Import libraries. Output versions.

```
import numpy
import pandas
import seaborn
import sklearn

print("Numpy Version:", numpy.__version__)
print("Pandas Version:", pandas.__version__)
print("Seaborn Version:", seaborn.__version__)
print("skLearn Version:", sklearn.__version__)
```

```
Numpy Version: 1.22.4
Pandas Version: 1.4.4
Seaborn Version: 0.12.2
skLearn Version: 1.2.2
```

Read the Auto data

Use pandas to read the data

```
df = pandas.read_csv('Auto.csv')
```

Output the first few rows

```
df.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	orig
0	18.0	8	307.0	130	3504	12.0	70.0	
1	15.0	8	350.0	165	3693	11.5	70.0	

Output the dimensions of the data

```
df.shape
```

```
(392, 9)
```

Data exploration with code

Use `describe()` on the mpg, weight, and year columns

```
df.mpg.describe()
```

```
count    392.000000
mean      23.445918
std        7.805007
min        9.000000
25%       17.000000
50%       22.750000
75%       29.000000
max       46.600000
Name: mpg, dtype: float64
```

```
df.weight.describe()
```

```
count     392.000000
mean     2977.584184
std       849.402560
min      1613.000000
25%      2225.250000
50%      2803.500000
75%      3614.750000
max      5140.000000
Name: weight, dtype: float64
```

```
df.year.describe()
```

```
count    390.000000
mean      76.010256
std        3.668093
min       70.000000
25%       73.000000
50%       76.000000
75%       79.000000
max       82.000000
Name: year, dtype: float64
```

write comments indicating the range and average of each column

Name	Average	Range
mpg	23.445918	37.6
cylinders	5.471939	5.0
displacement	194.411990	387.00
horsepower	104.469388	184.00
weight	2977.584184	3527.00
acceleration	15.554220	16.8
year	76.010256	12.0
origin	1.576531	2.0

Explore data types

check the data types of all columns

```
df.dtypes
```

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

change the cylinders column to categorical (use cat.codes)

```
df.cylinders = df.cylinders.astype("category")
```

change the origin column to categorical (don’t use cat.codes)

```
df.origin = df.origin.astype("category")
```

verify the changes with the dtypes attribute

```
df.dtypes
```

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

Deal with NAs

delete rows with NAs

```
df = df.dropna()
```

output the new dimensions

```
df.shape
```

```
(389, 9)
```

Modify columns

make a new column, mpg_high, and make it categorical: the column == 1 if mpg > average mpg, else == 0

```
mpg_high = (df.mpg > numpy.mean(df.mpg))
df["mpg_high"] = mpg_high.astype("int64").astype("category")
```

```
<ipython-input-118-4bd2c568be9b>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-
  df["mpg_high"] = mpg_high.astype("int64").astype("category")
```

delete the mpg and name columns (delete mpg so the algorithm doesn’t just learn to predict mpg_high from mpg)

```
df = df.drop(columns = ["mpg", "name"])
```

output the first few rows of the modified data frame

```
df.head()
```

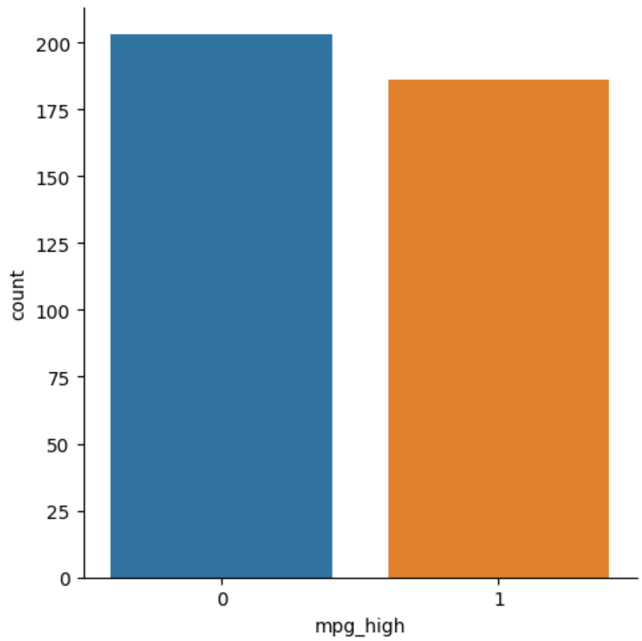
	cylinders	displacement	horsepower	weight	acceleration	year	origin	m
0	8	307.0	130	3504	12.0	70.0	1	
1	8	350.0	165	3693	11.5	70.0	1	
2	8	318.0	150	3436	11.0	70.0	1	

Data exploration with graphs

seaborn catplot on the mpg_high column

```
seaborn.catplot(x="mpg_high", kind="count", data=df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f5d0213a820>
```



seaborn relplot with horsepower on the x axis, weight on the y axis, setting hue or style to mpg_high

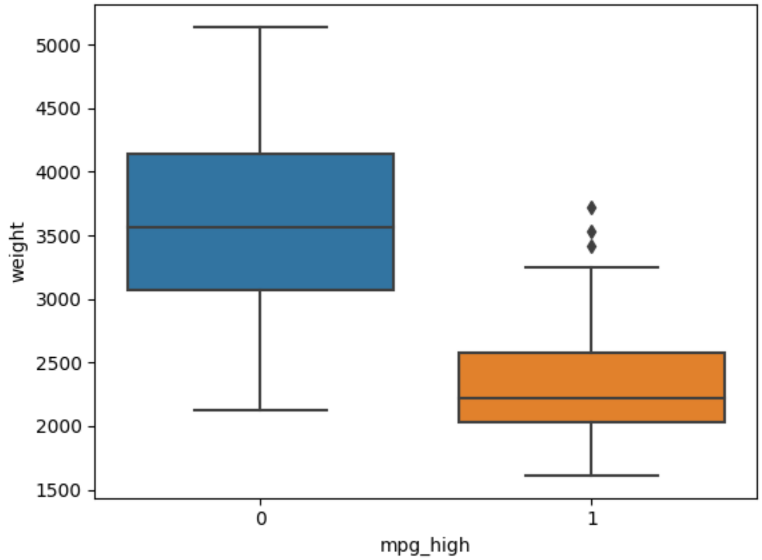
```
seaborn.relplot(x="horsepower", y="weight", data=df, hue=df.mpg_high)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f5d02587b20>
```

seaborn boxplot with mpg_high on the x axis and weight on the y axis

```
seaborn.boxplot(x="mpg_high", y = "weight", data=df)
```

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



for each graph, write a comment indicating one thing you learned about the data from the graph

Catplot

More cars have a below average mpg than have an above average mpg. This means that the median mpg is below the average mpg.

Relplot

Above average mpg correlates with lower horsepower

Boxplot

Above average mpg correlates with lower weight

Train/test split

X contains all remaining columns except `mpg_high`

```
X = df.drop(columns="mpg_high")
y = df.mpg_high
```

80/20, seed 1234

```
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=1234)
```

output the dimensions of train and test

```
print("Dimensions:")
print("X train:", X_train.shape)
```

```
print("X test:", X_test.shape)
print("y train:", y_train.shape)
print("y test:", y_test.shape)
```

```
Dimensions:
X train: (311, 7)
X test: (78, 7)
y train: (311,)
y test: (78,)
```

Logistic Regression

train a logistic regression model using solver lbfgs

```
from sklearn.linear_model import LogisticRegression
logistic_regression_model = LogisticRegression(solver="lbfgs", max_iter=9999999999999)
logistic_regression_model.fit(X_train, y_train)
```

```
▾          LogisticRegression
LogisticRegression(max_iter=9999999999999)
```

test and evaluate

```
logistic_regression_predictions = logistic_regression_model.predict(X_test)

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print('accuracy score: ', accuracy_score(y_test, logistic_regression_predictions))
print('precision score: ', precision_score(y_test, logistic_regression_predictions))
print('recall score: ', recall_score(y_test, logistic_regression_predictions))
print('f1 score: ', f1_score(y_test, logistic_regression_predictions))
```

```
accuracy score:  0.8717948717948718
precision score:  0.75
recall score:  0.9642857142857143
f1 score:  0.8437499999999999
```

print metrics using the classification report

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

```
              precision    recall  f1-score   support

           0       0.98      0.82      0.89        50
           1       0.75      0.96      0.84        28

    accuracy                           0.87        78
   macro avg       0.86      0.89      0.87        78
weighted avg       0.89      0.87      0.87        78
```

Decision Tree

train a decision tree

```
from sklearn.tree import DecisionTreeClassifier

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

▾ DecisionTreeClassifier

test and evaluate

```
decision_tree_predictions = decision_tree_model.predict(X_test)

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print('accuracy score: ', accuracy_score(y_test, decision_tree_predictions))
print('precision score: ', precision_score(y_test, decision_tree_predictions))
print('recall score: ', recall_score(y_test, decision_tree_predictions))
print('f1 score: ', f1_score(y_test, decision_tree_predictions))
```

```
accuracy score:  0.9102564102564102
precision score:  0.8620689655172413
recall score:  0.8928571428571429
f1 score:  0.8771929824561403
```

print the classification report metrics

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

```
              precision    recall  f1-score   support

           0       0.94      0.92      0.93        50
           1       0.86      0.89      0.88        28

    accuracy                           0.91        78
   macro avg       0.90      0.91      0.90        78
weighted avg       0.91      0.91      0.91        78
```

Plot the tree

```
from sklearn import tree
tree.plot_tree(decision_tree_model)
```

```
 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[1] <= 131.0\ngini =
0.278\nsamples = 6\nvalue = [5, 1]'),
 Text(0.20588235294117646, 0.16666666666666666, 'gini = 0.0\nsamples =
4\nvalue = [4, 0]'),
 Text(0.2647058823529412, 0.16666666666666666, 'x[5] <= 73.0\ngini =
0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.23529411764705882, 0.05555555555555555, 'gini = 0.0\nsamples =
1\nvalue = [0, 1]'),
 Text(0.29411764705882354, 0.05555555555555555, 'gini = 0.0\nsamples =
1\nvalue = [1, 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(0.38235294117647056, 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]'),
```

Neural Network

Scale 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 a neural network, choosing a network topology of your choice

```
from sklearn.neural_network import MLPClassifier

neural_network_1 = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5, 3), max_iter=500, random_state=1234)
neural_network_1.fit(X_train_scaled, y_train)
```

```
▾                          MLPClassifier
MLPClassifier(hidden_layer_sizes=(5, 3), max_iter=500, random_state=1234,
              solver='lbfgs')
```

test and evaluate

```
neural_network_1_predictions = neural_network_1.predict(X_test_scaled)

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print('accuracy score: ', accuracy_score(y_test, neural_network_1_predictions))
```

```
print('precision score: ', precision_score(y_test, neural_network_1_predictions))
print('recall score: ', recall_score(y_test, neural_network_1_predictions))
print('f1 score: ', f1_score(y_test, neural_network_1_predictions))

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

```
accuracy score:  0.8974358974358975
precision score:  0.7941176470588235
recall score:  0.9642857142857143
f1 score:  0.8709677419354839
              precision    recall  f1-score   support

           0       0.98      0.82      0.89        50
           1       0.75      0.96      0.84        28

    accuracy                           0.87        78
   macro avg       0.86      0.89      0.87        78
weighted avg       0.89      0.87      0.87        78
```

train a second network with a different topology and different settings

```
from sklearn.neural_network import MLPClassifier

neural_network_2 = MLPClassifier(solver='sgd', hidden_layer_sizes=(4,), max_iter=1500, random_state=1234)
neural_network_2.fit(X_train_scaled, y_train)
```

```
MLPClassifier
MLPClassifier(hidden_layer_sizes=(4,), max_iter=1500, random_state=1234,
              solver='sgd')
```

test and evaluate

```
neural_network_2_predictions = neural_network_2.predict(X_test_scaled)

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print('accuracy score: ', accuracy_score(y_test, neural_network_2_predictions))
print('precision score: ', precision_score(y_test, neural_network_2_predictions))
print('recall score: ', recall_score(y_test, neural_network_2_predictions))
print('f1 score: ', f1_score(y_test, neural_network_2_predictions))

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

```
accuracy score:  0.8461538461538461
precision score:  0.7352941176470589
recall score:  0.8928571428571429
f1 score:  0.806451612903226
              precision    recall  f1-score   support

           0       0.98      0.82      0.89        50
           1       0.75      0.96      0.84        28

    accuracy                           0.87        78
   macro avg       0.86      0.89      0.87        78
weighted avg       0.89      0.87      0.87        78
```

compare the two models and why you think the performance was same/different

The first neural network used the `lbfgs` solver and hidden layers of sizes 5 and 3. It had an accuracy of 0.90.

The second neural network used the `sgd` solver and has a hidden layer of size 4. It had an accuracy of 0.85.

The performance of the neural networks were quite similar but the first one was a little bit better. This probably occurred because the first neural network had an additional hidden layer, which allows it to better model complex relationships between the inputs and output.

Analysis

Which algorithm performed better?

Decision Tree had an accuracy of 0.91. Neural Networks had an accuracy of 0.90. Logistic Regression had an accuracy of 0.87. Decision Trees performed the best.

compare accuracy, recall and precision metrics by class

The precision for not being `mpg_high` is around 0.98 and the precision for being `mpg_high` is around 0.75. The recall for not being `mpg_high` is around 0.82 and the recall for being `mpg_high` is around 0.96.

give your analysis of why the better-performing algorithm might have outperformed the other

Logistic regression could have performed the worst because it is a high bias algorithm. The nature of logistic regression prevents it from fitting the data as well as neural networks or decision trees. Neural networks require a larger amount of configuration to receive optimal results when compared to the configuration required for decision trees. This is why when a traditional machine learning algorithm suffices, neural networks are not recommended. Decision Trees probably slightly outperformed neural networks because of suboptimal configuration choices that are hard to identify.

write a couple of sentences comparing your experiences using R versus sklearn. Feel free to express strong preferences.

I am much more familiar with Python than R. I like that there is a single package providing common machine learning functions in Python, because I like having similar syntax. Overall, I enjoy machine learning with Python more than machine learning with R.

✓ 0s completed at 10:29 PM