SKLearn

April 12, 2023

[302]:
```python
import pandas as pd

#Read the Auto Data
df=pd.read_csv('https://raw.githubusercontent.com/Benton7/CS4375_Portfolio/main/
 ↪Auto.csv')

#Output the first few rows
print("Auto data: \n", df.head())

#Output the dimensions of the data
print("Dimensions of data: \n", df.shape)
```

```
Auto data:
     mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0  18.0          8         307.0         130    3504          12.0  70.0
1  15.0          8         350.0         165    3693          11.5  70.0
2  18.0          8         318.0         150    3436          11.0  70.0
3  16.0          8         304.0         150    3433          12.0  70.0
4  17.0          8         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
Dimensions of data:
 (392, 9)
```

[303]:
```python
#Use describe() on the mpg, weight, and year columns
ddf=df[['mpg', 'weight', 'year']]
print("Using describe() on the mpg, weight, and year: \n", ddf.describe())

#Write comments indicating the range and average of each column
print('\nmpg range: ', df['mpg'].max() - df['mpg'].min())
print('mpg average: ', df['mpg'].mean())
print('weight range: ', df['weight'].max() - df['weight'].min())
print('weight average: ', df['weight'].mean())
```

```
print('year range: ', df['year'].max() - df['year'].min())
print('year average: ', df['year'].mean())
```

```
Using describe() on the mpg, weight, and year:
              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

mpg range:  37.6
mpg average:  23.445918367346938
weight range:  3527
weight average:  2977.5841836734694
year range:  12.0
year average:  76.01025641025642
```

[304]:

```
#Check the data types of all columns
print("Data type of each column: \n", df.dtypes)

#Change the cylinders column to categorical(use cat.codes)
df.cylinders = df.cylinders.astype('category').cat.codes

#Change the origins column to categorical(don't use cat.codes)
df.origin = df.origin.astype('category')

#Verify the changes with the dtypes attribute
print("Data type of each column(after categorical changes): \n", df.dtypes)
```

```
Data type of each column:
 mpg             float64
cylinders         int64
displacement    float64
horsepower        int64
weight            int64
acceleration    float64
year            float64
origin            int64
name             object
dtype: object
Data type of each column(after categorical changes):
 mpg              float64
cylinders           int8
displacement     float64
```

```
horsepower          int64
weight              int64
acceleration      float64
year              float64
origin           category
name               object
dtype: object
```

[305]:

```
#Delete rows with NAs
df = df.dropna()

#Output the new dimensions
print("New dimensions(after deleting rows with NAs): \n", df.shape)
```

```
New dimensions(after deleting rows with NAs):
 (389, 9)
```

