ALZHEIMER'S DISEASE PREDICTION

ADVANCED ANALYSIS PROJECT II

GROUP 09

S 15547 - NISALI NUWANDI
S 15643 - RANUSHI NIMTHARA
S 15618 - YASHIKA ASIRIWARDHANA

1. Abstract

This report summarizes the findings from an exploratory data analysis conducted on the "Alzheimer's Disease Dataset," obtained from the Kaggle website. The primary objective of this analysis was to explore the dataset comprehensively and identify factors associated with Alzheimer's disease, focusing on various demographic and clinical aspects. Through the application of data visualization and analytical techniques, we uncovered significant insights and patterns that deepen our understanding of Alzheimer's disease. The key findings from this exploratory analysis are presented in this report, laying the groundwork for potential predictive modeling efforts.

2. Table of Content

3. List of Figures

4. List of Tables

5. Introduction

"Alzheimer's... it is a barren disease, as empty and lifeless as a desert. It is a thief of hearts and souls and memories."

— Nicholas Sparks

Alzheimer's disease is a progressive and irreversible neurological disorder that gradually erodes memory, cognitive function, and the essence of one's identity. As the most common form of dementia, it affects millions of people worldwide, placing a profound emotional and physical burden on both patients and their caregivers. As the world's population ages, Alzheimer's disease becomes more common and creates a serious threat to public health. Because interventions can help slow the progression of the disease and enhance the quality of life for patients and their families, early and accurate diagnosis is essential for effective management and treatment. By performing an exploratory data analysis on the majority of our variables, we are seeking an understanding of the factors that are linked to the state of Alzheimer's disease using the variables included in the dataset. The study also aims to develop a predictive model that can accurately predict the Alzheimer's disease status of an individual.

6. Description of the Question

Alzheimer's disease often starts slowly and without clear symptoms, making it hard to detect early. Genetic, lifestyle, and environmental factors all play a role, and they can vary widely between individuals. Additionally, symptoms of Alzheimer's can be similar to those of other types of dementia, adding to the challenge of pinpointing specific risk factors. Since Alzheimer's can affect anyone, identifying these risk factors is essential for developing effective prevention strategies. Understanding what contributes to the risk of Alzheimer's disease allows us to implement preventive measures before the condition progresses to a more severe stage. Therefore, we try understanding,

- What are the factors that influence a person to have Alzheimer's disease?

- To come up with a suitable statistical model that can predict whether an individual has a chance of being diagnosed with Alzheimer's disease.

7. Description of the Dataset

The dataset "'Alzheimer's Disease Dataset " was obtained by the Kaggle website.This dataset contains a total of 2149 observations and 35 variables including the response variable. The response variable is the "Diagnosis" which is about the Status of Alzheimer's Disease.

Table 1- Description of the dataset

Variable Name	Description	Variable Name	Description
Age	Age of the patient	**Diagnosis**	Alzheimer's status (0 – No, 1- Yes)
BMI	Body Mass Index of a patient	**Gender**	Gender of patient (0-Male, 1- Feamale)
AlcoholConsumption	Weekly alcohol consumption in units	**Ethnicity**	Ethnicity type (0 – Caucasian, 1-African American, 2-Asian, 3-Other)
PhysicalActivity	Weekly physical activity in hours	**EducationalLevel**	Level of education (0-None, 1-Highschool, 2-Bachelors, 3- Higher)
DietQuality	Diet quality score, ranging from 1 to 10	**Smoking**	Smoking status (0-No, 1-Yes)
SleepQuality	Sleep quality score, ranging from 4 to 10	**FamilyHistory**	Family history of Alzheimer's Disease (0-No, 1-Yes)
SystolicBP	Systolic Blood Pressure ranging from 90 to 180 mmHg	**CardiovascularDiseas**	Presence of Cardiovascullar disease (0-No, 1-Yes)
DiastolicBP	Diastolic Blood Pressure ranging from 60 to 120 mmHg	**Diabetes**	Presence of Diabetes (0-No, 1-Yes)
CholesterolTotal	Total cholesterol levels ranging from 150-300 mg/dL	**Depression**	Presence of depression (0-No, 1-Yes)
CholesterolLDL	Low-density lipoprotein cholesterol levels ranging from 50-200 mg/dL	**HeadInjury**	History of having head injuries (0-No, 1-Yes)
CholesterolHDL	High-density lipoproteincholesterol levels ranging from 20-100 mg/dL	**Hypertension**	Presence of hypertension (0-No, 1-Yes)
CholesterolTriglcerides	Triglycerids cholesterol levels ranging from 50-400 mg/dL	**MemoryComplaints**	Presence of memory complaints (0-No, 1-Yes)
MMSE	Mini-Mental State Examination score low scores indicate cognitive impairment	**BehavioralProblems**	Presence of Behavioral problems (0 – No, 1-Yes)
FunctionalAssessment	Functional Assessment score low scores indicate greater impairment	**Confusion**	Presence of confusion (0-No, 1-Yes)
ADL	Activities of Daily Living score, ranging from 0 to 10	**PersonalityChanges**	Presence of Personality changes (0-No, 1-Yes)
Disorientation	Presence of Disorientation (0-No, 1-Yes)	**DifficultyCompletingTask**	Presence of difficulty in completing tasks (0-No, 1-Yes)
Forgetfulness	Presence of forgetfulness (0-No, 1-Yes)	**PatientID**	A unique identifier assigned to patients

8. Main results in the descriptive analysis

8.1. Data Preprocessing

- There were no missing values or duplicate values.
- MMSE variable was recategorized.
- PatientID, DoctorInCharge and CholesterolTotal variables were dropped.
- CholesterolTotal column was dropped because its individual components—CholesterolLDL, CholesterolHDL, and CholesterolTriglycerides—are retained as separate variables. This allows for a more detailed analysis of the effects of each cholesterol subtype individually, making the total cholesterol variable redundant.
- Mahalanobis test was conducted for outlier detection and no outlier was found.

8.2.Response Variable

Diagnosis

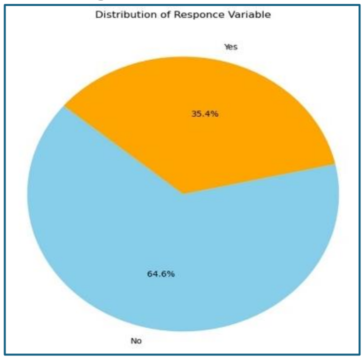


Figure 1-Pie chart of Alzheimer's Status

This is the response variable which is about the diagnosis status of the Alzheimer's Disease. It is clearly seen that the data is not balanced between the 2 categories which are "having Alzheimers" and "not having Alzheimer's" disease. Majority of the patients doesn't have Alzheimer's disease.

8.3.Bivariate Analysis

Cholesterol HDL vs Alzheimer's Status

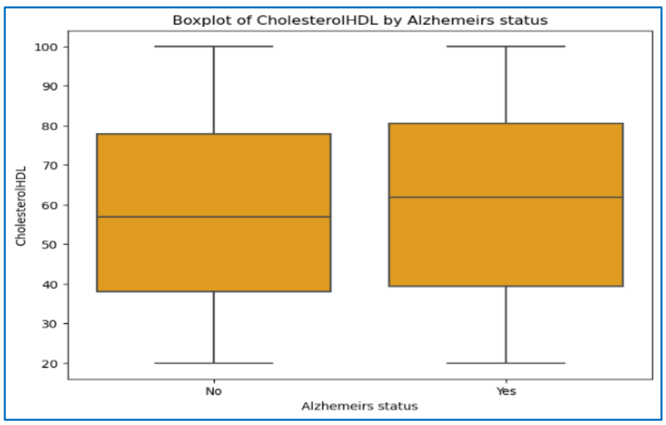


Figure 2-Boxplot for Cholesterol HDL vs Alzheimer's status

The boxplot illustrates the distribution of the Cholesterol/HDL ratio among patients grouped by Alzheimer's status. Both groups seems to have similar distributions, but median of the patients who are having Alzheimer's seems to have greater value for HDL cholesterol than the people with no Alzheimer's. Even though it is not a significant difference, it can indicate that there can be an association between Alzheimer's status and the HDL cholesterol level of a person. The absence of outliers further indicates that the data within each group is relatively consistent.

Sleep quality vs Alzheimer's Status

The boxplot compares the sleep quality scores between individuals with Alzheimer's and those without. Both groups exhibit very similar distributions in terms of sleep quality. However sleep quality is greater for non-Alzheimer's while Alzheimer's has low sleep quality value indicating that sleep quality may has some impact of Alzheimer's disease. This is also stated in the below article paper that sleeping quality of a person is associated with the Alzheimer's disease status of a person.

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9168575/

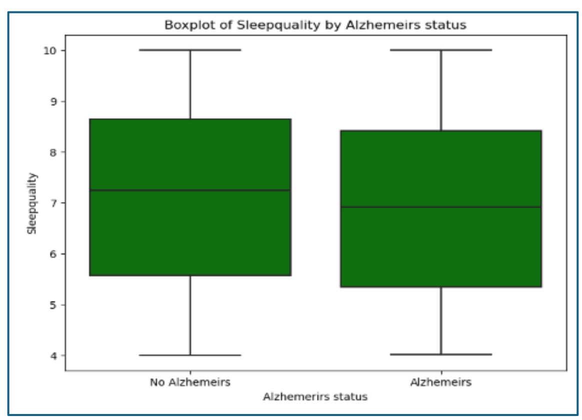


Figure 3-Boxplot of Sleep quality vs Alzheimer's Status

Memory complaints vs Alzheimer's status

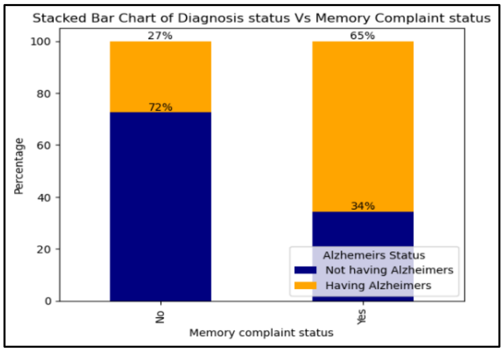
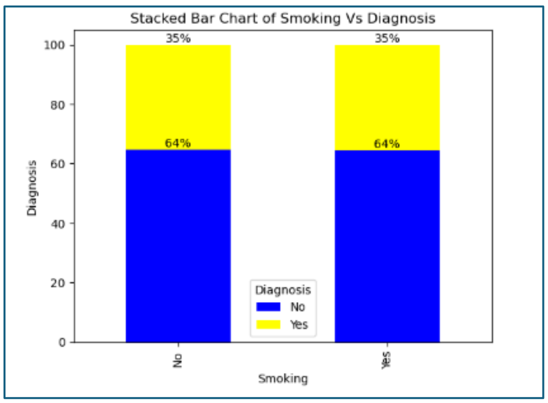


Figure 4-Stacked bar chart of Memory complaints vs Alzheimer's status

The stacked bar chart reveals a strong association between memory complaints and Alzheimer's disease. Individuals who report having memory issues are significantly more likely to be diagnosed with Alzheimer's compared to those without such complaints. This finding aligns with existing research, suggesting a potential link between memory problems and the development or progression of Alzheimer's.

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2740723/

Smoking vs Alzheimer's Status



The stacked bar chart suggests no significant association between smoking and diagnosis. Both smokers and non-smokers have similar rates of the condition. However, this finding contradicts existing research that strongly links smoking to an increased risk of various diseases, including Alzheimer's disease.
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4098701/

Figure 5-Stacked bar chart of Smoking vs Alzheimer's status

8.4. <u>Correlation Among Variables</u>

- GVIF values for all the variables were calculated and found out that all the values are less than 10. Therefore it was concluded that there was no multicollinearity among the variables.

8.5. <u>Factor Analysis for Mixed Data</u>

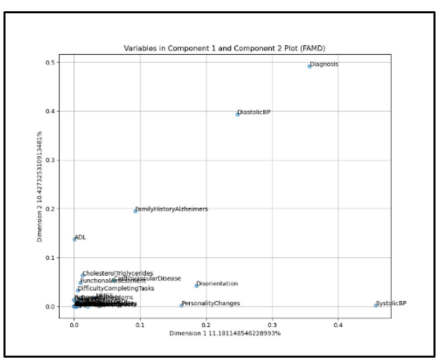


Figure 6-Loading plot of FAMD

With the main intention of visualizing all the variables in one frame FAMD was conducted.

It is seen from this score plot that the dimensions of the axis only explain 11.18% and 14.28%, which is comparatively low. Therefore, a better interpretation of this was not able. But to get an initial idea it can be seen that the variables such as DiagnosticBP, SystolicBP, Family History Alzheimer's have an association between the response variable. But it should be noted that these association can be wrong with the variability explained by the components are low.

Here two colours represent the two classes of the response variable. It is to be seen that the responses of two classes are separated in to two sides approximately. But if we look at the center of the scattering of observations it is to be seen that the both classes are scattered together and there's no clear separation between them. Therefore it is not clearly decidable whether to use linear or non linear classification models in the advanced analysis and therefore it is decided to try both types of models in the advanced analysis.

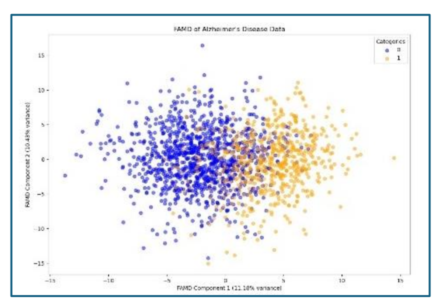


Figure 7-Score plot of FAMD

8.6 Cluster Analysis

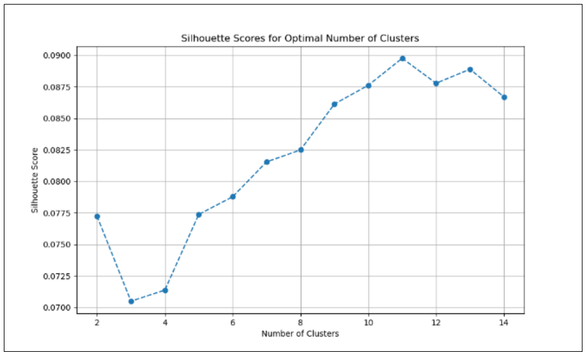


Figure 8-Silhouette scores plot of K-Means clustering

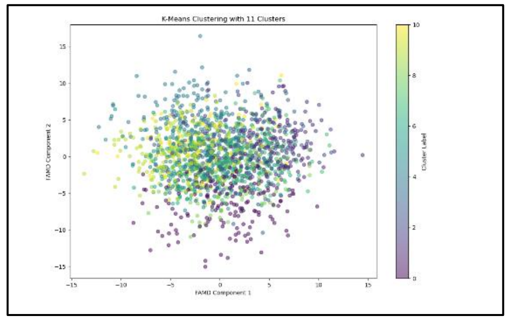


Figure 9-Score plot of clustering

The cluster analysis was implemented to identify distinct groups or clusters of patients based on their health conditions and general life characteristics using the results taken from the FAMD.

Checking whether clusters exist in the data, Maximum silhouette score is located at k=11, according to the silhouette score plot shown below. At that time, the silhouette width is measured, and it turns out to be 0.208. It can be inferred that there is no quality clustering in the dataset as the value is less than 0.5.

Important Findings of Descriptive Analysis

Based on the point biserial test, variables such as Sleep Quality, CholesterolHDL ratio, Functional Assessment, Behavioral Problems, Activities of Daily Living (ADL), Memory Complaints, and MMSE were identified as significantly associated with Alzheimer's disease. The Variance Inflation Factor (VIF) analysis confirmed that there are no multicollinearity issues within the dataset. However, cluster analysis revealed that the dataset lacks meaningful clustering, and no potential outliers were detected. It was also noted that the dataset is imbalanced, which necessitates steps to address this imbalance in future analyses.

Following a thorough descriptive analysis the goal of the analysis is to identify the best model based on evaluation metrics. Logistic regression is suggested as a benchmark model, with additional consideration given to Discriminant Analysis, K-Nearest Neighbors (KNN), Support Vector Machine, LASSO, Ridge, ElasticNet Regression, and ensemble methods such as Random Forest and XGBoost. Evaluation will focus on metrics such as the F1 score, accuracy, and precision.

9. Advanced Analysis

9.1. Logistic Regression

Logistic Linear Regression With the response variable being binary, Logistic linear regression was selected as the benchmark model.

Table 2-Evaluation table of Logistic regression

	Train Accuracy	Test Accuracy	Train F1		Overall	Test F1		Overall
			1	0		1	0	
Logistic Regression	0.9017	0.8860	0.8576	0.9249	0.9011	0.8339	0.9133	0.8852
Logistic Regression (After SMOTE)	0.8740	0.8698	0.8764	0.8714	0.8739	0.8261	0.8959	0.8712

As the Base model the performance is good in logistic regression. Since the dataset is imbalanced SMOTE technique was applied to balance the dataset and as above table shows the overall Test F1 score has been reduced by a small amount.

9.2. Discriminant Analysis

Discriminant Analysis is a statistical technique used to classify observations into predefined groups based on their characteristics. It is particularly useful for understanding the differences between groups and making predictions about which group new observations are likely to belong to.

9.2.1. Linear Discriminant Analysis

LDA aims to find a linear combination of features that best separates two or more classes of objects or events. It is often used when the data is linearly separable

Table 3-Evaluation table of Linear discriminant analysis

	Train Accuracy	Test Accuracy	Train F1		Overall	Test F1		Overall
			1	0		1	0	
LDA	0.8906	0.8744	0.8423	0.9163	0.8901	0.8200	0.9036	0.8740
LDA (After SMOTE)	0.8618	0.8488	0.8672	0.8561	0.8616	0.8024	0.8776	0.8510

9.2.2. Quadratic Discriminant Analysis

QDA also finds a combination of features that separate classes but allow for different covariance matrices for each class. This makes it suitable for data that is not linearly separable.

Table 4-Evaluation Table of Quadratic Discriminant analysis

	Train Accuracy	Test Accuracy	Train F1		Overall	Test F1		Overall
			1	0		1	0	
QDA	0.8889	0.8558	0.8521	0.9110	0.8902	0.8038	0.8860	0.8570
QDA (After SMOTE)	0.8515	0.7767	0.8584	0.8439	0.8515	0.7000	0.8222	0.7790

Further the lasso regression was applied because there are 35 predictor variables with the intension of applying the variable selection.

9.3. Lasso Regression

Table 5-Evaluation Table of Lasso Regression

	Train Accuracy	Test Accuracy	Train F1		Overall	Test F1		Overall
			1	0		1	0	
Lasso Regression	0.9034	0.8860	0.9261	0.8607	0.9030	0.9133	0.8339	0.8852
Lasso Regression (After SMOTE)	0.8929	0.8930	0.8945	0.8912	0.8929	0.8571	0.9145	0.8942

After applying the lasso regression (Regularization) the overall Test f1 score has not been changed. SMOTE technique has improved the performance without any overfitting.

9.4.SVM

Since it is identified that in 2-dimensional space, we can't exactly say that observations are linearly separable between the 2 classes, the SVM model also fitted as well.

Table 6-Evaluation Table of SVM

	Train Accuracy	Test Accuracy	Train F1 score		Overall	Test F1 score		Overall
			1	0		1	0	
SVM-without tuning	0.9354	0.8884	0.9049	0.9511	0.9348	0.8310	0.9167	0.8864
SVM-without tuning (SMOTE)	0.9509	0.8957	0.9512	0.9507	0.9509	0.8930	0.8982	0.8956
SVM-with tuning	0.8959	0.8907	0.8502	0.9202	0.8954	0.8396	0.9171	0.8897
SVM-with tuning (SMOTE)	0.8758	0.8	0.8752	0.8737	0.8744	0.8893	0.8947	0.8920

The test f1 scores between 2 classes are somewhat different without applying the SMOTE, after applying the SMOTE those 2 f1 scores are approximately similar and the overall f1 score is also higher in SMOTE. Also, without parameter tuning gives good f1 scores.

9.5.Random Forest

Table 7 - Evaluation Table of Random Forest

	Train Accuracy	Test Accuracy	Train F1		Overall	Test F1		Overall
			1	0		1	0	
Random Forest without parameter tune	1	0.9512	1	1	1	0.9293	0.9627	0.9509
Random Forest without parameter tune (After SMOTE)	1	0.9488	1	1	1	0.9262	0.9609	0.9486
Random Forest with parameter tune	0.9988	0.9512	0.9984	0.9991	0.9988	0.9298	0.9626	0.9510
Random Forest with parameter tune (After SMOTE)	1	0.9488	1	1	1	0.9262	0.9609	0.9486

Applying SMOTE method hasn't improved the performance of RF but has increased overfitting.

9.6.XG Boost

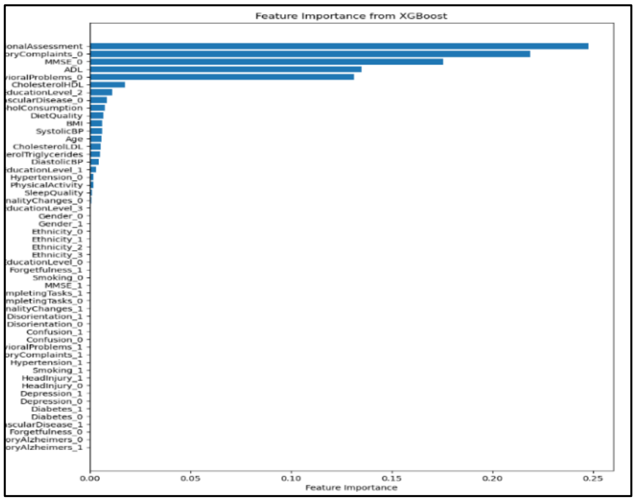
XGBoost is a powerful and widely used boosting method which is well known for its high predictive power and efficiency. Here it is visible that using SMOTE

Table 8 - Evaluation Table of XGBoost

	Train Accuracy	Test Accuracy	Train F1 score		Overall	Test F1 score		Overall
			1	0		1	0	
XGboost-without tuning	1	0.9488	1	1	1	0.9267	0.9607	0.9490
XGboost-without tuning (SMOTE)	1	0.9245	1	1	1	0.9219	0.9268	0.9245
XGboost-with tuning	0.9570	0.9535	0.9382	0.9670	0.9571	0.9333	0.9643	0.9533
XGboost-with tuning (SMOTE)	0.9914	0.9335	0.9915	0.9914	0.9914	0.9314	0.9354	0.9334

In xgboost applying SMOTE gives some overfitting results. After parameter tuning Xgboost has taken good accuracies and the F1 scores and also the between the 2 classes F1 scores are really similar.

Now, after comparing all the fitted models, it can be identified that without applying the SMOTE, the tuned Xg Boost model has approached the highest overall test F1 score and accuracy. Therefore, here we identify the important features the model obtained.



The most important features are, **"Functional Assessment", "Memory Complaints_0", "MMSE_0", "ADL" and "Behavioral Problems_0"**

Figure 10-Feature importance plot of Xg boost after parameter tuning

With the important variables,

Table 9-Evaluation Table of XGboost with important variables

Train Accuracy	Test Accuracy	Train F1 score		Overall	Test F1 score		Overall
		1	0		1	0	
0.9575	0.9512	0.9391	0.9674	0.9577	0.9302	0.9624	0.9511

Since, here model has used only 5 predictor variables out of 34 variables and accuracies and F1 scores are really closer to the corresponding values of previous best model. So, this is the best model.

Partial Dependency plots

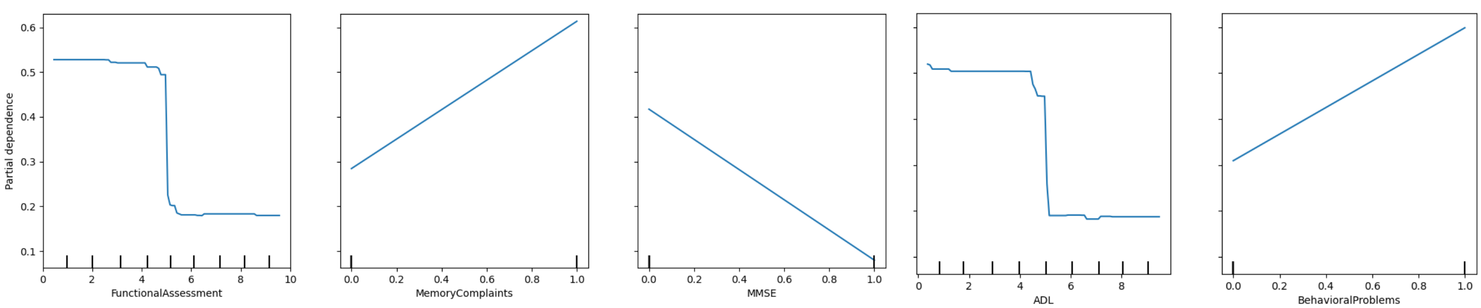


Figure 11-Partial Dependency Plots

As these partial dependency plots show, both memory complaints and Behavioral problems when the value takes 1 as they are present there is a higher probability of being an Alzheimer's patient. But in MMSE, when it takes the category of "25-30", the probability of being Alzheimer's patient is very low. Also, the gaps between those probabilities are considerable value, So, those variables are significant. Both Functional Assessment and ADL variables also have similar behavior where values greater than 5 indicates the lower probabilities of having Alzheimer's disease.

10. Issues encountered & Solutions proposed

The dataset that used for the study is an imbalanced one, therefore in some models that were fitted result overfitting and also the F1 scores of the 2 classes are quite different in some of the models, therefore we applied the smote as well in order to mitigate that issue. However, some fitted models were performed well without the SMOTE as well.

11. Discussion and conclusion

In the data preprocessing part, the variables "Patient ID" and "DoctorInCharge" have been removed as they are not important to the analysis and the objectives. Also, the variable "CholesterolTotal" was removed, because that variable made a redundancy among the variables. Then the variable "MMSE" which is a continuous variable converted to the categorical variable based on a reference that we found to seek more interpretation.

In the analysis, it is found that there is no multicollinearity and also no outliers as well. From the Descriptive analysis, it was found that "Sleep Quality", "CholesterolHDL", "Functional Assessment", "Behavioral Problems", "Activities of Daily Living (ADL)", "Memory Complaints", and "MMSE" variables have a significant association with the Response variable which is Alzheimer's disease status of a person. This also was confirmed by the advanced analysis as well because it was found that "Functional Assessment", "Behavioral Problems", "Activities of Daily Living (ADL)", "Memory Complaints", and "MMSE" are the important variables by using the feature importance plot.

As an interesting finding it was found that in our data set, there are no linear decision boundaries in 2-dimensional space after applying the FAMD. But since we can't directly ignore the fact that there is no linear boundaries in data by only observing the score plot of FAMD, both linear and non-linear models were carried out. So, the logistic regression was used as the benchmark model and along with that LDA, QDA, Lasso regression, SVM, Random Forest and XGboost models were utilized.

The evaluation method that we applied is the F1 score along with the accuracy, precision and recall because the F1 score provides a balanced evaluation of a model's performance especially when an imbalanced dataset is present. The balancing technique SMOTE applied but it was not given good accuracies and F1-scores for some models. So, finally, it was identified that XGboost model after hyper parameter tuning without applying SMOTE was the model which was performed better than the other model among before and after applying SMOTE.

Table 10-Comparison of the models

Model	Train Accuracy	Test Accuracy	Train F1 score	Test F1 score
Logistic Regression	0.9017	0.8860	0.9011	0.8852
Linear Discriminant	0.8906	0.8744	0.8901	0.8740
Quadratic Discriminant	0.8889	0.8558	0.8902	0.8570
Lasso	0.9034	0.8860	0.9030	0.8852
SVM	0.8959	0.8907	0.8954	0.8897
Random forest	0.9988	0.9512	0.9988	0.9510
XGboost	0.9570	0.9535	0.9571	0.9533

12. Appendix

Random Forest Without Parameter Tune : Imbalance

```
## Random Forest Without balancing
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

```
# Define which columns are numerical and which are categorical
numerical_features = x_train.select_dtypes(include=['float64', 'int64']).columns
categorical_features = x_train.select_dtypes(include=['category']).columns
```

```
# Preprocessing for numerical data (scaling)
numerical_transformer = StandardScaler()

# Preprocessing for categorical data (one-hot encoding)
categorical_transformer = OneHotEncoder(handle_unknown='ignore')

# Combine preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])
```

```
# Create a pipeline with preprocessing and the RandomForest classifier
rf_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(random_state=15643))
])
```

```
# Fit the model
rf_pipeline.fit(x_train, y_train)
```

```
Pipeline
preprocessor: ColumnTransformer
num                 cat
StandardScaler      OneHotEncoder
RandomForestClassifier
```

```
# Predict on the training data
y_pred = rf_pipeline.predict(x_train)
# Evaluate the model on training data
accuracy = accuracy_score(y_train, y_pred)
print(f"Training Accuracy: {accuracy:.2f}")
```

```
print("\nClassification Report:")
print(classification_report(y_train, y_pred,digits=4))
```

```
Classification Report:
              precision    recall  f1-score   support

           0     1.0000    1.0000    1.0000      1111
           1     1.0000    1.0000    1.0000       608

    accuracy                         1.0000      1719
   macro avg     1.0000    1.0000    1.0000      1719
weighted avg     1.0000    1.0000    1.0000      1719
```

```
# Predict on the training data
y_pred_test = rf_pipeline.predict(x_test)
# Evaluate the model on training data
accuracy = accuracy_score(y_test, y_pred_test)
print(f"Training Accuracy: {accuracy:.2f}")
```

```
Training Accuracy: 0.95
```

```
print("\nClassification Report:")
print(classification_report(y_test, y_pred_test,digits=4))
```

```
Classification Report:
              precision    recall  f1-score   support

           0     0.9509    0.9748    0.9627       278
           1     0.9517    0.9079    0.9293       152

    accuracy                         0.9512       430
   macro avg     0.9513    0.9414    0.9460       430
weighted avg     0.9512    0.9512    0.9509       430
```

Random Forest Without Parameter Tune : Balanced

```
#!pip install imbalanced-learn
```

```
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification
```

```
x_train = x_train.astype(float)
```

```
smote = SMOTE(random_state=15643)
X_train_resampled, y_train_resampled = smote.fit_resample(x_train, y_train)
```

```
from collections import Counter

print("Original dataset shape:", Counter(y_train))
print("Resampled dataset shape:", Counter(y_train_resampled))
```

```
Original dataset shape: Counter({0: 1111, 1: 608})
Resampled dataset shape: Counter({0: 1111, 1: 1111})
```

```
##extract numerical and categorical variables from x_train
##numerical
df2 =X_train_resampled[['Age', 'BMI', 'AlcoholConsumption','PhysicalActivity','DietQuality','SleepQuality','SystolicBP','Diastoli
             'CholesterolLDL','CholesterolHDL','CholesterolTriglycerides','FunctionalAssessment','ADL']]
###categorical
df3=X_train_resampled[['Gender','Ethnicity','EducationLevel','Smoking','FamilyHistoryAlzheimers','CardiovascularDisease','MMSE','
       'PersonalityChanges','BehavioralProblems',
       'Diabetes','Depression','HeadInjury','Hypertension','Confusion','Disorientation','DifficultyCompletingTasks','Forgetfulne
```

```
#####scale the features
# Import the StandardScaler
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled_s = scaler.fit_transform(X_train_resampled)
X_test_scaled_s = scaler.transform(x_test)
```

```
# Initialize the Random Forest Classifier
rf = RandomForestClassifier(random_state=15643)

# Train the model
rf.fit(X_train_scaled_s, y_train_resampled)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=15643)
```

```
####predict on train data
y_pred_tr = rf.predict(X_train_scaled_s)
y_pred = rf.predict(X_test_scaled_s)

accuracy_test = accuracy_score(y_test, y_pred)
print("Accuracy test:", accuracy_test)
accuracy_train = accuracy_score(y_train_resampled, y_pred_tr)
print("Accuracy train:", accuracy_train)

from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_pred,y_test)

print("Confusion Matrix:")
print(conf_matrix)
```

```
Accuracy test: 0.9488372093023256
Accuracy train: 1.0
Confusion Matrix:
[[270  14]
 [  8 138]]
```

Random Forest With Grid Search : Imbalanced

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
```

```
# Initialize the Random Forest model
rf = RandomForestClassifier(random_state=15643)

# Define the parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],        # Number of trees in the forest
    'max_depth': [10, 20, 30],              # Maximum depth of the tree
    'min_samples_split': [2, 5, 10],        # Minimum number of samples required to split an internal node
    'min_samples_leaf': [1, 2, 4],          # Minimum number of samples required to be at a leaf node
    'max_features': ['auto', 'sqrt'],       # Number of features to consider at each split
    'bootstrap': [True, False]              # Whether bootstrap samples are used when building trees
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2, scoring='accuracy')

# Fit the model
grid_search.fit(x_train, y_train)

# Get the best parameters
print("Best Parameters:", grid_search.best_params_)

# Use the best estimator to make predictions
best_rf = grid_search.best_estimator_
y_pred = best_rf.predict(x_test)
```

```
####F1 score test
y_pred = best_rf.predict(x_test)
y_pred_tr = best_rf.predict(x_train)
from sklearn.metrics import classification_report
report=classification_report(y_test,y_pred,digits=4)
print("test set")
print(report)
#f1 score train
report1=classification_report(y_train,y_pred_tr,digits=4)
print("train set")
print(report1)
```

test set

	precision	recall	f1-score	support
0	0.9541	0.9712	0.9626	278
1	0.9456	0.9145	0.9298	152
accuracy			0.9512	430
macro avg	0.9498	0.9428	0.9462	430
weighted avg	0.9511	0.9512	0.9510	430

train set

	precision	recall	f1-score	support
0	0.9991	0.9991	0.9991	1111
1	0.9984	0.9984	0.9984	608
accuracy			0.9988	1719
macro avg	0.9987	0.9987	0.9987	1719
weighted avg	0.9988	0.9988	0.9988	1719

Random Forest With Grid Search : Balanced

```
# Initialize the Random Forest model
rf = RandomForestClassifier(random_state=15643)

# Define the parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],       # Number of trees in the forest
    'max_depth': [10, 20, 30],             # Maximum depth of the tree
    'min_samples_split': [2, 5, 10],       # Minimum number of samples required to split an internal node
    'min_samples_leaf': [1, 2, 4],         # Minimum number of samples required to be at a leaf node
    'max_features': ['auto', 'sqrt'],      # Number of features to consider at each split
    'bootstrap': [True, False]             # Whether bootstrap samples are used when building trees
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2, scoring='accuracy')

# Fit the model
grid_search.fit(X_train_scaled_s,y_train_resampled)

# Get the best parameters
print("Best Parameters:", grid_search.best_params_)

# Use the best estimator to make predictions
best_rf = grid_search.best_estimator_
y_pred = best_rf.predict(X_test_scaled_s)
```

```
####F1 score test
y_pred = best_rf.predict(X_test_scaled_s)
y_pred_tr = best_rf.predict(X_train_scaled_s)
####F1 score test
from sklearn.metrics import classification_report
report=classification_report(y_test,y_pred,digits=4)
print("test set")
print(report)
#f1 score train
report1=classification_report(y_train_resampled,y_pred_tr,digits=4)
print("train set")
print(report1)
```

test set

	precision	recall	f1-score	support
0	0.9507	0.9712	0.9609	278
1	0.9452	0.9079	0.9262	152
accuracy			0.9488	430
macro avg	0.9480	0.9396	0.9435	430
weighted avg	0.9488	0.9488	0.9486	430

train set

	precision	recall	f1-score	support
0	1.0000	1.0000	1.0000	1111
1	1.0000	1.0000	1.0000	1111
accuracy			1.0000	2222
macro avg	1.0000	1.0000	1.0000	2222
weighted avg	1.0000	1.0000	1.0000	2222

Lasso Regression : Imbalanced

```
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LogisticRegression
```

```
lasso_logistic_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(penalty='l1', solver='saga', C=1.0, random_state=15643, max_iter=10000))
])
```

```
# Fit the model on the training data
lasso_logistic_pipeline.fit(x_train, y_train)

# Predict on the test data
y_pred = lasso_logistic_pipeline.predict(x_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("Classification Report:")
print(classification_report(y_test, y_pred,digits=4))
```

Classification Report:

	precision	recall	f1-score	support
0	0.8990	0.9281	0.9133	278
1	0.8601	0.8092	0.8339	152
accuracy			0.8860	430
macro avg	0.8795	0.8686	0.8736	430
weighted avg	0.8852	0.8860	0.8852	430

```
# Predict on the test data
y_pred_tr = lasso_logistic_pipeline.predict(x_train)

# Evaluate the model
accuracy = accuracy_score(y_train, y_pred_tr)
print(f"Accuracy: {accuracy:.2f}")
print("Confusion Matrix:")
print(confusion_matrix(y_train, y_pred_tr))
print("Classification Report:")
print(classification_report(y_train, y_pred_tr,digits=4))
```

```
Accuracy: 0.90
Confusion Matrix:
[[1040   71]
 [  95  513]]
Classification Report:
```

	precision	recall	f1-score	support
0	0.9163	0.9361	0.9261	1111
1	0.8784	0.8438	0.8607	608
accuracy			0.9034	1719
macro avg	0.8974	0.8899	0.8934	1719
weighted avg	0.9029	0.9034	0.9030	1719

Lasso Regression : Balanced

```
## Lasso after smote
```

```
# Initialize the Random Forest Classifier
lasso = LogisticRegression(penalty='l1', solver='saga', C=1.0, random_state=15643, max_iter=10000)

# Train the model
lasso.fit(X_train_scaled_s, y_train_resampled)
```

```
LogisticRegression
LogisticRegression(max_iter=10000, penalty='l1', random_state=15643,
                   solver='saga')
```

```
####predict on train data
y_pred_tr = lasso.predict(X_train_scaled_s)
y_pred = lasso.predict(X_test_scaled_s)

accuracy_test = accuracy_score(y_test, y_pred)
print("Accuracy test:", accuracy_test)
accuracy_train = accuracy_score(y_train_resampled, y_pred_tr)
print("Accuracy train:", accuracy_train)

from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_pred,y_test)

print("Confusion Matrix:")
print(conf_matrix)
```

```
####F1 score test
from sklearn.metrics import classification_report
report=classification_report(y_test,y_pred,digits=4)
print("test set")
print(report)
#f1 score train
report1=classification_report(y_train_resampled,y_pred_tr,digits=4)
print("train set")
print(report1)
```

XGBoost

With default parameters

```
# XgBoost with the default parameters
categorical_col = x_train.select_dtypes(include='category').columns.tolist()
numeric_col = x_train.select_dtypes(include=['Int64', 'float64','object']).columns.tolist()

# Need to scale the numerical variables and encode the categorical variables before applying xgboost
transformer = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_col),  # Apply StandardScaler to 'numerical' column
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_col), # Apply OneHotEncoder to 'categorical' column
    ],
    remainder='passthrough'  # Include remaining columns as-is
)

xgb_model = Pipeline(steps=[
    ('preprocessor', transformer),
    ('xgb', XGBClassifier())])

xgb_model.fit(x_train,y_train)
```

```
y_pred = xgb_model.predict(x_train)
accuracy = accuracy_score(y_pred,y_train)
print("Accuracy train: {:.2f}%".format(accuracy * 100))

matrix = classification_report(y_pred,y_train,digits=4)
print("confusion matrix : \n",matrix)
```

With parameter tuning

```
y_pred_test = xgb_model.predict(x_test)
accuracy_test = accuracy_score(y_pred_test,y_test)
print("Accuracy test: {:.2f}%".format(accuracy_test * 100))

matrix = classification_report(y_pred_test,y_test,digits=4)
print("confusion matrix : \n",matrix)
```

SMOTE applying

```
categorical_col = x_train.select_dtypes(include='category').columns.tolist()
numeric_col = x_train.select_dtypes(include=['Int64', 'float64','object']).columns.tolist()

# Need to scale the numerical variables and encode the categorical variables before applying xgboost
transformer = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_col),  # Apply StandardScaler to 'numerical' column
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_col), # Apply OneHotEncoder to 'categorical' column
    ],
    remainder='passthrough'  # Include remaining columns as-is
)

xgb_model = Pipeline(steps=[
    ('preprocessor', transformer),
    ('xgb', XGBClassifier(random_state=100))])

xgb_param_grid = {
    'xgb__n_estimators': [50, 100, 200,250,300,400,450,500],
    'xgb__max_depth': [3,4, 5, 6, 7],
    'xgb__learning_rate': [0.01, 0.001,0.0001]
}

xgb_grid_search = GridSearchCV(xgb_model, xgb_param_grid,cv=5, n_jobs=-1, verbose=2)
xgb_grid_search.fit(x_train,y_train)
```

Best model fitting after getting the feature importance plot

```
imp_var = ["FunctionalAssessment","MemoryComplaints","MMSE","ADL","BehavioralProblems"]
X = x_train[imp_var]
```

```
# Feature Importance plot
xgb_classfier = best_model_xgb.named_steps["xgb"]
# Extract the feature importance values
feature_importance = xgb_classfier.feature_importances_

best_model_xgb.named_steps.keys()
#Feature names according to the encoding
transformer = best_model_xgb.named_steps['preprocessor']
numerical = transformer.transformers_[0][2]
categorical = transformer.transformers_[1][1]
cat_feature_names = categorical.get_feature_names_out()
all_feature_names = np.concatenate([numerical, cat_feature_names])
sort = np.argsort(feature_importance)[::-1]

plt.figure(figsize=(10, 12))
plt.barh(all_feature_names[sort], feature_importance[sort])
plt.xlabel('Feature Importance')
plt.title('Feature Importance from XGBoost')
plt.gca().invert_yaxis()  # Highest importance at the top
plt.savefig('feature importance.png')
plt.show()
```

Partial dependency plots

```
# Partial dependency plot
from sklearn import inspection
from sklearn.inspection import PartialDependenceDisplay

features_to_plot =[0,1,2,3,4]   # Example: Plotting partial dependence for the feature at index 0,1,2,3,4
xgb_classfier_new = best_model_xgb_new.named_steps["xgb"]

# Plot partial dependence
fig, ax = plt.subplots(figsize=(15, 10))
PartialDependenceDisplay.from_estimator(best_model_xgb_new, X, features_to_plot, ax=ax)
plt.title("Partial Dependence Plots")
plt.savefig('partialdependency.png')
plt.show()
```

SVM default parameters

```
# SVM
from sklearn.svm import SVC

categorical_col = x_train.select_dtypes(include='category').columns.tolist()
numeric_col = x_train.select_dtypes(include=['Int64', 'float64','object']).columns.tolist()

# Need to scale the numerical variables and encode the categorical variables before applying xgboost
transformer = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_col),  # Apply StandardScaler to 'numerical' column
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_col), # Apply OneHotEncoder to 'categorical' column
    ],
    remainder='passthrough'  # Include remaining columns as-is
)

svm_model = Pipeline(steps=[
    ('preprocessor', transformer),
    ("svm",SVC())])

svm_model.fit(x_train,y_train)
```

SVM Parameter tuning

```
from sklearn.svm import SVC

categorical_col = x_train.select_dtypes(include='category').columns.tolist()
numeric_col = x_train.select_dtypes(include=['Int64', 'float64','object']).columns.tolist()

# Need to scale the numerical variables and encode the categorical variables before applying xgboost
transformer = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_col),  # Apply StandardScaler to 'numerical' column
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_col), # Apply OneHotEncoder to 'categorical' column
    ],
    remainder='passthrough'
)

svm_model = Pipeline(steps=[
    ('preprocessor', transformer),
    ("svc",SVC())])


param_grid = {'svc__C': [0.1, 1, 10, 100,1000],
              'svc__gamma': [1, 0.1,10,100,1000],
              'svc__kernel': ['rbf','linear','poly']}
svm_gridsearch = GridSearchCV(svm_model,param_grid,cv=5, n_jobs=-1)
svm_gridsearch.fit(x_train,y_train)
```

```
# For unbalanced data sets
from imblearn.over_sampling import SMOTE
smote= SMOTE()
x_train_re, y_train_re =smote.fit_resample(x_train, y_train)
x_test_re, y_test_re =smote.fit_resample(x_test, y_test)
```

Logistic Regression

```
#####scale the features
# Import the StandardScaler
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(x_train)
X_test_scaled = scaler.transform(x_test)
```

```
X_train_scaled
```

```
### Logistic regression(without class inbalance)
# import the class
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Create an instance of LogisticRegression
model = LogisticRegression(max_iter=200)  # You can adjust max_iter as needed

# Fit the model to the training data
model.fit(X_train_scaled,y_train)

####predict on train data
y_pred_tr = model.predict(X_train_scaled)
y_pred = model.predict(X_test_scaled)

accuracy_test = accuracy_score(y_test, y_pred)
print("Accuracy test:", accuracy_test)
accuracy_train = accuracy_score(y_train, y_pred_tr)
print("Accuracy train:", accuracy_train)

from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_pred,y_test)

print("Confusion Matrix:")
print(conf_matrix)
```

```
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
```

```
####F1 score test
report=classification_report(y_test,y_pred,digits=4)
print("test set")
print(report)
#f1 score train
report1=classification_report(y_train,y_pred_tr,digits=4)
print("train set")
print(report1)
```

```
test set
              precision    recall  f1-score   support

           0     0.8990    0.9281    0.9133       278
           1     0.8601    0.8092    0.8339       152

    accuracy                         0.8860       430
   macro avg     0.8795    0.8686    0.8736       430
weighted avg     0.8852    0.8860    0.8852       430

train set
              precision    recall  f1-score   support

           0     0.9132    0.9370    0.9249      1111
           1     0.8791    0.8372    0.8576       608

    accuracy                         0.9017      1719
   macro avg     0.8961    0.8871    0.8913      1719
weighted avg     0.9011    0.9017    0.9011      1719
```

Linear Discriminant Analysis

```
################discriminant analysis
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

```
# Initialize the LDA model
lda = LinearDiscriminantAnalysis()

# Fit the model to the training data
lda.fit(X_train_scaled, y_train)
```

```
▸ LinearDiscriminantAnalysis
LinearDiscriminantAnalysis()
```

```
# Predict the classes for the test set
y_pred = lda.predict(X_test_scaled)
y_pred_tr = lda.predict(X_train_scaled)


accuracy_test = accuracy_score(y_test, y_pred)
print("Accuracy test:", accuracy_test)
accuracy_train = accuracy_score(y_train, y_pred_tr)
print("Accuracy train:", accuracy_train)

from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_pred,y_test)

print("Confusion Matrix:")
print(conf_matrix)
```

```
Accuracy test: 0.8744186046511628
Accuracy train: 0.8906340895869692
Confusion Matrix:
[[253  29]
 [ 25 123]]
```

```
####F1 score test
report=classification_report(y_test,y_pred,digits=4)
print("For test set",report)
#f1 score train
report1=classification_report(y_train,y_pred_tr,digits=4)
print("For train set",report1)
```

```
For test set               precision    recall  f1-score   support

           0     0.8972    0.9101    0.9036       278
           1     0.8311    0.8092    0.8200       152

    accuracy                         0.8744       430
   macro avg     0.8641    0.8596    0.8618       430
weighted avg     0.8738    0.8744    0.8740       430

For train set               precision    recall  f1-score   support

           0     0.9066    0.9262    0.9163      1111
           1     0.8596    0.8257    0.8423       608

    accuracy                         0.8906      1719
   macro avg     0.8831    0.8759    0.8793      1719
weighted avg     0.8900    0.8906    0.8901      1719
```

Quadratic Discriminant Analysis

```
######Quadratic discriminant analysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
```

```
# Initialize the QDA model
qda = QuadraticDiscriminantAnalysis()

# Fit the model to the training data
qda.fit(X_train_scaled, y_train)
```

```
▸ QuadraticDiscriminantAnalysis
QuadraticDiscriminantAnalysis()
```

```
# Predict the classes for the test set
y_pred = qda.predict(X_test_scaled)
y_pred_tr = qda.predict(X_train_scaled)
```

```
accuracy_test = accuracy_score(y_test, y_pred)
print("Accuracy test:", accuracy_test)
accuracy_train = accuracy_score(y_train, y_pred_tr)
print("Accuracy train:", accuracy_train)

from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_pred,y_test)

print("Confusion Matrix:")
print(conf_matrix)
```

```
####F1 score test
report=classification_report(y_test,y_pred,digits=4)
print("test set")
print(report)
#f1 score train
report1=classification_report(y_train,y_pred_tr,digits=4)
print("train set")
print(report1)
```