

PREDICTION OF STROKE USING MACHINE LEARNING TECHNIQUE

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering degree in

INFORMATION TECHNOLOGY

By

THILAK DARSHAN.V(Reg No 38120085)

PRAVIN.I(Reg No 38120062)



**DEPARTMENT OF INFORMATION TECHNOLOGY
SCHOOL OF COMPUTING**

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with Grade “A” by NAAC | 12B Status by UGC | Approved by AICTE

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with "A" grade by NAAC | 12B Status by UGC | Approved by AICTE

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai – 600 119

www.sathyabama.ac.in

SCHOOL OF COMPUTING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the Bonafide work of **THILAKDARSHAN.V (Reg no:38120085)** and **PRAVI.I (Reg no:38120062)** carried out the project entitled **"PREDICTION OF STROKE USING MACHINE LEARNING TECHNIQUE"** under our supervision from **OCTOBER 2021** to **JANUARY 2022**

Internal Guide

Head of the Department

**Dr. P. AJITHA M.E., Ph.D.,
Ph.D.,**

Dr. R. SUBHASHINI M.E.,

Submitted for Vivavoce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

We, **THILAK DARSHAN.V** and **PRAVIN.I** hereby declare that the Project Report entitled done by me under the guidance of **Dr. P. AJITHA M.E., Ph.D.**, at Sathyabama Institute of Science and Technology (Deemed to be University), Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai - 600119 is submitted in partial fulfillment of the requirements for the award of Bachelor of Technology degree in Information Technology.

1.

2.

DATE:

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

We are pleased to acknowledge my sincere thanks to Board of management of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. We are grateful to them.

We convey my thanks to **Dr. T. SASIKALA M.E., Ph.D., Dean, School of Computing** and **Dr. R. SUBHASHINI M.E., Ph.D., Head of the Department, Department of Information Technology** for providing me the necessary support and details at the right time during the progressive reviews.

I would like to express our sincere and deep sense of gratitude to our Project Guide

Dr. P. AJITHA M.E., Ph.D.,
for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of our project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the Department of **INFORMATION TECHNOLOGY** who were helpful in many ways for the completion of the project.

PREDICTION OF STROKE USING MACHINE LEARNING TECHNIQUE

TABLE OF CONTENT

SL.NO	TITLE	PAGE.NO
01	ABSTRACT	12
02	EXISTING SYSTEM	13
	2.1 DRAWBACKS	14
03	INTRODUCTION	14
	3.1 DATA SCIENCE	15
	3.2 ARTIFICIAL INTELLIGENCE	16
04	MACHINE LEARNING	21
05	PREPARING DATASET	23
06	PROPOSED SYSTEM	23
	6.1 ADVANTAGES	25
07	LITERATURE SURVEY	25
08	SYSTEM STUDY	31
	8.1 OBJECTIVES	31
	8.2 PROJECT GOAL	31
	8.3 SCOPE OF THE PROJECT	32

09	FEASIBILITY STUDY	32
10	LIST OF MODULES	34
11	PROJECT REQUIREMENTS	35
	11.1 FUNCTIONAL REQUIREMENTS	35
	11.2 NON-FUNCTIONAL REQUIREMENTS	35
12	ENVIRONMENT REQUIREMENT	36
13	SOFTWARE DESCRIPTION	36
	13.1 ANACONDA NAVIGATOR	37
	13.2 JUPYTER NOTEBOOK	41
14	PYTHON	47
15	SYSTEM ARCHITECTURE	60
16	WORKFLOW DIAGRAM	61
17	USECASE DIAGRAM	62
18	CLASS DIAGRAM	63
19	ACTIVITY DIAGRAM	64
20	SEQUENCE DIAGRAM	65
21	ER – DIAGRAM	67
22	MODULE DESCRIPTION	68
	22.1 MODULE DIAGRAM	
	22.2 MODULE GIVEN INPUT EXPECTED OUTPUT	
23	DEPLOYMENT	89

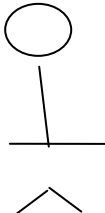
24	HTML	97
25	CSS	101
26	CODING	107
27	CONCLUSION	133
28	FUTURE WORK	133

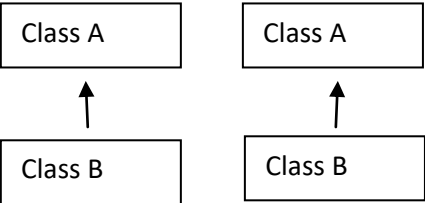
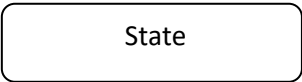
LIST OF FIGURES

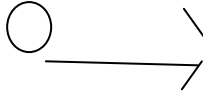
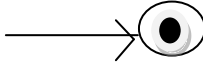
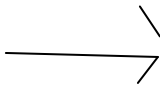
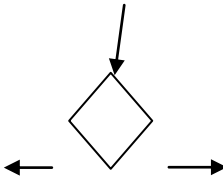
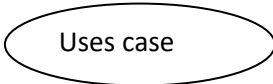
SL.NO	TITLE	PAGE.NO
01	SYSTEM ARCHITECTURE	56
02	WORKFLOW DIAGRAM	57
03	USECASE DIAGRAM	58
04	CLASS DIAGRAM	59
05	ACTIVITY DIAGRAM	60
06	SEQUENCE DIAGRAM	61
07	ER – DIAGRAM	62
08	MODULE DIAGRAM	63

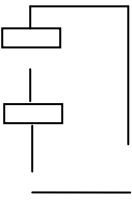
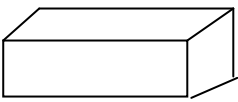
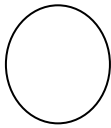
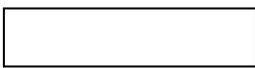
LIST OF SYMBOLS

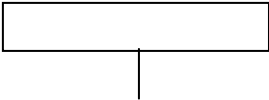
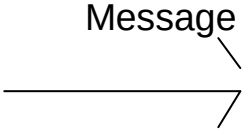
S.NO	NOTATION NAME	NOTATION	DESCRIPTION

1.	Class	+ public - private # protected Class Name - attribute - attribute + operation + operation	Represents a collection of similar entities grouped together.
2.	Association	Class A NAME Class B Class A Class B	Associations represents static relationships between classes. Roles represents the way the two classes see each other.
3.	Actor		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 of the process.

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.	Use case		Interaction 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 action.
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		Represents the message exchanged.

Prediction Of Stroke Using Machine Learning Technique

1. Abstract:

A stroke occurs when the blood supply to part of your brain is interrupted or reduced, preventing brain tissue from getting oxygen and nutrients. Brain cells begin to die in minutes. A heart attack occurs when blood flow to a part of the heart is blocked, usually by a blood clot. Without oxygenated blood, the heart muscle begins to die. A stroke is a brain attack, cutting off vital blood flow and oxygen to the brain. Stroke happens when a blood vessel feeding the brain gets clogged or bursts. Nowadays machine learning is applied to healthcare system where there is a chance of predicting the disease early. The main necessity of Artificial intelligence is data. The past dataset is collected and that dataset is used to build a machine learning model. The necessary pre-processing techniques are applied like univariate analysis and bivariate analysis are implemented. The data is visualised for better understanding of the features and based on that a classification model is built by using machine learning algorithm and comparison of algorithms are done based on their performance metrics like accuracy, F1 score recall etc.

2. Existing System:

Stroke has become a leading cause of death and long-term disability in the world with no effective treatment. Deep learning-based

approaches have the potential to outperform existing stroke risk prediction models, but they rely on large well-labeled data. Due to the strict privacy protection policy in health-care systems, stroke data is usually distributed among different hospitals in small pieces. In addition, the positive and negative instances of such data are extremely imbalanced. Transfer learning can solve small data issue by exploiting the knowledge of a correlated domain, especially when multiple source of data are available. balanced stroke data. We have proposed a novel Hybrid Deep Transfer Learning-based Stroke Risk Prediction (HDTLSRP) framework which consists of three key components: (1) Generative Instance Transfer (GIT) for making use of the external stroke data distribution among multiple hospitals while preserving the privacy, (2) Network Weight Transfer (NWT) for making use of data from highly correlated diseases (i.e., hypertension or diabetes), (3) Active Instance Transfer (AIT) for balancing the stroke data with the most informative generated instances. It is found that the proposed HDTL-SRP framework outperforms the state-of-the-art SRP models in both synthetic and real-world scenarios.

2.1 Disadvantages:

- They are not classifying stroke machine learning classification technique and not mention any accuracy results.
- There are not using any artificial intelligence technique
- It can't thereby better determine the regularity of stroke disease prediction data and achieve more accurate prediction results.

3. INTRODUCTION

Domain overview

3.1 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.

Required Skills for a Data Scientist:

- **Programming:** Python, SQL, Scala, Java, R, MATLAB.
- **Machine Learning:** Natural Language Processing, Classification, Clustering.
- **Data Visualization:** Tableau, SAS, D3.js, Python, Java, R libraries.
- **Big data platforms:** MongoDB, Oracle, Microsoft Azure, Cloudera.

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 demonstrated by machines, as opposed to the natural intelligence displayed by humans or animals. Leading AI textbooks define the field as the study of "intelligent agents" 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, 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.

Natural Language Processing (NLP):

Natural language processing (NLP) allows machines to read and understand human language. A sufficiently powerful natural language processing system would enable natural-language user interfaces and the acquisition of knowledge directly from human-written sources, such as newswire texts. Some straightforward applications of natural language processing include information retrieval, text mining, question answering and machine translation. Many current approaches use word co-occurrence frequencies to construct syntactic representations of text. "Keyword spotting" strategies for search are popular and scalable but dumb; a search query for "dog" might only match documents with the literal word "dog" and miss a document with the word "poodle". "Lexical affinity" strategies use the occurrence of words such as "accident" to assess the sentiment of a document. Modern statistical NLP approaches can combine all these strategies as well as others, and often achieve acceptable accuracy at the page or paragraph level. Beyond semantic NLP, the ultimate goal of "narrative" NLP is to embody a full understanding of commonsense reasoning. By

2019, transformer-based deep learning architectures could generate coherent text.

4. 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.



Supervised Machine 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”.

5. Preparing the Dataset :

This dataset contains 5110 records of features. It is classified into 2 classes:

- Normal
- Stroke

6. Proposed System:

Exploratory Data Analysis of pulmonary disease Prediction

Stroke datasets from different sources would be combined to form a generalized dataset, and then different machine learning algorithms would be applied to extract patterns and to obtain results with maximum accuracy.

Data Wrangling

In this section of the report will load in the data, check for cleanliness, and then trim and clean given dataset for analysis. Make sure that the document steps carefully and justify for cleaning decisions.

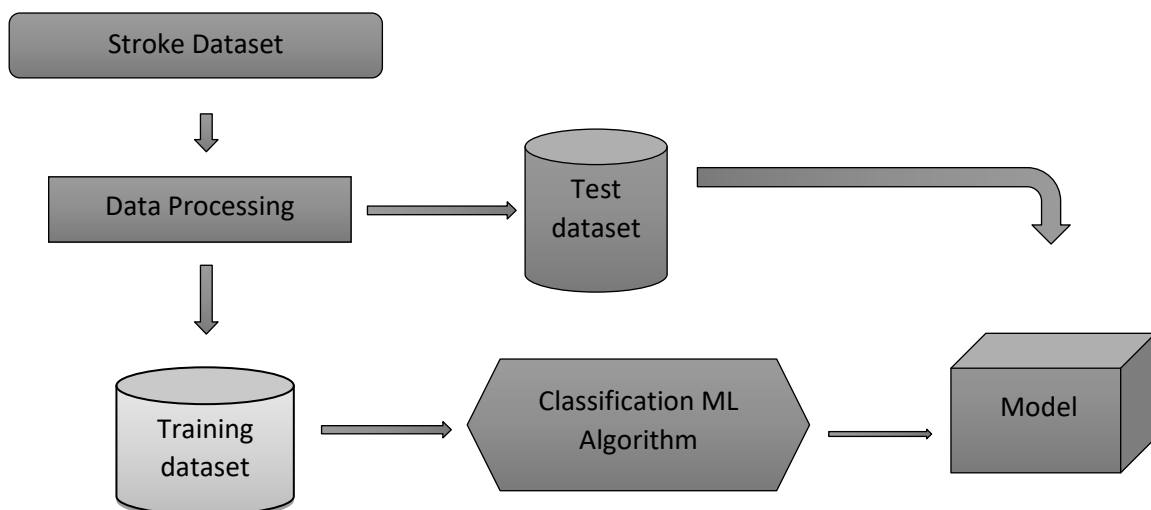
Data collection

The data set collected for predicting given data is split into Training set and Test set. Generally, 7:3 ratios are applied to split the Training set and Test set. The Data Model which was created using machine learning algorithms are applied on the Training set and based on the test result accuracy, Test set prediction is done.

Building the classification model

The predicting Stroke, ML algorithms prediction model is effective because of the following reasons: It provides better results in classification problem.

- It is strong in preprocessing outliers, irrelevant variables, and a mix of continuous, categorical and discrete variables.
- It produces out of bag estimate error which has proven to be unbiased in many tests and it is relatively easy to tune with.



Architecture of Proposed model

6.1 Advantages:

- These reports are to the investigation of applicability of machine learning techniques for stroke prediction in operational conditions. Finally, it highlights some observations on future research issues, challenges, and needs.

7. Literature Survey:

General

A literature review is a body of text that aims to review the critical points of current knowledge on and/or methodological approaches to a particular topic. It is secondary sources and discuss published information in a particular subject area and sometimes information in a particular subject area within a certain time period. Its ultimate goal is to bring the reader up to date with current literature on a topic and forms the basis for another goal, such as future research that may be needed in the area and precedes a research proposal and may be just a simple summary of sources. Usually, it has an organizational pattern and combines both summary and synthesis.

A summary is a recap of important information about the source, but a synthesis is a re-organization, reshuffling of information. It might give a new interpretation of old material or combine new with old interpretations or it might trace the intellectual progression of the field, including major debates. Depending on the situation, the literature review may evaluate the

sources and advise the reader on the most pertinent or relevant of them.

Loan default trends have been long studied from a socio-economic stand point. Most economics surveys believe in empirical modeling of these complex systems in order to be able to predict the loan default rate for a particular individual. The use of machine learning for such tasks is a trend which it is observing now. Some of the survey's to understand the past and present perspective of loan approval or not.

