# Welcome to the Gradient-Boosted Trees notebook!

This notebook was created at San Francisco State University for the PINC and gSTAR programs by Dr Pleuni Pennings, Lucy Moctezuma Tan and Lorena Benitez Rivera. We acknowledge help from Dr Adegoke Ojewole and Dr Hector Corrada Bravo from Genentech.

## Opening the file location and loading libraries

```
# Below we are importing necessary libraries
import pandas as pd
import numpy as np

# Importing packages for Creating ML model and Evaluating it
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
import xgboost as xgb
from sklearn import preprocessing
from sklearn import metrics
from sklearn.metrics import accuracy_score

# Importing library for plots
from matplotlib import pyplot as plt
from xgboost import plot_tree
from sklearn.metrics import ConfusionMatrixDisplay
import seaborn as sns
```

Read the dataset "PatData_cleaned.csv", this dataset is already cleaned, we have dropped all missing values and it should only contain the main diagnoses:

| Diagnosis | Meaning |
|---|---|
| NL | Cognitively normal |
| MCI | Mild Cognitive Impairment |
| Dementia | Person that has Alzheimer's Disease |

```
# Reading cleaned Dataset from Github
url = "https://raw.githubusercontent.com/pleunipennings/CSC508Data/main/PatData_cleaned.csv"
data = pd.read_csv(url)
data.head()
```

| | PTID | AGE | PTGENDER | PTEDUCAT | PTETHCAT | PTRACCAT | PTMARRY | APOE4 | DX | Ventricles | Hippocampus | WholeBrain | Entorhinal | Fusiform | MidTemp | ICV |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 002_S_0295 | 84.8 | 2 | 18 | 1 | 1 | 1 | 1.0 | NL | 43332.500000 | 6805.125000 | 1.071568e+06 | 3752.625000 | 17693.875000 | 19420.125000 | 1.649602e+06 |
| **1** | 002_S_0413 | 76.3 | 1 | 16 | 1 | 1 | 1 | 0.0 | NL | 31936.454545 | 6824.636364 | 1.055413e+06 | 4131.090909 | 20095.909091 | 20235.545455 | 1.600009e+06 |
| **2** | 002_S_0559 | 79.3 | 2 | 16 | 1 | 1 | 2 | 1.0 | NL | 38410.666667 | 7496.666667 | 1.092807e+06 | 3998.333333 | 18993.000000 | 22226.000000 | 1.703968e+06 |
| **3** | 002_S_0619 | 77.5 | 2 | 12 | 1 | 1 | 1 | 2.0 | Dementia | 120529.500000 | 5812.000000 | 1.093932e+06 | 2773.000000 | 20675.000000 | 19959.000000 | 2.070530e+06 |
| **4** | 002_S_0685 | 89.6 | 1 | 16 | 1 | 1 | 1 | 0.0 | NL | 40921.571429 | 7063.250000 | 9.800458e+05 | 3894.375000 | 14152.250000 | 18133.625000 | 1.521331e+06 |

後續步驟： 使用 data生成程式碼 查看建議的圖表 New interactive sheet

```
# checking counts for people with each type of diagnosis
data['DX'].value_counts()
```

| DX | count |
|---|---|
| **MCI** | 559 |
| **Dementia** | 522 |
| **NL** | 398 |

**dtype:** int64

## Preparing Training Data and Creating Gradient Boosted Tree Model Object

Split the data in labels (the diagnosis) and features (the other columns).
Every algorithm works a bit differently depending on how each package is written, which is why it is always important to be updated on changes of your more used packages. In this particular case we see that for the Gradient Boosted tree from sklearn package the labels need to be numeric.

```
# Separating labels from the general dataframe
labels = data["DX"]

# Creating a label encoder object
le = preprocessing.LabelEncoder()

# Fitting the label encoder into the labels columns
le.fit(data["DX"])

# Transforming the classes into numbers
labels_t = le.transform(data["DX"])
```

Below we can see our actual named Diagnosis, and then our transformed labels. As you can see now:

- Dementia = 0
- MCI = 1
- NL = 2

```
# Printing the classes we have
list(le.classes_)
```

```
['Dementia', 'MCI', 'NL']
```

```
# Printing how labels got transformed
np.unique(labels_t)
```

```
array([0, 1, 2])
```

Now for our features we will drop patient ID, because it does not help us make any predictions and we drop the diagnosis since that is our label. All other columns should be considered as predictor features.

```
# Dropping unnecessary columns for our features
features = data.drop(columns=['PTID','DX'])
```

The next part we should be pretty familiar with at this point:

- We will separate our training and testing datasets using the labels and features we have been stablished.
- We create the ML model object in this case it is Our Gradient Boosted Trees

**NOTE:** Notice that I have set ahead of time a couple of hyperparameters for my Gradient Boosted Tree already, such as a **seed** (for reproducible results), **eval_metric** (metric used to measure error, in this case is: "merror"), **max_depth** (each tree created will have a max of 4 layers deep before getting to a leaf), **learning_rate** (How fast we want our model to learn), **n_estimators** (Number of trees created by our model per class)

```
# As mentioned in the textbook, we use about 70-80% of our data as the training data and the rest as test da
features_train, features_test, labels_train, labels_test = train_test_split(features, labels_t, test_size=0.3, random_stat

#Create a Gradient Boosted Tree
gbt = XGBClassifier(seed = 42, eval_metric="merror", max_depth=4, learning_rate=0.5, n_estimators=50)
```

## Training using validation Data Gradient Boosted Tree

In order to illustrate how each tree created in the Gradient Boosted Tree improves, I will use a small validation dataset from the training data everytime our model makes a new tree.

- **Validation Data:** Must come from training data because, our test set is reserved for our final evaluation of the model. We use it in our example below so that you can see behind the scenes how much each iteration of trees get better results. In our case we will use 20 percent of our training data for our validation test.

**NOTE:** We should never use the test data during training because that would bias our model. This would be like knowing ahead of time the exact questions for an exam and then scoring high!

```
# Extracting validation data from training data
X_train, X_valid, Y_train, Y_valid = train_test_split(features_train, labels_train, test_size=0.2, random_state=42)

# Validation set
validation_set = [(X_train,Y_train),(X_valid, Y_valid)]

# Train a gradient Boosted Tree with validation data
gbt.fit(features_train, labels_train , eval_set = validation_set)
```

```
[0]	validation_0-merror:0.33696	validation_1-merror:0.27536
[1]	validation_0-merror:0.29710	validation_1-merror:0.25121
[2]	validation_0-merror:0.26691	validation_1-merror:0.23671
[3]	validation_0-merror:0.26570	validation_1-merror:0.25604
[4]	validation_0-merror:0.24275	validation_1-merror:0.22222
[5]	validation_0-merror:0.22585	validation_1-merror:0.21256
[6]	validation_0-merror:0.21256	validation_1-merror:0.20773
[7]	validation_0-merror:0.18478	validation_1-merror:0.17874
[8]	validation_0-merror:0.17271	validation_1-merror:0.16908
[9]	validation_0-merror:0.16908	validation_1-merror:0.16908
[10]	validation_0-merror:0.16063	validation_1-merror:0.15459
[11]	validation_0-merror:0.15459	validation_1-merror:0.13043
[12]	validation_0-merror:0.13647	validation_1-merror:0.12560
[13]	validation_0-merror:0.13043	validation_1-merror:0.13527
[14]	validation_0-merror:0.12077	validation_1-merror:0.11594
[15]	validation_0-merror:0.11232	validation_1-merror:0.11111
[16]	validation_0-merror:0.10145	validation_1-merror:0.10628
[17]	validation_0-merror:0.09541	validation_1-merror:0.09179
[18]	validation_0-merror:0.08213	validation_1-merror:0.10145
[19]	validation_0-merror:0.07126	validation_1-merror:0.10145
[20]	validation_0-merror:0.06643	validation_1-merror:0.09662
[21]	validation_0-merror:0.06039	validation_1-merror:0.09179
[22]	validation_0-merror:0.05556	validation_1-merror:0.08213
[23]	validation_0-merror:0.05072	validation_1-merror:0.07246
[24]	validation_0-merror:0.04710	validation_1-merror:0.07246
[25]	validation_0-merror:0.03744	validation_1-merror:0.05797
[26]	validation_0-merror:0.03502	validation_1-merror:0.04831
[27]	validation_0-merror:0.03019	validation_1-merror:0.03865
[28]	validation_0-merror:0.02657	validation_1-merror:0.03382
[29]	validation_0-merror:0.02415	validation_1-merror:0.03382
[30]	validation_0-merror:0.02053	validation_1-merror:0.02899
[31]	validation_0-merror:0.01812	validation_1-merror:0.02415
[32]	validation_0-merror:0.01570	validation_1-merror:0.01932
[33]	validation_0-merror:0.01691	validation_1-merror:0.01449
[34]	validation_0-merror:0.01449	validation_1-merror:0.01449
[35]	validation_0-merror:0.01208	validation_1-merror:0.00966
[36]	validation_0-merror:0.01087	validation_1-merror:0.00966
[37]	validation_0-merror:0.00966	validation_1-merror:0.00966
[38]	validation_0-merror:0.00966	validation_1-merror:0.00966
[39]	validation_0-merror:0.00604	validation_1-merror:0.00483
[40]	validation_0-merror:0.00604	validation_1-merror:0.00483
[41]	validation_0-merror:0.00483	validation_1-merror:0.00483
[42]	validation_0-merror:0.00483	validation_1-merror:0.00483
[43]	validation_0-merror:0.00362	validation_1-merror:0.00483
[44]	validation_0-merror:0.00362	validation_1-merror:0.00483
[45]	validation_0-merror:0.00362	validation_1-merror:0.00483
[46]	validation_0-merror:0.00362	validation_1-merror:0.00000
[47]	validation_0-merror:0.00362	validation_1-merror:0.00000
[48]	validation_0-merror:0.00242	validation_1-merror:0.00000
[49]	validation_0-merror:0.00242	validation_1-merror:0.00000
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='merror',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=0.5, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=4,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=50,
              n_jobs=None, num_parallel_tree=None, objective='multi:softprob', ...)
```

The output above shows us 2 columns, the first shows us how our model is doing in 80% percent of our training data, whereas the the other column shows what our validation set (20% of training data) is doing at each iteration. Looking at both columns at the same time we can get an

idea of whether we might be overfirring or not. Since we are using the **merror** (Multiclass classification Error Rate) as our metric, we want to see that both columns show a decrease in this metric. As you can see they both do.

**NOTE:** For Multiclass Classifications the Gradient Boosted Tree Model actually creates 50 trees for each class! Dementia, MCI and NL.

Below you can actually get a summary of all the trees that were created. There are a lot of trees so we wont print all of them considering there is (50 X 3 classes) 150 trees total!

```
# creating a list of all the trees created
gbt_treelist = gbt.get_booster().get_dump()
# Getting total amount of trees from XGboost Classifier model
print(len(gbt_treelist))
```

```
150
```

Here we can for example look at the first 2 trees created. The results below shows you the index for each node it creates for both of the trees. For example the **root node [index = 0] for the first tree is [Entorhinal<3239.82129]**

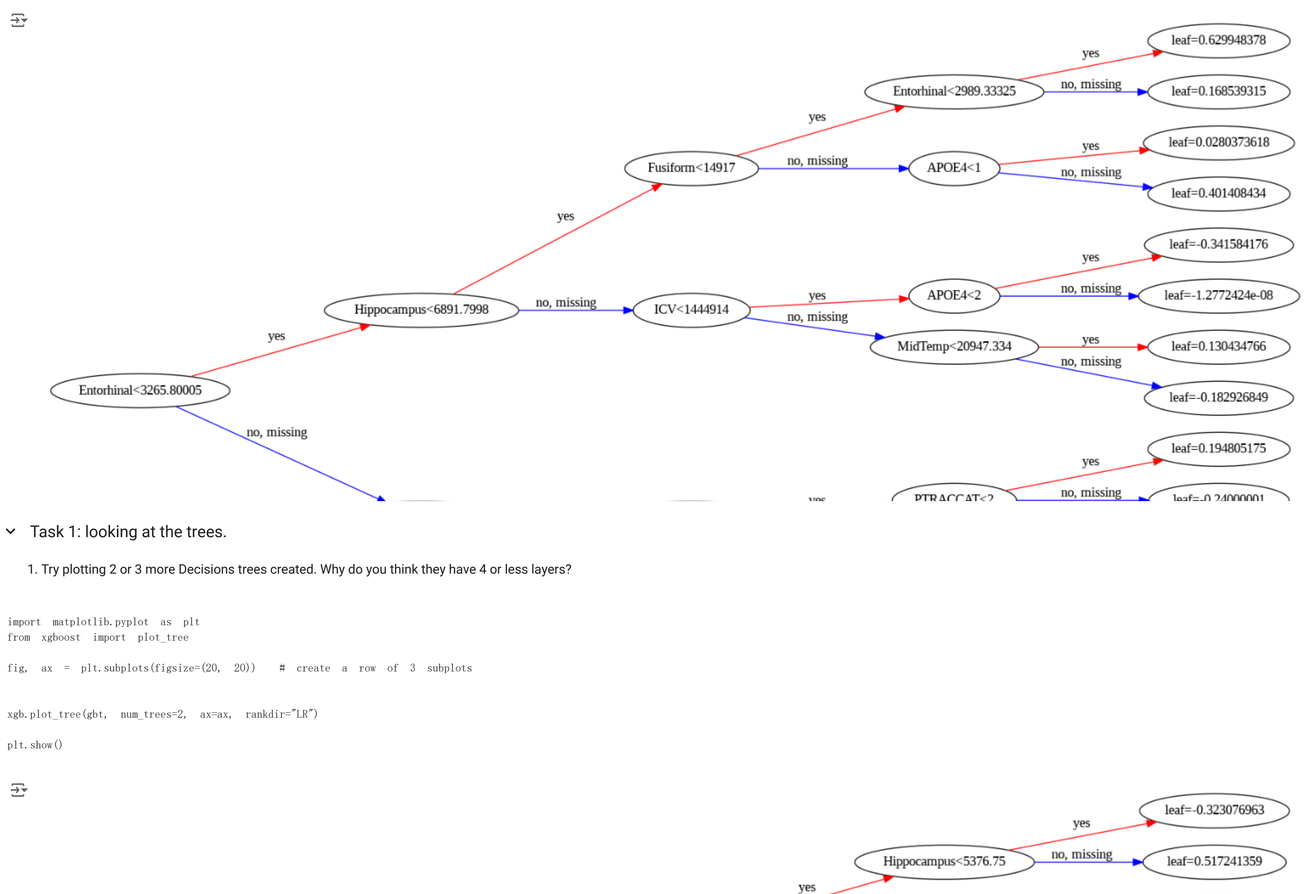
```
# Check the total amopunt of trees
for tree in gbt_treelist[0:2]:
    print(tree)
```

```
                                16:leaf=0.168539315
                        8:[APOE4<1] yes=17,no=18,missing=18
                                17:leaf=0.0280373618
                                18:leaf=0.401408434
                4:[ICV<1444914] yes=9,no=10,missing=10
                        9:[APOE4<2] yes=19,no=20,missing=20
                                19:leaf=0.341584176
                                20:leaf=-1.2772424e-08
                        10:[MidTemp<20947.334] yes=21,no=22,missing=22
                                21:leaf=0.130434766
                                22:leaf=-0.182926849
        2:[Ventricles<58069.75] yes=5,no=6,missing=6
                5:[Hippocampus<5920] yes=11,no=12,missing=12
                        11:[PTRACCAT<2] yes=23,no=24,missing=24
                                23:leaf=0.194805175
                                24:leaf=-0.24000001
                        12:[APOE4<2] yes=25,no=26,missing=26
                                25:leaf=-0.301121235
                                26:leaf=-0.049586799
                6:[ICV<1736485] yes=13,no=14,missing=14
                        13:[Hippocampus<5973] yes=27,no=28,missing=28
                                27:leaf=0.32432431
                                28:leaf=-0.156108618
                        14:[Entorhinal<4123.5] yes=29,no=30,missing=30
                                29:leaf=0.327272713
                                30:leaf=-0.200000018

0:[MidTemp<16193.7998] yes=1,no=2,missing=2
        1:[Entorhinal<3405.25] yes=3,no=4,missing=4
                3:[Fusiform<14893.25] yes=7,no=8,missing=8
                        7:[ICV<1205816] yes=15,no=16,missing=16
                                15:leaf=-1.62558127e-08
                                16:leaf=-0.335664362
                        8:[Fusiform<15071.5] yes=17,no=18,missing=18
                                17:leaf=0.428571403
                                18:leaf=-0.181528687
                4:[Ventricles<27104] yes=9,no=10,missing=10
                        9:[AGE<79.0999985] yes=19,no=20,missing=20
                                19:leaf=0.517241359
                                20:leaf=-1.2772424e-08
                        10:[WholeBrain<897133.5] yes=21,no=22,missing=22
                                21:leaf=-0.283783793
                                22:leaf=0.0365853496
        2:[AGE<64.9000015] yes=5,no=6,missing=6
                5:[Hippocampus<6553.3335] yes=11,no=12,missing=12
                        11:[AGE<62.2999992] yes=23,no=24,missing=24
                                23:leaf=-0.306122482
                                24:leaf=0.153061211
                        12:[PTRACCAT<3] yes=25,no=26,missing=26
                                25:leaf=0.534782588
                                26:leaf=-1.78813941e-08
                6:[Hippocampus<5482.3335] yes=13,no=14,missing=14
                        13:[APOE4<1] yes=27,no=28,missing=28
                                27:leaf=0.083999984
                                28:leaf=-0.220000029
                        14:[Hippocampus<7614.25] yes=29,no=30,missing=30
                                29:leaf=0.14622882
                                30:leaf=-0.018518541
```

## Visualizing one tree from our Gradient Boosted Trees Model

The text summary above looks so convoluted and ugly! Below we can visualize one of our trees from our Gradient Boosted Tree Model. Lets choose in this case to look at the very first tree. You can change the index of the tree by changing the argument **num_trees**. Notice that this tree is just laying on its side, the rootnode is on the left while the leaves are towards the right.

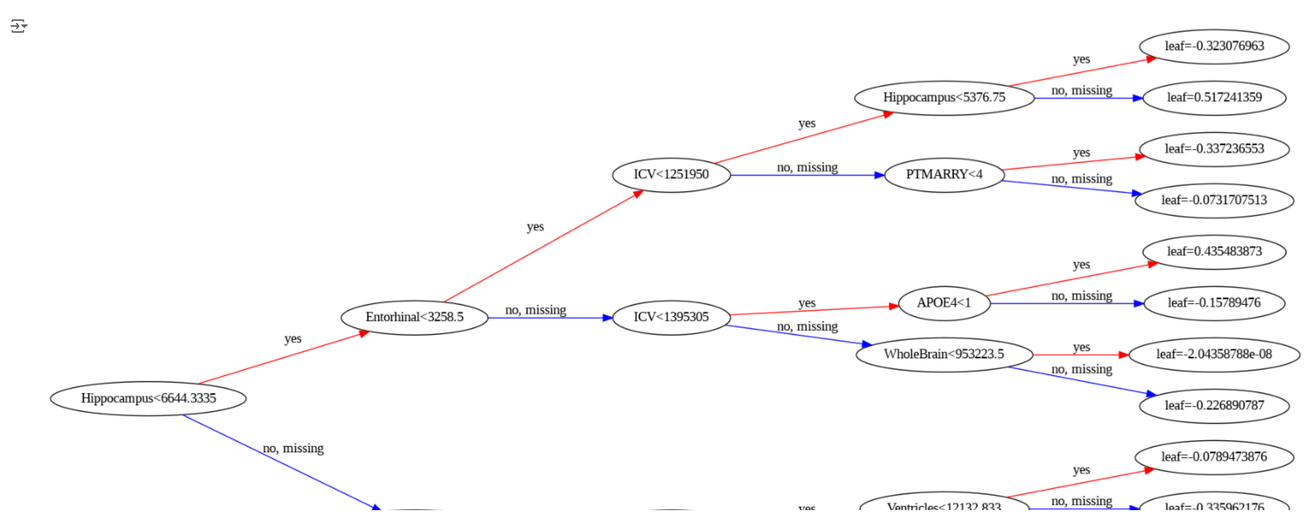
```
# Making a graph for our very first tree in Gradient Boosted Tree model
fig, ax = plt.subplots(figsize=(20, 20))
xgb.plot_tree(gbt, num_trees=0, ax=ax, rankdir="LR")
plt.show()
```




## Task 1: looking at the trees.

1. Try plotting 2 or 3 more Decisions trees created. Why do you think they have 4 or less layers?

```
import matplotlib.pyplot as plt
from xgboost import plot_tree

fig, ax = plt.subplots(figsize=(20, 20))  # create a row of 3 subplots


xgb.plot_tree(gbt, num_trees=2, ax=ax, rankdir="LR")

plt.show()
```




**Answer to task 1**

Recall however that one of the biggest difference between Random Forests and Gradient Boosted Trees is that not all trees have equal amount of say on the final decision. As each tree created in Gradient Boosted Tree model tries to take into account the errors from the previous one, the trees with the lowest errors should have more say than the ones with more errors.

## Evaluating our Gradient Boosted Tree

Below we will finally use our **TESTING DATA** to evaluate our model. Our Testing data has never been seen before by our model, so this evaluation would emulate how our model could perform once it is deployed. Below we will use our model to make predictions four out test data and the look at the first 10 predicted values

```
#Predict the response for test dataset
labels_pred = gbt.predict(features_test)

# Look at the predicted values.
print(labels_pred[:10])
#Compare with the real data from the test data set.
print(labels_test[:10])
```

```
[0 1 2 0 2 0 2 1 2 1]
[0 1 2 0 1 0 2 1 1 1]
```

The way a Gradient Boosted tree predicts each of the classes is through calculating different probabilities for each class. In the output below you can see that each row constitutes one prediction. Within each prediction we see 3 numbers. If you sum all the three number you get 100%. The index with the highest number is the final label predicted. Below is an example for the first predicted label:

| DX: | Dementia | Mild Cognitive Impairment | Normal |
|---|---|---|---|
| Index | 0 | 1 | 2 |
| Probabilities | 0.9004508 | 0.05835567 | 0.0411935 |

```
# getting the probabilities predicted for each class
preds_proba = gbt.predict_proba(features_test)
print(preds_proba)
```

```
[[0.946817   0.02875213 0.02443093]
 [0.12491538 0.46436474 0.41071984]
 [0.13575764 0.06168981 0.8025525 ]
 ...
 [0.00983566 0.59263474 0.3975296 ]
 [0.53526217 0.43542042 0.02931739]
 [0.4863515  0.3839223  0.12972619]]
```

## Task 2:

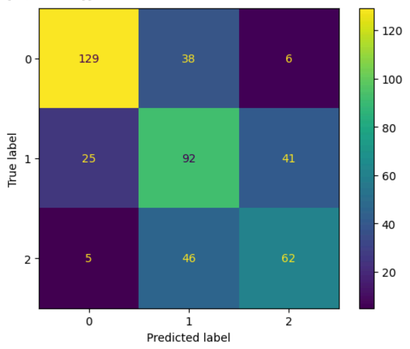
What are the 10 first diagnosis predicted by our Gradient Boosted Tree?

**Answer for Task 2** 1.Dementia 2.MCI 3.NL 4.Dementia 5.NL 6.MCI 7.NL 8.MCI 9.MCI 10.MCI

Below we will be plotting a Confusion matrix to Check to Check how our Gradient Boosted Tree has performed. We will also be calculating it's accuracy.

```
#Let's visualize how well the GBT does.
print(metrics.confusion_matrix(labels_test, labels_pred))
plt2 = metrics.ConfusionMatrixDisplay.from_estimator(gbt, features_test, labels_test)
plt.grid(False)
```

```
[[129  38   6]
 [ 25  92  41]
 [  5  46  62]]
```




```
# We want to check the accuracy in predicting the test data to make sure the model is not overfitted to the training data
accuracy = accuracy_score(labels_test, labels_pred)
print("Accuracy: %.1f%%" % (accuracy * 100))
```

```
Accuracy: 63.7%
```

## Task 3: Boosted trees

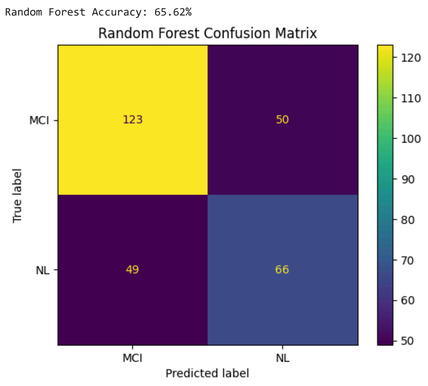
Now it's your turn to train a Gradient-boosted tree model and a Random Forest model, and see which one does better in terms of overall accuracy.

To make sure your model is a bit different from what we did previously in this notebook, I want you to choose just two of the diagnosis categories (NL, MCI, Dementia). With just two categories, the accuracy may become better than what we had before.

1. Create your smaller dataset with just two diagnosis categories.
2. Split label and features, split training and test.
3. Fit your models (gradient boosted trees and random forest).
4. Predict for your test data and calculate accuracies.
5. Plot your results in a confusion matrix.
6. Which of the model does better? Is it a big difference? Do you think that any hyperparameter tuning could improve them?

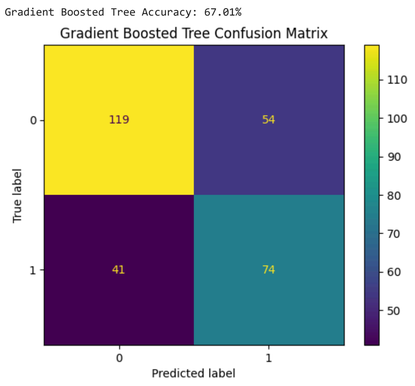
```
# Import necessary libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split

# Filter dataset for two categories (e.g., NL and MCI)
data_two_classes = data[data['DX'].isin(['NL', 'MCI'])]  # Change categories as needed
labels_two_classes = data_two_classes['DX']
features_two_classes = data_two_classes.drop(['PTID', 'DX'], axis=1)

# Split data
features_train, features_test, labels_train, labels_test = train_test_split(features_two_classes, labels_two_classes, test_size=0.3, random_state=42)

# Train Random Forest Model
rf = RandomForestClassifier(n_estimators=50, max_depth=4, random_state=42)
rf.fit(features_train, labels_train)

# Predict and evaluate
rf_pred = rf.predict(features_test)
rf_accuracy = accuracy_score(labels_test, rf_pred)
print(f"Random Forest Accuracy: {rf_accuracy * 100:.2f}%")

# Display confusion matrix
ConfusionMatrixDisplay.from_estimator(rf, features_test, labels_test)
plt.title("Random Forest Confusion Matrix")
plt.show()
```

```
Random Forest Accuracy: 65.62%
```




```
from sklearn.preprocessing import LabelEncoder

# Encode the labels to numeric values
le = LabelEncoder()
labels_train_encoded = le.fit_transform(labels_train)
labels_test_encoded = le.transform(labels_test)

# Train Gradient Boosted Tree Model with encoded labels
gbt = XGBClassifier(seed=42, eval_metric="error", max_depth=4, learning_rate=0.5, n_estimators=50)
gbt.fit(features_train, labels_train_encoded)

# Predict and evaluate using encoded labels
gbt_pred = gbt.predict(features_test)
gbt_accuracy = accuracy_score(labels_test_encoded, gbt_pred)
print(f"Gradient Boosted Tree Accuracy: {gbt_accuracy * 100:.2f}%")

# Display confusion matrix
ConfusionMatrixDisplay.from_predictions(labels_test_encoded, gbt_pred)
plt.title("Gradient Boosted Tree Confusion Matrix")
plt.show()
```

```
Gradient Boosted Tree Accuracy: 67.01%
```




**Answers for task 3**

Steps
1. Data Preparation
The dataset was filtered to focus on two diagnosis categories: NL and MCI, creating a binary classification problem. Labels (`DX`) were separated from the features, and columns like `PTID` and `DX` were removed to prevent bias from irrelevant data.
2. Data Splitting
The data was split into 70% training and 30% testing sets to evaluate the model on unseen data. `train_test_split` was used with `random_state` 42 for reproducibility.
3. Model Training and Evaluation
Random Forest:
Trained a Random Forest Classifier with 50 estimators and a max depth of 4.
Calculated accuracy on the test set and created a confusion matrix for performance evaluation.

Gradient Boosted Tree:
Applied label encoding for categorical data.

Trained `XGBClassifier` with `max_depth=4`, `learning_rate=0.5`, and `n_estimators=50` to balance underfitting and overfitting.

Made predictions on the test set and calculated accuracy, along with a confusion matrix for assessment.

4. Results
The accuracies of both models were calculated and printed.
Confusion matrices provided insights into true positive, false positive, true negative, and false negative counts, helping identify any bias towards a specific class.

5. Comparison and Analysis
The model with higher accuracy was deemed the better performer for this dataset.
Confusion matrix observations indicated which model misclassified categories more frequently.
Hyperparameter tuning potential was considered for improving performance.