



Website for predication of diabetes using machine learing

**Supervision by:
DR. Alaa Allakany
Prof.DR .Mohamed El-Ghanam**

**Bio-Informatics department
Faculty of computer & information
Kafr el-sheikh University**

2022-2023

Team member

- 1- Moataz Mohamed**
- 2- Mahmoud Salah**
- 3- Abdelaziz Mahmoud**
- 4- Bassant Adel**
- 5- Noha AbdulHawahab**
- 6- Haidy Sobhi**
- 7- Mirna Mahmoud**
- 8- Gehad Mahmoud**

Acknowledgment:

First of all, alhamdulillah for everything. We can never thank Allah enough for the countless blessings He blessed us with. And we would take this opportunity to express our sincere thanks and gratitude to our **DR. Alaa Allakany** for her vital role, support and guidance in completing this project.

We would also like to extend our gratitude and thanks to **Eng.Manar El-Hoseny** Who helped and made a double effort to help us complete this project

We must thank prof.Mohamed Elghanam for his effort

Also, we do not want to forget to pray to Allah to have mercy on Dr. Osama Abu Zaid

Last but not least, I thank my family as they give us support and love all the way and I could have never reached this far without their support and encouragement

Abstract

Diabetes was one of the first diseases described, with an Egyptian manuscript from c. 1500 BCE mentioning "too great emptying of the urine." The Ebers papyrus includes a recommendation for a drink to take in such cases. The first described cases are believed to have been type 1 diabetes. Indian physicians around the same time identified the disease and classified it as madhumeha or "honey urine", noting the urine would attract ants.

The term "diabetes" or "to pass through" was first used in 230 BCE by the Greek Apollonius of Memphis. The disease was considered rare during the time of the Roman empire, with Galen commenting he had only seen two cases during his career. This is possibly due to the diet and lifestyle of the ancients, or because the clinical symptoms were observed during the advanced stage of the disease. Galen named the disease "diarrhea of the urine" (diarrhea urinosa).

In 2017, 425 million people had diabetes worldwide, up from an estimated 382 million people in 2013 and from 108 million in 1980. Accounting for the shifting age structure of the global population, the prevalence of diabetes is 8.8% among adults, nearly double the rate of 4.7% in 1980. Type 2 makes up about 90% of the cases. Some data indicate rates are roughly equal in women and men, but male excess in diabetes has been found in many populations with higher type 2 incidence, possibly due to sex-related differences in insulin sensitivity, consequences of obesity and regional body fat deposition, and other contributing factors such as high blood pressure, tobacco smoking, and alcohol intake.

The WHO estimates that diabetes resulted in 1.5 million deaths in 2012, making it the 8th leading cause of death. However another 2.2 million deaths worldwide were attributable to high blood glucose and the increased risks of cardiovascular disease and other associated complications (e.g. kidney failure), which often lead to premature death and are often listed as the underlying cause on death certificates rather than diabetes.

We design a system to detect diabetes with high accuracy using machine learning and artificial intelligence that require enter a list of information such as Pregnancies rate, Glucose rate, Blood Pressure rate, Skin Thickness rate, Insulin rate, BMI, Diabetes Pedigree Function and Age.

Catalog

Chapter1 : Introduction.....	7
1.1 Purpose.....	7
1.2 scope	7
1.3 Intended audience.....	7
1.4 Intended use	7
Chapter 2 : System Requirment and Analsis.....	9
2.1 Machine Learning.....	9
2.2 user need.....	13
2.3 Dependencies and assumptions	13
2.4 system context	14
Chapter3:Software design	16
3.1 introduction.....	17
3.2 UML.....	17
Chapter 4 :Used software	20
4.1 introduction.....	21
4.2Ml model.....	23
4.3Explain full model.....	47
4.4 Analysis & Design	52
4.5 Implementation.....	51
Chapter 5: Database	53
Chapter 6:UI	64
CHAPTER 7:SYSTEM TESTING	83
7.1 introduction.....	84
7.2Strategic approach to software testing.....	84
7.3Steps of system testing.....	85
Chapter 8:System Security	92
CHAPTER 9:CONCLUSION	101
9.1 conclusion	102
9.2future work	102
Chapter 10: References	103

List of figures

Fig (1) data set	24
Fig(2)overview data set.....	28
Figs(3,4,5,6,7,8,9) graphs of out puts.....	34,35,36,37
Fig(10) glucose vs blood pressure.....	38
Fig (11) glucose vs age.....	40
Fig (12)BMI vs age.....	41
Fig (13)skin thickness DPF	42
Fig (14) confusion matrix.....	44
Fig(15) Feature importance of variables.....	46
Fig (16) stream-lit app with plotly chart	64

CHAPTER1 : INTRODUCRION

1.1 Purpose

The purpose of predicting diabetes is to identify individuals who are at risk of developing the condition or who may already have un-diagnosed diabetes. By predicting diabetes, healthcare professionals can intervene early, provide appropriate medical care, and implement preventive measures to manage the disease effectively.

1.2 Scope for project

Predictive models for diabetes can be based on various factors such as age, gender, family history, lifestyle habits, and medical data like blood glucose levels, body mass index (BMI), and blood pressure. These models use statistical analysis and machine learning algorithms to assess the likelihood of an individual developing diabetes within a certain time-frame.

The prediction of diabetes serves several important purposes:

- ✧ Early detection
- ✧ Prevention and lifestyle interventions
- ✧ Personalized medicine
- ✧ Public health planning

1.3Intended audience

1. Healthcare Professionals
2. Public Health Officials
3. Researchers
4. Patients and Individuals
5. Healthcare Administrators and Policy Makers

1.4 Intended Use

1. Early Detection and Diagnosis
2. Personalized Risk Assessment
3. Preventive Interventions
4. Resource Allocation
5. Research and Development
6. Public Health Planning

CHAPTER 2 : SYSTEM REQUIREMENT AND ANALYSIS

2.Overall Description

2.1.1-Machine learning

is the process of using mathematical model of data to help a computer learn without direct instruction. it's considered a subset of artificial intelligence. machine learning uses algorithms to identify patterns within data and those patterns are then used to create a data model that can make predictions. with increased data and experience the results of machine learning are more accurate much like how humans improve with more practice.

The adaptability of machine learning makes it a great choice in scenarios where the data always changing the nature of the request or task are always shifting or coding a solution would be effectively impossible

Machine learning techniques:

-Supervised learning

Addressing datasets with labels or structure, data acts as a teacher and “trains” the machine, increasing in its ability to make a prediction or decision.

-Unsupervised learning

Addressing datasets without any labels or structure, finding patterns and relationships by grouping data into clusters.

-Reinforcement learning

Replacing the human operator, an agent—a computer program acting on behalf of someone or something—helps determine outcome based upon a feedback loop.

How machine learning works to solve problems

Step 1: Collect and prepare the data

Step 2: Train the model

Step 3: Validate the model

Step 4: Interpret the results

Gradient descent (GD) algorithm

is an iterative first-order optimization algorithm used to find a local minimum/maximum of a given function. This method is commonly used in machine learning (ML) and deep learning (DL) to minimize a cost/loss function (e.g. in a linear regression)

Randomized algorithm:

is a technique that uses a source of randomness as part of its logic. It is typically used to reduce either the running time, or time complexity; or the memory used, or space complexity, in a standard algorithm. The algorithm works by generating a random number, r , within a specified range of numbers, and making decisions based on r 's value.

2.1.2- Cloud computing (using Co-lab)

Cloud computing is a very important technology that makes us ready to do our work on the cloud using servers and tools of remote distance companies that provide high quality and speed in programming and using applications that are linked with the cloud.

Cloud in our project:

- machine learning work is done in Google colab that provides us with more ram memory and GPUs than normal PCs to make it faster and easier for training and testing our model.
- Colab also provides us with libraries.
- We used Kaggle website and Google Colab for choosing the dataset and doing the training and testing and we found a lot of datasets for diabetes and we chose one that's well organized and easy to handle.

2.1.3- Analysis model

The model that is basically being followed is the **WATERFALL MODEL**, which states that the phases are organized in a linear order. First of all the feasibility study is done. The design starts after the requirement analysis is complete and the coding begins after the design is complete. Once the programming is completed, the testing is done.

In this model the sequence of activities performed in a software development Project are: -Requirement Analysis

- Project Planning
- System design
- Coding
- Unit testing
- System integration & testing

WATERFALL MODEL was chosen because all requirements were known beforehand and the objective of our software development is the computerization/automation of an already existing manual working system.

2.1.4-Hardware specifications

4.1-Hardware requirements:

- 2.8 GHz Processor or more
- 8GB RAM or more
- SSD M2 128 GB Hard Disk Space or more

4.2- Software requirements:

- Windows OS (8/10)
- Python 3.7 or above
- SQLite
- pycharm
- numpy library
- pandas library
- seaborn library
- matplotlib library
- sklearn library
- gradientboosting classifier library
- StandardScaler library
- Train_test_split library
- KFold library
- Accuracy_score library
- Confusion_matrix library
- Mlxtend library
- Classification_report library

2.1.5-Accuracy and Cost Function:

we achieved not so good accuracy (90%), but over time we improved the model, and the accuracy increased until we achieved accuracy of almost 99% percent

2.1.6- Software requirement specification

Purpose: The main purpose is to determine the operating characteristics of the system.

Scope: This Document plays a vital role in the development life cycle (SDLC) and it describes the complete requirement of the system. It is meant to be used by the developers and will be the basic during the testing phase. Any changes made to the requirements in the future will have to go through a formal change approval process.

2.1.6.2-Error avoidance

At this stage care is to be taken to ensure that input data remains accurate from the stage at which it is recorded up to the stage in which the data is accepted by the system. This can be achieved only by means of careful control each time the data is handled.

2.1.6.3-Error detection

Even though every effort is made to avoid the occurrence of errors, still a small proportion of errors is always likely to occur, these types of errors can be discovered by using validations to check the input data.

2.1.6.4-Data validation

Procedures are designed to detect errors in data at a lower level of detail. Data validations have been included in the system in almost every area where there is possibility for the user to commit errors. The system is designed to be a user friendly one. In other words the system has been designed to communicate effectively with the user.

2.1.6.5-User interface design

It is essential to consult the system users and discuss their needs while designing the user interface.

2.1.6.6-Performance requirements

Performance is measured in terms of the output provided by the application. Requirement specification plays an important part in the analysis of a system. Only when the requirement specifications are properly given, it is possible to design a system, which will fit into the required environment. It rests largely on the part of the users of the existing system to give the requirement specifications because they are the people who finally use the system. This is because the requirements have to be known during the initial stages so that the system can be designed according to those requirements.

2.1.6.7- Input and output

The main inputs, outputs and major functions of the system are as follows

Inputs:

- 1- User enters his or her user id and name for login.
- 2- User enters:
 - Pregnancies rate
 - Glucose rate
 - Blood Pressure rate
 - Skin Thickness rate
 - Insulin rate
 - BMI rate

- Diabetes Pedigree Function rate
- 3-User requests for prediction.
- 4-

Outputs:

Whether has diabetes or no

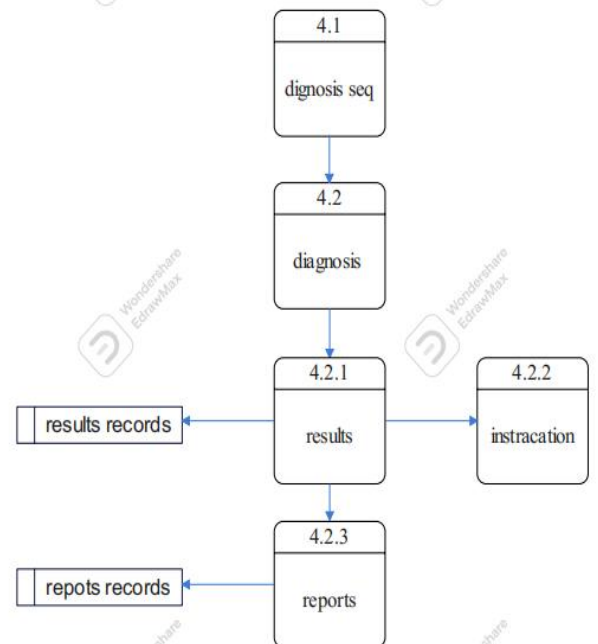
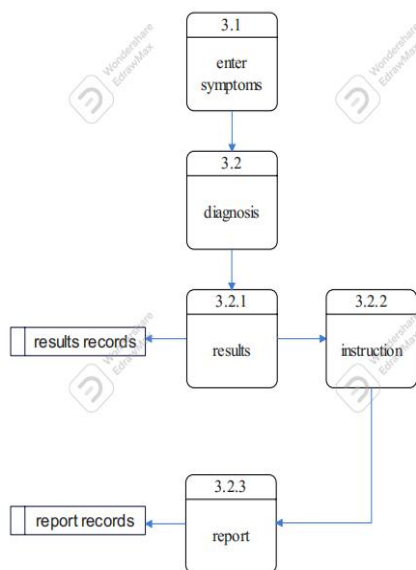
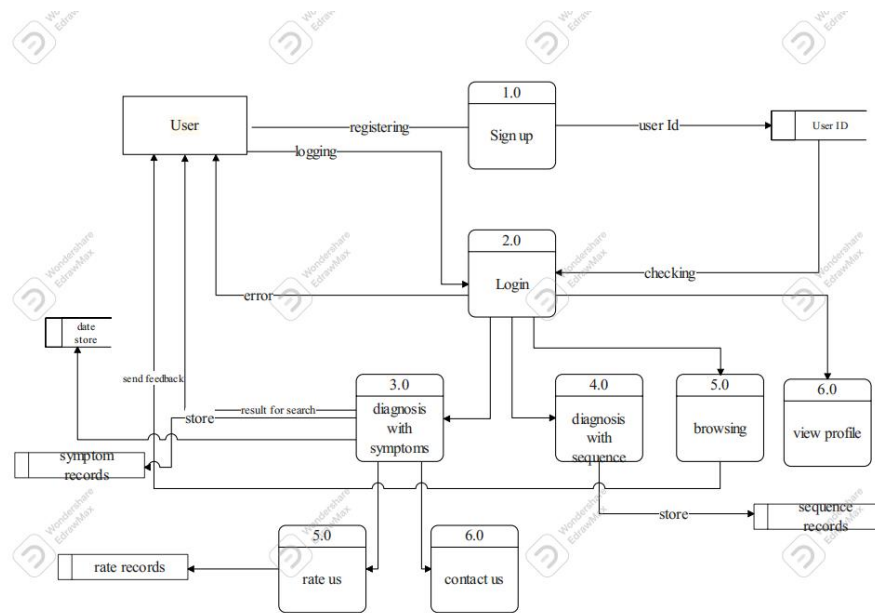
2.2user needs

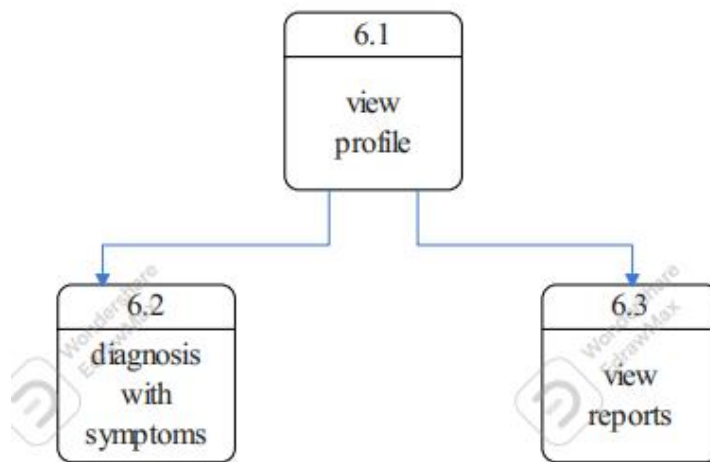
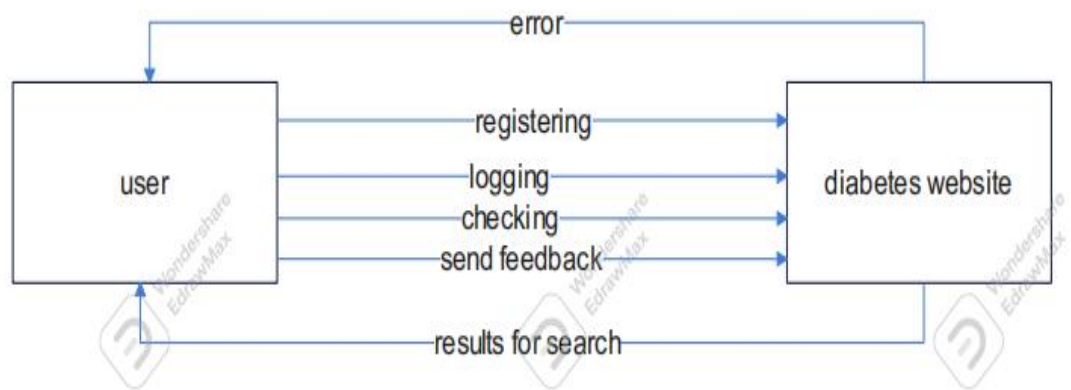
1. Healthcare Professionals:
 - Accurate risk assessment
 - Early detection:
 - Decision support:
 - Integration with clinical workflow:
2. Individuals/Patients:
 - Personalized risk assessment:
 - Empowerment and education:
 - Clear communication:
 - Privacy and data security:
2. Public Health Officials:
 - Population-level risk assessment
 - Resource allocation
 - Evaluation of interventions
 - Data for research
4. Researchers:
 - Access to data
 - Collaboration and knowledge sharing
 - Transparent and reproducible methodology

2.3Dependencies and assumptions

1. Data Availability
2. Data Integrity and Quality
3. Representative Sample
4. Risk Factor Relevance
5. Statistical Assumptions
6. Generalizability
7. Assumption of Causality
8. Ethical Considerations

2.4 SYSTEM CONTEXT





Chapter3:Software design

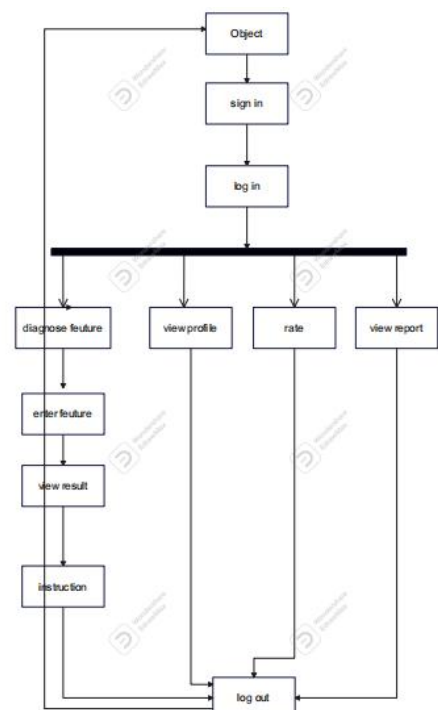
3.1 Introduction

Software design sits at the technical kernel of the software engineering process and is applied regardless of the development paradigm and area of application. The designer's goal is to produce a model or representation of an entity that will later be built. Beginning, once system requirement have been specified and analyzed, system design is the first of the three technical activities design, code and test that is required to build and verify software. The importance can be stated with a single word "Quality". Design is the only way that we can accurately translate a customer's view into a finished software product or system. Without a strong design we risk building an unstable system one that will be difficult to test, one whose quality cannot be assessed until the last stage. During design, progressive refinement of data structure, program structure, and procedural details are developed reviewed and documented. System design can be viewed from either technical or project management perspective. From the technical point of view, design is comprised of four activities architectural design, data structure design, interface design and procedural design.

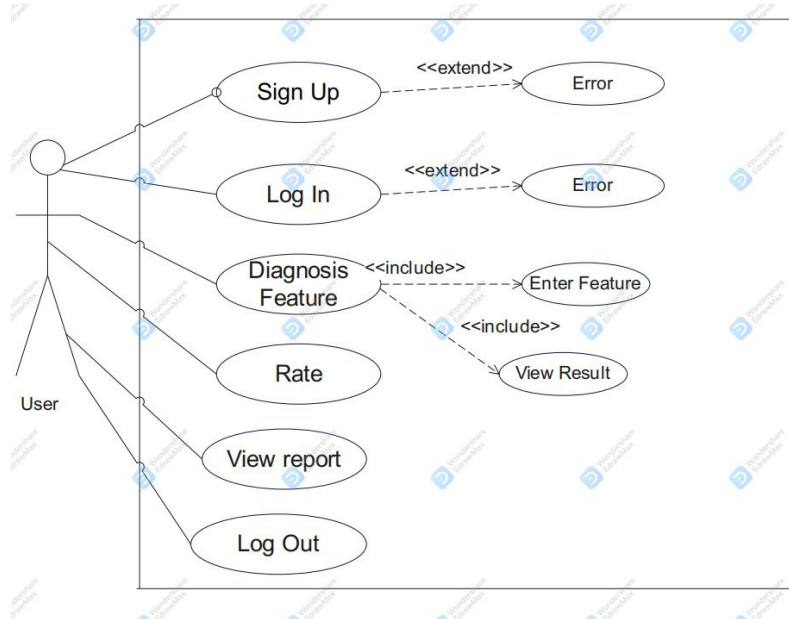
3.2 UML

- Behavioral UML Diagram
 - State Diagram
 - Communication Diagram
 - Use Case Diagram
 - Activity Diagram
 - Sequence Diagram
- Structural UML Diagram
 - Class Diagram

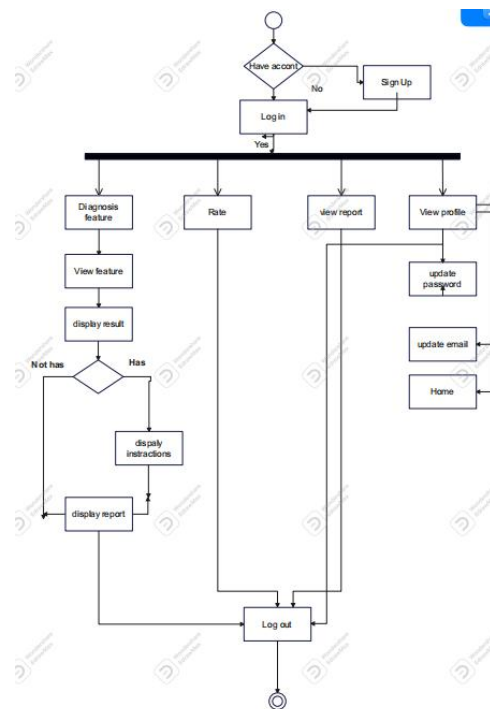
➤ State Diagram



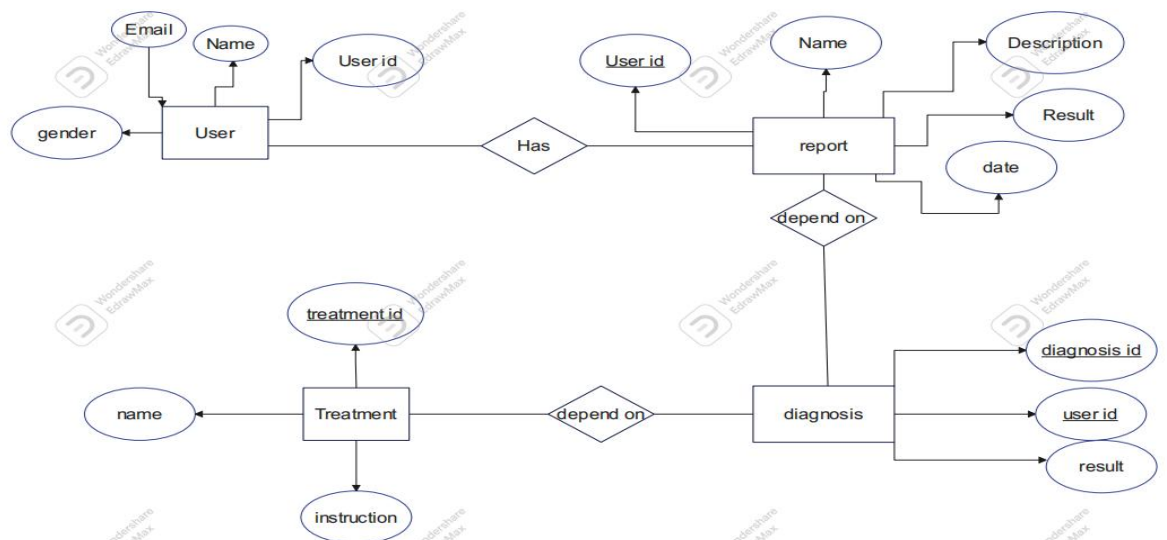
➤ Use Case Diagram



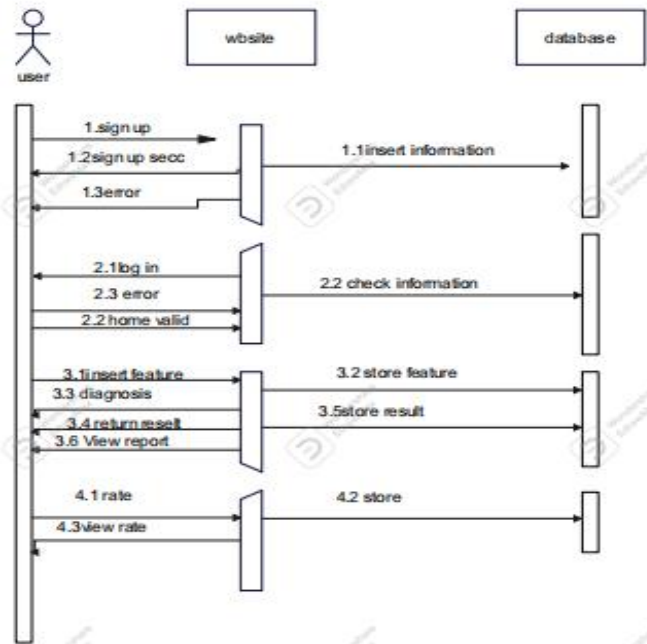
➤ Activity Diagram



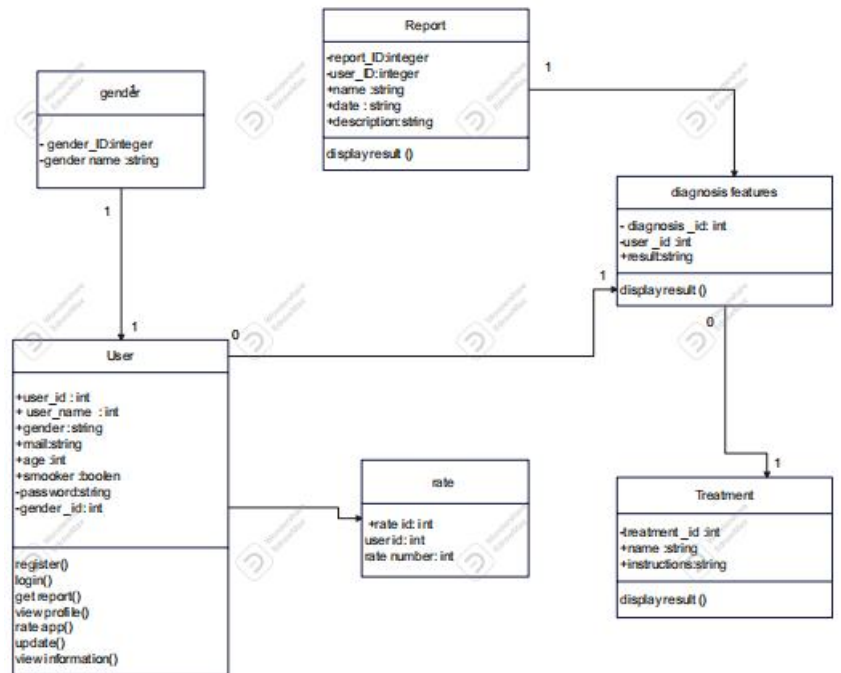
➤ ERD



➤ Sequence Diagram



➤ Class Diagram



CHAPTER 4 :USED SOFTWARE

4.1 Introduction

project management is crucial for the success of a prediction of diabetes project. Here are some key aspects to consider for project management:

- ◆ Project Planning
 - ◆ Team Composition and Roles
 - ◆ Data Management
 - ◆ Risk Management
 - ◆ Iterative Development
 - ◆ Documentation
 - ◆ Communication and Collaboration
 - ◆ Quality Assurance
 - ◆ Stakeholder Engagement
 - ◆ Project Evaluation and Continuous Improvement
-
- ◆ Project Planning: Define clear project objectives, scope, and deliverables. Break down the project into manageable tasks and create a project schedule with realistic timelines. Identify the key stakeholders and establish communication channels to ensure effective collaboration and coordination.
 - ◆ Team Composition and Roles: Form a multidisciplinary team with members having expertise in data analysis, machine learning, healthcare domain knowledge, and software development. Clearly define team roles and responsibilities, ensuring that each team member understands their tasks and contributions to the project.
 - ◆ Data Management: Establish data management protocols to ensure data quality, privacy, and security. Determine data sources, data collection methods, and establish procedures for data cleaning, pre-processing, and storage. Adhere to relevant regulations and ethical guidelines for handling sensitive health data.
 - ◆ Risk Management: Identify potential risks and develop strategies to mitigate them. Assess risks related to data quality, model performance, resource availability, and technical challenges. Regularly monitor and evaluate risks throughout the project lifecycle and update risk mitigation strategies as needed.

- ◆ **Iterative Development:** Adopt an iterative approach to model development, evaluation, and refinement. Break down the project into smaller iterations or sprints, allowing for continuous feedback and adjustments. Regularly review and validate the prediction model's performance against predefined metrics and incorporate feedback from stakeholders.
- ◆ **Documentation:** Maintain comprehensive documentation of the project, including project plans, data sources, data preprocessing steps, model development methodologies, and evaluation results. Document decisions made throughout the project, as well as code documentation, to ensure reproducibility and facilitate knowledge transfer.
- ◆ **Communication and Collaboration:** Establish effective communication channels and regular project meetings to keep all team members informed about progress, challenges, and next steps. Foster collaboration among team members, encourage knowledge sharing, and facilitate cross-functional discussions.
- ◆ **Quality Assurance:** Implement quality assurance practices to ensure the accuracy and reliability of the prediction model. Conduct rigorous testing and validation of the model, including performance evaluation on independent datasets and comparing with existing benchmarks or expert opinions.
- ◆ **Stakeholder Engagement:** Engage with stakeholders, including healthcare professionals, patients, researchers, and policymakers, throughout the project. Gather feedback, incorporate their perspectives, and involve them in the validation and evaluation of the prediction model to ensure its relevance and usefulness.
- ◆ **Project Evaluation and Continuous Improvement:** Regularly assess the project's progress, performance, and adherence to project objectives. Seek feedback from stakeholders and evaluate the project's impact on clinical outcomes or public health. Use lessons learned to refine the prediction model, update documentation, and improve future iterations or follow-up projects.

By implementing strong project management practices, the prediction of diabetes project can be effectively executed, delivering a high-quality and valuable solution for diabetes prediction and management.

4.2 Machine learning model

1.Import some necessary libraries for data analysis and plotting. The code includes the following libraries:

- numpy: a library for mathematical and scientific operations in Python.
 - pandas: a library for managing and analyzing data in Python.
 - seaborn: a library for plotting beautiful and complex visualizations in Python.
 - matplotlib: a library for plotting simple and basic visualizations in Python.
- warnings are disabled using the "warnings.filterwarnings('ignore')" property to avoid any warning messages from appearing on the screen.

2.Imports several machine learning libraries used for building and evaluating a classification model. The code includes the following libraries:

- Sklearn: a popular machine learning library that provides a wide range of algorithms and tools for data preprocessing, feature selection, model selection, and evaluation.
- GradientBoostingClassifier: a machine learning algorithm that builds an ensemble of decision trees iteratively to improve the model's performance.
- StandardScaler: a preprocessing technique that scales the input features to have zero mean and unit variance. This is often used to standardize the input data before feeding it to a machine learning model.
- Train_test_split: a function that splits the data into training and testing sets. The training set is used to train the model, while the testing set is used to evaluate its performance.
- KFold: a cross-validation technique that splits the data into k folds and trains the model k times, each time using a different fold as the testing set and the remaining folds as the training set. This helps to reduce overfitting and obtain a more reliable estimate of the model's performance.
- Accuracy_score: a metric that measures the proportion of correctly classified instances out of the total number of instances.
- Confusion_matrix: a matrix that shows the number of true positives, false positives, true negatives, and false negatives for a binary classification problem.
- Mlxtend: a library that provides additional visualization tools for machine learning models, including a function for plotting confusion matrices.

- `Classification_report`: a function that generates a report that includes several metrics, such as precision, recall, and F1 score, for each class in a multi-class classification problem.

By importing these libraries, the programmer can leverage their functionality to preprocess the data, train and evaluate a classification model, and generate reports and visualizations to interpret the model's performance.

3. Sets the default figure size for plots created using the matplotlib library. The code first retrieves the current figure size using `"plt.rcParams['figure.figsize']"` and stores it in a variable called `"fig_size"`. The default figure size is usually (6.4, 4.8) inches.

The next two lines of code set the width and height of the figure to 8 and 6 inches, respectively, by modifying the elements of the `"fig_size"` list.

Finally, the new figure size is set globally using `"plt.rcParams['figure.figsize'] = fig_size"`, which updates the default size for all subsequent plots created using Matplotlib.

By setting the figure size, the programmer can control the dimensions of the plot, which can help to make it more readable and visually appealing. A larger figure size may also be necessary to include more details or to visualize complex relationships between variables.

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

4. Read a CSV file containing data related to diabetes and stores it in a pandas DataFrame called `"df"`. The code uses the `"read_csv"` function from the pandas library to read the CSV file located at

`"C:\\Users\\FreeComp\\OneDrive\\Desktop\\graduation\\diabetes.csv"` and convert it into a DataFrame.

The `"head"` method is then called on the DataFrame to display the first few rows of the data. This is a useful way to quickly check that the data has been read in correctly and to get a sense of its structure and contents.

By storing the data in a DataFrame, the programmer can perform various data manipulation and analysis tasks using pandas and other related libraries. For example, the DataFrame can be used to filter, sort, group, and aggregate the data, as well as to visualize it using various plotting libraries.

5.Retrieves the shape of the pandas DataFrame "df" created in the previous code snippet. The code uses the "shape" attribute of the DataFrame, which returns a tuple representing the dimensions of the DataFrame.

The tuple contains two elements: the number of rows and the number of columns in the DataFrame. The number of rows corresponds to the number of instances or observations in the dataset, while the number of columns corresponds to the number of features or variables.

For example, the code "df.shape" may return a tuple like (768, 9), indicating that the DataFrame contains 768 rows and 9 columns. This means that there are 768 instances in the dataset, each with 9 features or variables.

By retrieving the shape of the DataFrame, the programmer can get a sense of the size and structure of the dataset, which can be useful for data preprocessing, model selection, and evaluation. For instance, knowing the number of features can help the programmer to determine which features are relevant and which ones can be dropped or transformed.

6.Displays information about the pandas DataFrame "df" created in the previous code snippet. The code uses the "info" method of the DataFrame, which provides a concise summary of the DataFrame's structure and contents.

The summary includes the following information:

- The number of non-null values and the data type of each column
- The memory usage of the DataFrame
- An optional description of the DataFrame, if provided

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 768 entries, 0 to 767
```

```
Data columns (total 9 columns):
```

```
Pregnancies          768 non-null int64
```

```
Glucose              768 non-null int64
```

```
BloodPressure       768 non-null int64
```

```
SkinThickness       768 non-null int64
```

```
Insulin              768 non-null int64
```

```
BMI                  768 non-null float64
```

```
DiabetesPedigreeFunction 768 non-null float64
```

```
Age                  768 non-null int64
```

```
Outcome              768 non-null int64
```

```
dtypes: float64(2), int64(7)
```

memory usage: 54.1 KB

...

This output shows that the DataFrame has 768 rows and 9 columns, with column names such as "Pregnancies", "Glucose", "BloodPressure", etc. It also shows that all columns have 768 non-null values, indicating that there are no missing values in the dataset. The memory usage of the DataFrame is also displayed, along with the data types of each column (e.g., int64,

float64). Finally, it shows that the DataFrame has two float columns and seven integer columns.

By using the "info" method, the programmer can quickly check the data types and completeness of the data, which can help to identify any potential data quality issues and to determine the appropriate data preprocessing steps.

7. Displays descriptive statistics of the pandas DataFrame "df" created in the previous code snippet. The code uses the "describe" method of the DataFrame, which provides a summary of the central tendency, dispersion, and shape of the distribution of each numerical column in the DataFrame. The summary includes the following statistics:

- count: the number of non-missing values in the column
- mean: the arithmetic mean of the values in the column
- std: the standard deviation of the values in the column
- min: the minimum value in the column
- 25%: the 25th percentile of the values in the column
- 50%: the median or 50th percentile of the values in the column
- 75%: the 75th percentile of the values in the column

This output shows the summary statistics for each numerical column in the DataFrame, including the count, mean, standard deviation, and various percentiles. For example, the "Glucose" column has a mean of 120.89, a standard deviation of 31.97, and a median of 117.00.

By using the "describe" method, the programmer can quickly get a sense of the distribution of each numerical column in the DataFrame, which can help to identify any potential outliers, skewness, or other data quality issues.

8. Replaces all the 0 values in the specified columns of the pandas DataFrame "df" with NaN or missing values. The code uses the "replace" method of the DataFrame to replace 0 values in the columns "Glucose", "BloodPressure", "SkinThickness", "Insulin", and "BMI" with the NumPy NaN value, which represents missing or undefined data.

The code specifies the columns to be replaced using the DataFrame indexing operator, which selects a subset of the columns based on their column labels. The `".replace(0,np.NaN)"` part of the code specifies the replacement values, where "0" is the value to be replaced and `"np.NaN"` is the replacement value.

The resulting DataFrame is then displayed using the `"head"` method, which shows the first three rows of the modified DataFrame.

By replacing the 0 values with NaN, the programmer can treat them as missing values and handle them accordingly during data preprocessing and analysis. For example, the programmer can use methods such as `"isnull"` or `"fillna"` to identify or impute the missing values, or remove the rows or columns that contain them using the `"dropna"` method. This can help to improve the accuracy and reliability of the analysis and models built on the data.

9. Creates a box plot to visualize the distribution of each variable in the pandas DataFrame `"df"`. The code uses the seaborn library to create the box plot and matplotlib to customize its appearance.

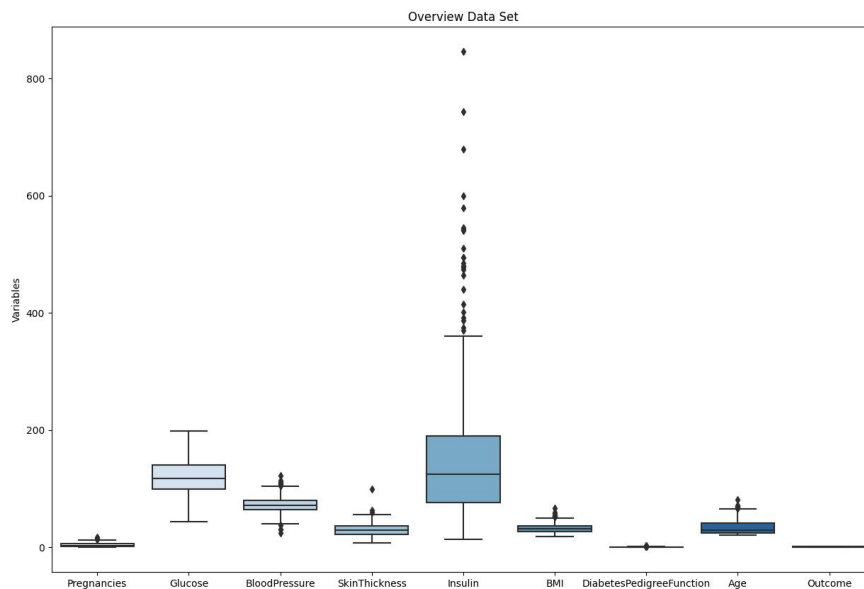
The code creates a new figure with a size of 15 by 10 inches using the `"subplots"` method of matplotlib. It then sets the x-axis limits to `(-0.05, 200)` using the `"set"` method of the axis object. This limits the display range of the x-axis, which can be useful for focusing on specific ranges of the data.

The code sets the y-axis label to `"Variables"` and the plot title to `"Overview Data Set"` using the `"ylabel"` and `"title"` methods of matplotlib.

The code then creates the box plot using the `"boxplot"` method of seaborn. The `"data"` parameter specifies the DataFrame to be used, while the `"orient"` parameter specifies the orientation of the plot as vertical (`'v'`). The `"palette"` parameter specifies the color palette to be used for the plot.

The resulting box plot displays the distribution of each variable in the DataFrame as a box-and-whisker plot. The box represents the interquartile range (IQR), which contains the middle 50% of the data. The whiskers extend to the minimum and maximum values within 1.5 times the IQR, while any points beyond the whiskers are considered outliers and are plotted individually.

By using box plots, the programmer can quickly identify the range, median, outliers, and skewness of each variable in the DataFrame, which can help to guide data preprocessing and modeling decisions.



#Out put

10. Checks for the number of missing values in each column of the pandas DataFrame "df". The code uses the "isnull" method of the DataFrame to create a Boolean mask that indicates whether each element of the DataFrame is missing (True) or not (False). It then uses the "sum" method of the DataFrame to count the number of True values in each column, which corresponds to the number of missing values in each column.

The resulting output is a pandas Series that shows the total number of missing values in each column of the DataFrame.

This output shows that there are missing values in the "Glucose", "BloodPressure", "SkinThickness", "Insulin", and "BMI" columns, with different numbers of missing values in each column. The other columns have no missing values.

By checking for missing values, the programmer can identify which columns and rows have missing data and determine how to handle them during data preprocessing. Common

methods for handling missing values include imputation (replacing missing values with estimated values), deletion (removing rows or columns with missing values), or using specific models that can handle missing data.

11. Defines a function called "median_target" that calculates the mean value of a specified variable in the pandas DataFrame "df" based on the "Outcome" column. The code takes one argument "var", which specifies the name of the variable whose mean is to be calculated.

The code first creates a temporary DataFrame called "temp" by selecting only the rows of "df" where the specified variable is not null, using the "notnull" method of the DataFrame. This filters out any rows where the variable has missing values. The code then selects the specified variable and the "Outcome" column from the temporary DataFrame and groups them by the "Outcome" column using the

"groupby" method of the DataFrame. The resulting grouped DataFrame is then aggregated by taking the mean of the specified variable for each value of the "Outcome" column using the "mean" method of the DataFrame. The resulting DataFrame is then rounded to one decimal place using the "round" method of the DataFrame, and the "reset_index" method is used to reset the index of the DataFrame. The function then returns the resulting DataFrame, which shows the mean value of the specified variable for each value of the "Outcome" column. By defining this function, the programmer can easily calculate the mean (or any other statistic) of any variable in the DataFrame based on the "Outcome" column, which can be useful for identifying any differences in the variable between the two groups defined by the "Outcome" column

12.Calls the previously defined "median_target" function to calculate the mean value of the "Glucose" variable for each value of the "Outcome" column in the pandas DataFrame "df". The second and third lines of the code use the "loc" method of the DataFrame to select the rows where the "Outcome" column is equal to 0 or 1, and the "Glucose" column is null (missing values). The code then assigns the mean value of the "Glucose" variable for the corresponding value of the "Outcome" column to the missing values using the equals sign. In other words, the code imputes missing values in the "Glucose" column with the mean value of the "Glucose" variable for each value of the "Outcome" column. If a missing value belongs to the "Outcome" column value of 0, then it is imputed with the mean value of "Glucose" for the group of "Outcome" equal to 0, which is 110.6. Similarly, if a missing value belongs to the "Outcome" column value of 1, then it is imputed with the mean value of "Glucose" for the group of "Outcome" equal to 1, which is 142.3. Imputing missing values can help to avoid data loss and provide more accurate and complete data for analysis and modeling. However, the imputation method used should be carefully chosen based on the nature and characteristics of the data, as well as any assumptions made about the missing data.

13.Calls the previously defined "median_target" function to calculate the mean value of the "BloodPressure" variable for each value of the "Outcome" column in the pandas DataFrame "df". The second and third lines of the code use the "loc" method of the DataFrame to select the rows where the "Outcome" column is equal to 0 or 1, and the "BloodPressure" column is null

(missing values). The code then assigns the mean value of the "BloodPressure" variable for the corresponding value of the "Outcome" column to the missing values using the equals sign.

In other words, the code imputes missing values in the "BloodPressure" column with the mean value of the "BloodPressure" variable for each value of the "Outcome" column. If a missing value belongs to the "Outcome" column value of 0, then it is imputed with the mean value of "BloodPressure" for the group of "Outcome" equal to 0, which is 70.9. Similarly, if a missing value belongs to the "Outcome" column value of 1, then it is imputed with the mean value of "BloodPressure" for the group of "Outcome" equal to 1, which is 75.3.

Imputing missing values can help to avoid data loss and provide more accurate and complete data for analysis and modeling. However, the imputation method

used should be carefully chosen based on the nature and characteristics of the data, as well as any assumptions made about the missing data.

14. Calls the previously defined "median_target" function to calculate the mean value of the "SkinThickness" variable for each value of the "Outcome" column in the pandas DataFrame "df".

The second and third lines of the code use the "loc" method of the DataFrame to select the rows where the "Outcome" column is equal to 0 or 1, and the "SkinThickness" column is null (missing values). The code then assigns the mean value of the "SkinThickness" variable for the corresponding value of the "Outcome" column to the missing values using the equals sign.

In other words, the code imputes missing values in the "SkinThickness" column with the mean value of the "SkinThickness" variable for each value of the "Outcome" column. If a missing value belongs to the "Outcome" column value of 0, then it is imputed with the mean value of "SkinThickness" for the group of "Outcome" equal to 0, which is 27.2. Similarly, if a missing value belongs to the "Outcome" column value of 1, then it is imputed with the mean value of "SkinThickness" for the group of "Outcome" equal to 1, which is 33.0.

Imputing missing values can help to avoid data loss and provide more accurate and complete data for analysis and modeling. However, the imputation method used should be carefully chosen based on the nature and characteristics of the data, as well as any assumptions made about the missing data. In this case, imputing with the mean value of each group could introduce bias if the missing values are not missing at random (i.e. the missingness is related to some other variable in the dataset).

15.Calls the previously defined "median_target" function to calculate the mean value of the "Insulin" variable for each value of the "Outcome" column in the pandas DataFrame "df".

The second and third lines of the code use the "loc" method of the DataFrame to select the rows where the "Outcome" column is equal to 0 or 1, and the "Insulin" column is null (missing values). The code then assigns the mean value of the "Insulin" variable for the corresponding value of the "Outcome" column to the missing values using the equals sign.

In other words, the code imputes missing values in the "Insulin" column with the mean value of the "Insulin" variable for each value of the "Outcome" column. If a missing value belongs to the "Outcome" column value of 0, then it is imputed with the mean value of "Insulin" for the group of "Outcome" equal to 0, which is 130.3. Similarly, if a missing value belongs to the "Outcome" column value of 1, then it is imputed with the mean value of "Insulin" for the group of "Outcome" equal to 1, which is 206.8.

Imputing missing values can help to avoid data loss and provide more accurate and complete data for analysis and modeling. However, the imputation method used should be carefully chosen based on the nature and characteristics of the data, as well as any assumptions made about the missing data. In this case, imputing with the mean value of each group could introduce bias if the missing values are not missing at random (i.e. the missingness is related to some other variable in the dataset).

16.Calls the previously defined "median_target" function to calculate the mean value of the "BMI" variable for each value of the "Outcome" column in the pandas DataFrame "df".

The second and third lines of the code use the "loc" method of the DataFrame to select the rows where the "Outcome" column is equal to 0 or 1, and the "BMI" column is null (missing values). The code then assigns the mean value of the "BMI" variable for the corresponding value of the "Outcome" column to the missing values using the equals sign.

In other words, the code imputes missing values in the "BMI" column with the mean value of the "BMI" variable for each value of the "Outcome" column. If a missing value belongs to the "Outcome" column value of 0, then it is imputed with the mean value of "BMI" for the group of "Outcome" equal to 0, which is 30.9. Similarly, if a missing value belongs to the "Outcome" column value of 1, then it is imputed with the mean value of "BMI" for the group of "Outcome" equal to 1, which is 35.4.

Imputing missing values can help to avoid data loss and provide more accurate and complete data for analysis and modeling. However, the imputation method used should be carefully chosen based on the nature and characteristics of the data, as well as any assumptions made about the missing data. In this case, imputing with the mean value of each group could introduce bias if the missing values are not missing at random (i.e. the missingness is related to some other variable in the dataset).

17. Creates a histogram plot using Seaborn library in Python. The histogram is based on the "Outcome" variable in a pandas DataFrame "df".

The first line of the code uses Seaborn's "histplot" function to create a histogram plot. It specifies that "Outcome" should be used as the x-axis variable and "Outcome" should be used for the hue variable, which means that the bars in the histogram will be stacked on top of each other based on the value of the "Outcome" variable (0 or 1). The "multiple" parameter is set to 'stack' to make sure that the bars are stacked on top of each other. The "palette" parameter is set to 'Set3' to choose the color palette for the histogram.

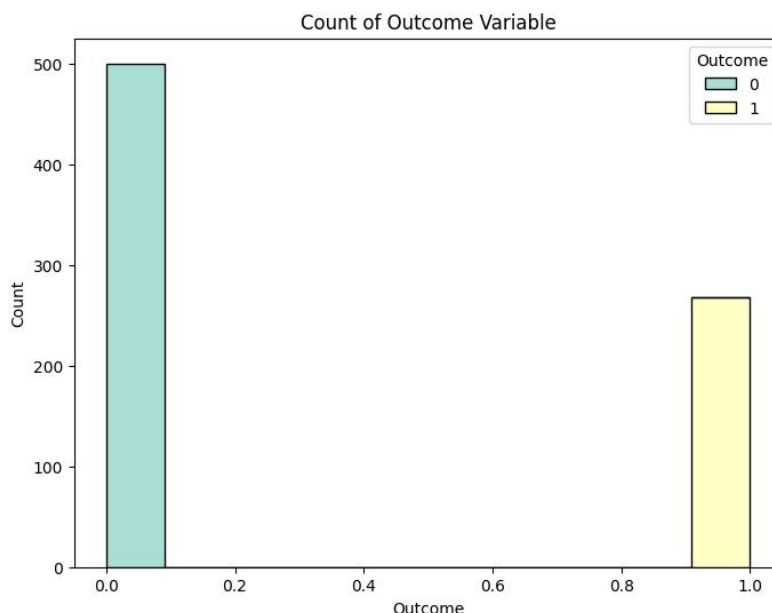
The second line of the code sets the title of the plot to "Count of Outcome Variable" using the "set_title" method of the plot object created in the previous line.

The third line of the code sets the label for the x-axis of the plot to "Outcome" using the "set_xlabel" method of the plot object.

The fourth line of the code sets the label for the y-axis of the plot to "Count" using the "set_ylabel" method of the plot object.

Finally, the "plt.show()" function is called to display the plot.

Overall, the code creates a histogram plot that shows the count of observations in the "df" DataFrame for each value of the "Outcome" variable, and stacks the bars on top of each other based on the value of the "Outcome" variable. This can be useful for getting a quick sense of the distribution of the "Outcome" variable in the dataset.



#output

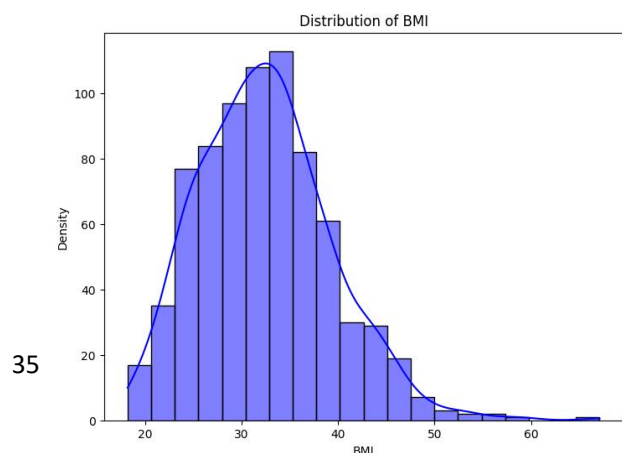
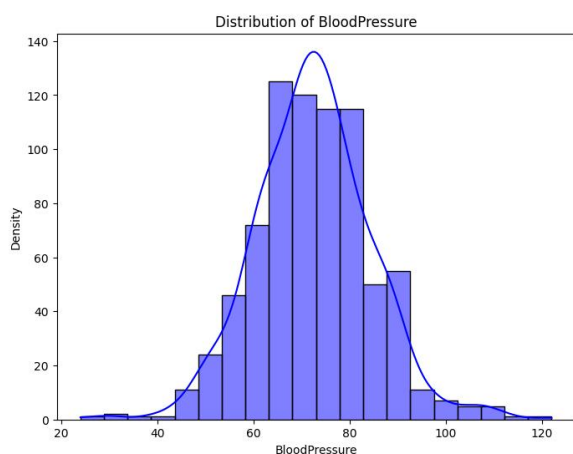
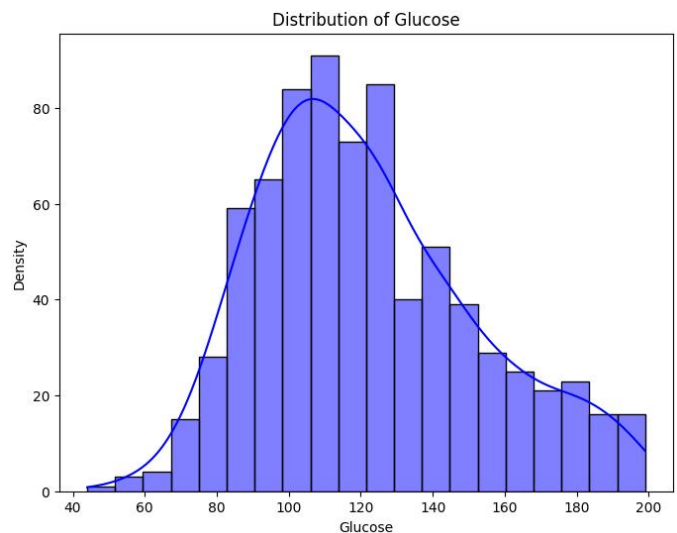
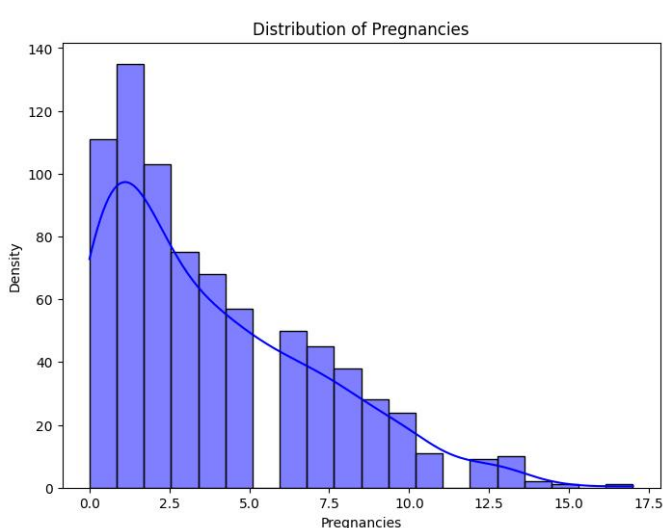
19.creates a series of histogram plots using Seaborn and Matplotlib libraries in Python. The histogram plots show the distribution of each variable in a pandas DataFrame "df".

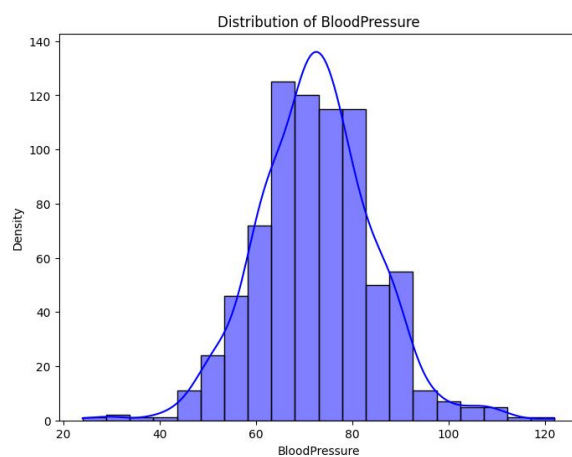
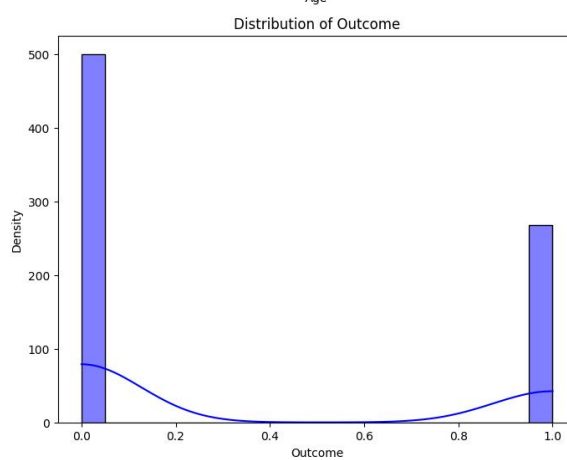
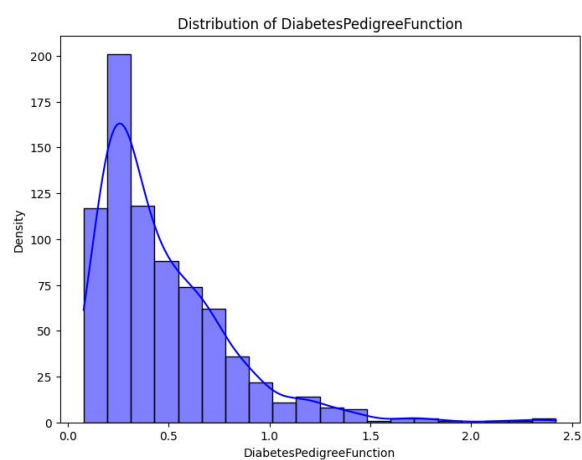
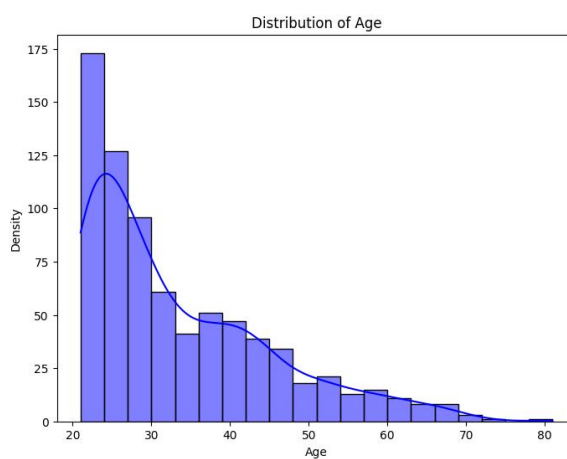
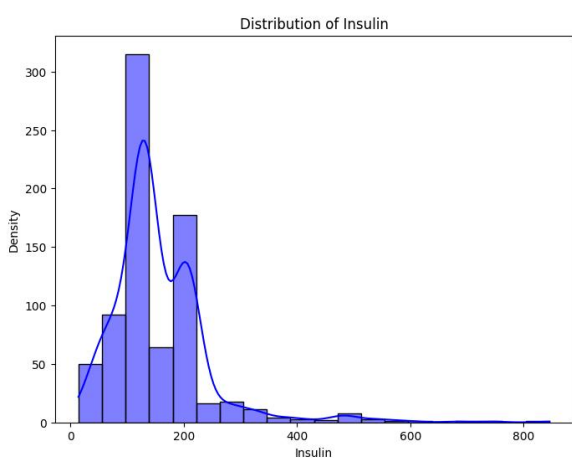
The code uses a for loop to iterate over each column in the DataFrame "df". For each column, the code creates a histogram plot using Seaborn's "histplot" function. The column values are passed as the data argument. The "kde" argument is set to True to display the kernel density estimate of the distribution on top of the histogram bars. The "bins" argument is set to 20 to specify the number of bins in the histogram. The "color" argument is set to 'blue' to choose the color of the histogram bars. The "edgecolor" argument is set to 'black' to choose the color of the edges of the bars.

The code then sets the label for the x-axis of the plot to the column name using the "xlabel" method of the Matplotlib plot object. The label for the y-axis is set to "Density" using the "ylabel" method. The title of the plot is set to "Distribution of [column name]" using the "title" method.

Finally, the "plt.show()" function is called to display the plot.

Overall, the code creates a series of histogram plots that show the distribution of each variable in the DataFrame "df". This can be useful for getting a quick sense of the range, shape, and skewness of each variable in the dataset. The kernel density estimate overlaid on the histogram bars provides additional information about the shape of the distribution.





#out puts

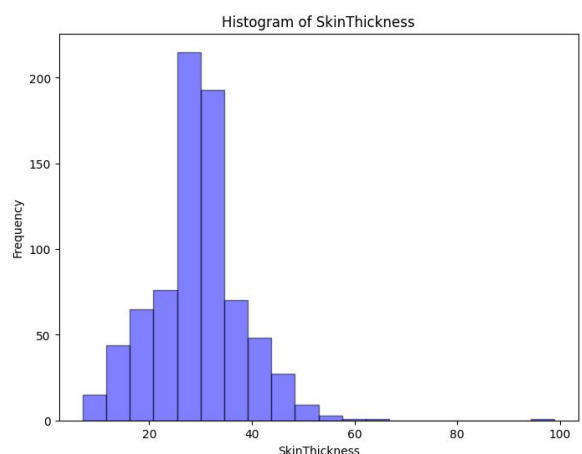
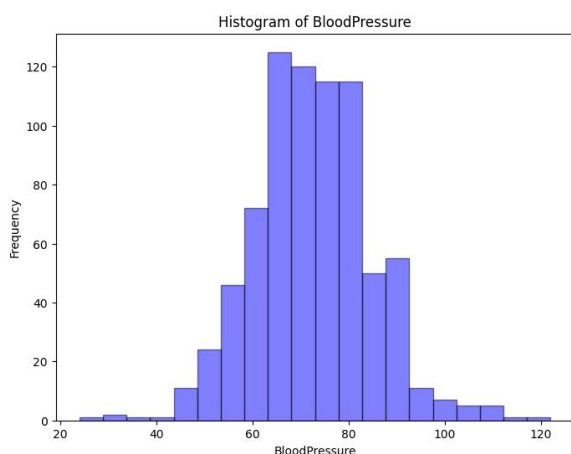
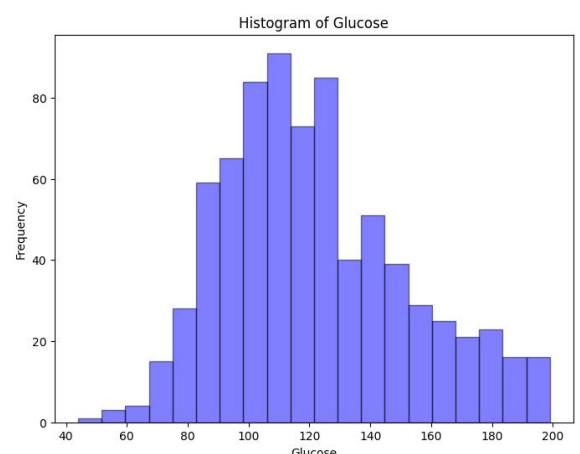
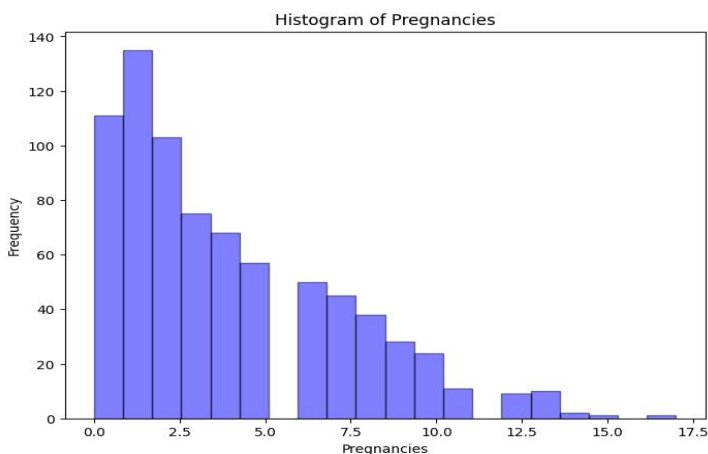
20. Creates a series of histogram plots using Matplotlib library in Python. The histogram plots show the distribution of each variable in a pandas DataFrame "df".

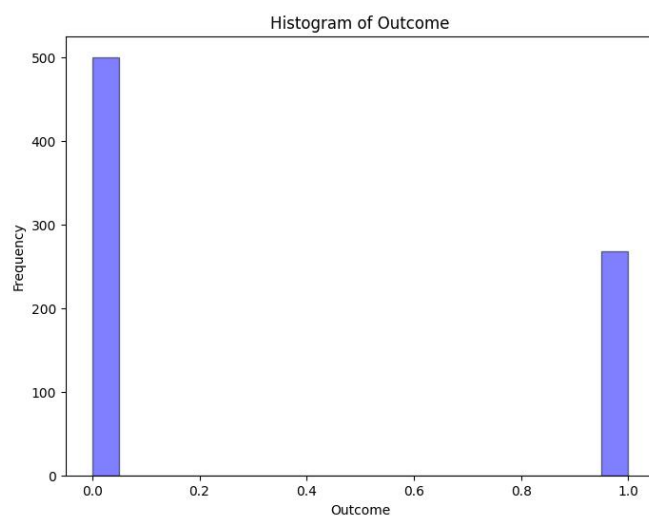
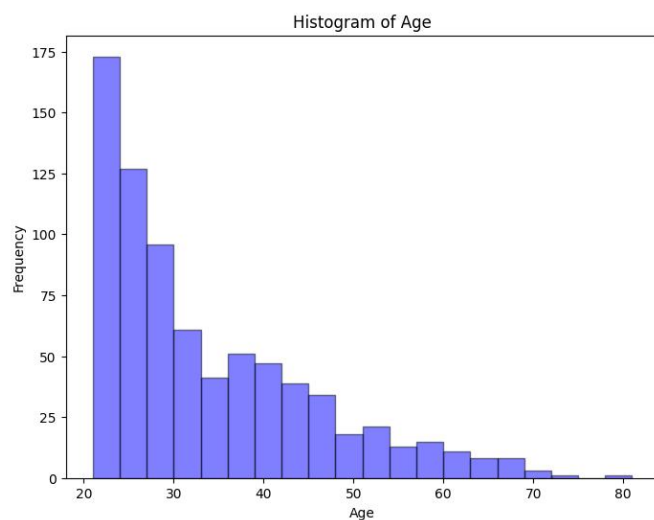
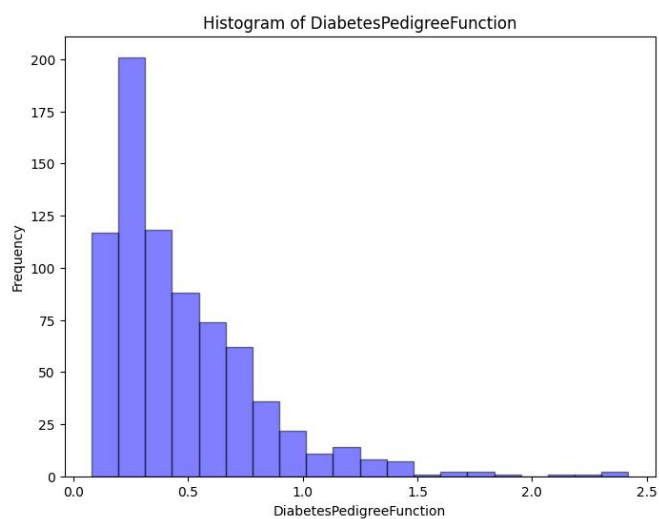
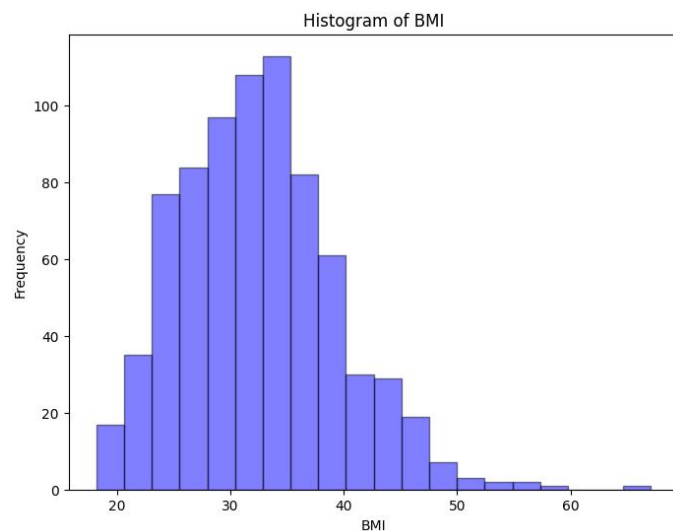
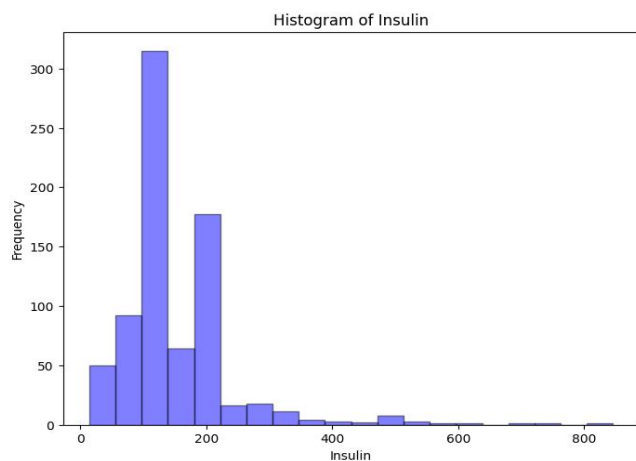
The code uses a for loop to iterate over each column in the DataFrame "df". For each column, the code creates a histogram plot using Matplotlib's "hist" function. The column values are passed as the data argument. The "bins" argument is set to 20 to specify the number of bins in the histogram. The "alpha" argument is set to 0.5 to adjust the transparency of the histogram bars. The "color" argument is set to 'blue' to choose the color of the histogram bars. The "edgecolor" argument is set to 'black' to choose the color of the edges of the bars.

The code then sets the label for the x-axis of the plot to the column name using the "xlabel" method of the Matplotlib plot object. The label for the y-axis is set to "Frequency" using the "ylabel" method. The title of the plot is set to "Histogram of [column name]" using the "title" method.

Finally, the "plt.show()" function is called to display the plot.

Overall, the code creates a series of histogram plots that show the distribution of each variable in the DataFrame "df". This can be useful for getting a quick sense of the range, shape, and skewness of each variable in the dataset. However, the use of transparency and the overlapping of the histograms in the same plot could make it difficult to compare the distributions across different variables.





#out puts

21. Creates a scatter plot using Seaborn and Matplotlib libraries in Python. The scatter plot shows the relationship between two variables, "Glucose" and "BloodPressure", in a pandas DataFrame "df".

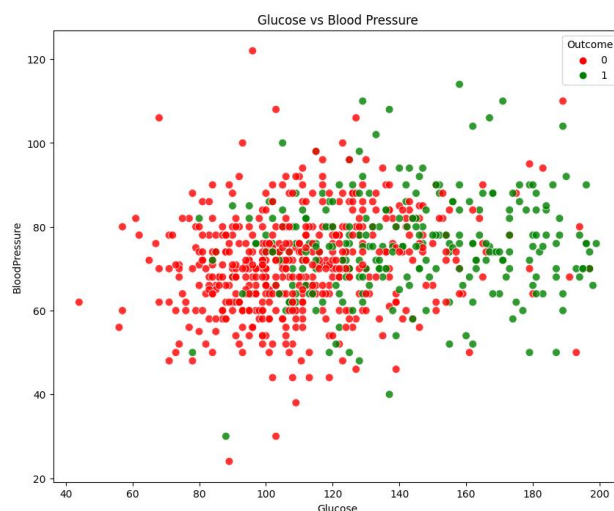
The first line of the code sets the size of the figure to be displayed using the "rcParams" function of Matplotlib. The size of the figure is set to 10 inches by 8 inches.

The second line of the code creates a custom color palette for the scatter plot using a list of colors. The first color in the list is used for the "Outcome" value of 0, and the second color is used for the "Outcome" value of 1.

The third line of the code uses Seaborn's "scatterplot" function to create a scatter plot. It specifies "Glucose" as the x-axis variable and "BloodPressure" as the y-axis variable. The "hue" argument is set to 'Outcome' to color the points based on the value of the "Outcome" variable. The "data" argument is set to the "df" DataFrame. The "s" argument is set to 60 to adjust the size of the points in the scatter plot. The "alpha" argument is set to 0.8 to adjust the transparency of the points. The "color" argument is set to 'green' to choose the color of the points. The "palette" argument is set to the custom_palette created in the previous line to choose the colors for the different values of the "Outcome" variable.

The fourth line of the code sets the title of the plot to "Glucose vs Blood Pressure" using the "title" method of the Matplotlib plot object.

Overall, the code creates a scatter plot that shows the relationship between "Glucose" and "BloodPressure" variables in the "df" DataFrame. The points are colored based on the value of the "Outcome" variable, which can help to visually identify any patterns or differences in the relationship between the variables for different values of the "Outcome" variable.



#out put

22. Creates a scatter plot using Seaborn and Matplotlib libraries in Python. The scatter plot shows the relationship between two variables, "Insulin" and "BloodPressure", in a pandas DataFrame "df".

The first line of the code sets the size of the figure to be displayed using the "rcParams" function of Matplotlib. The size of the figure is set to 10 inches by 8 inches.

The second line of the code creates a custom color palette for the scatter plot using a list of colors. The colors in the list are used for the different values of the "Outcome" variable.

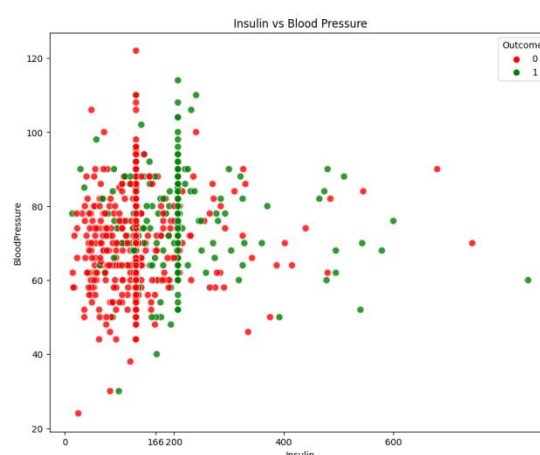
The third line of the code uses Seaborn's "scatterplot" function to create a scatter plot. It specifies "Insulin" as the x-axis variable and "BloodPressure" as the y-axis variable. The

"hue" argument is set to 'Outcome' to color the points based on the value of the "Outcome" variable. The "data" argument is set to the "df" DataFrame. The "s" argument is set to 60 to adjust the size of the points in the scatter plot. The "alpha" argument is set to 0.8 to adjust the transparency of the points. The "color" argument is set to 'orange' to choose the color of the points. The "palette" argument is set to the custom_palette created in the previous line to choose the colors for the different values of the "Outcome" variable.

The fourth line of the code sets the tick positions for the x-axis using the "xticks" method of the Matplotlib plot object. The tick positions are set to [0, 166, 200, 400, 600], which are the values of the "Insulin" variable that correspond to the quartiles of the distribution in the "df" DataFrame.

The fifth line of the code sets the title of the plot to "Insulin vs Blood Pressure" using the "title" method of the Matplotlib plot object.

Overall, the code creates a scatter plot that shows the relationship between "Insulin" and "BloodPressure" variables in the "df" DataFrame. The points are colored based on the value of the "Outcome" variable, which can help to visually identify any patterns or differences in the relationship between the variables for different values of the "Outcome" variable. The tick positions on the x-axis provide additional information about the distribution of the "Insulin" variable.



#out put

23. Creates a scatter plot using Seaborn and Matplotlib libraries in Python. The scatter plot shows the relationship between two variables, "Glucose" and "Age", in a pandas DataFrame "df".

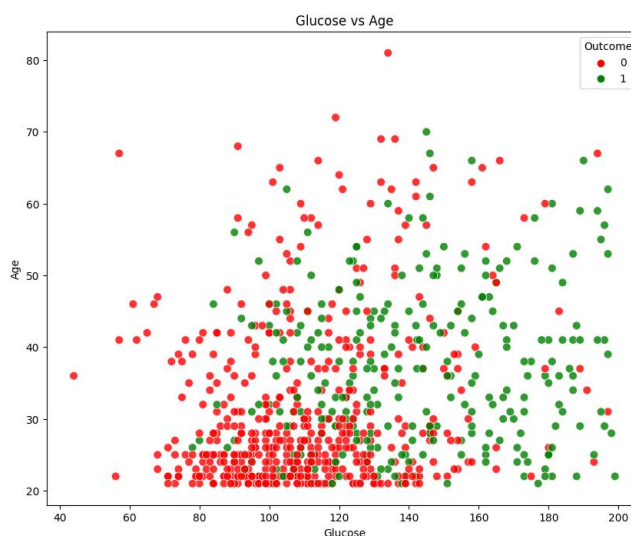
The first line of the code sets the size of the figure to be displayed using the "rcParams" function of Matplotlib. The size of the figure is set to 10 inches by 8 inches.

The second line of the code creates a custom color palette for the scatter plot using a list of colors. The colors in the list are used for the different values of the "Outcome" variable.

The third line of the code uses Seaborn's "scatterplot" function to create a scatter plot. It specifies "Glucose" as the x-axis variable and "Age" as the y-axis variable. The "hue" argument is set to 'Outcome' to color the points based on the value of the "Outcome" variable. The "data" argument is set to the "df" DataFrame. The "s" argument is set to 60 to adjust the size of the points in the scatter plot. The "alpha" argument is set to 0.8 to adjust the transparency of the points. The "color" argument is set to 'teal' to choose the color of the points. The "palette" argument is set to the custom_palette created in the previous line to choose the colors for the different values of the "Outcome" variable.

The fourth line of the code sets the title of the plot to "Glucose vs Age" using the "title" method of the Matplotlib plot object.

Overall, the code creates a scatter plot that shows the relationship between "Glucose" and "Age" variables in the "df" DataFrame. The points are colored based on the value of the "Outcome" variable, which can help to visually identify any patterns or differences in the relationship between the variables for different values of the "Outcome" variable. This type of plot can be useful for identifying potential non-linear relationships between the variables.



#out put

24. Creates a scatter plot using Seaborn and Matplotlib libraries in Python. The scatter plot shows the relationship between two variables, "BMI" and "Age", in a pandas DataFrame "df".

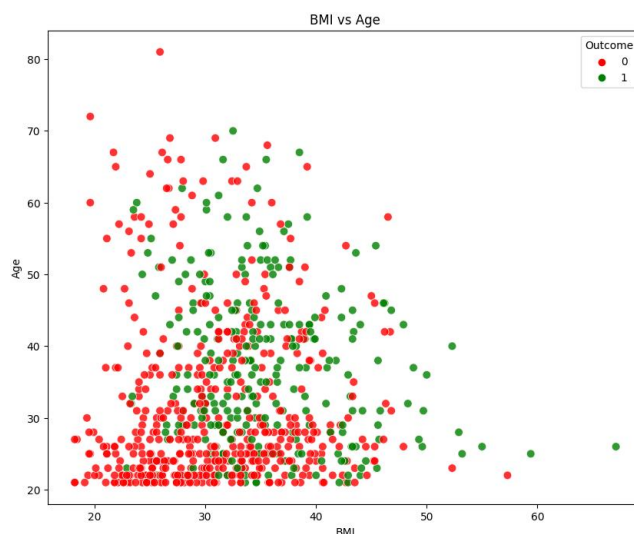
The first line of the code sets the size of the figure to be displayed using the "rcParams" function of Matplotlib. The size of the figure is set to 10 inches by 8 inches.

The second line of the code creates a custom color palette for the scatter plot using a list of colors. The colors in the list are used for the different values of the "Outcome" variable.

The third line of the code uses Seaborn's "scatterplot" function to create a scatter plot. It specifies "BMI" as the x-axis variable and "Age" as the y-axis variable. The "hue" argument is set to 'Outcome' to color the points based on the value of the "Outcome" variable. The "data" argument is set to the "df" DataFrame. The "s" argument is set to 60 to adjust the size of the points in the scatter plot. The "alpha" argument is set to 0.8 to adjust the transparency of the points. The "color" argument is set to 'orange' to choose the color of the points. The "palette" argument is set to the custom_palette created in the previous line to choose the colors for the different values of the "Outcome" variable.

The fourth line of the code sets the title of the plot to "BMI vs Age" using the "title" method of the Matplotlib plot object.

Overall, the code creates a scatter plot that shows the relationship between "BMI" and "Age" variables in the "df" DataFrame. The points are colored based on the value of the "Outcome" variable, which can help to visually identify any patterns or differences in the relationship between the variables for different values of the "Outcome" variable. This type of plot can be useful for identifying potential non-linear relationships between the variables.



#out put

25. Creates a scatter plot using Seaborn and Matplotlib libraries in Python. The scatter plot shows the relationship between two variables, "SkinThickness" and "DiabetesPedigreeFunction" (DPF), in a pandas DataFrame "df".

The first line of the code sets the size of the figure to be displayed using the "rcParams" function of Matplotlib. The size of the figure is set to 10 inches by 8 inches.

The second line of the code creates a custom color palette for the scatter plot using a list of colors. The colors in the list are used for the different values of the "Outcome" variable.

The third line of the code uses Seaborn's "scatterplot" function to create a scatter plot. It specifies "SkinThickness" as the x-axis variable and "DiabetesPedigreeFunction" (DPF) as the y-axis variable. The "hue" argument is set to 'Outcome' to color the points based on the value of the "Outcome" variable. The "data" argument is set to the "df" DataFrame. The "s" argument is set to 60 to adjust the size of the points in the scatter plot. The "alpha" argument is set to 0.8 to adjust the transparency of the points. The "color" argument is set to 'purple' to choose the color of the points. The "palette" argument is set to the custom_palette created in the previous line to choose the colors for the different values of the "Outcome" variable.

The fourth line of the code sets the title of the plot to "Skin Thickness vs DPF" using the "title" method of the Matplotlib plot object.

Overall, the code creates a scatter plot that shows the relationship between "SkinThickness" and "DiabetesPedigreeFunction" (DPF) variables in the "df" DataFrame. The points are colored based on the value of the "Outcome" variable, which can help to visually identify any patterns or differences in the relationship between the variables for different values of the "Outcome" variable. This type of plot can be useful for identifying potential non-linear relationships between the variables.



#out put

26. Performing data preprocessing tasks before building a machine learning model. It is splitting the original DataFrame "df" into two separate DataFrames: one for the input features (X) and one for the output target (y).

The first line of the code creates a DataFrame "X" that contains all columns of the original DataFrame "df" except for the "Outcome" column. This is done using the Pandas "drop" method, which drops the specified column from the DataFrame.

The second line of the code creates a Series "y" that contains only the "Outcome" column of the original DataFrame "df".

The third line of the code creates a StandardScaler object from the scikit-learn library, which is used to transform the data by scaling each feature to have a mean of 0 and a standard deviation of 1.

The fourth line of the code applies the scaler to the DataFrame "X" using the "fit_transform" method of the StandardScaler object. This scales all the features in the DataFrame.

The fifth line of the code splits the preprocessed data into training and testing data. The "train_test_split" function from scikit-learn is used to split the data. The "test_size" argument is set to 0.3, which means that 30% of the data will be used for testing and 70% will be used for training. The "random_state" argument is set to 1, which ensures that the data is split in the same way every time the code is run.

The resulting variables are:

- "X": a DataFrame containing the input features (pregnancies, glucose, blood pressure, skin thickness, insulin, BMI, diabetes pedigree function, age) after scaling.
- "y": a Series containing the output target (0 or 1) for each observation in the original DataFrame.
- "x_train": a DataFrame containing the input features for the training set.
- "x_test": a DataFrame containing the input features for the testing set.
- "y_train": a Series containing the output target for the training set.
- "y_test": a Series containing the output target for the testing set.

27. Defines a function "classification_model" that takes in several arguments, including the true values of the output target for the testing set ("y_test"), the predicted values of the output target for the testing set ("prediction"), the machine learning model used to make the predictions ("model"), and the input features and output target for the training set ("x_train" and "y_train", respectively).

The function starts by calculating the accuracy score and F1 score for the predictions made by the model on the testing set. It then performs cross-

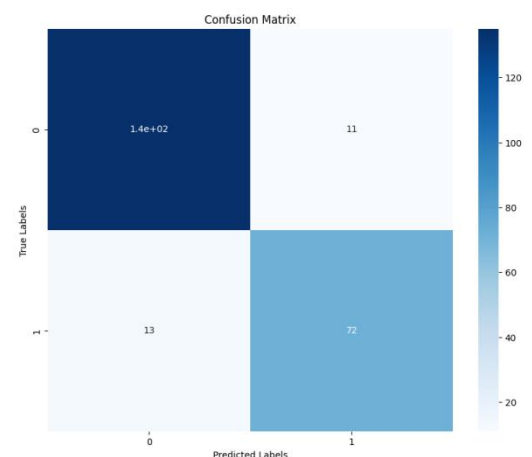
validation with 5 folds on the training set using the KFold function from scikit-learn. The function splits the training set into 5 subsets, trains the model on 4 of the subsets, and validates the model on the remaining subset. It repeats this process for each subset and calculates the accuracy score for each validation set. Finally, the function reports the mean accuracy score across all validation sets as the cross-validation score.

The code then defines another function "histogram_plot" that takes in a DataFrame "df" and creates a histogram plot for each column in the DataFrame using Seaborn's "histplot" function. The resulting plot shows the distribution of each feature in the DataFrame.

The next part of the code creates an instance of the GradientBoostingClassifier from scikit-learn, fits the model to the training set using the "fit" method, and makes predictions on the testing set using the "predict" method. It then prints the classification report using the "classification_report" function from scikit-learn, which shows several metrics including precision, recall, and F1 score for each class in the output target. The code also creates a confusion matrix using Seaborn's "heatmap" function and displays it using Matplotlib. The confusion matrix shows the number of true positive, false positive, true negative, and false negative predictions made by the model.

Overall, the code performs model evaluation for a Gradient Boosting Classifier on the preprocessed diabetes dataset. It calculates accuracy and F1 scores, performs cross-validation, and displays the classification report and confusion matrix. The histogram plot shows the distribution of each feature in the preprocessed dataset.

...	precision	recall	f1-score	support
0	0.91	0.92	0.92	146
1	0.87	0.85	0.86	85
accuracy			0.90	231
macro avg	0.89	0.89	0.89	231
weighted avg	0.90	0.90	0.90	231



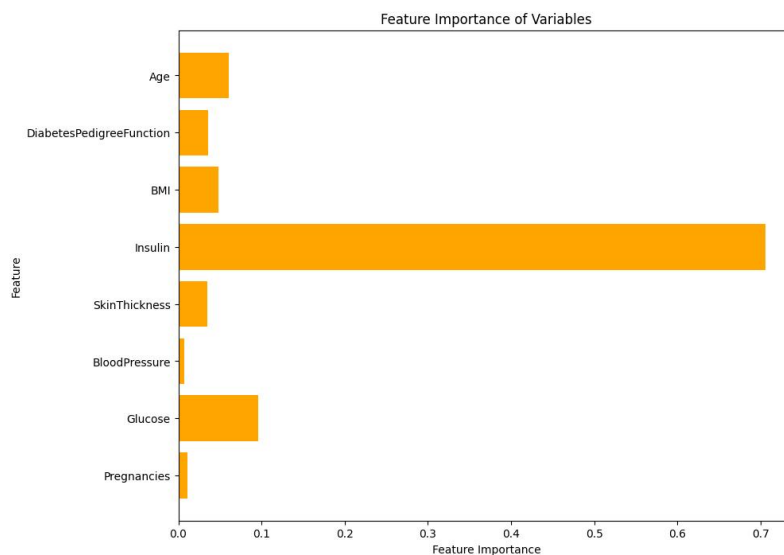
#out put

28. Creates a horizontal bar plot using Matplotlib to display the feature importance of each input feature in the Gradient Boosting Classifier model. The feature importance values are obtained from the "feature_importances_" attribute of the trained model.

The first line of the code assigns the feature importance values to the variable "feature_importances".

The second line of the code creates a horizontal bar plot using Matplotlib's "barh" function. The x-axis values are the feature importance values, which are plotted as horizontal bars. The y-axis values are the names of the input features in the training set, which are obtained from the column names of the "x_train" DataFrame. The color of the bars is set to orange using the "color" argument. The x-axis label is set to "Feature Importance" using the "xlabel" method. The y-axis label is set to "Feature" using the "ylabel" method. The title of the plot is set to "Feature Importance of Variables" using the "title" method.

The resulting plot shows the relative importance of each input feature in the trained Gradient Boosting Classifier model. Features with higher importance values are considered to have a greater impact on the model's predictions. This type of plot can be useful for feature selection and identifying the most important variables for a given problem.



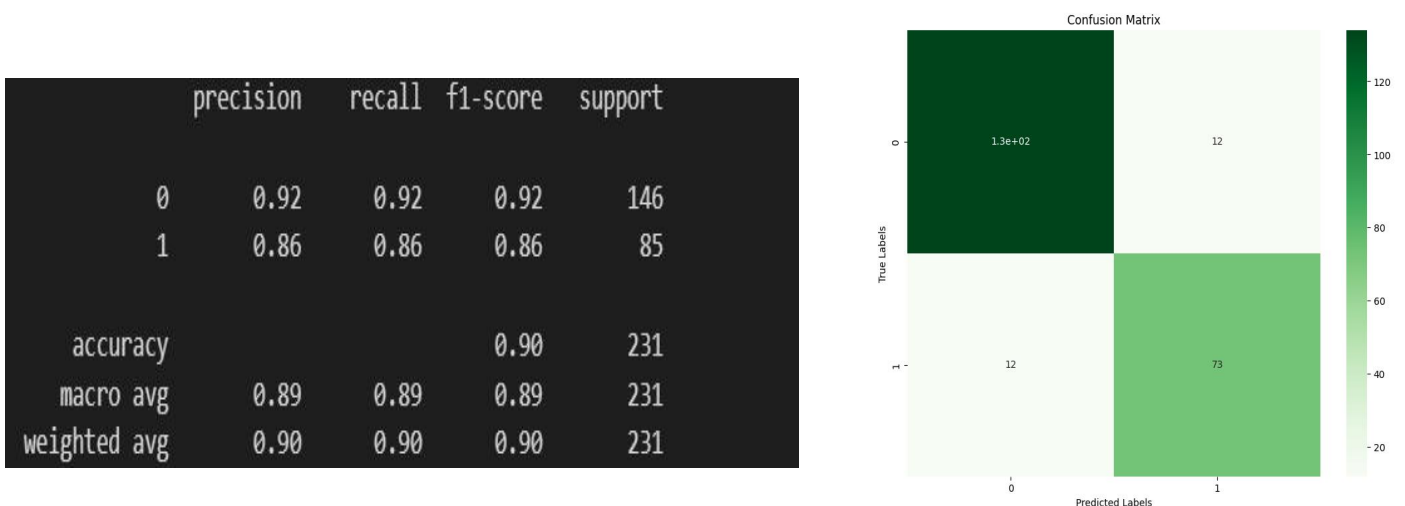
#out put

29. Creates two new DataFrames "x_tr" and "x_te" that contain only a subset of the original input features. Specifically, the DataFrames contain the "Insulin", "Glucose", "BMI", "Age", and "SkinThickness" features. This is done using the Pandas "loc" method to select only the desired columns from the original DataFrames "x_train" and "x_test".

The code then creates a new instance of the GradientBoostingClassifier model and fits it to the training set using the new subset of input features in "x_tr" and the output target in "y_train". The model is then used to make predictions on the testing set using the "predict" method and the resulting predictions are assigned to the variable "prediction5".

The code prints the classification report using the "classification_report" function from scikit-learn, which shows several metrics including precision, recall, and F1 score for each class in the output target. The code also creates a confusion matrix using Seaborn's "heatmap" function and displays it using Matplotlib. The confusion matrix shows the number of true positive, false positive, true negative, and false negative predictions made by the model.

The resulting plot and classification report show the performance of the Gradient Boosting Classifier model on the modified subset of input features. By selecting only a subset of the original input features, the model may perform better or worse than the previous model that used all input features. The classification report and confusion matrix can help to evaluate the performance of the model and the feature subset.



#out put

4.3 Explain full code

This code is a machine learning project for predicting diabetes in patients based on various features. Here is a step-by-step explanation of the code:

- ❖ . The required libraries are imported:
 - numpy and pandas for working with data
 - seaborn and matplotlib for data visualization
 - warnings to ignore warning messages
 - sklearn for machine learning algorithms and metrics
 - mlxtend for plotting confusion matrices
- ❖ The diabetes dataset is read in using pandas.
- ❖ Zeros in the columns Glucose, Blood Pressure, SkinThickness, Insulin, and BMI are replaced with NaN values.
- ❖ A box plot is created to visualize the distribution of the variables in the datasets.
- ❖ The number of null values in each column is checked.
- ❖ A function is defined to find the mean value of each variable grouped by the outcome column.
- ❖ The null values in Glucose, Blood-pressure, SkinThickness, Insulin, and BMI are replaced with the mean values calculated in step 6.
- ❖ A histogram is plotted to show the count of the outcome variable.
- ❖ A correlation plot is created to visualize the correlation between the variables in the datasets.
- ❖ Histograms are plotted for each column in the dataset.
- ❖ Scatter plots are created to visualize the relationship between pairs of variables.
- ❖ The dataset is split into training and testing sets.
- ❖ The training set is scaled using StandardScaler.
- ❖ A function is defined to evaluate the performance of a classification model.
- ❖ A Gradient Boosting Classifier is trained on the training set.
- ❖ The performance of the model is evaluated using classification report and confusion matrix.
- ❖ The feature importance of the variables is visualized using a bar chart.
- ❖ A new training and testing set is created with only the most important variables.
- ❖ A new Gradient Boosting Classifier is trained on the new training set.

The performance of the model is evaluated using classification report and confusion matrix

The main objective of this project is to predict whether a patient has diabetes or not based on various features such as glucose level, blood pressure, skin thickness, insulin level, BMI, age, and pregnancy history. This is a binary classification problem, where the target variable is the outcome column, which takes on two values: 0 (no diabetes) and 1 (diabetes).

The dataset used in this project is the Pima Indians Diabetes Database, which contains 768 observations and 9 variables. The dataset has some missing values and some variables have zeros that are not valid for that feature, such as zero glucose level or zero blood pressure. Thus, some data pre-processing steps are performed to handle these missing values and zeros.

The data pre-processing steps include replacing the zeros with NaN values, checking for null values, replacing the NaN values with the mean values of each variable grouped by the outcome column, and splitting the dataset into training and testing sets.

After the data preprocessing, exploratory data analysis is performed to gain insights into the data and visualize the relationships between variables. The EDA includes plotting box plots, histograms, correlation matrix, scatter plots, etc.

The machine learning algorithm used in this project is Gradient Boosting Classifier, which is a powerful ensemble learning algorithm that combines weak learners (decision trees) to create a strong classifier. The model is trained on the training set and evaluated on the testing set using various metrics such as accuracy, precision, recall, F1 score, and confusion matrix.

The feature importance of the variables is also visualized using a bar chart to identify the most important variables that contribute most to the model's performance. Finally, a new model is trained using only the most important variables to see if the model's performance improves.

Explain the Gradient Boosting Classifier

Gradient Boosting for classification. This algorithm builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage $n_classes_$ regression trees are fit on the negative gradient of the loss function, e.g. binary or multi-class log loss.

Gradient Boosting Classifier is an ensemble machine learning algorithm that is used for both regression and classification problems. It works by combining multiple weak learners, typically decision trees, to create a strong predictive

model. The algorithm is called "gradient boosting" because it uses gradient descent to minimize the loss function at each iteration.

The idea behind gradient boosting is to iteratively add new decision trees to the model, and each new tree is trained to correct the errors of the previous trees.

During each iteration, the algorithm calculates the negative gradient of the loss function with respect to the predicted values, which is then used as the target variable for the next decision tree. The output of the final model is the sum of the predictions of all the decision trees.

Gradient Boosting Classifier has several advantages over other machine learning algorithms. Firstly, it is very powerful and can achieve high accuracy even with complex datasets. Secondly, it is less prone to over-fitting than other algorithms such as decision trees or random forests. Finally, it can handle both continuous and categorical data, making it a versatile algorithm for many different types of problems.

However, Gradient Boosting Classifier also has some limitations. It can be computationally expensive and requires careful tuning of hyper-parameters such as learning rate, number of trees, and depth of trees. Additionally, it can be sensitive to outliers and noise in the data, so it is important to preprocess the data carefully before using this algorithm.

Overall, Gradient Boosting Classifier is a popular and effective algorithm for solving many different types of classification problems, especially when accuracy is of utmost importance.

Sure, here are some additional details about Gradient Boosting Classifier:

Gradient Boosting is a boosting algorithm, which means it trains a series of weak models sequentially and each model tries to correct the errors of the previous model. In the case of Gradient Boosting Classifier, each weak model is a decision tree, and the final model is a combination of all the decision trees. The algorithm starts with a single decision tree and iteratively adds more decision trees to the model until a predefined stopping criterion is met.

The key idea behind Gradient Boosting is to use the gradient of the loss function as a guide to update the model. The loss function measures the difference between the predicted values

and the actual values. During each iteration, the algorithm calculates the gradient of the loss function with respect to the predicted values and uses it to update the model. The gradient descent algorithm is used to minimize the loss function by adjusting the parameters of the model.

Gradient Boosting Classifier has several hyperparameters that need to be tuned to achieve the best performance. The most important hyperparameters are:

- ◆ learning rate: determines how much each decision tree contributes to the final prediction. A smaller learning rate makes the training slower but can result in better generalization.
- ◆ number of trees: determines the number of decision trees to be added to the model.
- ◆ max depth of trees: determines the maximum depth of the decision trees. A deeper tree can better capture the complex patterns in the data but can also lead to overfitting.

Gradient Boosting Classifier is a powerful algorithm that can handle a wide range of data types and achieve high accuracy. It is widely used in industry and academia for various applications such as fraud detection, customer churn prediction, and medical diagnosis. However, it can also be computationally expensive and requires careful tuning of hyper-parameters, so it is important to use it judiciously and with proper care.

Sure, here are some more details about Gradient Boosting Classifier:

Advantages of Gradient Boosting Classifier

- Its ability to handle a variety of data types, including both numerical and categorical data. This makes it a versatile algorithm that can be used for many different types of problems. The algorithm is also known for its high accuracy, especially when compared to other machine learning algorithms such as logistic regression or decision trees.
- Its ability to handle missing values and outliers in the data. The algorithm can use surrogate splits and robust loss functions to deal with missing values and outliers, respectively.

However, Gradient Boosting Classifier can also be computationally expensive, especially when dealing with large datasets or a large number of features. The algorithm also requires careful tuning of hyperparameters, such as the learning rate, number of trees, and maximum depth of trees. If these hyperparameters are not tuned properly, the algorithm may overfit the training data or underfit the test data.

To address these issues, researchers have proposed several extensions and variations of Gradient Boosting Classifier. One such extension is XGBoost (Extreme Gradient Boosting), which is designed to be faster and more accurate than standard Gradient Boosting Classifier. XGBoost uses a technique called

"regularized boosting" to prevent overfitting and can also handle missing values and sparse data.

Another variation of Gradient Boosting Classifier is LightGBM (Light Gradient Boosting Machine), which is designed to be even faster and more scalable than XGBoost. LightGBM uses a technique called "gradient-based one-side sampling" to speed up the training process and reduce memory usage.

Overall, Gradient Boosting Classifier is a powerful and versatile algorithm that can be used for many different types of classification problems. However, it requires careful tuning of hyper-parameters and can be computationally expensive, so it is important to use it judiciously and with proper care.

4.4 Analysis & Design

4.2.1 Analysis

The analysis will also give answer to questions like what tools and technologies should be or is possible to use, what can be the project schedule, make better budget estimations, etc. The output of this phase is a written analysis document, which includes at least:

- Detailed system description.
- Identification of project phases.
- Resource estimation.
- Time schedule.
- Identification of possible problems known at this stage and suggestion how to prevent them.
- List and recommendations of technologies to be used.

Tools Used:

- Edrawmax.exe
- OPEN PROJECT
- PROJECT STAR UML
- MS AXCEL SHHET

4.5 Implementation

4.5.1 Website

Website development is divided into two main parts:

- Front End
- Back End

The development environment of Front end included programs such as STREAMLIT The development environment of Back end included programs such as:

VISIUL STUDIO 2022 & MYSQ

Chapter 5: Database

Using SQLite as a database for a prediction of diabetes project can be a suitable choice, especially for smaller-scale projects or when there is a need for a lightweight and embedded database solution. Here's how you can utilize SQLite for your project:

1. **Database Design:** Define the database schema that will store the relevant data for your diabetes prediction project. Identify the tables and their respective fields necessary to store information such as patient demographics, risk factors, medical history, and outcomes.
2. **SQLite Installation:** Download and install the SQLite database management system appropriate for your operating system. SQLite is a server-less, file-based database, so there is no need for a separate server installation.
3. **Database Connection:** Establish a connection to the SQLite database within your project's programming language. Popular programming languages like Python, R, and Java provide libraries or packages for SQLite connectivity. For example, in Python, you can use the `'sqlite3'` module to connect to and interact with an SQLite database.
4. **Database Operations:** Perform CRUD (Create, Read, Update, Delete) operations on the SQLite database to manage the data. Create tables, insert data, retrieve records based on specific criteria, update existing data, and delete records as needed. Ensure proper data validation and handling to maintain data integrity.
5. **Data Import and Export:** Load the relevant datasets into the SQLite database for training and testing the prediction model. You can import data from various file formats such as CSV, JSON, or Excel into SQLite tables. Additionally, export query results or processed data from SQLite for further analysis or model training.
6. **Database Queries:** Utilize SQL (Structured Query Language) statements to query the SQLite database and retrieve specific data required for model development, feature selection, or validation. Write SQL queries to filter, aggregate, join, or transform data as per your project requirements.
7. **Database Optimization:** Optimize the database performance by creating appropriate indexes on columns frequently used in queries. Analyze query execution plans to identify potential bottlenecks and optimize slow-performing

queries. Efficient indexing and query optimization can improve the overall responsiveness of the database.

8. Backup and Security: Implement regular database backups to ensure data preservation in case of accidental data loss or corruption. Additionally, consider implementing security measures to protect sensitive data, such as encryption and access controls.

9. Integration with Model Development: Integrate the SQLite database with your prediction model development pipeline. Retrieve the required data from the database, preprocess it, and use it for training and testing the prediction model.

10. Table Relationships: Define appropriate relationships between tables in your SQLite database if you need to represent complex associations or dependencies. For example, you might have tables for patients, medical records, and risk factors, and establish foreign key relationships between them to ensure data consistency and integrity.

11. Indexing: Identify columns that are frequently used in queries and create indexes on those columns to improve query performance. Indexing can speed up data retrieval, especially when dealing with large datasets or complex queries. However, be mindful of the trade-off between query performance and the impact on data modification operations (inserts, updates, deletes) as indexes can slightly slow down these operations.

12. Database Backup and Recovery: Implement a regular backup strategy to safeguard your SQLite database from data loss or corruption. Create scheduled backups or automate backup processes to ensure that you have copies of your database in case of any unforeseen events. Additionally, consider implementing a recovery plan to restore the database in case of a failure or corruption.

13. SQL Optimization: Optimize your SQL queries to enhance performance. Analyze query execution plans using the EXPLAIN statement to identify inefficient query operations and optimize them. Techniques such as query rewriting, proper indexing, and selective retrieval of data can significantly improve query performance.

14. Transactions: Utilize transactions to ensure data consistency and integrity, especially when dealing with multiple database operations that should be treated as a single unit of work. Transactions help maintain the integrity of the database by allowing you to either commit or roll back changes as a group, ensuring that all or none of the operations are applied.

15. Testing and Validation: Develop thorough testing procedures to verify the accuracy and reliability of your database operations. Create test datasets, write test cases, and perform extensive testing to ensure that your database functions as expected. Validate that the data retrieved from the database matches the expected results based on predefined criteria.

16. Scalability Considerations: SQLite is well-suited for small to medium-scale projects, but it may not be the best choice for highly concurrent or large-scale applications. Evaluate the scalability requirements of your prediction project and assess if SQLite can handle the anticipated data volume and user load. Consider alternative database solutions if scalability becomes a concern.

In summary

-Using SQLite for a prediction of diabetes project offers a lightweight and embedded database solution. Key considerations include designing the database schema, establishing connections, performing CRUD operations, and utilizing SQL queries for data manipulation and retrieval. Additional aspects to keep in mind are table relationships, indexing for query performance, backup and recovery strategies, SQL optimization, transaction management, testing and validation, and scalability considerations. SQLite is suitable for small to medium-scale projects, but its limitations in terms of scalability and concurrent user access should be evaluated based on project requirements. Continuously refer to the SQLite documentation and online resources for detailed guidance throughout your project.

-The prediction of diabetes project aims to develop a diabetes prediction model using SQLite as the database system. SQLite is chosen for its lightweight and embedded nature, making it suitable for small to medium-scale projects. The project involves designing the database schema, establishing connections, and performing CRUD operations using SQL queries.

-Key considerations include table relationships, indexing for query performance, backup and recovery strategies, SQL optimization, transaction management,

testing, and scalability. Data import/export capabilities are utilized to import relevant datasets and export query results. Database optimization techniques, such as gathering statistics and configuring SQLite parameters, are employed to enhance performance.

-Security measures are implemented to protect sensitive data, including file-level encryption and application-level security. Cross-platform compatibility is ensured, allowing the project to work across different operating systems. Replication and synchronization mechanisms are considered for distributed systems or offline functionality.

-Monitoring tools and profilers are used to analyze performance and optimize resource utilization. Proper error handling and exception management techniques are implemented to handle potential errors gracefully. Extensive documentation and community support resources are utilized for guidance and assistance.

-Thorough testing, validation, and regular updates are performed to ensure the accuracy, reliability, and efficiency of the SQLite implementation. By effectively utilizing SQLite, the project aims to efficiently manage data, ensure security, optimize performance, and leverage its cross-platform compatibility and rich feature set for successful diabetes prediction.

In India, diabetes is a major issue. Between 1971 and 2000, the incidence of diabetes rose ten times, from 1.2% to 12.1%. 61.3 million people 20–79 years of age in India are estimated living with diabetes (Expectations of 2011). It is expected that by 2030 this number will rise to 101.2 million. In India there are reportedly 77.2 million people with prediabetes. In 2012, nearly 1 million people in India died of diabetes. 1 out of 4 individuals living in Chennai's urban slums suffer from diabetes, which is about 7 per cent by three times the national average. One third of the deaths in India involve people under non-communicable diseases Sixty years old. Indians get diabetes 10 years before their Western counterparts on average. Changes in lifestyle lead to physical decreases Increased fat, sugar and activities activity calories and higher insulin cortisol levels Obesity and vulnerability. In 2011, India cost around \$38 billion annually as a result of diabetes.

International Diabetes Federation (IDF) stated that 382 million people are living with diabetes worldwide. Over the last few years, the impact of diabetes has been increased drastically, which makes it a global threat. At present, Diabetes has

steadily been listed in the top position as a major cause of death. The number of affected people will reach up to 629 million i.e. 48% increase by 2045. However, diabetes is largely preventable and can be avoided by making lifestyle changes. These changes can also lower the chances of developing heart disease and cancer. So, there is a dire need for a prognosis tool that can help the doctors with early detection of the disease and hence can recommend the lifestyle changes required to stop the progression of the deadly disease.

Diabetes can be considered as one of the main challenges in the healthcare community worldwide and its impact is increasing at a very high pace.

Consequently, it is the seventh major reason for the premature death rate in 2016 worldwide mentioned by the World Health Organization (WHO) . According to the diabetes global pervasiveness, 1.6 million got died each year because of diabetes . WHO has demonstrated in its first global report that the number of persons suffering from diabetes increased from 108 million (4.2%) to 422 million (8.5%) till the end of 2014 . On world diabetes day 2018, WHO has joined the partners from all over the world for showcasing the impact of diabetes.

According to WHO, 1 in 3 adult is reported overweight and the problem is increasing day by day. Diabetes is convicted as the main reason for heart attack, kidney failure and stroke blindness .

Diabetes can be considered as a chronic disease in which glucose (blood sugar) is not metabolized in the body (glucose is produced from the food we eat); therefore, it increases the level of sugar in the blood over the acceptable limits. In diabetes, the body is not able to generate insulin or to respond to the produced insulin. Diabetes is incurable until now, but it can be prevented with early knowledge. A person having diabetes is prone to severe complications like nerve damage, heart attack, kidney failure, and stroke. High levels of glucose in the body can cause the problem of hyperglycemia, which results in abnormalities in the cardiovascular system and also causes serious problems in the functioning of various human organs like eyes, kidneys, and nerves.

In Early diagnosis, the prediction and diagnosis of the disease are analyzed through a doctor's knowledge and experience, but that can be inaccurate and susceptible. Healthcare Industry collects a huge amount of data related to healthcare, but that data is unable to perceive undetected patterns for making effective decisions . Since manual decisions can be highly dangerous for early disease diagnosis as they are based on the healthcare official's observations and judgment which is not always correct . There can be some patterns that remain

hidden and can impact observations and outcomes. As a result, patients are getting a low quality of service; therefore an advance mechanism is required for early detection of disease with an automated diagnosis and better accuracy. Various undetected errors and hidden patterns give rise to diverse data mining and machine algorithms which can draw efficient results with reliable accuracy .

Due to the day to day growing impact of diabetes, a variety of data mining algorithms have been introduced for collecting hidden patterns from large healthcare data. Further, that data can be used for feature selection and automated prediction of diabetes .

The main intent of this research work is to propose the development of a prognostic tool for early diabetes prediction and detection with improved accuracy. There have been an extensive amount of data and datasets available on the internet or external sources and the PIMA dataset which has been used in this work is one of the most widely used dataset in many researches and it is collected by the National Institute of Diabetes and Digestive and Kidney Diseases (NIDDK). This research work represents comprehensive studies done on the PIMA datasets using data mining algorithms like DT, NB, ANN, and DL .The comparison of algorithms is represented in a logical and well-organized manner from which DL provides more effective and prominent results. DL is a technology that self-learns from data and is used effectively for predicting diabetes nowadays . A DL network is a technique that uses ANN properties in which neurons are interconnected to each other with lot of representation layers . DL learns the representation of data by enlarging the level of consideration from one layer to another hence increasing the accuracy . The model achieves high accuracy of 98.07% by employing DL in RapidMiner tool which proposes a well-structured diabetes knowledge formatted for medical officials and practitioners. Moreover, the task is to reduce the efforts and to provide better results in comparison with the traditional methods . These machine learning methods tend to improve the accuracy of the available methods. But DL and ANN provide the best results as they are more reliable, robust and accurate in terms of prediction of the disease.

The remaining part of the paper is organized in the following manner: third section puts forth the previous important work done on diabetes prediction using data mining algorithms. Fourth section of the paper presents the dataset description, data pre-processing process, and proposed methodology. Fifth section covers the results and discussion part. The paper concludes in sixth section along with future scope.

The dataset used for the study is PIMA Indian dataset (PID) by NIDDK. The main motivation behind using the PIMA dataset is that most of the population in today's world follows a similar lifestyle having a higher dependency on processed foods with a decline in physical activity. PID is a long term cohort study since 1965 by NIDDK because of the maximum risk of diabetes. The dataset contained certain diagnostic parameters and measurement through which the patient can be identified with any kind of chronic disease or diabetes before time. All of the Participants in PID are females and at least 21 years old. PID

composed of a total of 768 instances, from which 268 samples were identified as diabetic and 500 were non-diabetics. The 8 most influencing attributes that contributed towards the prediction of diabetes are as follows: several pregnancies the patient has had, BMI, insulin level, age, Blood Pressure, Skin thickness, Glucose, DiabetesPedigreeFunction with label outcome (Table 1).

Figure 3 demonstrates the diverse characteristics of each attribute and their range used in the PIMA dataset in graphical form.

Data pre-processing

Most of the collected data is liable to be influenced as reckless. Besides this, the data quality is important as it affects the prediction results and accuracy to a large extent. Therefore, Datasets need to be properly balanced and divided between testing and training data at a certain ratio, So that sampling can be done efficiently for better prediction outcomes. Sampling is a process of selecting a representative portion of data for extracting characteristics and parameters from large datasets consistently; therefore, it can contribute in a better manner concerning the training model of machine. For maintaining that consistency we need to apply some sampling techniques (linear sampling, shuffled sampling, stratified sampling, and automatic sampling) on the dataset, that sampling techniques randomly splits the dataset into subsets and evaluates the prediction model. Those diverse sampling techniques perform dissimilar permutation and combination of a representative set of information from the collected data which are shown here.

- Linear sampling: This sampling technique linearly divides the dataset into partitions as dataset representative. Along with that, it doesn't change the sequence of tuples and fields in the subsets.

- **Shuffled sampling:** This sampling technique split the dataset randomly and builds subsets from the applied dataset. Data selected arbitrarily for assembling subsets.
 - **Stratified sampling:** This sampling technique split dataset arbitrarily and constructs the subsets. But the method also certifies that the distribution of class should be static all over the dataset. For example, if the used dataset has used binominal classification, then stratified sampling method constructs build arbitrary subsets in such a way that every subset include roughly the same proportions of the two value of class labels.
 - **Automatic:** The automated sampling method uses stratified sampling as the default sampling technique depends on the features of the dataset. If the technique doesn't go with the type of data then it uses a suitable one.
- The main object of this study is to present the most promising features that are needed to predict the patient having diabetes at an early stage. An ample amount of research work has been done on the invasive automated discovery of diabetes. Therefore, it all depends on what features were extracted and which type of classifier has been applied upon to get the maximum outcome. Hence, Diversity of learning has been analyzed that these factors of the dataset can be applied for classifying diverse risk factors for prophecy . This paper presents the all-embracing studies conducted on the PIMA dataset. The association of diverse classification algorithms evaluated on the various strata will maintain a well-organized format for the discovery of diabetes along with managing their hazard dimensions and treatment strategies for medical practitioners.
 - This proposed methodology consists of two main parts, first how accuracy is obtained using diverse classification models and second is model validation. There are varied machine learning methodologies available that are constructive to analyze the undetected patterns for evaluation of risk factors in diseases like diabetes. Further, it is being observed that the presentation of conventional methods is not up to the acceptance level in speech and object recognition because of a high dimension of data . The inadequacy of machine learning algorithms boosted the DL research and it tends to produce more accurate results and dominates other algorithms in terms of accuracy. A lot of research has been done in healthcare by implementing DL in anomaly detection. Related to diabetes prediction, our

- proposed model achieved the highest accuracy to date on the PIMA dataset i.e. 98.07%.
- Four data mining algorithms i.e. DT, NB, ANN, and DL are applied on the PIMA dataset for the evaluation of efficiency that is directly proportioned to the accurate decisions. Our proposed method has been chosen based on a task associated with the prophecy of diabetes disease. Rapid miner provides a user-friendly and interactive Graphical user interface for assembling prediction models and pre-processing of data with efficient accuracy in minimal time. Therefore, Rapid miner Studio 9.2.000 has been used in our proposed methodology; it has different features like drag and drop, wisdom of crowds and many more for hands-on suggestions during the workflow. Rapid miner provides 400 additional operators for many data mining aspects which are not available in Weka. These additional 400 operators contain diverse classification techniques, pre-processing methods, validation, and visualization techniques that are not available within Weka. As rapid miner user-interface is very convenient, therefore all the work has been done on this tool which is time-efficient for a researcher when compared to the programming language . Other than the interface rapid miner has more merits which are further illustrated.
- Usability can be considered as one of the first merits of rapidminer because the dataflow in a rapid miner is same as the tree-based structure. It ensures the automatic validations and optimization for large scale data mining which is a bit complicated and difficult in graph-based layout. The second merit of the rapid miner is efficiency as it has been observed by users that rapidminer can handle larger datasets with minimal memory consumption

Figure

Machine learning is an extensive technique of artificial intelligence which studies relationships from data without being programmed explicitly and without defining the prior relationship among the data elements . DL is a form of Machine learning which is different from traditional methods in a way that it learns from various representations of raw data. It allows different computational models that contain several processing layers based on ANN to process and represent data with various abstraction levels .

DL is a multilayer feed-forward perceptron based model which also facilitates the properties of ANN and trained with stochastic gradient descent using back-propagation. The network is a collection of four layers emulating nodes and neurons, directed in uni-direction (one-way connection). Each node is connected to the next node in a single way connection and contains two hidden layers where each node trains a copy of global model parameters by applying its local data.

Further, it uses multiple threads to process the model and apply the averaging for contributing to the model access across the whole network. The learning model uses stochastic gradient descent training using backpropagation and hidden layer's neurons which enable more advance features like tanh, rectifier and maxout activation, learning rate, rate annealing. Among all the activation methods maxout provides the most prominent results. Our proposed model used one Input layer for data entry

Chapter 6: UI

How to build apps with Streamlit Python?

Web-based applications are a great way to display and share data insights. But they often require front-end experience, which takes much effort and time to learn and implement. Using Streamlit, we can easily build beautiful and interactive web apps within minutes, all in Python.

Following this quick Streamlit Python tutorial, you'll learn:

What is Streamlit?

How to add text, media, sidebar?

How to display data?

How to add data visualizations?

How to create interactive widgets?

How to build forms?

How to set up the environment?

Write a basic Streamlit app and Run it

Display and format text

Display data/table

Visualize the data

Add interactive widgets

Control flow with form

Change the layout

Add media content

Cache for better performance

The complete script and Run it through URL

➤ What is Streamlit?

Stream-lit is an open-source (free) Python library, which provides a fast way to build interactive web apps. It is a relatively new package launched in 2019 but has been growing tremendously. It is designed and built especially for machine learning engineers or other data science professionals. Once you are done with your analysis in Python, you can quickly turn those scripts into web apps/tools to share with others.

As long as you can code in Python, it should be straightforward for you to write Stream-lit apps. Imagine the app as the canvas. We can write Python scripts from top to bottom to add all kinds of elements, including text, charts, widgets, tables, etc.

We'll build an example app with Stream-lit in Python. You'll be able to use the common elements and commands from Stream-lit and expand to others on your own!

➤ How to set up the environment?

This Stream-lit Python tutorial will use the PyCharm Editor – Community Edition. It's free and has many useful features for writing Python code. But you can stick to your favorite editor.

We understand that many of you use the Jupyter environment for data analysis. But we should use one of the Python editors when developing web applications. We want to focus on building and running the apps as a whole script. Plus, the editors often offer more features such as syntax check, which help us to code easier.

It's also necessary to use the `pip install stream-lit` command in your terminal to install Stream-lit before using it. After the installation, it is optional to test using the command `stream-lit hello`. An app should open in the web browser if the installation works.

Now we can start writing code to build our first Stream-lit app.

➤ Write a basic Stream-lit app and Run it

After creating a Python script and naming it `stream-lit_example.py`, we can start writing our Stream-lit app.

We import the below libraries. Note that we also put the complete script for the app at the end of the tutorial.

```
import stream-lit as st
import pandas as pd
import plotly.express as px
```

Then, let's also add another line of code:

```
'# Avocado Prices dashboard'
```

This is simply a string with the Markdown format. But we already have a basic Stream-lit app!

Let's run and see it.

Once you've entered and run the below command in the terminal, you should see your first app opening in your web browser.

```
stream-lit run stream-lit_example.py
```

It should look like this – a heading level 1 'Avocado Prices dashboard'.

Avocado prices dashboard

This is the magic feature of Stream-lit. Every time we put a literal value, or a variable on its own line, Stream-lit automatically writes it to our apps. That's why we can display the Markdown string '# Avocado Prices dashboard' without using any Stream-lit commands.

Alternatively, we can use the below code. It will show the same results. Since the Stream-lit magic is using the function `st.write` to display the elements behind the scenes.

```
st.write('# Avocado Prices dashboard')
```

And there's even more `st.write` can do. According to Stream-lit, `st.write` is the 'Swiss Army knife'. It can write various things to the app for us. Besides the Markdown string, `st.write` (or the magic feature) can also display other arguments such as Data-frame, function, and chart.

In short, the magic feature or `st.write` can display almost anything easily! They will inspect the objects we throw in and try to show them most appropriately. So you might be wondering, since `st.write` can display almost everything, why do I need other functions?

Due to its generic nature, `st.write` is limited in customizations and further modifications. So I would recommend trying `st.write` when you are not sure. Then if you've found that you need specific customizations, you can use other functions we'll cover below.

All right! Now you've learned the most basic and powerful concept in Streamlit: the magic and `st.write`. Let's move on to more details.

Display and format text

In the previous section, we displayed a Markdown string using the magic or the `st.write` command. There are also dedicated functions for text.

Let's add some of them to our app using the below code:

- ❖ `st.title`: display text in title formatting. We comment it out next to the previous command since it returns the same level 1 heading.
- ❖ `st.markdown`: display Markdown-formatted string. We ask it to display a description and the data source with a hyperlink. Please ensure there are two spaces after `:avocado:` to break onto the next line. Again, we could put the argument directly and use the magic to display it as well.
- ❖ `st.header`: display text in header formatting. Each document should only have one `st.title`, and multiple `st.headers`. Since `st.header` creates headers that are lower levels than `st.title`.

```
st.write('# Avocado Prices dashboard') #st.title('Avocado Prices dashboard')
st.markdown("""This is a dashboard showing the *average prices* of different types
of :avocado: Data
source:[Kaggle](https://www.kaggle.com/datasets/timmate/avocadoprices2020)""")
st.header('Summary statistics')
st.header('Line chart by geographies')
```

After adding and saving the above new code, you should notice the page with your app has the below options. Stream-lit rerun options

This is because when Stream-lit detects the changes in the script, it asks us whether we want to rerun the app. Let's select the option 'Always rerun' so that the new app will be displayed automatically when we change and save the script. Now you should be able to see the new app. It has a title, some description with Markdown formatting, and two headers. We'll create two sections: one to show the summary statistics, and the other to show a line chart.

Stream lit app with text

Display data/table

We'll start filling out the summary statistics section. As the header suggested, we'll put a data table showing statistics.

So we'll add the below lines of code under this header. We grab the avocado prices data-set and calculate its statistics (the average prices by avocado types), then display the results using st.data-frame. The function st.data-frame displays the Data-frames as interactive tables.

```
st.header('Summary statistics')
avocado = pd.read_csv('avocado.csv')
avocado_stats = avocado.groupby('type')['average_price'].mean()
st.dataframe(avocado_stats)
```

Now, if we go to the browser, you should see the new app. Stream-lit app with Data-frame

Since it's an interactive table, you can try, for example, clicking on the column header average_price and sorting the data by it.

Great!

That's it for the summary statistics section.

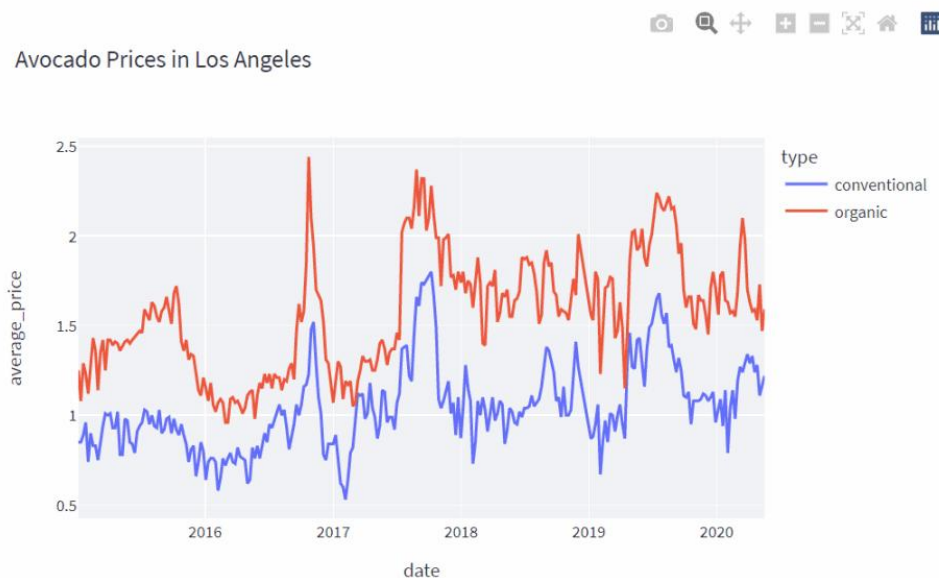
Visualize the data

Now we'll work on the line chart section. Let's first display a simple line chart. There are multiple ways to display charts in Streamlit, such as Streamlit functions, Matplotlib, Altair, and Plotly. We'll use the plotly chart as an example. So please put the below code into your script. We can generate a line figure using the Plotly Express library and then display it using `st.plotly_chart`. And since there are different geographies in the dataset, we use 'Los Angeles' as an example.

```
st.header('Line chart by geographies')
line_fig = px.line(avocado[avocado['geography'] == 'Los Angeles'],
                  x='date', y='average_price',
                  color='type',
                  title='Avocado Prices in Los Angeles')
```

In the new app, you should be able to find the plotly line chart under the 'Line chart by geographies' section.

Line chart by geographies



Stream-lit app with Plotly chart

So this is a line chart for only one geography. We want to explore this same chart from different geographies other than 'Los Angeles'.

How can we do that?

Let's add some interactive widgets to this section.

Add interactive widgets

Now let's make our line chart interactive.

We add two widgets: a select box, and a button. So please replace the code from the last section as below. Please note that in Streamlit, we treat widgets like variables.

```
selected_geography = st.selectbox(label='Geography',
options=avocado['geography'].unique())
submitted = st.button('Submit')
if submitted:
    filtered_avocado = avocado[avocado['geography'] == selected_geography]
    line_fig = px.line(filtered_avocado,
                        x='date', y='average_price',
                        color='type',
                        title=f'Avocado Prices in {selected_geography}')
    st.plotly_chart(line_fig)
```

Let's take a closer look at what we've added:

`st.selectbox`: this widget provides options for unique geographies. Since we use widgets like variables in Streamlit, we assign it to the variable `selected_geography`. Whenever the user selects geography on the select box, its value is stored in such variable

`st.button`: this widget displays a button. Then we use an if statement to define its behavior. If the button is clicked (True), we filter for the dataset being that specific geography chosen on the select box, then create and display its line chart. Now you should be able to see the new widgets within the app. Try to select geography from the select box, and click the button. You should see a line chart displayed like below.

Stream-lit app with widgets

This is good! But there's one thing unusual with it.

Now, if we select another geography from the select box, the line chart for the previous selection disappears. This is because every time we interact with a widget like the select box, the app reruns everything and updates its value. But it is not what we expect from a usual app. The chart should stay there until we click the button to update it.

We'll fix that by containing these elements in a form.

Control flow with form

So let's replace the previous code with the below.
with `st.form('line_chart')`:

```

selected_geography = st.selectbox(label='Geography',
options=avocado['geography'].unique())
submitted = st.form_submit_button('Submit')
if submitted:
    filtered_avocado = avocado[avocado['geography'] == selected_geography]
    line_fig = px.line(filtered_avocado,
                        x='date', y='average_price',
                        color='type',
                        title=f'Avocado Prices in {selected_geography}')
    st.plotly_chart(line_fig)

```

In the above code, we add `st.form`. This function creates a form that batches elements with a ‘Submit’ style button. Whenever that Submit button is clicked in the form, all widgets values in the form will be processed in a batch.

We use the `with` statement to add elements to `st.form`. Within the form, the elements mostly stay the same as before, except for the button. Within the form, we must define it as `st.form_submit_button`, instead of `st.button`.

Now, if we go to our new app, you should see the widgets are grouped visually inside a form.

The line chart will appear again if we select geography and hit the submit button. While as shown in the screenshot below, if we change the selection to another geography, the previous line chart remains until we hit the submit button again. Because now the information is batched and processed together in the form. This is exactly the behavior we are used to!

Streamlit app with form

Change the layout

All right, so our app already has its main content. We can also add something to show extra information about it.

We’ll add a sidebar to the left panel of it.

Please add the below code to your script to display the sidebar. We use the `with` statement again to put elements (a subheader and a Markdown string) into the sidebar.

with `st.sidebar`:

```

st.subheader('About')
st.markdown('This dashboard is made by Just into Data, using **Streamlit**')

```

Now looking at the new app, you should see the sidebar with text.

Streamlit app with sidebar

Add media content

In the end, let’s add some media: an image to our dashboard.

We use `st.sidebar.image` below, which is an alternative to adding `st.image` into the `with` statement of `st.sidebar`. This applies to other elements as well. We can

either add `st.sidebar.element` directly, or add to the with `st.sidebar` statement, `st.element`. We use a Streamlit logo online for the image and set its width to 50.

with `st.sidebar`:

```
st.subheader('About')
st.markdown('This dashboard is made by Just into Data, using Streamlit')
```

```
st.sidebar.image('https://streamlit.io/images/brand/streamlit-mark-color.png',
width=50)
```

You should be able to see such an image displayed within the sidebar.

Streamlit app with image

Cache for better performance

One last thing we'd like to cover in this Streamlit Python tutorial. We can replace the code for loading the dataset with the below function wrapped with the `st.cache` decorator.

```
@st.cache
```

```
def load_data(path):
```

```
    dataset = pd.read_csv(path)
```

```
    return dataset
```

```
avocado = load_data('avocado.csv')
```

When we mark a function with `@st.cache`, Streamlit first runs it and stores the results in a local cache. So that the next time the function is called, Streamlit will skip the execution of the function if nothing has been changed. You can read about the details [here](#).

Within our example, it's fast to load our small dataset. You won't notice any difference. But this is extremely useful if you put really slow operations within the function.

The complete script and Run it through URL

If you haven't gotten the chance to run your app, look [here](#). We have deployed this app on the Streamlit Cloud so that you can interact with it as a user. Select different geographies within the select box and see the updated chart.

Here is the complete script for our app.

```
import streamlit as st
```

```
import pandas as pd
```

```
import plotly.express as px
```

```
st.write('# Avocado Prices dashboard') #st.title('Avocado Prices dashboard')
```

```
st.markdown("""
```

```
This is a dashboard showing the average prices of different types of :avocado:
```



```

Data source: [Kaggle](https://www.kaggle.com/datasets/timmate/avocado-prices-2020)") st.header('Summary statistics')
@st.cache
def load_data(path):
    dataset = pd.read_csv(path)
    return dataset
avocado = load_data('avocado.csv')
avocado_stats = avocado.groupby('type')['average_price'].mean()
st.dataframe(avocado_stats)
st.header('Line chart by geographies')
with st.form('line_chart'):
    selected_geography = st.selectbox(label='Geography',
options=avocado['geography'].unique())
    submitted = st.form_submit_button('Submit')
    if submitted:
        filtered_avocado = avocado[avocado['geography'] == selected_geography]
        line_fig = px.line(filtered_avocado,
                        x='date', y='average_price',
                        color='type',
                        title=f'Avocado Prices in {selected_geography}')
        st.plotly_chart(line_fig)

with st.sidebar:
    st.subheader('About')
    st.markdown('This dashboard is made by Just into Data, using **Streamlit**')
st.sidebar.image('https://stream-lit.io/images/brand/stream-lit-mark-color.png',
width=50)
One last tip here. We can run Stream-lit script stored in an URL. We've saved
the complete script on GitHub. So you can run the command below in your
terminal to launch the example app as well.
streamlit run
https://github.com/liannewriting/streamlit_example/blob/main/streamlit_example.
py

```

The relation between stream-lit python and prediction of diabetes

Stream-lit can be a valuable tool for building the user interface and interactive components of a diabetes prediction project. Here's how Stream-lit can be related to a prediction of diabetes project:

1. Interactive Data Exploration: Stream-lit allows you to create interactive visualizations and widgets to explore and analyze the diabetes-related data. You can use Stream-lit to display charts, histograms, scatter plots, and other visual

2. representations of the data. This can help in understanding the patterns and relationships in the data set and identifying relevant features for the prediction model.

2. Model Evaluation and Visualization: Stream-lit can be used to showcase the performance and evaluation metrics of the diabetes prediction model. You can display accuracy, precision, recall, and other evaluation metrics using Stream-lit components like tables, progress bars, and visualizations. This helps in assessing the effectiveness of the prediction model and communicating the results to stakeholders.

3. User Interface for Inputting Data: Stream-lit provides interactive components like sliders, drop downs, and text inputs that enable users to input relevant information for diabetes prediction, such as patient demographics, medical history, and risk factors. This allows users to input their own data and obtain personalized predictions or explore different scenarios.

4. Real-Time Predictions: Stream-lit supports real-time updates, which can be useful for displaying predictions as users modify input parameters. You can use Stream-lit to connect the prediction model with the user interface, allowing users to see immediate updates in predictions based on the input provided.

5. Deployment and Sharing: Stream-lit allows you to easily deploy your diabetes prediction application as a web app, making it accessible to users without requiring them to run the code locally. This enables easy sharing and collaboration with stakeholders, healthcare professionals, or patients who can access the app through a web browser.

By leveraging Stream-lit in your diabetes prediction project, you can create a user-friendly and interactive interface, showcase model performance, allow for personalized predictions, and easily deploy and share the application with relevant stakeholders. Stream-lit simplifies the process of building and deploying web applications, enhancing the usability and accessibility of your diabetes prediction project.

***some more features and benefits of using Stream-lit:**

_Simple and intuitive API: Stream-lit's API is designed to be simple and intuitive, making it easy for developers to quickly build interactive web applications without having to learn complex web development frameworks.

_Fast prototyping: Stream-lit's fast prototyping capabilities allow developers to quickly experiment with different ideas and iterate on their projects in real-time.

Seamless integration with Python libraries: Stream-lit integrates seamlessly with popular Python libraries such as Pandas, Matplotlib, and Scikit-learn, making it easy to create visualizations and machine learning models.

Interactive components: Stream-lit provides a variety of built-in components such as sliders, text inputs, and drop-down menus, which make it easy to create interactive interfaces that respond to user input.

Sharing: Stream-lit makes it easy to share your applications with others by providing a simple command to deploy your application to the web.

Custom components: Stream-lit allows you to create custom components using Python and JavaScript, giving you the flexibility to add custom functionality and extend the capabilities of the library.

Multiple layout options: Stream-lit provides multiple layout options, including columns, grids, and tabs, allowing you to organize and present your data in a variety of ways.

Responsive design: Stream-lit's responsive design allows your application to automatically adjust to different screen sizes, making it easy to create applications that are optimized for desktop, tablet, and mobile devices.

Open-source: Stream-lit is an open-source library, which means that it is free to use and can be customized and extended by anyone.

Easy deployment: Stream-lit makes it easy to deploy your application to a variety of environments, including local machines, cloud platforms, and containerized environments.

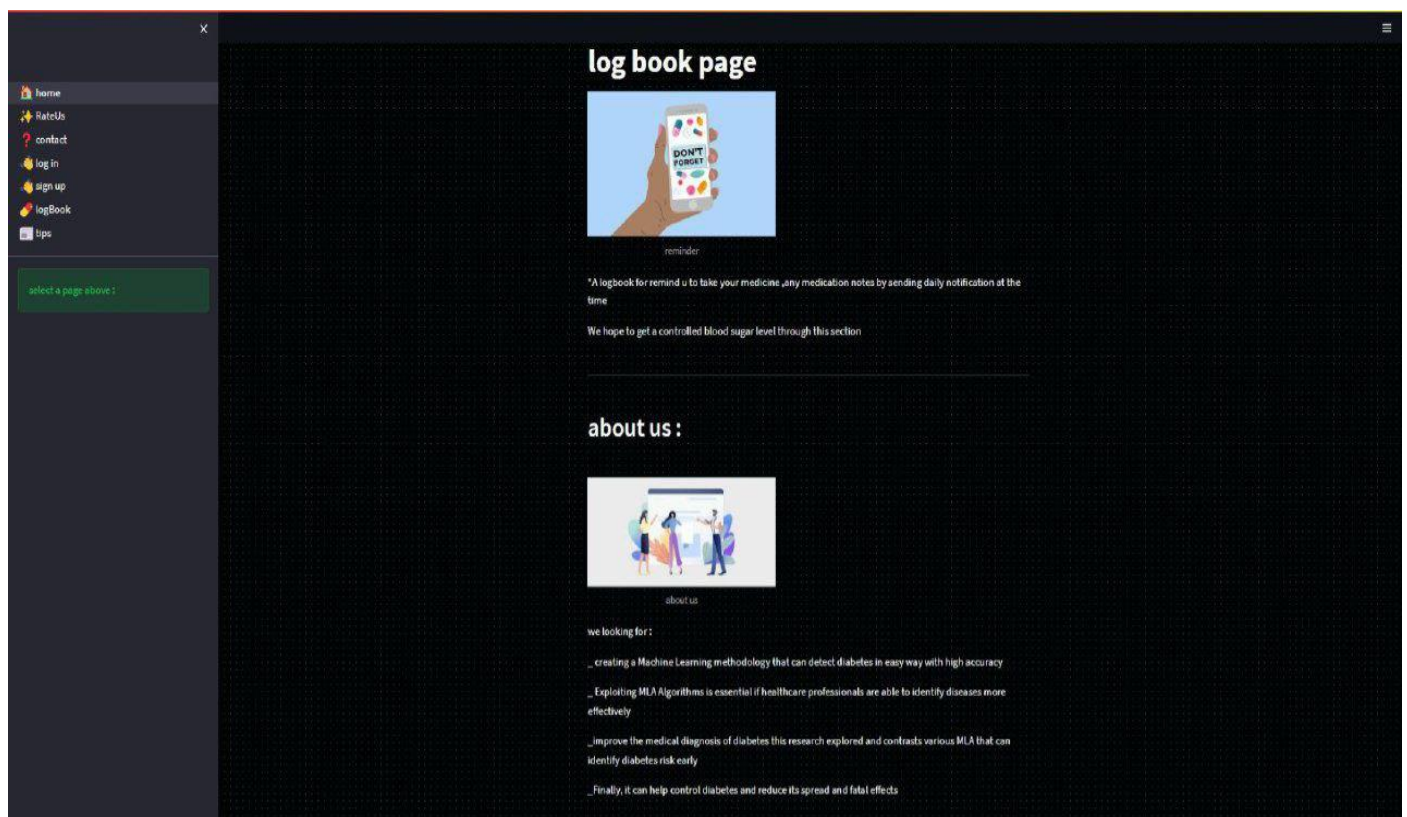
we divided our multipage web application into 7 pages :

1- Home page :

It contains everything the user wants to know about the web application , For what it used, the services that we provide, what do we aspire to through this application, which is improve the medical diagnosis of diabetes this research explored and contrasts various MLA that can identify diabetes risk early ,Helping more people detect diabetes early/or avoid it and help people with diabetes live a healthier life by controlling their diabetes.

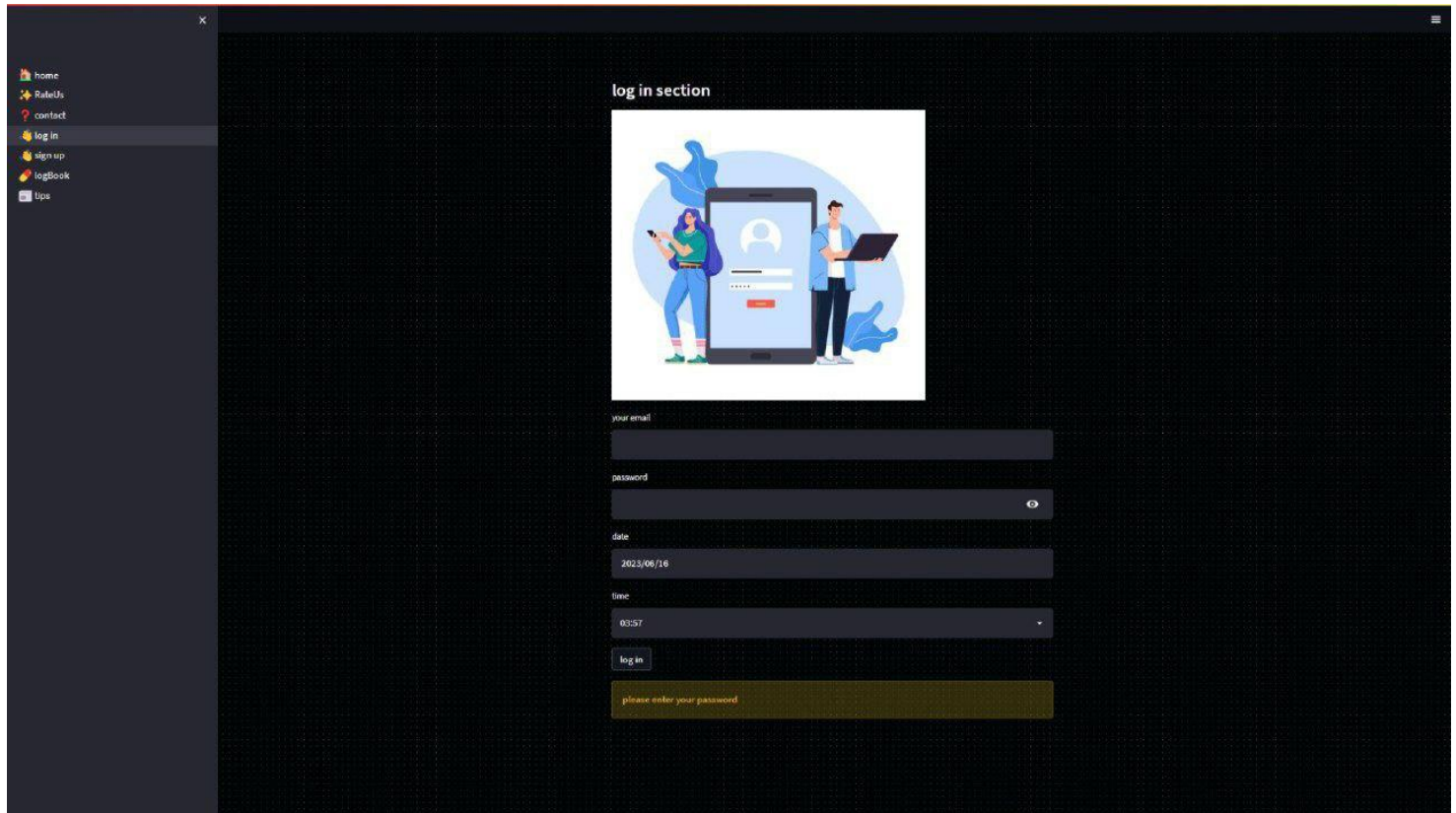
*We have used CSS and HTML to add better services to the application, such as the Contact Us section, to do this wonderful interactive section between the user and developers, which has been linked to a special G-mail for the speed of response to user messages.

*We also used a CSS pattern to add a different patterned background to our web application .



2-log in section:

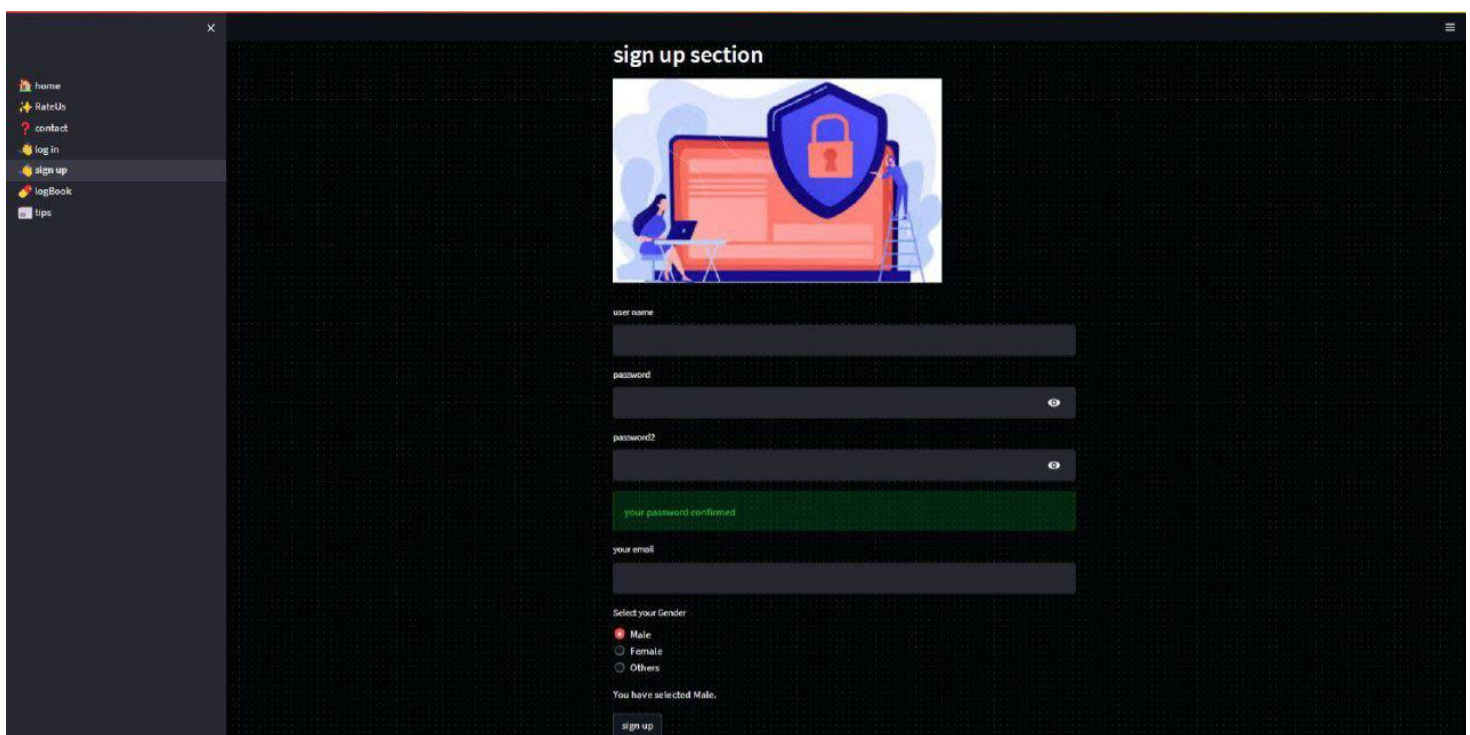
This section enables the registered user who already has an account to log in , detect diabetes and enjoy other services easily .



The screenshot displays the 'log in section' of a web application. On the left, a dark sidebar contains a menu with icons and labels for 'home', 'RateUs', 'contact', 'log in', 'sign up', 'logBook', and 'tips'. The 'log in' option is highlighted. The main content area has a dark background. At the top, the text 'log in section' is displayed above an illustration of two people interacting with a large smartphone. Below the illustration, there are input fields for 'your email', 'password', 'date' (with a date picker showing '2023/06/16'), and 'time' (with a time picker showing '03:57'). A 'log in' button is positioned below these fields. A green error message 'please enter your password' is visible at the bottom of the form.

3-sign up section :

This section enables the new user to create an account with us and save the data of this account,detect diabetes and use our services.



The screenshot displays the 'sign up section' of a web application. On the left, a dark sidebar contains a menu with icons and labels for 'home', 'RateUs', 'contact', 'log in', 'sign up', 'logBook', and 'tips'. The 'sign up' option is highlighted. The main content area has a dark background. At the top, the text 'sign up section' is displayed above an illustration of a person sitting at a desk with a laptop, with a large shield icon in the background. Below the illustration, there are input fields for 'user name', 'password', 'password2', and 'your email'. A green success message 'your password confirmed' is displayed above the 'your email' field. Below the email field, there is a section for 'Select your Gender' with radio buttons for 'Male', 'Female', and 'Others'. The 'Male' option is selected, and a message 'You have selected Male.' is shown. A 'sign up' button is at the bottom of the form.

4-log book section:

This section enables the user to enjoy the service of sending reminders notification with the name and time of the medication that user determined (whether it is related to diabetes or not), as well as adding any necessary medical notes daily .

log book section

name of the medicine

time of medicine daily

21:26

any medication notes u want to remind

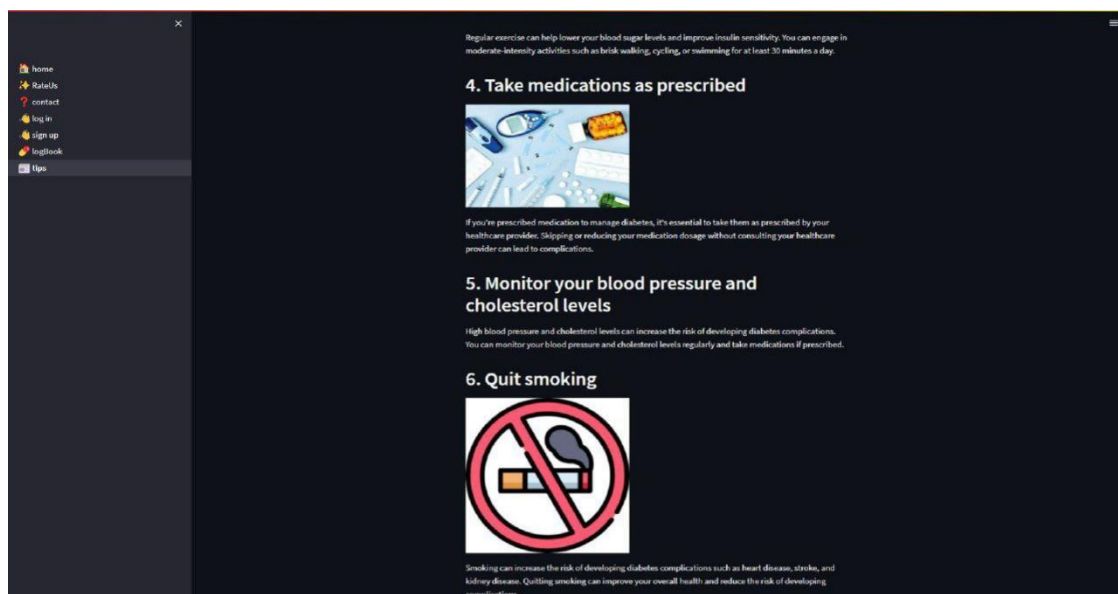
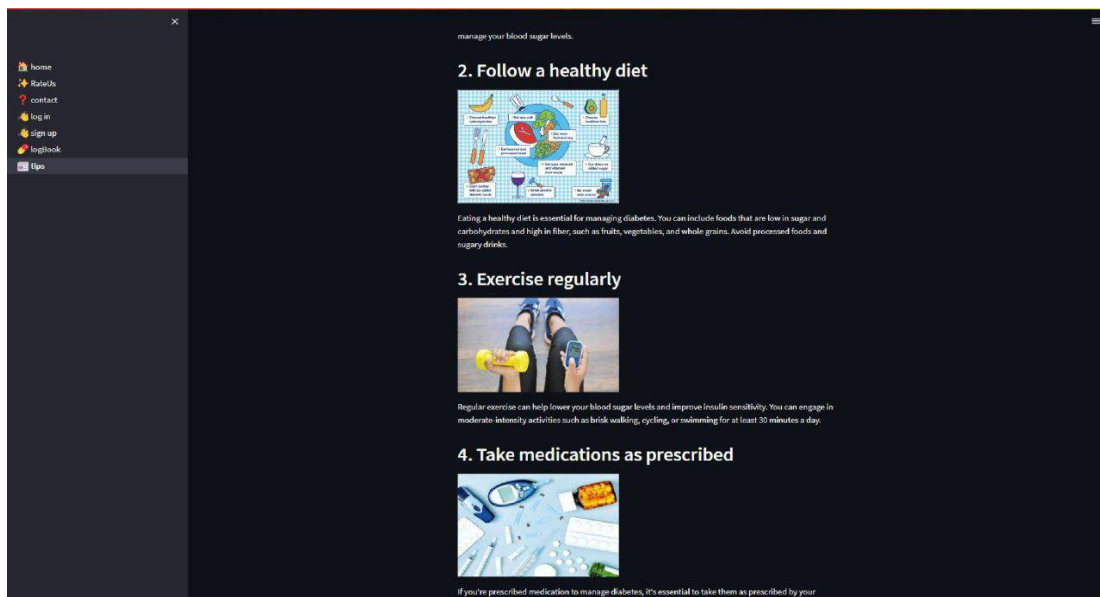
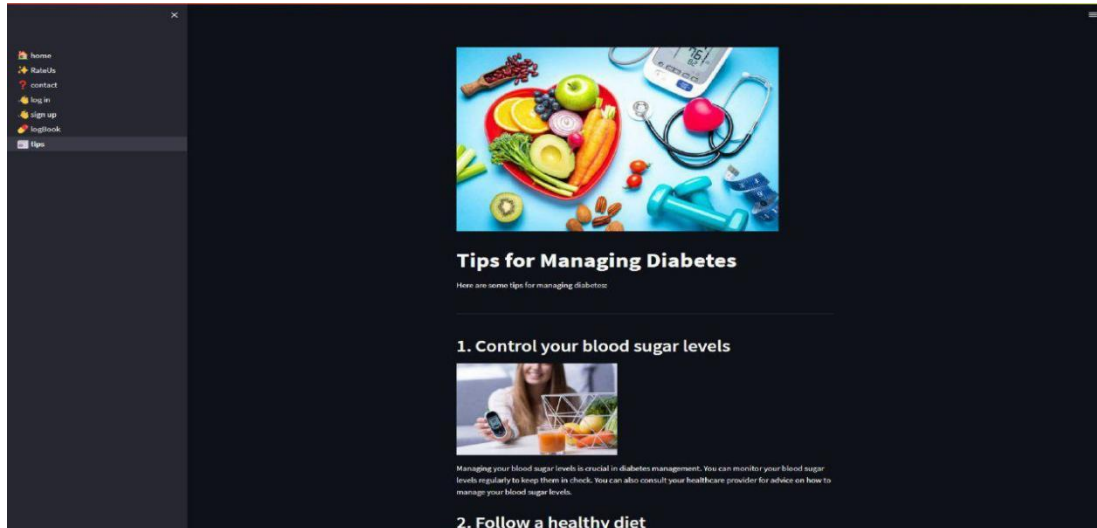
confirm

please enter your medicine name and time to remind u

Made with Streamlit

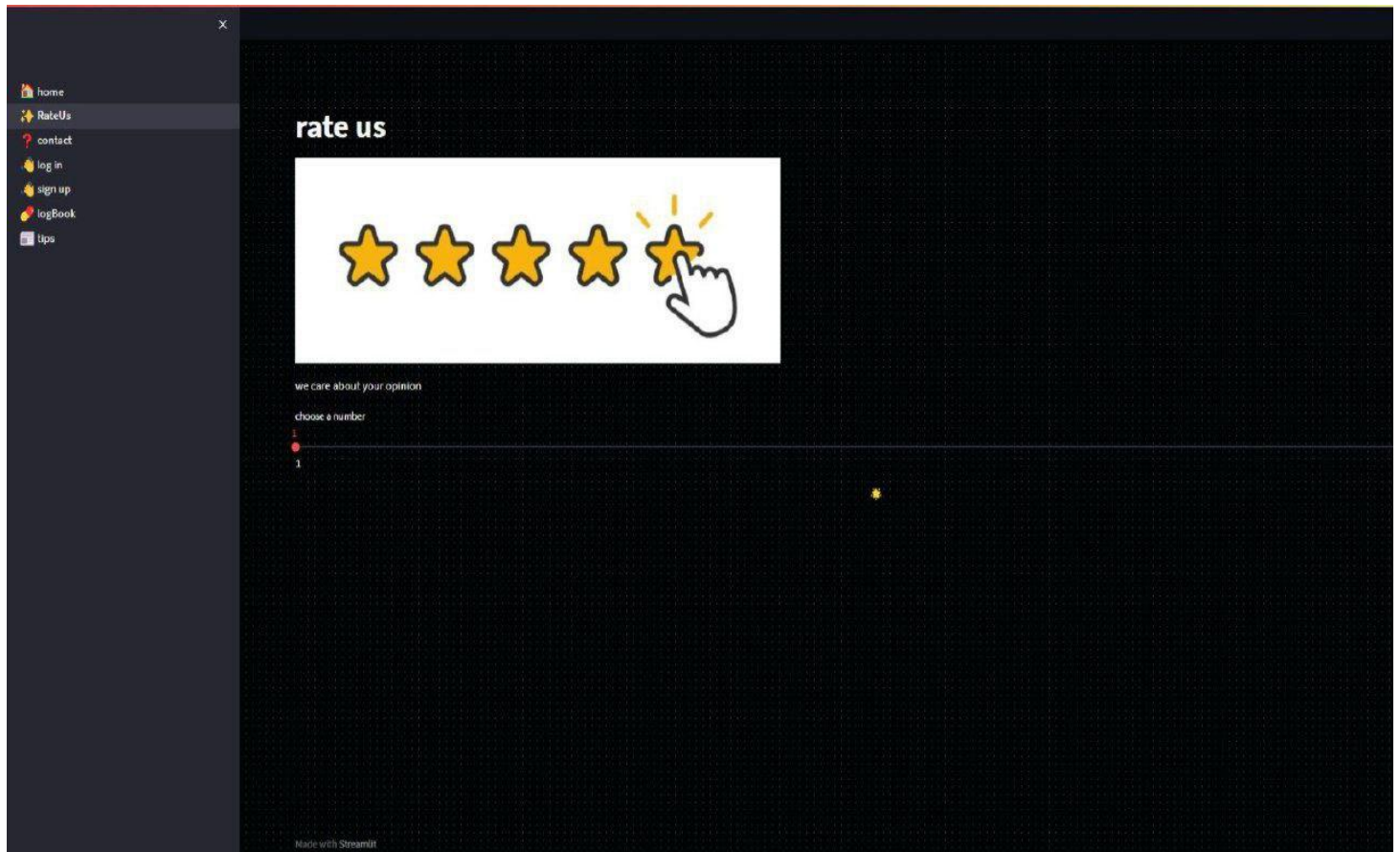
5-Tips section :

- This section includes tips to help diabetics live with the disease and control blood sugar levels through a healthy lifestyle
- It also contains common general symptoms of diabetes, which necessitate using the application / visiting a doctor



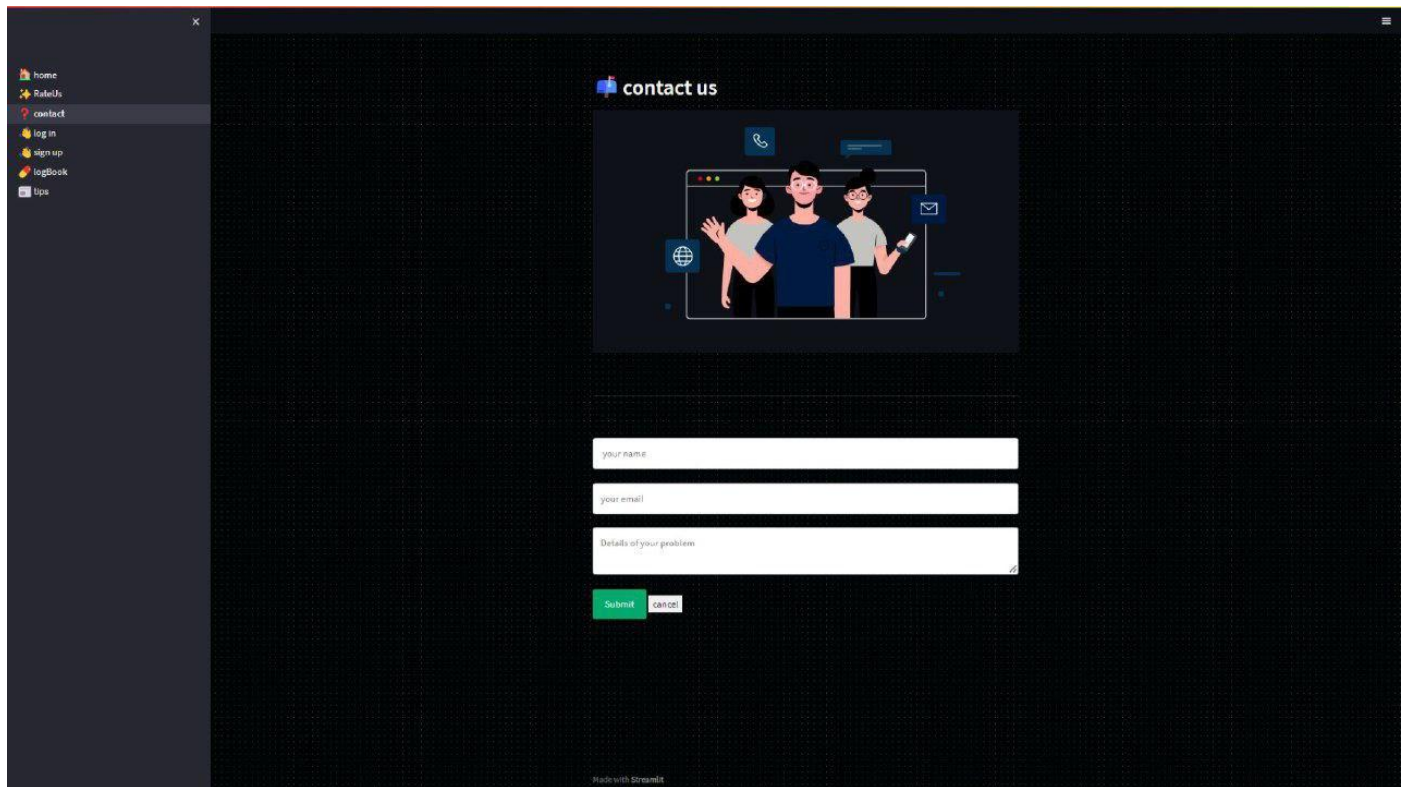
6-Rrate us section :

This section allows the user to rate our web application as 1:5 stars to help us improve our services



7-Contact us section:

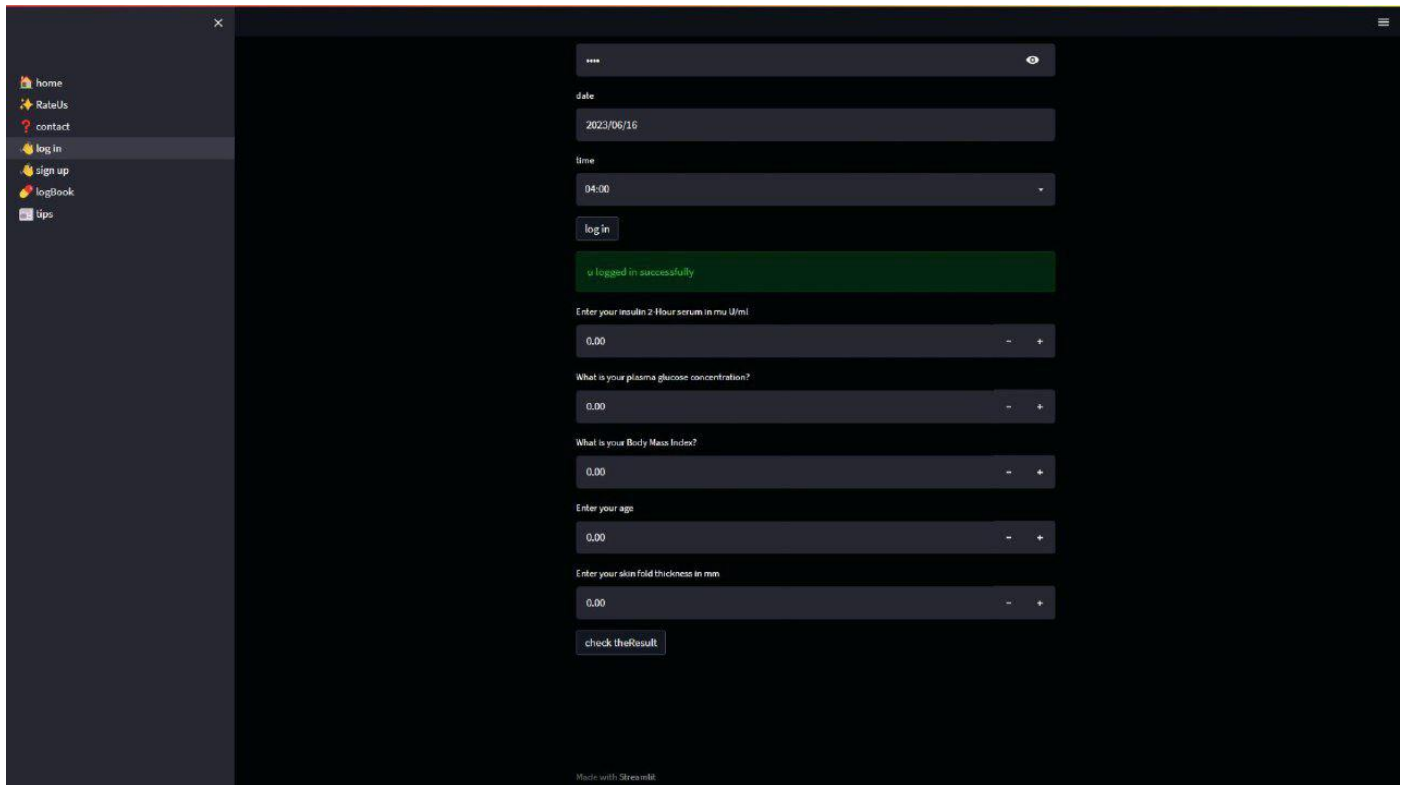
This section enables the user to send complaints/suggestions to those in charge of the application as instant messages to our Gmail



The screenshot displays a web application interface with a dark theme. On the left, a sidebar menu contains links: home, RateUs, contact (highlighted), log in, sign up, logBook, and tips. The main content area is titled 'contact us' and features an illustration of three people in a video call window. Below the illustration are three input fields: 'your name', 'your email', and 'Details of your problem'. At the bottom of the form are two buttons: 'Submit' (green) and 'cancel' (white). The footer of the page reads 'Made with Elementor'.

Diagnosis feature

The prediction for diabetes is depend on (pregnancies - glucose level - blood pressure - skin thickness - insulin level - BMI- Diabetes Pedigree Function)



The screenshot shows a Streamlit web application interface for diabetes diagnosis. On the left, there is a dark sidebar with navigation links: home, RateUs, contact, log in, sign up, logBook, and tips. The main content area has a dark background and contains several input fields and buttons. At the top, there is a text input field for 'name' with a visibility toggle. Below it is a 'date' input field showing '2023/06/16'. Then, a 'time' input field showing '04:00'. A 'log in' button is present, followed by a green success message: 'u logged in successfully'. Below this, there are five input fields with numerical values, each with minus and plus buttons for adjustment: 'Enter your insulin 2-Hour serum in mu U/ml' (0.00), 'What is your plasma glucose concentration?' (0.00), 'What is your Body Mass Index?' (0.00), 'Enter your age' (0.00), and 'Enter your skin fold thickness in mm' (0.00). At the bottom of the form is a 'check theResult' button. The footer of the application says 'Made with Streamlit'.

Overall, Stream-lit is a powerful tool that provides a variety of features and benefits to developers who want to create interactive web applications for data science and machine learning projects. Whether you are a data scientist, machine learning practitioner, or a developer with a background in Python, Streamlit makes it easy to create and deploy web applications that are fast, scalable, and secure, while providing a variety of customization options and a large and active community of developers.

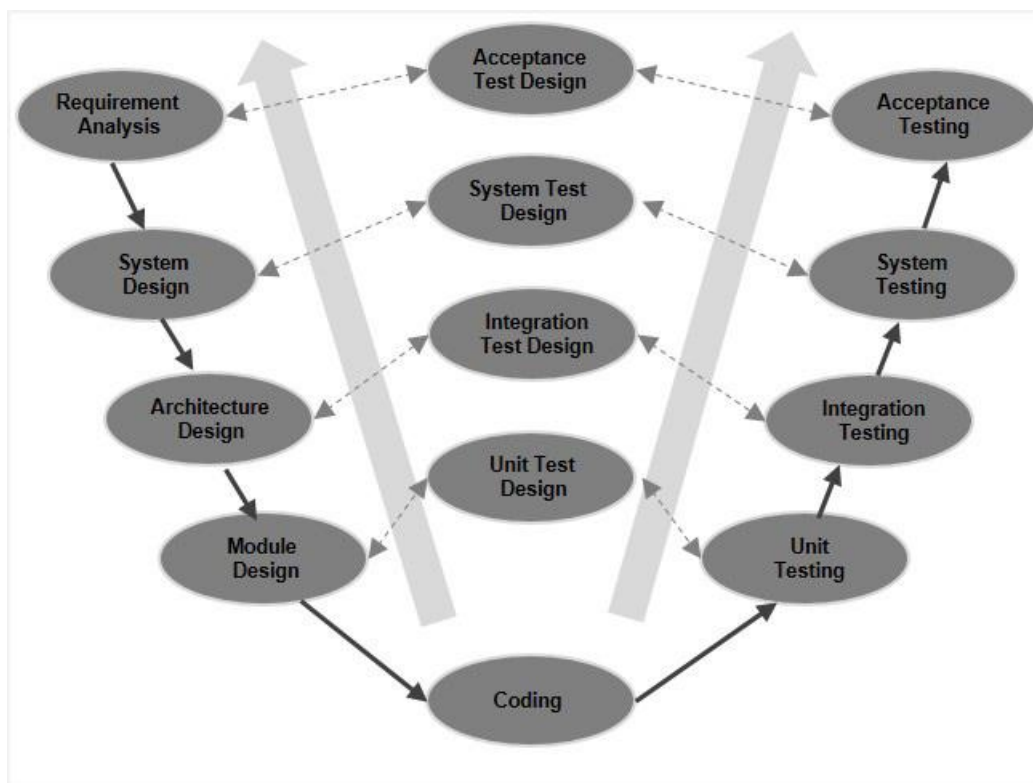
CHAPTER 7:SYSTEM TESTING

7.1 Introduction

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. In fact, testing is the one step in the software engineering process that could be viewed as destructive rather than constructive.

A strategy for software testing integrates software test case design methods into a well-planned series of steps that result in the successful construction of software. Testing is the set of activities that can be planned in advance and conducted systematically. The underlying motivation of program testing is to affirm software quality with methods that can economically and effectively apply to both strategic to both large and small-scale systems.

7.2 Strategic approach to software testing



Validation Phases

Unit Testing

Unit tests designed in the module design phase are executed on the code during this validation phase. Unit testing is the testing at code level and helps eliminate bugs at an early stage, though all defects cannot be uncovered by unit testing.

White box testing ensures that:

- All independent paths have been exercised at least once
- All logical decisions have been exercised on their true and false sides
- All loops are executed at their boundaries and within their operational bounds

Integration Testing

Integration testing is associated with the architectural design phase. Integration tests are performed to test the coexistence and communication of the internal modules within the system.

System Testing

System testing is directly associated with the system design phase. System tests check the entire system functionality and the communication of the system under development with external systems. Most of the software and hardware compatibility issues can be uncovered during this system test execution.

System testing

Acceptance Testing

Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment. Acceptance tests uncover the compatibility issues with the other systems available in the user environment. It also discovers the non-functional issues such as load and performance defects in the actual user environment.

BLACK BOX TESTING. Testing without knowledge of the internal workings of the item being tested. Tests are usually functional.

7.3Steps of system testing

System testing for a prediction of diabetes involves testing the entire system as a whole to ensure that all components work together correctly and meet the desired requirements. Here's an overview of the system testing process for such a project:

1. **Test Plan Development:** Create a comprehensive test plan that outlines the objectives, scope, and approach for system testing. Define the test cases, test scenarios, and test data that will be used to evaluate the system's functionality and performance.
2. **Test Environment Setup:** Set up the test environment, including the required hardware, software, and database configurations. Install and configure the necessary dependencies and ensure that the environment closely mimics the production environment.
3. **Functional Testing:** Perform functional testing to validate that the prediction system functions as expected. Test various scenarios and inputs to verify that the predictions are accurate and reliable. This includes testing different combinations of patient demographics, medical history, and risk factors to assess the system's ability to provide accurate predictions.
4. **Integration Testing:** Conduct integration testing to ensure that the prediction system integrates seamlessly with the underlying components, such as the database, prediction model, and user interface. Verify that the data flows correctly between the components and that the system as a whole behaves as intended.
5. **Performance Testing:** Evaluate the performance of the system under expected loads and stress conditions. Measure response times, resource utilization, and scalability to ensure that the system can handle the anticipated user load and provide predictions within acceptable time frames.
6. **Usability Testing:** Assess the usability and user-friendliness of the prediction system. Involve users or stakeholders in the testing process to gather feedback on the user interface, input mechanisms, and overall user experience. Identify any usability issues or areas for improvement and make necessary adjustments.

7. **Security Testing:** Conduct security testing to identify vulnerabilities and ensure that the prediction system is secure. Test for common security risks, such as injection attacks, data breaches, and unauthorized access. Implement appropriate security measures, such as encryption of sensitive data and user authentication, to protect the system and user information.
8. **Error Handling and Exception Testing:** Verify that the system handles errors and exceptions gracefully. Test various error scenarios, such as invalid inputs or system failures, to ensure that the system provides appropriate error messages, recovers gracefully, and maintains data integrity.
9. **Compatibility Testing:** Validate the compatibility of the prediction system with different browsers, operating systems, and devices. Test the system on a variety of platforms to ensure a consistent user experience and functionality across different environments.
10. **Documentation and Reporting:** Document the testing process, including test cases, results, and any identified issues or defects. Provide a comprehensive report summarizing the system testing efforts, outcomes, and recommendations for improvement.
11. **Boundary Testing:** Test the system with boundary values to ensure that it handles extreme or edge cases correctly. For example, test with the minimum and maximum values for input parameters such as age, blood glucose levels, or body mass index (BMI). Verify that the system produces accurate predictions and handles boundary conditions appropriately.
12. **Stress Testing:** Perform stress testing to assess the system's performance and stability under high loads or peak usage conditions. Generate a significant number of concurrent requests or simulate a large dataset to evaluate how the system handles increased traffic and resource demands. Monitor performance metrics, such as response times and resource utilization, to identify any bottlenecks or performance degradation.
13. **Regression Testing:** Conduct regression testing to ensure that new system updates or changes do not introduce regressions or unexpected issues. Re-run previously executed test cases to verify that existing functionality remains intact after implementing new features or modifications. This helps to identify any unintended side effects and maintain the overall quality and stability of the system.

14. **Data Integrity Testing:** Validate the integrity of the data stored in the system's database. Verify that the system accurately stores and retrieves data, maintains data consistency, and handles data updates and deletions correctly. Test scenarios such as updating patient records, adding new data points, or removing records to ensure the integrity and reliability of the data.

15. **Compatibility Testing:** Expand compatibility testing to include different versions of dependencies, libraries, and frameworks used in the prediction system. Verify that the system functions correctly with different versions of software components and ensure backward compatibility with previous versions.

16 **Load Testing:** Evaluate the system's performance under sustained loads for an extended period. Simulate realistic user scenarios and continuously monitor system performance to identify any performance degradation, memory leaks, or resource exhaustion over time. This helps to validate the system's stability and performance under prolonged usage.

17. **User Acceptance Testing (UAT):** Involve end-users or stakeholders in user acceptance testing to validate that the system meets their requirements and expectations. Gather feedback, evaluate user satisfaction, and address any usability or functionality concerns raised during the UAT phase.

18. **Test Automation:** Consider automating the execution of test cases using appropriate test automation frameworks and tools. This helps to streamline the testing process, increase test coverage, and facilitate regression testing during future updates or releases.

Remember to document and track any issues or defects encountered during system testing and ensure they are appropriately addressed before the system is deployed. System testing plays a critical role in ensuring the reliability, performance, and accuracy of the prediction system, and it helps to deliver a high-quality product to end-users.

By conducting thorough system testing, you can ensure that the prediction of diabetes project functions correctly, performs well under different conditions, and provides accurate and reliable predictions. Testing helps in identifying and addressing any issues or vulnerabilities before the system is deployed and used in a production environment.

Important of Software Testing

If there are any bugs or errors in the software, it can be identified early and can be solved before delivery of the software product. Properly tested software product ensures reliability, security and high performance which further results in time saving, cost effectiveness and customer satisfaction.

Testing is important because software bugs could be expensive or even dangerous. Software bugs can potentially cause monetary and human loss, and history is full of such examples.

In April 2015, Bloomberg terminal in London crashed due to software glitch affected more than 300,000 traders on financial markets. It forced the government to postpone a 3bn pound debt sale.

Nissan cars recalled over 1 million cars from the market due to software failure in the airbag sensory detectors. There has been reported two accident due to this software failure.

Starbucks was forced to close about 60 percent of stores in the U.S and Canada due to software failure in its POS system. At one point, the store served coffee for free as they were unable to process the transaction.

Some of Amazon's third-party retailers saw their product price is reduced to 1p due to a software glitch. They were left with heavy losses.

Vulnerability in Windows 10. This bug enables users to escape from security sandboxes through a flaw in the win32k system.

In 2015 fighter plane F-35 fell victim to a software bug, making it unable to detect targets correctly.

China Airlines Airbus A300 crashed due to a software bug on April 26, 1994, killing 264 innocents live

In 1985, Canada's Therac-25 radiation therapy machine malfunctioned due to software bug and delivered lethal radiation doses to patients, leaving 3 people dead and critically injuring 3 others.

In April of 1999, a software bug caused the failure of a \$1.2 billion military satellite launch, the costliest accident in history

In May of 1996, a software bug caused the bank accounts of 823 customers of a major U.S. bank to be credited with 920 million US dollars.

Here are the benefits of using software testing:

Cost-Effective: It is one of the important advantages of software testing. Testing any IT project on time helps you to save your money for the long term. In case if the bugs caught in the earlier stage of software testing, it costs less to fix.

Security: It is the most vulnerable and sensitive benefit of software testing. People are looking for trusted products. It helps in removing risks and problems earlier.

Product quality: It is an essential requirement of any software product. Testing ensures a quality product is delivered to customers.

Customer Satisfaction: The main aim of any product is to give satisfaction to their customers. UI/UX Testing ensures the best user experience.

Errors which Software Testing can discovered

Software testing can help discover various types of errors or defects in a software application. Here are some common errors that can be identified through testing:

1. **Functional Errors:** These errors occur when the software does not perform its intended functions correctly or fails to meet the specified requirements.

Functional testing helps uncover such errors and ensures that the software behaves as expected.

2. **Performance Issues:** Testing can reveal performance-related errors such as slow response times, excessive memory usage, or bottlenecks in the software. Performance testing helps identify and address these issues, ensuring that the software performs well under various load and stress conditions.

3. **Security Vulnerabilities:** Testing can uncover security vulnerabilities, including potential loopholes, unauthorized access points, or weaknesses in encryption and authentication mechanisms. Security testing helps identify and mitigate these risks, protecting the software and its data from potential breaches.

4. **User Interface (UI) and User Experience (UX) Problems:** Testing can uncover UI/UX issues such as layout problems, inconsistent design elements, poor usability, or confusing navigation. Usability testing and user acceptance testing help ensure that the software provides a smooth and intuitive user experience.

5. **Compatibility Issues:** Testing can reveal compatibility problems with different operating systems, browsers, hardware configurations, or third-party integrations. Compatibility testing ensures that the software works correctly across various environments and platforms.

6. Data Integrity and Validation Errors: Testing can identify errors related to data input, processing, storage, and output. This includes issues like incorrect data validation, data corruption, data loss, or improper data transformation. Data-related testing helps ensure the accuracy and integrity of data within the software.

7. Error Handling and Exception Management: Testing can help uncover errors related to error handling, exception management, and recovery mechanisms. This ensures that the software handles unexpected scenarios gracefully and avoids system failures or crashes.

SYSTEM SECURITY

refers to the technical innovations and procedures applied to the hardware and operation systems to protect against deliberate or accidental damage from a defined threat.

DATA Security

is the protection of data from loss, disclosure, modification and destruction.

SYSTEM INTEGRITY

refers to the proper functioning of hardware and programs, appropriate physical security and safety against external threats such as eavesdropping and wiretapping.

PRIVACY

defines the rights of the user or organizations to determine what information they are willing to share with or accept from others and how the organization can be protected against unwelcome, unfair or excessive dissemination of information about it.

CONFIDENTIALITY

is a special status given to sensitive information in a database to minimize the possible invasion of privacy. It is an attribute of information that characterizes its need for protection.

These are just a few examples of errors that software testing can help discover. Testing plays a crucial role in ensuring software quality, reliability, and user satisfaction by uncovering and addressing these types of errors before the software is released to users.

Chapter 8: System Security

System security is crucial for this project to protect sensitive data, ensure confidentiality, and prevent unauthorized access. Here are some key considerations for ensuring system security in such a project:

1. **Data Encryption:** Implement encryption mechanisms to protect sensitive data stored in the system's database. Use industry-standard encryption algorithms to encrypt data at rest and in transit. This includes encrypting personal health information (PHI), medical records, and any other sensitive data elements.
2. **User Authentication and Authorization:** Implement robust user authentication mechanisms to ensure that only authorized users can access the system. Utilize strong password policies, multi-factor authentication (MFA), or other authentication methods to validate user identities. Implement role-based access control (RBAC) to restrict access to sensitive features and data based on user roles and permissions.
3. **Secure Communication:** Secure the communication channels between the user interface and the server by using secure protocols such as HTTPS. Encrypt data transmission to prevent unauthorized interception or tampering of sensitive information during transit.
4. **Input Validation:** Validate and sanitize all user inputs to prevent common security vulnerabilities, such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF). Implement input validation and sanitization techniques to ensure that user-supplied data does not compromise the system's security or integrity.
5. **Regular Security Updates:** Keep all software components, frameworks, and libraries up to date with the latest security patches and updates. Regularly review and update the system's dependencies to address any known vulnerabilities.
6. **Security Auditing and Logging:** Implement logging mechanisms to track and monitor system activities, user actions, and access attempts. Maintain audit logs to identify and investigate any suspicious activities or security breaches. Regularly review and analyze the logs for potential security issues.

7. Secure Deployment: Securely deploy the system on a trusted and hardened infrastructure. Follow best practices for server configuration, firewall settings, and access controls to minimize the system's exposure to potential threats. Utilize secure hosting environments or cloud providers that offer robust security measures.

8. Regular Security Assessments: Conduct periodic security assessments and penetration testing to identify and address vulnerabilities in the system. Engage security professionals to perform comprehensive security audits and penetration tests to ensure the system's resilience against attacks.

9. Employee Training and Awareness: Provide security training and awareness programs for employees working on the project. Educate them about secure coding practices, data handling guidelines, and the importance of maintaining security protocols. Promote a culture of security consciousness throughout the development and maintenance processes.

10. Compliance with Regulations: Ensure that the system complies with relevant data protection and privacy regulations, such as the General Data Protection Regulation (GDPR) or the Health Insurance Portability and Accountability Act (HIPAA), depending on the jurisdiction and nature of the data being handled.

11. Role-Based Access Control (RBAC): Implement fine-grained access control based on user roles and responsibilities. Restrict access to sensitive functionalities and data based on user roles, ensuring that each user has the appropriate level of access privileges.

12. Secure Password Storage: Store user passwords securely by using strong hashing algorithms (such as bcrypt or Argon2) with salted hashes. Avoid storing plain-text passwords or weakly hashed passwords, as they can be vulnerable to password-related attacks.

13. Session Management: Implement secure session management techniques to ensure that session tokens or cookies are protected against unauthorized access or tampering. Enforce session timeouts to automatically terminate inactive sessions and require re-authentication.

14. Cross-Site Scripting (XSS) Prevention: Apply input sanitization and output encoding techniques to prevent cross-site scripting attacks. Validate and sanitize

user inputs to remove or neutralize potentially malicious code, and encode output to prevent unintended script execution.

15. Cross-Site Request Forgery (CSRF) Protection: Implement CSRF tokens to protect against CSRF attacks. Include unique tokens with each request and validate them on the server-side to ensure that requests are legitimate and originated from the intended user.

16. Secure Error Handling: Be cautious with error messages displayed to users, as they can potentially reveal sensitive information. Ensure that error messages are generic and do not disclose system details or sensitive data that could aid attackers.

17. Security Testing and Code Reviews: Perform regular security testing and code reviews to identify vulnerabilities and security flaws in the system. Utilize automated security testing tools and manual code reviews to identify common security issues and potential vulnerabilities.

18. Secure Third-Party Integrations: If the project involves integrating third-party services or APIs, ensure that proper security measures are implemented. Validate the security practices of the third-party providers and ensure that data transmitted to and from these services is encrypted and protected.

19. Data Backup and Recovery: Implement regular data backups to prevent data loss and ensure business continuity. Store backups in secure locations, preferably off-site or in encrypted storage, and regularly test the restoration process to ensure data can be recovered if needed.

20. Continuous Monitoring and Incident Response: Set up systems for continuous monitoring of the application and infrastructure to detect any potential security breaches or abnormal activities. Establish an incident response plan to address security incidents promptly and effectively, including procedures for investigation, containment, mitigation, and recovery.

Remember that security is an ongoing process, and it's important to stay updated on the latest security best practices, vulnerabilities, and mitigation techniques. Regularly assess and update security measures to adapt to evolving threats and protect the prediction of diabetes project from potential security risks.

By incorporating robust security measures into the prediction of diabetes project, you can safeguard sensitive data, protect the system against threats, and maintain the privacy and confidentiality of user information. It is essential to consider security throughout the development life-cycle and regularly update security measures to address emerging threats and vulnerabilities.

Security engineering is an area of increasing emphasis in the defense domain. Baldwin et al. (2012) provide a survey of the issues and a detailed reference list. Purpose of System Security (INCOSE Systems Security Working Group 2020; Used with Permission of Keith Willett)

The purpose of system security is to help assure system value-delivery while undergoing adversity. The primary objective of System Security Engineering (SSE) is to minimize or contain defense system vulnerabilities to known or postulated security threats and to ensure that developed systems protect against these threats. Engineering principles and practices are applied during all system development phases to identify and reduce these system vulnerabilities to the identified system threats.

The basic premise of SSE is recognition that an initial investment in “engineering out” security vulnerabilities and “designing-in” countermeasures is a long-term benefit and cost saving measure. Further, SSE provides a means to ensure adequate consideration of security requirements, and, when appropriate, that specific security-related designs are incorporated into the overall system design during the engineering development program. Security requirements include: physical; personnel; procedural; emission; transmission; cryptographic; communications; operations; and, computer security.

There may be some variation in the SSE process from program to program, due mainly to the level of design assurance—that is, ensuring that appropriate security controls have been implemented correctly as planned—required of the contractor. These assurance requirements are elicited early in the program (where they can be adequately planned), implemented, and verified in due course of the system development.

The System Security Engineering Management Plan (SSEMP) is a key document to develop for SSE. The SSEMP identifies the planned security tasks for the

program and the organizations and individuals responsible for security aspects of the system. The goals of the SSEMP are to ensure that pertinent security issues are raised at the appropriate points in the program, to ensure adequate precautions are taken during design, implementation, test, and fielding, and to ensure that only an acceptable level of risk is incurred when the system is released for fielding. The SSEMP forms the basis for an agreement with SSE representing the developer, the government program office, the certifier, the accreditor, and any additional organizations that have a stake in the security of the system. The SSEMP identifies the major tasks for certification & accreditation (C&A), document preparation, system evaluation, and engineering; identifies the responsible organizations for each task; and presents a schedule for the completion of those tasks.

SSE security planning and risk management planning includes task and event planning associated with establishing statements of work and detailed work plans as well as preparation and negotiation of SSE plans with project stakeholders. For each program, SSE provides the System Security Plan (SSP) or equivalent. An initial system security Concept of Operations (CONOPS) may also be developed. The SSP provides: the initial planning of the proposed SSE work scope; detailed descriptions of SSE activities performed throughout the system development life cycle; the operating conditions of the system; the security requirements; the initial SSE risk assessment (includes risks due to known system vulnerabilities and their potential impacts due to compromise and/or data loss); and, the expected verification approach and validation results. These plans are submitted with the proposal and updated as required during engineering development. In the case where a formal C&A is contracted and implemented, these plans comply with the government's C&A process, certification responsibilities, and other agreement details, as appropriate. The C&A process is the documented agreement between the customer and contractor on the certification boundary. Upon agreement of the stakeholders, these plans guide SSE activities throughout the system development life cycle.

System security describes the controls and safeguards that an organization takes to ensure its networks and resources are safe from downtime, interference or malicious intrusion.

The security of a computer system is a crucial task. It is a process of ensuring the confidentiality and integrity of the OS. Security is one of most important as well as the major task in order to keep all the threats or other malicious tasks or attacks or program away from the computer's software system.

A system is said to be secure if its resources are used and accessed as intended

under all the circumstances, but no system can guarantee absolute security from several of various malicious threats and unauthorized access.

The security of a system can be threatened via two violations:

Threat: A program that has the potential to cause serious damage to the system.

Attack: An attempt to break security and make unauthorized use of an asset.

Security violations affecting the system can be categorized as malicious and accidental threats. Malicious threats, as the name suggests are a kind of harmful computer code or web script designed to create system vulnerabilities leading to back doors and security breaches. Accidental Threats, on the other hand, are comparatively easier to be protected against.

Every computer system and software design must handle all security risks and implement the necessary measures to enforce security policies. At the same time, it's critical to strike a balance because strong security measures might increase costs while also limiting the system's usability, utility, and smooth operation. As

A result, system designers must assure efficient performance without compromising security.

In this article, you will learn about operating system security with its issues and other features.

What is Operating System Security?

The process of ensuring OS availability, confidentiality, integrity is known as operating system security. OS security refers to the processes or measures taken to protect the operating system from dangers, including viruses, worms, malware, and remote hacker intrusions. Operating system security comprises all preventive-control procedures that protect any system assets that could be stolen, modified, or deleted if OS security is breached.

Security refers to providing safety for computer system resources like software, CPU, memory, disks, etc. It can protect against all threats, including viruses and unauthorized access. It can be enforced by assuring the operating system's integrity, confidentiality, and availability. If an illegal user runs a computer application, the computer or data stored may be seriously damaged.

The goal of Security System

There are several goals of system security. Some of them are as follows:

1. Integrity

Unauthorized users must not be allowed to access the system's objects, and users with insufficient rights should not modify the system's critical files and resources.

2. Secrecy

The system's objects must only be available to a small number of authorized users. The system files should not be accessible to everyone.

3. Availability

All system resources must be accessible to all authorized users, i.e., no single user/process should be able to consume all system resources. If such a situation arises, service denial may occur. In this case, malware may restrict system resources and preventing legitimate processes from accessing them.

How to ensure Operating System Security?

There are various ways to ensure operating system security. These are as follows:

Authentication

The process of identifying every system user and associating the programs executing with those users is known as authentication. The operating system is responsible for implementing a security system that ensures the authenticity of a user who is executing a specific program. In general, operating systems identify and authenticate users in three ways.

1. Username/Password

Every user contains a unique username and password that should be input correctly before accessing a system.

3. User Attribution

These techniques usually include biometric verification, such as fingerprints, retina scans, etc. This authentication is based on user uniqueness and is compared to database samples already in the system. Users can only allow access if there is a match.

3. User card and Key

To login into the system, the user must punch a card into a card slot or enter a key produced by a key generator into an option provided by the operating system.

One Time passwords

Along with standard authentication, one-time passwords give an extra layer of security. Every time a user attempts to log into the One-Time Password system, a unique password is needed. Once a one-time password has been used, it cannot be reused. One-time passwords may be implemented in several ways.

1. Secret Key

The user is given a hardware device that can generate a secret id that is linked to the user's id. The system prompts for such a secret id, which must be generated each time you log in.

2. Random numbers

Users are given cards that have alphabets and numbers printed on them. The system requests numbers that correspond to a few alphabets chosen at random.

3. Network password

Some commercial applications issue one-time passwords to registered mobile/email addresses, which must be input before logging in. Firewalls

Firewalls are essential for monitoring all incoming and outgoing traffic. It imposes local security, defining the traffic that may travel through it. Firewalls are an efficient way of protecting network systems or local systems from any network-based security threat.

Physical Security

The most important method of maintaining operating system security is physical security. An attacker with physical access to a system may edit, remove, or steal important files since operating system code and configuration files are stored on the hard drive.

Operating System Security Policies and Procedures

Various operating system security policies may be implemented based on the organization that you are working in. In general, an OS security policy is a document that specifies the procedures for ensuring that the operating system maintains a specific level of integrity, confidentiality, and availability.

OS Security protects systems and data from worms, malware, threats, ransomware, backdoor intrusions, viruses, etc. Security policies handle all

preventative activities and procedures to ensure an operating system's protection, including steal, edited, and deleted data.

As OS security policies and procedures cover a large area, there are various techniques to addressing them. Some of them are as follows:

Installing and updating anti-virus software

Ensure the systems are patched or updated regularly

Implementing user management policies to protect user accounts and privileges.

Installing a firewall and ensuring that it is properly set to monitor all incoming and outgoing traffic.

OS security policies and procedures are developed and implemented to ensure that you must first determine which assets, systems, hardware, and data are the most vital to your organization. Once that is completed, a policy can be developed to secure and safeguard them properly

CHAPTER 9:CONCLUSION

9.1 Conclusion

The Internet has become a major resource in modern business. Artificial intelligence is one of the most rapidly growing domains today. It doesn't refer to just one thing; it's an umbrella term that can be applied to many different concepts and techniques. The best proof of machine learning progress is machine learning applications such as Facebook ads (Facebook can suggest ads you are interested in from studying your behavior on Facebook), face recognition (on Facebook and security of opening mobiles) and diagnosing systems by machine learning (Drug Discovery and Manufacturing).

Deep learning has been the most popular subset of machine learning in recent times. Deep learning can probably perform numerous tasks. Deep learning is the term given to machine learning architectures that join many multi-layer perceptrons together, so that there isn't just one hidden layer but many hidden layers. The "deeper" that the deep neural network is, the more sophisticated patterns the network can learn. The deep layer networks of neurons are sometimes referred to as fully connected networks or fully connected layers, referencing the fact that a given neuron maintains a connection to all the neurons surrounding it. Fully connected networks can be combined with other machine learning functions to create different deep learning architectures. With the spread of brain tumors, it's expected to develop Android and web apps for diagnosing cancer and find solutions or treatments.

9.2 Future work

This website depend on Machine learning , stream-lit and UI which is most advanced techniques so this website is on a fixed land but We have some ideas for better use and more audience

- 1- This app depends on some features that must be obtained from data-set, we can predict through Retinal Images Using Deep Neural Network
- 2- The accuracy may be better by making better for the true label by using a bigger data-set.

Chapter 10: References

- 1- <https://www.nature.com/articles/s41467-021-23458-5>
- 2- <https://www.techscience.com/iasc/v31n1/44320/html>
- 3- <https://www.hindawi.com/journals/cin/2022/7887908/>
- 4- [sklearn.ensemble.GradientBoostingClassifier — scikit-learn 1.2.2 documentation](#)
- 5- [How to build apps with Streamlit Python \(quick Tutorial\) - Just into Data](#)
- 6- [What is SQLite? And When to Use It? \(simplilearn.com\)](#)