AD_classification_version_2

September 5, 2025

1 Classifying Alzheimer Disease through Feature Analysis

- Original Paper - https://ieeexplore.ieee.org/document/11041645
- Dataset - https://www.kaggle.com/datasets/brsdincer/alzheimer-features/
- (Version 1_Experiment 1) Use demented, nondemented, converted data and categorize demented and converted as class 1, nondemented as class 0 - https://colab.research.google.com/drive/10LysLErBy-Qg2FWZvh2yAr04m2w0cDgq?usp=sharing
- (**This Colab_Version 2_Experiment 2~4**) Use only demented and nondemented group, drop converted group which may introduce noise and make the boundary not that clear - https://colab.research.google.com/drive/1N_CkOWYlKjEEvL2YLxS_ZuBVN-AgeYhO?usp=sharing
- (Version 3) Use dataset from original OASIS 2 website, focus on first visit data(150 samples) - https://colab.research.google.com/drive/1TgLG24-YDWskGqbID0ZdeBd-RB9uHVZw?usp=sharing

2 Libraries installing

[ ]:
```
!pip install lime shap catboost
```

```
Collecting lime
  Downloading lime-0.2.0.1.tar.gz (275 kB)
                              275.7/275.7

kB 2.3 MB/s eta 0:00:00
  Preparing metadata (setup.py) … done
Requirement already satisfied: shap in /usr/local/lib/python3.11/dist-packages
(0.48.0)
Collecting catboost
  Downloading catboost-1.2.8-cp311-cp311-manylinux2014_x86_64.whl.metadata (1.2
kB)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-
packages (from lime) (3.10.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages
(from lime) (2.0.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages
```

```
(from lime) (1.16.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages
(from lime) (4.67.1)
Requirement already satisfied: scikit-learn>=0.18 in
/usr/local/lib/python3.11/dist-packages (from lime) (1.6.1)
Requirement already satisfied: scikit-image>=0.12 in
/usr/local/lib/python3.11/dist-packages (from lime) (0.25.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages
(from shap) (2.2.2)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.11/dist-
packages (from shap) (25.0)
Requirement already satisfied: slicer==0.0.8 in /usr/local/lib/python3.11/dist-
packages (from shap) (0.0.8)
Requirement already satisfied: numba>=0.54 in /usr/local/lib/python3.11/dist-
packages (from shap) (0.60.0)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.11/dist-
packages (from shap) (3.1.1)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.11/dist-packages (from shap) (4.14.1)
Requirement already satisfied: graphviz in /usr/local/lib/python3.11/dist-
packages (from catboost) (0.21)
Requirement already satisfied: plotly in /usr/local/lib/python3.11/dist-packages
(from catboost) (5.24.1)
Requirement already satisfied: six in /usr/local/lib/python3.11/dist-packages
(from catboost) (1.17.0)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in
/usr/local/lib/python3.11/dist-packages (from numba>=0.54->shap) (0.43.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.11/dist-packages (from pandas->shap) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-
packages (from pandas->shap) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-
packages (from pandas->shap) (2025.2)
Requirement already satisfied: networkx>=3.0 in /usr/local/lib/python3.11/dist-
packages (from scikit-image>=0.12->lime) (3.5)
Requirement already satisfied: pillow>=10.1 in /usr/local/lib/python3.11/dist-
packages (from scikit-image>=0.12->lime) (11.3.0)
Requirement already satisfied: imageio!=2.35.0,>=2.33 in
/usr/local/lib/python3.11/dist-packages (from scikit-image>=0.12->lime) (2.37.0)
Requirement already satisfied: tifffile>=2022.8.12 in
/usr/local/lib/python3.11/dist-packages (from scikit-image>=0.12->lime)
(2025.6.11)
Requirement already satisfied: lazy-loader>=0.4 in
/usr/local/lib/python3.11/dist-packages (from scikit-image>=0.12->lime) (0.4)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-
packages (from scikit-learn>=0.18->lime) (1.5.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn>=0.18->lime) (3.6.0)
```

```
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->lime) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-
packages (from matplotlib->lime) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->lime) (4.59.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->lime) (1.4.9)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.11/dist-packages (from matplotlib->lime) (3.2.3)
Requirement already satisfied: tenacity>=6.2.0 in
/usr/local/lib/python3.11/dist-packages (from plotly->catboost) (9.1.2)
Downloading catboost-1.2.8-cp311-cp311-manylinux2014_x86_64.whl (99.2 MB)
                         99.2/99.2 MB
25.1 MB/s eta 0:00:00
Building wheels for collected packages: lime
  Building wheel for lime (setup.py) … done
  Created wheel for lime: filename=lime-0.2.0.1-py3-none-any.whl size=283834
sha256=7a304cfa1ffa5039c8e358222fd4a3fd0bd421747827ce31a82b38b6dcc9d988
  Stored in directory: /root/.cache/pip/wheels/85/fa/a3/9c2d44c9f3cd77cf4e533b58
900b2bf4487f2a17e8ec212a3d
Successfully built lime
Installing collected packages: lime, catboost
Successfully installed catboost-1.2.8 lime-0.2.0.1
```

[ ]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
import shap
import lime
```

[ ]:
```python
from sklearn.metrics import accuracy_score, f1_score,␣
 ↪roc_auc_score,confusion_matrix, classification_report, precision_score,␣
 ↪recall_score, roc_curve
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV, RepeatedStratifiedKFold,␣
 ↪cross_validate

from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
```

```
from catboost import CatBoostClassifier
from collections import Counter
```

3 Dataset loading

[ ]:
```
df = pd.read_csv('alzheimer.csv')
```

[ ]:
```
df.shape
```

[ ]:
```
(373, 10)
```

The dataset has 373 samples with 10 features

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

[ ]:
```
         Group M/F  Age  EDUC  SES  MMSE  CDR  eTIV   nWBV    ASF
0  Nondemented   M   87    14  2.0  27.0  0.0  1987  0.696  0.883
1  Nondemented   M   88    14  2.0  30.0  0.0  2004  0.681  0.876
2     Demented   M   75    12  NaN  23.0  0.5  1678  0.736  1.046
3     Demented   M   76    12  NaN  28.0  0.5  1738  0.713  1.010
4     Demented   M   80    12  NaN  22.0  0.5  1698  0.701  1.034
```

[ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 373 entries, 0 to 372
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Group   373 non-null    object
 1   M/F     373 non-null    object
 2   Age     373 non-null    int64
 3   EDUC    373 non-null    int64
 4   SES     354 non-null    float64
 5   MMSE    371 non-null    float64
 6   CDR     373 non-null    float64
 7   eTIV    373 non-null    int64
 8   nWBV    373 non-null    float64
 9   ASF     373 non-null    float64
dtypes: float64(5), int64(3), object(2)
memory usage: 29.3+ KB
```

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

[ ]:
```
              Age        EDUC         SES        MMSE         CDR  \
count  373.000000  373.000000  354.000000  371.000000  373.000000
```

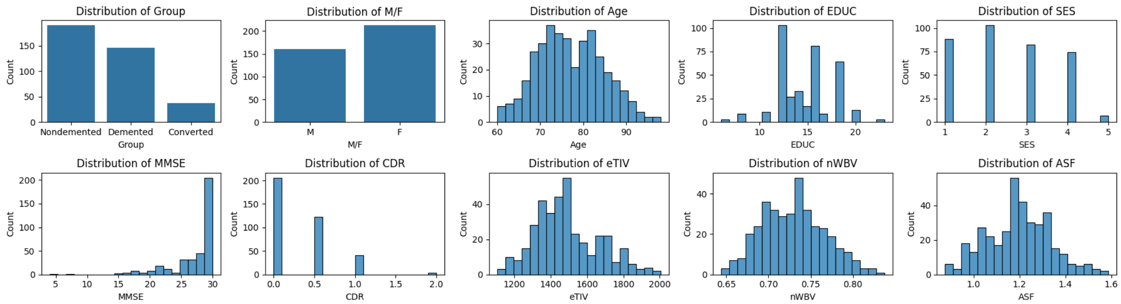
```
mean     77.013405   14.597855    2.460452   27.342318    0.290885
std       7.640957    2.876339    1.134005    3.683244    0.374557
min      60.000000    6.000000    1.000000    4.000000    0.000000
25%      71.000000   12.000000    2.000000   27.000000    0.000000
50%      77.000000   15.000000    2.000000   29.000000    0.000000
75%      82.000000   16.000000    3.000000   30.000000    0.500000
max      98.000000   23.000000    5.000000   30.000000    2.000000

              eTIV        nWBV         ASF
count   373.000000  373.000000  373.000000
mean   1488.128686    0.729568    1.195461
std     176.139286    0.037135    0.138092
min    1106.000000    0.644000    0.876000
25%    1357.000000    0.700000    1.099000
50%    1470.000000    0.729000    1.194000
75%    1597.000000    0.756000    1.293000
max    2004.000000    0.837000    1.587000
```

[ ]:

4 Initial data exploration

[ ]:

```
# Distribution chart of each feature
numerical_cols = ['Age', 'EDUC', 'SES', 'MMSE', 'CDR', 'eTIV', 'nWBV', 'ASF']
categorical_cols = ['Group', 'M/F']
all_cols = categorical_cols + numerical_cols

# Set up the matplotlib figure
plt.figure(figsize=(18, 5))

# Loop through each column and plot distribution
for i, col in enumerate(all_cols):
    plt.subplot(2, 5, i + 1)

    if col in numerical_cols:
        sb.histplot(df[col].dropna(), bins=20)
        plt.ylabel("Count")
    else:
        sb.countplot(x=col, data=df)
        plt.ylabel("Count")

    plt.title(f'Distribution of {col}')
    plt.xlabel(col)

plt.tight_layout()
plt.show()
```



#Preprocessing

##Handling Missing Values

From below, we can say that there are 19 missing values in SES and 2 missing values in MMSE.

[ ]:
```
df.isnull().sum()
```

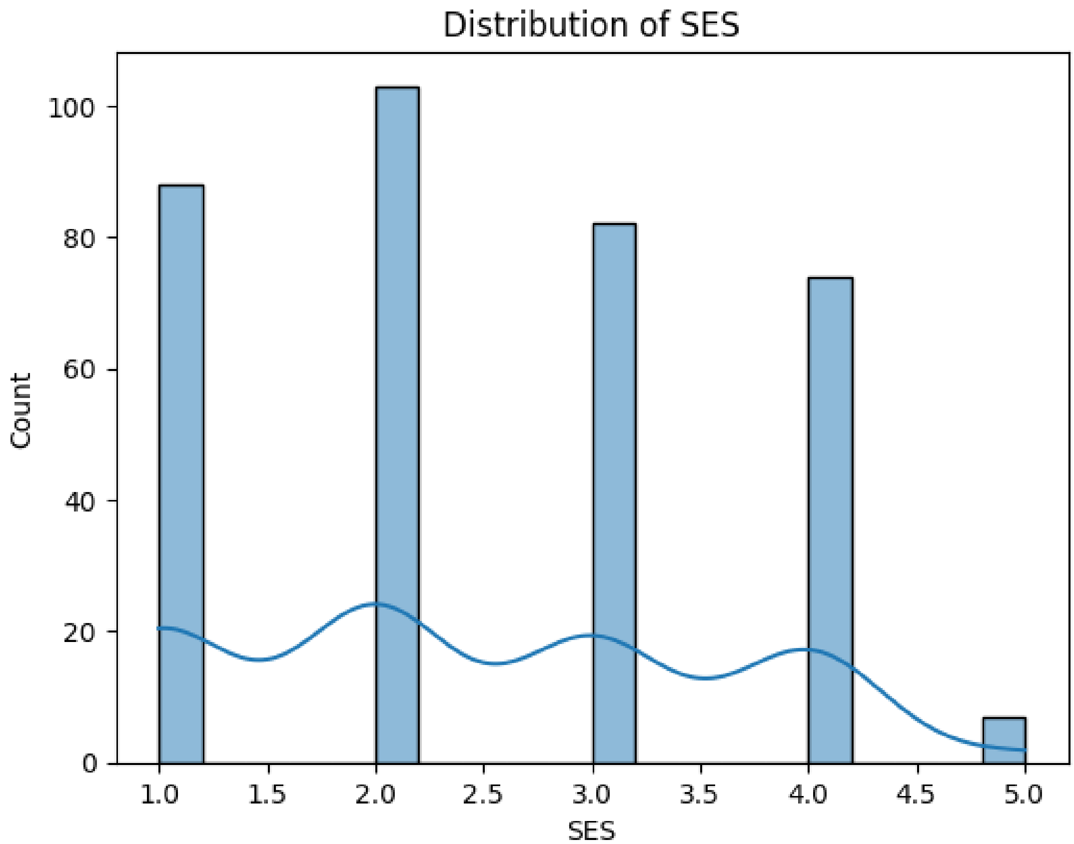
[ ]:
```
Group      0
M/F        0
Age        0
EDUC       0
SES       19
MMSE       2
CDR        0
eTIV       0
nWBV       0
ASF        0
dtype: int64
```

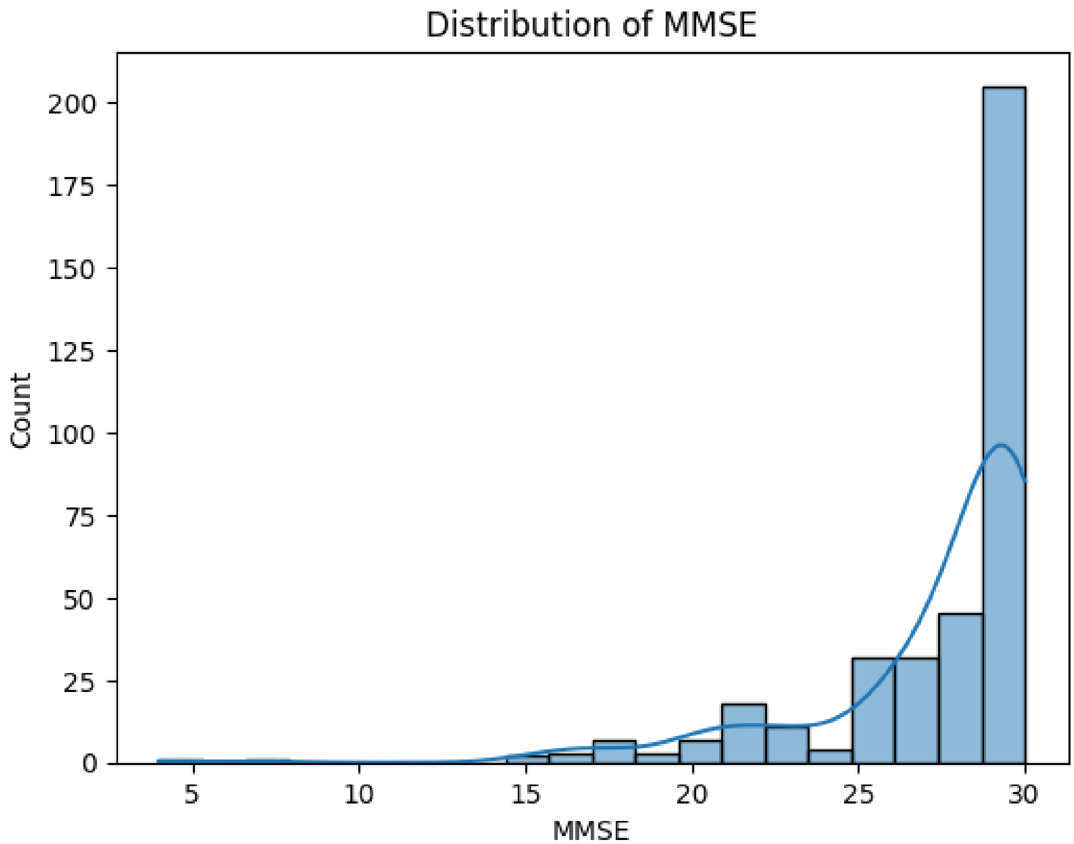
Observing the distribution of ‘SES’ and ‘MMSE’ to understand the strategy for imputing missing values.

From below graphs we can say that ‘SES’ is evenly distributed and we can use ‘Mean’ as the strategy for imputing, where as ‘MMSE’ is highly negatively skewed and has outliers, where median would be a better strategy for imputing.

[ ]:
```
def plot_distribution(column_name):
    sb.histplot(df[column_name], kde=True, bins=20)
    plt.title(f"Distribution of {column_name}")
    plt.xlabel(column_name)
    plt.ylabel("Count")
    plt.show()
```

[ ]:
```
plot_distribution('SES')
plot_distribution('MMSE')
```


Distribution of SES
Count
SES
0
20
40
60
80
100
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0



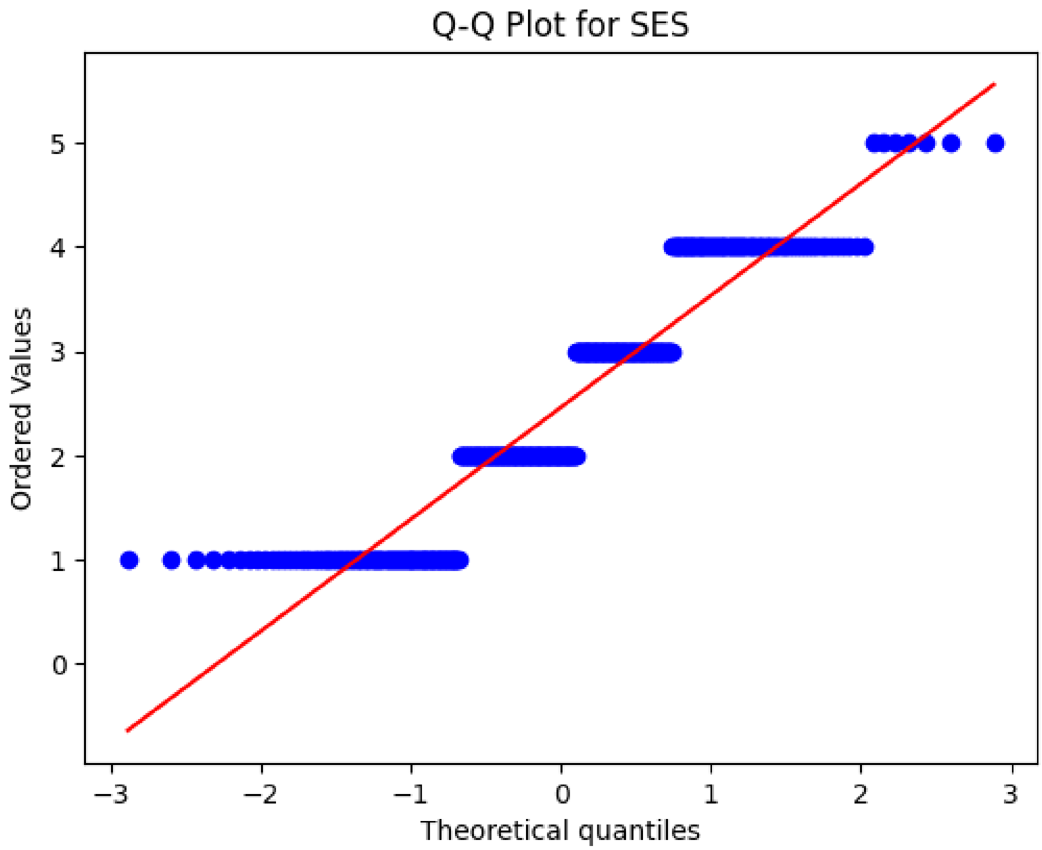
[ ]:
```
skew_value = df['SES'].skew()
print(f"Skewness of SES: {skew_value:.2f}")
```

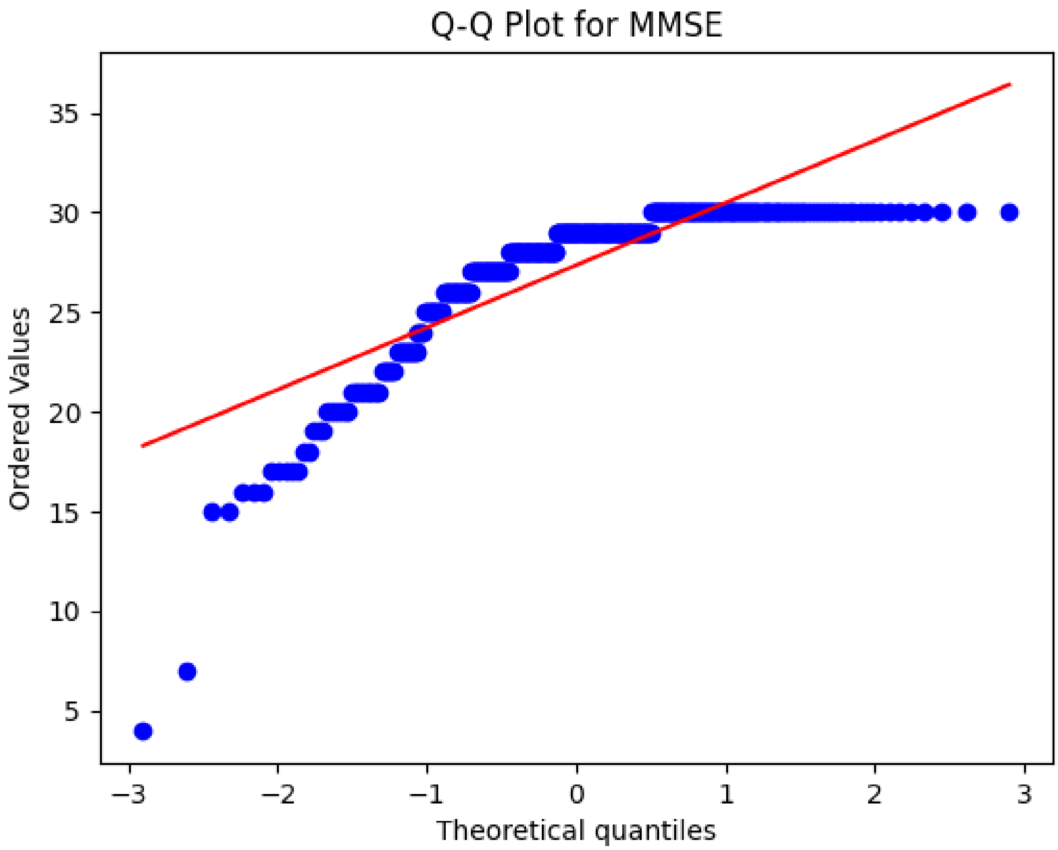
Skewness of SES: 0.22

[ ]:
```
skew_value = df['MMSE'].skew()
print(f"Skewness of MMSE: {skew_value:.2f}")
```

Skewness of MMSE: -2.37

[ ]:
```
import scipy.stats as stats

stats.probplot(df['SES'].dropna(), dist="norm", plot=plt)
plt.title("Q-Q Plot for SES")
plt.show()

stats.probplot(df['MMSE'].dropna(), dist="norm", plot=plt)
plt.title("Q-Q Plot for MMSE")
plt.show()
```


Q-Q Plot for SES
Ordered Values
Theoretical quantiles
5
4
3
2
1
0
−3
−2
−1
0
1
2
3



[ ]:
```
# fill missing values in SES and MMSE columns with average
df['SES'] = df['SES'].fillna(df['SES'].mean())
df['MMSE'] = df['MMSE'].fillna(df['MMSE'].median())
```

[ ]:
```
df.isna().sum().sum()
```

[ ]:
```
np.int64(0)
```

[ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 373 entries, 0 to 372
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Group   373 non-null    object
 1   M/F     373 non-null    object
 2   Age     373 non-null    int64
 3   EDUC    373 non-null    int64
 4   SES     373 non-null    float64
```

```
 5   MMSE    373 non-null    float64
 6   CDR     373 non-null    float64
 7   eTIV    373 non-null    int64
 8   nWBV    373 non-null    float64
 9   ASF     373 non-null    float64
dtypes: float64(5), int64(3), object(2)
memory usage: 29.3+ KB
```

##Label Encoding

(Modified)The dataset has three variations of the target value, being Demented(146), Nondemented(190), and Converted(37). For binary classification, it's better to have clear boundary and not to introduce too much noise or unclear information. We will use only demented and nondemented data for this classsification task.

[ ]:
```
df['M/F'] = df['M/F'].map({'M': 0, 'F': 1})
```

[ ]:
```
df['Group'].value_counts()
```

[ ]:
```
Group
Nondemented    190
Demented       146
Converted       37
Name: count, dtype: int64
```

[ ]:
```
# make a new dataframe to keep only converted data as an additional test dataset
df_converted = df[df['Group'] == 'Converted']
```

[ ]:
```
df_converted.head()
df_converted.shape
```

[ ]:
```
(37, 10)
```

[ ]:
```
# keep only df['Group'] nondemented and demented, remove converted
df = df[df['Group'].isin(['Demented', 'Nondemented'])]
df['Group'].value_counts()
```

[ ]:
```
Group
Nondemented    190
Demented       146
Name: count, dtype: int64
```

[ ]:
```
df['Group'] = df['Group'].map({'Nondemented': 0, 'Demented': 1})
```

[ ]:
```
df['Group'].value_counts()
```

[ ]:
```
Group
0    190
1    146
```

```
Name: count, dtype: int64
```

```
[ ]: print(df.head(10))
```

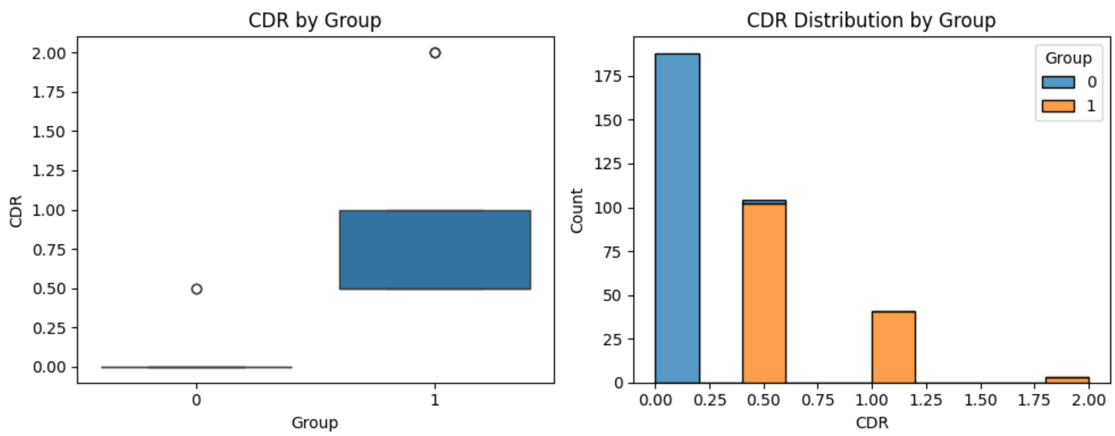
```
   Group  M/F  Age  EDUC       SES  MMSE  CDR  eTIV   nWBV    ASF
0      0    0   87    14  2.000000  27.0  0.0  1987  0.696  0.883
1      0    0   88    14  2.000000  30.0  0.0  2004  0.681  0.876
2      1    0   75    12  2.460452  23.0  0.5  1678  0.736  1.046
3      1    0   76    12  2.460452  28.0  0.5  1738  0.713  1.010
4      1    0   80    12  2.460452  22.0  0.5  1698  0.701  1.034
5      0    1   88    18  3.000000  28.0  0.0  1215  0.710  1.444
6      0    1   90    18  3.000000  27.0  0.0  1200  0.718  1.462
7      0    0   80    12  4.000000  28.0  0.0  1689  0.712  1.039
8      0    0   83    12  4.000000  29.0  0.5  1701  0.711  1.032
9      0    0   85    12  4.000000  30.0  0.0  1699  0.705  1.033
```

```
[ ]: # export current df to csv file
     df.to_csv('alzheimer_cleaned.csv', index=False)
```

5 Quick check on subjective features: CDR and MMSE

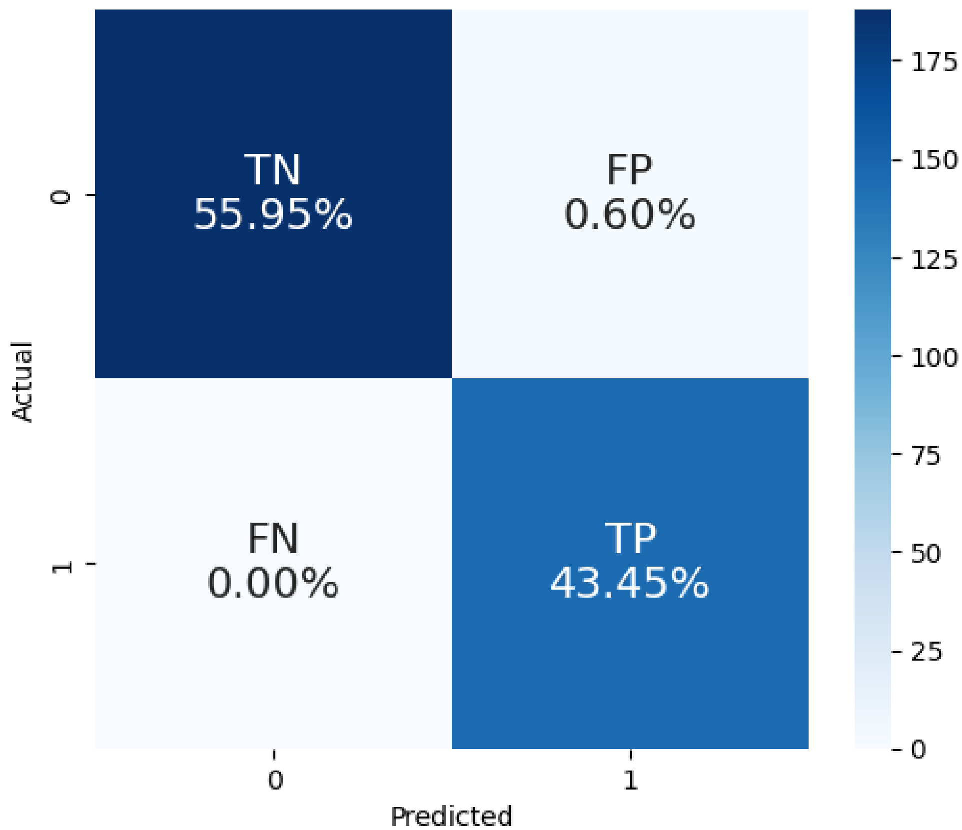
5.1 Check CDR

```
[ ]: # Create a figure with 1 row and 2 columns of subplots
     fig, axes = plt.subplots(1, 2, figsize=(10, 4))

     # First subplot: Boxplot of CDR by Group
     sb.boxplot(x='Group', y='CDR', data=df, ax=axes[0])
     axes[0].set_title('CDR by Group')
     axes[0].set_xlabel('Group')
     axes[0].set_ylabel('CDR')

     # Second subplot: Distribution of CDR colored by Group
     sb.histplot(data=df, x='CDR', hue='Group', bins=10, multiple='stack',␣
      ↪ax=axes[1]) # or multiple='dodge'
     axes[1].set_title('CDR Distribution by Group')
     axes[1].set_xlabel('CDR')
     axes[1].set_ylabel('Count')

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



[ ]:
```
# Binarize CDR: 0 stays 0, anything >0 becomes 1
df2 = df[['Group','CDR']].dropna().copy()
df2['CDR_bin'] = (df2['CDR'] > 0).astype(int)
```

[ ]:
```
cf_matrix = confusion_matrix(df2['Group'], df2['CDR_bin'])

tn, fp, fn, tp = cf_matrix.ravel()/np.sum(cf_matrix)

labels = np.array([
    [f"TN\n{tn:.2%}", f"FP\n{fp:.2%}"],
    [f"FN\n{fn:.2%}", f"TP\n{tp:.2%}"]
])
plt.figure(figsize=(6, 5))

sb.heatmap(cf_matrix, annot=labels,\
            fmt='', cmap='Blues', annot_kws={'size':16})
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



Group is the ground truth, use CDR_bin to predict ground truth could get 99.4% accuracy. which means that they are almost identical, use CDR_bin is like cheating or data leakage.

[ ]:
```
# A quick model running on only one feature_CDR to check if it's cheating to␣
↪use it

# Use CDR as the only feature
X = df[['CDR']]
y = df['Group']

# train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# train model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# evaluate
y_pred = model.predict(X_test)
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        38
           1       1.00      1.00      1.00        30

    accuracy                           1.00        68
   macro avg       1.00      1.00      1.00        68
weighted avg       1.00      1.00      1.00        68
```

Using only one feature CDR: The value of all the metrics(Accuracy precision recall f1-score) are all 1, means that CDR is highly predictive-strong evidence of target leakage, like another representation of target group. Using CDR as a feature to predict group target is cheating which must be removed from our dataset.

In addition, just because CDR is a cheating feature to predict group target, we can exploy it as the key feature to test our converted dataset(a combination of demented and non-demented) to determine their real group(non-demented or demented)

[ ]:
```
X_test_converted = df_converted[['CDR']]
y_pred_converted = model.predict(X_test_converted)
```

[ ]:
```
df_converted['Group'] = y_pred_converted
```

[ ]:
```
df_converted.head(5)
```

[ ]:

	Group	M/F	Age	EDUC	SES	MMSE	CDR	eTIV	nWBV	ASF
33	0	1	87	14	1.0	30.0	0.0	1406	0.715	1.248
34	0	1	88	14	1.0	29.0	0.0	1398	0.713	1.255
35	1	1	92	14	1.0	27.0	0.5	1423	0.696	1.234
36	0	0	80	20	1.0	29.0	0.0	1587	0.693	1.106
37	1	0	82	20	1.0	28.0	0.5	1606	0.677	1.093

[ ]:
```
# combine df and df_converted dataset
# df_combined = pd.concat([df, df_converted])
```

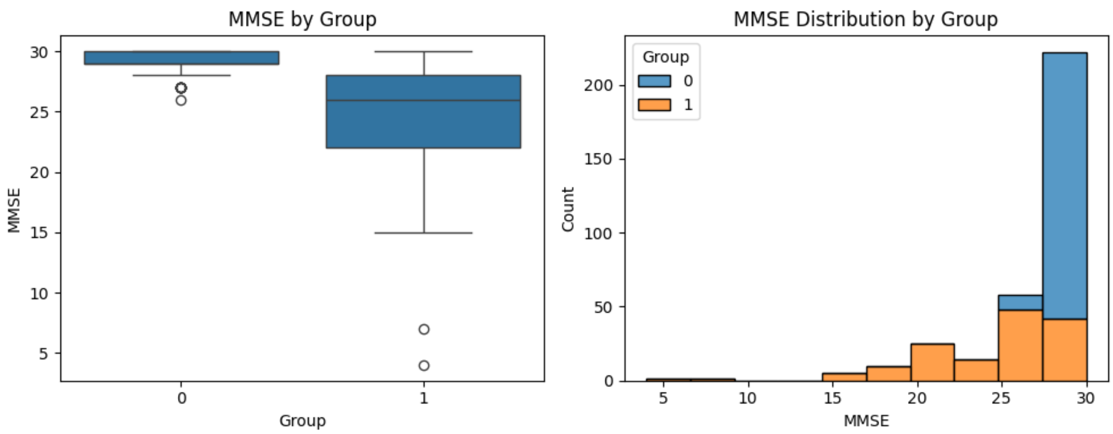
[ ]:
```
# df_combined['Group'].value_counts()
```

[ ]:
```
# df_combined.head(5)
```

5.2 Check MMSE

[ ]:
```
# Create a figure with 1 row and 2 columns of subplots
fig, axes = plt.subplots(1, 2, figsize=(10, 4))

# First subplot: Boxplot of MMSE by Group
```

```
sb.boxplot(x='Group', y='MMSE', data=df, ax=axes[0])
axes[0].set_title('MMSE by Group')
axes[0].set_xlabel('Group')
axes[0].set_ylabel('MMSE')

# Second subplot: Distribution of MMSE colored by Group
sb.histplot(data=df, x='MMSE', hue='Group', bins=10, multiple='stack',␣
↪ax=axes[1]) # or multiple='dodge'
axes[1].set_title('MMSE Distribution by Group')
axes[1].set_xlabel('MMSE')
axes[1].set_ylabel('Count')

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



[ ]:
```
# A quick model running on only one feature_MMSE to check if it's cheating to␣
↪use it

# use only MMSE as feature
X = df[['MMSE']]
y = df['Group']

# train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# train model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
```

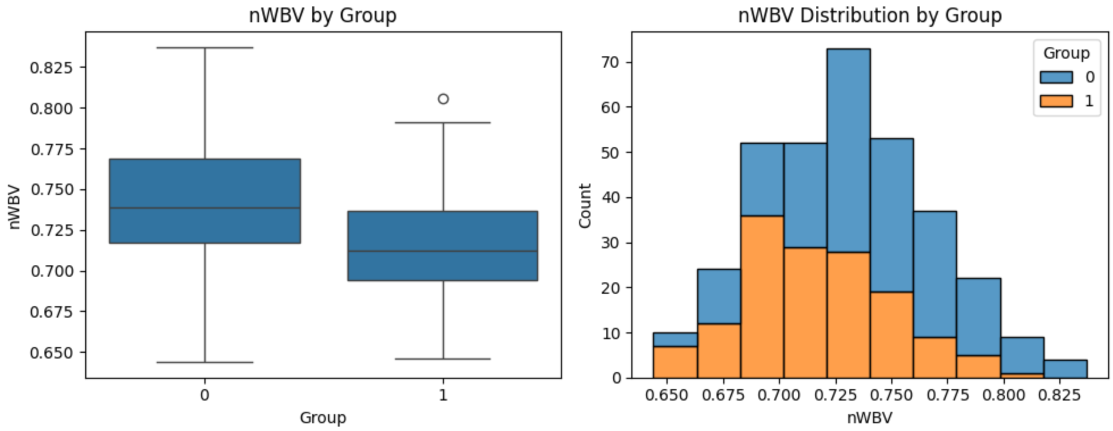
```
# evaluate
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.97      0.93        38
           1       0.96      0.83      0.89        30

    accuracy                           0.91        68
   macro avg       0.92      0.90      0.91        68
weighted avg       0.92      0.91      0.91        68
```

Using only one feature MMSE: Accuracy is pretty high 0.91, means that MMSE is also a highly predictive-strong evidence of target leakage. Using MMSE as a feature to predict group target is like cheating which must be also removed from our dataset for fairness.

5.3 (additional) Check nWBV

[ ]:
```
# Create a figure with 1 row and 2 columns of subplots
fig, axes = plt.subplots(1, 2, figsize=(10, 4))

# First subplot: Boxplot of nWBV by Group
sb.boxplot(x='Group', y='nWBV', data=df, ax=axes[0])
axes[0].set_title('nWBV by Group')
axes[0].set_xlabel('Group')
axes[0].set_ylabel('nWBV')

# Second subplot: Distribution of nWBV colored by Group
sb.histplot(data=df, x='nWBV', hue='Group', bins=10, multiple='stack',␣
↪ax=axes[1]) # or multiple='dodge'
axes[1].set_title('nWBV Distribution by Group')
axes[1].set_xlabel('nWBV')
axes[1].set_ylabel('Count')

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



[ ]:
```
# A quick model running on only one feature_MMSE to check if it's cheating to␣
↪use it

# use only MMSE as feature
X = df[['nWBV']]
y = df['Group']

# train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# train model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# evaluate
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.56      0.58      0.57        38
           1       0.45      0.43      0.44        30

    accuracy                           0.51        68
   macro avg       0.51      0.51      0.51        68
weighted avg       0.51      0.51      0.51        68
```

Using only one feature nWBV: Accuracy is only 0.50 which means that only nWBV cannot predict the reseult very well.

6 Experiment 2 _Original process(key paper replication with all features)

[ ]:
```
y = df['Group']
X = df.drop(['Group'], axis = 1)
```

[ ]:
```
print(X.head(10))
```

```
   M/F  Age  EDUC       SES  MMSE  CDR  eTIV   nWBV    ASF
0    0   87    14  2.000000  27.0  0.0  1987  0.696  0.883
1    0   88    14  2.000000  30.0  0.0  2004  0.681  0.876
2    0   75    12  2.460452  23.0  0.5  1678  0.736  1.046
3    0   76    12  2.460452  28.0  0.5  1738  0.713  1.010
4    0   80    12  2.460452  22.0  0.5  1698  0.701  1.034
5    1   88    18  3.000000  28.0  0.0  1215  0.710  1.444
6    1   90    18  3.000000  27.0  0.0  1200  0.718  1.462
7    0   80    12  4.000000  28.0  0.0  1689  0.712  1.039
8    0   83    12  4.000000  29.0  0.5  1701  0.711  1.032
9    0   85    12  4.000000  30.0  0.0  1699  0.705  1.033
```

[ ]:
```
print(y.head(10))
```

```
0    0
1    0
2    1
3    1
4    1
5    0
6    0
7    0
8    0
9    0
Name: Group, dtype: int64
```

Here we can observe that the data contains nearly double the counts of '0' to '1'.

[ ]:
```
y.value_counts()
```

[ ]:
```
Group
0    190
1    146
Name: count, dtype: int64
```

##Train test split

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
↪random_state = 42)
```

6.1 Model Training

Classifier Models

[ ]:
```
models = [
    "XGBoost",
    "LightGBM",
    "CatBoost"
]

classifiers = [
    XGBClassifier(),
    LGBMClassifier(verbosity=-1),
    CatBoostClassifier(verbose = False)
]
```

##Hyper-Parameter tuning

Parameter-grid for Hyper-parameter tuning

[ ]:
```
parameters_grid = [
    {
        'xgbclassifier__n_estimators': [100, 200],
        'xgbclassifier__max_depth': [3, 5, 7],
        'xgbclassifier__learning_rate': [0.001, 0.01, 0.1],
        'xgbclassifier__subsample':[0.7, 0.8, 0.9],
        'xgbclassifier__colsample_bytree': [0.7, 0.8, 1.0],
        'xgbclassifier__min_child_weight': [1, 3, 5]
    },

    {
        'lgbmclassifier__n_estimators': [100, 200],
        'lgbmclassifier__max_depth': [3, 5, 7],
        'lgbmclassifier__learning_rate': [0.001, 0.01, 0.1],
        'lgbmclassifier__num_leaves': [31, 50, 70]
    },

    {
        'catboostclassifier__min_data_in_leaf': [20, 40, 60],
        'catboostclassifier__rsm': [0.7, 0.8, 1.0],
        'catboostclassifier__iterations': [100, 200],
        'catboostclassifier__depth': [3, 5, 7],
        'catboostclassifier__learning_rate': [0.001, 0.01, 0.1],
        'catboostclassifier__l2_leaf_reg': [1, 3, 5],
        'catboostclassifier__bagging_temperature': [0, 0.5, 1]
    }
]
```

[ ]:
```python
from sklearn.base import TransformerMixin, BaseEstimator

class DataFrameStandardScaler(TransformerMixin, BaseEstimator):
    def __init__(self):
        self.scaler = StandardScaler()
        self.columns = None

    def fit(self, X, y=None):
        self.columns = X.columns
        self.scaler.fit(X)
        return self

    def transform(self, X):
        scaled = self.scaler.transform(X)
        return pd.DataFrame(scaled, columns=self.columns, index=X.index)
```

[ ]:
```python
# A dictionary to store trained models
trained_models = {}

for name, clf, param_grid in zip(models, classifiers, parameters_grid):
  pipeline = make_pipeline(DataFrameStandardScaler(), clf)

  grid = GridSearchCV(pipeline, param_grid, cv = 5, scoring = 'accuracy')
  grid.fit(X_train, y_train)

  best_model = grid.best_estimator_

  # Storing the models with their best parameters for SHAP analysis
  trained_models[name] = {
      'model': best_model,
      'best_params': grid.best_params_
  }

  y_pred = best_model.predict(X_test)

  print("--------------------------------------------")

  print(name)
  print("Accuracy:", accuracy_score(y_test, y_pred))
  print("Precision:", precision_score(y_test, y_pred))
  print("Recall:", recall_score(y_test, y_pred))
  print("F1 Score:", f1_score(y_test, y_pred))
  print("AUC:", roc_auc_score(y_test, y_pred))
  print("Best Parameters:", grid.best_params_)
```

```
-------------------------------------------
XGBoost
```

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
AUC: 1.0
Best Parameters: {'xgbclassifier__colsample_bytree': 0.7,
'xgbclassifier__learning_rate': 0.01, 'xgbclassifier__max_depth': 3,
'xgbclassifier__min_child_weight': 1, 'xgbclassifier__n_estimators': 100,
'xgbclassifier__subsample': 0.7}
-------------------------------------------
LightGBM
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
AUC: 1.0
Best Parameters: {'lgbmclassifier__learning_rate': 0.001,
'lgbmclassifier__max_depth': 3, 'lgbmclassifier__n_estimators': 200,
'lgbmclassifier__num_leaves': 31}
-------------------------------------------
CatBoost
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
AUC: 1.0
Best Parameters: {'catboostclassifier__bagging_temperature': 0,
'catboostclassifier__depth': 3, 'catboostclassifier__iterations': 100,
'catboostclassifier__l2_leaf_reg': 1, 'catboostclassifier__learning_rate':
0.001, 'catboostclassifier__min_data_in_leaf': 20, 'catboostclassifier__rsm':
0.7}
```

##Cross-Validation

[ ]:
```
from sklearn.model_selection import RepeatedStratifiedKFold, cross_validate,␣
 ↪LeaveOneOut

print("Cross Validation: ")

scoring = {
    'accuracy': 'accuracy',
    'precision': 'precision',
    'recall': 'recall',
    'f1': 'f1',
    'roc_auc': 'roc_auc'
}

for i, (name, data) in enumerate(trained_models.items()):
```

```
    model = data['model']

    # Different random state for each model
    cv_different = RepeatedStratifiedKFold(n_splits=5, n_repeats=2,␣
↪random_state=42+i*10)

    scores = cross_validate(model, X_train, y_train, cv=cv_different,␣
↪scoring=scoring)

    print(f"\n{name} (Random State {42+i*10}):")
    for metric in scoring.keys():
        mean_score = scores[f'test_{metric}'].mean()
        std_score = scores[f'test_{metric}'].std()
        print(f"{metric.capitalize()}: {mean_score:.3f} ± {std_score:.3f}")
```

```
Cross Validation:

XGBoost (Random State 42):
Accuracy: 0.993 ± 0.012
Precision: 0.984 ± 0.027
Recall: 1.000 ± 0.000
F1: 0.992 ± 0.014
Roc_auc: 0.997 ± 0.004

LightGBM (Random State 52):
Accuracy: 0.993 ± 0.009
Precision: 0.984 ± 0.020
Recall: 1.000 ± 0.000
F1: 0.992 ± 0.010
Roc_auc: 0.998 ± 0.003

CatBoost (Random State 62):
Accuracy: 0.993 ± 0.009
Precision: 0.984 ± 0.020
Recall: 1.000 ± 0.000
F1: 0.992 ± 0.010
Roc_auc: 0.995 ± 0.006
```

[ ]:
```
from sklearn.model_selection import cross_val_score, LeaveOneOut

loo = LeaveOneOut()

print("Leave-One-Out Cross Validation:")

for i, (name, data) in enumerate(trained_models.items()):
    model = data['model']
```

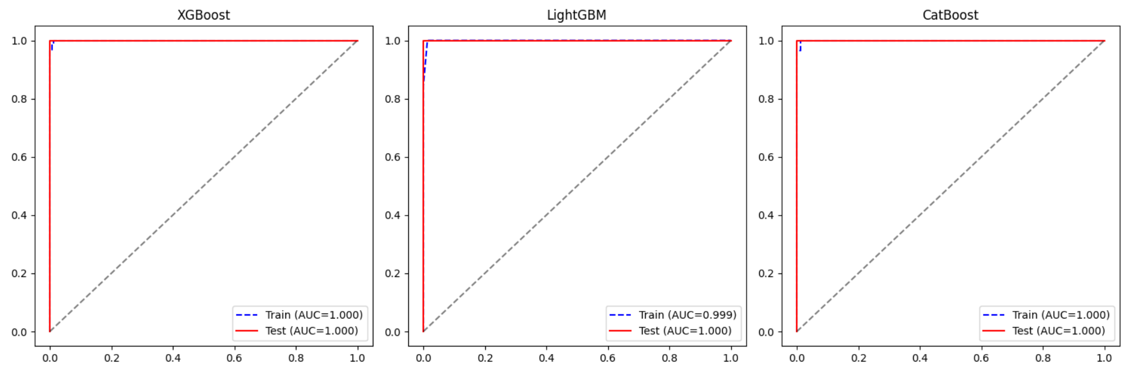
```
    # cross_val_score automatically fits the model on each train/test split
    accuracy_scores = cross_val_score(model, X_train, y_train, cv=loo,␣
 ↪scoring='accuracy')

    print(f"\n{name}:")
    print(f"Accuracy: {accuracy_scores.mean():.3f} ± {accuracy_scores.std():.
 ↪3f}")
    print(f"Individual Scores (first 10): {[f'{score:.3f}' for score in␣
 ↪accuracy_scores[:10]]}")
```

```
Leave-One-Out Cross Validation:

XGBoost:
Accuracy: 0.993 ± 0.086
Individual Scores (first 10): ['1.000', '1.000', '1.000', '1.000', '1.000',
'1.000', '1.000', '1.000', '1.000', '1.000']

LightGBM:
Accuracy: 0.993 ± 0.086
Individual Scores (first 10): ['1.000', '1.000', '1.000', '1.000', '1.000',
'1.000', '1.000', '1.000', '1.000', '1.000']

CatBoost:
Accuracy: 0.993 ± 0.086
Individual Scores (first 10): ['1.000', '1.000', '1.000', '1.000', '1.000',
'1.000', '1.000', '1.000', '1.000', '1.000']
```

##ROC Curve

[ ]:
```
from sklearn.metrics import roc_curve, auc

fig, axes = plt.subplots(1, 3, figsize=(15, 5))

for i, (name, data) in enumerate(trained_models.items()):
    # Train ROC
    y_train_proba = data['model'].predict_proba(X_train)[:, 1]
    fpr_train, tpr_train, _ = roc_curve(y_train, y_train_proba)
    axes[i].plot(fpr_train, tpr_train, 'b--', label=f'Train (AUC={auc(fpr_train,␣
 ↪tpr_train):.3f})')

    # Test ROC
    y_test_proba = data['model'].predict_proba(X_test)[:, 1]
    fpr_test, tpr_test, _ = roc_curve(y_test, y_test_proba)
    axes[i].plot(fpr_test, tpr_test, 'r-', label=f'Test (AUC={auc(fpr_test,␣
 ↪tpr_test):.3f})')

    axes[i].plot([0,1], [0,1], 'k--', alpha=0.5)
```

```
    axes[i].set_title(name)
    axes[i].legend()

plt.tight_layout()
plt.show()
```



##Confusion Matrix

[ ]:
```
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix:
 [[38  0]
 [ 0 30]]
```

##Classification Report

[ ]:
```
print("Classification Report:\n", classification_report(y_test, y_pred))
```

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

           0       1.00      1.00      1.00        38
           1       1.00      1.00      1.00        30

    accuracy                           1.00        68
   macro avg       1.00      1.00      1.00        68
weighted avg       1.00      1.00      1.00        68
```

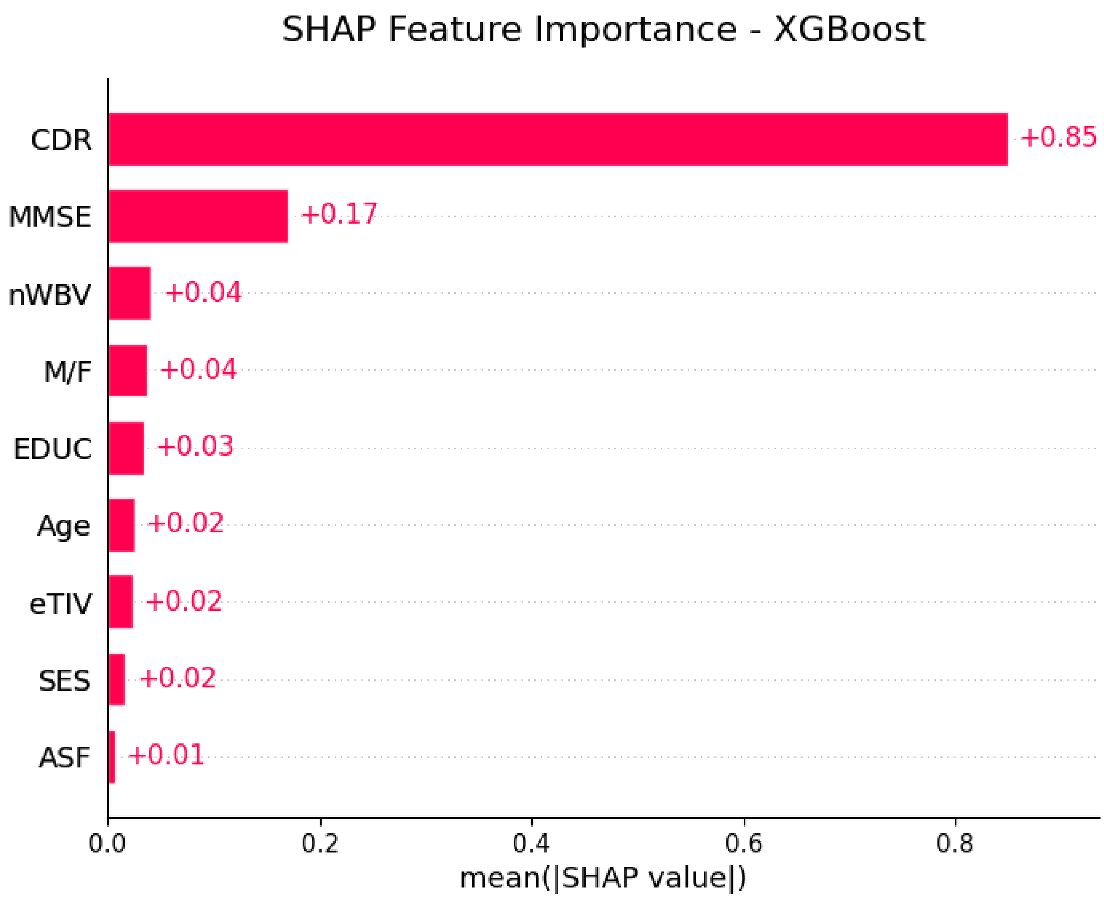
##SHAP Analysis

###SHAP Bar plot

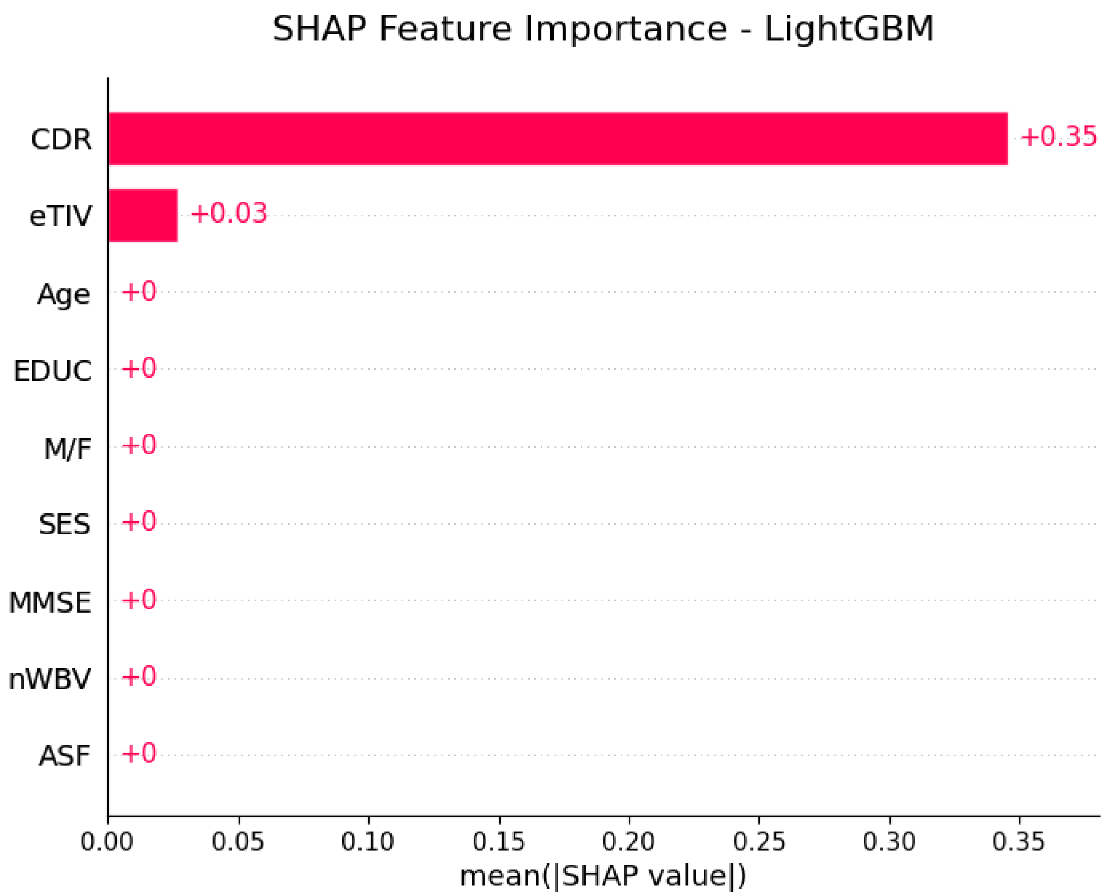
[ ]:
```
for name, data in trained_models.items():
    model = data["model"].named_steps[list(data["model"].named_steps.
 ↪keys())[-1]]
```

```
    explainer = shap.Explainer(model)
    shap_values = explainer(X_test)

    print(f"\n{name} - Best Parameters: {data['best_params']}")
    print(f"SHAP values shape: {shap_values.values.shape}")
    print(f"Data shape: {shap_values.data.shape}")

    fig = plt.figure(figsize=(10,6))
    plt.text(0.5, 1.05, f"SHAP Feature Importance - {name}", ha='center',␣
↪fontsize=16, transform=plt.gca().transAxes)

    shap.plots.bar(shap_values)
```

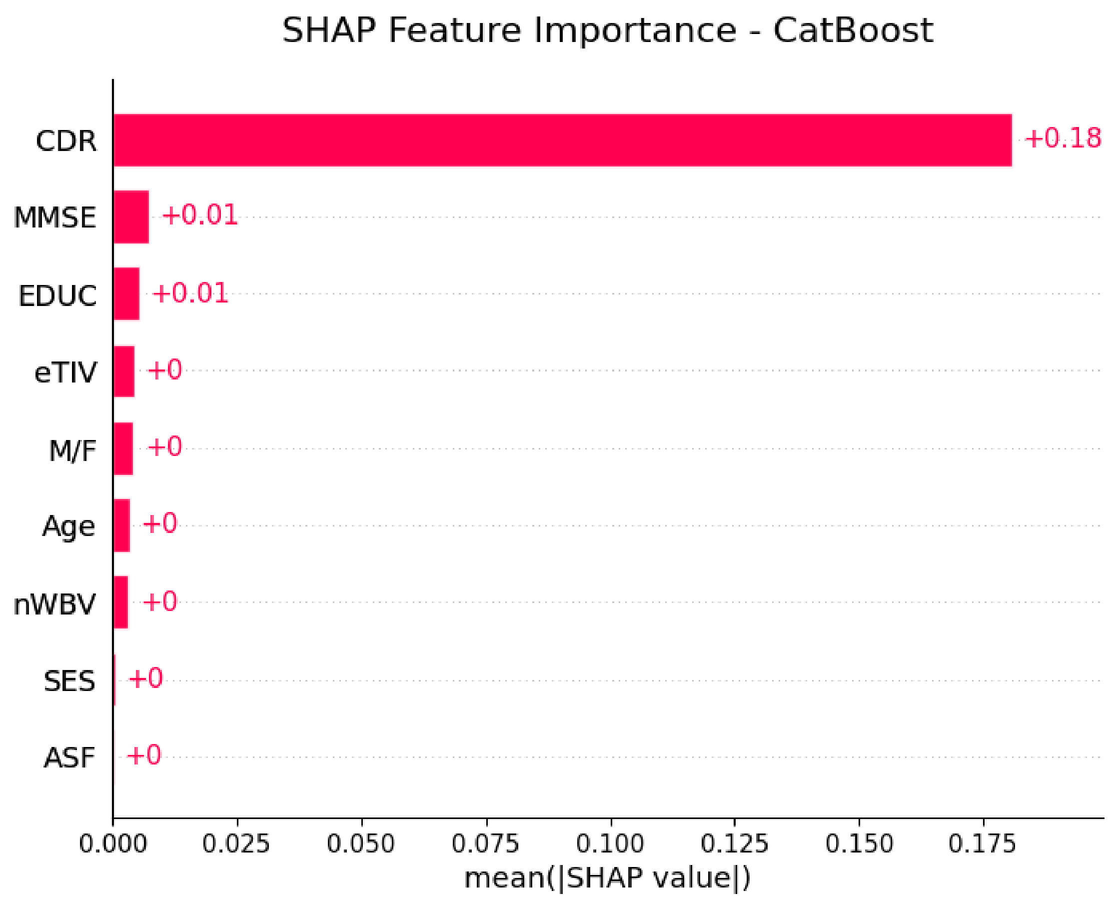
```
XGBoost - Best Parameters: {'xgbclassifier__colsample_bytree': 0.7,
'xgbclassifier__learning_rate': 0.01, 'xgbclassifier__max_depth': 3,
'xgbclassifier__min_child_weight': 1, 'xgbclassifier__n_estimators': 100,
'xgbclassifier__subsample': 0.7}
SHAP values shape: (68, 9)
Data shape: (68, 9)
```



```
LightGBM - Best Parameters: {'lgbmclassifier__learning_rate': 0.001,
'lgbmclassifier__max_depth': 3, 'lgbmclassifier__n_estimators': 200,
'lgbmclassifier__num_leaves': 31}
SHAP values shape: (68, 9)
Data shape: (68, 9)
```

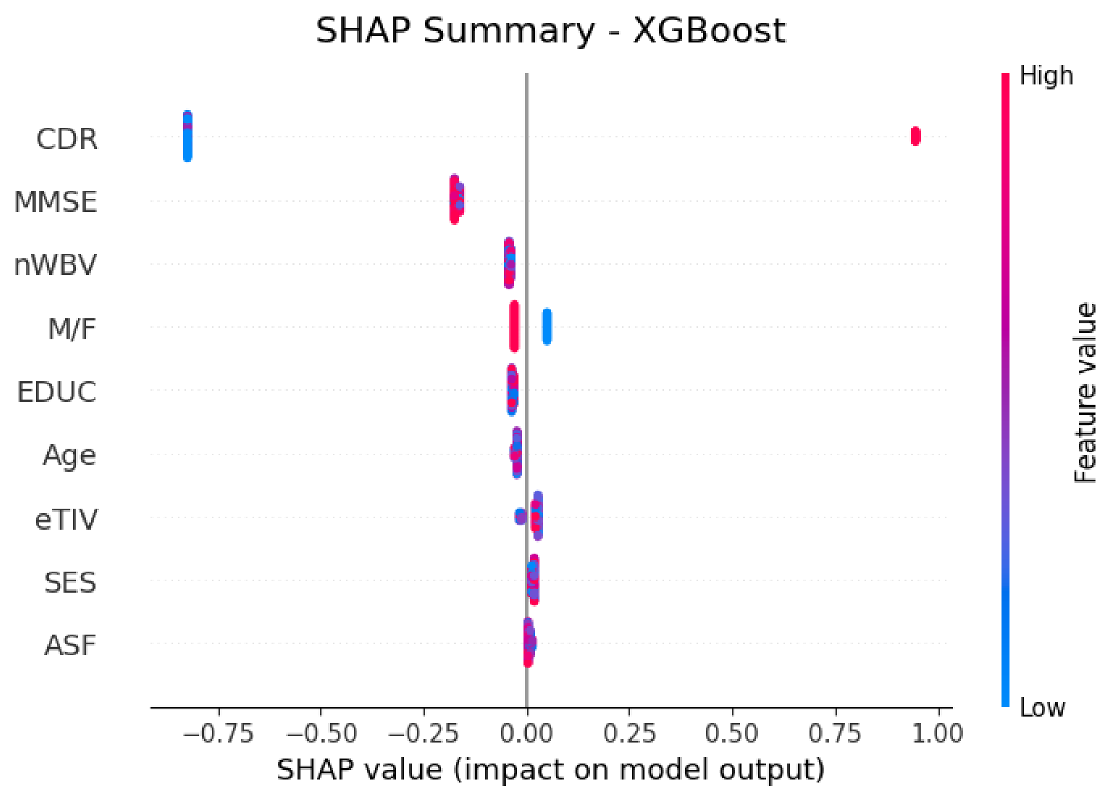


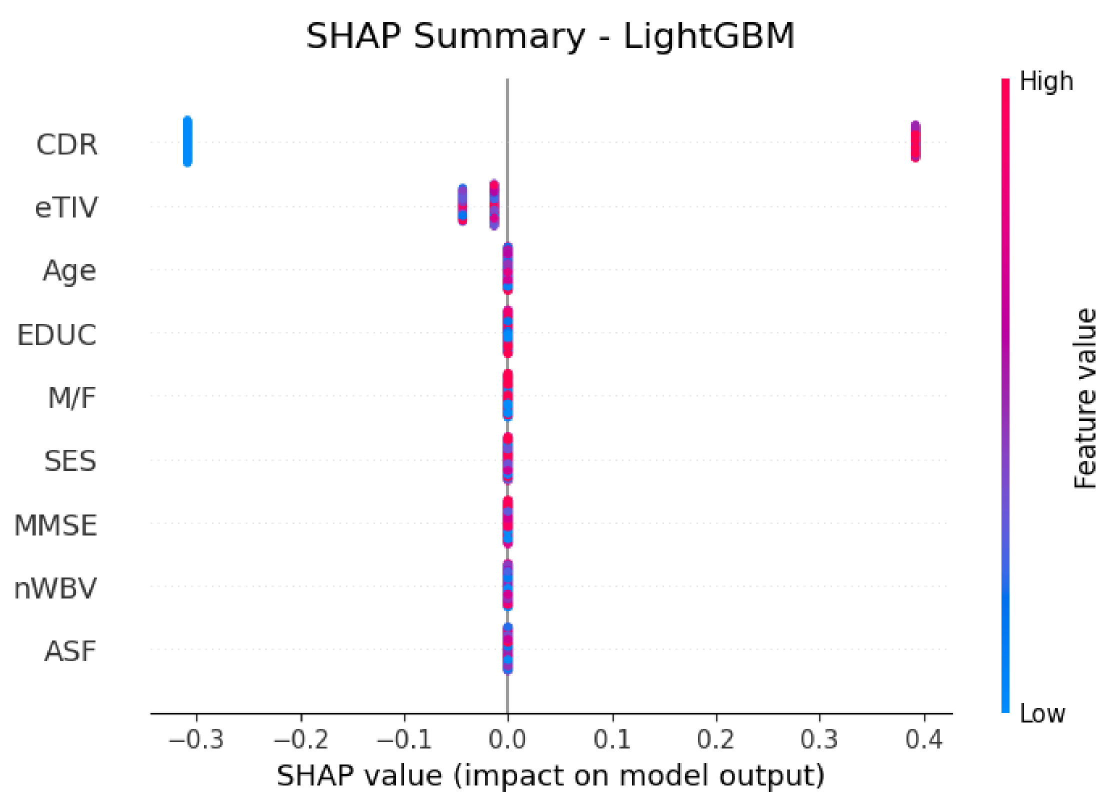

```
CatBoost - Best Parameters: {'catboostclassifier__bagging_temperature': 0,
'catboostclassifier__depth': 3, 'catboostclassifier__iterations': 100,
'catboostclassifier__l2_leaf_reg': 1, 'catboostclassifier__learning_rate':
0.001, 'catboostclassifier__min_data_in_leaf': 20, 'catboostclassifier__rsm':
0.7}
SHAP values shape: (68, 9)
Data shape: (68, 9)
```

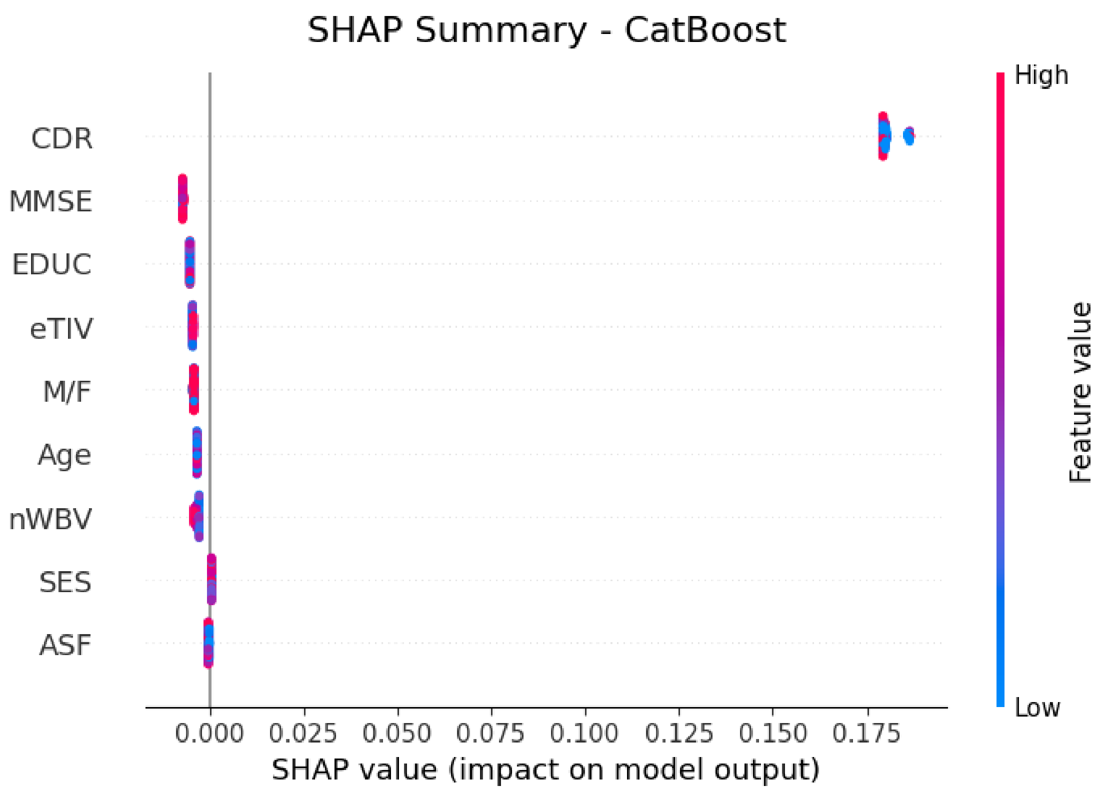


###SHAP Beeswarm plot

[ ]:
```
for name, data in trained_models.items():
    model = data["model"].named_steps[list(data["model"].named_steps.
 ↪keys())[-1]]
    explainer = shap.Explainer(model)
    shap_values = explainer(X_test)

    fig = plt.figure(figsize=(10,6))
    plt.text(0.5, 1.05, f"SHAP Summary - {name}", ha='center', fontsize=16,␣
 ↪transform=plt.gca().transAxes)

    shap.plots.beeswarm(shap_values)
```


SHAP Summary - XGBoost
CDR
MMSE
nWBV
M/F
EDUC
Age
eTIV
SES
ASF
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
SHAP value (impact on model output)
High
Low
Feature value


SHAP Summary - LightGBM
CDR
eTIV
Age
EDUC
M/F
SES
MMSE
nWBV
ASF
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
SHAP value (impact on model output)
High
Low
Feature value



7 Experiment 3 & 4 _Parallel Process(remove CDR and MMSE)

7.1 Preprocessing

[ ]:
```
df.shape
```

[ ]:
```
(336, 10)
```

[ ]:
```

```

[ ]:
```
df['Group'].value_counts()
```

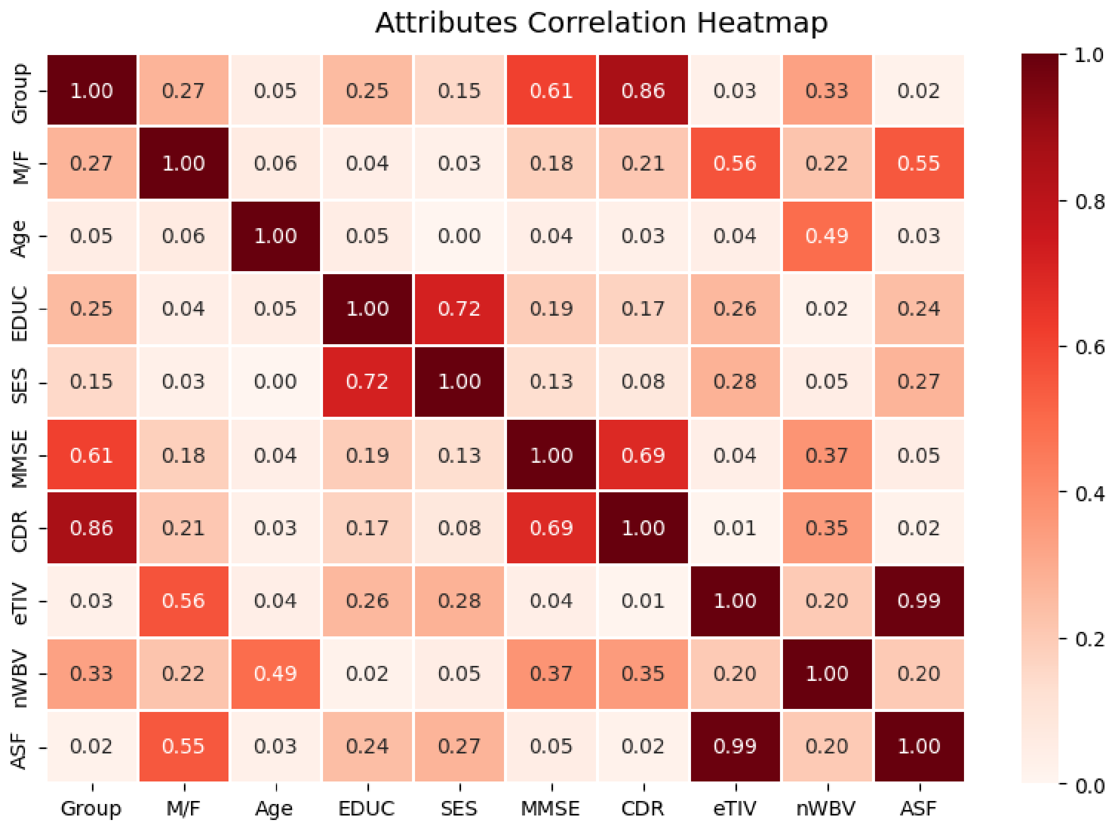
[ ]:
```
Group
0    190
1    146
Name: count, dtype: int64
```

[ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 336 entries, 0 to 372
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Group   336 non-null    int64
 1   M/F     336 non-null    int64
 2   Age     336 non-null    int64
 3   EDUC    336 non-null    int64
 4   SES     336 non-null    float64
 5   MMSE    336 non-null    float64
 6   CDR     336 non-null    float64
 7   eTIV    336 non-null    int64
 8   nWBV    336 non-null    float64
 9   ASF     336 non-null    float64
dtypes: float64(5), int64(5)
memory usage: 28.9 KB
```

[ ]:

```
# for visualizing correlations
f, ax = plt.subplots(figsize=(10, 6))
corr = df.corr().abs()
hm = sb.heatmap(round(corr,2), annot=True, ax=ax, cmap="Reds",fmt='.2f',
            linewidths=.05)
f.subplots_adjust(top=0.93)
t= f.suptitle('Attributes Correlation Heatmap', fontsize=14)
```



ASF and ETIV are strongly related with correaltion coefficient 0.99 > 0.85(threshhold), which make sense because:

eTIV Estimated Total Intracranial Volume   Template Intracranial Volume * ASF (Atlas Scaling Factor),

we can consider keep only one of them.

In previous process, we found that use MMSE or CDR to predict is a possible cheating method, so we will remove both of them in our models.

[ ]:
```
X = df.drop(['Group', 'CDR', 'MMSE', 'ASF'], axis = 1)
y = df['Group']
```

[ ]:
```
# Split dataset into training and test sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Standardize the features skip this because we prefer to see the original data␣
 ↪for better explanation
#scaler = StandardScaler()
#X_train = scaler.fit_transform(X_train)
#X_test = scaler.transform(X_test)
```

[ ]:
```
X_test.columns
```

[ ]:
```
Index(['M/F', 'Age', 'EDUC', 'SES', 'eTIV', 'nWBV'], dtype='object')
```

7.2 Model Training

7.2.1 XGBoost

[ ]:
```
# Initialize a XGBoost classifier
XGBoost= XGBClassifier(random_state=42)

# Define the parameter grid for Grid
XGB_param_dist = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.001, 0.01, 0.1],
    'subsample': [0.7, 0.8, 0.9],
    'colsample_bytree': [0.7, 0.8, 1.0],
    'min_child_weight': [1, 3, 5],
}

grid_search = GridSearchCV(XGBoost, XGB_param_dist, cv=5, scoring='f1',
 ↪n_jobs=-1)
grid_search.fit(X_train, y_train)
print("Best hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

XGBoost_mdl = grid_search.best_estimator_
y_pred = XGBoost_mdl.predict(X_test)
```

```
Best hyperparameters found by GridSearchCV:
{'colsample_bytree': 1.0, 'learning_rate': 0.1, 'max_depth': 5,
'min_child_weight': 1, 'n_estimators': 200, 'subsample': 0.9}
```

XGBoost Best hyperparameters found by GridSearchCV: {'colsample_bytree': 1.0, 'learning_rate': 0.1, 'max_depth': 5, 'min_child_weight': 1, 'n_estimators': 200, 'subsample': 0.9}

[ ]:
```
print("XGBoost: ")
print("\nClassification Report on Test Set:")
print(classification_report(y_test, y_pred))
print("-------------------------------------------------------")

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
print("ROC_AUC:", roc_auc_score(y_test, y_pred))
print("-------------------------------------------------------")
```

```python
# Get feature importances
XGBoost_feature_importances = pd.DataFrame({
    "Feature": X_test.columns,
    "Importance": XGBoost_mdl.feature_importances_
})

XGBoost_feature_importances = XGBoost_feature_importances.
 ↪sort_values(by="Importance", ascending=False)
XGBoost_feature_importances
```

```
XGBoost:

Classification Report on Test Set:
              precision    recall  f1-score   support

           0       0.85      0.87      0.86        38
           1       0.83      0.80      0.81        30

    accuracy                           0.84        68
   macro avg       0.84      0.83      0.84        68
weighted avg       0.84      0.84      0.84        68

------------------------------------------------------
Accuracy: 0.8382352941176471
Precision: 0.8275862068965517
Recall: 0.8
F1 Score: 0.8135593220338984
ROC_AUC: 0.8342105263157895
------------------------------------------------------
```

[ ]:

	Feature	Importance
2	EDUC	0.236373
0	M/F	0.216841
5	nWBV	0.152848
3	SES	0.137124
1	Age	0.131612
4	eTIV	0.125202

[ ]:

```python
xgb = XGBClassifier(random_state=42,n_jobs=-1)

param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.001, 0.01, 0.1],
    'subsample': [0.7, 0.8, 0.9],
    'colsample_bytree': [0.7, 0.8, 1.0],
```

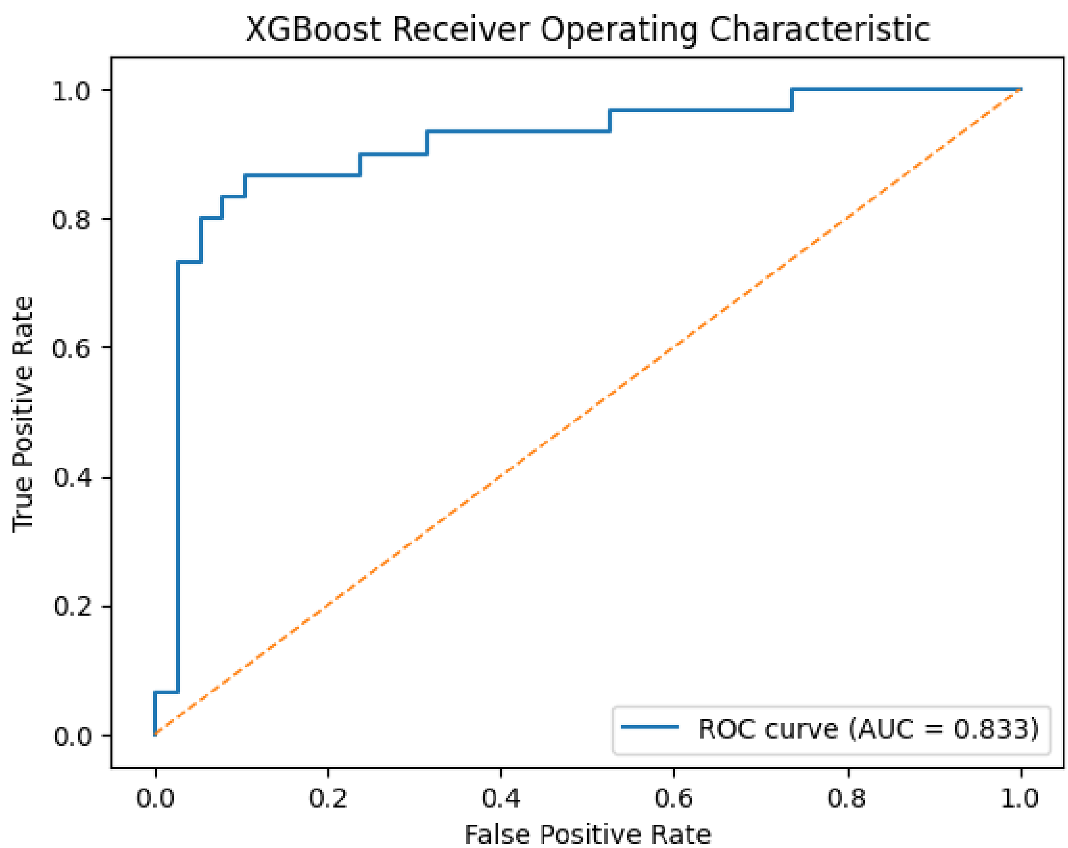
```python
    'min_child_weight': [1, 3, 5],
}

inner_cv = RepeatedStratifiedKFold(
    n_splits=5, n_repeats=2, random_state=42
)

grid = GridSearchCV(
    estimator=xgb,
    param_grid=param_grid,
    scoring='f1',
    cv=inner_cv,
    n_jobs=-1,
    verbose=0
)

grid.fit(X_train, y_train)
best_model = grid.best_estimator_
print("XGBoost classifier:")
print("Best params:", grid.best_params_)

scoring = {
    'accuracy': 'accuracy',
    'precision': 'precision',
    'recall': 'recall',
    'f1': 'f1',
    'roc_auc': 'roc_auc'
}
cv_res = cross_validate(
    best_model, X_train, y_train,
    cv=inner_cv, scoring=scoring, n_jobs=-1, return_train_score=False
)

def show(name):
    vals = cv_res[f'test_{name}']
    print(f"CV {name:<9}: {vals.mean():.4f} ± {vals.std():.4f}")

print("\n=== Repeated Stratified 5×2 CV on TRAIN ===")
for m in ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']:
    show(m)

y_pred = best_model.predict(X_test)
y_prob = best_model.predict_proba(X_test)[:, 1]

acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
```

```python
f1 = f1_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_prob)

print("\n=== Final Performance on HOLD-OUT TEST ===")
print(f"Accuracy : {acc:.4f}")
print(f"Precision: {prec:.4f}")
print(f"Recall   : {rec:.4f}")
print(f"F1 Score : {f1:.4f}")
print(f"ROC_AUC  : {auc:.4f}")

print("\nClassification Report on Test Set:\n", classification_report(y_test,
  y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

feat_imp = pd.DataFrame({
    "Feature": X_train.columns,
    "Importance": best_model.feature_importances_
}).sort_values(by="Importance", ascending=False)
print("\nTop feature importances:\n", feat_imp)

XGBoost_mdl = best_model
```

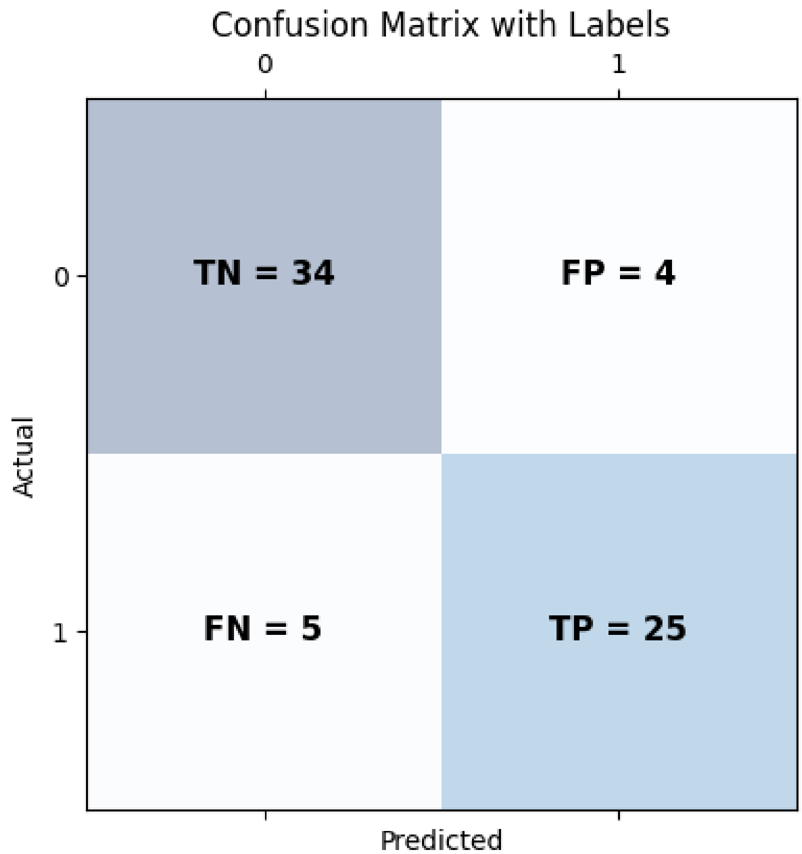
```
XGBoost classifier:
Best params: {'colsample_bytree': 0.7, 'learning_rate': 0.1, 'max_depth': 7,
'min_child_weight': 1, 'n_estimators': 200, 'subsample': 0.9}

=== Repeated Stratified 5×2 CV on TRAIN ===
CV accuracy : 0.8117 ± 0.0621
CV precision: 0.7983 ± 0.0856
CV recall   : 0.7681 ± 0.0716
CV f1       : 0.7804 ± 0.0660
CV roc_auc  : 0.8674 ± 0.0541

=== Final Performance on HOLD-OUT TEST ===
Accuracy : 0.8676
Precision: 0.8621
Recall   : 0.8333
F1 Score : 0.8475
ROC_AUC  : 0.9123

Classification Report on Test Set:
               precision    recall  f1-score   support

           0       0.87      0.89      0.88        38
           1       0.86      0.83      0.85        30

    accuracy                           0.87        68
```

```
   macro avg       0.87      0.86      0.87        68
weighted avg       0.87      0.87      0.87        68

Confusion Matrix:
 [[34  4]
 [ 5 25]]

Top feature importances:
   Feature  Importance
0      M/F    0.263357
2     EDUC    0.227721
4     eTIV    0.147784
5     nWBV    0.137234
3      SES    0.121665
1      Age    0.102240
```

[ ]:
```
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = recall_score(y_test, y_pred)

plt.figure()
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc_auc:.3f})')
plt.plot([0, 1], [0, 1], linestyle='--', linewidth=1)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('XGBoost Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()
```



[ ]:
```
cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()

labels = np.array([
    [f"TN = {tn}", f"FP = {fp}"],
    [f"FN = {fn}", f"TP = {tp}"]
])

fig, ax = plt.subplots()
ax.matshow(cm, cmap=plt.cm.Blues, alpha=0.3)
for i in range(2):
    for j in range(2):
        ax.text(j, i, labels[i, j],
                va='center', ha='center', fontsize=12, fontweight='bold')

plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix with Labels")
plt.show()
```



[ ]:
```
# Find all FN indices in the full test set
FN_all = (~y_pred) & (y_test == 1)
FN_indices = y_test[FN_all].index
print("False Negative indices:", FN_indices)

# Find all FP indices in the full test set
FP_all = (y_pred) & (y_test == 0)
FP_indices = y_test[FP_all].index
print("False Positive indices:", FP_indices)

# Find all TP indices in the full test set
TP_all = (y_pred) & (y_test == 1)
TP_indices = y_test[TP_all].index
print("True Positive indices:", TP_indices)

# Find all FN indices in the full test set
TN_all = (~y_pred) & (y_test == 0)
TN_indices = y_test[TN_all].index
print("True Negative indices:", TN_indices)
```

```
False Negative indices: Index([172, 52, 300, 299, 94], dtype='int64')
False Positive indices: Index([146, 198, 130, 64], dtype='int64')
True Positive indices: Index([124, 332, 250, 317, 154,  25,  90, 106, 285,  87,
215, 127,   3, 239,
       162, 345,  72,  39,  89,  51,  88,  16, 329, 365, 275],
      dtype='int64')
True Negative indices: Index([ 84, 122, 311,  48, 336, 213,   9, 210, 167, 113,
85, 363,  66,   5,
       153, 291, 370, 158, 102, 243, 132, 306, 337, 196, 351, 179, 199, 309,
       197, 362,   7, 209, 333,  96],
      dtype='int64')
```

[ ]:
```
FN_sample_test_idx = X_test.index.get_indexer_for([172, 52, 300, 299, 94])
FP_sample_test_idx = X_test.index.get_indexer_for([146, 198, 130, 64])
TP_sample_test_idx = X_test.index.get_indexer_for([124, 332, 250, 317, 154, 
 ↪25,  90, 106, 285,  87, 215, 127,   3, 239,
       162, 345,  72,  39,  89,  51,  88,  16, 329, 365, 275])
TN_sample_test_idx = X_test.index.get_indexer_for([ 84, 122, 311,  48, 336,
 ↪213,   9, 210, 167, 113,  85, 363,  66,   5,
       153, 291, 370, 158, 102, 243, 132, 306, 337, 196, 351, 179, 199, 309,
       197, 362,   7, 209, 333,  96])
```

[ ]:
```
print("FN_sample_test_idx: ", FN_sample_test_idx)
print("FP_sample_test_idx: ", FP_sample_test_idx)
print("TP_sample_test_idx: ", TP_sample_test_idx)
print("TN_sample_test_idx: ", TN_sample_test_idx)
```

```
FN_sample_test_idx:  [16 30 31 37 61]
FP_sample_test_idx:  [21 42 50 60]
TP_sample_test_idx:  [ 1  2  6  7  9 11 13 14 20 22 28 29 33 35 36 39 40 41 47
49 55 56 64 65
 67]
TN_sample_test_idx:  [ 0  3  4  5  8 10 12 15 17 18 19 23 24 25 26 27 32 34 38
43 44 45 46 48
 51 52 53 54 57 58 59 62 63 66]
```

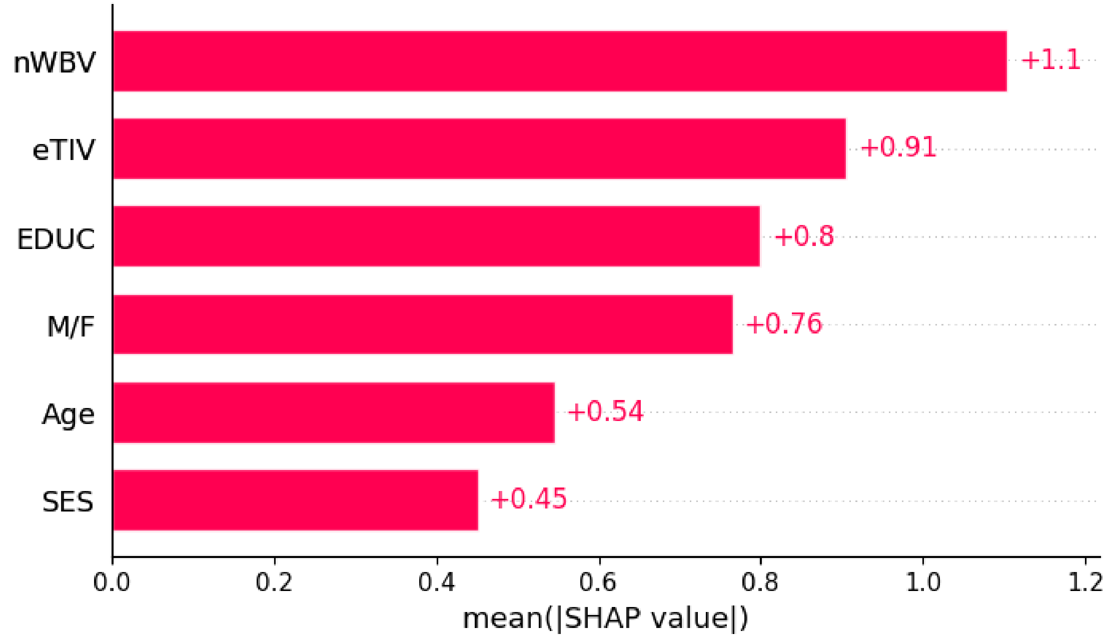
[ ]:
```
XGB_explainer = shap.Explainer(XGBoost_mdl)
XGB_shap = XGB_explainer(X_test)
print(type(XGB_explainer))
```

```
<class 'shap.explainers._tree.TreeExplainer'>
```

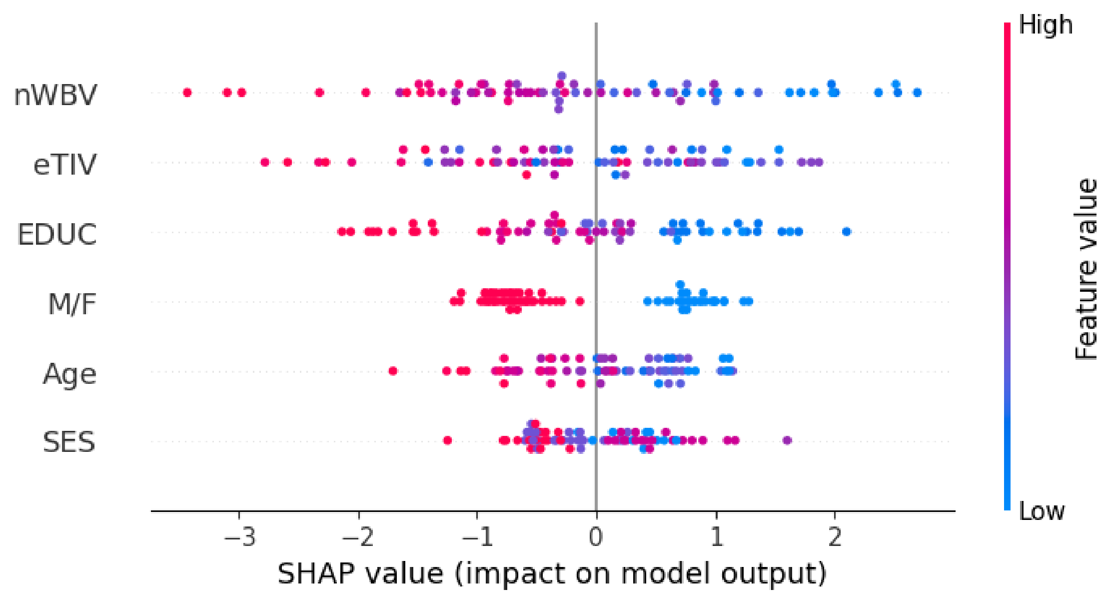
[ ]:
```
print("Values dimensions: %s" % (XGB_shap.values.shape,))
print("Data dimensions:   %s" % (XGB_shap.data.shape,))
```

```
Values dimensions: (68, 6)
Data dimensions:   (68, 6)
```

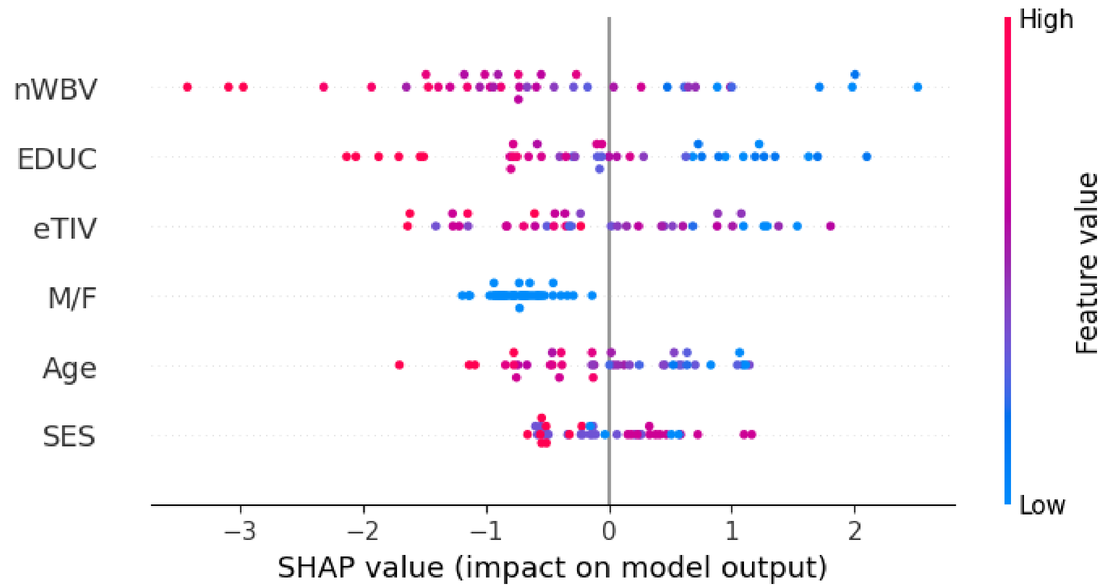
```
[ ]: sb.reset_orig()
     shap.plots.bar(XGB_shap)
```

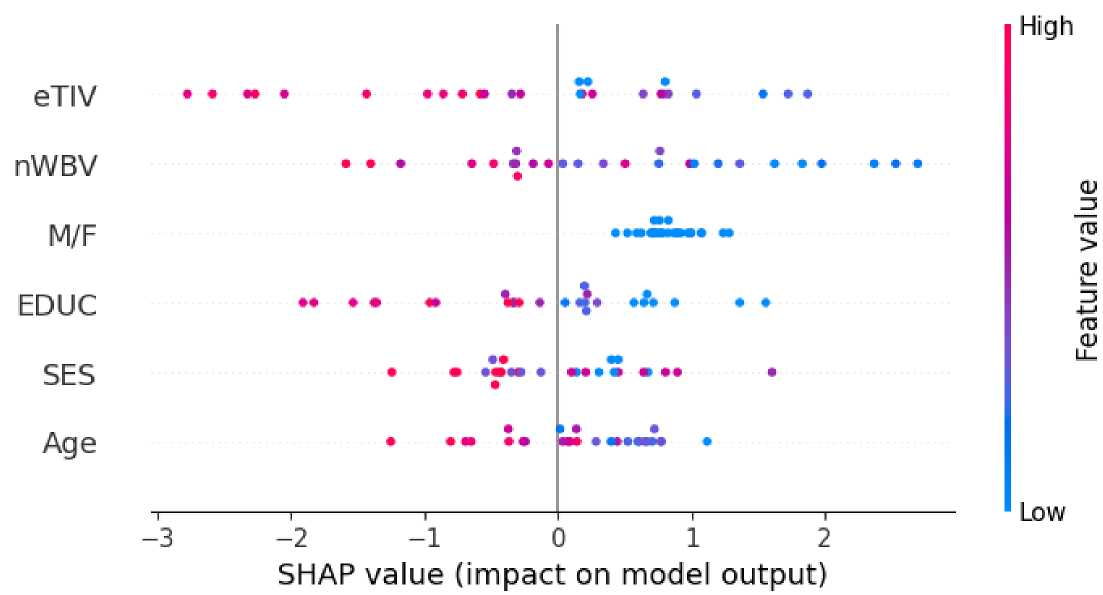



```
[ ]: shap.plots.beeswarm(XGB_shap)
```



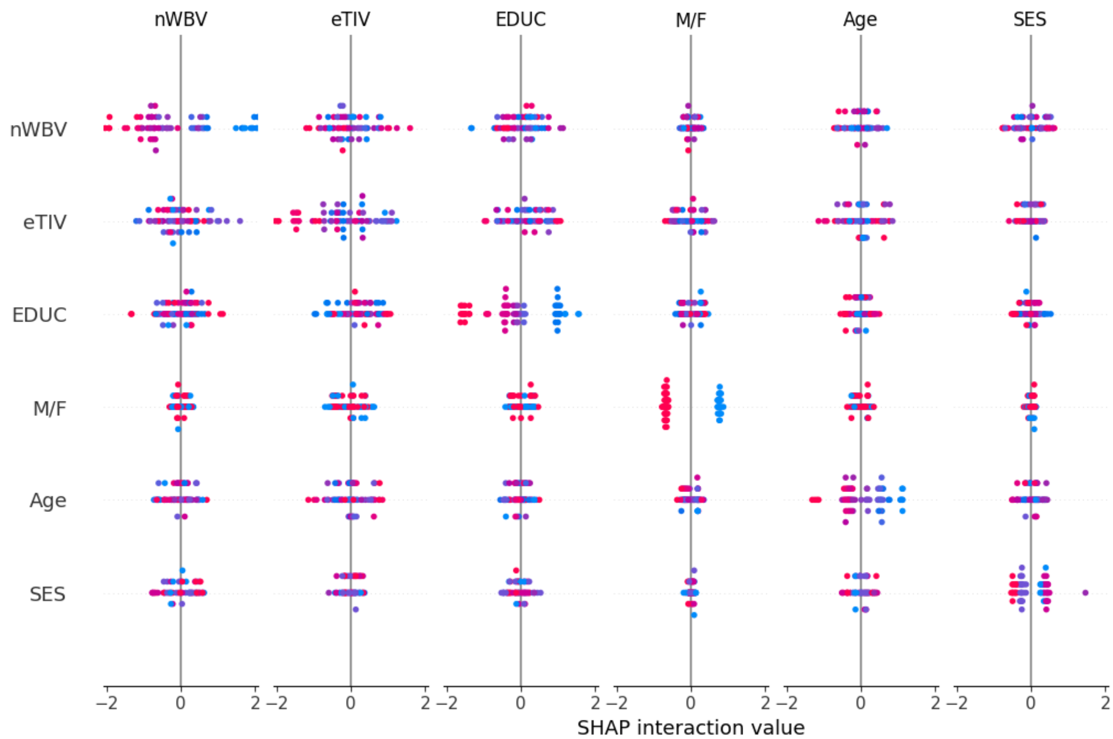
[ ]:
```
mask_f = (X_test['M/F'].values == 1)
mask_m = (X_test['M/F'].values == 0)

shap.plots.beeswarm(XGB_shap[mask_f], show=True)
shap.plots.beeswarm(XGB_shap[mask_m], show=True)
```





From the shap plot above, we can see that M/F(M=0, F=1) female and older age tend to push class to nondemented side.

[ ]:
```python
# Compute SHAP interaction values
XGB_shap_interaction_values = XGB_explainer.shap_interaction_values(X_test)

# Visualize pairwise interactions (summary plot)
shap.summary_plot(XGB_shap_interaction_values, X_test)
```



[ ]:
```python
interaction_values = XGB_shap_interaction_values
features = X_test.columns

mean_abs_interactions = np.abs(interaction_values).mean(axis=0)

# Absolute mean measures how strong the interaction is, regardless of direction.

interaction_df = pd.DataFrame(mean_abs_interactions, index=features,␣
 ↪columns=features)
interaction_df = interaction_df.where(np.triu(np.ones(interaction_df.shape),␣
 ↪k=1).astype(bool))

interaction_df = interaction_df.stack().reset_index()
interaction_df.columns = ['Feature 1', 'Feature 2', 'Mean |Interaction Value|']
interaction_df = interaction_df.sort_values(by='Mean |Interaction Value|',␣
 ↪ascending=False)
```

```
print(interaction_df.head(10))
```

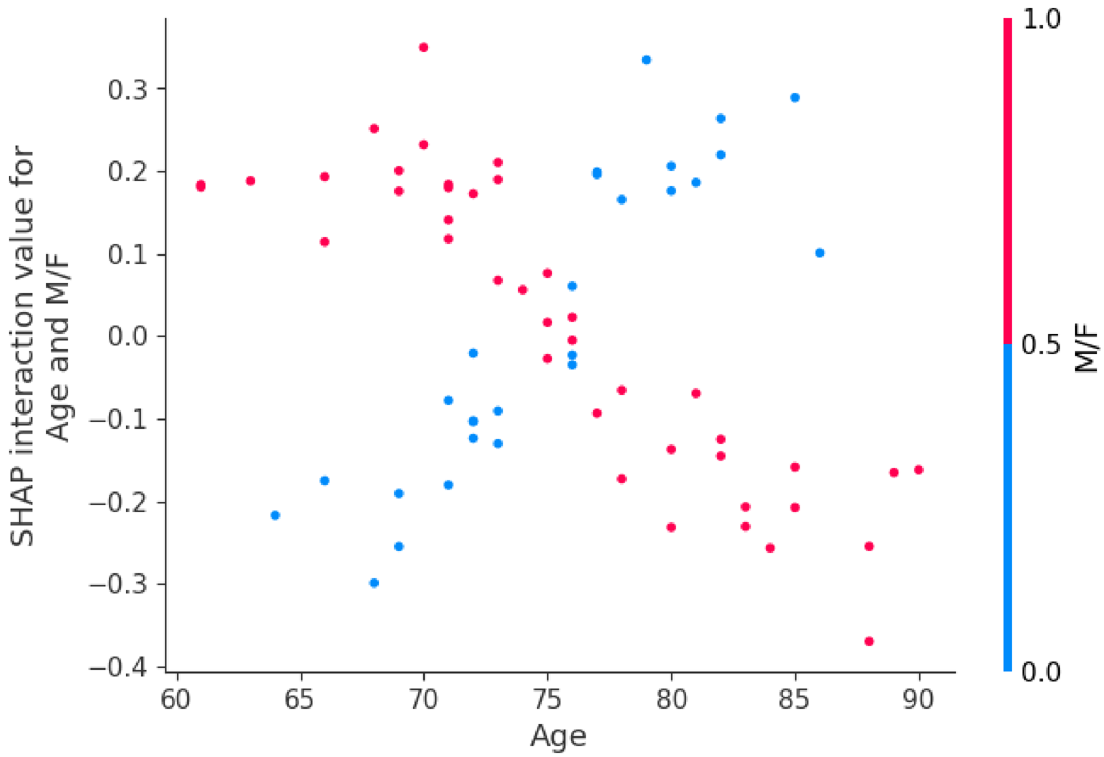
```
   Feature 1 Feature 2  Mean |Interaction Value|
10      EDUC      eTIV                  0.220816
14      eTIV      nWBV                  0.212721
11      EDUC      nWBV                  0.188962
13       SES      nWBV                  0.167425
7        Age      eTIV                  0.163801
3        M/F      eTIV                  0.142362
8        Age      nWBV                  0.137670
1        M/F      EDUC                  0.107418
6        Age       SES                  0.099767
9       EDUC       SES                  0.098318
```

[ ]:
```
# Age and Sex interation
# Signed mean measures the average directional effect of the interaction
# - positive means on average Age × M/F pushes toward demented,
#- negative means pushes toward nondemented.

age_idx = X_test.columns.get_loc('Age')
gender_idx = X_test.columns.get_loc('M/F')

interaction_values = XGB_shap_interaction_values[age_idx][:, gender_idx]
print("Mean interaction Age × Gender =", interaction_values.mean())
```

```
Mean interaction Age × Gender = 0.206249
```

Positive mean SHAP interaction age * gender value(0.20161696) represnts gender from male to female(0->1) will improve age's contribution to predict the probability of dementia.

[ ]:
```
shap.dependence_plot(
    ('Age', 'M/F'),
    XGB_shap_interaction_values,
    X_test
)
```



SHAP interaction value for Age and M/F: Represents the contribution of the interaction between age and gender to predicting the probability of dementia.

Observation:

We obeserved a cross-over patten (points of different colors intersect across age ranges) in the SHAP interaction value for Age and M/F(red: female; blue: male)

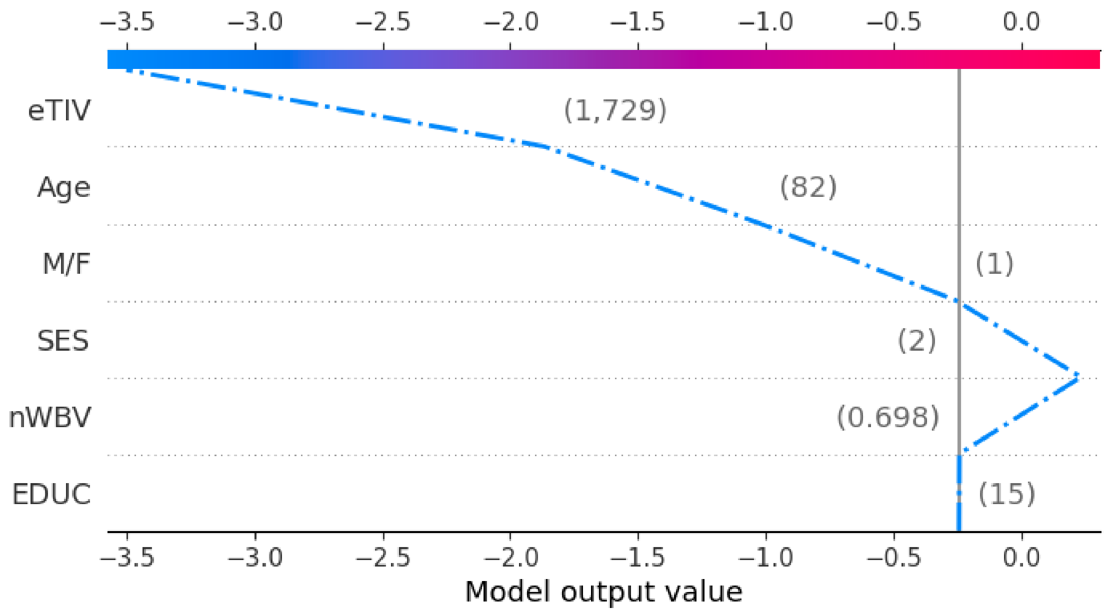
- Younger ages (60-70): More red points are positive, more blue points are negative → women tend to push toward demented group; men tend to push toward nondemented group
- Mid-range ages (around 75 years): Interaction values are close to 0 → Gender has little to no influence on the effect of age.
- Older ages (80+ years): Blue points (male) are mostly positive, red points (female) are mostly negative → For older men, increasing age tends to push toward dementia; for older women, it tends to reduce the risk.
- Direction reversal: Across different age ranges, the direction of gender's influence on the age effect flips. This is one of the main reasons why, in our earlier global SHAP analysis, the effects of Age and Sex appeared contrary to expectations — the model has learned locally reversed patterns.

Possible reasons: * Gender proportions differ significantly across age groups in the dataset.

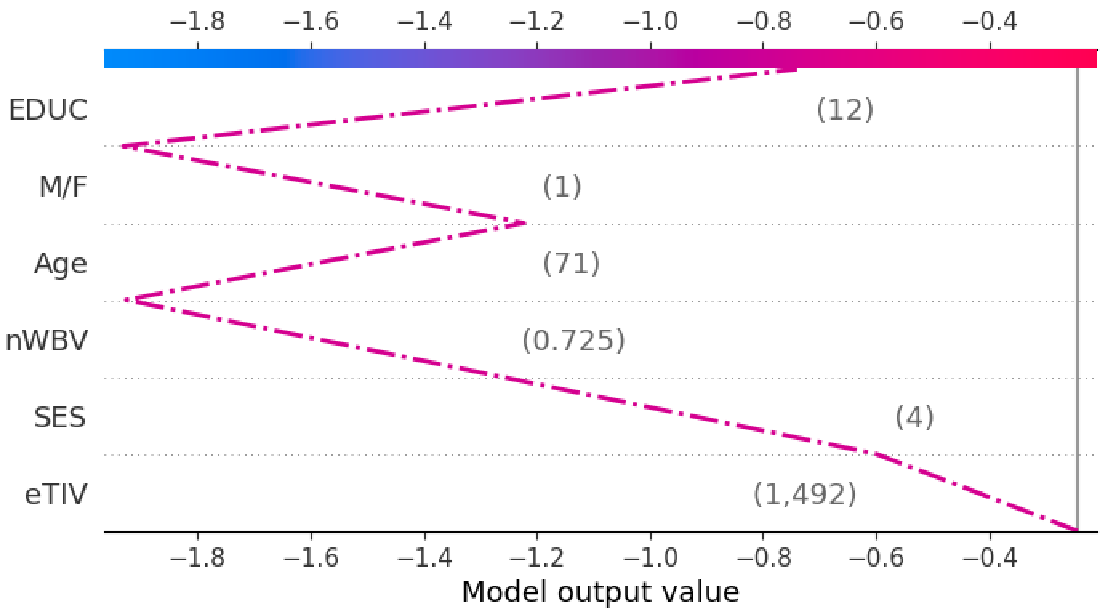
- In the oldest age group, more surviving women are healthy, so "older women" are more likely to be nondemented in the data, leading the model to learn a reversed association.

- The number of male samples is relatively small(M 147: F 189), and for certain age ranges, limited data leads the model to fit local patterns.

```
[ ]: # FN -31
expected_value = XGB_explainer.expected_value
shap.decision_plot(expected_value, XGB_shap.values[31], X_test.iloc[31],␣
 ↪highlight=0)
```




```
[ ]: # FN -16
expected_value = XGB_explainer.expected_value
shap.decision_plot(expected_value, XGB_shap.values[16], X_test.iloc[16],␣
 ↪highlight=0)
```




```
[ ]: # FN -31
shap.initjs()
expected_value = XGB_explainer.expected_value
shap.force_plot(expected_value, XGB_shap.values[31], X_test.iloc[31])
```

```
<IPython.core.display.HTML object>
```

```
[ ]: <shap.plots._force.AdditiveForceVisualizer at 0x7f38fc56a690>
```

```
[ ]: # FN -16
shap.initjs()
expected_value = XGB_explainer.expected_value
shap.force_plot(expected_value, XGB_shap.values[16], X_test.iloc[16])
```

```
<IPython.core.display.HTML object>
```

```
[ ]: <shap.plots._force.AdditiveForceVisualizer at 0x7f39207ec610>
```

```
[ ]: lime_XGB_explainer = lime.lime_tabular.LimeTabularExplainer(X_test.values,
                                                      feature_names=X_test.columns,
                                                      class_names=['Nondemented',␣
 ↪'Demented'])
```

```
[ ]: # FN case -31
lime_XGB_explainer.explain_instance(X_test.iloc[31].values,\
                                    XGBoost_mdl.predict_proba,\
                                    num_features=6).\
```

```
                                show_in_notebook(predict_proba=True)
```

```
<IPython.core.display.HTML object>
```

[ ]:
```
# FN case-16
lime_XGB_explainer.explain_instance(X_test.iloc[16].values,\
                                    XGBoost_mdl.predict_proba,\
                                    num_features=6).\
                                show_in_notebook(predict_proba=True)
```

```
<IPython.core.display.HTML object>
```

[ ]:
```
# store all fn_results
fn_results = []
feature_counter = Counter()

FN_indices = [16, 30, 31, 37, 61]

for fn_idx in FN_indices:
    instance_values = X_test.iloc[fn_idx].values

    exp = lime_XGB_explainer.explain_instance(
        instance_values,
        XGBoost_mdl.predict_proba,
        num_features=6
    )

    exp_list = exp.as_list()

    pushed_non = [f for f, w in exp_list if w < 0]
    pushed_dem = [f for f, w in exp_list if w > 0]

    fn_results.append({
        'Index': fn_idx,
        'Pushed_Nondemented': pushed_non,
        'Pushed_Demented': pushed_dem
    })

    feature_counter.update(pushed_non)

fn_df = pd.DataFrame(fn_results)

top_causes = pd.DataFrame(feature_counter.most_common(), columns=['Feature',␣
 ↪'Count'])

print(fn_df.head())
```

```
print("\n=== False Negative Feature Frequency ===")
print(top_causes)
```

```
   Index                                  Pushed_Nondemented  \
0     16  [0.00 < M/F <= 1.00, SES > 3.25, 1491.50 < eTI…
1     30                       [EDUC > 16.25, SES <= 2.00]
2     31  [eTIV > 1669.00, 0.00 < M/F <= 1.00, Age > 80…
3     37  [0.00 < M/F <= 1.00, eTIV > 1669.00, 75.00 < A…
4     61  [0.00 < M/F <= 1.00, Age > 80.25, 0.73 < nWBV …

                                     Pushed_Demented
0  [EDUC <= 12.00, Age <= 71.00, 0.70 < nWBV <= 0…
1  [M/F <= 0.00, Age <= 71.00, 0.70 < nWBV <= 0.7…
2            [nWBV <= 0.70, 12.00 < EDUC <= 15.00]
3            [nWBV <= 0.70, 12.00 < EDUC <= 15.00]
4  [1391.00 < eTIV <= 1491.50, 12.00 < EDUC <= 15…

=== False Negative Feature Frequency ===
                    Feature  Count
0         0.00 < M/F <= 1.00      4
1                SES <= 2.00      4
2             eTIV > 1669.00      2
3                Age > 80.25      2
4                 SES > 3.25      1
5  1491.50 < eTIV <= 1669.00      1
6               EDUC > 16.25      1
7        75.00 < Age <= 80.25      1
8         0.73 < nWBV <= 0.76      1
```

7.2.2 LightGBM

[ ]:
```
# Initialize a LightGBM classifier
LightGBM= LGBMClassifier(random_state=42, verbosity=-1)

# Define the parameter grid for Grid
LightGBM_param_dist = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.001, 0.01, 0.1],
    'num_leaves': [31, 50, 70],
}

grid_search = GridSearchCV(LightGBM, LightGBM_param_dist, cv=5,␣
 ↪scoring='recall', n_jobs=-1)
grid_search.fit(X_train, y_train)
print("Best hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```

```
LightGBM_mdl = grid_search.best_estimator_
y_pred = LightGBM_mdl.predict(X_test)
```

```
Best hyperparameters found by GridSearchCV:
{'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 200, 'num_leaves': 31}
```

LightGBM Best hyperparameters found by GridSearchCV: {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 200, 'num_leaves': 31}

[ ]:
```
print("LightGBM: ")
print("\nClassification Report on Test Set:")
print(classification_report(y_test, y_pred))
print("-------------------------------------------------------")

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
print("ROC_AUC:", roc_auc_score(y_test, y_pred))
print("-------------------------------------------------------")

# Get feature importances
feature_importances = pd.DataFrame({
    "Feature": X_test.columns,
    "Importance": LightGBM_mdl.feature_importances_
})

feature_importances = feature_importances.sort_values(by="Importance",␣
 ↪ascending=False)
feature_importances
```

```
LightGBM:

Classification Report on Test Set:
              precision    recall  f1-score   support

           0       0.89      0.87      0.88        38
           1       0.84      0.87      0.85        30

    accuracy                           0.87        68
   macro avg       0.87      0.87      0.87        68
weighted avg       0.87      0.87      0.87        68

-------------------------------------------------------
Accuracy: 0.8676470588235294
Precision: 0.8387096774193549
Recall: 0.8666666666666667
```

```
F1 Score: 0.8524590163934426
ROC_AUC: 0.8675438596491228
------------------------------------------------------
```

[ ]:

	Feature	Importance
4	eTIV	610
5	nWBV	478
1	Age	332
2	EDUC	215
3	SES	177
0	M/F	68

[ ]:

```python
LightGBM = LGBMClassifier(random_state=42, verbosity=-1)

param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.001, 0.01, 0.1],
    'num_leaves': [31, 50, 70],
}

inner_cv = RepeatedStratifiedKFold(
    n_splits=5, n_repeats=2, random_state=42
)

grid = GridSearchCV(
    estimator=LightGBM,
    param_grid=param_grid,
    scoring='f1',
    cv=inner_cv,
    n_jobs=-1,
    verbose=0
)

grid.fit(X_train, y_train)
best_model = grid.best_estimator_
print("LightGBM classifier:")
print("Best params:", grid.best_params_)

scoring = {
    'accuracy': 'accuracy',
    'precision': 'precision',
    'recall': 'recall',
    'f1': 'f1',
    'roc_auc': 'roc_auc'
}
cv_res = cross_validate(
```

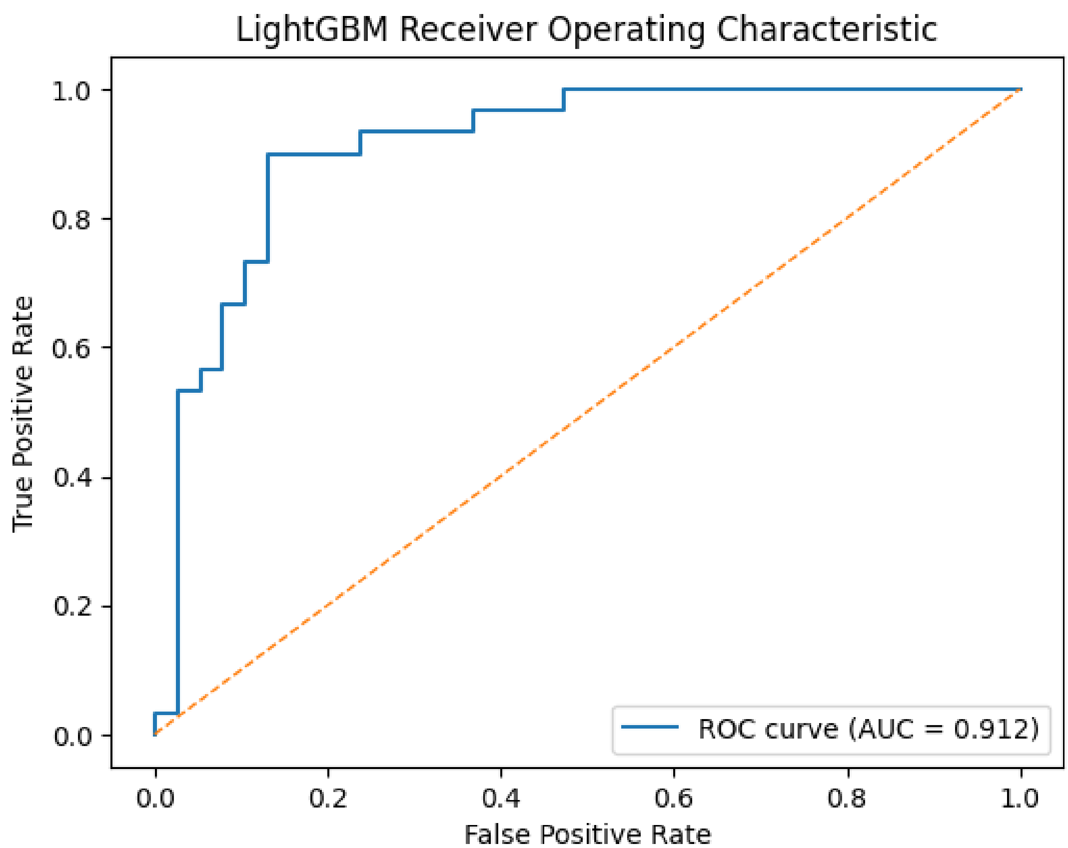
```
    best_model, X_train, y_train,
    cv=inner_cv, scoring=scoring, n_jobs=-1, return_train_score=False
)

def show(name):
    vals = cv_res[f'test_{name}']
    print(f"CV {name:<9}: {vals.mean():.4f} ± {vals.std():.4f}")

print("\n=== Repeated Stratified 5×2 CV on TRAIN ===")
for m in ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']:
    show(m)

y_pred = best_model.predict(X_test)
y_prob = best_model.predict_proba(X_test)[:, 1]

acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_prob)

print("\n=== Final Performance on HOLD-OUT TEST ===")
print(f"Accuracy : {acc:.4f}")
print(f"Precision: {prec:.4f}")
print(f"Recall   : {rec:.4f}")
print(f"F1 Score : {f1:.4f}")
print(f"ROC_AUC  : {auc:.4f}")

print("\nClassification Report on Test Set:\n", classification_report(y_test,␣
↪y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

feat_imp = pd.DataFrame({
    "Feature": X_train.columns,
    "Importance": best_model.feature_importances_
}).sort_values(by="Importance", ascending=False)
print("\nTop feature importances:\n", feat_imp)

LightGBM_mdl = best_model
```

```
LightGBM classifier:
Best params: {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 200,
'num_leaves': 31}

=== Repeated Stratified 5×2 CV on TRAIN ===
CV accuracy : 0.7969 ± 0.0714
CV precision: 0.7726 ± 0.0991
```

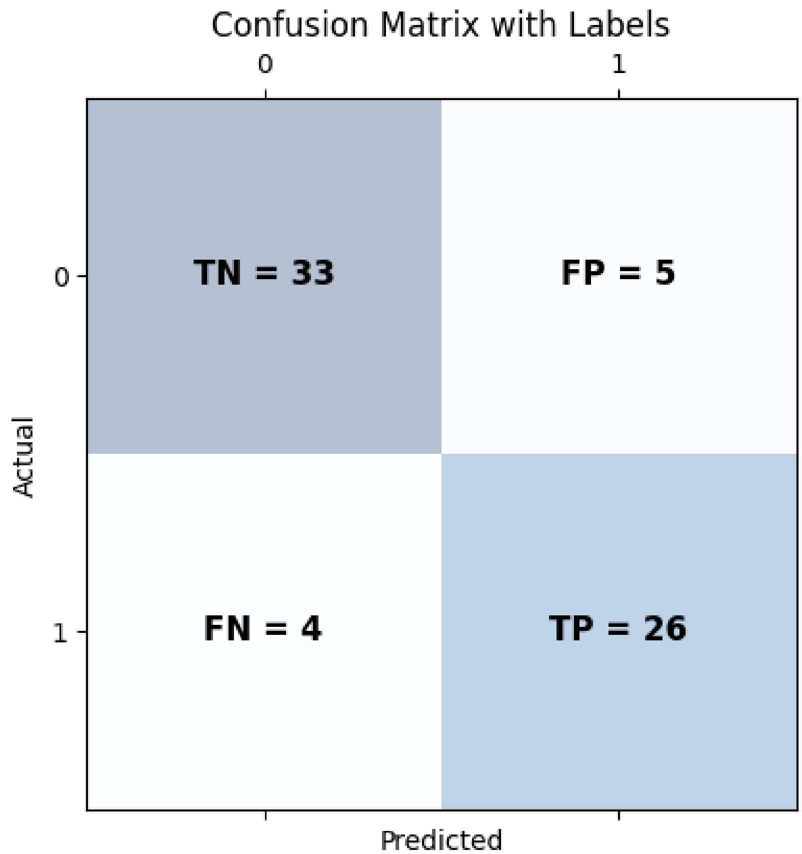
```
CV recall   : 0.7721 ± 0.0675
CV f1       : 0.7692 ± 0.0711
CV roc_auc  : 0.8443 ± 0.0613

=== Final Performance on HOLD-OUT TEST ===
Accuracy : 0.8676
Precision: 0.8387
Recall   : 0.8667
F1 Score : 0.8525
ROC_AUC  : 0.9123

Classification Report on Test Set:
              precision    recall  f1-score   support

           0       0.89      0.87      0.88        38
           1       0.84      0.87      0.85        30

    accuracy                           0.87        68
   macro avg       0.87      0.87      0.87        68
weighted avg       0.87      0.87      0.87        68

Confusion Matrix:
 [[33  5]
 [ 4 26]]

Top feature importances:
   Feature  Importance
4    eTIV         610
5    nWBV         478
1     Age         332
2    EDUC         215
3     SES         177
0     M/F          68
```

[ ]:

```
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = roc_auc_score(y_test, y_prob)

plt.figure()
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc_auc:.3f})')
plt.plot([0, 1], [0, 1], linestyle='--', linewidth=1)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('LightGBM Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()
```



[ ]:
```
cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()

labels = np.array([
    [f"TN = {tn}", f"FP = {fp}"],
    [f"FN = {fn}", f"TP = {tp}"]
])

fig, ax = plt.subplots()
ax.matshow(cm, cmap=plt.cm.Blues, alpha=0.3)
for i in range(2):
    for j in range(2):
        ax.text(j, i, labels[i, j],
                va='center', ha='center', fontsize=12, fontweight='bold')

plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix with Labels")
plt.show()
```




```python
[ ]: # Find all FN indices in the full test set
FN_all = (~y_pred) & (y_test == 1)
FN_indices = y_test[FN_all].index
print("False Negative indices:", FN_indices)
```

```
False Negative indices: Index([172, 300, 299, 94], dtype='int64')
```

```python
[ ]: FN_sample_test_idx = X_test.index.get_indexer_for([172, 300, 299, 94])
FN_sample_test_idx
```

```
[ ]: array([16, 31, 37, 61])
```

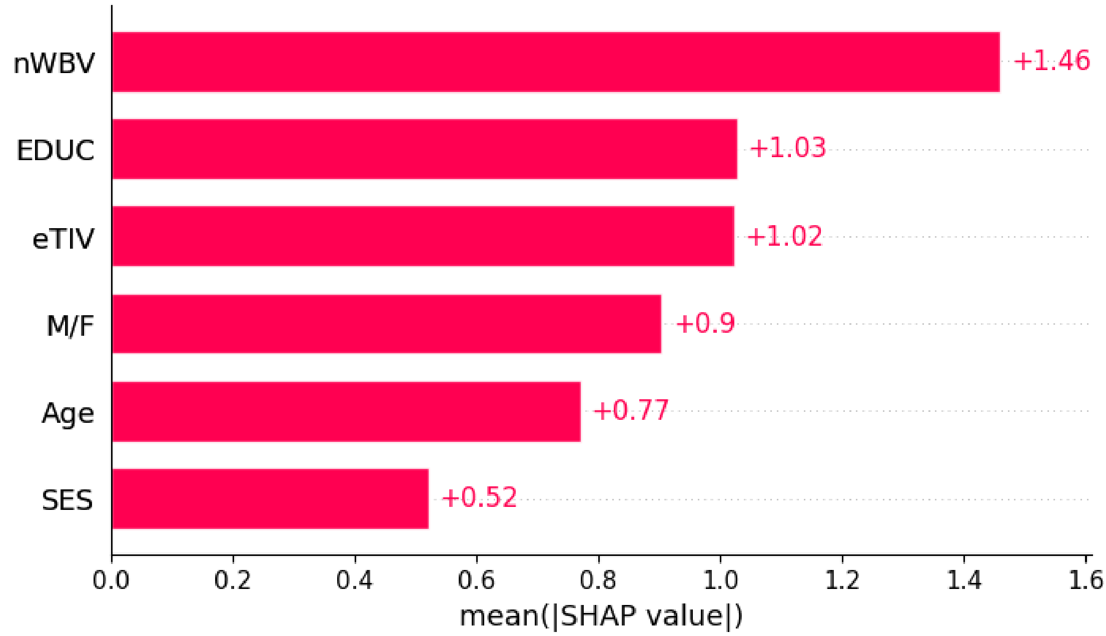
```python
[ ]: LGB_explainer = shap.Explainer(LightGBM_mdl)
LGB_shap = LGB_explainer(X_test)
print(type(LGB_explainer))
```

```
<class 'shap.explainers._tree.TreeExplainer'>
```

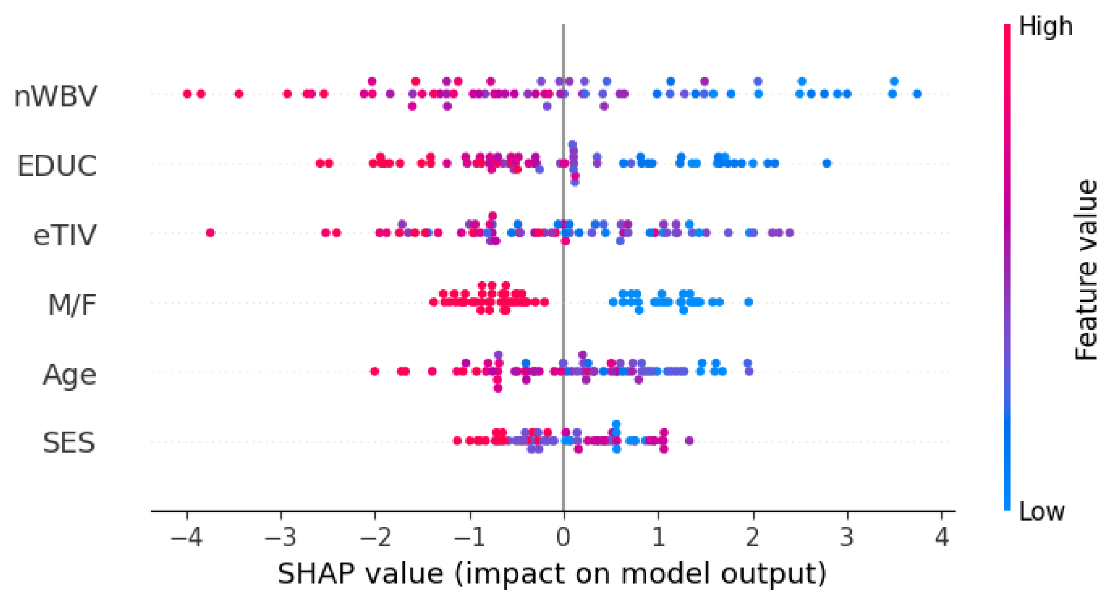
```python
[ ]: print("Values dimensions: %s" % (LGB_shap.values.shape,))
print("Data dimensions:   %s" % (LGB_shap.data.shape,))
```

```
Values dimensions: (68, 6)
Data dimensions:   (68, 6)
```

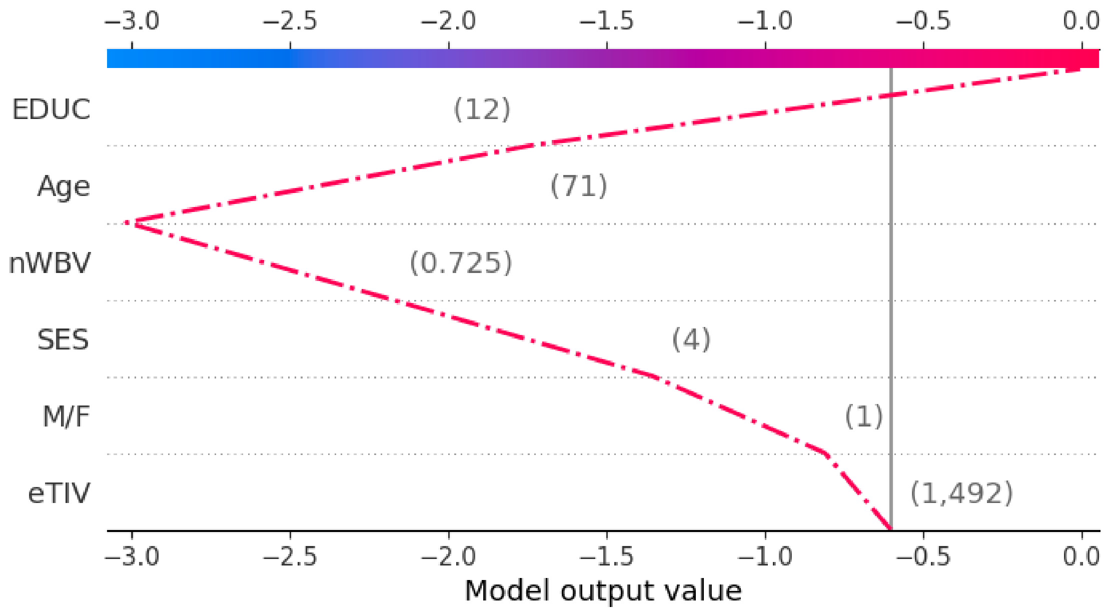
[ ]:
```
sb.reset_orig()
shap.plots.bar(LGB_shap)
```



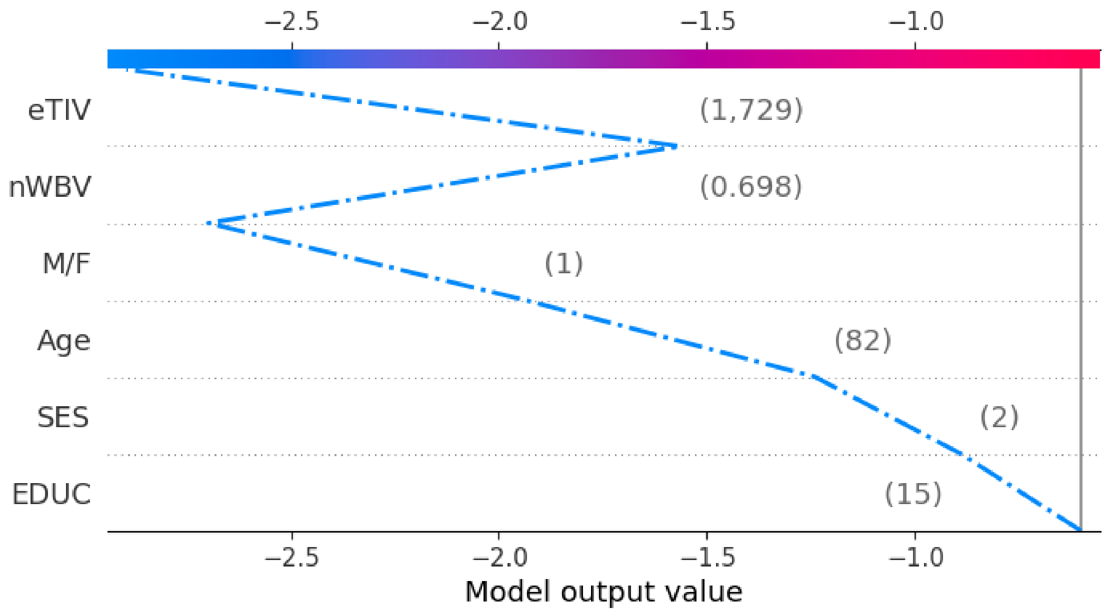
[ ]:
```
shap.plots.beeswarm(LGB_shap)
```



[ ]:
```
# LGB-FN-16
sb.reset_orig()
expected_value = LGB_explainer.expected_value
shap.decision_plot(expected_value, LGB_shap.values[16], X_test.iloc[16],
    highlight=0)
```



[ ]:
```
# LGB-FN-31
expected_value = LGB_explainer.expected_value
shap.decision_plot(expected_value, LGB_shap.values[31], X_test.iloc[31],
    highlight=0)
```




```
[ ]: # LGB-FN-16
     shap.initjs()
     expected_value = LGB_explainer.expected_value
     shap.force_plot(expected_value, LGB_shap.values[16], X_test.iloc[16])
```

```
<IPython.core.display.HTML object>
```

```
[ ]: <shap.plots._force.AdditiveForceVisualizer at 0x7f38fdbe9650>
```

```
[ ]: # LGB-FN-31
     shap.initjs()
     expected_value = LGB_explainer.expected_value
     shap.force_plot(expected_value, LGB_shap.values[31], X_test.iloc[31])
```

```
<IPython.core.display.HTML object>
```

```
[ ]: <shap.plots._force.AdditiveForceVisualizer at 0x7f391eb30f50>
```

```
[ ]: lime_LGB_explainer = lime.lime_tabular.LimeTabularExplainer(X_test.values,
                                                   feature_names=X_test.columns,
                                                   class_names=['Nondemented',␣
       ↪'Demented'])
```

```
[ ]: # LGB-FN-16
     lime_LGB_explainer.explain_instance(X_test.iloc[16].values,\
                                        LightGBM_mdl.predict_proba,\
                                        num_features=6).\
```

```
                                show_in_notebook(predict_proba=True)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739:
UserWarning: X does not have valid feature names, but LGBMClassifier was fitted
with feature names
  warnings.warn(
```

<IPython.core.display.HTML object>

[ ]:
```
# LGB-FN-31
lime_LGB_explainer.explain_instance(X_test.iloc[31].values,\
                                    LightGBM_mdl.predict_proba,\
                                    num_features=6).\
                                show_in_notebook(predict_proba=True)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739:
UserWarning: X does not have valid feature names, but LGBMClassifier was fitted
with feature names
  warnings.warn(
```

<IPython.core.display.HTML object>

[ ]:
```
# store all fn_results
fn_results = []
feature_counter = Counter()

LGB_FN_indices = [16, 31, 37, 61]

for fn_idx in LGB_FN_indices:
    LGB_instance_values = X_test.iloc[fn_idx].values

    exp = lime_LGB_explainer.explain_instance(
        LGB_instance_values,
        LightGBM_mdl.predict_proba,
        num_features=6
    )

    exp_list = exp.as_list()

    pushed_non = [f for f, w in exp_list if w < 0]
    pushed_dem = [f for f, w in exp_list if w > 0]

    fn_results.append({
        'Index': fn_idx,
        'Pushed_Nondemented': pushed_non,
        'Pushed_Demented': pushed_dem
    })
```

```python
    feature_counter.update(pushed_non)

fn_df = pd.DataFrame(fn_results)

top_causes = pd.DataFrame(feature_counter.most_common(), columns=['Feature',
 ↪'Count'])

print(fn_df.head())

print("\n=== LGB False Negative Feature Frequency ===")
print(top_causes)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739:
UserWarning: X does not have valid feature names, but LGBMClassifier was fitted
with feature names
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739:
UserWarning: X does not have valid feature names, but LGBMClassifier was fitted
with feature names
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739:
UserWarning: X does not have valid feature names, but LGBMClassifier was fitted
with feature names
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739:
UserWarning: X does not have valid feature names, but LGBMClassifier was fitted
with feature names
  warnings.warn(
```

```
   Index                                 Pushed_Nondemented  \
0     16  [0.00 < M/F <= 1.00, SES > 3.25, 1491.50 < eTI…
1     31  [eTIV > 1669.00, 0.00 < M/F <= 1.00, Age > 80…
2     37  [0.00 < M/F <= 1.00, eTIV > 1669.00, 75.00 < A…
3     61  [0.00 < M/F <= 1.00, Age > 80.25, 0.73 < nWBV …

                                   Pushed_Demented
0  [EDUC <= 12.00, Age <= 71.00, 0.70 < nWBV <= 0…
1               [nWBV <= 0.70, 12.00 < EDUC <= 15.00]
2               [nWBV <= 0.70, 12.00 < EDUC <= 15.00]
3  [1391.00 < eTIV <= 1491.50, 12.00 < EDUC <= 15…

=== LGB False Negative Feature Frequency ===
                Feature  Count
0          0.00 < M/F <= 1.00      4
1               SES <= 2.00      3
2            eTIV > 1669.00      2
3               Age > 80.25      2
4                SES > 3.25      1
```

```
5  1491.50 < eTIV <= 1669.00      1
6       75.00 < Age <= 80.25      1
7         0.73 < nWBV <= 0.76      1
```

###CatBoost

```
[ ]: # Initialize a CatBoost classifier
CatBoost = CatBoostClassifier(random_state=42, verbose = False)

# Define the parameter grid for Grid
CatBoost_param_dist = {
    'min_data_in_leaf': [20, 40, 60],
    'rsm': [0.7, 0.8, 1.0],
    'iterations': [100, 200],
    'depth': [3, 5, 7],
    'learning_rate': [0.001, 0.01, 0.1],
    'l2_leaf_reg': [1, 3, 5],
    'bagging_temperature': [0, 0.5, 1.0],
}

grid_search = GridSearchCV(CatBoost, CatBoost_param_dist, cv=5,␣
 ↪scoring='recall', n_jobs=-1)
grid_search.fit(X_train, y_train)
print("Best hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

CatBoost_mdl = grid_search.best_estimator_
y_pred = CatBoost_mdl.predict(X_test)
```

```
Best hyperparameters found by GridSearchCV:
{'bagging_temperature': 0, 'depth': 7, 'iterations': 200, 'l2_leaf_reg': 5,
'learning_rate': 0.1, 'min_data_in_leaf': 20, 'rsm': 0.8}
```

CatBoost Best hyperparameters found by GridSearchCV:

{'bagging_temperature': 0, 'depth': 7, 'iterations': 200, 'l2_leaf_reg': 5, 'learning_rate': 0.1, 'min_data_in_leaf': 20, 'rsm': 0.8}

```
[ ]: print("CatBoost: ")
print("\nClassification Report on Test Set:")
print(classification_report(y_test, y_pred))
print("------------------------------------------------------")

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
print("ROC_AUC:", roc_auc_score(y_test, y_pred))
print("------------------------------------------------------")
```

```python
# Get feature importances
feature_importances = pd.DataFrame({
    "Feature": X_test.columns,
    "Importance": CatBoost_mdl.feature_importances_
})

feature_importances = feature_importances.sort_values(by="Importance",␣
↪ascending=False)
feature_importances
```

```
CatBoost:

Classification Report on Test Set:
              precision    recall  f1-score   support

           0       0.85      0.92      0.89        38
           1       0.89      0.80      0.84        30

    accuracy                           0.87        68
   macro avg       0.87      0.86      0.86        68
weighted avg       0.87      0.87      0.87        68

------------------------------------------------------
Accuracy: 0.8676470588235294
Precision: 0.8888888888888888
Recall: 0.8
F1 Score: 0.8421052631578947
ROC_AUC: 0.8605263157894737
------------------------------------------------------
```

[ ]:

	Feature	Importance
4	eTIV	22.606796
5	nWBV	21.504276
2	EDUC	17.237605
1	Age	16.420387
3	SES	14.022414
0	M/F	8.208523

[ ]:

```python
CatBoost = CatBoostClassifier(random_state=42)

param_grid = {
    'min_data_in_leaf': [20, 40, 60],
    'rsm': [0.7, 0.8, 1.0],
    'iterations': [100, 200],
    'depth': [3, 5, 7],
    'learning_rate': [0.001, 0.01, 0.1],
```

```
    'l2_leaf_reg': [1, 3, 5],
    'bagging_temperature': [0, 0.5, 1.0],
}

inner_cv = RepeatedStratifiedKFold(
    n_splits=5, n_repeats=2, random_state=42
)

grid = GridSearchCV(
    estimator=CatBoost,
    param_grid=param_grid,
    scoring='f1',
    cv=inner_cv,
    n_jobs=-1,
    verbose=0
)

grid.fit(X_train, y_train)
best_model = grid.best_estimator_
print("CatBoostClassifier:")
print("Best params:", grid.best_params_)

scoring = {
    'accuracy': 'accuracy',
    'precision': 'precision',
    'recall': 'recall',
    'f1': 'f1',
    'roc_auc': 'roc_auc'
}
cv_res = cross_validate(
    best_model, X_train, y_train,
    cv=inner_cv, scoring=scoring, n_jobs=-1, return_train_score=False
)

def show(name):
    vals = cv_res[f'test_{name}']
    print(f"CV {name:<9}: {vals.mean():.4f} ± {vals.std():.4f}")

print("\n=== Repeated Stratified 5×2 CV on TRAIN ===")
for m in ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']:
    show(m)

y_pred = best_model.predict(X_test)
y_prob = best_model.predict_proba(X_test)[:, 1]

acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
```

```python
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_prob)

print("\n=== Final Performance on HOLD-OUT TEST ===")
print(f"Accuracy : {acc:.4f}")
print(f"Precision: {prec:.4f}")
print(f"Recall    : {rec:.4f}")
print(f"F1 Score : {f1:.4f}")
print(f"ROC_AUC   : {auc:.4f}")

print("\nClassification Report on Test Set:\n", classification_report(y_test,␣
 ↪y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

feat_imp = pd.DataFrame({
    "Feature": X_train.columns,
    "Importance": best_model.feature_importances_
}).sort_values(by="Importance", ascending=False)
print("\nTop feature importances:\n", feat_imp)

CatBoost_mdl = best_model
```

```
0:      learn: 0.6276889        total: 47.7ms   remaining: 9.5s
1:      learn: 0.5745076        total: 49ms     remaining: 4.85s
2:      learn: 0.5643154        total: 49.6ms   remaining: 3.26s
3:      learn: 0.5272631        total: 50.8ms   remaining: 2.49s
4:      learn: 0.4983803        total: 51.9ms   remaining: 2.02s
5:      learn: 0.4609295        total: 53.1ms   remaining: 1.72s
6:      learn: 0.4404177        total: 54ms     remaining: 1.49s
7:      learn: 0.4183808        total: 55.1ms   remaining: 1.32s
8:      learn: 0.3995695        total: 56.3ms   remaining: 1.2s
9:      learn: 0.3771613        total: 57.8ms   remaining: 1.1s
10:     learn: 0.3653814        total: 59.2ms   remaining: 1.02s
11:     learn: 0.3591362        total: 60.3ms   remaining: 945ms
12:     learn: 0.3424529        total: 61.4ms   remaining: 883ms
13:     learn: 0.3281506        total: 62.3ms   remaining: 828ms
14:     learn: 0.3188232        total: 63.4ms   remaining: 782ms
15:     learn: 0.3085403        total: 64.3ms   remaining: 739ms
16:     learn: 0.2969568        total: 65.3ms   remaining: 703ms
17:     learn: 0.2809187        total: 66.4ms   remaining: 672ms
18:     learn: 0.2730252        total: 67.4ms   remaining: 642ms
19:     learn: 0.2641290        total: 68.5ms   remaining: 617ms
20:     learn: 0.2565324        total: 69.5ms   remaining: 592ms
21:     learn: 0.2436529        total: 70.6ms   remaining: 571ms
22:     learn: 0.2405098        total: 71.4ms   remaining: 549ms
23:     learn: 0.2348255        total: 72.4ms   remaining: 531ms
```

```
24:     learn: 0.2293342        total: 73.5ms   remaining: 514ms
25:     learn: 0.2215201        total: 74.4ms   remaining: 498ms
26:     learn: 0.2125021        total: 75.5ms   remaining: 484ms
27:     learn: 0.2037552        total: 76.5ms   remaining: 470ms
28:     learn: 0.1996584        total: 77.4ms   remaining: 456ms
29:     learn: 0.1929344        total: 78.3ms   remaining: 444ms
30:     learn: 0.1845759        total: 79.3ms   remaining: 432ms
31:     learn: 0.1794482        total: 80.4ms   remaining: 422ms
32:     learn: 0.1721649        total: 81.2ms   remaining: 411ms
33:     learn: 0.1670736        total: 82ms     remaining: 401ms
34:     learn: 0.1623246        total: 83.1ms   remaining: 392ms
35:     learn: 0.1572181        total: 84.1ms   remaining: 383ms
36:     learn: 0.1572009        total: 84.5ms   remaining: 372ms
37:     learn: 0.1539729        total: 85.5ms   remaining: 365ms
38:     learn: 0.1535214        total: 86ms     remaining: 355ms
39:     learn: 0.1505825        total: 87ms     remaining: 348ms
40:     learn: 0.1451667        total: 88.1ms   remaining: 342ms
41:     learn: 0.1424512        total: 89.2ms   remaining: 336ms
42:     learn: 0.1386530        total: 90.2ms   remaining: 329ms
43:     learn: 0.1344911        total: 91.2ms   remaining: 324ms
44:     learn: 0.1334826        total: 91.9ms   remaining: 316ms
45:     learn: 0.1312075        total: 92.9ms   remaining: 311ms
46:     learn: 0.1277087        total: 93.9ms   remaining: 306ms
47:     learn: 0.1252132        total: 94.9ms   remaining: 300ms
48:     learn: 0.1224295        total: 95.9ms   remaining: 295ms
49:     learn: 0.1205716        total: 96.9ms   remaining: 291ms
50:     learn: 0.1168845        total: 97.9ms   remaining: 286ms
51:     learn: 0.1135585        total: 99ms     remaining: 282ms
52:     learn: 0.1132123        total: 99.5ms   remaining: 276ms
53:     learn: 0.1100292        total: 100ms    remaining: 272ms
54:     learn: 0.1086760        total: 101ms    remaining: 267ms
55:     learn: 0.1058472        total: 102ms    remaining: 263ms
56:     learn: 0.1002297        total: 103ms    remaining: 259ms
57:     learn: 0.0977531        total: 104ms    remaining: 255ms
58:     learn: 0.0958838        total: 105ms    remaining: 252ms
59:     learn: 0.0935505        total: 106ms    remaining: 248ms
60:     learn: 0.0909632        total: 107ms    remaining: 245ms
61:     learn: 0.0873322        total: 108ms    remaining: 241ms
62:     learn: 0.0847598        total: 109ms    remaining: 238ms
63:     learn: 0.0831981        total: 110ms    remaining: 235ms
64:     learn: 0.0801172        total: 111ms    remaining: 231ms
65:     learn: 0.0778482        total: 113ms    remaining: 229ms
66:     learn: 0.0762141        total: 114ms    remaining: 226ms
67:     learn: 0.0745551        total: 115ms    remaining: 222ms
68:     learn: 0.0733157        total: 116ms    remaining: 220ms
69:     learn: 0.0709646        total: 117ms    remaining: 217ms
70:     learn: 0.0693339        total: 118ms    remaining: 214ms
71:     learn: 0.0676360        total: 119ms    remaining: 211ms
```

```
72:     learn: 0.0666511        total: 119ms    remaining: 208ms
73:     learn: 0.0658717        total: 120ms    remaining: 205ms
74:     learn: 0.0642397        total: 122ms    remaining: 203ms
75:     learn: 0.0637420        total: 122ms    remaining: 200ms
76:     learn: 0.0627294        total: 123ms    remaining: 197ms
77:     learn: 0.0614020        total: 124ms    remaining: 195ms
78:     learn: 0.0597003        total: 125ms    remaining: 192ms
79:     learn: 0.0581487        total: 127ms    remaining: 190ms
80:     learn: 0.0576556        total: 128ms    remaining: 187ms
81:     learn: 0.0559929        total: 129ms    remaining: 185ms
82:     learn: 0.0550454        total: 130ms    remaining: 183ms
83:     learn: 0.0540910        total: 131ms    remaining: 180ms
84:     learn: 0.0525559        total: 132ms    remaining: 178ms
85:     learn: 0.0508644        total: 133ms    remaining: 176ms
86:     learn: 0.0494802        total: 134ms    remaining: 174ms
87:     learn: 0.0484818        total: 135ms    remaining: 171ms
88:     learn: 0.0476333        total: 136ms    remaining: 169ms
89:     learn: 0.0466150        total: 137ms    remaining: 167ms
90:     learn: 0.0459159        total: 138ms    remaining: 165ms
91:     learn: 0.0442754        total: 139ms    remaining: 163ms
92:     learn: 0.0439857        total: 139ms    remaining: 160ms
93:     learn: 0.0433327        total: 140ms    remaining: 158ms
94:     learn: 0.0421936        total: 141ms    remaining: 156ms
95:     learn: 0.0412142        total: 142ms    remaining: 154ms
96:     learn: 0.0400072        total: 143ms    remaining: 152ms
97:     learn: 0.0386757        total: 144ms    remaining: 150ms
98:     learn: 0.0377154        total: 145ms    remaining: 148ms
99:     learn: 0.0369504        total: 146ms    remaining: 146ms
100:    learn: 0.0363817        total: 147ms    remaining: 144ms
101:    learn: 0.0357451        total: 148ms    remaining: 142ms
102:    learn: 0.0352235        total: 149ms    remaining: 140ms
103:    learn: 0.0346965        total: 150ms    remaining: 139ms
104:    learn: 0.0344951        total: 151ms    remaining: 137ms
105:    learn: 0.0341423        total: 152ms    remaining: 135ms
106:    learn: 0.0338168        total: 153ms    remaining: 133ms
107:    learn: 0.0331411        total: 154ms    remaining: 131ms
108:    learn: 0.0321822        total: 155ms    remaining: 130ms
109:    learn: 0.0319019        total: 156ms    remaining: 128ms
110:    learn: 0.0314680        total: 157ms    remaining: 126ms
111:    learn: 0.0305845        total: 158ms    remaining: 124ms
112:    learn: 0.0300843        total: 159ms    remaining: 122ms
113:    learn: 0.0296826        total: 160ms    remaining: 121ms
114:    learn: 0.0289181        total: 161ms    remaining: 119ms
115:    learn: 0.0286427        total: 162ms    remaining: 117ms
116:    learn: 0.0284335        total: 163ms    remaining: 116ms
117:    learn: 0.0279876        total: 164ms    remaining: 114ms
118:    learn: 0.0276698        total: 165ms    remaining: 112ms
119:    learn: 0.0272298        total: 166ms    remaining: 110ms
```

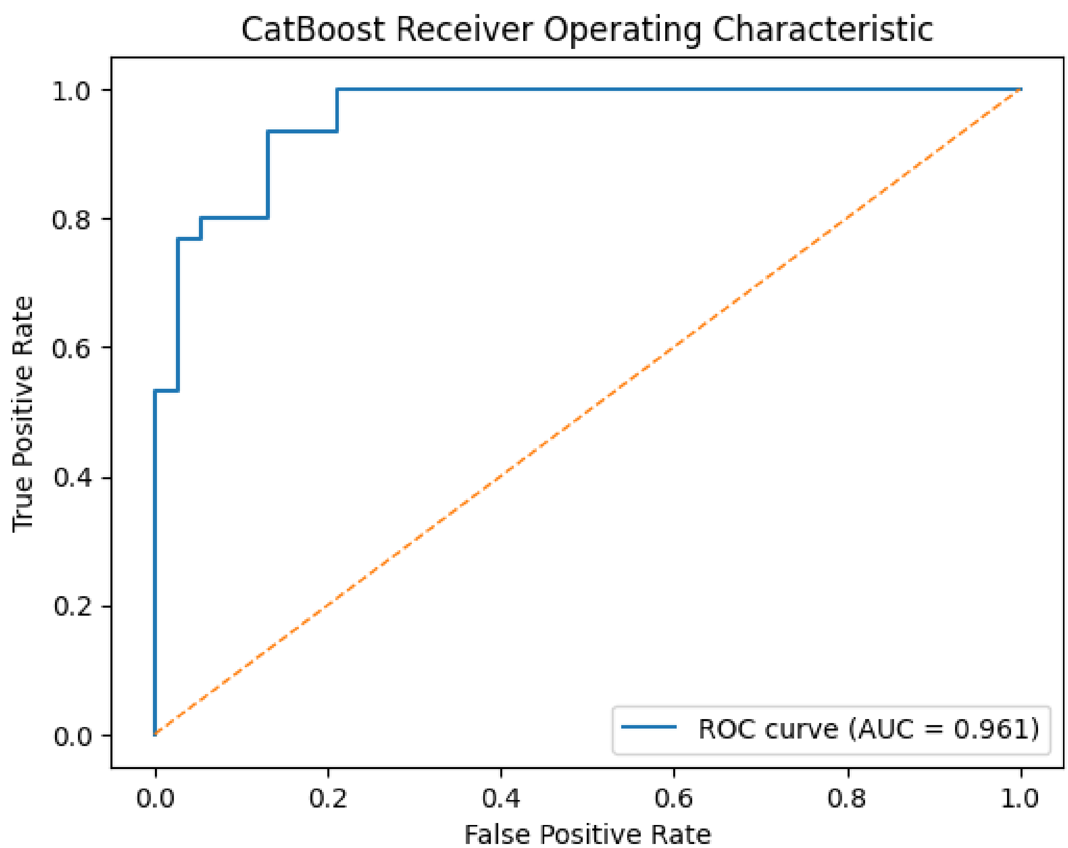
```
120:    learn: 0.0269485        total: 167ms    remaining: 109ms
121:    learn: 0.0266267        total: 168ms    remaining: 107ms
122:    learn: 0.0262445        total: 169ms    remaining: 106ms
123:    learn: 0.0257281        total: 170ms    remaining: 104ms
124:    learn: 0.0254373        total: 171ms    remaining: 102ms
125:    learn: 0.0250424        total: 172ms    remaining: 101ms
126:    learn: 0.0247979        total: 173ms    remaining: 99.2ms
127:    learn: 0.0241344        total: 174ms    remaining: 97.6ms
128:    learn: 0.0237783        total: 175ms    remaining: 96.1ms
129:    learn: 0.0234832        total: 175ms    remaining: 94.5ms
130:    learn: 0.0233156        total: 176ms    remaining: 93ms
131:    learn: 0.0231681        total: 177ms    remaining: 91.4ms
132:    learn: 0.0228606        total: 179ms    remaining: 90.1ms
133:    learn: 0.0223851        total: 180ms    remaining: 88.5ms
134:    learn: 0.0219639        total: 181ms    remaining: 87ms
135:    learn: 0.0217298        total: 182ms    remaining: 85.6ms
136:    learn: 0.0215157        total: 183ms    remaining: 84.1ms
137:    learn: 0.0212368        total: 184ms    remaining: 82.6ms
138:    learn: 0.0210671        total: 185ms    remaining: 81ms
139:    learn: 0.0206571        total: 185ms    remaining: 79.5ms
140:    learn: 0.0201631        total: 186ms    remaining: 78ms
141:    learn: 0.0197564        total: 187ms    remaining: 76.4ms
142:    learn: 0.0195333        total: 188ms    remaining: 74.9ms
143:    learn: 0.0192705        total: 189ms    remaining: 73.4ms
144:    learn: 0.0190205        total: 190ms    remaining: 72ms
145:    learn: 0.0187692        total: 191ms    remaining: 70.6ms
146:    learn: 0.0184775        total: 192ms    remaining: 69.2ms
147:    learn: 0.0183194        total: 193ms    remaining: 67.8ms
148:    learn: 0.0181327        total: 194ms    remaining: 66.4ms
149:    learn: 0.0180176        total: 195ms    remaining: 65ms
150:    learn: 0.0177636        total: 196ms    remaining: 63.6ms
151:    learn: 0.0175364        total: 197ms    remaining: 62.2ms
152:    learn: 0.0173777        total: 198ms    remaining: 60.9ms
153:    learn: 0.0172088        total: 199ms    remaining: 59.5ms
154:    learn: 0.0169582        total: 200ms    remaining: 58.1ms
155:    learn: 0.0167966        total: 201ms    remaining: 56.7ms
156:    learn: 0.0165438        total: 202ms    remaining: 55.3ms
157:    learn: 0.0163692        total: 203ms    remaining: 54ms
158:    learn: 0.0162543        total: 204ms    remaining: 52.6ms
159:    learn: 0.0160601        total: 205ms    remaining: 51.3ms
160:    learn: 0.0158619        total: 206ms    remaining: 49.9ms
161:    learn: 0.0156578        total: 207ms    remaining: 48.5ms
162:    learn: 0.0155133        total: 208ms    remaining: 47.2ms
163:    learn: 0.0153781        total: 209ms    remaining: 45.9ms
164:    learn: 0.0152523        total: 210ms    remaining: 44.5ms
165:    learn: 0.0150373        total: 211ms    remaining: 43.2ms
166:    learn: 0.0149248        total: 212ms    remaining: 41.9ms
167:    learn: 0.0147364        total: 213ms    remaining: 40.6ms
```

```
168:    learn: 0.0146610        total: 214ms    remaining: 39.2ms
169:    learn: 0.0144629        total: 215ms    remaining: 37.9ms
170:    learn: 0.0142976        total: 216ms    remaining: 36.6ms
171:    learn: 0.0141761        total: 217ms    remaining: 35.3ms
172:    learn: 0.0140164        total: 218ms    remaining: 34ms
173:    learn: 0.0138057        total: 219ms    remaining: 32.7ms
174:    learn: 0.0136879        total: 220ms    remaining: 31.5ms
175:    learn: 0.0136131        total: 221ms    remaining: 30.2ms
176:    learn: 0.0134512        total: 222ms    remaining: 28.9ms
177:    learn: 0.0133220        total: 223ms    remaining: 27.6ms
178:    learn: 0.0132624        total: 224ms    remaining: 26.3ms
179:    learn: 0.0131619        total: 225ms    remaining: 25ms
180:    learn: 0.0130354        total: 226ms    remaining: 23.8ms
181:    learn: 0.0129432        total: 227ms    remaining: 22.5ms
182:    learn: 0.0128168        total: 228ms    remaining: 21.2ms
183:    learn: 0.0126758        total: 229ms    remaining: 19.9ms
184:    learn: 0.0125520        total: 230ms    remaining: 18.7ms
185:    learn: 0.0124253        total: 231ms    remaining: 17.4ms
186:    learn: 0.0122516        total: 232ms    remaining: 16.2ms
187:    learn: 0.0121749        total: 233ms    remaining: 14.9ms
188:    learn: 0.0121189        total: 234ms    remaining: 13.6ms
189:    learn: 0.0120047        total: 235ms    remaining: 12.4ms
190:    learn: 0.0119277        total: 236ms    remaining: 11.1ms
191:    learn: 0.0117016        total: 237ms    remaining: 9.89ms
192:    learn: 0.0115816        total: 238ms    remaining: 8.65ms
193:    learn: 0.0114349        total: 240ms    remaining: 7.42ms
194:    learn: 0.0113053        total: 241ms    remaining: 6.17ms
195:    learn: 0.0112673        total: 242ms    remaining: 4.93ms
196:    learn: 0.0111205        total: 243ms    remaining: 3.7ms
197:    learn: 0.0109900        total: 244ms    remaining: 2.47ms
198:    learn: 0.0108626        total: 245ms    remaining: 1.23ms
199:    learn: 0.0107928        total: 246ms    remaining: 0us
LightGBM classifier:
Best params: {'bagging_temperature': 0, 'depth': 7, 'iterations': 200,
'l2_leaf_reg': 1, 'learning_rate': 0.1, 'min_data_in_leaf': 20, 'rsm': 0.8}

=== Repeated Stratified 5×2 CV on TRAIN ===
CV accuracy : 0.8601 ± 0.0651
CV precision: 0.8547 ± 0.0891
CV recall   : 0.8236 ± 0.0728
CV f1       : 0.8370 ± 0.0711
CV roc_auc  : 0.9116 ± 0.0463

=== Final Performance on HOLD-OUT TEST ===
Accuracy : 0.8824
Precision: 0.9231
Recall   : 0.8000
F1 Score : 0.8571
```

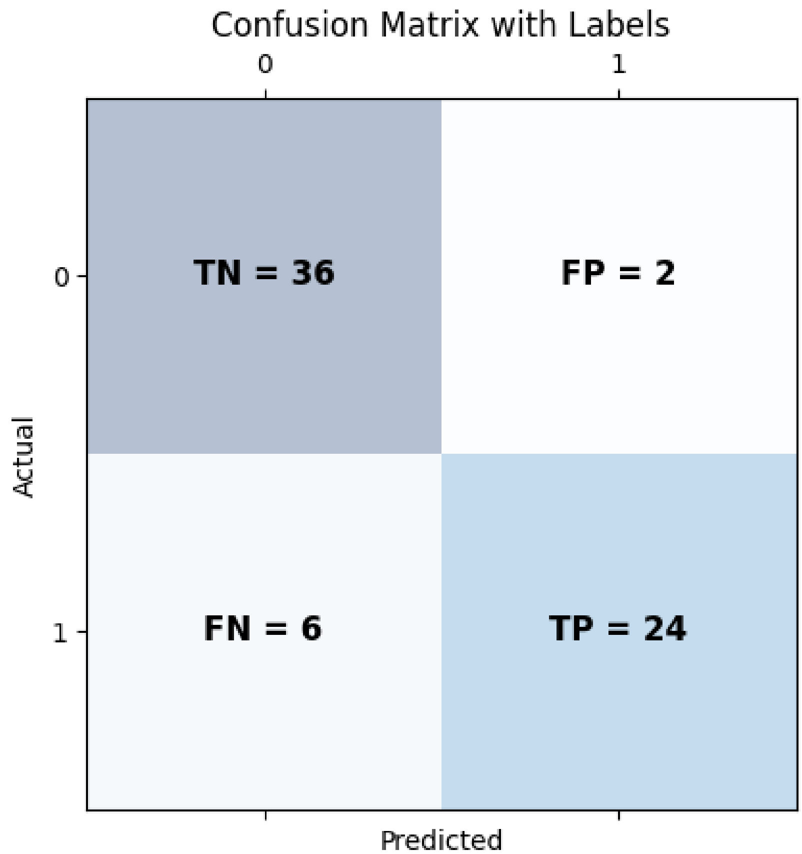
```
ROC_AUC  : 0.9605

Classification Report on Test Set:
              precision    recall  f1-score   support

           0       0.86      0.95      0.90        38
           1       0.92      0.80      0.86        30

    accuracy                           0.88        68
   macro avg       0.89      0.87      0.88        68
weighted avg       0.89      0.88      0.88        68

Confusion Matrix:
 [[36  2]
 [ 6 24]]

Top feature importances:
   Feature  Importance
4     eTIV   22.004060
5     nWBV   21.028797
3      SES   17.430076
2     EDUC   15.873306
1      Age   15.560894
0      M/F    8.102866
```

[ ]:

```
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = roc_auc_score(y_test, y_prob)

plt.figure()
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc_auc:.3f})')
plt.plot([0, 1], [0, 1], linestyle='--', linewidth=1)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('CatBoost Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()
```



[ ]:
```
cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()

labels = np.array([
    [f"TN = {tn}", f"FP = {fp}"],
    [f"FN = {fn}", f"TP = {tp}"]
])

fig, ax = plt.subplots()
ax.matshow(cm, cmap=plt.cm.Blues, alpha=0.3)
for i in range(2):
    for j in range(2):
        ax.text(j, i, labels[i, j],
                va='center', ha='center', fontsize=12, fontweight='bold')

plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix with Labels")
plt.show()
```




```
[ ]: # Find all FN indices in the full test set
     FN_all = (~y_pred) & (y_test == 1)
     FN_indices = y_test[FN_all].index
     print("False Negative indices:", FN_indices)
```

```
False Negative indices: Index([332, 52, 300, 299, 51, 94], dtype='int64')
```

```
[ ]: FN_sample_test_idx = X_test.index.get_indexer_for([332, 52, 300, 299, 51, 94])
     FN_sample_test_idx
```

```
[ ]: array([ 2, 30, 31, 37, 49, 61])
```

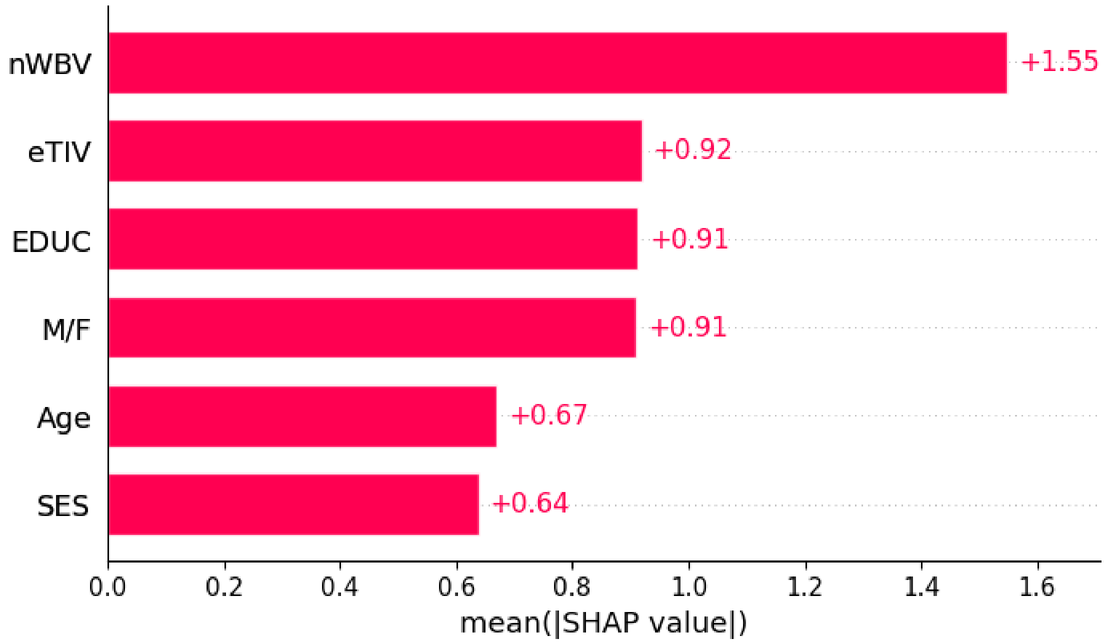
```
[ ]: CB_explainer = shap.Explainer(CatBoost_mdl)
     CB_shap = CB_explainer(X_test)
     print(type(CB_explainer))
```

```
<class 'shap.explainers._tree.TreeExplainer'>
```

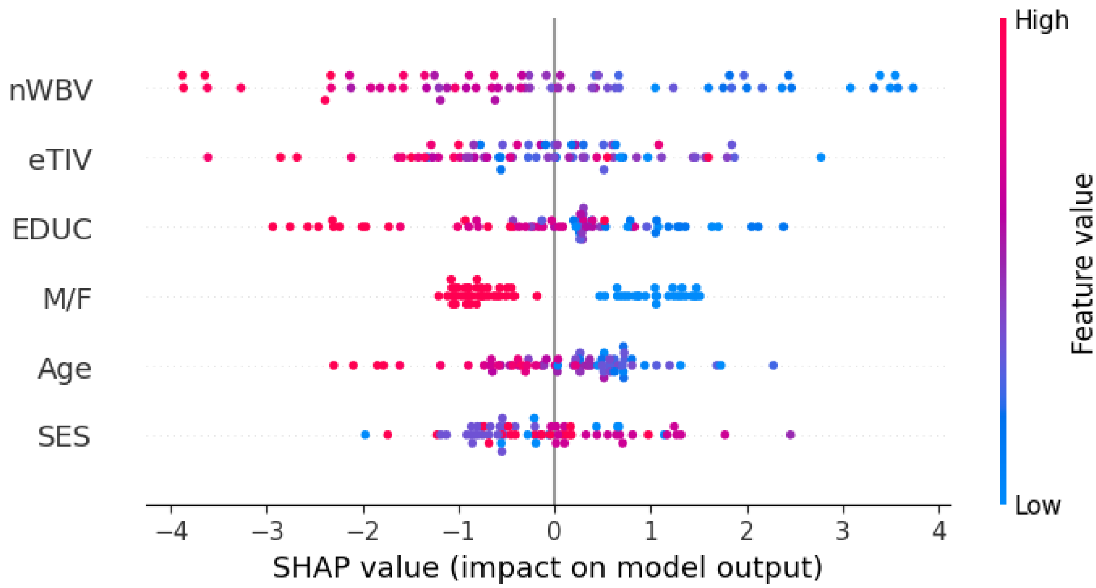
```
[ ]: print("Values dimensions: %s" % (CB_shap.values.shape,))
     print("Data dimensions:   %s" % (CB_shap.data.shape,))
```

```
Values dimensions: (68, 6)
Data dimensions:   (68, 6)
```

[ ]:
```
sb.reset_orig()
shap.plots.bar(CB_shap)
```



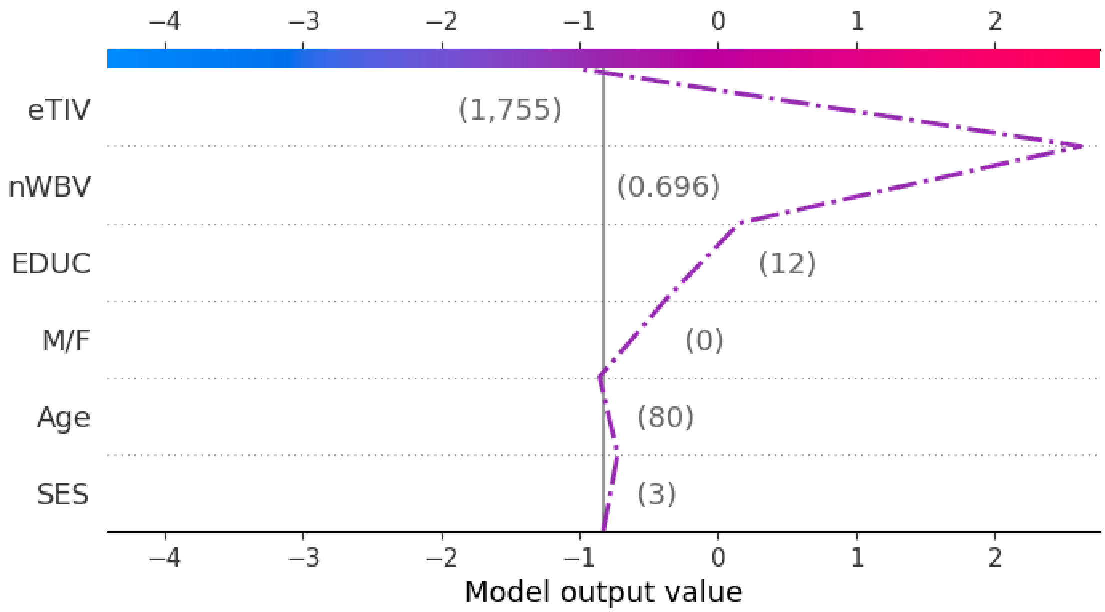
[ ]:
```
shap.plots.beeswarm(CB_shap)
```



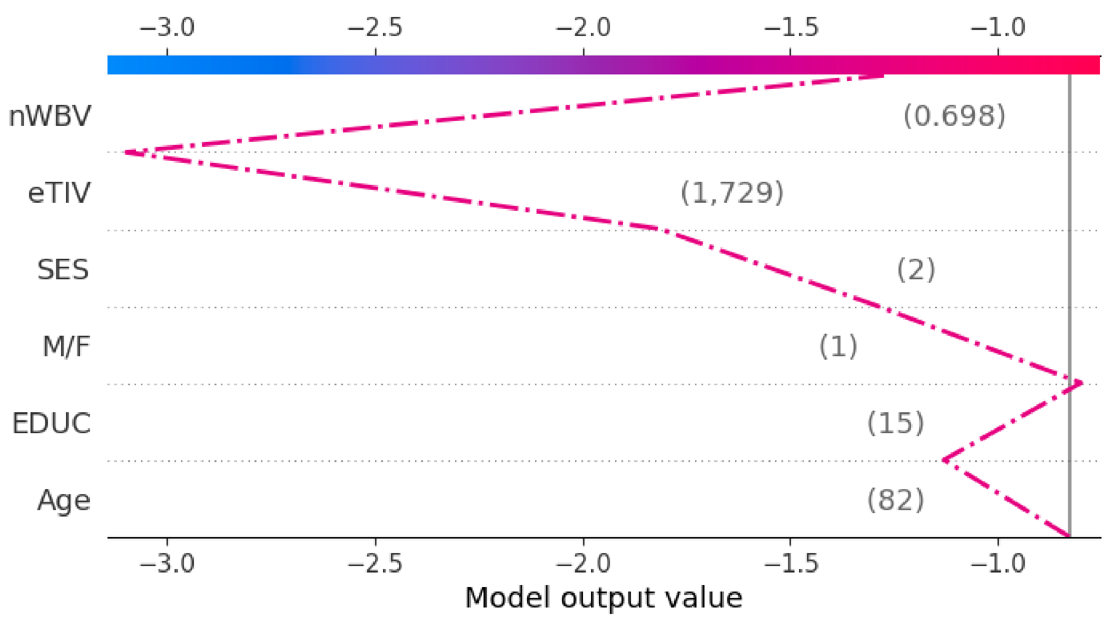
[ ]:
```
print("X_test.iloc[2]: ")
print(X_test.iloc[2])
print(y_test.iloc[2], y_pred[2])
print("------------------")
print("X_test.iloc[31]: ")
print(X_test.iloc[31])
print(y_test.iloc[31], y_pred[31])
```

```
X_test.iloc[2]:
M/F         0.000
Age        80.000
EDUC       12.000
SES         3.000
eTIV     1755.000
nWBV        0.696
Name: 332, dtype: float64
1 0
-----------------
X_test.iloc[31]:
M/F         1.000
Age        82.000
EDUC       15.000
SES         2.000
eTIV     1729.000
nWBV        0.698
Name: 300, dtype: float64
1 0
```

[ ]:
```
# CB-FN-2
expected_value = CB_explainer.expected_value
shap.decision_plot(expected_value, CB_shap.values[2], X_test.iloc[2],
  ↪highlight=0)
```




```
[ ]: # CB-FN-31
expected_value = CB_explainer.expected_value
shap.decision_plot(expected_value, CB_shap.values[31], X_test.iloc[31],
 ↪highlight=0)
```



[ ]:
```
# CB-FN-2
shap.initjs()
expected_value = CB_explainer.expected_value
shap.force_plot(expected_value, CB_shap.values[2], X_test.iloc[2])
```

<IPython.core.display.HTML object>

[ ]: <shap.plots._force.AdditiveForceVisualizer at 0x7f38fd7e2690>

[ ]:
```
# CB-FN-31
shap.initjs()
expected_value = CB_explainer.expected_value
shap.force_plot(expected_value, CB_shap.values[31], X_test.iloc[31])
```

<IPython.core.display.HTML object>

[ ]: <shap.plots._force.AdditiveForceVisualizer at 0x7f38fd554710>

[ ]:
```
lime_CB_explainer = lime.lime_tabular.LimeTabularExplainer(X_test.values,
                                                   feature_names=X_test.columns,
                                                   class_names=['Nondemented',␣
 ↪'Demented'])
```

[ ]:
```
# CB-FN-2
lime_CB_explainer.explain_instance(X_test.iloc[2].values,\
                                    CatBoost_mdl.predict_proba,\
                                    num_features=6).\
                                show_in_notebook(predict_proba=True)
```

<IPython.core.display.HTML object>

[ ]:
```
# CB-FN-31
lime_CB_explainer.explain_instance(X_test.iloc[31].values,\
                                    CatBoost_mdl.predict_proba,\
                                    num_features=6).\
                                show_in_notebook(predict_proba=True)
```

<IPython.core.display.HTML object>

[ ]:
```
# store all fn_results
fn_results = []
feature_counter = Counter()

CB_FN_indices = [ 2, 30, 31, 37, 49, 61]

for fn_idx in CB_FN_indices:
    CB_instance_values = X_test.iloc[fn_idx].values
```

```python
    exp = lime_CB_explainer.explain_instance(
        CB_instance_values,
        CatBoost_mdl.predict_proba,
        num_features=6
    )

    exp_list = exp.as_list()

    pushed_non = [f for f, w in exp_list if w < 0]
    pushed_dem = [f for f, w in exp_list if w > 0]

    fn_results.append({
        'Index': fn_idx,
        'Pushed_Nondemented': pushed_non,
        'Pushed_Demented': pushed_dem
    })

    feature_counter.update(pushed_non)

fn_df = pd.DataFrame(fn_results)

top_causes = pd.DataFrame(feature_counter.most_common(), columns=['Feature',␣
 ↪'Count'])

print(fn_df.head())

print("\n=== CB False Negative Feature Frequency ===")
print(top_causes)
```

```
   Index                                 Pushed_Nondemented  \
0      2              [eTIV > 1669.00, 75.00 < Age <= 80.25]
1     30  [EDUC > 16.25, SES <= 2.00, 1491.50 < eTIV <= …
2     31  [Age > 80.25, 0.00 < M/F <= 1.00, eTIV > 1669…
3     37  [0.00 < M/F <= 1.00, eTIV > 1669.00, 75.00 < A…
4     49   [EDUC > 16.25, 0.73 < nWBV <= 0.76, SES <= 2.00]

                                     Pushed_Demented
0  [nWBV <= 0.70, EDUC <= 12.00, M/F <= 0.00, 2.0…
1   [Age <= 71.00, M/F <= 0.00, 0.70 < nWBV <= 0.73]
2             [nWBV <= 0.70, 12.00 < EDUC <= 15.00]
3             [nWBV <= 0.70, 12.00 < EDUC <= 15.00]
4  [Age <= 71.00, M/F <= 0.00, 1491.50 < eTIV <= …

=== CB False Negative Feature Frequency ===
                 Feature  Count
0            SES <= 2.00      5
1         eTIV > 1669.00      3
```

```
2         0.00 < M/F <= 1.00      3
3       75.00 < Age <= 80.25      2
4                EDUC > 16.25      2
5                 Age > 80.25      2
6        0.73 < nWBV <= 0.76      2
7   1491.50 < eTIV <= 1669.00      1
```

7.3 Additional testing

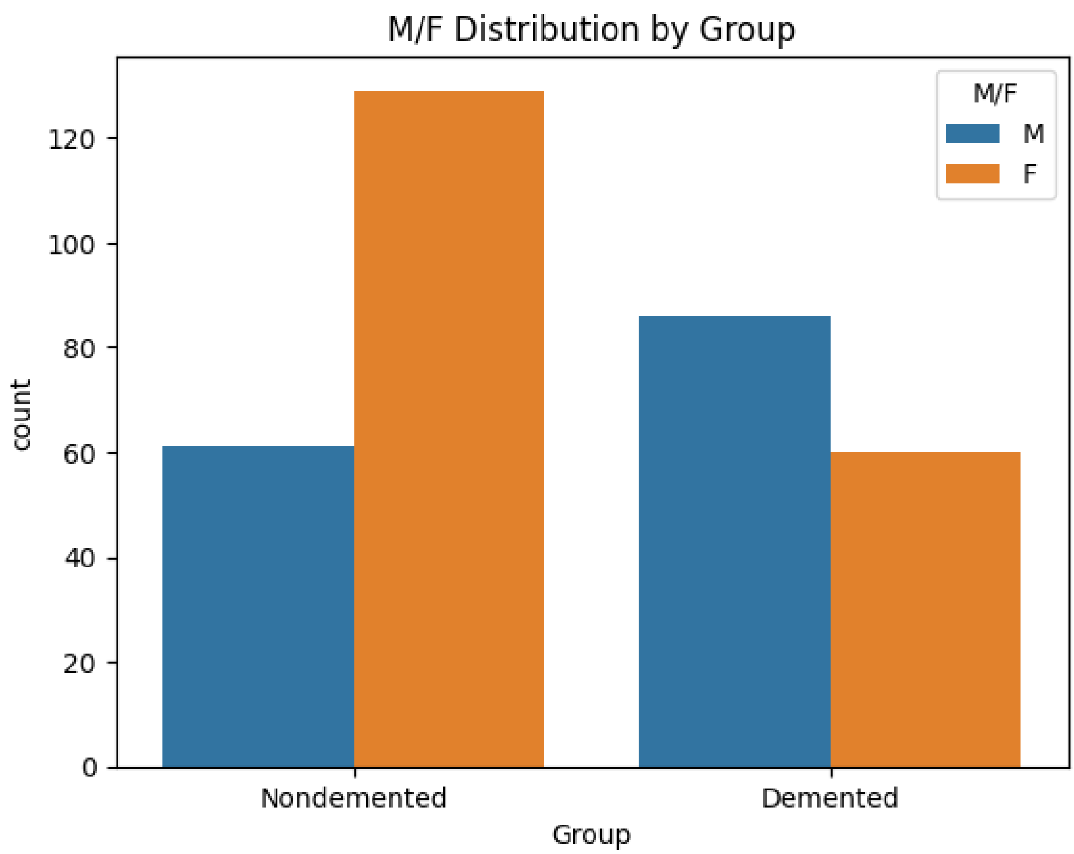
Among all three gradient boosting models, from the shap global analysis, we found that older age and female tend to push result to the nondemented side,which is contrary to the facts. In reality, age is positively correlated with Alzheimer's disease. Women, because they live longer than men, are more likely to be included in Alzheimer's disease samples than men. (Some studies have also shown that no significant gender differences were found in analyses of the same age group.)

This result maybe due to sample selection bias.

7.3.1 Check Gender and Age distribution by Group

[ ]:
```
sb.countplot(x='Group', hue='M/F', data=df)

pct = pd.crosstab(df['Group'], df['M/F'], normalize='index') * 100
plt.title('M/F Distribution by Group')
print(pct)
```

```
M/F                  F          M
Group
Demented     41.095890  58.904110
Nondemented  67.894737  32.105263
```



[ ]:
```
df['M/F'].value_counts()
```

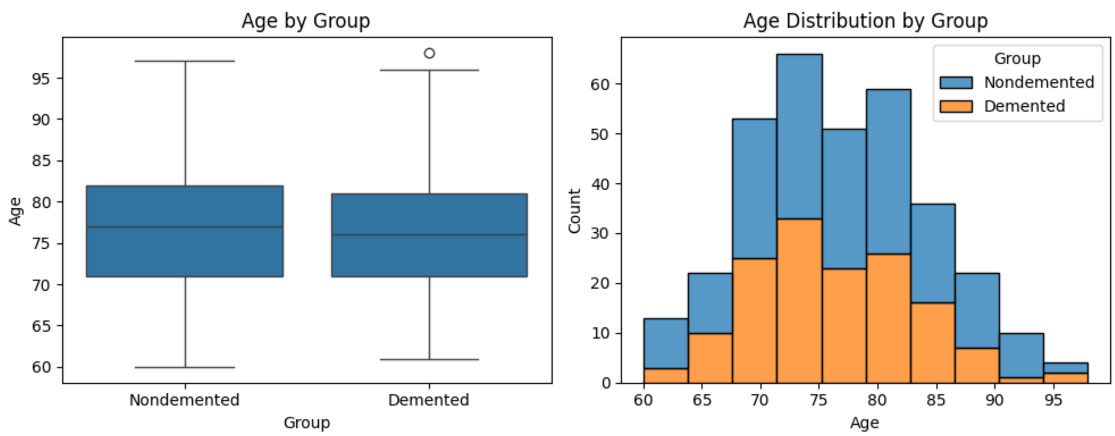
[ ]:
```
M/F
F    189
M    147
Name: count, dtype: int64
```

in Nondemented group 68% is female

in demented group, 41% is female.

[ ]:
```
# double check Age by group
# Create a figure with 1 row and 2 columns of subplots
fig, axes = plt.subplots(1, 2, figsize=(10, 4))

# First subplot: Boxplot of Age by group
sb.boxplot(x='Group', y='Age', data=df, ax=axes[0])
axes[0].set_title('Age by Group')
axes[0].set_xlabel('Group')
axes[0].set_ylabel('Age')
```

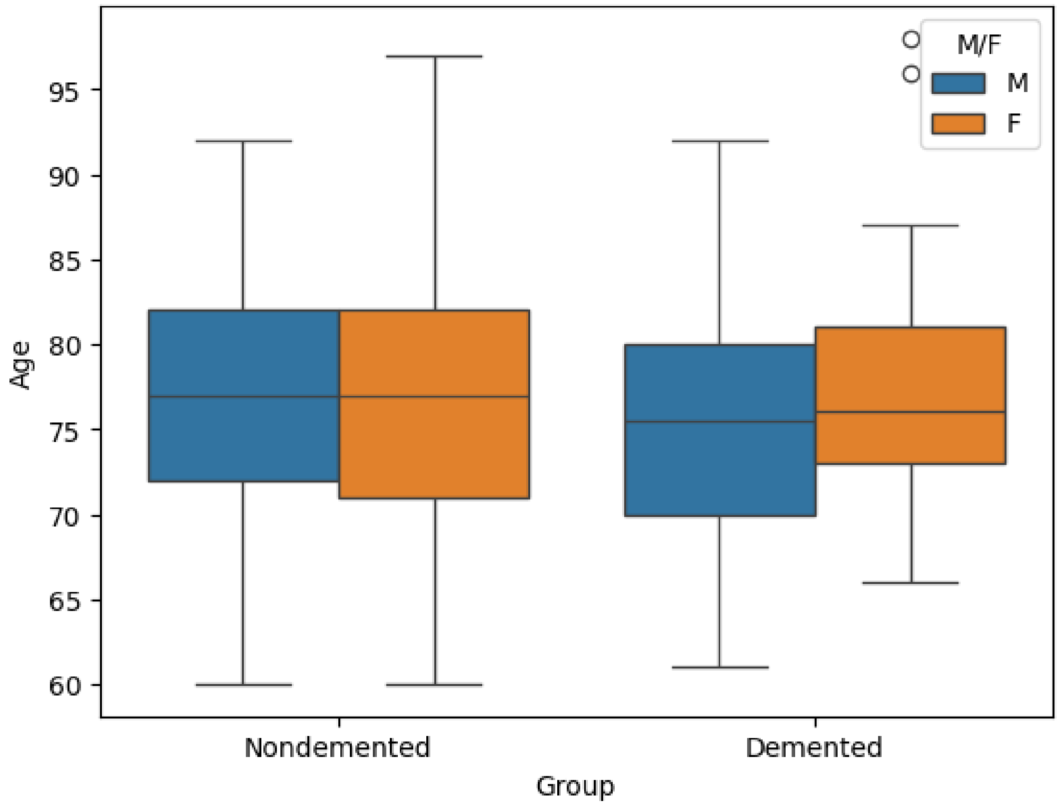
```
# Second subplot: Distribution of Age colored by Group
sb.histplot(data=df, x='Age', hue='Group', bins=10, multiple='stack',␣
↪ax=axes[1]) # or multiple='dodge'
axes[1].set_title('Age Distribution by Group')
axes[1].set_xlabel('Age')
axes[1].set_ylabel('Count')

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



[ ]:
```
sb.boxplot(x='Group', y='Age', hue='M/F', data=df)
df.groupby('Group')['Age'].describe()
```

[ ]:

	count	mean	std	min	25%	50%	75%	max
Group								
Demented	146.0	76.260274	6.940193	61.0	71.0	76.0	81.0	98.0
Nondemented	190.0	77.057895	8.096104	60.0	71.0	77.0	82.0	97.0



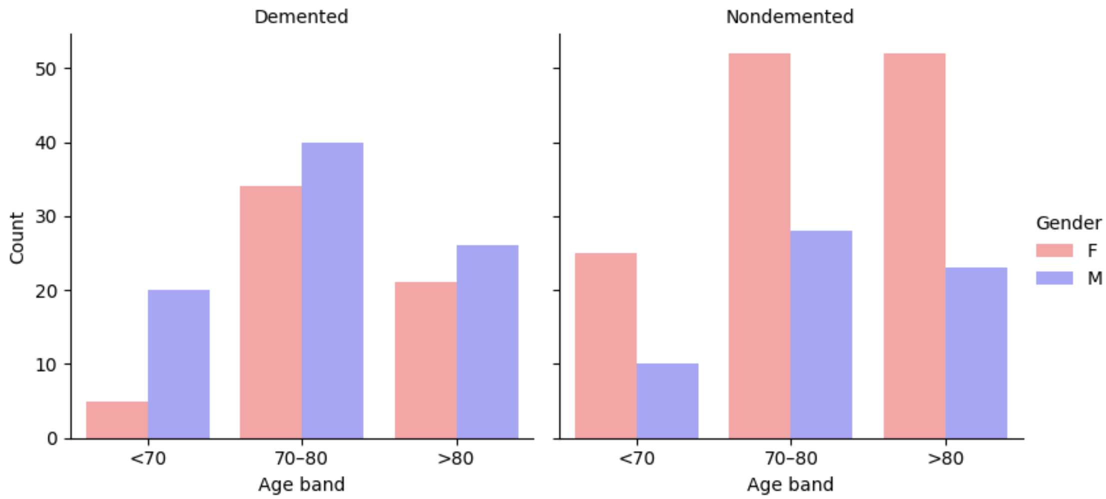
[ ]:
```
from scipy.stats import ttest_ind, mannwhitneyu

nd_age = df[df['Group']=='Nondemented']['Age']
d_age = df[df['Group']=='Demented']['Age']

# t test
t_stat, p_val = ttest_ind(nd_age, d_age, equal_var=False)
print(f"T-test: t={t_stat:.3f}, p={p_val:.3f}")

# Mann-Whitney U
u_stat, p_val_u = mannwhitneyu(nd_age, d_age, alternative='two-sided')
print(f"Mann-Whitney U: U={u_stat:.3f}, p={p_val_u:.3f}")
```

```
T-test: t=0.971, p=0.332
Mann-Whitney U: U=14745.000, p=0.321
```

[ ]:
```
# 1) age group
bins = [0, 70, 80, np.inf]
labels = ['<70', '70-80', '>80']
df['AgeBand'] = pd.cut(df['Age'].astype(float), bins=bins, labels=labels, right␣
↪= False)
```

```
counts = df.groupby(['AgeBand', 'Group', 'M/F']).size().
    ↪reset_index(name='count')

g = sb.catplot(
    data=counts,
    x='AgeBand',
    y='count',
    hue='M/F',
    col='Group',
    kind='bar',
    palette={'F': '#FF9999', 'M': '#9999FF'},
    height=4,
    aspect=1
)
g.set_axis_labels("Age band", "Count")
g.set_titles("{col_name}")
g._legend.set_title("Gender")

plt.show()

# 2) count every group's  Demented% and Female%
summary = (
    df.groupby('AgeBand')
      .agg(
          n_total    = ('Group', 'size'),
          n_demented = ('Group', lambda x: (x.str.lower() == 'demented').sum()),
          n_female   = ('M/F',   lambda x: (x.str.upper() == 'F').sum())
      )
)

summary['pct_demented'] = (summary['n_demented'] / summary['n_total'] * 100).
    ↪round(1)
summary['pct_female']   = (summary['n_female']   / summary['n_total'] * 100).
    ↪round(1)

print(summary.reset_index())
```

```
/tmp/ipython-input-1705621936.py:6: FutureWarning: The default of observed=False
is deprecated and will be changed to True in a future version of pandas. Pass
observed=False to retain current behavior or observed=True to adopt the future
default and silence this warning.
  counts = df.groupby(['AgeBand', 'Group',
'M/F']).size().reset_index(name='count')
```




```
  AgeBand  n_total  n_demented  n_female  pct_demented  pct_female
0     <70       60          25        30          41.7        50.0
1   70-80      154          74        86          48.1        55.8
2     >80      122          47        73          38.5        59.8
```

```
/tmp/ipython-input-1705621936.py:27: FutureWarning: The default of
observed=False is deprecated and will be changed to True in a future version of
pandas. Pass observed=False to retain current behavior or observed=True to adopt
the future default and silence this warning.
  df.groupby('AgeBand')
```

1. Dementia proportion (pct_demented)

- The <70 group has the lowest proportion (41.7%).
- The 70–80 age group has the highest dementia proportion (48.1%).
- In the older group (>80), the dementia proportion actually drops to 38.5%.

This pattern does not fully align with the common expectation that "older age means higher risk," suggesting the presence of healthy older survivors (especially women) in our dataset. This survivor bias likely contributes to the lower dementia proportion in the >80 group.

2. Female proportion (pct_female)

The proportion of females increases with age: <70 (50%) → 70–80 (55.8%) → >80 (59.8%), which is an expected pattern, as women generally live longer, leading to a higher proportion of females in the oldest age group.

3. Implication for our SHAP results

In the >80 group, the proportion of females is very high, and a substantial portion of these women are nondemented. This can lead the model to learn the pattern "female & older age → nondemented."

Combined with the drop in dementia proportion for the >80 group, these two factors together may explain why, in the global SHAP analysis, Age and Female show contributions toward the nondemented class — a direction contrary to typical clinical expectations.

7.3.2 Model training-XGBoost_remove SES

[ ]:
```
# Try remove SES feature
X = df.drop(['Group', 'CDR', 'MMSE', 'ASF', 'SES'], axis = 1)
y = df['Group']

# Split dataset into training and test sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[ ]:
```
X_test.columns
```

[ ]:
```
Index(['M/F', 'Age', 'EDUC', 'eTIV', 'nWBV'], dtype='object')
```

[ ]:
```
xgb = XGBClassifier(random_state=42,n_jobs=-1)

param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.001, 0.01, 0.1],
    'subsample': [0.7, 0.8, 0.9],
    'colsample_bytree': [0.7, 0.8, 1.0],
    'min_child_weight': [1, 3, 5],
}

inner_cv = RepeatedStratifiedKFold(
    n_splits=5, n_repeats=2, random_state=42
)

grid = GridSearchCV(
    estimator=xgb,
    param_grid=param_grid,
    scoring='f1',
    cv=inner_cv,
    n_jobs=-1,
    verbose=0
)

grid.fit(X_train, y_train)
best_model = grid.best_estimator_
print("XGBoost classifier:")
print("Best params:", grid.best_params_)
```

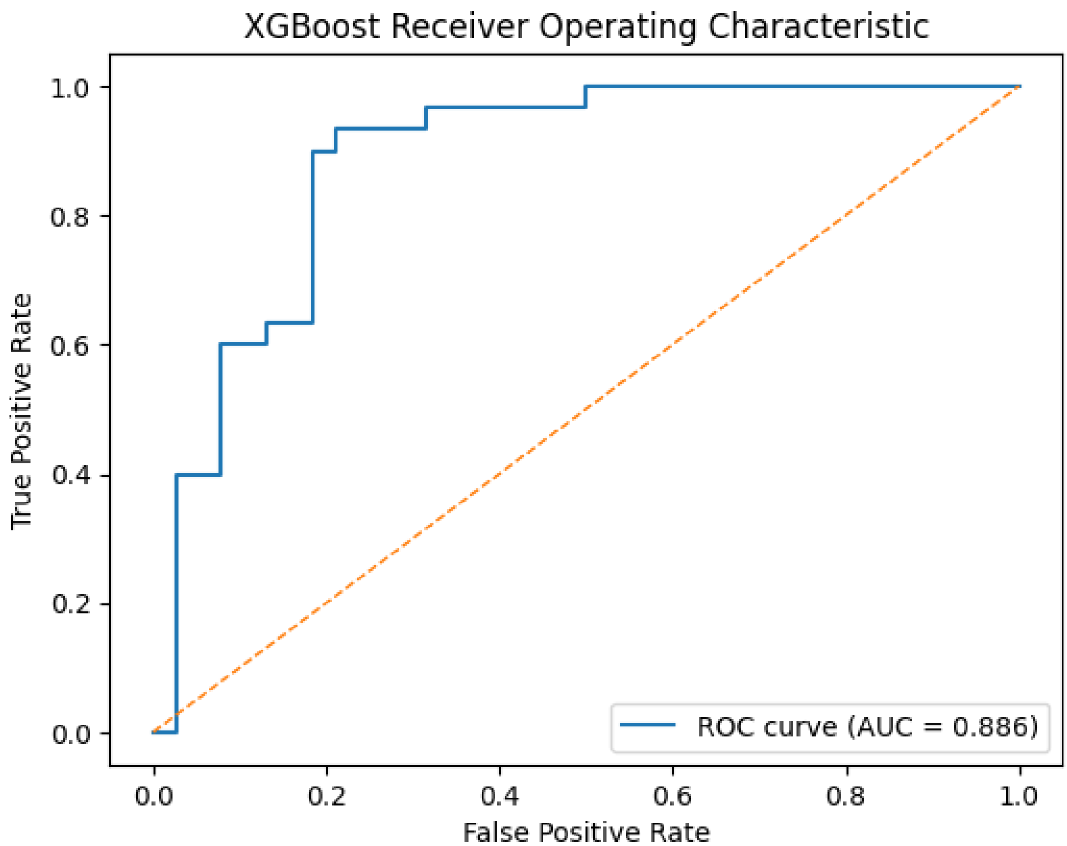
```python
scoring = {
    'accuracy': 'accuracy',
    'precision': 'precision',
    'recall': 'recall',
    'f1': 'f1',
    'roc_auc': 'roc_auc'
}
cv_res = cross_validate(
    best_model, X_train, y_train,
    cv=inner_cv, scoring=scoring, n_jobs=-1, return_train_score=False
)

def show(name):
    vals = cv_res[f'test_{name}']
    print(f"CV {name:<9}: {vals.mean():.4f} ± {vals.std():.4f}")

print("\n=== Repeated Stratified 5×2 CV on TRAIN ===")
for m in ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']:
    show(m)

y_pred = best_model.predict(X_test)
y_prob = best_model.predict_proba(X_test)[:, 1]

acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_prob)

print("\n=== Final Performance on HOLD-OUT TEST ===")
print(f"Accuracy : {acc:.4f}")
print(f"Precision: {prec:.4f}")
print(f"Recall   : {rec:.4f}")
print(f"F1 Score : {f1:.4f}")
print(f"ROC_AUC  : {auc:.4f}")

print("\nClassification Report on Test Set:\n", classification_report(y_test,␣
 ↪y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

feat_imp = pd.DataFrame({
    "Feature": X_train.columns,
    "Importance": best_model.feature_importances_
}).sort_values(by="Importance", ascending=False)
print("\nTop feature importances:\n", feat_imp)

XGBoost_mdl = best_model
```

```
XGBoost classifier:
Best params: {'colsample_bytree': 1.0, 'learning_rate': 0.1, 'max_depth': 5,
'min_child_weight': 1, 'n_estimators': 200, 'subsample': 0.9}

=== Repeated Stratified 5×2 CV on TRAIN ===
CV accuracy : 0.8005 ± 0.0613
CV precision: 0.7818 ± 0.0930
CV recall   : 0.7638 ± 0.0690
CV f1       : 0.7696 ± 0.0647
CV roc_auc  : 0.8551 ± 0.0533

=== Final Performance on HOLD-OUT TEST ===
Accuracy : 0.8382
Precision: 0.7879
Recall   : 0.8667
F1 Score : 0.8254
ROC_AUC  : 0.8860

Classification Report on Test Set:
              precision    recall  f1-score   support

           0       0.89      0.82      0.85        38
           1       0.79      0.87      0.83        30

    accuracy                           0.84        68
   macro avg       0.84      0.84      0.84        68
weighted avg       0.84      0.84      0.84        68

Confusion Matrix:
 [[31  7]
 [ 4 26]]

Top feature importances:
   Feature  Importance
2    EDUC    0.279822
0     M/F    0.253008
4    nWBV    0.174518
3    eTIV    0.149782
1     Age    0.142870
```

[ ]:

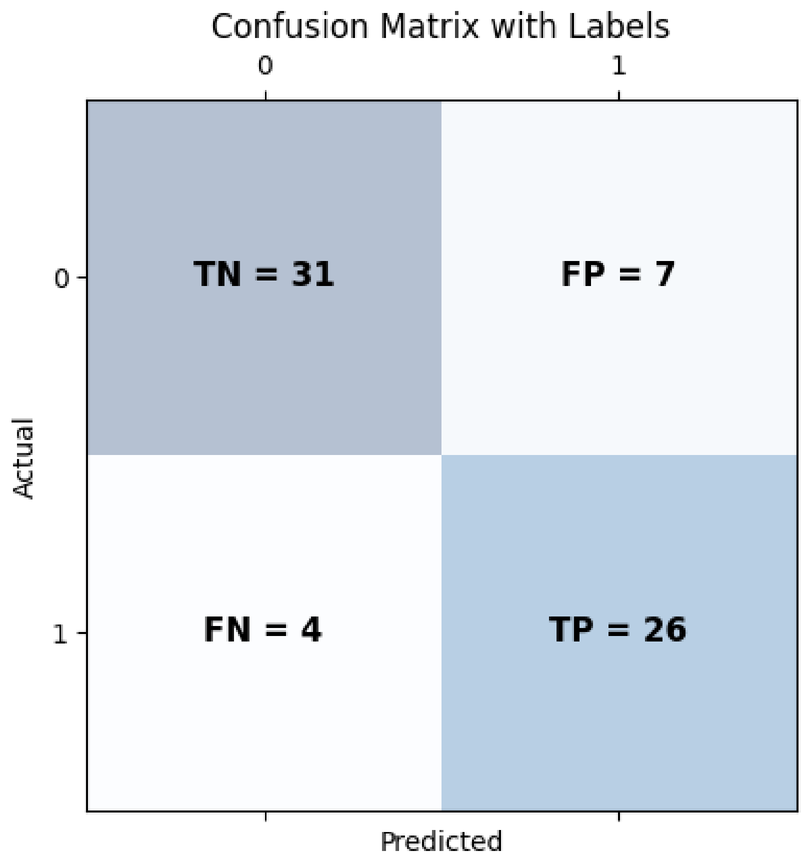
```
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = roc_auc_score(y_test, y_prob)

plt.figure()
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc_auc:.3f})')
plt.plot([0, 1], [0, 1], linestyle='--', linewidth=1)
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.title('XGBoost Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()
```



[ ]:
```
cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()

labels = np.array([
    [f"TN = {tn}", f"FP = {fp}"],
    [f"FN = {fn}", f"TP = {tp}"]
])

fig, ax = plt.subplots()
ax.matshow(cm, cmap=plt.cm.Blues, alpha=0.3)
for i in range(2):
    for j in range(2):
        ax.text(j, i, labels[i, j],
                va='center', ha='center', fontsize=12, fontweight='bold')
```

```
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix with Labels")
plt.show()
```



[ ]:
```
# Find all FN indices in the full test set
FN_all = (~y_pred) & (y_test == 1)
FN_indices = y_test[FN_all].index
print("False Negative indices:", FN_indices)

# Find all FP indices in the full test set
FP_all = (y_pred) & (y_test == 0)
FP_indices = y_test[FP_all].index
print("False Positive indices:", FP_indices)

# Find all TP indices in the full test set
TP_all = (y_pred) & (y_test == 1)
TP_indices = y_test[TP_all].index
print("True Positive indices:", TP_indices)
```

```
# Find all FN indices in the full test set
TN_all = (~y_pred) & (y_test == 0)
TN_indices = y_test[TN_all].index
print("True Negative indices:", TN_indices)
```

```
False Negative indices: Index([300, 299, 51, 94], dtype='int64')
False Positive indices: Index([167, 146, 198, 130, 199, 7, 64], dtype='int64')
True Positive indices: Index([124, 332, 250, 317, 154,  25,  90, 106, 172, 285,
87, 215, 127,  52,
         3, 239, 162, 345,  72,  39,  89,  88,  16, 329, 365, 275],
      dtype='int64')
True Negative indices: Index([ 84, 122, 311,  48, 336, 213,   9, 210, 113,  85,
363,  66,   5, 153,
       291, 370, 158, 102, 243, 132, 306, 337, 196, 351, 179, 309, 197, 362,
       209, 333,  96],
      dtype='int64')
```

[ ]:
```
FN_sample_test_idx = X_test.index.get_indexer_for([300, 299, 51, 94])
FP_sample_test_idx = X_test.index.get_indexer_for([167, 146, 198, 130, 199, 7,␣
↪64])
TP_sample_test_idx = X_test.index.get_indexer_for([124, 332, 250, 317, 154, ␣
↪25,  90, 106, 172, 285,  87, 215, 127,  52,
         3, 239, 162, 345,  72,  39,  89,  88,  16, 329, 365, 275])
TN_sample_test_idx = X_test.index.get_indexer_for([84, 122, 311,  48, 336, 213,␣
↪   9, 210, 113,  85, 363,  66,   5, 153,
       291, 370, 158, 102, 243, 132, 306, 337, 196, 351, 179, 309, 197, 362,
       209, 333,  96])
```

[ ]:
```
print("FN_sample_test_idx: ", FN_sample_test_idx)
print("FP_sample_test_idx: ", FP_sample_test_idx)
print("TP_sample_test_idx: ", TP_sample_test_idx)
print("TN_sample_test_idx: ", TN_sample_test_idx)
```

```
FN_sample_test_idx:  [31 37 49 61]
FP_sample_test_idx:  [17 21 42 50 53 59 60]
TP_sample_test_idx:  [ 1  2  6  7  9 11 13 14 16 20 22 28 29 30 33 35 36 39 40
41 47 55 56 64
 65 67]
TN_sample_test_idx:  [ 0  3  4  5  8 10 12 15 18 19 23 24 25 26 27 32 34 38 43
44 45 46 48 51
 52 54 57 58 62 63 66]
```

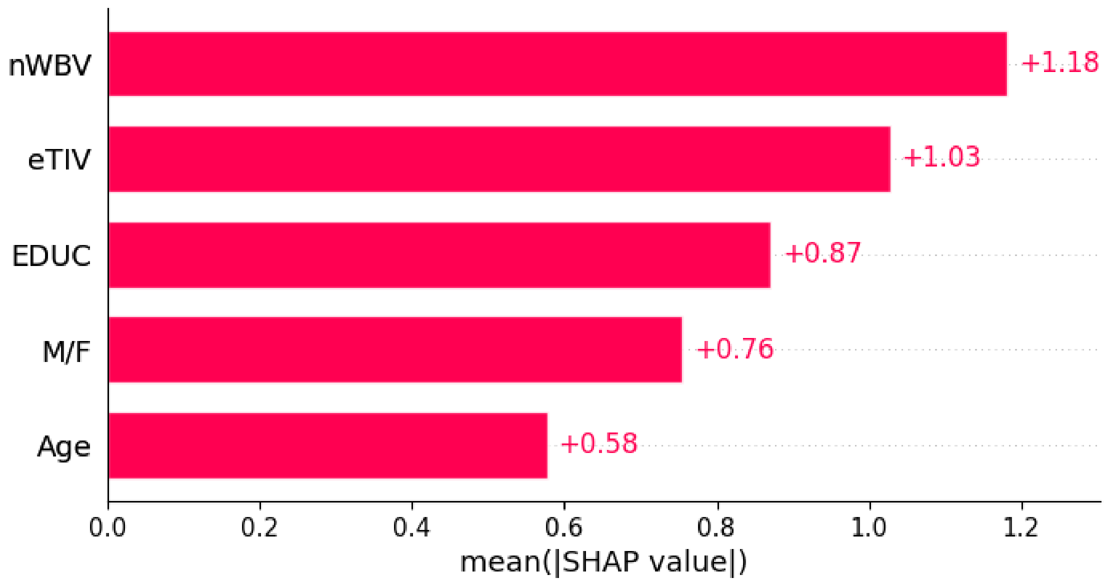
[ ]:
```
XGB_explainer = shap.Explainer(XGBoost_mdl)
XGB_shap = XGB_explainer(X_test)
print(type(XGB_explainer))
```

```
<class 'shap.explainers._tree.TreeExplainer'>
```

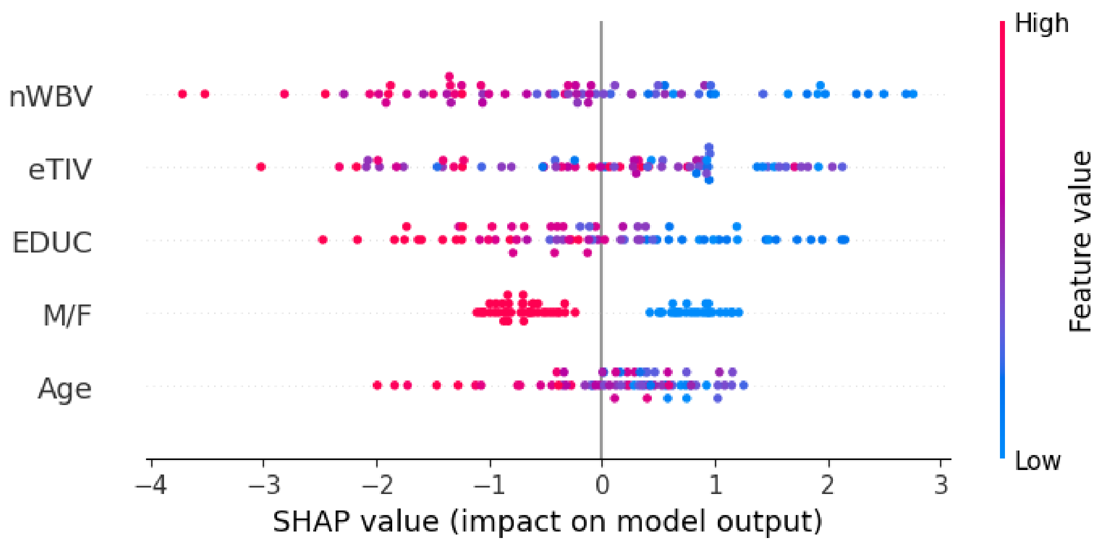
[ ]:
```
print("Values dimensions: %s" % (XGB_shap.values.shape,))
print("Data dimensions:   %s" % (XGB_shap.data.shape,))
```

```
Values dimensions: (68, 5)
Data dimensions:   (68, 5)
```

[ ]:
```
sb.reset_orig()
shap.plots.bar(XGB_shap)
```



[ ]:
```
shap.plots.beeswarm(XGB_shap)
```

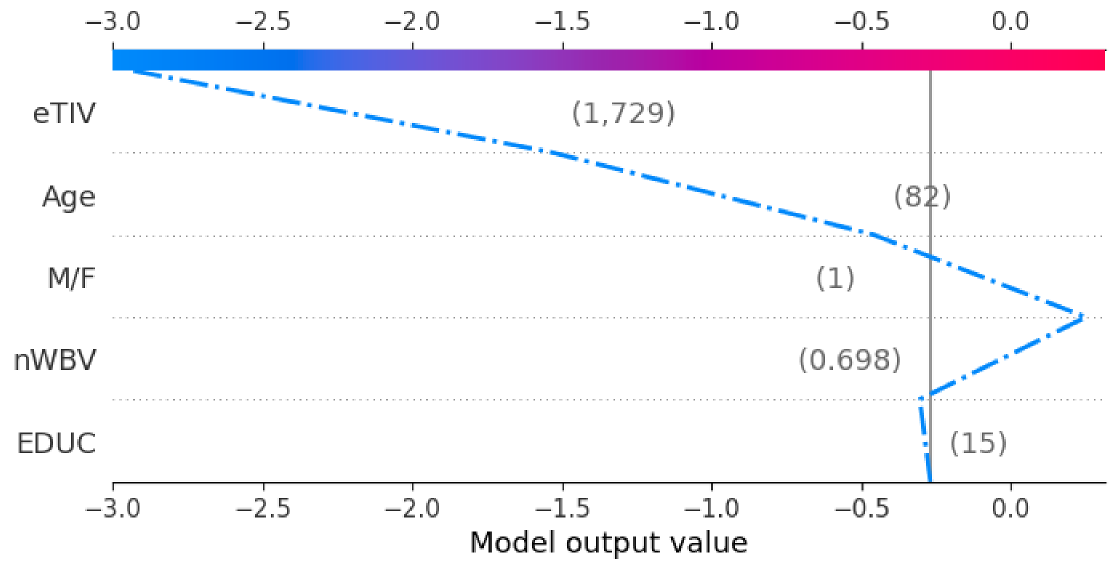


From the shap plot above, we can see that M/F(M=0, F=1) female and older age tend to push class to nondemented side.

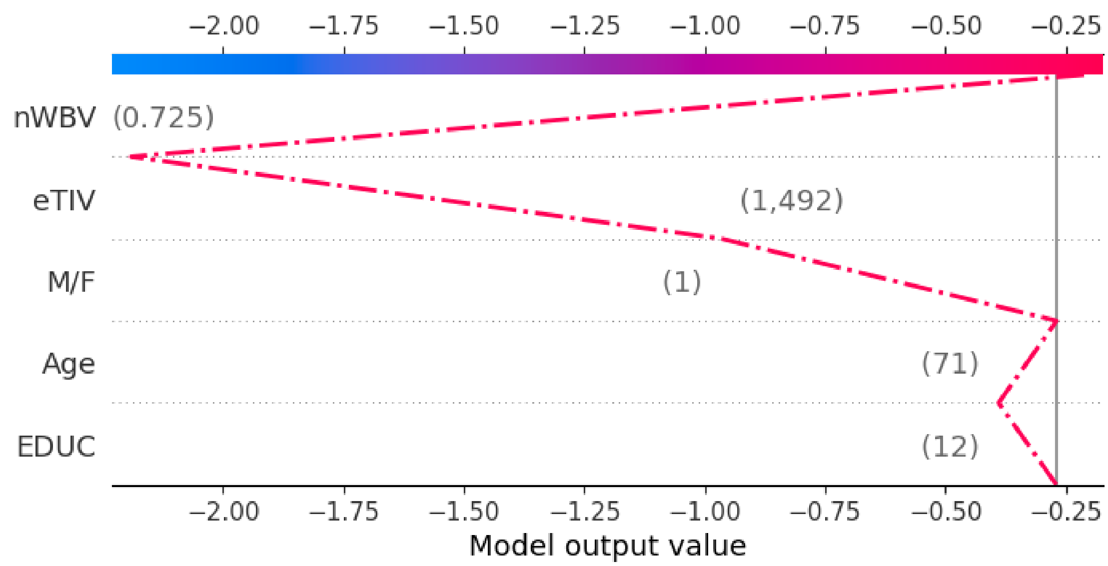
[ ]:
```
interaction_values = XGB_shap_interaction_values
features = X_test.columns

mean_abs_interactions = np.abs(interaction_values).mean(axis=0)

# Absolute mean measures how strong the interaction is, regardless of direction.

interaction_df = pd.DataFrame(mean_abs_interactions, index=features,
 ↪columns=features)
interaction_df = interaction_df.where(np.triu(np.ones(interaction_df.shape),
 ↪k=1).astype(bool))

interaction_df = interaction_df.stack().reset_index()
interaction_df.columns = ['Feature 1', 'Feature 2', 'Mean |Interaction Value|']
interaction_df = interaction_df.sort_values(by='Mean |Interaction Value|',
 ↪ascending=False)

print(interaction_df.head(10))
```

```
  Feature 1 Feature 2  Mean |Interaction Value|
5       Age      eTIV                  0.304100
9      eTIV      nWBV                  0.282089
7      EDUC      eTIV                  0.268376
8      EDUC      nWBV                  0.227692
6       Age      nWBV                  0.170623
1       M/F      EDUC                  0.157386
2       M/F      eTIV                  0.134400
4       Age      EDUC                  0.115562
0       M/F       Age                  0.094171
3       M/F      nWBV                  0.062230
```

[ ]:
```
# FN -31
expected_value = XGB_explainer.expected_value
shap.decision_plot(expected_value, XGB_shap.values[31], X_test.iloc[31],
 ↪highlight=0)
```




```
[ ]: # FN -37
expected_value = XGB_explainer.expected_value
shap.decision_plot(expected_value, XGB_shap.values[37], X_test.iloc[16],␣
 ↪highlight=0)
```




```
[ ]: # FN -31
shap.initjs()
expected_value = XGB_explainer.expected_value
shap.force_plot(expected_value, XGB_shap.values[31], X_test.iloc[31])
```

<IPython.core.display.HTML object>

[ ]: <shap.plots._force.AdditiveForceVisualizer at 0x786ee8999350>

[ ]:
```
# FN -37
shap.initjs()
expected_value = XGB_explainer.expected_value
shap.force_plot(expected_value, XGB_shap.values[16], X_test.iloc[16])
```

<IPython.core.display.HTML object>

[ ]: <shap.plots._force.AdditiveForceVisualizer at 0x786ee9146010>

[ ]:
```
lime_XGB_explainer = lime.lime_tabular.LimeTabularExplainer(X_test.values,
                                                   feature_names=X_test.columns,
                                                   class_names=['Nondemented',␣
 ↪'Demented'])
```

[ ]:
```
# FN case -31
lime_XGB_explainer.explain_instance(X_test.iloc[31].values,\
                                    XGBoost_mdl.predict_proba,\
                                    num_features=5).\
                                show_in_notebook(predict_proba=True)
```

<IPython.core.display.HTML object>

[ ]:
```
# FN case-37
lime_XGB_explainer.explain_instance(X_test.iloc[37].values,\
                                    XGBoost_mdl.predict_proba,\
                                    num_features=5).\
                                show_in_notebook(predict_proba=True)
```

<IPython.core.display.HTML object>

[ ]:
```
# store all fn_results
fn_results = []
feature_counter = Counter()

FN_indices = [31, 37, 49, 61]

for fn_idx in FN_indices:
    instance_values = X_test.iloc[fn_idx].values

    exp = lime_XGB_explainer.explain_instance(
        instance_values,
        XGBoost_mdl.predict_proba,
        num_features=6
    )
```

```python
    exp_list = exp.as_list()

    pushed_non = [f for f, w in exp_list if w < 0]
    pushed_dem = [f for f, w in exp_list if w > 0]

    fn_results.append({
        'Index': fn_idx,
        'Pushed_Nondemented': pushed_non,
        'Pushed_Demented': pushed_dem
    })

    feature_counter.update(pushed_non)

fn_df = pd.DataFrame(fn_results)

top_causes = pd.DataFrame(feature_counter.most_common(), columns=['Feature',␣
↪'Count'])

print(fn_df.head())

print("\n=== False Negative Feature Frequency ===")
print(top_causes)
```

```
   Index                                 Pushed_Nondemented  \
0     31   [0.00 < M/F <= 1.00, Age > 80.25, eTIV > 1669.00]
1     37   [0.00 < M/F <= 1.00, eTIV > 1669.00, 75.00 < A…
2     49   [EDUC > 16.25, 0.73 < nWBV <= 0.76, 1491.50 < …
3     61   [0.00 < M/F <= 1.00, Age > 80.25, 0.73 < nWBV …

                                    Pushed_Demented
0              [nWBV <= 0.70, 12.00 < EDUC <= 15.00]
1              [nWBV <= 0.70, 12.00 < EDUC <= 15.00]
2                        [M/F <= 0.00, Age <= 71.00]
3  [1391.00 < eTIV <= 1491.50, 12.00 < EDUC <= 15…

=== False Negative Feature Frequency ===
                   Feature  Count
0         0.00 < M/F <= 1.00      3
1                Age > 80.25      2
2             eTIV > 1669.00      2
3         0.73 < nWBV <= 0.76      2
4        75.00 < Age <= 80.25      1
5               EDUC > 16.25      1
6  1491.50 < eTIV <= 1669.00      1
```

removing SES did't notably decrease XGB model's performance. Fewer FN cases(5–>4), recall is improved

7.3.3 Model training-LightGBM_remove SES

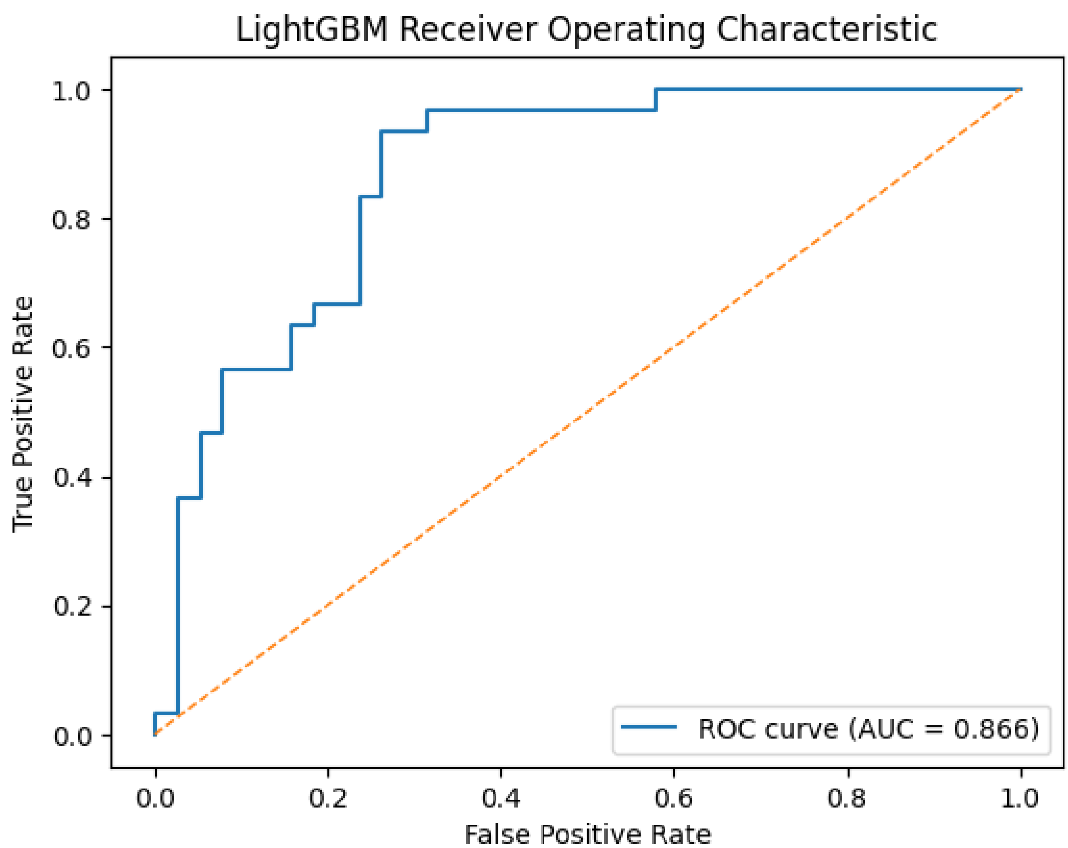
[ ]:
```
LightGBM = LGBMClassifier(random_state=42, verbosity=-1)

param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.001, 0.01, 0.1],
    'num_leaves': [31, 50, 70],
}

inner_cv = RepeatedStratifiedKFold(
    n_splits=5, n_repeats=2, random_state=42
)

grid = GridSearchCV(
    estimator=LightGBM,
    param_grid=param_grid,
    scoring='f1',
    cv=inner_cv,
    n_jobs=-1,
    verbose=0
)

grid.fit(X_train, y_train)
best_model = grid.best_estimator_
print("LightGBM classifier:")
print("Best params:", grid.best_params_)

scoring = {
    'accuracy': 'accuracy',
    'precision': 'precision',
    'recall': 'recall',
    'f1': 'f1',
    'roc_auc': 'roc_auc'
}
cv_res = cross_validate(
    best_model, X_train, y_train,
    cv=inner_cv, scoring=scoring, n_jobs=-1, return_train_score=False
)

def show(name):
    vals = cv_res[f'test_{name}']
    print(f"CV {name:<9}: {vals.mean():.4f} ± {vals.std():.4f}")

print("\n=== Repeated Stratified 5×2 CV on TRAIN ===")
for m in ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']:
```

```
    show(m)

y_pred = best_model.predict(X_test)
y_prob = best_model.predict_proba(X_test)[:, 1]

acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_prob)

print("\n=== Final Performance on HOLD-OUT TEST ===")
print(f"Accuracy : {acc:.4f}")
print(f"Precision: {prec:.4f}")
print(f"Recall   : {rec:.4f}")
print(f"F1 Score : {f1:.4f}")
print(f"ROC_AUC  : {auc:.4f}")

print("\nClassification Report on Test Set:\n", classification_report(y_test,
 ↪y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

feat_imp = pd.DataFrame({
    "Feature": X_train.columns,
    "Importance": best_model.feature_importances_
}).sort_values(by="Importance", ascending=False)
print("\nTop feature importances:\n", feat_imp)

LightGBM_mdl = best_model
```

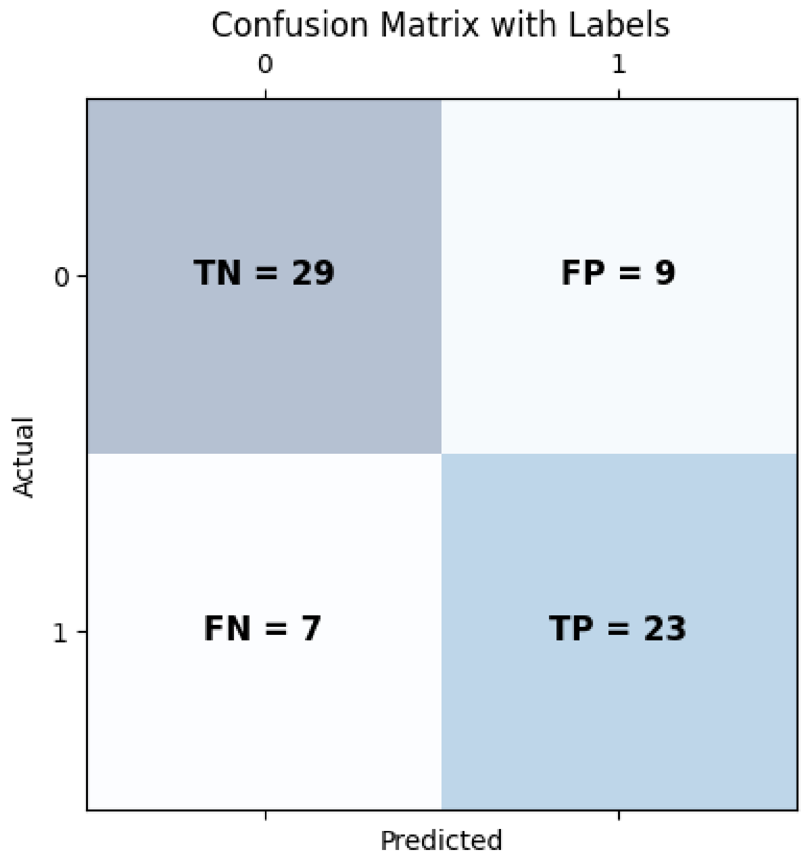
```
LightGBM classifier:
Best params: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 200,
'num_leaves': 31}

=== Repeated Stratified 5×2 CV on TRAIN ===
CV accuracy : 0.7783 ± 0.0724
CV precision: 0.7604 ± 0.1032
CV recall   : 0.7335 ± 0.0954
CV f1       : 0.7416 ± 0.0783
CV roc_auc  : 0.8376 ± 0.0706

=== Final Performance on HOLD-OUT TEST ===
Accuracy : 0.7647
Precision: 0.7188
Recall   : 0.7667
F1 Score : 0.7419
ROC_AUC  : 0.8658
```

```
Classification Report on Test Set:
              precision    recall  f1-score   support

           0       0.81      0.76      0.78        38
           1       0.72      0.77      0.74        30

    accuracy                           0.76        68
   macro avg       0.76      0.76      0.76        68
weighted avg       0.77      0.76      0.77        68

Confusion Matrix:
 [[29  9]
 [ 7 23]]

Top feature importances:
   Feature  Importance
3     eTIV         617
4     nWBV         434
1      Age         337
2     EDUC         194
0      M/F          68
```

[ ]:
```
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = roc_auc_score(y_test, y_prob)

plt.figure()
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc_auc:.3f})')
plt.plot([0, 1], [0, 1], linestyle='--', linewidth=1)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('LightGBM Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()
```



[ ]:
```
cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()

labels = np.array([
    [f"TN = {tn}", f"FP = {fp}"],
    [f"FN = {fn}", f"TP = {tp}"]
])

fig, ax = plt.subplots()
ax.matshow(cm, cmap=plt.cm.Blues, alpha=0.3)
for i in range(2):
    for j in range(2):
        ax.text(j, i, labels[i, j],
                va='center', ha='center', fontsize=12, fontweight='bold')

plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix with Labels")
plt.show()
```




```
[ ]: # Find all FN indices in the full test set
     FN_all = (~y_pred) & (y_test == 1)
     FN_indices = y_test[FN_all].index
     print("False Negative indices:", FN_indices)
```

```
False Negative indices: Index([154, 300, 299, 39, 51, 94, 329], dtype='int64')
```

```
[ ]: FN_sample_test_idx = X_test.index.get_indexer_for([154, 300, 299, 39, 51, 94,␣
      ↪329])
     FN_sample_test_idx
```

```
[ ]: array([ 9, 31, 37, 41, 49, 61, 64])
```

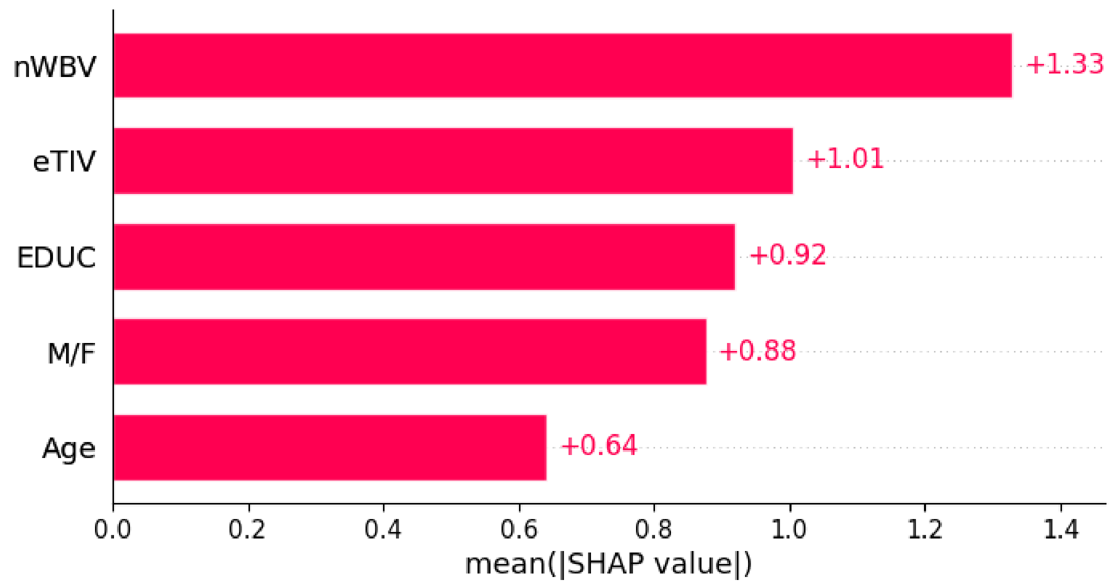
```
[ ]: LGB_explainer = shap.Explainer(LightGBM_mdl)
     LGB_shap = LGB_explainer(X_test)
     print(type(LGB_explainer))
```

```
<class 'shap.explainers._tree.TreeExplainer'>
```

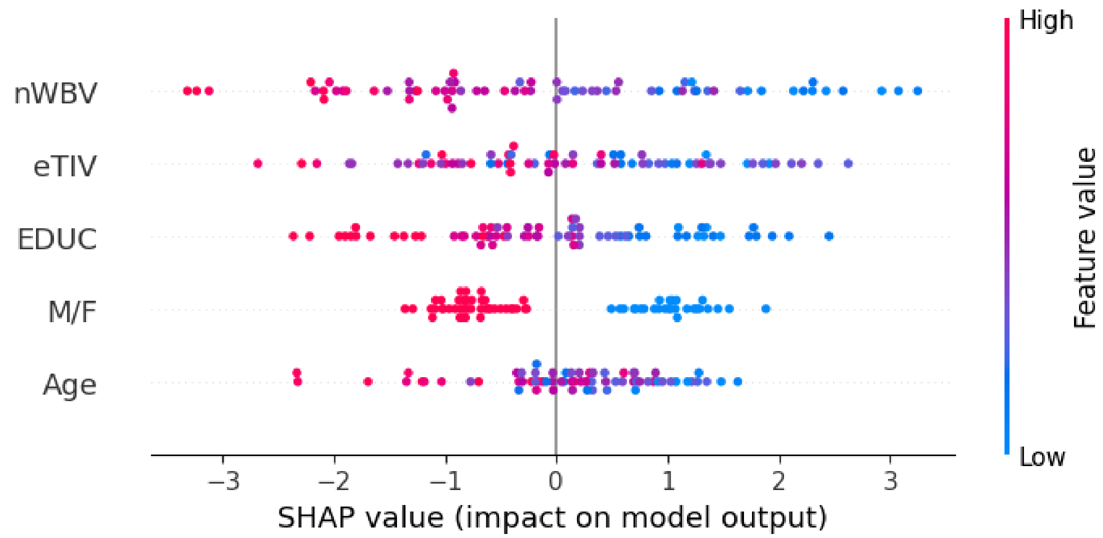
```
[ ]: print("Values dimensions: %s" % (LGB_shap.values.shape,))
     print("Data dimensions:   %s" % (LGB_shap.data.shape,))
```

```
Values dimensions: (68, 5)
Data dimensions:   (68, 5)
```

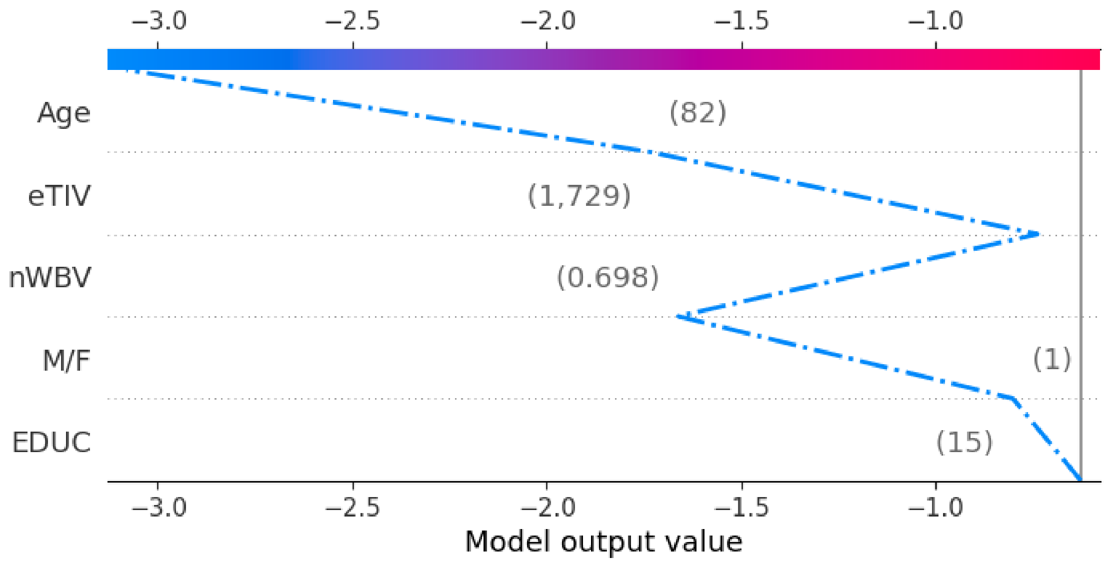
```
[ ]: sb.reset_orig()
     shap.plots.bar(LGB_shap)
```



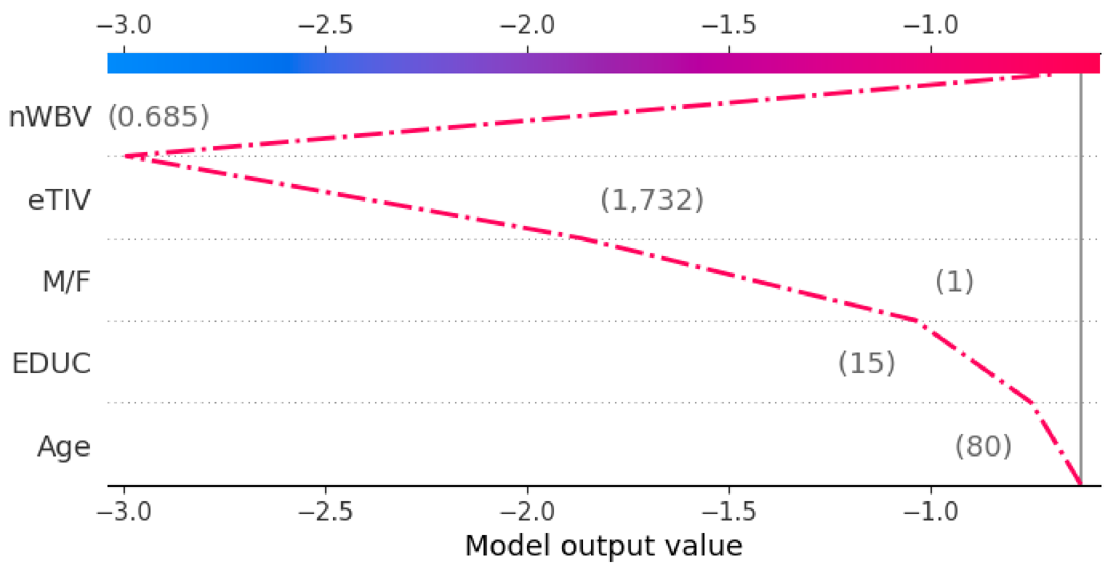

```
[ ]: shap.plots.beeswarm(LGB_shap)
```



```
[ ]: # LGB-FN-31
sb.reset_orig()
expected_value = LGB_explainer.expected_value
shap.decision_plot(expected_value, LGB_shap.values[31], X_test.iloc[31],
    highlight=0)
```




```
[ ]: # LGB-FN-37
expected_value = LGB_explainer.expected_value
shap.decision_plot(expected_value, LGB_shap.values[37], X_test.iloc[37],
    highlight=0)
```



```
[ ]: # LGB-FN-31
shap.initjs()
expected_value = LGB_explainer.expected_value
shap.force_plot(expected_value, LGB_shap.values[31], X_test.iloc[31])
```

```
<IPython.core.display.HTML object>
```

```
[ ]: <shap.plots._force.AdditiveForceVisualizer at 0x786ee95a1110>
```

```
[ ]: # LGB-FN-37
shap.initjs()
expected_value = LGB_explainer.expected_value
shap.force_plot(expected_value, LGB_shap.values[37], X_test.iloc[37])
```

```
<IPython.core.display.HTML object>
```

```
[ ]: <shap.plots._force.AdditiveForceVisualizer at 0x786eea188450>
```

```
[ ]: lime_LGB_explainer = lime.lime_tabular.LimeTabularExplainer(X_test.values,
                                                               feature_names=X_test.columns,
                                                               class_names=['Nondemented',␣
 ↪'Demented'])
```

```
[ ]: # LGB-FN-31
lime_LGB_explainer.explain_instance(X_test.iloc[31].values,\
                                    LightGBM_mdl.predict_proba,\
                                    num_features=6).\
                                show_in_notebook(predict_proba=True)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739:
UserWarning: X does not have valid feature names, but LGBMClassifier was fitted
with feature names
  warnings.warn(
```

```
<IPython.core.display.HTML object>
```

```
[ ]: # LGB-FN-37
lime_LGB_explainer.explain_instance(X_test.iloc[37].values,\
                                    LightGBM_mdl.predict_proba,\
                                    num_features=6).\
                                show_in_notebook(predict_proba=True)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739:
UserWarning: X does not have valid feature names, but LGBMClassifier was fitted
with feature names
  warnings.warn(
```

```
<IPython.core.display.HTML object>
```

[ ]:
```
# LGB-FN-49
lime_LGB_explainer.explain_instance(X_test.iloc[49].values,\
                                    LightGBM_mdl.predict_proba,\
                                    num_features=6).\
                                show_in_notebook(predict_proba=True)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739:
UserWarning: X does not have valid feature names, but LGBMClassifier was fitted
with feature names
  warnings.warn(
```

```
<IPython.core.display.HTML object>
```

[ ]:
```
# store all fn_results
fn_results = []
feature_counter = Counter()

LGB_FN_indices = [ 9, 31, 37, 41, 49, 61, 64]

for fn_idx in LGB_FN_indices:
    LGB_instance_values = X_test.iloc[fn_idx].values

    exp = lime_LGB_explainer.explain_instance(
        LGB_instance_values,
        LightGBM_mdl.predict_proba,
        num_features=6
    )

    exp_list = exp.as_list()

    pushed_non = [f for f, w in exp_list if w < 0]
    pushed_dem = [f for f, w in exp_list if w > 0]

    fn_results.append({
        'Index': fn_idx,
        'Pushed_Nondemented': pushed_non,
        'Pushed_Demented': pushed_dem
    })

    feature_counter.update(pushed_non)

fn_df = pd.DataFrame(fn_results)

top_causes = pd.DataFrame(feature_counter.most_common(), columns=['Feature',␣
 ↪'Count'])
```

```python
print(fn_df.head())

print("\n=== LGB False Negative Feature Frequency ===")
print(top_causes)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739:
UserWarning: X does not have valid feature names, but LGBMClassifier was fitted
with feature names
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739:
UserWarning: X does not have valid feature names, but LGBMClassifier was fitted
with feature names
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739:
UserWarning: X does not have valid feature names, but LGBMClassifier was fitted
with feature names
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739:
UserWarning: X does not have valid feature names, but LGBMClassifier was fitted
with feature names
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739:
UserWarning: X does not have valid feature names, but LGBMClassifier was fitted
with feature names
  warnings.warn(
```

```
   Index                                Pushed_Nondemented  \
0      9                    [0.00 < M/F <= 1.00, Age > 80.25]
1     31   [0.00 < M/F <= 1.00, eTIV > 1669.00, Age > 80.25]
2     37   [0.00 < M/F <= 1.00, eTIV > 1669.00, 75.00 < A…
3     41   [EDUC > 16.25, eTIV > 1669.00, 71.00 < Age <= …
4     49   [EDUC > 16.25, 0.73 < nWBV <= 0.76, 1491.50 < …

                                    Pushed_Demented
0  [nWBV <= 0.70, eTIV <= 1391.00, 12.00 < EDUC <…
1              [nWBV <= 0.70, 12.00 < EDUC <= 15.00]
2              [nWBV <= 0.70, 12.00 < EDUC <= 15.00]
3                 [M/F <= 0.00, 0.70 < nWBV <= 0.73]
4                        [M/F <= 0.00, Age <= 71.00]

=== LGB False Negative Feature Frequency ===
                  Feature  Count
0         0.00 < M/F <= 1.00      4
1             eTIV > 1669.00      4
2                Age > 80.25      3
3               EDUC > 16.25      3
4         0.73 < nWBV <= 0.76      3
5       75.00 < Age <= 80.25      2
```

```
6       71.00 < Age <= 75.00      1
7   1491.50 < eTIV <= 1669.00      1
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739:
UserWarning: X does not have valid feature names, but LGBMClassifier was fitted
with feature names
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739:
UserWarning: X does not have valid feature names, but LGBMClassifier was fitted
with feature names
  warnings.warn(
```

compared with before removing SES, performance of LightGBM decreases. More FN cases(4->7)

7.3.4 Model training-CatBoost_remove SES

[ ]:
```
CatBoost = CatBoostClassifier(random_state=42)

param_grid = {
    'min_data_in_leaf': [20, 40, 60],
    'rsm': [0.7, 0.8, 1.0],
    'iterations': [100, 200],
    'depth': [3, 5, 7],
    'learning_rate': [0.001, 0.01, 0.1],
    'l2_leaf_reg': [1, 3, 5],
    'bagging_temperature': [0, 0.5, 1.0],
}

inner_cv = RepeatedStratifiedKFold(
    n_splits=5, n_repeats=2, random_state=42
)

grid = GridSearchCV(
    estimator=CatBoost,
    param_grid=param_grid,
    scoring='f1',
    cv=inner_cv,
    n_jobs=-1,
    verbose=0
)

grid.fit(X_train, y_train)
best_model = grid.best_estimator_
print("CatBoostClassifier:")
print("Best params:", grid.best_params_)

scoring = {
    'accuracy': 'accuracy',
```

```
    'precision': 'precision',
    'recall': 'recall',
    'f1': 'f1',
    'roc_auc': 'roc_auc'
}
cv_res = cross_validate(
    best_model, X_train, y_train,
    cv=inner_cv, scoring=scoring, n_jobs=-1, return_train_score=False
)

def show(name):
    vals = cv_res[f'test_{name}']
    print(f"CV {name:<9}: {vals.mean():.4f} ± {vals.std():.4f}")

print("\n=== Repeated Stratified 5×2 CV on TRAIN ===")
for m in ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']:
    show(m)

y_pred = best_model.predict(X_test)
y_prob = best_model.predict_proba(X_test)[:, 1]

acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_prob)

print("\n=== Final Performance on HOLD-OUT TEST ===")
print(f"Accuracy : {acc:.4f}")
print(f"Precision: {prec:.4f}")
print(f"Recall   : {rec:.4f}")
print(f"F1 Score : {f1:.4f}")
print(f"ROC_AUC  : {auc:.4f}")

print("\nClassification Report on Test Set:\n", classification_report(y_test,␣
 ↪y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

feat_imp = pd.DataFrame({
    "Feature": X_train.columns,
    "Importance": best_model.feature_importances_
}).sort_values(by="Importance", ascending=False)
print("\nTop feature importances:\n", feat_imp)

CatBoost_mdl = best_model
```

```
0:      learn: 0.6593913        total: 47.4ms   remaining: 9.44s
```

```
1:      learn: 0.6161128        total: 48.4ms   remaining: 4.79s
2:      learn: 0.5747853        total: 49.4ms   remaining: 3.25s
3:      learn: 0.5599539        total: 50ms     remaining: 2.45s
4:      learn: 0.5253404        total: 51.1ms   remaining: 1.99s
5:      learn: 0.5001249        total: 52.1ms   remaining: 1.68s
6:      learn: 0.4951362        total: 52.7ms   remaining: 1.45s
7:      learn: 0.4747394        total: 53.9ms   remaining: 1.29s
8:      learn: 0.4546658        total: 55ms     remaining: 1.17s
9:      learn: 0.4454416        total: 55.6ms   remaining: 1.06s
10:     learn: 0.4294976        total: 56.5ms   remaining: 971ms
11:     learn: 0.4152974        total: 57.5ms   remaining: 900ms
12:     learn: 0.3948456        total: 58.4ms   remaining: 840ms
13:     learn: 0.3842047        total: 59.2ms   remaining: 787ms
14:     learn: 0.3773444        total: 60.1ms   remaining: 741ms
15:     learn: 0.3576999        total: 61.1ms   remaining: 702ms
16:     learn: 0.3499457        total: 62ms     remaining: 668ms
17:     learn: 0.3356367        total: 63.1ms   remaining: 638ms
18:     learn: 0.3232779        total: 64ms     remaining: 610ms
19:     learn: 0.3199074        total: 64.6ms   remaining: 582ms
20:     learn: 0.3055672        total: 65.7ms   remaining: 560ms
21:     learn: 0.2965582        total: 66.7ms   remaining: 540ms
22:     learn: 0.2830015        total: 67.9ms   remaining: 522ms
23:     learn: 0.2797083        total: 69.2ms   remaining: 507ms
24:     learn: 0.2692293        total: 70.2ms   remaining: 492ms
25:     learn: 0.2608171        total: 71.3ms   remaining: 477ms
26:     learn: 0.2554928        total: 72.3ms   remaining: 463ms
27:     learn: 0.2477168        total: 73.3ms   remaining: 450ms
28:     learn: 0.2382749        total: 74.3ms   remaining: 438ms
29:     learn: 0.2352668        total: 75.2ms   remaining: 426ms
30:     learn: 0.2282407        total: 76.2ms   remaining: 415ms
31:     learn: 0.2227499        total: 77.2ms   remaining: 405ms
32:     learn: 0.2161551        total: 78.1ms   remaining: 395ms
33:     learn: 0.2127196        total: 79.2ms   remaining: 387ms
34:     learn: 0.2070432        total: 80.1ms   remaining: 378ms
35:     learn: 0.2017828        total: 81.2ms   remaining: 370ms
36:     learn: 0.1926881        total: 82.2ms   remaining: 362ms
37:     learn: 0.1885964        total: 83.2ms   remaining: 355ms
38:     learn: 0.1843307        total: 84.3ms   remaining: 348ms
39:     learn: 0.1812445        total: 85.3ms   remaining: 341ms
40:     learn: 0.1789055        total: 86.2ms   remaining: 334ms
41:     learn: 0.1770552        total: 87.4ms   remaining: 329ms
42:     learn: 0.1769425        total: 87.9ms   remaining: 321ms
43:     learn: 0.1749386        total: 88.8ms   remaining: 315ms
44:     learn: 0.1749027        total: 89.3ms   remaining: 308ms
45:     learn: 0.1717094        total: 90.2ms   remaining: 302ms
46:     learn: 0.1689454        total: 91.2ms   remaining: 297ms
47:     learn: 0.1684493        total: 91.9ms   remaining: 291ms
48:     learn: 0.1652703        total: 92.8ms   remaining: 286ms
```

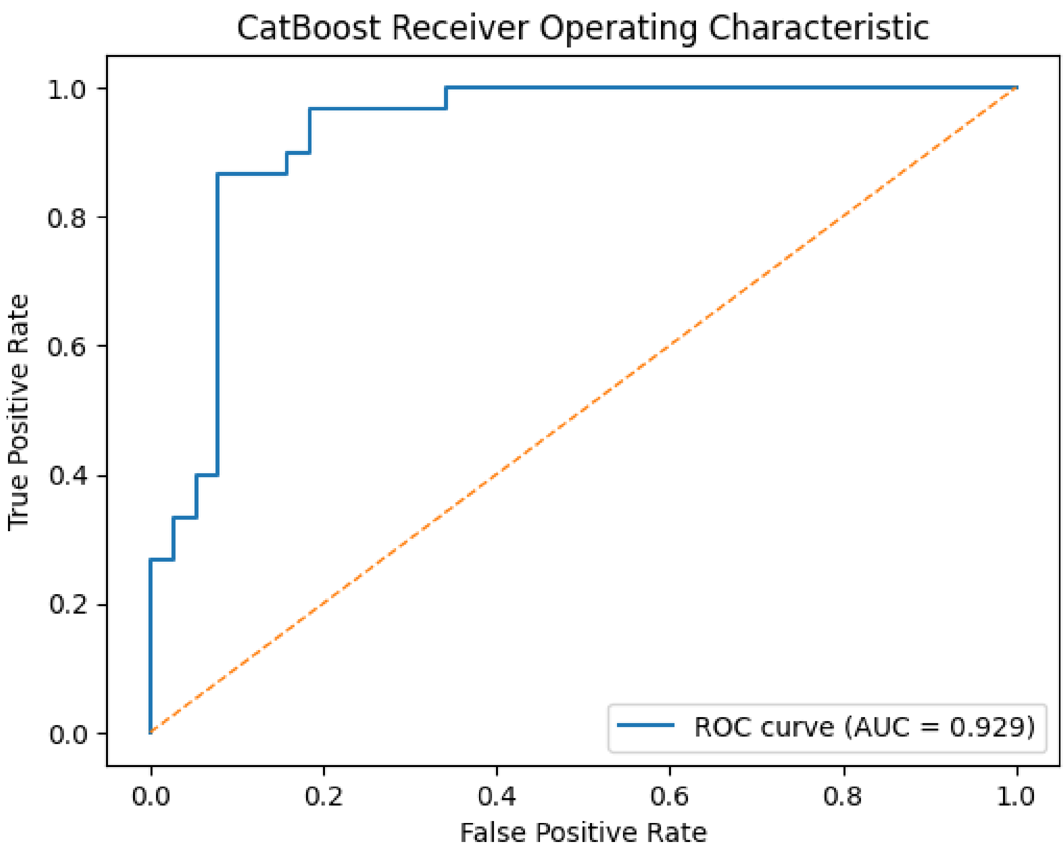
```
49:	learn: 0.1602030	total: 93.8ms	remaining: 281ms
50:	learn: 0.1569368	total: 94.7ms	remaining: 277ms
51:	learn: 0.1532159	total: 95.7ms	remaining: 272ms
52:	learn: 0.1470671	total: 96.7ms	remaining: 268ms
53:	learn: 0.1443544	total: 97.8ms	remaining: 265ms
54:	learn: 0.1423886	total: 99ms	remaining: 261ms
55:	learn: 0.1357377	total: 100ms	remaining: 257ms
56:	learn: 0.1304753	total: 101ms	remaining: 254ms
57:	learn: 0.1277747	total: 102ms	remaining: 250ms
58:	learn: 0.1250609	total: 103ms	remaining: 247ms
59:	learn: 0.1200222	total: 104ms	remaining: 243ms
60:	learn: 0.1170928	total: 105ms	remaining: 239ms
61:	learn: 0.1144087	total: 106ms	remaining: 236ms
62:	learn: 0.1119146	total: 107ms	remaining: 233ms
63:	learn: 0.1101312	total: 108ms	remaining: 230ms
64:	learn: 0.1078354	total: 109ms	remaining: 227ms
65:	learn: 0.1064887	total: 110ms	remaining: 224ms
66:	learn: 0.1042670	total: 111ms	remaining: 221ms
67:	learn: 0.0997478	total: 112ms	remaining: 218ms
68:	learn: 0.0978650	total: 113ms	remaining: 215ms
69:	learn: 0.0968160	total: 114ms	remaining: 212ms
70:	learn: 0.0952747	total: 115ms	remaining: 209ms
71:	learn: 0.0936632	total: 116ms	remaining: 206ms
72:	learn: 0.0911765	total: 117ms	remaining: 204ms
73:	learn: 0.0877513	total: 118ms	remaining: 201ms
74:	learn: 0.0860806	total: 119ms	remaining: 199ms
75:	learn: 0.0849196	total: 120ms	remaining: 196ms
76:	learn: 0.0836052	total: 121ms	remaining: 194ms
77:	learn: 0.0809637	total: 122ms	remaining: 192ms
78:	learn: 0.0797554	total: 124ms	remaining: 189ms
79:	learn: 0.0769329	total: 125ms	remaining: 187ms
80:	learn: 0.0760161	total: 126ms	remaining: 185ms
81:	learn: 0.0752541	total: 127ms	remaining: 183ms
82:	learn: 0.0740942	total: 128ms	remaining: 180ms
83:	learn: 0.0726738	total: 129ms	remaining: 178ms
84:	learn: 0.0708871	total: 130ms	remaining: 176ms
85:	learn: 0.0698076	total: 131ms	remaining: 174ms
86:	learn: 0.0685995	total: 132ms	remaining: 172ms
87:	learn: 0.0675179	total: 133ms	remaining: 170ms
88:	learn: 0.0662989	total: 134ms	remaining: 167ms
89:	learn: 0.0648178	total: 135ms	remaining: 165ms
90:	learn: 0.0637765	total: 137ms	remaining: 164ms
91:	learn: 0.0626972	total: 139ms	remaining: 163ms
92:	learn: 0.0617175	total: 140ms	remaining: 161ms
93:	learn: 0.0609860	total: 141ms	remaining: 159ms
94:	learn: 0.0596594	total: 142ms	remaining: 157ms
95:	learn: 0.0587644	total: 143ms	remaining: 155ms
96:	learn: 0.0574071	total: 144ms	remaining: 153ms
```

```
97:     learn: 0.0566022        total: 145ms    remaining: 151ms
98:     learn: 0.0557687        total: 147ms    remaining: 150ms
99:     learn: 0.0546145        total: 148ms    remaining: 148ms
100:    learn: 0.0538387        total: 149ms    remaining: 146ms
101:    learn: 0.0530472        total: 150ms    remaining: 144ms
102:    learn: 0.0519145        total: 151ms    remaining: 142ms
103:    learn: 0.0511723        total: 152ms    remaining: 140ms
104:    learn: 0.0503818        total: 153ms    remaining: 138ms
105:    learn: 0.0494722        total: 154ms    remaining: 137ms
106:    learn: 0.0485186        total: 155ms    remaining: 135ms
107:    learn: 0.0477060        total: 156ms    remaining: 133ms
108:    learn: 0.0468543        total: 157ms    remaining: 131ms
109:    learn: 0.0456000        total: 158ms    remaining: 129ms
110:    learn: 0.0443978        total: 159ms    remaining: 128ms
111:    learn: 0.0435653        total: 160ms    remaining: 126ms
112:    learn: 0.0432561        total: 161ms    remaining: 124ms
113:    learn: 0.0419331        total: 162ms    remaining: 122ms
114:    learn: 0.0417388        total: 163ms    remaining: 121ms
115:    learn: 0.0409675        total: 164ms    remaining: 119ms
116:    learn: 0.0405151        total: 165ms    remaining: 117ms
117:    learn: 0.0399008        total: 166ms    remaining: 115ms
118:    learn: 0.0384224        total: 167ms    remaining: 114ms
119:    learn: 0.0377821        total: 168ms    remaining: 112ms
120:    learn: 0.0374588        total: 169ms    remaining: 110ms
121:    learn: 0.0367639        total: 170ms    remaining: 109ms
122:    learn: 0.0365966        total: 171ms    remaining: 107ms
123:    learn: 0.0361286        total: 172ms    remaining: 105ms
124:    learn: 0.0356571        total: 173ms    remaining: 104ms
125:    learn: 0.0352078        total: 174ms    remaining: 102ms
126:    learn: 0.0345354        total: 175ms    remaining: 101ms
127:    learn: 0.0339649        total: 176ms    remaining: 98.9ms
128:    learn: 0.0335745        total: 177ms    remaining: 97.4ms
129:    learn: 0.0325093        total: 178ms    remaining: 95.9ms
130:    learn: 0.0319434        total: 179ms    remaining: 94.3ms
131:    learn: 0.0312770        total: 180ms    remaining: 92.7ms
132:    learn: 0.0310424        total: 181ms    remaining: 91.2ms
133:    learn: 0.0301217        total: 182ms    remaining: 89.6ms
134:    learn: 0.0298561        total: 183ms    remaining: 88.1ms
135:    learn: 0.0294238        total: 184ms    remaining: 86.5ms
136:    learn: 0.0289752        total: 185ms    remaining: 85ms
137:    learn: 0.0286738        total: 186ms    remaining: 83.6ms
138:    learn: 0.0284155        total: 187ms    remaining: 82ms
139:    learn: 0.0282291        total: 188ms    remaining: 80.5ms
140:    learn: 0.0278695        total: 189ms    remaining: 79.1ms
141:    learn: 0.0275022        total: 190ms    remaining: 77.6ms
142:    learn: 0.0273527        total: 191ms    remaining: 76.1ms
143:    learn: 0.0270758        total: 192ms    remaining: 74.6ms
144:    learn: 0.0268138        total: 193ms    remaining: 73.3ms
```

```
145:    learn: 0.0261164        total: 194ms    remaining: 71.9ms
146:    learn: 0.0254907        total: 195ms    remaining: 70.5ms
147:    learn: 0.0253650        total: 196ms    remaining: 69ms
148:    learn: 0.0251657        total: 197ms    remaining: 67.6ms
149:    learn: 0.0248262        total: 198ms    remaining: 66.1ms
150:    learn: 0.0246040        total: 199ms    remaining: 64.7ms
151:    learn: 0.0243856        total: 200ms    remaining: 63.3ms
152:    learn: 0.0243007        total: 201ms    remaining: 61.8ms
153:    learn: 0.0240222        total: 202ms    remaining: 60.4ms
154:    learn: 0.0239106        total: 203ms    remaining: 59ms
155:    learn: 0.0235797        total: 204ms    remaining: 57.6ms
156:    learn: 0.0233511        total: 205ms    remaining: 56.2ms
157:    learn: 0.0231489        total: 206ms    remaining: 54.8ms
158:    learn: 0.0228279        total: 207ms    remaining: 53.4ms
159:    learn: 0.0225688        total: 208ms    remaining: 52.1ms
160:    learn: 0.0220480        total: 209ms    remaining: 50.7ms
161:    learn: 0.0217246        total: 210ms    remaining: 49.3ms
162:    learn: 0.0215687        total: 211ms    remaining: 48ms
163:    learn: 0.0213136        total: 212ms    remaining: 46.6ms
164:    learn: 0.0211194        total: 213ms    remaining: 45.2ms
165:    learn: 0.0208881        total: 214ms    remaining: 43.8ms
166:    learn: 0.0207319        total: 215ms    remaining: 42.5ms
167:    learn: 0.0205932        total: 216ms    remaining: 41.1ms
168:    learn: 0.0205057        total: 217ms    remaining: 39.7ms
169:    learn: 0.0204389        total: 217ms    remaining: 38.4ms
170:    learn: 0.0200581        total: 219ms    remaining: 37.1ms
171:    learn: 0.0199939        total: 220ms    remaining: 35.7ms
172:    learn: 0.0198272        total: 221ms    remaining: 34.4ms
173:    learn: 0.0195395        total: 222ms    remaining: 33.1ms
174:    learn: 0.0194205        total: 223ms    remaining: 31.8ms
175:    learn: 0.0192243        total: 224ms    remaining: 30.5ms
176:    learn: 0.0190095        total: 225ms    remaining: 29.2ms
177:    learn: 0.0188153        total: 226ms    remaining: 27.9ms
178:    learn: 0.0185343        total: 227ms    remaining: 26.6ms
179:    learn: 0.0182716        total: 228ms    remaining: 25.3ms
180:    learn: 0.0180765        total: 229ms    remaining: 24ms
181:    learn: 0.0179598        total: 231ms    remaining: 22.8ms
182:    learn: 0.0177268        total: 232ms    remaining: 21.5ms
183:    learn: 0.0175665        total: 234ms    remaining: 20.3ms
184:    learn: 0.0173862        total: 235ms    remaining: 19.1ms
185:    learn: 0.0171053        total: 237ms    remaining: 17.8ms
186:    learn: 0.0169377        total: 240ms    remaining: 16.7ms
187:    learn: 0.0168117        total: 241ms    remaining: 15.4ms
188:    learn: 0.0167093        total: 242ms    remaining: 14.1ms
189:    learn: 0.0166480        total: 244ms    remaining: 12.9ms
190:    learn: 0.0164774        total: 246ms    remaining: 11.6ms
191:    learn: 0.0164050        total: 247ms    remaining: 10.3ms
192:    learn: 0.0163232        total: 249ms    remaining: 9.04ms
```

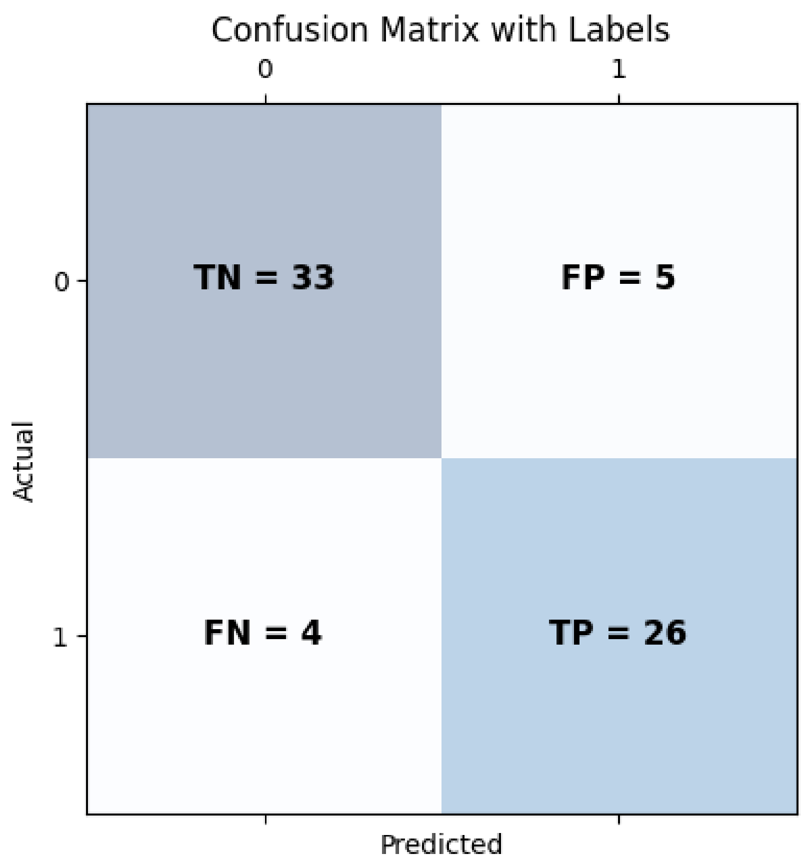
```
193:	learn: 0.0162641	total: 251ms	remaining: 7.76ms
194:	learn: 0.0161572	total: 253ms	remaining: 6.47ms
195:	learn: 0.0159926	total: 254ms	remaining: 5.17ms
196:	learn: 0.0158394	total: 255ms	remaining: 3.88ms
197:	learn: 0.0157380	total: 256ms	remaining: 2.58ms
198:	learn: 0.0156695	total: 257ms	remaining: 1.29ms
199:	learn: 0.0155247	total: 258ms	remaining: 0us
CatBoostClassifier:
Best params: {'bagging_temperature': 0, 'depth': 7, 'iterations': 200,
'l2_leaf_reg': 1, 'learning_rate': 0.1, 'min_data_in_leaf': 20, 'rsm': 0.8}

=== Repeated Stratified 5×2 CV on TRAIN ===
CV accuracy : 0.8267 ± 0.0677
CV precision: 0.8023 ± 0.0864
CV recall   : 0.8025 ± 0.0837
CV f1       : 0.8007 ± 0.0756
CV roc_auc  : 0.8841 ± 0.0492

=== Final Performance on HOLD-OUT TEST ===
Accuracy : 0.8676
Precision: 0.8387
Recall   : 0.8667
F1 Score : 0.8525
ROC_AUC  : 0.9289

Classification Report on Test Set:
              precision    recall  f1-score   support

           0       0.89      0.87      0.88        38
           1       0.84      0.87      0.85        30

    accuracy                           0.87        68
   macro avg       0.87      0.87      0.87        68
weighted avg       0.87      0.87      0.87        68

Confusion Matrix:
 [[33  5]
 [ 4 26]]

Top feature importances:
   Feature  Importance
3    eTIV   28.136046
2    EDUC   23.609229
4    nWBV   21.381349
1     Age   17.861711
0     M/F    9.011665
```

[ ]:
```python
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = roc_auc_score(y_test, y_prob)

plt.figure()
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc_auc:.3f})')
plt.plot([0, 1], [0, 1], linestyle='--', linewidth=1)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('CatBoost Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()
```



[ ]:
```python
cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()

labels = np.array([
    [f"TN = {tn}", f"FP = {fp}"],
    [f"FN = {fn}", f"TP = {tp}"]
])
```

```python
fig, ax = plt.subplots()
ax.matshow(cm, cmap=plt.cm.Blues, alpha=0.3)
for i in range(2):
    for j in range(2):
        ax.text(j, i, labels[i, j],
                va='center', ha='center', fontsize=12, fontweight='bold')

plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix with Labels")
plt.show()
```



[ ]:
```python
# Find all FN indices in the full test set
FN_all = (~y_pred) & (y_test == 1)
FN_indices = y_test[FN_all].index
print("False Negative indices:", FN_indices)
```

```
False Negative indices: Index([300, 299, 51, 94], dtype='int64')
```

```
[ ]: FN_sample_test_idx = X_test.index.get_indexer_for([300, 299, 51, 94])
     FN_sample_test_idx
```

```
[ ]: array([31, 37, 49, 61])
```

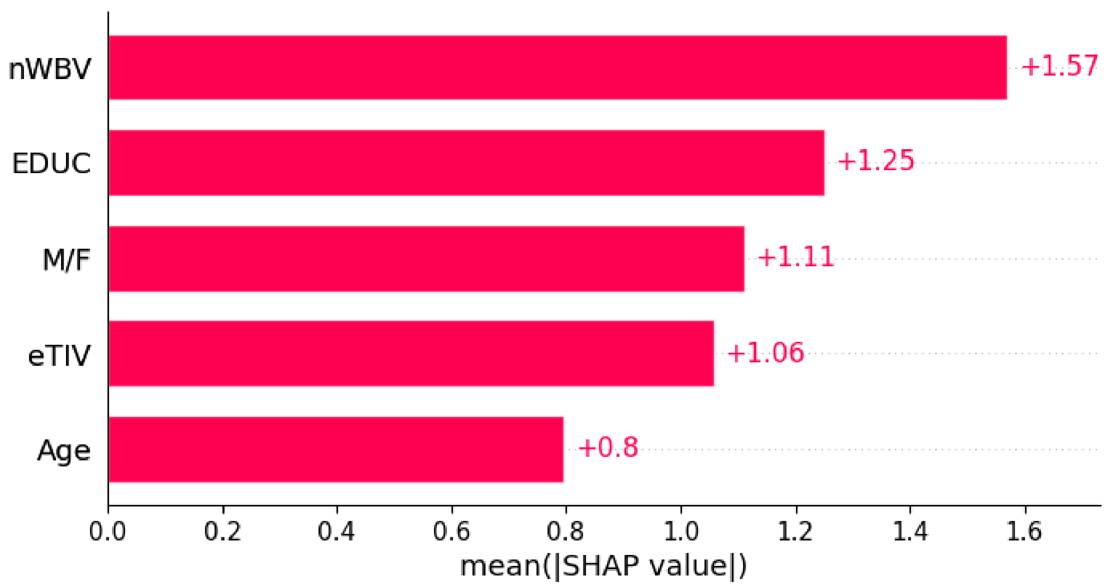
```
[ ]: CB_explainer = shap.Explainer(CatBoost_mdl)
     CB_shap = CB_explainer(X_test)
     print(type(CB_explainer))
```

```
<class 'shap.explainers._tree.TreeExplainer'>
```

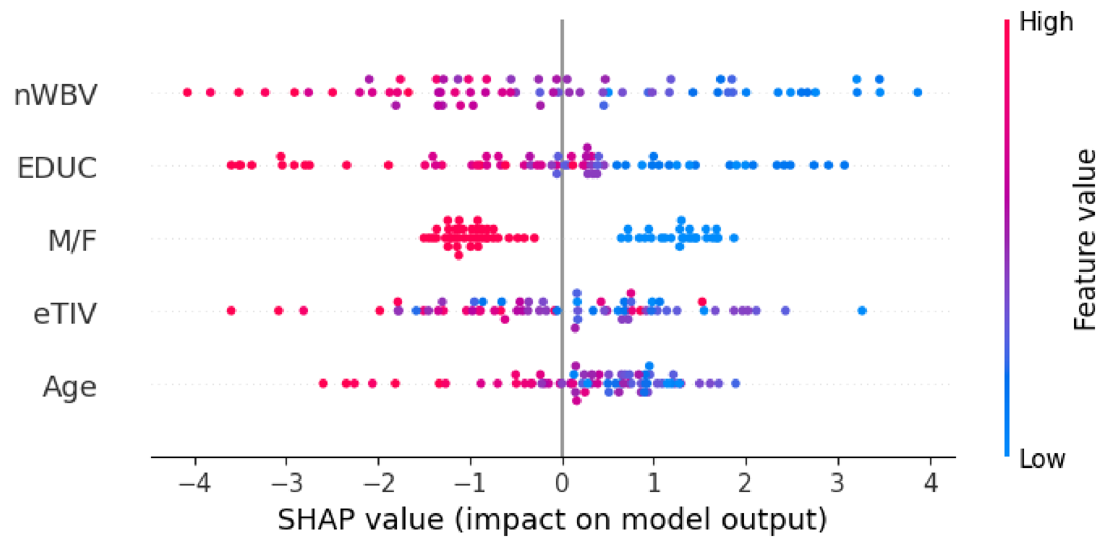
```
[ ]: print("Values dimensions: %s" % (CB_shap.values.shape,))
     print("Data dimensions:   %s" % (CB_shap.data.shape,))
```

```
Values dimensions: (68, 5)
Data dimensions:   (68, 5)
```

```
[ ]: sb.reset_orig()
     shap.plots.bar(CB_shap)
```




```
[ ]: shap.plots.beeswarm(CB_shap)
```

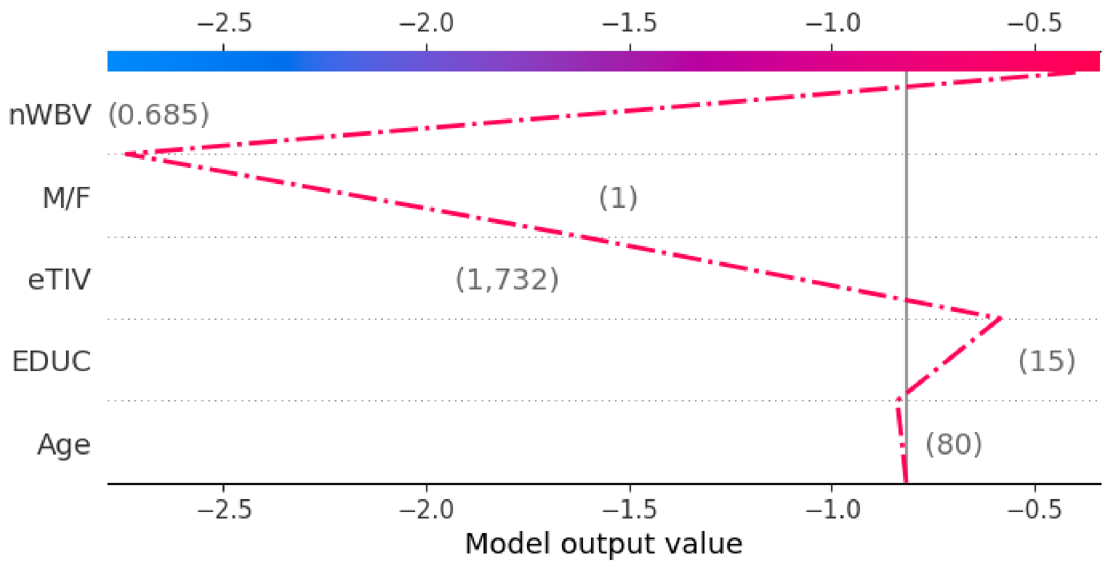


[ ]:
```
print("X_test.iloc[37]: ")
print(X_test.iloc[37])
print(y_test.iloc[37], y_pred[37])
print("-----------------")
print("X_test.iloc[31]: ")
print(X_test.iloc[31])
print(y_test.iloc[31], y_pred[31])
```

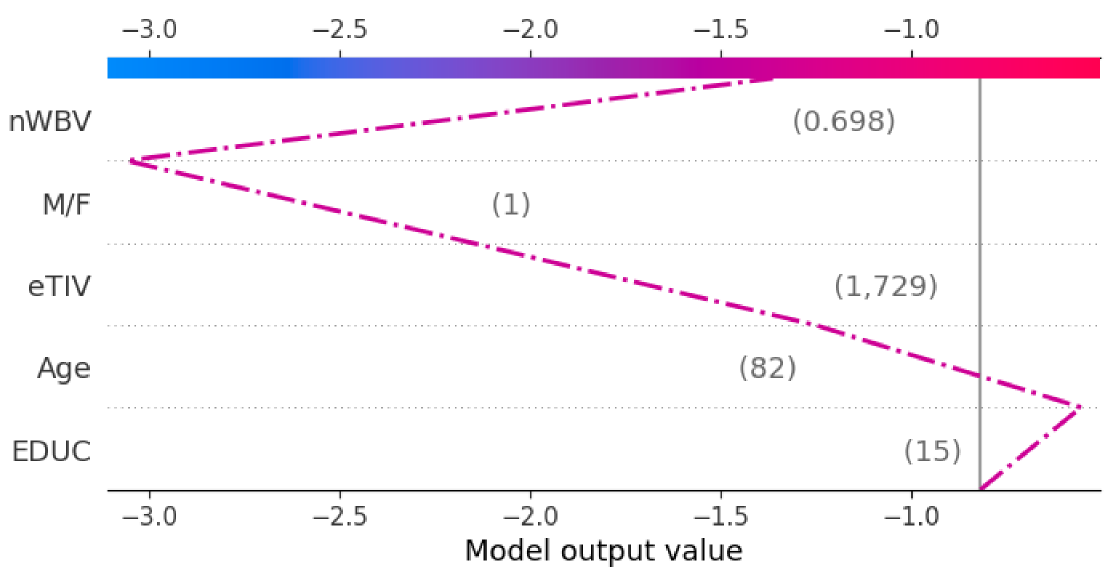
```
X_test.iloc[37]:
M/F          1.000
Age         80.000
EDUC        15.000
eTIV      1732.000
nWBV         0.685
Name: 299, dtype: float64
1 0
-----------------
X_test.iloc[31]:
M/F          1.000
Age         82.000
EDUC        15.000
eTIV      1729.000
nWBV         0.698
Name: 300, dtype: float64
1 0
```

[ ]:
```
# CB-FN-37
expected_value = CB_explainer.expected_value
```

```
shap.decision_plot(expected_value, CB_shap.values[37], X_test.iloc[37],␣
 ↪highlight=0)
```



[ ]:
```
# CB-FN-31
expected_value = CB_explainer.expected_value
shap.decision_plot(expected_value, CB_shap.values[31], X_test.iloc[31],␣
 ↪highlight=0)
```



```
[ ]: # CB-FN-37
shap.initjs()
expected_value = CB_explainer.expected_value
shap.force_plot(expected_value, CB_shap.values[37], X_test.iloc[37])
```

<IPython.core.display.HTML object>

```
[ ]: <shap.plots._force.AdditiveForceVisualizer at 0x786eec91ba50>
```

```
[ ]: # CB-FN-31
shap.initjs()
expected_value = CB_explainer.expected_value
shap.force_plot(expected_value, CB_shap.values[31], X_test.iloc[31])
```

<IPython.core.display.HTML object>

```
[ ]: <shap.plots._force.AdditiveForceVisualizer at 0x786eeb9ffe90>
```

```
[ ]: lime_CB_explainer = lime.lime_tabular.LimeTabularExplainer(X_test.values,
                                                   feature_names=X_test.columns,
                                                   class_names=['Nondemented',␣
 ↪'Demented'])
```

```
[ ]: # CB-FN-37
lime_CB_explainer.explain_instance(X_test.iloc[37].values,\
                                    CatBoost_mdl.predict_proba,\
                                    num_features=6).\
                                show_in_notebook(predict_proba=True)
```

<IPython.core.display.HTML object>

```
[ ]: # CB-FN-31
lime_CB_explainer.explain_instance(X_test.iloc[31].values,\
                                    CatBoost_mdl.predict_proba,\
                                    num_features=6).\
                                show_in_notebook(predict_proba=True)
```

<IPython.core.display.HTML object>

```
[ ]: # store all fn_results
fn_results = []
feature_counter = Counter()

CB_FN_indices = [31, 37, 49, 61]

for fn_idx in CB_FN_indices:
    CB_instance_values = X_test.iloc[fn_idx].values
```

```python
    exp = lime_CB_explainer.explain_instance(
        CB_instance_values,
        CatBoost_mdl.predict_proba,
        num_features=6
    )

    exp_list = exp.as_list()

    pushed_non = [f for f, w in exp_list if w < 0]
    pushed_dem = [f for f, w in exp_list if w > 0]

    fn_results.append({
        'Index': fn_idx,
        'Pushed_Nondemented': pushed_non,
        'Pushed_Demented': pushed_dem
    })

    feature_counter.update(pushed_non)

fn_df = pd.DataFrame(fn_results)

top_causes = pd.DataFrame(feature_counter.most_common(), columns=['Feature',␣
 ↪'Count'])

print(fn_df.head())

print("\n=== CB False Negative Feature Frequency ===")
print(top_causes)
```

```
   Index                                 Pushed_Nondemented  \
0     31  [0.00 < M/F <= 1.00, Age > 80.25, eTIV > 1669.00]
1     37  [0.00 < M/F <= 1.00, eTIV > 1669.00, 75.00 < A…
2     49  [EDUC > 16.25, 0.73 < nWBV <= 0.76, 1491.50 < …
3     61  [0.00 < M/F <= 1.00, Age > 80.25, 0.73 < nWBV …

                                    Pushed_Demented
0              [nWBV <= 0.70, 12.00 < EDUC <= 15.00]
1              [nWBV <= 0.70, 12.00 < EDUC <= 15.00]
2                         [M/F <= 0.00, Age <= 71.00]
3  [1391.00 < eTIV <= 1491.50, 12.00 < EDUC <= 15…

=== CB False Negative Feature Frequency ===
                 Feature  Count
0         0.00 < M/F <= 1.00      3
1                Age > 80.25      2
2             eTIV > 1669.00      2
3        0.73 < nWBV <= 0.76      2
```

```
4       75.00 < Age <= 80.25       1
5                EDUC > 16.25      1
6   1491.50 < eTIV <= 1669.00      1
```

removing SES did't notably decrease CatBoost model's performance. Fewer FN cases(6–>4), recall is improved

7.3.5 Model training-XGBoost_remove SES and Gender

[ ]:
```python
# Try remove SES feature
X = df.drop(['Group', 'CDR', 'MMSE', 'ASF', 'M/F', 'SES'], axis = 1)
y = df['Group']

# Split dataset into training and test sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[ ]:
```python
X_test.columns
```

[ ]:
```
Index(['Age', 'EDUC', 'eTIV', 'nWBV'], dtype='object')
```

[ ]:
```python
xgb = XGBClassifier(random_state=42,n_jobs=-1)

param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.001, 0.01, 0.1],
    'subsample': [0.7, 0.8, 0.9],
    'colsample_bytree': [0.7, 0.8, 1.0],
    'min_child_weight': [1, 3, 5],
}

inner_cv = RepeatedStratifiedKFold(
    n_splits=5, n_repeats=2, random_state=42
)

grid = GridSearchCV(
    estimator=xgb,
    param_grid=param_grid,
    scoring='f1',
    cv=inner_cv,
    n_jobs=-1,
    verbose=0
)

grid.fit(X_train, y_train)
best_model = grid.best_estimator_
print("XGBoost classifier:")
```

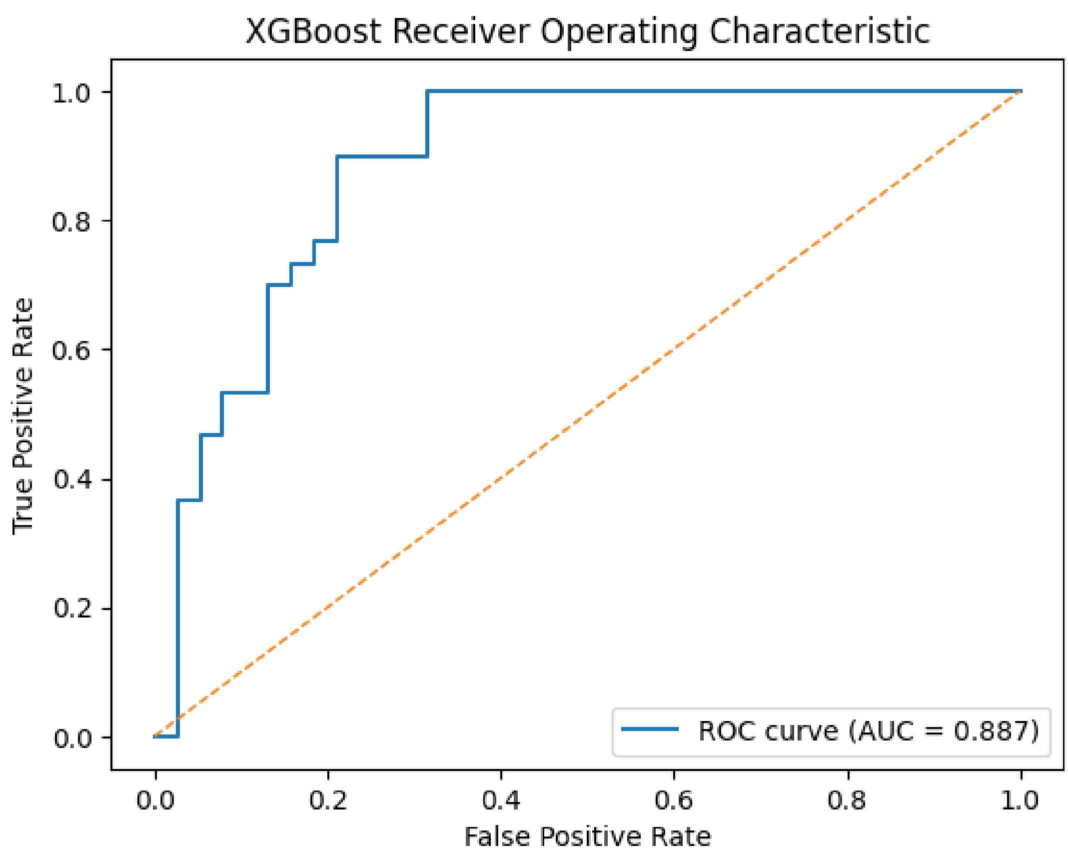
```python
print("Best params:", grid.best_params_)

scoring = {
    'accuracy': 'accuracy',
    'precision': 'precision',
    'recall': 'recall',
    'f1': 'f1',
    'roc_auc': 'roc_auc'
}
cv_res = cross_validate(
    best_model, X_train, y_train,
    cv=inner_cv, scoring=scoring, n_jobs=-1, return_train_score=False
)

def show(name):
    vals = cv_res[f'test_{name}']
    print(f"CV {name:<9}: {vals.mean():.4f} ± {vals.std():.4f}")

print("\n=== Repeated Stratified 5×2 CV on TRAIN ===")
for m in ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']:
    show(m)

y_pred = best_model.predict(X_test)
y_prob = best_model.predict_proba(X_test)[:, 1]

acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_prob)

print("\n=== Final Performance on HOLD-OUT TEST ===")
print(f"Accuracy : {acc:.4f}")
print(f"Precision: {prec:.4f}")
print(f"Recall   : {rec:.4f}")
print(f"F1 Score : {f1:.4f}")
print(f"ROC_AUC  : {auc:.4f}")

print("\nClassification Report on Test Set:\n", classification_report(y_test,␣
 ↪y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

feat_imp = pd.DataFrame({
    "Feature": X_train.columns,
    "Importance": best_model.feature_importances_
}).sort_values(by="Importance", ascending=False)
print("\nTop feature importances:\n", feat_imp)
```

```
XGBoost_mdl = best_model
```

```
XGBoost classifier:
Best params: {'colsample_bytree': 1.0, 'learning_rate': 0.1, 'max_depth': 7,
'min_child_weight': 1, 'n_estimators': 200, 'subsample': 0.9}

=== Repeated Stratified 5×2 CV on TRAIN ===
CV accuracy : 0.7781 ± 0.0621
CV precision: 0.7500 ± 0.0845
CV recall   : 0.7424 ± 0.0859
CV f1       : 0.7433 ± 0.0713
CV roc_auc  : 0.8492 ± 0.0526

=== Final Performance on HOLD-OUT TEST ===
Accuracy : 0.8235
Precision: 0.7647
Recall   : 0.8667
F1 Score : 0.8125
ROC_AUC  : 0.8868

Classification Report on Test Set:
              precision    recall  f1-score   support

           0       0.88      0.79      0.83        38
           1       0.76      0.87      0.81        30

    accuracy                           0.82        68
   macro avg       0.82      0.83      0.82        68
weighted avg       0.83      0.82      0.82        68

Confusion Matrix:
 [[30  8]
 [ 4 26]]

Top feature importances:
   Feature  Importance
1     EDUC    0.382794
3     nWBV    0.239561
2     eTIV    0.204472
0      Age    0.173173
```

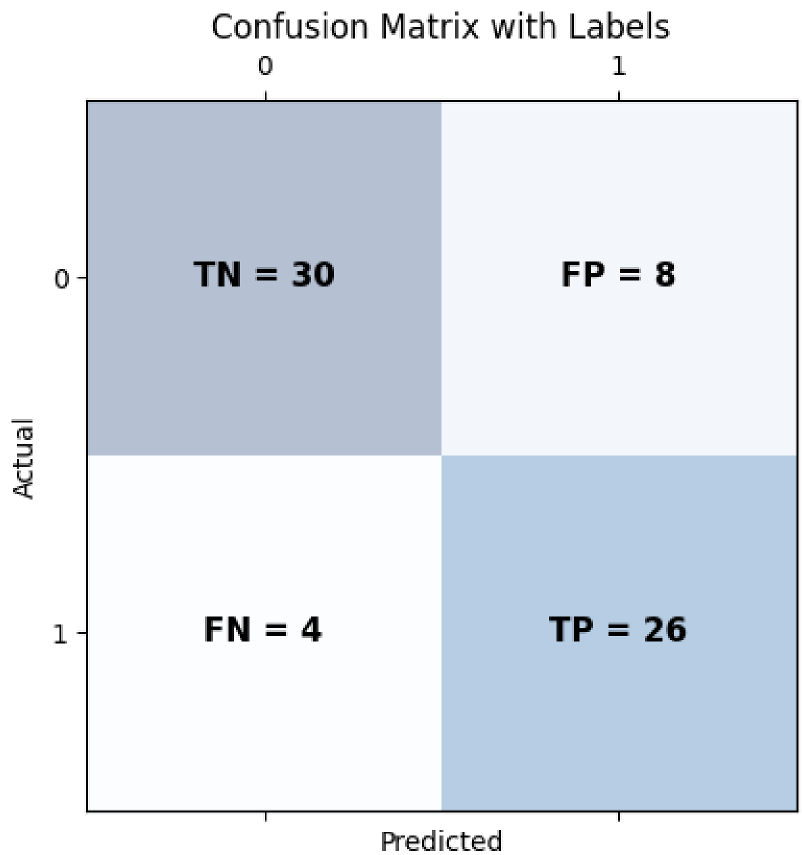
[ ]:
```
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = roc_auc_score(y_test, y_prob)

plt.figure()
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc_auc:.3f})')
```

```python
plt.plot([0, 1], [0, 1], linestyle='--', linewidth=1)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('XGBoost Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()
```



[ ]:
```python
cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()

labels = np.array([
    [f"TN = {tn}", f"FP = {fp}"],
    [f"FN = {fn}", f"TP = {tp}"]
])

fig, ax = plt.subplots()
ax.matshow(cm, cmap=plt.cm.Blues, alpha=0.3)
for i in range(2):
    for j in range(2):
        ax.text(j, i, labels[i, j],
```

```
                va='center', ha='center', fontsize=12, fontweight='bold')

plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix with Labels")
plt.show()
```



[ ]:
```
# Find all FN indices in the full test set
FN_all = (~y_pred) & (y_test == 1)
FN_indices = y_test[FN_all].index
print("False Negative indices:", FN_indices)

# Find all FP indices in the full test set
FP_all = (y_pred) & (y_test == 0)
FP_indices = y_test[FP_all].index
print("False Positive indices:", FP_indices)

# Find all TP indices in the full test set
TP_all = (y_pred) & (y_test == 1)
```

```
TP_indices = y_test[TP_all].index
print("True Positive indices:", TP_indices)

# Find all FN indices in the full test set
TN_all = (~y_pred) & (y_test == 0)
TN_indices = y_test[TN_all].index
print("True Negative indices:", TN_indices)
```

```
False Negative indices: Index([52, 300, 51, 94], dtype='int64')
False Positive indices: Index([167, 85, 146, 198, 130, 199, 7, 64],
dtype='int64')
True Positive indices: Index([124, 332, 250, 317, 154,  25,  90, 106, 172, 285,
87, 215, 127,   3,
       239, 162, 299, 345,  72,  39,  89,  88,  16, 329, 365, 275],
      dtype='int64')
True Negative indices: Index([ 84, 122, 311,  48, 336, 213,   9, 210, 113, 363,
66,   5, 153, 291,
       370, 158, 102, 243, 132, 306, 337, 196, 351, 179, 309, 197, 362, 209,
       333,  96],
      dtype='int64')
```

[ ]:
```
FN_sample_test_idx = X_test.index.get_indexer_for([300, 299, 51, 94])
FP_sample_test_idx = X_test.index.get_indexer_for([167, 146, 198, 130, 199, 7,␣
↪64])
TP_sample_test_idx = X_test.index.get_indexer_for([124, 332, 250, 317, 154, ␣
↪25,  90, 106, 172, 285,  87, 215, 127,  52,
         3, 239, 162, 345,  72,  39,  89,  88,  16, 329, 365, 275])
TN_sample_test_idx = X_test.index.get_indexer_for([84, 122, 311,  48, 336, 213,␣
↪   9, 210, 113,  85, 363,  66,   5, 153,
       291, 370, 158, 102, 243, 132, 306, 337, 196, 351, 179, 309, 197, 362,
       209, 333,  96])
```

[ ]:
```
print("FN_sample_test_idx: ", FN_sample_test_idx)
print("FP_sample_test_idx: ", FP_sample_test_idx)
print("TP_sample_test_idx: ", TP_sample_test_idx)
print("TN_sample_test_idx: ", TN_sample_test_idx)
```

```
FN_sample_test_idx:  [31 37 49 61]
FP_sample_test_idx:  [17 21 42 50 53 59 60]
TP_sample_test_idx:  [ 1  2  6  7  9 11 13 14 16 20 22 28 29 30 33 35 36 39 40
41 47 55 56 64
 65 67]
TN_sample_test_idx:  [ 0  3  4  5  8 10 12 15 18 19 23 24 25 26 27 32 34 38 43
44 45 46 48 51
 52 54 57 58 62 63 66]
```

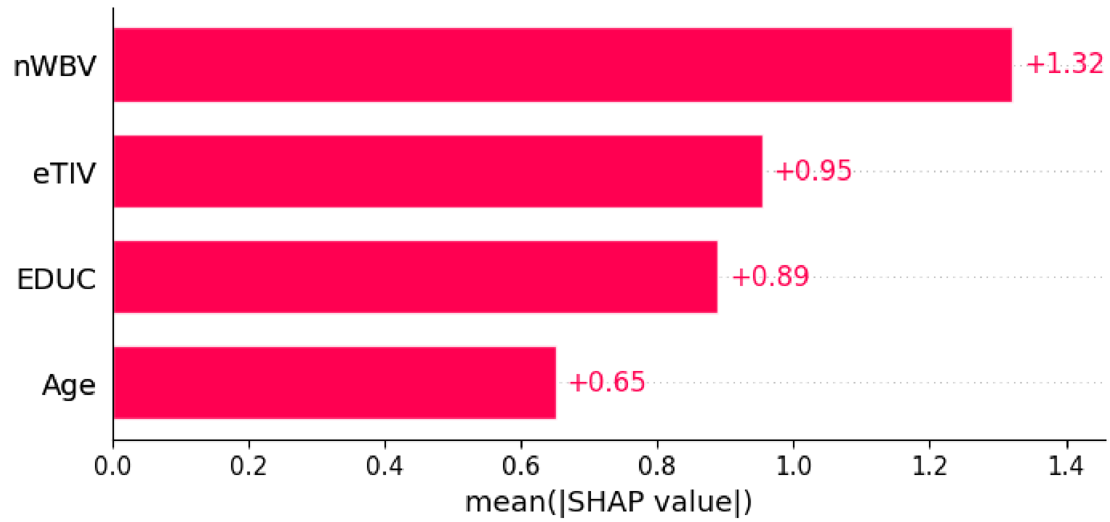
[ ]:
```python
XGB_explainer = shap.Explainer(XGBoost_mdl)
XGB_shap = XGB_explainer(X_test)
print(type(XGB_explainer))
```

```
<class 'shap.explainers._tree.TreeExplainer'>
```

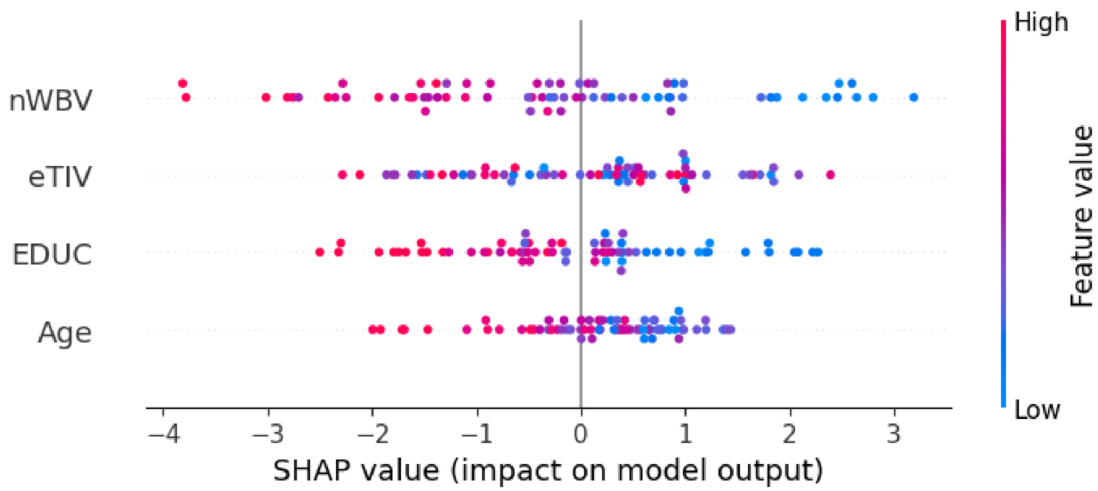
[ ]:
```python
print("Values dimensions: %s" % (XGB_shap.values.shape,))
print("Data dimensions:   %s" % (XGB_shap.data.shape,))
```

```
Values dimensions: (68, 4)
Data dimensions:   (68, 4)
```

[ ]:
```python
sb.reset_orig()
shap.plots.bar(XGB_shap)
```



[ ]:
```python
shap.plots.beeswarm(XGB_shap)
```



From the shap plot above, we can see that M/F(M=0, F=1) female and older age tend to push class to nondemented side.

7.3.6 Model training-LightGBM_remove SES and Gender

[ ]:
```
LightGBM = LGBMClassifier(random_state=42, verbosity=-1)

param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.001, 0.01, 0.1],
    'num_leaves': [31, 50, 70],
}

inner_cv = RepeatedStratifiedKFold(
    n_splits=5, n_repeats=2, random_state=42
)

grid = GridSearchCV(
    estimator=LightGBM,
    param_grid=param_grid,
    scoring='f1',
    cv=inner_cv,
    n_jobs=-1,
    verbose=0
)

grid.fit(X_train, y_train)
best_model = grid.best_estimator_
print("LightGBM classifier:")
```

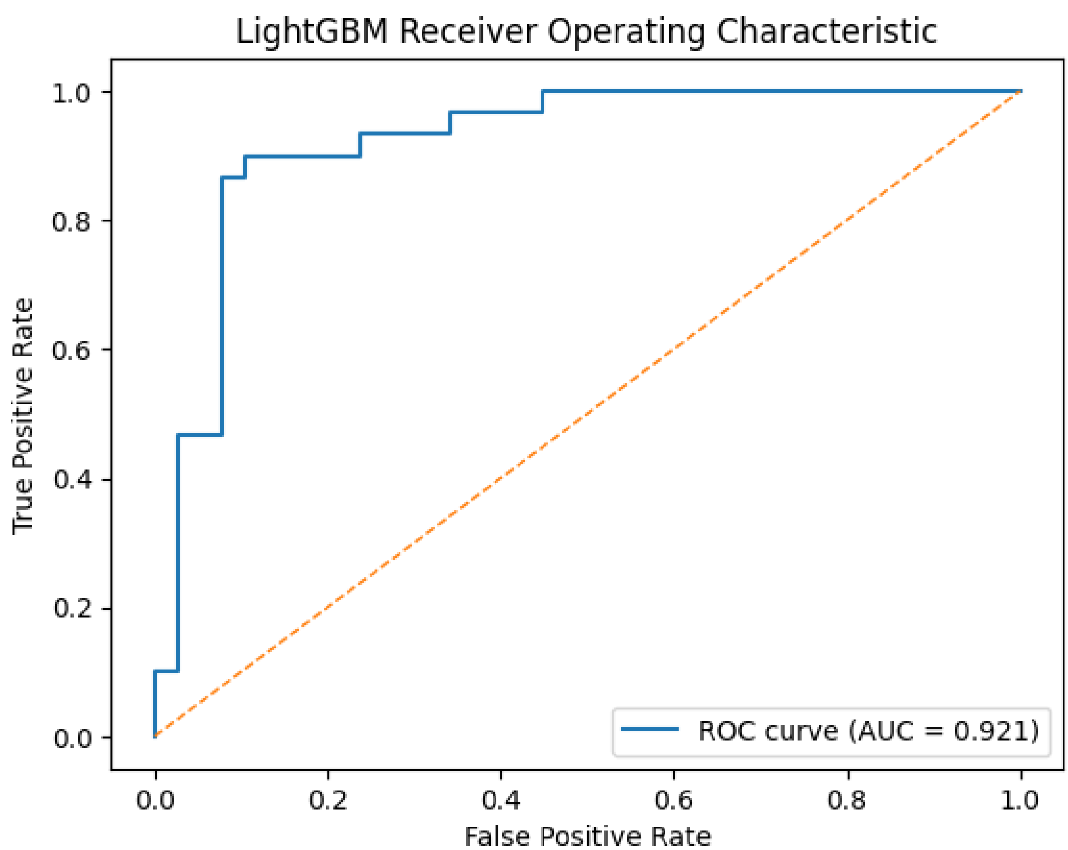
```python
print("Best params:", grid.best_params_)

scoring = {
    'accuracy': 'accuracy',
    'precision': 'precision',
    'recall': 'recall',
    'f1': 'f1',
    'roc_auc': 'roc_auc'
}
cv_res = cross_validate(
    best_model, X_train, y_train,
    cv=inner_cv, scoring=scoring, n_jobs=-1, return_train_score=False
)

def show(name):
    vals = cv_res[f'test_{name}']
    print(f"CV {name:<9}: {vals.mean():.4f} ± {vals.std():.4f}")

print("\n=== Repeated Stratified 5×2 CV on TRAIN ===")
for m in ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']:
    show(m)

y_pred = best_model.predict(X_test)
y_prob = best_model.predict_proba(X_test)[:, 1]

acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_prob)

print("\n=== Final Performance on HOLD-OUT TEST ===")
print(f"Accuracy : {acc:.4f}")
print(f"Precision: {prec:.4f}")
print(f"Recall   : {rec:.4f}")
print(f"F1 Score : {f1:.4f}")
print(f"ROC_AUC  : {auc:.4f}")

print("\nClassification Report on Test Set:\n", classification_report(y_test,␣
↪y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

feat_imp = pd.DataFrame({
    "Feature": X_train.columns,
    "Importance": best_model.feature_importances_
}).sort_values(by="Importance", ascending=False)
print("\nTop feature importances:\n", feat_imp)
```

```python
LightGBM_mdl = best_model
```

```
LightGBM classifier:
Best params: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 200,
'num_leaves': 31}

=== Repeated Stratified 5×2 CV on TRAIN ===
CV accuracy : 0.7632 ± 0.0448
CV precision: 0.7329 ± 0.0758
CV recall   : 0.7333 ± 0.0536
CV f1       : 0.7294 ± 0.0406
CV roc_auc  : 0.8323 ± 0.0565

=== Final Performance on HOLD-OUT TEST ===
Accuracy : 0.7941
Precision: 0.7500
Recall   : 0.8000
F1 Score : 0.7742
ROC_AUC  : 0.8860

Classification Report on Test Set:
              precision    recall  f1-score   support

           0       0.83      0.79      0.81        38
           1       0.75      0.80      0.77        30

    accuracy                           0.79        68
   macro avg       0.79      0.79      0.79        68
weighted avg       0.80      0.79      0.79        68

Confusion Matrix:
 [[30  8]
 [ 6 24]]

Top feature importances:
   Feature  Importance
2    eTIV         674
3    nWBV         423
0     Age         340
1    EDUC         200
```

[ ]:
```python
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = roc_auc_score(y_test, y_prob)

plt.figure()
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc_auc:.3f})')
```

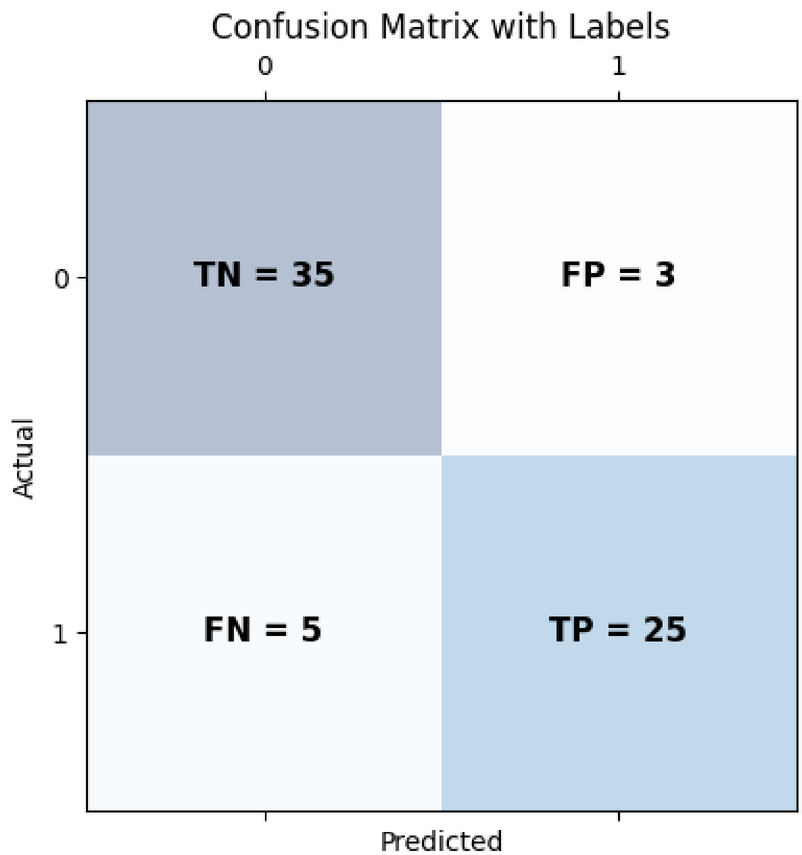
```
plt.plot([0, 1], [0, 1], linestyle='--', linewidth=1)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('LightGBM Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()
```



[ ]:

```
cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()

labels = np.array([
    [f"TN = {tn}", f"FP = {fp}"],
    [f"FN = {fn}", f"TP = {tp}"]
])

fig, ax = plt.subplots()
ax.matshow(cm, cmap=plt.cm.Blues, alpha=0.3)
for i in range(2):
    for j in range(2):
        ax.text(j, i, labels[i, j],
```

```
                va='center', ha='center', fontsize=12, fontweight='bold')

plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix with Labels")
plt.show()
```



[ ]:
```
# Find all FN indices in the full test set
FN_all = (~y_pred) & (y_test == 1)
FN_indices = y_test[FN_all].index
print("False Negative indices:", FN_indices)
```

```
False Negative indices: Index([52, 300, 345, 51, 94], dtype='int64')
```

[ ]:
```
FN_sample_test_idx = X_test.index.get_indexer_for([154, 300, 299, 39, 51, 94,␣
 ↪329])
FN_sample_test_idx
```

[ ]:
```
array([ 9, 31, 37, 41, 49, 61, 64])
```

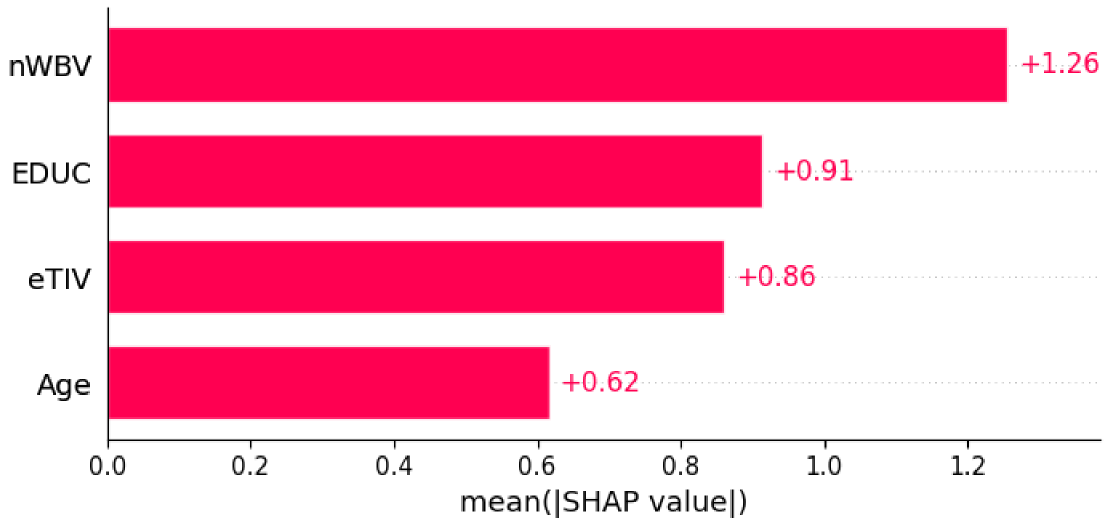
[ ]:
```python
LGB_explainer = shap.Explainer(LightGBM_mdl)
LGB_shap = LGB_explainer(X_test)
print(type(LGB_explainer))
```

```
<class 'shap.explainers._tree.TreeExplainer'>
```

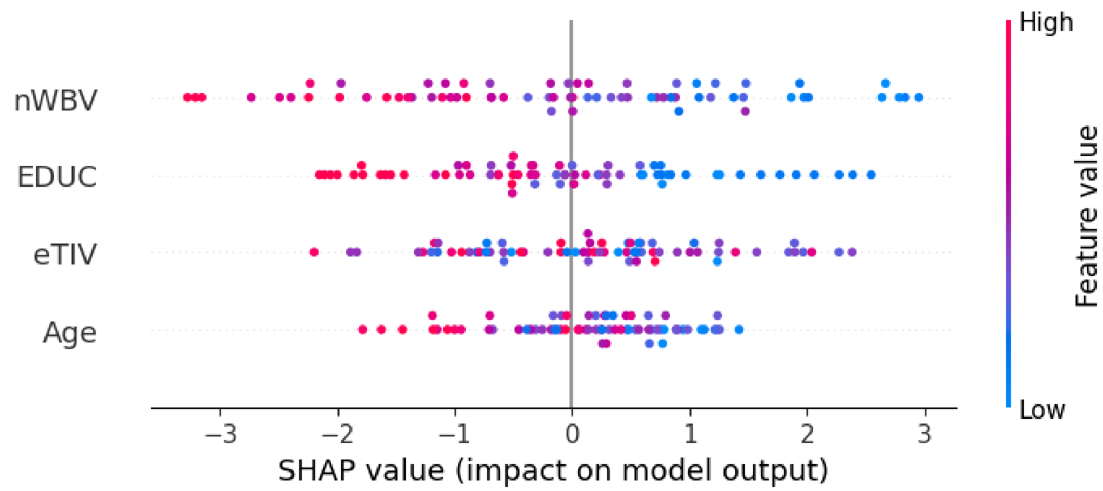
[ ]:
```python
print("Values dimensions: %s" % (LGB_shap.values.shape,))
print("Data dimensions:   %s" % (LGB_shap.data.shape,))
```

```
Values dimensions: (68, 4)
Data dimensions:   (68, 4)
```

[ ]:
```python
sb.reset_orig()
shap.plots.bar(LGB_shap)
```



[ ]:
```python
shap.plots.beeswarm(LGB_shap)
```



7.3.7 Model training-CatBoost_remove SES and Gender

[ ]:
```
CatBoost = CatBoostClassifier(random_state=42)

param_grid = {
    'min_data_in_leaf': [20, 40, 60],
    'rsm': [0.7, 0.8, 1.0],
    'iterations': [100, 200],
    'depth': [3, 5, 7],
    'learning_rate': [0.001, 0.01, 0.1],
    'l2_leaf_reg': [1, 3, 5],
    'bagging_temperature': [0, 0.5, 1.0],
}

inner_cv = RepeatedStratifiedKFold(
    n_splits=5, n_repeats=2, random_state=42
)

grid = GridSearchCV(
    estimator=CatBoost,
    param_grid=param_grid,
    scoring='f1',
    cv=inner_cv,
    n_jobs=-1,
    verbose=0
)

grid.fit(X_train, y_train)
best_model = grid.best_estimator_
print("CatBoostClassifier:")
```

```python
print("Best params:", grid.best_params_)

scoring = {
    'accuracy': 'accuracy',
    'precision': 'precision',
    'recall': 'recall',
    'f1': 'f1',
    'roc_auc': 'roc_auc'
}
cv_res = cross_validate(
    best_model, X_train, y_train,
    cv=inner_cv, scoring=scoring, n_jobs=-1, return_train_score=False
)

def show(name):
    vals = cv_res[f'test_{name}']
    print(f"CV {name:<9}: {vals.mean():.4f} ± {vals.std():.4f}")

print("\n=== Repeated Stratified 5×2 CV on TRAIN ===")
for m in ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']:
    show(m)

y_pred = best_model.predict(X_test)
y_prob = best_model.predict_proba(X_test)[:, 1]

acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_prob)

print("\n=== Final Performance on HOLD-OUT TEST ===")
print(f"Accuracy : {acc:.4f}")
print(f"Precision: {prec:.4f}")
print(f"Recall   : {rec:.4f}")
print(f"F1 Score : {f1:.4f}")
print(f"ROC_AUC  : {auc:.4f}")

print("\nClassification Report on Test Set:\n", classification_report(y_test,␣
 ↪y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

feat_imp = pd.DataFrame({
    "Feature": X_train.columns,
    "Importance": best_model.feature_importances_
}).sort_values(by="Importance", ascending=False)
print("\nTop feature importances:\n", feat_imp)
```

```
CatBoost_mdl = best_model
```

```
0:	learn: 0.6433030	total: 48.1ms	remaining: 9.57s
1:	learn: 0.6072231	total: 49.6ms	remaining: 4.91s
2:	learn: 0.5877394	total: 50.6ms	remaining: 3.33s
3:	learn: 0.5621783	total: 51.9ms	remaining: 2.54s
4:	learn: 0.5421796	total: 53ms	remaining: 2.07s
5:	learn: 0.5222894	total: 53.9ms	remaining: 1.74s
6:	learn: 0.5080146	total: 55ms	remaining: 1.52s
7:	learn: 0.4919294	total: 56.1ms	remaining: 1.35s
8:	learn: 0.4779497	total: 57ms	remaining: 1.21s
9:	learn: 0.4587318	total: 58.1ms	remaining: 1.1s
10:	learn: 0.4423055	total: 59.1ms	remaining: 1.02s
11:	learn: 0.4201557	total: 60.9ms	remaining: 954ms
12:	learn: 0.4078103	total: 62.5ms	remaining: 899ms
13:	learn: 0.4023800	total: 63.8ms	remaining: 847ms
14:	learn: 0.3942624	total: 64.9ms	remaining: 800ms
15:	learn: 0.3764204	total: 66ms	remaining: 759ms
16:	learn: 0.3745783	total: 66.5ms	remaining: 716ms
17:	learn: 0.3642996	total: 67.7ms	remaining: 685ms
18:	learn: 0.3580881	total: 68.8ms	remaining: 656ms
19:	learn: 0.3519850	total: 70.1ms	remaining: 631ms
20:	learn: 0.3396768	total: 71.2ms	remaining: 607ms
21:	learn: 0.3359188	total: 72.3ms	remaining: 585ms
22:	learn: 0.3311288	total: 73.5ms	remaining: 565ms
23:	learn: 0.3220628	total: 74.5ms	remaining: 546ms
24:	learn: 0.3180902	total: 75.6ms	remaining: 529ms
25:	learn: 0.3087211	total: 76.7ms	remaining: 513ms
26:	learn: 0.3037849	total: 77.8ms	remaining: 498ms
27:	learn: 0.2990716	total: 78.9ms	remaining: 485ms
28:	learn: 0.2913764	total: 80.2ms	remaining: 473ms
29:	learn: 0.2869979	total: 81.9ms	remaining: 464ms
30:	learn: 0.2856476	total: 83.2ms	remaining: 454ms
31:	learn: 0.2817315	total: 84.3ms	remaining: 442ms
32:	learn: 0.2797068	total: 85.5ms	remaining: 433ms
33:	learn: 0.2781957	total: 86.6ms	remaining: 423ms
34:	learn: 0.2757684	total: 87.6ms	remaining: 413ms
35:	learn: 0.2724397	total: 88.8ms	remaining: 405ms
36:	learn: 0.2662034	total: 89.9ms	remaining: 396ms
37:	learn: 0.2573439	total: 91ms	remaining: 388ms
38:	learn: 0.2537094	total: 92.1ms	remaining: 380ms
39:	learn: 0.2480944	total: 93.3ms	remaining: 373ms
40:	learn: 0.2405935	total: 94.5ms	remaining: 366ms
41:	learn: 0.2383741	total: 95.6ms	remaining: 360ms
42:	learn: 0.2321652	total: 96.7ms	remaining: 353ms
43:	learn: 0.2302174	total: 97.8ms	remaining: 347ms
```

```
44:	learn: 0.2276557	total: 98.9ms	remaining: 341ms
45:	learn: 0.2245745	total: 100ms	remaining: 336ms
46:	learn: 0.2199735	total: 102ms	remaining: 331ms
47:	learn: 0.2170957	total: 103ms	remaining: 325ms
48:	learn: 0.2124202	total: 104ms	remaining: 320ms
49:	learn: 0.2119652	total: 105ms	remaining: 315ms
50:	learn: 0.2090747	total: 106ms	remaining: 310ms
51:	learn: 0.2059814	total: 107ms	remaining: 305ms
52:	learn: 0.2032197	total: 108ms	remaining: 300ms
53:	learn: 0.1995946	total: 109ms	remaining: 296ms
54:	learn: 0.1973423	total: 111ms	remaining: 291ms
55:	learn: 0.1948246	total: 112ms	remaining: 287ms
56:	learn: 0.1907309	total: 113ms	remaining: 283ms
57:	learn: 0.1895778	total: 114ms	remaining: 278ms
58:	learn: 0.1884515	total: 115ms	remaining: 274ms
59:	learn: 0.1854442	total: 116ms	remaining: 271ms
60:	learn: 0.1809831	total: 117ms	remaining: 268ms
61:	learn: 0.1782131	total: 119ms	remaining: 264ms
62:	learn: 0.1737153	total: 120ms	remaining: 261ms
63:	learn: 0.1716256	total: 121ms	remaining: 257ms
64:	learn: 0.1670977	total: 122ms	remaining: 254ms
65:	learn: 0.1629564	total: 123ms	remaining: 251ms
66:	learn: 0.1605172	total: 124ms	remaining: 247ms
67:	learn: 0.1575741	total: 125ms	remaining: 244ms
68:	learn: 0.1557338	total: 127ms	remaining: 240ms
69:	learn: 0.1525787	total: 128ms	remaining: 237ms
70:	learn: 0.1496277	total: 129ms	remaining: 234ms
71:	learn: 0.1469448	total: 130ms	remaining: 231ms
72:	learn: 0.1454756	total: 131ms	remaining: 229ms
73:	learn: 0.1438074	total: 133ms	remaining: 226ms
74:	learn: 0.1412499	total: 134ms	remaining: 224ms
75:	learn: 0.1394253	total: 136ms	remaining: 221ms
76:	learn: 0.1378607	total: 137ms	remaining: 218ms
77:	learn: 0.1371424	total: 138ms	remaining: 216ms
78:	learn: 0.1351813	total: 139ms	remaining: 213ms
79:	learn: 0.1327478	total: 141ms	remaining: 212ms
80:	learn: 0.1280091	total: 143ms	remaining: 210ms
81:	learn: 0.1265301	total: 144ms	remaining: 207ms
82:	learn: 0.1230572	total: 145ms	remaining: 204ms
83:	learn: 0.1224970	total: 146ms	remaining: 202ms
84:	learn: 0.1202086	total: 148ms	remaining: 200ms
85:	learn: 0.1184823	total: 149ms	remaining: 197ms
86:	learn: 0.1173750	total: 150ms	remaining: 194ms
87:	learn: 0.1166478	total: 151ms	remaining: 192ms
88:	learn: 0.1140979	total: 152ms	remaining: 190ms
89:	learn: 0.1128446	total: 154ms	remaining: 188ms
90:	learn: 0.1101893	total: 156ms	remaining: 187ms
91:	learn: 0.1088455	total: 158ms	remaining: 185ms
```

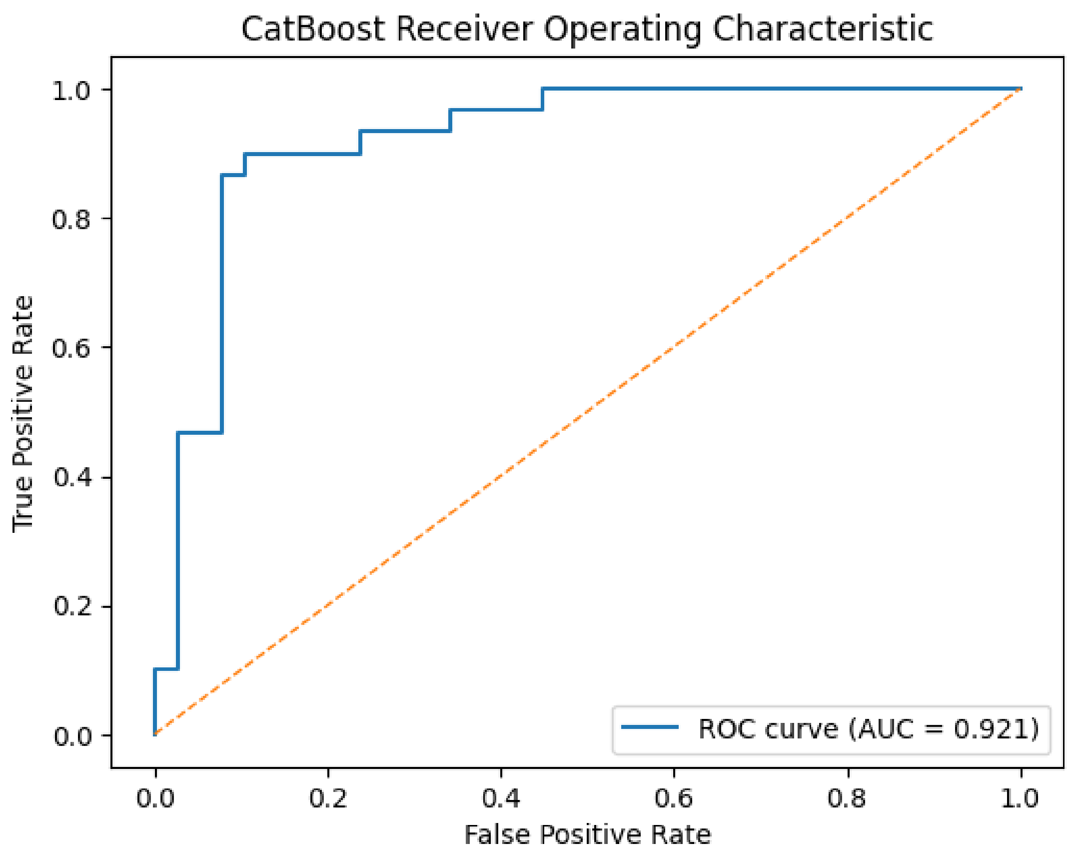
```
92:	learn: 0.1067866	total: 159ms	remaining: 183ms
93:	learn: 0.1045246	total: 160ms	remaining: 181ms
94:	learn: 0.1039400	total: 161ms	remaining: 178ms
95:	learn: 0.1017326	total: 162ms	remaining: 176ms
96:	learn: 0.0982394	total: 163ms	remaining: 173ms
97:	learn: 0.0977207	total: 164ms	remaining: 171ms
98:	learn: 0.0973069	total: 166ms	remaining: 169ms
99:	learn: 0.0967707	total: 167ms	remaining: 167ms
100:	learn: 0.0944164	total: 168ms	remaining: 165ms
101:	learn: 0.0938359	total: 169ms	remaining: 162ms
102:	learn: 0.0923871	total: 170ms	remaining: 160ms
103:	learn: 0.0916017	total: 171ms	remaining: 158ms
104:	learn: 0.0905539	total: 172ms	remaining: 155ms
105:	learn: 0.0887147	total: 173ms	remaining: 153ms
106:	learn: 0.0875213	total: 174ms	remaining: 151ms
107:	learn: 0.0866559	total: 175ms	remaining: 149ms
108:	learn: 0.0854698	total: 176ms	remaining: 147ms
109:	learn: 0.0842835	total: 177ms	remaining: 145ms
110:	learn: 0.0837254	total: 178ms	remaining: 143ms
111:	learn: 0.0808711	total: 179ms	remaining: 141ms
112:	learn: 0.0803991	total: 180ms	remaining: 139ms
113:	learn: 0.0793167	total: 182ms	remaining: 137ms
114:	learn: 0.0785392	total: 183ms	remaining: 135ms
115:	learn: 0.0766991	total: 184ms	remaining: 134ms
116:	learn: 0.0759175	total: 186ms	remaining: 132ms
117:	learn: 0.0754021	total: 187ms	remaining: 130ms
118:	learn: 0.0742354	total: 189ms	remaining: 128ms
119:	learn: 0.0739136	total: 190ms	remaining: 126ms
120:	learn: 0.0733437	total: 191ms	remaining: 125ms
121:	learn: 0.0729341	total: 191ms	remaining: 122ms
122:	learn: 0.0721195	total: 192ms	remaining: 120ms
123:	learn: 0.0718087	total: 193ms	remaining: 118ms
124:	learn: 0.0703325	total: 194ms	remaining: 117ms
125:	learn: 0.0700263	total: 195ms	remaining: 115ms
126:	learn: 0.0693124	total: 196ms	remaining: 113ms
127:	learn: 0.0687718	total: 198ms	remaining: 111ms
128:	learn: 0.0685893	total: 198ms	remaining: 109ms
129:	learn: 0.0675148	total: 199ms	remaining: 107ms
130:	learn: 0.0669490	total: 200ms	remaining: 106ms
131:	learn: 0.0658678	total: 202ms	remaining: 104ms
132:	learn: 0.0655645	total: 203ms	remaining: 102ms
133:	learn: 0.0646241	total: 204ms	remaining: 100ms
134:	learn: 0.0638795	total: 205ms	remaining: 98.5ms
135:	learn: 0.0631970	total: 206ms	remaining: 96.8ms
136:	learn: 0.0629538	total: 207ms	remaining: 95ms
137:	learn: 0.0626684	total: 208ms	remaining: 93.3ms
138:	learn: 0.0621399	total: 209ms	remaining: 91.5ms
139:	learn: 0.0616209	total: 209ms	remaining: 89.8ms
```

```
140:    learn: 0.0604167        total: 210ms    remaining: 88ms
141:    learn: 0.0593850        total: 211ms    remaining: 86.3ms
142:    learn: 0.0588888        total: 212ms    remaining: 84.6ms
143:    learn: 0.0584728        total: 213ms    remaining: 83ms
144:    learn: 0.0575776        total: 214ms    remaining: 81.3ms
145:    learn: 0.0573503        total: 215ms    remaining: 79.6ms
146:    learn: 0.0570246        total: 216ms    remaining: 78ms
147:    learn: 0.0564381        total: 217ms    remaining: 76.3ms
148:    learn: 0.0562654        total: 218ms    remaining: 74.7ms
149:    learn: 0.0556245        total: 219ms    remaining: 73.1ms
150:    learn: 0.0551304        total: 220ms    remaining: 71.5ms
151:    learn: 0.0549703        total: 221ms    remaining: 69.9ms
152:    learn: 0.0543406        total: 222ms    remaining: 68.3ms
153:    learn: 0.0537970        total: 224ms    remaining: 66.8ms
154:    learn: 0.0533642        total: 225ms    remaining: 65.4ms
155:    learn: 0.0528259        total: 227ms    remaining: 63.9ms
156:    learn: 0.0523555        total: 229ms    remaining: 62.7ms
157:    learn: 0.0515599        total: 230ms    remaining: 61.2ms
158:    learn: 0.0514442        total: 232ms    remaining: 59.7ms
159:    learn: 0.0512947        total: 233ms    remaining: 58.2ms
160:    learn: 0.0512666        total: 234ms    remaining: 56.6ms
161:    learn: 0.0510238        total: 235ms    remaining: 55ms
162:    learn: 0.0505527        total: 236ms    remaining: 53.5ms
163:    learn: 0.0498957        total: 237ms    remaining: 51.9ms
164:    learn: 0.0494456        total: 238ms    remaining: 50.5ms
165:    learn: 0.0492686        total: 240ms    remaining: 49.2ms
166:    learn: 0.0489407        total: 241ms    remaining: 47.6ms
167:    learn: 0.0485755        total: 242ms    remaining: 46.1ms
168:    learn: 0.0480240        total: 243ms    remaining: 44.6ms
169:    learn: 0.0471546        total: 245ms    remaining: 43.2ms
170:    learn: 0.0464787        total: 246ms    remaining: 41.7ms
171:    learn: 0.0459925        total: 247ms    remaining: 40.1ms
172:    learn: 0.0455186        total: 248ms    remaining: 38.7ms
173:    learn: 0.0453493        total: 249ms    remaining: 37.2ms
174:    learn: 0.0449091        total: 250ms    remaining: 35.7ms
175:    learn: 0.0443691        total: 251ms    remaining: 34.2ms
176:    learn: 0.0440176        total: 252ms    remaining: 32.7ms
177:    learn: 0.0435075        total: 253ms    remaining: 31.3ms
178:    learn: 0.0432383        total: 254ms    remaining: 29.8ms
179:    learn: 0.0429571        total: 255ms    remaining: 28.3ms
180:    learn: 0.0427847        total: 256ms    remaining: 26.9ms
181:    learn: 0.0422452        total: 257ms    remaining: 25.4ms
182:    learn: 0.0418478        total: 258ms    remaining: 24ms
183:    learn: 0.0414921        total: 259ms    remaining: 22.6ms
184:    learn: 0.0411721        total: 260ms    remaining: 21.1ms
185:    learn: 0.0409214        total: 261ms    remaining: 19.7ms
186:    learn: 0.0405452        total: 262ms    remaining: 18.2ms
187:    learn: 0.0402071        total: 263ms    remaining: 16.8ms
```

```
188:    learn: 0.0398397        total: 264ms    remaining: 15.4ms
189:    learn: 0.0393759        total: 265ms    remaining: 14ms
190:    learn: 0.0392496        total: 266ms    remaining: 12.5ms
191:    learn: 0.0388633        total: 267ms    remaining: 11.1ms
192:    learn: 0.0384548        total: 268ms    remaining: 9.73ms
193:    learn: 0.0382617        total: 269ms    remaining: 8.33ms
194:    learn: 0.0381814        total: 270ms    remaining: 6.93ms
195:    learn: 0.0379160        total: 271ms    remaining: 5.54ms
196:    learn: 0.0377706        total: 273ms    remaining: 4.15ms
197:    learn: 0.0374555        total: 274ms    remaining: 2.76ms
198:    learn: 0.0373077        total: 275ms    remaining: 1.38ms
199:    learn: 0.0370488        total: 276ms    remaining: 0us
CatBoostClassifier:
Best params: {'bagging_temperature': 0, 'depth': 7, 'iterations': 200,
'l2_leaf_reg': 3, 'learning_rate': 0.1, 'min_data_in_leaf': 20, 'rsm': 0.8}

=== Repeated Stratified 5×2 CV on TRAIN ===
CV accuracy : 0.8194 ± 0.0687
CV precision: 0.7995 ± 0.0876
CV recall   : 0.7855 ± 0.1006
CV f1       : 0.7893 ± 0.0823
CV roc_auc  : 0.8852 ± 0.0432

=== Final Performance on HOLD-OUT TEST ===
Accuracy : 0.8824
Precision: 0.8929
Recall   : 0.8333
F1 Score : 0.8621
ROC_AUC  : 0.9211

Classification Report on Test Set:
              precision    recall  f1-score   support

           0       0.88      0.92      0.90        38
           1       0.89      0.83      0.86        30

    accuracy                           0.88        68
   macro avg       0.88      0.88      0.88        68
weighted avg       0.88      0.88      0.88        68

Confusion Matrix:
 [[35  3]
 [ 5 25]]

Top feature importances:
   Feature  Importance
2    eTIV   32.923596
3    nWBV   25.126673
```

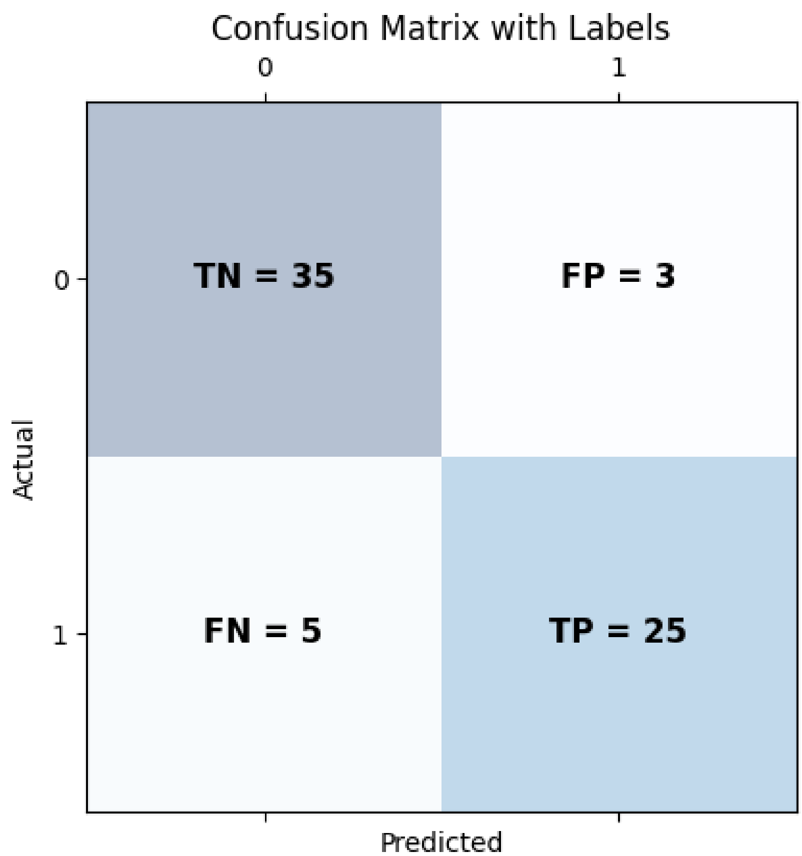
```
1     EDUC    24.059391
0      Age    17.890340
```

[ ]:
```python
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = roc_auc_score(y_test, y_prob)

plt.figure()
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc_auc:.3f})')
plt.plot([0, 1], [0, 1], linestyle='--', linewidth=1)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('CatBoost Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()
```



[ ]:
```python
cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()

labels = np.array([
    [f"TN = {tn}", f"FP = {fp}"],
```

```python
    [f"FN = {fn}", f"TP = {tp}"]
])

fig, ax = plt.subplots()
ax.matshow(cm, cmap=plt.cm.Blues, alpha=0.3)
for i in range(2):
    for j in range(2):
        ax.text(j, i, labels[i, j],
                va='center', ha='center', fontsize=12, fontweight='bold')

plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix with Labels")
plt.show()
```



[ ]:
```python
# Find all FN indices in the full test set
FN_all = (~y_pred) & (y_test == 1)
FN_indices = y_test[FN_all].index
print("False Negative indices:", FN_indices)
```

```
False Negative indices: Index([52, 300, 345, 51, 94], dtype='int64')
```

```
[ ]: FN_sample_test_idx = X_test.index.get_indexer_for([300, 299, 51, 94])
     FN_sample_test_idx
```

```
[ ]: array([31, 37, 49, 61])
```

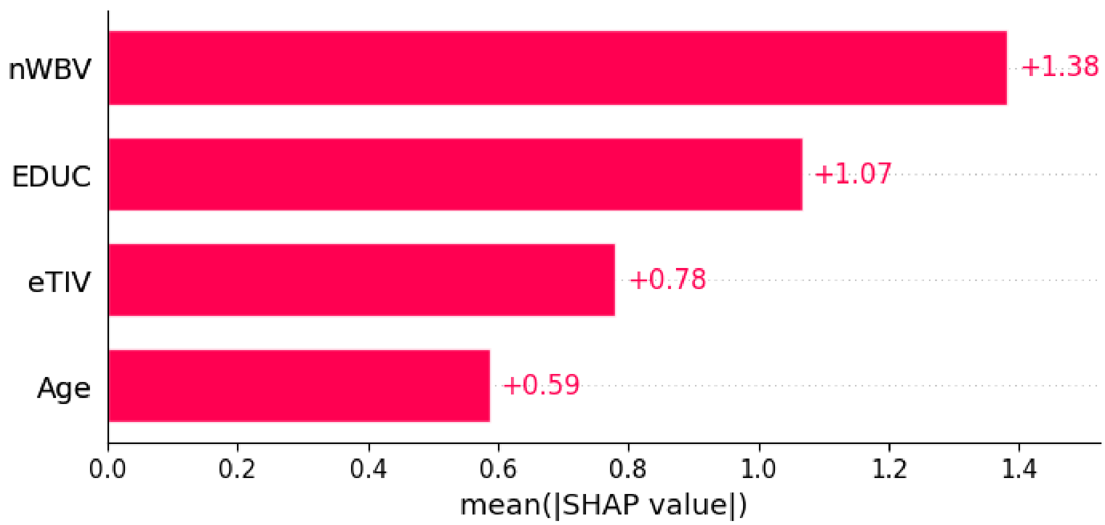
```
[ ]: CB_explainer = shap.Explainer(CatBoost_mdl)
     CB_shap = CB_explainer(X_test)
     print(type(CB_explainer))
```

```
<class 'shap.explainers._tree.TreeExplainer'>
```

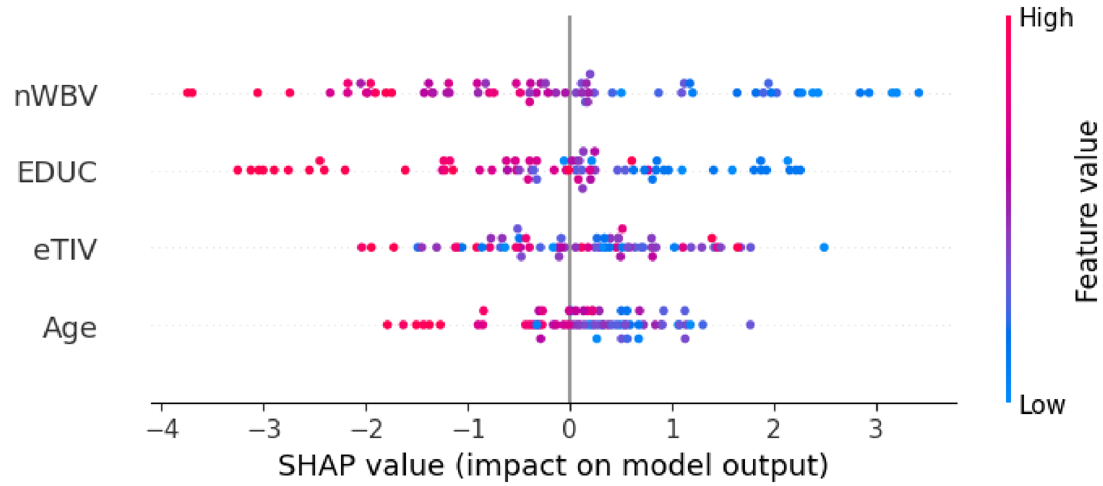
```
[ ]: print("Values dimensions: %s" % (CB_shap.values.shape,))
     print("Data dimensions:   %s" % (CB_shap.data.shape,))
```

```
Values dimensions: (68, 4)
Data dimensions:   (68, 4)
```

```
[ ]: sb.reset_orig()
     shap.plots.bar(CB_shap)
```




```
[ ]: shap.plots.beeswarm(CB_shap)
```



7.3.8 Model training-CatBoost_remove Age and Gender

[ ]:

```
# Initialize a CatBoost classifier
CatBoost = CatBoostClassifier(random_state=42, verbose = False)

# Define the parameter grid for Grid
CatBoost_param_dist = {
    'min_data_in_leaf': [20, 40, 60],
    'rsm': [0.7, 0.8, 1.0],
    'iterations': [100, 200],
    'depth': [3, 5, 7],
    'learning_rate': [0.001, 0.01, 0.1],
    'l2_leaf_reg': [1, 3, 5],
    'bagging_temperature': [0, 0.5, 1.0],
}

grid_search = GridSearchCV(CatBoost, CatBoost_param_dist, cv=5,␣
↪scoring='recall', n_jobs=-1)
grid_search.fit(X_train, y_train)
print("Best hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

CatBoost_mdl = grid_search.best_estimator_
y_pred = CatBoost_mdl.predict(X_test)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/tmp/ipython-input-3511066618.py in <cell line: 0>()
     14
```

```
     15 grid_search = GridSearchCV(CatBoost, CatBoost_param_dist, cv=5,␣
 ↪scoring='recall', n_jobs=-1)
---> 16 grid_search.fit(X_train, y_train)
     17 print("Best hyperparameters found by GridSearchCV:")
     18 print(grid_search.best_params_)

/usr/local/lib/python3.11/dist-packages/sklearn/base.py in wrapper(estimator,␣
 ↪*args, **kwargs)
   1387                 )
   1388             ):
-> 1389                 return fit_method(estimator, *args, **kwargs)
   1390
   1391         return wrapper

/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py in␣
 ↪fit(self, X, y, **params)
   1022                 return results
   1023
-> 1024             self._run_search(evaluate_candidates)
   1025
   1026             # multimetric is determined here because in the case of a␣
 ↪callable

/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py in␣
 ↪_run_search(self, evaluate_candidates)
   1569     def _run_search(self, evaluate_candidates):
   1570         """Search all candidates in param_grid"""
-> 1571         evaluate_candidates(ParameterGrid(self.param_grid))
   1572
   1573

/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py in␣
 ↪evaluate_candidates(candidate_params, cv, more_results)
    968                     )
    969
--> 970                 out = parallel(

    971                     delayed(_fit_and_score)(
    972                         clone(base_estimator),

/usr/local/lib/python3.11/dist-packages/sklearn/utils/parallel.py in␣
 ↪__call__(self, iterable)
     75             for delayed_func, args, kwargs in iterable
     76         )
---> 77         return super().__call__(iterable_with_config)
     78
     79
```

```
/usr/local/lib/python3.11/dist-packages/joblib/parallel.py in __call__(self,␣
↪iterable)
   2070         next(output)
   2071
-> 2072         return output if self.return_generator else list(output)
   2073
   2074     def __repr__(self):

/usr/local/lib/python3.11/dist-packages/joblib/parallel.py in _get_outputs(self ␣
↪iterator, pre_dispatch)
   1680
   1681             with self._backend.retrieval_context():
-> 1682                 yield from self._retrieve()
   1683
   1684         except GeneratorExit:

/usr/local/lib/python3.11/dist-packages/joblib/parallel.py in _retrieve(self)
   1798                     self._jobs[0].get_status(timeout=self.timeout) ==␣
↪TASK_PENDING
   1799                 ):
-> 1800                     time.sleep(0.01)
   1801                     continue
   1802

KeyboardInterrupt:
```

CatBoost Best hyperparameters found by GridSearchCV:

{'bagging_temperature': 0, 'depth': 5, 'iterations': 200, 'l2_leaf_reg': 1, 'learning_rate': 0.1, 'min_data_in_leaf': 20, 'rsm': 1.0}

[ ]:
```
print("CatBoost: ")
print("\nClassification Report on Test Set:")
print(classification_report(y_test, y_pred))
print("------------------------------------------------------")

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
print("ROC_AUC:", roc_auc_score(y_test, y_pred))
print("------------------------------------------------------")

# Get feature importances
feature_importances = pd.DataFrame({
    "Feature": X_test.columns,
    "Importance": CatBoost_mdl.feature_importances_
```

```
})

feature_importances = feature_importances.sort_values(by="Importance",␣
↪ascending=False)
feature_importances
```

```
CatBoost:

Classification Report on Test Set:
              precision    recall  f1-score   support

           0       0.85      0.92      0.89        38
           1       0.89      0.80      0.84        30

    accuracy                           0.87        68
   macro avg       0.87      0.86      0.86        68
weighted avg       0.87      0.87      0.87        68

------------------------------------------------------
Accuracy: 0.8676470588235294
Precision: 0.8888888888888888
Recall: 0.8
F1 Score: 0.8421052631578947
ROC_AUC: 0.8605263157894737
------------------------------------------------------
```

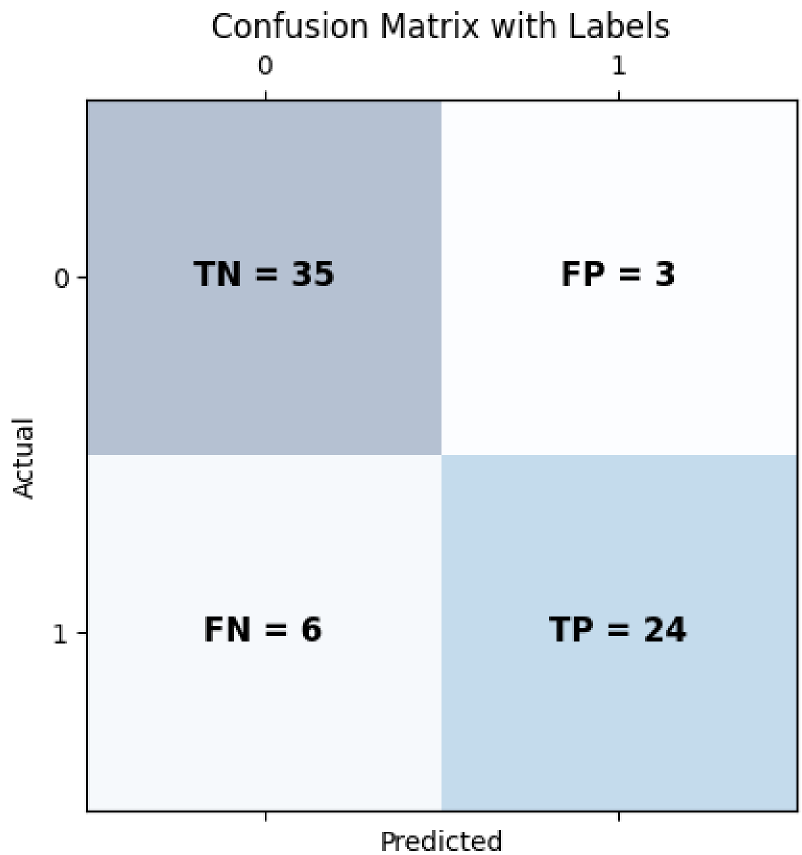
[ ]:

	Feature	Importance
2	eTIV	37.023811
3	nWBV	25.111532
1	SES	19.110291
0	EDUC	18.754366

[ ]:

```
cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()

labels = np.array([
    [f"TN = {tn}", f"FP = {fp}"],
    [f"FN = {fn}", f"TP = {tp}"]
])

fig, ax = plt.subplots()
ax.matshow(cm, cmap=plt.cm.Blues, alpha=0.3)
for i in range(2):
    for j in range(2):
        ax.text(j, i, labels[i, j],
                va='center', ha='center', fontsize=12, fontweight='bold')
```

```python
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix with Labels")
plt.show()
```



[ ]:
```python
# Find all FN indices in the full test set
FN_all = (~y_pred) & (y_test == 1)
FN_indices = y_test[FN_all].index
print("False Negative indices:", FN_indices)
```

```
False Negative indices: Index([332, 52, 300, 299, 51, 94], dtype='int64')
```

[ ]:
```python
FN_sample_test_idx = X_test.index.get_indexer_for([332, 52, 300, 299, 51, 94])
FN_sample_test_idx
```

[ ]:
```
array([ 2, 30, 31, 37, 49, 61])
```

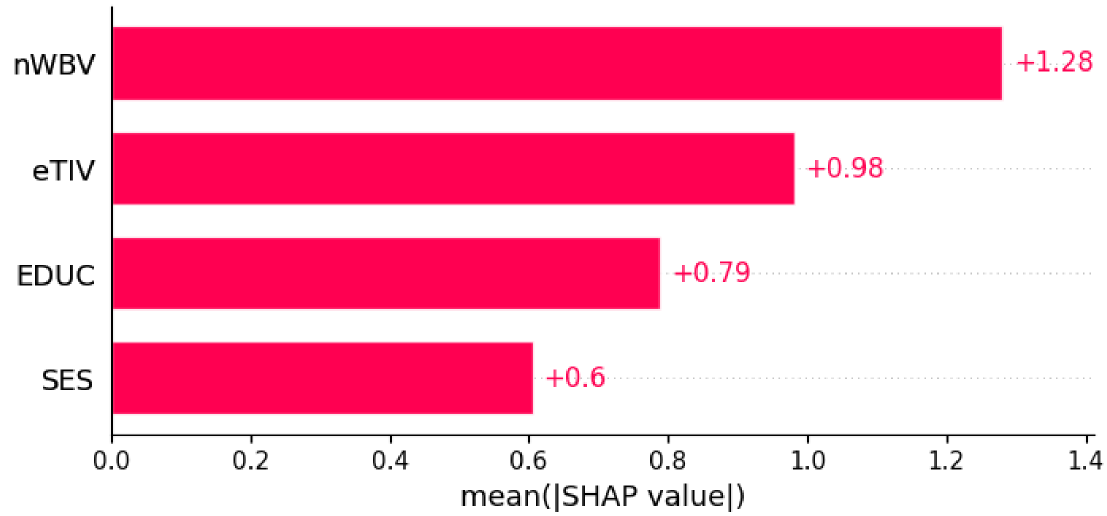
[ ]:
```python
CB_explainer = shap.Explainer(CatBoost_mdl)
CB_shap = CB_explainer(X_test)
print(type(CB_explainer))
```

```
<class 'shap.explainers._tree.TreeExplainer'>
```

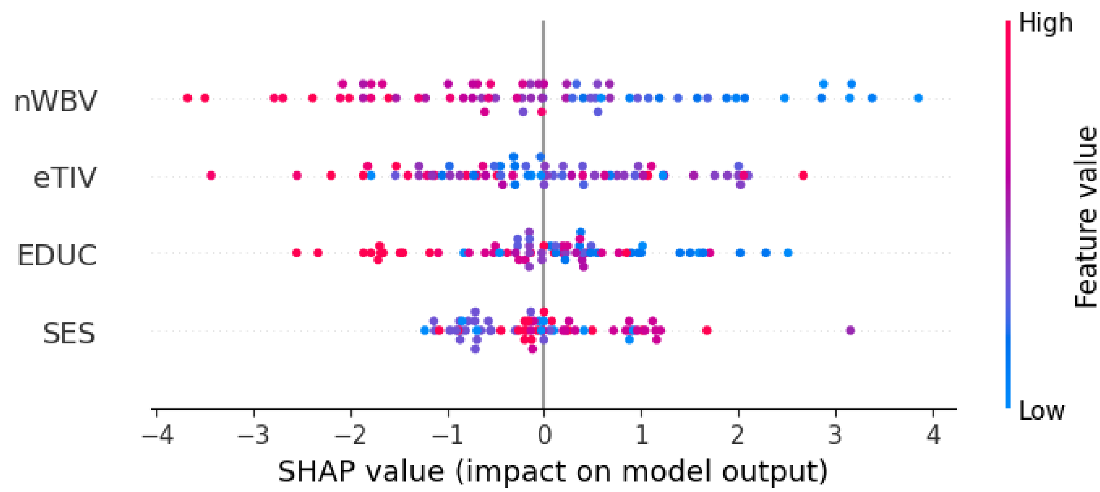
[ ]:
```
print("Values dimensions: %s" % (CB_shap.values.shape,))
print("Data dimensions:   %s" % (CB_shap.data.shape,))
```

```
Values dimensions: (68, 4)
Data dimensions:   (68, 4)
```

[ ]:
```
sb.reset_orig()
shap.plots.bar(CB_shap)
```



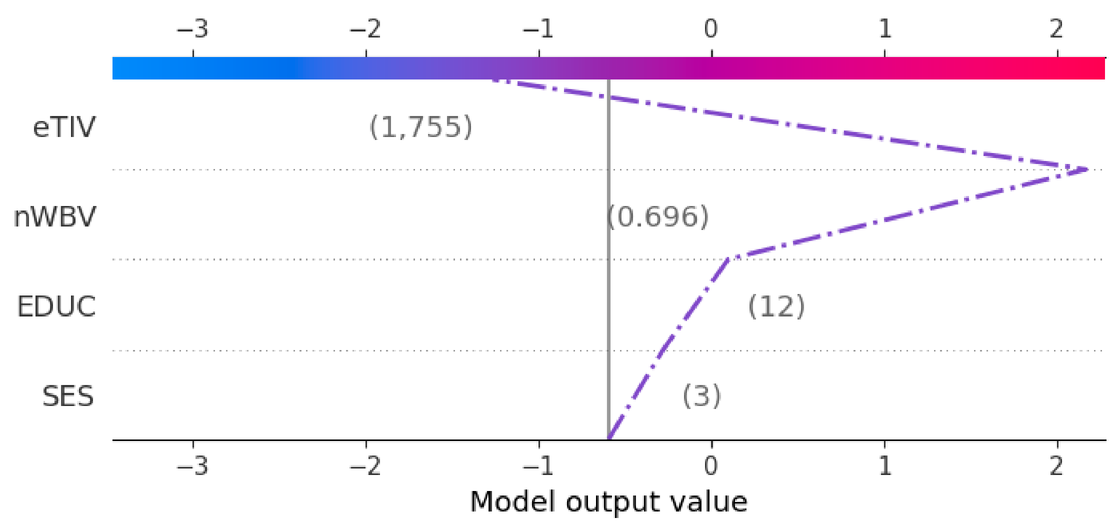
[ ]:
```
shap.plots.beeswarm(CB_shap)
```



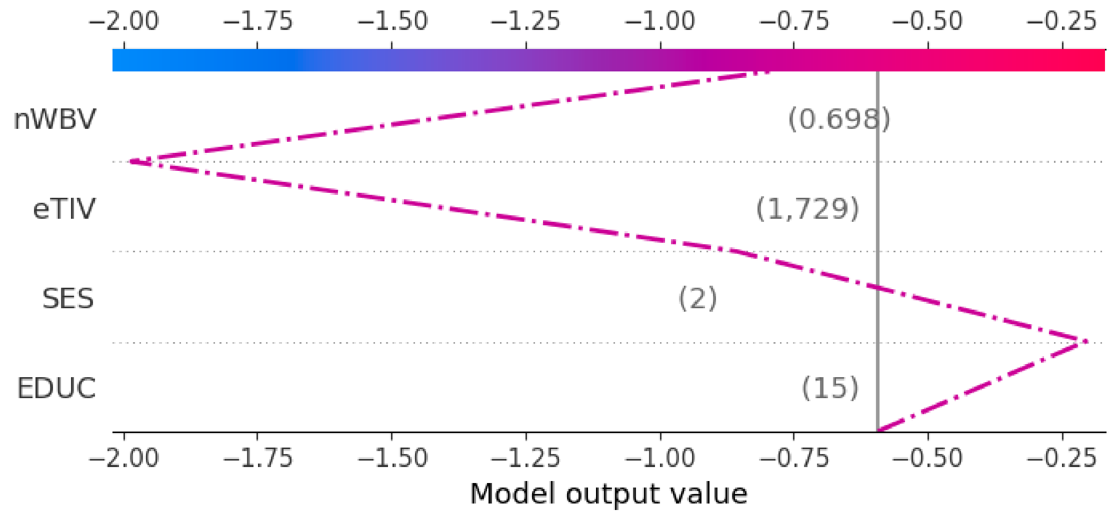
[ ]:
```python
print("X_test.iloc[2]: ")
print(X_test.iloc[2])
print(y_test.iloc[2], y_pred[2])
print("-----------------")
print("X_test.iloc[31]: ")
print(X_test.iloc[31])
print(y_test.iloc[31], y_pred[31])
```

```
X_test.iloc[2]:
EDUC       12.000
SES         3.000
eTIV     1755.000
nWBV        0.696
Name: 332, dtype: float64
1 0
-----------------
X_test.iloc[31]:
EDUC       15.000
SES         2.000
eTIV     1729.000
nWBV        0.698
Name: 300, dtype: float64
1 0
```

[ ]:
```python
# CB-FN-2
expected_value = CB_explainer.expected_value
shap.decision_plot(expected_value, CB_shap.values[2], X_test.iloc[2],␣
 ↪highlight=0)
```



[ ]:
```
# CB-FN-31
expected_value = CB_explainer.expected_value
shap.decision_plot(expected_value, CB_shap.values[31], X_test.iloc[31],
  ↪highlight=0)
```



[ ]:
```
# CB-FN-2
shap.initjs()
expected_value = CB_explainer.expected_value
shap.force_plot(expected_value, CB_shap.values[2], X_test.iloc[2])
```

```
<IPython.core.display.HTML object>
```

[ ]:
```
<shap.plots._force.AdditiveForceVisualizer at 0x7dda42593b50>
```

[ ]:
```
# CB-FN-31
shap.initjs()
expected_value = CB_explainer.expected_value
shap.force_plot(expected_value, CB_shap.values[31], X_test.iloc[31])
```

```
<IPython.core.display.HTML object>
```

[ ]:
```
<shap.plots._force.AdditiveForceVisualizer at 0x7dda147b7690>
```

[ ]:
```
lime_CB_explainer = lime.lime_tabular.LimeTabularExplainer(X_test.values,
                                                     feature_names=X_test.columns,
                                                     class_names=['Nondemented',
  ↪'Demented'])
```

[ ]:
```
# CB-FN-2
lime_CB_explainer.explain_instance(X_test.iloc[2].values,\
                                    CatBoost_mdl.predict_proba,\
                                    num_features=6).\
                                show_in_notebook(predict_proba=True)
```

<IPython.core.display.HTML object>

[ ]:
```
# CB-FN-31
lime_CB_explainer.explain_instance(X_test.iloc[31].values,\
                                    CatBoost_mdl.predict_proba,\
                                    num_features=6).\
                                show_in_notebook(predict_proba=True)
```

<IPython.core.display.HTML object>

[ ]:
```
# store all fn_results
fn_results = []
feature_counter = Counter()

CB_FN_indices = [ 2, 30, 31, 37, 49, 61]

for fn_idx in CB_FN_indices:
    CB_instance_values = X_test.iloc[fn_idx].values

    exp = lime_CB_explainer.explain_instance(
        CB_instance_values,
        CatBoost_mdl.predict_proba,
        num_features=6
    )

    exp_list = exp.as_list()

    pushed_non = [f for f, w in exp_list if w < 0]
    pushed_dem = [f for f, w in exp_list if w > 0]

    fn_results.append({
        'Index': fn_idx,
        'Pushed_Nondemented': pushed_non,
        'Pushed_Demented': pushed_dem
    })

    feature_counter.update(pushed_non)

fn_df = pd.DataFrame(fn_results)
```

```python
top_causes = pd.DataFrame(feature_counter.most_common(), columns=['Feature',␣
↪'Count'])

print(fn_df.head())

print("\n=== CB False Negative Feature Frequency ===")
print(top_causes)
```

```
   Index                                 Pushed_Nondemented  \
0      2                                     [eTIV > 1669.00]
1     30                         [EDUC > 16.25, SES <= 2.00]
2     31                       [eTIV > 1669.00, SES <= 2.00]
3     37                       [eTIV > 1669.00, SES <= 2.00]
4     49  [EDUC > 16.25, SES <= 2.00, 0.73 < nWBV <= 0.76]

                                     Pushed_Demented
0  [nWBV <= 0.70, EDUC <= 12.00, 2.00 < SES <= 3.00]
1   [1491.50 < eTIV <= 1669.00, 0.70 < nWBV <= 0.73]
2              [nWBV <= 0.70, 12.00 < EDUC <= 15.00]
3              [nWBV <= 0.70, 12.00 < EDUC <= 15.00]
4                        [1491.50 < eTIV <= 1669.00]

=== CB False Negative Feature Frequency ===
               Feature  Count
0          SES <= 2.00      5
1       eTIV > 1669.00      3
2          EDUC > 16.25      2
3  0.73 < nWBV <= 0.76      2
```

After removing gender and age features, only LightGBM perfomance drop, Catboot and XGBoost are almost the same.

We have caculated XGBoost model feature importance: Age top 5, gender top2; LightGBM model feature importance: Age top 3, gender top6; CatBoost model feature importance: Age top 4, gender top6. From model feature importance, LightGBM didn't priorize age and gender over the other two models.

From shap value side: (MeanAbsSHAP(Age) + MeanAbsSHAP(Gender) proportion of total SHAP.)
XGBoost = 1.3/4.68=0.277; LightGBM=1.67/5.7=0.293; CatBoost = 1.29/4.23=0.305. LightGBM's proportion is not the highest.

so from both feature importance and SHAP proportion, LightGBM is not obviously relying on Age and Gender more than XGBoost or CatBoost. That means the performance drop in LightGBM probably isn't just because of "over-reliance" on those two features — it's more likely about how LightGBM's tree structure uses them.