UNIVERSITI TUNKU ABDUL RAHMAN

LEE KONG CHIAN FACULTY OF ENGINEERING AND SCIENCE

UECS3483 DATA MINING

ASSIGNMENT 2&3: MODELS, EVALUATION & DEPLOYMENT

LECTURER'S NAME: MR. SU LEE SENG

Name	Student ID	Programme
1. Chin Yuan Xi	2005686	SE
2. Chong Chun Wei	2204537	SE
3. Darren Kong Yan Ren	2207505	SE

Google Collab: https://colab.research.google.com/drive/1rSenBj7-CAg7WkTAN-YFFT4yNhByXk6L?usp=sharing

Imbalance Class Processing

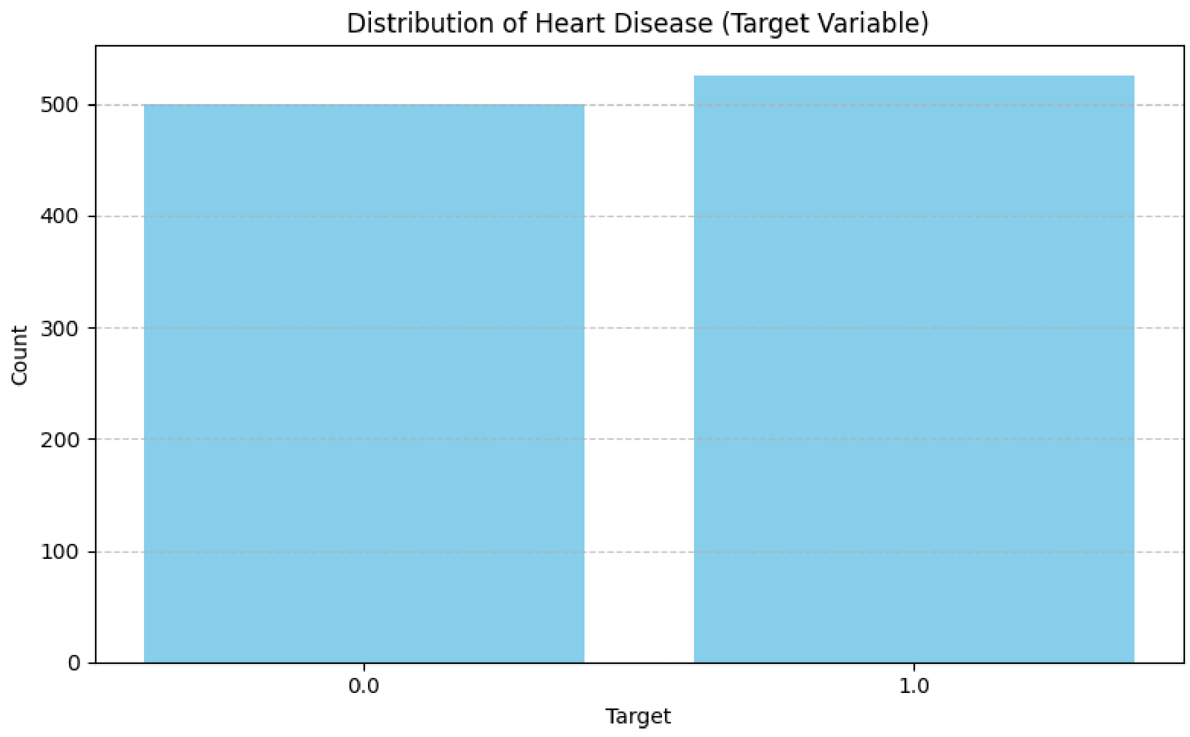
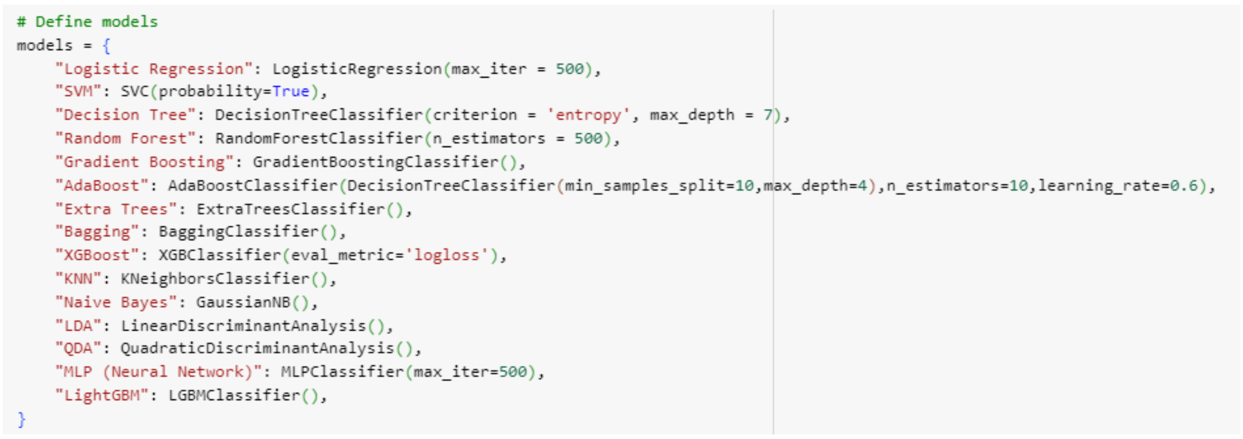


Figure 1: Column Chart for Independent Variable (target)

The first step was to inspect our target variable to see if there was any class imbalance, a common issue in healthcare datasets. If one class significantly dominates the other, it can mislead the model. Fortunately, our dataset showed that the target was almost balanced. This means both classes had a similar number of records, so we didn't need to apply resampling techniques like SMOTE or under sampling.

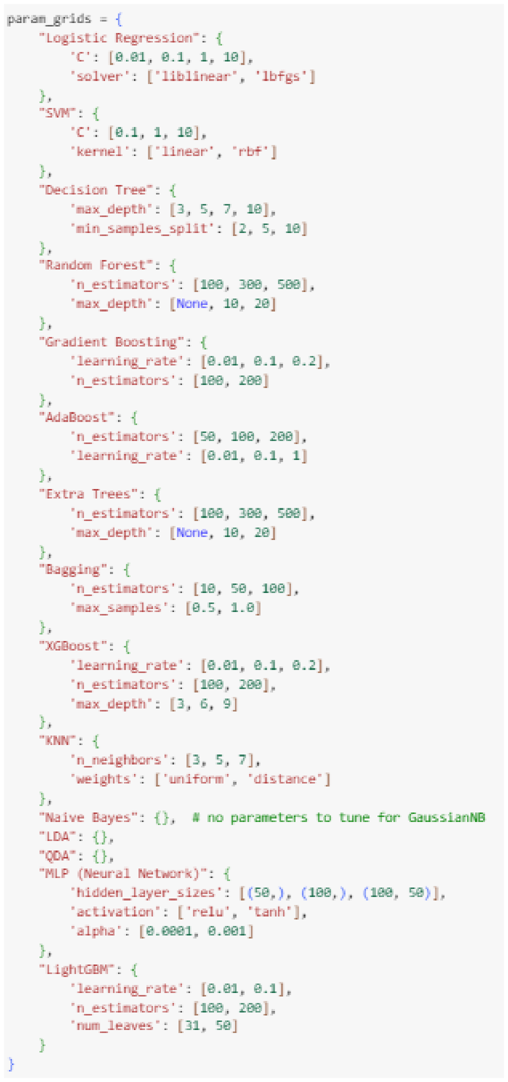
Algorithm Selection



```
# Define models
models = {
    "Logistic Regression": LogisticRegression(max_iter = 500),
    "SVM": SVC(probability=True),
    "Decision Tree": DecisionTreeClassifier(criterion = 'entropy', max_depth = 7),
    "Random Forest": RandomForestClassifier(n_estimators = 500),
    "Gradient Boosting": GradientBoostingClassifier(),
    "AdaBoost": AdaBoostClassifier(DecisionTreeClassifier(min_samples_split=10,max_depth=4),n_estimators=10,learning_rate=0.6),
    "Extra Trees": ExtraTreesClassifier(),
    "Bagging": BaggingClassifier(),
    "XGBoost": XGBClassifier(eval_metric='logloss'),
    "KNN": KNeighborsClassifier(),
    "Naive Bayes": GaussianNB(),
    "LDA": LinearDiscriminantAnalysis(),
    "QDA": QuadraticDiscriminantAnalysis(),
    "MLP (Neural Network)": MLPClassifier(max_iter=500),
    "LightGBM": LGBMClassifier(),
}
```

This code defines a dictionary called models that contains a list of machine learning classification models from various families, using scikit-learn and other libraries. Each key in the dictionary is a string representing the name of the model, while the value is the instantiated model object with specific parameters (if any). The models included are: Logistic Regression, Support Vector Classifier (SVM) with probability estimates enabled, Decision Tree using the 'entropy' criterion and max depth of 7, Random Forest with 500 trees, Gradient Boosting, AdaBoost with a custom Decision Tree base estimator, Extra Trees, Bagging, XGBoost with 'logloss' evaluation metric, K-Nearest Neighbors (KNN), Naive Bayes, Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), Multi-layer Perceptron (MLP) Neural Network with 500 max iterations, and LightGBM. This setup is typically used for comparing the performance of different classifiers on the same dataset, especially in binary or multi-class classification tasks.

Model Training



```
param_grids = {
    "Logistic Regression": {
        'C': [0.01, 0.1, 1, 10],
        'solver': ['liblinear', 'lbfgs']
    },
    "SVM": {
        'C': [0.1, 1, 10],
        'kernel': ['linear', 'rbf']
    },
    "Decision Tree": {
        'max_depth': [3, 5, 7, 10],
        'min_samples_split': [2, 5, 10]
    },
    "Random Forest": {
        'n_estimators': [100, 300, 500],
        'max_depth': [None, 10, 20]
    },
    "Gradient Boosting": {
        'learning_rate': [0.01, 0.1, 0.2],
        'n_estimators': [100, 200]
    },
    "AdaBoost": {
        'n_estimators': [50, 100, 200],
        'learning_rate': [0.01, 0.1, 1]
    },
    "Extra Trees": {
        'n_estimators': [100, 300, 500],
        'max_depth': [None, 10, 20]
    },
    "Bagging": {
        'n_estimators': [10, 50, 100],
        'max_samples': [0.5, 1.0]
    },
    "XGBoost": {
        'learning_rate': [0.01, 0.1, 0.2],
        'n_estimators': [100, 200],
        'max_depth': [3, 6, 9]
    },
    "KNN": {
        'n_neighbors': [3, 5, 7],
        'weights': ['uniform', 'distance']
    },
    "Naive Bayes": {},  # no parameters to tune for GaussianNB
    "LDA": {},
    "QDA": {},
    "MLP (Neural Network)": {
        'hidden_layer_sizes': [(50,), (100,), (100, 50)],
        'activation': ['relu', 'tanh'],
        'alpha': [0.0001, 0.001]
    },
    "LightGBM": {
        'learning_rate': [0.01, 0.1],
        'n_estimators': [100, 200],
        'num_leaves': [31, 50]
    }
}
```

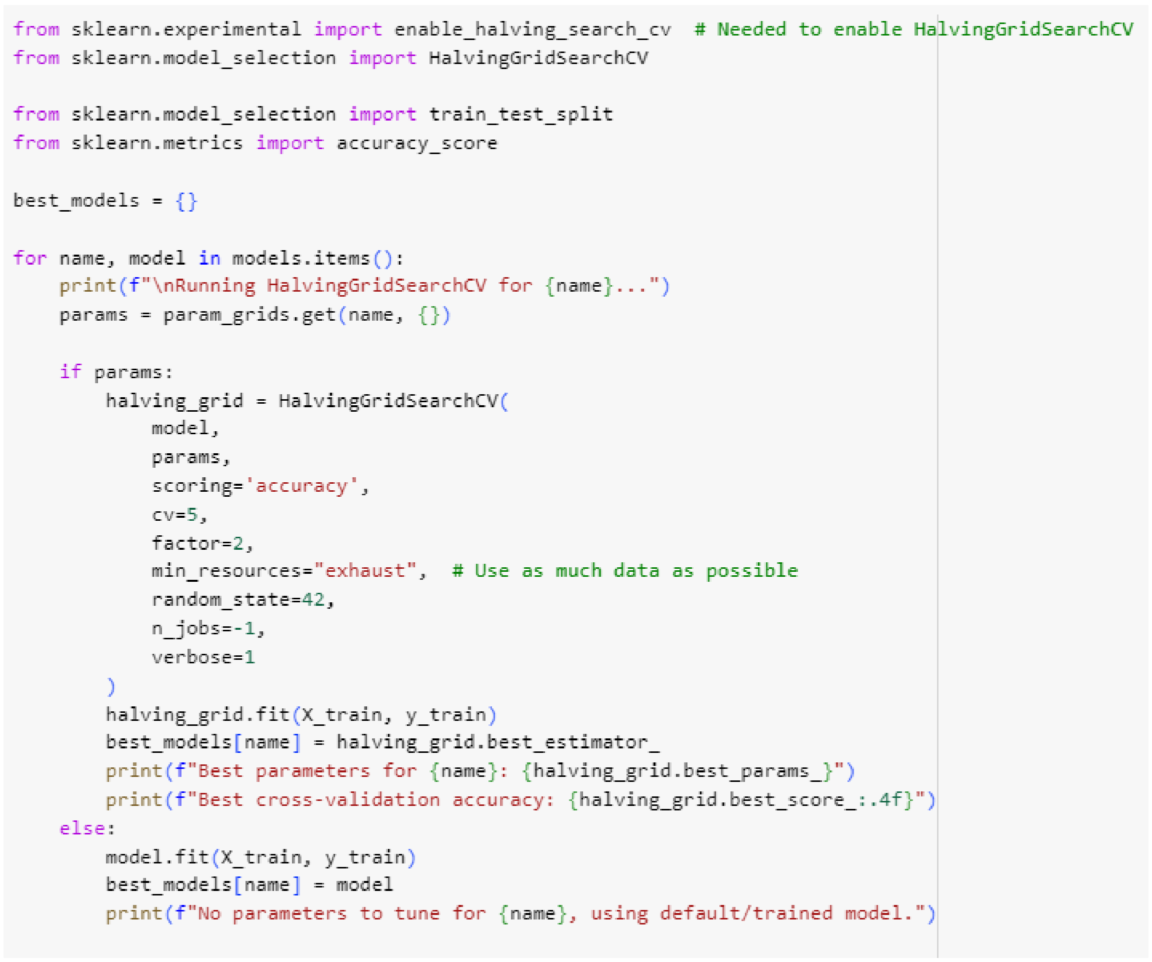
The given code defines a dictionary called param_grids, which specifies the initial hyperparameter search spaces for a variety of machine learning models. These parameter grids are typically used with technique, HalvingGridSearchCV to perform hyperparameter tuning and identify the best-performing configuration for each model. For Logistic Regression, it tunes the regularization strength 'C' with values ranging from 0.01 to 10 and chooses between 'liblinear' and 'lbfgs' solvers. The SVM model explores different 'C' values and kernel types ('linear', 'rbf'). The Decision Tree model is tested with various values for 'max_depth' and 'min_samples_split', while the Random Forest model varies its 'n_estimators' and 'max_depth'.

The Gradient Boosting model is optimized based on its 'learning_rate' and 'n_estimators', and the AdaBoost model includes both of these parameters as well. Extra Trees and Bagging models adjust the number of estimators, and in the case of Bagging, also test different 'max_samples' values. The XGBoost model is fine-tuned using 'learning_rate', 'n_estimators', and 'max_depth'.

For K-Nearest Neighbors (KNN), the number of neighbors and the weighting strategy ('uniform' or 'distance') are considered. Some models like Naive Bayes, LDA, and QDA have no parameters defined here, meaning they will use default settings during training.

The MLP (Neural Network) model is tuned with different hidden layer sizes, activation functions ('relu', 'tanh'), and alpha values (L2 regularization). Lastly, LightGBM is optimized based on 'learning_rate', 'n_estimators', and 'num_leaves', which control the boosting process and tree complexity. Overall, this parameter grid ensures a wide yet efficient search across each model's critical hyperparameters, allowing for systematic performance comparison and improvement.

Using HalvingGridSearchCV to Get Optimum Parameters



```python
from sklearn.experimental import enable_halving_search_cv  # Needed to enable HalvingGridSearchCV
from sklearn.model_selection import HalvingGridSearchCV

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

best_models = {}

for name, model in models.items():
    print(f"\nRunning HalvingGridSearchCV for {name}...")
    params = param_grids.get(name, {})

    if params:
        halving_grid = HalvingGridSearchCV(
            model,
            params,
            scoring='accuracy',
            cv=5,
            factor=2,
            min_resources="exhaust",  # Use as much data as possible
            random_state=42,
            n_jobs=-1,
            verbose=1
        )
        halving_grid.fit(X_train, y_train)
        best_models[name] = halving_grid.best_estimator_
        print(f"Best parameters for {name}: {halving_grid.best_params_}")
        print(f"Best cross-validation accuracy: {halving_grid.best_score_:.4f}")
    else:
        model.fit(X_train, y_train)
        best_models[name] = model
        print(f"No parameters to tune for {name}, using default/trained model.")
```

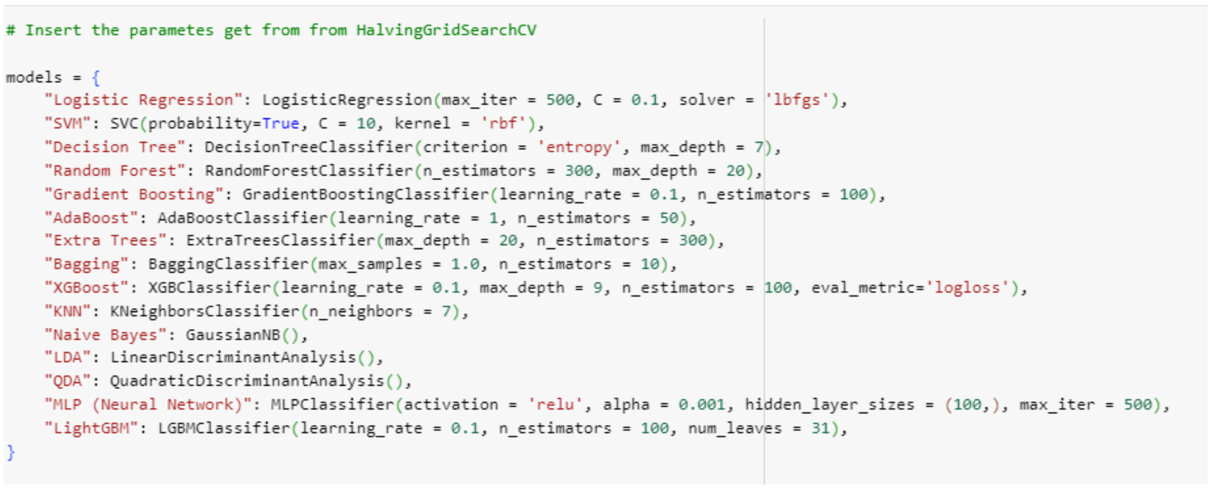
The code provided demonstrates the use of HalvingGridSearchCV, a progressive search strategy used to optimize hyperparameters efficiently by using successively larger portions of the training dataset. The process begins by iterating over a predefined collection of machine learning models stored in a dictionary called models. For each model, the corresponding parameter grid is retrieved from the param_grids dictionary using the model's name as the key. This parameter grid contains different combinations of hyperparameters that the tuning algorithm will evaluate to find the most optimal settings.

If a given model has parameters defined in the grid, the script proceeds to initialize a HalvingGridSearchCV object. This object is configured to optimize the model’s performance based on the 'accuracy' metric and uses 5-fold cross-validation (cv=5) to evaluate each set of parameters. The factor=2 setting means that only half of the candidate estimators will be selected at each iteration, thereby progressively narrowing down to the best-performing models. The min_resources="exhaust" option ensures that the algorithm uses as much of the available training data as possible during each evaluation stage. Additional settings such as random_state=42 (to ensure reproducibility), n_jobs=-1 (to leverage all available CPU cores), and verbose=1 (to provide progress updates) is also specified.

Once the halving search is set up, the model is trained using the training dataset (X_train, y_train). The best estimator found from the search is saved in the best_models dictionary, keyed by the model’s name. The best-performing hyperparameters and the corresponding cross-validation accuracy score are also printed for reference.

In cases where a model does not have any parameters defined in the grid (such as models like Naive Bayes, LDA, or QDA), the script bypasses the halving search and directly fits the model using the training data. This trained model is then stored in the best_models dictionary with its default settings.

Putting the new parameters to the model



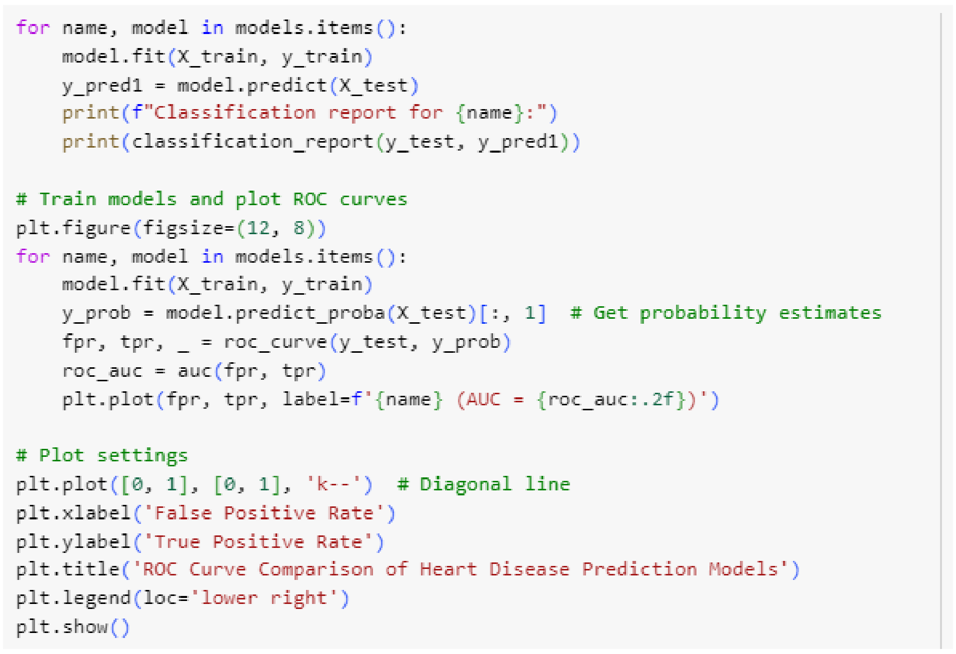
```python
# Insert the parametes get from from HalvingGridSearchCV

models = {
    "Logistic Regression": LogisticRegression(max_iter = 500, C = 0.1, solver = 'lbfgs'),
    "SVM": SVC(probability=True, C = 10, kernel = 'rbf'),
    "Decision Tree": DecisionTreeClassifier(criterion = 'entropy', max_depth = 7),
    "Random Forest": RandomForestClassifier(n_estimators = 300, max_depth = 20),
    "Gradient Boosting": GradientBoostingClassifier(learning_rate = 0.1, n_estimators = 100),
    "AdaBoost": AdaBoostClassifier(learning_rate = 1, n_estimators = 50),
    "Extra Trees": ExtraTreesClassifier(max_depth = 20, n_estimators = 300),
    "Bagging": BaggingClassifier(max_samples = 1.0, n_estimators = 10),
    "XGBoost": XGBClassifier(learning_rate = 0.1, max_depth = 9, n_estimators = 100, eval_metric='logloss'),
    "KNN": KNeighborsClassifier(n_neighbors = 7),
    "Naive Bayes": GaussianNB(),
    "LDA": LinearDiscriminantAnalysis(),
    "QDA": QuadraticDiscriminantAnalysis(),
    "MLP (Neural Network)": MLPClassifier(activation = 'relu', alpha = 0.001, hidden_layer_sizes = (100,), max_iter = 500),
    "LightGBM": LGBMClassifier(learning_rate = 0.1, n_estimators = 100, num_leaves = 31),
}
```

The code shown above updates the previously defined machine learning models with the **best hyperparameters obtained from HalvingGridSearchCV. Each model in the `models` dictionary is now instantiated with its corresponding optimal parameters, replacing the initial default or arbitrary values. These parameters were selected through a systematic tuning process that evaluated different configurations and identified the ones that yielded the highest cross-validation accuracy.

For example, the Logistic Regression model is now configured with `C=0.1` and `solver='lbfgs'`, while the SVM model uses a regularization parameter `C=10` and the `'rbf'` kernel. Similarly, Decision Tree, Random Forest, and Extra Trees models have specific `max_depth` values that control tree complexity, and Gradient Boosting, AdaBoost, and XGBoost models are fine-tuned with ideal combinations of `learning_rate`, `n_estimators`, and `max_depth`. Other models such as KNN use the best number of neighbors (`n_neighbors=7`), while MLP Neural Network is now configured with the best performing `activation`, `alpha`, and `hidden_layer_sizes`.

By applying these tuned parameters, each model is now better equipped to learn from the training data and make more accurate predictions. This optimization process not only enhances model performance but also helps avoid overfitting or underfitting. It ensures that each classifier operates under conditions that are empirically proven to yield the best results based on the training dataset. As a result, these refined models can now be used confidently for further evaluation, testing, or deployment in a real-world scenario.

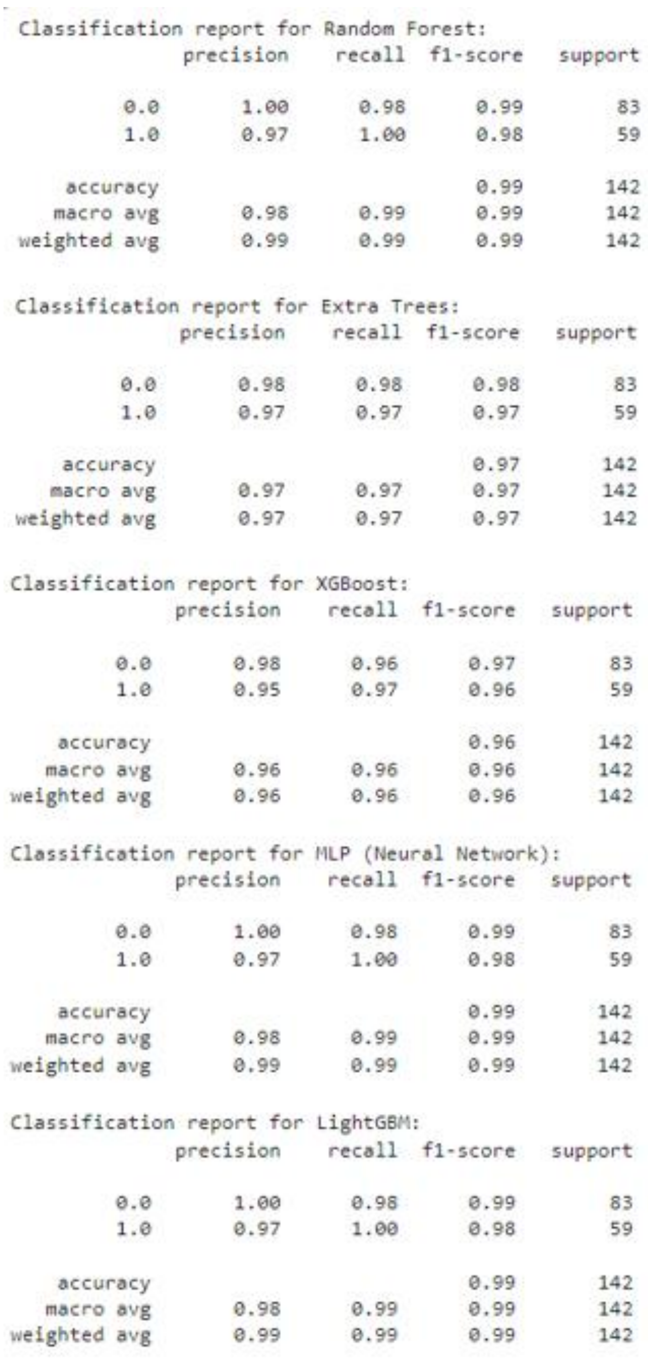
Deploy the models



```python
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred1 = model.predict(X_test)
    print(f"Classification report for {name}:")
    print(classification_report(y_test, y_pred1))

# Train models and plot ROC curves
plt.figure(figsize=(12, 8))
for name, model in models.items():
    model.fit(X_train, y_train)
    y_prob = model.predict_proba(X_test)[:, 1]  # Get probability estimates
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'{name} (AUC = {roc_auc:.2f})')

# Plot settings
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison of Heart Disease Prediction Models')
plt.legend(loc='lower right')
plt.show()
```

The code provided is used to evaluate and visualize the performance of multiple machine learning models after training. It consists of two main parts, which are generating classification reports and plotting ROC curves.

Model Evaluation [For training dataset]



```
Classification report for Random Forest:
              precision    recall  f1-score   support

         0.0       1.00      0.98      0.99        83
         1.0       0.97      1.00      0.98        59

    accuracy                           0.99       142
   macro avg       0.98      0.99      0.99       142
weighted avg       0.99      0.99      0.99       142

Classification report for Extra Trees:
              precision    recall  f1-score   support

         0.0       0.98      0.98      0.98        83
         1.0       0.97      0.97      0.97        59

    accuracy                           0.97       142
   macro avg       0.97      0.97      0.97       142
weighted avg       0.97      0.97      0.97       142

Classification report for XGBoost:
              precision    recall  f1-score   support

         0.0       0.98      0.96      0.97        83
         1.0       0.95      0.97      0.96        59

    accuracy                           0.96       142
   macro avg       0.96      0.96      0.96       142
weighted avg       0.96      0.96      0.96       142

Classification report for MLP (Neural Network):
              precision    recall  f1-score   support

         0.0       1.00      0.98      0.99        83
         1.0       0.97      1.00      0.98        59

    accuracy                           0.99       142
   macro avg       0.98      0.99      0.99       142
weighted avg       0.99      0.99      0.99       142

Classification report for LightGBM:
              precision    recall  f1-score   support

         0.0       1.00      0.98      0.99        83
         1.0       0.97      1.00      0.98        59

    accuracy                           0.99       142
   macro avg       0.98      0.99      0.99       142
weighted avg       0.99      0.99      0.99       142
```

The image presents the classification reports for five different machine learning models: Random Forest, Extra Trees, XGBoost, MLP (Neural Network), and LightGBM, all evaluated on the same test dataset. Each report provides a detailed breakdown of the models' performance across two classes using key metrics such as precision, recall, f1-score, and support (number of true instances per class). Additionally, the overall accuracy, macro average, and weighted average scores are included to summarize performance.

The Random Forest and MLP Neural Network models perform exceptionally well, each achieving 99% accuracy, with near-perfect scores across all metrics. Both models show high consistency in precision and recall for both classes, indicating strong generalization and low bias toward any particular class.

The Extra Trees model also performs well, achieving 97% accuracy, with slightly lower recall and f1-scores for class 1 (0.97). Despite the minor drop, it still shows balanced performance.

XGBoost shows comparatively lower metrics, particularly for class 1, with a recall of 0.95 and an f1-score of 0.96, leading to an overall accuracy of 96%. While still strong, it lags slightly behind the other models in this comparison.

Finally, LightGBM performs on par with Random Forest and MLP, also achieving 99% accuracy. Its precision, recall, and f1-scores are all excellent, indicating that this model is well-calibrated and handles both classes effectively.

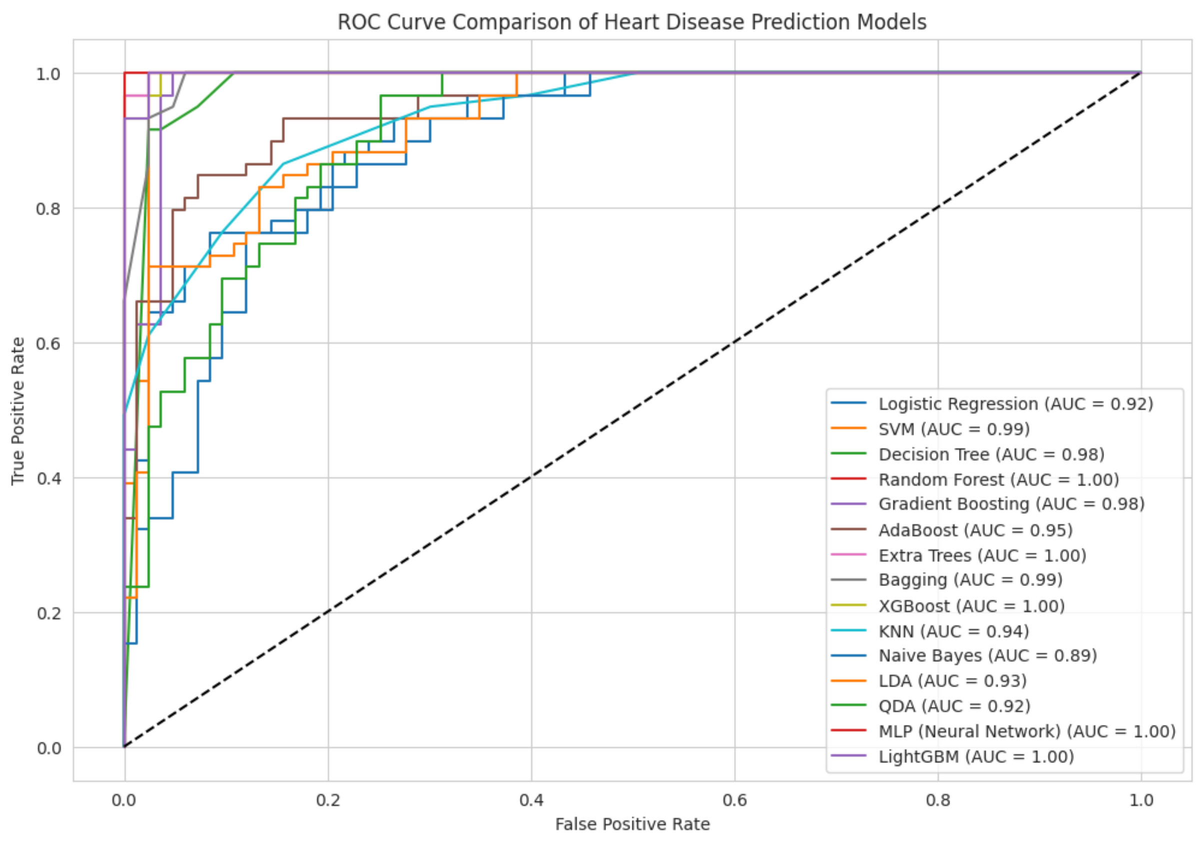
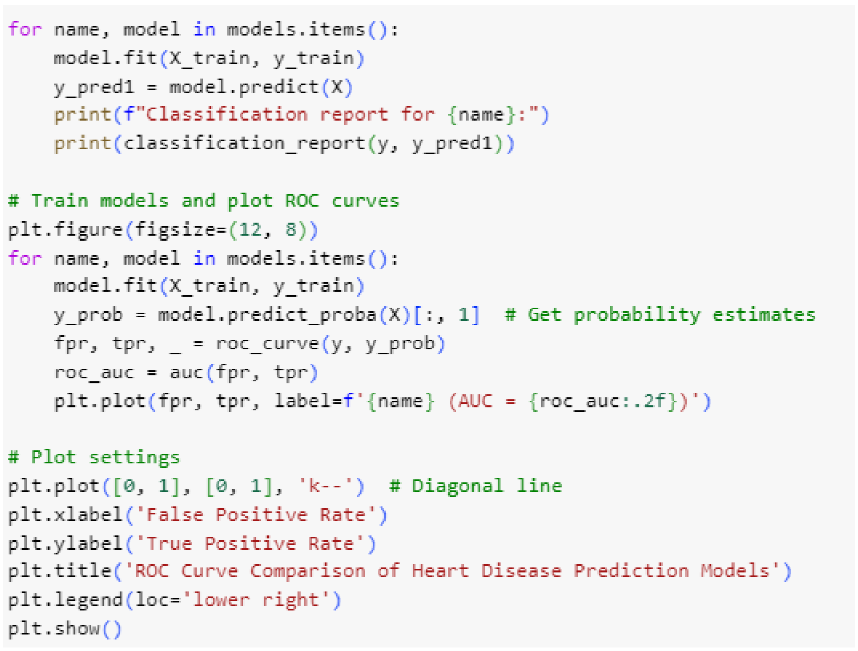


Figure 2: Receiver-operating characteristic (ROC) curve for training dataset

Based on the ROC shown above, the best performing models are Random Forest, Extra Trees, XGBoost, MLP (Neural Network) and LightGBM with their Area Under the Curve being equal to 1.00. There are four other models that not the top performers but are also powerful as their Area under the Curve is between 0.98 and 0.99. The models are SVM (0.99), Bagging (0.99), Decision Tree (0.98) and Gradient Boosting (0.98). Models that have their Area under the Curve between 0.92 and 0.95 are moderately powerful models. These models are Logistic Regression (0.92), KNN (0.94), LDA (0.93), QDA (0.92) and AdaBoost (0.95). The lowest performing model is Naive Bayes where the Area under the Curve is 0.89

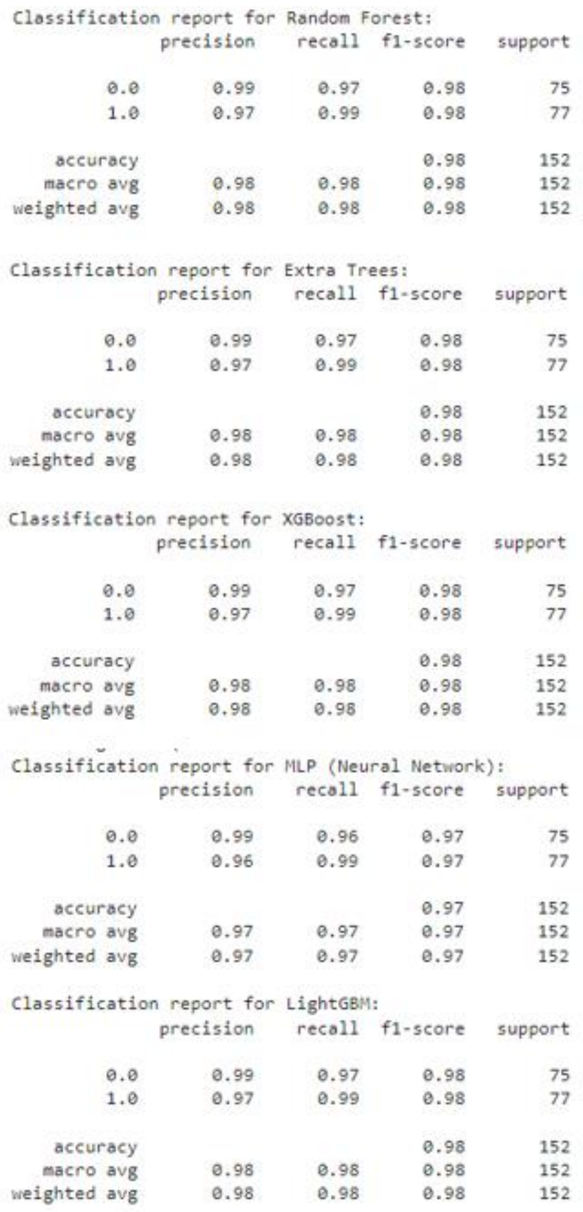
Comparison and Selection

Validation Dataset



```python
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred1 = model.predict(X)
    print(f"Classification report for {name}:")
    print(classification_report(y, y_pred1))

# Train models and plot ROC curves
plt.figure(figsize=(12, 8))
for name, model in models.items():
    model.fit(X_train, y_train)
    y_prob = model.predict_proba(X)[:, 1]  # Get probability estimates
    fpr, tpr, _ = roc_curve(y, y_prob)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'{name} (AUC = {roc_auc:.2f})')

# Plot settings
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison of Heart Disease Prediction Models')
plt.legend(loc='lower right')
plt.show()
```

The code provided is used to evaluate and visualize the performance of multiple machine learning models after training. The same training dataset is used to train the model but this time, all the data in validation dataset was used as prediction.



```
Classification report for Random Forest:
              precision    recall  f1-score   support

         0.0       0.99      0.97      0.98        75
         1.0       0.97      0.99      0.98        77

    accuracy                           0.98       152
   macro avg       0.98      0.98      0.98       152
weighted avg       0.98      0.98      0.98       152

Classification report for Extra Trees:
              precision    recall  f1-score   support

         0.0       0.99      0.97      0.98        75
         1.0       0.97      0.99      0.98        77

    accuracy                           0.98       152
   macro avg       0.98      0.98      0.98       152
weighted avg       0.98      0.98      0.98       152

Classification report for XGBoost:
              precision    recall  f1-score   support

         0.0       0.99      0.97      0.98        75
         1.0       0.97      0.99      0.98        77

    accuracy                           0.98       152
   macro avg       0.98      0.98      0.98       152
weighted avg       0.98      0.98      0.98       152

Classification report for MLP (Neural Network):
              precision    recall  f1-score   support

         0.0       0.99      0.96      0.97        75
         1.0       0.96      0.99      0.97        77

    accuracy                           0.97       152
   macro avg       0.97      0.97      0.97       152
weighted avg       0.97      0.97      0.97       152

Classification report for LightGBM:
              precision    recall  f1-score   support

         0.0       0.99      0.97      0.98        75
         1.0       0.97      0.99      0.98        77

    accuracy                           0.98       152
   macro avg       0.98      0.98      0.98       152
weighted avg       0.98      0.98      0.98       152
```

The image presents the classification reports for five machine learning models, which are Random Forest, Extra Trees, XGBoost, MLP (Neural Network), and LightGBM evaluated using a validation dataset.

The Random Forest model performs with excellent balance, achieving 98% accuracy. It demonstrates a high recall for both class 0 (0.97) and class 1 (0.99), indicating its ability to detect both negative and positive cases effectively with minimal misclassification. Extra Trees matches this performance closely, also hitting 98% accuracy, with very similar precision and recall values, showing it can generalize as reliably as Random Forest.

XGBoost also performs strongly, maintaining 98% accuracy and balanced precision-recall values, particularly a high recall of 0.99 for class 1. This suggests its effectiveness in detecting positive cases accurately, an important trait for medical predictions. MLP (Neural Network) scores slightly lower, with an overall 97% accuracy, due to a marginally lower recall of 0.96 for class 0 and 0.98 for class 1. Despite this, its performance remains highly respectable, especially considering neural networks are typically more complex and data dependent.

LightGBM, like Random Forest and Extra Trees, also achieves 98% accuracy, with excellent precision and recall for both classes. Its scores reflect strong consistency and low error rates, showing it is a robust and efficient gradient boosting model.

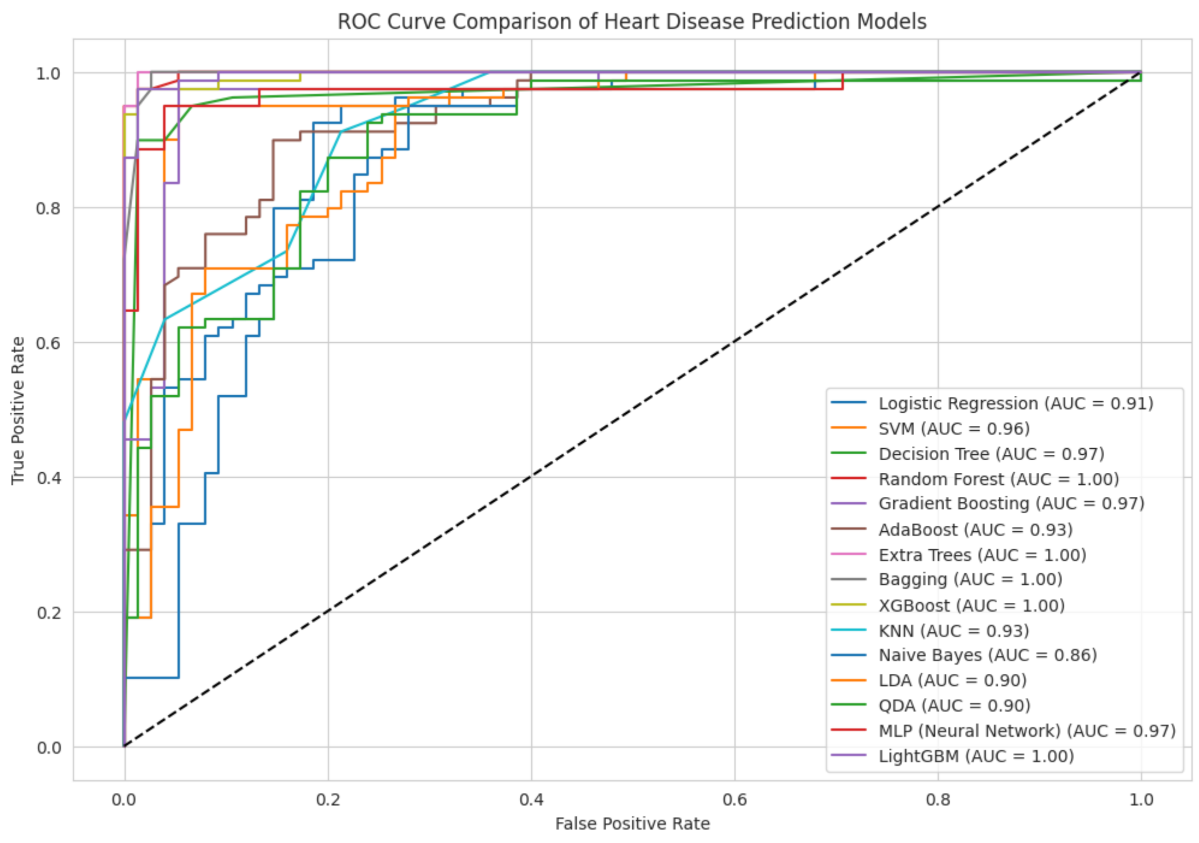
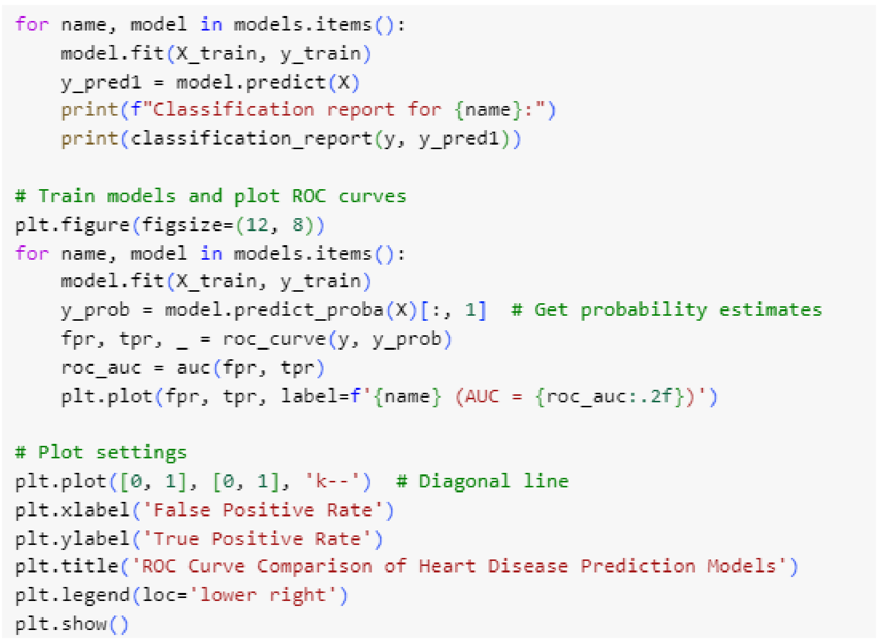


Figure 3: Receiver-operating characteristic (ROC) curve for validation dataset

Based on the ROC shown above, the best performing models are Random Forest, Extra Trees, Bagging, XGBoost and LightGBM with their Area Under the Curve being equal to 1.00. There are four other models that not the top performers but are also powerful as their Area under the Curve is between 0.96 and 0.97. The models are SVM (0.96), MLP Neural Network (0.97), Decision Tree (0.97) and Gradient Boosting (0.97). Models that have their Area under the Curve between 0.90 and 0.93 are weak models. These models are Logistic Regression (0.91`), KNN (0.93), LDA (0.90), QDA (0.90) and AdaBoost (0.93). The lowest performing model is Naive Bayes where the Area under the Curve is 0.86.

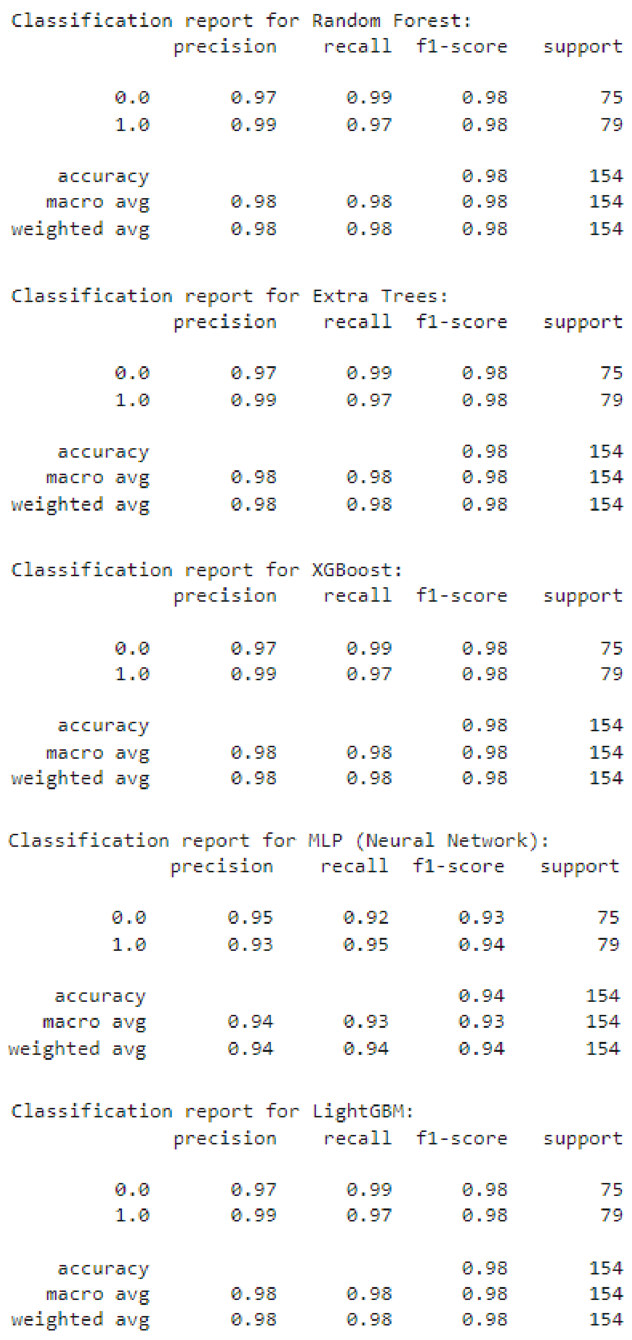
Compared to training dataset, the AUC does not have significant different when the models are used to predict the independent variable in validation dataset. This can indicate that our models do not have overfitting issue.

Testing dataset



```python
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred1 = model.predict(X)
    print(f"Classification report for {name}:")
    print(classification_report(y, y_pred1))

# Train models and plot ROC curves
plt.figure(figsize=(12, 8))
for name, model in models.items():
    model.fit(X_train, y_train)
    y_prob = model.predict_proba(X)[:, 1]  # Get probability estimates
    fpr, tpr, _ = roc_curve(y, y_prob)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'{name} (AUC = {roc_auc:.2f})')

# Plot settings
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison of Heart Disease Prediction Models')
plt.legend(loc='lower right')
plt.show()
```

The code provided is used to evaluate and visualize the performance of multiple machine learning models after training. The same training dataset is used to train the model but this time, all the data in testing dataset was used as prediction.



```
Classification report for Random Forest:
              precision    recall  f1-score   support

         0.0       0.97      0.99      0.98        75
         1.0       0.99      0.97      0.98        79

    accuracy                           0.98       154
   macro avg       0.98      0.98      0.98       154
weighted avg       0.98      0.98      0.98       154

Classification report for Extra Trees:
              precision    recall  f1-score   support

         0.0       0.97      0.99      0.98        75
         1.0       0.99      0.97      0.98        79

    accuracy                           0.98       154
   macro avg       0.98      0.98      0.98       154
weighted avg       0.98      0.98      0.98       154

Classification report for XGBoost:
              precision    recall  f1-score   support

         0.0       0.97      0.99      0.98        75
         1.0       0.99      0.97      0.98        79

    accuracy                           0.98       154
   macro avg       0.98      0.98      0.98       154
weighted avg       0.98      0.98      0.98       154

Classification report for MLP (Neural Network):
              precision    recall  f1-score   support

         0.0       0.95      0.92      0.93        75
         1.0       0.93      0.95      0.94        79

    accuracy                           0.94       154
   macro avg       0.94      0.93      0.93       154
weighted avg       0.94      0.94      0.94       154

Classification report for LightGBM:
              precision    recall  f1-score   support

         0.0       0.97      0.99      0.98        75
         1.0       0.99      0.97      0.98        79

    accuracy                           0.98       154
   macro avg       0.98      0.98      0.98       154
weighted avg       0.98      0.98      0.98       154
```

The image presents the classification reports for five machine learning models, Random Forest, Extra Trees, XGBoost, MLP (Neural Network), and LightGBM evaluated using testing dataset.

The Random Forest model achieves a high 98% accuracy, with excellent recall scores of 0.99 for class 0 and 0.97 for class 1. This reflects its strong ability to correctly classify both negative and positive cases, with only a few misclassifications. Extra Trees mirrors this performance identically, also achieving 98% accuracy and showing nearly identical precision-recall values, confirming its stability and efficiency in classification tasks.

XGBoost similarly demonstrates strong performance, matching the 98% accuracy benchmark and maintaining balanced recall values of 0.99 for class 0 and 0.97 for class 1. This suggests that XGBoost is just as reliable in identifying true cases of heart disease as it is in avoiding false alarms. On the other hand, MLP (Neural Network) lags slightly behind, with an overall accuracy of 94%. Its recall values—0.92 for class 0 and 0.95 for class 1—indicate a few more misclassifications compared to the ensemble models, though its performance remains respectable for a deep learning approach.

Finally, LightGBM performs just as well as the top models, also achieving 98% accuracy with recall scores of 0.99 for class 0 and 0.97 for class 1. This confirms that LightGBM is an effective and efficient model for high-dimensional, structured data, handling the classification task with minimal error.

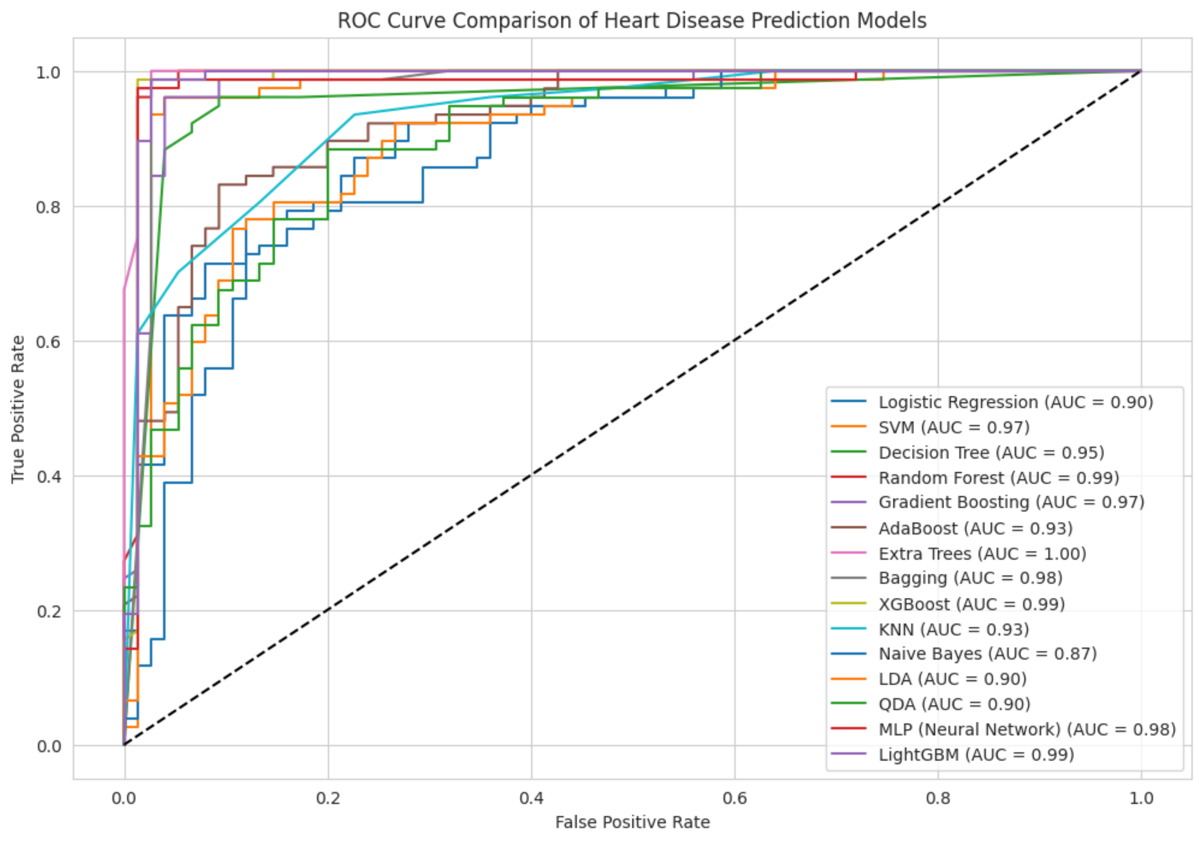


Figure 4: Receiver-operating characteristic (ROC) curve for testing dataset

Based on the ROC shown above, the best performing models are Extra Trees with its Area Under the Curve being equal to 1.00. There are four other models that not the top performers but are also powerful as their Area under the Curve is between 0.98 and 0.99. The models are

XGBoost (0.99), LightGBM (0.99), Random Forest (0.99), Bagging (0.98) and MLP (Neural Net) (0.98). Models that have their Area under the Curve between 0.93 and 0.97 are moderately powerful models. These models are SVM (0.97), Gradient Boosting (0.97), KNN (0.97), Decision Tree (0.95) and AdaBoost (0.93). Models that have their area under the curve at 0.90 or lower are weak models. These models are LDA (0.90), QDA (0.90) and Logistic Regression (0.90). The lowest performing model is Naive Bayes where the Area under the Curve is 0.87

Since many models achieved very high AUC scores, it became challenging to determine which one truly performs best. To address this, the entire process was repeated using the same dataset, but this time without replacing the outliers. The goal was to observe whether retaining the outliers would lead to any significant changes in the AUC scores.

<u>ROC curve (outliers are remains unchanged in data preprocessing part)</u>

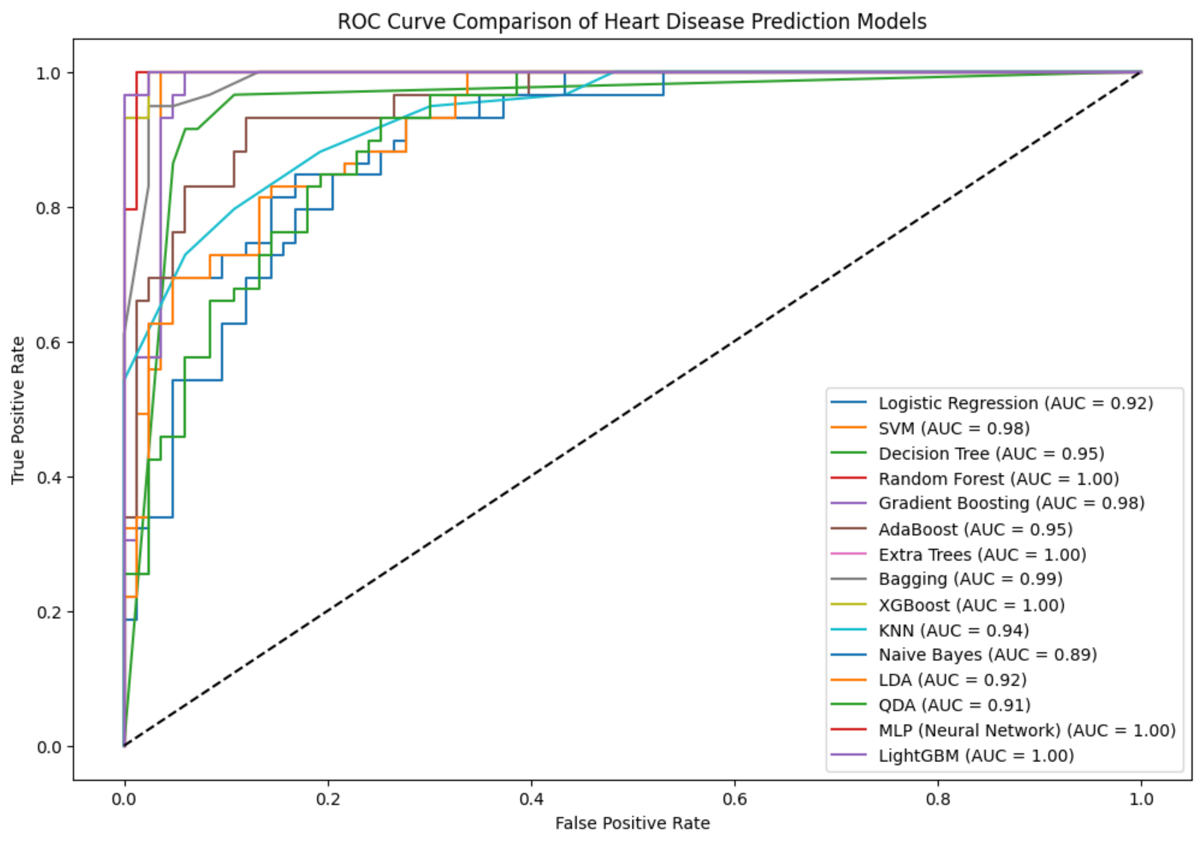


Figure 5: Receiver-operating characteristic (ROC) curve for training dataset (with outlier)

Based on the ROC shown above, the best performing models are Random Forest, Extra Trees, MLP (Neural Network) and LightGBM with their Area Under the Curve being equal to 1.00. There are three other models that not the top performers but are also powerful as their Area under the Curve is between 0.98 and 0.99. The models are SVM (0.98), Gradient Boosting (0.98) and Bagging (0.99). Models that have their Area under the Curve between 0.93 and 0.97 are moderately powerful models. These models are Decision Tree (0.95), KNN (0.94), and AdaBoost (0.95). Models that have their area under the curve between 0.90 and 9.2 are weak models. These models are LDA (0.92) and QDA (0.91). The lowest performing model is Naive Bayes where the Area under the Curve is 0.89.

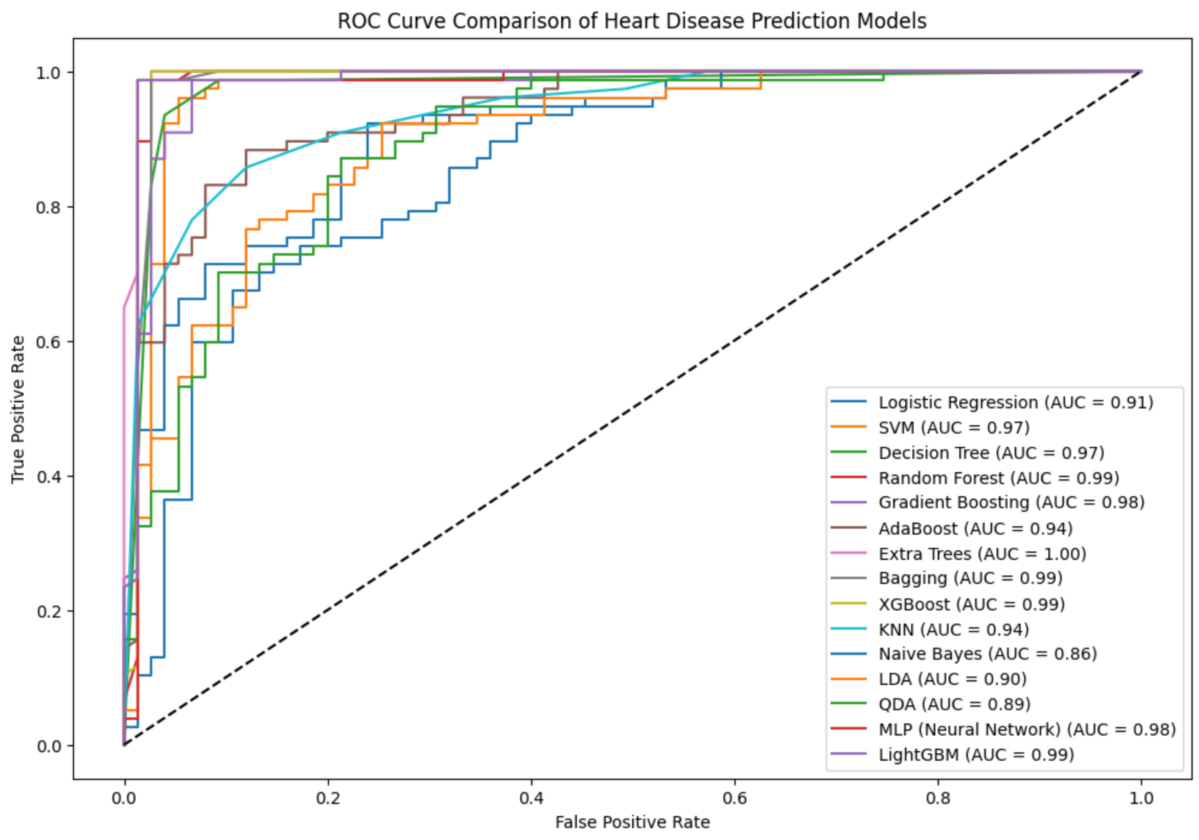


Figure 6: Receiver-operating characteristic (ROC) curve for validation dataset (with outlier)

Based on the ROC shown above, the best performing models is only Extra Trees with their Area Under the Curve being equal to 1.00. There are six other models that not the top performers but are also powerful as their Area under the Curve is between 0.98 and 0.99. The models are Random Forest (0.98), Gradient Boosting (0.98), XGBoost (0.99), MLP (Neural Network) (0.98), LightGBM (0.99) and Bagging (0.99). Models that have their Area under the Curve between 0.93 and 0.97 are moderately powerful models. These models are SVM (0.97), Decision Tree (0.97), AdaBoost (0.94) and KNN (0.94). Models that have their area under the curve between 0.90 and 9.2 are weak models. These models are LDA (0.92) and QDA (0.91). The lowest performing model is Naive Bayes where the Area under the Curve is 0.86

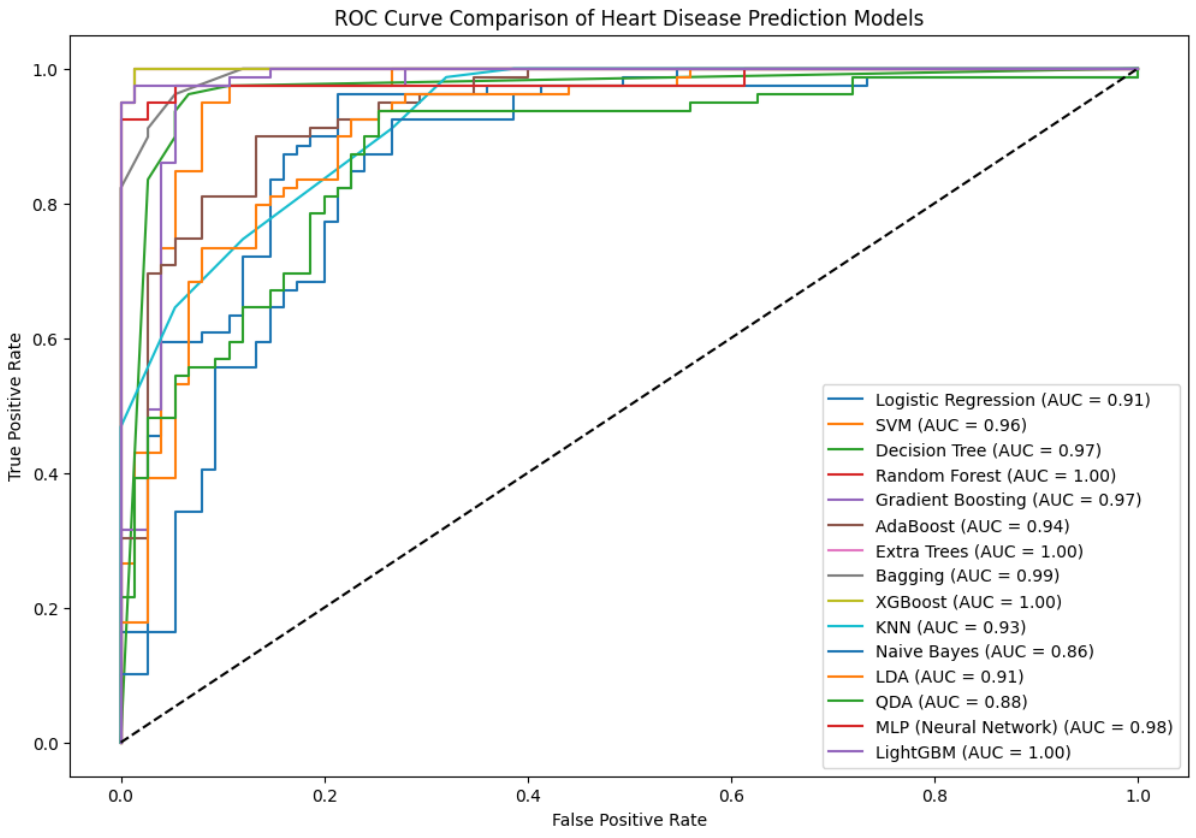


Figure 7: Receiver-operating characteristic (ROC) curve for testing dataset (with outlier)

Based on the ROC shown above, the best performing models is only Random Forest, Extra Trees, XGBoost and LightGBM with their Area Under the Curve being equal to 1.00. There are six other models that not the top performers but are also powerful as their Area under the Curve is between 0.98 and 0.99. The models are Random Forest (0.98), Gradient Boosting (0.98), XGBoost (0.99), MLP (Neural Network) (0.98), LightGBM (0.99) and Bagging (0.99)

Models that have their Area under the Curve between 0.93 and 0.97 are moderately powerful models. These models are SVM (0.96), Decision Tree (0.97), Gradient Boosting (0.97), AdaBoost (0.94) and KNN (0.93). Models that have their area under the curve between 0.90 and 9.2 are weak models. These models are LDA (0.91) and Logistic Regression. (0.91) QDA model has an AUC of 0.88 which makes it a very weak model for doing prediction as it can make predictions with 88% accuracy. The lowest performing model is Naive Bayes where the Area under the Curve is 0.86.

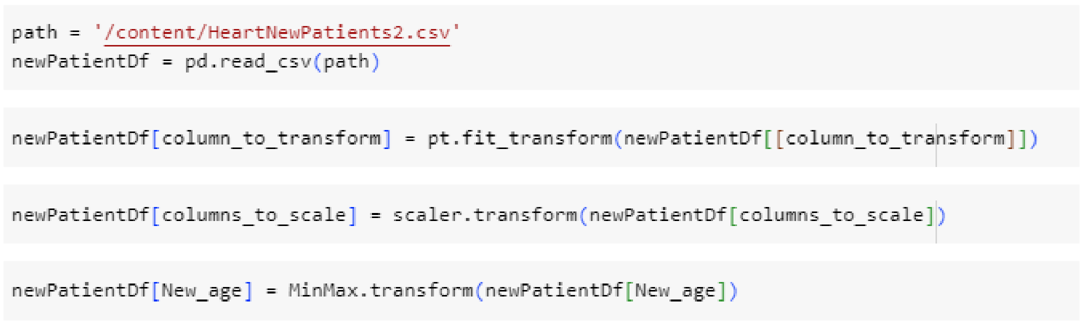
From the graphs above, an interesting observation is that even with the inclusion of outliers during model training, the AUC scores remain largely unaffected. This suggests that the presence of outliers does not significantly impact the models' ability to distinguish between classes. As a result, the AUC score alone is not a reliable metric for selecting the best model for deployment in this case. Hence, the comparison between top 5 best models is done using the matrix below:

Model	**Advantages**	**Disadvantages**
Random Forest	• Works well with numerical and categorical data • Handles overfitting well • Good for categorical	• Need Encoding • Sensitive to outlier • Poor with non-linear data
Extra Trees	• Strong against overfitting • Can handle categorical data after encoding • Fast Training	• Not high accuracy • Doesn't handle categorical before encoding
XGBoost	• Learns data well • Build-in regularization to reduce overfitting	• Require preprocessing for categorical data • Slow for large data set
MLP (Neural Network)	• Learns complex linear boundary	• Doesn't handle categorical • Prone to overfitting • More time to train
LightGBM	• Best for categorical Data • Fast Training • Low memory use • High Accuracy • Can handle complex patterns • Does not need encoding	• Can overfit small datasets if no tuning • Requires tuning

<u>**Choose the best model**</u>

Based on the results of predictions made by all the models and accounting the suitability of model for the data set. The best model is LightGBM. One of the main reasons is that it does not require encoding, and it would not affect the accuracy of the prediction. Another factor is that the model is very fast in training and uses very little memory while being able to handle complex patterns.

<u>Prediction for HeartNewPatient Data</u>



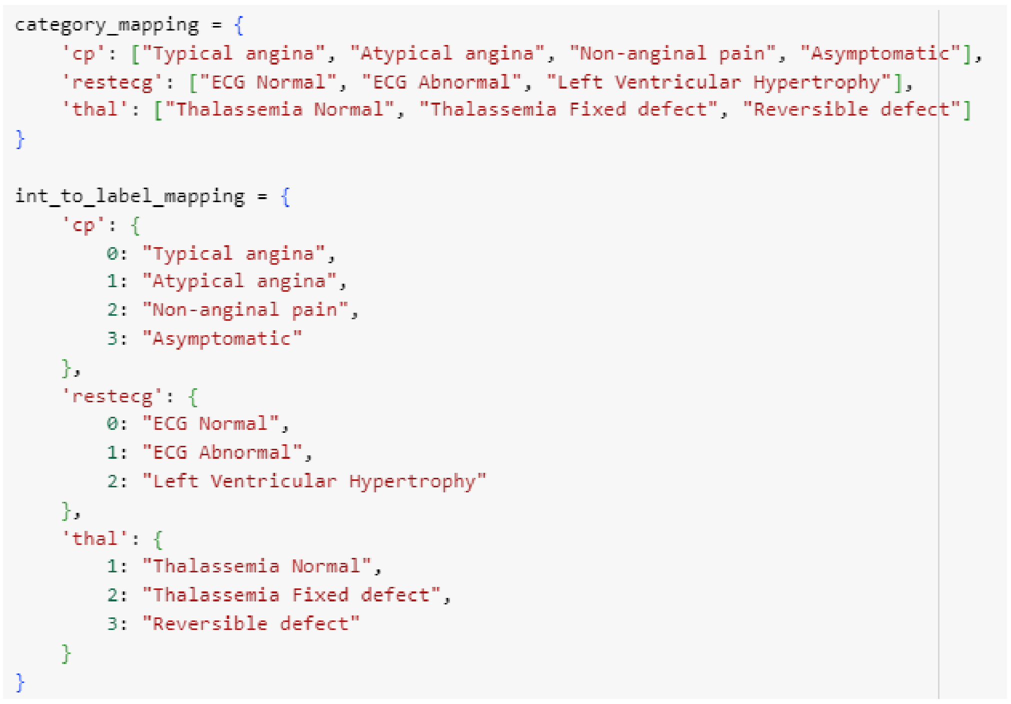
```
path = '/content/HeartNewPatients2.csv'
newPatientDf = pd.read_csv(path)
```

```
newPatientDf[column_to_transform] = pt.fit_transform(newPatientDf[[column_to_transform]])
```

```
newPatientDf[columns_to_scale] = scaler.transform(newPatientDf[columns_to_scale])
```

```
newPatientDf[New_age] = MinMax.transform(newPatientDf[New_age])
```

Firstly, we need to perform the data preprocessing steps to ensure that the data can be fitted to the LightGBM model. The steps include data transform, standardization, normalization and one-hot encoding.



```
category_mapping = {
    'cp': ["Typical angina", "Atypical angina", "Non-anginal pain", "Asymptomatic"],
    'restecg': ["ECG Normal", "ECG Abnormal", "Left Ventricular Hypertrophy"],
    'thal': ["Thalassemia Normal", "Thalassemia Fixed defect", "Reversible defect"]
}

int_to_label_mapping = {
    'cp': {
        0: "Typical angina",
        1: "Atypical angina",
        2: "Non-anginal pain",
        3: "Asymptomatic"
    },
    'restecg': {
        0: "ECG Normal",
        1: "ECG Abnormal",
        2: "Left Ventricular Hypertrophy"
    },
    'thal': {
        1: "Thalassemia Normal",
        2: "Thalassemia Fixed defect",
        3: "Reversible defect"
    }
}
```

The code above defines two category-to-label mappings for categorical data in the new data set. This is useful for interpreting model predictions and promoting human readable labels from numerical values. This is important to do as the model will only receive numerical values, but humans will need a label to help with identifying and making predictions

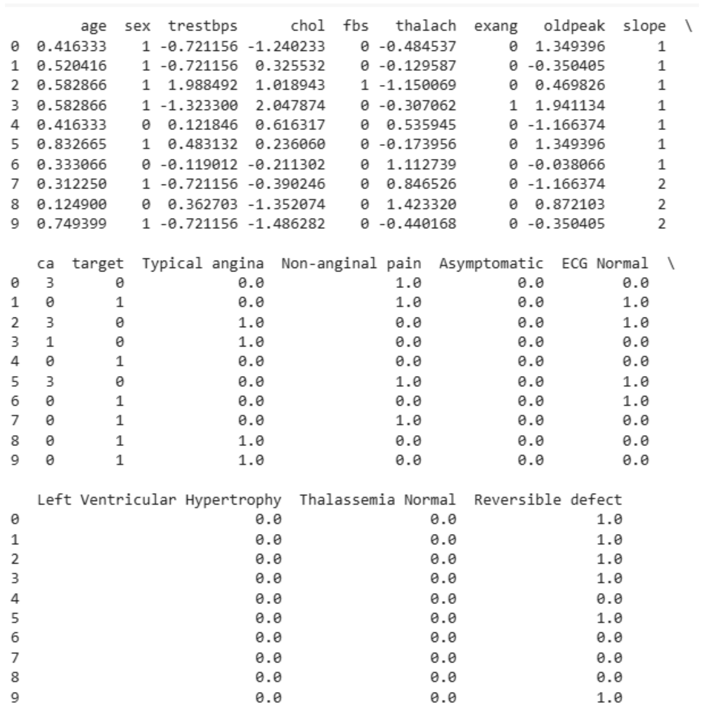
```python
def one_hot_encode_columns_newPatient(df, columns, category_mapping, int_to_label_mapping):
    df_encoded = df.copy()

    for col in columns:
        if col not in df_encoded.columns:
            print(f"Warning: Column '{col}' not in DataFrame. Skipping.")
            continue

        # 1. Convert integers to descriptive labels
        if col in int_to_label_mapping:
            df_encoded[col] = df_encoded[col].map(int_to_label_mapping[col])
        else:
            raise ValueError(f"No int_to_label_mapping provided for column '{col}'")

        # 2. Create encoder with descriptive category names
        encoder = OneHotEncoder(
            sparse_output=False,
            handle_unknown='ignore',
            categories=[category_mapping[col]]
        )

        encoded_array = encoder.fit_transform(df_encoded[[col]])
        feature_names = encoder.get_feature_names_out([col])  # Includes names
        feature_names = [name.split('_', 1)[-1] for name in feature_names]

        encoded_array = np.delete(encoded_array, 1, axis=1)
        feature_names.pop(1)  # Remove the second feature name as well

        # 3. Create one-hot encoded DataFrame
        encoded_df = pd.DataFrame(encoded_array, columns=feature_names, index=df_encoded.index)

        # 4. Drop original column and join new ones
        df_encoded = df_encoded.drop(columns=[col]).join(encoded_df)

    return df_encoded
```

```python
encoded_heartNewPatientsDf = one_hot_encode_columns_newPatient(newPatientDf, ['cp', 'restecg', 'thal'], category_mapping, int_to_label_mapping)
```

The code above is designed to perform one-hot encoding on integer-labeled categorical columns, transforming them into human-readable descriptive labels based on a predefined mapping. It first loops through the specified columns, converting any integers into descriptive text labels according to the provided mapping and storing the results in a new dataframe called df_encoded. For each column, an encoder is created to generate one-hot encoded features, ensuring that the categories are properly named.

After encoding, the feature names are processed to remove any unwanted column prefixes to make the names cleaner. Additionally, the second feature is deliberately removed to prevent potential multicollinearity issues. The resulting one-hot encoded features are stored in a new dataframe called encoded_df. Finally, the original column is dropped from df_encoded, and the newly created encoded features are joined in its place. This approach ensures that the categorical variables are properly prepared for machine learning models while maintaining interpretability.

Result of encoding:



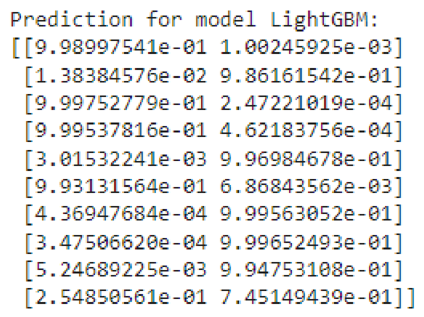
```
        age  sex  trestbps      chol  fbs   thalach  exang   oldpeak  slope  \
0  0.416333    1 -0.721156 -1.240233    0 -0.484537      0  1.349396      1
1  0.520416    1 -0.721156  0.325532    0 -0.129587      0 -0.350405      1
2  0.582866    1  1.988492  1.018943    1 -1.150069      0  0.469826      1
3  0.582866    1 -1.323300  2.047874    0 -0.307062      1  1.941134      1
4  0.416333    0  0.121846  0.616317    0  0.535945      0 -1.166374      1
5  0.832665    1  0.483132  0.236060    0 -0.173956      0  1.349396      1
6  0.333066    0 -0.119012 -0.211302    0  1.112739      0 -0.038066      1
7  0.312250    1 -0.721156 -0.390246    0  0.846526      0 -1.166374      2
8  0.124900    0  0.362703 -1.352074    0  1.423320      0  0.872103      2
9  0.749399    1 -0.721156 -1.486282    0 -0.440168      0 -0.350405      2

   ca  target  Typical angina  Non-anginal pain  Asymptomatic  ECG Normal  \
0   3       0             0.0               1.0           0.0         0.0
1   0       1             0.0               1.0           0.0         1.0
2   3       0             1.0               0.0           0.0         1.0
3   1       0             1.0               0.0           0.0         0.0
4   0       1             0.0               0.0           0.0         0.0
5   3       0             0.0               1.0           0.0         1.0
6   0       1             0.0               0.0           0.0         1.0
7   0       1             0.0               1.0           0.0         0.0
8   0       1             1.0               0.0           0.0         0.0
9   0       1             1.0               0.0           0.0         0.0

   Left Ventricular Hypertrophy  Thalassemia Normal  Reversible defect
0                           0.0                 0.0                1.0
1                           0.0                 0.0                1.0
2                           0.0                 0.0                1.0
3                           0.0                 0.0                1.0
4                           0.0                 0.0                0.0
5                           0.0                 0.0                1.0
6                           0.0                 0.0                0.0
7                           0.0                 0.0                0.0
8                           0.0                 0.0                0.0
9                           0.0                 0.0                1.0
```

Deploy the best model to HeartNewPatient dataset

```python
X = encoded_heartNewPatientsDf.drop(columns=["target"])
y = encoded_heartNewPatientsDf["target"]
```

```python
# Define only the LightGBM model
models = {
    "LightGBM": LGBMClassifier(learning_rate = 0.1, n_estimators = 100, num_leaves = 31),
}

# Deploy the model for real data and print the result
plt.figure(figsize=(12, 8))

# Run only LightGBM model
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict_proba(X)
    print(f"Prediction for model {name}:")
    print(f"{y_pred}")
```

The code above will use to perform predictions on LightGBM model. The following below are the models with the best predictions for heart disease.



```
Prediction for model LightGBM:
[[9.98997541e-01 1.00245925e-03]
 [1.38384576e-02 9.86161542e-01]
 [9.99752779e-01 2.47221019e-04]
 [9.99537816e-01 4.62183756e-04]
 [3.01532241e-03 9.96984678e-01]
 [9.93131564e-01 6.86843562e-03]
 [4.36947684e-04 9.99563052e-01]
 [3.47506620e-04 9.99652493e-01]
 [5.24689225e-03 9.94753108e-01]
 [2.54850561e-01 7.45149439e-01]]
```

Figure 8: The result of predictions from LightGBM

The output shown represents the predicted probabilities from the LightGBM model. In each prediction pair, the first value corresponds to the probability of being classified as class 0 (lower risk of heart disease), while the second value corresponds to class 1 (higher risk of heart disease).

Patient	Target (0: Negative/ 1: Positive)
1	0
2	1
3	0
4	0
5	1
6	0
7	1
8	1
9	1
10	1

Table 1: The result of predictions in binary form

Analyzing the prediction results, LightGBM predicts that out of the 10 individuals, 4 are likely negative cases for heart disease (higher probability towards class 0) and 6 are likely positive cases (higher probability towards class 1). This means that for the given dataset of 10 people, the model estimates that 4 out of 10 individuals are at low risk of developing heart disease, while 6 out of 10 individuals are at high risk and may require further medical attention or monitoring.