In [1]:

```python
#Basic Data Types
x = 3
print(type(x)) # Prints "<class 'int'>"
print(x)       # Prints "3"
print(x + 1)   # Addition; prints "4"
print(x - 1)   # Subtraction; prints "2"
print(x * 2)   # Multiplication; prints "6"
print(x ** 2)  # Exponentiation; prints "9"
x += 1
print(x)  # Prints "4"
x *= 2
print(x)  # Prints "8"
y = 2.5
print(type(y)) # Prints "<class 'float'>"
print(y, y + 1, y * 2, y ** 2) # Prints "2.5 3.5 5.0 6.25"
```

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

In [2]:

```python
#Logical Operator
t = True
f = False
print(type(t)) # Prints "<class 'bool'>"
print(t and f) # Logical AND; prints "False"
print(t or f)  # Logical OR; prints "True"
print(not t)   # Logical NOT; prints "False"
print(t != f)  # Logical XOR; prints "True"
```

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

In [4]:

```python
#String Operations
hello = 'hello'    # String literals can use single quotes
world = "world"    # or double quotes; it does not matter.
print(hello)       # Prints "hello"
print(len(hello))  # String length; prints "5"
hw = hello + ' ' + world  # String concatenation
print(hw)  # prints "hello world"
hw12 = '%s %s %d' % (hello, world, 12)  # sprintf style string formatting
print(hw12)  # prints "hello world 12"

s = "hello"
print(s.capitalize())  # Capitalize a string; prints "Hello"
print(s.upper())       # Convert a string to uppercase; prints "HELLO"
print(s.rjust(7))      # Right-justify a string, padding with spaces; prints "  hello"
print(s.center(7))     # Center a string, padding with spaces; prints " hello "
print(s.replace('l', '(ell)'))  # Replace all instances of one substring with another;
                                # prints "he(ell)(ell)o"
print('  world '.strip())  # Strip leading and trailing whitespace; prints "world"
```

```
hello
5
hello world
hello world 12
Hello
HELLO
  hello
 hello
he(ell)(ell)o
world
```

In [5]:

```python
#List
xs = [3, 1, 2]    # Create a list
print(xs, xs[2])  # Prints "[3, 1, 2] 2"
print(xs[-1])     # Negative indices count from the end of the list; prints "2"
xs[2] = 'foo'     # Lists can contain elements of different types
print(xs)         # Prints "[3, 1, 'foo']"
xs.append('bar')  # Add a new element to the end of the list
print(xs)         # Prints "[3, 1, 'foo', 'bar']"
x = xs.pop()      # Remove and return the last element of the list
print(x, xs)      # Prints "bar [3, 1, 'foo']"
```

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

In [6]:

```python
#slicing
nums = list(range(5))     # range is a built-in function that creates a list of integers
print(nums)               # Prints "[0, 1, 2, 3, 4]"
print(nums[2:4])          # Get a slice from index 2 to 4 (exclusive); prints "[2, 3]"
print(nums[2:])           # Get a slice from index 2 to the end; prints "[2, 3, 4]"
print(nums[:2])           # Get a slice from the start to index 2 (exclusive); prints "[0, 1]"
print(nums[:])            # Get a slice of the whole list; prints "[0, 1, 2, 3, 4]"
print(nums[:-1])          # Slice indices can be negative; prints "[0, 1, 2, 3]"
nums[2:4] = [8, 9]        # Assign a new sublist to a slice
print(nums)               # Prints "[0, 1, 8, 9, 4]"
```

```
[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 [7]:

```python
#Loop
#1.For Loop
animals = ['cat', 'dog', 'monkey']
for animal in animals:
    print(animal)
#2
for idx, animal in enumerate(animals):
    print('#%d: %s' % (idx + 1, animal))
```

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

In [9]:

```python
#List Comprehension
#Normal Method
nums = [0, 1, 2, 3, 4]
squares = []
for x in nums:
    squares.append(x ** 2)
print(squares)   # Prints [0, 1, 4, 9, 16]

#Easy Shorten method
square = [x ** 2 for x in nums]
print(square)   # Prints [0, 1, 4, 9, 16]

#Another Method
nums = [0, 1, 2, 3, 4]
even_squares = [x ** 2 for x in nums if x % 2 == 0]
print(even_squares)  # Prints "[0, 4, 16]"
```

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

In [10]:

```python
#Array using Numpy
#Basic
import numpy as np

a = np.array([1, 2, 3])   # Create a rank 1 array
print(type(a))            # Prints "<class 'numpy.ndarray'>"
print(a.shape)            # Prints "(3,)"
print(a[0], a[1], a[2])   # Prints "1 2 3"
a[0] = 5                  # Change an element of the array
print(a)                  # Prints "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array
print(b.shape)                     # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"
```

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

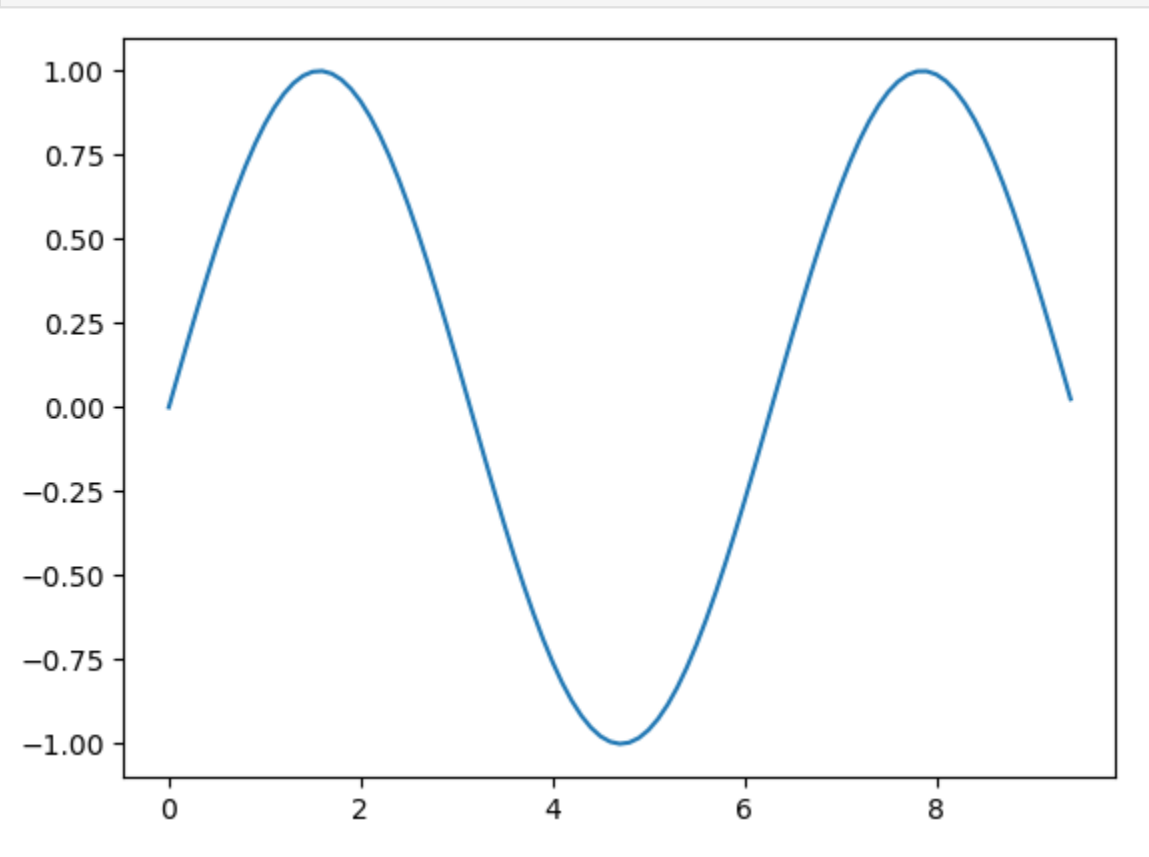
In [11]:

```python
#Array with Built-in Functions

import numpy as np

a = np.zeros((2,2))   # Create an array of all zeros
print(a)              # Prints "[[ 0.  0.]
                      #          [ 0.  0.]]"

b = np.ones((1,2))    # Create an array of all ones
print(b)              # Prints "[[ 1.  1.]]"

c = np.full((2,2), 7)  # Create a constant array
print(c)               # Prints "[[ 7.  7.]
                       #          [ 7.  7.]]"

d = np.eye(2)         # Create a 2x2 identity matrix
print(d)              # Prints "[[ 1.  0.]
                      #          [ 0.  1.]]"

e = np.random.random((2,2))  # Create an array filled with random values
print(e)                     # Might print "[[ 0.91940167  0.08143941]
                             #               [ 0.68744134  0.87236687]]"
```

```
[[0. 0.]
 [0. 0.]]
[[1. 1.]]
[[7 7]
 [7 7]]
[[1. 0.]
 [0. 1.]]
[[0.52696678 0.46687876]
 [0.21859556 0.77106362]]
```

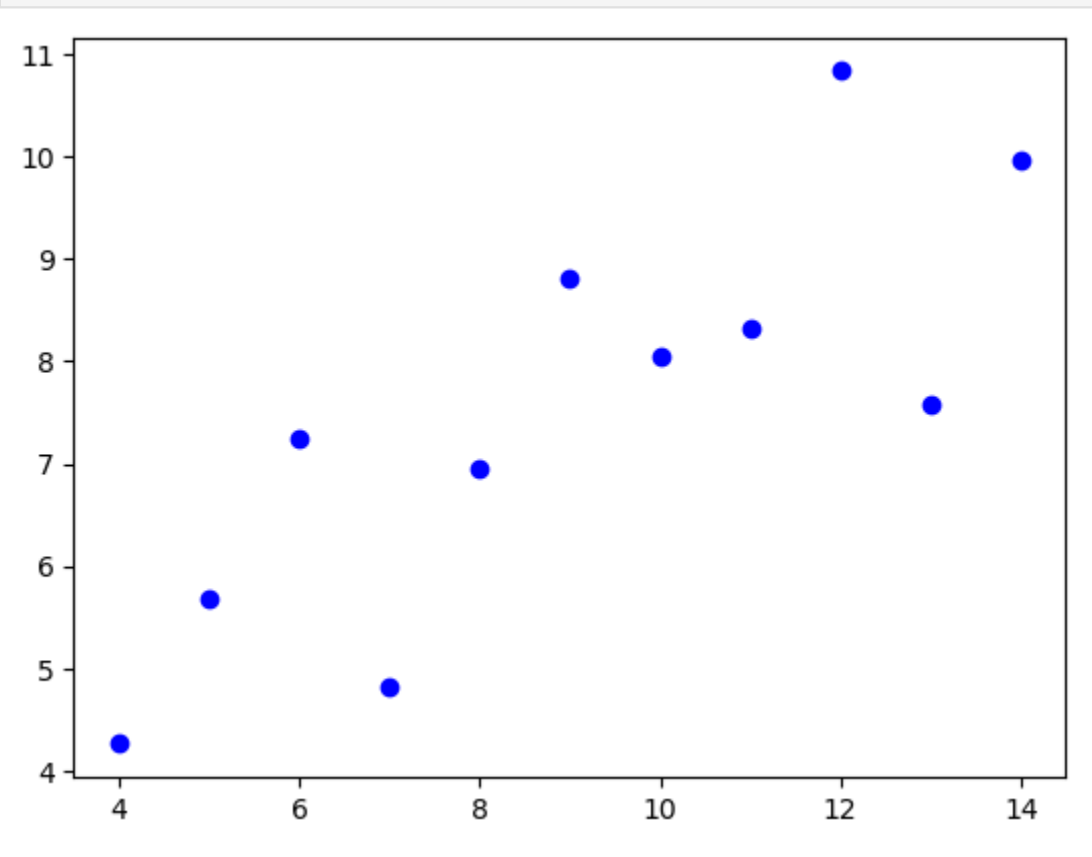
In [12]:

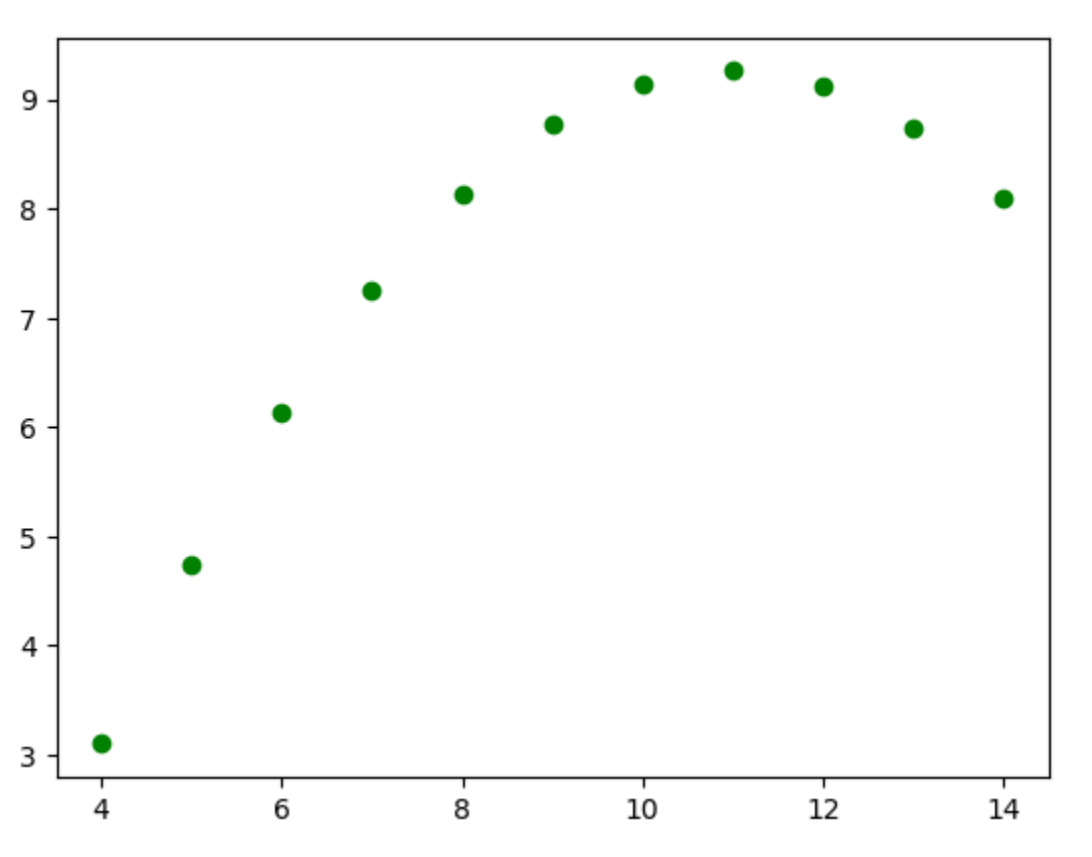
```python
#Using Matlab
#Plotting
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

# Plot the points using matplotlib
plt.plot(x, y)
plt.show()  # You must call plt.show() to make graphics appear.
```

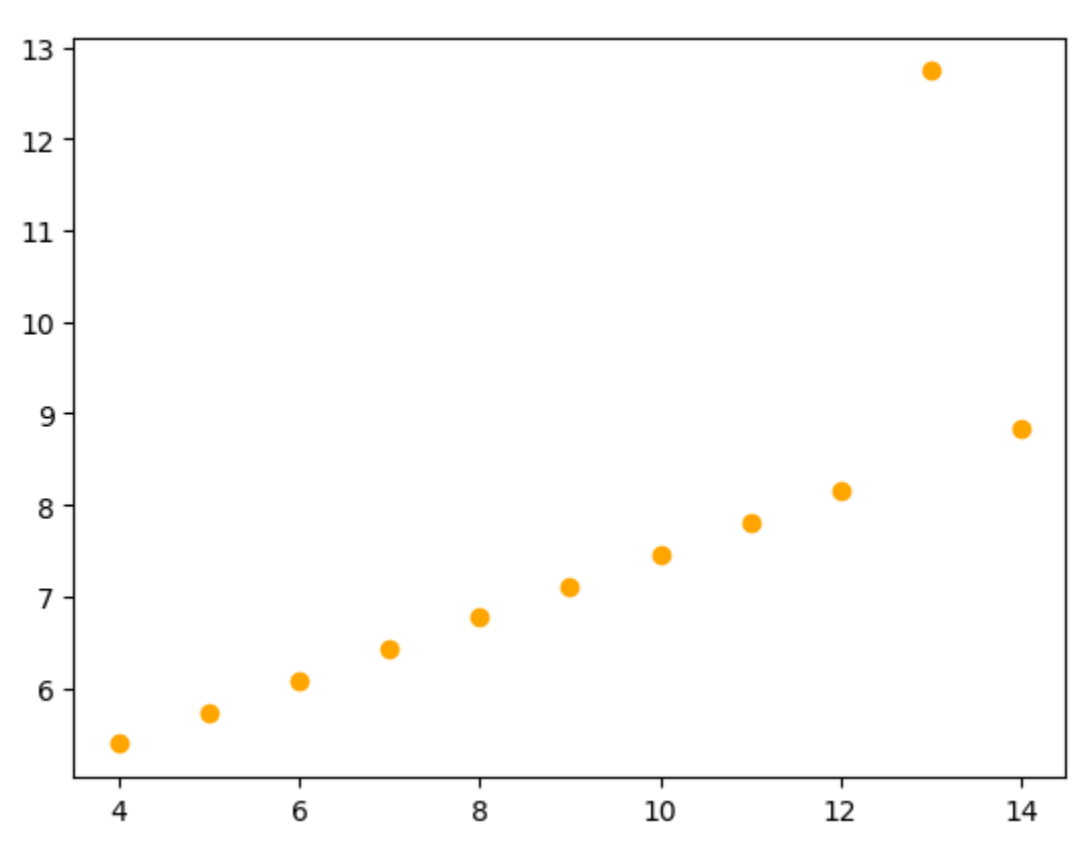


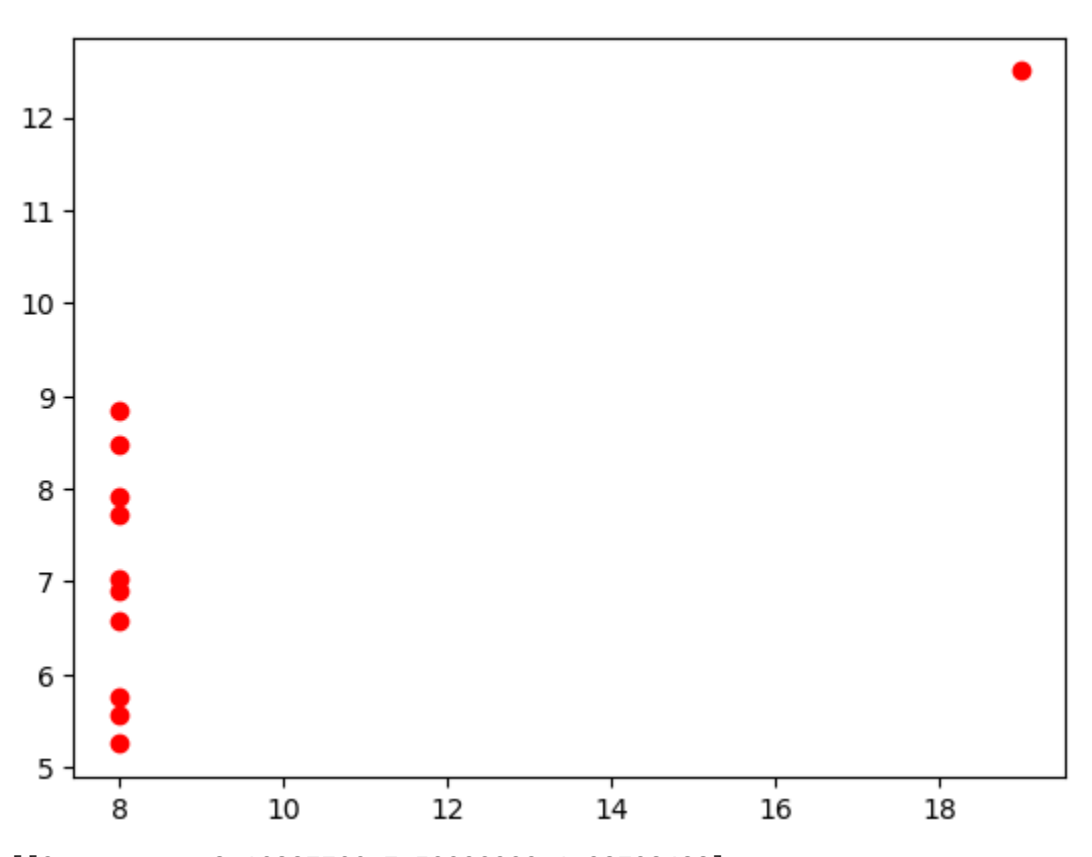
In [20]:

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

#populate the data (x and y)
x1=np.array([10,8,13,9,11,14,6,4,12,7,5])
x4=np.array([8,8,8,8,8,8,8,19,8,8,8])
y1=np.array([8.04,6.95,7.58,8.81,8.33,9.96,7.24,4.26,10.84,4.82,5.68])
y2=np.array([9.14,8.14,8.74,8.77,9.26,8.1,6.13,3.1,9.13,7.26,4.74])
y3=np.array([7.46,6.77,12.74,7.11,7.81,8.84,6.08,5.39,8.15,6.42,5.73])
y4=np.array([6.58,5.76,7.71,8.84,8.47,7.04,5.25,12.5,5.56,7.91,6.89])
#plot data set-1
plt.scatter(x1,y1,c="blue")
plt.show()
plt.scatter(x1,y2,c="green")
plt.show()
plt.scatter(x1,y3,c="orange")
plt.show()
plt.scatter(x4,y4,c="red")
plt.show()
array =np.array([[np.mean(x1),np.std(x1),np.mean(y1),np.std(y1)],
                 [np.mean(x1),np.std(x1),np.mean(y2),np.std(y2)],
                 [np.mean(x1),np.std(x1),np.mean(y3),np.std(y3)],
                 [np.mean(x4),np.std(x4),np.mean(y4),np.std(y4)]])

print(array)
pcc1 = np.corrcoef(x1,y1)
pcc2 = np.corrcoef(x1,y1)
pcc3 = np.corrcoef(x1,y1)
pcc4 = np.corrcoef(x1,y1)
print(pcc1,pcc2,pcc3,pcc4)
```





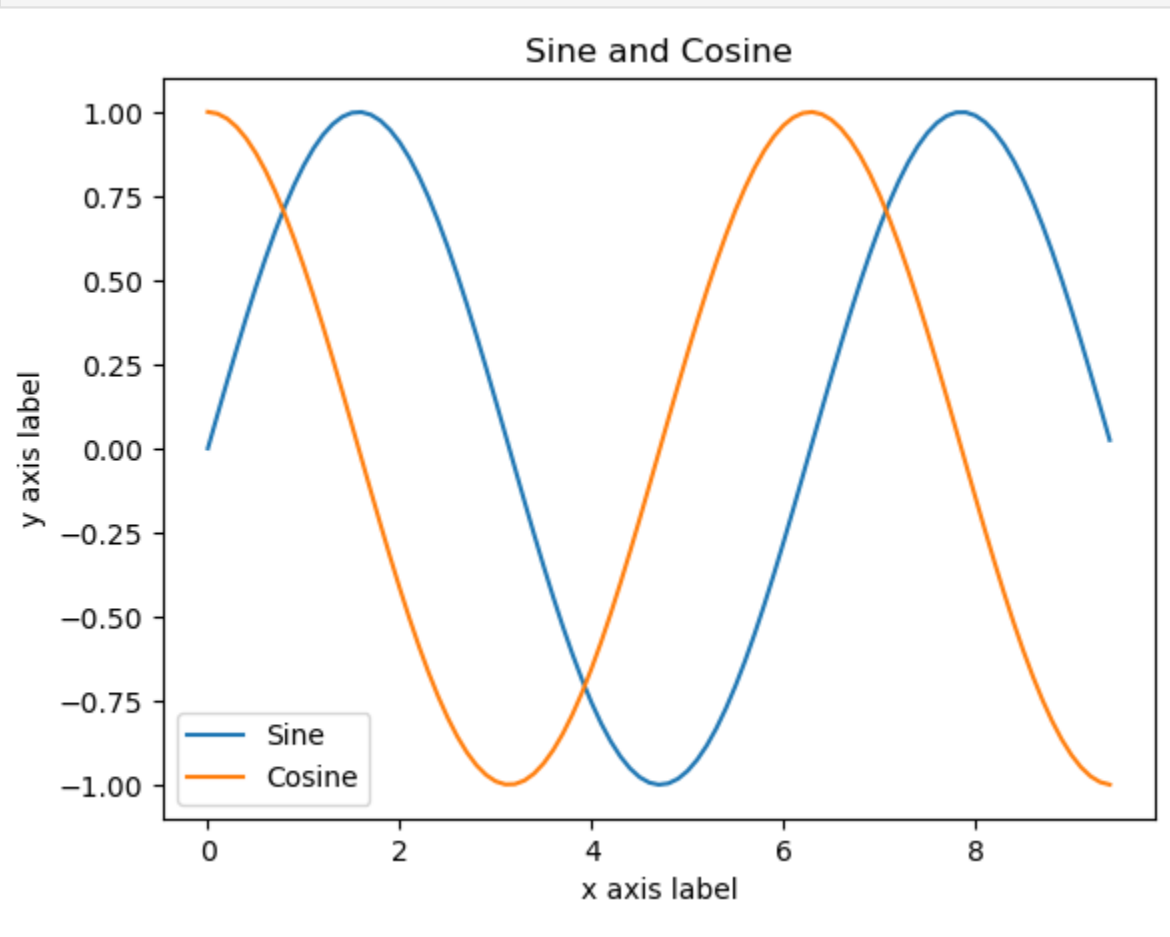




```
[[9.         3.16227766 7.50090909 1.93702422]
 [9.         3.16227766 7.50090909 1.93710869]
 [9.         3.16227766 7.5        1.93593294]
 [9.         3.16227766 7.50090909 1.93608065]]
[[1.         0.81642052]
 [0.81642052 1.        ]] [[1.         0.81642052]
 [0.81642052 1.        ]] [[1.         0.81642052]
 [0.81642052 1.        ]] [[1.         0.81642052]
 [0.81642052 1.        ]]
```

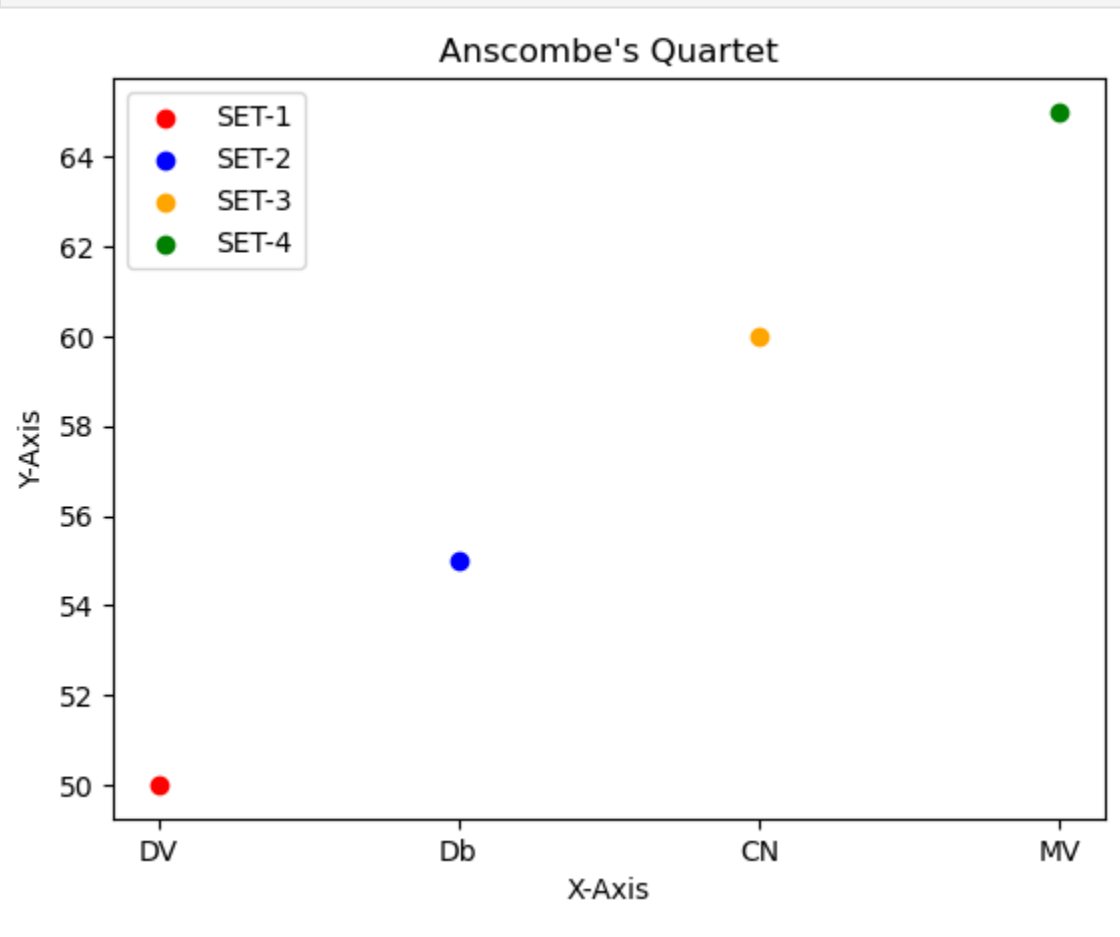
In [1]:

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

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Plot the points using matplotlib
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()
```



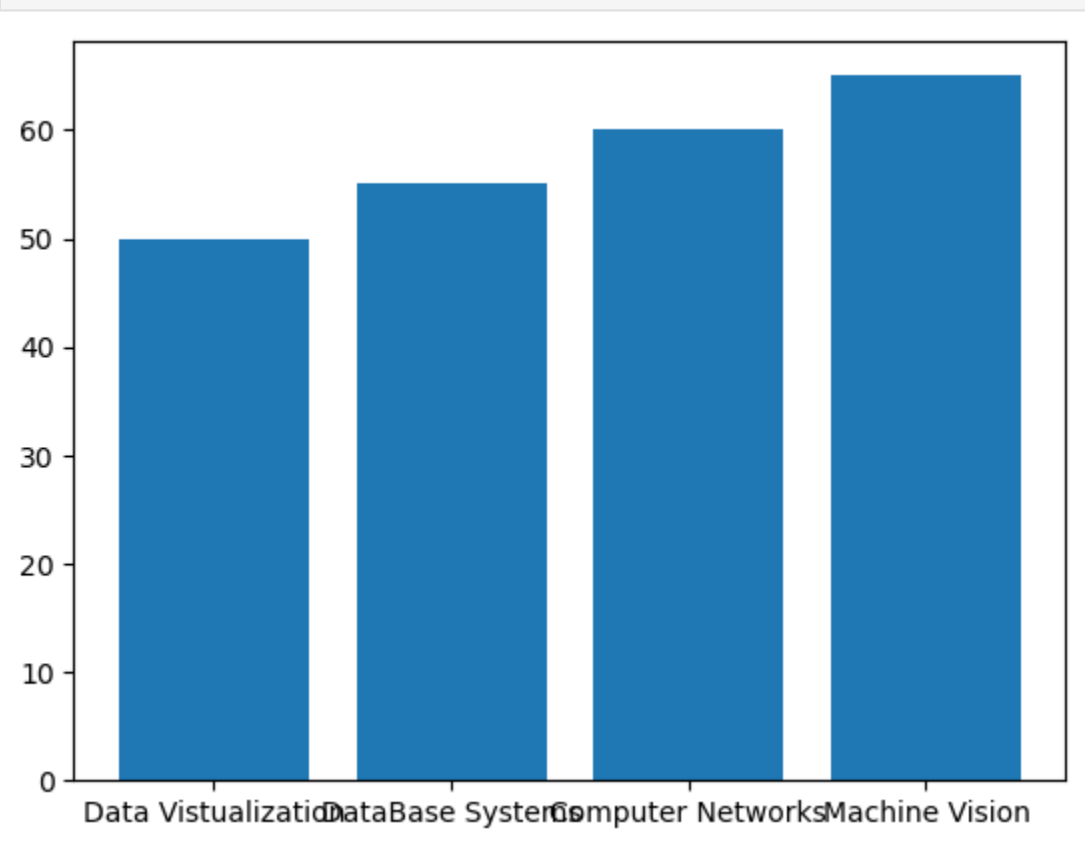
In [2]:

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

x=["Data Vistualization","DataBase Systems","Computer Networks","Machine Vision"]
y=[50,55,60,65]
x1=["DV"]
x2=["Db"]
x3=["CN"]
x4=["MV"]

y1=[50]
y2=[55]
y3=[60]
y4=[65]
plt.scatter(x1,y1,label="SET-1",c="red")
plt.scatter(x2,y2,label="SET-2",c="blue")
plt.scatter(x3,y3,label="SET-3",c="orange")
plt.scatter(x4,y4,label="SET-4",c="green")
plt.title("Anscombe's Quartet");
plt.xlabel("X-Axis");
plt.ylabel("Y-Axis");
plt.legend();
plt.show();
```



In [3]:

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

x=["Data Vistualization","DataBase Systems","Computer Networks","Machine Vision"]
y=[50,55,60,65]

plt.bar(x,y)
plt.show()
```



In [6]:

```python
#Slicing1
import numpy as np

# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Use slicing to pull out the subarray consisting of the first 2 rows
# and columns 1 and 2; b is the following array of shape (2, 2):
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]

# A slice of an array is a view into the same data, so modifying it
# will modify the original array.
print(a[0, 1])   # Prints "2"
b[0, 0] = 77     # b[0, 0] is the same piece of data as a[0, 1]
print(a[0, 1])   # Prints "77"
```

```
2
77
```

In [5]:

```python
#Slicing2
#slicing Examples
#nums = list(range(5))    # range is a built-in function that creates a list of integers
nums=[0,1,2,3,4,5,6]
print(nums)         # Prints "[0, 1, 2, 3, 4, 5, 6]"
print(nums[2:4])    # Get a slice from index 2 to 4 (exclusive); prints "[2, 3]"
print(nums[2:])     # Get a slice from index 2 to the end; prints "[2, 3, 4, 5, 6]"
print(nums[:2])     # Get a slice from the start to index 2 (exclusive); prints "[0, 1]"
print(nums[:])      # Get a slice of the whole list; prints ["0, 1, 2, 3, 4, 5, 6]"
print(nums[:-1])    # Slice indices can be negative; prints ["0, 1, 2, 3, 4, 5]"
nums[2:4] = [8, 9] # Assign a new sublist to a slice
print(nums)
```

```
[0, 1, 2, 3, 4, 5, 6]
[2, 3]
[2, 3, 4, 5, 6]
[0, 1]
[0, 1, 2, 3, 4, 5, 6]
[0, 1, 2, 3, 4, 5]
[0, 1, 8, 9, 4, 5, 6]
```

In [ ]: