Mental Health prediction Using Machine learning

Mental Health Prediction Using Machine Learning:

Mental health plays a crucial role in a person's overall well-being. However, due to stress, workload, personal struggles, and lack of awareness, many people suffer from mental health issues without proper diagnosis or treatment. Early detection of such issues can help in providing timely support and improving the quality of life.

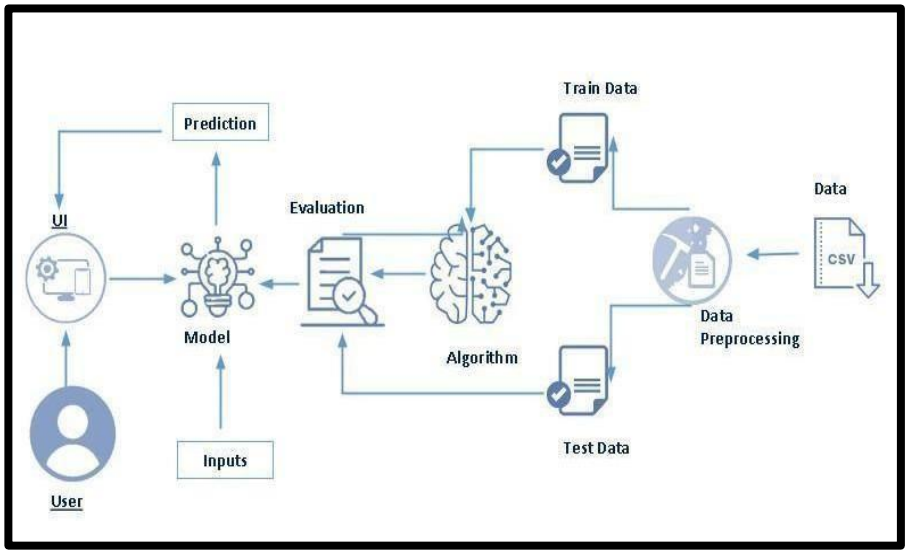
The Mental Health Prediction System uses machine learning techniques to identify whether a person is likely to need mental health treatment based on various factors such as age, gender, work environment, family history, support at the workplace, and more.

The main objective of this project is to:

- Predict mental health conditions using survey or input data.
- Support healthcare professionals and organizations in identifying individuals who may need help.
- Raise awareness and encourage early intervention and support.

This system helps reduce the stigma around mental health, improves workplace well-being, and supports informed decision-making.

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

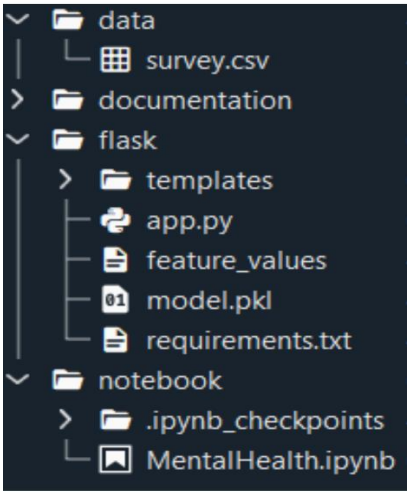
Prior Knowledge:

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

- ML Concepts
    - Supervised learning: https://www.javatpoint.com/supervised-machine-learning
    - Unsupervised learning: https://www.javatpoint.com/unsupervised-machine-learning
    - 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/
- Flask Basics : https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Structure:

Create the Project folder which contains files as shown below



- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
- model.pkl is our saved model. Further we will use this model for flask integration.

- Data Folder contains the Dataset used
- The Notebook file contains procedure for building th model.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem

Refer Project Description

Activity 2: Business requirements

A mental health prediction project can have several important business requirements depending on the end users (e.g., healthcare providers, counselors, or organizations). Below are some key requirements:

- Accurate and Up-to-Date Predictions:
  The system should be built using recent and reliable mental health data to ensure accurate predictions that reflect current behavioral health trends and workplace conditions.

- Flexibility and Scalability:
  The model should be capable of adapting to new data sources, evolving mental health patterns, and different organizational setups (remote work, startups, enterprises, etc.).

- Compliance with Ethical and Legal Standards:
  The project must follow data privacy laws (like HIPAA or GDPR if deployed globally) and ethical guidelines for mental health screening, ensuring users' confidentiality and informed consent.

- User-Friendly Interface:
  The system should be accessible and easy to use for both healthcare professionals and individual users. A simple web interface can help users input basic information and receive results without technical expertise.

- Explainability and Trust:
  The system should offer insights into which factors contributed to a prediction, increasing trust and allowing professionals to make informed decisions based on model outputs.

- Integration Capability:
  The solution should allow easy integration with existing healthcare systems, employee wellness platforms, or university mental health portals.

Activity 3: Literature Survey

A literature survey for a mental health prediction project involves researching and reviewing existing research papers, clinical studies, articles, and publicly available datasets related to mental health detection using machine learning. The objective is to understand the current approaches, methodologies, challenges, and advancements in this domain.

This survey focuses on exploring:

- Existing mental health prediction models used in academic and healthcare sectors.

- Commonly used datasets, such as the OSMI Mental Health in Tech Survey, which offer features related to workplace stress, family history, and treatment-seeking behavior.

- Machine learning techniques like Logistic Regression, Decision Trees, Random Forest, AdaBoost, and Neural Networks that have been applied in mental health diagnosis tasks.

- Strengths and limitations of current models in terms of accuracy, interpretability, fairness, and real-world usability.

- Challenges such as class imbalance, stigma-related reporting bias, and the sensitivity of mental health data.
- Gaps in existing work, such as the lack of real-time deployable models and user-friendly interfaces for non-clinical environments.

Insights from the literature help guide the selection of appropriate algorithms, preprocessing strategies, and evaluation metrics. The findings also support the development of a more practical and ethical mental health prediction system that can be integrated into workplaces, schools, or telehealth platforms.

Activity 4: Social or Business Impact.

Social Impact :-

Early Detection and Intervention:
The mental health prediction system helps identify individuals at risk of mental health issues before conditions become severe. This enables timely support, counseling, or medical intervention, improving overall mental well-being in society.

Reducing Stigma:
By integrating mental health prediction into routine assessments (e.g., at schools, colleges, or workplaces), the system normalizes conversations around mental health, helping reduce societal stigma and encouraging more people to seek help.

Enhanced Workplace Wellness:
Organizations can use the tool to promote mental wellness, implement supportive policies, and reduce burnout—leading to healthier, more productive environments.

- Business Model/Impact:

Preventive Healthcare Cost Savings:
By detecting issues early, employers, insurance companies, and healthcare providers can reduce long-term treatment costs and loss of productivity due to mental health-related absences.

Integration into Health Platforms:
The model can be embedded into digital health solutions or HR wellness platforms as a value-added service, generating revenue through SaaS (Software as a Service) or subscription models.

Policy and Program Design:
Organizations and government bodies can use the insights from aggregated predictions to design targeted mental health programs or interventions based on data-driven trends.

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/osmi/mental-health-in-tech-survey

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 numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, GradientBoosti
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report, confusion_matrix
import warnings
import pickle
from scipy import stats
warnings.filterwarnings('ignore')
plt.style.use('fivethirtyeight')
```

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.

```
#1.Loading Data Set

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
data = pd.read_csv(r"C:\Users\jaind\OneDrive\Desktop\ML_Project\data\survey.csv")

data
```

	Timestamp	Age	Gender	Country	state	self_employed	family_history	treatment	work_interfere	no_employees	...	leave	mental_health_consequence
0	2014-08-27 11:29:31	37	Female	United States	IL	NaN	No	Yes	Often	6-25	...	Somewhat easy	No
1	2014-08-27 11:29:37	44	M	United States	IN	NaN	No	No	Rarely	More than 1000	...	Don't know	Maybe
2	2014-08-27 11:29:44	32	Male	Canada	NaN	NaN	No	No	Rarely	6-25	...	Somewhat difficult	No
3	2014-08-27 11:29:46	31	Male	United Kingdom	NaN	NaN	Yes	Yes	Often	26-100	...	Somewhat difficult	Yes
4	2014-08-27 11:30:22	31	Male	United States	TX	NaN	No	No	Never	100-500	...	Don't know	No
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1254	2015-09-12 11:17:21	26	male	United Kingdom	NaN	No	No	Yes	NaN	26-100	...	Somewhat easy	No

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.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.

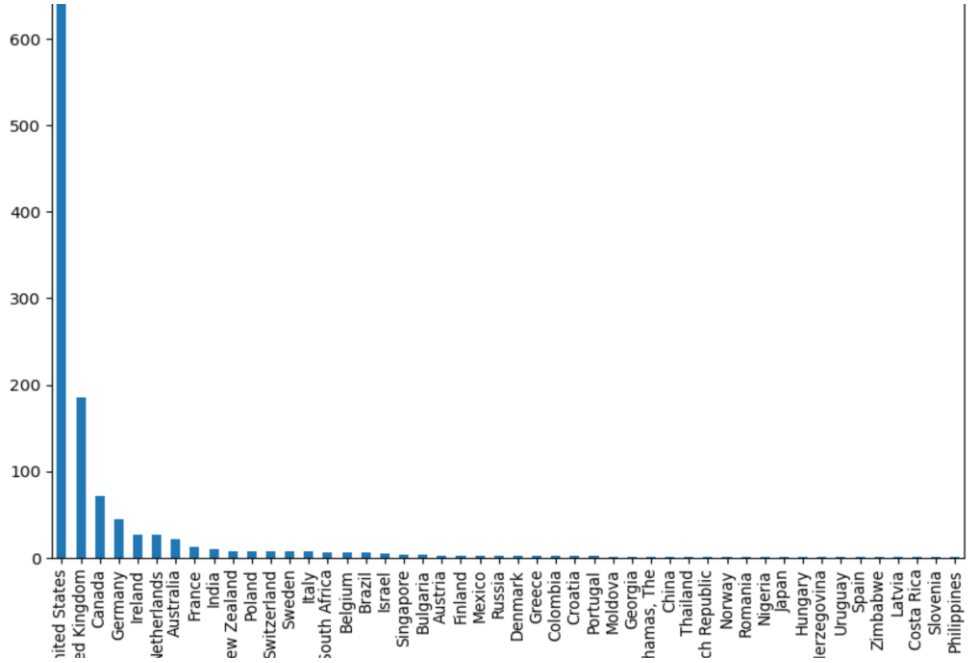
```
#Handling Null values
data.isnull().sum()
```

```
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
```

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 policy_annual_premium feature with some mathematical formula.

- From the below diagram, we could visualize that policy_annual_premium feature has outliers. Boxplot from seaborn library is used here.



.

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive statistical

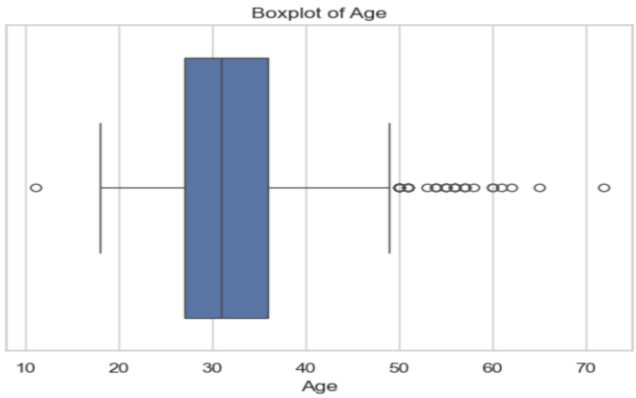
Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values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

	Age	Gender	self_employed	family_history	treatment	work_interfere	no_employees	remote_work	tech_company	benefits	...	seek_help
count	1252.000000	1252	1252	1252	1252	1252	1252	1252	1252	1252	...	1252
unique	NaN	38	2	2	2	4	6	2	2	3	...	3
top	NaN	Male	No	No	Yes	Sometimes	6-25	No	Yes	Yes	...	No
freq	NaN	820	1109	763	632	726	289	880	1026	473	...	642
mean	32.059904	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
std	7.309669	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
min	11.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
25%	27.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
50%	31.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
75%	36.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
max	72.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN

11 rows × 22 columns

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.



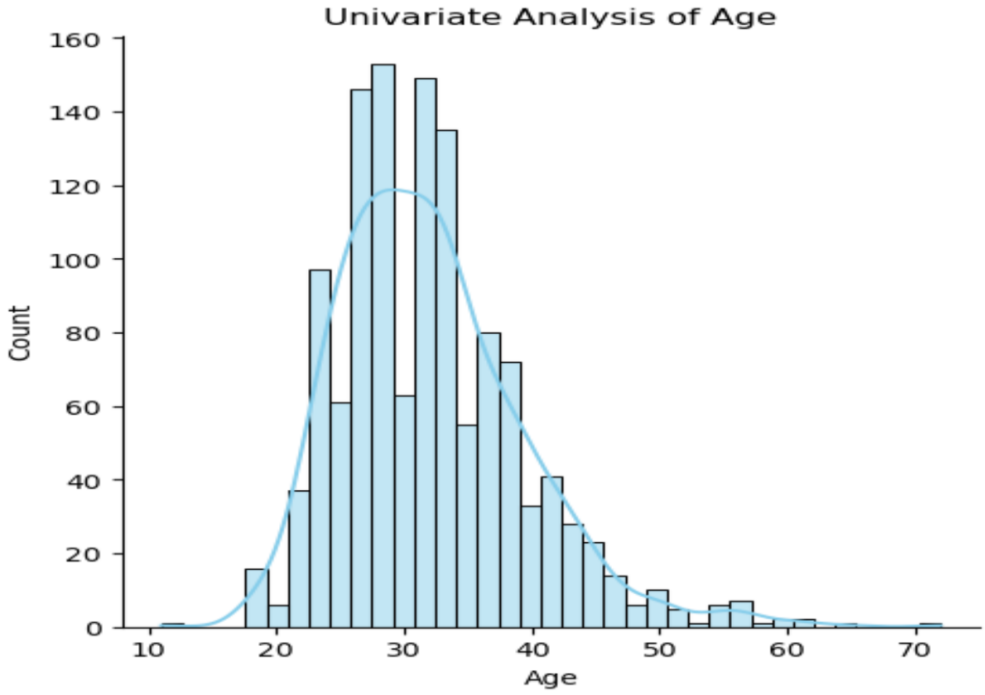
