Recap: Python For AI + Numpy

Rina Buoy

AMERICAN UNIVERSITY
OF PHNOM PENH
STUDY LOCALLY. LIVE GLOBALLY.

Containers - lists, dictionaries, sets, and tuples.

A list is the Python equivalent of an array, but is resizeable and can contain elements of different types.

```
xs = [3, 1, 2]   # Create a list
xs.append('bar') # Add a new element to the end of the list

nums = list(range(5)) # [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]"
```

Containers - lists

Iterating a list:

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

animals = ['cat', 'dog', 'monkey']
for idx, animal in enumerate(animals):
  print('#{}: {}'.format(idx + 1, animal))
```

Containers - lists

Iterating a list:

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

```
animals = ['cat', 'dog', 'monkey']
for idx, animal in enumerate(animals):
   print('#{}: {}'.format(idx + 1, animal))
```

Containers - lists

List comprehension:

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

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

even_squares = [x ** 2 for x in nums if x % 2 == 0]
```

Containers - Dictionaries

A dictionary stores (key, value) pairs, similar to a `Map` in Java or an object in Javascript.

```
d = {'cat': 'cute', 'dog': 'furry'}  # Create a new dictionary with some data
print(d['cat'])       # Get an entry from a dictionary; prints "cute"
print('cat' in d)     # Check if a dictionary has a given key; prints "True“

d['fish'] = 'wet'    # Set an entry in a dictionary
print(d['fish'])      # Prints "wet“

print(d['monkey'])  # KeyError: 'monkey' not a key of d

print(d.get('monkey', 'N/A'))  # Get an element with a default; prints "N/A"
print(d.get('fish', 'N/A'))    # Get an element with a default; prints "wet"
```

Containers - Dictionaries

Iterating a dictionary:

```
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal, legs in d.items():
   print('A {} has {} legs'.format(animal, legs))

nums = [0, 1, 2, 3, 4]
even_num_to_square = {x: x ** 2 for x in nums if x % 2 == 0}
print(even_num_to_square)
```

Containers - Sets

A set is an unordered collection of distinct elements.

```
animals = {'cat', 'dog'}
print('cat' in animals)   # Check if an element is in a set; prints "True"
print('fish' in animals)  # prints "False“

animals.add('fish')      # Add an element to a set
print('fish' in animals)
print(len(animals))

animals.add('cat')       # Adding an element that is already in the set does nothing
print(len(animals))
animals.remove('cat')    # Remove an element from a set
print(len(animals))
```

Containers - Sets

Iterating a set:

```
animals = {'cat', 'dog', 'fish'}
for idx, animal in enumerate(animals):
    print('#{}: {}'.format(idx + 1, animal))

print({int(sqrt(x)) for x in range(30)})
```

Containers - Tuples

A tuple is an (immutable) ordered list of values.

```
t = (5, 6)       # Create a tuple
print(type(t))

t[0] = 1 #TypeError: 'tuple' object does not support item assignment
```

Containers - Tuples

A tuple is an (immutable) ordered list of values.

```
t = (5, 6)      # Create a tuple
print(type(t))

t[0] = 1 #TypeError: 'tuple' object does not support item assignment
```

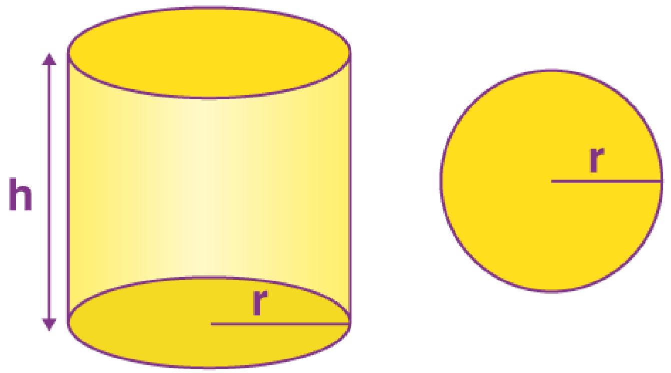
Class

Classes are a way by which, in object-oriented programming, we can create our own type of data and give them names.

```
class BacGrade:
   def __init__(self):
      self.grades = ["F","A","C","B","D","E"]

   def top(self):
      grades_sorted = sorted(self.grades)
      return grades_sorted[0]

   @staticmethod
   def format(grade='f'):

      return f'My BacII grade is {grade.upper()}'
```

```
bac = BacGrade()
bac.top() # class method

BacGrade.format('a’) #static method
```

Class - Inheritance



Inheritance is, perhaps, the most powerful feature of object-oriented programming. It just so happens that you can create a class that “inherits” methods, variables, and attributes from another class.

```
class Circle:
   def __init__(self, radius=1.0):
      self.radius = radius

   def get_area_or_volume(self):
      return self.radius**2*math.pi

c = Circle()
print(c.get_area_or_volume())
```

```
class Cylinder(Circle):
   def __init__(self, radius=1.0, height=2.0):
      super().__init__(radius)
      self.height = height
   def get_area_or_volume(self):
      return self.radius**2*math.pi*self.height

cy = Cylinder()
print(cy.get_area_or_volume())
```

File/IO

File I/O is the ability of a program to take a file as input or create a file as output.

```
open("names.txt", "w") # open or create a names.txt file and overwrite new content.
```

```
open("names.txt", “a") # open a names.txt file and add new content.
```

```
open("names.txt", “r") # open a names.txt file and read content.
```

```
file = open("names.txt", "a")
file.write(name)
file.close()
```

Or

```
with open("names.txt", "a") as file:
   file.write(f"{name}\n")
```

File/IO

File I/O is the ability of a program to take a file as input or create a file as output.

```
open("names.txt", "w") # open or create a names.txt file and overwrite new content.
```

```
open("names.txt", “a") # open a names.txt file and add new content.
```

```
open("names.txt", “r") # open a names.txt file and read content.
```

```
file = open("names.txt", "a")
file.write(name)
file.close()
```

Or

```
with open("names.txt", "a") as file:
   file.write(f"{name}\n")
```

For an image file,

```
from PIL import Image
image = Image.open(path)
```

ChatGPT

Prompt: Write a python script to open a RGB image file and convert it to grayscale

```
from PIL import Image
# Open the RGB image file
image = Image.open("input_image.jpg")
# Convert the image to grayscale
grayscale_image = image.convert("L")
# Save the grayscale image
grayscale_image.save("output_grayscale_image.jpg")
# Show the grayscale image
grayscale_image.show()
```

Remarks

- Put in 10,000 hours of deliberate effort and work to become a successful machine learning (or python) practitioner.

Andrej Karpathy

Numpy

- Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

```
import numpy as np
print(np.__version__)
```

Numpy

Creating a numpy array:

a = np.array([1, 2, 3]) #1D - i.e., rank 1

b = np.array([[1,2,3],[4,5,6]]) #2D - i.e., rank 2

Other ways:

np.zeros (shape) # e.g. np.zeros((2,3))

np.ones, np.eye, np.full etc.

Numpy - Array Indexing

You can index a numpy array in the same way you can index a list.

```
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
b = a[:2, 1:3] # a subview of a
a[0, 1] # an item of a

row_r1 = a[1, :]    # 1D view of the second row of a
row_r2 = a[1:2, :]  # 2D view of the second row of a
row_r3 = a[[1], :]  # 2D view of the second row of a
```

Numpy -Array Indexing

Integer array indexing:

```
a = np.array([[1,2], [3, 4], [5, 6]])
a[[0, 1, 2], [0, 1, 0]] # 1D view
np.array([a[0, 0], a[1, 1], a[2, 0]]) # 1D view
```

Boolean array indexing:

```
a = np.array([[1,2], [3, 4], [5, 6]])
bool_idx = (a > 2)
a[bool_idx] # OR
a[a>2]
```

Datatypes & Operations

Datatypes:

```
x = np.array([1, 2])  # Let numpy choose the datatype
y = np.array([1.0, 2.0])  # Let numpy choose the datatype
z = np.array([1, 2], dtype=np.int64)  # Force a particular datatype
```

Operations:

```
np.add(x,y) # x + y
np.substract(x,y) # x - y
np.multiply(x,y) # x * y
np.divide(x,y) # x / y
```

Matrix Multiplication:

```
x = np.array([[1,2],[3,4]]) # (2,2)
y = np.array([[5,6],[7,8]]) # (2,2)

x.dot(y)
np.dot(x, y)
x @ y
```

Datatypes & Operations

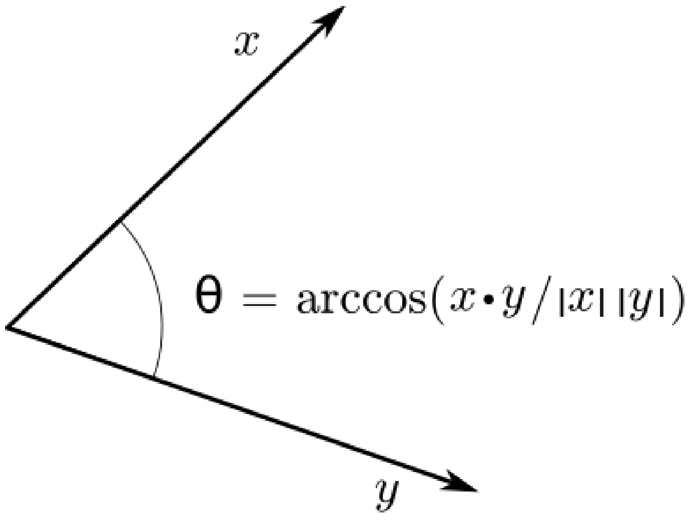
Matrix-Vector Multiplication:

```
x = np.array([[1,2],[3,4]]) # (2,2)
v = np.array([9,10]) # (2,)

print(x.dot(v)) # (2,)
print(np.dot(x, v)) # (2,)
print(x @ v) # (2,)
```

Datatypes & Operations

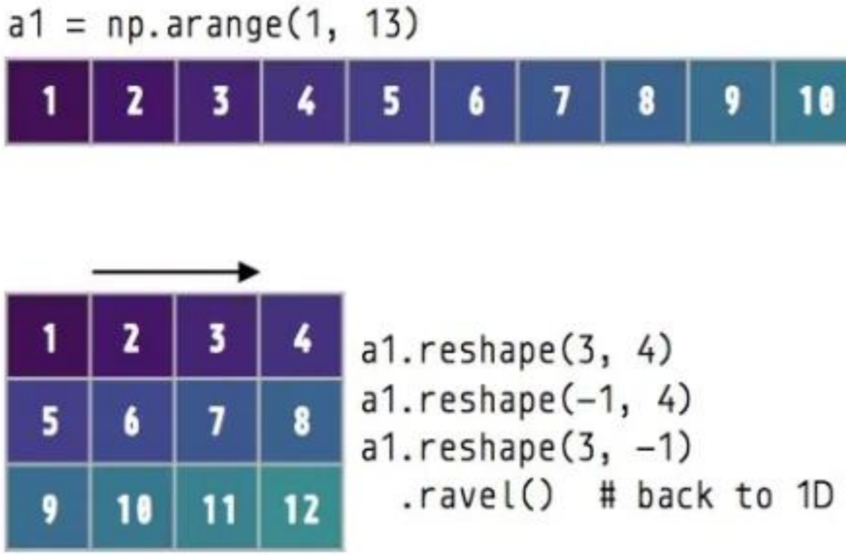
Dot product:

```
v = np.array([9,10])
w = np.array([11, 12])
# Inner product of vectors; both produce 219
print(v.dot(w))
print(np.dot(v, w))
```



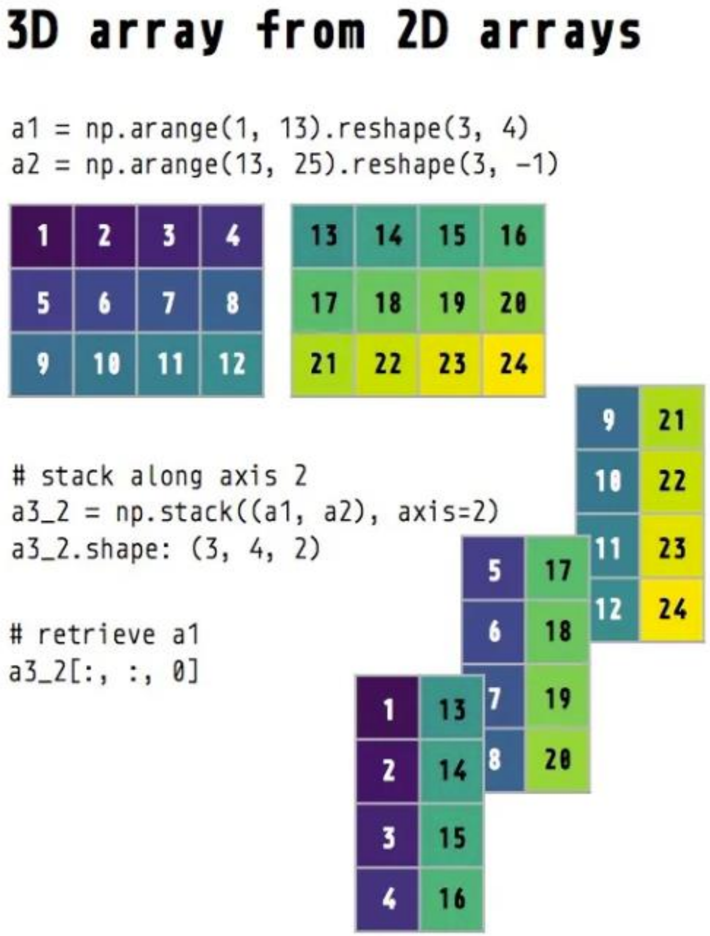
https://en.wikipedia.org/wiki/File:Inner-product-angle.svg

Numpy Array Reshaping & Ravelling



```
a1 = np.arange(1, 13)
```

```
a1.reshape(3, 4)
a1.reshape(-1, 4)
a1.reshape(3, -1)
  .ravel()  # back to 1D
```



3D array from 2D arrays

```
a1 = np.arange(1, 13).reshape(3, 4)
a2 = np.arange(13, 25).reshape(3, -1)
```

```
# stack along axis 2
a3_2 = np.stack((a1, a2), axis=2)
a3_2.shape: (3, 4, 2)

# retrieve a1
a3_2[:, :, 0]
```

https://towardsdatascience.com/reshaping-numpy-arrays-in-python-a-step-by-step-pictorial-tutorial-aed5f471cf0b

Broadcasting

X

1	0	1
1	2	3
1	2	3
1	2	2
1	2	1
2	2	1
1	1	1

y

1	0	1

Y_stracked

1	0	1
1	0	1
1	0	1
1	0	1
1	0	1
1	0	1
1	0	1

How to compute **X** + y:

Option 1:

```
for i in range(7):
        X[i, :] + y
```

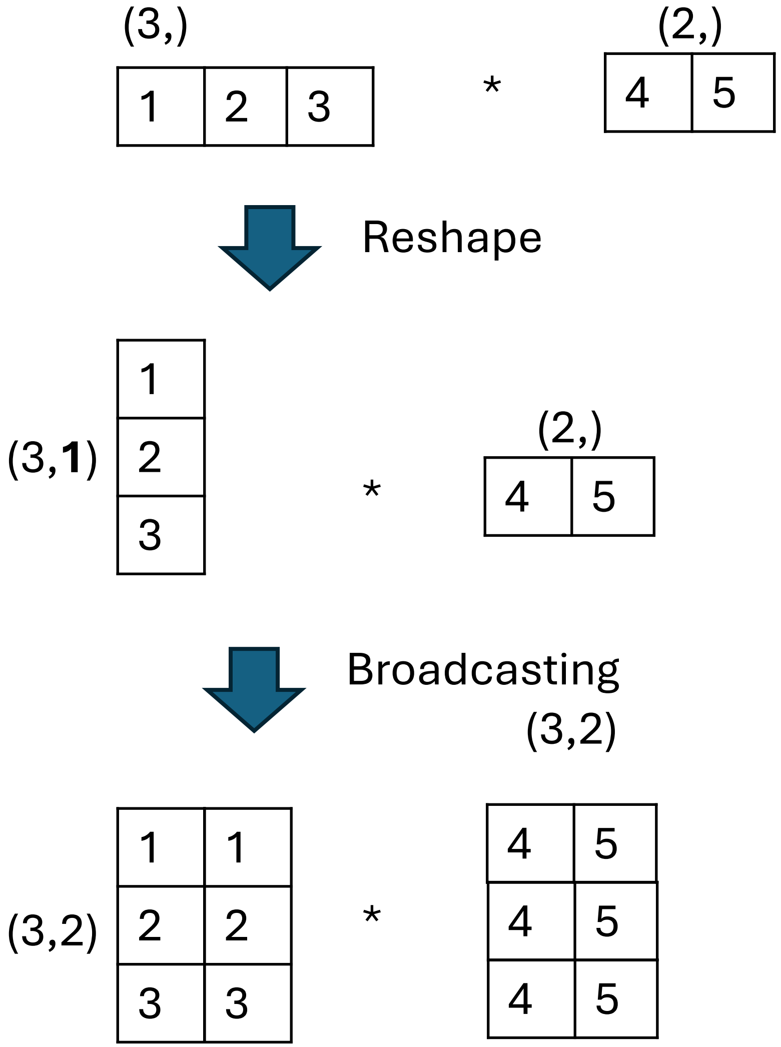
Option 2:

```
X + Y_stracked
```

Broadcasting:

```
X + y # (7,3)
```

Broadcasting – Cases



(3,)

1	2	3

*

(2,)

4	5

Reshape

(3,**1**)

1
2
3

*

(2,)

4	5

Broadcasting

(3,2)

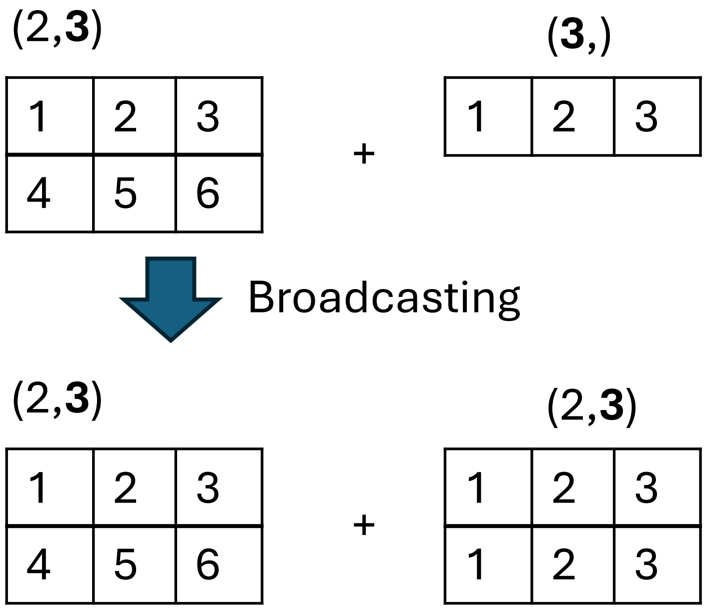
1	1
2	2
3	3

*

(3,2)

4	5
4	5
4	5

Broadcasting – Cases



(2,**3**)

1	2	3
4	5	6

+

(**3**,)

1	2	3

Broadcasting

(2,**3**)

1	2	3
4	5	6

+

(2,**3**)

1	2	3
1	2	3

Matplotlib

Import:

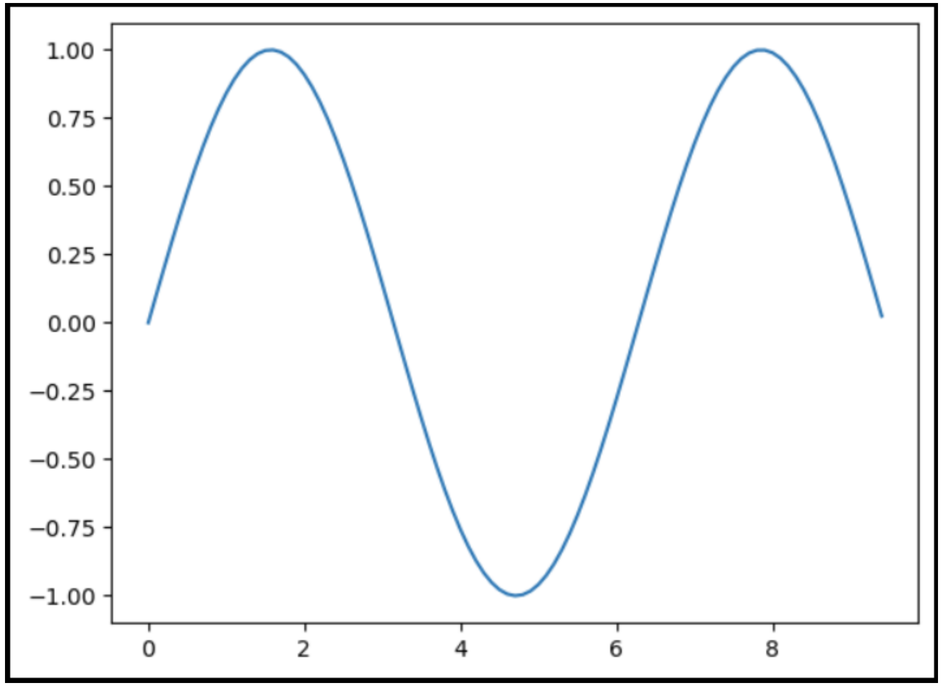
import matplotlib.pyplot as plt

%matplotlib inline

Matplotlib

plt.plot -> Line plot :

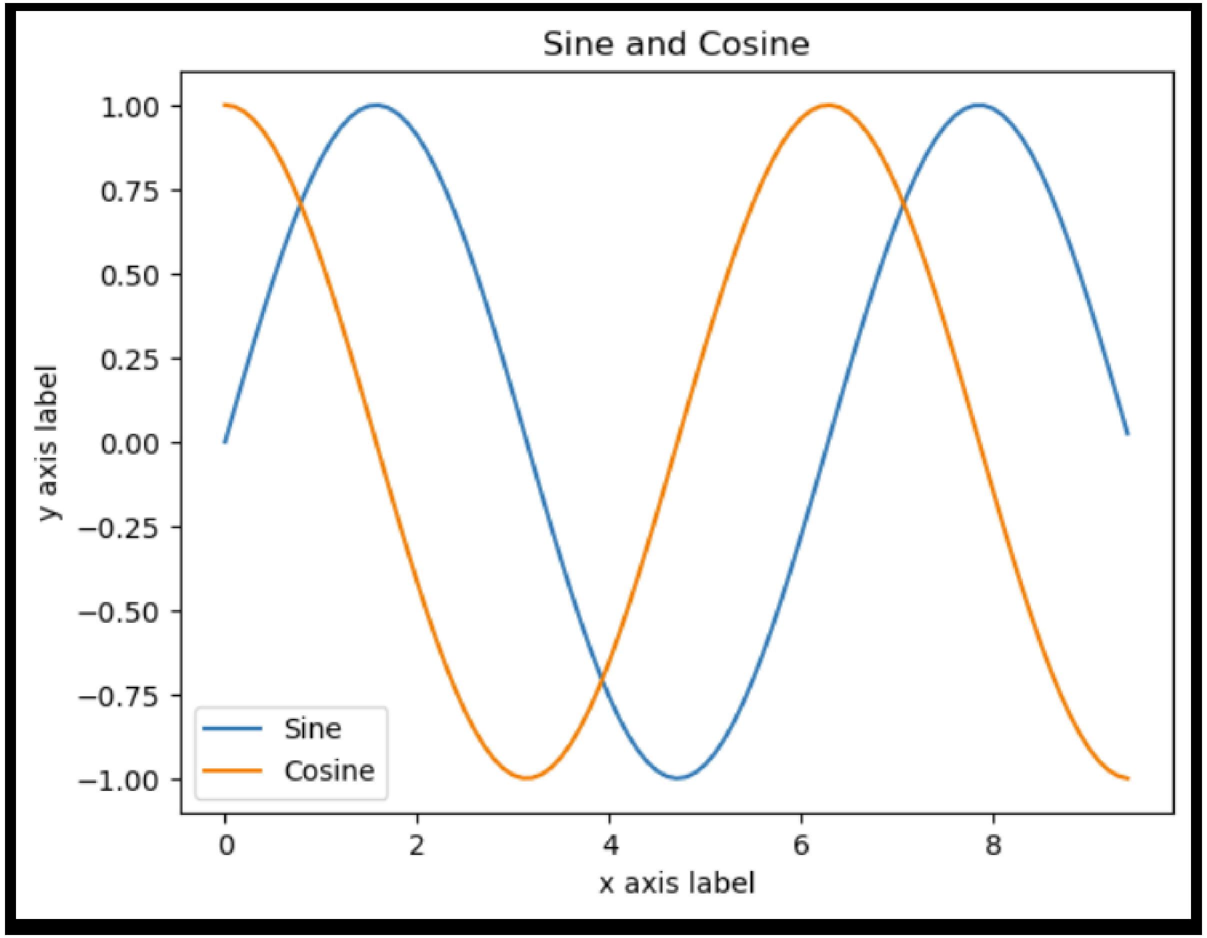
```
# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

# Plot the points using matplotlib
plt.plot(x, y)
```



Matplotlib

plt.plot -> Line plot :

```
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'])
```

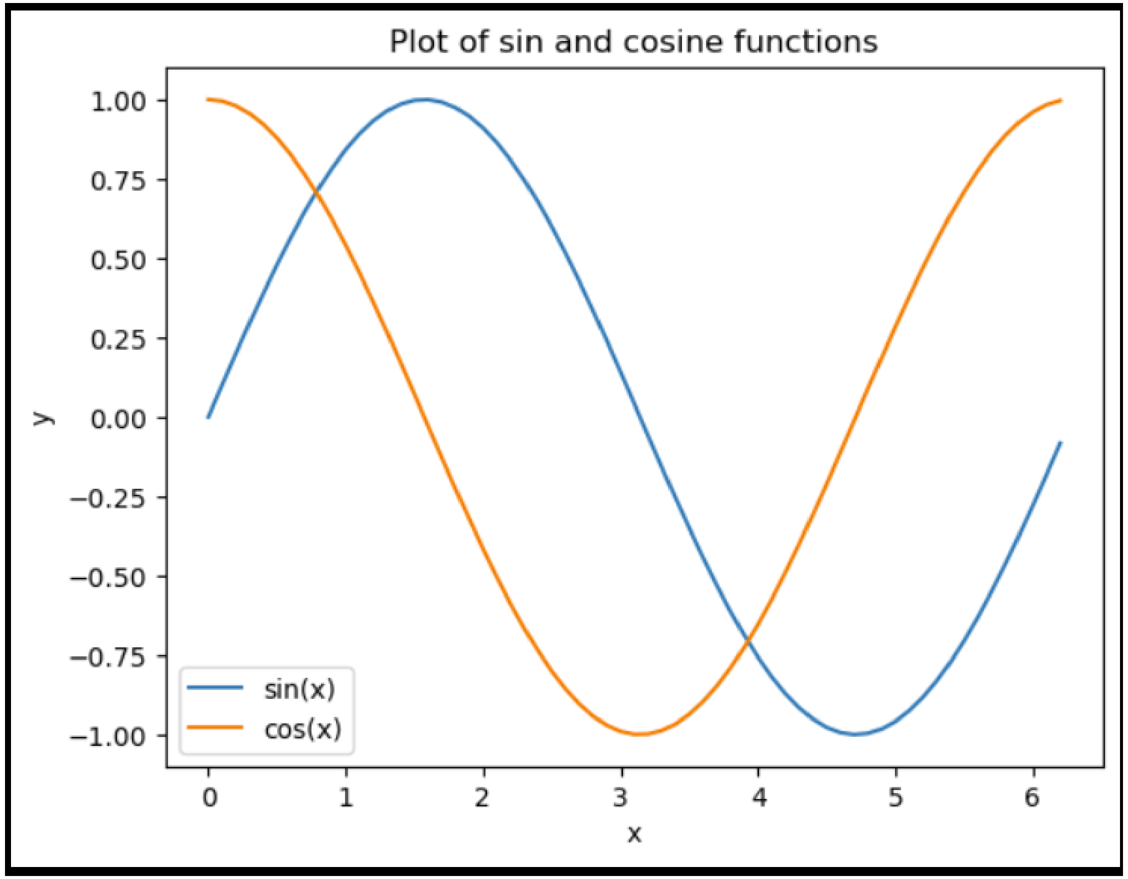


ChatGPT

Prompt: Write a python script to generate a line plot of sin and cosine functions with proper axis labels

plt.plot -> Line plot :

```
# Generate x values from 0 to 2*pi with a step of 0.1
x = np.arange(0, 2*np.pi, 0.1)
# Calculate sin and cosine values for the x values
y_sin = np.sin(x)
y_cos = np.cos(x)
# Plot sin and cosine functions
plt.plot(x, y_sin, label='sin(x)')
plt.plot(x, y_cos, label='cos(x)')
# Add labels and title
plt.xlabel('x')
plt.ylabel('y')
plt.title('Plot of sin and cosine functions')
```



Matplotlib

plt.plot -> Line plot :

#grid that has height 2 and width 1,

# and set the first such subplot as active.

plt.subplot(2, 1, 1)

# Make the first plot

plt.plot(x, y_sin)

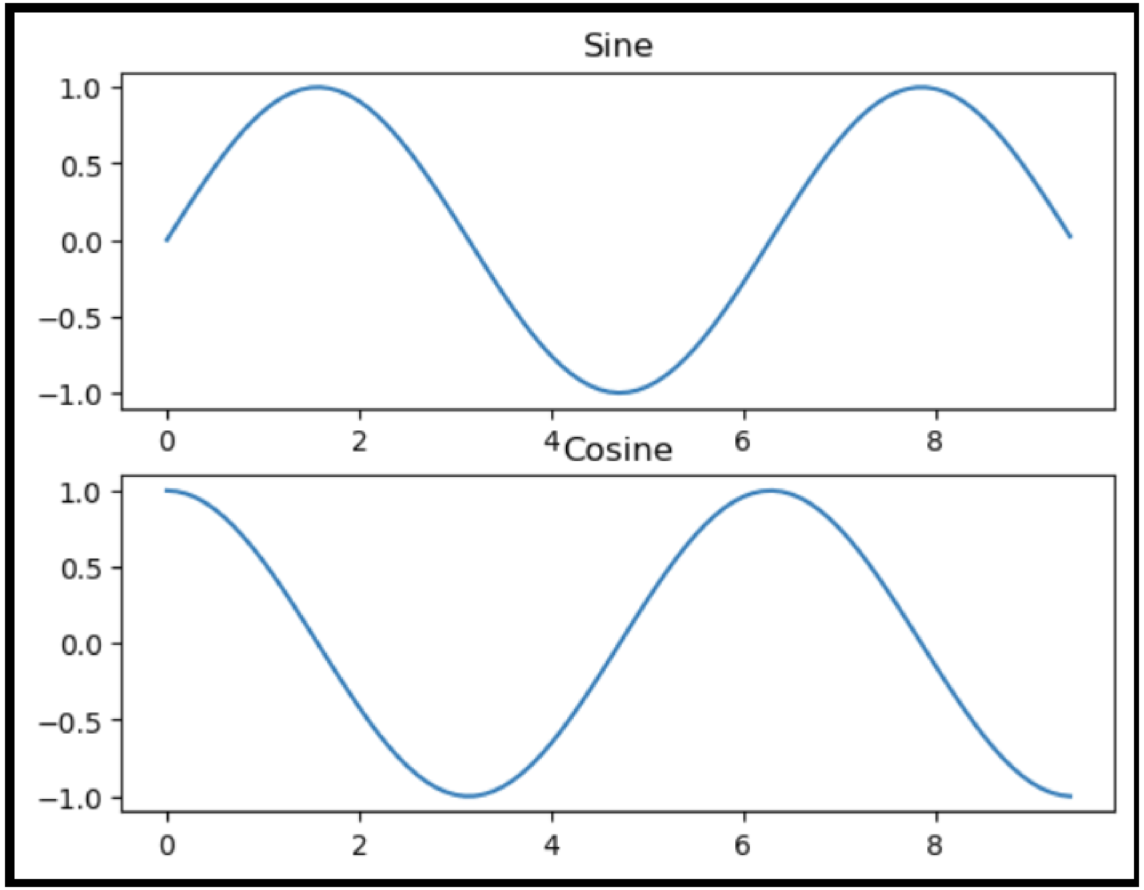
plt.title('Sine')

# Set the second subplot as active, and make the second plot.

plt.subplot(2, 1, 2)

plt.plot(x, y_cos)

plt.title('Cosine')



Matplotlib

plt.plot -> Line plot :

#grid that has height 2 and width 1,

# and set the first such subplot as active.

plt.subplot(2, 1, 1)

# Make the first plot

plt.plot(x, y_sin)

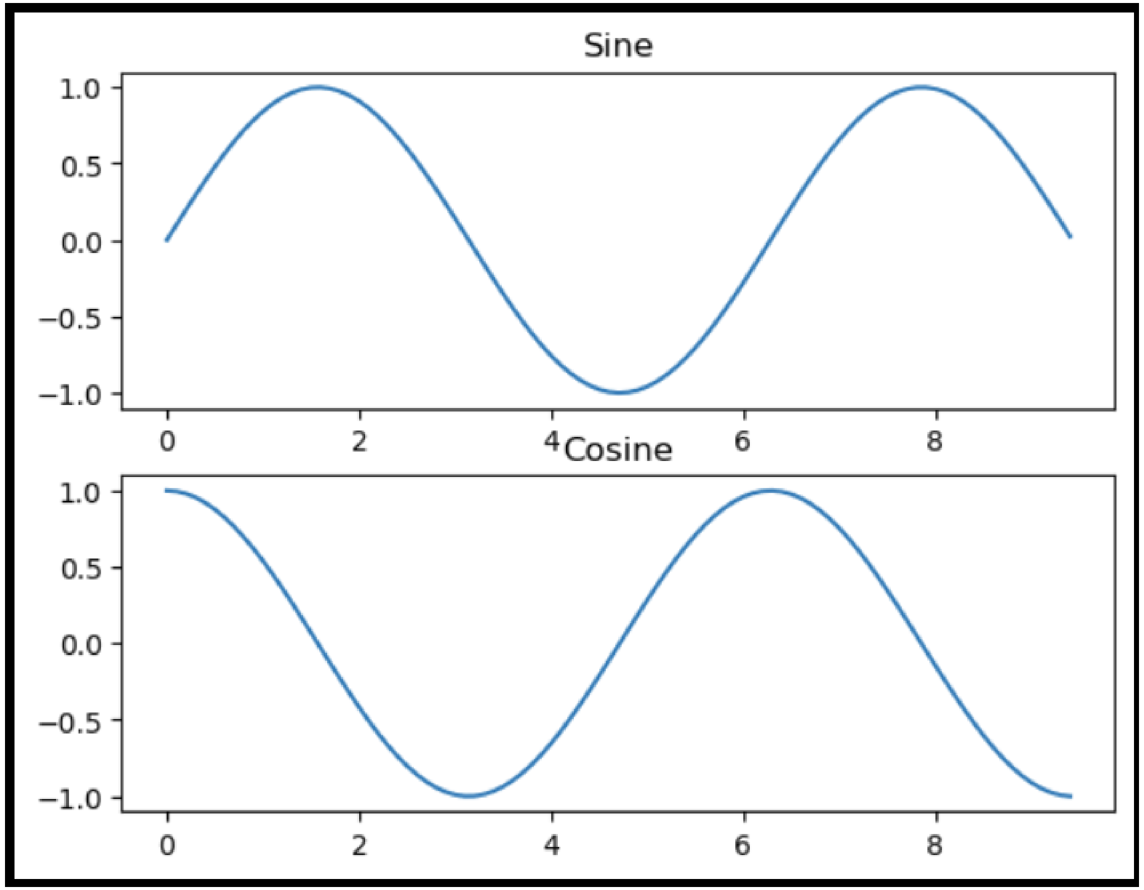
plt.title('Sine’)

# Set the second subplot as active, and make the second plot.

plt.subplot(2, 1, 2)

plt.plot(x, y_cos)

plt.title('Cosine')

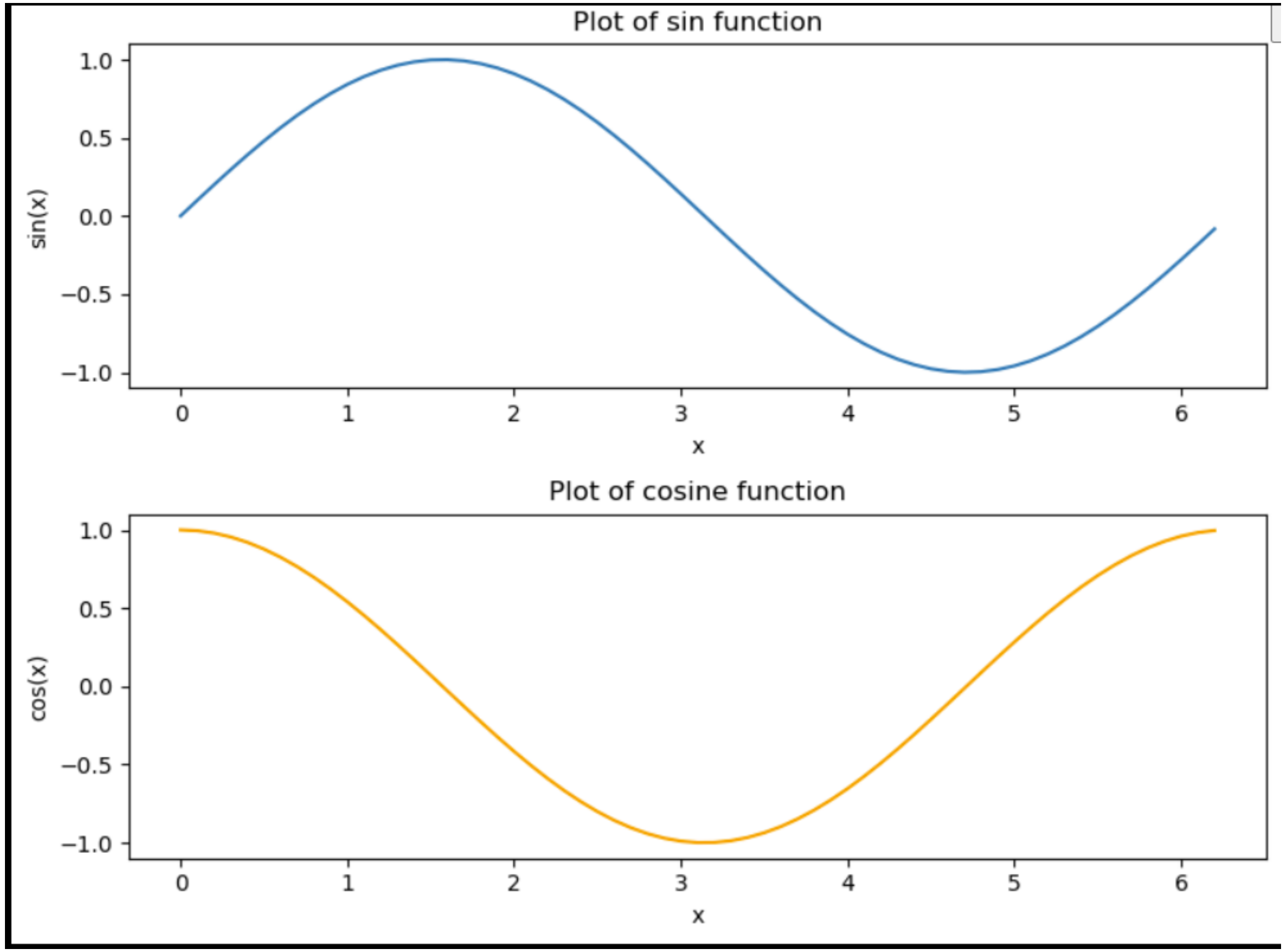


ChatGPT

Prompt: Write a python script to generate two subplots of sin and cosine functions with proper axis labels

plt.plot -> Line plot :

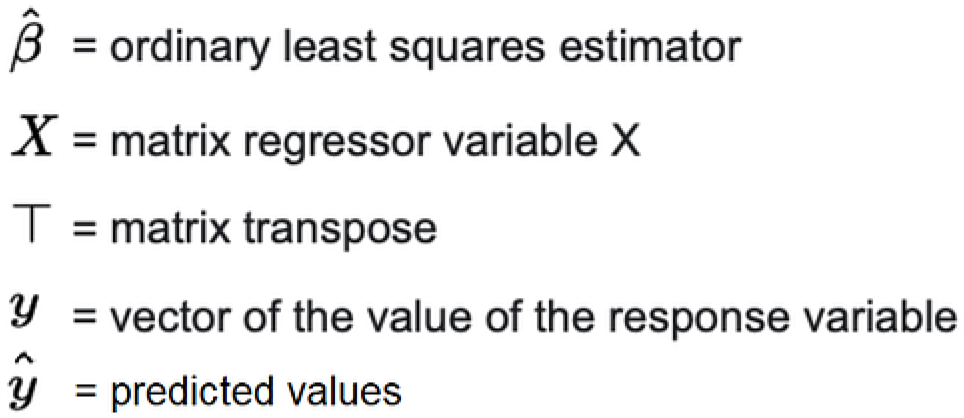
```
# Create subplots
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(8, 6))
# Plot sin function in the first subplot
ax1.plot(x, y_sin, label='sin(x)')
ax1.set_xlabel('x')
ax1.set_ylabel('sin(x)')
ax1.set_title('Plot of sin function')
# Plot cosine function in the second subplot
ax2.plot(x, y_cos, label='cos(x)', color='orange')
ax2.set_xlabel('x')
ax2.set_ylabel('cos(x)')
ax2.set_title('Plot of cosine function')
# Adjust layout to prevent overlap
plt.tight_layout()
```



Simple Linear Regression

$$\hat{\beta} = \left(\mathbf{X}^{\top}\mathbf{X}\right)^{-1}\mathbf{X}^{\top}\mathbf{y}$$

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\beta}$$



$\hat{\beta}$ = ordinary least squares estimator

X = matrix regressor variable X

$\top$ = matrix transpose

y = vector of the value of the response variable

$\hat{y}$ = predicted values

$$\hat{\beta} = \begin{bmatrix}\beta_0\\ \beta_1\end{bmatrix}$$

https://medium.com/@shuv.sdr/simple-linear-regression-in-python-a0069b325bf8