Exploration into Predicting Alzheimer’s Disease using Machine Learning Techniques

**Abstract**

Alzheimer’s disease is a progressive neurodegenerative disorder in which patients experience loss of memory, impaired cognitive function, and disrupt their daily activities. As the disease is closely associated with aging, the rise in Alzheimer’s cases is a cause of concern as it burdens healthcare and government resources. It also decreases quality of life as patients age. Therefore, it is important to make the detection of Alzheimer’s more convenient and faster. This paper outlines machine learning techniques that can be used to detect Alzheimer’s in patients using MRI images and brain morphometry. It makes use of classifiers while also utilizing Convolutional Neural Networks and Transformers.

**keywords:** machine learning, classifiers, Alzheimer’s, predict

# **Introduction**

In 1906, German psychiatrist, Alois Alzheimer, was the first to discover Alzheimer’s disease after his patient Auguste Deter passed away. She was riddled with sleep issues, memory loss, confusion, and cognitive impairment. Given that these symp- toms were atypical for a fifty-year-old, she was brought to the institute where Alzheimer worked for treatment [1]. After she passed away and an autopsy was conducted, his report noted the presence of distinctive plaques and neurofibrillary tangles in the brain histology [1].

While Alzheimer’s discovery did not gain much attention at the time, it has since become a pressing issue. Alzheimer’s is a progressive neurodegenerative disorder characterized by difficulties in memory, disturbances in language and other cognitive functions, changes in behaviors, and impairments in activities of daily living [2]. It is one of the most common causes of dementia, accounting for 75 percent of all dementia cases [2]. According to the Alzheimer’s Association, nearly 7 million Americans are currently living with Alzheimer’s and the number is predicted to double by 2050 [3]. Moreover, Alzheimer’s disease (AD) was the fifth leading cause of death among people aged 65 and older in 2021 [3].

The proportion of older people is expected to increase from 420 million in 2000 to nearly 1 billion by 2030, with the percentage of older people increasing from 7 percent to 12 percent [2]. This is worrying as the occurrence of Alzheimer’s disease is strongly associated with an increase in age. About 1 in 9 people age 65 and older have Alzheimer’s [3]. This will burden government and hospital resources, making it harder to cater to everyone suffering if sufficient resources are not allocated [2]. Health and long-term care costs for people suffering from dementia are expected to reach 360 billion dollars in 2024 and 1 trillion dollars by 2030 [3].

Alzheimer’s disease (AD) generally begins with mild cog- nitive impairment (MCI), a condition in which individuals experience a slight but noticeable decline in cognitive abilities, such as memory and thinking skills. Over time, MCI can progress into AD, which significantly impacts daily living by causing memory loss and difficulty performing simple tasks. Patients with MCI are at a higher risk of developing AD than others. The challenge with AD is that its effects on the brain start decades before noticeable symptoms appear, making early detection crucial [4].

Given the risk to human life and the strain on resources, it has become imperative to diagnose the cognitive impair- ment associated with Alzheimer’s beforehand. This would help healthcare facilities and elder care homes to plan better for the future, ensuring that the patients can avail treatment and medicines without delay. Moreover, it would also mean a reduced burden on unpaid caregivers for those riddled with AD. According to the Alzheimer’s Association, unpaid caregivers provided an estimated 18.4 billion hours of care valued at nearly 350 billion dollars in the year 2023 alone [3]. Moreover, diagnosing the onset of AD beforehand would also mean patients can prevent further cognitive deterioration, which would worsen without treatment. In “individuals over 60 years of age, dementia contributes 11.2 percent of the years lived with disability, compared with 9.5 percent for stroke, 8.9 percent for musculoskeletal disorders, and 5.0 percent for cardiovascular disease” [2]. It would improve the quality of life of the aged population, helping prevent functional disability and institutionalization [2].

Traditional machine learning methods for early AD diagnosis often rely on two types of features: region of interest (ROI)- based and voxel-based features. ROI-based features focus on specific brain areas, such as cortical thickness or hippocampal volume, while voxel-based features analyze small, 3D blocks of brain tissue in more detail. These techniques depend on assumptions related to changes in cortical thickness (the outer layer of the brain), hippocampal volume (which plays a crucial role in memory formation), and gray matter volume (the brain tissue that processes information), all of which are indicators of cognitive decline associated with AD [4].

In this study, we aim to employ machine-learning techniques to detect Alzheimer’s in patients. The dataset consists of Mag- netic Resonance Imaging (MRI) images and morphometric analysis. The samples are distributed in Mild Demented, Mod- erate Demented, Non-Demented, and Very Mild Demented patients. Using machine learning techniques such as classifiers like Logistic Regression, XGBoost, and Decision Trees, we aim to classify test images with higher accuracy. Moreover, we will be using transformers and Convolutional Neural Networks (CNN) to make our prediction systems even more robust.

# **Literature Review**

The existing literature on predicting Alzheimer’s using machine learning can be divided into two broad categories based on the datasets used. The first category uses MRI images and computer vision techniques whereas the second approach focuses on demographic data and the impact on brain regions. The first category is further divided into different machine- learning techniques such as using classifiers or a Convolution Neural Network (CNN).

In the research paper, ‘Early-Stage Alzheimer’s Disease Prediction Using Machine Learning Models’, authors have used techniques like Decision Tree, Random Forest, Support Vector Machine, Gradient Boosting, and Voting classifiers to identify the best parameters for Alzheimer’s disease prediction [5]. They have used parameters like precision, recall, accuracy, and f1-score to gauge the performance of their models. The average accuracy of these models was 83 percent on test data, acquired from the Open Access Series of Imaging Studies (OASIS) database [5]. It is important to note that using these classification models and achieving significant accuracy denotes that predicting early-stage Alzheimer’s does not have to be a complex task.

Similarly, the research paper, ‘A Novel Approach Utilizing Machine Learning for the Early Diagnosis of Alzheimer’s Disease’, tested a series of machine learning models to predict Alzheimer’s disease in patients using MRI data from the OA- SIS longitudinal dataset [6]. The models tested were Gaussian Naive Bayes, XGBoost, Decision Tree, Random Forest, Gra- dient Boosting, and Voting Classifier. The highest validation accuracy was achieved by the Voting Classifier, achieving 96 percent. This example demonstrates the effectiveness of combining multiple models to enhance prediction performance [6].

A different approach was done in [4]. In this study, an end- to-end Alzheimer’s disease early detection and classification (E2AD2C) framework is introduced, focusing on deep learning techniques, specifically convolutional neural networks (CNN). The framework is designed to classify four stages of AD:

(I) Clinically Stable or Normal Control (NC), (II) Early Mild Cognitive Impairment (EMCI), (III) Late Mild Cognitive Impairment (LMCI), and (IV) Alzheimer’s disease (AD) [7]. Two methods are proposed; one is a simple CNN architecture that processes 2D and 3D brain scans, while the other uses transfer learning using the pre-trained model VGG19. Re- garding the results from both methods, very promising results were obtained. The results showed that the fine-tuned VGG19 model achieved the highest accuracy at 97%, followed by the proposed 3D-M2-IC model at 95.17 percent and the 2D-M2- IC model at 93.6 percent [4].

Another interesting approach is the study of morphometric features in [9] using FreeSurfer to conduct MRI analysis. FreeSurfer is a software, which can be used to convert MRI sessions into statistical data about the brain. The datasets used were the Alzheimer’s Disease Neuroimaging Initiative (ADNI) database (adni.loni.usc.edu) and the Open Access Series of Imaging Studies (OASIS) project database [9]. The study divides each hemisphere into 34 regions of interest (ROI) which are most commonly linked with cognitive decline. For each of these regions they obtained 8 measures were obtained “(surface area, volume, average cortical thickness, the standard deviation of the cortical thickness, integrated rectified mean curvature, integrated rectified Gaussian curvature, folding index, and intrinsic curvature index)” [9]. Expanding upon this, they used these features as inputs to binary classifiers. Various classifiers are used in combination with voting to conclude. It was observed that hippocampal features were the strongest contributors to the classification decisions, followed by temporal, cingulate, and frontal regions [9]. This is consistent with our current understanding of AD and its progression. The focus on different brain regions in this study can prove to be helpful in potential clinical use as it gives healthcare providers an insight into which brain regions contribute most to cognitive decline.

A new architecture called DEMNET was proposed by [7], a convolutional neural network (CNN) model designed to enhance accuracy in the classification of Alzheimer’s disease (AD) and dementia through the extraction of discriminative features from MRI images. Due to the dataset imbalance, SMOTE was implemented to fix this issue. Therefore, the model was tested on two different datasets, one with SMOTE and the other without SMOTE. Without SMOTE, the model achieved an overall training accuracy of 96% but a validation accuracy of 78 percent, indicating overfitting due to the dataset imbalance. However, after applying SMOTE, the model’s per- formance improved significantly, achieving 99 percent training and 94 percent validation accuracy. Regarding testing accu- racy, the model achieved 85 percent without SMOTE and 95.23 percent with SMOTE [7].

For an early diagnosis of Alzheimer’s using deep learning [8], The researchers proposed a deep learning architecture consisting of stacked sparse autoencoders and a softmax regression layer for multiclass classification. The model operates semi-supervised, potentially developing into a fully unsuper- vised method in the future. The proposed method achieved an overall accuracy of 87.76 percent for binary classification between Normal Control (NC) and Alzheimer’s Disease (AD) [8].

Moreover, the authors of [10] have used grey-wolf optimization (GWO) to tune their hyperparameters. The deep learning models used were VGG16, DenseNet, InceptionV3, and DenseNet-LSTM. The paper also proposes a hybrid model VGG16 and InceptionV3. All of these models are optimized using grey-wolf optimization. “Using GWO, the hybrid model (VGG16 + InceptionV3) achieved the best results with a 100 percent accuracy rate on the SPECT DaTscan MRI dataset and 99.94 percent accuracy on the T1, T2-weighted MRI dataset” [10]. This shows the effectiveness of GWO in improving model performance through hyperparameter selection [10].

Despite extensive research, there remains a gap in the literature when it comes to utilizing the analysis of morphometric features to predict early stage Alzheimer’s. While the authors in [9] have used FreeSurfer data to link brain regions with cognitive decline and predict Alzheimer’s, this study aims to go further and study how this approach can be combined with MRI images to improve AD prediction. Given that the proportion of the total population is gradually increasing, AD is a potential issue for healthcare resources in the future. This means that being able to predict it in time with fewer resources would give hospitals the time to advise patients, and ensure efficient care is provided.

# **Methodology**

The methodology used in this study to predict Alzheimer’s can be divided into MRI image analysis and brain morphometric feature analysis. While the image analysis mostly relies on understanding and identifying changes in MRI images which could help predict AD, the morphometric analysis predicts it using changes in brain region volumes in Alzheimer’s patients.

## **Datasets**

This study is based on multiple datasets. For training classifiers and CNN, an Open Access Series of Imaging Studies (OASIS-1) Alzheimer’s dataset was used [11].

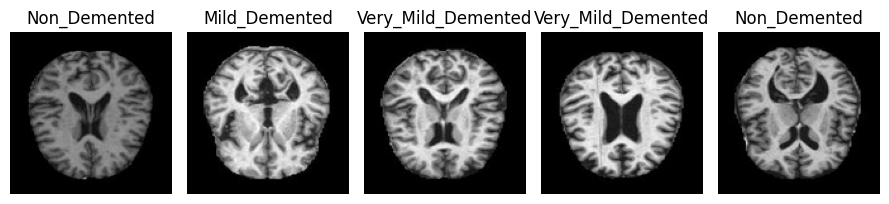
Dataset: <https://drive.google.com/drive/folders/136q2HeSw-HVxYaEDrwUqKLBicsdswPzd?usp=share_link>

This dataset consists of MRI images for non-demented, very mild dementia, mild dementia, and moderate dementia patients. Each of these categories highlights a progressive neuro-degeneration in patients. The non-demented patients show no sign of dementia, the very mild category is for patients showing more signs of cognitive impairment and moderate dementia is the category of patients most impaired due to dementia. The dataset was obtained from Kaggle and was pre-processed from Nifti format (.nii) into .img files. It consists of MRI images for 461 patients.

This code was used to visualize the MRI images:

| fig, axes = plt.subplots(1, 5, figsize=(9, 3))  **for** i **in** range(5):  ax = axes[i]  ax.imshow(x\_train[i], cmap='viridis')  ax.set\_title(main\_labels[y\_train[i]])  ax.axis('off')  plt.tight\_layout() plt.show() |
| --- |

Following is the visualization of the MRI images:



Moreover, this study also uses the Alzheimer’s Disease Neuroimaging Initiative (ADNI) data set to train the transformer [12]. This dataset was obtained from the ADNI image collection. It consists of not only three-dimensional MRI images of patients but also their age, gender, and the group of their diagnosis. The diagnosis groups in ADNI are Cognitive Normal (CN), Mild Cognitive Impairment (MCI), and Alzheimer’s Disease (AD).

Dataset link:

ADNI Dataset:

<https://drive.google.com/file/d/1D9MsW6Zm7ow3u1fEcUrGzEPPpPT8G9sQ/view?usp=sharing>

### **Morphometric Analysis Dataset Construction**

The dataset for morphometric brain analysis has been constructed using OASIS-1 [11]. Firstly, we obtained the clinical and FreeSurfer data from the OASIS website. The clinical data defines patient diagnosis, age and gender. The FreeSurfer data, on the other hand, provided MRI analysis with the volume, surface area and other quantitative information about brain regions.

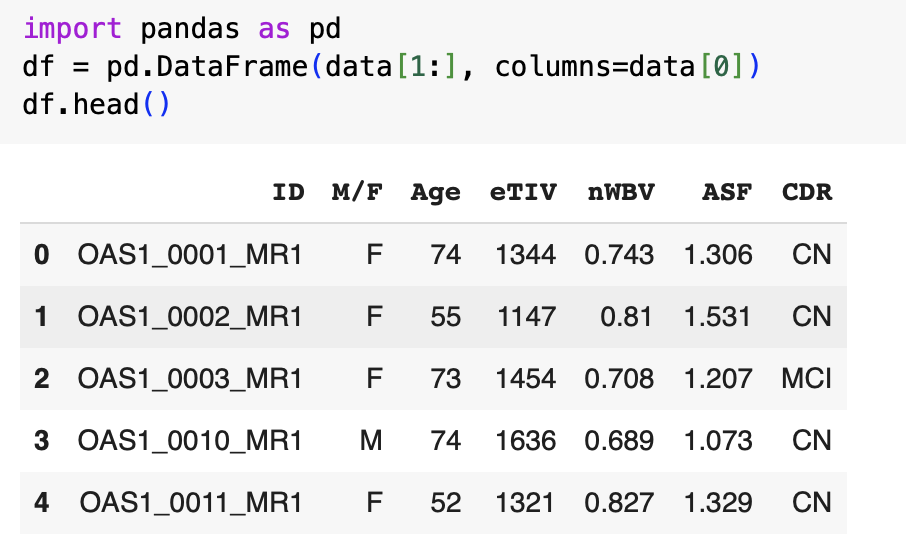
Dataset link:

FreeSurfer Data: <https://drive.google.com/drive/folders/1kMRMOQRP3CsFRU4XJepEifXDfOm8qG50?usp=sharing>

Clinical data:

<https://docs.google.com/spreadsheets/d/1_BZJAbp-m-jPCghHklySJJeAn2W7uH6hpktqf7-TGOs/edit?usp=sharing>

The clinical data is shown below:



**‘ID’** stands for the MRI identifier. **‘M/F’** identifies the patient’s gender. **‘eTIV’** is estimated total intracranial volume , **‘nWBV’** refers to normalized whole-brain volume , **‘ASF’** is the Atlas scoring factor , and **‘CDR’** is the clinical dementia rating.

For each MRI conducted, we extracted the volumes of left-hippocampus, right-hippocampus, left- amygdala, right-amygdala, left-cerebral-cortex, right-cerebral- cortex, left-thalamus-proper and right-thalamus-proper. These brain regions were selected as the hippocampus is responsible for memory formation, the amygdala is involved in processing emotions, the cortex is responsible for logic building and facial recognition, and the thalamus is involved in language and motor skills. All of these functions are impacted in Alzheimer’s patients. To retrieve this information, we started by defining the list of features we are interested in.

| alzheimers\_features = [  'Left-Hippocampus',  'Right-Hippocampus',  'Left-Amygdala',  'Right-Amygdala',  'Left-Cerebral-Cortex',  'Right-Cerebral-Cortex',  'Left-Thalamus-Proper',  'Right-Thalamus-Proper' ] |
| --- |

Then we defined a function that reads the ‘aseg.stats’ file for each MRI ‘ID’ and extracts the features.

| **import** os  patients\_data = {} **def** **extract\_volumes**(path, file\_name, diagnosis):  folder\_path = os.path.join(path, file\_name)   **if** **not** os.path.isdir(folder\_path):  print(f"Folder {file\_name} not found")    #path for the folder containing all the volumes  stats\_path = os.path.join(folder\_path, 'stats/aseg.stats')   # Check if the stats file exists  **if** **not** os.path.isfile(stats\_path):  print(f"aseg.stats file not found in {file\_name}")  **return** **None**   **with** open(stats\_path, 'r') **as** file:  lines = file.readlines()   #finding where the information starts  start = **None**  **for** i, line **in** enumerate(lines):  **if** "ColHeaders" **in** line:  start = i + 1  **break**   data = lines[start:]   id\_brain\_data = []   **for** line **in** data:  columns = line.split()   # incomplete data  **if** len(columns) < 10:  **continue**   struct\_name = columns[4]  volume = float(columns[3])   **if** struct\_name **in** alzheimers\_features:  id\_brain\_data.append((str(struct\_name),volume))   id\_brain\_data.append(("diagnosis", diagnosis))  patients\_data[file\_name] = id\_brain\_data |
| --- |

First, it reads the ‘aseg.stats’ file and filters the data to be read only after ‘ColHeaders’ as this is where the MRI information begins. Then, it ensures that incomplete data is not being read. It reads the ‘struct\_name’ and its corresponding volume and only adds it to the patient’s data if we wish to read data for this feature as defined in our ‘alzheimers\_features’ list.

To perform extraction, keeping only ‘CDR’ from the clinical dataset:

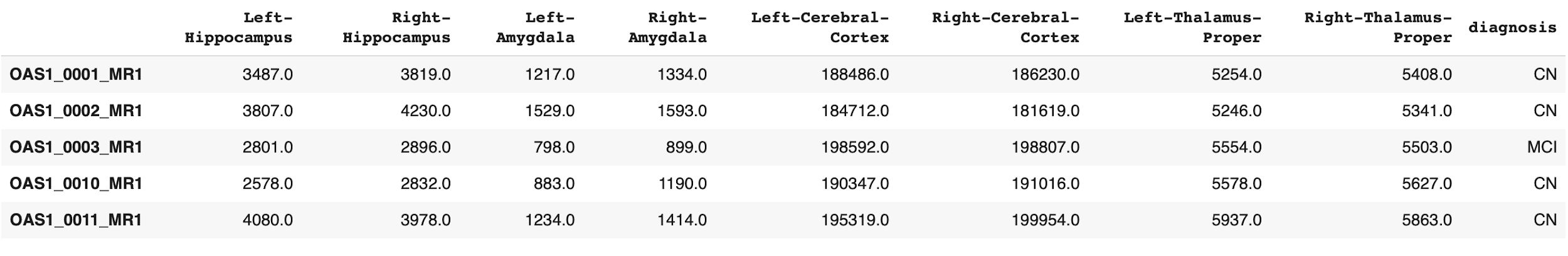
| **for** id, cdr **in** zip(df['ID'], df['CDR']):  extract\_volumes("/content/drive/MyDrive/Project COMP MSc 2024 25/Morphometric Features/disc1",id, cdr ) |
| --- |

We only kept ‘CDR’ due to limited resources to process more features.

For the purpose of processing data, we defined another function. This converts the data stored in ‘patients\_data’ into a data frame that we can analyse.

| columns = ['Left-Hippocampus',  'Right-Hippocampus',  'Left-Amygdala',  'Right-Amygdala',  'Left-Cerebral-Cortex',  'Right-Cerebral-Cortex',  'Left-Thalamus-Proper',  'Right-Thalamus-Proper','diagnosis']  rows = {}  **for** patient\_id, brain\_data **in** patients\_data.items():  row = {region: value **for** region, value **in** brain\_data}  rows[patient\_id] = row  df = pd.DataFrame.from\_dict(rows, orient='index', columns=columns)  df.head() |
| --- |

In the end, the following dataset was achieved:



With the dataset, the logistic regression model was trained to test the morphometric features dataset and see which features of the brain are the most influential in predicting Alzheimer's disease in this specific dataset.

| **import** pandas **as** pd **from** sklearn.model\_selection **import** train\_test\_split **from** sklearn.linear\_model **import** LogisticRegression **from** sklearn.metrics **import** accuracy\_score, confusion\_matrix, classification\_report  X = df.drop('diagnosis', axis=1) y = df['diagnosis'] # Target: 'CN', 'MCI', 'AD'  X\_train, X\_test, y\_train, y\_test = train\_test\_split(  X, y,  test\_size=0.2,  random\_state=42,  stratify=y )  model = LogisticRegression(multi\_class='multinomial', solver='lbfgs', max\_iter=1000) model.fit(X\_train, y\_train)  y\_pred = model.predict(X\_test)  print("Accuracy:", accuracy\_score(y\_test, y\_pred)) print("\nConfusion Matrix:\n", confusion\_matrix(y\_test, y\_pred)) print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))  print("Coefficients:", model.coef\_) print("Intercepts:", model.intercept\_) |
| --- |

To determine which morphometric features had the biggest influence in the prediction of Alzheimer’s, we used a graph in which we plotted the absolute coefficient of every morphometric feature, meaning the higher the value the higher the influence or importance.

| **import** numpy **as** np **import** pandas **as** pd **import** matplotlib.pyplot **as** plt  #Compute coefficients per class coef\_df = pd.DataFrame(model.coef\_, columns=X.columns) coef\_df.index = model.classes\_ # Each row corresponds to a diagnosis label print("Coefficients for each class:\n", coef\_df)  #Compute average absolute importance across classes mean\_importance = np.mean(np.abs(model.coef\_), axis=0) importance\_df = pd.DataFrame({  'Feature': X.columns,  'Average\_Absolute\_Coefficient': mean\_importance }) importance\_df = importance\_df.sort\_values(by='Average\_Absolute\_Coefficient', ascending=**False**) print("\nAverage absolute coefficient for each feature (higher means more impact):") print(importance\_df)  #Plot the feature importance as a horizontal bar chart importance\_df\_sorted = importance\_df.sort\_values(by='Average\_Absolute\_Coefficient', ascending=**True**)  plt.figure(figsize=(10, 6)) plt.barh(importance\_df\_sorted['Feature'], importance\_df\_sorted['Average\_Absolute\_Coefficient'], color='skyblue') plt.xlabel('Average Absolute Coefficient') plt.ylabel('Brain Region') plt.title('Feature Importance in Multiclass Logistic Regression Model') plt.tight\_layout() plt.show() |
| --- |

## **MRI Image Classifiers**

To build a base for this study, we began with basic classifiers for the detection of Alzheimer’s using the OASIS dataset. We selected Logistic Regression, Naive Bayes, XGBoost, and Random Forest for this purpose.

Logistic regression is a simple model that serves as a basis for our research. However, the downside of this model is that the features are not extracted making the data very complex. On the other hand, Naive Bayes assumes that features are independent of each other. Moreover, XGboost handles non- linearity well and Random Forest handles overfitting well. Each of these models has its own advantages, which makes them suitable for this purpose. This section will explain the methodology implemented in the classifiers as well how their results were achieved.

### **Data Preprocessing**

To access the data from the folder downloaded from Kaggle, the following code was used:

| test\_dataframe = pd.read\_parquet('/content/drive/MyDrive/Project COMP MSc 2024 25/Kaggle & Oasis Dataset/Data/test-00000-of-00001-44110b9df98c5585.parquet') test\_dataframe.head()  train\_dataframe = pd.read\_parquet('/content/drive/MyDrive/Project COMP MSc 2024 25/Kaggle & Oasis Dataset/Data/train-00000-of-00001-c08a401c53fe5312.parquet') train\_dataframe.head() |
| --- |

However, this needed to be processed further to be able to use the MRI images for classification.

| **def** **preprocess\_dataset**(data):  x = []  y = []  **for** i **in** range(len(data)):  instance = data.iloc[i]  img = instance['image']  img\_bytes = img['bytes']  img\_label = instance['label']  np\_array = np.frombuffer(img\_bytes, np.uint8)  image = cv2.imdecode(np\_array, cv2.IMREAD\_COLOR)  **if** image **is** **not** **None**:  image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)  **if** image **is** **not** **None**:  x.append(image)  y.append(img\_label)  **else**:  print("ignoring an image")   **return** x, y |
| --- |

To do this, we iterated through each instance of the data and extracted the label as well as converted image byte data into an array to be able to use it for classification. To construct the train and test data, we used the following code:

| x\_train, y\_train = preprocess\_dataset(train\_dataframe) x\_test, y\_test = preprocess\_dataset(test\_dataframe) |
| --- |

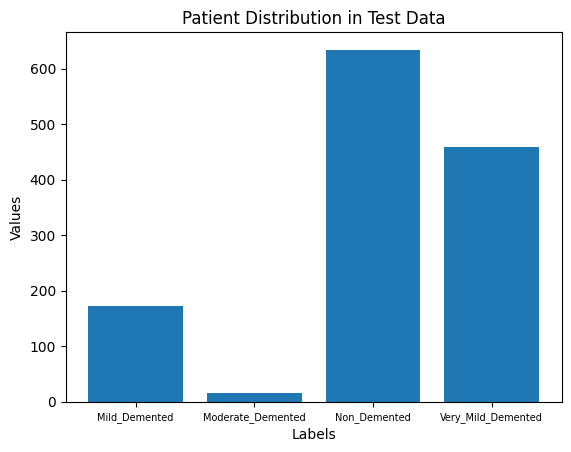
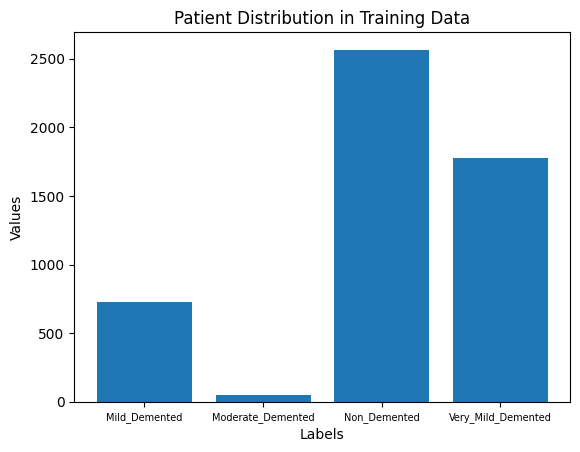
### **Exploratory Data Analysis**

Exploratory data analysis was conducted to understand the OASIS dataset. The data contains 5120 train samples and 1280 test samples.

Moreover, it was discovered that there was a data imbalance in the patient distribution in the training and testing samples.

| main\_labels = ['Mild\_Demented', 'Moderate\_Demented', 'Non\_Demented', 'Very\_Mild\_Demented']  #bar chart for all train sample patient distribution y\_tr\_values = [y\_train.count(0), y\_train.count(1), y\_train.count(2), y\_train.count(3)] y\_te\_values = [y\_test.count(0), y\_test.count(1), y\_test.count(2), y\_test.count(3)]  #bar chart for train sample plt.bar(main\_labels, y\_tr\_values)  plt.xlabel('Labels') plt.ylabel('Values') plt.title('Patient Distribution in Training Data') plt.xticks(fontsize=7) plt.show() |
| --- |

This code visualizes the distribution of all the classes in the train dataset, showing us how patients in the dataset are distributed. The same was performed for the test dataset.



*Figure 1: Patient Distribution in Training Data Figure 2: Patient Distribution in Testing Data*

As shown in Figures 1 and 2, mild and moderate dementia patients are less compared to nondemented and very mild dementia patients. As this imbalance would impact the results generated by the classifiers, data augmentation was performed to generate more samples in the mild and moderate dementia classes. As this data augmentation was used for CNN, it is explained [here.](#bookmark=id.2s8eyo1)

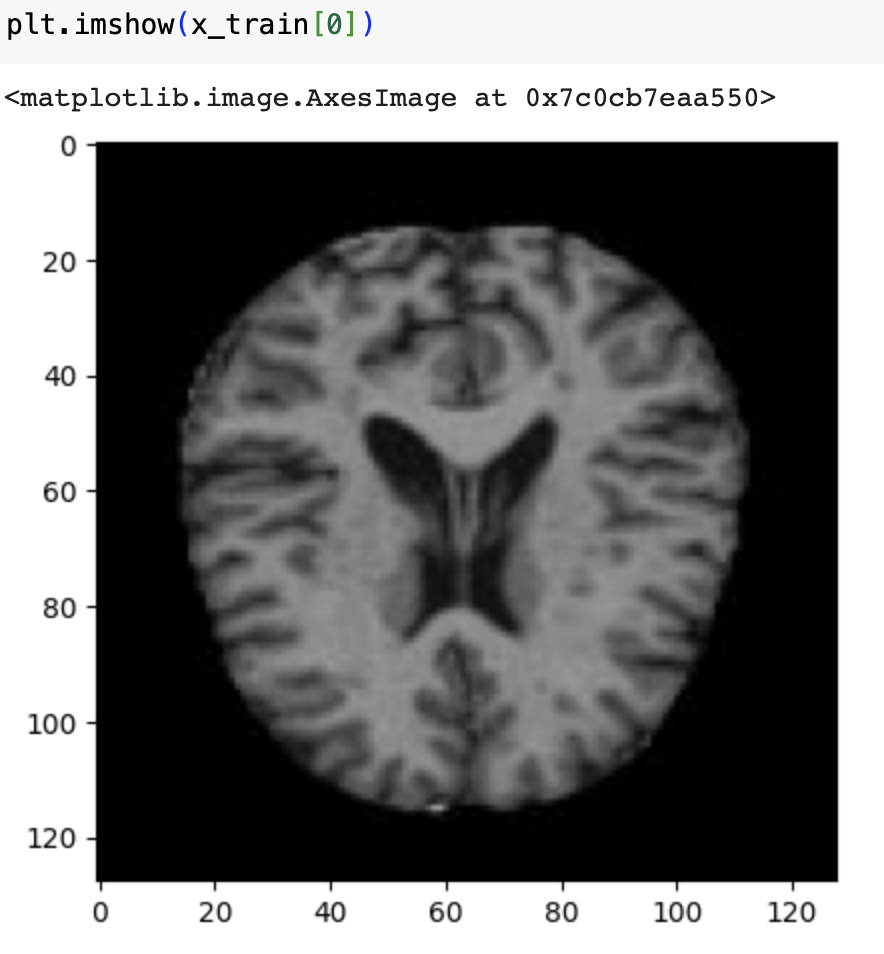
### **Image Enhancement**

AS MRI images are mostly black and white, brain images must be enhanced to help detect brain matter. Initially, we wished to achieve skull stripping to enhance the results, however due to having limited resources we could not implement skull stripping as Google Collab would crash.

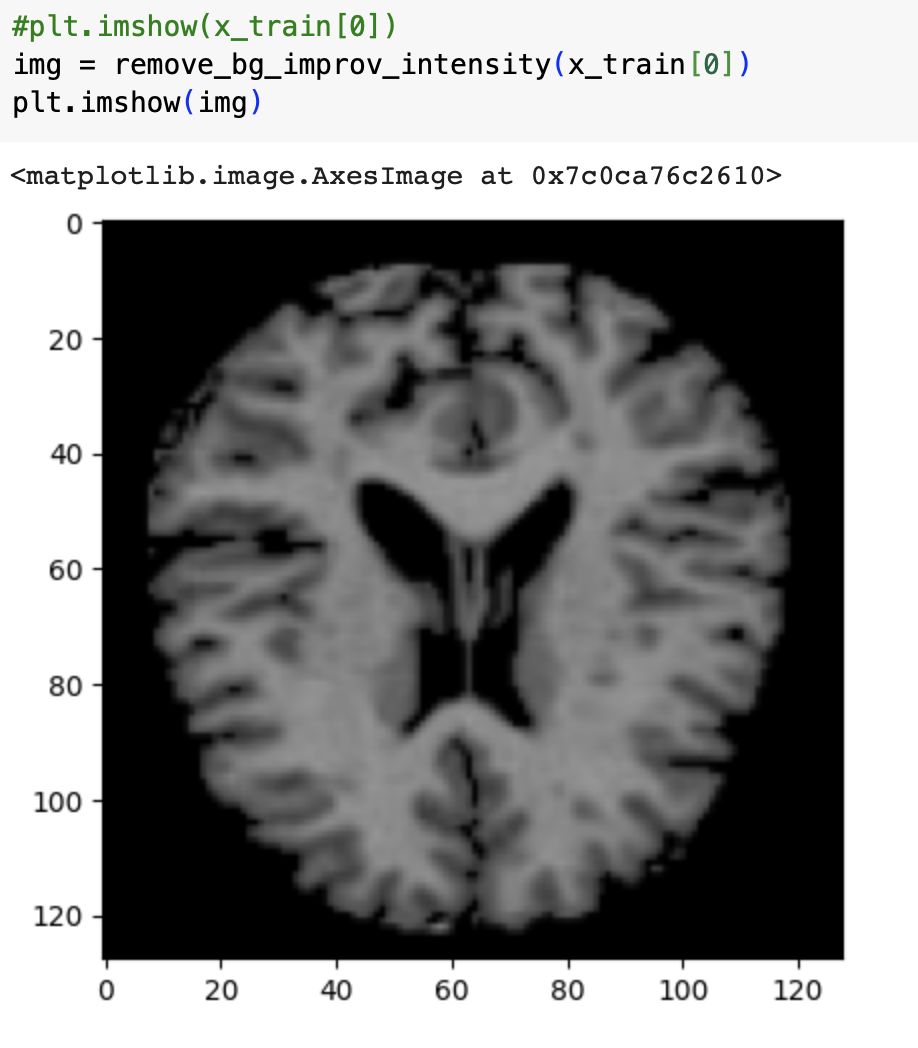
Hence we settled for a simple cropping technique.

| **def** **remove\_bg\_improv\_intensity**(image):  non\_zero = np.nonzero(image)   #find min and max cords of the brain region  min\_coords = [np.min(indices) **for** indices **in** non\_zero]  max\_coords = [np.max(indices) **for** indices **in** non\_zero]   slices = tuple(slice(min\_coord, max\_coord + 1) **for** min\_coord, max\_coord **in** zip(min\_coords, max\_coords))  cropped = image[slices]   ''' increasing intensity reduced accuracy  factor = 2  #increase img intensity by 2  increased\_intensity = cropped \* factor  #make sure its not more than 255  final\_img = np.clip(increased\_intensity, 0, 255)'''  cropped[cropped < 50] = 0  resized = cv2.resize(cropped, (128,128))  **return** resized |
| --- |

This code takes the minimum and maximum coordinates where the brain region exists and crops out the background. Increasing the intensity of the brain region to make brain matter more visible only reduced the accuracy of the classifiers. The images were resized to ensure it can be input into the models. The following image is without enhancement:



And this is with enhancement:



**Classifiers**

The images were flattened to be input into the classifiers:

| flatten\_ximg = [x\_train[i].flatten() **for** i **in** range(len(x\_train))] flatten\_yimg = [x\_test[i].flatten() **for** i **in** range(len(x\_test))] |
| --- |

The library used for these models is ‘sklearn’. Each of the models follow the same method of fitting the mode on train images and train labels. Then the labels are predicted using the test images. It also prints the accuracy for each model and visualizes the confusion matrix.

### **Logistic regression**

| model = LogisticRegression(multi\_class='multinomial', solver='lbfgs', max\_iter=150) model.fit(flatten\_ximg, y\_train)  # Make predictions y\_pred = model.predict(flatten\_yimg)  accuracy = accuracy\_score(y\_test, y\_pred) print(f"Test Accuracy: {accuracy:.4f}")  conf\_matrix = confusion\_matrix(y\_test, y\_pred)  print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))  #Create a heatmap plot of the confusion matrix plt.figure(figsize=(8, 6)) sns.heatmap(conf\_matrix, annot=**True**, fmt='d', cmap='Blues', cbar=**False**,  xticklabels=['Mild Demented', 'Moderate Demented', 'Non Demented', 'Very Mild Demented'],  yticklabels=['Mild Demented', 'Moderate Demented', 'Non Demented', 'Very Mild Demented'])  plt.title('Confusion Matrix') plt.xlabel('Predicted Labels') plt.ylabel('True Labels') plt.show() |
| --- |

In the Logistic Regression function:

* multi\_class='multinomial’ is used for multinomial classification
* solver='lbfgs' is used for optimization and it is the best one for small to medium sized data sets
* max\_iter=150 is the number of iterations of optimization

### **Naive Bayes**

| gaussian\_nb = GaussianNB() gaussian\_nb.fit(flatten\_ximg, y\_train)  y\_pred = gaussian\_nb.predict(flatten\_yimg)   accuracy = accuracy\_score(y\_test, y\_pred) print(f"Test Accuracy: {accuracy:.4f}")  conf\_matrix = confusion\_matrix(y\_test, y\_pred)  print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))  #Create a heatmap plot of the confusion matrix plt.figure(figsize=(8, 6)) sns.heatmap(conf\_matrix, annot=**True**, fmt='d', cmap='Blues', cbar=**False**,  xticklabels=['Mild Demented', 'Moderate Demented', 'Non Demented', 'Very Mild Demented'],  yticklabels=['Mild Demented', 'Moderate Demented', 'Non Demented', 'Very Mild Demented'])  plt.title('Confusion Matrix') plt.xlabel('Predicted Labels') plt.ylabel('True Labels') plt.show() |
| --- |

### **XGBOOST**

| xgb = XGBClassifier(eval\_metric='mlogloss') xgb.fit(flatten\_ximg, y\_train)  # Make predictions y\_pred = xgb.predict(flatten\_yimg)   accuracy = accuracy\_score(y\_test, y\_pred) print(f"Test Accuracy: {accuracy:.4f}")  conf\_matrix = confusion\_matrix(y\_test, y\_pred)  print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))  #Create a heatmap plot of the confusion matrix plt.figure(figsize=(8, 6)) sns.heatmap(conf\_matrix, annot=**True**, fmt='d', cmap='Blues', cbar=**False**,  xticklabels=['Mild Demented', 'Moderate Demented', 'Non Demented', 'Very Mild Demented'],  yticklabels=['Mild Demented', 'Moderate Demented', 'Non Demented', 'Very Mild Demented'])  plt.title('Confusion Matrix') plt.xlabel('Predicted Labels') plt.ylabel('True Labels') plt.show() |
| --- |

In the XGBClassifier function:

* eval\_metric='mlogloss' defines the loss function and this is multiclass log loss used for multiclass classification problems.

### **Random Forest**

| # Initialize the Random Forest model rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)  # Train the model rf\_model.fit(flatten\_ximg, y\_train)  # Predict on the test set y\_pred = rf\_model.predict(flatten\_yimg)  # Evaluate the model's accuracy accuracy = accuracy\_score(y\_test, y\_pred) print(f"Test Accuracy: {accuracy:.4f}")  # Print a detailed classification report print(classification\_report(y\_test, y\_pred))  #Generate the confusion matrix conf\_matrix = confusion\_matrix(y\_test, y\_pred)  #Create a heatmap plot of the confusion matrix plt.figure(figsize=(8, 6)) sns.heatmap(conf\_matrix, annot=**True**, fmt='d', cmap='Blues', cbar=**False**,  xticklabels=['Mild Demented', 'Moderate Demented', 'Non Demented', 'Very Mild Demented'],  yticklabels=['Mild Demented', 'Moderate Demented', 'Non Demented', 'Very Mild Demented'])  plt.title('Confusion Matrix') plt.xlabel('Predicted Labels') plt.ylabel('True Labels') plt.show() |
| --- |

In the RandomForestClassifer function:

* n\_estimators=100 is the number of decision trees in the forest. It is used for increasing the performance.
* random\_state=42 is the random seed to ensure reproducibility of results.

### **Classifiers with Image Enhancement and Data Augmentation**

### We performed the same classifications with a dataset that had been processed through the ‘Image Enhancement’ function as well as augmented data. However, there was no significant improvement in results with both of these approaches.

### **Image Testing**

The classifiers were tested individually with random images from the testing data set, we define a function where it provides the predicting probability per class and the class with the higher probability is the selected output from the model.

| **def** **predict\_with\_probabilities**(image, model, label\_names, expected\_label=None):  # Flatten the image  image\_flattened = image.flatten().reshape(1, -1)  # Get prediction probabilities  probabilities = model.predict\_proba(image\_flattened)[0]  # Get the predicted label  predicted\_label\_index = np.argmax(probabilities)  predicted\_label\_name = label\_names[predicted\_label\_index]  predicted\_probability = probabilities[predicted\_label\_index] \* 100   # Format probabilities for output  probability\_output = "\n".join(  [f"{label}: {prob:.2f}%" **for** label, prob **in** zip(label\_names, probabilities \* 100)]  )   # Format the output string  output = (  f"Prediction Probabilities:\n\n{probability\_output}\n\n"  f"Predicted Class: {predicted\_label\_name} {predicted\_probability:.2f}%"  )  **if** expected\_label **is** **not** **None**:  output += f"\nExpected Class: {label\_names[expected\_label]}"   **return** output |
| --- |

We get a random image per class, in this case, four classes.

| # Get a random image from each type of image rand\_mild\_image = random.choice(arr\_mild) rand\_moderate\_image = random.choice(arr\_moderate) rand\_non\_image = random.choice(arr\_non) rand\_very\_mild\_image = random.choice(arr\_very\_mild)  # Define label names label\_names = ['Mild Demented', 'Moderate Demented', 'Non Demented', 'Very Mild Demented'] |
| --- |

Now, we test the images with all the classifiers; we did this with every classifier to validate the results and get a more in-depth understanding of how the model performs with one random image at a time.

| # Predict for each randomly selected image predict\_with\_separator(rand\_mild\_image, xgb, label\_names, expected\_label=0, title="Prediction for a Random Mild Demented Image") predict\_with\_separator(rand\_moderate\_image, xgb, label\_names, expected\_label=1, title="Prediction for a Random Moderate Demented Image") predict\_with\_separator(rand\_non\_image, xgb, label\_names, expected\_label=2, title="Prediction for a Random Non Demented Image") predict\_with\_separator(rand\_very\_mild\_image, xgb, label\_names, expected\_label=3, title="Prediction for a Random Very Mild Demented Image") |
| --- |

### **Explainable AI**

For this project, we also attempted to build explainable AI that would explain how accurate our results were. Due to lack of clinical resources, we could not verify results through a doctor or nurse. Hence, we resorted to AI. For this purpose, we use the Lime Image Explainer.

| **from** lime.lime\_image **import** LimeImageExplainer explainer = LimeImageExplainer()  # Explain the image being tested # this explainer generates 10 perturbed images of the input image turning # some pixels on and off explanation = explainer.explain\_instance(  process\_yimg[200],  predict,  top\_labels=3,  hide\_color=0,  num\_samples=1000 )  #pred label |
| --- |

In the code above, LimeImageExplainer is used to start an explanation for the ‘process\_yimg[200]’ image that is being predicted using the ‘predict’ function and the top 3 labels are being selected. The predict function is the following:

| **import** numpy **as** np  **def** **predict**(images):  flattened\_images = images.reshape(images.shape[0], -1)  pred = model.predict\_proba(flattened\_images)  **return** pred |
| --- |

Then, we compare the predicted and actual labels.

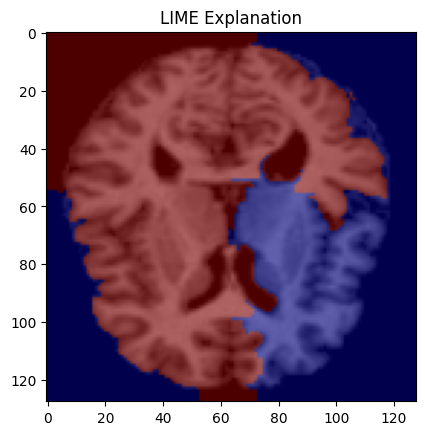
| pred\_label = explanation.top\_labels print(f"Predicted label: {pred\_label[0]}") print(f"Actual label: {y\_test[200]} ") |
| --- |

And then we visualize the original image versus the region that contributes most to the prediction according to the lime image explainer.

| temp, mask = explanation.get\_image\_and\_mask(  explanation.top\_labels[0], positive\_only=**True**, num\_features=10, hide\_rest=**False** ) **import** matplotlib.pyplot **as** plt plt.imshow(temp, cmap='gray') #you can control how much you wanna highlight the mask with alpha plt.imshow(mask, alpha=0.6, cmap='jet') # Overlay explanation on the image plt.title('LIME Explanation') plt.show() |
| --- |

And the output is as follows:

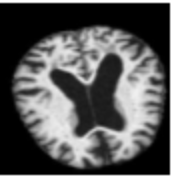
The jet colormap used is a color gradient where dark blue is the least important region and the red is the most important region contributing to the results.



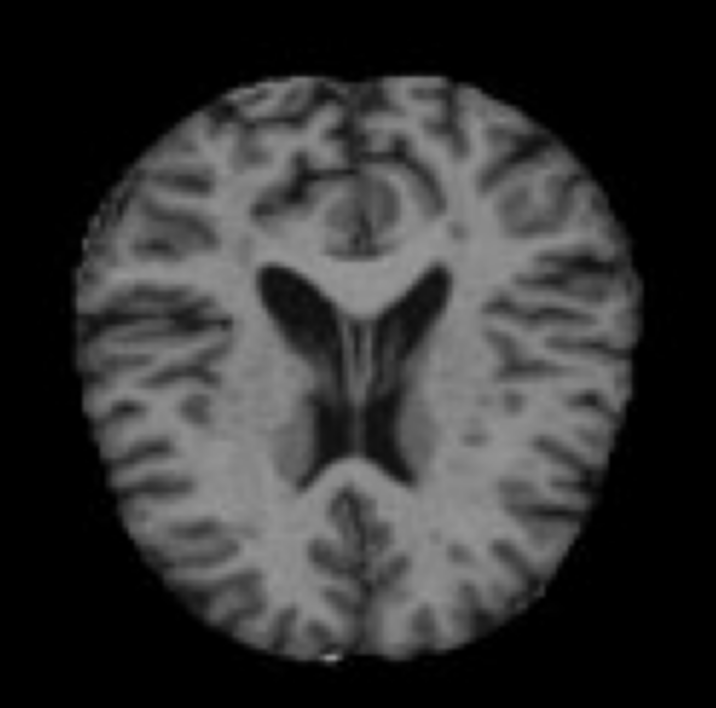
## **Convolutional Neural Networks**

In addition to this, the study also uses the Convolutional Neural Network (CNN) to classify the images. Figure 5 shows the architecture of CNN. The architecture was designed to efficiently extract features from the MRI images and optimize the classification accuracy.

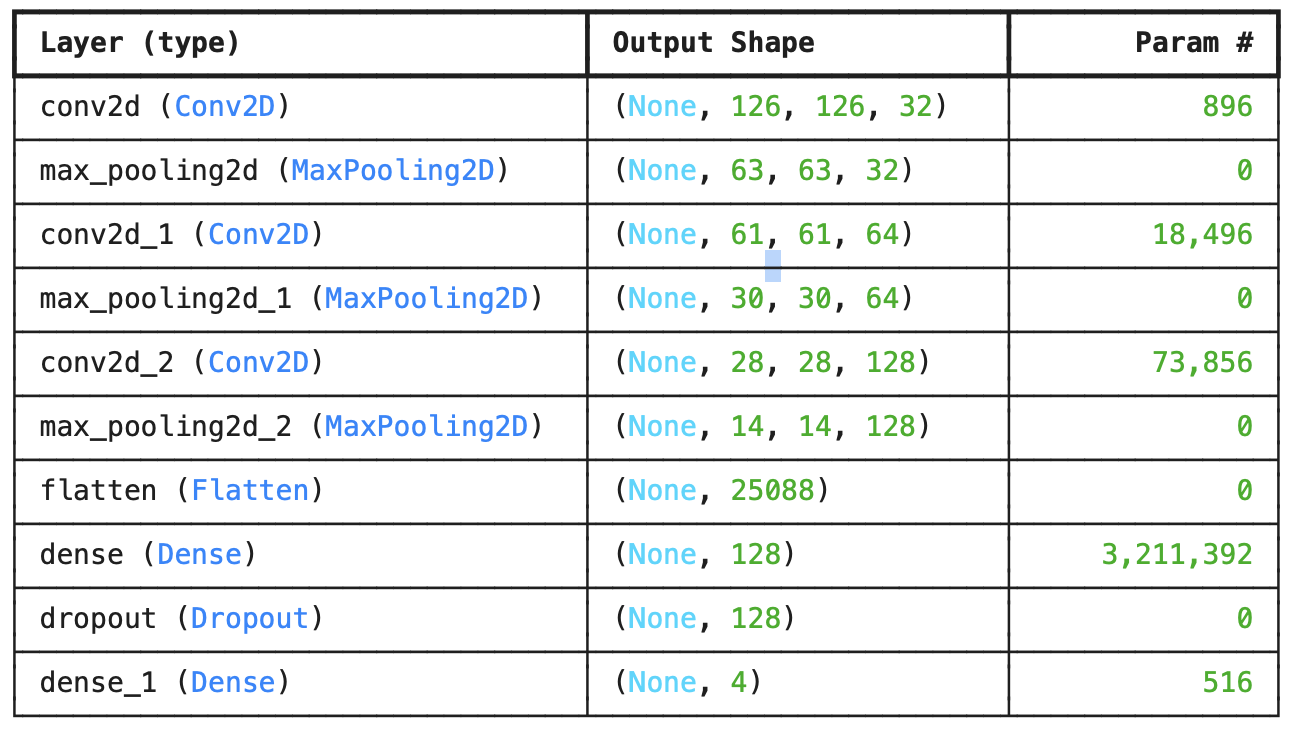
To address data imbalances in the Kaggle dataset, particularly for the mild and moderate demented classes with fewer samples, targeted data augmentation was implemented as part of the CNN training process. The following augmentation parameters were used: a rotation range of 20 degrees, width, and height shifts of up to 10%, a shear range of 0.2, a zoom range of 0.2, and horizontal flipping. These techniques enhanced the training dataset, allowing the CNN to learn robust features from a more diverse set of images while preserving critical anatomical structures. Figure 3 shows an original non-augmented brain MRI, and Figure 4 presents an example of an augmented image.



*Figure 3: Augmented MRI (Kaggle Dataset)*



*Figure 4: No Augmentation MRI (Kaggle Dataset)*



*Figure 5: CNN Architecture*

As mentioned, a series of MRI images from Kaggle were used for the CNN dataset. This dataset was in .parquet format, and we utilized the Pandas library to read the data from it.

| test\_dataset = pd.read\_parquet('~/Documents/caoa/Lakehead/Final\_Project/test-00000-of-00001-44110b9df98c5585.parquet') test\_dataset.head() |
| --- |



The function preprocess\_dataset takes a pandas DataFrame as input and processes each row to extract image data and labels for further analysis. It iterates through the DataFrame, retrieving the image bytes and the corresponding label for each instance. The image bytes are converted into a NumPy array and then decoded into an image using OpenCV. If the image is successfully decoded, it is converted from BGR to RGB color format (since OpenCV uses BGR by default) and appended to a list of images, while the label is added to a corresponding list of labels. If an image cannot be processed, a message is printed, and that image is skipped. Finally, the function returns both the images and labels as NumPy arrays, preparing them for subsequent tasks such as model training.

| **def** **preprocess\_dataset**(data):  x = []  y = []  **for** i **in** range(len(data)):  instance = data.iloc[i]  img = instance['image']  img\_bytes = img['bytes']  img\_label = instance['label']  np\_array = np.frombuffer(img\_bytes, np.uint8)  image = cv2.imdecode(np\_array, cv2.IMREAD\_COLOR)  **if** image **is** **not** **None**:  image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)  **if** image **is** **not** **None**:  x.append(image)  y.append(img\_label)  **else**:  print("ignoring an image")   **return** np.array(x), np.array(y) |
| --- |

This was our first step in data pre-processing the images from the Kaggle dataset. Our next step was to implement data augmentation on the training dataset. This was done to alleviate the class imbalances from the dataset, specifically Moderate Demented and Mild Demented. When the CNN was first trained without the data augmentation, we noticed a significant decrease in prediction accuracy in these classes; therefore, the targeted data augmentation for these two classes was implemented to improve the accuracy.

Once we have the data, we proceed with the data augmentation. Our first step in this process was to organize the training data by class. The code first creates a dictionary where each key is a unique label from the training labels (y\_train), and each value is an empty list. Then, by iterating over the training labels, the corresponding training sample from X\_train is appended to the list associated with its label. Finally, the code converts each list of samples into a NumPy array, resulting in a dictionary where each key maps to an array of training samples belonging to that specific class.

| # Separate the data by class X\_train\_by\_class = {label: [] **for** label **in** np.unique(y\_train)} **for** i, label **in** enumerate(y\_train):  X\_train\_by\_class[label].append(X\_train[i])  # Convert lists to arrays for each class X\_train\_by\_class = {label: np.array(images) **for** label, images **in** X\_train\_by\_class.items()} |
| --- |

After separating the data by class, we proceed with data augmentation using Keras’ TensorFlow library. We define an image generator that applies several augmentation techniques to expand our dataset. The generator randomly rotates images by up to 20 degrees, shifts them horizontally and vertically by up to 10% of their respective dimensions, and applies a shear transformation to skew the images effectively. Additionally, it randomly zooms in or out by up to 20% and flips images horizontally. Any empty areas created by these transformations are filled using the nearest pixel values.

| # Initialize ImageDataGenerator for augmentation datagen = ImageDataGenerator(  rotation\_range=20,  width\_shift\_range=0.1,  height\_shift\_range=0.1,  shear\_range=0.2,  zoom\_range=0.2,  horizontal\_flip=**True**,  fill\_mode='nearest' ) |
| --- |

Finally, we iterate through the training dataset, generating the new augmented images for classes 0 and 1, which correspond to Moderated Demented and Mild Demented in batches until we achieve our desired amount of images. After the augmentation loop, the augmented images list is converted into a 4-dimensional NumPy array.

| # Augment the underrepresented classes augmented\_images = [] augmented\_labels = []  **for** label, images **in** X\_train\_by\_class.items():  current\_count = len(images)  **if** current\_count < (len(images) + 785) **and** label == 0 **or** label == 1:  needed = 785  print(f"Generating {needed} new images for class {label}...")   # Use datagen to create the necessary samples  i = 0  **for** X\_batch, \_ **in** datagen.flow(images, np.full(current\_count, label), batch\_size=32):  **for** img **in** X\_batch:  augmented\_images.append(img)  augmented\_labels.append(label)  i += 1  **if** i >= needed:  **break**  **if** i >= needed:  **break**  # \*\*Convert augmented\_images to a NumPy array with 4 dimensions\*\* augmented\_images = np.array(augmented\_images)  # Check shapes before concatenation print("Original X\_train shape:", X\_train.shape) print("Augmented images shape:", augmented\_images.shape)  # Concatenate the original and augmented images and labels X\_train\_balanced = np.concatenate([X\_train, augmented\_images], axis=0) y\_train\_balanced = np.concatenate([y\_train, augmented\_labels], axis=0) |
| --- |

The last thing to do is to train the CNN; we defined an image preprocess function to prepare all the images for the CNN, making sure their values are normalized between 0 and 1, ensure is the correct shape we are expecting for the CNN, and transforms a single integer label into a one-hot encoded vector using TensorFlow's tf.one\_hot function.

| **def** **preprocess\_image**(image, label):  # Normalize the image to range [0, 1]  image = tf.cast(image, tf.float32) / 255.0   # Ensure image has shape (128, 128, 3)  image = tf.ensure\_shape(image, (128, 128, 3))   # Convert label to one-hot encoding (4 classes)  label = tf.one\_hot(label, depth=4)   **return** image, label |
| --- |

After preprocessing, the balanced training and test data are converted into TensorFlow datasets and prepared by batching, shuffling (for training), and prefetching to improve efficiency. We then define a CNN model consisting of three convolutional layers with increasing filter sizes—each followed by a max pooling layer—to extract spatial features. The output is flattened and passed through a dense layer with dropout to mitigate overfitting, and finally, a softmax output layer produces class probabilities for the four classes. The model is compiled using the Adam optimizer with categorical cross-entropy loss and trained for 20 epochs using the augmented training data, with validation on the test set. Once trained, the model is saved for future use.

| # Convert balanced NumPy arrays to tf.data.Dataset train\_data\_balanced = tf.data.Dataset.from\_tensor\_slices((X\_train\_balanced, y\_train\_balanced)) test\_data = tf.data.Dataset.from\_tensor\_slices((X\_test, y\_test))  # Apply preprocessing train\_data\_balanced = train\_data\_balanced.map(preprocess\_image, num\_parallel\_calls=tf.data.experimental.AUTOTUNE) test\_data = test\_data.map(preprocess\_image, num\_parallel\_calls=tf.data.experimental.AUTOTUNE)  # Batch, shuffle, and prefetch the datasets BATCH\_SIZE = 32 train\_data\_balanced = train\_data\_balanced.batch(BATCH\_SIZE).shuffle(1024).prefetch(tf.data.experimental.AUTOTUNE) test\_data = test\_data.batch(BATCH\_SIZE).prefetch(tf.data.experimental.AUTOTUNE)  **def** **create\_cnn\_model**():  model = Sequential()    # Convolutional layer 1  model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(128, 128, 3)))  model.add(MaxPooling2D(pool\_size=(2, 2)))   # Convolutional layer 2  model.add(Conv2D(64, (3, 3), activation='relu'))  model.add(MaxPooling2D(pool\_size=(2, 2)))   # Convolutional layer 3  model.add(Conv2D(128, (3, 3), activation='relu'))  model.add(MaxPooling2D(pool\_size=(2, 2)))   # Flatten the layers  model.add(Flatten())   # Fully connected layer  model.add(Dense(128, activation='relu'))  model.add(Dropout(0.5)) # Dropout to reduce overfitting   # Output layer  model.add(Dense(4, activation='softmax')) # 4 classes   # Compile the model  model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])   **return** model  # Create an instance of the model model = create\_cnn\_model()  # Print the model summary model.summary()  # Train the model history = model.fit(  train\_data\_balanced,  validation\_data=test\_data,  epochs=20  )  # Save the model model.save("alzheimer\_cnn\_augmented\_model.keras") |
| --- |

The CNN was validated with a different dataset from kaggle in order to ensure it was generalizing well with unseen data, the following code uses tensorflow to validate the trained cnn with the new dataset.

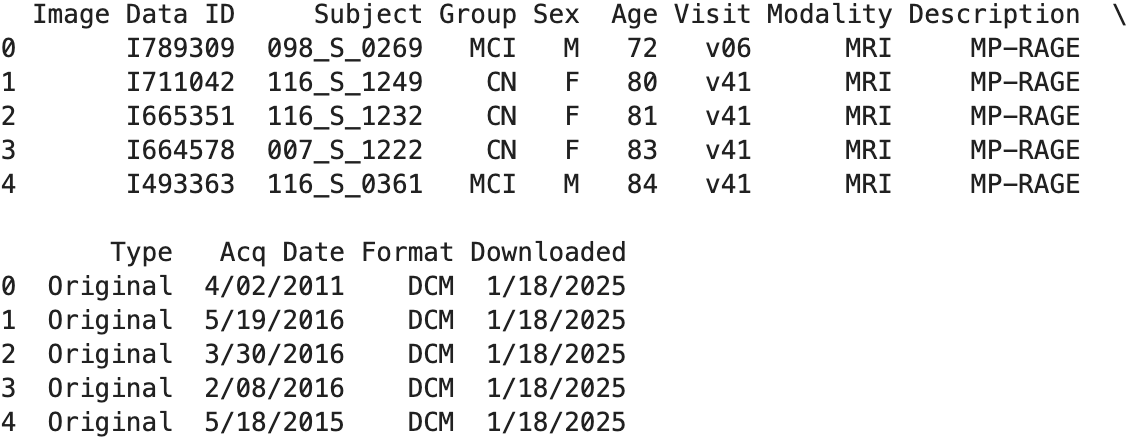
| **import** tensorflow **as** tf **from** tensorflow.keras.preprocessing.image **import** ImageDataGenerator **from** tensorflow.keras.models **import** load\_model **import** os  #Specify the path to your combined dataset base\_dir = "Combined Dataset" test\_dir = os.path.join(base\_dir, "test")  #Create an ImageDataGenerator for normalization datagen = ImageDataGenerator(rescale=1./255)  #Point the generator to your "test" folder test\_generator = datagen.flow\_from\_directory(  directory=test\_dir,  target\_size=(128, 128),   batch\_size=32,  shuffle=**False**,  class\_mode='categorical' )  #Load pre-trained model model = load\_model("alzheimer\_cnn\_augmented\_model-Copy1.keras")  #Evaluate the model on the test set results = model.evaluate(test\_generator, verbose=1) print(f"Test Loss: {results[0]:.2f}") print(f"Test Accuracy: {results[1]:.2f}") |
| --- |

## **Transformers**

In this research, the study employed a pre-trained Vision Transformer model (google/vit-base-patch16-224) provided by the Hugging Face Transformers library. This model was originally trained on ImageNet, which makes it a strong choice for image classification tasks, as it leverages robust learned representations that can be fine-tuned for our specific dataset.

Our first step was to extract the dataset from and excel file which is the form the ADNI provided the dataset.

| **import** pandas **as** pd # Load the Excel file excel\_path = "~/Documents/caoa/Lakehead/Final\_Project/Research\_Papers/ADNI\_Dataset/projectMRI\_1\_18\_2025.csv" metadata = pd.read\_csv(excel\_path)  # Inspect the first few rows print(metadata.head()) print(metadata.info()) |
| --- |



We decided to drop some unnecessary columns that did not influence at all the prediction of the model as well as checking for null values in the dataset

| # Drop unnecessary columns metadata = metadata.drop(columns=["Modality", "Description", "Type", "Format", "Downloaded"])  # Check for missing values print(metadata.isnull().sum())  # Encode the Group column group\_mapping = {"CN": 0, "MCI": 1, "AD": 2}  metadata["Group"] = metadata["Group"].map(group\_mapping)  print(metadata.head()) |
| --- |

For simplicity we wanted to add the path directory of the images of every patient since the images where deeply nested in different files this was going to help when creating the training, testing and validation set. So in the end we had one folder per patient and in each folder we have all the images from the patient.

| # Get all the file paths per user and create a new column on the metadata dataset called filepaths  **import** os # Path to your DICOM directory dicom\_dir = os.path.expanduser("~/Documents/caoa/Lakehead/Final\_Project/Research\_Papers/ADNI\_Dataset/ADNI2")  # Add file paths to metadata **def** **get\_file\_path**(subject\_id):  """Retrieve all .dcm files for a given subject."""  **for** root, dirs, files **in** os.walk(dicom\_dir): # Recursively walk through directories  **if** subject\_id **in** root: # Check if subject\_id is part of the current path  dicom\_files = [os.path.join(root, f) **for** f **in** files **if** f.endswith(".dcm")]  **if** dicom\_files:  **return** dicom\_files # Return all .dcm files for this subject  **return** **None**  # Debugging: Print all subjects and DICOM directory content print("Subjects in metadata:", metadata["Subject"].unique()) print("Folders in DICOM directory:", os.listdir(dicom\_dir))  # Apply function to find paths metadata["FilePaths"] = metadata["Subject"].apply(get\_file\_path)  # Drop rows with missing paths metadata = metadata.dropna(subset=["FilePaths"]) print(metadata.head()) |
| --- |

Due to the lack of resources we had we are not abel to train the model on the total amount of images each patient had, so we took the decision of selecting the most recent mri from each patient to create the dataset, we added this to the dataset as a column called “Selected File”.

| #Select only one mri image per subject and add it to a new column called select file  **def** **select\_dicom\_file**(dicom\_files):  **if** dicom\_files:  **return** dicom\_files[0]   **return** **None**  # Apply selection logic metadata["SelectedFile"] = metadata["FilePaths"].apply(select\_dicom\_file)  # Drop rows without a selected file metadata = metadata.dropna(subset=["SelectedFile"])  # Verify the selected files print(metadata[["Subject", "SelectedFile"]].head()) |
| --- |

Now still in the image pre-processing, all the images were in a dicom format, the transformer only takes png or jpeg images so we had to convert all the images to png.

| **def** **convert\_dicom\_to\_png**(dicom\_path, save\_path):  """Convert a DICOM file to a PNG image."""  ds = pydicom.dcmread(dicom\_path)  image = ds.pixel\_array # Extract pixel data  normalized\_image = (image - np.min(image)) / (np.max(image) - np.min(image)) # Normalize (0,1)  Image.fromarray((normalized\_image \* 255).astype(np.uint8)).save(save\_path) # remove floating values and convert to image from array |
| --- |

We splitted the dataset into three different folders train, testing and validation.

| # Create subdirectories for each class **for** split **in** [train\_dir, val\_dir, test\_dir]:  **for** group **in** metadata["Group"].unique():  os.makedirs(os.path.join(split, str(group)), exist\_ok=**True**)  # Split the data into train, val, and test sets train\_metadata, test\_metadata = train\_test\_split(metadata, test\_size=0.2, stratify=metadata["Group"], random\_state=42) train\_metadata, val\_metadata = train\_test\_split(train\_metadata, test\_size=0.25, stratify=train\_metadata["Group"], random\_state=42)  # Function to copy images to respective directories **def** **copy\_images**(metadata\_split, target\_dir):  **for** index, row **in** metadata\_split.iterrows():  # Adjust image name based on naming convention  image\_name = f"{row['Subject']}\_0.png" # Use '\_0.png' if multiple images per subject  source\_path = os.path.join(source\_images\_dir, image\_name)  target\_path = os.path.join(target\_dir, str(row["Group"]), image\_name)  **if** os.path.exists(source\_path):  shutil.copy(source\_path, target\_path)  **else**:  print(f"Image not found: {source\_path}") # Debugging output for missing images  # Copy images to train, val, and test directories copy\_images(train\_metadata, train\_dir) copy\_images(val\_metadata, val\_dir) copy\_images(test\_metadata, test\_dir) |
| --- |

We do the final checks to make sure each image is normalize and the correct size for the vision transformer after that we proceed to load the dataset and create the data loaders.

| **from** torchvision **import** transforms **from** torchvision.datasets **import** ImageFolder **from** torch.utils.data **import** DataLoader  # Define image transformations image\_transforms = transforms.Compose([  transforms.Resize((224, 224)), # Resize to the input size of ViT  transforms.ToTensor(),  transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]) # Normalize to [-1, 1] ])  # Load datasets train\_dataset = ImageFolder(train\_dir, transform=image\_transforms) val\_dataset = ImageFolder(val\_dir, transform=image\_transforms) test\_dataset = ImageFolder(test\_dir, transform=image\_transforms)  # Create dataloaders train\_loader = DataLoader(train\_dataset, batch\_size=16, shuffle=**True**) val\_loader = DataLoader(val\_dataset, batch\_size=16) test\_loader = DataLoader(test\_dataset, batch\_size=16) |
| --- |

We proceed to load the vision transformer in this case we are using a pre trained vision transformer from the hugging face library called google/vit-base-patch16-224, we set our number of labels to three since we only have three classes in this dataset.

| **from** transformers **import** ViTForImageClassification **import** torch  # Load the pretrained Vision Transformer model = ViTForImageClassification.from\_pretrained(  "google/vit-base-patch16-224",  num\_labels=3, # Set the number of classes to 3  ignore\_mismatched\_sizes=**True** # Ignore mismatched layers ) |
| --- |

Finally we setup the ADAM optimizer and the crossentropy loss function and we proceed to train our vision transformer for 10 epochs.

| # Define optimizer optimizer = Adam(model.parameters(), lr=5e-5) # Adjust learning rate as needed  # Define loss function criterion = CrossEntropyLoss()  num\_epochs = 10 # Number of training epochs **for** epoch **in** range(num\_epochs):  print(f"\nEpoch {epoch + 1}/{num\_epochs}")    # Training phase  model.train()  train\_loss = 0  train\_progress = tqdm(train\_loader, desc="Training", leave=**False**) # Progress bar for training  **for** images, labels **in** train\_progress:  images, labels = images.to(device), labels.to(device)   # Forward pass  outputs = model(pixel\_values=images).logits  loss = criterion(outputs, labels)   # Backward pass and optimization  optimizer.zero\_grad()  loss.backward()  optimizer.step()   train\_loss += loss.item()  train\_progress.set\_postfix({"Loss": f"{loss.item():.4f}"}) # Update progress bar   # Validation phase  model.eval()  val\_loss = 0  correct = 0  total = 0  val\_progress = tqdm(val\_loader, desc="Validating", leave=**False**) # Progress bar for validation  **with** torch.no\_grad():  **for** images, labels **in** val\_progress:  images, labels = images.to(device), labels.to(device)  outputs = model(pixel\_values=images).logits  loss = criterion(outputs, labels)  val\_loss += loss.item()   # Calculate accuracy  \_, predicted = torch.max(outputs, 1)  correct += (predicted == labels).sum().item()  total += labels.size(0)   val\_progress.set\_postfix({"Val Loss": f"{loss.item():.4f}"}) # Update progress bar   val\_accuracy = correct / total  print(f"Epoch {epoch + 1}/{num\_epochs}, Train Loss: {train\_loss / len(train\_loader):.4f}, "  f"Validation Loss: {val\_loss / len(val\_loader):.4f}, Validation Accuracy: {val\_accuracy:.4f}") |
| --- |

## **Morphometric Analysis**

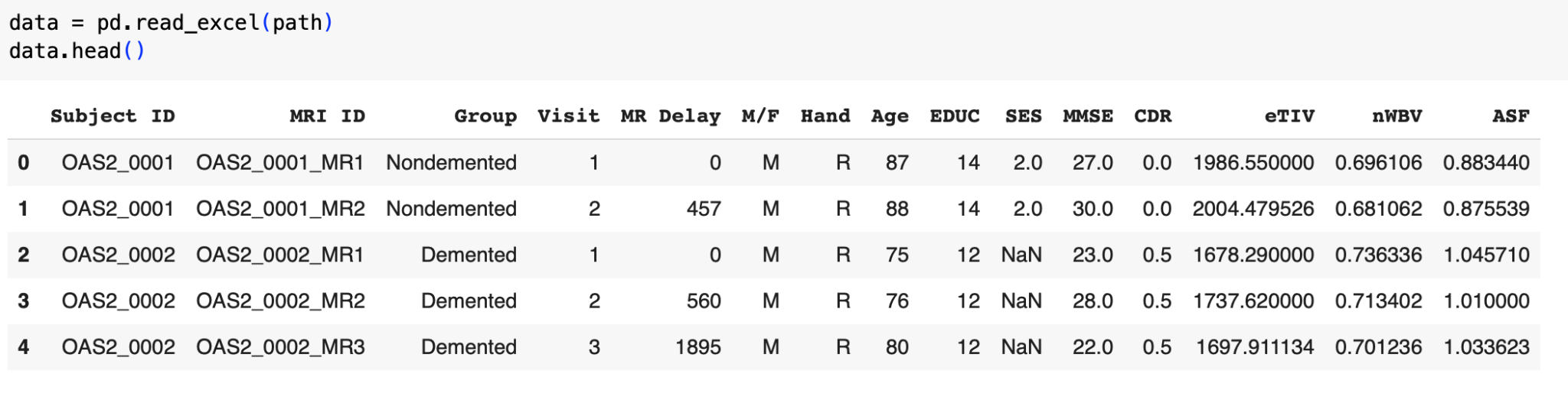
To understand the impact of the brain regions on Alzheimer’s, this study has used logistic regression for this area of study. We selected logistic regression primarily for its interpretability; the models’ coefficients give us an easy, readable, and straightforward means to quantify the influence of each brain region on the model’s prediction. In order to summarize the overall importance of each feature in all diagnostic classes, we computed the average absolute coefficient for each feature. This metric aggregates the absolute values of the coefficients across the different classes, yielding a single value for each brain region that reflects its overall contribution to the model’s predictions. A higher average absolute coefficient indicates that a feature plays a more significant role in influencing the model’s decision, while a lower value suggests a relatively minor impact.

## **Demographic Analysis**

For demographic analysis, we used the OASIS 2 demographic dataset.

Dataset link: <https://docs.google.com/spreadsheets/d/1saOvydyB7F6KQ_B1mqOPl1wIoMQ4ZGTS/edit?usp=sharing&ouid=112086475870669427226&rtpof=true&sd=true>

This is what the dataset looks like:



Group refers to diagnosis of the patient. The features important for our analysis are explained below:

**Demographics**

Gender (M/F), Handedness (Hand), Age, Education (Educ), socioeconomic status (SES)

Education codes correspond to the following levels of education: 1: less than high school graduate, 2: high school graduate, 3: some college, 4: college graduate, 5: beyond college.

**Clinical**

Mini-Mental State Examination (MMSE), Clinical Dementia Rating (CDR; 0=

non-demented; 0.5 – very mild dementia; 1 = mild dementia; 2 = moderate dementia).

All participants with dementia (CDR >0) were diagnosed with probable AD.

**Derived anatomical volumes**

1. Estimated total intracranial volume (eTIV) (mm3): an approximation of the total volume inside the skull
2. Atlas scaling factor (ASF): a term used in neuroimaging, particularly in the context of brain volume studies, to adjust or normalize individual brain measurements.
3. Normalized whole brain volume (nWBV): a measure used in neuroimaging to represent the volume of brain tissue (both gray and white matter) as a proportion of the total intracranial volume (eTIV).

**Preprocessing**

We began by dropping columns with a majority of null values and imputing null rows with meal values.

| clean\_null\_values = data.dropna(axis=1, thresh=len\_data/2) **for** i **in** range(len(col\_null)):  **try**:  data[col\_null[i]].fillna(data[col\_null[i]].mean(), inplace=**True**)  **except**:  **pass** |
| --- |

Then, we encoded columns with non-numeric values.

| #encoding columns with string values encoder = LabelEncoder() data['M/F'] = encoder.fit\_transform(data['M/F'])  mapping = dict(zip(encoder.classes\_, range(len(encoder.classes\_)))) print("Mapping:", mapping)  data['Hand'] = encoder.fit\_transform(data['Hand'])  mapping = dict(zip(encoder.classes\_, range(len(encoder.classes\_)))) print("Mapping:", mapping)  data['Group'] = encoder.fit\_transform(data['Group'])  mapping = dict(zip(encoder.classes\_, range(len(encoder.classes\_)))) print("Mapping:", mapping) |
| --- |

As a result, the features were encoded as follows:

Mapping: {'F': 0, 'M': 1}

Mapping: {'R': 0}

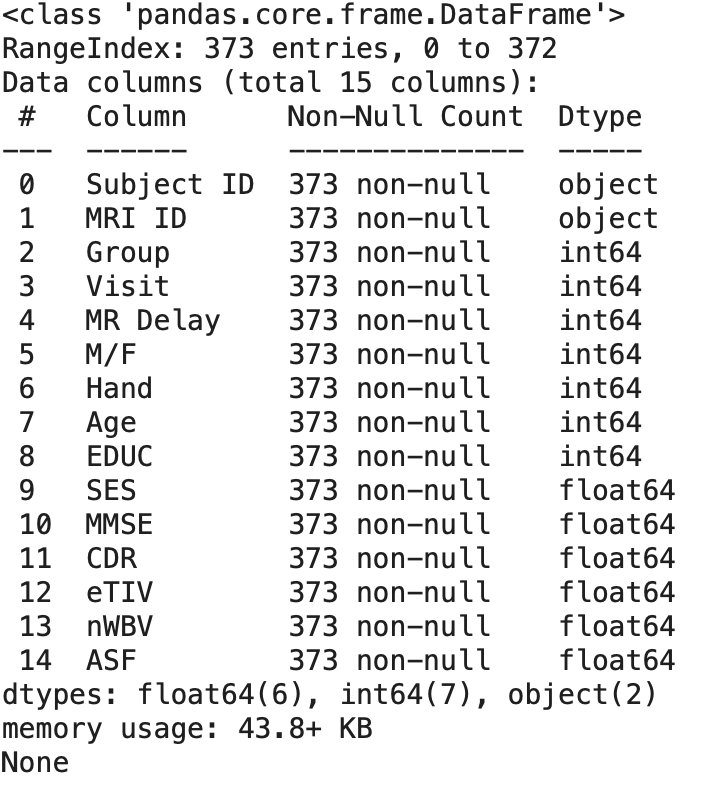
Mapping: {'Converted': 0, 'Demented': 1, 'Nondemented': 2}

**Exploratory Data Analysis**

We conducted EDA to understand the dataset further.

| print(data.info()) |
| --- |

This showed the number of entries as well as the types of features in the dataset. This was the output:

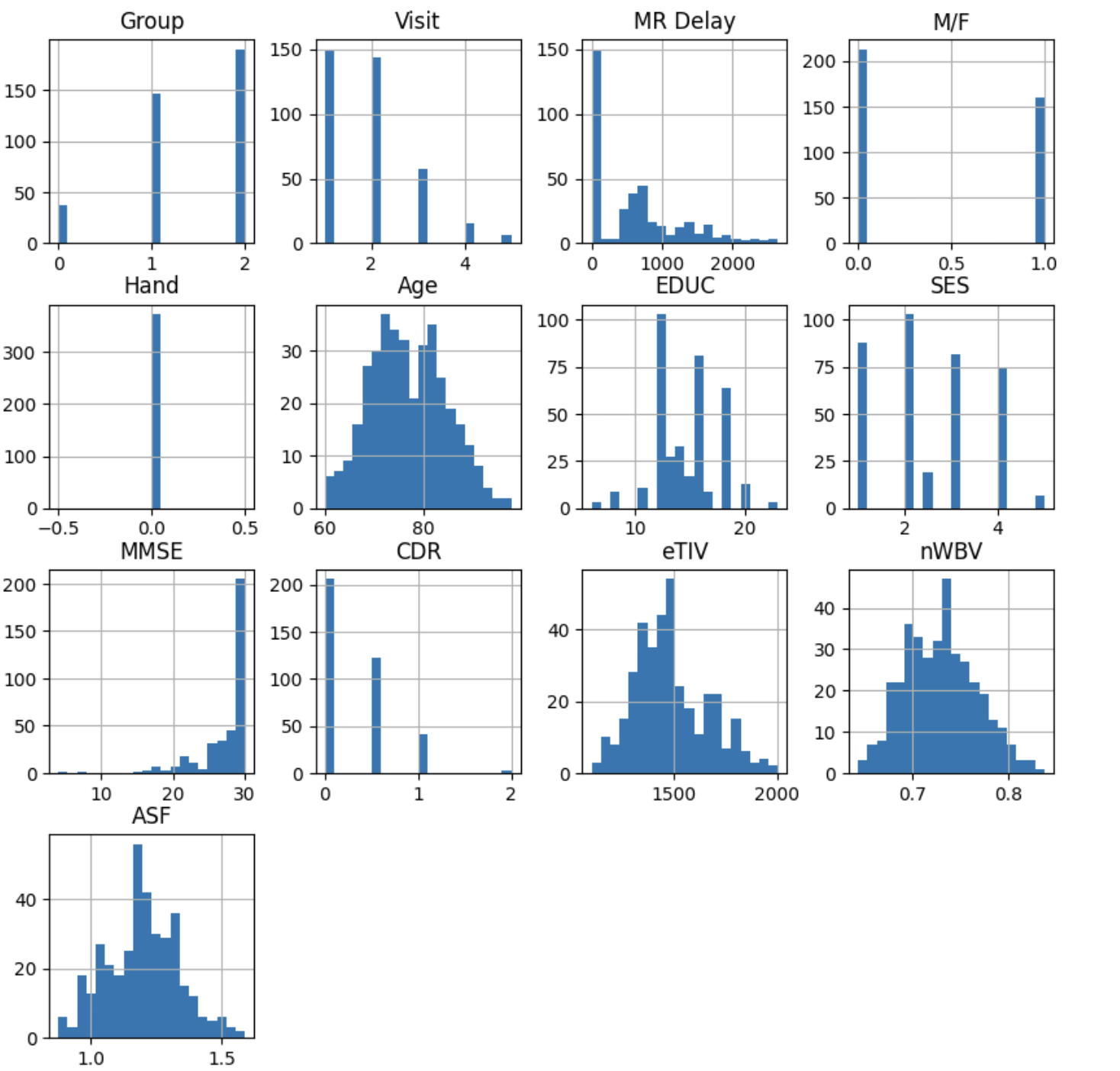


This shows the dataset has 373 entries with data types of object, integer and floating point. Moreover, there are 15 features.

The numerical dataset was visualized using histogram.

| numerical = data.select\_dtypes(include=['number']).columns data[numerical].hist(figsize=(10, 10), bins=20) plt.show() |
| --- |

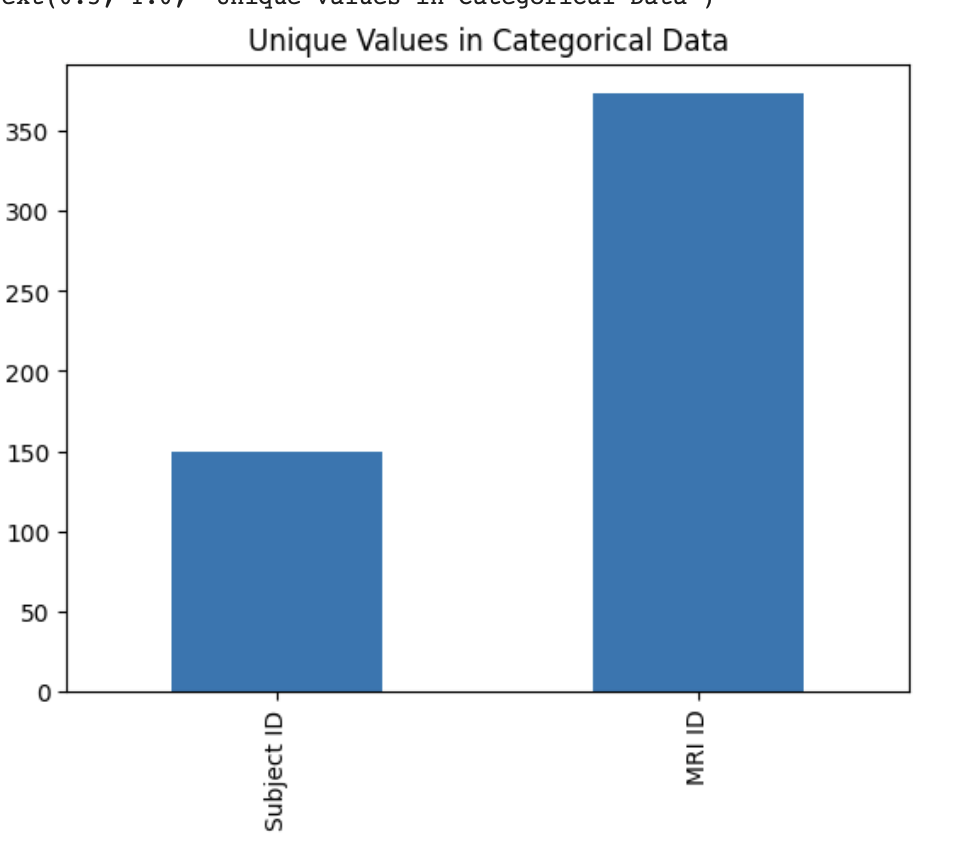
This was the output:



It showed that the ASF, Age, Education, eTIV and nWBV features were normally distributed.

We also explored the unique subject ID to see how many patients were in this dataset.

| data[categorical].nunique().plot(kind='bar') plt.title("Unique Values in Categorical Data") |
| --- |



**Models**

Two main functions were defined. The ‘perf’ function is used to output the accuracy and confusion matrix for each model. The ‘visualize’ function on the other hand is used to show how each coefficient is impacting the prediction. The ‘coolwarm’ cmap means that the darkest color has the highest impact on the prediction. It also outputs the coefficients greater than 0 and odd ratios greater than 1 to see the most impact.

| #this function outputs the performance of a model **def** **perf**(test\_lab, pred\_lab):  accuracy = accuracy\_score(test\_lab, pred\_lab)  print(f"Accuracy:" + str(accuracy))   print("\n")   class\_report = classification\_report(test\_lab, pred\_lab)  print("Classification Report: \n" + str(class\_report))  #this visualizes how each feature is impacting a model in prediction **def** **visualize**(coeff):  coeff.set\_index('Feature', inplace=**True**)   plt.figure(figsize=(8, 6))  sns.heatmap(coeff, annot=**True**, cmap='coolwarm', fmt=".2f", cbar\_kws={'label': 'Value'})  plt.title('Feature Impact Heatmap')  plt.show()   positive\_coeff = coeff[coeff['Coefficient'] > 0]  positive\_ratios = coeff[coeff['Odd Ratios'] > 1]   print("\n")  print("Factors contributing to likelihood according to coeff: ")  print(tabulate(positive\_coeff, headers='keys', tablefmt='grid'))  print("\n")  print("Factors contributing to likelihood according to odd ratios:")  print(tabulate(positive\_ratios, headers='keys', tablefmt='grid')) |
| --- |

Each of the following sections show how each model was trained then visualizes the accuracy, classification report and the coefficients. After training logistic regression, we realised a regularization was used to not overfit on the limited data so we trained all models with regularization parameters.

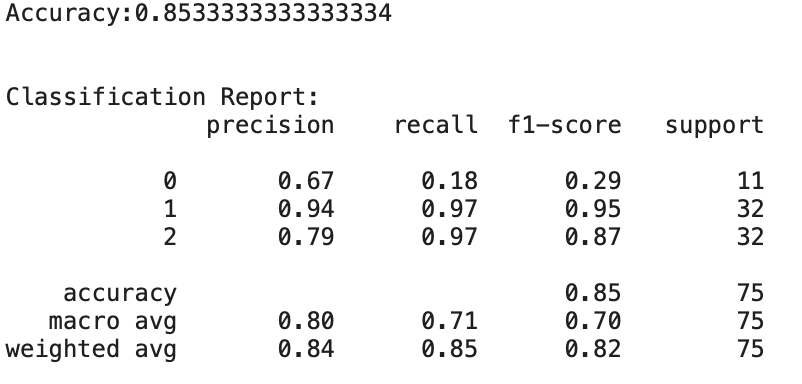
**Logistic Regression**

‘sklearn’ was used to train the models. The datasets are also scaled.

| X = data[['MR Delay', 'M/F', 'Hand',  'Age', 'EDUC', 'SES', 'MMSE', 'CDR', 'eTIV', 'nWBV', 'ASF']]  y = data[['Group']]  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  model = LogisticRegression(max\_iter=1100)  #using standard scaler as features with different scales can distort the importance scaler = StandardScaler() X\_train\_scaled = scaler.fit\_transform(X\_train) X\_test\_scaled = scaler.fit\_transform(X\_test)  model.fit(X\_train\_scaled, y\_train)  y\_pred = model.predict(X\_test\_scaled) |
| --- |

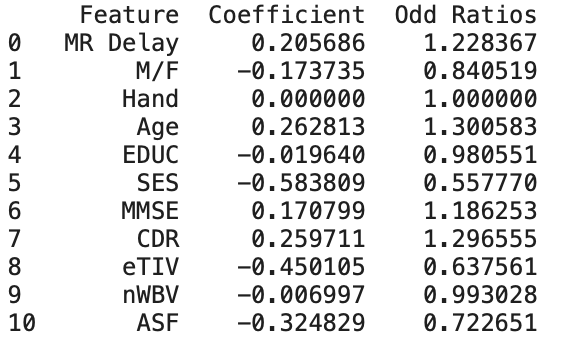
Printing the accuracy and classification report of logistic regression:

| perf(y\_test,y\_pred) |
| --- |

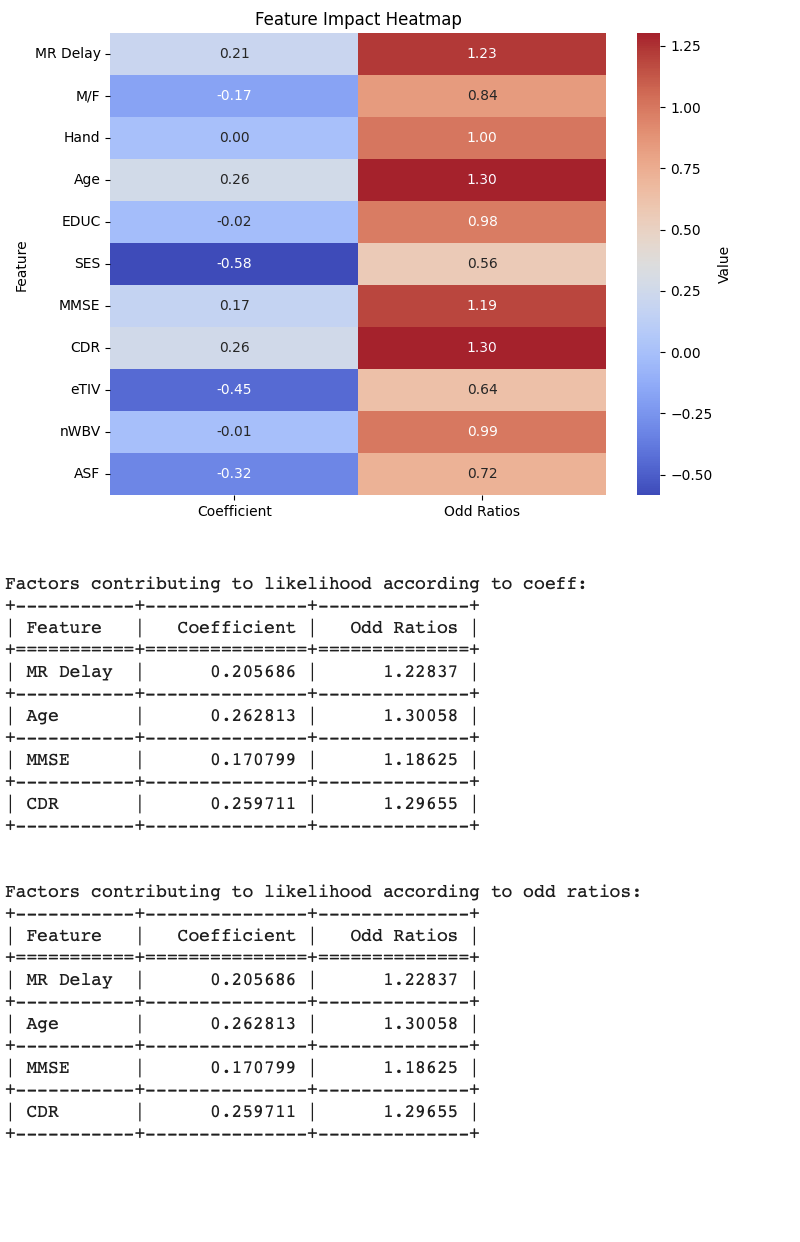


Visualizing coefficients and odd ratios:

| coeff = pd.DataFrame({  'Feature': X.columns.tolist(),  'Coefficient': model.coef\_[0] })  coeff['Odd Ratios'] = np.exp(coeff['Coefficient']) print(coeff) |
| --- |

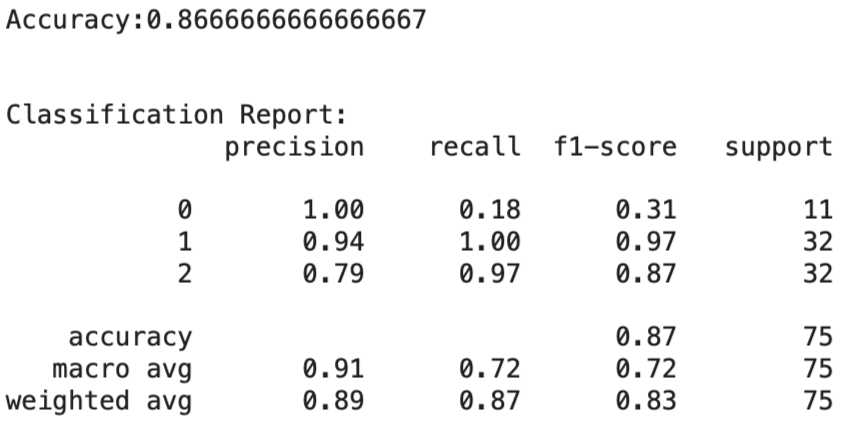


| visualize(coeff) |
| --- |

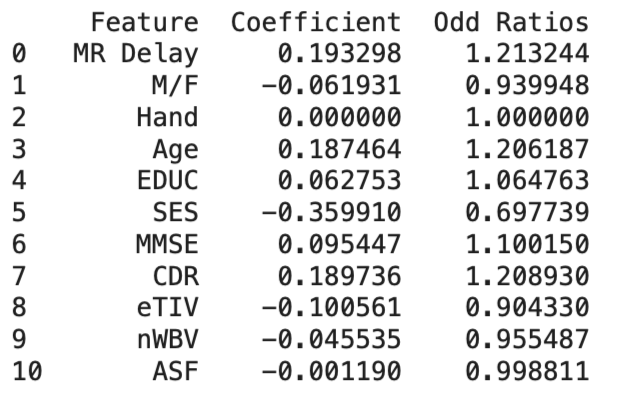


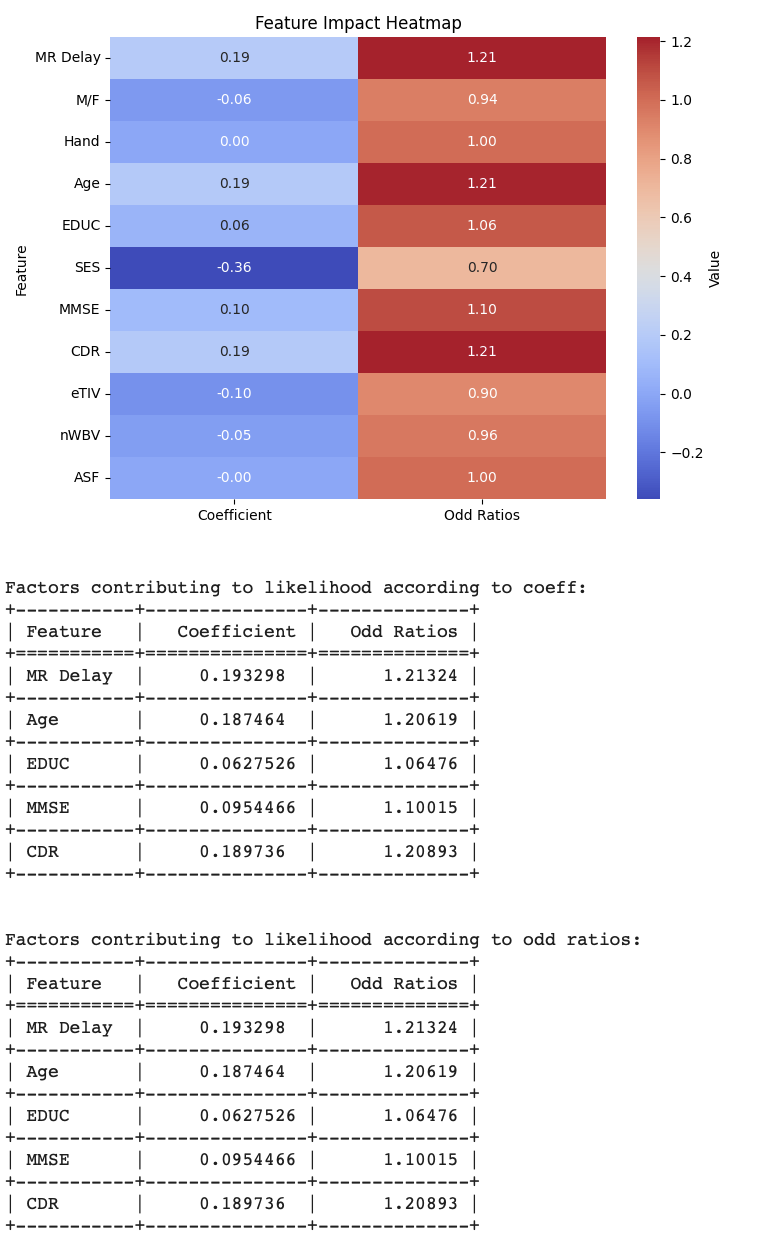
**Logistic Regression with regularization**

| model = LogisticRegression(penalty='l2', C=0.1, max\_iter=1000) scaler = StandardScaler() X\_train\_scaled = scaler.fit\_transform(X\_train) X\_test\_scaled = scaler.fit\_transform(X\_test)  model.fit(X\_train\_scaled, y\_train)  y\_pred = model.predict(X\_test\_scaled) perf(y\_test,y\_pred) |
| --- |



| coeff = pd.DataFrame({  'Feature': X.columns.tolist(),  'Coefficient': model.coef\_[0] })  coeff['Odd Ratios'] = np.exp(coeff['Coefficient']) print(coeff) visualize(coeff) |
| --- |

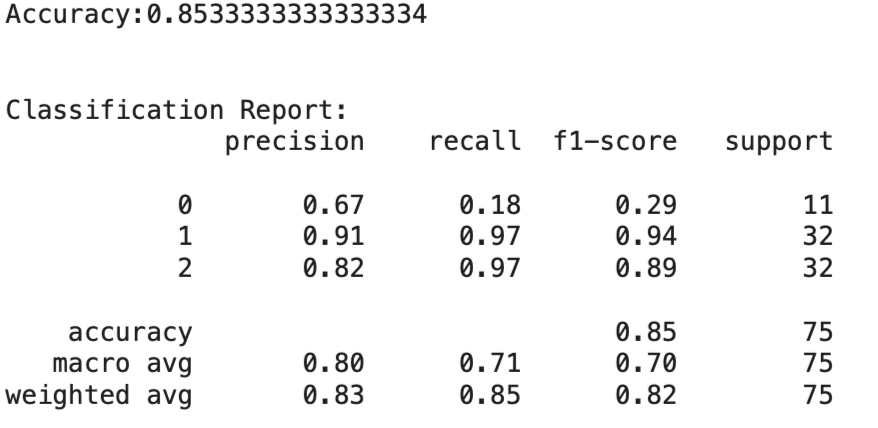




**Gradient Boosting with regularization**

| model = GradientBoostingClassifier(n\_estimators=100, learning\_rate=0.05, max\_depth=3, random\_state=42) scaler = StandardScaler() X\_train\_scaled = scaler.fit\_transform(X\_train) X\_test\_scaled = scaler.fit\_transform(X\_test)  model.fit(X\_train\_scaled, y\_train)  y\_pred = model.predict(X\_test\_scaled)  perf(y\_test,y\_pred) |
| --- |

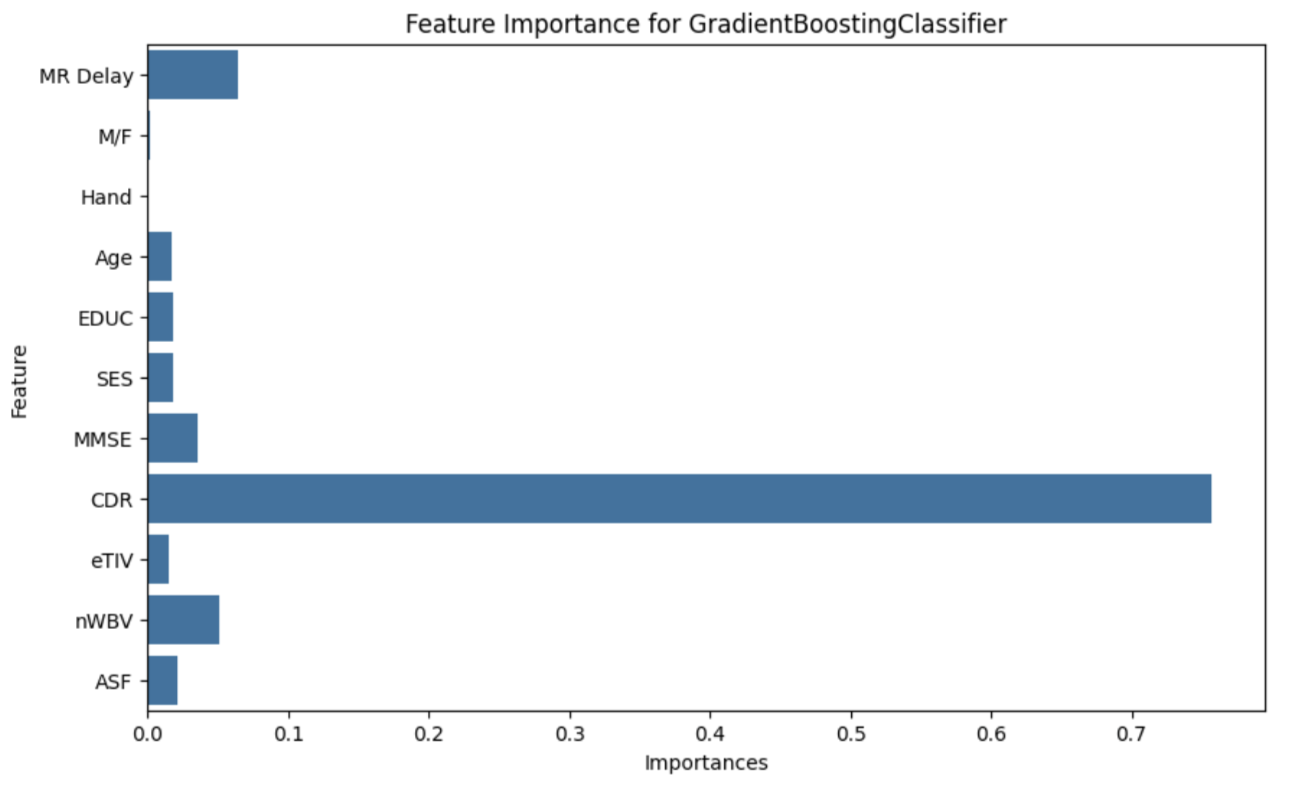
|  |
| --- |



Since gradient boosting doesn’t have a ‘coeff’ attribute due to being a tree-based model, we used feature importance to understand it.

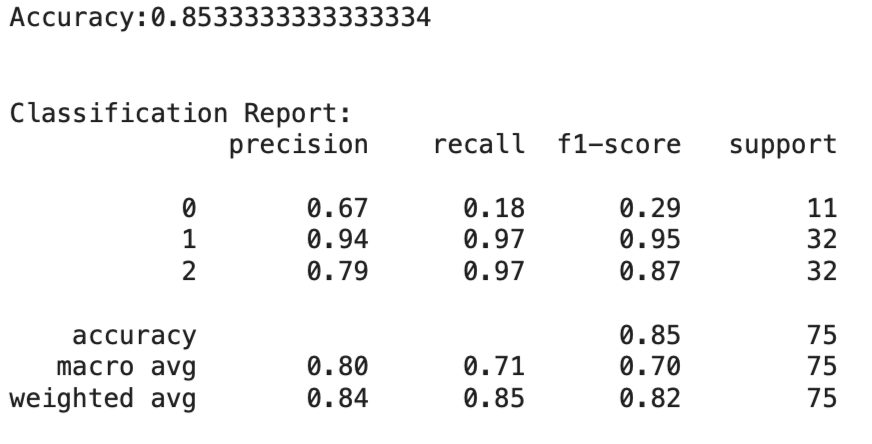
| print(model.feature\_importances\_) |
| --- |

| feature\_imp = pd.DataFrame({  'Feature': X.columns.tolist(),  'Importances': model.feature\_importances\_ })  plt.figure(figsize=(10, 6)) sns.barplot(x='Importances', y='Feature', data=feature\_imp) plt.title('Feature Importance for GradientBoostingClassifier') plt.show() |
| --- |

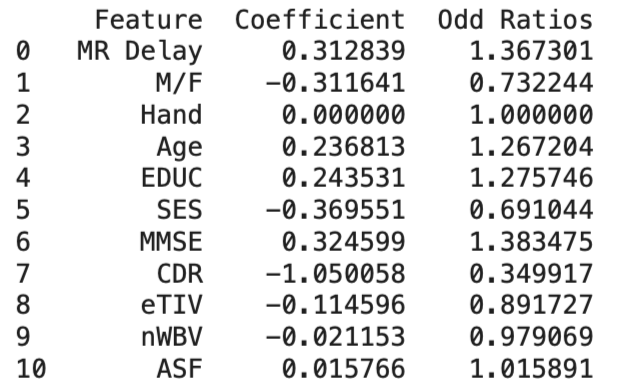


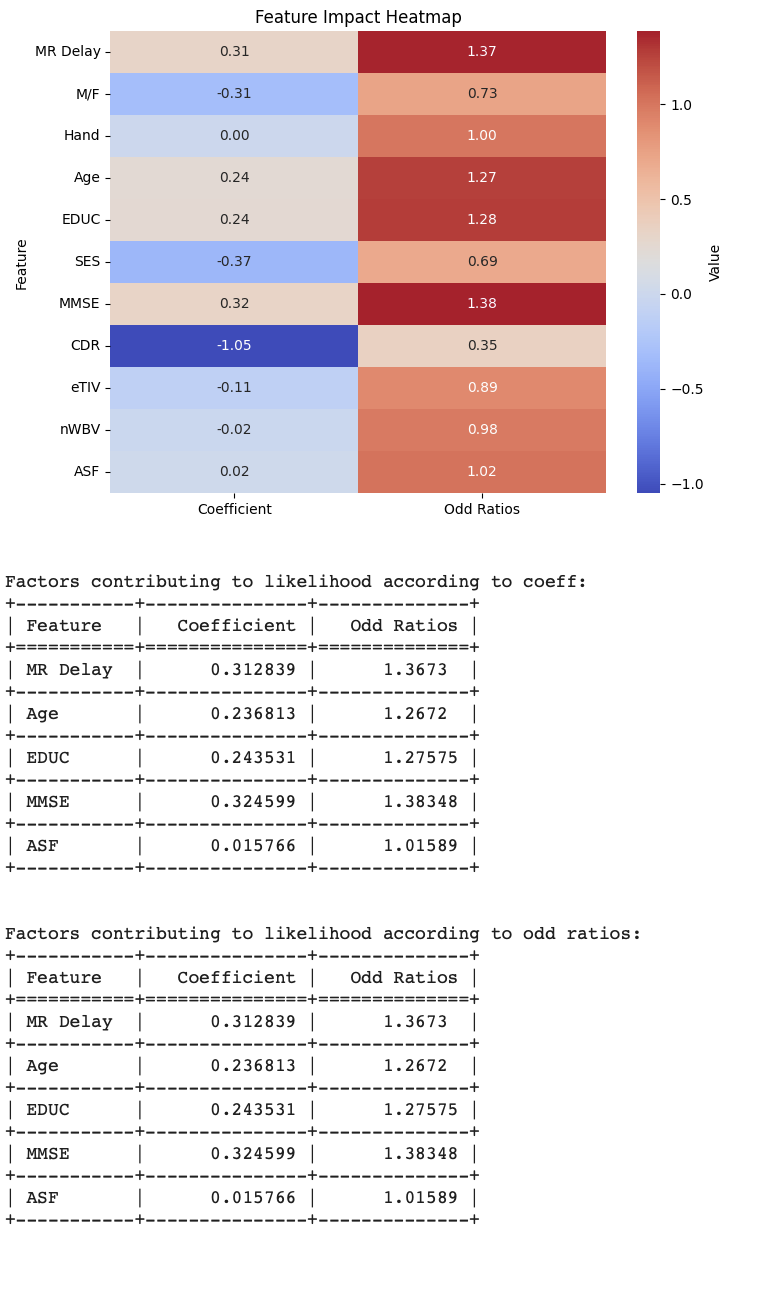
**Support Vector Machine with Regularization**

| **from** sklearn.svm **import** SVC  model = SVC(kernel='linear', C=0.1) scaler = StandardScaler() X\_train\_scaled = scaler.fit\_transform(X\_train) X\_test\_scaled = scaler.fit\_transform(X\_test)  model.fit(X\_train\_scaled, y\_train)  y\_pred = model.predict(X\_test\_scaled) perf(y\_test,y\_pred) |
| --- |

****

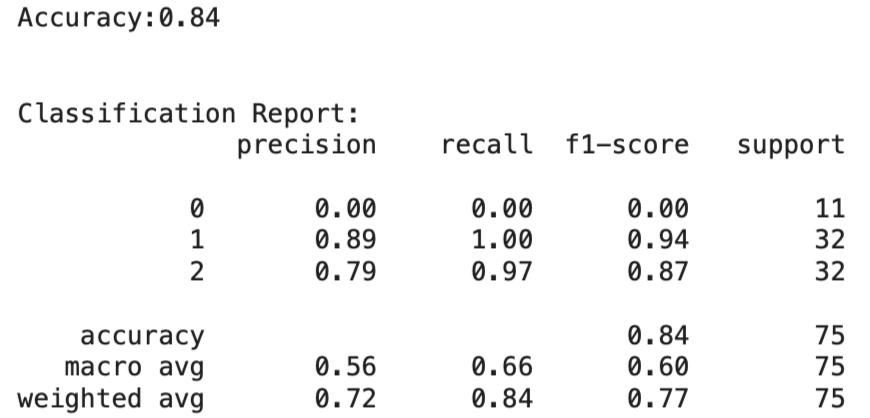
| coeff = pd.DataFrame({  'Feature': X.columns.tolist(),  'Coefficient': model.coef\_[0] })  coeff['Odd Ratios'] = np.exp(coeff['Coefficient']) print(coeff) visualize(coeff) |
| --- |

****

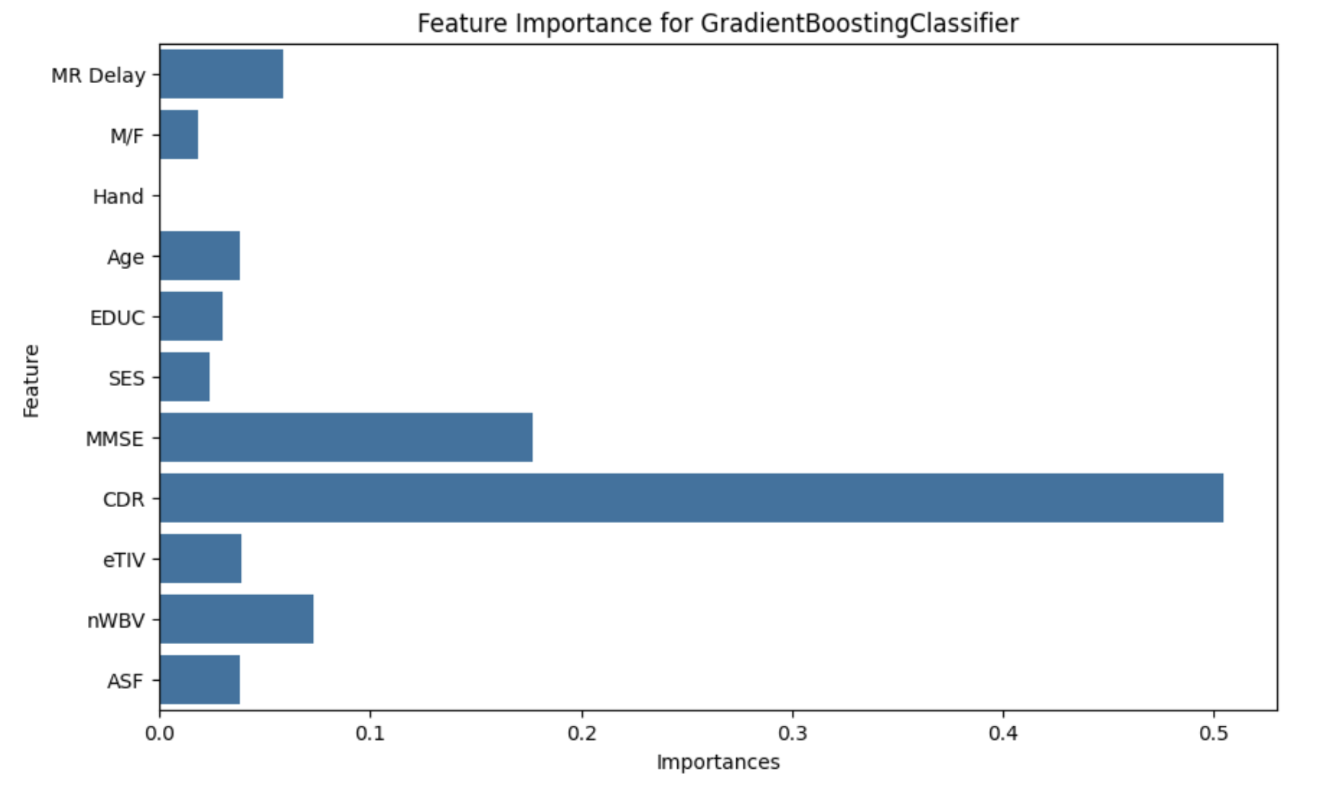
****

**Random Forest with Regularization**

| **from** sklearn.ensemble **import** RandomForestClassifier model = RandomForestClassifier(n\_estimators=100, max\_depth=10, min\_samples\_split=4, min\_samples\_leaf=2, random\_state=42) scaler = StandardScaler() X\_train\_scaled = scaler.fit\_transform(X\_train) X\_test\_scaled = scaler.fit\_transform(X\_test)  model.fit(X\_train\_scaled, y\_train)  y\_pred = model.predict(X\_test\_scaled) perf(y\_test,y\_pred) |
| --- |



| feature\_imp = pd.DataFrame({  'Feature': X.columns.tolist(),  'Importances': model.feature\_importances\_ })  plt.figure(figsize=(10, 6)) sns.barplot(x='Importances', y='Feature', data=feature\_imp) plt.title('Feature Importance for GradientBoostingClassifier') plt.show() |
| --- |

****

# **Results**

This study used a variety of approaches to determine which models are most accurate in predicting Alzheimer’s disease. As mentioned, the models range from basic classifiers to more complex approaches like CNNs and Vision Transformers.

## **Classifiers Results**

*Classifiers With Image Enhancement*

| **Model** | **Accuracy (%)** | **Precision** | **Recall** | **F1-score** |
| --- | --- | --- | --- | --- |
| Logistic Regression | 93.00 | 95.00 | 92.00 | 93.00 |
| Naive Bayes | 48.00 | 36.00 | 39.00 | 34.00 |
| XGBoost | 97.00 | 98.00 | 90.00 | 94.00 |
| Random Forest | 92.00 | 95.00 | 73.00 | 78.00 |

*Classifiers Without Image Enhancement*

| **Model** | **Accuracy (%)** | **Precision** | **Recall** | **F1-score** |
| --- | --- | --- | --- | --- |
| Logistic Regression | 94.00 | 95.00 | 91.00 | 93.00 |
| Naive Bayes | 17.00 | 38.00 | 40.00 | 15.00 |
| XGBoost | 96.00 | 98.00 | 90.00 | 93.00 |
| Random Forest | 73.00 | 61.00 | 46.00 | 48.00 |

Regarding the models without image enhancement, the worst model was Naive Bayes, with a total Test Accuracy of 48%, while XGBoost achieved the highest accuracy of 97% Test Accuracy. Regarding image enhancement, we didn’t see an improvement in the models. For example, in Random Forest, the accuracy decreased from 92% to 73%, which is not what we expected. Although Logistic Regression slightly increased from 93% to 94% overall, the enhancement did not provide the increase in prediction we were expecting.

Regarding the models without image enhancement, the worst-performing model was Naive Bayes, achieving only 17% accuracy, likely due to its assumption of feature independence, which does not hold well for MRI image data. In contrast, XGBoost achieved the highest test accuracy (96%), demonstrating its robustness in handling complex feature relationships, see Table [1](#bookmark=id.44sinio) for detail view on the other models performance.

Surprisingly, image enhancement did not consistently improve model performance. While Logistic Regression saw a slight increase in accuracy from 93% to 94% and Naive Bayes from 17% to 48%, other models showed mixed results. Notably, Random Forest experienced a significant accuracy drop from 92% to 73%, suggesting that image enhancement may have introduced noise or redundant information rather than useful features for this model. See Table [2](#bookmark=id.2jxsxqh) for detail view on the other models performance

These findings indicate that the impact of image enhancement depends on the model architecture. While deep learning models like CNNs often benefit from enhanced feature extraction, traditional machine learning models may not always see improvements. Overall, XGBoost remained the most effective classifier, both with and without image enhancement, making it a strong candidate for detecting Alzheimer’s disease.

## **CNN Results**

*CNN Performance with Data Augmentation*

| **Class** | **Precision** | **Recall** | **F1-score** |
| --- | --- | --- | --- |
| Mild Demented | 99.00 | 90.00 | 94.00 |
| Moderate Demented | 100.00 | 93.00 | 97.00 |
| Non Demented | 96.00 | 100.00 | 98.00 |
| Very Mild Demented | 96.00 | 95.00 | 96.00 |
| **Accuracy: 97.00** | | | |
| Macro Avg | 98.00 | 95.00 | 96.00 |
| Weighted Avg | 97.00 | 97.00 | 97.00 |

*CNN Performance without Data Augmentation*

| **Class** | **Precision** | **Recall** | **F1-score** |
| --- | --- | --- | --- |
| Mild Demented | 95.00 | 90.00 | 93.00 |
| Moderate Demented | 100.00 | 60.00 | 75.00 |
| Non Demented | 98.00 | 97.00 | 97.00 |
| Very Mild Demented | 93.00 | 97.00 | 95.00 |
| **Accuracy: 96.00** | | | |
| Macro Avg | 96.00 | 86.00 | 90.00 |
| Weighted Avg | 96.00 | 96.00 | 96.00 |

The CNN model was evaluated under two different training conditions, and the results are presented in Table [3](#bookmark=id.3j2qqm3) and Table [4](#bookmark=id.1y810tw). As seen in Table [4](#bookmark=id.1y810tw) it shows that the initial CNN model (without augmentation) achieved a 96% validation accuracy. However, the Moderate Demented class exhibited low recall (60%) and F1-score (75%), indicating that the model struggled to classify these classes correctly.

The model performance improved significantly after applying targeted data augmentation Table [3](#bookmark=id.3j2qqm3). The overall accuracy increased to 97%, and more importantly, the recall and F1-score for the Moderate Demented class improved from 60% and 75% to 93% and 97%, respectively. This suggests that the augmentation effectively addressed the class imbalance, enabling the model to better generalize to underrepresented categories.

These results highlight the impact of balanced training data in medical image classification. While the augmentation improved overall performance, further refinements—such as testing different augmentation techniques—could be explored to optimize classification, particularly for borderline cases between dementia stages.

*CNN Validation*

| **Class** | **Precision** | **Recall** | **F1-score** |
| --- | --- | --- | --- |
| Mild Impairment | 99.00 | 99.00 | 100.00 |
| Moderate Impairment | 100.00 | 100.00 | 100.00 |
| No Impairment | 99.00 | 99.00 | 99.00 |
| Very Mild Impairment | 99.00 | 98.00 | 99.00 |
| **Accuracy: 99.00** | | | |
| Macro Avg | 99.00 | 99.00 | 99.00 |
| Weighted Avg | 99.00 | 99.00 | 99.00 |

To evaluate the generalization capabilities of our CNN, we validated it using an independent dataset obtained from Kaggle [15]. Notably, the Kaggle dataset uses different labeling, with classes defined as "Mild Impairment", "Moderate Impairment", "No Impairment", "Very Mild Impairment". As shown in , Table IV t the model achieved an overall accuracy of 99% on this external dataset, outperforming the 97% accuracy obtained on the original test set. These results demonstrate that the model generalizes exceptionally well to new, unseen data, even when the labeling conventions differ.

## **Transformer Results**

*Vision Transformer Performance*

| **Class** | **Precision** | **Recall** | **F1-score** |
| --- | --- | --- | --- |
| CN | 96.00 | 97.00 | 96.00 |
| MCI | 96.00 | 97.00 | 97.00 |
| AD | 93.00 | 89.00 | 91.00 |
| **Accuracy: 96.00%** | | | |

The transformer model was trained on a dataset different from the previous classifiers. Instead of using the Kaggle Alzheimer’s MRI Disease Classification Dataset, we utilized the ADNI dataset, which follows a three-class classification scheme (CN, MCI, AD) rather than four classes.

Table [5](#bookmark=id.2xcytpi) presents the classification performance for each class during testing. The model achieved an overall accuracy of 96%, demonstrating strong predictive performance for Alzheimer’s disease using MRI images. Notably, both the Cognitively Normal (CN) and Mild Cognitive Impairment (MCI) classes achieved high recall (97%), indicating that the model effectively identifies these conditions. However, the Alzheimer’s Disease (AD) class exhibited a slightly lower recall (89%), suggesting some misclassification within this category.

These results suggest that the transformer model can serve as an effective diagnostic tool for Alzheimer’s disease detection.

*Logistic Regression Morphometric Features*

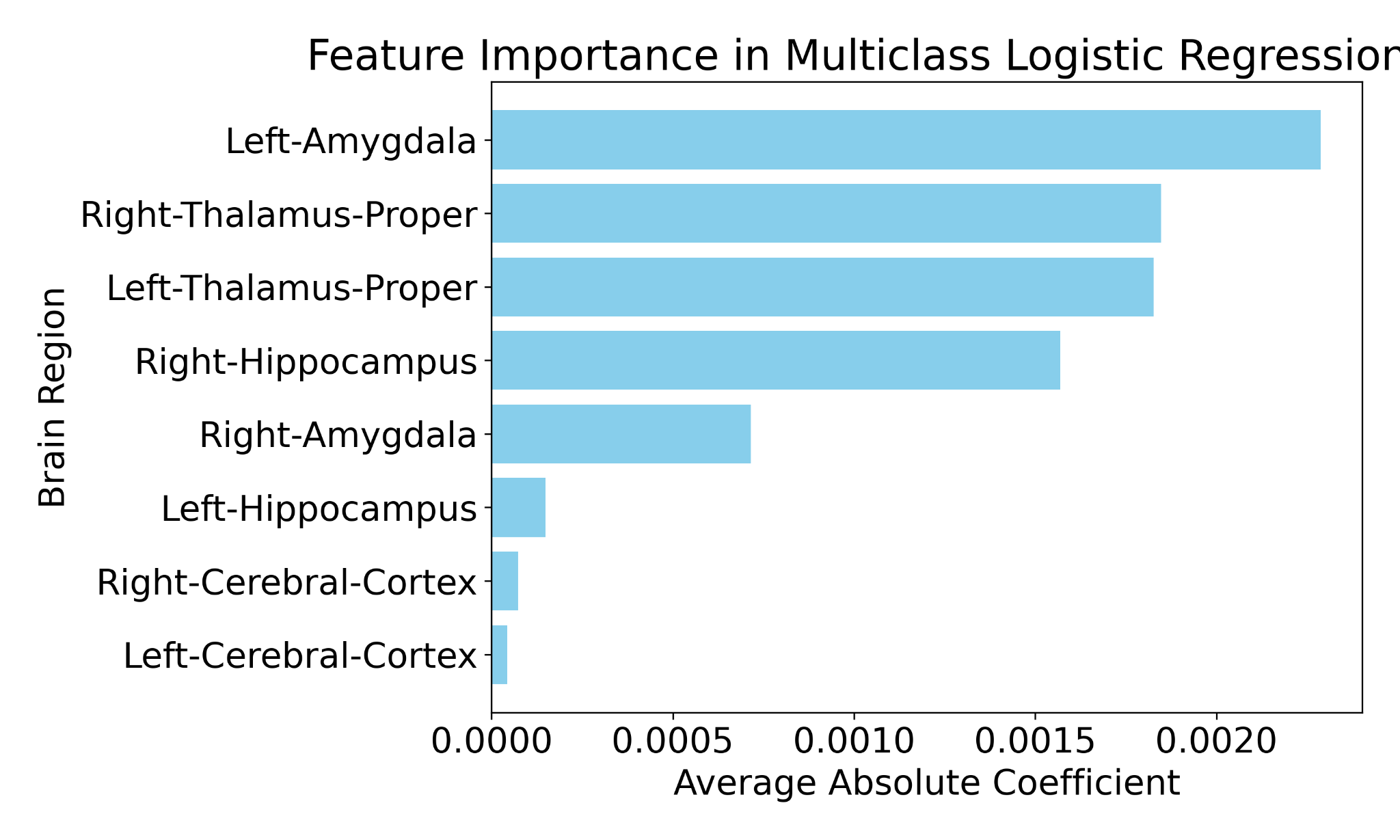
| **Class** | **Precision** | **Recall** | **F1-score** |
| --- | --- | --- | --- |
| AD | 0.00 | 0.00 | 0.00 |
| CN | 88.00 | 78.00 | 82.00 |
| MCI | 62.00 | 100.00 | 77.00 |
| **Accuracy: 75.00** | | | |
| Macro Avg | 50.00 | 59.00 | 53.00 |
| Weighted Avg | 69.00 | 75.00 | 70.00 |

## Moreover, in this study, we validated the Transformer model using two distinct datasets with different labeling schemes. The Kaggle

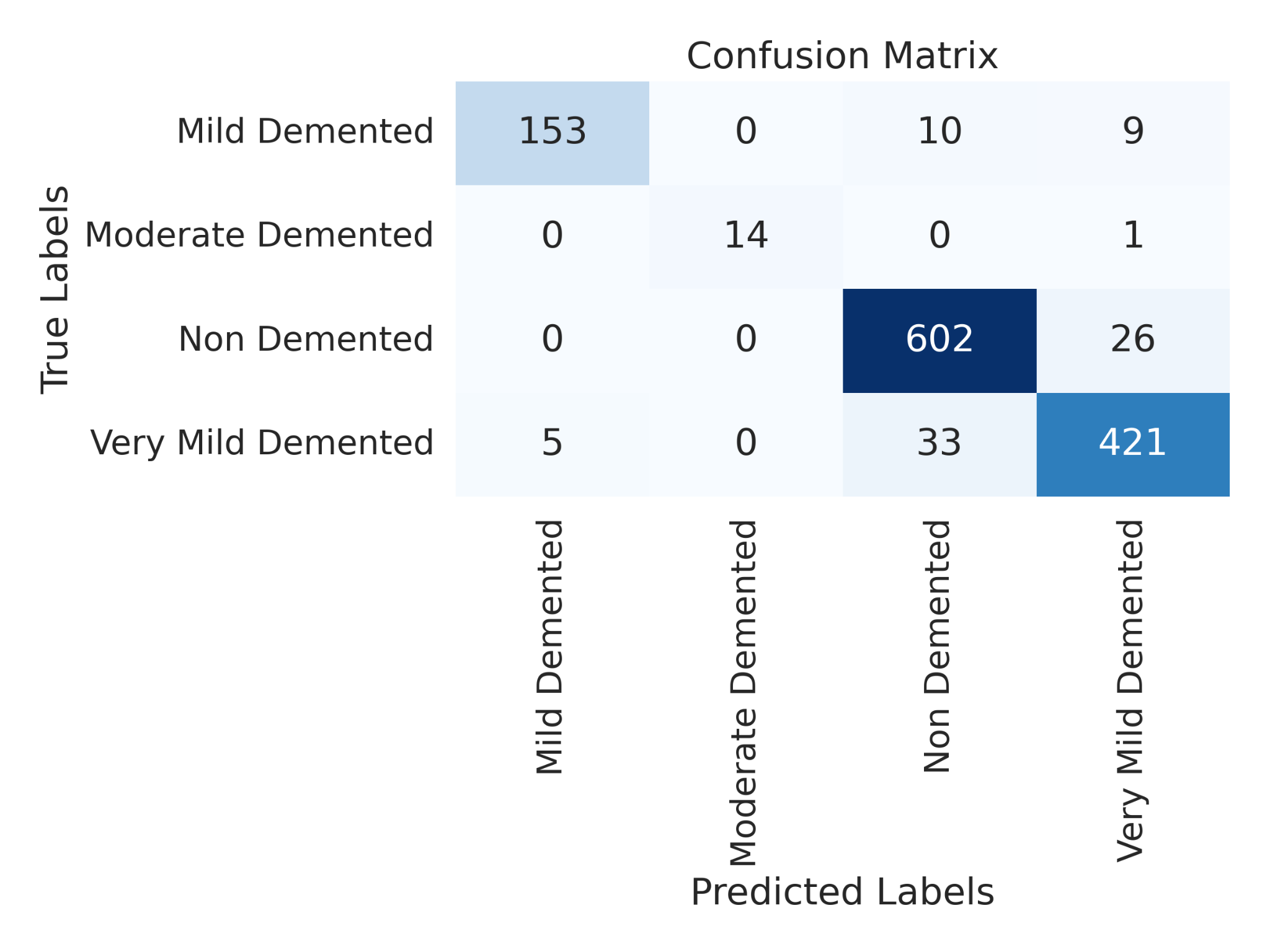
## Alzheimer’s MRI Disease Classification Dataset originally provided four labels (Nondemented, Very Mild Demented, Mild Demented, and Moderated Demented), while the ADNI dataset comprised three labels (CN, MCI, AD). Due to the unavailability of a dataset that exactly matched the ADNI labels, we merged the "Very Mild Demented" and "Mild Demented" classes in the Kaggle dataset to create a three-label configuration. Despite this adjustment, the Transformer achieved only 54.46\% accuracy on the Kaggle data, suggesting that the model struggled with the inherent discrepancies between the datasets. However, when the Transformer was tested on a separate portion of the ADNI dataset that was not used during training or initial testing, it reached an accuracy of 94.21\%. These findings indicate that while the model performs exceptionally well on data similar to its training set, it faces challenges generalizing across datasets with different label definitions and underlying characteristics.

## **Morphometric Features Results**

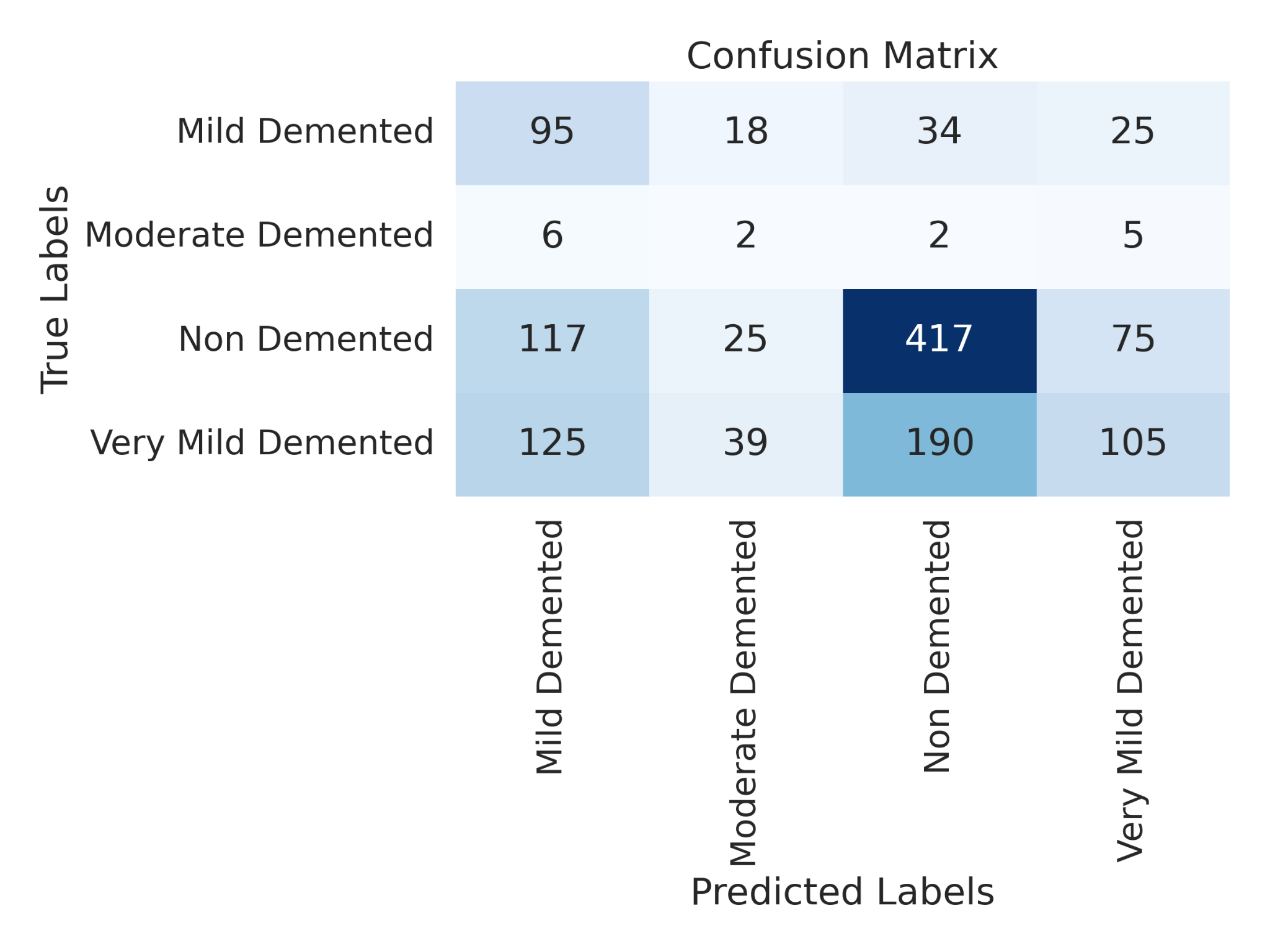
For the morphometric features, we selected a logistic regression model. Although its overall accuracy is only 75% (as shown in Table [6](#bookmark=id.1ci93xb))—a performance likely impacted by our small dataset—our primary focus is on identifying the brain regions that most influence the model’s predictions. As illustrated in Figure [6](#bookmark=id.2bn6wsx), the left amygdala exhibits the highest coefficient, indicating it has the strongest impact on the model’s output, while the left cerebral cortex shows the lowest coefficient, suggesting minimal influence. reports that the amygdala, including the left side, undergoes significant volume loss in early symptomatic AD, correlating with clinical symptom severity. In addition, explores subcortical and cortical atrophy patterns in AD, noting that structures such as the amygdala exhibit distinct atrophy patterns that vary with disease stage and onset age. Although the literature supports our findings, the extremely small dataset—with very few AD patients—limits the generalizability of our results; therefore, these findings should be interpreted with caution.



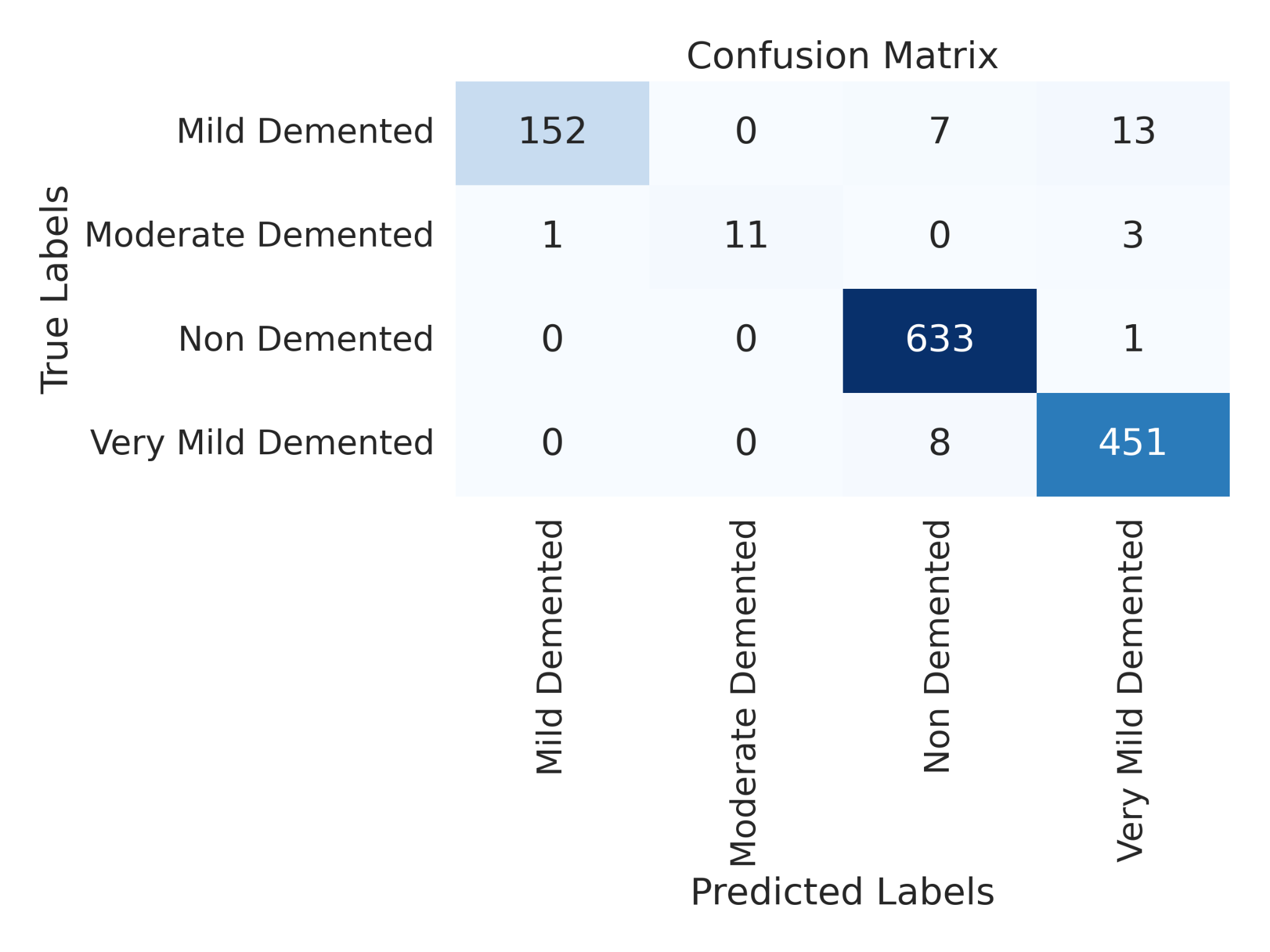
*Average Absolute Coefficient For Brain Regions*



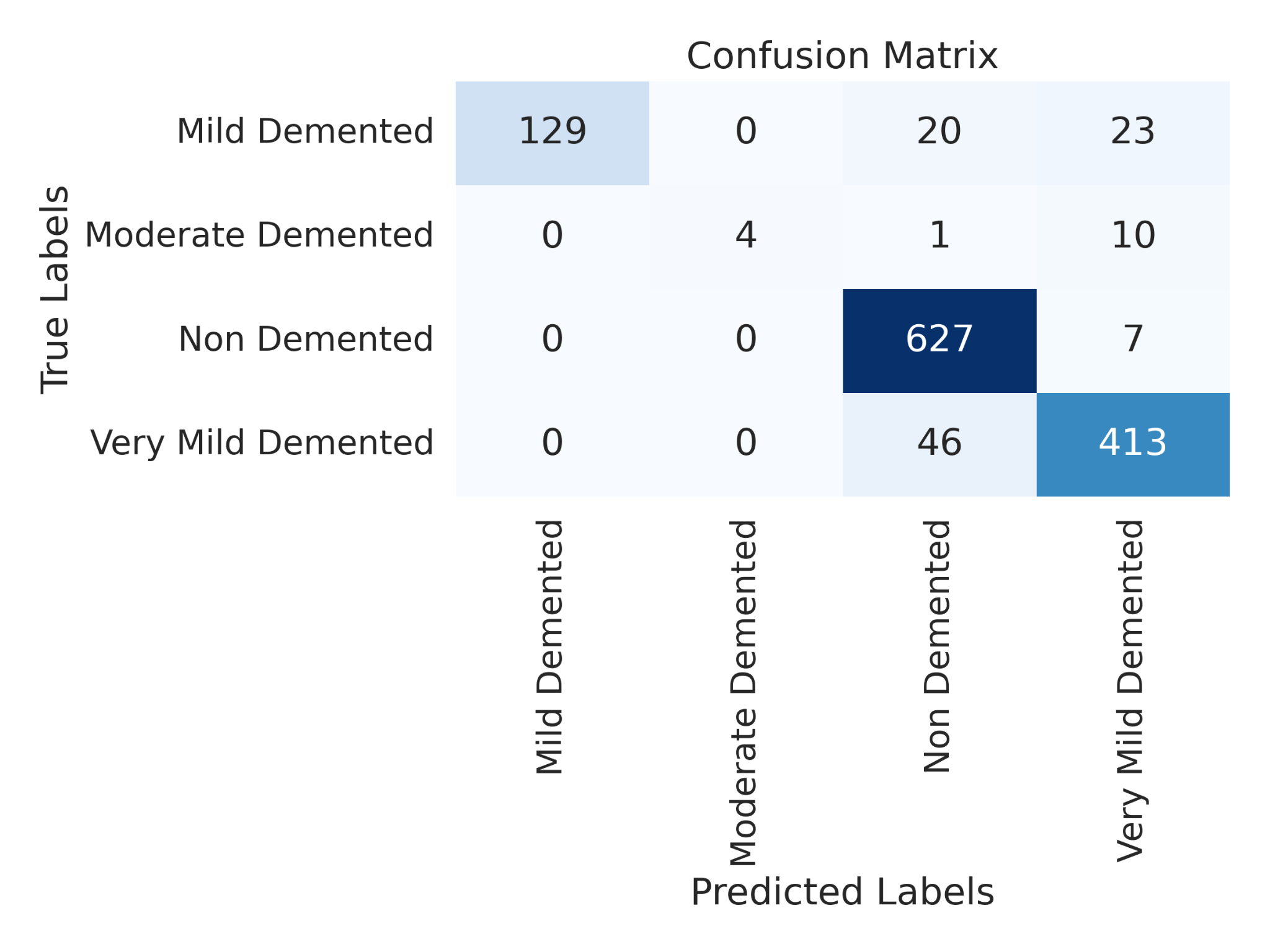
*Logistic Regression Confusion Matrix (Kaggle Dataset)*



*Naive Bayes Confusion Matrix (Kaggle Dataset)*



*XGBoost Confusion Matrix (Kaggle Dataset)*



*Random Forest Confusion Matrix (Kaggle Dataset)*

## **Confusion Matrices**

The model Logistic Regression Figure [7](#bookmark=id.qsh70q) performs well in identifying Non Demented cases and perfectly classifies Moderate Demented samples. However, it shows some confusion between Mild and Very Mild Demented classes, likely due to the subtle differences between these categories. Naive Bayes Figure [8](#bookmark=id.3as4poj) exhibits significant misclassification across all classes. Its simplifying assumptions appear to be insufficient for the complex distinctions required, leading to broad confusion among the dementia stages. XGBoost Figure [9](#bookmark=id.1pxezwc) achieves the highest accuracy with minimal misclassifications. It correctly identifies Non Demented instances almost perfectly and maintains strong performance across the other classes, demonstrating the advantage of its gradient boosting framework in capturing complex patterns. Random Forest Figure [10](#bookmark=id.49x2ik5) also performs robustly, accurately classifying the majority of cases. It shows slight confusion, particularly between Very Mild Demented and Non Demented samples, but overall, its ensemble approach delivers strong results. In summary, XGBoost stands out for its superior performance, while Random Forest follows closely. Logistic Regression provides a decent baseline, and Naive Bayes, due to its oversimplified assumptions, struggles to accurately differentiate the classes.

# **Conclusion**

In Conclusion, our research shows that the choice of the model architecture and the method of training said models play a vital role in accurately predicting diseases, in this case, Alzheimer’s disease. Advanced techniques, specifically XGBoost and deep learning models, usually outperformed simpler classifiers, with XGBoost achieving up to 97% accuracy regardless of the image enhancement made during testing. Although image enhancement did not achieve the expected results in performance, for instance, causing a significant drop in Random Forest accuracy, it did prove beneficial when coupled with data-targeted augmentation in CNNs. This augmentation not only boosted the overall accuracy of the CNN from 96% to 97% but also substantially improved the classification of the underrepresented Moderate Demented class, thereby addressing issues of class imbalance with the dataset.

The promising results from the transformer model, which achieved a 96% accuracy on a distinct three-class dataset, underscore the potential of modern architectures in handling the complexities inherent in medical imaging data. These findings show that although traditional methods may struggle with the nuances of MRI images, modern and advanced algorithms can take advantage of raw and augmented data to provide a more robust diagnostic performance. Future research should be done to explore different alternatives in data augmentation techniques and refine preprocessing strategies further to enhance performance across different model types, ultimately paving the way for more reliable and early detection tools in clinical practice.

# **References**

1. Hippius, Hanns, and Gabriele Neundo¨rfer.“The discovery of Alzheimer’s disease.” Dialogues in clinical neuro- science 5, no. 1 (2003): 101-108.
2. Atri, Alireza. ”The Alzheimer’s disease clinical spectrum: diagnosis and management.” Medical Clinics of North America 103, no. 2 (2019): 263293.
3. Alzheimer’s Association. ”Alzheimer’s Disease Facts and Figures”. Ac- cessed March 9, 2025. [https://www.alz.org/alzheimers-dementia/facts-](http://www.alz.org/alzheimers-dementia/facts-) figures
4. Helaly, Hadeer A., Mahmoud Badawy, and Amira Y. Haikal. ”Deep learning approach for early detection of Alzheimer’s disease.” Cognitive computation 14, no. 5 (2022): 1711-1727.
5. Kavitha, C., Vinodhini Mani, S. R. Srividhya, Osamah Ibrahim Khalaf, and Carlos Andre´s Tavera Romero. ”Early-stage Alzheimer’s disease prediction using machine learning models.” Frontiers in public health 10 (2022): 853294.
6. Uddin, Khandaker Mohammad Mohi, Mir Jafikul Alam, Md Ashraf Uddin, and Sunil Aryal. ”A novel approach utilizing machine learning for the early diagnosis of Alzheimer’s disease.” Biomedical Materials & Devices 1, no. 2 (2023): 882-898.
7. S. Murugan et al., ”DEMNET: A Deep Learning Model for Early Diagnosis of Alzheimer Diseases and Dementia From MR Images,” in IEEE Access, vol. 9, pp. 90319-90329, 2021, doi: 10.1109/AC- CESS.2021.3090474.
8. S. Liu, S. Liu, W. Cai, S. Pujol, R. Kikinis and D. Feng, ”Early diagnosis of Alzheimer’s disease with deep learning,” 2014 IEEE 11th International Symposium on Biomedical Imaging (ISBI), Beijing, China, 2014, pp. 1015-1018, doi: 10.1109/ISBI.2014.6868045.
9. Diogo, Vasco Sa´, Hugo Alexandre Ferreira, Diana Prata,and Alzheimer’s Disease Neuroimaging Initiative. ”Early diagnosis of Alzheimer’s disease using machine learning: a multi-diagnostic, gener- alizable approach.” Alzheimer’s Research & Therapy 14, no. 1 (2022): 107.
10. Majhi, Babita, Aarti Kashyap, Siddhartha Suprasad Mohanty, Sujata Dash, Saurav Mallik, Aimin Li, and Zhongming Zhao. ”An improved method for diagnosis of Parkinson’s disease using deep learning models enhanced with metaheuristic algorithm.” BMC medical imaging 24, no. 1 (2024): 156.
11. Kaggle. ”OASIS Alzheimer’s Detection”. Accessed March 9, 2025. [https://www.kaggle.com/datasets/ninadaithal/imagesoasis](http://www.kaggle.com/datasets/ninadaithal/imagesoasis)
12. Alzheimer’s Disease Neuroimaging Initiative. https://adni.loni.usc.edu.
13. Poulin, S. P., Dautoff, R., Morris, J. C., Barrett, L. F., & Dickerson, B. C. ”Amygdala atrophy is prominent in early Alzheimer’s disease and relates to symptom severity.” *Psychiatry Research: Neuroimaging*, 194(1), 7–13, 2011.
14. Mo¨ller, C., Vrenken, H., Jiskoot, L. C., Versteeg, A., Barkhof, F., & Scheltens, P. ”Different patterns of gray matter atrophy in early- and late-onset Alzheimer’s disease.” *Neurobiology of Aging*, 58, 205–213, 2017.
15. Chugh, L. "Best Alzheimer MRI Dataset 99\% Accuracy." {Kaggle}, accessed March 9, 2025, https://www.kaggle.com/datasets/lukechugh/best-alzheimer-mri-dataset-99-accuracy.