EARLY DETECTION OF ALZHEIMER'S DISEASE: FROM LOGISTIC REGRESSION TO XGBOOST

ABSTRACT

This study aims to fit, evaluate and choose from logistic regression and eXtreme Gradient Boosting (XGB) model to make an early diagnosis for Alzheimer's disease. While logistic regression yields about 78% accuracy and maintains a high precision of over 90%, it suffers from a relatively high false negative rate and relatively low accuracy. In contrast, XGBoost delivers superior performance, achieving nearly 96.5% accuracy, over 95% precision, and greatly reduced false positive and false negative rates. Its AUC of approximately 0.965 indicates stronger discriminative power. Feature importance analysis shows functional and cognitive assessments (e.g., ADL, MMSE) as key predictors. Although logistic regression's simplicity is appealing, the advanced gradient-boosted model better addresses the need to minimize missed early cases. Kaggle leaderboard scores (public: 0.95195, private: 0.91025) confirm its robustness and reliability. Overall, we found XGBoost is better for accurate and clinically meaningful early Alzheimer's disease classification.

Team Members with ID:

Haocheng Sun:	1008550097
Chi Zhang:	1003093509
Xile Chen:	1007850522

Kaggle Team Name: Vegetable Dogs

Final Ranking on Kaggle: Public: 53, Private: 108

Prediction Score: Public: 95.195%, Private: 91.025%

Dataset: Alzheimer's Disease Dataset (Rabie El Kharoua, 2023)

1 Introduction

Alzheimer's disease is a progressive disease that affects the brain and is the most common type of dementia, affecting between 60-70% of patients (Rosselli et al., 2022). It is characterized by the gradual loss of cognitive, behavioral, and functional skills due to the deposition of beta-amyloid plaques and tau proteins in the brain.

Although the mechanism of AD development is not fully known, the contribution of genetic background, environmental influences, and lifestyle factors has been suggested. The Lancet Commission (Livingston et al., 2020) identified 12 modifiable risk factors, of which hypertension, diabetes, physical inactivity, and smoking alone contribute to 40% of the global dementia burden. In addition, structural brain changes and cognitive markers, such as loss of memory and spatial judgment, have been recognized as some of the earliest signs of the disease.

Alzheimer's disease is prevalent, and it is estimated that more than 55 million people are affected by the disease globally. This number is expected to triple by 2050 because of the increasing number of older people (Rosselli et al., 2022). Although AD is becoming more common, it is still not detected very often across the globe, especially in developing countries where undetected cases of AD are greater than 90% (Lazarova et al., 2023). However, most diagnoses are made at moderate to severe stages, leading to limited treatment effectiveness (Li et al., 2024), and the impact of available treatments will be significantly reduced.

Due to its high incidence, long preclinical period, and the absence of effective treatments, early diagnosis is crucial. It allows intervention measures to reduce disease progression and enhance patient outcomes. This involves developing easy, efficient, and cost-effective screening tools that do not need the presence of specialist medical personnel and can be done in the community. These should have high accuracy and a low rate of false negative findings so that the right people are referred to specialized healthcare facilities for evaluation.

The aim of this project is to develop a statistical and machine-learning model for the early classification of Alzheimer's disease. The data we will use in this model will be demographical, behavioral, and cognitive. It will classify the individuals as having a high or low risk of developing the disease. In order to identify the most effective factors for Alzheimer's disease, we will use logistic regression (Lazarova et al., 2023) and XGBoost (Li et al., 2024). At the same time, we will try to make the model efficient and easy to apply. The ultimate goal is to design a tool for community screening of Alzheimer's and related disorders that can link the gap of underdiagnosis and early intervention to enhance care and management of the disease.

2 Data

2.1 Data Source

We use the *Alzheimer's Disease dataset* (Kharoua, 2023), which contains extensive health information for 2,149 patients, each uniquely identified with IDs ranging from 1 to 2149. The dataset includes demographic details, lifestyle factors, medical history, clinical measurements, cognitive and functional assessments, symptoms, and a diagnosis of Alzheimer's Disease.

2.2 Data Preprocessing

2.2.1 Exploratory Data Analysis (EDA)

1. **Missing Values:** There's no data with missing values.

2. **Categorical Variables:**

   There are 19 Number categorical variables. It includes: Gender, Ethnicity, EducationLevel, Smoking, FamilyHistoryAlzheimers, CardiovascularDisease, Diabetes, Depression, HeadInjury, Hypertension, MemoryComplaints, BehavioralProblems, Confusion, Disorientation, PersonalityChanges, DifficultyCompletingTasks, Forgetfulness, Diagnosis, DoctorInCharge. We drop DoctorInCharge as it is same for all data. Our target variable is *Diagonosis*, which will be 0 if our model predicts a low probability of Alzheimer's disease, and be 1 if our model predicts a high probability of Alzheimer's disease.

   For encoding of these categorical variables, all of them have been labeled properly into different levels in natural numbers. There're 17 of them are binary which don't need to encode. Two of them are multiple, Ethnicity and Education Level. The summary of the categorical variables are shown in Table 1.

   Table 1: Summary the data of the categorical variables

Category	**Value (Count)**
Gender	1 (765), 0 (739)
Ethnicity	0 (890), 1 (309), 2 (154), 3 (151)
EducationLevel	1 (588), 2 (454), 0 (311), 3 (151)
Smoking	0 (1077), 1 (427)
FamilyHistoryAlzheimers	0 (1136), 1 (368)
CardiovascularDisease	0 (1302), 1 (202)
Diabetes	0 (1264), 1 (240)
Depression	0 (1191), 1 (313)
HeadInjury	0 (1361), 1 (143)
Hypertension	0 (1276), 1 (228)
MemoryComplaints	0 (1195), 1 (309)
BehavioralProblems	0 (1276), 1 (228)
Confusion	0 (1199), 1 (305)
Disorientation	0 (1269), 1 (235)
PersonalityChanges	0 (1268), 1 (236)
DifficultyCompletingTasks	0 (1260), 1 (244)
Forgetfulness	0 (1053), 1 (451)
Diagnosis	0 (972), 1 (532)
DoctorInCharge	XXXConfid (1504)

3. **Numerical Variables:** There are 15 numerical variables. It includes: ADL, Age, AlcoholConsumption, BMI, CholesterolHDL, CholesterolLDL, CholesterolTotal, CholesterolTriglycerides, DiastolicBP, DietQuality, FunctionalAssessment, MMSE, PhysicalActivity, SleepQuality, SystolicBP. The summary of the numerical variables are shown in Table 2 and Table 3. After checking the histograms and boxplots(Figure 2.2.1), there are no outliers need to deal with and the distributions are close to uniform and not skewed.

Table 2: Summary Statistics of Categorical Variables (ADL to CholesterolTriglycerides)

Statistic	ADL	Age	AlcCons	BMI	CholHDL	CholLDL	CholTot	CholTrig
Count	1504.0	1504.0	1504.0	1504.0	1504.0	1504.0	1504.0	1504.0
Mean	5.0	75.0	10.0	28.0	60.0	125.0	225.0	227.0
Std Dev	3.0	9.0	6.0	7.0	23.0	43.0	42.0	102.0
Min	0.0	60.0	0.0	15.0	20.0	50.0	150.0	50.0
25%	2.0	67.0	5.0	21.0	39.0	88.0	190.0	136.0
50%	5.0	75.0	10.0	28.0	60.0	125.0	224.0	230.0
75%	8.0	83.0	15.0	34.0	79.0	162.0	262.0	313.0
Max	10.0	90.0	20.0	40.0	100.0	200.0	300.0	400.0

Table 3: Summary Statistics of Categorical Variables (DiastolicBP to SystolicBP)

Statistic	DiaBP	DietQ	FuncAssess	MMSE	PhysAct	SleepQ	SysBP
Count	1504.0	1504.0	1504.0	1504.0	1504.0	1504.0	1504.0
Mean	90.0	5.0	5.0	15.0	5.0	7.0	135.0
Std Dev	18.0	3.0	3.0	9.0	3.0	2.0	26.0
Min	60.0	0.0	0.0	0.0	0.0	4.0	90.0
25%	74.0	2.0	3.0	7.0	3.0	5.0	112.0
50%	90.0	5.0	5.0	14.0	5.0	7.0	135.0
75%	105.0	8.0	8.0	22.0	7.0	9.0	156.0
Max	119.0	10.0	10.0	30.0	10.0	10.0	179.0

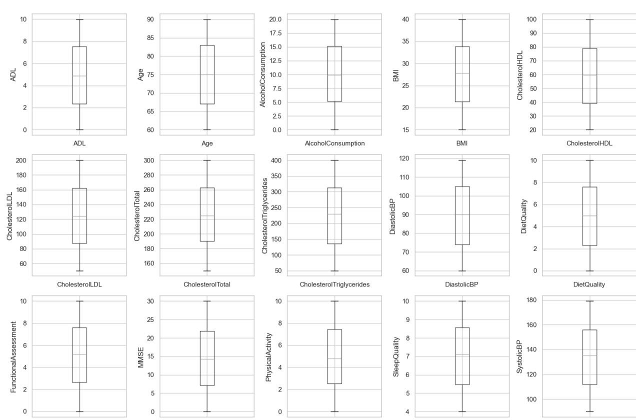


Figure 1: Boxplots of Numerical Variables

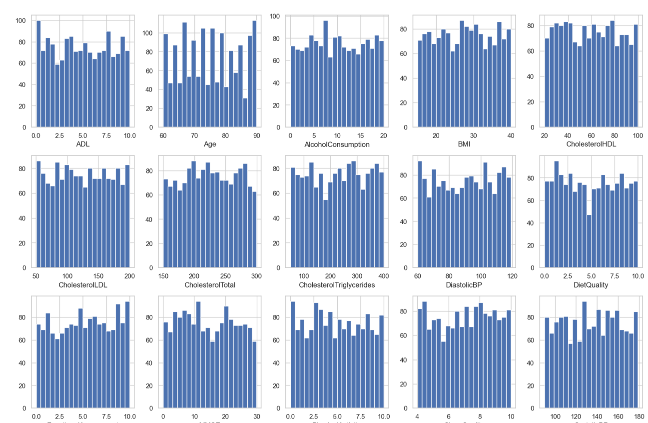


Figure 2: Higrams of Numerical Variables

2.2.2 Data Scaling

In Logistic regression, variables that are measured at different scales do not contribute equally to the model fitting and model learned function which might end up creating bias. We use MinMaxScaler to transform our data. All features will be transformed into the range [0,1] meaning that the minimum and maximum value of a feature/variable is going to be 0 and 1, respectively. The reason we choose MinMaxScalar is that the features in dataset have different ranges, no outliers(which prefers RobustScaler), not normal(which prefers StandardScaler), not sparse(which prefers Normalizer). The formula of MinMaxScalar is

$$x_{\text{scaled}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

3 Model

In this section, we train and evaluate model from two distinct ML models for the classification of Alzheimer's disease: a conventional Logistic Regression (LR) model and an eXtreme Gradient Boosting (XGB) model. Logistic Regression serves as a strong baseline on simplicity, interpretability, and widespread use in medical classification tasks. On the other hand, XGB is a gradient boosting model known for its robustness and capacity to handle non-linear relationships, complex feature interactions effectively.

3.1 Logistic Regression

3.1.1 Set-up and Training

For encoding, we encode the multiple categorical variables *Ethnicity, EducationLevel* by one hot encoding, which will be in [0,1] satisfying the scaling. The remaining binary categorical variables do not need to be encoded under MinMaxScaler. We use a 5-fold cross-validation score in accuracy as the selection criterion.

We first compare the logistic regression model penalty by L1 in different λ-values using grid searching by our criterion. Then, we compare the logistic regression model penalty by L2 in different λ-values by our criterion. Finally, we chose the best one between these two as our final model to fit. Our final model is the logistic regression penalty by L1 with $\lambda = 1/0.14$. Under L1 penalty, the feature selection is achieved at the same time.

3.1.2 Model justification

Our model has accuracy: 0.7832, Precision: 0.9064, False Positive Rate: 0.2345, False Negative Rate: 0.2106 with default threshold 0.5. The ROC_AUC Score is 0.8249.

The logistic regression does not work well. The reasons for this include: (1) Our features may have a non-linear, complex relation with our target; (2) Logistic Regression requires average or no multicollinearity between independent variables; however, in our features, for example, MemoryComplaints, DifficultyCompletingTasks, Forgetfulness are highly related.

Hence, a more complex tree-based model is needed for better performance in predicting AD.

3.2 eXtreme Gradient Boosting (XGB)

3.2.1 Set-up and Training

Considering the ability of XGB to handle the complexity of the data, we employ all 32 variables from our dataset. Since the scale of our hyperparameter tuning process is large, and the XGB model is highly sensitive to input parameters, we did not use k-fold cross-validation for hyperparameter tuning, opting instead to simply split the data with a portion as a validation set. After several trials, we discovered that a 1/4 train-validation data split is relatively optimal.

For hyperparameter tuning, we implement a grid search with a search space that includes variations in the number of estimators (n_estimators from 1 to 395 in increments of 5), tree depth (max_depth from 1 to 15 in increments of 2), learning rates (0.1 and 0.05), and both the subsampling ratio (subsample) and the column subsampling ratio (colsample_bytree) ranging from 0.7 to 0.9. There are a total of 32,000 parameters to search. The metric used to evaluate models during tuning is the accuracy of the model on the 1/4 validation set. To efficiently compute this exhaustive search, we leverage joblib's parallelization capabilities. Joblib is a Python library that enables easy parallelization of computing tasks, allowing parallel code to run faster by taking advantage of multiple CPU cores. The estimated time for an Intel Core i7-11370H processor and 16GB DDR5 RAM to complete this job is approximately 45 to 60 minutes. (Varoquaux et al., 2023)

3.2.2 Model justification

After hyperparameter tuning, the final chosen parameters—(36, 3, 0.1, 0.9, 0.85) for (n_estimators, max_depth, learning_rate, subsample, colsample_bytree), it reflect a configuration that maximizes accuracy on the validation set, while maintaining computational feasibility that it uses 50 minutes of parallelized training. The resulting model achieved a test accuracy of approximately 96.5% in our validation set, with precision (95.2%), recall (95.9%), and specificity (97.0%), it also acheived high F1-score (95.5%) and AUC (0.965). These metrics indicate not only that the model correctly classifies a substantial proportion of both positive and negative cases but also that it balances the trade-offs between precision and recall effectively.

4 Results

4.1 Choosing and Evaluating Two Models by Metrics

We compare two model based on five metric: accuracy, precision, FPR, FNR and ROC-AUC score.

Table 4: Logistic Regression Model Performance Metrics

Metric	**Value**
Accuracy	0.7832
Precision	0.9064
False Positive Rate (FPR)	0.2345
False Negative Rate (FNR)	0.2106
ROC_AUC Score	0.8249

Table 5: XGB Model Performance Metrics

Metric	**Value**
Accuracy	0.9654
Precision	0.9521
False Positive Rate (FPR)	0.0303
False Negative Rate (FNR)	0.0414
ROC_AUC Score	0.9652

The logistic regression model, although straightforward and interpretable, but only achieves an accuracy of about 78.3%, a precision of approximately 90.6%, and a false negative rate (FNR) of 21.1%. Its ROC_AUC score is around 0.825, suggesting a reasonably strong ability to distinguish between positive and negative classes. However, the relatively high false negative rate and the somewhat modest accuracy suggests that when deciding the decision boundary to make sure the precision to be higher than 90% (since it is a medical diagonosis, we should ensure most positive cases are captured), we have to make it to have a moderate accuracy.

The more advanced XGB model shows significant improved metrics. It acheives an accuracy near 96.5%, a precision over 95.2%, and crucially, also an much lower FPR and FNR, both less than 5%. This reduction in both FPR and FNR means that the model not only misclassify negative cases as positive less, but also misses very few positive cases. The AUC of about 0.965 underscores its better discriminative ability. This higher AUC means that the model ranks positive instances well above negative ones with much better consistency.

Taken together, these results strongly suggest that the XGB model outperforms logistic regression on both global (precision and accuracy) and class-specific metrics. Its high accuracy, balanced precision, and improved AUC suggest a more reliable classification performance. Although logistic regression model has better simplicity and interpretability, the complexity and flexibility of the gradient-boosted model are more suitable for our task. Thus, choosing the XGBoost model would likely yield more accurate and clinically useful results.

So the final model to be chosen is XGB model with 1/4 split of validation and train data, with parameter (36, 3, 0.1, 0.9, 0.85) for (n_estimators, max_depth, learning_rate, subsample, colsample_bytree), using 'XGBClassifier' in package 'xgboost' in python. (Chen and Guestrin, 2016) The confusion matrix is as follows:

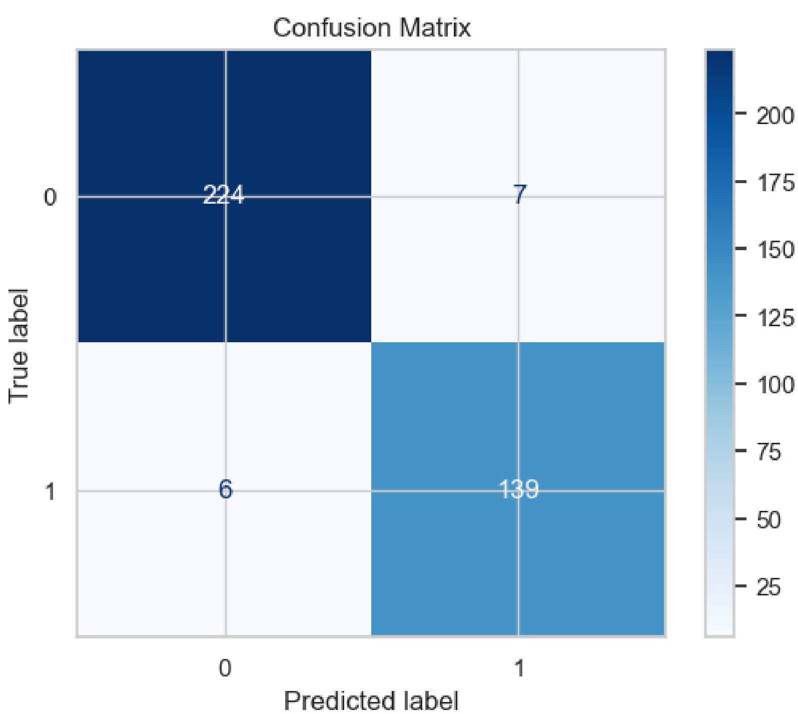


Figure 3: Confusion Matrix of the Model Predictions

4.2 Feature Importance

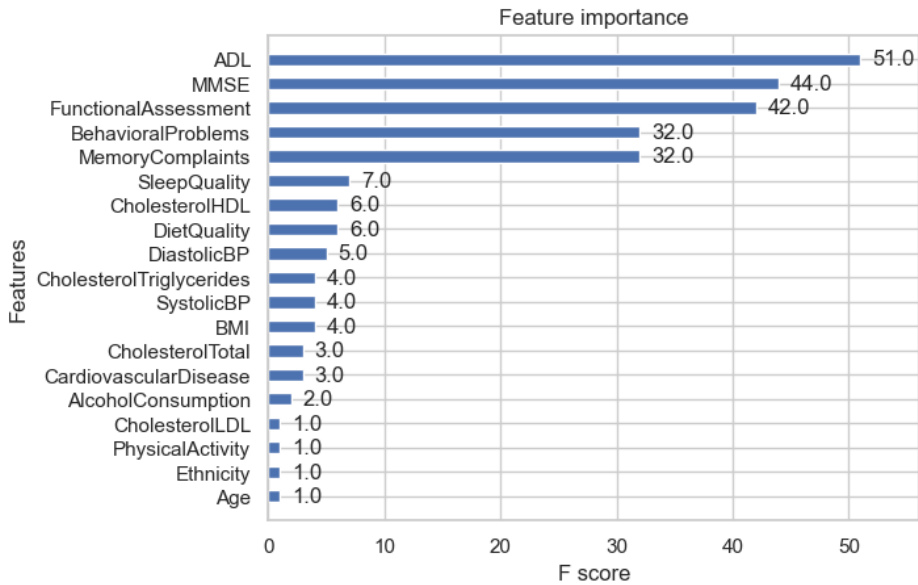


Figure 4: XGBoost feature importance (based on the F-score)

The feature importance plot presented in Figure 4 provides information on which variables most strongly influence the model's classification decisions. The Activities of Daily Living (ADL) score, the Mini-Mental State Examination (MMSE), and FunctionalAssessment emerge as the top three factors, underscoring the significance of functional and cognitive assessments in early Alzheimer's disease detection (Albert et al., 2011). Other high-ranking features, such as BehavioralProblems and MemoryComplaints, align with clinical intuition: they reflect cognitive impairment, behavioral disturbances, and the subjective perception of memory decline—key indicators associated with the onset of dementia. Furthermore, the importance attributed to variables like SleepQuality and DietQuality suggests that lifestyle and overall health patterns also play a relatively important role in the disease's progression or manifestation. In contrast, features like Ethnicity and Age, though clinically relevant, hold relatively lower importance in our model, possibly due to their indirect or weaker predictive influence on early AD detection. (Chen and Guestrin, 2016)

4.3 Kaggle Accuracy

For our Kaggle submission, predicted using the optimized XGBoost model, achieved a notable performance on the competition leaderboard. The public score based on a subset of the test data, was approximately 0.95195, reflecting the model's ability to generalize well beyond the training environment. However, there is a little gap between the public performance and private one of 0.91025. This difference exist probably because the slight distinct distrubution of public and private data. Nonetheless, the final result still tells a strong accuracy, demonstrating the effectiveness of our exhaust and careful hyperparameter tuning in delivering reliable and competitive model.

5 Discussion

5.1 Balancing Predictive Performance and Interpretability

In early Alzheimer's disease diagnosis, prioritizing the reduction of false negatives is crucial, as missing early case risks would delay beneficial interventions (Dukart et al., 2013). This is the reason when choosing the decision boundary of logistic regression, we set a requirement the precision is larger than 90%. Although it provides interpretability through clear coefficients, its performance is not satisfactory. By contrast, XGBoost's ability to capture non-linear relationships leads to more accurate and robust classifications. In a clinical setting, optimizing diagnostic performance should result in early detection of at-risk individuals. Also, enhancing model explainability using built-in functions could also provide interpretability, Figure 4 allowing clinicians to see which feature is significant in early diagnosis.

5.2 Data Limitations and Future Directions

The current dataset is just large enough to train a single model. However, improving the model's predictive power could be achieved by constructing a meta-model, a concept that often requires more extensive data (Wolpert, 1992). The proposed meta-model approach is as follows:

During several hyperparameter tuning trials, we observed that while multiple parameter configurations achieved high accuracy, the specific data points they misclassified were not consistent. This suggests that by saving multiple trained XGBoost models and implementing a voting mechanism, we could potentially identify "hard-to-predict" cases. For instance, if a small subset of data points receives nearly split votes (e.g., 48% predicted as class 0 and 52% predicted as class 1), we might designate those cases as "unpredictable" by XGBoost alone. These difficult instances could then be isolated and handled by training a secondary model specifically designed for them.

In practice, however, the current dataset is too limited for this approach. After applying the voting scheme, only about 10 data points were identified as "unpredictable" using a 1/3 validation split, making us impractical to train an additional model (e.g., a Support Vector Machine) to improve performance. A rough estimate suggests that at least ten times (so that "unpredictable" data would have around 100) the current data volume would be necessary to implement the meta-model strategy effectively.

Appendix

A Python Notebook of Logistic Regression Model

STEP0: IMPORT NECESSARY LIBRARIES

In [27]:
```
# Import Necessary Libraries
import pandas as pd  # For data manipulation
import numpy as np  # For numerical computations
import matplotlib.pyplot as plt  # For plotting
import seaborn as sns  # For advanced data visualization
from sklearn.model_selection import train_test_split  # For splitting the da
from sklearn.preprocessing import LabelEncoder, StandardScaler  # For encodi
from sklearn.linear_model import LogisticRegression  # For logistic regressi
from sklearn.metrics import classification_report, confusion_matrix, accurac
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.feature_selection import RFE  # For Recursive Feature Eliminati

# Set Seaborn style for better aesthetics
sns.set(style="whitegrid")
```

STEP1: LOAD THE DATA SET

In [28]:
```
# Replace 'train.csv' with the actual path to your dataset if it's located e
try:
    df = pd.read_csv('train.csv')
    print("Dataset loaded successfully.")
except FileNotFoundError:
    print("The file 'train.csv' was not found. Please check the file path.")
    exit()
```

```
Dataset loaded successfully.
```

STEP2: EDA

In [29]:
```
# View the First Few Rows
print("\nFirst five rows of the dataset:")
print(df.head())
# Dataset Information
print("\nDataset Information:")
print(df.info())
# Check for Missing Values
print("\nMissing Values in Each Column:")
print(df.isnull().sum())
## notice the data set is perfect, don't have any missing values

# we remove the PatientID, Diagnostics and DoctorInCharge
all_variables = df.columns.difference(['PatientID', 'Diagnosis', 'DoctorInCh
print(all_variables)
print("Number of variables that we can use:", len(all_variables))
```

```
First five rows of the dataset:
   PatientID  Age  Gender  Ethnicity  EducationLevel        BMI  Smoking  \
0          1   67       0          3               0  37.205177        0
1          2   65       1          0               0  35.141843        1
2          3   62       0          1               1  17.875103        0
3          4   67       0          0               1  37.503437        1
4          5   65       1          0               2  29.187863        1

   AlcoholConsumption  PhysicalActivity  DietQuality  ...  MemoryComplaints
\
0           12.215677          7.780544     6.433890  ...                 1
1           17.111404          6.645284     1.112379  ...                 0
2           13.525546          9.585769     4.266008  ...                 0
3           19.952014          1.953946     6.797333  ...                 0
4            0.533209          8.759570     6.364302  ...                 1

   BehavioralProblems       ADL  Confusion  Disorientation  \
0                   0  6.009376          0               0
1                   0  7.519209          0               0
2                   0  8.573933          0               0
3                   0  6.217530          0               0
4                   0  5.193683          1               0

   PersonalityChanges  DifficultyCompletingTasks  Forgetfulness  Diagnosis
\
0                   0                          1              1          0
1                   0                          0              1          0
2                   0                          0              0          0
3                   0                          0              1          0
4                   0                          0              1          0

   DoctorInCharge
0       XXXConfid
1       XXXConfid
2       XXXConfid
3       XXXConfid
4       XXXConfid

[5 rows x 35 columns]

Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1504 entries, 0 to 1503
Data columns (total 35 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   PatientID                  1504 non-null   int64
 1   Age                        1504 non-null   int64
 2   Gender                     1504 non-null   int64
 3   Ethnicity                  1504 non-null   int64
 4   EducationLevel             1504 non-null   int64
 5   BMI                        1504 non-null   float64
 6   Smoking                    1504 non-null   int64
 7   AlcoholConsumption         1504 non-null   float64
 8   PhysicalActivity           1504 non-null   float64
 9   DietQuality                1504 non-null   float64
```

```
 10  SleepQuality               1504 non-null   float64
 11  FamilyHistoryAlzheimers    1504 non-null   int64
 12  CardiovascularDisease      1504 non-null   int64
 13  Diabetes                   1504 non-null   int64
 14  Depression                 1504 non-null   int64
 15  HeadInjury                 1504 non-null   int64
 16  Hypertension               1504 non-null   int64
 17  SystolicBP                 1504 non-null   int64
 18  DiastolicBP                1504 non-null   int64
 19  CholesterolTotal           1504 non-null   float64
 20  CholesterolLDL             1504 non-null   float64
 21  CholesterolHDL             1504 non-null   float64
 22  CholesterolTriglycerides   1504 non-null   float64
 23  MMSE                       1504 non-null   float64
 24  FunctionalAssessment       1504 non-null   float64
 25  MemoryComplaints           1504 non-null   int64
 26  BehavioralProblems         1504 non-null   int64
 27  ADL                        1504 non-null   float64
 28  Confusion                  1504 non-null   int64
 29  Disorientation             1504 non-null   int64
 30  PersonalityChanges         1504 non-null   int64
 31  DifficultyCompletingTasks  1504 non-null   int64
 32  Forgetfulness              1504 non-null   int64
 33  Diagnosis                  1504 non-null   int64
 34  DoctorInCharge             1504 non-null   object
dtypes: float64(12), int64(22), object(1)
memory usage: 411.4+ KB
None

Missing Values in Each Column:
PatientID                    0
Age                          0
Gender                       0
Ethnicity                    0
EducationLevel               0
BMI                          0
Smoking                      0
AlcoholConsumption           0
PhysicalActivity             0
DietQuality                  0
SleepQuality                 0
FamilyHistoryAlzheimers      0
CardiovascularDisease        0
Diabetes                     0
Depression                   0
HeadInjury                   0
Hypertension                 0
SystolicBP                   0
DiastolicBP                  0
CholesterolTotal             0
CholesterolLDL               0
CholesterolHDL               0
CholesterolTriglycerides     0
MMSE                         0
FunctionalAssessment         0
MemoryComplaints             0
```

```
BehavioralProblems          0
ADL                         0
Confusion                   0
Disorientation              0
PersonalityChanges          0
DifficultyCompletingTasks   0
Forgetfulness               0
Diagnosis                   0
DoctorInCharge              0
dtype: int64
Index(['ADL', 'Age', 'AlcoholConsumption', 'BMI', 'BehavioralProblems',
       'CardiovascularDisease', 'CholesterolHDL', 'CholesterolLDL',
       'CholesterolTotal', 'CholesterolTriglycerides', 'Confusion',
       'Depression', 'Diabetes', 'DiastolicBP', 'DietQuality',
       'DifficultyCompletingTasks', 'Disorientation', 'EducationLevel',
       'Ethnicity', 'FamilyHistoryAlzheimers', 'Forgetfulness',
       'FunctionalAssessment', 'Gender', 'HeadInjury', 'Hypertension', 'MMS
E',
       'MemoryComplaints', 'PersonalityChanges', 'PhysicalActivity',
       'SleepQuality', 'Smoking', 'SystolicBP'],
      dtype='object')
Number of variables that we can use: 32
```

EXplore problems within categorical variables

In [30]:
```
# find categorical variables
possible_categoricals = df.select_dtypes(include=['int64', 'object']).columr
categorical = []
for col in possible_categoricals:
    if df[col].dtype == 'object' or df[col].nunique() < 10:  # Threshold car
        categorical.append(col)
print("Number of Categorical variables:", len(categorical))
print("Categorical variables:", categorical)

# we already know that there are no missing values in categorical variables
# Now, I will check the frequency counts of categorical variables.
for var in categorical:
    print(df[var].value_counts())

# By checking the date set, we don't need to encode and reformate :)
```

```
Number of Categorical variables: 19
Categorical variables: ['Gender', 'Ethnicity', 'EducationLevel', 'Smoking',
'FamilyHistoryAlzheimers', 'CardiovascularDisease', 'Diabetes', 'Depressio
n', 'HeadInjury', 'Hypertension', 'MemoryComplaints', 'BehavioralProblems',
'Confusion', 'Disorientation', 'PersonalityChanges', 'DifficultyCompletingTa
sks', 'Forgetfulness', 'Diagnosis', 'DoctorInCharge']
Gender
1    765
0    739
Name: count, dtype: int64
Ethnicity
0    890
1    309
2    154
3    151
Name: count, dtype: int64
EducationLevel
1    588
2    454
0    311
3    151
Name: count, dtype: int64
Smoking
0    1077
1     427
Name: count, dtype: int64
FamilyHistoryAlzheimers
0    1136
1     368
Name: count, dtype: int64
CardiovascularDisease
0    1302
1     202
Name: count, dtype: int64
Diabetes
0    1264
1     240
Name: count, dtype: int64
Depression
0    1191
1     313
Name: count, dtype: int64
HeadInjury
0    1361
1     143
Name: count, dtype: int64
Hypertension
0    1276
1     228
Name: count, dtype: int64
MemoryComplaints
0    1195
1     309
Name: count, dtype: int64
BehavioralProblems
0    1276
```

```
1     228
Name: count, dtype: int64
Confusion
0    1199
1     305
Name: count, dtype: int64
Disorientation
0    1269
1     235
Name: count, dtype: int64
PersonalityChanges
0    1268
1     236
Name: count, dtype: int64
DifficultyCompletingTasks
0    1260
1     244
Name: count, dtype: int64
Forgetfulness
0    1053
1     451
Name: count, dtype: int64
Diagnosis
0    972
1    532
Name: count, dtype: int64
DoctorInCharge
XXXConfid    1504
Name: count, dtype: int64
```

Use one hot encoding for non-binary categorical variables

In [31]:
```
non_binary_cat=['Ethnicity', 'EducationLevel']

pd.get_dummies(df.Ethnicity, drop_first=True, dtype=int).head()
```

Out[31]:

	1	2	3
0	0	0	1
1	0	0	0
2	1	0	0
3	0	0	0
4	0	0	0

In [32]:
```
pd.get_dummies(df.EducationLevel, drop_first=True, dtype=int).head()
```

Out[32]:

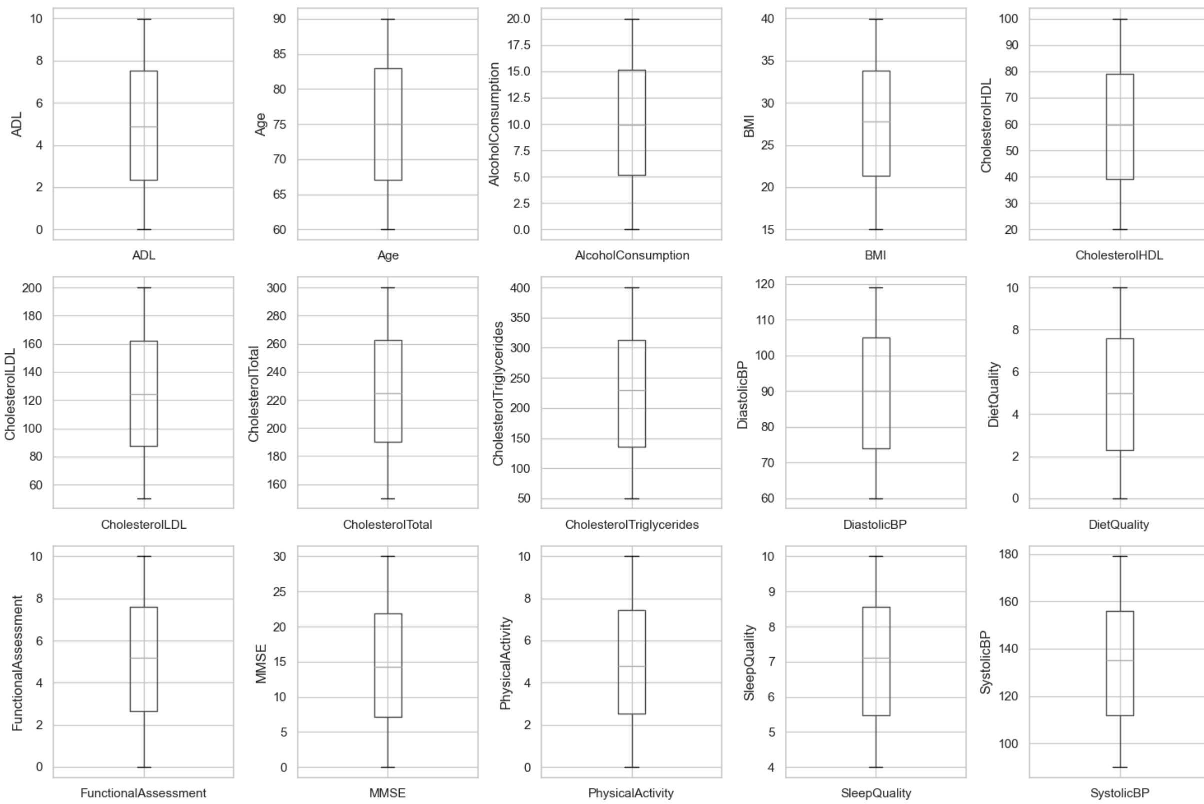
	1	2	3
0	0	0	0
1	0	0	0
2	1	0	0
3	1	0	0
4	0	1	0

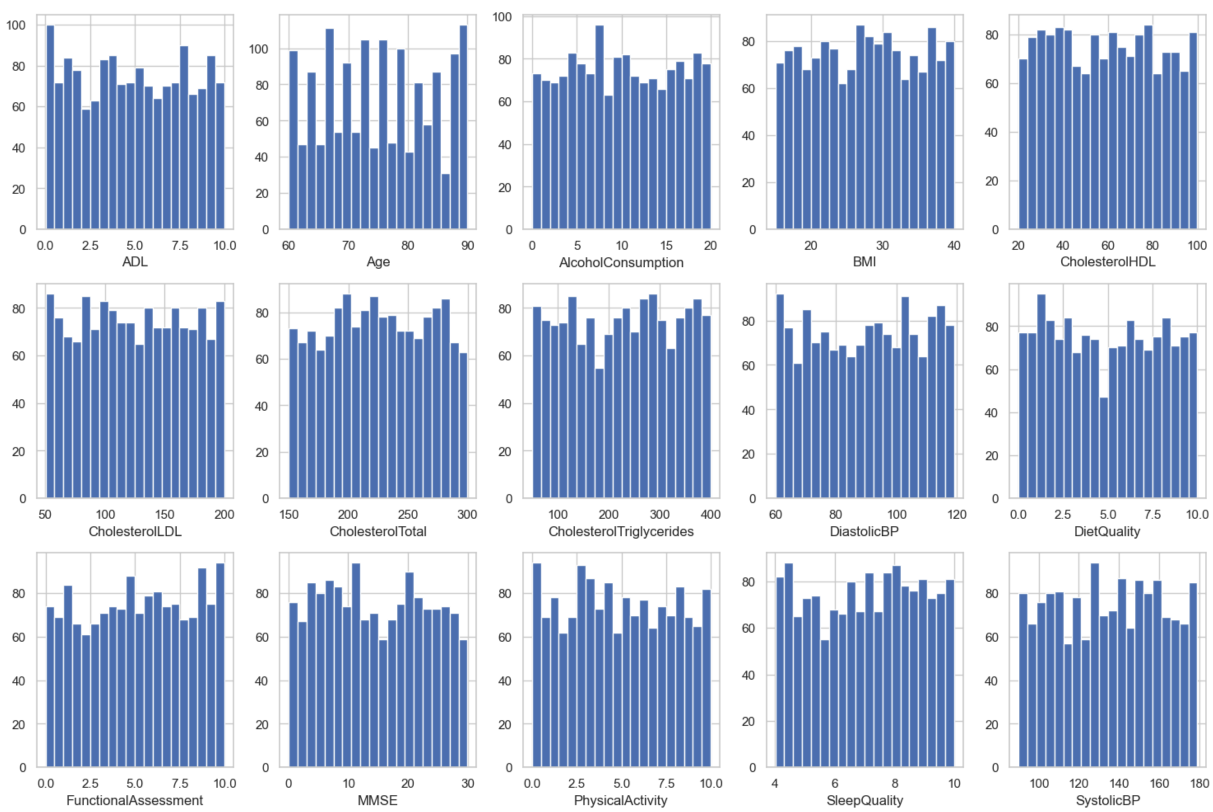
Explore problems within numerical variables

In [33]:

```
# find numerical variables
numerical = all_variables.difference(categorical)
print('There are {} numerical variables\n'.format(len(numerical)))
print('The numerical variables are :', numerical)
# we already know that there are no missing values in categorical variables

# view summary statistics in numerical variables
print(round(df[numerical].describe()))


# draw boxplots to visualize outliers
plt.figure(figsize=(15, 10))
for i, col in enumerate(numerical, 1):
    plt.subplot(3, 5, i)
    fig = df.boxplot(column=col)
    fig.set_title('')
    fig.set_ylabel(col)

plt.tight_layout()  # Adjust layout to prevent overlap
plt.show()

# plot histogram to check distribution
plt.figure(figsize=(15, 10))
for i, col in enumerate(numerical, 1):
    plt.subplot(3, 5, i)
    fig = df[col].hist(bins = 20)
    fig.set_xlabel(col)

plt.tight_layout()  # Adjust layout to prevent overlap
plt.show()
```

```
There are 15 numerical variables

The numerical variables are : Index(['ADL', 'Age', 'AlcoholConsumption', 'BM
I', 'CholesterolHDL',
       'CholesterolLDL', 'CholesterolTotal', 'CholesterolTriglycerides',
       'DiastolicBP', 'DietQuality', 'FunctionalAssessment', 'MMSE',
       'PhysicalActivity', 'SleepQuality', 'SystolicBP'],
      dtype='object')
          ADL     Age  AlcoholConsumption     BMI  CholesterolHDL  \
count  1504.0  1504.0              1504.0  1504.0          1504.0   
mean      5.0    75.0                10.0    28.0            60.0   
std       3.0     9.0                 6.0     7.0            23.0   
min       0.0    60.0                 0.0    15.0            20.0   
25%       2.0    67.0                 5.0    21.0            39.0   
50%       5.0    75.0                10.0    28.0            60.0   
75%       8.0    83.0                15.0    34.0            79.0   
max      10.0    90.0                20.0    40.0           100.0   

       CholesterolLDL  CholesterolTotal  CholesterolTriglycerides  \
count          1504.0            1504.0                    1504.0   
mean            125.0             225.0                     227.0   
std              43.0              42.0                     102.0   
min              50.0             150.0                      50.0   
25%              88.0             190.0                     136.0   
50%             125.0             224.0                     230.0   
75%             162.0             262.0                     313.0   
max             200.0             300.0                     400.0   

       DiastolicBP  DietQuality  FunctionalAssessment    MMSE  \
count       1504.0       1504.0                1504.0  1504.0   
mean          90.0          5.0                   5.0    15.0   
std           18.0          3.0                   3.0     9.0   
min           60.0          0.0                   0.0     0.0   
25%           74.0          2.0                   3.0     7.0   
50%           90.0          5.0                   5.0    14.0   
75%          105.0          8.0                   8.0    22.0   
max          119.0         10.0                  10.0    30.0   

       PhysicalActivity  SleepQuality  SystolicBP  
count            1504.0        1504.0      1504.0  
mean                5.0           7.0       135.0  
std                 3.0           2.0        26.0  
min                 0.0           4.0        90.0  
25%                 3.0           5.0       112.0  
50%                 5.0           7.0       135.0  
75%                 7.0           9.0       156.0  
max                10.0          10.0       179.0  
```





STEP3: Declare feature vector and target variable

In [34]:
```
# Define features (X) and target variable (y)
X = df.drop(['PatientID', 'Diagnosis', "DoctorInCharge"], axis=1)
y = df['Diagnosis']
print("\nFeatures and target variable defined.")
```

```
Features and target variable defined.
```

STEP4: Split data into separate training and test set

In [35]:
```
X_train, y_train,= X, y
X_train.shape
```

Out[35]:
```
(1504, 32)
```

STEP5: Feature Engineering Notice there are no missing values for all variables, so we don't need to engineering missing values here. Since we don't see any skewed distribution for our numerical variables, we don't deal with outliers. We don't need to encode categorical variables

In [36]:
```
X_train = pd.concat(
    [
        X_train[numerical],  # Include numerical features
        pd.get_dummies(X_train.Ethnicity, prefix="Ethnicity", dtype=int),
        pd.get_dummies(X_train.EducationLevel, prefix="EducationLevel", dtyp
    ],
    axis=1
)
```

In [37]:
```
X_train.head()
```

Out[37]:

	ADL	Age	AlcoholConsumption	BMI	CholesterolHDL	CholesterolLDL	C
0	6.009376	67	12.215677	37.205177	78.049441	118.891075	
1	7.519209	65	17.111404	35.141843	21.152397	100.588771	
2	8.573933	62	13.525546	17.875103	36.973033	184.974822	
3	6.217530	67	19.952014	37.503437	62.169786	150.744105	
4	5.193683	65	0.533209	29.187863	82.865450	125.429630	

5 rows × 23 columns

STEP6: Feature scaling

In [38]:
```
X_train.describe()
cols = X_train.columns
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_train = pd.DataFrame(X_train, columns=[cols])
X_train.describe()
```

Out[38]:

	ADL	Age	AlcoholConsumption	BMI	CholesterolHDL
count	1504.000000	1504.000000	1504.000000	1504.000000	1504.000000
mean	0.491476	0.496853	0.501754	0.503037	0.494035
std	0.293748	0.298336	0.287435	0.288628	0.288981
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.236172	0.233333	0.260293	0.254997	0.239369
50%	0.488900	0.500000	0.496456	0.511689	0.495028
75%	0.753675	0.766667	0.757444	0.752965	0.736498
max	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 23 columns

STEP7: Model training

This is the full model with usual ridge penalty $\lambda = \frac{1}{C} = 1$

In [39]:

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

# Define the range of C values
C_values = np.arange(0.01, 5.01, 0.01)

# Initialize variables to store the best C and its score
best_C = None
best_score = -np.inf  # Start with a very low score
scores_dict = {}  # To store scores for all C values

# Loop over each value of C
for C in C_values:
    # Instantiate the model with the current C value
    logreg = LogisticRegression(penalty='l1', C=C, solver='liblinear', rando

    # Apply 5-Fold Cross Validation
    scores = cross_val_score(logreg, X_train, y_train, cv=5, scoring='accura

    # Compute Average cross-validation score
    avg_score = scores.mean()
    scores_dict[C] = avg_score  # Store the score for this C value

    # Update best C and score if this is the best so far
    if avg_score > best_score:
        best_C = C
        best_score = avg_score

# Print the best C and its score
print("\nBest C value:", best_C)
print("Best cross-validation score: {:.4f}".format(best_score))
```

```
Best C value: 0.14
Best cross-validation score: 0.7806
```

In [40]:

```python
# Define the range of C values
C_values = np.arange(0.01, 5.01, 0.01)

# Initialize variables to store the best C and its score
best_C = None
best_score = -np.inf  # Start with a very low score
scores_dict = {}  # To store scores for all C values

# Loop over each value of C
for C in C_values:
    # Instantiate the model with the current C value
    logreg = LogisticRegression(penalty='l2', C=C, solver='liblinear', rando

    # Apply 5-Fold Cross Validation
    scores = cross_val_score(logreg, X_train, y_train, cv=5, scoring='accura

    # Compute Average cross-validation score
    avg_score = scores.mean()
    scores_dict[C] = avg_score  # Store the score for this C value


    # Update best C and score if this is the best so far
    if avg_score > best_score:
        best_C = C
        best_score = avg_score

# Print the best C and its score
print("\nBest C value:", best_C)
print("Best cross-validation score: {:.4f}".format(best_score))
```

```
Best C value: 1.28
Best cross-validation score: 0.7753
```

In [41]:

```python
final_model1 = LogisticRegression(penalty='l1', C=0.14, solver='liblinear',
final_model1.fit(X_train, y_train)
```

Out[41]:

```
LogisticRegression

LogisticRegression(C=0.14, penalty='l1', random_state=0, solver='li
blinear')
```

In [42]:

```python
final_model2 = LogisticRegression(penalty='l2', C=1.28, solver='liblinear',
final_model2.fit(X_train, y_train)
```

Out[42]:

```
LogisticRegression

LogisticRegression(C=1.28, random_state=0, solver='liblinear')
```

STEP10: Confusion matrix

A confusion matrix is a tool for summarizing the performance of a classification algorithm. A confusion matrix will give us a clear picture of classification model performance and the types of errors produced by the model. It gives us a summary of correct and incorrect predictions broken down by each category. The summary is represented in a tabular form.

Four types of outcomes are possible while evaluating a classification model performance. These four outcomes are described below:

True Positives (TP) – True Positives occur when we predict an observation belongs to a certain class and the observation actually belongs to that class.

True Negatives (TN) – True Negatives occur when we predict an observation does not belong to a certain class and the observation actually does not belong to that class.

False Positives (FP) – False Positives occur when we predict an observation belongs to a certain class but the observation actually does not belong to that class. This type of error is called Type I error.

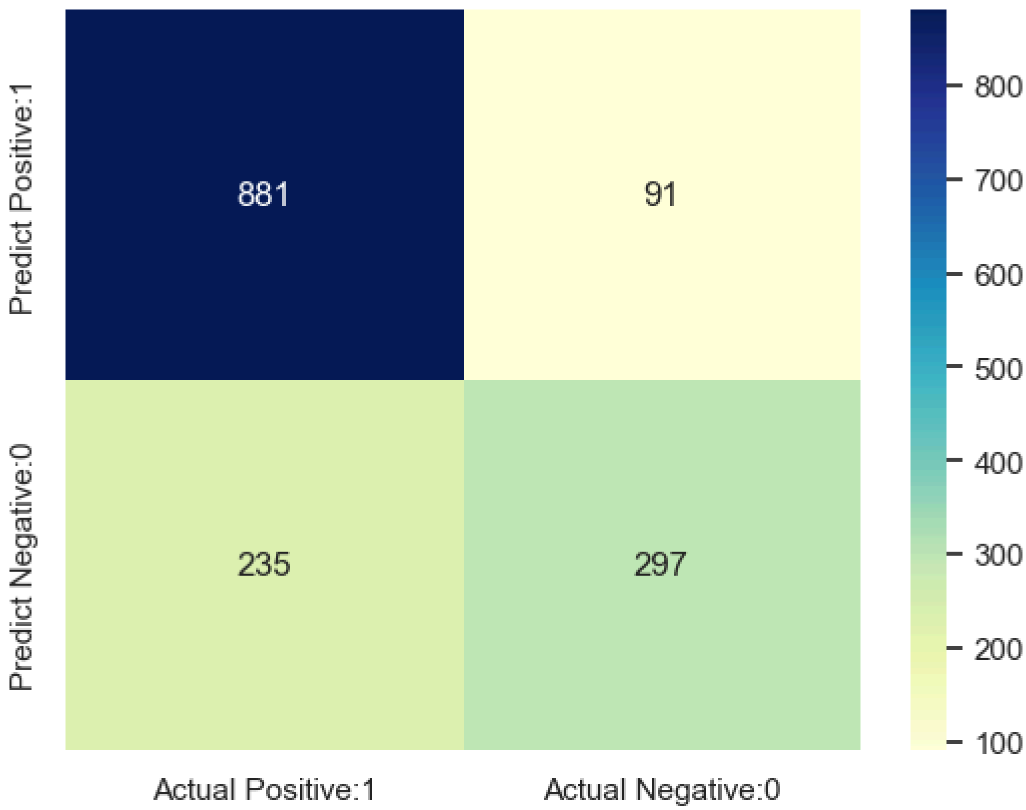
False Negatives (FN) – False Negatives occur when we predict an observation does not belong to a certain class but the observation actually belongs to that class. This is a very serious error and it is called Type II error.

These four outcomes are summarized in a confusion matrix given below.

In [43]:
```
y_pred_test1 = final_model1.predict(X_train)
y_pred_test1
# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_train, y_pred_test1)

print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])

# visualize confusion matrix with seaborn heatmap

cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Nega
                                 index=['Predict Positive:1', 'Predict Negat

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

```
Confusion matrix

 [[881  91]
 [235 297]]

True Positives(TP) =  881

True Negatives(TN) =  297

False Positives(FP) =  91

False Negatives(FN) =  235
```

Out[43]: <Axes: >



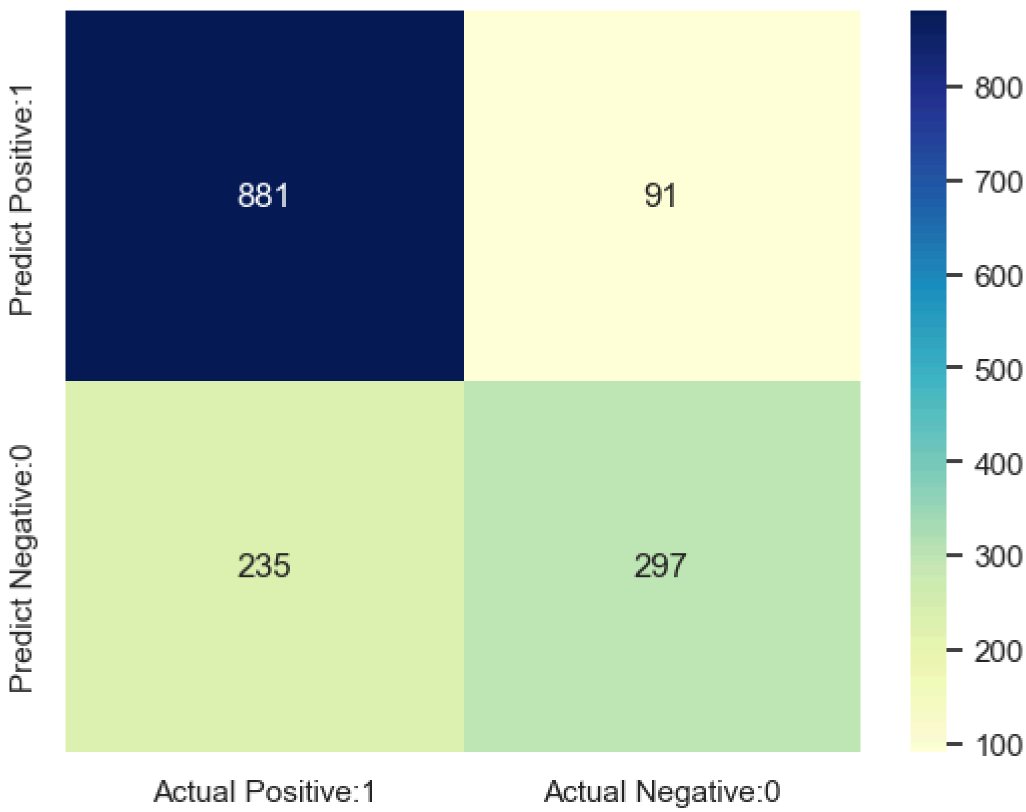
In [44]:

```python
y_pred_test2 = final_model2.predict(X_train)
y_pred_test2
# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_train, y_pred_test1)

print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])
```

```
# visualize confusion matrix with seaborn heatmap

cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Nega
                                  index=['Predict Positive:1', 'Predict Negat

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

```
Confusion matrix

 [[881  91]
 [235 297]]

True Positives(TP) =  881

True Negatives(TN) =  297

False Positives(FP) =  91

False Negatives(FN) =  235
```

Out[44]: <Axes: >



STEP11: Classification metrices

In [45]:
```
from sklearn.metrics import classification_report

print(classification_report(y_train, y_pred_test1))
TP = cm[0,0]
TN = cm[1,1]
FP = cm[0,1]
FN = cm[1,0]
```

```
# print classification accuracy

classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)

print('Classification accuracy : {0:0.4f}'.format(classification_accuracy))

# print classification error

classification_error = (FP + FN) / float(TP + TN + FP + FN)

print('Classification error : {0:0.4f}'.format(classification_error))

# print precision score

precision = TP / float(TP + FP)


print('Precision : {0:0.4f}'.format(precision))

recall = TP / float(TP + FN)

print('Recall or Sensitivity : {0:0.4f}'.format(recall))

true_positive_rate = TP / float(TP + FN)


print('True Positive Rate : {0:0.4f}'.format(true_positive_rate))

false_positive_rate = FP / float(FP + TN)


print('False Positive Rate : {0:0.4f}'.format(false_positive_rate))

specificity = TN / (TN + FP)

print('Specificity : {0:0.4f}'.format(specificity))
```

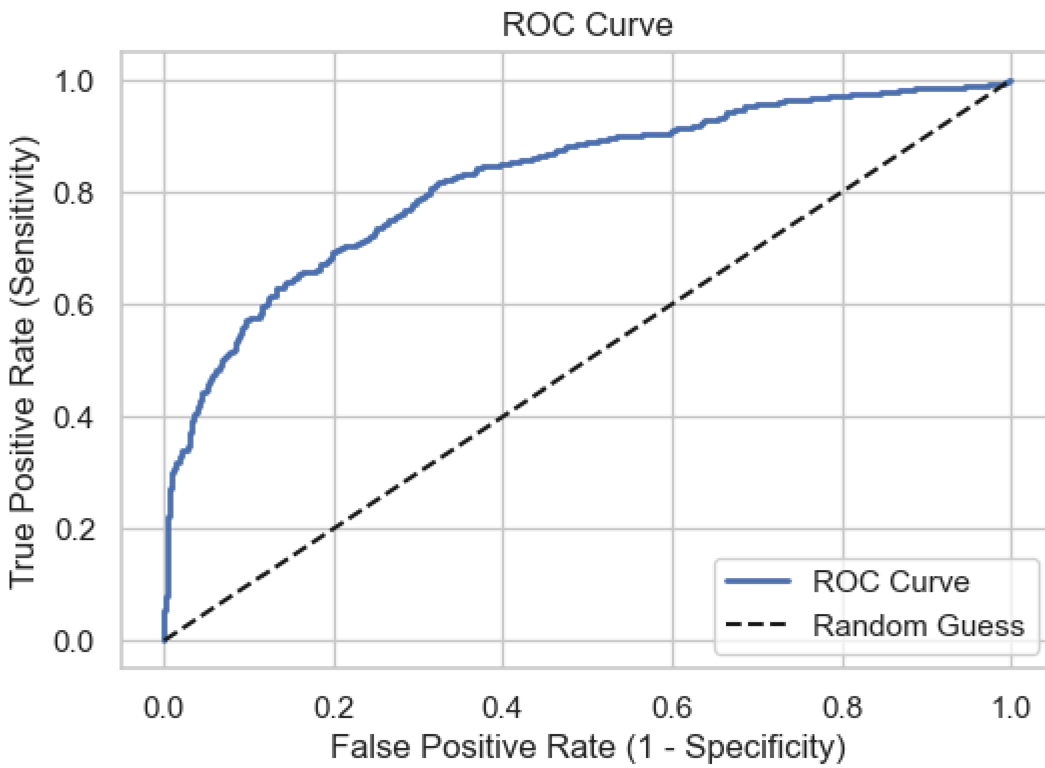
```
              precision    recall  f1-score   support

           0       0.79      0.91      0.84       972
           1       0.77      0.56      0.65       532

    accuracy                           0.78      1504
   macro avg       0.78      0.73      0.74      1504
weighted avg       0.78      0.78      0.77      1504

Classification accuracy : 0.7832
Classification error : 0.2168
Precision : 0.9064
Recall or Sensitivity : 0.7894
True Positive Rate : 0.7894
False Positive Rate : 0.2345
Specificity : 0.7655
```

In [46]:
```
from sklearn.metrics import classification_report

print(classification_report(y_train, y_pred_test2))
TP = cm[0,0]
TN = cm[1,1]
FP = cm[0,1]
FN = cm[1,0]

# print classification accuracy

classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)

print('Classification accuracy : {0:0.4f}'.format(classification_accuracy))

# print classification error

classification_error = (FP + FN) / float(TP + TN + FP + FN)

print('Classification error : {0:0.4f}'.format(classification_error))

# print precision score

precision = TP / float(TP + FP)


print('Precision : {0:0.4f}'.format(precision))

recall = TP / float(TP + FN)

print('Recall or Sensitivity : {0:0.4f}'.format(recall))

true_positive_rate = TP / float(TP + FN)


print('True Positive Rate : {0:0.4f}'.format(true_positive_rate))

false_positive_rate = FP / float(FP + TN)


print('False Positive Rate : {0:0.4f}'.format(false_positive_rate))

specificity = TN / (TN + FP)

print('Specificity : {0:0.4f}'.format(specificity))
```

```
              precision    recall  f1-score   support

           0       0.80      0.88      0.84       972
           1       0.73      0.60      0.66       532

    accuracy                           0.78      1504
   macro avg       0.77      0.74      0.75      1504
weighted avg       0.78      0.78      0.78      1504

Classification accuracy : 0.7832
Classification error : 0.2168
Precision : 0.9064
Recall or Sensitivity : 0.7894
True Positive Rate : 0.7894
False Positive Rate : 0.2345
Specificity : 0.7655
```

STEP12: ROC-AOC

In [47]:
```
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt

# Assuming y_test has binary values 0 and 1
y_pred1 = final_model1.predict_proba(X_train)[:, 1]  # Get positive class pr

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_train, y_pred1, pos_label=1)  # Change po

# Plot the ROC curve
plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, linewidth=2, label='ROC Curve')
plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
plt.rcParams['font.size'] = 12
plt.title('ROC Curve')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.legend(loc='lower right')
plt.show()
```



In [48]:
```
# compute ROC AUC

from sklearn.metrics import roc_auc_score

ROC_AUC = roc_auc_score(y_train, y_pred1)

print('ROC AUC : {:.4f}'.format(ROC_AUC))
```

```
ROC AUC : 0.8249
```

Final step: test our model

In [49]:
```
# Load and preprocess the test data
test_data = pd.read_csv('test.csv')
X_test = test_data.drop(['PatientID', 'DoctorInCharge'], axis=1)
X_test_final = pd.concat(
    [
        X_test[numerical],  # Include numerical features
        pd.get_dummies(X_test.Ethnicity, prefix="Ethnicity", dtype=int),
        pd.get_dummies(X_test.EducationLevel, prefix="EducationLevel", dtype
    ],
    axis=1
)

X_test_final_scaled = scaler.transform(X_test_final)

# Make predictions with a custom threshold of 0.35
test_probabilities = final_model1.predict_proba(X_test_final_scaled)[:, 1]
test_predictions = (test_probabilities >= 0.37).astype(int)  # Apply the 0.4

print("Predictions made on test data with a threshold of 0.37.")
```

```
# Prepare and save the submission file
submission = pd.DataFrame({
    'PatientID': test_data['PatientID'],
    'Diagnosis': test_predictions
})

# Save to CSV
submission.to_csv('test_predictions.csv', index=False)
print("\nSubmission file 'test_predictions.csv' created successfully with th
print(submission.head())
```

```
Predictions made on test data with a threshold of 0.37.

Submission file 'test_predictions.csv' created successfully with the followi
ng format:
   PatientID  Diagnosis
0       1505          1
1       1506          0
2       1507          0
3       1508          1
4       1509          1
```

B Python Notebook of XGBoosting

STEP0: IMPORT NECESSARY LIBRARIES

In [13]:
```python
# Import Necessary Libraries
import pandas as pd  # For data manipulation
import numpy as np  # For numerical computations
import matplotlib.pyplot as plt  # For plotting
import seaborn as sns  # For advanced data visualization
from sklearn.model_selection import train_test_split  # For splitting the datase
from sklearn.preprocessing import LabelEncoder, StandardScaler  # For encoding a
from sklearn.linear_model import LogisticRegression  # For logistic regression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_sc
from statsmodels.stats.outliers_influence import variance_inflation_factor  # Fo
from sklearn.feature_selection import RFE  # For Recursive Feature Elimination

# Set Seaborn style for better aesthetics
sns.set(style="whitegrid")
```

STEP1: LOAD THE DATA SET

In [14]:
```python
# Replace 'train.csv' with the actual path to your dataset if it's located elsew
try:
    df = pd.read_csv('train.csv')
    print("Dataset loaded successfully.")
except FileNotFoundError:
    print("The file 'train.csv' was not found. Please check the file path.")
    exit()
```

```
Dataset loaded successfully.
```

STEP2: EDA

In [15]:
```python
# View the First Few Rows
print("\nFirst five rows of the dataset:")
print(df.head())
# Dataset Information
print("\nDataset Information:")
print(df.info())
# Check for Missing Values
print("\nMissing Values in Each Column:")
print(df.isnull().sum())
## notice the data set is perfect, don't have any missing values

# we remove the PatientID, Diagnostics and DoctorInCharge
all_variables = df.columns.difference(['PatientID', 'Diagnosis', 'DoctorInCharge
print(all_variables)
print("Number of variables that we can use:", len(all_variables))
```

```
First five rows of the dataset:
   PatientID  Age  Gender  Ethnicity  EducationLevel        BMI  Smoking  \
0          1   67       0          3               0  37.205177        0
1          2   65       1          0               0  35.141843        1
2          3   62       0          1               1  17.875103        0
3          4   67       0          0               1  37.503437        1
4          5   65       1          0               2  29.187863        1

   AlcoholConsumption  PhysicalActivity  DietQuality  ...  MemoryComplaints  \
0           12.215677          7.780544     6.433890  ...                 1
1           17.111404          6.645284     1.112379  ...                 0
2           13.525546          9.585769     4.266008  ...                 0
3           19.952014          1.953946     6.797333  ...                 0
4            0.533209          8.759570     6.364302  ...                 1

   BehavioralProblems       ADL  Confusion  Disorientation  \
0                   0  6.009376          0               0
1                   0  7.519209          0               0
2                   0  8.573933          0               0
3                   0  6.217530          0               0
4                   0  5.193683          1               0

   PersonalityChanges  DifficultyCompletingTasks  Forgetfulness  Diagnosis  \
0                   0                          1              1          0
1                   0                          0              1          0
2                   0                          0              0          0
3                   0                          0              1          0
4                   0                          0              1          0

   DoctorInCharge
0       XXXConfid
1       XXXConfid
2       XXXConfid
3       XXXConfid
4       XXXConfid

[5 rows x 35 columns]

Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1504 entries, 0 to 1503
Data columns (total 35 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   PatientID                  1504 non-null   int64
 1   Age                        1504 non-null   int64
 2   Gender                     1504 non-null   int64
 3   Ethnicity                  1504 non-null   int64
 4   EducationLevel             1504 non-null   int64
 5   BMI                        1504 non-null   float64
 6   Smoking                    1504 non-null   int64
 7   AlcoholConsumption         1504 non-null   float64
 8   PhysicalActivity           1504 non-null   float64
 9   DietQuality                1504 non-null   float64
 10  SleepQuality               1504 non-null   float64
 11  FamilyHistoryAlzheimers    1504 non-null   int64
 12  CardiovascularDisease      1504 non-null   int64
 13  Diabetes                   1504 non-null   int64
 14  Depression                 1504 non-null   int64
 15  HeadInjury                 1504 non-null   int64
```

```
 16  Hypertension               1504 non-null   int64  
 17  SystolicBP                 1504 non-null   int64  
 18  DiastolicBP                1504 non-null   int64  
 19  CholesterolTotal           1504 non-null   float64
 20  CholesterolLDL             1504 non-null   float64
 21  CholesterolHDL             1504 non-null   float64
 22  CholesterolTriglycerides   1504 non-null   float64
 23  MMSE                       1504 non-null   float64
 24  FunctionalAssessment       1504 non-null   float64
 25  MemoryComplaints           1504 non-null   int64  
 26  BehavioralProblems         1504 non-null   int64  
 27  ADL                        1504 non-null   float64
 28  Confusion                  1504 non-null   int64  
 29  Disorientation             1504 non-null   int64  
 30  PersonalityChanges         1504 non-null   int64  
 31  DifficultyCompletingTasks  1504 non-null   int64  
 32  Forgetfulness              1504 non-null   int64  
 33  Diagnosis                  1504 non-null   int64  
 34  DoctorInCharge             1504 non-null   object 
dtypes: float64(12), int64(22), object(1)
memory usage: 411.4+ KB
None

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

```
Index(['ADL', 'Age', 'AlcoholConsumption', 'BMI', 'BehavioralProblems',
       'CardiovascularDisease', 'CholesterolHDL', 'CholesterolLDL',
       'CholesterolTotal', 'CholesterolTriglycerides', 'Confusion',
       'Depression', 'Diabetes', 'DiastolicBP', 'DietQuality',
       'DifficultyCompletingTasks', 'Disorientation', 'EducationLevel',
       'Ethnicity', 'FamilyHistoryAlzheimers', 'Forgetfulness',
       'FunctionalAssessment', 'Gender', 'HeadInjury', 'Hypertension', 'MMSE',
       'MemoryComplaints', 'PersonalityChanges', 'PhysicalActivity',
       'SleepQuality', 'Smoking', 'SystolicBP'],
      dtype='object')
Number of variables that we can use: 32
```

Explore problems within categorical variables

In [16]:

```python
# find categorical variables
possible_categoricals = df.select_dtypes(include=['int64', 'object']).columns
categorical = []
for col in possible_categoricals:
    if df[col].dtype == 'object' or df[col].nunique() < 10:  # Threshold can be
        categorical.append(col)
print("Number of Categorical variables:", len(categorical))
print("Categorical variables:", categorical)

# we already know that there are no missing values in categorical variables :)
# Now, I will check the frequency counts of categorical variables.
for var in categorical:
    print(df[var].value_counts())

# By checking the date set, we don't need to encode and reformate :)
```

```
Number of Categorical variables: 19
Categorical variables: ['Gender', 'Ethnicity', 'EducationLevel', 'Smoking', 'Fami
lyHistoryAlzheimers', 'CardiovascularDisease', 'Diabetes', 'Depression', 'HeadInj
ury', 'Hypertension', 'MemoryComplaints', 'BehavioralProblems', 'Confusion', 'Dis
orientation', 'PersonalityChanges', 'DifficultyCompletingTasks', 'Forgetfulness',
'Diagnosis', 'DoctorInCharge']
Gender
1    765
0    739
Name: count, dtype: int64
Ethnicity
0    890
1    309
2    154
3    151
Name: count, dtype: int64
EducationLevel
1    588
2    454
0    311
3    151
Name: count, dtype: int64
Smoking
0    1077
1     427
Name: count, dtype: int64
FamilyHistoryAlzheimers
0    1136
1     368
Name: count, dtype: int64
CardiovascularDisease
0    1302
1     202
Name: count, dtype: int64
Diabetes
0    1264
1     240
Name: count, dtype: int64
Depression
0    1191
1     313
Name: count, dtype: int64
HeadInjury
0    1361
1     143
Name: count, dtype: int64
Hypertension
0    1276
1     228
Name: count, dtype: int64
MemoryComplaints
0    1195
1     309
Name: count, dtype: int64
BehavioralProblems
0    1276
1     228
Name: count, dtype: int64
Confusion
0    1199
```

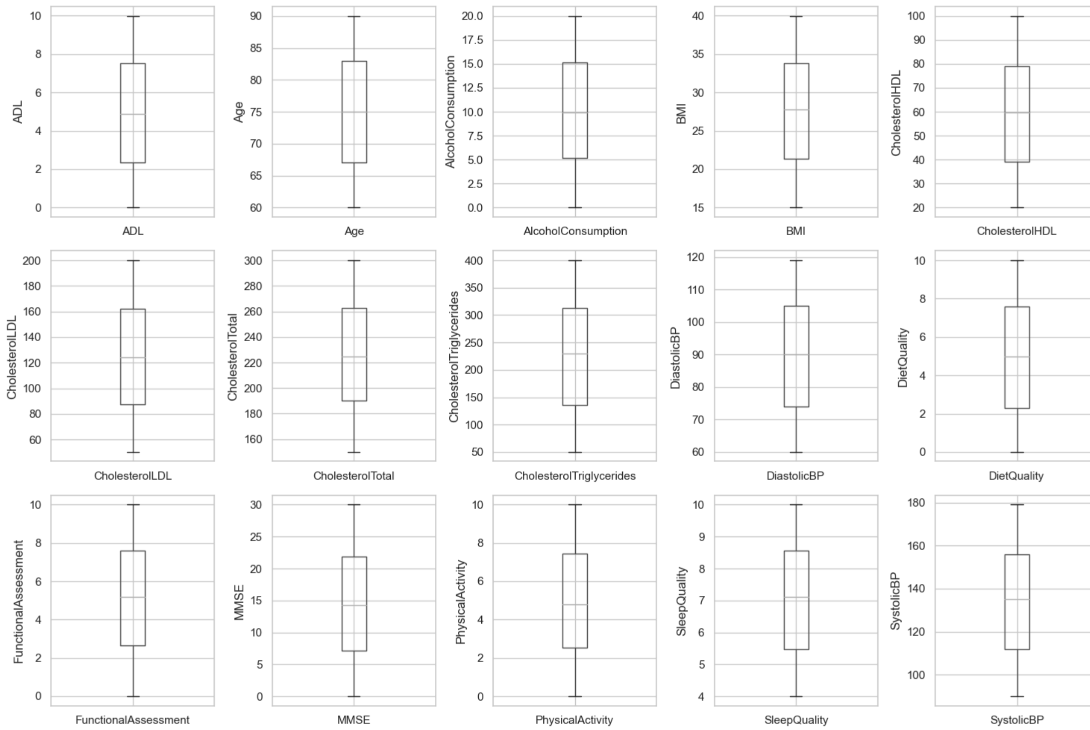
```
1     305
Name: count, dtype: int64
Disorientation
0    1269
1     235
Name: count, dtype: int64
PersonalityChanges
0    1268
1     236
Name: count, dtype: int64
DifficultyCompletingTasks
0    1260
1     244
Name: count, dtype: int64
Forgetfulness
0    1053
1     451
Name: count, dtype: int64
Diagnosis
0    972
1    532
Name: count, dtype: int64
DoctorInCharge
XXXConfid    1504
Name: count, dtype: int64
```

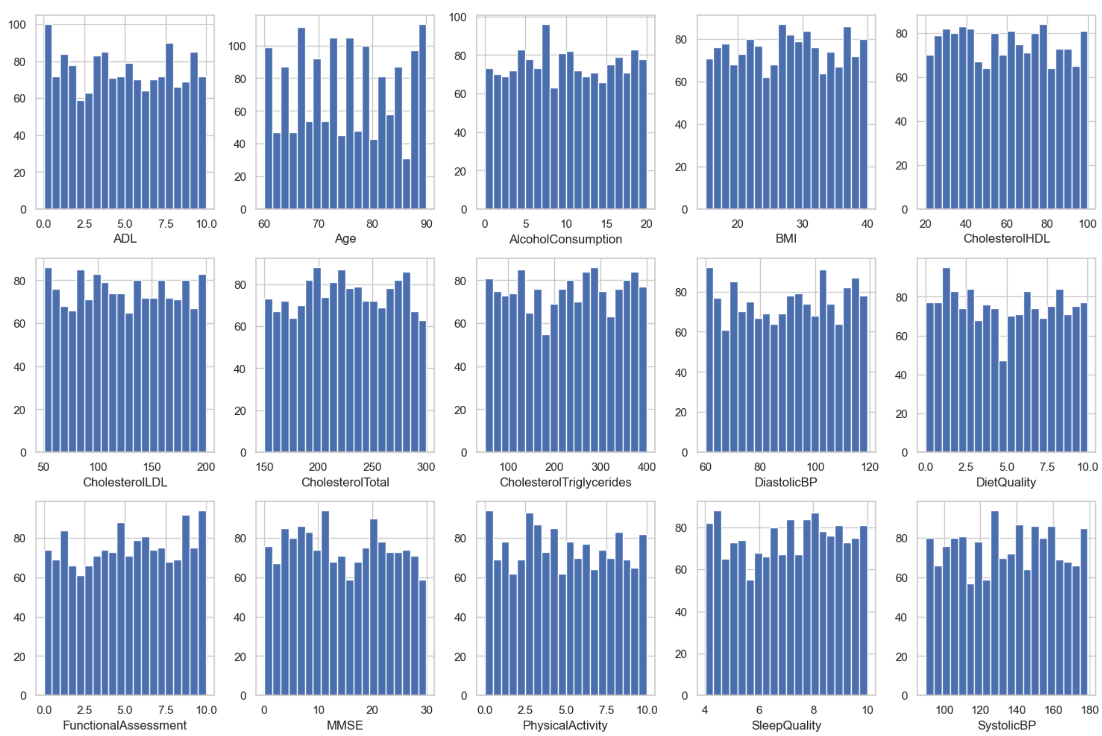
Explore problems within numerical variables

In [17]:

```
# find numerical variables
numerical = all_variables.difference(categorical)
print('There are {} numerical variables\n'.format(len(numerical)))
print('The numerical variables are :', numerical)
# we already know that there are no missing values in categorical variables :)

# view summary statistics in numerical variables
print(round(df[numerical].describe()))


# draw boxplots to visualize outliers
plt.figure(figsize=(15, 10))
for i, col in enumerate(numerical, 1):
    plt.subplot(3, 5, i)
    fig = df.boxplot(column=col)
    fig.set_title('')
    fig.set_ylabel(col)

plt.tight_layout()  # Adjust layout to prevent overlap
plt.show()

# plot histogram to check distribution
plt.figure(figsize=(15, 10))
for i, col in enumerate(numerical, 1):
    plt.subplot(3, 5, i)
    fig = df[col].hist(bins = 20)
    fig.set_xlabel(col)

plt.tight_layout()  # Adjust layout to prevent overlap
plt.show()
```

```
There are 15 numerical variables

The numerical variables are : Index(['ADL', 'Age', 'AlcoholConsumption', 'BMI',
'CholesterolHDL',
       'CholesterolLDL', 'CholesterolTotal', 'CholesterolTriglycerides',
       'DiastolicBP', 'DietQuality', 'FunctionalAssessment', 'MMSE',
       'PhysicalActivity', 'SleepQuality', 'SystolicBP'],
      dtype='object')
          ADL     Age  AlcoholConsumption     BMI  CholesterolHDL  \
count  1504.0  1504.0              1504.0  1504.0          1504.0
mean      5.0    75.0                10.0    28.0            60.0
std       3.0     9.0                 6.0     7.0            23.0
min       0.0    60.0                 0.0    15.0            20.0
25%       2.0    67.0                 5.0    21.0            39.0
50%       5.0    75.0                10.0    28.0            60.0
75%       8.0    83.0                15.0    34.0            79.0
max      10.0    90.0                20.0    40.0           100.0

       CholesterolLDL  CholesterolTotal  CholesterolTriglycerides  \
count          1504.0            1504.0                    1504.0
mean            125.0             225.0                     227.0
std              43.0              42.0                     102.0
min              50.0             150.0                      50.0
25%              88.0             190.0                     136.0
50%             125.0             224.0                     230.0
75%             162.0             262.0                     313.0
max             200.0             300.0                     400.0

       DiastolicBP  DietQuality  FunctionalAssessment    MMSE  \
count       1504.0       1504.0                1504.0  1504.0
mean          90.0          5.0                   5.0    15.0
std           18.0          3.0                   3.0     9.0
min           60.0          0.0                   0.0     0.0
25%           74.0          2.0                   3.0     7.0
50%           90.0          5.0                   5.0    14.0
75%          105.0          8.0                   8.0    22.0
max          119.0         10.0                  10.0    30.0

       PhysicalActivity  SleepQuality  SystolicBP
count            1504.0        1504.0      1504.0
mean                5.0           7.0       135.0
std                 3.0           2.0        26.0
min                 0.0           4.0        90.0
25%                 3.0           5.0       112.0
50%                 5.0           7.0       135.0
75%                 7.0           9.0       156.0
max                10.0          10.0       179.0
```





STEP3: Declare feature vector and target variable

In [18]:
```python
# Define features (X) and target variable (y)
X = df.drop(['PatientID', 'Diagnosis', "DoctorInCharge"], axis=1)
y = df['Diagnosis']
print("\nFeatures and target variable defined.")
```

```
Features and target variable defined.
```

STEP4: Split data of a portion of one-fourth as validation data and remaining of training data. This is a manually tunned relatively optimal proportion for this data set.

In [19]:
```
# split X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, rand
# check the shape of X_train and X_test
X_train.shape, X_test.shape
```

Out[19]:
```
((1128, 32), (376, 32))
```

STEP5: Feature Engineering Notice there are no missing values for all variables, so we don't need to engineering missing values here. Since we don't see any skewed distribution for our numerical variables, we don't deal with outliers. We don't need to encode categorical variables

STEP6: Feature scaling

In [20]:
```
X_train.describe()
cols = X_train.columns
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_train = pd.DataFrame(X_train, columns=[cols])
X_test = pd.DataFrame(X_test, columns=[cols])
X_train.describe()
```

Out[20]:

	Age	Gender	Ethnicity	EducationLevel	BMI	Sm
count	1128.000000	1128.000000	1128.000000	1128.000000	1128.000000	1128.0
mean	0.492258	0.505319	0.240248	0.430851	0.500901	0.2
std	0.297137	0.500193	0.336283	0.301735	0.284645	0.4
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	0.233333	0.000000	0.000000	0.333333	0.254608	0.0
50%	0.500000	1.000000	0.000000	0.333333	0.511034	0.0
75%	0.733333	1.000000	0.333333	0.666667	0.741618	1.0
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.0

8 rows × 32 columns

STEP7: Model training, with hyperparameter tuning of total of 32000 models.

In [ ]:
```
from joblib import Parallel, delayed
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
from itertools import product

# Define the parameter grid
param_grid = {
    'n_estimators': list(range(1, 400, 5)),
    'max_depth': list(range(1, 16, 2)),
    'learning_rate': [0.1, 0.05],  # Added learning rate parameter
```

```
    'subsample': [0.7, 0.75, 0.8, 0.85, 0.9],           # Similar to max_feature
    'colsample_bytree': [0.7, 0.75, 0.8, 0.85, 0.9]     # Similar to max_feature
}

# Function to train and evaluate a model with given parameters
def train_evaluate(params):
    n_estimators, max_depth, learning_rate, subsample, colsample_bytree = params
    xgb = XGBClassifier(
        n_estimators=n_estimators,
        max_depth=max_depth,
        learning_rate=learning_rate,
        subsample=subsample,
        colsample_bytree=colsample_bytree,
        use_label_encoder=False,
        random_state=0,
        eval_metric="logloss"  # Suppresses warnings
    )
    # Fit the model on the training data
    xgb.fit(X_train, y_train)
    # Predict on the test set
    y_pred = xgb.predict(X_test)
    # Calculate accuracy score on the test set
    acc_score = accuracy_score(y_test, y_pred)
    return (acc_score, params, xgb)

# Create a list of all parameter combinations
param_combinations = list(product(
    param_grid['n_estimators'],
    param_grid['max_depth'],
    param_grid['learning_rate'],
    param_grid['subsample'],
    param_grid['colsample_bytree']
))

print('number of tasks is: ' + str(len(param_combinations)))

# Use Parallel to evaluate all combinations
results = Parallel(n_jobs=4, verbose=10)(
    delayed(train_evaluate)(params) for params in param_combinations
)

# Find the best model based on accuracy score
best_acc_score, best_params, best_model = max(results, key=lambda x: x[0])

# Output the results
print("Best Parameters:", best_params)
print("Best Accuracy Score on Test Set:", best_acc_score)
```

```
number of tasks is: 32000
```

```
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done   5 tasks      | elapsed:    3.4s
[Parallel(n_jobs=4)]: Done  10 tasks      | elapsed:    3.4s
[Parallel(n_jobs=4)]: Done  17 tasks      | elapsed:    3.5s
[Parallel(n_jobs=4)]: Done  24 tasks      | elapsed:    3.6s
[Parallel(n_jobs=4)]: Batch computation too fast (0.1908912459486539s.) Setting b
atch_size=2.
[Parallel(n_jobs=4)]: Done  34 tasks      | elapsed:    3.9s
[Parallel(n_jobs=4)]: Done  52 tasks      | elapsed:    4.2s
[Parallel(n_jobs=4)]: Batch computation too fast (0.19596543849312903s.) Setting
batch_size=4.
[Parallel(n_jobs=4)]: Done  74 tasks      | elapsed:    4.5s
[Parallel(n_jobs=4)]: Done 104 tasks      | elapsed:    4.8s
[Parallel(n_jobs=4)]: Done 156 tasks      | elapsed:    5.4s
[Parallel(n_jobs=4)]: Done 208 tasks      | elapsed:    5.9s
[Parallel(n_jobs=4)]: Done 268 tasks      | elapsed:    6.7s
[Parallel(n_jobs=4)]: Done 328 tasks      | elapsed:    7.5s
[Parallel(n_jobs=4)]: Done 396 tasks      | elapsed:    8.9s
[Parallel(n_jobs=4)]: Done 464 tasks      | elapsed:    9.8s
[Parallel(n_jobs=4)]: Done 540 tasks      | elapsed:   10.9s
[Parallel(n_jobs=4)]: Done 616 tasks      | elapsed:   11.9s
[Parallel(n_jobs=4)]: Done 700 tasks      | elapsed:   13.6s
[Parallel(n_jobs=4)]: Done 784 tasks      | elapsed:   15.2s
[Parallel(n_jobs=4)]: Done 876 tasks      | elapsed:   16.2s
[Parallel(n_jobs=4)]: Done 968 tasks      | elapsed:   17.6s
[Parallel(n_jobs=4)]: Done 1068 tasks      | elapsed:   19.6s
[Parallel(n_jobs=4)]: Done 1168 tasks      | elapsed:   22.3s
[Parallel(n_jobs=4)]: Done 1276 tasks      | elapsed:   23.9s
[Parallel(n_jobs=4)]: Done 1384 tasks      | elapsed:   25.8s
[Parallel(n_jobs=4)]: Done 1500 tasks      | elapsed:   29.0s
[Parallel(n_jobs=4)]: Done 1616 tasks      | elapsed:   32.6s
[Parallel(n_jobs=4)]: Done 1740 tasks      | elapsed:   34.4s
[Parallel(n_jobs=4)]: Done 1864 tasks      | elapsed:   37.7s
[Parallel(n_jobs=4)]: Done 1996 tasks      | elapsed:   42.8s
[Parallel(n_jobs=4)]: Done 2128 tasks      | elapsed:   44.9s
[Parallel(n_jobs=4)]: Done 2268 tasks      | elapsed:   49.2s
[Parallel(n_jobs=4)]: Done 2408 tasks      | elapsed:   54.3s
[Parallel(n_jobs=4)]: Done 2556 tasks      | elapsed:   56.6s
[Parallel(n_jobs=4)]: Done 2704 tasks      | elapsed:  1.0min
[Parallel(n_jobs=4)]: Done 2860 tasks      | elapsed:  1.1min
[Parallel(n_jobs=4)]: Done 3016 tasks      | elapsed:  1.2min
[Parallel(n_jobs=4)]: Done 3180 tasks      | elapsed:  1.3min
[Parallel(n_jobs=4)]: Done 3344 tasks      | elapsed:  1.3min
[Parallel(n_jobs=4)]: Done 3516 tasks      | elapsed:  1.5min
[Parallel(n_jobs=4)]: Done 3688 tasks      | elapsed:  1.6min
[Parallel(n_jobs=4)]: Done 3868 tasks      | elapsed:  1.7min
[Parallel(n_jobs=4)]: Done 4048 tasks      | elapsed:  1.8min
[Parallel(n_jobs=4)]: Done 4236 tasks      | elapsed:  1.9min
[Parallel(n_jobs=4)]: Done 4424 tasks      | elapsed:  2.0min
[Parallel(n_jobs=4)]: Done 4620 tasks      | elapsed:  2.1min
[Parallel(n_jobs=4)]: Done 4816 tasks      | elapsed:  2.3min
[Parallel(n_jobs=4)]: Done 5020 tasks      | elapsed:  2.4min
[Parallel(n_jobs=4)]: Batch computation too slow (2.038421491947693s.) Setting ba
tch_size=1.
[Parallel(n_jobs=4)]: Done 5197 tasks      | elapsed:  2.6min
[Parallel(n_jobs=4)]: Batch computation too fast (0.18956323768553476s.) Setting
batch_size=2.
[Parallel(n_jobs=4)]: Batch computation too fast (0.1887986660003662s.) Setting b
atch_size=4.
[Parallel(n_jobs=4)]: Done 5324 tasks      | elapsed:  2.6min
```

```
[Parallel(n_jobs=4)]: Done 5536 tasks      | elapsed:  2.8min
[Parallel(n_jobs=4)]: Done 5756 tasks      | elapsed:  2.9min
[Parallel(n_jobs=4)]: Batch computation too slow (2.0008755324233243s.) Setting b
atch_size=1.
[Parallel(n_jobs=4)]: Done 5952 tasks      | elapsed:  3.1min
[Parallel(n_jobs=4)]: Done 6009 tasks      | elapsed:  3.2min
[Parallel(n_jobs=4)]: Batch computation too fast (0.19490258954291195s.) Setting
batch_size=2.
[Parallel(n_jobs=4)]: Done 6108 tasks      | elapsed:  3.2min
[Parallel(n_jobs=4)]: Done 6226 tasks      | elapsed:  3.3min
[Parallel(n_jobs=4)]: Done 6344 tasks      | elapsed:  3.4min
[Parallel(n_jobs=4)]: Done 6466 tasks      | elapsed:  3.5min
[Parallel(n_jobs=4)]: Done 6588 tasks      | elapsed:  3.6min
[Parallel(n_jobs=4)]: Done 6714 tasks      | elapsed:  3.7min
[Parallel(n_jobs=4)]: Done 6840 tasks      | elapsed:  3.9min
[Parallel(n_jobs=4)]: Done 6970 tasks      | elapsed:  3.9min
[Parallel(n_jobs=4)]: Done 7100 tasks      | elapsed:  4.1min
[Parallel(n_jobs=4)]: Done 7234 tasks      | elapsed:  4.2min
[Parallel(n_jobs=4)]: Done 7368 tasks      | elapsed:  4.3min
[Parallel(n_jobs=4)]: Done 7506 tasks      | elapsed:  4.5min
[Parallel(n_jobs=4)]: Done 7644 tasks      | elapsed:  4.6min
[Parallel(n_jobs=4)]: Done 7786 tasks      | elapsed:  4.7min
[Parallel(n_jobs=4)]: Done 7928 tasks      | elapsed:  4.9min
[Parallel(n_jobs=4)]: Done 8074 tasks      | elapsed:  5.0min
[Parallel(n_jobs=4)]: Done 8220 tasks      | elapsed:  5.1min
[Parallel(n_jobs=4)]: Done 8370 tasks      | elapsed:  5.3min
[Parallel(n_jobs=4)]: Done 8520 tasks      | elapsed:  5.4min
[Parallel(n_jobs=4)]: Done 8674 tasks      | elapsed:  5.6min
[Parallel(n_jobs=4)]: Done 8828 tasks      | elapsed:  5.8min
[Parallel(n_jobs=4)]: Done 8986 tasks      | elapsed:  5.9min
[Parallel(n_jobs=4)]: Done 9144 tasks      | elapsed:  6.1min
[Parallel(n_jobs=4)]: Done 9306 tasks      | elapsed:  6.3min
[Parallel(n_jobs=4)]: Done 9468 tasks      | elapsed:  6.4min
[Parallel(n_jobs=4)]: Done 9634 tasks      | elapsed:  6.6min
[Parallel(n_jobs=4)]: Done 9800 tasks      | elapsed:  6.8min
[Parallel(n_jobs=4)]: Done 9970 tasks      | elapsed:  7.0min
[Parallel(n_jobs=4)]: Done 10140 tasks      | elapsed:  7.1min
[Parallel(n_jobs=4)]: Done 10314 tasks      | elapsed:  7.3min
[Parallel(n_jobs=4)]: Done 10488 tasks      | elapsed:  7.5min
[Parallel(n_jobs=4)]: Done 10666 tasks      | elapsed:  7.7min
[Parallel(n_jobs=4)]: Done 10844 tasks      | elapsed:  7.9min
[Parallel(n_jobs=4)]: Done 11026 tasks      | elapsed:  8.1min
[Parallel(n_jobs=4)]: Done 11208 tasks      | elapsed:  8.4min
[Parallel(n_jobs=4)]: Done 11394 tasks      | elapsed:  8.5min
[Parallel(n_jobs=4)]: Done 11580 tasks      | elapsed:  8.8min
[Parallel(n_jobs=4)]: Done 11770 tasks      | elapsed:  8.9min
[Parallel(n_jobs=4)]: Done 11960 tasks      | elapsed:  9.2min
[Parallel(n_jobs=4)]: Done 12154 tasks      | elapsed:  9.4min
[Parallel(n_jobs=4)]: Done 12348 tasks      | elapsed:  9.7min
[Parallel(n_jobs=4)]: Done 12546 tasks      | elapsed:  9.9min
[Parallel(n_jobs=4)]: Done 12744 tasks      | elapsed: 10.1min
[Parallel(n_jobs=4)]: Done 12946 tasks      | elapsed: 10.3min
[Parallel(n_jobs=4)]: Done 13148 tasks      | elapsed: 10.6min
[Parallel(n_jobs=4)]: Done 13354 tasks      | elapsed: 10.8min
[Parallel(n_jobs=4)]: Done 13560 tasks      | elapsed: 11.2min
[Parallel(n_jobs=4)]: Done 13770 tasks      | elapsed: 11.4min
[Parallel(n_jobs=4)]: Done 13980 tasks      | elapsed: 11.7min
[Parallel(n_jobs=4)]: Done 14194 tasks      | elapsed: 11.9min
[Parallel(n_jobs=4)]: Done 14408 tasks      | elapsed: 12.3min
[Parallel(n_jobs=4)]: Done 14626 tasks      | elapsed: 12.5min
```

```
[Parallel(n_jobs=4)]: Done 14844 tasks      | elapsed: 12.8min
[Parallel(n_jobs=4)]: Done 15066 tasks      | elapsed: 13.1min
[Parallel(n_jobs=4)]: Done 15288 tasks      | elapsed: 13.4min
[Parallel(n_jobs=4)]: Done 15514 tasks      | elapsed: 13.7min
[Parallel(n_jobs=4)]: Done 15740 tasks      | elapsed: 13.9min
[Parallel(n_jobs=4)]: Done 15970 tasks      | elapsed: 14.3min
[Parallel(n_jobs=4)]: Done 16200 tasks      | elapsed: 14.6min
[Parallel(n_jobs=4)]: Batch computation too slow (2.0060205765539103s.) Setting b
atch_size=1.
[Parallel(n_jobs=4)]: Done 16397 tasks      | elapsed: 14.9min
[Parallel(n_jobs=4)]: Done 16514 tasks      | elapsed: 15.0min
[Parallel(n_jobs=4)]: Done 16633 tasks      | elapsed: 15.2min
[Parallel(n_jobs=4)]: Done 16752 tasks      | elapsed: 15.4min
[Parallel(n_jobs=4)]: Done 16873 tasks      | elapsed: 15.6min
[Parallel(n_jobs=4)]: Done 16994 tasks      | elapsed: 15.7min
[Parallel(n_jobs=4)]: Done 17117 tasks      | elapsed: 16.0min
[Parallel(n_jobs=4)]: Done 17240 tasks      | elapsed: 16.2min
[Parallel(n_jobs=4)]: Done 17365 tasks      | elapsed: 16.3min
[Parallel(n_jobs=4)]: Done 17490 tasks      | elapsed: 16.5min
[Parallel(n_jobs=4)]: Done 17617 tasks      | elapsed: 16.8min
[Parallel(n_jobs=4)]: Done 17744 tasks      | elapsed: 16.9min
[Parallel(n_jobs=4)]: Done 17873 tasks      | elapsed: 17.1min
[Parallel(n_jobs=4)]: Done 18002 tasks      | elapsed: 17.4min
[Parallel(n_jobs=4)]: Done 18133 tasks      | elapsed: 17.5min
[Parallel(n_jobs=4)]: Done 18264 tasks      | elapsed: 17.7min
[Parallel(n_jobs=4)]: Done 18397 tasks      | elapsed: 18.0min
[Parallel(n_jobs=4)]: Done 18530 tasks      | elapsed: 18.2min
[Parallel(n_jobs=4)]: Done 18665 tasks      | elapsed: 18.4min
[Parallel(n_jobs=4)]: Done 18800 tasks      | elapsed: 18.7min
[Parallel(n_jobs=4)]: Done 18937 tasks      | elapsed: 18.8min
[Parallel(n_jobs=4)]: Done 19074 tasks      | elapsed: 19.1min
[Parallel(n_jobs=4)]: Done 19213 tasks      | elapsed: 19.4min
[Parallel(n_jobs=4)]: Done 19352 tasks      | elapsed: 19.5min
[Parallel(n_jobs=4)]: Done 19493 tasks      | elapsed: 19.8min
[Parallel(n_jobs=4)]: Done 19634 tasks      | elapsed: 20.1min
[Parallel(n_jobs=4)]: Done 19777 tasks      | elapsed: 20.2min
[Parallel(n_jobs=4)]: Done 19920 tasks      | elapsed: 20.5min
[Parallel(n_jobs=4)]: Done 20065 tasks      | elapsed: 20.8min
[Parallel(n_jobs=4)]: Done 20210 tasks      | elapsed: 21.0min
[Parallel(n_jobs=4)]: Done 20357 tasks      | elapsed: 21.3min
[Parallel(n_jobs=4)]: Done 20504 tasks      | elapsed: 21.5min
[Parallel(n_jobs=4)]: Done 20653 tasks      | elapsed: 21.8min
[Parallel(n_jobs=4)]: Done 20802 tasks      | elapsed: 22.1min
[Parallel(n_jobs=4)]: Done 20953 tasks      | elapsed: 22.2min
[Parallel(n_jobs=4)]: Done 21104 tasks      | elapsed: 22.6min
[Parallel(n_jobs=4)]: Done 21257 tasks      | elapsed: 22.8min
[Parallel(n_jobs=4)]: Done 21410 tasks      | elapsed: 23.1min
[Parallel(n_jobs=4)]: Done 21565 tasks      | elapsed: 23.4min
[Parallel(n_jobs=4)]: Done 21720 tasks      | elapsed: 23.6min
[Parallel(n_jobs=4)]: Done 21877 tasks      | elapsed: 23.9min
[Parallel(n_jobs=4)]: Done 22034 tasks      | elapsed: 24.3min
[Parallel(n_jobs=4)]: Done 22193 tasks      | elapsed: 24.6min
[Parallel(n_jobs=4)]: Done 22352 tasks      | elapsed: 25.0min
[Parallel(n_jobs=4)]: Done 22513 tasks      | elapsed: 25.2min
[Parallel(n_jobs=4)]: Done 22674 tasks      | elapsed: 25.5min
[Parallel(n_jobs=4)]: Done 22837 tasks      | elapsed: 26.0min
[Parallel(n_jobs=4)]: Done 23000 tasks      | elapsed: 26.3min
[Parallel(n_jobs=4)]: Done 23165 tasks      | elapsed: 26.8min
[Parallel(n_jobs=4)]: Done 23330 tasks      | elapsed: 27.2min
[Parallel(n_jobs=4)]: Done 23497 tasks      | elapsed: 27.6min
```

```
[Parallel(n_jobs=4)]: Done 23664 tasks      | elapsed: 28.1min
[Parallel(n_jobs=4)]: Done 23833 tasks      | elapsed: 28.5min
[Parallel(n_jobs=4)]: Done 24002 tasks      | elapsed: 29.0min
[Parallel(n_jobs=4)]: Done 24173 tasks      | elapsed: 29.4min
[Parallel(n_jobs=4)]: Done 24344 tasks      | elapsed: 30.0min
[Parallel(n_jobs=4)]: Done 24517 tasks      | elapsed: 30.3min
[Parallel(n_jobs=4)]: Done 24690 tasks      | elapsed: 30.8min
[Parallel(n_jobs=4)]: Done 24865 tasks      | elapsed: 31.2min
[Parallel(n_jobs=4)]: Done 25040 tasks      | elapsed: 31.6min
[Parallel(n_jobs=4)]: Done 25217 tasks      | elapsed: 32.1min
[Parallel(n_jobs=4)]: Done 25394 tasks      | elapsed: 32.4min
[Parallel(n_jobs=4)]: Done 25573 tasks      | elapsed: 32.9min
[Parallel(n_jobs=4)]: Done 25752 tasks      | elapsed: 33.2min
[Parallel(n_jobs=4)]: Done 25933 tasks      | elapsed: 33.7min
[Parallel(n_jobs=4)]: Done 26114 tasks      | elapsed: 34.1min
[Parallel(n_jobs=4)]: Done 26297 tasks      | elapsed: 34.6min
[Parallel(n_jobs=4)]: Done 26480 tasks      | elapsed: 35.0min
[Parallel(n_jobs=4)]: Done 26665 tasks      | elapsed: 35.5min
[Parallel(n_jobs=4)]: Done 26850 tasks      | elapsed: 35.9min
[Parallel(n_jobs=4)]: Done 27037 tasks      | elapsed: 36.4min
[Parallel(n_jobs=4)]: Done 27224 tasks      | elapsed: 36.9min
[Parallel(n_jobs=4)]: Done 27413 tasks      | elapsed: 37.3min
[Parallel(n_jobs=4)]: Done 27602 tasks      | elapsed: 37.8min
[Parallel(n_jobs=4)]: Done 27793 tasks      | elapsed: 38.2min
[Parallel(n_jobs=4)]: Done 27984 tasks      | elapsed: 38.7min
[Parallel(n_jobs=4)]: Done 28177 tasks      | elapsed: 39.1min
[Parallel(n_jobs=4)]: Done 28370 tasks      | elapsed: 39.7min
[Parallel(n_jobs=4)]: Done 28565 tasks      | elapsed: 40.1min
[Parallel(n_jobs=4)]: Done 28760 tasks      | elapsed: 40.7min
[Parallel(n_jobs=4)]: Done 28957 tasks      | elapsed: 41.1min
[Parallel(n_jobs=4)]: Done 29154 tasks      | elapsed: 41.7min
[Parallel(n_jobs=4)]: Done 29353 tasks      | elapsed: 42.1min
[Parallel(n_jobs=4)]: Done 29552 tasks      | elapsed: 42.8min
[Parallel(n_jobs=4)]: Done 29753 tasks      | elapsed: 43.2min
[Parallel(n_jobs=4)]: Done 29954 tasks      | elapsed: 43.9min
[Parallel(n_jobs=4)]: Done 30157 tasks      | elapsed: 44.4min
[Parallel(n_jobs=4)]: Done 30360 tasks      | elapsed: 45.3min
[Parallel(n_jobs=4)]: Done 30565 tasks      | elapsed: 46.3min
[Parallel(n_jobs=4)]: Done 30770 tasks      | elapsed: 47.0min
[Parallel(n_jobs=4)]: Done 30977 tasks      | elapsed: 47.4min
[Parallel(n_jobs=4)]: Done 31184 tasks      | elapsed: 48.1min
[Parallel(n_jobs=4)]: Done 31393 tasks      | elapsed: 48.5min
[Parallel(n_jobs=4)]: Done 31602 tasks      | elapsed: 49.1min
[Parallel(n_jobs=4)]: Done 31813 tasks      | elapsed: 49.4min
[Parallel(n_jobs=4)]: Done 32000 out of 32000 | elapsed: 50.0min finished
Best Parameters: (36, 3, 0.1, 0.9, 0.85)
Best Accuracy Score on Test Set: 0.9654255319148937
```

In [ ]:

```
final_model = best_model
y_pred = final_model.predict(X_test)
accuracy_score(y_test, y_pred)
```

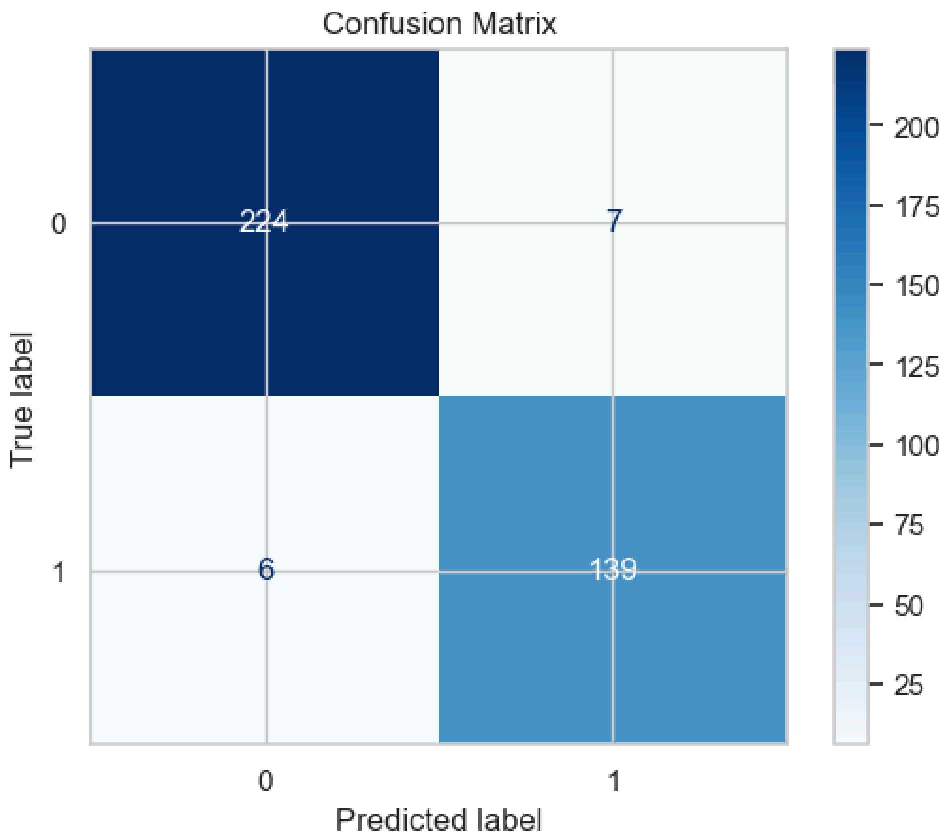
Out[ ]: 0.9654255319148937

Computing confusion matrix

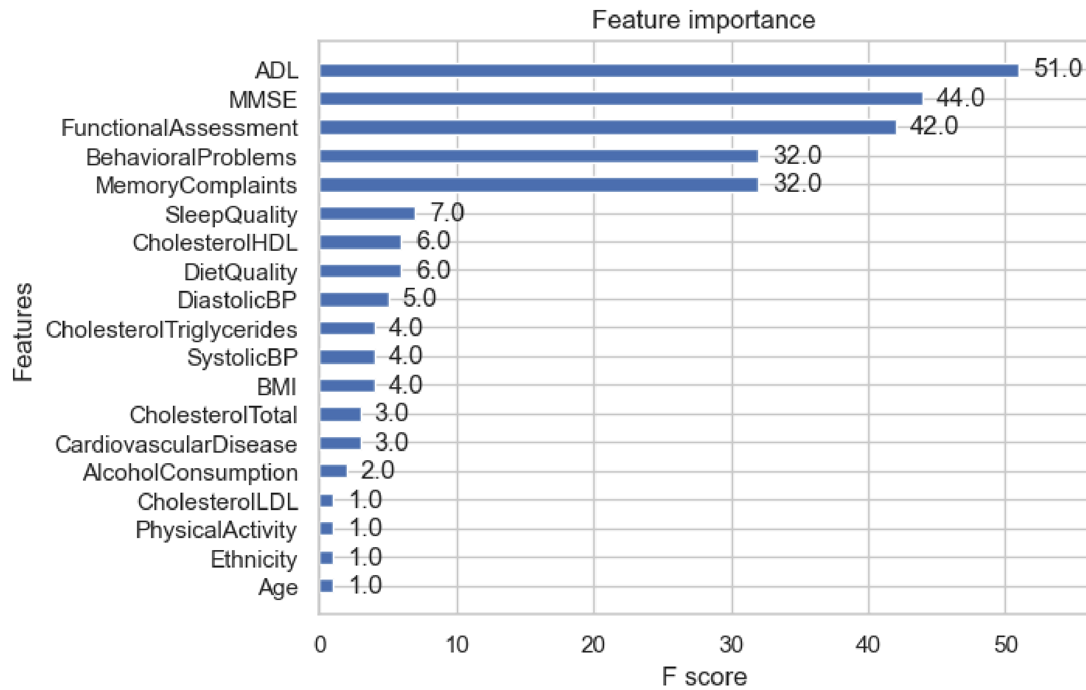
In [26]:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

```
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Display confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=final_model.cl
disp.plot(cmap='Blues')
plt.title("Confusion Matrix")
plt.show()
```



In [31]:
```
from xgboost import plot_importance
import matplotlib.pyplot as plt

# Plot feature importance with all features
plot_importance(final_model, max_num_features=32, importance_type='weight', heig
plt.show()
```



Compute all metrics and ROC, AUC

In [33]:

```
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_

# Compute confusion matrix components
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

# Calculate metrics
accuracy = (tp + tn) / (tp + tn + fp + fn)
precision = tp / (tp + fp)
recall = tp / (tp + fn)
fpr = fp / (fp + tn)  # False Positive Rate
fnr = fn / (fn + tp)  # False Negative Rate
f1 = 2 * (precision * recall) / (precision + recall)

# Calculate AUC
y_pred_prob = final_model.predict_proba(X_test)[:, 1]
auc = roc_auc_score(y_test, y_pred_prob)

print(f"Accuracy & {accuracy:.4f}")
print(f"Precision & {precision:.4f}")
print(f"False Positive Rate (FPR) & {fpr:.4f}")
print(f"False Negative Rate (FNR) & {fnr:.4f}")
print(f"ROC_AUC Score & {auc:.4f}")
```

```
Accuracy & 0.9654
Precision & 0.9521
False Positive Rate (FPR) & 0.0303
False Negative Rate (FNR) & 0.0414
ROC_AUC Score & 0.9652
```

In [25]:

```
# Load and preprocess the test data
test_data = pd.read_csv('test.csv')
X_test_final = test_data.drop(['PatientID', 'DoctorInCharge'], axis=1)
X_test_final_scaled = scaler.transform(X_test_final)
```

```python
# Make predictions with a custom threshold of 0.35
test_predictions = final_model.predict(X_test_final_scaled)  # Get probabilities


# Prepare and save the submission file
submission = pd.DataFrame({
    'PatientID': test_data['PatientID'],
    'Diagnosis': test_predictions
})

# Save to CSV
submission.to_csv('test_predictions_final_XGB.csv', index=False)
print("\nSubmission file 'test_predictions.csv' created successfully with the fo
print(submission.head())
```

```
Submission file 'test_predictions.csv' created successfully with the following fo
rmat:
   PatientID  Diagnosis
0       1505          1
1       1506          0
2       1507          0
3       1508          0
4       1509          1
```

References

Albert, M. S., DeKosky, S. T., Dickson, D., Dubois, B., Feldman, H. H., Fox, N. C., Gamst, A., Holtzman, D. M., Jagust, W. J., Petersen, R. C., et al. (2011). The diagnosis of mild cognitive impairment due to alzheimer's disease: Recommendations from the national institute on aging-alzheimer's association workgroups on diagnostic guidelines for alzheimer's disease. *Alzheimer's & Dementia*, *7*(3), 270–279.

Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794. https://doi.org/10.1145/2939672.2939785

Dukart, J., Mueller, K., Barthel, H., Villringer, A., Sabri, O., Hameister, D., Frölich, L., & Schroeter, M. L. (2013). Meta-analysis of csf and pet biomarkers for alzheimer's disease suggests that early detection is critical. *Frontiers in Aging Neuroscience*, *5*, 58. https://doi.org/10.3389/fnagi.2013.00058

Kharoua, R. E. (2023). Alzheimer's disease dataset [Accessed: December 8, 2024]. https://www.kaggle.com/datasets/rabieelkharoua/alzheimers-disease-dataset

Lazarova, S., Grigorova, D., Petrova-Antonova, D., & Initiative, F. T. A. D. N. (2023). Detection of alzheimer's disease using logistic regression and clock drawing errors. *Brain Sciences*, *13*(8), 1139. https://doi.org/10.3390/brainsci13081139

Li, M., Liu, H., Li, Y., Wang, Z., Yuan, Y., & Dai, H. (2024). Intelligent diagnosis of alzheimer's disease based on machine learning. https://arxiv.org/abs/2402.08539

Livingston, G., et al. (2020). Dementia prevention, intervention, and care: 2020 report of the lancet commission. *The Lancet*, *396*(10248), 413–446.

Rosselli, M., Uribe, I. V., Ahne, E., & Shihadeh, L. (2022). Culture, ethnicity, and level of education in alzheimer's disease. *Neurotherapeutics: The Journal of the American Society for Experimental NeuroTherapeutics*, *19*(1), 26–54. https://doi.org/10.1007/s13311-022-01193-z

Varoquaux, G., Grisel, O., Pedregosa, F., et al. (2023). Joblib: Running python code in parallel.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*, *5*(2), 241–259.