**PREDICT THE DISEASE BASED ON SYMPTOMS USING NLP TECHNIQUES**

**TABLE OF CONTENT**

|  |  |  |
| --- | --- | --- |
| **SL.NO** | **TITLE** | **PAGE.NO** |
| **01.** | **ABSTRACT** |  |
| **02.** | **EXISTING SYSTEM**  2.1 DRAWBACKS |  |
| **03.** | **INTRODUCTION**  3.1 DATA SCIENCE  3.2 ARTIFICIAL INTELLIGENCE |  |
| **04.** | **MACHINE LEARNING** |  |
| **05.** | **DEEP LEARNING** |  |
| **06.** | **PREPARING THE DATASET** |  |
| **07.** | **PROPOSED SYSTEM**  7.1 ADVANTAGES |  |
| **08.** | **LITERATURE SURVEY** |  |
| **09.** | **SYSTEM STUDY**  9.1 PROJECT GOAL  9.2 OBJECTIVES  9.3 SCOPE OF THE PROJECT |  |
| **10.** | **OUTLINE OF THE PROJECT**  10.1 OVERVIEW OF THE SYSTEM |  |
| **11.** | **PROJECT REQUIREMENT**  11.1 FUNCTIONAL REQUIREMENT  11.2 NON-FUNCTIONAL REQUIREMENT |  |
| **12.** | **FEASIBILITY STUDY**  12.1 DATAFLOW DIAGRAM |  |
| **13.** | **SYSTEM ARCHITECTURE** |  |
| **14.** | **WORKFLOW DIAGRAM** |  |
| **15.** | **USECASE DIAGRAM** |  |
| **16.** | **CLASS DIAGRAM** |  |
| **17.** | **ACTIVITY DIAGRAM** |  |
| **18.** | **SEQUENCE DIAGRAM** |  |
| **19.** | **E.R – DIAGRAM** |  |
| **20.** | **COLLABORATION DIAGRAM** |  |
| **21.** | **ENVIRONMENT REQUIREMENT** |  |
| **22.** | **SOFTWARE DESCRIPTION**  22.1 ANACONDA NAVIGATOR  22.2 JUPYTER NOTEBOOK  22.3 PYCHARM |  |
| **23.** | **PYTHON** |  |
| **24.** | **METHODOLOGY** |  |
| **25.** | **ARTIFICIAL NEURAL NETWORK (ANN)** |  |
| **26.** | **ARCHITECTURE OF ANN** |  |
| **27.** | **RNN**  27.LSTM ARCHITECTURE  27.2 GRATED RECURREND UNIT ARCHIETECTURE |  |
| **28.** | **LIST OF MODULES** |  |
| **29.** | **MODULE DESCRIPTION**  29.1 IMPORT THE GIVEN IMAGE FROM DATASET  29.2 TO TRAIN THE MODULE BY GIVEN IMAGE DATASET  29.3 WORKING PROCESS OF LAYERS IN ANN MODEL  29.4 CLASSIFICATION IDENTIFICATION |  |
| **30.** | **DEPLOY**  30.1 DJANGO  30.2 HTML  30.3 CSS |  |
| **31.** | **CODING** |  |
| **32.** | **CONCLUSION** |  |
| **33.** | **FUTURE WORK** |  |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **SL.NO** | **TITLE** | **PAGE.NO** |
| **1.** | **DATAFLOW DIAGRAM** |  |
| **2.** | **SYSTEM ARCHITECTURE** |  |
| **3.** | **WORKFLOW DIAGRAM** |  |
| **4.** | **USECASE DIAGRAM** |  |
| **5.** | **CLASS DIAGRAM** |  |
| **6.** | **ACTIVITY DIAGRAM** |  |
| **7.** | **SEQUENCE DIAGRAM** |  |
| **8.** | **E.R – DIAGRAM** |  |
| **9.** | **COLLABORATION DIAGRAM** |  |

**LIST OF SYSMBOLS**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **NOTATION**  **NAME** | **NOTATION** | **DESCRIPTION** |
| 1. | Class | *+ public*  *-private*  *# protected*  *Class Name*  *-attribute*  *-attribute*  *+operation*  *+operation*  *+operation* | Represents a collection of similar entities grouped together. |
| 2. | Association | nAME  Class B  Class A  Class A  Class B | Associations represents static relationships between classes. Roles representsthe way the two classes see each other. |
| 3. | Actor | Class A  Class A  Class B  Class B | It aggregates several classes into a single classes. |
| 4. | Aggregation | Interaction between the system and external environment |

|  |  |  |  |
| --- | --- | --- | --- |
| 5. | Relation  (uses) | uses | Used for additional process communication. |
| 6. | Relation  (extends) | EXTENDS | Extends relationship is used when one use case is similar to another use case but does a bit more. |
| 7. | Communication |  | Communication between various use cases. |
| 8. | State | State | State of the processs. |
| 9. | Initial State |  | Initial state of the object |
| 10. | Final state |  | Final state of the object |
| 11. | Control flow |  | Represents various control flow between the states. |
| 12. | Decision box |  | Represents decision making process from a constraint |
| 13. | Usecase |  | Interact ion between the system and external environment. |

|  |  |  |  |
| --- | --- | --- | --- |
| 14. | Component |  | Represents physical modules which is a collection of components. |
| 15. | Node |  | Represents physical modules which are a collection of components. |
| 16. | Data Process/State |  | A circle in DFD represents a state or process which has been triggered due to some event or acion. |
| 17. | External entity |  | Represents external entities such as keyboard,sensors,etc. |
| 18. | Transition |  | Represents communication that occurs between processes. |
| 19. | Object Lifeline |  | Represents the vertical dimensions that the object communications. |
| 20. | Message | Message | Represents the message exchanged. |

**PREDICT THE DISEASE BASED ON SYMPTOMS USING NLP TECHNIQUES**

**ABSTRACT:**

Predicting diseases based on symptoms using Natural Language Processing (NLP) techniques is a burgeoning field with significant potential for improving healthcare diagnostics. NLP enables the analysis and interpretation of unstructured text data, such as patient symptoms recorded in electronic health records or online health forums. This abstract explores the application of NLP techniques in disease prediction based on symptoms, highlighting the benefits, challenges, and future directions of this approach. The abstract begins by introducing the importance of accurate disease prediction and the role of NLP in processing textual symptom data. NLP techniques enable the extraction of relevant information from symptom descriptions, such as the presence, severity, and temporal aspects of symptoms. These techniques also facilitate the identification of symptom co-occurrences and relationships, which can aid in disease prediction.

Keywords: Disease prediction, Symptoms, Natural Language Processing (NLP), Text analysis, Named Entity Recognition (NER), Semantic Role Labeling (SRL), Sentiment analysis, Medical ontologies, Deep learning, Transfer learning, Multimodal data.

**EXISTING SYSTEM:**

This systematic review surveyed automatic emotion recognition systems applied in real clinical contexts, focusing on populations with pathologies. The review included 52 scientific papers meeting the inclusion criteria. Clinical applications primarily targeted neuro-developmental, neurological, and psychiatric disorders, aiming to diagnose, monitor, or treat emotional symptoms. Observational study designs were common for monitoring and diagnosis, while interventional approaches were used for treatment. Video and audio signals were the most adopted, and supervised shallow learning was the prevalent approach for emotion recognition algorithms. Clinical limitations included small sample sizes, absence of control groups, and lack of real-life conditions testing. Technically, heterogeneity in performance metrics, datasets, and algorithms challenged result comparability, robustness, reliability, and reproducibility. Suggested guidelines were provided to address these challenges and guide future research.

**DEMERITS:**

* Focused on only emotion Recognition medical diagnosis.
* They did not compared more than an algorithms to getting better accuracy level.
* They did not build the deploy model.
* Limited scope.
* The accuracy level and performance level are low.

**INTRODUCTION:**

The advent of Natural Language Processing (NLP) techniques has ushered in a trans-formative era in healthcare, offering innovative approaches to disease prediction and diagnosis. In recent years, there has been a growing interest in harnessing the power of NLP to predict diseases based on symptomatic information. This paradigm shift represents a convergence of computational linguistics, medical expertise, and machine learning, aiming to enhance the accuracy and efficiency of disease identification.

Traditionally, disease prediction has relied on structured data, such as lab results and medical histories. However, the incorporation of NLP allows us to extract valuable insights from unstructured textual data, such as electronic health records, clinical notes, and patient-reported symptoms. This unstructured data often contains nuanced information, reflecting the subtleties of patient experiences that might elude structured approaches.

The complexity of human language, with its variability, context-dependency, and ambiguity, presents both a challenge and an opportunity. NLP-techniques enable the development of models capable of understanding and interpreting the intricate relationships between symptoms and underlying medical conditions. By leveraging advanced machine learning algorithms, these models can discern patterns, associations, and dependencies within textual data, empowering healthcare professionals with timely and accurate disease predictions.

This paper delves into the current landscape of disease prediction using NLP-techniques, exploring the methodologies, challenges, and potential applications in healthcare. Through the integration of linguistics and machine learning, the goal is to contribute to a more comprehensive and personalized approach to disease prediction, ultimately improving patient outcomes and optimizing healthcare delivery.

**Domain overview:**

**Data Science:**

Data science is an interdisciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from structured and unstructured data, and apply knowledge and actionable insights from data across a broad range of application domains.

The term "data science" has been traced back to 1974, when Peter Naur proposed it as an alternative name for computer science. In 1996, the International Federation of Classification Societies became the first conference to specifically feature data science as a topic. However, the definition was still in flux.

The term “data science” was first coined in 2008 by D.J. Patil, and Jeff Hammerbacher, the pioneer leads of data and analytics efforts at LinkedIn and Facebook. In less than a decade, it has become one of the hottest and most trending professions in the market.

Data science is the field of study that combines domain expertise, programming skills, and knowledge of mathematics and statistics to extract meaningful insights from data.

Data science can be defined as a blend of mathematics, business acumen, tools, algorithms and machine learning techniques, all of which help us in finding out the hidden insights or patterns from raw data which can be of major use in the formation of big business decisions.

**Data Scientist:**

Data scientists examine which questions need answering and where to find the related data. They have business acumen and analytical skills as well as the ability to mine, clean, and present data. Businesses use data scientists to source, manage, and analyze large amounts of unstructured data.

**3.2 Artificial Intelligence**:

Artificial intelligence (AI) refers to the simulation of human intelligence in machines that are programmed to think like humans and mimic their actions. The term may also be applied to any machine that exhibits traits associated with a human mind such as learning and problem-solving.

Artificial intelligence (AI) is [intelligence](https://en.wikipedia.org/wiki/Intelligence) demonstrated by [machines](https://en.wikipedia.org/wiki/Machine), as opposed to the natural intelligence [displayed by humans](https://en.wikipedia.org/wiki/Human_intelligence) or [animals](https://en.wikipedia.org/wiki/Animal_cognition). Leading AI textbooks define the field as the study of "[intelligent agents](https://en.wikipedia.org/wiki/Intelligent_agent)" any system that perceives its environment and takes actions that maximize its chance of achieving its goals. Some popular accounts use the term "artificial intelligence" to describe machines that mimic "cognitive" functions that humans associate with the [human mind](https://en.wikipedia.org/wiki/Human_mind), such as "learning" and "problem solving", however this definition is rejected by major AI researchers.

Artificial intelligence is the simulation of human intelligence processes by machines, especially computer systems. Specific applications of AI include expert systems, natural language processing, speech recognition and machine vision.

AI applications include advanced web search engines, recommendation systems (used by Youtube, Amazon and Netflix), Understanding human speech (such as Siri or Alexa), self-driving cars (e.g. Tesla), and competing at the highest level in strategic game systems (such as chess and Go), As machines become increasingly capable, tasks considered to require "intelligence" are often removed from the definition of AI, a phenomenon known as the AI effect. For instance, optical character recognition is frequently excluded from things considered to be AI, having become a routine technology.

Artificial intelligence was founded as an academic discipline in 1956, and in the years since has experienced several waves of optimism, followed by disappointment and the loss of funding (known as an "AI winter"), followed by new approaches, success and renewed funding. AI research has tried and discarded many different approaches during its lifetime, including simulating the brain, modeling human problem solving, formal logic, large databases of knowledge and imitating animal behavior. In the first decades of the 21st century, highly mathematical statistical machine learning has dominated the field, and this technique has proved highly successful, helping to solve many challenging problems throughout industry and academia.

The various sub-fields of AI research are centered around particular goals and the use of particular tools. The traditional goals of AI research include reasoning, knowledge representation, planning, learning, natural language processing, perception and the ability to move and manipulate objects. General intelligence (the ability to solve an arbitrary problem) is among the field's long-term goals. To solve these problems, AI researchers use versions of search and mathematical optimization, formal logic, artificial neural networks, and methods based on statistics, probability and economics. AI also draws upon computer science, psychology, linguistics, philosophy, and many other fields.

The field was founded on the assumption that human intelligence "can be so precisely described that a machine can be made to simulate it". This raises philosophical arguments about the mind and the ethics of creating artificial beings endowed with human-like intelligence. These issues have been explored by myth, fiction and philosophy since antiquity. Science fiction and futurology have also suggested that, with its enormous potential and power, AI may become an existential risk to humanity.

As the hype around AI has accelerated, vendors have been scrambling to promote how their products and services use AI. Often what they refer to as AI is simply one component of AI, such as machine learning. AI requires a foundation of specialized hardware and software for writing and training machine learning algorithms. No one programming language is synonymous with AI, but a few, including Python, R and Java, are popular.

In general, AI systems work by ingesting large amounts of labeled training data, analyzing the data for correlations and patterns, and using these patterns to make predictions about future states. In this way, a chatbot that is fed examples of text chats can learn to produce life like exchanges with people, or an image recognition tool can learn to identify and describe objects in images by reviewing millions of examples.

AI programming focuses on three cognitive skills: learning, reasoning and self-correction.

**Learning processes.** This aspect of AI programming focuses on acquiring data and creating rules for how to turn the data into actionable information. The rules, which are called algorithms, provide computing devices with step-by-step instructions for how to complete a specific task.

**Reasoning processes.** This aspect of AI programming focuses on choosing the right algorithm to reach a desired outcome.

**Self-correction processes.** This aspect of AI programming is designed to continually fine-tune algorithms and ensure they provide the most accurate results possible.

AI is important because it can give enterprises insights into their operations that they may not have been aware of previously and because, in some cases, AI can perform tasks better than humans. Particularly when it comes to repetitive, detail-oriented tasks like analyzing large numbers of legal documents to ensure relevant fields are filled in properly, AI tools often complete jobs quickly and with relatively few errors.

Artificial neural networks and deep learning artificial intelligence technologies are quickly evolving, primarily because AI processes large amounts of data much faster and makes predictions more accurately than humanly possible.

**MACHINE LEARNING**

Machine learning is to predict the future from past data. Machine learning (ML) is a type of artificial intelligence (AI) that provides computers with the ability to learn without being explicitly programmed. Machine learning focuses on the development of Computer Programs that can change when exposed to new data and the basics of Machine Learning, implementation of a simple machine learning algorithm using python. Process of training and prediction involves use of specialized algorithms. It feed the training data to an algorithm, and the algorithm uses this training data to give predictions on a new test data. Machine learning can be roughly separated in to three categories. There are supervised learning, unsupervised learning and reinforcement learning. Supervised learning program is both given the input data and the corresponding labeling to learn data has to be labeled by a human being beforehand. Unsupervised learning is no labels. It provided to the learning algorithm. This algorithm has to figure out the clustering of the input data. Finally, Reinforcement learning dynamically interacts with its environment and it receives positive or negative feedback to improve its performance.

Data scientists use many different kinds of machine learning algorithms to discover patterns in python that lead to actionable insights. At a high level, these different algorithms can be classified into two groups based on the way they “learn” about data to make predictions: supervised and unsupervised learning. Classification is the process of predicting the class of given data points. Classes are sometimes called as targets/ labels or categories. Classification predictive modeling is the task of approximating a mapping function from input variables(X) to discrete output variables(y). In machine learning and statistics, classification is a supervised learning approach in which the computer program learns from the data input given to it and then uses this learning to classify new observation. This data set may simply be bi-class (like identifying whether the person is male or female or that the mail is spam or non-spam) or it may be multi-class too. Some examples of classification problems are: speech recognition, handwriting recognition, bio metric identification, document classification etc.

Analyses Predicts

Machine Learning

Past Dataset

Trains

Process of Machine learning

[Supervised Machine Learning](https://www.geeksforgeeks.org/supervised-unsupervised-learning/) **is the** majority of practical machine learning uses supervised learning. Supervised learning is where have input variables (X) and an output variable (y) and use an algorithm to learn the mapping function from the input to the output**is y = f(X).** The goal is to approximate the mapping function so well that when you have new input data (X) that you can predict the output variables (y) for that data. Techniques of Supervised Machine Learning algorithms include **logistic regression, multi-class classification, Decision Trees**and**support vector machines etc.** Supervised learning requires that the data used to train the algorithm is already labeled with correct answers. Supervised learning problems can be further grouped into **Classification** problems. This problem has as goal the construction of a succinct model that can predict the value of the dependent attribute from the attribute variables. The difference between the two tasks is the fact that the dependent attribute is numerical for categorical for classification. A classification model attempts to draw some conclusion from observed values. Given one or more inputs a classification model will try to predict the value of one or more outcomes. A classification problem is when the output variable is a category, such as “red” or “blue”.

**Natural Language Processing (NLP):**

**NLP Fundamentals:**

Natural Language Processing (NLP) is a multidisciplinary field that combines computer science, artificial intelligence, and linguistics to enable machines to interact with and understand human language. Its primary goal is to bridge the communication gap between humans and computers, allowing for more intuitive and context-aware interactions.

**Key Components of NLP:**

1. Tokenization:

Tokenization involves breaking down a text into smaller units, such as words or phrases, to facilitate analysis. It forms the foundational step for many NLP tasks.

2. Part-of-Speech Tagging:

Part-of-speech tagging assigns grammatical categories (like noun, verb, adjective) to each word in a sentence. This information is crucial for understanding the syntactic structure of a sentence.

3. Named Entity Recognition (NER):

NER identifies and classifies entities such as names of people, organizations, locations, and more within a text. It's pivotal for extracting structured information.

4. Syntax and Parsing:

Syntax analysis involves understanding the grammatical structure of sentences, and parsing helps create a hierarchical representation of this structure.

5. Semantic Analysis:

Semantic analysis goes beyond syntax to understand the meaning of words and sentences. It explores the relationships between words and their contextual interpretations.

**Advanced NLP Tasks:**

1. Sentiment Analysis:

Sentiment analysis gauges the sentiment expressed in a piece of text—whether it's positive, negative, or neutral. It finds applications in customer reviews, social media monitoring, and market research.

2. Language Translation:

NLP powers machine translation systems that can automatically translate text from one language to another. This has been a transformative application in breaking down language barriers.

3. Text Summarization:

Text summarization involves condensing lengthy documents or articles into concise summaries while retaining the essential information.

4. Question Answering Systems:

NLP contributes to systems capable of understanding and responding to user queries, extracting relevant information from diverse sources.

5. Chatbots and Virtual Assistants:

Conversational agents, such as chatbots and virtual assistants, utilize NLP to engage in natural language conversations, providing assistance and information.

**Challenges and Future Directions:**

1. Ambiguity and Context:

NLP systems often struggle with understanding ambiguous language and context-dependent meanings. Resolving these challenges is essential for more accurate language interpretation.

2. Multilingual Understanding:

Achieving robust NLP across multiple languages remains a significant research area. Adapting models to diverse linguistic nuances is an ongoing challenge.

3. Ethical Considerations:

As NLP applications become more prevalent, ethical considerations, including bias mitigation and privacy concerns, are gaining prominence in the development and deployment of NLP systems.

**DEEP LEARNING**

Deep learning is a branch of machine learning which is completely based on artificial neural networks, as neural network is going to mimic the human brain so deep learning is also a kind of mimic of human brain. It’s on hype nowadays because earlier we did not have that much processing power and a lot of data. A formal definition of deep learning is- neurons Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones. In brain approximately 100 billion neurons all together this is a picture of an individual neuron and each neuron is connected through thousands of their neighbors. The question here is how it recreates these neurons in a computer. So, it creates an artificial structure called an artificial neural net where we have nodes or neurons. It has some neurons for input value and some for output value and in between, there may be lots of neurons interconnected in the hidden layer.

# It need to identify the actual problem in order to get the right solution and it should be understood, the feasibility of the Deep Learning should also be checked (whether it should fit Deep Learning or not). It needs to identify the relevant data which should correspond to the actual problem and should be prepared accordingly. Choose the Deep Learning Algorithm appropriately. Algorithm should be used while training the dataset. Final testing should be done on the dataset

Deep learning (also known as deep structured learning) is part of a broader family of machine learning methods based on artificial neural networks with representation learning. Learning can be supervised, semi-supervised or unsupervised.

Deep-learning architectures such as deep neural networks, deep belief networks, deep reinforcement learning, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, speech recognition, natural language processing, machine translation, bioinformatics, drug design, medical image analysis, material inspection and board game programs, where they have produced results comparable to and in some cases surpassing human expert performance.

Artificial neural networks (ANNs) were inspired by information processing and distributed communication nodes in biological systems. ANNs have various differences from biological brains. Specifically, neural networks tend to be static and symbolic, while the biological brain of most living organisms is dynamic (plastic) and analogue.

The adjective "deep" in deep learning refers to the use of multiple layers in the network. Early work showed that a linear perceptron cannot be a universal classifier, but that a network with a non-polynomial activation function with one hidden layer of unbounded width can. Deep learning is a modern variation which is concerned with an unbounded number of layers of bounded size, which permits practical application and optimized implementation, while retaining theoretical universality under mild conditions. In deep learning the layers are also permitted to be heterogeneous and to deviate widely from biologically informed connectionist models, for the sake of efficiency, trainability and understandability, whence the "structured" part.

Deep learning is a class of [machine learning](https://en.wikipedia.org/wiki/Machine_learning) [algorithms](https://en.wikipedia.org/wiki/Algorithm) that uses multiple layers to progressively extract higher-level features from the raw input. For example, in [image processing](https://en.wikipedia.org/wiki/Image_processing), lower layers may identify edges, while higher layers may identify the concepts relevant to a human such as digits or letters or faces.

**Interpretations:**

Deep neural networks are generally interpreted in terms of the universal approximation theorem or probabilistic inference.

The classic universal approximation theorem concerns the capacity of feed-forward neural networks with a single hidden layer of finite size to approximate continuous functions. In 1989, the first proof was published by George Cybenko for sigmoid activation functions and was generalised to feed-forward multi-layer architectures in 1991 by Kurt Hornik. Recent work also showed that universal approximation also holds for non-bounded activation functions such as the rectified linear unit.

The universal approximation theorem for deep neural networks concerns the capacity of networks with bounded width but the depth is allowed to grow proved that if the width of a deep neural network with ReLU activation is strictly larger than the input dimension, then the network can approximate any Lebesgue integrable function; If the width is smaller or equal to the input dimension, then deep neural network is not a universal approximator.

The probabilistic interpretation derives from the field of machine learning. It features inference, as well as the optimization concepts of training and testing, related to fitting and generalization, respectively. More specifically, the probabilistic interpretation considers the activation nonlinearity as a cumulative distribution function. The probabilistic interpretation led to the introduction of dropout as regularizer in neural networks. The probabilistic interpretation was introduced by researchers including Hopfield, Widrow and Narendra and popularized in surveys such as the one by Bishop.

**Deep learning revolution:**

In 2012, a team led by George E. Dahl won the "Merck Molecular Activity Challenge" using multi-task deep neural networks to predict the biomolecular target of one drug. In 2014, Hochreiter's group used deep learning to detect off-target and toxic effects of environmental chemicals in nutrients, household products and drugs and won the "Tox21 Data Challenge" of NIH, FDA and NCATS.

Significant additional impacts in image or object recognition were felt from 2011 to 2012. Although ANNs trained by back-propagation had been around for decades, and GPU implementations of NNs for years, including ANNs, fast implementations of ANNs on GPUs were needed to progress on computer vision. In 2011, this approach achieved for the first time superhuman performance in a visual pattern recognition contest. Also in 2011, it won the ICDAR Chinese handwriting contest, and in May 2012, it won the ISBI image segmentation contest. Until 2011, ANNs did not play a major role at computer vision conferences, but in June 2012, a paper by Ciresan et al. at the leading conference CVPR showed how max-pooling ANNs on GPU can dramatically improve many vision benchmark records. In October 2012, a similar system by Krizhevsky et al. won the large-scale ImageNet competition by a significant margin over shallow machine learning methods. In November 2012, Ciresan et al.'s system also won the ICPR contest on analysis of large medical images for cancer detection, and in the following year also the MICCAI Grand Challenge on the same topic. In 2013 and 2014, the error rate on the ImageNet task using deep learning was further reduced, following a similar trend in large-scale speech recognition.

Image classification was then extended to the more challenging task of generating descriptions (captions) for images, often as a combination of ANNs and LSTMs.

Some researchers state that the October 2012 ImageNet victory anchored the start of a "deep learning revolution" that has transformed the AI industry.

In March 2019, Yoshua Bengio, Geoffrey Hinton and Yann LeCun were awarded the Turing Award for conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing.

**PREPARING THE DATASET:**

Utilize medical literature and ontologies to create a structured dataset mapping symptoms to diseases, ensuring a comprehensive and accurate representation for training NLP models in disease prediction.

**PROPOSED SYSTEM:**

The proposed system aims to leverage NLP techniques for disease prediction based on symptoms, incorporating advancements in the field to enhance accuracy and usability. The system consists of several components and methodologies to achieve its objective. The key components of the proposed system are as follows. Data Collection: The system will gather a comprehensive dataset comprising symptom descriptions along with their corresponding disease labels. This dataset can be obtained from electronic health records, patient forums, medical literature, or crowd sourced platforms. The dataset will serve as the foundation for training and evaluating the disease prediction model. Pre-processing and Text Analysis: The system will employ NLP techniques for pre-processing and text analysis. This includes tasks such as tokenization, part-of-speech tagging, and syntactic parsing to extract relevant information from symptom descriptions. Named Entity Recognition (NER) will be applied to identify medical entities (symptoms, body parts, medical conditions) mentioned in the text. Semantic Role Labeling (SRL) will help understand the relationships between symptoms and modifiers, aiding in accurate symptom interpretation.

**MERITS:**

* We compared more than a two algorithms to getting better accuracy level.
* We predict symtoms to disease using nlp techniques.
* We improve the accuracy.
* We develop a full stack framework based application for deployment purpose.
* We improved the accuracy level and performance level.

**LITERATURE REVIEW:**

**Title**: NLP based Segmentation Protocol for Predicting Diseases and Finding Doctors

**Author**:Aswathy K P1, Rathi R2, Shyam Shankar E P3

**Year**: 2019

-In these decades the Web-browsing have become a habitual or imperative thing for the mankind. The people have been using many browsers, online/offline Apps and Software for their daily needs including their healthcare queries. The proper clarification of such healthcare queries and accurate disease-predictions will be very serviceable for the society. In light of this we developed a medical query system which has the ability not only to predict the disease by analyzing the user’s symptoms but also to suggest doctors in user’s locality if entered the locale while registering. In this paper we proposed a NLP based Unstructured Data Processing Method (NUDPM) which enables the facile segmentation and extraction of medical terms/symptoms from the user queries. The analysis of this NUDPM computed data with a predefined ‘Disease data base’ will predict the disease accurately. As well as the execution of Doctor-Type selection methods will choose appropriate doctors near to user’s locality. Finally the system exhibits the details of the disease and nearby doctors to user.

**Title**: Multiple Disease Prognostication Based On Symptoms Using Machine Learning Techniques

**Author**: Kajal Patil , Sakshee Pawar , Pramita Sandhyan and Jyoti Kundale

**Year**: 2022

. Disease Prediction system that uses Machine Learning forecasts the ailments on the basis of the data pertaining to the symptoms entered by the user and provides trustworthy findings based on that data. If the patient isn’t in any danger and the user merely wants to know what kind of ailment he or she has had. It is a system that gives the user suggestions and methods on how to keep their health system in good shape, as well as a way to find out if they have a sickness utilizing this forecast. Due to a diversity of diseases and a lower doctor-patient ratio, the use of particular disease prediction technologies as well as concerns about health has risen. We are focusing on offering customers with an instant and accurate disease prognosis based on the symptoms they enter, as well as the severity of the condition projected. It will provide the best algorithm and doctor consultation. Different machine learning algorithms are employed to forecast illnesses, ensuring speedy and reliable predictions

**Title**: Chatbot For Disease Prediction Using Machine Learning

**Author**: V. Sai Susmita, N.N.G. Tarun, S.D. Adhitya Kalyan, D. Dharani, V.S.V.S. Murthy

**Year**: 2023

: During pandemic situations like Covid, it is not advisable for patients to visit the doctors in a physical environment. In this paper, we present a chatbot application that can advise the patients about their physical health based upon symptom inputs. Users can ask about anything regarding the health problems they are facing and find a solution to the problem. The system enables users to describe symptoms and quickly recall a course of treatment by relying on natural language based interaction. To predict the disease and generate the response, Stochastic Gradient Descent algorithm has been used

**Title** : Multi-Modal Point-of-Care Diagnostics for COVID-19 Based on Acoustics and Symptoms

**Author**:SRIKANTH RAJ CHETUPALLI ,PRASHANT KRISHNAN

**Year**: 2022

Background: The COVID-19 pandemic has highlighted the need to invent alternative respiratory health diagnosis methodologies which provide improvement with respect to time, cost, physical distancing and detection performance. In this context, identifying acoustic bio-markers of respiratory diseases has received renewed interest. Objective: In this paper, we aim to design COVID-19 diagnostics based on analyzing the acoustics and symptoms data. Towards this, the data is composed of cough, breathing, and speech signals, and health symptoms record, collected using a web-application over a period of twenty months. Methods: We investigate the use of time-frequency features for acoustic signals and binary features for encoding different health symptoms. We experiment with use of classifiers like logistic regression, support vector machines and long-short term memory (LSTM) network models on the acoustic data, while decision tree models are proposed for the symptoms data. Results: We show that a multi-modal integration of inference from different acoustic signal categories and symptoms achieves an area-under-curve (AUC) of 96.3%, a statistically significant improvement when compared against any individual modality (p < 0.05). Experimentation with different feature representations suggests that the mel-spectrogram acoustic features performs relatively better across the three kinds of acoustic signals. Further, a score analysis with data recorded from newer SARS-CoV-2 variants highlights the generalization ability of the proposed diagnostic approach for COVID-19 detection. Conclusion: The proposed method shows a promising direction for COVID-19 detection using a multi-modal dataset, while generalizing to new COVID variants.

**Title**: Mining Symptom and Disease Web Data with NLP and Open Linked Data

**Author**:Hong Qing Yu

**Year**: 2019

Machine Learning (ML) technologies in recent years are widely applied in various areas to assist knowledge gaining and decision-making on healthcare. However, there is no reliable dataset that contains semantic structured knowledge on symptom and disease enable to apply advanced machine learning algorithms such clustering or prediction. In this paper, we propose a framework that can extract data from web with apply Natural Language Processing (NLP) process and semantic annotation to create Open Linked Data (OLD) based knowledge graph. At the end, the knowledge graph can be used for ML algorithms and graph oriented Deep Learning techniques

**SYSTEM STUDY:**

**Overview of the system**

The system utilizes Natural Language Processing (NLP) techniques to analyze and predict diseases based on symptoms, extracting meaningful patterns from textual symptom descriptions to enable accurate and timely disease identification.

**AIM:**

Develop a disease prediction system leveraging NLP techniques to analyse and interpret textual symptom descriptions, facilitating accurate and timely diagnosis for improved healthcare outcomes.

**OBJECTIVE:**

**Develop a predictive model using Natural Language Processing (NLP) techniques to analyse and correlate textual symptom descriptions with medical knowledge, enabling accurate disease prediction and diagnosis.**

**SCOPE:**

Developing and refining Natural Language Processing (NLP) models to accurately predict diseases based on textual symptoms, enabling early diagnosis and personalised healthcare interventions.

**Construction of a Predictive Model**

## Machine learning needs data gathering have lot of past data’s. Data gathering have sufficient historical data and raw data. Before data pre-processing, raw data can’t be used directly. It’s used to pre-process then, what kind of algorithm with model. Training and testing this model working and predicting correctly with minimum errors. Tuned model involved by tuned time to time with improving the accuracy.

Data Gathering

Data Pre-Processing

Choose model

Train model

Test model

Tune model

Prediction

Process of dataflow diagram

**PROJECT REQUIREMENTS:**

**General:**

Requirements are the basic constrains that are required to develop a system. Requirements are collected while designing the system. The following are the requirements that are to be discussed.

1. Functional requirements

2. Non-Functional requirements

3. Environment requirements

A. Hardware requirements

B. software requirements

**Functional requirements:**

The software requirements specification is a technical specification of requirements for the software product. It is the first step in the requirements analysis process. It lists requirements of a particular software system. The following details to follow the special libraries like sk-learn, pandas, numpy, matplotlib and seaborn.

**Non-Functional Requirements:**

Process of functional steps,

1. Problem define
2. Preparing data
3. Evaluating algorithms
4. Improving results
5. Prediction the result

**ENVIRONMENTAL REQUIREMENTS:**

1. Software Requirements:

Operating System: Windows

Tool : Anaconda with Jupyter Notebook

2. Hardware requirements:

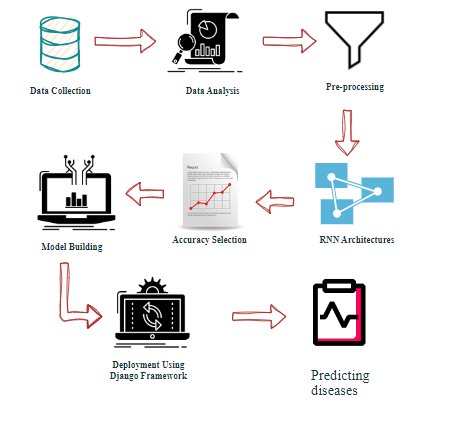
Processor : Pentium IV/III

Hard disk: minimum 80 GB

RAM : minimum 8 GB

**DESIGN ARCHITECTURE:**

**System Architecture:**



**Work flow diagram:**

Data Processing and Cleaning

Source Data

Testing Dataset

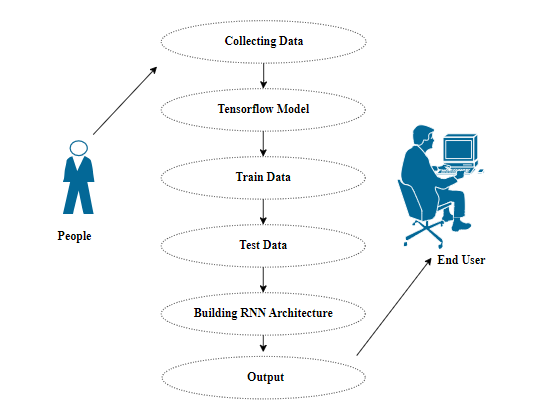
Training Dataset

Best Model by Accuracy

Predicting diseases

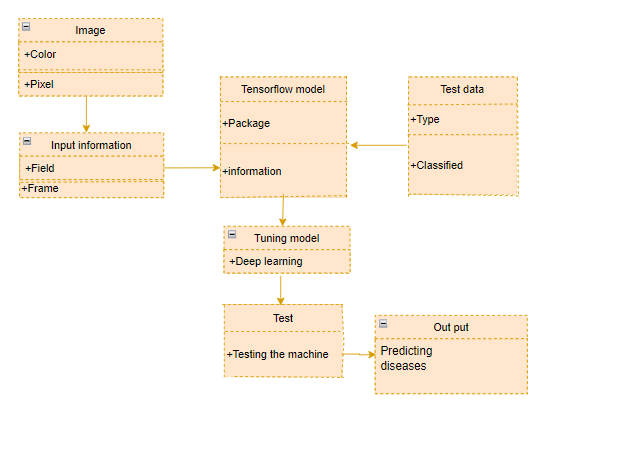
Classification Algorithms

**Use Case Diagram:**



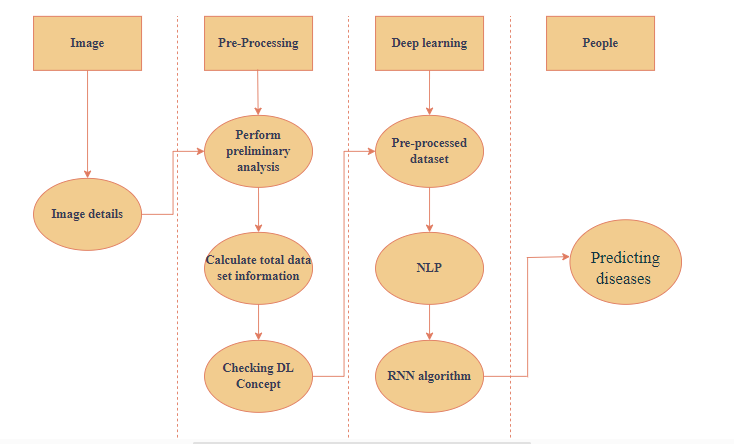
Use case diagrams are considered for high level requirement analysis of a system. So when the requirements of a system are analyzed the functionalities are captured in use cases. So, it can say that uses cases are nothing but the system functionalities written in an organized manner.

**Class Diagram:**



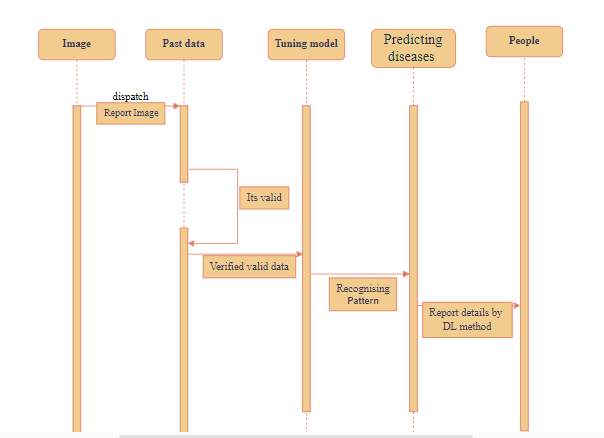
Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. So a collection of class diagrams represent the whole system. The name of the class diagram should be meaningful to describe the aspect of the system. Each element and their relationships should be identified in advance Responsibility (attributes and methods) of each class should be clearly identified for each class minimum number of properties should be specified and because, unnecessary properties will make the diagram complicated. Use notes whenever required to describe some aspect of the diagram and at the end of the drawing it should be understandable to the developer/coder. Finally, before making the final version, the diagram should be drawn on plain paper and rework as many times as possible to make it correct.

**Activity Diagram:**



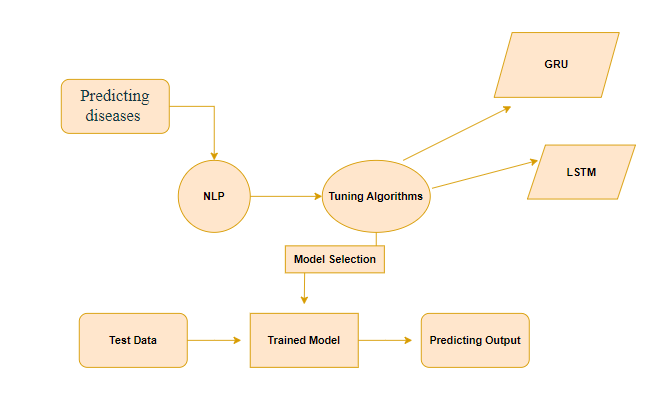
Activity is a particular operation of the system. Activity diagrams are not only used for visualizing dynamic nature of a system but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in activity diagram is the message part. It does not show any message flow from one activity to another. Activity diagram is some time considered as the flow chart. Although the diagrams looks like a flow chart but it is not. It shows different flow like parallel, branched, concurrent and single.

**Sequence Diagram:**



Sequence diagrams model the flow of logic within your system in a visual manner, enabling you both to document and validate your logic, and are commonly used for both analysis and design purposes. Sequence diagrams are the most popular UML artifact for dynamic modelling, which focuses on identifying the behaviour within your system. Other dynamic modelling techniques include [activity diagramming](http://agilemodeling.com/artifacts/activityDiagram.htm), [communication diagramming](http://agilemodeling.com/artifacts/communicationDiagram.htm), [timing diagramming](http://agilemodeling.com/artifacts/timingDiagram.htm), and [interaction overview diagramming](http://agilemodeling.com/artifacts/interactionOverviewDiagram.htm). Sequence diagrams, along with [class diagrams](http://agilemodeling.com/artifacts/classDiagram.htm) and [physical data models](http://agiledata.org/essays/dataModeling101.html) are in my opinion the most important design-level models for modern business application development.

**Entity Relationship Diagram (ERD):**



An entity relationship diagram (ERD), also known as an entity relationship model, is a graphical representation of an information system that depicts the relationships among people, objects, places, concepts or events within that system. An ERD is a [data modeling](https://searchdatamanagement.techtarget.com/definition/data-modeling) technique that can help define business processes and be used as the foundation for a [relational database](https://searchdatamanagement.techtarget.com/definition/relational-database). Entity relationship diagrams provide a visual starting point for database design that can also be used to help determine information system requirements throughout an organization. After a relational database is rolled out, an ERD can still serve as a referral point, should any debugging or business process re-engineering be needed later.

**SOFTWARE DESCRIPTION**

Anaconda is a free and open-source distribution of the [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) and [R](https://en.wikipedia.org/wiki/R_(programming_language)) programming languages for [scientific computing](https://en.wikipedia.org/wiki/Scientific_computing) ([data science](https://en.wikipedia.org/wiki/Data_science), [machine learning](https://en.wikipedia.org/wiki/Machine_learning) applications, large-scale data processing, [predictive analytics](https://en.wikipedia.org/wiki/Predictive_analytics), etc.), that aims to simplify [package management](https://en.wikipedia.org/wiki/Package_management) and deployment. Package versions are managed by the [package management system](https://en.wikipedia.org/wiki/Package_manager) “Conda”. The Anaconda distribution is used by over 12 million users and includes more than 1400 popular data-science packages suitable for Windows, Linux, and MacOS. So, Anaconda distribution comes with more than 1,400 packages as well as the [Conda](https://en.wikipedia.org/wiki/Conda_(package_manager)" \t "Conda (package manager)) package and virtual environment manager called Anaconda Navigator and it eliminates the need to learn to install each library independently. The open source packages can be individually installed from the Anaconda repository with the conda install command or using the pip install command that is installed with Anaconda. [Pip packages](https://en.wikipedia.org/wiki/Pip_(package_manager)) provide many of the features of conda packages and in most cases they can work together. Custom packages can be made using the conda build command, and can be shared with others by uploading them to Anaconda Cloud, [PyPI](https://en.wikipedia.org/wiki/Python_Package_Index" \t "Python Package Index) or other repositories. The default installation of Anaconda2 includes Python 2.7 and Anaconda3 includes Python 3.7. However, you can create new environments that include any version of Python packaged with conda.

**ANACONDA NAVIGATOR**

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda® distribution that allows you to launch applications and easily manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on Anaconda.org or in a local Anaconda Repository.

Anaconda. Now, if you are primarily doing data science work, Anaconda is also a great option. Anaconda is created by Continuum Analytics, and it is a Python distribution that comes preinstalled with lots of useful python libraries for data science.

Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment.

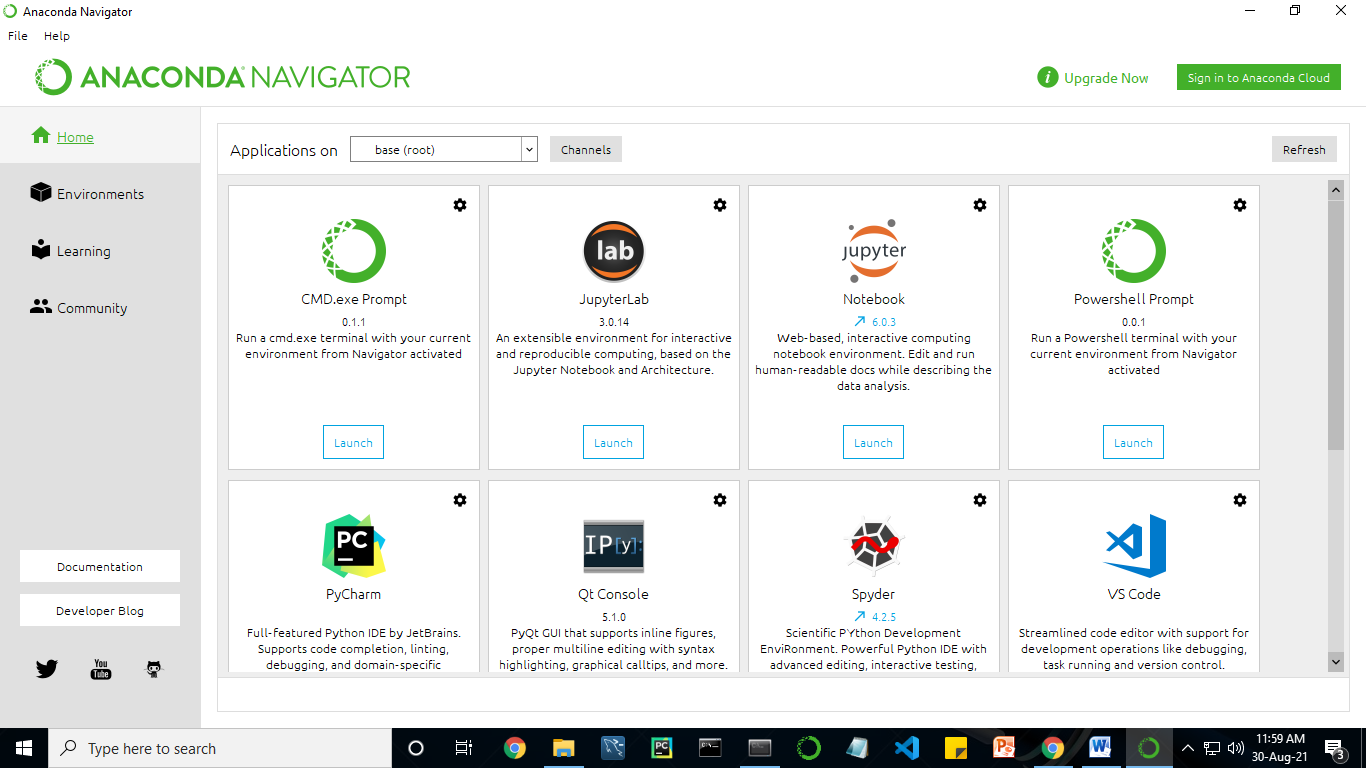
In order to run, many scientific packages depend on specific versions of other packages. Data scientists often use multiple versions of many packages and use multiple environments to separate these different versions.

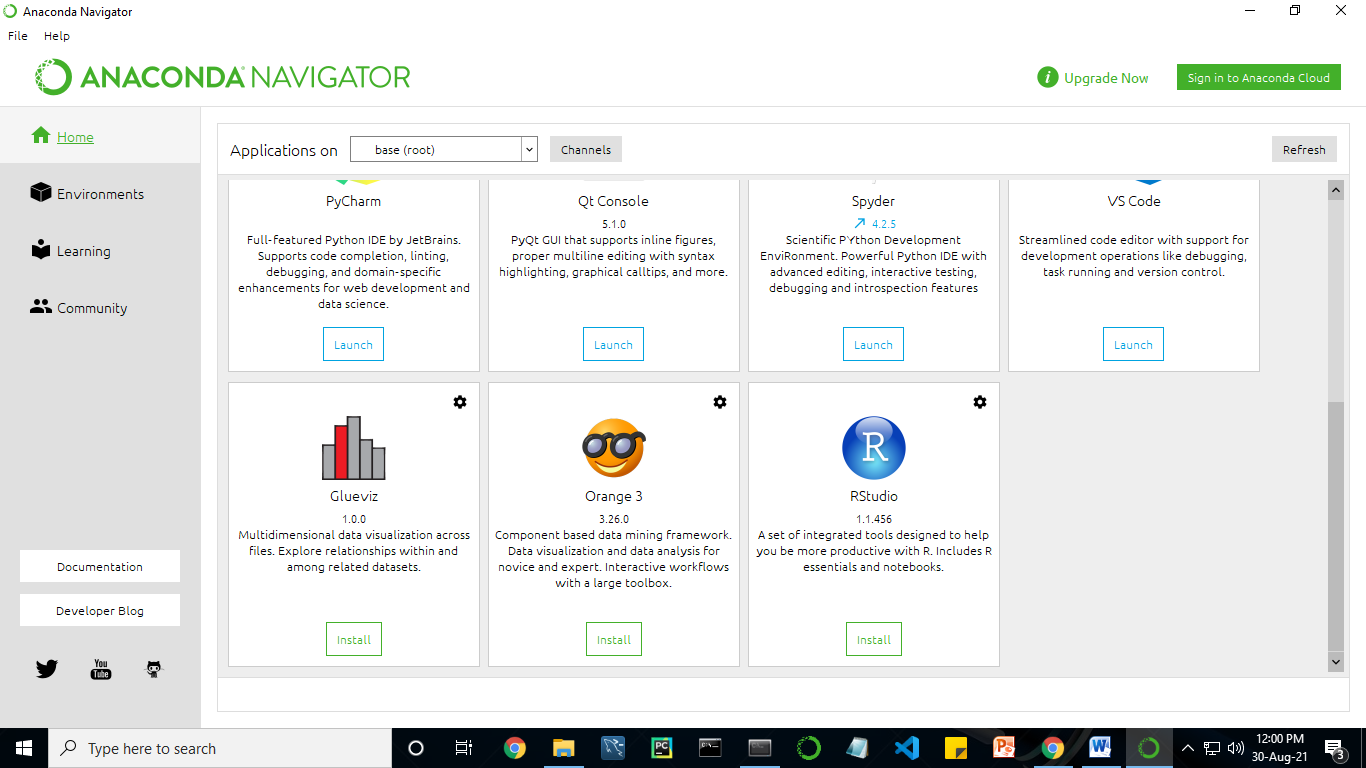
The command-line program conda is both a package manager and an environment manager. This helps data scientists ensure that each version of each package has all the dependencies it requires and works correctly.

Navigator is an easy, point-and-click way to work with packages and environments without needing to type conda commands in a terminal window. You can use it to find the packages you want, install them in an environment, run the packages, and update them – all inside Navigator.

The following applications are available by default in Navigator:

* [JupyterLab](https://jupyterlab.readthedocs.io/en/stable/)
* [Jupyter Notebook](https://jupyter.readthedocs.io/en/latest/)
* [Spyder](https://www.spyder-ide.org/)
* [PyCharm](https://www.jetbrains.com/pycharm/documentation/)
* [VSCode](https://code.visualstudio.com/docs)
* [Glueviz](http://glueviz.org/en/stable/)
* [Orange 3 App](http://orange.biolab.si/docs/)
* [RStudio](http://docs.rstudio.com/)
* Anaconda Prompt (Windows only)
* Anaconda PowerShell (Windows only)





Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution.

Navigator allows you to launch common Python programs and easily manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository.

Anaconda comes with many built-in packages that you can easily find with conda list on your anaconda prompt. As it has lots of packages (many of which are rarely used), it requires lots of space and time as well. If you have enough space, time and do not want to burden yourself to install small utilities like JSON, YAML, you better go for Anaconda.

**Conda :**

Conda is an open source, cross-platform, language-agnostic package manager and environment management systemthat installs, runs, and updates packages and their dependencies. It was created for Python programs, but it can package and distribute software for any language (e.g., R), including multi-language projects. The conda package and environment manager is included in all versions of Anaconda, Miniconda, and Anaconda Repository.

Anaconda is freely available, open source distribution of python and R programming languages which is used for scientific computations. If you are doing any machine learning or deep learning project then this is the best place for you. It consists of many softwares which will help you to build your machine learning project and deep learning project. These softwares have great graphical user interface and these will make your work easy to do. You can also use it to run your python script. These are the software carried by anaconda navigator.

**JUPYTER NOTEBOOK**

This website acts as “meta” documentation for the Jupyter ecosystem. It has a collection of resources to navigate the tools and communities in this ecosystem, and to help you get started.

Project Jupyter is a project and community whose goal is to "develop open-source software, open-standards, and services for interactive computing across dozens of programming languages". It was spun off from IPython in 2014 by Fernando Perez.

Notebook documents are documents produced by the [Jupyter Notebook App](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html" \l "notebook-app), which contain both computer code (e.g. python) and rich text elements (paragraph, equations, figures, links, etc…). Notebook documents are both human-readable documents containing the analysis description and the results (figures, tables, etc.) as well as executable documents which can be run to perform data analysis.

## Installation: The easiest way to install the Jupyter Notebook App is installing a scientific python distribution which also includes scientific python packages. The most common distribution is called **Anaconda**

# Running the Jupyter Notebook

## Launching Jupyter Notebook App: The [Jupyter Notebook App](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html" \l "notebook-app) can be launched by clicking on the Jupyter Notebook icon installed by Anaconda in the start menu (Windows) or by typing in a terminal (cmd on Windows): “jupyter notebook”

## This will launch a new browser window (or a new tab) showing the [Notebook Dashboard](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#dashboard), a sort of control panel that allows (among other things) to select which notebook to open.

## When started, the [Jupyter Notebook App](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html" \l "notebook-app) can access only files within its start-up folder (including any sub-folder). No configuration is necessary if you place your notebooks in your home folder or subfolders. Otherwise, you need to choose a [Jupyter Notebook App](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html" \l "notebook-app) start-up folder which will contain all the notebooks.

## Save notebooks: Modifications to the notebooks are automatically saved every few minutes. To avoid modifying the original notebook, make a copy of the notebook document (menu file -> make a copy…) and save the modifications on the copy.

## **Executing a notebook:**

## Download the notebook you want to execute and put it in your notebook folder (or a sub-folder of it).

* Launch the jupyter notebook app
* In the [Notebook Dashboard](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#dashboard) navigate to find the notebook: clicking on its name will open it in a new browser tab.
* Click on the menu Help -> User Interface Tour for an overview of the [Jupyter Notebook App](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html" \l "notebook-app) user interface.
* You can run the notebook document step-by-step (one cell a time) by pressing shift + enter.
* You can run the whole notebook in a single step by clicking on the menu Cell -> Run All.
* To restart the [kernel](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#kernel) (i.e. the computational engine), click on the menu Kernel -> Restart. This can be useful to start over a computation from scratch (e.g. variables are deleted, open files are closed, etc…).

[Purpose](https://www.google.com/search?q=project+jupyter+purpose&sa=X&ved=2ahUKEwin49vtmdjyAhXx4zgGHXSOCuwQ6BMoADAkegQINxAC&cshid=1630307847256010): To support [interactive](https://www.google.com/search?q=interactive&stick=H4sIAAAAAAAAAONgVuLUz9U3MM0uyYpfxMqdmVeSWpSYXJJZlgoApkTFPhsAAAA&sa=X&ved=2ahUKEwin49vtmdjyAhXx4zgGHXSOCuwQmxMoATAkegQINxAD&cshid=1630307847256010) data science and scientific computing across all programming languages.

File Extension: An IPYNB file is a notebook document created by Jupyter Notebook, an interactive computational environment that helps scientists manipulate and analyze data using Python.

**JUPYTER Notebook App:**

The Jupyter Notebook App is a server-client application that allows editing and running [notebook documents](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-document) via a web browser.

The Jupyter Notebook App can be executed on a local desktop requiring no internet access (as described in this document) or can be installed on a remote server and accessed through the internet.

In addition to displaying/editing/running notebook documents, the Jupyter Notebook App has a “Dashboard” ([Notebook Dashboard](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#dashboard)), a “control panel” showing local files and allowing to open notebook documents or shutting down their [kernels](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#kernel).

## [**Kernel**](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#id7): A notebook kernel is a “computational engine” that executes the code contained in a [Notebook document](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-document). The ipython kernel, referenced in this guide, executes python code. Kernels for many other languages exist ([official kernels](http://jupyter.readthedocs.org/en/latest/#kernels)).

When you open a [Notebook document](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-document), the associated kernel is automatically launched. When the notebook is executed (either cell-by-cell or with menu Cell -> Run All), the kernel performs the computation and produces the results.

Depending on the type of computations, the kernel may consume significant CPU and RAM. Note that the RAM is not released until the kernel is shut-down

## [**Notebook Dashboard**](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#id8)**:**

## The Notebook Dashboard is the component which is shown first when you launch [Jupyter Notebook App](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html" \l "notebook-app). The Notebook Dashboard is mainly used to open [notebook documents](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#notebook-document), and to manage the running [kernels](https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html#kernel) (visualize and shutdown).

The Notebook Dashboard has other features similar to a file manager, namely navigating folders and renaming/deleting files

**Working Process:**

* Download and install anaconda and get the most useful package for machine learning in Python.
* Load a dataset and understand its structure using statistical summaries and data visualization.
* Machine learning models, pick the best and build confidence that the accuracy is reliable.

Python is a popular and powerful interpreted language. Unlike R, Python is a complete language and platform that you can use for both research and development and developing production systems. There are also a lot of modules and libraries to choose from, providing multiple ways to do each task. It can feel overwhelming.

The best way to get started using Python for machine learning is to complete a project.

* It will force you to install and start the Python interpreter (at the very least).
* It will give you a bird’s eye view of how to step through a small project.
* It will give you confidence, maybe to go on to your own small projects.

When you are applying machine learning to your own datasets, you are working on a project. A machine learning project may not be linear, but it has a number of well-known steps:

* Define Problem.
* Prepare Data.
* Evaluate Algorithms.
* Improve Results.
* Present Results.

The best way to really come to terms with a new platform or tool is to work through a machine learning project end-to-end and cover the key steps. Namely, from loading data, summarizing data, evaluating algorithms and making some predictions.

Here is an overview of what we are going to cover:

1. Installing the Python anaconda platform.
2. Loading the dataset.
3. Summarizing the dataset.
4. Visualizing the dataset.
5. Evaluating some algorithms.
6. Making some predictions.

**PYTHON**

**Introduction:**

Python is an [interpreted](https://en.wikipedia.org/wiki/Interpreted_language) [high-level](https://en.wikipedia.org/wiki/High-level_programming_language) [general-purpose programming language](https://en.wikipedia.org/wiki/General-purpose_programming_language). Its design philosophy emphasizes [code readability](https://en.wikipedia.org/wiki/Code_readability) with its use of [significant indentation](https://en.wikipedia.org/wiki/Off-side_rule). Its [language constructs](https://en.wikipedia.org/wiki/Language_construct) as well as its [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) approach aim to help [programmers](https://en.wikipedia.org/wiki/Programmers) write clear, logical code for small and large-scale projects.

Python is [dynamically-typed](https://en.wikipedia.org/wiki/Type_system#DYNAMIC) and [garbage-collected](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)). It supports multiple [programming paradigms](https://en.wikipedia.org/wiki/Programming_paradigms), including [structured](https://en.wikipedia.org/wiki/Structured_programming) (particularly, [procedural](https://en.wikipedia.org/wiki/Procedural_programming)), object-oriented and [functional programming](https://en.wikipedia.org/wiki/Functional_programming). It is often described as a "batteries included" language due to its comprehensive [standard library](https://en.wikipedia.org/wiki/Standard_library).

[Guido van Rossum](https://en.wikipedia.org/wiki/Guido_van_Rossum) began working on Python in the late 1980s, as a successor to the [ABC programming language](https://en.wikipedia.org/wiki/ABC_(programming_language)), and first released it in 1991 as Python 0.9.0.[Python 2.0 was released in 2000 and introduced new features, such as](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-32)[list comprehensions](https://en.wikipedia.org/wiki/List_comprehension) and a garbage collection system using [reference counting](https://en.wikipedia.org/wiki/Reference_counting). Python 3.0 was released in 2008 and was a major revision of the language that is not completely [backward-compatible](https://en.wikipedia.org/wiki/Backward_compatibility). Python 2 was discontinued with version 2.7.18 in 2020.

Python consistently ranks as one of the most popular programming languages

**History:**

Python was conceived in the late 1980s by [Guido van Rossum](https://en.wikipedia.org/wiki/Guido_van_Rossum) at [Centrum Wiskunde & Informatica](https://en.wikipedia.org/wiki/Centrum_Wiskunde_%26_Informatica) (CWI) in the [Netherlands](https://en.wikipedia.org/wiki/Netherlands) as a successor to [ABC programming language](https://en.wikipedia.org/wiki/ABC_(programming_language)), which was inspired by [SETL](https://en.wikipedia.org/wiki/SETL),  capable of [exception handling](https://en.wikipedia.org/wiki/Exception_handling) and interfacing with the [Amoeba](https://en.wikipedia.org/wiki/Amoeba_(operating_system)) operating system. Its implementation began in December 1989.  Van Rossum shouldered sole responsibility for the project, as the lead developer, until 12 July 2018, when he announced his "permanent vacation" from his responsibilities as Python's [Benevolent Dictator for Life](https://en.wikipedia.org/wiki/Benevolent_Dictator_For_Life), a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker. In January 2019, active Python core developers elected a 5-member "Steering Council" to lead the project.  As of 2021, the current members of this council are Barry Warsaw, Brett Cannon, Carol Willing, Thomas Wouters, and Pablo Galindo Salgado.

Python 2.0 was released on 16 October 2000, with many major new features, including a [cycle-detecting](https://en.wikipedia.org/wiki/Cycle_detection) [garbage collector](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)) and support for [Unicode](https://en.wikipedia.org/wiki/Unicode).

Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely [backward-compatible](https://en.wikipedia.org/wiki/Backward_compatibility). Many of its major features were [backported](https://en.wikipedia.org/wiki/Backporting) to Python 2.6.x and 2.7.x version series. Releases of Python 3 include the 2 to 3 utility, which automates (at least partially) the translation of Python 2 code to Python 3.

Python 2.7's [end-of-life](https://en.wikipedia.org/wiki/End-of-life_(product)) date was initially set at 2015 then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3. No more security patches or other improvements will be released for it. With Python 2's [end-of-life](https://en.wikipedia.org/wiki/End-of-life_(product)), only Python 3.6.x and later are supported.

Python 3.9.2 and 3.8.8 were expeditedas all versions of Python (including 2.7) had security issues, leading to possible [remote code execution](https://en.wikipedia.org/wiki/Remote_code_execution) and [web cache poisoning](https://en.wikipedia.org/wiki/Cache_poisoning).

**Design Philosophy & Feature**

Python is a [multi-paradigm programming language](https://en.wikipedia.org/wiki/Multi-paradigm_programming_language). [Object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming) and [structured programming](https://en.wikipedia.org/wiki/Structured_programming) are fully supported, and many of its features support functional programming and [aspect-oriented programming](https://en.wikipedia.org/wiki/Aspect-oriented_programming) (including by [meta-programming](https://en.wikipedia.org/wiki/Metaprogramming) and [meta-objects](https://en.wikipedia.org/wiki/Metaobject) (magic methods)). Many other paradigms are supported via extensions, including [design by contract](https://en.wikipedia.org/wiki/Design_by_contract) and [logic programming](https://en.wikipedia.org/wiki/Logic_programming).

Python uses [dynamic typing](https://en.wikipedia.org/wiki/Dynamic_typing) and a combination of [reference counting](https://en.wikipedia.org/wiki/Reference_counting) and a cycle-detecting garbage collector for [memory management](https://en.wikipedia.org/wiki/Memory_management). It also features dynamic [name resolution](https://en.wikipedia.org/wiki/Name_resolution_(programming_languages)) ([late binding](https://en.wikipedia.org/wiki/Late_binding)), which binds method and variable names during program execution.

Python's design offers some support for functional programming in the [Lisp](https://en.wikipedia.org/wiki/Lisp_(programming_language)) tradition. It has filter, map and reduce functions; [list comprehensions](https://en.wikipedia.org/wiki/List_comprehension), [dictionaries](https://en.wikipedia.org/wiki/Associative_array), sets, and [generator](https://en.wikipedia.org/wiki/Generator_(computer_programming)) expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from [Haskell](https://en.wikipedia.org/wiki/Haskell_(programming_language)) and [Standard ML](https://en.wikipedia.org/wiki/Standard_ML).

The language's core philosophy is summarized in the document The [Zen of Python](https://en.wikipedia.org/wiki/Zen_of_Python) (PEP 20), which includes [aphorisms](https://en.wikipedia.org/wiki/Aphorism) such as:

* Beautiful is better than ugly.
* Explicit is better than implicit.
* Simple is better than complex.
* Complex is better than complicated.
* Readability counts.

Rather than having all of its functionality built into its core, Python was designed to be highly [extensible](https://en.wikipedia.org/wiki/Extensibility) (with modules). This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with [ABC](https://en.wikipedia.org/wiki/ABC_(programming_language)), which espoused the opposite approach.

Python strives for a simpler, less-cluttered syntax and grammar while giving developers a choice in their coding methodology. In contrast to [Perl](https://en.wikipedia.org/wiki/Perl)'s "[there is more than one way to do it](https://en.wikipedia.org/wiki/There_is_more_than_one_way_to_do_it)" motto, Python embraces a "there should be one— and preferably only one —obvious way to do it" design philosophy. [Alex Martelli](https://en.wikipedia.org/wiki/Alex_Martelli), a [Fellow](https://en.wikipedia.org/wiki/Fellow) at the [Python Software Foundation](https://en.wikipedia.org/wiki/Python_Software_Foundation) and Python book author, writes that "To describe something as 'clever' is not considered a compliment in the Python culture."

Python's developers strive to avoid [premature optimization](https://en.wikipedia.org/wiki/Premature_optimization), and reject patches to non-critical parts of the [C-Python](https://en.wikipedia.org/wiki/CPython) reference implementation that would offer marginal increases in speed at the cost of clarity. When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or use [PyPy](https://en.wikipedia.org/wiki/PyPy" \t "PyPy), a [just-in-time compiler](https://en.wikipedia.org/wiki/Just-in-time_compilation). [Cython](https://en.wikipedia.org/wiki/Cython" \t "Cython) is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

Python's developers aim to keep the language fun to use. This is reflected in its name a tribute to the British comedy group [Monty Python](https://en.wikipedia.org/wiki/Monty_Python) and in occasionally playful approaches to tutorials and reference materials, such as examples that refer to spam and eggs (a reference to a [Monty Python sketch](https://en.wikipedia.org/wiki/Spam_(Monty_Python))) instead of the standard [foo and bar](https://en.wikipedia.org/wiki/Foobar).

A common [neologism](https://en.wikipedia.org/wiki/Neologism) in the Python community is pythonic, which can have a wide range of meanings related to program style. To say that code is pythonic is to say that it uses Python idioms well, that it is natural or shows fluency in the language, and that it conforms to Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called unpythonic.

