Import packages and dataset

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans, DBSCAN
```

In [2]:
```
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score
```

In [3]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
```

In [4]:
```
data = pd.read_csv('oasis_cross-sectional.csv')
data
```

Out[4]:

	ID	M/F	Hand	Age	Educ	SES	MMSE	CDR	eTIV	nWBV	ASF	Delay
0	OAS1_0001_MR1	F	R	74	2.0	3.0	29.0	0.0	1344	0.743	1.306	NaN
1	OAS1_0002_MR1	F	R	55	4.0	1.0	29.0	0.0	1147	0.810	1.531	NaN
2	OAS1_0003_MR1	F	R	73	4.0	3.0	27.0	0.5	1454	0.708	1.207	NaN
3	OAS1_0004_MR1	M	R	28	NaN	NaN	NaN	NaN	1588	0.803	1.105	NaN
4	OAS1_0005_MR1	M	R	18	NaN	NaN	NaN	NaN	1737	0.848	1.010	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...
431	OAS1_0285_MR2	M	R	20	NaN	NaN	NaN	NaN	1469	0.847	1.195	2.0
432	OAS1_0353_MR2	M	R	22	NaN	NaN	NaN	NaN	1684	0.790	1.042	40.0
433	OAS1_0368_MR2	M	R	22	NaN	NaN	NaN	NaN	1580	0.856	1.111	89.0
434	OAS1_0379_MR2	F	R	20	NaN	NaN	NaN	NaN	1262	0.861	1.390	2.0
435	OAS1_0395_MR2	F	R	26	NaN	NaN	NaN	NaN	1283	0.834	1.368	39.0

436 rows × 12 columns

Data preprocessing

Data cleaning

In [5]:
```
data.dtypes
```

Out[5]:
```
ID          object
M/F         object
Hand        object
Age          int64
Educ       float64
SES        float64
MMSE       float64
CDR        float64
eTIV         int64
nWBV       float64
ASF        float64
Delay      float64
dtype: object
```

In [6]:
```
#Delete the ID column

data = data.drop(columns=['ID'])
data
```

Out[6]:

	M/F	Hand	Age	Educ	SES	MMSE	CDR	eTIV	nWBV	ASF	Delay
0	F	R	74	2.0	3.0	29.0	0.0	1344	0.743	1.306	NaN
1	F	R	55	4.0	1.0	29.0	0.0	1147	0.810	1.531	NaN
2	F	R	73	4.0	3.0	27.0	0.5	1454	0.708	1.207	NaN
3	M	R	28	NaN	NaN	NaN	NaN	1588	0.803	1.105	NaN
4	M	R	18	NaN	NaN	NaN	NaN	1737	0.848	1.010	NaN
...	...	...	...	...	...	...	...	...	...	...	...
431	M	R	20	NaN	NaN	NaN	NaN	1469	0.847	1.195	2.0
432	M	R	22	NaN	NaN	NaN	NaN	1684	0.790	1.042	40.0
433	M	R	22	NaN	NaN	NaN	NaN	1580	0.856	1.111	89.0
434	F	R	20	NaN	NaN	NaN	NaN	1262	0.861	1.390	2.0
435	F	R	26	NaN	NaN	NaN	NaN	1283	0.834	1.368	39.0

436 rows × 11 columns

In [7]:

```
# Check missing data
data.isna().sum()
```

Out[7]:

```
M/F        0
Hand       0
Age        0
Educ     201
SES      220
MMSE     201
CDR      201
eTIV       0
nWBV       0
ASF        0
Delay    416
dtype: int64
```

In [8]:

```
# Drop the 'Delay' variable given that 95% of the data is missing
data = data.drop(columns = ['Delay'])
data
```

Out[8]:

	M/F	Hand	Age	Educ	SES	MMSE	CDR	eTIV	nWBV	ASF
0	F	R	74	2.0	3.0	29.0	0.0	1344	0.743	1.306
1	F	R	55	4.0	1.0	29.0	0.0	1147	0.810	1.531
2	F	R	73	4.0	3.0	27.0	0.5	1454	0.708	1.207
3	M	R	28	NaN	NaN	NaN	NaN	1588	0.803	1.105
4	M	R	18	NaN	NaN	NaN	NaN	1737	0.848	1.010
...	...	...	...	...	...	...	...	...	...	...
431	M	R	20	NaN	NaN	NaN	NaN	1469	0.847	1.195
432	M	R	22	NaN	NaN	NaN	NaN	1684	0.790	1.042
433	M	R	22	NaN	NaN	NaN	NaN	1580	0.856	1.111
434	F	R	20	NaN	NaN	NaN	NaN	1262	0.861	1.390
435	F	R	26	NaN	NaN	NaN	NaN	1283	0.834	1.368

436 rows × 10 columns

In [9]:

```
# Drop all datapoints with null values
data1 = data.dropna()
```

In [10]:

```
data1.isna().sum()
```

Out[10]:

```
M/F      0
Hand     0
Age      0
Educ     0
SES      0
MMSE     0
CDR      0
eTIV     0
nWBV     0
ASF      0
dtype: int64
```

In [11]:

```
# Drop Handedness variable given that all data are 'R'-Right-handed and thus not informative
data1 = data1.drop(columns=['Hand'])
data1
```

Out[11]:

	M/F	Age	Educ	SES	MMSE	CDR	eTIV	nWBV	ASF
0	F	74	2.0	3.0	29.0	0.0	1344	0.743	1.306
1	F	55	4.0	1.0	29.0	0.0	1147	0.810	1.531
2	F	73	4.0	3.0	27.0	0.5	1454	0.708	1.207
8	M	74	5.0	2.0	30.0	0.0	1636	0.689	1.073
9	F	52	3.0	2.0	30.0	0.0	1321	0.827	1.329
...	...	...	...	...	...	...	...	...	...
411	F	70	1.0	4.0	29.0	0.5	1295	0.748	1.355
412	F	73	3.0	2.0	23.0	0.5	1536	0.730	1.142
413	F	61	2.0	4.0	28.0	0.0	1354	0.825	1.297
414	M	61	5.0	2.0	30.0	0.0	1637	0.780	1.072
415	F	62	3.0	3.0	26.0	0.0	1372	0.766	1.279

216 rows × 9 columns

In [12]:

```
# Double-check cleaned dataset
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 216 entries, 0 to 415
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   M/F     216 non-null    object 
 1   Age     216 non-null    int64  
 2   Educ    216 non-null    float64
 3   SES     216 non-null    float64
 4   MMSE    216 non-null    float64
 5   CDR     216 non-null    float64
 6   eTIV    216 non-null    int64  
 7   nWBV    216 non-null    float64
 8   ASF     216 non-null    float64
dtypes: float64(6), int64(2), object(1)
memory usage: 16.9+ KB
```

Feature Engineering

In [13]:

```
# Transform our Target variable into binary results (nondemented ==0 and all other values being demented)
data1['CDR']
```

Out[13]:
```
0      0.0
1      0.0
2      0.5
8      0.0
9      0.0
      ...
411    0.5
412    0.5
413    0.0
414    0.0
415    0.0
Name: CDR, Length: 216, dtype: float64
```

In [14]:
```
data1.loc[data1['CDR'] == 0, 'CDR'] = 0
data1.loc[data1['CDR'] != 0, 'CDR'] = 1
```

In [15]:
```
data1['CDR']
```

Out[15]:
```
0      0.0
1      0.0
2      1.0
8      0.0
9      0.0
      ...
411    1.0
412    1.0
413    0.0
414    0.0
415    0.0
Name: CDR, Length: 216, dtype: float64
```

In [69]:
```
# Transform features with string values into numeric values for calculating covariance matrix
data1['M/F'] = data1['M/F'].replace(['F','M'], [0,1])
data1
```

Out[69]:

	M/F	Age	Educ	SES	MMSE	CDR	eTIV	nWBV	ASF
0	0	74	2.0	3.0	29.0	0.0	1344	0.743	1.306
1	0	55	4.0	1.0	29.0	0.0	1147	0.810	1.531
2	0	73	4.0	3.0	27.0	1.0	1454	0.708	1.207
8	1	74	5.0	2.0	30.0	0.0	1636	0.689	1.073
9	0	52	3.0	2.0	30.0	0.0	1321	0.827	1.329
...	...	...	...	...	...	...	...	...	...
411	0	70	1.0	4.0	29.0	1.0	1295	0.748	1.355
412	0	73	3.0	2.0	23.0	1.0	1536	0.730	1.142
413	0	61	2.0	4.0	28.0	0.0	1354	0.825	1.297
414	1	61	5.0	2.0	30.0	0.0	1637	0.780	1.072
415	0	62	3.0	3.0	26.0	0.0	1372	0.766	1.279

216 rows × 9 columns

Exploratory Data Analysis

- Distribution of variables
- Exploratory PCAs
- Model implementations
    - Linear regression model with automatic feature selection
    - Automatic ML model selection (RandomforestClassifier)
    - Model testing & Results visualization

Distributions

In [17]:

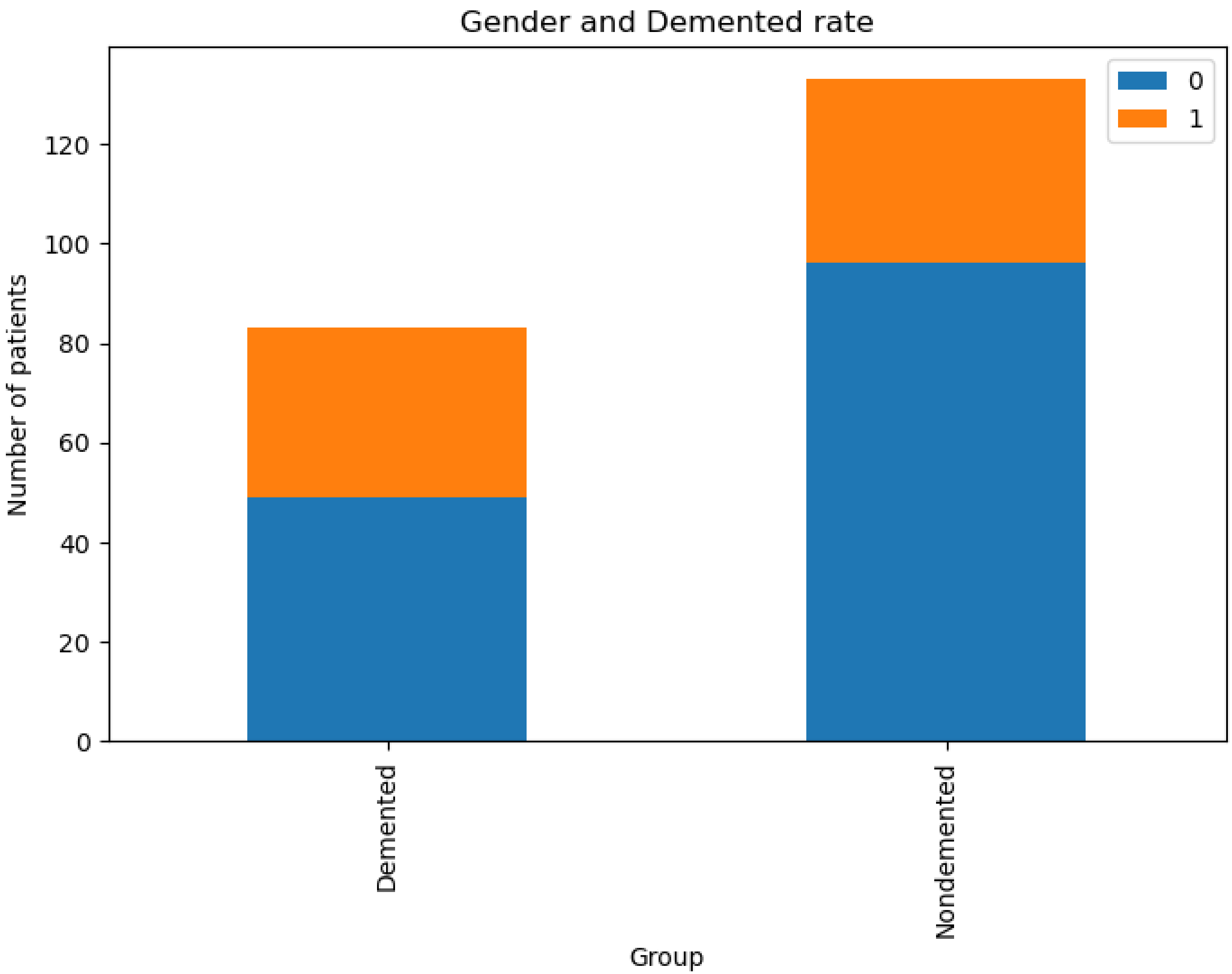
```
# Graph distributions of current dataset to recognize possible skewness

# Create bar_chart function
def bar_chart(feature):
    Demented = data1[data1['CDR']==1][feature].value_counts()
    Nondemented = data1[data1['CDR']==0][feature].value_counts()
    df_bar = pd.DataFrame([Demented,Nondemented])
```

```
    df_bar.index = ['Demented','Nondemented']
    df_bar.plot(kind='bar',stacked=True, figsize=(8,5))
```

In [18]:
```
bar_chart('M/F')
plt.xlabel('Group')
plt.ylabel('Number of patients')
plt.legend()
plt.title('Gender and Demented rate')
```

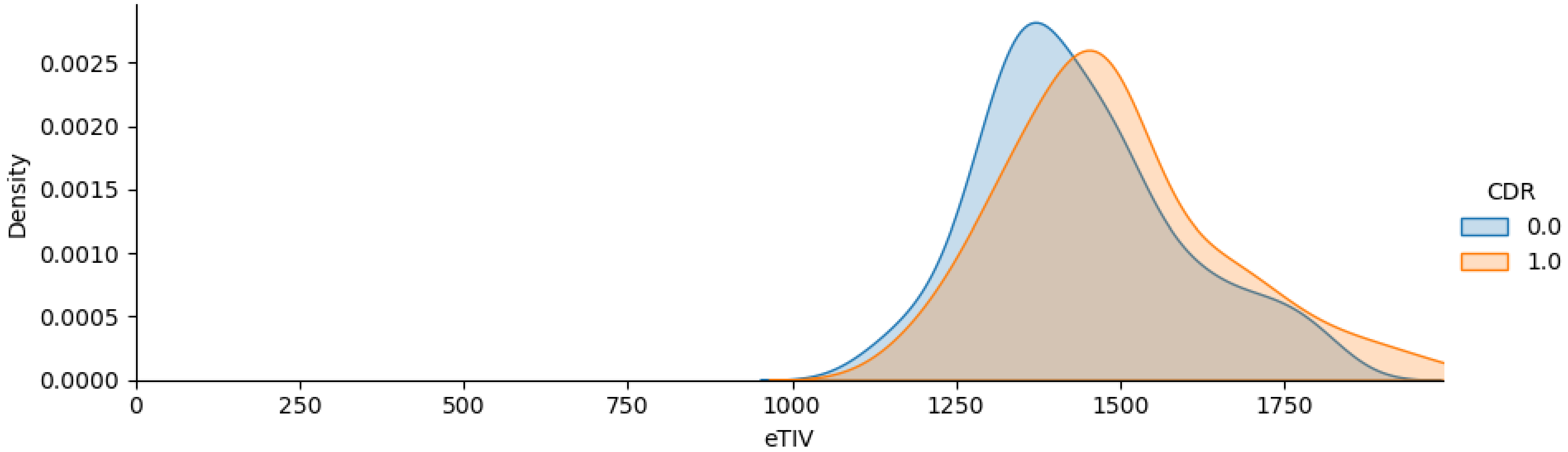
Out[18]:
```
Text(0.5, 1.0, 'Gender and Demented rate')
```

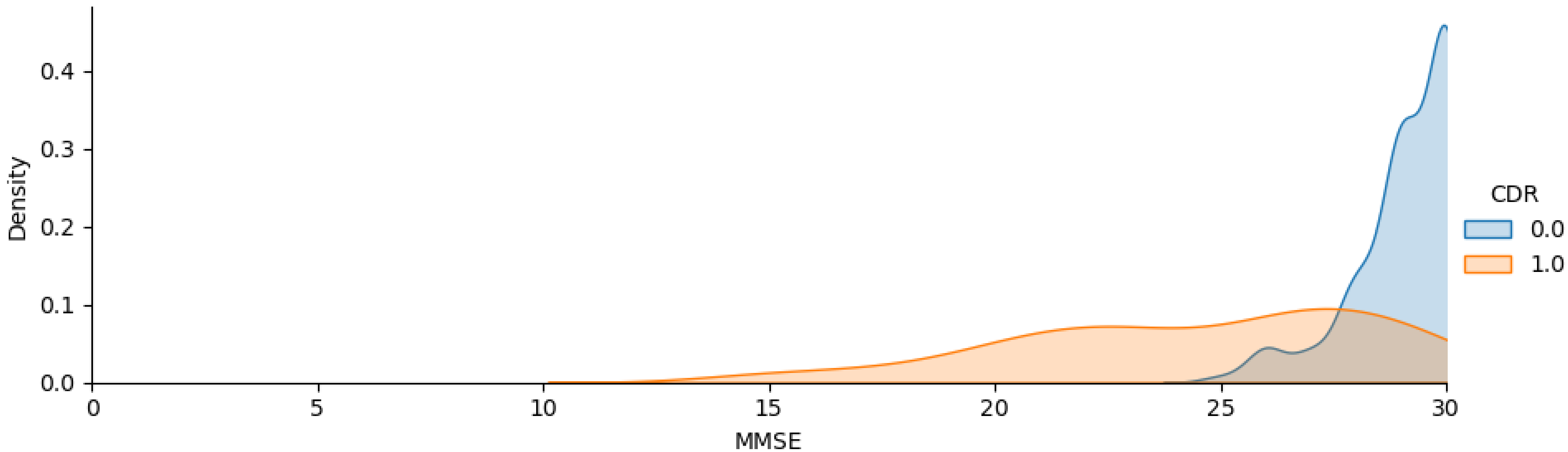


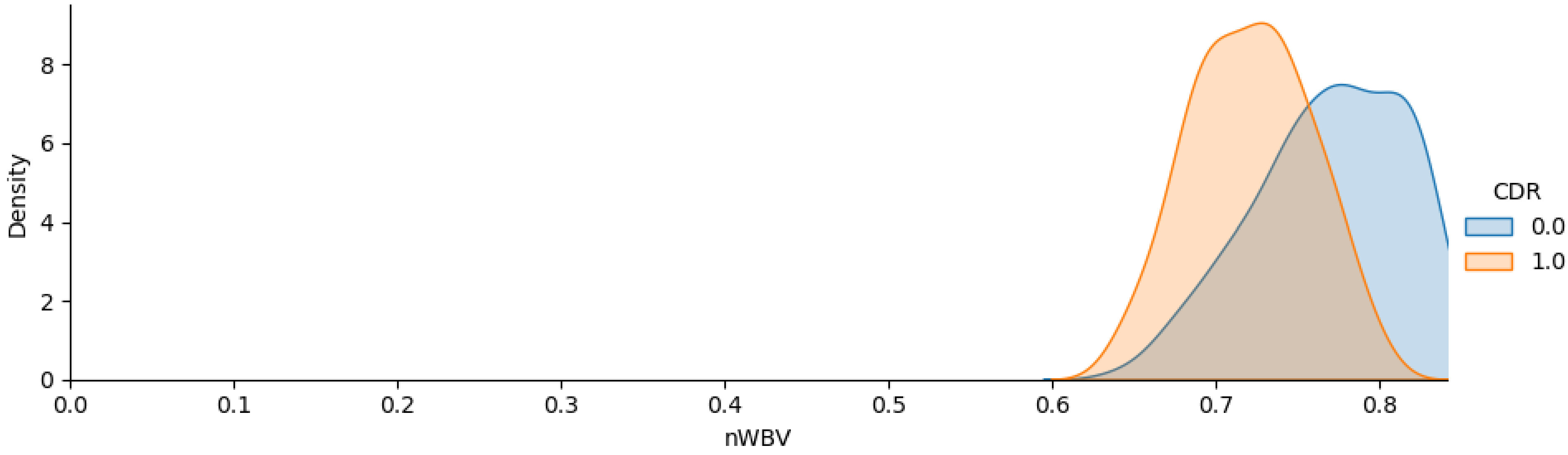
In [19]:
```
# Plot distributions of features
def density(feature):
```

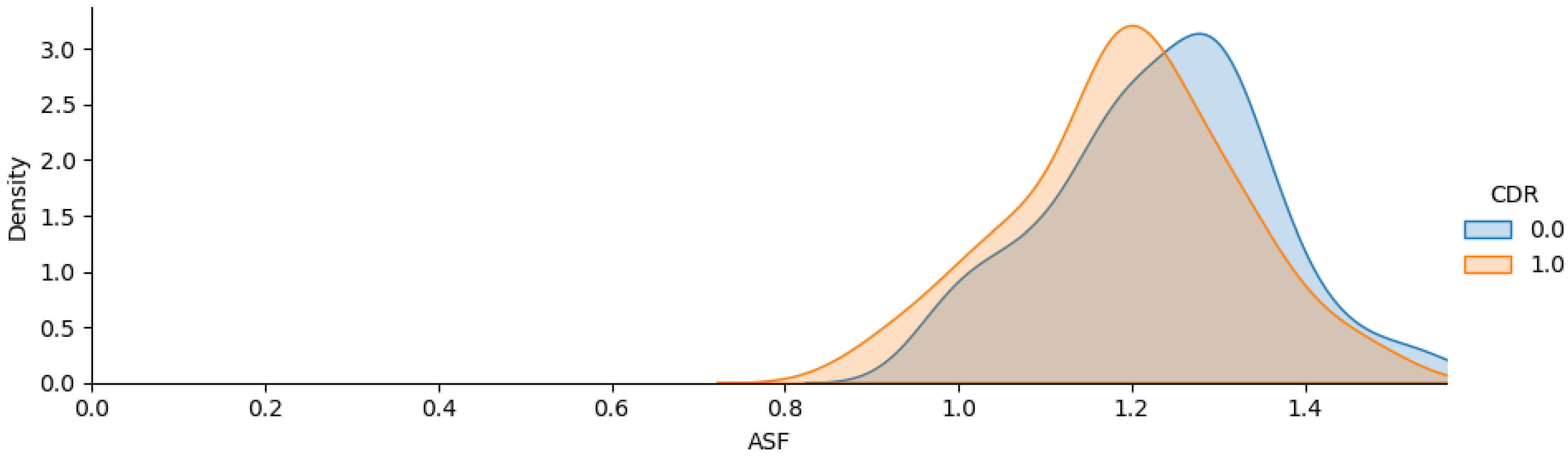
```
    facet= sns.FacetGrid(data1,hue="CDR", aspect=3)
    facet.map(sns.kdeplot,feature,fill= True)
    facet.set(xlim=(0, data1[feature].max()))
    facet.add_legend()
```

In [20]:
```
# Brain measures
density('eTIV')
density('MMSE')
density('nWBV')
density('ASF')
```



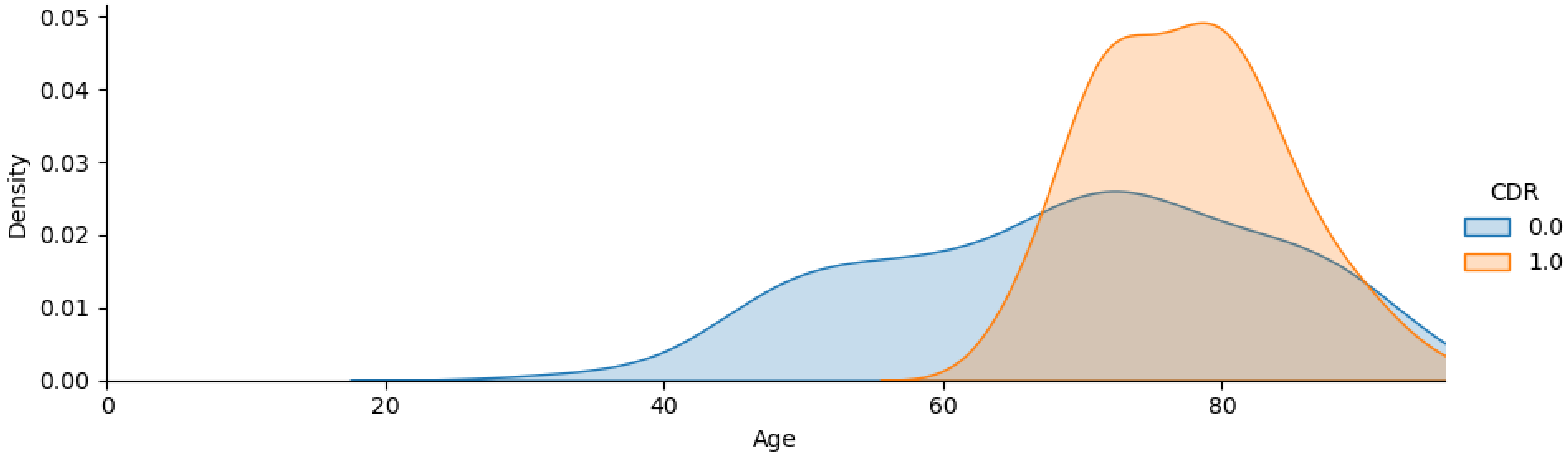


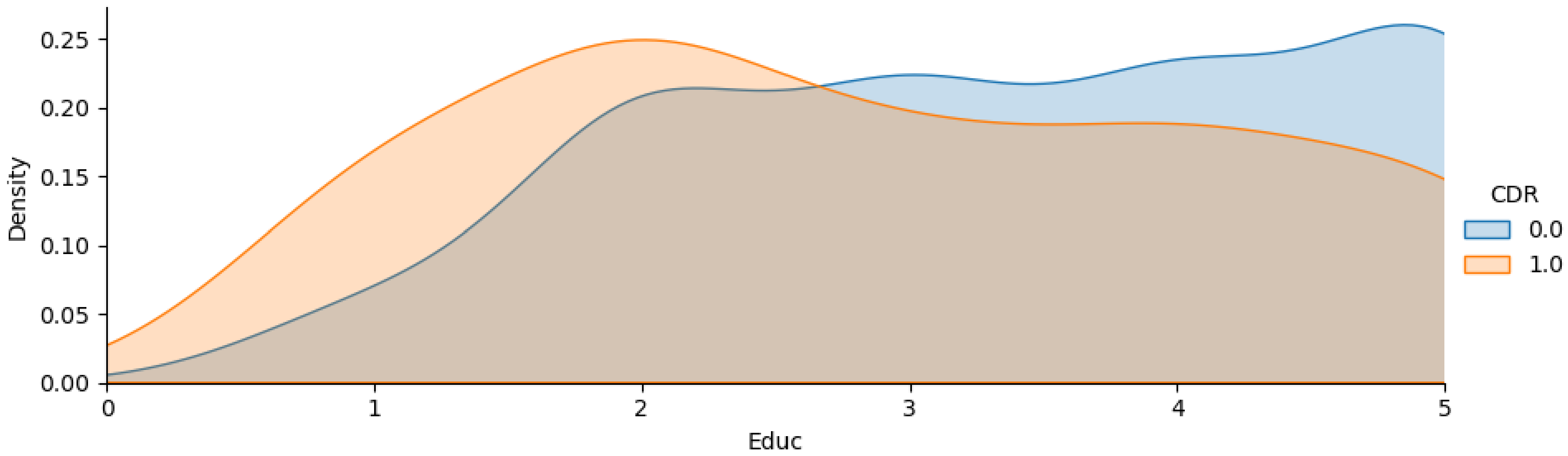




In [21]:

```
# Social factors
density('Age')
density('Educ')
```





Exploratory Data analysis -- PCAs

- General PCA
- PCA without age
- PCA with social factors only (no results because the dataset is only 2-D)
- PCA with brain measurement factors

In [22]:
```python
X = data1.drop('CDR', axis=1)  # Features
y = data1['CDR']  # Target variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=47)

# Print the shapes of the resulting sets
print("Training set shape:", X_train.shape, y_train.shape)
print("Test set shape:", X_test.shape, y_test.shape)
```

```
Training set shape: (172, 8) (172,)
Test set shape: (44, 8) (44,)
```

In [23]:
```python
# Calculate covariance matrix from training dataset for pca

X_train = StandardScaler().fit_transform(X_train)
```

In [24]:
```python
# PCA analysis

pca = PCA(n_components=2)
principalComponents = pca.fit_transform(X_train)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['PC1', 'PC2'])
finalDf = pd.concat([principalDf, data1[['CDR']]], axis = 1)
```

In [25]:
```python
finalDf
```

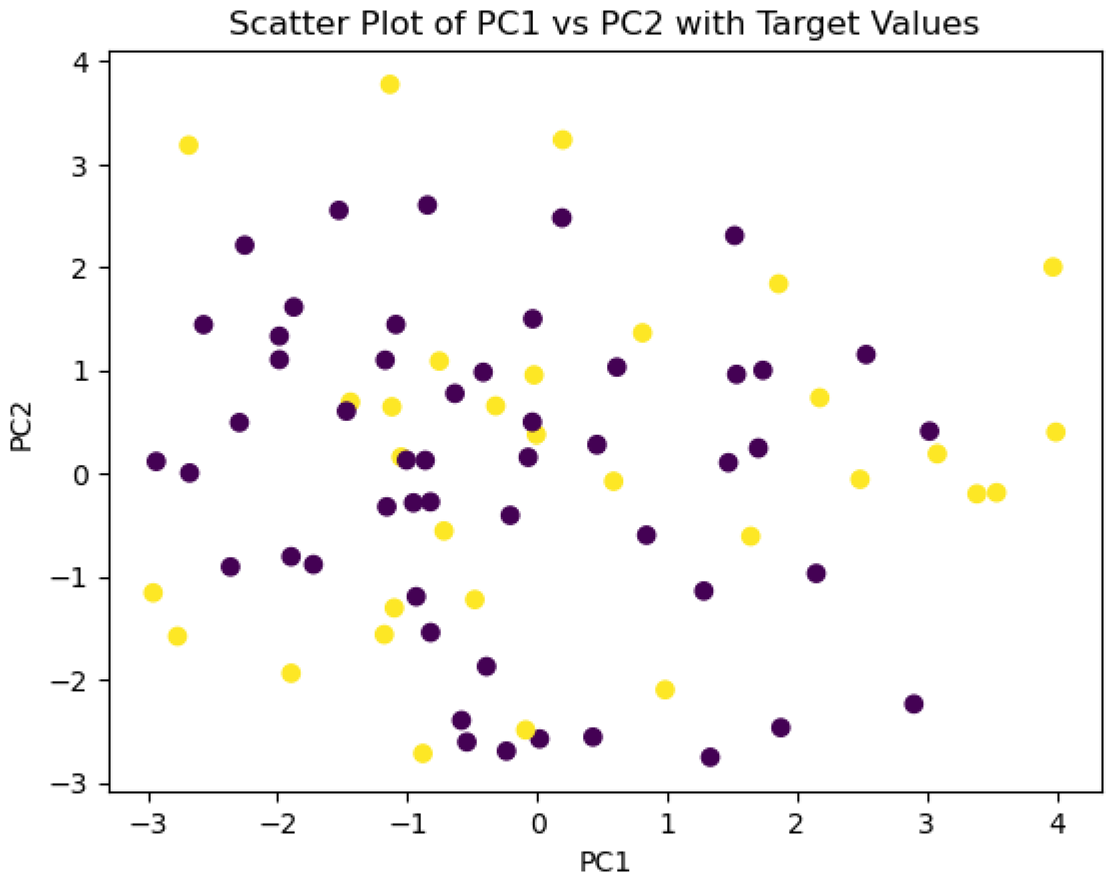
Out[25]:

	PC1	PC2	CDR
0	-1.874614	1.613698	0.0
1	1.736401	1.000079	0.0
2	-0.479995	-1.221007	1.0
3	-1.865039	2.591256	NaN
4	2.742506	-1.813851	NaN
...	...	...	...
411	NaN	NaN	1.0
412	NaN	NaN	1.0
413	NaN	NaN	0.0
414	NaN	NaN	0.0
415	NaN	NaN	0.0

306 rows × 3 columns

In [26]:

```
# Plot out pca results
scatter = plt.scatter(finalDf['PC1'], finalDf['PC2'], c=finalDf['CDR'])
plt.title('Scatter Plot of PC1 vs PC2 with Target Values')
plt.xlabel('PC1')
plt.ylabel('PC2')
#plt.legend(handles=scatter.legend_elements()[0], labels=finalDf['CDR'].unique())
plt.show()
```



In [27]:

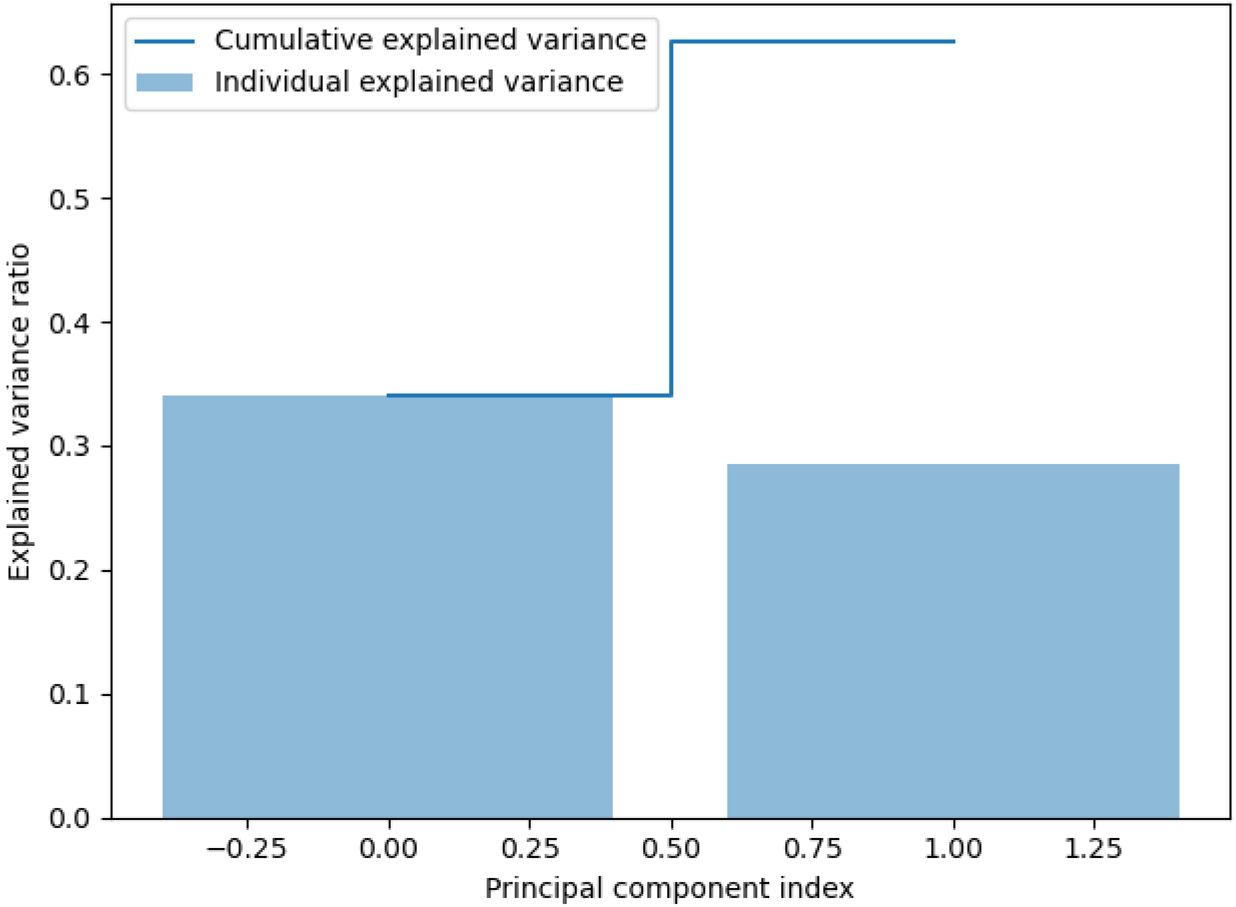
```
# Determine explained variance of each PC
exp_var_pca = pca.explained_variance_ratio_

# Visualize Cumulative sum of eigenvalues
cum_sum_eigenvalues = np.cumsum(exp_var_pca)

# Print out the result
print("The explained variances by PC1 and PC2 is", exp_var_pca)
print("The total explained variance by PCA is", cum_sum_eigenvalues[1])

# Create the visualization plot
plt.bar(range(0,len(exp_var_pca)), exp_var_pca, alpha=0.5, align='center', label='Individual explained variance')
plt.step(range(0,len(cum_sum_eigenvalues)), cum_sum_eigenvalues, where='mid',label='Cumulative explained variance')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal component index')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```

```
The explained variances by PC1 and PC2 is [0.34073802 0.28484481]
The total explained variance by PCA is 0.6255828327408242
```



Inferring the PCA results

In [28]:

```
# Get eigenvectors to see which individual variables contribute the most to the principal components
eigenvectors = pca.components_
eigenvectors
```

Out[28]:

```
array([[ 0.43829675, -0.03454681,  0.29904305, -0.30799529,  0.0638827 ,
         0.55369213, -0.10709964, -0.5479857 ],
       [ 0.09280395,  0.482203  , -0.34705162,  0.28885494, -0.47398471,
         0.12421983, -0.54564549, -0.1310138 ]])
```

In [29]:

```
# Match with the individual features
vars = list(data1.columns)
vars = np.array(vars)
```

```
vars = np.delete(vars,np.where(vars=='CDR'))
vars
```

Out[29]:

```
array(['M/F', 'Age', 'Educ', 'SES', 'MMSE', 'eTIV', 'nWBV', 'ASF'],
      dtype='<U4')
```

In [30]:

```
# Create result table of individual contributions
np.delete(vars,np.where(vars=='CDR'))
contributions = pd.DataFrame({'vars':vars, 'to_PC1':eigenvectors[0],'to_PC2':eigenvectors[1]})
contributions
```

Out[30]:

	vars	to_PC1	to_PC2
0	M/F	0.438297	0.092804
1	Age	-0.034547	0.482203
2	Educ	0.299043	-0.347052
3	SES	-0.307995	0.288855
4	MMSE	0.063883	-0.473985
5	eTIV	0.553692	0.124220
6	nWBV	-0.107100	-0.545645
7	ASF	-0.547986	-0.131014

Data2 without Age

Since Age is a huge predictor of Alzheimer's, it is possible that the PCA results were skewed by the Age feature.

So this section tries the entire process without Age

In [31]:

```
data_noage = data1.drop(columns=['Age'])
data_noage
```

Out[31]:

	M/F	Educ	SES	MMSE	CDR	eTIV	nWBV	ASF
0	0	2.0	3.0	29.0	0.0	1344	0.743	1.306
1	0	4.0	1.0	29.0	0.0	1147	0.810	1.531
2	0	4.0	3.0	27.0	1.0	1454	0.708	1.207
8	1	5.0	2.0	30.0	0.0	1636	0.689	1.073
9	0	3.0	2.0	30.0	0.0	1321	0.827	1.329
...	...	...	...	...	...	...	...	...
411	0	1.0	4.0	29.0	1.0	1295	0.748	1.355
412	0	3.0	2.0	23.0	1.0	1536	0.730	1.142
413	0	2.0	4.0	28.0	0.0	1354	0.825	1.297
414	1	5.0	2.0	30.0	0.0	1637	0.780	1.072
415	0	3.0	3.0	26.0	0.0	1372	0.766	1.279

216 rows × 8 columns

In [32]:

```
X_noage = data_noage.drop('CDR', axis=1)  # Features
y_noage = data_noage['CDR']  # Target variable

X_train1, X_test1, y_train1, y_test1 = train_test_split(X_noage, y_noage, test_size=0.2, random_state=47)

# Print the shapes of the resulting sets
print("Training set shape:", X_train1.shape, y_train1.shape)
print("Test set shape:", X_test1.shape, y_test1.shape)
```

```
Training set shape: (172, 7) (172,)
Test set shape: (44, 7) (44,)
```

In [33]:

```
# Calculate covariance matrix from training dataset for pca

X_train1 = StandardScaler().fit_transform(X_train1)
```

In [34]:

```
# PCA analysis

pca_noage = PCA(n_components=2)
principalComponents_noage = pca_noage.fit_transform(X_train1)
principalDf_noage = pd.DataFrame(data = principalComponents_noage
             , columns = ['PC1', 'PC2'])
finalDf_noage = pd.concat([principalDf_noage, data_noage[['CDR']]], axis = 1)
finalDf_noage
```

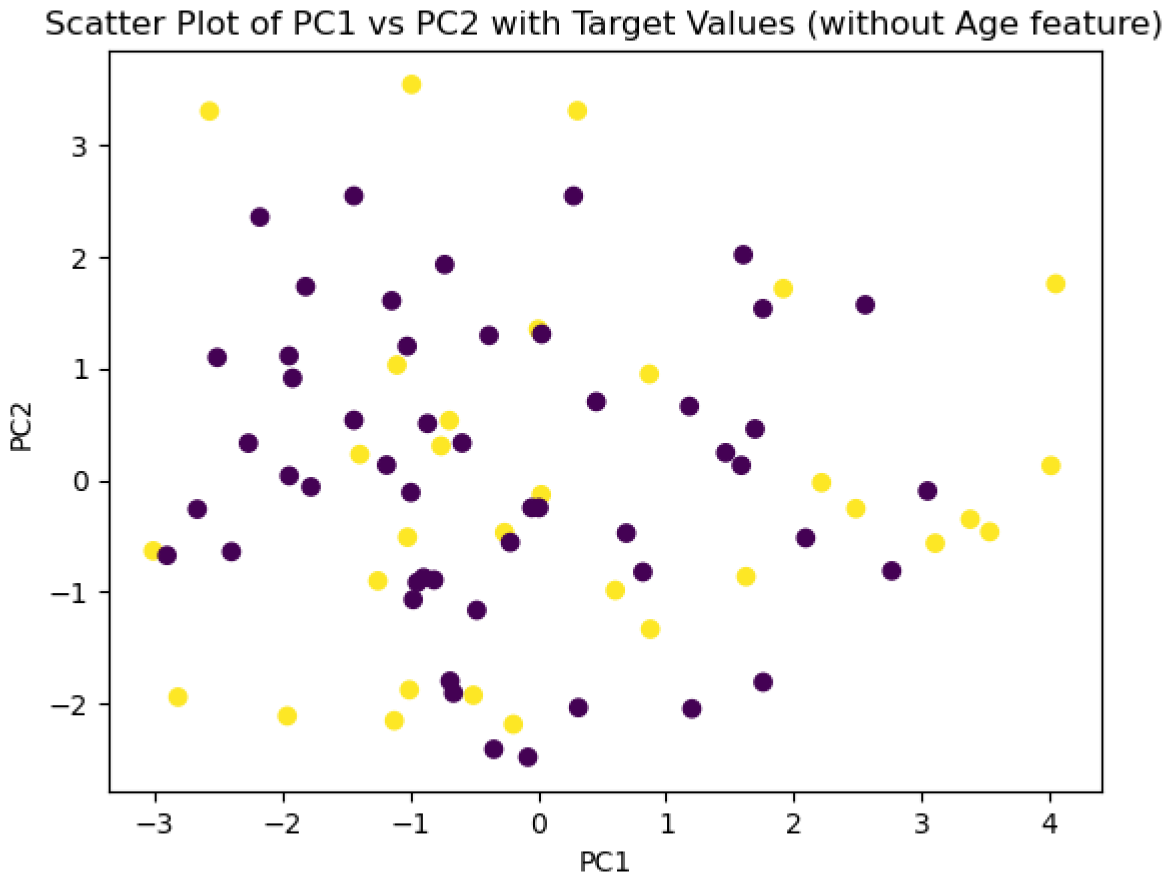
Out[34]:

	PC1	PC2	CDR
0	-1.816204	1.733007	0.0
1	1.759053	1.535174	0.0
2	-0.505545	-1.925298	1.0
3	-1.774946	2.793992	NaN
4	2.646523	-0.964920	NaN
...	...	...	...
411	NaN	NaN	1.0
412	NaN	NaN	1.0
413	NaN	NaN	0.0
414	NaN	NaN	0.0
415	NaN	NaN	0.0

306 rows × 3 columns

In [35]:

```
# Plot out pca results
scatter_noage = plt.scatter(finalDf_noage['PC1'], finalDf_noage['PC2'], c=finalDf_noage['CDR'])
plt.title('Scatter Plot of PC1 vs PC2 with Target Values (without Age feature)')
plt.xlabel('PC1')
plt.ylabel('PC2')
#plt.legend(handles=scatter.legend_elements()[0], labels=finalDf['CDR'].unique())
plt.show()
```



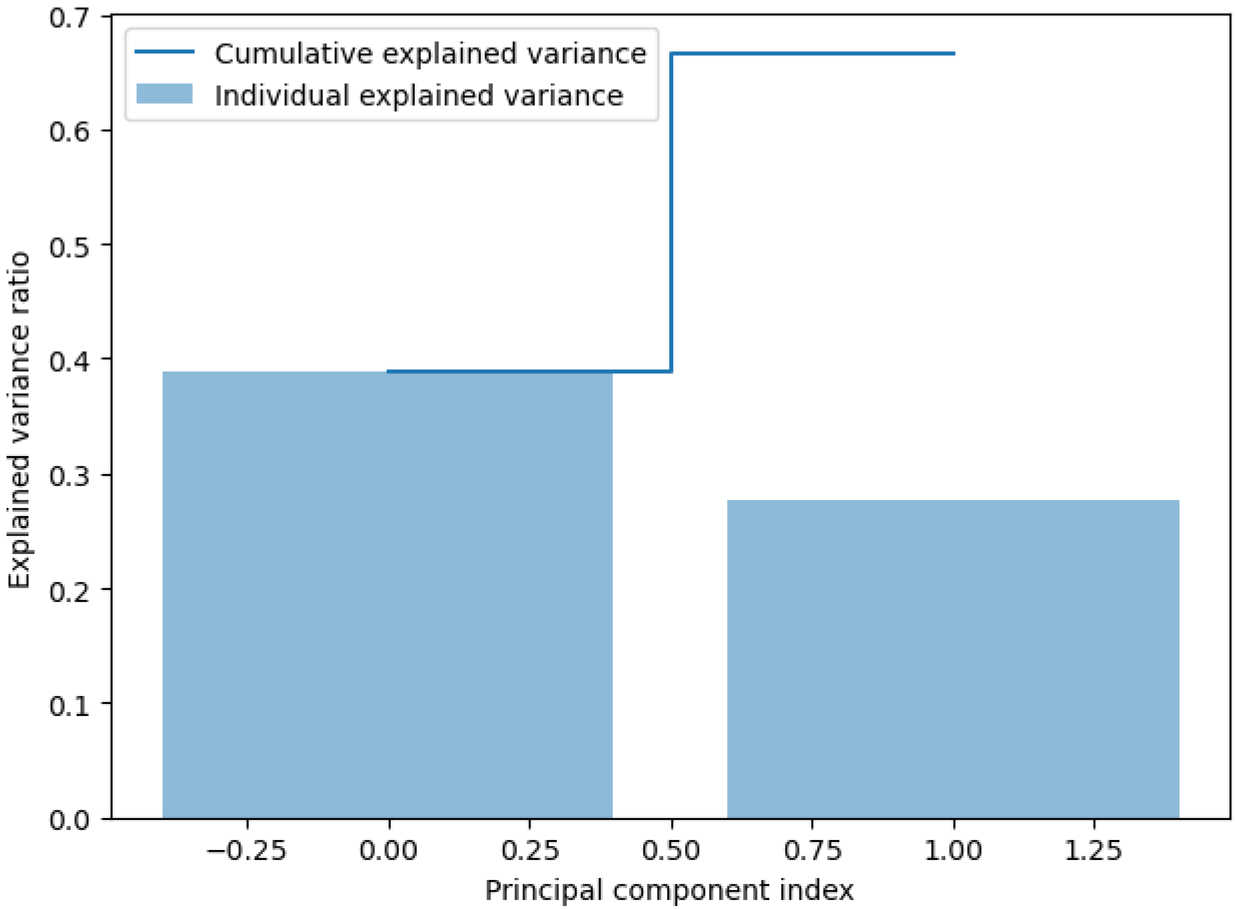
In [36]:

```
# Determine explained variance of each PC
exp_var_pca_noage = pca_noage.explained_variance_ratio_

# Visualize Cumulative sum of eigenvalues
cum_sum_eigenvalues_noage = np.cumsum(exp_var_pca_noage)
print("The explained variances by PC1 and PC2 is", exp_var_pca_noage)
print("The total explained variance by PCA is", cum_sum_eigenvalues_noage[1])

# Create the visualization plot
plt.bar(range(0,len(exp_var_pca_noage)), exp_var_pca_noage, alpha=0.5, align='center', label='Individual explained variance')
plt.step(range(0,len(cum_sum_eigenvalues_noage)), cum_sum_eigenvalues_noage, where='mid',label='Cumulative explained variance')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal component index')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```

```
The explained variances by PC1 and PC2 is [0.38922668 0.27753814]
The total explained variance by PCA is 0.666764812569268
```



In [37]:
```
eigenvectors_noage = pca_noage.components_
eigenvectors_noage

# Match with the individual features
vars_noage = list(data_noage.columns)
vars_noage = np.array(vars_noage)

vars_noage = np.delete(vars_noage,np.where(vars_noage=='CDR'))
vars_noage

# Create result table of individual contributions
np.delete(vars_noage,np.where(vars_noage=='CDR'))
contributions_noage = pd.DataFrame({'vars':vars_noage, 'to_PC1':eigenvectors_noage[0],'to_PC2':eigenvectors_noage[1]})
contributions_noage
```

Out[37]:

	vars	to_PC1	to_PC2
0	M/F	0.439875	0.149675
1	Educ	0.289599	-0.498214
2	SES	-0.300694	0.450204
3	MMSE	0.049019	-0.526259
4	eTIV	0.556647	0.159572
5	nWBV	-0.127619	-0.441487
6	ASF	-0.551072	-0.171388

In [38]:

```
#Build codes for automatically discarding vars
#color by different vars
```

In [39]:

```
social = data1.drop(columns=['Educ','SES'])
social
```

Out[39]:

	M/F	Age	MMSE	CDR	eTIV	nWBV	ASF
0	0	74	29.0	0.0	1344	0.743	1.306
1	0	55	29.0	0.0	1147	0.810	1.531
2	0	73	27.0	1.0	1454	0.708	1.207
8	1	74	30.0	0.0	1636	0.689	1.073
9	0	52	30.0	0.0	1321	0.827	1.329
...	...	...	...	...	...	...	...
411	0	70	29.0	1.0	1295	0.748	1.355
412	0	73	23.0	1.0	1536	0.730	1.142
413	0	61	28.0	0.0	1354	0.825	1.297
414	1	61	30.0	0.0	1637	0.780	1.072
415	0	62	26.0	0.0	1372	0.766	1.279

216 rows × 7 columns

In [40]:

```
socialX = social.drop('CDR', axis=1)  # Features
socialY= social['CDR']  # Target variable
```

```
nosocial_train1, social_test1, social_train1, social_test1 = train_test_split(socialX, socialY, test_size=0.2, random_state=47)

# Print the shapes of the resulting sets
print("Training set shape:", nosocial_train1.shape, nosocial_train1.shape)
print("Test set shape:", social_test1.shape, social_test1.shape)
```

```
Training set shape: (172, 6) (172, 6)
Test set shape: (44,) (44,)
```

In [41]:

```
# Calculate covariance matrix from training dataset for pca
nosocial_train1 = StandardScaler().fit_transform(nosocial_train1)
```

In [42]:

```
# PCA analysis
pca_social = PCA(n_components=2)
principalComponents_social = pca_social.fit_transform(nosocial_train1)
principalDf_nosocial = pd.DataFrame(data = principalComponents_social
             , columns = ['PC1','PC2'])
finalDf_nosocial = pd.concat([principalDf_nosocial, social[['CDR']]], axis = 1)
finalDf_nosocial
```

Out[42]:

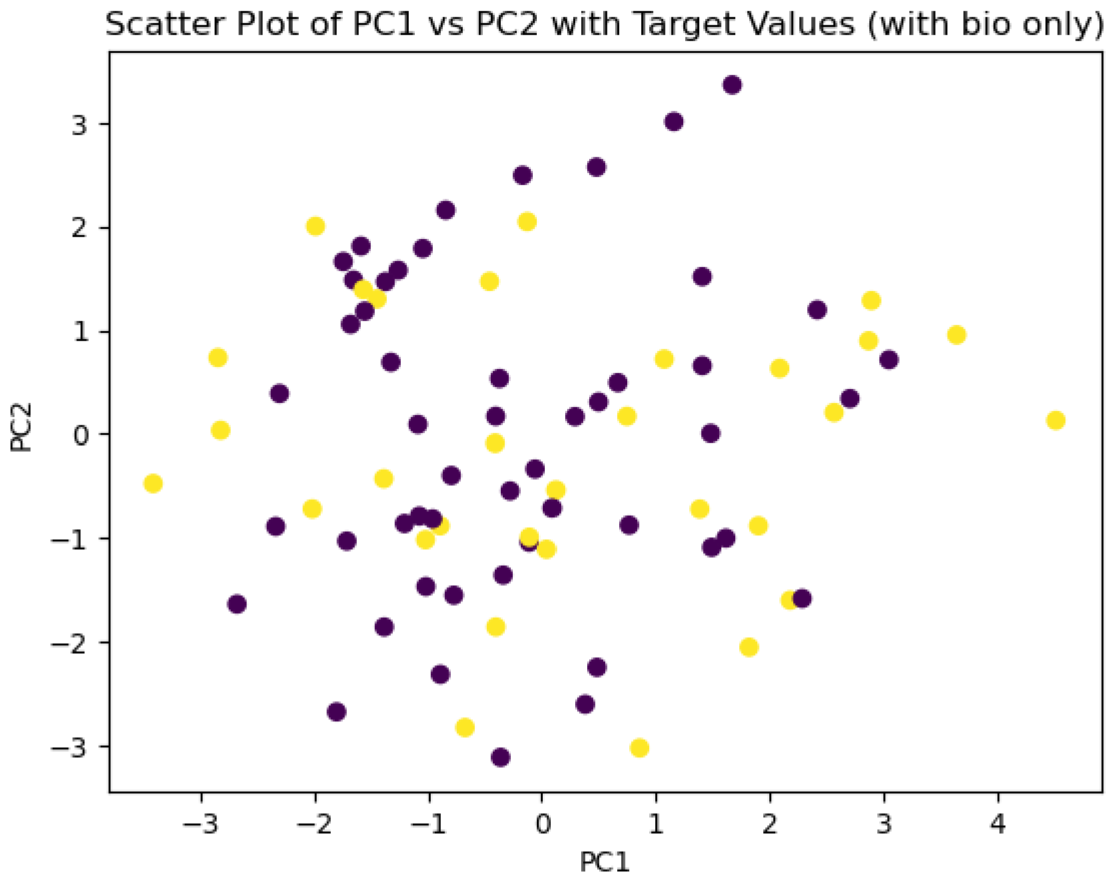
	PC1	PC2	CDR
0	-0.770960	-1.550299	0.0
1	2.419336	1.196371	0.0
2	-1.386507	-0.426424	1.0
3	-0.276261	-2.160778	NaN
4	1.719195	2.819163	NaN
...	...	...	...
411	NaN	NaN	1.0
412	NaN	NaN	1.0
413	NaN	NaN	0.0
414	NaN	NaN	0.0
415	NaN	NaN	0.0

306 rows × 3 columns

In [43]:

```
# Plot out pca results
scatter_social = plt.scatter(finalDf_nosocial['PC1'],finalDf_nosocial['PC2'], c=finalDf_nosocial['CDR'])
plt.title('Scatter Plot of PC1 vs PC2 with Target Values (with bio only)')
plt.xlabel('PC1')
plt.ylabel('PC2')
```

```
#plt.legend(handles=scatter.legend_elements()[0], labels=finalDf['CDR'].unique())
plt.show()
```



In [44]:

```
eigenvectors_nosocial = pca_social.components_
eigenvectors_nosocial

# Match with the individual features
vars_nosocial = list(social.columns)
vars_nosocial = np.array(vars_nosocial)

vars_nosocial = np.delete(vars_nosocial,np.where(vars_nosocial=='CDR'))
vars_nosocial

# Create result table of individual contributions
np.delete(vars_nosocial,np.where(vars_nosocial=='CDR'))
contributions_nosocial = pd.DataFrame({'vars':vars_nosocial, 'to_PC1':eigenvectors_nosocial[0],'to_PC2':eigenvectors_nosocial[1
contributions_nosocial
```

Out[44]:

	vars	to_PC1	to_PC2
0	M/F	0.461271	0.184471
1	Age	0.134123	-0.587461
2	MMSE	-0.131233	0.468814
3	eTIV	0.577746	0.194324
4	nWBV	-0.292716	0.571614
5	ASF	-0.576667	-0.191230

Model implementation -- (1) Linear regression Model

With Automatic Feature Selection

In [45]:

```
import itertools
import time
import statsmodels.api as sm
```

In [46]:

```
# training dataset
X = data1.drop('CDR', axis=1)  # Features
y = data1['CDR']  # Target variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=47)
```

In [47]:

```
# credit to https://www.science.smith.edu/~jcrouser/SDS293/labs/lab8-py.html
def processSubset(feature_set):
    # Fit model on feature_set and calculate RSS
    model = sm.OLS(y_train,X_train[list(feature_set)])
    regr = model.fit()
    RSS = ((regr.predict(X_train[list(feature_set)]) - y) ** 2).sum()
    return {"model":regr, "RSS":RSS}

def backward(predictors):

    tic = time.time()

    results = []

    for combo in itertools.combinations(predictors, len(predictors)-1):
        results.append(processSubset(combo))

    # Wrap everything up in a nice dataframe
```

```
    models = pd.DataFrame(results)

    # Choose the model with the highest RSS
    best_model = models.loc[models['RSS'].argmin()]

    toc = time.time()
    #print("Processed ", models.shape[0], "models on", len(predictors)-1, "predictors in", (toc-tic), "seconds.")

    # Return the best model, along with some other useful information about the model
    return models

def forward(predictors):

    # Pull out predictors we still need to process
    remaining_predictors = [p for p in X_train.columns if p not in predictors]

    tic = time.time()

    results = []

    for p in remaining_predictors:
        results.append(processSubset(predictors+[p]))

    # Wrap everything up in a nice dataframe
    models = pd.DataFrame(results)

    # Choose the model with the highest RSS
    #best_model = models.loc[models['RSS'].argmin()]

    toc = time.time()
    #print("Processed ", models.shape[0], "models on", len(predictors)+1, "predictors in", (toc-tic), "seconds.")

    # Return the best model, along with some other useful information about the model
    return models
```

In [48]:

```
predictors = X_train.columns

#Backward
models_back = backward(predictors)
print(models_back)


print(models_back.loc[6, "model"].summary())
best_back = models_back.loc[6, "model"].summary()

#Convert to pd.DataFrame
best_back_df = (best_back.tables[1])
best_back_df
```

```
                                               model        RSS
0  <statsmodels.regression.linear_model.Regressio...  21.324458
1  <statsmodels.regression.linear_model.Regressio...  20.374411
2  <statsmodels.regression.linear_model.Regressio...  22.886511
3  <statsmodels.regression.linear_model.Regressio...  26.465201
4  <statsmodels.regression.linear_model.Regressio...  19.488219
5  <statsmodels.regression.linear_model.Regressio...  19.336666
6  <statsmodels.regression.linear_model.Regressio...  19.296314
7  <statsmodels.regression.linear_model.Regressio...  19.393491
                                 OLS Regression Results
=======================================================================================
Dep. Variable:                    CDR   R-squared (uncentered):                   0.708
Model:                            OLS   Adj. R-squared (uncentered):              0.695
Method:                 Least Squares   F-statistic:                              57.05
Date:                Sat, 09 Dec 2023   Prob (F-statistic):                    7.27e-41
Time:                        15:48:43   Log-Likelihood:                         -55.926
No. Observations:                 172   AIC:                                      125.9
Df Residuals:                     165   BIC:                                      147.9
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
M/F            0.0651      0.068      0.952      0.342      -0.070       0.200
Educ           0.0195      0.031      0.621      0.535      -0.042       0.081
SES            0.0454      0.035      1.286      0.200      -0.024       0.115
MMSE          -0.0705      0.009     -7.944      0.000      -0.088      -0.053
eTIV           0.0015      0.000      8.978      0.000       0.001       0.002
nWBV          -2.7252      0.646     -4.222      0.000      -4.000      -1.451
ASF            1.5881      0.240      6.610      0.000       1.114       2.063
==============================================================================
Omnibus:                       10.640   Durbin-Watson:                   1.803
Prob(Omnibus):                  0.005   Jarque-Bera (JB):               11.549
Skew:                           0.629   Prob(JB):                      0.00311
Kurtosis:                       2.836   Cond. No.                     3.78e+04
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 3.78e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Out[48]:

	coef	std err	t	P>|t|	[0.025	0.975]
M/F	0.0651	0.068	0.952	0.342	-0.070	0.200
Educ	0.0195	0.031	0.621	0.535	-0.042	0.081
SES	0.0454	0.035	1.286	0.200	-0.024	0.115
MMSE	-0.0705	0.009	-7.944	0.000	-0.088	-0.053
eTIV	0.0015	0.000	8.978	0.000	0.001	0.002
nWBV	-2.7252	0.646	-4.222	0.000	-4.000	-1.451
ASF	1.5881	0.240	6.610	0.000	1.114	2.063

In [49]:

```
# Forward
models_fwd = forward(predictors)
print(models_fwd)
```

```
Empty DataFrame
Columns: []
Index: []
```

Model implementation -- (2) Automatic ML Model Selection

In [73]:

```
data1.dtypes
```

Out[73]:

```
M/F        int64
Age        int64
Educ     float64
SES      float64
MMSE     float64
CDR      float64
eTIV       int64
nWBV     float64
ASF      float64
dtype: object
```

In [78]:

```
data2 = data1.copy()
data2.astype({"M/F": object, "CDR": object, "Age": float, "eTIV": float}).dtypes
```

Out[78]:
```
M/F       object
Age      float64
Educ     float64
SES      float64
MMSE     float64
CDR       object
eTIV     float64
nWBV     float64
ASF      float64
dtype: object
```

In [80]:
```
X2 = data2.drop('CDR', axis=1)  # Features
y2 = data2['CDR']  # Target variable
```

In [129…
```
# Step 1: Data Preprocessing
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

from sklearn.base import clone

# Load your dataset
# Assuming 'X' contains features and 'y' contains labels
X_train, X_test, y_train, y_test = train_test_split(X2, y2, test_size=0.2, random_state=47)

# Define preprocessing steps
numeric_features = X_train.select_dtypes(include=['int64', 'float64']).columns
categorical_features = X_train.select_dtypes(include=['object']).columns

numeric_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Step 2: Model Selection
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score

# Define a list of models to evaluate
models = [
    RandomForestClassifier(),
    SVC(),
    KNeighborsClassifier()
]

# Step 3: Model Training and Evaluation
best_model = None
best_score = 0

for model in models:
    # Create a pipeline with preprocessing and the current model
    clf = Pipeline(steps=[('preprocessor', preprocessor),
                          ('classifier', model)])

    # Evaluate the model using cross-validation on the training set
    scores = cross_val_score(clf, X_train, y_train, cv=5, scoring='accuracy')
    avg_score = scores.mean()

    # Update the best model if the current model performs better
    if avg_score > best_score:
        best_score = avg_score
        #best_model = model
        best_model = clone(clf)

# Step 4: Hyperparameter Tuning (Optional)
# You can use GridSearchCV or RandomizedSearchCV to tune hyperparameters

# Step 5: Final Model Evaluation

#final_model = Pipeline(steps=[('preprocessor', preprocessor),
#                              ('classifier', best_model)])
final_model = Pipeline(steps=[('preprocessor', preprocessor),
                              ('classifier', best_model.named_steps['classifier'])])

# Train the final model on the full training set
final_model.fit(X_train, y_train)

# Evaluate the final model on the test set
test_score = final_model.score(X_test, y_test)

print(f"Best Model: {best_model}")
print(f"Cross-Validation Accuracy: {best_score}")
print(f"Test Accuracy: {test_score}")
```

```
Best Model: Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('scaler',
                                                                   StandardScaler())]),
                                                  Index(['M/F', 'Age', 'Educ', 'SES', 'MMSE', 'eTIV', 'nWBV', 'ASF'], dtype='ob
ject')),
                                                 ('cat',
                                                  Pipeline(steps=[('onehot',
                                                                   OneHotEncoder(handle_unknown='ignore'))]),
                                                  Index([], dtype='object'))])),
                ('classifier', RandomForestClassifier())])
Cross-Validation Accuracy: 0.8601680672268905
Test Accuracy: 0.7727272727272727
```

In [107…

```python
from sklearn.tree import export_graphviz
from IPython.display import Image
import graphviz
```

In [121…

```python
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
```

In [122…

```python
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score, ConfusionMatrixDisplay
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.7272727272727273
```

In [126…

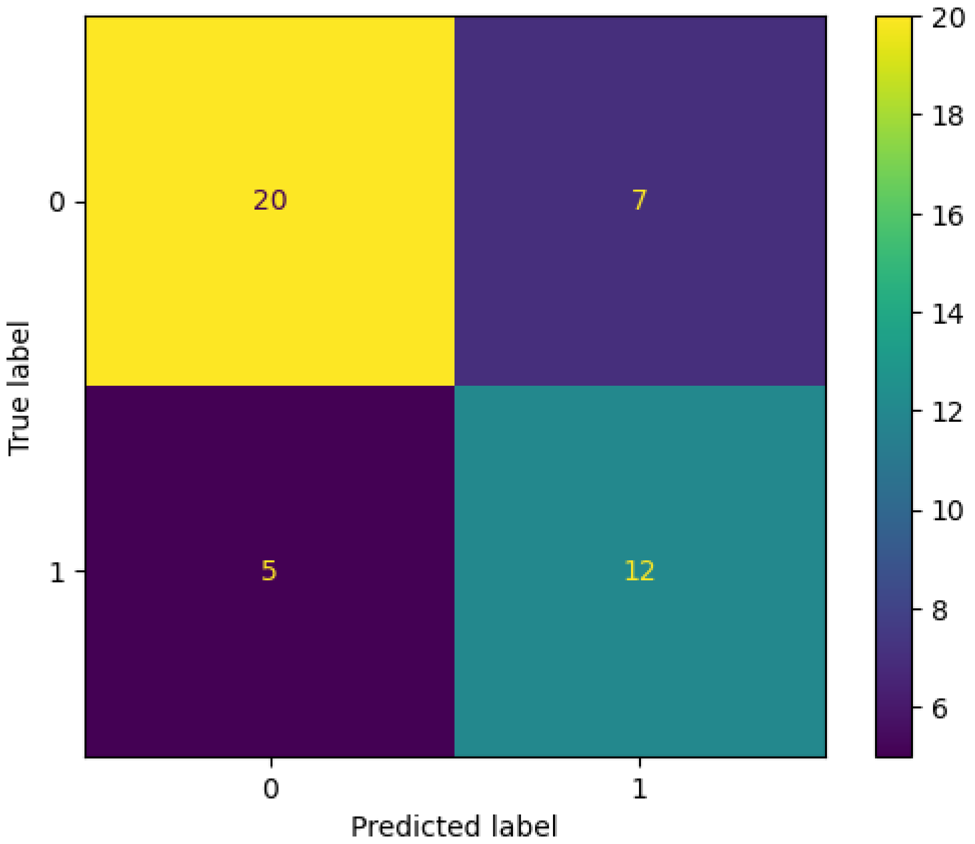
```python
# Generate predictions with the best model
y_pred = rf.predict(X_test)

# Create the confusion matrix
cm = confusion_matrix(y_test, y_pred)

ConfusionMatrixDisplay(confusion_matrix=cm).plot()
```

Out[126]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fc2439d2610>
```



Codes not working

In [103…
```
# Visualize the Decision tree

!pip install graphviz

#!conda install python-graphviz
```
```
Requirement already satisfied: graphviz in /Users/yufeimeng/opt/anaconda3/lib/python3.9/site-packages (0.20.1)
```

In [120…
```
#!conda remove graphviz
#!conda install python-graphviz
```

In [116…
```
# Export the first three decision trees from the forest
from IPython.display import display

#for i in range(3):
```

```python
tree = rf.estimators_[0]
dot_data = export_graphviz(tree,
                           feature_names=X_train.columns,
                           filled=True,
                           max_depth=2,
                           impurity=False,
                           proportion=True)
graph = graphviz.Source(dot_data)
#display(graph)

with open("tree") as f:
    dot_graph = f.read()
    display(graphviz.Source(dot_graph))
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Input In [116], in <cell line: 15>()
     12 graph = graphviz.Source(dot_data)
     13 #display(graph)
---> 15 with open("tree") as f:
     16     dot_graph = f.read()
     17     display(graphviz.Source(dot_graph))

FileNotFoundError: [Errno 2] No such file or directory: 'tree'
```

In [89]:

```python
from sklearn.linear_model import LogisticRegression
# Create an instance of the model
model = LogisticRegression()
# Train the model
model.fit(X_train, y_train)
```

```
/Users/yufeimeng/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs fai
led to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[89]:

```
▾ LogisticRegression
LogisticRegression()
```

Alternative model selection method: MDR

(Trying to implement)

In [50]:
```
pip install scikit-mdr
```

```
Collecting scikit-mdr
  Downloading scikit_MDR-0.4.5-py3-none-any.whl (15 kB)
Requirement already satisfied: scikit-learn in /Users/yufeimeng/opt/anaconda3/lib/python3.9/site-packages (from scikit-mdr) (1.
2.2)
Requirement already satisfied: scipy in /Users/yufeimeng/opt/anaconda3/lib/python3.9/site-packages (from scikit-mdr) (1.10.1)
Requirement already satisfied: numpy in /Users/yufeimeng/opt/anaconda3/lib/python3.9/site-packages (from scikit-mdr) (1.21.5)
Requirement already satisfied: matplotlib in /Users/yufeimeng/opt/anaconda3/lib/python3.9/site-packages (from scikit-mdr) (3.7.
1)
Requirement already satisfied: fonttools>=4.22.0 in /Users/yufeimeng/opt/anaconda3/lib/python3.9/site-packages (from matplotlib
->scikit-mdr) (4.25.0)
Requirement already satisfied: python-dateutil>=2.7 in /Users/yufeimeng/opt/anaconda3/lib/python3.9/site-packages (from matplot
lib->scikit-mdr) (2.8.2)
Requirement already satisfied: cycler>=0.10 in /Users/yufeimeng/opt/anaconda3/lib/python3.9/site-packages (from matplotlib->sci
kit-mdr) (0.11.0)
Requirement already satisfied: pyparsing>=2.3.1 in /Users/yufeimeng/opt/anaconda3/lib/python3.9/site-packages (from matplotlib-
>scikit-mdr) (3.0.9)
Requirement already satisfied: kiwisolver>=1.0.1 in /Users/yufeimeng/opt/anaconda3/lib/python3.9/site-packages (from matplotlib
->scikit-mdr) (1.4.4)
Requirement already satisfied: packaging>=20.0 in /Users/yufeimeng/opt/anaconda3/lib/python3.9/site-packages (from matplotlib->
scikit-mdr) (23.0)
Requirement already satisfied: importlib-resources>=3.2.0 in /Users/yufeimeng/opt/anaconda3/lib/python3.9/site-packages (from m
atplotlib->scikit-mdr) (5.2.0)
Requirement already satisfied: contourpy>=1.0.1 in /Users/yufeimeng/opt/anaconda3/lib/python3.9/site-packages (from matplotlib-
>scikit-mdr) (1.0.5)
Requirement already satisfied: pillow>=6.2.0 in /Users/yufeimeng/opt/anaconda3/lib/python3.9/site-packages (from matplotlib->sc
ikit-mdr) (9.4.0)
Requirement already satisfied: zipp>=3.1.0 in /Users/yufeimeng/opt/anaconda3/lib/python3.9/site-packages (from importlib-resour
ces>=3.2.0->matplotlib->scikit-mdr) (3.11.0)
Requirement already satisfied: six>=1.5 in /Users/yufeimeng/opt/anaconda3/lib/python3.9/site-packages (from python-dateutil>=2.
7->matplotlib->scikit-mdr) (1.16.0)
Requirement already satisfied: joblib>=1.1.1 in /Users/yufeimeng/opt/anaconda3/lib/python3.9/site-packages (from scikit-learn->
scikit-mdr) (1.1.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /Users/yufeimeng/opt/anaconda3/lib/python3.9/site-packages (from scikit-
learn->scikit-mdr) (2.2.0)
Installing collected packages: scikit-mdr
Successfully installed scikit-mdr-0.4.5
Note: you may need to restart the kernel to use updated packages.
```

In [51]:
```
from mdr import MDR
```

In [60]:
```
my_mdr = MDR()
features = X_train.copy()
labels = y_train.copy()

features
```

Out[60]:

	M/F	Age	Educ	SES	MMSE	eTIV	nWBV	ASF
352	0	77	2.0	4.0	22.0	1350	0.736	1.300
300	1	72	1.0	3.0	29.0	1734	0.762	1.012
200	0	75	5.0	1.0	30.0	1317	0.742	1.332
212	0	77	1.0	4.0	20.0	1376	0.701	1.275
189	1	51	5.0	2.0	29.0	1714	0.819	1.024
...	...	...	...	...	...	...	...	...
153	0	74	2.0	3.0	29.0	1395	0.787	1.258
17	0	89	5.0	1.0	30.0	1536	0.715	1.142
152	0	81	2.0	3.0	28.0	1495	0.687	1.174
262	1	83	3.0	2.0	26.0	1992	0.706	0.881
263	0	73	2.0	2.0	19.0	1274	0.745	1.377

172 rows × 8 columns

In [61]:
```
my_mdr.fit(features, labels)
my_mdr.transform(features)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File ~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/indexes/base.py:3802, in Index.get_loc(self, key, method, toleranc
e)
   3801 try:
-> 3802     return self._engine.get_loc(casted_key)
   3803 except KeyError as err:

File ~/opt/anaconda3/lib/python3.9/site-packages/pandas/_libs/index.pyx:138, in pandas._libs.index.IndexEngine.get_loc()

File ~/opt/anaconda3/lib/python3.9/site-packages/pandas/_libs/index.pyx:165, in pandas._libs.index.IndexEngine.get_loc()

File pandas/_libs/hashtable_class_helper.pxi:5745, in pandas._libs.hashtable.PyObjectHashTable.get_item()

File pandas/_libs/hashtable_class_helper.pxi:5753, in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 0

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Input In [61], in <cell line: 1>()
----> 1 my_mdr.fit(features, labels)
      2 my_mdr.transform(features)

File ~/opt/anaconda3/lib/python3.9/site-packages/mdr/mdr.py:81, in MDRBase.fit(self, features, class_labels)
     79 self.class_count_matrix = defaultdict(lambda: defaultdict(int))
     80 for row_i in range(features.shape[0]):
---> 81     feature_instance = tuple(features[row_i])
     82     self.class_count_matrix[feature_instance][class_labels[row_i]] += 1
     83 self.class_count_matrix = dict(self.class_count_matrix)

File ~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/frame.py:3807, in DataFrame.__getitem__(self, key)
   3805 if self.columns.nlevels > 1:
   3806     return self._getitem_multilevel(key)
-> 3807 indexer = self.columns.get_loc(key)
   3808 if is_integer(indexer):
   3809     indexer = [indexer]

File ~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/indexes/base.py:3804, in Index.get_loc(self, key, method, toleranc
e)
   3802     return self._engine.get_loc(casted_key)
   3803 except KeyError as err:
-> 3804     raise KeyError(key) from err
   3805 except TypeError:
   3806     # If we have a listlike key, _check_indexing_error will raise
   3807     #  InvalidIndexError. Otherwise we fall through and re-raise
   3808     #  the TypeError.
```

```
   3809     self._check_indexing_error(key)

KeyError: 0
```

In [ ]: