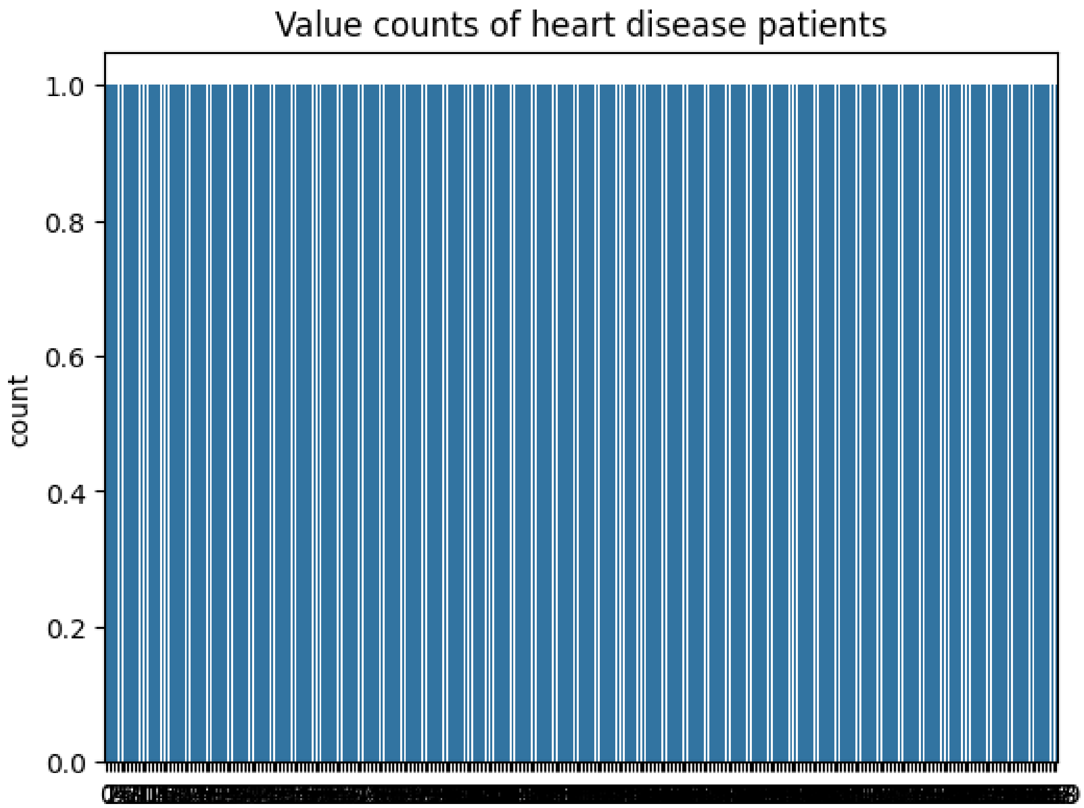
RandomForest

April 8, 2025

[1]:
```python
# Importing the required libraries
import pandas as pd, numpy as np
import matplotlib.pyplot as plt, seaborn as sns
%matplotlib inline
```

[2]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv('heart_v2.csv')
print(df.head())
sns.countplot(df['heart disease'])
plt.title('Value counts of heart disease patients')
plt.show()
```

```
   age  sex   BP  cholestrol  heart disease
0   70    1  130         322              1
1   67    0  115         564              0
2   57    1  124         261              1
3   64    1  128         263              0
4   74    0  120         269              0
```



[3]:
```
# Putting feature variable to X
X = df.drop('heart disease',axis=1)
# Putting response variable to y
y = df['heart disease']
```

[4]:
```
# now lets split the data into train and test
from sklearn.model_selection import train_test_split

# Splitting the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7,␣
↪random_state=42)
X_train.shape, X_test.shape
```

[4]: ((189, 4), (81, 4))

[11]:
```
import time
from sklearn.ensemble import RandomForestClassifier

# Initialize the classifier
classifier_rf = RandomForestClassifier(random_state=42, n_jobs=-1, max_depth=5,
                                       n_estimators=100, oob_score=True)
```

```
# Start the timer
start_time = time.time()

# Fit the model
classifier_rf.fit(X_train, y_train)

# Calculate and print the elapsed time
elapsed_time = time.time() - start_time
print(f"Time taken to fit the model: {elapsed_time} seconds")
```

```
Time taken to fit the model: 0.14409875869750977 seconds
```

[12]:
```
from sklearn.ensemble import RandomForestClassifier

# Initialize the classifier with oob_score enabled
classifier_rf = RandomForestClassifier(random_state=42, n_jobs=-1, max_depth=5,
                                       n_estimators=100, oob_score=True)

# Fit the model
classifier_rf.fit(X_train, y_train)

# Checking the oob score
print(f"OOB Score: {classifier_rf.oob_score_}")
```

```
OOB Score: 0.656084656084656
```

[13]:
```
import time
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Define the Random Forest classifier
rf = RandomForestClassifier(random_state=42, n_jobs=-1)

# Define the parameter grid
params = {
    'max_depth': [2, 3, 5, 10, 20],
    'min_samples_leaf': [5, 10, 20, 50, 100, 200],
    'n_estimators': [10, 25, 30, 50, 100, 200]
}

# Instantiate
```

[14]:
```
import time
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
```

```
# Define the Random Forest classifier
rf = RandomForestClassifier(random_state=42, n_jobs=-1)

# Define the parameter grid
params = {
    'max_depth': [2, 3, 5, 10, 20],
    'min_samples_leaf': [5, 10, 20, 50, 100, 200],
    'n_estimators': [10, 25, 30, 50, 100, 200]
}

# Instantiate the grid search model
grid_search = GridSearchCV(estimator=rf,
                           param_grid=params,
                           cv=4,
                           n_jobs=-1,
                           verbose=1,
                           scoring="accuracy")

# Start the timer
start_time = time.time()

# Fit the grid search model
grid_search.fit(X_train, y_train)

# Calculate and print the elapsed time
elapsed_time = time.time() - start_time
print(f"Time taken for grid search: {elapsed_time:.2f} seconds")

# Ensure the grid search is finished and best_score_ is available
if grid_search.best_score_:
    print(f"Best cross-validation accuracy score: {grid_search.best_score_}")
else:
    print("Best score is not available yet.")

# Optionally, print the best parameters found
print(f"Best parameters: {grid_search.best_params_}")
```

```
Fitting 4 folds for each of 180 candidates, totalling 720 fits
Time taken for grid search: 7.97 seconds
Best cross-validation accuracy score: 0.6985815602836879
Best parameters: {'max_depth': 5, 'min_samples_leaf': 10, 'n_estimators': 10}
```

[15]:
```
import time
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Define the Random Forest classifier
```

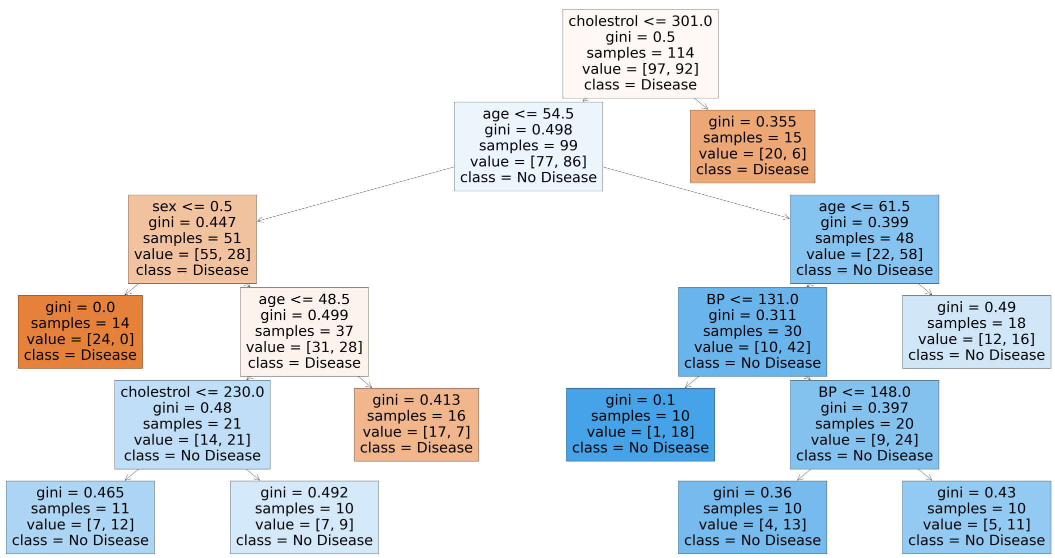
```
rf = RandomForestClassifier(random_state=42, n_jobs=-1)

# Define the parameter grid
params = {
    'max_depth': [2, 3, 5, 10, 20],
    'min_samples_leaf': [5, 10, 20, 50, 100, 200],
    'n_estimators': [10, 25, 30, 50, 100, 200]
}

# Instantiate the grid search model
grid_search = GridSearchCV(estimator=rf,
                           param_grid=params,
                           cv=4,
                           n_jobs=-1,
                           verbose=1,
                           scoring="accuracy")

# Start the timer
start_time = time.time()

# Fit the grid search model
grid_search.fit(X_train, y_train)

# Calculate and print the elapsed time
elapsed_time = time.time() - start_time
print(f"Time taken for grid search: {elapsed_time:.2f} seconds")

# Check if the best estimator is available after the grid search is done
if hasattr(grid_search, 'best_estimator_'):
    rf_best = grid_search.best_estimator_
    print(f"Best model: {rf_best}")
else:
    print("Best estimator not available. There may have been an issue with the␣
 ↪grid search.")
```

```
Fitting 4 folds for each of 180 candidates, totalling 720 fits
Time taken for grid search: 7.37 seconds
Best model: RandomForestClassifier(max_depth=5, min_samples_leaf=10,
n_estimators=10,
                       n_jobs=-1, random_state=42)
```

[16]:
```
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# Ensure grid search has been run and rf_best is defined
# Assign the best estimator after the grid search
rf_best = grid_search.best_estimator_
```

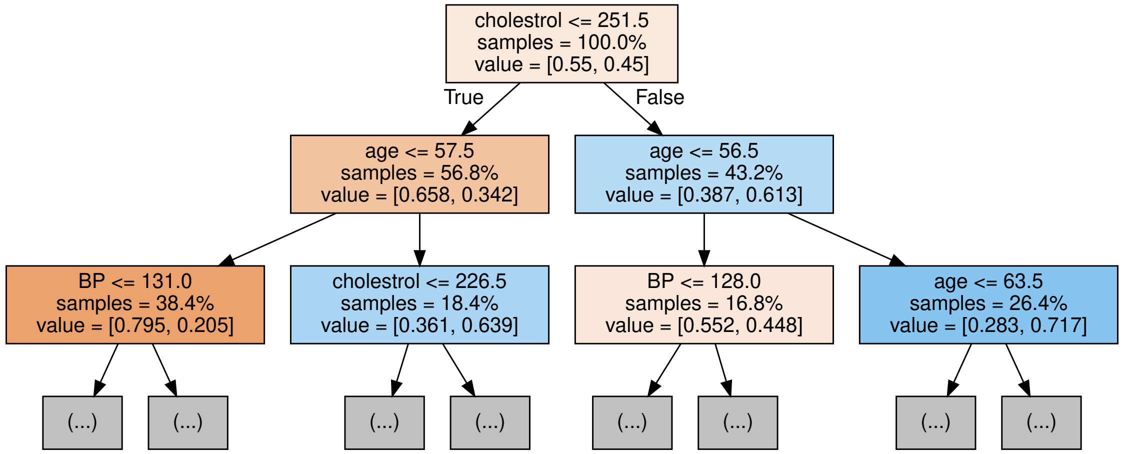
```python
# Make sure the model has enough estimators (trees)
print(f"Number of trees in the best model: {len(rf_best.estimators_)}")

# Plotting one of the decision trees from the forest
plt.figure(figsize=(80, 40))
plot_tree(rf_best.estimators_[5], feature_names=X.columns,␣
↪class_names=['Disease', 'No Disease'], filled=True)
plt.show()
```

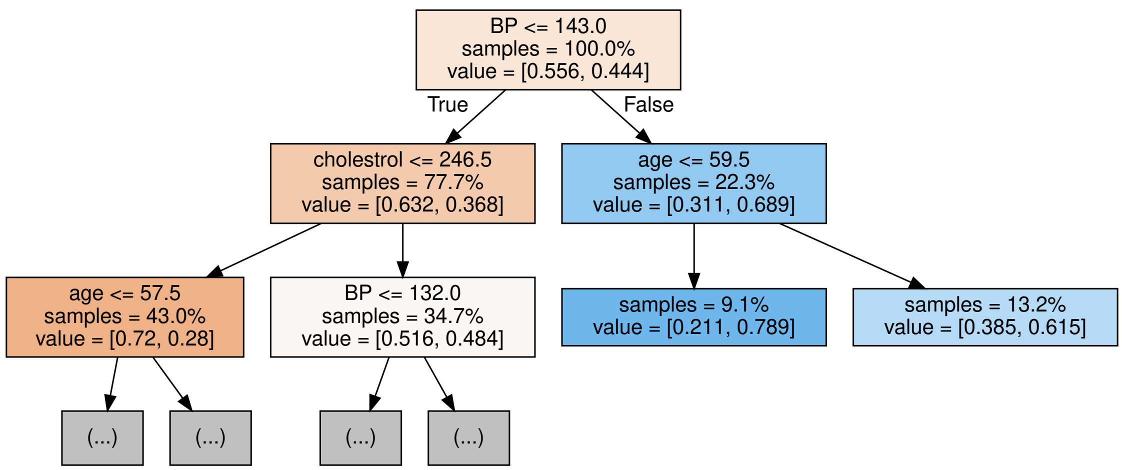
```
Number of trees in the best model: 10
```

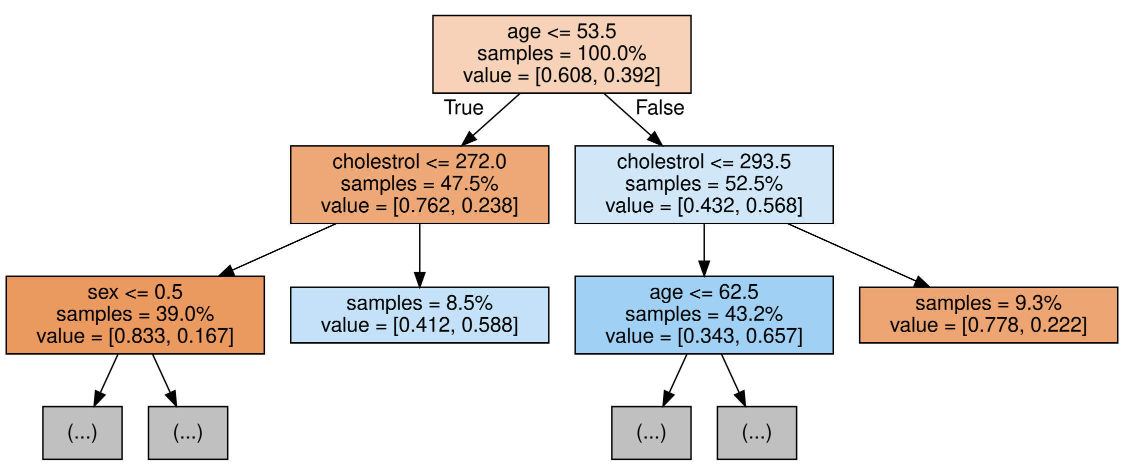


[18]:
```python
from sklearn.tree import export_graphviz
import graphviz

# Ensure the RandomForestClassifier is fitted first
rf_best.fit(X_train, y_train)  # If you're using rf_best from grid search

# Export the first three decision trees from the forest
for i in range(3):
    tree = rf_best.estimators_[i]
    dot_data = export_graphviz(tree,
                               feature_names=X_train.columns,
                               filled=True,
                               max_depth=2,
                               impurity=False,
                               proportion=True)
    graph = graphviz.Source(dot_data)
```

```
display(graph)
```







[20]:
```
from scipy.stats import randint
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV

# Define the parameter distribution
param_dist = {'n_estimators': randint(50, 500),
              'max_depth': randint(1, 20)}

# Create a random forest classifier
rf = RandomForestClassifier()

# Use random search to find the best hyperparameters
rand_search = RandomizedSearchCV(rf,
                                 param_distributions=param_dist,
                                 n_iter=5,
                                 cv=5,
                                 n_jobs=-1,  # Use multiple cores for faster␣
↪computation
                                 verbose=1)

# Fit the random search object to the data
rand_search.fit(X_train, y_train)

# You can now access the best parameters found by RandomizedSearchCV
print(f"Best parameters: {rand_search.best_params_}")
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
Best parameters: {'max_depth': 3, 'n_estimators': 397}
```

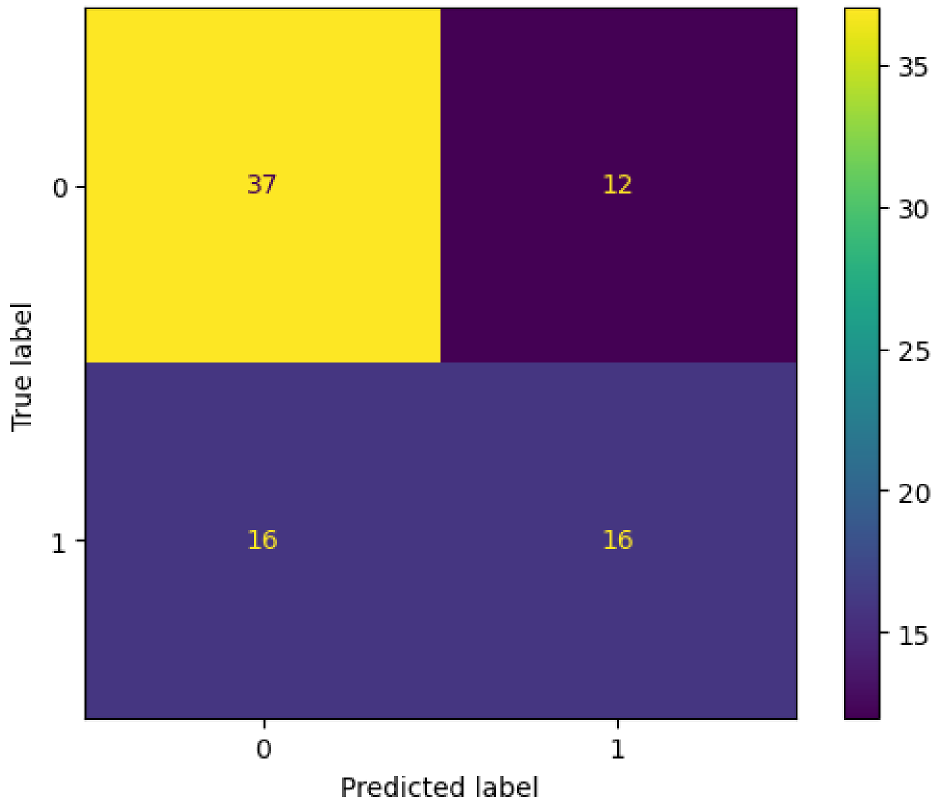
[21]:
```
# Create a variable for the best model
best_rf = rand_search.best_estimator_

# Print the best hyperparameters
print('Best hyperparameters:',  rand_search.best_params_)
```

```
Best hyperparameters: {'max_depth': 3, 'n_estimators': 397}
```

[23]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Generate predictions with the best model
y_pred = best_rf.predict(X_test)

# Create the confusion matrix
cm = confusion_matrix(y_test, y_pred)
```

```python
# Display the confusion matrix
ConfusionMatrixDisplay(confusion_matrix=cm).plot()
```

[23]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f0fc2f5ff70>

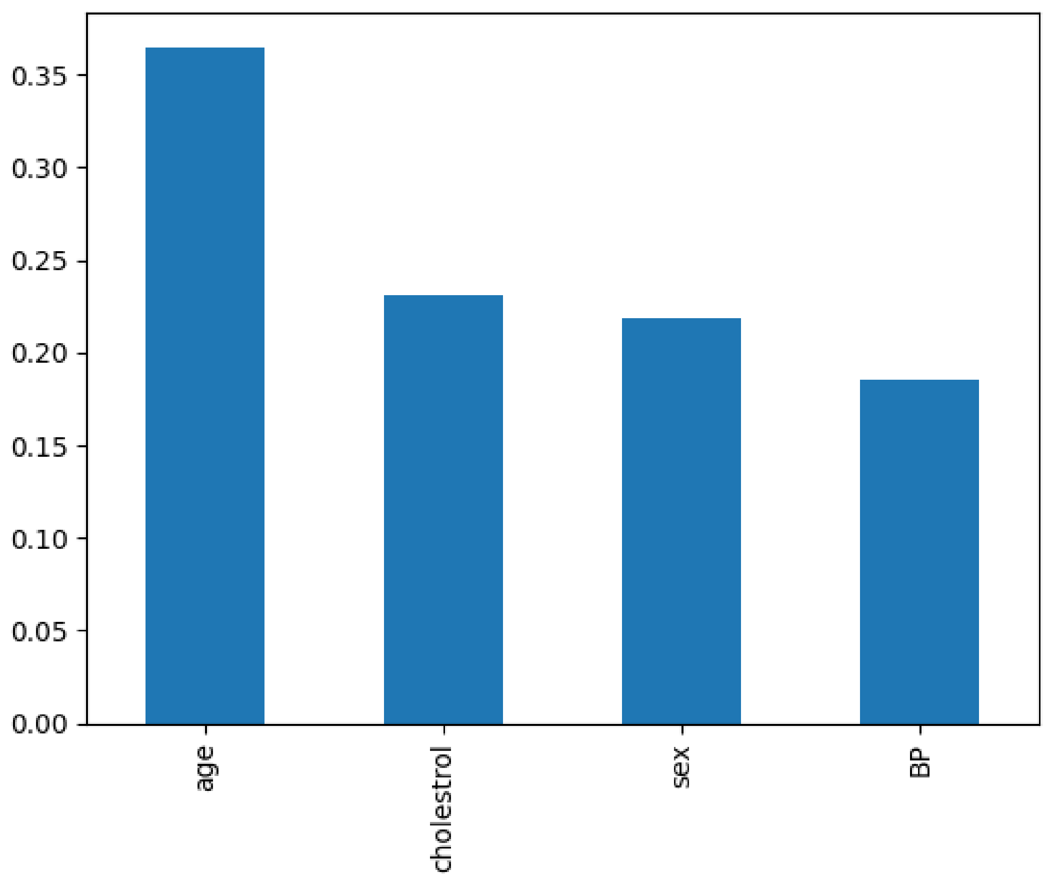


[25]:
```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Define the KNN classifier
knn = KNeighborsClassifier(n_neighbors=5)

# Train the KNN model (assuming you have X_train and y_train)
knn.fit(X_train, y_train)

# Generate predictions with the trained KNN model
y_pred = knn.predict(X_test)

# Calculate metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
```

```
recall = recall_score(y_test, y_pred)

# Print the metrics
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
```

```
Accuracy: 0.654320987654321
Precision: 0.5526315789473685
Recall: 0.65625
```

[26]:
```
# Create a series containing feature importances from the model and feature␣
↪names from the training data
feature_importances = pd.Series(best_rf.feature_importances_, index=X_train.
↪columns).sort_values(ascending=False)

# Plot a simple bar chart
feature_importances.plot.bar();
```



```
[ ]:
```