Mental Health Prediction Using ML

Project Description:

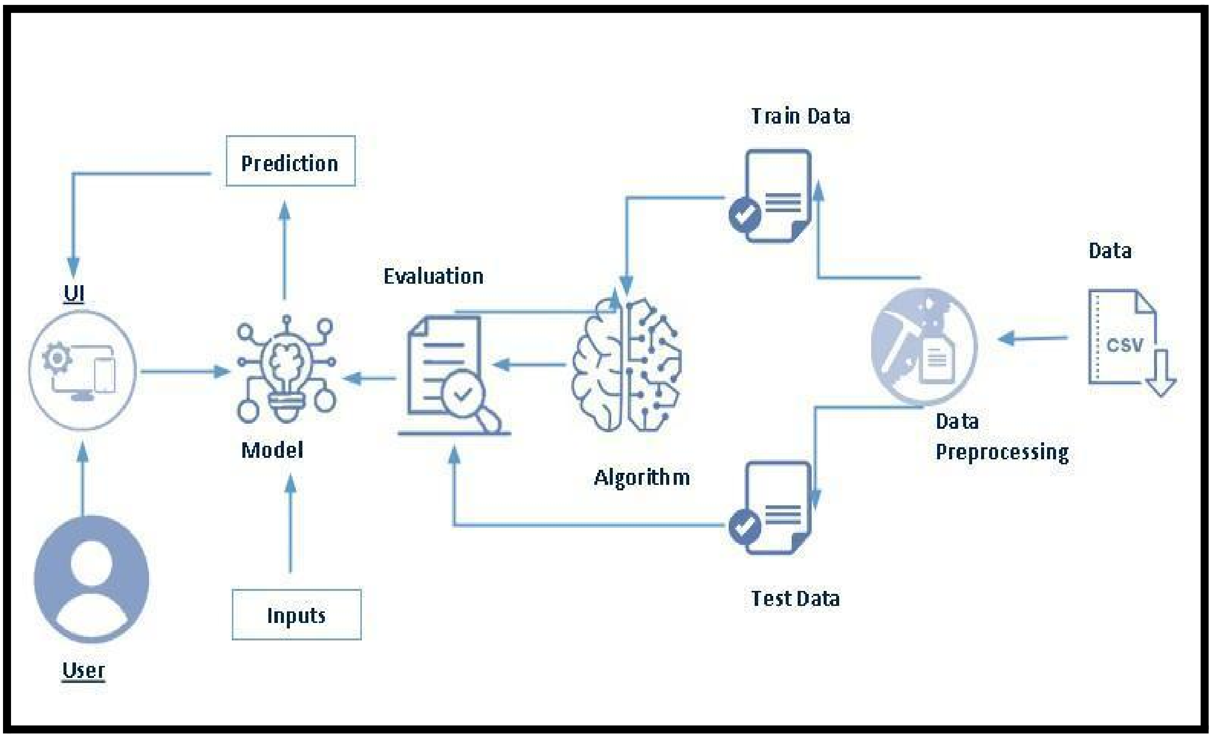
Mental Health First Aid teaches participants how to notice and support an individual who may be experiencing a mental health or substance use concern or crisis and connect them with the appropriate employee resources.

Employers can offer robust benefit packages to support employees who go through mental health issues. That includes Employee Assistance Programs, Wellness programs that focus on mental and physical health, Health and Disability Insurance or flexible working schedules or time off policies.Organisations that incorporate mental health awareness help to create a healthy and productive work environment that reduces the stigma associated with mental illness, increases the organisations mental health literacy and teaches the skills to safely and responsibly respond to a co-workers mental health concern.

The main purpose of the Mental Health Prediction system is to predict whether a person needs to seek Mental health treatment or not based on inputs provided by them.

We will be using classification algorithms such as Logistic Regression, KNN, Decision tree, Random forest, AdaBoost, GradientBoost and XGBoost. We will train and test the data with these algorithms. From this the best model is selected and saved in pkl format. We will also be deploying our model locally using Flask.

Technical Architecture:



Pre requisites:

To complete this project, you will require the following softwares, concepts and packages

- **Anaconda navigator:**
  - Refer the link below to download anaconda navigator
  - Link : https://youtu.be/1ra4zH2G4o0
- **Python packages:**
  - Open anaconda prompt as administrator
  - Type “pip install numpy” and click enter.
  - Type “pip install pandas” and click enter.
  - Type “pip install scikit-learn” and click enter.
  - Type ”pip install matplotlib” and click enter.
  - Type ”pip install scipy” and click enter.
  - Type ”pip install pickle-mixin” and click enter.
  - Type ”pip install seaborn” and click enter.
  - Type “pip install Flask” and click enter.

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
  - Regression and classification
    - Logistic regression: https://www.javatpoint.com/logistic-regression-in-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/
    - AdaBoost: https://www.analyticsvidhya.com/blog/2021/09/adaboost-algorithm-a-c

omplete-guide-for-beginners/

- Gradient Boost: https://www.analyticsvidhya.com/blog/2021/09/gradient-boosting-algorithm-a-complete-guide-for-beginners/
- 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 Objectives:

By the end of this project you will:

- Know fundamental concepts and techniques used for machine learning.
- Gain a broad understanding about data.
- Have knowledge on pre-processing the data/transformation techniques and some visualisation concepts before building the model
- Learn how to build a machine learning model and tune it for better performance
- Know how to evaluate the model and deploy it using flask

Project Flow:

- User interacts with the UI to enter the input.
- Entered input is analysed by the model which is integrated.
- The predictions made by the model is showcased on the UI

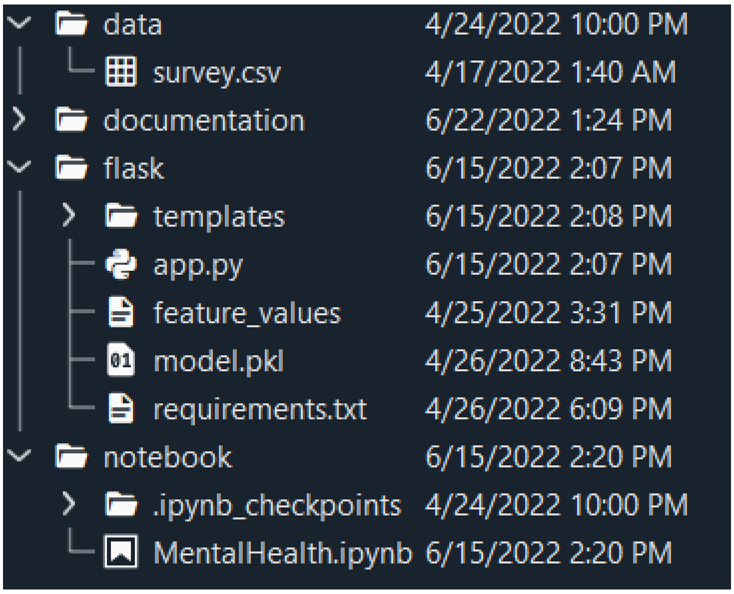
To accomplish this, we have to complete all the activities listed below,

- Data collection
  - Collect the dataset or create the dataset
- Data pre-processing
  - Removing unnecessary columns
  - Checking for null values
- Visualising and analysing data
  - Univariate analysis
  - Bivariate analysis
  - Descriptive analysis
- Model building
  - Handling categorical values
  - Dividing data into train and test sets
  - Import the model building libraries
  - Comparing accuracy of various models
  - Hyperparameter tuning of the selected model
  - Evaluating performance of models

    - Save the model
- Application Building
    - Create an HTML file
    - Build python code

Project Structure:

Create the Project folder which contains files as shown below



- data folder contains the .csv file used for our project
- We are building a flask application which 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.

Milestone 1: Data Collection

Activity 1: Download the dataset

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

In this project we have used survey.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

Load the dataset using read_csv() function:

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

```python
data = pd.read_csv('D:/SB_Projects/Mental Health Prediction using ML/data/survey.csv')
```

Inside the read_csv() function, specify the path to your dataset.

To observe the first 5 rows of our data, we use the head() method and to observe the last 5 rows of the data, we use the tail() method.

```python
data.head()
```

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

```python
data.tail()
```

	Timestamp	Age	Gender	Country	state	self_employed	family_history	treatment	work_interfere	no_employees	...	leave	mental_health_cons
1254	2015-09-12 11:17:21	26	male	United Kingdom	NaN	No	No	Yes	NaN	26-100	...	Somewhat easy	
1255	2015-09-26 01:07:35	32	Male	United States	IL	No	Yes	Yes	Often	26-100	...	Somewhat difficult	
1256	2015-11-07 12:36:58	34	male	United States	CA	No	Yes	Yes	Sometimes	More than 1000	...	Somewhat difficult	
1257	2015-11-30 21:25:06	46	f	United States	NC	No	No	No	NaN	100-500	...	Don't know	
1258	2016-02-01 23:04:31	25	Male	United States	IL	No	Yes	Yes	Sometimes	26-100	...	Don't know	

We can use the 'shape' attribute of the dataframe to know the shape of our dataset:

```python
data.shape
```

```
(1259, 27)
```

From the above figure, we can say that our dataset has 1259 rows and 27 columns

Next, we will have to see the information pertaining to each of the 27 columns. For that, we will be using info() function:

```
1 data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1259 entries, 0 to 1258
Data columns (total 27 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Timestamp                  1259 non-null   object
 1   Age                        1259 non-null   int64
 2   Gender                     1259 non-null   object
 3   Country                    1259 non-null   object
 4   state                      744 non-null    object
 5   self_employed              1241 non-null   object
 6   family_history             1259 non-null   object
 7   treatment                  1259 non-null   object
 8   work_interfere             995 non-null    object
 9   no_employees               1259 non-null   object
 10  remote_work                1259 non-null   object
 11  tech_company               1259 non-null   object
 12  benefits                   1259 non-null   object
 13  care_options               1259 non-null   object
 14  wellness_program           1259 non-null   object
 15  seek_help                  1259 non-null   object
 16  anonymity                  1259 non-null   object
 17  leave                      1259 non-null   object
 18  mental_health_consequence  1259 non-null   object
 19  phys_health_consequence    1259 non-null   object
 20  coworkers                  1259 non-null   object
 21  supervisor                 1259 non-null   object
 22  mental_health_interview    1259 non-null   object
 23  phys_health_interview      1259 non-null   object
 24  mental_vs_physical         1259 non-null   object
 25  obs_consequence            1259 non-null   object
 26  comments                   164 non-null    object
dtypes: int64(1), object(26)
memory usage: 265.7+ KB
```

Milestone 2: Data Pre-processing

We need to pre-process the collected data before gaining insights and building our model.

We need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

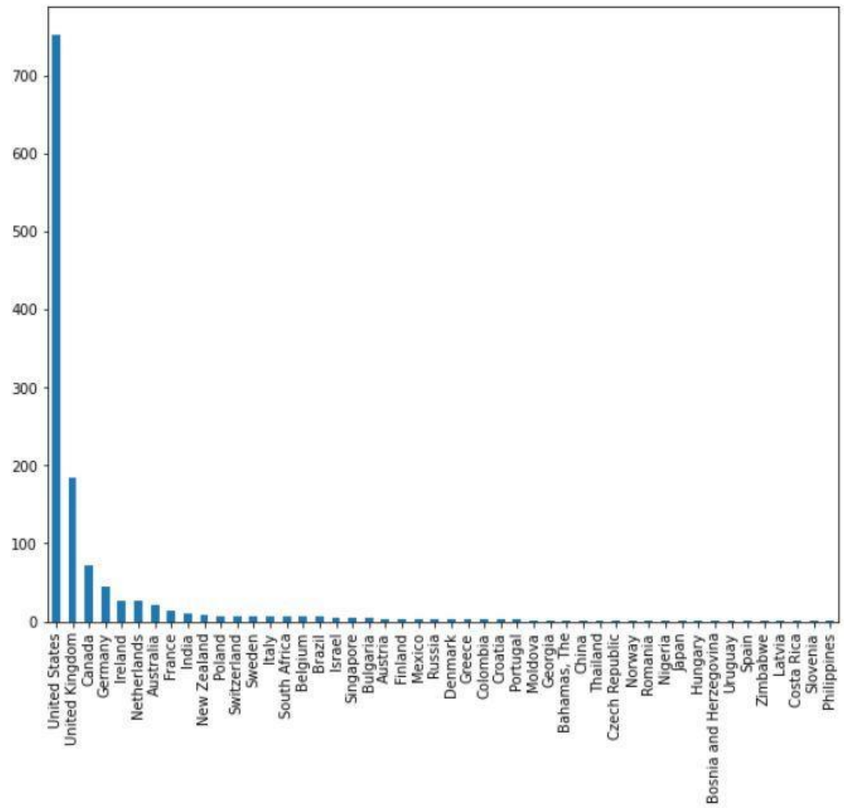
- Removing unnecessary columns
- Handling Null values and dealing with wrongly entered data

Activity 1: Removing unnecessary columns:

- The below picture shows the distribution of countries

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

]: <AxesSubplot:>



Since the countries are not evenly distributed, keeping this column will induce bias in our model. So we will be removing country and state columns. We will also remove timestamp and comments columns as they do not contribute to providing relevant information.

Activity 2: Handling Null values and dealing with wrongly entered data

To check for null values, .isnull() function is used along with .sum() function to the dataframe.

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

```
Age                            0
Gender                         0
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
dtype: int64
```

We observe that 2 columns - self_employed and work_interfere contain null values. Let us fill the self_employed column with 'No' and work_interfere column with 'N/A' in place of null values. In order to do this, we will make use of .fillna() function

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

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

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

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

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

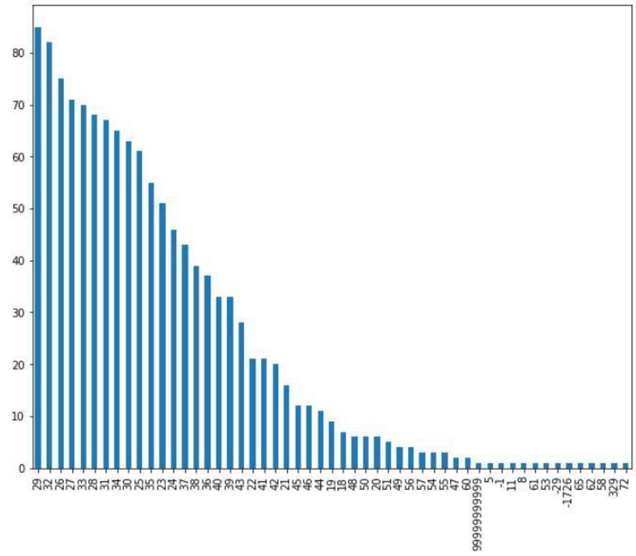
```
data['work_interfere'].fillna('N/A',inplace=True)
```

Now our dataset is free of null values.

Let us now handle data that might have been entered wrongly. Consider the Age column of our dataset.

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

<AxesSubplot:>



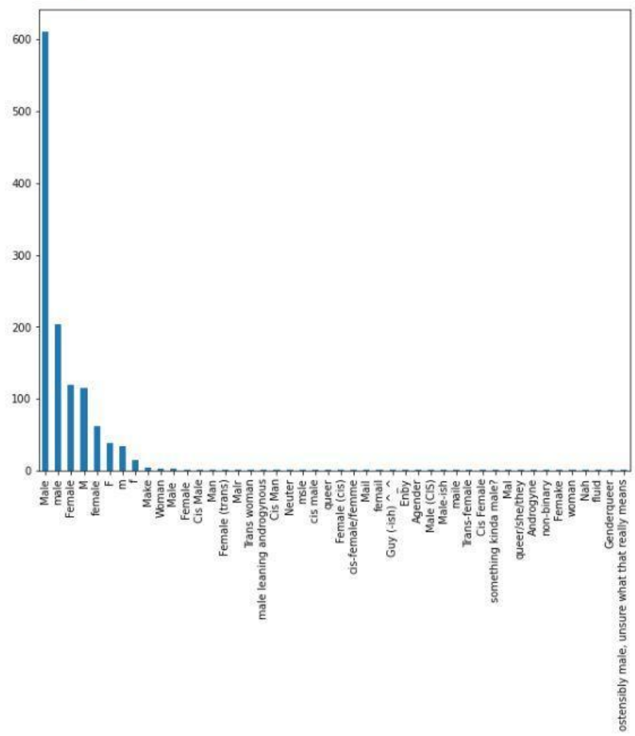
If we observe the Age column, it is seen that some impractical values have been entered. So, let us remove the rows with ages greater than 60 and less than 18 using the code below.

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

Next, consider the Gender column of our dataset.

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

<AxesSubplot:>



It is observed that different names are used for the same category of gender. Let us group them into 3 major categories- Male, Female and Non-Binary using the .replace() function.

```
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)
```

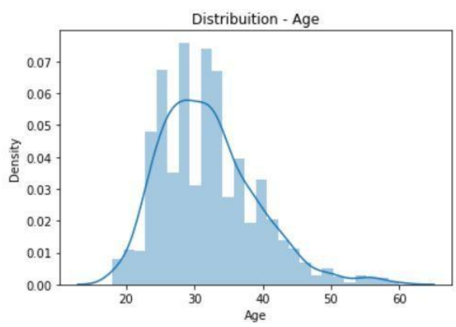
Milestone 3: Data analysis and visualisation

Activity 1: Univariate analysis

In simple words, univariate analysis is understanding the data with a single feature.

- Seaborn package provides a function called distplot, which helps us to find the distribution of specific features in our dataset. Let us observe the distribution of age.

```
sb.distplot(data["Age"])
plt.title("Distribuition - Age")
plt.xlabel("Age")
```

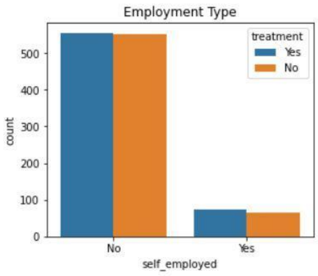


Activity 2: Bivariate analysis

We use bivariate analysis to find the relation between two features. Here we are visualising the relationship of various features with respect to treatment, which is our target variable.

- Employment type and treatment

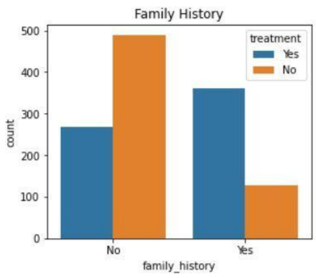
```
plt.figure(figsize=(10,40))
plt.subplot(9,2,1)
sb.countplot(data['self_employed'], hue = data['treatment'])
plt.title('Employment Type')
```



We observe that though there is a vast difference between people who are self employed or not, the number of people who seek treatment in both the categories is more or less similar.

- Family history and treatment

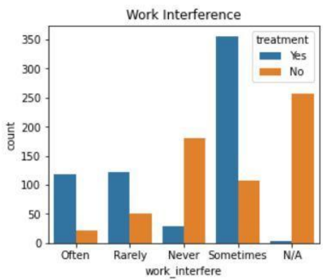
```
plt.figure(figsize=(10,40))
plt.subplot(9,2,2)
sb.countplot(data['family_history'], hue = data['treatment'])
plt.title('Family History')
```



We observe that treatment is directly proportional to family history. Hence this is an important factor.

- Work interference and treatment

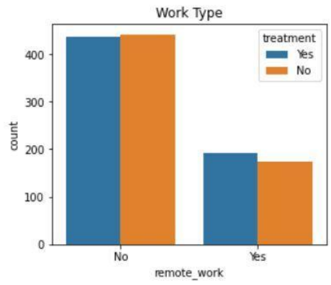
```
plt.figure(figsize=(10,40))
plt.subplot(9,2,3)
sb.countplot(data['work_interfere'], hue = data['treatment'])
plt.title('Work Interference')
```



We observe that the people who chose Sometimes were the largest who wanted to get treatment. These group of people are the ones who are reluctant to choose either of the extreme categories.

- Work type and treatment

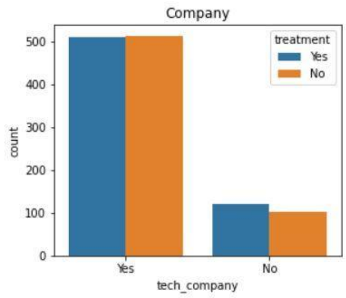
```
plt.figure(figsize=(10,40))
plt.subplot(9,2,4)
sb.countplot(data['remote_work'], hue = data['treatment'])
plt.title('Work Type')
```



We observe that the number of people who seek treatment in both the categories is more or less similar and it does not affect our target variable.

- Company and treatment

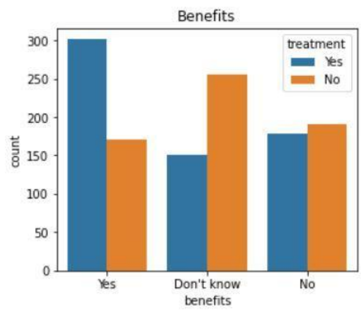
```
plt.figure(figsize=(10,40))
plt.subplot(9,2,5)
sb.countplot(data['tech_company'], hue = data['treatment'])
plt.title('Company')
```



We can conclude that irrespective of the field the company of the people falls in, mental health is a big issue.

- Benefits and treatment

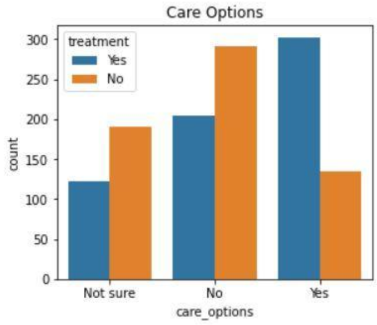
```
plt.figure(figsize=(10,40))
plt.subplot(9,2,6)
sb.countplot(data['benefits'], hue = data['treatment'])
plt.title('Benefits')
```



We see that a large group among the people who wanted mental health benefits wanted to seek treatment and also a significant number of people who said No too, wanted to seek treatment.

- Care options and treatment

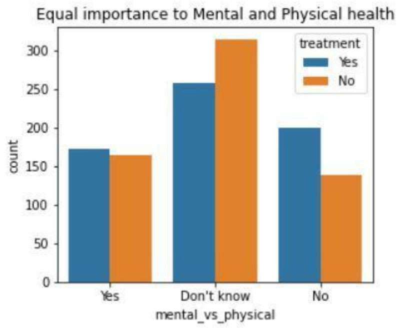
```
plt.figure(figsize=(10,40))
plt.subplot(9,2,7)
sb.countplot(data['care_options'], hue = data['treatment'])
plt.title('Care Options')
```



This graph is quite similar to the benefits column.

- Mental vs Physical health

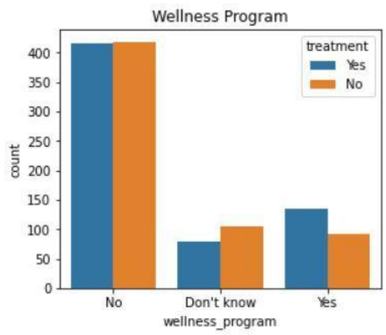
```
plt.figure(figsize=(10,40))
plt.subplot(9,2,8)
sb.countplot(data['mental_vs_physical'], hue = data['treatment'])
plt.title('Equal importance to Mental and Physical health')
```



We observe that half of the people are not aware of the importance given to mental health as compared to physical health, whereas almost equal parts of the other halves answered Yes and No.

- Wellness program and treatment

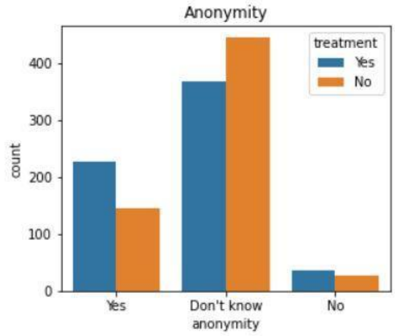
```
plt.figure(figsize=(10,40))
plt.subplot(9,2,9)
sb.countplot(data['wellness_program'], hue = data['treatment'])
plt.title('Wellness Program')
```



We observe that almost half of the people who said No want to seek treatment, which means their company has to take steps in arranging for the same.

- Anonymity and treatment

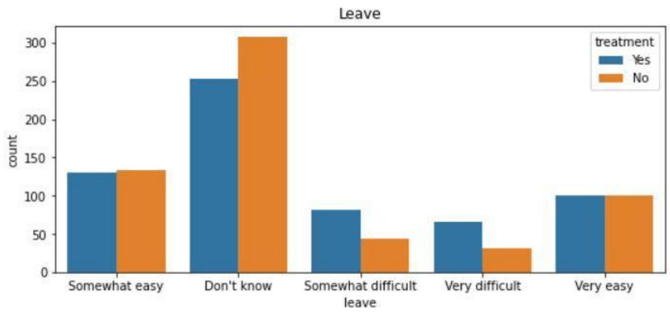
```
plt.figure(figsize=(10,40))
plt.subplot(9,2,10)
sb.countplot(data['anonymity'], hue = data['treatment'])
plt.title('Anonymity')
```



We observe that most people either answered yes or they are not aware if their anonymity will be protected.

- Leave and treatment

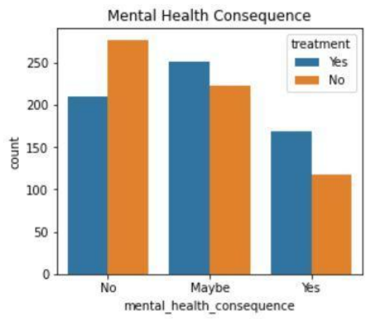
```
plt.figure(figsize=(20,40))
plt.subplot(9,2,11)
sb.countplot(data['leave'], hue = data['treatment'])
plt.title('Leave')
```



We see that around half of the total people don't know how easy it is to get a leave due to a mental health condition and they are the ones who want to seek treatment the most.

- Mental health consequence and treatment

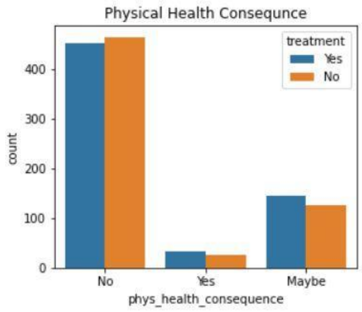
```
plt.figure(figsize=(10,40))
plt.subplot(9,2,12)
sb.countplot(data['mental_health_consequence'], hue = data['treatment'])
plt.title('Mental Health Consequence')
```



We observe that the majority answered either No or Maybe but around 1/3rd of the people answered yes and they want to seek treatment.

- Physical health consequence and treatment

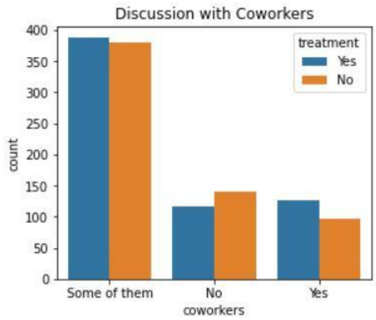
```
plt.figure(figsize=(10,40))
plt.subplot(9,2,13)
sb.countplot(data['phys_health_consequence'], hue = data['treatment'])
plt.title('Physical Health Consequnce')
```



As completely opposed to the above results, a very small number of people answered yes to this question, which means that a major number of people do not face negative consequences by discussing their physical health conditions with their employers.

- Discussion with coworkers and treatment

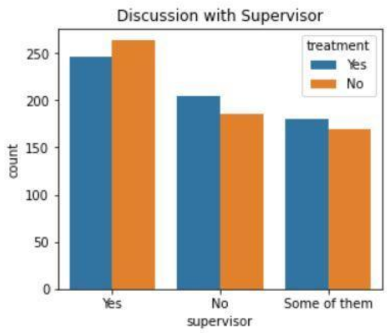
```
plt.figure(figsize=(10,40))
plt.subplot(9,2,14)
sb.countplot(data['coworkers'], hue = data['treatment'])
plt.title('Discussion with Coworkers')
```



We can observe that more than half of the people were willing to discuss their mental health problems with some or all of their coworkers.

- Discussion with supervisor and treatment

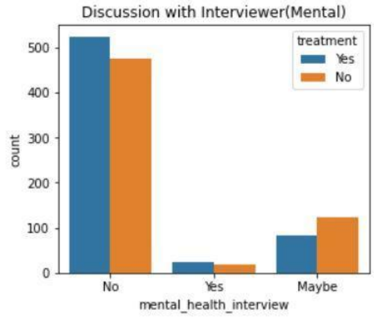
```
plt.figure(figsize=(10,40))
plt.subplot(9,2,15)
sb.countplot(data['supervisor'], hue = data['treatment'])
plt.title('Discussion with Supervisor')
```



This graph again, is in contrast to the above one. Though majority of people are comfortable with discussing their physical health problems with their supervisor, about 1/3rd of them aren't comfortable with it.

- Mental health discussion during interview and treatment

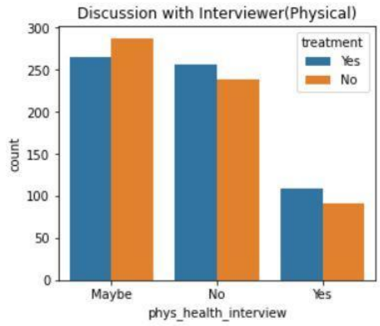
```
plt.figure(figsize=(10,40))
plt.subplot(9,2,16)
sb.countplot(data['mental_health_interview'], hue = data['treatment'])
plt.title('Discussion with Interviewer(Mental)')
```



We observe that very few people are comfortable in bringing up their mental health issue in front of a potential employer.

- Physical health discussion during interview and treatment

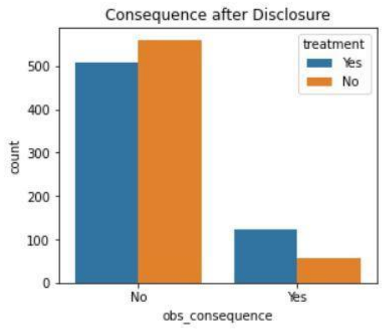
```
plt.figure(figsize=(10,40))
plt.subplot(9,2,17)
sb.countplot(data['phys_health_interview'], hue = data['treatment'])
plt.title('Discussion with Interviewer(Physical)')
```



Though this graph is similar to the above one with respect to not being comfortable in bringing up a physical health issue with a potential employer, a good number of people may be willing to or are willing to bring it up.

- Consequence after disclosure and treatment

```
plt.figure(figsize=(10,40))
plt.subplot(9,2,18)
sb.countplot(data['obs_consequence'], hue = data['treatment'])
plt.title('Consequence after Disclosure')
```



We observe that majority of people have not faced any negative consequences after discussing their mental health conditions in their workplace

Activity 3: Descriptive analysis

Descriptive analysis is to study the basic statistical features of data. We can achieve it by using the .describe() function. With this describe function we can understand the unique, top and frequent values of categorical features. Also, we can find mean, std, min, max and percentile values of numerical features.

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

	Age	Gender	self_employed	family_history	treatment	work_interfere	no_employees	remote_work	tech_company
count	1247.000000	1247	1247	1247	1247	1247	1247	1247	1247
unique	NaN	3	2	2	2	5	6	2	2
top	NaN	Male	No	No	Yes	Sometimes	6-25	No	Yes
freq	NaN	983	1107	759	630	463	288	879	1023
mean	31.971131	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
std	7.052598	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
min	18.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
25%	27.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
50%	31.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
75%	36.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
max	60.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Milestone 4: Model Building

Activity 1: Handling Categorical Values

As we can see our dataset has categorical data. Before training our model, we must convert the categorical data into a numeric form.

There are multiple encoding techniques to convert the categorical columns into numerical columns. For this project we will be using ordinal encoding for our features and label encoding for our target.

So, for that we first need to divide our data into features and target.

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

Here X contains our features and y contains our target.

The next step is to apply respective encoding techniques on features and target. To apply ordinal encoding on our features, we will be using column transformer.

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

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

```
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)
```

We need to save the column transformer instance so that we can use it during our model deployment.

```
import joblib
joblib.dump(ct,'feature_values')
```

After executing the above lines, ct will be saved in a file known as feature_values.

Activity 2: Splitting data into train and test

For splitting the data into train and test sets, we are using the 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
```

```
((872, 22), (375, 22), (872,), (375,))
```

Activity 3: Comparing accuracy of various models

We will be considering multiple models to train our data and choose the one that performs the best. So, we need to import the necessary libraries and create a dictionary of our models.

```python
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
```

```python
model_dict = {}

model_dict['Logistic regression']= LogisticRegression(solver='liblinear',random_state=49)
model_dict['KNN Classifier'] = KNeighborsClassifier()
model_dict['Decision Tree Classifier'] = DecisionTreeClassifier(random_state=49)
model_dict['Random Forest Classifier'] = RandomForestClassifier(random_state=49)
model_dict['AdaBoost Classifier'] = AdaBoostClassifier(random_state=49)
model_dict['Gradient Boosting Classifier'] = GradientBoostingClassifier(random_state=49)
model_dict['XGB Classifier'] = XGBClassifier(random_state=49)
```

Next, we will define a function known as model_test() that accepts 6 parameters - X_train, X_test, y_train, y_test, model, model_name. We will obtain y_pred by using .predict() function and compute the accuracy score for every model by iterating through the dictionary.

```python
def model_test(X_train, X_test, y_train, y_test,model,model_name):
    model.fit(X_train,y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test,y_pred)
    print('======================================{}======================================'.format(model_name))
    print('Score is : {}'.format(accuracy))

    print()
```

```python
for model_name,model in model_dict.items():
    model_test(X_train, X_test, y_train, y_test, model, model_name)
```

```
======================================Logistic regression======================================
Score is : 0.848

======================================KNN Classifier======================================
Score is : 0.7813333333333333

======================================Decision Tree Classifier======================================
Score is : 0.7946666666666666

======================================Random Forest Classifier======================================
Score is : 0.8533333333333334

======================================AdaBoost Classifier======================================
Score is : 0.864

======================================Gradient Boosting Classifier======================================
Score is : 0.84

======================================XGB Classifier======================================
Score is : 0.8106666666666666
```

From the above results, it is clear that AdaBoost Classifier provides the best accuracy. So let us create a different variable to fit and make predictions using the model.

```python
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= 0.864
```

Activity 4: Hyperparameter tuning of selected model

To further improve the model performance, we are going to carry out a process known as hyperparameter tuning. Every model will have multiple hyperparameters. Please find the hyperparameters for AdaBoost Classifier from the below link:

[AdaBoostClassifier-docs]().

Of these, we will be tuning n_estimators and learning_rate.

For hyperparameter tuning, we can either use GridSearchCV or RandomizedSearchCV. RandomizedSearchCV is more fast, efficient and preferred so we will be using it for our project.

```
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]}
```

For n_estimators, we are taking 15 equally spaced values from 1 to 50 and for learning rate, we are trying various values close to 1.

There are various parameters to be passed in RandomizedSearchCV. To know what each parameter signifies, please refer to the below link:

[RandomizedSearchCV-docs]()

Next, let us fit our data and check what are the best hyperparameters using the .best_params_ attribute.

```
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)
```

```
abc_random.best_params_
```

```
{'n_estimators': 11, 'learning_rate': 1.02}
```

So, our model will perform the best if n_estimators are equal to 11 and learning_rate is equal to 1.02. Let us add these values to train our model, make predictions and check accuracy.

```python
abc_tuned = AdaBoostClassifier(random_state=49,n_estimators=11, learning_rate=1.02)
abc_tuned.fit(X_train,y_train)
pred_abc_tuned = abc_tuned.predict(X_test)
print('Accuracy of AdaBoost(tuned)=',accuracy_score(y_test,pred_abc_tuned))
```

```
Accuracy of AdaBoost(tuned)= 0.8693333333333333
```

We observe that the accuracy has increased approximately by 0.5%. Though this is not a very great improvement, it is at least better than our previous model.

Activity 5: Evaluating performance of models

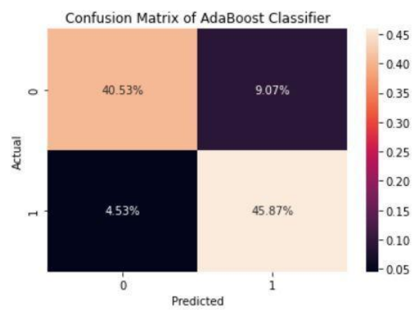
We will compare the confusion matrix, ROC curve and classification report for both models.

In order to obtain these, we will be using the confusion_matrix(), roc_curve() and classification_report() functions from sklearn.metrics.

Confusion matrix:

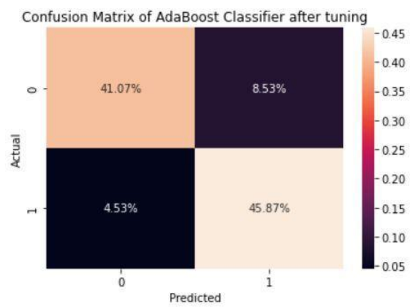
```python
cf_matrix = confusion_matrix(y_test, pred_abc)
sb.heatmap(cf_matrix/np.sum(cf_matrix), annot=True, fmt='.2%')
plt.title('Confusion Matrix of AdaBoost Classifier')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

```
Text(33.0, 0.5, 'Actual')
```



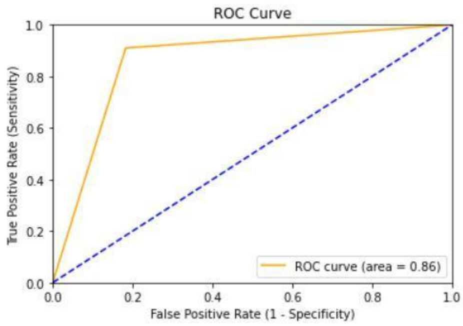

```python
cf_matrix = confusion_matrix(y_test, pred_abc_tuned)
sb.heatmap(cf_matrix/np.sum(cf_matrix), annot=True, fmt='.2%')
plt.title('Confusion Matrix of AdaBoost Classifier after tuning')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

```
Text(33.0, 0.5, 'Actual')
```

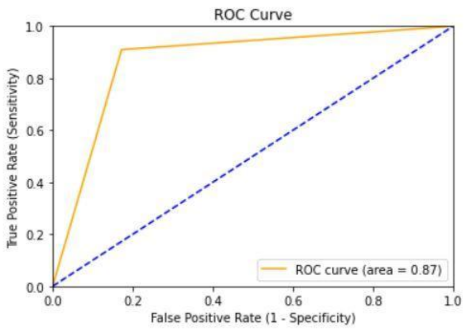


ROC Curve:

```python
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)
```




```python
fpr_abc_tuned, tpr_abc_tuned, thresholds_abc_tuned = roc_curve(y_test, pred_abc_tuned)
roc_auc_abc_tuned = metrics.auc(fpr_abc_tuned, tpr_abc_tuned)
plt.plot(fpr_abc_tuned, tpr_abc_tuned, color='orange', label='ROC curve (area = %0.2f)' % roc_auc_abc_tuned)
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_tuned)
```



Classification report:

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

```
              precision    recall  f1-score   support

           0       0.90      0.82      0.86       186
           1       0.83      0.91      0.87       189

    accuracy                           0.86       375
   macro avg       0.87      0.86      0.86       375
weighted avg       0.87      0.86      0.86       375
```

```python
print(classification_report(y_test,pred_abc_tuned))
```

```
              precision    recall  f1-score   support

           0       0.96      0.72      0.82       195
           1       0.76      0.97      0.85       180

    accuracy                           0.84       375
   macro avg       0.86      0.84      0.84       375
weighted avg       0.87      0.84      0.84       375
```

Activity 6: Saving the model

The final step is saving our model. We can do it by using pickle.dump().

```python
import pickle
pickle.dump(abc_tuned,open('model.pkl','wb'))
```

Milestone 5: Application Building

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 entered 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

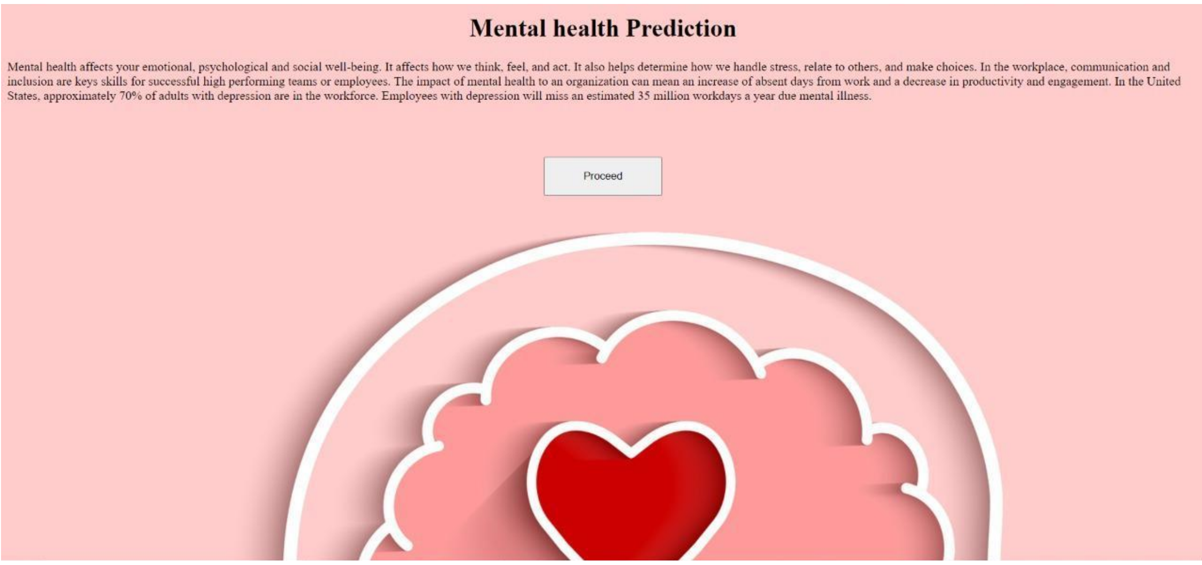
Activity1: Building Html Pages:

For this project create three HTML files namely

- home.html
- index.html
- output.html

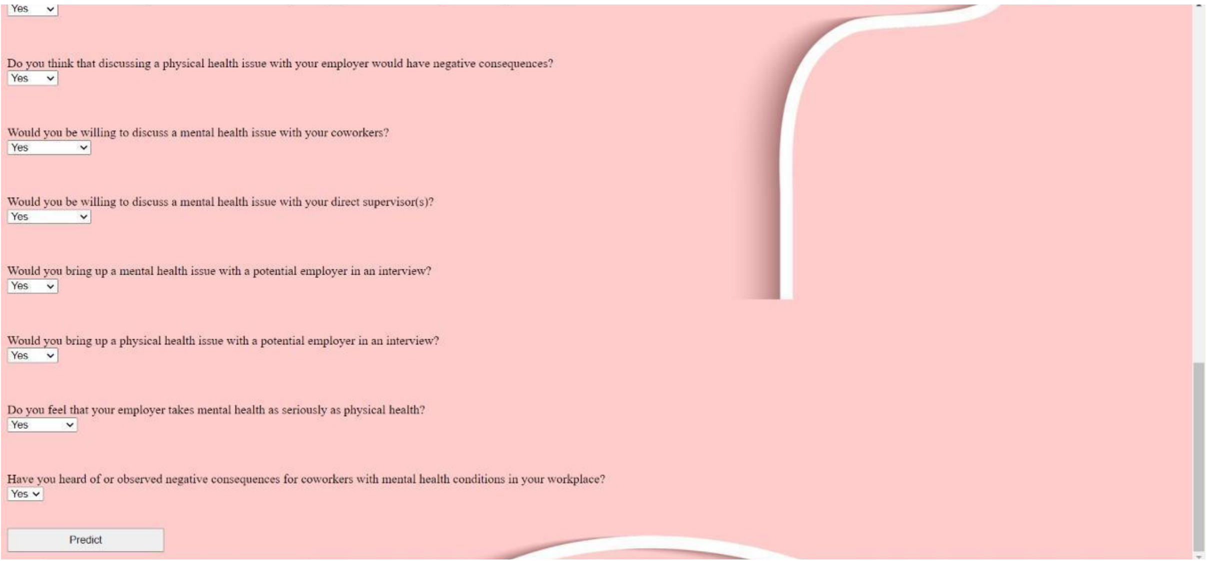
and save them in the templates folder.

Let's see how our home.html page looks like:



Now when you click on proceed button, you will get redirected to index.html

Let us look how our index.html page looks like:



Now when you click on predict button from left bottom corner you will get redirected to output.html

Let us look how our output.html page looks like:

Activity 2: Build Python code:

Import the required libraries and load model and ct

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

app = Flask(__name__)

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

Render home.html and index.html pages:

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

@app.route('/pred')
def predict():
    return render_template("index.html")
```

The values entered in can be retrieved using the POST Method.

Retrieves the value from UI:

```python
@app.route('/out', methods =["POST"])
def output():
    age = request.form["age"]
    gender = request.form["gender"]
    self_employed = request.form["self_employed"]
    family_history = request.form["family_history"]
    work_interfere = request.form["work_interfere"]
    no_employees = request.form["no_employees"]
    remote_work = request.form["remote_work"]
    tech_company = request.form["tech_company"]
    benefits = request.form["benefits"]
    care_options = request.form["care_options"]
    wellness_program = request.form["wellness_program"]
    seek_help = request.form["seek_help"]
    anonymity = request.form["anonymity"]
    leave = request.form["leave"]
    mental_health_consequence = request.form["mental_health_consequence"]
    phys_health_consequence = request.form["phys_health_consequence"]
    coworkers = request.form["coworkers"]
    supervisor = request.form["supervisor"]
    mental_health_interview = request.form["mental_health_interview"]
    phys_health_interview = request.form["phys_health_interview"]
    mental_vs_physical = request.form["mental_vs_physical"]
    obs_consequence = request.form["obs_consequence"]

    data = [[age,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]]

    feature_cols = ['Age', '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']

    pred = model.predict(ct.transform(pd.DataFrame(data,columns=feature_cols)))
    pred = pred[0]
    if pred:
        return render_template("output.html",y="This person requires mental health treatment ")
    else:
        return render_template("output.html",y="This person doesn't require mental health treatment ")
```

Here we are routing our app to output() 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 output.html page earlier.

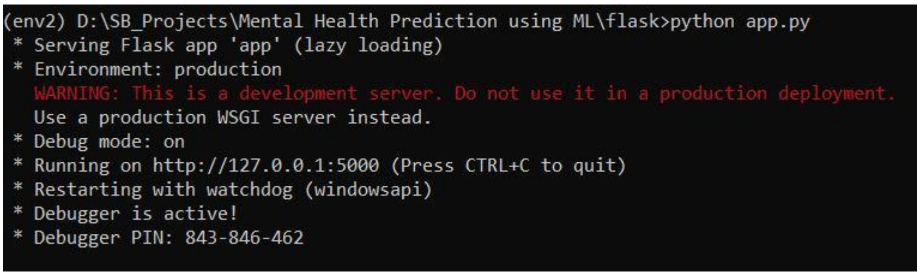
Main Function:

```python
if __name__ == '__main__':
    app.run(debug = True)
```

Activity 3: Run the 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 proceed button, enter the inputs, click on the predict button, and see the result/prediction on the web.



```
(env2) D:\SB_Projects\Mental Health Prediction using ML\flask>python app.py
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 843-846-462
```