ML with sklearn

Reading the data using pandas:

In [1]:
```
import pandas as pd
import numpy as np
df = pd.read_csv('auto.csv')
df.head()
```

Out[1]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8	307.0	130	3504	12.0	70.0	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70.0	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70.0	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70.0	1	amc rebel sst
4	17.0	8	302.0	140	3449	NaN	70.0	1	ford torino

In [2]:
```
print('\nDimensions of data frame:', df.shape)
```

```
Dimensions of data frame: (392, 9)
```

Data Exploration with code

In [3]:
```
df_new = df[['mpg','weight', 'year']]
df_new.describe()
```

Out[3]:

	mpg	weight	year
count	392.000000	392.000000	390.000000
mean	23.445918	2977.584184	76.010256
std	7.805007	849.402560	3.668093
min	9.000000	1613.000000	70.000000
25%	17.000000	2225.250000	73.000000
50%	22.750000	2803.500000	76.000000
75%	29.000000	3614.750000	79.000000
max	46.600000	5140.000000	82.000000

The Range of mpg was from 9 to 46.6 mpg. Its average 23.45 mpg. Range of weigth varies from 1613lb to 5140lb. Its aveage is 2977.58lb. Range of year varies from 70 to 82. Average year was 76.

Exploring the data types

In [4]:
```
df.dtypes
```

Out[4]:
```
mpg             float64
cylinders         int64
displacement    float64
horsepower        int64
weight            int64
acceleration    float64
year            float64
origin            int64
name             object
dtype: object
```

- Mpg, displacement, acceleration, and year are columns with float values.
- Cylinders, horsepower, weight, and origin are columns with int values.
- Name is object type.

In [5]:
```
df.cylinders = df.cylinders.astype('category').cat.codes
print(df.dtypes, "\n")
print(df.head())
```

```
mpg             float64
cylinders          int8
displacement    float64
horsepower        int64
weight            int64
acceleration    float64
year            float64
origin            int64
name             object
dtype: object

    mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0  18.0          4         307.0         130    3504          12.0  70.0
1  15.0          4         350.0         165    3693          11.5  70.0
2  18.0          4         318.0         150    3436          11.0  70.0
3  16.0          4         304.0         150    3433          12.0  70.0
4  17.0          4         302.0         140    3449           NaN  70.0

   origin                       name
0       1  chevrolet chevelle malibu
1       1          buick skylark 320
2       1         plymouth satellite
3       1              amc rebel sst
4       1                ford torino
```

In [6]:
```
df.origin = df.origin.astype('category')
print(df.dtypes, "\n")
print(df.head())
```

```
mpg              float64
cylinders           int8
displacement     float64
horsepower         int64
weight             int64
acceleration     float64
year             float64
origin          category
name              object
dtype: object

    mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0  18.0          4         307.0         130    3504          12.0  70.0
1  15.0          4         350.0         165    3693          11.5  70.0
2  18.0          4         318.0         150    3436          11.0  70.0
3  16.0          4         304.0         150    3433          12.0  70.0
4  17.0          4         302.0         140    3449           NaN  70.0

  origin                       name
0      1  chevrolet chevelle malibu
1      1          buick skylark 320
2      1         plymouth satellite
3      1              amc rebel sst
4      1                ford torino
```

In [7]:
```
print("Null values in the auto columns:")
df.isnull().sum()
```

```
Null values in the auto columns:
```

Out[7]:
```
mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    1
year            2
origin          0
name            0
dtype: int64
```

Drop the NA's

Dealing with NA's

In [8]:
```
df = df.dropna()
df.isnull().sum()
```

Out[8]:
```
mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
year            0
origin          0
name            0
dtype: int64
```

In [9]:
```
print('\nDimensions of data frame:', df.shape)
```

```
Dimensions of data frame: (389, 9)
```

Modifying the columns

In [10]:
```
df['mpg_high'] = np.where(df.mpg > df.mpg.mean(), 1, 0)
df.head()
```

Out[10]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name	mpg_hig
0	18.0	4	307.0	130	3504	12.0	70.0	1	chevrolet chevelle malibu	
1	15.0	4	350.0	165	3693	11.5	70.0	1	buick skylark 320	
2	18.0	4	318.0	150	3436	11.0	70.0	1	plymouth satellite	
3	16.0	4	304.0	150	3433	12.0	70.0	1	amc rebel sst	
6	14.0	4	454.0	220	4354	9.0	70.0	1	chevrolet impala	

In [11]:
```
df = df.drop(columns=['mpg', 'name'])
df.head()
```

Out[11]:

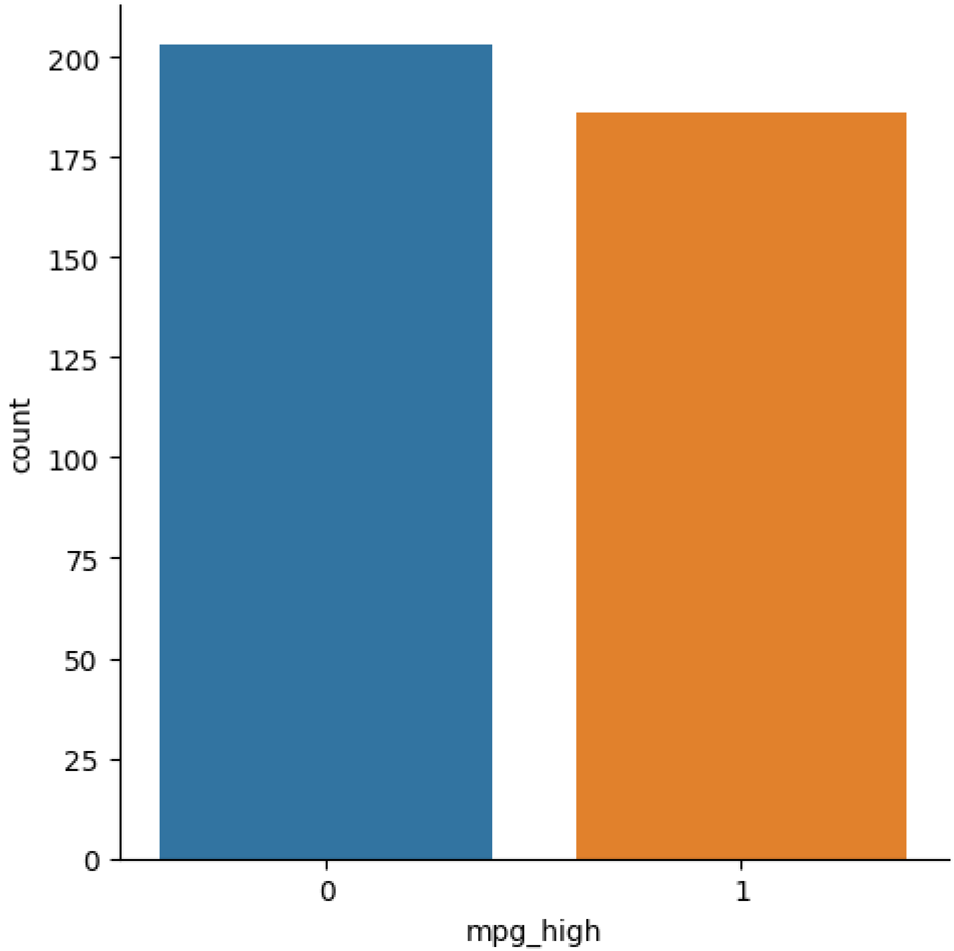
	cylinders	displacement	horsepower	weight	acceleration	year	origin	mpg_high
0	4	307.0	130	3504	12.0	70.0	1	0
1	4	350.0	165	3693	11.5	70.0	1	0
2	4	318.0	150	3436	11.0	70.0	1	0
3	4	304.0	150	3433	12.0	70.0	1	0
6	4	454.0	220	4354	9.0	70.0	1	0

Data exploration with graphs

In [12]:
```
import seaborn as sb
```

In [13]:
```
sb.catplot(data=df, x="mpg_high", kind="count")
```

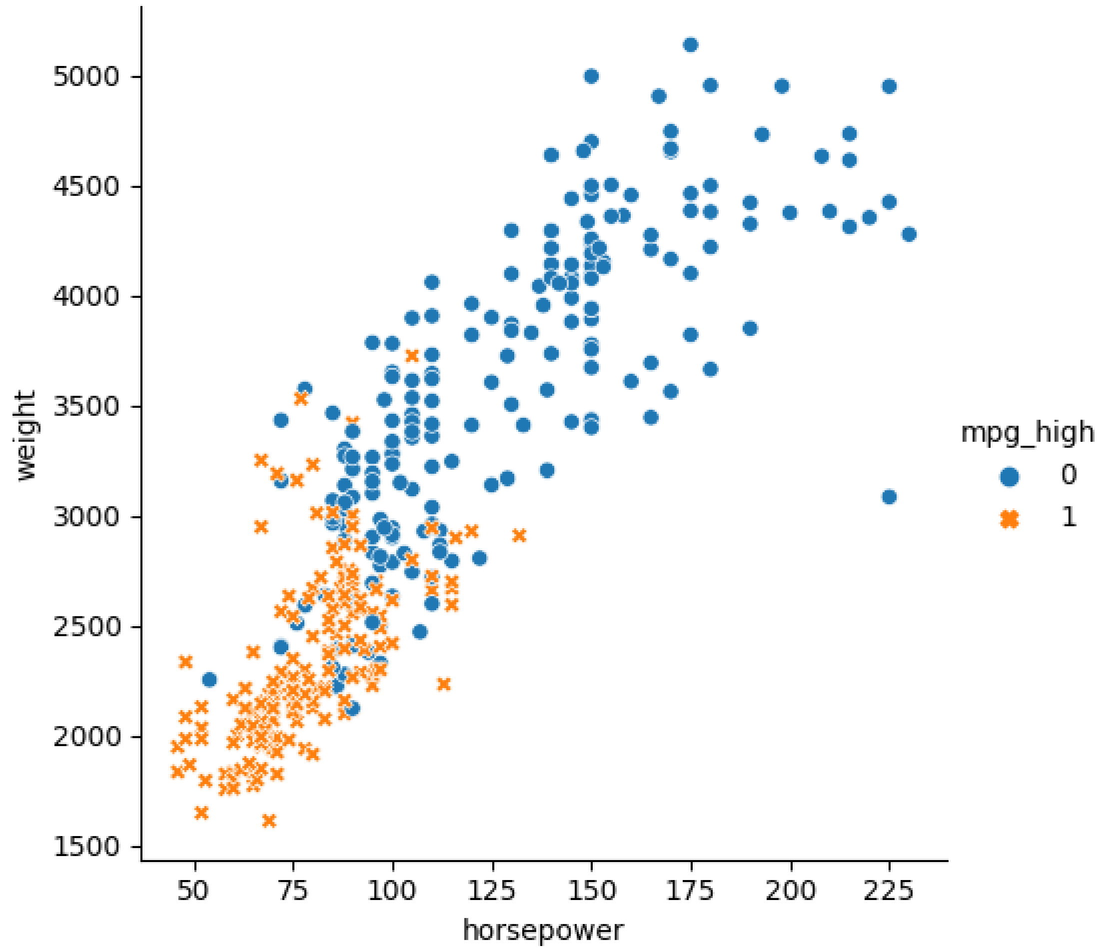
Out[13]: <seaborn.axisgrid.FacetGrid at 0x159aebb1d90>



With this bar graph, we can see that there is a higher number of vehicles that are under the average Mpg. This might set a trend on how the algorithms will predict data.

In [14]:
```
ub.relplot(x='horsepower', y='weight', data=df, hue=df.mpg_high, style=df.mpg_high)
```

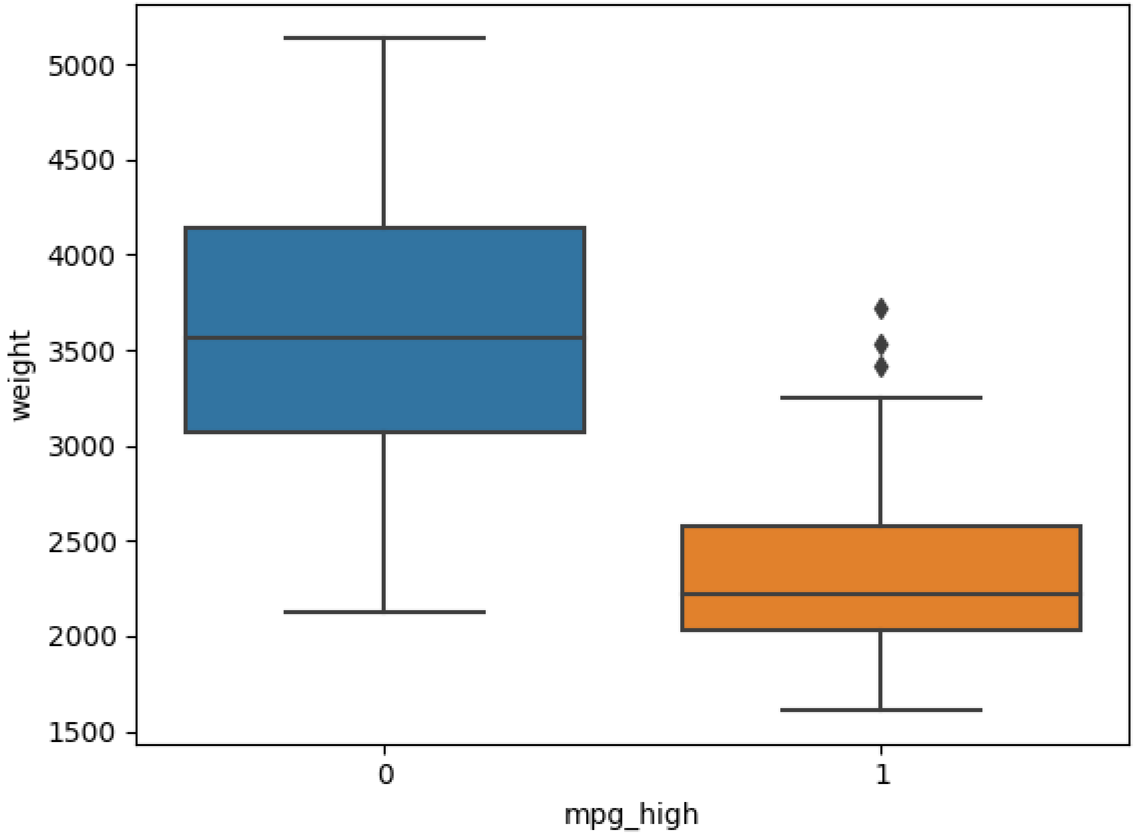
Out[14]: <seaborn.axisgrid.FacetGrid at 0x159aebb1b20>



In the plot graph above, the orange crosses mean cars that have a higher Mpg than the average Mpg. We can see that these cars with a higher Mpg have lower horsepower and weight lower than cars under the average Mpg.

In [39]:
```
sb.boxplot(x='mpg_high', y='weight', data=df)
```

Out[39]: <AxesSubplot: xlabel='mpg_high', ylabel='weight'>



Cars with a higher Mpg than average (0) tend to be clustered together at around a weight of 3000 to 4100 lbs.

Train/Test split

Doing the train/test split

In [16]:
```
from sklearn.model_selection import train_test_splt
```

In [17]:
```
df_y = df.mpg_high
df_x = df.drop(columns=['mpg_high'])

x = df.loc[:, ['cylinders','displacement', 'horsepower', 'weight', 'acceleration', 'ye
y = df.mpg_high

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=

print('X train size:', X_train.shape)
print('X test size:', X_test.shape)
print('Y train size:', y_train.shape)
print('Y test size:', y_test.shape)
```

```
X train size: (311, 7)
X test size: (78, 7)
Y train size: (311,)
Y test size: (78,)
```

Logical regression

In [18]:
```
print('X train size:', X_train.isnull().sum())
print('X test size:', X_test.isnull().sum())
print('Y train size:', y_train.isnull().sum())
print('Y test size:', y_test.isnull().sum())
```

```
X train size: cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
year            0
origin          0
dtype: int64
X test size: cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
year            0
origin          0
dtype: int64
Y train size: 0
Y test size: 0
```

In [19]:
```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(solver="lbfgs")
clf.fit(X_train, y_train)
clf.score(X_train, y_train)
```

Out[19]:
```
0.9003215434083601
```

In [20]:
```
# make predictions

pred = clf.predict(X_test)
```

In [21]:
```
# evaluate
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))
```

```
accuracy score:  0.8717948717948718
precision score:  0.8857142857142857
recall score:  0.8378378378378378
f1 score:  0.8611111111111112
```

In [22]:
```
# confusion matrix
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test, pred)
```

Out[22]:
```
array([[37,  4],
       [ 6, 31]], dtype=int64)
```

In [23]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.90      0.88        41
           1       0.89      0.84      0.86        37

    accuracy                           0.87        78
   macro avg       0.87      0.87      0.87        78
weighted avg       0.87      0.87      0.87        78
```

Decision tree

In [24]:
```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
```

Out[24]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [25]:
```
pred = clf.predict(X_test)
```

In [26]:
```
# evaluate
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))
```

```
accuracy score:  0.9102564102564102
precision score:  0.9166666666666666
recall score:  0.8918918918918919
f1 score:  0.9041095890410958
```

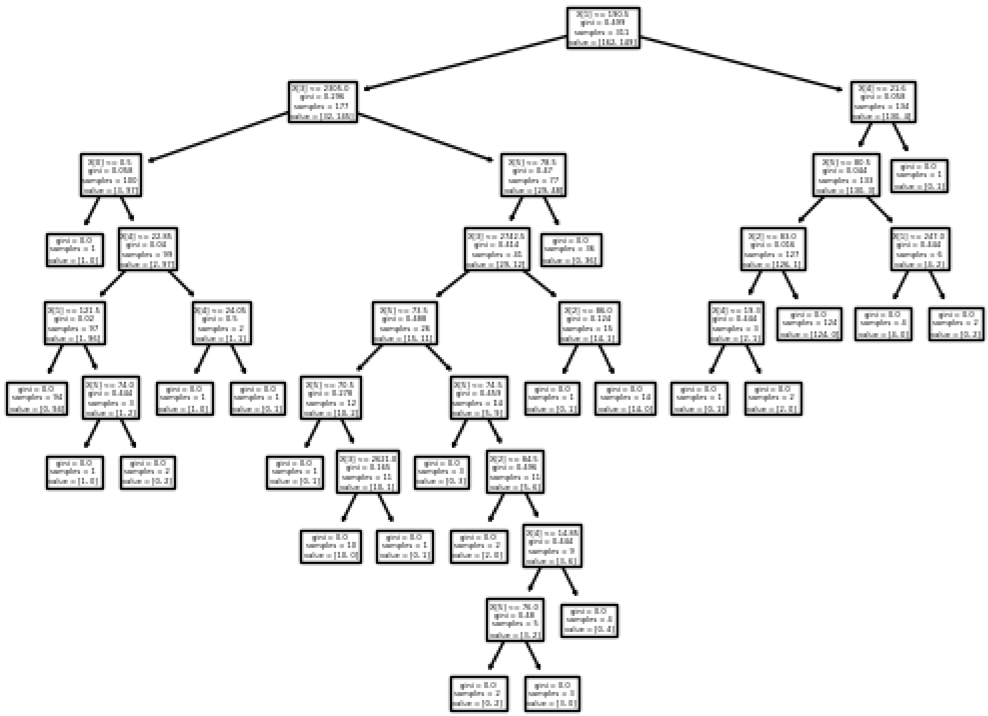
In [27]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.90      0.93      0.92        41
           1       0.92      0.89      0.90        37

    accuracy                           0.91        78
   macro avg       0.91      0.91      0.91        78
weighted avg       0.91      0.91      0.91        78
```

In [28]:
```
from sklearn import tree
tree.plot_tree(clf)
```

Out[28]:
```
[Text(0.6064814814814815, 0.95, 'X[1] <= 190.5\ngini = 0.499\nsamples = 311\nvalue = [162, 149]'),
 Text(0.32407407407407407, 0.85, 'X[3] <= 2305.0\ngini = 0.296\nsamples = 177\nvalue = [32, 145]'),
 Text(0.1111111111111111, 0.75, 'X[0] <= 0.5\ngini = 0.058\nsamples = 100\nvalue = [3, 97]'),
 Text(0.07407407407407407, 0.65, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.14814814814814814, 0.65, 'X[4] <= 22.85\ngini = 0.04\nsamples = 99\nvalue = [2, 97]'),
 Text(0.07407407407407407, 0.55, 'X[1] <= 121.5\ngini = 0.02\nsamples = 97\nvalue = [1, 96]'),
 Text(0.037037037037037035, 0.45, 'gini = 0.0\nsamples = 94\nvalue = [0, 94]'),
 Text(0.1111111111111111, 0.45, 'X[5] <= 74.0\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
 Text(0.07407407407407407, 0.35, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.14814814814814814, 0.35, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.2222222222222222, 0.55, 'X[4] <= 24.05\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.18518518518518517, 0.45, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.25925925925925924, 0.45, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.5370370370370371, 0.75, 'X[5] <= 78.5\ngini = 0.47\nsamples = 77\nvalue = [2
9, 48]'),
 Text(0.5, 0.65, 'X[3] <= 2742.5\ngini = 0.414\nsamples = 41\nvalue = [29, 12]'),
 Text(0.4074074074074074, 0.55, 'X[5] <= 73.5\ngini = 0.488\nsamples = 26\nvalue = [1
5, 11]'),
 Text(0.3333333333333333, 0.45, 'X[5] <= 70.5\ngini = 0.278\nsamples = 12\nvalue = [1
0, 2]'),
 Text(0.2962962962962963, 0.35, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.37037037037037035, 0.35, 'X[3] <= 2621.0\ngini = 0.165\nsamples = 11\nvalue = [10, 1]'),
 Text(0.3333333333333333, 0.25, 'gini = 0.0\nsamples = 10\nvalue = [10, 0]'),
 Text(0.4074074074074074, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.48148148148148145, 0.45, 'X[5] <= 74.5\ngini = 0.459\nsamples = 14\nvalue = [5, 9]'),
 Text(0.4444444444444444, 0.35, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.5185185185185185, 0.35, 'X[2] <= 84.5\ngini = 0.496\nsamples = 11\nvalue = [5, 6]'),
 Text(0.48148148148148145, 0.25, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.5555555555555556, 0.25, 'X[4] <= 14.85\ngini = 0.444\nsamples = 9\nvalue = [3, 6]'),
 Text(0.5185185185185185, 0.15, 'X[5] <= 76.0\ngini = 0.48\nsamples = 5\nvalue = [3, 2]'),
 Text(0.48148148148148145, 0.05, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.5555555555555556, 0.05, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.5925925925925926, 0.15, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.5925925925925926, 0.55, 'X[2] <= 86.0\ngini = 0.124\nsamples = 15\nvalue = [1
4, 1]'),
 Text(0.5555555555555556, 0.45, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.6296296296296297, 0.45, 'gini = 0.0\nsamples = 14\nvalue = [14, 0]'),
 Text(0.5740740740740741, 0.65, 'gini = 0.0\nsamples = 36\nvalue = [0, 36]'),
 Text(0.8888888888888888, 0.85, 'X[4] <= 21.6\ngini = 0.058\nsamples = 134\nvalue = [130, 4]'),
 Text(0.8518518518518519, 0.75, 'X[5] <= 80.5\ngini = 0.044\nsamples = 133\nvalue = [130, 3]'),
 Text(0.7777777777777778, 0.65, 'X[2] <= 83.0\ngini = 0.016\nsamples = 127\nvalue = [126, 1]'),
 Text(0.7407407407407407, 0.55, 'X[4] <= 19.3\ngini = 0.444\nsamples = 3\nvalue = [2,
```

```
1]'),
 Text(0.7037037037037037, 0.45, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.7777777777777778, 0.45, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.8148148148148148, 0.55, 'gini = 0.0\nsamples = 124\nvalue = [124, 0]'),
 Text(0.9259259259259259, 0.65, 'X[1] <= 247.0\ngini = 0.444\nsamples = 6\nvalue =
[4, 2]'),
 Text(0.8888888888888888, 0.55, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]'),
 Text(0.9629629629629629, 0.55, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.9259259259259259, 0.75, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]')]
```



Neural networks

First Model:

In [29]:
```python
# normalize the data
from sklearn import preprocessing

scaler = preprocessing.StandardScaler().fit(X_train)

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [30]:
```python
# train
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5, 2), max_iter=5000, random_s
clf.fit(X_train_scaled, y_train)
```

Out[30]:

```
MLPClassifier
MLPClassifier(hidden_layer_sizes=(5, 2), max_iter=5000, random_state=1234,
              solver='lbfgs')
```

In [31]:

```
# make predictions

pred = clf.predict(X_test_scaled)
```

In [32]:

```
# output results

print('accuracy = ', accuracy_score(y_test, pred))

confusion_matrix(y_test, pred)
```

```
accuracy =  0.8974358974358975
```

Out[32]:

```
array([[39,  2],
       [ 6, 31]], dtype=int64)
```

In [33]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.95      0.91        41
           1       0.94      0.84      0.89        37

    accuracy                           0.90        78
   macro avg       0.90      0.89      0.90        78
weighted avg       0.90      0.90      0.90        78
```

Model 2: Using different topology and settings

In [34]:

```
# try different settings

clf = MLPClassifier(solver='sgd', hidden_layer_sizes=(3,), max_iter=9000, random_state
clf.fit(X_tran_scaled, y_train)
```

Out[34]:

```
MLPClassifier
MLPClassifier(hidden_layer_sizes=(3,), max_iter=9000, random_state=1234,
              solver='sgd')
```

In [35]:

```
# make predictions
pred = clf.predict(X_test_scaled)

print('accuracy = ', accuracy_score(y_test, pred))

# confusion matrix
confusion_matrix(y_test, pred)
```

```
accuracy =  0.8974358974358975
```

```
Out[35]: array([[37,  4],
                [ 4, 33]], dtype=int64)
```

```
In [36]: print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.90      0.90      0.90        41
           1       0.89      0.89      0.89        37

    accuracy                           0.90        78
   macro avg       0.90      0.90      0.90        78
weighted avg       0.90      0.90      0.90        78
```

Comparing the two neural network models, we see that the using the sgd solver at a higher max number of iterations does not provide a significant improvement. In fact, for the lbfgs solver, it was more precise on predicting 1's. It is to be noted that sgd solver had a higher precision. Both models had the same accuracy predicting the test set.

Analysis

- a. From all the algotirthms, the one that performed the best was the decision tree. The neural networks had an accuracy of 0.89 and the logical regression model came in at 0.87
- b. Accuracy between the algorithms was really similar. The decision tree model had an accuracy of 0.91. The neural network model had an accuracy of 0.89. The logical regression model had an accuracy of 0.87. The recall and precision metrics were also similar. The decision tree had a recall score of 0.89 and precision metric of 0.92.The neural network model had a recall score of 0.90 and a precision score of 0.90. The logical regression model had a recall score of 0.84 and a precission score of 0.89.
- c. I believe that the decision tree performed better because of the type of data we were given. Neural networks tend to perform better in larger sets of data. Logical regression performs well in small sets of data that don't have many out of range variables. The decision tree performs well in both cases but tends to overfit the data.
- d. After using both R and sklearn, I feel it is easier to use sklearn than R, in some cases. It is easier to do certain models in R but some algorithms do take longer to run in R, compared to sklearn. It is also more intuitive using sklearn, as it is easier to set up data and modify it, something I struggled with in R. Overall, I believe sklearn is a friendlier way of doing machine learning.