**22AIE 205 INTRODUCTION TO PYTHON**

A Project Report

Submitted by

***GROUP 4***

**Arya Palackal Shijish - CB.EN.U4AIE22008**

**Riya Rajeev - CB.EN.U4AIE22042**

**Vamshidhar Reddy - CB.EN.U4AIE22028**

**Tarun Kumar - CB.EN.U4AIE22068**

A logo with red letters

Description automatically generated

**Centre for Computational Engineering and Networking**

**AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE**

**AMRITA VISHWA VIDYAPEETHAM**

COIMBATORE - 641 112 (INDIA)

**DECEMBER - 2023**

**AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE**

**AMRITA VISHWA VIDYAPEETHAM**

**COIMBATORE - 641 112**

**A logo with red letters

Description automatically generated**

**BONAFIDE CERTIFICATE**

This is to certify that the thesis entitled “Multiple disease Prediction System” in Python Programming Language submitted by Arya Palackal Shijish (CB.EN.U4AIE22008), Riya Rajeev (CB.EN.U4AIE22042), Vamshidhar Reddy (CB.EN.U4AIE22028), Tarun Kumar (CB.EN.U4AIE22068) is a bonafide record of the work carried out by his/her under our guidance and supervision at Amrita School of Artificial Intelligence, Coimbatore.

**Ms. Sreelakshmi K**

Project Guide

**Dr. K.P.Soman**

Professor and Head CEN

**AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE**

**AMRITA VISHWA VIDYAPEETHAM**

**COIMBATORE - 641 112**

**DECLARATION**

Arya Palackal Shijish (CB.EN.U4AIE22008), Riya Rajeev (CB.EN.U4AIE22042), Vamshidhar Reddy (CB.EN.U4AIE22028), Tarun Kumar (CB.EN.U4AIE22068) hereby declare that this thesis entitled “Multiple disease Prediction System in Python Programming language”, is the record of the original work done by me under the guidance of Mrs. Sreelakshmi K, Assistant Professor, Centre for Computational Engineering and Networking, Amrita School of Artificial Intelligence, Coimbatore. To the best of my knowledge this work has not formed the basis for the award of any degree/diploma/ associate ship/fellowship/or a similar award to any candidate in any University.

**Place: Coimbatore**

**Date:10-12-2023**

**Acknowledgment**

We are deeply thankful to Center for Excellence in Computational Engineering and Networking (CEN) at Amrita Vishwa Vidyapeetham, Coimbatore for providing us such a wonderful environment to peruse our research. We would like to express our sincere gratitude to Mrs. Sreelakshmi K, Asst. Professor, Department of Center for Excellence in CEN, Amrita Vishwa Vidyapeetham. We found the research area, topic, and problem with her suggestions. She guided us with our study, and supplied us many research papers and academic resources in this area. She is patient and responsible. When we had questions and needed her help, she would always find time to meet and discuss with us no matter how busy she was.

**Contents**

[**CHAPTER 1: HEART DISEASE PREDICTION** 7](#_Toc153139877)

[1.1 ABSTRACT 7](#_Toc153139878)

[1.2 INTRODUCTION 8](#_Toc153139879)

[1.3 METHODOLOGY 8](#_Toc153139880)

[1.3.1 Work Flow Explanation 9](#_Toc153139881)

[1.3.2 Libraries 11](#_Toc153139882)

[1.3.3 Data Collection and Processing 12](#_Toc153139883)

[1.3.4 Splitting the Features and Target 14](#_Toc153139884)

[1.3.5 Splitting the data into training data and test data 15](#_Toc153139885)

[1.3.6 Model Training with Logistic Regression 16](#_Toc153139886)

[1.3.7 Model Evaluation with Accuracy Score 17](#_Toc153139887)

[1.3.7.1 Accuracy Score of training data 18](#_Toc153139888)

[1.3.7.2 Accuracy Score of testing data 19](#_Toc153139889)

[1.3.8 Saving model in sav 20](#_Toc153139890)

[1.3.9 Streamlit Explation 20](#_Toc153139891)

[1.4 EXPERIMENTS AND RESULTS 22](#_Toc153139892)

[1.5 CONCLUSION 23](#_Toc153139893)

[1.6 REFERENCES 23](#_Toc153139894)

[**CHAPTER 2: DIABETES PREDICTION SYSTEM** 24](#_Toc153139895)

[2.1 ABSTRACT: 24](#_Toc153139896)

[2.2 INTRODUCTION: 24](#_Toc153139897)

[2.3 METHODOLOGY: 25](#_Toc153139898)

[2.3.1 Importing the dependencies 26](#_Toc153139899)

[2.3.2 Data collection and analysis 26](#_Toc153139900)

[2.3.3 Separating data and labels 27](#_Toc153139901)

[2.3.4 Data Standardization 27](#_Toc153139902)

[2.3.5 Splitting data into training and test data 29](#_Toc153139903)

[2.3.6 Training the model 29](#_Toc153139904)

[2.3.7 Model Evaluation: 30](#_Toc153139905)

[2.3.8 Making of Predictive system 31](#_Toc153139906)

[2.3.9 Implementation of Diabetes disease prediction in streamlit 32](#_Toc153139907)

[2.4 EXPERIMENTS AND RESULTS: 34](#_Toc153139908)

[2.5 CONCLUSION: 36](#_Toc153139909)

[2.6 REFERENCES 37](#_Toc153139910)

[**CHAPTER 3: PARKINSON’S DISEASE PREDICTION** 38](#_Toc153139911)

[3.1 ABSTRACT 38](#_Toc153139912)

[3.2 INTRODUCTION 38](#_Toc153139913)

[3.3 METHODOLOGY 39](#_Toc153139914)

[3.3.1 The Functions used 39](#_Toc153139915)

[3.3.2 Importing the libraries 39](#_Toc153139916)

[3.3.3 The Datasets 40](#_Toc153139917)

[3.3.4 Training and Testing sets 41](#_Toc153139918)

[3.3.5 X and Y features 42](#_Toc153139919)

[3.3.6 Streamlit 43](#_Toc153139920)

[3.4 EXPERIMENTS AND RESULT 45](#_Toc153139921)

[3.5 CONCLUSION 46](#_Toc153139922)

[3.6 REFERENCES 46](#_Toc153139923)

[**CHAPTER 4: LUNG CANCER** 47](#_Toc153139924)

[4.1 ABSTRACT 47](#_Toc153139925)

[4.2 INTRODUCTION 47](#_Toc153139926)

[4.3 METHODOLOGY 48](#_Toc153139927)

[4.3.1 Work Flow Explanation 48](#_Toc153139928)

[4.3.2 Libraries 49](#_Toc153139929)

[4.3.3 Data Collection and Processing: 51](#_Toc153139930)

[4.3.4 General code explanation: 54](#_Toc153139931)

[4.3.5 Splittting the data into graph: 56](#_Toc153139932)

[4.3.6 Model Training with Logistic Regression: 57](#_Toc153139933)

[4.3.7 Model Evaluation with Accuracy Score: 57](#_Toc153139934)

[4.3.7.1 Calculating the Mean : 59](#_Toc153139935)

[4.3.8 Streamlit Explation: 60](#_Toc153139936)

[4.4 EXPERIMENTS AND RESULTS 60](#_Toc153139937)

[4.5 CONCLUSION 61](#_Toc153139938)

**List of Figures**

[Figure 1.3.1: Work Flow. 8](#_Toc153139740)

[Figure 1.3.1.1: Heart ECG signal. 10](#_Toc153139741)

[Figure 1.3.2.1: Imported Libraries. 11](#_Toc153139742)

[Figure 1.3.3.1: CSV file information. 12](#_Toc153139743)

[Figure 1.3.3.2: Statistical measures. 13](#_Toc153139744)

[Figure 1.3.5.1: Splitting of data. 15](#_Toc153139745)

[Figure 1.3.7.1.1: Training data Accuracy. 17](#_Toc153139746)

[Figure 1.3.7.2.1: Testing data Accuracy. 18](#_Toc153139747)

[Figure 1.3.8.1: saved trained model. 19](#_Toc153139748)

[Figure 1.3.9.2: User inputs Code. 20](#_Toc153139749)

[Figure 2.3.1: Workflow 24](#_Toc153139750)

[Figure 2.3.1.1: Importing libraries 25](#_Toc153139751)

[Figure 2.3.2.1: Data Collection 25](#_Toc153139752)

[Figure 2.3.5.1: Splitting of data 28](#_Toc153139753)

[Figure 2.4.2: Result 1 33](#_Toc153139754)

[Figure 2.5.1: Prediction flowchart 36](#_Toc153139755)

[Figure 3.3.2.1: Libraries 39](#_Toc153139756)

[Figure 3.3.3.1: Datasets 39](#_Toc153139757)

[Figure 3.3.4.1: Training and Testing sets 40](#_Toc153139758)

[Figure 3.3.5.1: X and Y features 41](#_Toc153139759)

[Figure 3.4.1: Experiment 43](#_Toc153139760)

[Figure 3.4.2: Result 43](#_Toc153139761)

[Figure 4.3.3.1: CSV file information. 50](#_Toc153139762)

[Figure 4.3.3.2: CSV file information. 51](#_Toc153139763)

[Figure 4.3.3.3: Statistical measures. 52](#_Toc153139764)

[Figure 4.3.4.2: lung\_cancer coloumn plotting. 52](#_Toc153139765)

[Figure 4.3.4.4: seaborn plotting. 53](#_Toc153139766)

[Figure 4.3.5.1: Splitting of data(x). 54](#_Toc153139767)

[figure 4.3.5.1: Splitting of data 55](#_Toc153139768)

[Figure 4.3.7.1: Training data Accuracy 56](#_Toc153139769)

[Figure 4.4.1: Experiment output 59](#_Toc153139770)

[Figure 4.4.2: Streamlit output. 59](#_Toc153139771)

# CHAPTER 1: HEART DISEASE PREDICTION

## 1.1 ABSTRACT

This project focuses on developing a heart disease prediction system using the powerful logistic regression algorithm. The objective is to construct a machine learning model that accurately predicts the likelihood of an individual having heart disease based on a range of health parameters, such as age, cholesterol level, and blood pressure. Leveraging Logistic Regression, a well-suited supervised learning algorithm for binary classification, the model is trained on a dataset containing patients' medical details and their heart disease status. Rigorous data preprocessing is applied, including analysis and normalization, to ensure the data's compatibility with the logistic regression model. The dataset is then divided into training and testing sets to evaluate the model's accuracy.

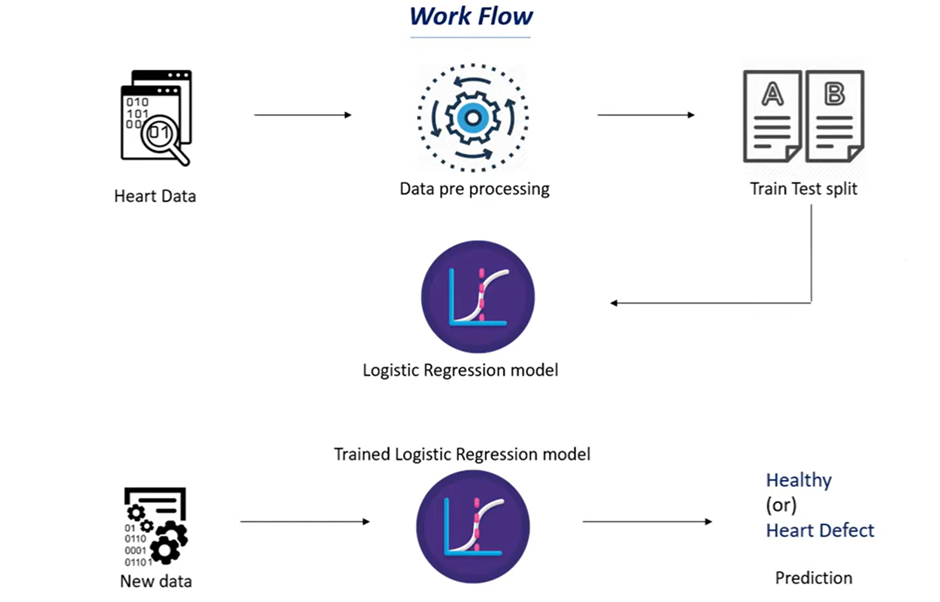
The logistic regression classifier is employed to categorize individuals into those likely to have heart disease and those not. This predictive capability establishes a robust mechanism for early detection and intervention. The project's significance lies in its potential to support healthcare professionals by providing a tool for the timely identification of individuals at risk of heart disease. The outlined workflow serves as a comprehensive guide for developing the Heart Disease Prediction System, contributing to advancements in healthcare through the application of machine learning.

## 1.2 INTRODUCTION

Heart disease is a major global health concern, contributing significantly to morbidity and mortality. Early detection and prediction of heart disease can be crucial in preventing its progression and improving patient outcomes. In this context, the Heart Disease Prediction System project aims to develop a robust and accurate system for predicting the likelihood of heart disease in individuals.

## 1.3 METHODOLOGY

First, we look into the workflow of our Heart disease Prediction.



#### Figure 1.3.1: Work Flow.

## 1.3.1 Work Flow Explanation

**Heart data:**

This data set contains several health parameters that correspond to a person's healthiness of the heart.

**Data preprocessing:**

We need to process this data set; we cannot feed this raw data into our machine-learning algorithm.

We need to process this data set to make it fit and be compatible with our machine-learning algorithm.

**Test and train:**

Once we process the data, we need to split our data into training data and testing data.

This is because we often train our machine learning algorithm with training data, and we will evaluate the performance of our model using the test data. This part is called a **strain test split.**

We will split our original data set into training data and test data.

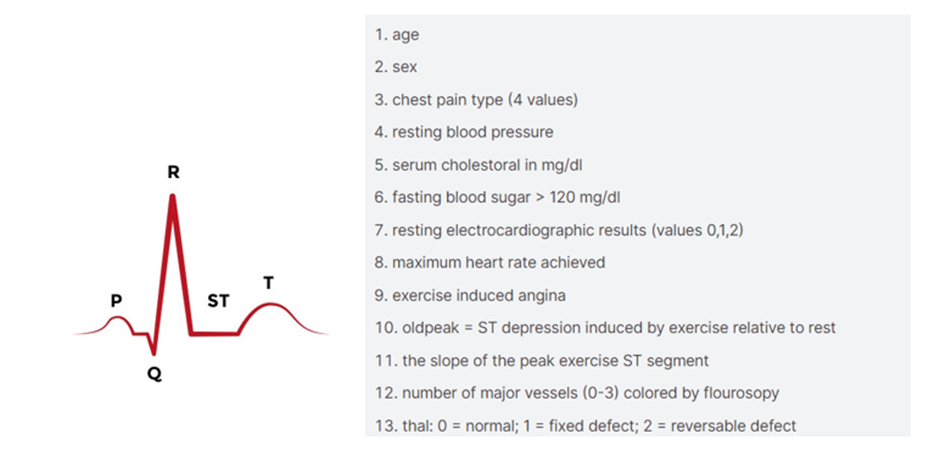
**Logistic regression model:**

We will feed our training data to our machine-learning model. In this case, we are going to use a logistic regression model because this particular use case is a binary classification. So, we are going to classify whether a person has a deceased heart or not.

In those binary classifications, the logistic regression model is very useful. So, it's the best model when it comes to binary classification. Once we train this logistic regression model with our training data, we get trained in logistic regression mode.

**Trained logistic regression model:**

When we feed new data, our model can predict whether the person has heart disease or not.



#### Figure 1.3.1.1: Heart ECG signal.

* If a person has a healthy heart, the ECG signal of the heart is shown in the first image.
* From that ECG signal, we add features or variables to our project. That is shown in the second image.

Now we will develop a heart disease prediction model.

### 1.3.2 Libraries

A screen shot of a computer code

Description automatically generated

#### Figure 1.3.2.1: Imported Libraries.

**Numpy :**

The libraries include NumPy for arrays.

**Pandas:**

Pandas for creating structured tables (data frames)

**scikit-learn:**

* scikit-learn for model selection and evaluation. The logistic regression model is chosen for this project, and its implementation is facilitated by importing '**LogisticRegression**' from the scikit-learn linear model module. Additionally, we import the '**train\_test\_split**' function for data splitting and the '**accuracy\_score**' for evaluating the model's performance.
* From this model, we are importing train\_test\_split and logistic regression.
* The accuracy score is used to evaluate our model and check how well it is performing.

### 1.3.3 Data Collection and Processing

After importing the necessary libraries, the next step is to load the dataset into a Pandas data frame.

**heart\_data = pd.read\_csv('heart.CSV)**

* For that, we are using **Pd.read\_csv for loading CSV data into a Pandas Dataframe.**

**heart\_data.head()**

**heart\_data.tail()**

**heart\_data.shape**

* To load sample data from a CSV file, use the keywords **head() and tail().** This can get sample data from a CSV file.
* To get the number of rows and columns in the CSV file, we use the keyword “**shape”.**
* To get information from data, we use the keyword **info().**
* We get the info as below image.

**heart\_data.info()**

**A screenshot of a computer

Description automatically generated**

#### Figure 1.3.3.1: CSV file information.

* Here, we observe 1025 entries, 14 columns, and 1025 non-null values.
* Null values mean missing values; non-null values mean normal values. Dtype represents the data type.
* To check missing values, the keyword we used is heart\_data. **isnull(). sum().**
* It gives us the number of missing values in each column.

To load statistical measures of the table, we use the keyword **describe()**

**heart\_data.describe()**

The 'describe' function provides statistical measures for each column, including count (number of data points), mean (average), standard deviation (a measure of data dispersion), minimum values for each column, 25th percentile, 50th percentile (median), 75th percentile, and maximum values for each column.

A screenshot of a computer

Description automatically generated

#### Figure 1.3.3.2: Statistical measures.

Checking the distribution of target variables to see how many people with heart disease.

**heart\_data['target'].value\_counts()**

From the above keyword, we get a count of heart disease people in the given data set.

### 1.3.4 Splitting the Features and Target

From the above steps, we process the data. Now we have to split the data into training data and testing data.

We are splitting the data into **features** and **targets**.

**X = heart\_data.drop(columns='target', axis=1)**

**Y = heart\_data ['target’]**

Here we dropped the target column from our data set using the keyword **drop(),** and axis = 1 **represents the** column.

### 1.3.5 Splitting the data into training data and test data

**X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, stratify=Y, random\_state=2)**

**train\_test\_split:**

* This function is part of scikit-learn's **model\_selection** module and is used to split the dataset into training and testing sets.

**X and Y:**

* **X** typically represents the feature matrix (independent variables), and **Y** represents the target variable (dependent variable) in a machine learning model.

**test\_size=0.2:**

* This parameter specifies the proportion of the dataset to include in the test split. In this case, it's set to 0.2, indicating that 20% of the data will be used for testing and the remaining 80% will be used for training.

**stratify=Y:**

* The **stratify** parameter ensures that the splitting process maintains the same distribution of the target variable **Y** in both the training and testing sets. This is particularly useful when dealing with imbalanced datasets. If you don’t mention stratify = y, there is a possibility that all the values in the x test may contain 0 or all the values may contain 1.

**random\_state=2:**

* **random\_state** is used to set a seed for reproducibility. By providing a fixed seed (in this case, 2), the randomness in the splitting process is controlled, resulting in the same train-test split whenever the code is executed. This is useful for reproducibility in machine-learning experiments.

**X\_train, X\_test, Y\_train, Y\_test:**

* These variables store the resulting training and testing sets for features (**X\_train** and **X\_test**) and target variables (**Y\_train** and **Y\_test**).

A screenshot of a computer program

Description automatically generated

#### Figure 1.3.5.1: Splitting of data.

* It split X into 80% training data and 20% testing data.

### 1.3.6 Model Training with Logistic Regression

In the process of model training for logistic regression, the algorithm learns from a set of labeled data (features and corresponding outcomes). Using techniques like gradient descent, the model adjusts its parameters to minimize the difference between its predictions and the actual outcomes, effectively learning the patterns associated with heart disease.

**model = LogisticRegression()**

The line **model = LogisticRegression()** initializes a Logistic Regression model, creating a blank slate for subsequent training and prediction tasks in your code.

**model.fit(X\_train, Y\_train)**

**model. fit(X\_train, Y\_train)** is like the phase where our Logistic Regression model learns from examples. The **X\_train** contains the features or details about heart health (like age, cholesterol, etc.), and the **Y\_train** has the corresponding outcomes (whether a person has heart disease or not). The **fit** process is where the model analyzes these examples, figuring out the relationships between features and outcomes so that it can later predict whether new individuals are likely to have heart disease based on similar characteristics.

### 1.3.7 Model Evaluation with Accuracy Score

Once the logistic regression model is trained, it is essential to evaluate its performance. The accuracy score is a metric that measures the proportion of correctly predicted instances. It is calculated by dividing the number of correct predictions by the total number of predictions. While accuracy provides an overall performance assessment, it's important to consider other metrics, especially in imbalanced datasets, to gain a comprehensive understanding of the model's effectiveness.

### 1.3.7.1 Accuracy Score of training data

A screenshot of a computer code

Description automatically generated

#### Figure 1.3.7.1.1: Training data Accuracy.

* **model.predict(X\_train)**: The model makes predictions on the training data (**X\_train**), trying to guess whether each person has heart disease or not based on the features in the training set.
* **accuracy\_score(X\_train\_prediction, Y\_train)**: The accuracy score is then calculated by comparing these predictions (**X\_train\_prediction**) to the actual outcomes in the training data (**Y\_train**). It measures the proportion of correct predictions out of the total predictions, giving you an idea of how well your model is performing on the data it was trained on.

If we print, the training data accuracy score is found to be 85%.

### 1.3.7.2 Accuracy Score of testing data

A screenshot of a computer code

Description automatically generated

#### Figure 1.3.7.2.1: Testing data Accuracy.

* **model.predict(X\_test)**: The model makes predictions on the test data (**X\_test**), attempting to forecast whether each individual in the test set has heart disease or not based on the learned patterns.
* **accuracy\_score(X\_test\_prediction, Y\_test)**: The accuracy score is then computed by comparing these predictions (**X\_test\_prediction**) to the actual outcomes in the test data (**Y\_test**). This score indicates how well your model generalizes to new, unseen data, helping you assess its overall performance.

If we print, the training data accuracy score is found to be 80%.

### 1.3.8 Saving model in sav

A screenshot of a computer program

Description automatically generated

#### Figure 1.3.8.1: saved trained model.

### 1.3.9 Streamlit Explation

I implemented the Heart Disease Prediction System using the Logistic Regression model in a Streamlit web application. This application offers a user-friendly interface, allowing individuals to predict the likelihood of heart disease based on their health parameters. The sidebar navigation enhances the user experience, enabling easy switching between different disease prediction models.

In the Heart Disease Prediction section, users provide crucial health information such as age, sex, chest pain type, and other relevant features. The code efficiently manages the conversion of categorical features like sex and chest pain type into numeric values, ensuring compatibility with the logistic regression model.

When the user clicks the "Heart Disease Test Result" button, the logistic regression model is activated to generate a prediction. The result is presented in a clear and straightforward format, informing the user about the likelihood of the individual having heart disease. This approach aims to provide accessible and valuable health insights to users through a user-centric web application.

A screenshot of a computer program

Description automatically generated

*Figure 1.3.9.1: load heart modal and sidebar navigator codes*

*A screenshot of a computer program

Description automatically generated**.*

#### Figure 1.3.9.2: User inputs Code.

## 1.4 EXPERIMENTS AND RESULTS

A screenshot of a computer code

Description automatically generated

*Figure 1.4.1: Experiment output.*

**

*Figure 1.4.2: Streamlit output.*

## 1.5 CONCLUSION

In conclusion, the Heart Disease Prediction System employing Logistic Regression stands as a promising tool for early detection and intervention in cardiovascular health. The meticulously crafted model, trained on a comprehensive dataset and utilizing robust preprocessing techniques, demonstrates the potential of machine learning in assisting healthcare professionals. By accurately predicting the likelihood of heart disease based on individual health parameters, the system holds the key to proactive and personalized care. Moving forward, this project not only contributes to advancements in predictive healthcare but also underscores the significance of integrating machine learning into medical practices for improved patient outcomes and early disease management.

## 1.6 REFERENCES

[1] [https://www.youtube.com/watch?v=qmqCYC-MBQo&list=PPSV](https://www.youtube.com/watch?v=qmqCYC-MBQo&list=PPSV%20)

[2]<https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset>

[3] [https://github.com/g-shreekant/Heart-Disease-Prediction-using-Machine-Learnin](https://github.com/g-shreekant/Heart-Disease-Prediction-using-Machine-Learning)

# CHAPTER 2: DIABETES PREDICTION SYSTEM

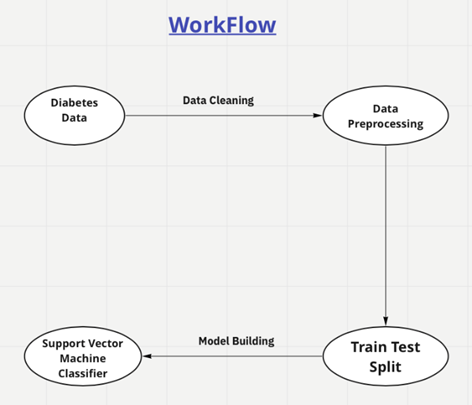
## 2.1 ABSTRACT:

This project focuses on the development of a Diabetes Prediction System using the Support Vector Machine (SVM) algorithm. The goal is to create a machine learning model capable of accurately predicting whether an individual has diabetes based on various medical parameters, such as blood glucose level, insulin level, and BMI. The Support Vector Machine, a powerful supervised learning algorithm, is employed for its effectiveness in classifying data into distinct categories. The model is trained on a dataset comprising medical information of patients labeled with their diabetes status. The workflow involves meticulous data preprocessing steps, including analysis and standardization, to ensure the suitability of the data for the SVM model. The dataset is then split into training and testing sets to assess the model's accuracy. The SVM classifier is employed to classify whether a person is diabetic or non-diabetic, providing a reliable prediction mechanism. The significance of this project lies in its potential to assist healthcare professionals in early diabetes detection, enabling timely intervention and personalized care. The abstracted workflow serves as a comprehensive guide for developing and understanding the Diabetes Prediction System, contributing to the field of healthcare and machine learning applications.

## 2.2 INTRODUCTION:

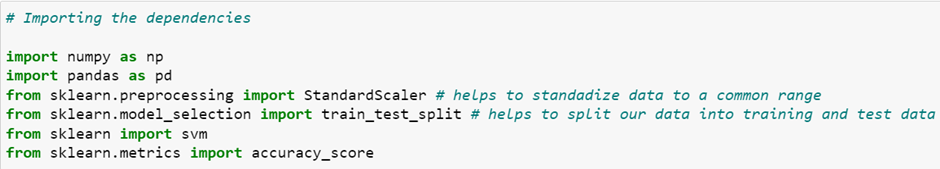
In this report, we present the development of a Diabetes Prediction System using the Support Vector Machine (SVM) algorithm. The project revolves around the imperative need for accurate and early prediction of diabetes, a widespread health concern. The SVM algorithm, known for its efficacy in classification tasks, is employed to analyse a dataset containing vital medical parameters. Our approach involves meticulous data preprocessing, including analysis and standardization, followed by the training of the SVM model and its evaluation for accuracy. The project underscores the intersection of advanced machine learning techniques and healthcare, contributing to the ongoing efforts to enhance diagnostic capabilities for better patient outcomes.

## 2.3 METHODOLOGY:



#### Figure 2.3.1: Workflow

### 2.3.1 Importing the dependencies



#### Figure 2.3.1.1: Importing libraries

### 2.3.2 Data collection and analysis

This dataset contains information about patients, specifically focusing on females, distinguishing between those with diabetes and those without the condition. It includes various parameters such as the number of pregnancies, blood glucose levels, insulin levels, among others.



#### Figure 2.3.2.1: Data Collection

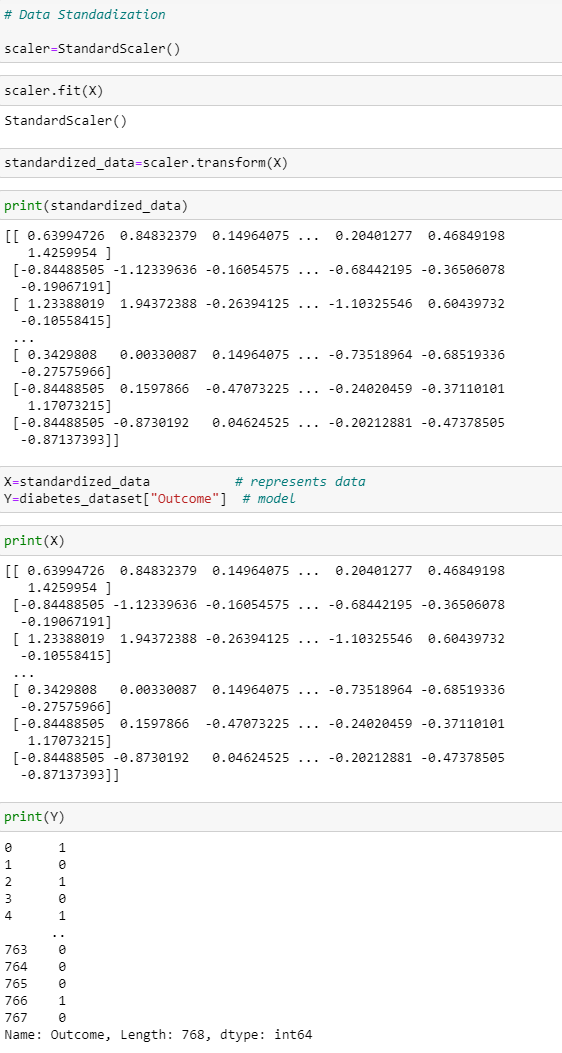
Loading the diabetes dataset to a pandas DataFrame

### 2.3.3 Separating data and labels

To segregate the data and labels, I'll create two variables, x and y. In the process of separating the data and labels, I'll use the drop function to exclude the 'outcome' column from the dataset and store the remaining data in the variable x. This is achieved by utilizing the drop method on the 'diabetes\_dataset' dataframe, specifying the column 'outcome' and setting the axis to one, indicating column-wise operation. Simultaneously, I'll capture all the labels, contained in the 'outcome' column, and store them in the variable y. This ensures that 'x' holds all the data except for the labels, while 'y' specifically contains the label information.

### 2.3.4 Data Standardization

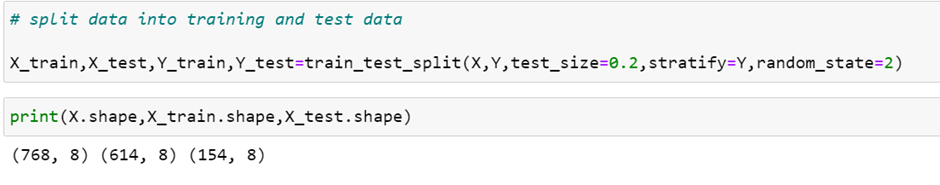
In our dataset, we have various parameters such as the number of pregnancies, glucose value, blood pressure value, BMI, and more. Each of these parameters operates within a different numerical range. For instance, the range of pregnancies might be one, two, or three, while glucose levels could vary between 100 and 150, and blood pressure levels could range from 60 to 70. To ensure that the machine learning model can effectively make predictions, it's imperative to standardize the data, bringing all these parameters into a consistent range. Standardization is achieved using the StandardScaler function, which we previously imported. By creating a variable called 'scalar' and fitting our data using 'standard\_scalar.fit(x),' we initiate the process of transforming the data into a standardized format. This standardized data will facilitate more accurate predictions from our machine learning model.



*Figure 2.3.4.1: Data Standardization*

### 2.3.5 Splitting data into training and test data

Next, we need to split our dataset into training and testing data using the train\_test\_split function from the sklearn.model\_selection module. This function requires four variables: x\_train, x\_test, y\_train, and y\_test. The test\_size=0.2 parameter designates 20% of the data for testing, and stratify=y ensures a proportional distribution of labels in both sets. Additionally, random\_state=2 provides reproducibility. The resulting sets allow us to train our machine learning model on x\_train and y\_train, and subsequently evaluate its performance on x\_test and y\_test.



#### Figure 2.3.5.1: Splitting of data

### 2.3.6 Training the model

I've created a variable named 'classifier,' and we'll utilize the SVM (Support Vector Machine) classifier for training our model. Specifically, we use the SVC function from the SVM module, designating the 'linear' kernel to implement a linear model. This function loads our support vector machine classifier, and the resulting model is stored in the 'classifier' variable. Now, with our model in place, we proceed to train it using the training data. The classifier.fit function is employed for this purpose, where we provide the training data to facilitate the training of our support vector machine classifier.

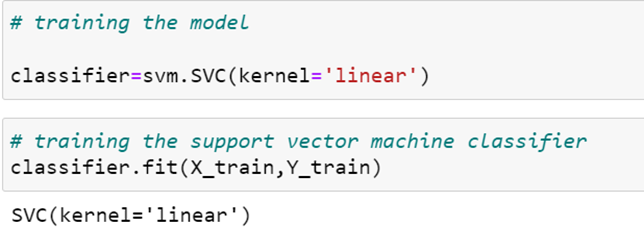
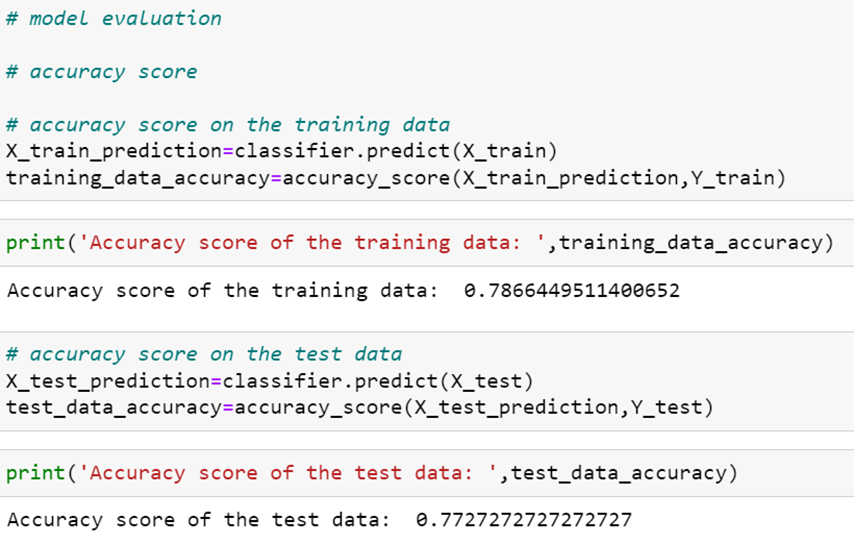


Figure 2.3.6.1: Training the model

### 2.3.7 Model Evaluation:

We begin by assessing the accuracy score, a fundamental metric in gauging the model's effectiveness. First, we predict the outcomes on the training data using our trained classifier, and subsequently, we calculate the accuracy score on the training data. This score is obtained by comparing the predicted outcomes (X\_train\_prediction) with the actual labels (Y\_train). The process is mirrored for the test data, where we predict outcomes using the classifier and compute the accuracy score by comparing the predictions (X\_test\_prediction) with the actual test labels (Y\_test).



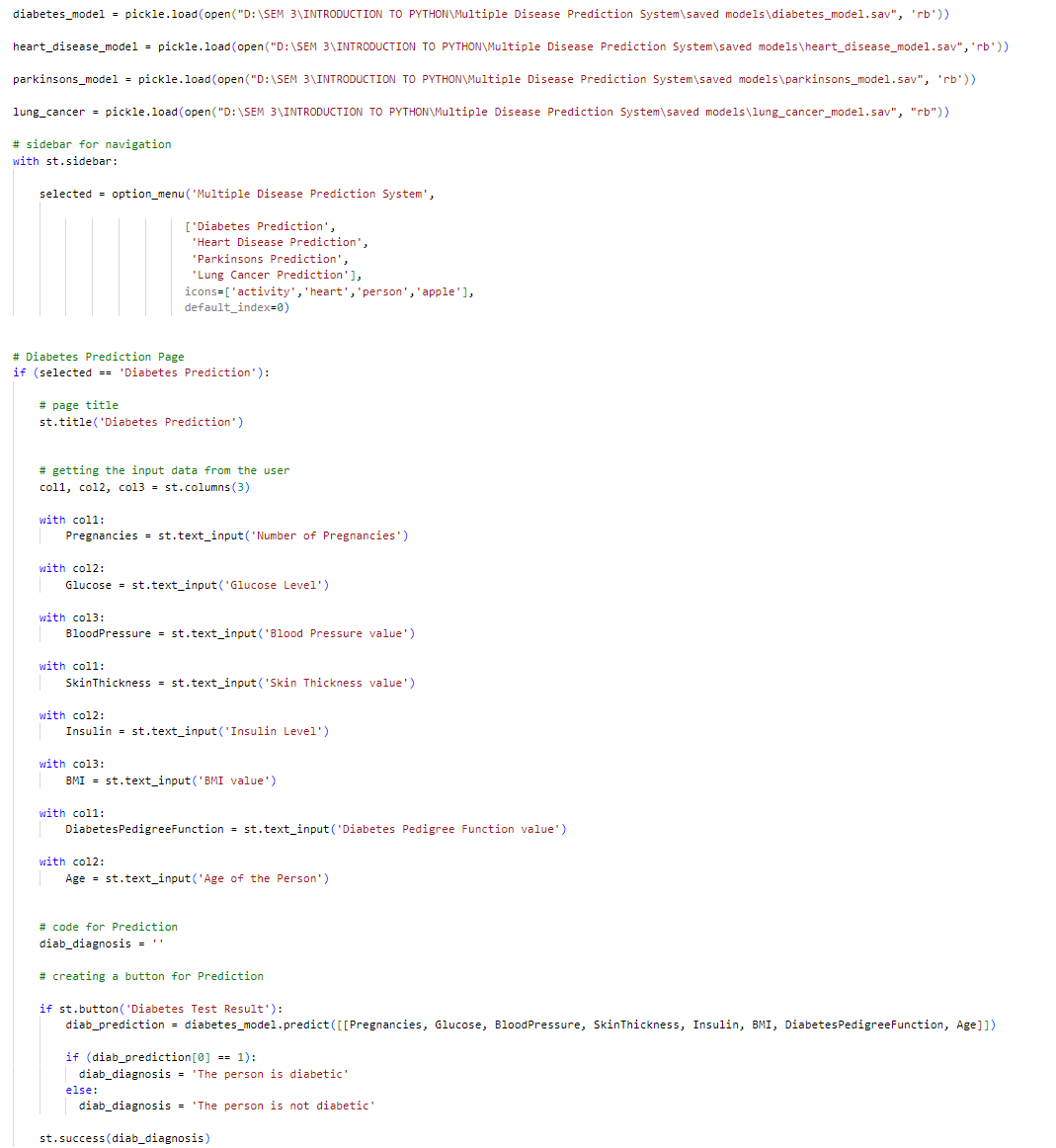
*Figure 2.3.7.1: Model Evaluation*

### 2.3.8 Making of Predictive system

Here, we have a sample input data represented by a tuple, denoting various medical parameters. To make this data suitable for prediction, we convert it into a NumPy array, reshape it as we are predicting for a single instance, and standardize the input data using the previously defined scaler. The standardized data is then fed into our trained classifier for prediction. The outcome of the prediction is printed, with a subsequent conditional statement determining whether the person is diabetic or not based on the prediction.

### 2.3.9 Implementation of Diabetes disease prediction in streamlit

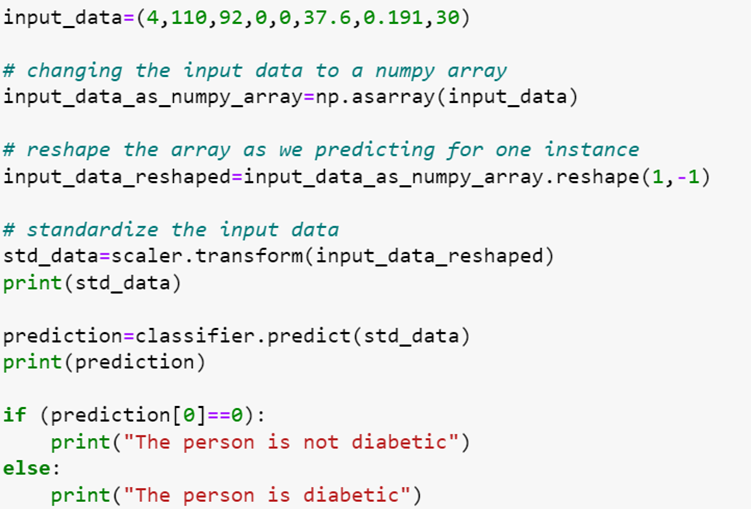
The Diabetes disease prediction module implemented in Streamlit provides an interactive and user-friendly platform for predicting the likelihood of diabetes based on input parameters. Leveraging machine learning models, specifically a diabetes prediction model trained on relevant health data, the system allows users to input information such as the number of pregnancies, glucose level, blood pressure, skin thickness, insulin level, BMI, diabetes pedigree function, and age. The Streamlit app's intuitive design features a sidebar navigation for easy access to different disease prediction modules. Upon submitting the input data, the model processes the information and produces a prediction result. The user is presented with a clear and concise message indicating whether the individual is likely to be diabetic or not, along with the corresponding confidence level.



*Figure 2.3.9.1: Implementation in streamlit*

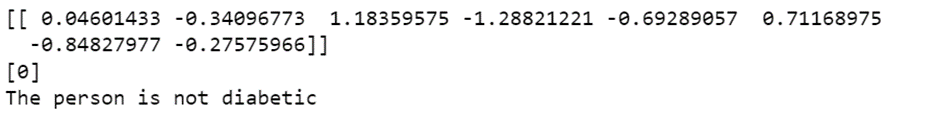
## 2.4 EXPERIMENTS AND RESULTS:

Experiment 1:



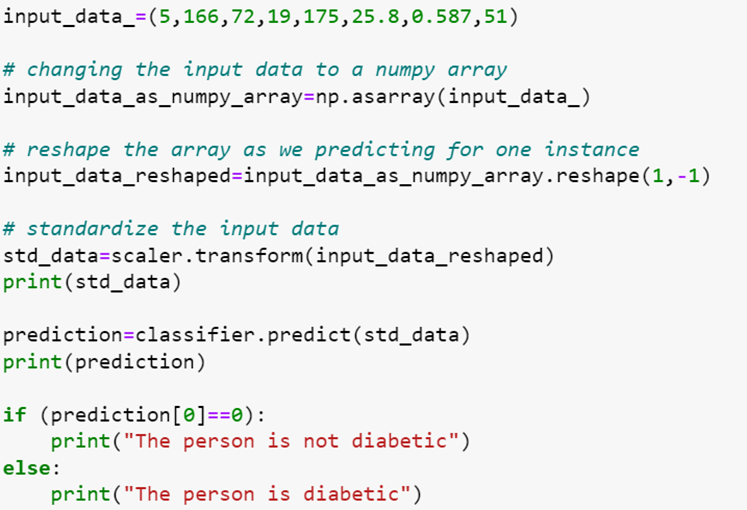
*Figure 2.4.1: Experiment 1*

Result:



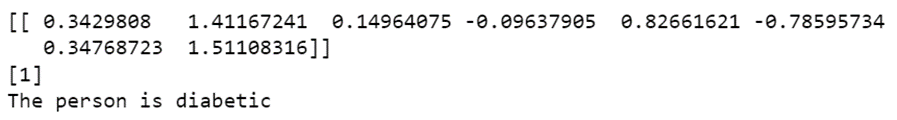
#### Figure 2.4.2: Result 1

Experiment 2:



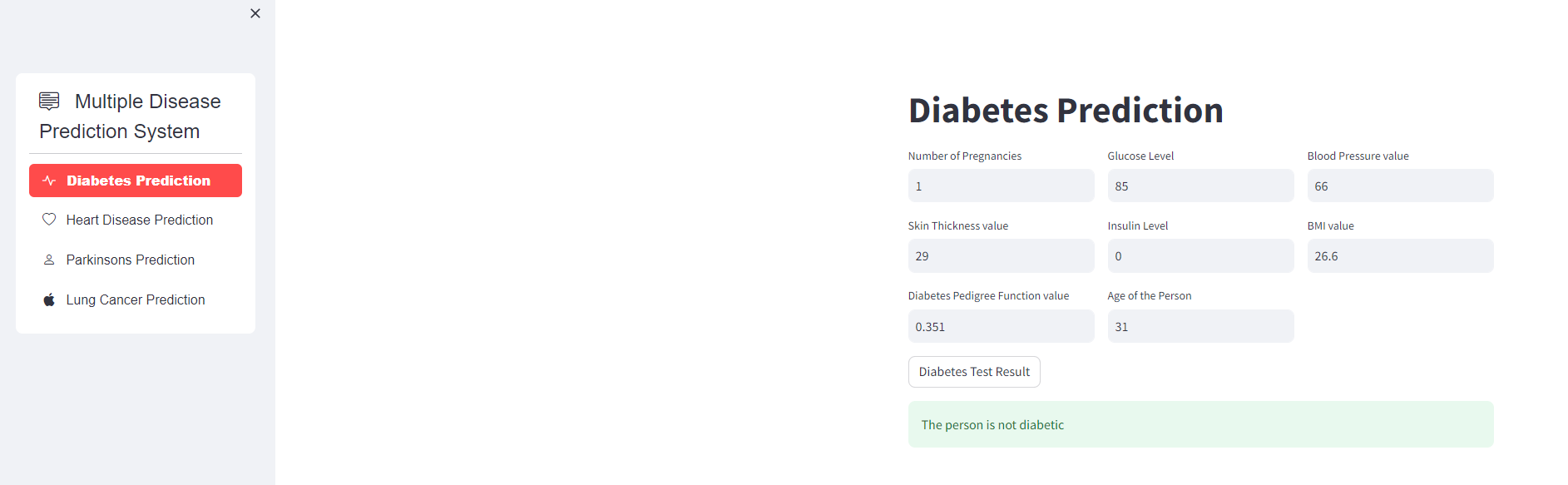
*Figure 2.4.3: Experiment 2*

Result:



*Figure 2.4.4: Result 2*

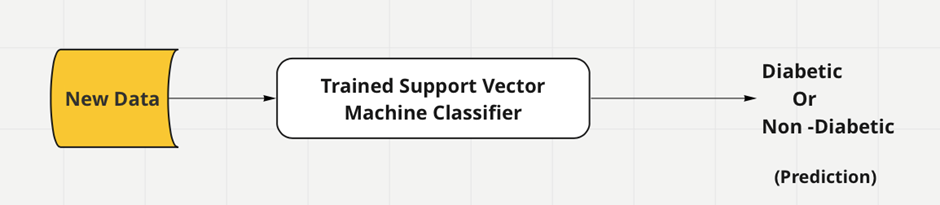
Experiment 3: Streamlit



*Figure 2.4.5: Streamlit Output*

## 2.5 CONCLUSION:

In conclusion, this project successfully implemented a Diabetes Prediction System using the Support Vector Machine (SVM) algorithm. The workflow encompassed crucial steps, including data preprocessing, standardization, and model evaluation, ensuring the effectiveness of the predictive model. The accuracy scores on both training and test datasets provided insights into the model's performance. The practical application of the predictive system demonstrated its utility in real-world scenarios, contributing to early diabetes detection and personalized healthcare. Ongoing refinement and exploration of additional features present avenues for future enhancements, highlighting the project's contribution to the intersection of machine learning and healthcare.



#### Figure 2.5.1: Prediction flowchart

## 2.6 REFERENCES

[1] <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10107388/>

[2] <https://www.kaggle.com/code/ahmetcankaraolan/diabetes-prediction-using-machine-learning>

[3] <https://ieeexplore.ieee.org/document/10128216>

[4] <https://www.javatpoint.com/diabetes-prediction-using-machine-learning>

# CHAPTER 3: PARKINSON’S DISEASE PREDICTION

## 3.1 ABSTRACT

A Support Vector Machine (SVM) model for classifying Parkinson's disease based on a dataset loaded from a CSV file is created. The dataset is preprocessed, exploring its structure, handling missing values and obtaining statistical measures. The SVM model is trained using the linear kernel and its accuracy is evaluated on both the training and test datasets. Subsequently, the model is used to predict Parkinson's disease status for a given set of input features. The serialization of the trained model using pickle is included for future use. The saved model is loaded and the features used in the training process are printed for reference.

## 3.2 INTRODUCTION

Parkinson's disease is a progressive disorder that affects the nervous system and the parts of the body controlled by the nerves. Parkinson's disease symptoms can be different for everyone and the symptoms include tremor, slowed movement, rigid muscles, impaired posture, speech changes, etc. In this project we have primarily focused on building, training, evaluating using machine learning model to predict whether a person has Parkinson’s disease based on a set of features. This is done using SVM. Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks. It is a powerful and versatile algorithm that is effective in high-dimensional spaces, making it suitable for a wide range of applications, including image classification, text categorization and bioinformatics.

## 3.3 METHODOLOGY

Prediction on whether a person has Parkinson’s disease is done with the help of Python programming language where end to end machine learning tasks, from data loading to exploration to model training, evaluation and prediction is done using an SVM(Support Vector Machine) model.

### 3.3.1 The Functions used

The major functions used are:

• pd.read\_csv(): Reads a CSV file and creates a Pandas DataFrame.

• shape: Returns the number of rows and columns in the DataFrame.

• describe(): Generates descriptive statistics of the DataFrame.

• value\_counts(): Counts the occurrences of unique values in a column.

• train\_test\_split(): Splits the dataset into training and testing sets.

• svm.SVC(): Creates a Support Vector Machine (SVM) classifier

• accuracy\_score(): Computes the accuracy of a classification model.

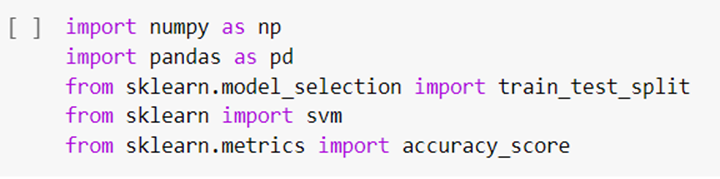
• predict(): Makes predictions using a trained model.

• reshape(): Reshapes an array.

• pickle.dump(): Saves an object to a file using pickle.

• pickle.load(): Loads an object from a file using pickle.

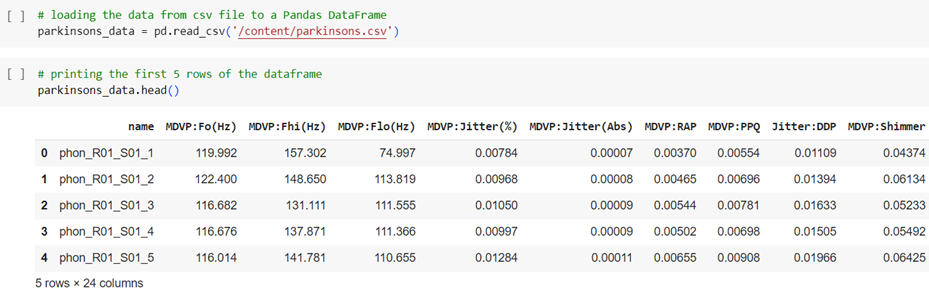
### 3.3.2 Importing the libraries



#### Figure 3.3.2.1: Libraries

The necessary libraries: NumPy for numerical operations, Pandas for data manipulation, train\_test\_split for splitting the dataset, svm for Support Vector Machines and accuracy\_score for evaluating model accuracy are imported.

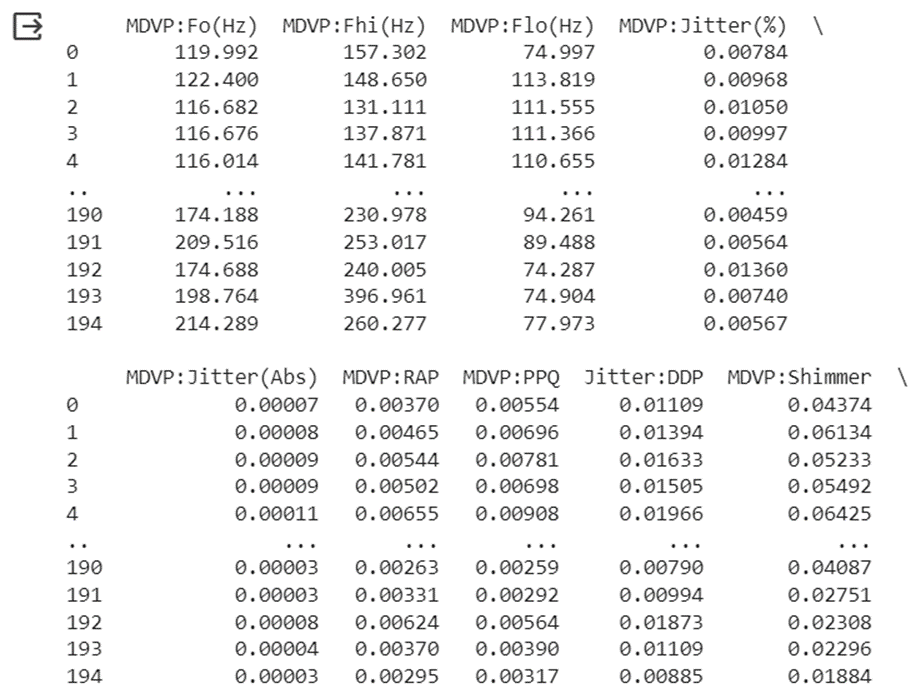
### 3.3.3 The Datasets

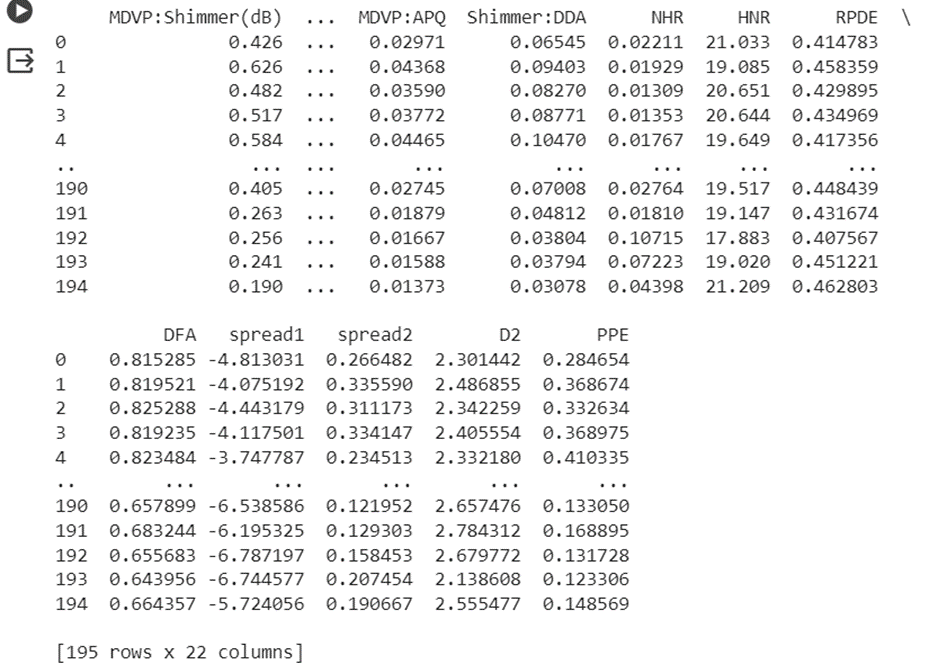


#### Figure 3.3.3.1: Datasets

The csv file containing Parkinson’s data is read and stored in a variable. Details about the number of rows, columns, data types, etc are printed. The dataset mainly contains the voice recordings of various people. These recordings are often obtained through speech samples or specific vocal tasks. The features include characteristics like pitch, jitter, shimmer, formants, energy and other relevant attributes.

### 3.3.4 Training and Testing sets

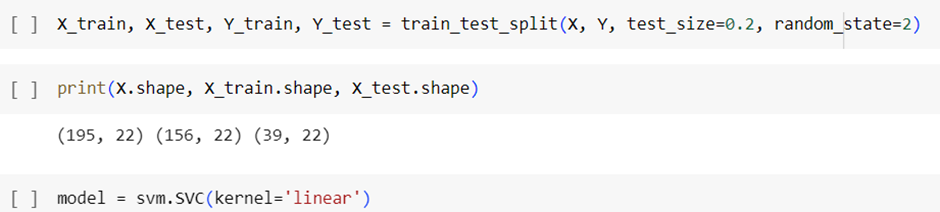




#### Figure 3.3.4.1: Training and Testing sets

The statistical measures for the numerical columns is presented. The occurrences of each unique value in the 'status' column, which seems to be the target variable is counted. The data is grouped and mean for each group is calculated.

### 3.3.5 X and Y features

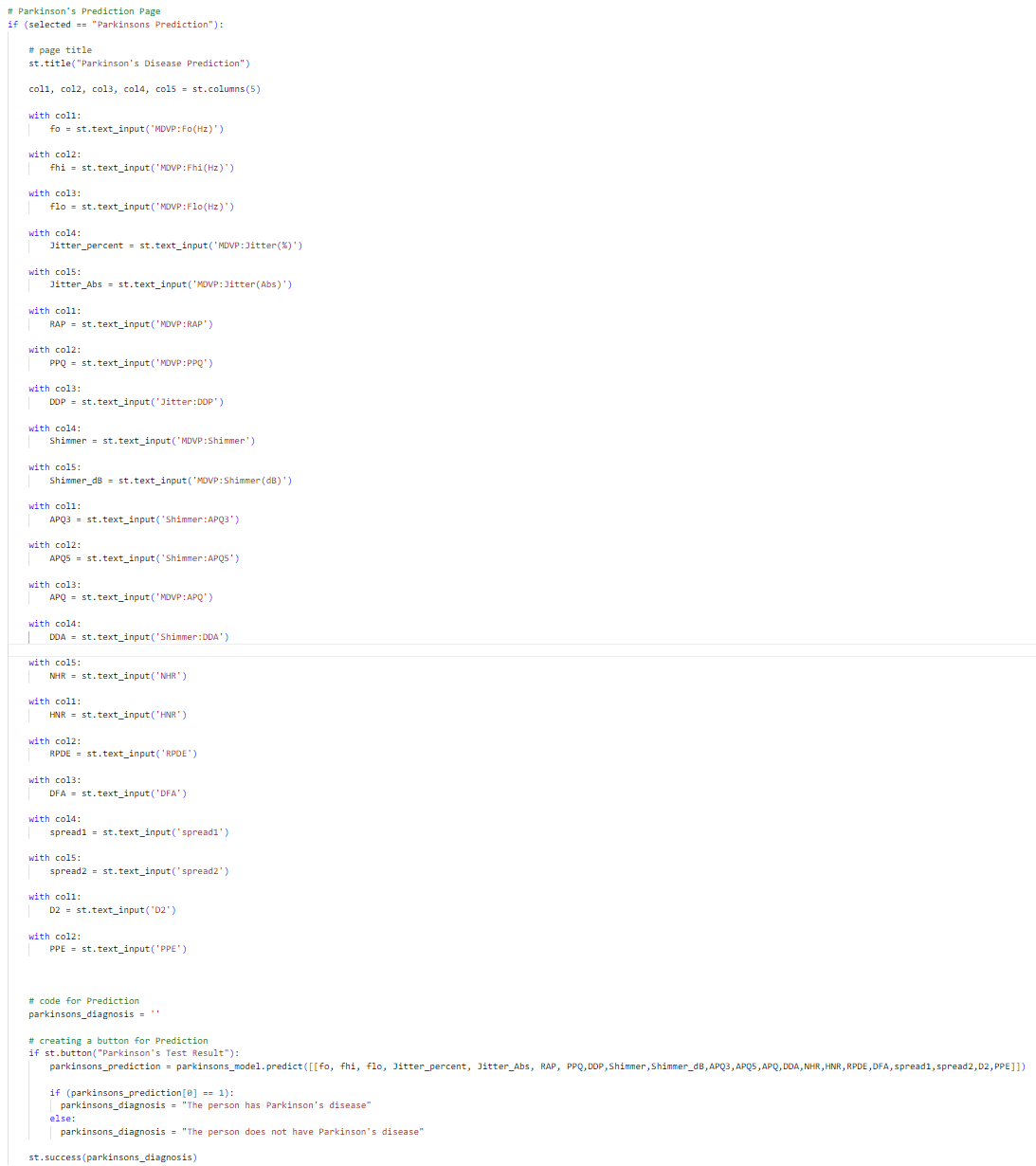


#### Figure 3.3.5.1: X and Y features

The features X and target variable Y is separated. The data is split into training and testing sets. An SVM model with linear kernel is created and trained using training data. The accuracy score of model is calculated and printed on training and test data. Input data is converted into a NumpyArray and predictions are made using trained model. The trained model is saved into a file using pickle. The predicted results are then displayed.

### 3.3.6 Streamlit

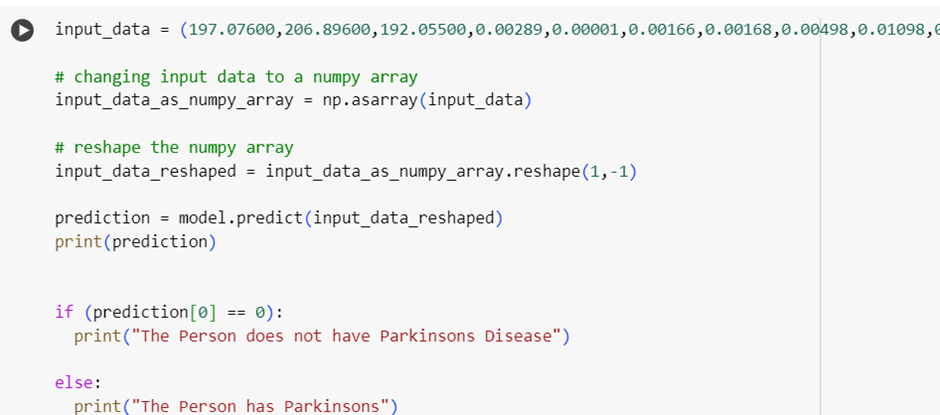
The Streamlit application designed for predicting Parkinson's disease offers an intuitive platform for users to assess the likelihood of the condition based on vocal features. The user interface is structured with columns for easy data input, allowing individuals to provide information on various acoustic measures such as fundamental frequency, jitter, shimmer, and more. Leveraging a pre-trained machine learning model specifically tailored for Parkinson's disease prediction, the application swiftly processes the entered data upon user submission. The resulting prediction is then communicated to the user through a clear and straightforward message, stating whether the individual is likely to have Parkinson's disease or not.



*Figure 3.3.6.1: Streamlit implementation.*

## 3.4 EXPERIMENTS AND RESULT

Experiment



#### Figure 3.4.1: Experiment

Result



#### Figure 3.4.2: Result

## 3.5 CONCLUSION

The Parkinson’s disease model is a machine learning model used to predict the presence of Parkinson's disease based on acoustic features that are extracted from voice recordings of various people. The dataset which is a CSV file, included various acoustic measures that are commonly used in the analysis of speech signals. The primary focus was on using Support Vector Machine (SVM) with a linear kernel for classification. The model if further refined and validated could contribute to early and non-invasive diagnosis of Parkinson's disease in clinical or home settings.

## 3.6 REFERENCES

[1]<https://www.mayoclinic.org/diseases-conditions/parkinsons-disease/symptoms-causes/syc-20376055>

[2]<https://www.kaggle.com/datasets/vikasukani/parkinsons-disease-data-set>

# CHAPTER 4: LUNG CANCER

## 4.1 ABSTRACT

This project employs logistic regression to develop a lung cancer prediction system, utilizing a dataset of health parameters. Rigorous preprocessing, including analysis and normalization, ensures optimal model compatibility. The logistic regression classifier accurately categorizes individuals based on the likelihood of developing lung cancer. This system enables early detection and intervention, supporting healthcare professionals in timely risk identification. The project contributes to healthcare advancements by providing a comprehensive guide to lung cancer prediction through machine learning.

## 4.2 INTRODUCTION

Lung cancer poses a substantial worldwide health challenge, representing a leading cause of illness and mortality. Timely detection and prediction of lung cancer are pivotal in mitigating its impact and enhancing patient prognosis. The Lung Cancer Prediction System project endeavors to create a reliable and precise system for forecasting the probability of lung cancer in individuals.

4.3 METHODOLOGY

### 4.3.1 Work Flow Explanation

**Lung cancer:**

It representation directly in this text format is challenging, but I can describe a simple flowchart for a Lung Cancer Prediction System.

**Data Collection:**

* Collect a dataset with age, smoking history, genetic factors, and respiratory indicators. Ensure a balanced representation of individuals with and without lung cancer for comprehensive analysis

**Data Preprocessing:**

* Perform exploratory data analysis.
* Handle missing values and outliers.
* Normalize numerical features and encode categorical variables.
* Explore feature correlations.

**Feature Selection:**

* Select relevant features based on their importance in predicting lung cancer.

**Data Splitting:**

* Divide the dataset into training and testing sets.

**Logistic Regression Model:**

* Train a logistic regression model using the training dataset.
* Fine-tune hyperparameters for optimal performance.

**Model Evaluation:**

* Evaluate the model's performance using metrics like accuracy, precision, recall, and F1-score.
* Create a confusion matrix to visualize true positives, true negatives, false positives, and false negatives.

### 4.3.2 Libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

import seaborn as sns

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

from sklearn import preprocessing

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import plot\_roc\_curve

from sklearn.metrics import accuracy\_score, mean\_absolute\_error , mean\_squared\_error, confusion\_matrix, median\_absolute\_error,classification\_report, f1\_score,recall\_score,precision\_score

**import pandas as pd:**

Imports the pandas library and aliases it as pd. Pandas is used for data manipulation and analysis.

**import numpy as np**:

Imports the NumPy library and aliases it as np. NumPy is used for numerical operations in Python.

**import matplotlib.pyplot as plt**:

This line imports the matplotlib.pyplot module, providing a MATLAB-like interface for creating plots, and aliases it as plt in Python

**import seaborn as sns:**

Imports the Seaborn library for statistical data visualization. Seaborn works well with Pandas data structures.

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

Imports the train\_test\_split function from scikit-learn, which is used to split a dataset into training and testing sets.

**from sklearn import preprocessing:**

Imports the preprocessing module from scikit-learn, which is used for data preprocessing tasks like scaling and encoding.

**from sklearn.linear\_model import LogisticRegression**:

Imports the Logistic Regression algorithm from scikit-learn. Logistic Regression is used for binary classification problems.

**from sklearn.metrics import plot\_roc\_curve:**

Imports the plot\_roc\_curve function from scikit-learn, which is used to plot the Receiver Operating Characteristic (ROC) curve.

### 4.3.3 Data Collection and Processing:

After importing the necessary libraries, the next step is to load the dataset into a Pandas data frame.

**lung\_cancer = pd.read\_csv("lung cancer.csv")**

This line reads a CSV file named "lung cancer.csv" using pandas and assigns the resulting DataFrame to the variable lung\_cancer.

**lung\_cancer.head()**

**A screenshot of a calendar

Description automatically generated**

**lung\_cancer.shape**

**lung\_cancer.info()**

lung\_cancer.shape:Retrieves the number of rows and columns in the CSV file, displaying the dataset dimensions.

lung\_cancer.info(): Offers comprehensive details about the CSV file, including data types, non-null values, and memory usage.

A screenshot of a computer

Description automatically generated

#### Figure 4.3.3.1: CSV file information.

This output from **lung\_cancer.info()** provides a concise summary of the CSV file:

The DataFrame has 309 entries (rows) and 16 columns.

Data types include integers (int64) and objects.

The 'GENDER' and 'LUNG\_CANCER' columns are of object type, possibly indicating categorical variables.

No null values are present in any column.

**lung\_cancer.isnull().sum()**

The line **lung\_cancer.isnull().sum() is** used to check for missing values in the lung\_cancer DataFrame and obtain the count of null values for each column.

**A list of medical information

Description automatically generated with medium confidence**

#### Figure 4.3.3.2: CSV file information.

**lung\_cancer.describe()**

The 'describe' function would output statistical information such as count, mean, standard deviation, minimum, 25th percentile, median, 75th percentile, and maximum for each numerical column in the lung\_cancer DataFrame.

A table with text on it

Description automatically generated with medium confidence

#### Figure 4.3.3.3: Statistical measures.

Checking the distribution of target variables to see how many people with lung cancer

**print(lung\_cancer.GENDER.value\_counts())**

From the above keyword, we get a count of lung cancer people in the given data set.

### 4.3.4 General code explanation:

This code iterates through each column in the lung\_cancer DataFrame, creating count plots for categorical variables using Seaborn. The plots display the distribution of unique values in each column, aiding in visualizing the categorical data distribution. The loop ensures a separate plot for each column, and plt.show() displays all the plots.

A computer code on a white background

Description automatically generated

#### Figure 4.3.4.2: lung\_cancer coloumn plotting.

This code iterates through each column in the lung\_cancer DataFrame, creating pie charts for categorical variables. Each pie chart represents the distribution of unique values in a specific column. The value\_counts() method computes the frequency of each unique value, and plot(kind="pie") generates the pie chart. The autopct="%1.1f%%" parameter displays the percentage on each pie slice. The loop ensures a separate pie chart for each column, and plt.show() displays all the charts. Rotation of x-axis labels by 45 degrees enhances readability.

This code iterates through each column in the lung\_cancer DataFrame, creating count plots for categorical variables using Seaborn. The plots visually represent the distribution of unique values in each column, aiding in data exploration. The loop ensures a dedicated plot for each column, and **plt.show()** displays all the plots together for comprehensive visualization.

A close-up of a computer screen

Description automatically generated

#### Figure 4.3.4.4: seaborn plotting.

**plt.legend(["Cancer", "Not Cancer"]):**

Adds a legend to the plot, providing labels for the different categories represented by the **"SHORTNESS OF BREATH"** variable·

**plt.xticks(rotation=90):**

Rotates the x-axis labels by 90 degrees for better readability, especially useful when dealing with categorical variables.

### 4.3.5 Splittting the data into graph:

A screenshot of a computer

Description automatically generated

#### Figure 4.3.5.1: Splitting of data(x).

**lung\_cancer.drop**(columns="LUNG\_CANCER", **axis=1):** This part removes the column labeled "LUNG\_CANCER" from the DataFrame lung\_cancer. The drop method is used to exclude specific columns, and axis=1 indicates that we are dropping columns (as opposed to rows)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size = 0.2, random\_state = 3, stratify = Y)

*A screenshot of a computer code

Description automatically generated*

#### figure 4.3.5.1: Splitting of data

### 4.3.6 Model Training with Logistic Regression:

The logistic regression model training process for lung cancer prediction, the algorithm learns from a labeled dataset comprising features and corresponding outcomes.

**model = LogisticRegression()**

The line **model = LogisticRegression()** initializes a Logistic Regression model, creating a blank slate for subsequent training and prediction tasks in your code.

**X\_train\_prediction = model.predict(X\_train):**

This line uses the trained model (model) to make predictions on the training dataset **(X\_train).** The predict method applies the learned patterns to the input features and generates corresponding predictions, which are stored in the variable **X\_train\_prediction.**

### 4.3.7 Model Evaluation with Accuracy Score:

Once the logistic regression model is trained, it is essential to evaluate its performance. The accuracy score is a metric that measures the proportion of correctly predicted instances. It is calculated by dividing the number of correct predictions by the total number of predictions.

**A screenshot of a computer code

Description automatically generated**

#### Figure 4.3.7.1: Training data Accuracy

**model.predict(X\_train):** The model makes predictions on the training data **(X\_train) ,**trying to guess whether each person has lung cancer or not based on the features in the training set.

**test\_data\_accuracy** = round(**accuracy\_score**(X\_test\_prediction, Y\_test)\*100,2)

print("Accuracy on Test Data: ", test\_data\_accuracy)

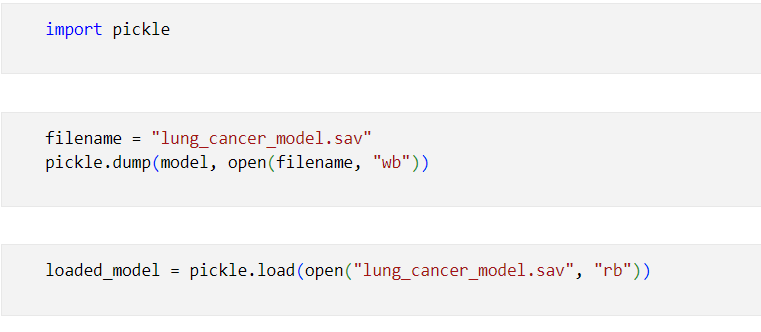
These lines of code calculate and display the accuracy of a logistic regression model on the test dataset. The accuracy\_score function compares the model's predictions **(X\_test\_prediction)** with the actual outcomes **(Y\_test).** The result is converted to a percentage, rounded to two decimal places, and stored in the variable test\_data\_accuracy

### 4.3.7.1 Calculating the Mean :

print("Mean Absolute Error:", mean\_absolute\_error**(Y\_test, X\_test\_prediction))**: This line calculates and prints the Mean Absolute Error (MAE) between the actual outcomes (Y\_test) and the predicted values **(X\_test\_prediction**).

train\_prediction = model.predict**(X\_train)**

training\_data\_accuracy=accuracy\_score**(X\_train\_prediction, Y\_train)**

****

*Figure 4.3.7.1.1: saving sav file.*

* Utilizes the pickle.load() method to deserialize and load the content of the file. This reconstructs the original machine learning model object that was previously saved.
* **open("lung\_cancer\_model.sav", "rb"**): Opens the file named "lung\_cancer\_model.sav" in binary read mode **("rb").**

### 4.3.8 Streamlit Explation:

Implemented a user-friendly Lung Cancer Prediction System using Logistic Regression in a Streamlit web app. The interface allows users to predict lung cancer likelihood based on health parameters, featuring a sidebar for model switching. Users input data like age, gender, and smoking history; code efficiently converts categorical features to numeric for model compatibility. Clicking "Lung Cancer Test Result" activates the logistic regression model, presenting clear predictions for proactive health insights. The approach prioritizes accessibility, contributing to user-centric health management.

## 4.4 EXPERIMENTS AND RESULTS

#### Figure 4.4.1: Experiment output

A screenshot of a computer

Description automatically generated

#### Figure 4.4.2: Streamlit output.

## 4.5 CONCLUSION

In conclusion, Using logistic regression, the Lung Cancer Prediction System proves to be a crucial tool for respiratory health early identification and care. Built upon a foundation of rigorous preprocessing and data analysis, the model illustrates how machine learning may revolutionise the way healthcare practitioners are supported. With accurate prognoses of lung cancer risk based on many health metrics, this method provides proactive personalised care. In the long run, this study will lead to improved patient outcomes and proactive disease management in the field of lung cancer by advancing predictive healthcare and highlighting the critical role that machine learning integration into medical practices plays in this regard.

4.6 REFERENCES

[1][https://github.com/g-shreekant/Heart-Disease-Prediction-using-Machine-Learnin](https://github.com/g-shreekant/Heart-Disease-Prediction-using-Machine-Learning)

[2]<https://www.kaggle.com/datasets/yusufdede/lung-cancer-dataset>