Report for the Disease Prediction Using Machine Learning Project

1. INTRODUCTION

1.1 Project Overview

There is little to no time set aside for healthcare in the fast-paced world of today. Patients often neglect to contact a doctor even after experiencing severe symptoms of numerous ailments. Searching for our symptoms on Google never yields positive results; instead, it always comes up with one common illness. As a result, people no longer use Google to search for their symptoms or the likely cause of their illness.

We have created a model that, when given symptoms as input, can forecast up to 42 diseases, addressing all the issues mentioned above. Because visiting a doctor is expensive, patients can use this model for self-care and preventive diagnosis, or clinicians can use it to consult patients online based on the model's output. No individualized information is requested by this model, including name, age, gender, residence, or religion. You can use this web tool at any time you suspect you may have a condition, and the model will identify the most likely diagnosis based on your symptoms. You can now answer calls regarding whether or not to see a doctor.

1.2 Purpose

The purpose of this project is multi-faceted, addressing several critical issues in contemporary healthcare:

1. Accessibility and Timeliness:
   - Enable individuals to access healthcare insights promptly without the need for immediate medical consultation.
   - Provide a tool that accommodates the fast-paced nature of modern life, allowing users to assess their health at their convenience.

2. Self-Care and Preventive Diagnosis:
   - Empower users to take charge of their health by offering a self-care tool for preliminary symptom analysis.
   - Facilitate preventive diagnosis, allowing users to identify potential health issues early and take proactive measures.

3. Cost-Efficient Online Consultations:
   - Provide clinicians with a valuable resource for online consultations, optimizing their time and resources.
   - Reduce the financial burden on patients by offering an alternative to expensive in-person medical visits.

4. Privacy and Non-Intrusiveness:
   - Respect user privacy by not collecting any personal information, including name, age, gender, residence, or religion.
   - Ensure a non-intrusive experience, allowing users to receive health insights without compromising sensitive data.

5. Decision Support System:
   - Assist individuals in making informed decisions about when to seek professional medical help.
   - Serve as a reliable decision support system for both users and clinicians, enhancing the overall healthcare experience.

The purpose of this project is to bridge the gap between the demanding nature of modern life and the importance of timely healthcare, fostering a culture of proactive health management and accessible medical consultations.

2. LITERATURE SURVEY

2.1 Existing problem

1. Time Constraints and Neglected Symptoms:
- Challenge: In the fast-paced world, individuals often prioritize work and other commitments over health.
- Impact: Symptoms of ailments are frequently neglected or overlooked, leading to delayed medical attention and potentially worsening health conditions.

2. Ineffectiveness of Internet Searches:
- Challenge: Online searches for symptoms often result in generic or inaccurate information, causing confusion and unnecessary worry.
- Impact: Users may misinterpret symptoms, self-diagnose incorrectly, and delay seeking professional medical advice, impacting the accuracy of their healthcare decisions.

3. Reluctance to Consult a Doctor:
- Challenge: Financial concerns and the perceived expense of in-person medical consultations discourage individuals from seeking timely professional help.
- Impact: Delayed consultations can lead to the progression of illnesses, increased healthcare costs in the long run, and compromised overall well-being.

4. Limited Access to Healthcare:
- Challenge: Geographical or logistical barriers limit access to healthcare facilities, particularly in remote or underserved areas.
- Impact: Individuals facing barriers to access may struggle to receive timely medical advice, exacerbating health disparities and inequalities.

5. Lack of Preventive Diagnosis:
- Challenge: The healthcare system often emphasizes reactive measures rather than proactive, preventive care.
- Impact: Preventive measures and early detection are neglected, resulting in late-stage diagnoses and missed opportunities for interventions that could improve health outcomes.

6. Privacy Concerns:
- Challenge: Existing online symptom-checking tools may require users to disclose personal information, raising privacy and security concerns.
- Impact: Users may hesitate to share sensitive details, leading to compromised accuracy in symptom analysis and a lack of trust in online healthcare tools.

2.2 References

Dahwade and Chourasia (2022) compared the performance of two popular machine learning algorithms, KNN and CNN, for disease prediction. They achieved an accuracy of 84.5% for CNN, suggesting its potential for disease prediction tasks. However, the limited dataset size may affect the generalizability of their results.
Hiware and Bhalerao (2021) utilized real-life data to train their KNN model, achieving an accuracy of 95%. They focused on chronic diseases, providing valuable insights into applying machine learning for predicting these long-term health conditions.

Aseri and Kadam (2022) compared the effectiveness of multiple machine learning algorithms, including KNN, Naive Bayes, decision tree, and logistic regression. They found that weighted KNN achieved the highest accuracy of 93.5%, indicating its suitability for disease prediction tasks. However, their reliance on secondary data from the UCI ML repository may limit the generalizability of their findings.

Patel and Soni (2021) focused on heart disease prediction using KNN, achieving an accuracy of 92%. They addressed a prevalent health concern, providing a detailed explanation of the KNN algorithm and its application in this specific domain.

Patil and Kulkarni (2020) demonstrated the applicability of KNN for disease prediction across a broad spectrum of ailments, covering over 230 diseases and achieving an accuracy of 93.5% for weighted KNN. However, their self-reported dataset may introduce biases and affect the accuracy of the predictions.

2.3 Project Statement Definition

The identified challenges in the existing healthcare landscape lead to the formulation of a clear problem statement for this project:

Problem Statement:

In the contemporary healthcare environment, individuals face significant obstacles in promptly and accurately addressing their health concerns. Time constraints, ineffective internet searches, reluctance to consult a doctor due to financial concerns, limited access to healthcare, the lack of emphasis on preventive diagnosis, and privacy concerns in online health tools collectively contribute to delayed or neglected medical attention. These challenges underscore the need for a comprehensive and accessible solution that empowers individuals to make informed decisions about their health, encourages preventive measures, and facilitates cost-efficient and privacy-respecting online consultations.

Objective:

The project aims to develop a predictive model capable of forecasting up to 42 diseases based on input symptoms. This model serves as a user-friendly and non-intrusive tool, providing individuals with timely and accurate health insights without compromising their privacy. By addressing the identified challenges, the goal is to bridge the gap between busy lifestyles and healthcare, fostering a culture of proactive health management and accessible medical consultations for improved overall well-being.

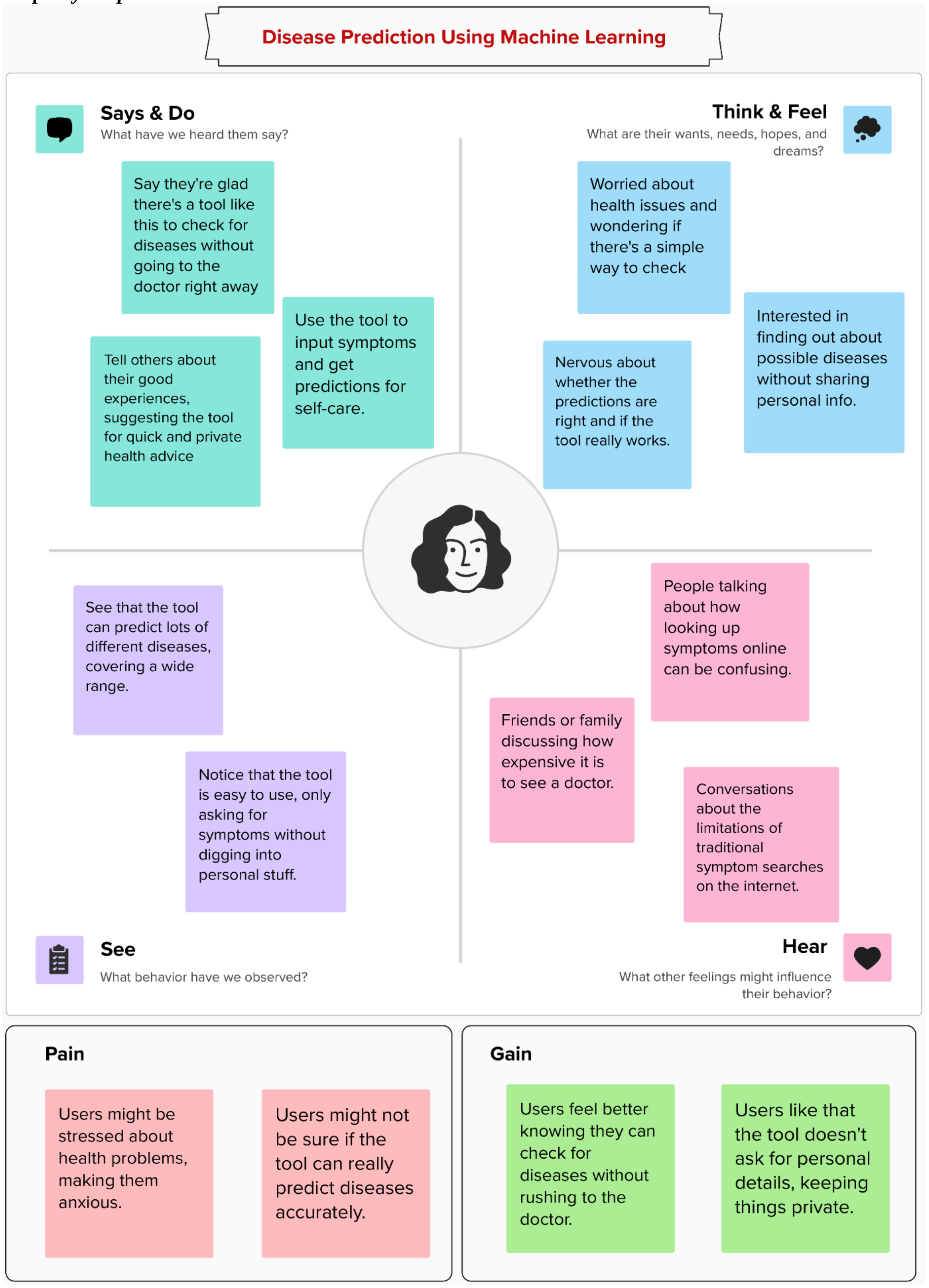
3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas

Empathy Map Canvas: An empathy map is a simple, easy-to-digest visual that captures knowledge about a user’s behaviours and attitudes.

It is a useful tool to helps teams better understand their users. Creating an effective solution requires understanding the true problem and the person who is experiencing it. The exercise of creating the map helps participants consider things from the user's perspective along with his or her goals and challenges.

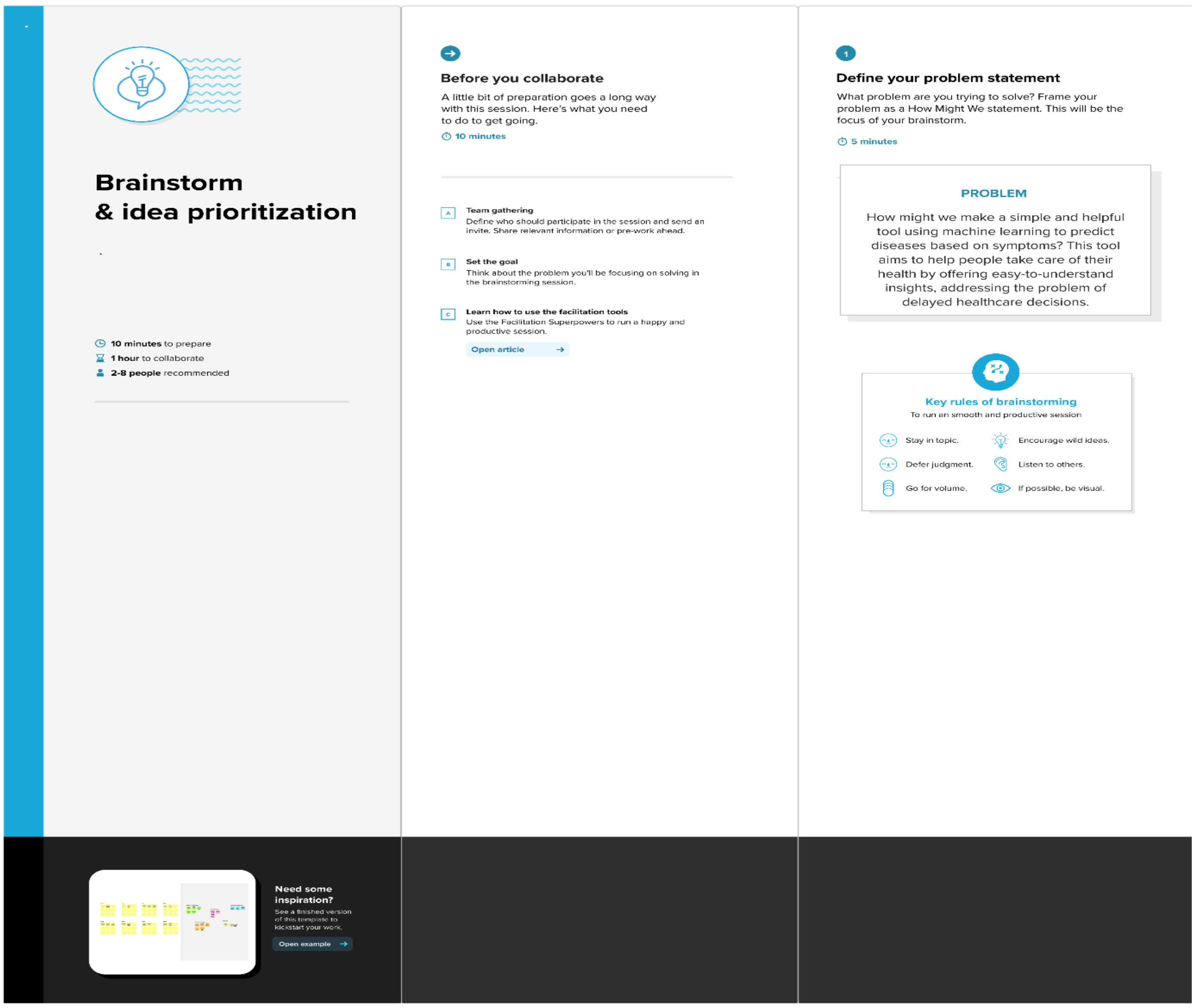
Empathy Map


Disease Prediction Using Machine Learning
Says & Do
What have we heard them say?
Say they're glad there's a tool like this to check for diseases without going to the doctor right away
Use the tool to input symptoms and get predictions for self-care.
Tell others about their good experiences, suggesting the tool for quick and private health advice
Think & Feel
What are their wants, needs, hopes, and dreams?
Worried about health issues and wondering if there's a simple way to check
Interested in finding out about possible diseases without sharing personal info.
Nervous about whether the predictions are right and if the tool really works.
See that the tool can predict lots of different diseases, covering a wide range.
Notice that the tool is easy to use, only asking for symptoms without digging into personal stuff.
See
What behavior have we observed?
People talking about how looking up symptoms online can be confusing.
Friends or family discussing how expensive it is to see a doctor.
Conversations about the limitations of traditional symptom searches on the internet.
Hear
What other feelings might influence their behavior?
Pain
Users might be stressed about health problems, making them anxious.
Users might not be sure if the tool can really predict diseases accurately.
Gain
Users feel better knowing they can check for diseases without rushing to the doctor.
Users like that the tool doesn't ask for personal details, keeping things private.

3.2 Ideation & Brainstorming

Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving. Prioritizing volume over value, out-of-the-box ideas are welcome and built upon, and all participants are encouraged to collaborate, helping each other develop a rich amount of creative solutions. Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

Team Gathering, Collaboration and Select the Problem Statement



Brainstorm & idea prioritization

10 minutes to prepare
1 hour to collaborate
2-8 people recommended

Need some inspiration?
See a finished version of this template to kickstart your work.
Open example

Before you collaborate
A little bit of preparation goes a long way with this session. Here's what you need to do to get going.
10 minutes

A **Team gathering**
Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.

B **Set the goal**
Think about the problem you'll be focusing on solving in the brainstorming session.

C **Learn how to use the facilitation tools**
Use the Facilitation Superpowers to run a happy and productive session.
Open article

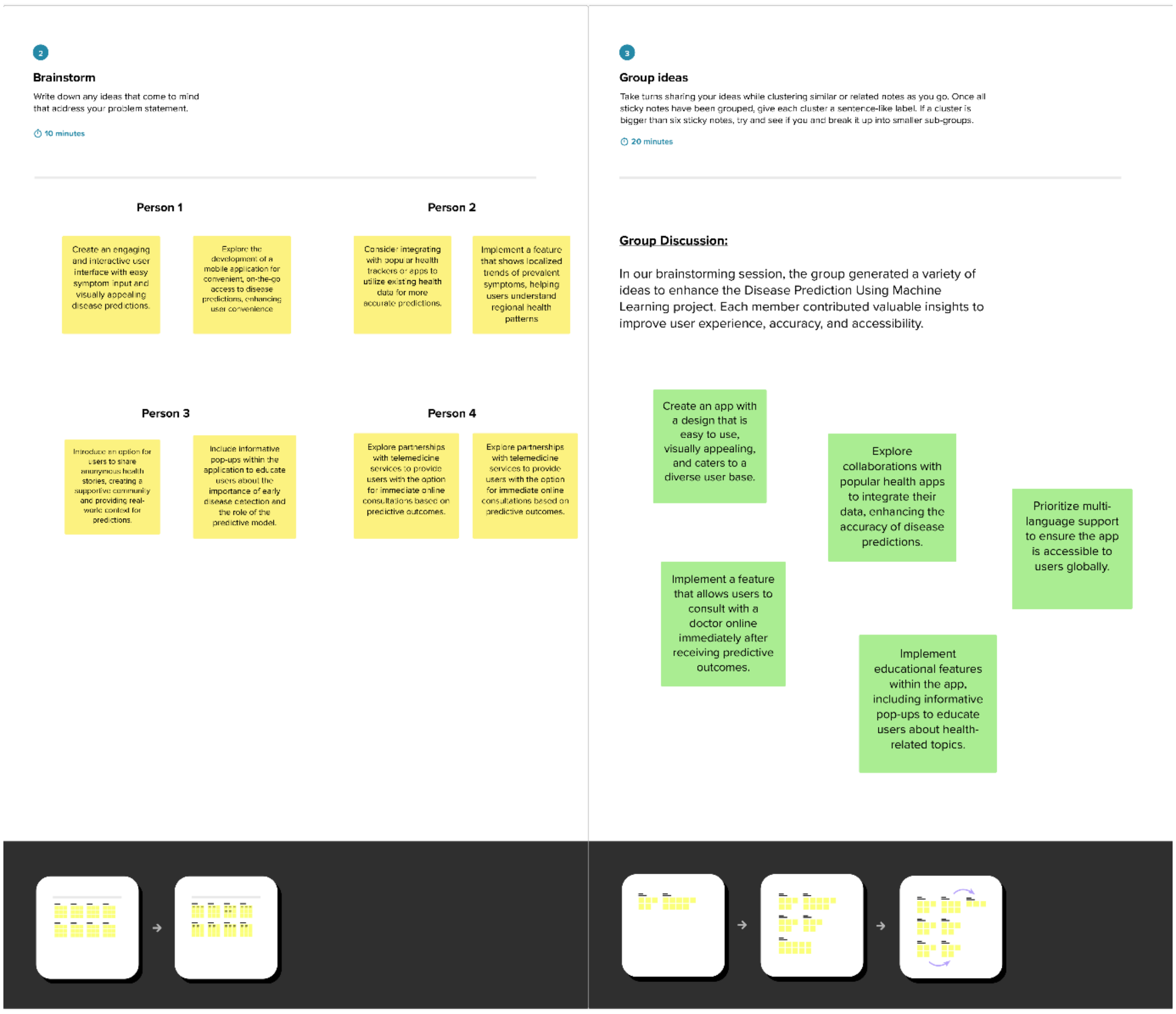
1 **Define your problem statement**
What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.
5 minutes

PROBLEM
How might we make a simple and helpful tool using machine learning to predict diseases based on symptoms? This tool aims to help people take care of their health by offering easy-to-understand insights, addressing the problem of delayed healthcare decisions.