Activity 2.1: Univariate analysis

Univariate analysis means understanding one feature (column) at a time. It helps us explore how values are distributed in a single feature. In this project, we used two types of graphs — Countplot and Pie Chart — to analyze individual features.

The Seaborn library provides a useful function called countplot, which is great for categorical features like gender, family history, or work interference. It shows how many times each value appears in a feature.

For example, using a countplot on the treatment column (whether a person has sought mental health treatment), we found that:

- A large number of people have not taken treatment.
- A smaller group has taken treatment — showing a possible lack of awareness or support.



Activity 2.2: Bivariate analysis

Bivariate Analysis: Employment Type vs Treatment

The graph shows the relationship between a person's **employment type** (whether they are self-employed or not) and whether they have **sought treatment** for mental health.

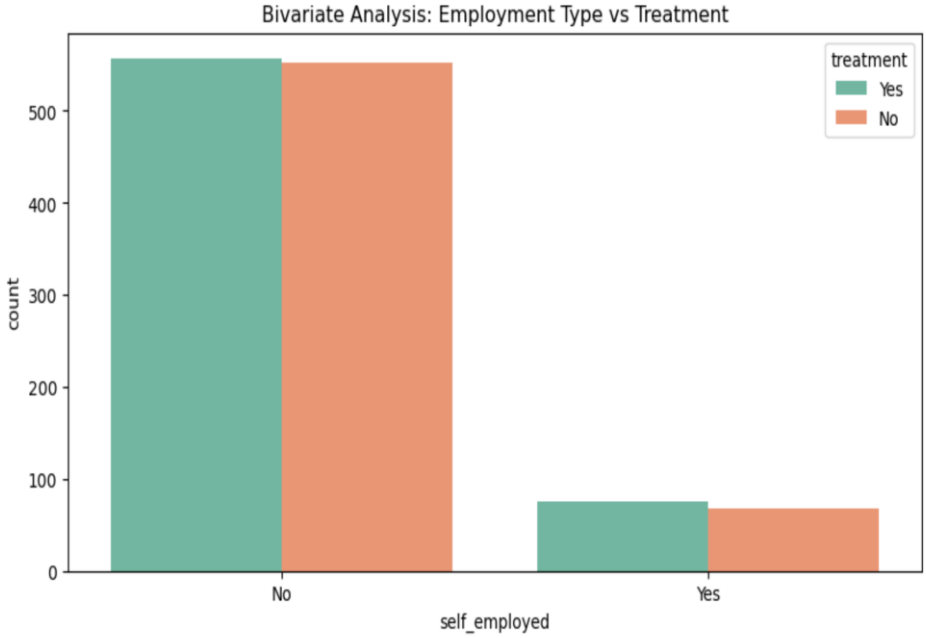
X-axis (horizontal): Shows if the person is self-employed (Yes or No)

Y-axis (vertical): Shows the number of people (count)

Bars: Each group (Yes or No) is split into two colored bars:

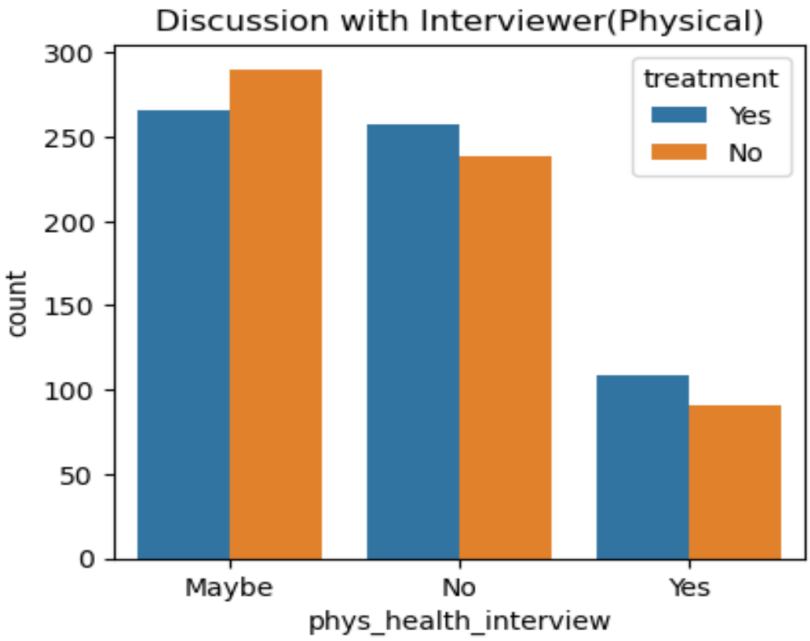
▨ Green (Yes): People who did take mental health treatment

▨ Orange (No): People who did not take treatmentare of 50,000 to 60,000 from the years 1995 to 2015.



```
#Physical health discussion during interview and treatment
plt.figure(figsize=(10,40))
plt.subplot(9,2,17)
sb.countplot(x='phys_health_interview', hue='treatment', data=data)
plt.title('Discussion with Interviewer(Physical)')
```

```
Text(0.5, 1.0, 'Discussion with Interviewer(Physical)')
```



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.

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.

```
X = data.drop('treatment', axis=1)
y = data['treatment']

# ✅ Step 4: Encode the target variable
le = LabelEncoder()
y = le.fit_transform(y)

# ✅ Step 5: Identify numeric and categorical features
num_cols = ['Age']
cat_cols = [col for col in X.columns if col not in num_cols]

# ✅ Step 6: Define column transformer (scaling + encoding)
ct = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), num_cols),
        ('cat', OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1), cat_cols)
    ]
)
```

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.

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

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

```
ct = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), num_cols),
        ('cat', OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1), cat_cols
    ]
)

# ✅ Step 7: Apply transformations to the features
X_transformed = ct.fit_transform(X)
```

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.

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.

```
from sklearn.tree import DecisionTreeClassifier
dec=DecisionTreeClassifier(random_state=49)
base = DecisionTreeClassifier(max_depth=3)
dec.fit(X_train,y_train)
pred_abc = dec.predict(X_test)
print(classification_report(y_test,pred_abc))
print(confusion_matrix(y_test,pred_abc))
```

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.

```
from sklearn.ensemble import RandomForestClassifier
rad=RandomForestClassifier(random_state=49)
base = DecisionTreeClassifier(max_depth=3)
rad.fit(X_train,y_train)
pred_abc = rad.predict(X_test)
print(classification_report(y_test,pred_abc))
print(confusion_matrix(y_test,pred_abc))
```

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.

```
kn=KNeighborsClassifier()
base = DecisionTreeClassifier(max_depth=3)
kn.fit(X_train,y_train)
pred_abc = kn.predict(X_test)
print(classification_report(y_test,pred_abc))
print(confusion_matrix(y_test,pred_abc))
```

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.

```
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegressionCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, classification_report
lg = LogisticRegressionCV(solver='lbfgs', max_iter=5000, cv=10)
lg.fit(X_train, y_train)
print(confusion_matrix(y_test,lrg_pred))
```

Activity 1.5: Ada Boost Model

Ada Boost is imported from sklearn Library then Adaboostalgorithm 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. We can find the Train and Test accuracy by X_train and X_test.

```
abc = AdaBoostClassifier(random_state=99)
base = DecisionTreeClassifier(max_depth=3)
abc.fit(X_train,y_train)
pred_abc = abc.predict(X_test)
print(classification_report(y_test,pred_abc))
print(confusion_matrix(y_test,pred_abc))
```

Activity 1.6: Gradient Boost

Gradient Boost Model is imported from sklearn Library then GB algorithm is initialized 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.

```
gb=GradientBoostingClassifier(random_state=49)
base = DecisionTreeClassifier(max_depth=3)
gb.fit(X_train,y_train)
pred_abc = gb.predict(X_test)
print(classification_report(y_test,pred_abc))
print(confusion_matrix(y_test,pred_abc))
```

Activity 2: Testing the model

Here we have tested with AdaBoost. You can test with all algorithm. With the help of predict() function.

```
#best accuracy is Adaboost
abc = AdaBoostClassifier(random_state=99)
base = DecisionTreeClassifier(max_depth=3)
abc.fit(X_train,y_train)
pred_abc = abc.predict(X_test)
print('Accuracy of AdaBoost=',accuracy_score(y_test,pred_abc))
```

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.

```
for name, model in model_dict.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print("="*30 + f" {name} " + "="*30)
    print(f"Accuracy: {accuracy:.4f}")
    print("Classification Report:")
    print(classification_report(y_test, y_pred))
    print("\n")
```

Activity 2: Comparing model accuracy before & after applying Grid Search(

Evaluating performance of the model From sklearn, cross_val_score is used to evaluate the score of the model. On the parameters, we have given rf (model name), x, y, cv (as 5 folds). Our model is performing well.

```
Cross-Validation Scores (5-Fold):

Logistic regression:
  Accuracy Scores: [0.70916335 0.74501992 0.704      0.736      0.696     ]
  Mean Accuracy: 0.7180
  Std Deviation: 0.0190
--------------------------------------------------
KNN Classifier:
  Accuracy Scores: [0.58964143 0.66135458 0.636      0.656      0.616     ]
  Mean Accuracy: 0.6318
  Std Deviation: 0.0265
--------------------------------------------------
Decision Tree Classifier:
  Accuracy Scores: [0.70119522 0.65737052 0.672      0.632      0.68      ]
  Mean Accuracy: 0.6685
  Std Deviation: 0.0231
--------------------------------------------------
Random Forest Classifier:
  Accuracy Scores: [0.74900398 0.76095618 0.728      0.72       0.728     ]
  Mean Accuracy: 0.7372
  Std Deviation: 0.0153
--------------------------------------------------
AdaBoost Classifier:
  Accuracy Scores: [0.7689243  0.79282869 0.72       0.732      0.74      ]
  Mean Accuracy: 0.7508
  Std Deviation: 0.0265
--------------------------------------------------
Gradient Boosting Classifier:
  Accuracy Scores: [0.73306773 0.77290837 0.704      0.736      0.748     ]
  Mean Accuracy: 0.7388
  Std Deviation: 0.0224
--------------------------------------------------
```

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.

```
import pickle, joblib
best_model = grid.best_estimator_

pickle.dump(best_model, open("model.pkl", "wb"))
joblib.dump(ct, "feature_values.pkl")
joblib.dump(le, "label_encoder.pkl")
```

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. R

Activity 2.2: Build Python code:

Import the libraries

```
from flask import Flask, render_template, request
import pickle
import joblib
import pandas as pd

app = Flask(__name__, template_folder='templates')
```

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.

```
model = pickle.load(open("model.pkl", "rb"))
ct = joblib.load("feature_values.pkl")
```

Render HTML page:

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

@app.route('/pred')  # Proceed to form
def pred():
    return render_template("index.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:

```
def output():
    try:
        # ✅ Get and validate age
        age_value = request.form["age"]
        try:
            age = float(age_value)
        except ValueError:
            return f"❌ Error: Invalid age value '{age_value}'"

        # ✅ Extract form inputs (without 'Leave')
        data = [[
            age,
            request.form["gender"],
            request.form["self_employed"],
            request.form["family_history"],
            request.form["work_interfere"],
            request.form["no_employees"],
            request.form["remote_work"],
            request.form["tech_company"],
            request.form["benefits"],
            request.form["care_options"],
            request.form["wellness_program"],
            request.form["seek_help"],
            request.form["anonymity"],
            request.form["mental_health_consequence"],
            request.form["phys_health_consequence"],
            request.form["coworkers"],
            request.form["supervisor"],
            request.form["mental_health_interview"],
            request.form["phys_health_interview"],
            request.form["mental_vs_physical"],
            request.form["obs_consequence"]
        ]]

        # ✅ Column names — ensure matches what model expects
        feature_cols = [
            'Age', 'Gender', 'self_employed', 'family_history', 'work_interfere',
            'no_employees', 'remote_work', 'tech_company', 'benefits', 'care_options',
            'wellness_program', 'seek_help', 'anonymity',
            'mental_health_consequence', 'phys_health_consequence', 'coworkers',
            'supervisor', 'mental_health_interview', 'phys_health_interview',
            'mental_vs_physical', 'obs_consequence'
        ]

        # ✅ Convert to DataFrame
        df = pd.DataFrame(data, columns=feature_cols)

        # ✅ Transform and predict
        transformed = ct.transform(df)
        prediction = model.predict(transformed)[0]

        result = "requires" if prediction == 1 else "doesn't require"
        return render_template("output.html", y=f"This person {result} mental health treatment.")

    except Exception as e:
        return f"❌ Error: {str(e)}"
```

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:

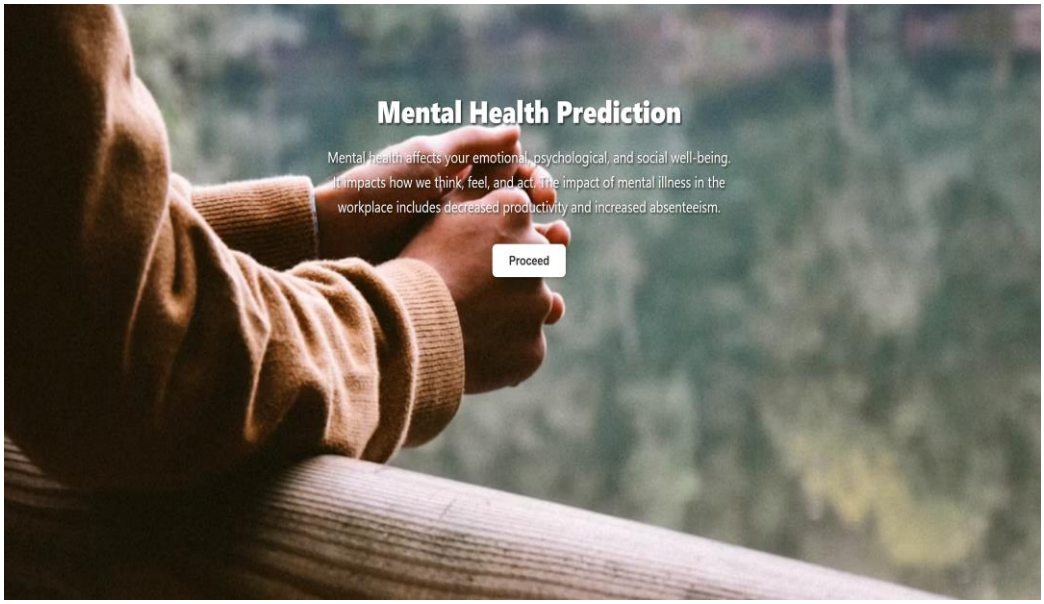
```
if __name__ == "__main__":
    app.run(debug=True, port=5000)
```

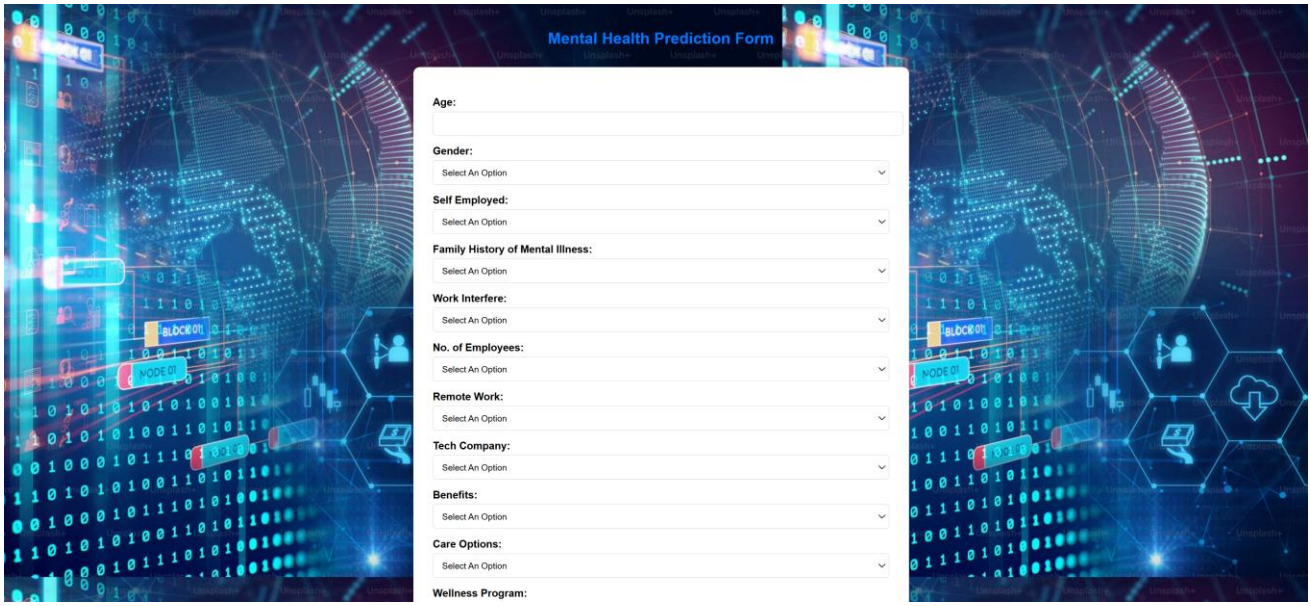
Activity 2.3: Run the web application

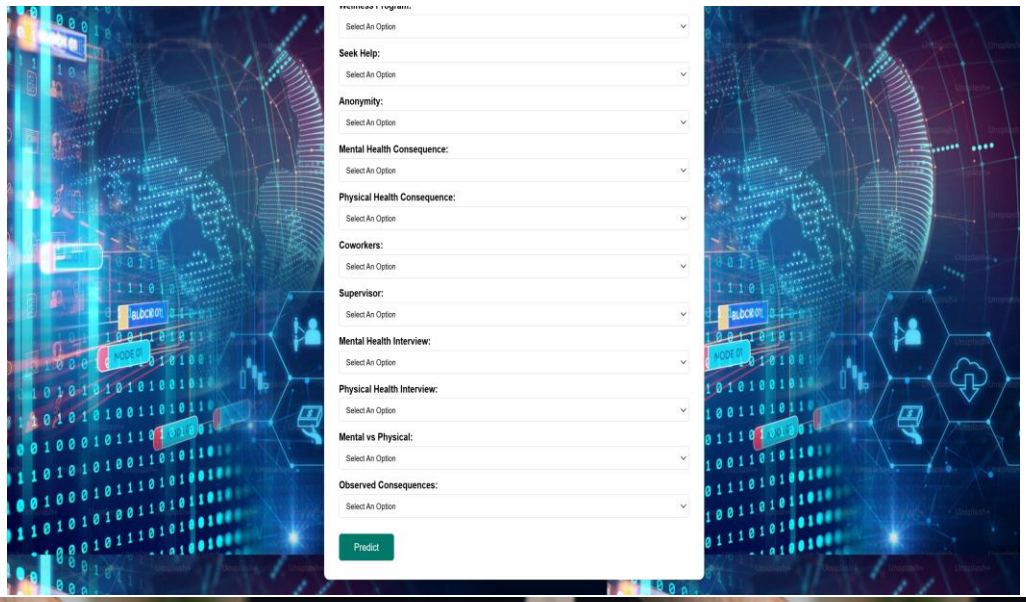
- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top left corner, enter the inputs, click on the submit button, and see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

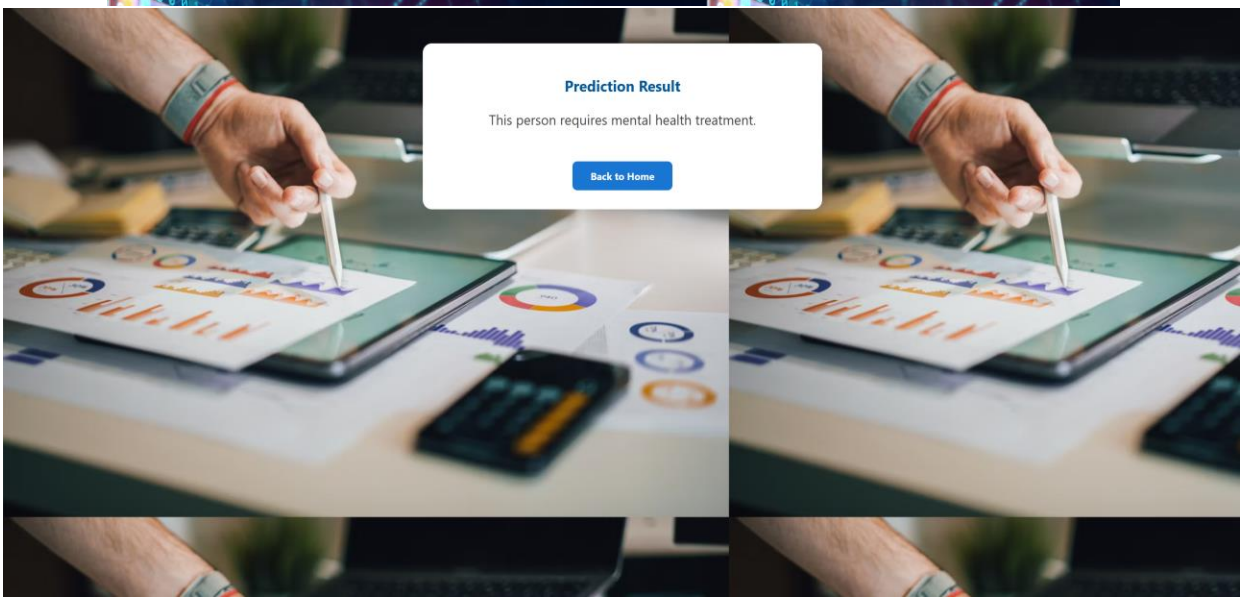
```
(base) C:\Users\jaind>cd "C:\Users\jaind\OneDrive\Desktop\ML_Project\Flask"

(base) C:\Users\jaind\OneDrive\Desktop\ML_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 http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 875-543-388
```

Now,Go the web browser and write the localhost url (http://127.0.0.1:5000) to get the below result




Mental Health Prediction Form
Age:
Gender:
Select An Option
Self Employed:
Select An Option
Family History of Mental Illness:
Select An Option
Work Interfere:
Select An Option
No. of Employees:
Select An Option
Remote Work:
Select An Option
Tech Company:
Select An Option
Benefits:
Select An Option
Care Options:
Select An Option
Wellness Program:


Select An Option
Seek Help:
Select An Option
Anonymity:
Select An Option
Mental Health Consequence:
Select An Option
Physical Health Consequence:
Select An Option
Coworkers:
Select An Option
Supervisor:
Select An Option
Mental Health Interview:
Select An Option
Physical Health Interview:
Select An Option
Mental vs Physical:
Select An Option
Observed Consequences:
Select An Option
Predict


Prediction Result
This person requires mental health treatment.
Back to Home

Milestone 7: Project Demonstration & Documentation

Git Hub Link: https://github.com/DavidJain/Mental-Health-Prediction

Demonstration Link: https://www.youtube.com/watch?v=mheDDjMOYLY