ML_with_SKLearn_

April 8, 2023

1 Machine Learning with sklearn

[ ]:
```
import pandas as pd
import io
import seaborn as sb
```

1.1 Read the Auto Data

[ ]:
```
from google.colab import files
uploaded=files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving Auto.csv to Auto.csv
```

[ ]:
```
df = pd.read_csv(io.BytesIO(uploaded['Auto.csv']))
print(df)
```

```
      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
..    …          …           …          …     …           …   …
387  27.0          4         140.0          86    2790          15.6  82.0
388  44.0          4          97.0          52    2130          24.6  82.0
389  32.0          4         135.0          84    2295          11.6  82.0
390  28.0          4         120.0          79    2625          18.6  82.0
391  31.0          4         119.0          82    2720          19.4  82.0

     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
..      …                        …
```

```
387        1             ford mustang gl
388        2                   vw pickup
389        1               dodge rampage
390        1                 ford ranger
391        1                  chevy s-10

[392 rows x 9 columns]
```

[ ]:
```
df.head()
```

[ ]:
```
    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
```

[ ]:
```
print(df.shape)
```

```
(392, 9)
```

##Data Exploration with code

[ ]:
```
print(df[['mpg', 'year', 'weight']].describe())
'''
For the mpg, year and weight columns the:
averages are 392, 290 and 392 respectively
ranges are 37.2, 12 and 3572 respectively
'''
```

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

[ ]: '\nFor the mpg, year and weight columns the:\naverages are 392, 290 and 392 respectively\nranges are 37.2, 12 and 3572 respectively\n'

##Explore data types

[ ]:
```python
print(df.dtypes)
```

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

[ ]:
```python
df.cylinders = df.cylinders.astype('category').cat.codes
print(df.dtypes)
```

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

[ ]:
```python
df.origin = df.origin.astype('category')
print(df.dtypes)
```

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

##Deal with NAs

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

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

##Modify columns

[ ]:
```
avg_mpg = df['mpg'].mean()
df['mpg_high'] = (df['mpg'] > avg_mpg).astype('category')
print(df.dtypes)
df.head()
```

```
mpg              float64
cylinders           int8
displacement     float64
horsepower         int64
weight             int64
acceleration     float64
year             float64
origin          category
name              object
mpg_high        category
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
6  14.0          4         454.0         220    4354           9.0  70.0

  origin                       name mpg_high
0      1  chevrolet chevelle malibu    False
1      1          buick skylark 320    False
2      1         plymouth satellite    False
3      1              amc rebel sst    False
6      1           chevrolet impala    False
```

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

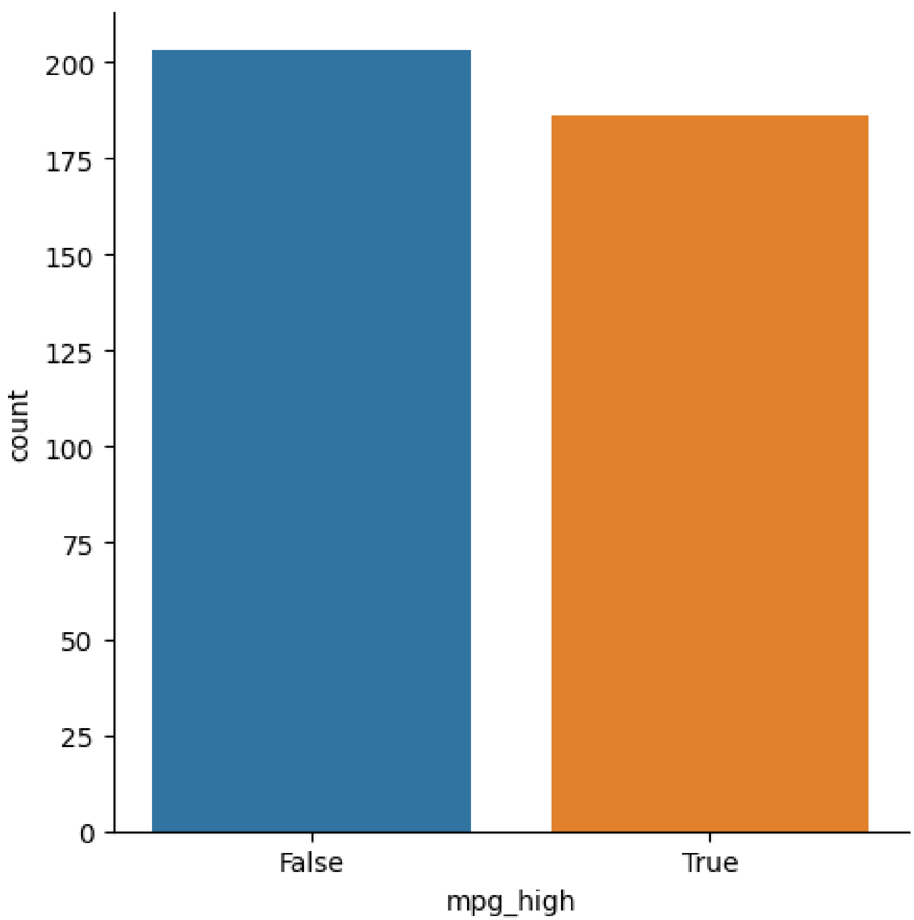
```
   cylinders  displacement  horsepower  weight  acceleration  year origin  \
0          4         307.0         130    3504          12.0  70.0      1
1          4         350.0         165    3693          11.5  70.0      1
2          4         318.0         150    3436          11.0  70.0      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    False
1    False
2    False
3    False
6    False
```

##Data exploration with graphs

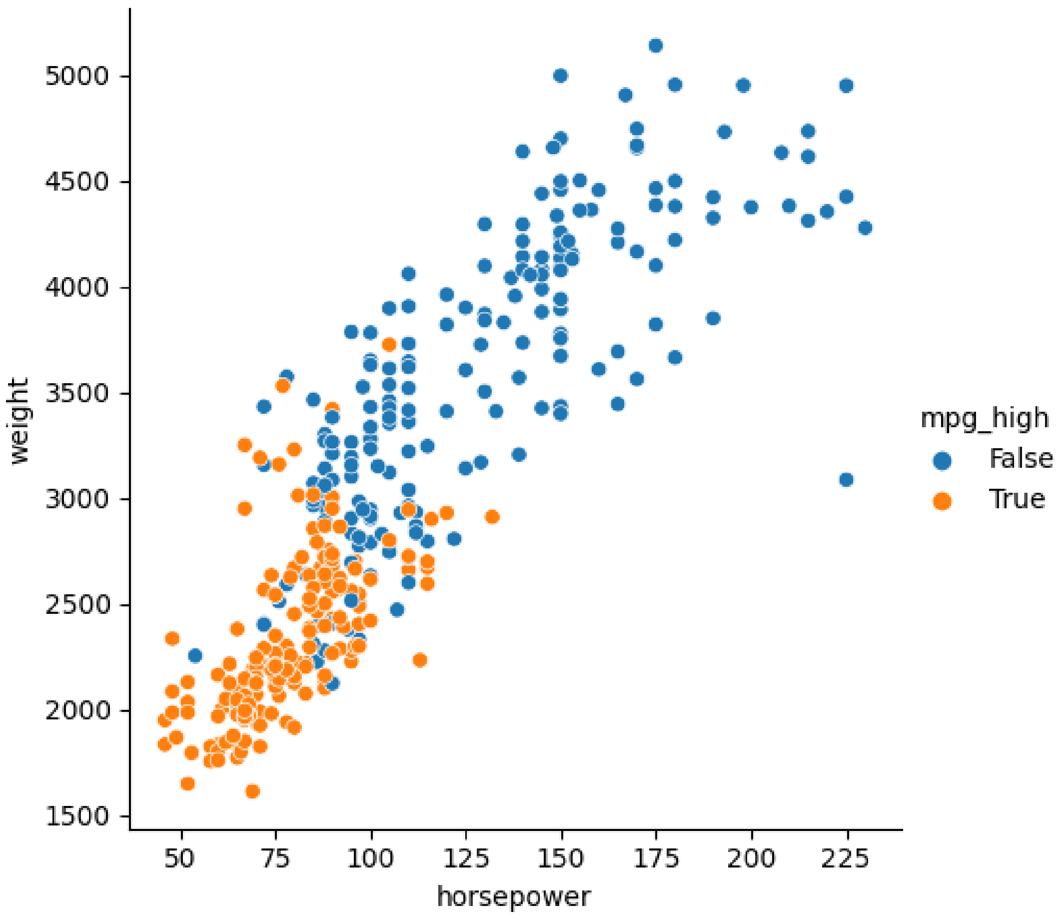
```
[ ]: sb.catplot(x="mpg_high", kind='count', data=df)
     #this tells us the number of cars that have a higher than average mpg(true) and␣
      ↪ones that have lower.
     #We can see from the graph that about 180 cars have a higher than average mpg␣
      ↪and about 200 have lower. Its almost the same number.
```

```
[ ]: <seaborn.axisgrid.FacetGrid at 0x7fde24572640>
```



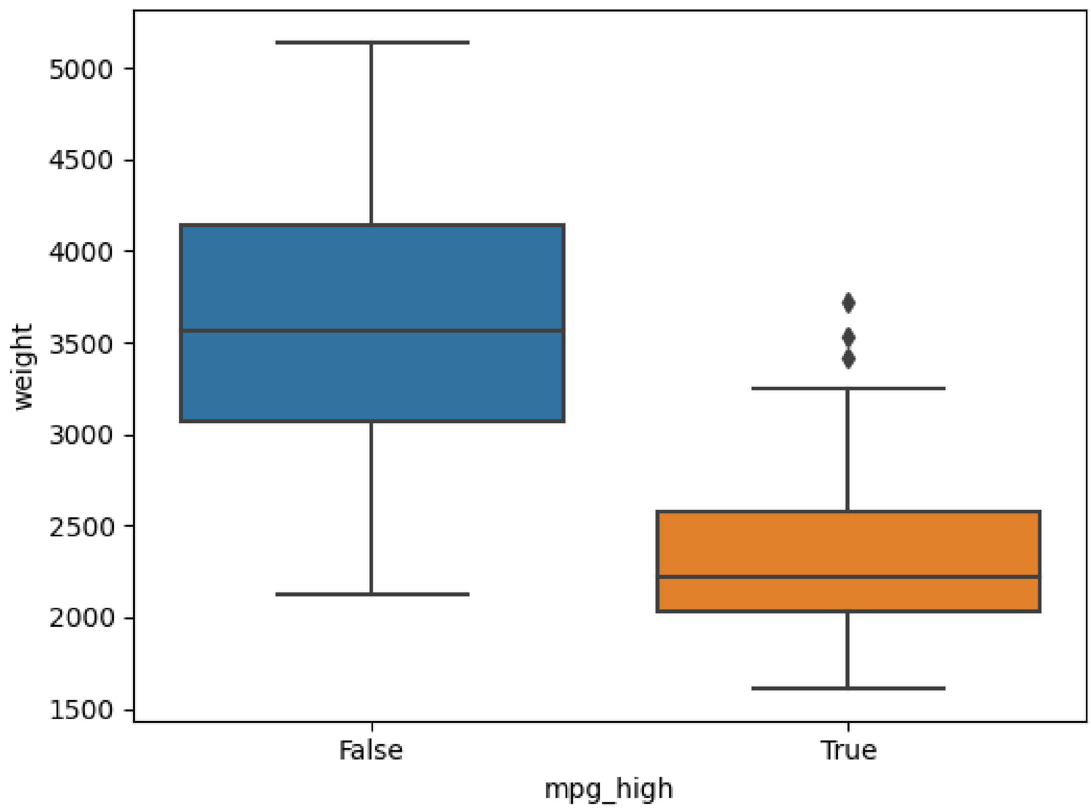
```
[ ]: sb.relplot(x='horsepower', y='weight', data=df, hue=df.mpg_high)
     #We want to see the relationship between horsepower and weight. And also if␣
      ↪factoring in mpg_high makes a difference. We notice an almost linear
     #relationship between horsepower and weight. There also seems to be a positive␣
      ↪correlation. Furthermore, cars with false mpg_highs(lower than
     #average) tend to have higher weirght and horsepower in general
```

```
[ ]: <seaborn.axisgrid.FacetGrid at 0x7fde1cc5eb50>
```




```
[ ]: sb.boxplot(x='mpg_high', y='weight', data=df)
     #This plot compares the variable mpg_high with the weight. We notice that in␣
      ↪general cars with false mpg_highs(below average) tend to weigh more
     #as can be seen from their median and IQR.
```

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



##Train/test split

[ ]:
```
from sklearn.model_selection import train_test_split
X = df.iloc[:, 0:7]
y = df.mpg_high
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=1234)

print('train size:', X_train.shape)
print('test size:', X_test.shape)
```

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

##Logistic Regression

[ ]:
```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(solver='lbfgs', max_iter=500)
clf.fit(X_train, y_train)
clf.score(X_train, y_train)
```

[ ]: 0.9067524115755627

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

[ ]:
```python
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test, pred)
```

[ ]:
```
array([[40, 10],
       [ 1, 27]])
```

[ ]:
```python
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:  0.8589743589743589
precision score:  0.7297297297297297
recall score:  0.9642857142857143
f1 score:  0.8307692307692307
```

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

```
              precision    recall  f1-score   support

         0.0       0.98      0.80      0.88        50
         1.0       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
```

##Decision Tree

[ ]:
```python
from sklearn.tree import DecisionTreeClassifier

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

# make predictions

pred = clf.predict(X_test)
```

[ ]:
```python
# 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:  0.9230769230769231
precision score:  0.84375
recall score:  0.9642857142857143
f1 score:  0.8999999999999999
```

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

confusion_matrix(y_test, pred)
```

[ ]:
```
array([[45,  5],
       [ 1, 27]])
```

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

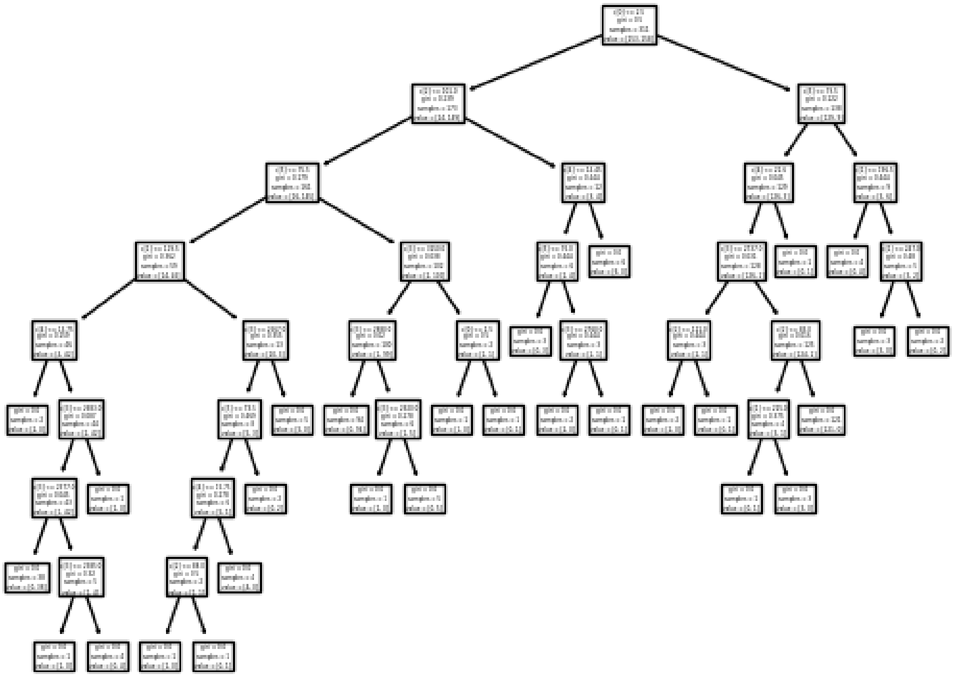
```
              precision    recall  f1-score   support

         0.0       0.98      0.90      0.94        50
         1.0       0.84      0.96      0.90        28

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

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

[ ]:
```
[Text(0.6597222222222222, 0.9444444444444444, 'x[0] <= 2.5\ngini = 0.5\nsamples
= 311\nvalue = [153, 158]'),
 Text(0.4583333333333333, 0.8333333333333334, 'x[2] <= 101.0\ngini =
0.239\nsamples = 173\nvalue = [24, 149]'),
 Text(0.3055555555555556, 0.7222222222222222, 'x[5] <= 75.5\ngini =
0.179\nsamples = 161\nvalue = [16, 145]'),
 Text(0.16666666666666666, 0.6111111111111112, 'x[1] <= 119.5\ngini =
0.362\nsamples = 59\nvalue = [14, 45]'),
 Text(0.05555555555555555, 0.5, 'x[4] <= 13.75\ngini = 0.159\nsamples =
46\nvalue = [4, 42]'),
```

```
 Text(0.027777777777777776, 0.3888888888888889, 'gini = 0.0\nsamples = 2\nvalue
= [2, 0]'),
 Text(0.08333333333333333, 0.3888888888888889, 'x[3] <= 2683.0\ngini =
0.087\nsamples = 44\nvalue = [2, 42]'),
 Text(0.05555555555555555, 0.2777777777777778, 'x[3] <= 2377.0\ngini =
0.045\nsamples = 43\nvalue = [1, 42]'),
 Text(0.027777777777777776, 0.16666666666666666, 'gini = 0.0\nsamples =
38\nvalue = [0, 38]'),
 Text(0.08333333333333333, 0.16666666666666666, 'x[3] <= 2385.0\ngini =
0.32\nsamples = 5\nvalue = [1, 4]'),
 Text(0.05555555555555555, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nvalue
= [1, 0]'),
 Text(0.1111111111111111, 0.05555555555555555, 'gini = 0.0\nsamples = 4\nvalue =
[0, 4]'),
 Text(0.1111111111111111, 0.2777777777777778, 'gini = 0.0\nsamples = 1\nvalue =
[1, 0]'),
 Text(0.2777777777777778, 0.5, 'x[3] <= 2567.0\ngini = 0.355\nsamples =
13\nvalue = [10, 3]'),
 Text(0.25, 0.3888888888888889, 'x[5] <= 73.5\ngini = 0.469\nsamples = 8\nvalue
= [5, 3]'),
 Text(0.2222222222222222, 0.2777777777777778, 'x[4] <= 15.75\ngini =
0.278\nsamples = 6\nvalue = [5, 1]'),
 Text(0.19444444444444445, 0.16666666666666666, 'x[2] <= 88.0\ngini =
0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.16666666666666666, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nvalue
= [1, 0]'),
 Text(0.2222222222222222, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.25, 0.16666666666666666, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]'),
 Text(0.2777777777777778, 0.2777777777777778, 'gini = 0.0\nsamples = 2\nvalue =
[0, 2]'),
 Text(0.3055555555555556, 0.3888888888888889, 'gini = 0.0\nsamples = 5\nvalue =
[5, 0]'),
 Text(0.4444444444444444, 0.6111111111111112, 'x[3] <= 3250.0\ngini =
0.038\nsamples = 102\nvalue = [2, 100]'),
 Text(0.3888888888888889, 0.5, 'x[3] <= 2880.0\ngini = 0.02\nsamples =
100\nvalue = [1, 99]'),
 Text(0.3611111111111111, 0.3888888888888889, 'gini = 0.0\nsamples = 94\nvalue =
[0, 94]'),
 Text(0.4166666666666667, 0.3888888888888889, 'x[3] <= 2920.0\ngini =
0.278\nsamples = 6\nvalue = [1, 5]'),
 Text(0.3888888888888889, 0.2777777777777778, 'gini = 0.0\nsamples = 1\nvalue =
[1, 0]'),
 Text(0.4444444444444444, 0.2777777777777778, 'gini = 0.0\nsamples = 5\nvalue =
[0, 5]'),
 Text(0.5, 0.5, 'x[0] <= 1.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.4722222222222222, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue =
```

```
[1, 0]'),
 Text(0.5277777777777778, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.6111111111111112, 0.7222222222222222, 'x[4] <= 14.45\ngini =
0.444\nsamples = 12\nvalue = [8, 4]'),
 Text(0.5833333333333334, 0.6111111111111112, 'x[5] <= 76.0\ngini =
0.444\nsamples = 6\nvalue = [2, 4]'),
 Text(0.5555555555555556, 0.5, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.6111111111111112, 0.5, 'x[3] <= 2760.0\ngini = 0.444\nsamples = 3\nvalue
= [2, 1]'),
 Text(0.5833333333333334, 0.3888888888888889, 'gini = 0.0\nsamples = 2\nvalue =
[2, 0]'),
 Text(0.6388888888888888, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.6388888888888888, 0.6111111111111112, 'gini = 0.0\nsamples = 6\nvalue =
[6, 0]'),
 Text(0.8611111111111112, 0.8333333333333334, 'x[5] <= 79.5\ngini =
0.122\nsamples = 138\nvalue = [129, 9]'),
 Text(0.8055555555555556, 0.7222222222222222, 'x[4] <= 21.6\ngini =
0.045\nsamples = 129\nvalue = [126, 3]'),
 Text(0.7777777777777778, 0.6111111111111112, 'x[3] <= 2737.0\ngini =
0.031\nsamples = 128\nvalue = [126, 2]'),
 Text(0.7222222222222222, 0.5, 'x[2] <= 111.0\ngini = 0.444\nsamples = 3\nvalue
= [2, 1]'),
 Text(0.6944444444444444, 0.3888888888888889, 'gini = 0.0\nsamples = 2\nvalue =
[2, 0]'),
 Text(0.75, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.8333333333333334, 0.5, 'x[2] <= 83.0\ngini = 0.016\nsamples = 125\nvalue
= [124, 1]'),
 Text(0.8055555555555556, 0.3888888888888889, 'x[1] <= 225.0\ngini =
0.375\nsamples = 4\nvalue = [3, 1]'),
 Text(0.7777777777777778, 0.2777777777777778, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.8333333333333334, 0.2777777777777778, 'gini = 0.0\nsamples = 3\nvalue =
[3, 0]'),
 Text(0.8611111111111112, 0.3888888888888889, 'gini = 0.0\nsamples = 121\nvalue
= [121, 0]'),
 Text(0.8333333333333334, 0.6111111111111112, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.9166666666666666, 0.7222222222222222, 'x[1] <= 196.5\ngini =
0.444\nsamples = 9\nvalue = [3, 6]'),
 Text(0.8888888888888888, 0.6111111111111112, 'gini = 0.0\nsamples = 4\nvalue =
[0, 4]'),
 Text(0.9444444444444444, 0.6111111111111112, 'x[1] <= 247.0\ngini =
0.48\nsamples = 5\nvalue = [3, 2]'),
 Text(0.9166666666666666, 0.5, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.9722222222222222, 0.5, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]')]
```



##Neural Networks

[ ]:
```
# 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)

# train
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(3, 2), max_iter=1000,
 ↪random_state=1234)
clf.fit(X_train_scaled, y_train)

# make predictions

pred = clf.predict(X_test_scaled)

# output results
```

```
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))

confusion_matrix(y_test, pred)

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

```
accuracy score:  0.8717948717948718
precision score:  0.8
recall score:  0.8571428571428571
f1 score:  0.8275862068965518
              precision    recall  f1-score   support

         0.0       0.92      0.88      0.90        50
         1.0       0.80      0.86      0.83        28

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

[ ]:
```
# try different settings

clf = MLPClassifier(solver='sgd', hidden_layer_sizes=(3,), max_iter=1500,␣
 ↪random_state=1234)
clf.fit(X_train_scaled, y_train)

# make predictions
pred = clf.predict(X_test_scaled)

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))

# confusion matrix
confusion_matrix(y_test, pred)

print(classification_report(y_test, pred))
```

```
accuracy score:  0.8333333333333334
precision score:  0.7142857142857143
recall score:  0.8928571428571429
f1 score:  0.7936507936507937
```

	precision	recall	f1-score	support
0.0	0.93	0.80	0.86	50
1.0	0.71	0.89	0.79	28
accuracy			0.83	78
macro avg	0.82	0.85	0.83	78
weighted avg	0.85	0.83	0.84	78

The rules of thumb for hidden layers and nodes is that it should be between 1 and the number of predictors, two thirds of the input layer size plus the size of the output layer and less than twice the input layer size. There could be many reasons why one performed worse than the other. The side of the hidden layer in a neural network can have a significant impact on the accuracy of the model. There may be differences due to overfitting, optimization, and complexity of the problem. The first had an accuracy of 87% and the second had an accuracy of 83%

##Analysis

If we compare logistic regression, decision trees and neural networks. Decision trees ended up being the best at classification. We can see this in the following:

Decision trees had an accuracy score of 92%, a precision score of 84%, a recall score of 96% and an f1 score of 90%. Logistic regression had an accuracy score of 86%, a precision score of 73%, a recall score of 96% and an f1 score of 83%. The better performing neural network model had an accuracy score of 87%, a precision score of 80%, a recall score of 86% and an f1 score of 83%.

I think that the decision trees performed better is because they are better at handling non-linear relationships. Decision trees are also more robust to outliers in comparison to the other two.

I personally prefer to use R for machine learning because I have more experience with it as a programming language in comparison to Python. However, I can understand why most people would prefer python since it was a lot faster. In the future, I plan to improve my python skills and am curious to see if I would end up preferring sklearn.