In [12]:

```
#Integer, floats and some mathmatical operation
x=3
print(type(x))
print(x)
print(x+1)
print(x-1)
print(x-1)
print(x*2)
print(x**2)
x+=1
print(x)
x *=2
print(x)
y=2.5
print(type(y))
```

```
<class 'int'>
3
4
2
2
6
9
4
8
<class 'float'>
2.5 3.5 5.0 6.25
```

In [11]:

```
#Bolleans
t=True
f=False
print(type(t))
print(t and f)
print(t or f)
print(not t)
```

```
<class 'bool'>
False
True
False
True
```

In [15]:

```
# Strings
hello='hello'
world='world'
print(hello)
print(len(hello))
hw=hello+''+world
print(hw)
hw12='%s %s %d'%(hello,world,12)
```

```
hello
```

In [38]:

```python
# Useful methods for strings
s='hello'
print(s.capitalize())
print(s.upper())
print(s.rjust(7))
print(s.center(10))
print(s.replace('h','all'))
```

```
Hello
HELLO
  hello
  hello
allello
world
```

In [45]:

```python
# List
xs = [3, 1, 2]
print(xs, xs[2])
print(xs[-1])
xs[2] = 'foo'
print(xs)
xs.append('bar')
print(xs)
x = xs.pop()
```

```
[3, 1, 2] 2
2
[3, 1, 'foo']
[3, 1, 'foo', 'bar']
bar [3, 1, 'foo']
```

In [47]:

```python
#Slicing
nums = list(range(5))
print(nums)
print(nums[2:4])
print(nums[2:])
print(nums[:2])
print(nums[:])
print(nums[:-1])
nums[2:4] = [8, 9]
print(nums)
```

```
[0, 1, 2, 3, 4]
[2, 3]
[2, 3, 4]
[0, 1]
[0, 1, 2, 3, 4]
[0, 1, 2, 3]
[0, 1, 8, 9, 4]
```

In [48]:

```python
#loop1
animals = ['cat', 'dog', 'monkey']
for animal in animals:
 print(animal)
```

```
cat
dog
monkey
```

In [49]:

```
#loop2
animals = ['cat', 'dog', 'monkey']
for idx, animal in enumerate(animals):
```

```
#1: cat
#2: dog
#3: monkey
```

In [50]:

```
#loop3
nums = [0, 1, 2, 3, 4]
squares = []
for x in nums:
 squares.append(x ** 2)
```

```
[0, 1, 4, 9, 16]
```

In [51]:

```
#loop4
nums = [0, 1, 2, 3, 4]
squares = [x ** 2 for x in nums]
```

```
[0, 1, 4, 9, 16]
```

In [58]:

```
#loop5
nums = [0, 1, 2, 3, 4]
even_squares = [x ** 2 for x in nums if x % 2 == 0]
print(even_squares)
```

```
[0, 4, 16]
```

In [57]:

```
#Dictionaries
d = {'cat': 'cute', 'dog': 'furry'}
print(d['cat'])
print('cat' in d)
d['fish'] = 'wet'
print(d['fish'])
print(d.get('monkey', 'N/A'))
print(d.get('fish', 'N/A'))
del d['fish']
```

```
cute
True
wet
N/A
wet
N/A
```

In [59]:

```
# Iterate keys in a dictonary
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal in d:
 legs = d[animal]
 print('A %s has %d legs' % (animal, legs))
```

```
A person has 2 legs
A cat has 4 legs
A spider has 8 legs
```

In [60]:

```
# 'items' method to access keys and corresponding values
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal, legs in d.items():
```

```
A person has 2 legs
A cat has 4 legs
A spider has 8 legs
```

In [61]:

```
#similar to list comprehensions, but allow you to easily construct dictio
nums = [0, 1, 2, 3, 4]
even_num_to_square = {x: x ** 2 for x in nums if x % 2 == 0}
```

```
{0: 0, 2: 4, 4: 16}
```

In [63]:

```
#Set
animals = {'cat', 'dog'}
print('cat' in animals)
print('fish' in animals)
animals.add('fish')
print('fish' in animals)
print(len(animals))
animals.add('cat')
print(len(animals))
animals.remove('cat')
```

```
True
False
True
3
3
2
```

In [64]:

```
#Iterating over a set
animals = {'cat', 'dog', 'fish'}
for idx, animal in enumerate(animals):
 print('#%d: %s' % (idx + 1, animal))
```

```
#1: fish
#2: cat
#3: dog
```

In [65]:

```
#construct sets using set comprehensions:
from math import sqrt
nums = {int(sqrt(x)) for x in range(30)}
```

```
{0, 1, 2, 3, 4, 5}
```

In [66]:

```
#Tples
d = {(x, x + 1): x for x in range(10)}
```

```
t = (5, 6)
print(type(t))
print(d[t])
```

```
<class 'tuple'>
5
1
```

In [67]:
```
#Numpy array
import numpy as np
a = np.array([1, 2, 3])
print(type(a))
print(a.shape)
print(a[0], a[1], a[2])
a[0] = 5
print(a)
b = np.array([[1,2,3],[4,5,6]])
print(b.shape)
print(b[0, 0], b[0, 1], b[1, 0])
```

```
<class 'numpy.ndarray'>
(3,)
1 2 3
[5 2 3]
(2, 3)
1 2 4
```

In [68]:
```
#Functions for Numpy array
import numpy as np
a = np.zeros((2,2))
print(a)
b = np.ones((1,2))
print(b)
c = np.full((2,2), 7)
print(c)

d = np.eye(2)
print(d)

e = np.random.random((2,2))
```

```
[[0. 0.]
 [0. 0.]]
[[1. 1.]]
[[7 7]
 [7 7]]
[[1. 0.]
 [0. 1.]]
[[0.9966884  0.03442652]
 [0.5654238  0.17367813]]
```

In [77]:
```
#Array indexing
import numpy as np
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
print(a)
#arrary slicing
```

```
b = a[:2, 1:3]
print(b)
print(a[0, 1])
b[0, 0] = 77
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
[[2 3]
 [6 7]]
2
77
```

In [84]:

```
##arrary slicing
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
print(a)
row_r1 = a[1, :]
print(row_r1)
row_r2 = a[1:2, :]
print(row_r2)
print(row_r1, row_r1.shape)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
[5 6 7 8]
[[5 6 7 8]]
[5 6 7 8] (4,)
[[5 6 7 8]] (1, 4)
```

In [85]:

```
#accessing columns of an array:
col_r1 = a[:, 1]
col_r2 = a[:, 1:2]
print(col_r1, col_r1.shape)
```

```
[ 2  6 10] (3,)
[[ 2]
 [ 6]
 [10]] (3, 1)
```

In [90]:

```
import numpy as np
a = np.array([[1,2], [3, 4], [5, 6]])
print(a)
print(a[[0, 1, 2], [0, 1, 0]])
print(np.array([a[0, 0], a[1, 1], a[2, 0]]))
print(a[[0, 0], [1, 1]])
```

```
[[1 2]
 [3 4]
 [5 6]]
[1 4 5]
[1 4 5]
[2 2]
[2 2]
```

In [93]:

```
# Create a new array from which we will select elements
```

```
a = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
print(a)
b = np.array([0, 2, 0, 1])
print(a[np.arange(4), b])
a[np.arange(4), b] += 10
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
[ 1  6  7 11]
[[11  2  3]
 [ 4  5 16]
 [17  8  9]
 [10 21 12]]
```

In [1]:

```
import numpy as np
a = np.array([[1,2], [3, 4], [5, 6]])
bool_idx = (a > 2)
```

```
[[False False]
 [ True  True]
 [ True  True]]
```

In [3]:

```
print(a[bool_idx])
```

```
[3 4 5 6]
[3 4 5 6]
```

In [4]:

```
import numpy as np
x = np.array([1, 2])
print(x.dtype)
x = np.array([1.0, 2.0])
print(x.dtype)
x = np.array([1, 2], dtype=np.int64)
```

```
int32
float64
int64
```

In [13]:

```
import numpy as np
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)
print(x + y)
print(np.add(x, y))
print(x - y)
print(np.subtract(x, y))
print(x * y)
print(np.multiply(x, y))
print(x / y)
print(np.divide(x, y))
```

```
[[ 6.  8.]
 [10. 12.]]
[[ 6.  8.]
 [10. 12.]]
[[-4. -4.]
 [-4. -4.]]
[[-4. -4.]
 [-4. -4.]]
[[ 5. 12.]
 [21. 32.]]
[[ 5. 12.]
 [21. 32.]]
[[0.2        0.33333333]
```

In [15]:

```
import numpy as np
x = np.array([[1,2],[3,4]])
y = np.array([[5,6],[7,8]])
v = np.array([9,10])
w = np.array([11, 12])
print(v.dot(w))
print(np.dot(v, w))
print(x.dot(v))
print(np.dot(x, v))
print(x.dot(y))
```

```
219
219
[29 67]
[29 67]
[[19 22]
 [43 50]]
[[19 22]
 [43 50]]
```

In [16]:

```
import numpy as np
x = np.array([[1,2],[3,4]])
print(np.sum(x))
print(np.sum(x, axis=0))
```

```
10
[4 6]
[3 7]
```

In [17]:

```
import numpy as np
x = np.array([[1,2], [3,4]])
print(x)
print(x.T)
v = np.array([1,2,3])
print(v)
```

```
[[1 2]
 [3 4]]
[[1 3]
 [2 4]]
[1 2 3]
[1 2 3]
```

In [18]:

```
import numpy as np

x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = np.empty_like(x)
for i in range(4):
 y[i, :] = x[i, :] + v
```

```
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```

In [19]:

```
import numpy as np
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
vv = np.tile(v, (4, 1))
print(vv)
y = x + vv
```

```
[[1 0 1]
 [1 0 1]
 [1 0 1]
 [1 0 1]]
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```

In [21]:

```
import numpy as np
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = x + v
```

```
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```

In [23]:

```
import numpy as np

v = np.array([1,2,3])
w = np.array([4,5])
print(np.reshape(v, (3, 1)) * w)
x = np.array([[1,2,3], [4,5,6]])
print(x + v)
print((x.T + w).T)
print(x + np.reshape(w, (2, 1)))
print(x * 2)
```

```
[[ 4  5]
 [ 8 10]
 [12 15]]
[[2 4 6]
 [5 7 9]]
[[ 5  6  7]
```

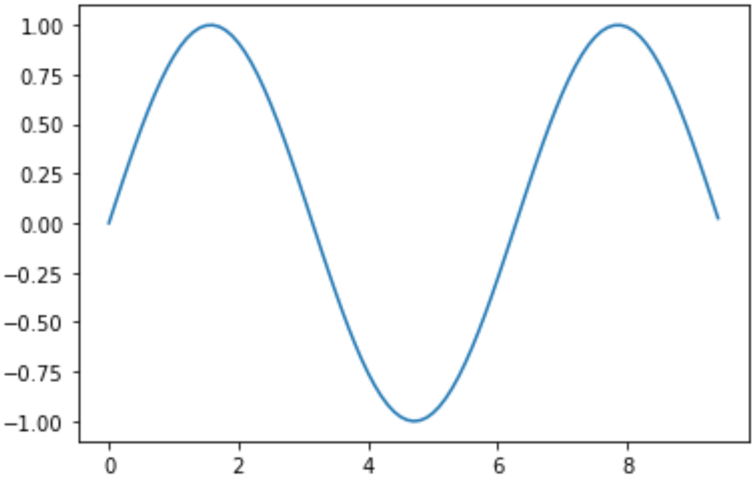
In [25]:

```
#SciPy
import numpy as np
from scipy.spatial.distance import pdist, squareform
x = np.array([[0, 1], [1, 0], [2, 0]])
print(x)
d = squareform(pdist(x, 'euclidean'))
print(d)
```

```
[[0 1]
 [1 0]
 [2 0]]
[[0.         1.41421356 2.23606798]
 [1.41421356 0.         1.        ]
 [2.23606798 1.         0.        ]]
```

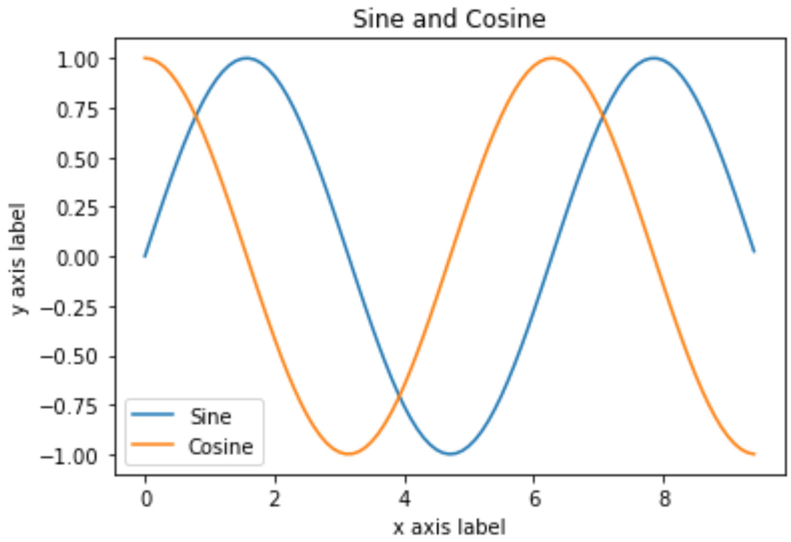
In [26]:

```
#Matplotlib
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)
plt.plot(x, y)
plt.show()
```



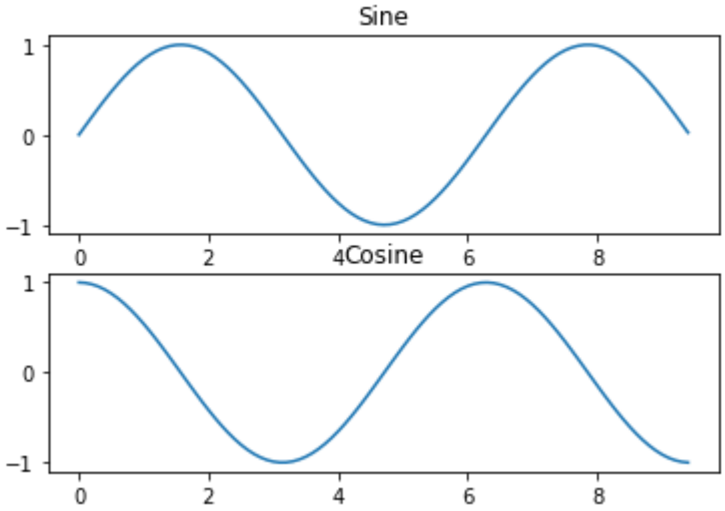
In [28]:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)
plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
plt.show()
```



In [29]:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

plt.subplot(2, 1, 1)

plt.plot(x, y_sin)
plt.title('Sine')

plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

plt.show()
```



In [44]:

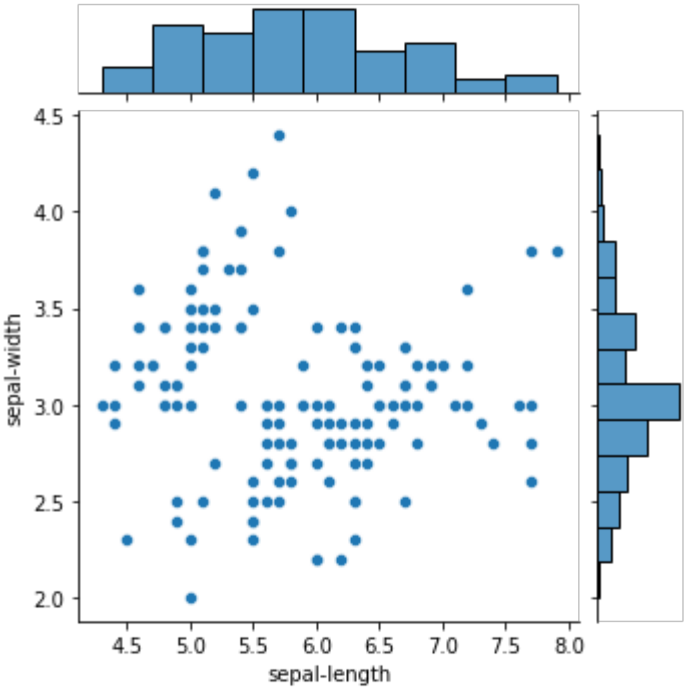
```
# summarize the data
from pandas import read_csv
# Load dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.c
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width',
'class']
dataset = read_csv(url, names=names)
print(dataset.shape)
# head
print(dataset.head(20))
# descriptions
print(dataset.describe())
# class distribution
print(dataset.groupby('class').size())
```

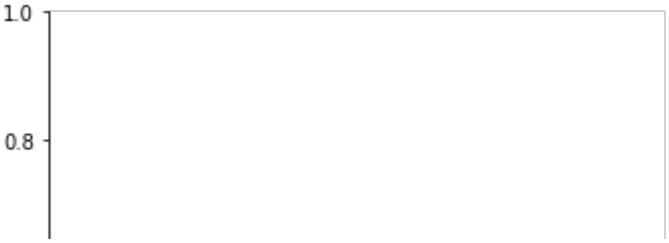
```
(150, 5)
    sepal-length  sepal-width  petal-length  petal-width        class
0            5.1          3.5           1.4          0.2  Iris-setosa
1            4.9          3.0           1.4          0.2  Iris-setosa
2            4.7          3.2           1.3          0.2  Iris-setosa
3            4.6          3.1           1.5          0.2  Iris-setosa
4            5.0          3.6           1.4          0.2  Iris-setosa
5            5.4          3.9           1.7          0.4  Iris-setosa
6            4.6          3.4           1.4          0.3  Iris-setosa
7            5.0          3.4           1.5          0.2  Iris-setosa
8            4.4          2.9           1.4          0.2  Iris-setosa
9            4.9          3.1           1.5          0.1  Iris-setosa
10           5.4          3.7           1.5          0.2  Iris-setosa
11           4.8          3.4           1.6          0.2  Iris-setosa
12           4.8          3.0           1.4          0.1  Iris-setosa
13           4.3          3.0           1.1          0.1  Iris-setosa
14           5.8          4.0           1.2          0.2  Iris-setosa
15           5.7          4.4           1.5          0.4  Iris-setosa
16           5.4          3.9           1.3          0.4  Iris-setosa
17           5.1          3.5           1.4          0.3  Iris-setosa
18           5.7          3.8           1.7          0.3  Iris-setosa
19           5.1          3.8           1.5          0.3  Iris-setosa
       sepal-length  sepal-width  petal-length  petal-width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.054000      3.758667     1.198667
std        0.828066     0.433594      1.764420     0.763161
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
```

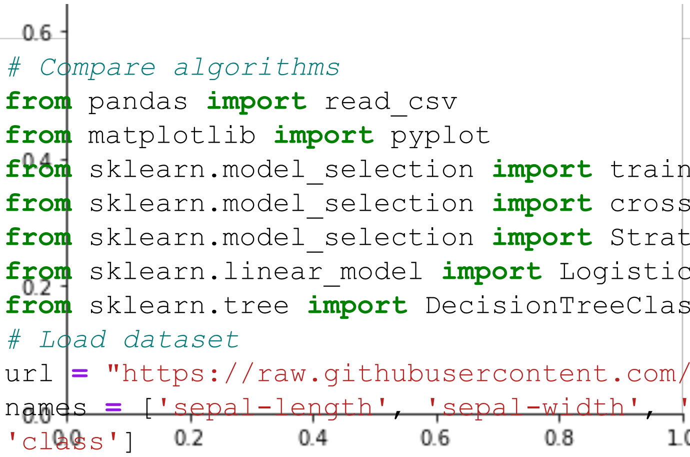
In [49]:

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.jointplot(x='sepal-length',y='sepal-width',data=dataset,height=5)
sns.FacetGrid(dataset,hue='class',height=5).map(plt.scatter,'sepal•length
plt.show()
```

```
-----------------------------------------------------------------------
-----
KeyError                                  Traceback (most recent call
last)
C:\Users\TASLIM~1\AppData\Local\Temp/ipykernel_3324/2287344929.py in
<module>
      2 import matplotlib.pyplot as plt
      3 sns.jointplot(x='sepal-length',y='sepal-width',data=dataset,he
ight=5)
----> 4 sns.FacetGrid(dataset,hue='class',height=5).map(plt.scatter,'s
epallength','sepal-width').add_legend()
      5 plt.show()

~\anaconda3\lib\site-packages\seaborn\axisgrid.py in map(self, func, *
args, **kwargs)
    698
    699             # Get the actual data we are going to plot with
--> 700             plot_data = data_ijk[list(args)]
    701             if self._dropna:
    702                 plot_data = plot_data.dropna()

~\anaconda3\lib\site-packages\pandas\core\frame.py in __getitem__(sel
f, key)
   3462             if is_iterator(key):
   3463                 key = list(key)
-> 3464             indexer = self.loc._get_listlike_indexer(key, axis
=1)[1]
   3465
```



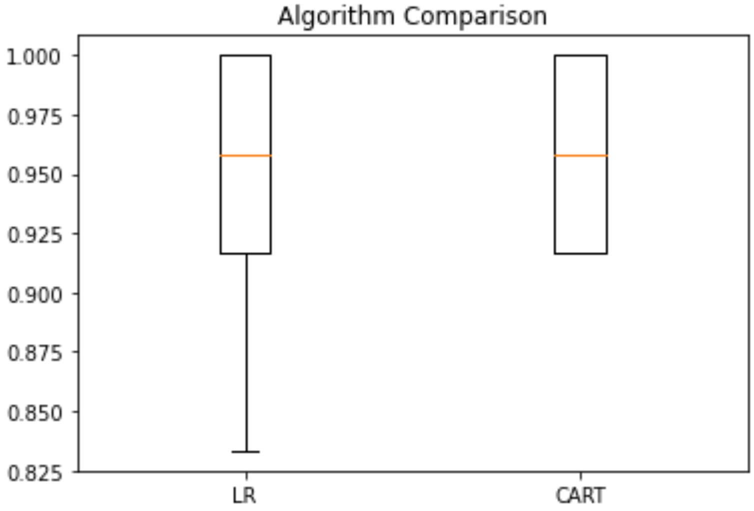




In [50]:

```
# Compare algorithms
from pandas import read_csv
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
# Load dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.c
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width',
'class']
dataset = read_csv(url, names=names)
# Split-out validation dataset
array = dataset.values
X = array[:,0:4]
y = array[:,4]
X_train, X_validation, Y_train, Y_validation = train_test_split(X, y,
test_size=0.20, random_state=1, shuffle=True)
# Spot Check Algorithms
models = []
models.append(('LR', LogisticRegression(solver='liblinear',
multi_class='ovr')))
models.append(('CART', DecisionTreeClassifier()))
# Evaluate each model in turn
results = []
names = []
for name, model in models:
    kfold = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold,scorin
    results.append(cv_results)
    names.append(name)
print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))
# Compare Algorithms
pyplot.boxplot(results, labels=names)
pyplot.title('Algorithm Comparison')
```

```
CART: 0.958333 (0.041667)
```



In [ ]: