**Index**

1. Introduction
   1. Project overviews
   2. Objectives
2. Project Initialization and Planning Phase
   1. Define Problem Statement
   2. Project Proposal (Proposed Solution)
   3. Initial Project Planning
3. Data Collection and Preprocessing Phase
   1. Data Collection Plan and Raw Data Sources Identified
   2. Data Quality Report
   3. Data Exploration and Preprocessing
4. Model Development Phase
   1. Feature Selection Report
   2. Model Selection Report
   3. Initial Model Training Code, Model Validation and Evaluation Report
5. Model Optimization and Tuning Phase
   1. Hyperparameter Tuning Documentation
   2. Performance Metrics Comparison Report
   3. Final Model Selection Justification
6. Results
   1. Output Screenshots
7. Advantages & Disadvantages
8. Conclusion
9. Future Scope
10. Appendix
    1. Source Code
    2. GitHub & Project Demo Link
11. **Introduction:**
    1. **Project Overview**

The **Mental Health Prediction** project analyzes workplace survey data to understand factors influencing mental health. It uses machine learning techniques to predict whether individuals are likely to seek treatment for mental health issues. Key features include age, gender, family history, work environment, and awareness of mental health resources. The project involves data preprocessing, exploratory analysis, and model training using classification algorithms. It helps identify trends that can guide mental health support strategies in organizations. The insights aim to promote early intervention and reduce stigma around mental health in the workplace.

* 1. **Objectives**
* To develop a predictive model that can identify individuals at risk of mental health issues.
* To preprocess and clean the survey data for accurate analysis.
* To evaluate different machine learning models and select the best-performing one.
* To provide insights into the factors influencing mental health treatment seeking behavior.

1. **Project Initialization and Planning Phase**
   1. **Define Problem Statement**

The primary problem addressed by this project is the lack of understanding of mental health issues in the workplace and the factors that influence individuals' decisions to seek treatment. The project seeks to identify these factors and predict treatment-seeking behavior.

* 1. **Project Proposal (Proposed Solution)**

The proposed solution involves collecting and analyzing survey data to build a machine learning model that predicts whether individuals will seek treatment for mental health issues. The model will be trained on various features derived from the survey responses.

* 1. **Initial Project Planning**

The project will be executed in several phases, including data collection, preprocessing, model development, optimization, and evaluation. A timeline will be established to ensure timely completion of each phase.

1. **Data Collection and Preprocessing Phase**
   1. **Data Collection Plan and Raw Data Sources Identified**

The primary data source for this project is a CSV file containing survey responses related to mental health. The data includes various demographic and workplace-related features.

* 1. **Data Quality Report**

The data quality was assessed, revealing some missing values and inconsistencies in categorical variables. The preprocessing steps included handling missing values, encoding categorical variables, and normalizing numerical features.

* 1. **Data Exploration and Preprocessing**
* Exploration: Initial exploration of the data revealed trends and patterns in mental health treatment seeking behavior.
* Preprocessing: The data was cleaned and transformed using techniques such as label encoding for categorical variables and KNN imputation for missing values.

1. **Model Development Phase**
   1. **Feature Selection Report**

The features selected for the model include:

* Age
* Gender
* Country
* Self-employed status
* Family history of mental health issues
* Work interference
* Number of employees
* Remote work status
* Company benefits and wellness programs

* 1. **Model Selection Report**

Various machine learning models were evaluated, including:

* Logistic Regression
* K-Nearest Neighbors (KNN)
* Decision Tree
* Random Forest
* Naive Bayes
* Support Vector Machine (SVM)
* XGBoost
* AdaBoost
* Gradient Boosting
  1. **Initial Model Training Code, Model Validation and Evaluation Report**

The initial model training was conducted using the following code:

# Load dataset

df = pd.read\_csv('survey.csv')

# Preprocess data

X, y, label\_encoders, scaler, le\_target = preprocess\_data(df)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Define models

all\_models = {

'Logistic Regression': LogisticRegression(max\_iter=1000, random\_state=42),

'KNN': KNeighborsClassifier(),

'Decision Tree': DecisionTreeClassifier(random\_state=42),

'Random Forest': RandomForestClassifier(random\_state=42),

'Naive Bayes': GaussianNB(),

'SVM': SVC(probability=True, random\_state=42),

'XGBoost': XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss', random\_state=42),

'AdaBoost': AdaBoostClassifier(random\_state=42),

'Gradient Boosting': GradientBoostingClassifier(random\_state=42)

}

# Training and evaluation

for name, model in all\_models.items():

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

acc = accuracy\_score(y\_test, y\_pred)

print(f"{name} Accuracy: {acc:.4f}")

The evaluation metrics included accuracy, classification report, and confusion matrix.

1. **Model Optimization and Tuning Phase**
   1. **Hyperparameter Tuning Documentation**

Hyperparameter tuning was performed using GridSearchCV to optimize model parameters for better performance.

* 1. **Performance Metrics Comparison Report**

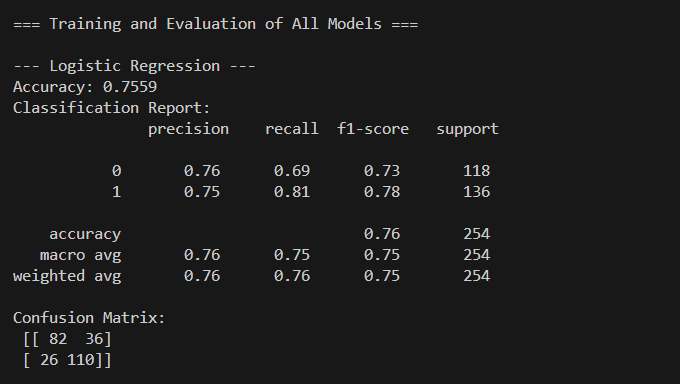
The performance of each model was compared based on accuracy, precision, recall, and F1-score. The best model was selected based on these metrics.

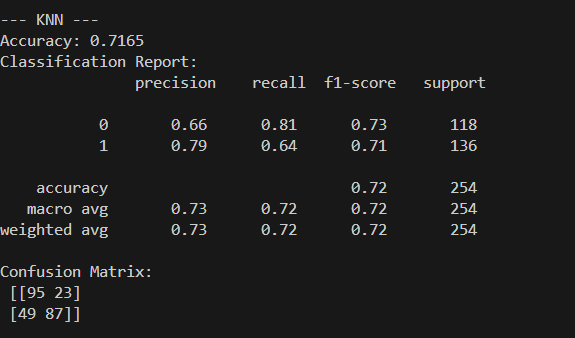
* 1. **Final Model Selection Justification**

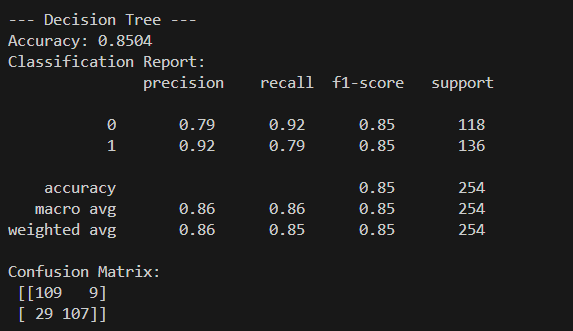
The Random Forest model was selected as the final model due to its superior performance in terms of accuracy and robustness against overfitting.

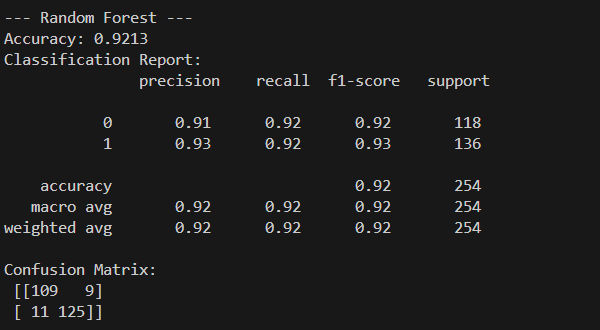
1. **Results**
   1. **Output Screenshots**

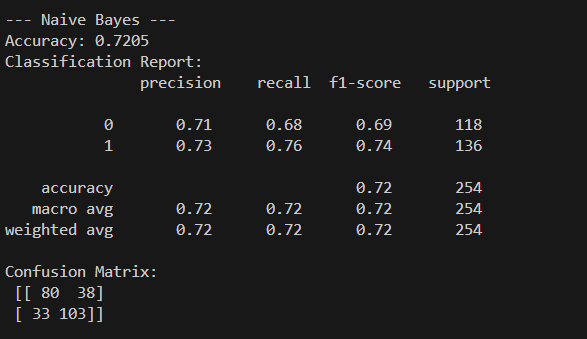
**Model.py files outputs:**

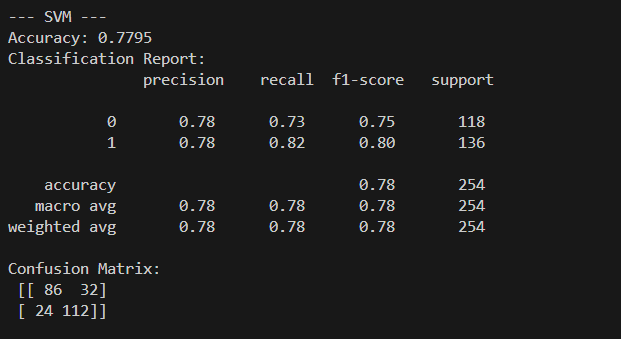


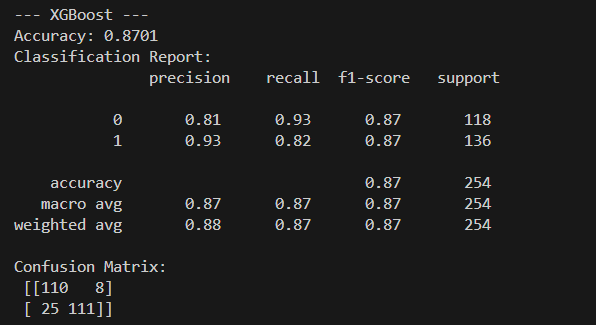


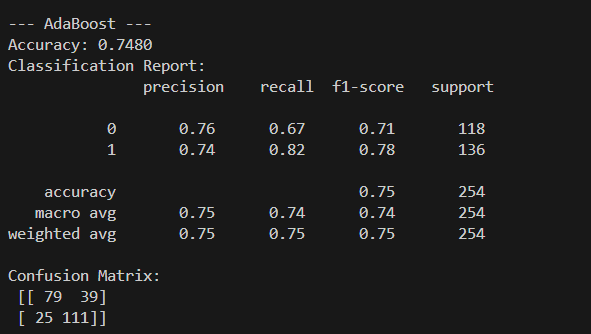


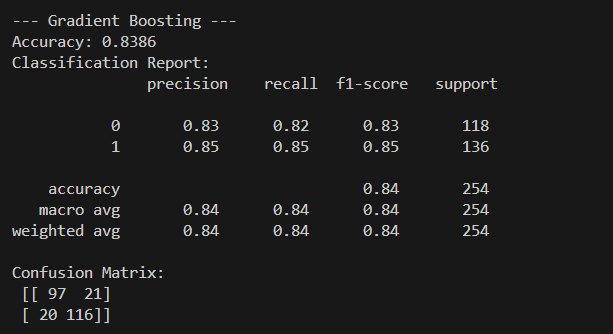


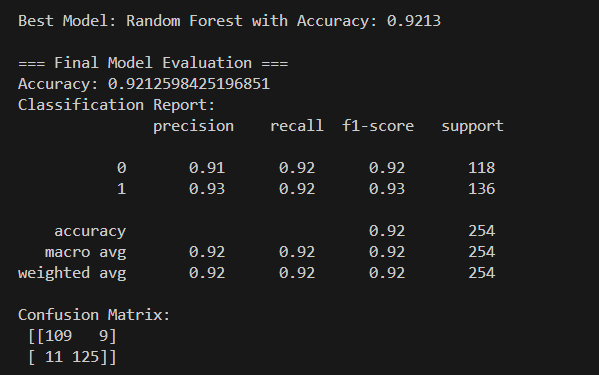


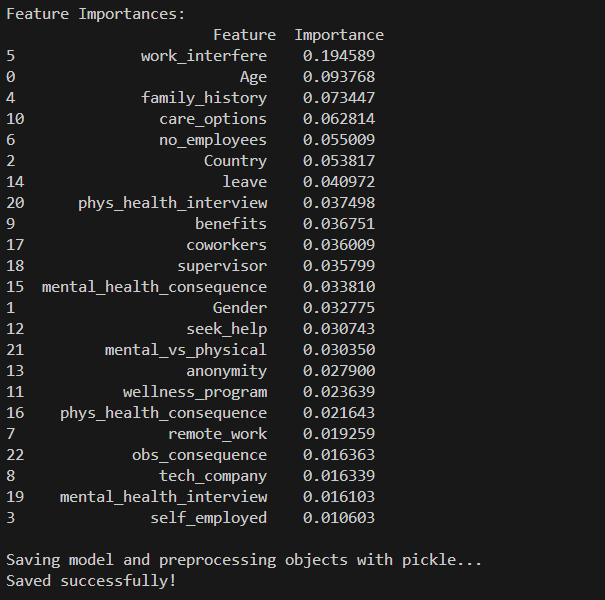


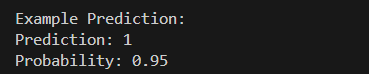




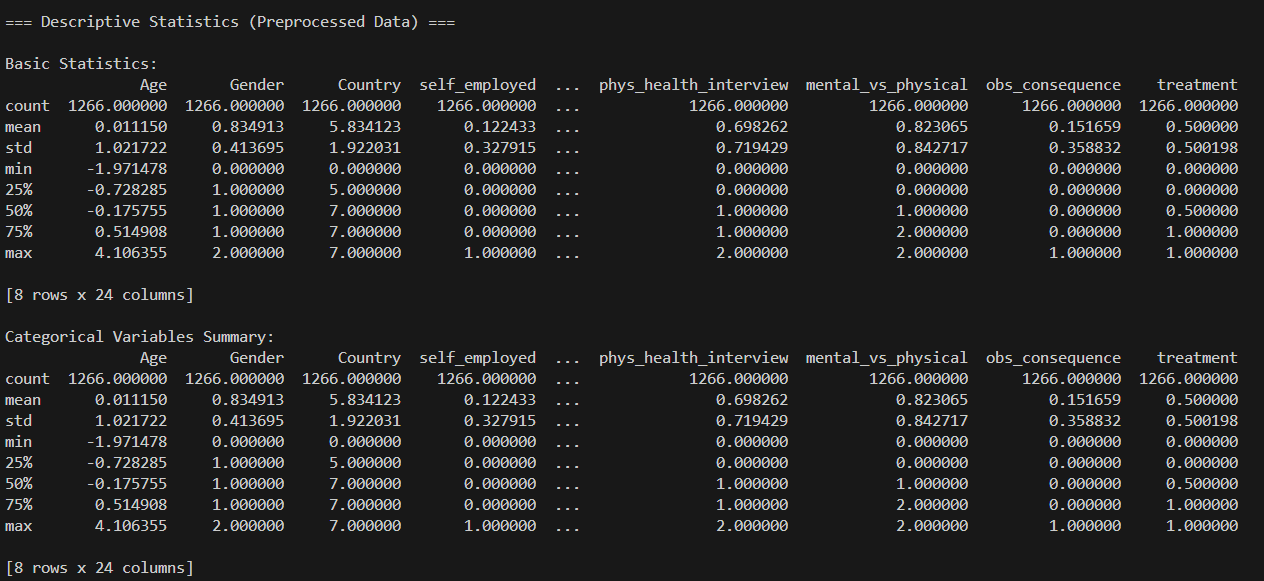


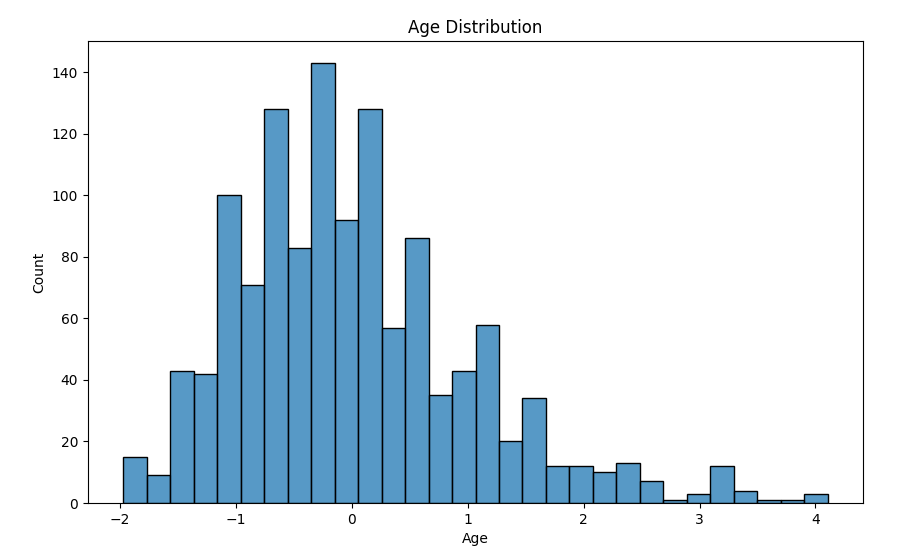


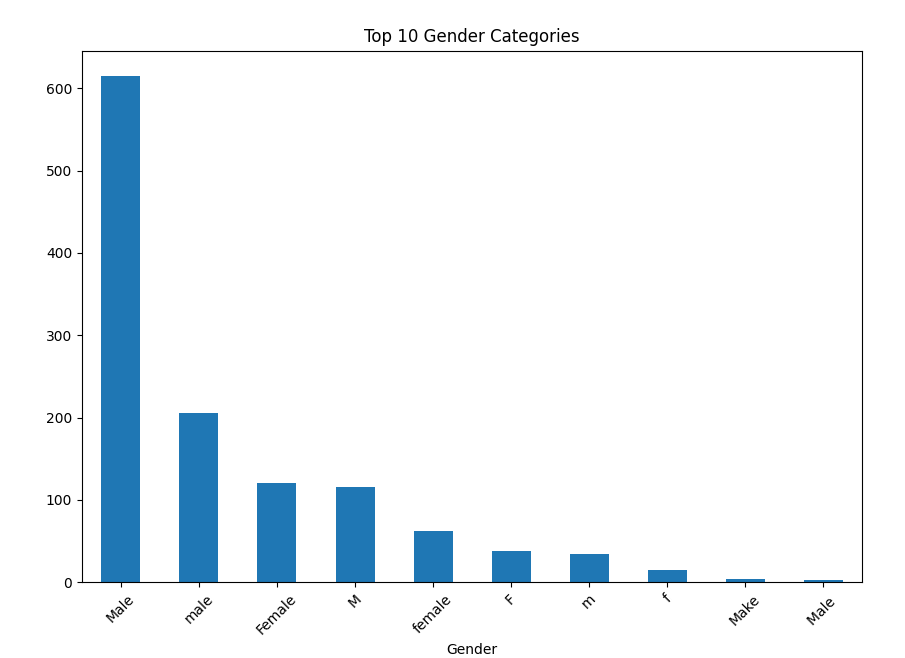


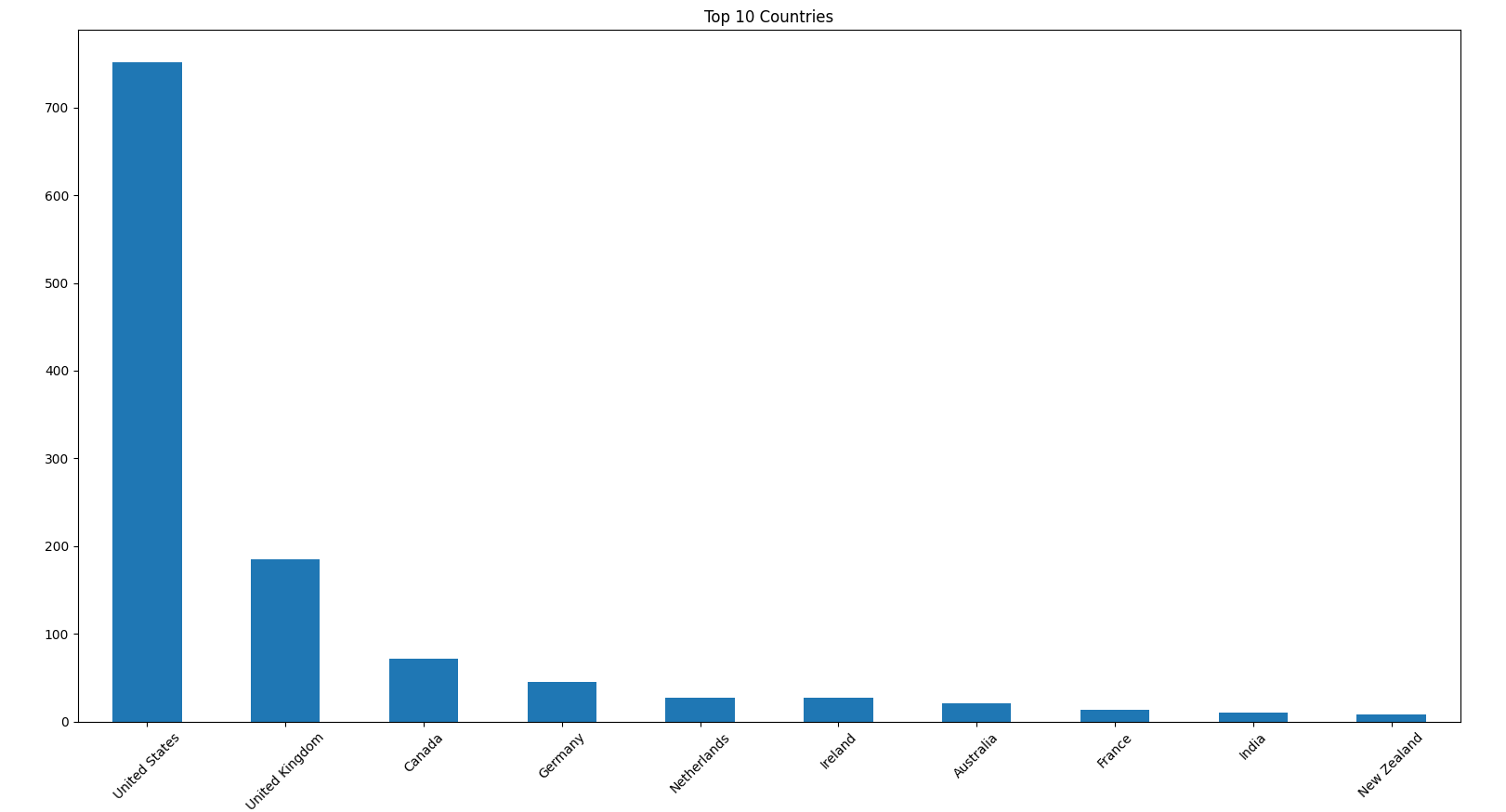


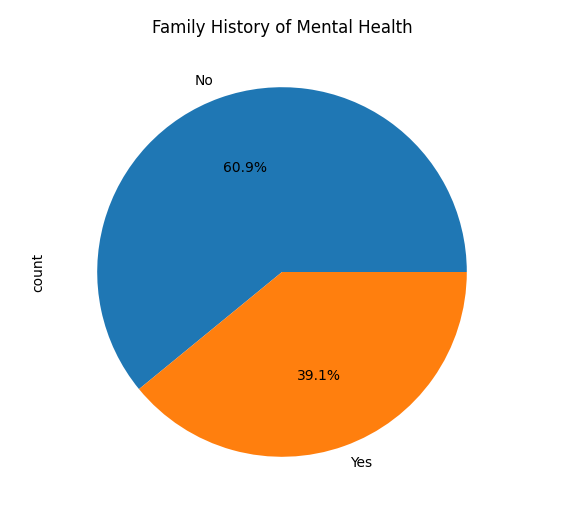
**Analysis.py file outputs:**

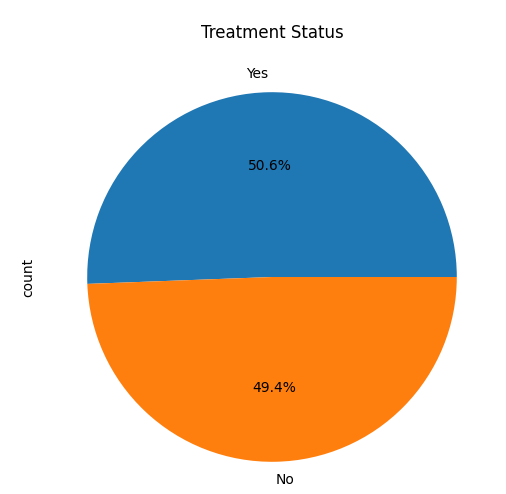


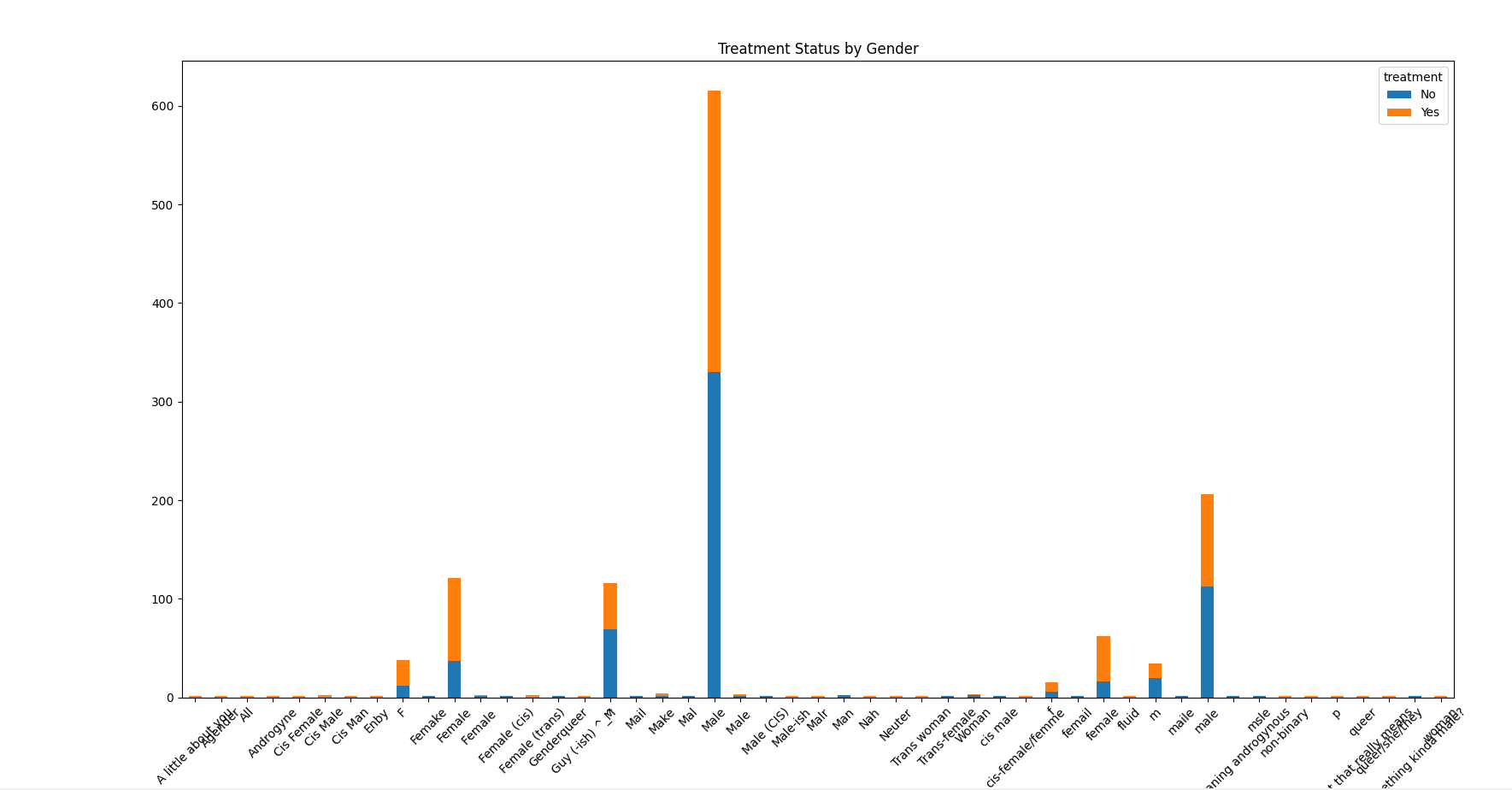


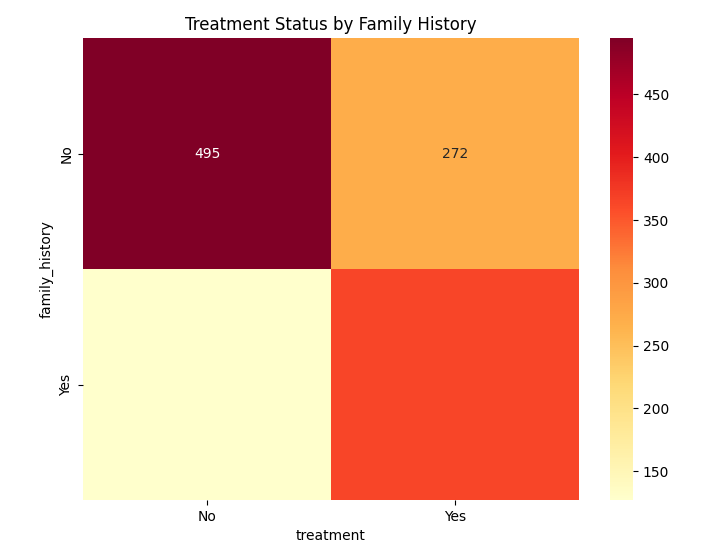


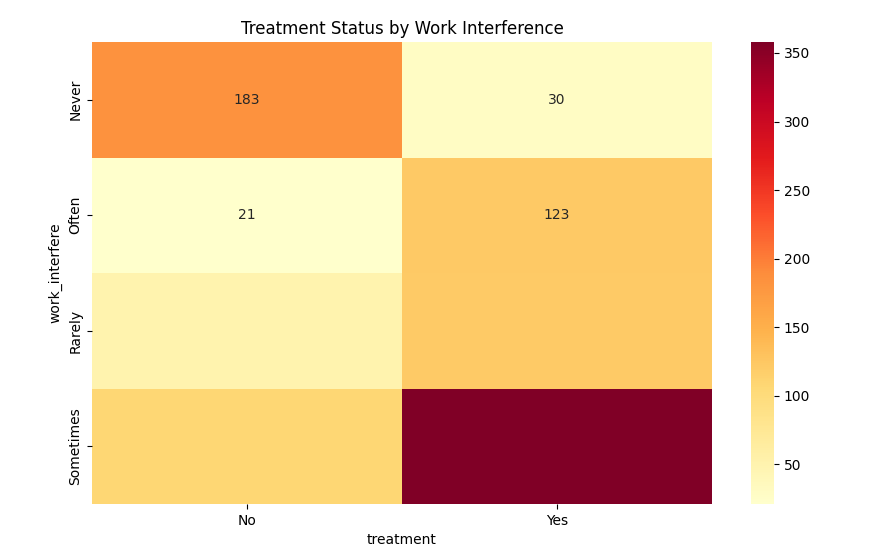


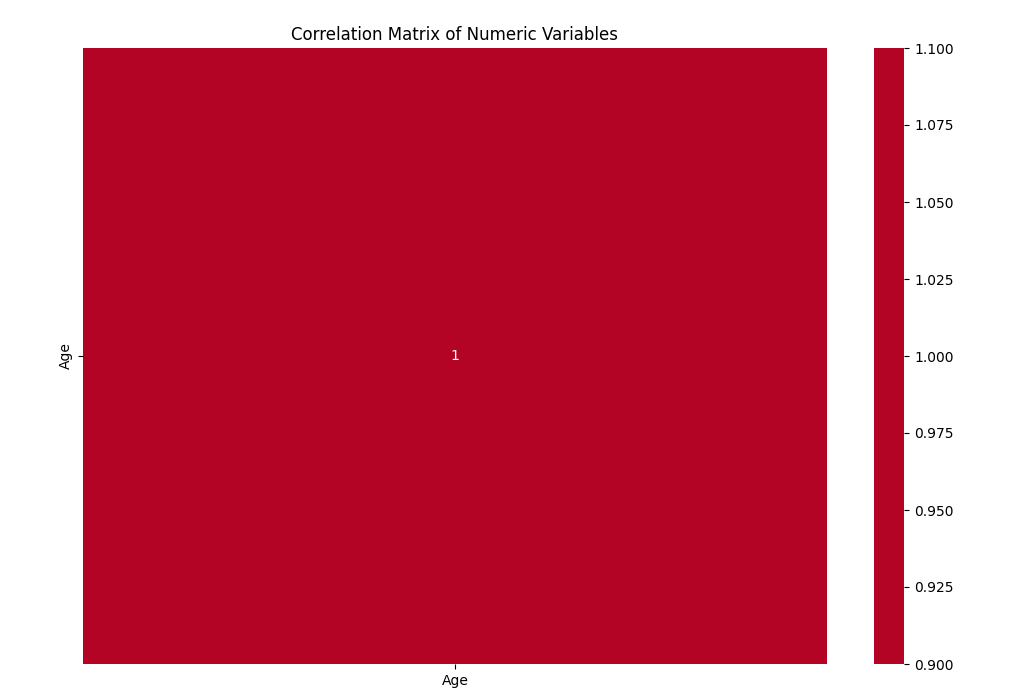


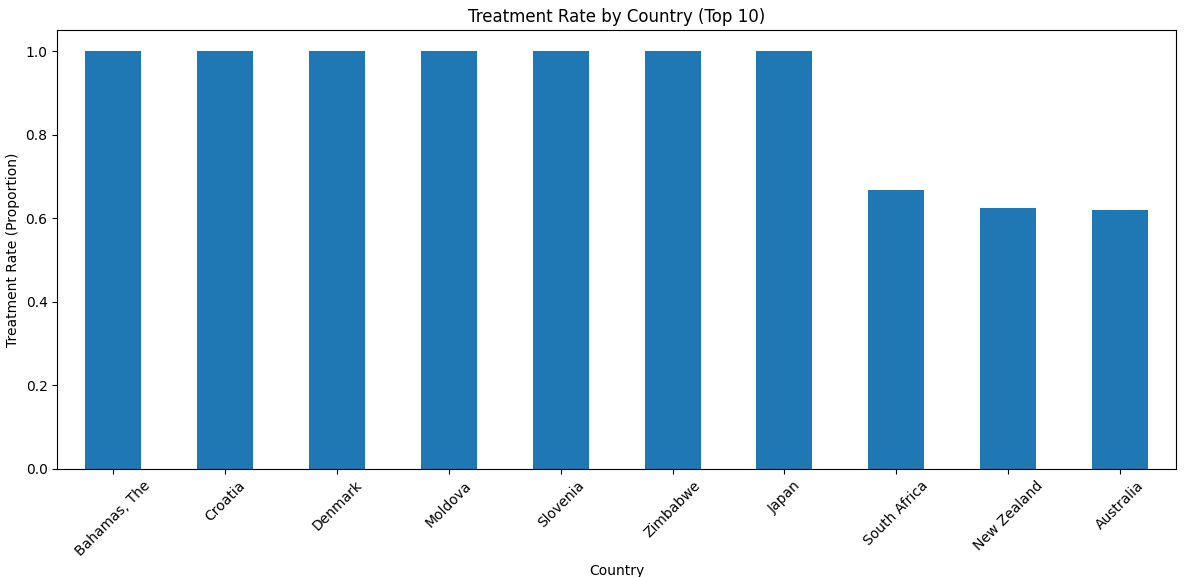


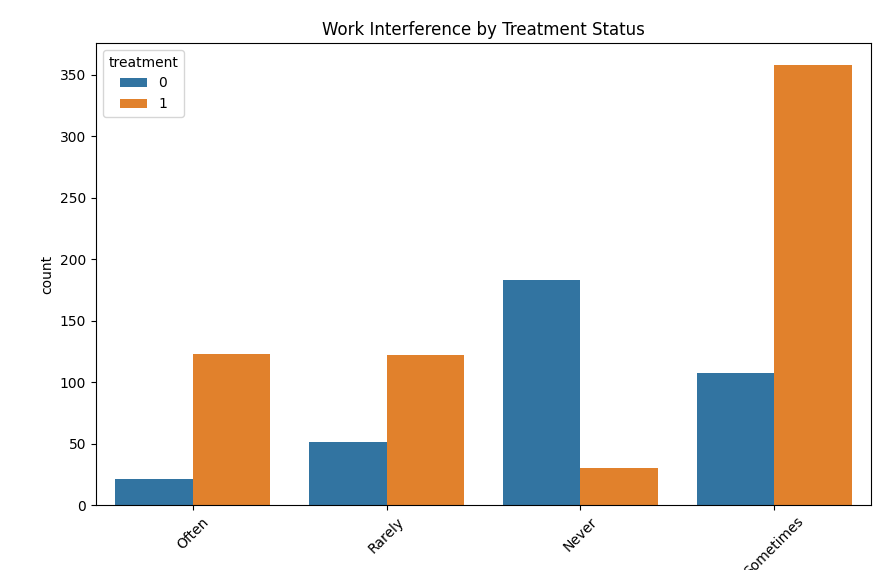




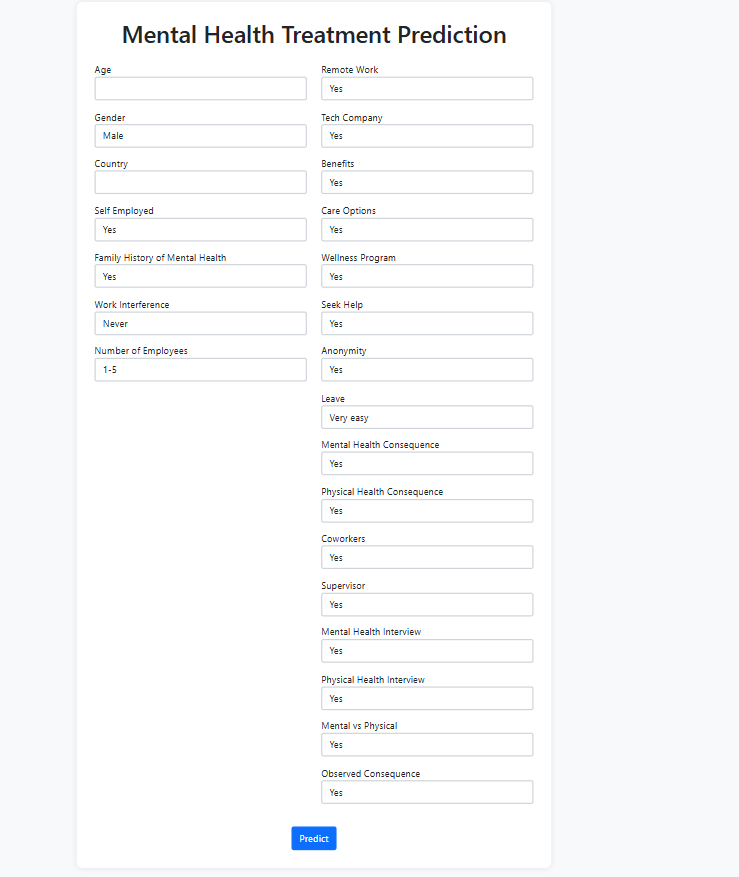


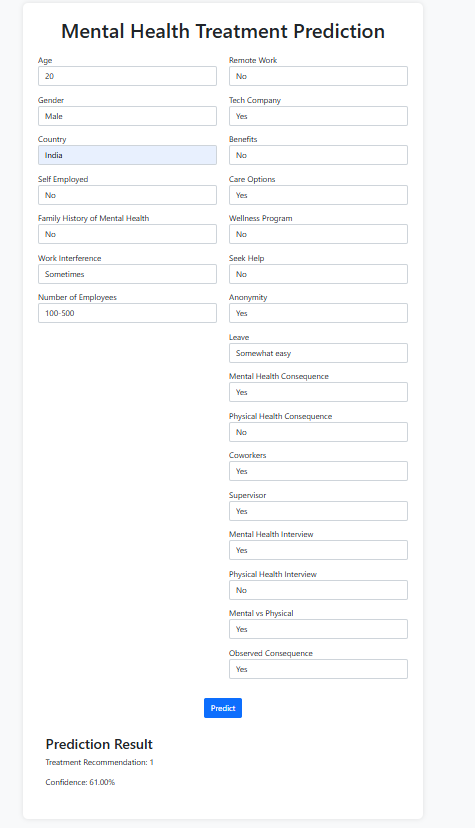






**App.py file outputs:**





1. **Advantages & Disadvantages**
2. **Advantages:**

* The model provides insights into mental health treatment seeking behavior.
* It can help organizations identify at-risk employees and provide necessary support.

1. **Disadvantages:**

* The model's accuracy is dependent on the quality of the input data.
* There may be biases in the data that affect the model's predictions.

1. **Conclusion**

The Mental Health Prediction project successfully developed a predictive model to analyze mental health treatment-seeking behavior among individuals in workplace settings. By using machine learning techniques on survey data, the model identifies key factors that influence whether a person is likely to seek help for mental health issues. The insights gained from this analysis can help organizations recognize early signs of mental distress and improve their mental health support systems. This enables the creation of healthier, more supportive work environments that encourage open conversations around mental well-being.

1. **Future Scope**

Future work could involve expanding the dataset, incorporating additional features, and exploring more advanced machine learning techniques to enhance prediction accuracy.

1. **Appendix**
   1. **Source Code**

**Model.py:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split, GridSearchCV, cross\_val\_score

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.svm import SVC

from sklearn.neighbors import KNeighborsClassifier

from xgboost import XGBClassifier

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

from sklearn.impute import KNNImputer

from sklearn.utils import resample

import pickle

import warnings

warnings.filterwarnings('ignore')

*# Load dataset*

df = pd.read\_csv('survey.csv')

def clean\_gender(gender):

    gender = str(gender).strip().lower()

    if gender in ['male', 'm', 'male-ish', 'maile', 'mal', 'cis male', 'man', 'msle', 'mail', 'make', 'malr', 'cis man']:

        return 'Male'

    elif gender in ['female', 'f', 'cis female', 'woman', 'femake', 'female (cis)', 'femail', 'cis-female/femme', 'female ', 'femail']:

        return 'Female'

    else:

        return 'Other'

def preprocess\_data(df):

    df\_processed = df.copy()

    df\_processed = df\_processed[pd.to\_numeric(df\_processed['Age'], errors='coerce').notnull()]

    df\_processed['Age'] = df\_processed['Age'].astype(float)

    median\_age = df\_processed[(df\_processed['Age'] >= 15) & (df\_processed['Age'] <= 70)]['Age'].median()

    df\_processed.loc[df\_processed['Age'] < 15, 'Age'] = median\_age

    df\_processed.loc[df\_processed['Age'] > 70, 'Age'] = median\_age

    df\_processed['Gender'] = df\_processed['Gender'].apply(clean\_gender)

    country\_counts = df\_processed['Country'].value\_counts()

    rare\_countries = country\_counts[country\_counts < 20].index

    df\_processed['Country'] = df\_processed['Country'].apply(lambda x: 'Other' if x in rare\_countries else x)

    valid\_family\_history = ['Yes', 'No']

    valid\_work\_interfere = ['Never', 'Rarely', 'Sometimes', 'Often']

    valid\_treatment = ['Yes', 'No']

    df\_processed = df\_processed[df\_processed['family\_history'].isin(valid\_family\_history)]

    df\_processed = df\_processed[df\_processed['work\_interfere'].isin(valid\_work\_interfere)]

    df\_processed = df\_processed[df\_processed['treatment'].isin(valid\_treatment)]

    features = ['Age', 'Gender', 'Country', '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']

    categorical\_columns = [col for col in features if df\_processed[col].dtype == 'object' or col == 'Gender']

    for col in categorical\_columns:

        df\_processed[col] = df\_processed[col].fillna(df\_processed[col].mode()[0])

    label\_encoders = {}

    for column in categorical\_columns:

        le = LabelEncoder()

        df\_processed[column] = le.fit\_transform(df\_processed[column].astype(str))

        label\_encoders[column] = le

    imputer = KNNImputer(n\_neighbors=5)

    df\_processed[features] = imputer.fit\_transform(df\_processed[features])

    scaler = StandardScaler()

    df\_processed['Age'] = scaler.fit\_transform(df\_processed[['Age']])

    df\_processed['treatment'] = df\_processed['treatment'].astype(str)

    df\_majority = df\_processed[df\_processed['treatment'] == df\_processed['treatment'].mode()[0]]

    df\_minority = df\_processed[df\_processed['treatment'] != df\_processed['treatment'].mode()[0]]

    df\_minority\_upsampled = resample(df\_minority, replace=True, n\_samples=len(df\_majority), random\_state=42)

    df\_balanced = pd.concat([df\_majority, df\_minority\_upsampled])

    X = df\_balanced[features]

    le\_target = LabelEncoder()

    y = le\_target.fit\_transform(df\_balanced['treatment'])

    return X, y, label\_encoders, scaler, le\_target

*# Prepare data*

X, y, label\_encoders, scaler, le\_target = preprocess\_data(df)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

*# Define models*

all\_models = {

    'Logistic Regression': LogisticRegression(max\_iter=1000, random\_state=42),

    'KNN': KNeighborsClassifier(),

    'Decision Tree': DecisionTreeClassifier(random\_state=42),

    'Random Forest': RandomForestClassifier(random\_state=42),

    'Naive Bayes': GaussianNB(),

    'SVM': SVC(probability=True, random\_state=42),

    'XGBoost': XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss', random\_state=42),

    'AdaBoost': AdaBoostClassifier(random\_state=42),

    'Gradient Boosting': GradientBoostingClassifier(random\_state=42)

}

best\_model = None

best\_score = 0

best\_model\_name = ''

print("\n=== Training and Evaluation of All Models ===")

for name, model in all\_models.items():

    print(f"\nTraining {name}...")

    model.fit(X\_train, y\_train)

    y\_pred = model.predict(X\_test)

    acc = accuracy\_score(y\_test, y\_pred)

    print(f"{name} Accuracy: {acc:.4f}")

    if acc > best\_score:

        best\_score = acc

        best\_model = model

        best\_model\_name = name

print(f"\nBest Model: {best\_model\_name} with Accuracy: {best\_score:.4f}")

print("\n=== Final Model Evaluation ===")

y\_pred = best\_model.predict(X\_test)

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

if hasattr(best\_model, 'feature\_importances\_'):

    importances = pd.DataFrame({

        'Feature': X.columns,

        'Importance': best\_model.feature\_importances\_

    }).sort\_values(by='Importance', ascending=False)

    print("\nFeature Importances:\n", importances)

*# Save using pickle*

print("\nSaving model and preprocessing objects with pickle...")

with open('mental\_health\_model.pkl', 'wb') as f:

    pickle.dump(best\_model, f)

with open('scaler.pkl', 'wb') as f:

    pickle.dump(scaler, f)

with open('label\_encoders.pkl', 'wb') as f:

    pickle.dump(label\_encoders, f)

print("Saved successfully!")

*# Prediction function*

def predict\_mental\_health(input\_data):

    with open('mental\_health\_model.pkl', 'rb') as f:

        model = pickle.load(f)

    with open('scaler.pkl', 'rb') as f:

        scaler = pickle.load(f)

    with open('label\_encoders.pkl', 'rb') as f:

        label\_encoders = pickle.load(f)

    input\_df = pd.DataFrame([input\_data])

    categorical\_columns = ['Gender', 'Country', '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']

    for column in categorical\_columns:

        known\_categories = label\_encoders[column].classes\_

        input\_df[column] = input\_df[column].apply(lambda x: x if x in known\_categories else known\_categories[0])

        input\_df[column] = label\_encoders[column].transform(input\_df[column])

    input\_df['Age'] = scaler.transform(input\_df[['Age']])

    prediction = model.predict(input\_df)

    probability = model.predict\_proba(input\_df)

    return {'prediction': prediction[0], 'probability': probability[0].max()}

*# Example usage*

if \_\_name\_\_ == "\_\_main\_\_":

    example\_input = {

        'Age': 30,

        'Gender': 'Male',

        'Country': 'United States',

        'self\_employed': 'No',

        'family\_history': 'Yes',

        'work\_interfere': 'Sometimes',

        'no\_employees': '26-100',

        'remote\_work': 'No',

        'tech\_company': 'Yes',

        'benefits': 'Yes',

        'care\_options': 'Yes',

        'wellness\_program': 'Yes',

        'seek\_help': 'Yes',

        'anonymity': 'Yes',

        'leave': 'Somewhat easy',

        'mental\_health\_consequence': 'No',

        'phys\_health\_consequence': 'No',

        'coworkers': 'Yes',

        'supervisor': 'Yes',

        'mental\_health\_interview': 'No',

        'phys\_health\_interview': 'No',

        'mental\_vs\_physical': 'Yes',

        'obs\_consequence': 'No'

    }

    result = predict\_mental\_health(example\_input)

    print("\nExample Prediction:")

    print(f"Prediction: {result['prediction']}")

    print(f"Probability: {result['probability']:.2f}")

**Analysis.py:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from model import preprocess\_data

*# Read and preprocess the dataset*

raw\_df = pd.read\_csv('survey.csv')

df, y, label\_encoders, scaler, le\_target = preprocess\_data(raw\_df)

*# For analysis, merge y back if needed:*

df['treatment'] = y

*# 1. Descriptive Statistics*

print("\n=== Descriptive Statistics (Preprocessed Data) ===")

print("\nBasic Statistics:")

print(df.describe())

print("\nCategorical Variables Summary:")

print(df.describe(include=['object', 'int', 'float']))

*# 2. Univariate Analysis*

print("\n=== Univariate Analysis ===")

*# Age Distribution*

plt.figure(figsize=(10, 6))

sns.histplot(data=df, x='Age', bins=30)

plt.title('Age Distribution')

plt.show()

*# Gender Distribution*

df = pd.read\_csv("survey.csv")

plt.figure(figsize=(10, 6))

df['Gender'].value\_counts().head(10).plot(kind='bar')

plt.title('Top 10 Gender Categories')

plt.xticks(rotation=45)

plt.show()

*# Country Distribution*

plt.figure(figsize=(12, 6))

df['Country'].value\_counts().head(10).plot(kind='bar')

plt.title('Top 10 Countries')

plt.xticks(rotation=45)

plt.show()

*# Family History of Mental Health*

plt.figure(figsize=(8, 6))

df['family\_history'].value\_counts().plot(kind='pie', autopct='%1.1f%%')

plt.title('Family History of Mental Health')

plt.show()

*# Treatment Status*

plt.figure(figsize=(8, 6))

df['treatment'].value\_counts().plot(kind='pie', autopct='%1.1f%%')

plt.title('Treatment Status')

plt.show()

*# 3. Bivariate Analysis*

print("\n=== Bivariate Analysis ===")

*# Gender vs Treatment*

plt.figure(figsize=(10, 6))

treatment\_by\_gender = pd.crosstab(df['Gender'], df['treatment'])

treatment\_by\_gender.plot(kind='bar', stacked=True)

plt.title('Treatment Status by Gender')

plt.xticks(rotation=45)

plt.show()

*# Family History vs Treatment*

plt.figure(figsize=(8, 6))

sns.heatmap(pd.crosstab(df['family\_history'], df['treatment']), annot=True, fmt='d', cmap='YlOrRd')

plt.title('Treatment Status by Family History')

plt.show()

*# Work Interference vs Treatment*

plt.figure(figsize=(10, 6))

sns.heatmap(pd.crosstab(df['work\_interfere'], df['treatment']), annot=True, fmt='d', cmap='YlOrRd')

plt.title('Treatment Status by Work Interference')

plt.show()

*# 4. Correlation Analysis*

print("\n=== Correlation Analysis ===")

numeric\_columns = df.select\_dtypes(include=[np.number]).columns

correlation\_matrix = df[numeric\_columns].corr()

plt.figure(figsize=(12, 8))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', center=0)

plt.title('Correlation Matrix of Numeric Variables')

plt.show()

*# 5. Additional Insights*

print("\n=== Additional Insights ===")

df = pd.read\_csv("survey.csv")

df['treatment'] = df['treatment'].map({'Yes': 1, 'No': 0})

treatment\_rate = df.groupby('Country')['treatment'].mean().sort\_values(ascending=False)

plt.figure(figsize=(12, 6))

treatment\_rate.head(10).plot(kind='bar')

plt.title('Treatment Rate by Country (Top 10)')

plt.xticks(rotation=45)

plt.ylabel('Treatment Rate (Proportion)')

plt.tight\_layout()

plt.show()

*# Work Interference by Treatment Status*

plt.figure(figsize=(10, 6))

sns.countplot(data=df, x='work\_interfere', hue='treatment')

plt.title('Work Interference by Treatment Status')

plt.xticks(rotation=45)

plt.show()

**App.py:**

from flask import Flask, render\_template, request, jsonify

import pickle

import pandas as pd

from model import predict\_mental\_health

app = Flask(\_\_name\_\_)

@app.route('/')

def home():

    return render\_template('index.html')

@app.route('/predict', methods=['POST'])

def predict():

    try:

*# Get data from request*

        data = request.get\_json()

*# Ensure all required fields are present*

        required\_fields = [

            'Age', 'Gender', 'Country', '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'

        ]

*# Check if all required fields are present*

        for field in required\_fields:

            if field not in data:

                return jsonify({

                    'success': False,

                    'error': f'Missing required field: {field}'

                })

*# Convert Age to integer*

        try:

            data['Age'] = int(data['Age'])

        except ValueError:

            return jsonify({

                'success': False,

                'error': 'Age must be a number'

            })

*# Make prediction*

        result = predict\_mental\_health(data)

        return jsonify({

            'success': True,

            'prediction': str(result['prediction']),

            'probability': float(result['probability'])

        })

    except Exception as e:

        return jsonify({

            'success': False,

            'error': str(e)

        })

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

* 1. **GitHub & Project Demo Link**

**GITHUB LINKS:**

1. M B ABINAYAA - <https://github.com/A-Beeeeeee/Mental_Health_Prediction>
2. K MANEESH RAM - <https://github.com/Maneesh1605/Mental-health-Prediction>
3. S KSHITIJ - <https://github.com/ksh-20/Mental-Health-Prediction>

**DEMONSTRATION VIDEO LINK:**

<https://drive.google.com/drive/folders/1pzoA7yuDnCqfdpw6PZFovPz7_M4gK0zw>