HealthCare Application with Diseases prediction

Supervised by:
Prof. Doaa Saad Elzanfaly

Table of Contents

HELWAN UNIVERSITY
Faculty of Computers and Artificial Intelligence
Information Systems Department

Yousra Ezaldien Mohammed Salih	201900969
Mona Attya Ahmed Yousef Mandor	201900854
Yara Ahmed Ahmed Ali	201900952
Abeer Salah Bayoumy Mohamed	201900472
Hany Youssef Fahem FaragAllh	201900925
Mina Ashraf Ayad Sedhom	201900875

Submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Computers & Artificial Intelligence, at the Information Systems Department, the Faculty of Computers & Artificial Intelligence, Helwan University

Supervised by:

Prof. Doaa Saad Elzanfaly

June 2023

ACKNOWLEDGEMENT

(وَآخِرُ دَعْوَاهُمْ أَنِ الْحَمْدُ لِلَّهِ رَبِّ الْعَالَمِينَ)
اللهم لك الحمد حتى ترضى ولك الحمد إذا رضيت ولك الحمد بعد الرضى

To our families & frineds, this would have been a much more difficult feat without you. Thank you all for your unwavering support and love throughout this journey.

Also, we would like to thank our supervisor Prof. Doaa Saad Elzanfaly for her gaudiness throughout this project and for all of her work and patience over the past years, we are lucky to be one of your students and truly grateful to have had you as both professor and supervisor.

CHAPTER 1: Introduction

Overview

Abstract

Technology is transforming the way people bank, travel, and shop. But, it has yet to make significant inroads into the healthcare industry. The healthcare industry is one of the world's largest and fastest-growing industries. Consuming over 10 percent of the gross domestic product (GDP) of most developed nations. Therefore, in this project, our aim is to *highlight the value* of someone's health, *providing assistance for physicians, and* medical *professionals.*

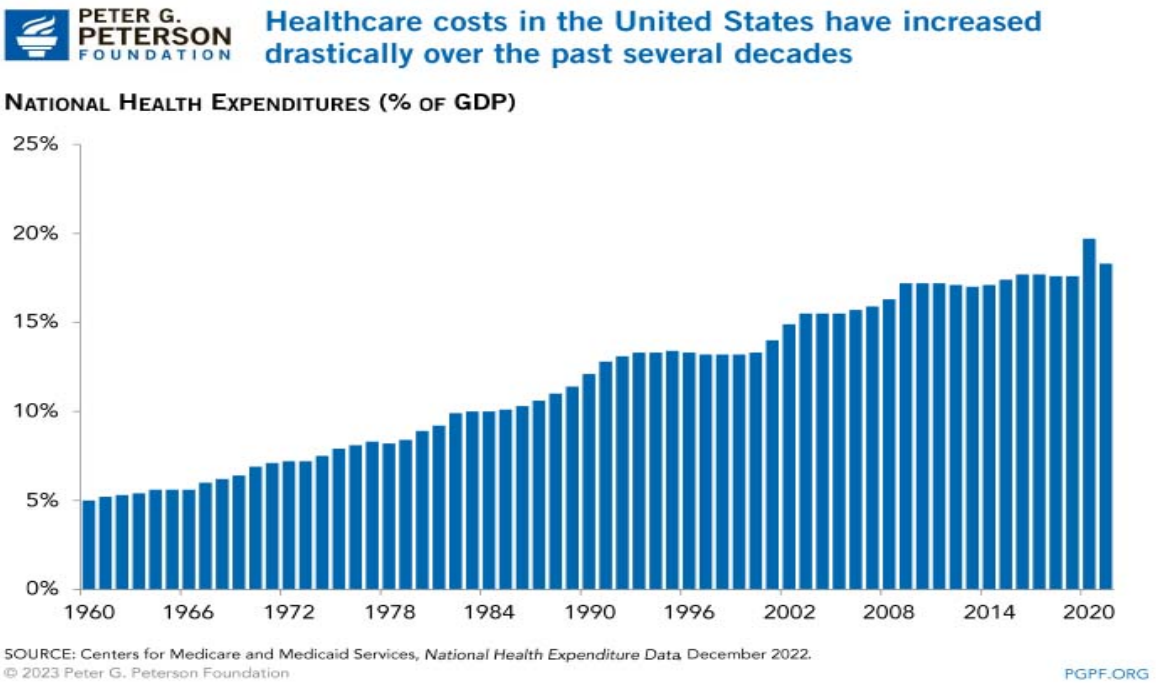
Project Motivation

We live in a high-speed world. Work and personal life are often rushed and mashed together in a blur of activity. Thus, due to people's busy life, people tend to forget about their scheduled clinic appointments, A study focused on the reasoning behind missed clinic scheduled appointments showed that out of 100 patients included in the study, 36 patients reported forgetting about the clinic appointment, Patients that do not attend a scheduled clinic appointment (Termed: No-shows) cause administrative issues to the resident's practice, negatively impact the workflow, and affect the financial status of the facility, continuity of patient care, resident education, and clinic efficiency.

People also forget to take their own medications, in the U.S., about 50% of medications for long-term conditions aren't taken the way they were prescribed. And up to 30% of prescriptions are never even filled, not taking medications properly leads to unnecessary hospital admissions, illness, and even deaths. It also costs the healthcare system billions of dollars every year, in costs both direct (e.g., hospitalizations) and indirect (e.g., lost productivity).
In addition, in the past two years, much has changed since the start of the coronavirus disease 19 (COVID-19) pandemic. The lack of interaction with physicians during COVID-19 was one of the major issues people faced during that era, which increased the difficulty of obtaining medical diagnoses. Therefore, in times of crisis, immense creativity often comes to the fore, that's when Artificial intelligence and Machine learning play a major role in the healthcare industry to help both individuals and corporations, Artificial Intelligence (AI) technology is being used in the diagnosis of diseases, as well as to offer customized solutions, Now-days, people face various diseases due to the environmental condition and their living habits. So the prediction of disease at an earlier stage becomes an important task. But the accurate prediction on the basis of symptoms becomes -sometimes- too difficult for physicians. The correct prediction of disease is the most challenging task. To overcome this problem Machine learning is being used to predict the disease which can help produce accurate results.

Problem Statement

Healthcare is one of the most important industries, and developing it helps cure many, even sometimes, rare diseases. Mainly thanks to the technological advances in the medical sector, which enabled a deeper understanding of humanity. Technology plays a huge role by giving more insights into the human body. However, progress cannot happen without considering other things stopping the way to a better and healthier future. Among others are rising healthcare costs, growing inequities, and climate change. Therefore In this proposed system, we presented a flexible solution to one of the main drawbacks of the healthcare industry, which is the healthcare cost. For example, The United States has one of the highest costs of healthcare in the world. In 2021, U.S. healthcare spending reached $4.3 trillion, which averages to about $12,900 per person. By comparison, the average cost of healthcare per person in other wealthy countries is only about half as much. While the COVID-19 pandemic exacerbated the trend in rising healthcare costs, such spending has been increasing long before COVID-19 began. Relative to the size of the economy, healthcare costs have increased over the past few decades, from 5 percent of GDP in 1960 to 18 percent in 2021.



Without insurance, the cost of going to a doctor typically ranges from $300 to $600. This price will vary depending on whether you see a specialist if lab tests are completed, and if any procedures are done.

Providing access to a disease prediction system will result in an improvement on both organizational levels and individual levels. In addition to the high cost of healthcare, COVID-19 pandemic was a major challenge that resulted in a lack of interaction with physicians, most countries followed lockdowns and social distancing to curb the spread. However, with the help of technology, remote diagnoses helped (non-COVID) to receive improved patient care.

Another solution we presented in this application is, Electronic Health Record (EHR) gone are the days of bulky files and worn-out papers, Electronic Health Records (HRs) or electronic medical records help save summaries of a patient's medical records digitally. The digital summary can include lab reports, diagnoses, surgical interventions, prescriptions, and even details of hospital stays. Electronic medical records offer better insights into a patient's health leading to accurate diagnosis and better patient care. Digital records facilitate the sharing of information between specialists and labs thereby improving coordination. If properly maintained and implemented, digital records can also help increase accountability and reduce medical negligence. Simply put, HRs consume less time to create and are easier to maintain. They help medical professionals reduce the chances of making mistakes and make their life easier.

In this proposed system, EHR is accessible and used by patients, patients can upload their digital summary including lab reports, diagnoses, surgical interventions, and prescriptions. These digital summaries are accessible 24/7.

Project Aim and Objectives

In this proposed system, our main aim is to provide high-quality solutions for both organizations and individuals. Serving organizations results in a better healthcare system, due to records progress in biomedical and healthcare groups, correct have a look at clinical data advantages of early disorder recognition, patient care, and network services. Early disorder recognition for patients can benefit organizations in many ways, new disorders can be discovered due to similar patterns that may not be noticeable to human experts, boosting the organization’s reputation as a reliable healthcare organization, therefore, guaranteeing more revenue, reducing diagnosis time, improving lab results, improving the overall healthcare system, therefore patients will have a better-hospitalized experience.

Individuals can also benefit from disease prediction in several ways, early detection can be critical in the successful treatment and management of many diseases. Improved outcomes, by providing more accurate and timely diagnoses, reducing the likelihood of misdiagnosis, and helping to identify patients who are at high risk of developing certain diseases. reduced healthcare costs. Early detection can lead to reduced healthcare costs by avoiding unnecessary treatments, reducing hospitalizations, and minimizing the need for expensive diagnostic tests.

Overall, disease prediction by machine learning has the potential to revolutionize healthcare by improving the accuracy and efficiency of disease diagnosis and treatment, leading to better outcomes for patients.

Moreover, Electronic Health Records (EHRs) have several benefits for individuals, including improving quality of healthcare, EHRs provide healthcare providers with access to a patient's complete medical history, including previous diagnoses, medications, and lab results. This information can help healthcare providers make more informed decisions about a patient's care, leading to improved quality of care. Increased Patient Safety, EHRs can help reduce medical errors by providing alerts for potential drug interactions, allergies, and other safety concerns. This can help prevent adverse drug events and improve patient safety. Better Communication, EHRs can improve communication between healthcare providers, as they can share patient information and collaborate on treatment plans. This can lead to better coordination of care and improved outcomes for patients. Convenience: EHRs allow patients to access their medical records from anywhere, at any time. This can be convenient for patients who need to share their medical information with other healthcare providers or keep track of their health data

Project Limitations

Patient privacy is always a concern when it comes to digitalizing healthcare processes, the use of mobile devices puts patient data at risk. Unauthorized access to patient information isn’t a priority. The system will only allow authorized users to use this application. The system will never use any data from the EHR without the user's permission.

Project Expected Output

In this system we expect the output to meet the requirements. Both users and healthcare professionals can use disease prediction in order to predict the likelihood of a patient developing a particular disease, electronic medical records are used to help save summaries of a patient's medical records digitally. The digital summary can include lab reports, diagnoses, surgical interventions, prescriptions, and even details of hospital stays. Electronic medical records offer better insights into a patient's health leading to accurate diagnosis and better patient care.

CHAPTER 2: Analysis

Use Case Diagram

Alternative Flow	*Register Account :*
Description	***The system will make sure that the values entered by the user are valid or not***
Actors	***System***
Pre-Condition	***Valid values must be specified***

Use Case Name	*Register*
Use Case Description	***User Registers an account***
Actors	***User***
Pre-Condition	***User Does not have any account***
Post - Condition	***The User's details are stored in the database***
Main Flow	***1- The use case starts when user clicks the "sign up" button*** ***2- The System displays the sign up page that allow user to fil his information*** ***3- After that the user click on sign up button*** ***4- The system display the login page to allow to the user to fil his email and password*** ***To be able to login to the system***

Alternative Flow	*Register Account : Invalid Fields*
Description	***User tries to register an account with one or more invalid details***
Actors	***User***
Pre-Condition	***User has Attempted to register for an account***
Main Flow	***1- The alternative flow starts after step number 3 for the main flow*** ***2- The system find one or more invalid details*** ***3- System redisplay the sign up page with invalid sign up massage*** ***4- The user re-enter the correct details***
Post-condition	***none***

Use Case Name	*Login*
Use Case Description	***A user login to the system to access the functionality of the system***
Actors	***User***
Pre-Condition	***User has been make a successful register or already have account***
Post - Condition	***User enter the system***
Main Flow	***1- User enter the user name / mail and his password*** ***2- User after fil his details click on login button*** ***3- The system will validate the user name and password*** ***4- The system will allow to the user to access the system***

Alternative Flow	*Login failed*
Description	***User fil invalid details***
Actors	***User***
Pre-Condition	***User have an account & user make unsuccessful login***
Main Flow	***1- The alternative flow starts after step number 3 for the main flow*** ***2- The system find one or two invalid details*** ***3- The system will mark on the invalid field*** ***4- The system will allow to the user to re-enter his details***
Post-condition	***Login successful***

Use Case Name	*Edit profile*
Use Case Description	***User can edit his profile by add or remove details***
Actors	***User***
Pre-Condition	***User already has account & user login to the system***
Post - Condition	***The system updated the user details***
Main Flow	1- User select edit profile button 2- System displays categories of profile (name , mail , profile photo …. And so on) 3- User select category 4- User updates detail, presses "Done" 5- System validates data as required, updates user profile

Use Case Name	*Make Schedule*
Use Case Description	***User can make a schedule to his medicines on a medical calendar***
Actors	***User***
Pre-Condition	***User login to the system***
Post - Condition	***System will store user details on the system calendar***
Main Flow	1- User click on calendar section button 2- System displays the User's medical calendar 3- User select the day that will fil it about the medicines or update it 4- System will store the new details

Use Case Name	*Enter Symptoms*
Use Case Description	***The user can enter his symptoms to the system to know his illness***
Actors	***User***
Pre-Condition	***User login to the system***
Post - Condition	***The system will give the user information about his illness such as name***
Main Flow	1- User click on the section to be able to enter his symptoms 2- User writes his symptoms to the system 3- The system takes the symptoms from the user 4- The system make processing on the user's answer 5- The system displays the final result to the user

Use Case Name	*Check Nearest Hospital*
Use Case Description	***The system will get information about the nearest hospitals around the user location*** ***And display that information to the user***
Actors	***User***
Pre-Condition	***User login to the system & open location service & allow system to get information about user location***
Post - Condition	***System display the needed information***
Main Flow	1- User click on the section to be able to get information about nearest hospitals 2- System will check the location service information 3- The system displays the final result to the user

Use Case Name	*Upload Medical Report*
Use Case Description	***User can upload images about his medical report to keep a copy of it on the system***
Actors	***User***
Pre-Condition	***User login to the system & allow the system to access the gallery***
Post - Condition	***System will store the photographic medical report***
Main Flow	1- User click on the section to be able to access the gallery 2- User selects images of his medical reports to be uploaded to the system 3- System upload the selected images 4- System display the uploaded images

Functional Requirements

1. User Registration and Profile Management

1. Allow users to create an account and register with the app.
2. Collect basic user information such as name, emergency number, email, phone number, age, and blood type.
3. Provide options to edit and update user profile information.

