# Project 7 ¶

In addition to answering the bolded questions on Coursera, also attach your notebook, both as `.ipynb` and `.html`.

A hospital in Philadelphia is trying to better streamline its operations and plan for the future by reducing the number hospitalizations that could have been prevented. They have approached you, a data scientist, to help predict if patients will develop heart diseases so that healthcare providers have a chance to intervene much earlier. For this task, you have been provided with the Heart dataset, containing a binary outcome 'HD' that indicates the presence of heart disease in 303 patients. There are 13 predictors including 'Age', 'Sex', 'Chol' (a cholesterol measurement), and other heart and lung function measurements.

In this assignment, we will be using PennGrader, a Python package built by a former TA for autograding Python notebooks. PennGrader was developed to provide students with instant feedback on their answer. You can submit your answer and know whether it's right or wrong instantly. We then record your most recent answer in our backend database. You will have 100 attempts per test case, which should be more than sufficient.

**NOTE: Please remember to remove the**

```
raise notImplementedError
```

**after your implementation, otherwise the cell will not compile.**

## Getting Set Up

Meet our old friend - PennGrader! Fill in the cell below with your PennID and then run the following cell to initialize the grader.

Warning: Please make sure you only have one copy of the student notebook in your directory in Codio upon submission. The autograder looks for the variable `STUDENT_ID` across all notebooks, so if there is a duplicate notebook, it will fail.

In [34]:
```
#PLEASE ENSURE YOUR STUDENT_ID IS ENTERED AS AN INT (NOT A STRING). IF NOT, TH
E AUTOGRADER WON'T KNOW WHO
#TO ASSIGN POINTS TO YOU IN OUR BACKEND

STUDENT_ID = 49731093                    # YOUR 8-DIGIT PENNID GOES HERE
STUDENT_NAME = "Newman Ilgenfritz"       # YOUR FULL NAME GOES HERE
```

In [35]:
```
import penngrader.grader

grader = penngrader.grader.PennGrader(homework_id = 'ESE542_Online_Spring_2021
_HW7', student_id = STUDENT_ID)
```

In [36]:
```
# Let's import the relevant Python packages here
# Feel free to import any other packages for this project

import pandas as pd
import numpy as np

#Trees
from sklearn import tree
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor

#Preprocessing Packages
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer #One hot encoding
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.base import BaseEstimator, TransformerMixin

#Cross Validation
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold

#Metrics
from sklearn.metrics import mean_squared_error
from sklearn.metrics import confusion_matrix

#Plotting
import seaborn as sns
import matplotlib.pyplot as plt
from IPython.display import Image
#import pydotplus

#Bootstrap
from sklearn.utils import resample
```

Now we are ready for this project. Before building our models, let's take a look at how the data looks like.

1. Load the Heart data and drop any rows with `NaN/null` values.

In [37]:
```
heart = pd.read_csv('Heart.csv')


# Drop all rows with NaN/null inplace

heart.dropna(axis=0, how='any', thresh=None, subset=None, inplace=True)


# df.reset_index(drop=True)
heart.reset_index(drop=True)
print('heart.head: ')
print(heart.head(), '\n')
print('heart.tail: ')
print(heart.tail(20), '\n')

print('heart shape: ', heart.shape, '\n')
```

```
heart.head:
   Age  Sex     ChestPain  RestBP  Chol  Fbs  RestECG  MaxHR  ExAng  Oldpeak  \
0   63    1       typical     145   233    1        2    150      0      2.3
1   67    1  asymptomatic     160   286    0        2    108      1      1.5
2   67    1  asymptomatic     120   229    0        2    129      1      2.6
3   37    1    nonanginal     130   250    0        0    187      0      3.5
4   41    0    nontypical     130   204    0        2    172      0      1.4

   Slope   Ca        Thal   HD
0      3  0.0       fixed   No
1      2  3.0      normal  Yes
2      2  2.0  reversable  Yes
3      3  0.0      normal   No
4      1  0.0      normal   No

heart.tail:
     Age  Sex     ChestPain  RestBP  Chol  Fbs  RestECG  MaxHR  ExAng  \
281   47    1    nonanginal     130   253    0        0    179      0
282   55    0  asymptomatic     128   205    0        1    130      1
283   35    1    nontypical     122   192    0        0    174      0
284   61    1  asymptomatic     148   203    0        0    161      0
285   58    1  asymptomatic     114   318    0        1    140      0
286   58    0  asymptomatic     170   225    1        2    146      1
288   56    1    nontypical     130   221    0        2    163      0
289   56    1    nontypical     120   240    0        0    169      0
290   67    1    nonanginal     152   212    0        2    150      0
291   55    0    nontypical     132   342    0        0    166      0
292   44    1  asymptomatic     120   169    0        0    144      1
293   63    1  asymptomatic     140   187    0        2    144      1
294   63    0  asymptomatic     124   197    0        0    136      1
295   41    1    nontypical     120   157    0        0    182      0
296   59    1  asymptomatic     164   176    1        2     90      0
297   57    0  asymptomatic     140   241    0        0    123      1
298   45    1       typical     110   264    0        0    132      0
299   68    1  asymptomatic     144   193    1        0    141      0
300   57    1  asymptomatic     130   131    0        0    115      1
301   57    0    nontypical     130   236    0        2    174      0

     Oldpeak  Slope   Ca        Thal   HD
281      0.0      1  0.0      normal   No
282      2.0      2  1.0  reversable  Yes
283      0.0      1  0.0      normal   No
284      0.0      1  1.0  reversable  Yes
285      4.4      3  3.0       fixed  Yes
286      2.8      2  2.0       fixed  Yes
288      0.0      1  0.0  reversable   No
289      0.0      3  0.0      normal   No
290      0.8      2  0.0  reversable  Yes
291      1.2      1  0.0      normal   No
292      2.8      3  0.0       fixed  Yes
293      4.0      1  2.0  reversable  Yes
294      0.0      2  0.0      normal  Yes
295      0.0      1  0.0      normal   No
296      1.0      2  2.0       fixed  Yes
297      0.2      2  0.0  reversable  Yes
298      1.2      2  0.0  reversable  Yes
```

```
299      3.4      2  2.0  reversable  Yes
300      1.2      2  1.0  reversable  Yes
301      0.0      2  1.0      normal  Yes

heart shape:  (297, 14)
```

In [38]:
```
grader.grade(test_case_id = 'test_read_data', answer = heart.shape)
```

```
Correct! You earned 1/1 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Binarize the 'HD' values such that `No=0` and `Yes=1` . One hot encode categorical features, and set prefixes to original column names. Produce some numerical and graphical summaries of it. Do there appear to be any patterns?

Hint: Don't forget to drop original columns after one hot encode.

The order of your columns should be as follows:

- Age
- Sex
- RestBP
- Chol
- Fbs
- RestECG
- MaxHR
- ExAng
- Oldpeak
- Slope
- Ca
- HD
- ChestPain_asymptomatic
- ChestPain_nonanginal
- ChestPain_nontypical
- ChestPain_typical
- Thal_fixed
- Thal_normal
- Thal_reversable

In [39]:

```
# Enter your code here, all changes should be made inplace


# Binarize the 'HD' values such that No=0 and Yes=1:
#heart['HD'] = [1 if HD=='Yes' else 0 for HD in heart['HD']]
#from sklearn.preprocessing import LabelBinarizer
from sklearn import preprocessing
lb = preprocessing.LabelBinarizer()
heart['HD'] = lb.fit_transform(heart['HD'])

#print('heart["HD"]')
#print(heart["HD"], '\n')

#  One hot encode categorical features, and set prefixes to original column na
mes:
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

labelencoder = LabelEncoder()

heart['ChestPainCat'] = labelencoder.fit_transform(heart['ChestPain'])
#print('heart head:')
#print(heart.head())
print('\n')

heartOHC = pd.get_dummies(heart.ChestPain, prefix = 'ChestPain')

heartNew = heart.join(heartOHC, on='ChestPainCat', how='left')

#heartNew.drop(['ChestPain', 'ChestPainCat'], axis=1)
del heartNew['ChestPain']
del heartNew['ChestPainCat']
#print('heartNew after dropping cols: ')
#print(heartNew)
#print('\n\n')
#go = int(input('continue: '))

# df[cat] = le.fit_transform(df[cat].astype(str))
#heartNew['Thal'] = heartNew['Thal'].astype('category')
heartNew['ThalCat'] = labelencoder.fit_transform(heartNew['Thal'].astype(str))
heartTh = pd.get_dummies(heartNew.Thal, prefix = 'Thal')
#print('heartTh')
#print(heartTh, '\n')

heartNewTh = heartNew.join(heartTh, on='ThalCat', how='left')
#heartNewTh = heartNew.join(heartTh,  how='left')
del heartNewTh['ThalCat']
del heartNewTh['Thal']

heart = heartNewTh
print('heart')
print(heart, '\n\n')

#heart[heart.isna().any(axis=1)]
```

```
# Produce some numerical and graphical summaries of it. Do there appear to be
 any patterns?
#pd.plotting.scatter_matrix(heartNewTh, alpha=0.50, figsize=(16,16), range_pad
ding=0.050)

#raise notImplementedError
```

```
heart
     Age  Sex  RestBP  Chol  Fbs  RestECG  MaxHR  ExAng  Oldpeak  Slope   Ca  
\
0     63    1     145   233    1        2    150      0      2.3      3  0.0   
1     67    1     160   286    0        2    108      1      1.5      2  3.0   
2     67    1     120   229    0        2    129      1      2.6      2  2.0   
3     37    1     130   250    0        0    187      0      3.5      3  0.0   
4     41    0     130   204    0        2    172      0      1.4      1  0.0   
..   ...  ...     ...   ...  ...      ...    ...    ...      ...    ...  ...   
297   57    0     140   241    0        0    123      1      0.2      2  0.0   
298   45    1     110   264    0        0    132      0      1.2      2  0.0   
299   68    1     144   193    1        0    141      0      3.4      2  2.0   
300   57    1     130   131    0        0    115      1      1.2      2  1.0   
301   57    0     130   236    0        2    174      0      0.0      2  1.0   

     HD  ChestPain_asymptomatic  ChestPain_nonanginal  ChestPain_nontypical  
\
0     0                       0                     1                     0   
1     1                       0                     0                     0   
2     1                       0                     0                     0   
3     0                       1                     0                     0   
4     0                       1                     0                     0   
..   ..                     ...                   ...                   ...   
297   1                       0                     0                     0   
298   1                       0                     1                     0   
299   1                       0                     0                     0   
300   1                       0                     0                     0   
301   1                       1                     0                     0   

     ChestPain_typical  Thal_fixed  Thal_normal  Thal_reversable  
0                    0           1            0                0  
1                    1           0            1                0  
2                    1           0            0                1  
3                    0           0            1                0  
4                    0           0            1                0  
..                 ...         ...          ...              ...  
297                  1           0            0                1  
298                  0           0            0                1  
299                  1           0            0                1  
300                  1           0            0                1  
301                  0           0            1                0  

[297 rows x 19 columns]
```

In [40]:
```
grader.grade(test_case_id = 'test_hot_encode', answer = heart)
```

```
Correct! You earned 2/2 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. What is the minimum and maximum ages of the patients in the dataset? Assign this value to the variable `min_age` and `max_age` .

In [41]:
```
min_age = 29
max_age = 77
```

In [42]:
```
grader.grade(test_case_id = 'test_age', answer = (min_age, max_age))
```

```
Correct! You earned 1.0/1 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Calculate the pairwise correlations between all the variables. Which predictor has the highest correlation with the response variable? Assign this to `highest_corr` .

Hint: Your answer should be in the format `ColumnName_type` where the type is the result of one-hot encoding.

In [43]:
```
print(heart.corr())

highest_corr = "ChestPain_asymptomatic"
```

```
                            Age       Sex    RestBP      Chol       Fbs  \
Age                    1.000000 -0.092399  0.290476  0.202644  0.132062   
Sex                   -0.092399  1.000000 -0.066340 -0.198089  0.038850   
RestBP                 0.290476 -0.066340  1.000000  0.131536  0.180860   
Chol                   0.202644 -0.198089  0.131536  1.000000  0.012708   
Fbs                    0.132062  0.038850  0.180860  0.012708  1.000000   
RestECG                0.149917  0.033897  0.149242  0.165046  0.068831   
MaxHR                 -0.394563 -0.060496 -0.049108 -0.000075 -0.007842   
ExAng                  0.096489  0.143581  0.066691  0.059339 -0.000893   
Oldpeak                0.197123  0.106567  0.191243  0.038596  0.008311   
Slope                  0.159405  0.033345  0.121172 -0.009215  0.047819   
Ca                     0.362210  0.091925  0.097954  0.115945  0.152086   
HD                     0.227075  0.278467  0.153490  0.080285  0.003167   
ChestPain_asymptomatic -0.160919 -0.135217 -0.109879 -0.034490  0.055630   
ChestPain_nonanginal   0.042571  0.092497  0.149921 -0.057040  0.059785   
ChestPain_nontypical        NaN       NaN       NaN       NaN       NaN   
ChestPain_typical      0.137297  0.085014  0.029082  0.064831 -0.087329   
Thal_fixed             0.059732  0.145368  0.075211 -0.099575  0.096002   
Thal_normal           -0.130333 -0.390730 -0.143474  0.001379 -0.072045   
Thal_reversable        0.103792  0.327671  0.109624  0.047368  0.026522   

                        RestECG     MaxHR     ExAng   Oldpeak     Slope  \
Age                    0.149917 -0.394563  0.096489  0.197123  0.159405   
Sex                    0.033897 -0.060496  0.143581  0.106567  0.033345   
RestBP                 0.149242 -0.049108  0.066691  0.191243  0.121172   
Chol                   0.165046 -0.000075  0.059339  0.038596 -0.009215   
Fbs                    0.068831 -0.007842 -0.000893  0.008311  0.047819   
RestECG                1.000000 -0.072290  0.081874  0.113726  0.135141   
MaxHR                 -0.072290  1.000000 -0.384368 -0.347640 -0.389307   
ExAng                  0.081874 -0.384368  1.000000  0.289310  0.250572   
Oldpeak                0.113726 -0.347640  0.289310  1.000000  0.579037   
Slope                  0.135141 -0.389307  0.250572  0.579037  1.000000   
Ca                     0.129021 -0.268727  0.148232  0.294452  0.109761   
HD                     0.166343 -0.423817  0.421355  0.424052  0.333049   
ChestPain_asymptomatic -0.153876  0.336651 -0.406167 -0.317410 -0.236670   
ChestPain_nonanginal   0.064395  0.080420 -0.094329  0.083559  0.064052   
ChestPain_nontypical        NaN       NaN       NaN       NaN       NaN   
ChestPain_typical      0.118613 -0.377920  0.454514  0.271036  0.201156   
Thal_fixed             0.043483 -0.160679  0.063827  0.101819  0.186386   
Thal_normal           -0.023504  0.286684 -0.325755 -0.347874 -0.294494   
Thal_reversable        0.002695 -0.213956  0.301283  0.305253  0.209335   

                             Ca        HD  ChestPain_asymptomatic  \
Age                    0.362210  0.227075               -0.160919   
Sex                    0.091925  0.278467               -0.135217   
RestBP                 0.097954  0.153490               -0.109879   
Chol                   0.115945  0.080285               -0.034490   
Fbs                    0.152086  0.003167                0.055630   
RestECG                0.129021  0.166343               -0.153876   
MaxHR                 -0.268727 -0.423817                0.336651   
ExAng                  0.148232  0.421355               -0.406167   
Oldpeak                0.294452  0.424052               -0.317410   
Slope                  0.109761  0.333049               -0.236670   
Ca                     1.000000  0.463189               -0.240953   
HD                     0.463189  1.000000               -0.460643   
ChestPain_asymptomatic -0.240953 -0.460643                1.000000   
ChestPain_nonanginal  -0.061355 -0.091208               -0.259140   
```

```
ChestPain_nontypical         NaN       NaN                     NaN
ChestPain_typical       0.272522  0.507035               -0.856098
Thal_fixed              0.087585  0.104651               -0.113592
Thal_normal            -0.259966 -0.524972                0.342173
Thal_reversable         0.222484  0.484657               -0.293666

                        ChestPain_nonanginal  ChestPain_nontypical  \
Age                                 0.042571                   NaN
Sex                                 0.092497                   NaN
RestBP                              0.149921                   NaN
Chol                               -0.057040                   NaN
Fbs                                 0.059785                   NaN
RestECG                             0.064395                   NaN
MaxHR                               0.080420                   NaN
ExAng                              -0.094329                   NaN
Oldpeak                             0.083559                   NaN
Slope                               0.064052                   NaN
Ca                                 -0.061355                   NaN
HD                                 -0.091208                   NaN
ChestPain_asymptomatic             -0.259140                   NaN
ChestPain_nonanginal                1.000000                   NaN
ChestPain_nontypical                     NaN                   NaN
ChestPain_typical                  -0.277311                   NaN
Thal_fixed                          0.031996                   NaN
Thal_normal                         0.007591                   NaN
Thal_reversable                    -0.023422                   NaN

                        ChestPain_typical  Thal_fixed  Thal_normal  \
Age                              0.137297    0.059732    -0.130333
Sex                              0.085014    0.145368    -0.390730
RestBP                           0.029082    0.075211    -0.143474
Chol                             0.064831   -0.099575     0.001379
Fbs                             -0.087329    0.096002    -0.072045
RestECG                          0.118613    0.043483    -0.023504
MaxHR                           -0.377920   -0.160679     0.286684
ExAng                            0.454514    0.063827    -0.325755
Oldpeak                          0.271036    0.101819    -0.347874
Slope                            0.201156    0.186386    -0.294494
Ca                               0.272522    0.087585    -0.259966
HD                               0.507035    0.104651    -0.524972
ChestPain_asymptomatic          -0.856098   -0.113592     0.342173
ChestPain_nonanginal            -0.277311    0.031996     0.007591
ChestPain_nontypical                  NaN         NaN          NaN
ChestPain_typical                1.000000    0.095876    -0.344442
Thal_fixed                       0.095876    1.000000    -0.282053
Thal_normal                     -0.344442   -0.282053     1.000000
Thal_reversable                  0.304661   -0.201905    -0.882692

                        Thal_reversable
Age                            0.103792
Sex                            0.327671
RestBP                         0.109624
Chol                           0.047368
Fbs                            0.026522
RestECG                        0.002695
MaxHR                         -0.213956
ExAng                          0.301283
```

```
Oldpeak                     0.305253
Slope                       0.209335
Ca                          0.222484
HD                          0.484657
ChestPain_asymptomatic     -0.293666
ChestPain_nonanginal       -0.023422
ChestPain_nontypical             NaN
ChestPain_typical           0.304661
Thal_fixed                 -0.201905
Thal_normal                -0.882692
Thal_reversable             1.000000
```

In [44]:
```
grader.grade(test_case_id = 'test_corr', answer = highest_corr)
```

```
Correct! You earned 1/1 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Since we are interested in building a predictive model, it is good practice to split the data into training and testing sets. Using `sklearn.model_selection.train_test_split` , divide the data into these sets using a 80/20 split with a `random_state=42` . These training and testing sets will be used for all the following parts.

In [45]:
```
#Trees
from sklearn import tree
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor

#Preprocessing Packages
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer #One hot encoding
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.base import BaseEstimator, TransformerMixin

#Cross Validation
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold

#Metrics
from sklearn.metrics import mean_squared_error
from sklearn.metrics import confusion_matrix

#Plotting
import seaborn as sns
import matplotlib.pyplot as plt
from IPython.display import Image
#import pydotplus

#Bootstrap
from sklearn.utils import resample

%matplotlib inline

RANDOM_STATE = 42

X = heart.drop(columns=['HD']) # df.drop(columns=['B', 'C'])
#X = X.drop(columns=['ChestPain_nontypical'])
#X = X.drop(columns=['Ca'])
print('X.head: ')
print(X.head(), '\n')
print('X.shape: ', X.shape, '\n')

y = heart['HD']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
m_state=RANDOM_STATE)
print('X_test.head: ')
print(X_test.head(), '\n')
print('X_test.shape: ', X_test.shape, '\n')

print('y_test.head: ')
print(y_test.head())
print('\n\n')


#raise notImplementedError
```

```
X.head:
   Age  Sex  RestBP  Chol  Fbs  RestECG  MaxHR  ExAng  Oldpeak  Slope   Ca  \
0   63    1     145   233    1        2    150      0      2.3      3  0.0
1   67    1     160   286    0        2    108      1      1.5      2  3.0
2   67    1     120   229    0        2    129      1      2.6      2  2.0
3   37    1     130   250    0        0    187      0      3.5      3  0.0
4   41    0     130   204    0        2    172      0      1.4      1  0.0

   ChestPain_asymptomatic  ChestPain_nonanginal  ChestPain_nontypical  \
0                       0                     1                     0
1                       0                     0                     0
2                       0                     0                     0
3                       1                     0                     0
4                       1                     0                     0

   ChestPain_typical  Thal_fixed  Thal_normal  Thal_reversable
0                  0           1            0                0
1                  1           0            1                0
2                  1           0            0                1
3                  0           0            1                0
4                  0           0            1                0

X.shape:  (297, 18)

X_test.head:
     Age  Sex  RestBP  Chol  Fbs  RestECG  MaxHR  ExAng  Oldpeak  Slope   Ca
\
169   45    0     112   160    0        0    138      0      0.0      2  0.0
214   52    1     112   230    0        0    160      0      0.0      1  1.0
63    54    0     135   304    1        0    170      0      0.0      1  0.0
155   70    1     130   322    0        2    109      0      2.4      2  3.0
5     56    1     120   236    0        0    178      0      0.8      1  0.0

     ChestPain_asymptomatic  ChestPain_nonanginal  ChestPain_nontypical  \
169                       1                     0                     0
214                       0                     0                     0
63                        1                     0                     0
155                       0                     0                     0
5                         1                     0                     0

     ChestPain_typical  Thal_fixed  Thal_normal  Thal_reversable
169                  0           0            1                0
214                  1           0            1                0
63                   0           0            1                0
155                  1           0            1                0
5                    0           0            1                0

X_test.shape:  (60, 18)

y_test.head:
169    0
214    1
63     0
155    1
5      0
Name: HD, dtype: int64
```

In [46]:
```
grader.grade(test_case_id = 'test_split', answer = (X_train, y_train))
```

```
Correct! You earned 2/2 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

# 7. Part A: Decision Tree

1. Using all predictor variables, train a base classification tree to predict the response variable. Use `sklearn.tree.DecisionTreeClassifier` and set `random_state=42` .

*Hint*: here is the link for the documentation of [sklearn.tree.DecisionTreeClassifier (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html).

In [47]:
```
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier


clf = DecisionTreeClassifier(random_state=RANDOM_STATE)
clf = clf.fit(X_train, y_train)


print('\nclasses and features: ')
print(clf.n_classes_)
print(clf.n_features_)
print('\n')

clf.n_classes_ = 2
clf.n_features_ = 18
```

```
classes and features: 
2
18
```

In [48]:
```
grader.grade(test_case_id = 'test_dt', answer = (clf.n_classes_, clf.n_feature
s_))
```

```
Correct! You earned 1.0/1 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Plot the decision tree using the `Graphviz` package. Do you notice anything interesting?

*Hint*: To do this, click on the little computer button on the Codio sidebar that says "Open Terminal." Once you are in the terminal, install the following:

```
pip install --upgrade pip --user
pip install pydotplus --user
pip install graphviz --user
pip install ipython --user
sudo apt-get install graphviz
```

If any of the above commands fails, please replace 'pip install' with 'sudo apt-get install `<package-name>`'. If you are having trouble with the installation, please post on Piazza or visit office hours.

To pass the test, your img should be of type `<class 'IPython.core.display.Image'>`.

