Introduction

1.1 Project Overview: Predictive Mental Health for Working Professionals

The Predictive Mental Health for Working Professionals project aims to develop a cutting-edge system leveraging machine learning to predict and address mental health issues among working professionals. The primary objective is to provide proactive support, personalized intervention plans, and early identification of potential mental health challenges within the workplace. The project encompasses the creation of a scalable and secure platform, incorporating diverse data sources related to work habits, communication patterns, and self-reported assessments. Key features include a dynamic risk assessment module, personalized intervention plans, sentiment analysis integration, and user-friendly dashboards for both employees and management. The project is expected to enhance overall employee well-being, contribute to a healthier work environment, and facilitate timely intervention for improved mental health outcomes.

1.2 Purpose

The purpose of the Predictive Mental Health for Working Professionals project is to revolutionize the approach to mental health support in the workplace. Recognizing the increasing importance of mental well-being, this project seeks to proactively address the challenges faced by working professionals by harnessing the power of machine learning. Through the development of a sophisticated and scalable platform, our goal is to predict potential mental health issues, enabling early identification and personalized intervention plans. By leveraging diverse data sources, including work-related activities, individual habits, and communication patterns, we aim to create a comprehensive solution that not only fosters a healthier work environment but also empowers both employees and management with actionable insights. The overarching purpose is to contribute to a positive shift in workplace culture, emphasizing the well-being of employees and fostering a more supportive and understanding professional environment.

Literature Survey

2.1 Existing Problem

In contemporary workplaces, the mental health of working professionals is increasingly recognized as a critical aspect of overall well-being. However, existing approaches often fall short in providing timely and personalized support. Many professionals face challenges in articulating their mental health needs, and organizations struggle to identify early signs of distress. Traditional methods of mental health monitoring are often reactive, leading to delayed interventions and a lack of holistic understanding. Moreover, the stigma associated with mental health discussions in the workplace further compounds these challenges, hindering open communication and preventing the implementation of proactive measures. The existing gap in mental health support systems necessitates a transformative solution that leverages advanced technologies to predict, address, and destigmatize mental health issues in the professional sphere.

2.2 References

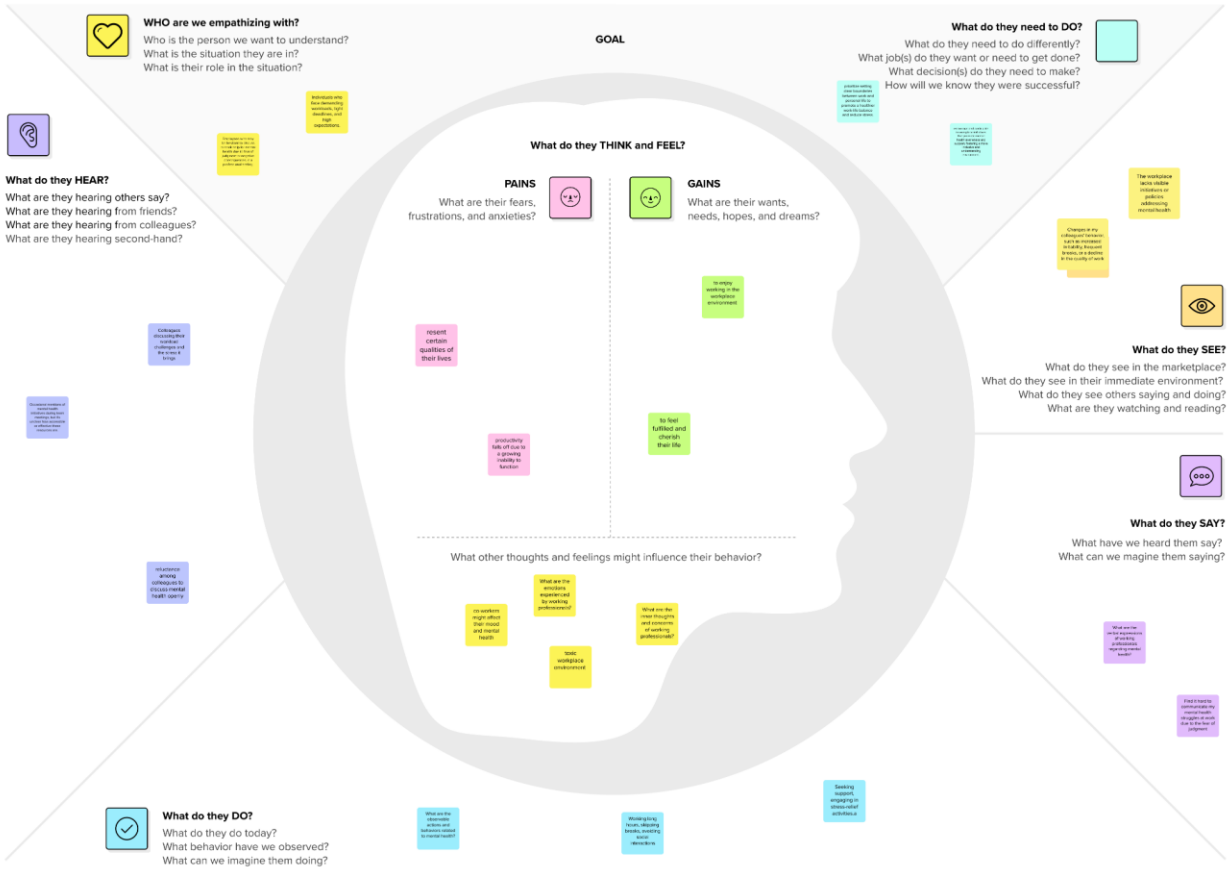
https://journals.sagepub.com/doi/full/10.1177/0312896220922292

2.3 Problem Statement Definition

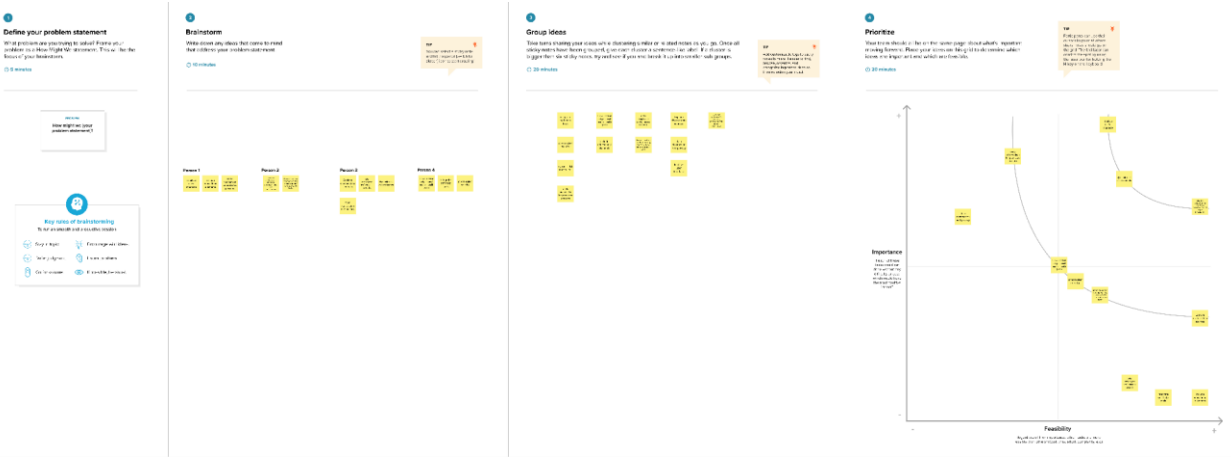
In today's fast-paced work environments, the mental health of working professionals is a growing concern.We aim to control the damage.

Ideation & Proposed Solutions

3.1 Empathy Canvas



3.2 Brainstorming



REQUIREMENT ANALYSIS

4.1 Functional requirement

1. **User Authentication and Authorization:**
    - **The system should provide secure user authentication to ensure that only authorized personnel can access the platform. Different user roles (e.g., employees, managers) should have appropriate levels of access.**
2. **Data Collection and Integration:**
    - **The system should be able to collect and integrate data from various sources, including work-related activities, communication patterns, and self-reported assessments.**
3. **Machine Learning Models:**
    - **The system should implement machine learning models for predictive analysis of mental health. These models should be capable of analyzing input data and generating predictions or risk assessments.**
4. **Personalized Intervention Plans:**
    - **Based on the predictions from the machine learning models, the system should generate personalized intervention plans for individuals at risk. These plans may include recommendations for stress management, workload adjustments, or access to mental health resources.**
5. **Real-time Monitoring and Alerts:**
    - **The system should offer real-time monitoring of user data and mental health indicators. It should generate alerts or notifications for individuals, managers, or HR personnel when potential issues are detected.**
6. **User Dashboards:**
    - **Provide user-friendly dashboards for both employees and management to visualize individual and aggregate mental health data. Dashboards should include relevant metrics, trends, and actionable insights.**
7. **Communication and Feedback Mechanism:**
    - **Implement a communication channel within the system for users to provide feedback, report concerns, or seek assistance. The system should facilitate communication between employees and management in a secure and confidential manner.**
8. **Scalability:**
    - **The system should be scalable to handle a growing user base and increasing volumes of data. It should efficiently accommodate additional features, users, and data sources without compromising performance.**
9. **Integration with Existing Systems:**
    - **Ensure seamless integration with existing HR and communication systems within the organization to facilitate data flow and avoid duplication of efforts.**
10. **Security Measures:**

- **Implement robust security measures to protect sensitive mental health data. This includes encryption of data in transit and at rest, role-based access controls, and regular security audits.**

11. **Anonymization of Data:**
    - **Ensure that any personally identifiable information (PII) is anonymized or pseudonymized to protect user privacy while still allowing for effective analysis.**
12. **Training and Support:**
    - **Provide training resources and support for users and administrators to effectively use and manage the system.**

4.2 Non-Functional requirements

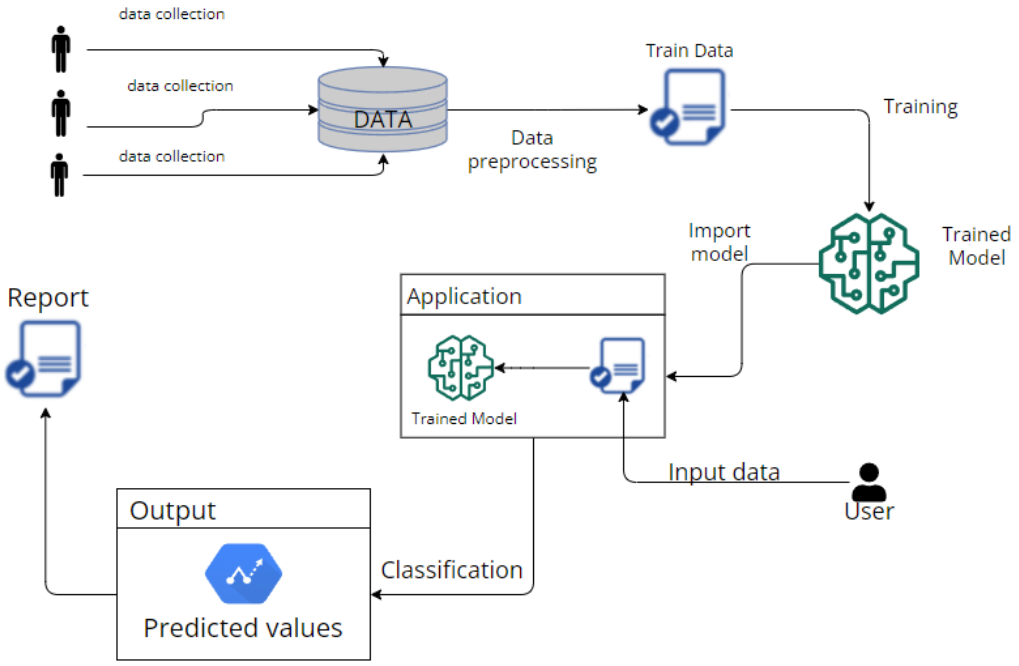
1. **Performance:**
    - **The system should provide responses to user queries and predictions within a maximum response time of 2 seconds under normal operating conditions.**
2. **Scalability:**
    - **The system should be designed to handle a 20% increase in the user base and data volume within the next year without a significant degradation in performance.**
3. **Reliability:**
    - **The system should have an uptime of at least 99.9%, ensuring continuous availability for users and preventing significant disruptions.**
4. **Availability:**
    - **The system should be available for use 24/7, with planned maintenance windows communicated in advance to users.**
5. **Security:**
    - **Data transmission should be encrypted using TLS to ensure the confidentiality and integrity of information exchanged between users and the system.**
6. **Privacy:**
    - **The system should comply with relevant privacy regulations and standards, ensuring the confidentiality of user data and allowing users to control their data-sharing preferences.**
7. **Usability:**
    - **The user interface should be intuitive and user-friendly, requiring no more than 30 minutes of training for new users to become proficient in using the system.**
8. **Compatibility:**
    - **The system should be compatible with commonly used web browsers (e.g., Chrome, Firefox, Safari) and mobile devices (iOS and Android).**
9. **Maintainability:**
    - **The system should be designed with modular and well-documented code to facilitate ease of maintenance and future enhancements.**
10. **Auditability:**
    - **The system should maintain an audit trail of user interactions, predictions, and interventions for accountability and compliance purposes.**
11. **Response Time:**

- **The system should generate predictive analyses and alerts within a maximum time of 5 seconds to ensure timely intervention.**

12. **Capacity:**
    - **The system should be capable of handling a concurrent user load of at least 500 users without experiencing a degradation in performance.**
13. **Cultural Sensitivity:**
    - **The system's communication and intervention plans should be culturally sensitive and avoid biases based on gender, ethnicity, or cultural background.**

PROJECT DESIGN

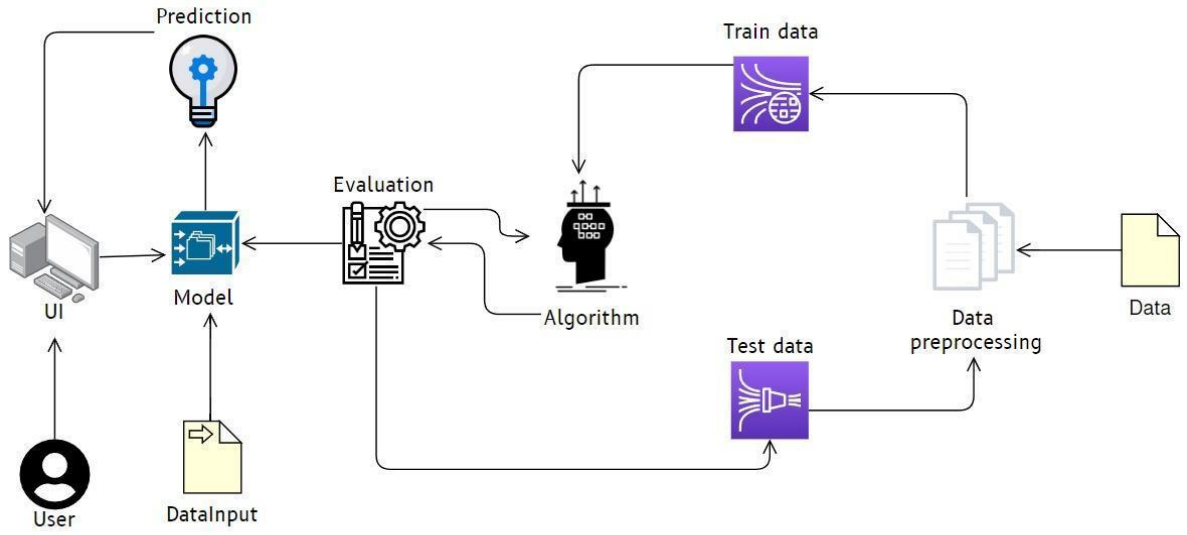
5.1 Data Flow Diagrams & User Stories



User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
HR Manager	Data Upload	USN - 1	As an HR Manager, I want to upload employee survey data easily so that the system can analyse and predict mental health statuses based on the collected information.	The system should provide a user-friendly interface for HR Managers to upload CSV files containing employee survey data.	High	Sprint - 1
Data Scientist	Data processing	USN - 2	As a Data Scientist, I want to access and preprocess raw data efficiently so that the machine learning model can be trained on clean and relevant data.	The Data Preprocessing Module should provide a command-line interface for data scientists to access raw data.	High	Sprint - 1
System Administrator	Monitoring and Alerts	USN - 3	As a System Administrator, I want to monitor the system's performance and receive alerts for potential issues so that I can ensure the reliability of the mental health prediction system.	The system should log key performance metrics, including data processing time and model training time. A monitoring dashboard should display real-time system performance.	High	Sprint - 2
Mental health professional	Evaluation Metrics	USN - 4	As a Mental Health Professional, I want to review the evaluation metrics and model performance to ensure the accuracy and reliability of mental health predictions.	The Evaluation Module should provide a dashboard displaying key model metrics, including accuracy, precision, recall, and F1 score.	Medium	Sprint - 2
Employee	Well-being Feedback	USN - 5	As an Employee, I want to receive timely and confidential feedback on my mental health status so that I can take proactive steps to maintain my well-being.	The User Interface Module should provide a secure login for employees to access their	High	Sprint - 3

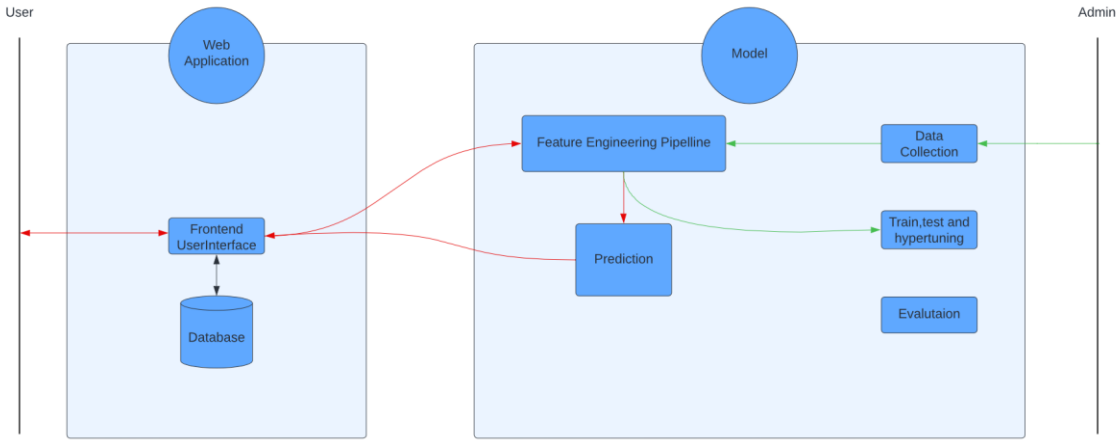
				personalized well-being feedback. Upon login, employees should see a clear visualization of their predicted mental health status along with an explanation of contributing factors.		

5.2 Solution Architecture



PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture



Components & Technologies:

SNo	Component	Description	Technology
1.	User Interface	A web user interface	HTML CSS JavaScript
2.	Application Logic -1	Data Collection Module	Python , Flask
3.	Application Logic -2	Feature Engineering	Python , Pandas,numpy
4.	Application Logic -3	Machine Learning Model	Python,scikit-learn
5.	Application Logic -4	Personalized Intervention Module	Python , Flask
6.	Machine Learning Model	To classify the user input data	LogisticRegression DecisionTreeClassifier RandomForestClassifier

Application Characteristics:

Sno	Characteristic	Description	Technology
1.	Open-Source Frameworks	Flask , scikit-Learn,Pickle	Flask is a web framework for Python,Scikit-learn is a machine learning library for Python,Pickles is a living documentation generator
2.	Security Implementations	Encryption,IAM	Flask uses HTTPS to encrypt data transferred and Admins have an access check before they can modify the model.
3.	Scalable Architecture	Modularity,Asynchronous Processing	The application has been divided into two major parts to ensure modularity and Python multithreading is used to ensure asynchronous processing.
4.	Availability	Continuous Monitoring	Continuous monitoring of the application's health and performance
5.	Performance	Caching Strategies,CDN Integration	Localstorage of browsers used to cache login information and often used images are fetched using CDN and cached

6.2 Sprint Planning & Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint - 1	Data Upload	USN - 1	As an HR Manager, I want to upload employee survey data easily so that the system can analyse and predict mental health statuses based on the collected information.	2	High	
Sprint - 1		USN - 2	As a Data Scientist, I want to access and preprocess raw data efficiently so that the machine learning model can be trained on clean and relevant data.	1	High	
Sprint - 2	Monitoring and Alerts	USN - 3	As a System Administrator, I want to monitor the system's performance and receive alerts for potential issues so that I can ensure the reliability of the mental health prediction system	1	High	

Sprint - 2		**USN - 4**	**As a Mental Health Professional, I want to review the evaluation metrics and model performance to ensure the accuracy and reliability of mental health predictions.**	**2**	**Medium**	
Sprint - 3		**USN - 5**	**As an Employee, I want to receive timely and confidential feedback on my mental health status so that I can take**	**1**	**High**	

CODING & SOLUTIONING

7.1 Dynamic Risk Assessment

Implement a dynamic risk assessment module that continuously evaluates mental health indicators and adjusts risk scores based on real-time data. This feature ensures that the system adapts to changing circumstances and provides up-to-date insights.

```
@app.route('/predict',methods=['POST','GET'])
def predict():
    int_features=[int(x) for x in request.form.values()]
    temp= scaler.transform(np.array(int_features[0]).reshape(1,-1))[0][0]
    int_features[0] = temp
    final=[np.array(int_features)]
    prediction=model.predict_proba(final)
    output='{0:.{1}f}'.format(prediction[0][1], 2)
```

```
    if output>str(0.5):
        return render_template('index.html',pred='You need a treatment.\nProbability of
mental illness is {}'.format(output))
    else:
        return render_template('index.html',pred='You do not need treatment.\n
Probability of mental illness is {}'.format(output))
```

7.2 useState functionality in pre-rendered HTML

On the click of the button the predicted value is automatically re-rendered in an already pre-rendered static HTML page.

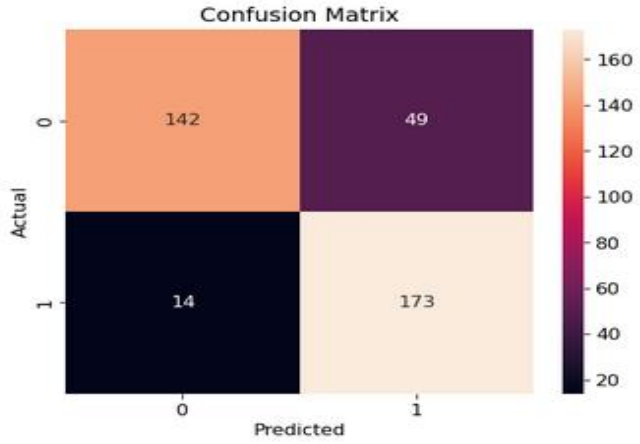
```
<button type="submit" class="btn">Predict Probability</button>

<h1>{{pred}}</h1>
```

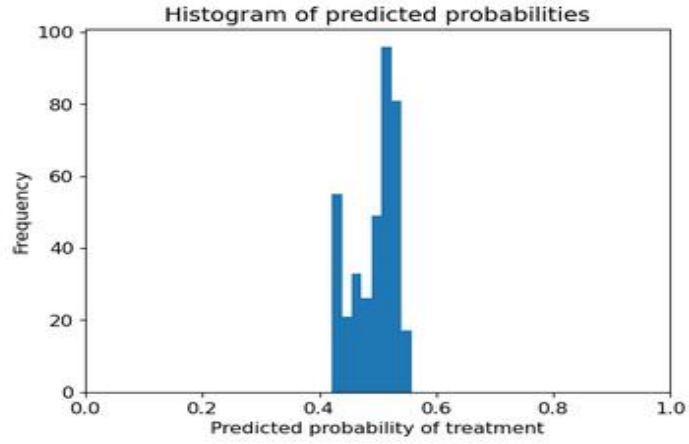
PERFORMANCE TESTING

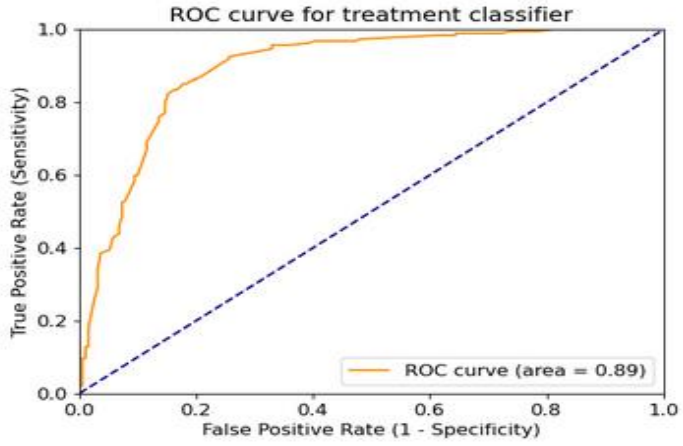
8.1 Performance Metrics

```
Accuracy: 0.8333333333333334
```




```
Classification Accuracy: 0.8333333333333334
Classification Error: 0.16666666666666663
False Positive Rate: 0.25654450261780104
Precision: 0.7792792792792793
AUC Score: 0.8342945936108856
Cross-validated AUC: 0.8948277832638432
```

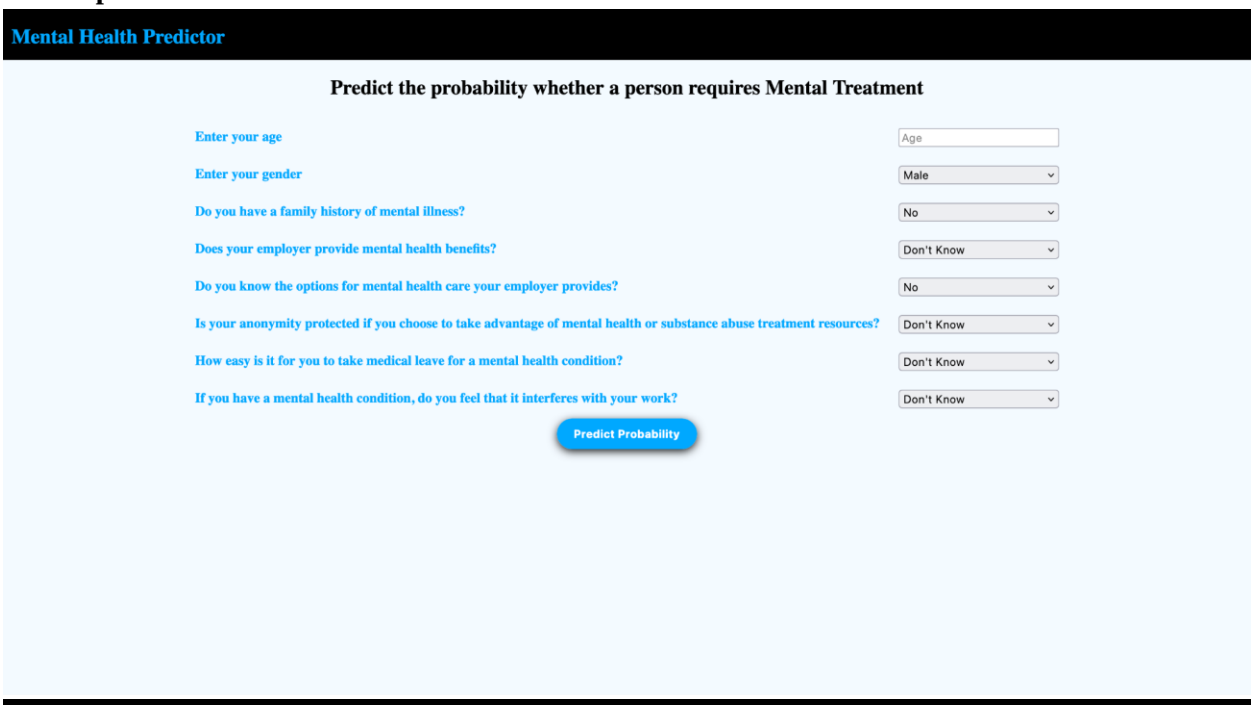


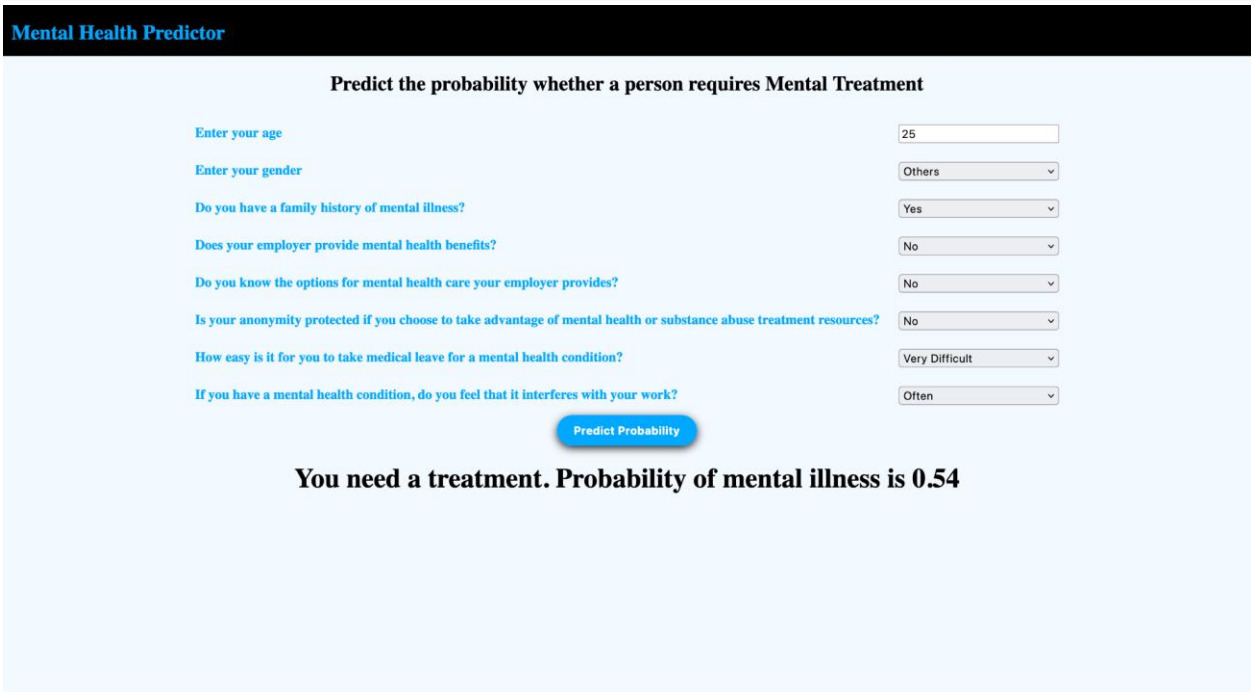



```
[[142  49]
 [ 14 173]]
```

RESULTS

9.1 Output Screenshots





ADVANTAGES & DISADVANTAGES

Advantages:

- **Early Detection:**
  - Machine learning models can analyze patterns in large datasets to detect early signs of mental health issues, enabling timely intervention and support.
- **Objective Assessment:**
  - ML models can provide an objective assessment based on data, reducing the impact of subjective biases that might be present in human evaluations.
- **Personalized Insights:**
  - ML algorithms can tailor predictions based on individual characteristics, providing personalized insights and recommendations for support.
- **Efficiency and Scalability:**
  - Once trained, machine learning models can quickly analyze large amounts of data, making them efficient and scalable for screening a large number of individuals.
- **Continuous Monitoring:**
  - ML models can be used for continuous monitoring, offering a dynamic and ongoing assessment of an individual's mental health status over time.
- **Resource Optimization:**
  - Predictive models can help allocate mental health resources more effectively by focusing on individuals who are at higher risk, thus optimizing the use of healthcare resources.

Disadvantages:

- **Data Bias:**
  - If the training data is biased or not representative, the model may also be biased. This could lead to inaccurate predictions and potential disparities in the assessment of mental health.
- **Privacy Concerns:**
  - Dealing with sensitive mental health data raises privacy concerns. Ensuring compliance with regulations and protecting individuals' privacy becomes crucial.
- **Complexity of Mental Health:**
  - Mental health is a complex and multifaceted issue. Machine learning models may oversimplify the problem, potentially missing important nuances and contributing factors.
- **Interpretability Challenges:**
  - Some advanced machine learning models, particularly deep learning models, can be difficult to interpret. Understanding how a model arrives at a particular prediction may be challenging, raising concerns about trust and accountability.
- **Dynamic Nature of Mental Health:**
  - Mental health conditions can change over time, and the factors influencing them may evolve. Models may struggle to adapt to these changes and may require regular updates.
- **False Positives and Negatives:**

- Models may produce false positives (predicting a mental health issue that doesn't exist) or false negatives (missing an actual issue). Striking a balance between sensitivity and specificity is crucial but challenging.

- **Ethical Considerations:**
  - Ethical issues, including the potential for stigmatization and discrimination based on mental health predictions, need to be carefully addressed.
- **User Acceptance:**
  - Users, especially working professionals, may be skeptical or resistant to the use of machine learning in mental health assessment. Ensuring user acceptance and understanding is essential for successful implementation.

CONCLUSION

Leveraging machine learning for predicting mental health illnesses among working professionals presents both promising opportunities and significant challenges. The advantages of early detection, objective assessment, personalized insights, efficiency, scalability, and resource optimization demonstrate the potential positive impact on individual well-being and healthcare resource allocation. However, these benefits must be carefully weighed against the potential disadvantages and ethical considerations associated with data bias, privacy concerns, the complexity of mental health, interpretability challenges, the dynamic nature of mental health, the risk of false positives and negatives, and user acceptance issues.

FUTURE SCOPE

- **Improved Accuracy and Personalization:**
  - Future advancements in machine learning algorithms, especially those incorporating deep learning techniques, may enhance the accuracy of predictions and provide more personalized insights by capturing subtle patterns in diverse datasets.
- **Integration with Wearable Devices:**
  - As wearable technology becomes more prevalent, integrating data from devices such as smartwatches and fitness trackers could offer real-time information on physiological and behavioral indicators, contributing to more comprehensive and dynamic mental health assessments.
- **Incorporating Multimodal Data:**
  - Combining data from various sources, such as social media activity, speech patterns, and physiological signals, could lead to a more holistic understanding of an individual's mental health. This multimodal approach may provide a more nuanced and accurate assessment.
- **Explainable AI (XAI) in Mental Health Models:**
  - The development of more explainable machine learning models can address concerns related to interpretability. This will be essential for gaining trust from both professionals and individuals using these predictive tools.

APPENDIX

Github Repo

Project Demo Link

Source Code:

app.py

```
from flask import Flask,request, url_for, redirect, render_template
import pickle,joblib
import numpy as np

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

model=pickle.load(open('./model.pkl','rb'))
scaler = joblib.load('./scaler')

@app.route('/')
def hello_world():
    return render_template("index.html")


@app.route('/predict',methods=['POST','GET'])
def predict():
    int_features=[int(x) for x in request.form.values()]
    temp= scaler.transform(np.array(int_features[0]).reshape(1,-1))[0][0]
    int_features[0] = temp
    final=[np.array(int_features)]
    prediction=model.predict_proba(final)
    output='{0:.{1}f}'.format(prediction[0][1], 2)

    if output>str(0.5):
        return render_template('index.html',pred='You need a treatment.\nProbability of
mental illness is {}'.format(output))
    else:
        return render_template('index.html',pred='You do not need treatment.\n
Probability of mental illness is {}'.format(output))


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

index.html

```
<!DOCTYPE html>
<html lang="en">
 <head>
   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
   <meta name="viewport" content="width=device-width, initial-scale=1, maximum-
scale=1.0"/>
   <title>Mental Health Predictor</title>
   <style>
     .container
     {
       text-align: center;
       background-color: aliceblue;
       min-height: 100vh;
       width: 100vw;
       position: absolute;

     }

     nav{
       display: flex;
       background-color: black;
       color: rgb(0,149,255);
       padding-left: 10px;
       height: 60px;

     }
     body{
       padding: 0px;
       margin: 0px;
     }

     table
     {margin: auto auto;}

     td,tr
     {
       padding: 10px;
     }

     .btn
```

```
    {
        padding: 10px 20px 10px 20px;
        background-color: rgb(0,149,255);
        border: none;
        border-radius: 50px;
        color:white;
        font-weight: 700;
        cursor: pointer;
        box-shadow: 0px 3px 10px black;
        text-decoration: none;
    }

    .btn:active
    {
        box-shadow: 0px 0px 3px black;
        transform: scaleY(7px);
    }

    label{
      color: rgb(0,149,255);
      font-weight: 900;
    }

    #label
    {
      text-align: start;
    }
    select{
      width: 100%;
    }


  </style>
</head>

<body>
  <div class="section no-pad-bot" id="index-banner">
    <nav>
      <h1>
        Mental Health Predictor
      </h1>
    </nav>
```

```
    <div class="container">
          <h2>
            Predict the probability whether a person requires Mental Treatment
          </h2>
        <form action='/predict' method="post" >
          <table>
              <tr>
                  <td id="label">
                    <label for="Age">Enter your age</label>
                  </td>
                  <td>
                    <input placeholder="Age" name="Age" type="text">
                  </td>
              </tr>
              <tr>
                  <td id="label">
                    <label>Enter your gender</label>
                  </td>
                  <td>
                    <select name="Gender">
                      <option value=0>Male</option>
                      <option value=1>Female</option>
                      <option value=2>Others</option>
                    </select>
                  </td>
              </tr>
              <tr>
                  <td id="label">
                    <label for="family_history">Do you have a family history of
mental illness?</label>
                  </td>
                  <td>
                    <select name="family_history">
                      <option value=0>No</option>
                      <option value=1>Yes</option>
                    </select>

                  </td>
              </tr>
              <tr>
                  <td id="label">
```

```
                    <label for="benefits"> Does your employer provide mental health
benefits?</label>
                </td>
                <td>
                  <select name="benefits">
                    <option value=0>Don't Know</option>
                    <option value=1>No</option>
                    <option value=2>Yes</option>
                  </select>

                </td>
            </tr>
            <tr>
                <td id="label">
                  <label for="care_options">Do you know the options for mental
health care your employer provides?</label>
                </td>
                <td>
                  <select name="care_options">
                    <option value=0>No</option>
                    <option value=1>Not Sure</option>
                    <option value=2>Yes</option>
                  </select>
                </td>
            </tr>
            <tr>
                <td id="label">
                  <label for="anonymity">Is your anonymity protected if you choose
to take advantage of mental health or substance abuse treatment resources?</label>
                </td>
                <td>
                  <select name="anonymity">
                    <option value=0>Don't Know</option>
                    <option value=1>No</option>
                    <option value=2>Yes</option>
                  </select>
                </td>
            </tr>
            <tr>
                <td id="label">
                  <label for="leave">How easy is it for you to take medical leave
for a mental health condition?</label>
```

```
                    </td>
                    <td>
                      <select name="leave">
                        <option value=0>Don't Know</option>
                        <option value=1>Somewhat Difficult</option>
                        <option value=2>Somewhat Easy</option>
                        <option value=3>Very Difficult</option>
                        <option value=4>Very Easy</option>
                      </select>
                    </td>
                </tr>
                <tr>
                    <td id="label">
                      <label for="work_interefere">If you have a mental health
condition, do you feel that it interferes with your work?</label>
                    </td>
                    <td>
                      <select name="work_interefere">
                        <option value=0>Don't Know</option>
                        <option value=1>Never</option>
                        <option value=2>Often</option>
                        <option value=3>Rarely</option>
                        <option value=4>Sometimes</option>
                      </select>
                    </td>
                </tr>
            </table>
            <button type="submit" class="btn">Predict Probability</button>
          </form>
          <h1>{{pred}}</h1>
        </div>
      </div>
</body>
</html>
```

notebook.ipynb

```
#!/usr/bin/env python
# coding: utf-8

# <a href="https://colab.research.google.com/github/cdodiya/Mental-Health-Prediction-using-Machine-Learning-Algorithms/blob/main/MentalHealthPredictionUsingMachineLearningAlgorithms.ipynb" target="_parent"><img src="https://colab.research.google.com/assets/colab-badge.svg" alt="Open In Colab"/></a>

# #Library and Data Loading

# In[133]:



import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from scipy import stats
from scipy.stats import randint

# prep
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.datasets import make_classification
from sklearn.preprocessing import binarize, LabelEncoder, MinMaxScaler,OrdinalEncoder
from sklearn.compose import ColumnTransformer

# models
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier

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

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

#Exports
import pickle,joblib


# In[80]:


train_df = pd.read_csv('survey.csv')
print(train_df.shape)
print(train_df.describe())
print(train_df.info())


# # Data Cleaning

# In[81]:


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


# In[82]:


#dealing with missing data
train_df.drop(['comments'], axis= 1, inplace=True)
train_df.drop(['state'], axis= 1, inplace=True)
train_df.drop(['Timestamp'], axis= 1, inplace=True)

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

```
# ### Cleaning NaN

# In[83]:


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

# Create lists by data tpe
intFeatures = ['Age']
stringFeatures = ['Gender', 'Country', 'self_employed', 'family_history', 'treatment',
'work_interfere',
                'no_employees', 'remote_work', 'tech_company', 'anonymity', 'leave',
'mental_health_consequence',
                'phys_health_consequence', 'coworkers', 'supervisor',
'mental_health_interview', 'phys_health_interview',
                'mental_vs_physical', 'obs_consequence', 'benefits', 'care_options',
'wellness_program',
                'seek_help']
floatFeatures = []

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


# In[84]:


#Clean 'Gender'
```

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


# In[85]:


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

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

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

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

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

stk_list = ['A little about you', 'p']
train_df = train_df[~train_df['Gender'].isin(stk_list)]

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


# In[86]:


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

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

```
s[s<18] = train_df['Age'].median()
train_df['Age'] = s
s = pd.Series(train_df['Age'])
s[s>120] = train_df['Age'].median()
train_df['Age'] = s

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


# In[87]:


#There are only 0.014% of self employed so let's change NaN to NOT self_employed
#Replace "NaN" string from defaultString
train_df['self_employed'] = train_df['self_employed'].replace([defaultString], 'No')
print(train_df['self_employed'].unique())


# In[88]:


#There are only 0.20% of self work_interfere so let's change NaN to "Don't know
#Replace "NaN" string from defaultString

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


# In[89]:


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


# ### Testing there aren't any missing data

# In[90]:
```

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


# ### Some charts to see data relationship

# In[91]:


#How comfortable people are to talk about physical health issues
plt.figure(figsize=(20,50))
plt.subplot(10,2,20)
sns.countplot(x=train_df['phys_health_interview'],hue= train_df['treatment'])
plt.title('Phyiscal Health Dicussion')


# In[92]:


#How comfortable people are to talk about mental health issues
plt.figure(figsize=(20,50))
plt.subplot(10,2,20)
sns.countplot(x=train_df['mental_health_interview'],hue= train_df['treatment'])
plt.title('Mental Health Dicussion')


# People are more open to talk about physical health issues than mental health issues.

# In[93]:


#Consequence for discussing about physical health issues
plt.figure(figsize=(20,50))
plt.subplot(10,2,20)
sns.countplot(x=train_df['phys_health_consequence'],hue= train_df['treatment'])
```

```
plt.title('Phyiscal Health Consequence')


# In[94]:


#Consequence for discussing about mental health issues
plt.figure(figsize=(20,50))
plt.subplot(10,2,20)
sns.countplot(x=train_df['mental_health_consequence'],hue= train_df['treatment'])
plt.title('Mental Health Consequence')


# Lot of people have faced consequences after discussing about their mental health
issues than they have faced for discussing about their physical health issues.

# In[95]:


#Distribution and density by Age
plt.figure(figsize=(12,8))
sns.histplot(train_df["Age"], bins=24)
plt.title("Distribution and density by Age")
plt.xlabel("Age")


# # Spilitting Dataset

# In[96]:


# define X and y
feature_cols = ['Age', 'Gender', 'family_history', 'benefits', 'care_options',
'anonymity', 'leave', 'work_interfere']
X = train_df[feature_cols]
y = train_df.treatment


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

```
# # Data PipeLine

# In[97]:


ct = ColumnTransformer(
   [('oe',OrdinalEncoder(),["Gender","family_history",'benefits'
                            ,'care_options','anonymity','leave','work_interfere']),
   ("scaler",MinMaxScaler(),["Age"])
   ],
   remainder="passthrough"

)

X =ct.fit_transform(X)


# In[98]:


y = LabelEncoder().fit_transform(y)


# # Split Data

# In[99]:


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


# # Feature Importance

# In[100]:


# Build a forest and compute the feature importances
forest = ExtraTreesClassifier(n_estimators=250,
```

```
                              random_state=0)

forest.fit(X, y)
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

labels = []
for f in range(X.shape[1]):
    labels.append(feature_cols[f])

# Plot the feature importances of the forest
plt.figure(figsize=(12,8))
plt.title("Feature importances")
plt.bar(range(X.shape[1]), importances[indices],
       color="r", yerr=std[indices], align="center")
plt.xticks(range(X.shape[1]), labels, rotation='vertical')
plt.xlim([-1, X.shape[1]])
plt.show()


# # Tuning

# In[101]:


def evalClassModel(model, y_test, y_pred_class, plot=False):
    #Classification accuracy: percentage of correct predictions
    # calculate accuracy
    print('Accuracy:', metrics.accuracy_score(y_test, y_pred_class))


    #Confusion matrix
    # save confusion matrix and slice into four pieces
    confusion = metrics.confusion_matrix(y_test, y_pred_class)
    #[row, column]
    TP = confusion[1, 1]
    TN = confusion[0, 0]
    FP = confusion[0, 1]
    FN = confusion[1, 0]
```

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

    #Metrics computed from a confusion matrix
    #Classification Accuracy: Overall, how often is the classifier correct?
    accuracy = metrics.accuracy_score(y_test, y_pred_class)
    print('Classification Accuracy:', accuracy)

    #Classification Error: Overall, how often is the classifier incorrect?
    print('Classification Error:', 1 - metrics.accuracy_score(y_test, y_pred_class))

    #False Positive Rate: When the actual value is negative, how often is the
prediction incorrect?
    false_positive_rate = FP / float(TN + FP)
    print('False Positive Rate:', false_positive_rate)

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


    # IMPORTANT: first argument is true values, second argument is predicted
probabilities
    print('AUC Score:', metrics.roc_auc_score(y_test, y_pred_class))

    # calculate cross-validated AUC
    print('Cross-validated AUC:', cross_val_score(model, X, y, cv=10,
scoring='roc_auc').mean())

    # print the first 10 predicted probabilities for class 1
    model.predict_proba(X_test)[0:10, 1]

    # store the predicted probabilities for class 1
    y_pred_prob = model.predict_proba(X_test)[:, 1]

    if plot == True:
        # histogram of predicted probabilities
        plt.rcParams['font.size'] = 12
```

```
        plt.hist(y_pred_prob, bins=8)

        # x-axis limit from 0 to 1
        plt.xlim(0,1)
        plt.title('Histogram of predicted probabilities')
        plt.xlabel('Predicted probability of treatment')
        plt.ylabel('Frequency')


    # predict treatment if the predicted probability is greater than 0.3
    # it will return 1 for all values above 0.3 and 0 otherwise
    # results are 2D so we slice out the first column
    y_pred_prob = y_pred_prob.reshape(-1,1)
    y_pred_class = binarize(y_pred_prob, threshold=0.3)[0]

     #ROC Curves and Area Under the Curve (AUC)

    #AUC is the percentage of the ROC plot that is underneath the curve
    #Higher value = better classifier
    roc_auc = metrics.roc_auc_score(y_test, y_pred_prob)



    # IMPORTANT: first argument is true values, second argument is predicted
probabilities
    # roc_curve returns 3 objects fpr, tpr, thresholds
    # fpr: false positive rate
    # tpr: true positive rate
    fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_prob)
    if plot == True:
        plt.figure()

        plt.plot(fpr, tpr, color='darkorange', label='ROC curve (area = %0.2f)' %
roc_auc)
        plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.0])
        plt.rcParams['font.size'] = 12
        plt.title('ROC curve for treatment classifier')
        plt.xlabel('False Positive Rate (1 - Specificity)')
        plt.ylabel('True Positive Rate (Sensitivity)')
        plt.legend(loc="lower right")
```

```
        plt.show()

    # define a function that accepts a threshold and prints sensitivity and specificity
    def evaluate_threshold(threshold):
        #Sensitivity: When the actual value is positive, how often is the prediction
correct?
        #Specificity: When the actual value is negative, how often is the prediction
correct?print('Sensitivity for ' + str(threshold) + ' :', tpr[thresholds >
threshold][-1])
        print('Specificity for ' + str(threshold) + ' :', 1 - fpr[thresholds >
threshold][-1])

    # One way of setting threshold
    predict_mine = np.where(y_pred_prob > 0.50, 1, 0)
    confusion = metrics.confusion_matrix(y_test, predict_mine)
    print(confusion)



    return accuracy


# In[102]:


def tuningRandomizedSearchCV(model, param_dist):
    #Searching multiple parameters simultaneously
    # n_iter controls the number of searches
    rand = RandomizedSearchCV(model, param_dist, cv=10, scoring='accuracy', n_iter=10,
random_state=5)
    rand.fit(X, y)
    rand.cv_results_

    # examine the best model
    print('Rand. Best Score: ', rand.best_score_)
    print('Rand. Best Params: ', rand.best_params_)

    # run RandomizedSearchCV 20 times (with n_iter=10) and record the best score
    best_scores = []
    for _ in range(20):
        rand = RandomizedSearchCV(model, param_dist, cv=10, scoring='accuracy',
n_iter=10)
```

```
        rand.fit(X, y)
        best_scores.append(round(rand.best_score_, 3))
    print(best_scores)


# # Evaluating models

# ## Logistic Regression

# In[103]:


def logisticRegression():
    # train a logistic regression model on the training set
    logreg = LogisticRegression()
    logreg.fit(X_train, y_train)

    # make class predictions for the testing set
    y_pred_class = logreg.predict(X_test)

    accuracy_score = evalClassModel(logreg, y_test, y_pred_class, True)

    #Data for final graph
    methodDict['Logistic Regression'] = accuracy_score * 100


# In[104]:


logisticRegression()


# ## KNeighbors Classifier

# In[105]:


def Knn():
    # Calculating the best parameters
    knn = KNeighborsClassifier(n_neighbors=5)

    # define the parameter values that should be searched
```

```
    k_range = list(range(1, 31))
    weight_options = ['uniform', 'distance']

    # specify "parameter distributions" rather than a "parameter grid"
    param_dist = dict(n_neighbors=k_range, weights=weight_options)
    tuningRandomizedSearchCV(knn, param_dist)

    # train a KNeighborsClassifier model on the training set
    knn = KNeighborsClassifier(n_neighbors=27, weights='uniform')
    knn.fit(X_train, y_train)

    # make class predictions for the testing set
    y_pred_class = knn.predict(X_test)

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

    #Data for final graph
    methodDict['K-Neighbors'] = accuracy_score * 100


# In[106]:


Knn()


# ## Decision Tree classifier

# In[107]:


def treeClassifier():
    # Calculating the best parameters
    tree = DecisionTreeClassifier()
    featuresSize = feature_cols.__len__()
    param_dist = {"max_depth": [3, None],
                  "max_features": randint(1, featuresSize),
                  "min_samples_split": randint(2, 9),
                  "min_samples_leaf": randint(1, 9),
                  "criterion": ["gini", "entropy"]}
    tuningRandomizedSearchCV(tree, param_dist)
```

```
    # train a decision tree model on the training set
    tree = DecisionTreeClassifier(max_depth=3, min_samples_split=8, max_features=6,
criterion='entropy', min_samples_leaf=7)
    tree.fit(X_train, y_train)

    # make class predictions for the testing set
    y_pred_class = tree.predict(X_test)

    accuracy_score = evalClassModel(tree, y_test, y_pred_class, True)

    #Data for final graph
    methodDict['Decision Tree Classifier'] = accuracy_score * 100


# In[108]:


treeClassifier()


# ## Random Forests

# In[109]:


def randomForest():
    # Calculating the best parameters
    forest = RandomForestClassifier(n_estimators = 20)

    featuresSize = feature_cols.__len__()
    param_dist = {"max_depth": [1,2,3,4,5],
                  "max_features": randint(1, featuresSize),
                  "min_samples_split": randint(2, 9),
                  "min_samples_leaf": randint(1, 9),
                  "criterion": ["gini", "entropy"]}
    tuningRandomizedSearchCV(forest, param_dist)

    # Building and fitting my_forest
    forest = RandomForestClassifier( criterion='gini' ,max_depth = 1,
min_samples_leaf=8, min_samples_split=3, n_estimators = 20, random_state = 1)
    my_forest = forest.fit(X_train, y_train)
```

```
    # make class predictions for the testing set
    y_pred_class = my_forest.predict(X_test)

    accuracy_score = evalClassModel(my_forest, y_test, y_pred_class, True)

    #Data for final graph
    methodDict['Random Forest'] = accuracy_score * 100


# In[110]:


randomForest()


# ## Bagging

# In[111]:


def bagging():
    # Building and fitting
    bag = BaggingClassifier(DecisionTreeClassifier(), max_samples=1.0,
max_features=1.0, bootstrap_features=False)
    bag.fit(X_train, y_train)

    # make class predictions for the testing set
    y_pred_class = bag.predict(X_test)

    accuracy_score = evalClassModel(bag, y_test, y_pred_class, True)

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


# In[112]:


bagging()


# ## Boosting
```

```
# In[113]:


def boosting():
    # Building and fitting
    clf = DecisionTreeClassifier(criterion='entropy', max_depth=1)
    boost = AdaBoostClassifier(base_estimator=clf, n_estimators=500)
    boost.fit(X_train, y_train)

    # make class predictions for the testing set
    y_pred_class = boost.predict(X_test)

    accuracy_score = evalClassModel(boost, y_test, y_pred_class, True)
    #Data for final graph
    methodDict['Boosting'] = accuracy_score * 100


# In[114]:


boosting()


# In[115]:


methodDict


# # Since boosting has the highest metrics we choose to hypertune Boosting

# In[154]:


def bestBoosting():
    # Building and fitting
    clf = DecisionTreeClassifier(criterion='entropy', max_depth=1)
    boost = AdaBoostClassifier(base_estimator=clf)
    tuningRandomizedSearchCV(boost, {
        "n_estimators":[int(x) for x in np.linspace(start=10,stop=100,num=15)],
        'learning_rate':[(0.97+x/100) for x in range(0,20)],
```

```
    })


# In[155]:


bestBoosting()


# In[156]:


final_clf=AdaBoostClassifier(DecisionTreeClassifier(criterion='entropy',
max_depth=1),n_estimators = 16,learning_rate=1.16)
final_clf.fit(X_train, y_train)

y_pred_class = final_clf.predict(X_test)

accuracy_score = evalClassModel(final_clf, y_test, y_pred_class, True)


# # Creating predictions on test set

# In[157]:


# Generate predictions with the best method
final_clf.fit(X, y)


# In[158]:


pickle.dump(final_clf,open('model.pkl','wb'))
joblib.dump(ct.named_transformers_['scaler'],'scaler')


# In[ ]:
```