Department of Computer Science & Engineering

Project

Semester: Fall-2023
Course Code: CSE366
Course Title: Artificial Intelligence
Section: 04

SUBMITTED TO:

Amit Mandal

Department of CSE, East West University

SUBMITTED BY:

Student Name	Student Id
Simanta Saha	2020-1-60-085
Samiha Afrin	2020-2-60-121
Korobi Sarkar	2020-1-60-161
Noor Shat Zahan	2020-2-60-014

“Heart Attack Prediction”

Heart attack prediction projects involve developing machine learning models to predict the likelihood of an individual experiencing a heart attack based on various health related features. This project aims to automate the process of risk assessment and early detection of potential heart related issues by leveraging advanced data science techniques. The project typically follows a structured workflow. Starting with the acquisition of a comprehensive heart attack prediction dataset. Machine learning models, such as Random Forest and decision tree Algorithm, are then trained on this data to accurately classify and predict Heart attack possibility. The goal is to create a reliable system that can predict the possibility of heart attack based on their visual characteristics.

In this project we are going to predict heart attack possibilities according to our dataset.

our project Colab link:
https://colab.research.google.com/drive/1yPNNizch-3AeiZFJTKbgPdlcmgP-PfQU?usp=sharing&fbclid=IwAR0NVUrH6ZnP7XNq7PK3VcOgvDgsSV_zJqrc579zvJLCg3XEBzfyIoW7xV8#scrollTo=ODoXl4A7d5gp

Key Phases for Our Project:

- Dataset Collection
- Data Preprocessing
- Feature Extraction
- Model Selection
- Model Training
- Model Evaluation
- Visualization
- Hyperparameter

Libraries Used:

Pandas (import pandas as pd):

Pandas is a popular Python library for data manipulation and analysis. It provides data structures and functions needed to work with structured data seamlessly.

NumPy(import numpy as np):

A robust Python library for numerical computations is called NumPy. Large, multi-dimensional arrays and matrices are supported, and mathematical functions can be applied to these components.

Matplotlib (import matplotlib.pyplot as plt):

A well-liked charting tool for Python that can be used to create static, interactive, and animated visualizations is called Matplotlib.A straightforward interface for making different kinds of plots and charts is offered by the pyplot package.

Seaborn (import seaborn as sns):

A Matplotlib-based library for statistical data visualization is called Seaborn. It offers a sophisticated drawing tool for creating eye-catching and educational statistical visuals.

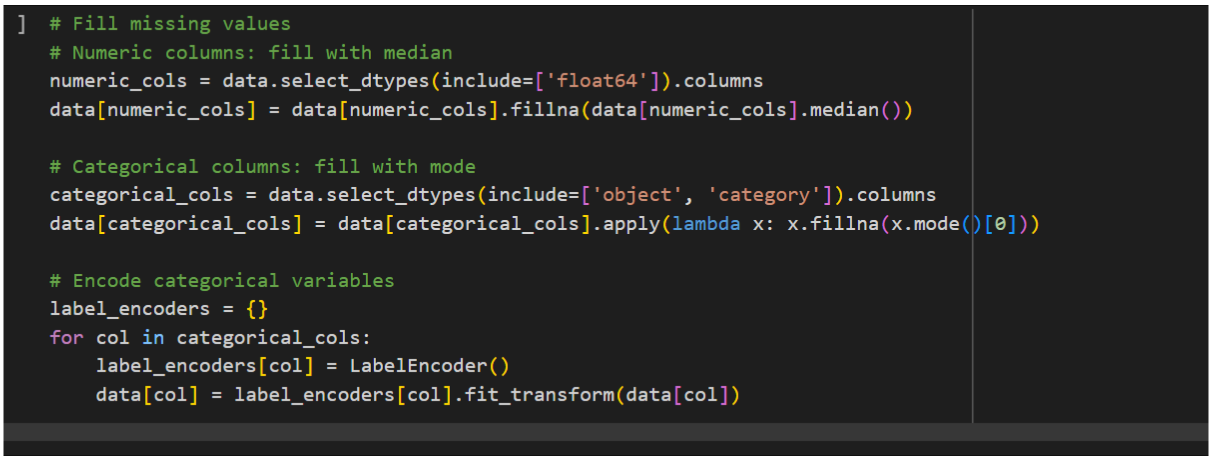
Import Dataset:

```
file_path = 'heart_AI.csv'

data = pd.read_csv(file_path)
```

Normal Prediction:

Data Preprocessing:

Data preprocessing is a crucial step in the data analysis and machine learning pipeline. It involves cleaning and transforming raw data into a format that is suitable for analysis or training machine learning models. The goal of data preprocessing is to enhance the quality of the data, address missing values or errors, and prepare it for effective use in downstream tasks.

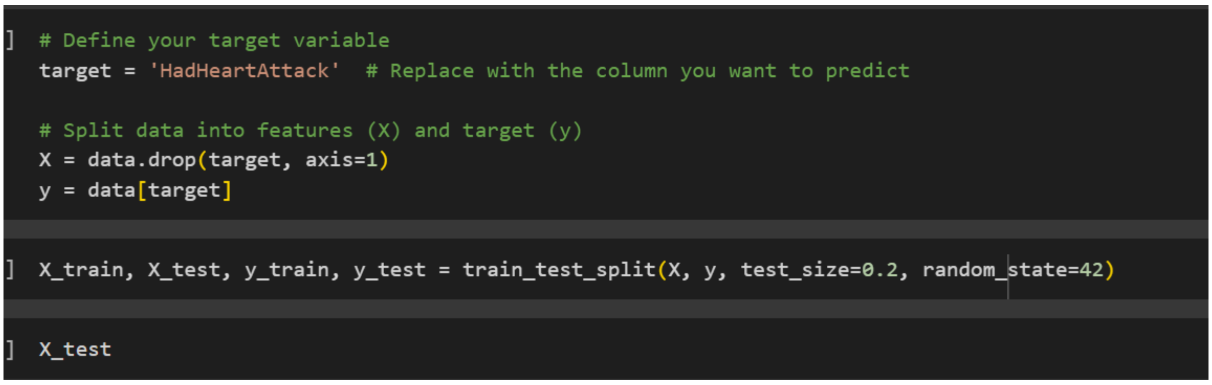


```
] # Fill missing values
  # Numeric columns: fill with median
  numeric_cols = data.select_dtypes(include=['float64']).columns
  data[numeric_cols] = data[numeric_cols].fillna(data[numeric_cols].median())

  # Categorical columns: fill with mode
  categorical_cols = data.select_dtypes(include=['object', 'category']).columns
  data[categorical_cols] = data[categorical_cols].apply(lambda x: x.fillna(x.mode()[0]))

  # Encode categorical variables
  label_encoders = {}
  for col in categorical_cols:
      label_encoders[col] = LabelEncoder()
      data[col] = label_encoders[col].fit_transform(data[col])
```

Dataset split:

In the Dataset split first we split the dataset into train & test data.

We again split the train dataset into train data and validation data.

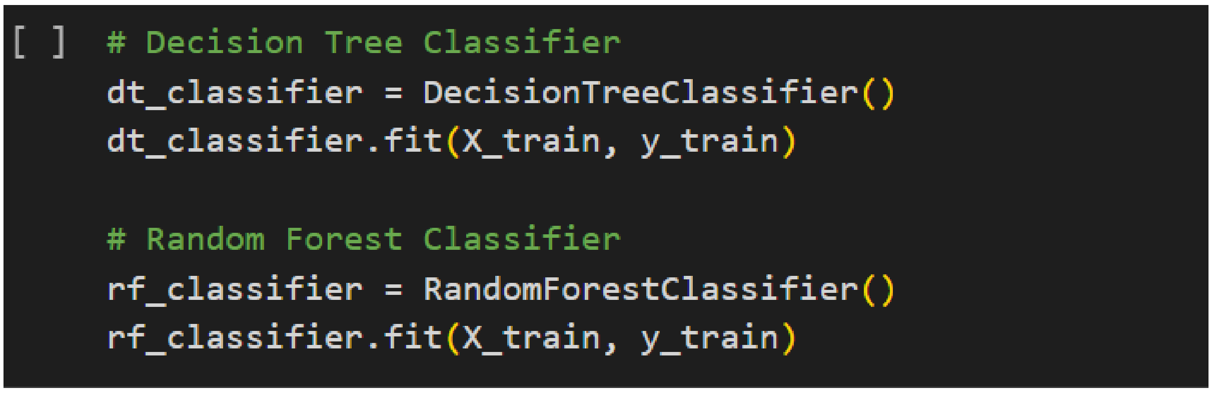


```
] # Define your target variable
  target = 'HadHeartAttack'  # Replace with the column you want to predict

  # Split data into features (X) and target (y)
  X = data.drop(target, axis=1)
  y = data[target]
```

```
] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

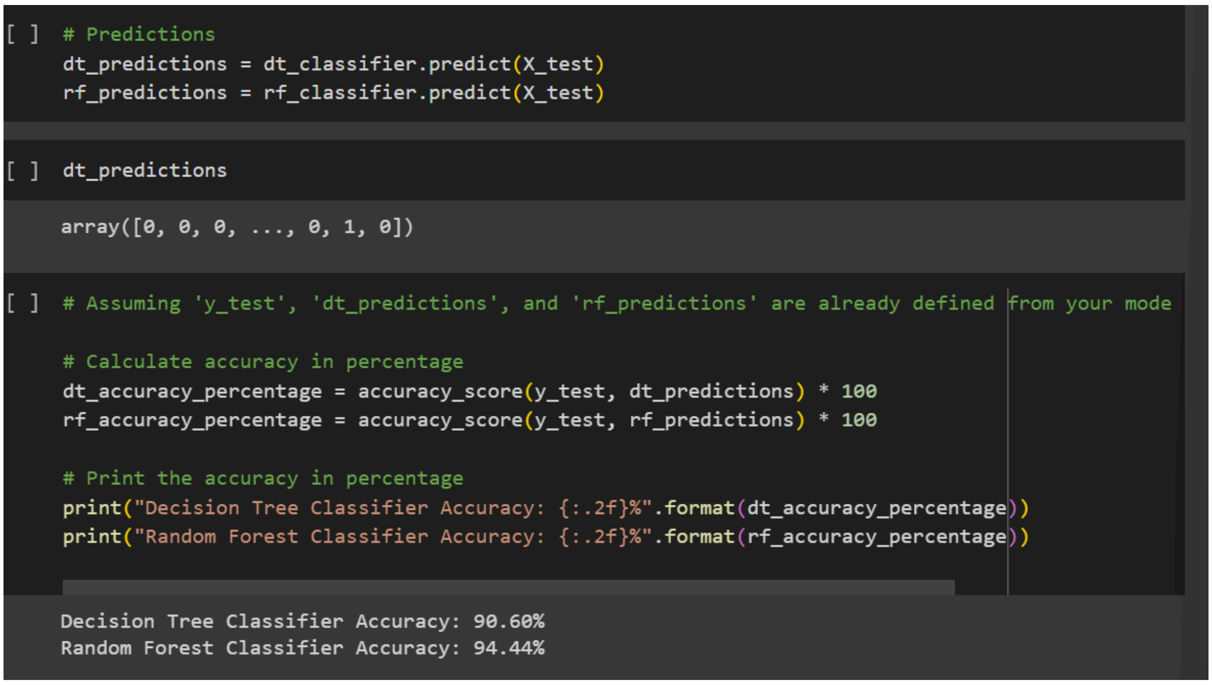
```
] X_test
```

Applied classifier:



```
[ ]  # Decision Tree Classifier
     dt_classifier = DecisionTreeClassifier()
     dt_classifier.fit(X_train, y_train)

     # Random Forest Classifier
     rf_classifier = RandomForestClassifier()
     rf_classifier.fit(X_train, y_train)
```

Predictions:



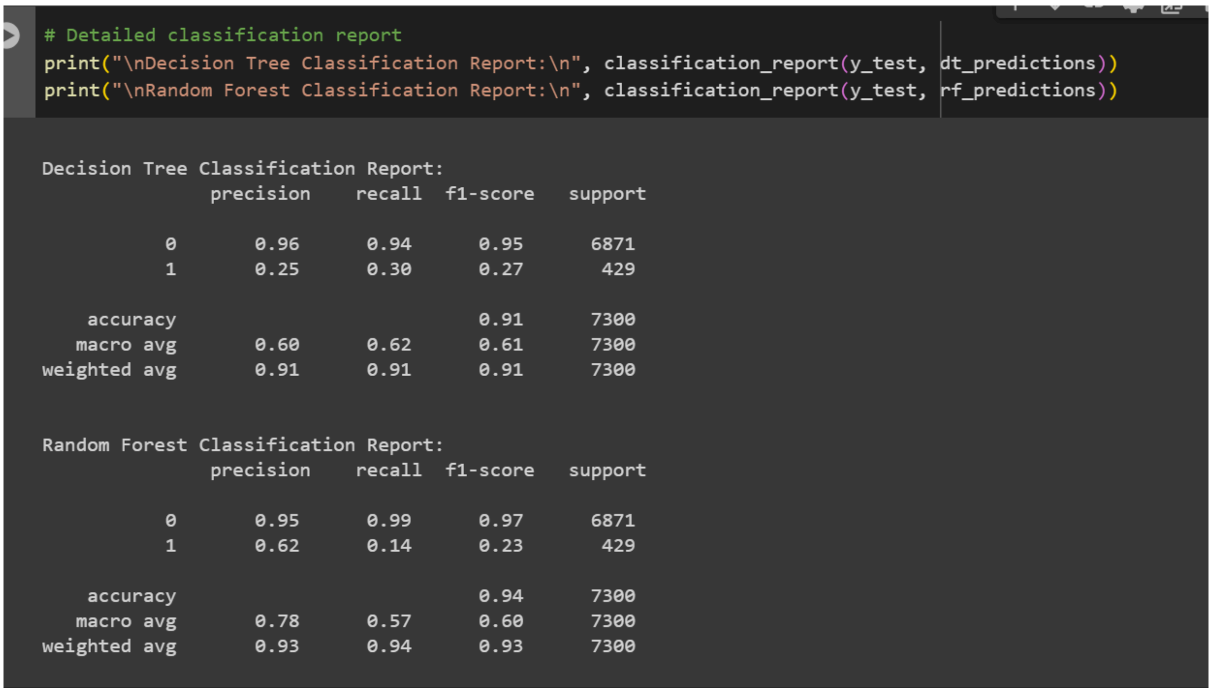
```
[ ]  # Predictions
     dt_predictions = dt_classifier.predict(X_test)
     rf_predictions = rf_classifier.predict(X_test)
```

```
[ ]  dt_predictions

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

```
[ ]  # Assuming 'y_test', 'dt_predictions', and 'rf_predictions' are already defined from your mode

     # Calculate accuracy in percentage
     dt_accuracy_percentage = accuracy_score(y_test, dt_predictions) * 100
     rf_accuracy_percentage = accuracy_score(y_test, rf_predictions) * 100

     # Print the accuracy in percentage
     print("Decision Tree Classifier Accuracy: {:.2f}%".format(dt_accuracy_percentage))
     print("Random Forest Classifier Accuracy: {:.2f}%".format(rf_accuracy_percentage))

     Decision Tree Classifier Accuracy: 90.60%
     Random Forest Classifier Accuracy: 94.44%
```

Here we see that Random Forest Classifier accuracy is more then Decision tree classifier

Detailed classification report:

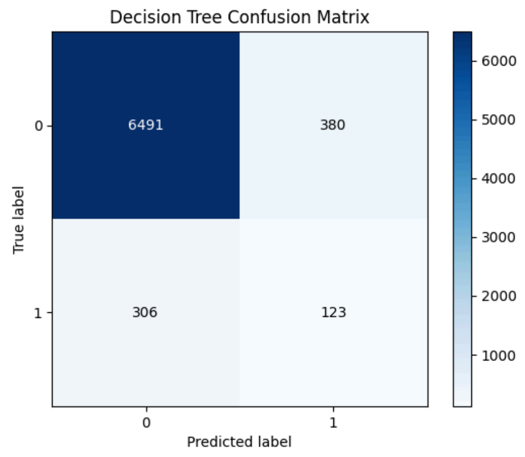


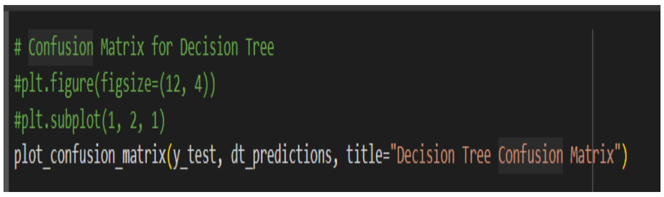
```
# Detailed classification report
print("\nDecision Tree Classification Report:\n", classification_report(y_test, dt_predictions))
print("\nRandom Forest Classification Report:\n", classification_report(y_test, rf_predictions))


Decision Tree Classification Report:
               precision    recall  f1-score   support

           0       0.96      0.94      0.95      6871
           1       0.25      0.30      0.27       429

    accuracy                           0.91      7300
   macro avg       0.60      0.62      0.61      7300
weighted avg       0.91      0.91      0.91      7300


Random Forest Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.99      0.97      6871
           1       0.62      0.14      0.23       429

    accuracy                           0.94      7300
   macro avg       0.78      0.57      0.60      7300
weighted avg       0.93      0.94      0.93      7300
```

Visualization

Confusion Matrix:

Decision Tree Confusion Matrix:

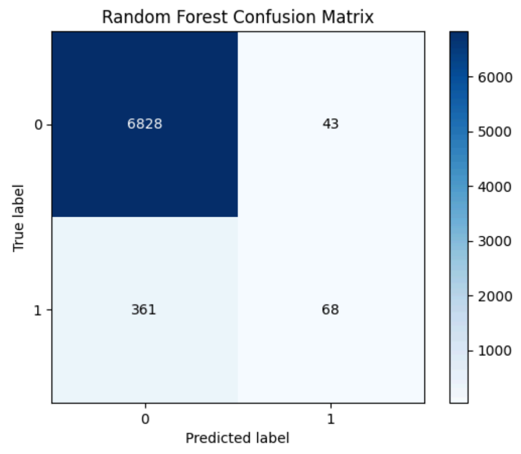


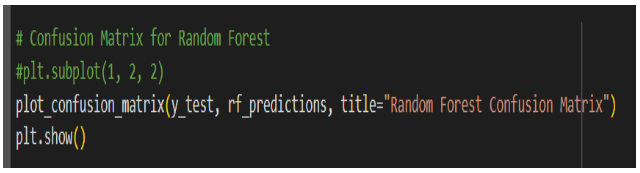


```
# Confusion Matrix for Decision Tree
#plt.figure(figsize=(12, 4))
#plt.subplot(1, 2, 1)
plot_confusion_matrix(y_test, dt_predictions, title="Decision Tree Confusion Matrix")
```

This code snippet generates a confusion matrix for a Decision Tree classifier using scikit-learn's plot_confusion_matrix function. The confusion matrix provides a visual representation of the classifier's performance by showing the counts of true positive, true negative, false positive, and false negative predictions. The confusion matrix is a valuable tool for assessing the classifier's performance, especially in binary classification scenarios.

Random Forest Confusion Matrix:



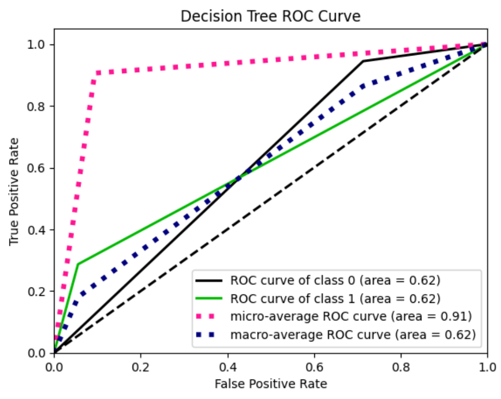


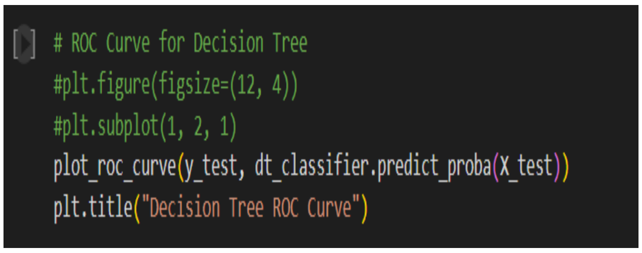
```
# Confusion Matrix for Random Forest
#plt.subplot(1, 2, 2)
plot_confusion_matrix(y_test, rf_predictions, title="Random Forest Confusion Matrix")
plt.show()
```

In this code, y_test represents the true labels of the test set, and rf_predictions represents the predicted labels from the Random Forest classifier for the same test set. The title of the confusion matrix plot is set to "Random Forest Confusion Matrix." The plt.subplot(1, 2, 2) indicates an intention to create a subplot, with the assumption that the previous confusion matrix for the Decision Tree was in the first subplot. This organisation allows for side-by-side comparison of confusion matrices.

ROC Curve:

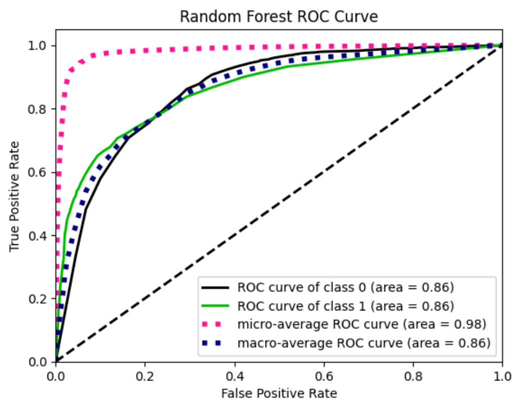
The ROC (Receiver Operating Characteristic) curve is a graphical representation of the performance of a binary classification model across various classification thresholds. It plots the true positive rate (sensitivity) against the false positive rate (1 - specificity) for different threshold values.

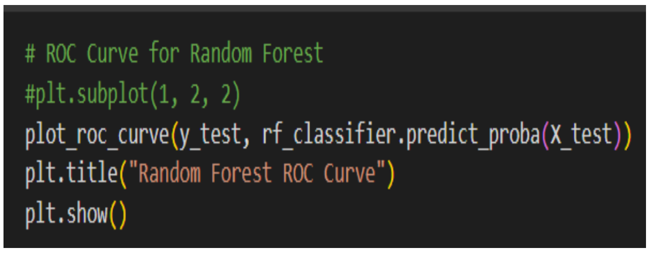




```
# ROC Curve for Decision Tree
#plt.figure(figsize=(12, 4))
#plt.subplot(1, 2, 1)
plot_roc_curve(y_test, dt_classifier.predict_proba(X_test))
plt.title("Decision Tree ROC Curve")
```

The Decision Tree ROC Curve provides a visual representation of the model's discrimination ability across different thresholds. The fact that we have a curve above the random baseline suggests that the model is performing better than random chance. The specific shape and position of the curve give insights into the trade-offs between sensitivity and specificity for the Decision Tree model.

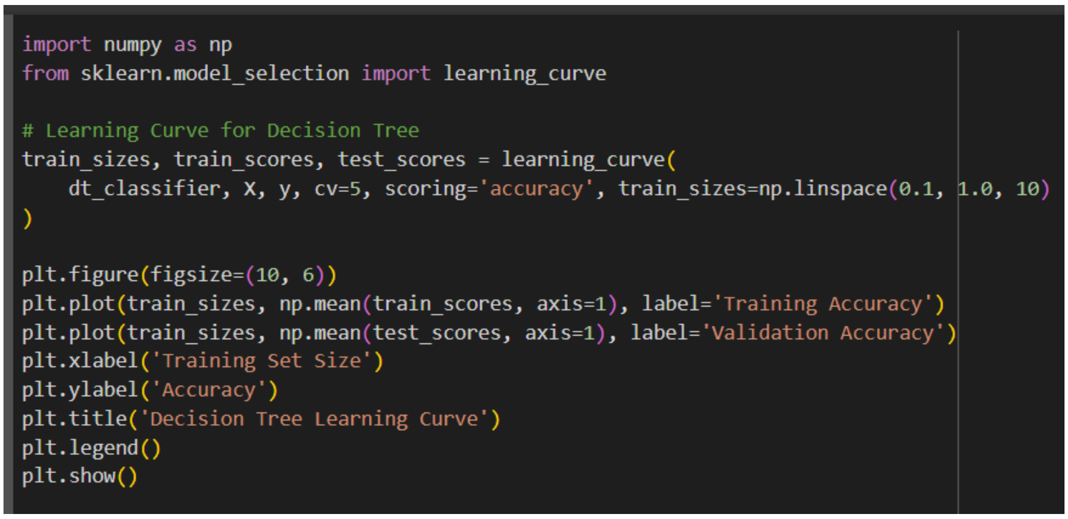




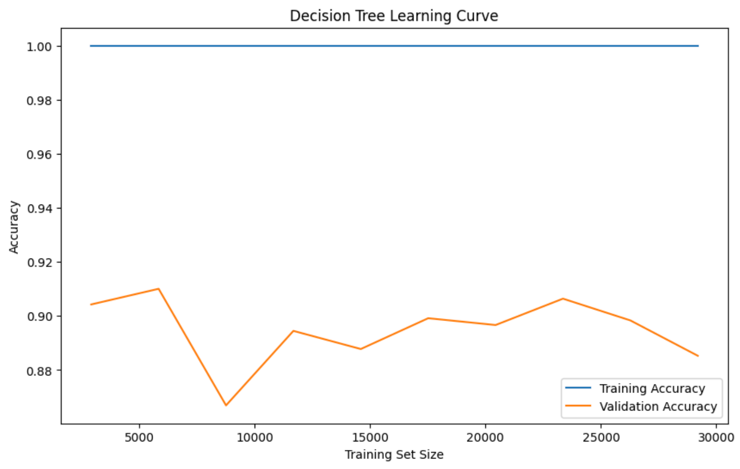
```
# ROC Curve for Random Forest
#plt.subplot(1, 2, 2)
plot_roc_curve(y_test, rf_classifier.predict_proba(X_test))
plt.title("Random Forest ROC Curve")
plt.show()
```

The Random Forest ROC Curve visually represents the discriminatory ability of our model across different probability thresholds. The plot allows us to assess the trade-offs between true positive rate and false positive rate, providing additional insights beyond the metrics in the Classification Report.

Learning Curve:
Decision tree learning curve:

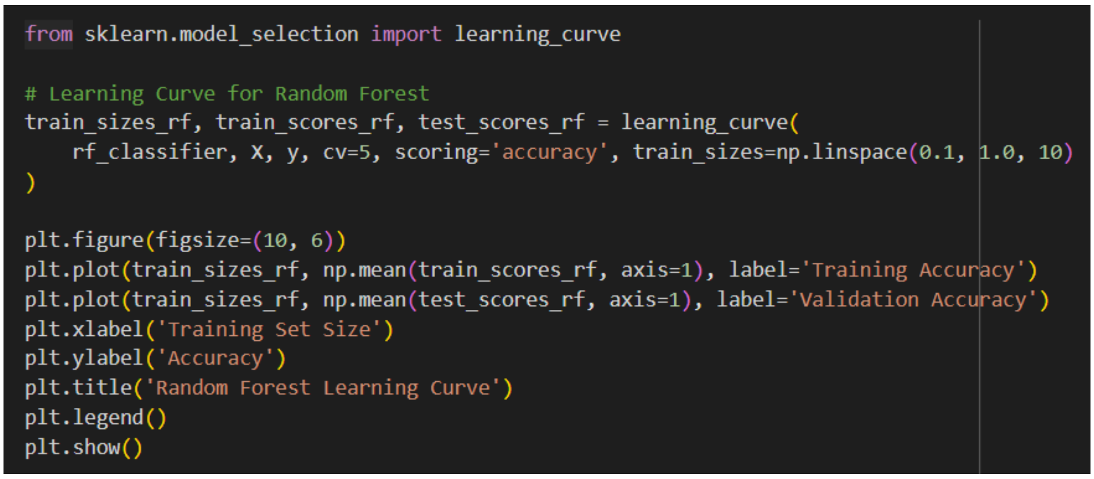


```
import numpy as np
from sklearn.model_selection import learning_curve

# Learning Curve for Decision Tree
train_sizes, train_scores, test_scores = learning_curve(
    dt_classifier, X, y, cv=5, scoring='accuracy', train_sizes=np.linspace(0.1, 1.0, 10)
)

plt.figure(figsize=(10, 6))
plt.plot(train_sizes, np.mean(train_scores, axis=1), label='Training Accuracy')
plt.plot(train_sizes, np.mean(test_scores, axis=1), label='Validation Accuracy')
plt.xlabel('Training Set Size')
plt.ylabel('Accuracy')
plt.title('Decision Tree Learning Curve')
plt.legend()
plt.show()
```

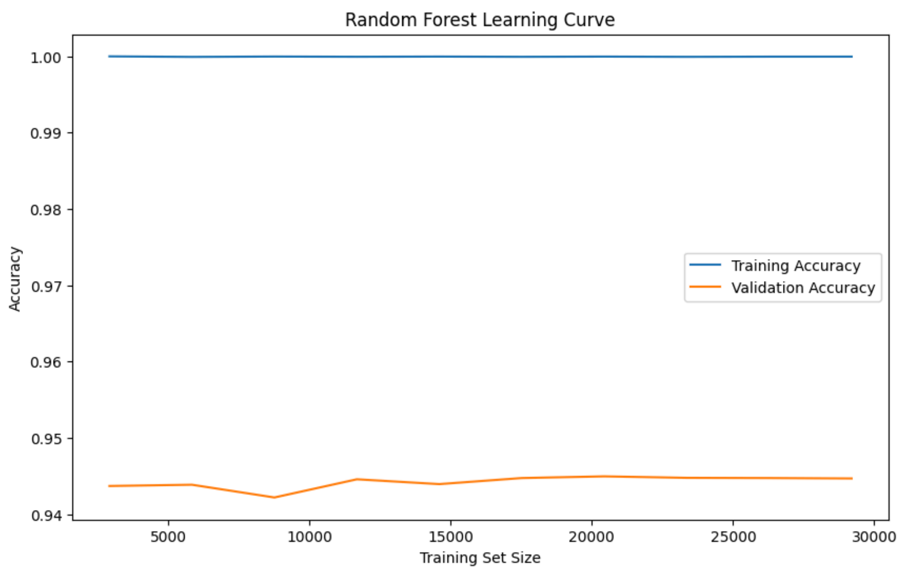


The learning curve illustrates how the model's performance changes as the size of the training set increases. The code specifies a range of training set sizes (np.linspace(0.1, 1.0, 10)) and computes training and validation accuracies for each size through cross-validation (cv=5). The plot shows the average training accuracy and validation accuracy across cross-validation folds for different training set sizes. Interpreting the curve involves assessing whether the model benefits from more data (convergence of training and validation curves) or if there are signs of overfitting or underfitting (gaps or diverging curves). This visualisation provides insights into the model's behaviour and helps make informed decisions about the dataset size and model performance.

Random Forest learning curve:

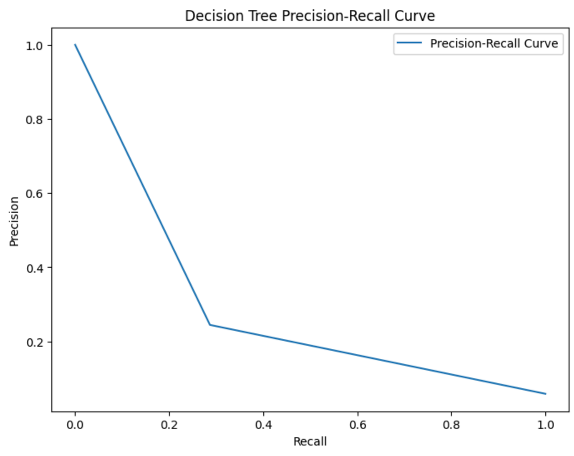


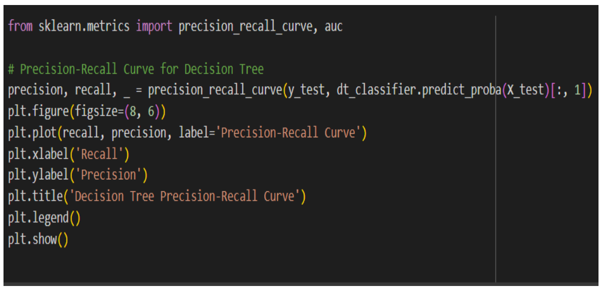
```
from sklearn.model_selection import learning_curve

# Learning Curve for Random Forest
train_sizes_rf, train_scores_rf, test_scores_rf = learning_curve(
    rf_classifier, X, y, cv=5, scoring='accuracy', train_sizes=np.linspace(0.1, 1.0, 10)
)

plt.figure(figsize=(10, 6))
plt.plot(train_sizes_rf, np.mean(train_scores_rf, axis=1), label='Training Accuracy')
plt.plot(train_sizes_rf, np.mean(test_scores_rf, axis=1), label='Validation Accuracy')
plt.xlabel('Training Set Size')
plt.ylabel('Accuracy')
plt.title('Random Forest Learning Curve')
plt.legend()
plt.show()
```



This code generates a learning curve for a Random Forest classifier using scikit-learn's learning_curve function. Similar to the Decision Tree learning curve, it examines how the model's performance changes with varying training set sizes. Interpreting the curve involves assessing whether the Random Forest model benefits from additional data (convergence of training and validation curves) or if there are signs of overfitting or underfitting (gaps or diverging curves). This visualisation provides insights into the model's behaviour and aids in making decisions about dataset size and model performance.

Decision Tree Precision-recall curve:

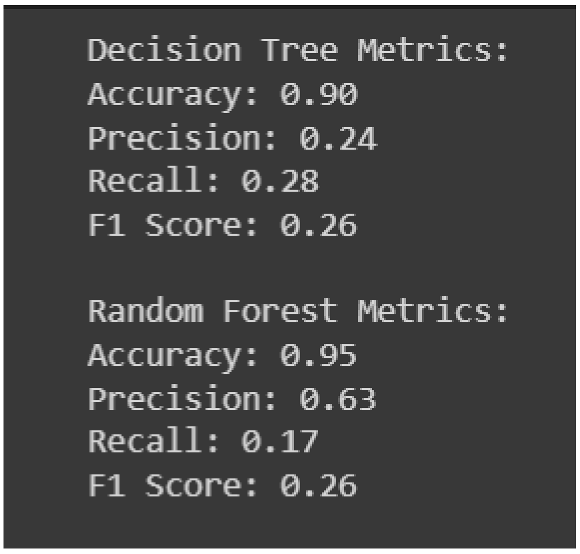




```
from sklearn.metrics import precision_recall_curve, auc

# Precision-Recall Curve for Decision Tree
precision, recall, _ = precision_recall_curve(y_test, dt_classifier.predict_proba(X_test)[:, 1])
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, label='Precision-Recall Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Decision Tree Precision-Recall Curve')
plt.legend()
plt.show()
```

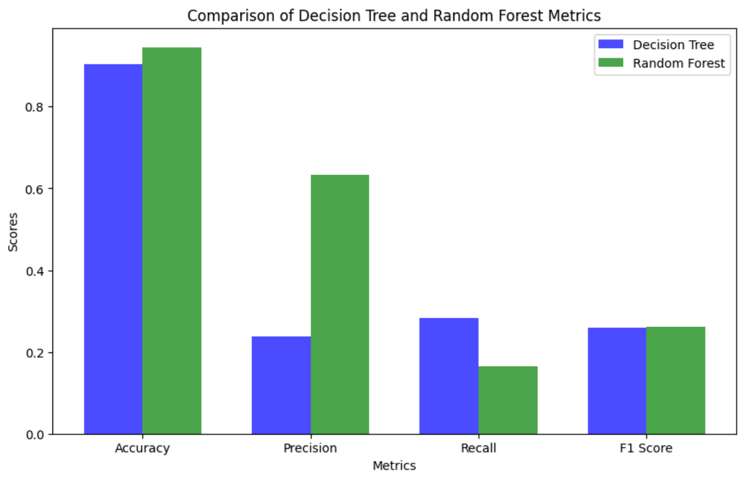
This code snippet generates a Precision-Recall (PR) curve for a Decision Tree classifier using scikit-learn's precision recall curve and Matplotlib for visualisation. The PR curve provides insights into the trade-off between precision and recall for different probability thresholds in a binary classification model.

Classifier Metrics:

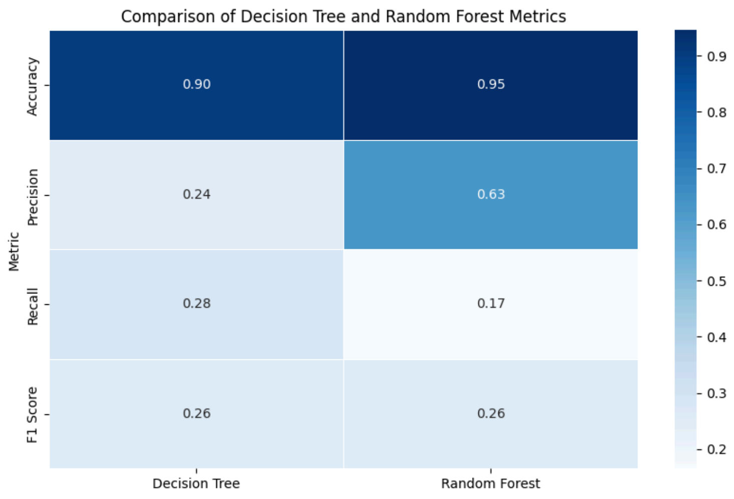


```
Decision Tree Metrics:
Accuracy: 0.90
Precision: 0.24
Recall: 0.28
F1 Score: 0.26

Random Forest Metrics:
Accuracy: 0.95
Precision: 0.63
Recall: 0.17
F1 Score: 0.26
```

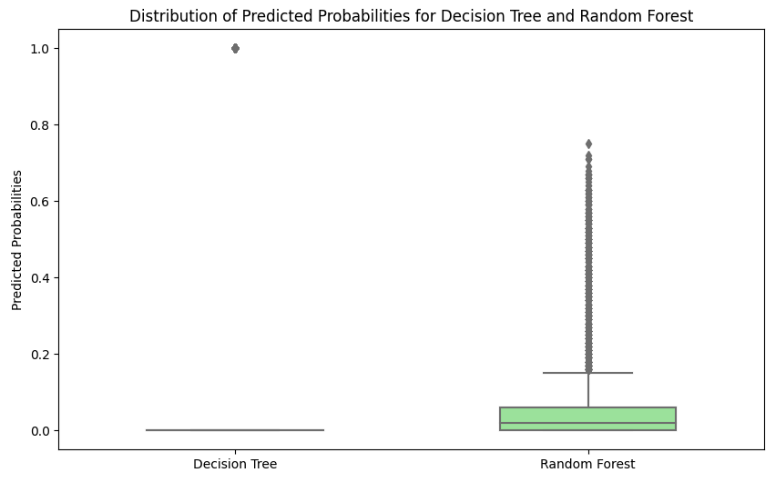
The Random Forest exhibits higher accuracy and precision but lower recall compared to the Decision Tree. The choice between the two models depends on the specific goals of the application. For instance, if capturing all positive instances is crucial, one might prioritise the Decision Tree's higher recall, even though it comes at the cost of lower precision. If a balance between precision and overall correctness is more important, the Random Forest might be preferred.



This bar plot allows for a visual comparison of how the Decision Tree and Random Forest models perform across different metrics. It is a concise and effective way to communicate the relative strengths and weaknesses of the two classifiers.

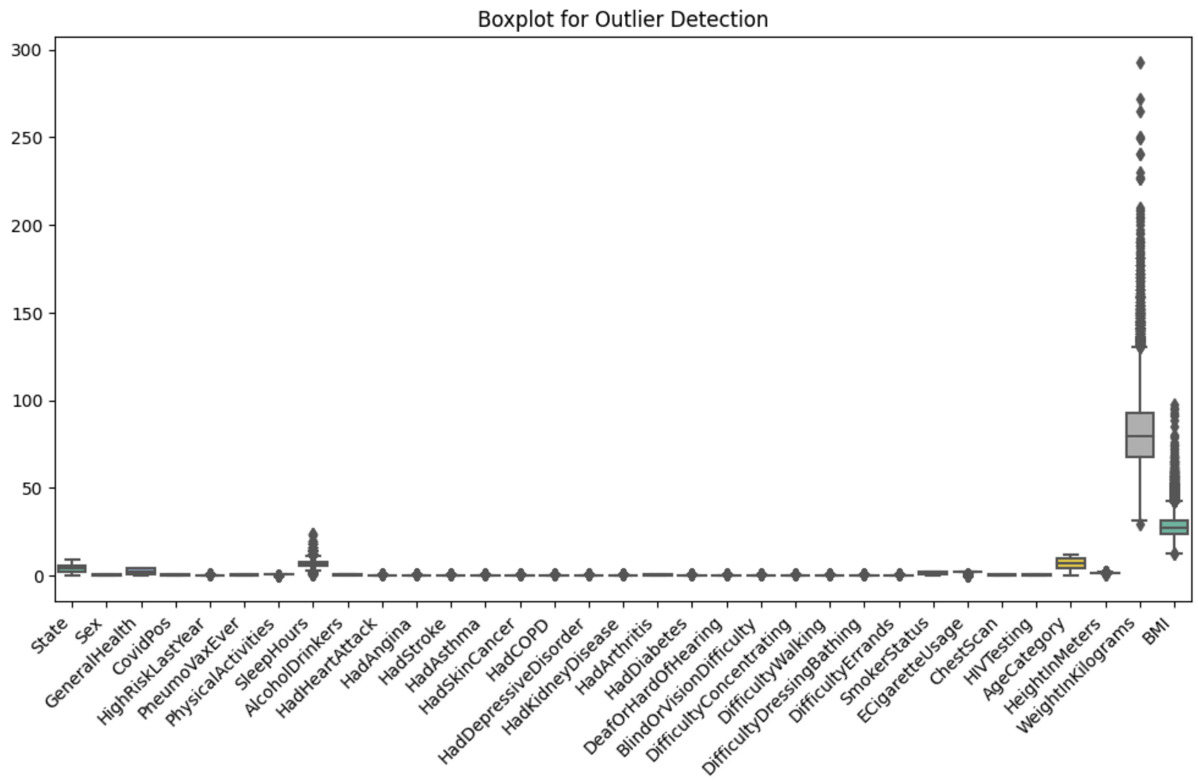


A heatmap is a great way to visualise the relative performance of different models or algorithms across multiple metrics. Darker shades typically indicate higher values, allowing for easy identification of areas where one model outperforms the other. In this case, the heatmap provides a clear visual comparison of how the Decision Tree and Random Forest perform across various metrics.

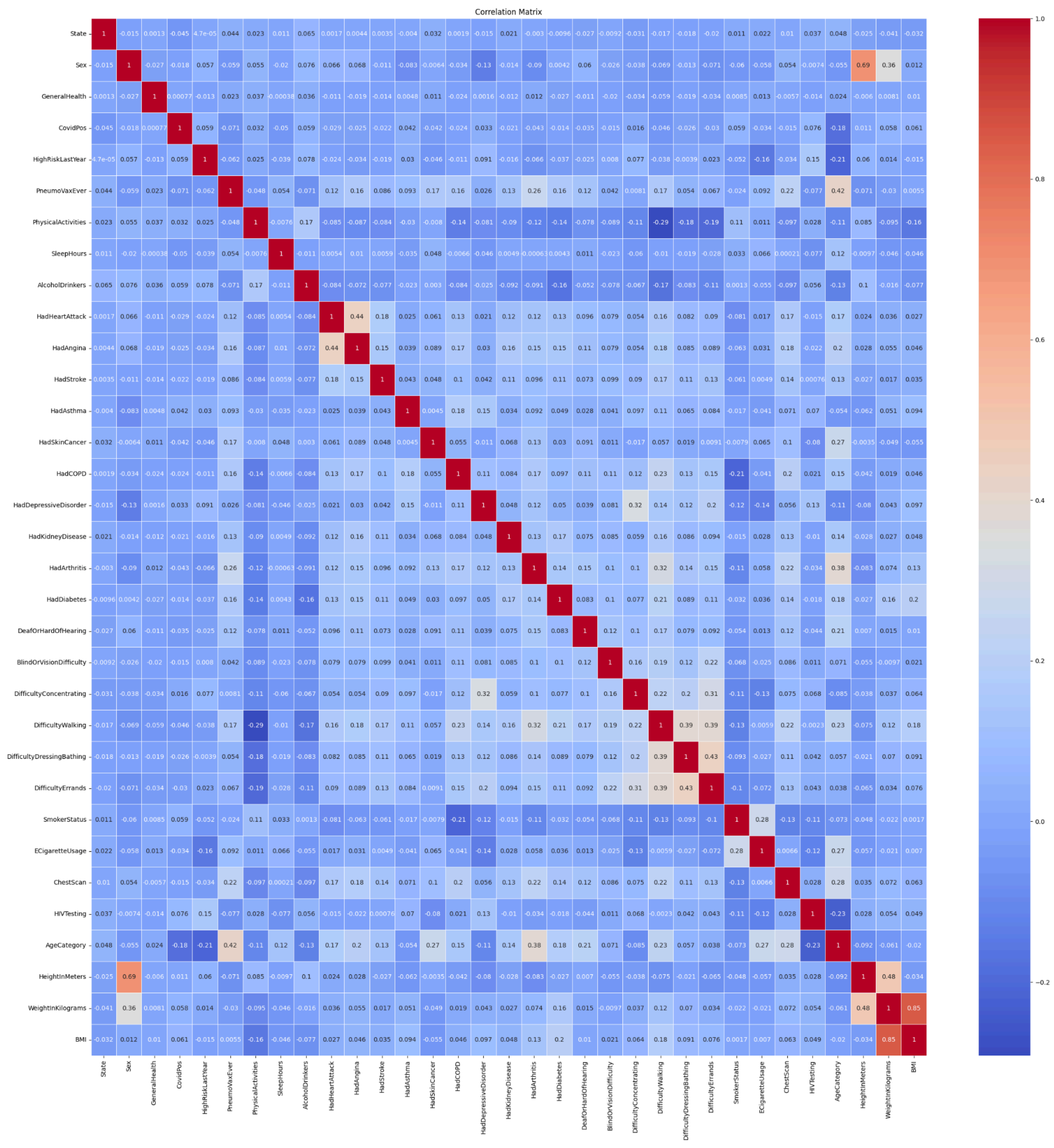


This visualisation allows for a comparative analysis of the distribution of predicted probabilities generated by the Decision Tree and Random Forest classifiers. The boxplot provides insights into the spread, central tendency, and potential outliers in the predicted probability values for each classifier. The use of different colours helps differentiate between the two models.

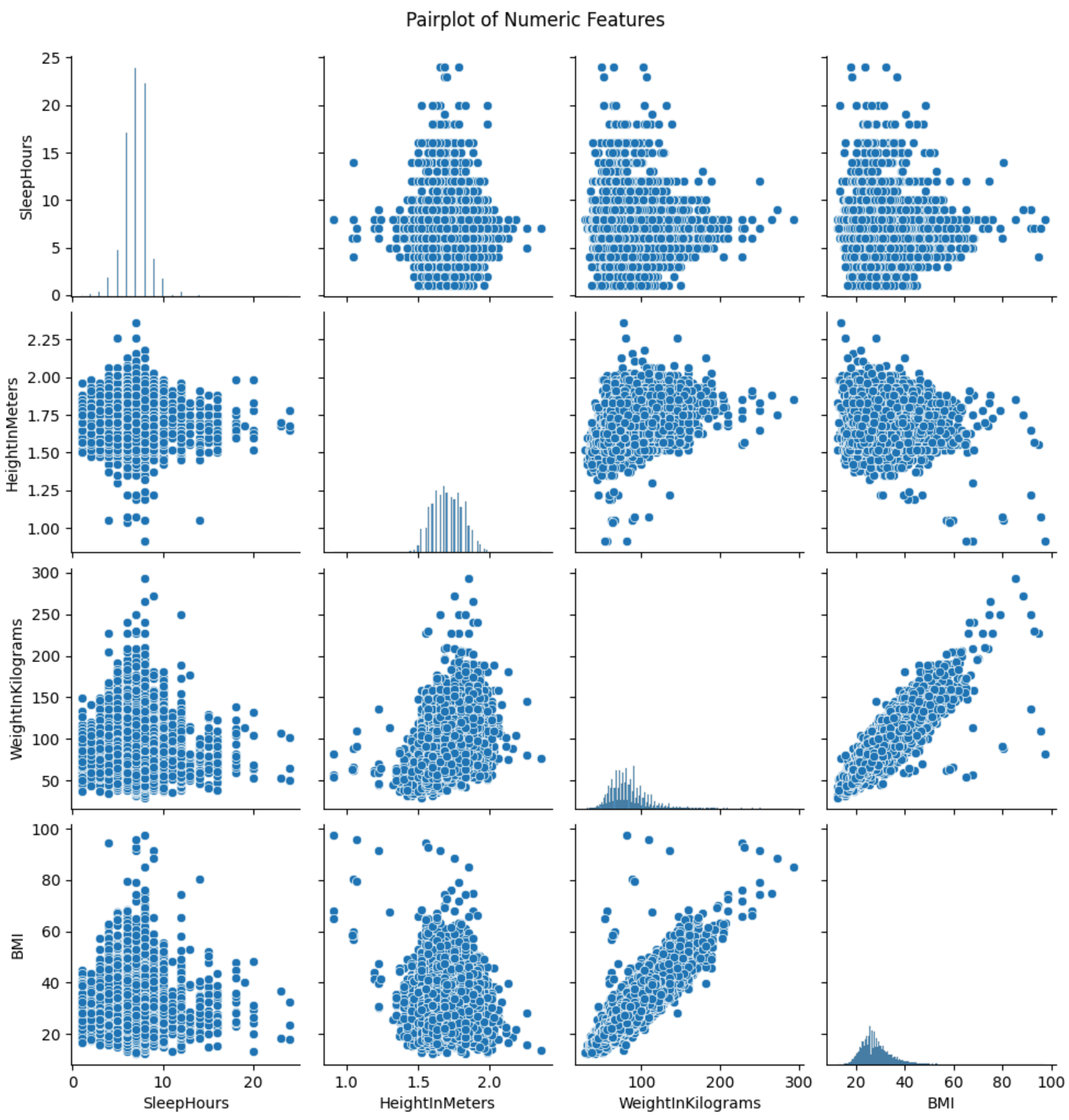
Boxplot:



This type of boxplot is particularly useful for identifying potential outliers in the dataset. Outliers are data points that significantly differ from the majority of the data and can be visually identified as points beyond the "whiskers" of the boxes. The use of different colours in the palette helps distinguish between the features in the dataset. If you notice any points outside the whiskers, those may be considered as potential outliers that warrant further investigation.

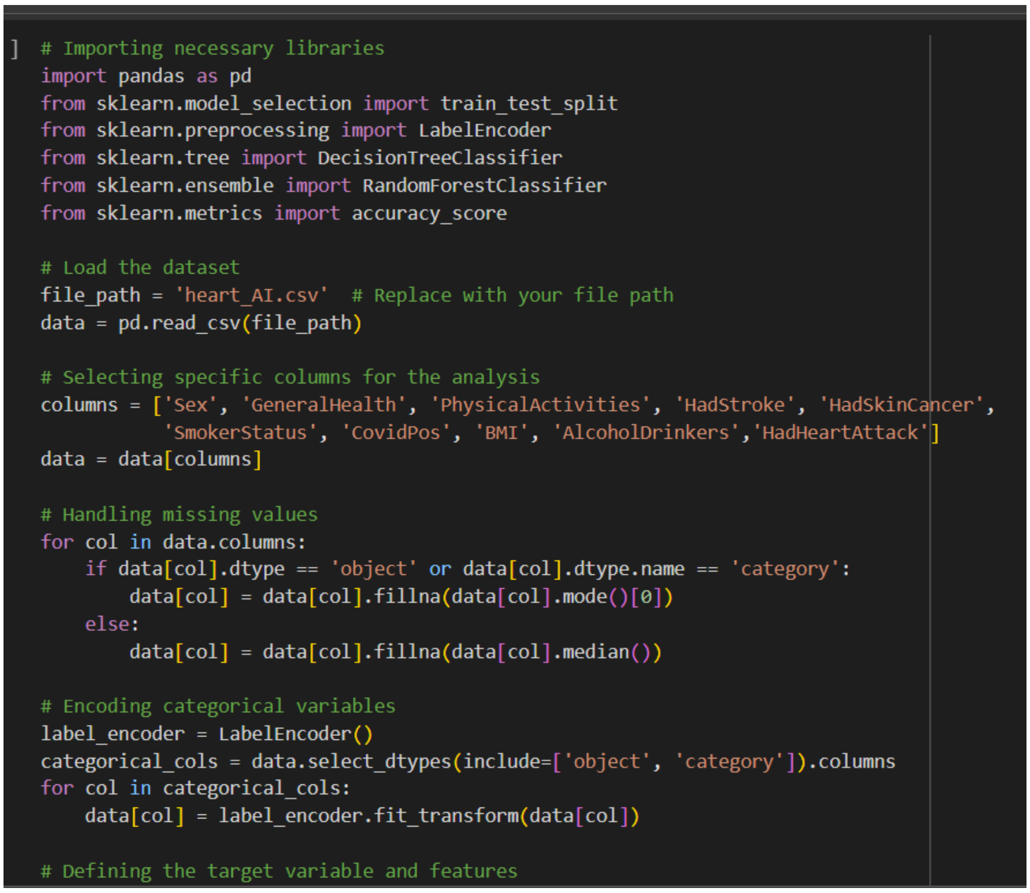


The heatmap visually represents the correlations between pairs of numerical features in the dataset. The colour intensity and the numeric annotations provide insights into the strength and direction of these correlations. Strong positive correlations are often indicated by warmer colours (e.g., red), while strong negative correlations are indicated by cooler colours (e.g., blue). This visualisation is helpful for identifying patterns and relationships between features in the dataset.

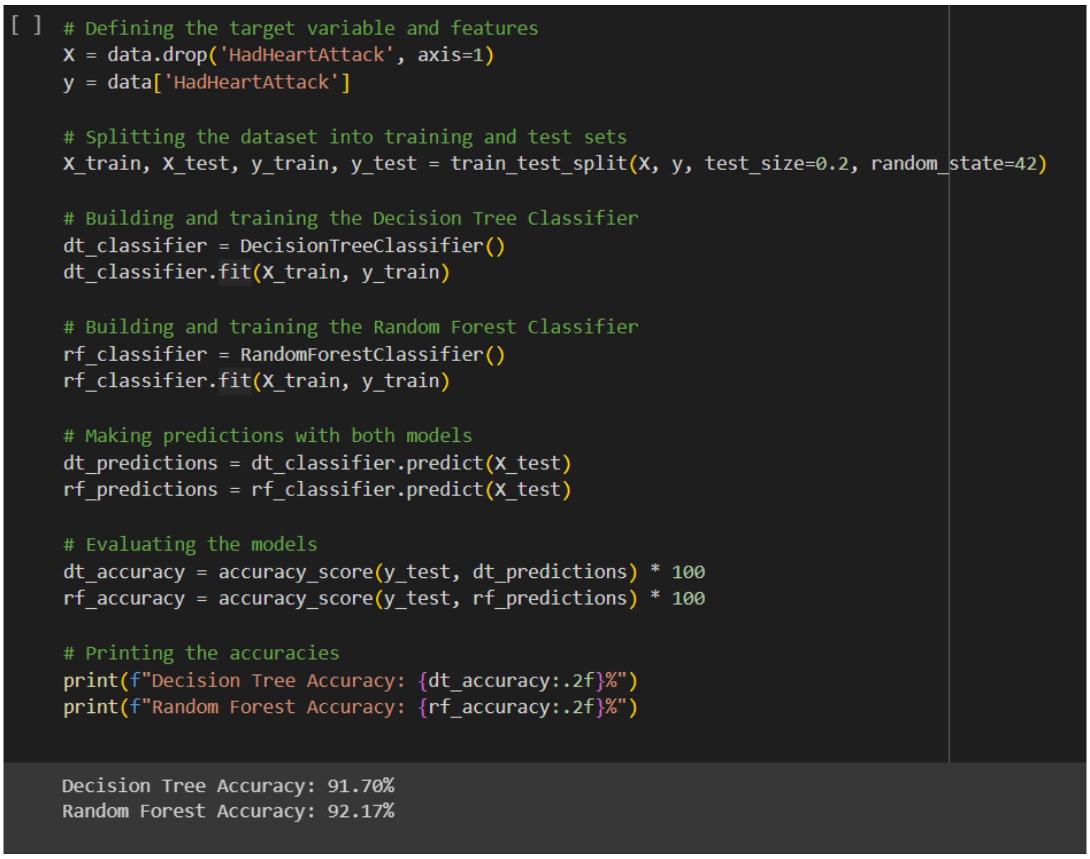


A pairplot is a grid of scatterplots that shows relationships between pairs of variables. Along the diagonal, it displays histograms for each variable. In the off-diagonal plots, each point represents a data point, and you can observe how variables relate to each other. The pairplot is a helpful tool for visualising the distribution of individual features and exploring potential relationships between them. It can also be used to identify patterns and outliers in the data.

Prediction with specific columns:



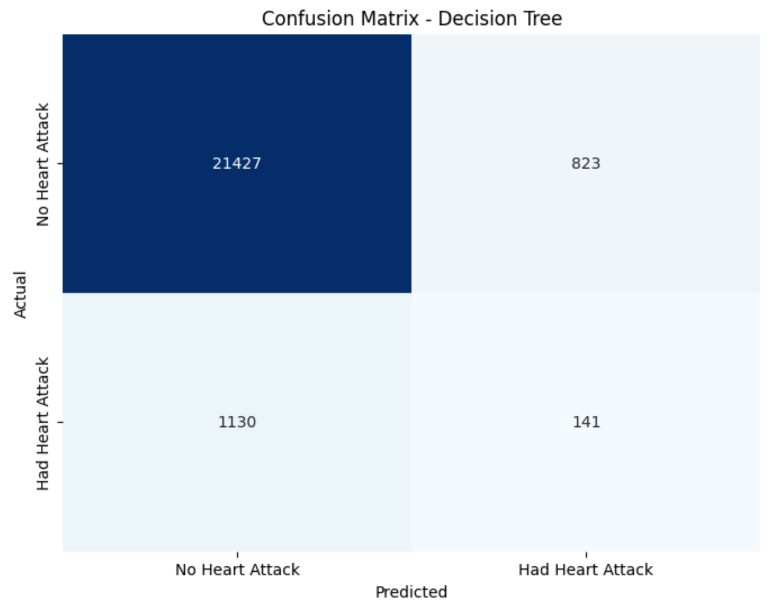
```
]  # Importing necessary libraries
   import pandas as pd
   from sklearn.model_selection import train_test_split
   from sklearn.preprocessing import LabelEncoder
   from sklearn.tree import DecisionTreeClassifier
   from sklearn.ensemble import RandomForestClassifier
   from sklearn.metrics import accuracy_score

   # Load the dataset
   file_path = 'heart_AI.csv'  # Replace with your file path
   data = pd.read_csv(file_path)

   # Selecting specific columns for the analysis
   columns = ['Sex', 'GeneralHealth', 'PhysicalActivities', 'HadStroke', 'HadSkinCancer',
              'SmokerStatus', 'CovidPos', 'BMI', 'AlcoholDrinkers','HadHeartAttack']
   data = data[columns]

   # Handling missing values
   for col in data.columns:
       if data[col].dtype == 'object' or data[col].dtype.name == 'category':
           data[col] = data[col].fillna(data[col].mode()[0])
       else:
           data[col] = data[col].fillna(data[col].median())

   # Encoding categorical variables
   label_encoder = LabelEncoder()
   categorical_cols = data.select_dtypes(include=['object', 'category']).columns
   for col in categorical_cols:
       data[col] = label_encoder.fit_transform(data[col])

   # Defining the target variable and features
```



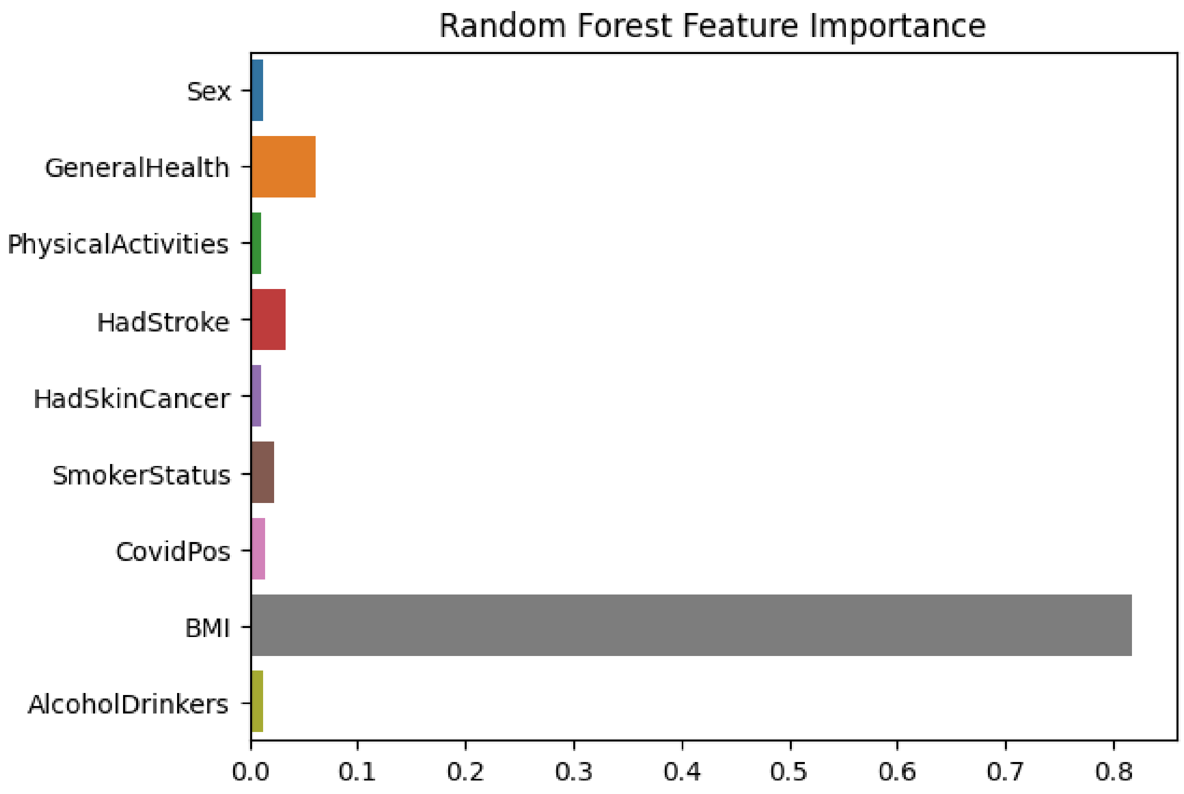
```
[ ]  # Defining the target variable and features
     X = data.drop('HadHeartAttack', axis=1)
     y = data['HadHeartAttack']

     # Splitting the dataset into training and test sets
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

     # Building and training the Decision Tree Classifier
     dt_classifier = DecisionTreeClassifier()
     dt_classifier.fit(X_train, y_train)

     # Building and training the Random Forest Classifier
     rf_classifier = RandomForestClassifier()
     rf_classifier.fit(X_train, y_train)

     # Making predictions with both models
     dt_predictions = dt_classifier.predict(X_test)
     rf_predictions = rf_classifier.predict(X_test)

     # Evaluating the models
     dt_accuracy = accuracy_score(y_test, dt_predictions) * 100
     rf_accuracy = accuracy_score(y_test, rf_predictions) * 100

     # Printing the accuracies
     print(f"Decision Tree Accuracy: {dt_accuracy:.2f}%")
     print(f"Random Forest Accuracy: {rf_accuracy:.2f}%")
```

```
Decision Tree Accuracy: 91.70%
Random Forest Accuracy: 92.17%
```

Confusion Matrix:

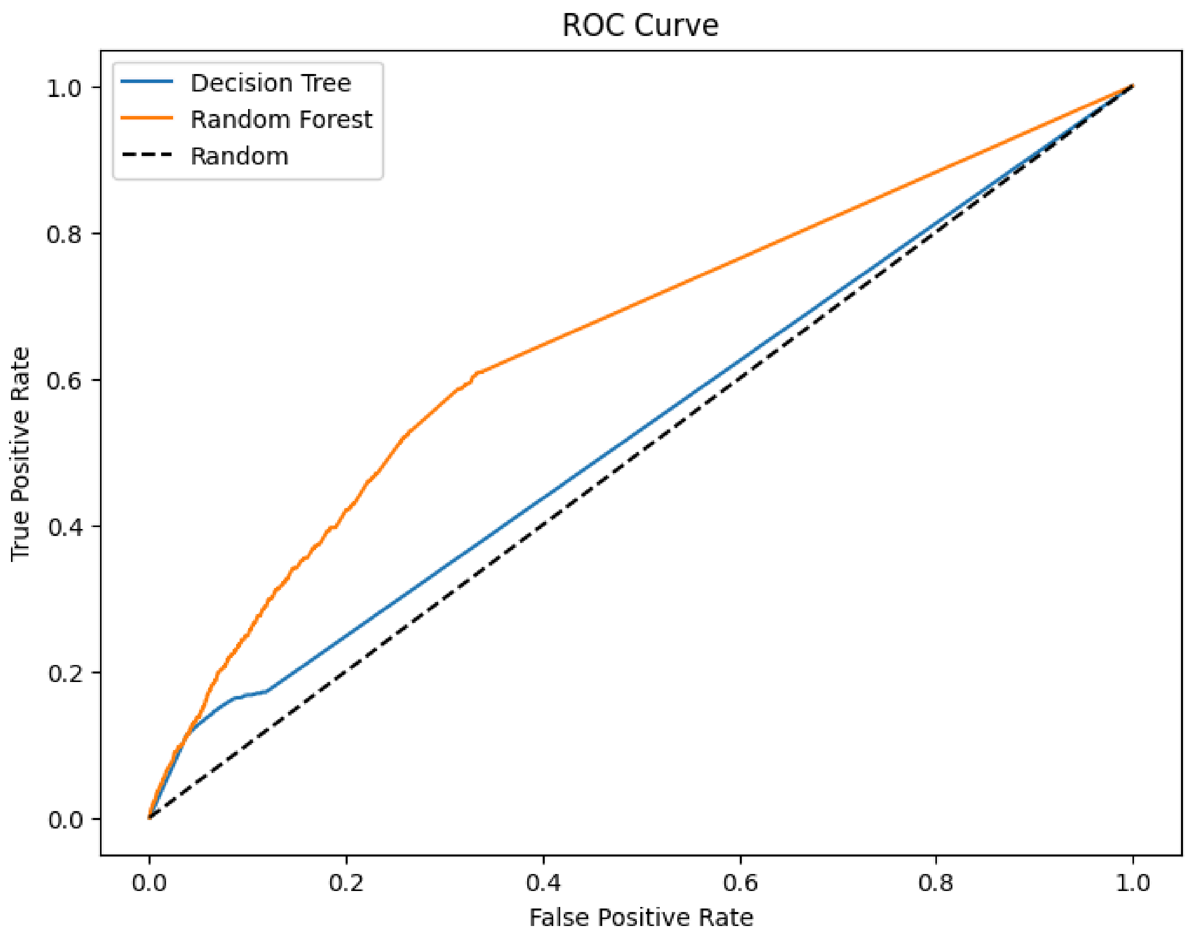


The confusion matrix provides a detailed view of the model's performance by showing the counts of true positive, true negative, false positive, and false negative predictions. The heatmap visualisation makes it easy to identify patterns and errors in the model's predictions.

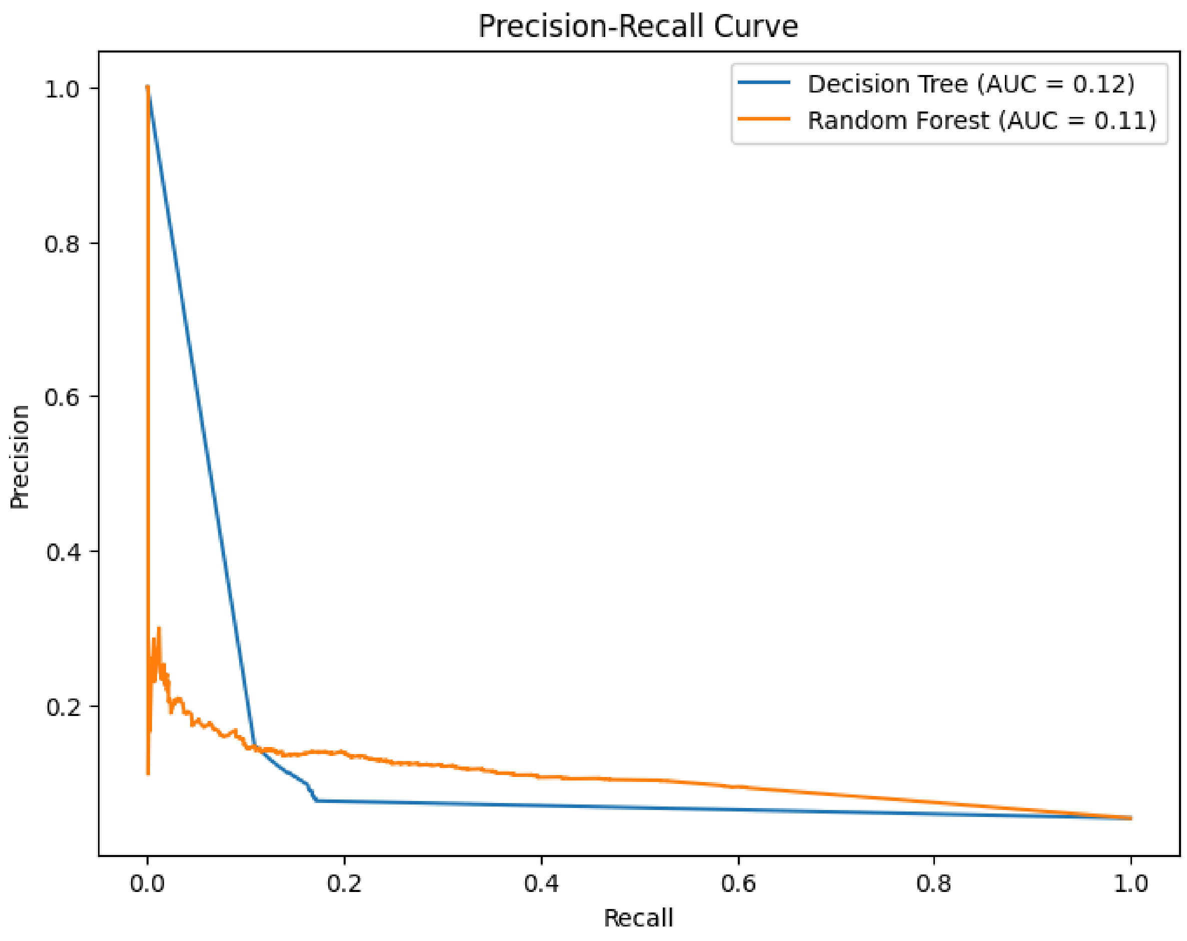


This bar plot provides a visual representation of the importance of each feature in the Random Forest model. Features with higher importance values contribute more to the model's predictions.

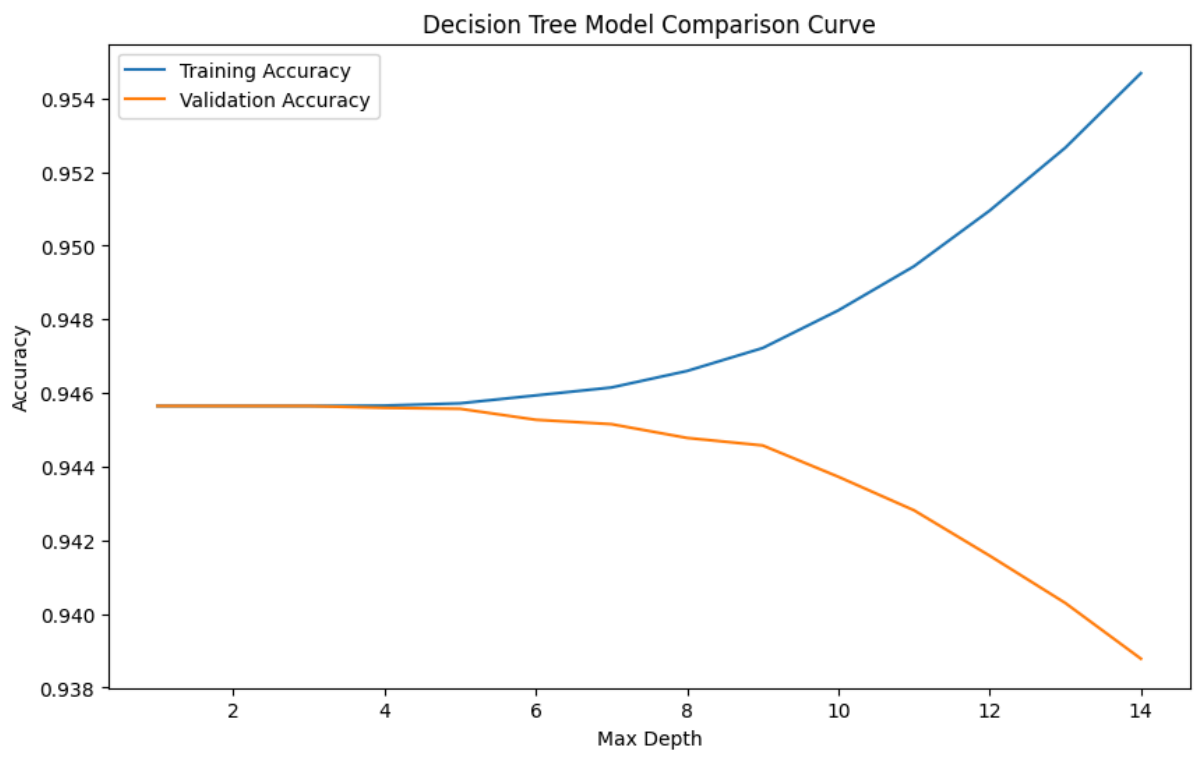
Understanding feature importance can be valuable for feature selection, model interpretation, and gaining insights into the factors that influence the model's decisions.



The ROC curve visually represents the trade-off between the true positive rate and false positive rate for different classification thresholds. A steeper curve and a higher area under the ROC curve (AUC-ROC) generally indicate better model performance. The diagonal line represents random guessing, and a model should aim to be above this line. The legend distinguishes between the ROC curves of the Decision Tree and Random Forest models.



The Precision-Recall curve is particularly useful when dealing with imbalanced datasets. It shows how well the model balances precision (positive predictive value) and recall (sensitivity) for different classification thresholds. A higher AUC indicates better overall performance. The legend differentiates between the curves for the Decision Tree and Random Forest models, and the AUC values in the legend provide a quantitative measure of model performance.

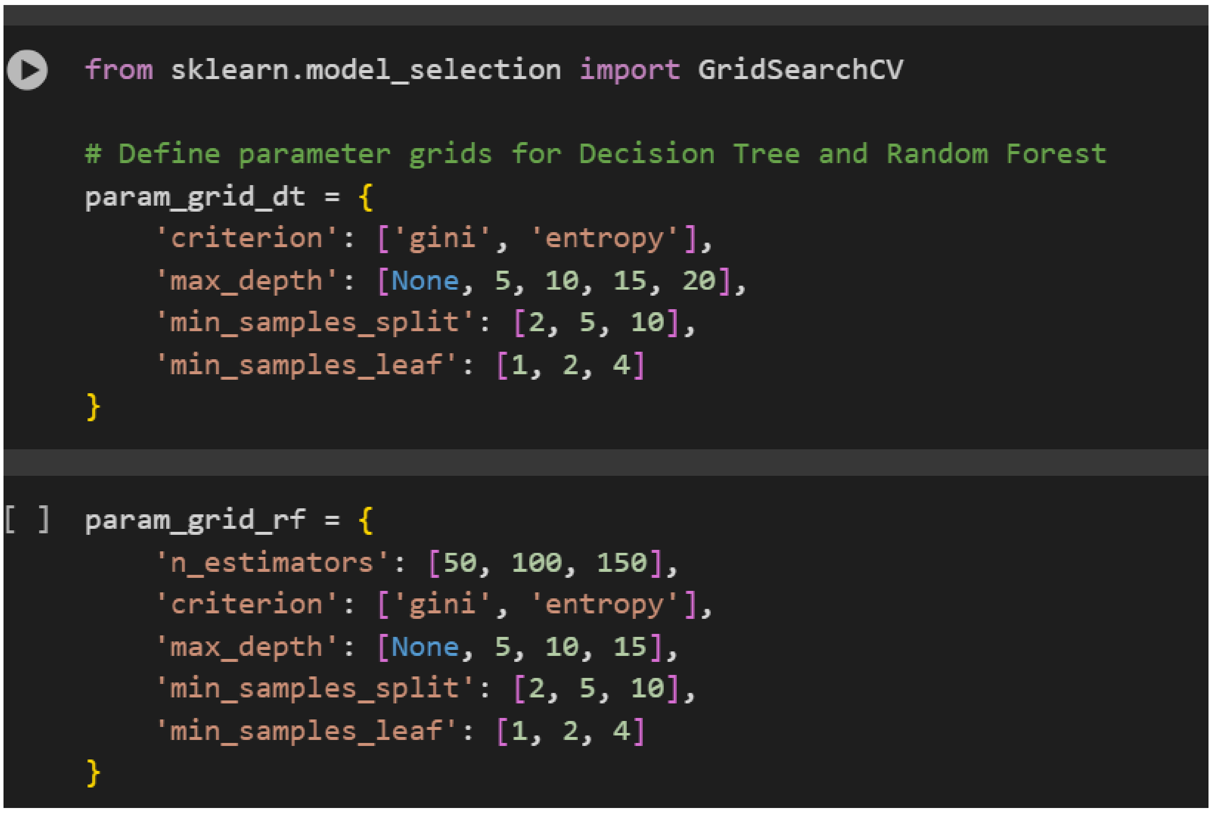


The validation curve helps in understanding how the performance of the model changes with different values of the hyperparameter. It is useful for selecting an appropriate value for the hyperparameter to prevent overfitting or underfitting. In this case, it helps visualise the trade-off between training and validation accuracy as the max depth of the Decision Tree varies.

Model Tuning

The purpose of tuning a model is to ensure that it performs at its best. This process involves adjusting various elements of the model to achieve optimal results. By fine-tuning the model, you can maximise its performance and get the highest rate of performance possible.

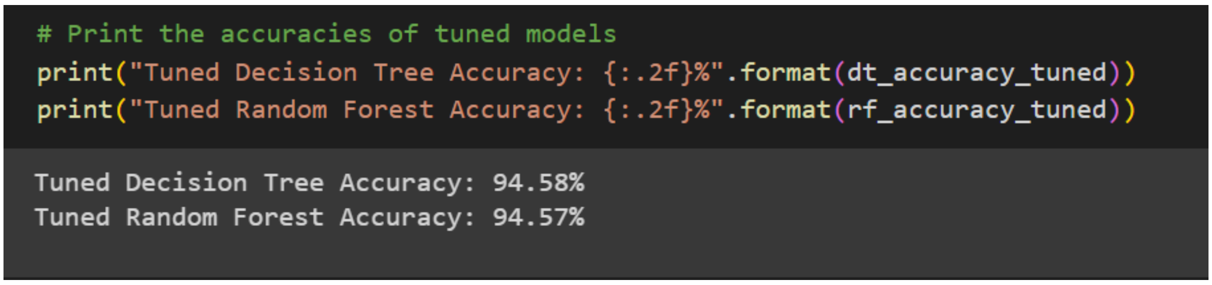
We import GridSearchCV for Model Tuning



```python
from sklearn.model_selection import GridSearchCV

# Define parameter grids for Decision Tree and Random Forest
param_grid_dt = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5, 10, 15, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

```python
[ ]  param_grid_rf = {
         'n_estimators': [50, 100, 150],
         'criterion': ['gini', 'entropy'],
         'max_depth': [None, 5, 10, 15],
         'min_samples_split': [2, 5, 10],
         'min_samples_leaf': [1, 2, 4]
     }
```

Applied model tuning we get more percent accuracy from before

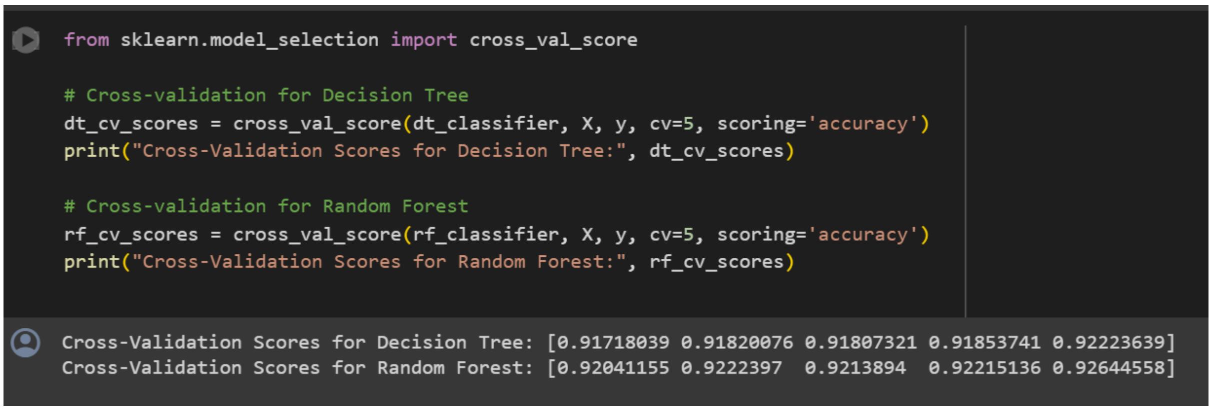


```python
# Print the accuracies of tuned models
print("Tuned Decision Tree Accuracy: {:.2f}%".format(dt_accuracy_tuned))
print("Tuned Random Forest Accuracy: {:.2f}%".format(rf_accuracy_tuned))
```

```
Tuned Decision Tree Accuracy: 94.58%
Tuned Random Forest Accuracy: 94.57%
```

Cross Validation:

Cross-validation is a resampling technique used in machine learning to assess the performance of a predictive model and limit the problems like overfitting or underfitting. It is particularly useful when the dataset is limited, as it helps in maximizing the use of available data for both training and testing.



```
from sklearn.model_selection import cross_val_score

# Cross-validation for Decision Tree
dt_cv_scores = cross_val_score(dt_classifier, X, y, cv=5, scoring='accuracy')
print("Cross-Validation Scores for Decision Tree:", dt_cv_scores)

# Cross-validation for Random Forest
rf_cv_scores = cross_val_score(rf_classifier, X, y, cv=5, scoring='accuracy')
print("Cross-Validation Scores for Random Forest:", rf_cv_scores)
```

```
Cross-Validation Scores for Decision Tree: [0.91718039 0.91820076 0.91807321 0.91853741 0.92223639]
Cross-Validation Scores for Random Forest: [0.92041155 0.9222397  0.9213894  0.92215136 0.92644558]
```

Challenges:

The most complicated part was finding the dataset and choosing the proper training parameters. Some complications were faced while tuning the model. The grid search based on the model was complicated while performing. The generated processing of model tuning took a lot of time that made the work challenging.

Conclusion:

The Heart Attack Prediction project serves as a foundation for developing robust and reliable tools that contribute to proactive healthcare, aiding both medical practitioners and individuals in managing and preventing heart-related risks. The intersection of machine learning and healthcare holds immense promise for improving patient outcomes and fostering a data-driven approach to health management.