable data privacy laws and ethical standards, including HIPAA, GDPR, and mental health care regulations, ensuring responsible handling of sensitive patient information.
- User-Centric Design: The interface should be intuitive and accessible for diverse user groups—including healthcare professionals, counselors, and support staff—providing clear outputs and actionable insights with minimal training.
- Integration Capability: The system should support seamless integration with existing electronic health records (EHR), telehealth platforms, and mental health helpline systems to enable smooth operational workflows.
- Performance and Efficiency: The model should deliver predictions with high speed and minimal latency, ensuring timely decisions in both clinical and support environments.

Activity 3: Literature Survey

1. Chung, J., & Teo, J. (2022)

Citation/Source:

Chung, J., & Teo, J. (2022). *Mental Health Prediction Using Machine Learning: Taxonomy Applications, and Challenges*. *Applied Computational Intelligence and Soft Computing*.

Purpose of the Study:

The study aimed to systematically review and evaluate machine learning techniques used for predicting various mental health disorders, and to identify challenges and future research directions in this domain.

Methodology/Approach:

Using the PRISMA framework, the authors selected and analyzed 30 relevant studies. These were grouped by disorder type—schizophrenia, bipolar disorder, depression and anxiety, PTSD, and childhood mental health. The review also classified the ML models used (e.g., SVM, random forest, deep learning) and evaluated performance metrics like accuracy, AUC, and sensitivity.

Key Findings:

The review found that supervised learning was most widely used. Deep learning and ensemble methods like random forest often outperformed traditional models. However, limitations included small datasets, lack of standardization, and limited real-world deployment. Some models achieved over 90% accuracy in specific disorders, like schizophrenia and depression.

Relevance:

This review is highly valuable for researchers and developers aiming to build predictive Mental health tools. It highlights promising methods, practical challenges, and gaps in current research, making it a solid foundation for future work in ML-based mental health prediction.

2. Ahmed, M., et al. (2019)

Citation/Source:

Ahmed, M., Hussain, M., & Hussain, A. (2019). Early prediction of depression and anxiety using a machine learning approach. *Multimedia Tools and Applications*.

Purpose of the Study:

The study aimed to identify anxiety and depression in individuals at an early stage using Machine learning models applied to psychological test data.

Methodology/Approach:
Two datasets were used, and machine learning algorithms including Convolutional Neural Network (CNN), Support Vector Machine (SVM), Linear Discriminant Analysis (LDA), and K-Nearest Neighbour (KNN) were applied to classify the intensity levels of anxiety and depression.
Key Findings:
CNN achieved the highest accuracy: 96% for anxiety and 96.8% for depression. SVM also performed well with accuracies above 95%. LDA achieved over 90% in both categories, while KNN performed the lowest with accuracies below 82%.
Relevance:
This study shows that deep learning models like CNN can be powerful tools for early detection of mental health conditions. It supports the development of intelligent, automated systems to assist psychologists in diagnosing and treating patients efficiently.

3. **Sau, A., & Bhakta, I. (2017)**
Citation/Source:
Sau, A., & Bhakta, I. (2017). Predicting anxiety and depression among elderly patients using machine learning techniques. *Healthcare Technology Letters*.
Purpose of the Study:
The study focused on developing predictive models to identify anxiety and depression in elderly patients, aiming to improve mental health diagnosis through machine learning.
Methodology/Approach:
A dataset of 510 geriatric patients was used. Ten machine learning classifiers, including Random Forest, J48, Naive Bayes, Bayesian Network, and Logistic Regression, were applied and evaluated using 10-fold cross-validation.
Key Findings:
Random Forest achieved the highest accuracy of 89%, followed by J48 with 87.8%, and Random Subspace with 87.5%. Logistic Regression performed the lowest with an accuracy of 72.4%. Overall, ensemble methods outperformed individual classifiers.
Relevance:
This study demonstrates the effectiveness of ensemble machine learning in predicting mental health issues in older adults. It emphasizes the value of integrating ML into healthcare for early detection and better intervention planning.

4. **Conrad, R. C., et al. (2019)**
Citation/Source:
Conrad, R. C., Haghgoo, B., & Hammen, C. (2019). Predicting post-traumatic stress disorder using machine learning techniques in conflict-affected populations. *International Journal of Environmental Research and Public Health*.
Purpose of the Study:
This study aimed to predict PTSD in trauma-exposed individuals using machine learning, focusing on simple interpretable models suitable for binary (yes/no) classification in humanitarian health screening.
Methodology/Approach:
Using data from 441 trauma-exposed individuals in Uganda, the authors applied Random Forest (a decision tree ensemble), LASSO, and Logistic Regression models. The dataset included clinical, demographic, and exposure-related features.
Key Findings:
Random Forest with conditional inference achieved the highest accuracy at 77.25%, followed by Logistic Regression (75.36%) and LASSO (74.88%). Decision-tree-based models were effective in identifying PTSD risk using limited inputs.
Relevance:
This study demonstrates that decision tree methods are practical for binary classification of mental health disorders in low-resource and post-conflict settings. Their explainability and performance make them ideal for early screening and intervention efforts

5. Papini, S., et al. (2018)

Citation/Source:

Papini, S., Pisner, D. A., Shumake, J., & Kilts, C. D. (2018). Predicting PTSD using machine learning with biological and psychological features. *Psychiatry Research*, 267, 1–9.

Purpose of the Study:

The goal was to develop an accurate machine learning model for predicting post-traumatic stress disorder (PTSD) using a combination of clinical, psychological, and environmental data.

Methodology/Approach:

The authors applied gradient-boosted decision trees (GBDT) to a dataset of 341 subjects (110 PTSD patients, 231 trauma-exposed controls). Multiple variables, including trauma exposure, clinical history, and localization factors, were used for prediction.

Key Findings:

The GBDT model achieved a prediction accuracy of 78%, outperforming traditional statistical models. It handled mixed data types and captured complex feature interactions effectively for binary classification of PTSD presence.

Relevance:

This study illustrates the power of decision tree-based models like GBDT in accurately predicting mental health disorders with structured clinical data. Its strong performance and interpretability support its use in mental health screening tools

Activity 4: Social or Business Impact.

Social Impact:

The use of decision tree machine learning in mental healthcare provides a powerful tool for early detection and prevention of mental health issues like anxiety, depression, and PTSD. Because decision trees are simple and interpretable, they can be used in community health programs, schools, or mobile health apps to provide quick, understandable assessments. This improves public mental health awareness, reduces the burden on mental health professionals, and increases access to mental health screening, especially in rural or underserved areas. Ultimately, it helps in reducing suicide rates, improving emotional well-being, and promoting early intervention..

Business Model Impact:

From a business perspective, this technology can be integrated into digital health platforms, wellness apps, or employee assistance programs as a low-cost mental health screening service. Healthcare providers, insurance companies, and telemedicine startups can use decision tree models to automate initial assessments, triage patients, and reduce diagnosis time. The simplicity of decision trees also lowers implementation and training costs, making it ideal for scalable commercial solutions. This creates new opportunities for mental health tech products and services in both public and private sectors.

Milestone 2: Data Collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

Activity 1: Collect the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: https://www.kaggle.com/datasets/buntyshah/auto-insurance-claims-data

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 1.1: Importing the libraries

Import the necessary libraries as shown in the image. (optional) Here we have used visualisation style as fivethirtyeight.

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

Activity 1.2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

●For checking the null values, df.isna().any( ) function is used. To sum those null values we use .sum() function. From the below image we found that there are no null values present in our dataset. So we can skip handling the missing values step.

```
data = pd.read_csv(r'C:\Users\Shagun Khandelwal\Project\data\survey.csv')
```

```
data.head()
```

	Age	Gender	self_employed	family_history	treatment	work_interfere	no_employees	remote_work	tech_company	benefits
0	37	Female	No	No	Yes	Often	6-25	No	Yes	Yes
1	44	Male	No	No	No	Rarely	More than 1000	No	No	Don't know
2	32	Male	No	No	No	Rarely	6-25	No	Yes	No
3	31	Male	No	Yes	Yes	Often	26-100	No	Yes	No
4	31	Male	No	No	No	Never	100-500	Yes	Yes	Yes

5 rows × 23 columns

Activity 2: Data Preparation

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

.Handling missing values

.Handling Outliers

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Activity 2.1: Handling missing values

●For checking the null values, df.isnull().sum( ) function is used. To sum those null values we use .sum() function. From the below image we found that there are null values present in our dataset. So we now we handle the missing values.

```
data.isnull().sum()
```

```
Timestamp                      0
Age                            0
Gender                         0
Country                        0
state                        515
self_employed                 18
family_history                 0
treatment                      0
work_interfere               264
no_employees                   0
remote_work                    0
tech_company                   0
benefits                       0
care_options                   0
wellness_program               0
seek_help                      0
anonymity                      0
leave                          0
mental_health_consequence      0
phys_health_consequence        0
coworkers                      0
supervisor                     0
mental_health_interview        0
phys_health_interview          0
mental_vs_physical             0
obs_consequence                0
comments                    1095
dtype: int64
```

Now removing all missing values:

1.handling Employment:

```
data['self_employed'].value_counts()
```

```
self_employed
No      1095
Yes      146
Name: count, dtype: int64
```

```
data['self_employed'].fillna('No',inplace=True)
```

2.handling work interface:

```
data['work_interfere'].value_counts()
```

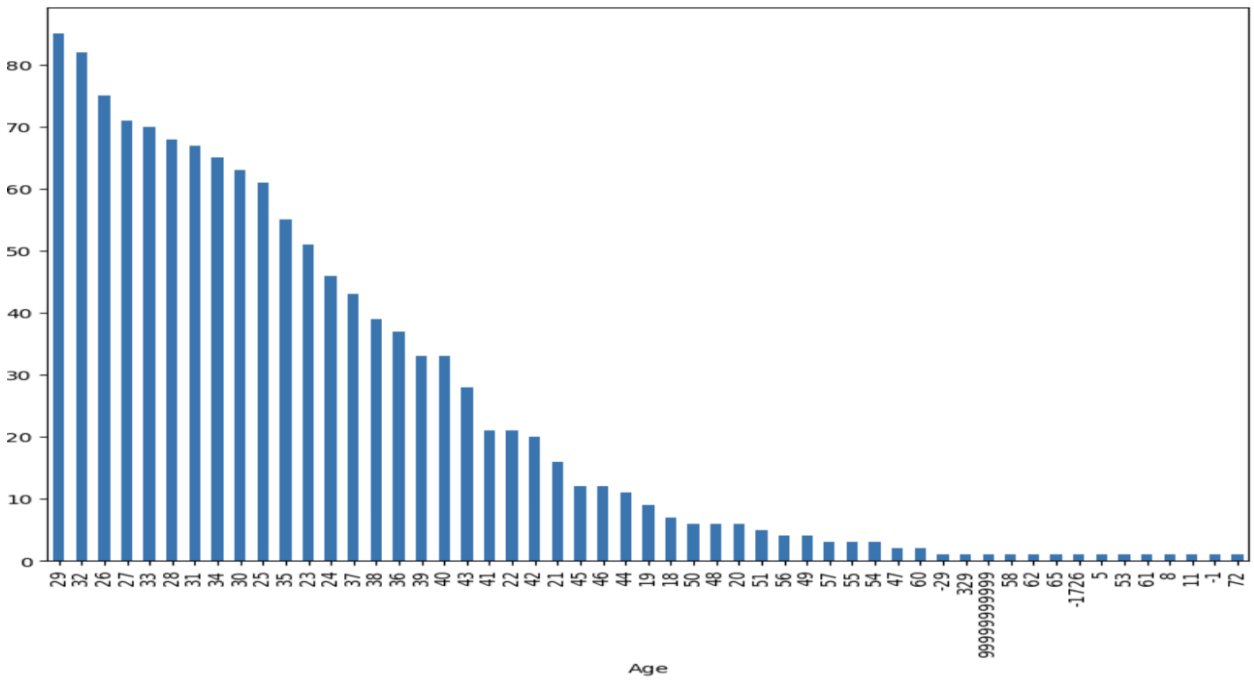
```
work_interfere
Sometimes    465
Never        213
Rarely       173
Often        144
Name: count, dtype: int64
```

```
data['work_interfere'].fillna('N/A',inplace=True)
```

3.Handling Age:

```
data['Age'].value_counts().plot(kind='bar',figsize=(10,8))
```

```
<Axes: xlabel='Age'>
```

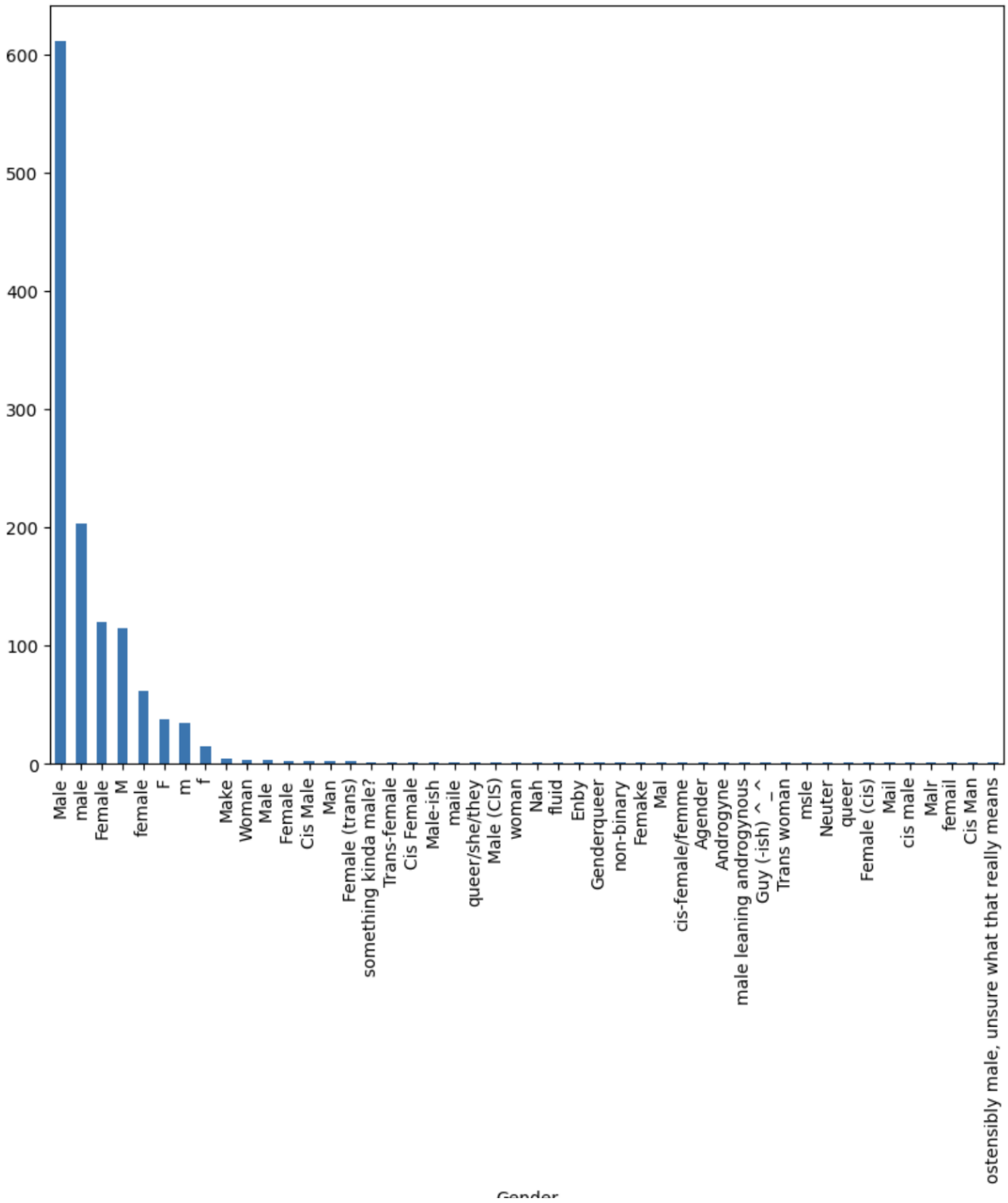



```
data.drop(data[(data['Age']>60)| (data['Age']<18)].index,inplace=True)
```

4.Handling Gender:

```
data['Gender'].value_counts().plot(kind='bar',figsize=(10,8))
```

```
<Axes: xlabel='Gender'>
```



```
data['Gender'].replace(['Male ', 'male', 'M', 'm', 'Male', 'Cis Male',
                        'Man', 'cis male', 'Mail', 'Male-ish', 'Male (CIS)',
                        'Cis Man', 'msle', 'Malr', 'Mal', 'maile', 'Make '],
                       'Male', inplace=True)

data['Gender'].replace(['Female ', 'female', 'F', 'f', 'Woman', 'Female',
                        'femail', 'Cis Female', 'cis-female/femme', 'Femake', 'Female (cis)',
                        'woman'],
                       'Female', inplace=True)

data["Gender"].replace(['Female (trans)', 'queer/she/they', 'non-binary',
                        'fluid', 'queer', 'Androgyne', 'Trans-female', 'male leaning androgynous',
                        'Agender', 'A little about you', 'Nah', 'All',
                        'Ostensibly male, unsure what that really means',
                        'Genderqueer', 'Enby', 'p', 'Neuter', 'something kinda male?',
                        'Guy (-ish) ^_^', 'Trans woman',],
                       'Non-Binary', inplace=True)
```

Activity 2.2: Handling Outliers

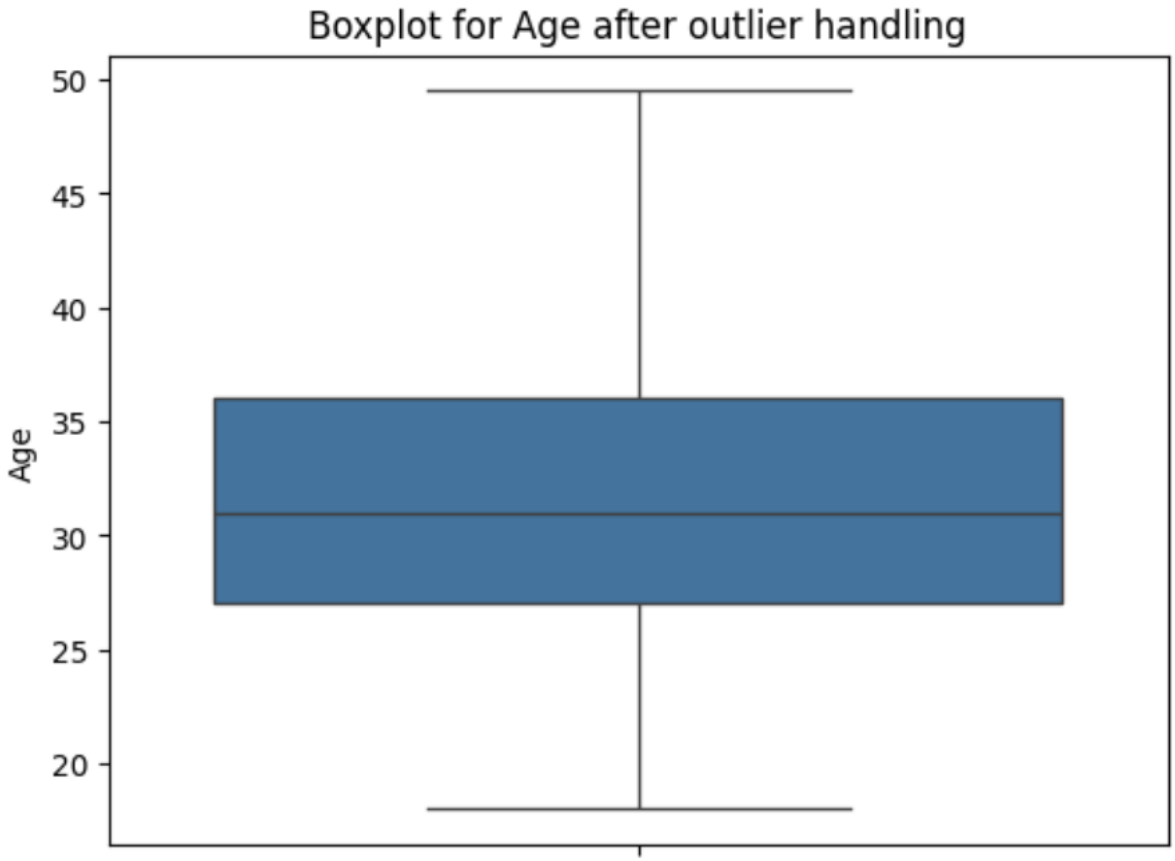
With the help of boxplot, outliers are visualized. And here we are going to find upper bound and lower bound of Age feature with some mathematical formula.

·To find upper bound we have to multiply IQR (Interquartile range) with 1.5 and add it with 3rd quantile. To find lower bound instead of adding, subtract it with 1st quantile. Take image attached below as your reference.

·To handle the outliers transformation technique is used. Here log transformation is used. We have created a function to visualize the distribution and probability plot of Age feature.

```
num_features = data.select_dtypes(include=['int64', 'float64']).columns
print("Numeric columns:", list(num_features))


IQR = []
lower = []
upper = []

for col in num_features:
    Q1 = data[col].quantile(0.25)
    Q3 = data[col].quantile(0.75)
    iqr = Q3 - Q1
    IQR.append(iqr)

    lower.append(Q1 - 1.5 * iqr)
    upper.append(Q3 + 1.5 * iqr)


for i, col in enumerate(num_features):
    data[col] = np.where(data[col] > upper[i], upper[i],
                         np.where(data[col] < lower[i], lower[i], data[col]))


for col in num_features:
    print(f"Boxplot after capping: {col}")
    sb.boxplot(data[col])
    plt.title(f"Boxplot for {col} after outlier handling")
    plt.show()
```

```
Numeric columns: ['Age']
Boxplot after capping: Age
```



Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive statistical

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

```
data.describe(include='all')
```

	Timestamp	Age	Gender	Country	state	self_employed	family_history	treatment	work_interfere
count	1247	1247.000000	1247	1247	735	1247	1247	1247	1247
unique	1235	NaN	5	46	45	2	2	2	5
top	2014-08-27 12:43:28	NaN	Male	United States	CA	No	No	Yes	Sometimes
freq	2	NaN	979	743	137	1107	759	630	463
mean	NaN	31.971131	NaN	NaN	NaN	NaN	NaN	NaN	NaN
std	NaN	7.052598	NaN	NaN	NaN	NaN	NaN	NaN	NaN
min	NaN	18.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
25%	NaN	27.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
50%	NaN	31.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
75%	NaN	36.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
max	NaN	60.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN

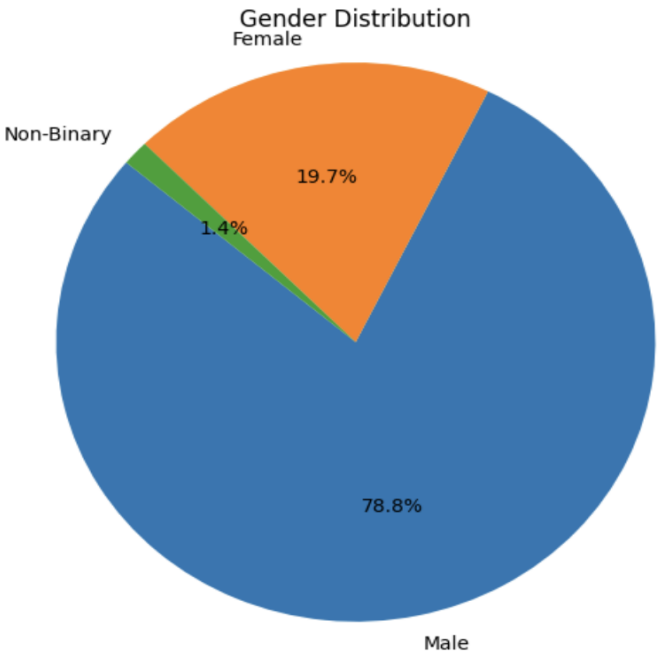
Activity 2: Visual analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

Activity 2.1: Univariate analysis

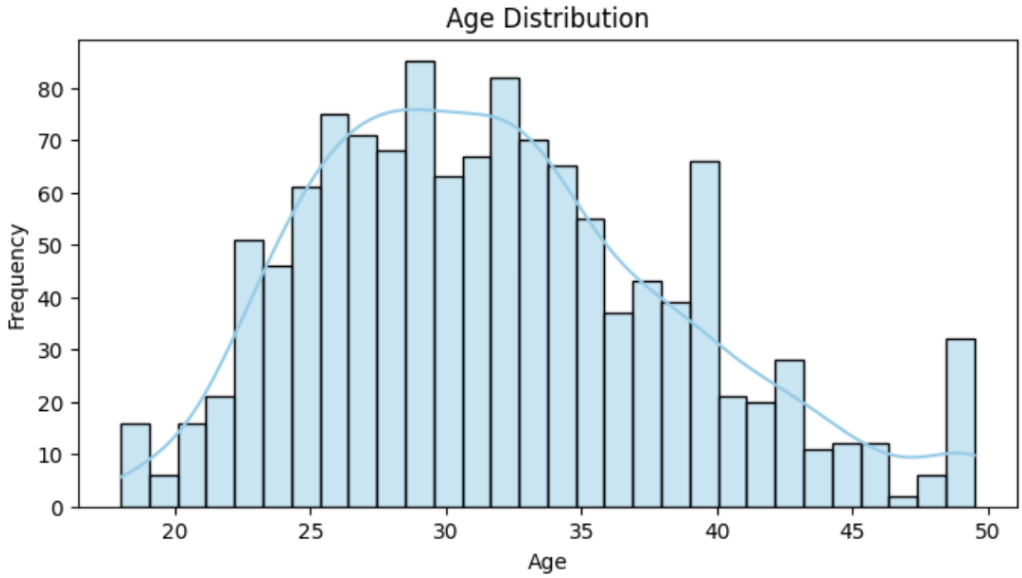
In simple words, univariate analysis is understanding the data with single feature. Here we have displayed two different graphs such as Piechart and countplot.

Seaborn package provides a wonderful function countplot. It is more useful for categorical features.



- **Pie chart describes the composition of Gender Distribution.** It shows that **78.8%** of the individuals are **Male**, forming a significant majority. **Female** individuals make up **19.7%**, representing a relatively smaller group. Only **1.4%** identify as **Non-Binary**, making it the least represented category in the dataset.
- **This chart indicates a skewed distribution** towards the male gender, showing that most of the data points or observations are from male individuals. Female and Non-Binary representations are considerably lower, highlighting a possible gender imbalance in the dataset.

```
plt.figure(figsize=(8, 4))
sb.histplot(data['Age'], kde=True, bins=30, color='skyblue')
plt.title("Age Distribution")
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.show()
```

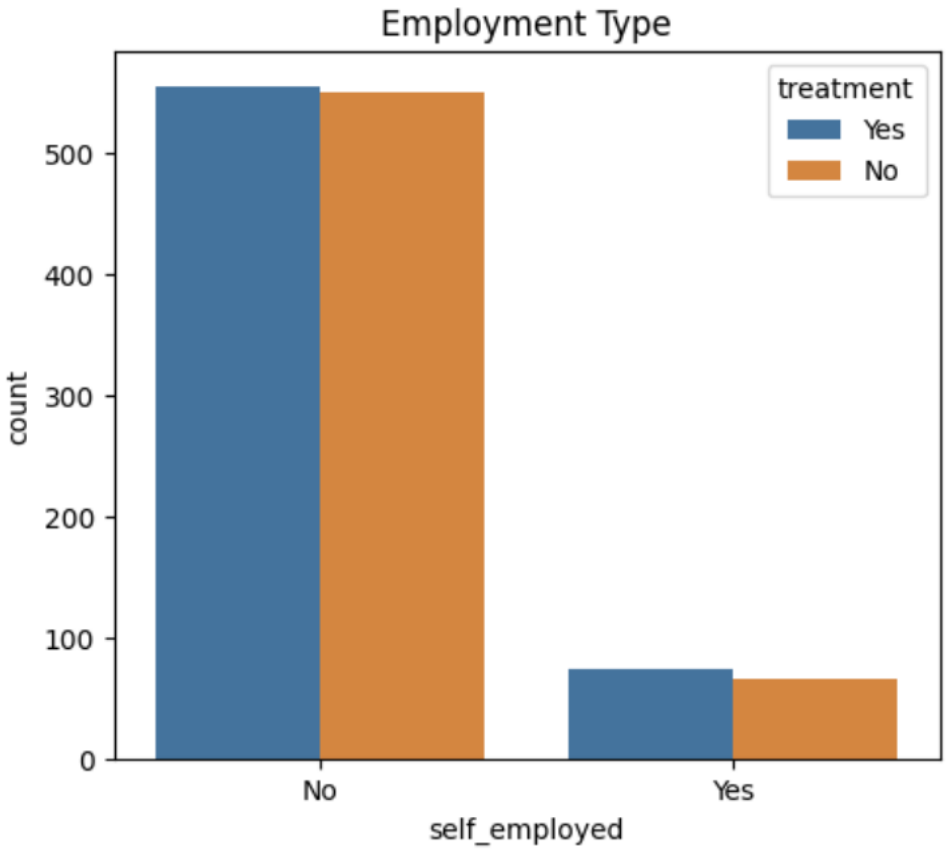


Activity 2.2: Bivariate analysis

To find the relation between two features we use bivariate analysis. Here we can used barplot.

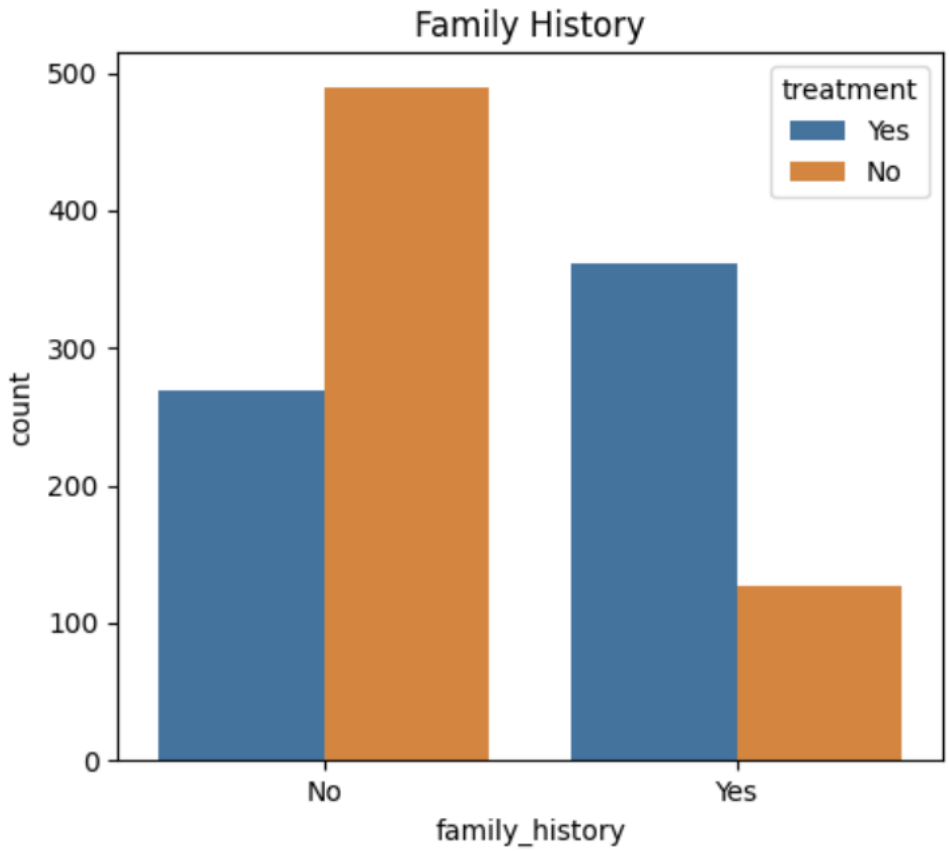
The bar chart shows that most individuals are not self-employed, with nearly equal treatment distribution, while self-employed individuals are fewer, slightly more of whom have sought treatment.

```
plt.figure(figsize=(10,40))
plt.subplot(9,2,1)
sb.countplot(data = data, x = 'self_employed', hue='treatment')
plt.title('Employment Type')
```



·The bar chart shows that individuals with a family history of mental illness are more likely to seek treatment, while those without such a history are less likely to do so.

```
plt.subplot(9, 2, 2)
sb.countplot(x='family_history', data=data, hue='treatment')
plt.title('Family History')
```



Activity 2.3: Multivariate analysis

In simple words, multivariate analysis is to find the relation between multiple features. Here we have used heatmap from seaborn package.

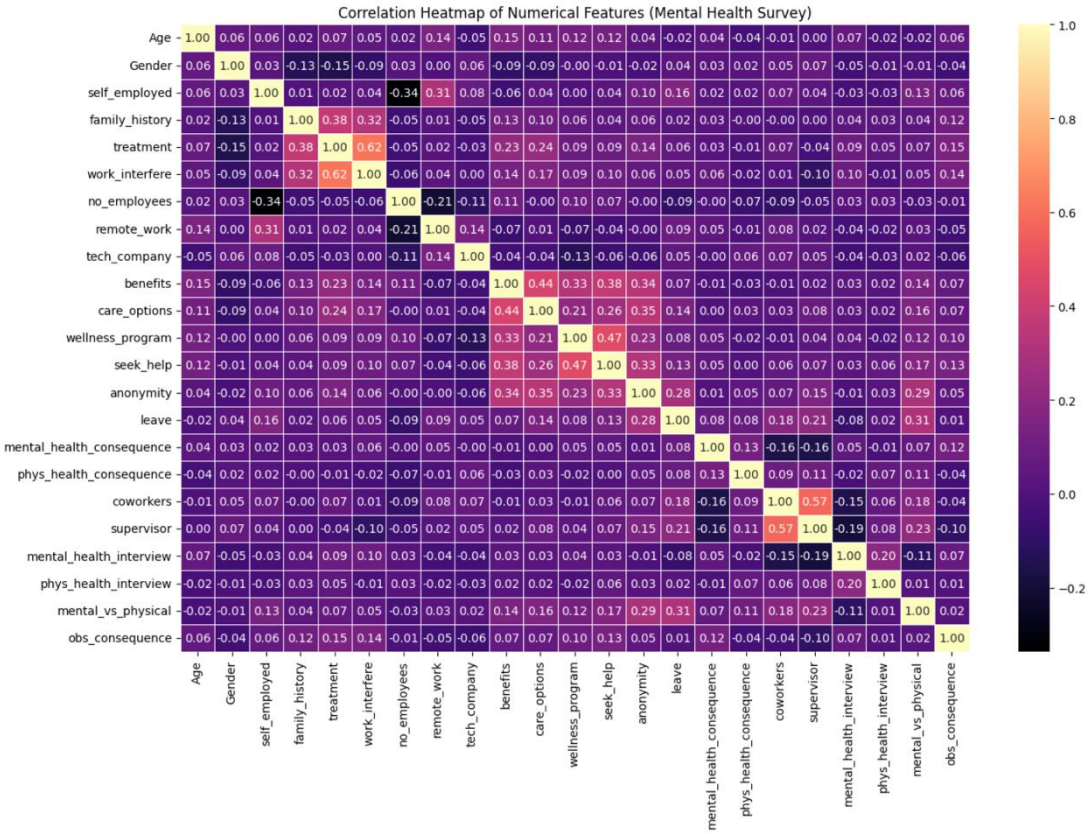
- From the correlation heatmap, we observe that certain features are moderately to highly correlated.

- Treatment and work interference show a positive correlation of 0.62, suggesting higher treatment rates among those affected at work.

- Family history is correlated with treatment (0.38), indicating its potential influence on mental health awareness.

- Wellness program, care options, and seek help are moderately correlated, reflecting the impact of organizational support.

- Most other features show low correlation and can be retained, while multicollinearity remains minimal.



Encoding the Categorical Features:

- The categorical Features are can't be passed directly to the Machine Learning Model. So we convert them into Numerical data based on their order. This Technique is called Encoding.
- Here we are importing Label Encoder from the Sklearn Library.
- Here we are applying fit_transform to transform the categorical features to numerical features.

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import LabelEncoder, OrdinalEncoder


ct = ColumnTransformer([('oe', OrdinalEncoder(), ['Gender', 'self_employed', 'family_history',
    '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'])], remainder='passthrough')

X = ct.fit_transform(X)

le = LabelEncoder()
y = le.fit_transform(y)
```

Final Encoding and cleaning the Features:

```
# Drop columns that won't be used for prediction
data = data.drop(['Timestamp', 'Country', 'state', 'comments'], axis=1)

# Handle missing values
data['self_employed'].fillna('No', inplace=True)
data['work_interfere'].fillna('N/A', inplace=True)

# Clean Age column
data = data[(data['Age'] >= 18) & (data['Age'] <= 60)]

# Clean Gender column
gender_map = {
    'Male ': 'Male', 'male': 'Male', 'M': 'Male', 'm': 'Male',
    'Male': 'Male', 'Cis Male': 'Male', 'Man': 'Male', 'cis male': 'Male',
    'Mail': 'Male', 'Male-ish': 'Male', 'Male (CIS)': 'Male',
    'Cis Man': 'Male', 'msle': 'Male', 'Malr': 'Male', 'Mal': 'Male',
    'maile': 'Male', 'Make': 'Male',
    'Female ': 'Female', 'female': 'Female', 'F': 'Female', 'f': 'Female',
    'Woman': 'Female', 'Female': 'Female', 'femail': 'Female',
    'Cis Female': 'Female', 'cis-female/femme': 'Female', 'Femake': 'Female',
    'Female (cis)': 'Female', 'woman': 'Female'
}
data['Gender'] = data['Gender'].map(gender_map).fillna('Non-Binary')
```

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OrdinalEncoder

# List of categorical columns to encode
categorical_cols = [
    'Gender', 'self_employed', 'family_history', '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'
]

ct = ColumnTransformer([
    ('encoder', OrdinalEncoder(), categorical_cols)
], remainder='passthrough')

X = ct.fit_transform(data.drop('treatment', axis=1))
y = data['treatment'].apply(lambda x: 1 if x == 'Yes' else 0)
```

Splitting data into train and test

Now let's split the Dataset into train and test sets. First split the dataset into x and y and then split the data set

Here x and y variables are created. On x variable, df is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=49)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Handling Imbalanced dataset

- Imbalanced data is a common problem in machine learning and data analysis, where the number of observations in one class is significantly higher or lower than the other class. Handling imbalanced data is important to ensure that the model is not biased towards the majority class and can accurately predict the minority class.
- Here we are using SMOTE Technique.

```
[ ]  from imblearn.over_sampling import SMOTE
     smt=SMOTE()
     X_train, y_train = smt.fit_resample(X_train, y_train)
```

Scaling

- Scaling is a technique used to transform the values of a dataset to a similar scale to improve the performance of machine learning algorithms. Scaling is important because many machine learning algorithms are sensitive to the scale of the input features.
- Here we are using Standard Scaler.
- This scales the data to have a mean of 0 and a standard deviation of 1. The formula is given by:
  X_scaled = (X - X_mean) / X_std

```
from sklearn.preprocessing import StandardScaler
Std_scaler=StandardScaler()
```

```
X_train = Std_scaler.fit_transform(X_train)
X_train = pd.DataFrame(X_train, columns=x.columns)
```

```
X_test = Std_scaler.transform(X_test)
X_test = pd.DataFrame(X_test, columns=x.columns)
```

Milestone 4: Model Building

Activity 1: Training the model in multiple algorithms

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying three classification algorithms. The best model is saved based on its performance. We start by import all the models.

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier
from xgboost.sklearn import XGBClassifier
from sklearn.metrics import accuracy_score, roc_curve, confusion_matrix, classification_report, auc
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import accuracy_score, classification_report
from sklearn.pipeline import make_pipeline
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

Activity 1.1: Decision tree model

First Decision Tree is imported from sklearn Library then DecisionTreeClassifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. We can find the Train and Test accuracy by X_train and X_test.

```
dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
y_pred = dtc.predict(X_test)

dtc_train_acc = accuracy_score(y_train, dtc.predict(X_train))
dtc_test_acc = accuracy_score(y_test, y_pred)
```

Activity 1.2: Random forest model

First Random Forest Model is imported from sklearn Library then RandomForestClassifier algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in a new variable. We can find the Train and Test accuracy by X_train and X_test.

```
rfc = RandomForestClassifier(
    criterion='entropy',
    max_depth=10,
    max_features='sqrt',
    min_samples_leaf=1,
    min_samples_split=3,
    n_estimators=140,
    random_state=42
)

rfc.fit(X_train, y_train)

y_pred = rfc.predict(X_test)

rfc_train_acc = accuracy_score(y_train, rfc.predict(X_train))
rfc_test_acc = accuracy_score(y_test, y_pred)
```

Activity 1.3: KNN model

KNN Model is imported from sklearn Library then KNeighborsClassifier algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

knn_train_acc = accuracy_score(y_train, knn.predict(X_train))
knn_test_acc = accuracy_score(y_test, y_pred)
```

Activity 1.4: Logistic Regression model

Logistic Regression Model is imported from sklearn Library then Logistic Regression algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix is done.

```
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)

lr_train_acc = accuracy_score(y_train, lr.predict(X_train))
lr_test_acc = accuracy_score(y_test, y_pred)
```

Activity 1.5: Gaussian Naïve Bayes:
GaussianNB is imported from sklearn and initialized to create the Gaussian Naïve Bayes model. The training data is passed using the `.fit()` function. Predictions on the test set are made using `.predict()`. Accuracy for both training and testing sets is calculated using X_train and X_test to evaluate model performance.

```
nb = GaussianNB()
nb.fit(X_train, y_train)
y_pred = nb.predict(X_test)

nb_train_acc = accuracy_score(y_train, nb.predict(X_train))
nb_test_acc = accuracy_score(y_test, y_pred)
```

Activity 1.6: SVC model

SVC is imported from the sklearn library and initialized to create the Support Vector Classifier model. The training dataset is passed to the model using the `.fit()` function. The `.predict()` function is used on the test data to generate predictions. Accuracy for both training and testing is calculated using X_train and X_test.

```
svc = SVC(kernel='rbf')
svc.fit(X_train, y_train)
y_pred = svc.predict(X_test)

svc_train_acc = accuracy_score(y_train, svc.predict(X_train))
svc_test_acc = accuracy_score(y_test, y_pred)
```

1.7: ADA Booster Classifer:

AdaBoostClassifier is imported from sklearn. The model is initialized and trained using `.fit()` on the training dataset. It combines weak learners to improve accuracy iteratively. The `.predict()` function is used on the test data, and accuracy is computed using predictions on X_train and X_test.

```
abc = AdaBoostClassifier(n_estimators=100, learning_rate=1.0)
abc.fit(X_train, y_train)
y_pred = abc.predict(X_test)

abc_train_acc = accuracy_score(y_train, abc.predict(X_train))
abc_test_acc = accuracy_score(y_test, y_pred)
```

1.8: Gradient Boosting Classifier:

GradientBoostingClassifier is imported from sklearn. The classifier is initialized and trained with the `.fit()` method using training data. It builds trees sequentially to reduce errors. The `.predict()` method is used for test predictions, and model accuracy is measured using both X_train and X_test.

```
gbc = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3)
gbc.fit(X_train, y_train)
y_pred = gbc.predict(X_test)

gbc_train_acc = accuracy_score(y_train, gbc.predict(X_train))
gbc_test_acc = accuracy_score(y_test, y_pred)
```

1.9: XGBoost Classifier:

XGBClassifier is imported from the xgboost library and initialized. The training data is fitted to the model using .fit(). It uses optimized gradient boosting for performance. Test predictions are obtained via .predict() and model accuracy is determined using X_train and X_test.

```
xgb = XGBClassifier(n_estimators=100, learning_rate=0.1, max_depth=3, use_label_encoder=False, eval_metric='logl
xgb.fit(X_train, y_train)
y_pred = xgb.predict(X_test)

xgb_train_acc = accuracy_score(y_train, xgb.predict(X_train))
xgb_test_acc = accuracy_score(y_test, y_pred)
```

Milestone 5: Performance Testing & Hyperparameter Tuning

Activity 1: Testing model with multiple evaluation metrics

Multiple evaluation metrics means evaluating the model's performance on a test set using different performance measures. This can provide a more comprehensive understanding of the model's strengths and weaknesses. We are using evaluation metrics for classification tasks including accuracy, precision, recall, support and F1-score.

Activity 1.1: Compare the model

For comparing the above four models, the compareModel function is defined.

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=49)

def evaluate_model(model, name):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(f"\n{name} Accuracy: {accuracy_score(y_test, y_pred):.4f}")


models = {
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'KNN': make_pipeline(StandardScaler(), KNeighborsClassifier()),
    'Logistic Regression': make_pipeline(StandardScaler(), LogisticRegression(max_iter=1000)),
    'Naive Bayes': GaussianNB(),
    'SVM': make_pipeline(StandardScaler(), SVC(probability=True)),
    'AdaBoost': AdaBoostClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'XGBoost': XGBClassifier( eval_metric='logloss')
}
for name, model in models.items():
    evaluate_model(model, name)
```

```
Decision Tree Accuracy: 0.7627

Random Forest Accuracy: 0.8373

KNN Accuracy: 0.7733

Logistic Regression Accuracy: 0.8507

Naive Bayes Accuracy: 0.8453

SVM Accuracy: 0.8533

AdaBoost Accuracy: 0.8693

Gradient Boosting Accuracy: 0.8400

XGBoost Accuracy: 0.8400
```

Activity 2: Selecting best model and calculating the model parameters:

```
abc = AdaBoostClassifier(random_state=99)
abc.fit(X_train,y_train)
pred_abc = abc.predict(X_test)
print('Accuracy of AdaBoost-',accuracy_score(y_test,pred_abc))
```

```
Accuracy of AdaBoost- 0.8693333333333333
```

```
from sklearn.model_selection import RandomizedSearchCV

params_abc = {
    'n_estimators': [int(x) for x in np.linspace(start=1, stop=50, num=15)],
    'learning_rate': [(0.97 + x / 100) for x in range(0, 8)],
}

abc_random = RandomizedSearchCV(
    random_state=49,
    estimator=abc,
    param_distributions=params_abc,
    n_iter=50,
    cv=5,
    n_jobs=-1
)
params_abc
```

```
{'n_estimators': [1, 4, 8, 11, 15, 18, 22, 25, 29, 32, 36, 39, 43, 46, 50],
 'learning_rate': [0.97, 0.98, 0.99, 1.0, 1.01, 1.02, 1.03, 1.04]}
```

```
abc_random.fit(X_train, y_train)
```

```
RandomizedSearchCV(cv=5, estimator=AdaBoostClassifier(random_state=99),
                   n_iter=50, n_jobs=-1,
                   param_distributions={'learning_rate': [0.97, 0.98, 0.99, 1.0,
                                                          1.01, 1.02, 1.03,
                                                          1.04],
                                        'n_estimators': [1, 4, 8, 11, 15, 18,
                                                         22, 25, 29, 32, 36, 39,
                                                         43, 46, 50]},
                   random_state=49)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
abc_random.best_params_
```

```
{'n_estimators': 46, 'learning_rate': 0.99}
```

```
abc_tuned = AdaBoostClassifier(random_state=49, n_estimators=11, learning_rate=1.02)
```

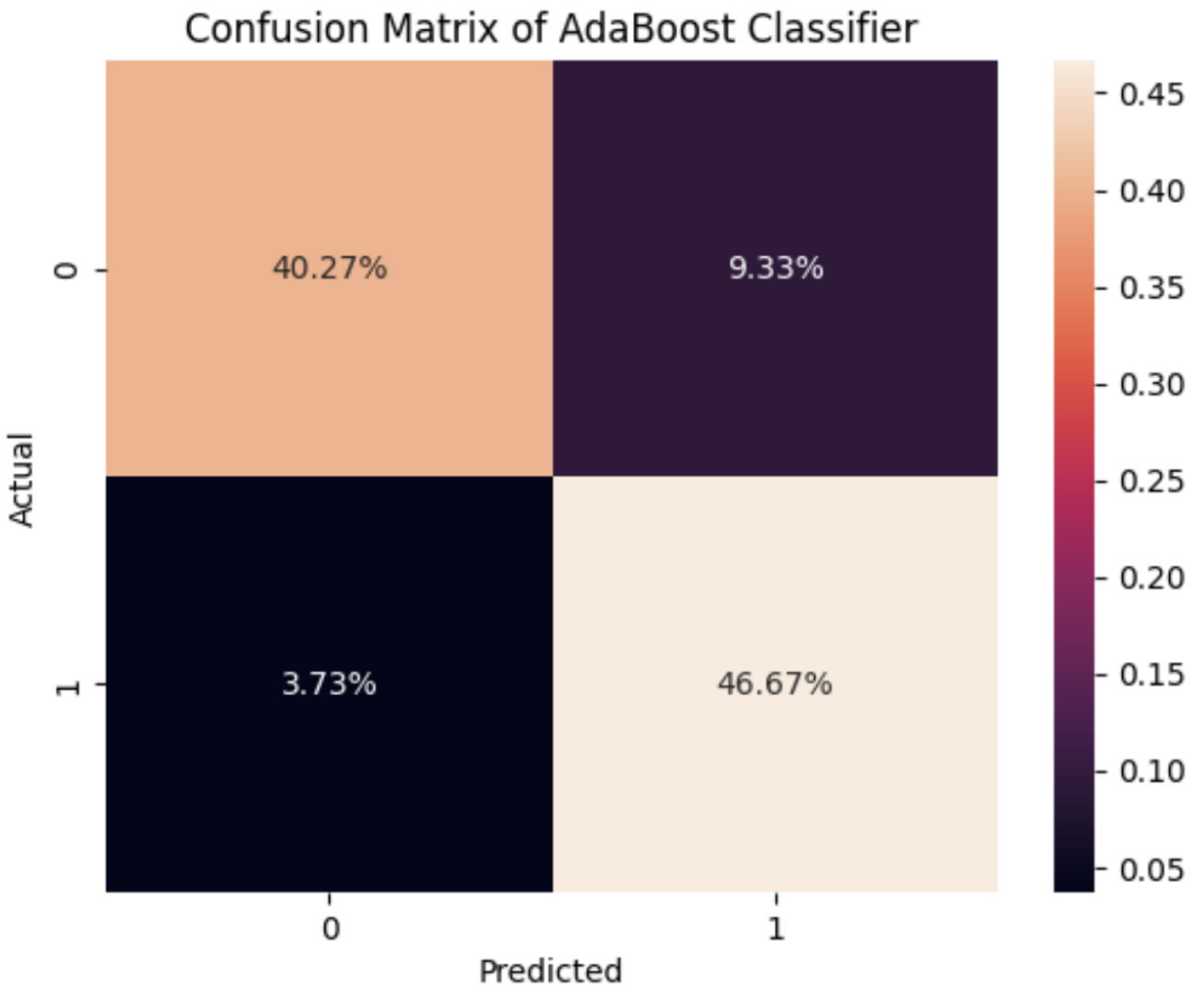
```
pred_abc_tuned = abc_tuned.predict(X_test)
```

```
print("Accuracy of AdaBoost (tuned):", accuracy_score(y_test, pred_abc_tuned))
```

```
Accuracy of AdaBoost (tuned): 0.872
```

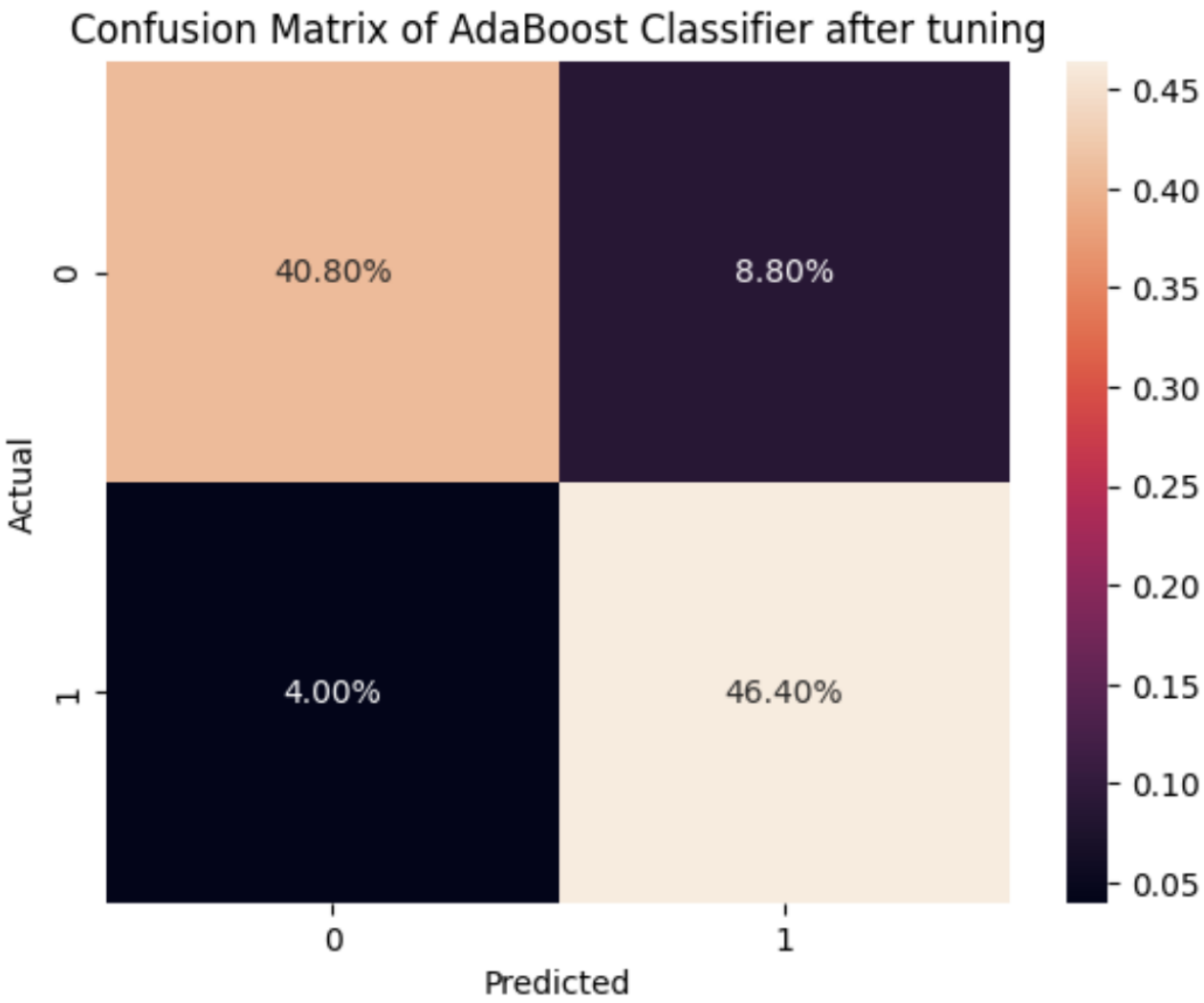
```
cf_matrix = confusion_matrix(y_test, pred_abc)
```

```
sb.heatmap(cf_matrix/np.sum(cf_matrix), annot=True, fmt='.2%')
plt.title('Confusion Matrix of AdaBoost Classifier')
plt.xlabel('Predicted')
plt.ylabel('Actual')
cf_matrix = confusion_matrix(y_test, pred_abc_tuned)
```

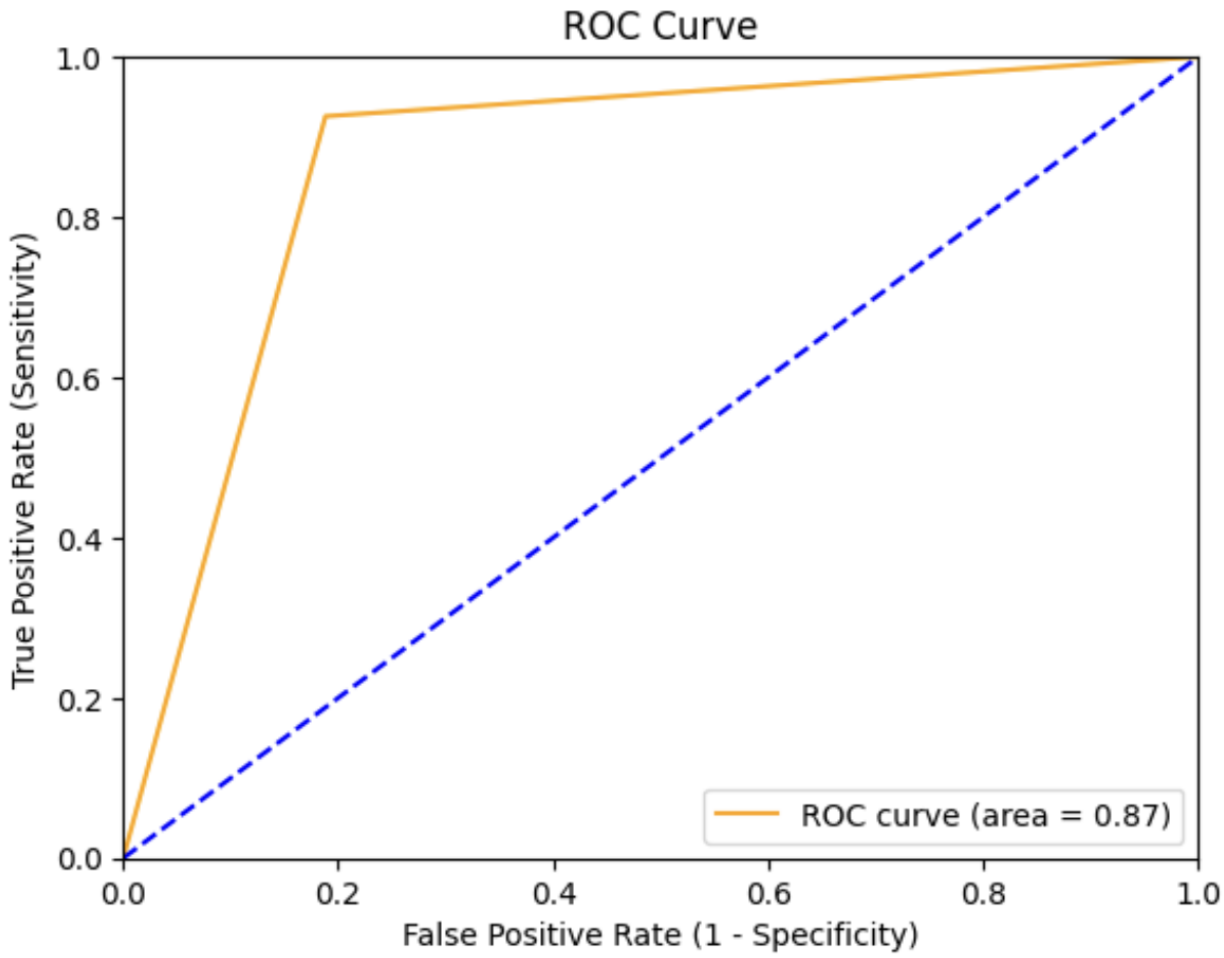



```
sb.heatmap(cf_matrix/np.sum(cf_matrix), annot=True, fmt='.2%')
plt.title('Confusion Matrix of AdaBoost Classifier after tuning')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

```
Text(50.722222222222214, 0.5, 'Actual')
```




```
from sklearn import metrics
fpr_abc, tpr_abc, thresholds_abc = roc_curve(y_test, pred_abc)
roc_auc_abc = metrics.auc(fpr_abc, tpr_abc)
plt.plot(fpr_abc, tpr_abc, color='orange', label='ROC curve (area = %0.2f)' % roc_auc_abc)
plt.plot([0, 1], [0, 1], color='blue', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.title('ROC Curve')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.legend(loc="lower right")
plt.show()
roc_curve(y_test, pred_abc)
```




```
(array([0.        , 0.18817204, 1.        ]),
 array([0.        , 0.92592593, 1.        ]),
 array([inf,  1.,  0.]))
```

```
print(classification_report(y_test, pred_abc))
```

```
              precision    recall  f1-score   support

           0       0.92      0.81      0.86       186
           1       0.83      0.93      0.88       189

    accuracy                           0.87       375
   macro avg       0.87      0.87      0.87       375
weighted avg       0.87      0.87      0.87       375
```

```
print(classification_report(y_test, pred_abc_tuned))
import pickle
```

```
              precision    recall  f1-score   support

           0       0.91      0.82      0.86       186
           1       0.84      0.92      0.88       189

    accuracy                           0.87       375
   macro avg       0.88      0.87      0.87       375
weighted avg       0.88      0.87      0.87       375
```

Milestone 6: Model Deployment

Activity 1: Save the best model

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance. This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.

```
joblib.dump(abc_tuned, 'model.pkl')
joblib.dump(ct, 'feature_values.joblib')
```

```
['feature_values.joblib']
```

Activity 2: Integrate with Web Framework

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building server-side script
- Run the web application

Activity 2.1: Building Html Page:

For this project create HTML file namely index.html
and save them in the templates folder. Refer this link for templates.

Activity 2.2: Build Python code:

Import the libraries:

```
from flask import Flask, render_template, request, session, redirect, url_for
import pandas as pd
import joblib
from sklearn.compose import ColumnTransformer
import random
import time
```

Load the saved model. Importing the flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (_name_) as argument.

```
from flask import Flask, render_template, request, session, redirect, url_for
import pandas as pd
import joblib
from sklearn.compose import ColumnTransformer
import random
import time
```

Render HTML page:

```
@app.route('/')
def home():
    session.clear()
    return render_template('home.html')
```

Here we will be using a declared constructor to route to the HTML page which we have created earlier.

In the above example, '/' URL is bound with the index.html function. Hence, when the home page of the web server is opened in the browser, the html page will be rendered.

Whenever you enter the values from the html page the values can be retrieved using POST Method.Retrieves the value from UI:

```python
@app.route('/assessment/complete')
def complete_assessment():
    # Start with default values
    data = DEFAULT_VALUES.copy()

    # Update with user answers, with proper type conversion and standardization
    for key, value in session.get('answers', {}).items():
        # Handle emoji responses by extracting text
        if isinstance(value, str) and ' ' in value:
            value = value.split(' ')[-1]  # Take last word after space

        # Standardize the response to match model expectations
        standardized_value = standardize_response(key, value)

        # Convert to proper types
        if key == 'Age':  # Numeric field
            try:
                data[key] = int(standardized_value) if standardized_value else DEFAULT_VALUES[key]
                # Ensure age is within reasonable bounds
                data[key] = max(18, min(100, data[key]))
            except (ValueError, TypeError):
                data[key] = DEFAULT_VALUES[key]
        else:  # Categorical fields
            data[key] = str(standardized_value).strip() if standardized_value else DEFAULT_VALUES[key]

    # Convert to DataFrame
    input_df = pd.DataFrame([data])

    try:
        # Ensure all columns are present and in correct order
        expected_columns = ct.feature_names_in_

        # Add any missing columns with default values
        for col in expected_columns:
            if col not in input_df.columns:
                input_df[col] = DEFAULT_VALUES.get(col, 'Unknown')

        # Remove any extra columns
        input_df = input_df[expected_columns]

        # Debug: Print the data being sent to the model
        app.logger.info(f"Input data for model: {input_df.to_dict('records')[0]}")

        # Apply transformations
        X = ct.transform(input_df)
        prediction = model.predict(X)[0]

        # Add some delay for dramatic effect
        time.sleep(2)

        return render_template('output.html', prediction=prediction)

    except Exception as e:
        app.logger.error(f"Error during transformation: {str(e)}")
        return render_template('error.html',
                               error="We encountered an issue processing your responses. Please try again.")
```

Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function.

This function returns the prediction. And this prediction value will be rendered to the text that we have mentioned in the submit.html page earlier.

Main Function:

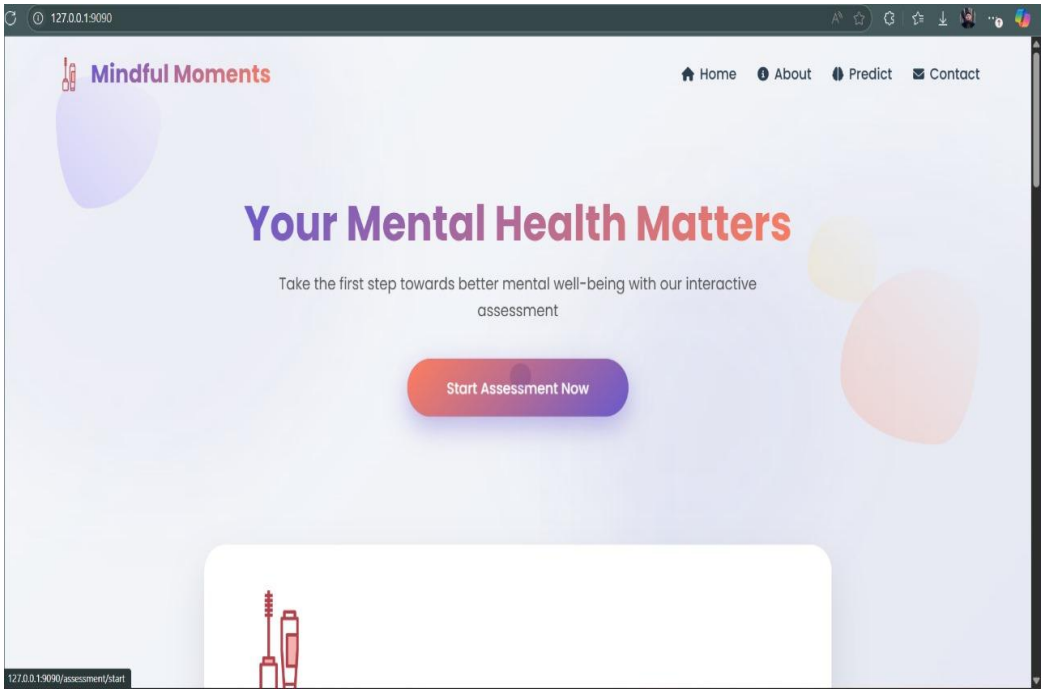
```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=9090, debug=True)
```

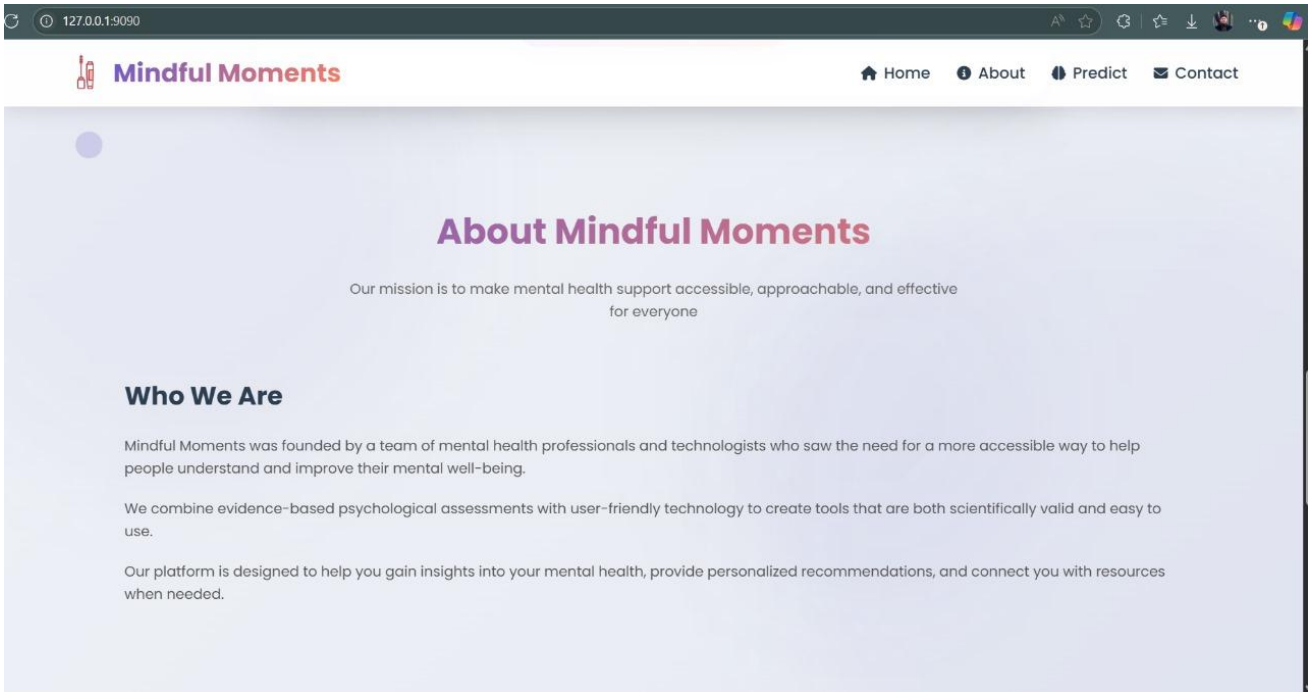
Activity 2.3: Run the web application

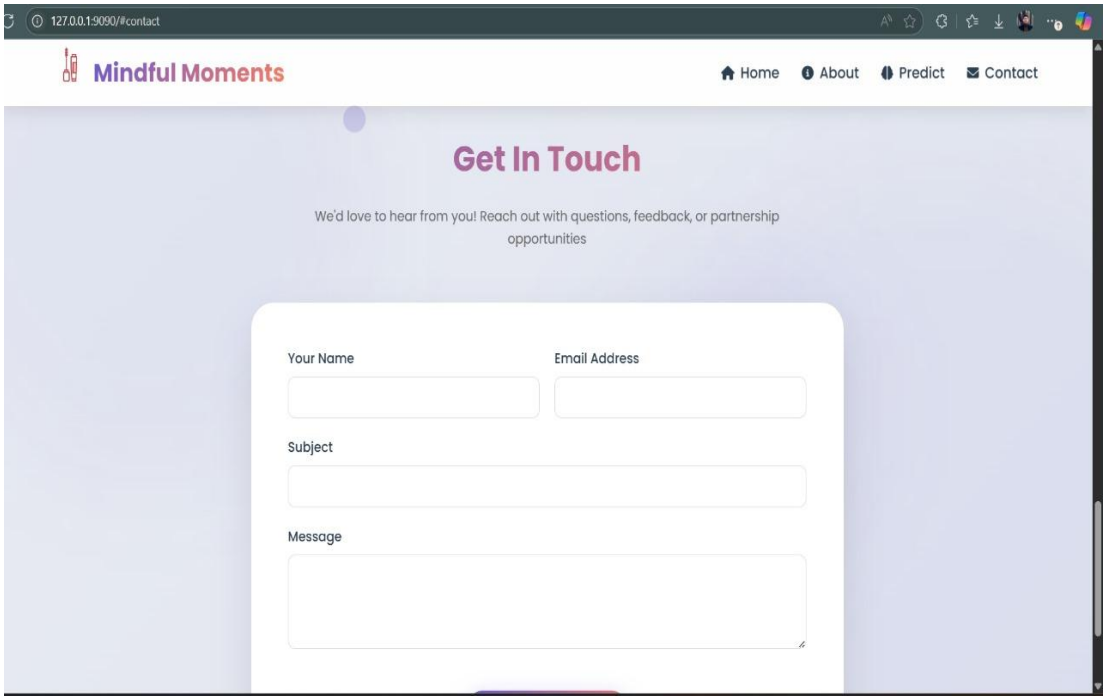
- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top left corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

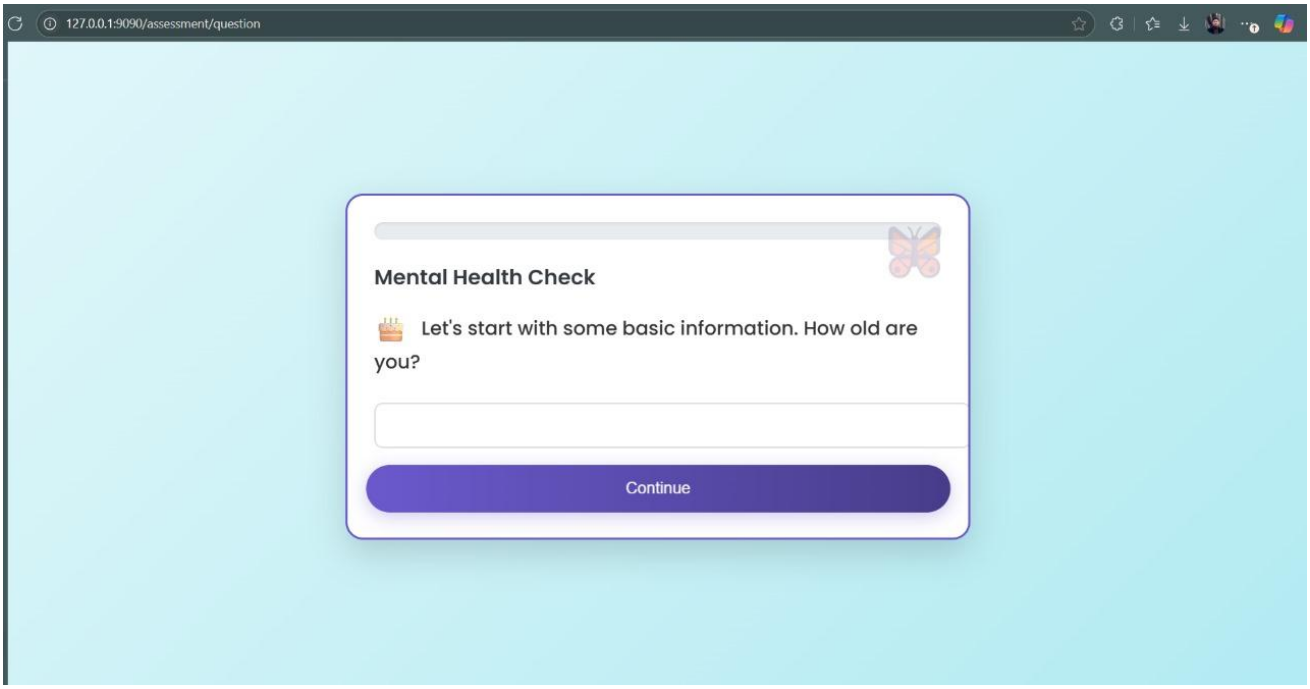
```
(.venv) C:\Users\Shagun Khandelwal\Project\flask>python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:9090
 * Running on http://192.168.149.59:9090
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 933-402-900
```

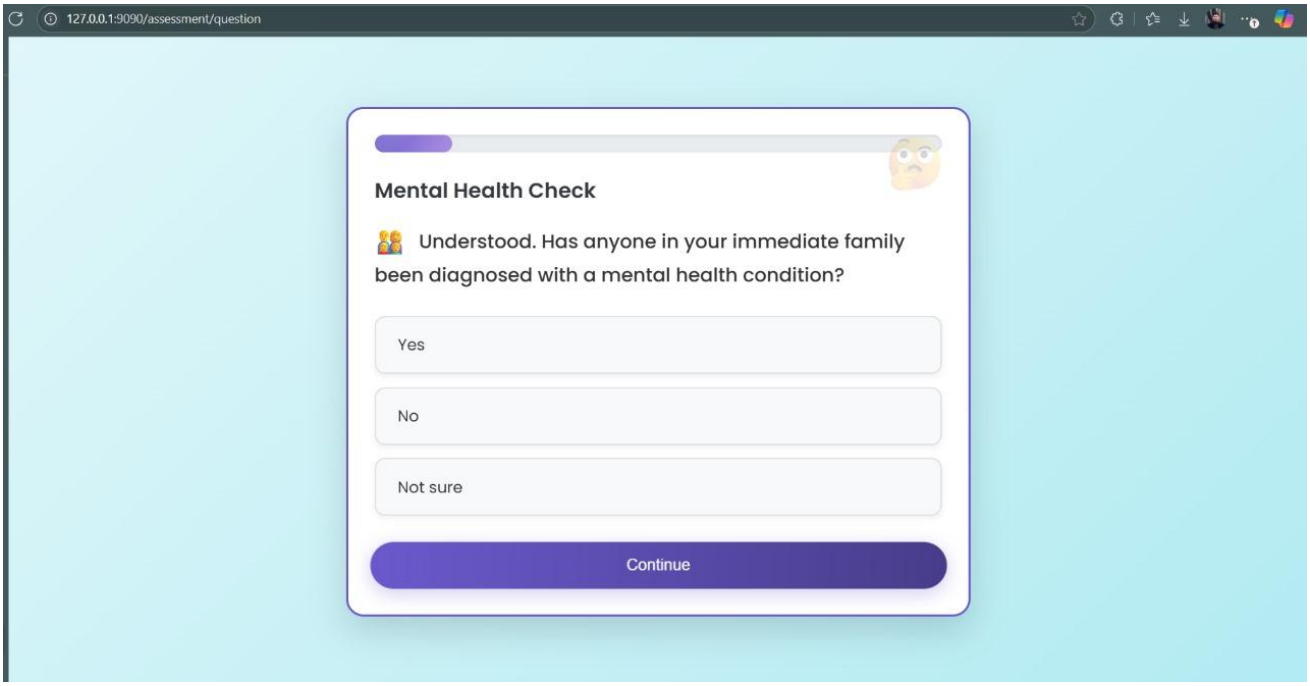
Now,Go the web browser and write the localhost url (http://127.0.0.1:9090) to get the below result

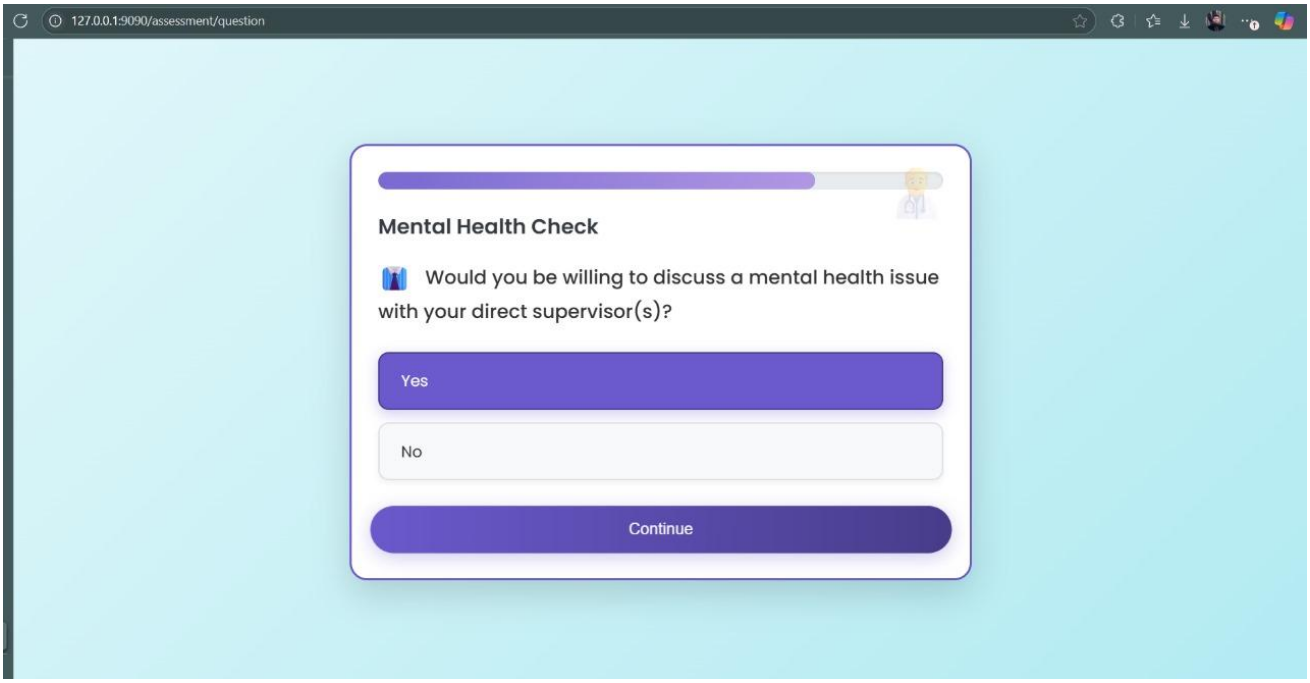


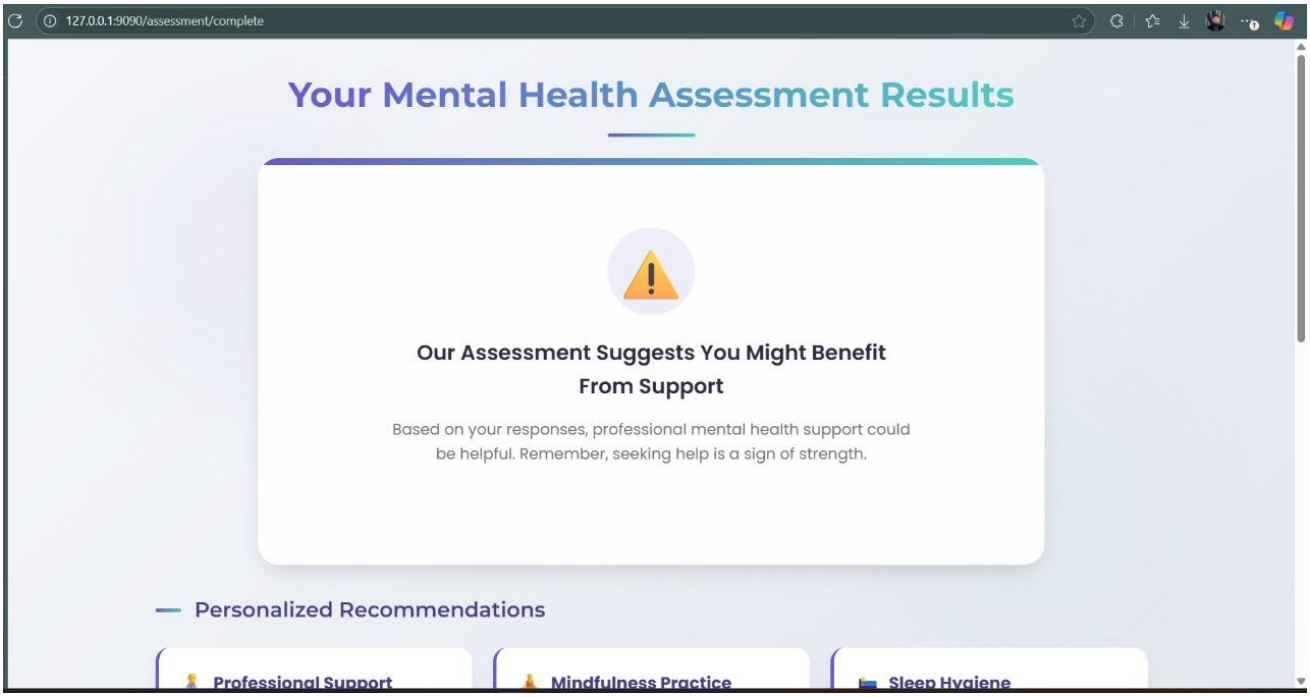

127.0.0.1:9090
Mindful Moments
Home
About
Predict
Contact
About Mindful Moments
Our mission is to make mental health support accessible, approachable, and effective for everyone
Who We Are
Mindful Moments was founded by a team of mental health professionals and technologists who saw the need for a more accessible way to help people understand and improve their mental well-being.
We combine evidence-based psychological assessments with user-friendly technology to create tools that are both scientifically valid and easy to use.
Our platform is designed to help you gain insights into your mental health, provide personalized recommendations, and connect you with resources when needed.

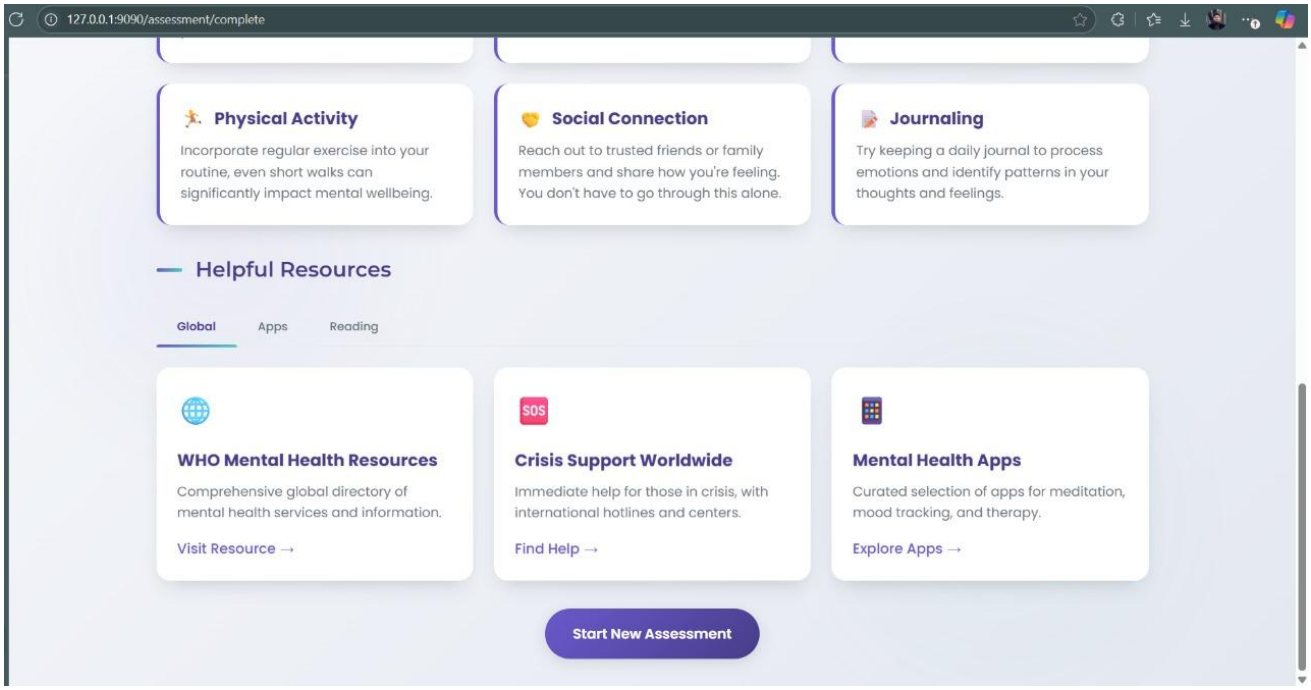

127.0.0.1:9090/#contact
Mindful Moments
Home
About
Predict
Contact
Get In Touch
We'd love to hear from you! Reach out with questions, feedback, or partnership opportunities
Your Name
Email Address
Subject
Message


127.0.0.1:9090/assessment/question
Mental Health Check
Let's start with some basic information. How old are you?
Continue


127.0.0.1:9090/assessment/question
Mental Health Check
Understood. Has anyone in your immediate family been diagnosed with a mental health condition?
Yes
No
Not sure
Continue


127.0.0.1:9090/assessment/question
Mental Health Check
Would you be willing to discuss a mental health issue with your direct supervisor(s)?
Yes
No
Continue


127.0.0.1:9090/assessment/complete
Your Mental Health Assessment Results
Our Assessment Suggests You Might Benefit From Support
Based on your responses, professional mental health support could be helpful. Remember, seeking help is a sign of strength.
Personalized Recommendations
Professional Support
Mindfulness Practice
Sleep Hygiene



Milestone 7: Project Demonstration & Documentation

Below mentioned deliverables to be submitted along with other deliverables

Activity 1:- Record explanation Video for project end to end solution

Activity 2:- Project Documentation-Step by step project development procedure

Create document as per the template provided