Team ID	Team- 591754
Project Name	Predicting Mental Health Illness of Working Professionals

TEAM MEMBERS – G.N.M.SHYAM

B.SUBHASH

S.KRISHNA REDDY

L.V.JAYANTH

1.INTRODUCTION

1.1 Project overview

1. **Problem Definition:**
    - Clearly define the problem you want to address: predicting mental health illnesses in working professionals.
    - Specify the scope, such as the types of mental health issues you aim to predict.
2. **Data Collection:**
    - Gather a dataset containing relevant features and labels indicating the presence or absence of mental health illnesses.
    - Ensure the data is representative and diverse, considering various industries, roles, and demographics.
3. **Data Preprocessing:**
    - Handle missing data, outliers, and perform data cleaning.
    - Encode categorical variables and normalize/standardize numerical features.
    - Explore the data to gain insights into patterns and relationships.
4. **Feature Selection/Engineering:**

- Identify the most relevant features that contribute to predicting mental health illnesses.
- Consider creating new features that might enhance the predictive power of the model.

5. **Model Selection:**
    - Choose a suitable machine learning algorithm. Common choices include Random Forest, Support Vector Machines, Logistic Regression, etc.
    - Experiment with multiple models to find the one that performs best for your specific problem.
6. **Model Training:**
    - Split the dataset into training and testing sets.
    - Train the selected model using the training data.
7. **Model Evaluation:**
    - Evaluate the model's performance on the testing set.
    - Metrics may include accuracy, precision, recall, F1-score, and ROC-AUC, depending on the nature of the problem.
8. **Hyperparameter Tuning:**
    - Fine-tune the model's hyperparameters to optimize performance.
    - Consider using techniques like grid search or random search.
9. **Ethical Considerations:**
    - Given the sensitive nature of mental health data, prioritize ethical considerations.
    - Ensure privacy and confidentiality of individuals in the dataset.
    - Collaborate with mental health professionals to ensure responsible model development and usage.

1.2 PURPOSE

1. **Early Identification and Intervention:**
    - Detect potential signs of mental health issues in working professionals at an early stage.
    - Enable timely intervention and support to prevent the escalation of mental health problems.

2. **Promoting Workplace Well-being:**
    - Foster a workplace culture that prioritizes employee well-being and mental health.
    - Provide tools for organizations to proactively address mental health challenges within their workforce.
3. **Customized Support Services:**
    - Tailor support services and resources based on individual needs identified through the predictive model.
    - Offer personalized assistance to employees facing mental health difficulties.
4. **Resource Allocation Optimization:**
    - Assist organizations in allocating resources efficiently by targeting interventions toward individuals at higher risk.
    - Optimize the utilization of mental health support programs and services.
5. **Reducing Stigma and Promoting Open Communication:**
    - Contribute to reducing the stigma associated with mental health discussions in the workplace.
    - Encourage open communication and destigmatize seeking help for mental health concerns.
6. **Data-Driven Decision Making:**
    - Provide organizations with data-driven insights to inform decision-making regarding employee well-being initiatives.
    - Enable evidence-based strategies for creating a mentally healthy work environment.
7. **Employee Productivity and Satisfaction:**
    - Enhance overall employee productivity and job satisfaction by addressing mental health concerns.
    - Improve employee engagement and retention through targeted mental health support.
8. **Preventing Burnout and Absenteeism:**
    - Identify individuals at risk of burnout or high levels of absenteeism due to mental health issues.

- Implement measures to prevent burnout and reduce absenteeism, contributing to a healthier workplace.

9. **Contributing to Research and Knowledge:**
    - Contribute valuable insights to the broader field of mental health research and workplace well-being.
    - Facilitate a better understanding of the factors contributing to mental health challenges in professional settings.
10. **Aligning with Corporate Social Responsibility (CSR):**
    - Demonstrate a commitment to corporate social responsibility by actively addressing mental health concerns among employees.
    - Showcase an organization's dedication to creating a supportive and inclusive workplace.
11. **Legal and Ethical Compliance:**
    - Ensure compliance with legal and ethical standards regarding the handling and analysis of sensitive mental health data.
    - Uphold privacy and confidentiality while using machine learning for mental health predictions.
12. **Long-term Health and Sustainability:**
    - Contribute to the long-term health and sustainability of both individuals and organizations by fostering a mentally healthy workplace culture.

2. LITERATURE SURVEY

2.1 Existing problem

1. **Data Quality and Availability:**
    - Limited availability of high-quality labeled datasets that accurately represent the diversity of mental health issues in working professionals.
    - Potential bias in the data, as individuals may underreport or misrepresent their mental health status due to stigma or fear of consequences.
2. **Privacy Concerns:**
    - The sensitive nature of mental health data raises significant privacy concerns. Ensuring compliance with data protection regulations (such as GDPR or HIPAA) is crucial.
    - Striking a balance between utilizing valuable data for prediction and protecting individuals' privacy rights is a challenging ethical consideration.

3. **Complex and Multifaceted Nature of Mental Health:**
    - Mental health is a complex and multifaceted phenomenon, making it challenging to capture and predict using a limited set of features.
    - Different mental health disorders may manifest differently, and there is often significant variability in individual experiences.
4. **Interpersonal Variability:**
    - Personal experiences and responses to stressors vary widely among individuals. Predicting mental health issues requires accounting for this variability and considering individual differences.
5. **Stigma and Bias:**
    - The stigma associated with mental health issues can lead to underreporting or hiding of symptoms, potentially introducing bias into the training data.
    - Models may inadvertently perpetuate existing biases if not carefully developed and validated.
6. **Ethical Considerations:**
    - Developing and deploying models for mental health prediction requires careful ethical considerations, including transparency, consent, and the responsible handling of sensitive information.
    - Ensuring that the use of machine learning models aligns with ethical guidelines and standards in the field of mental health.
7. **User Acceptance and Trust:**
    - Employees may be reluctant to participate in mental health screenings or may not trust the predictive models, affecting the accuracy and effectiveness of the system.
    - Establishing trust and ensuring user acceptance is crucial for the success of any mental health prediction tool.
8. **Limited Understanding of Mental Health Factors:**
    - The understanding of factors contributing to mental health issues, especially in a workplace context, may be limited.
    - Lack of comprehensive knowledge about the relationship between work-related stressors and mental health can impact the development of accurate predictive models.
9. **Dynamic Nature of Mental Health:**

- Mental health conditions can change over time, and predicting these changes requires models that can adapt to evolving circumstances.
- Longitudinal data collection and updating models to reflect changing mental health dynamics are challenging but essential.

10. **Integration with Support Systems:**
    - Successful prediction is only part of the solution. Integrating the predictions into effective support systems and interventions is critical for the overall impact on employee well-being.

2.2 References

Research Papers and Journals:

1. **Title:** "Predicting Mental Health from Early Social Media Interactions"
    - **Authors:** Glen Coppersmith, Mark Dredze, Craig Harman
    - **Link:** Predicting Mental Health from Early Social Media Interactions
2. **Title:** "Predictive modeling of mental health recovery in schizophrenia using clinical, demographic, and cognitive variables"
    - **Authors:** Isaac M. Marks, J. J. Špaček, et al.
    - **Link:** Predictive modeling of mental health recovery in schizophrenia

Ethical Considerations:

3. **Title:** "Ethical Machine Learning in Mental Health: A Systematic Review"
    - **Authors:** Camille Nebeker, Daniel O. Gill, et al.
    - **Link:** Ethical Machine Learning in Mental Health
4. **Title:** "Ethical and Legal Considerations for Mental Health Apps"
    - **Authors:** John Torous, Hannah Wisniewski, et al.
    - **Link:** Ethical and Legal Considerations for Mental Health Apps

Data Sources:

5. **Title:** "Predictive modeling with big data: Is bigger really better?"
    - **Authors:** Russell Greiner, Mark McClearn, et al.
    - **Link:** Predictive modeling with big data
6. **Title:** "Predictive modeling opportunities in population health"
    - **Authors:** W. Nick Street, Marko A. Rodriguez, et al.

- **Link:** Predictive modeling opportunities in population health

Guidelines and Reports:

7. **Title:** "Machine Learning: The power and promise of computers that learn by example"
    - **Author:** Royal Society
    - **Link:** Machine Learning Report
8. **Title:** "Guidelines for the use of artificial intelligence in mental health care"
    - **Authors:** World Psychiatric Association (WPA) Section on Artificial Intelligence in Mental Health
    - **Link:** Guidelines for the use of AI in mental health care

Industry Reports and Case Studies:

9. **Title:** "Mental Health and Wellness in the Workplace: A New Imperative for Employers"
    - **Organization:** Deloitte
    - **Link:** Mental Health and Wellness in the Workplace
10. **Title:** "How Companies Are Using Artificial Intelligence to Foster Employee Well-Being"
    - **Organization:** Harvard Business Review
    - **Link:** AI for Employee Well-Being

2.3 Problem statement definition

Background: Mental health issues among working professionals have become a significant concern in modern workplaces. Stress, burnout, and other mental health challenges can have severe consequences for both individuals and organizations. Predictive modeling using machine learning offers a potential solution to identify and address these issues early, fostering a supportive work environment.

Problem Statement: The challenge is to develop a machine learning model that accurately predicts the likelihood of mental health illnesses among working professionals based on relevant features. This model aims to assist organizations in proactively managing employee well-being, reducing the stigma associated with mental health discussions, and providing personalized support.

Objectives:

1. **Early Detection:** Develop a model capable of early detection of potential mental health issues by analyzing diverse sets of features, such as work-related stressors, job satisfaction, and personal well-being indicators.
2. **Personalized Support:** Create a system that tailors support services and resources based on individual predictions, offering personalized interventions to employees at higher risk of mental health challenges.
3. **Privacy and Ethical Considerations:** Address privacy concerns by implementing ethical data handling practices, ensuring compliance with data protection regulations, and respecting individuals' rights to confidentiality.
4. **Integration with Existing Systems:** Ensure seamless integration of the predictive model with existing support systems and HR processes, facilitating effective interventions and resource allocation.
5. **User Acceptance and Trust:** Develop strategies to promote user acceptance and trust in the predictive model, considering the sensitive nature of mental health data and potential concerns among employees.
6. **Validation and Improvement:** Continuously validate and improve the model's accuracy and reliability by incorporating feedback from mental health professionals, employees, and stakeholders. Implement mechanisms for ongoing model refinement.
7. **Interdisciplinary Collaboration:** Facilitate collaboration between data scientists, mental health professionals, ethicists, and organizational stakeholders to ensure a comprehensive and ethical approach to mental health prediction in the workplace.

Expected Outcomes:

- A reliable machine learning model capable of accurately predicting mental health illnesses among working professionals.
- Improved organizational strategies for addressing mental health challenges, leading to a healthier and more supportive workplace culture.
- Enhanced early intervention and personalized support services, contributing to the overall well-being and satisfaction of employees.

Potential Impact: The successful implementation of this project could lead to a positive impact on the mental health and well-being of working professionals, fostering a workplace culture that prioritizes employee support and engagement. Additionally, it can contribute valuable insights to the broader field of predictive modeling in mental health and guide future research and interventions.

3.IDEATION & PROPOSED SOLUTION

3.1 Empathy Map

3.2 Brainstorming

Brainstorming Ideas and Voting

Brainstorming is a collaborative creativity technique by which efforts are made to find a conclusion for a specific problem by gathering a list of ideas spontaneously contributed by its members.

Brainstorming for predicting mental health illness of working professionals

In today's fast-paced and demanding work environments, the mental well-being of employees is crucial for both individual success and overall productivity. Leveraging the power of machine learning, we propose an innovative solution to predict and address mental health issues in working professionals. It holds immense potential to enhance workplace well-being and productivity. By leveraging the power of data and algorithms, we can identify individuals at risk, provide timely interventions, and create a more supportive work environment.

Regular Mental Health Surveys ☑

Social Network Analysis ☑

NLP on Communication channels ☑☑☑

Provide Mental Health Education and Awareness Training ☑

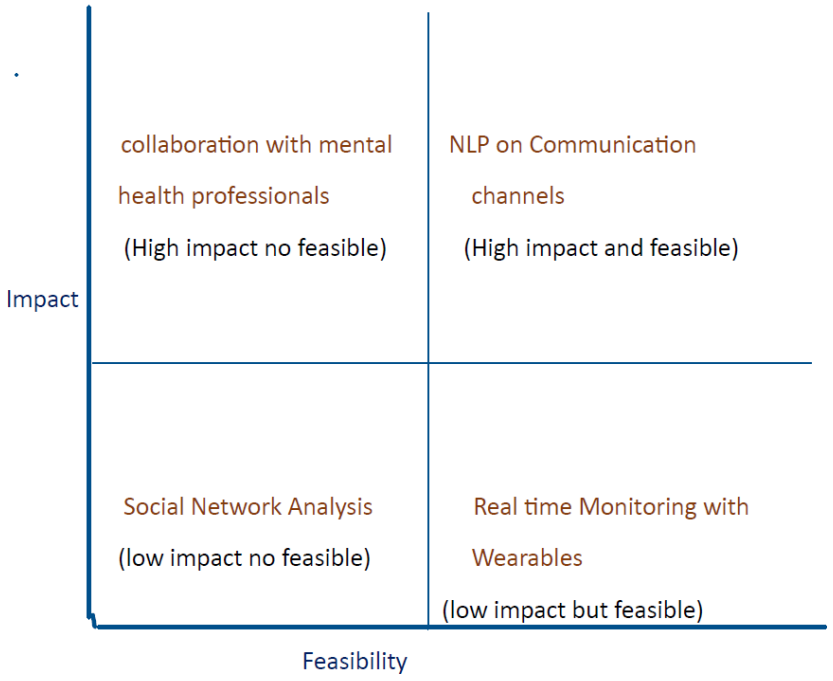
Create AI-powered chatbots for mental Health Screnning ☑☑

Collaboration with Mental Health Professionals ☑

Real-time Monitoring with Wearables ☑☑

Idea Prioritization

the process of evaluating and ranking ideas based on their potential value and feasibility, to determine which should be pursued and which should be set aside.



In conclusion, the selection of NLP on Communication Channels as our top priority is a strategic decision based on its high impact and feasibility.

4.REQUIREMENT ANALYSIS

Project Design Phase-I
Proposed Solution Template

Date	12 november 2023
Team ID	591754
Project Name	Predicting mental health illness of working professionals using machine learning
Maximum Marks	2 Marks

Proposed Solution Template:

Project team shall fill the following information in proposed solution template.

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	Mental health illness is a growing concern among working professionals. According to the World Health Organization (WHO), depression is the leading cause of disability worldwide, and it is estimated that 1 in 4 people will experience mental health illness in their lifetime. Working professionals are particularly vulnerable to mental health problems due to the pressures of work, such as long hours, demanding workloads, and job insecurity.

2.	Idea / Solution description	A predictive mental health solution for working professionals utilizes machine learning algorithms to analyze vast amounts of data and identify patterns indicative of potential mental health concerns. This solution aims to provide early intervention and support to individuals at risk, fostering a proactive approach to mental well-being in the workplace.

3.	Novelty / Uniqueness	predicting mental health illness among working professionals requires a holistic approach that considers the unique interplay of workplace dynamics, individual differences, data diversity, and ethical considerations. Building models that account for these complexities enables more accurate predictions and facilitates the development of targeted interventions to support employee well-being.
4.	Social Impact / Customer Satisfaction	While the potential for positive social impact is significant, it is crucial to manage potential risks, such as privacy concerns and ethical considerations, to ensure that predictive models are implemented responsibly and with a focus on the well-being of individuals and society as a whole.
5.	Business Model (Revenue Model)	The specific revenue model for a predictive mental health business will depend on a number of factors, including the target market, the type of data used, and the technology used to develop the models. However, all revenue models should be based on a clear understanding of the value that the business can provide to its customers.

6.	Scalability of the Solution	In order to be scalable, a predictive mental health solution must be able to handle large amounts of data and process it efficiently. Additionally, the solution must be able to be deployed to a large number of users without compromising its accuracy or performance.

5.PROJECT DESIGN

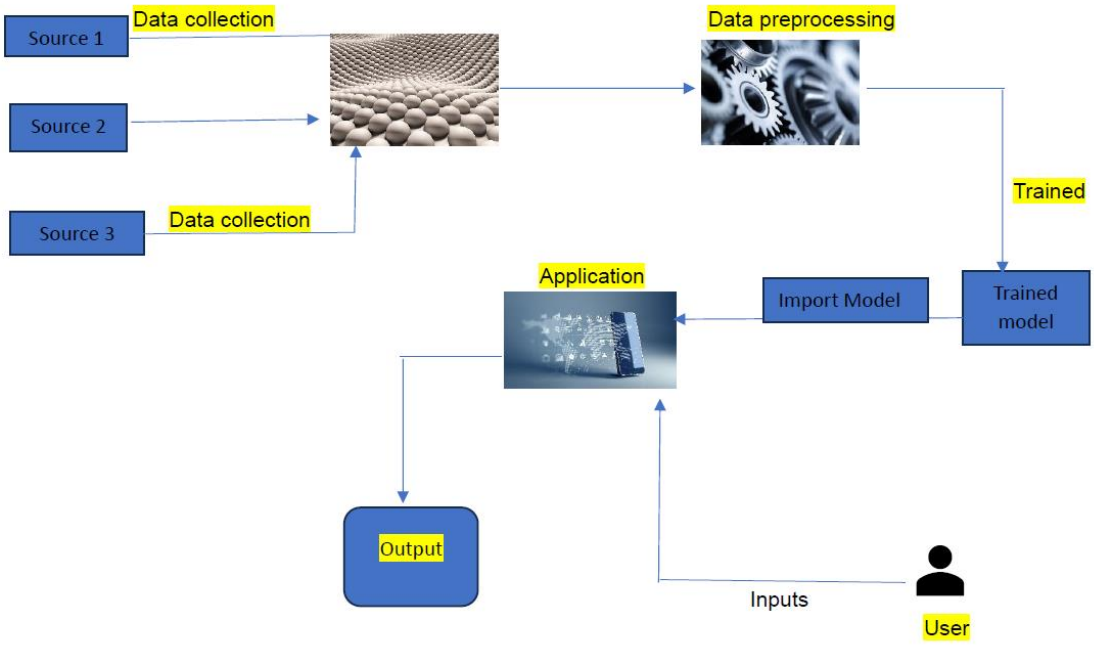
5.1 Data Flow Diagram & User Stories

Project Design Phase II
Data Flow Diagram
&
User stories

Date	17 November 2023
Team ID	591754
Project Name	Predicting Mental Health Illness of Working Professionals
Maximum Marks	4 Marks

Data Flow Diagrams:

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.



User Stories:

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Working Professionals	The system should have a user registration module with fields for personal details, work-related information, and historical mental health data.	USN-1	As a working professional, I want to register on the platform and input my data so that the system can analyze my mental health.	the registration form is accessible. All mandatory fields are validated. User data is stored securely.	High	Sprint 1
Data Scientist	The system should include a data preprocessing module to clean and format raw user data.	USN-2	As a user, I want the system to preprocess my data to ensure accurate analysis.	Raw data is successfully imported. Data cleaning processes are applied. Formatted data is stored for analysis.	medium	Sprint 2
HR	The system should implement a feature extraction process to identify key factors for mental health analysis.	USN-3	As a user, I want the system to extract relevant features for mental health prediction.	feature extraction algorithms are applied	medium	Sprint 2

Data Scientist	The system should include a machine learning model training module using historical data and extracted features.	USN-4	As a data scientist, I want the system to train a machine learning model using historical and extracted data..	Machine learning model is trained successfully.	High	Sprint 3
System Administration	The system should perform a risk priority assessment based on predicted user types and historical data.	USN-5	As a system administrator, I want the platform to assess the priority of mental health risks for intervention.	Priority levels are assigned accurately.	High	Sprint 3
Business Analyst	The system should generate notifications and reports for users based on risk priority assessments.	USN-6	As a user, I want to receive notifications and reports regarding my mental health prediction and priority level.	Reports are accessible and informative.	High	Sprint 3
Manager and Supervisiors	The system should provide managers and supervisors with access to a dashboard that flags employees who are at high risk of mental health distress.	USN-7	As a manager or supervisor, I want to be able to identify signs of mental health distress in my employees so that I can have conversations with them about their mental health and direct them to appropriate resources.	The dashboard should provide managers and supervisors with information about the signs of mental health distress, and it should also provide them with resources for talking to	High	Sprint 1

5.2 Solution Architecture

Project Design Phase-I
Solution Architecture

Date	12 November 2023
Team ID	591754
Project Name	Predicting Mental Health Illness of Working Professionals using machine learning
Maximum Marks	4 Marks

Solution Architecture:

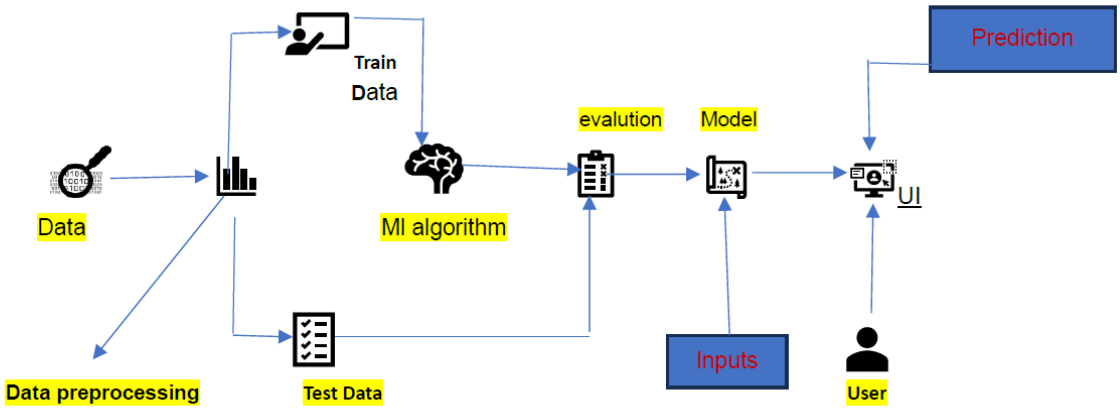
Develop a machine learning model to predict the risk of mental illness in working professionals. The model should be able to identify individuals at high risk of developing mental health problems, such as depression, anxiety, and burnout, so that they can receive timely and appropriate intervention.

A comprehensive solution architecture for predicting mental health illness of working professionals using machine learning encompasses data collection, preprocessing, model training, deployment, and monitoring. Each stage plays a crucial role in ensuring the effectiveness and ethical implementation of this technology.

- Data Collection
- Data Preprocessing
- Model Training

- Model Training
- Deployment and monitoring

Solution Architecture Diagram



6.PROJECT PLANNING & SCHEDULING

Project planning

Product Backlog, Sprint Schedule, and Estimation (4 Marks)

Sprint	User Story Number	User Story / Task	Story Points	Priority	Team Members
1	USN-1	As a data scientist, I want to collect a diverse dataset containing mental health-related features to build a prediction	2	High	jayanth

		model.			
	USN-2	As a data scientist, I want to perform exploratory data analysis (EDA) to understand the dataset's characteristics.	2	Medium	subhash
2	USN-3	As a data scientist, I want to choose appropriate machine learning algorithms and techniques for building a mental health prediction model.	1	High	krishna
	USN-4	As a data scientist, I want to evaluate the model's performance using relevant evaluation metrics.	2	Medium	shyam

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as of Planned	Sprint Release Date (Actual)

					End Date)	
Sprint-1	**3**	**2 Days**	**28 Oct 2023**	**30 Oct 2023**	**3**	**30 Oct 2023**
Sprint-2	**5**	**2 Days**	**31 Oct 2023**	**2 Nov 2023**	8	2 Nov 2023
Sprint-3	**10**	**5 Days**	**3 Nov 2023**	**8 Nov 2023**	18	8 Nov 2023
Sprint-4	**1**	**4 Days**	**9 Nov 2023**	**13 Nov 2023**	19	13 Nov 2023

Velocity (average points per sprint): 20

To calculate the team's average velocity per iteration unit (story points per day), you would need to consider the sprint duration. Assuming a 6-day sprint:
Average Velocity (AV) = Velocity (points per sprint) / Sprint Duration (days)

***AV** = sprint duration/velocity=2+2+5+4+2/5*

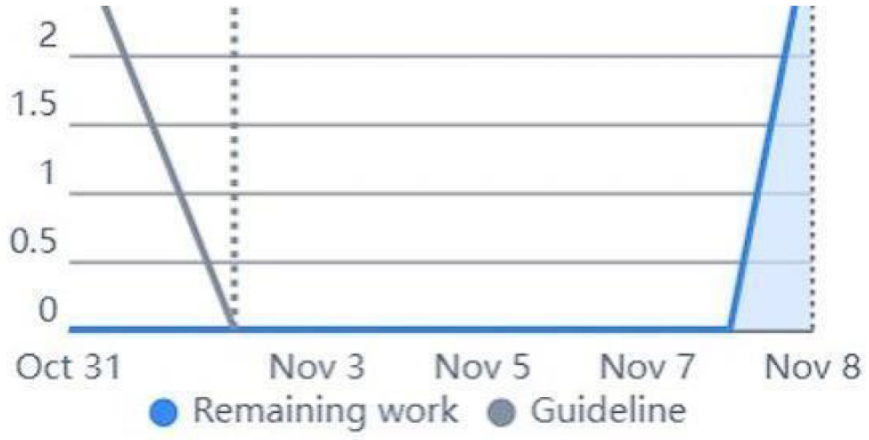
=15/5

=3

AVERAGE VELOCITY=3

Burndown Chart :

A burn down chart is a graphical representation of work left to do versus time. It is often used in agile software



6.1 Technical architecture

Technology Stack (Architecture & Stack)

Date	17 November 2023
Team ID	Team-591754
Project Name	Predicting Mental Health Illness of Working Professionals
Maximum Marks	4 Marks

Technical Architecture:

The Deliverable shall include the architectural diagram as below and the information as per the table1 & table

Guidelines:

1. **Ethical Data Handling:** Prioritize privacy and ethical considerations, obtaining informed consent and anonymizing data to protect participants.
2. **Holistic Data Collection:** Gather diverse data, including work-related and personal factors, employing surveys and objective metrics comprehensively assess contributors to mental health.
3. **Appropriate Modeling**: Choose suitable machine learning models and emphasize feature selection to ensure accurate predictions, considering decision trees, support vector machines, or neural networks.
4. **Validation and Interpretability**: Use robust validation techniques, like cross-validation, and prioritize model interpretability to ensure generalizability and actionable insights.
5. **Collaboration with Experts**: Collaborate with mental health professionals to validate the approach, refine the model, and establish responsible usage guidelines aligned with ethical considerations.

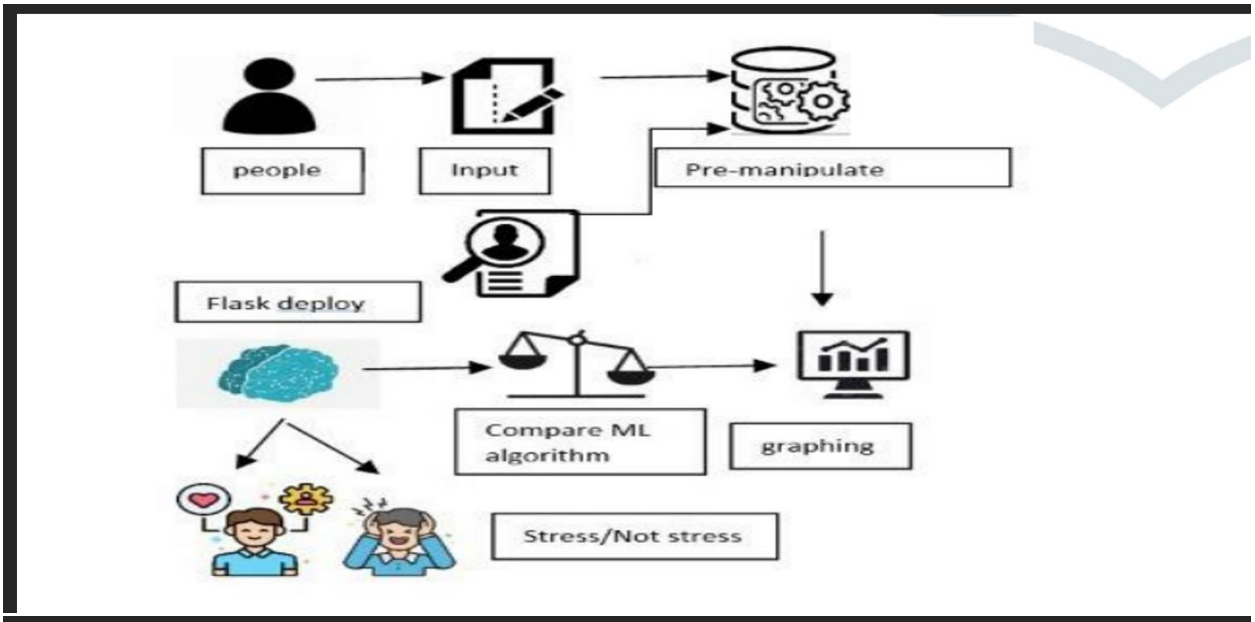


Table-1 : Components & Technologies:

S.No	Component	Description	Technology
1.	User Interface	user interacts with application with Web UI	HTML, CSS ,Flask
2.	Application Logic-1	Logic for a process in the application	Python
3.	Database	Data Type, Configurations etc.	MySQL, NoSQL, etc.
4.	File Storage	File storage requirements	Local Filesystem & Storage
5.	Machine Learning Model	Purpose of Machine Learning Model	Predictive Model, etc.
6.	Infrastructure (Server / Cloud)	Application Deployment on Local System Configuration :	Local

6.2 Sprint planning and estimation

Product Backlog, Phase Schedule, and Estimation (4 Marks):

Phase	Number	Requirement (Epic)	User Story	Story Points	Priority
Phase-2	1	AI Car Parking System	As a driver, I want to detect empty parking spots.	8	High
Phase-2	2	AI Car Parking System	As a driver, I want the AI to guide my car into a parking spot.	13	Medium
Phase-2	3	AI Car Parking System	As a parking lot owner, I want to monitor available parking spaces in real-time.	5	High
Phase-3	4	AI Car Parking System	As a driver, I want to receive notifications when a parking spot becomes available.	3	Low
Phase-1	5	AI Car Parking System	As a driver, I want a clean interface to track my movement easily without any delay.		

6.3 Sprint delivery schedule

Project Tracker, Velocity & Burndown Chart: (4 Marks)

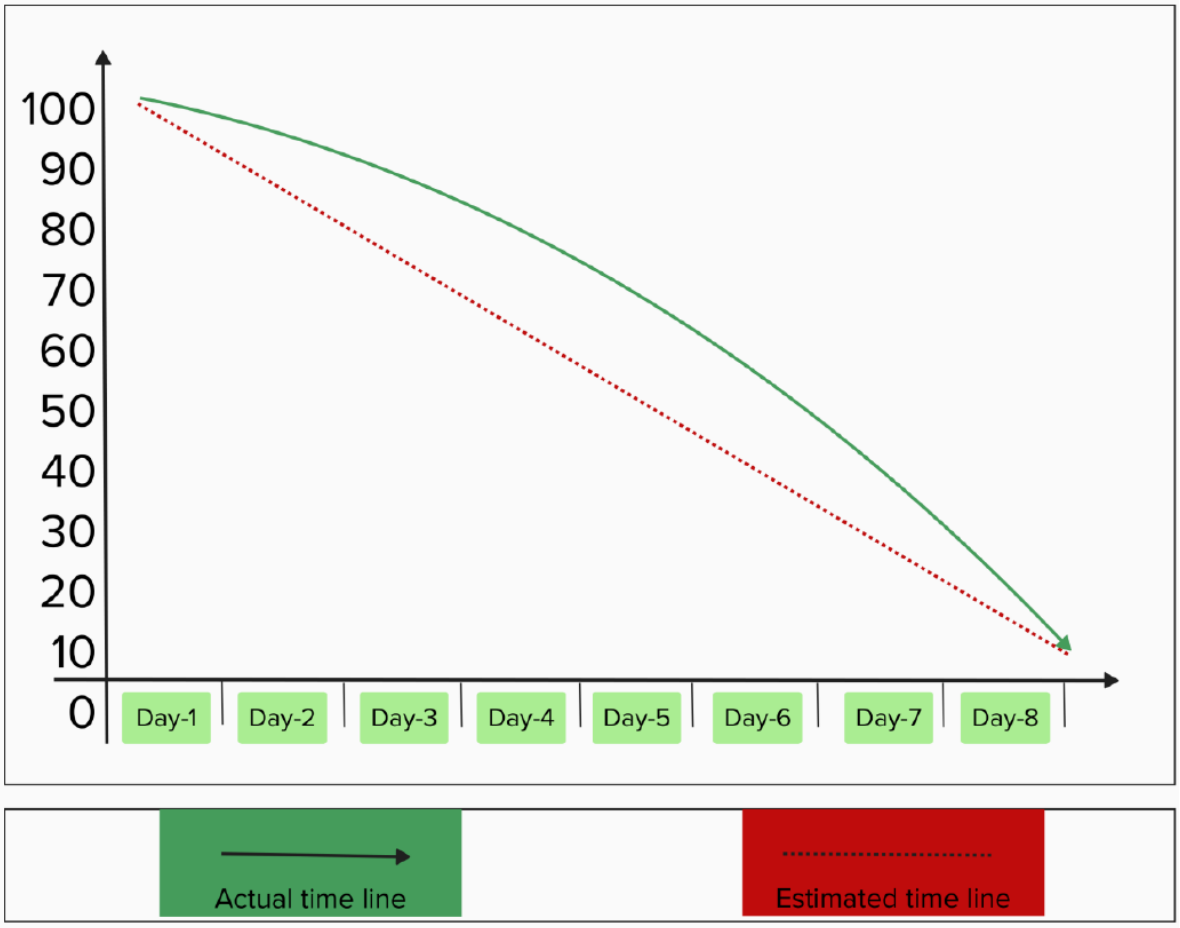
Phase	Total Story Points	Duration	Phase Start Date	Phase End Date (Planned)	Story Points Completed (as on Planned End Date)	Phase Release Date (Actual)
Phase-1	20	6 Days	07 Nov 2023	10 Nov 2023	20	11 Nov 2023
Phase-2	20	6 Days	10 Nov 2023	13 Nov 2023		
Phase-3	20	6 Days	14 Nov 2023	18 Nov 2023		
Phase-4	20	6 Days	18 Nov 2023	21 Nov 2023		

Velocity:

Imagine we have a 10-day sprint duration, and the velocity of the team is 20 (points per sprint). Let's calculate the team's average velocity (AV) per iteration unit (story points per day)

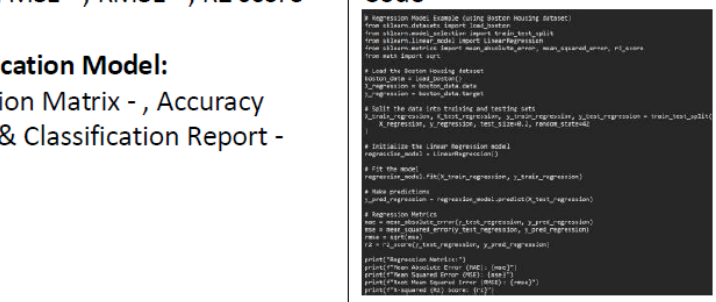
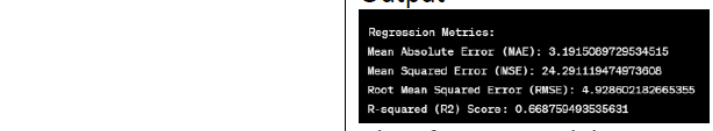
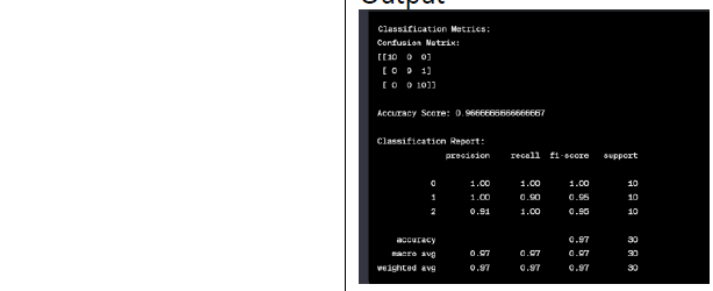
$$AV = \frac{sprint\ duration}{velocity} = \frac{20}{10} = 2$$

Burndown Chart:

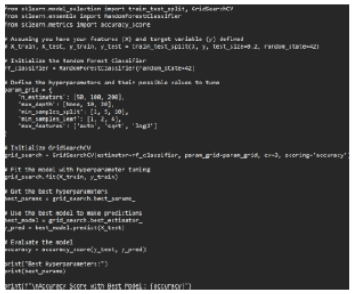


7.CODING & SOLUTIONING

7.1 Feauture 1

S.No.	Parameter	Values	Screenshot
1.	Metrics	**Regression Model:** MAE - , MSE - , RMSE - , R2 score - **Classification Model:** Confusion Matrix - , Accuracy Score- & Classification Report -	Regression model Code  Output  Classification model Code  Output 

7.2 Feauture 2

2.	Tune the Model	Hyperparameter Tuning - Validation Method -	Hyperparameter Tuning & validation method Code  Output 

8 . PERFOMANCE TESTING

8.1 performance metrics

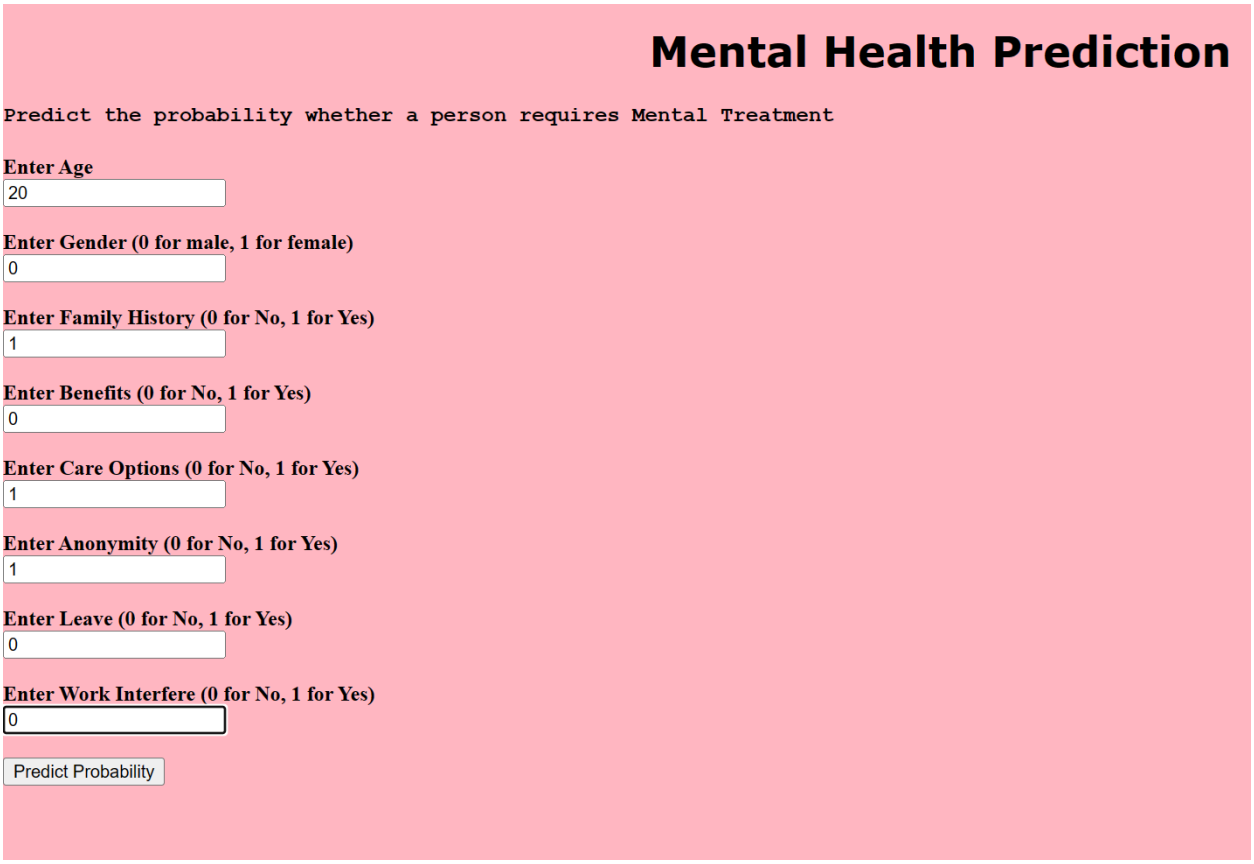
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

# Assuming y_true contains the true labels and y_pred contains the predicted labels

# Sample data (replace this with your actual data)
y_true = [1, 0, 1, 1, 0, 1, 0, 0, 1, 1]
y_pred = [1, 0, 1, 1, 0, 0, 1, 1, 1, 0]

# Accuracy
accuracy = accuracy_score(y_true, y_pred)
print(f'Accuracy: {accuracy:.4f}')

# Precision
precision = precision_score(y_true, y_pred)
print(f'Precision: {precision:.4f}')

# Recall
recall = recall_score(y_true, y_pred)
print(f'Recall: {recall:.4f}')

# F1 Score
f1 = f1_score(y_true, y_pred)
print(f'F1 Score: {f1:.4f}')

# Confusion Matrix
conf_matrix = confusion_matrix(y_true, y_pred)
print(f'Confusion Matrix:\n{conf_matrix}')
```

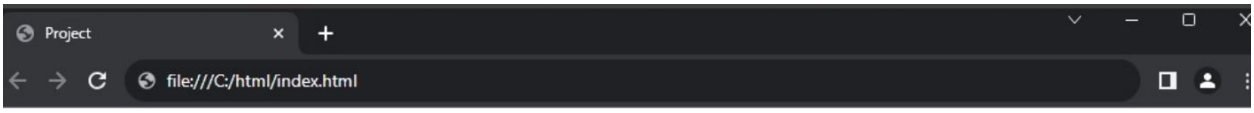
```
Accuracy: 0.8000
Precision: 0.7500
Recall: 0.8000
F1 Score: 0.7742
Confusion Matrix:
[[3 1]
 [1 5]]
```

```
[[3 1]  --> Actual negatives (0) = 3, Predicted negatives (0) = 3, Predicted positives (1) = 1
 [1 5]] --> Actual positives (1) = 1, Predicted negatives (0) = 1, Predicted positives (1) = 5
```

9.RESULTS

9.1Output screenshots





10.CONCLUSION

In conclusion, predicting mental health illness among working professionals using machine learning holds great promise, but it comes with its own set of challenges and considerations. The intersection of mental health and technology offers innovative solutions for early detection, personalized interventions, and scalable approaches. However, the ethical, privacy, and interpretability aspects must be carefully navigated to ensure responsible and effective implementation.

Machine learning models provide the advantage of objective and data-driven assessments, enabling the identification of patterns and risk factors that may not be immediately apparent through traditional methods. The ability to continuously monitor and analyze diverse data sources allows for dynamic and adaptive interventions tailored to individual needs.

Despite these advantages, it is crucial to acknowledge the potential pitfalls. Biases in both data and models, concerns about data privacy, and the ethical implications of algorithmic decision-making require meticulous attention. The interpretability of models is essential for building trust and facilitating collaboration between technology and mental health professionals.

Moreover, the dynamic nature of mental health, coupled with the subjective and complex aspects of human well-being, poses challenges for the development of accurate and reliable prediction models. The multidisciplinary collaboration of mental health experts, data scientists, ethicists, and technology developers is essential to address these challenges comprehensively.

In the journey toward predicting mental health in working professionals, a balanced approach is necessary. Recognizing the limitations and potential risks of machine learning in this context is as important as leveraging its capabilities. It is not a replacement for human expertise but a tool that, when used responsibly, can complement and enhance mental health support.

As the field continues to evolve, ongoing research, validation, and refinement of machine learning models should be prioritized. Moreover, user-centered design and a focus on human-computer interaction will be key to ensuring that these technologies are accessible, understandable, and accepted by the individuals they aim to assist.

In summary, the integration of machine learning in predicting mental health among working professionals is a complex yet promising avenue. Through careful consideration of ethical, privacy, and interpretability issues, coupled with ongoing collaboration and research, it is possible to harness the potential of technology to positively impact mental health outcomes in the workplace.

11.FUTURE SCOPE

The future scope of predicting mental health illness among working professionals using machine learning is promising, with several potential avenues for growth and development. Here are some key areas of future exploration and advancement:

1. **Integration of Multi-Modal Data:**
    - Future models could benefit from the integration of various data types, including physiological data (e.g., heart rate, sleep patterns), digital communication patterns, and electronic health records. Combining these sources may provide a more comprehensive understanding of an individual's mental health.
2. **Longitudinal Studies and Dynamic Models:**
    - Conducting longitudinal studies to track individuals over time will enhance our understanding of the dynamic nature of mental health. Developing models that can adapt to changes in an individual's mental health status over time will be crucial for more accurate predictions.
3. **Explainable AI and Model Interpretability:**
    - Improving the interpretability of machine learning models is essential for gaining the trust of both professionals and individuals. Future research could focus on developing more explainable AI models that provide clear insights into the features influencing predictions.
4. **Personalized Interventions and Support Systems:**
    - Advancements in machine learning can contribute to the development of more personalized interventions and support systems. Tailoring recommendations and interventions based on an individual's unique characteristics, preferences, and responses could enhance the effectiveness of mental health strategies.
5. **Human-Computer Interaction and User Experience:**
    - Future developments should prioritize creating user-friendly interfaces that facilitate effective human-computer interaction.

Ensuring that individuals feel comfortable and informed while interacting with mental health prediction tools is crucial for widespread acceptance and adoption.

6. **Ethical and Fair AI Practices:**
    - Continued emphasis on ethical considerations, fairness, and the reduction of biases in machine learning models is paramount. Research should focus on developing strategies to mitigate biases and ensure that predictive models are fair and equitable across diverse populations.
7. **Collaboration Between Disciplines:**
    - Encouraging collaboration between mental health professionals, data scientists, ethicists, and other relevant stakeholders is essential for the successful integration of machine learning into mental health care. Interdisciplinary teams can address complex challenges and ensure a holistic approach to well-being.
8. **Validation and Real-World Implementation:**
    - Rigorous validation studies and real-world implementation trials are necessary to assess the effectiveness and practicality of machine learning models in predicting mental health in working professionals. Continuous refinement and adaptation based on real-world feedback will be crucial for success.
9. **Public Awareness and Education:**
    - Promoting awareness and education about the capabilities and limitations of machine learning in mental health is essential. This includes educating both professionals and the general public about the role of technology in mental health care and fostering a responsible and informed approach to its use.
10. **Global Mental Health Initiatives:**
    - Considering the global nature of mental health challenges, future research and implementation efforts should address the cultural, social, and economic factors influencing mental health.

Collaborative efforts on a global scale can lead to more inclusive and effective solutions.

In summary, the future of predicting mental health illness among working professionals using machine learning involves a continuous commitment to research, innovation, and ethical practices. The evolution of technology, combined with a deep understanding of mental health dynamics, can pave the way for transformative advancements in workplace well-being.

13.APPENDEX

Source code

Library and Data Loading

In [ ]:

```
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

# 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


#Neural Network

from sklearn.neural_network import MLPClassifier

from sklearn.model_selection import RandomizedSearchCV


#Bagging

from sklearn.ensemble import BaggingClassifier, AdaBoostClassifier

from sklearn.neighbors import KNeighborsClassifier


#Naive bayes

from sklearn.naive_bayes import GaussianNB


#Stacking

from mlxtend.classifier import StackingClassifier
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/externals/six.py:31: FutureWar
ning: The module is deprecated in version 0.21 and will be removed in version
0.23 since we've dropped support for Python 2.7. Please rely on the official
version of six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", FutureWarning)
```

In [ ]:

```
from google.colab import files

uploaded = files.upload()
```

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving survey.csv to survey.csv
```

In [ ]:

```
train_df = pd.read_csv('survey.csv')

print(train_df.shape)

print(train_df.describe())

print(train_df.info())
```

```
(1259, 27)
                Age
count  1.259000e+03
mean   7.942815e+07
std    2.818299e+09
min   -1.726000e+03
25%    2.700000e+01
50%    3.100000e+01
75%    3.600000e+01
max    1.000000e+11
<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
None
```

Data Cleaning

In [ ]:

```
#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)
```

```
                           Total   Percent
comments                    1095  0.869738
state                        515  0.409055
work_interfere               264  0.209690
self_employed                 18  0.014297
benefits                       0  0.000000
Age                            0  0.000000
Gender                         0  0.000000
Country                        0  0.000000
family_history                 0  0.000000
treatment                      0  0.000000
no_employees                   0  0.000000
remote_work                    0  0.000000
tech_company                   0  0.000000
care_options                   0  0.000000
obs_consequence                0  0.000000
wellness_program               0  0.000000
seek_help                      0  0.000000
anonymity                      0  0.000000
leave                          0  0.000000
mental_health_consequence      0  0.000000
phys_health_consequence        0  0.000000
coworkers                      0  0.000000
supervisor                     0  0.000000
mental_health_interview        0  0.000000
```

```
phys_health_interview          0  0.000000
mental_vs_physical             0  0.000000
Timestamp                      0  0.000000
```

In [ ]:

```
#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)
```

Out[ ]:

	Age	Gender	Country	self_employed	family_history	treatment	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
0	37	Female	United S	NaN	No	Yes	Often	6-25	No	Yes	Yes	Not su	No	Yes	Yes	Somewha	No	No	Some of	Yes	No	Maybe	Yes	No

	Age	Gender	Country	self_employed	family_history	treatment	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
1	44	M	United States	NaN	No	No	Rarely	More than 1000	No	No	Don't know	No	Don't know	Don't know	Don't know	Don't know	Maybe	No	No	No	No	No	Don't know	No
2	32	Male	Canada	NaN	No	No	Rarely	6-25	No	Yes	No	No	No	No	Don't k	Somewh	No	No	Yes	Yes	Yes	Yes	No	No

	Age	Gender	Country	self_employed	family_history	treatment	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
3	31	Male	United Kingdom	NaN	Yes	Yes	Often	26-100	No	Yes	No	Yes	No	No	No	somewhat difficult	Yes	Yes	Some of them	No	Maybe	Maybe	No	Yes

	Age	Gender	Country	self_employed	family_history	treatment	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
4	31	Male	United States	NaN	No	No	Never	100-500	Yes	Yes	Yes	No	Don't know	Don't know	Don't know	Don't know	No	No	Some of them	Yes	Yes	Yes	Don't know	No

Cleaning NaN

In [ ]:

```
# 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()
```

Out[ ]:

	Age	Gender	Country	self_employed	family_history	treatment	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
0	37	Female	United States	NaN	No	Yes	Often	6-25	No	Yes	Yes	Not sure	No	Yes	Yes	Somewhat easy	No	No	Some of them	Yes	No	Maybe	Yes	No
1	44	M	United States	NaN	No	No	Rarely	More than 1000	No	No	Don't know	No	Don't know	Don't know	Don't know	Don't know	Maybe	No	No	No	No	No	Don't know	No

	Age	Gender	Country	self_employed	family_history	treatment	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
2	32	Male	Canada	NaN	No	No	Rarely	6-25	No	Yes	No	No	No	No	Don't know	Somewhat difficult	No	No	Yes	Yes	Yes	Yes	No	No
3	31	Male	United Ki	NaN	Yes	Yes	Often	26-100	No	Yes	No	Yes	No	No	No	Somewhat	Yes	Yes	Some of th	No	Maybe	Maybe	No	Yes

	Age	Gender	Country	self_employed	family_history	treatment	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
4	31	Male	United Kingdom	NaN	No	No	Never	100-500	Yes	Yes	Yes	No	Don't know	Don't know	Don't know	Don't know	No	No	Some of them	Yes	Yes	Yes	Don't know	No

In [ ]:

```
#Clean 'Gender'
```

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

print(gender)
```

```
['Female' 'M' 'Male' 'male' 'female' 'm' 'Male-ish' 'maile' 'Trans-female'
 'Cis Female' 'F' 'something kinda male?' 'Cis Male' 'Woman' 'f' 'Mal'
 'Male (CIS)' 'queer/she/they' 'non-binary' 'Femake' 'woman' 'Make' 'Nah'
 'All' 'Enby' 'fluid' 'Genderqueer' 'Female ' 'Androgyne' 'Agender'
 'cis-female/femme' 'Guy (-ish) ^_^' 'male leaning androgynous' 'Male '
 'Man' 'Trans woman' 'msle' 'Neuter' 'Female (trans)' 'queer'
 'Female (cis)' 'Mail' 'cis male' 'A little about you' 'Malr' 'p' 'femail'
 'Cis Man' 'ostensibly male, unsure what that really means']
```

In [ ]:

```
#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)
```

```
#Get rid of bullshit
stk_list = ['A little about you', 'p']
train_df = train_df[~train_df['Gender'].isin(stk_list)]

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

```
['female' 'male' 'trans']
```

In [ ]:

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

# Fill with media() 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 [ ]:

```
#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())
```

```
['No' 'Yes']
```

In [ ]:

```
#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())
```

```
['Often' 'Rarely' 'Never' 'Sometimes' "Don't know"]
```

Encoding Data

In [ ]:

```
#Encoding data
labelDict = {}
for feature in train_df:
    le = preprocessing.LabelEncoder()
    le.fit(train_df[feature])
    le_name_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
    train_df[feature] = le.transform(train_df[feature])
    # Get labels
    labelKey = 'label_' + feature
    labelValue = [*le_name_mapping]
    labelDict[labelKey] =labelValue

for key, value in labelDict.items():
    print(key, value)
```

```
label_Age [18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34
, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 53, 54,
55, 56, 57, 58, 60, 61, 62, 65, 72]
label_Gender ['female', 'male', 'trans']
label_Country ['Australia', 'Austria', 'Belgium', 'Bosnia and Herzegovina', '
Brazil', 'Bulgaria', 'Canada', 'China', 'Colombia', 'Costa Rica', 'Croatia',
'Czech Republic', 'Denmark', 'Finland', 'France', 'Georgia', 'Germany', 'Gree
ce', 'Hungary', 'India', 'Ireland', 'Israel', 'Italy', 'Japan', 'Latvia', 'Me
xico', 'Moldova', 'Netherlands', 'New Zealand', 'Nigeria', 'Norway', 'Philipp
ines', 'Poland', 'Portugal', 'Romania', 'Russia', 'Singapore', 'Slovenia', 'S
outh Africa', 'Spain', 'Sweden', 'Switzerland', 'Thailand', 'United Kingdom',
'United States', 'Uruguay', 'Zimbabwe']
label_self_employed ['No', 'Yes']
label_family_history ['No', 'Yes']
label_treatment ['No', 'Yes']
label_work_interfere ["Don't know", 'Never', 'Often', 'Rarely', 'Sometimes']
label_no_employees ['1-5', '100-500', '26-100', '500-1000', '6-25', 'More tha
n 1000']
label_remote_work ['No', 'Yes']
label_tech_company ['No', 'Yes']
label_benefits ["Don't know", 'No', 'Yes']
label_care_options ['No', 'Not sure', 'Yes']
label_wellness_program ["Don't know", 'No', 'Yes']
label_seek_help ["Don't know", 'No', 'Yes']
label_anonymity ["Don't know", 'No', 'Yes']
label_leave ["Don't know", 'Somewhat difficult', 'Somewhat easy', 'Very diffi
cult', 'Very easy']
label_mental_health_consequence ['Maybe', 'No', 'Yes']
label_phys_health_consequence ['Maybe', 'No', 'Yes']
label_coworkers ['No', 'Some of them', 'Yes']
label_supervisor ['No', 'Some of them', 'Yes']
label_mental_health_interview ['Maybe', 'No', 'Yes']
label_phys_health_interview ['Maybe', 'No', 'Yes']
label_mental_vs_physical ["Don't know", 'No', 'Yes']
label_obs_consequence ['No', 'Yes']
label_age_range ['0-20', '21-30', '31-65', '66-100']
```

In [ ]:

```
#Get rid of 'Country'

train_df = train_df.drop(['Country'], axis= 1)

train_df.head()
```

Out[ ]:

	Age	Gender	self_employed	family_history	treatment	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	age_range
0	19	0	0	0	1	2	4	0	1	2	1	1	2	2	2	1	1	1	2	1	0	2	0	2
1	26	1	0	0	0	3	5	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	2
2	14	1	0	0	0	3	4	0	1	1	0	1	1	0	1	1	1	2	2	2	2	1	0	2
3	13	1	0	1	1	2	2	0	1	1	2	1	1	1	1	2	2	1	0	0	0	1	1	2
4	13	1	0	0	0	1	1	1	1	2	0	0	0	0	0	1	1	1	2	2	2	0	0	2

Testing there aren't any missing data

In [ ]:

```
#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)
```

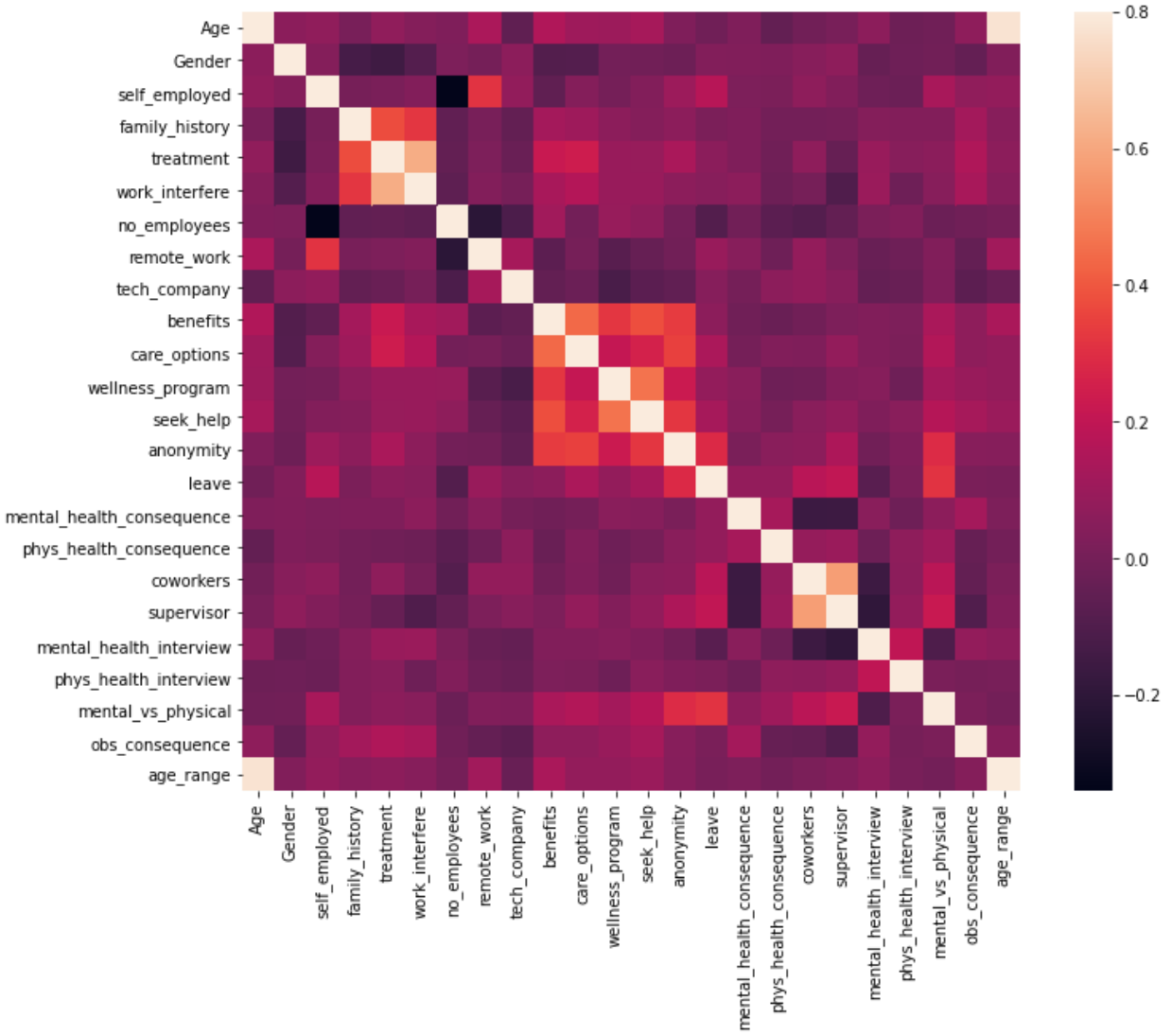
```
                           Total  Percent
age_range                      0      0.0
obs_consequence                0      0.0
Gender                         0      0.0
self_employed                  0      0.0
family_history                 0      0.0
treatment                      0      0.0
work_interfere                 0      0.0
no_employees                   0      0.0
remote_work                    0      0.0
tech_company                   0      0.0
benefits                       0      0.0
care_options                   0      0.0
wellness_program               0      0.0
seek_help                      0      0.0
anonymity                      0      0.0
leave                          0      0.0
mental_health_consequence      0      0.0
phys_health_consequence        0      0.0
coworkers                      0      0.0
supervisor                     0      0.0
mental_health_interview        0      0.0
phys_health_interview          0      0.0
mental_vs_physical             0      0.0
Age                            0      0.0
```

Features Scaling: We're going to scale age, because it is extremely different from the other ones.

Covariance Matrix. Variability comparison between categories of variables

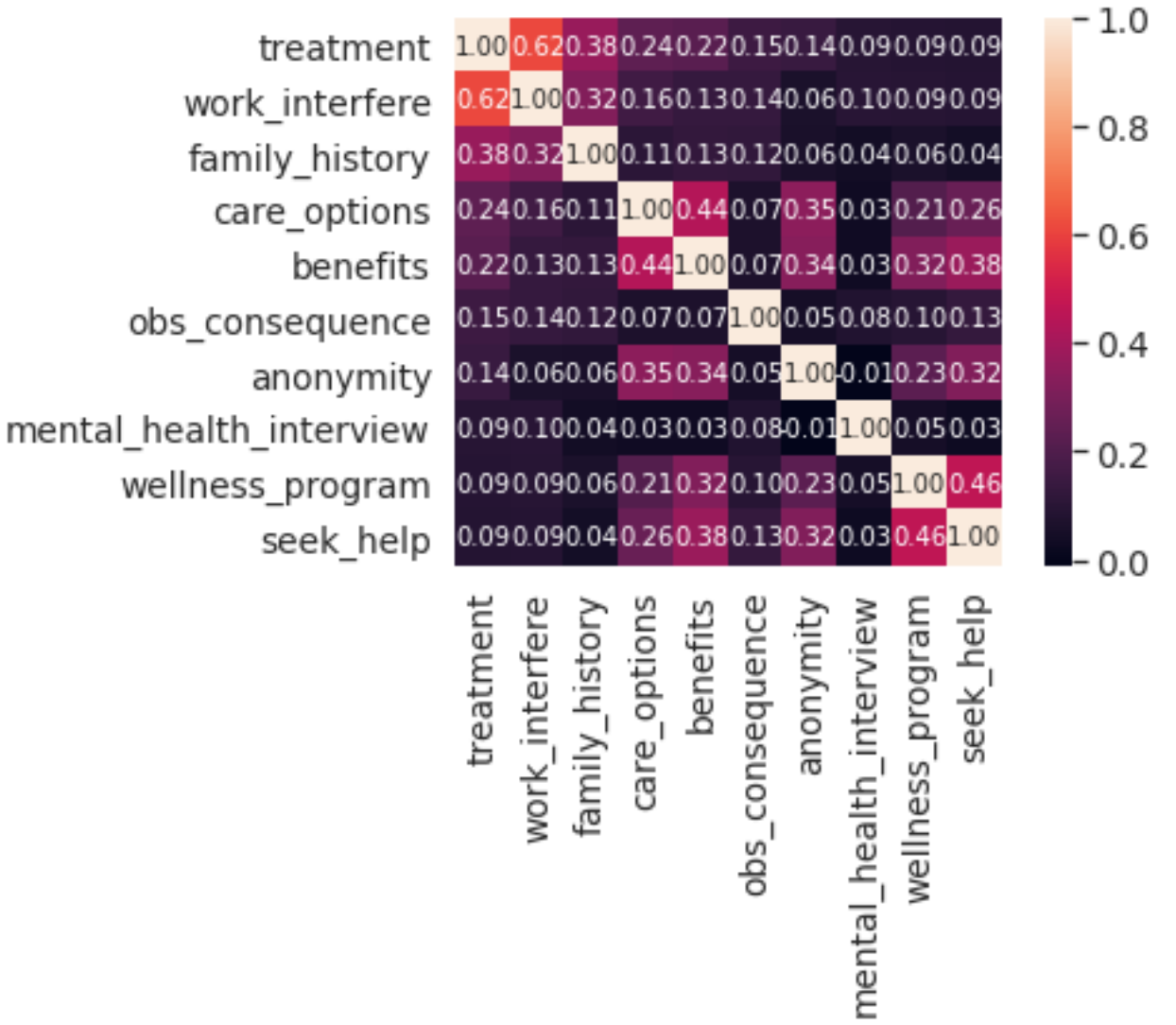
In [ ]:

```
#correlation matrix

corrmat = train_df.corr()

f, ax = plt.subplots(figsize=(12, 9))

sns.heatmap(corrmat, vmax=.8, square=True);

plt.show()
```



In [ ]:

```
#treatment correlation matrix
k = 10 #number of variables for heatmap
cols = corrmat.nlargest(k, 'treatment')['treatment'].index
cm = np.corrcoef(train_df[cols].values.T)
sns.set(font_scale=1.25)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f',
annot_kws={'size': 10}, yticklabels=cols.values, xticklabels=cols.values)
plt.show()
```



Some charts to see data relationship

Distribution and density by Age

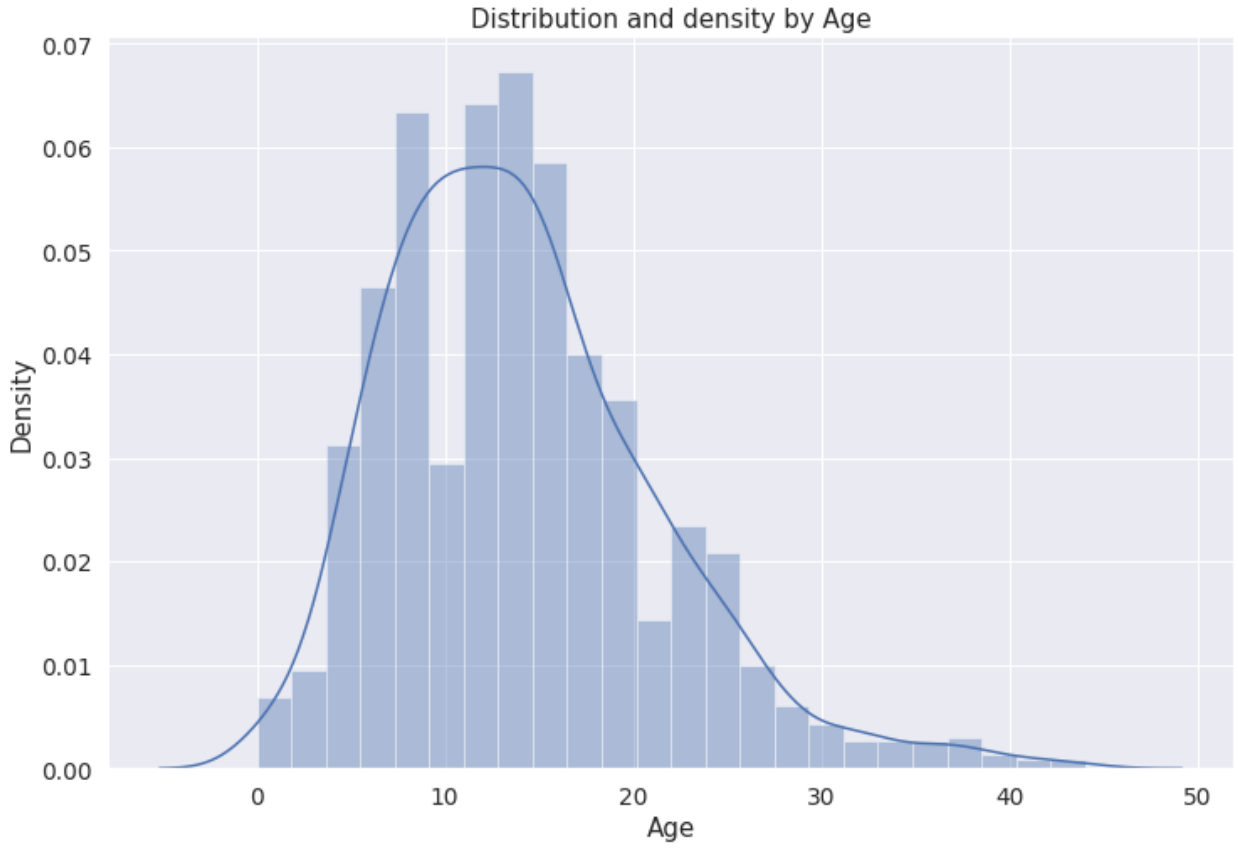
In [ ]:

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

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histog
rams).
  warnings.warn(msg, FutureWarning)
```

Out[ ]:

```
Text(0.5, 0, 'Age')
```



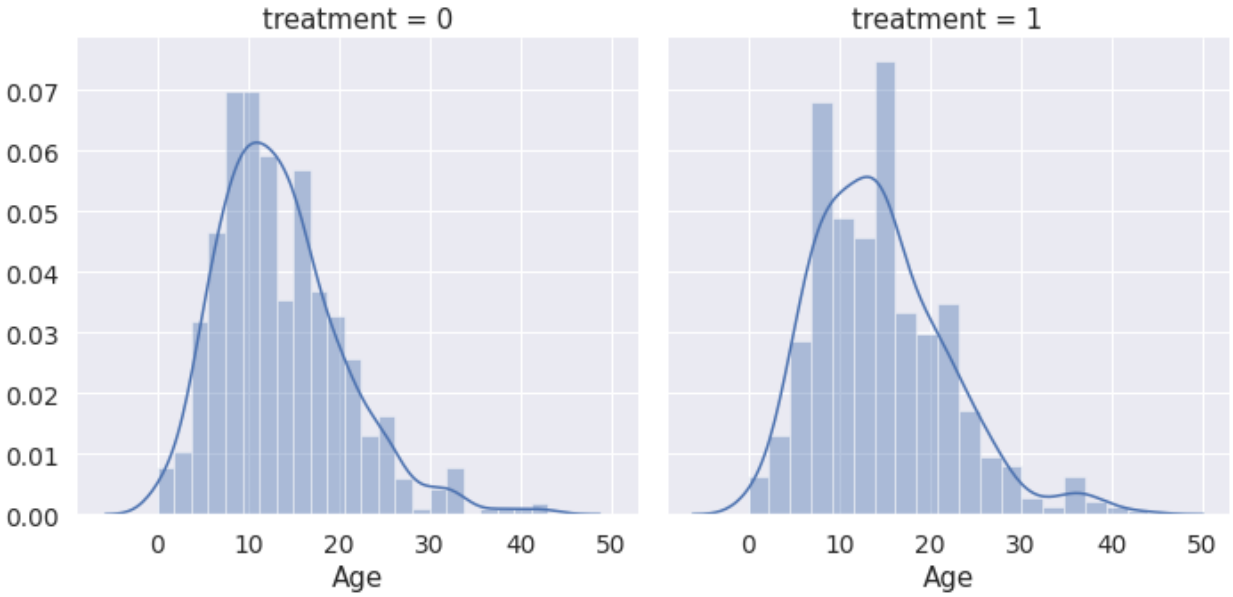
Separate by treatment

In [ ]:

```
g = sns.FacetGrid(train_df, col='treatment', size=5)
g = g.map(sns.distplot, "Age")
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/axisgrid.py:316: UserWarning:
The `size` parameter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
```

```
on with similar flexibility) or `histplot` (an axes-level function for histog
rams).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureW
arning: `distplot` is a deprecated function and will be removed in a future v
ersion. Please adapt your code to use either `displot` (a figure-level functi
on with similar flexibility) or `histplot` (an axes-level function for histog
rams).
  warnings.warn(msg, FutureWarning)
```



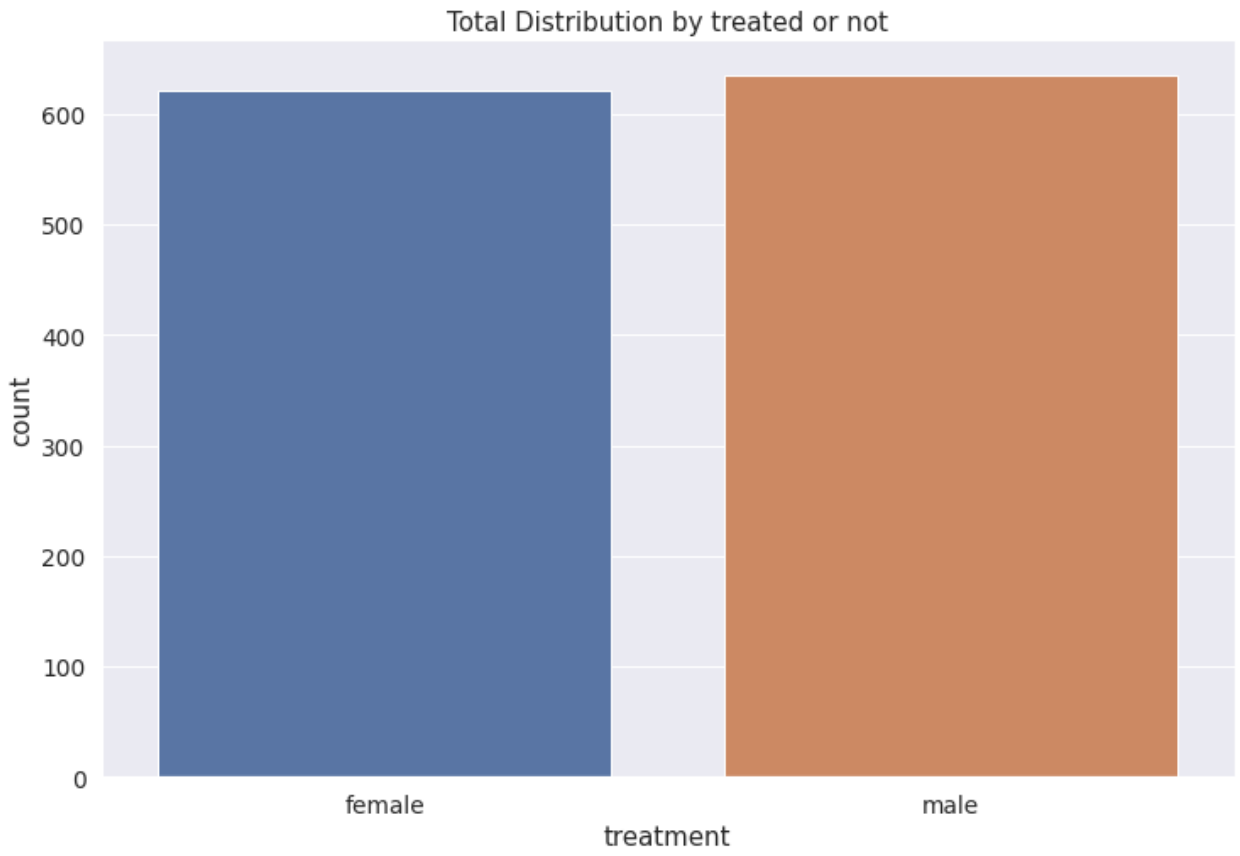
How many people has been treated?

In [ ]:

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

labels = labelDict['label_Gender']

g = sns.countplot(x="treatment", data=train_df)

g.set_xticklabels(labels)


plt.title('Total Distribution by treated or not')
```

Out[ ]:

```
Text(0.5, 1.0, 'Total Distribution by treated or not')
```



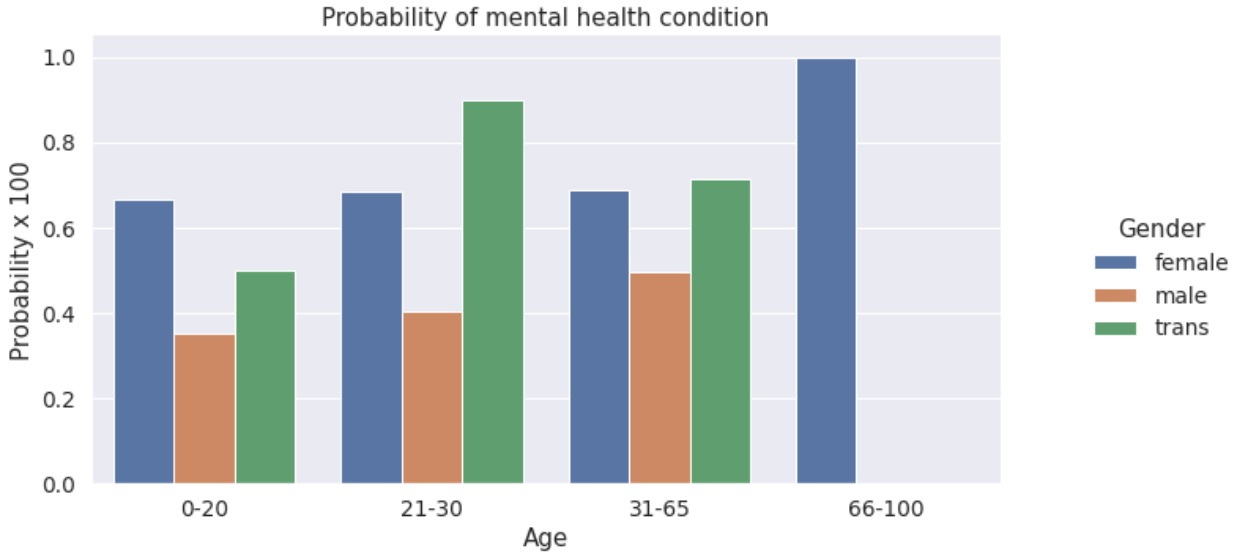
Nested barplot to show probabilities for class and sex

In [ ]:

```
o = labelDict['label_age_range']


g = sns.factorplot(x="age_range", y="treatment", hue="Gender",
data=train_df, kind="bar",  ci=None, size=5, aspect=2, legend_out = True)

g.set_xticklabels(o)


plt.title('Probability of mental health condition')

plt.ylabel('Probability x 100')

plt.xlabel('Age')

# replace legend labels


new_labels = labelDict['label_Gender']
```

```
for t, l in zip(g._legend.texts, new_labels): t.set_text(l)


# Positioning the legend
g.fig.subplots_adjust(top=0.9,right=0.8)


plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:3714: UserWarni
ng: The `factorplot` function has been renamed to `catplot`. The original nam
e will be removed in a future release. Please update your code. Note that the
default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catplot`
.
  warnings.warn(msg)
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:3720: UserWarni
ng: The `size` parameter has been renamed to `height`; please update your cod
e.
  warnings.warn(msg, UserWarning)
```



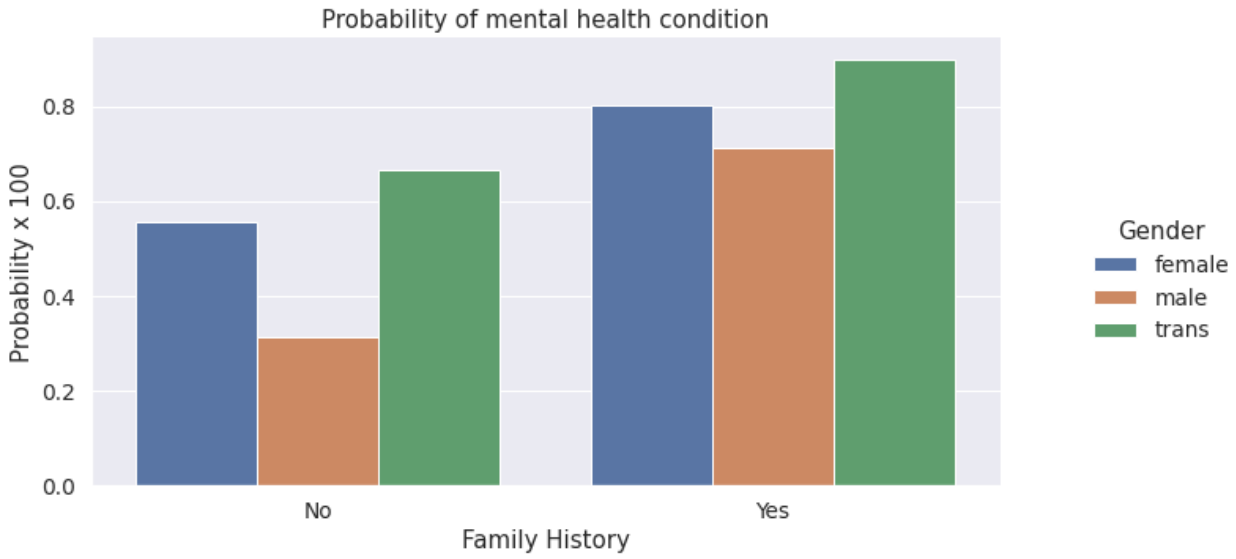
Barplot to show probabilities for family history

In [ ]:

```
o = labelDict['label_family_history']

g = sns.factorplot(x="family_history", y="treatment", hue="Gender",
data=train_df, kind="bar", ci=None, size=5, aspect=2, legend_out = True)

g.set_xticklabels(o)

plt.title('Probability of mental health condition')
```

```
plt.ylabel('Probability x 100')

plt.xlabel('Family History')


# replace legend labels

new_labels = labelDict['label_Gender']

for t, l in zip(g._legend.texts, new_labels): t.set_text(l)


# Positioning the legend

g.fig.subplots_adjust(top=0.9,right=0.8)


plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:3714: UserWarni
ng: The `factorplot` function has been renamed to `catplot`. The original nam
e will be removed in a future release. Please update your code. Note that the
default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catplot`
.
  warnings.warn(msg)
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:3720: UserWarni
ng: The `size` parameter has been renamed to `height`; please update your cod
e.
  warnings.warn(msg, UserWarning)
```

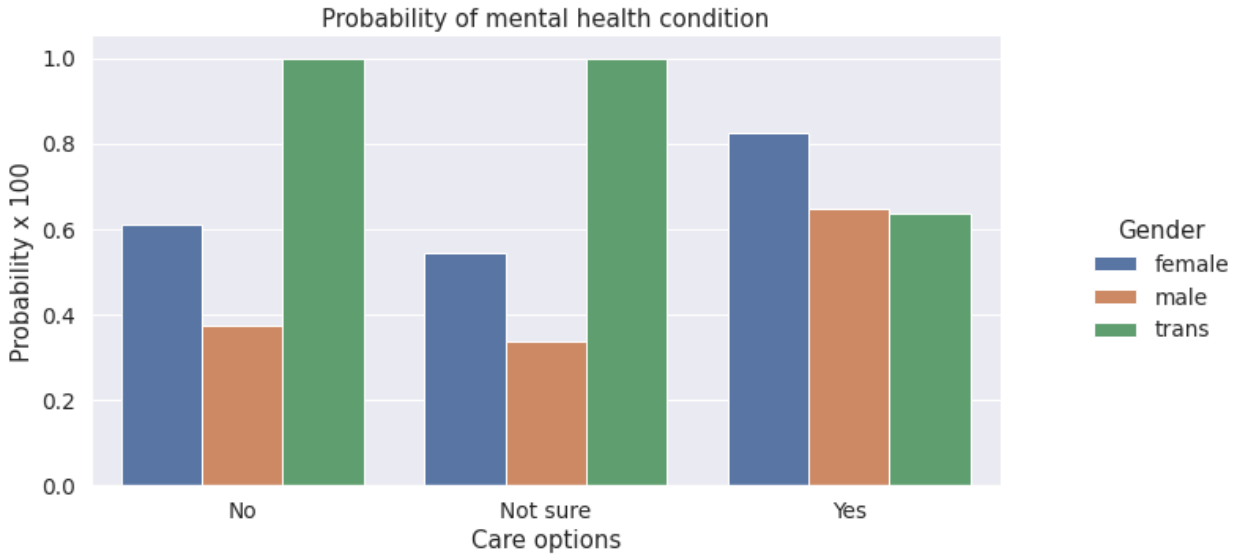


Barplot to show probabilities for care options

In [ ]:

```
o = labelDict['label_care_options']

g = sns.factorplot(x="care_options", y="treatment", hue="Gender",
data=train_df, kind="bar", ci=None, size=5, aspect=2, legend_out = True)

g.set_xticklabels(o)

plt.title('Probability of mental health condition')

plt.ylabel('Probability x 100')

plt.xlabel('Care options')


# replace legend labels

new_labels = labelDict['label_Gender']

for t, l in zip(g._legend.texts, new_labels): t.set_text(l)


# Positioning the legend

g.fig.subplots_adjust(top=0.9,right=0.8)

plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:3714: UserWarni
ng: The `factorplot` function has been renamed to `catplot`. The original nam
e will be removed in a future release. Please update your code. Note that the
default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catplot`
.
  warnings.warn(msg)
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:3720: UserWarni
ng: The `size` parameter has been renamed to `height`; please update your cod
e.
  warnings.warn(msg, UserWarning)
```



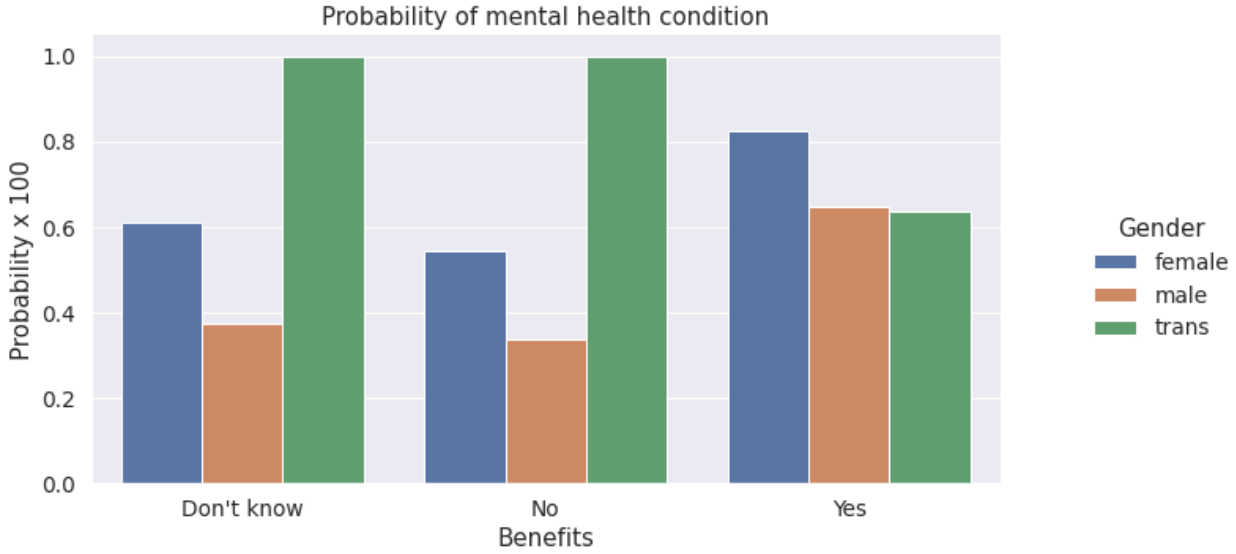
Barplot to show probabilities for benefits

In [ ]:

```
o = labelDict['label_benefits']

g = sns.factorplot(x="care_options", y="treatment", hue="Gender",
data=train_df, kind="bar", ci=None, size=5, aspect=2, legend_out = True)

g.set_xticklabels(o)

plt.title('Probability of mental health condition')

plt.ylabel('Probability x 100')

plt.xlabel('Benefits')


# replace legend labels

new_labels = labelDict['label_Gender']

for t, l in zip(g._legend.texts, new_labels): t.set_text(l)


# Positioning the legend

g.fig.subplots_adjust(top=0.9,right=0.8)

plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:3714: UserWarni
ng: The `factorplot` function has been renamed to `catplot`. The original nam
e will be removed in a future release. Please update your code. Note that the
```

```
default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catplot`
.
  warnings.warn(msg)
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:3720: UserWarni
ng: The `size` parameter has been renamed to `height`; please update your cod
e.
  warnings.warn(msg, UserWarning)
```



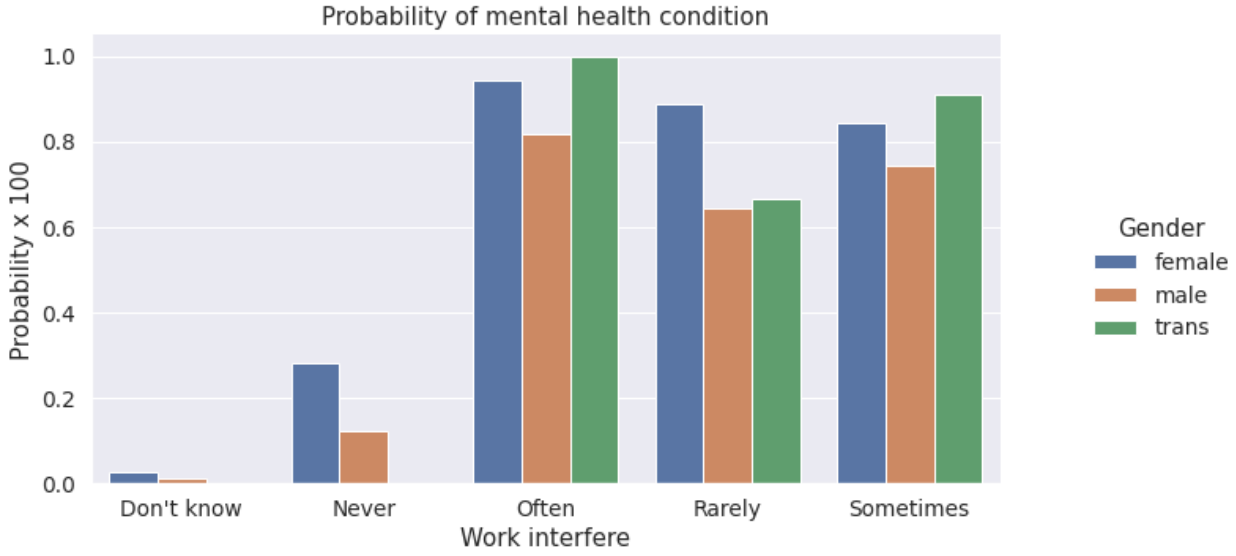
Barplot to show probabilities for work interfere

In [ ]:

```
o = labelDict['label_work_interfere']

g = sns.factorplot(x="work_interfere", y="treatment", hue="Gender",
data=train_df, kind="bar", ci=None, size=5, aspect=2, legend_out = True)

g.set_xticklabels(o)

plt.title('Probability of mental health condition')

plt.ylabel('Probability x 100')

plt.xlabel('Work interfere')


# replace legend labels

new_labels = labelDict['label_Gender']

for t, l in zip(g._legend.texts, new_labels): t.set_text(l)


# Positioning the legend

g.fig.subplots_adjust(top=0.9,right=0.8)
```

```
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:3714: UserWarni
ng: The `factorplot` function has been renamed to `catplot`. The original nam
e will be removed in a future release. Please update your code. Note that the
default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catplot`
.
  warnings.warn(msg)
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:3720: UserWarni
ng: The `size` parameter has been renamed to `height`; please update your cod
e.
  warnings.warn(msg, UserWarning)
```



Scaling and Fitting

Features Scaling We're going to scale age, because is extremely different from the othere ones.

In [ ]:

```
# Scaling Age

scaler = MinMaxScaler()

train_df['Age'] = scaler.fit_transform(train_df[['Age']])

train_df.head()
```

Out[ ]:

	Age	Gender	self_employed	family_history	treatment	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	age_range
0	0.431818	0	0	0	1	2	4	0	1	2	1	1	2	2	2	1	1	1	2	1	0	2	0	2
1	0.590909	1	0	0	0	3	5	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	2
2	0.318181	1	0	0	0	3	4	0	1	1	0	1	1	0	1	1	1	2	2	2	2	1	0	2

	Age	Gender	self_employed	family_history	treatment	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	age_range
3	0.295455	1	0	1	1	2	2	0	1	1	2	1	1	1	1	2	2	1	0	0	0	1	1	2
4	0.295455	1	0	0	0	1	1	1	1	2	0	0	0	0	0	1	1	1	2	2	2	0	0	2

Spilitting Dataset

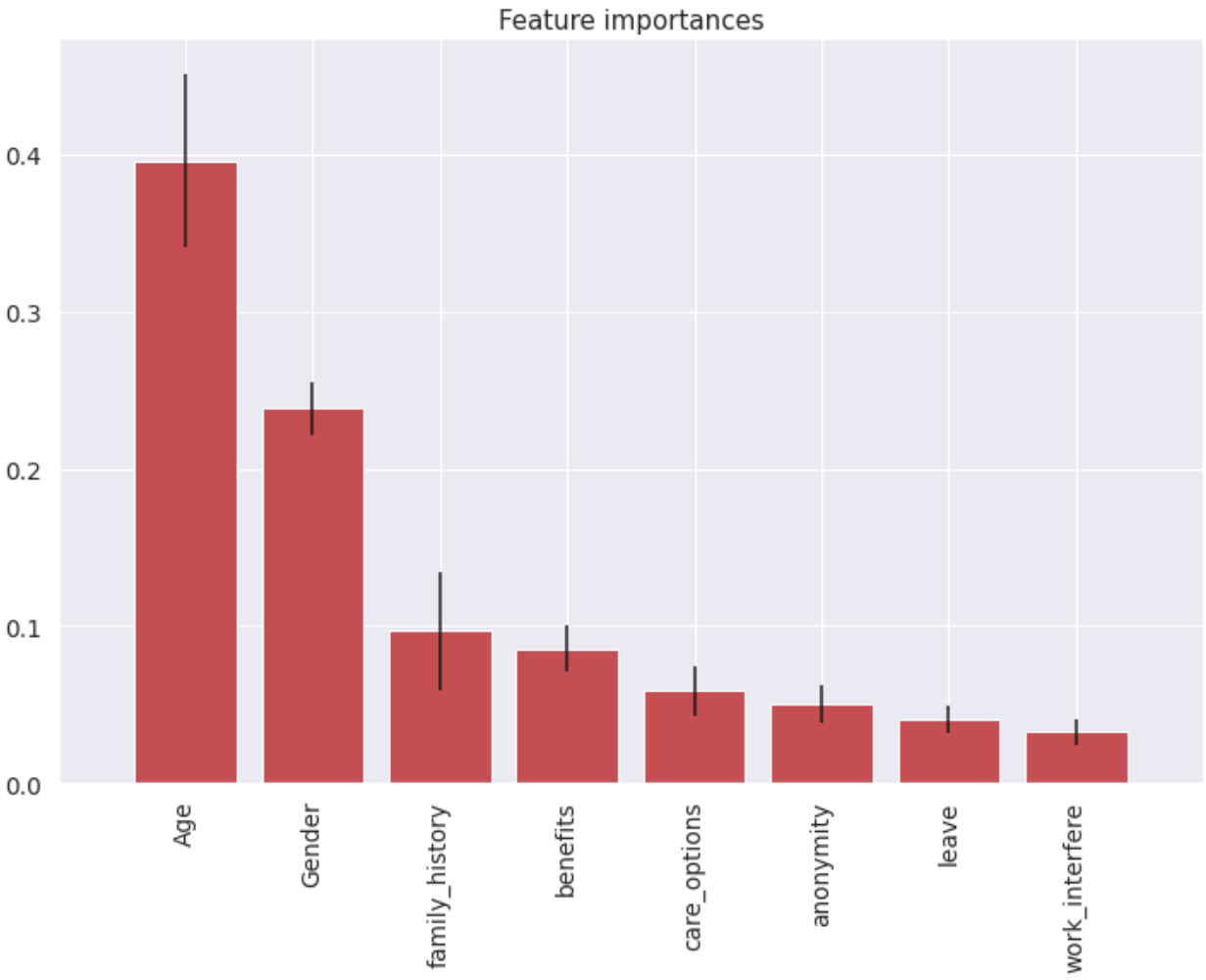
ln [ ]:

```
# 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


# 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)


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

In [ ]:

```
# 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 [ ]:

```
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))

    #Null accuracy: accuracy that could be achieved by always predicting
the most frequent class
    # examine the class distribution of the testing set (using a Pandas
Series method)
    print('Null accuracy:\n', y_test.value_counts())

    # calculate the percentage of ones
    print('Percentage of ones:', y_test.mean())

    # calculate the percentage of zeros
    print('Percentage of zeros:',1 - y_test.mean())

    #Comparing the true and predicted response values
    print('True:', y_test.values[0:25])
    print('Pred:', y_pred_class[0:25])

    #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())


    ########################################
```

```
    #Adjusting the classification threshold
    ########################################
    # print the first 10 predicted responses
    print('First 10 predicted responses:\n', model.predict(X_test)[0:10])

    # print the first 10 predicted probabilities of class membership
    print('First 10 predicted probabilities of class members:\n',
model.predict_proba(X_test)[0:10])

    # 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, 0.3)[0]

    # print the first 10 predicted probabilities
    print('First 10 predicted probabilities:\n', y_pred_prob[0:10])

    ########################################
    #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
```

Tuning with cross validation score

In [ ]:

```
def tuningCV(knn):


    # search for an optimal value of K for KNN
    k_range = list(range(1, 31))
    k_scores = []
    for k in k_range:
```

```
        knn = KNeighborsClassifier(n_neighbors=k)
        scores = cross_val_score(knn, X, y, cv=10, scoring='accuracy')
        k_scores.append(scores.mean())
    print(k_scores)
    # plot the value of K for KNN (x-axis) versus the cross-validated
accuracy (y-axis)
    plt.plot(k_range, k_scores)
    plt.xlabel('Value of K for KNN')
    plt.ylabel('Cross-Validated Accuracy')
    plt.show()
```

Tuning with GridSearchCV

In [ ]:

```
def tuningGridSerach(knn):
    #More efficient parameter tuning using GridSearchCV
    k_range = list(range(1, 31))
    print(k_range)

    # create a parameter grid: map the parameter names to the values that
should be searched
    param_grid = dict(n_neighbors=k_range)
    print(param_grid)

    # instantiate the grid
    grid = GridSearchCV(knn, param_grid, cv=10, scoring='accuracy')

    # fit the grid with data
    grid.fit(X, y)

    # view the complete results (list of named tuples)
    grid.grid_scores_
```

```
    # examine the first tuple
    print(grid.grid_scores_[0].parameters)
    print(grid.grid_scores_[0].cv_validation_scores)
    print(grid.grid_scores_[0].mean_validation_score)

    # create a list of the mean scores only
    grid_mean_scores = [result.mean_validation_score for result in
grid.grid_scores_]
    print(grid_mean_scores)

    # plot the results
    plt.plot(k_range, grid_mean_scores)
    plt.xlabel('Value of K for KNN')
    plt.ylabel('Cross-Validated Accuracy')
    plt.show()

    # examine the best model
    print('GridSearch best score', grid.best_score_)
    print('GridSearch best params', grid.best_params_)
    print('GridSearch best estimator', grid.best_estimator_)
```

Tuning with RandomizedSearchCV

In [ ]:

```
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)
```

Tuning with searching multiple parameters simultaneously

In [ ]:

```
def tuningMultParam(knn):

    #Searching multiple parameters simultaneously
    # define the parameter values that should be searched
    k_range = list(range(1, 31))
    weight_options = ['uniform', 'distance']

    # create a parameter grid: map the parameter names to the values that
should be searched
    param_grid = dict(n_neighbors=k_range, weights=weight_options)
    print(param_grid)

    # instantiate and fit the grid
    grid = GridSearchCV(knn, param_grid, cv=10, scoring='accuracy')
```

```
    grid.fit(X, y)

    # view the complete results
    print(grid.grid_scores_)

    # examine the best model
    print('Multiparam. Best Score: ', grid.best_score_)
    print('Multiparam. Best Params: ', grid.best_params_)
```

Evaluating models

Logistic Regression

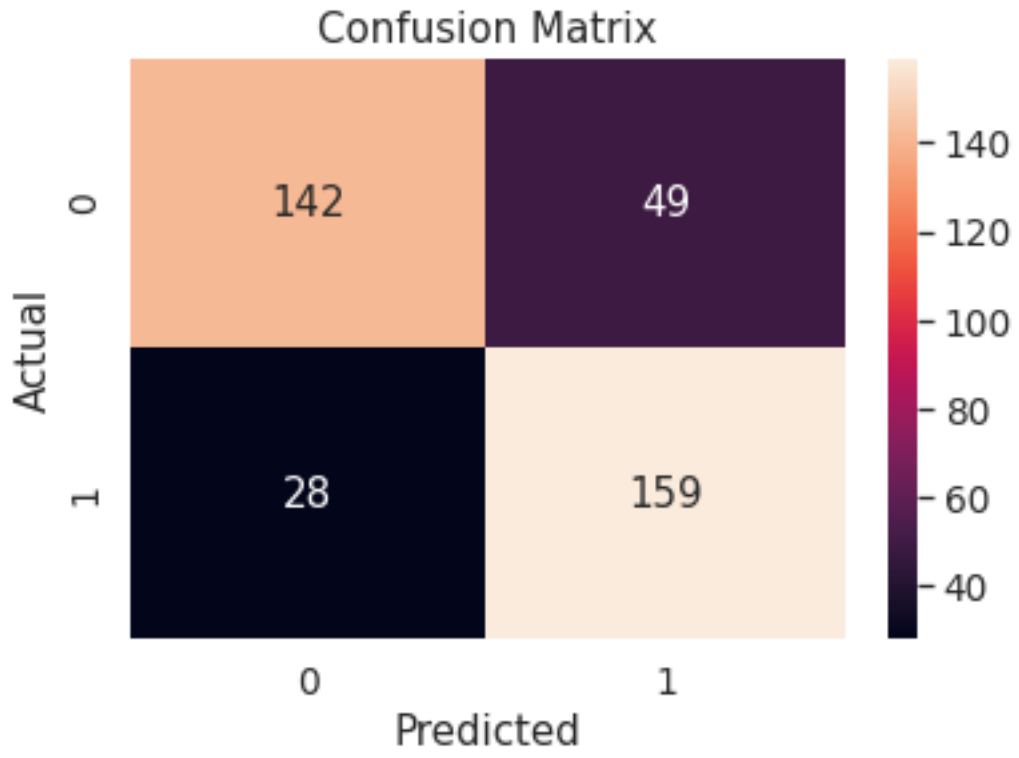
In [ ]:

```
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['Log. Regression'] = accuracy_score * 100
```

In [ ]:

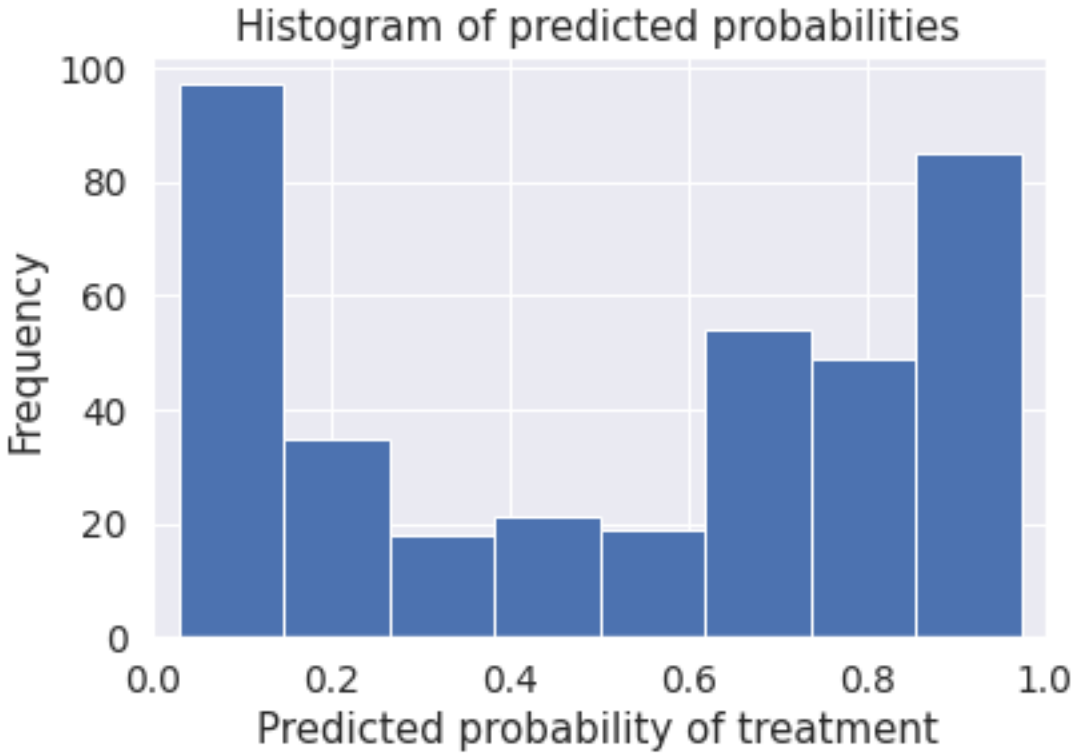
```
logisticRegression()
```

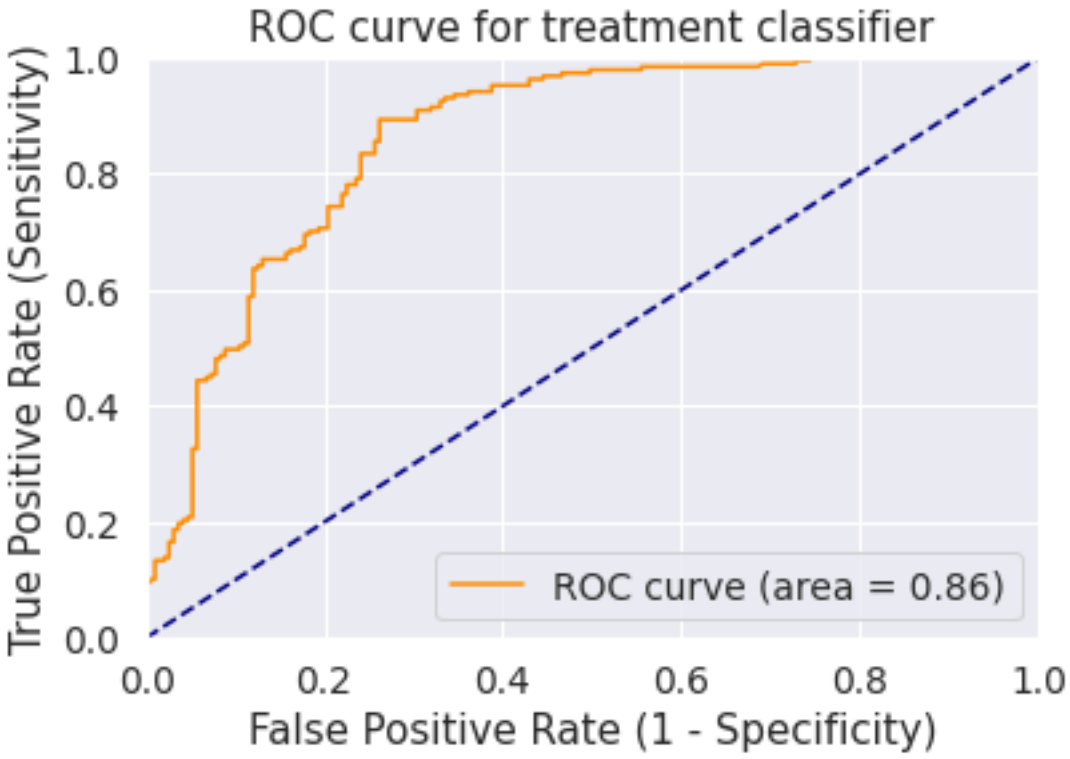
```
Accuracy: 0.7962962962962963
Null accuracy:
 0    191
1    187
Name: treatment, dtype: int64
Percentage of ones: 0.4947089947089947
Percentage of zeros: 0.5052910052910053
True: [0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1 0 0]
Pred: [1 0 0 0 1 1 0 1 0 1 0 1 1 0 1 1 1 1 0 0 0 0 1 0 0]
```




```
Classification Accuracy: 0.7962962962962963
Classification Error: 0.20370370370370372
False Positive Rate: 0.25654450261780104
Precision: 0.7644230769230769
AUC Score: 0.7968614385306716
Cross-validated AUC: 0.8753623882722146
First 10 predicted responses:
 [1 0 0 0 1 1 0 1 0 1]
First 10 predicted probabilities of class members:
 [[0.09193053 0.90806947]
 [0.95991564 0.04008436]
 [0.96547467 0.03452533]
 [0.78757121 0.21242879]
 [0.38959922 0.61040078]
 [0.05264207 0.94735793]
 [0.75035574 0.24964426]
 [0.19065116 0.80934884]
 [0.61612081 0.38387919]
 [0.47699963 0.52300037]]
```

```
First 10 predicted probabilities:
 [[0.90806947]
 [0.04008436]
 [0.03452533]
 [0.21242879]
 [0.61040078]
 [0.94735793]
 [0.24964426]
 [0.80934884]
 [0.38387919]
 [0.52300037]]
```






```
[[142  49]
 [ 28 159]]
```

KNeighbors Classifier

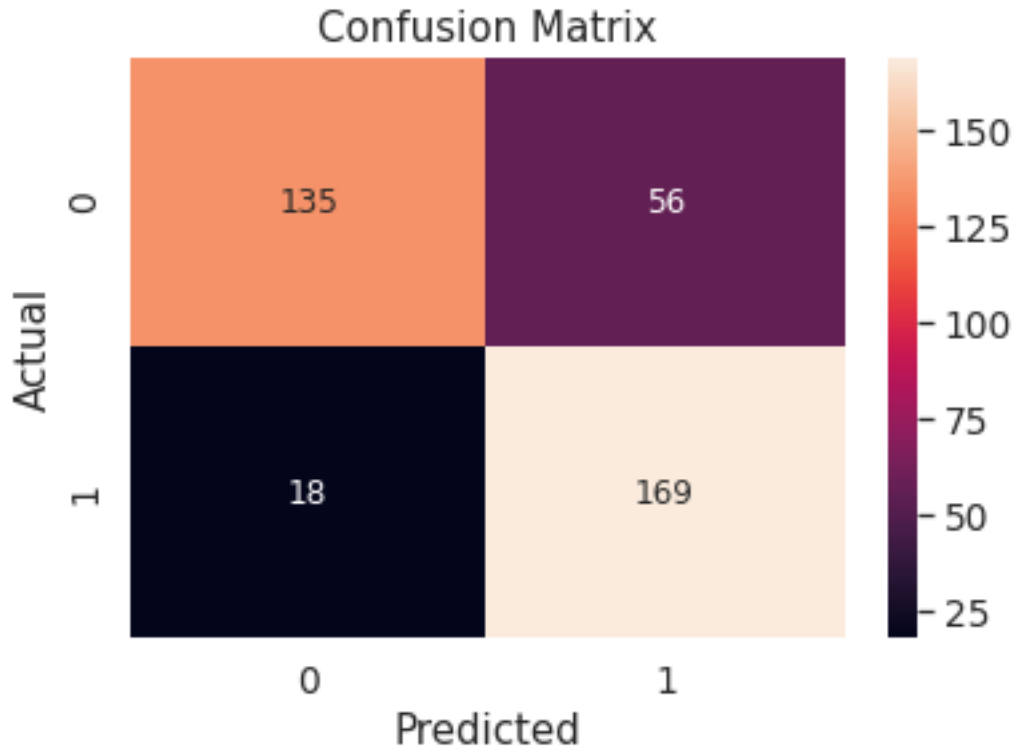
In [ ]:

```
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
```

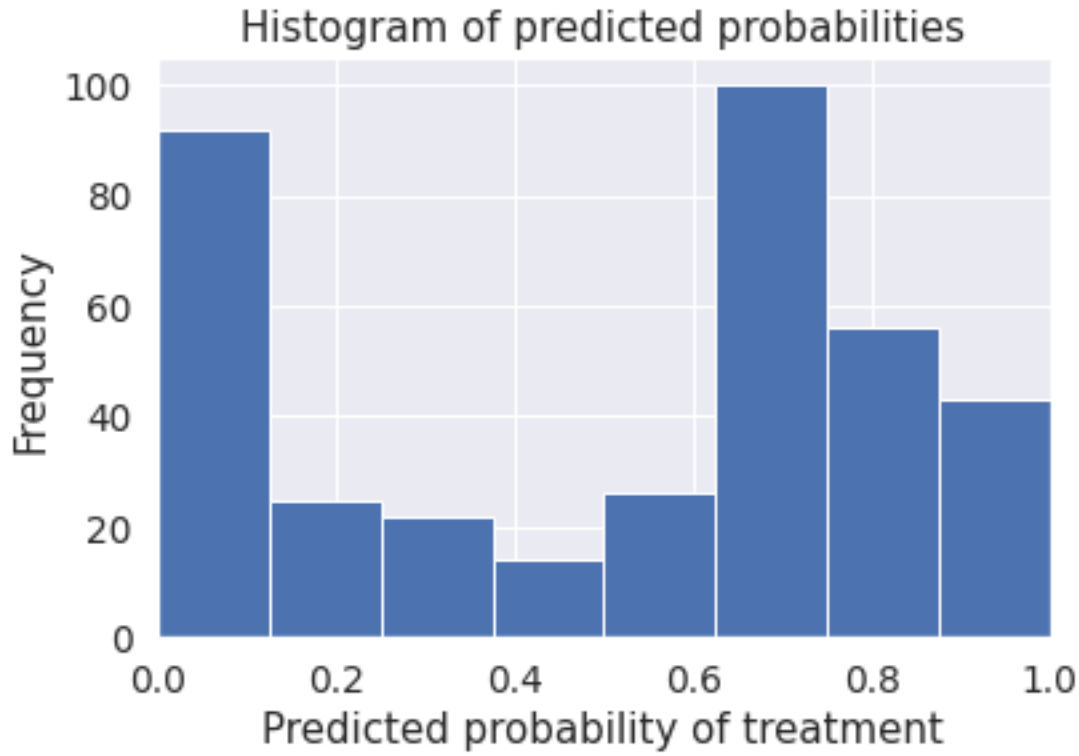
In [ ]:

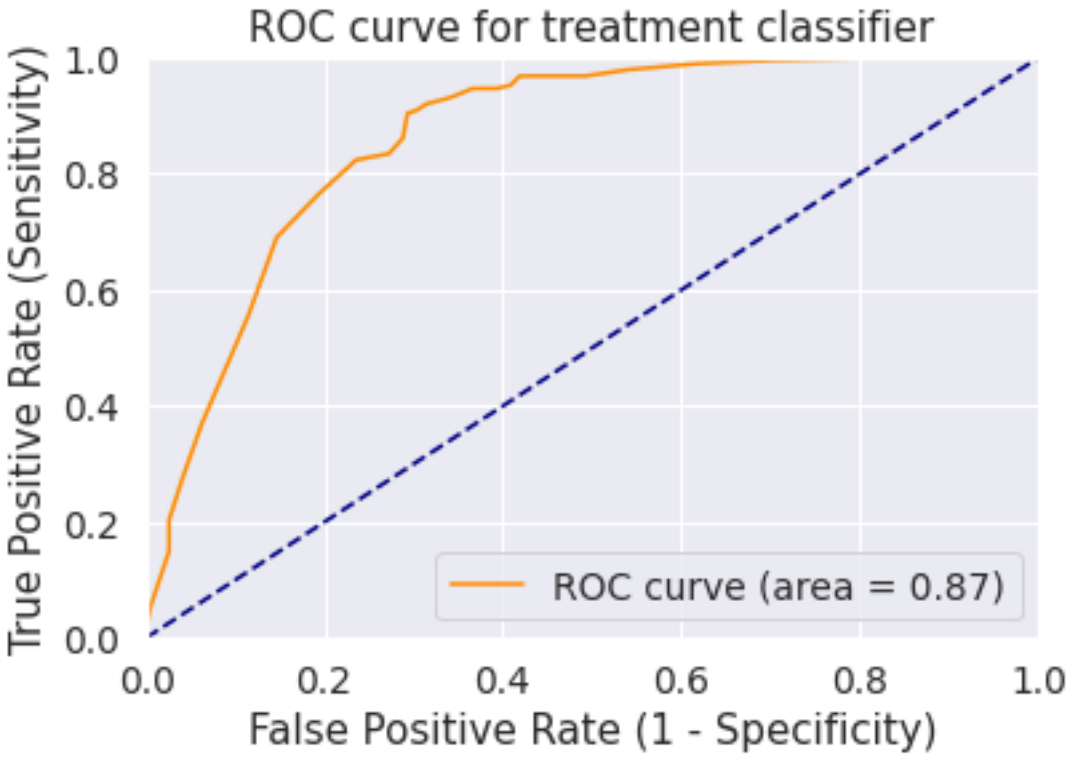
```
Knn()


Rand. Best Score:  0.8209714285714286
Rand. Best Params:  {'weights': 'uniform', 'n_neighbors': 27}
[0.816, 0.812, 0.821, 0.823, 0.823, 0.818, 0.821, 0.821, 0.815, 0.812, 0.819,
0.811, 0.819, 0.818, 0.82, 0.815, 0.803, 0.821, 0.823, 0.815]
Accuracy: 0.8042328042328042
Null accuracy:
 0    191
1    187
Name: treatment, dtype: int64
Percentage of ones: 0.4947089947089947
Percentage of zeros: 0.5052910052910053
True: [0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1 0 0]
Pred: [1 0 0 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 0 0 0 0 1 0 0]
```




```
Classification Accuracy: 0.8042328042328042
Classification Error: 0.1957671957671958
False Positive Rate: 0.2931937172774869
Precision: 0.7511111111111111
AUC Score: 0.8052747991152673
Cross-validated AUC: 0.8782819116296456
First 10 predicted responses:
 [1 0 0 0 1 1 0 1 1 1]
First 10 predicted probabilities of class members:
 [[0.33333333 0.66666667]
 [1.         0.        ]
 [1.         0.        ]
 [0.66666667 0.33333333]
 [0.37037037 0.62962963]
 [0.03703704 0.96296296]
 [0.59259259 0.40740741]
 [0.37037037 0.62962963]
 [0.33333333 0.66666667]
 [0.33333333 0.66666667]]
First 10 predicted probabilities:
 [[0.66666667]
 [0.        ]
 [0.        ]
 [0.33333333]
 [0.62962963]
 [0.96296296]
 [0.40740741]
 [0.62962963]
```

```
[0.66666667]
[0.66666667]]
```






```
[[135  56]
```

```
 [ 18 169]]
```

Decision Tree classifier

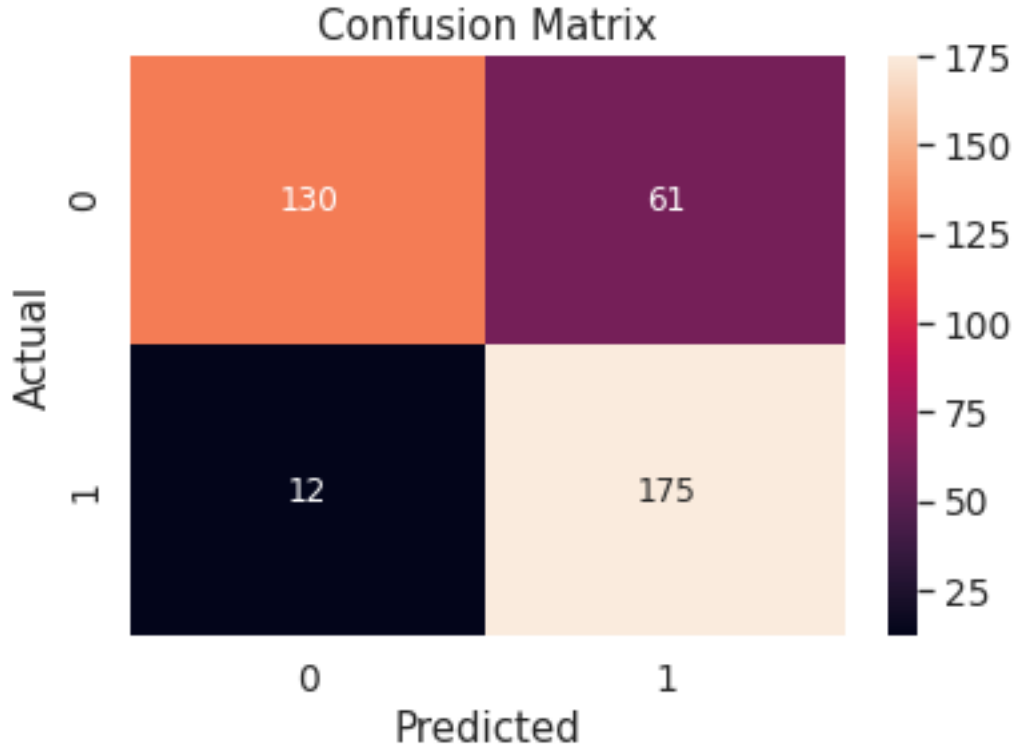
In [ ]:

```
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 [ ]:

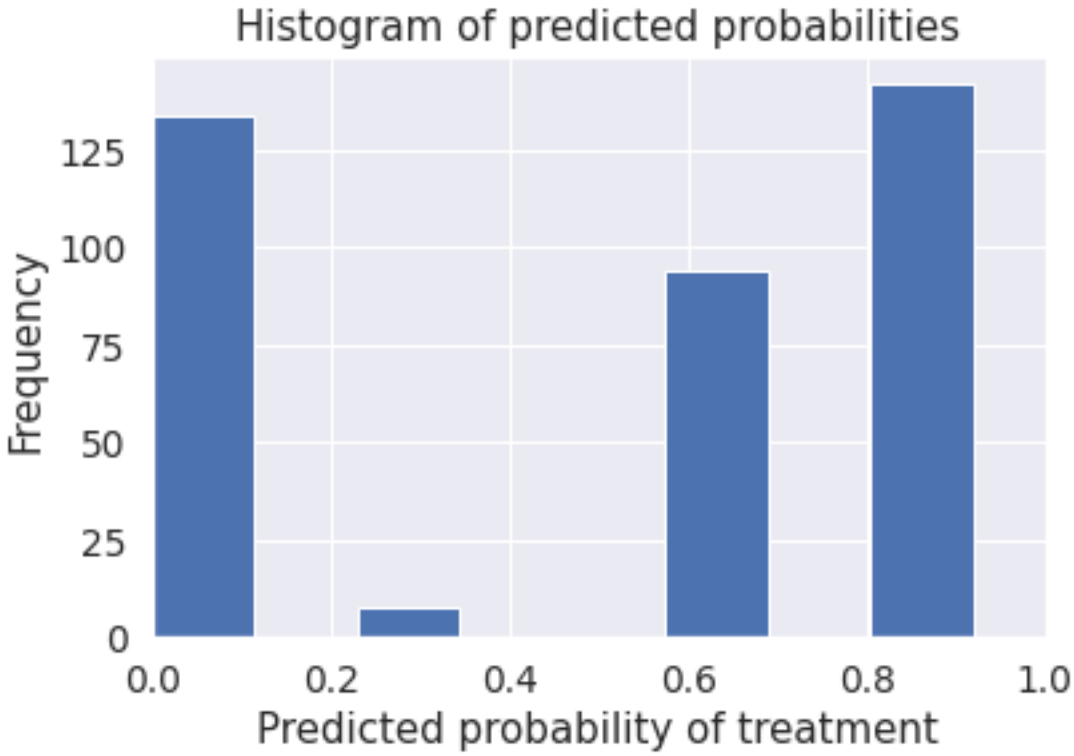
```
treeClassifier()
```

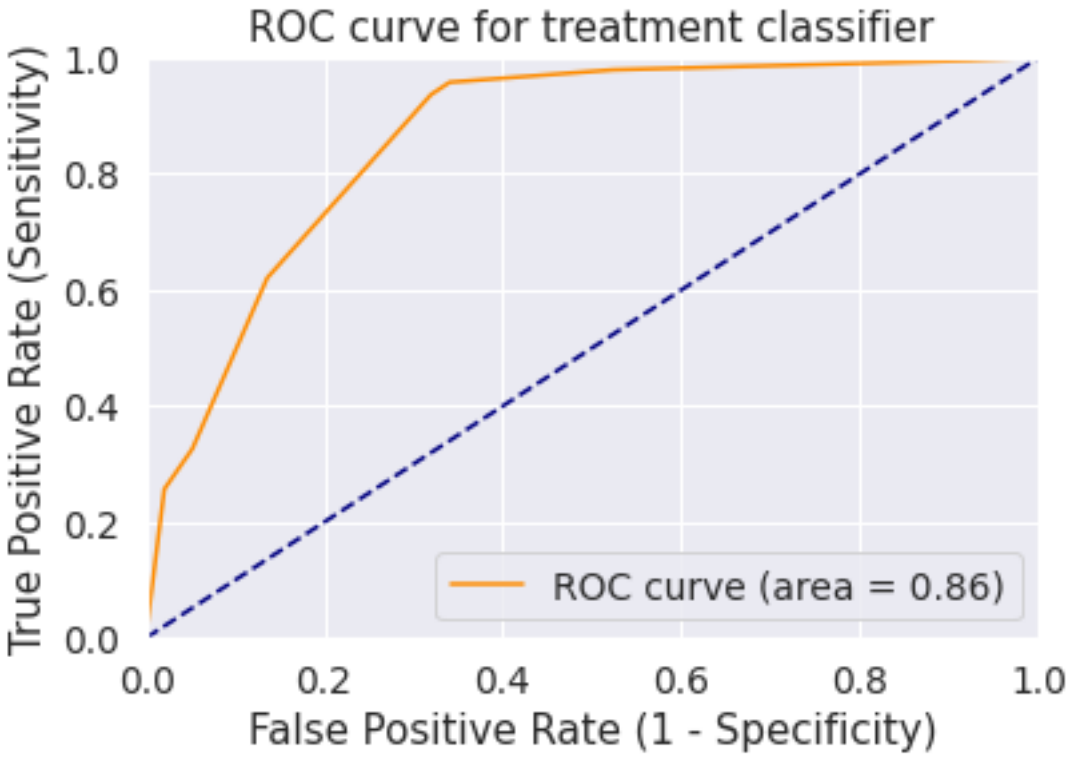
```
Rand. Best Score:  0.8305206349206349
Rand. Best Params:  {'criterion': 'entropy', 'max_depth': 3, 'max_features':
6, 'min_samples_leaf': 7, 'min_samples_split': 8}
[0.83, 0.827, 0.831, 0.829, 0.831, 0.83, 0.783, 0.831, 0.821, 0.831, 0.831, 0
.831, 0.8, 0.79, 0.831, 0.831, 0.831, 0.829, 0.831, 0.831]
Accuracy: 0.8068783068783069
Null accuracy:
 0    191
1    187
Name: treatment, dtype: int64
Percentage of ones: 0.4947089947089947
Percentage of zeros: 0.5052910052910053
True: [0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1 0 0]
Pred: [1 0 0 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 0 0 0 0 1 0 0]
```




```
Classification Accuracy: 0.8068783068783069
Classification Error: 0.19312169312169314
False Positive Rate: 0.3193717277486911
Precision: 0.7415254237288136
AUC Score: 0.8082285746283282
Cross-validated AUC: 0.8818789291403538
First 10 predicted responses:
 [1 0 0 0 1 1 0 1 1 1]
First 10 predicted probabilities of class members:
 [[0.18       0.82      ]
 [0.96534653 0.03465347]
 [0.96534653 0.03465347]
 [0.89473684 0.10526316]
 [0.36097561 0.63902439]
```

```
 [0.18       0.82      ]
 [0.89473684 0.10526316]
 [0.11320755 0.88679245]
 [0.36097561 0.63902439]
 [0.36097561 0.63902439]]
First 10 predicted probabilities:
 [[0.82      ]
 [0.03465347]
 [0.03465347]
 [0.10526316]
 [0.63902439]
 [0.82      ]
 [0.10526316]
 [0.88679245]
 [0.63902439]
 [0.63902439]]
```






```
[[130  61]
 [ 12 175]]
```

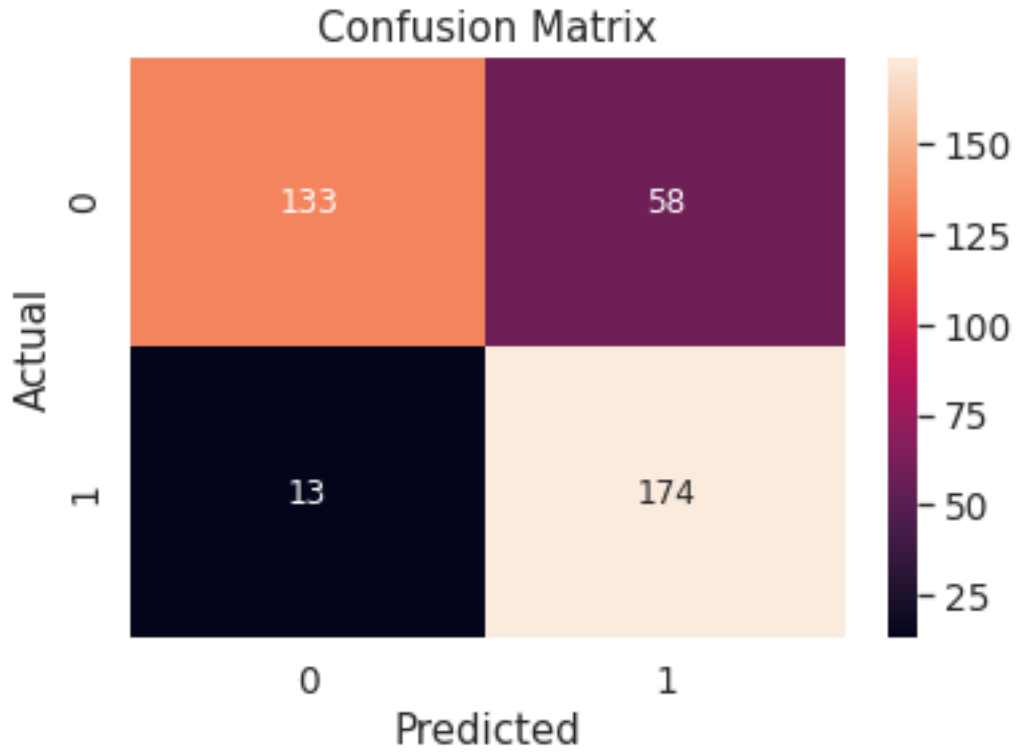
Random Forests

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

    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(forest, param_dist)
```

```
    # Building and fitting my_forest

    forest = RandomForestClassifier(max_depth = None, min_samples_leaf=8,
min_samples_split=2, 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
```

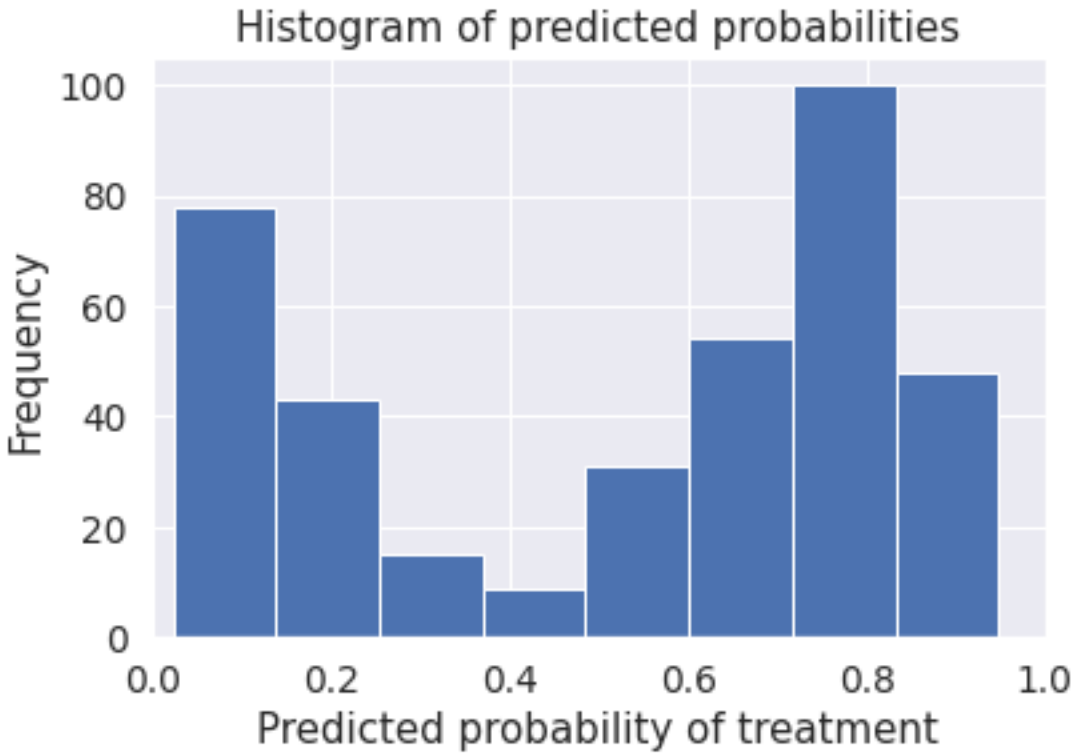
```
randomForest()
```

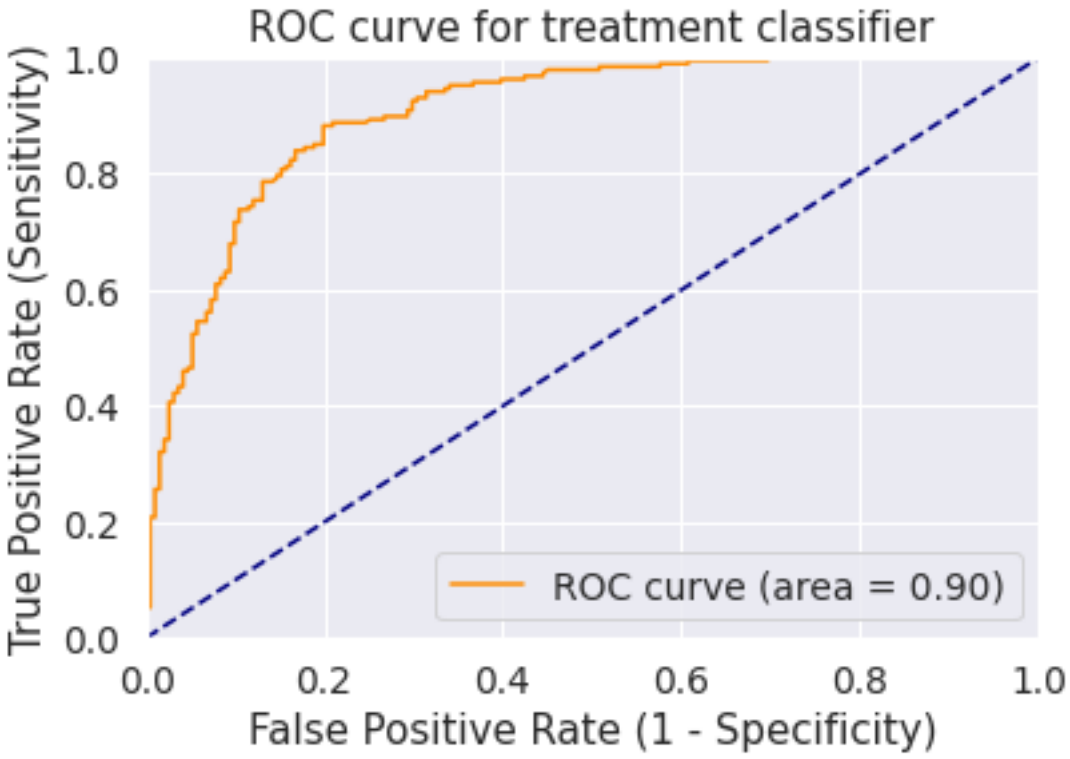
```
Rand. Best Score:  0.8305206349206349
Rand. Best Params:  {'criterion': 'entropy', 'max_depth': 3, 'max_features':
6, 'min_samples_leaf': 7, 'min_samples_split': 8}
[0.831, 0.831, 0.831, 0.831, 0.831, 0.831, 0.831, 0.832, 0.831, 0.831, 0.831,
0.831, 0.837, 0.834, 0.831, 0.832, 0.831, 0.831, 0.831, 0.831]
Accuracy: 0.8121693121693122
Null accuracy:
 0    191
1    187
Name: treatment, dtype: int64
Percentage of ones: 0.4947089947089947
Percentage of zeros: 0.5052910052910053
True: [0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1 0 0]
Pred: [1 0 0 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 0 0 0 0 1 0 0]
```




```
Classification Accuracy: 0.8121693121693122
Classification Error: 0.1878306878306878
False Positive Rate: 0.3036649214659686
Precision: 0.75
AUC Score: 0.8134081809782457
Cross-validated AUC: 0.8934280651104528
First 10 predicted responses:
 [1 0 0 0 1 1 0 1 1 1]
First 10 predicted probabilities of class members:
 [[0.2555794  0.7444206 ]
 [0.95069083 0.04930917]
 [0.93851009 0.06148991]
 [0.87096597 0.12903403]
 [0.40653554 0.59346446]
 [0.17282958 0.82717042]
 [0.89450448 0.10549552]
 [0.4065912  0.5934088 ]
 [0.20540631 0.79459369]
 [0.19337644 0.80662356]]
First 10 predicted probabilities:
 [[0.7444206 ]
 [0.04930917]
 [0.06148991]
 [0.12903403]
 [0.59346446]
 [0.82717042]
 [0.10549552]
 [0.5934088 ]
```

```
[0.79459369]
[0.80662356]]
```






```
[[133  58]
```

```
 [ 13 174]]
```

Bagging

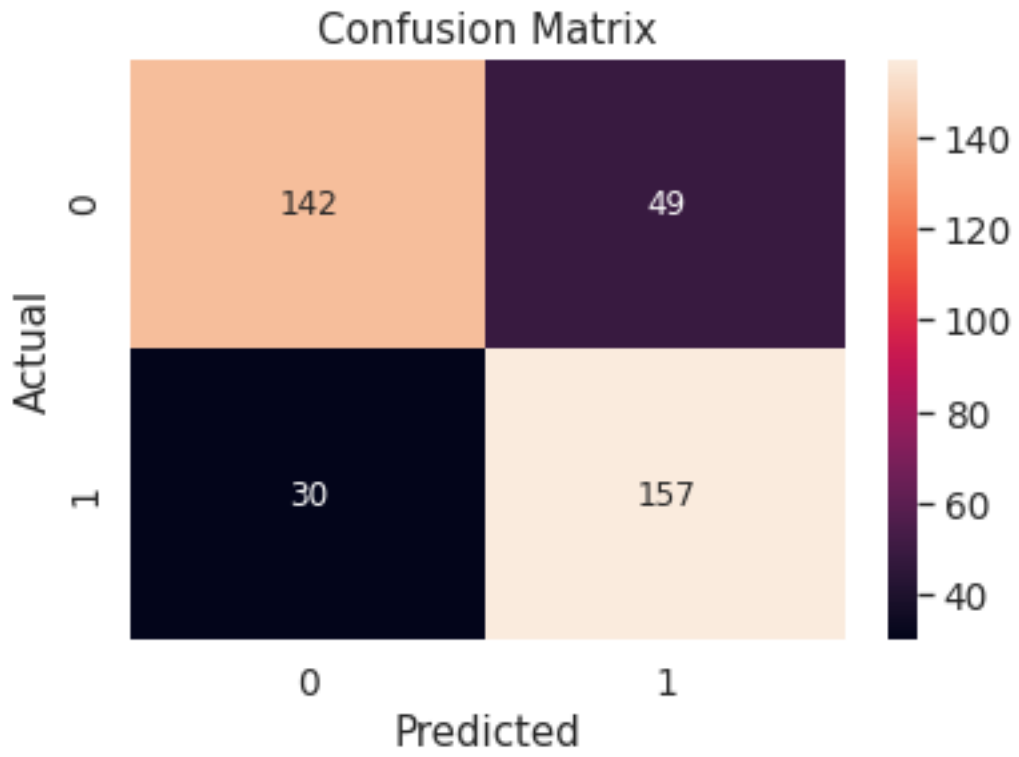
In [ ]:

```
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 [ ]:

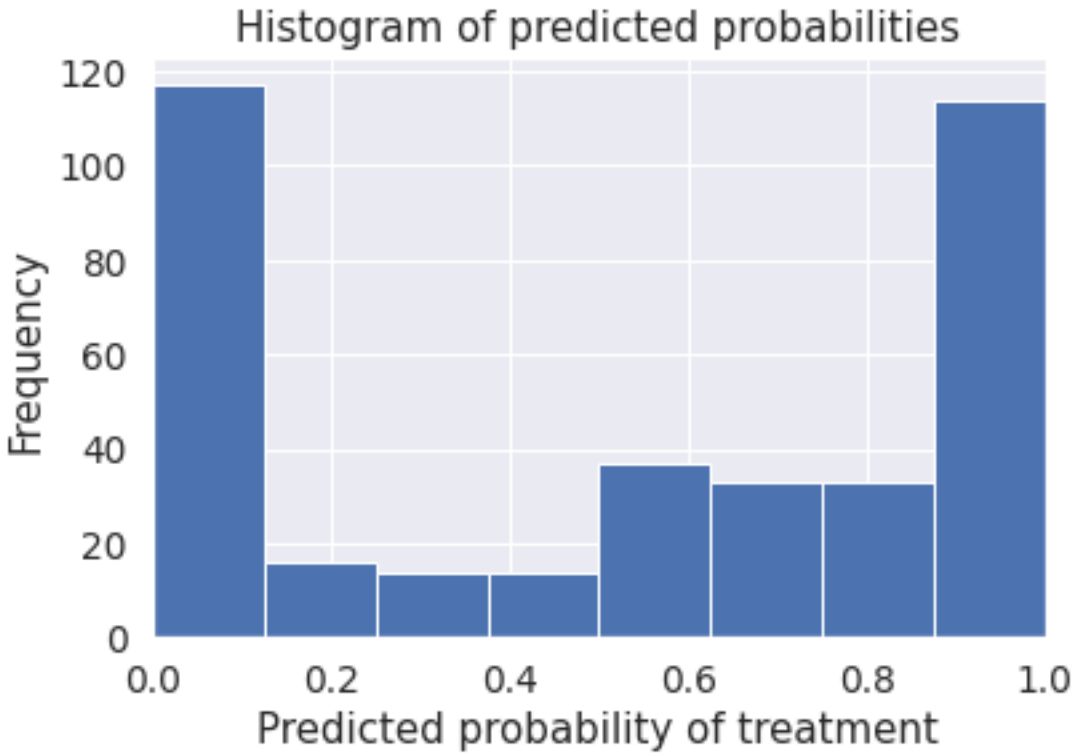
```
bagging()
```

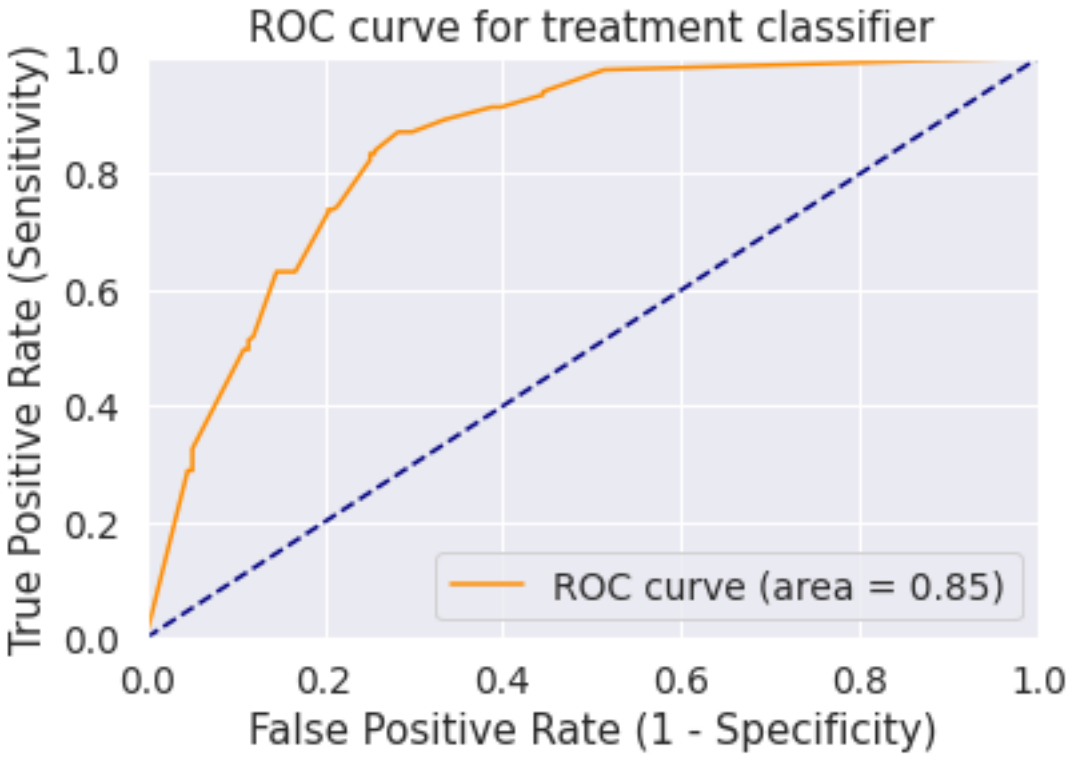
```
Accuracy: 0.791005291005291
Null accuracy:
 0    191
1    187
Name: treatment, dtype: int64
Percentage of ones: 0.4947089947089947
Percentage of zeros: 0.5052910052910053
True: [0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1 0 0]
Pred: [1 0 0 0 0 1 0 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 1 0 0]
```




```
Classification Accuracy: 0.791005291005291
Classification Error: 0.20899470899470896
False Positive Rate: 0.25654450261780104
Precision: 0.7621359223300971
AUC Score: 0.791513844947784
Cross-validated AUC: 0.8445159569577532
First 10 predicted responses:
 [1 0 0 0 0 1 0 1 1 1]
First 10 predicted probabilities of class members:
 [[0.33333333 0.66666667]
 [1.         0.        ]
 [1.         0.        ]
 [0.8        0.2       ]
 [0.5        0.5       ]
 [0.3        0.7       ]
 [1.         0.        ]
 [0.4        0.6       ]
 [0.         1.        ]
 [0.3        0.7       ]]
First 10 predicted probabilities:
 [[0.66666667]
 [0.        ]
 [0.        ]
 [0.2       ]
 [0.5       ]
 [0.7       ]
 [0.        ]
 [0.6       ]
```

```
[1.        ]
[0.7       ]]
```


Histogram of predicted probabilities
Frequency
120
100
80
60
40
20
0
0.0
0.2
0.4
0.6
0.8
1.0
Predicted probability of treatment


ROC curve for treatment classifier
True Positive Rate (Sensitivity)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate (1 - Specificity)
ROC curve (area = 0.85)

```
[[142  49]
```

```
 [ 30 157]]
```

Boosting

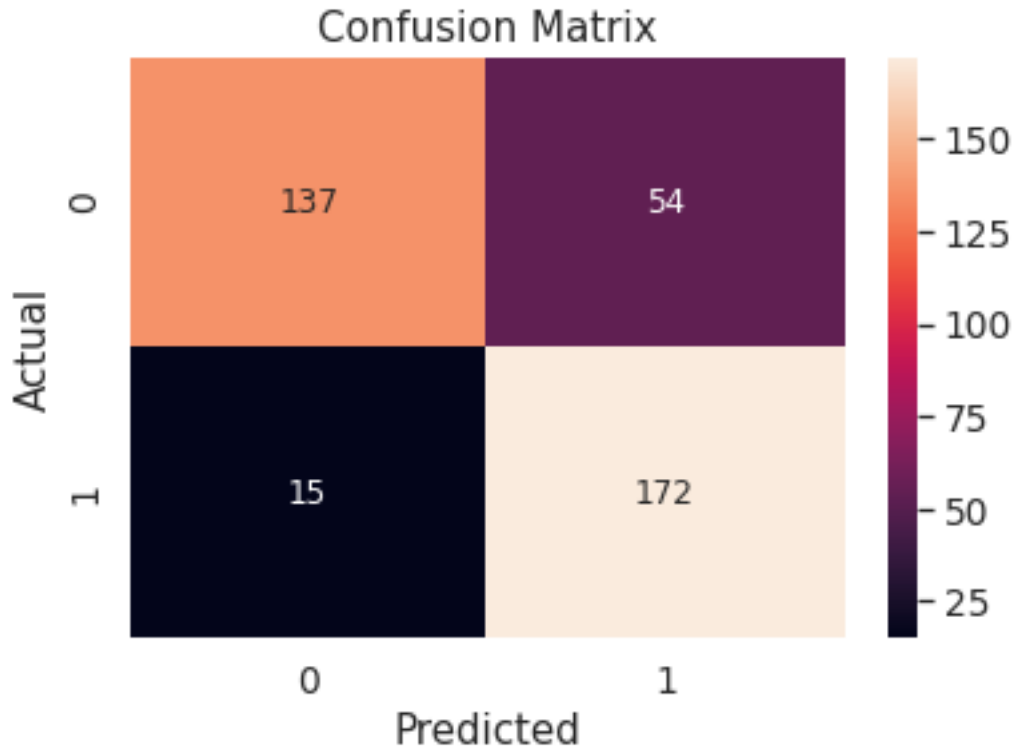
In [ ]:

```
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 [ ]:

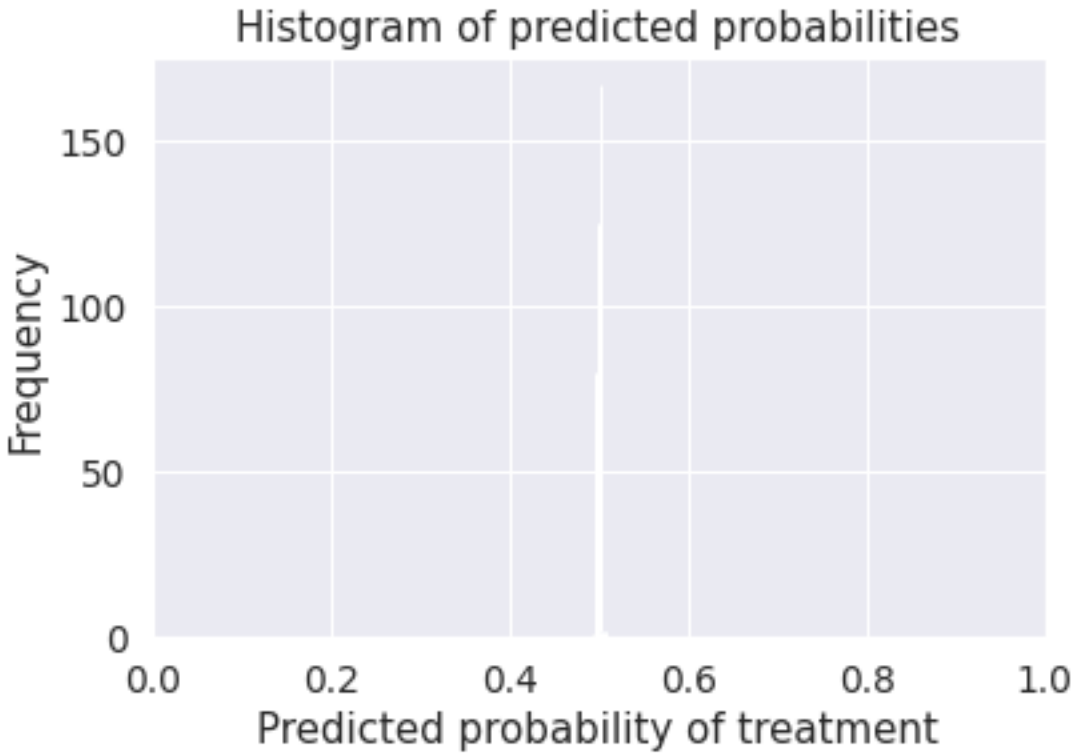
```
boosting()
```

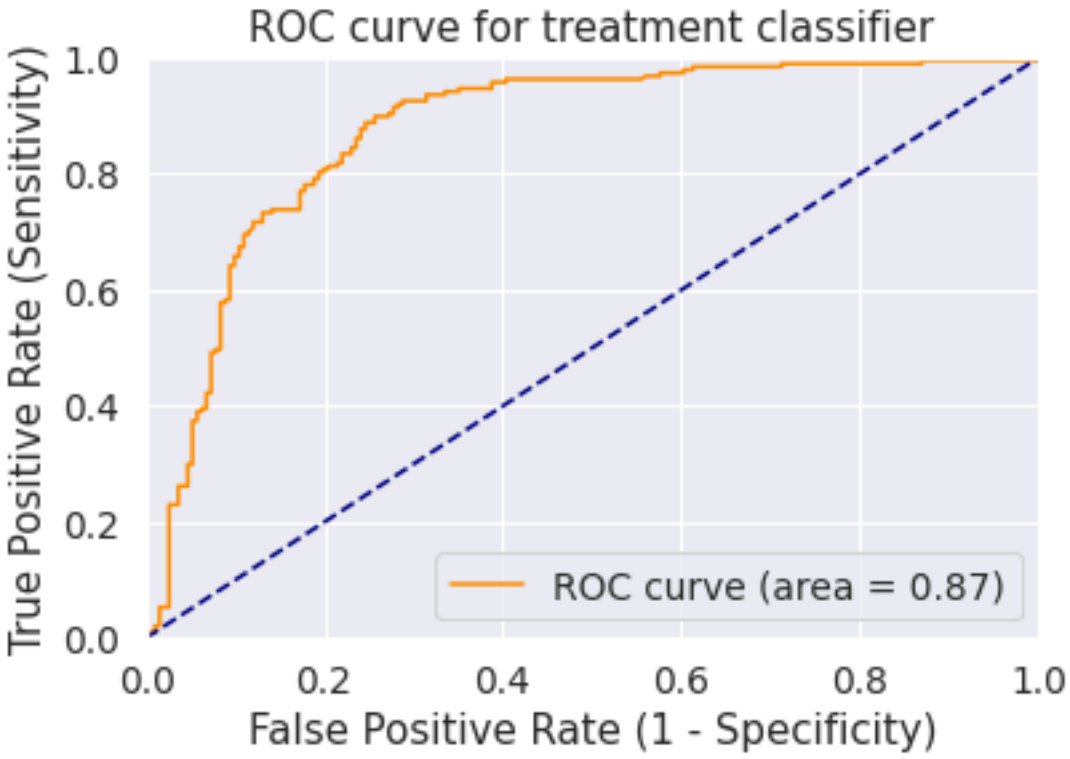
```
Accuracy: 0.8174603174603174
Null accuracy:
 0    191
1    187
Name: treatment, dtype: int64
Percentage of ones: 0.4947089947089947
Percentage of zeros: 0.5052910052910053
True: [0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1 0 0]
Pred: [1 0 0 0 0 1 0 1 1 1 0 1 1 0 1 1 1 1 0 0 0 0 1 0 0]
```




```
Classification Accuracy: 0.8174603174603174
Classification Error: 0.18253968253968256
False Positive Rate: 0.28272251308900526
Precision: 0.7610619469026548
AUC Score: 0.8185317915838397
Cross-validated AUC: 0.8746279095195426
First 10 predicted responses:
 [1 0 0 0 0 1 0 1 1 1]
First 10 predicted probabilities of class members:
 [[0.49924555 0.50075445]
 [0.50285507 0.49714493]
 [0.50291786 0.49708214]
 [0.50127788 0.49872212]
 [0.50013552 0.49986448]
 [0.49796157 0.50203843]
 [0.50046371 0.49953629]
 [0.49939483 0.50060517]
 [0.49921757 0.50078243]
 [0.49897133 0.50102867]]
First 10 predicted probabilities:
 [[0.50075445]
 [0.49714493]
 [0.49708214]
 [0.49872212]
 [0.49986448]
 [0.50203843]
 [0.49953629]
 [0.50060517]
```

```
[0.50078243]
[0.50102867]]
```






```
[[137  54]
```

```
 [ 15 172]]
```

Stacking

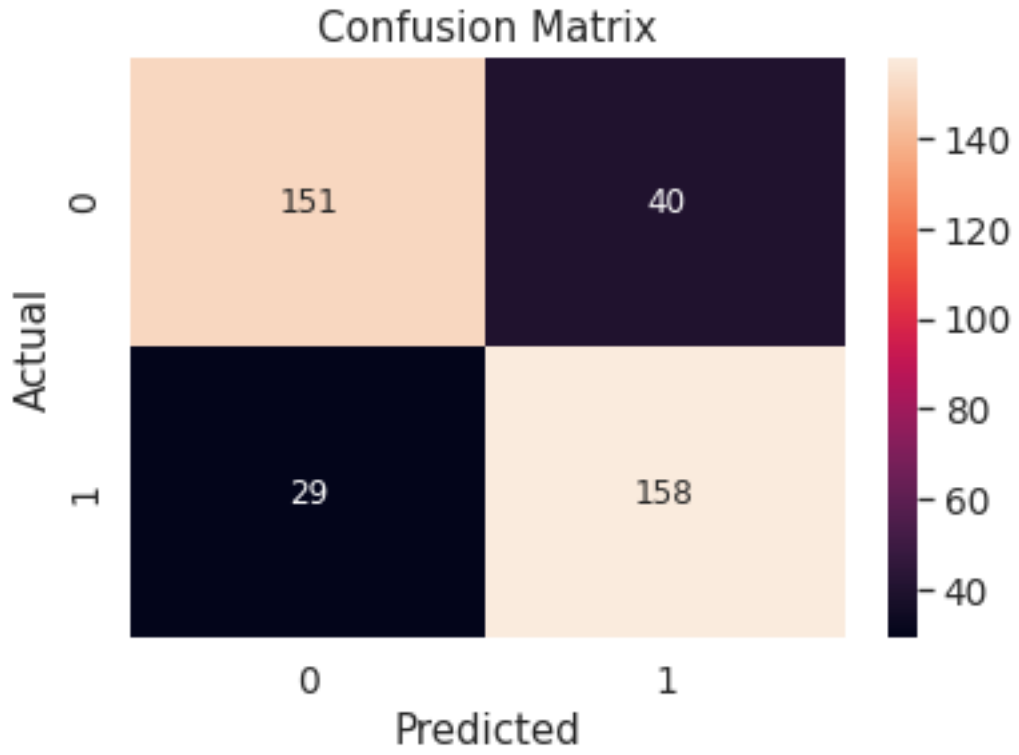
In [ ]:

```
def stacking():
    # Building and fitting
    clf1 = KNeighborsClassifier(n_neighbors=1)
    clf2 = RandomForestClassifier(random_state=1)
    clf3 = GaussianNB()
    lr = LogisticRegression()
    stack = StackingClassifier(classifiers=[clf1, clf2, clf3],
meta_classifier=lr)
    stack.fit(X_train, y_train)

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

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

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

In [ ]:

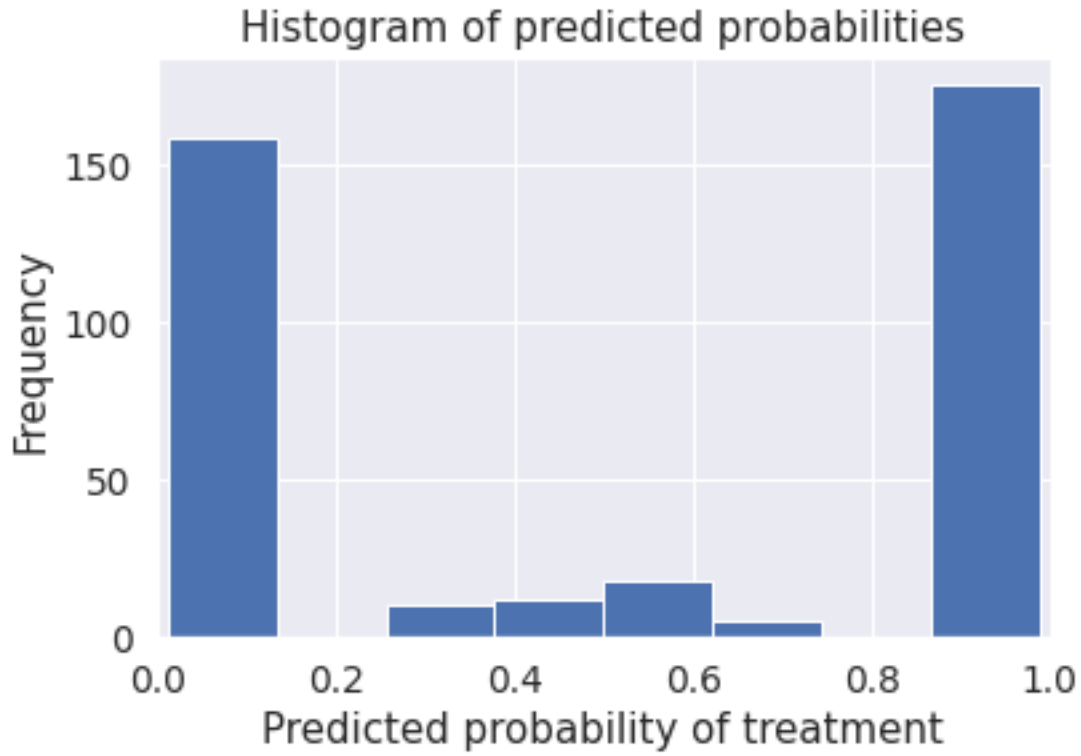
```
stacking()
```

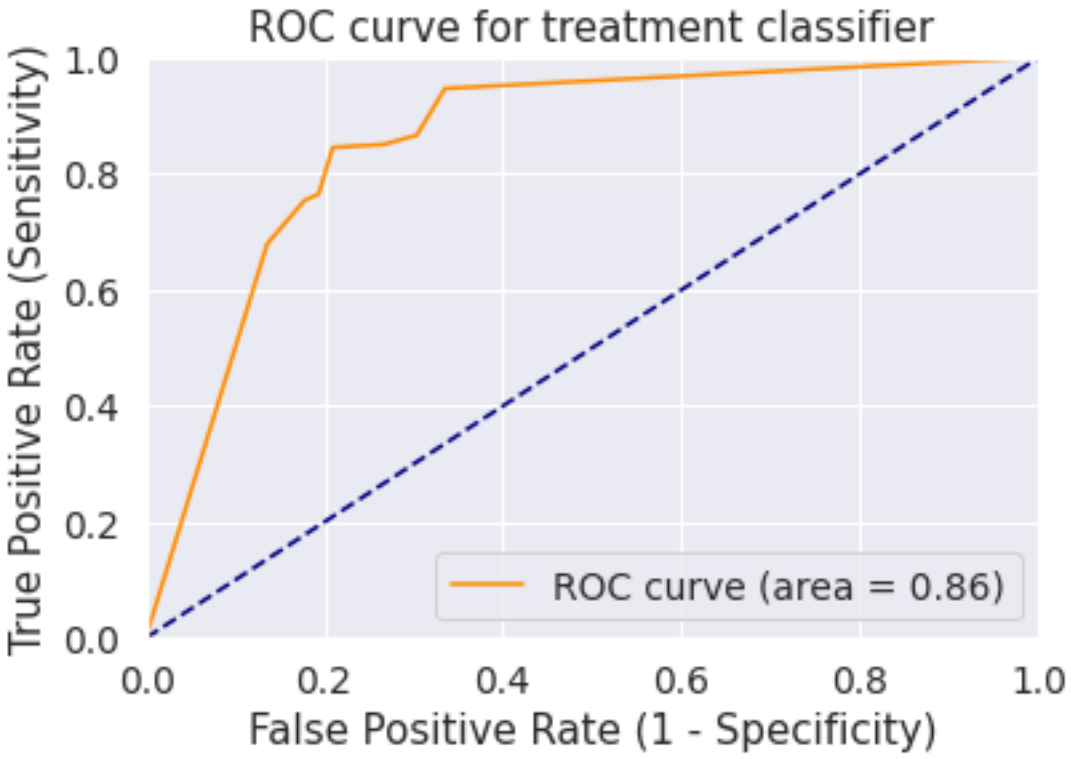
```
Accuracy: 0.8174603174603174
Null accuracy:
 0    191
1    187
Name: treatment, dtype: int64
Percentage of ones: 0.4947089947089947
Percentage of zeros: 0.5052910052910053
True: [0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1 0 0]
Pred: [1 0 0 0 0 1 0 0 1 1 0 1 1 1 1 1 0 1 0 0 0 0 1 0 0]
```




```
Classification Accuracy: 0.8174603174603174
Classification Error: 0.18253968253968256
False Positive Rate: 0.2094240837696335
Precision: 0.797979797979798
AUC Score: 0.8177478511633116
Cross-validated AUC: 0.8404446106773056
First 10 predicted responses:
 [1 0 0 0 0 1 0 0 1 1]
First 10 predicted probabilities of class members:
 [[0.0125215  0.9874785 ]
 [0.98811762 0.01188238]
 [0.98811762 0.01188238]
 [0.98811762 0.01188238]
 [0.98811762 0.01188238]
 [0.0125215  0.9874785 ]
 [0.98811762 0.01188238]
 [0.96810597 0.03189403]
 [0.03357297 0.96642703]
 [0.0125215  0.9874785 ]]
First 10 predicted probabilities:
 [[0.9874785 ]
 [0.01188238]
 [0.01188238]
 [0.01188238]
 [0.01188238]
 [0.9874785 ]
 [0.01188238]
 [0.03189403]
```

```
[0.96642703]
[0.9874785 ]]
```






```
[[151  40]
```

```
 [ 29 158]]
```

Predicting with Neural Network

Create input function

In [ ]:

```
%tensorflow_version 1.x
import tensorflow as tf
import argparse
```

```
TensorFlow 1.x selected.
```

In [ ]:

```
print(tf.__version__)
```

```
1.15.2
```

In [ ]:

```
batch_size = 100
train_steps = 1000

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
random_state=0)

def train_input_fn(features, labels, batch_size):
    """An input function for training"""
    # Convert the inputs to a Dataset.
    dataset = tf.data.Dataset.from_tensor_slices((dict(features), labels))

    # Shuffle, repeat, and batch the examples.
    return dataset.shuffle(1000).repeat().batch(batch_size)

def eval_input_fn(features, labels, batch_size):
```

```
    """An input function for evaluation or prediction"""
    features=dict(features)
    if labels is None:
        # No labels, use only features.
        inputs = features
    else:
        inputs = (features, labels)

    # Convert the inputs to a Dataset.
    dataset = tf.data.Dataset.from_tensor_slices(inputs)

    # Batch the examples
    assert batch_size is not None, "batch_size must not be None"
    dataset = dataset.batch(batch_size)

    # Return the dataset.
    return dataset
```

Define the feature columns

In [ ]:

```
# Define Tensorflow feature columns
age = tf.feature_column.numeric_column("Age")
gender = tf.feature_column.numeric_column("Gender")
family_history = tf.feature_column.numeric_column("family_history")
benefits = tf.feature_column.numeric_column("benefits")
care_options = tf.feature_column.numeric_column("care_options")
anonymity = tf.feature_column.numeric_column("anonymity")
leave = tf.feature_column.numeric_column("leave")
work_interfere = tf.feature_column.numeric_column("work_interfere")
```

```
feature_columns = [age, gender, family_history, benefits, care_options,
anonymity, leave, work_interfere]
```

Instantiate an Estimator

In [ ]:

```
# Build a DNN with 2 hidden layers and 10 nodes in each hidden layer.
model = tf.estimator.DNNClassifier(feature_columns=feature_columns,
                                   hidden_units=[10, 10],

optimizer=tf.train.ProximalAdagradOptimizer(
                                     learning_rate=0.1,
                                     l1_regularization_strength=0.001
                                   ))
```

```
INFO:tensorflow:Using default config.
WARNING:tensorflow:Using temporary folder as model directory: /tmp/tmpvhzghvo
w
INFO:tensorflow:Using config: {'_model_dir': '/tmp/tmpvhzghvow', '_tf_random_
seed': None, '_save_summary_steps': 100, '_save_checkpoints_steps': None, '_s
ave_checkpoints_secs': 600, '_session_config': allow_soft_placement: true
graph_options {
  rewrite_options {
    meta_optimizer_iterations: ONE
  }
}
, '_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000, '_log_s
tep_count_steps': 100, '_train_distribute': None, '_device_fn': None, '_proto
col': None, '_eval_distribute': None, '_experimental_distribute': None, '_exp
erimental_max_worker_delay_secs': None, '_session_creation_timeout_secs': 720
0, '_service': None, '_cluster_spec': <tensorflow.python.training.server_lib.
ClusterSpec object at 0x7fdbe67cab50>, '_task_type': 'worker', '_task_id': 0,
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master': '', '_is_chi
ef': True, '_num_ps_replicas': 0, '_num_worker_replicas': 1}
```

Train the model

In [ ]:

```
model.train(input_fn=lambda:train_input_fn(X_train, y_train, batch_size),
steps=train_steps)
```

```
WARNING:tensorflow:From /tensorflow-1.15.2/python3.7/tensorflow_core/python/training/training_util.py:236: Variable.initialized_value (from tensorflow.python.ops.variables) is deprecated and will be removed in a future version.
Instructions for updating:
Use Variable.read_value. Variables in 2.X are initialized automatically both in eager and graph (inside tf.defun) contexts.
INFO:tensorflow:Calling model_fn.
WARNING:tensorflow:Entity <bound method _DNNModel.call of <tensorflow_estimator.python.estimator.canned.dnn._DNNModel object at 0x7fdc27ff0290>> could not be transformed and will be executed as-is. Please report this to the AutoGraph team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output. Cause: module 'gast' has no attribute 'Index'
WARNING: Entity <bound method _DNNModel.call of <tensorflow_estimator.python.estimator.canned.dnn._DNNModel object at 0x7fdc27ff0290>> could not be transformed and will be executed as-is. Please report this to the AutoGraph team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output. Cause: module 'gast' has no attribute 'Index'
WARNING:tensorflow:From /tensorflow-1.15.2/python3.7/tensorflow_core/python/ops/resource_variable_ops.py:1630: calling BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
WARNING:tensorflow:From /tensorflow-1.15.2/python3.7/tensorflow_estimator/python/estimator/canned/head.py:437: to_float (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use `tf.cast` instead.
WARNING:tensorflow:From /tensorflow-1.15.2/python3.7/tensorflow_core/python/ops/nn_impl.py:183: where (from tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 0 into /tmp/tmpvhzghvow/model.ckpt.
INFO:tensorflow:loss = 83.05777, step = 1
INFO:tensorflow:global_step/sec: 377.535
INFO:tensorflow:loss = 31.241507, step = 101 (0.274 sec)
INFO:tensorflow:global_step/sec: 417.82
INFO:tensorflow:loss = 36.68381, step = 201 (0.233 sec)
INFO:tensorflow:global_step/sec: 490.258
INFO:tensorflow:loss = 39.932377, step = 301 (0.204 sec)
INFO:tensorflow:global_step/sec: 426.259
INFO:tensorflow:loss = 35.767494, step = 401 (0.238 sec)
INFO:tensorflow:global_step/sec: 468.688
INFO:tensorflow:loss = 33.39394, step = 501 (0.213 sec)
INFO:tensorflow:global_step/sec: 538.558
```

```
INFO:tensorflow:loss = 47.758324, step = 601 (0.185 sec)
INFO:tensorflow:global_step/sec: 529.012
INFO:tensorflow:loss = 33.860725, step = 701 (0.186 sec)
INFO:tensorflow:global_step/sec: 524.729
INFO:tensorflow:loss = 42.056705, step = 801 (0.190 sec)
INFO:tensorflow:global_step/sec: 520.137
INFO:tensorflow:loss = 30.41483, step = 901 (0.195 sec)
INFO:tensorflow:Saving checkpoints for 1000 into /tmp/tmpvhzghvow/model.ckpt.
INFO:tensorflow:Loss for final step: 30.373766.
```

Out[ ]:

```
<tensorflow_estimator.python.estimator.canned.dnn.DNNClassifier at 0x7fdbe67c
a350>
```

Evaluate the model

In [ ]:

```
# Evaluate the model.
eval_result = model.evaluate(
    input_fn=lambda:eval_input_fn(X_test, y_test, batch_size))

print('\nTest set accuracy: {accuracy:0.2f}\n'.format(**eval_result))

#Data for final graph
accuracy = eval_result['accuracy'] * 100
methodDict['Neural Network'] = accuracy
```

```
INFO:tensorflow:Calling model_fn.
WARNING:tensorflow:Entity <bound method _DNNModel.call of <tensorflow_estimat
or.python.estimator.canned.dnn._DNNModel object at 0x7fdbda178410>> could not
be transformed and will be executed as-is. Please report this to the AutoGrap
h team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOG
RAPH_VERBOSITY=10`) and attach the full output. Cause: module 'gast' has no a
ttribute 'Index'
WARNING: Entity <bound method _DNNModel.call of <tensorflow_estimator.python.
estimator.canned.dnn._DNNModel object at 0x7fdbda178410>> could not be transf
ormed and will be executed as-is. Please report this to the AutoGraph team. W
hen filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERB
OSITY=10`) and attach the full output. Cause: module 'gast' has no attribute
'Index'
WARNING:tensorflow:From /tensorflow-1.15.2/python3.7/tensorflow_core/python/o
ps/metrics_impl.py:2026: div (from tensorflow.python.ops.math_ops) is depreca
ted and will be removed in a future version.
Instructions for updating:
Deprecated in favor of operator or tf.math.divide.
```

```
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect PR-AUCs; pl
ease switch to "careful_interpolation" instead.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect PR-AUCs; pl
ease switch to "careful_interpolation" instead.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2021-06-14T17:30:33Z
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from /tmp/tmpvhzghvow/model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Finished evaluation at 2021-06-14-17:30:34
INFO:tensorflow:Saving dict for global step 1000: accuracy = 0.7962963, accur
acy_baseline = 0.505291, auc = 0.8854047, auc_precision_recall = 0.8693078, a
verage_loss = 0.45305556, global_step = 1000, label/mean = 0.49470899, loss =
42.81375, precision = 0.73504275, prediction/mean = 0.5135381, recall = 0.919
7861
INFO:tensorflow:Saving 'checkpoint_path' summary for global step 1000: /tmp/t
mpvhzghvow/model.ckpt-1000

Test set accuracy: 0.80
```

Making predictions (inferring) from the trained model

In [ ]:

```
predictions = list(model.predict(input_fn=lambda:eval_input_fn(X_train,
y_train, batch_size=batch_size)))
```

```
INFO:tensorflow:Calling model_fn.
WARNING:tensorflow:Entity <bound method _DNNModel.call of <tensorflow_estimat
or.python.estimator.canned.dnn._DNNModel object at 0x7fdbe5f66550>> could not
be transformed and will be executed as-is. Please report this to the AutoGrap
h team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOG
RAPH_VERBOSITY=10`) and attach the full output. Cause: module 'gast' has no a
ttribute 'Index'
WARNING: Entity <bound method _DNNModel.call of <tensorflow_estimator.python.
estimator.canned.dnn._DNNModel object at 0x7fdbe5f66550>> could not be transf
ormed and will be executed as-is. Please report this to the AutoGraph team. W
hen filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERB
OSITY=10`) and attach the full output. Cause: module 'gast' has no attribute
'Index'
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from /tmp/tmpvhzghvow/model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
```

In [ ]:

```
# Generate predictions from the model
template = ('\nIndex: "{}", Prediction is "{}" ({:.1f}%), expected "{}"')

# Dictionary for predictions
col1 = []
col2 = []
col3 = []


for idx, input, p in zip(X_train.index, y_train, predictions):
    v  = p["class_ids"][0]
    class_id = p['class_ids'][0]
    probability = p['probabilities'][class_id] # Probability

    # Adding to dataframe
    col1.append(idx) # Index
    col2.append(v) # Prediction
    col3.append(input) # Expecter


    #print(template.format(idx, v, 100 * probability, input))


results = pd.DataFrame({'index':col1, 'prediction':col2, 'expected':col3})
results.head()
```

Out[ ]:

	index	prediction	expected
0	929	0	0

	index	prediction	expected
1	901	1	1
2	579	1	1
3	367	1	1
4	615	1	1

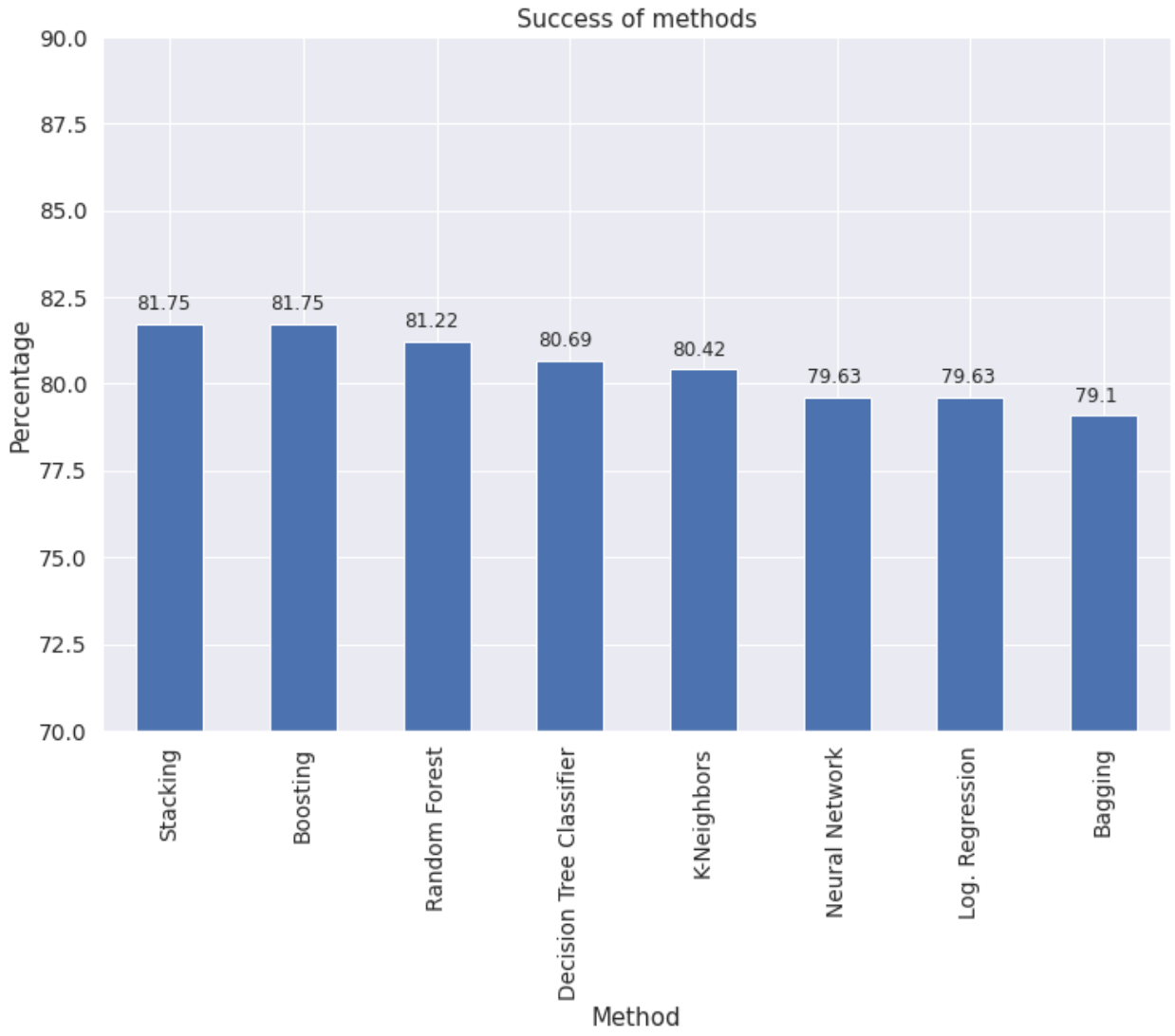
Success method plot

In [ ]:

```
def plotSuccess():
    s = pd.Series(methodDict)
    s = s.sort_values(ascending=False)
    plt.figure(figsize=(12,8))
    #Colors
    ax = s.plot(kind='bar')
    for p in ax.patches:
        ax.annotate(str(round(p.get_height(),2)), (p.get_x() * 1.005,
p.get_height() * 1.005))
    plt.ylim([70.0, 90.0])
    plt.xlabel('Method')
    plt.ylabel('Percentage')
    plt.title('Success of methods')


    plt.show()
```

In [ ]:

```
plotSuccess()
```



Creating predictions on test set

In [ ]:

```
# Generate predictions with the best method
clf = AdaBoostClassifier()
clf.fit(X, y)
dfTestPredictions = clf.predict(X_test)

# Write predictions to csv file
```

```
# We don't have any significative field so we save the index
results = pd.DataFrame({'Index': X_test.index, 'Treatment':
dfTestPredictions})
# Save to file
# This file will be visible after publishing in the output section
results.to_csv('results.csv', index=False)
results.head()
```

Out[ ]:

	Index	Treatment
0	5	1
1	494	0
2	52	0
3	984	0
4	186	0

Submission

In [ ]:

```
# We don't have any significative field so we save the index
results = pd.DataFrame({'Index': X_test.index, 'Treatment':
dfTestPredictions})
results
```

Out[ ]:

	Index	Treatment
0	5	1
1	494	0
2	52	0
3	984	0
4	186	0
...	...	...
373	1084	1
374	506	0
375	1142	0
376	1124	0
377	689	1

378 rows × 2 columns

<!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>



   <!-- CSS  -->

   <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">

   <link href="./static/css/materialize.css" type="text/css" rel="stylesheet"
media="screen,projection"/>

   <link href="./static/css2/style.css" type="text/css" rel="stylesheet" media="screen,projection"/>

 </head>



 <body>

   <div class="section no-pad-bot" id="index-banner">

    <div class="container">

     <br><br>

     <h1 class="header center orange-text">Mental Health Prediction</h1>

      <div class="row center">

        <h5 class="header col s12 light">Predict the probability whether a person requires Mental
Treatment

          <br>

        </h5>

      </div>

     <div class="row center">

      <h6 class="header col s12 light">Instructions to fill form<br>1. Enter age in years<br>2. For
Gender: Enter 0 for male, 1 for female and 2 for transgender<br>3. For Family History: Enter 0 for No
and 1 for Yes</h6>

     </div>
```

```
      <div class="row">
        <form action='/predict' method="post" class="col s12">
          <div class="row">
            <div class="input-field col s4">
              <label for="first_name"><b>Age</b></label>
              <br>
              <input placeholder="Age" name="Age" id="first_name" type="text" class="validate">
            </div>
            <div class="input-field col s4">
              <label for="last_name"><b>Gender </b></label>
              <br>
              <input id="last_name" name="Gender" placeholder="Gender"  type="text" class="validate">
            </div>
            <div class="input-field col s4">
              <label for="_name"><b>Family History</b></label>
              <br>
              <input id="_name" name="Family_history" placeholder="Family History" type="text"
class="validate">
            </div>
          </div>
          <div class="row center">
            <button type="submit" class="btn-large waves-effect waves-light orange">Predict
Probability</button>
          </div>
        </form>
```

```
      </div>

      <br>

      <div class="row center">

        <h6 class="waves-effect waves-light orange">{{pred}}<br></h6>

      </div>

    </div>

  </div>

  <br><br>

 </div>

</div>

</body>

</html>
```

Demo Link https://teams.microsoft.com/l/meetup-join/19:gJ22plE7XeryaFq4EoHDDpi1AQOVdTV8vYHJI3CbAYc1@thread.tacv2/1700669306048?context=%7B%22Tid%22:%22ff335ba2-bb68-489a-bbdd-f49ab4319838%22,%22Oid%22:%2215c4f47e-e9ca-4978-a1f6-966121e615be%22%7D