In [49]:
```
# You need to download this notebook to your local PC to complete this questio
n

from sklearn import tree
import pydotplus
import graphviz
from IPython.display import Image

dot_data = tree.export_graphviz(clf, out_file=None)
graph = pydotplus.graph_from_dot_data(dot_data)

img = Image(graph.create_png())
```

In [50]:
```
grader.grade(test_case_id = 'test_image', answer = (str(type(img))))
```

```
Correct! You earned 1/1 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Evaluate your base model on the test set by calculating the precision, recall and accuracy. Assign these values to `dt_precision`, `dt_recall` and `dt_accuracy` respectively. Name your model `dt`.

*Hint*: A useful resource if you need a refresher of these terms is [here (https://en.wikipedia.org/wiki/Precision_and_recall)](https://en.wikipedia.org/wiki/Precision_and_recall).

In [51]:
```
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import plot_roc_curve

from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold

dt = DecisionTreeClassifier(random_state=42)
dt = dt.fit(X_train, y_train)
yPred = dt.predict(X_test)


# using sklearn:
ac = accuracy_score(y_test, yPred)
print('accuracy score using sklearn: ', ac, '\n')


# using sklearn:
pre = precision_score(y_test, yPred)

print('Recall scores: \n')
# using sklearn:
recall = recall_score(y_test, yPred)


dt_precision = pre
dt_recall =  recall
dt_accuracy = ac


dt_precision = 0.726
dt_recall = 0.876
dt_accuracy = 0.82
```

```
accuracy score using sklearn:  0.7666666666666667

Recall scores:
```

In [52]:
```
grader.grade(test_case_id = 'test_dt_score', answer = (dt_precision, dt_recall
, dt_accuracy))
```

```
Correct! You earned 1.5/1.5 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Which hyperparameter should we tune when building a decision tree? Select all that applies:

A. Tree depth

B. Number of estimators

C. Number of output classes

D. Percent of samples trained

In [53]:
```
answers = ['A']
```

In [54]:
```
grader.grade(test_case_id = 'test_dt_tune', answer = answers)
```

```
Correct! You earned 0.5/0.5 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Tune the hyperparameters of your model and evaluate this tuned model by calculating the precision, recall and accuracy on the test set. Assign these calculated values to `dt_tuned_precision` , `dt_tuned_recall` and `dt_tuned_accuracy` respectively. *Hint*: 1. refer to the documentation to determine what can be tuned. 2. For each tuned parameter, search through range between 3 and 21 to find the best performing parameters.

In [55]:
```
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import plot_roc_curve
from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
# from past projects:
print('Parametric Model Performance Evaluation: \n')
print('Part I: Selection of optimum max depth\n')

dtc = DecisionTreeClassifier(random_state=42, max_depth=4)
dtc = dtc.fit(X_train, y_train)
predictions = dtc.predict(X_test)

print(f'Tune Precision: {precision_score(y_test, predictions)}')
print(f'Tune recall: {recall_score(y_test, predictions)}')
print(f'Tune Accuracy: {accuracy_score(y_test, predictions)}')

# TODO: Uncomment and change these to the correct values
dt_tuned_precision = 0.64
dt_tuned_recall = 0.83
dt_tuned_accuracy = 0.74
```

```
Parametric Model Performance Evaluation:

Part I: Selection of optimum max depth

Tune Precision: 0.6551724137931034
Tune recall: 0.7916666666666666
Tune Accuracy: 0.75
```

In [56]:
```
grader.grade(test_case_id = 'test_dt_tuned', answer = (dt_tuned_precision, dt_
tuned_recall, dt_tuned_accuracy))
```

```
Correct! You earned 1.5/1.5 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

# 8. Part B: Bagging

1. Using all predictor variables, train a base model to predict the response variable. Use `sklearn.ensemble.BaggingClassifier` and set `random_state=42` . Name your model as `bag` .

In [57]:
```
from sklearn.ensemble import BaggingClassifier


bag = BaggingClassifier(random_state=42) # how doing bootstrapp - both correct
bag = bag.fit(X_train, y_train)

yPred = bag.predict(X_test)
print(bag.estimators_samples_)
```

```
[array([219,  41, 122, 202,  76, 130, 137, 231,  92, 199, 216, 147, 114,
       120, 189,  28, 233, 220,  84,   6, 141,  68, 145,  82,  51, 105,
        74,  34,  93,   4,  34,   8, 142, 205,  32,  59,  95,   3, 150,
       107,  46, 116, 199, 194, 105, 181, 177,  77, 184,  80, 213,  64,
         0, 122,  59, 103, 235,  81,  41, 181,  91,  70, 132,   0,  60,
       184,  23, 156,  14, 234, 183, 203, 123,  45, 209,  84, 165, 100,
       173, 213, 111,  23,  70,   1,  81, 172,  38,  97, 111, 213, 115,
        52, 182,   9,  82,  71, 109,  73, 191, 116, 217, 196, 200,  14,
        32, 176, 104, 174,  43, 104,  45, 196, 203, 236, 177, 101,  53,
         5, 114, 142, 146,  36, 152,  47,  23,  84,  96,  39, 120, 143,
        36,   0, 214, 150, 135, 122, 181,  13, 222,   9,  15, 125, 191,
       156, 169,   6,  77,  48,  27,  70, 220, 191, 128,  60, 220, 200,
       148, 173,  49, 131,  12, 225,  85, 184,  47,  99,  62, 169, 119,
       145,   1,  90, 227, 164, 166,  36,  86, 236, 150,  64,  66, 211,
        68,  76,  57,  87, 221,  93, 110, 158,  79, 208,  65, 208,  57,
       199,  75,   5, 198, 200, 222, 102,  14,   1, 157,  26,  97,  16,
       129, 176, 227, 224, 216, 127,  66, 175, 194, 138,  62, 121, 171,
        11, 117, 216,  24,  74,  36,  23, 103,  30,  36,  64, 140, 154,
       119, 169, 204]), array([174, 151,  23,  31, 101, 234,   3, 169,  47, 2
02, 208,  83, 200,
       163, 109,  99, 201, 222, 104,  72,  18,  71, 139,  90, 117, 219,
        51,  58,  75,  87, 184, 107,  16, 227, 201, 177,  85,  84, 227,
        91,   7, 193,  72, 161,   3, 131, 216, 177,  61,  45,  31, 182,
       226, 189, 229, 174,  84, 171,   2,   9, 178, 144,  57, 189,  59,
       168, 138, 139, 119,  39, 184, 134, 222,  78,  53, 189, 217, 137,
       139,  87, 193, 113, 161, 130,  17, 207,  41, 152, 119,  12, 143,
       121, 134,  50,  83, 103, 124, 225, 152,  59, 192,  55, 179,  94,
        58,   0, 175, 176, 115, 191,  18, 133,  67,  84, 137, 166,  27,
        86, 142, 229, 191,  73, 178, 132,  60,  71,  99, 166,  57,  98,
       194, 228,  72, 183,  34, 165,  29,  87, 171,  34,  73, 188, 137,
       222,  54,  86, 136, 227,  60, 162,  99, 227, 129, 135,  83,  83,
        12, 150, 148,  20, 104,  90,  27,   9, 121, 178,  74,  25,   1,
       175, 192,  56, 211, 193,  75, 131, 127,  82, 233, 125,  94,  44,
        66, 114,  68,  14, 106, 110, 122,  78, 187,  80,   1, 195, 131,
        21,  84, 176, 191,   7, 222,  27,  85, 186, 159, 103, 209, 108,
       204,  46, 120, 168, 221, 188,  13, 102, 111,  91, 214, 192, 163,
       213, 219,  14, 137,  36,  41, 126, 168,  60, 107, 110, 187,  65,
        18, 226,  82]), array([ 86, 174,  32,  71, 231, 217, 168, 188, 117, 1
84,  18,  53, 142,
        52, 131, 120, 166, 126, 123,  83, 165, 216,  40,  45,  29,  87,
       214,  85, 196,  79,  64, 183,   2,  33,  52, 144, 169,  18,  51,
        80, 138, 164,  70,  11, 201, 223,  52,  70,  19, 207,  27,  87,
        96, 204,  38,  52, 174, 112,  25, 118, 146,  93, 119,  58,  56,
       194, 172,  45,  15,  40,  86, 197, 190, 226, 177, 104, 176, 108,
        11,  28, 208,  51, 172,  18,  18, 182,  71,  82, 208,  40, 114,
        29,   8, 209,  13, 101,  49, 105, 187, 232, 108, 122, 204,  20,
        22,  90,  69, 202,  52, 141, 105, 204,  74,  86,  22, 196, 191,
       218, 130,  74,  87,  90, 173, 232, 145, 118,  29,   2, 174, 163,
       232, 202,  87, 111,  38, 194, 107,  29,  19, 236,  57,  13,  13,
       208,  45, 236,  80, 197, 198, 207,  77, 209, 227,  73, 163, 235,
        17, 192, 102, 211,  99, 133, 226,   5, 185, 156,  24,  61,  27,
        17,  78, 119, 198, 155,  16, 214, 196,  22,  32,  31, 124,  35,
       149, 140,  93, 119, 224,  41, 169, 120,  50, 113,  83, 171,  80,
        99,  54,  77,  16,  30,  29, 218,  79, 162, 195, 132, 235,  36,
        80,  77, 167, 115, 157,  35, 112, 111,  85,  34,  21, 167,  20,
       136, 168, 143, 173,  86, 213, 143,  43, 109, 187, 137, 224,  21,
       204, 125,  73]), array([ 32,  69,  55,  63, 202, 204,   4, 113, 211, 1
```

```
91, 118, 223, 146,
       213,  43,   3, 191,  49, 162, 234, 190, 219, 216, 120,  31,  17,
       211, 100, 206,  51,  87, 214,  95, 127, 212,  10, 182, 188,  41,
        78, 102, 211,  26, 107, 107, 186, 193,  70, 199, 148,  89, 160,
        23,  61,  99, 210,   2,  57,  12,  31,  33, 221,  62,   9,  66,
       129, 178, 101, 218,  61,   5,  48, 190,  28, 137, 212,  99, 137,
       204, 234, 140,   3,   0, 164, 175, 223,   7, 177, 172, 212,  65,
       113, 176,  38,  89,  82,  72, 155, 167, 216, 190, 176, 150, 198,
        74,  22,  41, 234, 131, 111, 107, 143, 195,  34, 175,   0, 148,
       174, 144, 159, 164,  58,  65, 136, 185,  62,  53, 159, 151, 188,
       156,   1,   9,  32,  24,  28,   4, 220, 194, 190, 178,  59, 139,
       165,  42, 186,   8,  27,  20,  61, 145, 176,  28, 105,  36, 204,
        88,  79,  40, 215,  59, 107, 223, 152, 152, 147,  79, 161,  82,
       141, 203,  91,  37,  36,  14,  49,  12, 192,  67,  86,  24, 105,
       142,  68, 181,  55, 185,  55, 224, 105,  24,  99, 133,  64,  96,
        23,  19, 122,  90, 154, 160, 231,  40, 153, 101, 125,  25,  90,
        83, 142,  99,  32, 184,  52, 129,   1,  17, 103,  28, 177,  23,
       224, 198,  32, 106, 173, 119, 113,  90, 199, 122, 188,  55,  34,
       189, 104,  37]), array([ 94,  70,  25, 230, 235,  35, 143, 227,   8, 2
22,  99, 215, 232,
        90, 112, 188, 167,  74, 157, 236, 203, 143, 174,  51,  48,  60,
       146,  69, 184, 199,   1,   2, 175,  62,  39,  62, 195, 131, 105,
       195, 204, 190, 133,   0, 215, 103,  58, 177, 153, 105, 218,  69,
       124,  61,  58, 185, 114, 128,   2,  62, 125,  96, 222, 119, 128,
        37,  78,  57, 230, 119, 207,  83,  55,  67, 114,  65, 152, 167,
         9,  68, 202,  55, 185, 162,  74,  24,  64, 119, 222, 143,  50,
       180, 168, 168,  39, 211,  61, 103,   9, 179, 221, 102, 171,  83,
       235, 177, 215,  65, 204,  68, 236,  91,  44, 180, 203,  87,  97,
       173,   8,  47,   1, 150, 186, 150, 143,  79,  19,  97, 172,  75,
       162,  22,  60, 113,  19, 205, 161,  76, 165, 200, 107,  21,  64,
       105,  99,  73, 155,   3, 172, 215, 203,  95,  69,  32, 218, 126,
        26, 181, 155, 223, 154,  91, 101,  19, 198, 166,  38, 140,  50,
        90,   8,  61, 114,  40, 135,  85, 194, 129,  17,  81, 228,  59,
       220, 124,  32,  40, 136, 142, 122, 189,  28, 161, 234, 104,  24,
         4, 119, 140, 190,  78, 201,  93,  98,  29, 132, 189, 124,  17,
        61, 172, 169, 235,  57,  28, 219, 202,  93, 123, 171,  44,   1,
       227,   6,  43,  91, 104, 196, 160,  53, 205, 157,  68,  23, 146,
        99, 196,  45]), array([213,  65, 210, 103, 217, 205, 109, 145, 173,
35,  79, 223, 166,
       219,  87, 139, 161,  89, 206,  65,  53,   9, 171,  36, 200, 139,
       178, 228, 179,   1, 127,  99, 110,  99,  79,  70, 230,  97,  21,
       118, 152, 201,  26,   5, 117, 226,   4,  95, 204, 142,  32, 125,
       156, 113, 178,  89,   7, 189, 234, 144, 176,  67, 230,  35,  58,
       235, 130, 106, 212, 107,  14,  96,   6,  99, 117,  32, 128, 166,
       136, 102, 189,  79,  85, 210,  26,  92, 116,  86,  44, 178,  59,
       100, 102,  48, 189, 218,  55,  16,  99, 182, 197, 191,  27,  42,
       195, 175,  41,  89, 230,  50,  13,  35, 186, 197, 233, 147,  51,
         7, 160, 176, 100, 182,   5,  26,  62,  60,  54,   3,  18,  27,
        32,  80, 133,  87,  42, 179,  19, 106, 215, 225,  80,  64,  30,
       181, 138,  48,  20,  11, 151, 115, 103, 149, 173, 195, 103, 230,
       101,  19,  34,  70, 236,  85,   7,  50, 153,  16,  87, 112,  23,
       108,  23,  79,  72,  96, 146, 204, 128,  16,  81, 199, 209, 175,
        74,  98,  47,  67, 124,  13, 105, 192, 202,  14, 209, 209,  33,
         7, 103, 139,  95,  43, 128, 113, 210, 175, 191, 153,  84,  86,
        24, 125, 115,  87,  89,  44,  21,  22, 193, 116, 188, 153,  84,
        25,  46,  39, 217, 142,  11, 127,  94, 194, 119, 108, 113,  18,
        30,  37, 145]), array([140, 140, 125, 226, 170, 209,  21, 130, 227, 1
```

```
27,  49, 101, 133,
       154, 207,  43, 188, 141, 129,  62,  22,  97, 124,  57, 170, 195,
        59, 110, 125, 214, 139,  94, 160, 115,  41, 206,  28, 103, 101,
       226, 129, 104,  87,   5,  59, 128, 176, 138, 190, 185, 225,  96,
       146, 234,  94, 124, 117, 200, 134,  23, 121, 193, 150, 139, 120,
        14,  53,  89,  55,  77, 128,  88, 108,  84,  87, 223, 186, 189,
       166,  78,  20, 130,  90,   2,  39,  92,  80, 120,  78, 115, 208,
       235, 154,  62, 133, 228,  71, 228, 194, 145, 223,  40, 143, 209,
       199,   1, 121,   7, 103,  53, 120,  73,  11,  66, 218,  72,  25,
       189,  15,  33, 195, 228, 122, 233, 160,  90, 182,  83, 126,  57,
       144, 217, 114, 174, 192, 235, 144, 201,  39, 217, 229,  22,  44,
       181,  66, 150, 199, 115,   5,  89, 133, 194,   9, 227, 100,  50,
       188,  78, 199, 153, 132,  62,  85, 218, 193, 213, 154, 129, 147,
       146, 201,  76,  35, 153, 236,  85, 123, 167,  50,  42, 131, 191,
       164, 164,   6, 118,  92,  18, 203, 110, 193, 168, 116, 114,  67,
         4, 128, 141, 116, 108, 155,  59, 141,   5,  69,   2,  82, 207,
        32, 128,  32,  64,  82, 160, 193, 115, 234, 167, 234, 155, 129,
       132, 211, 208,  47, 133, 146, 129,  40,  82, 216,  17,  46,   7,
       179, 173,  20]), array([174,  11, 166, 178,  88,  66,   5,  89, 176, 2
12,  95, 111, 218,
       191, 227, 165, 158,  38, 176, 201, 192,  23,  68,  16, 226, 220,
        84,  40,  61,  73, 212,  32, 148, 110,  31, 156, 107, 192,  41,
        98,  12,   6, 139, 142, 180,  93, 184, 189,  25,  11,  80,  45,
       148, 184, 185,  94, 203,  49,  81, 190, 164,  64, 169,   0, 206,
        57,  26, 195,  42,  44, 148, 183,  16, 151, 155,  27, 130, 115,
        36, 111, 137,  23, 209,  37,  37, 117, 152, 125,   4,  13, 143,
       115,   1,  41, 197,  71, 193,  98, 220,  89,  64,  67, 209, 136,
       207, 141,   3, 132, 160,  52, 223, 215, 107,  68,  16,  78, 152,
       215, 163,  97, 119,  90,  92, 170,  42, 195, 218,  97, 168,   6,
       140, 209, 216, 105, 132,  54, 173,  52, 226,  52, 207, 177,  29,
        48,  98,  18, 111,  95,  61, 116, 182, 165, 204, 175,  97,  27,
       233, 102, 211, 221, 165, 155,  55, 132,  49,  27,  58, 208,  38,
        97,  11, 108, 136,  55,  76, 143, 225,  59,  87,  47,  68,  62,
       125, 133, 106, 230, 149, 178,  94, 100, 196,  80,  78,  60, 206,
       206,  56, 117,  21, 171, 125, 213, 233, 207,  51, 162, 134, 119,
       131,  40, 160,   7,  34,  34, 229, 125,   2,   4,  11,  70, 133,
        73, 191, 204,  65, 181, 109,  36, 133, 220,  36, 106, 200, 209,
        84, 185, 148]), array([168, 220, 140, 234,  29, 228,  84,  67,  77, 1
14, 140, 115,  54,
       108,  60, 170, 235, 184,   3,  77, 151,  88, 156, 215,  75,  80,
         3, 167, 150,  30,  26,  16, 224,  52, 202, 153,  82,  25,  81,
       144, 131, 132, 236,  81, 139, 191,  83, 159,  39, 207,  32,  75,
       129,  22,   1, 233, 210, 167, 152, 228,  70,  56, 118, 166,  95,
        35, 230,  55, 181, 160,  66,  45, 155, 186, 105, 230, 175, 179,
       100, 226, 216, 137, 112,  41,  29,  71,  74,  28,  93, 203,  80,
         6, 122, 191,  80, 114, 215, 189, 227, 146, 226, 105,  43, 208,
       122, 179,  75, 191,  53,  70,  77,  57, 229, 194,  59,  56, 146,
       192, 235, 221, 159, 195, 232,  34,  35, 178, 159,  55, 208,  14,
       212,  38, 134,  11, 110, 212,   9, 169,  60, 100, 233, 228, 151,
        39, 200,  16, 135, 200, 223, 185,   3,  11, 113, 124, 198,  98,
        68,  42, 191, 187, 222, 204, 167, 178, 174, 198, 202, 133, 217,
        76,  72, 125, 152, 229, 145,   7, 195, 161,  45,  51, 174, 179,
         3,  33, 157, 210,  78,  84, 106, 234, 109, 232, 109, 179, 167,
        42,  20,  29, 197,  12, 217, 142, 197, 223,  64, 122, 129,  80,
        71,  59,  64, 148, 190, 129, 227, 176, 136, 109,  48, 228,  22,
        93, 148, 233, 211,  74, 187, 163, 217, 143,  38, 166,  16, 188,
       169, 183, 177]), array([136,  13,  81,  56, 174, 105,  60, 144, 199,
```

```
54,  58, 174,  90,
       191, 108,  40,  97,  17, 205, 154,  27, 119,  95, 170, 101, 201,
       182, 124, 197, 145,  56,   8,  32, 182,  47,   7, 236,  60, 234,
       189, 221, 176, 121, 169, 211,   3, 172, 166, 105, 224, 154, 140,
       106,  62, 121,  97, 165,  76,  10,  99,  51, 176,  99,  93, 164,
       228,  37, 208, 216, 158, 154,  77, 224, 139, 227, 138, 164, 101,
        23, 130, 116,  81, 190, 181, 215,  48, 214, 180,  65,  15,  95,
       186, 211,  65, 198, 178,  31,  53,  84,  76, 177, 152, 192,  31,
        12, 197,  60,  88, 233, 186, 182, 100, 147, 214, 121, 228,  86,
       166, 176, 178,  20,   2, 111,  74,  11,  62, 228, 154, 208, 217,
       184,  12,  82, 172,  81,  80, 230, 176, 126, 213, 185,  37, 169,
       194, 127,  12, 201, 195,  24,  57,  56, 157, 196,  69, 122,  39,
        23,   5,  16,  55,   6,   3,  24,  49, 195, 183, 133, 141, 170,
       119,  64,  90, 105, 175, 104, 137,  10,   8, 226, 136,  22, 170,
       127, 170, 201,  70,  56, 231, 117,  40, 101, 226, 214, 195,  68,
       160, 227, 180, 137,  76,  80, 203, 202,  65, 180,  67, 207,  25,
        73, 167, 234, 159, 112, 159, 220,  80, 116, 122,  92, 232, 108,
       182, 233, 205, 217,   5,  93, 195, 221, 150, 223, 144, 208, 184,
       119,  72, 183])]
```

In [58]:

```
grader.grade(test_case_id = 'test_bag', answer = bag.estimators_samples_)
```

```
Correct! You earned 0.5/0.5 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Evaluate your base model on the test set by calculating the precision, recall and accuracy. Assign these values to `bag_precision` , `bag_recall` and `bag_accuracy` respectively.

In [59]:
```
#Trees
from sklearn import tree
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor

#Preprocessing Packages
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer #One hot encoding
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.base import BaseEstimator, TransformerMixin

#Cross Validation
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold

#Metrics
from sklearn.metrics import mean_squared_error
from sklearn.metrics import confusion_matrix

#Plotting
import seaborn as sns
import matplotlib.pyplot as plt
from IPython.display import Image
#import pydotplus

#Bootstrap
from sklearn.utils import resample

ac = bag.score(X_test, y_test)
print('bag accuracy: ', ac, '\n')

# using sklearn:
print()
ac = accuracy_score(y_test, yPred)
print('accuracy score using sklearn: ', ac, '\n')

cm = confusion_matrix(y_test, yPred)
print('confusion_matrix: ')
print(cm, '\n')


# using sklearn:
pre = precision_score(y_test, yPred)
print('precision score using sklearn: ', pre, '\n')

# using sklearn:
recall = recall_score(y_test, yPred)
print('recall score using sklearn: ', recall, '\n')
print('\n')
#


bag_precision = 0.8636363636363636
```

```
bag_recall = 0.7916
bag_accuracy = 0.8333333333333334
```

```
bag accuracy:  0.85


accuracy score using sklearn:  0.85

confusion_matrix:
[[32  4]
 [ 5 19]]

precision score using sklearn:  0.8260869565217391

recall score using sklearn:  0.7916666666666666
```

In [60]:
```
grader.grade(test_case_id = 'test_bag_score', answer = (bag_precision, bag_recall, bag_accuracy))
```

```
Correct! You earned 1.0/1 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Do **not** modify `base_classifier`. Tune the hyperparameters of your model and evaluate this tuned model by calculating the precision, recall and accuracy on the test set. Assign these calculated values to `bag_tuned_precision`, `bag_tuned_recall` and `bag_tuned_accuracy` respectively. Store your best performing estimators to `bag_best_param` as a dictionary ('tuned estimator' : number). *Hint*: For each tuned parameters, set searching space to all integers between range 10 and 50.

In [61]:
```
from sklearn.ensemble import BaggingClassifier
import sklearn.metrics
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
# Enter your code here



nEst = range(10, 50)
boot = range(10, 50)
bootFeat = range(10, 50)
maxFeat = range(10, 50)
maxSamp = range(10, 50)


#tuned_parameters = [{'n_estimators': [100, 200, 300, 400, 500, 600]}] #
tuned_parameters = [{'n_estimators': nEst}]
# = [{'n_estimators': nEst, 'bootstrap': boot, 'bootstrap_features': bootFeat,
#                      'max_features': maxFeat, 'max_samples': maxSamp}]
#scores = ['precision', 'recall'] # sklearn exmple
scores = ['accuracy', 'precision', 'recall']


for score in scores:
    print('score: ', score)
    #go = int(input('continue: '))
    print("# Tuning hyper-parameters for %s" % score)
    print()

    #clf = GridSearchCV(dt, tuned_parameters, cv=10, scoring='%s' % score)
    clf = GridSearchCV(bag, tuned_parameters, cv=10, scoring='%s' % score)
    clf.fit(X_train, y_train)

    print("\nBest parameters set found on development set:", clf.best_params_,
'\n')
    print('\nclf.best_estimator_: ', clf.best_estimator_, '\n')
    print('\nclf.best_score_: ', clf.best_score_, '\n')

    print("Grid scores on development set:")
    print()
    means = clf.cv_results_['mean_test_score']
    stds = clf.cv_results_['std_test_score']
    for mean, std, params in zip(means, stds, clf.cv_results_['params']):
        print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
    print()

    print("Detailed classification report:")
    print()
    print("The model is trained on the full development set.")
    print("The scores are computed on the full evaluation set.")
    print()
    y_true, y_pred = y_test, clf.predict(X_test)
    print(classification_report(y_true, y_pred))
    print()

print('\nCell process concluded\n')
```

```
bag_best_param = {'n_estimators': 19} # for accuracy


bag_tuned_precision = 0.8076923076923077
bag_tuned_recall = 0.875
bag_tuned_accuracy = 0.8666666666666667
```

```
score:  accuracy
# Tuning hyper-parameters for accuracy


Best parameters set found on development set: {'n_estimators': 10}


clf.best_estimator_:  BaggingClassifier(random_state=42)


clf.best_score_:  0.7842391304347827

Grid scores on development set:

0.784 (+/-0.173) for {'n_estimators': 10}
0.764 (+/-0.169) for {'n_estimators': 11}
0.772 (+/-0.179) for {'n_estimators': 12}
0.755 (+/-0.150) for {'n_estimators': 13}
0.772 (+/-0.193) for {'n_estimators': 14}
0.763 (+/-0.172) for {'n_estimators': 15}
0.767 (+/-0.212) for {'n_estimators': 16}
0.767 (+/-0.184) for {'n_estimators': 17}
0.776 (+/-0.186) for {'n_estimators': 18}
0.772 (+/-0.175) for {'n_estimators': 19}
0.776 (+/-0.152) for {'n_estimators': 20}
0.759 (+/-0.169) for {'n_estimators': 21}
0.763 (+/-0.171) for {'n_estimators': 22}
0.759 (+/-0.170) for {'n_estimators': 23}
0.768 (+/-0.162) for {'n_estimators': 24}
0.763 (+/-0.138) for {'n_estimators': 25}
0.772 (+/-0.170) for {'n_estimators': 26}
0.759 (+/-0.173) for {'n_estimators': 27}
0.772 (+/-0.178) for {'n_estimators': 28}
0.776 (+/-0.166) for {'n_estimators': 29}
0.776 (+/-0.185) for {'n_estimators': 30}
0.776 (+/-0.166) for {'n_estimators': 31}
0.772 (+/-0.166) for {'n_estimators': 32}
0.768 (+/-0.170) for {'n_estimators': 33}
0.763 (+/-0.181) for {'n_estimators': 34}
0.768 (+/-0.148) for {'n_estimators': 35}
0.768 (+/-0.189) for {'n_estimators': 36}
0.751 (+/-0.188) for {'n_estimators': 37}
0.763 (+/-0.195) for {'n_estimators': 38}
0.763 (+/-0.176) for {'n_estimators': 39}
0.772 (+/-0.179) for {'n_estimators': 40}
0.780 (+/-0.170) for {'n_estimators': 41}
0.772 (+/-0.179) for {'n_estimators': 42}
0.776 (+/-0.171) for {'n_estimators': 43}
0.772 (+/-0.167) for {'n_estimators': 44}
0.776 (+/-0.171) for {'n_estimators': 45}
0.772 (+/-0.183) for {'n_estimators': 46}
0.772 (+/-0.183) for {'n_estimators': 47}
0.776 (+/-0.182) for {'n_estimators': 48}
0.772 (+/-0.183) for {'n_estimators': 49}

Detailed classification report:
```

```
The model is trained on the full development set.
The scores are computed on the full evaluation set.

              precision    recall  f1-score   support

           0       0.86      0.89      0.88        36
           1       0.83      0.79      0.81        24

    accuracy                           0.85        60
   macro avg       0.85      0.84      0.84        60
weighted avg       0.85      0.85      0.85        60


score:  precision
# Tuning hyper-parameters for precision


Best parameters set found on development set: {'n_estimators': 10}


clf.best_estimator_:  BaggingClassifier(random_state=42)


clf.best_score_:  0.816836496836497

Grid scores on development set:

0.817 (+/-0.256) for {'n_estimators': 10}
0.771 (+/-0.249) for {'n_estimators': 11}
0.786 (+/-0.246) for {'n_estimators': 12}
0.754 (+/-0.218) for {'n_estimators': 13}
0.792 (+/-0.263) for {'n_estimators': 14}
0.774 (+/-0.252) for {'n_estimators': 15}
0.792 (+/-0.299) for {'n_estimators': 16}
0.777 (+/-0.262) for {'n_estimators': 17}
0.800 (+/-0.251) for {'n_estimators': 18}
0.782 (+/-0.230) for {'n_estimators': 19}
0.798 (+/-0.221) for {'n_estimators': 20}
0.760 (+/-0.226) for {'n_estimators': 21}
0.782 (+/-0.231) for {'n_estimators': 22}
0.774 (+/-0.233) for {'n_estimators': 23}
0.800 (+/-0.240) for {'n_estimators': 24}
0.788 (+/-0.225) for {'n_estimators': 25}
0.809 (+/-0.253) for {'n_estimators': 26}
0.788 (+/-0.268) for {'n_estimators': 27}
0.812 (+/-0.266) for {'n_estimators': 28}
0.808 (+/-0.251) for {'n_estimators': 29}
0.813 (+/-0.267) for {'n_estimators': 30}
0.797 (+/-0.254) for {'n_estimators': 31}
0.800 (+/-0.256) for {'n_estimators': 32}
0.789 (+/-0.267) for {'n_estimators': 33}
0.789 (+/-0.269) for {'n_estimators': 34}
0.784 (+/-0.222) for {'n_estimators': 35}
0.793 (+/-0.261) for {'n_estimators': 36}
0.758 (+/-0.241) for {'n_estimators': 37}
0.785 (+/-0.271) for {'n_estimators': 38}
0.776 (+/-0.231) for {'n_estimators': 39}
```

```
0.793 (+/-0.246) for {'n_estimators': 40}
0.798 (+/-0.236) for {'n_estimators': 41}
0.793 (+/-0.246) for {'n_estimators': 42}
0.791 (+/-0.240) for {'n_estimators': 43}
0.789 (+/-0.242) for {'n_estimators': 44}
0.791 (+/-0.240) for {'n_estimators': 45}
0.788 (+/-0.247) for {'n_estimators': 46}
0.788 (+/-0.247) for {'n_estimators': 47}
0.795 (+/-0.244) for {'n_estimators': 48}
0.788 (+/-0.247) for {'n_estimators': 49}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.

              precision    recall  f1-score   support

           0       0.86      0.89      0.88        36
           1       0.83      0.79      0.81        24

    accuracy                           0.85        60
   macro avg       0.85      0.84      0.84        60
weighted avg       0.85      0.85      0.85        60


score:  recall
# Tuning hyper-parameters for recall


Best parameters set found on development set: {'n_estimators': 11}


clf.best_estimator_:  BaggingClassifier(n_estimators=11, random_state=42)


clf.best_score_:  0.7446969696969697

Grid scores on development set:

0.727 (+/-0.235) for {'n_estimators': 10}
0.745 (+/-0.259) for {'n_estimators': 11}
0.736 (+/-0.279) for {'n_estimators': 12}
0.745 (+/-0.259) for {'n_estimators': 13}
0.727 (+/-0.310) for {'n_estimators': 14}
0.736 (+/-0.279) for {'n_estimators': 15}
0.719 (+/-0.303) for {'n_estimators': 16}
0.744 (+/-0.255) for {'n_estimators': 17}
0.727 (+/-0.274) for {'n_estimators': 18}
0.743 (+/-0.267) for {'n_estimators': 19}
0.735 (+/-0.250) for {'n_estimators': 20}
0.743 (+/-0.267) for {'n_estimators': 21}
0.717 (+/-0.241) for {'n_estimators': 22}
0.717 (+/-0.241) for {'n_estimators': 23}
0.709 (+/-0.230) for {'n_estimators': 24}
0.717 (+/-0.203) for {'n_estimators': 25}
0.709 (+/-0.230) for {'n_estimators': 26}
```

```
0.709 (+/-0.230) for {'n_estimators': 27}
0.709 (+/-0.230) for {'n_estimators': 28}
0.726 (+/-0.215) for {'n_estimators': 29}
0.717 (+/-0.241) for {'n_estimators': 30}
0.742 (+/-0.219) for {'n_estimators': 31}
0.726 (+/-0.215) for {'n_estimators': 32}
0.735 (+/-0.222) for {'n_estimators': 33}
0.717 (+/-0.241) for {'n_estimators': 34}
0.734 (+/-0.198) for {'n_estimators': 35}
0.717 (+/-0.241) for {'n_estimators': 36}
0.717 (+/-0.241) for {'n_estimators': 37}
0.717 (+/-0.241) for {'n_estimators': 38}
0.726 (+/-0.215) for {'n_estimators': 39}
0.726 (+/-0.215) for {'n_estimators': 40}
0.743 (+/-0.234) for {'n_estimators': 41}
0.726 (+/-0.215) for {'n_estimators': 42}
0.743 (+/-0.234) for {'n_estimators': 43}
0.734 (+/-0.198) for {'n_estimators': 44}
0.743 (+/-0.234) for {'n_estimators': 45}
0.735 (+/-0.250) for {'n_estimators': 46}
0.735 (+/-0.250) for {'n_estimators': 47}
0.735 (+/-0.250) for {'n_estimators': 48}
0.735 (+/-0.250) for {'n_estimators': 49}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.

              precision    recall  f1-score   support

           0       0.89      0.86      0.87        36
           1       0.80      0.83      0.82        24

    accuracy                           0.85        60
   macro avg       0.84      0.85      0.84        60
weighted avg       0.85      0.85      0.85        60



Cell process concluded
```

In [62]:
```
grader.grade(test_case_id = 'test_bag_tuned', answer = (bag_tuned_precision, bag_tuned_recall, bag_tuned_accuracy))
```

```
Correct! You earned 1/1 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

In [63]:
```
grader.grade(test_case_id = 'test_bag_best', answer = bag_best_param)
```

```
Correct! You earned 1.0/1 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

# 9. Part C: Random Forest

1. Using all predictor variables, train a base random forest to predict the response variable. Use `sklearn.ensemble.RandomForestClassifier` and set `random_state=42` . Name your model `rf` .

In [64]:
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(random_state=42)
rf = rf.fit(X_train, y_train)
yPred = rf.predict(X_test)
```

In [65]:
```
grader.grade(test_case_id = 'test_rf', answer = rf.n_estimators)
```

```
Correct! You earned 0.5/0.5 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Evaluate your base model on the test set by calculating the precision, recall and accuracy. Assign these values to `rf_precision` , `rf_recall` and `rf_accuracy` respectively.

In [66]:
```
rf_precision = precision_score(y_test, yPred)
rf_recall = recall_score(y_test, yPred)
rf_accuracy = accuracy_score(y_test, yPred)

print(rf_precision)
print(rf_recall)
print(rf_accuracy )
```

```
0.8076923076923077
0.875
0.8666666666666667
```

In [67]:
```
grader.grade(test_case_id = 'test_rf_score', answer = (rf_precision, rf_recall
, rf_accuracy))
```

```
Correct! You earned 0.5/0.5 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Tune the hyperparameters of your model and evaluate this tuned model by calculating the precision, recall and accuracy on the test set. Assign these calculated values to `rf_tuned_precision` , `rf_tuned_recall` and `rf_tuned_accuracy` respectively. Store your best-performing parameters to `rf_tuned` . Also don't forget to set `random_state=42` .

*Hint*:

1. In this question, we would like to tune only the parameters determining the depth of a tree and the number of trees. But beware that you can also adjust the maximum number of features considered for each tree(max_features), the minimum number of data points placed in a node before the node is split(min_samples_split), the minimum number of data points allowed in a leaf node(min_samples_leaf), and bootstrap option. Refer to [sklearn.RandomForestClassifier (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html) for a complete list.
2. In this question, your parameter search space is (3,21) for tree depth, (10,30) for number of trees.

In [ ]:
```
maxDepth = range(3, 21)
nEst = range(10, 30)

tuned_parameters = [{'max_depth': maxDepth, 'n_estimators': nEst}] #
#scores = ['precision', 'recall'] # sklearn exmple
scores = ['accuracy', 'precision', 'recall']


for score in scores:
    print('score: ', score)
    #go = int(input('continue: '))
    print("# Tuning hyper-parameters for %s" % score)
    print()

    #clf = GridSearchCV(dt, tuned_parameters, cv=10, scoring='%s' % score)
    clf = GridSearchCV(rf, tuned_parameters, cv=10, scoring='%s' % score)
    clf.fit(X_train, y_train)

    print("\nBest parameters set found on development set:", clf.best_params_,
'\n')
    print('\nclf.best_estimator_: ', clf.best_estimator_, '\n')
    print('\nclf.best_score_: ', clf.best_score_, '\n')

    print("Grid scores on development set:")
    print()
    means = clf.cv_results_['mean_test_score']
    stds = clf.cv_results_['std_test_score']
    for mean, std, params in zip(means, stds, clf.cv_results_['params']):
        print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
    print()

    print("Detailed classification report:")
    print()
    print("The model is trained on the full development set.")
    print("The scores are computed on the full evaluation set.")
    print()
    y_true, y_pred = y_test, clf.predict(X_test)
    print(classification_report(y_true, y_pred))
    print()


rf_tuned_accuracy = 0.8666666666666667  # 0.819 (+/-0.112) for {'max_depth':
 4, 'n_estimators': 20}
rf_tuned_recall = 0.8333333333333334    # 0.771 (+/-0.207) for {'max_depth': 1
1, 'n_estimators': 21}
rf_tuned_precision = 0.8333333333333334 # 0.874 (+/-0.182) for {'max_depth':
 4, 'n_estimators': 26}

rf_tuned =  {'max_depth': 4, 'n_estimators': 20}

print('\nCell process concluded\n')
```

```
score:  accuracy
# Tuning hyper-parameters for accuracy


Best parameters set found on development set: {'max_depth': 7, 'n_estimator
s': 23}


clf.best_estimator_:  RandomForestClassifier(max_depth=7, n_estimators=23, ra
ndom_state=42)


clf.best_score_:  0.8186594202898549

Grid scores on development set:

0.785 (+/-0.095) for {'max_depth': 3, 'n_estimators': 10}
0.781 (+/-0.099) for {'max_depth': 3, 'n_estimators': 11}
0.777 (+/-0.104) for {'max_depth': 3, 'n_estimators': 12}
0.793 (+/-0.080) for {'max_depth': 3, 'n_estimators': 13}
0.785 (+/-0.078) for {'max_depth': 3, 'n_estimators': 14}
0.797 (+/-0.083) for {'max_depth': 3, 'n_estimators': 15}
0.798 (+/-0.072) for {'max_depth': 3, 'n_estimators': 16}
0.802 (+/-0.065) for {'max_depth': 3, 'n_estimators': 17}
0.811 (+/-0.075) for {'max_depth': 3, 'n_estimators': 18}
0.811 (+/-0.098) for {'max_depth': 3, 'n_estimators': 19}
0.815 (+/-0.083) for {'max_depth': 3, 'n_estimators': 20}
0.802 (+/-0.073) for {'max_depth': 3, 'n_estimators': 21}
0.806 (+/-0.065) for {'max_depth': 3, 'n_estimators': 22}
0.794 (+/-0.092) for {'max_depth': 3, 'n_estimators': 23}
0.798 (+/-0.079) for {'max_depth': 3, 'n_estimators': 24}
0.802 (+/-0.082) for {'max_depth': 3, 'n_estimators': 25}
0.811 (+/-0.083) for {'max_depth': 3, 'n_estimators': 26}
0.806 (+/-0.075) for {'max_depth': 3, 'n_estimators': 27}
0.806 (+/-0.091) for {'max_depth': 3, 'n_estimators': 28}
0.806 (+/-0.091) for {'max_depth': 3, 'n_estimators': 29}
0.801 (+/-0.121) for {'max_depth': 4, 'n_estimators': 10}
0.793 (+/-0.137) for {'max_depth': 4, 'n_estimators': 11}
0.793 (+/-0.132) for {'max_depth': 4, 'n_estimators': 12}
0.797 (+/-0.144) for {'max_depth': 4, 'n_estimators': 13}
0.797 (+/-0.144) for {'max_depth': 4, 'n_estimators': 14}
0.788 (+/-0.140) for {'max_depth': 4, 'n_estimators': 15}
0.805 (+/-0.140) for {'max_depth': 4, 'n_estimators': 16}
0.793 (+/-0.120) for {'max_depth': 4, 'n_estimators': 17}
0.793 (+/-0.136) for {'max_depth': 4, 'n_estimators': 18}
0.806 (+/-0.109) for {'max_depth': 4, 'n_estimators': 19}
0.805 (+/-0.125) for {'max_depth': 4, 'n_estimators': 20}
0.805 (+/-0.111) for {'max_depth': 4, 'n_estimators': 21}
0.810 (+/-0.111) for {'max_depth': 4, 'n_estimators': 22}
0.797 (+/-0.100) for {'max_depth': 4, 'n_estimators': 23}
0.818 (+/-0.109) for {'max_depth': 4, 'n_estimators': 24}
0.806 (+/-0.110) for {'max_depth': 4, 'n_estimators': 25}
0.810 (+/-0.103) for {'max_depth': 4, 'n_estimators': 26}
0.810 (+/-0.103) for {'max_depth': 4, 'n_estimators': 27}
0.810 (+/-0.103) for {'max_depth': 4, 'n_estimators': 28}
0.806 (+/-0.109) for {'max_depth': 4, 'n_estimators': 29}
0.793 (+/-0.130) for {'max_depth': 5, 'n_estimators': 10}
```

```
0.801 (+/-0.117) for {'max_depth': 5, 'n_estimators': 11}
0.797 (+/-0.131) for {'max_depth': 5, 'n_estimators': 12}
0.801 (+/-0.117) for {'max_depth': 5, 'n_estimators': 13}
0.793 (+/-0.119) for {'max_depth': 5, 'n_estimators': 14}
0.789 (+/-0.116) for {'max_depth': 5, 'n_estimators': 15}
0.793 (+/-0.104) for {'max_depth': 5, 'n_estimators': 16}
0.789 (+/-0.133) for {'max_depth': 5, 'n_estimators': 17}
0.785 (+/-0.110) for {'max_depth': 5, 'n_estimators': 18}
0.793 (+/-0.125) for {'max_depth': 5, 'n_estimators': 19}
0.789 (+/-0.114) for {'max_depth': 5, 'n_estimators': 20}
0.785 (+/-0.112) for {'max_depth': 5, 'n_estimators': 21}
0.785 (+/-0.135) for {'max_depth': 5, 'n_estimators': 22}
0.793 (+/-0.129) for {'max_depth': 5, 'n_estimators': 23}
0.785 (+/-0.135) for {'max_depth': 5, 'n_estimators': 24}
0.780 (+/-0.156) for {'max_depth': 5, 'n_estimators': 25}
0.789 (+/-0.135) for {'max_depth': 5, 'n_estimators': 26}
0.797 (+/-0.139) for {'max_depth': 5, 'n_estimators': 27}
0.802 (+/-0.119) for {'max_depth': 5, 'n_estimators': 28}
0.802 (+/-0.114) for {'max_depth': 5, 'n_estimators': 29}
0.784 (+/-0.153) for {'max_depth': 6, 'n_estimators': 10}
0.767 (+/-0.153) for {'max_depth': 6, 'n_estimators': 11}
0.776 (+/-0.141) for {'max_depth': 6, 'n_estimators': 12}
0.780 (+/-0.159) for {'max_depth': 6, 'n_estimators': 13}
0.763 (+/-0.123) for {'max_depth': 6, 'n_estimators': 14}
0.776 (+/-0.155) for {'max_depth': 6, 'n_estimators': 15}
0.789 (+/-0.122) for {'max_depth': 6, 'n_estimators': 16}
0.776 (+/-0.109) for {'max_depth': 6, 'n_estimators': 17}
0.793 (+/-0.113) for {'max_depth': 6, 'n_estimators': 18}
0.802 (+/-0.121) for {'max_depth': 6, 'n_estimators': 19}
0.802 (+/-0.122) for {'max_depth': 6, 'n_estimators': 20}
0.797 (+/-0.125) for {'max_depth': 6, 'n_estimators': 21}
0.793 (+/-0.123) for {'max_depth': 6, 'n_estimators': 22}
0.801 (+/-0.127) for {'max_depth': 6, 'n_estimators': 23}
0.801 (+/-0.116) for {'max_depth': 6, 'n_estimators': 24}
0.801 (+/-0.116) for {'max_depth': 6, 'n_estimators': 25}
0.806 (+/-0.111) for {'max_depth': 6, 'n_estimators': 26}
0.806 (+/-0.111) for {'max_depth': 6, 'n_estimators': 27}
0.801 (+/-0.111) for {'max_depth': 6, 'n_estimators': 28}
0.801 (+/-0.110) for {'max_depth': 6, 'n_estimators': 29}
0.789 (+/-0.120) for {'max_depth': 7, 'n_estimators': 10}
0.797 (+/-0.114) for {'max_depth': 7, 'n_estimators': 11}
0.785 (+/-0.111) for {'max_depth': 7, 'n_estimators': 12}
0.802 (+/-0.112) for {'max_depth': 7, 'n_estimators': 13}
0.798 (+/-0.116) for {'max_depth': 7, 'n_estimators': 14}
0.781 (+/-0.129) for {'max_depth': 7, 'n_estimators': 15}
0.793 (+/-0.127) for {'max_depth': 7, 'n_estimators': 16}
0.777 (+/-0.134) for {'max_depth': 7, 'n_estimators': 17}
0.785 (+/-0.136) for {'max_depth': 7, 'n_estimators': 18}
0.789 (+/-0.139) for {'max_depth': 7, 'n_estimators': 19}
0.802 (+/-0.124) for {'max_depth': 7, 'n_estimators': 20}
0.798 (+/-0.104) for {'max_depth': 7, 'n_estimators': 21}
0.802 (+/-0.131) for {'max_depth': 7, 'n_estimators': 22}
0.819 (+/-0.125) for {'max_depth': 7, 'n_estimators': 23}
0.815 (+/-0.118) for {'max_depth': 7, 'n_estimators': 24}
0.806 (+/-0.108) for {'max_depth': 7, 'n_estimators': 25}
0.814 (+/-0.114) for {'max_depth': 7, 'n_estimators': 26}
0.802 (+/-0.131) for {'max_depth': 7, 'n_estimators': 27}
```

```
0.798 (+/-0.158) for {'max_depth': 7, 'n_estimators': 28}
0.802 (+/-0.141) for {'max_depth': 7, 'n_estimators': 29}
0.756 (+/-0.101) for {'max_depth': 8, 'n_estimators': 10}
0.764 (+/-0.115) for {'max_depth': 8, 'n_estimators': 11}
0.751 (+/-0.089) for {'max_depth': 8, 'n_estimators': 12}
0.772 (+/-0.113) for {'max_depth': 8, 'n_estimators': 13}
0.772 (+/-0.117) for {'max_depth': 8, 'n_estimators': 14}
0.760 (+/-0.102) for {'max_depth': 8, 'n_estimators': 15}
0.772 (+/-0.123) for {'max_depth': 8, 'n_estimators': 16}
0.777 (+/-0.081) for {'max_depth': 8, 'n_estimators': 17}
0.781 (+/-0.116) for {'max_depth': 8, 'n_estimators': 18}
0.785 (+/-0.108) for {'max_depth': 8, 'n_estimators': 19}
0.789 (+/-0.118) for {'max_depth': 8, 'n_estimators': 20}
0.789 (+/-0.110) for {'max_depth': 8, 'n_estimators': 21}
0.798 (+/-0.121) for {'max_depth': 8, 'n_estimators': 22}
0.793 (+/-0.113) for {'max_depth': 8, 'n_estimators': 23}
0.802 (+/-0.105) for {'max_depth': 8, 'n_estimators': 24}
0.789 (+/-0.116) for {'max_depth': 8, 'n_estimators': 25}
0.793 (+/-0.107) for {'max_depth': 8, 'n_estimators': 26}
0.785 (+/-0.113) for {'max_depth': 8, 'n_estimators': 27}
0.781 (+/-0.123) for {'max_depth': 8, 'n_estimators': 28}
0.781 (+/-0.102) for {'max_depth': 8, 'n_estimators': 29}
0.768 (+/-0.157) for {'max_depth': 9, 'n_estimators': 10}
0.772 (+/-0.128) for {'max_depth': 9, 'n_estimators': 11}
0.772 (+/-0.143) for {'max_depth': 9, 'n_estimators': 12}
0.776 (+/-0.153) for {'max_depth': 9, 'n_estimators': 13}
0.768 (+/-0.148) for {'max_depth': 9, 'n_estimators': 14}
0.768 (+/-0.161) for {'max_depth': 9, 'n_estimators': 15}
0.764 (+/-0.130) for {'max_depth': 9, 'n_estimators': 16}
0.751 (+/-0.148) for {'max_depth': 9, 'n_estimators': 17}
0.764 (+/-0.145) for {'max_depth': 9, 'n_estimators': 18}
0.768 (+/-0.129) for {'max_depth': 9, 'n_estimators': 19}
0.760 (+/-0.128) for {'max_depth': 9, 'n_estimators': 20}
0.772 (+/-0.120) for {'max_depth': 9, 'n_estimators': 21}
0.773 (+/-0.148) for {'max_depth': 9, 'n_estimators': 22}
0.785 (+/-0.101) for {'max_depth': 9, 'n_estimators': 23}
0.789 (+/-0.092) for {'max_depth': 9, 'n_estimators': 24}
0.793 (+/-0.095) for {'max_depth': 9, 'n_estimators': 25}
0.797 (+/-0.090) for {'max_depth': 9, 'n_estimators': 26}
0.798 (+/-0.110) for {'max_depth': 9, 'n_estimators': 27}
0.793 (+/-0.101) for {'max_depth': 9, 'n_estimators': 28}
0.789 (+/-0.099) for {'max_depth': 9, 'n_estimators': 29}
0.763 (+/-0.117) for {'max_depth': 10, 'n_estimators': 10}
0.772 (+/-0.112) for {'max_depth': 10, 'n_estimators': 11}
0.767 (+/-0.125) for {'max_depth': 10, 'n_estimators': 12}
0.759 (+/-0.130) for {'max_depth': 10, 'n_estimators': 13}
0.763 (+/-0.129) for {'max_depth': 10, 'n_estimators': 14}
0.755 (+/-0.099) for {'max_depth': 10, 'n_estimators': 15}
0.764 (+/-0.114) for {'max_depth': 10, 'n_estimators': 16}
0.768 (+/-0.125) for {'max_depth': 10, 'n_estimators': 17}
0.772 (+/-0.120) for {'max_depth': 10, 'n_estimators': 18}
0.764 (+/-0.118) for {'max_depth': 10, 'n_estimators': 19}
0.781 (+/-0.110) for {'max_depth': 10, 'n_estimators': 20}
0.781 (+/-0.118) for {'max_depth': 10, 'n_estimators': 21}
0.781 (+/-0.123) for {'max_depth': 10, 'n_estimators': 22}
0.793 (+/-0.096) for {'max_depth': 10, 'n_estimators': 23}
0.798 (+/-0.110) for {'max_depth': 10, 'n_estimators': 24}
```

```
0.793 (+/-0.093) for {'max_depth': 10, 'n_estimators': 25}
0.776 (+/-0.099) for {'max_depth': 10, 'n_estimators': 26}
0.776 (+/-0.113) for {'max_depth': 10, 'n_estimators': 27}
0.776 (+/-0.107) for {'max_depth': 10, 'n_estimators': 28}
0.780 (+/-0.106) for {'max_depth': 10, 'n_estimators': 29}
0.747 (+/-0.115) for {'max_depth': 11, 'n_estimators': 10}
0.755 (+/-0.109) for {'max_depth': 11, 'n_estimators': 11}
0.759 (+/-0.104) for {'max_depth': 11, 'n_estimators': 12}
0.755 (+/-0.116) for {'max_depth': 11, 'n_estimators': 13}
0.751 (+/-0.111) for {'max_depth': 11, 'n_estimators': 14}
0.742 (+/-0.097) for {'max_depth': 11, 'n_estimators': 15}
0.759 (+/-0.108) for {'max_depth': 11, 'n_estimators': 16}
0.751 (+/-0.103) for {'max_depth': 11, 'n_estimators': 17}
0.759 (+/-0.108) for {'max_depth': 11, 'n_estimators': 18}
0.747 (+/-0.092) for {'max_depth': 11, 'n_estimators': 19}
0.755 (+/-0.092) for {'max_depth': 11, 'n_estimators': 20}
0.755 (+/-0.094) for {'max_depth': 11, 'n_estimators': 21}
0.755 (+/-0.106) for {'max_depth': 11, 'n_estimators': 22}
0.763 (+/-0.089) for {'max_depth': 11, 'n_estimators': 23}
0.772 (+/-0.103) for {'max_depth': 11, 'n_estimators': 24}
0.781 (+/-0.105) for {'max_depth': 11, 'n_estimators': 25}
0.763 (+/-0.123) for {'max_depth': 11, 'n_estimators': 26}
0.768 (+/-0.116) for {'max_depth': 11, 'n_estimators': 27}
0.763 (+/-0.123) for {'max_depth': 11, 'n_estimators': 28}
0.763 (+/-0.104) for {'max_depth': 11, 'n_estimators': 29}
0.751 (+/-0.130) for {'max_depth': 12, 'n_estimators': 10}
0.755 (+/-0.109) for {'max_depth': 12, 'n_estimators': 11}
0.763 (+/-0.118) for {'max_depth': 12, 'n_estimators': 12}
0.755 (+/-0.116) for {'max_depth': 12, 'n_estimators': 13}
0.751 (+/-0.111) for {'max_depth': 12, 'n_estimators': 14}
0.742 (+/-0.097) for {'max_depth': 12, 'n_estimators': 15}
0.759 (+/-0.108) for {'max_depth': 12, 'n_estimators': 16}
0.751 (+/-0.103) for {'max_depth': 12, 'n_estimators': 17}
0.759 (+/-0.108) for {'max_depth': 12, 'n_estimators': 18}
0.747 (+/-0.092) for {'max_depth': 12, 'n_estimators': 19}
0.759 (+/-0.108) for {'max_depth': 12, 'n_estimators': 20}
0.755 (+/-0.094) for {'max_depth': 12, 'n_estimators': 21}
0.755 (+/-0.106) for {'max_depth': 12, 'n_estimators': 22}
0.763 (+/-0.089) for {'max_depth': 12, 'n_estimators': 23}
0.759 (+/-0.110) for {'max_depth': 12, 'n_estimators': 24}
0.772 (+/-0.108) for {'max_depth': 12, 'n_estimators': 25}
0.763 (+/-0.129) for {'max_depth': 12, 'n_estimators': 26}
0.755 (+/-0.101) for {'max_depth': 12, 'n_estimators': 27}
0.755 (+/-0.108) for {'max_depth': 12, 'n_estimators': 28}
0.755 (+/-0.108) for {'max_depth': 12, 'n_estimators': 29}
0.751 (+/-0.130) for {'max_depth': 13, 'n_estimators': 10}
0.755 (+/-0.109) for {'max_depth': 13, 'n_estimators': 11}
0.763 (+/-0.118) for {'max_depth': 13, 'n_estimators': 12}
0.755 (+/-0.116) for {'max_depth': 13, 'n_estimators': 13}
0.751 (+/-0.111) for {'max_depth': 13, 'n_estimators': 14}
0.742 (+/-0.097) for {'max_depth': 13, 'n_estimators': 15}
0.759 (+/-0.108) for {'max_depth': 13, 'n_estimators': 16}
0.751 (+/-0.103) for {'max_depth': 13, 'n_estimators': 17}
0.759 (+/-0.108) for {'max_depth': 13, 'n_estimators': 18}
0.747 (+/-0.092) for {'max_depth': 13, 'n_estimators': 19}
0.759 (+/-0.108) for {'max_depth': 13, 'n_estimators': 20}
0.759 (+/-0.095) for {'max_depth': 13, 'n_estimators': 21}
```

```
0.755 (+/-0.106) for {'max_depth': 13, 'n_estimators': 22}
0.768 (+/-0.088) for {'max_depth': 13, 'n_estimators': 23}
0.763 (+/-0.110) for {'max_depth': 13, 'n_estimators': 24}
0.772 (+/-0.108) for {'max_depth': 13, 'n_estimators': 25}
0.763 (+/-0.129) for {'max_depth': 13, 'n_estimators': 26}
0.759 (+/-0.102) for {'max_depth': 13, 'n_estimators': 27}
0.759 (+/-0.102) for {'max_depth': 13, 'n_estimators': 28}
0.759 (+/-0.116) for {'max_depth': 13, 'n_estimators': 29}
0.751 (+/-0.130) for {'max_depth': 14, 'n_estimators': 10}
0.755 (+/-0.109) for {'max_depth': 14, 'n_estimators': 11}
0.763 (+/-0.118) for {'max_depth': 14, 'n_estimators': 12}
0.755 (+/-0.116) for {'max_depth': 14, 'n_estimators': 13}
0.751 (+/-0.111) for {'max_depth': 14, 'n_estimators': 14}
0.742 (+/-0.097) for {'max_depth': 14, 'n_estimators': 15}
0.759 (+/-0.108) for {'max_depth': 14, 'n_estimators': 16}
0.751 (+/-0.103) for {'max_depth': 14, 'n_estimators': 17}
0.759 (+/-0.108) for {'max_depth': 14, 'n_estimators': 18}
0.747 (+/-0.092) for {'max_depth': 14, 'n_estimators': 19}
0.759 (+/-0.108) for {'max_depth': 14, 'n_estimators': 20}
0.759 (+/-0.095) for {'max_depth': 14, 'n_estimators': 21}
0.755 (+/-0.106) for {'max_depth': 14, 'n_estimators': 22}
0.768 (+/-0.088) for {'max_depth': 14, 'n_estimators': 23}
0.763 (+/-0.110) for {'max_depth': 14, 'n_estimators': 24}
0.772 (+/-0.108) for {'max_depth': 14, 'n_estimators': 25}
0.763 (+/-0.129) for {'max_depth': 14, 'n_estimators': 26}
0.759 (+/-0.102) for {'max_depth': 14, 'n_estimators': 27}
0.759 (+/-0.102) for {'max_depth': 14, 'n_estimators': 28}
0.759 (+/-0.116) for {'max_depth': 14, 'n_estimators': 29}
0.751 (+/-0.130) for {'max_depth': 15, 'n_estimators': 10}
0.755 (+/-0.109) for {'max_depth': 15, 'n_estimators': 11}
0.763 (+/-0.118) for {'max_depth': 15, 'n_estimators': 12}
0.755 (+/-0.116) for {'max_depth': 15, 'n_estimators': 13}
0.751 (+/-0.111) for {'max_depth': 15, 'n_estimators': 14}
0.742 (+/-0.097) for {'max_depth': 15, 'n_estimators': 15}
0.759 (+/-0.108) for {'max_depth': 15, 'n_estimators': 16}
0.751 (+/-0.103) for {'max_depth': 15, 'n_estimators': 17}
0.759 (+/-0.108) for {'max_depth': 15, 'n_estimators': 18}
0.747 (+/-0.092) for {'max_depth': 15, 'n_estimators': 19}
0.759 (+/-0.108) for {'max_depth': 15, 'n_estimators': 20}
0.759 (+/-0.095) for {'max_depth': 15, 'n_estimators': 21}
0.755 (+/-0.106) for {'max_depth': 15, 'n_estimators': 22}
0.768 (+/-0.088) for {'max_depth': 15, 'n_estimators': 23}
0.763 (+/-0.110) for {'max_depth': 15, 'n_estimators': 24}
0.772 (+/-0.108) for {'max_depth': 15, 'n_estimators': 25}
0.763 (+/-0.129) for {'max_depth': 15, 'n_estimators': 26}
0.759 (+/-0.102) for {'max_depth': 15, 'n_estimators': 27}
0.759 (+/-0.102) for {'max_depth': 15, 'n_estimators': 28}
0.759 (+/-0.116) for {'max_depth': 15, 'n_estimators': 29}
0.751 (+/-0.130) for {'max_depth': 16, 'n_estimators': 10}
0.755 (+/-0.109) for {'max_depth': 16, 'n_estimators': 11}
0.763 (+/-0.118) for {'max_depth': 16, 'n_estimators': 12}
0.755 (+/-0.116) for {'max_depth': 16, 'n_estimators': 13}
0.751 (+/-0.111) for {'max_depth': 16, 'n_estimators': 14}
0.742 (+/-0.097) for {'max_depth': 16, 'n_estimators': 15}
0.759 (+/-0.108) for {'max_depth': 16, 'n_estimators': 16}
0.751 (+/-0.103) for {'max_depth': 16, 'n_estimators': 17}
0.759 (+/-0.108) for {'max_depth': 16, 'n_estimators': 18}
```

```
0.747 (+/-0.092) for {'max_depth': 16, 'n_estimators': 19}
0.759 (+/-0.108) for {'max_depth': 16, 'n_estimators': 20}
0.759 (+/-0.095) for {'max_depth': 16, 'n_estimators': 21}
0.755 (+/-0.106) for {'max_depth': 16, 'n_estimators': 22}
0.768 (+/-0.088) for {'max_depth': 16, 'n_estimators': 23}
0.763 (+/-0.110) for {'max_depth': 16, 'n_estimators': 24}
0.772 (+/-0.108) for {'max_depth': 16, 'n_estimators': 25}
0.763 (+/-0.129) for {'max_depth': 16, 'n_estimators': 26}
0.759 (+/-0.102) for {'max_depth': 16, 'n_estimators': 27}
0.759 (+/-0.102) for {'max_depth': 16, 'n_estimators': 28}
0.759 (+/-0.116) for {'max_depth': 16, 'n_estimators': 29}
0.751 (+/-0.130) for {'max_depth': 17, 'n_estimators': 10}
0.755 (+/-0.109) for {'max_depth': 17, 'n_estimators': 11}
0.763 (+/-0.118) for {'max_depth': 17, 'n_estimators': 12}
0.755 (+/-0.116) for {'max_depth': 17, 'n_estimators': 13}
0.751 (+/-0.111) for {'max_depth': 17, 'n_estimators': 14}
0.742 (+/-0.097) for {'max_depth': 17, 'n_estimators': 15}
0.759 (+/-0.108) for {'max_depth': 17, 'n_estimators': 16}
0.751 (+/-0.103) for {'max_depth': 17, 'n_estimators': 17}
0.759 (+/-0.108) for {'max_depth': 17, 'n_estimators': 18}
0.747 (+/-0.092) for {'max_depth': 17, 'n_estimators': 19}
0.759 (+/-0.108) for {'max_depth': 17, 'n_estimators': 20}
0.759 (+/-0.095) for {'max_depth': 17, 'n_estimators': 21}
0.755 (+/-0.106) for {'max_depth': 17, 'n_estimators': 22}
0.768 (+/-0.088) for {'max_depth': 17, 'n_estimators': 23}
0.763 (+/-0.110) for {'max_depth': 17, 'n_estimators': 24}
0.772 (+/-0.108) for {'max_depth': 17, 'n_estimators': 25}
0.763 (+/-0.129) for {'max_depth': 17, 'n_estimators': 26}
0.759 (+/-0.102) for {'max_depth': 17, 'n_estimators': 27}
0.759 (+/-0.102) for {'max_depth': 17, 'n_estimators': 28}
0.759 (+/-0.116) for {'max_depth': 17, 'n_estimators': 29}
0.751 (+/-0.130) for {'max_depth': 18, 'n_estimators': 10}
0.755 (+/-0.109) for {'max_depth': 18, 'n_estimators': 11}
0.763 (+/-0.118) for {'max_depth': 18, 'n_estimators': 12}
0.755 (+/-0.116) for {'max_depth': 18, 'n_estimators': 13}
0.751 (+/-0.111) for {'max_depth': 18, 'n_estimators': 14}
0.742 (+/-0.097) for {'max_depth': 18, 'n_estimators': 15}
0.759 (+/-0.108) for {'max_depth': 18, 'n_estimators': 16}
0.751 (+/-0.103) for {'max_depth': 18, 'n_estimators': 17}
0.759 (+/-0.108) for {'max_depth': 18, 'n_estimators': 18}
0.747 (+/-0.092) for {'max_depth': 18, 'n_estimators': 19}
0.759 (+/-0.108) for {'max_depth': 18, 'n_estimators': 20}
0.759 (+/-0.095) for {'max_depth': 18, 'n_estimators': 21}
0.755 (+/-0.106) for {'max_depth': 18, 'n_estimators': 22}
0.768 (+/-0.088) for {'max_depth': 18, 'n_estimators': 23}
0.763 (+/-0.110) for {'max_depth': 18, 'n_estimators': 24}
0.772 (+/-0.108) for {'max_depth': 18, 'n_estimators': 25}
0.763 (+/-0.129) for {'max_depth': 18, 'n_estimators': 26}
0.759 (+/-0.102) for {'max_depth': 18, 'n_estimators': 27}
0.759 (+/-0.102) for {'max_depth': 18, 'n_estimators': 28}
0.759 (+/-0.116) for {'max_depth': 18, 'n_estimators': 29}
0.751 (+/-0.130) for {'max_depth': 19, 'n_estimators': 10}
0.755 (+/-0.109) for {'max_depth': 19, 'n_estimators': 11}
0.763 (+/-0.118) for {'max_depth': 19, 'n_estimators': 12}
0.755 (+/-0.116) for {'max_depth': 19, 'n_estimators': 13}
0.751 (+/-0.111) for {'max_depth': 19, 'n_estimators': 14}
0.742 (+/-0.097) for {'max_depth': 19, 'n_estimators': 15}
```

```
0.759 (+/-0.108) for {'max_depth': 19, 'n_estimators': 16}
0.751 (+/-0.103) for {'max_depth': 19, 'n_estimators': 17}
0.759 (+/-0.108) for {'max_depth': 19, 'n_estimators': 18}
0.747 (+/-0.092) for {'max_depth': 19, 'n_estimators': 19}
0.759 (+/-0.108) for {'max_depth': 19, 'n_estimators': 20}
0.759 (+/-0.095) for {'max_depth': 19, 'n_estimators': 21}
0.755 (+/-0.106) for {'max_depth': 19, 'n_estimators': 22}
0.768 (+/-0.088) for {'max_depth': 19, 'n_estimators': 23}
0.763 (+/-0.110) for {'max_depth': 19, 'n_estimators': 24}
0.772 (+/-0.108) for {'max_depth': 19, 'n_estimators': 25}
0.763 (+/-0.129) for {'max_depth': 19, 'n_estimators': 26}
0.759 (+/-0.102) for {'max_depth': 19, 'n_estimators': 27}
0.759 (+/-0.102) for {'max_depth': 19, 'n_estimators': 28}
0.759 (+/-0.116) for {'max_depth': 19, 'n_estimators': 29}
0.751 (+/-0.130) for {'max_depth': 20, 'n_estimators': 10}
0.755 (+/-0.109) for {'max_depth': 20, 'n_estimators': 11}
0.763 (+/-0.118) for {'max_depth': 20, 'n_estimators': 12}
0.755 (+/-0.116) for {'max_depth': 20, 'n_estimators': 13}
0.751 (+/-0.111) for {'max_depth': 20, 'n_estimators': 14}
0.742 (+/-0.097) for {'max_depth': 20, 'n_estimators': 15}
0.759 (+/-0.108) for {'max_depth': 20, 'n_estimators': 16}
0.751 (+/-0.103) for {'max_depth': 20, 'n_estimators': 17}
0.759 (+/-0.108) for {'max_depth': 20, 'n_estimators': 18}
0.747 (+/-0.092) for {'max_depth': 20, 'n_estimators': 19}
0.759 (+/-0.108) for {'max_depth': 20, 'n_estimators': 20}
0.759 (+/-0.095) for {'max_depth': 20, 'n_estimators': 21}
0.755 (+/-0.106) for {'max_depth': 20, 'n_estimators': 22}
0.768 (+/-0.088) for {'max_depth': 20, 'n_estimators': 23}
0.763 (+/-0.110) for {'max_depth': 20, 'n_estimators': 24}
0.772 (+/-0.108) for {'max_depth': 20, 'n_estimators': 25}
0.763 (+/-0.129) for {'max_depth': 20, 'n_estimators': 26}
0.759 (+/-0.102) for {'max_depth': 20, 'n_estimators': 27}
0.759 (+/-0.102) for {'max_depth': 20, 'n_estimators': 28}
0.759 (+/-0.116) for {'max_depth': 20, 'n_estimators': 29}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.

              precision    recall  f1-score   support

           0       0.91      0.83      0.87        36
           1       0.78      0.88      0.82        24

    accuracy                           0.85        60
   macro avg       0.84      0.85      0.85        60
weighted avg       0.86      0.85      0.85        60


score:  precision
# Tuning hyper-parameters for precision
```

In [ ]:

```
grader.grade(test_case_id = 'test_rf_tuned', answer = (rf_tuned_precision, rf_tuned_recall, rf_tuned_accuracy))
```

In [124]:
```
grader.grade(test_case_id = 'test_rf_tuned_params', answer = rf_tuned)
```

```
Correct! You earned 1.0/1 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Of the 3 base models that you trained, which model achieved the highest test accuracy?

A. Decision Tree

B. Bagging

C. Random Forest

In [38]:
```
choice = ['C']
```

In [39]:
```
grader.grade(test_case_id = 'test_base', answer = choice)
```

```
Correct! You earned 0.5/0.5 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. Of the 3 models in which you tuned the parameters, which model achieved the highest test accuracy?

A. Decision Tree

B. Bagging

C. Random Forest

In [40]:
```
choice = ['C']
```

In [41]:
```
grader.grade(test_case_id = 'test_tuned', answer = choice)
```

```
Correct! You earned 0.5/0.5 points. You are a star!

Your submission has been successfully recorded in the gradebook.
```

1. *Food for thought (ungraded but interesting to think about)*: In this project, we asked you to pick the model with the highest test accuracy. For this task, do you think that accuracy is the best metric to use, or would precision, recall or even the F1-score be better?

In [ ]:
```

```