

A Project Report On

Machine Learning Based Patient Classification In Emergency Department

Submitted by

K. NANDHINI	(209L1A0567)
E. DILEEP	(209L1A0530)
G. SREELEKHA	(209L1A0541)
P. SRIKANTH	(209L1A0587)
T. KARTHIK	(219L5A0507)

in partial fulfillment for the award of the degree of

Bachelor of technology

in

Computer Science and Engineering

Under the Guidance of

Ms. T.PRAMEELA

Associate Professor, Department of CSE.



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SIDDARTHA EDUCATIONAL ACADEMY GROUP OF INSTITUTIONS

An ISO 9001:2015 & ISO 14001:2015 Certified Institutions

(Approved by AICTE & Affiliated to JNTU Anantapuramu)

C. Gollapalli, Tirupati – 517505

(2020-2024)

SIDDARTHA EDUCATIONAL ACADEMY GROUP OF INSTITUTIONS

An ISO 9001:2015 & ISO 14001:2015 Certified Institutions

(Approved by AICTE & Affiliated to JNTU Anantapuramu)

C. GOLLAPALLI, TIRUPATI – 517505 (A.P), INDIA



CERTIFICATE

This is to certify that the project entitled **“MACHINE LEARNING BASED PATIENT CLASSIFICATION IN EMERGENCY DEPARTMENT”** is the bonafide work carried out by

K. NANDHINI	(209L1A0567)
E. DILEEP	(209L1A0530)
G.SREELEKHA	(209L1A0541)
P. SRIKANTH	(209L1A0587)
T. KARTHIK	(219L5A0507)

B. Tech, students of SEAGI, Affiliated to JNTUA, Anantapuramu in partial fulfilment of the requirements for the award of the Degree of **BACHELOR OF TECHNOLOGY** with the specialization in **COMPUTER SCIENCE AND ENGINEERING** during the Academic year 2023- 2024.

Signature of the Guide

Head of the Department

Viva-Voice held on

Internal Examiner

External Examiner

ACKNOWLEDGEMENT

All endeavors over a long period can be successful only with the advice and support of many well-wishers. I take this opportunity to express my gratitude and appreciation to all of them. we wish to express deep sense of gratitude to my beloved and respected guide **Ms. T.Prameela**, Associate Professor of CSE, Siddartha Educational Academy Group of Institutions, Tirupati, for his valuable guidance, suggestions and constant encouragement and keen interest enriched throughout the course of project work.

We extend sincere thanks to the HOD **Mr. N. ANAND REDDY** for his kind co-operation in completing and making this project a success.

We extend sincere thanks to the **Principal, Prof. K. RAJASEKHAR** for his kind co-operation in completing and making this project a success.

We would like to thank the **Management** for their kind co-operation and for providing infrastructure facilities.

We extend thanks to all the **Teaching staff** of the Department of CSE for their support and encouragement during the course of my project work. I also thank the **non-Teaching staff** of CSE department for being helpful in many ways in successful completion of my work.

Finally, I thank all those who helped me directly or indirectly in successful completion of this project work.

K.NANDHINI	(209L1A0567)
E.DILEEP	(209L1A0530)
G.SREELEKHA	(209L1A0541)
P.SRIKANTH	(209L1A0587)
T. KARTHIK	(219L5A0507)

SIDDARTHA EDUCATIONAL ACADEMY GROUP OF INSTITUTIONS

An ISO 9001:2015 & ISO 14001:2015 Certified Institutions

C. GOLLAPALLI, TIRUPATI – 517505 (A.P), INDIA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



Declaration

I hereby declare that the project work entitled **“MACHINE LEARNING BASED PATIENT CLASSIFICATION IN EMERGENCY DEPARTMENT”** is entirely my original work carried out under the guidance of **Ms. T. Prameela**, Department of Computer Science and Engineering, Siddhartha Educational Academy Group of Institutions, C. Gollapalli, Tirupati, JNTU Anantapur, A.P, India for the award of the degree of **BACHELOR OF TECHNOLOGY** with the specialization in **COMPUTER SCIENCE AND ENGINEERING**. The results carried out in this project report have not been submitted in a part or full for the award of any degree or diploma of this or any other university or institute.

K. NANDHINI	(209L1A0567)
E. DILEEP	(209L1A0530)
G. SREELEKHA	(209L1A0541)
P. SRIKANTH	(209L1A0587)
T. KARTHIK	(219L5A0507)

ABSTRACT

This work contains the classification of patients in an Emergency Department in a hospital according to their critical conditions. Machine learning can be applied based on the patients condition to quickly determine if the patient requires urgent medical intervention from the clinicians or not. Basic vital signs like Systolic Blood Pressure (SBP), Diastolic Blood Pressure (DBP), Respiratory Rate (RR), Oxygen saturation (SPO2), Random Blood Sugar (RBS), Temperature, Pulse Rate (PR) are used as the input for the patients risk level identification. High-risk or non-risk categories are considered as the output for patient classification. Basic machine learning techniques such as LR, Gaussian NB, SVM, KNN and DT are used for the classification. Precision, recall, and F1-score are considered for the evaluation. The decision tree gives best F1-score of 77.67 for the risk level classification of the imbalanced dataset.

CONTENTS

	Page No.
1. INTRODUCTION	1-2
2. LITERATURE SURVEY	3-5
3. SYSTEM STUDY	6-7
3.1 Feasibility Study	6
3.1.1 Economical Feasibility	6
3.1.2 Technical Feasibility	7
3.1.3 Social Feasibility	7
4. SYSTEM ANALYSIS	8-10
4.1 Existing System	8
4.1.1 Disadvantages of Existing System	8
4.2 Proposed System	9
4.2.1 Advantages of Proposed System	10
5. SYSTEM SPECIFICATION	11
5.1 Hardware Requirements	11
5.2 Software Requirements	11
6. SOFTWARE ENVIRONMENT	12-33
6.1 Python	12-23
6.2 Django	24-33
7. SYSTEM DESIGN	34-41
7.1 System Architecture	34
7.2 Data Flow Diagram	35
7.3 UML Diagrams	36
7.3.1 USE case diagram	37
7.3.2 CLASS diagram	38
7.3.3 SEQUENCE diagram	39
7.4 Input And Output Design	40-41

8. IMPLEMENTATION	42-43
9. SAMPLE CODE	44-52
10.SCREENSHOTS	53-64
11.SYSTEM TESTING	64-68
12.CONCLUSION	69
13.BIBLIOGRAPHY	70-71

List of Figures

1. Django database Model	24
2. Django Architecture Model	25
3. System Architecture	34
4. USE case diagram	37
5. CLASS diagram	38
6. SEQUENCE diagram	39
7. Home page	53
8. Register form	55
9. Admin login page	56
10. Admin Home Page	57
11. Users List	58
12. Users Login Page	59
13. Users Home Page	60
14. Data set View	61
15. Machine Learning results	62
16. Result 1	63
17. Result 2	64

1. INTRODUCTION

In the Emergency Departments of hospitals, patients are sorted based on their need for immediate medical treatment. This sorting is done according to the urgency or severity of the health conditions of patients. When a patient arrives, an ER (emergency room) nurse performs a brief, focused assessment and assigns the patient a triage acuity level, also known as a triage score. Triage [1] establishes priorities for care and determines the clinical area of treatment. The acuity level is a proxy measure of how long the patient can safely wait for medical evaluation and treatment. For this purpose, healthcare workers categorize them as per their risk level. Priority level 1 patients are critically ill or high-risk category patients and need immediate medical attention to save their life. This is done by nurses or the assigned staff at hospital triage considering their vital signs and clinical observations. Priority level 2 patients are those who need medical attention but can wait as long as 30 minutes for assessment and treatment. These patients are considered medium-risk level patients. Other cases are considered low-risk patients. They can wait for medical help. This type of patient classification is done by considering their basic vital signs and clinical conditions.

Triage is the prioritization of injured or sick individuals based on their need for emergency treatment. Each organization will have its own triage system, which often includes color coded categories. Triage may be used to meet an organization's short or long-term needs to help determine who gets care first. Based on these results, Machine Learning can determine the patient's criticality. In this study, basic vital parameters are used for patient classification as input. Medium-risk patients and low-risk patients are considered non-critical patients while high-risk cases are considered critical patients. The vital parameters used are Systolic Blood Pressure (SBP), Diastolic Blood Pressure (DBP), Respiratory Rate (RR), Oxygen saturation (SPO₂), Random Blood Sugar (RBS), Temperature, and Pulse Rate (PR). The output is taken as the patient whose condition is critical falls under class 1 and non-critical patients are classified under class 0.

This work focuses on machine learning algorithms to automatically classify critical and noncritical patients based on measured signs. Machine Learning algorithms executed in this work are Gaussian Naïve Bayes Classifier (NBC), Logistic Regression (LR), Support Vector Machine (SVM), K Nearest Neighbours (KNN), and Decision Trees (DTs). All obtained results are compared to evaluate the effectiveness of each method.

The remaining part of the paper is organized as follows: Section II is Literature Survey describing already existing works. Section III is Methodology which highlights the dataset being worked on and the proposed algorithms. Section IV describes the Experimental results that are obtained after building classifiers and discusses the performance evaluation of all classifiers. Section V is the Conclusion that concludes the overall work

2.LITERATURE SURVEY

[1] Tintinalli, Judith E, “Disaster Preparedness”, Tintinalli’s Emergency Medicine: A Comprehensive Study Guide, 9th Edition, McGraw-Hill Education, 2019, ISBN: 1260019934.

Disasters have claimed millions of lives and cost billions of dollars worldwide in the past few decades. Examples of large-scale disasters include the terrorist attacks of September 11, 2001; the 2004 Pacific Ocean tsunami; the 2010 earthquake in Haiti; the 2011 earthquake and tsunami in Japan; and Superstorm Sandy of 2012. Emergency physicians frequently have extensive responsibilities for community and hospital-level disaster preparedness and response. This chapter discusses the definition of a disaster, disaster preparedness and planning, the hospital emergency operations plan, field disaster response, and the ED disaster response.

[2] D.A Debal, T.M. Sitote, “Chronic kidney disease prediction using machine learning techniques”, Journal of Big Data 9, Nov 2022, 10.1186/s40537-022-006575

Goal three of the UN’s Sustainable Development Goal is good health and well-being where it clearly emphasized that non-communicable diseases is emerging challenge. One of the objectives is to reduce premature mortality from non-communicable disease by third in 2030. Chronic kidney disease (CKD) is among the significant contributor to morbidity and mortality from non-communicable diseases that can affected 10–15% of the global population. Early and accurate detection of the stages of CKD is believed to be vital to minimize impacts of patient’s health complications such as hypertension, anemia (low blood count), mineral bone disorder, poor nutritional health, acid base abnormalities, and neurological complications with timely intervention through appropriate medications. Various researches have been carried out using machine learning techniques on the detection of CKD at the premature stage. Their focus was not mainly on the specific stages prediction. In this study, both binary and multi classification for stage prediction have been carried out. The prediction models used include Random Forest (RF), Support Vector Machine (SVM) and Decision Tree (DT). Analysis of variance and recursive feature elimination using cross validation have been applied for feature selection. Evaluation of the models was done using tenfold cross-validation. The results from the experiments indicated that RF based on recursive feature elimination with cross validation has better performance than SVM and DT.

[3] K.M. Almustafa, “Prediction of chronic kidney disease using different classification algorithms”, Informatics in Medicine Unlocked, 2021, Volume 24, 100631, ISSN 2352-9148, <https://doi.org/10.1016/j.imu.2021.100631>.

Diabetes mellitus is a serious health issue in healthcare industry, which is a type of uncontrolled level of sugar. It is a chronic disease happened to the person who are having low insulin production and increase level of blood glucose because glucose is not properly utilized by body. In the medical field, predicting the correct diabetes is an important area that is under research to define a good predictive system to help the doctors to diagnose the disease. In the predictive system, feature selection plays on vital role to select the relevant feature for classification. There are several algorithms were applied on classification of diabetes data. In this proposed work, the features are transformed into high dimensional space before selection. So that the transformation of the features will give the better selection of attributes. With this effort, the proposed work implements the Kernel Principal Component Analysis for dimensionality reduction. KPCA will reduce the features space better than PCA. Once the features are transformed, the proposed work uses Genetic Algorithm to select the relevant and optimal features from the dataset. Then at the last Support Vector Machine is used as a classifier to classify the diabetes mellitus data. The proposed research on applying feature reduction before feature selection will reduce the irrelevant features that will improve the accuracy of the classification based on the selected relevant features. This proposed algorithm on diabetes mellitus data will compare with the existing algorithms to prove the effectiveness of the algorithm.

[4] T. Nibareke, J. Laassiri, “Using Big Data-machine learning models for diabetes prediction and flight delays analytics Journal of Big Data, 2020, 7, pp1-18 10.1186/s40537-020-00355-0

Nowadays large data volumes are daily generated at a high rate. Data from health system, social network, financial, government, marketing, bank transactions as well as the sensors and smart devices are increasing. The tools and models have to be optimized. In this paper we applied and compared Machine Learning algorithms (Linear Regression, Naïve bayes, Decision Tree) to predict diabetes. Further more, we performed analytics on flight delays. The main contribution of this paper is to give an overview of Big Data tools and machine learning models. We highlight some metrics that allow us to choose a more accurate model. We predict diabetes disease using three machine learning models and then compared their performance.

[5]M. Deepika and K. Kalaiselvi, “A Empirical study on Disease Diagnosis using Data Mining Techniques”, 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), 2018, pp. 615-620, doi: 10.1109/ICICCT.2018.8473185.

Data mining is an essential part in learning disclosure process where intelligent agents are incorporated for pattern extraction. In the process of developing data mining applications the most challenging and interesting task is the disease prediction. This paper will be helpful for diagnosing accurate disease by medical practitioners and analysts, portraying various data mining techniques. Data mining applications in medicinal services holds colossal potential and convenience. However the efficiency of data mining techniques on healthcare domain depends on the availability of refined healthcare data. In our current study we discuss few classifier techniques used in medical data analysis. Also few disease prediction analysis like breast cancer prediction, heart disease diagnosis, thyroid prediction and diabetic are considered. The result shows that Decision Tree algorithm suits well for disease prediction as it produces better accuracy results.

3. SYSTEM STUDY

3.1 Feasibility Study

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are,

3.1.1 Economical Feasibility

3.1.2 Technical Feasibility

3.1.3 Social Feasibility

3.1.1 Economical Feasibility

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

- **System development costs:** Costs associated with data collection, model training, software/hardware infrastructure, integration with existing systems.
- **Operational costs:** Costs of maintaining, monitoring, and updating the system over time as new data becomes available.
- **Training costs:** Expenses related to training medical staff on the appropriate use and interpretation of the system's outputs.
- **Potential cost savings:** The system could potentially lead to efficiency gains, better resource allocation, and cost savings in areas like reduced lengths of stay or fewer medical errors. However, these would need to be quantified.
- **Return on investment:** Analyzing whether the potential benefits (cost savings, improved outcomes) outweigh the development and operational costs over a suitable timeframe.

3.1.2 Technical Feasibility

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

The technical feasibility of applying various standard machine learning algorithms like logistic regression, naive Bayes, SVM, KNN, and decision trees for the risk classification task using patient vital sign data. These are well-established techniques that can be implemented using available machine learning libraries and tools.

3.1.3 Social Feasibility

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

- **Acceptance by healthcare professionals:** There may be concerns or resistance from doctors, nurses or other medical staff about relying on an automated system for critical patient triage decisions that could impact care.
- **Patient trust:** Patients may have apprehensions about having their risk levels determined by an algorithmic system rather than human clinical judgment.
- **Transparency and explainability:** The "black-box" nature of some machine learning models could raise concerns about transparency and the ability to explain/justify the system's predictions.
- **Ethical considerations:** There could be ethical debates around the use of automated systems for high-stakes healthcare decisions, potential biases, and the need for human oversight.
- **Change management:** Introducing such a system would require change management efforts to integrate it into existing workflows and train staff on its appropriate use.

4. SYSTEM ANALYSIS

4.1 Existing System

The existing system for machine learning based patient classification in emergency department involves the following steps:

- **Data collection and preparation:** The system collects patient data, including vital signs (e.g., blood pressure, heart rate, respiratory rate, temperature, oxygen saturation), demographic information, medical history, and presenting symptoms. This data can be gathered from various sources, including electronic health records (EHRs), wearable devices, and manual input by medical staff.
- **Feature selection:** Feature selection involves selecting and preprocessing relevant features from the collected data. Feature selection may include extracting or transforming variables to make them suitable for machine learning algorithms..
- **Model training:** Train the selected models using the training dataset. Tune hyperparameters using techniques like grid search or randomized search to optimize model performance.
- **Model evaluation:** The trained model is evaluated on a held-out test set to assess its performance. The evaluation metrics typically include accuracy, precision, recall, and F1 score.
- **Model deployment:** Once the model is evaluated and deemed to be performing well, it can be deployed to production for patient classification in emergency department prediction.

4.1.1 Disadvantage in the Existing System

- **Limited feature set:** The classification models are built using only 7 basic vital sign features like blood pressure, respiratory rate, etc. Having a limited number of features may not capture the full complexity of a patient's condition and could impact the accuracy of risk level classification.
- **Imbalanced dataset:** The paper mentions that the dataset used is imbalanced, with only 519 out of 2578 cases being critical/high-risk. Imbalanced datasets can lead to bias in the classification models towards the majority class.

- **Single hospital data:** The data is collected from only one multi-specialty hospital, which may not be representative of the broader population or different hospital settings. The models trained on this data may not generalize well to other hospitals or regions.
- **Missing data:** The dataset has missing values for several features, which were imputed using statistical mean imputation. While this is a common approach, it may not accurately capture the true underlying values and could introduce bias.
- **Lack of explainability:** Traditional machine learning models like those used in the paper (e.g., decision trees, SVMs) are often considered black-box models, making it difficult to interpret and explain the reasoning behind their predictions. This could be a limitation in a healthcare setting where interpretability is important.
- **Performance limitations:** While the decision tree model achieved a reasonably good F1-score of 77.67%, there is still room for improvement in the classification performance, especially for applications with high stakes like patient triage.

4.2 Proposed System

The proposed system employs various machine learning algorithms to build a system that classifies patients into different risk categories based on their basic vital statistics.

We have considered 2 classes according to the patients risk level

- Class 1: high risk level
- Class 0: low risk level

4.2.1 Advantages in the proposed system:

- **Faster Triage:** Machine learning can quickly assess and classify patients based on their critical conditions and urgency, helping medical staff prioritize care for those who need it most urgently. This can lead to faster treatment for high-risk patients.
- **Improved Patient Outcomes:** Faster and more accurate patient classification can lead to better outcomes, particularly for critical cases that require immediate attention.
- **Patient Safety:** Faster triage and identification of high-risk patients enhance patient safety by ensuring that critical cases receive immediate attention.
- **Emergency Planning:** Historical data and insights generated by the system can aid in emergency planning and resource allocation during disasters or crises.
- **Scalability:** It can handle a large volume of patient data and adapt to varying patient loads, making it suitable for busy Emergency Departments.

5. SYSTEM SPECIFICATION

5.1 HARDWARE REQUIREMENTS:

- System : Intel i3
- Hard Disk : 1 TB.
- Monitor : 14' Color Monitor.
- Mouse : Optical Mouse.
- Ram : 4GB.

5.2 SOFTWARE REQUIREMENTS:

- Operating system : Windows 10.
- Coding Language : Python.
- Front-End : Html. CSS
- Designing : HTML, CSS, JavaScript
- Data Base : SQLite.

6. SOFTWARE ENVIRONMENT

6.1PYTHON

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. An interpreted language, Python has a design philosophy that emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer lines of code than might be used in languages such as C++ or Java. It provides constructs that enable clear programming on both small and large scales. Python interpreters are available for many operating systems. Python, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations. Python is managed by the non-profit Python Software Foundation. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Interactive Mode Programming

Invoking the interpreter without passing a script file as a parameter brings up the following prompt – \$
python

Python 2.4.3 (#1, Nov 11 2010, 13:34:43)

[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2

Type "help", "copyright", "credits" or "license" for more information. >>>

Type the following text at the Python prompt and press the Enter –

```
>>> print "Hello, Python!"
```

If you are running new version of Python, then you would need to use print statement with parenthesis as in print ("Hello, Python!");. However in Python version 2.4.3, this produces the following result – Hello, Python!

Script Mode Programming

Invoking the interpreter with a script parameter begins execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active.

Let us write a simple Python program in a script. Python files have extension .py Type the following source code in a test.py file – Live Demo

```
print "Hello, Python!"
```

We assume that you have Python interpreter set in PATH variable. Now, try to run this program as follows –

```
$ python test.py
```

This produces the following result –

```
Hello, Python!
```

Let us try another way to execute a Python script. Here is the modified test.py file –

Live Demo

```
#!/usr/bin/python
```

```
print "Hello, Python!"
```

We assume that you have Python interpreter available in /usr/bin directory. Now, try to run this program as follows –

```
$ chmod +x test.py # This is to make file executable
```

```
$/test.py
```

This produces the following result –

```
Hello, Python!
```

We Assume that you have Python interpreter set in PATH variable. Now, try to run this program as follows

—

```
$ python test.py
```

This produces the following result – Hello, Python!

Let us try another way to execute a Python script. Here is the modified test.py file – Live Demo

```
#!/usr/bin/python print
```

```
"Hello, Python!"
```

We assume that you have Python interpreter available in /usr/bin directory. Now, try to run this program as follows –

```
$ chmod +x test.py # This is to make file executable $./test.py
```

This produces the following result – Hello, Python!

Python Identifiers

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (`_`) followed by zero or more letters, underscores and digits (0 to 9).

Python does not allow punctuation characters such as `@`, `$`, and `%` within identifiers. Python is a case sensitive programming language. Thus, Manpower and manpower are two different identifiers in Python.

Here are naming conventions for Python identifiers –

Class names start with an uppercase letter. All other identifiers start with a lowercase letter. Starting an identifier with a single leading underscore indicates that the identifier is private.

Starting an identifier with two leading underscores indicates a strongly private identifier.

If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

Reserved Words

The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only. And `exec` `not` `assert` `finally` `or` `break` `for` `pass` `class` `from` `print` `continue` `global` `raise` `def` `if` `return` `del` `import` `try` `elif` `in` `while` `else` `is` `with` `expect` `lambda` `yield` Lines and Indentation Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. For example – `if True:`

```
print "True" else:  
print "False"
```

However, the following block generates an error – if

```
True: print "Answer"  
print "True" else: print  
"Answer" print  
"False"
```

Thus, in Python all the continuous lines indented with same number of spaces would form a block. The following example has various statement blocks –

Note – Do not try to understand the logic at this point of time. Just make sure you understood various blocks even if they are without braces.

```
#!/usr/bin/python import  
sys try:  
  
    # open file stream    file = open(file_name, "w") except  
IOError:    print "There was an error writing to", file_name  
sys.exit() print "Enter '", file_finish, print "' When finished"  
while file_text != file_finish:  
    file_text = raw_input("Enter text: ")  
if file_text == file_finish:  
# close the file  
file.close    break  
file.write(file_text)  
file.write("\n")  
file.close() file_name =  
raw_input("Enter
```

```
filename: ") if len(file_name) == 0:
print "Next time please enter something"
sys.exit() try:    file = open(file_name,
"r") except IOError:    print "There was
an error reading file"
sys.exit() file_text = file.read() file.close()
print file_text
```

Multi-Line Statements

Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue. For example –

```
total      =      item_one      +      \
item_two + \
      item_three
```

Statements contained within the [], {}, or () brackets do not need to use the line continuation character. For example –

```
days = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday']
```

Quotation in Python

Python accepts single ('), double (") and triple (" or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes are used to span the string across multiple lines. For example, all the following are legal –

```
word = 'word' sentence = "This
is a sentence." paragraph =
"""This is a paragraph. It is made
up of multiple lines and
sentences."""
```

Comments in Python

A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.

```
Live Demo  #!/usr/bin/python # First
comment print "Hello, Python!" # second
comment
```

This produces the following result – Hello, Python!

You can type a comment on the same line after a statement or expression – name

```
= "Madisetti" # This is again comment
```

You can comment multiple lines as follows – # This
is a comment.

```
# This is a comment, too.
```

```
# This is a comment, too.
```

```
# I said that already.
```

Following triple-quoted string is also ignored by Python interpreter and can be used as a multiline comments:

```
"""
This is a multiline comment.
"""
```

Using Blank Lines

A line containing only whitespace, possibly with a comment, is known as a blank line and Python totally ignores it.

In an interactive interpreter session, you must enter an empty physical line to terminate a multiline statement.

Waiting for the User

The following line of the program displays the prompt, the statement saying “Press the enter key to exit”, and waits for the user to take action – `#!/usr/bin/python raw_input("\n\nPress the enter key to exit.")` Here, `"\n\n"` is used to create two new lines before displaying the actual line. Once the user presses the key, the program ends. This is a nice trick to keep a console window open until the user is done with an application.

Multiple Statements on a Single Line

The semicolon (;) allows multiple statements on the single line given that neither statement starts a new code block. Here is a sample snip using the semicolon. `import sys; x = 'foo'; sys.stdout.write(x + '\n')`

Multiple Statement Groups as Suites

A group of individual statements, which make a single code block are called suites in Python. Compound or complex statements, such as `if`, `while`, `def`, and `class` require a header line and a suite.

Header lines begin the statement (with the keyword) and terminate with a colon (:) and are followed by one or more lines which make up the suite. For example – `if expression :` suite `elif expression :`

```
    suite
else :    suite
```

Command Line Arguments

Many programs can be run to provide you with some basic information about how they should be run.

Python enables you to do this with `-h` –

```
$ python -h usage: python [option] ... [-c cmd | -m mod | file | -]
```

```
[arg] ... Options and arguments (and corresponding environment variables):
```

```
-c cmd : program passed in as string (terminates option list)
```

```
-d      : debug output from parser (also PYTHONDEBUG=x)
```

```
-E      : ignore environment variables (such as PYTHONPATH)
```

```
-h      : print this help message and exit
```

You can also program your script in such a way that it should accept various options. Command Line Arguments is an advanced topic and should be studied a bit later once you have gone through rest of the Python concepts.

Python Lists

The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.

Creating a list is as simple as putting different comma-separated values between square brackets. For example – list1 = ['physics', 'chemistry', 1997, 2000]; list2 = [1, 2, 3, 4, 5]; list3 = ["a", "b", "c", "d"] Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on.

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also. For example – tup1 = ('physics', 'chemistry', 1997, 2000); tup2 = (1, 2, 3, 4, 5); tup3 = "a", "b", "c", "d";

The empty tuple is written as two parentheses containing nothing – tup1 = ();

To write a tuple containing a single value you have to include a comma, even though there is only one value – tup1 = (50,);

Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

Accessing Values in Tuples

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example –

```
Live Demo #!/usr/bin/python
tup1 = ('physics', 'chemistry', 1997, 2000); tup2
= (1, 2, 3, 4, 5, 6, 7 ); print "tup1[0]:
", tup1[0]; print "tup2[1:5]: ", tup2[1:5];
```

When the above code is executed, it produces the following result – tup1[0]:

```
physics tup2[1:5]: [2, 3, 4, 5]
```

Updating Tuples

Accessing Values in Dictionary

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value. Following is a simple example –

Live Demo

```
#!/usr/bin/python dict = {'Name': 'Zara', 'Age':  
7, 'Class': 'First'} print "dict['Name']: ",  
dict['Name'] print "dict['Age']: ", dict['Age']
```

When the above code is executed, it produces the following result – dict['Name']:

Zara dict['Age']: 7

If we attempt to access a data item with a key, which is not part of the dictionary, we get an error as follows

–

Live Demo

```
#!/usr/bin/python dict = {'Name': 'Zara', 'Age':  
7, 'Class': 'First'} print "dict['Alice']: ",  
dict['Alice']
```

When the above code is executed, it produces the following result –

dict['Alice']:

Traceback (most recent call last): File

"test.py", line 4, in <module> print

"dict['Alice']: ", dict['Alice'];

KeyError: 'Alice'

Updating Dictionary

You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown below in the simple example – Live Demo `#!/usr/bin/python`

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}  
dict['Age'] = 8; # update existing entry dict['School']  
= "DPS School"; # Add new entry print "dict['Age']:  
", dict['Age'] print  
"dict['School']:", dict['School']
```

When the above code is executed, it produces the following result – dict['Age']:

```
8 dict['School']: DPS School
```

Delete Dictionary Elements

You can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.

To explicitly remove an entire dictionary, just use the del statement. Following is a simple example – Live Demo

```
#!/usr/bin/python dict = {'Name': 'Zara', 'Age': 7,  
'Class': 'First'} del dict['Name']; # remove entry  
with key 'Name' dict.clear(); # remove all entries  
in dict del dict ; # delete entire dictionary print  
"dict['Age']:", dict['Age'] print  
"dict['School']:", dict['School']
```

This produces the following result. Note that an exception is raised because after del dict dictionary does not exist any more – dict['Age']:

```
Traceback (most recent call last): File  
"test.py", line 8, in <module> print  
"dict['Age']:", dict['Age'];
```

TypeError: 'type' object is unsubscriptable

Note – del() method is discussed in subsequent section.

Properties of Dictionary Keys

Dictionary values have no restrictions. They can be any arbitrary Python object, either standard objects or user-defined objects. However, same is not true for the keys.

There are two important points to remember about dictionary keys –

(a) More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins. For example

Live Demo

```
#!/usr/bin/python dict = {'Name': 'Zara', 'Age': 7,  
'Name': 'Manni'} print "dict['Name']: ",  
dict['Name']
```

When the above code is executed, it produces the following result – dict['Name']: Manni

(b) Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed. Following is a simple example –

Live Demo

```
#!/usr/bin/python dict =  
{['Name']: 'Zara', 'Age': 7} print  
"dict['Name']: ", dict['Name']
```

When the above code is executed, it produces the following result –

Traceback (most recent call last): File "test.py", line 3, in

```
<module> dict = {'Name': 'Zara', 'Age': 7};
```

TypeError: unhashable type: 'list'

Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to create new tuples as the following example demonstrates – Live Demo

```
#!/usr/bin/python tup1  
= (12, 34.56); tup2 =  
( 'abc', 'xyz');  
# Following action is not valid for tuples  
# tup1[0] = 100;
```

So let's create a new tuple as follows tup3

```
= tup1 + tup2;
```

```
print tup3;
```

When the above code is executed, it produces the following result –

```
(12, 34.56, 'abc', 'xyz')
```

Delete Tuple Elements

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

To explicitly remove an entire tuple, just use the del statement. For example –

Live Demo

```
#!/usr/bin/python tup = ('physics',  
'chemistry', 1997, 2000); print tup; del  
tup; print "After deleting tup : "; print  
tup;
```

This produces the following result. Note an exception raised, this is because after del tup tuple does not exist any more –

```
('physics', 'chemistry', 1997, 2000) After deleting  
tup :
```

```
Traceback (most recent call last):  File
```

```
"test.py", line 9, in <module>    print
```

```
tup;
```

```
NameError: name 'tup' is not defined
```

6.2 DJANGO

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.

Django's primary goal is to ease the creation of complex, database-driven websites. Django emphasizes reusability and "pluggability" of components, rapid development, and the principle of don't repeat yourself. Python is used throughout, even for settings files and data models.

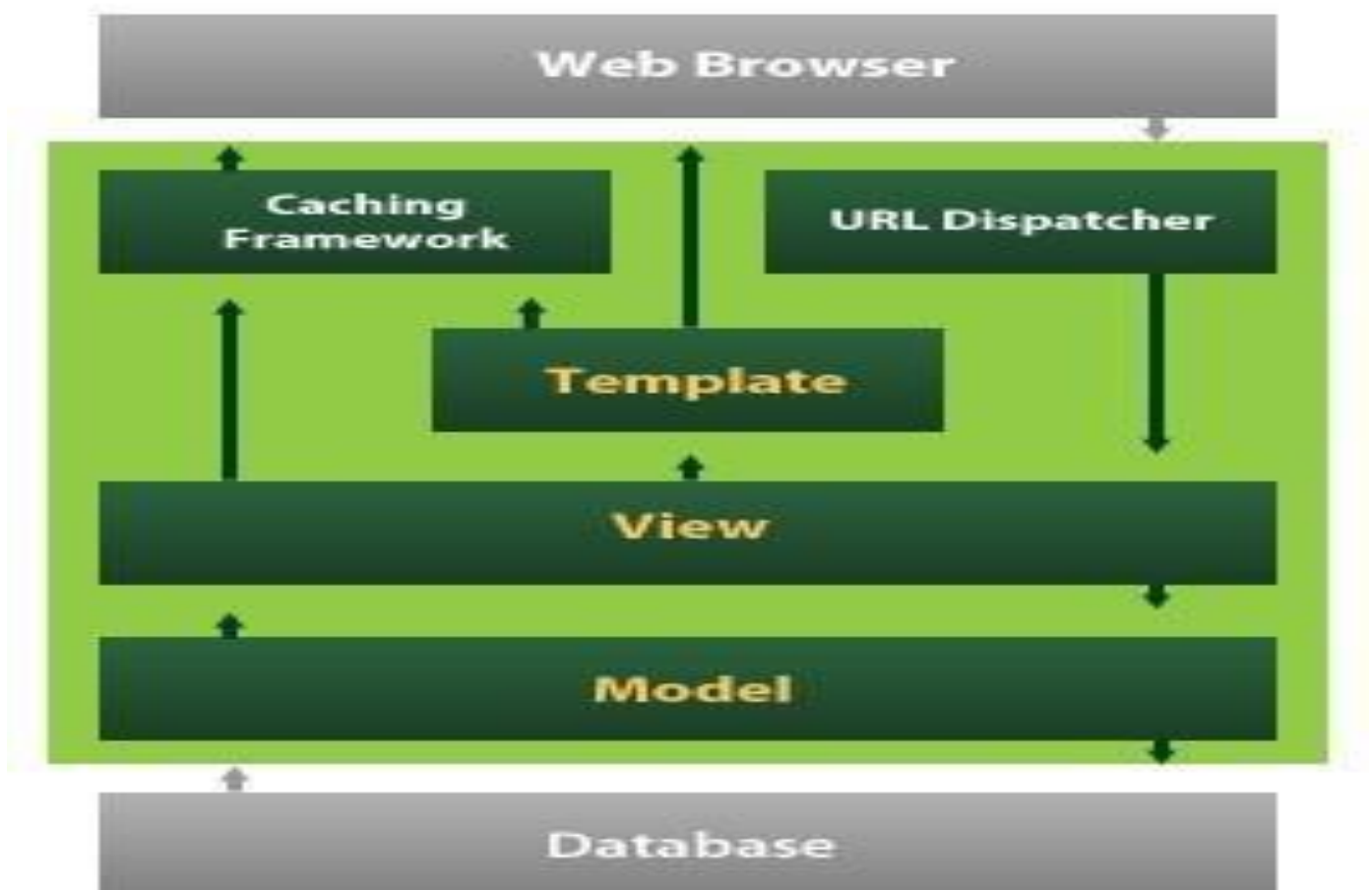


Fig 6.2.1 Django Database Model

Django also provides an optional administrative create, read, update and delete interface that is generated dynamically through introspection and configured via admin models

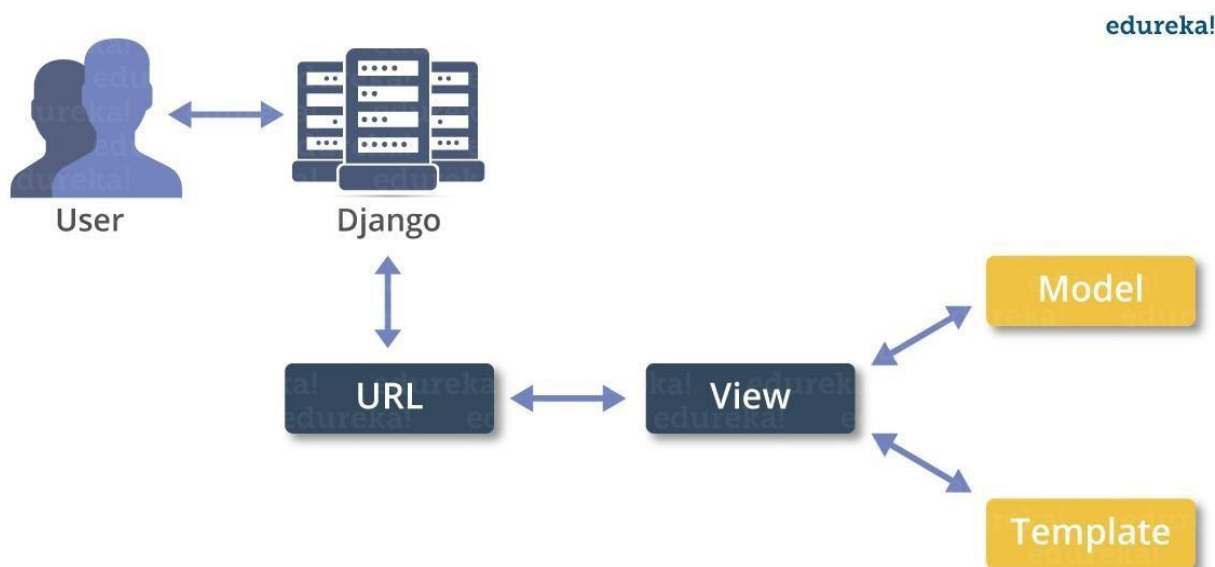


Fig 6.2.2 Django Administrative Creation

Create a Project

Whether you are on Windows or Linux, just get a terminal or a cmd prompt and navigate to the place you want your project to be created, then use this code – `$ django-admin startproject myproject`

This will create a "myproject" folder with the following structure – myproject/

```

manage.py  myproject/
__init__.py
settings.py  urls.py
wsgi.py
  
```

The Project Structure

The “myproject” folder is just your project container, it actually contains two elements – `manage.py` – This file is kind of your project local `django-admin` for interacting with your project via command line (start the development server, sync db...). To get a full list of command accessible via `manage.py` you can use the code –

```
$ python manage.py help
```

The “myproject” subfolder – This folder is the actual python package of your project. It contains four files

`__init__.py` – Just for python, treat this folder as package. `settings.py` – As the name indicates, your project settings. `urls.py` – All links of your project and the function to call. A kind of ToC of your project. `wsgi.py`

– If you need to deploy your project over WSGI.

Setting Up Your Project

Your project is set up in the subfolder `myproject/settings.py`. Following are some important options you might need to set –

`DEBUG = True`

This option lets you set if your project is in debug mode or not. Debug mode lets you get more information about your project's error. Never set it to 'True' for a live project. However, this has to be set to 'True' if you want the Django light server to serve static files. Do it only in the development mode.

`DATABASES = {`

`'default': {`

`'ENGINE': 'django.db.backends.sqlite3',`

`'NAME': 'database.sql',`

`'USER': '',`

`'PASSWORD': '',`

`'HOST': '',`

`'PORT': '',`

`}`

`}`

Database is set in the 'Database' dictionary. The example above is for SQLite engine. As stated earlier,

Django also supports –

MySQL (django.db.backends.mysql)

PostgreSQL (django.db.backends.postgresql_psycopg2)

Oracle (django.db.backends.oracle) and NoSQL DB

MongoDB (django_mongodb_engine)

Before setting any new engine, make sure you have the correct db driver installed.

You can also set others options like: TIME_ZONE, LANGUAGE_CODE, TEMPLATE...

Now that your project is created and configured make sure it's working –

```
$ python manage.py runserver
```

You will get something like the following on running the above code – Validating models...

0 errors found

September 03, 2015 - 11:41:50

Django version 1.6.11, using settings 'myproject.settings'

Starting development server at http://127.0.0.1:8000/ Quit the server with CONTROL-C.

A project is a sum of many applications. Every application has an objective and can be reused into another project, like the contact form on a website can be an application, and can be reused for others. See it as a module of your project.

Create an Application

We assume you are in your project folder. In our main “myproject” folder, the same folder then manage.py

```
$ python manage.py startapp myapp
```

You just created myapp application and like project, Django create a “myapp” folder with the application structure – myapp/ __init__.py admin.py models.py tests.py views.py

__init__.py – Just to make sure python handles this folder as a package. admin.py

– This file helps you make the app modifiable in the admin interface. models.py –

This is where all the application models are stored.

tests.py – This is where your unit tests are. views.py

– This is where your application views are.

Get the Project to Know About Your Application

At this stage we have our "myapp" application, now we need to register it with our Django project "myproject". To do so, update INSTALLED_APPS tuple in the settings.py file of your project (add your app name) –

```
INSTALLED_APPS = (  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'myapp',  
)
```

Creating forms in Django, is really similar to creating a model. Here again, we just need to inherit from Django class and the class attributes will be the form fields. Let's add a forms.py file in myapp folder to contain our app forms. We will create a login form. myapp/forms.py

```
#-*- coding: utf-8 -*- from django
```

```
import forms class
```

```
LoginForm(forms.Form):
```

```
    user = forms.CharField(max_length = 100)
```

```
    password = forms.CharField(widget = forms.PasswordInput())
```

As seen above, the field type can take "widget" argument for html rendering; in our case, we want the password to be hidden, not displayed. Many others widget are present in Django: DateInput for dates, CheckboxInput for checkboxes, etc.

Using Form in a View

There are two kinds of HTTP requests, GET and POST. In Django, the request object passed as parameter to your view has an attribute called "method" where the type of the request is set, and all data passed via POST can be accessed via the request.POST dictionary.

Let's create a login view in our myapp/views.py –

```
#-*- coding: utf-8 -*- from
```

```
myapp.forms import LoginForm def login(request):
```

```
    username = "not logged in"    if
```

```
request.method == "POST":
```

```
    #Get the posted form
```

```
    MyLoginForm = LoginForm(request.POST)    if
```

```
MyLoginForm.is_valid():        username        =
```

```
    MyLoginForm.cleaned_data['username']    else:
```

```
    MyLoginForm = Loginform()
```

```
    return render(request, 'loggedin.html', {"username" : username})
```

The view will display the result of the login form posted through the loggedin.html. To test it, we will first need the login form template. Let's call it login.html.

```
<html>

<body>
<form name = "form" action = "{% url 'myapp.views.login' %}"          method
= "POST" >{% csrf_token %}
    <div style = "max-width:470px;">

        <center>

            <input type = "text" style = "margin-left:20%;"
placeholder = "Identifiant" name = "username" />
        </center>

    </div>

    <br>

    <div style = "max-width:470px;">

        <center>

            <input type = "password" style = "margin-left:20%;"
placeholder = "password" name = "password" />
        </center>

    </div>

    <br>

    <div style = "max-width:470px;">

        <center>

            <button style = "border:0px; background-color:#4285F4; margin-top:8%;
height:35px; width:80%;margin-left:19%;" type = "submit"
value = "Login" >
            <strong>Login</strong>

        </button>

    </center>
```

```
</div>

    </form>

</body>

</html>
```

The template will display a login form and post the result to our login view above. You have probably noticed the tag in the template, which is just to prevent Cross-site Request Forgery (CSRF) attack on your site. {% csrf_token %}

Once we have the login template, we need the loggedin.html template that will be rendered after form treatment. <html>

```
<body>

    You are : <strong>{{username}}</strong>

</body>

</html>
```

```
Now, we just need our pair of URLs to get started: myapp/urls.py from
django.conf.urls import patterns, url from django.views.generic import TemplateView
urlpatterns = patterns('myapp.views',
url(r'^connection/', TemplateView.as_view(template_name = 'login.html')),
url(r'^login/', 'login', name = 'login'))
```

When accessing `"/myapp/connection"`, we will get the following `login.html` template rendered – Setting Up Sessions

In Django, enabling session is done in your project `settings.py`, by adding some lines to the `MIDDLEWARE_CLASSES` and the `INSTALLED_APPS` options. This should be done while creating the project, but it's always good to know, so `MIDDLEWARE_CLASSES` should have –

```
'django.contrib.sessions.middleware.SessionMiddleware'
```

And `INSTALLED_APPS` should have –

```
'django.contrib.sessions'
```

By default, Django saves session information in database (`django_session` table or collection), but you can configure the engine to store information using other ways like: in file or in cache.

When session is enabled, every request (first argument of any view in Django) has a session (dict) attribute.

Let's create a simple sample to see how to create and save sessions. We have built a simple login system before (see Django form processing chapter and Django Cookies Handling chapter). Let us save the username in a cookie so, if not signed out, when accessing our login page you won't see the login form. Basically, let's make our login system we used in Django Cookies handling more secure, by saving cookies server side.

For this, first let's change our login view to save our username cookie server side –

```
def login(request):
    username = 'not logged in'
    if request.method == 'POST':
```



```
MyLoginForm = LoginForm(request.POST)    if
MyLoginForm.is_valid():    username =
MyLoginForm.cleaned_data['username']
request.session['username'] = username    else:
MyLoginForm = LoginForm()
    return render(request, 'loggedin.html', {"username" : username})
```

Then let us create formView view for the login form, where we won't display the form if cookie is set –

```
def formView(request):    if request.session.has_key('username'):    username =
request.session['username']    return render(request, 'loggedin.html', {"username" : username})    else:
    return render(request, 'login.html', {})
```

Now let us change the url.py file to change the url so it pairs with our new view –

```
from django.conf.urls import patterns, url from django.views.generic import
TemplateView urlpatterns = patterns('myapp.views',    url(r'^connection/', 'formView',
name = 'loginform'),    url(r'^login/', 'login', name
= 'login'))
```

7.SYSTEM DESIGN

7.1 SYSTEM ARCHITECTURE

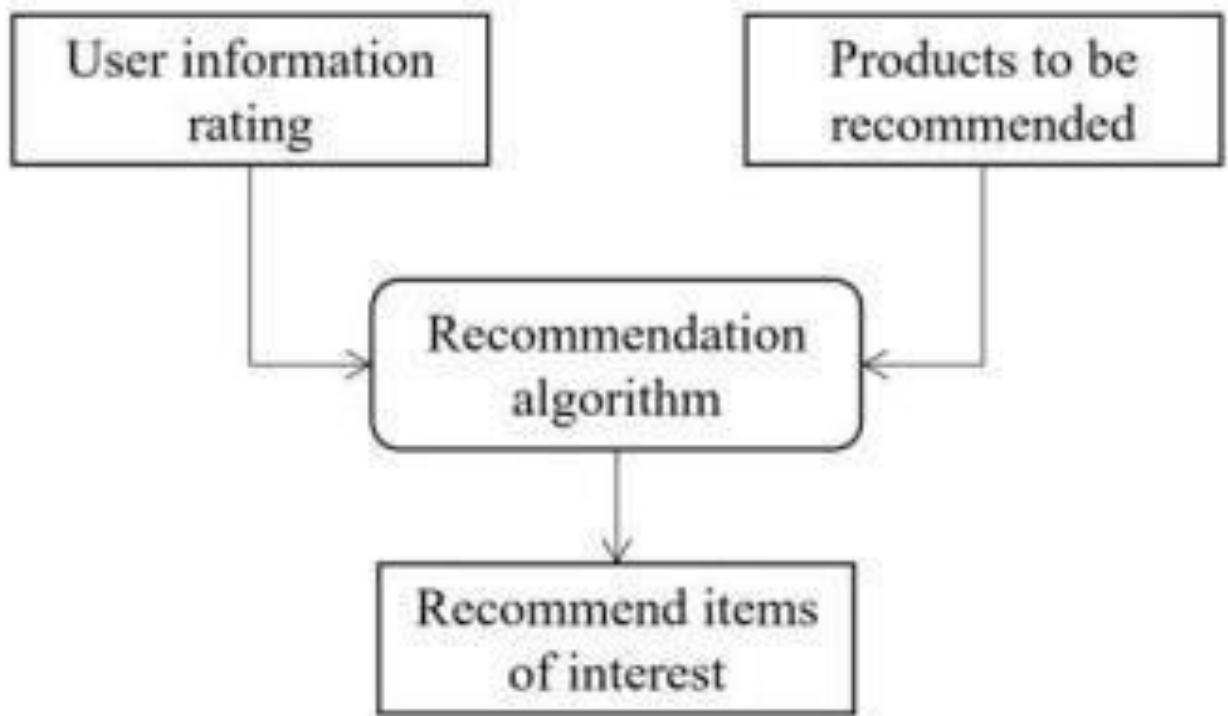


Fig 7.1 System Architecture

7.2 DATA FLOW DIAGRAM:

- The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.
- The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.
- DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.
- DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional detail.

7.3 UML DIAGRAMS

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

GOALS:

- The Primary goals in the design of the UML are as follows:
- Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
- Provide extendibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development process.
- Provide a formal basis for understanding the modeling language.
- Encourage the growth of OO tools market.
- Support higher level development concepts such as collaborations, frameworks, patterns and components.
- Integrate best practices.

7.3.1 USE CASE DIAGRAM:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

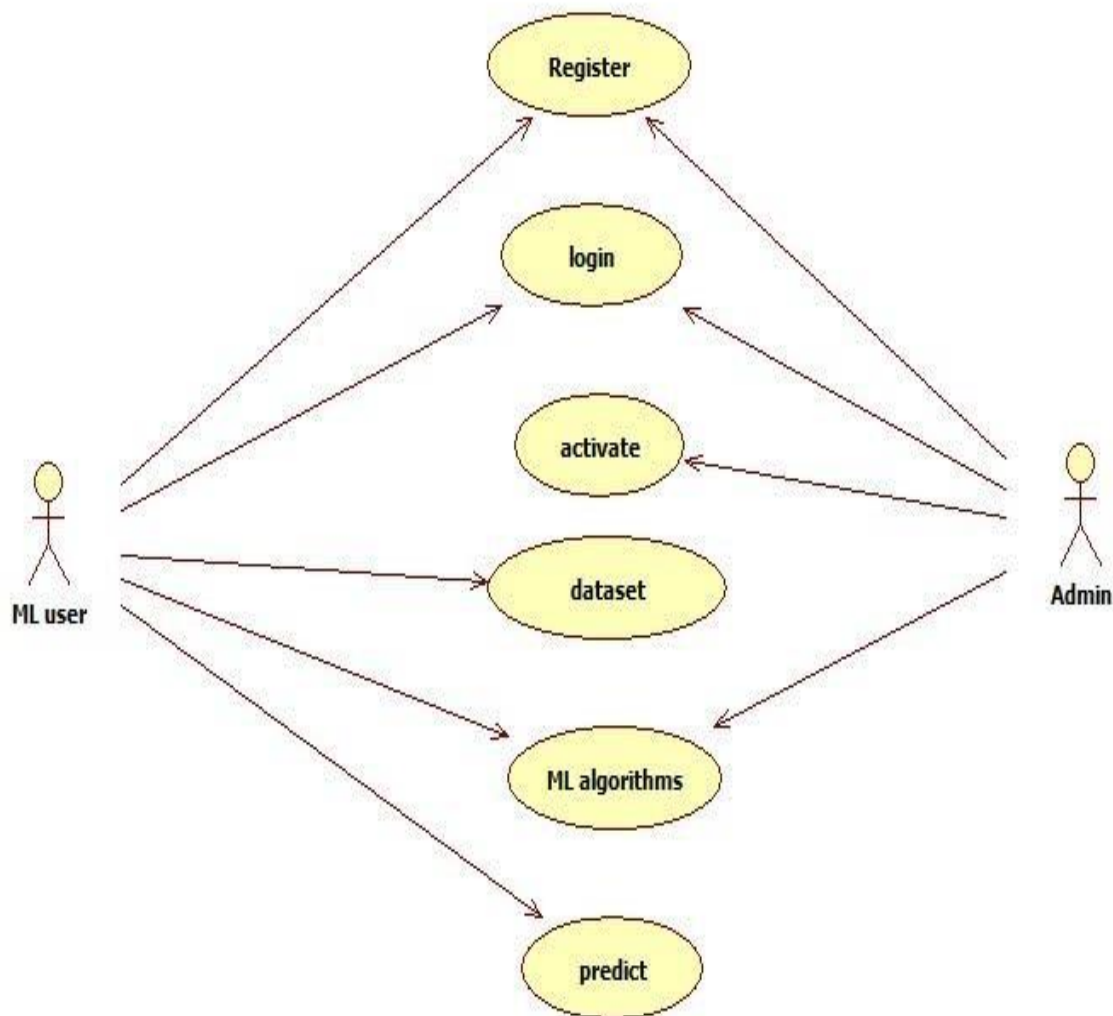


Fig 7.3.1 Use Case Diagram

7.3.2 CLASS DIAGRAM:

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information

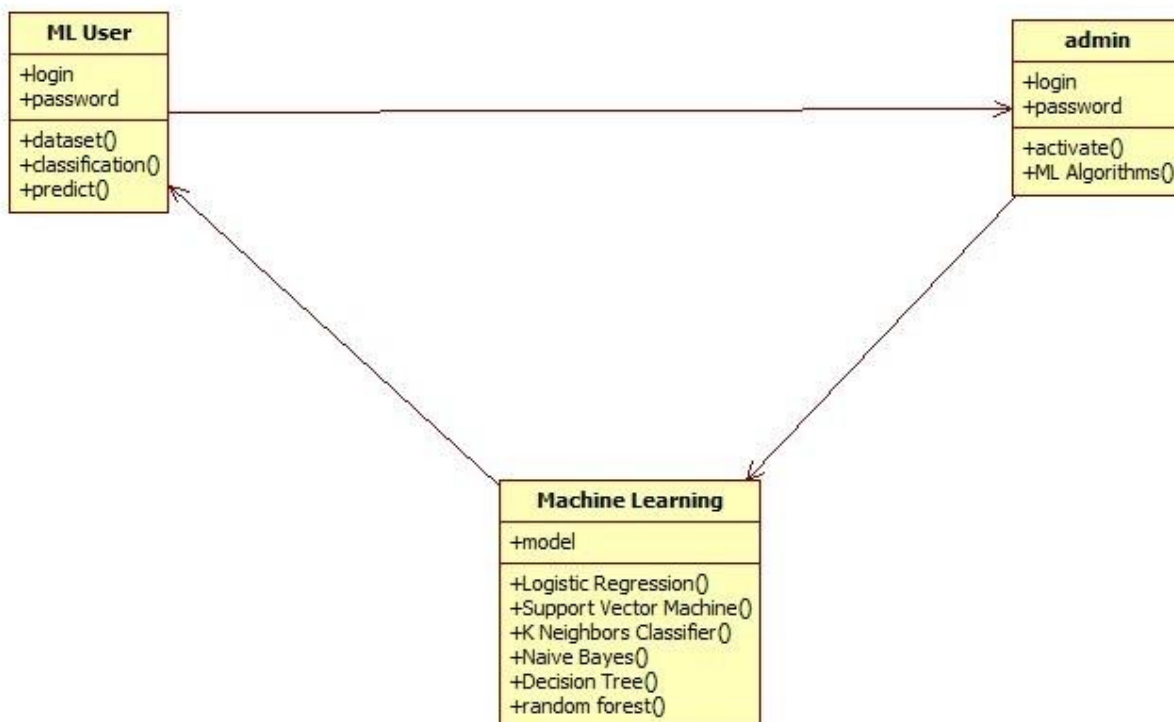


Fig 7.3.2 Class Diagram

7.3.3 SEQUENCE DIAGRAM:

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

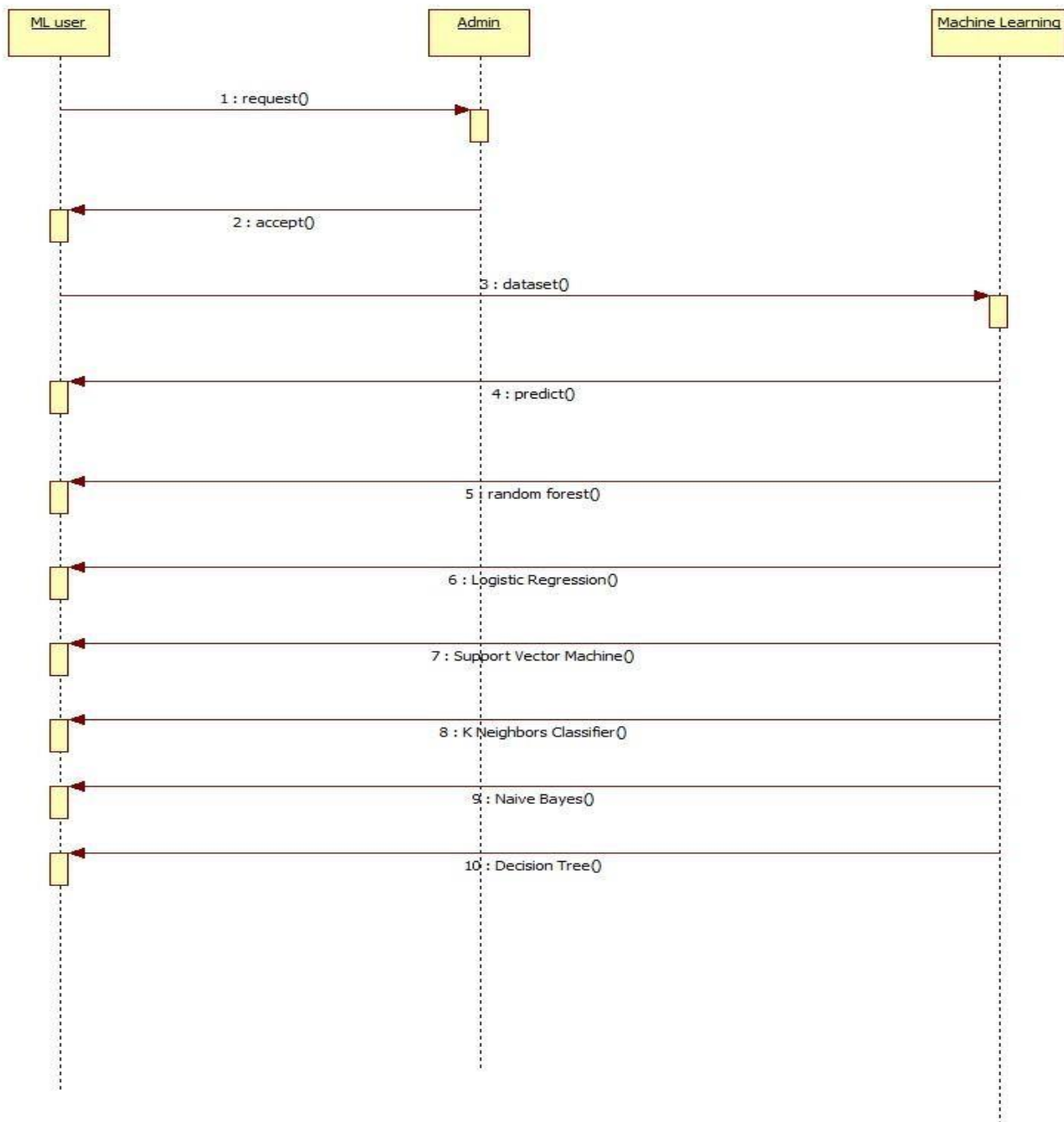


Fig 7.3.3 Sequence Diagram

7.7 INPUT AND OUTPUT DESIGN

INPUT DESIGN

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:

- What data should be given as input?
- How the data should be arranged or coded?
- The dialog to guide the operating personnel in providing input.
- Methods for preparing input validations and steps to follow when error occur.

OBJECTIVES

- Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.
- It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.
- When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user will not be in maize of instant. Thus the objective of input design is to create an input layout that is easy to follow.

OUTPUT DESIGN

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system's relationship to help user decision-making.

- Designing computer output should proceed in an organized, well thought out manner; the right output must be developed while ensuring that each output element is designed so that people will find the system can use easily and effectively. When analysis design computer output, they should Identify the specific output that is needed to meet the requirements.
- 2.Select methods for presenting information.
- 3.Create document, report, or other formats that contain information produced by the system.

8. IMPLEMENTATION

MODULES:

- ML User
- Admin

MODULES DESCRIPTION:

Admin:

The Admin module allows administrators to manage the system and its users. The Admin can perform the following tasks:

- Create and manage users: The Admin can create new ML User accounts, update existing accounts, and delete accounts.
- Manage models: The Admin can monitor the performance of deployed models and update them as needed.
- Manage system resources: The Admin can ensure that the system has enough resources to run smoothly and efficiently.

Example

Here is an example of how the ML User and Admin modules might be used in a quantum machine learning system for diabetes prediction:

- A ML User uploads a dataset of patients with diabetes and non-diabetes to the system.
- The ML User selects the features that they want to use for diabetes prediction.
- The ML User trains a quantum machine learning model on the selected features.
- The ML User evaluates the trained model on a held-out test set.
- If the model performs well, the ML User deploys it to production for diabetes prediction.
- The Admin monitors the performance of the deployed model and updates it as needed.

Machine learning User:

The ML User module allows users to interact with the system to perform the following tasks:

- Upload data: Users can upload datasets of patients with diabetes and non-diabetes to the system.
- Select features: Users can select the features that they want to use for diabetes prediction.
- Train models: Users can train quantum machine learning models on the selected features.
- Evaluate models: Users can evaluate the trained models on held-out test sets to assess their performance.
- Deploy models: Users can deploy the trained models to production for diabetes prediction

9.SOURCE CODE

User side views:

```
from django.shortcuts import render,
HttpResponse from django.contrib import
messages from .forms import
UserRegistrationForm from .models import
UserRegistrationModel from django.conf import
settings import os import pickle

# Create your views here.
def UserRegisterActions(request):
if request.method == 'POST':
    form =
UserRegistrationForm(request.POST)    if
form.is_valid():        print('Data is Valid')
form.save()
        messages.success(request, 'You have been successfully registered')
form = UserRegistrationForm()
        return render(request, 'UserRegistrations.html', {'form': form})
else:
        messages.success(request, 'Email or Mobile Already
Existed')        print("Invalid form")    else:
        form = UserRegistrationForm()
        return render(request, 'UserRegistrations.html', {'form': form})

def UserLoginCheck(request):
if request.method == "POST":
    loginid    =    request.POST.get('loginname')
pswd = request.POST.get('pswd')
    print("Login ID = ", loginid, ' Password = ', pswd)
```

```
try:
    check = UserRegistrationModel.objects.get(loginid=loginid,
password=pswd)      status = check.status      print('Status is = ', status)
if status == "activated":
    request.session['id'] = check.id
request.session['loggeduser'] = check.name
request.session['loginid'] = loginid
request.session['email'] = check.email
print("User id At", check.id, status)
    return render(request, 'users/UserHome.html', {})
else:
    messages.success(request, 'Your Account Not at
activated')      return render(request, 'UserLogin.html')
except Exception as e:
    print('Exception is ', str(e))
pass
    messages.success(request, 'Invalid Login id and password')
return render(request, 'UserLogin.html', {})
```

```
def UserHome(request):      return render(request,
'users/UserHome.html', {})
```

```
def usersViewDataset(request):      dataset =
os.path.join(settings.MEDIA_ROOT, 'EmmergencyDataset.csv')      import
pandas as pd      df = pd.read_csv(dataset)
```

```
df = df.to_html(index=None)

return render(request, 'users/viewData.html', {'data': df})


def userClassificationResults(request):
import pandas as pd

    from .utility import EmmergencyClassi

    rf_report = EmmergencyClassi.process_randomForest()
dt_report = EmmergencyClassi.process_decesionTree()
nb_report = EmmergencyClassi.process_naiveBayes()
gb_report = EmmergencyClassi.process_knn()

    lg_report =
EmmergencyClassi.process_LogisticRegression()
svm_report = EmmergencyClassi.process_SVM()    rf_report =
pd.DataFrame(rf_report).transpose()    rf_report =
pd.DataFrame(rf_report)    dt_report =
pd.DataFrame(dt_report).transpose()    dt_report =
pd.DataFrame(dt_report)

    nb_report =
pd.DataFrame(nb_report).transpose()    nb_report
= pd.DataFrame(nb_report)    gb_report =
pd.DataFrame(gb_report).transpose()    gb_report
= pd.DataFrame(gb_report)    lg_report =
pd.DataFrame(lg_report).transpose()    lg_report =
pd.DataFrame(lg_report)

    svm_report =
pd.DataFrame(svm_report).transpose()    svm_report
= pd.DataFrame(svm_report)    #
report_df.to_csv("rf_report.csv")    return
render(request, 'users/ml_results.html',

        {'lg':    lg_report.to_html,    'svm':    svm_report.to_html,    'rf':    rf_report.to_html,    'dt':
```

```
dt_report.to_html,
        'nb': nb_report.to_html, 'gb': gb_report.to_html}))

def UserPredictions(request):
    if request.method == 'POST':
        age = int(request.POST.get('age'))
        gender = int(request.POST.get('gender'))
        pulse = int(request.POST.get('pulse'))
        systolicBloodPressure = int(request.POST.get('systolicBloodPressure'))
        diastolicBloodPressure = int(request.POST.get('diastolicBloodPressure'))
        respiratoryRate = int(request.POST.get('respiratoryRate'))

        spo2 = float(request.POST.get('spo2'))
        randomBloodSugar = int(request.POST.get('randomBloodSugar'))
        temperature = float(request.POST.get('temperature'))

        test_data = [age, gender, pulse, systolicBloodPressure, diastolicBloodPressure, respiratoryRate, spo2, randomBloodSugar, temperature] # noqa: E501

        model_path = os.path.join(settings.MEDIA_ROOT,
                                   'alexmodel.pkl')
        model = pickle.load(open(model_path, 'rb'))
        result = model.predict([test_data])
        if result[0] == 0:
            msg = 'not Needed'
        else:
            msg = 'Needed'
        print("Result=", result)
        return render(request, "users/predictForm.html", {'result': msg})
    else:
        return render(request, "users/predictForm.html", {})
```

Base.html:

```
{% load static %}
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, maximum-scale=1">
  <title>Machine Learning Based Patient Classification In Emergency Department</title>
  <link rel="icon" href="{%static 'favicon.png'%}" type="image/png">
  <link href="{%static 'css/bootstrap.min.css'%}" rel="stylesheet" type="text/css">
  <link href="{%static 'css/style.css'%}" rel="stylesheet" type="text/css">
  <link href="{%static 'css/font-awesome.css'%}" rel="stylesheet" type="text/css">
  <link href="{%static 'css/animate.css'%}" rel="stylesheet" type="text/css">

</head>
<body>

<!--Header_section-->
<header id="header_wrapper">
  <div class="container">
    <div class="header_box">
      <div class="logo"><a href="#"><h2>Emergency Classification</h2></a></div>
      <nav class="navbar navbar-inverse" role="navigation">
        <div class="navbar-header">
          <button type="button" id="nav-toggle" class="navbar-toggle" data-toggle="collapse"
            data-target="#main-nav"><span class="sr-only">Toggle navigation</span> <span
            class="icon-bar"></span> <span class="icon-bar"></span> <span class="icon-
bar"></span>
          </button>
        </div>
        <div id="main-nav" class="collapse navbar-collapse navStyle">
          <ul class="nav navbar-nav" id="mainNav">
            <li><a href="{%url 'index'%}">Home</a></li>
            <li><a href="{%url 'UserLogin'%}">Login</a></li>
            <li><a href="{%url 'AdminLogin'%}">Admin</a></li>
            <li><a href="{%url 'UserRegister'%}">Registrations</a></li>
          </ul>
        </div>
      </nav>
    </div>
  </div>
</div>
```



```
</header>
<!--Header_section-->
{%block contents%}

<!--Aboutus-->
{%endblock%}
<!--Footer-->
<footer class="footer_wrapper" id="contact">
  <div class="container">
    <div class="footer_bottom"><span>Copyright © 2024,  Template by <a
      href="#">Alex Corporations</a>. </span></div>
  </div>
</footer>

<script type="text/javascript" src="{%static 'js/jquery-1.11.0.min.js'%}"></script>
<script type="text/javascript" src="{%static 'js/bootstrap.min.js'%}"></script>
<script type="text/javascript" src="{%static 'js/jquery-scrolltofixed.js'%}"></script>
<script type="text/javascript" src="{%static 'js/jquery.nav.js'%}"></script>
<script type="text/javascript" src="{%static 'js/jquery.easing.1.3.js'%}"></script>
<script type="text/javascript" src="{%static 'js/jquery.isotope.js'%}"></script>
<script type="text/javascript" src="{%static 'js/wow.js'%}"></script>
<script type="text/javascript" src="{%static 'js/custom.js'%}"></script>

</body>
</html>
```

Admin side views:

```
from django.shortcuts import render,HttpResponse from
django.contrib import messages
from users.models import UserRegistrationModel

# Create your views here.

def AdminLoginCheck(request):
    if request.method == 'POST':
        usrid = request.POST.get('loginid')
        pswd = request.POST.get('pswd')
        print("User ID is = ", usrid)    if usrid ==
        'admin' and pswd == 'admin':
            return render(request, 'admins/AdminHome.html')
        elif usrid == 'Admin' and pswd == 'Admin':
            return render(request, 'admins/AdminHome.html')
        else:
            messages.success(request, 'Please Check Your Login Details')

    return render(request, 'AdminLogin.html', {})

def AdminHome(request):    return render(request,
'admins/AdminHome.html')

def ViewRegisteredUsers(request):    data =
UserRegistrationModel.objects.all()
    return render(request, 'admins/RegisteredUsers.html', {'data': data})
```

```
def
AdminActivaUsers(request):
if request.method == 'GET':
id = request.GET.get('uid')
status = 'activated'
print("PID = ", id, status)
    UserRegistrationModel.objects.filter(id=id).update(status=status)
data = UserRegistrationModel.objects.all()
    return render(request, 'admins/RegisteredUsers.html', {'data': data})
```

```
def adminResults(request):
import pandas as pd
    from users.utility import EmmergencyClassi    rf_report =
EmmergencyClassi.process_randomForest()    dt_report =
EmmergencyClassi.process_decesionTree()    nb_report =
EmmergencyClassi.process_naiveBayes()    gb_report =
EmmergencyClassi.process_knn()    lg_report =

EmmergencyClassi.process_LogisticRegression()
svm_report = EmmergencyClassi.process_SVM()    rf_report =
pd.DataFrame(rf_report).transpose()    rf_report =
pd.DataFrame(rf_report)    dt_report =
pd.DataFrame(dt_report).transpose()
```

```
dt_report = pd.DataFrame(dt_report)
    nb_report =
pd.DataFrame(nb_report).transpose()    nb_report
= pd.DataFrame(nb_report)    gb_report =
pd.DataFrame(gb_report).transpose()    gb_report
= pd.DataFrame(gb_report)    lg_report =
pd.DataFrame(lg_report).transpose()    lg_report =
pd.DataFrame(lg_report)
    svm_report =
pd.DataFrame(svm_report).transpose()    svm_report
= pd.DataFrame(svm_report)    #
report_df.to_csv("rf_report.csv")    return
render(request, 'admins/results.html',
        {'lg':    lg_report.to_html,    'svm':    svm_report.to_html,    'rf':    rf_report.to_html,    'dt':
dt_report.to_html,
        'nb': nb_report.to_html, 'gb': gb_report.to_html})
```

10. SCREEN SHOTS

Home page



Fig 10.1 Home page

Register Form

EMERGENCY CLASSIFICATION

HOME LOGIN ADMIN REGISTRATIONS

Registration Form

User Name

Login ID

Password

Mobile

email

Locality

Address

City

State

Register


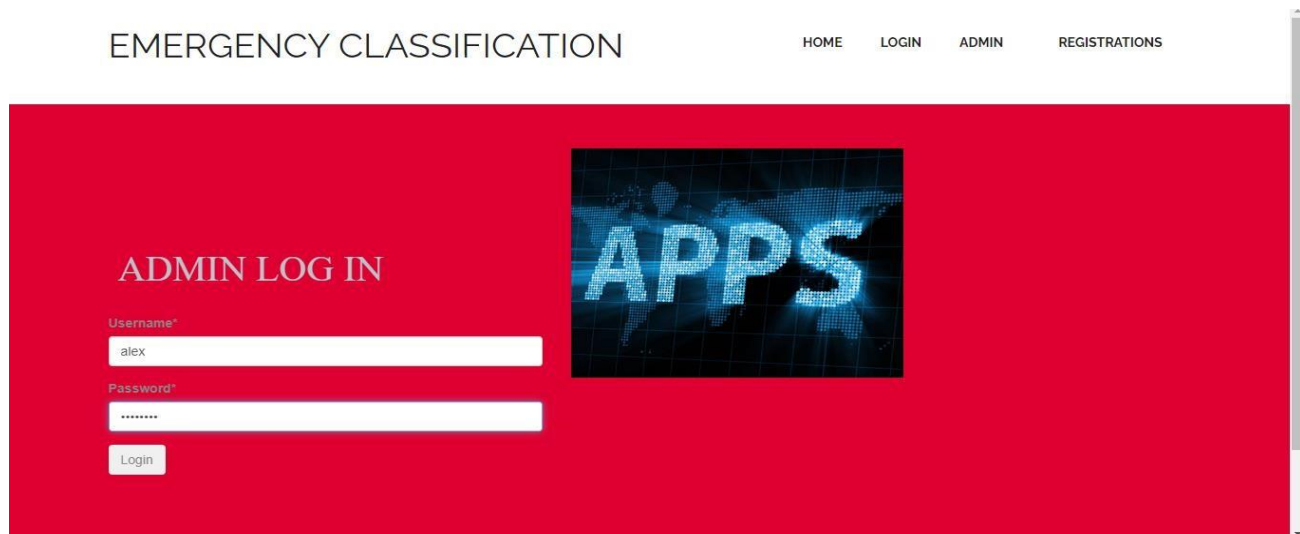


Fig 10.2 Register Form

Admin Login Page



The screenshot shows a web application interface for an emergency department. At the top, the title "EMERGENCY CLASSIFICATION" is displayed on the left, and a navigation menu with links "HOME", "LOGIN", "ADMIN", and "REGISTRATIONS" is on the right. The "ADMIN" link is highlighted. Below the navigation bar is a large red rectangular area. On the left side of this red area, the text "ADMIN LOG IN" is written in white. Below this text are two input fields: "Username*" with the value "alex" and "Password*" with masked characters "*****". A "Login" button is positioned below the password field. To the right of the input fields is a square image with a dark background, a blue grid pattern, and the word "APPS" in large, glowing blue letters.

Fig 10.3 Admin Login Page

Admin Home Page

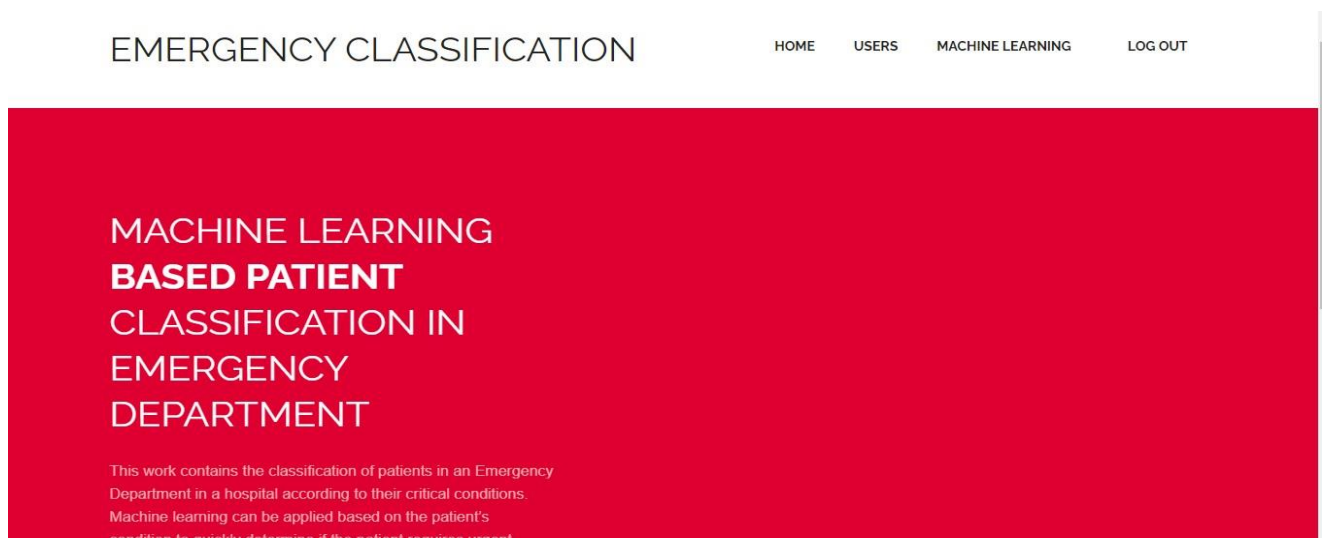


Fig 10.4 Admin Home Page

Users Lists:

EMERGENCY CLASSIFICATION				HOME	USERS	MACHINE LEARNING	LOG OUT
View RegisterUser Details							
S.No	Name	Login ID	Mobile	Email	Locality	Status	Activate
1	alex	alex	9849812345	lx160cm@gmail.com	Hyderabad	activated	Activated
2	niharika	niharika	9875012456	niharika@gmail.com	Hyderabad	activated	Activated

Copyright © 2024, Template by Alex Corporations.

Fig 10.5 Users Lists

User login Page

EMERGENCY CLASSIFICATION

HOME LOGIN ADMIN REGISTRATIONS

MACHINE LEARNING
USER LOGIN FORM

Username*

Password*

Login



Fig 10.6 User Login Page

User Home

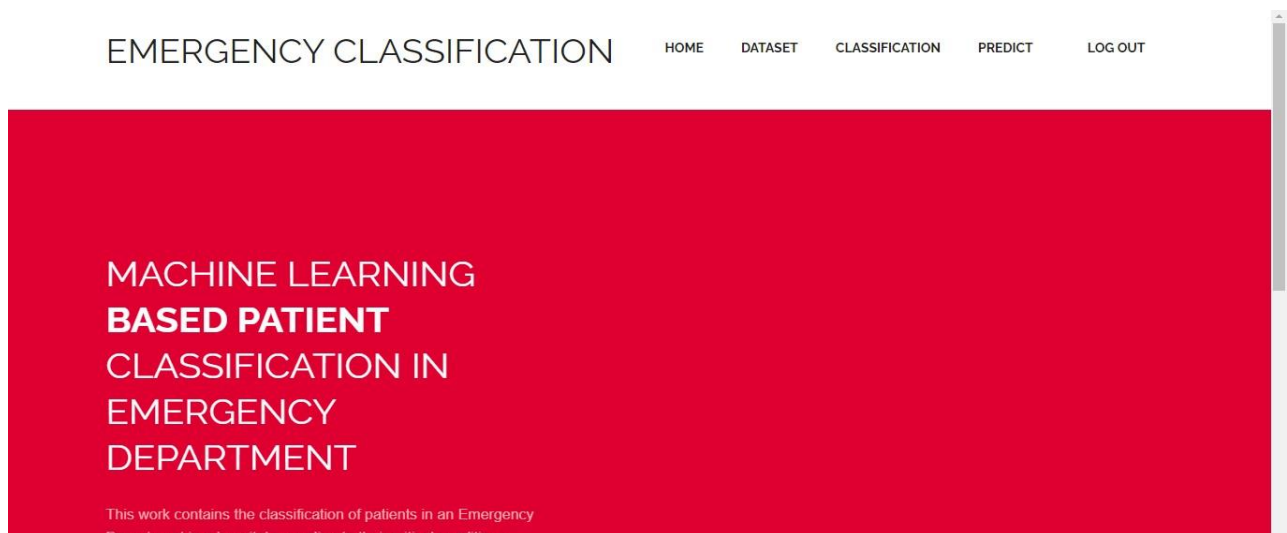


Fig 10.7 User Home

Dataset view

EMERGENCY CLASSIFICATION										HOME	DATASET	CLASSIFICATION	PREDICT	LOG OUT
Age	Gender	Pulse	SystolicBloodPressure	DiastolicBloodPressure	RespiratoryRate	SPO2	RandomBloodSugar	Temperature	Status					
59	1	95	82	131	24	95.87	92	101.8	0					
61	1	62	87	140	16	93.49	155	100.0	1					
82	0	73	120	126	30	95.65	80	103.2	1					
57	1	56	124	103	26	96.28	162	101.3	1					
61	0	135	123	90	10	95.56	133	99.3	1					
28	0	143	186	104	17	98.30	163	98.4	1					
4	0	91	188	105	13	94.37	82	100.1	0					
37	0	56	198	119	24	95.16	146	102.5	1					
78	0	48	187	56	13	98.72	103	104.0	0					
28	1	123	183	49	15	98.92	104	100.8	1					
36	1	104	192	78	30	97.33	110	98.0	1					

Fig 10.8 Dataset View

Machine Learning Results

EMERGENCY CLASSIFICATION					HOME	USERS	MACHINE LEARNING	LOG OUT
MACHINE LEARNING RESULTS								
Metrics					Metrics			
Random Forest					Decision Tree			
	precision	recall	f1-score	support		precision	recall	f1-score
0	0.933333	0.784483	0.852459	232.000000	0	0.758772	0.748890	0.762174
1	0.913844	0.978015	0.943800	542.000000	1	0.891941	0.898524	0.895221
accuracy	0.918905	0.918905	0.918905	0.918905	accuracy	0.882713	0.852713	0.852713
macro avg	0.923489	0.880249	0.898130	774.000000	macro avg	0.825357	0.822107	0.823997
weighted avg	0.910540	0.910550	0.910421	774.000000	weighted avg	0.852050	0.852713	0.853344
Naive Bayes					K Neighbors Classifier			
	precision	recall	f1-score	support		precision	recall	f1-score
0	1.000000	0.719828	0.837093	232.000000	0	0.902222	0.875000	0.888403
1	0.882910	1.000000	0.943429	542.000000	1	0.947177	0.959410	0.953254
accuracy	0.918021	0.918021	0.918021	0.918021	accuracy	0.934109	0.934109	0.934109
macro avg	0.940458	0.859914	0.890261	774.000000	macro avg	0.924099	0.917255	0.920828
weighted avg	0.925014	0.918021	0.911556	774.000000	weighted avg	0.933702	0.934109	0.933815
Logistic Regression					Support Vector Machine			

Fig 10.9 Machine Learning Results

Prediction page

EMERGENCY CLASSIFICATION

HOME DATASET CLASSIFICATION PREDICT LOG OUT

MACHINE LEARNING PREDICTION FORM

Lable	Input
Age	<input type="text"/>
Gender	<input type="text" value="--Select Gender--"/>
Pulse	<input type="text"/>
Systolic Blood Pressure	<input type="text"/>
Diastolic Blood Pressure	<input type="text"/>
RespiratoryRate	<input type="text"/>
SP02	<input type="text"/>
RandomBloodSugar	<input type="text"/>
Temperature	<input type="text"/>
The Emergency:	<input type="button" value="submit"/>




Fig 10.10 Prediction Page

Results:

Result 1: Emergency needed

EMERGENCY CLASSIFICATION

HOME DATASET CLASSIFICATION PREDICT LOG OUT

MACHINE LEARNING PREDICTION FORM

Lable	Input
Age	<input type="text"/>
Gender	<input type="text" value="-Select Gender--"/>
Pulse	<input type="text"/>
Systolic Blood Pressure	<input type="text"/>
Diastolic Blood Pressure	<input type="text"/>
RespiratoryRate	<input type="text"/>
SPO2	<input type="text"/>
RandomBloodSugar	<input type="text"/>
Temperature	<input type="text"/>
The Emergency: Needed	<input type="button" value="submit"/>




Fig 10.11 Result-1

Result 2: Emergency NOT needed

EMERGENCY CLASSIFICATION

HOME DATASET CLASSIFICATION PREDICT LOG OUT

MACHINE LEARNING PREDICTION FORM

Label	Input
Age	<input type="text"/>
Gender	--Select Gender--
Pulse	<input type="text"/>
Systolic Blood Pressure	<input type="text"/>
Diastolic Blood Pressure	<input type="text"/>
RespiratoryRate	<input type="text"/>
SPO2	<input type="text"/>
RandomBloodSugar	<input type="text"/>
Temperature	<input type="text"/>
The Emergency: not Needed	<input type="button" value="submit"/>




Fig 10.12 Result-2

11. SYSTEM TEST

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

TYPES OF TESTS

I. Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

II. Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfactory, as shown by successful unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

III. Functional test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures : interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

IV. System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

V. White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

VI. Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

VII. Unit Testing

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

VIII. Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

IX. Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

12. CONCLUSION

The healthcare industry can be benefited from machine learning especially in classification of patients condition. The algorithms proved that it is more useful in classifying the risk level of patients' conditions in triage as critical or non_critical. This system would help in reducing time delay to classify patients at triage in the Emergency Department of a hospital. This work proved that for the unbalanced Triage Vital Dataset, Decision Tree experimentally verified F1-score of 77.67 with a high specificity of 97.18. Moreover, this system can be useful in a pandemic situation in which all the resources are exhausted that happened during the Covid 19. This will maximize the efficiency of the health infrastructure and if the health industry takes this approach, it will reduce the workload of doctors and allow them to provide proper treatment to patients as soon as possible. Finally, it can be stated that the proposed system will benefit both doctors and patients.

13. BIBLIOGRAPHY

- [1] Tintinalli, Judith E, “Disaster Preparedness”, Tintinalli’s Emergency Medicine: A Comprehensive Study Guide, 9th Edition, McGraw-Hill Education, 2019, ISBN: 1260019934.
- [2] D. Sharathchandra and M. R. Ram, ”ML Based Interactive Disease Prediction Model,” 2022 IEEE Delhi Section Conference (DELCON), 2022, pp. 1-5, doi: 10.1109/DELCON54057.2022.9752947.
- [3] R. Krishnamoorthi, S. Joshi, H.Z. Almarzouki, P.K. Shukla, A. Rizwan, C. Kalpana, B. Tiwari, “A Novel Diabetes Healthcare Disease Prediction Framework Using Machine Learning Techniques”, Journal of healthcare engineering, 1684017, Jan 2022 <https://doi.org/10.1155/2022/1684017>
- [4] D.A Debal, T.M. Sitote, “Chronic kidney disease prediction using machine learning techniques”, Journal of Big Data 9, Nov 2022, 10.1186/s40537-022-00657-5.
- [5] K.M. Almustafa, “Prediction of chronic kidney disease using different classification algorithms”, Informatics in Medicine Unlocked, 2021, Volume 24, 100631, ISSN 2352-9148, <https://doi.org/10.1016/j.imu.2021.100631>
- [6] A. R. Rao and B. S. Renuka, ”A Machine Learning Approach to Predict Thyroid Disease at Early Stages of Diagnosis,” 2020 IEEE International Conference for Innovation in Technology (INOCON), 2020, pp. 1-4, doi: 10.1109/INOCON50539.2020.9298252.
- [7] T. Nibareke, J. Laassiri, “Using Big Data-machine learning models for diabetes prediction and flight delays analytics Journal of Big Data, 2020, 7, pp1-18 10.1186/s40537-020-00355-0
- [8] S. Uddin, A. Khan, M.E. Hossain, M.A. Moni, “Comparing different supervised machine learning algorithms for disease prediction” BMC medical informatics and decision making, Dec 2019, 19(1), 281, doi:10.1186/s12911-019-1004-8
- [9] T. T. Han, H. Y. Pham, D. S. L. Nguyen, Y. Iwata, T. T. Do, K. Ishibashi, G. Sun, “Machine learning based classification model for screening of infected patients using vital signs”, Informatics in Medicine Unlocked, Volume 24, 2021, 100592, ISSN 2352-9148, <https://doi.org/10.1016/j.imu.2021.100592>.

- [10] M. Deepika and K. Kalaiselvi, "A Empirical study on Disease Diagnosis using Data Mining Techniques", 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), 2018, pp. 615-620, doi: 10.1109/ICICCT.2018.8473185.
- [11] A. Mir and S. N. Dhage, "Diabetes Disease Prediction Using Machine Learning on Big Data of Healthcare," 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), 2018, pp. 1-6, doi: 10.1109/ICCUBEA.2018.8697439.
- [12] E. A. Choi et al. "Prediction of COPD severity based on clinical data using Machine Learning," 2021 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), 2021, pp. 1646-1648, doi: 10.1109/BIBM52615.2021.9669887, ET-SIP-2254415.2022.9791739