MSc Data Science Dissertation

Title

AI-Driven Alzheimer's Diagnosis and Stage Prediction Using Clinical Data and MRI Scans

Author

Syed Ali Murtaza

Supervisor

Maybin Muyeba

May-2025

A Dissertation Submitted in Partial Fulfilment of the Requirements for the Degree of Master of Data Science at University of Salford

Title

AI-Driven Alzheimer's Diagnosis and Stage Prediction Using Clinical Data and MRI Scans

By

Syed Ali Murtaza

Supervisor

Maybin Muyeba

School of Science, Engineering and Environment University of Salford, Manchester, United Kingdom

Academic Year: May-2025

Abstract

Alzheimer disease is very progressive neurodegenerative disorder that significantly effects a lot of people in United Kingdom and as well as in many other countries. There is a lot of research going on to understand and control this disease. Early prediction and accurate staging are important for timely intervention and care planning.

As a result, this project includes two data sets (Clinical data set, MRI scans data set) to investigate both early prediction and stage classification of Alzheimer disease.

In the first part of the project, supervised machine learning techniques were applied to clinical data set to diagnose whether a patient has Alzheimer disease or not. The analysis reveals that certain features like behavioural problem, memory complaints etc have greater impact for predicting the presence of disease. The best model which was Random Forest achieved an accuracy of 95% on test set. Explainable AI was also used in this part to highlight which features are influencing the prediction model's decision.

In the second part of the project, a CNN model was developed and trained it on MRI scans to further classify the patients into 4 categories i.e. Non-Dementia, Very Mild Dementia, Mild Dementia, and Moderate Dementia. This model achieved an accuracy of 86% on test set. Explainable AI (LIME) was also used in this part to highlight the brain region influencing the CNN model's decision.

This project aimed to help doctors to make informed decisions about their patients by using the miracles of ML and DL techniques.

Keywords: -

Alzheimer's Disease, Dementia Classification, MRI Scans, Machine Learning, LIME, Medical Diagnosis.

Acknowledgements: -

First of all, I would like to thank **Maybin Muyeba**, my project supervisor, for his guidance, support, and encouragement during this dissertation period. His thoughtful inputs, constructive criticism, and constant motivation played a very important role in shaping the quality and focus of this research. His emphasis on research ethics and academic rigor has been truly inspiring.

I am also sincerely thankful to the **faculty members of the School of Computing, Science & Engineering department** for their continuous support, encouragement, and for providing a conducive environment for learning and research.

Lastly, I would like to thank my **parents** for their financial support, emotional support, and patience during this degree of MSC Data Science. Their support has been a constant source of strength during the journey.

Any unintentional omission in this brief acknowledgement does not reflect a lack of gratitude.

Table of Contents

Catalog

List of Figures

List of Tables

List of Abbreviations

EDA	Exploratory Data Analysis
MRI	Magnetic Resonance Imaging
LIME	Local-Interpretable-Model-Agnostic Explanations
HIPAA	Health-Insurance-Portability and Accountability-Act
GDPR	General-Data-Protection-Regulation
AUC	Area Under Curve
CNN	Convolutional Neural Network
MMSE	Mini-Mental State Exam
ADL	Activities-of-Daily-Living
CRISP-DM	Cross-Industry-Standard-Process-for-Data-Mining
GPUs	Graphics-Processing-Units
TPUs	Tensor-Processing-Units
RFECV	Recursive-Feature-Elimination-with-Cross-Validation
BMI	Body-Mass-Index
ROC	Receiver-Operating-Characteristic
e.g	for example
i.e	that is
etc	and so forth
AUC-ROC	Area Under the Receiver Operating Characteristic curve
CI	Confidence of Interval
ReLU	Rectified Linear Unit
TP	True-Positives
TN	True-Negatives
FN	False-Negatives
FP	False-Positives
AI	Artificial Intelligence

Chapter 1: Introduction

1.1 General Overview : -

Cognitive decline, memory loss, and substantial impairment in daily routine are symptoms of Alzheimer's disease, a neurodegenerative brain disease. It is a significant public health concern in the UK and many other nations, and it is also one of the leading causes of dementia. Everyone is now highly interested in this. To better understand and manage this condition, a great deal of research is ongoing. The majority of researchers have shown that this illness disproportionately affects the elderly. So, maybe we can manage this sickness if we can foresee it timely. Reducing healthcare costs and enhancing patients' quality of life are two additional benefits of early illness prediction and accurate disease labelling.

The use of Data Science has been on the rise, with encouraging results, in recent years. One term for the healthcare industry in the past was "information rich but knowledge poor" due to the massive amounts of untapped patient data. Now, however, by using various data science and AI tools, these patterns may be discovered, opening the door to prospects for personalised therapy and prompt intervention.

This project focuses on two key approaches for Alzheimer disease prediction.

1. Clinical data in a structured format: 2149 patients' medical records were analysed using 35 characteristics. The data was then subjected to several approaches, such as EDA, feature selection, predictive modelling, and so on. When it came time to forecast whether or not the patient had Alzheimer's, the final model reached a precision of 95%.
2. Unstructured MRI Scan: -A convolutional neural network (CNN) model was trained using 33984 unstructured enhanced MRI images to categorise Alzheimer's disease into four additional subtypes: non-dementia, mild dementia, very mild dementia, and moderate dementia. For the goal of illness classification into four more categories, this CNN model obtains an accuracy of 86%.
3. To help the expert understand which elements are contributing to the decision-making process, an explainable AI (Lime) was also integrated with the two methodologies mentioned earlier.

The goal of this research is to develop a model that can aid the doctor in making a diagnosis by integrating data-driven modelling with explainable AI approaches such as LIME. This project adhered to a traditional methodology.

In this section, I will describe the methodology that would allow this Alzheimer's detection system to operate in a real-world clinical context. Imagine a real-life situation where a patient goes to the doctor and describes their symptoms, such as memory loss, changes in behaviour, MMSE scores, age, etc. The doctor then inputs all of the patient's symptoms into our system and runs the model. If the model predicts a positive result, meaning the patient has Alzheimer's, the doctor will then suggest an MRI brain scan to further investigate. The process of uploading the MRI scan to the system begins after its acquisition. Next, a convolutional neural network (CNN) model evaluates the scan and assigns it to one of four dementia severity levels: mild, moderate, very

mild, or non-existent. The decision-making process is further aided by an explainable AI (LIME) that reveals where region of the brain is involved.

This two-pronged strategy aids in both early diagnosis and the appropriate staging of diseases using medical imaging. It helps doctors and nurses make quick, well-informed judgements about patient diagnosis and treatment.

1.2 Aims and Objectives: -

The primary objective of this research is to apply deep learning and machine learning methods to MRI scan and clinical datasets. Accurate disease labelling and early prediction will benefit from this. In order to aid professionals in making timely and well-informed choices about patient care and treatment planning, this initiative aims to provide a two-stage diagnostic method.

The objectives of the project are

1. Specifically, to scour the clinical data set for information pertaining to Alzheimer's disease symptoms and medical background.
2. Look for patterns and trends in all the characteristics.
3. Find the most important characteristics by using feature extraction methods.
4. Get the data ready for training by preprocessing it.
5. The goal is to create and test an AI model that can use clinical data to determine the likelihood of Alzheimer's disease in any given patient.
6. One goal of explainable AI is to decipher and clarify the significance of features on the model's decision-making process during the initial phase.
7. Making a system that doctors may use to help diagnose Alzheimer's disease based on a patient's symptoms entered by the doctor.
8. In order to thoroughly examine and study the MRI pictures.
9. Examining its economic impact review.
10. Getting every picture to seem the same in terms of size, shape, resolution, etc.
11. Image preprocessing on magnetic resonance imaging (MRI) images makes them CNN model trainable.
12. Creating a convolutional neural network (CNN) model to categorise magnetic resonance imaging (MRI) brain images into four distinct phases of Alzheimer's disease: non-dementia, mild dementia, moderate dementia, and very mild dementia.
13. In the second stage, we will use explainable AI to pinpoint which parts of the brain impact the CNN's categorisation results.
14. The goal is to compare and contrast the two models, the ML model and the CNN model, by using suitable measures including recall, confusion matrix, accuracy, and precision.
15. In order to show how this dual-modality method might be used in a practical clinical setting to aid clinicians in diagnosing and staging Alzheimer's disease.

1.3 Adapted approach: -

In order to properly integrate ML and DL models for Alzheimer disease diagnosis and tagging, our project adhered to the CRISP-DM approach. A CRISP-DM is a popular methodology for data science projects; it typically includes a six-stage life cycle that helps with data science project conception, development, and deployment. The techniques that were used are summarised below.

1. Business Background: This project's goal is to aid medical providers in the early detection and stage prediction of Alzheimer's disease. A two-step procedure is involved in this undertaking. Using a variety of supervised machine learning algorithms trained on an Alzheimer disease clinical data set, the project's initial stage aims to create a doctor-assisted system that can detect the illness based on the information a doctor provides about a patient. To determine the patient's stage, the second step involves training the algorithm with 33986 MRI scans. In addition, they created a method that doctors may use to help determine a patient's dementia stage by uploading an MRI image. Both models are accompanied by an explainable AI for more clarity.
2. A clinical data collection including patient demographics, behavioural issues, and test scores (e.g., MMSE, food control, etc.) was one of two sets of data used in this study. Different from this dataset are MRI pictures of the brain. Structure, data types, distribution of values, missing value handling, various correlations, etc. were the primary areas of attention in the first investigation. Additional details are provided in the EDA section of the study.
3. In order to get the data ready for training, this step included cleaning and preparing it. The clinical data set underwent feature extraction, feature normalisation, and the removal of irrelevant information. The section of this study devoted to data preparation goes into more depth on the subject. Rescaling to 244*244, normalisation, greyscale conversation, and other image preparation steps are used for MRI data sets. The report's later portion goes into much detail on this point.
4. Modelling: The first of two sections, this phase involves using several supervised machine learning algorithms to clinical data sets in order to determine whether a patient has Alzheimer's disease or not. Section 2 included training a convolutional neural network (CNN) model using magnetic resonance imaging (MRI) scans in order to categorise the severity of the illness into four levels: non-dementia, very mild, mild, and moderate.
5. A combination of the clinical ML model's and the MRI scans' accuracy, precision, recall, and F1-score were used to assess the model's performance. Both parts, such feature significance score in the first and visualising critical brain areas affecting the CNN's judgements in the second, brought explainability using LIME. In comparison to the CNN model's 86% accuracy, the clinical ML model achieved 95% accuracy on the test set.
6. Setting up: Despite the model's lack of live deployment, it may still be used in a real-world situation on the notebook: A doctor may use a patient's symptoms to obtain a rough idea of whether they have Alzheimer's disease. If the results come back positive, they can order an MRI and use their laptop to use a convolutional neural network (CNN) model to determine the patient's illness stage. For the purpose of Alzheimer's disease staging and early diagnosis, this two-stage approach is useful.

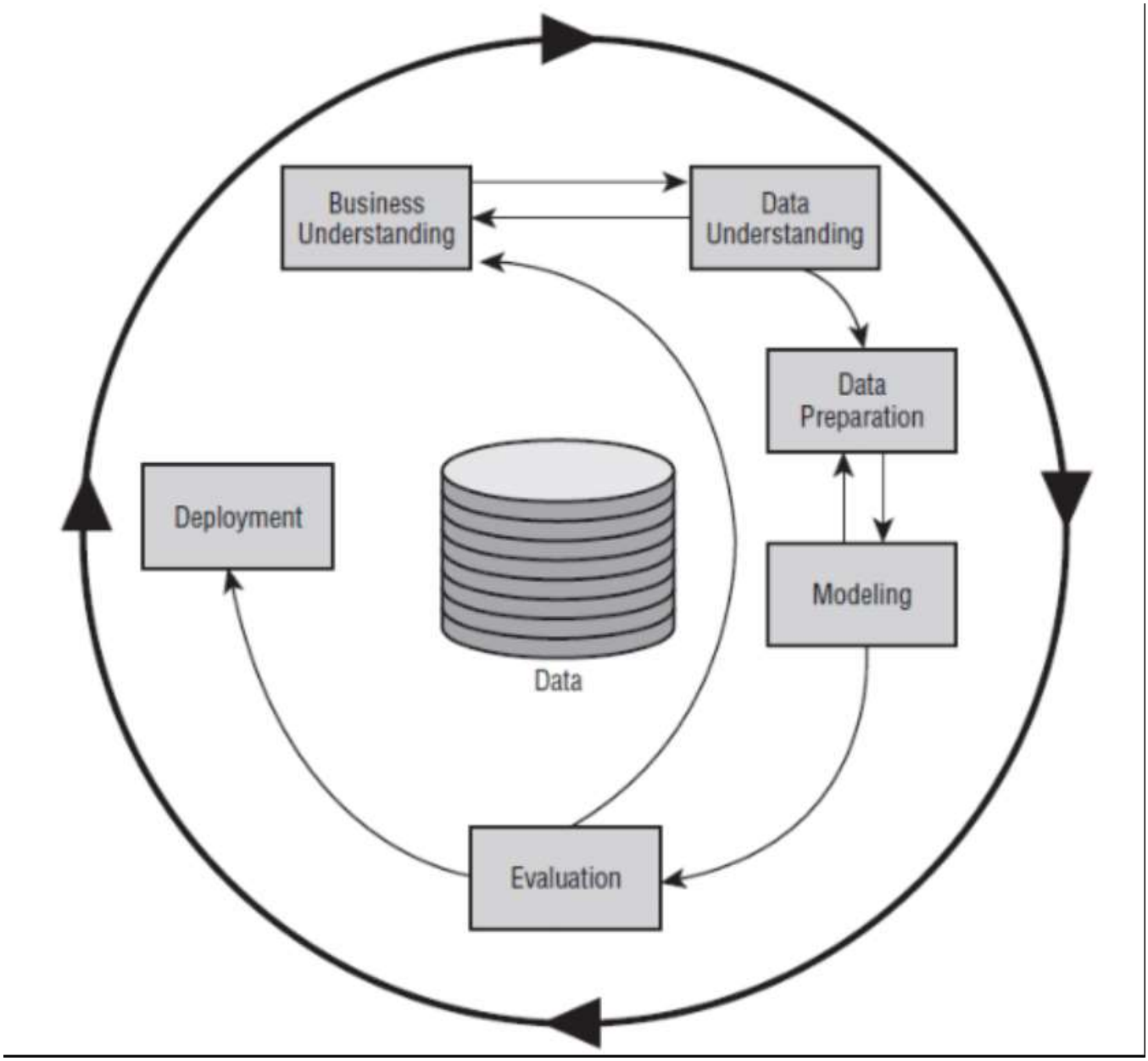


Figure 1-1 CRISP-DM Flow chart(Source: Wikipedia contributors. (n.d.). *Cross-industry standard process for data mining*. Wikipedia. https://en.wikipedia.org/wiki/Cross-industry_standard_process_for_data_mining)

1.4 Tools for Data Analysis: -

As this project is two-step detection system so for the first step which is for clinical data set, Jupiter notebook was used and for the second step, which was of MRI scan, google collab was used.

Jupiter Notebook: -

For this project, Jupiter notebook was used for clinical data set of Alzheimer disease. This platform allowed step-by-step exploration of patient data, cleaning, feature engineering, model training and evaluation. Its attractive nature made it ideal for prototyping the machine learning workflow for predicting the presence of Alzheimer disease from clinical symptoms.

Google Collab: -

For this project, google collab was used for training and evaluating the Convolutional Neural Network (CNN) on MRI scan data. This platform provided the required computational power like GPUs/TPUs to handle high level resolution of brain scan images and to accelerate the training of deep learning model. In addition, it also facilitated integration with Google drive, allowing efficient data storage, loading and sharing.

1.5 Ethical Consideration: -

These are the following ethical considerations that were kept in mind while making the project.

Data Privacy and Confidentiality: -

The data of Alzehmier patients in this project were anonymised to protect the patient identity and ensure complience with data protection regulations, including the UK General Data Protection Regulation (UK GDPR) and the Data Protection Act 2018. No personal information like name, NI number, address was used at any stage of this project.

Informed Consent: -

The data for this project was taken from publicly available repositories like Kaggle and it is assumed that dataset provider obtained informed consent from patients for their data to be used for research purposes.

Ethical Approval: -

Since this project was conducted as a part of academic project for MSC data science by using the publicly available data, however formal ethical approval from a research ethics committee of university of Salford was taken.

Responsible use of AI: -

Special care was taken that the machine learning and deep learning models developed were not misused.

Transparency and Interpretability: -

Explainable AI like Lime was used in this project to ensure transparency, helping both the researchers and healthcare professionals understand how model decisions were made.

1.6 Project Outline: -

Chapter 1: This Chapter just outlines the general introduction. What approaches will be used, what tools will be used, what ethical issues were kept in mind while making this project.

Chapter 2: In this chapter, several literature sources were discussed regarding deep learning and machine learning.

Chapter 3: This chapter is further divided into two parts.

1- Data Underacting, Data Preparation and EDA for Clinical data set.
2- Data Underacting, Data Preparation and EDA for MRI scans.

Each part contains the following: -

- Number of rows, columns, and data types.
- Description of key features and the target variable.
- Handling missing values (imputation or removal).
- Removing duplicates and correcting inconsistent data.
- Fixing data types and formatting.
- Scaling or normalizing numeric features.
- Handling outliers, creating train-test split.
- Creating or modifying features to improve model performance.
- Applying feature selection methods (e.g., RFECV).
- Removing irrelevant or redundant features.
- Visualizing data distributions (histograms, boxplots, etc.).
- Identifying correlations and feature-target relationships.
- Detecting class imbalance or unusual patterns.
- Summarizing insights that informed model choice.

Chapter 4: This chapter is further divided into two parts.

- Analysis (e.g., Modelling, Evaluating, Deploying) for Clinical data set.
- Analysis (e.g., Modelling, Evaluating, Deploying) for MRI scans.

Each part consists of following: -

- First, a brief overview.
- An account of the primary issues encountered and the measures taken to address them.
- Outline of the analysis (e.g., developing models, assessing results).
- Detailed account of the steps taken to fulfil the criteria using the created model.
- The procedures and resources used should be explained and justified.
- The outcomes and discoveries.
- In summary.

Chapter 5: This chapter contain the Critical Evaluation

It has following objectives

- Assessment in light of predetermined goals.
- The strategy is reviewed, and any changes are explained.
- Practical knowledge gained from the undertaking.

Chapter 6: This chapter contains the conclusions and future work of project

It contains following: -

- Analysed the results and how they stack up against the current golden standard.
- Potential enhancements to the product and additional tasks, if needed.
- A consideration of any matters pertaining to law, society, ethics, or the profession.

Chapter 7: This chapter includes the references

A devastating brain disorder, Alzheimer's disease disproportionately affects the elderly. The condition worsens with time, progressing from minor memory difficulties to major ones to the point that the affected individual may have trouble recognising loved ones or even remembering crucial information. If physicians can detect the illness early and determine its stage, they can provide timely and appropriate therapy. However, due to a shortage of medical professionals and an excess of patients, prompt diagnosis and treatment are becoming more challenging. This is why helpful smart systems are necessary.

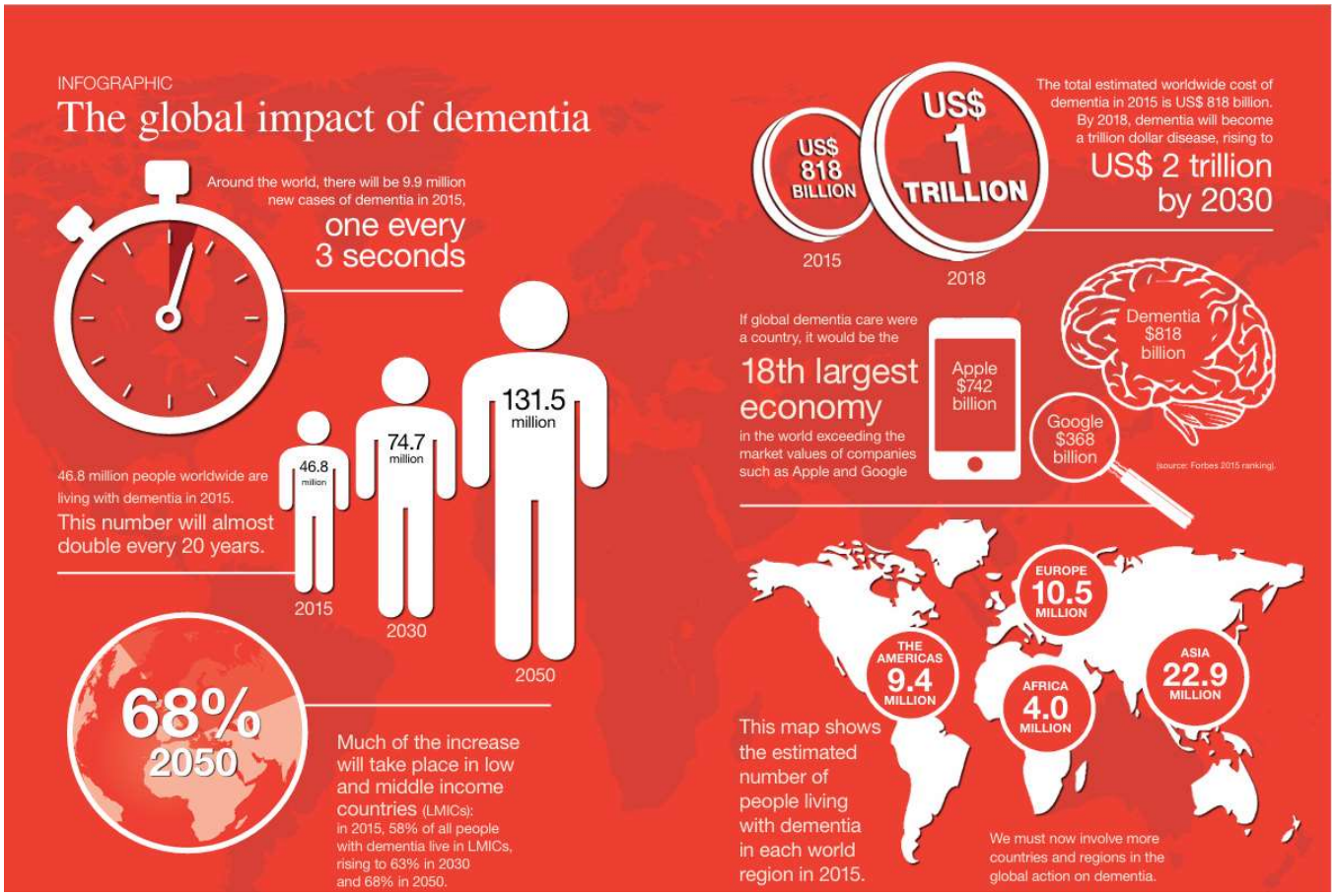


Figure 2-1 Statistics of Alzehmier Patients in world(Source: International, Alzheimer's Disease, et al. *World Alzheimer Report 2015: The Global Impact of Dementia: An Analysis of Prevalence, Incidence, Cost and Trends*. Sept. 2015. *www.alzint.org*, https://www.alzint.org/resource/world- alzheimer-report-2015/.)

Recently, tools like **machine learning (ML)** and **deep learning (DL)** have become popular in healthcare. These tools can help us understand patterns in medical data and make useful predictions. In their book, **Han, Kamber, and Pei (2012)** explain how data mining techniques like classification and pattern finding are useful in health data. Similarly, **Larose and Larose (2015)** show how ML techniques like Random Forest, Logistic Regression, and Support Vector Machine can be used to find out if someone has a disease early on.

In this project, I worked on two parts. The first part uses patient data like age, clinical test results, and family history to detect if someone might have Alzheimer's. I used GridSearchCV on different machine learning models like **Random Forest**, **Logistic Regression**, and **Support Vector Machine**, after which i came to know that random forest is the best model for my project and these models are explained in the book by **Tan, Steinbach, and Kumar (2014)**.

The second part looks at **MRI brain images** to figure out the stage of Alzheimer's using **Convolutional Neural Networks (CNNs)**. I used methods from **Zollanvari (2023)** and **Lee (2019)** who wrote about how to use Python and deep learning for images.

Before building the models, I cleaned and analysed the data using **Exploratory Data Analysis (EDA) and** remove irrelevant features to help the model learn better. These steps were based on methods described by **Leskovec, Rajaraman, and Ullman (2014)**.

I also used a tool called **LIME** to explain how the AI is making decisions. This is very important in healthcare, because doctors need to trust the system and understand how it works (**Zollanvari, 2023**).

To sum it up, this project builds a two part AI system first one that uses patient records and another that uses MRI scans to help doctors detect and stage Alzheimer's disease in a smart, clear, and helpful way.

2.1 Data Mining for Healthcare Prediction

The foundation for this project was in understanding patterns within clinical data. According to Han, Kamber, and Pei (2012), data mining enables the discovery of meaningful patterns from large datasets, which is crucial in healthcare where early detection of diseases like Alzheimer's can make a major difference. The author of this book emphasizes classification techniques like decision trees and ensemble models, and to make a gridSearchCV for selecting the better model and in this way, I came up with **Random Forest,** because it was suggested by GridSearchCV as the best performing model for my project**.**

Tan, Steinbach, and Kumar (2014) also explain that classification problems in detail.

2.2 Feature Selection and Preprocessing

Feature engineering and selection are critical to improving model accuracy. Larose and Larose (2015) highlight that eliminating irrelevant features reduces overfitting and training time. In this project, **Recursive Feature Elimination with Cross-Validation (RFECV)** was used to select the most informative features from the clinical dataset, aligning with best practices discussed in book *Data Mining and Predictive Analytics*.

The preprocessing steps such as handling null values, scaling, and encoding were implemented based on recommendations in Leskovec, Rajaraman, and Ullman (2014), who emphasize the importance of transforming raw data into a form suitable for mining and analysis.

2.3 Model Building and Evaluation

Zollanvari (2023) offers a detailed theoretical view of machine learning model development, including supervised learning techniques, cross-validation, and model tuning. This guided the model building process, which included training, tuning with **GridSearchCV**, and evaluation through metrics such as accuracy, precision, recall, and F1-score.

For model evaluation, Lee (2019) stresses the use of proper validation techniques and confusion matrices to understand model performance holistically. These strategies were implemented in both parts of the project e.g clinical data classification and MRI image staging .

2.4 Deep Learning for MRI Analysis

The second part of the project involved MRI-based Alzheimer's staging. CNNs are widely used in this domain due to their strength in extracting spatial features from images. While the books used focused more on general data science techniques, the theoretical knowledge from *Machine Learning with Python* by Zollanvari (2023) helped in understanding the role of convolutional layers, activation functions, and transfer learning strategies in deep learning.

2.5 Explainability and Ethical AI

Transparency and explainability are critical in healthcare applications. The project uses **LIME** (Local Interpretable Model-Agnostic Explanations) to help explain predictions to clinicians and patients. The ethical responsibility of data scientists is addressed by Zollanvari (2023), who emphasizes model interpretability and transparency as professional obligations in real-world deployments.

Similarly, data protection and fairness are key concerns. Han et al. (2012) mention the importance of respecting data integrity and ensuring fairness in predictive model.

Chapter 3. Data Underacting, Data Preparation and EDA

This chapter describes the data understanding, data preparation and EDA that was carried out before training the model. So, for this project 2 data sets were used.

1. A structured clinical dataset consisting of tabular patient data.
2. An unstructured MRI brain scan dataset consisting of thousands of images.

Both datasets were individually understood, analysed, pre-processed through tailor techniques suitable for their nature.

3.1 Clinical Dataset

3.1.1 Data structure and description

The clinical data set contains medical records of 2149 patients with 35 features.

Key aspects:

- Data type: Structured (CSV format)
- Source: publicly available at Kaggle (Alzheimer's Disease Dataset)
- Target label: Diagnosis (binary classification)

Variable description: -

Patient Information:

Table 3-1 Patient Information

Variable	Description
PatientID	It is the distinct patient identifier for each patient ranging from 4751 to 6900.

Demographic Details:

Table 3-2 Demographic Details

Variable	Description
Age	Age of patients (60years–90 years)
Gender	Gender : Male is coded as 0, Female as 1
Ethnicity	Ethnic groups: 0 represents Caucasian, 1 represents African American, 2 represents Asian, and 3 represents Other.
EducationLevel	For education level, 0 corresponds to None, 1 to High School, 2 to Bachelor's, and 3 to Higher education

Lifestyle Factors:

Table 3-3 Lifestyle Factors

Variable	Description
BMI	The Body Mass Index (BMI) falls within the range of 15 to 40.
Smoking	Smoking status: 0 indicates Non-smoker, 1 indicates Smoker..
AlcoholConsumption	The alcohol consumption varies from 0 to 20 units.
PhysicalActivity	Physical activity duration: Between 0 and 10 hours
DietQuality	Diet quality rating: Scored between 0 and 10.
SleepQuality	Sleep quality rating: Ranges from 4 to 10.

Medical History:

Table 3-4 Medical History

Variable	Description
FamilyHistoryAlzheimers	Presence of Alzheimer's in family history: 0 for No, 1 for Yes
CardiovascularDisease	Cardiovascular disease status: 0 indicates Absent, 1 indicates Present
Diabetes	Diabetes presence: 0 represents No, 1 represents Yes
Depression	Depression status: 0 indicates Absent, 1 indicates Present.
HeadInjury	Head injury history: 0 indicates No, 1 indicates Yes
Hypertension	Hypertension status: 0 indicates Absent, 1 indicates Present.

Clinical Measurements:

Table 3-5 Clinical Measurements

Variable	Description
SystolicBP	Systolic blood pressure levels range from 90 to 180 mmHg.
DiastolicBP	Diastolic blood pressure values fall between 60 and 120 mmHg
CholesterolTotal	Total cholesterol concentrations range from 150 to 300 mg/dL.
CholesterolLDL	LDL cholesterol values fall within the 50–200 mg/dL range
CholesterolHDL	HDL cholesterol levels fall in the range of 20 to 100 mg/dL
CholesterolTriglycerides	Triglyceride concentration varied between 50 and 400 mg/dL

Cognitive and Functional Assessments:

Table 3-6 Cognitive and Functional Assessments

Variable	Description
MMSE	This assessment tool measures cognitive performance, producing a score between 0 and 30.
FunctionalAssessment	Functional independence is evaluated on a scale of 0 to 10, with higher scores reflecting better ability.
MemoryComplaints	Participants reported whether they experienced memory issues, coded as 0 for "No" and 1 for "Yes."
BehavioralProblems	Any noticeable shifts in behavior were recorded using a binary scale: 0 for no change and 1 for observed change).
ADL	Daily living independence was rated on a 0–10 scale, where higher scores indicate greater self-sufficiency.

Symptoms

Table 3-7 Symptoms

Variable	Description
Confusion	Whether the patient exhibits confusion (0 for No, 1 for Yes).
Disorientation	Disorientation observed (0 for No, 1 for Yes).
PersonalityChanges	Personality changes noted (0 for No, 1 for Yes).
DifficultyCompletingTasks	Trouble with task completion (0 for No, 1 for Yes).
Forgetfulness	Forgetfulness symptoms (0 for No, 1 for Yes).

Diagnosis Information

Table 3-8 Diagnosis Information

Variable	Description
Diagnosis	Final Alzheimer's diagnosis (0 = No, 1 = Yes).

Confidential Information

Table 3-9 Confidential Information

Variable	Description

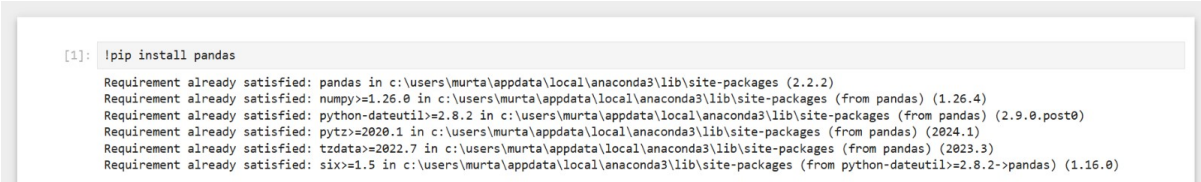
DoctorInCharge	Confidential variable with placeholder value “XXXConfid”.

3.1.2 Initial Data Exploration of Clinical Dataset

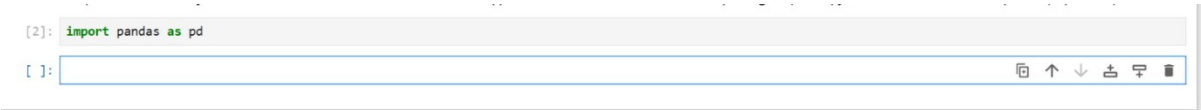
EDA on clinical dataset was produced using the python libraries such as pandas, Seaborn, and Matplotlib

First, I opened Jupiter and opened a new notebook by clicking on New> notebook. Then I selected python3 as a kernel.

As we know panda's library is famous for manipulation and structuring of data efficiently. So, first I installed it.



After installation, pandas was imported as follow



Than all possible libraries were installed as follow

```
# Importing necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier  # Added import
from sklearn.neighbors import KNeighborsClassifier  # Added import
from sklearn.linear_model import LogisticRegression  # Added import
from sklearn.svm import SVC  # Added import
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
from sklearn.feature_selection import SelectFromModel
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
from tabulate import tabulate
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.feature_selection import RFECV
import seaborn as sns
import matplotlib.pyplot as plt


import warnings
warnings.filterwarnings("ignore")

%matplotlib inline
```

Data importing: -

The data set was loaded into a dataframe from its location using the following code and showed its content as

```
[11]: df = pd.read_csv(r"C:\Users\murta\OneDrive\Desktop\Dessertation\datasets\alzheimers_disease_data.csv")
```

```
[12]: df
```

[12]:

	PatientID	Age	Gender	Ethnicity	EducationLevel	BMI	Smoking	AlcoholConsumption	PhysicalActivity	DietQuality	...	MemoryComplaints	BehavioralProl
0	4751	73	0	0	2	22.927749	0	13.297218	6.327112	1.347214	...	0	
1	4752	89	0	0	0	26.827681	0	4.542524	7.619885	0.518767	...	0	
2	4753	73	0	3	1	17.795882	0	19.555085	7.844988	1.826335	...	0	
3	4754	74	1	0	1	33.800817	1	12.209266	8.428001	7.435604	...	0	
4	4755	89	0	0	0	20.716974	0	18.454356	6.310461	0.795498	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
2144	6895	61	0	0	1	39.121757	0	1.561126	4.049964	6.555306	...	0	
2145	6896	75	0	0	2	17.857903	0	18.767261	1.360667	2.904662	...	0	
2146	6897	77	0	0	1	15.476479	0	4.594670	9.886002	8.120025	...	0	
2147	6898	78	1	3	1	15.299911	0	8.674505	6.354282	1.263427	...	0	
2148	6899	72	0	0	2	33.289738	0	7.890703	6.570993	7.941404	...	0	

2149 rows × 35 columns

Figure 3-1View of data

Understanding the Data: -

As data frames was addressed through an index and a column label, So I used index and column label to have a complete look on the data frames as follows

```
[14]: df.iloc[:,:]
```

[14]:

	PatientID	Age	Gender	Ethnicity	EducationLevel	BMI	Smoking	AlcoholConsumption	PhysicalActivity	DietQuality	...	MemoryComplaints	BehavioralProl
0	4751	73	0	0	2	22.927749	0	13.297218	6.327112	1.347214	...	0	
1	4752	89	0	0	0	26.827681	0	4.542524	7.619885	0.518767	...	0	
2	4753	73	0	3	1	17.795882	0	19.555085	7.844988	1.826335	...	0	
3	4754	74	1	0	1	33.800817	1	12.209266	8.428001	7.435604	...	0	
4	4755	89	0	0	0	20.716974	0	18.454356	6.310461	0.795498	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
2144	6895	61	0	0	1	39.121757	0	1.561126	4.049964	6.555306	...	0	
2145	6896	75	0	0	2	17.857903	0	18.767261	1.360667	2.904662	...	0	
2146	6897	77	0	0	1	15.476479	0	4.594670	9.886002	8.120025	...	0	
2147	6898	78	1	3	1	15.299911	0	8.674505	6.354282	1.263427	...	0	
2148	6899	72	0	0	2	33.289738	0	7.890703	6.570993	7.941404	...	0	

2149 rows × 35 columns

```
[15]: df.head()
```

[15]:

	PatientID	Age	Gender	Ethnicity	EducationLevel	BMI	Smoking	AlcoholConsumption	PhysicalActivity	DietQuality	...	MemoryComplaints	BehavioralProbler
0	4751	73	0	0	2	22.927749	0	13.297218	6.327112	1.347214	...	0	
1	4752	89	0	0	0	26.827681	0	4.542524	7.619885	0.518767	...	0	
2	4753	73	0	3	1	17.795882	0	19.555085	7.844988	1.826335	...	0	
3	4754	74	1	0	1	33.800817	1	12.209266	8.428001	7.435604	...	0	
4	4755	89	0	0	0	20.716974	0	18.454356	6.310461	0.795498	...	0	

5 rows × 35 columns

Figure 3-2 Head of Data

In this way I got a complete view of data.

The head of data frame was viewed as Figure 3-2 Head of Data

The tail of data frame was viewed as follow

```
[16]: df.tail()
```

[16]:

	PatientID	Age	Gender	Ethnicity	EducationLevel	BMI	Smoking	AlcoholConsumption	PhysicalActivity	DietQuality	...	MemoryComplaints	BehavioralProl
2144	6895	61	0	0	1	39.121757	0	1.561126	4.049964	6.555306	...	0	
2145	6896	75	0	0	2	17.857903	0	18.767261	1.360667	2.904662	...	0	
2146	6897	77	0	0	1	15.476479	0	4.594670	9.886002	8.120025	...	0	
2147	6898	78	1	3	1	15.299911	0	8.674505	6.354282	1.263427	...	0	
2148	6899	72	0	0	2	33.289738	0	7.890703	6.570993	7.941404	...	0	

5 rows × 35 columns

Figure 3-3Tail of Data

Shape of data: -

The shape of data frame was viewed as

```
[18]: df.shape
[18]: (2149, 35)
```

Figure 3-4 Shape of Data

The dataset had 35 features and 2149 instances. So, it was rich enough to train a ML model.

Overview of data: -

The quick overview of the data frame like non-null entries in each column and the data type of each column was seen as follows

```
[19]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2149 entries, 0 to 2148
Data columns (total 35 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   PatientID                  2149 non-null   int64
 1   Age                        2149 non-null   int64
 2   Gender                     2149 non-null   int64
 3   Ethnicity                  2149 non-null   int64
 4   EducationLevel             2149 non-null   int64
 5   BMI                        2149 non-null   float64
 6   Smoking                    2149 non-null   int64
 7   AlcoholConsumption         2149 non-null   float64
 8   PhysicalActivity           2149 non-null   float64
 9   DietQuality                2149 non-null   float64
 10  SleepQuality               2149 non-null   float64
 11  FamilyHistoryAlzheimers    2149 non-null   int64
 12  CardiovascularDisease      2149 non-null   int64
 13  Diabetes                   2149 non-null   int64
 14  Depression                 2149 non-null   int64
 15  HeadInjury                 2149 non-null   int64
 16  Hypertension               2149 non-null   int64
 17  SystolicBP                 2149 non-null   int64
 18  DiastolicBP                2149 non-null   int64
 19  CholesterolTotal           2149 non-null   float64
 20  CholesterolLDL             2149 non-null   float64
 21  CholesterolHDL             2149 non-null   float64
 22  CholesterolTriglycerides   2149 non-null   float64
 23  MMSE                       2149 non-null   float64
 24  FunctionalAssessment       2149 non-null   float64
 25  MemoryComplaints           2149 non-null   int64
 26  BehavioralProblems         2149 non-null   int64
 27  ADL                        2149 non-null   float64
 28  Confusion                  2149 non-null   int64
 29  Disorientation             2149 non-null   int64
 30  PersonalityChanges         2149 non-null   int64
 31  DifficultyCompletingTasks  2149 non-null   int64
 32  Forgetfulness              2149 non-null   int64
 33  Diagnosis                  2149 non-null   int64
 34  DoctorInCharge             2149 non-null   object
dtypes: float64(12), int64(22), object(1)
memory usage: 587.7+ KB
```

Figure 3-5 Data type and missing value in each column

From above we can see that there is no null value in whole data set and there are different data types for each column.

Unique Value Analysis of Features: -

df.nunique function was applied to determine the number of unique values in each column. This analysis helped in identifying that which features are numeric and which ones are categorical features. Identifying features was necessary for appropriate preprocessing.

```
[20]: df.nunique()

[20]: PatientID                2149
      Age                        31
      Gender                      2
      Ethnicity                   4
      EducationLevel              4
      BMI                      2149
      Smoking                     2
      AlcoholConsumption       2149
      PhysicalActivity         2149
      DietQuality              2149
      SleepQuality             2149
      FamilyHistoryAlzheimers     2
      CardiovascularDisease       2
      Diabetes                    2
      Depression                  2
      HeadInjury                  2
      Hypertension                2
      SystolicBP                 90
      DiastolicBP                60
      CholesterolTotal         2149
      CholesterolLDL           2149
      CholesterolHDL           2149
      CholesterolTriglycerides 2149
      MMSE                     2149
      FunctionalAssessment     2149
      MemoryComplaints            2
      BehavioralProblems          2
      ADL                      2149
      Confusion                   2
      Disorientation              2
      PersonalityChanges          2
      DifficultyCompletingTasks   2
      Forgetfulness               2
      Diagnosis                   2
      DoctorInCharge              1
      dtype: int64
```

Figure 3-6 Value Count for each column

From above it can be observed that the features which have high numberings are numerical features and features which have low numberings (like 2,4) are categorical features.

Duplicate Rows: -

Duplicate rows were checked within the dataset using the df.duplicated() function, and the total count of duplicate records was printed. This step ensured the removal of redundant data that could bias model training. Following this, null values were also assessed using df.isnull().sum() to further justify data quality and determine whether imputation or removal was necessary during preprocessing.

```
[21]: print("Duplicate rows: ", sum(df.duplicated()))
      print(df.isnull().sum())

Duplicate rows:  0
PatientID                  0
Age                        0
Gender                     0
Ethnicity                  0
EducationLevel             0
BMI                        0
Smoking                    0
AlcoholConsumption         0
PhysicalActivity           0
DietQuality                0
SleepQuality               0
FamilyHistoryAlzheimers    0
CardiovascularDisease      0
Diabetes                   0
Depression                 0
HeadInjury                 0
Hypertension               0
SystolicBP                 0
DiastolicBP                0
CholesterolTotal           0
CholesterolLDL             0
CholesterolHDL             0
CholesterolTriglycerides   0
MMSE                       0
FunctionalAssessment       0
MemoryComplaints           0
BehavioralProblems         0
ADL                        0
Confusion                  0
Disorientation             0
PersonalityChanges         0
DifficultyCompletingTasks  0
Forgetfulness              0
Diagnosis                  0
DoctorInCharge             0
dtype: int64
```

Figure 3-7 Duplicate rows

From above it can be observed that our data is correctly structured that there are no duplicate rows and there are no null values as well.

Feature drop and summary statistics: -

In this step, the PatientID and DoctorInCharger were dropped from the dataset, as they do not contribute to any predictive value for model training. Their presence could introduce unnecessary noise.

Following the removal of these irrelevant attributes, summary statistics such as **mean**, **standard deviation**, **minimum**, **maximum**, and **quartiles** were generated using the df.describe() function. This provided valueable insight into the data distribution and potential outliers.

```
[22]: # drop unnecessary columns
df.drop(["PatientID","DoctorInCharge"], axis=1, inplace=True)

df.describe().T # transpose
```

[22]:

	count	mean	std	min	25%	50%	75%	max
Age	2149.0	74.908795	8.990221	60.000000	67.000000	75.000000	83.000000	90.000000
Gender	2149.0	0.506282	0.500077	0.000000	0.000000	1.000000	1.000000	1.000000
Ethnicity	2149.0	0.697534	0.996128	0.000000	0.000000	0.000000	1.000000	3.000000
EducationLevel	2149.0	1.286645	0.904527	0.000000	1.000000	1.000000	2.000000	3.000000
BMI	2149.0	27.655697	7.217438	15.008851	21.611408	27.823924	33.869778	39.992767
Smoking	2149.0	0.288506	0.453173	0.000000	0.000000	0.000000	1.000000	1.000000
AlcoholConsumption	2149.0	10.039442	5.757910	0.002003	5.139810	9.934412	15.157931	19.989293
PhysicalActivity	2149.0	4.920202	2.857191	0.003616	2.570626	4.766424	7.427899	9.987429
DietQuality	2149.0	4.993138	2.909055	0.009385	2.458455	5.076087	7.558625	9.998346
SleepQuality	2149.0	7.051081	1.763573	4.002629	5.482997	7.115646	8.562521	9.999840
FamilyHistoryAlzheimers	2149.0	0.252210	0.434382	0.000000	0.000000	0.000000	1.000000	1.000000
CardiovascularDisease	2149.0	0.144253	0.351428	0.000000	0.000000	0.000000	0.000000	1.000000
Diabetes	2149.0	0.150768	0.357906	0.000000	0.000000	0.000000	0.000000	1.000000
Depression	2149.0	0.200558	0.400511	0.000000	0.000000	0.000000	0.000000	1.000000
HeadInjury	2149.0	0.092601	0.289940	0.000000	0.000000	0.000000	0.000000	1.000000
Hypertension	2149.0	0.148906	0.356079	0.000000	0.000000	0.000000	0.000000	1.000000
SystolicBP	2149.0	134.264774	25.949352	90.000000	112.000000	134.000000	157.000000	179.000000
DiastolicBP	2149.0	89.847836	17.592496	60.000000	74.000000	91.000000	105.000000	119.000000
CholesterolTotal	2149.0	225.197519	42.542233	150.093316	190.252963	225.086430	262.031657	299.993352
CholesterolLDL	2149.0	124.335944	43.366584	50.230707	87.195798	123.342593	161.733733	199.965665
CholesterolHDL	2149.0	59.463533	23.139174	20.003434	39.095698	59.768237	78.939050	99.980324
CholesterolTriglycerides	2149.0	228.281496	101.986721	50.407194	137.583222	230.301983	314.839046	399.941862
MMSE	2149.0	14.755132	8.613151	0.005312	7.167602	14.441660	22.161028	29.991381
FunctionalAssessment	2149.0	5.080055	2.892743	0.000460	2.566281	5.094439	7.546981	9.996467
MemoryComplaints	2149.0	0.208004	0.405974	0.000000	0.000000	0.000000	0.000000	1.000000
BehavioralProblems	2149.0	0.156817	0.363713	0.000000	0.000000	0.000000	0.000000	1.000000
ADL	2149.0	4.982958	2.949775	0.001288	2.342836	5.038973	7.581490	9.999747
Confusion	2149.0	0.205212	0.403950	0.000000	0.000000	0.000000	0.000000	1.000000
Disorientation	2149.0	0.158213	0.365026	0.000000	0.000000	0.000000	0.000000	1.000000
PersonalityChanges	2149.0	0.150768	0.357906	0.000000	0.000000	0.000000	0.000000	1.000000
DifficultyCompletingTasks	2149.0	0.158678	0.365461	0.000000	0.000000	0.000000	0.000000	1.000000
Forgetfulness	2149.0	0.301536	0.459032	0.000000	0.000000	0.000000	1.000000	1.000000
Diagnosis	2149.0	0.353653	0.478214	0.000000	0.000000	0.000000	1.000000	1.000000

Figure 3-8 Statistics of data

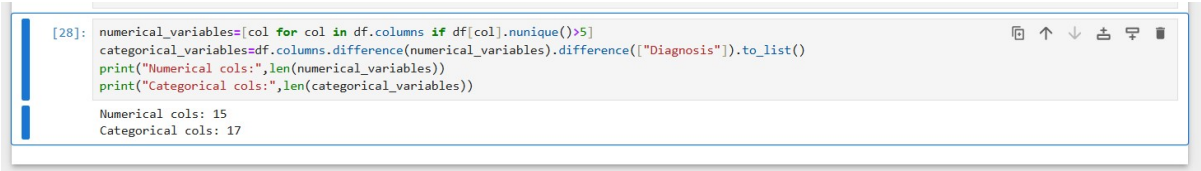
Initial Data Exploration Summary

- The dataset contains a total of **2,149 observations**
- All values are **non-null** & **numerical**
- There are **no duplicate** records or **missing values**
- Now there are **33 total features but original was 35.**

3.1.3 EDA of clinical dataset

Distribution of Numerical & Categorical Features: -

The categorization of numerical and categorical features was performed using the logic that the features which have count more than 5 are considered as numerical variables and the variables which are not numeric and not diagnoses are considered as categorical variables.



```
[28]: numerical_variables=[col for col in df.columns if df[col].nunique()>5]
      categorical_variables=df.columns.difference(numerical_variables).difference(["Diagnosis"]).to_list()
      print("Numerical cols:",len(numerical_variables))
      print("Categorical cols:",len(categorical_variables))

      Numerical cols: 15
      Categorical cols: 17
```

Figure 3-9 Distribution of variables

From above it can be observed that there are 15 numerical columns and 17 categorical columns.

Categorical Features: -

The distribution of all the categorical features were displayed by using the Count plots. For this a smooth colour palette with 5 different colours and custom labels were applied for each categorical column and plot them using count plots.

```
[35]: palette=sns.color_palette('husl',5)
custom_labels={
    'Gender':['Male','Female'],
    'Ethnicity': ['Caucasian', 'African American', 'Asian', 'Other'],
    'EducationLevel': ['None', 'High School', 'Bachelor\'s', 'Higher'],
    'Smoking': ['No', 'Yes'],
    'FamilyHistoryAlzheimers': ['No', 'Yes'],
    'CardiovascularDisease': ['No', 'Yes'],
    'Diabetes': ['No', 'Yes'],
    'Depression': ['No', 'Yes'],
    'HeadInjury': ['No', 'Yes'],
    'Hypertension': ['No', 'Yes'],
    'MemoryComplaints': ['No', 'Yes'],
    'BehavioralProblems': ['No', 'Yes'],
    'Confusion': ['No', 'Yes'],
    'Disorientation': ['No', 'Yes'],
    'PersonalityChanges': ['No', 'Yes'],
    'DifficultyCompletingTasks': ['No', 'Yes'],
    'Forgetfulness': ['No', 'Yes']
}
for col in categorical_variables:
    plt.figure(figsize=(8,5))
    sns.countplot(data=df,x=col,palette=palette)
    plt.title(f'Countplot of {col}')
    labels=custom_labels[col]
    ticks=range(len(labels))
    plt.xticks(ticks=ticks,labels=labels)

plt.show()
```

This generated a count plot for each categorical column, which enabled us to observe each categorical column thoroughly.

Behavioural Problems: -

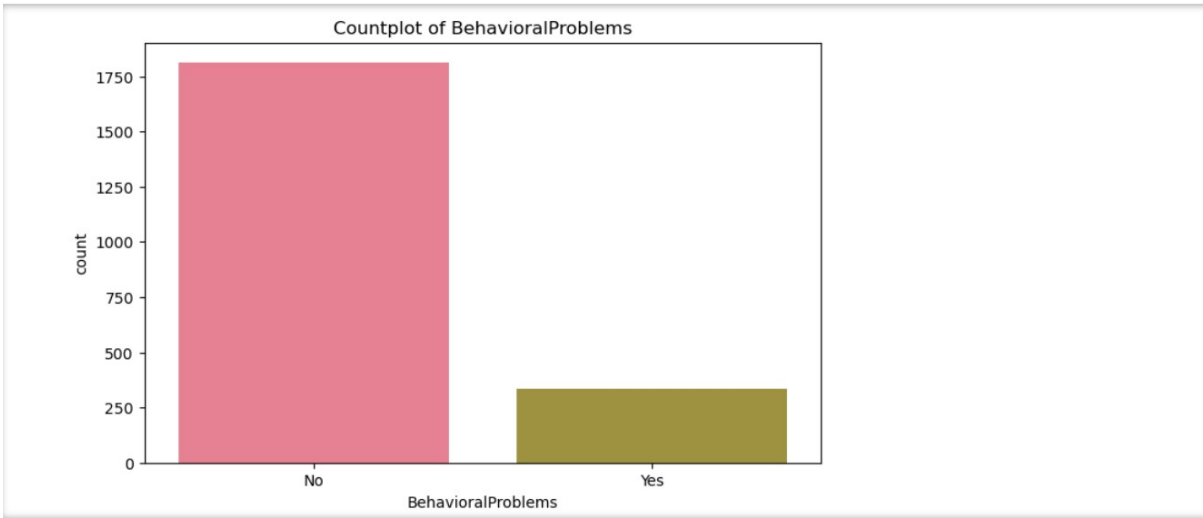


Figure 3-10 Behavioural Problems

From above it can be observed that more people have no behavioural problems in this dataset .

Cardiovascular Disease: -

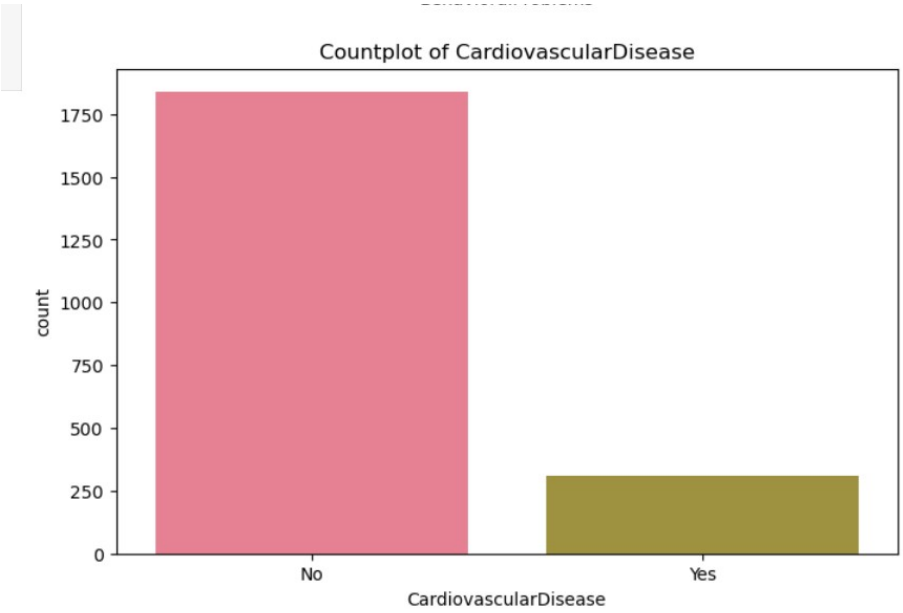


Figure 3-11 Cardiovascular Disease

From above it can be observed that most people do not have cardiovascular disease in this dataset.

Confusion Problem: -

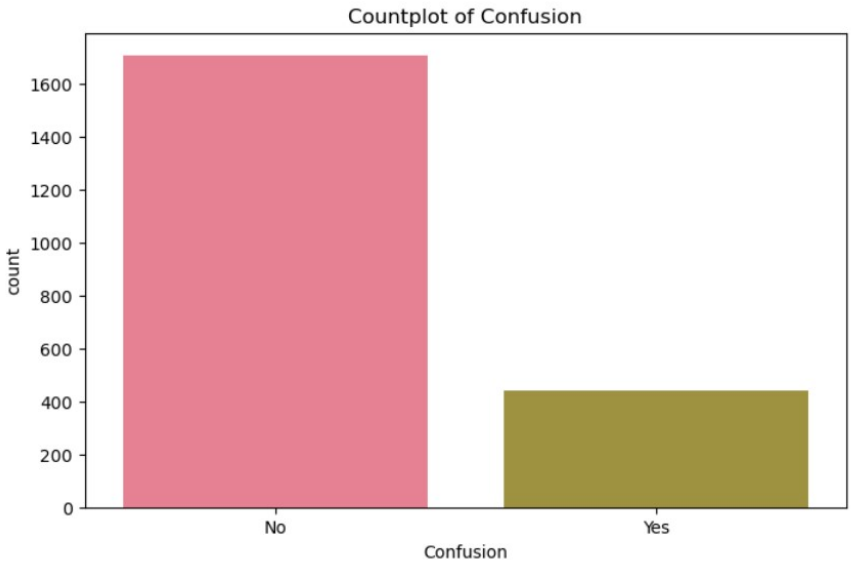


Figure 3-12 Confusion

From above it can be observed that most people do not have confusion problem in this dataset.

Depression: -

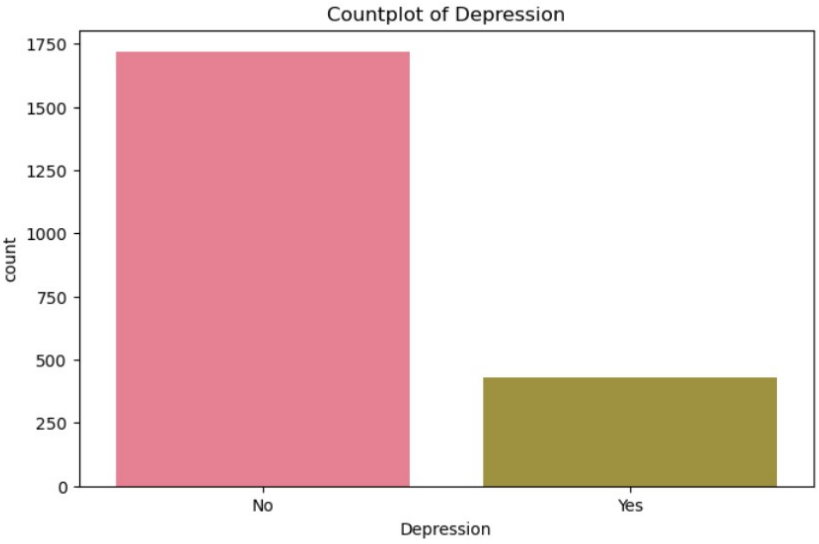


Figure 3-13 Depression

From above it can be observed that most people do not have depression in this dataset.

Diabetes: -

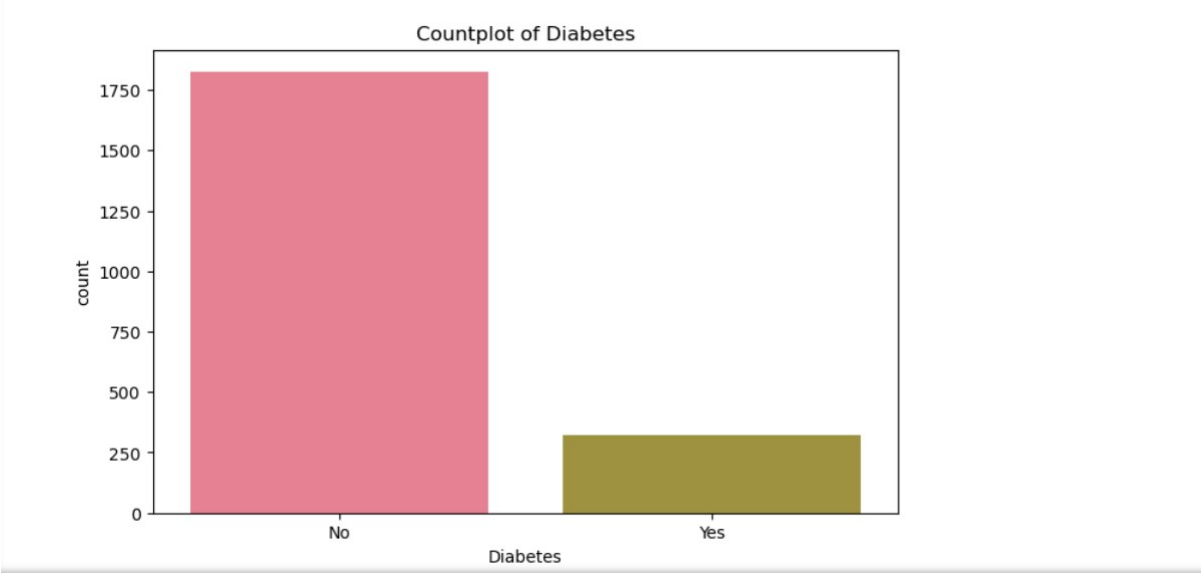


Figure 3-14 diabetes

From above it can be observed that most people do not have diabetes in this dataset.

Difficulty in Completing Tasks: -

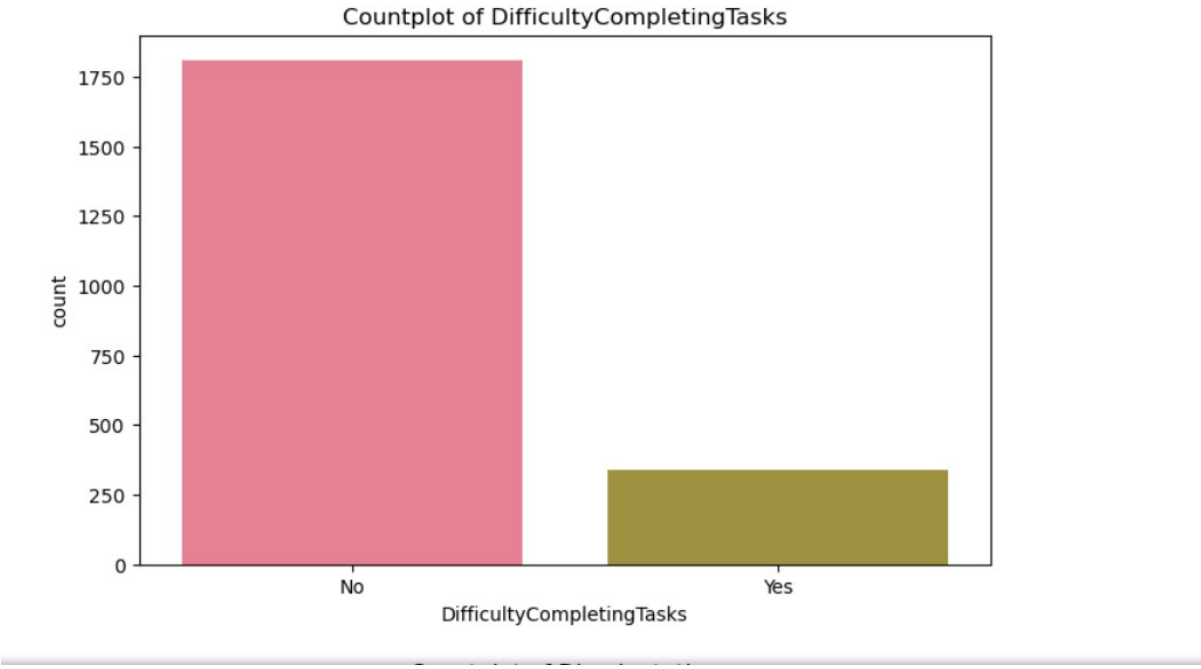


Figure 3-15 Difficulty completing task

From above it can be observed that most people do not have difficulty in completing their tasks in this dataset.

Disorientation: -

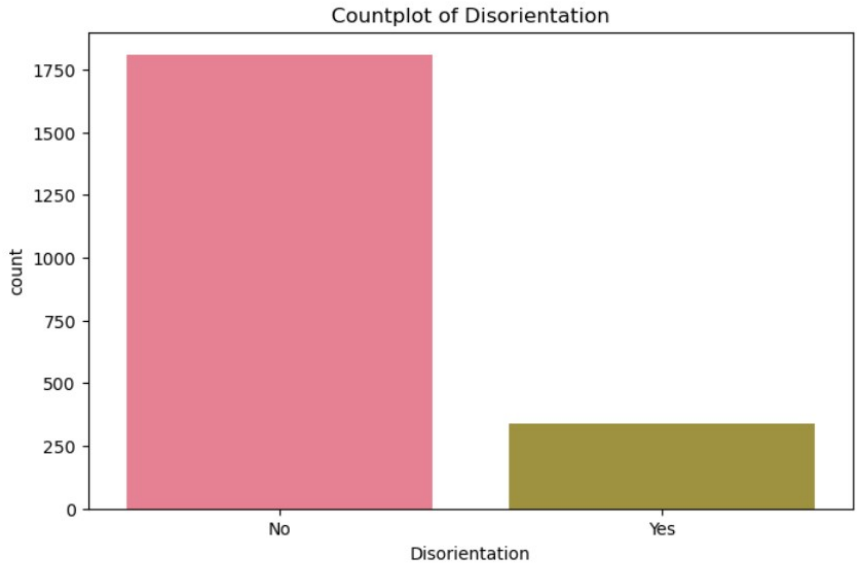


Figure 3-16 Disorientation

From above it can be observed that most people do not have disorientation in this dataset.

Educational Group: -

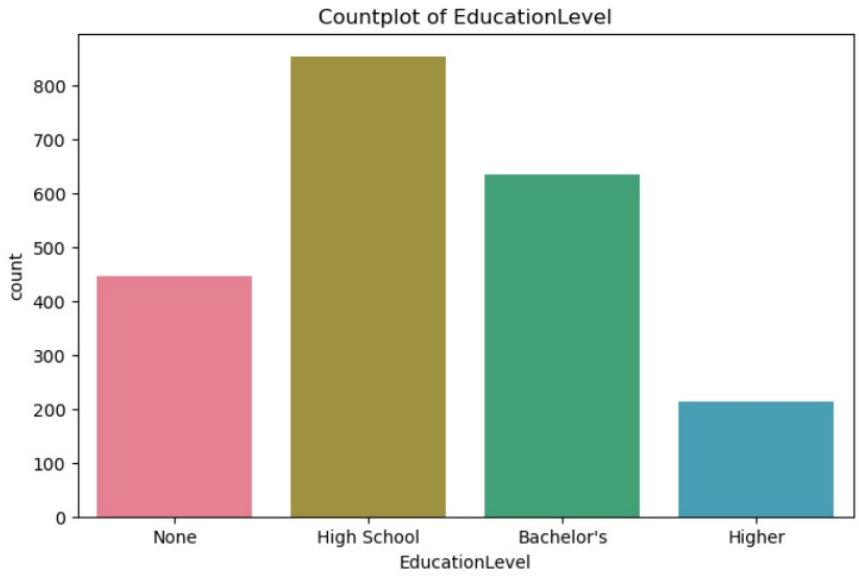


Figure 3-17 Education level

From above it can be observed that high school constitute the highest count in this dataset.

Ethnicity Group: -

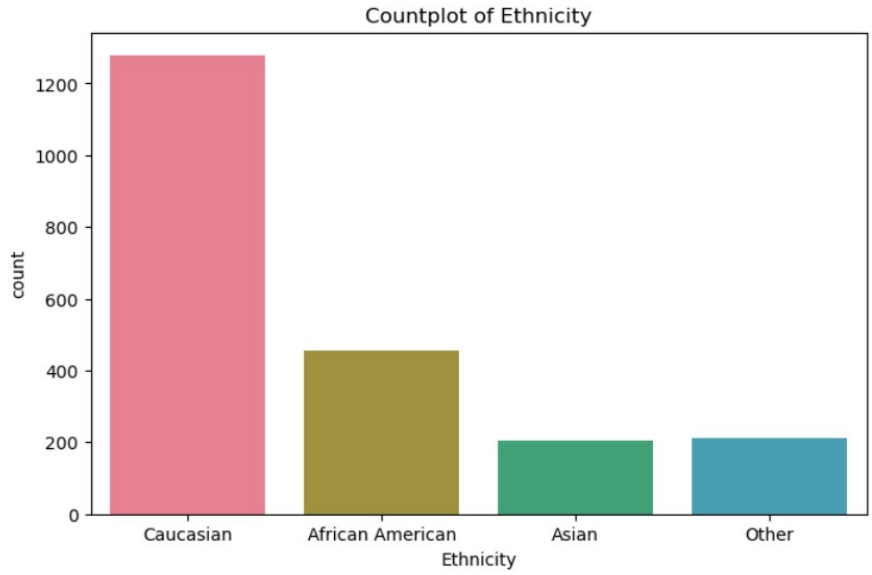


Figure 3-18 Ethnicity

From above it can be observed that Caucasian is considered as highest ethnicity group in this dataset.

Family History of Alzheimer's Disease: -

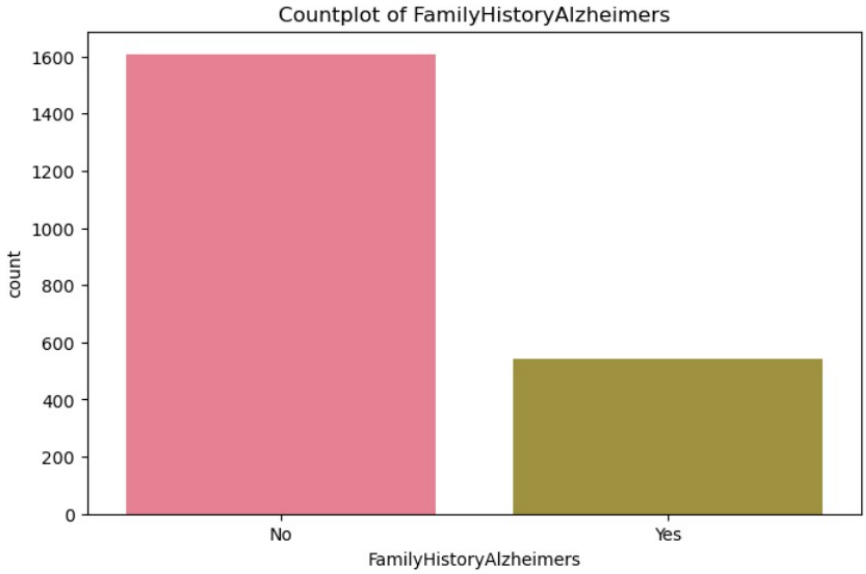


Figure 3-19 Family History of Alzheimer

From above it can be observed that most people do not have any family history of Alzheimer's disease in this dataset.

Forgetfulness: -

Figure 3-20 Forgetfulness

From above it can be observed that most people do not have a problem of forgetfulness.

Gender: -

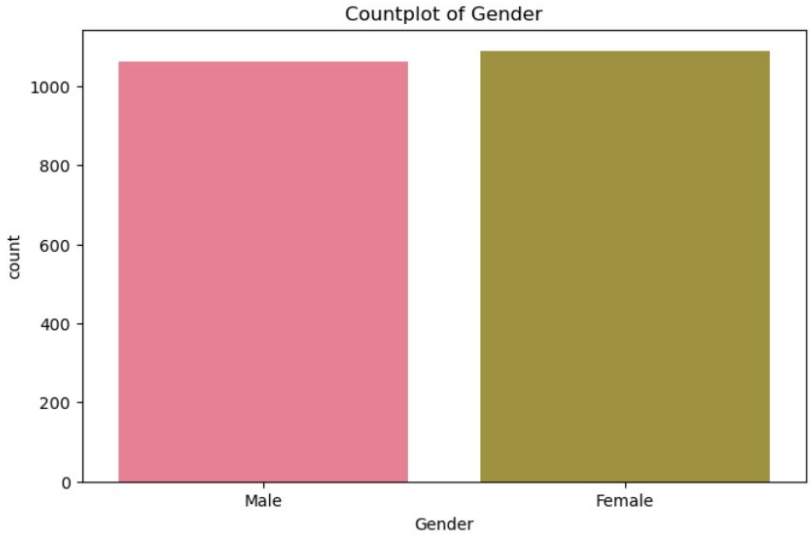


Figure 3-21 Gender

From above it can be observed that the representation of male and female are almost same in this dataset.

Head Injury: -

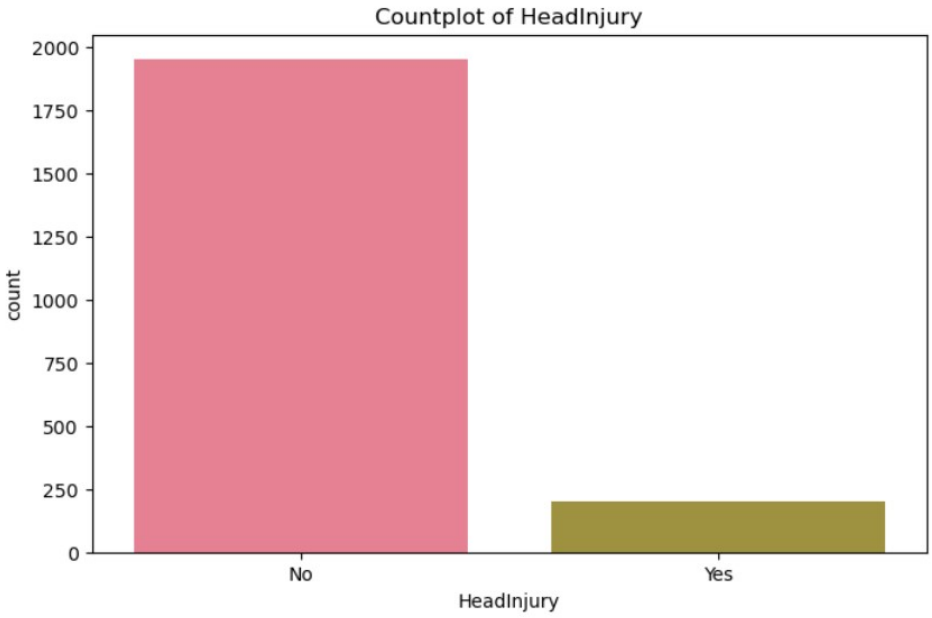


Figure 3-22 HeadInjury

From above it can be observed that most people do not have head injury in this dataset.

Hypertension: -

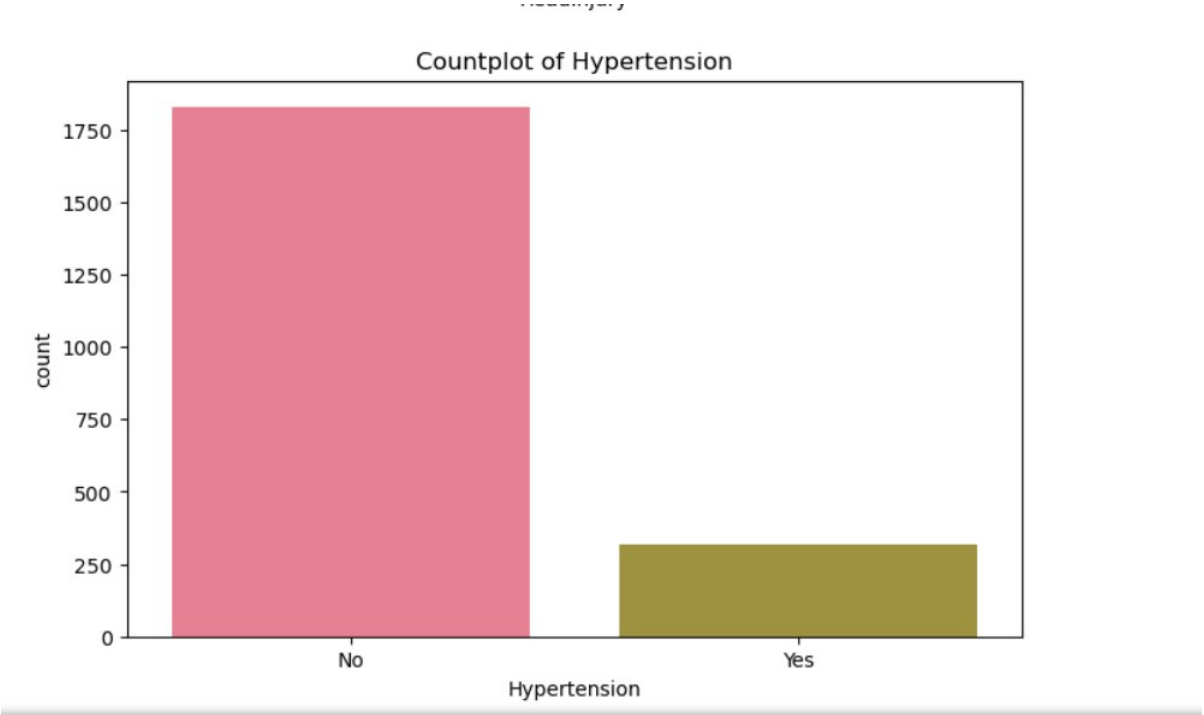


Figure 3-23 Hypertension

From above it can be observed that most people do not have hypertension in this dataset.

Personality Changes: -

Figure 3-24 Personality Changes

From above we can see that most people do not feel personality changes in this dataset.

Smoking: -

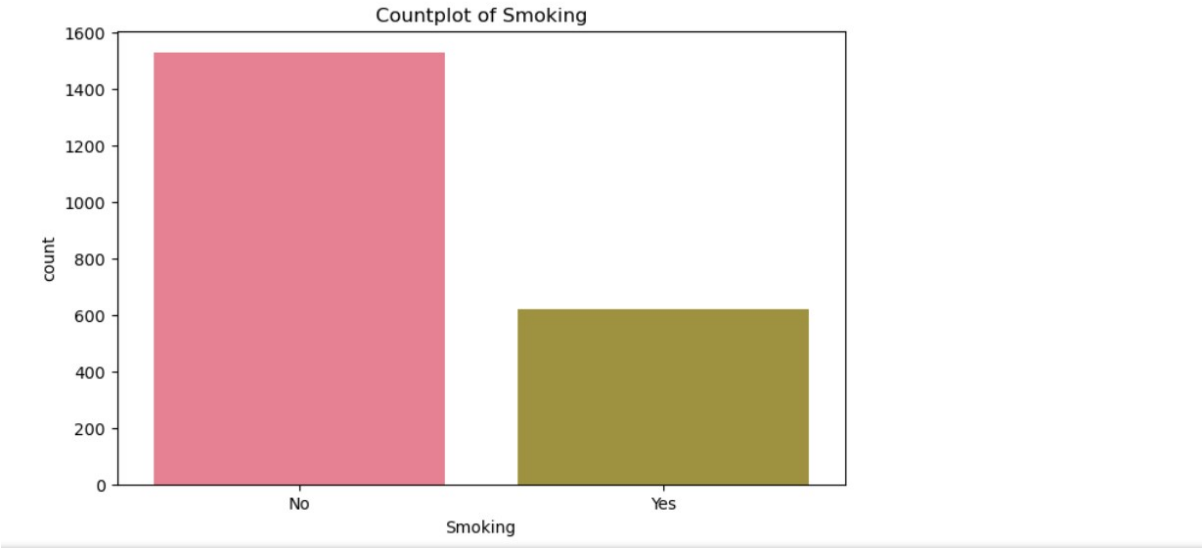


Figure 3-25 Smoking

From above it can be observed that most people do not do smoking in this dataset.

Summary of categorical variables: -

- The dataset predominantly consists of individuals **without disease or other health issues**.
- **Caucasian** is the most represented ethnicity.
- **High School** constitutes the largest educational group, followed by **bachelor's** degree.
- Both **Female** & **Male** are closely equally represented in the dataset.

Numerical variables: -

To explore the distribution of each numerical variable, histograms were created, which allowed assess the distribution patterns (e.g., normal or skewed), detect potential outliers, and identify any trends or patterns in the data. Below is a detailed analysis of each numerical variable based on these histograms.

```
[37]: for col in numerical_variables:
          plt.figure(figsize=(8,5))
          sns.histplot(data=df,x=col,kde=True, bins=20)
          plt.title(f'Distribution of {col}')
          plt.show()
```

Age: -

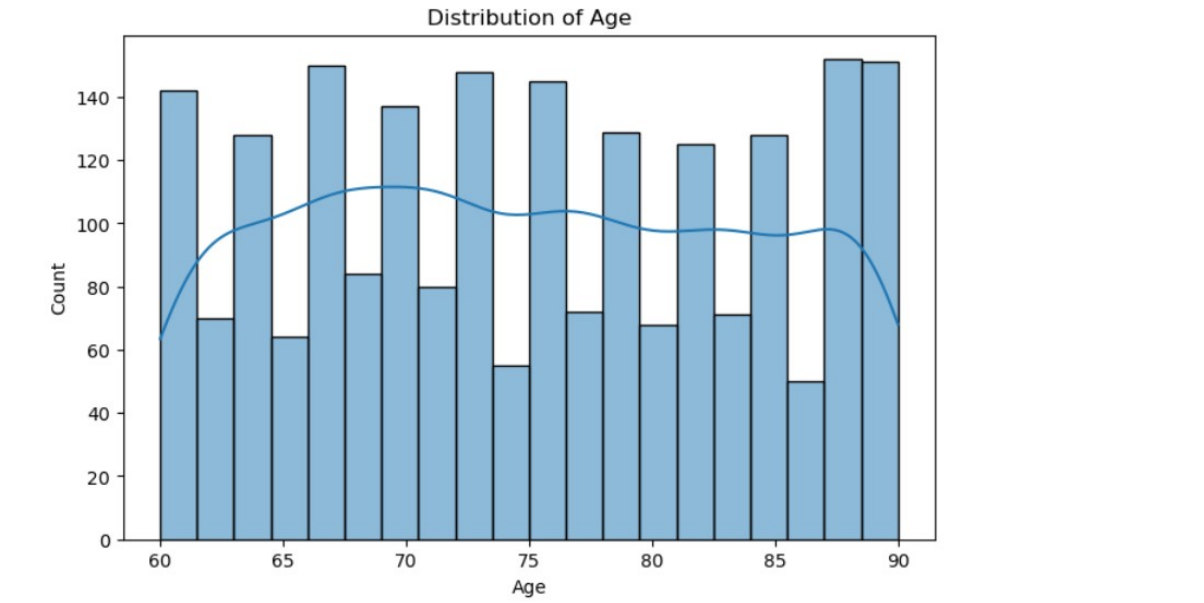


Figure 3-26 Age

From the above graph it can be observed that people's age ranging from 60 years to 90 years. The highest count of people is of 70 years.

BMI: -

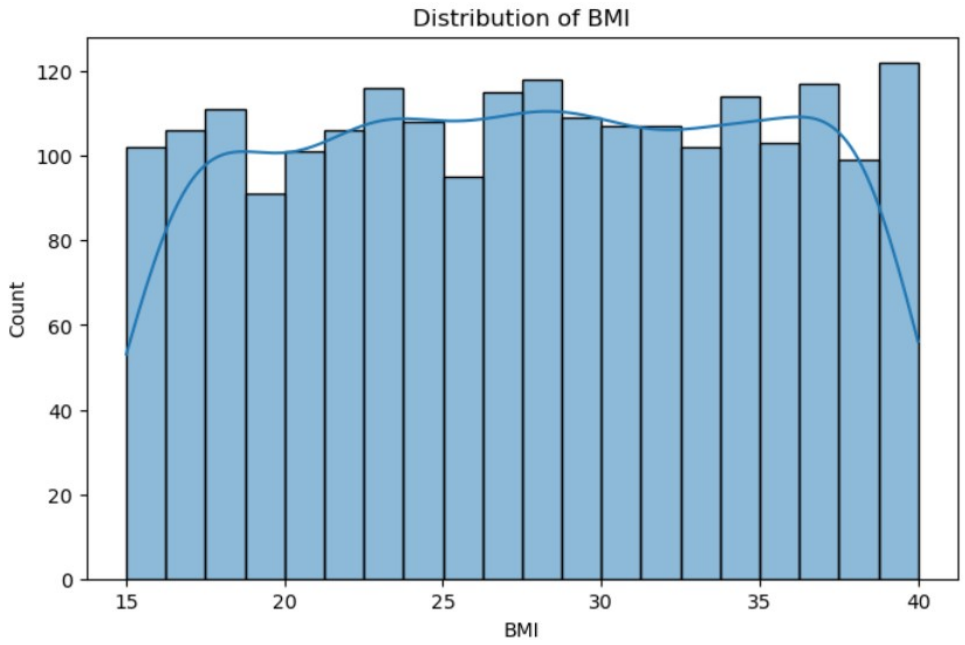


Figure 3-27 BMI

The BMI graph appeared fairly uniform with slight variations. BMI values were evenly distributed across all ages.

Alcohol consumption: -

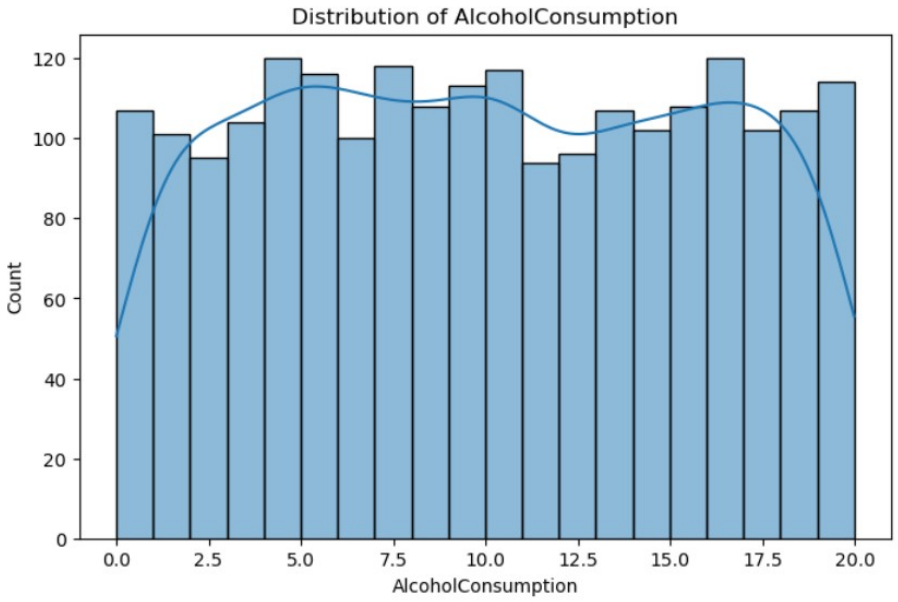


Figure 3-28 Alcohol Consumption

The graph depicting alcohol consumption revealed a relatively even distribution, indicating that the values were spread uniformly among participants.

Physical Activity: -

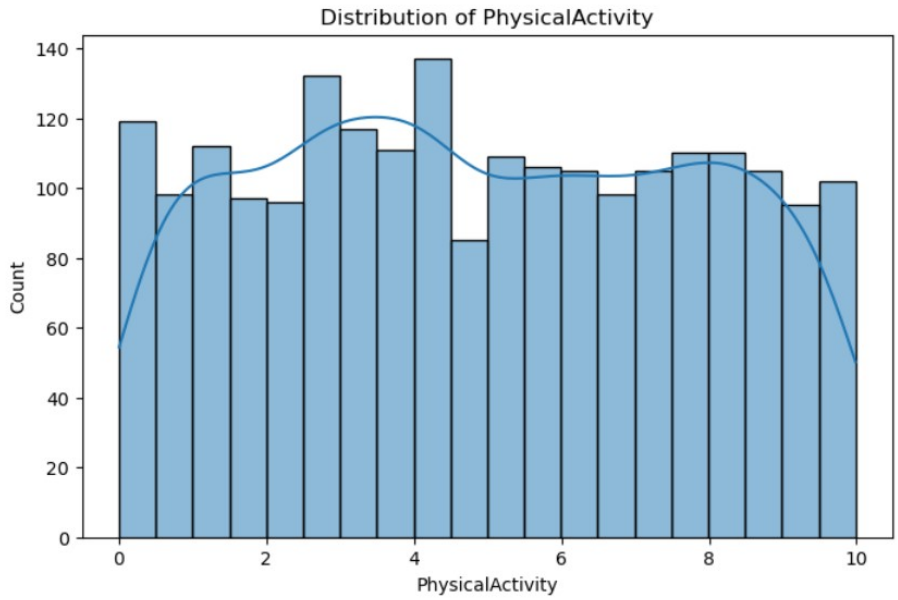


Figure 3-29 Physical Activity

The graph representing physical activity displayed a relatively even distribution, suggesting that the activity levels were uniformly spread among the participants.

Diet Quality: -

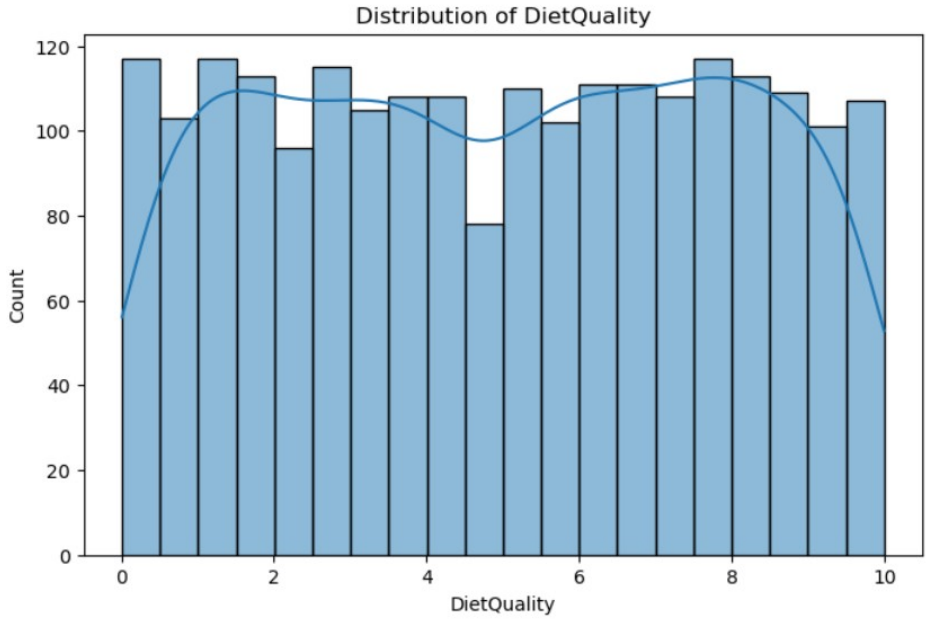


Figure 3-30 Diet Quality

The graph illustrating diet quality showed a relatively even distribution, indicating that scores were spread uniformly across the population.

Sleep Quality: -

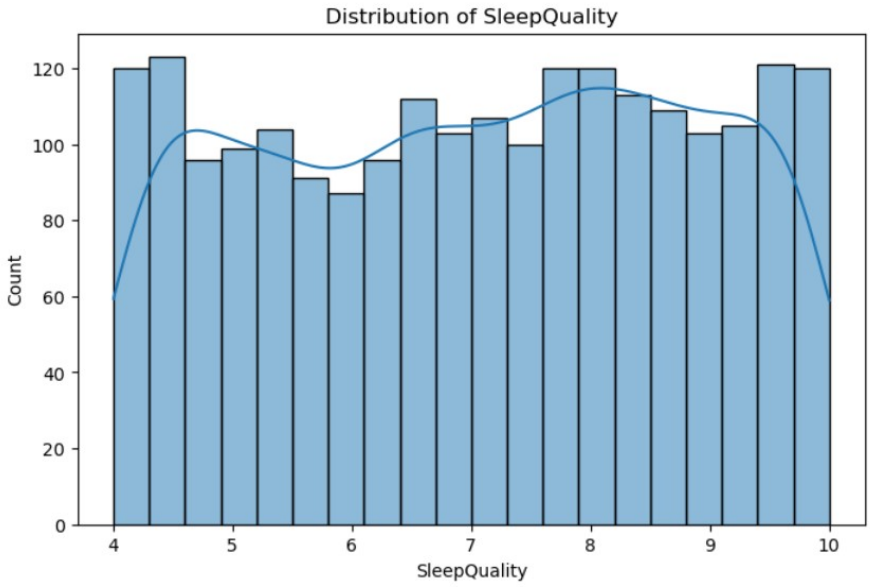


Figure 3-31 Sleep quality

The graph displaying sleep quality showed a fairly even distribution, suggesting that scores were consistently spread across the participants.

SystolicBP: -

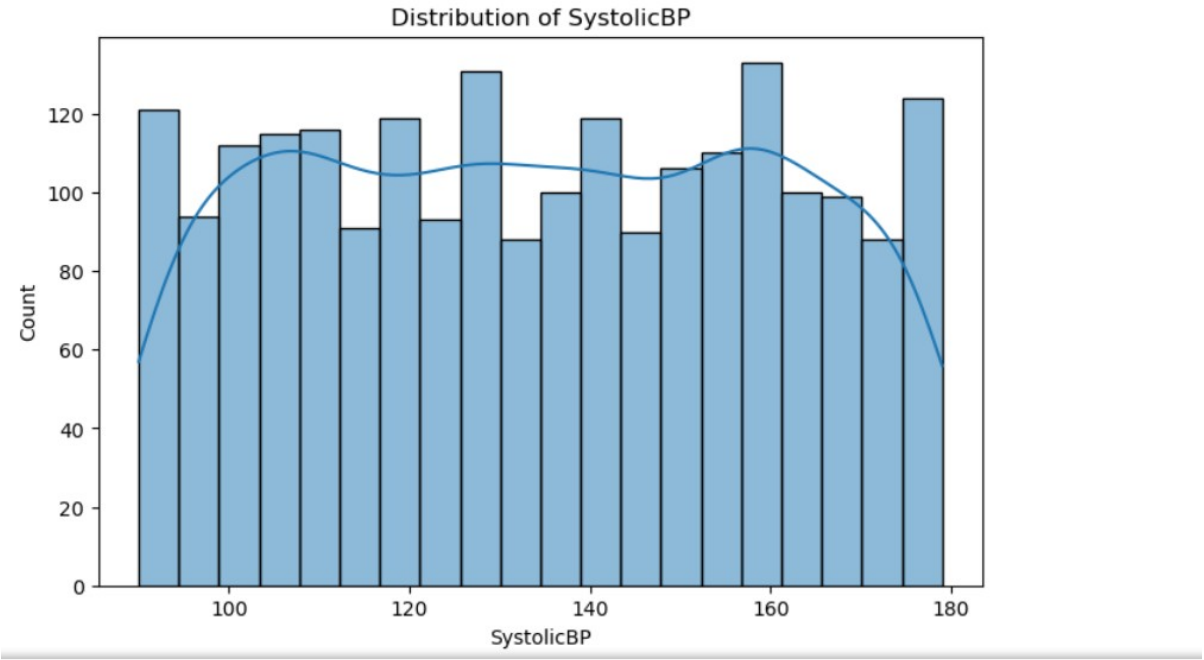


Figure 3-32 Systolic BP

The graph displaying SystolicBP showed a fairly even distribution, suggesting that scores were consistently spread across the participants

Diastolic BP:-

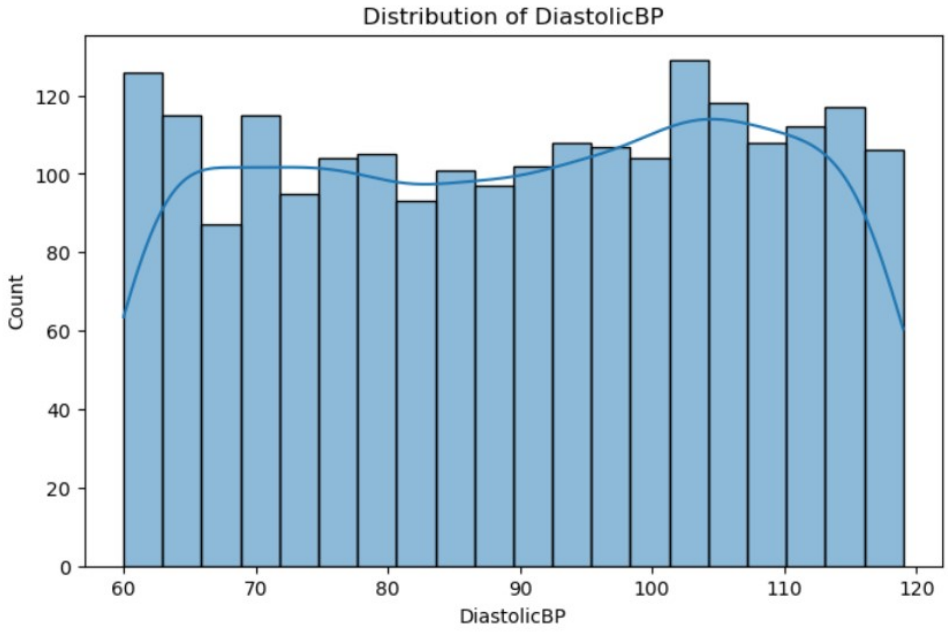


Figure 3-33 Diastolic BP

The graph displaying DiastolicBP showed a fairly even distribution, suggesting that scores were consistently spread across the participants

Cholesterol Total: -

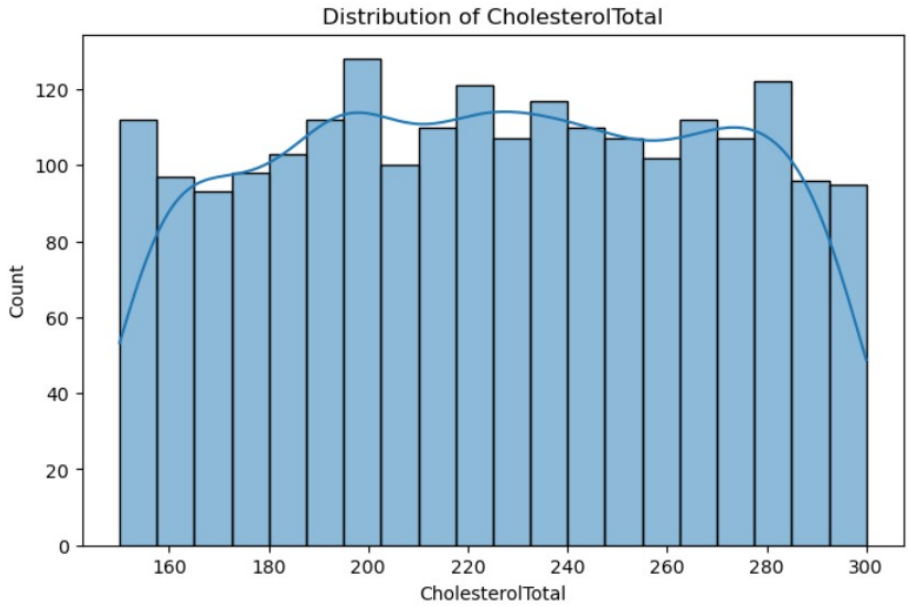


Figure 3-34 Cholesterol Total

The graph displaying CholesterolTotal showed a fairly even distribution, suggesting that scores were consistently spread across the participants

CholesterolHDL:-

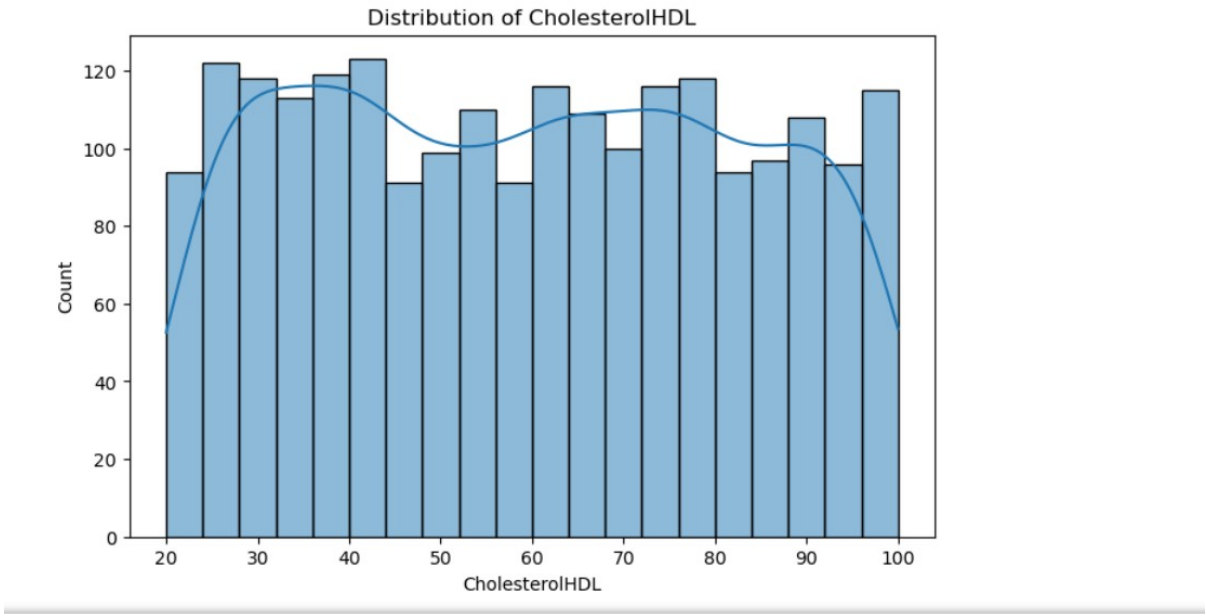


Figure 3-35 Cholesterol HDL

The graph displaying CholesterolHDL showed a fairly even distribution, suggesting that scores were consistently spread across the participants.

CholesterolTriglycerides: -

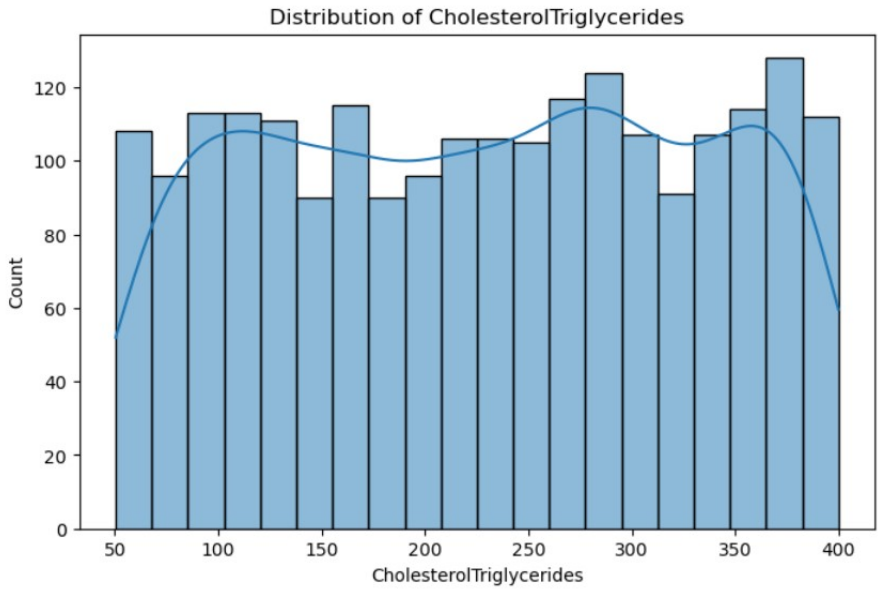


Figure 3-36 Cholesterol Triglycerides

The graph displaying **Cholesterol Triglycerides** showed a fairly even distribution, suggesting that scores were consistently spread across the participants.

MMSE: -

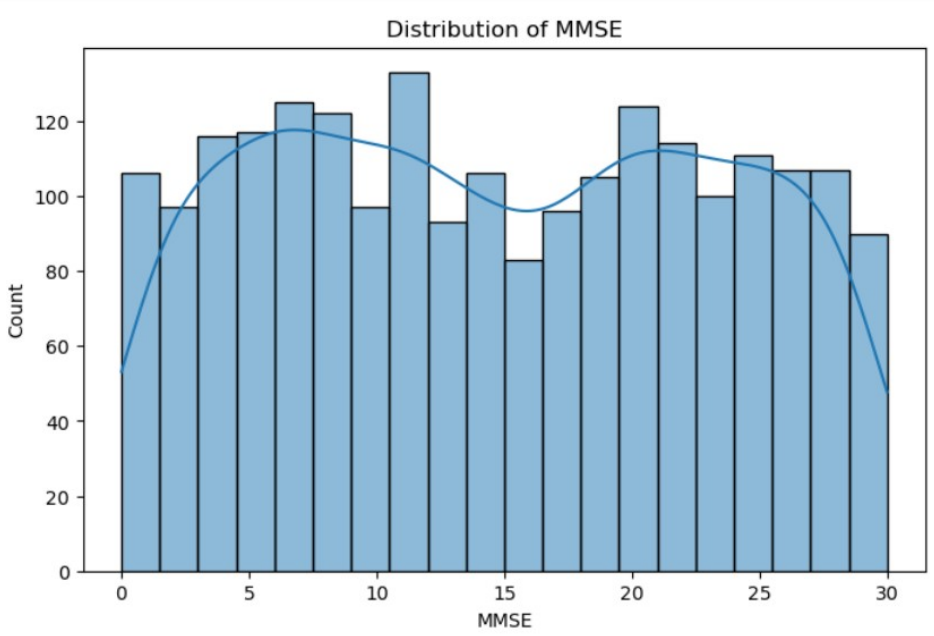


Figure 3-37 MMSE

The "MMSE" (Mini-Mental State Examination) distribution appears to be **Bimodal**, indicating two 2 distinct groups within the data.

Functional Assessment: -

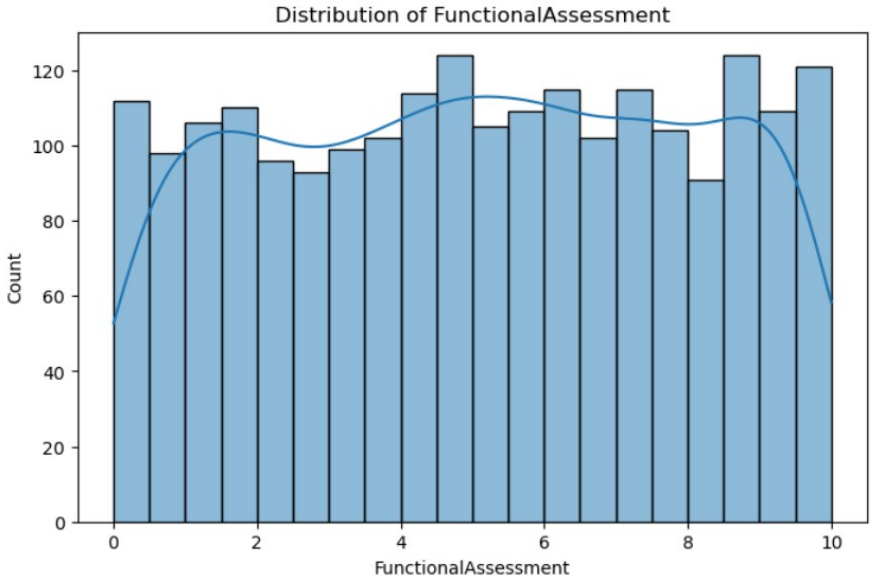


Figure 3-38 Functional Assessment

The graph displaying **function assessment** showed a fairly even distribution, suggesting that scores were consistently spread across the participants.

ADL: -

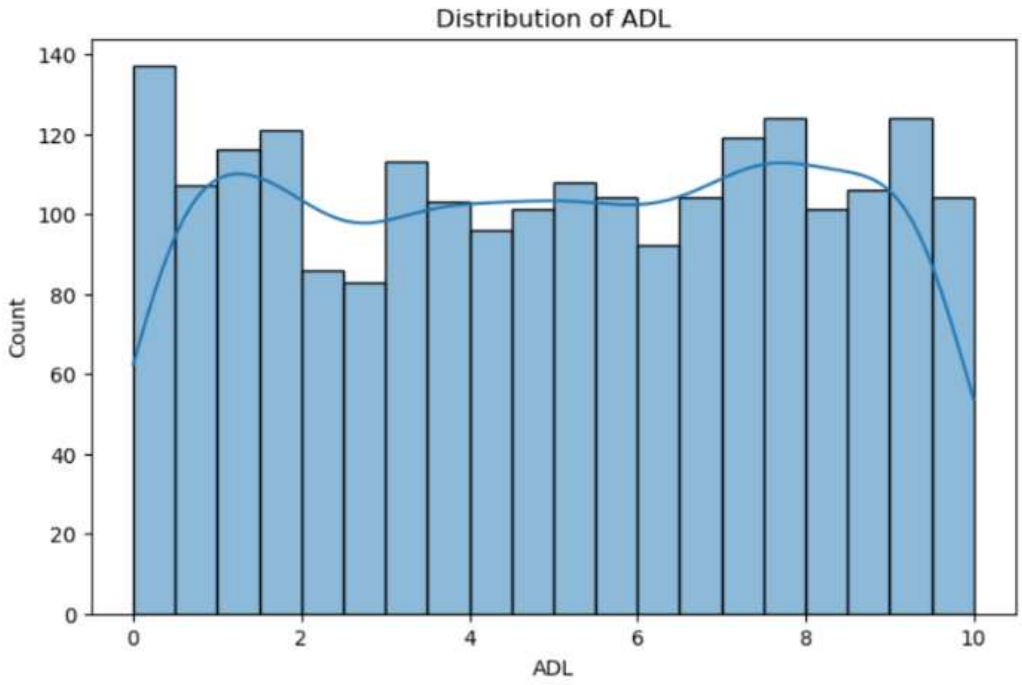


Figure 3-39 ADL

The graph displaying ADL showed a fairly even distribution, suggesting that scores were consistently spread across the participants.

Summary:-

- Most features show a **fairly uniform** distribution
- The "MMSE" (Mini-Mental State Examination) distribution appears to be **Bimodal**, indicating two 2 distinct groups within the data.

Distribution of the Target Variable: -

The target variable, "diagnoses," was analysed by plotting a pie chart to visualize the distribution of diagnoses. This will allow for the examination of the percentage of individuals diagnosed as positive versus negative, providing insights into the balance of the target classes.

```
[38]: categories=[0,1]
counts=df.Diagnosis.value_counts().tolist()
colors=sns.color_palette("muted")
plt.figure(figsize=(6,6))
plt.pie(counts,labels=categories,autopct='%1.1f%%',startangle=140,colors=colors)
plt.title('Diagnosis distribution')
plt.show()
```

As the target variable is binary, a category list was created in which the values 0 and 1 were stored. The "diagnosis" column was then selected from the dataset, and the value_counts method was applied to generate a frequency distribution. Subsequently, a pie chart was plotted to visualize the proportions of individuals diagnosed as positive and negative, as shown below

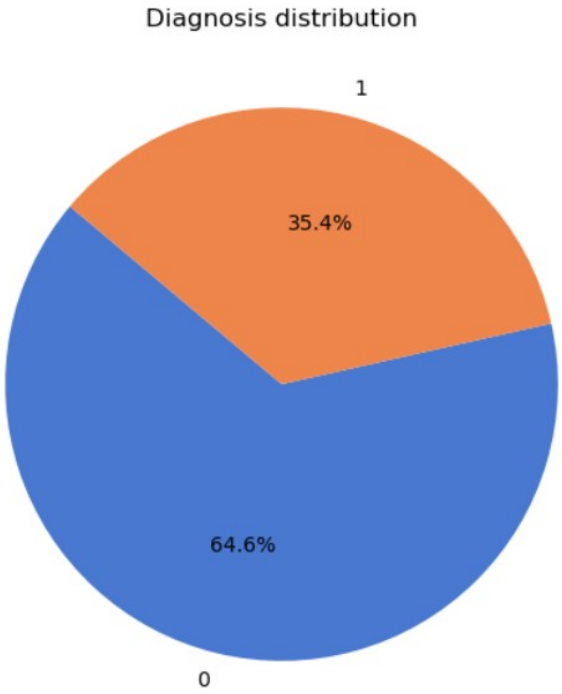


Figure 3-40 Diagnosis Distribution

From the above pie chart, it can be observed that the data set mostly consist of people who do not have Alzheimer disease, and we have also seen earlier in the categorical features part that most people have not Alzheimer disease. The target variable is moderately imbalance.

Relation of different variable: -

Here is the deep inside the data that what is the relation of different variable with the target variable(diagnoses.)

So here I explored the age, gender, ethnicity and education level to better understand the effect of Alzheimer disease.

```
[80]: colors=sns.color_palette('pastel')[0:5]
#as we are going to build pie chart so we plot 2 row and 2 column
#fig,axs=plt.subplots(2,2, figsize(6,6))
fig, axs = plt.subplots(2, 2, figsize=(6, 6))


#now pie chart for age
df['bins']=pd.cut(df['Age'],bins=[60,69,79,90],labels=["60-69","70-79","80-90"])

axs[0, 0].pie(df.groupby('bins').size(), labels=df.groupby('bins').size().index, colors=colors,autopct='%.0f%%', radius=0.8)

axs[0,0].set_title("age")
#drop bins
df.drop(['bins'],axis=1,inplace=True)

#now pie chart for gender
axs[0,1].pie(df['Gender'].value_counts(),labels=['Female','Male'],colors=colors,radius=0.8)
axs[0,1].set_title("gender")

#now pie chart for ethnicity
axs[1,0].pie(df['Ethnicity'].value_counts(),labels=['Caucasian', 'African-American', 'Other', 'Asian'],autopct='%.0f%%',colors=colors,radius=0.8)
axs[1,0].set_title("ethnicity")

#now pie chart for education level
axs[1, 1].pie(df['EducationLevel'].value_counts(), labels=['High School', "Bachelor's", 'None', 'Higher'], colors=colors, autopct='%.0f%%',radius=0.8)
axs[1, 1].set_title("Educational Level")
plt.show()
```

A 2x2 subplot was created to visualize the distribution of each independent variable. First, a bin column for "age" was generated, and a pie chart was plotted to display the distribution. Afterward, the bins were removed, and pie charts were plotted for the remaining variables, with each variable being assigned to its respective subplot.

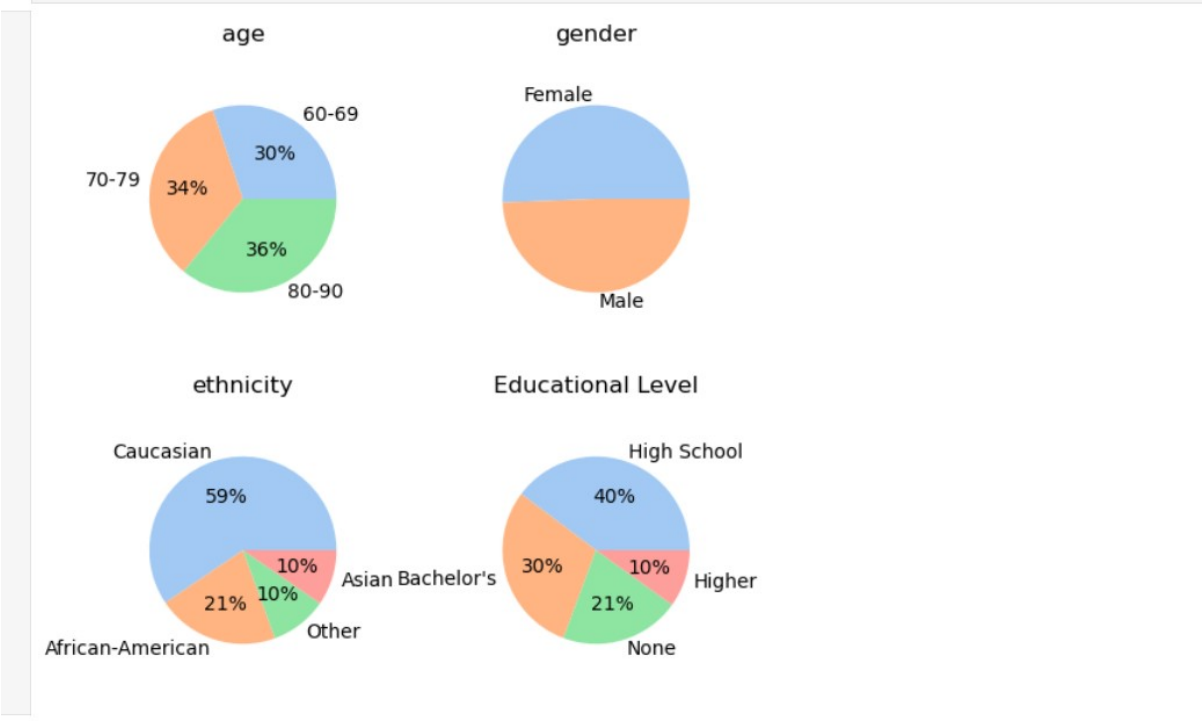


Figure 3-41 Relation of different variables

From above it can be observed that our previous assumptions are true.

- The highest disease occurs in people from age 80 to 90.
- This disease equally effect both men and women.
- Caucasian is the ethnicity which is highly affected by this disease.
- The highest education level is high school.

Correlation: -

Then i created a correlation matrix in between independent variable and dependent variable to check which variable contributes more to the target variable.

```
[87]: #creating a mask for the upper triangle
      mask=np.triu(np.ones_like(df.corr(),dtype=bool))

      #plot heatmap
      plt.figure(figsize=(12,10))
      sns.heatmap(df.corr(),cmap="Spectral",cbar_kws={"shrink":.5},mask=mask)
      plt.show()
      <Figure size 1200x1000 with 0 Axes>
```

As the dataset was very large so that's why i created a mask which will extract only the upper triangle of diagonal values while ignoring the lower triangle or false values. In this way i get the clear picture of correlation between different variables.

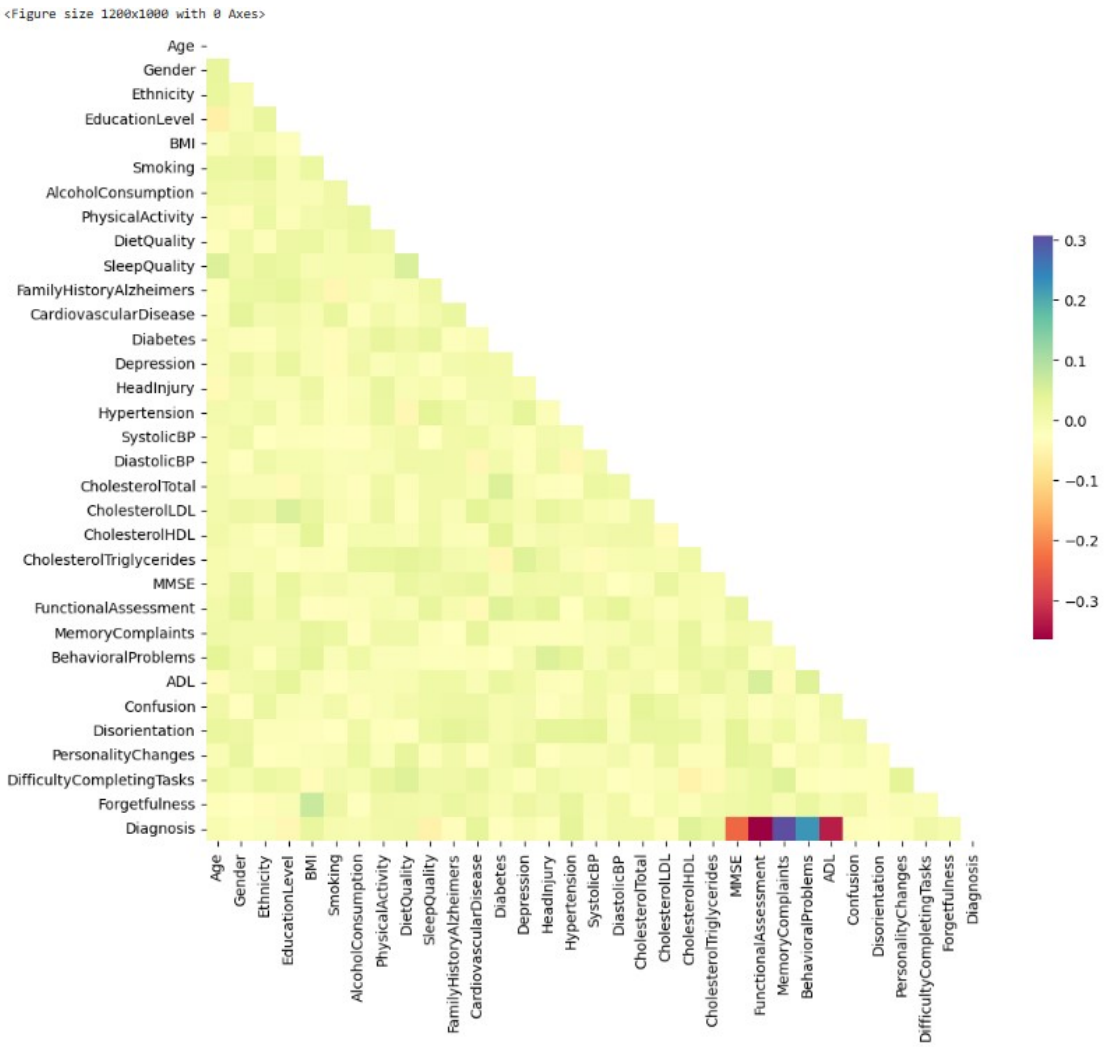


Figure 3-42 Correlation

From above Figure 3-43, we can see that on the right-hand side there is pre-define parameter. So, most of the blocks are yellow means that all the independent features have no correlation with each other. But when we see at the bottom line which is target variable, we can see that these 5 features have a correlation with the target variable. These features are MMSE, Functional Assessment, Memory Complains, Behavorial Problems, ADL. So, we can say that these variables are the most important for detecting the disease.

```
[97]: corr_matrix=df.corr()['Diagnosis']
heatcol=df.columns[corr_matrix.abs() > 0.1]
df[heatcol].corr()['Diagnosis'].sort_values().plot(kind='bar',color='blue')
plt.axhline(0, color="k")
plt.ylabel('Correlation to Diagnosis')
plt.xticks( rotation=30)
plt.show()
```

Then i take the correlation values which are greater than abs 0.1 and then i correlate them with each value of target column and then plot them using bars so that i can see which feature are positive correlated and which are negative correlated.

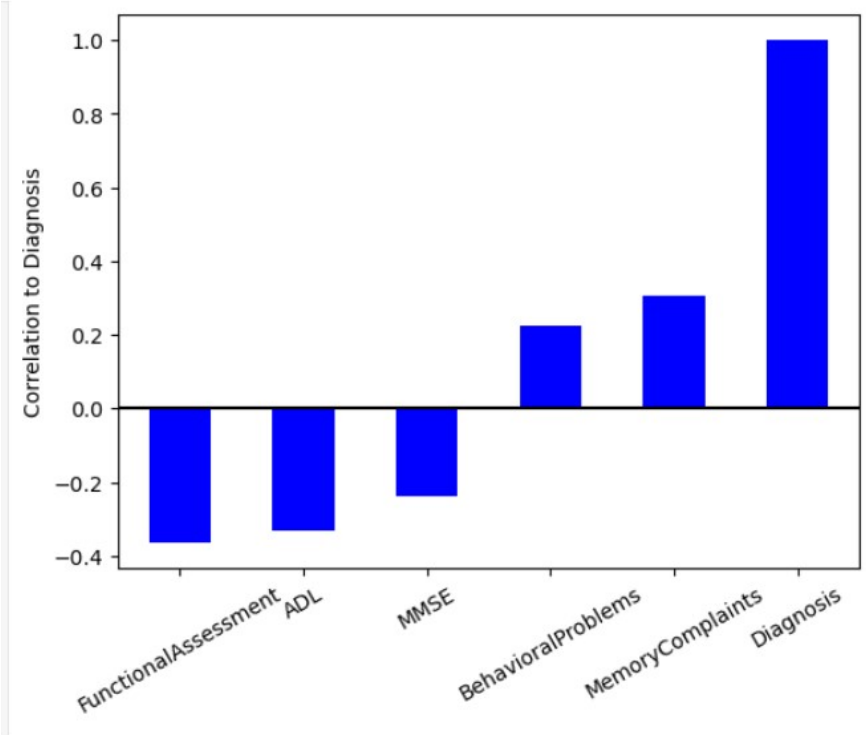


Figure 3 -44 Correlation to diagnosis

From above Figure 3 -45 ,we can see that 3 features (**Functional Assessment**, **ADL** (Activities of Daily Living), and **MMSE** (Mini-Mental State Examination)) are negatively correlated indicating that lower score of these variables will increase the likelihood of Alzheimer. While on the other hand 2 features (**Behavioral Problems** and **Memory Complaints**) are positively correlated indicating that higher value of these feature will increase the risk of Alzheimer.

3.1.4 Clinical Data Preparation and Preprocessing: -

Splitting the data: -

[17]:
```
#preprocessing


# Split data into features (X) and target (y)
X = df.drop(columns=['Diagnosis'])
y = df['Diagnosis']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

As part of the preprocessing pipeline, the dataset was first divided into input features and the target variable. The target variable, **Diagnosis**, which indicates the presence or absence of Alzheimer's disease, was separated from the feature set.

The data was partitioned into training and testing subsets with an 80/20 split, where 80% of the data was allocated for training and 20% for testing. To preserve the class distribution of the target variable in both subsets, stratified sampling was utilized. This approach ensured that the model was assessed on a representative sample. The random_state parameter was set to guarantee the consistency of the results.

Feature Importance:-

Now I am going to do feature importance of each feature in the dataset using the random forest.

```
[6]: from sklearn.ensemble import RandomForestClassifier
import pandas as pd
model=RandomForestClassifier(n_estimators=100)
model.fit(X_train,y_train)
importance=model.feature_importances_
feature_importance_df=pd.DataFrame(
    {
    'Feature':X_train.columns,
     'Importance':importance
    }
)
feature_importance_df=feature_importance_df.sort_values(by='Importance',ascending=False)
print(feature_importance_df)
```

In this code i use n_estimators=100 so that 100 trees take part in make the decision of random forest. Conventionally I use 100 if the results are not good i will increase the number. Then I fit the X_train and y_train data into the model, when the model is trained than I extract the feature importance and make a data frame in which features are extracted from X_train data set columns and importance score are extracted from model and then sort them in ascending order.

	Feature	Importance
23	FunctionalAssessment	0.179749
26	ADL	0.168546
22	MMSE	0.118821
24	MemoryComplaints	0.090866
25	BehavioralProblems	0.047997
8	DietQuality	0.030652
18	CholesterolTotal	0.029596
21	CholesterolTriglycerides	0.029573
7	PhysicalActivity	0.029107
4	BMI	0.028926
20	CholesterolHDL	0.028478
9	SleepQuality	0.028025
6	AlcoholConsumption	0.027473
19	CholesterolLDL	0.027044
16	SystolicBP	0.025323
17	DiastolicBP	0.025267
0	Age	0.022755
3	EducationLevel	0.012067
2	Ethnicity	0.007146
31	Forgetfulness	0.003936
13	Depression	0.003927
1	Gender	0.003496
30	DifficultyCompletingTasks	0.003461
15	Hypertension	0.003435
5	Smoking	0.003386
10	FamilyHistoryAlzheimers	0.003294
27	Confusion	0.003257
12	Diabetes	0.003224
11	CardiovascularDisease	0.003203
28	Disorientation	0.002978
29	PersonalityChanges	0.002553
14	HeadInjury	0.002439

Table 10 Feature Importance

Now for better understanding I make a bar plot of above results as

```
top_features=feature_importance_df.head(36)
plt.figure(figsize=(10,6))
sns.barplot(x='Importance',y='Feature',data=top_features)
plt.show()
```

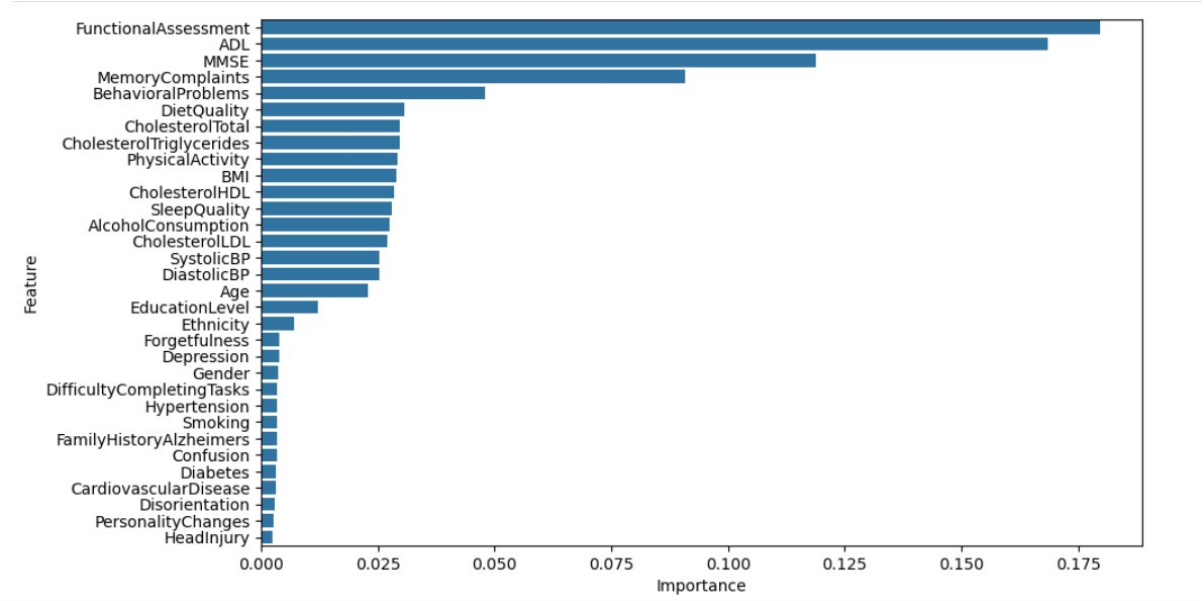


Figure 3-46 Feature Importance

from Figure 3-47, we can see the importance score for each feature.

Elbow Method: -

Then I use elbow method for the feature selection as a source of double verification

```
1]:
# Function to find the optimal value of k using the elbow method
def find_best_k_elbow(X_train, y_train, max_k):
    scores = []
    k_values = range(1, max_k + 1, 5)  # Testing k values in steps of 5 (adjust as needed)

    for k in k_values:
        selector = SelectKBest(f_classif, k=k)
        model = Pipeline([
            ('feature_selection', selector),
            ('classifier', RandomForestClassifier(random_state=42))  # You can change the classifier
        ])

        # Evaluate using cross-validation
        score = np.mean(cross_val_score(model, X_train, y_train, cv=5))
        scores.append(score)

    # Find the elbow point
    knee_locator = KneeLocator(k_values, scores, curve="convex", direction="decreasing")
    best_k = knee_locator.knee

    # Plot k vs. score with elbow point
    plt.plot(k_values, scores, marker='o', label='CV Score')
    if best_k:
        plt.axvline(x=best_k, color='r', linestyle='--', label=f'Elbow at k={best_k}')
    plt.xlabel('Number of Features (k)')
    plt.ylabel('Cross-Validation Score')
    plt.title('Elbow Method for Optimal k')
    plt.legend()
    plt.show()

    return best_k
```

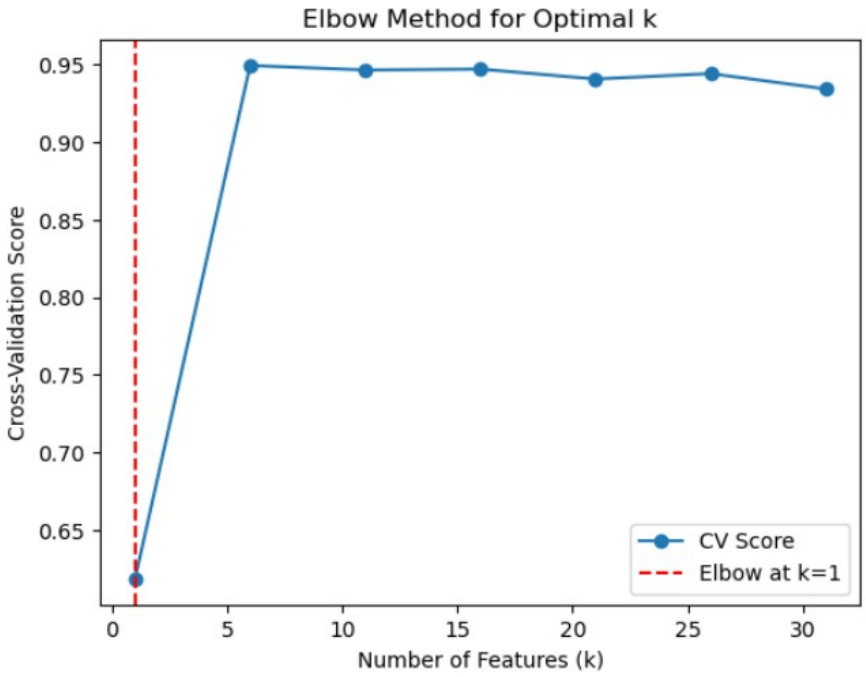


Figure 3-48 Number of best features

Figure 3-45 Number of best features also shows that there are 6 best features using the elbow method.

Recursive feature elimination with cross validation: -

```
[*]: from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import RFECV
rf=RandomForestClassifier(random_state=0,class_weight='balanced')
# Apply RFECV for feature selection

rfecv=RFECV(rf,cv=5,step=1,scoring='accuracy')
X_train_fs=rfecv.fit_transform(X_train,y_train)
X_test_fs=rfecv.transform(X_test)
```

Then I apply RFECV using random forest. Actually, random forest has inbuild features extraction method which is very robust and efficient for extracting the features. So, I use it, first i run the random forest classifier with random_state=0 so that it will give every time same results and with balance class weights. Then I apply RFECV using cross validation of 5 and step=1 mean in every run it excludes only one feature and scoring of features would be based on accuracy of features. Then implement these features on X_train and X_test.

Then I plot a line graph to see the number of features included and their accuracy.

```
[42]: plt.figure(figsize=(15, 6))
plt.title('Number of Features Included vs Accuracy')
plt.xlabel('Number of Features Included')
plt.ylabel('Model Accuracy')

# Getting the number of features selected at each stage
num_features = np.arange(1, len(rfecv.cv_results_['mean_test_score']) + 1)

plt.plot(num_features, rfecv.cv_results_['mean_test_score'])
plt.show()
```

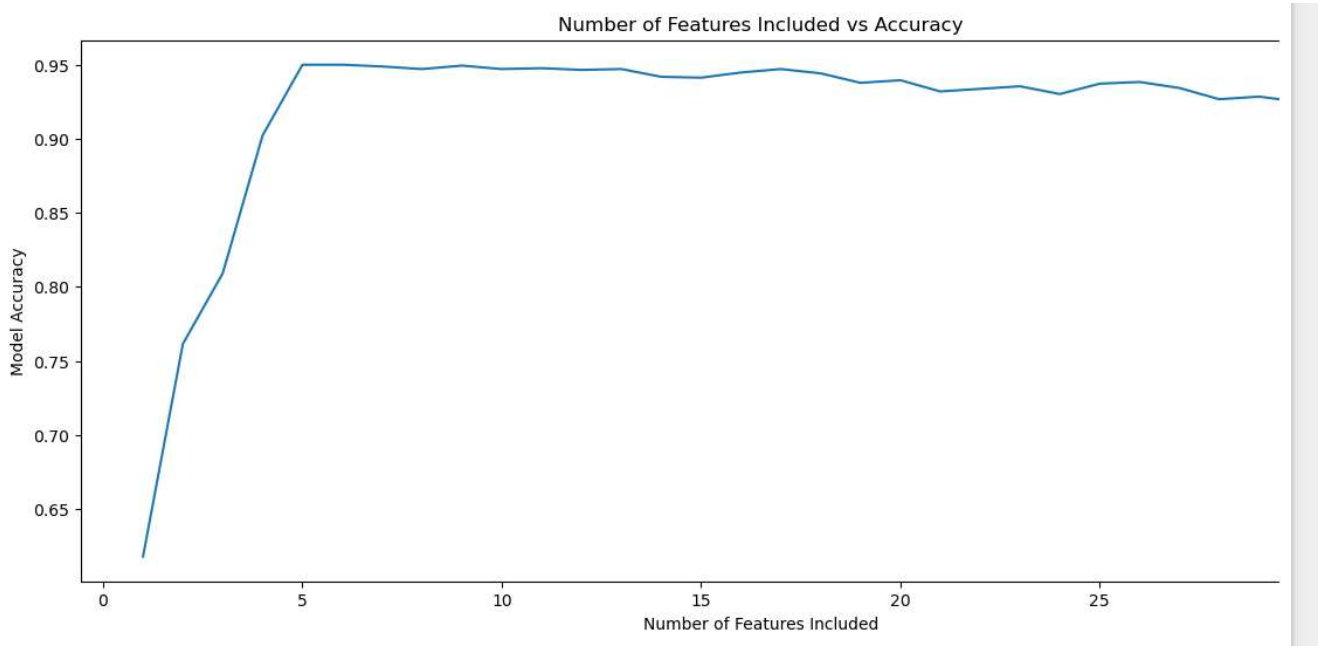


Figure 3-49 Number of Features Included

From Figure 3-46 Number of Features Included, we can see that only top 6 features are contributing to the 95% accuarcy of the model. Its mean it is better for us to include only 6 features to avoid overfitting and gain greater accuracy.

Then I print the top selected features as

```
[35]: # Print selected features
selected_feature_names = X_train.columns[rfecv.support_]
print(f"Selected Features: {selected_feature_names}")

Selected Features: Index(['DietQuality', 'MMSE', 'FunctionalAssessment', 'MemoryComplaints',
       'BehavioralProblems', 'ADL'],
      dtype='object')
```

Figure 3.47 Top selected Features

The top selected features are Diet Quality, MMSE, functional Assessment, Memory complaints, behavioural Problems.

Feature Scaling: -

Based on the exploratory data analysis (EDA), it was observed that the distributions of the features are approximately normal. Consequently, **StandardScaler** was applied to the features selected by **RFECV** to ensure the data is standardized. Standardization is particularly useful in maintaining equal scaling across features, which is important for many machine learning algorithms. The scaler was **fit on the training data** and then used to **transform both the training and test datasets**, preserving the integrity of the test set and preventing data leakage.

```
[32]: #Standard scaler
from sklearn.preprocessing import StandardScaler

# Initialize the scaler
scaler = StandardScaler()

# Fit the scaler on the selected features of the training data
X_train_fs = scaler.fit_transform(X_train_fs)

# Transform the selected features of the test data using the same scaler
X_test_fs= scaler.transform(X_test_fs)
```

```
[126]: import matplotlib.pyplot as plt

# Plot the distribution after StandardScaler for each feature
for i, feature in enumerate(numerical_variables):

    # After standardization
    plt.hist(df[feature], bins=30, color='lightgreen', edgecolor='black')
    plt.title(f"After Standardization: {feature}")
    plt.xlabel('Scaled Value')
    plt.ylabel('Frequency')
    plt.show()  # Display each plot one by one
```

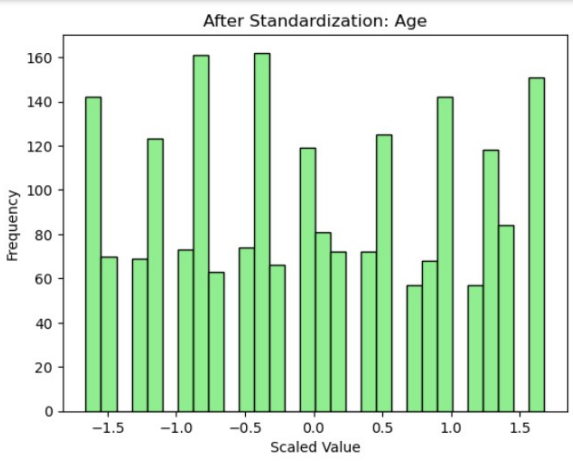


Figure 3-50 Feature Scaling

From above it can be seen that the transformation is applied to all the numerical features.

Class Imbalance: -

As the target variable which is diagnosis is moderately imbalance with the ratio of 65% patients with no Alzheimer disease while 35% patients with Alzheimer disease. So, if we train the model without data handling the class imbalance than obviously the model would be biased towards the majority class. Thats why to handle this issue I am going to apply SMOTE to only training data set while test and validation data set remains the same.

```
[36]:
# Apply SMOTE to handle class imbalance
smote = SMOTE(random_state=42)
X_train_fs, y_train = smote.fit_resample(X_train_fs, y_train)
```

Here I apply 42 as random_state so that every time it gives me same answer and fit this to on X_train and y_train, mean only to training data set.

```
[37]: print("Class distribution after SMOTE:")
print(pd.Series(y_train).value_counts())

Class distribution after SMOTE:
Diagnosis
0    1111
1    1111
Name: count, dtype: int64
```

Figure 3.48 Class balance

From Figure 3.48 Class balance, we can see that now the training dataset is completely balanced.

Summary: -

- The dataset was split into train and test with the ratio of 80% and 20%.
- Feature importance was carried out using inbuild function of random forest.
- Top features were extracted using RFCV and elbow method.
- Standard scaler was applied on dataset to standardize the features between 0 and 1.
- SMOTE was applied to handle class imbalance.

3.2 MRI Scan Dataset: -

Magnetic Resonance Imaging (MRI) is a medical imaging technique which is used to visualize the internal structures of body, including the brain. The main use of MRI is to diagnose the brain disorder and abnormalities.

So in this part of the project we aim to use these MRI images data to train a machine learning model which can predict whether a person has Alzheimer or not. If yes than it will also tells at which stage the disease is.
So we are going to build a CNN learning model which is capable of predicting brain health status based on these MRI data so that it will assist clinicians in diagnosing a patient

This dataset comprises MRI brain scans categorized into 4 classes based on the progression of dementia.

This dataset was the augmentation of original dataset.

Mild Dementia:-

This subcategory includes the MRI pictures which represents mild dementia. These are total 8960 in count.

Moderate Dementia: -

This subcategory includes the MRI pictures which represents moderate dementia. These are total 6464 in count.

Non Dementia: -

This subcategory includes the MRI pictures which represents non dementia. These are total 9600 in count.

Very Mild Dementia: -

This subcategory includes the MRI pictures which represents very mild dementia. These are total 8960 in count.

Dataset Composition: -

The dataset is divided into two main directories:

Original Images: These images represent the raw MRI images as collected in the original dataset.

Augmented Images: These are the synthetically generated images derived from the originals using image augmentation techniques.

In total the data set comprises 40,000 images

Important Note: -

For this project only the augmented image dataset was used, and original dataset was not used. But original dataset was retained for potential use in future validation studies.

Key aspects:

- Data type: Unstructured (image format - PNG/JPG)
- Source: Kaggle – Alzheimer's MRI Classification dataset(MRI Classification)

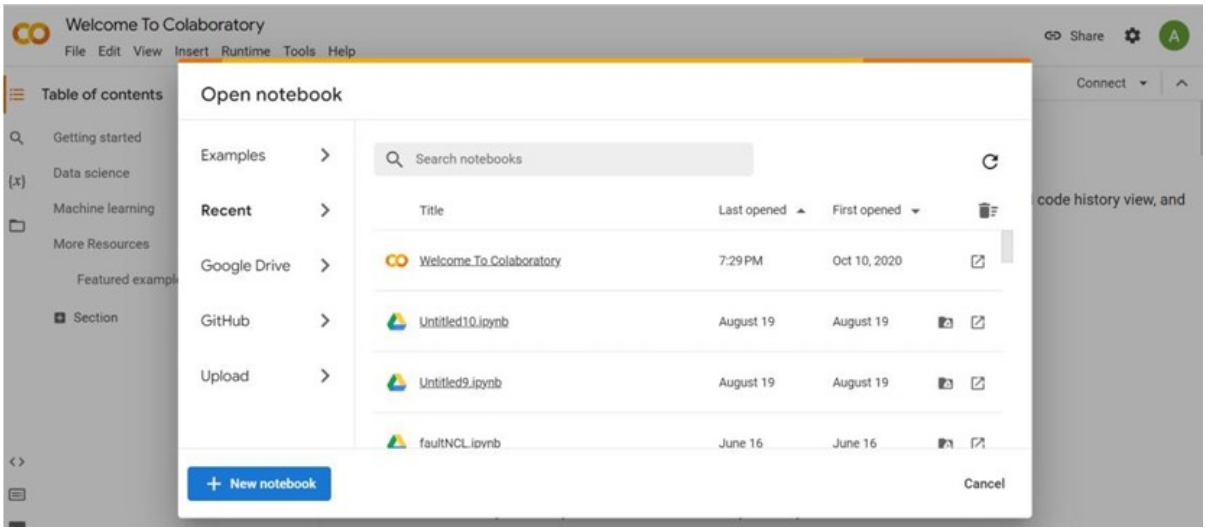
- Target label: Dementia stage (multiclass classification)

3.2.1 Initial Exploration of Dataset: -

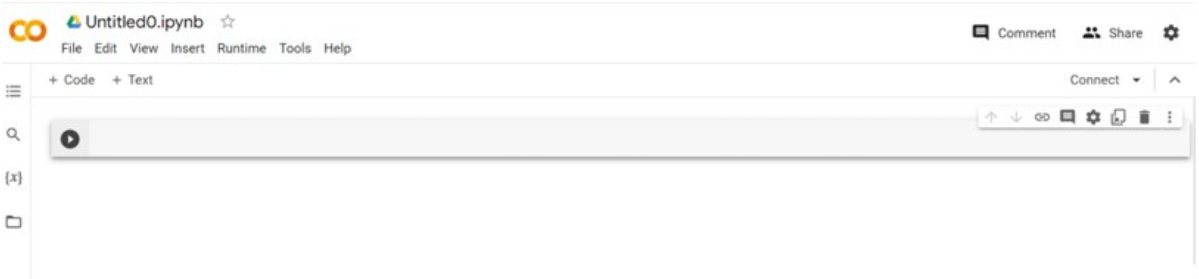
Creating the Notebook: -

First I downloaded the data set from MRI dataset and make it a zip file.

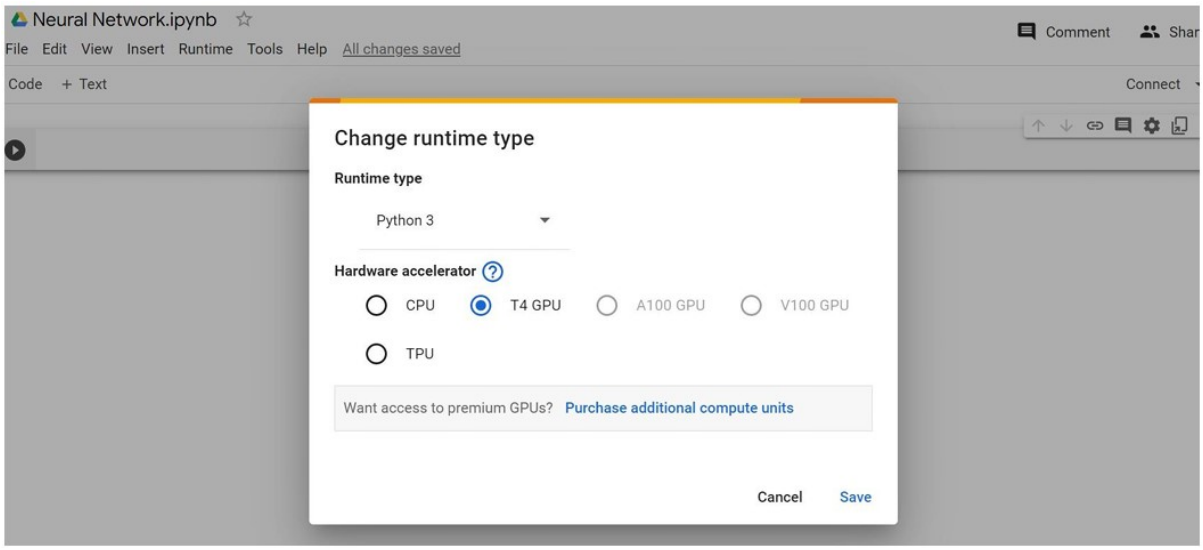
Then i open the google collab as.



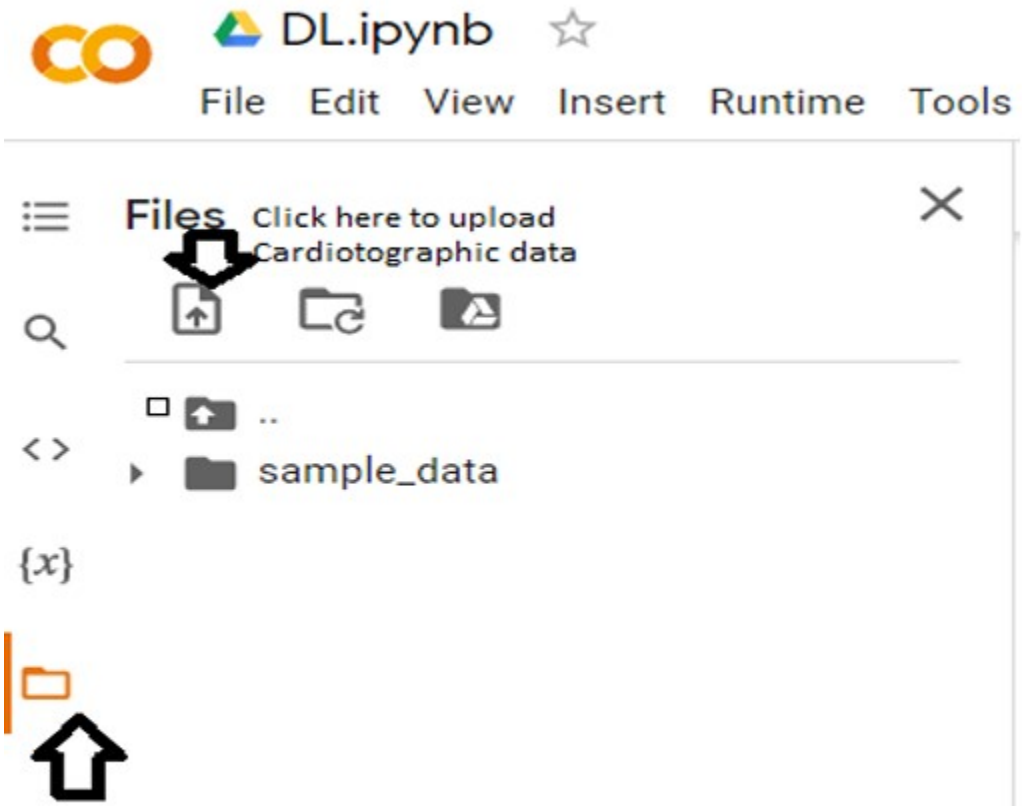
Than i click on new notebook



Than i click on runtime and select change runtime type, by setting the Hardware accelerator to T4 GPU and selecting Save



Than i upload the data to the collab by clicking on the file icon to open the ‘file explorer’ and then clicking on File upload icon at the top of this pane .


DL.ipynb
File Edit View Insert Runtime Tools
Files
Click here to upload
Cardiotographic data
..
sample_data

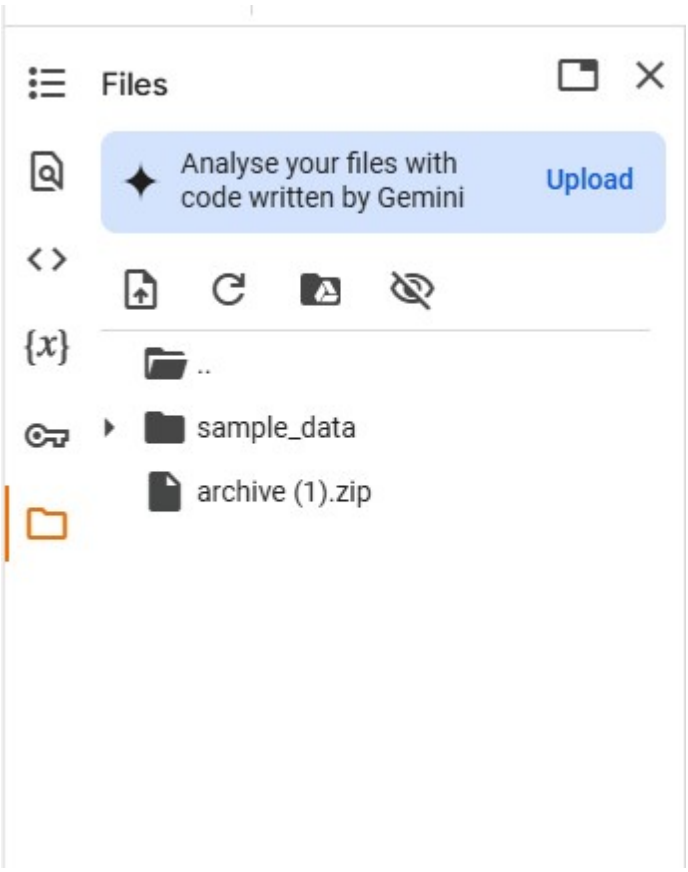


Figure 3.2.1

In figure 3.2.1, it is clear that the file with the name archive1.zip is stored there.

Then we imported some of the necessary libraries that were used in this project as follows

```
[ ] import tensorflow as tf
    import matplotlib.pyplot as plt
    import numpy as np
    import os
    import tensorflow as tf
    from tensorflow import keras
    from keras import layers
    import matplotlib.image as img
    %matplotlib inline
    import pandas as pd
    import numpy as np
    import keras
    import warnings
    warnings.filterwarnings(action="ignore")
    import matplotlib.pyplot as plt
    %matplotlib inline
    import seaborn as sns

    from keras.models import Sequential
    from keras.layers import Dense, Dropout, Flatten
    from keras.layers import Conv2D, MaxPooling2D
    from keras import backend as K

    from sklearn.preprocessing import StandardScaler
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import classification_report
    import tensorflow as tf
    from tensorflow import keras
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
    from tensorflow.keras.applications.vgg19 import VGG19
    from tensorflow.keras.optimizers import Adam
    from tensorflow.keras.losses import SparseCategoricalCrossentropy
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    from tensorflow.keras.callbacks import TensorBoard,EarlyStopping

    import sklearn.metrics as metrics
    from keras.callbacks import LearningRateScheduler
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    annealer = LearningRateScheduler(lambda x: 1e-3 * 0.95 ** x, verbose=0)
```

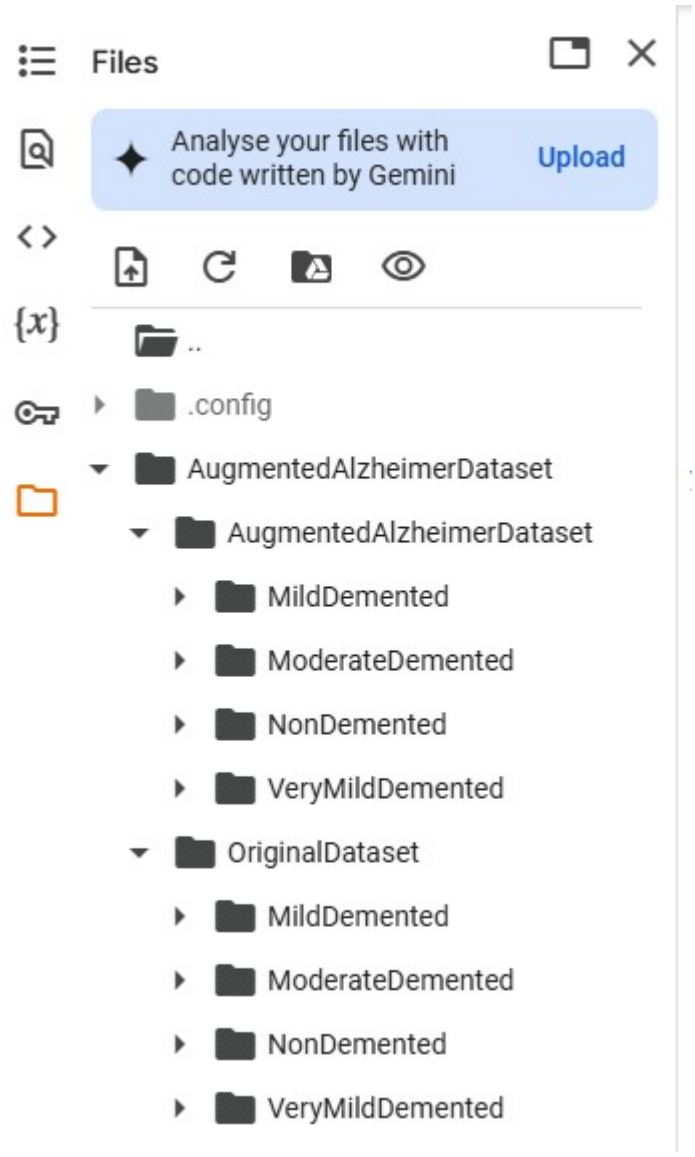
Figure 3.57 libraries

Than i define the path of that zip file and define a path where i will extract the content of that zip file. After that i use the zipfile command to unzip the zip folder and extract the data into the directory name AugmentedAlzheimerDataset

```
import zipfile
import os
# Path to the uploaded ZIP file
zip_path = "/content/archive (1).zip"

# Path to extract the dataset
extract_path = "/content/AugmentedAlzheimerDataset"



# Extract the ZIP file
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

print(f"Extracted to: {extract_path}")
```

```
Extracted to: /content/AugmentedAlzheimerDataset
```

From here we can see that the data is now extracted safely in the folder name AugmentedAlzheimerDataset.



Now iam going to classify each category of alzehmier disease as

```python
import os
import pandas as pd

# Set the paths to the extracted dataset
MildDemented_dir = r'/content/AugmentedAlzheimerDataset/AugmentedAlzheimerDataset/MildDemented'
ModerateDemented_dir = r'/content/AugmentedAlzheimerDataset/AugmentedAlzheimerDataset/ModerateDemented'
NonDemented_dir = r'/content/AugmentedAlzheimerDataset/AugmentedAlzheimerDataset/NonDemented'
VeryMildDemented_dir = r'/content/AugmentedAlzheimerDataset/AugmentedAlzheimerDataset/VeryMildDemented'


#Now assigning paths to each image
filepaths=[]
labels=[]
dict_list=[MildDemented_dir , ModerateDemented_dir , NonDemented_dir , VeryMildDemented_dir]
class_labels=['mild demented' , 'moderate demented' , 'non demented' , 'very mild demented']

for i, j in enumerate(dict_list):
  flist=os.listdir(j)
  for f in flist:
    fpath=os.path.join(j,f)
    filepaths.append(fpath)
    labels.append(class_labels[i])


#now concatenate both filepath of each image and their labels
Fseries=pd.Series(filepaths,name='filepaths')
Lseries=pd.Series(labels,name='labels')
Alzheimer_data=pd.concat([Fseries,Lseries],axis=1)
Alzheimer_df=pd.DataFrame(Alzheimer_data)#

#display first few rows
print(Alzheimer_df.head())
```

```
                                   filepaths         labels
0  /content/AugmentedAlzheimerDataset/AugmentedAl...  mild demented
1  /content/AugmentedAlzheimerDataset/AugmentedAl...  mild demented
2  /content/AugmentedAlzheimerDataset/AugmentedAl...  mild demented
3  /content/AugmentedAlzheimerDataset/AugmentedAl...  mild demented
4  /content/AugmentedAlzheimerDataset/AugmentedAl...  mild demented
```

Now in the above code first of all i imported pandas library and os library. Then i create directories which corresponds to each stage of Alzheimer disease by setting its directory path. Then I create empty lists with the name filepaths and labels which will be used for storing the file path and human readable labels. Then I create a directory list with the name of dict_list and create the class labels with the name of class_labels.

Then i create a for loop which is used to get the full file path and their corresponding labels. Then i convert them into 2 pandas' series with name of F series and L series and then concatenate them using dataframe to the final structure name Alzheimer_data and displays head of that data frame.

Shape: -

Then i view the shape of dataframe (Alzheimer_data) as

```python
Alzheimer_df.shape
```

```
(33984, 2)
```

Figure 3-51 Shape of MRI data

- In above Figure 3-51 shows that Alzheimer_data have 33984 rows and each row represents an image.
- And there are 2 columns which is filepath and labels.

Image shapes: -
This code is used to create empty list which will be latter used for storing length and width of images. Then I created a for loop which will iterate through first 33984 images (which is total number of images), load each image from its file path and extract only its length and width and append it to image_size. Then i convert image_sizes into a NumPy array for easier analyses and extract the unique shapes.

```
image_sizes = []

# Loop through all 33984 images and store their shape
for img_path in Alzheimer_df.filepaths[:33984]:
    img = plt.imread(img_path)
    image_sizes.append(img.shape[:2])  # Save (height, width)

# Convert to NumPy array
image_sizes = np.array(image_sizes)

# Get unique image shapes
unique_shapes = np.unique(image_sizes, axis=0)
print(f"Unique Image Shapes in Dataset: {unique_shapes}")
```

```
Unique Image Shapes in Dataset: [[180 180]
 [190 200]]
```

Figure 3-52 Unique Image Shapes

The preliminary analysis of the MRI dataset revealed inconsistencies in image resolutions. Such variation can lead to challenges in model training and evaluation, including reduced performance and potential processing errors if not properly addressed.

Inconsistent image dimensions must be standardized to ensure compatibility with the input layer of the deep learning models. Resizing images to a consistent resolution is a critical preprocessing step. This process, however, involves trade-offs:

- **Higher resolutions** preserve more image details, which can enhance model accuracy, but they significantly increase computational cost, memory usage, and training time.
- **Lower resolutions** improve processing speed and reduce resource demands, but may result in the loss of important diagnostic features.

To balance these considerations, the resizing of images will be carried out *after* the dataset is split into training, validation, and test subsets. This strategy ensures that the original data remains unchanged, allowing for flexibility in experimentation and maintaining the integrity of the dataset. For the initial stage of analysis, the dataset will be used in its original form.

Statistical description of data: -

To analyze the statistical properties of the dataset, I have written a script that generates a statistical summary, identifies any missing values, and provides a detailed description of the data.

```
# Check the first few rows of the dataset
print(Alzheimer_df.head())

# Get dataset information (column names, non-null values, data types)
print(Alzheimer_df.info())

# Check for missing values
print("\nMissing Values:\n", Alzheimer_df.isnull().sum())

# Get dataset summary (numerical stats if applicable)
print("\nDataset Summary:\n", Alzheimer_df.describe())
```

Output: -

```
                                          filepaths         labels
0  /content/AugmentedAlzheimerDataset/AugmentedAl...  mild demented
1  /content/AugmentedAlzheimerDataset/AugmentedAl...  mild demented
2  /content/AugmentedAlzheimerDataset/AugmentedAl...  mild demented
3  /content/AugmentedAlzheimerDataset/AugmentedAl...  mild demented
4  /content/AugmentedAlzheimerDataset/AugmentedAl...  mild demented
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 33984 entries, 0 to 33983
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   filepaths  33984 non-null  object
 1   labels     33984 non-null  object
dtypes: object(2)
memory usage: 531.1+ KB
None

Missing Values:
 filepaths    0
labels       0
dtype: int64

Dataset Summary:
                                                 filepaths        labels
count                                                33984         33984
unique                                               33984             4
top     /content/AugmentedAlzheimerDataset/AugmentedAl...  non demented
freq                                                     1          9600
```

Figure 3-53 Statistics of MRI data

In above it shows the head of dataframe which is fine. Then it shows there is no missing values in filepaths and labels and at the end it describes the data that it has 33984 images and labels which is also fine.

Duplicate images: -

I write a code to see is there any duplicate image in the data set as

```
[ ]  # Check for duplicate file paths
     duplicates = Alzheimer_df['filepaths'].duplicated().sum()
     print(f"Number of Duplicate Images: {duplicates}")

     Number of Duplicate Images: 0
```

Figure 3-54 Number of duplicate images

From above Figure 3-54, we can see that the dataset does not have any duplicate images.

Images in each category: -

Here are the number of images in each category of Alzheimer disease.

```
#display count of images in each category
print(Alzheimer_df['labels'].value_counts())

labels
non demented          9600
mild demented         8960
very mild demented    8960
moderate demented     6464
Name: count, dtype: int64
```

Figure 3-55 Number of Images in each category

3.2.2 EDA and Preprocessing of MRI dataset: -

Distribution of different Alzheimer's disease categories: -

This code visualizes the distribution of different Alzehmier's disease categories in the dataset using the pie chart as

```
import matplotlib.pyplot as plt

# Count the number of images in each category
label_counts = Alzheimer_df['labels'].value_counts()

# Define labels and sizes for the pie chart
labels = label_counts.index.tolist()  # Get the category names
sizes = label_counts.values.tolist()  # Get the count of images in each category

# Define colors for the pie chart
colors = ['#66b3ff', '#99ff99', '#ffcc99', '#ff9999']

# Plot the pie chart
plt.figure(figsize=(8, 8))  # Set the size of the pie chart
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=90, shadow=True)

# Add a title
plt.title('Distribution of Alzheimer\'s Dataset Categories', fontsize=16)

# Display the pie chart
plt.show()
```

Output:-

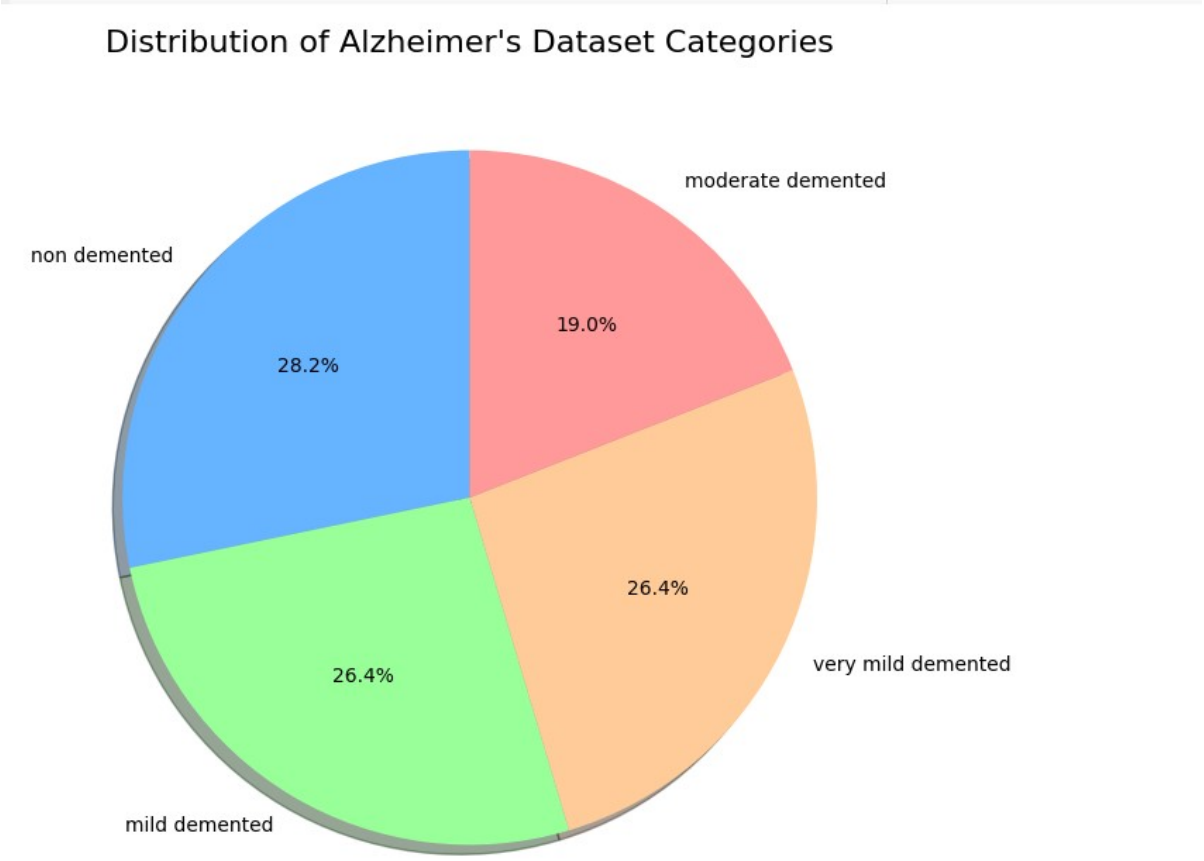


Figure 3-56 Distribution of dataset

This pie chart in Figure 3-56 shows the distribution of images in each category of Alzehmier disease. Like the dataset have 28% images which is showing non demented and 26% images showing mild demented etc.

Displaying images: -

This code displays 16 random images from the dataset along with their labels.

```
import numpy as np
import matplotlib.pyplot as plt

#display 16 pictures of data
random_index=np.random.randint(0,len(Alzheimer_df),16)
fig,axes=plt.subplots(nrows=4,ncols=4,figsize=(10,10),
                      subplot_kw={'xticks':[],'yticks':[]})

for i,ax in enumerate(axes.flat):
  ax.imshow(plt.imread(Alzheimer_df.filepaths[random_index[i]]))
  ax.set_title(Alzheimer_df.labels[random_index[i]])
plt.tight_layout()
plt.show()
```

Every time you run the code you will get different images.

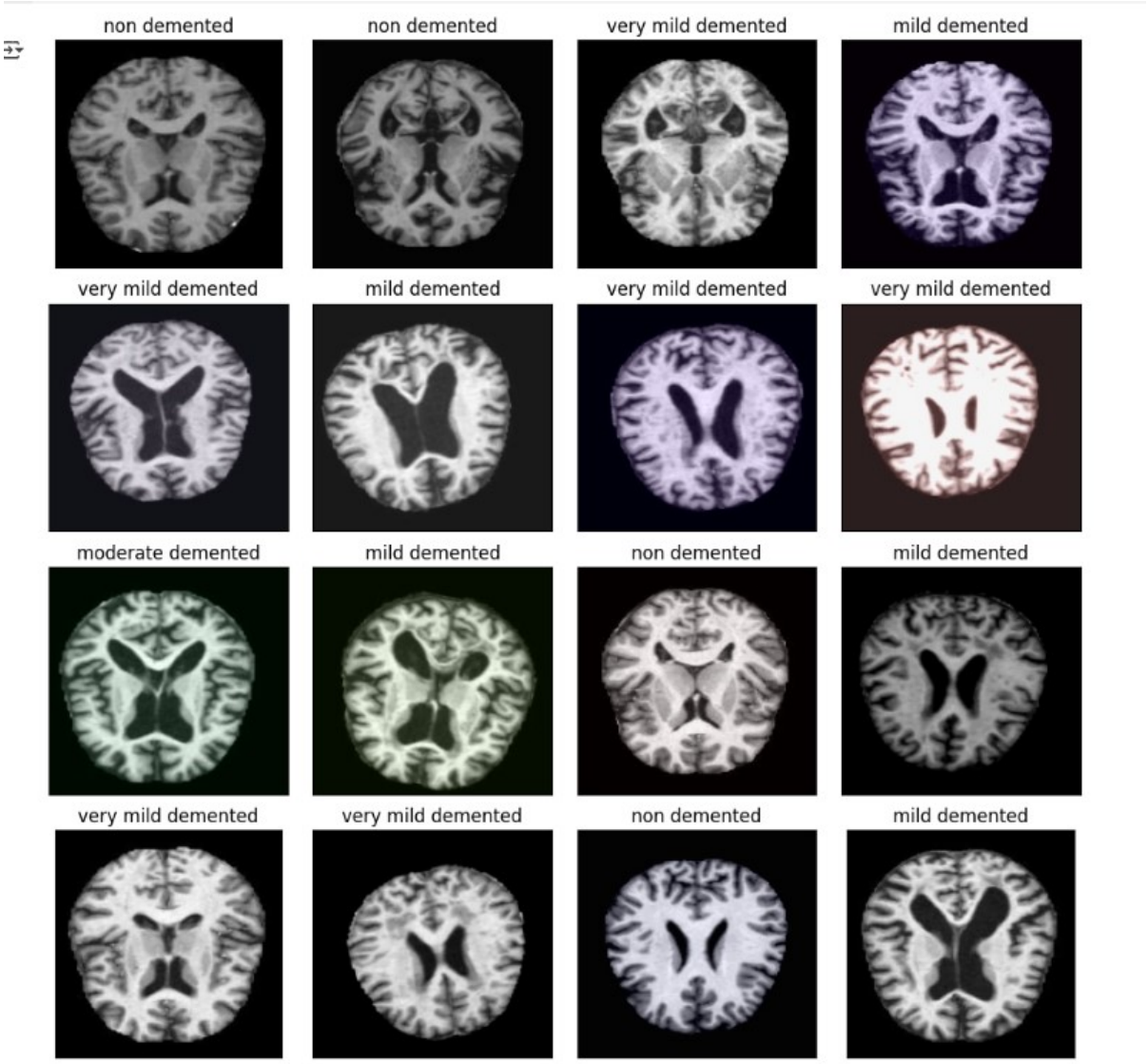


Figure 3-57 Display MRI Images

Splitting data: -

First of all, I installed a python library split-folder which is used to split the data set of images into train, test and validation. I installed it because my dataset is stored in folders (like each category has its own folder), so split-folder library will automatically divide the data set into train, test and validation without manually copying files.

```
pip install split-folders
```

```
Collecting split-folders
  Downloading split_folders-0.5.1-py3-none-any.whl.metadata (6.2 kB)
Downloading split_folders-0.5.1-py3-none-any.whl (8.4 kB)
Installing collected packages: split-folders
Successfully installed split-folders-0.5.1
```

Now I am going to split the data set into 2 portions i.e. training and testing. I assign 30% of data for testing and 70% of data for training. Then I split training data set further into 2 parts i.e. training 80% and validation 20%. Then i print the number of images ready for training, validation and testing.

```
[ ] from sklearn.model_selection import train_test_split

    # Step 1: Split into train (70%) and test (30%)
    train_images, test_images = train_test_split(Alzheimer_df, test_size=0.3, random_state=42)

    # Step 2: Split the training set further into train (80%) and validation (20%)
    train_set, val_set = train_test_split(train_images, test_size=0.2, random_state=42)

    print(f"Train set: {len(train_set)} images")
    print(f"Validation set: {len(val_set)} images")
    print(f"Test set: {len(test_images)} images")
```

```
Train set: 19030 images
Validation set: 4758 images
Test set: 10196 images
```

Shapes of Images: -

Then i print the shapes of images which are ready for training, validation and testing.

```
[ ] print(train_set.shape)
    print(test_images.shape)
    print(val_set.shape)
    print(train_images.shape)
```

```
(19030, 2)
(10196, 2)
(4758, 2)
(23788, 2)
```

In the above it shows number of rows as number of images, and it shows there are 2 columns. These 2 columns are of file path and label.

Here I can verify it as

```
print(train_set.head())  # Display first 5 rows of the training dataset
print(test_images.head())   # Display first 5 rows of the test dataset
print(val_set.head())    # Display first 5 rows of the validation dataset
```

```
                                            filepaths         labels
2544   /content/AugmentedAlzheimerDataset/AugmentedAl...  mild demented
924    /content/AugmentedAlzheimerDataset/AugmentedAl...  mild demented
6586   /content/AugmentedAlzheimerDataset/AugmentedAl...  mild demented
16720  /content/AugmentedAlzheimerDataset/AugmentedAl...   non demented
2211   /content/AugmentedAlzheimerDataset/AugmentedAl...  mild demented
                                            filepaths             labels
16224  /content/AugmentedAlzheimerDataset/AugmentedAl...       non demented
1188   /content/AugmentedAlzheimerDataset/AugmentedAl...      mild demented
21512  /content/AugmentedAlzheimerDataset/AugmentedAl...       non demented
2111   /content/AugmentedAlzheimerDataset/AugmentedAl...      mild demented
12968  /content/AugmentedAlzheimerDataset/AugmentedAl...  moderate demented
                                            filepaths              labels
32755  /content/AugmentedAlzheimerDataset/AugmentedAl...  very mild demented
29264  /content/AugmentedAlzheimerDataset/AugmentedAl...  very mild demented
6171   /content/AugmentedAlzheimerDataset/AugmentedAl...       mild demented
7865   /content/AugmentedAlzheimerDataset/AugmentedAl...       mild demented
4518   /content/AugmentedAlzheimerDataset/AugmentedAl...       mild demented
```

Distribution of images in train, validation and test sets: -

Here is the bar plot showing the distribution of images in train, validation and test sets.

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Dataset sizes
train_size = len(train_set)
val_size = len(val_set)
test_size = len(test_images)

# Labels and values
labels = ['Train', 'Validation', 'Test']
values = [train_size, val_size, test_size]

# Plot bar chart
plt.figure(figsize=(8, 5))
sns.barplot(x=labels, y=values, palette='viridis')

# Add value annotations
for i, v in enumerate(values):
    plt.text(i, v + 50, str(v), ha='center', fontsize=12)

plt.xlabel("Dataset Split")
plt.ylabel("Number of Images")
plt.title("Distribution of Images in Train, Validation, and Test Sets")
plt.ylim(0, max(values) + 200)
plt.show()
```

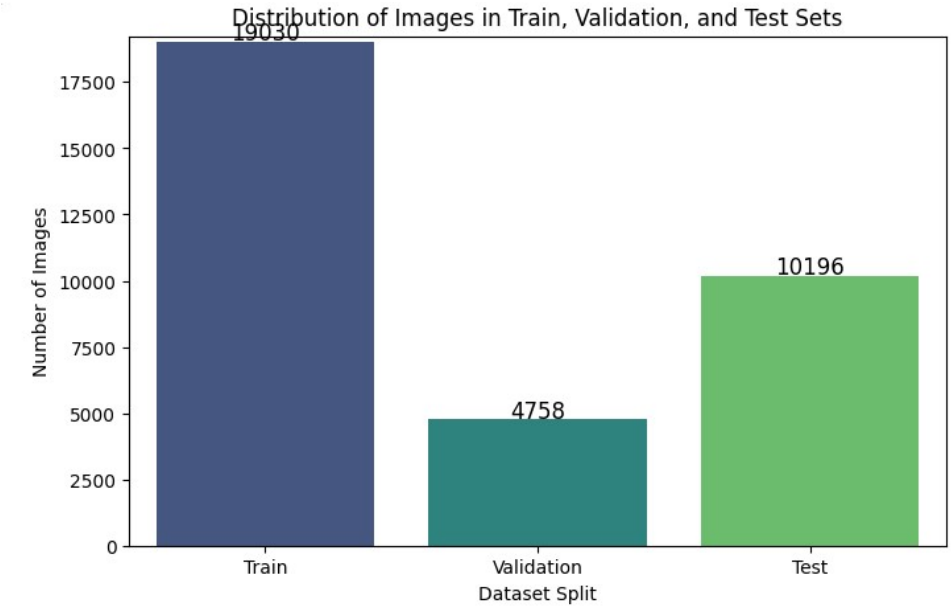


Figure 0-58 Distribution of images in train, validation and test sets

Data set pre-processing: -

I use ImageDataGenerator for the purpose of pre-processing of data set because it allows dynamics augmentation and normalization without modifying the original data set.
To ensure consistency across all the images in data set i use MobileNetV2's preprocessing function to normalize pixels and enhance model performance.

Then I structure dataset into **train, val, and test sets**, ensuring proper evaluation of the deep learning model.

I use flow_from_dataframe(), to load each dataset partition along with their file path and labels. The images are reseized to 244*244 pixels, which is a standard resolution for MobileNetV2-based models. The other parameters specified include:

- **color_mode='rgb'**: Ensuring all images are processed in the RGB color format.
- **class_mode='categorical'**: Used for multi-class classification, as Alzheimer's disease severity is divided into four categories.
- **batch_size=32**: The model processes 32 images per batch during training.
- **shuffle=True (for training and validation)**: Enhancing model generalization by preventing the network from memorizing the order of images.
- **shuffle=False (for testing)**: Maintaining order during model evaluation for consistent results.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf

# Define ImageDataGenerator with resizing
image_gen = ImageDataGenerator(
    preprocessing_function=tf.keras.applications.mobilenet_v2.preprocess_input  # Normalize pixels
)

# Resize images dynamically within the generator without modifying original dataset
target_size = (224, 224)  # Set a consistent resolution

train = image_gen.flow_from_dataframe(
    dataframe=train_set,
    x_col="filepaths",
    y_col="labels",
    target_size=target_size,  # Resize images
    color_mode='rgb',
    class_mode="categorical",  # Suitable for classification
    batch_size=32,
    shuffle=True  # Enable shuffle for better training
)

test = image_gen.flow_from_dataframe(
    dataframe=test_images,
    x_col="filepaths",
    y_col="labels",
    target_size=target_size,  # Resize images
    color_mode='rgb',
    class_mode="categorical",
    batch_size=32,
    shuffle=False  # No shuffle to maintain order for evaluation
)

val = image_gen.flow_from_dataframe(
    dataframe=val_set,
    x_col="filepaths",
    y_col="labels",
    target_size=target_size,  # Resize images
    color_mode='rgb',
    class_mode="categorical",
    batch_size=32,
    shuffle=True
)

print("✅ Dataset split successfully with resized images!")
```

```
Found 19030 validated image filenames belonging to 4 classes.
Found 10196 validated image filenames belonging to 4 classes.
Found 4758 validated image filenames belonging to 4 classes.
✅ Dataset split successfully with resized images!
```

Chapter 4: Analysis Approach, Implementation and Evaluation

In this chapter, the process of model building, evaluating, and deploying for real world use case is outlined. The goal was to develop a robust model which is capable of accurately diagnosing and predicting the stage of disease using the clinical and MRI data set. Multiple machine learning algorithms and deep learning algorithums were tested, and a series of evaluation metrics were employed to identify the best performing model. The model was evaluated using various metrics such as accuracy, precision, recall, F1-score, and AUC-ROC etc .

This chapter is further divided into two sections

- Analysis Approach and Implementation for clinical data set
- Analysis Approach and Implementation for MRI data set.

4.1 Analysis Approach, Implementation and Results for clinical data set: -

Introduction: -

In this chapter, the process of model building, evaluating, and selecting the best-performing model is outlined. The goal was to develop a robust model capable of detecting a disease using clinical data set. For this purpose, multiple machine learning algorithms were tested, and a series of evaluation matrix were applied to identify the best performing model. Grid search was used to choose best performing hyperparameters. Model was evaluated using various metrics such as accuracy, precision, recall etc.

Main Problems and Solutions: -

Several challenges were encountered during the model process. The primary challenge was that the data set contained 35 features, but these features were not contributing equally to prediction accuracy. Recursive Feature Elimination and Cross-Validation (RFECV) was used to remove the irrelevant features. It was surprising for me that only top 6 features are contribution to the 95% accuracy of model prediction. So reducing the dataset to most relevant ones, thereby improved model performance and training time.

Another challenge was dealing with class imbalance in the dataset, as the dataset showed a disproportionate number of positive and negative diagnoses. To address this, the models were trained with the class_weight='balanced' parameter, which automatically adjusts weights for the class imbalance.

Model Building, Evaluation, and Analysis: -

Multiple models were considered and evaluated using **GridSearchCV** to find the best combination of hyperparameters for each model. The following models were tested:

- **Random Forest Classifier**
- **Logistic Regression**
- **Support Vector Machine (SVM)**
- **Gradient Boosting Classifier**

The models were evaluated based on their cross-validation accuracy scores. The best-performing model was selected for further evaluation and fine-tuning. Various metrics, including **accuracy, precision, recall, F1-score, and AUC-ROC**, were used to assess model performance.

Machine Learning Models and Hyperparameter Tuning: -

After performing data preprocessing and exploratory data analysis, I proceeded to apply various machine learning models to the dataset. To ensure optimal performance from each algorithm, I conducted hyperparameter tuning using Grid Search with Cross-Validation. This method systematically searches through a specified parameter grid for each model, evaluating performance using cross-validation to reduce the risk of overfitting and provide more robust results.

Here are different machine learning models that i used with their param grid.

```
models = {
    'RandomForest': {
        'model': RandomForestClassifier(random_state=0, class_weight='balanced'),
        'params': {
            'n_estimators': [50, 100, 200],
            'max_depth': [5, 10, 15],
            'min_samples_split': [10, 20, 30],
            'min_samples_leaf': [5, 10, 15],
            'max_features': ['sqrt', 'log2']
        }
    },
    'LogisticRegression': {
        'model': LogisticRegression(random_state=0, class_weight='balanced'),
        'params': {
            'C': [0.1, 1, 10],
            'penalty': ['l2'],
            'solver': ['lbfgs', 'liblinear']
        }
    },
    'SVM': {
        'model': SVC(random_state=0, class_weight='balanced', probability=True),
        'params': {
            'C': [0.1, 1, 10],
            'kernel': ['linear', 'rbf'],
            'gamma': ['scale', 'auto']
        }
    },
    'GradientBoosting': {
        'model': GradientBoostingClassifier(random_state=0),
        'params': {
            'n_estimators': [50, 100, 200],
            'learning_rate': [0.01, 0.1, 0.2],
            'max_depth': [3, 5, 10],
            'min_samples_split': [10, 20, 30]
        }
    }
}
```

Evaluating model using gridSearchCV:-

```
#Evaluate model using GridSearchCv
best_model=None
best_score=0
best_model_name=''

for model_name, model_info in models.items():
    print(f"Training and tuning {model_name}...")
    grid_search=GridSearchCV(
        estimator=model_info['model'],
        param_grid=model_info['params'],
        cv=5,
        scoring='accuracy',
        n_jobs=-1
    )
    grid_search.fit(X_train_fs,y_train)

#get best model
# Get the best model and its score
    if grid_search.best_score_ > best_score:
        best_score = grid_search.best_score_
        best_model = grid_search.best_estimator_
        best_model_name = model_name

    # Print results for the current model
    print(f"Best parameters for {model_name}: {grid_search.best_params_}")
    print(f"Best cross-validation accuracy for {model_name}: {grid_search.best_score_ * 100:.2f}%")
    print("-" * 50)

# Train the best model on the full training set
best_model.fit(X_train_fs, y_train)
```

For each model:

- **GridSearchCV** was applied with **5-fold cross-validation**, meaning it will take test data 5 times to evaluate the best model and best hyperparameters.
- The **scoring metric** used was **accuracy**, aiming to maximize correct predictions.
- The parameter n_jobs=-1 was specified, so that all the CPU's will be used.
- Then **grid search** was fitted on the training data (X_train, y_train), which allowing the model to identify the best combination of hyperparameters based on cross-validated performance.

After the grid search was completed for all models:

- The **best model** and its **corresponding hyperparameters** were printed for further evaluation.
- Finally, the best model which was determined by the highest cross-validation accuracy **refited on the full training dataset** to prepare it for evaluation on the test set.

```
Training and tuning RandomForest...
Best parameters for RandomForest: {'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 5, 'min_samples_split': 10, 'n_estimators': 200}
Best cross-validation accuracy for RandomForest: 95.50%
--------------------------------------------------
Training and tuning LogisticRegression...
Best parameters for LogisticRegression: {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
Best cross-validation accuracy for LogisticRegression: 84.07%
--------------------------------------------------
Training and tuning SVM...
Best parameters for SVM: {'C': 10, 'gamma': 'auto', 'kernel': 'rbf'}
Best cross-validation accuracy for SVM: 90.28%
--------------------------------------------------
Training and tuning GradientBoosting...
Best parameters for GradientBoosting: {'learning_rate': 0.2, 'max_depth': 10, 'min_samples_split': 10, 'n_estimators': 100}
Best cross-validation accuracy for GradientBoosting: 95.46%
--------------------------------------------------
40]: RandomForestClassifier
RandomForestClassifier(class_weight='balanced', max_depth=10,
                       min_samples_leaf=5, min_samples_split=10,
                       n_estimators=200, random_state=0)
```

Figure 4.1 Best Model

From the above Figure 4.1, it can be observed that the Random Forest Classifier emerged as the best-performing model, achieving an accuracy of 95%. The optimal hyperparameters identified for this model were:

- max_depth = 10
- min_samples_leaf = 5
- min_samples_split = 10
- n_estimators = 200
- random_state = 0

Test accuracy: -

Then I fit the best model on test set and determine its accuracy on the test set.

```
[79]: # Evaluate the best model on the test set
y_pred = best_model.predict(X_test_fs)
y_pred_proba = best_model.predict_proba(X_test_fs)[:, 1]  # For AUC-ROC# Evaluation metrics
test_accuracy = accuracy_score(y_test, y_pred)
print(f"\nBest Model: {best_model_name}")
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")

Best Model: RandomForest
Test Accuracy: 95.12%
```

Figure 4-1 Test Accuracy

From Figure 4-1, we can see that model is performing on the test set with an accuracy of 95.12%. Which is very good.

Generalization Performance: -

Test accuracy=95.12%

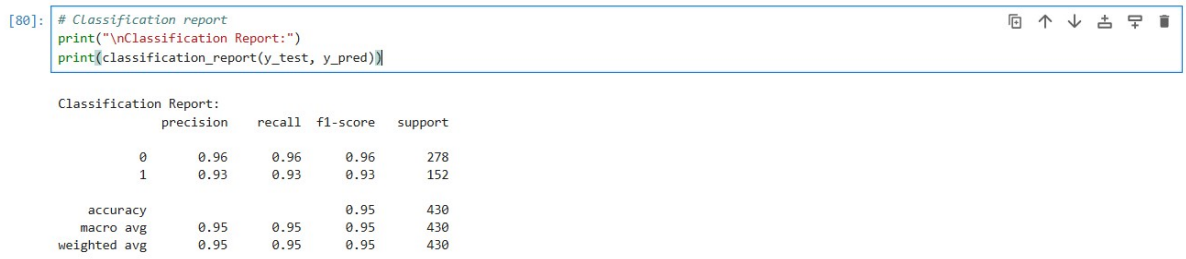
Train accuracy=95.46%

With an accuracy of 95.12% on the test set, the model showed great generalisation performance and high predictive capabilities. Also, the model seems to be avoiding overfitting and successfully collecting data patterns without learning the training set from memory, as there is only a little discrepancy between training and test accuracy (95.12% vs. 95.46%).

Classification Report: -

A classification report was generated by comparing the predicted labels (y_pred) with the actual labels (y_test). This evaluation provided a comprehensive understanding of the model's performance using the following metrics:

- The percentage of accurate predictions relative to the total number of occurrences is called accuracy.
- Preciseness: the percentage of positively anticipated cases that really occurred out of all the positive instances projected.
- Recall is the fraction of true positives that were accurately predicted relative to the total number of positives.
- *F1-score:* The harmonic mean of Precision and Recall, providing a balance between the two



```
[80]: # Classification report
      print("\nClassification Report:")
      print(classification_report(y_test, y_pred))

      Classification Report:
                    precision    recall  f1-score   support

                 0       0.96      0.96      0.96       278
                 1       0.93      0.93      0.93       152

          accuracy                           0.95       430
         macro avg       0.95      0.95      0.95       430
      weighted avg       0.95      0.95      0.95       430
```

Figure 4-2 Classification Report

Analysis: -

In this Figure 4-2 we can see that

- When it comes to illness detection, the model has a high recall, which means it gets the majority of instances right.
- As a result of the model's high accuracy, fewer false alarms are likely to occur.

- The high F1-score indicates a well-balanced compromise between recall and accuracy.

Confusion matrix: -

```
[39]: # Confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

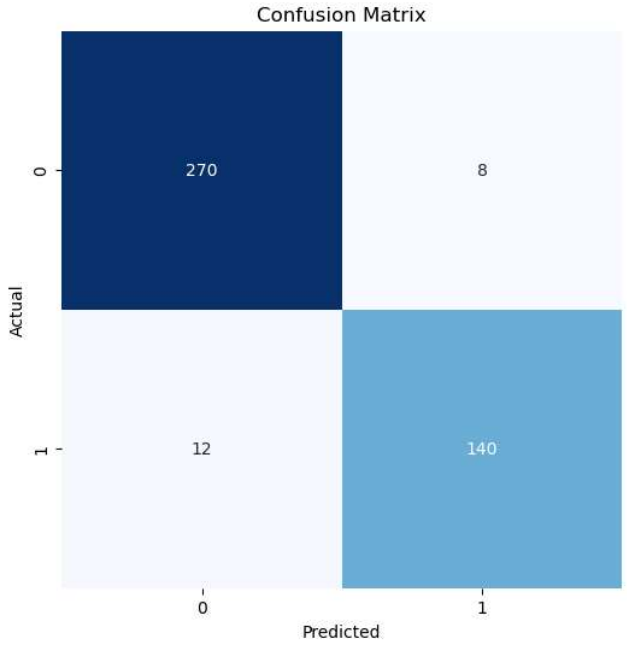


Figure 4-4 Confusion matrix

From the above Figure 4-4 it can be observed that the model has True Positive of 270 mean it predicted 270 people which have no disease. On the other hand, its prediction on Ture negative of 140 people, mean it predicts 140 people with the Alzheimer disease. It has 8 False negative, mean it predicts 8 people wrongly predicted with no disease but in actual they have disease. It has 12 False negative, means it predicts 11 people with disease but in actual they don't have.

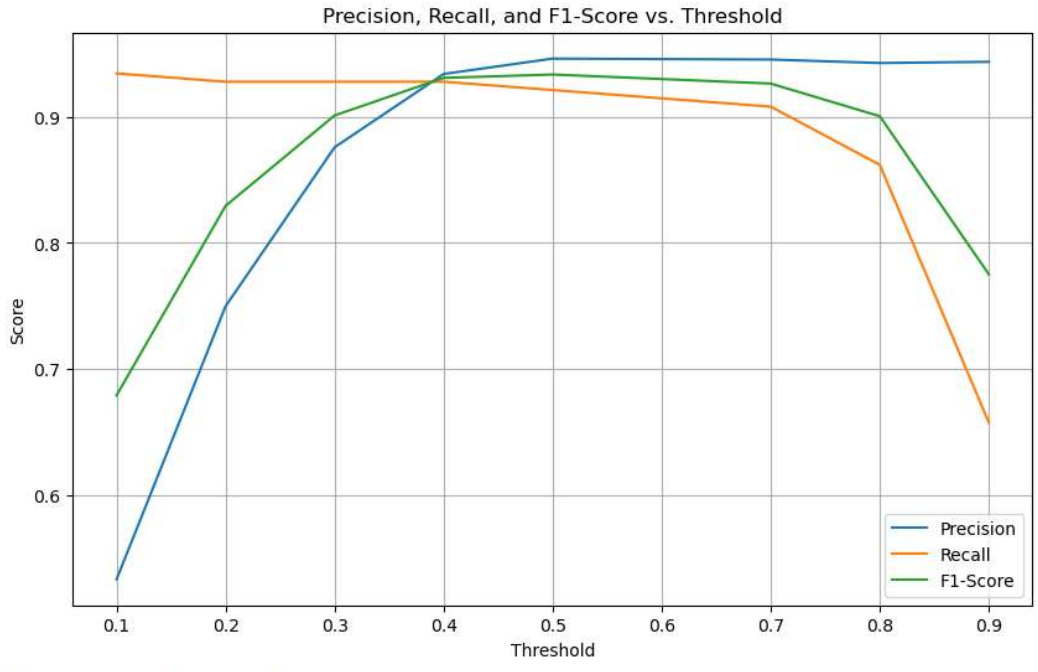
- **True Positives (TP):** The model correctly predicted 270 individuals as not having Alzheimer's disease.
- **True Negatives (TN):** 140 individuals were correctly identified as having Alzheimer's disease.
- **False Negatives (FN):** 8 individuals were incorrectly predicted as not having the disease, while in reality, they did have it.
- **False Positives (FP):** 12 individuals were wrongly predicted to have the disease, even though they did not.

Overall, the confusion matrix suggests a strong performance with high accuracy and balanced prediction.

Threshold Tuning: -

From above we can see that the false positive is 8, in medical we are too much sensitive about false positive because if a person has a disease and he was not detectable by the system than it would have some serious conceconsies. So, system should have minimum false positives. So, to minimize the false positives I am going to do threshold tuning to get the best threshold and use that threshold to minimize the false positives .

```
[109]: from sklearn.metrics import precision_score, recall_score, f1_score

# Define a range of thresholds
thresholds = np.arange(0.1, 1.0, 0.1)

# Evaluate precision, recall, and F1-score for each threshold
results = []
for threshold in thresholds:
    y_pred_threshold = (y_pred_proba >= threshold).astype(int)
    precision = precision_score(y_test, y_pred_threshold)
    recall = recall_score(y_test, y_pred_threshold)
    f1 = f1_score(y_test, y_pred_threshold)
    results.append([threshold, precision, recall, f1])

# Convert results to a DataFrame
results_df = pd.DataFrame(results, columns=['Threshold', 'Precision', 'Recall', 'F1-Score'])

# Plot results
plt.figure(figsize=(10, 6))
plt.plot(results_df['Threshold'], results_df['Precision'], label='Precision')
plt.plot(results_df['Threshold'], results_df['Recall'], label='Recall')
plt.plot(results_df['Threshold'], results_df['F1-Score'], label='F1-Score')
plt.xlabel('Threshold')
plt.ylabel('Score')
plt.title('Precision, Recall, and F1-Score vs. Threshold')
plt.legend()
plt.grid(True)
plt.show()

# Print the best threshold based on F1-Score
best_threshold = results_df.loc[results_df['F1-Score'].idxmax(), 'Threshold']
print(f"Best Threshold for F1-Score: {best_threshold:.2f}")
```




```
Best Threshold for F1-Score: 0.50
```

Figure 4-3Threshold tuning

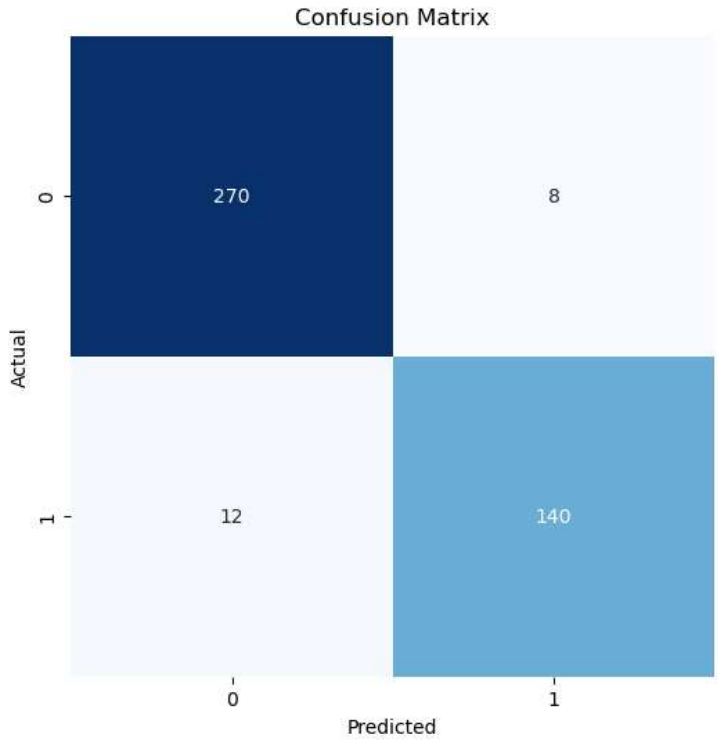
From above Figure 4-4Threshold tuning, we can see that the best threshold is 0.50.

Updating the model: -

Actually, the default threshold is already 0.5 and model also predicts that the best threshold for f1-score is 0.50. So, let's try with updating my model with the new threshold as 0.5

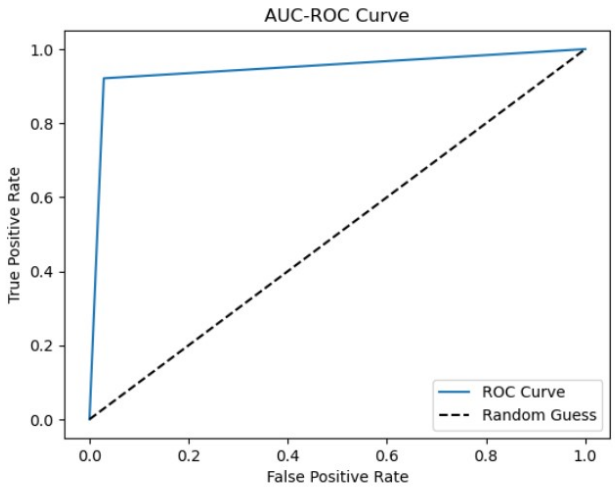
```
6]: from sklearn.metrics import precision_recall_curve

# Get predicted probabilities for class 1 (Alzheimer's)
y_pred_proba = best_model.predict_proba(X_test_fs)[:, 1]

# Experiment with different thresholds
threshold = 0.5

# Convert probabilities to binary predictions based on the new threshold
y_pred_adjusted = (y_pred_proba >= threshold).astype(int)

# Recalculate confusion matrix & recall
cm_adjusted = confusion_matrix(y_test, y_pred_adjusted)
print("Adjusted Confusion Matrix:\n", cm_adjusted)

# New Recall Score
new_recall = cm_adjusted[1, 1] / (cm_adjusted[1, 0] + cm_adjusted[1, 1])
print(f"New Recall: {new_recall:.2f}")

Adjusted Confusion Matrix:
 [[270   8]
 [ 12 140]]
New Recall: 0.92
```

This is the same confusion matrix for the same model, means no difference is confusion matrix.



AUC Curve: -

```
[124]: # AUC-ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_adjusted )
plt.plot(fpr, tpr, label='ROC Curve')
plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('AUC-ROC Curve')
plt.legend()
plt.show()
print(f"AUC-ROC Score: {roc_auc_score(y_test, y_pred_proba):.2f}")
```




```
AUC-ROC Score: 0.94
```

Figure 4-4 AUC Curve

This Figure 4-5 AUC Curve, represents the classification performance of our model at different threshold. It shows AUC-ROC score of 94% which means it has a very good ability to distinguish between positive and negative cases, suggesting a strong predictive power.

The curve is significantly above the random guess line suggests that model is significantly better than random guesses.

Cross-validation accuracy: -

```
[83]: # Cross-Validation to evaluate generalization
cv_scores = cross_val_score(best_model, X_train_fs, y_train, cv=5, scoring='accuracy')
print(f"Cross-Validation Accuracy: {cv_scores.mean() * 100:.2f}%")
```

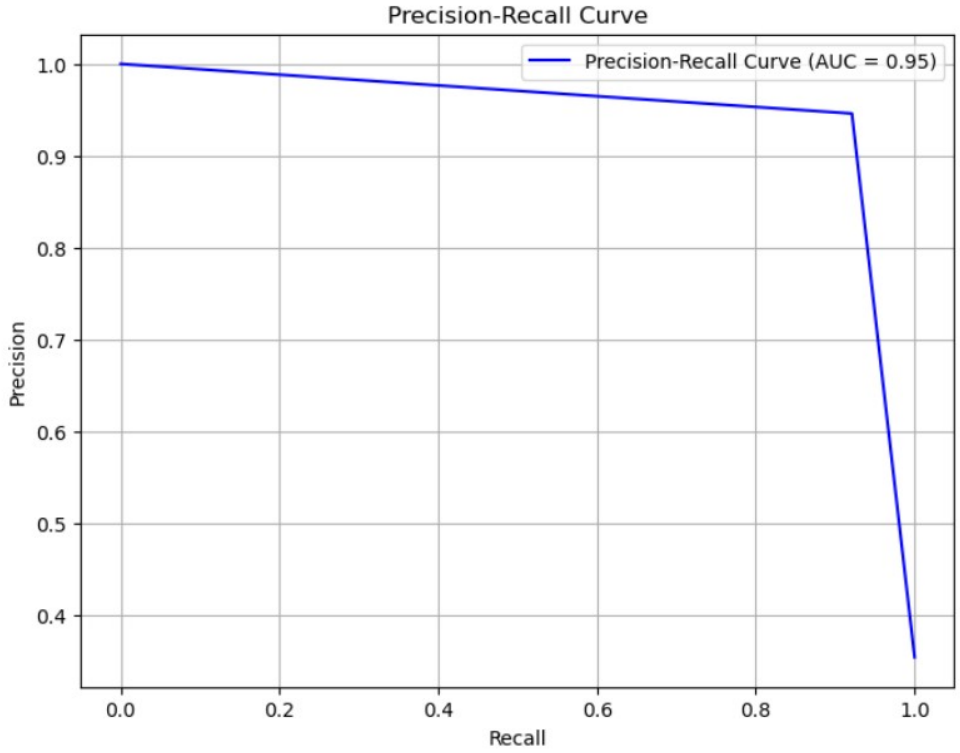
```
Cross-Validation Accuracy: 95.50%
```

Figure 4.7 Cross Validation Accuracy

From above Figure 4.7 Cross Validation Accuracy, we can see that the model cross-validation accuracy is 95% on training dataset. It is a reliable matrix to ensure that the model generalizes well on the unseen data as well.

Precision-recall curve: -

```
[125]: from sklearn.metrics import precision_recall_curve, auc

# Calculate precision-recall curve values
precision, recall, _ = precision_recall_curve(y_test, y_pred_adjusted )

# Calculate AUC of Precision-Recall curve
pr_auc = auc(recall, precision)

# Plot Precision-Recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='b', label=f'Precision-Recall Curve (AUC = {pr_auc:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='best')
plt.grid(True)
plt.show()

# Print Precision-Recall AUC score
print(f"Precision-Recall AUC Score: {pr_auc:.2f}")
```




```
Precision-Recall AUC Score: 0.95
```

Figure 4-5 Precision-recall curve

Figure 4-5 Precision-recall curve shows the precision-recall curve with an accuracy of 92%, indicating a strong ability of model to classify positive and negative cases. The curve itself illustrates the trade-off between precision and recall as the classification threshold changes.

Learning Curves to diagnose overfitting: -

```
[84]: # Plot Learning Curves to diagnose overfitting
train_sizes, train_scores, val_scores = learning_curve(
    best_model, X_train_fs, y_train, cv=5, scoring='accuracy', train_sizes=np.linspace(0.1, 1.0, 10)
)

plt.plot(train_sizes, train_scores.mean(axis=1), label='Training Accuracy')
plt.plot(train_sizes, val_scores.mean(axis=1), label='Validation Accuracy')
plt.xlabel('Training Set Size')
plt.ylabel('Accuracy')
plt.title('Learning Curves')
plt.legend()
plt.show()
```

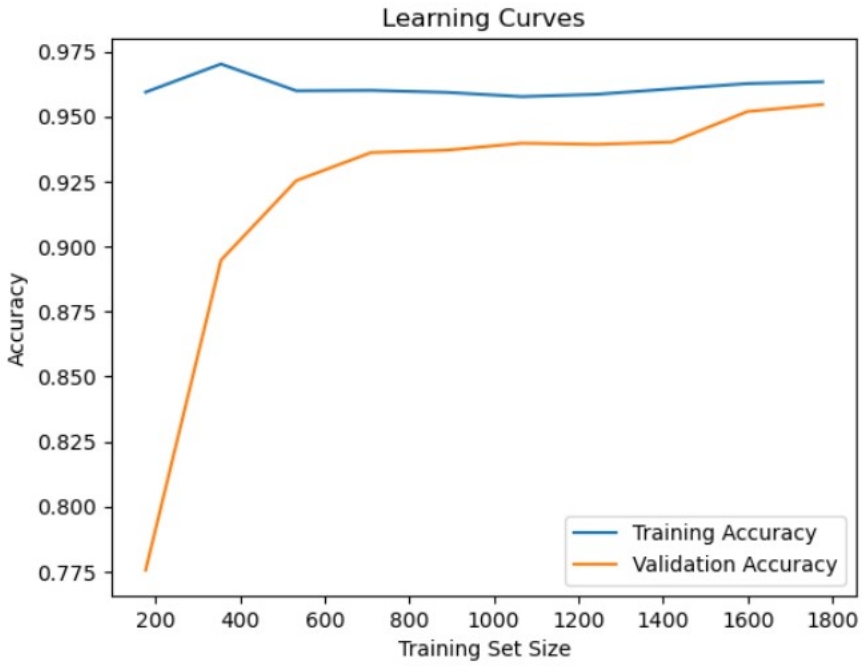


Figure 4-6 Learning Curves to diagnose overfitting

From this graph, we can see that there is not much difference in between training and validation accuracy which suggest the model is not overfitted. After sample size 400 the training and validation accuracy has a difference of only 0.04 to 0.05, which is very good.

Calibration Curve: -

```
[127]: from sklearn.calibration import calibration_curve

# Calculate calibration curve
prob_true, prob_pred = calibration_curve(y_test, y_pred_adjusted , n_bins=10)

# Plot calibration curve
plt.figure(figsize=(8, 6))
plt.plot(prob_pred, prob_true, marker='o', label='Calibration Curve')
plt.plot([0, 1], [0, 1], linestyle='--', label='Perfectly Calibrated')
plt.xlabel('Predicted Probability')
plt.ylabel('True Probability')
plt.title('Calibration Curve')
plt.legend()
plt.grid(True)
plt.show()
```

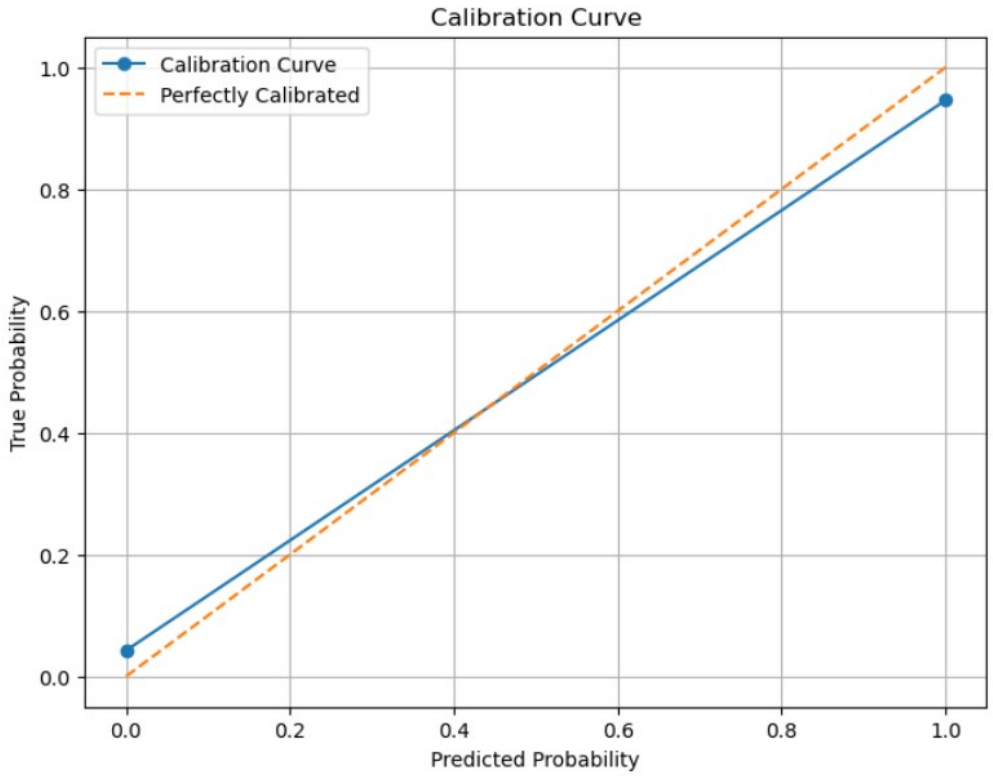


Figure 4-7 Calibration Curve

From the above we can see that our model's calibrated curves are perfectly aligns with the perfectly calibrated curve with a little difference.

Out-of-Fold AUC-ROC score: -

```
[56]: from sklearn.model_selection import cross_val_predict

# Get out-of-fold predictions
oof_predictions = cross_val_predict(best_model, X_train_fs, y_train, cv=5, method='predict_proba')[:, 1]

# Evaluate out-of-fold AUC-ROC
oof_auc = roc_auc_score(y_train, oof_predictions)
print(f"Out-of-Fold AUC-ROC Score: {oof_auc:.2f}")

Out-of-Fold AUC-ROC Score: 0.97
```

Here out-of-fold AUC-ROC score was calculated as 0.97 which means that model is generalizing well and making excellent predictions.

Confidence Interval for Accuracy: -

```
[57]: from scipy.stats import sem, t

# Calculate confidence interval for accuracy
confidence = 0.95
n = len(y_test)
mean_accuracy = test_accuracy
std_error = sem([accuracy_score(y_test, best_model.predict(X_test_fs)) for _ in range(100)])  # Bootstrap
confidence_interval = t.interval(confidence, n-1, loc=mean_accuracy, scale=std_error)

print(f"Confidence Interval for Accuracy: {confidence_interval}")

Confidence Interval for Accuracy: (0.9534883720930233, 0.9534883720930233)
```

A confidence of interval gave a range within which the true accuracy of model is likely to fall. Since from above it can be observed that confidence of interval starts and ends at the same value which indicates that the estimate for model accuracy is very precise. In other words, accuracy is 95% with very little uncertainty in that estimate.

Saving the best model: -

First, I saved the best model and the same standard scaler parameter that I used to train the model as follow

```
[ ]: import pandas as pd
import joblib
from sklearn.preprocessing import StandardScaler
joblib.dump(scaler, 'scaler.pkl')
```

```
[ ]: import joblib

# Save the best model to a file
joblib.dump(best_model, 'best_model.pkl')
```

4.1.1 A description of how the developed model was implemented to meet the requirements

Doctor-Assisted Diagnosis System: -

To enhance clinical decision-making, I implemented a **doctor-assisted machine learning model** . My system allows a healthcare professional to input a patient's symptoms and receive a real-time prediction regarding the presence of the disease.

System Workflow:

1. The doctor enters patient symptoms corresponding to the **six selected features from RFECV**.
2. The input is preprocessed and standardized using the previously fitted scaler.
3. The best-performing machine learning model (selected via **GridSearchCV**) which is random Forest makes a prediction.
4. The system outputs a **binary diagnosis (Disease / No Disease)** along with a **confidence score**.

```
# Load the trained best model and scaler (Assuming they were saved earlier)
best_model = joblib.load("best_model.pkl")  # Save your best model after training
scaler = joblib.load("scaler.pkl")  # Save the scaler after fitting on training data

# Selected features from RFECV (Replace with actual feature names)
selected_features = ['DietQuality', 'MMSE', 'FunctionalAssessment', 'MemoryComplaints','BehavioralProblems', 'ADL']

# Function for doctor to enter symptoms
def get_patient_input():
    print("\nEnter patient symptoms for the following features:")
    patient_data = []
    for feature in selected_features:
        value = float(input(f"{feature}: "))  # Input values for selected features
        patient_data.append(value)

    # Convert input into NumPy array and reshape
    patient_array = np.array(patient_data).reshape(1, -1)

    # Standardize input using previously fitted scaler
    patient_array_scaled = scaler.transform(patient_array)

    return patient_array_scaled

# Function to make prediction
def predict_disease():
    patient_input = get_patient_input()
    prediction = best_model.predict(patient_input)
    probability = best_model.predict_proba(patient_input)[:, 1]

    # Print the result
    print("\nDiagnosis Result:")
    if prediction[0] == 1:
        print("❌ The patient is likely to have the disease.")
    else:
        print("✅ The patient is not likely to have the disease.")

    print(f"Confidence: {probability[0] * 100:.2f}%")

# Run prediction system
predict_disease()
```

Explainable AI:-

Then I add explainable AI which is LIME which will be useful for doctor to see why system is saying the patient has disease or not, or how the system is predicting the disease. This will make junior doctor to make more better decisions.

```
# Function to explain prediction using LIME with better formatting
def explain_prediction_lime(patient_input, raw_input):
    # Initialize LIME explainer
    explainer = lime.lime_tabular.LimeTabularExplainer(
        training_data=scaler.transform(np.random.rand(100, len(selected_features))),  # Use random data as training data for explanation
        feature_names=selected_features,
        class_names=[str(i) for i in range(best_model.n_classes_)],  # Assumes a multi-class classifier
        mode="classification"
    )

    # Explain the prediction for the given input
    explanation = explainer.explain_instance(patient_input[0], best_model.predict_proba, num_features=len(selected_features))

    # Fetch prediction probabilities
    probabilities = best_model.predict_proba(patient_input)[0]

    # Display prediction probabilities in a clean way
    print("\n==================== Prediction Probabilities ====================")
    for idx, prob in enumerate(probabilities):
        print(f"Class {idx} (Class {idx}): {prob*100:.2f}%")
    print("================================================================")

    # Feature Contributions (LIME's Explanation of the Prediction)
    print("\n==================== Feature Contributions ====================")
    feature_contributions = explanation.as_list()

    # Create a table for feature contributions
    table_data = []
    for feature, contribution in feature_contributions:
        table_data.append([feature, f"{contribution:.4f}"])

    # Display the table using tabulate for better readability
    print(tabulate(table_data, headers=["Feature", "Contribution"], tablefmt="fancy_grid"))
    print("================================================================")
```

```
    # Display input feature values
    print("\n==================== Input Feature Values ====================")
    feature_values = zip(selected_features, raw_input)
    for feature, value in feature_values:
        print(f"{feature}: {value:.2f}")
    print("================================================================")

    # Visualize the explanation with styled plot
    fig = explanation.as_pyplot_figure()
    fig.suptitle('LIME Explanation of the Prediction', fontsize=14, fontweight='bold')
    plt.show()


# Function to make prediction and print results in a more readable format
def predict_disease():
    patient_input, raw_input = get_patient_input()
    prediction = best_model.predict(patient_input)
    probability = best_model.predict_proba(patient_input)

    # Print the results in a styled way
    print("\n==================== Diagnosis Result ====================")
    if prediction[0] == 1:
        print(f"Predicted Class: 1 (Class 1) - Patient likely has Alzheimer's disease")
    else:
        print(f"Predicted Class: 0 (Class 0) - Patient likely does not have Alzheimer's disease")

    print("Class Probabilities:")
    print(f"╒═══════════╤═══════════════╕")
    print(f"│  Class    │  Probability  │")
    print(f"╞═══════════╪═══════════════╡")
    for idx, prob in enumerate(probability[0]):
        print(f"│ Class {idx} │    {prob*100:.2f}%    │")
    print(f"╘═══════════╧═══════════════╛")

    # Explain the prediction using LIME
    explain_prediction_lime(patient_input, raw_input)

# Run prediction system
predict_disease()
```

Output: -

```
Enter patient symptoms for the following features:
DietQuality:  1
MMSE:  1
FunctionalAssessment:  1
MemoryComplaints:  1
BehavioralProblems:  1
ADL:  1

==================== Diagnosis Result ====================
Predicted Class: 1 (Class 1) - Patient likely has Alzheimer's disease
Class Probabilities:
```

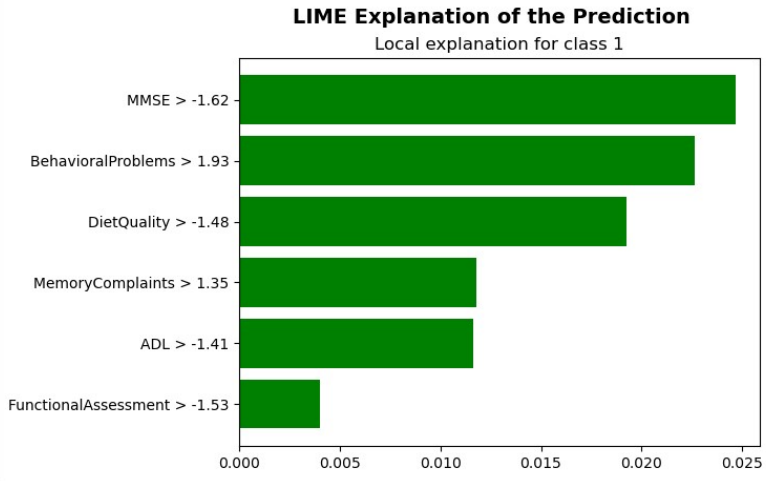
Class	Probability
Class 0	2.89%
Class 1	97.11%

```
==================== Prediction Probabilities ====================
Class 0 (Class 0): 2.89%
Class 1 (Class 1): 97.11%
================================================================

==================== Feature Contributions ====================
```

Feature	Contribution
MMSE > -1.62	0.0247
BehavioralProblems > 1.93	0.0226
DietQuality > -1.48	0.0193
MemoryComplaints > 1.35	0.0118
ADL > -1.41	0.0116
FunctionalAssessment > -1.53	0.004

```
================================================================
```

```
==================== Input Feature Values ====================
DietQuality: 1.00
MMSE: 1.00
FunctionalAssessment: 1.00
MemoryComplaints: 1.00
BehavioralProblems: 1.00
ADL: 1.00
================================================================
```



Input Feature Values

At the top, it shows the actual input values for the 6 features used:

DietQuality: 1.00

MMSE: 1.00

FunctionalAssessment: 1.00

MemoryComplaints: 1.00

BehavioralProblems: 1.00

ADL: 1.00

These were the values fed into the model for this individual prediction.

Bar Graph: Contribution of Each Feature

Each bar shows:

- Feature Condition (e.g., MMSE > -1.62)
- Positive Influence (in green): These features pushed the prediction toward Class 1 (disease).
- Length of the Bar: Indicates how strongly that feature influenced the decision.

For example:

- MMSE > -1.62 had the strongest influence in predicting the disease.
- BehavioralProblems > 1.93 and DietQuality > -1.48 also contributed positively.

All bars are green, meaning every listed feature helped push the prediction toward Class 1.

In Summary

- LIME breaks down a complex black-box model prediction into understandable parts.
- It tells why the model predicted disease for this specific person.
- Useful in clinical settings for interpretability and trust, especially for doctors.

4.1.2 A justification of the methods and tools adopted

- **RFECV**: There were 35 features in the data set but all of them was not contributing to the prediction accuracy that's why for avoiding overfitting I used RFECV to remove irrelevant features from the data set and only selects the relevant features. So, in this way I would get more accuracy for prediction of disease and less processing time as well.

- **GridSearchCV**: GridSearchCV was used to find the best hyperparameters by testing all combinations using 5-fold cross-validation. It ensures optimal performance and robustness.

- **Random Forest:** Random Forest was selected as the best model based on its performance. It gave 95% accuracy which was highest from all other models. It handled class imbalance well, reduced overfitting, and provided stable results across different data splits.

- **LIME:** LIME was added to explain model predictions, helping junior doctors understand why the system predicts disease or not. This build trust and supports better decision-making .

4.1.3 Conclusion: -

Throughout this project, different machine learning models were tested like Random Forest, SVM, Logistic Regression, and Gradient Boosting to find out the most accurate one. To ensure fair comparison and fine-tuning, GridSearchCV with 5-fold cross-validation was used. After testing, the **Random Forest Classifier** stood out by delivering the best results, with a **95.12% accuracy on the test set**, and very close training accuracy, which shows the model isn't overfitting.

We looked at various performance metrics such as precision, recall, F1-score, AUC, and confusion matrix. These showed that the model consistently performs well in distinguishing between patients with and without Alzheimer's disease.

To make the model more usable in a real-world healthcare setting, we built a system that allows doctors to input symptoms and get immediate predictions along with confidence levels. To make these predictions understandable, we also integrated **LIME**, an explainable AI tool, so doctors can see which features influenced the decision. This is especially helpful for building trust and assisting junior medical staff to make informed decision about the patient.

In short, we developed an accurate, reliable, and explainable model tailored for clinical use, combining technical performance with practical usability.

4.2 Analysis Approach, Implementation and Results for MRI data set: -

Introduction: -

Magnetic Resonance Imaging (MRI) is a widely used, non-invasive method in healthcare for identifying and tracking conditions like tumours, brain lesions, and other degenerative diseases. While it's incredibly valuable, going through MRI scans by hand takes a lot of time, depends heavily on the experience of the radiologist, and can lead to inconsistent results. Because of this, there's a growing need for automated systems that can analyse MRI data accurately, consistently, and on a larger scale. In this chapter, the goal was to develop a robust model capable of classifying a disease into different categories using MRI data set. For this purpose, a convolutional neural network was used, and a series of evaluation matrix were applied to identify the best performing parameters. Model was evaluated using various metrics such as accuracy, precision, recall etc.

Main Problems and Solutions: -

During this project, a few key challenges came up. First, MRI data is high-dimensional and quite complex, which can make models prone to overfitting or inefficient performance. Another challenge was the variability in image quality, since the scans were taken using different machines or settings. This kind of inconsistency can affect how well a model learns and generalizes.

To address these issues

- We applied a series of preprocessing steps to clean and normalize the images. Techniques like ImageDataGenerator with resizing and application of applications.mobilenet_v2 helped bring consistency across the dataset.
- We used **batch normalization** to stabilize and accelerate training.
- Regularization techniques, such as **dropout**, were used to prevent overfitting.
- Finally, we implemented a deep learning model that could automatically learn complex features from the MRI scans.

Model building: -

```
[ ]  model = keras.models.Sequential([
        keras.layers.Conv2D(filters=128, kernel_size=(8, 8), strides=(3, 3), activation='relu', input_shape=(224, 224, 3)),
        keras.layers.BatchNormalization(),

        keras.layers.Conv2D(filters=256, kernel_size=(5, 5), strides=(1, 1), activation='relu', padding="same"),
        keras.layers.BatchNormalization(),
        keras.layers.MaxPool2D(pool_size=(3, 3)),

        keras.layers.Conv2D(filters=256, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding="same"),
        keras.layers.BatchNormalization(),
        keras.layers.Conv2D(filters=256, kernel_size=(1, 1), strides=(1, 1), activation='relu', padding="same"),
        keras.layers.BatchNormalization(),
        keras.layers.Conv2D(filters=256, kernel_size=(1, 1), strides=(1, 1), activation='relu', padding="same"),
        keras.layers.BatchNormalization(),

        keras.layers.Conv2D(filters=512, kernel_size=(3, 3), activation='relu', padding="same"),
        keras.layers.BatchNormalization(),
        keras.layers.MaxPool2D(pool_size=(2, 2)),

        keras.layers.Conv2D(filters=512, kernel_size=(3, 3), activation='relu', padding="same"),
        keras.layers.BatchNormalization(),

        keras.layers.Conv2D(filters=512, kernel_size=(3, 3), activation='relu', padding="same"),
        keras.layers.BatchNormalization(),

        keras.layers.MaxPool2D(pool_size=(2, 2)),

        keras.layers.Conv2D(filters=512, kernel_size=(3, 3), activation='relu', padding="same"),
        keras.layers.BatchNormalization(),

        keras.layers.MaxPool2D(pool_size=(2, 2)),

        keras.layers.Conv2D(filters=512, kernel_size=(3, 3), activation='relu', padding="same"),
        keras.layers.BatchNormalization(),

        keras.layers.MaxPool2D(pool_size=(2, 2)),

        keras.layers.Flatten(),
        keras.layers.Dense(1024, activation='relu'),
        keras.layers.Dropout(0.5),
        keras.layers.Dense(1024, activation='relu'),
        keras.layers.Dropout(0.5),
        keras.layers.Dense(4, activation='softmax')
    ])
```

first of all, i created a model of sequential class. Once i got the sequential model i started adding the layers.

Input layer: -

The model takes 244*244 RGB images as input.

Convolution layers and feature extraction: -

This model contains multiple convolution layers with increasing filter sizes and ReLU activation is applied to introduce non-linearity. Batch normalization is also used to stabilize training.

First Conv Layer: -

So the first layer we going to have is convolutional layer having 128 filters of size 8 by 8 and stride is 3 by 3 and activation function is relu and input shape is 244,244,3. This layer is followed by a batch normalization which is used to normalize the activation of a layer before passing it to next layer.

Second Conv layer: -

Then I add a second convolution layer with the following features:

Filter size: 128

Kernal size:5,5

Stride:1,1

Activation :ReLU

Deeper Conv layer: -

Then I add several layers with 256 and 512 filters to extract high-level features.

Pooling Layers (Dimensionality Reduction): -

MaxPooling layers are applied after certain convolution blocks. These pool sizes vary from (3,3) to (2,2) to reduce spatial dimensions and retain key features.

Fully Connected (Dense) Layers: -

Then I flattened the extracted features and passed them through two dense layers of 1024 neurons each. I also apply dropout layer of 0.5 to prevent overfitting.

Output Layer: -

Then I apply the final dense layer consisting of 4 neurons, corresponding to the 4 Alzheimer disease categories. Then at the end i apply softmax activation function to output probability for classification.

Summery: -

```
[ ] model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 73, 73, 128)	24,704
batch_normalization (BatchNormalization)	(None, 73, 73, 128)	512
conv2d_1 (Conv2D)	(None, 73, 73, 256)	819,456
batch_normalization_1 (BatchNormalization)	(None, 73, 73, 256)	1,024
max_pooling2d (MaxPooling2D)	(None, 24, 24, 256)	0
conv2d_2 (Conv2D)	(None, 24, 24, 256)	590,080
batch_normalization_2 (BatchNormalization)	(None, 24, 24, 256)	1,024
conv2d_3 (Conv2D)	(None, 24, 24, 256)	65,792
batch_normalization_3 (BatchNormalization)	(None, 24, 24, 256)	1,024
conv2d_4 (Conv2D)	(None, 24, 24, 256)	65,792
batch_normalization_4 (BatchNormalization)	(None, 24, 24, 256)	1,024
conv2d_5 (Conv2D)	(None, 24, 24, 512)	1,180,160
batch_normalization_5 (BatchNormalization)	(None, 24, 24, 512)	2,048
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 512)	0
conv2d_6 (Conv2D)	(None, 12, 12, 512)	2,359,808
batch_normalization_6 (BatchNormalization)	(None, 12, 12, 512)	2,048
conv2d_7 (Conv2D)	(None, 12, 12, 512)	2,359,808
batch_normalization_7 (BatchNormalization)	(None, 12, 12, 512)	2,048
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 512)	0
conv2d_8 (Conv2D)	(None, 6, 6, 512)	2,359,808
batch_normalization_8 (BatchNormalization)	(None, 6, 6, 512)	2,048
max_pooling2d_3 (MaxPooling2D)	(None, 3, 3, 512)	0
flatten (Flatten)	(None, 4608)	0
dense (Dense)	(None, 1024)	4,719,616
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 1024)	1,049,600
dropout_1 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 4)	4,100

Total params: 15,611,526 (59.55 MB)
Trainable params: 15,605,124 (59.53 MB)
Non-trainable params: 6,400 (25.00 KB)
Optimizer params: 2 (12.00 B)

Convolution layer: -

In the above summary we can see that in first convolution layer when we have 128 filters, the output shape (number of features) becomes 73 by 73 due to strides of 3 by 3 and the number of parameters is 24,706 which can be calculated in this way

Parameters = (filter_size * filter_size * input_channels + bias) * num_filters

$$(8 \times 8 \times 3 + 1) \times 128 = 24,704$$

So, the similar calculation is applied for subsequent convolution layers.

Batch Normalization Layers: -

- It is used to normalize the pixels before moving on to the next layer and adds learnable scaling and shifting parameters.
- Parameters are calculated in this way = *2** (number of filters) = $2 \times 128 = 512$

Max Pooling layers: -

Max pooling layers are used to extract only the important features. As in above summary you can see that when we apply max pooling layers than the number of features become almost half. There is no trainable parameters as well only it performs down sampling.

Fully Connected Layers (Dense): -

These layers connect all neurons to form a classification decision.

The number of parameters in these layers are calculated as follows

Parameters = (input neurons * output neurons) + bias

- $4608 \times 1024 + 1024 = 4,719,6164608$

Dropout Layers (Dropout): -

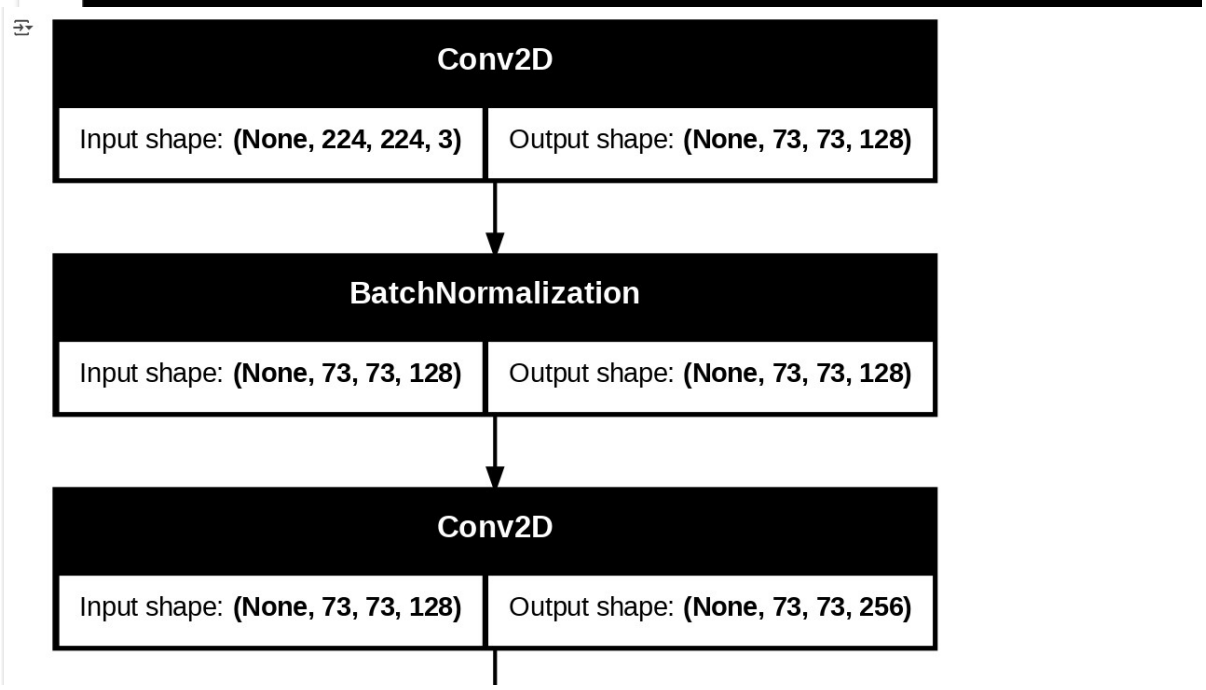
This layer is used to prevent overfitting by randomly deactivating neurons.

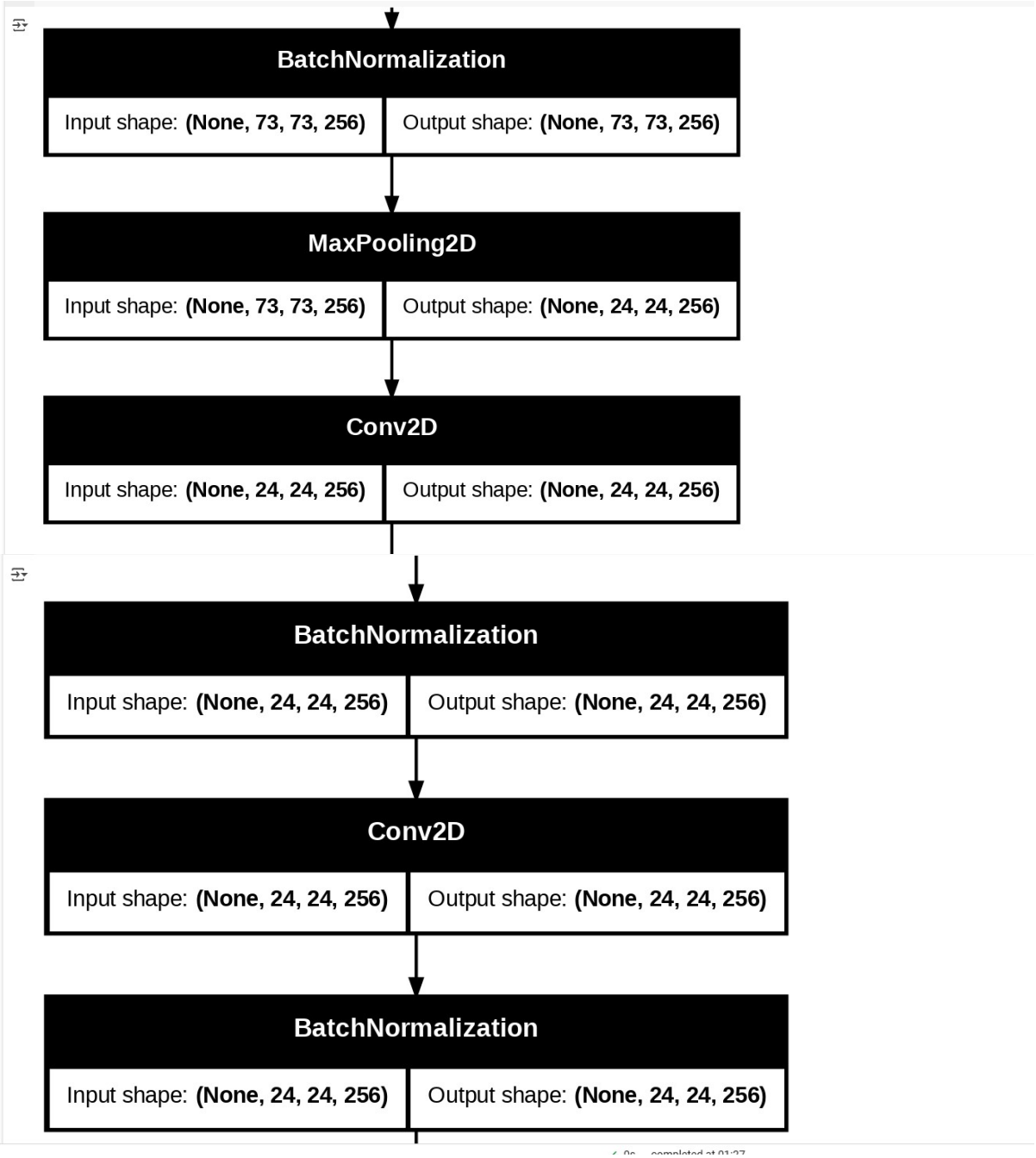
Output Layer: -

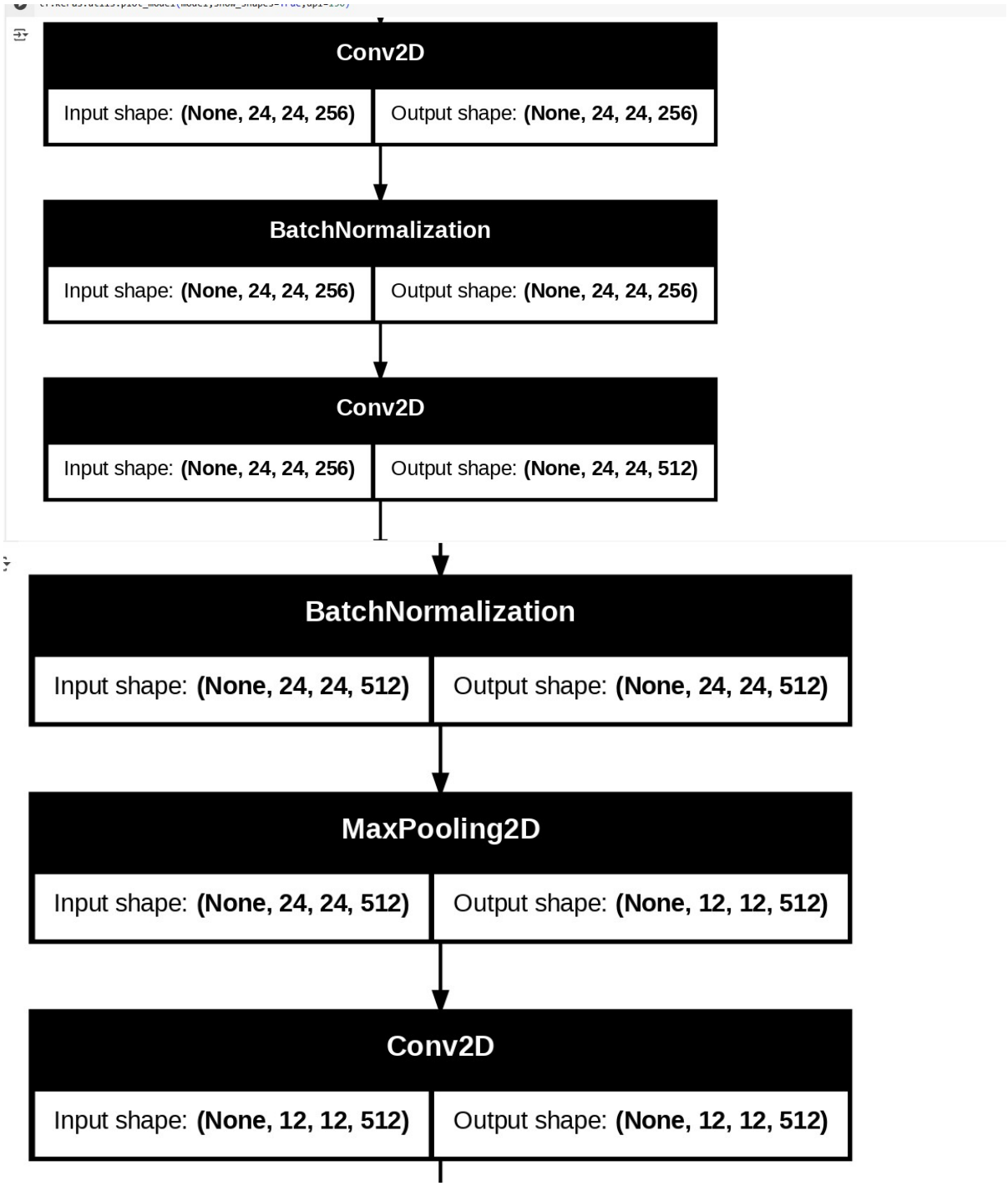
At the end i use final classification layer with 4 output neurons (for 4 Alzheimer categories) and uses **Softmax activation** to output probabilities.

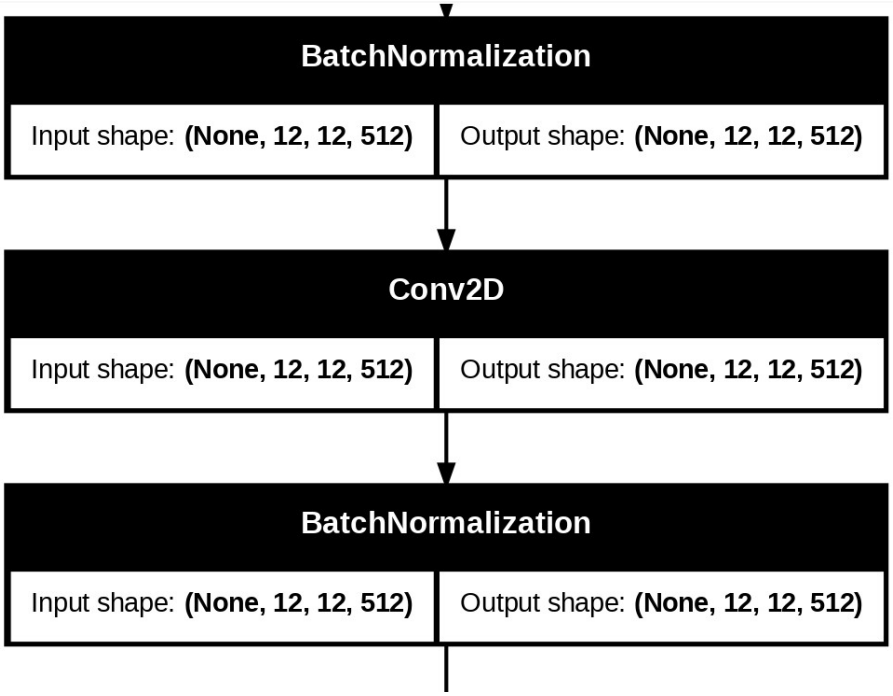
Visualizing the architecture of model: -

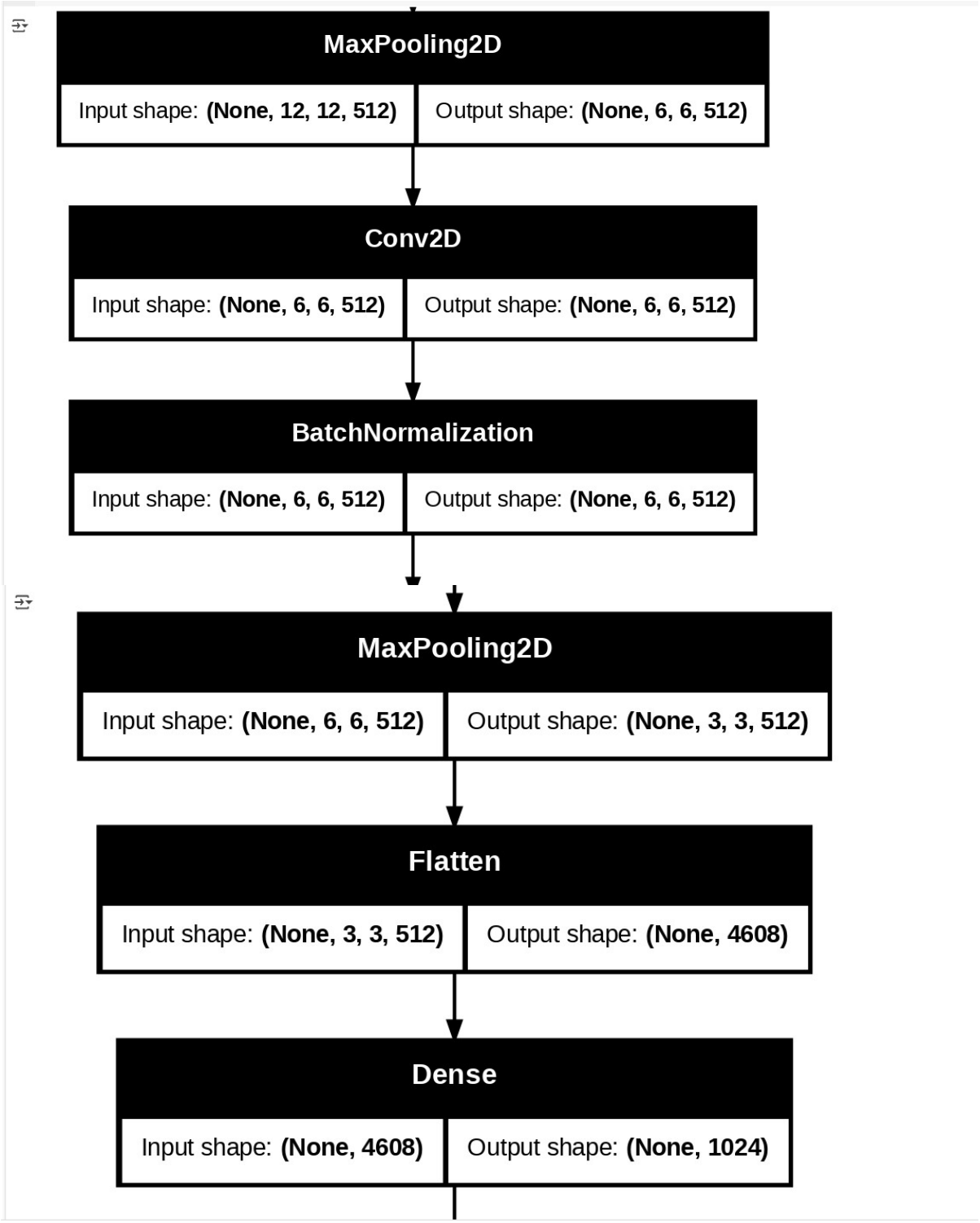
```
tf.keras.utils.plot_model(model,show_shapes=True,dpi=150)
```




BatchNormalization
Input shape: (None, 73, 73, 256)
Output shape: (None, 73, 73, 256)
MaxPooling2D
Input shape: (None, 73, 73, 256)
Output shape: (None, 24, 24, 256)
Conv2D
Input shape: (None, 24, 24, 256)
Output shape: (None, 24, 24, 256)
BatchNormalization
Input shape: (None, 24, 24, 256)
Output shape: (None, 24, 24, 256)
Conv2D
Input shape: (None, 24, 24, 256)
Output shape: (None, 24, 24, 256)
BatchNormalization
Input shape: (None, 24, 24, 256)
Output shape: (None, 24, 24, 256)


Conv2D
Input shape: (None, 24, 24, 256)
Output shape: (None, 24, 24, 256)
BatchNormalization
Input shape: (None, 24, 24, 256)
Output shape: (None, 24, 24, 256)
Conv2D
Input shape: (None, 24, 24, 256)
Output shape: (None, 24, 24, 512)
BatchNormalization
Input shape: (None, 24, 24, 512)
Output shape: (None, 24, 24, 512)
MaxPooling2D
Input shape: (None, 24, 24, 512)
Output shape: (None, 12, 12, 512)
Conv2D
Input shape: (None, 12, 12, 512)
Output shape: (None, 12, 12, 512)


BatchNormalization
Input shape: (None, 12, 12, 512)
Output shape: (None, 12, 12, 512)
Conv2D
Input shape: (None, 12, 12, 512)
Output shape: (None, 12, 12, 512)
BatchNormalization
Input shape: (None, 12, 12, 512)
Output shape: (None, 12, 12, 512)


MaxPooling2D
Input shape: (None, 12, 12, 512)
Output shape: (None, 6, 6, 512)
Conv2D
Input shape: (None, 6, 6, 512)
Output shape: (None, 6, 6, 512)
BatchNormalization
Input shape: (None, 6, 6, 512)
Output shape: (None, 6, 6, 512)
MaxPooling2D
Input shape: (None, 6, 6, 512)
Output shape: (None, 3, 3, 512)
Flatten
Input shape: (None, 3, 3, 512)
Output shape: (None, 4608)
Dense
Input shape: (None, 4608)
Output shape: (None, 1024)

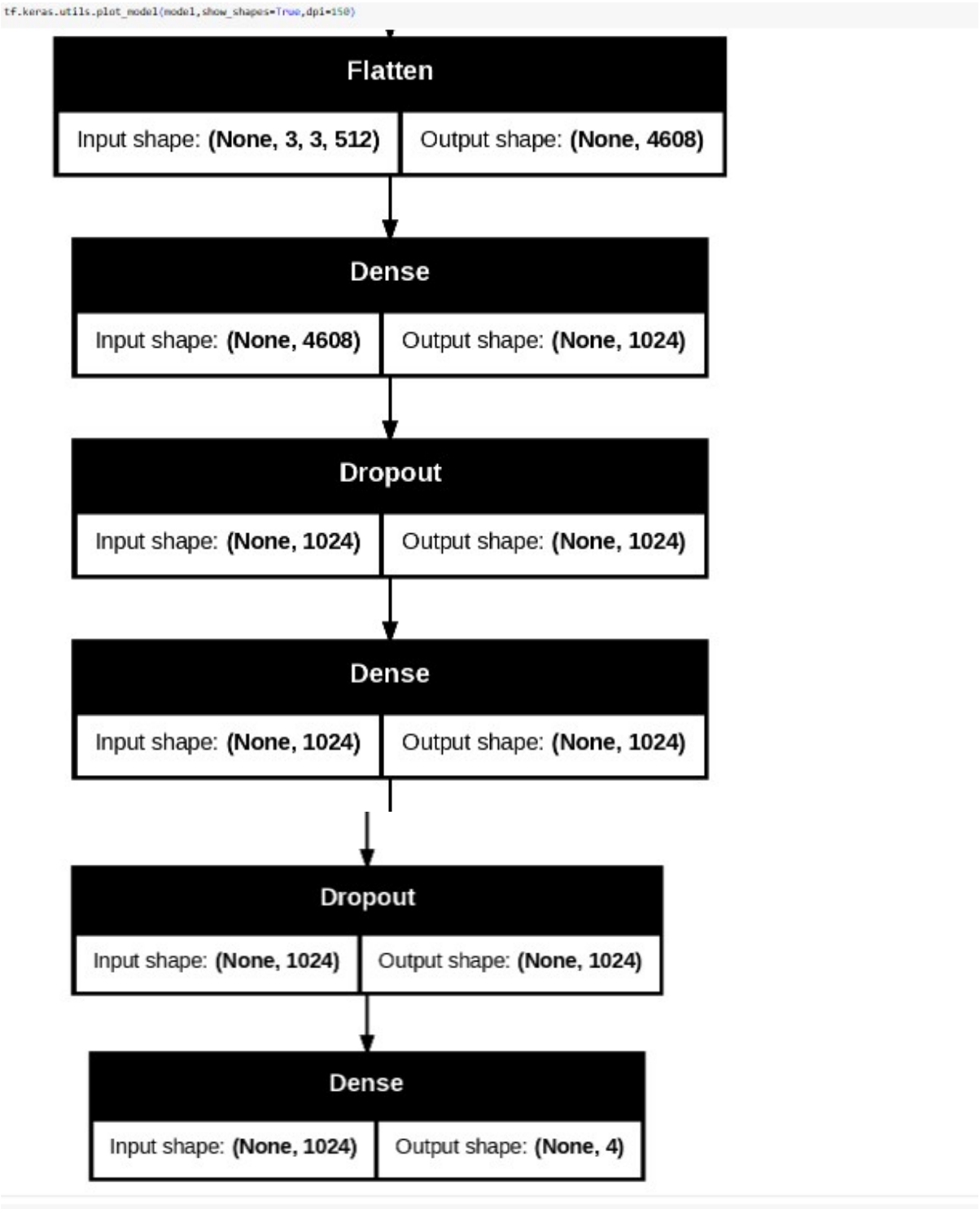


Figure 4-1 Visualizing the architecture of model:

Model Compilation: -

After defining the architecture of CNN, the model was compiled using the SGD as the optimization algorithm. I selected categorical cross-entropy as a loss function because the classification task involves multiple categories like mild dementia, moderate dementia, very mild dementia and non-dementia. Then I use accuracy as the primary metric to evaluate the model performance.

The compilation of model consists of these serval parameters

- **Loss Function:** Categorical Cross-Entropy
- **Optimizer:** Stochastic Gradient Descent (SGD)
- **Learning Rate:** 0.001
- **Evaluation Metric:** Accuracy

```
[ ]  model.compile(
         loss='categorical_crossentropy',
         optimizer=tf.optimizers.SGD(learning_rate=0.001),
         metrics=['accuracy']
     )
```

Model training: -

Then I trained the complied CNN model using the training dataset with a total of 20 epochs. This training process contains mini-batch gradient descent, where batches of images are processed in each iteration to update model parameters.

I use validation dataset to monitor the model's performance and detecting potential overfitting. I set the validation accuracy to 1, meaning that the model was evaluated on the validation set after each epoch.

The training of model contains following parameters

- **Training Data:** train dataset
- **Number of Epochs:** 20
- **Validation Data:** val dataset
- **Validation Frequency:** 1 (Validation performed after every epoch)

```
[ ] # Pass class weights to the fit method
    history = model.fit(
        train,
        epochs=20,
        validation_data=val,
        validation_freq=1
    )
```

```
Epoch 1/20
595/595 ━━━━━━━━━━━━━━━━━━━━ 138s 195ms/step - accuracy: 0.3310 - loss: 1.8228 - val_accuracy: 0.4807 - val_loss: 1.1351
Epoch 2/20
595/595 ━━━━━━━━━━━━━━━━━━━━ 91s 152ms/step - accuracy: 0.4869 - loss: 1.1334 - val_accuracy: 0.5904 - val_loss: 0.9351
Epoch 3/20
595/595 ━━━━━━━━━━━━━━━━━━━━ 92s 155ms/step - accuracy: 0.5639 - loss: 0.9579 - val_accuracy: 0.6330 - val_loss: 0.8203
Epoch 4/20
595/595 ━━━━━━━━━━━━━━━━━━━━ 92s 155ms/step - accuracy: 0.5950 - loss: 0.8705 - val_accuracy: 0.6393 - val_loss: 0.7818
Epoch 5/20
595/595 ━━━━━━━━━━━━━━━━━━━━ 92s 155ms/step - accuracy: 0.6332 - loss: 0.7972 - val_accuracy: 0.6759 - val_loss: 0.7154
Epoch 6/20
595/595 ━━━━━━━━━━━━━━━━━━━━ 93s 157ms/step - accuracy: 0.6550 - loss: 0.7407 - val_accuracy: 0.6898 - val_loss: 0.6879
Epoch 7/20
595/595 ━━━━━━━━━━━━━━━━━━━━ 94s 157ms/step - accuracy: 0.6951 - loss: 0.6711 - val_accuracy: 0.6978 - val_loss: 0.6502
Epoch 8/20
595/595 ━━━━━━━━━━━━━━━━━━━━ 91s 153ms/step - accuracy: 0.7262 - loss: 0.6154 - val_accuracy: 0.7442 - val_loss: 0.5803
Epoch 9/20
595/595 ━━━━━━━━━━━━━━━━━━━━ 143s 154ms/step - accuracy: 0.7599 - loss: 0.5480 - val_accuracy: 0.7619 - val_loss: 0.5471
Epoch 10/20
595/595 ━━━━━━━━━━━━━━━━━━━━ 93s 156ms/step - accuracy: 0.7945 - loss: 0.4883 - val_accuracy: 0.7257 - val_loss: 0.6136
Epoch 11/20
595/595 ━━━━━━━━━━━━━━━━━━━━ 93s 156ms/step - accuracy: 0.8275 - loss: 0.4212 - val_accuracy: 0.8094 - val_loss: 0.4569
Epoch 12/20
595/595 ━━━━━━━━━━━━━━━━━━━━ 91s 153ms/step - accuracy: 0.8536 - loss: 0.3578 - val_accuracy: 0.7694 - val_loss: 0.5531
Epoch 13/20
595/595 ━━━━━━━━━━━━━━━━━━━━ 91s 153ms/step - accuracy: 0.8800 - loss: 0.2979 - val_accuracy: 0.8174 - val_loss: 0.4677
Epoch 14/20
595/595 ━━━━━━━━━━━━━━━━━━━━ 91s 153ms/step - accuracy: 0.9089 - loss: 0.2365 - val_accuracy: 0.8468 - val_loss: 0.3787
Epoch 15/20
595/595 ━━━━━━━━━━━━━━━━━━━━ 91s 152ms/step - accuracy: 0.9185 - loss: 0.2120 - val_accuracy: 0.8468 - val_loss: 0.3952
Epoch 16/20
595/595 ━━━━━━━━━━━━━━━━━━━━ 91s 152ms/step - accuracy: 0.9383 - loss: 0.1651 - val_accuracy: 0.8613 - val_loss: 0.3827
Epoch 17/20
595/595 ━━━━━━━━━━━━━━━━━━━━ 91s 153ms/step - accuracy: 0.9552 - loss: 0.1262 - val_accuracy: 0.8485 - val_loss: 0.4452
Epoch 18/20
```

Learning Curves: -

Here are the learning curves for training data and validation data to check whether the model is overfitting or not.

```
[ ] import matplotlib.pyplot as plt

    def plot_loss_curves(history):
        # Plot training & validation accuracy values
        plt.figure(figsize=(12, 4))

        plt.subplot(1, 2, 1)
        plt.plot(history.history['accuracy'])
        plt.plot(history.history['val_accuracy'])
        plt.title('Model accuracy')
        plt.ylabel('Accuracy')
        plt.xlabel('Epoch')
        plt.legend(['Train', 'Validation'], loc='upper left')

        # Plot training & validation loss values
        plt.subplot(1, 2, 2)
        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
        plt.title('Model loss')
        plt.ylabel('Loss')
        plt.xlabel('Epoch')
        plt.legend(['Train', 'Validation'], loc='upper left')

        plt.show()
```

```
[ ] plot_loss_curves(history)
```

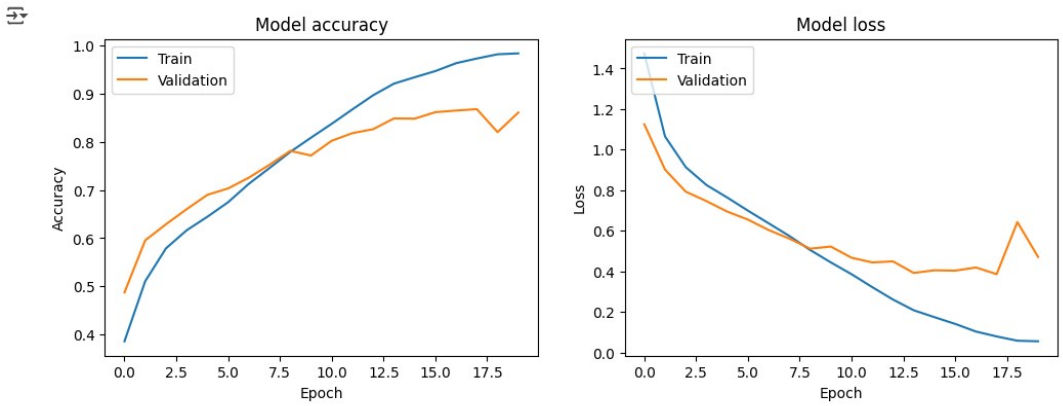


Figure 4-2 Learning Curves

Here in the above graphs, we can see that the difference between validation accuracy and training accuracy is not more than 5%. So, its mean that model is not overfitting. And the same thing goes for training loss and validation loss that the difference between them is not more than 5%. So, it means our model is not overfitted.

Train accuracy and test accuracy: -

Here is the code for displaying the train accuracy and test accuracy.

```
[ ] valid_loss, valid_accuracy = model.evaluate(val)

    print(f'\nTrain loss: {valid_loss:.2f}')
    print(f'Train Accuracy: {valid_accuracy*100:.2f} %')
```

```
149/149 ━━━━━━━━━━━━━━━━━━━━ 11s 74ms/step - accuracy: 0.8703 - loss: 0.4144

Train loss: 0.42
Train Accuracy: 87.07 %
```

```
[ ] loss, accuracy =model.evaluate(test)

    print(f'\nTest loss: {loss:.2f} ')
    print(f'Test Accuracy: {accuracy*100:.2f} %')
```

```
319/319 ━━━━━━━━━━━━━━━━━━━━ 24s 76ms/step - accuracy: 0.8626 - loss: 0.4176

Test loss: 0.42
Test Accuracy: 86.32 %
```

In the above we can see that the difference between the train accuracy and test accuracy is not much big only a difference of 1%, which suggests that the model is not overfitting, and it is generalizing well to unseen data.

Prediction Accuracy on Test Set: -

```
from sklearn.metrics import accuracy_score
# Get predictions on the test set
test_predictions = model.predict(test, verbose=1)

# Convert the predicted probabilities to class labels
predicted_class_labels = np.argmax(test_predictions, axis=1)

# Get the actual labels from the test set
true_class_labels = test.classes  # .classes contains the true class labels

# Calculate the accuracy using sklearn's accuracy_score
accuracy = accuracy_score(true_class_labels, predicted_class_labels)

# Print the accuracy
print(f"Prediction Accuracy on Test Set: {accuracy * 100:.2f}%")
```

```
319/319 ━━━━━━━━━━━━━━━━━━━━ 22s 65ms/step
Prediction Accuracy on Test Set: 85.27%
```

From above we can see that the prediction on test set is 85% which is quite well, and the model seems reliable to use in real world applications because of this result .

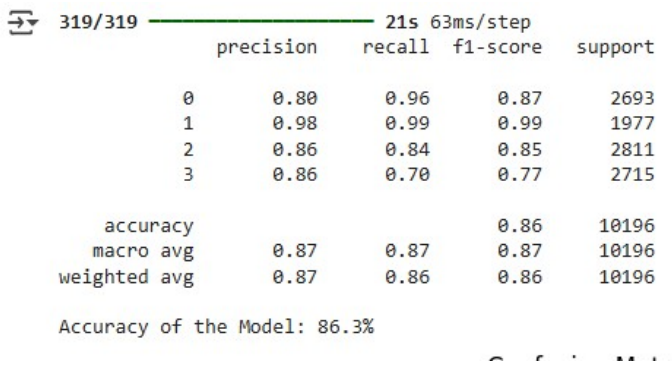
Confusion matrix and classification report: -

Here is the code for generating confusion matrix and classification report as

```
[ ] import numpy as np
    from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
    import seaborn as sns
    import matplotlib.pyplot as plt

    # Get predictions on the test set
    test_predictions = model.predict(test, verbose=1)

    # Convert predicted probabilities to class labels
    predicted_class_labels = np.argmax(test_predictions, axis=1)

    # Get the actual labels from the test set
    y_test = test.classes  # .classes contains the true class labels

    # Generate confusion matrix
    cm = confusion_matrix(y_test, predicted_class_labels)

    # Print classification report
    print(classification_report(y_test, predicted_class_labels))

    # Print accuracy of the model
    accuracy = accuracy_score(y_test, predicted_class_labels)
    print(f"Accuracy of the Model: {accuracy * 100:.1f}%")

    # Plot confusion matrix
    plt.figure(figsize=(10, 7))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=test.class_indices.keys(), yticklabels=test.class_indices.keys())
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()
```

Classification report: -

The classification report of our deep learning model was assessed using precision, recall, and F1-score metrics in the below table



```
319/319 ━━━━━━━━━━━━━━━━━━━━ 21s 63ms/step
              precision    recall  f1-score   support

           0       0.80      0.96      0.87      2693
           1       0.98      0.99      0.99      1977
           2       0.86      0.84      0.85      2811
           3       0.86      0.70      0.77      2715

    accuracy                           0.86     10196
   macro avg       0.87      0.87      0.87     10196
weighted avg       0.87      0.86      0.86     10196

Accuracy of the Model: 86.3%
```

Figure 4-3 Classification report

Overall, the model achieved an accuracy of 86%, indicating its effectiveness in differentiating the different categories of Alzheimer disease.

Analysis of Results: -

- The model exhibits high recall for mild dementia (0.96) and moderate dementia (0.99).
- However, the very mild dementia (class 3) has a lower recall of 0.70, which suggest that model misclassifies some cases from this category.
- The high precision of moderate dementia 0.98 indicates a lower false positive.
- Overall, the classification report suggest that the model performs well but needs a little bit improvement in very mild dementia class.

Confusion matrix: -

The performance of proposed deep learning model was further evaluated using a confusion matrix.

Observations from the Confusion Matrix

- The model exhibits **strong classification performance** for the **Mild Demented** and **Moderate Demented** classes, with **2,581** and **1,967** correctly classified instances, respectively.
- **Non-Demented** cases also show high accuracy, with **2,349** correctly classified samples. However, **255 instances were misclassified as Very Mild Demented**.
- The **Very Mild Demented category** has the **highest misclassification rate**, with **437 samples misclassified as Mild Demented** and **354 misclassified as Non-Demented**. This suggests some difficulty in differentiating **Very Mild Demented cases** from other categories.
- Misclassification is **minimal for the Moderate Demented class**, with only **9 samples misclassified as Mild Demented** and **1 sample misclassified as Very Mild Demented**, indicating the model's strong ability to distinguish this class.

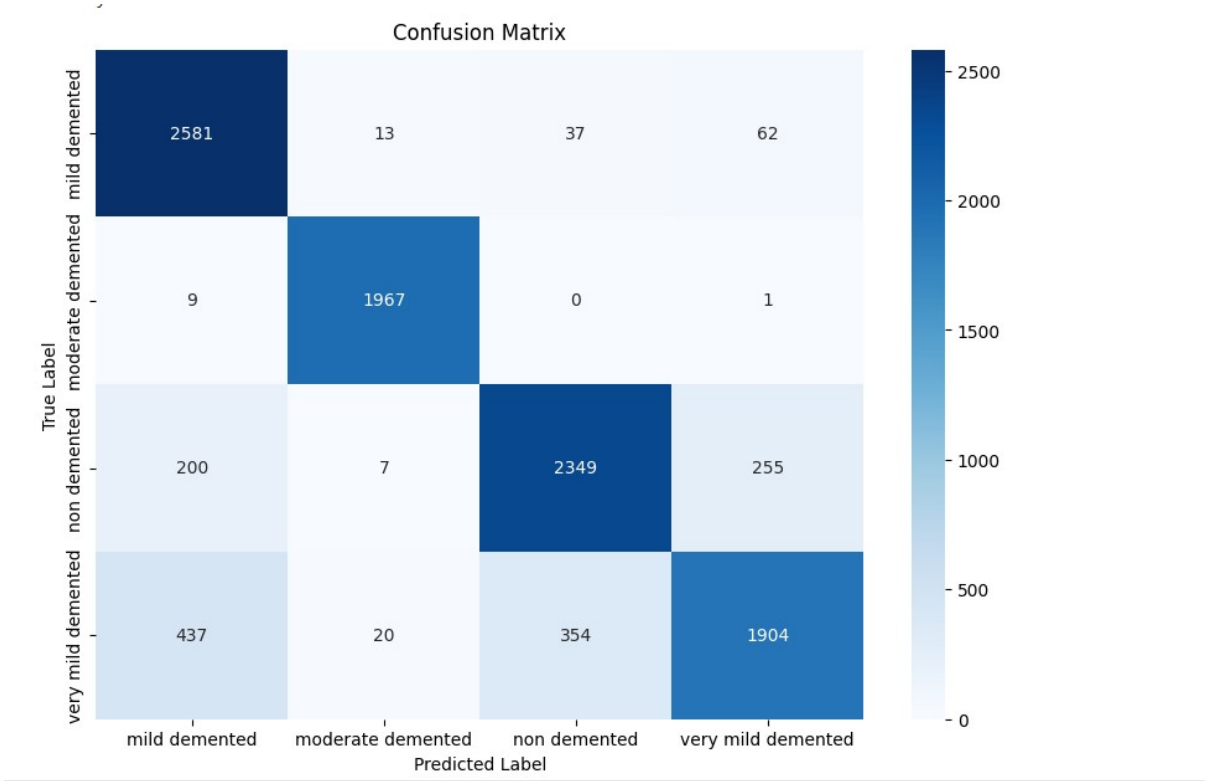


Figure 4-4 Confusion matrix

Saving the model and preprocessing: -

Then I saved the model in notebook and as well as in google drive. So that even if the runtime vanishes still, we can retrieve it back.

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import joblib

# Save the preprocessing function (used in ImageDataGenerator)
preprocessing_function = tf.keras.applications.mobilenet_v2.preprocess_input
joblib.dump(preprocessing_function, 'preprocessing_function.pkl')

# Save the trained model
model.save('alzheimer_model.keras')
```

```
[ ] from google.colab import drive

    # Mount Google Drive
    drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[ ] import os

    # Define the path where the model will be saved
    model_save_path = '/content/drive/My Drive/models5/'

    # Create the directory if it doesn't exist
    os.makedirs(model_save_path, exist_ok=True)

    # Now define the file path to save the model
    model_save_file = os.path.join(model_save_path, 'model_name.keras')

    # Save the model in the Keras format
    model.save(model_save_file)

    print(f"Model saved at {model_save_file}")
```

```
Model saved at /content/drive/My Drive/models5/model_name.keras
```

```
[ ] # Define the path where the model is saved
    model_save_path = '/content/drive/My Drive/models4/model_name.keras'

    # Load the model
    loaded_model = tf.keras.models.load_model(model_save_path)

    print("✅ Model loaded successfully!")
```

```
✅ Model loaded successfully!
```

4.2.1 A description of how the developed model was implemented to meet the requirements

Doctor-Assisted Diagnosis System: -

At the end we have created a doctor-assisted diagnosis system. When the doctor wants to predict to which category of Alzheimer disease the patient has, he just uploads his MRI scan to this system and this system will predict the stage of Alzheimer disease that patient have.

Here is the explanation that how this system works.

Installing important libraries

```
[ ] pip install lime

Collecting lime
  Downloading lime-0.2.0.1.tar.gz (275 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 275.7/275.7 kB 9.5 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (from lime) (3.10.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from lime) (1.26.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from lime) (1.13.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from lime) (4.67.1)
Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.11/dist-packages (from lime) (1.6.1)
Requirement already satisfied: scikit-image>=0.12 in /usr/local/lib/python3.11/dist-packages (from lime) (0.25.2)
Requirement already satisfied: networkx>=3.0 in /usr/local/lib/python3.11/dist-packages (from scikit-image>=0.12->lime) (3.4.2)
Requirement already satisfied: pillow>=10.1 in /usr/local/lib/python3.11/dist-packages (from scikit-image>=0.12->lime) (11.1.0)
Requirement already satisfied: imageio!=2.35.0,>=2.33 in /usr/local/lib/python3.11/dist-packages (from scikit-image>=0.12->lime) (2.37.0)
Requirement already satisfied: tifffile>=2022.8.12 in /usr/local/lib/python3.11/dist-packages (from scikit-image>=0.12->lime) (2025.2.18)
Requirement already satisfied: packaging>=21 in /usr/local/lib/python3.11/dist-packages (from scikit-image>=0.12->lime) (24.2)
Requirement already satisfied: lazy-loader>=0.4 in /usr/local/lib/python3.11/dist-packages (from scikit-image>=0.12->lime) (0.4)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=0.18->lime) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=0.18->lime) (3.5.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->lime) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib->lime) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->lime) (4.56.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->lime) (1.4.8)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->lime) (3.2.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib->lime) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib->lime) (1.17.0)
Building wheels for collected packages: lime
  Building wheel for lime (setup.py) ... done
  Created wheel for lime: filename=lime-0.2.0.1-py3-none-any.whl size=283834 sha256=341296cdeeccea1e698338e97b6fe1fbda13505579cabe4ecec1384b27f54cb7
  Stored in directory: /root/.cache/pip/wheels/85/fa/a3/9c2d44c9f3cd77cf4e533b58900b2bf4487f2a17e8ec212a3d
Successfully built lime
Installing collected packages: lime
Successfully installed lime-0.2.0.1

[ ] pip install scikit-image

Requirement already satisfied: scikit-image in /usr/local/lib/python3.11/dist-packages (0.25.2)
Requirement already satisfied: numpy>=1.24 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (1.26.4)
Requirement already satisfied: scipy>=1.11.4 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (1.13.1)
```

Model Loading and Preprocessing: -

The system starts with loading the pre-trained model and a preprocessing function. This preprocessing function ensures that the new MRI image will also go through the same transformation as the training data to maintain consistency in model prediction.

Image preprocessing: -

Before making the prediction, the system will perform following operations.

- **Resizing MRI** scan to 224 × 224 pixels (the required input size of the CNN model).
- **Color Conversion** (ensuring the MRI Scan is in RGB format).
- **Normalization and Scaling** (applying the same transformations as during training).
- **Batch Dimension Addition** (to match the model's input shape).

Prediction mechanisms: -

Once the MRI scan is pre-processed, the CNN model makes a prediction using SoftMax function, which also outputs probability scores for each class.

Explainable AI: -

This system has lime which act as a explainable AI for this system. LIME highlights the region of MRI scan which contribute much for making decision by the model.

- It Generates **perturbed versions** of the input MRI scan.
- It Evaluates how the model's predictions change with **local variations** in the MRI scan.
- It Displays **highlighted regions** that influenced the classification decision using **LIME visualizations**.

Code: -

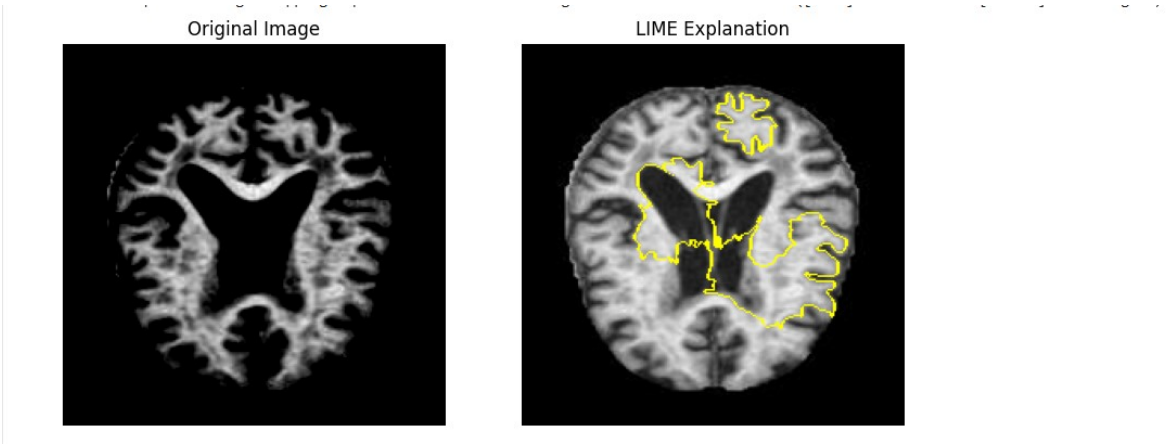
```
import tensorflow as tf
from tensorflow import keras
import numpy as np
from PIL import Image
import joblib
import lime
from lime import lime_image
from skimage.segmentation import mark_boundaries
import matplotlib.pyplot as plt

# Load the saved preprocessing function
# Ensure the preprocessing function is saved correctly during training
preprocessing_function = joblib.load('preprocessing_function.pkl')

# Load the saved model
model = keras.models.load_model('alzheimer_model.keras')

# Define class labels (modify based on your dataset)
class_labels = ["Mild Dementia", "Moderate Dementia", "Non Dementia", "Very Mild Dementia"]

# Function to preprocess the uploaded image
def preprocess_image(image_path, target_size=(224, 224)):
    """
    Preprocesses the image to match the model's input requirements.
    - Resizes the image to the target size.
    - Applies the same preprocessing as during training.
    - Adds a batch dimension.
    """
    # Load the image
    img = Image.open(image_path)

    # Convert to RGB if the image is grayscale
    if img.mode != 'RGB':
        img = img.convert('RGB')

    # Resize the image to match the model's input shape
    img = img.resize(target_size)
```

```
    # Convert to numpy array
    img_array = np.array(img)

    # Apply the same preprocessing as during training
    img_array = preprocessing_function(img_array)

    # Add batch dimension
    img_array = np.expand_dims(img_array, axis=0)

    return img_array

# Function to predict the category of the image
def predict_image(image_path):
    """
    Predicts the disease stage for the given image.
    - Preprocesses the image.
    - Makes a prediction using the model.
    - Returns the predicted class and confidence score.
    """
    # Preprocess the image
    img_array = preprocess_image(image_path)

    # Make a prediction
    predictions = model.predict(img_array)

    # Get the predicted class index
    predicted_class_index = np.argmax(predictions, axis=1)[0]

    # Get the predicted class label
    predicted_class_label = class_labels[predicted_class_index]

    # Get the confidence score
    confidence = np.max(predictions)

    return predicted_class_label, confidence, img_array[0]  # Return the preprocessed image for LIME
```

```python
# Function to explain the prediction using LIME
def explain_with_lime(image, model, class_labels, top_labels=5, num_samples=1000):
    """
    Explains the model's prediction using LIME.
    - Generates a LIME explanation for the image.
    - Visualizes the explanation.
    """
    # Create a LIME explainer
    explainer = lime_image.LimeImageExplainer()

    # Explain the model's prediction
    explanation = explainer.explain_instance(
        image,
        model.predict,
        top_labels=top_labels,
        hide_color=0,
        num_samples=num_samples
    )

    # Get the explanation for the top predicted class
    temp, mask = explanation.get_image_and_mask(
        explanation.top_labels[0],
        positive_only=True,
        num_features=5,
        hide_rest=False
    )

    # Display the original image and the explanation
    plt.figure(figsize=(10, 5))

    # Original Image
    plt.subplot(1, 2, 1)
    plt.imshow(image)
    plt.title("Original Image")
    plt.axis('off')

    # LIME Explanation
    plt.subplot(1, 2, 2)
    plt.imshow(mark_boundaries(temp / 2 + 0.5, mask))
    plt.title("LIME Explanation")
    plt.axis('off')

    plt.show()

# Upload and predict
image_path = input("Enter the path to the MRI scan: ")  # Example: "test_mri.jpg"
predicted_class, confidence, preprocessed_image = predict_image(image_path)

# Display the result
print(f"Predicted Disease Stage: {predicted_class}")
print(f"Confidence: {confidence:.2f}")

# Explain the prediction using LIME
explain_with_lime(preprocessed_image, model, class_labels)
```

Output: -

```
Enter the path to the MRI scan: /content/AugmentedAlzheimerDataset/OriginalDataset/VeryMildDemented/verymildDem1489.jpg
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 895ms/step
Predicted Disease Stage: Very Mild Dementia
Confidence: 0.99
100% ████████████████████ 1000/1000 [00:18<00:00, 62.13it/s]
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 119ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 122ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step
```



4.2.2 Justification of the Methods and Tools Adopted

To build an accurate and efficient model for classifying Alzheimer's disease from MRI scan, a combination of deep learning techniques and well-established tools were selected. Here's why:

CNN: -

As the dataset was of MRI scans which contains certain intricate spatial features, so CNN were the natural choice for this classification task. The use of multiple convolution layers, pooling, and dense layers helped the model learn both low-level and high-level features necessary for precise classification.

ReLU Activation Function: -

ReLU (Rectified Linear Unit) was used after each convolutional layer to introduce non-linearity. This helps the model learn complex patterns and speeds up training by reducing the risk of vanishing gradients.

Batch Normalization:
After each convolutional layer, batch normalization was added to help the model train faster and more reliably. It made the learning process more stable and helped the model handle variations in image brightness and contrast more effectively.

Max Pooling:
Max pooling layers were used to reduce the size of the feature maps, which helped in lowering the computational load without losing important information. It kept the most significant features while making the model faster and lighter.

Dropout:
To avoid overfitting, dropout layers were included in the fully connected parts of the network. During training, dropout randomly switches off certain neurons, forcing the model to learn more general patterns that perform better on new, unseen data.

Softmax Activation & Categorical Cross-Entropy Loss:
Since the model needed to classify MRI scans into one of four stages of Alzheimer's disease, the softmax function was used in the output layer. This function gives the probability of the scan belonging to each category. For the loss function, categorical cross-entropy was used, which is well-suited for multi-class classification problems like this one.

SGD Optimizer:
The Stochastic Gradient Descent (SGD) algorithm was used to train the model. It's a straightforward and effective method for updating model weights. When combined with the right learning rate, it helped the model learn smoothly and efficiently.

Tools That Made It Happen:

- **TensorFlow & Keras:**
  These were the core tools for building and training the model. They provided a simple and flexible framework to design and experiment with deep learning architectures.
- **Matplotlib & Seaborn:**
  Used to plot training/validation accuracy, loss graphs, and confusion matrices—these visualizations helped understand how the model was performing and where improvements were needed.
- **LIME (Local Interpretable Model-Agnostic Explanations):**
  To make the system more transparent and explainable, LIME was used. It helped highlight which parts of an MRI image influenced the model's decision, which is especially important in medical applications.
- **Google Colab & Drive:**
  Google Colab provided a free, cloud-based GPU environment to train the model faster. The trained model was saved to Google Drive to make sure progress wasn't lost and could be easily restored when needed.

4.2.3 Results and Findings

- **Overall Accuracy:** 86%
- The model performs especially well on Mild and Moderate Dementia classes.
- Performance on Very Mild Dementia is lower, suggesting the need for more data.

4.2.4 Conclusion

The deep learning model developed in this project has proven to be effective in classifying MRI scans into four stages of Alzheimer's disease. With 86% accuracy and strong generalization, it shows real-world potential. The addition of explainable AI (LIME) also enhances trust in the system, making it a valuable tool for supporting doctors in accurately diagnosing the different stages of Alzheimer's disease.

Chapter 5: Critical Evaluation

This chapter is further divided into two parts.

- Critical Evaluation of Clinical dataset.
- Critical Evaluation of MRI scans dataset.

5.1 Critical Evaluation of clinical dataset: -

The primary objective of this project was to develop an accurate, robust and explainable machine learning model to assist in diagnosing Alzheimer's disease using clinical dataset. The model needs to be generalized well to unseen data and offer high predictive accuracy and provide insights to clinicians as well.

These goals were successfully acheived as

- The **Random Forest Classifier**, chosen after model comparison and hyperparameter tuning via GridCVSearch, achieved a **95.12% accuracy** on the test set.
- Using **RFECV**, the model was able to identify the top 6 most informative features out of 35, and these 6 features were contributing to 95% of predictive accuracy.
- Integration of **LIME** added an interpretability layer, helping clinicians with transparency behind the predictions of AI box.
- The system design allowed doctors to input patient data and receive real-time diagnosis with an explanation, directly supporting clinical decision-making.

5.1.1 Review of the Plan and Deviations

The initial project plan involved applying multiple machine learning algorithms, performing feature selection, and tuning hyperparameters using GridSearchCV, followed by model evaluation on multiple metrics.

While the plan was largely followed, some deviations are described here:

- **Feature Reduction**: Initially, all 35 features were considered, but the accuracy was not satisfyable than experimentation with **RFECV** revealed that only 6 features were needed to maintain high accuracy.
- **Class Imbalance**: The issue of class imbalance emerged during data exploration. The dataset was moderately imbalance. Although this was not a part of the initial strategy, it was effectively addressed by using class_weight='balanced' in the model training.
- **Threshold Tuning**: Although the default threshold of 0.5 performed well, additional threshold tuning was explored to minimize **false negatives**, becasue in medical if a patient left undiagnosed than it will lead to some serious consequences.
- **Explainability (LIME)**: The use of LIME was not originally planned but was later added to make the model outputs more interpretable and clinically trustworthy. Because I have studied on google that doctor do not prefer to use AI models which do not have explainable AI attached to it.

5.1.2 Lessons Learned

These are some several key insights and lessons :

1. **Fewer Features Can Be More Powerful**: Feature importance doesn't always align with quantity as we have seen that only 6 features are contributing out of 35 features. Reducing the model to only the most relevant inputs not only enhanced accuracy but also made the system more interpretable and faster.
2. **Interpretability Is Crucial in Healthcare**: The addition of explainable AI tools like **LIME** proved vital for user trust, especially in clinical settings where decisions must be justified. Usually, doctors use only AI models that have explanation attached to it like LIME etc.
3. **Cross-Validation Prevents Overfitting**: The consistent results across training, validation, and test sets showed that cross-validation and proper tuning greatly improve model reliability.
4. **Early Detection of Class Imbalance Is Important**: Identifying and handling class imbalance early on avoids biased models and improves prediction for minority classes.
5. **Visualization Drives Understanding**: Tools like confusion matrices, ROC curves, and calibration plots provided a deeper understanding of the model's behaviour and performance.

5.2 Critical Evaluation of MRI scan: -

The main objective of this project was to make a reliable system which can classify different stages of Alzheimer disease using MRI scans. The system aimed to help doctors by providing quick support and reliable predictions to assist in diagnosis.

Looking at the outcome, the system achieves an accuracy of 86% with strong performance in identifying different stages. As there is a minimal difference in test accuracy and train accuracy which suggest that the model is generalizing well to an unseen data. Additionally, the use of LIME for explainability met the requirement for building trust in model predictions. Overall, this project successfully met its goal.

5.2.1 Review of the Plan and Deviation: -

Initially, the plan involved understanding the data set, do preprocessing, building a CNN model and train it on a balanced dataset. However, a few adjustments were made along the way:

Data processing: The original plan was to balance the data set before training the model but later realize that the data is very slightly imbalance and techniques like SMOTE etc is not required.

Model Complexity: The model ended up with more complex than initially planned, with deeper layers and additional techniques like batch normalization and drop out added to improve accuracy and prevent overfitting.

Explainability: The integration of explainable AI (LIME) was not the part of initial plan but added latter to improve transparency and make the model more trustworthy for real-world medical use.

5.2.3 Lesson Learned

Several lessons were gained throughout this project

- **Understanding the Data is Crucial:** Taking time to explore the dataset early helped avoid unnecessary preprocessing steps, such as dealing with class imbalance.
- **Model Explainability Matters:** In medical applications, it's not just about accuracy. Tools like LIME are important for helping healthcare professionals understand and trust AI-based decisions.
- **Regular Monitoring Helps:** Tracking metrics like validation accuracy, loss, and using visualization tools made it easier to catch and fix problems early, such as potential overfitting.
- **Flexibility is Key:** Being open to modifying the original plan based on results and challenges allowed the project to adapt and ultimately succeed.

Chapter 6 Conclusions and Future Work: -

6.1 Conclusion: -

In this project 2 different notebooks were created.

1st notebook was created to diagnosis whether the person has Alzheimer or not. This notebook uses the clinical + demographic data of patients and machine learning techniques were applied to detect whether a patient has Alzheimer or not.

2nd notebook was created to diagnosis at which stage of Alzheimer the patient is. This notebook uses the MRI scans of patients and CNN were applied on it to predict the stage of Alzheimer.

This project works in this way. Suppose a patient comes to doctor and shares his symptoms like clinical data + demographic data with the doctor and doctor will inputs all this information into our 1st notebook and runs the machine learning algorithm. If the model predicts that the patient has Alzheimer that doctor request an MRI scan from the patient. Upload that MRI scan to the 2nd notebook and runs the CNN model. This CNN model will predict at which stage of Alzheimer the patient is. In this way overall, this project aims to help doctors to make informed decisions about their patients using the miracles of machine learning and deep learning.

6.2 Future work

For future work, I would suggest making a user-friendly web interface in which the doctor input all the patient details and press enter. If the model predicts that the patient has disease than the system automatically requests an MRI scan to further predict at which stage the patient is. In this way we can come up with an easier way to diagnose and predict stagging of Alzheimer disease.

6.3 Possible improvements

- Expanding the dataset size, would improve model generalization and robustness.
- Alzheimer's is a progressive disease. Incorporating patient history and time-series data (e.g., symptom progression over time) could significantly improve early-stage detection.
- Using ensemble techniques like taking predictions from multiple models could further improve accuracy and reduce biases.
- The model should be tested in real world clinical environments with real patients to assess its practical effectiveness.

6.4 Reflection on Legal, Social, Ethical, and Professional Issues

Developing AI-driven medical tools carries several responsibilities:

- **Legal**: The dataset used in this project is publicly available from Kaggle and is anonymized, ensuring compliance with data privacy regulations like HIPAA and GDPR.
- **Ethical**: The system must be used as a **support tool**, not a replacement for clinical judgment. It's a clear disclaimer to avoid misuse or over-reliance.

- **Bias and Fairness**: Even though Kaggle datasets are typically curated, if they may still carry demographic or socioeconomic biases than I am not sure about this.
- **Transparency**: The inclusion of explainable AI (LIME) contributes to ethical transparency, allowing patients and doctors to understand model decisions and ensuring accountability.
- **Professional Responsibility**: As developers of this systems, I maintained up-to-date knowledge on best practices, document work clearly, and test models responsibly before deployment .

Chapter 7: References and Citations

7.1 Books

Han, J., Kamber, M., & Pei, J. (2012). *Data mining: Concepts and techniques* (3rd ed.) Elsevier.

Larose, D. T., & Larose, C. D. (2015). *Data mining and predictive analytics* (2nd ed.) John Wiley & Sons.

Tan, P.-N., Steinbach, M., & Kumar, V. (2014). *Introduction to data mining*. Pearson.

Leskovec, J., Rajaraman, A., & Ullman, J. D. (2014). *Mining of massive datasets* (2nd ed.). Cambridge University Press.

Zollanvari, A. (2023). *Machine learning with Python: Theory and implementation* (1st ed.) Springer.

Lee, W.-M. (2019). *Python machine learning* (1st ed.). Wiley.

7.2 Webpage: -

International, Alzheimer's Disease, et al. *World Alzheimer Report 2015: The Global Impact of Dementia: An Analysis of Prevalence, Incidence, Cost and Trends*. Sept. 2015. www.alzint.org, https://www.alzint.org/resource/world- alzheimer-report-2015/.