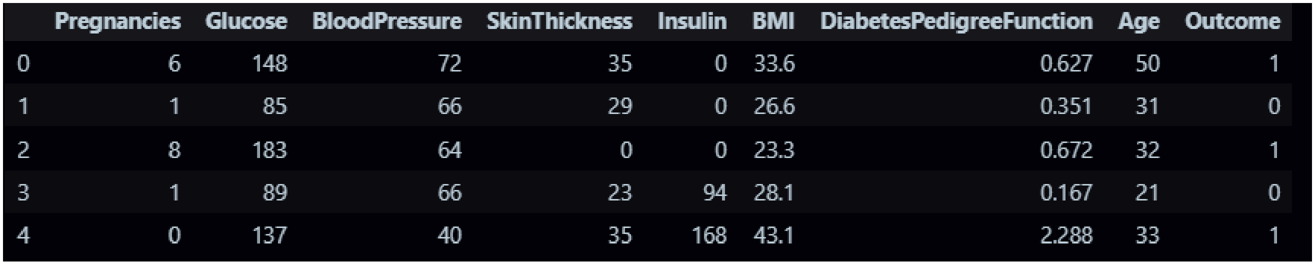
Importing Libraries:

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt
import math
```

import the dataset:

```
df = pd.read_csv('/content/diabetes.csv')
df.head()
```



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Step 1: Calculating Entropy for dataset

```
def calculate_entropy(data, target_column):
    total_rows = len(data)
    target_values = data[target_column].unique()
    entropy = 0
    for value in target_values:
        value_count = len(data[data[target_column] == value])
        proportion = value_count / total_rows
        entropy -= proportion * math.log2(proportion)
    return entropy
entropy_outcome = calculate_entropy(df, 'Outcome')
print(f"Entropy of the dataset: {entropy_outcome}")
```

Entropy of the dataset: 0.9331343166407831

Step 2: Calculating Entropy and Information Gain

```
def calculate_entropy(data, target_column):  # For each categorical variable
    total_rows = len(data)
    target_values = data[target_column].unique()
    entropy = 0
    for value in target_values:
        value_count = len(data[data[target_column] == value])
        proportion = value_count / total_rows
        entropy -= proportion * math.log2(proportion) if proportion != 0 else 0
    return entropy

def calculate_information_gain(data, feature, target_column):
    entropy_outcome = calculate_entropy(data, target_column)  # You must calculate this here
    unique_values = data[feature].unique()
    weighted_entropy = 0
    for value in unique_values:
        subset = data[data[feature] == value]
        proportion = len(subset) / len(data)
        weighted_entropy += proportion * calculate_entropy(subset, target_column)
    information_gain = entropy_outcome - weighted_entropy
    return information_gain
```

Step 3: Assessing best feature with highest information gain

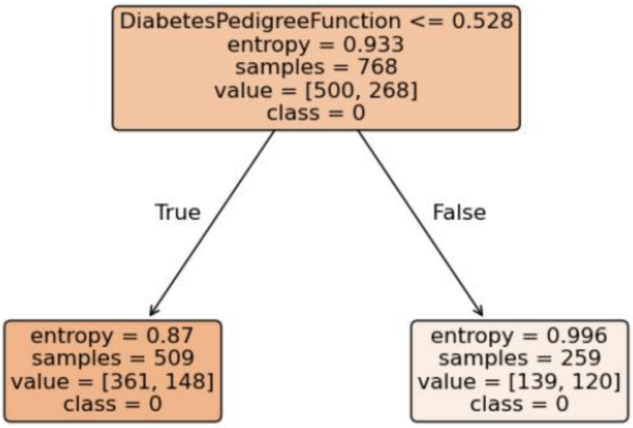
```
target_entropy = calculate_entropy(df, 'Outcome')
for column in df.columns:
    if column != 'Outcome':
        info_gain = calculate_information_gain(df, column, 'Outcome')
        print(f"{column} - Information Gain: {info_gain:.3f}")
```

```
Pregnancies - Information Gain: 0.062
Glucose - Information Gain: 0.304
BloodPressure - Information Gain: 0.059
SkinThickness - Information Gain: 0.082
Insulin - Information Gain: 0.277
BMI - Information Gain: 0.344
DiabetesPedigreeFunction - Information Gain: 0.651
Age - Information Gain: 0.141
```

Let's Plot the decision tree built so far-

```
selected_feature = 'DiabetesPedigreeFunction'
clf = DecisionTreeClassifier(criterion='entropy', max_depth=1)
X = df[[selected_feature]]
y = df['Outcome']
clf.fit(X, y)

plt.figure(figsize=(8, 6))
plot_tree(clf, feature_names=[selected_feature], class_names=['0', '1'], filled=True, rounded=True)
plt.show()
```



Step 4: Built ID3 Algorithm

```
def id3(data, target_column, features):
  if len(data[target_column].unique()) == 1:
    return data[target_column].iloc[0]
  if len(features) == 0:
    return data[target_column].mode().iloc[0]
  best_feature = max(features, key=lambda x: calculate_information_gain(data, x, target_column))
  tree = {best_feature: {}}
  features = [f for f in features if f != best_feature]
  for value in data[best_feature].unique():
    subset = data[data[best_feature] == value]
    tree[best_feature][value] = id3(subset, target_column, features)

  return tree
```

```
{'DiabetesPedigreeFunction': {0.627: 1, 0.351: 0, 0.672: 1, 0.167: 0, 2.288: 1, 0.201: 0, 0.248: {'Pregnancies': {3: 1, 1:
0}}, 0.134: 0, 0.158: .......................... {'Pregnancies': {0: 1, 2: 0}}, 0.133: 0, 0.155: 0, 1.162: 0, 1.292: 1, 0.182: 0,
1.394: 1, 0.217: 0, 0.631: 0, 0.88: 0, 0.614: 0, 0.332: 0, 0.366: 0, 0.181: 0, 0.828: 0, 0.335: 1, 0.856: 0, 0.886: 0, 0.439:
{'Pregnancies': {7: 1, 5: 0}}, 0.253: 0, 0.598: 0, 0.904: 0, 0.483: 0, 0.565: 1, 0.118: 0, 0.177: 0, 0.176: 0, 0.295: 0, 0.441:
1, 0.352: 0, 0.826: 1, 0.97: 1, 0.595: 0, 0.317: 0, 0.265: 0, 0.646: 1, 0.426: 0, 0.56: 0, 0.515: 0, 0.453: 0, 0.785: 1,
0.734: 1, 1.174: 0, 0.488: 0, 0.358: 1, 1.096: 0, 0.408: 1, 1.182: 1, 0.222: 1, 1.057: 1, 0.766: 0, 0.171: 0}
```