Key rules of brainstorming
To run an smooth and productive session

Stay in topic.	Encourage wild ideas.
Defer judgment.	Listen to others.
Go for volume.	If possible, be visual.

Brainstorm, Idea Listing and Grouping



2

Brainstorm

Write down any ideas that come to mind
that address your problem statement.

10 minutes

Person 1

Create an engaging and interactive user interface with easy symptom input and visually appealing disease predictions.

Explore the development of a mobile application for convenient, on-the-go access to disease predictions, enhancing user convenience

Person 2

Consider integrating with popular health trackers or apps to utilize existing health data for more accurate predictions.

Implement a feature that shows localized trends of prevalent symptoms, helping users understand regional health patterns

Person 3

Introduce an option for users to share anonymous health stories, creating a supportive community and providing real-world context for predictions.

Include informative pop-ups within the application to educate users about the importance of early disease detection and the role of the predictive model.

Person 4

Explore partnerships with telemedicine services to provide users with the option for immediate online consultations based on predictive outcomes.

Explore partnerships with telemedicine services to provide users with the option for immediate online consultations based on predictive outcomes.

3

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

20 minutes

Group Discussion:

In our brainstorming session, the group generated a variety of ideas to enhance the Disease Prediction Using Machine Learning project. Each member contributed valuable insights to improve user experience, accuracy, and accessibility.

Create an app with a design that is easy to use, visually appealing, and caters to a diverse user base.

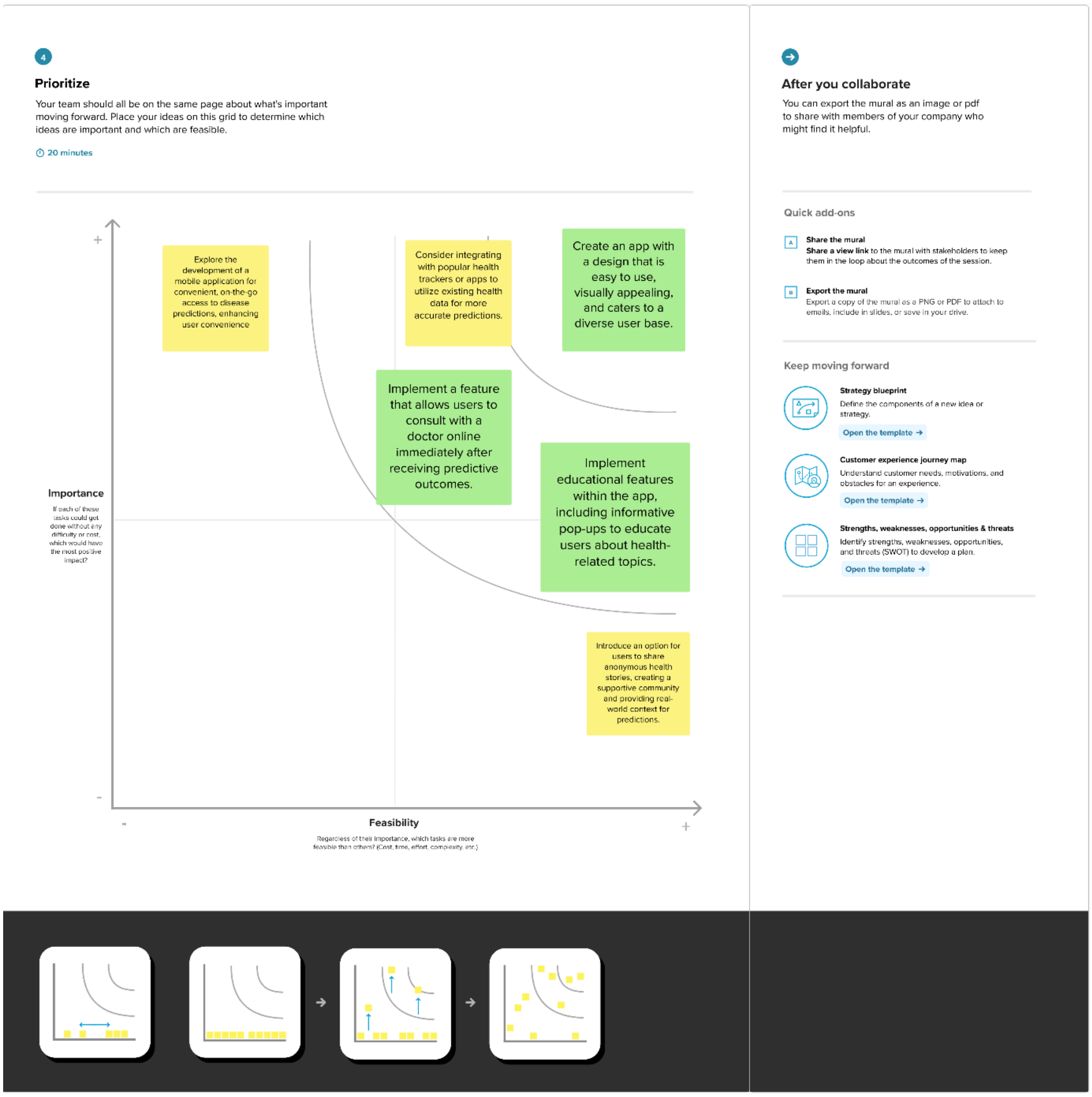
Explore collaborations with popular health apps to integrate their data, enhancing the accuracy of disease predictions.

Prioritize multi-language support to ensure the app is accessible to users globally.

Implement a feature that allows users to consult with a doctor online immediately after receiving predictive outcomes.

Implement educational features within the app, including informative pop-ups to educate users about health-related topics.

Idea Prioritization



4

Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes

Explore the development of a mobile application for convenient, on-the-go access to disease predictions, enhancing user convenience

Consider integrating with popular health trackers or apps to utilize existing health data for more accurate predictions.

Create an app with a design that is easy to use, visually appealing, and caters to a diverse user base.

Implement a feature that allows users to consult with a doctor online immediately after receiving predictive outcomes.

Implement educational features within the app, including informative pop-ups to educate users about health-related topics.

Introduce an option for users to share anonymous health stories, creating a supportive community and providing real-world context for predictions.

Importance
If each of these tasks could get done without any difficulty or cost, which would have the most positive impact?
Feasibility
Regardless of their importance, which tasks are more feasible than others? (Cost, time, effort, complexity, etc.)
+
-
-
+

After you collaborate

You can export the mural as an image or pdf to share with members of your company who might find it helpful.

Quick add-ons

A **Share the mural**
Share a view link to the mural with stakeholders to keep them in the loop about the outcomes of the session.

B **Export the mural**
Export a copy of the mural as a PNG or PDF to attach to emails, include in slides, or save in your drive.

Keep moving forward

Strategy blueprint
Define the components of a new idea or strategy.
Open the template →

Customer experience journey map
Understand customer needs, motivations, and obstacles for an experience.
Open the template →

Strengths, weaknesses, opportunities & threats
Identify strengths, weaknesses, opportunities, and threats (SWOT) to develop a plan.
Open the template →

4. REQUIREMENT ANALYSIS

4.1 Functional Requirements

To effectively address the identified challenges and fulfil the objectives of the project, the following functional requirements have been defined:

1. Symptom Input Interface:

- Description: Users should be able to input their symptoms through an intuitive and user-friendly interface.

- Rationale: A seamless and straightforward symptom input process ensures user engagement and accurate data collection.

2. Disease Forecasting Model:

- Description: Implement a robust disease forecasting model capable of analyzing input symptoms and predicting up to 42 diseases.

- Rationale: The core functionality of the project, this feature provides users and clinicians with valuable insights for informed decision-making.

3. Privacy-Respecting Design:

- Description: Ensure that the model does not collect any personalized information, including name, age, gender, residence, or religion.

- Rationale: Upholding user privacy is crucial for building trust and encouraging widespread adoption of the tool.

4. Decision Support System:

- Description: Provide clear and understandable recommendations based on the model's output, guiding users on whether to seek professional medical advice.

- Rationale: A user-friendly decision support system enhances the tool's utility for individuals making health-related decisions.

5. Accessibility and Availability:

- Description: Ensure the tool is accessible at any time, allowing users to perform self-assessments whenever needed.

- Rationale: Aligning with the project's purpose, accessibility supports users in managing their health conveniently.

6. Clinician Consultation Integration:

- Description: Facilitate integration with online platforms to enable clinicians to use the model's output for online consultations.

- Rationale: Enhances the tool's utility for healthcare professionals, fostering a collaborative approach to healthcare.

7. Cross-Platform Compatibility:

- Description: Develop the tool to be compatible with various devices and platforms, including web browsers and mobile applications.

- Rationale: Maximizes accessibility and usability, catering to a diverse user base.

8. User Education and Information:

- Description: Provide relevant information to users about the limitations of the tool, the importance of professional medical advice, and resources for further health education.

- Rationale: Promotes responsible use of the tool and encourages users to seek professional guidance when necessary.

9. System Alerts and Notifications:

- Description: Implement a notification system to remind users to consult a doctor if their symptoms persist or worsen.

- Rationale: Supports proactive health management and reinforces the importance of seeking professional medical attention.

These functional requirements collectively define the features and capabilities necessary to create a comprehensive and effective tool for symptom-based disease forecasting and decision support.

4.2 Non-Functional Requirements

In addition to the functional aspects, the project has defined non-functional requirements that focus on the performance, security, and usability of the tool:

1. Performance:

- Requirement: The system should respond to user inputs and generate disease forecasts within a maximum of 5 seconds.

- Rationale: Ensures a responsive and efficient user experience, minimizing wait times for results.

2. Security:

- Requirement: Implement robust security measures to protect user data and ensure the confidentiality of health information.

- Rationale: Safeguarding user privacy is paramount, fostering trust in the tool and encouraging users to provide accurate symptom information.

3. Scalability:

- Requirement: Design the system to handle a scalable number of users, ensuring performance remains consistent as the user base grows.

- Rationale: Accommodates potential increases in usage, especially during periods of heightened health concerns.

4. Usability:

- Requirement: Conduct user testing to ensure the tool's interface is intuitive and accessible to users with varying levels of technological proficiency.

- Rationale: Enhances user satisfaction and encourages regular usage of the tool.

5. Reliability:

- Requirement: The system should have a reliability rate of 99.9%, minimizing the risk of downtime or disruptions.

- Rationale: Ensures users can rely on the tool for accurate and timely health insights.

6. Compatibility:

   - Requirement: Ensure compatibility with a range of web browsers and mobile devices to cater to diverse user preferences.

   - Rationale: Maximizes the tool's accessibility and usability across different platforms.

7. Compliance with Healthcare Regulations:

   - Requirement: Adhere to relevant healthcare data protection and privacy regulations, ensuring legal compliance.

   - Rationale: Mitigates legal risks and demonstrates commitment to ethical data handling.

8. User Training and Support:

   - Requirement: Provide comprehensive user training materials and support resources to assist users in understanding and using the tool effectively.

   - Rationale: Supports user adoption and ensures users are equipped to make informed decisions based on the tool's output.

9. Data Backup and Recovery:

   - Requirement: Implement regular data backup procedures and a robust recovery mechanism to prevent data loss in case of system failures.

   - Rationale: Safeguards against the loss of critical user data and maintains the integrity of the tool.

10. Cross-Browser and Cross-Device Testing:

   - Requirement: Conduct thorough testing to ensure the tool functions consistently across various web browsers and devices.

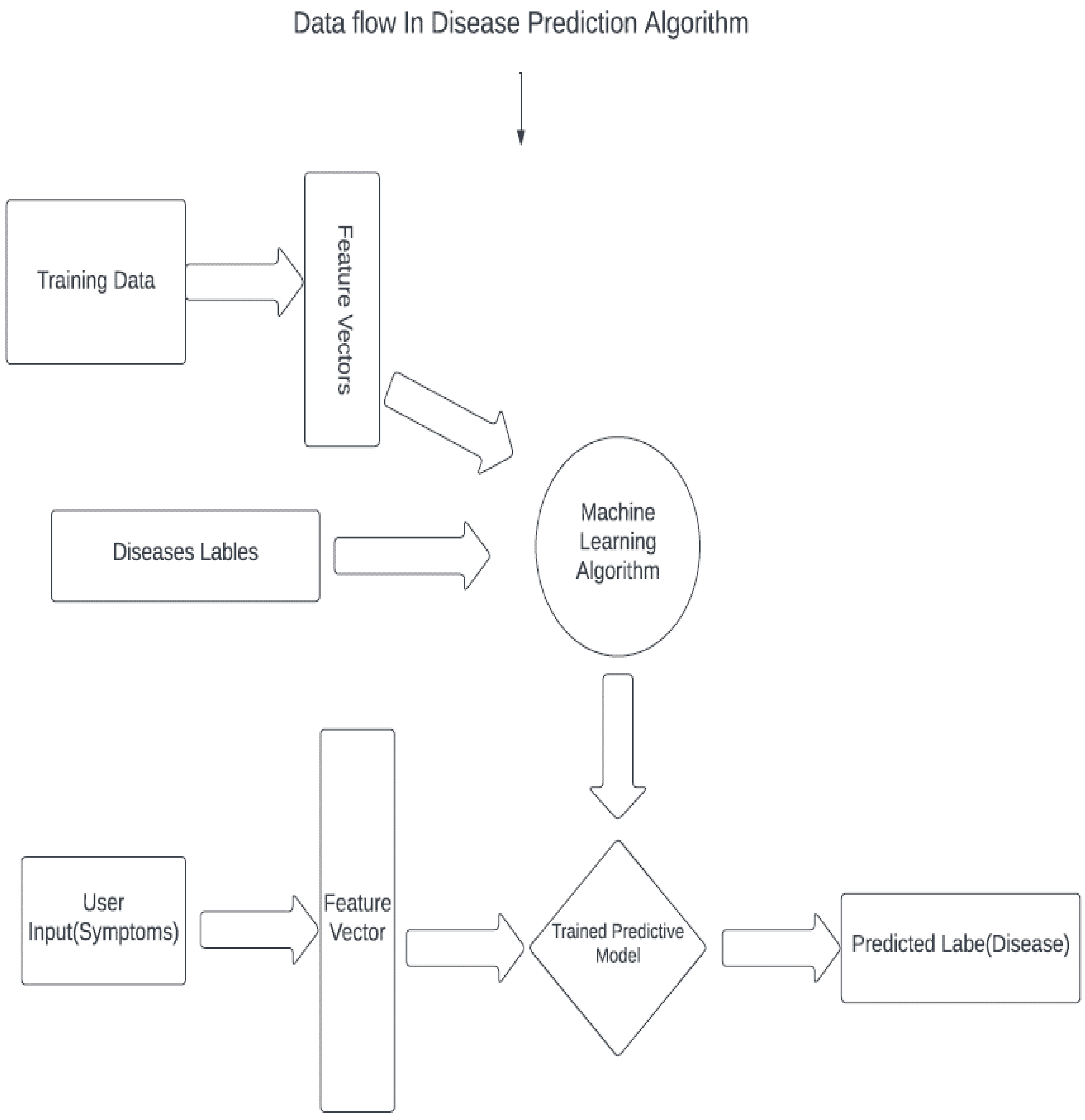
   - Rationale: Guarantees a seamless user experience regardless of the chosen platform.

These non-functional requirements complement the functional features, collectively contributing to the creation of a reliable, secure, and user-friendly tool for symptom-based disease forecasting.

5. Project Design

5.1 Data Flow Diagrams & User Stories

5.1.1 Data Flow Diagram:



1. Users enter their symptoms through the user interface.
2. Validate and sanitize user inputs to ensure they meet required standards.
3. Send the validated symptoms data to the backend system.
4. Preprocess the input data to handle missing values or any formatting issues.
5. Extract relevant features (symptoms) from the pre-processed data.
6. Input the extracted features into the trained machine learning model for disease prediction.
7. Receive the predicted disease output from the machine learning model.
8. Show the predicted disease to the user through the interface.

9. Log user inputs, predictions, and system events for monitoring and analysis.
10. Collect feedback from users to improve the model and system.

5.1.2 User Stories

User Type	**Functional Requirement (Epic)**	**User Story Number**	**User Story / Task**	**Acceptance criteria**	**Priority**	**Release**
Patient (Personal user)	Diagnosis	USN-1	As a patient, I want to be able to enter my symptoms into the disease prediction system so that it can predict the diseases I may have.	I can enter my symptoms into the system and receive a list of predicted diseases.	High	Sprint-1
Doctor (Commercial User)	Diagnosis	USN-2	As a doctor, I want to be able to use the disease prediction system to help me diagnose my patients.	I can enter my patient's symptoms into the system and receive a list of predicted diseases.	High	Sprint-1
Researcher	Research	USN-3	As a researcher, I want to be able to use the disease prediction system to conduct research on new diseases and treatments	I can access the system's data and machine learning models to conduct research.	Low	Sprint-3
Administrator	System Management	USN-4	As an administrator, I want to be able to manage the disease prediction system, including adding and removing users, managing permissions, and monitoring system performance.	I can add and remove users, manage permissions, and monitor system performance	High	Sprint-1

5.2 Solution Architecture

Solution Architecture:

Solution architecture for a "Disease Prediction Using Machine Learning" project involves several components. Here's a high-level overview of the architecture:

1. Data Collection and Preprocessing:

- Collect relevant medical data containing symptoms, patient history, and disease labels.
- Preprocess the data to handle missing values, normalize features, and ensure data quality.

2. Machine Learning Model Development:

- Choose appropriate machine learning algorithms for disease prediction.
- Split the dataset into training and testing sets for model training and evaluation.

- Train the machine learning model using the training data.

3. Feature Engineering:

- Identify and select relevant features (symptoms) that contribute to disease prediction.

- Apply any necessary transformations or feature engineering techniques.

4. Model Evaluation:

- Evaluate the performance of the machine learning model using the testing dataset.

- Use metrics such as accuracy, precision, recall, and F1-score to assess the model's effectiveness.

5. Integration with User Interface:

- Develop a user interface for inputting symptoms and receiving predictions.

- Integrate the trained machine learning model into the user interface to enable real-time predictions.

6. Security and Privacy:

- Implement security measures to protect sensitive health data.

- Ensure compliance with privacy regulations, such as HIPAA (Health Insurance Portability and Accountability Act) if applicable.

7. Scalability and Performance:

- Design the system to handle a scalable number of users and data inputs.

- Optimize the machine learning model for performance, considering response time and resource utilization.

8. Logging and Monitoring:

- Implement logging mechanisms to track user inputs, predictions, and system behaviour.

- Set up monitoring tools to detect and address any issues promptly.

9. Documentation:

- Create comprehensive documentation for the solution architecture, including data sources, model details, and system components.

10. Deployment and Maintenance:

- Deploy the solution to a production environment.

- Establish a maintenance plan for regular updates, model retraining, and addressing potential issues.