Users and admirers of Python, especially those considered knowledgeable or experienced, are often referred to as Pythonistas

**Syntax and Semantics:**

Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use [curly brackets](https://en.wikipedia.org/wiki/Curly_bracket_programming_language) to delimit blocks, and semicolons after statements are allowed but are rarely, if ever, used. It has fewer syntactic exceptions and special cases than [C](https://en.wikipedia.org/wiki/C_(programming_language)) or [Pascal](https://en.wikipedia.org/wiki/Pascal_(programming_language)).

**Indentation:**

Main article: [Python syntax and semantics & Indentation](https://en.wikipedia.org/wiki/Python_syntax_and_semantics#Indentation)

Python uses [whitespace](https://en.wikipedia.org/wiki/Whitespace_character) indentation, rather than [curly brackets](https://en.wikipedia.org/wiki/Curly_bracket_programming_language) or keywords, to delimit [blocks](https://en.wikipedia.org/wiki/Block_(programming)). An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents the program's semantic structure. This feature is sometimes termed the [off-side rule](https://en.wikipedia.org/wiki/Off-side_rule), which some other languages share, but in most languages indentation does not have any semantic meaning. The recommended indent size is four spaces.

**Statements and control flow:**

Python's [statements](https://en.wikipedia.org/wiki/Statement_(computer_science)) include:

* The [assignment](https://en.wikipedia.org/wiki/Assignment_(computer_science)) statement, using a single equals sign =.
* The if statement, which conditionally executes a block of code, along with else and elif (a contraction of else-if).
* The for statement, which iterates over an iterable object, capturing each element to a local variable for use by the attached block.
* The while statement, which executes a block of code as long as its condition is true.
* The Try statement, which allows exceptions raised in its attached code block to be caught and handled by except clauses; it also ensures that clean-up code in a finally block will always be run regardless of how the block exits.
* The raise statement, used to raise a specified exception or re-raise a caught exception.
* The class statement, which executes a block of code and attaches its local namespace to a [class](https://en.wikipedia.org/wiki/Class_(computer_science)), for use in object-oriented programming.
* The def statement, which defines a [function](https://en.wikipedia.org/wiki/Function_(computing)) or [method](https://en.wikipedia.org/wiki/Method_(computing)).
* The with statement, which encloses a code block within a context manager (for example, acquiring a [lock](https://en.wikipedia.org/wiki/Lock_(computer_science)) before the block of code is run and releasing the lock afterwards, or opening a [file](https://en.wikipedia.org/wiki/Computer_file) and then closing it), allowing [resource-acquisition-is-initialization](https://en.wikipedia.org/wiki/Resource_acquisition_is_initialization) (RAII) - like behavior and replaces a common try/finally idiom.
* The break statement, exits from a loop.
* The continue statement, skips this iteration and continues with the next item.
* The del statement, removes a variable, which means the reference from the name to the value is deleted and trying to use that variable will cause an error. A deleted variable can be reassigned.
* The pass statement, which serves as a [NOP](https://en.wikipedia.org/wiki/NOP_(code)). It is syntactically needed to create an empty code block.
* The assert statement, used during debugging to check for conditions that should apply.
* The yield statement, which returns a value from a [generator](https://en.wikipedia.org/wiki/Generator_(computer_programming)#Python) function and yield is also an operator. This form is used to implement [co-routines](https://en.wikipedia.org/wiki/Coroutine).
* The return statement, used to return a value from a function.
* The import statement, which is used to import modules whose functions or variables can be used in the current program.

The assignment statement (=) operates by binding a name as a [reference](https://en.wikipedia.org/wiki/Pointer_(computer_programming)) to a separate, dynamically-allocated [object](https://en.wikipedia.org/wiki/Object_(computer_science)). Variables may be subsequently rebound at any time to any object. In Python, a variable name is a generic reference holder and does not have a fixed [data type](https://en.wikipedia.org/wiki/Type_system) associated with it. However, at a given time, a variable will refer to some object, which will have a type. This is referred to as [dynamic typing](https://en.wikipedia.org/wiki/Dynamic_type) and is contrasted with [statically-typed](https://en.wikipedia.org/wiki/Statically-typed) programming languages, where each variable may only contain values of a certain type.

Python does not support [tail call](https://en.wikipedia.org/wiki/Tail_call) optimization or [first-class continuations](https://en.wikipedia.org/wiki/First-class_continuations), and, according to Guido van Rossum, it never will.[[80]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-AutoNT-55-80)[[81]](https://en.wikipedia.org/wiki/Python_(programming_language)#cite_note-AutoNT-56-81) However, better support for [co-routine](https://en.wikipedia.org/wiki/Coroutine)-like functionality is provided, by extending Python's [generators](https://en.wikipedia.org/wiki/Generator_(computer_programming)). Before 2.5, generators were [lazy](https://en.wikipedia.org/wiki/Lazy_evaluation) [iterators](https://en.wikipedia.org/wiki/Iterator); information was passed uni-directionally out of the generator. From Python 2.5, it is possible to pass information back into a generator function, and from Python 3.3, the information can be passed through multiple stack levels.

**Expressions:**

Some Python [expressions](https://en.wikipedia.org/wiki/Expression_(computer_science)) are similar to those found in languages such as C and [Java](https://en.wikipedia.org/wiki/Java_(programming_language)), while some are not:

* Addition, subtraction, and multiplication are the same, but the behaviour of division differs. There are two types of divisions in Python. They are floor division (or integer division) // and floating-point/division. Python also uses the \*\* operator for exponentiation.
* From Python 3.5, the new @ infix operator was introduced. It is intended to be used by libraries such as [NumPy](https://en.wikipedia.org/wiki/NumPy) for [matrix multiplication](https://en.wikipedia.org/wiki/Matrix_multiplication).
* From Python 3.8, the syntax: =, called the 'walrus operator' was introduced. It assigns values to variables as part of a larger expression.
* In Python, == compares by value, versus Java, which compares numerics by value and objects by reference. (Value comparisons in Java on objects can be performed with the equals () method.) Python's is operator may be used to compare object identities (comparison by reference). In Python, comparisons may be chained, for example A<=B<=C.
* Python uses the words and, or, not for or its boolean operators rather than the symbolic &&, ||,! Used in Java and C.
* Python has a type of expression termed a [list comprehension](https://en.wikipedia.org/wiki/List_comprehension#Python) as well as a more general expression termed a [generator](https://en.wikipedia.org/wiki/Generator_(computer_programming)) expression.
* [Anonymous functions](https://en.wikipedia.org/wiki/Anonymous_function) are implemented using [lambda expressions](https://en.wikipedia.org/wiki/Lambda_(programming)); however, these are limited in that the body can only be one expression.
* Conditional expressions in Python are written as x if c else y (different in order of operands from the c ? x : y operator common to many other languages).
* Python makes a distinction between [lists](https://en.wikipedia.org/wiki/List_(computer_science)) and [tuples](https://en.wikipedia.org/wiki/Tuple). Lists are written as [1, 2, 3], are mutable, and cannot be used as the keys of dictionaries (dictionary keys must be [immutable](https://en.wikipedia.org/wiki/Immutable) in Python). Tuples are written as (1, 2, 3), are immutable and thus can be used as the keys of dictionaries, provided all elements of the tuple are immutable. The + operator can be used to concatenate two tuples, which does not directly modify their contents, but rather produces a new tuple containing the elements of both provided tuples. Thus, given the variable t initially equal to (1, 2, 3), executing t = t + (4, 5) first evaluates t + (4, 5), which yields (1, 2, 3, 4, 5), which is then assigned back to t, thereby effectively "modifying the contents" of t, while conforming to the immutable nature of tuple objects. Parentheses are optional for tuples in unambiguous contexts.
* Python features sequence unpacking wherein multiple expressions, each evaluating to anything that can be assigned to (a variable, a writable property, etc.), are associated in an identical manner to that forming tuple literals and, as a whole, are put on the left-hand side of the equal sign in an assignment statement. The statement expects an iterable object on the right-hand side of the equal sign that produces the same number of values as the provided writable expressions when iterated through and will iterate through it, assigning each of the produced values to the corresponding expression on the left.
* Python has a "string format" operator %. This functions analogously ton printf format strings in C, e.g. “spam=%s eggs=%d” % (“blah”,2) evaluates to “spam=blah eggs=2”. In Python 3 and 2.6+, this was supplemented by the format() method of the str class, e.g. “spam={0} eggs={1}”.format(“blah”,2). Python 3.6 added "f-strings": blah = “blah”; eggs = 2; f‘spam={blah} eggs={eggs}’
* Strings in Python can be [concatenated](https://en.wikipedia.org/wiki/Concatenation), by "adding" them (same operator as for adding integers and floats). E.g. “spam” + “eggs” returns “spameggs”. Even if your strings contain numbers, they are still added as strings rather than integers. E.g. “2” + “2” returns “2”.
* Python has various kinds of [string literals](https://en.wikipedia.org/wiki/String_literal):
  + Strings delimited by single or double quote marks. Unlike in [Unix shells](https://en.wikipedia.org/wiki/Unix_shell), [Perl](https://en.wikipedia.org/wiki/Perl) and Perl-influenced languages, single quote marks and double quote marks function identically. Both kinds of string use the backslash (\) as an [escape character](https://en.wikipedia.org/wiki/Escape_character). [String interpolation](https://en.wikipedia.org/wiki/String_interpolation) became available in Python 3.6 as "formatted string literals".
  + Triple-quoted strings, which begin and end with a series of three single or double quote marks. They may span multiple lines and function like [here documents](https://en.wikipedia.org/wiki/Here_document) in shells, Perl and [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language)).
  + [Raw string](https://en.wikipedia.org/wiki/Raw_string) varieties, denoted by prefixing the string literal with an r. Escape sequences are not interpreted; hence raw strings are useful where literal backslashes are common, such as [regular expressions](https://en.wikipedia.org/wiki/Regular_expression) and [Windows](https://en.wikipedia.org/wiki/Microsoft_Windows)-style paths. Compare "@-quoting" in [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)).
* Python has [array index](https://en.wikipedia.org/wiki/Array_index) and [array slicing](https://en.wikipedia.org/wiki/Array_slicing) expressions on lists, denoted as a[Key], a[start:stop] or a[start:stop:step]. Indexes are [zero-based](https://en.wikipedia.org/wiki/Zero-based_numbering), and negative indexes are relative to the end. Slices take elements from the start index up to, but not including, the stop index. The third slice parameter, called step or stride, allows elements to be skipped and reversed. Slice indexes may be omitted, for example a[:] returns a copy of the entire list. Each element of a slice is a [shallow copy](https://en.wikipedia.org/wiki/Shallow_copy).

In Python, a distinction between expressions and statements is rigidly enforced, in contrast to languages such as [Common Lisp](https://en.wikipedia.org/wiki/Common_Lisp), [Scheme](https://en.wikipedia.org/wiki/Scheme_(programming_language)), or [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language)). This leads to duplicating some functionality. For example:

* [List comprehensions](https://en.wikipedia.org/wiki/List_comprehensions) vs. for-loops
* [Conditional](https://en.wikipedia.org/wiki/Conditional_(programming)) expressions vs. if blocks
* The eval() vs. exec() built-in functions (in Python 2, exec is a statement); the former is for expressions, the latter is for statements.

Statements cannot be a part of an expression, so list and other comprehensions or [lambda expressions](https://en.wikipedia.org/wiki/Lambda_(programming)), all being expressions, cannot contain statements. A particular case of this is that an assignment statement such as a=1 cannot form part of the conditional expression of a conditional statement. This has the advantage of avoiding a classic C error of mistaking an assignment operator = for an equality operator == in conditions: if (c==1) {…} is syntactically valid (but probably unintended) C code but if c=1: … causes a syntax error in Python.

**Methods:**

[Methods](https://en.wikipedia.org/wiki/Method_(programming)) on objects are [functions](https://en.wikipedia.org/wiki/Function_(programming)) attached to the object's class; the syntax instance.method(argument) is, for normal methods and functions, [syntactic sugar](https://en.wikipedia.org/wiki/Syntactic_sugar) for Class.method(instance, argument). Python methods have an explicit self parameter access [instance data](https://en.wikipedia.org/wiki/Instance_data), in contrast to the implicit self (or this) in some other object-oriented programming languages (e.g., [C++](https://en.wikipedia.org/wiki/C%2B%2B), Java, [Objective-C](https://en.wikipedia.org/wiki/Objective-C), or [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language))). Apart from this Python also provides methods, sometimes called d-under methods due to their names beginning and ending with double-underscores, to extend the functionality of custom class to support native functions such as print, length, comparison, support for arithmetic operations, type conversion, and many more.

### **Typing:**

Python uses duck typing and has typed objects but untyped variable names. Type constraints are not checked at compile time; rather, operations on an object may fail, signifying that the given object is not of a suitable type. Despite being dynamically-typed, Python is strongly-typed, forbidding operations that are not well-defined (for example, adding a number to a string) rather than silently attempting to make sense of them.

Python allows programmers to define their own types using classes, which are most often used for object-oriented programming. New instances of classes are constructed by calling the class (for example, SpamClass() or EggsClass()), and the classes are instances of the metaclass type (itself an instance of itself), allowing meta-programming and reflection.

Before version 3.0, Python had two kinds of classes: old-style and new-style.The syntax of both styles is the same, the difference being whether the class object is inherited from, directly or indirectly (all new-style classes inherit from object and are instances of type). In versions of Python 2 from Python 2.2 onwards, both kinds of classes can be used. Old-style classes were eliminated in Python 3.0.The long-term plan is to support gradual typing and from Python 3.5, the syntax of the language allows specifying static types but they are not checked in the default implementation, CPython[. An experimental optional static type checker named](https://en.wikipedia.org/wiki/CPython" \t "CPython)mypy supports compile-time type checking.

**LIST OF MODULES:**

* Data Pre-processing
* Data Analysis of Visualization
* GRU
* LSTM
* Deployment

**MODULE DESCRIPTION:**

**Data Pre-processing:**

Validation techniques in machine learning are used to get the error rate of the Machine Learning (ML) model, which can be considered as close to the true error rate of the dataset. If the data volume is large enough to be representative of the population, you may not need the validation techniques. However, in real-world scenarios, to work with samples of data that may not be a true representative of the population of given dataset. To finding the missing value, duplicate value and description of data type whether it is float variable or integer. The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyper parameters.

The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration. The validation set is used to evaluate a given model, but this is for frequent evaluation. It as machine learning engineers use this data to fine-tune the model hyper parameters. Data collection, data analysis, and the process of addressing data content, quality, and structure can add up to a time-consuming to-do list. During the process of data identification, it helps to understand your data and its properties; this knowledge will help you choose which algorithm to use to build your model.

A number of different **data cleaning** tasks using Python’s [Pandas library](https://pandas.pydata.org/) and specifically, it focus on probably the biggest data cleaning task, **missing values** and it able to **more**[**quickly clean data**](https://www.dataoptimal.com/data-cleaning-with-python-2018/). It wants to **spend less time cleaning data**, and more time exploring and modeling.

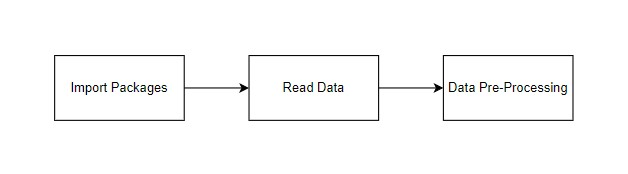
Some of these sources are just simple random mistakes. Other times, there can be a deeper reason why data is missing. It’s important to understand these [different types of missing data](https://en.wikipedia.org/wiki/Missing_data) from a statistics point of view. The type of missing data will influence how to deal with filling in the missing values and to detect missing values, and do some basic imputation and detailed statistical approach for [dealing with missing data](https://github.com/matthewbrems/ODSC-missing-data-may-18/blob/master/Analysis%20with%20Missing%20Data.pdf). Before, joint into code, it’s important to understand the sources of missing data. Here are some typical reasons why data is missing:

* User forgot to fill in a field.
* Data was lost while transferring manually from a legacy database.
* There was a programming error.
* Users chose not to fill out a field tied to their beliefs about how the results would be used or interpreted.

Variable identification with Uni-variate, Bi-variate and Multi-variate analysis:

* import libraries for access and functional purpose and read the given dataset
* General Properties of Analyzing the given dataset
* Display the given dataset in the form of data frame
* show columns
* shape of the data frame
* To describe the data frame
* Checking data type and information about dataset
* Checking for duplicate data
* Checking Missing values of data frame
* Checking unique values of data frame
* Checking count values of data frame
* Rename and drop the given data frame
* To specify the type of values
* To create extra columns

MODULE DIAGRAM



GIVEN INPUT EXPECTED OUTPUT

input : data

output : removing noisy data

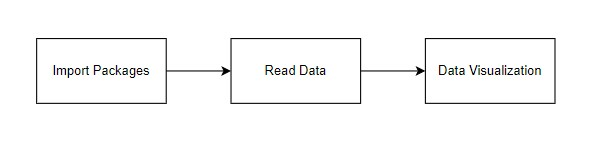
**Data visualization:**

Data visualization is an important skill in applied statistics and machine learning. Statistics does indeed focus on quantitative descriptions and estimations of data. Data visualization provides an important suite of tools for gaining a qualitative understanding. This can be helpful when exploring and getting to know a dataset and can help with identifying patterns, corrupt data, outliers, and much more. With a little domain knowledge, data visualizations can be used to express and demonstrate key relationships in plots and charts that are more visceral and stakeholders than measures of association or significance. Data visualization and exploratory data analysis are whole fields themselves and it will recommend a deeper dive into some the books mentioned at the end.

Sometimes data does not make sense until it can look at in a visual form, such as with charts and plots. Being able to quickly visualize of data samples and others is an important skill both in applied statistics and in applied machine learning. It will discover the many types of plots that you will need to know when visualizing data in Python and how to use them to better understand your own data.

* How to chart time series data with line plots and categorical quantities with bar charts.
* How to summarize data distributions with histograms and box plots.

MODULE DIAGRAM



GIVEN INPUT EXPECTED OUTPUT

Input: data

Output: visualized data

**Algorithm implementation:**

It is important to compare the performance of multiple different machine learning algorithms consistently and it will discover to create a test harness to compare multiple different machine learning algorithms in Python with scikit-learn. It can use this test harness as a template on your own machine learning problems and add more and different algorithms to compare. Each model will have different performance characteristics. Using resampling methods like cross validation, you can get an estimate for how accurate each model may be on unseen data. It needs to be able to use these estimates to choose one or two best models from the suite of models that you have created. When have a new dataset, it is a good idea to visualize the data using different techniques in order to look at the data from different perspectives. The same idea applies to model selection. You should use a number of different ways of looking at the estimated accuracy of your machine learning algorithms in order to choose the one or two to finalize. A way to do this is to use different visualization methods to show the average accuracy, variance and other properties of the distribution of model accuracies.

The below 2 different algorithms are compared:

* GRU Architecture
* Long-Short term memory networks

**GRATED RECURREND UNIT ARCHIETECTURE:**

Implementing the Gated Recurrent Unit (GRU) architecture involves utilizing a type of recurrent neural network (RNN) that incorporates gating mechanisms to manage information flow. By combining reset and update gates, GRU efficiently captures long-range dependencies in sequential data, making it suitable for tasks like Predicting diseases where contextual understanding is crucial for accurate classification.

**Basic Structure:**

An RNN consists of a series of interconnected layers. At each time step t, it takes an input vector (or sequence) and produces an output vector (or sequence).

The key feature of an RNN is its hidden state, denoted as "h." This hidden state is a representation of the network's memory, and it is updated at each time step.

**Input and Output:**

At each time step t, the RNN takes an input vector or element x(t). This input can be a single element of a sequence, a word in a sentence, a pixel in an image, etc.

The RNN produces an output vector or element y(t) at each time step. The output can be used for various tasks, such as predicting the next element in a sequence or classifying the sequence as a whole.

**Hidden State:**

The hidden state h(t) is a vector that captures information from previous time steps. It serves as the memory of the network.

The hidden state is computed at each time step using the current input x(t) and the previous hidden state h(t-1).

**Output Computation:**

The output at each time step can be computed based on the current hidden state or a combination of the hidden state and the input at that time step.

**Backpropagation Through Time (BPTT):**

Training an RNN involves using a variant of backpropagation called Backpropagation through Time.

It is similar to standard backpropagation but accounts for the sequential nature of the data.

The gradients are computed at each time step and accumulated over the entire sequence to update the network's weights.

**Issues with Standard RNNs:**

Standard RNNs have limitations, including the vanishing gradient problem, which makes it challenging to capture long-range dependencies in sequences.

To address these issues, more advanced RNN architectures have been developed, such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), which are designed to better capture long-term dependencies.

In summary, an RNN is a neural network architecture that can process sequential data by maintaining a hidden state that captures information from previous time steps. It is a fundamental building block for various sequence-based tasks in machine learning and deep learning.

MODULE DIAGRAM



GIVEN INPUT EXPECTED OUTPUT

Input: data

Output: getting accuracy

**LSTM ARCHITECTURE:**

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture designed to address the vanishing gradient problem and capture long-term dependencies in sequential data. It consists of memory cells, input gates, forget gates, and output gates. The memory cells store information over long sequences, while the gates regulate the flow of information, allowing LSTMs to effectively learn and remember patterns in time-series data. In the context of Predicting diseases architecture helps the model understand the context of user comments and detect spam based on intricate patterns and dependencies within the text.

**Basic Structure:**

An LSTM network is composed of LSTM cells arranged in a sequence. Each LSTM cell has an internal structure that enables it to store and retrieve information over long sequences.

Like standard RNNs, LSTM networks take input vectors or elements sequentially and produce output vectors or elements at each time step.

The key innovation in LSTM cells is their ability to maintain a cell state, which can capture long-term dependencies in the data.

**Components of an LSTM Cell:**

An LSTM cell consists of three main gates and a cell state:

Forget Gate: Decides what information from the cell state should be thrown away or kept.

Input Gate: Determines what new information should be added to the cell state.

Output Gate: Controls what information from the cell state should be used to generate the output.

Cell State: The cell state runs throughout the entire sequence and can carry information over long distances.

**Information Flow:**

The forget gate (f(t)) controls what information from the previous cell state (C(t-1)) should be retained.

The input gate (i(t)) determines what new information from the candidate cell state (ĉ(t)) should be added to the cell state.

The cell state (C(t)) is updated based on the forget gate, input gate, and candidate cell state.

The output gate (o(t)) controls what information from the cell state should be used to produce the hidden state (h(t)).

**Backpropagation Through Time (BPTT):**

LSTM networks are trained using Backpropagation Through Time, similar to standard RNNs. BPTT computes gradients for the network's parameters to minimize a loss function.

**Advantages of LSTMs:**

LSTMs can capture long-range dependencies in sequences.

They mitigate the vanishing gradient problem, allowing for more effective training on long sequences.

They are suitable for a wide range of sequence-based tasks and have been extended into more advanced variants like Gated Recurrent Units (GRUs).

In summary, LSTM networks are a type of recurrent neural network that incorporates memory cells with gates to selectively store, update, and retrieve information over long sequences. This architecture has proven effective in capturing complex patterns in sequential data and has become a cornerstone of deep learning in fields that involve sequences.

MODULE DIAGRAM



GIVEN INPUT EXPECTED OUTPUT

input: data

output: getting accuracy

**Deployment:**

**Django (Web Framework):**

Django is a micro web framework written in Python.

It is classified as a micro-framework because it does not require particular tools or libraries.

It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.

However, Django supports extensions that can add application features as if they were implemented in Django itself.

Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.

Django was created by [Armin Ronacher](https://en.wikipedia.org/wiki/Armin_Ronacher) of Pocoo, an international group of Python enthusiasts formed in 2004. According to Ronacher, the idea was originally an [April Fool’s](https://en.wikipedia.org/wiki/April_Fool's) joke that was popular enough to make into a serious application. The name is a play on the earlier [Bottle](https://en.wikipedia.org/wiki/Bottle_(web_framework)) framework.

When Ronacher and Georg Brand created a bulletin board system written in Python, the Pocoo projects Werkzeug and [Jinja](https://en.wikipedia.org/wiki/Jinja_(template_engine)) were developed.

In April 2016, the Pocoo team was disbanded and development of Django and related libraries passed to the newly formed Pallets project.

Django has become popular among Python enthusiasts. As of October 2020, it has second most stars on [GitHub](https://en.wikipedia.org/wiki/GitHub) among Python web-development frameworks, only slightly behind Django, and was voted the most popular web framework in the Python Developers Survey 2018.

The micro-framework Django is part of the Pallets Projects, and based on several others of them.

Django **is** based on Werkzeug, [Jinja2](http://quintagroup.com/cms/python/jinja2) and inspired by Sinatra Ruby framework, available under BSD licence. It was developed at pocoo by Armin Ronacher. Although Django is rather young compared to most [Python](https://quintagroup.com/services/python) frameworks, it holds a great promise and has already gained popularity among Python web developers. Let’s take a closer look into Django, so-called “micro” framework for Python.

**FEATURES:**

Django was designed to be **easy to use and extend**.  The idea behind Django is to build a solid foundation for web applications of different complexity. From then on you are free to**plug in any extensions** you think you need. Also you are free to build your own modules. Django is great for all kinds of projects.  It's especially good for prototyping. Django depends on two external libraries: the Jinja2 template engine and the Werkzeug WSGI toolkit.

Still the question remains why use Django as your web application framework if we have immensely powerful [Django](https://quintagroup.com/services/python/django), [Pyramid,](https://quintagroup.com/cms/python/pyramid) and don’t forget web mega-framework [Turbo-gears](https://quintagroup.com/cms/python/turbogears)? Those are supreme[Python web frameworks](https://quintagroup.com/services/python/python-web-development.png) BUT out-of-the-box Django is pretty impressive too with it’s:

* Built-In Development server and Fast debugger
* integrated support for unit testing
* RESTful request dispatching
* Uses [Jinja2](https://quintagroup.com/cms/python/jinja2) Templating
* support for secure cookies
* Unicode based
* Extensive Documentation
* Google App Engine Compatibility
* Extensions available to enhance features desired

Plus Django gives you so much more **CONTROL** on the development stage of **your project**. It follows the principles of minimalism and let you decide  how you will build your application.

* Django has a lightweight and modular design, so it easy to transform it to the web framework you need with a few extensions without weighing it down
* ORM-agnostic: you can plug in your favourite ORM e.g. [SQLAlchemy](https://quintagroup.com/cms/python/sqlalchemy).
* Basic foundation API is nicely shaped and coherent.
* Django documentation is comprehensive, full of examples and well structured. You can even try out some sample application to really get a feel of Django.
* It is super easy to deploy Django in production (Django is 100%WSGI 1.0 compliant”)
* HTTP request handling functionality
* High Flexibility

The configuration is even more flexible than that of Django, giving you plenty of solution for every production need.

To sum up, Django is one of the most polished and feature-rich micro frameworks, available. Still young, Django has a thriving community, first-class extensions, and an **elegant API**.  Django comes with all the benefits of fast templates, strong WSGI features, **thorough unit testability** at the web application and library level, **extensive documentation**. So next time you are starting a new project where you need some good features and a vast number of extensions, definitely check out Django.

Django is an API of Python that allows us to build up web-applications. It was developed by Armin Ronacher. Django's framework is more explicit than Django framework and is also easier to learn because it has less base code to implement a simple web-Application

Django is a micro web framework written in Python. It is classified as a micro-framework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.

Overview of Python Django Framework Web apps are developed to generate content based on retrieved data that changes based on a user’s interaction with the site. The server is responsible for querying, retrieving, and updating data. This makes web applications to be slower and more complicated to deploy than static websites for simple applications.

Django is an excellent web development framework for REST API creation. It is built on top of Python which makes it powerful to use all the python features.

Django is used for the backend, but it makes use of a templating language called Jinja2 which is used to create HTML, XML or other markup formats that are returned to the user via an HTTP request.

Django is considered to be more popular because it provides many out of box features and reduces time to build complex applications. Django is a good start if you are getting into web development. Django is a simple, un-opinionated framework; it doesn't decide what your application should look like developers do.

Django is a web framework. This means Django provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, and a wiki or go as big as a web-based calendar application or a commercial website.

**Advantages of Django:**

* Higher compatibility with latest technologies.
* Technical experimentation.
* Easier to use for simple cases.
* Codebase size is relatively smaller.
* High scalability for simple applications.
* Easy to build a quick prototype.
* Routing URL is easy.
* Easy to develop and maintain applications.

Framework Django is a web framework from Python language. Django provides a library and a collection of codes that can be used to build websites, without the need to do everything from scratch. But Framework Django still doesn't use the Model View Controller (MVC) method.

Django-RESTful is an extension for Django that provides additional support for building REST APIs. You will never be disappointed with the time it takes to develop an API. Django-Restful is a lightweight abstraction that works with the existing ORM/libraries. Django-RESTful encourages best practices with minimal setup.

Django Restful is an extension for Django that adds support for building REST APIs in Python using Django as the back-end. It encourages best practices and is very easy to set up. Django restful is very easy to pick up if you're already familiar with Django.

Django is a web framework for Python, meaning that it provides functionality for building web applications, including managing HTTP requests and rendering templates and also we can add to this application to create our API.

Start Using an API

1. Most APIs require an API key. ...
2. The easiest way to start using an API is by finding an HTTP client online, like REST-Client, Postman, or Paw.
3. The next best way to pull data from an API is by building a URL from existing API documentation.

The Django object implements a WSGI application and acts as the central object. It is passed the name of the module or package of the application. Once it is created it will act as a central registry for the view functions, the URL rules, template configuration and much more.

The name of the package is used to resolve resources from inside the package or the folder the module is contained in depending on if the package parameter resolves to an actual python package (a folder with an \_\_init\_\_.py file inside) or a standard module (just a .py file).

For more information about resource loading, see [open resource()](https://flask.palletsprojects.com/en/2.0.x/api/#flask.Flask.open_resource).

Usually you create a [Django](https://flask.palletsprojects.com/en/2.0.x/api/#flask.Flask) instance in your main module or in the \_\_init\_\_.py file of your package.

Parameters

* **rule** ([str](https://docs.python.org/3/library/stdtypes.html#str)) – The URL rule string.
* **endpoint** (Optional[[str](https://docs.python.org/3/library/stdtypes.html#str)]) – The endpoint name to associate with the rule and view function. Used when routing and building URLs. Defaults to view\_func.\_\_name\_\_.
* **view\_func** (Optional[Callable]) – The view function to associate with the endpoint name.
* **provide\_automatic\_options** (Optional[bool]) – Add the OPTIONS method and respond to OPTIONS requests automatically.
* **options** (Any) – Extra options passed to the [Rule](https://werkzeug.palletsprojects.com/en/2.0.x/routing/#werkzeug.routing.Rule) object.

Return type -- [None](https://docs.python.org/3/library/constants.html#None)

After\_Request(f)

Register a function to run after each request to this object.

The function is called with the response object, and must return a response object. This allows the functions to modify or replace the response before it is sent.

If a function raises an exception, any remaining after request functions will not be called. Therefore, this should not be used for actions that must execute, such as to close resources. Use [teardown\_request()](https://flask.palletsprojects.com/en/2.0.x/api/" \l "flask.Flask.teardown_request" \t "flask.Flask.teardown_request) for that.

**Parameters:**

**f** (Callable[[[Response](https://flask.palletsprojects.com/en/2.0.x/api/#flask.Response)], [Response](https://flask.palletsprojects.com/en/2.0.x/api/#flask.Response)])

Return type

Callable[[[Response](https://flask.palletsprojects.com/en/2.0.x/api/#flask.Response)], [Response](https://flask.palletsprojects.com/en/2.0.x/api/#flask.Response)]

after\_request\_funcs: t.Dict[AppOrBlueprintKey,

t.List[AfterRequestCallable]]

A data structure of functions to call at the end of each request, in the format {scope: [functions]}. The scope  key is the name of a blueprint the functions are active for, or None for all requests.

To register a function, use the [after\_request()](https://flask.palletsprojects.com/en/2.0.x/api/" \l "flask.Flask.after_request" \t "flask.Flask.after_request) decorator.

This data structure is internal. It should not be modified directly and its format may change at any time.

app\_context()

Create an [AppContext](https://flask.palletsprojects.com/en/2.0.x/api/" \l "flask.ctx.AppContext" \t "flask.ctx.AppContext). Use as a with block to push the context, which will make [current\_app](https://flask.palletsprojects.com/en/2.0.x/api/" \l "flask.current_app" \t "flask.current_app) point at this application.

An application context is automatically pushed by [RequestContext.push()](https://flask.palletsprojects.com/en/2.0.x/api/" \l "flask.ctx.RequestContext.push" \t "flask.ctx.RequestContext.push) when handling a request, and when running a CLI command. Use this to manually create a context outside of these situations.

With app.app\_context():

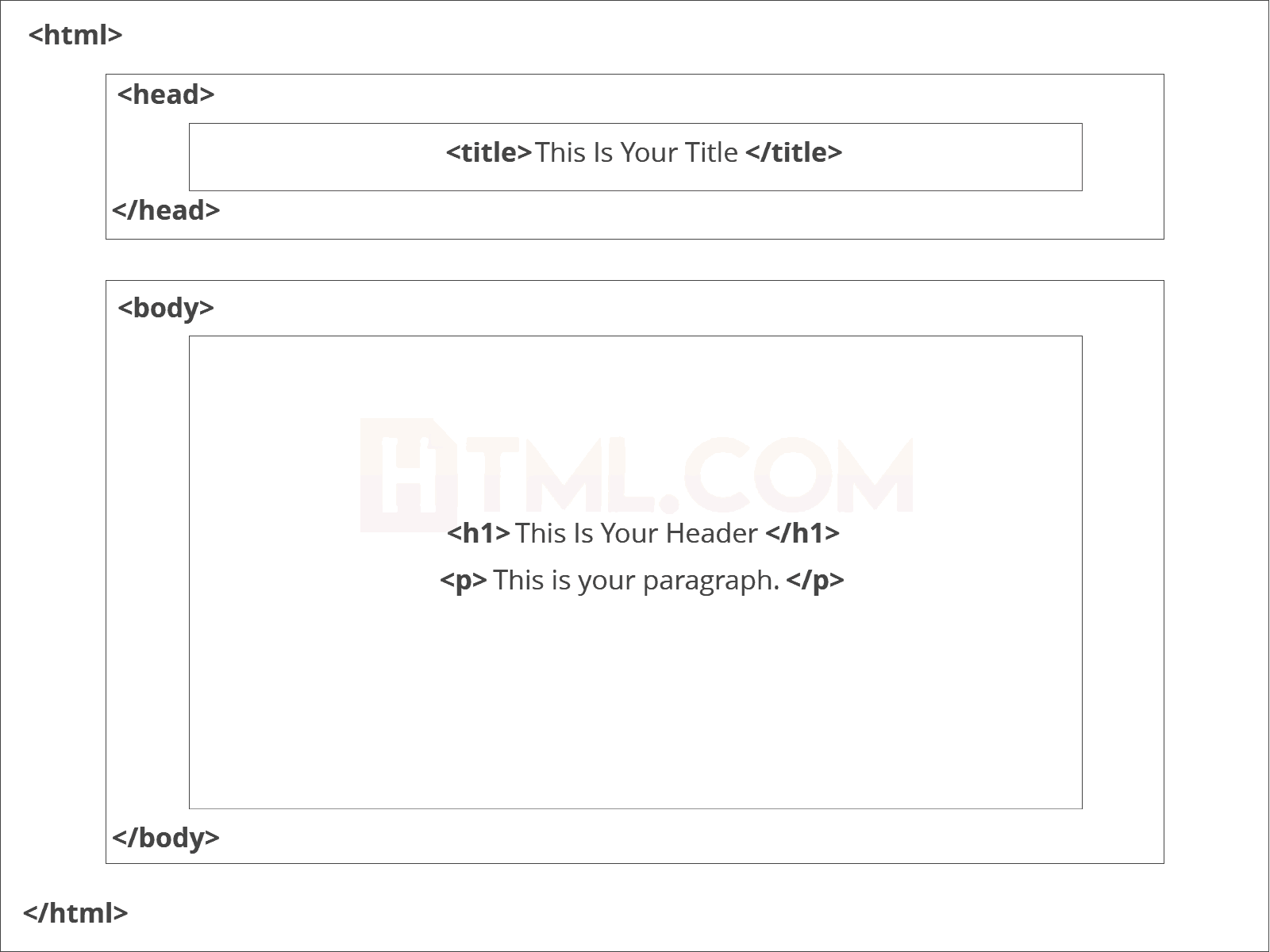
Init\_db()

**HTML INTRODUCTION:**

HTML stands for Hyper Text Markup Language. It is used to design web pages using a markup language. HTML is the combination of Hypertext and Markup language. Hypertext defines the link between the web pages. A markup language is used to define the text document within tag which defines the structure of web pages. This language is used to annotate (make notes for the computer) text so that a machine can understand it and manipulate text accordingly. Most markup languages (e.g. HTML) are human-readable. The language uses tags to define what manipulation has to be done on the text.

#### Basic Construction of an HTML Page

These tags should be placed underneath each other **at the top of every HTML page** that you create.



<!DOCTYPE html> — This tag **specifies the language** you will write on the page. In this case, the language is HTML 5.

<html> — This tag signals that from here on we are going to write in HTML code.

<head>[— This is where all the](https://html.com/tags/head/)**metadata for the page** goes — stuff mostly meant for search engines and other computer programs.

<body>[— This is where the](https://html.com/tags/body/)**content of the page** goes.

#### Further Tags

Inside the <head> tag, there is one tag that is always included: <title>, but there are others that are just as important:

<title>

This is where we **insert the page name** as it will appear at the top of the browser window or tab.

<meta>

This is where information about the document is stored: character encoding, name (page context), description.

Head Tag  
<head>

<title>My First Webpage</title>

<meta charset="UTF-8">

<meta name="description" content="This field contains information about your page. It is usually around two sentences long.">.

<meta name="author" content="Conor Sheils">

</header>

### Adding Content

Next, we will make<body> tag.

The HTML <body> is where we add the content which is designed for viewing by human eyes.

This includes **text, images, tables, forms**and everything else that we see on the internet each day.

#### Add HTML Headings To Web Page

In HTML, [headings](https://html.com/tags/heading/) are written in the following elements:

* <h1>
* <h2>
* <h3>
* <h4>
* <h5>
* <h6>

As you might have guessed <h1> and <h2> should be used for the most important titles, while the remaining tags should be used for sub-headings and less important text.

**Search engine bots use this order** when deciphering which information is most important on a page.

##### Creating Your Heading

Let’s try it out. On a new line in the HTML editor, type:

<h1> Welcome To My Page </h1>

And hit save. We will save this file as “index.html” in a new folder called “my webpage.”

Add Text In HTML

Adding text to our HTML page is simple using an element opened with the tag <p> which **creates a new paragraph**. We place all of our regular text inside the element <p>.

When we write text in HTML, we also have a number of other elements we can use to **control the text or make it appear in a certain way**.

#### Add Links In HTML

As you may have noticed, the internet is made up of lots of [links](https://html.com/anchors-links/).

Almost everything you click on while surfing the web is a link **takes you to another page** within the website you are visiting or to an external site.

Links are included in an attribute opened by the [**<a>**](https://html.com/tags/a/) tag. This element is the first that we’ve met which uses an attribute and so it **looks different to previously mentioned tags**.

<a href=[http://www.google.com](http://www.google.com/)>Google</a>

Image Tag

In today’s modern digital world, [images](https://html.com/blog/100-legal-sources-free-stock-images/) are everything. The [<img>](https://html.com/tags/img/) tag has everything you need to display images on your site. Much like the <a> anchor element, <img> also contains an attribute.

The attribute features information for your computer regarding the source, height, width and alt text of the image

<img src=”yourimage.jpg” alt=”Describe the image” height=“X” width=“X”>

**CASCADING STYLE SHEETS:**

CSS stands for Cascading Style Sheets. It is the language for describing the presentation of Web pages, including colours, layout, and fonts, thus making our web pages presentable to the users.CSS is designed to make style sheets for the web. It is independent of HTML and can be used with any XML-based markup language. Now let’s try to break the acronym:

* Cascading: Falling of Styles
* Style: Adding designs/Styling our HTML tags
* Sheets: Writing our style in different documents

## **CSS Syntax**

Selector {

Property 1 : value;

Property 2 : value;

Property 3 : value;

}

For example:

h1

{

Color: red;

Text-align: center;

}

#unique

{

color: green;

}

* Selector: selects the element you want to target
* Always remains the same whether we apply internal or external styling
* There are few basic selectors like tags, id’s, and classes
* All forms this key-value pair
* Keys: properties(attributes) like color, font-size, background, width, height,etc
* Value: values associated with these properties

## **CSS Comment**

* Comments don’t render on the browser
* Helps to understand our code better and makes it readable.
* Helps to debug our code
* Two ways to  comment:
  + Single line

## **CSS How-To**

* There are 3 ways to write CSS in our HTML file.
  + Inline CSS
  + Internal CSS
  + External CSS
* Priority order
  + Inline > Internal > External

**Inline CSS**

* Before CSS this was the only way to apply styles
* Not an efficient way to write as it has a lot of redundancy
* Self-contained
* Uniquely applied on each element
* The idea of separation of concerns was lost
* Example:

<h3 style = “color:red”> Have a great day </h3>

<p style = “color:green”> I did this, I did that </p>

**Internal CSS**

* With the help of style tag, we can apply styles within the HTML file
* Redundancy is removed
* But the idea of separation of concerns still lost
* Uniquely applied on a single document
* Example:

<style>

H1{

Color:red;

}

</style>

<h3> Have a great day </h3>

**External CSS**

* With the help of <link> tag in the head tag, we can apply styles
* Reference is added
* File saved with .css extension
* Redundancy is removed
* The idea of separation of concerns is maintained
* Uniquely applied to each document
* Example:

<head>

<link rel= “stylesheet” type= “text/css” href= “name of the CSS file”>

</head>

h1{

color:red; //.css file

}

## **CSS Selectors**

* The selector is used to target elements and apply CSS
* Three simple selectors
  + Element Selector
  + Id Selector
  + Class Selector
* Priority of Selectors

## **CSS Colors**

* There are different colouring schemes in CSS
* RGB-This starts with RGB and takes 3 parameter
* HEX-Hex code starts with # and comprises of 6 numbers which are further divided into 3 sets
* RGBA-This starts with RGB and takes 4 parameter

**CSS Background**

* There are different ways by which CSS can have an effect on HTML elements
* Few of them are as follows:
  + Color – used to set the color of the background
  + Repeat – used to determine if the image has to repeat or not and if it is repeating then how it should do that
  + Image – used to set an image as the background
  + Position – used to determine the position of the image
  + Attachment – It basically helps in controlling the mechanism of scrolling.

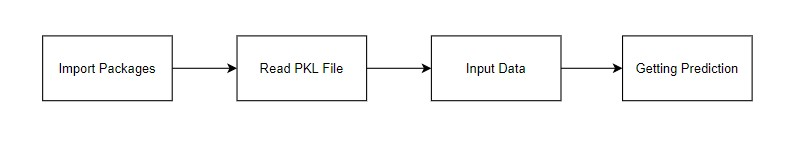
## **CSS BoxModel**

* Every element in CSS can be represented using the BOX model
* It allows us to add a border and define space between the content
* It helps the developer to develop and manipulate the elements
* It consists of 4 edges
  + Content edge – It comprises of the actual content
  + Padding edge – It lies in between content and border edge
  + Border edge – Padding is followed by the border edge
  + Margin edge – It is an outside border and controls the margin of the element

**Deploying the model predicting output**

In this module the trained machine learning model is converted into pickle data format file (.pkl file) which is then deployed for providing better user interface and predicting the deprssion of a person.

MODULE DIAGRAM



GIVEN INPUT EXPECTED OUTPUT

Input: data values

Output: predicting output

**CODING:**

**Module:1**

**#****DATA PREPROCESSING AND DATA CLEANING:**

In [ ]:

import pandas as pd

import numpy as np

In [ ]:

Data = pd.read\_csv('SYMPTOMS.csv', usecols=['label','text'])

Data.head()

In [ ]:

Data.tail()

In [ ]:

Data.shape

In [ ]:

Data = Data.dropna()

In [ ]:

Data.shape

In [ ]:

Data.size

In [ ]:

Data.isnull().sum()

In [ ]:

Data.info()

In [ ]:

Data.columns

In [ ]:

Data['label'].unique()

In [ ]:

Data['label'].value\_counts()

In [ ]:

Data.groupby('label').describe()

#### BEFORE LABEL ENCODER

In [ ]:

Data.head()

In [ ]:

from sklearn.preprocessing import LabelEncoder

var\_mod = ['text','label']

le = LabelEncoder()

for i in var\_mod:

Data[i] = le.fit\_transform(Data[i]).astype(int)

#### AFTER LABEL ENCODER

In [ ]:

Data.head()

In [ ]:

Data.duplicated()

In [ ]:

Data.duplicated().sum()

In [ ]:

Data = Data.drop\_duplicates()

In [ ]:

Data.duplicated().sum()

In [ ]:

**Module:2**

**#****DATA VISUALIZATION AND DATA ANALYSIS**

In [ ]:

import pandas as pd

import numpy as np

In [ ]:

import matplotlib.pyplot as plt

import seaborn as sns

In [ ]:

Data = pd.read\_csv('SYMPTOMS.csv', usecols=['label','text'])

Data.head()

In [ ]:

Data.tail()

In [ ]:

from sklearn.preprocessing import LabelEncoder

label\_encoder = LabelEncoder()

Data['label'] = label\_encoder.fit\_transform(Data['label'])

In [ ]:

sns.countplot(x='label',data=Data)

In [ ]:

plt.hist(Data['label'],color='green')

In [ ]:

Data['label'].plot(kind='density')

In [ ]:

sns.displot(Data['label'], color='purple')

In [ ]:

sns.violinplot(Data['label'], color='yellow')

In [ ]:

sns.ecdfplot(Data['label'], color='blue')

In [ ]:

sns.displot(Data['label'], color='RED')

In [ ]:

import re

from nltk.corpus import stopwords

from nltk.stem import PorterStemmer

from nltk.tokenize import word\_tokenize

In [ ]:

# Define preprocess function for text preprocessing

def preprocess\_text(text):

# Check for NaN values and handle them

if pd.isnull(text):

return ""

# Convert to lowercase

text = text.lower()

# Remove special characters and digits

text = re.sub(r'[^a-zA-Z\s]', '', text)

# Tokenization and remove stop words

stop\_words = set(stopwords.words('english'))

words = [word for word in word\_tokenize(text) if word not in stop\_words]

# Stemming

ps = PorterStemmer()

words = [ps.stem(word) for word in words]

# Join the preprocessed words back into a single string

preprocessed\_text = ' '.join(words)

return preprocessed\_text

In [ ]:

Data['text'] = Data['text'].apply(preprocess\_text)

In [ ]:

from sklearn.model\_selection import train\_test\_split

X,X\_test,y,y\_test = train\_test\_split(Data.loc[:,'text':],Data['label'],test\_size=0.2)

In [ ]:

from wordcloud import WordCloud

import matplotlib.pyplot as plt

In [ ]:

ACNE = ' '.join(Data.loc[Data['label'] == 0, 'text'].values)

ACNE\_text = WordCloud(background\_color='GREEN',max\_words=2000,width = 800, height = 800).generate(ACNE)

plt.figure(figsize=[10,30])

plt.imshow(ACNE\_text,interpolation='bilinear')

plt.title('ACNE DISEASE')

plt.axis('off')

In [ ]:

ALLERGY = ' '.join(Data.loc[Data['label'] == 1, 'text'].values)

ALLERGY\_text = WordCloud(background\_color='RED',max\_words=2000,width = 800, height = 800).generate(ALLERGY)

plt.figure(figsize=[10,30])

plt.imshow(ALLERGY\_text, interpolation='bilinear')

plt.axis('off')

plt.title('ALLERGY')

In [ ]:

CHICKENPOX = ' '.join(Data.loc[Data['label'] == 5, 'text'].values)

CHICKENPOX\_text = WordCloud(background\_color='PINK',max\_words=2000,width = 800, height = 800).generate(CHICKENPOX)

plt.figure(figsize=[10,30])

plt.imshow(CHICKENPOX\_text, interpolation='bilinear')

plt.axis('off')

plt.title('CHICKENPOX')

In [ ]:

**Module:3**

**#****GRATED RECURREND UNIT ARCHITECTURE**

In [ ]:

import pandas as pd

import numpy as np

In [ ]:

Data = pd.read\_csv('SYMPTOMS.csv', usecols=['label','text'])

Data.head()

In [ ]:

Data.tail()

In [ ]:

Data['label'].value\_counts()

In [ ]:

Data['text'] = Data['text'].apply(lambda x: x.lower() if pd.notna(x) else "")

In [ ]:

from sklearn.preprocessing import LabelEncoder

label\_encoder = LabelEncoder()

Data['label'] = label\_encoder.fit\_transform(Data['label'])

In [ ]:

num\_classes = len(label\_encoder.classes\_)

In [ ]:

x = Data['text']

y = Data['label']

In [ ]:

from tensorflow.keras.utils import to\_categorical

y = to\_categorical(y, num\_classes=num\_classes)

In [ ]:

from sklearn.model\_selection import train\_test\_split

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

In [ ]:

max\_words = 10000

max\_sequence\_length = 100

In [ ]:

from tensorflow.keras.preprocessing.text import Tokenizer

tokenizer = Tokenizer(num\_words=max\_words)

tokenizer.fit\_on\_texts(x\_train)

In [ ]:

x\_train\_sequences = tokenizer.texts\_to\_sequences(x\_train)

x\_test\_sequences = tokenizer.texts\_to\_sequences(x\_test)

In [ ]:

from tensorflow.keras.preprocessing.sequence import pad\_sequences

x\_train\_padded = pad\_sequences(x\_train\_sequences, maxlen=max\_sequence\_length)

x\_test\_padded = pad\_sequences(x\_test\_sequences, maxlen=max\_sequence\_length)

In [ ]:

embedding\_dim = 100

RNN\_units = 128

In [ ]:

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding

from tensorflow.keras.layers import Bidirectional

from tensorflow.keras.layers import GRU

from tensorflow.keras.layers import Dense

In [ ]:

model = Sequential()

model.add(Embedding(input\_dim=max\_words, output\_dim=embedding\_dim, input\_length=max\_sequence\_length))

model.add(Bidirectional(GRU(units=RNN\_units, dropout=0.2, recurrent\_dropout=0.2)))

model.add(Dense(units=num\_classes, activation='softmax'))

In [ ]:

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

In [ ]:

from tensorflow.keras.callbacks import ModelCheckpoint

model\_checkpoint = ModelCheckpoint('DISEASE1.h5',

monitor='accuracy',

save\_best\_only=True,

verbose=1,

mode='max')

In [ ]:

epochs = 10

batch\_size = 32

In [ ]:

model.fit(x\_train\_padded, y\_train, epochs=epochs, batch\_size=batch\_size, validation\_split=0.1,callbacks=[model\_checkpoint])

In [ ]:

y\_pred = model.predict(x\_test\_padded)

y\_pred\_classes = np.argmax(y\_pred, axis=1)

y\_true\_classes = np.argmax(y\_test, axis=1)

In [ ]:

from sklearn.metrics import accuracy\_score

AC = accuracy\_score(y\_pred\_classes,y\_true\_classes)

print("THE ACCURACY SCORE OF GRATED RECURREND UNIT ARCHITECTURE IS :",AC\*100)

In [ ]:

from sklearn.metrics import hamming\_loss

HL = hamming\_loss(y\_pred\_classes,y\_true\_classes)

print("THE HAMMING LOSS OF GRATED RECURREND UNIT ARCHITECTURE IS :",HL\*100)

In [ ]:

from sklearn.metrics import classification\_report

CR = classification\_report(y\_pred\_classes,y\_true\_classes)

print('THE PRECISION SCORE OF GRATED RECURREND UNIT ARCHITECTURE:\n\n\n',CR)

In [ ]:

from sklearn.metrics import confusion\_matrix

CM = confusion\_matrix(y\_pred\_classes,y\_true\_classes)

print('THE CONFUSION MATRIX SCORE OF GRATED RECURREND UNIT ARCHITECTURE:\n\n\n',CM)

In [ ]:

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

cm = confusion\_matrix(y\_true\_classes, y\_pred\_classes)

classes = np.arange(cm.shape[0])

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=classes)

disp.plot(cmap='viridis', values\_format='d')

plt.title('DISPLAY CONFUSION MATRIX OF GRATED RECURREND UNIT ARCHITECTURE\n\n')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.show()

In [ ]:

def graph():

import matplotlib.pyplot as plt

data=[AC]

alg="GRATED RECURREND UNIT ARCHITECTURE"

plt.figure(figsize=(5,5))

b=plt.bar(alg,data,color=("RED"))

plt.title("THE ACCURACY SCORE OF GRATED RECURREND UNIT ARCHITECTURE IS\n\n\n")

plt.legend(b,data,fontsize=9)

graph()

In [ ]:

**Module:4**

**#****LSTM ARCHITECTURE**

In [ ]:

import pandas as pd

import numpy as np

In [ ]:

Data = pd.read\_csv('SYMPTOMS.csv', usecols=['label','text'])

Data.head()

In [ ]:

Data.tail()

In [ ]:

Data['label'].value\_counts()

In [ ]:

Data['text'] = Data['text'].apply(lambda x: x.lower() if pd.notna(x) else "")

In [ ]:

from sklearn.preprocessing import LabelEncoder

label\_encoder = LabelEncoder()

Data['label'] = label\_encoder.fit\_transform(Data['label'])

In [ ]:

num\_classes = len(label\_encoder.classes\_)

In [ ]:

x = Data['text']

y = Data['label']

In [ ]:

from tensorflow.keras.utils import to\_categorical

y = to\_categorical(y, num\_classes=num\_classes)

In [ ]:

from sklearn.model\_selection import train\_test\_split

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

In [ ]:

max\_words = 10000

max\_sequence\_length = 100

In [ ]:

from tensorflow.keras.preprocessing.text import Tokenizer

tokenizer = Tokenizer(num\_words=max\_words)

tokenizer.fit\_on\_texts(x\_train)

In [ ]:

X\_train\_sequences = tokenizer.texts\_to\_sequences(x\_train)

X\_test\_sequences = tokenizer.texts\_to\_sequences(x\_test)

In [ ]:

from tensorflow.keras.preprocessing.sequence import pad\_sequences

X\_train\_padded = pad\_sequences(X\_train\_sequences, maxlen=max\_sequence\_length)

X\_test\_padded = pad\_sequences(X\_test\_sequences, maxlen=max\_sequence\_length)

In [ ]:

embedding\_dim = 100

lstm\_units = 128

In [ ]:

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding

from tensorflow.keras.layers import Bidirectional

from tensorflow.keras.layers import LSTM

from tensorflow.keras.layers import Dense

In [ ]:

model = Sequential()

model.add(Embedding(input\_dim=max\_words, output\_dim=embedding\_dim, input\_length=max\_sequence\_length))

model.add(Bidirectional(LSTM(units=lstm\_units, dropout=0.2, recurrent\_dropout=0.2)))

model.add(Dense(units=num\_classes, activation='softmax'))

In [ ]:

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

In [ ]:

from tensorflow.keras.callbacks import ModelCheckpoint

model\_checkpoint = ModelCheckpoint('DISEASE.h5',

monitor='accuracy',

save\_best\_only=True,

verbose=1,

mode='max')

In [ ]:

epochs = 50

batch\_size = 32

In [ ]:

model.fit(X\_train\_padded, y\_train, epochs=epochs, batch\_size=batch\_size, validation\_split=0.1,callbacks=[model\_checkpoint])

In [ ]:

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

y\_pred\_classes = np.argmax(y\_pred, axis=1)

y\_true\_classes = np.argmax(y\_test, axis=1)

In [ ]:

from sklearn.metrics import accuracy\_score

AC = accuracy\_score(y\_true\_classes,y\_pred\_classes)

print("THE ACCURACY SCORE OF LSTM ARCHITECTURE IS :",AC\*100)

In [ ]:

from sklearn.metrics import hamming\_loss

HL = hamming\_loss(y\_true\_classes,y\_pred\_classes)

print("THE HAMMING LOSS OF LSTM ARCHITECTURE IS :",HL\*100)

In [ ]:

from sklearn.metrics import classification\_report

CR = classification\_report(y\_pred\_classes,y\_true\_classes)

print('THE PRECISION SCORE OF LSTM ARCHITECTURE:\n\n\n',CR)

In [ ]:

from sklearn.metrics import confusion\_matrix

CM = confusion\_matrix(y\_true\_classes,y\_pred\_classes)

print('THE CONFUSION MATRIX SCORE OF LSTM ARCHITECTURE:\n\n\n',CM)

In [ ]:

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

# Assuming you have the y\_pred\_classes and y\_true\_classes arrays with predicted and true labels respectively.

# Calculate the confusion matrix

cm = confusion\_matrix(y\_true\_classes, y\_pred\_classes)

# Display the confusion matrix using ConfusionMatrixDisplay

classes = np.arange(cm.shape[0]) # Assuming your classes are integers from 0 to n\_classes-1

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=classes)

disp.plot(cmap='viridis', values\_format='d')

plt.title('DISPLAY CONFUSION MATRIX OF LSTM ARCHITECTURE\n\n')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.show()

In [ ]:

def graph():

import matplotlib.pyplot as plt

data=[AC]

alg="LSTM ARCHITECTURE"

plt.figure(figsize=(5,5))

b=plt.bar(alg,data,color=("YELLOW"))

plt.title("THE ACCURACY SCORE OF LSTM ARCHITECTURE IS\n\n\n")

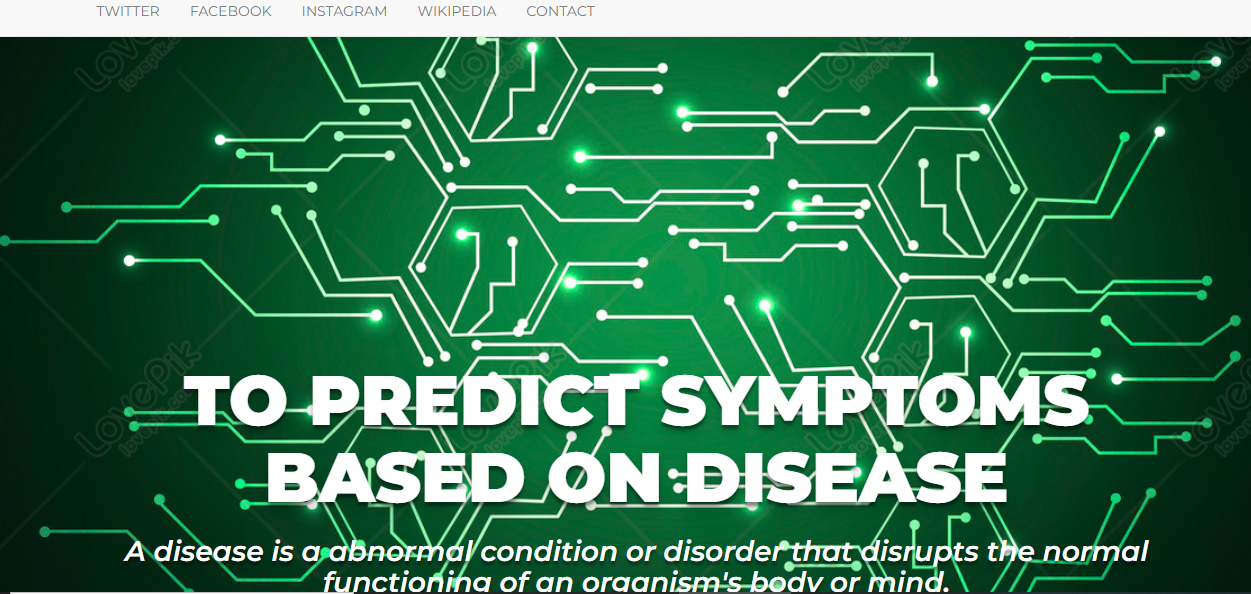
plt.legend(b,data,fontsize=9)

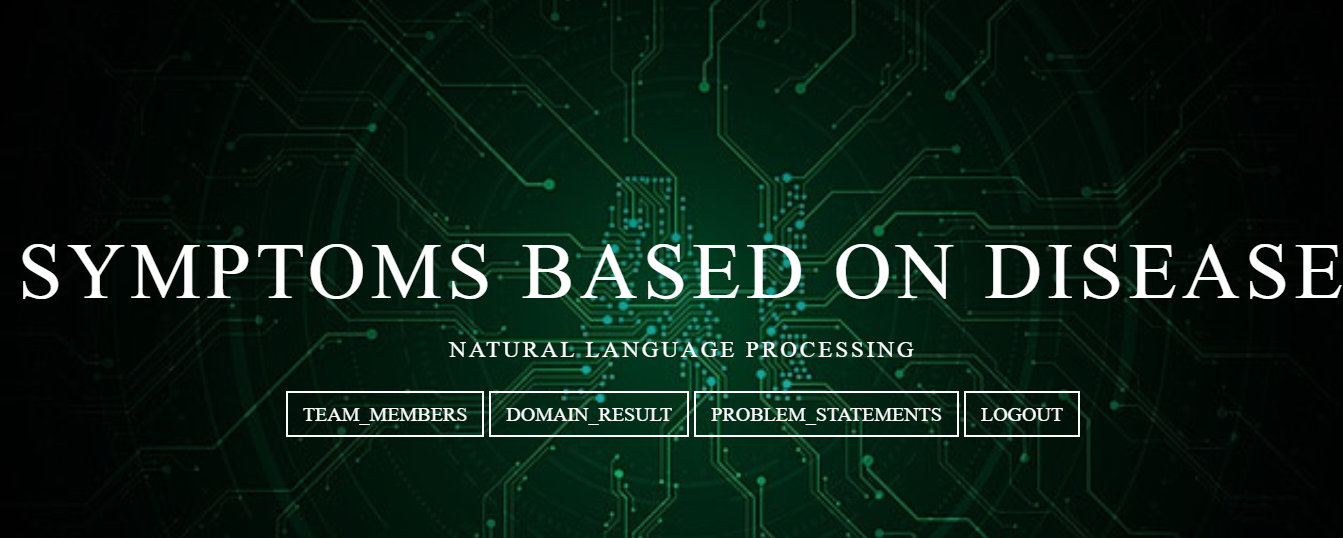
graph()

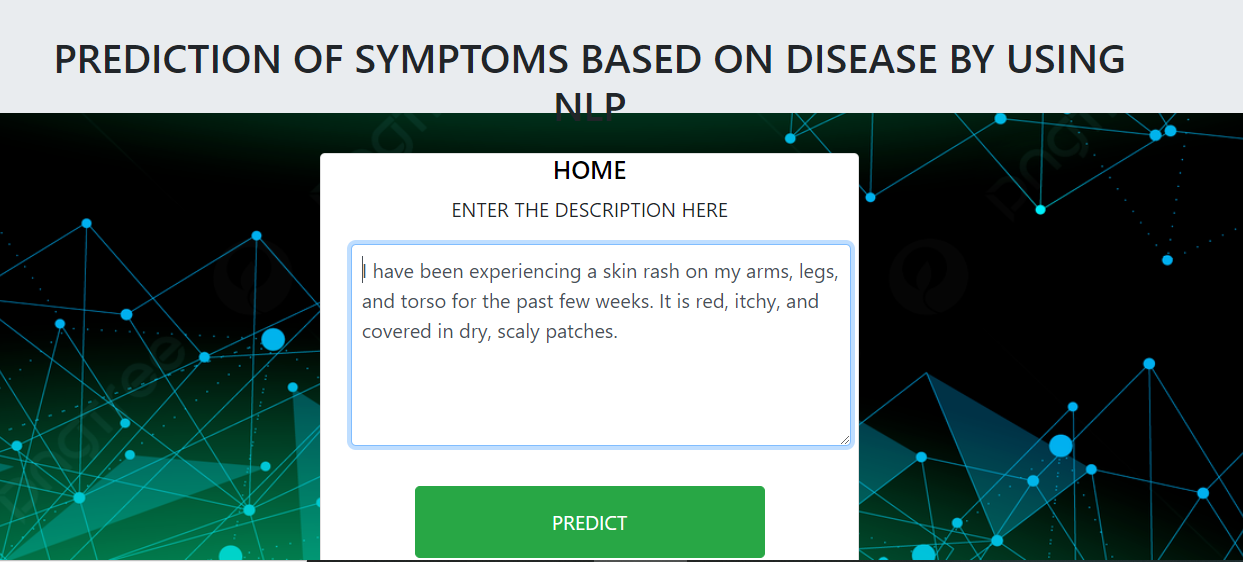
In [ ]:

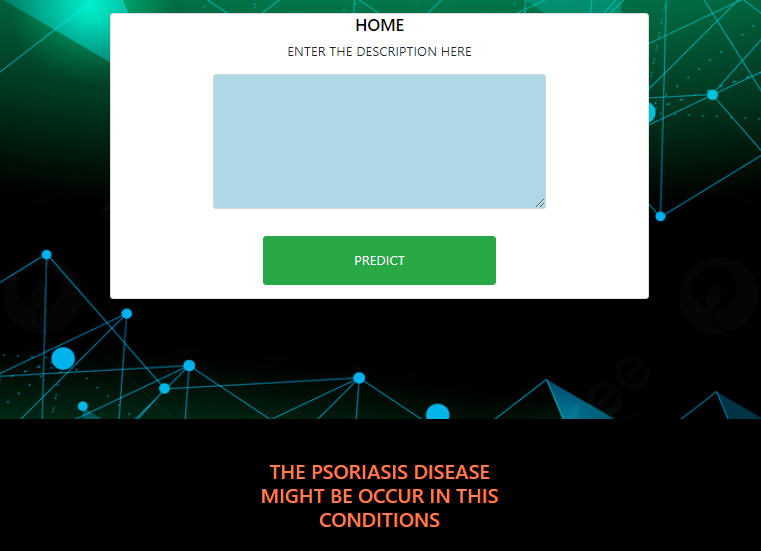
In [ ]:

**OUTPUT:**

****

****

****

****

**CONCLUSION:**

employing Natural Language Processing (NLP) techniques for disease prediction based on symptoms holds tremendous potential for revolutionizing healthcare. The integration of advanced machine learning models and semantic analysis allows for more accurate and timely identification of diseases, facilitating early intervention and personalized medical care. Further research and development in this field could significantly enhance the efficiency of diagnostic processes, ultimately leading to improved patient outcomes and healthcare system effectiveness.

**FUTURE WORK:**

1. Future work can focus on refining disease prediction through NLP by developing advanced semantic embeddings that capture subtle contextual relationships between symptoms, enhancing the model's ability to discern complex patterns in medical text data.
2. Explore the integration of continuous learning mechanisms to adapt and update the disease prediction model over time, accommodating new medical knowledge, evolving symptom patterns, and ensuring the system's relevance in dynamic healthcare environments.