2. Disease Archive

1. Provide a dedicated section for each user to maintain their disease archive.
2. Allow users to add diseases to their archive with the following information:
3. Name: The name of the disease.
4. Start Date: The date when the user found out they have this disease.
5. Drug Name: The medicines that the user takes for this disease.
6. State: The current condition/state of the disease.
7. Enable users to view, edit, and delete, entries in their disease archive.

3. Disease Categorization

1. Categorize diseases in the archive into three main categories: typical diseases, chronic diseases, and genetic diseases.
2. Provide options for users to assign diseases to the appropriate category during entry creation/editing.

4. Reports Page

1. Include a dedicated page within the app for storing and managing medical examination reports.
2. Allow users to upload and save photocopies or digital copies of their medical examination results.
3. Support file formats commonly used for medical reports (e.g., PDF, JPEG, PNG).
4. Provide options to view, download, delete, and organize reports.

5. Notifications and Reminders

1. Allow users to set reminders for taking medications, upcoming medical appointments, or other relevant events.
2. Send push notifications or alerts to remind users of scheduled tasks or events.

6. Symptom Checker

1. Users should be able to enter their symptoms into the app.
2. The app should analyze the symptoms and provide information about the possible disease(s) the user might have.
3. The response should include suggestions for further action, such as seeking medical attention or self-care tips.

7. Search and Filter Functionality

1. Enable users to search and filter their disease archive based on various parameters, such as disease name, category, start date, or current state.
2. Provide a user-friendly interface to facilitate efficient search and filtering.

Non-functional requirements

1. Performance

1. The app should provide fast response times and smooth performance, ensuring minimal delays in loading screens, data retrieval, and processing.
2. The ML-based disease diagnosis feature should deliver quick and accurate predictions, providing near real-time results to users.

2. Scalability

1. The app should be designed to handle a growing number of users, disease entries, and medical reports without significant performance degradation.
2. The backend infrastructure should be scalable to accommodate increased user activity and data storage requirements.

3. Compatibility

1. Develop the app to be compatible with a range of mobile devices and operating systems, ensuring a consistent user experience across different platforms.
2. Test the app on various devices, screen sizes, and resolutions to ensure proper functionality and visual appeal.

4. Reliability and Availability

1. The app should be highly reliable, minimizing downtime and ensuring continuous availability for users.

5. Usability and User Experience

1. Design the app with a user-friendly interface, making it intuitive and easy to navigate for users with varying levels of technological proficiency.
2. Use appropriate visual cues, such as icons and color schemes, to enhance the app's usability and aid users in quickly understanding information.
3. Ensure clear and concise instructions and labels throughout the app to guide users in entering symptoms, managing their disease archive, and accessing medical reports.

6. Security

1. Implement robust security measures to protect user data, including personal information, medical records, and symptom inputs.
2. Utilize industry-standard encryption techniques for data transmission and storage to prevent unauthorized access and to ensure that user data, including personal information and medical reports, is encrypted and stored securely.
3. Implement secure authentication mechanisms to protect user data.

7. Error Handling and Logging

1. Implement robust error-handling mechanisms to gracefully handle unexpected errors or exceptions that may occur during app usage.

8. Maintenance and Upgrades

1. Plan for regular maintenance and updates to address bug fixes, security patches, and feature enhancements.
2. Provide mechanisms to deliver updates seamlessly to users, ensuring a smooth transition between app versions without data loss or disruptions in functionality.

9. Documentation and Support:

1. Provide comprehensive documentation, including user guides and FAQs, to assist users in understanding the app's features and functionality.
2. Offer responsive customer support channels, such as email or in-app messaging, to address user inquiries, issues, and feedback in a timely manner.

10. Accessibility

1. Ensure that the app is accessible to users with disabilities, complying with accessibility standards and guidelines.
2. Support accessibility features such as screen readers, magnification, and color contrast adjustments to accommodate users with visual impairments or other accessibility needs.

Constraints

Regulatory compliance
The proposed system is compliant with regulations and standards established by the relevant authorities

Privacy and security
The proposed system guarantees the privacy and security of patient data, in addition to adhering to applicable regulations.

Accuracy and reliability
The proposed system delivers precise and dependable information to healthcare professionals and patients

Usability
The proposed system has been designed with user-friendliness in mind and is easily accessible to healthcare professionals and patients alike."

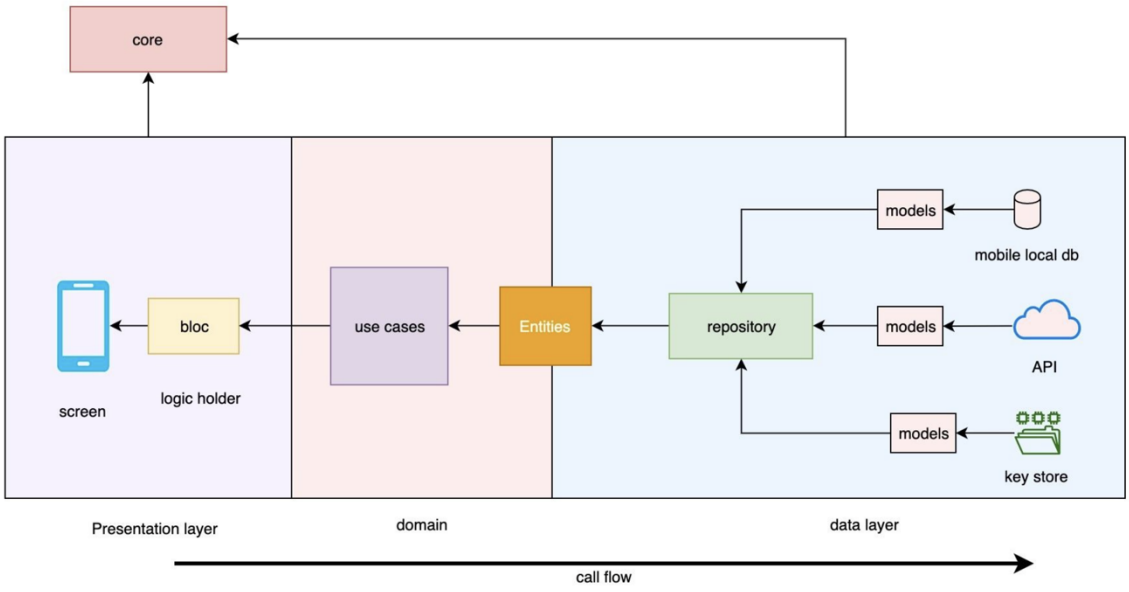
Interoperability
The proposed system can integrate with various healthcare systems and technologies, ensuring interoperability

Performance and scalability
The proposed system has the capability to manage substantial amounts of data and traffic and maintain consistent performance under diverse circumstances

CHAPTER 3: Software Design

Software Architecture

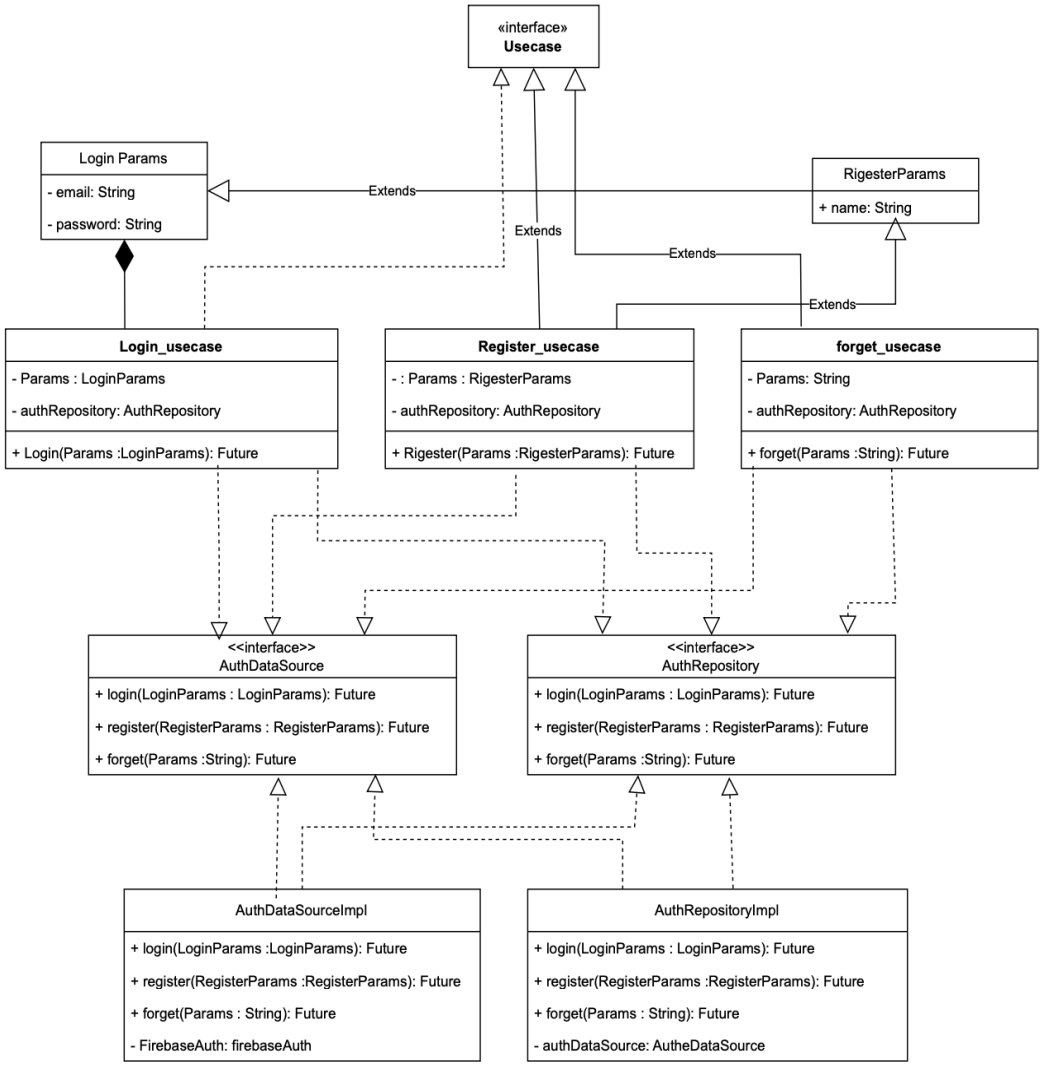
Clean architecture is used in this system, which is a design pattern that aims to make applications more maintainable and easier to develop. This pattern is implemented by dividing the application into different layers that work independently of each other and clearly defining the responsibilities and tasks of each. Our application is used in this context to implement the clean architecture pattern. for further details check the section Application and technology used in this system



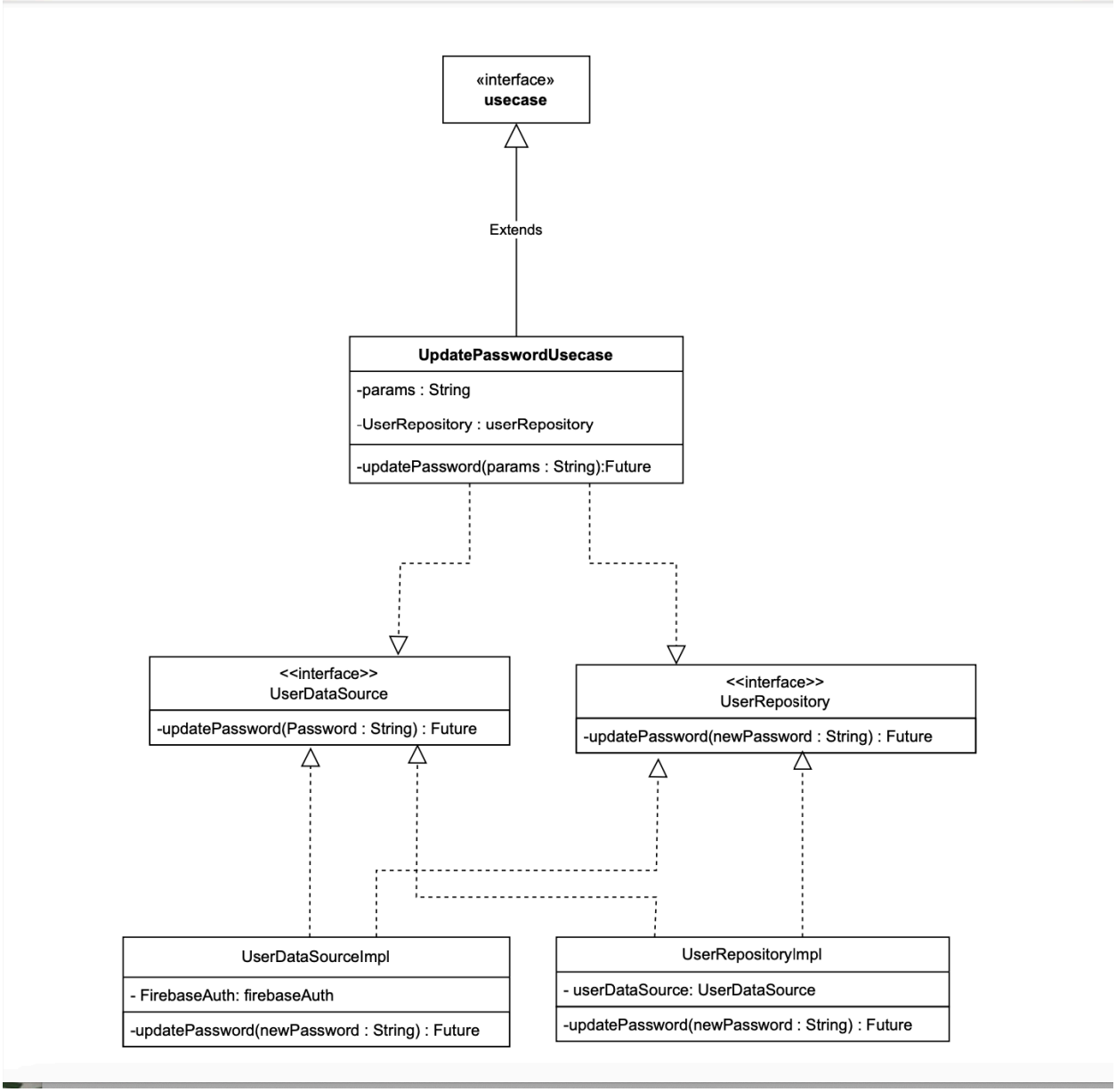
UML Diagrams

Class diagram

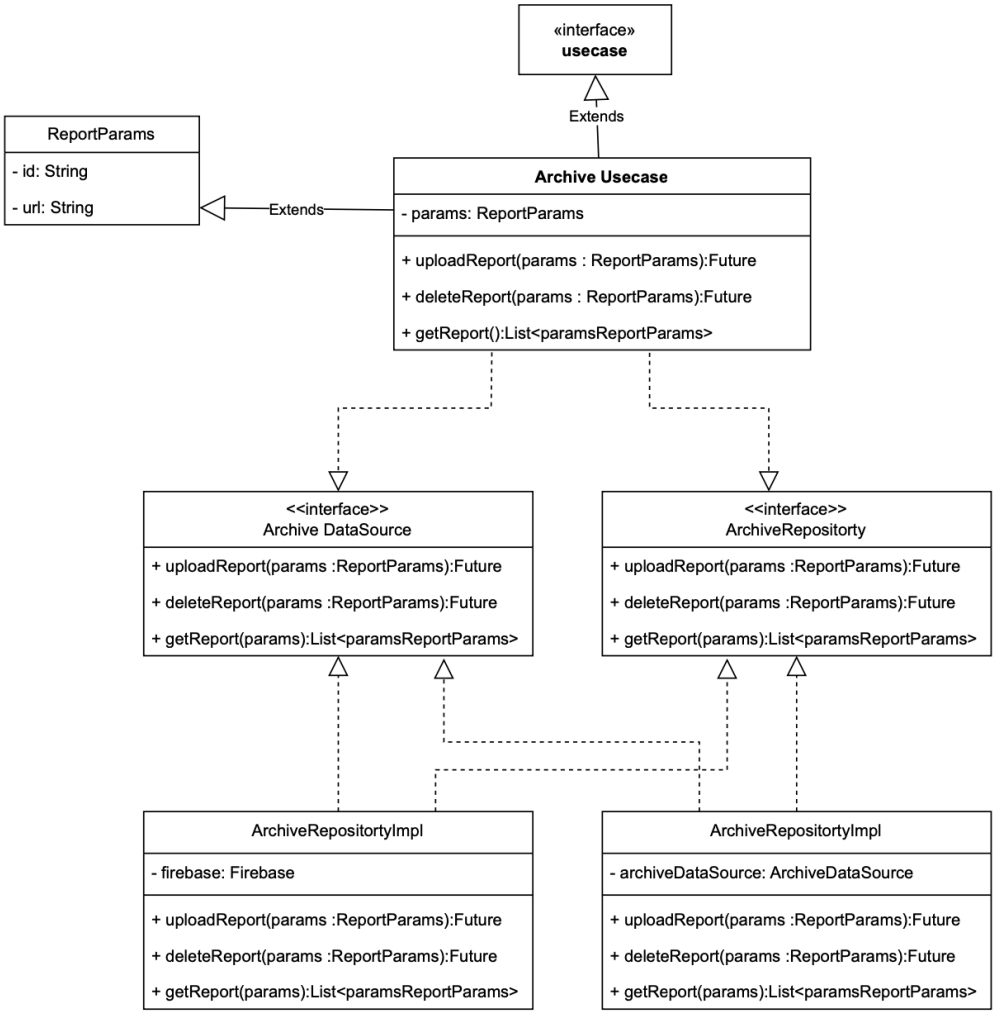
- **Authentication Diagram**



- **User diagram**

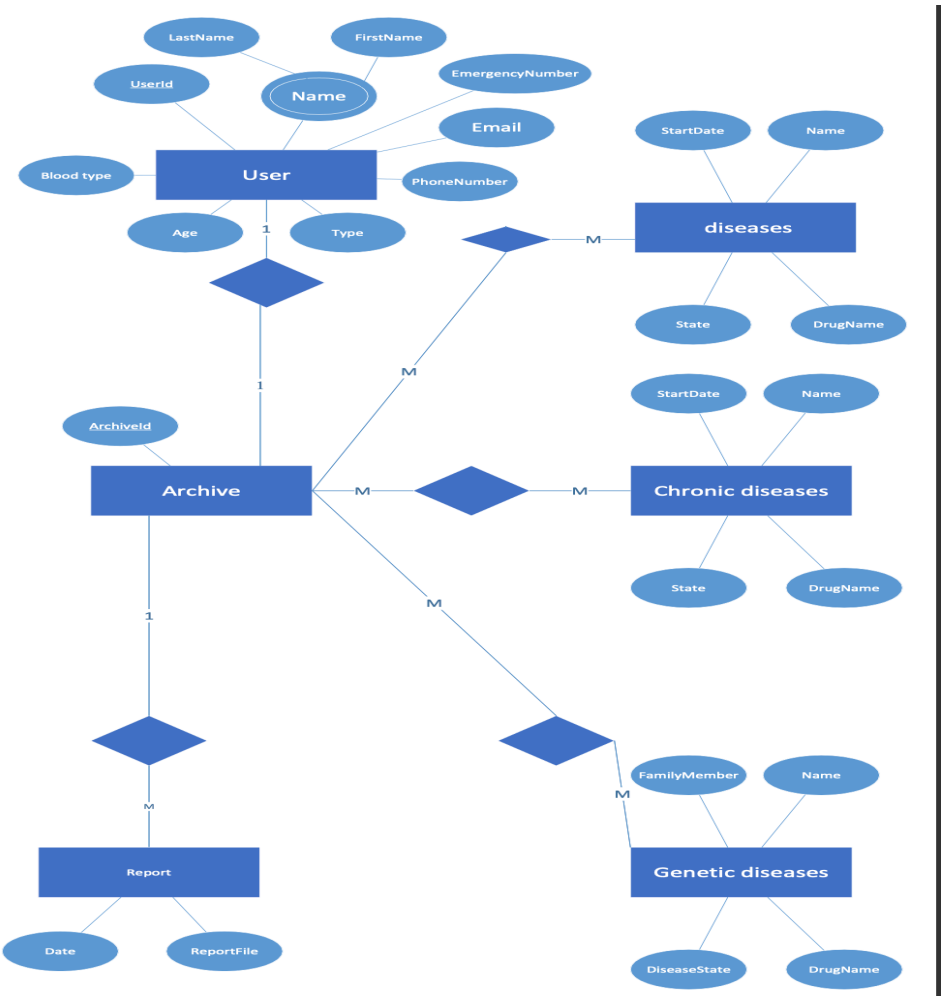


- **EHR Diagram**


«interface»
usecase
Extends
ReportParams
- id: String
- url: String
Extends
Archive Usecase
- params: ReportParams
+ uploadReport(params : ReportParams):Future
+ deleteReport(params : ReportParams):Future
+ getReport():List<paramsReportParams>
<<interface>>
Archive DataSource
+ uploadReport(params :ReportParams):Future
+ deleteReport(params :ReportParams):Future
+ getReport(params):List<paramsReportParams>
<<interface>>
ArchiveRepositorty
+ uploadReport(params :ReportParams):Future
+ deleteReport(params :ReportParams):Future
+ getReport(params):List<paramsReportParams>
ArchiveRepositortyImpl
- firebase: Firebase
+ uploadReport(params :ReportParams):Future
+ deleteReport(params :ReportParams):Future
+ getReport(params):List<paramsReportParams>
ArchiveRepositortyImpl
- archiveDataSource: ArchiveDataSource
+ uploadReport(params :ReportParams):Future
+ deleteReport(params :ReportParams):Future
+ getReport(params):List<paramsReportParams>

Database Design

ER diagram



User interface design

UI/UX Case Study

It is important for UX recruiters because they provide insight into your design process, the methods you use, and your thought processes.

Target Audience

- **Adults**

When the patient cannot provide enough money for the examination, especially in minor diseases

- **Doctors**

This application is used as an assistant to the doctor when it comes to diagnosis, particularly in critical situations

Overview

The Approach

User-Centric Design It is based on the understanding of a user, their demands, priorities, and experiences, when used, Finally it encompasses different research and design tactics that involve the user in each stage of the design process.

provide features which have very high usability and this includes:

1- Specify the context of the use
2- Specify requirements
3- Evaluate Features
4- Create design solutions and development

UCD asks questions about users and their tasks and goals, then uses the findings to make decisions about development and design.

1- Who are the users of the Application?

2- What are the users' experience levels with the Application, and with similar Applications?

3- How do users think the Application should work?

User-friendly interface

The application has a user-friendly and intuitive interface, which reduces the possibility of errors during transactions.

Collaboration

successful collaboration in UX/UI design for this App provides a deep understanding of the user's needs, as well as the ability to work effectively with diverse groups of stakeholders.

Design Thinking Process

The Design Thinking process is a problem-solving approach that is commonly used in UX/UI design. It involves understanding the needs and motivations of the users, generating ideas, prototyping, testing, and refining the solutions to ultimately create a user-centered design.

1-Empathy: 1-User Research,2- User Interview,3- Entrant Analysis

2-Define : 1- User Persona,2- User Journey Map,3- Empathy Map

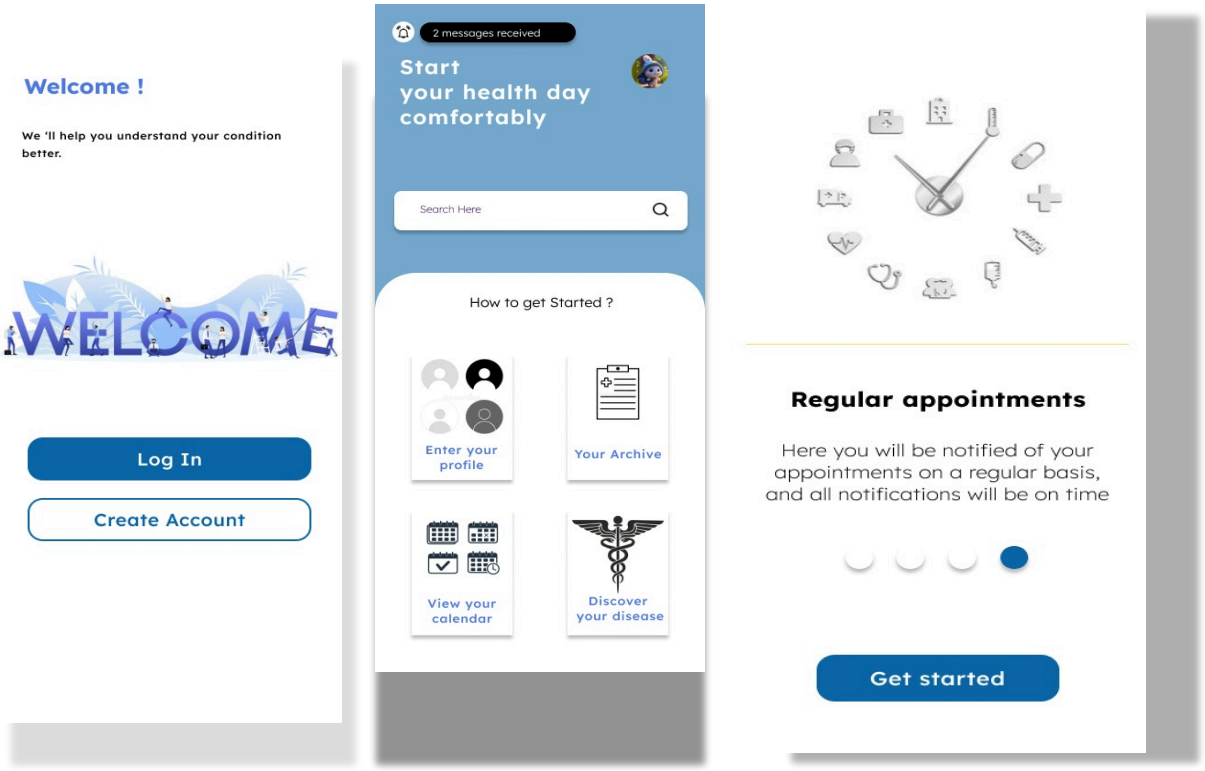
3-Ideate: 1- Brainstorming,2- User Flow

4-Design/Prototype : 1- sketches,2- wireframes,3- clickable mockups,4- Visual Design.

5-Test :1- Check Usability, 2- Survey Insight , 3- Improvements

Visual Design

Some of the application's design



CHAPTER 4: Implementation

Disease Prediction Using Machine Learning Model Documentation

Disease Prediction Using Machine Learning Model Documentation

1. Introduction

With the advancement in technology, Machine Learning is becoming more popular and commonly used technology by industry experts for solving problems faced in real life. Machine Learning is the scientific study of algorithms and statistical models that computers use to perform a specific task without using explicit instructions, relying on patterns and inference instead. The different machine learning algorithms are supervised, unsupervised, and reinforcement learning algorithms. In supervised learning, we use known or labeled data for the training data. Since the data is known, the learning is, therefore, supervised, i.e., directed into successful execution. Machine Learning is also used by the healthcare industry to bring advancement in their techniques so that they can provide better services to their patients. The disease prediction system predicts diseases based on the patient's symptoms.

2. Proposed system

In the proposed system, a disease prediction model is built using a Machine Learning algorithm that is, a decision tree Algorithm. Based on the symptoms that are input by the user, the disease is predicted. The system processes the symptoms provided by the user as input and gives the output as the probability of the disease. Decision Tree classifier is used in the prediction of the disease which is a supervised machine learning algorithm.

3. Steps to develop the model

1.1 Analyzing the problem statement & requirements

Analyze the problem in terms of what we want to predict and what kind of observation data we have to make those predictions. Predictions are generally a label or a target answer; it may be a yes/no label (binary classification) or a category (multiclass classification) or a real number (regression). In this case, the target label is the disease

1.2 Collect and clean the data

Identify what kind of historical data we have for prediction modeling, the next step is to collect the data from datasets or any other data sources.

The dataset used in this project

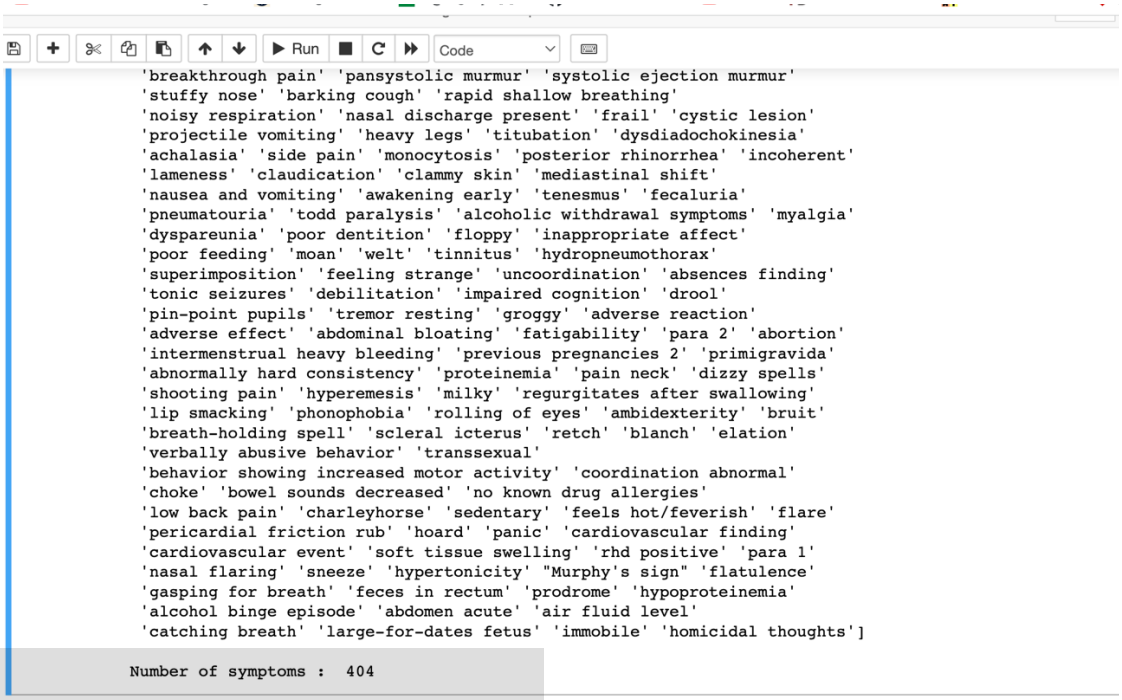
Getting to know the data

Oxford Dictionary defines a dataset as “a collection of data that is treated as a single unit by a computer”. This means that a dataset contains many separate pieces of data but can be used to train an algorithm to find predictable patterns inside the whole dataset.

The dataset used In this project is obtained from Columbia University. The table below is a knowledge database of disease-symptom associations generated by an automated method based on information in textual discharge summaries of patients at New York Presbyterian Hospital admitted in 2004. The first column shows the disease, the second the number of discharge summaries containing a positive and current mention of the disease, and the associated symptom. Associations for the 150 most frequent diseases based on these notes were computed and the symptoms are shown ranked based on the strength of association. The method used the MedLEE natural language processing system to obtain UMLS codes for diseases and symptoms from the notes; then statistical methods based on frequencies and co-occurrences were used to obtain the associations. A more detailed description of the automated method can be found in Wang X, Chused A, Elhadad N, Friedman C, and Markatou M. Automated knowledge acquisition from clinical reports. AMIA Annu Symp Proc. 2008. p. 783-7. PMCID: PMC2656103.

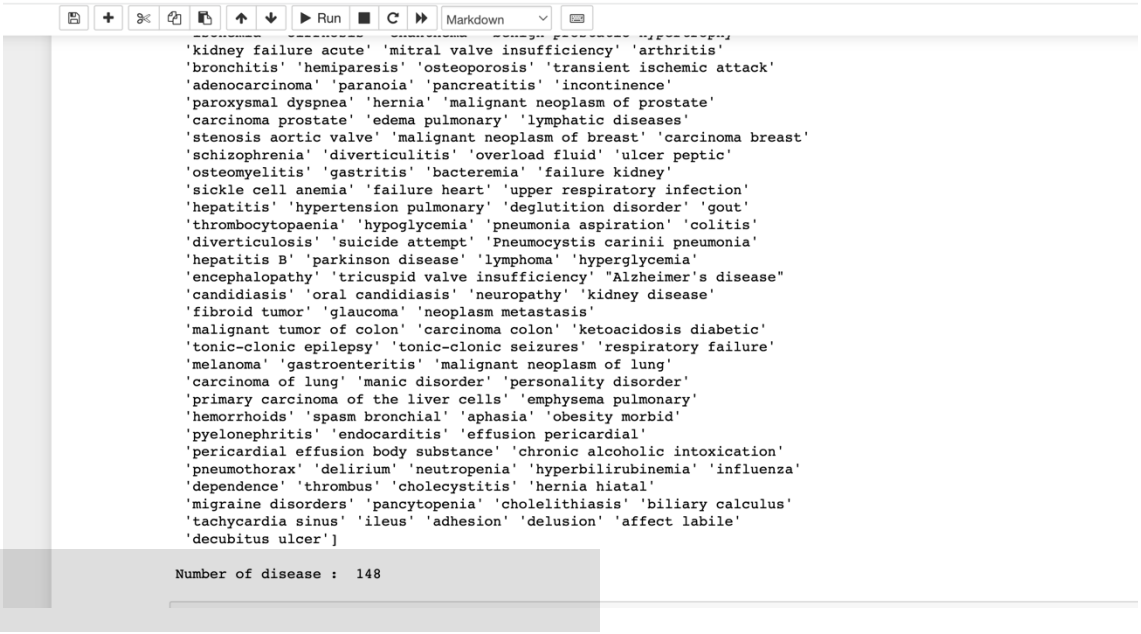
Disease	Count of Disease Occurrence	Symptom
UMLS:C0020538_hypertensive disease	3363	UMLS:C0008031_pain chest
		UMLS:C0392680_shortness of breath
		UMLS:C0012833_dizziness
		UMLS:C0004093_asthenia
		UMLS:C0085639_fall
		UMLS:C0039070_syncope
		UMLS:C0042571_vertigo
		UMLS:C0038990_sweat^UMLS:C0700590_sweating increased
		UMLS:C0030252_palpitation
		UMLS:C0027497_nausea
		UMLS:C0002962_angina pectoris
		UMLS:C0438716_pressure chest
UMLS:C0011847_diabetes	1421	UMLS:C0032617_polyuria
		UMLS:C0085602_polydypsia
		UMLS:C0392680_shortness of breath
		UMLS:C0008031_pain chest
		UMLS:C0004093_asthenia
		UMLS:C0027497_nausea
		UMLS:C0085619_orthopnea
		UMLS:C0034642_rale
		UMLS:C0038990_sweat^UMLS:C0700590_sweating increased
		UMLS:C0241526_unresponsiveness
		UMLS:C0856054_mental status changes
		UMLS:C0042571_vertigo
		UMLS:C0042963_vomiting
		UMLS:C0553668_labored breathing
UMLS:C0011570_depression mental^UMLS:C0011581_depressive disorder	1337	UMLS:C0424000_feeling suicidal
		UMLS:C0438696_suicidal
		UMLS:C0233762_hallucinations auditory
		UMLS:C0150041_feeling hopeless
		UMLS:C0424109_weepiness
		UMLS:C0917801_sleeplessness
		UMLS:C0424230_motor retardation
		UMLS:C0022107_irritable mood
		UMLS:C0312422_blackout

The dataset contains 404 symptoms assisted with 148 disease
404 symptoms are shown below:



```
'breakthrough pain' 'pansystolic murmur' 'systolic ejection murmur'
'stuffy nose' 'barking cough' 'rapid shallow breathing'
'noisy respiration' 'nasal discharge present' 'frail' 'cystic lesion'
'projectile vomiting' 'heavy legs' 'titubation' 'dysdiadochokinesia'
'achalasia' 'side pain' 'monocytosis' 'posterior rhinorrhea' 'incoherent'
'lameness' 'claudication' 'clammy skin' 'mediastinal shift'
'nausea and vomiting' 'awakening early' 'tenesmus' 'fecaluria'
'pneumatouria' 'todd paralysis' 'alcoholic withdrawal symptoms' 'myalgia'
'dyspareunia' 'poor dentition' 'floppy' 'inappropriate affect'
'poor feeding' 'moan' 'welt' 'tinnitus' 'hydropneumothorax'
'superimposition' 'feeling strange' 'uncoordination' 'absences finding'
'tonic seizures' 'debilitation' 'impaired cognition' 'drool'
'pin-point pupils' 'tremor resting' 'groggy' 'adverse reaction'
'adverse effect' 'abdominal bloating' 'fatigability' 'para 2' 'abortion'
'intermenstrual heavy bleeding' 'previous pregnancies 2' 'primigravida'
'abnormally hard consistency' 'proteinemia' 'pain neck' 'dizzy spells'
'shooting pain' 'hyperemesis' 'milky' 'regurgitates after swallowing'
'lip smacking' 'phonophobia' 'rolling of eyes' 'ambidexterity' 'bruit'
'breath-holding spell' 'scleral icterus' 'retch' 'blanch' 'elation'
'verbally abusive behavior' 'transsexual'
'behavior showing increased motor activity' 'coordination abnormal'
'choke' 'bowel sounds decreased' 'no known drug allergies'
'low back pain' 'charleyhorse' 'sedentary' 'feels hot/feverish' 'flare'
'pericardial friction rub' 'hoard' 'panic' 'cardiovascular finding'
'cardiovascular event' 'soft tissue swelling' 'rhd positive' 'para 1'
'nasal flaring' 'sneeze' 'hypertonicity' "Murphy's sign" 'flatulence'
'gasping for breath' 'feces in rectum' 'prodrome' 'hypoproteinemia'
'alcohol binge episode' 'abdomen acute' 'air fluid level'
'catching breath' 'large-for-dates fetus' 'immobile' 'homicidal thoughts']

Number of symptoms :  404
```

148 diseases are shown below:



```
'kidney failure acute' 'mitral valve insufficiency' 'arthritis'
'bronchitis' 'hemiparesis' 'osteoporosis' 'transient ischemic attack'
'adenocarcinoma' 'paranoia' 'pancreatitis' 'incontinence'
'paroxysmal dyspnea' 'hernia' 'malignant neoplasm of prostate'
'carcinoma prostate' 'edema pulmonary' 'lymphatic diseases'
'stenosis aortic valve' 'malignant neoplasm of breast' 'carcinoma breast'
'schizophrenia' 'diverticulitis' 'overload fluid' 'ulcer peptic'
'osteomyelitis' 'gastritis' 'bacteremia' 'failure kidney'
'sickle cell anemia' 'failure heart' 'upper respiratory infection'
'hepatitis' 'hypertension pulmonary' 'deglutition disorder' 'gout'
'thrombocytopaenia' 'hypoglycemia' 'pneumonia aspiration' 'colitis'
'diverticulosis' 'suicide attempt' 'Pneumocystis carinii pneumonia'
'hepatitis B' 'parkinson disease' 'lymphoma' 'hyperglycemia'
'encephalopathy' 'tricuspid valve insufficiency' "Alzheimer's disease"
'candidiasis' 'oral candidiasis' 'neuropathy' 'kidney disease'
'fibroid tumor' 'glaucoma' 'neoplasm metastasis'
'malignant tumor of colon' 'carcinoma colon' 'ketoacidosis diabetic'
'tonic-clonic epilepsy' 'tonic-clonic seizures' 'respiratory failure'
'melanoma' 'gastroenteritis' 'malignant neoplasm of lung'
'carcinoma of lung' 'manic disorder' 'personality disorder'
'primary carcinoma of the liver cells' 'emphysema pulmonary'
'hemorrhoids' 'spasm bronchial' 'aphasia' 'obesity morbid'
'pyelonephritis' 'endocarditis' 'effusion pericardial'
'pericardial effusion body substance' 'chronic alcoholic intoxication'
'pneumothorax' 'delirium' 'neutropenia' 'hyperbilirubinemia' 'influenza'
'dependence' 'thrombus' 'cholecystitis' 'hernia hiatal'
'migraine disorders' 'pancytopenia' 'cholelithiasis' 'biliary calculus'
'tachycardia sinus' 'ileus' 'adhesion' 'delusion' 'affect labile'
'decubitus ulcer']

Number of disease :  148
```

Cleaning and Processing data stage

Cleaning is the most important step in a machine-learning project. The quality of our data determines the quality of our machine-learning model. So it is always necessary to clean the data before feeding it to the model for training.

Data was first loaded from
https://impact.dbmi.columbia.edu/~friedma/Projects/DiseaseSymptomKB/index.html

As we mentioned before, the dataset contains both diseases and their corresponding symptoms

```
In [34]: data.head()
```

Out[34]:

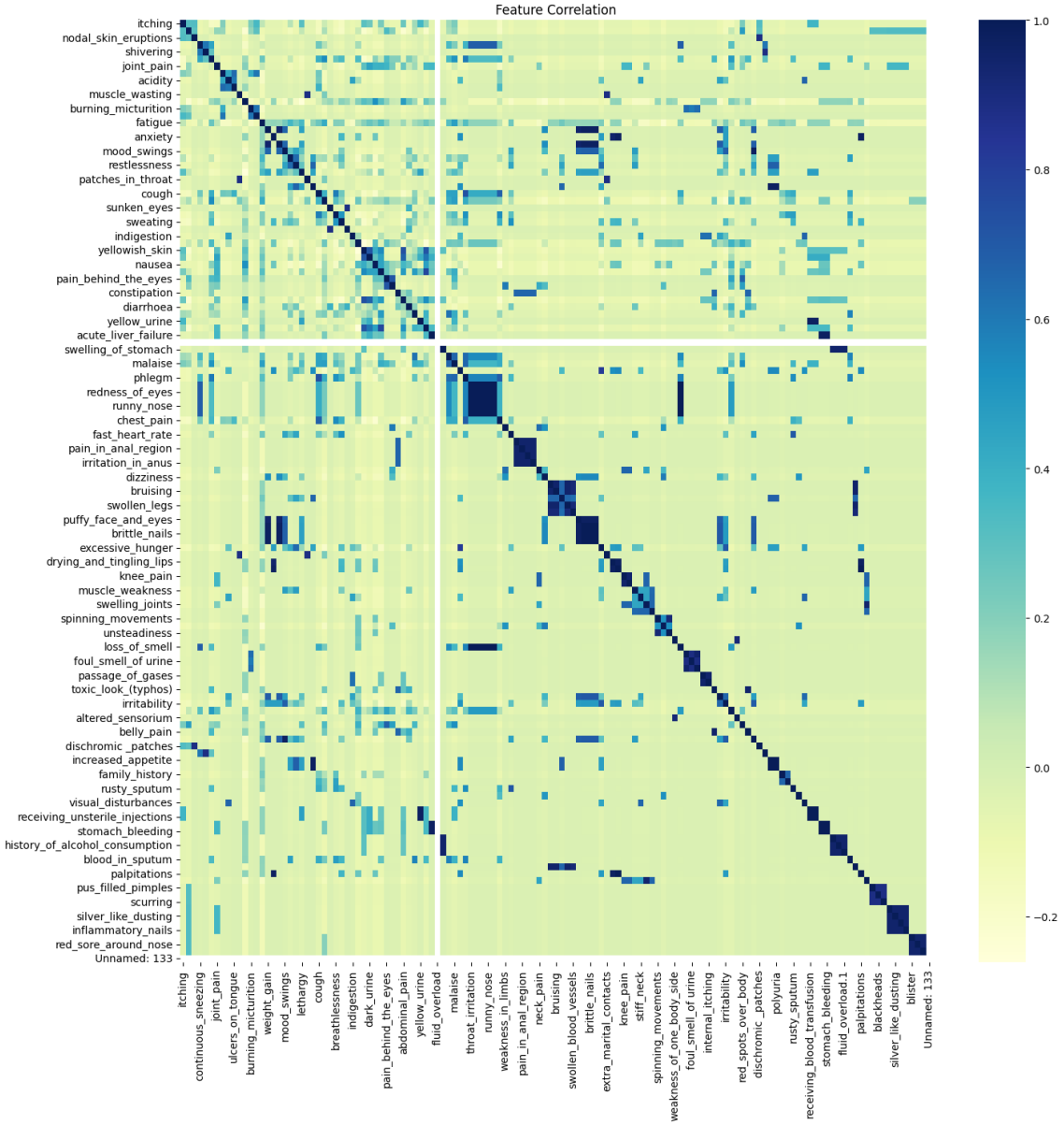
	Disease	Count of Disease Occurrence	Symptom
0	UMLS:C0020538_hypertensive disease	3363.0	UMLS:C0008031_pain chest
1	UMLS:C0020538_hypertensive disease	3363.0	UMLS:C0392680_shortness of breath
2	UMLS:C0020538_hypertensive disease	3363.0	UMLS:C0012833_dizziness
3	UMLS:C0020538_hypertensive disease	3363.0	UMLS:C0004093_asthenia
4	UMLS:C0020538_hypertensive disease	3363.0	UMLS:C0085639_fall

And then we extracted only the diseases, symptoms

```
['hypertensive disease' 'diabetes' 'depression mental'
 'depressive disorder' 'coronary arteriosclerosis'
 'coronary heart disease' 'pneumonia' 'failure heart congestive'
 'accident cerebrovascular' 'asthma' 'myocardial infarction'
 'hypercholesterolemia' 'infection' 'infection urinary tract' 'anemia'
 'chronic obstructive airway disease' 'dementia' 'insufficiency renal'
 'confusion' 'degenerative polyarthritis' 'hypothyroidism' 'anxiety state'
 'malignant neoplasms' 'primary malignant neoplasm'
 'acquired immuno-deficiency syndrome' 'HIV' 'hiv infections' 'cellulitis'
 'gastroesophageal reflux disease' 'septicemia' 'systemic infection'
 'sepsis (invertebrate)' 'deep vein thrombosis' 'dehydration' 'neoplasm'
 'embolism pulmonary' 'epilepsy' 'cardiomyopathy' 'chronic kidney failure'
 'carcinoma' 'hepatitis C' 'peripheral vascular disease'
 'psychotic disorder' 'hyperlipidemia' 'bipolar disorder' 'obesity'
 'ischemia' 'cirrhosis' 'exanthema' 'benign prostatic hypertrophy'
 'kidney failure acute' 'mitral valve insufficiency' 'arthritis'
 'bronchitis' 'hemiparesis' 'osteoporosis' 'transient ischemic attack'
 'adenocarcinoma' 'paranoia' 'pancreatitis' 'incontinence'
 'paroxysmal dyspnea' 'hernia' 'malignant neoplasm of prostate'
 'carcinoma prostate' 'edema pulmonary' 'lymphatic diseases'
 'stenosis aortic valve' 'malignant neoplasm of breast' 'carcinoma breast'
 'schizophrenia' 'diverticulitis' 'overload fluid' 'ulcer peptic'
 'osteomyelitis' 'gastritis' 'bacteremia' 'failure kidney'
 'sickle cell anemia' 'failure heart' 'upper respiratory infection'
 'hepatitis' 'hypertension pulmonary' 'deglutition disorder' 'gout'
 'thrombocytopaenia' 'hypoglycemia' 'pneumonia aspiration' 'colitis'
 'diverticulosis' 'suicide attempt' 'Pneumocystis carinii pneumonia'
 'hepatitis B' 'parkinson disease' 'lymphoma' 'hyperglycemia'
 'encephalopathy' 'tricuspid valve insufficiency' "Alzheimer's disease"
 'candidiasis' 'oral candidiasis' 'neuropathy' 'kidney disease'
 'fibroid tumor' 'glaucoma' 'neoplasm metastasis'
 'malignant tumor of colon' 'carcinoma colon' 'ketoacidosis diabetic'
```

Correlation coefficient

In this stage, we performed the correlation coefficient to get more insights of the dataset, The correlation coefficient is the unit of measurement used to calculate the intensity in the linear relationship between the variables involved in correlation analysis.



Values tending towards light yellow are negatively correlated, and those tending towards dark blue are positively correlated. The lighter the color, the closer the value is to 0.

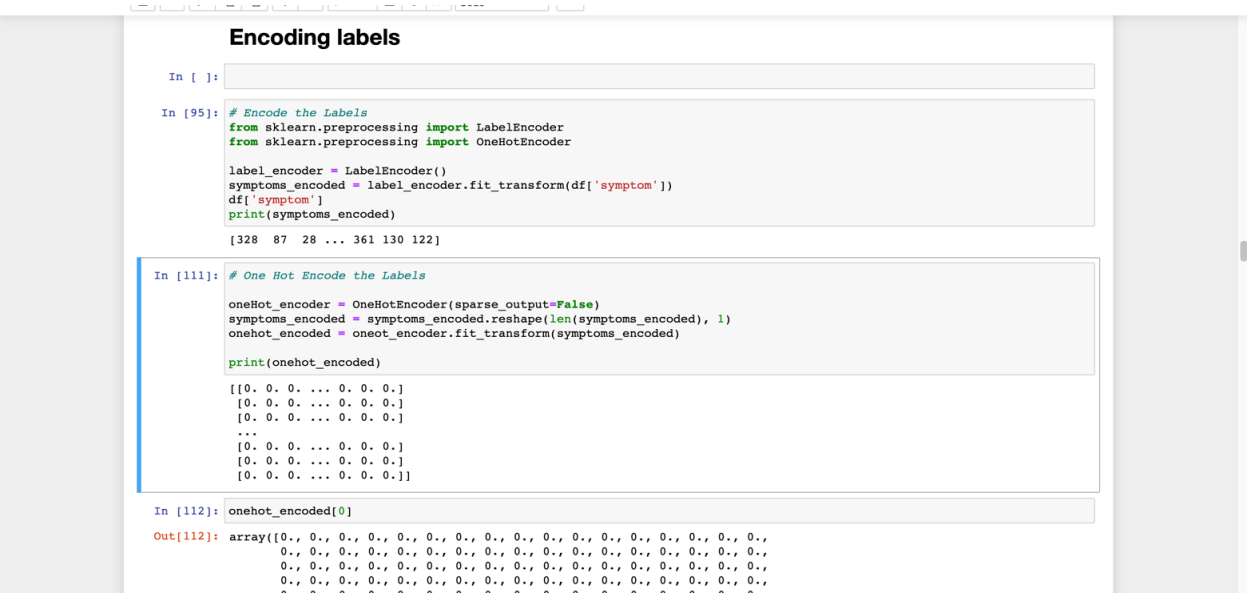
1.3 Prepare data for ML application

Transform the data in a form that the Machine Learning system can understand.

Encoding labels

Label Encoding is a technique that is used to convert categorical columns into numerical ones so that they can be fitted by machine learning models which only take numerical data. It is an important pre-processing step in a machine-learning project.

In our dataset all the columns should be numerical, the target column i.e. prognosis is a string type and is encoded to numerical form using a label encoder



Encoding labels

In [ ]:

In [95]:
```
# Encode the Labels
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder

label_encoder = LabelEncoder()
symptoms_encoded = label_encoder.fit_transform(df['symptom'])
df['symptom']
print(symptoms_encoded)
```

```
[328  87  28 ... 361 130 122]
```

In [111]:
```
# One Hot Encode the Labels

oneHot_encoder = OneHotEncoder(sparse_output=False)
symptoms_encoded = symptoms_encoded.reshape(len(symptoms_encoded), 1)
onehot_encoded = oneot_encoder.fit_transform(symptoms_encoded)

print(onehot_encoded)
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

In [112]:
```
onehot_encoded[0]
```

Out[112]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
```

In [117]:
```
df_ohe.head()
```

Out[117]:

	shortness of breath	dizziness	asthenia	fall	syncope	vertigo	sweat	sweating increased	palpitation	nausea	...	feces in rectum	prodrome	hypoproteinemia	alcohol binge episode	abdomen acute	a flui lev
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0

Since each disease has multiple symptoms, we combine all symptoms per disease per row

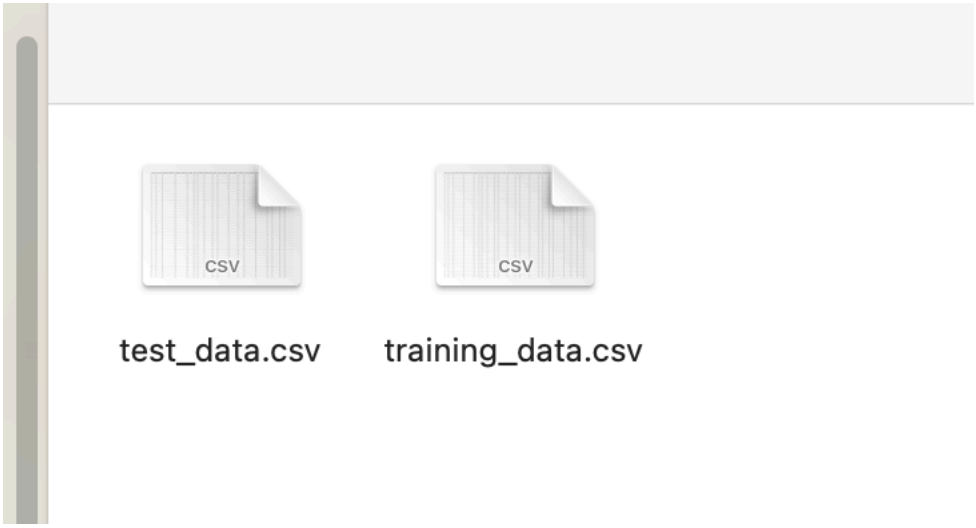
```
df_concat = df_concat.groupby('disease').sum()
df_concat = df_concat.reset_index()
df_concat[:5]
```

Out[126]:

	disease	shortness of breath	dizziness	asthenia	fall	syncope	vertigo	sweat	sweating increased	palpitation	...	feces in rectum	prodrome	hypoproteinemia	a ep
0	Alzheimer's disease	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
1	HIV	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
2	Pneumocystis carinii pneumonia	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
3	accident cerebrovascular	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
4	acquired immuno-deficiency syndrome	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	

1.4 Train the model Before training the model

It is essential to split the data into training and evaluation sets, as we need to monitor how well a model generalizes to unseen data. Now, the algorithm will learn the pattern and mapping between the feature and the label.



Model Building

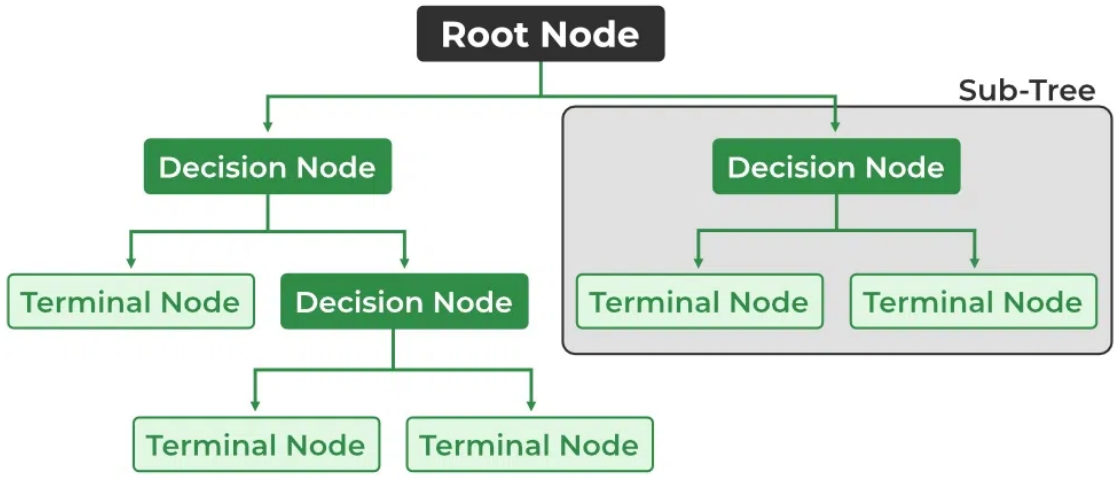
The algorithm used in this project

In this project, we have used the decision tree algorithm, A decision tree is one of the most powerful tools of supervised learning algorithms used for both classification and regression tasks.

A decision tree is a structure that can be used to divide up a large collection of records into successfully smaller sets of records by applying a sequence of simple decision trees, with each successive division, the members of the resulting sets become more and more similar to each other. It uses a set of rules for dividing a large heterogeneous population into smaller, more homogeneous (mutually exclusive) groups concerning a particular target.

During training, the Decision Tree algorithm selects the best attribute to split the data based on a metric such as entropy or Gini impurity, which measures the level of impurity or randomness in the subsets. The goal is to find the attribute that maximizes the information gain or the reduction in impurity after the split. The target variable is usually categorical and the decision tree is used either to calculate the probability that a given record belongs to each of the categories or to classify the record by assigning it to the most likely class (or category).

In this disease prediction system, a decision tree divides the symptoms as per their category and reduces the dataset difficulty.



The formula for entropy:

$$-\sum_{i=1}^{C} P(x_i) log_b P(x_i)$$

Information gain will use the following formula:

$$IG(T,A) = Entropy(T) - \sum_{v \epsilon A} \frac{|T_v|}{T} \cdot Entropy(T_v)$$

After gathering and cleaning the data, the data is ready and can be used to train a machine-learning model. We will be using this cleaned data to train the decision tree classifier. We will be using a confusion matrix to determine the quality of the models.

We applied this model twice, once on the whole dataset (the dataset that contains 404 symptoms and 148 diseases), and on a sample obtained from the whole data, the sample contains the most frequent diseases (133 symptoms and 42 diseases)

Model Training

```
In [216]: from sklearn.model_selection import train_test_split
          from sklearn.naive_bayes import MultinomialNB
          from sklearn import tree
          from sklearn.tree import export_graphviz
          from sklearn.metrics import accuracy_score
```

```
In [247]: # Train Test Split
          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
In [220]: X_test
```

asthenia	fall	syncope	vertigo	sweat	sweating increased	palpitation	nausea	...	feces in rectum	prodrome	hypoproteinemia	alcohol binge episode	abdomen acute	air fluid level	catching breath	large-for-dates fetus	immobile	homicidal thoughts
1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0

```
In [221]: y_test
```

Model performance on the whole dataset

```
Classification Acc  0.9745762711864406
Pred: depression mental
Actual: depressive disorder

Pred: septicemia
Actual: systemic infection

Pred: coronary arteriosclerosis
Actual: coronary heart disease
```

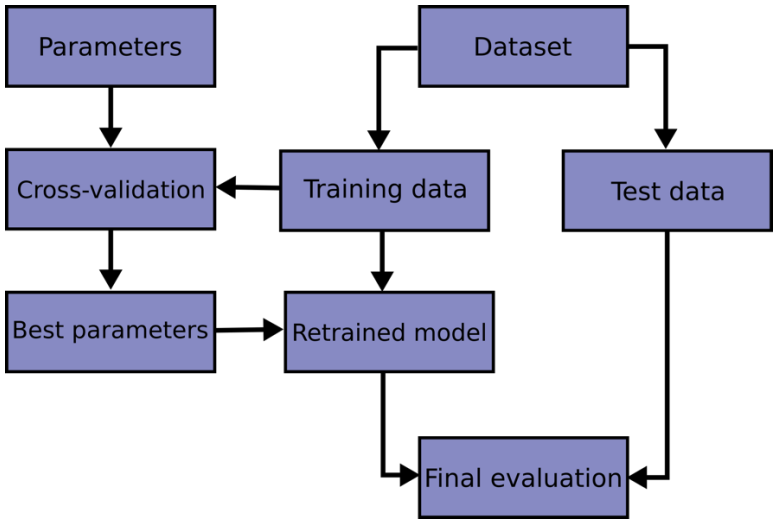
unpredicted labels

Model performance on the sample dataset

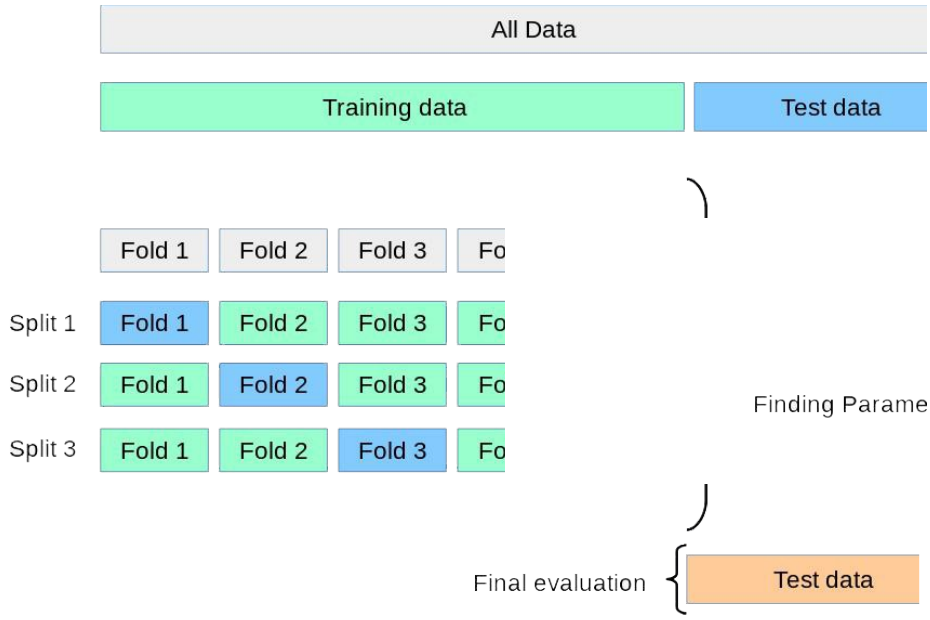
```
Length of Training Data:  (4920, 134)
Training Features:  (4920, 132)
Training Labels:  (4920,)
Length of Test Data:  (42, 133)
Test Features:  (42, 132)
Test Labels:  (42,)



Number of Training Features: 3296       Number of Training Labels: 3296
Number of Validation Features: 1624     Number of Validation Labels: 1624

Training Accuracy:  1.0
```

Cross-validation: evaluating estimator performance

Learning the parameters of a prediction function and testing it on the same data is a methodological mistake: a model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data. This situation is called **overfitting**. To avoid it, it is common practice when performing a (supervised) machine learning experiment to hold out part of the available data as a **test set** `X_test, y_test`. Note that the word "experiment" is not intended to denote academic use only, because even in commercial settings machine learning usually starts out experimentally. Here is a flowchart of typical cross-validation workflow in model training. The best parameters can be determined by grid search techniques.



A solution to this problem is a procedure called cross-validation (CV for short). A test set should still be held out for final evaluation, but the validation set is no longer needed when doing CV. In the basic approach, called *k*-fold CV, the training set is split into *k* smaller sets. The following procedure is followed for each of the *k* "folds":



3 K-fold used in this model:

```
Validation Confusion Matrix:
 [[47  0  0 ...  0  0  0]
 [ 0 50  0 ...  0  0  0]
 [ 0  0 41 ...  0  0  0]
 ...
 [ 0  0  0 ... 35  0  0]
 [ 0  0  0 ...  0 39  0]
 [ 0  0  0 ...  0  0 44]]

Cross Validation Score:
 [0.96678967 0.92791128 0.95748614]
```

1.5 Evaluate and improve model accuracy

Accuracy is the percentage of correct classifications that a trained machine learning model achieves, i.e., the number of correct predictions divided by the total number of predictions across all classes. It is often abbreviated as ACC.

Starting from the confusion matrix, we can see this relationship by deriving the statistical formula for accuracy. Note that we do so on binary classification for simplicity, but the same concept can be easily extended to more than two classes.

ACC is reported as a value between [0,1] or [0, 100], depending on the chosen scale. An accuracy of 0 means the classifier always predicts the wrong label, whereas an accuracy of 1, or 100, means that it always predicts the correct label.

A nice characteristic of this metric is that it has a direct relationship with all values of the confusion matrix. These are the four pillars of supervised machine learning evaluation: true positives, false positives, true negatives, and false negatives.
Accuracy is a proportional measure of the number of correct predictions over all predictions.
Correct predictions are composed of true positives (TP) and true negatives (TN).
All predictions are composed of the entirety of positive (P) and negative (N) examples.
P is composed of TP and false positives (FP), and N is composed of TN and false negatives (

To calculate performance evaluation in the experiment, first, we denote TP, TN, Fp, and Fnias true positive(the number of results correctly predicted as required), true negative (the number of results not required), false positive (the number of results incorrectly predicted as required), false negative(the number of results incorrectly predicted as not required)respectively. We can obtain four measurements: recall, precision, accuracy, and F1 measures as follows

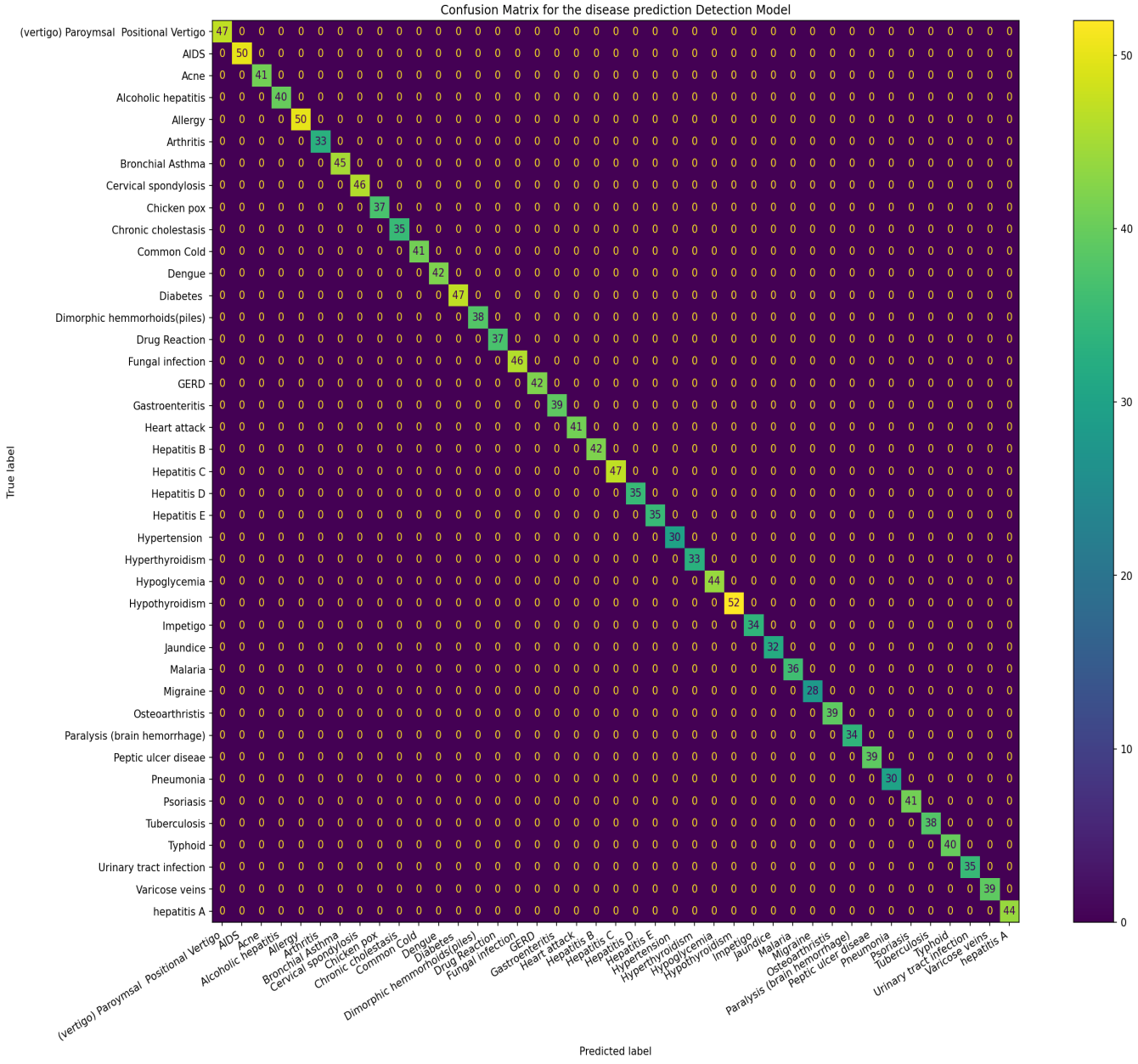
	Actual Positive Class	**Actual Negative Class**
Predicted Positive Class	True positive (*tp*)	False negative (*fn*)
Predicted Negative Class	False positive (*fp*)	True negative (*tn*)

Accuracy of the sample data

Test Data 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	1.00	1.00	1.00	1
Alcoholic hepatitis	1.00	1.00	1.00	1
Allergy	1.00	1.00	1.00	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)	1.00	1.00	1.00	1
Drug Reaction	1.00	1.00	1.00	1
Fungal infection	1.00	1.00	1.00	2
GERD	1.00	1.00	1.00	1
Gastroenteritis	1.00	1.00	1.00	1
Heart attack	1.00	1.00	1.00	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

The confusion matrix evaluated for this model



1.6 Model deployment

```python
import joblib
import numpy as np
import pandas as pd
from flask import Flask, jsonify, request

app = Flask(__name__)

#change port to 8080          $
app.config['SERVER_NAME'] = '10.0.2.2:8080'

model = joblib.load(str("decision_tree.joblib"))


@app.route('/predict', methods=['POST'])
def predict():
    try:
        print(request)
        data = request.json
        print(data)
        payload = request.json
        df_test = pd.DataFrame(columns=list(payload.keys()))
        print(df_test)
        df_test.loc[0] = np.array(list(payload.values()))
        prediction = model.predict(df_test)
        return jsonify({'data': prediction[0]})
    except Exception as e:
        return jsonify({'error': str(e)})
    finally:
        print('executed')

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080)
```

4. Technology used in this model

The increasing adoption of machine learning worldwide is a major factor contributing to its growing popularity. Python has become the favorite choice for data analytics, data science, machine learning, and AI – all thanks to its vast library ecosystem that lets machine learning practitioners access, handle, transform, and process data with ease, platform independence, less complexity, and better readability. Therefore, for this model, we have used Python Programming Language and its extensive collection of Libraries and Packages

Libraries used to build the model

scikit-learn
Numpy
Pandas
Collections
Seaborn
Matplotlib

Model deployment libraries and tools

Joblib
Flask
PostMan (an API platform for building and using APIs)

5. CONCLUSION

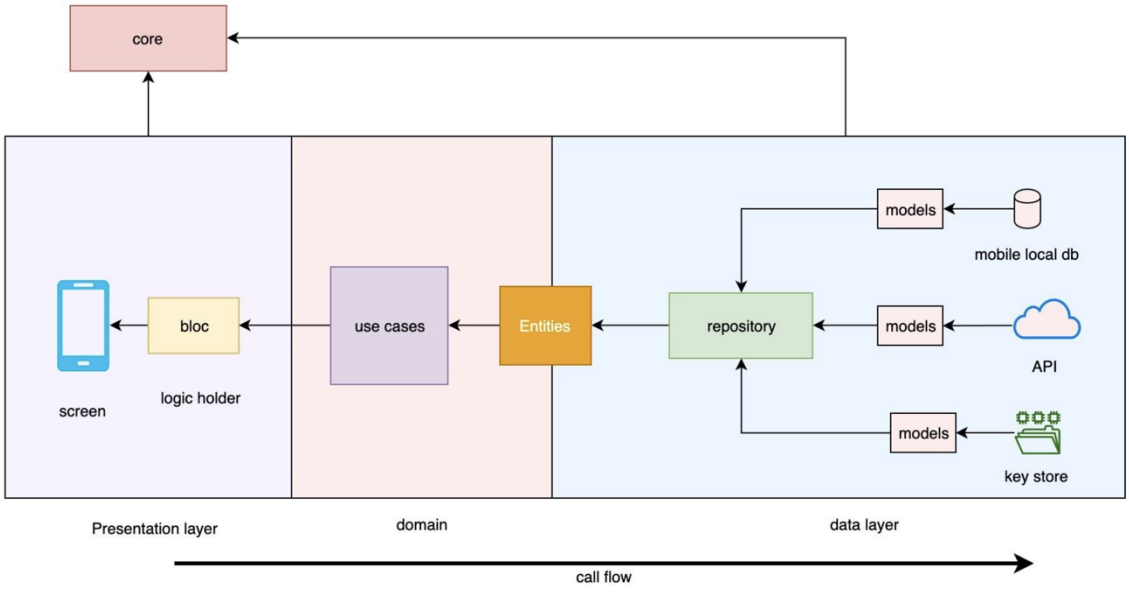
The main aim of this disease prediction system is to predict the disease on the basis of the symptoms. This system takes the symptoms of the user from which he or she suffers as input and generates the final output as a prediction of disease. An average prediction accuracy probability of 100% is obtained. Disease Predictor was successfully implemented using the Flutter framework. This system gives a user-friendly environment and is easy to use. As the system is based on an application, the user can use this system at any time, anywhere.

In conclusion, for disease risk modeling, the accuracy of risk prediction depends on the feature diversity of the hospital data. This review aims to determine the performance, limitations, and future use of software in health care. Findings may help inform future developers of Disease predictability software and promote personalized patient care. The program predicts patient diseases, which is done through user symbols. Predicting the disease at an early stage leads to early diagnosis. This system can assist doctors. System accuracy reaches 97.4% for the model performing on the whole dataset and 100% for the sample obtained from the whole dataset. Machine learning skills are designed to successfully predict outbreaks.

Application and technology used in this system

Clean architecture is used in the system implantation

clean architecture is a design pattern that aims to make applications more maintainable and easier to develop. This pattern is implemented by dividing the application into different layers that work independently of each other and clearly defining the responsibilities and tasks of each.



Our application is used in this context to implement the clean architecture pattern. Our application has a set of modules that divide the application into several features, including Archive, Home, Auth, Diagnosis, Calendar, and Profile. To better explain the importance of clean Architecture, we can look at the example of the profile element. The profile element in the Flutter application is divided into 3 different layers:
the domain layer, the data access layer, and the presentation layer of clean Architecture is used in this case to define the interface for communication between these 3 layers and enable their independent development. As a result, the quality of the application is improved, and its maintenance and development are made easier in the future.
In our application, we divided the feature into 3 layers.

Domain layer

domain layer is the part of the application that contains the business logic and rules that govern the behavior and functionality of the application. It is the layer where the core concepts and abstractions of the application are defined and implemented.

In the context of a clean architecture for working in the Arctic, the domain layer plays a critical role in defining the core concepts and abstractions of the application related to data. This includes:

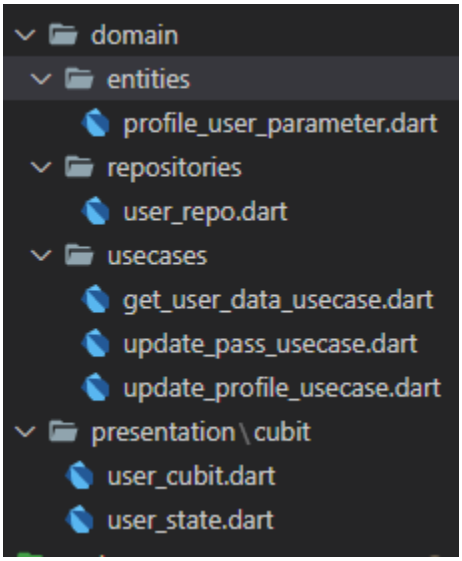
1. Defining the data models: The domain layer defines the data models and entities that represent the data used in the application. This includes defining the attributes and relationships of the data entities.

2. Implementing the business logic: The domain layer implements the business logic and rules that govern the behavior and functionality of the application related to data. This includes defining the validation rules, data processing rules, and data transformation rules.

3. Implementing the use cases: The domain layer implements the use cases and workflows that describe how the application interacts with the data. This includes defining the high-level functionality of the application related to data, such as data retrieval, data processing, and data storage.

By defining the core concepts and abstractions related to data, implementing the business logic and rules, and implementing the use cases and workflows, the domain layer provides a clear separation of concerns between the business logic and the data storage and retrieval mechanisms. This separation of concerns allows for greater flexibility and maintainability of the application, as changes to the data storage and retrieval mechanisms can be made without affecting the business logic and rules of the application.

Domain layer in our features:



domain layer contains
1-use-cases that contain use cases that we need like updateprofileusecase

```dart
import 'package:dartz/dartz.dart';
import 'package:gp/core/errors/failures.dart';
import 'package:gp/core/usecases/usecase.dart';
import 'package:gp/features/user/domain/entities/profile_user_parameter.dart';
import 'package:gp/features/user/domain/repositories/user_repo.dart';

class UpdateProfileUsecase extends UseCase<Unit,ProfileUserParameter>{
  final UserRepositroy userRepositroy;

  UpdateProfileUsecase(this.userRepositroy);
  @override
  Future<Either<Failure, Unit>> call(ProfileUserParameter params) {
    return userRepositroy.updateProfile(params);
  }

}
```

This code defines a class called `UpdateProfileUsecase` that implements a `UseCase` abstract class with two generic types - `Unit` and `ProfileUserParameter`. The `Unit` type represents a return value that does not carry any meaningful information, while the `ProfileUserParameter` type represents the input parameter required for updating a user's profile.

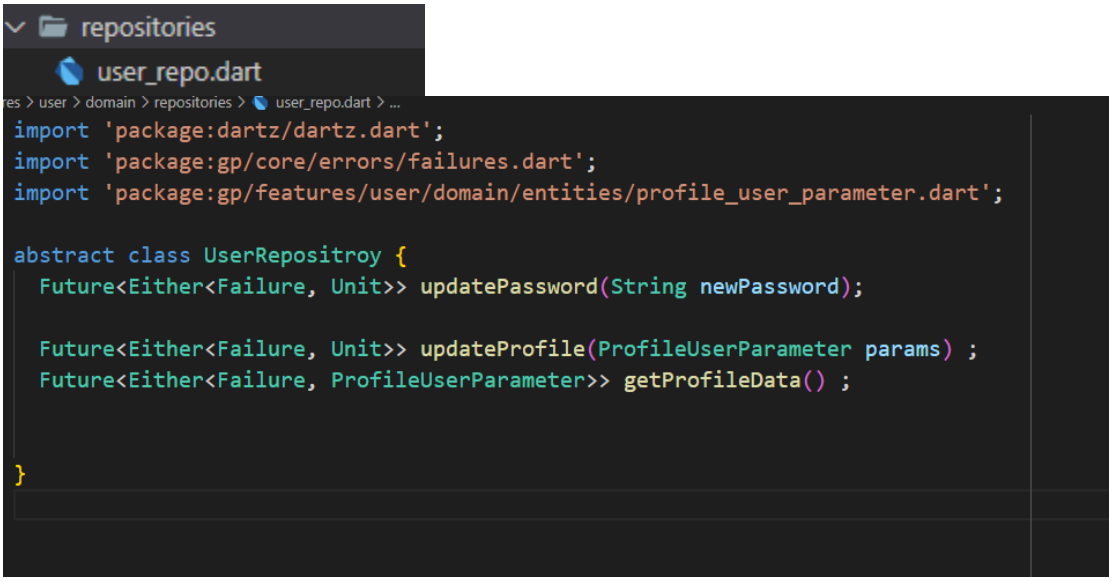
The `UpdateProfileUsecase` class has a constructor that takes a `UserRepository` object as a parameter. The `UserRepository` is an abstract class that defines methods for interacting with the .data layer of the application, such as retrieving and updating user data

The `UpdateProfileUsecase` class overrides the `call` method of the `UseCase` abstract class, which is responsible for executing the business logic of the use case. In this case, the `call` method takes in a `ProfileUserParameter` object and returns a `Future` that either contains a `Failure` object or a `Unit` object. The `Failure` object represents an error that occurred during the execution of the use case, while the `Unit` object represents a successful execution of the use case.

The implementation of the `call` method is very simple. It delegates the responsibility of updating the user's profile to the `UserRepository` object by calling the `updateProfile` method on it with the `ProfileUserParameter` object passed in as a parameter. The `updateProfile` method returns a `Future` that either contains a `Failure` object or a `Unit` object. The `call` method simply returns this `Future` object.

Overall, this code defines a use case for updating a user's profile, which delegates the responsibility of updating the profile to a `UserRepository` object. This use case can be used in a larger application to update user profiles in a way that is consistent with a clean Arctic workflow.

2-Repository that contains the functions that we need in use cases



This code defines an abstract class called `UserRepository`, which is responsible for defining methods for interacting with the data layer of the application related to user profiles .

The `UserRepository` abstract class defines three methods:

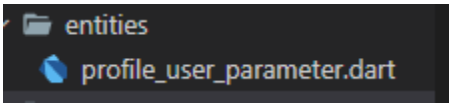
1` .updatePassword`: This method takes a new password as input and returns a `Future` that either contains a `Failure` object or a `Unit` object. The `Failure` object represents an error that occurred during the execution of the method, while the `Unit` object represents a successful execution of the method. This method is responsible for updating the user's password.

2` .updateProfile`: This method takes a `ProfileUserParameter` object as input and returns a `Future` that either contains a `Failure` object or a `Unit` object. The `Failure` object represents an error that occurred during the execution of the method, while the `Unit` object represents a successful execution of the method. This method is responsible for updating the user's profile information.

3` .getProfileData`: This method returns a `Future` that either contains a `Failure` object or a `ProfileUserParameter` object. The `Failure` object represents an error that occurred during the execution of the method, while the `ProfileUserParameter` object represents the user's profile information. This method is responsible for retrieving the user's profile information.

By defining these methods in an abstract class, the application can be designed to use any implementation of the `UserRepository` class, without knowing the specific implementation details. This allows for better separation of concerns in the application and makes it easier to change the implementation of the `UserRepository` class in the future if needed.

Overall, this code defines an interface between the domain layer and the data layer of the application, allowing for clean and consistent interactions between the different layers while ensuring that user profile data is managed in an environmentally responsible and sustainable manner..
3- entities that contain the data entities that we need



Data layer :
In the context of a clean architecture, the data layer is responsible for managing the storage, retrieval, and manipulation of data in an application. It is typically the layer closest to the physical data storage, such as a database or file system, and serves as an abstraction layer between the domain layer and the data storage systems.

In the context of a clean architecture for working in the Arctic, the data layer plays a critical role in ensuring that data is collected, stored, and processed in a way that is environmentally responsible and sustainable. This includes:

.1Using sustainable data storage technologies: The data layer should use data storage technologies that are energy-efficient and environmentally responsible. This might include using cloud-based storage solutions that use renewable energy sources, or using low-power servers and storage devices.

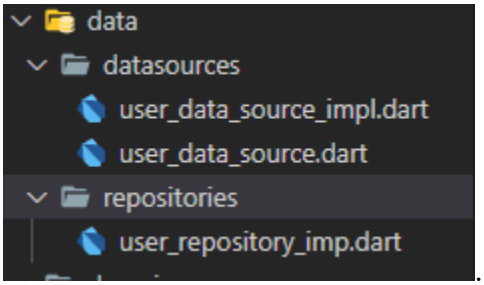
.2Implementing data privacy and security measures: The data layer should implement strong data privacy and security measures to protect sensitive data and ensure compliance with data privacy regulations. This might include using encryption, access controls, and other security measures to protect data at rest and in transit.

Minimizing data storage: The data layer should collect and store only the minimum amount of data necessary to support the application's functionality. This minimizes the amount of data that

needs to be stored and processed, reducing the environmental impact of data storage and .processing

Using sustainable data processing technologies: The data layer should use data processing technologies that are energy-efficient and environmentally responsible. This might include using distributed data processing frameworks that can scale up or down based on demand, or using data processing technologies that can leverage renewable energy sources.

By managing data in an environmentally responsible and sustainable manner, the data layer plays an important role in supporting the larger goals of working in the Arctic, which include minimizing "The clean architecture is a design pattern that aims to make applications more maintainable and easier to develop. This pattern is implemented by dividing the application into different layers that work independently of each other and clearly defining the responsibilities and tasks of each layer.



Data layer divided into 2 components:

1-data sources
the data layer is responsible for managing the storage, retrieval, and manipulation of data in the application. The data layer is typically divided into different data sources that represent the different ways that data can be stored and retrieved .

There are several types of data sources that might be used in a clean architecture for working in the Arctic:

1. Remote data source: This data source represents data that is stored on a remote server or cloud-based storage solution. Examples of remote data sources might include weather data APIs or satellite imagery services.

2. Local data source: This data source represents data that is stored locally on the user's device or on a local server. Examples of local data sources might include data stored in a local database or file system.

3. Cached data source: This data source represents data that has been previously retrieved and stored locally for faster access. Cached data sources can help reduce the amount of time and energy required to retrieve data from remote or local data sources.

4. In-memory data source: This data source represents data that is stored in memory and is used for fast access and processing. In-memory data sources are typically used for frequently accessed data that is not large enough to require storage on disk.

By dividing the data layer into different data sources, the application can more easily manage and optimize data storage and retrieval processes. For example, remote data sources might be optimized for energy efficiency by reducing the frequency of data requests and using data compression techniques to reduce the size of data transfers. Local and cached data sources might be optimized for fast access and reduced energy consumption by using low-power storage devices or implementing data caching strategies to minimize the need for frequent data retrieval. In-memory data sources might be optimized for fast processing and reduced energy consumption by using low-power memory devices or implementing data compression techniques to reduce memory usage

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:dartz/dartz.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:gp/core/errors/failures.dart';
import 'package:gp/features/user/data/datasources/user_data_source.dart';
import 'package:gp/features/user/domain/entities/profile_user_parameter.dart';

class UserDataSourceImpl implements UserDataSource {
  final FirebaseAuth firebaseAuth;
  final FirebaseFirestore firebaseFirestore;
  UserDataSourceImpl(this.firebaseAuth, this.firebaseFirestore);
  @override
  Future<Either<Failure, Unit>> updatePassword(String password) async {
    try {
      await firebaseAuth.currentUser?.updatePassword(password);
      return right(unit);
    } on FirebaseAuthException catch (e) {
      if (e.code == 'weak-password') {
        return left(ServerFailure(message: "password is weak"));
      } else if (e.code == 'requires-recent-login') {
        return left(ServerFailure(message: "Require Recent-login"));
      }
    }
    return left(OfflineFailure('Network not Avaliable', 'try Again'));
  }

  @override
  Future<Either<Failure, Unit>> updateProfile(ProfileUserParameter user) async {
    try {
      var updateParameters = <Object, Object?>{};
      if (user.name != null) {
        updateParameters['name'] = user.name;
```

```
    } if (user.age != null) {
      updateParameters['age'] = user.age;
    }
    if (user.email != null) {
      updateParameters['email'] = user.email;
    }
    if (user.phone != null) {
      updateParameters['phone'] = user.phone;
    }
    if (user.photoUrl != null) {
      updateParameters['photoUrl'] = user.photoUrl;
    }
    if (user.bloodType != null) {
      updateParameters['bloodType'] = user.bloodType;
    }
    if(user.gender!=null){
      updateParameters['gender']=user.gender;
    }
    var doc = firebaseFirestore.collection('users').doc(firebaseAuth.currentUser!.uid);
    //ensure that the user is exist
    var docSnap = await doc.get();
    if (!docSnap.exists) {
      var newMap = <String, dynamic>{};
      newMap = updateParameters.cast<String, dynamic>();
      await doc.set(newMap);
      return right(unit);
    } else {
      await doc.update(updateParameters);
      return right(unit);
    }
  } on Exception catch (e) {
    print(e);
    return left(ServerFailure(message: e.toString()));
  }
}

@override
Future<Either<Failure, ProfileUserParameter>> getProfileData() async{
  var res =await firebaseFirestore.collection('users').doc(firebaseAuth.currentUser!.uid).get();
  if(res.exists){
    var map = res.data() as Map<String,dynamic>;
    var user = ProfileUserParameter.fromMap(map);
    return right(user);
}
else{
  return right(ProfileUserParameter());
```

```
  }
}
}
```
.
. This is a Dart implementation of a `UserDataSource` interface, which provides methods for updating and retrieving user data. The implementation uses Firebase Authentication and Cloud Firestore for authentication and data storage, respectively.

The `UserDataSourceImpl` class implements the `UserDataSource` interface and has three methods:

` .1updatePassword`: This method updates the current user's password. It takes a `String` parameter `password` and returns a `Future` that returns either a `ServerFailure` if an error occurs during the update, or `unit` if the update is successful.

` .2updateProfile`: This method updates the current user's profile data. It takes a `ProfileUserParameter` object `user` that contains the updated user profile data, and returns a `Future` that returns either a `ServerFailure` if an error occurs during the update, or `unit` if the update is successful.

` .3getProfileData`: This method retrieves the current user's profile data. It returns a `Future` that returns either a `ProfileUserParameter` object with the user's profile data, or an empty `ProfileUserParameter` object if the user does not have a profile in the database.

The implementation uses Firebase Authentication to get the current user's ID, and Cloud Firestore to access the `users` collection where the user's profile data is stored. The `updateProfile` method constructs a map of the updated user profile data and uses it to update the user's profile document in the `users` collection. If the user does not have a profile in the database, a new profile document is created with the updated profile data. The `getProfileData` method retrieves the user's profile data from the `users` collection and constructs a `ProfileUserParameter` object from the retrieved data.

Overall, this implementation provides a way to manage user data in a clean and modular way, using the `UserDataSource` interface to abstract away the implementation details and allow for easy testing and swapping of different data sources

```
import 'package:dartz/dartz.dart';
import 'package:gp/core/errors/failures.dart';
import 'package:gp/features/user/domain/entities/profile_user_parameter.dart';

abstract class UserDataSource {
  Future<Either<Failure, Unit>> updatePassword(String password);

  Future<Either<Failure, Unit>> updateProfile(ProfileUserParameter user);

  Future<Either<Failure, ProfileUserParameter>> getProfileData();
}
```

This is a Dart interface called `UserDataSource`, which defines the methods that a data source should implement to manage user data in a clean and modular way .

The `UserDataSource` interface has three methods:

` .1updatePassword`: This method should update the current user's password. It takes a `String` parameter `password` and returns a `Future` that returns either a `ServerFailure` if an error occurs during the update, or `unit` if the update is successful.

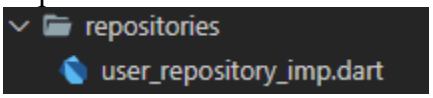
` .2updateProfile`: This method should update the current user's profile data. It takes a `ProfileUserParameter` object `user` that contains the updated user profile data, and returns a `Future` that returns either a `ServerFailure` if an error occurs during the update, or `unit` if the update is successful.

` .3getProfileData`: This method should retrieve the current user's profile data. It returns a `Future` that returns either a `ProfileUserParameter` object with the user's profile data, or a `ServerFailure` if an error occurs during the retrieval.

By defining these methods in an interface, the implementation details of how the user data is managed can be abstracted away and separated from the rest of the application. This allows for easier testing and swapping of different data sources in the future, such as changing from a Firebase data source to a local data source .

The `Either` type is used as the return type for each method to represent the possibility of an error occurring during the data operation. If an error occurs, a `Failure` object is returned to provide more information about the error, such as an error message or error code. If the data operation is successful, a `Unit` object is returned to indicate that the operation was successfully completed, without requiring any additional return data.

Overall, this interface provides a clear separation of concerns and allows for easy management of user data in a clean and modular way, while also providing error handling capabilities through the use of the `Either` type and `Failure` objects
Repositoris:




```
import 'package:gp/core/errors/failures.dart';
import 'package:dartz/dartz.dart';
import 'package:gp/features/user/data/datasources/user_data_source.dart';
import 'package:gp/features/user/domain/entities/profile_user_parameter.dart';
import 'package:gp/features/user/domain/repositories/user_repo.dart';

class UserRepositroyImpl implements UserRepositroy {
  final UserDataSource userDataSource;

  UserRepositroyImpl(this.userDataSource);
  @override
  Future<Either<Failure, Unit>> updatePassword(String newPassword) {
```

```
    return userDataSource.updatePassword(newPassword);
  }
  @override
  Future<Either<Failure, Unit>>  updateProfile(ProfileUserParameter user){
    return userDataSource.updateProfile(user);
  }

  @override
  Future<Either<Failure, ProfileUserParameter>> getProfileData() {
    return userDataSource.getProfileData();
  }
}
```

This is a Dart implementation of a `UserRepository` interface, which provides an abstraction layer between the data layer and the domain layer of the application. The implementation uses a `UserDataSource` object to manage user data operations, such as updating a user's password or retrieving a user's profile data.

The `UserRepositoryImpl` class implements the `UserRepository` interface and has three methods:

` .1updatePassword`: This method updates the current user's password. It takes a `String` parameter `newPassword` and returns a `Future` that returns either a `ServerFailure` if an error occurs during the update, or `unit` if the update is successful.

` .2updateProfile`: This method updates the current user's profile data. It takes a `ProfileUserParameter` object `user` that contains the updated user profile data, and returns a `Future` that returns either a `ServerFailure` if an error occurs during the update, or `unit` if the update is successful.

` .3getProfileData`: This method retrieves the current user's profile data. It returns a `Future` that returns either a `ProfileUserParameter` object with the user's profile data, or a `ServerFailure` if an error occurs during the retrieval.

The implementation uses a `UserDataSource` object to manage the user data operations, allowing for easy switching between different data sources in the future. The implementation simply calls the corresponding method on the `UserDataSource` object and returns the result.

By defining these methods in a repository interface, the implementation details of how the user data is managed can be abstracted away from the rest of the application, allowing the domain layer to only work with the interface methods. This provides a clear separation of concerns and allows for easier testing and swapping of different data sources in the future.

Overall, this implementation provides a clean and modular way to manage user data operations, separating the concerns of the domain layer and the data layer of the application, and allowing for easy testing and swapping of different data sources in the future.

CHAPTER 5: Conclusion and Results

Healthcare is one of the most important industries, and developing it helps cure many. Technology plays a huge role by giving more insights into the human body.

Providing access to a disease prediction system will result in an improvement on both organizational levels and individual levels. Overall, disease prediction by machine learning has the potential to revolutionize healthcare by improving the accuracy and efficiency of disease diagnosis and treatment, leading to better outcomes for patients.

Electronic Health Record (EHR) helps save summaries of a patient's medical records digitally. The digital summary can include lab reports, diagnoses, surgical interventions, and prescriptions. EHRs allow patients to access their medical records from anywhere, at any time. This can be convenient for patients who need to share their medical information with other healthcare providers or keep track of their health data.

References

Patients' reasons for missing scheduled clinic appointments and their solutions at a major urban-based academic medical center

The Dangers of Skipping Your Medications

International Journal of Scientific Research in Engineering and Management (IJSREM)

THE PREDICTION OF DISEASE USING MACHINE LEARNING
Dr C K Gomathy, Mr. A. Rohith Naidu Sri Chandrasekharendra Saraswathi Viswa Mahavidyalaya , Kanchipura
International Journal of Scientific Research in Engineering
And Management THE PREDICTION OF DISEASE USING MACHINE LEARNING Dr C K Gomathy, Mr. A. Rohith Naidu Sri Chandrasekharendra Saraswathi Viswa Mahavidyalaya , Kanchipura

Disease Prediction Using Machine Learning
PRIYANKA J. PANCHAL1 , SHAEEZAH A. MHASKAR2 ,
TEJAL S. ZIMAN3 1, 2, 3 Finolex Academy of Management
and Technology

Disease Prediction by Machine Learning from Healthcare Communities May 2019International Journal of Scientific Research in Science and Technology DOI:10.32628/IJSRST19633