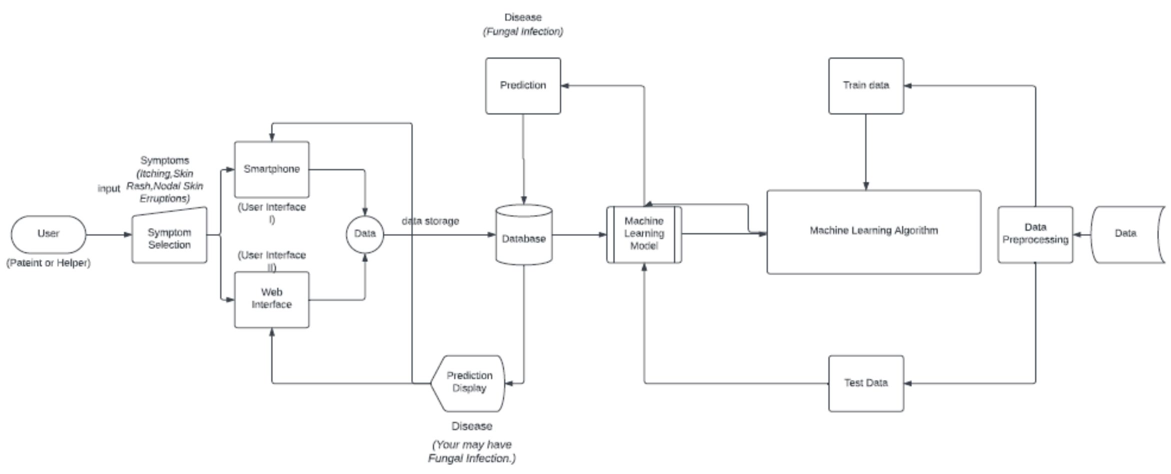
11. Collaboration with Healthcare Professionals:

- Collaborate with healthcare professionals to ensure the accuracy and relevance of the model predictions.

- Incorporate feedback from medical experts to continuously improve the model

Solution Architecture Diagram:

Solution Architecture for Disease Prediction Machine Learning Model



6. PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture

Technical Architecture:

Technology Stack

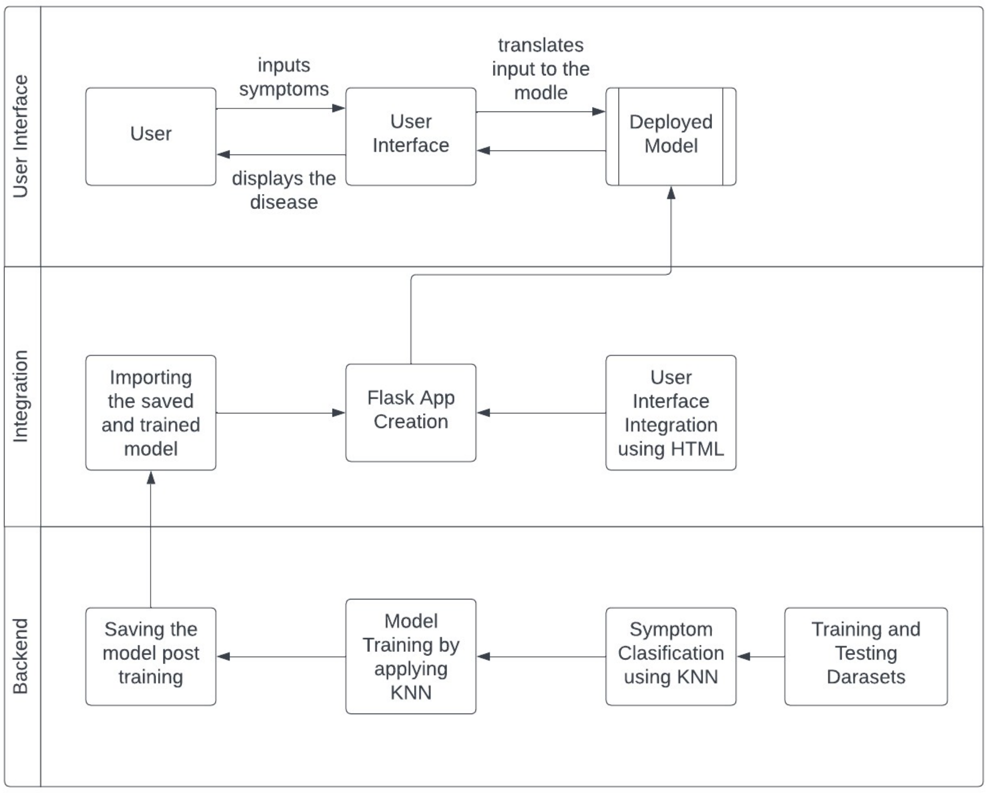


Table-1: Components & Technologies:

S.No	Component	Description	Technology
1.	User Interface	How user interacts with application e.g. Web UI	HTML
2.	Application Logic-1	Logic for a process in the application	Python
3.	Database	Collect the Dataset Based on the Problem Statement	File Manager
4.	File Storage/ Data	File storage requirements for Storing the dataset	Local System
5.	Frame Work	Used to Create a web Application, Integrating Frontend and Back End	Python, Flask
6.	Deep Learning Model	Purpose of Model	KNN,Transfer Learning
7.	Infrastructure (Server / Cloud)	Application Deployment on Local System / Cloud Local Server Configuration: File Explorer Windows	Local, Cloud Foundry, Kubernetes, etc.

Table-2: Application Characteristics:

S.No	Characteristics	Description	Technology
1.	Open-Source Frameworks	List the open-source frameworks used	Python's Flask
2.	Security Implementations	List all the security / access controls implemented, use of firewalls etc.	SHA-256
3.	Scalable Architecture	Justify the scalability of architecture (3 – tier, Microservices)	Kubernetes. Microservices architecture
4.	Availability	Justify the availability of application (e.g. use of load balancers, distributed servers etc.)	Load balancing
5.	Performance	Design consideration for the performance of the application (number of requests per sec, use of Cache, use of CDN's) etc.	Content Delivery Networks (CDNs)

6.2 Sprint Planning and Estimation

User Story Number	Functional Requirement (Epic)	User Story / Task	Story Points	Priority	Sprint	Team Members
USN-1	Project Setup & Infrastructure	Set up the development environment with the required tools and frameworks to start the disease prediction project	1	High	Sprint 1	Harshit
USN-2	Development Environment	Gather a diverse dataset symptoms and diseases for training the machine learning model.	2	High	Sprint 1	Harshit
USN-3	Data Collection	Preprocess the collected symptoms and disease dataset by standardizing symptom sizes, enhancing symptom quality, and splitting it into training and validation sets.	2	High	Sprint 2	Uday
USN-4	Data Preprocessing	Investigate and assess different machine learning methodologies to identify the optimal model for disease prediction based on symptom data.	3	High	Sprint 2	Uday
USN-5	Model Development	Train the chosen machine learning model on the pre-processed data and evaluate its performance on the validation set.	4	High	Sprint 3	Uday

USN-6	Training	Incorporate data augmentation techniques (such as rotation and flipping) to improve the model's capacity to detect disease patterns according to symptoms.	6	Medium	Sprint 3	Harshit
USN-7	Model Deployment & Integration	Deploy the trained machine learning model as an API or web service to enable disease prediction using symptoms. Integrate the model's API into a user-friendly web interface where users can submit symptoms for analysis.	1	Medium	Sprint 4	Harshit
USN-8	Testing & Quality Assurance	Rigorously test the model and web interface to uncover and report any problems or inaccuracies. Refine the model's hyperparameters and enhance its disease prediction accuracy based on user feedback and testing outcomes.	1	Medium	Sprint 5	Uday

6.3 Sprint Delivery Schedule

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
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
Sprint - 5	1	2 days	14 Nov 2023	16 Nov 2023	20	16 Nov 2023

7. CODING & SOLUTIONING (Explain the features added in the project along with code)

Activity 1: Collect the dataset

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

In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

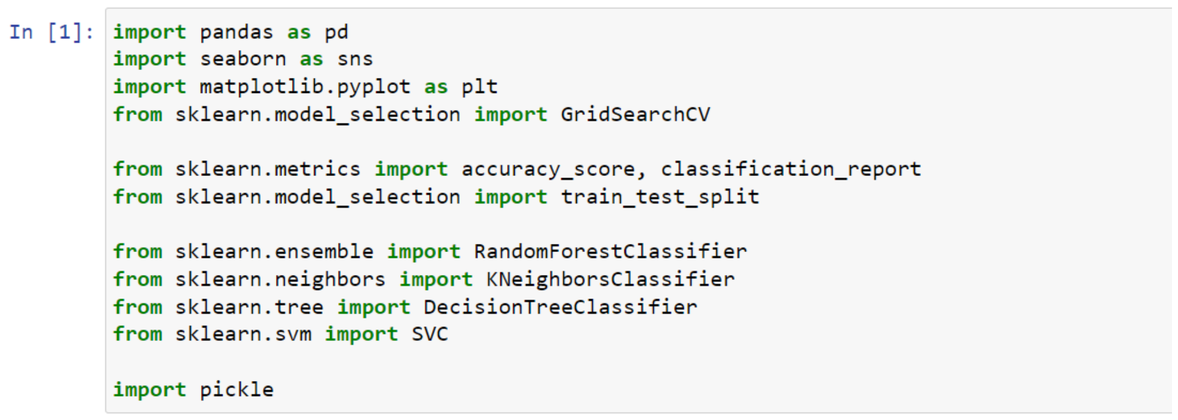
Link: https://www.kaggle.com/datasets/kaushil268/disease-prediction-using-machine-learning

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 1.1: Importing the Libraries

Import the necessary libraries as shown in the image. Some of them are optional and can be skipped according to your usage.

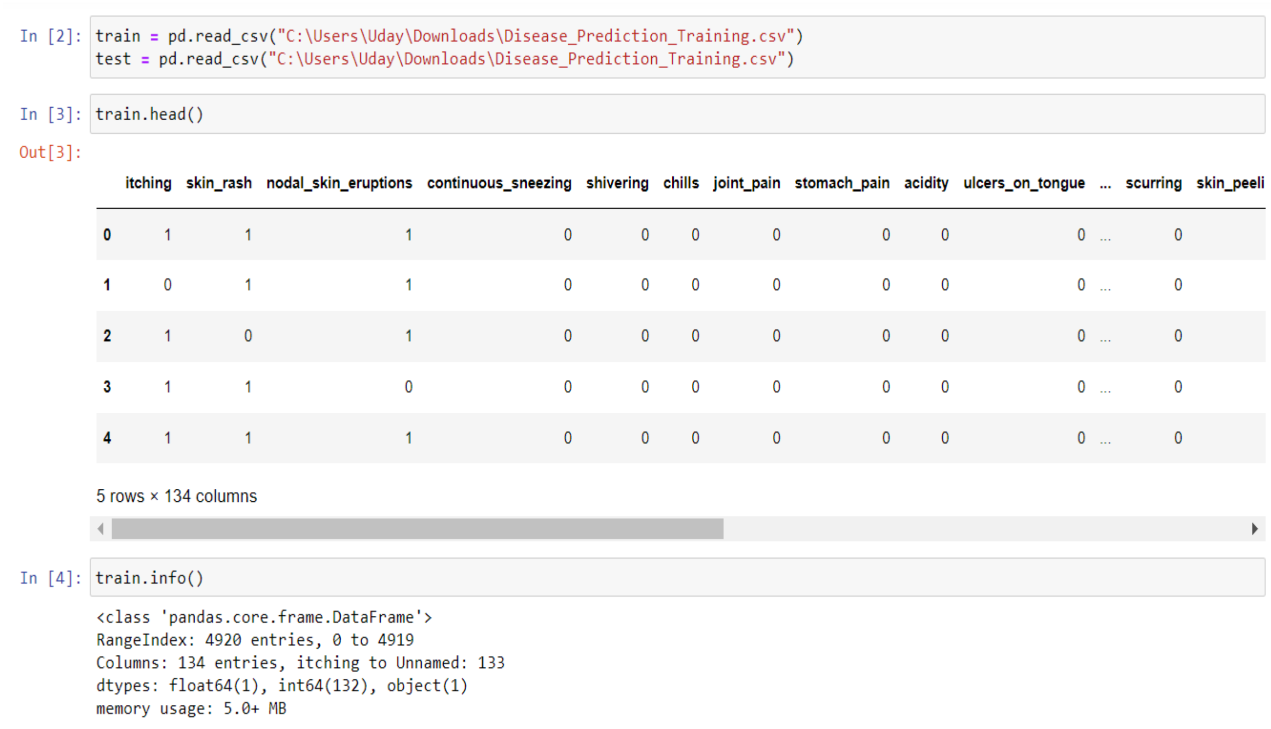


```
In [1]: import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plt
        from sklearn.model_selection import GridSearchCV

        from sklearn.metrics import accuracy_score, classification_report
        from sklearn.model_selection import train_test_split

        from sklearn.ensemble import RandomForestClassifier
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.svm import SVC

        import pickle
```

Activity 1.2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.



```
In [2]: train = pd.read_csv("C:\Users\Uday\Downloads\Disease_Prediction_Training.csv")
        test = pd.read_csv("C:\Users\Uday\Downloads\Disease_Prediction_Training.csv")

In [3]: train.head()

Out[3]:
```

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity	ulcers_on_tongue	...	scurring	skin_peeli
0	1	1	1	0	0	0	0	0	0	0	...	0	
1	0	1	1	0	0	0	0	0	0	0	...	0	
2	1	0	1	0	0	0	0	0	0	0	...	0	
3	1	1	0	0	0	0	0	0	0	0	...	0	
4	1	1	1	0	0	0	0	0	0	0	...	0	

5 rows × 134 columns

```
In [4]: train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4920 entries, 0 to 4919
Columns: 134 entries, itching to Unnamed: 133
dtypes: float64(1), int64(132), object(1)
memory usage: 5.0+ MB
```

As we have two datasets, one for training and other for testing we will import both the csv files.

Activity 2: Data Preparation

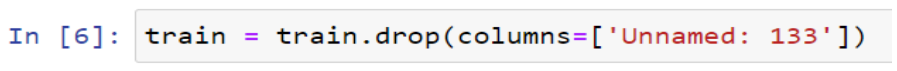
As we have understood how the data is, let us preprocess the collected data.

The Machine Learning model cannot be trained on the imported data directly. The dataset might have randomness, we might have to clean the dataset and bring it in the right form. This activity involves the following steps:

- Removing Redundant Columns
- Handling Missing Values

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

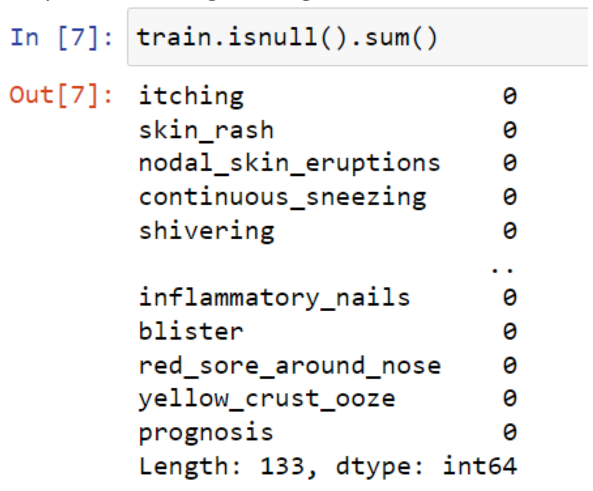
Activity 2.1: Removing Redundant Columns



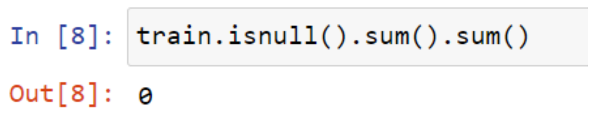
```
In [6]: train = train.drop(columns=['Unnamed: 133'])
```

Unnamed: 133 is the redundant column which does not have any values.

Activity 2.2: Handling Missing Values



```
In [7]: train.isnull().sum()

Out[7]: itching                  0
        skin_rash                0
        nodal_skin_eruptions     0
        continuous_sneezing      0
        shivering                0
                                ..
        inflammatory_nails       0
        blister                  0
        red_sore_around_nose     0
        yellow_crust_ooze        0
        prognosis                0
        Length: 133, dtype: int64
```



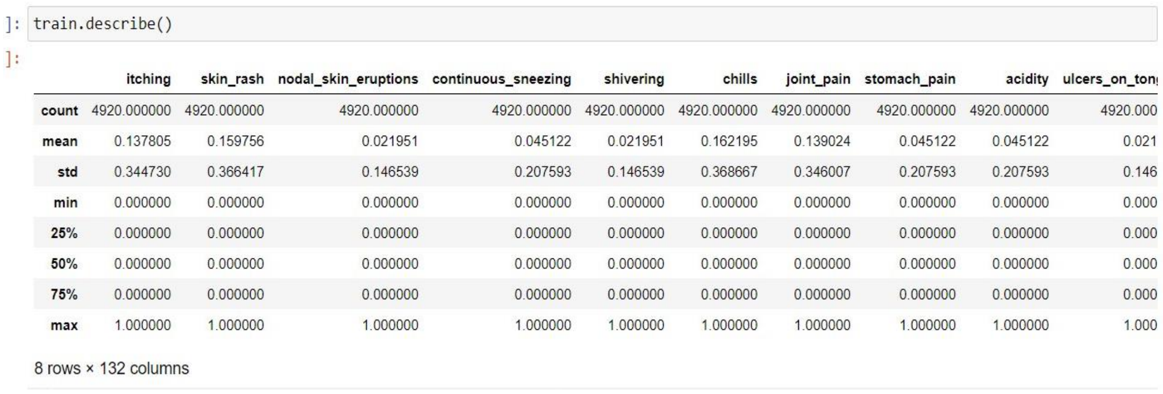
```
In [8]: train.isnull().sum().sum()

Out[8]: 0
```

There are nomissing values in the dataset. That is why we can skip this step.

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive Statistical

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



```
]: train.describe()
]:
```

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity	ulcers_on_ton
count	4920.000000	4920.000000	4920.000000	4920.000000	4920.000000	4920.000000	4920.000000	4920.000000	4920.000000	4920.000
mean	0.137805	0.159756	0.021951	0.045122	0.021951	0.162195	0.139024	0.045122	0.045122	0.021
std	0.344730	0.366417	0.146539	0.207593	0.146539	0.368667	0.346007	0.207593	0.207593	0.146
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000

8 rows × 132 columns

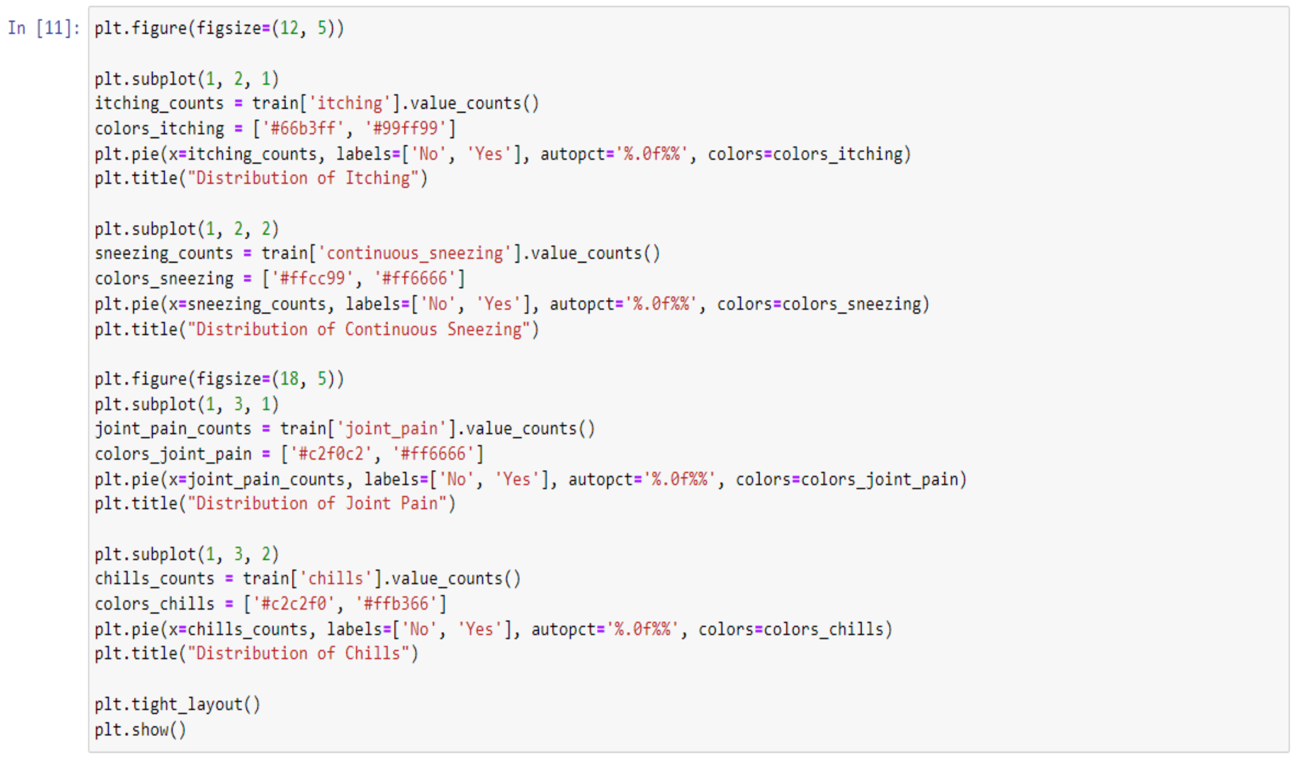
Activity 2: Visual Analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

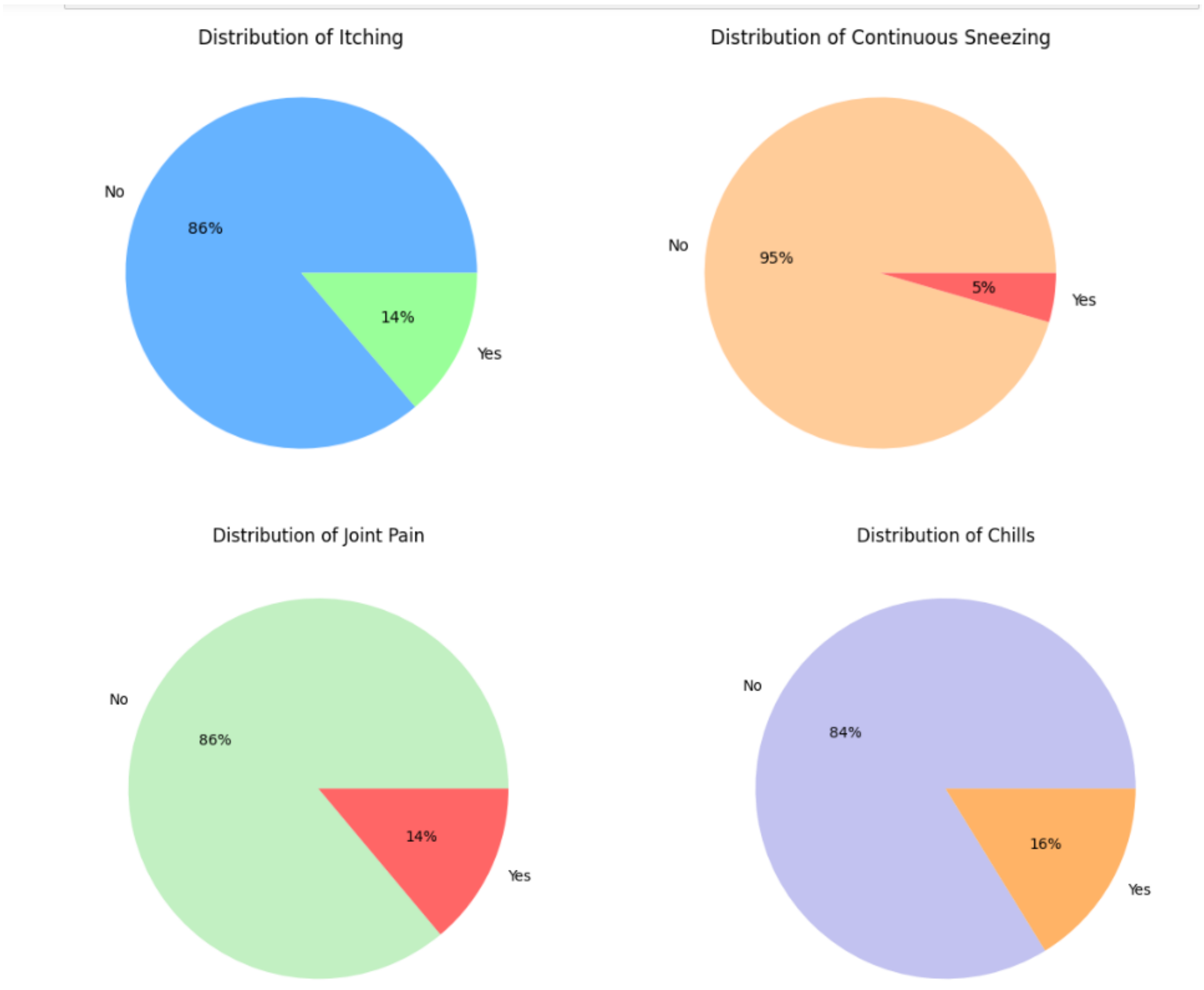
Activity 2.1: Univariate Analysis:

In simple words, univariate analysis is understanding the data with a single feature. We have displayed three different types of graphs and plots.

For simple visualizations we can use the matplotlib.pyplot library.



```
In [11]: plt.figure(figsize=(12, 5))

         plt.subplot(1, 2, 1)
         itching_counts = train['itching'].value_counts()
         colors_itching = ['#66b3ff', '#99ff99']
         plt.pie(x=itching_counts, labels=['No', 'Yes'], autopct='%.0f%%', colors=colors_itching)
         plt.title("Distribution of Itching")

         plt.subplot(1, 2, 2)
         sneezing_counts = train['continuous_sneezing'].value_counts()
         colors_sneezing = ['#ffcc99', '#ff6666']
         plt.pie(x=sneezing_counts, labels=['No', 'Yes'], autopct='%.0f%%', colors=colors_sneezing)
         plt.title("Distribution of Continuous Sneezing")

         plt.figure(figsize=(18, 5))
         plt.subplot(1, 3, 1)
         joint_pain_counts = train['joint_pain'].value_counts()
         colors_joint_pain = ['#c2f0c2', '#ff6666']
         plt.pie(x=joint_pain_counts, labels=['No', 'Yes'], autopct='%.0f%%', colors=colors_joint_pain)
         plt.title("Distribution of Joint Pain")

         plt.subplot(1, 3, 2)
         chills_counts = train['chills'].value_counts()
         colors_chills = ['#c2c2f0', '#ffb366']
         plt.pie(x=chills_counts, labels=['No', 'Yes'], autopct='%.0f%%', colors=colors_chills)
         plt.title("Distribution of Chills")

         plt.tight_layout()
         plt.show()
```

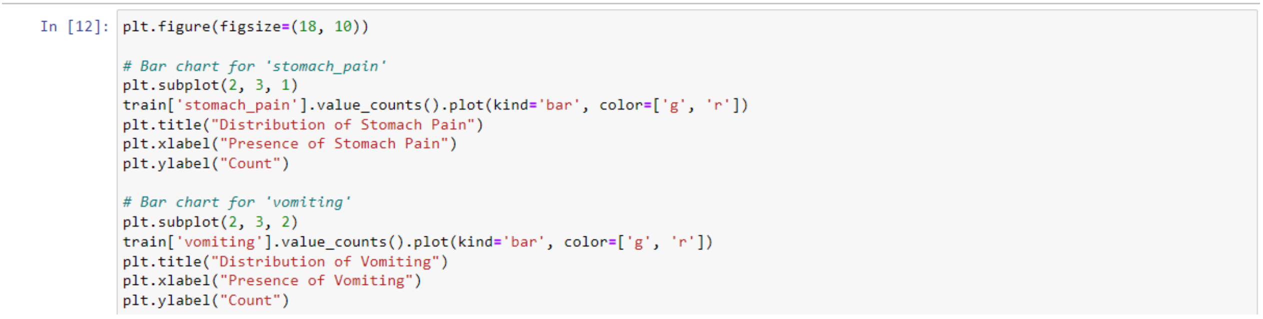


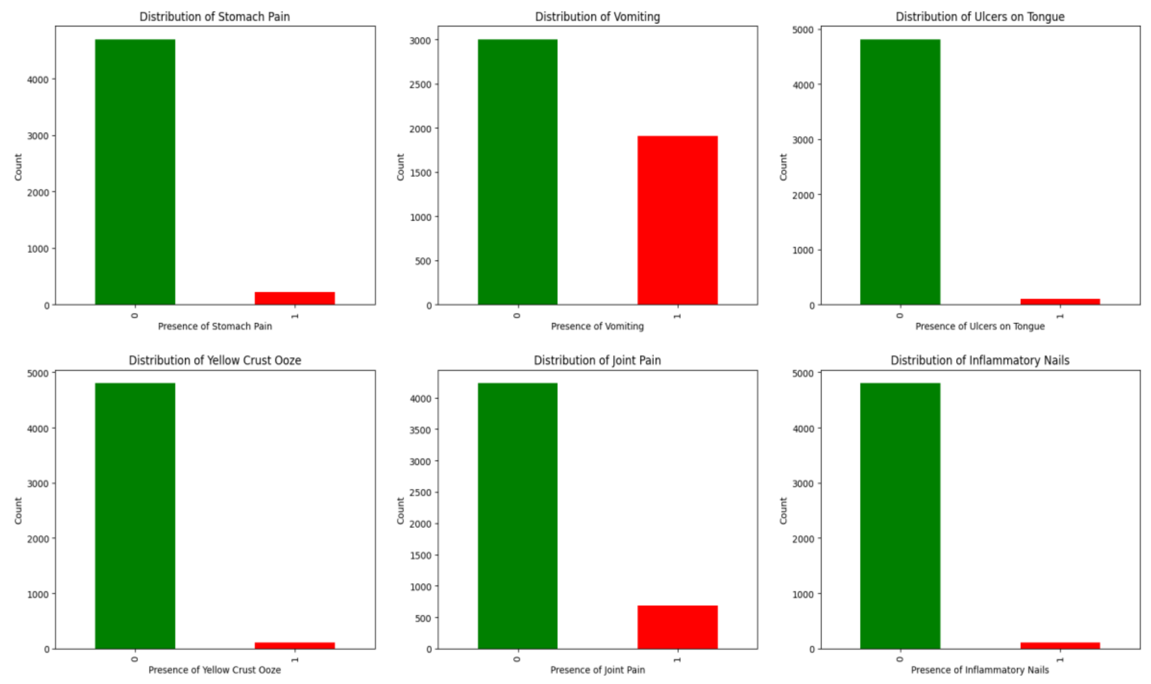
Here the plt.figure() command is used to determine the size of the plot. Then we divide the space for 4 pie plots.

The pie plot on the left shows the different values distribution in the Itching column. It shows that there are 86% observations where the itching symptom has value 0 and there are 14% observations where the itching symptom has value 1.

The pie plot on the right shows the different values distribution in the continuous_sneezing column. It shows that there are 95% observations where the continuous_sneezing symptom has value 0 and there are 5% observations where the continuous_sneezing symptom has value

1.


```
In [12]: plt.figure(figsize=(18, 10))

         # Bar chart for 'stomach_pain'
         plt.subplot(2, 3, 1)
         train['stomach_pain'].value_counts().plot(kind='bar', color=['g', 'r'])
         plt.title("Distribution of Stomach Pain")
         plt.xlabel("Presence of Stomach Pain")
         plt.ylabel("Count")

         # Bar chart for 'vomiting'
         plt.subplot(2, 3, 2)
         train['vomiting'].value_counts().plot(kind='bar', color=['g', 'r'])
         plt.title("Distribution of Vomiting")
         plt.xlabel("Presence of Vomiting")
         plt.ylabel("Count")
```



Here we have plotted 2 bar graphs. These bar graphs can be plotted without using any external library. We divide the plot into two subplots using subplot function from the matplotlib. pyplot library. We have plotted the bar graph using the inbuilt plot function in python.

The bar graph on the left shows the distribution of stomach pain symptom values. We can see that the 0 value has count of around 4700 and the 1 value has count of around 400.

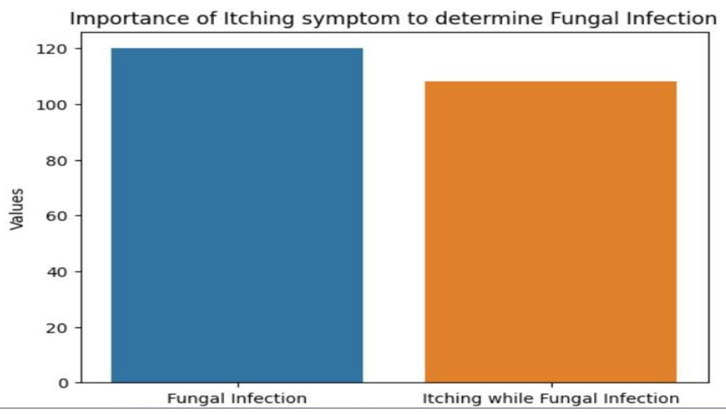
The graph on the right shows the distribution of vomiting symptom values. We can see that the 0 value has count of around 3000 and the 1 value has count of around 2000.

Activity 2.2: Bivariate Analysis:

To find the relation between two features we use bivariate analysis. Here we are visualising the relationship between prognosis where the values are Fungal Infection and Itching symptom.

```
]: a = len(train[train['prognosis'] == 'Fungal infection'])
   b = len(train[(train['itching'] == 1) & (train['prognosis'] == 'Fungal infection')])
   fi = pd.DataFrame(data = [a,b], columns=['Values'],index = ['Fungal Infection','Itching while Fungal Infection'])

   sns.barplot(data = fi, x = fi.index, y = fi['Values'])
   plt.title('Importance of Itching symptom to determine Fungal Infection')
```

```
]: Text(0.5, 1.0, 'Importance of Itching symptom to determine Fungal Infection')
```



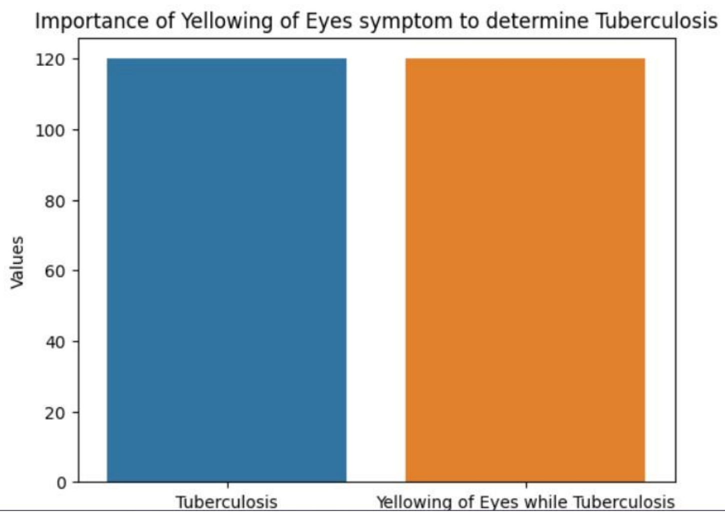
Here we use Boolean Indexing to filter out values from the prognosis column where the values are 'Fungal Infection'. These observations are stored in variable 'a'. Alse we filter out values from the prognosis where values are ‘Fungal Infection’ and also the values of Itching variable are 1. From the plot we can see that when there is Fungal Infection there is a high chance that the Itching column has value 1. There are 120 values where the value are fungal infection and there are 104 values where the value of itching column is 1. This shows that when there is a fungal infection there is a high chance that there is itching as a symptom.

Next we have also seen the relationship between prognosis when the disease is Tuberculosis and the symptom yellowing_of_eyes. . From the plot we can see that when there is

Tuberculosis there is a high chance that the yellowing of eyes column has value 1. There are 120 values where the value is Tuberculosis and there are 119 values where the value of yellowing_of_eyes column is 1. This shows that when there is tuberculosis there is a high chance that there is yellowing_of_eyes as a symptom.

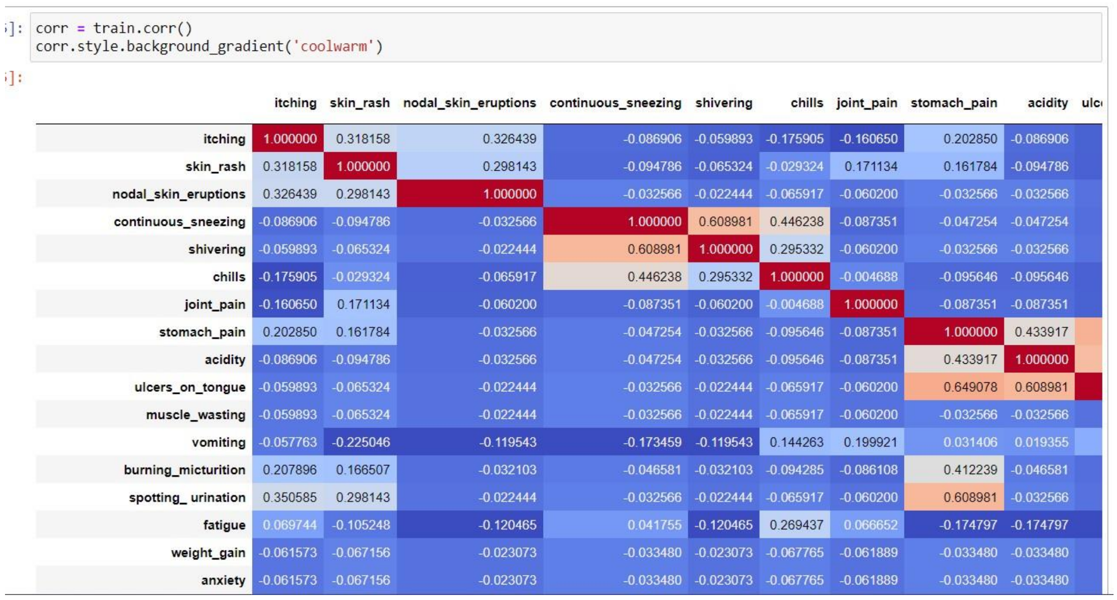
```
In [28]:
a = len(train[train['prognosis'] == 'Tuberculosis'])
b = len(train[(train['yellowing_of_eyes'] == 1) & (train['prognosis'] == 'Tuberculosis')])
fi = pd.DataFrame(data = [a,b], columns=['Values'],index = ['Tuberculosis','Yellowing of Eyes while Tuberculosis'])

sns.barplot(data = fi, x = fi.index, y = fi['Values'])
plt.title('Importance of Yellowing of Eyes symptom to determine Tuberculosis')
```

```
Out[28]: Text(0.5, 1.0, 'Importance of Yellowing of Eyes symptom to determine Tuberculosis')
```



Activity 2.3: Multivariate Analysis

In multivariate analysis we try to find the relation between multiple features. This can be done primarily with the help of Correlation matrix.



```
]: corr = train.corr()
   corr.style.background_gradient('coolwarm')
```

]:

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity	ulc
itching	1.000000	0.318158	0.326439	-0.086906	-0.059893	-0.175905	-0.160650	0.202850	-0.086906	
skin_rash	0.318158	1.000000	0.298143	-0.094786	-0.065324	-0.029324	0.171134	0.161784	-0.094786	
nodal_skin_eruptions	0.326439	0.298143	1.000000	-0.032566	-0.022444	-0.065917	-0.060200	-0.032566	-0.032566	
continuous_sneezing	-0.086906	-0.094786	-0.032566	1.000000	0.608981	0.446238	-0.087351	-0.047254	-0.047254	
shivering	-0.059893	-0.065324	-0.022444	0.608981	1.000000	0.295332	-0.060200	-0.032566	-0.032566	
chills	-0.175905	-0.029324	-0.065917	0.446238	0.295332	1.000000	-0.004688	-0.095646	-0.095646	
joint_pain	-0.160650	0.171134	-0.060200	-0.087351	-0.060200	-0.004688	1.000000	-0.087351	-0.087351	
stomach_pain	0.202850	0.161784	-0.032566	-0.047254	-0.032566	-0.095646	-0.087351	1.000000	0.433917	
acidity	-0.086906	-0.094786	-0.032566	-0.047254	-0.032566	-0.095646	-0.087351	0.433917	1.000000	
ulcers_on_tongue	-0.059893	-0.065324	-0.022444	-0.032566	-0.022444	-0.065917	-0.060200	0.649078	0.608981	
muscle_wasting	-0.059893	-0.065324	-0.022444	-0.032566	-0.022444	-0.065917	-0.060200	-0.032566	-0.032566	
vomiting	-0.057763	-0.225046	-0.119543	-0.173459	-0.119543	0.144263	0.199921	0.031406	0.019355	
burning_micturition	0.207896	0.166507	-0.032103	-0.046581	-0.032103	-0.094285	-0.086108	0.412239	-0.046581	
spotting_ urination	0.350585	0.298143	-0.022444	-0.032566	-0.022444	-0.065917	-0.060200	0.608981	-0.032566	
fatigue	0.069744	-0.105248	-0.120465	0.041755	-0.120465	0.269437	0.066652	-0.174797	-0.174797	
weight_gain	-0.061573	-0.067156	-0.023073	-0.033480	-0.023073	-0.067765	-0.061889	-0.033480	-0.033480	
anxiety	-0.061573	-0.067156	-0.023073	-0.033480	-0.023073	-0.067765	-0.061889	-0.033480	-0.033480	

As we have 131 columns which have numerical values the correlation matrix is of dimensions 131 X 131. These many features can only be parsed by scrolling.

From the correlation matrix we try to remove the values which are highly correlated with each other. When 2 values are highly correlated with each other, we can only remove one of them. We remove columns where the correlation between the columns is above 0.9

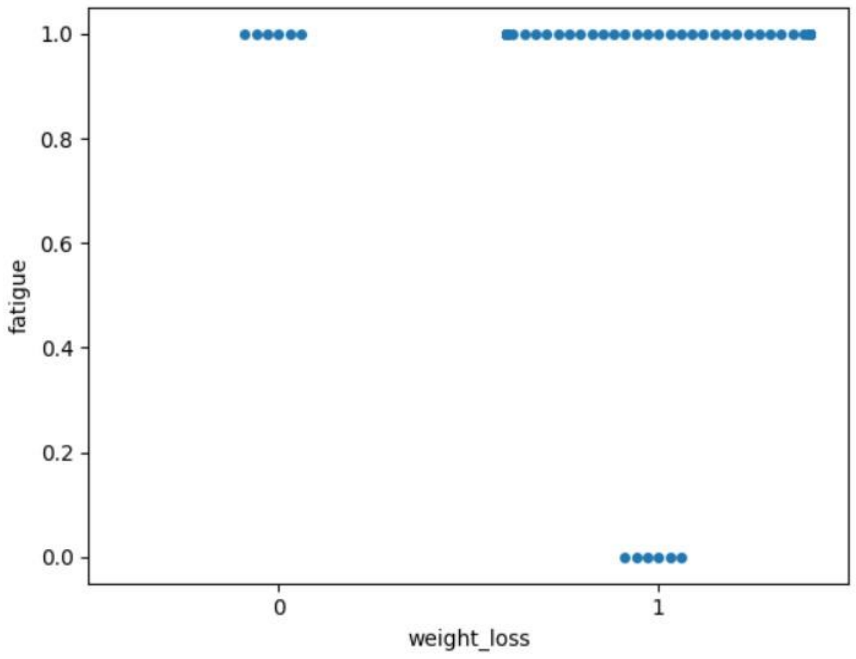
```
In [17]: #This code identifies and removes highly correlated features (columns) from a numeric DataFrame
         correlation_threshold = 0.9

         numeric_df = train.drop('prognosis', axis=1)
         corr_matrix = numeric_df.corr().abs()
         upper_triangle = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
         to_drop = [column for column in upper_triangle.columns if any(upper_triangle[column] > correlation_threshold)]


         df_filtered = numeric_df.drop(to_drop, axis=1)
```

```
In [18]: print("Dropped columns With high correlation:")
         print(to_drop)

         print("\n\nFiltered columns:")
         print(df_filtered.columns.tolist())
```

```
Dropped columns With high correlation:
['cold_hands_and_feets', 'redness_of_eyes', 'sinus_pressure', 'runny_nose', 'congestion', 'pain_in_anal_region', 'bloody_stoo
l', 'irritation_in_anus', 'bruising', 'swollen_legs', 'swollen_blood_vessels', 'puffy_face_and_eyes', 'enlarged_thyroid', 'brit
tle_nails', 'swollen_extremeties', 'drying_and_tingling_lips', 'slurred_speech', 'hip_joint_pain', 'unsteadiness', 'loss_of_sme
ll', 'continuous_feel_of_urine', 'internal_itching', 'altered_sensorium', 'belly_pain', 'abnormal_menstruation', 'increased_app
etite', 'polyuria', 'receiving_blood_transfusion', 'receiving_unsterile_injections', 'coma', 'stomach_bleeding', 'distention_of
_abdomen', 'history_of_alcohol_consumption', 'fluid_overload.1', 'prominent_veins_on_calf', 'palpitations', 'painful_walking',
'silver_like_dusting', 'small_dents_in_nails', 'inflammatory_nails', 'red_sore_around_nose', 'yellow_crust_ooze']
```

We can see the columns that are removed due to high correlation.

For multivariate analysis we will also plot a swarmplot of weightloss and fatigue column where the prognosis column is Tuberculosis.



From this swarm plot we can see that for Tuberculosis disease, there is no observation when the fatigue and weight loss is 0. There are some cases when there is only either of the two, but for Tuberculosis there is a high chance that the patient will have fatigue and weight loss as symptoms.

Preprocessing of Test data

The preprocessing needs to be done for the test data. We can create a function for test data preprocessing which will only leave us with the required features. This function will contain all the steps which we have done for the training data.

```
In [20]: def preprocess_test_data(test, to_drop):
             test.drop(columns=to_drop, inplace=True)
             return test
```

```
In [21]: test_data = preprocess_test_data(test, to_drop)
```

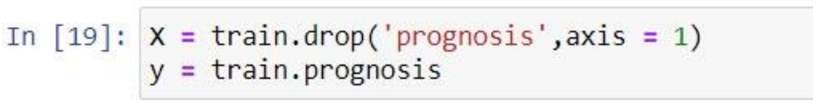
This function drops all the columns which needs to be dropped.

Here we call the function for the test data.

Activity 2.5: Split data into training, validation and testing data

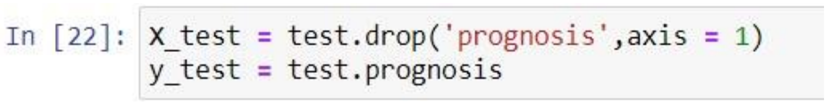
We have training and testing data given separately. We further split the training data into training and validation data. This validation data can be used for hyper parameter tuning.

We first need to separate the features and the target variable. The features are used to predict the target variable.



```
In [19]: X = train.drop('prognosis',axis = 1)
         y = train.prognosis
```

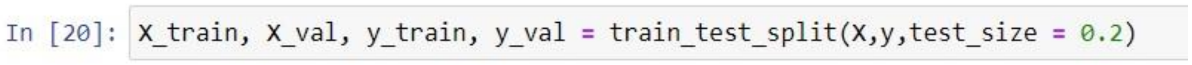
We split the training data into features(X) and target variable(y).



```
In [22]: X_test = test.drop('prognosis',axis = 1)
         y_test = test.prognosis
```

Here we split the test data into features(X_test) and the corresponding target variables(y_test)

Now we need to split the training data into training and validation data. It can be done using the following command.



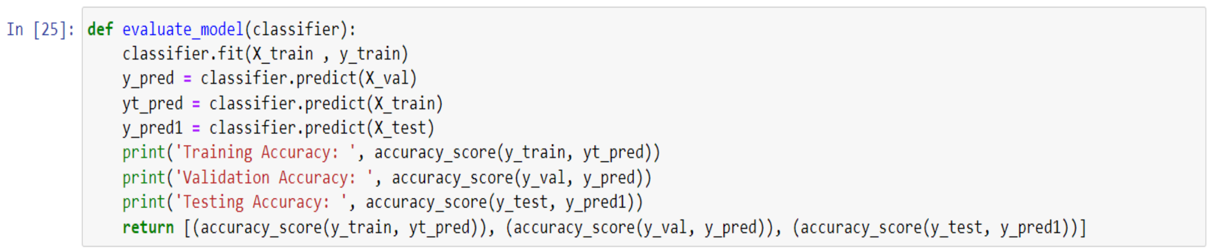
```
In [20]: X_train, X_val, y_train, y_val = train_test_split(X,y,test_size = 0.2)
```

We have kept 80 % data for training and 20% is used for validation.

Milestone 4: Model Building

Activity 1: Creating a function for model evaluation

We will be creating multiple models and then testing them. It will reduce our monotonous task if we directly write a function for model evaluation.



```
In [25]: def evaluate_model(classifier):
             classifier.fit(X_train , y_train)
             y_pred = classifier.predict(X_val)
             yt_pred = classifier.predict(X_train)
             y_pred1 = classifier.predict(X_test)
             print('Training Accuracy: ', accuracy_score(y_train, yt_pred))
             print('Validation Accuracy: ', accuracy_score(y_val, y_pred))
             print('Testing Accuracy: ', accuracy_score(y_test, y_pred1))
             return [(accuracy_score(y_train, yt_pred)), (accuracy_score(y_val, y_pred)), (accuracy_score(y_test, y_pred1))]
```

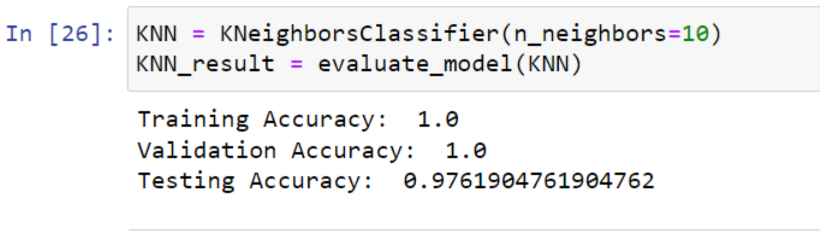
This function will show the accuracies of prediction of model for training, validation and testing data. It will also the return those accuracies.

Activity 2: Training and testing the models using multiple algorithms

Now that we have clean data, a function for evaluation it is time to build models to train the data. For this project we will be using 4 different classification algorithms to build our models. The best model will be used for prediction.

Activity 2.1: K Nearest Neighbors Model

A variable is created with name knn which has KNeighborsClassifier() algorithm initialised in it. The knn model is trained using the .fit() function. The model is trained on the X_train and y_train data that is the training features and training target variables. This model is then given to the model_evaluation function to check its performance.

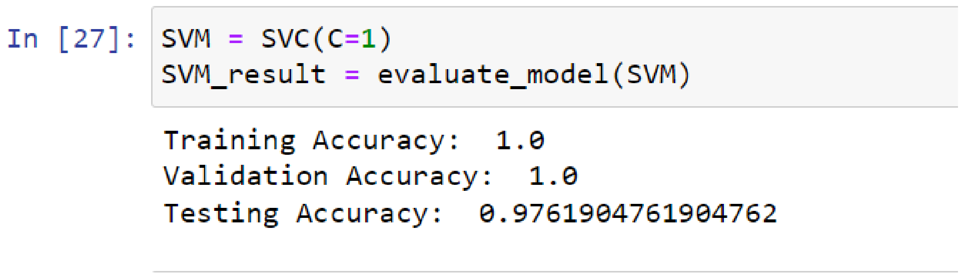


```
In [26]: KNN = KNeighborsClassifier(n_neighbors=10)
         KNN_result = evaluate_model(KNN)

         Training Accuracy:  1.0
         Validation Accuracy:  1.0
         Testing Accuracy:  0.9761904761904762
```

Here we can see that the model has achieved 100% accuracies on training, validation as well as testing data. As the accuracies are high, there is no need for hyperparameter tuning. The results are stored in a variable named knn_results.

Activity 2.2: SVM Model

A variable is created with name svm which has SVC() algorithm initialised in it. The svm model is trained using the .fit() function. The model is trained on the X_train and y_train data that is the training features and training target variables. This model is then given to the model_evaluation function to check its performance.

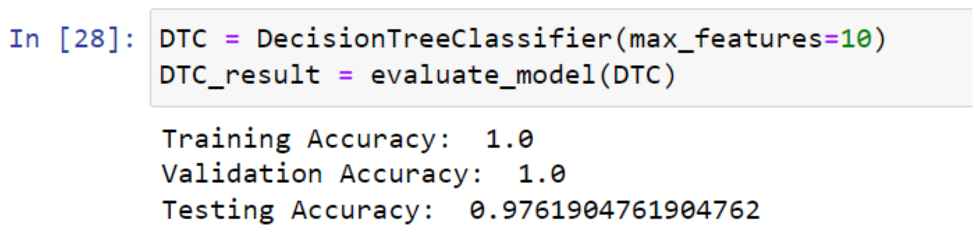


```
In [27]: SVM = SVC(C=1)
         SVM_result = evaluate_model(SVM)

         Training Accuracy:  1.0
         Validation Accuracy:  1.0
         Testing Accuracy:  0.9761904761904762
```

Here we can see that the model has achieved 100% accuracies on training, validation as well as testing data. As the accuracies are high, there is no need for hyperparameter tuning. The results are stored in a variable named svm_results.

Activity 2.3: Decision Tree Model

A variable is created with name dtc which has DecisionTreeClassifier() algorithm initialised in it with a parameter max_features set to 10. The dtc model is trained using the .fit() function. The model is trained on the X_train and y_train data that is the training features and training target variables. This model is then given to the model_evaluation function to check its performance.

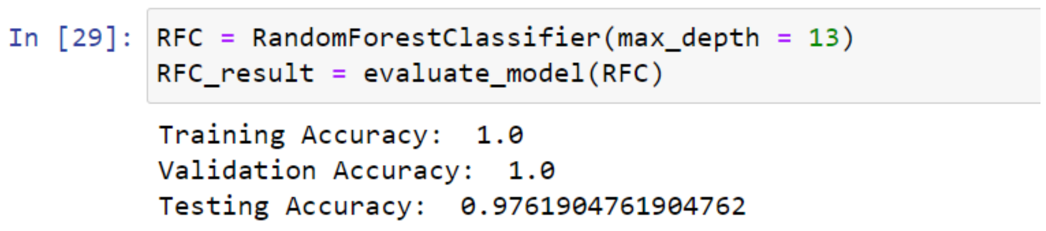


```
In [28]: DTC = DecisionTreeClassifier(max_features=10)
         DTC_result = evaluate_model(DTC)

         Training Accuracy:  1.0
         Validation Accuracy:  1.0
         Testing Accuracy:  0.9761904761904762
```

Here we can see that the model has achieved 100% accuracies on training and validation data . It has achieved 97.6% accuracy for testing data. As the accuracies are high, there is no need for hyperparameter tuning. The results are stored in a variable named dtc_results.

Activity 2.4: Random Forest Model

Random Forest Classifier is a Bagging model which utilises multiple decision trees and takes their aggregate to give a prediction. A variable is created with name rfc which has RandomForestClassifier() algorithm initialised in it with a parameter max_depth set to 13. The rfc model is trained using the .fit() function. The model is trained on the X_train and y_train data that is the training features and training target variables. This model is then given to the model_evaluation function to check its performance.



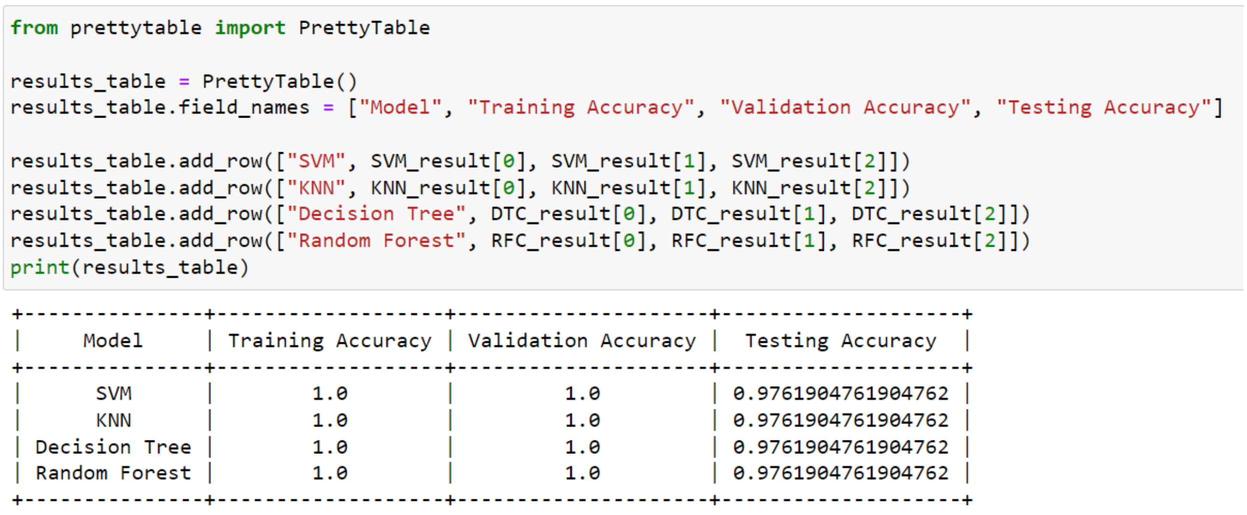
```
In [29]: RFC = RandomForestClassifier(max_depth = 13)
         RFC_result = evaluate_model(RFC)

         Training Accuracy:  1.0
         Validation Accuracy:  1.0
         Testing Accuracy:  0.9761904761904762
```

Here we can see that the model has achieved 100% accuracies on training and validation data . It has achieved 97.6% accuracy for testing data. As the accuracies are high, there is no need for hyperparameter tuning. The results are stored in a variable named rfc_results

Milestone 5 : Performance Testing & Hyperparameter Tuning

Activity 1: Testing model with Multiple Evaluatiton metrics

The data has 41 features, hence it is difficult to make confusion matrix as the dimensions of confusion matrix will be 41 X 41. We can check accuracy to test the model. We have already values of the training, validation, and test accuracies of various models. We can put them in a table and then check for the best model.



```python
from prettytable import PrettyTable

results_table = PrettyTable()
results_table.field_names = ["Model", "Training Accuracy", "Validation Accuracy", "Testing Accuracy"]

results_table.add_row(["SVM", SVM_result[0], SVM_result[1], SVM_result[2]])
results_table.add_row(["KNN", KNN_result[0], KNN_result[1], KNN_result[2]])
results_table.add_row(["Decision Tree", DTC_result[0], DTC_result[1], DTC_result[2]])
results_table.add_row(["Random Forest", RFC_result[0], RFC_result[1], RFC_result[2]])
print(results_table)
```

```
+---------------+-------------------+---------------------+--------------------+
|     Model     | Training Accuracy | Validation Accuracy |  Testing Accuracy  |
+---------------+-------------------+---------------------+--------------------+
|      SVM      |        1.0        |         1.0         | 0.9761904761904762 |
|      KNN      |        1.0        |         1.0         | 0.9761904761904762 |
| Decision Tree |        1.0        |         1.0         | 0.9761904761904762 |
| Random Forest |        1.0        |         1.0         | 0.9761904761904762 |
+---------------+-------------------+---------------------+--------------------+
```

From the table we can see that KNN and SVM models perform the best.

Activity 2: Comparing model accuracy before and after applying

hyperparameter tuning

As the accuracies are already so high, we need not do hyperparameter tuning for the models.

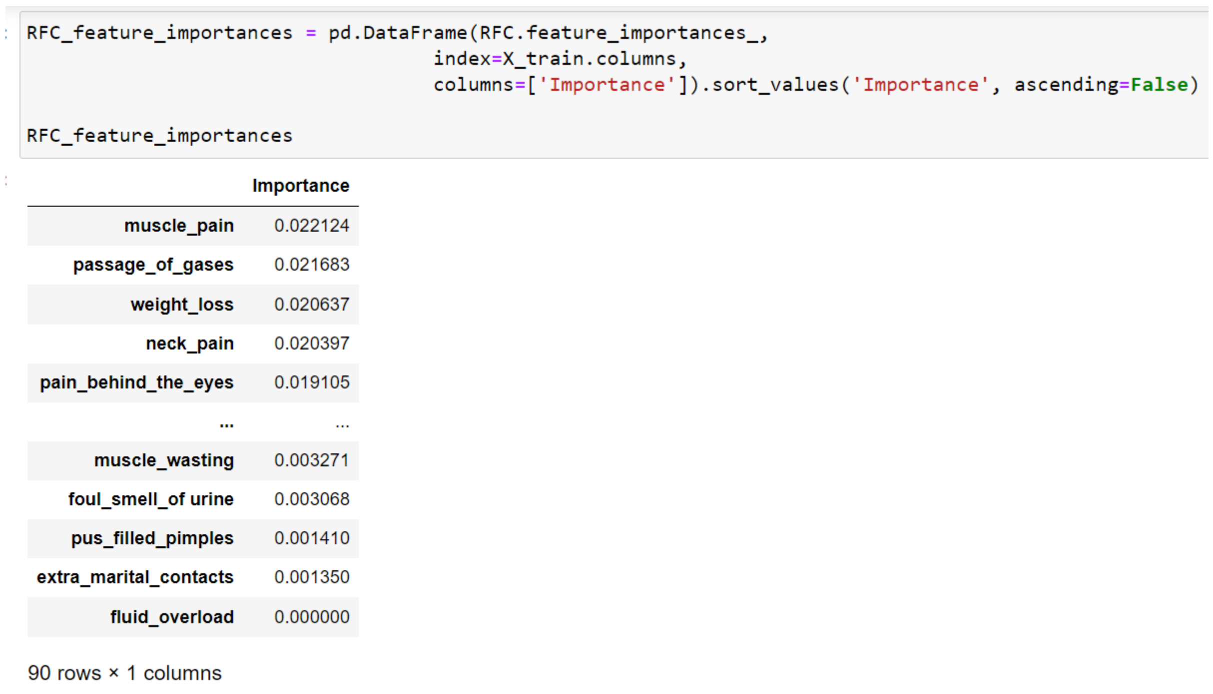
Activity 3: Comparing Model accuracy for different number of features.

Currently the training data has 90 features, which are a high number. If we need to reduce the number of features, we need to check the accuracies for various number of features.

We can check the feature importance using the Random Forest Classifier model.

In the figure below we have created a dictionary with the column names as indexes and the values as their feature importance. Much importance is not assigned to any feature. It is distributed among all the features.

We will keep some number of features for training and check for the accuracy. This process will be repeated a number of times.



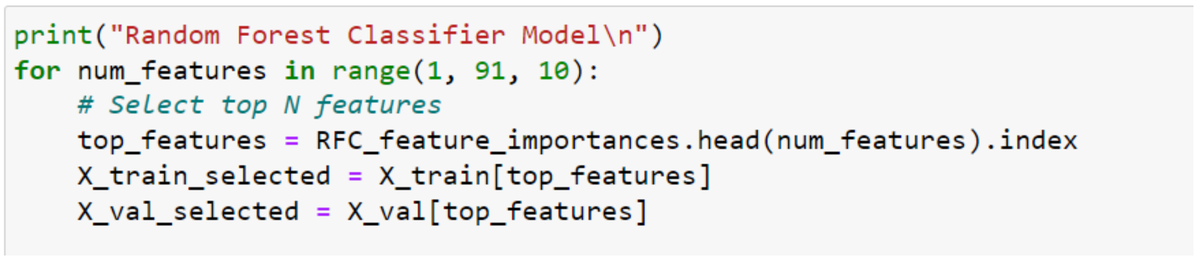
```python
RFC_feature_importances = pd.DataFrame(RFC.feature_importances_,
                                       index=X_train.columns,
                                       columns=['Importance']).sort_values('Importance', ascending=False)

RFC_feature_importances
```

	Importance
muscle_pain	0.022124
passage_of_gases	0.021683
weight_loss	0.020637
neck_pain	0.020397
pain_behind_the_eyes	0.019105
...	...
muscle_wasting	0.003271
foul_smell_of urine	0.003068
pus_filled_pimples	0.001410
extra_marital_contacts	0.001350
fluid_overload	0.000000

90 rows × 1 columns

We get a dictionary named feat_imp with 90 column names and their feature importance.

We will drop columns which have very less feature importance.



```python
print("Random Forest Classifier Model\n")
for num_features in range(1, 91, 10):
    # Select top N features
    top_features = RFC_feature_importances.head(num_features).index
    X_train_selected = X_train[top_features]
    X_val_selected = X_val[top_features]
```

Here we create 2 lists for 2 models, knn and random forest classifier.

The for loop will iterate over values given in the list one be one. The first value will be 1 and the last will be 81

There is a to_drop list created.

If the feature_importance is below threshold then the column name will be added to the to_drop list.

The columns whose name is in the to_drop list will be dropped.

The new data will be split into features and target variable. Further they will be split into training, validation and testing data.

```
    # Train the model
    RFC_selected = RandomForestClassifier()
    RFC_selected.fit(X_train_selected, y_train)

    # Evaluate accuracy on both training and validation sets
    train_accuracy = RFC_selected.score(X_train_selected, y_train)
    val_accuracy = RFC_selected.score(X_val_selected, y_val)

    print(f"Accuracy with {num_features} features - Training Accuracy: {train_accuracy} , Validation Accuracy: {val_accuracy}")
```

```
Random Forest Classifier Model

Accuracy with 1 features - Training Accuracy: 0.05157520325203252 , Validation Accuracy: 0.037601626016260166
Accuracy with 11 features - Training Accuracy: 0.4065040650406504 , Validation Accuracy: 0.3556910569105691
Accuracy with 21 features - Training Accuracy: 0.6847052845528455 , Validation Accuracy: 0.6636178861788617
Accuracy with 31 features - Training Accuracy: 0.8132621951219512 , Validation Accuracy: 0.801829268292683
Accuracy with 41 features - Training Accuracy: 0.931910569105691 , Validation Accuracy: 0.9186991869918699
Accuracy with 51 features - Training Accuracy: 0.9639227642276422 , Validation Accuracy: 0.9552845528455285
Accuracy with 61 features - Training Accuracy: 0.9898373983739838 , Validation Accuracy: 0.9857723577235772
Accuracy with 71 features - Training Accuracy: 0.9961890243902439 , Validation Accuracy: 0.9969512195121951
Accuracy with 81 features - Training Accuracy: 1.0 , Validation Accuracy: 1.0
```

Above random forest Classifier for various features. We can see that as the number of features go on increasing, the accuracies increase.

```
    # Train the model
    KNN_selected = KNeighborsClassifier()
    KNN_selected.fit(X_train_selected, y_train)

    # Evaluate accuracy on both training and validation sets
    train_accuracy = KNN_selected.score(X_train_selected, y_train)
    val_accuracy = KNN_selected.score(X_val_selected, y_val)
    print(f"Accuracy with {num_features} features - Training Accuracy: {train_accuracy} , Validation Accuracy: {val_accuracy}")
```

```
KNN Model

Accuracy with 1 features - Training Accuracy: 0.049796747967479675 , Validation Accuracy: 0.044715447154471545
Accuracy with 11 features - Training Accuracy: 0.40421747967479676 , Validation Accuracy: 0.3648373983739837
Accuracy with 21 features - Training Accuracy: 0.6834349593495935 , Validation Accuracy: 0.6626016260162602
Accuracy with 31 features - Training Accuracy: 0.788109756097561 , Validation Accuracy: 0.7865853658536586
Accuracy with 41 features - Training Accuracy: 0.9301321138211383 , Validation Accuracy: 0.9258130081300813
Accuracy with 51 features - Training Accuracy: 0.963160569105691 , Validation Accuracy: 0.9583333333333334
Accuracy with 61 features - Training Accuracy: 0.9895833333333334 , Validation Accuracy: 0.9867886178861789
Accuracy with 71 features - Training Accuracy: 0.9956808943089431 , Validation Accuracy: 0.9928861788617886
Accuracy with 81 features - Training Accuracy: 1.0 , Validation Accuracy: 1.0
```

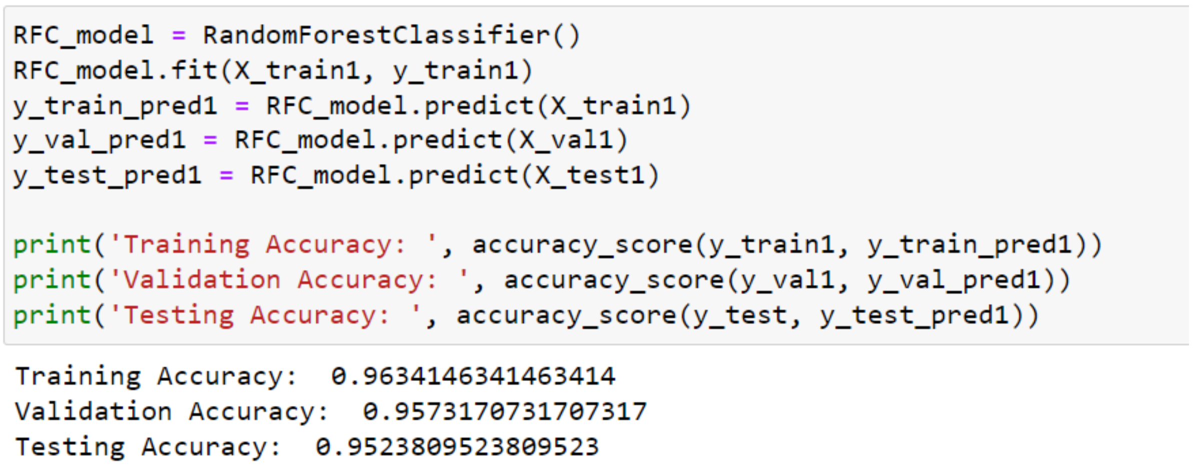
Above KNN model for various number of features. We can see that as the number of features go on increasing, the accuracies increase.

Activity 4: Building Model with appropriate features

From the above result tables, we can see that the accuracy does not change much from 51 features to 81 features. Hence we will choose 50 features for our training.

```
# Choosing the top 50 features as there is little change in accuracy from 50 to 80 features

top_features = RFC_feature_importances.head(50).index
X1 = X[top_features]
y1 = y
X_train1, X_val1, y_train1, y_val1 = train_test_split(X1, y1, train_size=0.8, random_state=42)
X_test1 = X_test[top_features]
```

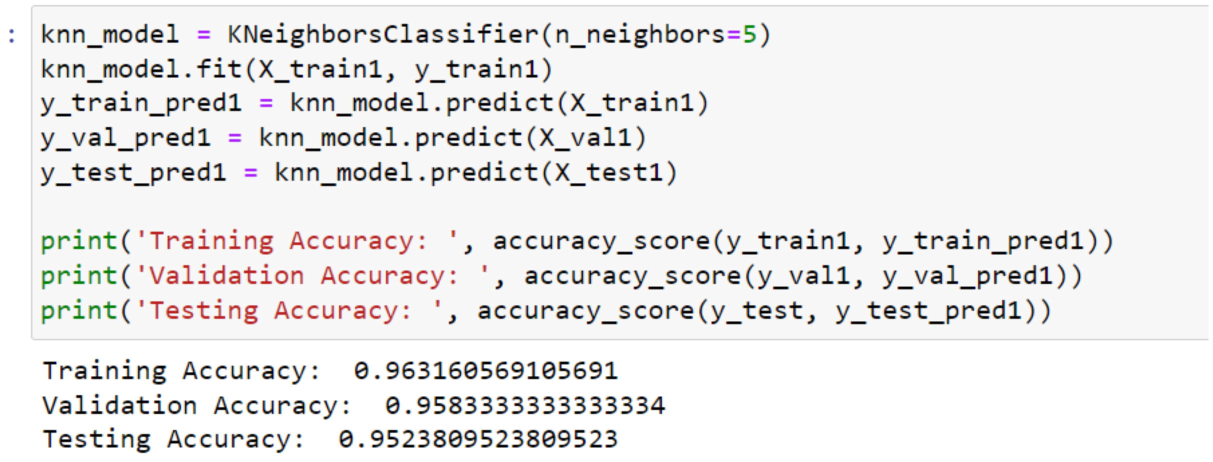
We dropped some features. Create new datasets for features and their labels. As we can see in the figure above we are left with 50 features now. We will build a Random Fores Classifier on the new data and check for the accuracies. As we can see in the figure below our model has achieved test accuracy of 95.2 % which is quite good for the number of features. Previously for 90 features we had similar accuracy for Random Forest Classifier. This states that there were many features which were not contributing much to our model.



```
RFC_model = RandomForestClassifier()
RFC_model.fit(X_train1, y_train1)
y_train_pred1 = RFC_model.predict(X_train1)
y_val_pred1 = RFC_model.predict(X_val1)
y_test_pred1 = RFC_model.predict(X_test1)

print('Training Accuracy: ', accuracy_score(y_train1, y_train_pred1))
print('Validation Accuracy: ', accuracy_score(y_val1, y_val_pred1))
print('Testing Accuracy: ', accuracy_score(y_test, y_test_pred1))
```

```
Training Accuracy:  0.9634146341463414
Validation Accuracy:  0.9573170731707317
Testing Accuracy:  0.9523809523809523
```

We will also train the model for KNN algorithm as KNN algorithm tends to perform better in such cases.

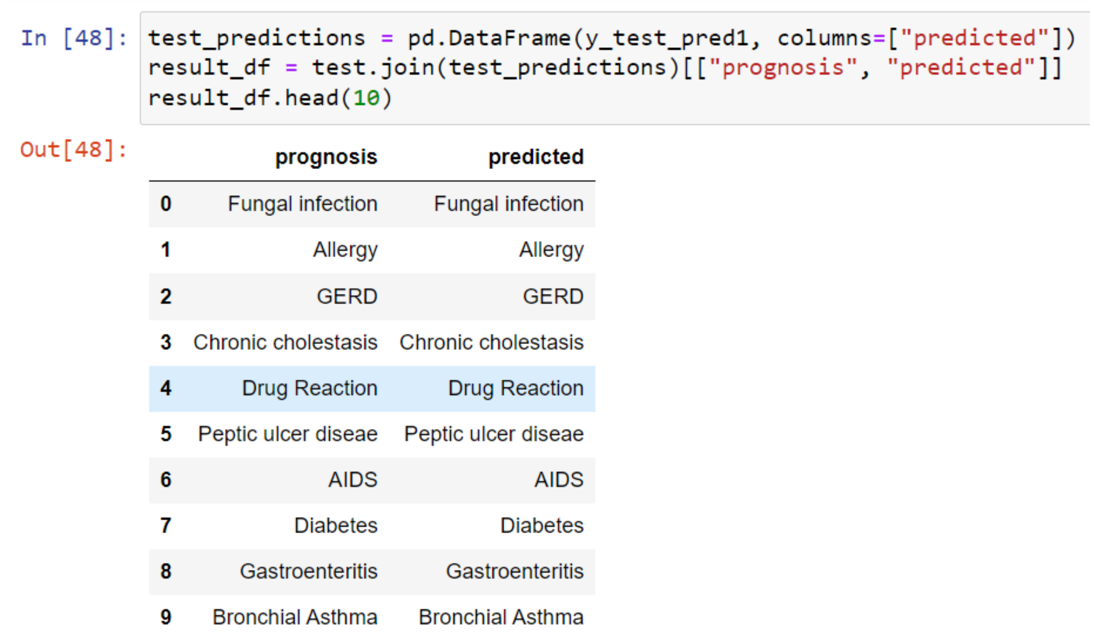


```
: knn_model = KNeighborsClassifier(n_neighbors=5)
  knn_model.fit(X_train1, y_train1)
  y_train_pred1 = knn_model.predict(X_train1)
  y_val_pred1 = knn_model.predict(X_val1)
  y_test_pred1 = knn_model.predict(X_test1)

  print('Training Accuracy: ', accuracy_score(y_train1, y_train_pred1))
  print('Validation Accuracy: ', accuracy_score(y_val1, y_val_pred1))
  print('Testing Accuracy: ', accuracy_score(y_test, y_test_pred1))
```

```
Training Accuracy:  0.963160569105691
Validation Accuracy:  0.9583333333333334
Testing Accuracy:  0.9523809523809523
```

After training the KNN model we check the accuracies. Our model has achieved 95.2 % accuracy for the test data.

To confirm let us check the compare our predicted results with the actual values.



```
In [48]: test_predictions = pd.DataFrame(y_test_pred1, columns=["predicted"])
         result_df = test.join(test_predictions)[["prognosis", "predicted"]]
         result_df.head(10)
```

Out[48]:

	prognosis	predicted
0	Fungal infection	Fungal infection
1	Allergy	Allergy
2	GERD	GERD
3	Chronic cholestasis	Chronic cholestasis
4	Drug Reaction	Drug Reaction
5	Peptic ulcer diseae	Peptic ulcer diseae
6	AIDS	AIDS
7	Diabetes	Diabetes
8	Gastroenteritis	Gastroenteritis
9	Bronchial Asthma	Bronchial Asthma

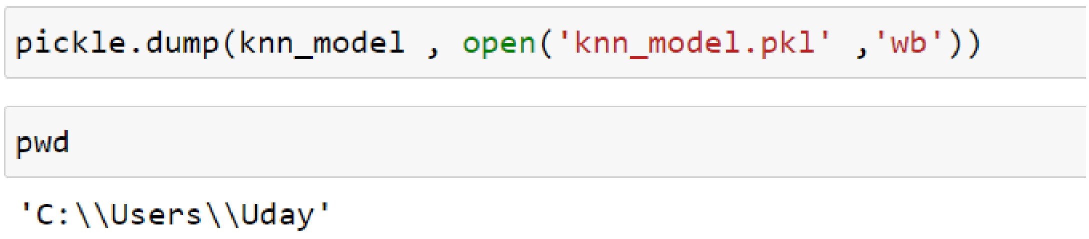
As we can see above that the values our model has predicted are same as the actual values. This shos that our model is performing good.

Milestone 6: Model Deployment

Activity 1: Save the best model

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance and saving its weights and configuration. This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future**.**

After checking the performance, we decide to save the knn model built with 45 features.



```
pickle.dump(knn_model , open('knn_model.pkl' ,'wb'))
```

```
pwd
```

```
'C:\\Users\\Uday'
```

We save the model using the pickle library into a file named model.pkl

Activity 2: Integrate with Web Framework

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI. This section has the following tasks

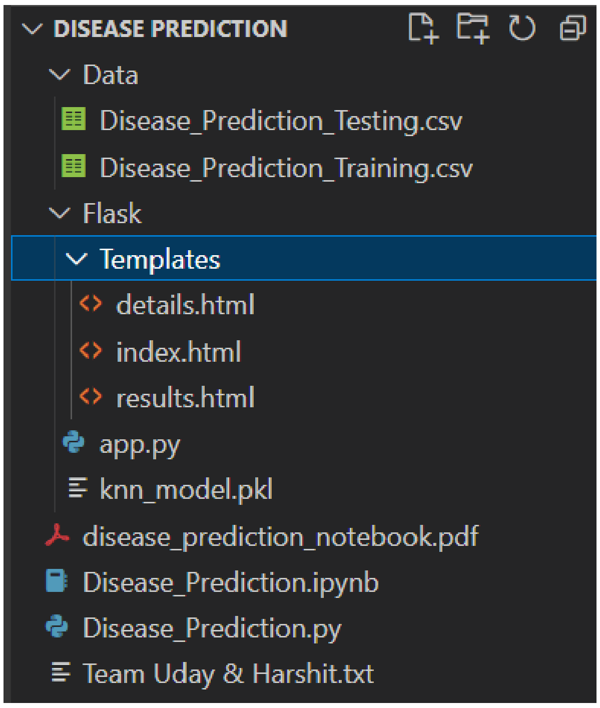
- Building HTML Pages
- Building server-side script
- Run the web application

Activity 2.1: Building HTML pages:

For this project we create three HTML files namely

- Index.html
- Details.html
- Results.html

And we will save them in the templates folder.

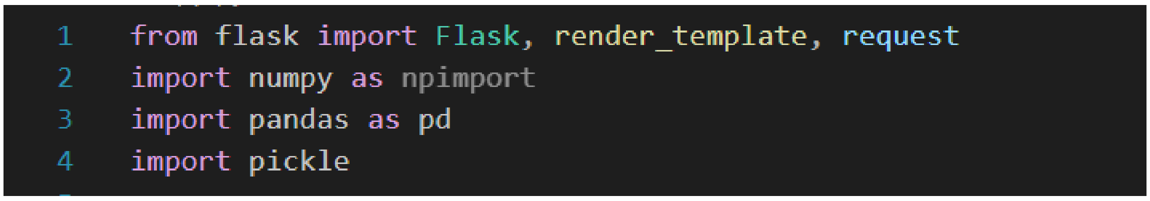


```
DISEASE PREDICTION
  Data
    Disease_Prediction_Testing.csv
    Disease_Prediction_Training.csv
  Flask
    Templates
      details.html
      index.html
      results.html
    app.py
    knn_model.pkl
  disease_prediction_notebook.pdf
  Disease_Prediction.ipynb
  Disease_Prediction.py
  Team Uday & Harshit.txt
```

Activity 2.2: Build Python code

Create a new app.py file which will be store in the Flask folder.

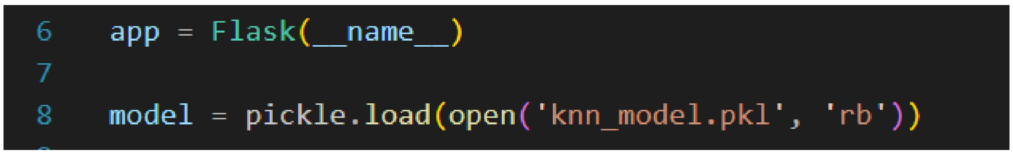
Import the necessary Libraries.



```python
from flask import Flask, render_template, request
import numpy as npimport
import pandas as pd
import pickle
```

This code first loads the saved Linear Regression model from the "bodyfat.pkl" file using the "pickle.load()" method. The "rb" parameter indicates that the file should be opened in binary mode to read data from it.

After loading the model, the code creates a new Flask web application object named "app" using the Flask constructor. The "name" argument tells Flask to use the current module as the name for the application.

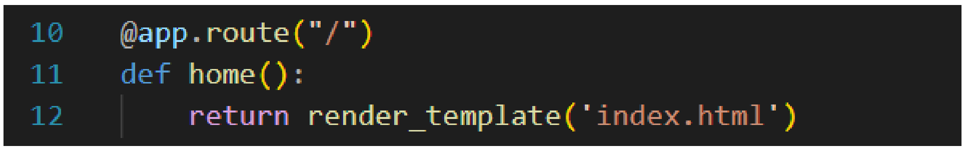


```python
app = Flask(__name__)

model = pickle.load(open('knn_model.pkl', 'rb'))
```

This code sets up a new route for the Flask web application using the "@app.route()" decorator. The route in this case is the root route "/", which is the default route when the website is accessed.

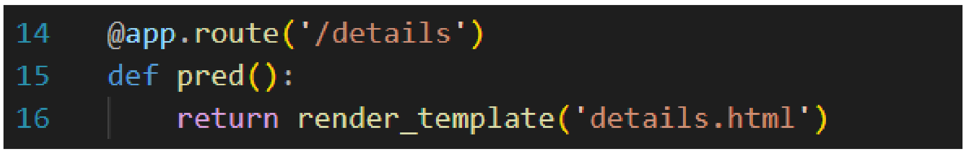
The function "home()" is then associated with this route. When a user accesses the root route of the website, this function is called.

The "render_template()" method is used to render an HTML template named "index.html". The “index.html” is the home page.



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

The route in this case is "/details". When a user accesses the "/predict" route of the website, this function is “index()” called. The "render_template()" method is used to render an HTML template named "details.html".



```python
@app.route('/details')
def pred():
    return render_template('details.html')
```

This Flask web application establishes a new route, "/predict", which accepts both GET and POST methods. The associated function, "predict()", handles the prediction process.

The variable 'col' is a list containing the names of 50 symptoms used in the model.

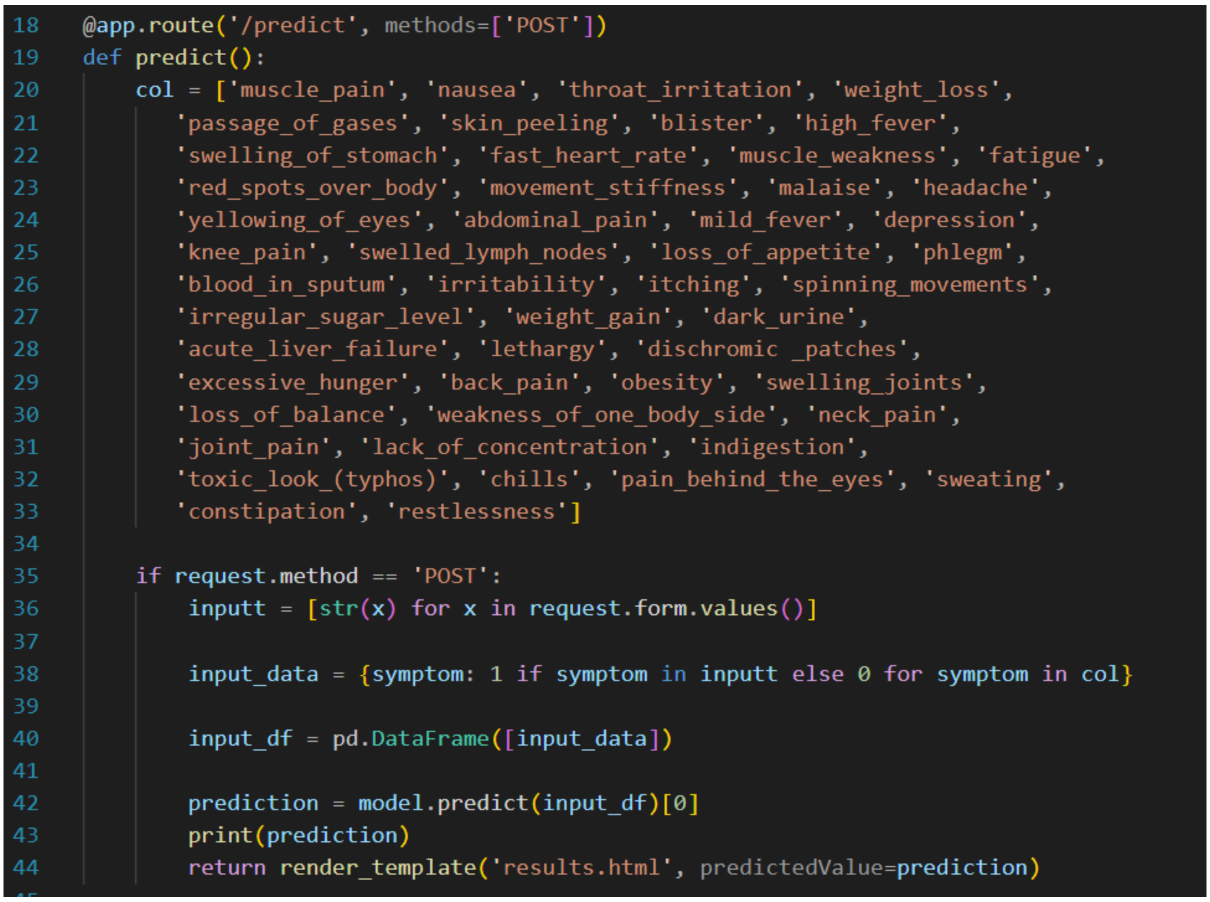
The function retrieves user inputs from a form in the details.html page, and these inputs are stored in the 'inputt' list as strings.

A list 'input_data' is created, initialized with 0s, and iterated through. For each symptom in 'col', if it matches a symptom in 'inputt', a corresponding 1 is set in 'input_data'.

The 'input_data' list is then converted into a DataFrame and used for making predictions with the pre-trained model.

The prediction result is printed and passed to the 'results.html' page using the 'render_template()' function.

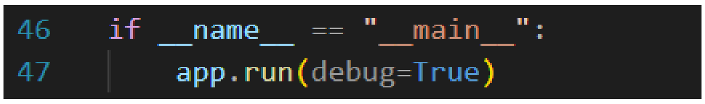
The code establishes a Flask route for predicting disease based on user-input symptoms. It efficiently processes user input, transforms it into a format suitable for the model, obtains predictions, and displays the result on a web page.



```python
@app.route('/predict', methods=['POST'])
def predict():
    col = ['muscle_pain', 'nausea', 'throat_irritation', 'weight_loss',
        'passage_of_gases', 'skin_peeling', 'blister', 'high_fever',
        'swelling_of_stomach', 'fast_heart_rate', 'muscle_weakness', 'fatigue',
        'red_spots_over_body', 'movement_stiffness', 'malaise', 'headache',
        'yellowing_of_eyes', 'abdominal_pain', 'mild_fever', 'depression',
        'knee_pain', 'swelled_lymph_nodes', 'loss_of_appetite', 'phlegm',
        'blood_in_sputum', 'irritability', 'itching', 'spinning_movements',
        'irregular_sugar_level', 'weight_gain', 'dark_urine',
        'acute_liver_failure', 'lethargy', 'dischromic _patches',
        'excessive_hunger', 'back_pain', 'obesity', 'swelling_joints',
        'loss_of_balance', 'weakness_of_one_body_side', 'neck_pain',
        'joint_pain', 'lack_of_concentration', 'indigestion',
        'toxic_look_(typhos)', 'chills', 'pain_behind_the_eyes', 'sweating',
        'constipation', 'restlessness']

    if request.method == 'POST':
        inputt = [str(x) for x in request.form.values()]

        input_data = {symptom: 1 if symptom in inputt else 0 for symptom in col}

        input_df = pd.DataFrame([input_data])

        prediction = model.predict(input_df)[0]
        print(prediction)
        return render_template('results.html', predictedValue=prediction)
```

Main Function:

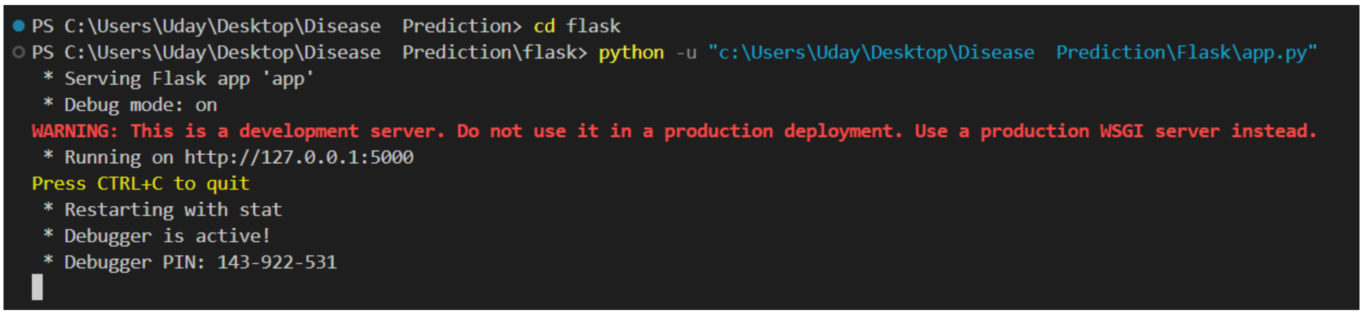
This code sets the entry point of the Flask application. The function “app.run()” is called, which starts the Flask deployment server.



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

Activity 2.3: Run the Web Application

When you execute the "app.py" file, a console or output terminal will display, and a window will open. Copy the URL provided, typically in the format http://127.0.0.1:5000 and paste it into your web browser's address bar. This will allow you to access and interact with the Flask web application locally.



```
● PS C:\Users\Uday\Desktop\Disease  Prediction> cd flask
○ PS C:\Users\Uday\Desktop\Disease  Prediction\flask> python -u "c:\Users\Uday\Desktop\Disease  Prediction\Flask\app.py"
   * Serving Flask app 'app'
   * Debug mode: on
  WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
   * Running on http://127.0.0.1:5000
  Press CTRL+C to quit
   * Restarting with stat
   * Debugger is active!
   * Debugger PIN: 143-922-531
```

When we paste the URL in a web browser, our index.html page will open.

It contains some sections such as Predict and Contact.
There is some information given on the web page about our model.

If you click on the Predict button on home page or in the header bar you will be redirected to the details.html page.

8. PERFORMANCE TESTING

8.1 Performance Metrics

Classification Report:

```
                                         precision    recall  f1-score   support

(vertigo) Paroymsal  Positional Vertigo       1.00      1.00      1.00         1
                                   AIDS       1.00      1.00      1.00         1
                                   Acne       0.00      0.00      0.00         1
                    Alcoholic hepatitis       1.00      1.00      1.00         1
                                Allergy       0.00      0.00      0.00         1
                              Arthritis       1.00      1.00      1.00         1
                       Bronchial Asthma       1.00      1.00      1.00         1
                   Cervical spondylosis       1.00      1.00      1.00         1
                            Chicken pox       1.00      1.00      1.00         1
                    Chronic cholestasis       1.00      1.00      1.00         1
                            Common Cold       1.00      1.00      1.00         1
                                 Dengue       1.00      1.00      1.00         1
                              Diabetes        1.00      1.00      1.00         1
           Dimorphic hemmorhoids(piles)       0.25      1.00      0.40         1
                          Drug Reaction       0.50      1.00      0.67         1
                       Fungal infection       1.00      0.50      0.67         2
                                   GERD       0.50      1.00      0.67         1
                        Gastroenteritis       1.00      1.00      1.00         1
                           Heart attack       0.00      0.00      0.00         1
                            Hepatitis B       1.00      1.00      1.00         1
                            Hepatitis C       1.00      1.00      1.00         1
                            Hepatitis D       1.00      1.00      1.00         1
                            Hepatitis E       1.00      1.00      1.00         1
                          Hypertension        1.00      1.00      1.00         1
                        Hyperthyroidism       1.00      1.00      1.00         1
                           Hypoglycemia       1.00      1.00      1.00         1
                         Hypothyroidism       1.00      1.00      1.00         1
                               Impetigo       1.00      1.00      1.00         1
                               Jaundice       1.00      1.00      1.00         1
                                Malaria       1.00      1.00      1.00         1
                               Migraine       1.00      1.00      1.00         1
                        Osteoarthristis       1.00      1.00      1.00         1
           Paralysis (brain hemorrhage)       1.00      1.00      1.00         1
                    Peptic ulcer diseae       1.00      1.00      1.00         1
                              Pneumonia       1.00      1.00      1.00         1
                              Psoriasis       1.00      1.00      1.00         1
                           Tuberculosis       1.00      1.00      1.00         1
                                Typhoid       1.00      1.00      1.00         1
                Urinary tract infection       0.00      0.00      0.00         1
                         Varicose veins       1.00      1.00      1.00         1
                            hepatitis A       1.00      1.00      1.00         1

                               accuracy                           0.88        42
                              macro avg       0.86      0.89      0.86        42
                           weighted avg       0.86      0.88      0.86        42
```

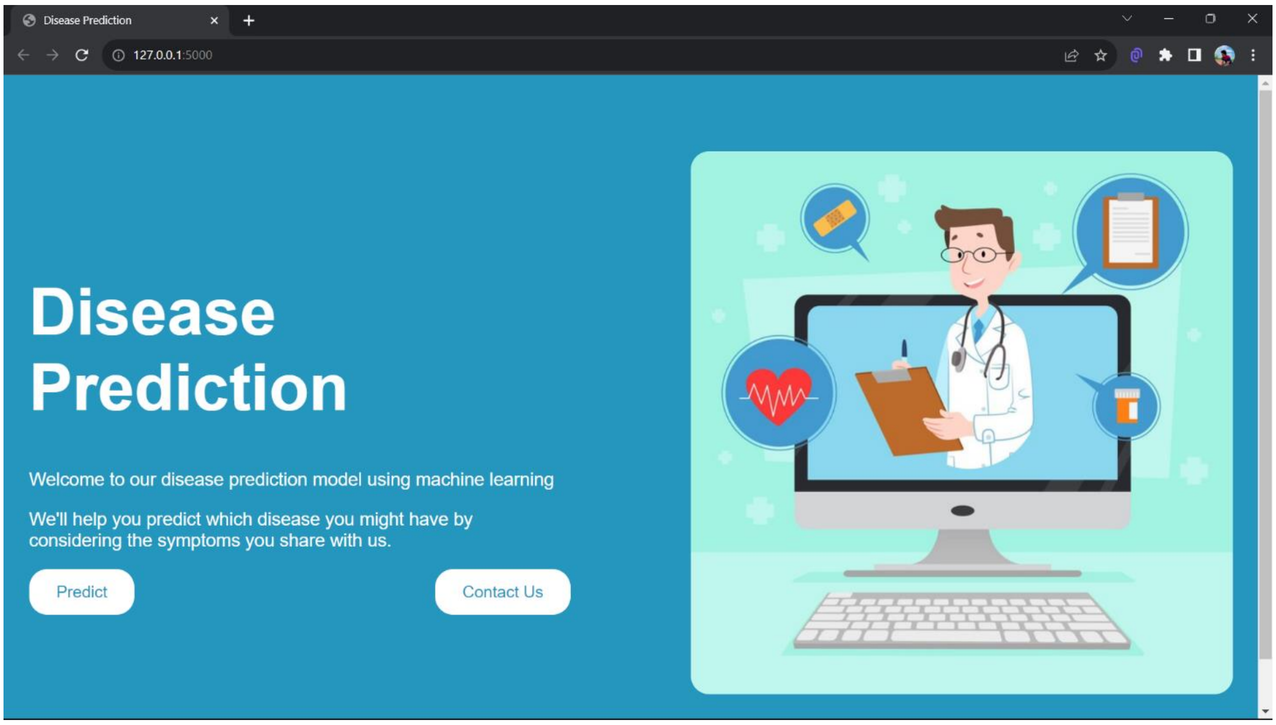
Accuracy Score:

```
            Accuracy  Precision    Recall  F1 Score
Training    0.935976   0.919978  0.935976  0.923063
Validation  0.926829   0.908109  0.926829  0.911851
Testing     0.928571   0.916667  0.928571  0.912698
```

```
Training Accuracy:  0.889989837398374
Validation Accuracy:  0.8851626016260162
Testing Accuracy:  0.8809523809523809
```

9. RESULTS

"Index.html" is shown as follows.



Disease Prediction

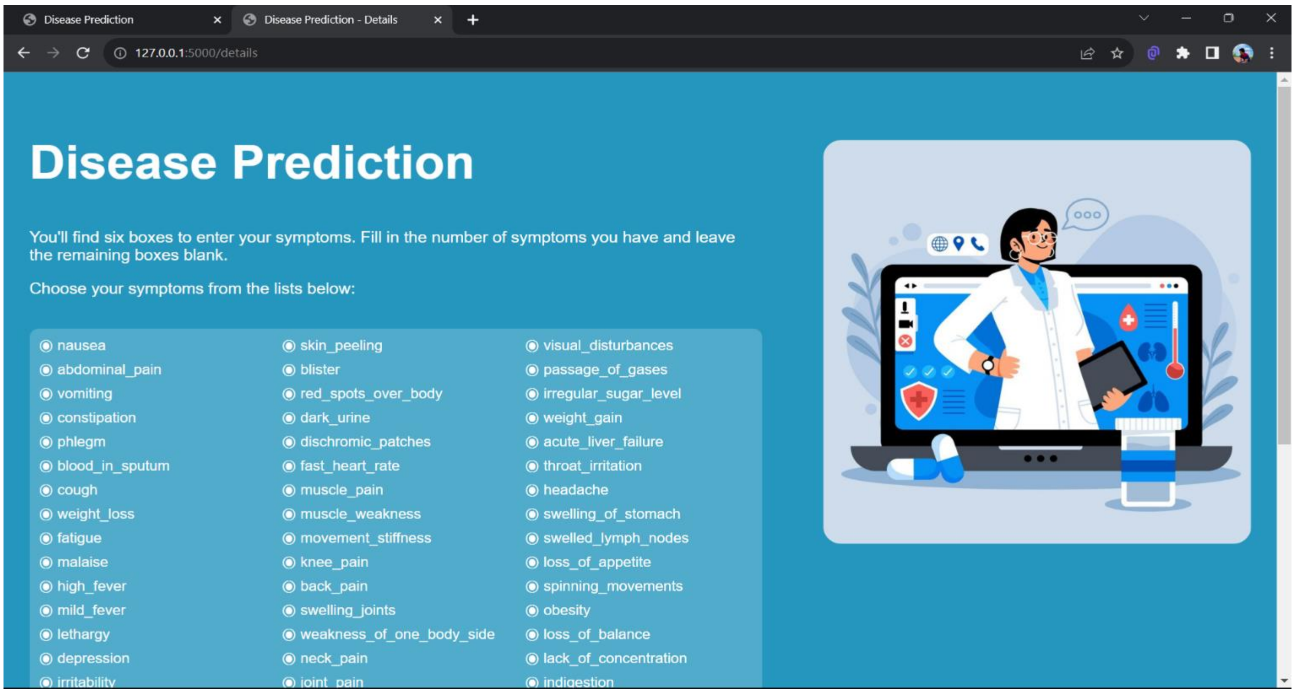
127.0.0.1:5000

Disease Prediction

Welcome to our disease prediction model using machine learning

We'll help you predict which disease you might have by considering the symptoms you share with us.

"Details.html" is shown as follows.



Disease Prediction | Disease Prediction - Details

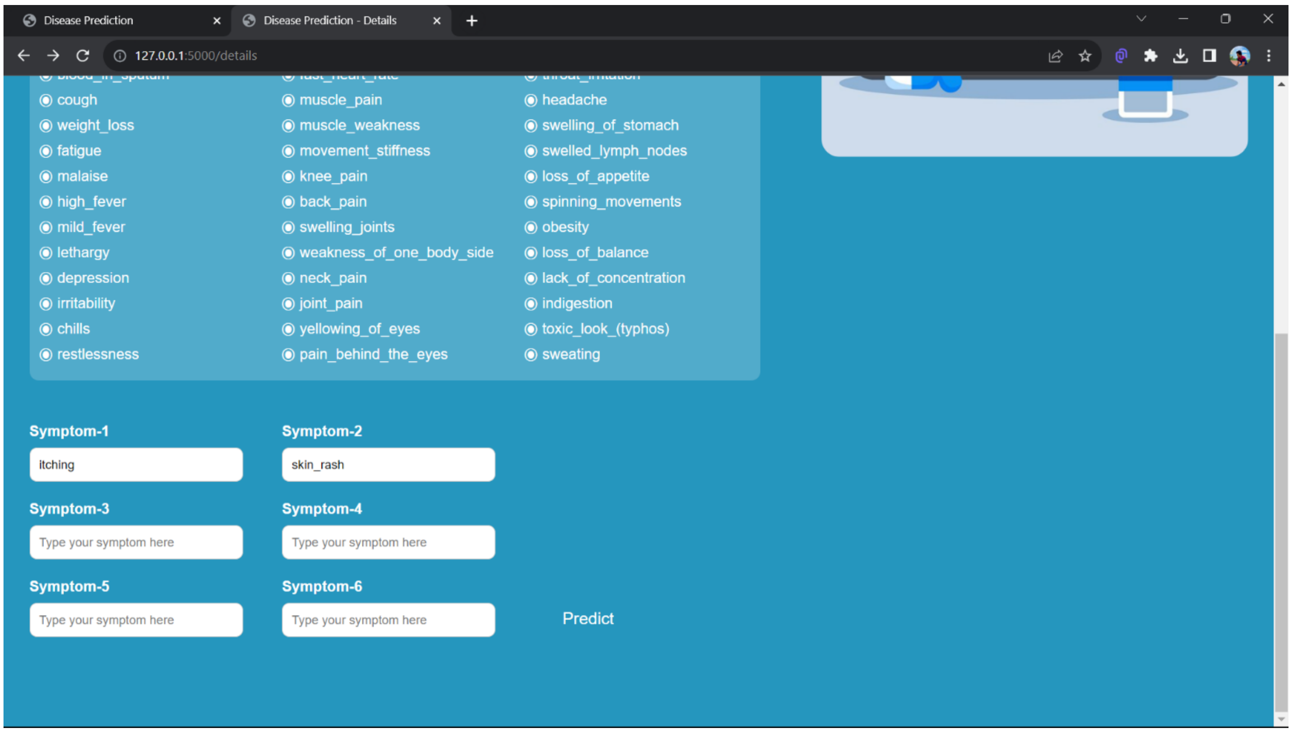
127.0.0.1:5000/details

Disease Prediction

You'll find six boxes to enter your symptoms. Fill in the number of symptoms you have and leave the remaining boxes blank.

Choose your symptoms from the lists below:

nausea	skin_peeling	visual_disturbances
abdominal_pain	blister	passage_of_gases
vomiting	red_spots_over_body	irregular_sugar_level
constipation	dark_urine	weight_gain
phlegm	dischromic_patches	acute_liver_failure
blood_in_sputum	fast_heart_rate	throat_irritation
cough	muscle_pain	headache
weight_loss	muscle_weakness	swelling_of_stomach
fatigue	movement_stiffness	swelled_lymph_nodes
malaise	knee_pain	loss_of_appetite
high_fever	back_pain	spinning_movements
mild_fever	swelling_joints	obesity
lethargy	weakness_of_one_body_side	loss_of_balance
depression	neck_pain	lack_of_concentration
irritability	joint_pain	indigestion



Disease Prediction | Disease Prediction - Details

127.0.0.1:5000/details

cough	muscle_pain	headache
weight_loss	muscle_weakness	swelling_of_stomach
fatigue	movement_stiffness	swelled_lymph_nodes
malaise	knee_pain	loss_of_appetite
high_fever	back_pain	spinning_movements
mild_fever	swelling_joints	obesity
lethargy	weakness_of_one_body_side	loss_of_balance
depression	neck_pain	lack_of_concentration
irritability	joint_pain	indigestion
chills	yellowing_of_eyes	toxic_look_(typhos)
restlessness	pain_behind_the_eyes	sweating

Symptom-1: itching
Symptom-2: skin_rash
Symptom-3: Type your symptom here
Symptom-4: Type your symptom here
Symptom-5: Type your symptom here
Symptom-6: Type your symptom here

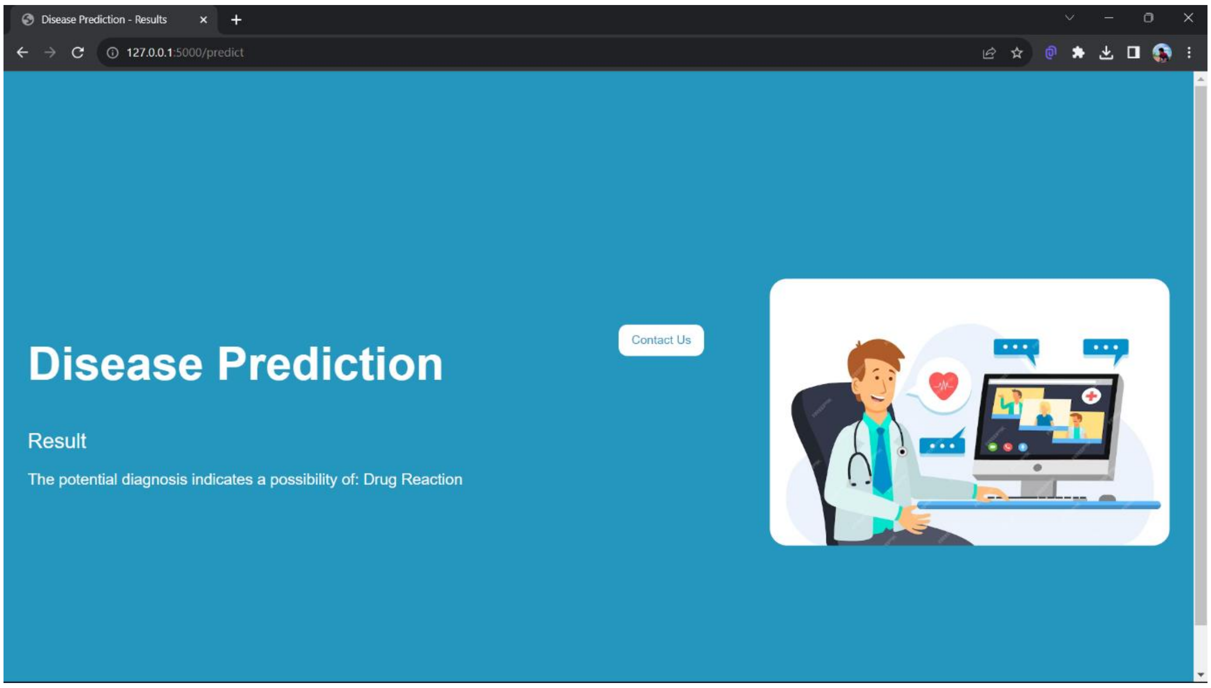
Predict

We have six input fields available for entering symptoms, and users have the flexibility to fill all six boxes or just one.

The list of symptoms is presented in bullet points above these input fields, and it's important to input symptoms exactly as listed since they correspond to the column names.

Users can enter symptoms like itching, skin rash, and more into the input fields. After entering the symptoms, clicking the "Predict" button initiates the prediction process.

Upon doing so, users will be redirected to the "Results.html" page, where the output will be displayed.



Disease Prediction

Result

The potential diagnosis indicates a possibility of: Drug Reaction

The results say that there are high chances that patient has **Drug Reaction** based on the various symptoms we have given as input.

10. ADVANTAGES & DISADVANTAGES

10.1 Advantages

10.1.1 Timely Health Insights

- Advantage: Users gain access to timely and immediate health insights, allowing for quick assessments of symptoms.

-Benefit: Early awareness can lead to proactive health management and the prevention of potential complications.

10.1.2 Privacy-Respecting Model

- Advantage: The model operates without collecting personal information, ensuring user privacy and fostering trust.

- Benefit: Encourages users to share accurate symptom information without compromising sensitive data.

10.1.3 Cost-Efficient Online Consultations

- Advantage: Clinicians can use the tool for online consultations, providing a cost-effective alternative to traditional in-person visits.

- Benefit: Enhances healthcare accessibility and reduces financial barriers for both patients and healthcare providers.

10.1.4 Decision Support for Users

- Advantage: The tool serves as a decision support system, guiding users on when to seek professional medical advice.

- Benefit: Empowers users to make informed decisions, promoting responsible healthcare practices.

10.1.5 Accessibility and Convenience

- Advantage: Users can access the tool at any time, promoting convenient and accessible healthcare.

- Benefit: Supports busy lifestyles and ensures individuals can prioritize their health without sacrificing time constraints.

10.2 Disadvantages

10.2.1 Reliance on Symptom-Based Analysis

- Disadvantage: The model relies on symptom-based analysis, which may not substitute for a comprehensive in-person examination by a healthcare professional.

- Challenge: Users should be aware of the limitations and use the tool as a supplementary resource, not a replacement for professional medical advice.

10.2.2 Technological Barriers

- Disadvantage: Individuals with limited technological proficiency may face challenges in using the tool effectively.

- Challenge: User education and support resources are crucial to mitigate this disadvantage and ensure inclusivity.

10.2.3 Lack of Physical Examination

- Disadvantage: The tool cannot perform physical examinations, limiting its ability to assess certain health conditions.

- Challenge: Users should recognize the tool's scope and understand the importance of combining digital insights with traditional healthcare practices.

10.2.4 Internet Dependency

- Disadvantage: The tool relies on internet connectivity, posing challenges for users in areas with limited or no access to the internet.

- Challenge: Efforts should be made to enhance offline functionality or provide alternative solutions for users with connectivity issues.

10.2.5 Ethical and Legal Considerations

- Disadvantage: Ensuring compliance with healthcare regulations and ethical considerations requires ongoing diligence.

- Challenge: Continuous monitoring and adaptation to evolving legal and ethical standards are necessary to mitigate potential risks.

11. CONCLUSION

In conclusion, the development and implementation of the symptom-based disease forecasting tool represent a significant step towards addressing the prevalent challenges in contemporary healthcare. The project's overarching goal was to provide individuals with a user-friendly, accessible, and privacy-respecting platform for timely health insights, ultimately bridging the gap between busy lifestyles and healthcare needs.

The tool's unique advantages, including the provision of immediate health insights, privacy-respecting design, and cost-efficient online consultations, contribute to a more proactive and informed approach to health management. By empowering users to make timely decisions about seeking professional medical advice and offering clinicians a valuable resource for online consultations, the tool aligns with the evolving needs of individuals and healthcare providers in the digital age.

However, it is essential to acknowledge the tool's limitations, such as its reliance on symptom-based analysis and the need for ongoing user education to address technological barriers. Users and healthcare professionals alike should view the tool as a supplementary resource, emphasizing the importance of combining digital insights with traditional healthcare practices.

In moving forward, continuous monitoring and adaptation to technological, ethical, and legal considerations will be crucial. By staying abreast of emerging healthcare standards and user needs, the tool can evolve to remain a reliable and relevant asset in the dynamic landscape of digital health solutions.

In essence, the symptom-based disease forecasting tool represents a proactive step towards democratizing healthcare, providing individuals with the tools they need to take control of their health while fostering collaboration between users and healthcare professionals. The journey does not end here; it marks the beginning of a continuous commitment to enhancing the tool's functionality, accessibility, and overall impact on healthcare practices.

12. FUTURE SCOPE

The symptom-based disease forecasting tool lays the foundation for a dynamic and evolving platform in the realm of digital health. As technology advances and healthcare landscapes continue to transform, the project has a promising future with several avenues for expansion and improvement:

1. Integration of Advanced Technologies

- Artificial Intelligence Enhancements: Incorporate advanced AI algorithms and machine learning techniques to continually improve the accuracy of disease forecasting based on symptom inputs.

- IoT Integration: Explore the integration of Internet of Things (IoT) devices for real-time health monitoring, providing a more comprehensive understanding of user health status.

2. Enhanced User Experience

- User Feedback Mechanism: Implement a robust user feedback system to gather insights on user experiences, ensuring continuous improvement based on user input.

- Personalized User Profiles: Introduce optional user profiles to enhance personalization, allowing users to track and manage their health data over time.

3. Expansion of Disease Database

- Continuous Research and Updates: Regularly update the disease database with the latest medical research findings and emerging health conditions, ensuring the tool remains current and relevant.

- Internationalization: Expand the tool's disease database to cover a broader range of international health conditions, catering to a diverse global user base.

4. Telemedicine and Remote Monitoring

- Telemedicine Integration: Strengthen the integration with telemedicine platforms, enabling seamless connectivity between users and healthcare professionals for remote consultations.

- Chronic Disease Management: Extend the tool's capabilities to support the remote monitoring and management of chronic health conditions.

5. Education and Awareness

- Health Education Resources: Expand the platform to include educational resources on various health topics, fostering greater awareness and understanding among users.

- Partnerships with Healthcare Organizations: Collaborate with healthcare organizations and professionals to disseminate accurate health information and promote responsible health practices.

6. Offline Functionality

- Offline Symptom Checker: Develop an offline mode for the symptom checker, allowing users in areas with limited internet connectivity to access basic health insights.

- Data Synchronization: Implement a secure data synchronization mechanism to ensure users can seamlessly transition between offline and online modes.

7. Wearable Technology Integration

- Wearable Device Compatibility: Explore integration with wearable health devices to enhance the accuracy of symptom data collection and provide continuous health monitoring.

- Health Data Aggregation: Aggregate data from wearable devices to offer users a more comprehensive view of their health status.

8. Regulatory Compliance and Ethical Considerations

- Continuous Compliance Monitoring: Stay vigilant in monitoring and adapting to evolving healthcare regulations, ensuring the platform remains in compliance with data protection and privacy standards.

- Ethical AI: Implement ethical AI principles to address biases and ensure fairness in disease predictions.

9. Research Collaborations

- Collaborations with Research Institutions: Foster collaborations with research institutions to contribute to ongoing medical research and potentially integrate cutting-edge findings into the tool.

- Clinical Validation Studies: Conduct clinical validation studies to assess the tool's accuracy and effectiveness in real-world healthcare scenarios.

As the project moves forward, these future scope considerations pave the way for a more robust, adaptive, and user-centric tool that continues to redefine the intersection of technology and healthcare. The commitment to innovation and user well-being ensures that the tool remains a valuable asset in the ever-evolving landscape of digital health solutions.

13. APPENDIX

Source Code:

GitHub Link - https://github.com/smartinternz02/SI-GuidedProject-611642-1700551037

Project Demo Link – https://drive.google.com/file/d/1t0TjmVdSWImp-i0jv6Ak3QMCaBv6ZulV/view?usp=sharing