Title : Prediction of Stroke using Deep Learning Model

Author: Pattanapong Chantamit-o-pas and Madhu Goyal

Year : 2007

Many predictive techniques have been widely applied in clinical decision making such as predicting occurrence of a disease or diagnosis, evaluating prognosis or outcome of diseases and assisting clinicians to recommend treatment of diseases. However, the conventional predictive models or techniques are still not effective enough in capturing the underlying knowledge because it is incapable of simulating the complexity on feature representation of the medical problem domains. This research reports predictive analytical techniques for stroke using deep learning model applied on heart disease dataset. The atrial fibrillation symptoms in heart patients are a major risk factor of stroke and share common variables to predict stroke. The outcomes of this research are more accurate than medical scoring

systems currently in use for warning heart patients if they are likely to develop stroke.

Title : Various Approaches for Predicting Stroke Prognosis using Magnetic Resonance Imaging Text Records

Author: Tak-Sung Heo, Chulho Kim, Jeong-Myeong Choi, Yeong-Seok Jeong, Yu-Seop Kim

Year : 2020

Stroke is one of the leading causes of death and disability worldwide. Stroke is treatable, but it is prone to disability after treatment. To grasp the degree of disability caused by stroke, we use magnetic resonance imaging text records to predict stroke and measure the performance according to the document-level and sentence-level representation. As a result of the experiment, the document-level representation shows better performance.

Title : Prediction Of Heart Stroke Diseases Using Machine Learning Technique Based Electromyographic Data

Author: G. Sasikala, G. Roja and D. Radhika

Year : 2021

Stroke is a prominent cause of disability in adults and the elderly, resulting in a slew of social and financial issues. Stroke can be fatal if left untreated. Patients with stroke have been found to have aberrant bio-signals in the majority of cases. Individuals can obtain appropriate therapy more rapidly if they are observed and have their biosignals detected and precisely assessed in real-

time. However, most stroke diagnostic and prediction systems rely on image analysis methods such as CT or MRI, which are costly and difficult to employ for real-time diagnosis. In this paper, we developed a stroke prediction system that detects stroke using real-time bio-signals with machine learning techniques. The purpose of this study was to analyse and diagnose electromyography data using machine learning. The data cleaning was carried out in accordance with the inclusion criteria that were specifically designed. Following that, two data sets were created, each containing 575 facial motor nerve conduction study reports and 233 auditory brainstem response reports. The data sets were then subjected to four machine learning algorithms: Reinforcement Learning, linear regression, support vector machine, and logistic regression. Comparisons of accuracy and recall rate among several algorithms show that the Reinforcement Learning algorithm outperforms the other two in both data sets, Congenital Heart Disease (CHD) and International Stroke Trial (IST). Furthermore, for each algorithm, comparisons were made with and without deviation standardization, and the results show that deviation standardization has an effect on accuracy improvement. The experiment employs three classification algorithms: linear regression, logistic regression, SVM (support vector machine), and Reinforcement Learning. As a result, Reinforcement Learning has been demonstrated to be an optimal algorithm for diagnosis. It is also worth noting that feature ranking in order of importance can facilitate clinical diagnosis and has clinical potential in diagnosis and diagnostic assessment.

Title : A Hybrid Efficient Data Analytics Framework for Stroke Prediction

Author: Hosam Alhakami, Shouq Alraddadi, Shurug Alseady, Abdullah Baz, Tahani Alsubait

Year : 2020

Stroke is considered one of the most universal diseases. To understand this disease in medical sciences, large and complex datasets are collected and analyzed. The analysis of this data has been recognized as a big challenge in modern life. Therefore, there is a need to find effective techniques to deal with such huge datasets. To predict the stroke, researchers study the impact of various risk factors on the onset of stroke in an individual. Then, they use the analyzed data to predict the probability of stroke occurrence using machine learning algorithms and techniques like neural networks, decision tree, random forest, linear regression, etc. In this research, recent studies that proposed stroke prediction frameworks using data mining approaches have been reviewed, and a new hybrid framework is proposed to predict stroke disease using two main steps, clustering and classification. Enhanced Hierarchical Clustering is applied on the dataset, then five classifiers are evaluated and compared. The used algorithms are Logistic Regression, Random Forest, Support Vector Machine, Neural Network and XGBoost. All of them show good results according to accuracy and AUC. The best result which is (97%) has been achieved by Random Forest classifier

Title : Interpretable Machine Learning Methods for Stroke Prediction

Author: Rebecca Zhang

Year : 2019

Machine learning has long been touted as the next big tool, revolutionizing scientific endeavors as well as impacting industries like retail and finance. Naturally, there is much interest in the potential of next improving healthcare. However, using traditional machine learning approaches in this domain has many difficulties, chief among which is the issue of interpretability. We focus on the medical condition of stroke, a particularly desirable problem to target because it is one of the most prevalent and yet preventable conditions affecting Americans today. In this thesis, we apply novel interpretable prediction techniques to the problem of predicting stroke presence, location, acuity, and mortality risk for patient populations at two different hospital systems. We show that using an interpretable, optimal tree-based approach is roughly as effective if not better than black-box approaches. Using the clinical learnings from these studies, we explore new interpretable methodologies designed with medical applications and their unique challenges in mind. We present a novel regression algorithm to predict outcomes when the population is comprised of notably different subpopulations, and demonstrate that this gives comparable performance with improved interpretability. Finally, we explore new natural language processing techniques for machine learning from text. We propose an alternate end-to-end framework for going from unprocessed textual data to predictions, with an interpretable linguistics-based approach to model words. Altogether, this work demonstrates the promise that new parsimonious, interpretable algorithms have in the domain of stroke and broader healthcare problems.

8. SYSTEM STUDY

8.1 Objectives

The goal is to develop a machine learning model for Stroke Prediction, to potentially replace the updatable supervised machine learning classification models by predicting results in the form of best accuracy by comparing supervised algorithm.

8.2 Project Goals

- Exploration data analysis of variable identification
 - Loading the given dataset
 - Import required libraries packages
 - Analyze the general properties
 - Find duplicate and missing values
 - Checking unique and count values
- Uni-variate data analysis
 - Rename, add data and drop the data
 - To specify data type
- Exploration data analysis of bi-variate and multi-variate
 - Plot diagram of pairplot, heatmap, bar chart and Histogram
- Method of Outlier detection with feature engineering

- Pre-processing the given dataset
 - Splitting the test and training dataset
 - Comparing the Decision tree and Logistic regression model and random forest etc.
- Comparing algorithm to predict the result
 - Based on the best accuracy

8.3 Scope Of The Project

The progress in reducing death around the world has been remarkable. This project helps to predict the stroke risk using prediction model in older people who are addicted to the risk factors as mentioned in the project. In future, the same project can be extended to give the stroke percentage using the output of current project. This project can also be used to find the stroke can also be used to find the stroke probabilities in young people and underage people by collecting respective risk factor informations and doctors consulting.

9. Feasibility study:

Data Wrangling

In this section of the report will load in the data, check for cleanliness, and then trim and clean given dataset for analysis. Make sure that the document steps carefully and justify for cleaning decisions.

Data collection

The data set collected for predicting given data is split into Training set and Test set. Generally, 7:3 ratios are applied to split the Training set and Test set. The Data Model which was created using Random Forest,

logistic, Decision tree algorithms and Support vector classifier (SVC) are applied on the Training set and based on the test result accuracy, Test set prediction is done.

Preprocessing

The data which was collected might contain missing values that may lead to inconsistency. To gain better results data need to be preprocessed so as to improve the efficiency of the algorithm. The outliers have to be removed and also variable conversion need to be done.

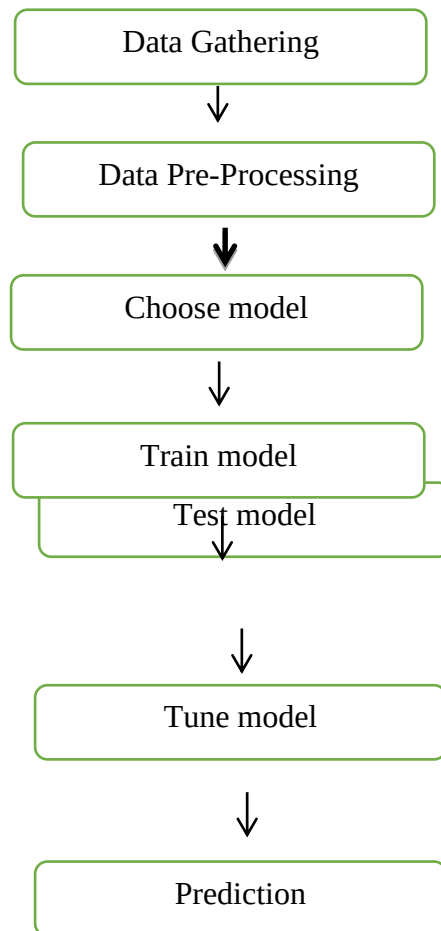
Building the classification model

The prediction of Stroke, A Random Forest Algorithm prediction model is effective because of the following reasons: It provides better results in classification problem.

- It is strong in preprocessing outliers, irrelevant variables, and a mix of continuous, categorical and discrete variables.
- It produces out of bag estimate error which has proven to be unbiased in many tests and it is relatively easy to tune with.

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 preprocess 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.



Process of dataflow diagram

10. List of Modules:

- Data Pre-processing
- Data Analysis of Visualization
- Comparing Algorithm with prediction in the form of best accuracy result
- Deployment Using Flask

11. Project Requirements

General:

Requirements are the basic constraints 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

11.1 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.

11.2 Non-Functional Requirements:

Process of functional steps,

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

12. 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 2 GB

13. SOFTWARE DESCRIPTION

Anaconda is a free and open-source 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. Package versions are managed by the package management system “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 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 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 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.

13.1 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.

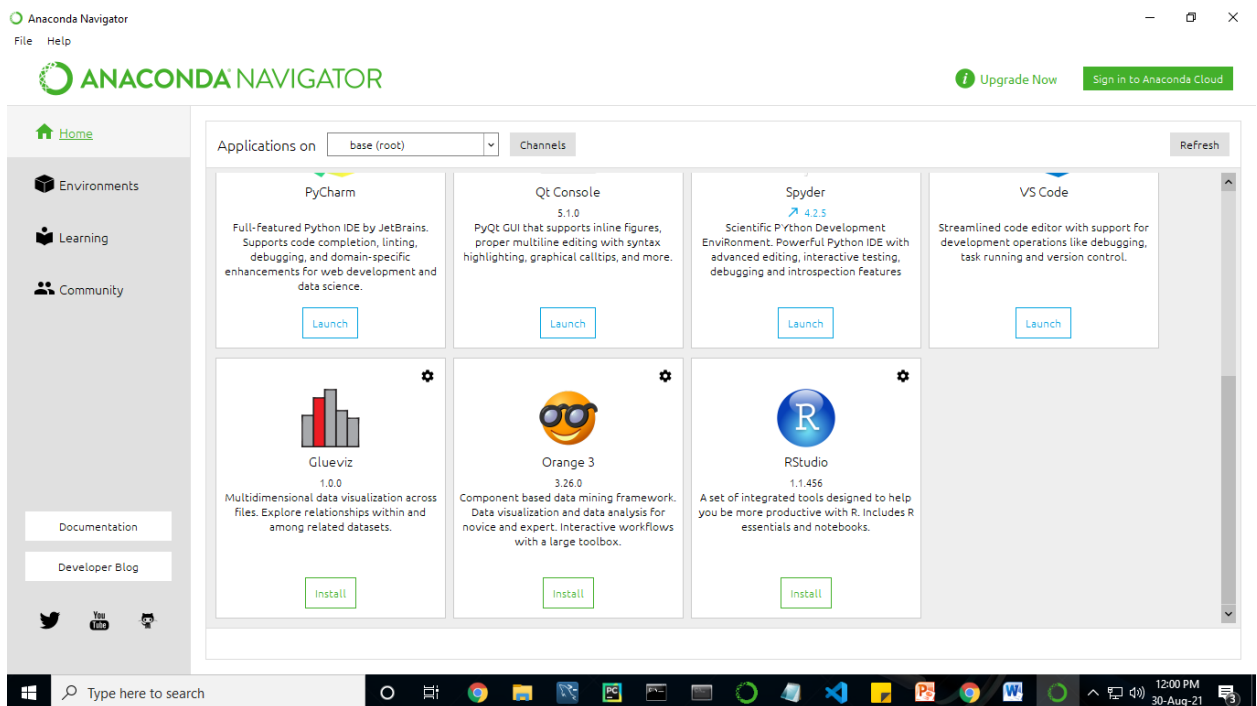
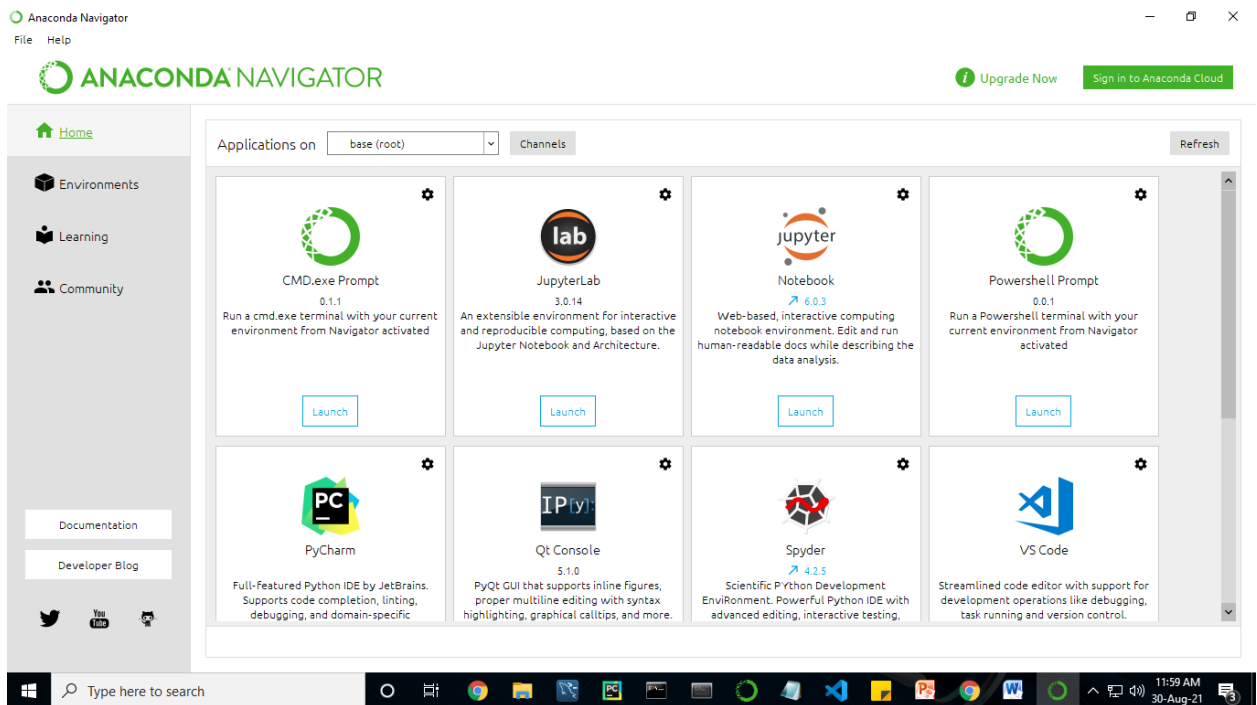
In order to run, many scientific packages depend on specific versions of other packages. Data scientists often use multiple versions of many packages and use multiple environments to separate these different versions.

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
- Jupyter Notebook
- Spyder
- PyCharm
- VSCode
- Glueviz
- Orange 3 App
- RStudio
- 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 system that 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.

13.2 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, 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 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, a sort of control panel that allows (among other things) to select which notebook to open.

When started, the Jupyter 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 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 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 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 (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: To support interactive 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 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), a “control panel” showing local files and allowing to open notebook documents or shutting down their kernels.

kernel: A notebook kernel is a “computational engine” that executes the code contained in a Notebook document. The ipython kernel, referenced in this guide, executes python code. Kernels for many other languages exist (official kernels).

When you open a 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: The Notebook Dashboard is the component which is shown first when you launch Jupyter Notebook App. The Notebook Dashboard is mainly used to open notebook documents, and to manage the running kernels (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.

14. PYTHON

Introduction:

Python is an interpreted high-level general-purpose programming language. Its design philosophy emphasizes code readability with its use of significant indentation. Its language constructs as well as its object-

oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

Guido van Rossum began working on Python in the late 1980s, as a successor to the 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 list comprehensions and a garbage collection system using reference counting. Python 3.0 was released in 2008 and was a major revision of the language that is not completely backward-compatible. 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 at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to ABC programming language, which was inspired by SETL, capable of exception handling and interfacing with the Amoeba 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, 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 garbage collector and support for Unicode.

Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely backward-compatible. Many of its major features were backported 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 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, only Python 3.6.x and later are supported.

Python 3.9.2 and 3.8.8 were expedited as all versions of Python (including 2.7) had security issues, leading to possible remote code execution and web cache poisoning.

Design Philosophy & Feature

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by meta-programming and meta-objects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Python's design offers some support for functional programming in the Lisp tradition. It has filter, map and reduce functions; list comprehensions, dictionaries, sets, and generator expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.

The language's core philosophy is summarized in the document The Zen of Python (PEP 20), which includes aphorisms 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 (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, 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's "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, a Fellow at the 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, and reject patches to non-critical parts of the C-Python 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, a just-in-time compiler. 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 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) instead of the standard foo and bar.

A common 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, that it conforms with 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 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 or Pascal.

Indentation :

Main article: Python syntax and semantics & Indentation

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. 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, 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 include:

- The assignment 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, for use in object-oriented programming.
- The def statement, which defines a function or method.
- The with statement, which encloses a code block within a context manager (for example, acquiring a lock before the block of code is run and releasing the lock afterwards, or opening a file and then closing it), allowing 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. 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 function and yield is also an operator. This form is used to implement co-routines.
- 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 to a separate, dynamically-allocated object. 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 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 and is contrasted with statically-typed programming languages, where each variable may only contain values of a certain type.

Python does not support tail call optimization or first-class continuations, and, according to Guido van Rossum, it never will.^{[80][81]} However, better support for co-routine-like functionality is provided, by extending Python's generators. Before 2.5, generators were lazy iterators; 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 are similar to those found in languages such as C and Java, while some are not:

- Addition, subtraction, and multiplication are the same, but the behavior 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 for 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 its boolean operators rather than the symbolic `&&`, `||`, `!` used in Java and C.
- Python has a type of expression termed a list comprehension as well as a more general expression termed a generator expression.
- Anonymous functions are implemented using `lambda` expressions; 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 and tuples. Lists are written as `[1, 2, 3]`, are mutable, and cannot be used as the keys of dictionaries (dictionary keys must be 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 to `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, 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:

- Strings delimited by single or double quote marks. Unlike in Unix shells, 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. 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 in shells, Perl and Ruby.
- 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 and Windows-style paths. Compare "@-quoting" in C#.
- Python has array index and array slicing expressions on lists, denoted as `a[Key]`, `a[start:stop]` or `a[start:stop:step]`. Indexes are zero-based, 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.

In Python, a distinction between expressions and statements is rigidly enforced, in contrast to languages such as Common Lisp, Scheme, or Ruby. This leads to duplicating some functionality. For example:

- List comprehensions vs. for-loops
- Conditional 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, 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 on objects are functions attached to the object's class; the syntax `instance.method(argument)` is, for normal methods and functions, syntactic sugar for `Class.method(instance, argument)`. Python methods have an explicit `self` parameter access instance data, in contrast to the implicit `self` (or `this`) in some other object-oriented programming languages (e.g., C++, Java, Objective-C, or Ruby). 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;

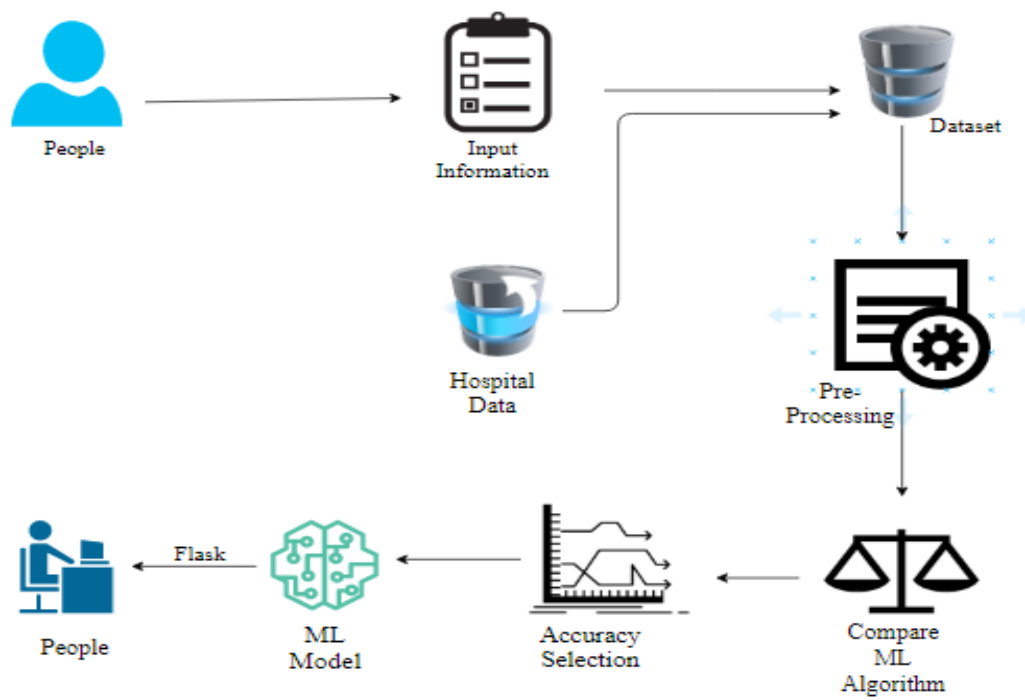
rather, operations on an object may fail, signifying that the given object is not of a suitable type. Despite being dynamically-typed, Python is strongly-typed, forbidding operations that are not well-defined (for example, adding a number to a string) rather than silently attempting to make sense of 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.

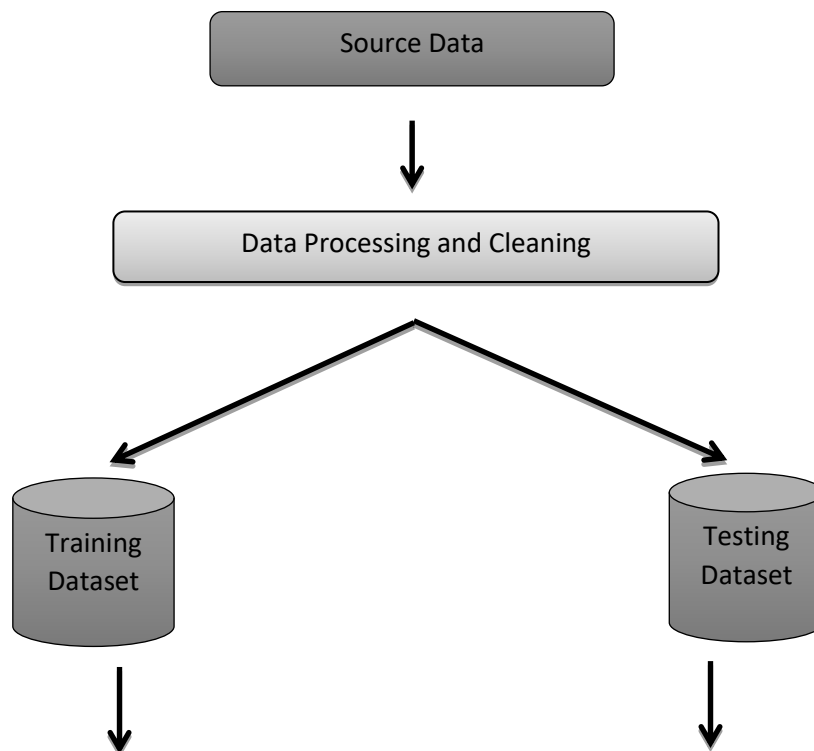
Before version 3.0, Python had two kinds of classes: old-style and new-style. The syntax of both styles is the same, the difference being whether the class object is inherited from, directly or indirectly (all new-style classes inherit from `object` and are instances of `type`). In versions of Python 2 from Python 2.2 onwards, both kinds of classes can be used. Old-style classes were eliminated in Python 3.0.

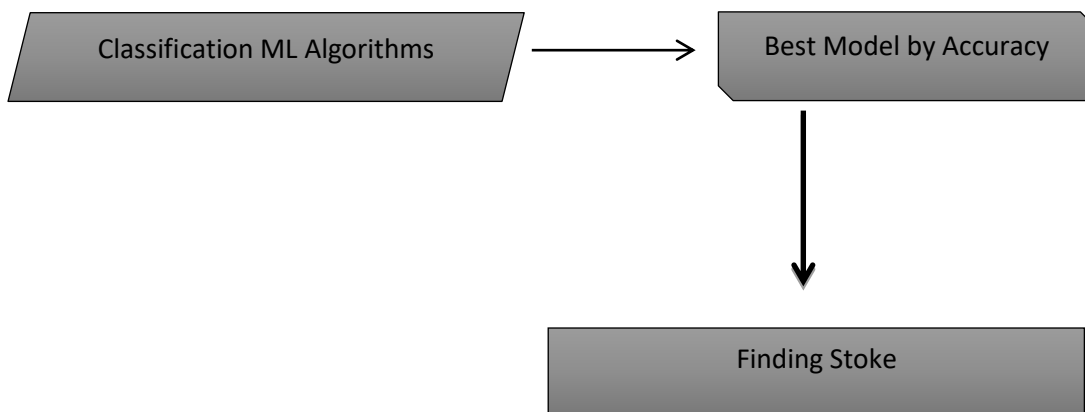
The long-term plan is to support gradual typing and from Python 3.5, the syntax of the language allows specifying static types but they are not checked in the default implementation, CPython. An experimental optional static type checker named `mypy` supports compile-time type checking.

15. System Architecture



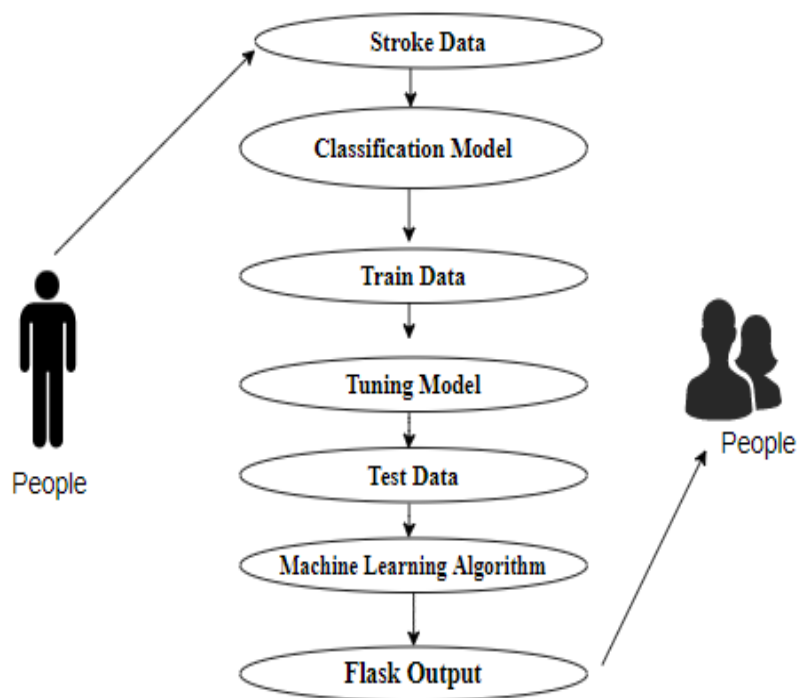
16. Work flow diagram





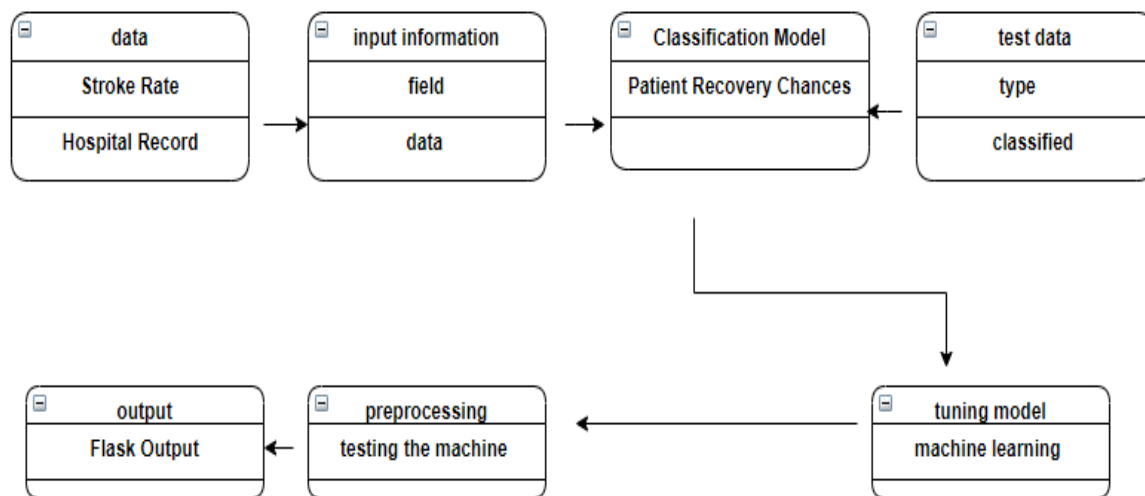
Workflow Diagram

17. Use Case Diagram



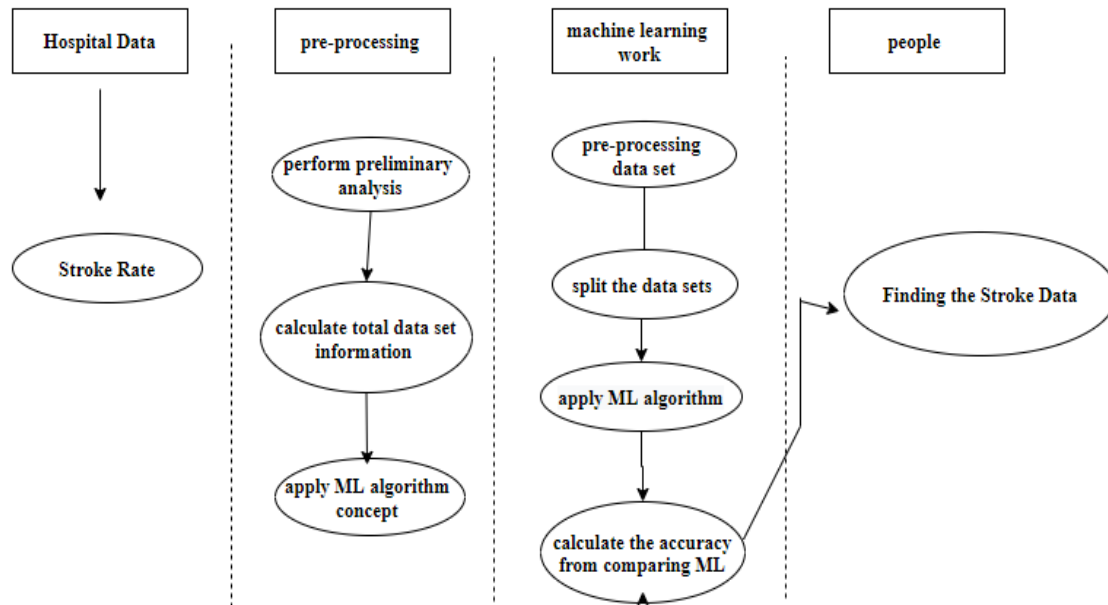
Use case diagrams are considered for high level requirement analysis of a system. So when the requirements of a system are analyzed the functionalities are captured in use cases. So, it can say that uses cases are nothing but the system functionalities written in an organized manner.

18. Class Diagram:



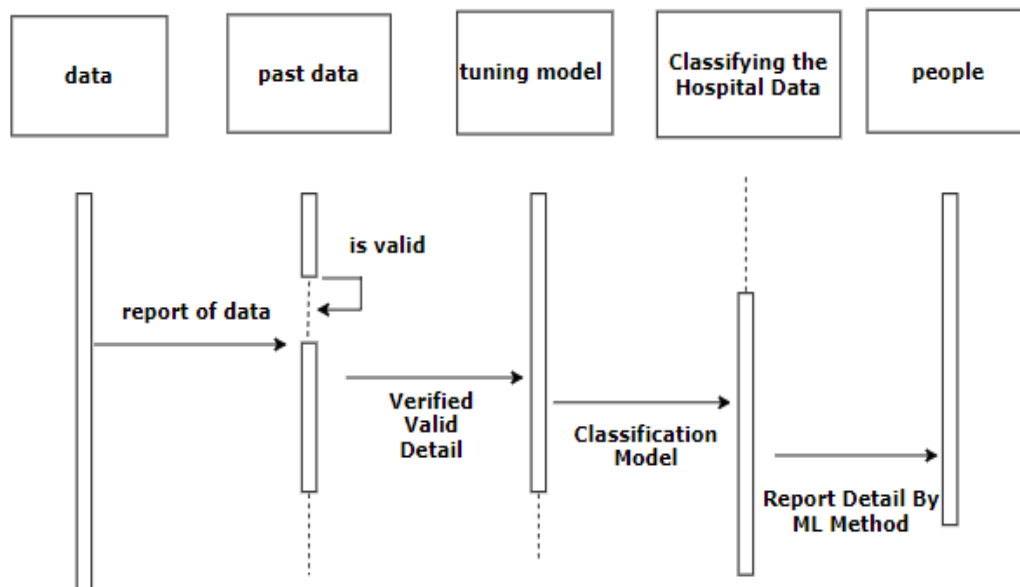
Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. So a collection of class diagrams represent the whole system. The name of the class diagram should be meaningful to describe the aspect of the system. Each element 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.

19. Activity Diagram:



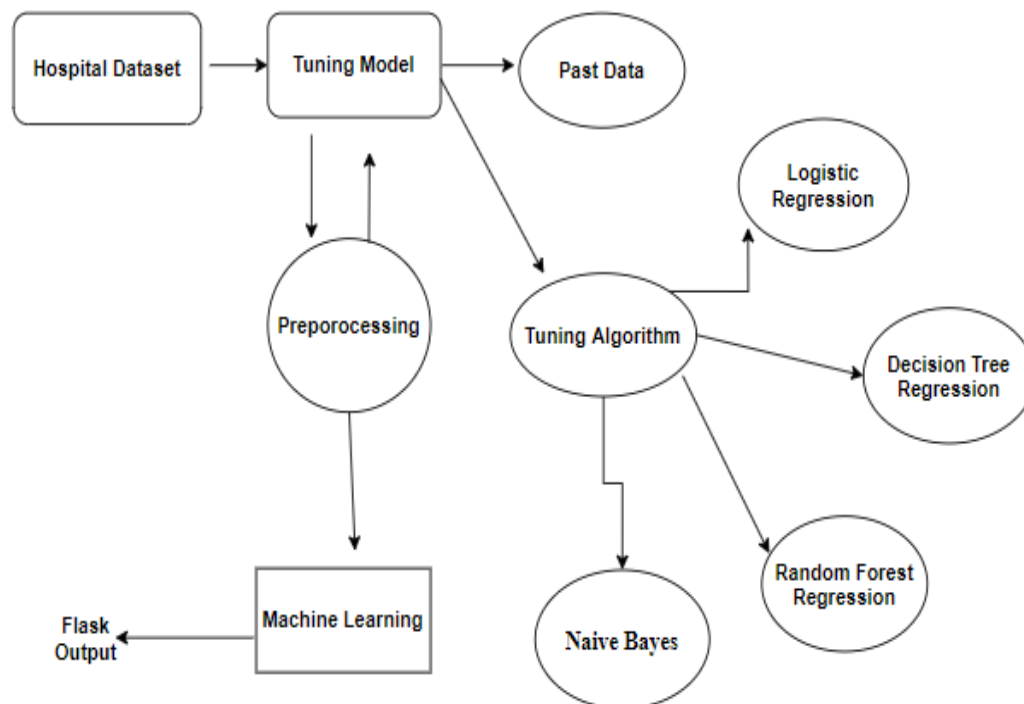
Activity is a particular operation of the system. Activity diagrams are not only used for visualizing dynamic nature of a system but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in activity diagram is the message part. 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.

20. 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 modeling, which focuses on identifying the behavior within your system. Other dynamic modeling techniques include activity diagramming, communication diagramming, timing diagramming, and interaction overview diagramming. Sequence diagrams, along with class diagrams and physical data models are in my opinion the most important design-level models for modern business application development.

21. 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 technique that can help define business processes and be used as the foundation for a 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.

22. 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 and specifically, it focus on probably the biggest data cleaning task, missing values and it able to more quickly clean data. It wants to spend lesstime 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 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. 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

Data Validation/ Cleaning/Preparing Process

Importing the library packages with loading given dataset. To analyzing the variable identification by data shape, data type and evaluating the missing values, duplicate values. A validation dataset is a sample of data held back from training your model that is used to give an estimate of model skill while tuning model's and procedures that you can use to make the best use of validation and test datasets when evaluating your models. Data cleaning / preparing by rename the given dataset and drop the column etc. to analyze the uni-variate, bi-variate and multi-variate process. The steps and techniques for data cleaning will vary from dataset to dataset. The primary goal of data cleaning is to detect and remove errors and anomalies to increase the value of data in analytics and decision making.

The screenshot shows a Jupyter Notebook titled 'M1-stroke' with the following code and output:

```
In [1]: import pandas as p
import numpy as n

In [2]: import warnings
warnings.filterwarnings('ignore')

In [3]: data = p.read_csv('stroke.csv')

In [4]: data.head()
Out[4]:
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1

```
In [5]: data.shape
Out[5]: (5110, 12)

In [6]: data.isnull().sum()
```

The screenshot shows the next steps in the Jupyter Notebook:

```
In [7]: df = data.dropna()

In [8]: df.shape
Out[8]: (4909, 12)

In [9]: df.isnull().sum()
Out[9]:
```

id	0
gender	0
age	0
hypertension	0
heart_disease	0
ever_married	0
work_type	0
Residence_type	0
avg_glucose_level	0
bmi	0
smoking_status	0
stroke	0
dtype:	int64

```
In [10]: df.columns
Out[10]: Index(['id', 'gender', 'age', 'hypertension', 'heart_disease', 'ever_married',
```

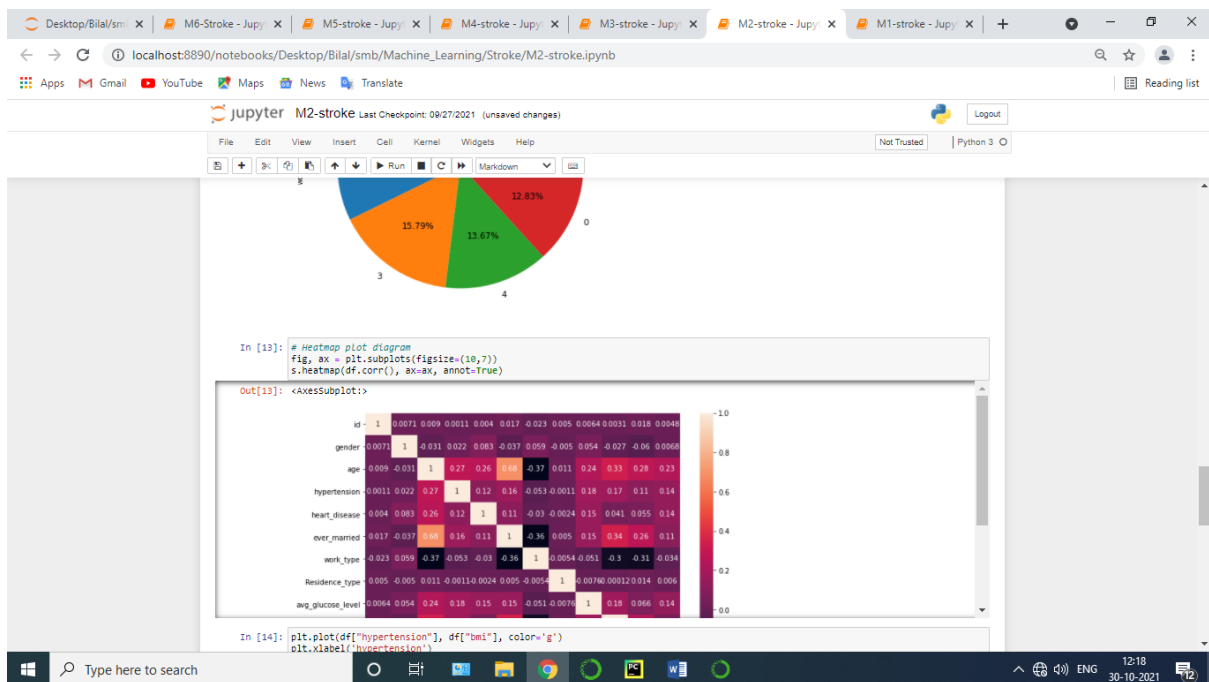
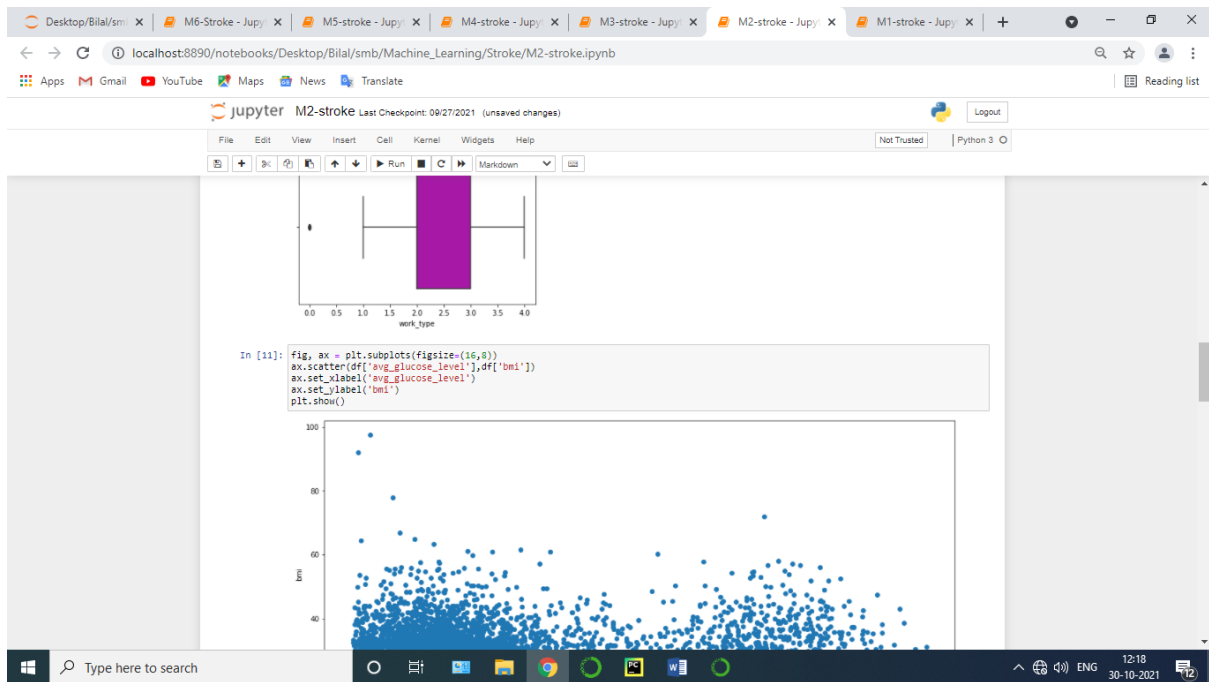
Exploration data analysis of visualization

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.



Pre-processing refers to the transformations applied to our data before feeding it to the algorithm. Data Preprocessing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis. To achieving better results from the applied model in Machine Learning method of the data

has to be in a proper manner. Some specified Machine Learning model needs information in a specified format, for example, Random Forest algorithm does not support null values. Therefore, to execute random forest algorithm null values have to be managed from the original raw data set. And another aspect is that data set should be formatted in such a way that more than one Machine Learning and Deep Learning algorithms are executed in given dataset.

False Positives (FP): A person who will pay predicted as defaulter. When actual class is no and predicted class is yes. E.g. if actual class says this passenger did not survive but predicted class tells you that this passenger will survive.

False Negatives (FN): A person who default predicted as payer. When actual class is yes but predicted class is no. E.g. if actual class value indicates that this passenger survived and predicted class tells you that passenger will die.

True Positives (TP): A person who will not pay predicted as defaulter. These are the correctly predicted positive values which means that the value of actual class is yes and the value of predicted class is also yes. E.g. if actual class value indicates that this passenger survived and predicted class tells you the same thing.

True Negatives (TN): A person who default predicted as payer. These are the correctly predicted negative values which means that the value of actual class is no and value of predicted class is also no. E.g. if actual class says this passenger did not survive and predicted class tells you the same thing.

Comparing Algorithm with prediction in the form of best accuracy result

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.

In the next section you will discover exactly how you can do that in Python with scikit-learn. The key to a fair comparison of machine learning algorithms is ensuring that each algorithm is evaluated in the same way on the same data and it can achieve this by forcing each algorithm to be evaluated on a consistent test harness.

In the example below 4 different algorithms are compared:

➤ Logistic Regression

- Random Forest
- Naive Bayes
- Decision Tree Classification

The K-fold cross validation procedure is used to evaluate each algorithm, importantly configured with the same random seed to ensure that the same splits to the training data are performed and that each algorithm is evaluated in precisely the same way. Before that comparing algorithm, Building a Machine Learning Model using install Scikit-Learn libraries. In this library package have to done preprocessing, linear model with logistic regression method, cross validating by KFold method, ensemble with random forest method and tree with decision tree classifier. Additionally, splitting the train set and test set. To predicting the result by comparing accuracy.

Prediction result by accuracy:

Logistic regression algorithm also uses a linear equation with independent predictors to predict a value. The predicted value can be anywhere between negative infinity to positive infinity. It need the output of the algorithm to be classified variable data. Higher accuracy predicting result is logistic regression model by comparing the best accuracy.

$$\text{True Positive Rate(TPR)} = \text{TP} / (\text{TP} + \text{FN})$$

$$\text{False Positive rate(FPR)} = \text{FP} / (\text{FP} + \text{TN})$$

Accuracy: The Proportion of the total number of predictions that is correct otherwise overall how often the model predicts correctly defaulters and non-defaulters.

Accuracy calculation:

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN)$$

Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. One may think that, if we have high accuracy then our model is best. Yes, accuracy is a great measure but only when you have symmetric datasets where values of false positive and false negatives are almost same.

Precision: The proportion of positive predictions that are actually correct.

$$\text{Precision} = TP / (TP + FP)$$

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. The question that this metric answer is of all passengers that labeled as survived, how many actually survived? High precision relates to the low false positive rate. We have got 0.788 precision which is pretty good.

Recall: The proportion of positive observed values correctly predicted. (The proportion of actual defaulters that the model will correctly predict)

$$\text{Recall} = TP / (TP + FN)$$

Recall(Sensitivity) - Recall is the ratio of correctly predicted positive observations to the all observations in actual class - yes.

F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it's better to look at both Precision and Recall.

General Formula:

$$F\text{- Measure} = \frac{2TP}{2TP + FP + FN}$$

F1-Score Formula:

$$F1 \text{ Score} = \frac{2 * (\text{Recall} * \text{Precision})}{(\text{Recall} + \text{Precision})}$$

ALGORITHM AND TECHNIQUES

Algorithm Explanation

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. In Supervised Learning, algorithms learn from labeled data. After understanding the data, the algorithm determines which label should be given to new data based on pattern and associating the patterns to the unlabeled new data.

Used Python Packages:

sklearn:

- In python, sklearn is a machine learning package which include a lot of ML algorithms.
- Here, we are using some of its modules like `train_test_split`, `DecisionTreeClassifier` or `Logistic Regression` and `accuracy_score`.

NumPy:

- It is a numeric python module which provides fast maths functions for calculations.
- It is used to read data in numpy arrays and for manipulation purpose.

Pandas:

- Used to read and write different files.
- Data manipulation can be done easily with data frames.

Matplotlib:

- Data visualization is a useful way to help with identify the patterns from given dataset.
- Data manipulation can be done easily with data frames.

Logistic Regression

It is a statistical method for analysing a data set in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes). The goal of logistic regression is to find the best fitting model to describe the relationship between the dichotomous characteristic of interest (dependent variable = response or outcome variable) and a set of independent (predictor or explanatory) variables. Logistic regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.).

In other words, the logistic regression model predicts $P(Y=1)$ as a function of X . Logistic regression Assumptions:

- Binary logistic regression requires the dependent variable to be binary.
- For a binary regression, the factor level 1 of the dependent variable should represent the desired outcome.
- Only the meaningful variables should be included.
- The independent variables should be independent of each other. That is, the model should have little.

- The independent variables are linearly related to the log odds.
- Logistic regression requires quite large sample sizes.

The screenshot shows a Jupyter Notebook titled "M3-stroke" with the following code and output:

```
In [1]: # Module 3 : Performance measurements of Logistic regression

In [2]: #import library packages
import pandas as p
import matplotlib.pyplot as plt
import seaborn as s
import numpy as n

In [3]: import warnings
warnings.filterwarnings('ignore')

In [4]: #Load given dataset
data = p.read_csv("stroke.csv")

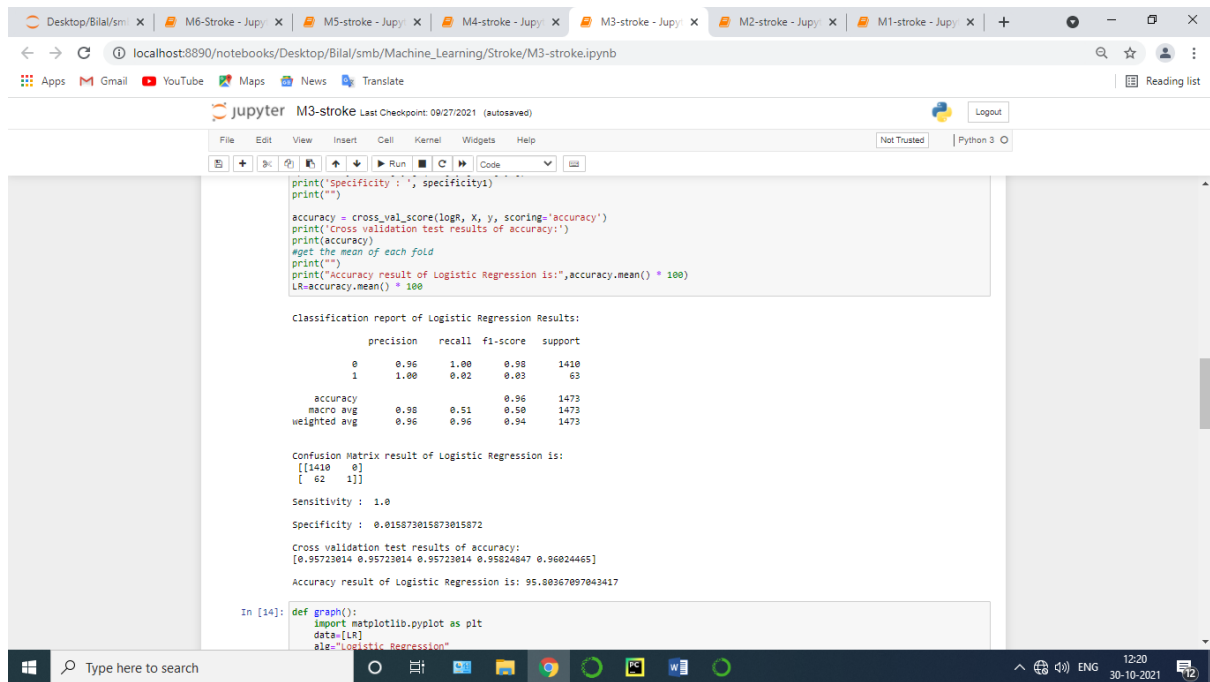
In [5]: data.head()

Out[5]:
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1

```
In [6]: df = data.dropna()
```

The notebook interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), a toolbar with icons for file operations and execution, and a status bar at the bottom showing the search bar, system tray, and date/time (12:20, 30-10-2021).



```
print('Specificity: ', specificity1)
print("")
accuracy = cross_val_score(logR, X, y, scoring='accuracy')
print('Cross validation test results of accuracy:')
print(accuracy)
#Get the mean of each fold
print("")
print("Accuracy result of Logistic Regression is:", accuracy.mean() * 100)
LR=accuracy.mean() * 100

Classification report of Logistic Regression Results:

      precision    recall  f1-score   support

     0       0.96       1.00       0.98       1410
     1       1.00       0.02       0.03         63

 accuracy          0.98          0.51          0.50       1473
 macro avg          0.98          0.51          0.50       1473
 weighted avg          0.96          0.96          0.94       1473

Confusion Matrix result of Logistic Regression is:
[[1410  0]
 [  62  1]]

Sensitivity : 1.0
Specificity : 0.015873015873015872

Cross validation test results of accuracy:
[0.95723814 0.95723814 0.95723814 0.95824847 0.96024465]

Accuracy result of Logistic Regression is: 95.80367097043417

In [14]: def graph():
import matplotlib.pyplot as plt
data=[LR]
plt.plot('Logistic Regression')
```

Random Forest Classifier

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of over fitting to their training set. Random forest is a type of supervised machine learning algorithm based on ensemble learning. Ensemble learning is a type of learning where you join different types of algorithms or same algorithm multiple times to form a more powerful prediction model. The random forest algorithm combines multiple algorithm of the same type i.e. multiple decision *trees*, resulting in a forest of trees, hence the name "Random Forest". The

random forest algorithm can be used for both regression and classification tasks.

The following are the basic steps involved in performing the random forest algorithm:

- Pick N random records from the dataset.
- Build a decision tree based on these N records.
- Choose the number of trees you want in your algorithm and repeat steps 1 and 2.
- In case of a regression problem, for a new record, each tree in the forest predicts a value for Y (output). The final value can be calculated by taking the average of all the values predicted by all the trees in forest. Or, in case of a classification problem, each tree in the forest predicts the category to which the new record belongs. Finally, the new record is assigned to the category that wins the majority vote.

Desktop/Bilal/smb x M6-Stroke - Jupy x M5-stroke - Jupy x M4-stroke - Jupy x M3-stroke - Jupy x M2-stroke - Jupy x M1-stroke - Jupy x +

localhost:8890/notebooks/Desktop/Bilal/smb/Machine_Learning/Stroke/M4-stroke.ipynb

Apps Gmail YouTube Maps News Translate

jupyter M4-stroke Last Checkpoint: 09/27/2021 (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

Module 4 : Performance measurements of Random Forest Algorithm

```
In [1]: #import library packages
import pandas as p
import matplotlib.pyplot as plt
import seaborn as s
import numpy as n

In [2]: import warnings
warnings.filterwarnings('ignore')

In [3]: data=p.read_csv('stroke.csv')

In [4]: data.head()

Out[4]:
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	6040	Male	67.0	0	1	Yes	Private	Urban	228.69	36.0	formerly smoked	1
1	51576	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	80182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1085	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1

```
In [5]: df=data.dropna()

In [6]: del df['id']
del df['age']

In [7]: df.columns

Out[7]: Index(['gender', 'hypertension', 'heart_disease', 'ever_married', 'work_type',
```

Type here to search

ENG 12:20 30-10-2021

Desktop/Bilal/smb x M6-Stroke - Jupy x M5-stroke - Jupy x M4-stroke - Jupy x M3-stroke - Jupy x M2-stroke - Jupy x M1-stroke - Jupy x +

localhost:8890/notebooks/Desktop/Bilal/smb/Machine_Learning/Stroke/M4-stroke.ipynb

Apps Gmail YouTube Maps News Translate

jupyter M4-stroke Last Checkpoint: 09/27/2021 (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```
5102 Female 0 0 Yes Private Rural 77.93 21.7 never smoked 0
5103 Female 0 0 No Private Urban 82.85 46.9 Unknown 0
5104 Female 0 0 No children Rural 103.08 18.6 Unknown 0
5106 Female 0 0 Yes Self-employed Urban 125.20 40.0 never smoked 0
5107 Female 0 0 Yes Self-employed Rural 82.99 30.6 never smoked 0
5108 Male 0 0 Yes Private Rural 198.29 25.8 formerly smoked 0
5109 Female 0 0 Yes Govt_Job Urban 85.28 29.2 Unknown 0
```

```
In [9]: from sklearn.preprocessing import LabelEncoder
var_mod = ['gender','ever_married','work_type','Residence_type','smoking_status']
le = LabelEncoder()
for i in var_mod:
    df[i] = le.fit_transform(df[i]).astype(int)

In [10]: #preprocessing, split test and dataset, split response variable
X = df.drop(labels='stroke', axis=1)
#Response variable
y = df.loc[:, 'stroke']

In [11]: '''We'll use a test size of 30%. We also stratify the split on the response variable,
which is very important to do because there are so few fraudulent transactions'''

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=1, stratify=y)
print("Number of training dataset: ", len(X_train))
print("Number of test dataset: ", len(X_test))
print("Total number of dataset: ", len(X_train)+len(X_test))

Number of training dataset: 3436
Number of test dataset: 1473
Total number of dataset: 4909

In [12]: #According to the cross-validated MCC scores, the random forest is the best-performing model, so now let's evaluate its performance
from sklearn.metrics import confusion_matrix, classification_report, matthews_corrcoef, cohen_kappa_score, accuracy_score, average
```

Type here to search

ENG 12:21 30-10-2021

Naive Bayes algorithm:

- The Naive Bayes algorithm is an intuitive method that uses the probabilities of each attribute belonging to each class to make a prediction. It is the supervised learning approach you would come up with if you wanted to model a predictive modeling problem probabilistically.
- Naive bayes simplifies the calculation of probabilities by assuming that the probability of each attribute belonging to a given class value is independent of all other attributes. This is a strong assumption but results in a fast and effective method.
- The probability of a class value given a value of an attribute is called the conditional probability. By multiplying the conditional probabilities together for each attribute for a given class value, we have a probability of a data instance belonging to that class. To make a prediction we can calculate probabilities of the instance belonging to each class and select the class value with the highest probability.
- Naive Bayes is a statistical classification technique based on Bayes Theorem. It is one of the simplest supervised learning algorithms. Naive Bayes classifier is the fast, accurate and reliable algorithm. Naive Bayes classifiers have high accuracy and speed on large datasets.
- Naive Bayes classifier assumes that the effect of a particular feature in a class is independent of other features. For example, a loan applicant is desirable or not depending on his/her income, previous loan and transaction history, age, and location.
- Even if these features are interdependent, these features are still considered independently. This assumption simplifies computation,

and that's why it is considered as naive. This assumption is called class conditional independence.

```
Desktop/Bilal/smb/ Machine_Learning/Stroke/M5-stroke.ipynb
jupyter M5-stroke Last Checkpoint: 09/27/2021 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
In [1]: # Module 5 : Performance measurements of Naive Bayes
In [2]: #import library packages
import pandas as p
import matplotlib.pyplot as plt
import seaborn as s
import numpy as n

In [3]: import warnings
warnings.filterwarnings('ignore')

In [4]: data=p.read_csv('stroke.csv')

In [5]: data.head()
Out[5]:
   id  gender  age  hypertension  heart_disease  ever_married  work_type  Residence_type  avg_glucose_level  bmi  smoking_status  stroke
0  6040   Male  67.0           0             1           Yes   Private      Urban          228.69  36.6  formerly smoked    1
1  51576  Female  61.0           0             0           Yes  Self-employed   Rural          202.21  NaN  never smoked    1
2  31112   Male  80.0           0             1           Yes   Private      Rural          105.92  32.5  never smoked    1
3  80182  Female  49.0           0             0           Yes   Private      Urban          171.23  34.4  smokes    1
4  1955   Female  70.0           1             0           Yes  Self-employed   Rural          174.12  24.0  never smoked    1

In [6]: df=data.dropna()

In [7]: del df['id']
del df['age']

In [8]: df.columns
Out[8]: Index(['gender', 'hypertension', 'heart_disease', 'ever_married', 'work_type',
              'Residence_type', 'avg_glucose_level', 'bmi', 'smoking_status'],
              dtype='object', name='Index')
```

```
Desktop/Bilal/smb/ Machine_Learning/Stroke/M5-stroke.ipynb
jupyter M5-stroke Last Checkpoint: 09/27/2021 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
In [14]:
0      0.97      0.89      0.93     1410
1      0.11      0.29      0.16         63

accuracy      0.84      0.59      0.54     1473
macro avg     0.93      0.57      0.89     1473
weighted avg

Confusion Matrix result of Naive Bayes is:
[[1259 151]
 [ 45 18]]

Sensitivity : 0.8929870814184397
Specificity : 0.2857142857142857

Cross validation test results of accuracy:
[0.8808554 0.87270876 0.87270876 0.86863544 0.88990626]

Accuracy result of Naive Bayes Algorithm is: 87.69633214372467

In [14]: def graph():
import matplotlib.pyplot as plt
data=[0]
alg="Naive Bayes"
plt.figure(figsize=(5,5))
b=plt.bar(alg,data,color="g")
plt.title("Accuracy comparison of Stroke",fontsize=15)
plt.legend(b,data,fontsize=9)

In [15]: graph()

Accuracy comparison of Stroke
80
60
Accuracy: 87.69633214372467
```

Decision Tree Classification:

It is one of the most powerful and popular algorithm. Decision-tree algorithm falls under the category of supervised learning algorithms. It works for both continuous as well as categorical output variables. Assumptions of Decision tree:

- At the beginning, we consider the whole training set as the root.
- Attributes are assumed to be categorical for information gain, attributes are assumed to be continuous.
- On the basis of attribute values records are distributed recursively.
- We use statistical methods for ordering attributes as root or internal node.

Decision tree builds classification or regression models in the form of a tree structure. It breaks down a data set into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. A decision node has two or more branches and a leaf node represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data. Decision tree builds classification or regression models in the form of a tree structure. It utilizes an if-then rule set which is mutually exclusive and exhaustive for classification. The rules are learned sequentially using the training data one at a time. Each time a rule is learned, the tuples covered by the rules are removed.

This process is continued on the training set until meeting a termination condition. It is constructed in a top-down recursive divide-and-conquer manner. All the attributes should be categorical. Otherwise, they should be discretized in advance. Attributes in the top of the tree have more impact towards in the classification and they are identified using the information gain concept. A decision tree can be easily over-fitted generating too many branches and may reflect anomalies due to noise or outliers.

The screenshot shows a Jupyter Notebook interface with the following code and output:

```

In [1]: import pandas as p
import numpy as n
import seaborn as s
import matplotlib.pyplot as plt

In [2]: import warnings
warnings.filterwarnings('ignore')

In [3]: data=p.read_csv('stroke.csv')

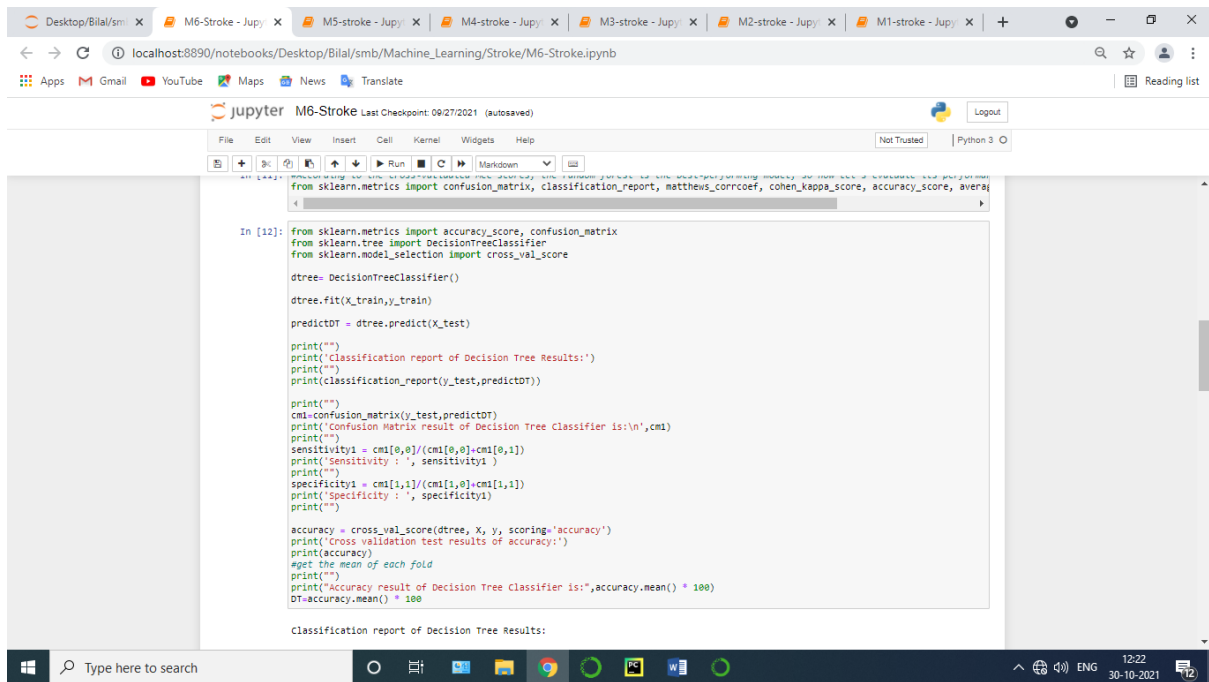
In [4]: data.head()
Out[4]:
  id  gender  age  hypertension  heart_disease  ever_married  work_type  Residence_type  avg_glucose_level  bmi  smoking_status  stroke
0  9040   Male  67.0           0             1           Yes   Private      Urban      228.99   36.6  formerly smoked    1
1  51676  Female  61.0           0             0           Yes  Self-employed   Rural      202.21   NaN   never smoked    1
2  31112   Male  80.0           0             1           Yes   Private      Rural      105.92   32.5  never smoked    1
3  90182  Female  49.0           0             0           Yes   Private      Urban      171.23   34.4     smokes    1
4  1065  Female  76.0           1             0           Yes  Self-employed   Rural      174.12   24.0  never smoked    1

In [5]: df=data.dropna()

In [6]: del df['id']
del df['age']

In [7]: df.columns
Out[7]: Index(['gender', 'hypertension', 'heart_disease', 'ever_married', 'work_type',
              'Residence_type', 'avg_glucose_level', 'bmi', 'smoking_status',
              'stroke'],
              dtype='object')

```



23. Deployment

Flask (Web Framework) :

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

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

Flask was created by Armin Ronacher of Pocoo, an international group of Python enthusiasts formed in 2004. According to Ronacher, the idea was originally an April Fool's joke that was popular enough to make into a serious application. The name is a play on the earlier Bottle framework.

When Ronacher and Georg Brand created a bulletin board system written in Python, the Pocoo projects Werkzeug and Jinja were developed.

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

Flask has become popular among Python enthusiasts. As of October 2020, it has second most stars on 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 Flask is part of the Pallets Projects, and based on several others of them.

Flask is based on Werkzeug, Jinja2 and inspired by Sinatra Ruby framework, available under BSD licence. It was developed at pocoo by Armin Ronacher. Although Flask is rather young compared to most Python frameworks, it holds a great promise and has already gained popularity among Python web developers. Let's take a closer look into Flask, so-called "micro" framework for Python.

FEATURES:

Flask was designed to be **easy to use and extend**. The idea behind Flask 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. Flask is great for all kinds of projects. It's especially good for prototyping. Flask depends on two external libraries: the Jinja2 template engine and the Werkzeug WSGI toolkit.

Still the question remains why use Flask as your web application framework if we have immensely powerful Django, Pyramid, and don't forget web mega-framework Turbo-gears? Those are supreme Python web frameworks BUT out-of-the-box Flask is pretty impressive too with its:

- Built-In Development server and Fast debugger
- integrated support for unit testing
- RESTful request dispatching
- Uses Jinja2 Templating
- support for secure cookies
- Unicode based
- Extensive Documentation
- Google App Engine Compatibility
- Extensions available to enhance features desired

Plus Flask 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.

- Flask 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.
- Basic foundation API is nicely shaped and coherent.
- Flask documentation is comprehensive, full of examples and well structured. You can even try out some sample application to really get a feel of Flask.
- It is super easy to deploy Flask in production (Flask 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, Flask is one of the most polished and feature-rich micro frameworks, available. Still young, Flask has a thriving community, first-class extensions, and an elegant API. Flask 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 Flask.

Flask is an API of Python that allows us to build up web-applications. It was developed by Armin Ronacher. Flask'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

Flask 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 Flask 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.

Flask 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.

Flask 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. Flask is a good start if you are getting into web development. Flask is a simple, un-opinionated framework; it doesn't decide what your application should look like developers do.

Flask is a web framework. This means flask 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, a wiki or go as big as a web-based calendar application or a commercial website.

Advantages of Flask:

- 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 Flask is a web framework from Python language. Flask 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 flask still doesn't use the Model View Controller (MVC) method.

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

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

Flask 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 flask 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()`.

Usually you create a Flask instance in your main module or in the `__init__.py` file of your package.

Parameters

- **rule** (str) – The URL rule string.
 - **endpoint** (Optional[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 object.
- Return type -- 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()` for that.

Parameters:

`f (Callable[[Response], Response])`

Return type

`Callable[[Response], 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()` 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`. Use as a `with` block to push the context, which will make `current_app` point at this application.

An application context is automatically pushed by `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()`

24. 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 metadata for the page goes — stuff mostly meant for search engines and other computer programs.

`<body>` — This is where the 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 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.

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>` 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>Google</a>`

Image Tag

In today's modern digital world, images are everything. The `<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

```

```

25. CSS

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

26. Coding:

Module 1: Data validation and pre-processing technique

```
import pandas as p
import numpy as n
```

```
import warnings
warnings.filterwarnings('ignore')
```

```
data=p.read_csv('stroke.csv')
```

```
data.head()
```

```
data.shape
```

```
data.isnull().sum()
```

Drop the Given Dataset

```
df=data.dropna()
```

```
df.shape
```

```
df.isnull().sum()
```

```
df.columns
```

```
df.describe()
```

```
df.columns
```

```
del df['id']  
del df['age']
```

```
df.columns
```

```
df.gender.unique()
```

```
df.hypertension.unique()
```

```
df.heart_disease.unique()
```

```
df.ever_married.unique()
```

```
df.work_type.unique()
```

```
df.Residence_type.unique()
```

```
df.avg_glucose_level.unique()
```

```
df.bmi.unique()
```

```
df.smoking_status.unique()
```

```
df.stroke.unique()
```

```
df.columns
```

```
p.crosstab(df.avg_glucose_level,df.smoking_status)
```

```
p.Categorical(df['gender']).describe()
```

```
p.Categorical(df['hypertension']).describe()
```

```
p.Categorical(df['heart_disease']).describe()
```

```
p.Categorical(df['heart_disease']).describe()
```

```
p.Categorical(df['work_type']).describe()
```

```
p.Categorical(df['Residence_type']).describe()
```

```
p.crosstab(df.hypertension,df.heart_disease)
```

```
p.Categorical(df['smoking_status']).describe()
```

```
p.Categorical(df['avg_glucose_level']).describe()
```

```
p.Categorical(df['bmi']).describe()
```

```
print("Smoking Status: ",sorted(df['smoking_status'].unique()))
```

```
print("Average Glucose Level: ",sorted(df['avg_glucose_level'].unique()))
```

```
print("BMI: ",sorted(df['bmi'].unique()))
```

```
df.info()
```

```
df.duplicated()
```

```
sum(df.duplicated())
```

```
df.columns
```

```
print("Minimum value of Glucose Level is:",df.avg_glucose_level.min())
print("Maximum value of Glucose Level is:",df.avg_glucose_level.max())
```

```
print("Minimum value of bmi Level is:",df.bmi.min())
print("Maximum value of bmi Level is:",df.bmi.max())
```

```
fromsklearn.preprocessingimportLabelEncoder
var_mod=['gender','ever_married','work_type','Residence_type','smoking_status']
le=LabelEncoder()
foriinvar_mod:
df[i]=le.fit_transform(df[i]).astype(int)

df.head()
```

Module 2: Exploration data analysis of visualization and training a model by given attributes

```
#import library packages
importpandasasp
importmatplotlib.pyplotasplt
importseabornass
importnumpyasn

data=p.read_csv("stroke.csv")

importwarnings
warnings.filterwarnings('ignore')

df=data.dropna()

df.columns

fromsklearn.preprocessingimportLabelEncoder
```

```

var_mod=['gender','ever_married','work_type','Residence_type','smoking
_status']
le=LabelEncoder()
for i in var_mod:
    df[i]=le.fit_transform(df[i]).astype(int)

df.head()

```

```

df['avg_glucose_level'].hist(figsize=(5,5),color='brown',alpha=1)
plt.xlabel('avg_glucose_level')
plt.ylabel('bmi')
plt.title('avg_glucose_level & bmi')

```

```

plt.boxplot(df['smoking_status'])
plt.show()

```

```

import seaborn as s
s.boxplot(df['work_type'],color='m')

```

```

fig,ax=plt.subplots(figsize=(16,8))
ax.scatter(df['avg_glucose_level'],df['bmi'])
ax.set_xlabel('avg_glucose_level')
ax.set_ylabel('bmi')
plt.show()

```

```

#Propagation by variable
def PropByVar(df,variable):
    dataframe_pie=df[variable].value_counts()
    ax=dataframe_pie.plot.pie(figsize=(8,8),autopct='%1.2f%%',fontsize=12)
    ax.set_title(variable+' \n',fontsize=15)
    return nn.round(dataframe_pie/df.shape[0]*100,2)

```

```

PropByVar(df,'work_type')

```

```

# Heatmap plot diagram
fig,ax=plt.subplots(figsize=(10,7))
s.heatmap(df.corr(),ax=ax,annot=True)

```

```

plt.plot(df["hypertension"],df["bmi"],color='g')

```

```
plt.xlabel('hypertension')
plt.ylabel('bmi')
plt.title('Stroke')
plt.show()
```

```
df.head()
```

Splitting Train / Test

```
#preprocessing, split test and dataset, split response variable
X=df.drop(labels='stroke',axis=1)
#Response variable
y=df.loc[:, 'stroke']
```

#We'll use a test size of 200%. We also stratify the split on the response variable, which is very important to do because there are so few fraudulent transactions.

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20,random
_state=1,stratify=y)
print("Number of training dataset: ",len(X_train))
print("Number of test dataset: ",len(X_test))
print("Total number of dataset: ",len(X_train)+len(X_test))
```

Module 3 : Performance measurements of Logistic regression

```
#import library packages
import pandas as p
import matplotlib.pyplot as plt
import seaborn as s
import numpy as n

import warnings
warnings.filterwarnings('ignore')

#Load given dataset
data = p.read_csv("stroke.csv")
```

```
data.head()
```

```
df = data.dropna()
```

```
del df['id']  
del df['age']
```

```
df.columns
```

```
from sklearn.preprocessing import LabelEncoder  
var_mod = ['gender','ever_married', 'work_type', 'Residence_type',  
'smoking_status']  
le = LabelEncoder()  
for i in var_mod:  
    df[i] = le.fit_transform(df[i]).astype(int)
```

```
#preprocessing, split test and dataset, split response variable  
X = df.drop(labels='stroke', axis=1)  
#Response variable  
y = df.loc[:, 'stroke']
```

```
"""We'll use a test size of 30%. We also stratify the split on the response  
variable,  
which is very important to do because there are so few fraudulent  
transactions"""
```

```
from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,  
random_state=1, stratify=y)  
print("Number of training dataset: ", len(X_train))  
print("Number of test dataset: ", len(X_test))  
print("Total number of dataset: ", len(X_train)+len(X_test))
```

```
#According to the cross-validated MCC scores, the random forest is the  
best-performing model, so now let's evaluate its performance on the test  
set.
```

```
from sklearn.metrics import confusion_matrix, classification_report,  
matthews_corrcoef, cohen_kappa_score, accuracy_score,  
average_precision_score, roc_auc_score
```

```

from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

logR= LogisticRegression()

logR.fit(X_train,y_train)

predictLR = logR.predict(X_test)

print("")
print('Classification report of Logistic Regression Results:')
print("")
print(classification_report(y_test,predictLR))

print("")
cm1=confusion_matrix(y_test,predictLR)
print('Confusion Matrix result of Logistic Regression is:\n',cm1)
print("")
sensitivity1 = cm1[0,0]/(cm1[0,0]+cm1[0,1])
print('Sensitivity : ', sensitivity1 )
print("")
specificity1 = cm1[1,1]/(cm1[1,0]+cm1[1,1])
print('Specificity : ', specificity1)
print("")

accuracy = cross_val_score(logR, X, y, scoring='accuracy')
print('Cross validation test results of accuracy:')
print(accuracy)
#get the mean of each fold
print("")
print("Accuracy result of Logistic Regression is:",accuracy.mean() * 100)
LR=accuracy.mean() * 100

def graph():
import matplotlib.pyplot as plt
    data=[LR]
    alg="Logistic Regression"
    plt.figure(figsize=(5,5))
    b=plt.bar(alg,data,color=("b"))
    plt.title("Accuracy comparison of Stroke",fontsize=15)
    plt.legend(b,data,fontsize=9)

```

```
graph()
```

```
TP = cm1[0][0]
FP = cm1[1][0]
FN = cm1[1][1]
TN = cm1[0][1]
print("True Positive :",TP)
print("True Negative :",TN)
print("False Positive :",FP)
print("False Negative :",FN)
print("")
TPR = TP/(TP+FN)
TNR = TN/(TN+FP)
FPR = FP/(FP+TN)
FNR = FN/(TP+FN)
print("True Positive Rate :",TPR)
print("True Negative Rate :",TNR)
print("False Positive Rate :",FPR)
print("False Negative Rate :",FNR)
print("")
PPV = TP/(TP+FP)
NPV = TN/(TN+FN)
print("Positive Predictive Value :",PPV)
print("Negative predictive value :",NPV)
```

```
def plot_confusion_matrix(cm1, title='Confusion matrix-
Logistic_Regression', cmap=plt.cm.Blues):
    target_names=['Predict','Actual']
    plt.imshow(cm1, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(target_names))
    plt.xticks(tick_marks, target_names, rotation=45)
    plt.yticks(tick_marks, target_names)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

```
cm1=confusion_matrix(y_test, predictLR)
print('Confusion matrix-Logistic_Regression:')
print(cm1)
```

```
plot_confusion_matrix(cm1)
```

Module 4 : Performance measurements of Random Forest Algorithm

```
#import library packages
```

```
import pandas as p
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as s
```

```
import numpy as n
```

```
import warnings
```

```
warnings.filterwarnings('ignore')
```

```
data=p.read_csv('stroke.csv')
```

```
data.head()
```

```
df=data.dropna()
```

```
del df['id']
```

```
del df['age']
```

```
df.columns
```

```
df.tail(10)
```

```
from sklearn.preprocessing import LabelEncoder
```

```
var_mod=['gender','ever_married','work_type','Residence_type','smoking  
_status']
```

```
le=LabelEncoder()
```

```
for i in var_mod:
```

```
df[i]=le.fit_transform(df[i]).astype(int)
```

```
#preprocessing, split test and dataset, split response variable
```

```
X=df.drop(labels='stroke',axis=1)
#Response variable
y=df.loc[:,'stroke']
```

'''We'll use a test size of 30%. We also stratify the split on the response variable,
which is very important to do because there are so few fraudulent transactions'''

```
fromsklearn.model_selectionimporttrain_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.30,random
_state=1,stratify=y)
print("Number of training dataset: ",len(X_train))
print("Number of test dataset: ",len(X_test))
print("Total number of dataset: ",len(X_train)+len(X_test))
```

#According to the cross-validated MCC scores, the random forest is the best-performing model, so now let's evaluate its performance on the test set.

```
fromsklearn.metricsimportconfusion_matrix,classification_report,matthe
ws_corrcoef,cohen_kappa_score,accuracy_score,average_precision_sc
ore,roc_auc_score
```

```
fromsklearn.metricsimportaccuracy_score,confusion_matrix
fromsklearn.ensembleimportRandomForestClassifier
fromsklearn.model_selectionimportcross_val_score
```

```
rfc=RandomForestClassifier()
```

```
rfc.fit(X_train,y_train)
```

```
predictRF=rfc.predict(X_test)
```

```
print("")
print('Classification report of Random Forest Results:')
print("")
print(classification_report(y_test,predictRF))
```

```
print("")
cm1=confusion_matrix(y_test,predictRF)
print('Confusion Matrix result of Random Forest Classifier is:\n',cm1)
```

```

print("")
sensitivity1=cm1[0,0]/(cm1[0,0]+cm1[0,1])
print('Sensitivity : ',sensitivity1)
print("")
specificity1=cm1[1,1]/(cm1[1,0]+cm1[1,1])
print('Specificity : ',specificity1)
print("")

accuracy=cross_val_score(rfc,X,y,scoring='accuracy')
print('Cross validation test results of accuracy:')
print(accuracy)
#get the mean of each fold
print("")
print("Accuracy result of Random Forest Classifier
is:",accuracy.mean()*100)
RF=accuracy.mean()*100

```

```

def graph():
import matplotlib.pyplot as plt
data=[RF]
alg="Random Forest Classifier"
plt.figure(figsize=(5,5))
b=plt.bar(alg,data,color=("red"))
plt.title("Accuracy comparison of Stroke",fontsize=15)
plt.legend(b,data,fontsize=9)

```

```
graph()
```

```

TP=cm1[0][0]
FP=cm1[1][0]
FN=cm1[1][1]
TN=cm1[0][1]
print("True Positive :",TP)
print("True Negative :",TN)
print("False Positive :",FP)
print("False Negative :",FN)
print("")
TPR=TP/(TP+FN)
TNR=TN/(TN+FP)
FPR=FP/(FP+TN)
FNR=FN/(TP+FN)
print("True Positive Rate :",TPR)

```

```

print("True Negative Rate :",TNR)
print("False Positive Rate :",FPR)
print("False Negative Rate :",FNR)
print("")
PPV=TP/(TP+FP)
NPV=TN/(TN+FN)
print("Positive Predictive Value :",PPV)
print("Negative predictive value :",NPV)

def plot_confusion_matrix(cm1,title='Confusion matrix-
RandomForestClassifier',cmap=plt.cm.Blues):
    target_names=['Predict','Actual']
    plt.imshow(cm1,interpolation='nearest',cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks=n.arange(len(target_names))
    plt.xticks(tick_marks,target_names,rotation=45)
    plt.yticks(tick_marks,target_names)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

cm1=confusion_matrix(y_test,predictRF)
print('Confusion matrix-RandomForestClassifier:')
print(cm1)
plot_confusion_matrix(cm1)

```

Creating pkl File

```

import joblib
joblib.dump(rfc,'rf.pkl')

```

Module 5 : Performance measurements of Naive Bayes

```

#import library packages
import pandas as p
import matplotlib.pyplot as plt
import seaborn as s

```

```

import numpy as n

import warnings
warnings.filterwarnings('ignore')

data=p.read_csv('stroke.csv')

data.head()

df=data.dropna()

del df['id']
del df['age']

df.columns

from sklearn.preprocessing import LabelEncoder
var_mod = ['gender','ever_married', 'work_type', 'Residence_type',
'smoking_status']
le = LabelEncoder()
for i in var_mod:
    df[i] = le.fit_transform(df[i]).astype(int)

#preprocessing, split test and dataset, split response variable
X = df.drop(labels='stroke', axis=1)
#Response variable
y = df.loc[:, 'stroke']

'''We'll use a test size of 30%. We also stratify the split on the response
variable,
which is very important to do because there are so few fraudulent
transactions'''

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
random_state=1, stratify=y)
print("Number of training dataset: ", len(X_train))
print("Number of test dataset: ", len(X_test))
print("Total number of dataset: ", len(X_train)+len(X_test))

```

#According to the cross-validated MCC scores, the random forest is the best-performing model, so now let's evaluate its performance on the test set.

```
from sklearn.metrics import confusion_matrix, classification_report,  
matthews_corrcoef, cohen_kappa_score, accuracy_score,  
average_precision_score, roc_auc_score
```

```
from sklearn.metrics import accuracy_score, confusion_matrix  
from sklearn.naive_bayes import GaussianNB  
from sklearn.model_selection import cross_val_score
```

```
gnb = GaussianNB()
```

```
gnb.fit(X_train,y_train)
```

```
predictNB = gnb.predict(X_test)
```

```
print("")  
print('Classification report of Naive Bayes Results:')  
print("")  
print(classification_report(y_test,predictNB))
```

```
print("")  
cm1=confusion_matrix(y_test,predictNB)  
print('Confusion Matrix result of Naive Bayes is:\n',cm1)  
print("")  
sensitivity1 = cm1[0,0]/(cm1[0,0]+cm1[0,1])  
print('Sensitivity : ', sensitivity1 )  
print("")  
specificity1 = cm1[1,1]/(cm1[1,0]+cm1[1,1])  
print('Specificity : ', specificity1)  
print("")
```

```
accuracy = cross_val_score(gnb, X, y, scoring='accuracy')  
print('Cross validation test results of accuracy:')  
print(accuracy)  
#get the mean of each fold  
print("")  
print("Accuracy result of Naive Bayes Algorithm is:",accuracy.mean() *  
100)  
NB=accuracy.mean() * 100
```

```

def graph():
import matplotlib.pyplot as plt
    data=[NB]
    alg="Naive Bayes"
    plt.figure(figsize=(5,5))
    b=plt.bar(alg,data,color=("g"))
    plt.title("Accuracy comparison of Stroke",fontsize=15)
    plt.legend(b,data,fontsize=9)

```

```

graph()
In [ ]:
TP = cm1[0][0]
FP = cm1[1][0]
FN = cm1[1][1]
TN = cm1[0][1]
print("True Positive :",TP)
print("True Negative :",TN)
print("False Positive :",FP)
print("False Negative :",FN)
print("")
TPR = TP/(TP+FN)
TNR = TN/(TN+FP)
FPR = FP/(FP+TN)
FNR = FN/(TP+FN)
print("True Positive Rate :",TPR)
print("True Negative Rate :",TNR)
print("False Positive Rate :",FPR)
print("False Negative Rate :",FNR)
print("")
PPV = TP/(TP+FP)
NPV = TN/(TN+FN)
print("Positive Predictive Value :",PPV)
print("Negative predictive value :",NPV)

```

```

def plot_confusion_matrix(cm1, title='Confusion matrix-Naive Bayes',
cmap=plt.cm.Blues):
    target_names=['Predict','Actual']
    plt.imshow(cm1, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = n.arange(len(target_names))
    plt.xticks(tick_marks, target_names, rotation=45)

```

```

plt.yticks(tick_marks, target_names)
plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')

cm1=confusion_matrix(y_test, predictNB)
print('Confusion matrix-Naive Bayes:')
print(cm1)
plot_confusion_matrix(cm1)

```

Module - 6

Decision Tree Classification

```

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

```

```

import warnings
warnings.filterwarnings('ignore')

```

```

data = pd.read_csv('stroke.csv')

```

```

data.head()

```

```

df = data.dropna()

```

```

del df['id']
del df['age']

```

```

df.columns

```

```

from sklearn.preprocessing import LabelEncoder
var_mod = ['gender', 'ever_married', 'work_type', 'Residence_type', 'smoking_status']
le = LabelEncoder()
for i in var_mod:

```

```
df[i]=le.fit_transform(df[i]).astype(int)
```

```
#preprocessing, split test and dataset, split response variable
```

```
X=df.drop(labels='stroke',axis=1)
```

```
#Response variable
```

```
y=df.loc[:, 'stroke']
```

```
'''We'll use a test size of 30%. We also stratify the split on the response variable, which is very important to do because there are so few fraudulent transactions'''
```

```
fromsklearn.model_selectionimporttrain_test_split
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.30,random_state=1,stratify=y)
```

```
print("Number of training dataset: ",len(X_train))
```

```
print("Number of test dataset: ",len(X_test))
```

```
print("Total number of dataset: ",len(X_train)+len(X_test))
```

```
#According to the cross-validated MCC scores, the random forest is the best-performing model, so now let's evaluate its performance on the test set.
```

```
fromsklearn.metricsimportconfusion_matrix,classification_report,matthews_corrcoef,cohen_kappa_score,accuracy_score,average_precision_score,roc_auc_score
```

```
fromsklearn.metricsimportaccuracy_score,confusion_matrix
```

```
fromsklearn.treeimportDecisionTreeClassifier
```

```
fromsklearn.model_selectionimportcross_val_score
```

```
dtree=DecisionTreeClassifier()
```

```
dtree.fit(X_train,y_train)
```

```
predictDT=dtree.predict(X_test)
```

```
print("""
```

```
print('Classification report of Decision Tree Results:')
```

```
print("""
```

```
print(classification_report(y_test,predictDT))
```

```

print("")
cm1=confusion_matrix(y_test,predictDT)
print('Confusion Matrix result of Decision Tree Classifier is:\n',cm1)
print("")
sensitivity1=cm1[0,0]/(cm1[0,0]+cm1[0,1])
print('Sensitivity : ',sensitivity1)
print("")
specificity1=cm1[1,1]/(cm1[1,0]+cm1[1,1])
print('Specificity : ',specificity1)
print("")

```

```

accuracy=cross_val_score(dtree,X,y,scoring='accuracy')
print('Cross validation test results of accuracy:')
print(accuracy)
#get the mean of each fold
print("")
print("Accuracy result of Decision Tree Classifier
is:",accuracy.mean()*100)
DT=accuracy.mean()*100

```

```

def graph():
import matplotlib.pyplot as plt
data=[DT]
alg="Decision Tree Classification"
plt.figure(figsize=(5,5))
b=plt.bar(alg,data,color=("y"))
plt.title("Accuracy comparison of Stroke",fontsize=15)
plt.legend(b,data,fontsize=9)

```

```

graph()

```

```

TP=cm1[0][0]
FP=cm1[1][0]
FN=cm1[1][1]
TN=cm1[0][1]
print("True Positive :",TP)
print("True Negative :",TN)
print("False Positive :",FP)
print("False Negative :",FN)
print("")
TPR=TP/(TP+FN)

```

```

TNR=TN/(TN+FP)
FPR=FP/(FP+TN)
FNR=FN/(TP+FN)
print("True Positive Rate :",TPR)
print("True Negative Rate :",TNR)
print("False Positive Rate :",FPR)
print("False Negative Rate :",FNR)
print("")
PPV=TP/(TP+FP)
NPV=TN/(TN+FN)
print("Positive Predictive Value :",PPV)
print("Negative predictive value :",NPV)

def plot_confusion_matrix(cm1,title='Confusion matrix-
DecisionTreeClassifier',cmap=plt.cm.Blues):
    target_names=['Predict','Actual']
    plt.imshow(cm1,interpolation='nearest',cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks=n.arange(len(target_names))
    plt.xticks(tick_marks,target_names,rotation=45)
    plt.yticks(tick_marks,target_names)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

cm1=confusion_matrix(y_test,predictDT)
print('Confusion matrix-DecisionTreeClassifier:')
print(cm1)
plot_confusion_matrix(cm1)

```

HTML Code:

```

<!DOCTYPE html>
<html >
<!--From https://codepen.io/frytyler/pen/EGdtg-->
<head>
<meta charset="UTF-8">

```

```

<title>TITLE</title>
<link rel="stylesheet" href="{{ url_for('static',
filename='css/bootstrap.min.css') }}">
<link href='https://fonts.googleapis.com/css?family=Pacifico'
rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Arimo'
rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Hind:300'
rel='stylesheet' type='text/css'>
<link
href='https://fonts.googleapis.com/css?family=Open+Sans+Condensed:
300' rel='stylesheet' type='text/css'>
<style>
.back{
    background-image: url('{{ url_for('static', filename='image/stroke1.jpg')
}}");
    background-repeat:no-repeat;
    background-size:cover;
}
.white{
    color:white;
}
.nspace{
    margin:15px 15px 30px 30px;
    padding:9px 10px;
    background: palegreen;
    width:500px
}
.space{

```

```

margin:10px 30px;
padding:10px 10px;
background: palegreen;
width:500px
}
.gap{
padding:10px 20px;
}
</style>
</head>
<body >
<div>
    <div class="jumbotron">
        <h1 style="text-align:center"> PREDICTION OF STROKE </h1>
    </div>
    <div class="back">
        <!-- Main Input For Receiving Query to our ML -->
        <form class="form-group" action="{{ url_for('predict')}}"method="post">

            <div class="row">
                <div class="gap col-md-6 ">
                    <label class="white" for="">GENDER</label>
                    <select class="nspace form-control" name="GENDER"
id="GENDER">
                        <option value=1>MALE</option>
                        <option value=0>FEMALE</option>
                        <option value=2>OTHER</option>
                    </select>

                    <label class="white" for="">HYPERTENSION</label>

```

```
<input type="number" class="space form-control"
step="0.01" name="HYPERTENSION" placeholder="HYPERTENSION"
required="required" /><br>
```

```
<label class="white" for="">HEART_DISEASE</label>
<input type="number" class="space form-control"
step="0.01" name="HEART_DISEASE"
placeholder="HEART_DISEASE" required="required" /><br>
```

```
<label class="white" for="">EVER_MARRIED</label>
```

```
<select class="n space form-control"
name="EVER_MARRIED" id="EVER_MARRIED">
<option value=1>YES</option>
<option value=0>NO</option>
</select>
```

```
<label class="white" for="">WORK_TYPE</label>
<select class="n space form-control" name="WORK_TYPE"
id="WORK_TYPE">
<option value=2>PRIVATE</option>
<option value=3>SELF-EMPLOYED</option>
<option value=0>GOVERNMENT_JOB</option>
<option value=4>CHILDREN</option>
<option value=1>NEVER_WORKED</option>
</select>
```

```
</div>
```

```
<div class="gap col-md-6">
```

```

        <label class="white"
for="">RESIDENCE_TYPE</label>
        <select class="nspace form-control"
name="RESIDENCE_TYPE" id="RESIDENCE_TYPE">
<option value=1>URBAN</option>
<option value=0>RURAL</option>
</select>

        <label class="white" for="">AVERAGE GLUCOSE
LEVEL</label>
        <input type="number" class="space form-control"
step="0.01" name="AVERAGE GLUCOSE LEVEL"
placeholder="AVERAGE GLUCOSE LEVEL" required="required" /><br>

        <label class="white" for="">BMI</label>
        <input type="number" class="space form-control"
step="0.01" name="BMI" placeholder="BMI" required="required" /><br>

        <label class="white" for="">SMOKING STATUS</label>

        <select class="nspace form-control" name="SMOKING
STATUS" id="SMOKING STATUS">
<option value=1>FORMLY-SMOKED</option>
<option value=2>NEVER-SMOKED</option>
<option value=3>SMOKES</option>
<option value=0>UNKNOWN</option>
</select>
</div>
</div>
<div style="padding:2% 35%">

```

```

<button type="submit" class="btn btn-success btn-block"
style="width:350px;padding:20px">Predict</button>
</div>
</form>
</div>
<br>
<br>

<div style="background:skyblue;padding:2% 40%">
    {{ prediction_text }}
</div>
</div>

</body>
</html>

```

Flask Deploy:

```

import numpy as np
from flask import Flask, request, jsonify, render_template

import pickle
import joblib

app = Flask(__name__)
model = joblib.load('rf.pkl')

@app.route('/')

```

```

def home():
    return render_template('index.html')

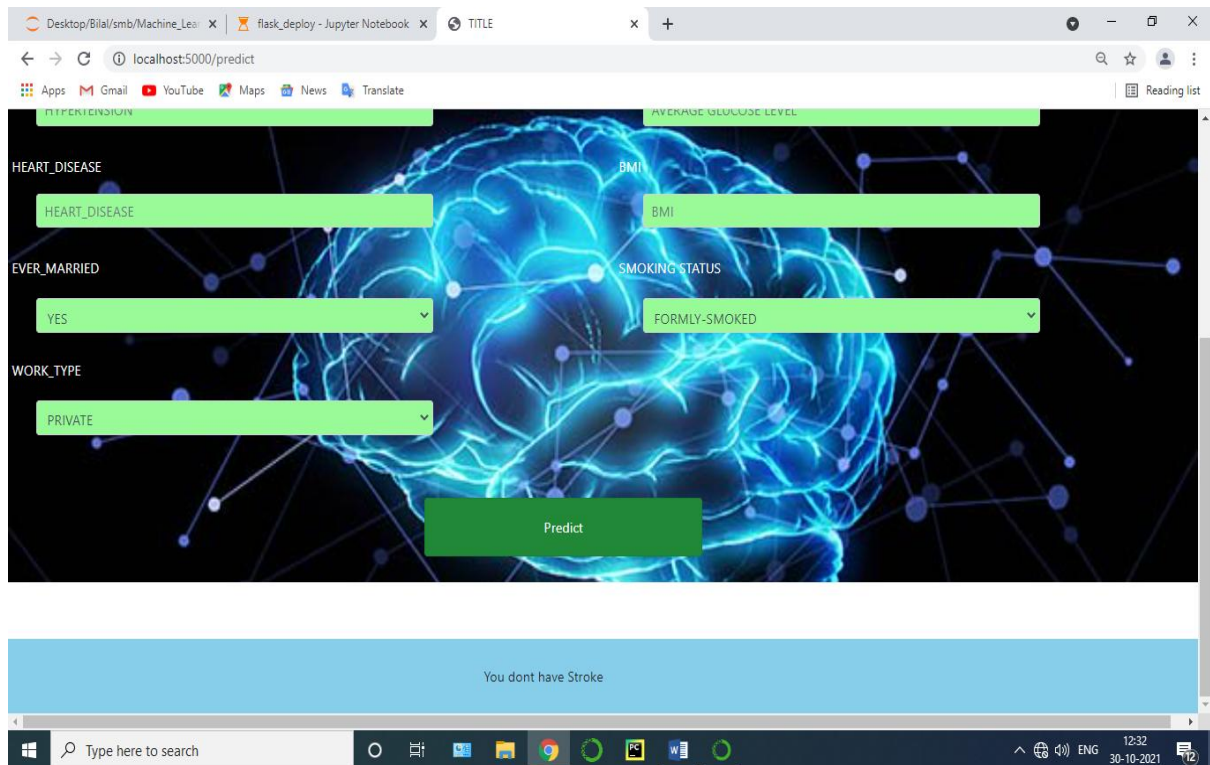
@app.route('/predict',methods=['POST'])
def predict():
    """
    For rendering results on HTML GUI
    """
    int_features = [(x) for x in request.form.values()]
    final_features = [np.array(int_features)]
    print(final_features)
    prediction = model.predict(final_features)

    output = prediction[0]
    if output==0:
        output='You dont have Stroke'
    else:
        output="You have Stroke"

    return render_template('index.html', prediction_text='{}'.format(output))

if __name__ == "__main__":
    app.run(host="localhost", port=5000)

```



27. Conclusion

The analytical process started from data cleaning and processing, missing value, exploratory analysis and finally model building and evaluation. The best accuracy on public test set is higher accuracy score is will be find out. This application can help to find the Prediction of Stroke.

28. Future Work

- Stroke prediction to connect with AI model.
- To automate this process by show the prediction result in web application or desktop application.
- To optimize the work to implement in Artificial Intelligence environment.