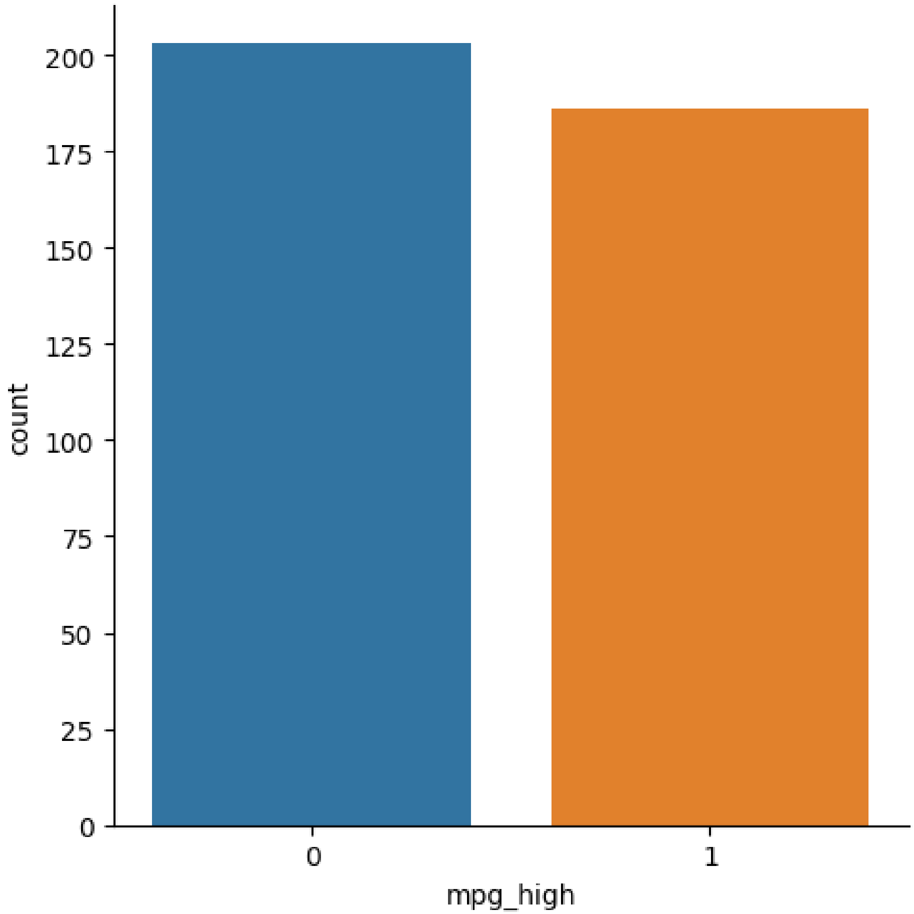
[306]:

```
#Make a new column, mpg_high, and make it categorical(the column==1 if mpg >␣
↪average mpg, else = 0
avg_mpg= df['mpg'].mean()
df['mpg_high'] = pd.cut(df['mpg'], bins=[0, avg_mpg, float('Inf')],␣
↪labels=[0,1])

#Delete the mpg and name columns(delete mpg so the algorithm doesn't just learn␣
↪to predict mpg_high from mpg)
df.drop('mpg', axis=1, inplace=True)
df.drop('name', axis=1, inplace=True)

#Output the first few rows of the modified data frame
print("First few rows of data(after removing two columns and creating one more):
↪ \n", df.head())
```

```
First few rows of data(after removing two columns and creating one more):
    cylinders  displacement  horsepower  weight  acceleration  year origin  \
0          4         307.0         130    3504          12.0  70.0      1
1          4         350.0         165    3693          11.5  70.0      1
2          4         318.0         150    3436          11.0  70.0      1
3          4         304.0         150    3433          12.0  70.0      1
6          4         454.0         220    4354           9.0  70.0      1

  mpg_high
0        0
1        0
2        0
3        0
6        0
```

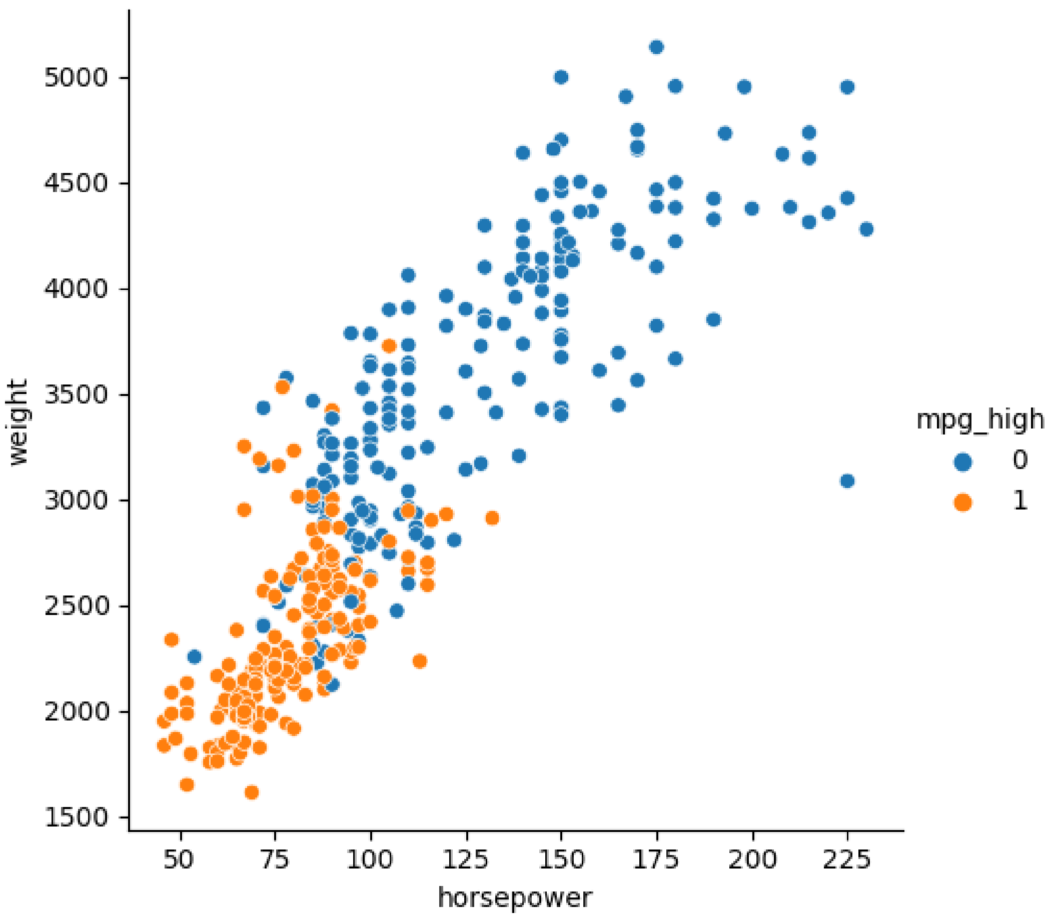
[307]:
```
import seaborn as sb
#Seaborn catplot on the mpg_high column
sb.catplot(x='mpg_high', kind='count', data=df)
```

[307]: <seaborn.axisgrid.FacetGrid at 0x23adb5b01d0>



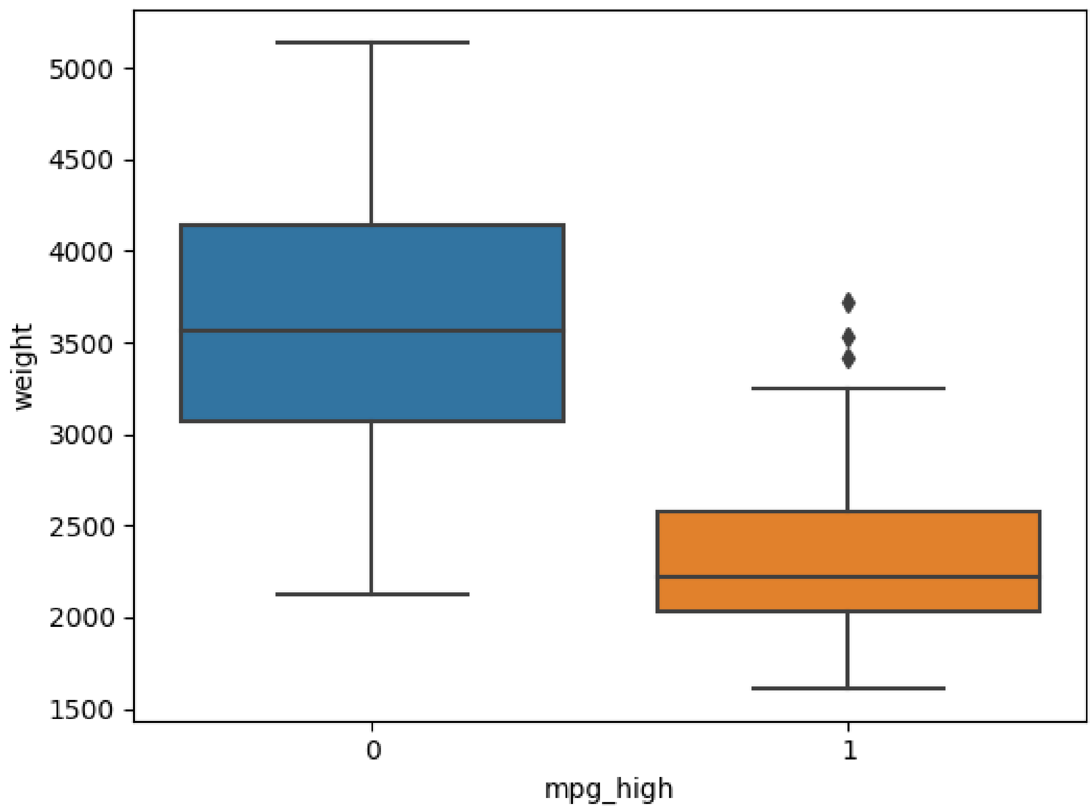
[308]:
```
#Seaborn relplot with horsepower on the x axis, weight on the y axis, setting␣
↪hue or style to mpg_high
sb.relplot(x='horsepower', y='weight', data=df, hue=df.mpg_high)
```

[308]: <seaborn.axisgrid.FacetGrid at 0x23adb5bd090>



[309]:
```
#Seaborn boxplot with mpg_high on the x axis and weight on the y axis
sb.boxplot(x='mpg_high', y='weight', data=df)
```

[309]: <Axes: xlabel='mpg_high', ylabel='weight'>



[310]:
```
#Train/test split 80/20
from sklearn.model_selection import train_test_split

#Use seed 1234 so we all get the same results
#Train/test x data frames consists of all remaining columns except mpg_high
x=df.drop('mpg_high', axis=1)
y=df['mpg_high']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.2,␣
↪random_state=1234)

#Output the dimensions of train and test
print('Train dimensions: ', x_train.shape)
print('Test dimensions: ', y_train.shape)
```

```
Train dimensions:  (311, 7)
Test dimensions:  (311,)
```

[311]:
```
#Train a logistic regression model using solver lbfgs
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
lr = LogisticRegression(solver='lbfgs')
```

```
lr.fit(x_train, y_train)
lr.score(x_train, y_train)

#Test and evaluate
lr_pred = lr.predict(x_test)

#Print metrics using the classification report
print(classification_report(y_test, lr_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.80      0.88        50
           1       0.73      0.96      0.83        28

    accuracy                           0.86        78
   macro avg       0.85      0.88      0.85        78
weighted avg       0.89      0.86      0.86        78
```

[312]:
```
#Train a decision tree
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(x_train, y_train)

#Test and evaluate
dt_pred = dt.predict(x_test)

#Print the classification report metrics
print(classification_report(y_test, dt_pred))

#Plot the tree(optional)
#from sklearn import tree
tree.plot_tree(dt)
```

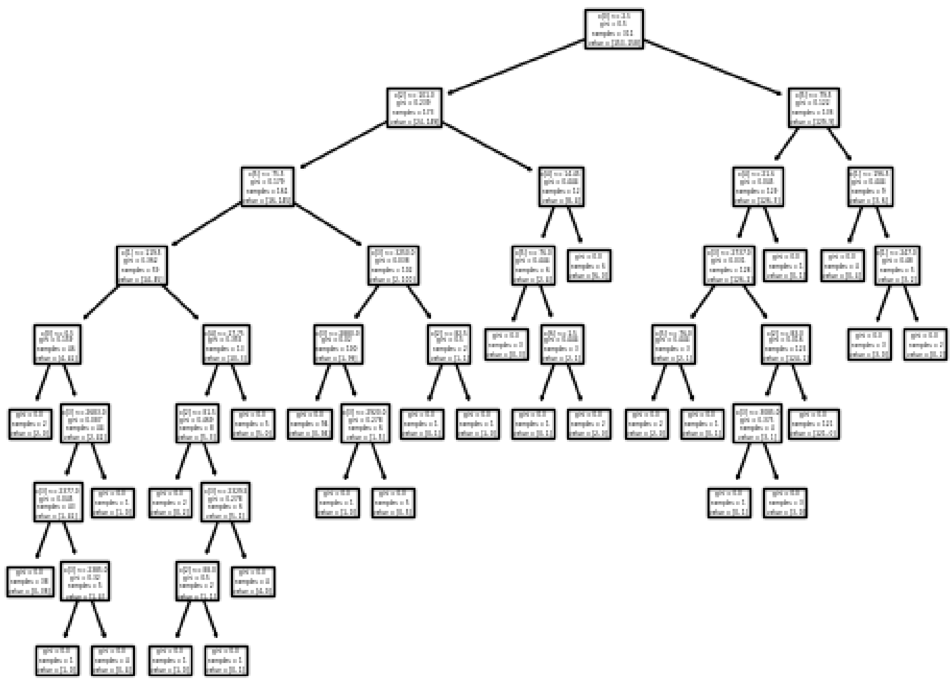
```
              precision    recall  f1-score   support

           0       0.96      0.92      0.94        50
           1       0.87      0.93      0.90        28

    accuracy                           0.92        78
   macro avg       0.91      0.92      0.92        78
weighted avg       0.93      0.92      0.92        78
```

[312]:
```
[Text(0.6433823529411765, 0.9444444444444444, 'x[0] <= 2.5\ngini = 0.5\nsamples
= 311\nvalue = [153, 158]'),
 Text(0.4338235294117647, 0.8333333333333334, 'x[2] <= 101.0\ngini =
0.239\nsamples = 173\nvalue = [24, 149]'),
```

```
 Text(0.27941176470588236, 0.7222222222222222, 'x[5] <= 75.5\ngini =
0.179\nsamples = 161\nvalue = [16, 145]'),
 Text(0.14705882352941177, 0.6111111111111112, 'x[1] <= 119.5\ngini =
0.362\nsamples = 59\nvalue = [14, 45]'),
 Text(0.058823529411764705, 0.5, 'x[0] <= 0.5\ngini = 0.159\nsamples = 46\nvalue
= [4, 42]'),
 Text(0.029411764705882353, 0.3888888888888889, 'gini = 0.0\nsamples = 2\nvalue
= [2, 0]'),
 Text(0.08823529411764706, 0.3888888888888889, 'x[3] <= 2683.0\ngini =
0.087\nsamples = 44\nvalue = [2, 42]'),
 Text(0.058823529411764705, 0.2777777777777778, 'x[3] <= 2377.0\ngini =
0.045\nsamples = 43\nvalue = [1, 42]'),
 Text(0.029411764705882353, 0.16666666666666666, 'gini = 0.0\nsamples =
38\nvalue = [0, 38]'),
 Text(0.08823529411764706, 0.16666666666666666, 'x[3] <= 2385.0\ngini =
0.32\nsamples = 5\nvalue = [1, 4]'),
 Text(0.058823529411764705, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nvalue
= [1, 0]'),
 Text(0.11764705882352941, 0.05555555555555555, 'gini = 0.0\nsamples = 4\nvalue
= [0, 4]'),
 Text(0.11764705882352941, 0.2777777777777778, 'gini = 0.0\nsamples = 1\nvalue =
[1, 0]'),
 Text(0.23529411764705882, 0.5, 'x[4] <= 17.75\ngini = 0.355\nsamples =
13\nvalue = [10, 3]'),
 Text(0.20588235294117646, 0.3888888888888889, 'x[2] <= 81.5\ngini =
0.469\nsamples = 8\nvalue = [5, 3]'),
 Text(0.17647058823529413, 0.2777777777777778, 'gini = 0.0\nsamples = 2\nvalue =
[0, 2]'),
 Text(0.23529411764705882, 0.2777777777777778, 'x[3] <= 2329.5\ngini =
0.278\nsamples = 6\nvalue = [5, 1]'),
 Text(0.20588235294117646, 0.16666666666666666, 'x[2] <= 88.0\ngini =
0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.17647058823529413, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nvalue
= [1, 0]'),
 Text(0.23529411764705882, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nvalue
= [0, 1]'),
 Text(0.2647058823529412, 0.16666666666666666, 'gini = 0.0\nsamples = 4\nvalue =
[4, 0]'),
 Text(0.2647058823529412, 0.3888888888888889, 'gini = 0.0\nsamples = 5\nvalue =
[5, 0]'),
 Text(0.4117647058823529, 0.6111111111111112, 'x[3] <= 3250.0\ngini =
0.038\nsamples = 102\nvalue = [2, 100]'),
 Text(0.35294117647058826, 0.5, 'x[3] <= 2880.0\ngini = 0.02\nsamples =
100\nvalue = [1, 99]'),
 Text(0.3235294117647059, 0.3888888888888889, 'gini = 0.0\nsamples = 94\nvalue =
[0, 94]'),
 Text(0.3823529411764706, 0.3888888888888889, 'x[3] <= 2920.0\ngini =
```

```
0.278\nsamples = 6\nvalue = [1, 5]'),
 Text(0.35294117647058826, 0.2777777777777778, 'gini = 0.0\nsamples = 1\nvalue =
[1, 0]'),
 Text(0.4117647058823529, 0.2777777777777778, 'gini = 0.0\nsamples = 5\nvalue =
[0, 5]'),
 Text(0.47058823529411764, 0.5, 'x[2] <= 82.5\ngini = 0.5\nsamples = 2\nvalue =
[1, 1]'),
 Text(0.4411764705882353, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.5, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.5882352941176471, 0.7222222222222222, 'x[4] <= 14.45\ngini =
0.444\nsamples = 12\nvalue = [8, 4]'),
 Text(0.5588235294117647, 0.6111111111111112, 'x[5] <= 76.0\ngini =
0.444\nsamples = 6\nvalue = [2, 4]'),
 Text(0.5294117647058824, 0.5, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.5882352941176471, 0.5, 'x[6] <= 1.5\ngini = 0.444\nsamples = 3\nvalue =
[2, 1]'),
 Text(0.5588235294117647, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.6176470588235294, 0.3888888888888889, 'gini = 0.0\nsamples = 2\nvalue =
[2, 0]'),
 Text(0.6176470588235294, 0.6111111111111112, 'gini = 0.0\nsamples = 6\nvalue =
[6, 0]'),
 Text(0.8529411764705882, 0.8333333333333334, 'x[5] <= 79.5\ngini =
0.122\nsamples = 138\nvalue = [129, 9]'),
 Text(0.7941176470588235, 0.7222222222222222, 'x[4] <= 21.6\ngini =
0.045\nsamples = 129\nvalue = [126, 3]'),
 Text(0.7647058823529411, 0.6111111111111112, 'x[3] <= 2737.0\ngini =
0.031\nsamples = 128\nvalue = [126, 2]'),
 Text(0.7058823529411765, 0.5, 'x[5] <= 76.0\ngini = 0.444\nsamples = 3\nvalue =
[2, 1]'),
 Text(0.6764705882352942, 0.3888888888888889, 'gini = 0.0\nsamples = 2\nvalue =
[2, 0]'),
 Text(0.7352941176470589, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.8235294117647058, 0.5, 'x[2] <= 83.0\ngini = 0.016\nsamples = 125\nvalue
= [124, 1]'),
 Text(0.7941176470588235, 0.3888888888888889, 'x[3] <= 3085.0\ngini =
0.375\nsamples = 4\nvalue = [3, 1]'),
 Text(0.7647058823529411, 0.2777777777777778, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.8235294117647058, 0.2777777777777778, 'gini = 0.0\nsamples = 3\nvalue =
[3, 0]'),
 Text(0.8529411764705882, 0.3888888888888889, 'gini = 0.0\nsamples = 121\nvalue
= [121, 0]'),
 Text(0.8235294117647058, 0.6111111111111112, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
```

```
 Text(0.9117647058823529, 0.7222222222222222, 'x[1] <= 196.5\ngini =
0.444\nsamples = 9\nvalue = [3, 6]'),
 Text(0.8823529411764706, 0.6111111111111112, 'gini = 0.0\nsamples = 4\nvalue =
[0, 4]'),
 Text(0.9411764705882353, 0.6111111111111112, 'x[1] <= 247.0\ngini =
0.48\nsamples = 5\nvalue = [3, 2]'),
 Text(0.9117647058823529, 0.5, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.9705882352941176, 0.5, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]')]
```



[313]:
```python
#Train a neural network, choosing a network topology of your choice
from sklearn.neural_network import MLPClassifier
from sklearn import preprocessing
scaler = preprocessing.StandardScaler().fit(x_train)
x_train_scaled = scaler.transform(x_train)
x_test_scaled = scaler.transform(x_test)
nn = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(30, 15), max_iter=500,
 ↪random_state=1234)
nn.fit(x_train_scaled, y_train)

#Test and evaluate
nn_pred = nn.predict(x_test_scaled)
print(classification_report(y_test, nn_pred))
```

```
              precision    recall  f1-score   support
```

```
              0       0.90      0.90      0.90        50
              1       0.82      0.82      0.82        28

       accuracy                           0.87        78
      macro avg       0.86      0.86      0.86        78
   weighted avg       0.87      0.87      0.87        78
```

[314]:
```
#Train a second network with a different topology and different settings
nn = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(12,6,2), max_iter=500,
 ↪random_state=1234)
nn.fit(x_train_scaled, y_train)

#Test and evaluate
nn_pred = nn.predict(x_test_scaled)
print(classification_report(y_test, nn_pred))
```

```
              precision    recall  f1-score   support

              0       0.91      0.86      0.89        50
              1       0.77      0.86      0.81        28

       accuracy                           0.86        78
      macro avg       0.84      0.86      0.85        78
   weighted avg       0.86      0.86      0.86        78
```

[315]:
```
#These results show us that the more nodes produce slightly better results. The
 ↪.01 increase in precision proves this fact about the initial model
 ↪performing better with more nodes.
```

[316]:
```
###Analysis
#Which algorithm performed better?
#Overall, the algorithms performed about the same.
```

[317]:
```
#Compare accuracy, recall, and precision metrics by class
#The Neural Network and Decision Trees had roughly the same metrics, however,
 ↪the Logistic Regression model had a higher precision and lower recall and
 ↪f1-score.
```

[319]:
```
#Write a couple of sentences comparing your experiences using R versus sklearn.
#I found it to be much easier to write in python using sklearn in comparison to
 ↪R. Python was much faster in terms of training and testing models than R. I
 ↪will admit to some bias due to my familiarity with Python making this whole
 ↪project easier is a large reason why I enjoyed it more. It was fun to get to
 ↪mess around with these data libraries in a language that I felt comfortable
 ↪in.
```