Método utilizado - Decison tree methods

In [1]:
```python
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import graphviz

# Configuration of the default size of figures generated by graphs
%matplotlib inline
plt.rcParams['figure.figsize'] = [16, 8]

# Import modules
from sklearn.model_selection import train_test_split, KFold, cross_val_predict, StratifiedKFold
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_graphviz
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import confusion_matrix, mean_squared_error, auc, roc_curve , f1_score
```

In [2]:
```python
# Import data
df = pd.read_csv("diabetes_binary_5050split_health_indicators_BRFSS2021.csv")
df.head()
```

Out[2]:

	Diabetes_binary	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits	...	AnyHealthcare	NoDocbcCost	GenHlth	MentHlth	PhysHlth	DiffWalk	Sex	Age	Educa
0	0.0	1	0.0	1	33.0	0.0	0.0	0.0	1	1	...	1	0.0	2.0	15.0	0.0	1.0	1	7	
1	0.0	0	1.0	1	27.0	1.0	0.0	0.0	1	0	...	1	0.0	2.0	1.0	2.0	0.0	1	7	
2	0.0	0	1.0	1	26.0	1.0	0.0	0.0	0	0	...	1	0.0	3.0	0.0	30.0	0.0	1	13	
3	0.0	0	0.0	1	19.0	1.0	0.0	0.0	1	1	...	1	0.0	3.0	0.0	0.0	0.0	0	11	
4	0.0	1	0.0	1	37.0	0.0	0.0	0.0	1	1	...	1	0.0	2.0	0.0	0.0	0.0	0	5	

5 rows × 22 columns

In [3]:
```python
# Split the dataframe into the output y and the input X and into train and test with a train size of 0.8
X = df.drop(['Diabetes_binary'], axis=1)
y = df.Diabetes_binary
seed = 0
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random_state=seed)
```

In [4]:
```python
# Application of the decision tree method
clf_dt = DecisionTreeClassifier(max_depth=6)
clf_dt.fit(X_train, y_train)
```

Out[4]:

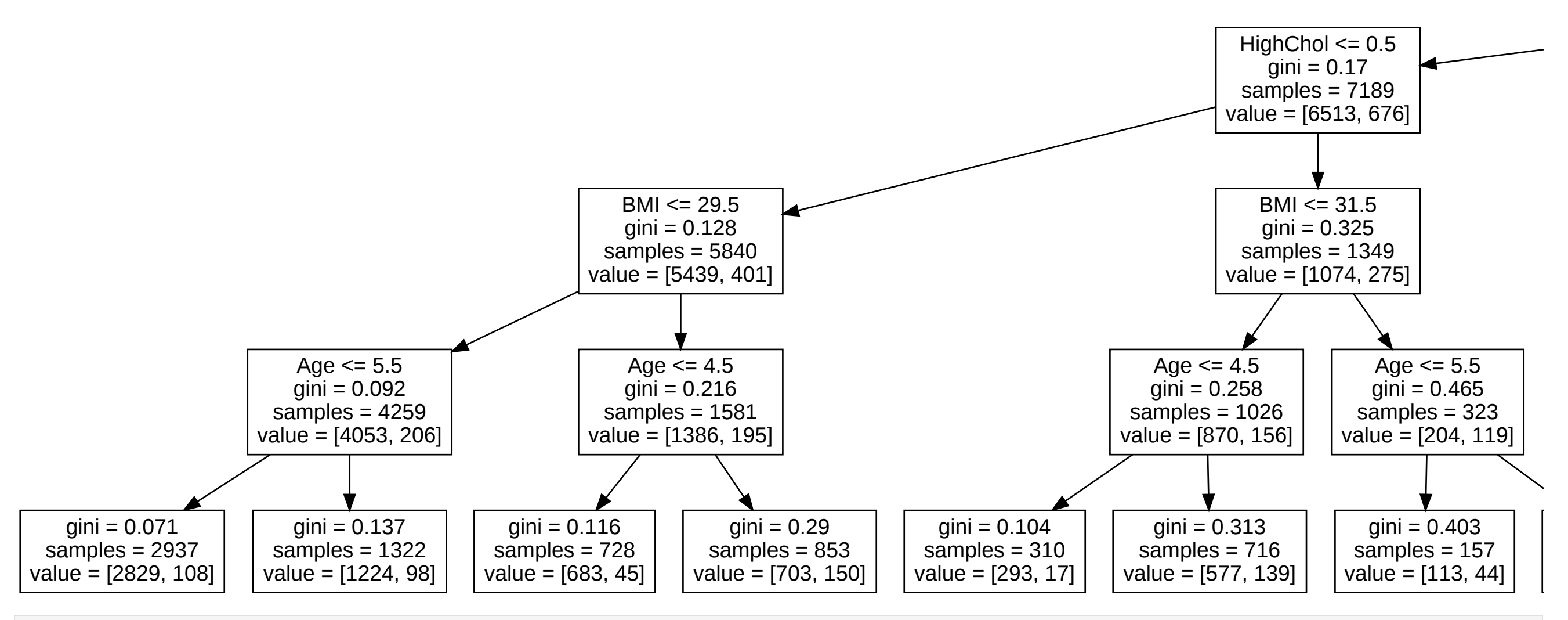
```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=6)
```

In [5]:
```python
# Predict the target variable using the Decision Tree classifier
y_pred = clf_dt.predict(X_test)
# Calculate the F1 score between predicted and actual values
f1 = f1_score(y_test, y_pred)
print('F1 Score = ' + str(f1))
```

```
F1 Score = 0.7422987040577863
```

In [6]:
```python
# Visualization of the decision tree and display the decision tree graph using Graphviz
export_graphviz(clf_dt, out_file="diabetes_tree.dot", feature_names=X_train.columns)
with open("diabetes_tree.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```

Out[6]:



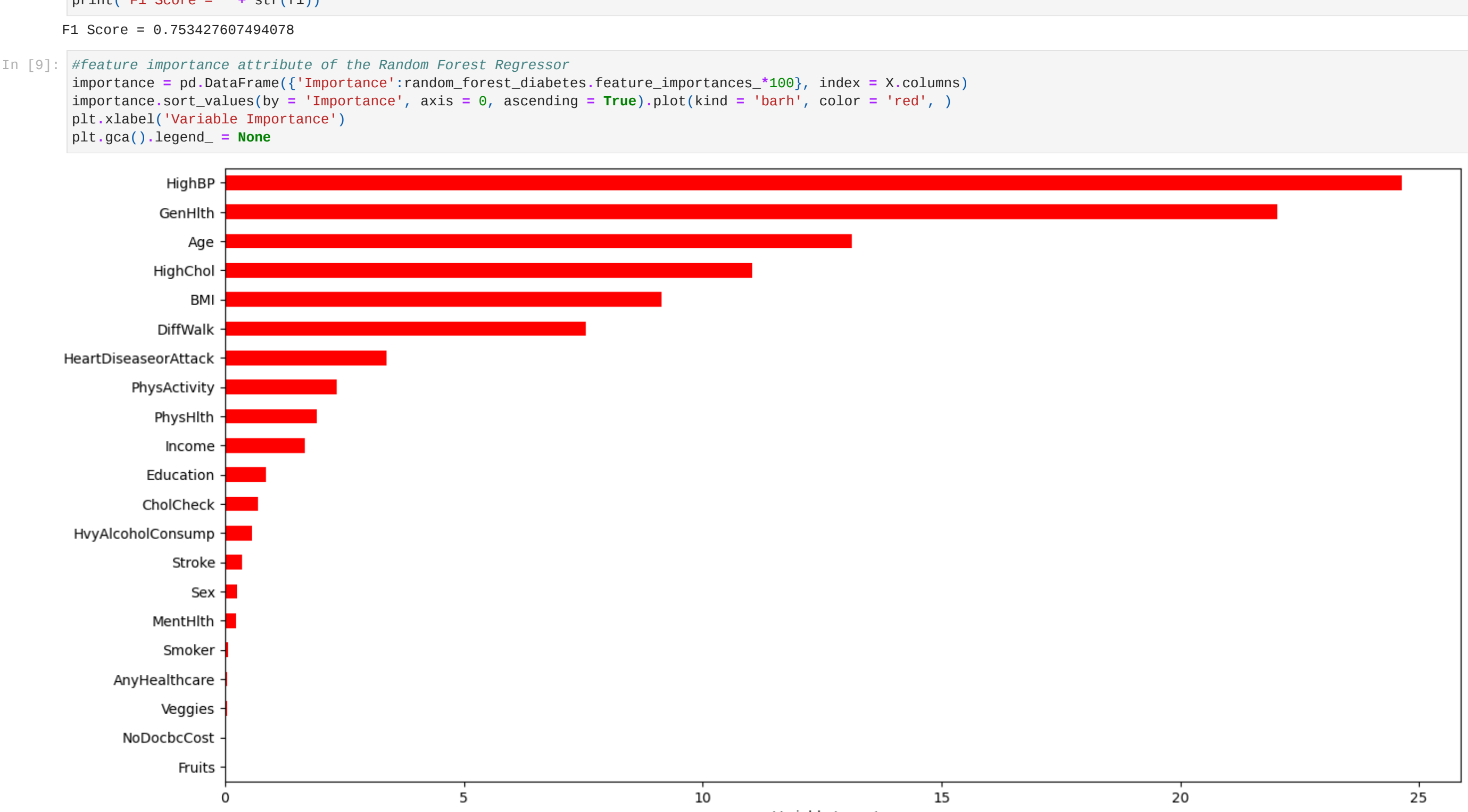
In [7]:
```python
# Confusion Matrix
y_pred = clf_dt.predict(X_test)
cm = pd.DataFrame(confusion_matrix(y_test, y_pred).T, index=['Negative', 'Positive'], columns=['Negative', 'Positive'])
print(cm)
```

```
          Negative  Positive
Negative      4548      1539
Positive      2100      5241
```

In [8]:
```python
# Application of random forests
random_forest_diabetes = RandomForestClassifier(n_estimators=100, max_depth=6 , random_state = seed)
random_forest_diabetes.fit(X_train, y_train)
# Predict the target variable using random forests
y_pred = random_forest_diabetes.predict(X_test)
# Calculate the F1 score between predicted and actual values
f1 = f1_score(y_test, y_pred)
print('F1 Score = ' + str(f1))
```

```
F1 Score = 0.753427607494078
```

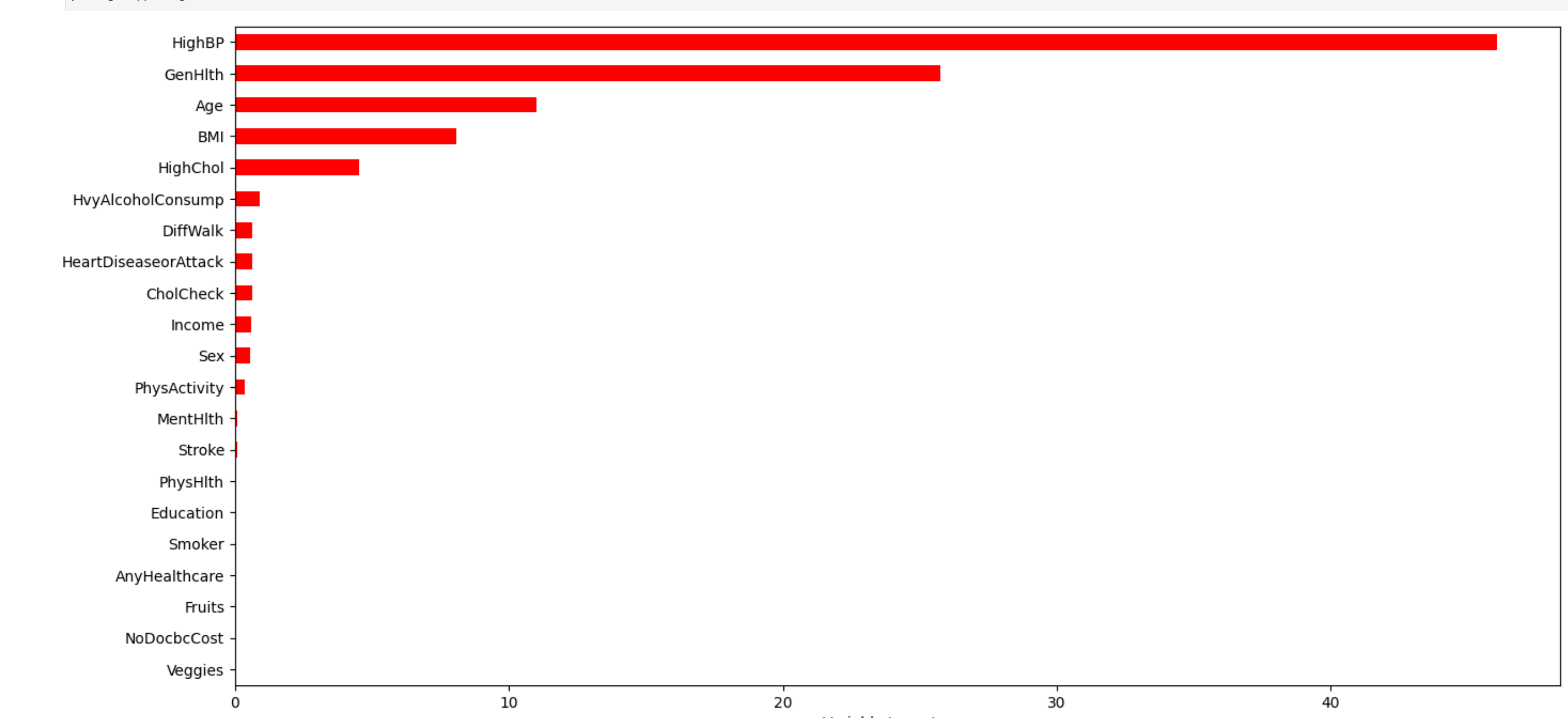
In [9]:
```python
#feature importance attribute of the Random Forest Regressor
importance = pd.DataFrame({'Importance':random_forest_diabetes.feature_importances_*100}, index = X.columns)
importance.sort_values(by = 'Importance', axis = 0, ascending = True).plot(kind = 'barh', color = 'red', )
plt.xlabel('Variable Importance')
plt.gca().legend_ = None
```



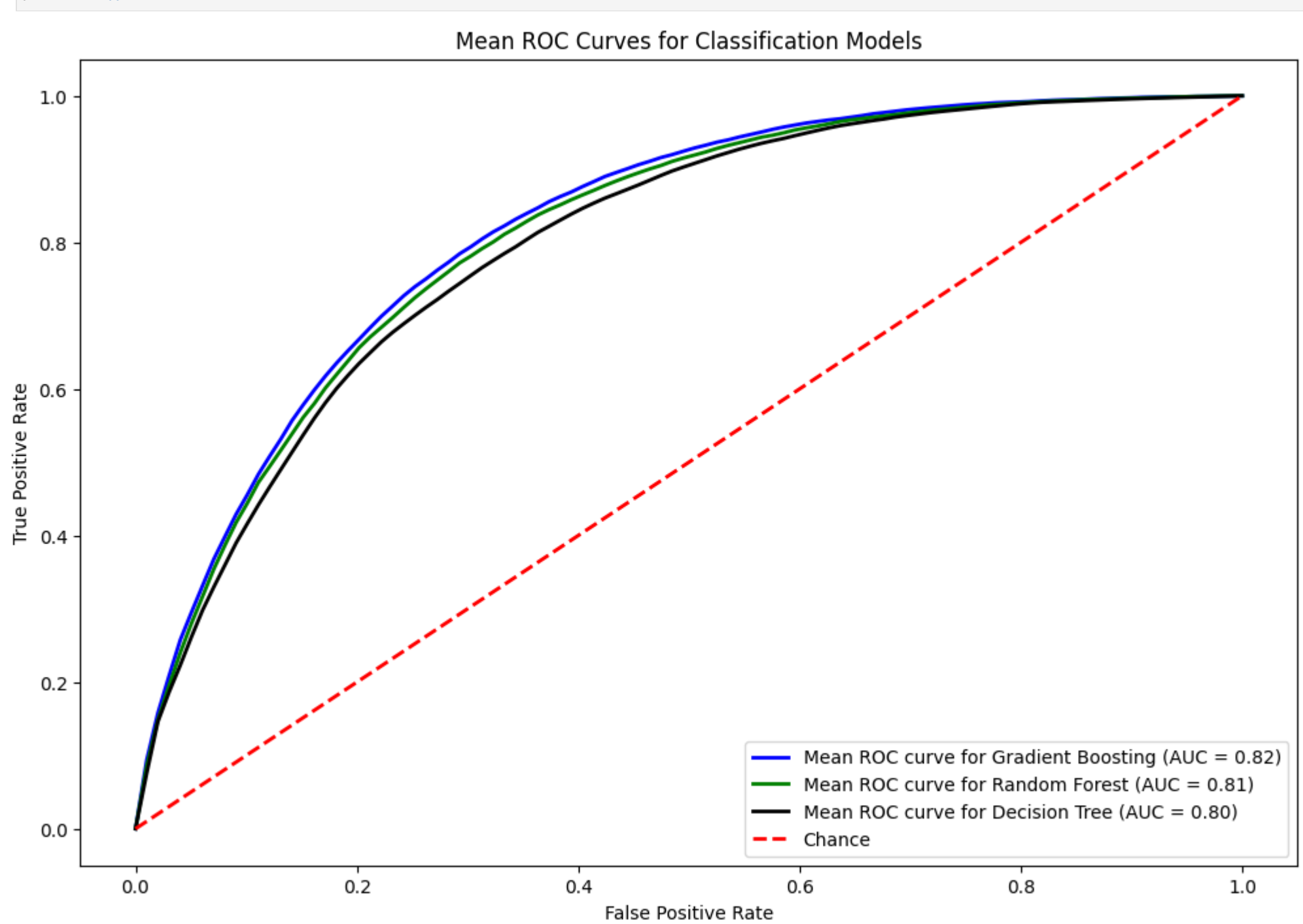
In [10]:
```python
# Application of Boosting
boosted_diabetes = GradientBoostingClassifier(n_estimators = 100, learning_rate = 0.01, max_depth = 6, random_state = seed)
boosted_diabetes.fit(X_train, y_train)
# Predict the target variable using boosting
y_pred = boosted_diabetes.predict(X_test)
# Calculate the F1 score between predicted and actual values
f1 = f1_score(y_test, y_pred)
print('F1 Score = ' + str(f1))
```

```
F1 Score = 0.7561303088120982
```

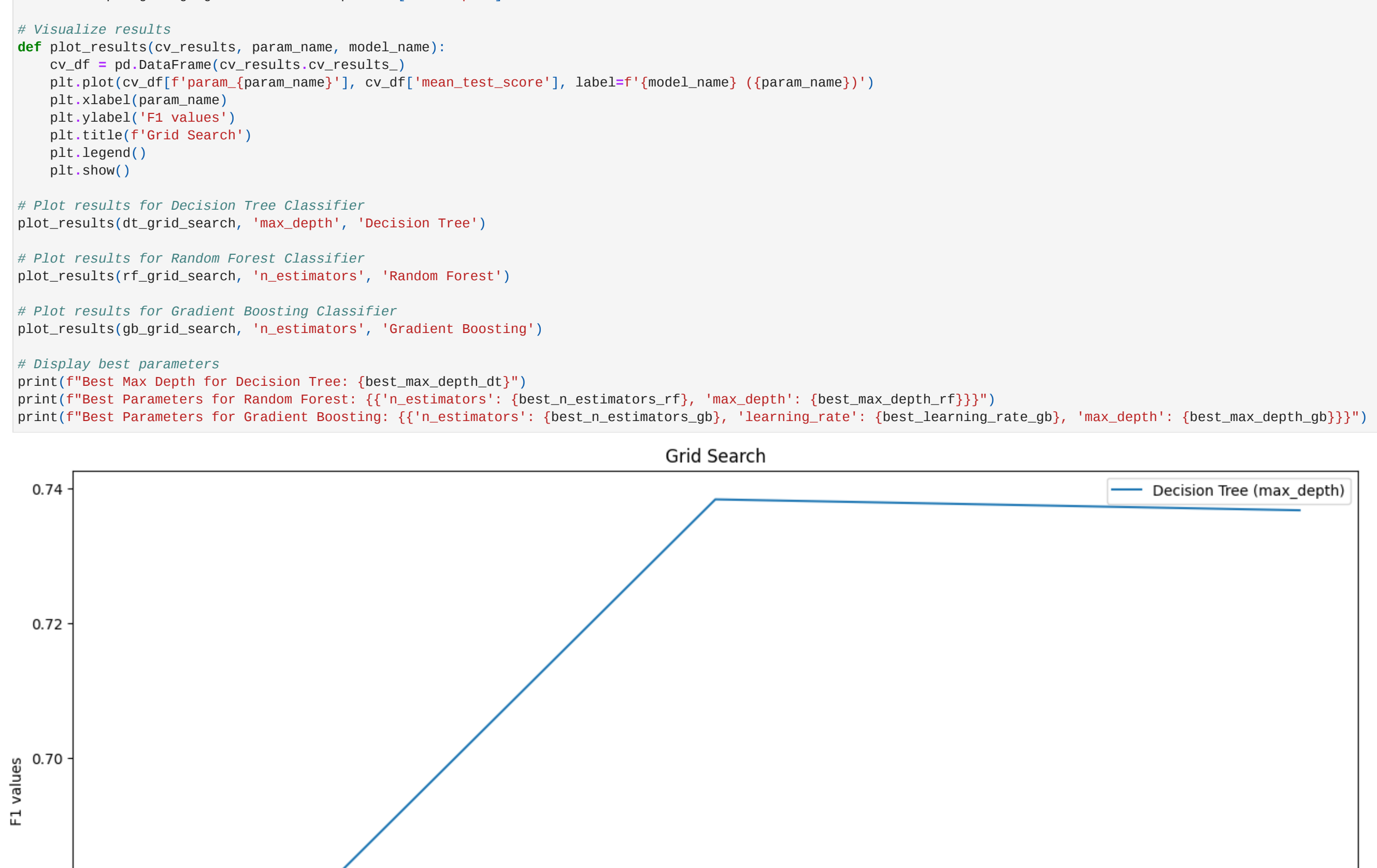
In [11]:
```python
#Plot feature importance attribute of the Gradient Boosting Regressor
feature_importance = boosted_diabetes.feature_importances_*100
rel_imp = pd.Series(feature_importance, index = X.columns).sort_values(inplace = False)
rel_imp.T.plot(kind = 'barh', color = 'r', )
plt.xlabel('Variable Importance')
plt.gca().legend_ = None
```

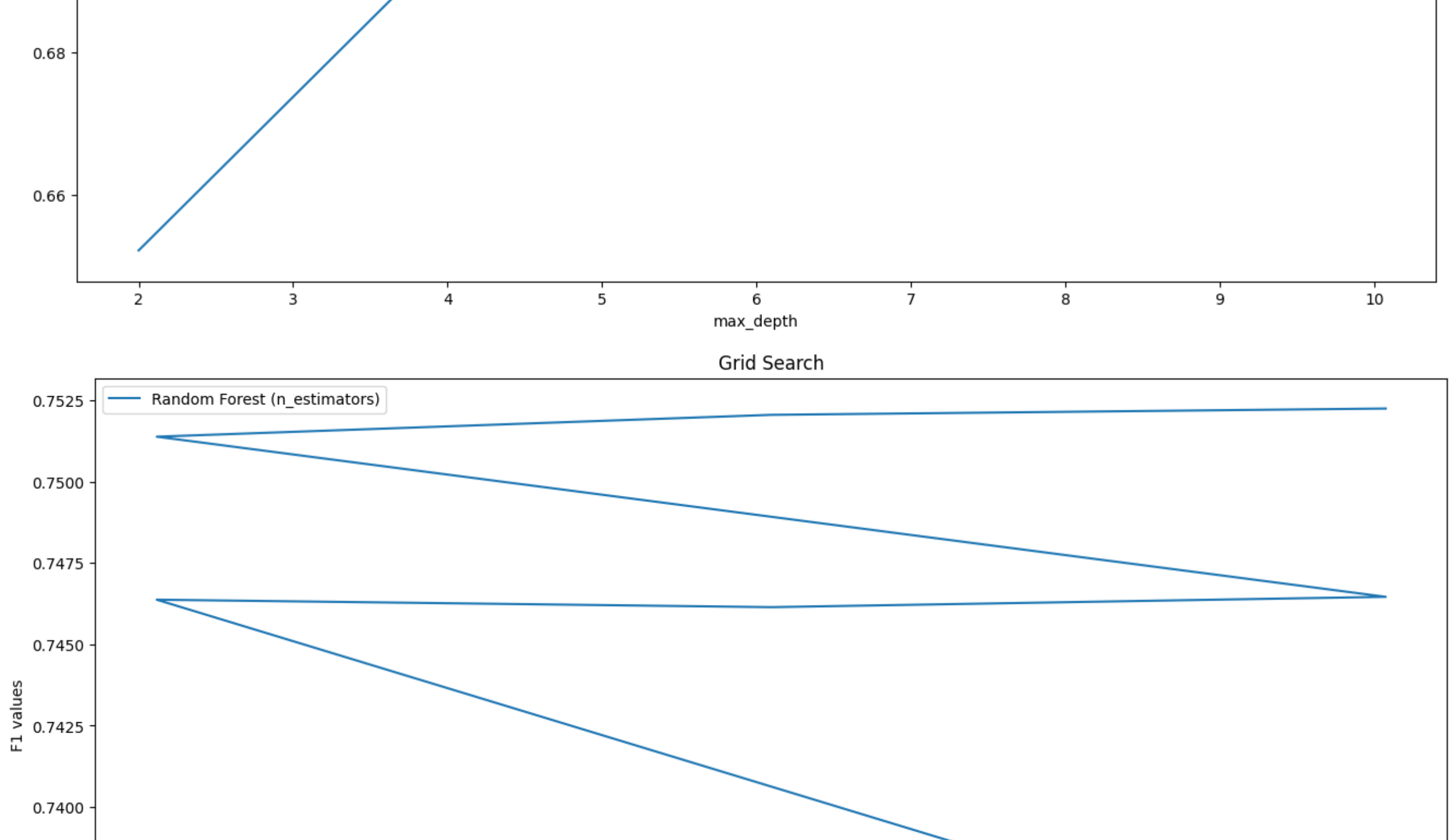


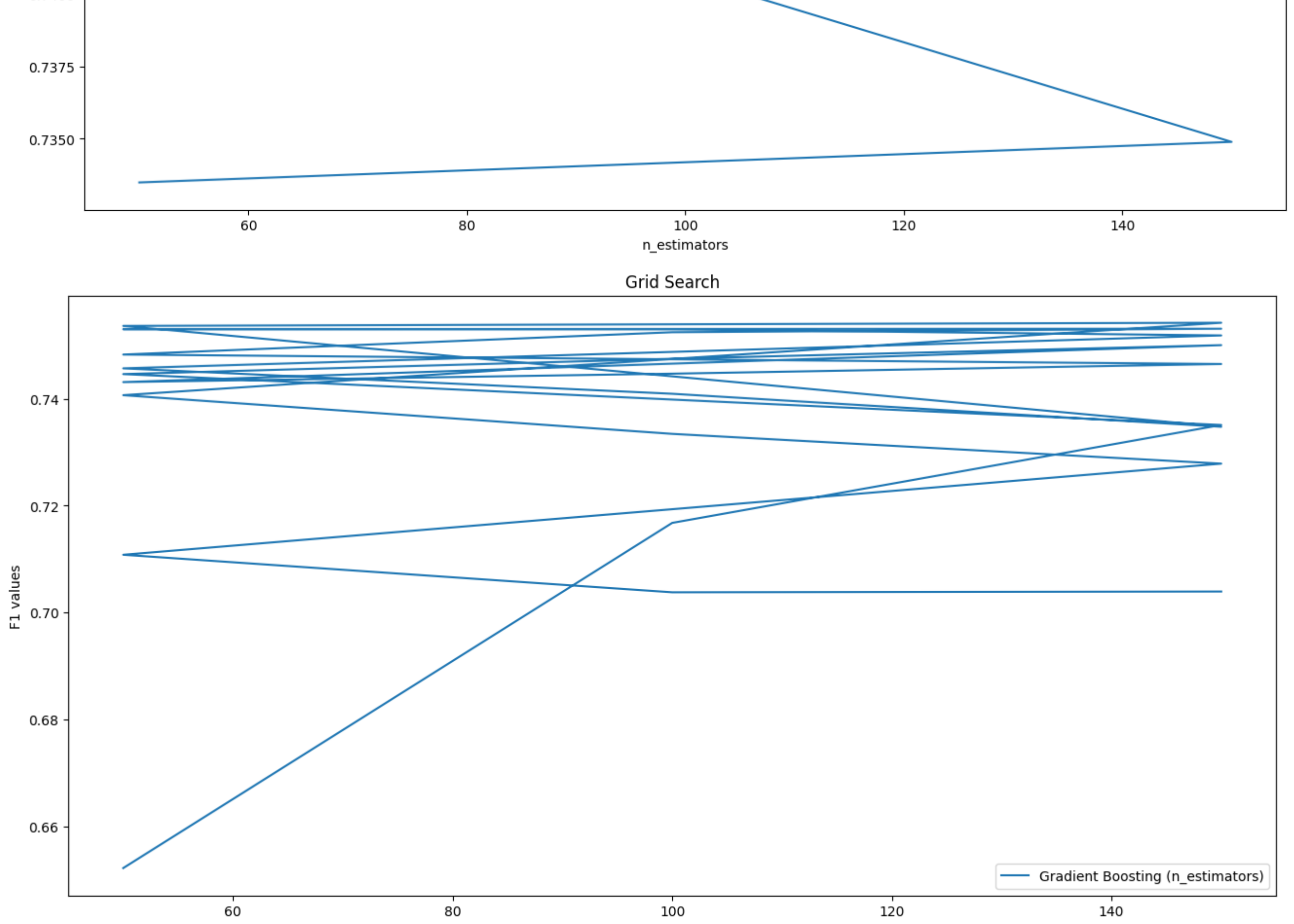
In [12]:
```python
# KFold cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Initialize lists to store true positive rates for each fold
mean_fpr_dt = np.linspace(0, 1, 100)
tpr_list_dt = []

# Initialize Decision Tree model before the loop
clf_dt = DecisionTreeClassifier(max_depth=6, random_state=seed)

# Initialize lists to store true positive rates for each fold
mean_fpr_gb = np.linspace(0, 1, 100)
tpr_list_gb = []

# Initialize Gradient Boosting model before the loop
gradient_boosting = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=6, random_state=seed)

# Initialize lists to store true positive rates for each fold
mean_fpr_rf = np.linspace(0, 1, 100)
tpr_list_rf = []

# Initialize Random Forest model before the loop
random_forest_classifier = RandomForestClassifier(n_estimators=100, max_depth=6, random_state=seed)

# Loop through the folds
for train, test in cv.split(X, y):
    clf_dt.fit(X.iloc[train], y.iloc[train])
    gradient_boosting.fit(X.iloc[train], y.iloc[train])
    random_forest_classifier.fit(X.iloc[train], y.iloc[train])

    # Decision Tree
    y_pred_prob_dt = clf_dt.predict_proba(X.iloc[test])[:, 1]
    fpr_dt, tpr_dt, thresholds_dt = roc_curve(y.iloc[test], y_pred_prob_dt)
    interp_tpr_dt = np.interp(mean_fpr_dt, fpr_dt, tpr_dt)
    interp_tpr_dt[0] = 0.0
    tpr_list_dt.append(interp_tpr_dt)

    # Gradient Boosting
    y_pred_prob_gb = gradient_boosting.predict_proba(X.iloc[test])[:, 1]
    fpr_gb, tpr_gb, thresholds_gb = roc_curve(y.iloc[test], y_pred_prob_gb)
    interp_tpr_gb = np.interp(mean_fpr_gb, fpr_gb, tpr_gb)
    interp_tpr_gb[0] = 0.0
    tpr_list_gb.append(interp_tpr_gb)

    # Random Forest
    y_pred_prob_rf = random_forest_classifier.predict_proba(X.iloc[test])[:, 1]
    fpr_rf, tpr_rf, thresholds_rf = roc_curve(y.iloc[test], y_pred_prob_rf)
    interp_tpr_rf = np.interp(mean_fpr_rf, fpr_rf, tpr_rf)
    interp_tpr_rf[0] = 0.0
    tpr_list_rf.append(interp_tpr_rf)

# Calculate the mean of ROC curves
mean_tpr_dt = np.mean(tpr_list_dt, axis=0)
mean_auc_dt = auc(mean_fpr_dt, mean_tpr_dt)

mean_tpr_gb = np.mean(tpr_list_gb, axis=0)
mean_auc_gb = auc(mean_fpr_gb, mean_tpr_gb)

mean_tpr_rf = np.mean(tpr_list_rf, axis=0)
mean_auc_rf = auc(mean_fpr_rf, mean_tpr_rf)

# Plot the mean ROC curves
plt.figure(figsize=(12, 8))
plt.plot(mean_fpr_gb, mean_tpr_gb, color='blue', lw=2, label='Mean ROC curve for Gradient Boosting (AUC = {:.2f})'.format(mean_auc_gb))
plt.plot(mean_fpr_rf, mean_tpr_rf, color='green', lw=2, label='Mean ROC curve for Random Forest (AUC = {:.2f})'.format(mean_auc_rf))
plt.plot(mean_fpr_dt, mean_tpr_dt, color='black', lw=2, label='Mean ROC curve for Decision Tree (AUC = {:.2f})'.format(mean_auc_dt))
plt.plot([0, 1], [0, 1], color="r", lw=2, label='Chance', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Mean ROC Curves for Classification Models')
plt.legend(loc='lower right')
plt.show()
```



In [13]:
```python
from sklearn.model_selection import GridSearchCV

# Decision Tree Classifier
dt_parameters = {'max_depth': [2, 6, 10]}
dt_clf = DecisionTreeClassifier()
dt_grid_search = GridSearchCV(dt_clf, dt_parameters, cv=5, scoring='f1')
dt_grid_search.fit(X_train, y_train)
best_max_depth_dt = dt_grid_search.best_params_['max_depth']

# Random Forest Classifier
rf_parameters = {'n_estimators': [50, 100, 150], 'max_depth': [2, 6, 10]}
rf_clf = RandomForestClassifier()
rf_grid_search = GridSearchCV(rf_clf, rf_parameters, cv=5, scoring='f1')
rf_grid_search.fit(X_train, y_train)
best_n_estimators_rf = rf_grid_search.best_params_['n_estimators']
best_max_depth_rf = rf_grid_search.best_params_['max_depth']

# Gradient Boosting Classifier
gb_parameters = {'n_estimators': [50, 100, 150], 'learning_rate': [0.01, 0.1, 0.5], 'max_depth': [2, 6, 10]}
gb_clf = GradientBoostingClassifier()
gb_grid_search = GridSearchCV(gb_clf, gb_parameters, cv=5, scoring='f1')
gb_grid_search.fit(X_train, y_train)
best_n_estimators_gb = gb_grid_search.best_params_['n_estimators']
best_learning_rate_gb = gb_grid_search.best_params_['learning_rate']
best_max_depth_gb = gb_grid_search.best_params_['max_depth']

# Visualize results
def plot_results(cv_results, param_name, model_name):
    cv_df = pd.DataFrame(cv_results.cv_results_)
    plt.plot(cv_df[f'param_{param_name}'], cv_df['mean_test_score'], label=f'{model_name} ({param_name})')
    plt.xlabel(param_name)
    plt.ylabel('F1 values')
    plt.title(f'Grid Search')
    plt.legend()
    plt.show()

# Plot results for Decision Tree Classifier
plot_results(dt_grid_search, 'max_depth', 'Decision Tree')

# Plot results for Random Forest Classifier
plot_results(rf_grid_search, 'n_estimators', 'Random Forest')

# Plot results for Gradient Boosting Classifier
plot_results(gb_grid_search, 'n_estimators', 'Gradient Boosting')

# Display best parameters
print(f"Best Max Depth for Decision Tree: {best_max_depth_dt}")
print(f"Best Parameters for Random Forest: {{'n_estimators': {best_n_estimators_rf}, 'max_depth': {best_max_depth_rf}}}")
print(f"Best Parameters for Gradient Boosting: {{'n_estimators': {best_n_estimators_gb}, 'learning_rate': {best_learning_rate_gb}, 'max_depth': {best_max_depth_gb}}}")
```








```
Best Max Depth for Decision Tree: 6
Best Parameters for Random Forest: {'n_estimators': 150, 'max_depth': 10}
Best Parameters for Gradient Boosting: {'n_estimators': 150, 'learning_rate': 0.5, 'max_depth': 2}
```

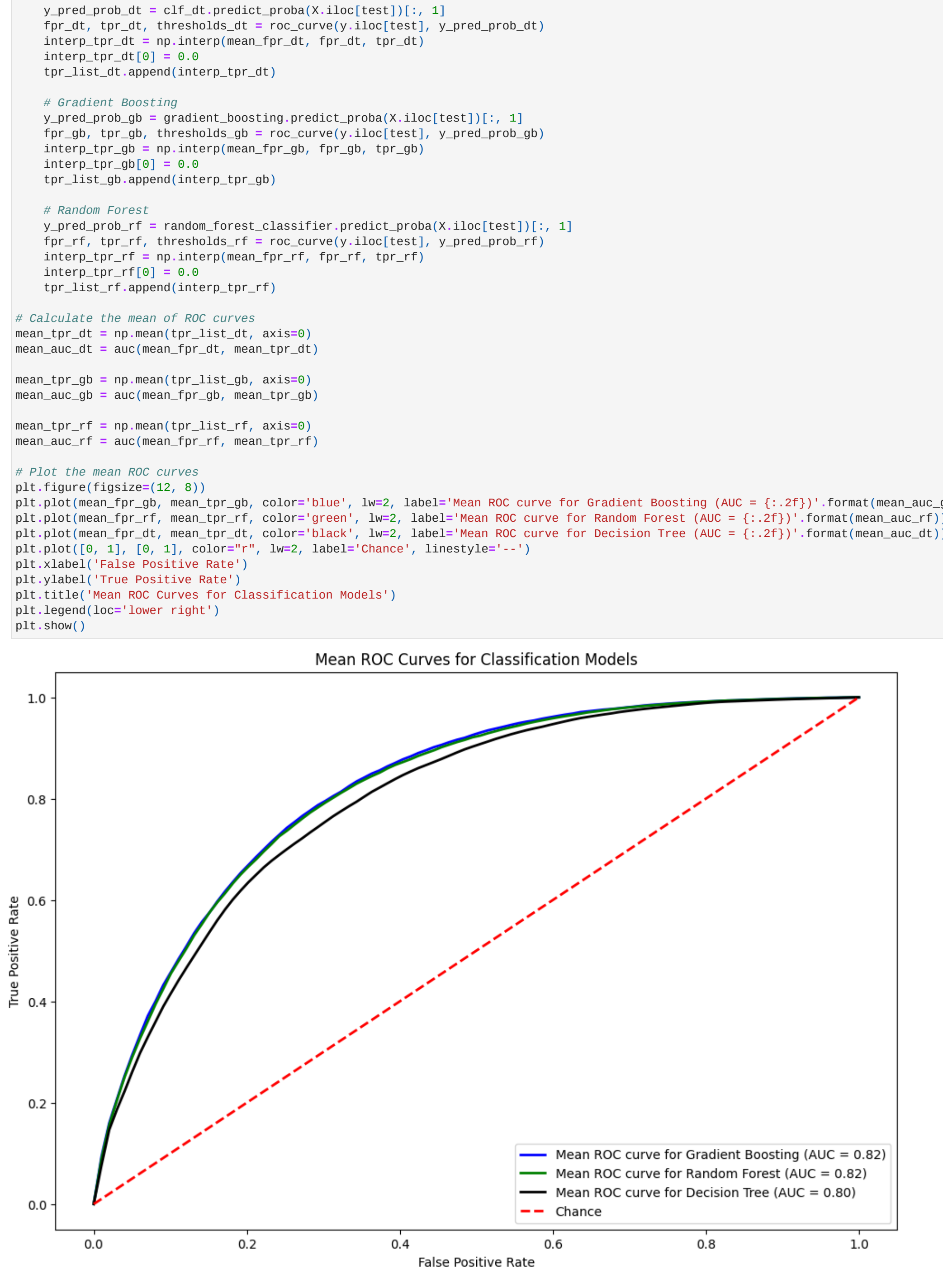
In [14]:
```python
# Application of random forests
random_forest_diabetes = RandomForestClassifier(n_estimators=150, max_depth=10 , random_state = seed)
random_forest_diabetes.fit(X_train, y_train)
# Predict the target variable using random forests
y_pred = random_forest_diabetes.predict(X_test)
# Calculate the F1 score between predicted and actual values
f1 = f1_score(y_test, y_pred)
print('F1 Score = ' + str(f1))
```

```
F1 Score = 0.7614097543609826
```

In [15]:
```python
# Application of Boosting
boosted_diabetes = GradientBoostingClassifier(n_estimators = 150, learning_rate = 0.5, max_depth = 2, random_state = seed)
boosted_diabetes.fit(X_train, y_train)
# Predict the target variable using boosting
y_pred = boosted_diabetes.predict(X_test)
# Calculate the F1 score between predicted and actual values
f1 = f1_score(y_test, y_pred)
print('F1 Score = ' + str(f1))
```

```
F1 Score = 0.7611439982882819
```

In [16]:
```python
# KFold cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Initialize lists to store true positive rates for each fold
mean_fpr_dt = np.linspace(0, 1, 100)
tpr_list_dt = []

# Initialize Decision Tree model before the loop
clf_dt = DecisionTreeClassifier(max_depth=6, random_state=seed)

# Initialize lists to store true positive rates for each fold
mean_fpr_gb = np.linspace(0, 1, 100)
tpr_list_gb = []

# Initialize Gradient Boosting model before the loop
gradient_boosting = GradientBoostingClassifier(n_estimators=150, learning_rate=0.5, max_depth=2, random_state=seed)

# Initialize lists to store true positive rates for each fold
mean_fpr_rf = np.linspace(0, 1, 100)
tpr_list_rf = []

# Initialize Random Forest model before the loop
random_forest_classifier = RandomForestClassifier(n_estimators=150, max_depth=10, random_state=seed)

# Loop through the folds
for train, test in cv.split(X, y):
    clf_dt.fit(X.iloc[train], y.iloc[train])
    gradient_boosting.fit(X.iloc[train], y.iloc[train])
    random_forest_classifier.fit(X.iloc[train], y.iloc[train])

    # Decision Tree
    y_pred_prob_dt = clf_dt.predict_proba(X.iloc[test])[:, 1]
    fpr_dt, tpr_dt, thresholds_dt = roc_curve(y.iloc[test], y_pred_prob_dt)
    interp_tpr_dt = np.interp(mean_fpr_dt, fpr_dt, tpr_dt)
    interp_tpr_dt[0] = 0.0
    tpr_list_dt.append(interp_tpr_dt)

    # Gradient Boosting
    y_pred_prob_gb = gradient_boosting.predict_proba(X.iloc[test])[:, 1]
    fpr_gb, tpr_gb, thresholds_gb = roc_curve(y.iloc[test], y_pred_prob_gb)
    interp_tpr_gb = np.interp(mean_fpr_gb, fpr_gb, tpr_gb)
    interp_tpr_gb[0] = 0.0
    tpr_list_gb.append(interp_tpr_gb)

    # Random Forest
    y_pred_prob_rf = random_forest_classifier.predict_proba(X.iloc[test])[:, 1]
    fpr_rf, tpr_rf, thresholds_rf = roc_curve(y.iloc[test], y_pred_prob_rf)
    interp_tpr_rf = np.interp(mean_fpr_rf, fpr_rf, tpr_rf)
    interp_tpr_rf[0] = 0.0
    tpr_list_rf.append(interp_tpr_rf)

# Calculate the mean of ROC curves
mean_tpr_dt = np.mean(tpr_list_dt, axis=0)
mean_auc_dt = auc(mean_fpr_dt, mean_tpr_dt)

mean_tpr_gb = np.mean(tpr_list_gb, axis=0)
mean_auc_gb = auc(mean_fpr_gb, mean_tpr_gb)

mean_tpr_rf = np.mean(tpr_list_rf, axis=0)
mean_auc_rf = auc(mean_fpr_rf, mean_tpr_rf)

# Plot the mean ROC curves
plt.figure(figsize=(12, 8))
plt.plot(mean_fpr_gb, mean_tpr_gb, color='blue', lw=2, label='Mean ROC curve for Gradient Boosting (AUC = {:.2f})'.format(mean_auc_gb))
plt.plot(mean_fpr_rf, mean_tpr_rf, color='green', lw=2, label='Mean ROC curve for Random Forest (AUC = {:.2f})'.format(mean_auc_rf))
plt.plot(mean_fpr_dt, mean_tpr_dt, color='black', lw=2, label='Mean ROC curve for Decision Tree (AUC = {:.2f})'.format(mean_auc_dt))
plt.plot([0, 1], [0, 1], color="r", lw=2, label='Chance', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Mean ROC Curves for Classification Models')
plt.legend(loc='lower right')
plt.show()
```



In [ ]: