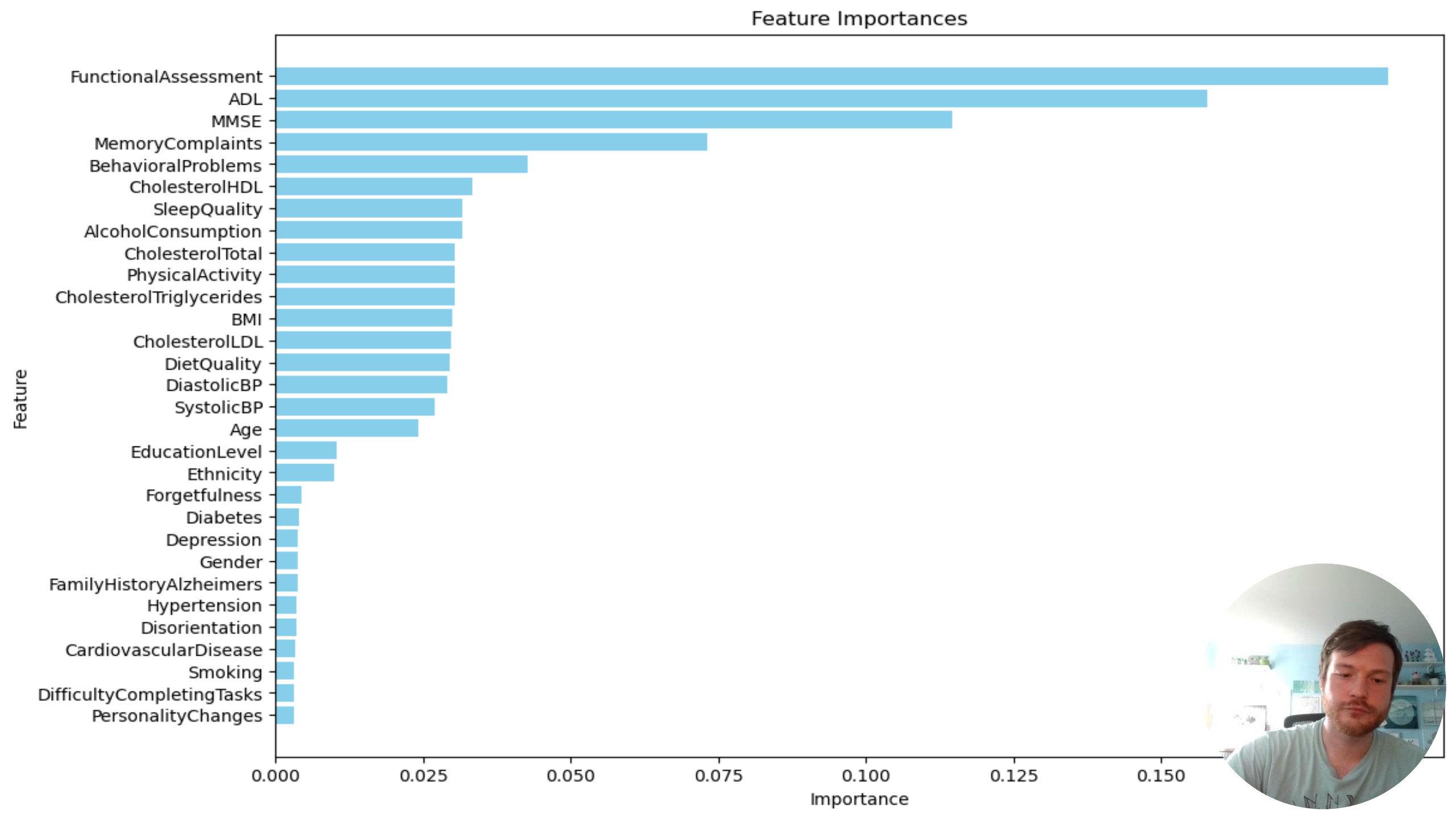
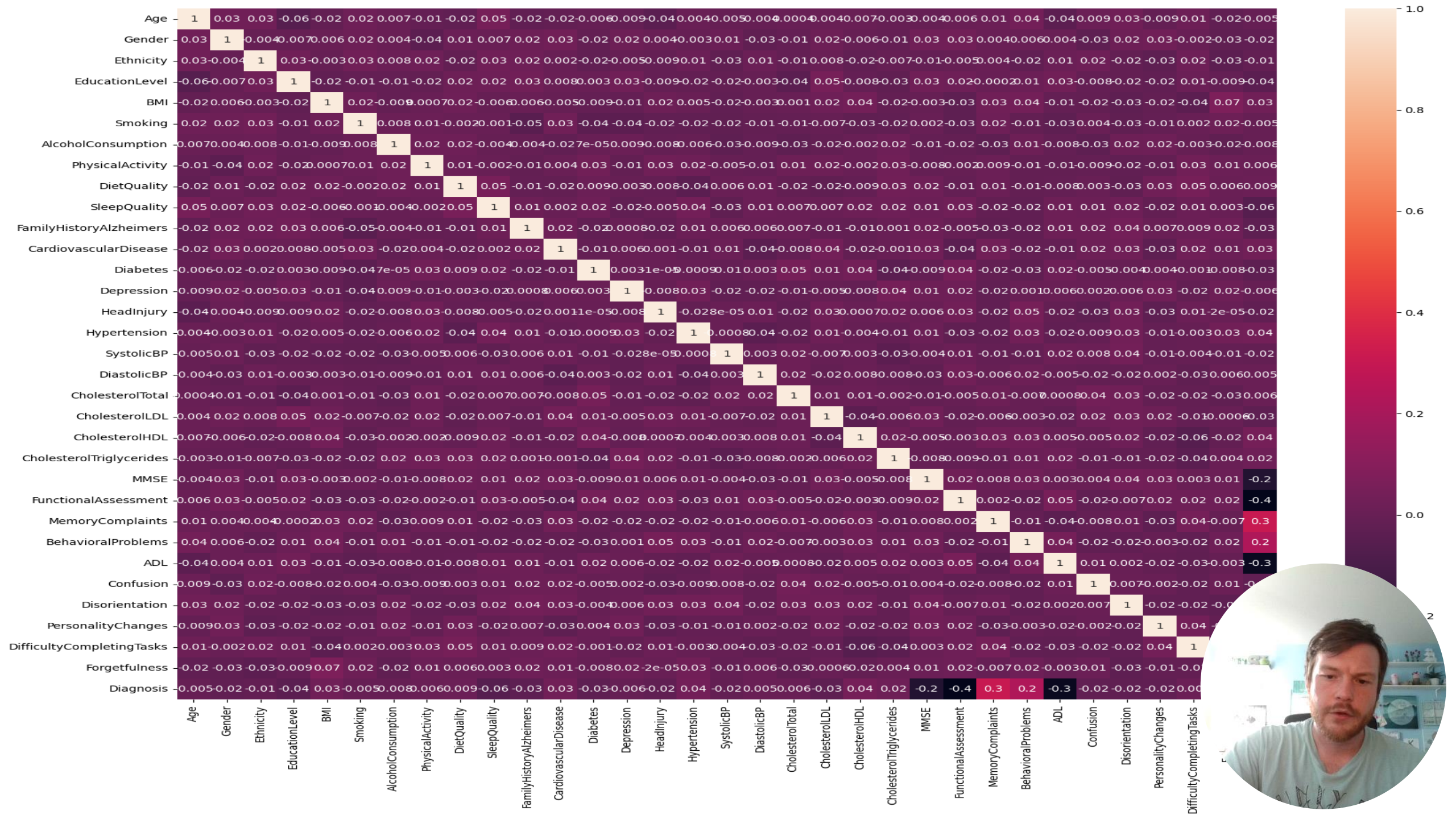
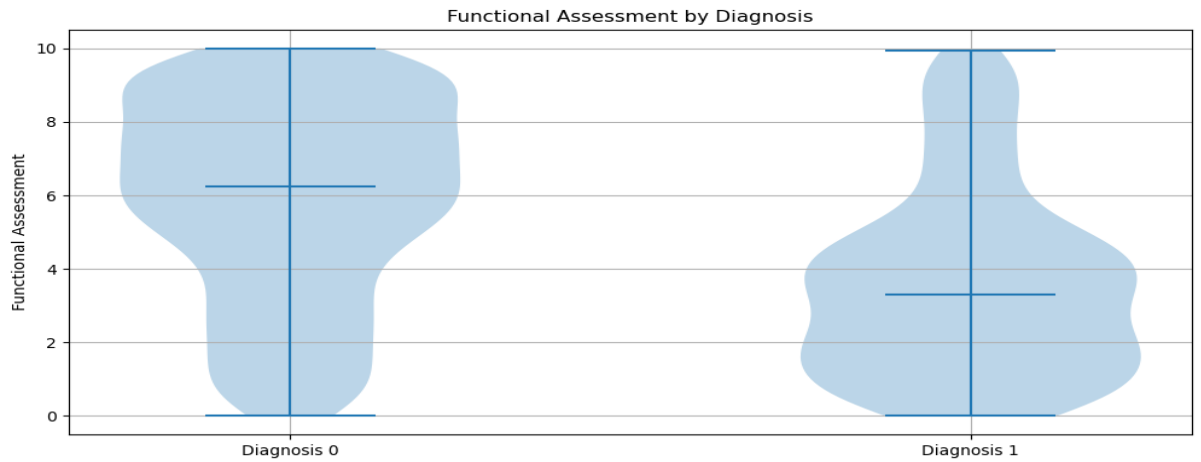
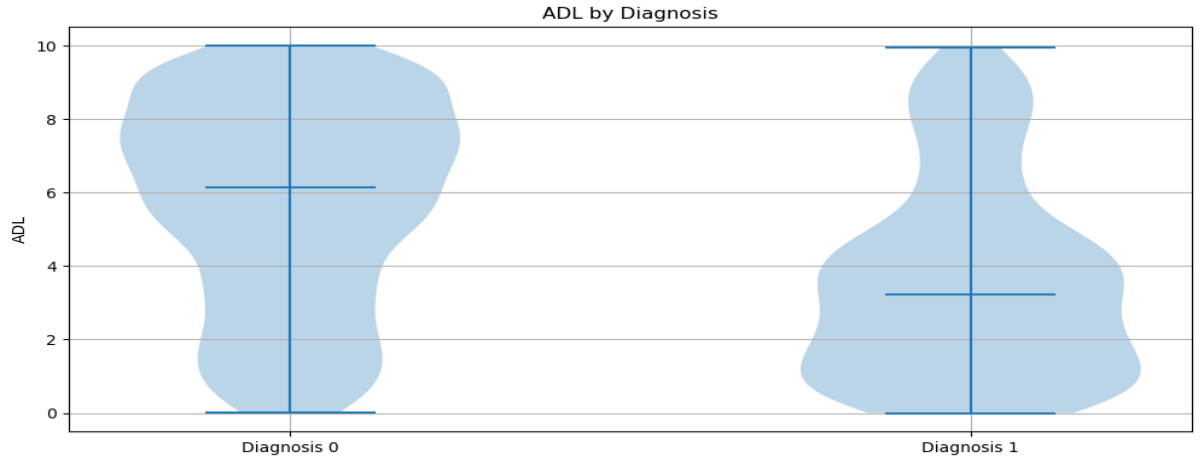
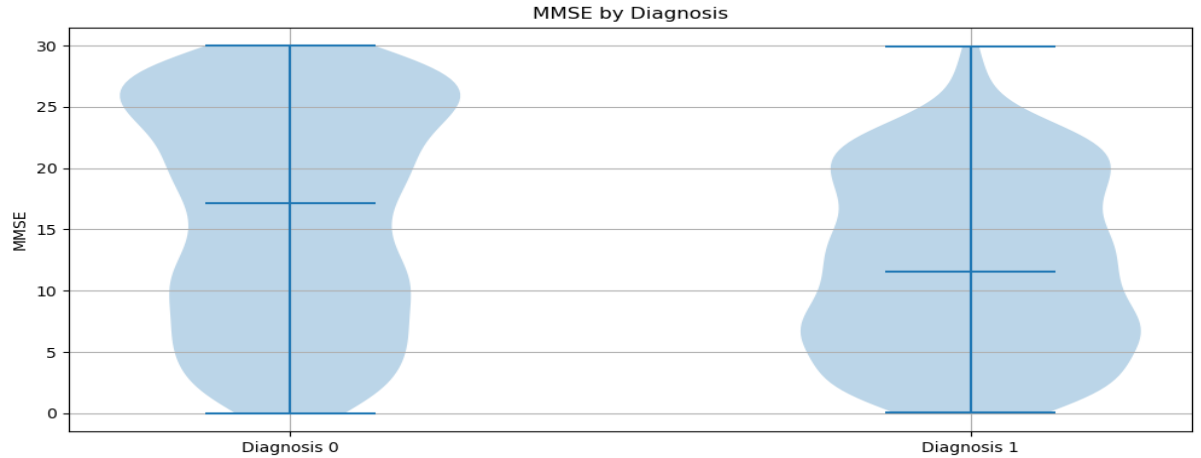
Alzheimer's Disease Analysis

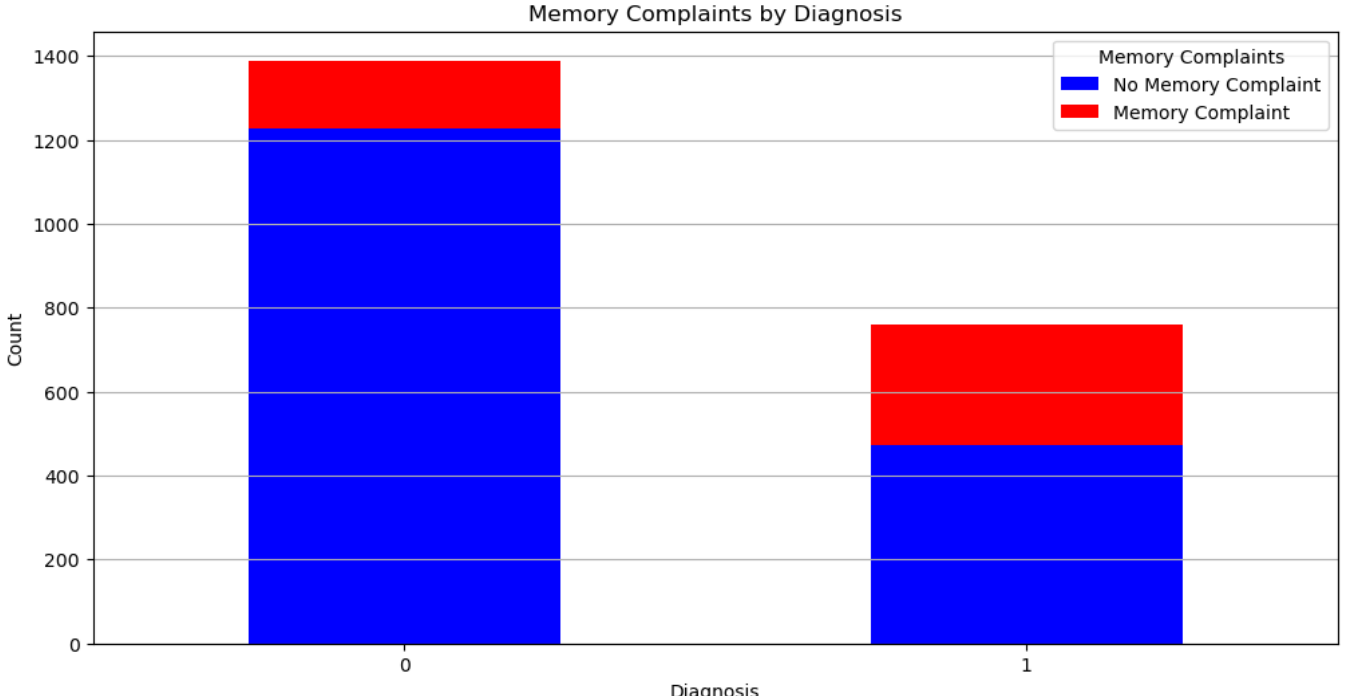

Feature Importances
FunctionalAssessment
ADL
MMSE
MemoryComplaints
BehavioralProblems
CholesterolHDL
SleepQuality
AlcoholConsumption
CholesterolTotal
PhysicalActivity
CholesterolTriglycerides
BMI
CholesterolLDL
DietQuality
DiastolicBP
SystolicBP
Age
EducationLevel
Ethnicity
Forgetfulness
Diabetes
Depression
Gender
FamilyHistoryAlzheimers
Hypertension
Disorientation
CardiovascularDisease
Smoking
DifficultyCompletingTasks
PersonalityChanges
Feature
0.000
0.025
0.050
0.075
0.100
0.125
0.150
Importance

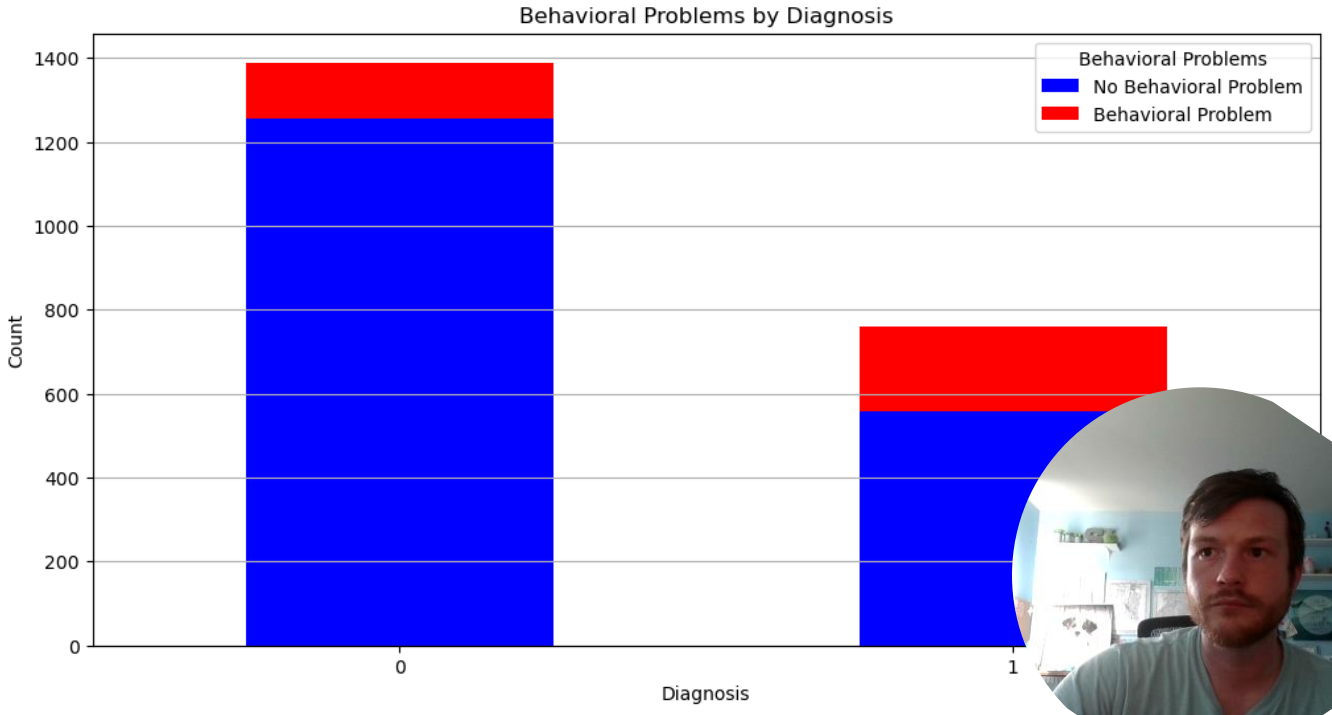

Age
Gender
Ethnicity
EducationLevel
BMI
Smoking
AlcoholConsumption
PhysicalActivity
DietQuality
SleepQuality
FamilyHistoryAlzheimers
CardiovascularDisease
Diabetes
Depression
HeadInjury
Hypertension
SystolicBP
DiastolicBP
CholesterolTotal
CholesterolLDL
CholesterolHDL
CholesterolTriglycerides
MMSE
FunctionalAssessment
MemoryComplaints
BehavioralProblems
ADL
Confusion
Disorientation
PersonalityChanges
DifficultyCompletingTasks
Forgetfulness
Diagnosis
1.0
0.8
0.6
0.4
0.2
0.0

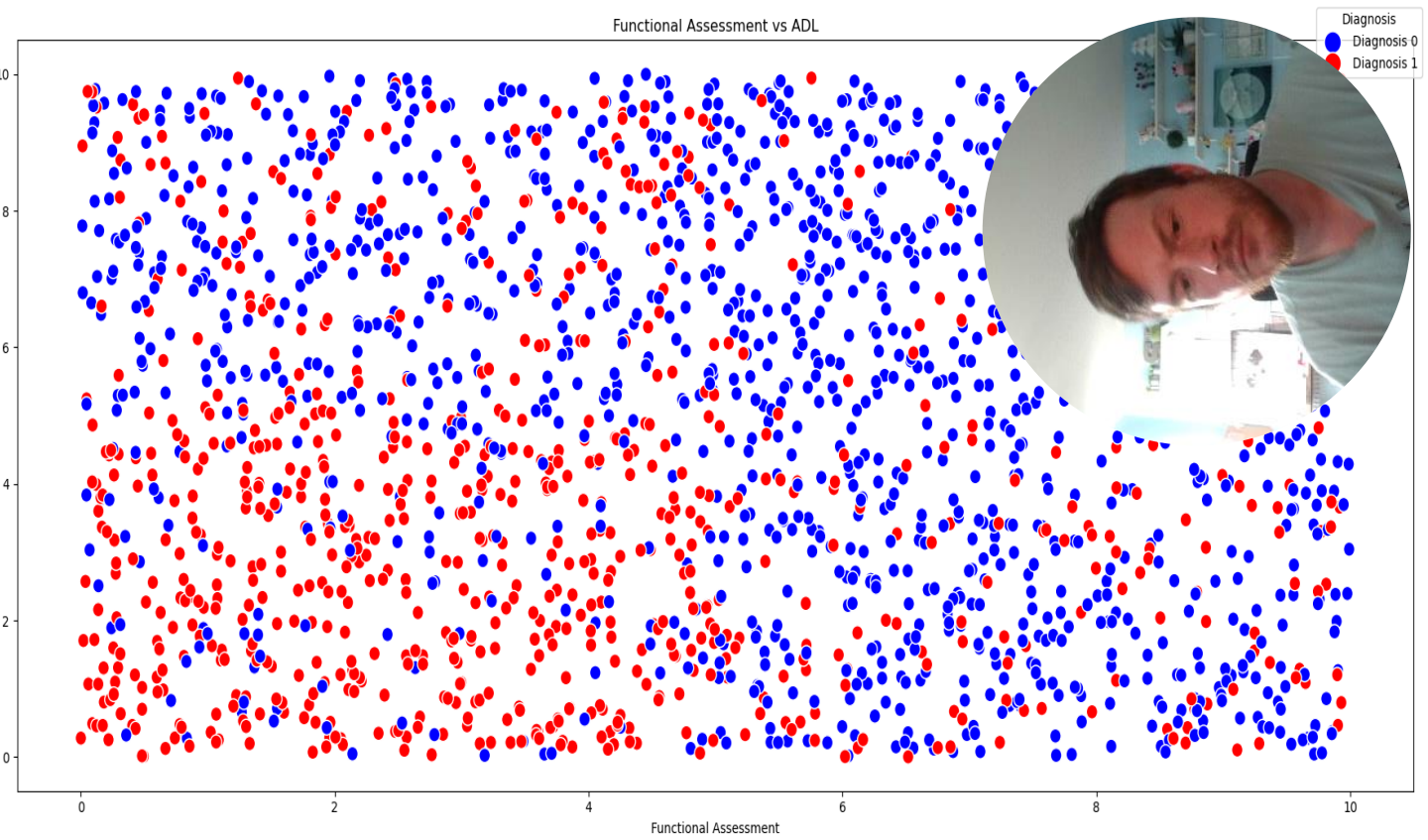

Functional Assessment by Diagnosis
Functional Assessment
Diagnosis 0
Diagnosis 1

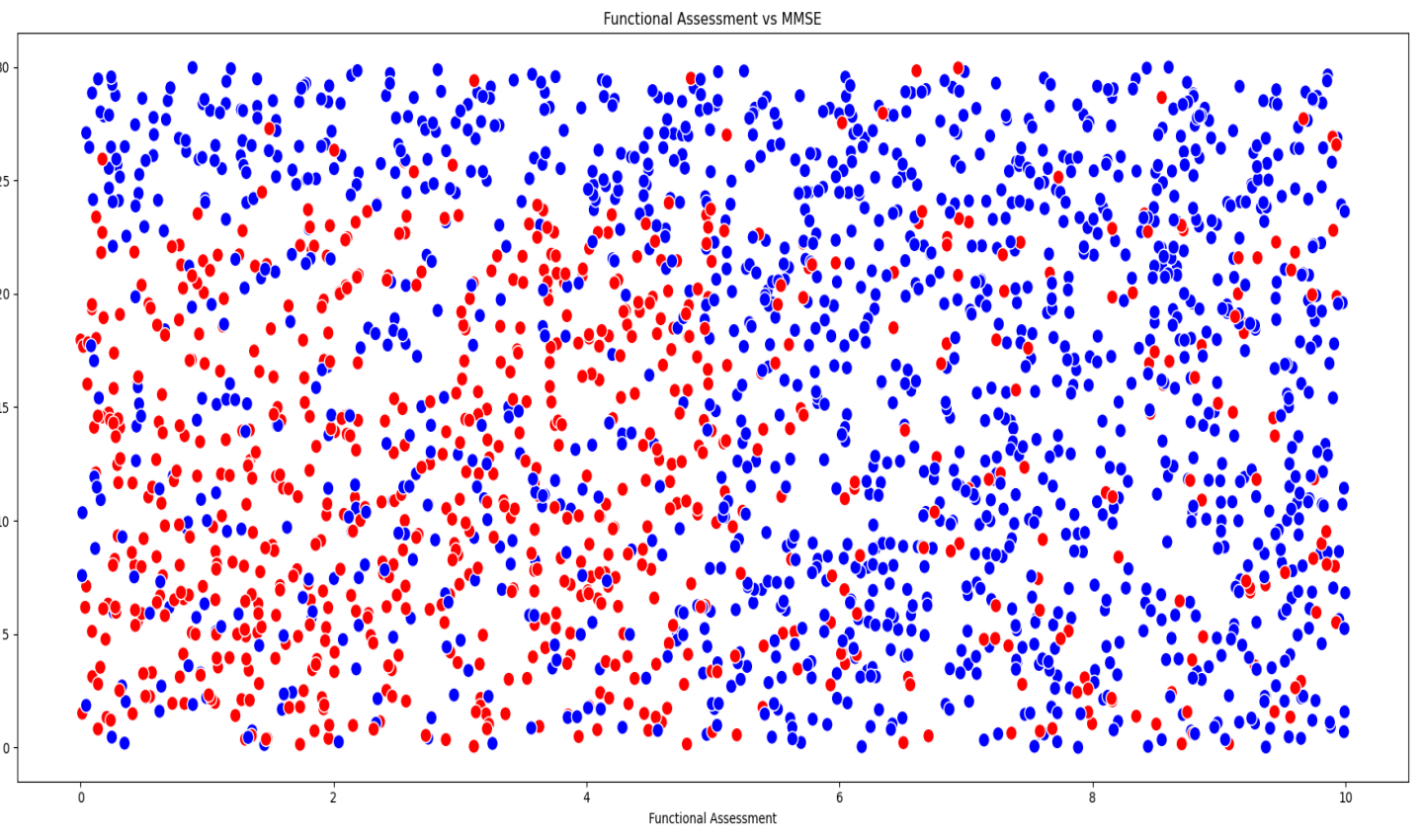

ADL by Diagnosis
ADL
Diagnosis 0
Diagnosis 1

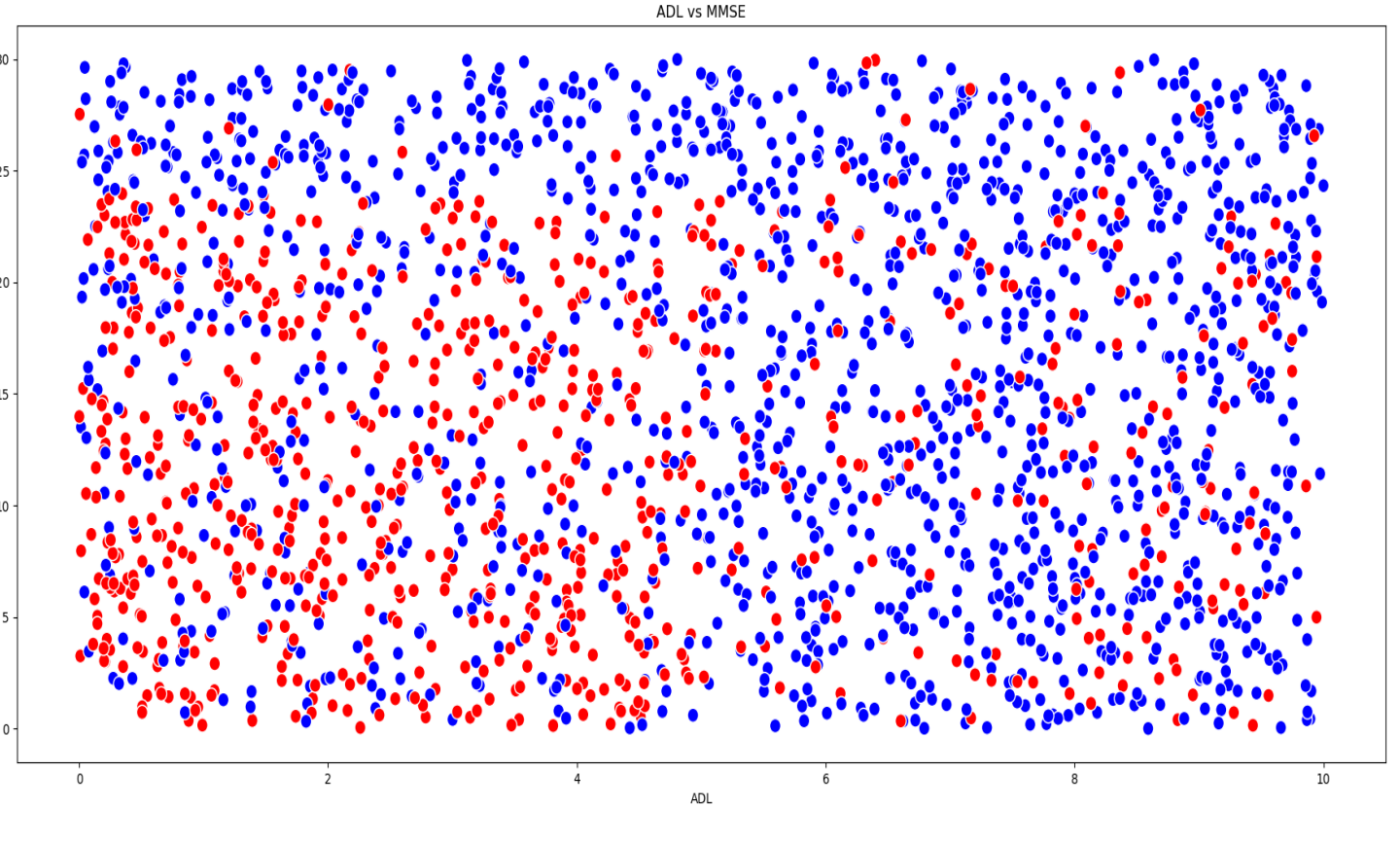

MMSE by Diagnosis
MMSE
Diagnosis 0
Diagnosis 1


Memory Complaints by Diagnosis
Memory Complaints
No Memory Complaint
Memory Complaint
Count
Diagnosis


Behavioral Problems by Diagnosis
Behavioral Problems
No Behavioral Problem
Behavioral Problem
Count
Diagnosis

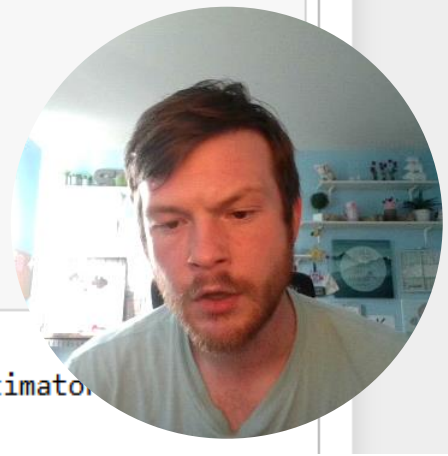

Functional Assessment vs ADL
Diagnosis
Diagnosis 0
Diagnosis 1
Functional Assessment


Functional Assessment vs MMSE
Functional Assessment


ADL vs MMSE
ADL

In [24]:
```
X_top = df_top.drop(columns=['Diagnosis'])
y_top = df_top['Diagnosis']
X_train_top, X_test_top, y_train_top, y_test_top = train_test_split(X_top, y_top, test_size=0.3, random_state=14)

param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# Initialize the RandomForestClassifier
rf = RandomForestClassifier(random_state=14)

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

# Fit the grid search to the data
grid_search.fit(X_train_top, y_train_top)

# Get the best parameters and best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print(f'Best Parameters: {best_params}')
print(f'Best Score: {best_score}')
y_pred = grid_search.best_estimator_.predict(X_test_top)
```

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
Best Parameters: {'bootstrap': True, 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimator
Best Score: 0.9567906976744185
```



Model Results

```
Accuracy: 0.9410852713178295
Precision: 0.9412472249517778
Recall: 0.9410852713178295
F1 Score: 0.9411561902850979
Confusion Matrix:
[[414  20]
 [ 18 193]]
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.95      0.96       434
           1       0.91      0.91      0.91       211

    accuracy                           0.94       645
   macro avg       0.93      0.93      0.93       645
weighted avg       0.94      0.94      0.94       645
```