Build a Random Forest classifier on any readily available disease dataset to predict the correct disease.

Compare the performance of the classifier with decision tree.

```
# import important packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler, MinMaxScaler
import pandas_profiling


# Load dataset
data = pd.read_csv("pima.csv")


data.sample(5)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
529	0	111	65	0	0	24.6	0.660	31	0
449	0	120	74	18	63	30.5	0.285	26	0
632	2	111	60	0	0	26.2	0.343	23	0
315	2	112	68	22	94	34.1	0.315	26	0
303	5	115	98	0	0	52.9	0.209	28	1

```
data.columns
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

```
# load data with selected features
X = data.drop(["Outcome", "SkinThickness"], axis=1)
y = data["Outcome"]

# standardize the dataset
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# split into train and test set
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, stratify=y, test_size=0.10, random_state=42
)


# Create a Random Classifier
clf = RandomForestClassifier(n_estimators=100)

# Train the model using the training sets
clf.fit(X_train, y_train)

# prediction on test set
y_pred = clf.predict(X_test)

# Calculate Model Accuracy,
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8181818181818182
```

```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
```

```
clf.fit(X_train,y_train)
```

```
DecisionTreeClassifier()
```

```
y_pred=clf.predict(X_test)
```

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
0.7662337662337663
```

Array Method for different parameters on RandomForest Classifier

```
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]
}
```

Grid Search CV

It is the process of performing hyperparameter tuning in order to determine the optimal values for a given model. As mentioned above, the performance of a model significantly depends on the value of hyperparameters. Note that there is no way to know in advance the best values for hyperparameters

so ideally, we need to try all possible values to know the optimal values. Doing this manually could take a considerable amount of time and resources and thus we use GridSearchCV to automate the tuning of hyperparameters.

GridSearchCV tries all the combinations of the values passed in the dictionary and evaluates the model for each combination using the Cross-Validation method. Hence after using this function we get accuracy/loss for every combination of hyperparameters and we can choose the one with the best performance.

```
from sklearn.model_selection import GridSearchCV
```

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

Unsupported Cell Type. Double-Click to inspect/edit the content.

```
grid_search.fit(X_train, y_train)
```

```
Fitting 4 folds for each of 180 candidates, totalling 720 fits
GridSearchCV(cv=4, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'max_depth': [2, 3, 5, 10, 20],
                         'min_samples_leaf': [5, 10, 20, 50, 100, 200],
                         'n_estimators': [10, 25, 30, 50, 100, 200]},
             scoring='accuracy', verbose=1)
```

```
grid_search.best_score_
```

```
0.7713066272348433
```

```
rf_best = grid_search.best_estimator_
rf_best
```

```
RandomForestClassifier(max_depth=3, min_samples_leaf=5, n_estimators=200)
```

```
rf_best.estimators_[5]
```

```
DecisionTreeClassifier(max_depth=3, max_features='auto', min_samples_leaf=5,
                       random_state=182683720)
```

```
rf_best.estimators_[4]
```

```
DecisionTreeClassifier(max_depth=3, max_features='auto', min_samples_leaf=5,
                       random_state=1182753142)
```

```
from sklearn.tree import plot_tree
plt.figure(figsize=(80,40))
plot_tree(rf_best.estimators_[5], feature_names = X.columns,class_names=['Disease', "No Disease"],filled=True);
plt.show()
```