ML_SP22_Project_1 (DIY Decision Tree)

Due Date: 4/15 23:59 pm

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
df = pd.read_csv('breast_cancer.csv')
X = df.drop(['diagnosis'], axis=1).to_numpy()
# B is benign and is encoded as 1, M is maligant and is encoded as 0
y = df['diagnosis'].apply(lambda x: 0 if x == 'M' else 1).to_numpy()
```

In [3]:

```
df.describe()
```

Out[3]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetr
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0

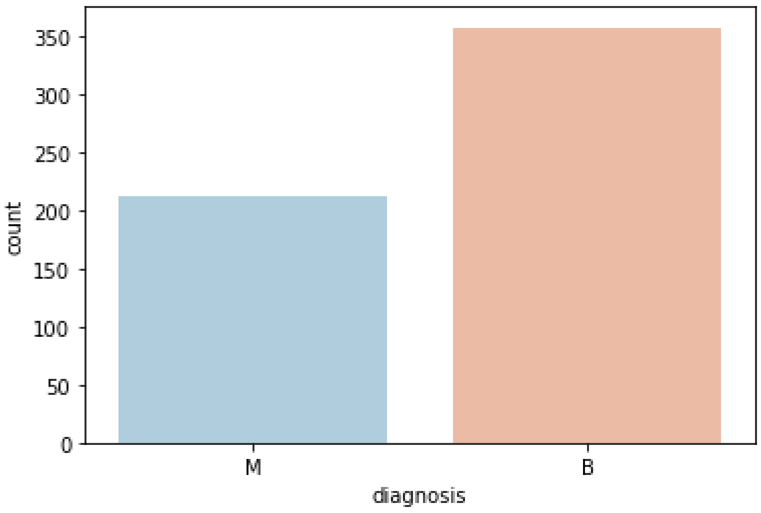
8 rows × 30 columns

In [4]:

```
sns.countplot(x='diagnosis',data=df, palette='RdBu_r')
```

Out[4]:

```
<AxesSubplot:xlabel='diagnosis', ylabel='count'>
```



In [5]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

First build the model with the standard sklearn library

In [6]:

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(max_depth=10)
model.fit(X_train, y_train)
```

Out[6]:

```
DecisionTreeClassifier(max_depth=10)
```

In [7]:

```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
predictions = model.predict(X_test)
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
print(accuracy_score(y_test, predictions))
```

```
[[38  4]
 [ 2 70]]
```

```
              precision    recall  f1-score   support

           0       0.95      0.90      0.93        42
           1       0.95      0.97      0.96        72

    accuracy                           0.95       114
   macro avg       0.95      0.94      0.94       114
weighted avg       0.95      0.95      0.95       114

0.9473684210526315
```

Second use the implementation of the blog to build the model

https://towardsdatascience.com/implementing-a-decision-tree-from-scratch-f5358ff9c4bb

In [8]:

```
from DT_orig import DecisionTree
model = DecisionTree(max_depth=10)
model.fit(X_train, y_train)
```

```
Done fitting
```

In [9]:

```
from DT_orig import accuracy_score
predictions = model.predict(X_test)
print(accuracy_score(y_test, predictions))
```

```
0.956140350877193
```

Note that the original implementation will not work if y is a categorical variable and it is expecting numpy array instead of DataFrame

In [10]:

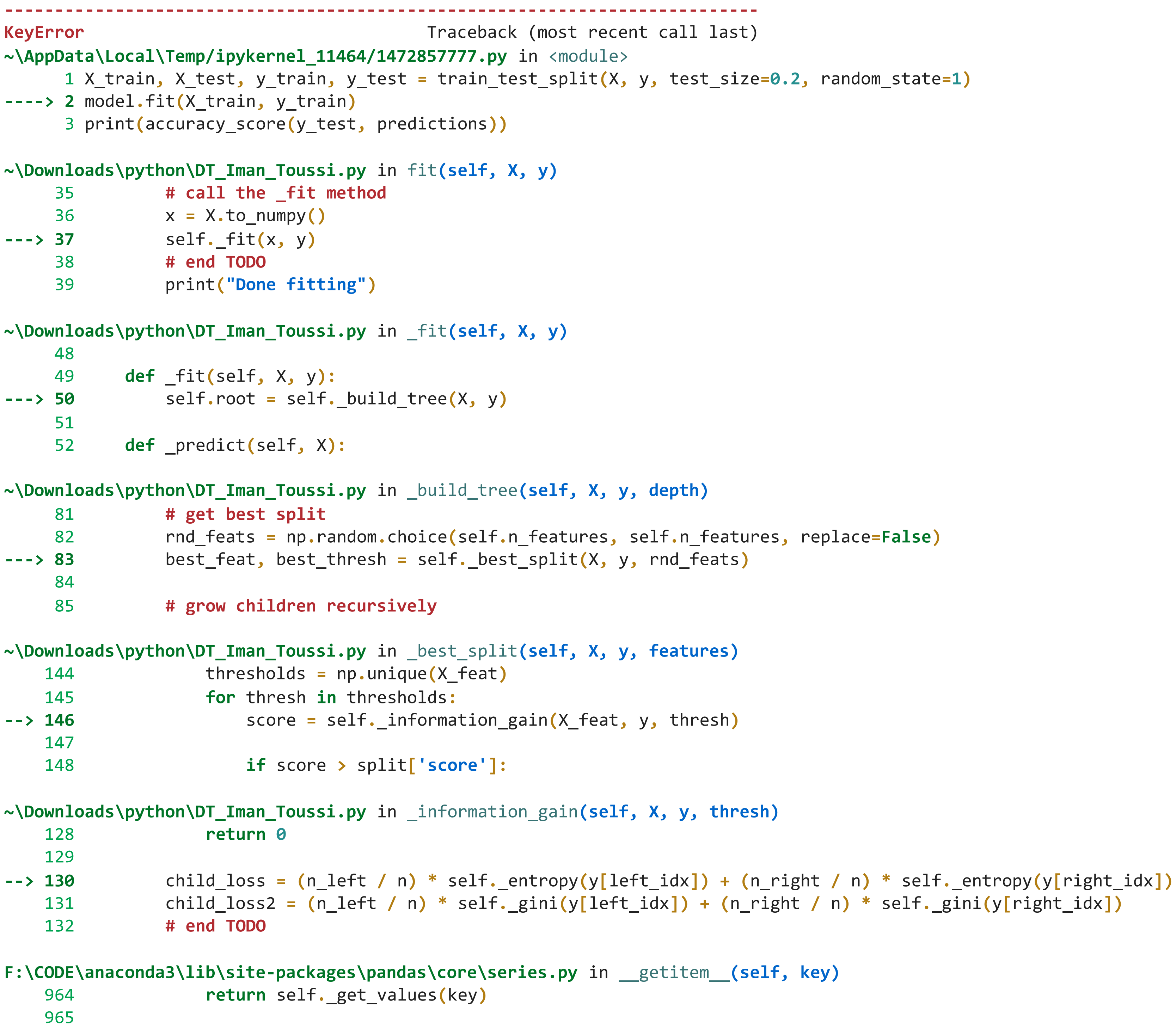
```
X = df.drop(['diagnosis'], axis=1).to_numpy()
#y = df['diagnosis'].apply(lambda x: 0 if x == 'M' else 1).to_numpy()
y = df['diagnosis'].to_numpy()
y[:10]
```

Out[10]:

```
array(['M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M'], dtype=object)
```

In [19]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
model.fit(X_train, y_train)
print(accuracy_score(y_test, predictions))
```



```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_11464/1472857777.py in <module>
      1 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
----> 2 model.fit(X_train, y_train)
      3 print(accuracy_score(y_test, predictions))

~\Downloads\python\DT_Iman_Toussi.py in fit(self, X, y)
     35         # call the _fit method
     36         x = X.to_numpy()
---> 37         self._fit(x, y)
     38         # end TODO
     39         print("Done fitting")

~\Downloads\python\DT_Iman_Toussi.py in _fit(self, X, y)
     48 
     49     def _fit(self, X, y):
---> 50         self.root = self._build_tree(X, y)
     51 
     52     def _predict(self, X):

~\Downloads\python\DT_Iman_Toussi.py in _build_tree(self, X, y, depth)
     81         # get best split
     82         rnd_feats = np.random.choice(self.n_features, self.n_features, replace=False)
---> 83         best_feat, best_thresh = self._best_split(X, y, rnd_feats)
     84 
     85         # grow children recursively

~\Downloads\python\DT_Iman_Toussi.py in _best_split(self, X, y, features)
    144             thresholds = np.unique(X_feat)
    145             for thresh in thresholds:
--> 146                 score = self._information_gain(X_feat, y, thresh)
    147 
    148                 if score > split['score']:

~\Downloads\python\DT_Iman_Toussi.py in _information_gain(self, X, y, thresh)
    128             return 0
    129 
--> 130         child_loss = (n_left / n) * self._entropy(y[left_idx]) + (n_right / n) * self._entropy(y[right_idx])
    131         child_loss2 = (n_left / n) * self._gini(y[left_idx]) + (n_right / n) * self._gini(y[right_idx])
    132         # end TODO

F:\CODE\anaconda3\lib\site-packages\pandas\core\series.py in __getitem__(self, key)
    964             return self._get_values(key)
    965 
```

```
--> 966         return self._get_with(key)
    967 
    968     def _get_with(self, key):

F:\CODE\anaconda3\lib\site-packages\pandas\core\series.py in _get_with(self, key)
    999             #  (i.e. self.iloc) or label-based (i.e. self.loc)
   1000             if not self.index._should_fallback_to_positional():
-> 1001                 return self.loc[key]
   1002             else:
   1003                 return self.iloc[key]

F:\CODE\anaconda3\lib\site-packages\pandas\core\indexing.py in __getitem__(self, key)
    929 
    930             maybe_callable = com.apply_if_callable(key, self.obj)
--> 931             return self._getitem_axis(maybe_callable, axis=axis)
    932 
    933     def _is_scalar_access(self, key: tuple):

F:\CODE\anaconda3\lib\site-packages\pandas\core\indexing.py in _getitem_axis(self, key, axis)
   1151                     raise ValueError("Cannot index with multidimensional key")
   1152 
-> 1153                 return self._getitem_iterable(key, axis=axis)
   1154 
   1155             # nested tuple slicing

F:\CODE\anaconda3\lib\site-packages\pandas\core\indexing.py in _getitem_iterable(self, key, axis)
   1091 
   1092         # A collection of keys
-> 1093         keyarr, indexer = self._get_listlike_indexer(key, axis)
   1094         return self.obj._reindex_with_indexers(
   1095             {axis: [keyarr, indexer]}, copy=True, allow_dups=True

F:\CODE\anaconda3\lib\site-packages\pandas\core\indexing.py in _get_listlike_indexer(self, key, axis)
   1312             keyarr, indexer, new_indexer = ax._reindex_non_unique(keyarr)
   1313 
-> 1314         self._validate_read_indexer(keyarr, indexer, axis)
   1315 
   1316         if needs_i8_conversion(ax.dtype) or isinstance(

F:\CODE\anaconda3\lib\site-packages\pandas\core\indexing.py in _validate_read_indexer(self, key, indexer, axis)
   1375 
   1376             not_found = list(ensure_index(key)[missing_mask.nonzero()[0]].unique())
-> 1377             raise KeyError(f"{not_found} not in index")
   1378 
   1379 
```

```
KeyError: '[62, 65, 111, 207] not in index'
```

Finally use your own improved implementation to build the model

In [12]:

```
from DT_Iman_Toussi import DecisionTreeModel
# replace the above with your version
model = DecisionTreeModel(max_depth=10)

X = df.drop(['diagnosis'], axis=1)
y = df['diagnosis'].apply(lambda x: 0 if x == 'B' else 1)
# make sure your model will work with y being a categorcal variable as well
#y = df['diagnosis']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

In [13]:

```
X_train.head()
```

Out[13]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_n
408	17.99	20.66	117.80	991.7	0.10360	0.13040	0.120100	0.088240	0.
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.198000	0.104300	0.
307	9.00	14.40	56.36	246.3	0.07005	0.03116	0.003681	0.003472	0.
386	12.21	14.09	78.78	462.0	0.08108	0.07823	0.068390	0.025340	0.
404	12.34	14.95	78.29	469.1	0.08682	0.04571	0.021090	0.020540	0.

5 rows × 30 columns

In [14]:

```
y_train.head()
```

Out[14]:

```
408    1
4      1
307    0
386    0
404    0
Name: diagnosis, dtype: int64
```

In [15]:

```
type(X_train)
```

Out[15]:

```
pandas.core.frame.DataFrame
```

In [16]:

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

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_11464/180087699.py in <module>
----> 1 model.fit(X_train, y_train)

~\Downloads\python\DT_Iman_Toussi.py in fit(self, X, y)
     35         # call the _fit method
     36         x = X.to_numpy()
---> 37         self._fit(x, y)
     38         # end TODO
     39         print("Done fitting")

~\Downloads\python\DT_Iman_Toussi.py in _fit(self, X, y)
     48 
     49     def _fit(self, X, y):
---> 50         self.root = self._build_tree(X, y)
     51 
     52     def _predict(self, X):

~\Downloads\python\DT_Iman_Toussi.py in _build_tree(self, X, y, depth)
     81         # get best split
     82         rnd_feats = np.random.choice(self.n_features, self.n_features, replace=False)
---> 83         best_feat, best_thresh = self._best_split(X, y, rnd_feats)
     84 
     85         # grow children recursively

~\Downloads\python\DT_Iman_Toussi.py in _best_split(self, X, y, features)
    144             thresholds = np.unique(X_feat)
    145             for thresh in thresholds:
--> 146                 score = self._information_gain(X_feat, y, thresh)
    147 
    148                 if score > split['score']:

~\Downloads\python\DT_Iman_Toussi.py in _information_gain(self, X, y, thresh)
    128             return 0
    129 
```

```
--> 130         child_loss = (n_left / n) * self._entropy(y[left_idx]) + (n_right / n) * self._entropy(y[right_idx])
    131         child_loss2 = (n_left / n) * self._gini(y[left_idx]) + (n_right / n) * self._gini(y[right_idx])
    132         # end TODO

F:\CODE\anaconda3\lib\site-packages\pandas\core\series.py in __getitem__(self, key)
    964             return self._get_values(key)
    965 
--> 966         return self._get_with(key)
    967 
    968     def _get_with(self, key):

F:\CODE\anaconda3\lib\site-packages\pandas\core\series.py in _get_with(self, key)
    999             #  (i.e. self.iloc) or label-based (i.e. self.loc)
   1000             if not self.index._should_fallback_to_positional():
-> 1001                 return self.loc[key]
   1002             else:
   1003                 return self.iloc[key]

F:\CODE\anaconda3\lib\site-packages\pandas\core\indexing.py in __getitem__(self, key)
    929 
    930             maybe_callable = com.apply_if_callable(key, self.obj)
--> 931             return self._getitem_axis(maybe_callable, axis=axis)
    932 
    933     def _is_scalar_access(self, key: tuple):

F:\CODE\anaconda3\lib\site-packages\pandas\core\indexing.py in _getitem_axis(self, key, axis)
   1151                     raise ValueError("Cannot index with multidimensional key")
   1152 
-> 1153                 return self._getitem_iterable(key, axis=axis)
   1154 
   1155             # nested tuple slicing

F:\CODE\anaconda3\lib\site-packages\pandas\core\indexing.py in _getitem_iterable(self, key, axis)
   1091 
   1092         # A collection of keys
-> 1093         keyarr, indexer = self._get_listlike_indexer(key, axis)
   1094         return self.obj._reindex_with_indexers(
   1095             {axis: [keyarr, indexer]}, copy=True, allow_dups=True

F:\CODE\anaconda3\lib\site-packages\pandas\core\indexing.py in _get_listlike_indexer(self, key, axis)
   1312             keyarr, indexer, new_indexer = ax._reindex_non_unique(keyarr)
   1313 
-> 1314         self._validate_read_indexer(keyarr, indexer, axis)
   1315 
   1316         if needs_i8_conversion(ax.dtype) or isinstance(
F:\CODE\anaconda3\lib\site-packages\pandas\core\indexing.py in _validate_read_indexer(self, key, indexer, axis)
   1372                 if use_interval_msg:
   1373                     key = list(key)
-> 1374                 raise KeyError(f"None of [{key}] are in the [{axis_name}]")
   1375 
   1376             not_found = list(ensure_index(key)[missing_mask.nonzero()[0]].unique())

KeyError: "None of [Int64Index([65], dtype='int64')] are in the [index]"
```

Call your own performance report

In [17]:

```python
from DT_Iman_Toussi import classification_report,confusion_matrix,accuracy_score
predictions = model.predict(X_test)
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
print(accuracy_score(y_test, predictions))
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_11464/3186627248.py in <module>
      1 from DT_Iman_Toussi import classification_report,confusion_matrix,accuracy_score
----> 2 predictions = model.predict(X_test)
      3 print(confusion_matrix(y_test,predictions))
      4 print(classification_report(y_test,predictions))
      5 print(accuracy_score(y_test, predictions))

~\Downloads\python\DT_Iman_Toussi.py in predict(self, X)
     43         # call the predict method
     44         x = X.to_numpy()
---> 45         self._predict(x)
     46         # return ...
     47         # end TODO

~\Downloads\python\DT_Iman_Toussi.py in _predict(self, X)
     51 
     52     def _predict(self, X):
---> 53         predictions = [self._traverse_tree(x, self.root) for x in X]
     54         return np.array(predictions)
     55 

~\Downloads\python\DT_Iman_Toussi.py in <listcomp>(.0)
     51 
     52     def _predict(self, X):
```

```
---> 53         predictions = [self._traverse_tree(x, self.root) for x in X]
     54         return np.array(predictions)
     55 

~\Downloads\python\DT_Iman_Toussi.py in _traverse_tree(self, x, node)
    159         node.
    160         '''
--> 161         if node.is_leaf():
    162             return node.value
    163 

AttributeError: 'NoneType' object has no attribute 'is_leaf'
```

Finally call your RandomForest Model just like the standard sklearn library

In [18]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score

rfc = RandomForestClassifier(n_estimators=100)
rfc.fit(X_train, y_train)
rfc_pred = rfc.predict(X_test)
print(classification_report(y_test, rfc_pred))
print(accuracy_score(y_test, rfc_pred))
```

```
              precision    recall  f1-score   support

           0       0.94      1.00      0.97        72
           1       1.00      0.88      0.94        42

    accuracy                           0.96       114
   macro avg       0.97      0.94      0.95       114
weighted avg       0.96      0.96      0.96       114

0.956140350877193
```

In [44]:

```
# Type your code here
from DT_Iman_Toussi import RandomForestModel
from DT_Iman_Toussi import classification_report,confusion_matrix,accuracy_score

rfc = RandomForestModel(n_estimators=100)
rfc.fit(X_train, y_train)
rfc_pred = rfc.predict(X_test)
```

```
print(classification_report(y_test, rfc_pred))
print(accuracy_score(y_test, rfc_pred))
```

```
Traceback (most recent call last):

  File "F:\CODE\anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line 3444, in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)

  File "C:\Users\psy\AppData\Local\Temp/ipykernel_5272/3218034784.py", line 2, in <module>
    from DT_Iman_Toussi import RandomForestModel

  File "C:\Users\psy\Downloads\python\DT_Iman_Toussi.py", line 46
    def _fit(self, X, y):
    ^
IndentationError: expected an indented block
```

For graduate students only, try different value for the impurity threshold for the Decision Tree Model comment on of the impact of the parameter (if there is any) on the model performance

In [22]:

```
# Type your code here
```

In [ ]:

```

```