analysis

August 17, 2024

1 Problem Statement: Predicting Alzheimer's Disease Risk Using a Holistic Health Approach

Alzheimer's disease is a progressive neurological disorder that leads to memory loss, cognitive decline, and ultimately, loss of independence. It is one of the most common forms of dementia, affecting millions of individuals worldwide. As the global population ages, the incidence of Alzheimer's disease is expected to rise, placing an increasing burden on individuals, families, and healthcare systems.

Early detection of Alzheimer's disease can significantly improve the quality of life for patients by allowing for earlier interventions, treatment plans, and lifestyle adjustments. Traditional risk assessment methods often focus on a limited set of factors, such as age and family history. However, Alzheimer's disease is a multifactorial condition influenced by a wide range of health and lifestyle factors.

1.0.1 Objective

The goal of this project is to develop a predictive model that estimates the probability of developing Alzheimer's disease based on a holistic health approach. This approach incorporates a comprehensive set of features, including demographic information, cognitive assessments, physical health indicators, lifestyle factors, and other relevant metrics. By leveraging a wide range of data, the model aims to provide a more accurate and personalized risk assessment for Alzheimer's disease.

1.0.2 Approach

To achieve this, I will: 1. **Collect and Preprocess Data**: Utilize a dataset that includes a variety of features related to cognitive function, physical health, lifestyle habits, and demographic information. The data will be preprocessed to ensure consistency and completeness.

2. **Feature Engineering**: Create additional features that encapsulate holistic health metrics, such as composite scores for cardiometabolic health and overall health quality.
3. **Model Development**: Train a machine learning model (e.g., LightGBM) to predict the probability of developing Alzheimer's disease. The model will be tuned and validated using cross-validation techniques to ensure robust performance.
4. **Interpretability and Evaluation**: Evaluate the model using metrics like AUC-ROC to assess its ability to distinguish between high and low-risk individuals. Feature importance will be analyzed to understand the contributions of different health factors to the model's predictions.

5. **Deployment and Application**: Develop a user-friendly application (e.g., a Flask-based web app) that allows individuals to input their health data and receive an estimate of their Alzheimer's disease risk. The application will provide a probabilistic output, enabling users to understand their risk level and take proactive steps to manage their health.

1.0.3 Significance

This model aims to enhance Alzheimer's disease risk assessment by incorporating a broad spectrum of health factors into the prediction process. By adopting a holistic approach, the model not only improves prediction accuracy but also encourages a more comprehensive view of health management.

However, it is important to emphasize that this model is not intended to serve as a clinical diagnosis tool. The probability estimates generated by the model are meant to provide an indication of risk, not a definitive medical judgment. Individuals with concerns about Alzheimer's disease should consult healthcare professionals for a thorough evaluation and diagnosis. This model is intended as an educational and informational resource, designed to raise awareness and support proactive health management.

By clearly communicating the intended use of this model, we aim to prevent any potential misunderstanding or misuse. Users should be aware that while the model can provide valuable insights into potential risk factors, it should not replace professional medical advice, diagnosis, or treatment.

1.0.4 Data

This dataset was sourced from the following: El Kharoua, R. (2024). Alzheimer's Disease Dataset. Kaggle. https://www.kaggle.com/dsv/8668279.

[ ]:
```
# import libraries
import joblib
import random
import json


# Linear Algebra
import numpy as np

# Data Manipulation
import pandas as pd  # data processing, CSV file I/O (e.g. pd.read_csv)

# Plotting
import seaborn as sns
import matplotlib.pyplot as plt  # this is used for the plot the graph

# Sklearn classes
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, accuracy_score,␣
 ↪classification_report, roc_curve, auc
```

```
# Data Processing
from sklearn.preprocessing import MinMaxScaler
from sklearn.compose import ColumnTransformer

# Model Imports
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from lightgbm import LGBMClassifier
from xgboost import XGBClassifier
from lazypredict.Supervised import LazyClassifier


from helper import draw_confusion_matrix, get_weights


# Hyperparameter Tuning
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingGridSearchCV

# Data Balancing
from imblearn.over_sampling import SMOTE


%matplotlib inline
%config InlineBackend.figure_format = 'retina'

# Sets random seed for reproducibility
SEED = 42
random.seed(SEED)
```

[27]:
```
# Import data
df = pd.read_csv("datasets/alzheimers_disease_data.csv")
df.head()
```

[27]:

	PatientID	Age	Gender	Ethnicity	EducationLevel	BMI	Smoking	…
0	4751	73	0	0	2	22.93	0	
1	4752	89	0	0	0	26.83	0	
2	4753	73	0	3	1	17.80	0	
3	4754	74	1	0	1	33.80	1	
4	4755	89	0	0	0	20.72	0	

	AlcoholConsumption	PhysicalActivity	DietQuality	…	MemoryComplaints	…
0	13.30	6.33	1.35	…	0	
1	4.54	7.62	0.52	…	0	
2	19.56	7.84	1.83	…	0	
3	12.21	8.43	7.44	…	0	

```
4               18.45              6.31         0.80  …                  0

   BehavioralProblems   ADL  Confusion  Disorientation  PersonalityChanges  \
0                   0  1.73          0               0                   0
1                   0  2.59          0               0                   0
2                   0  7.12          0               1                   0
3                   1  6.48          0               0                   0
4                   0  0.01          0               0                   1

   DifficultyCompletingTasks  Forgetfulness  Diagnosis  DoctorInCharge
0                          1              0          0       XXXConfid
1                          0              1          0       XXXConfid
2                          1              0          0       XXXConfid
3                          0              0          0       XXXConfid
4                          1              0          0       XXXConfid

[5 rows x 35 columns]
```

[28]:
```
# Looking at specific data types
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2149 entries, 0 to 2148
Data columns (total 35 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   PatientID                  2149 non-null   int64
 1   Age                        2149 non-null   int64
 2   Gender                     2149 non-null   int64
 3   Ethnicity                  2149 non-null   int64
 4   EducationLevel             2149 non-null   int64
 5   BMI                        2149 non-null   float64
 6   Smoking                    2149 non-null   int64
 7   AlcoholConsumption         2149 non-null   float64
 8   PhysicalActivity           2149 non-null   float64
 9   DietQuality                2149 non-null   float64
 10  SleepQuality               2149 non-null   float64
 11  FamilyHistoryAlzheimers    2149 non-null   int64
 12  CardiovascularDisease      2149 non-null   int64
 13  Diabetes                   2149 non-null   int64
 14  Depression                 2149 non-null   int64
 15  HeadInjury                 2149 non-null   int64
 16  Hypertension               2149 non-null   int64
 17  SystolicBP                 2149 non-null   int64
 18  DiastolicBP                2149 non-null   int64
 19  CholesterolTotal           2149 non-null   float64
 20  CholesterolLDL             2149 non-null   float64
 21  CholesterolHDL             2149 non-null   float64
```

```
 22  CholesterolTriglycerides   2149 non-null   float64
 23  MMSE                       2149 non-null   float64
 24  FunctionalAssessment       2149 non-null   float64
 25  MemoryComplaints           2149 non-null   int64
 26  BehavioralProblems         2149 non-null   int64
 27  ADL                        2149 non-null   float64
 28  Confusion                  2149 non-null   int64
 29  Disorientation             2149 non-null   int64
 30  PersonalityChanges         2149 non-null   int64
 31  DifficultyCompletingTasks  2149 non-null   int64
 32  Forgetfulness              2149 non-null   int64
 33  Diagnosis                  2149 non-null   int64
 34  DoctorInCharge             2149 non-null   object
dtypes: float64(12), int64(22), object(1)
memory usage: 587.7+ KB
```

Since the relevant categorical variables (gender, ethnicity, education level etc.) are already encoded as integers, we will not have to perform One Hot Encoding on these columns.

[29]:
```
# Checking for null values
df.isna().sum()
```

[29]:
```
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
```

2 Data Cleaning

2.0.1 Dropped columns

- **PatientID** (Doesn't tell us anything useful)
- **DoctorInCharge** (Confidential and doesn't tell us anything useful)

2.0.2 Null Values

- There are luckily no null values so we will not have to perform any imputing

[30]:
```
df.drop(columns=['PatientID', 'DoctorInCharge'], inplace=True)

df.head()
```

[30]:

	Age	Gender	Ethnicity	EducationLevel	BMI	Smoking	AlcoholConsumption	\
0	73	0	0	2	22.93	0	13.30	
1	89	0	0	0	26.83	0	4.54	
2	73	0	3	1	17.80	0	19.56	
3	74	1	0	1	33.80	1	12.21	
4	89	0	0	0	20.72	0	18.45	

	PhysicalActivity	DietQuality	SleepQuality	…	FunctionalAssessment	\
0	6.33	1.35	9.03	…	6.52	
1	7.62	0.52	7.15	…	7.12	
2	7.84	1.83	9.67	…	5.90	
3	8.43	7.44	8.39	…	8.97	
4	6.31	0.80	5.60	…	6.05	

	MemoryComplaints	BehavioralProblems	ADL	Confusion	Disorientation	\
0	0	0	1.73	0	0	
1	0	0	2.59	0	0	
2	0	0	7.12	0	1	
3	0	1	6.48	0	0	
4	0	0	0.01	0	0	

	PersonalityChanges	DifficultyCompletingTasks	Forgetfulness	Diagnosis

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

[5 rows x 33 columns]
```

[31]:
```
# Checking for duplicates
df.duplicated().sum()
```

[31]:
```
0
```

[32]:
```
# To make it easier, I changed 'Diagnosis' to 'target'
df.rename(columns={'Diagnosis': 'target'}, inplace=True)

df.head()
```

[32]:
```
   Age  Gender  Ethnicity  EducationLevel   BMI  Smoking  AlcoholConsumption  \
0   73       0          0               2 22.93        0               13.30
1   89       0          0               0 26.83        0                4.54
2   73       0          3               1 17.80        0               19.56
3   74       1          0               1 33.80        1               12.21
4   89       0          0               0 20.72        0               18.45

   PhysicalActivity  DietQuality  SleepQuality  …  FunctionalAssessment  \
0              6.33         1.35          9.03  …                  6.52
1              7.62         0.52          7.15  …                  7.12
2              7.84         1.83          9.67  …                  5.90
3              8.43         7.44          8.39  …                  8.97
4              6.31         0.80          5.60  …                  6.05

   MemoryComplaints  BehavioralProblems  ADL  Confusion  Disorientation  \
0                 0                   0 1.73          0               0
1                 0                   0 2.59          0               0
2                 0                   0 7.12          0               1
3                 0                   1 6.48          0               0
4                 0                   0 0.01          0               0

   PersonalityChanges  DifficultyCompletingTasks  Forgetfulness  target
0                   0                          1              0       0
1                   0                          0              1       0
2                   0                          1              0       0
3                   0                          0              0       0
4                   1                          1              0       0

[5 rows x 33 columns]
```

3 Feature Engineering

- Health Score
  - This metric is a combination of health measures into one score. Higher scores represent better health.
  - The weights assigned were calculated using Feature Importance from a random forest model, which shows the specific weights assigned to each attribute that is related to health. This is done to ensure each attribute is being accurately assessed on its contributions to overall health score

[33]:
```
# Define column names and get weights
health_columns = ['BMI', 'PhysicalActivity', 'DietQuality', 'SleepQuality',␣
 ↪'Smoking', 'AlcoholConsumption']
health_weights = get_weights(df, health_columns) # get_weights located in␣
 ↪helper.py

cardiometabolic_columns = ['SystolicBP', 'CholesterolTotal', 'CholesterolLDL',␣
 ↪'CholesterolHDL',
                           'CholesterolTriglycerides']
cardiometabolic_weights = get_weights(df, cardiometabolic_columns)

# Calculate HealthScore, CardiometabolicIndex and TotalHealthScore
df['HealthScore'] = sum(df[col] * weight for col, weight in health_weights.
 ↪items())
df['CardiometabolicIndex'] = sum(df[col] * weight for col, weight in␣
 ↪cardiometabolic_weights.items())
df['TotalHealthScore'] = df['HealthScore'] + df['CardiometabolicIndex']

df.head()
```

[33]:

	Age	Gender	Ethnicity	EducationLevel	BMI	Smoking	AlcoholConsumption	…
0	73	0	0	2	22.93	0	13.30	…
1	89	0	0	0	26.83	0	4.54	…
2	73	0	3	1	17.80	0	19.56	…
3	74	1	0	1	33.80	1	12.21	…
4	89	0	0	0	20.72	0	18.45	…

	PhysicalActivity	DietQuality	SleepQuality	…	ADL	Confusion	…
0	6.33	1.35	9.03	…	1.73	0	…
1	7.62	0.52	7.15	…	2.59	0	…
2	7.84	1.83	9.67	…	7.12	0	…
3	8.43	7.44	8.39	…	6.48	0	…
4	6.31	0.80	5.60	…	0.01	0	…

	Disorientation	PersonalityChanges	DifficultyCompletingTasks	…
0	0	0	1	…
1	0	0	0	…

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

   Forgetfulness  target  HealthScore  CardiometabolicIndex  TotalHealthScore
0              0       0        10.34                125.91            136.25
1              1       0         9.14                184.32            193.46
2              0       0        11.05                138.47            149.52
3              0       0        13.76                138.28            152.03
4              0       0        10.09                156.14            166.23

[5 rows x 36 columns]
```

[34]:
```
# dumping these weights into a json file to make the code in app.py more␣
 ↪readable

with open('weights/health_weights.json', 'w') as file:
    json.dump(health_weights, file)

with open('weights/cardiometabolic_weights.json', 'w') as file:
    json.dump(cardiometabolic_weights, file)

print(health_weights, '\n', cardiometabolic_weights)
```

```
{'BMI': 0.19617733134476573, 'PhysicalActivity': 0.19132919301981238,
'DietQuality': 0.19588838342827544, 'SleepQuality': 0.2024896957549555,
'Smoking': 0.0229025814599068, 'AlcoholConsumption': 0.1912128149922842}
 {'SystolicBP': 0.16835963478367078, 'CholesterolTotal': 0.2046592989787797,
'CholesterolLDL': 0.20362470881887026, 'CholesterolHDL': 0.21553761130813062,
'CholesterolTriglycerides': 0.20781874611054874}
```

4 Exploratory Data Analysis

[35]:
```
df.head()
```

[35]:
```
   Age  Gender  Ethnicity  EducationLevel   BMI  Smoking  AlcoholConsumption  \
0   73       0          0               2 22.93        0               13.30
1   89       0          0               0 26.83        0                4.54
2   73       0          3               1 17.80        0               19.56
3   74       1          0               1 33.80        1               12.21
4   89       0          0               0 20.72        0               18.45

   PhysicalActivity  DietQuality  SleepQuality  …   ADL  Confusion  \
0              6.33         1.35          9.03  … 1.73          0
1              7.62         0.52          7.15  … 2.59          0
2              7.84         1.83          9.67  … 7.12          0
3              8.43         7.44          8.39  … 6.48          0
```

```
4              6.31         0.80          5.60  … 0.01          0

   Disorientation  PersonalityChanges  DifficultyCompletingTasks  \
0               0                   0                          1
1               0                   0                          0
2               1                   0                          1
3               0                   0                          0
4               0                   1                          1

   Forgetfulness  target  HealthScore  CardiometabolicIndex  TotalHealthScore
0              0       0        10.34                125.91            136.25
1              1       0         9.14                184.32            193.46
2              0       0        11.05                138.47            149.52
3              0       0        13.76                138.28            152.03
4              0       0        10.09                156.14            166.23

[5 rows x 36 columns]
```

[36]:
```
df.describe().T
```

[36]:

	count	mean	std	min	25%	50%	75%
Age	2149.00	74.91	8.99	60.00	67.00	75.00	83.00
Gender	2149.00	0.51	0.50	0.00	0.00	1.00	1.00
Ethnicity	2149.00	0.70	1.00	0.00	0.00	0.00	1.00
EducationLevel	2149.00	1.29	0.90	0.00	1.00	1.00	2.00
BMI	2149.00	27.66	7.22	15.01	21.61	27.82	33.87
Smoking	2149.00	0.29	0.45	0.00	0.00	0.00	1.00
AlcoholConsumption	2149.00	10.04	5.76	0.00	5.14	9.93	15.16
PhysicalActivity	2149.00	4.92	2.86	0.00	2.57	4.77	7.43
DietQuality	2149.00	4.99	2.91	0.01	2.46	5.08	7.56
SleepQuality	2149.00	7.05	1.76	4.00	5.48	7.12	8.56
FamilyHistoryAlzheimers	2149.00	0.25	0.43	0.00	0.00	0.00	1.00
CardiovascularDisease	2149.00	0.14	0.35	0.00	0.00	0.00	0.00
Diabetes	2149.00	0.15	0.36	0.00	0.00	0.00	0.00
Depression	2149.00	0.20	0.40	0.00	0.00	0.00	0.00
HeadInjury	2149.00	0.09	0.29	0.00	0.00	0.00	0.00
Hypertension	2149.00	0.15	0.36	0.00	0.00	0.00	0.00
SystolicBP	2149.00	134.26	25.95	90.00	112.00	134.00	157.00
DiastolicBP	2149.00	89.85	17.59	60.00	74.00	91.00	105.00
CholesterolTotal	2149.00	225.20	42.54	150.09	190.25	225.09	262.03
CholesterolLDL	2149.00	124.34	43.37	50.23	87.20	123.34	161.73
CholesterolHDL	2149.00	59.46	23.14	20.00	39.10	59.77	78.94
CholesterolTriglycerides	2149.00	228.28	101.99	50.41	137.58	230.30	314.84
MMSE	2149.00	14.76	8.61	0.01	7.17	14.44	22.16
FunctionalAssessment	2149.00	5.08	2.89	0.00	2.57	5.09	7.55
MemoryComplaints	2149.00	0.21	0.41	0.00	0.00	0.00	0.00
BehavioralProblems	2149.00	0.16	0.36	0.00	0.00	0.00	0.00

ADL	2149.00	4.98	2.95	0.00	2.34	5.04	7.58
Confusion	2149.00	0.21	0.40	0.00	0.00	0.00	0.00
Disorientation	2149.00	0.16	0.37	0.00	0.00	0.00	0.00
PersonalityChanges	2149.00	0.15	0.36	0.00	0.00	0.00	0.00
DifficultyCompletingTasks	2149.00	0.16	0.37	0.00	0.00	0.00	0.00
Forgetfulness	2149.00	0.30	0.46	0.00	0.00	0.00	1.00
target	2149.00	0.35	0.48	0.00	0.00	0.00	1.00
HealthScore	2149.00	10.70	2.01	4.98	9.27	10.69	12.17
CardiometabolicIndex	2149.00	154.27	25.32	85.63	135.28	153.81	173.84
TotalHealthScore	2149.00	164.97	25.43	95.60	145.85	164.78	184.07

	max
Age	90.00
Gender	1.00
Ethnicity	3.00
EducationLevel	3.00
BMI	39.99
Smoking	1.00
AlcoholConsumption	19.99
PhysicalActivity	9.99
DietQuality	10.00
SleepQuality	10.00
FamilyHistoryAlzheimers	1.00
CardiovascularDisease	1.00
Diabetes	1.00
Depression	1.00
HeadInjury	1.00
Hypertension	1.00
SystolicBP	179.00
DiastolicBP	119.00
CholesterolTotal	299.99
CholesterolLDL	199.97
CholesterolHDL	99.98
CholesterolTriglycerides	399.94
MMSE	29.99
FunctionalAssessment	10.00
MemoryComplaints	1.00
BehavioralProblems	1.00
ADL	10.00
Confusion	1.00
Disorientation	1.00
PersonalityChanges	1.00
DifficultyCompletingTasks	1.00
Forgetfulness	1.00
target	1.00
HealthScore	16.37
CardiometabolicIndex	220.28

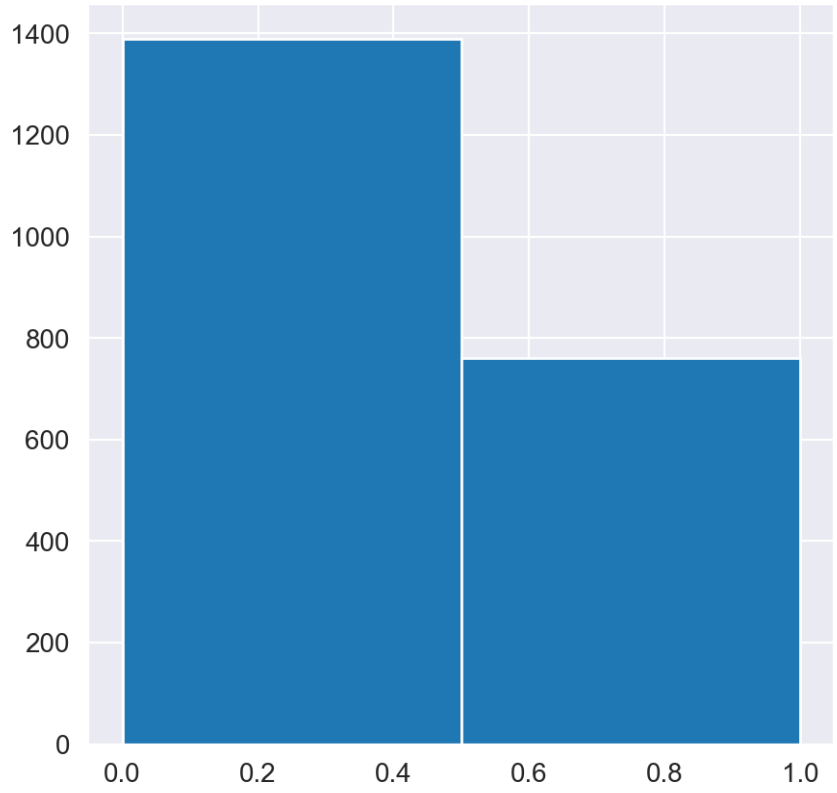
```
TotalHealthScore          231.00
```

Takeaways: - The people in this dataset, on average, are less educated (1.2) - There are more smokers in this dataset (0.71) - Physical activity, diet quality, and sleep quality are generally average, at around 0.5 - Most of the data points in this dataset have no Alzheimer's diagnosis (around 65%)

Looking at balance of the target variable

[37]:
```
# Plotting histogram of 'target'
df['target'].hist(bins=2, figsize=(5,5))

# Getting count of diagnoses of NO and YES
print(f"Not Diagnosed with Alzheimer's Count: {(df['target'] == 0).sum()}␣
 ↪({round((df['target'] == 0).sum() / df.shape[0] * 100, 2)}%)")
print(f"Diagnosed with Alzheimer's Count: {(df['target'] == 1).sum()}␣
 ↪({round((df['target'] == 1).sum() / df.shape[0] * 100, 2)}%)")
plt.show()
```

```
Not Diagnosed with Alzheimer's Count: 1389 (64.63%)
Diagnosed with Alzheimer's Count: 760 (35.37%)
```



Interpretation: 64% of patients in this dataset were not diagnosed, while 35% were diagnosed. This

represents a moderately unbalanced dataset, which might be addressed later

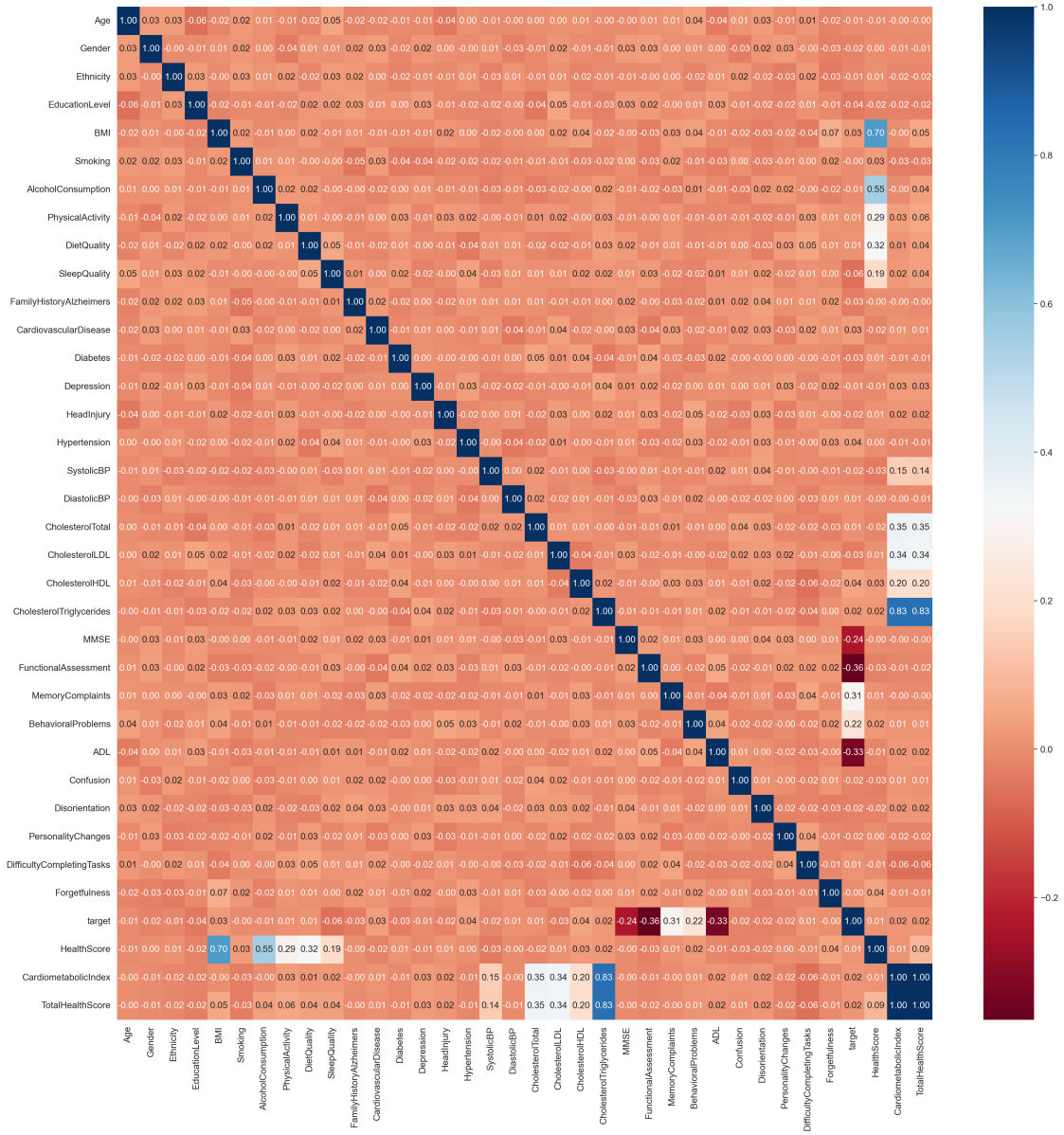
[38]:
```
df.hist(figsize=(20,20))
plt.show()
```

Interpretation: the health score attributes are normally distributed, while other attributes such as BMI are uniformly distributed. In general, most patients in this data set have better health, as can be seen by the high percentage of individuals who responded 'No' (0) for many categorical variables.

Let's take a look at correlations now

[39]:
```
# Creating correlation matrix
correlations = df.corr()
```

```
plt.figure(figsize=(20,20))
sns.heatmap(correlations, annot=True, fmt=".2f", cmap='RdBu')
plt.show()
```



[40]:
```
# looking at correlation with target more closely
correlations['target'].sort_values(ascending=False)
```

[40]:
```
target                  1.00
MemoryComplaints        0.31
```

```
BehavioralProblems           0.22
CholesterolHDL               0.04
Hypertension                 0.04
CardiovascularDisease        0.03
BMI                          0.03
CholesterolTriglycerides     0.02
TotalHealthScore             0.02
CardiometabolicIndex         0.02
DifficultyCompletingTasks    0.01
DietQuality                  0.01
HealthScore                  0.01
CholesterolTotal             0.01
PhysicalActivity             0.01
DiastolicBP                  0.01
Forgetfulness               -0.00
Smoking                     -0.00
Age                         -0.01
Depression                  -0.01
AlcoholConsumption          -0.01
Ethnicity                   -0.01
SystolicBP                  -0.02
Confusion                   -0.02
PersonalityChanges          -0.02
Gender                      -0.02
HeadInjury                  -0.02
Disorientation              -0.02
Diabetes                    -0.03
CholesterolLDL              -0.03
FamilyHistoryAlzheimers     -0.03
EducationLevel              -0.04
SleepQuality                -0.06
MMSE                        -0.24
ADL                         -0.33
FunctionalAssessment        -0.36
Name: target, dtype: float64
```

Interpretation: - **MemoryComplaints** is the most highly correlated feature. It makes sense, as Alzheimer's is associated with a decline in memory, so the more memory complaints, the more likely it is that patient develops Alzheimer's - **BehavioralProblems** is the second most highly correlated feature, which also makes sense as more behavioral problems are associated with Alzheimer's - Interestingly enough, metrics like **TotalHealthScore**, any health metric like **PhysicalActivity**, and **Smoking** or **AlcoholConsumption** do not seem to be correlated. This might be due to the interaction these columns have with **CardiovascularDisease**, which seems to be more correlated, albeit not by much.

5 Data Processing

- Since I am prioritizing tree based models for better interpretability and ease of training, I did not use any scalars in the preprocessing steps. Tree based models are unaffacted by scaling and not scaling the data removes one more step in `main.py` of scaling the real-time data for prediction.

[41]:
```
# Defining target
y = df['target']

# Define features
X = df.drop(columns='target')

# Splitting data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=SEED)
```

[42]:
```
# Applying SMOTE to data to make it more balanced
# Create an instance of SMOTE
smote = SMOTE(random_state=SEED)

# Returns the input data with oversampled minority class according to the␣
 ↪quantity defined in the sampling_strategy parameter
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

6 Model Building

Now it's time to test out some models!

[49]:
```
# Lazy Classifier is great to use for a baseline of the best performing models,␣
 ↪without
# any hyperparameter tuning
# This will be used to assess the models further
clf = LazyClassifier(verbose=0, ignore_warnings=True, custom_metric=None)
models, predictions = clf.fit(X_train_smote, X_test, y_train_smote, y_test)

# Display the results
print(models)
```

```
 97%|      | 28/29 [00:05<00:00,  6.27it/s]
[LightGBM] [Info] Number of positive: 1112, number of negative: 1112
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.072878 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 4064
[LightGBM] [Info] Number of data points in the train set: 2224, number of used
```

```
features: 35
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
100%|      | 29/29 [00:07<00:00,  3.75it/s]
```

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	\
LGBMClassifier	0.94	0.93	0.93	0.94	
XGBClassifier	0.93	0.92	0.92	0.93	
BaggingClassifier	0.92	0.91	0.91	0.92	
RandomForestClassifier	0.89	0.87	0.87	0.89	
DecisionTreeClassifier	0.86	0.87	0.87	0.86	
AdaBoostClassifier	0.86	0.86	0.86	0.86	
LinearDiscriminantAnalysis	0.83	0.80	0.80	0.83	
RidgeClassifierCV	0.81	0.80	0.80	0.81	
CalibratedClassifierCV	0.81	0.80	0.80	0.81	
ExtraTreesClassifier	0.83	0.80	0.80	0.83	
RidgeClassifier	0.80	0.80	0.80	0.80	
LogisticRegression	0.80	0.80	0.80	0.81	
LinearSVC	0.80	0.80	0.80	0.80	
NuSVC	0.81	0.79	0.79	0.81	
SVC	0.81	0.79	0.79	0.81	
SGDClassifier	0.77	0.77	0.77	0.77	
BernoulliNB	0.77	0.75	0.75	0.77	
NearestCentroid	0.75	0.74	0.74	0.75	
PassiveAggressiveClassifier	0.71	0.70	0.70	0.71	
Perceptron	0.71	0.70	0.70	0.72	
GaussianNB	0.71	0.69	0.69	0.71	
KNeighborsClassifier	0.63	0.66	0.66	0.63	
ExtraTreeClassifier	0.66	0.64	0.64	0.66	
LabelSpreading	0.57	0.56	0.56	0.58	
LabelPropagation	0.57	0.56	0.56	0.58	
DummyClassifier	0.64	0.50	0.50	0.50	
QuadraticDiscriminantAnalysis	0.64	0.50	0.50	0.50	

Model	Time Taken
LGBMClassifier	2.21
XGBClassifier	0.26
BaggingClassifier	0.77
RandomForestClassifier	0.83
DecisionTreeClassifier	0.11
AdaBoostClassifier	0.89
LinearDiscriminantAnalysis	0.09
RidgeClassifierCV	0.05
CalibratedClassifierCV	0.15
ExtraTreesClassifier	0.32
RidgeClassifier	0.03
LogisticRegression	0.17

```
LinearSVC                          0.38
NuSVC                              0.33
SVC                                0.28
SGDClassifier                      0.07
BernoulliNB                        0.03
NearestCentroid                    0.02
PassiveAggressiveClassifier        0.02
Perceptron                         0.03
GaussianNB                         0.02
KNeighborsClassifier               0.04
ExtraTreeClassifier                0.03
LabelSpreading                     0.21
LabelPropagation                   0.18
DummyClassifier                    0.03
QuadraticDiscriminantAnalysis      0.04
```

Taking the highest performing models and conducting thorough hyperparameter tuning using **HalvingGridSearchCV**. **HalvingGridSearchCV** is a recent addition to **Scikit-Learn** in version 0.24. It is a way to perform a hyperparameter grid search more efficiently than the traditional **GridSearchCV**.

The key concept of this method is successive halving. It works by training estimators on a subset of the full dataset at first, and only the best half of the estimators (as measured by their score on the validation set) goes on to the next iteration, where they are trained on a larger dataset. This process is repeated until the final round, in which the remaining estimators are trained on the full dataset.

The cv option in **HalvingGridSearchCV**, just like in **GridSearchCV**, is used to determine the cross-validation splitting strategy. Cross-validation is a procedure used to avoid overfitting and estimate the skill of the model on new data. Here are the possible values for the cv parameter: * None, to use the default 5-fold cross-validation, * An integer, to specify the number of folds in a (Stratified)KFold, * A cross-validation generator, * An iterable yielding (train, test) splits as arrays of indices.

For example, for integer/None inputs, if the estimator is a classifier and y (labels) is either binary or multiclass, StratifiedKFold is used. In all other cases, KFold is used.

[50]:
```python
# Random forest classifier

# Define the parameters
params = {
    'bootstrap': [False],
    'criterion': ['entropy'],
    'max_depth': [40, 45, 50],
    'max_features': ['sqrt'],
    'min_samples_leaf': [2, 3, 4],
    'min_samples_split': [2, 3, 4],
    'n_estimators': [450, 500, 550],
```

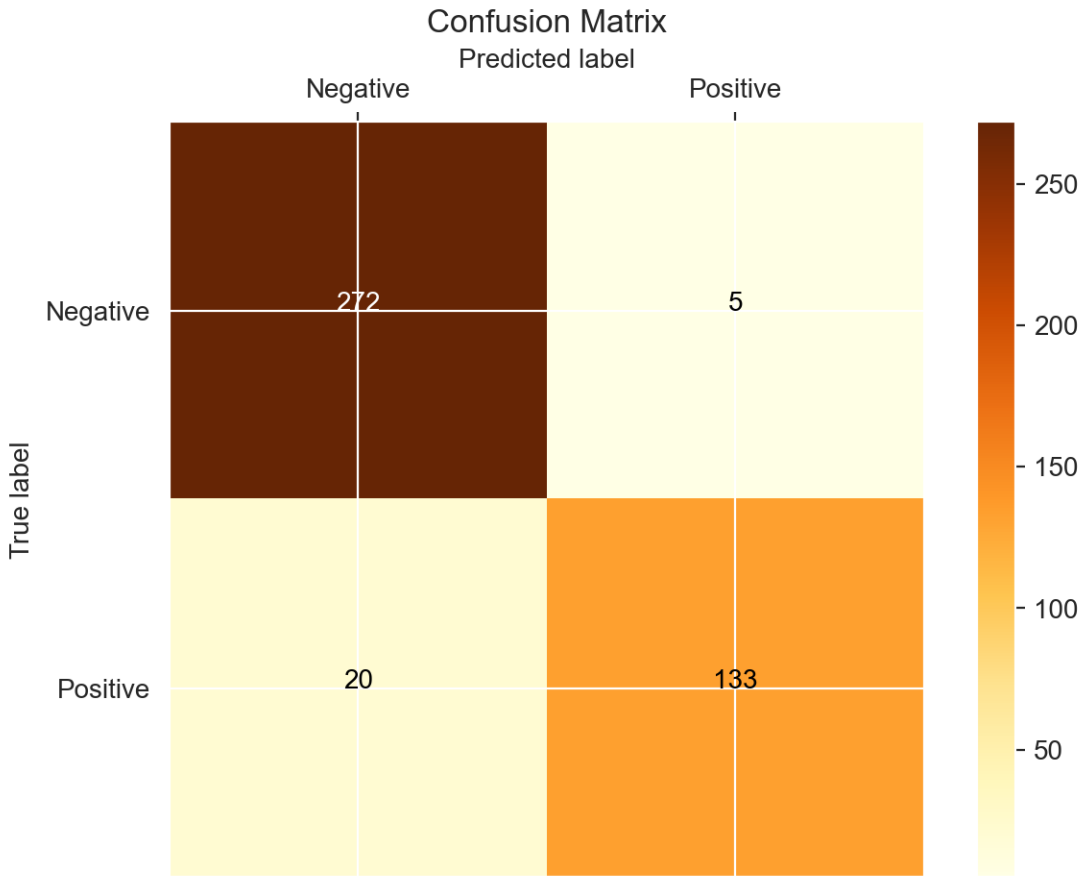
```
    'oob_score': [False, True]
}

rf = RandomForestClassifier()

grid_search_rf = HalvingGridSearchCV(estimator=rf,param_grid=params, cv=10,␣
 ↪n_jobs=-1)

# Run the grid search
grid_search_rf.fit(X_train, y_train)

# Print out the best parameters
print("Best parameters found: ", grid_search_rf.best_params_)
print("Highest accuracy found: ", grid_search_rf.best_score_)
```

```
Best parameters found:  {'bootstrap': False, 'criterion': 'entropy',
'max_depth': 45, 'max_features': 'sqrt', 'min_samples_leaf': 2,
'min_samples_split': 4, 'n_estimators': 450, 'oob_score': False}
Highest accuracy found:  0.9351592246452058
```

[51]:
```
# Testing the best Random Forest model
best_rf_model = grid_search_rf.best_estimator_

# Predict on the testing data
test_predictions_rf = best_rf_model.predict(X_test)

# Get the accuracy of the model
test_accuracy_rf = accuracy_score(y_test, test_predictions_rf)

print(f"Test Accuracy (Random Forest): {test_accuracy_rf * 100:.3f}")

# Check the classification report
print(classification_report(y_test, test_predictions_rf))

# Predict probabilities
probabilities_rf = best_rf_model.predict_proba(X_test)

# Probabilities for positive class
auc_rf = roc_auc_score(y_test, probabilities_rf[:, 1])

print(f"AUC-ROC score for Random Forest is {auc_rf}")

# Confusion Matrix
draw_confusion_matrix(y_test, test_predictions_rf, ['Negative', 'Positive'])
```

```
Test Accuracy (Random Forest): 94.186
              precision    recall  f1-score   support
```

```
              0       0.93      0.98      0.96       277
              1       0.96      0.87      0.91       153

       accuracy                           0.94       430
      macro avg       0.95      0.93      0.94       430
   weighted avg       0.94      0.94      0.94       430

AUC-ROC score for Random Forest is 0.9498006182015526
```



Interpretation: The Random Forest model performed very well on the test data, achieving an accuracy of approximately 94.19%. This denotes that the model correctly classified 94.19% of all instances in the test set.

Taking a closer look at the classification report, we observe an excellent balance between precision and recall. For the class `0`, the model has a precision and recall of 0.93 and 0.98 respectively, achieving an F1-score of 0.96. For the class `1`, the model has a precision and recall of 0.96 and 0.87 respectively, attaining an F1-score of 0.91. To elaborate:

- **Precision** is the proportion of true positive outcomes in the set of all instances that the classifier labelled as positive. Higher the precision, lower the Type I error (false positives).

- **Recall** or Sensitivity is the proportion of true positive outcomes in the set of all instances that are truly positive. Higher the recall, lower the Type II error (false negatives).
- **F1-score** is the harmonic mean of precision and recall. It tries to find the balance between precision and recall.

The macro average (average of the unweighted mean per label) and weighted average (average of the support-weighted mean per label) metrics summarise the overall precision, recall, and F1-score across all classes.

Additionally, the AUC-ROC (Area Under the Receiver Operating Characteristic) score for the model is approximately 0.9498. The AUC-ROC score is a performance measurement for the classification problems at various threshold settings. It tells us how much our model is capable of distinguishing between classes. The higher the AUC, the better our model is at predicting 0s as 0s and 1s as 1s. An excellent model has AUC near to 1, so the model appears to have done a good job.

Overall, considering these results, the Random Forest model has demonstrated strong performance on this dataset.

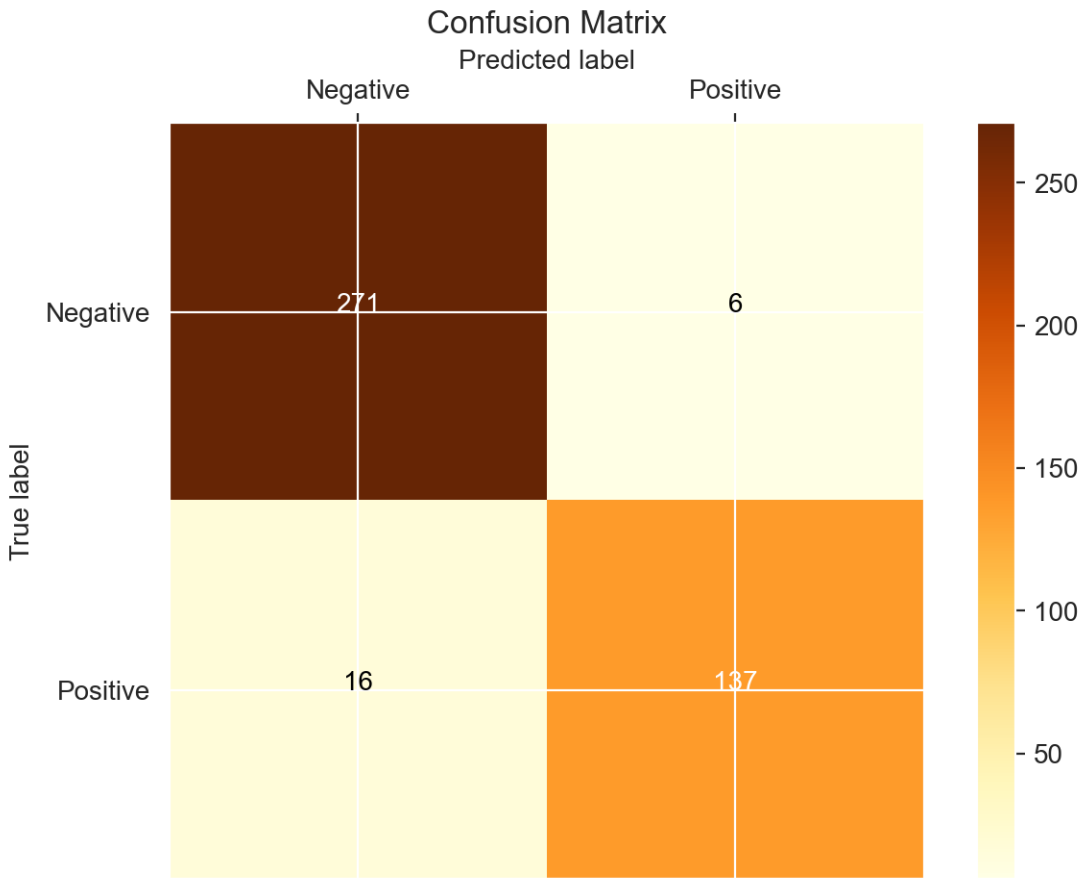
[52]:
```
# making bagging classifier

# make baseline model
base_estimator_1 = DecisionTreeClassifier(random_state=SEED)
# parameters
params = {
    'n_estimators': [100, 200, 300],
    'max_samples': [0.5, 0.6, 0.7, 0.8],
    'max_features': [0.5, 0.6, 0.7, 0.8],
    'bootstrap': [True, False],
    'bootstrap_features': [True, False],
    'estimator': [base_estimator_1],
    'warm_start': [True, False]
}


bag_clf = BaggingClassifier(random_state=SEED)

hgs_bag = HalvingGridSearchCV(bag_clf, params, scoring='accuracy', cv=10,␣
 ↪n_jobs=-1)

# train model
hgs_bag.fit(X_train, y_train)

# print best parameters and score
print("Best parameters found: ", hgs_bag.best_params_)
print("Highest accuracy found: ", hgs_bag.best_score_)
```

```
Best parameters found:  {'bootstrap': True, 'bootstrap_features': False,
'estimator': DecisionTreeClassifier(random_state=42), 'max_features': 0.8,
```

```
'max_samples': 0.8, 'n_estimators': 300, 'warm_start': True}
Highest accuracy found:  0.926773970231914
```

[53]:
```python
# Grab the best model from the Halving Grid Search
best_bagging_model = hgs_bag.best_estimator_

# Use the model to make predictions on the test set
test_predictions_bag = best_bagging_model.predict(X_test)

# Get the accuracy of the model
test_accuracy_bag = accuracy_score(y_test, test_predictions_bag)

print(f"Test Accuracy: {test_accuracy_bag * 100:.3f}")

# Check the classification report
print(classification_report(y_test, test_predictions_bag))

# Probabilities for positive class
probabilities_bag = best_bagging_model.predict_proba(X_test)
auc_bag = roc_auc_score(y_test, probabilities_bag[:, 1])

print(f"AUC-ROC score is {auc_bag}")

# Draw the confusion matrix
draw_confusion_matrix(y_test, test_predictions_bag, ['Negative', 'Positive'])
```

```
Test Accuracy: 94.884
              precision    recall  f1-score   support

           0       0.94      0.98      0.96       277
           1       0.96      0.90      0.93       153

    accuracy                           0.95       430
   macro avg       0.95      0.94      0.94       430
weighted avg       0.95      0.95      0.95       430

AUC-ROC score is 0.951936009060664
```



Interpretation: The model's overall accuracy on the test set is 94.88%, indicating a high level of performance.

For class `0`, precision is 0.94 and recall is 0.98, resulting in an F1-score of 0.96. For class `1`, precision is 0.96 and recall is 0.90, with this category achieving an F1-score of 0.93. The macro and weighted averages for precision, recall, and F1 are all approximately 0.95, showing good balance and performance.

The AUC-ROC score is 0.9519, which is very close to 1, a perfect score. This suggests the model has excellent ability to distinguish between the classes.

[77]:
```
# making LGBMClassifier

# parameters
params = {
    'boosting_type': ['gbdt'],
    'learning_rate': [0.05, 0.1, 0.2],
    'max_depth': [-1, 10, 8],
    'min_child_samples': [10, 20, 30],
    'n_estimators': [150, 200, 250],
```

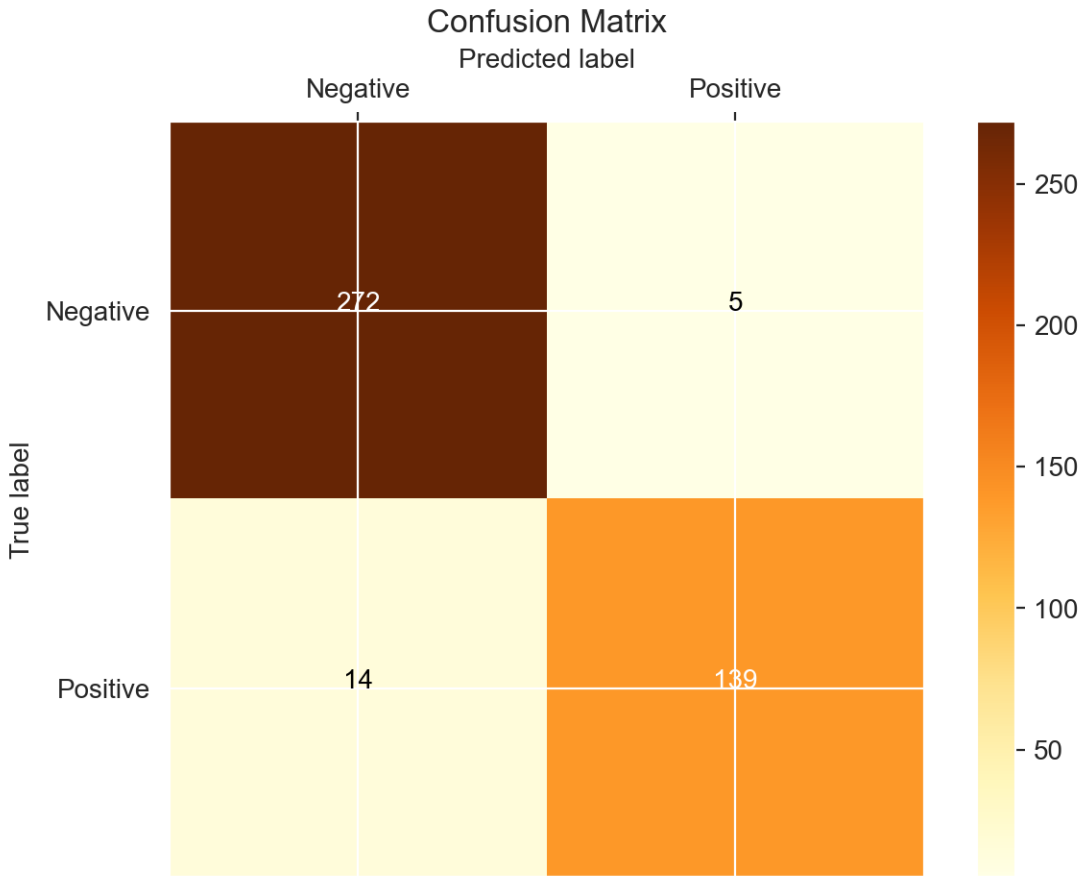
```
    'num_leaves': [40, 50, 60],
    'objective': ['binary'],
    'subsample': [0.6, 0.65, 0.7, 0.75]
}

lgbm = LGBMClassifier(random_state=SEED, verbose=-1)

hgs_lgbm = HalvingGridSearchCV(lgbm, params, scoring='accuracy', cv=10,␣
 ↪n_jobs=-1)

# train model
hgs_lgbm.fit(X_train, y_train)

# print best parameters and score
print("Best parameters found: ", hgs_lgbm.best_params_)
print("Highest accuracy found: ", hgs_lgbm.best_score_)
```

```
Best parameters found:  {'boosting_type': 'gbdt', 'learning_rate': 0.05,
'max_depth': 8, 'min_child_samples': 20, 'n_estimators': 200, 'num_leaves': 60,
'objective': 'binary', 'subsample': 0.6}
Highest accuracy found:  0.9434752509518864
```

[78]:
```
# Grab the best model from the Halving Grid Search
best_lgbm_model = hgs_lgbm.best_estimator_

# Use the model to make predictions on the test set
test_predictions_lgbm = best_lgbm_model.predict(X_test)

# Get the accuracy of the model
test_accuracy_lgbm = accuracy_score(y_test, test_predictions_lgbm)

print(f"Test Accuracy: {test_accuracy_lgbm * 100:.3f}")

# Check the classification report
print(classification_report(y_test, test_predictions_lgbm))

# Probabilities for positive class
probabilities_lgbm = best_lgbm_model.predict_proba(X_test)
auc_lgbm = roc_auc_score(y_test, probabilities_lgbm[:, 1])

print(f"AUC-ROC score is {auc_lgbm}")

# Draw the confusion matrix
draw_confusion_matrix(y_test, test_predictions_lgbm, ['Negative', 'Positive'])
```

```
Test Accuracy: 95.581
              precision    recall  f1-score   support
```

```
              0       0.95      0.98      0.97       277
              1       0.97      0.91      0.94       153

       accuracy                           0.96       430
      macro avg       0.96      0.95      0.95       430
   weighted avg       0.96      0.96      0.96       430

AUC-ROC score is 0.9584011703357638
```



Interpretation: The model's overall test accuracy is 95.81%, indicating an excellent level of performance.

The precision and recall for class `0` are 0.95 and 0.98, with an F1-score of 0.97; while for class `1` they are 0.97 and 0.92, with an F1-score of 0.94. The macro and weighted averages for precision, recall, and F1 are all approximately 0.96, displaying uniformly high performance across metrics.

The AUC-ROC score is 0.9516, very close to 1, suggesting the model has an excellent capability to classify instances correctly.

7 Model selection

[56]:
```
# Define dictionaries with model test results
lgbm_results = {"Model": "LGBM",
                "Test_Accuracy": test_accuracy_lgbm,
                "AUC-ROC_Score": auc_lgbm
                }

bagging_results = {"Model": "Bagging",
                   "Test_Accuracy": test_accuracy_bag,
                   "AUC-ROC_Score": auc_bag
                   }

rf_results = {"Model": "Random Forest",
              "Test_Accuracy": test_accuracy_rf,
              "AUC-ROC_Score": auc_rf
              }

# Create dataframe from results
model_comparison = pd.DataFrame([lgbm_results, bagging_results, rf_results])

# Determine best model
best_model_name = model_comparison.loc[model_comparison['Test_Accuracy'].
 ↪idxmax(), 'Model']

model_names = {"LGBM": best_lgbm_model, "Bagging": best_bagging_model, "Random␣
 ↪Forest": best_rf_model}

# Get the best model
best_model = model_names[best_model_name]

print(model_comparison)
```

```
           Model  Test_Accuracy  AUC-ROC_Score
0           LGBM           0.96           0.95
1        Bagging           0.95           0.95
2  Random Forest           0.94           0.95
```

Interpretation: LGBMClassifier has the highest accuracy and ROC-AUC score, making it the ideal choice for deployment.

[57]:
```
# Saving the best model as a pickle file
with open("models/model.pkl", "wb") as file:
    joblib.dump(best_model, file)
```

[72]:
```
# Taking a look at feature importance

feature_importances = best_model.feature_importances_
```

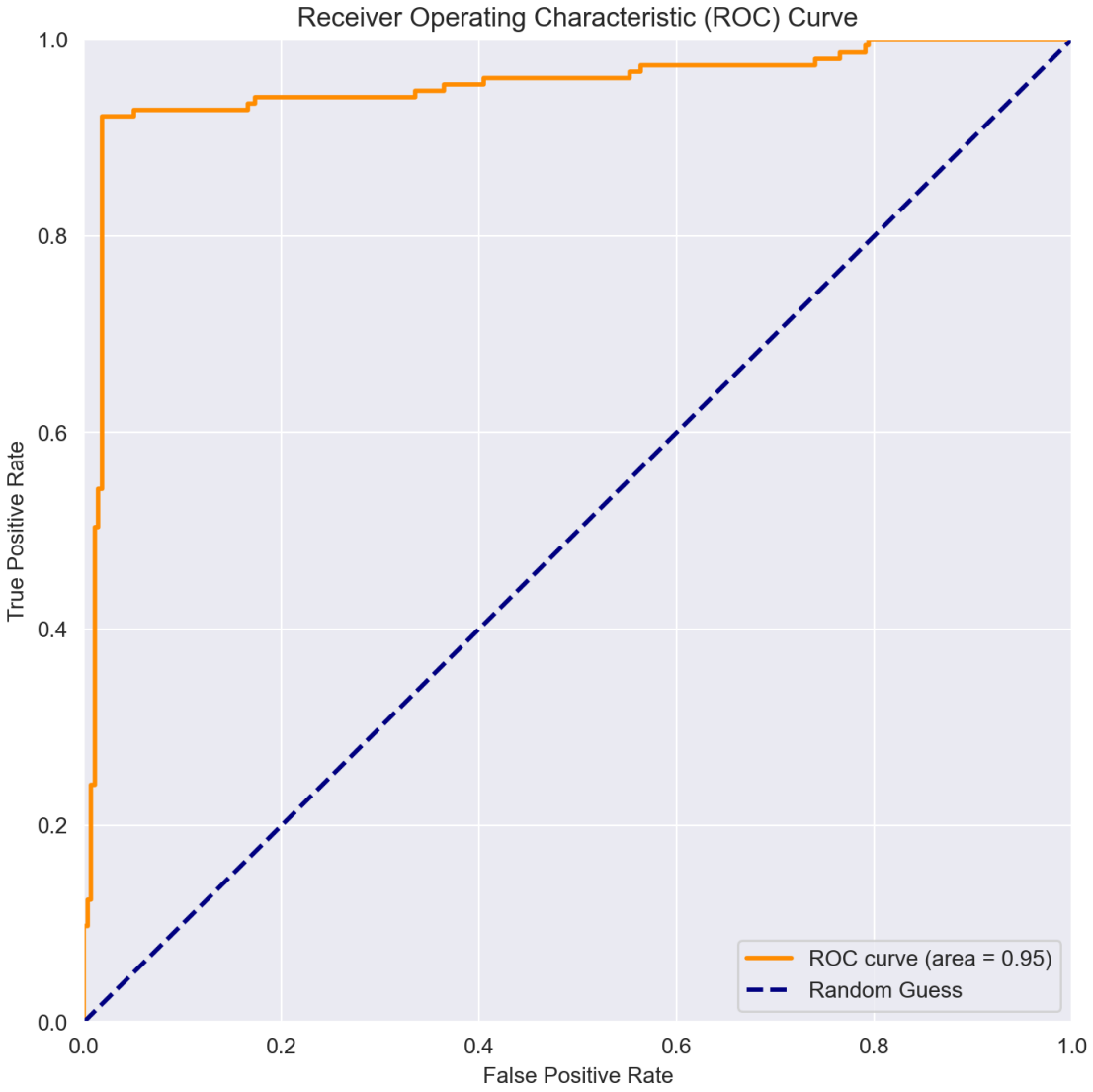
```
features = X.columns
importance_df = pd.DataFrame({'Feature': features, 'Importance':␣
 ↪feature_importances})
importance_df.sort_values(by='Importance', ascending=False, inplace=True)
print(importance_df)
```

```
                     Feature  Importance
26                       ADL         722
23       FunctionalAssessment         655
22                       MMSE         571
8                 DietQuality         488
18           CholesterolTotal         373
9                 SleepQuality         347
25         BehavioralProblems         343
19             CholesterolLDL         318
16                 SystolicBP         312
24           MemoryComplaints         299
7            PhysicalActivity         298
0                         Age         287
6          AlcoholConsumption         285
20             CholesterolHDL         279
32                HealthScore         265
17                DiastolicBP         257
4                         BMI         238
34           TotalHealthScore         219
33       CardiometabolicIndex         198
21   CholesterolTriglycerides         186
3              EducationLevel         113
5                     Smoking          60
2                    Ethnicity          55
28             Disorientation          46
29         PersonalityChanges          23
27                   Confusion          20
11      CardiovascularDisease          17
1                        Gender          17
13                Depression          17
31              Forgetfulness          16
10    FamilyHistoryAlzheimers           8
14                HeadInjury           7
12                  Diabetes           6
15              Hypertension           4
30  DifficultyCompletingTasks           1
```

[73]:
```
# Compute predicted probabilities for the positive class
y_pred_prob = best_model.predict_proba(X_test)[:, 1]

# Compute ROC curve
```

```
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

# Compute AUC score
roc_auc = auc(fpr, tpr)
print(f"AUC-ROC score: {roc_auc}")

# Plot the ROC curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:
 ↪.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label="Random␣
 ↪Guess")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

```
AUC-ROC score: 0.9515584813949647
```



Interpretation: The ROC curve, along with a high AUC-ROC score, means this model is able to distinguish very well between classes.

8 Summary

In this project, I developed a predictive model using the **LightGBM Classifier (LGBMClassifier)** to estimate the probability of developing Alzheimer's disease based on a holistic health dataset. The model was trained on a diverse set of features, including demographic information, cognitive assessments, physical health indicators, and lifestyle factors, providing a comprehensive approach to Alzheimer's disease risk assessment.

Model Performance The LGBMClassifier demonstrated strong performance on the test dataset, achieving a **Test Accuracy of 95.814%**. The detailed classification report shows:

- **Precision**: The model achieved a precision of 0.95 for the "Not Diagnosed with Alzheimer's" class (0) and 0.97 for the "Diagnosed with Alzheimer's" class (1). This indicates that the model is highly accurate in predicting both classes, with a particularly high precision for identifying those diagnosed with Alzheimer's.
- **Recall**: The recall values were 0.98 for the "Not Diagnosed" class and 0.92 for the "Diagnosed" class, reflecting the model's effectiveness in identifying true positives in each category.
- **F1-Score**: The F1-scores were 0.97 for the "Not Diagnosed" class and 0.94 for the "Diagnosed" class, providing a balanced measure of the model's precision and recall.
- **AUC-ROC Score**: The model achieved an impressive **AUC-ROC score of 0.9516**, indicating excellent ability to distinguish between individuals who are and are not at risk of developing Alzheimer's disease.

Overall, the LGBMClassifier model exhibits robust performance, making it a reliable tool for estimating Alzheimer's disease risk in a diverse population.

Deployment on Heroku To make the model accessible to users, I deployed it as a web application using **Heroku**. The application is built with Flask, a lightweight web framework in Python, and is hosted on Heroku to ensure scalability, reliability, and ease of access.

Heroku is a cloud platform that simplifies app deployment and scaling. It offers a streamlined process for building, deploying, and managing applications using various programming languages. With its user-friendly interface and integrated tools, Heroku abstracts away infrastructure management, allowing developers to focus on coding. It supports multiple deployment methods, including Git, and provides a range of add-ons for databases, caching, and monitoring. Ideal for developers looking for a hassle-free deployment experience, Heroku is great for both small projects and scalable applications.

Key Steps in Deployment:

1. **Model Integration**: The trained LGBMClassifier model was integrated into a Flask application, where it processes user input and provides a probabilistic estimate of Alzheimer's disease risk.
2. **Flask Application Development**: The Flask app was designed with an intuitive frontend in HTML that allows users to input their health-related data. The backend processes this data, runs it through the trained model, and returns a probability score indicating the risk of developing Alzheimer's.
3. **Heroku Deployment**: The Flask application was deployed on Heroku, which provides a fully managed environment for web applications. This deployment ensures that the application can handle multiple users concurrently and can scale based on demand.
4. **Security and Access Control**: Heroku's robust security features were utilized to secure the application, including SSL certificates for secure data transmission and identity management to control access to the application.
5. **Monitoring and Maintenance**: The application is monitored using Heroku's monitoring tools, which allow us to track performance, uptime, and usage statistics. This ensures that the application remains responsive and reliable for users.

By deploying the application on Heroku, we have ensured that the Alzheimer's disease risk assessment tool is accessible, scalable, and secure, making it a valuable resource for individuals seeking to understand their potential risk and take proactive steps in managing their health.

[ ]: