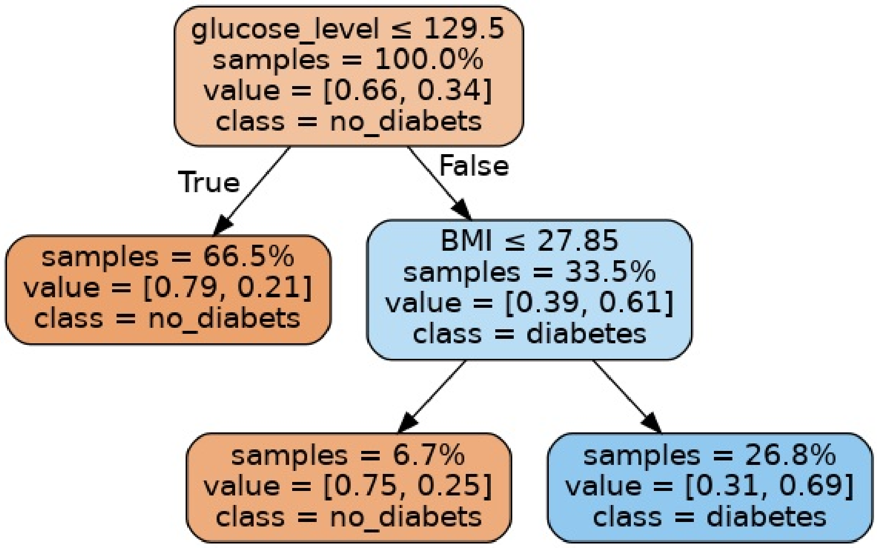
In [1]:

```
#IMPORTS:
import pandas as pd
import sklearn
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus

diabetes_data = pd.read_csv("diabetesMod_en.csv", header=0)

predictors = diabetes_data.drop(['diabetes_diagnose'],axis=1) # Features
# "drop" a column named 'diabetes_diagnose' from the data table..
# ... and store the remaining columns in a new table inside a variable called 'predictors'.

target = diabetes_data['diabetes_diagnose'] # Target variable
# pick from the dataframe  only a column named 'diabetes_diagnose' ...
# ... and store it inside a variable called 'target'.

from sklearn.model_selection import train_test_split
# "train_test_split" command can be found in the "model_selection" part in the "sklearn" library

predictors_teach, predictors_test, target_teach, target_test = sklearn.model_selection.train_test_split(predictor
s, target, test_size=0.3, random_state=1) # 70% training and 30% test

decision_tree = DecisionTreeClassifier(criterion="gini",min_impurity_decrease=0.02,max_depth=4 ,min_samples_leaf=
20)
# use a tool called DecisionTreeClassifier to create a new decision tree
# give it several parameters and I will now use four of them (explained in the text next)
# new decision tree to be slipped into a variable called "decision_tree"

decision_tree = decision_tree.fit(predictors_teach, target_teach)
# create a new decision tree before and slipped it into a variable called "decision_tree"

dot_data = StringIO()
# create the required data structure for visualizing the decision tree

export_graphviz(decision_tree, out_file=dot_data,
                filled=True, rounded=True,impurity=False, proportion=True,precision=2,
                special_characters=True, feature_names = predictors.columns,class_names=['no_diabets','diabetes']
)

# we use the export_graphviz tool to create the necessary file for visualization
# The first parameter is the decision tree, then we provide the data structure we just created
# The following 6 parameters define the visual appearance of the decision tree and the information that is writte
n to the cells.
# as the second last parameter (feature_names) we give the names of the predicted variables so that they can be a
ppended
# ... according to the visualization
# As the last parameter we give the names of the classes of the predicted variable (did not survive, survived)

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
# Code - explained:
# lets print the graph to screen
```

Out[1]:



In [2]:

```
prediction = decision_tree.predict(predictors_test)
from sklearn import metrics


from sklearn import metrics
# I know the sklearn library contains a tool called "metrics"

accuracy = metrics.accuracy_score(target_test, prediction)
# I know that "metrics" contains also "accuracy_score" which compares the prediction to the correct results.

print("accuracy_score:",accuracy)
```

```
accuracy_score: 0.7575757575757576
```

In [3]:

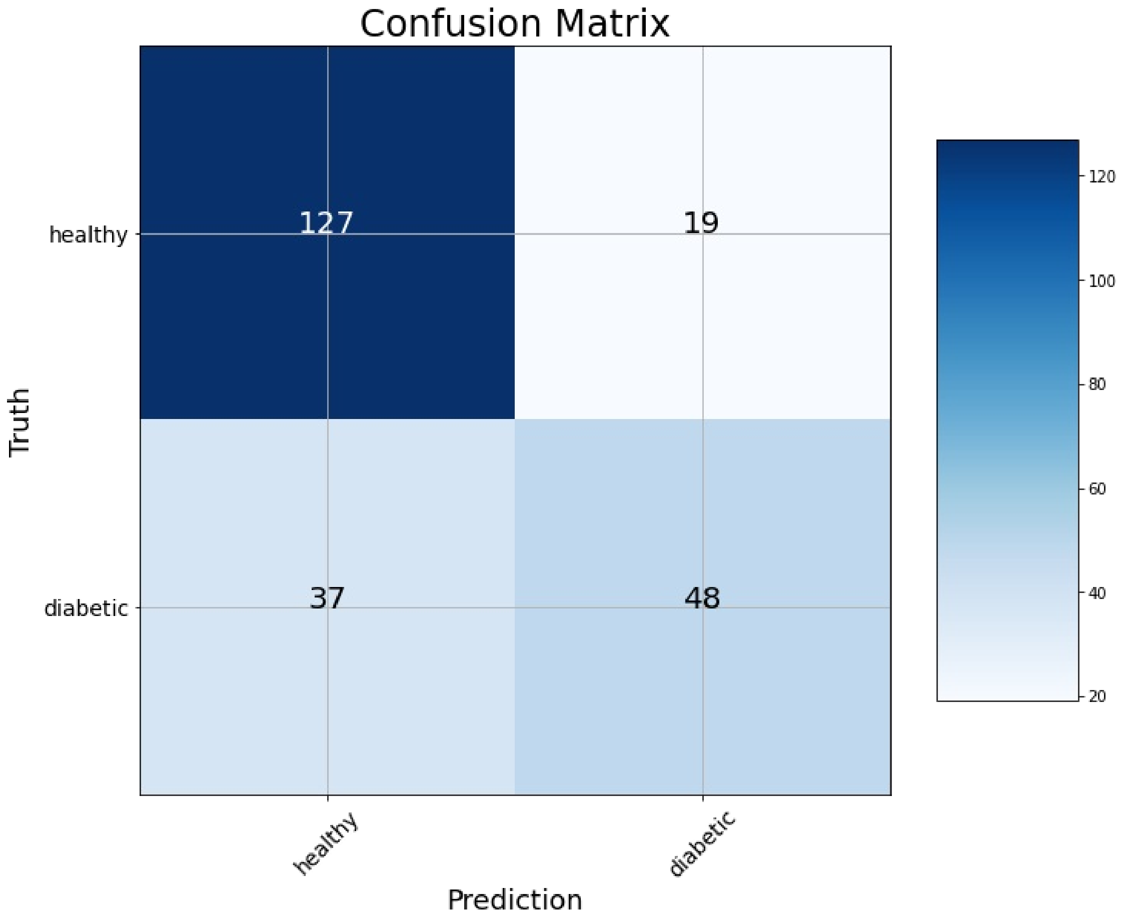
```
confusion_matrix = metrics.confusion_matrix(target_test, prediction)
# Code - Explained : "metrics" contains the command "confusion_matrix" which compares the prediction with the cor
rect results
# give the target variable and the prediction as a parameter.

print(confusion_matrix)
```

```
[[127  19]
 [ 37  48]]
```

In [4]:

```
from sklearn.metrics import confusion_matrix
import itertools
import matplotlib.pyplot as plt
import numpy as np

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion Matrix',
                          cmap=plt.cm.Blues):

    # plt.cm.Oranges
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    Source: http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.figure(figsize = (10, 10))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title, size = 24)
    plt.colorbar(aspect=4)
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45, size = 14)
    plt.yticks(tick_marks, classes, size = 14)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.

    # Labeling the plot
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt), fontsize = 20,
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.grid(None)
    plt.tight_layout()
    plt.ylabel('Truth', size = 18)
    plt.xlabel('Prediction', size = 18)
cm = confusion_matrix(target_test, prediction)
plot_confusion_matrix(cm, classes = ['healthy', 'diabetic'],
                      normalize = False,
                      title = 'Confusion Matrix')
```

```
Confusion matrix, without normalization
[[127  19]
 [ 37  48]]
```



```
In [ ]:
```