5th April Assignment

April 20, 2023

1 Assignment 59

You are a data scientist working for a healthcare company, and you have been tasked with creating a decision tree to help identify patients with diabetes based on a set of clinical variables. You have been given a dataset (diabetes.csv) with the following variables: 1. Pregnancies: Number of times pregnant (integer) 2. Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test (integer) 3. BloodPressure: Diastolic blood pressure (mm Hg) (integer) 4. SkinThickness: Triceps skin fold thickness (mm) (integer) 5. Insulin: 2-Hour serum insulin (mu U/ml) (integer) 6. BMI: Body mass index (weight in kg/(height in m)^2) (float) 7. DiabetesPedigreeFunction: Diabetes pedigree function (a function which scores likelihood of diabetes based on family history) (float) 8. Age: Age in years (integer) 9. Outcome: Class variable (0 if non-diabetic, 1 if diabetic) (integer)

Here's the dataset link:

Your goal is to create a decision tree to predict whether a patient has diabetes based on the other variables. Here are the steps you can follow:

https://drive.google.com/file/d/1Q4J8KS1wm4-_YTuc389enPh6O-eTNcx2/view?

usp=sharing

Q1. Import the dataset and examine the variables. Use descriptive statistics and visualizations to understand the distribution and relationships between the variables.

[74]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

[75]:
```
df = pd.read_csv('diabetes.csv')
df.head()
```

[75]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

```
   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

[76]:
```
df.info()
```

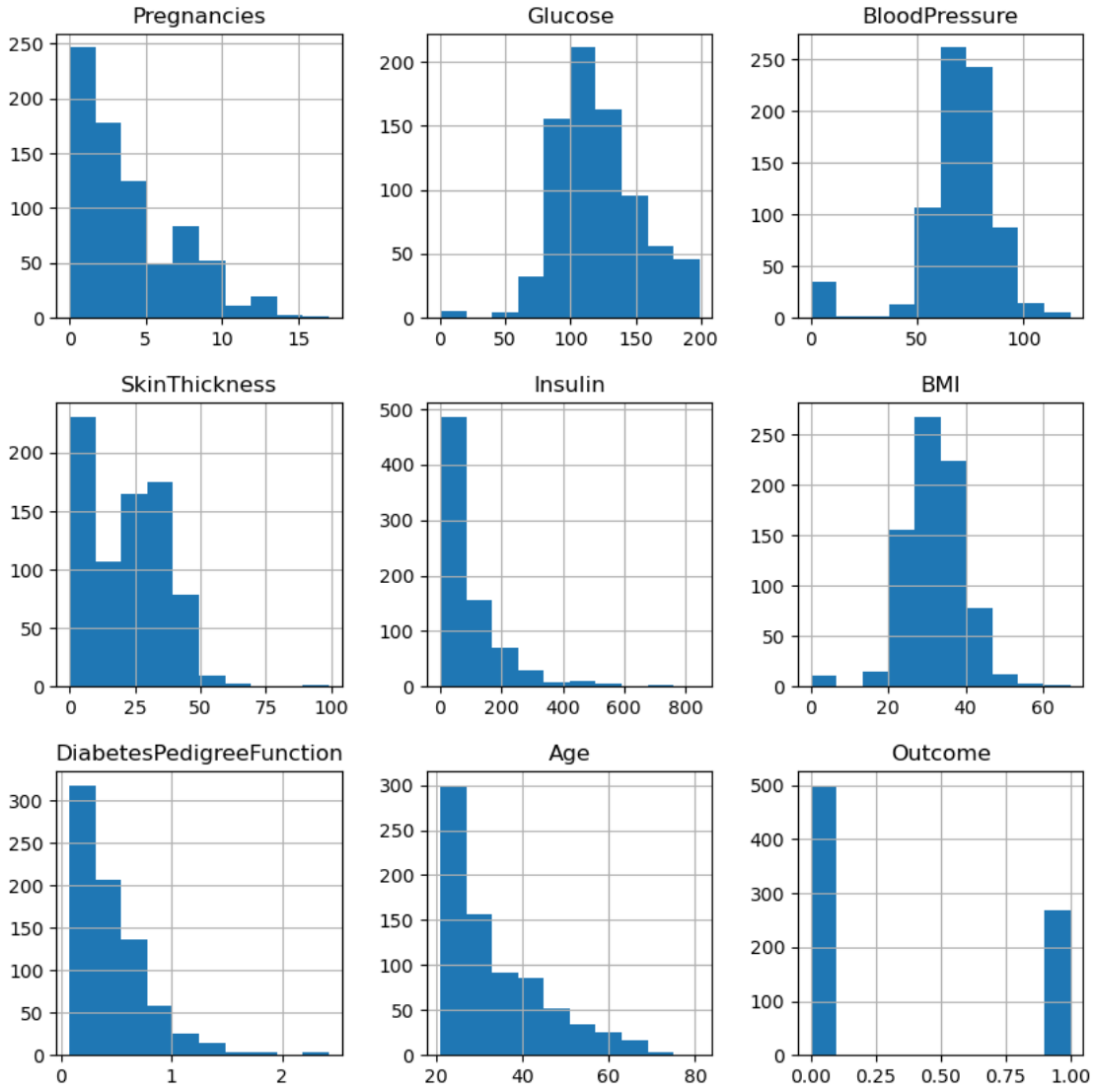
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

[77]:
```
df.describe()
```

[77]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000   
mean      3.845052  120.894531      69.105469      20.536458   79.799479   
std       3.369578   31.972618      19.355807      15.952218  115.244002   
min       0.000000    0.000000       0.000000       0.000000    0.000000   
25%       1.000000   99.000000      62.000000       0.000000    0.000000   
50%       3.000000  117.000000      72.000000      23.000000   30.500000   
75%       6.000000  140.250000      80.000000      32.000000  127.250000   
max      17.000000  199.000000     122.000000      99.000000  846.000000   

              BMI  DiabetesPedigreeFunction         Age     Outcome  
count  768.000000                768.000000  768.000000  768.000000  
mean    31.992578                  0.471876   33.240885    0.348958  
std      7.884160                  0.331329   11.760232    0.476951  
min      0.000000                  0.078000   21.000000    0.000000  
25%     27.300000                  0.243750   24.000000    0.000000  
50%     32.000000                  0.372500   29.000000    0.000000  
75%     36.600000                  0.626250   41.000000    1.000000  
```

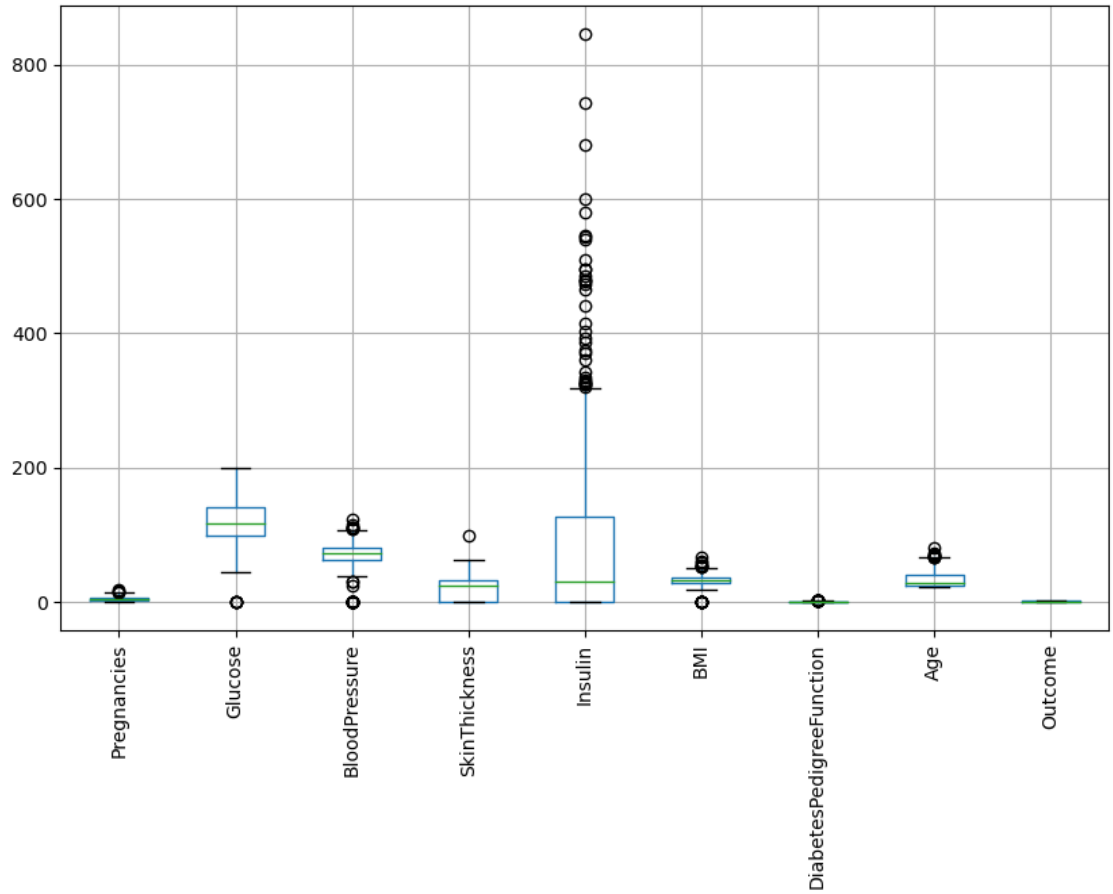
```
max      67.100000                    2.420000    81.000000     1.000000
```

[78]:
```
# Create histograms of each variable
df.hist(figsize=(10,10))
plt.show()
```



Q2. Preprocess the data by cleaning missing values, removing outliers, and transforming categorical variables into dummy variables if necessary.

[79]:
```
# Create box plots of each variable
df.boxplot(figsize=(10,6),rot=90)
plt.show()
```



[80]:
```
# Calculate the IQR of each variable
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

# Remove outliers
df = df[~((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)]
```

[81]:
```
# Check for categorical variables
print(df.nunique())
```

```
Pregnancies                 14
Glucose                    132
BloodPressure               39
SkinThickness               48
Insulin                    148
BMI                        227
DiabetesPedigreeFunction   442
Age                         46
```

```
Outcome                       2
dtype: int64
```

Q3. Split the dataset into a training set and a test set. Use a random seed to ensure reproducibility.

[82]:
```
# Split the dataset into features (X) and target (y)
X = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

[83]:
```
#train test split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=42,test_size=0.
 ↪33)
```

Q4. Use a decision tree algorithm, such as ID3 or C4.5, to train a decision tree model on the training set. Use cross-validation to optimize the hyperparameters and avoid overfitting.

[84]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
```

[85]:
```
# Define the decision tree classifier
tree = DecisionTreeClassifier(random_state=42)
```

[86]:
```
# Define the hyperparameters to optimize
params = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [2, 4, 6, 8, 10],
    'min_samples_split': [2, 4, 6, 8, 10],
    'min_samples_leaf': [1, 2, 3, 4, 5],
}
```

[87]:
```
# Perform grid search cross-validation to find the best hyperparameters
grid_search = GridSearchCV(tree, params, cv=5)
grid_search.fit(X_train, y_train)
```

[87]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=42),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [2, 4, 6, 8, 10],
                         'min_samples_leaf': [1, 2, 3, 4, 5],
                         'min_samples_split': [2, 4, 6, 8, 10]})
```

[88]:
```
grid_search.best_params_
```

[88]:
```
{'criterion': 'entropy',
 'max_depth': 4,
 'min_samples_leaf': 4,
```

```
 'min_samples_split': 2}
```

[89]:
```
tree=DecisionTreeClassifier(criterion='entropy',max_depth=4,min_samples_leaf=4,min_samples_spl
```

[90]:
```
tree.fit(X_train,y_train)
```

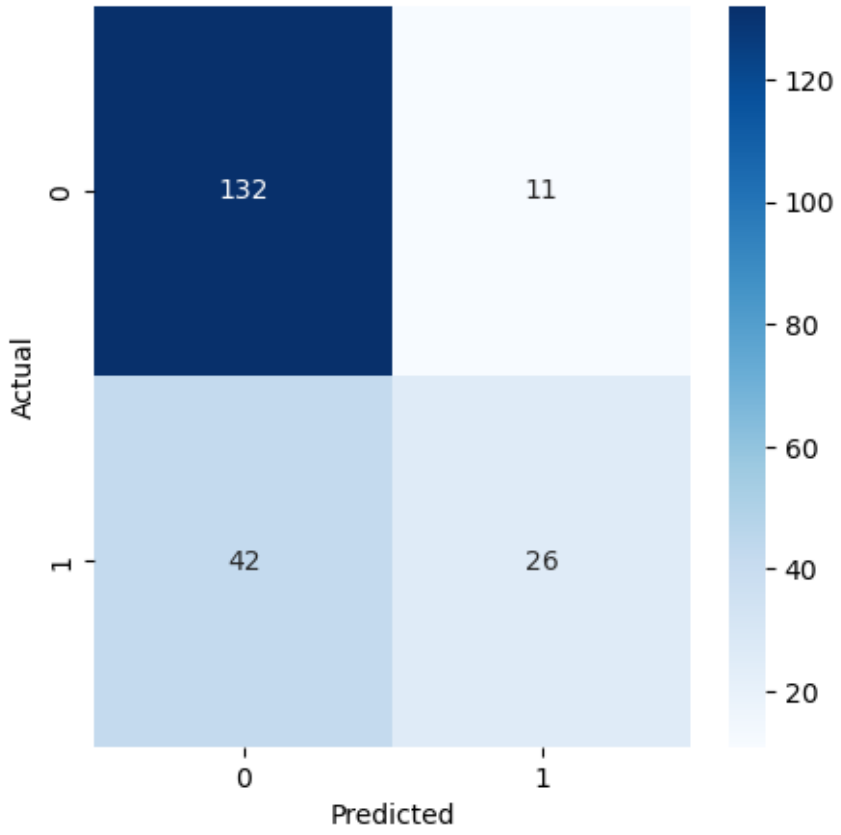
[90]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=4, min_samples_leaf=4)
```

Q5. Evaluate the performance of the decision tree model on the test set using metrics such as accuracy, precision, recall, and F1 score. Use confusion matrices and ROC curves to visualize the results.

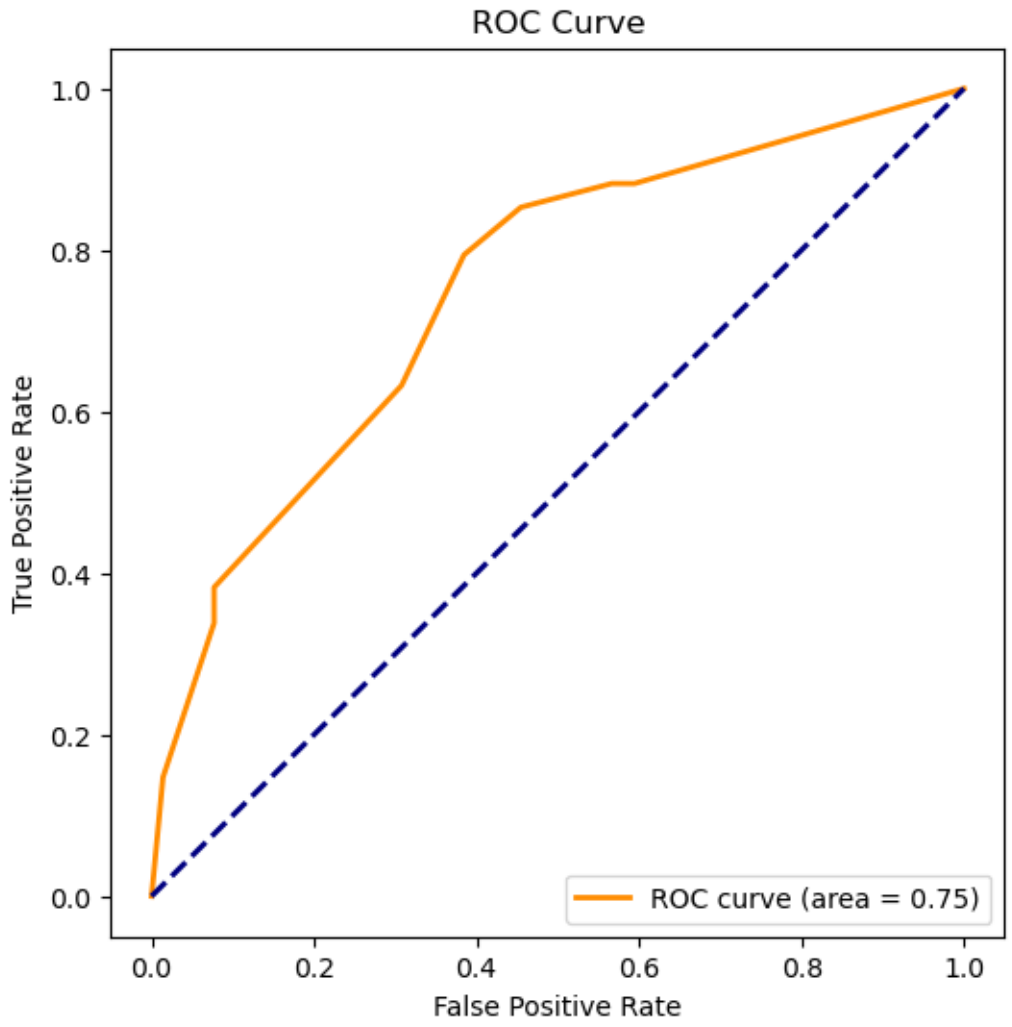
[91]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, roc_curve, auc, confusion_matrix

# Predict the class labels of the test set
y_pred = tree.predict(X_test)

# Calculate the evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print the evaluation metrics
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Accuracy: 0.7488151658767772
Precision: 0.7027027027027027
Recall: 0.38235294117647056
F1 Score: 0.4952380952380953
```

[92]:
```
# Plot the confusion matrix
conf_mat = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5, 5))
sns.heatmap(conf_mat, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



[93]:
```
# Plot the ROC curve
y_prob = tree.predict_proba(X_test)[:,1]
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
plt.figure(figsize=(6, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %␣
 ↪roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```



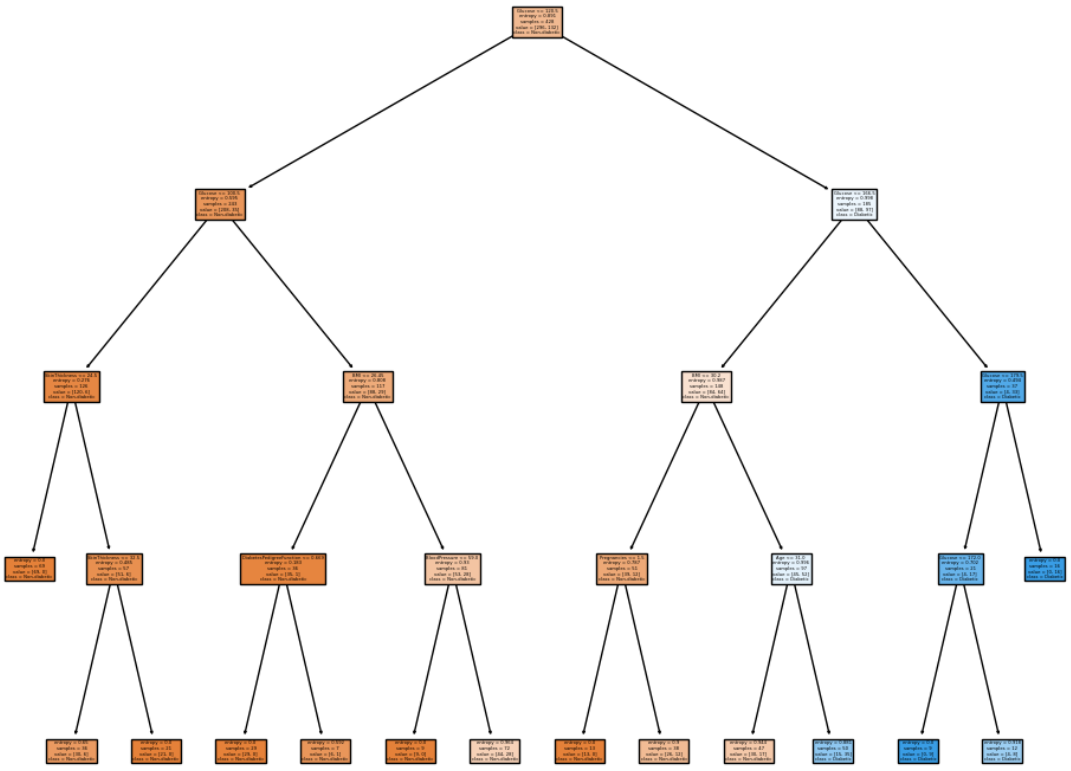
Q6. Interpret the decision tree by examining the splits, branches, and leaves. Identify the most important variables and their thresholds. Use domain knowledge and common sense to explain the patterns and trends.

[94]:
```
from sklearn.tree import plot_tree

# Visualize the decision tree
plt.figure(figsize=(12,10))
plot_tree(tree, feature_names=X_train.columns, class_names=['Non-diabetic',␣
↪'Diabetic'], filled=True)
```

[94]:
```
[Text(0.5, 0.9, 'Glucose <= 120.5\nentropy = 0.891\nsamples = 428\nvalue = [296,
132]\nclass = Non-diabetic'),
 Text(0.21153846153846154, 0.7, 'Glucose <= 100.5\nentropy = 0.595\nsamples =
243\nvalue = [208, 35]\nclass = Non-diabetic'),
```

```
 Text(0.07692307692307693, 0.5, 'SkinThickness <= 24.5\nentropy = 0.276\nsamples
= 126\nvalue = [120, 6]\nclass = Non-diabetic'),
 Text(0.038461538461538464, 0.3, 'entropy = 0.0\nsamples = 69\nvalue = [69,
0]\nclass = Non-diabetic'),
 Text(0.11538461538461539, 0.3, 'SkinThickness <= 32.5\nentropy = 0.485\nsamples
= 57\nvalue = [51, 6]\nclass = Non-diabetic'),
 Text(0.07692307692307693, 0.1, 'entropy = 0.65\nsamples = 36\nvalue = [30,
6]\nclass = Non-diabetic'),
 Text(0.15384615384615385, 0.1, 'entropy = 0.0\nsamples = 21\nvalue = [21,
0]\nclass = Non-diabetic'),
 Text(0.34615384615384615, 0.5, 'BMI <= 26.45\nentropy = 0.808\nsamples =
117\nvalue = [88, 29]\nclass = Non-diabetic'),
 Text(0.2692307692307692, 0.3, 'DiabetesPedigreeFunction <= 0.669\nentropy =
0.183\nsamples = 36\nvalue = [35, 1]\nclass = Non-diabetic'),
 Text(0.23076923076923078, 0.1, 'entropy = 0.0\nsamples = 29\nvalue = [29,
0]\nclass = Non-diabetic'),
 Text(0.3076923076923077, 0.1, 'entropy = 0.592\nsamples = 7\nvalue = [6,
1]\nclass = Non-diabetic'),
 Text(0.4230769230769231, 0.3, 'BloodPressure <= 59.0\nentropy = 0.93\nsamples =
81\nvalue = [53, 28]\nclass = Non-diabetic'),
 Text(0.38461538461538464, 0.1, 'entropy = 0.0\nsamples = 9\nvalue = [9,
0]\nclass = Non-diabetic'),
 Text(0.46153846153846156, 0.1, 'entropy = 0.964\nsamples = 72\nvalue = [44,
28]\nclass = Non-diabetic'),
 Text(0.7884615384615384, 0.7, 'Glucose <= 166.5\nentropy = 0.998\nsamples =
185\nvalue = [88, 97]\nclass = Diabetic'),
 Text(0.6538461538461539, 0.5, 'BMI <= 30.2\nentropy = 0.987\nsamples =
148\nvalue = [84, 64]\nclass = Non-diabetic'),
 Text(0.5769230769230769, 0.3, 'Pregnancies <= 1.5\nentropy = 0.787\nsamples =
51\nvalue = [39, 12]\nclass = Non-diabetic'),
 Text(0.5384615384615384, 0.1, 'entropy = 0.0\nsamples = 13\nvalue = [13,
0]\nclass = Non-diabetic'),
 Text(0.6153846153846154, 0.1, 'entropy = 0.9\nsamples = 38\nvalue = [26,
12]\nclass = Non-diabetic'),
 Text(0.7307692307692307, 0.3, 'Age <= 31.0\nentropy = 0.996\nsamples =
97\nvalue = [45, 52]\nclass = Diabetic'),
 Text(0.6923076923076923, 0.1, 'entropy = 0.944\nsamples = 47\nvalue = [30,
17]\nclass = Non-diabetic'),
 Text(0.7692307692307693, 0.1, 'entropy = 0.881\nsamples = 50\nvalue = [15,
35]\nclass = Diabetic'),
 Text(0.9230769230769231, 0.5, 'Glucose <= 179.5\nentropy = 0.494\nsamples =
37\nvalue = [4, 33]\nclass = Diabetic'),
 Text(0.8846153846153846, 0.3, 'Glucose <= 172.0\nentropy = 0.702\nsamples =
21\nvalue = [4, 17]\nclass = Diabetic'),
 Text(0.8461538461538461, 0.1, 'entropy = 0.0\nsamples = 9\nvalue = [0,
9]\nclass = Diabetic'),
 Text(0.9230769230769231, 0.1, 'entropy = 0.918\nsamples = 12\nvalue = [4,
```

```
8]\nclass = Diabetic'),
 Text(0.9615384615384616, 0.3, 'entropy = 0.0\nsamples = 16\nvalue = [0,
16]\nclass = Diabetic')]
```



Q7. Validate the decision tree model by applying it to new data or testing its robustness to changes in the dataset or the environment. Use sensitivity analysis and scenario testing to explore the uncertainty and risks.

Ans. Validating the decision tree model is an important step to ensure that it is robust and reliable when applied to new data or in different scenarios. Here are some ways to validate the decision tree model:

Test the model on a holdout dataset: We can further validate the decision tree model by testing it on a completely new dataset that was not used for training or tuning the hyperparameters. This will help us to determine if the model is overfitting to the training data and generalize well to new data.

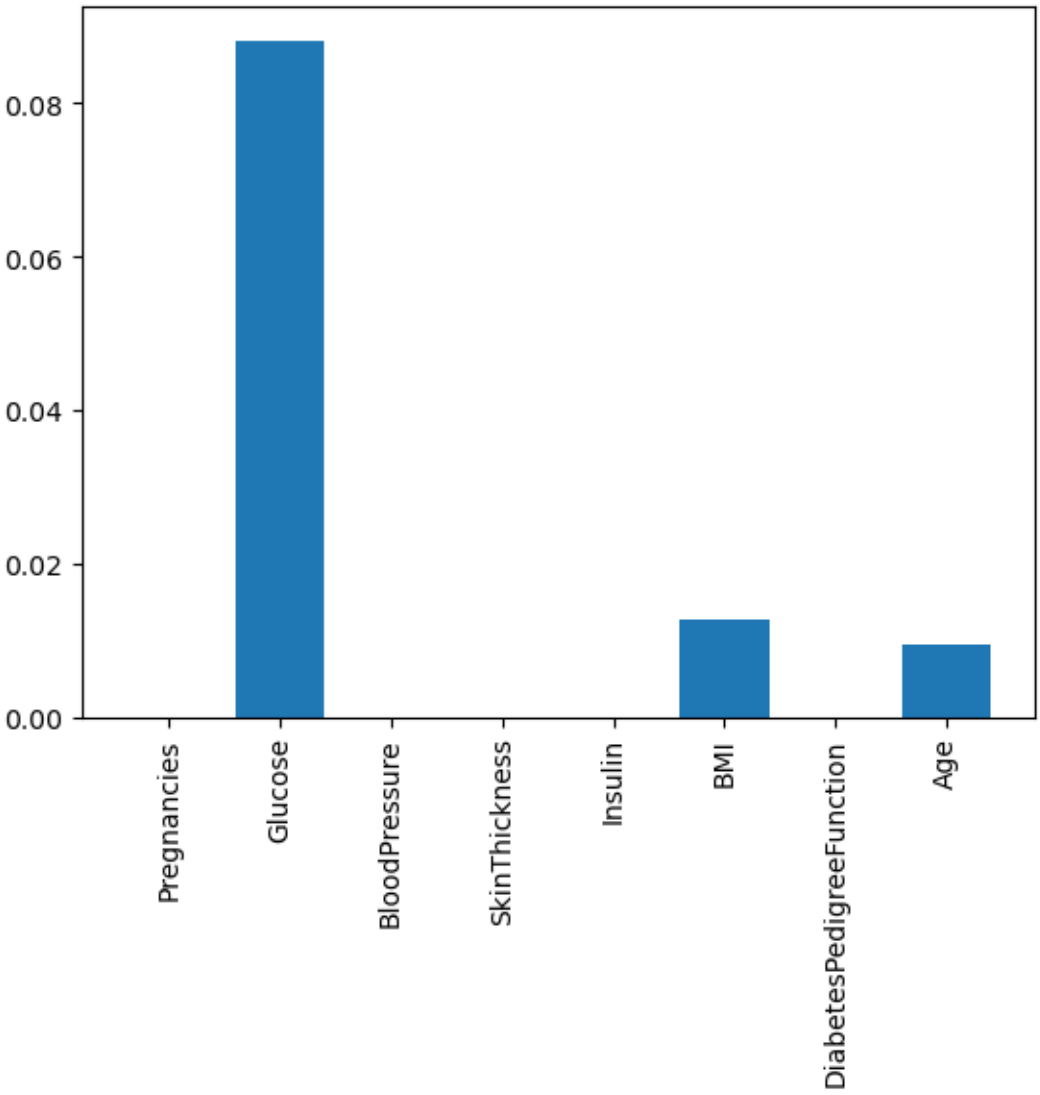
Perform sensitivity analysis: Sensitivity analysis involves testing the model's response to small changes in the input variables. We can use this technique to identify which variables have the most impact on the model's predictions, and to identify potential weaknesses or uncertainties in the model. For example, we can simulate different scenarios by varying the values of the input variables and observing the changes in the model's predictions.

Perform scenario testing: Scenario testing involves testing the model's performance under different conditions or assumptions. For example, we can simulate scenarios where certain variables are missing or where the distribution of the variables is different from what was observed in the training data. This will help us to identify potential risks or limitations of the model in different scenarios.

Compare the model's performance to other models: We can also validate the decision tree model by comparing its performance to other models that are commonly used for predicting diabetes, such as logistic regression, random forests, or neural networks. This will help us to determine if the decision tree model is the best choice for the particular application, and if there are alternative models that can provide better performance.

In summary, validating the decision tree model is a crucial step to ensure that it is robust and reliable when applied to new data or in different scenarios. By performing sensitivity analysis, scenario testing, and comparing the model's performance to other models, we can gain a better understanding of the strengths and weaknesses of the model, and make informed decisions about its use.

[95]:
```
from sklearn.inspection import permutation_importance

# Compute the feature importances using permutation importance
result = permutation_importance(tree, X_test, y_test, n_repeats=10,␣
 ↪random_state=42, n_jobs=-1)

# Plot the feature importances
plt.bar(X.columns, result.importances_mean)
plt.xticks(rotation=90)
plt.show()
```


0.08
0.06
0.04
0.02
0.00
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age