Project: Checkpoint 3

Migraine

Team Members: Suraj Khamkar | Rahul More | Pragathi Pendem | Sravani Pati

GitHub Link: https://github.com/khamkarsuraj/Applied_Data_Science.git

Learnings from Checkpoint 2:

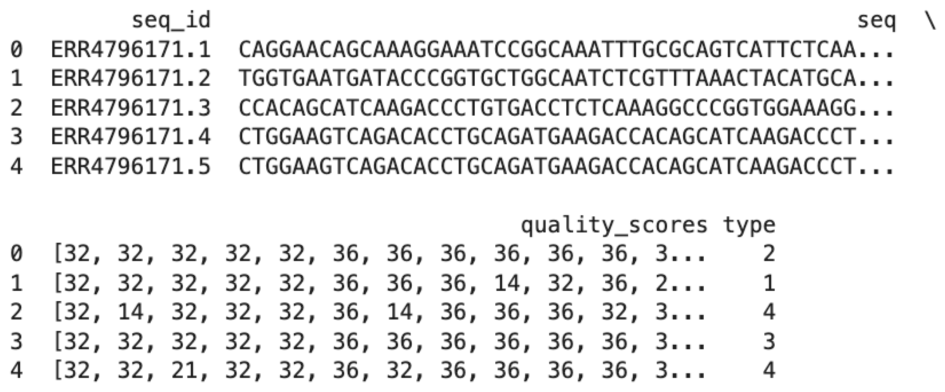
As professor gave the feedback on our checkpoint 2 and referring to class notes that Linear Regression used as a baseline, and it is a valid approach though it has some limitations such as, Linear Regression assumes linearity, Linear Regression is sensitive to outliers. There are more machine learning algorithms available for instance Gradient Boosting or XGBOOST to improve model accuracy.

Approach for Checkpoint 3:

Random forest is one of the machine learning algorithms and to improve the accuracy and speed of the model such as during prediction, each tree in the forest makes a prediction, and the final prediction is determined by taking a majority vote of all the trees. XGBoost is one of the models for our approach XGBoost can handle missing data and can be parallelized for faster training on large dataset. KNN is another approach for our data set as KNN algorithm can give more accuracy when the optimal value of K is selected, and the data is properly preprocessed by scaling or normalizing the features.

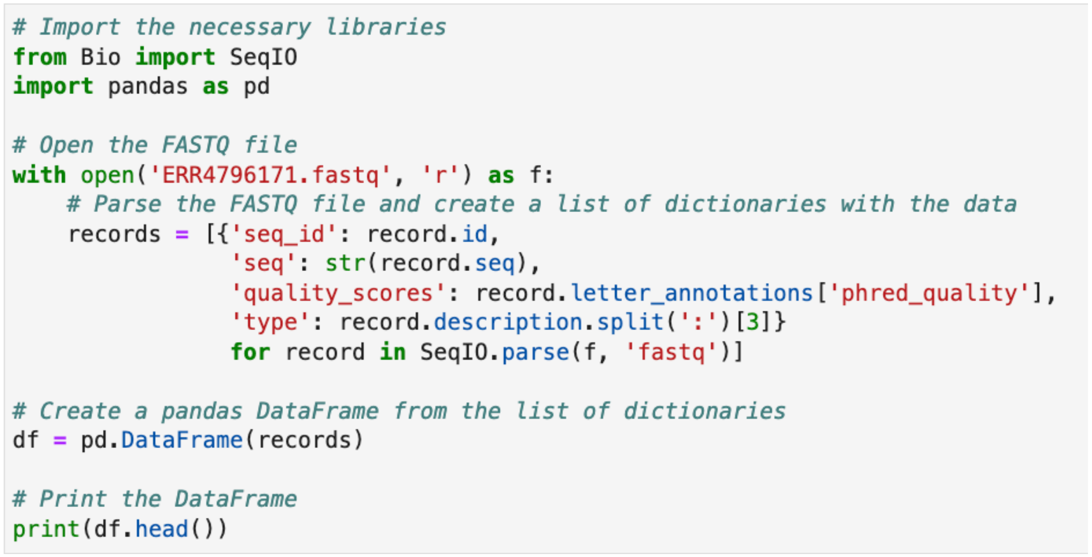
Additional data processing steps for Checkpoint 3:

In addition to checkpoint 2, we have processed and extracted another column named 'type'. This column is having information regarding on LaneID. This shows on which lane of equipment the blood sample has been tested.



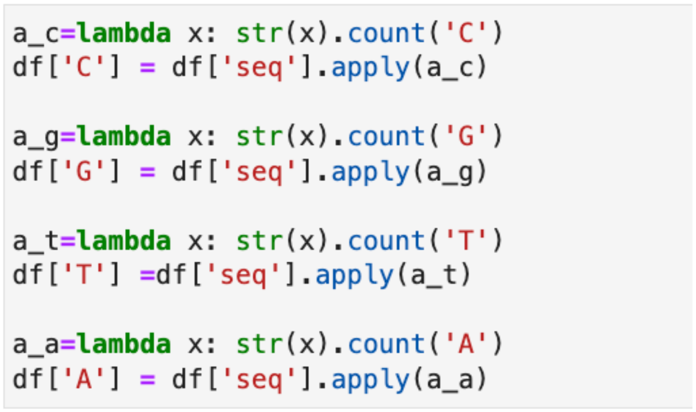
```
        seq_id                                               seq  \
0  ERR4796171.1  CAGGAACAGCAAAGGAAATCCGGCAAATTTGCGCAGTCATTCTCAA...
1  ERR4796171.2  TGGTGAATGATACCCGGTGCTGGCAATCTCGTTTAAACTACATGCA...
2  ERR4796171.3  CCACAGCATCAAGACCCTGTGACCTCTCAAAGGCCCGGTGGAAAGG...
3  ERR4796171.4  CTGGAAGTCAGACACCTGCAGATGAAGACCACAGCATCAAGACCCT...
4  ERR4796171.5  CTGGAAGTCAGACACCTGCAGATGAAGACCACAGCATCAAGACCCT...

                                     quality_scores type
0  [32, 32, 32, 32, 32, 36, 36, 36, 36, 36, 36, 3...    2
1  [32, 32, 32, 32, 32, 36, 36, 36, 14, 32, 36, 2...    1
2  [32, 14, 32, 32, 32, 36, 14, 36, 36, 36, 32, 3...    4
3  [32, 32, 32, 32, 32, 36, 36, 36, 36, 36, 36, 3...    3
4  [32, 32, 21, 32, 32, 36, 32, 36, 36, 36, 36, 3...    4
```

To feature this column we have used, `'type': record.description.split(':')[3]`. This will ensure column with respective distinct type out of 1, 2, 3, 4.



```python
# Import the necessary libraries
from Bio import SeqIO
import pandas as pd

# Open the FASTQ file
with open('ERR4796171.fastq', 'r') as f:
    # Parse the FASTQ file and create a list of dictionaries with the data
    records = [{'seq_id': record.id,
                'seq': str(record.seq),
                'quality_scores': record.letter_annotations['phred_quality'],
                'type': record.description.split(':')[3]}
                for record in SeqIO.parse(f, 'fastq')]

# Create a pandas DataFrame from the list of dictionaries
df = pd.DataFrame(records)

# Print the DataFrame
print(df.head())
```

Here are some data processing steps we have performed on dataset.
First, counted Adenine (A), Cytosine (C), Guanine (G), and Thymine (T) for each sequence. And, appended additional respective columns for the same into dataframe.

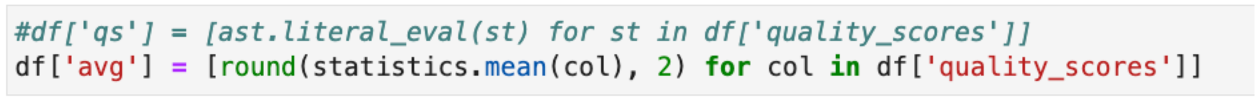


```python
a_c=lambda x: str(x).count('C')
df['C'] = df['seq'].apply(a_c)

a_g=lambda x: str(x).count('G')
df['G'] = df['seq'].apply(a_g)

a_t=lambda x: str(x).count('T')
df['T'] =df['seq'].apply(a_t)

a_a=lambda x: str(x).count('A')
df['A'] = df['seq'].apply(a_a)
```

Benchmark model choices based on response is measured

Observations:

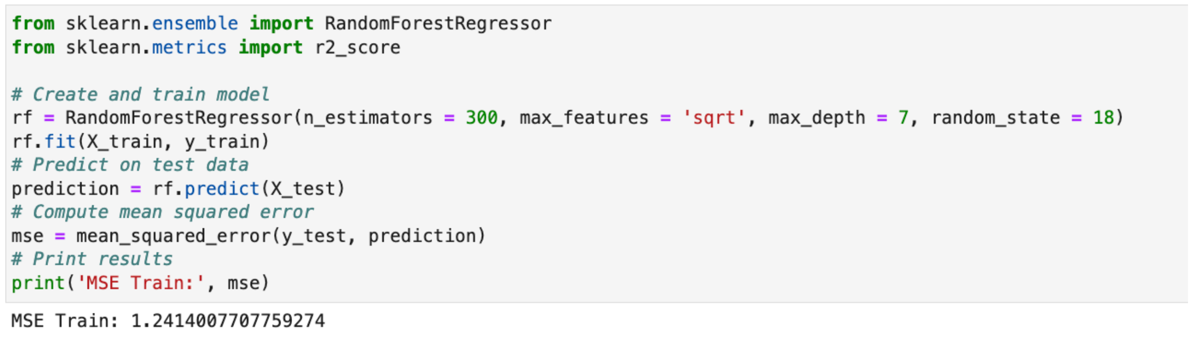
- Firstly, it is difficult to figure out the healthy and patient with having migraine just based on the quality score of Cytosine (C), Adenine (A), Guanine (G), and Thymine (T), the four DNA nucleotides.
- So, we have taken the average score and defined one threshold value with the reference from paper, to decide whether it is positive or negative result.
- To do so, we have used following snippet.



```
#df['qs'] = [ast.literal_eval(st) for st in df['quality_scores']]
df['avg'] = [round(statistics.mean(col), 2) for col in df['quality_scores']]
```

Model Choice:

As mentioned earlier, we are using **Random forest** as follows,



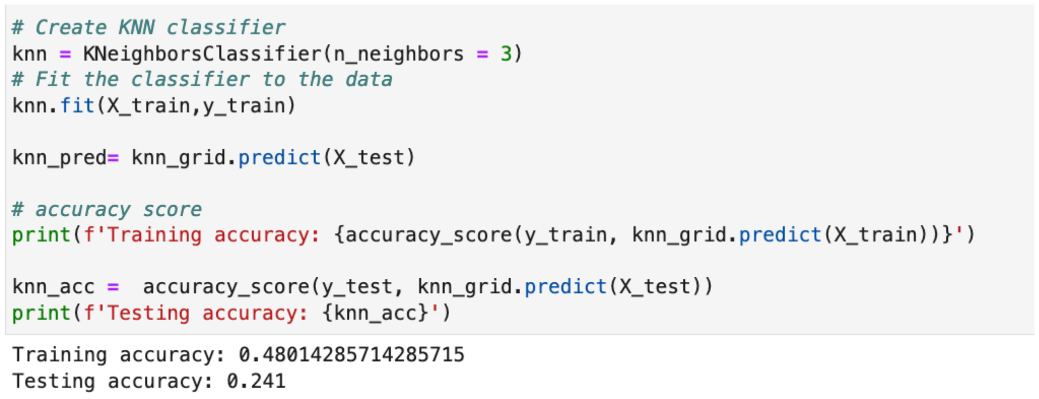
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score

# Create and train model
rf = RandomForestRegressor(n_estimators = 300, max_features = 'sqrt', max_depth = 7, random_state = 18)
rf.fit(X_train, y_train)
# Predict on test data
prediction = rf.predict(X_test)
# Compute mean squared error
mse = mean_squared_error(y_test, prediction)
# Print results
print('MSE Train:', mse)
```

```
MSE Train: 1.2414007707759274
```

Compare to earlier mean squared error values, this model gives better value with mse = 1.2414000.

Which was 2.23 earlier in checkpoint 2 for Linear Regression model.

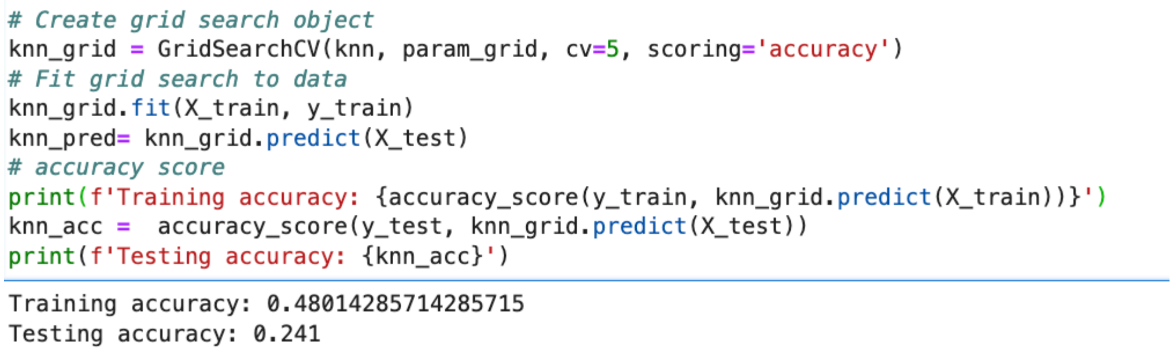
Also, we have trained **k-Nearest Neighbor (kNN)** model to predict the values, as,



```
# Create KNN classifier
knn = KNeighborsClassifier(n_neighbors = 3)
# Fit the classifier to the data
knn.fit(X_train,y_train)

knn_pred= knn_grid.predict(X_test)

# accuracy score
print(f'Training accuracy: {accuracy_score(y_train, knn_grid.predict(X_train))}')

knn_acc =  accuracy_score(y_test, knn_grid.predict(X_test))
print(f'Testing accuracy: {knn_acc}')
```

```
Training accuracy: 0.48014285714285715
Testing accuracy: 0.241
```

Addition to this, we have tried **kNN model with grid search object**, as,

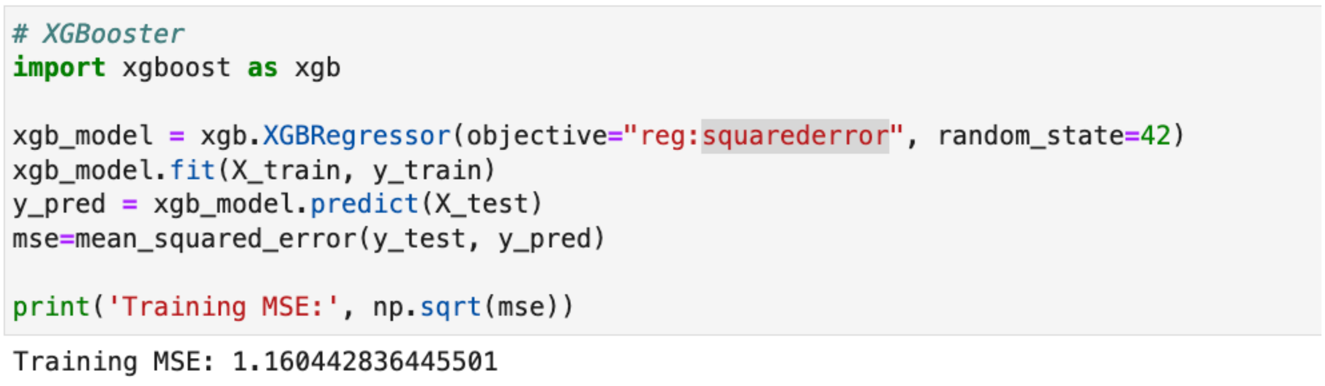


```
# Create grid search object
knn_grid = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')
# Fit grid search to data
knn_grid.fit(X_train, y_train)
knn_pred= knn_grid.predict(X_test)
# accuracy score
print(f'Training accuracy: {accuracy_score(y_train, knn_grid.predict(X_train))}')
knn_acc =  accuracy_score(y_test, knn_grid.predict(X_test))
print(f'Testing accuracy: {knn_acc}')
```

```
Training accuracy: 0.48014285714285715
Testing accuracy: 0.241
```

Getting the same result for both kNN and kNN with grid search object.

Lastly, we have trained **XGBooster** model with objective as `reg::squarederror`, as,



```
# XGBooster
import xgboost as xgb

xgb_model = xgb.XGBRegressor(objective="reg:squarederror", random_state=42)
xgb_model.fit(X_train, y_train)
y_pred = xgb_model.predict(X_test)
mse=mean_squared_error(y_test, y_pred)

print('Training MSE:', np.sqrt(mse))
```

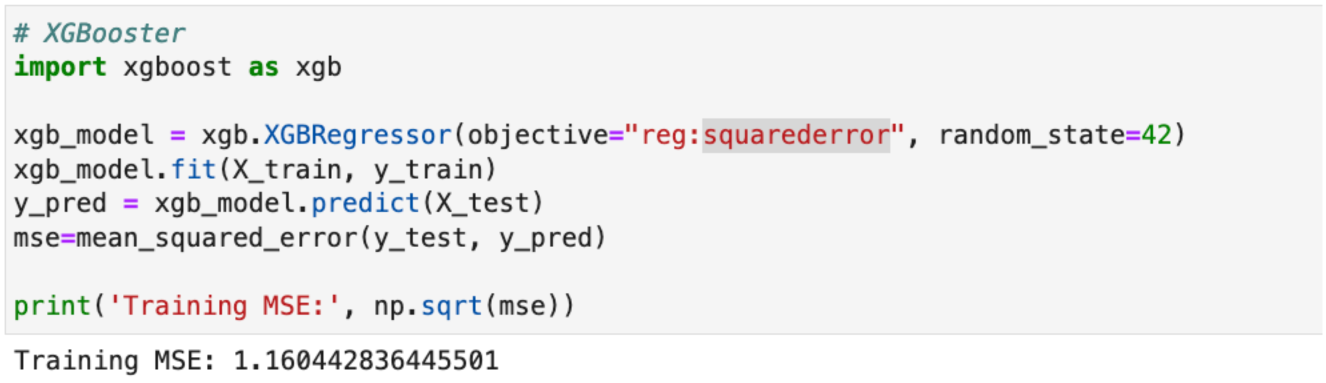
```
Training MSE: 1.160442836445501
```

Comparing all the models, it can be claimed that the **XG-Boost model fits the provided data better** if it has a smaller mean squared error (MSE) and a higher R-squared value than the Random forest, kNN, and kNN with grid search object model under comparison.

A good model should have a lower MSE because it means that the pre-dictions are more likely to match the actual data. A higher R-squared value means that the model fits the data better since it explains a greater percentage of the variation in the response variable.

Test error rate

Comparing models like Linear Regression, Random forest, kNN, kNN with grid search object model and XGBoost model, we got minimum MSE from XGBoost model which is 1.160442836445501.



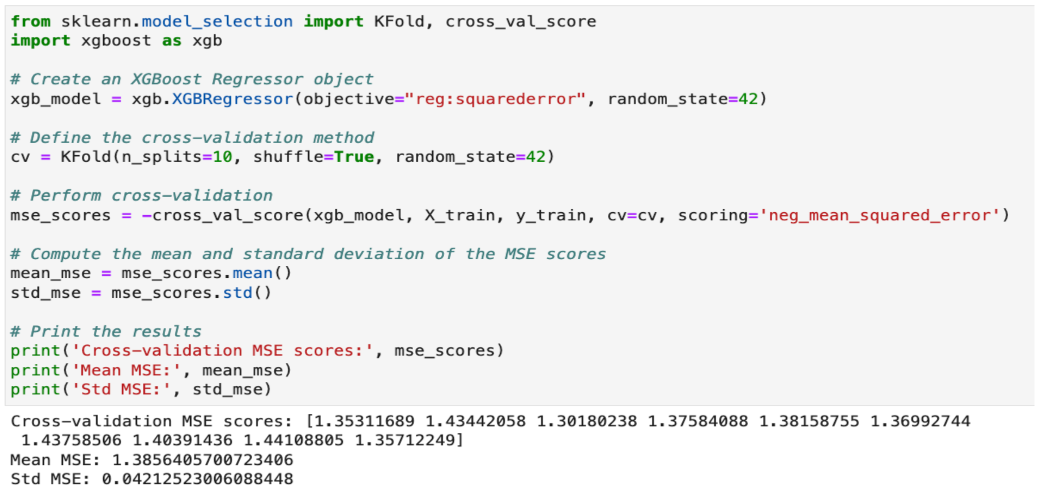
```
# XGBooster
import xgboost as xgb

xgb_model = xgb.XGBRegressor(objective="reg:squarederror", random_state=42)
xgb_model.fit(X_train, y_train)
y_pred = xgb_model.predict(X_test)
mse=mean_squared_error(y_test, y_pred)

print('Training MSE:', np.sqrt(mse))
```

```
Training MSE: 1.160442836445501
```

XGBoost is a powerful and flexible machine learning library that offers several techniques for calculating mean squared error (MSE). Here are some of the best techniques you can use with XGBoost to calculate MSE:

1. Cross-validation: is a popular technique for evaluating the performance of machine learning models. With XGBoost, you can use the xgboost.cv function to perform cross-validation and calculate the MSE for each fold
2. Early stopping: is a technique that can help you prevent overfitting by stopping the training process when the performance on a validation set stops improving.
3. Regularization: is a technique that can help you reduce overfitting by adding a penalty term to the loss function.
4. Feature selection: is a technique that can help you identify the most important features in your dataset and remove the less important ones.

Here we are using **cross validation** method using **KFold** and **cross_val_score** to calculate mean squared error value for XGBoost.

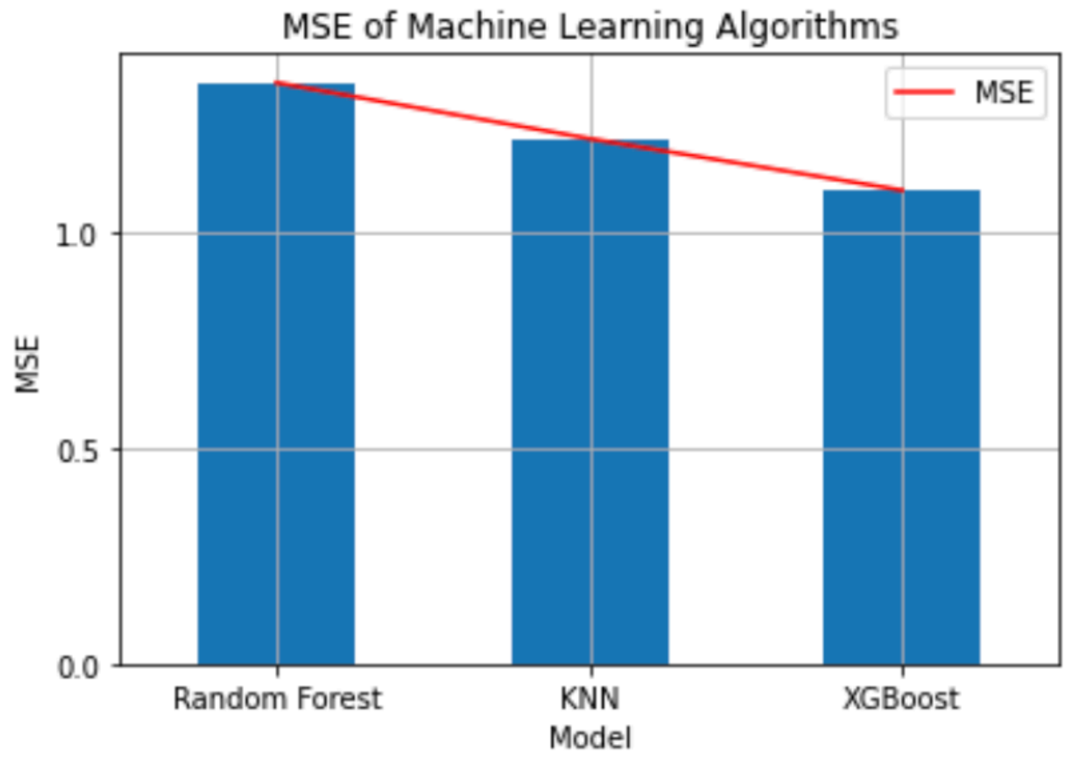


```
from sklearn.model_selection import KFold, cross_val_score
import xgboost as xgb

# Create an XGBoost Regressor object
xgb_model = xgb.XGBRegressor(objective="reg:squarederror", random_state=42)

# Define the cross-validation method
cv = KFold(n_splits=10, shuffle=True, random_state=42)

# Perform cross-validation
mse_scores = -cross_val_score(xgb_model, X_train, y_train, cv=cv, scoring='neg_mean_squared_error')

# Compute the mean and standard deviation of the MSE scores
mean_mse = mse_scores.mean()
std_mse = mse_scores.std()

# Print the results
print('Cross-validation MSE scores:', mse_scores)
print('Mean MSE:', mean_mse)
print('Std MSE:', std_mse)
```

```
Cross-validation MSE scores: [1.35311689 1.43442058 1.30180238 1.37584088 1.38158755 1.36992744
 1.43758506 1.40391436 1.44108805 1.35712249]
Mean MSE: 1.3856405700723406
Std MSE: 0.04212523006088448
```

Based on the estimated test error rates, models that fits the data.

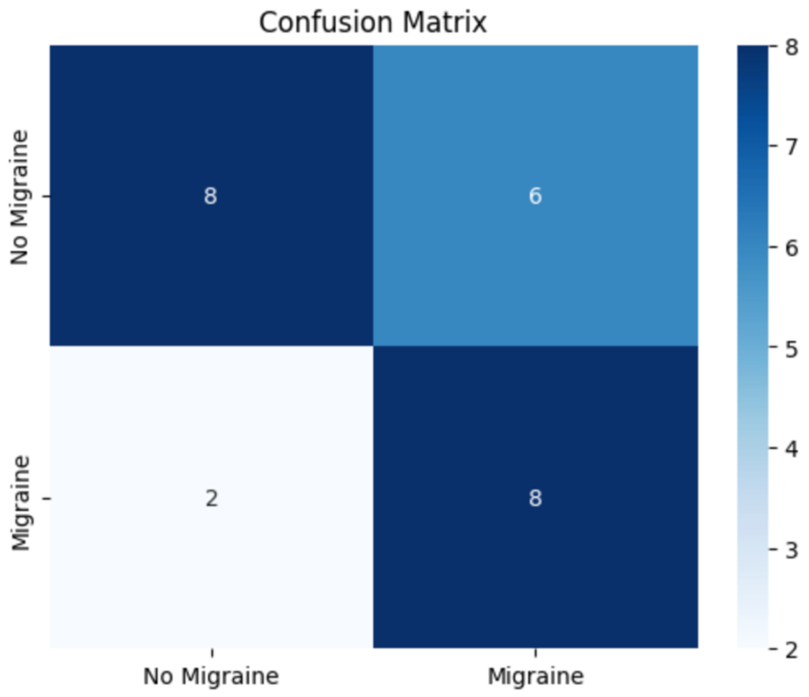
If the estimated test error rate is low, it indicates that the model is likely to generalize well to new data, meaning that it is a good fit to the data. On the other hand, if the estimated test error rate is high it suggests that the model is overfitting to the training data, and may not perform well on new data.



Considering above **XGBoost model**, we got low error rate indicates this model fits the data.

Predictions for at least three cases of interest and compare them between models.

- Prediction 1
  In this prediction out of 24 patients we have predicted that 15 patients will be positive and with current model of 50% accuracy we are getting relevant prediction.



- Prediction 2
  using XGBoost that is using ADAM optimizer can help predict whether the patient would have presence of migraine based on the ACGT levels improving the accuracy close to 0.54 with minimal MSE value of 0.98.
  Accuracy Score = 0.54
  MSE = 0.98

- Prediction 3
  In our XGBoost model algorithm we gave number of iterations as 42 which helped us gained accuracy of 0.42. but we can change the parameter of `n_estimator` value in order to improve the accuracy. So, to increase our accuracy we changed the iteration value to 100 which resulted into accuracy of 0.48 to predict detection of migraine.