

# CS229

## Python & Numpy

Jingbo Yang

# How is python related to with others?

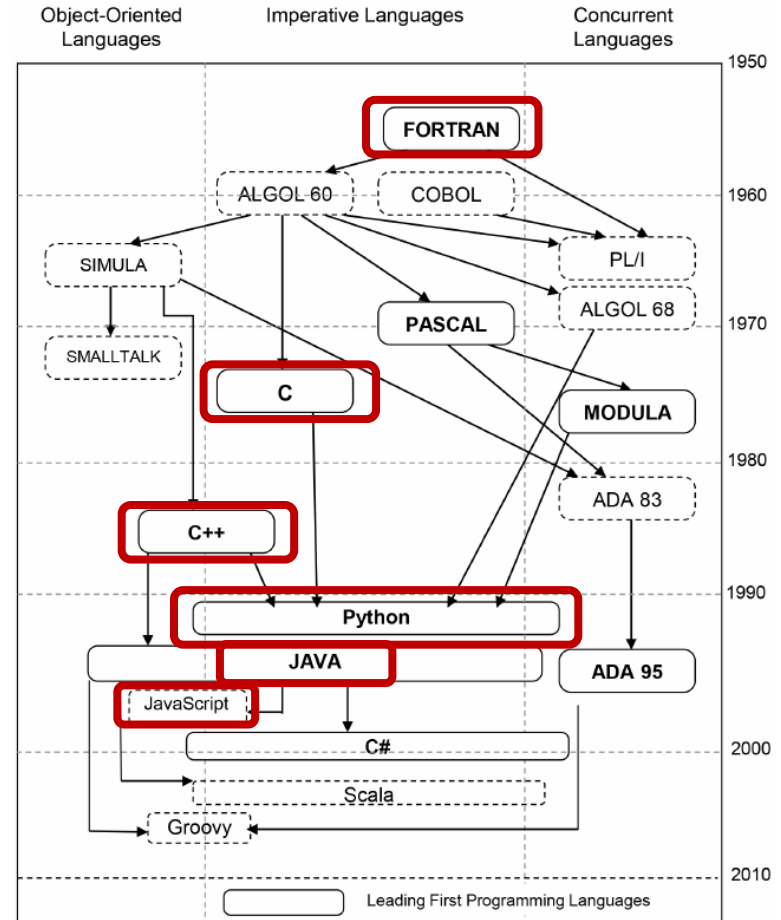
Python 2.0 released in 2000

(Python 2.7 “end-of-life” in 2020)

Python 3.0 released in 2008

(Python 3.6+ for CS 229)

Can run interpreted, like MATLAB



# Before you start

## Use Anaconda

Create a new environment (full Conda)

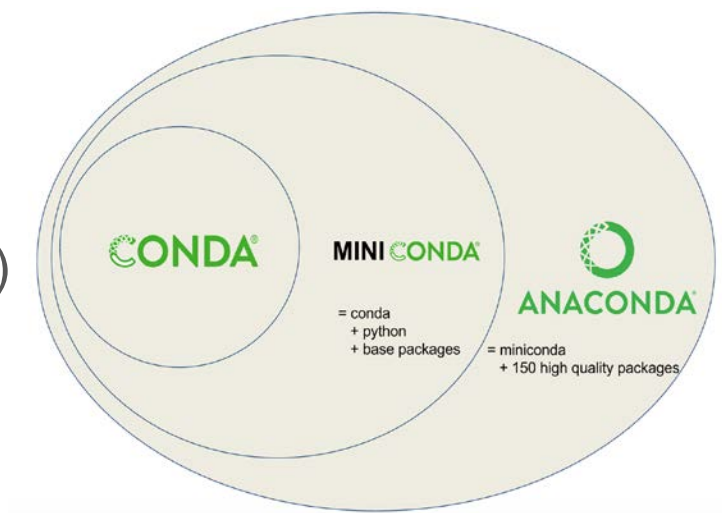
```
conda create -n cs229
```

Create an environment (Miniconda)

```
conda env create -f environment.yml
```

Activate an environment after creation

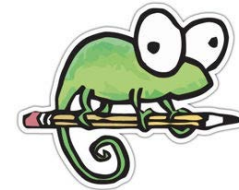
```
conda activate cs229
```



# Notepad is not your friend ...

Get a text editor/IDE

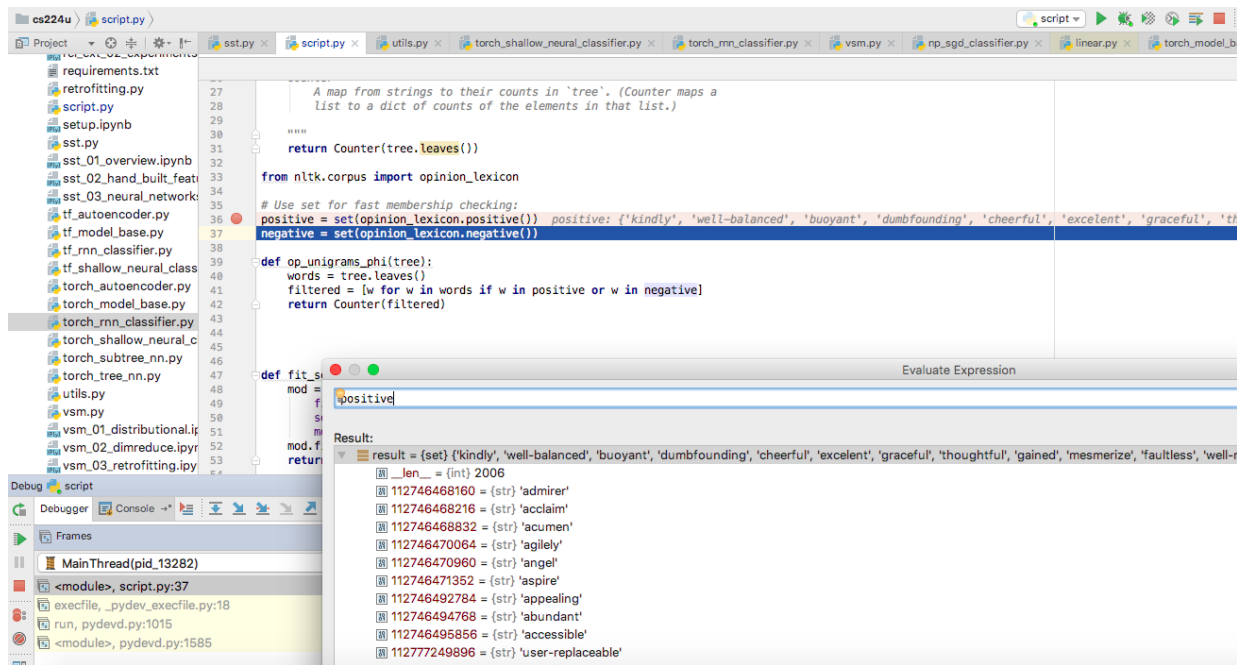
- PyCharm (IDE)
- Visual Studio Code (IDE??)
- Sublime Text (IDE??)
- Notepad ++/gedit
- Vim (for Linux)



# To make you more prepared

## PyCharm

- Great debugger
- Proper project management

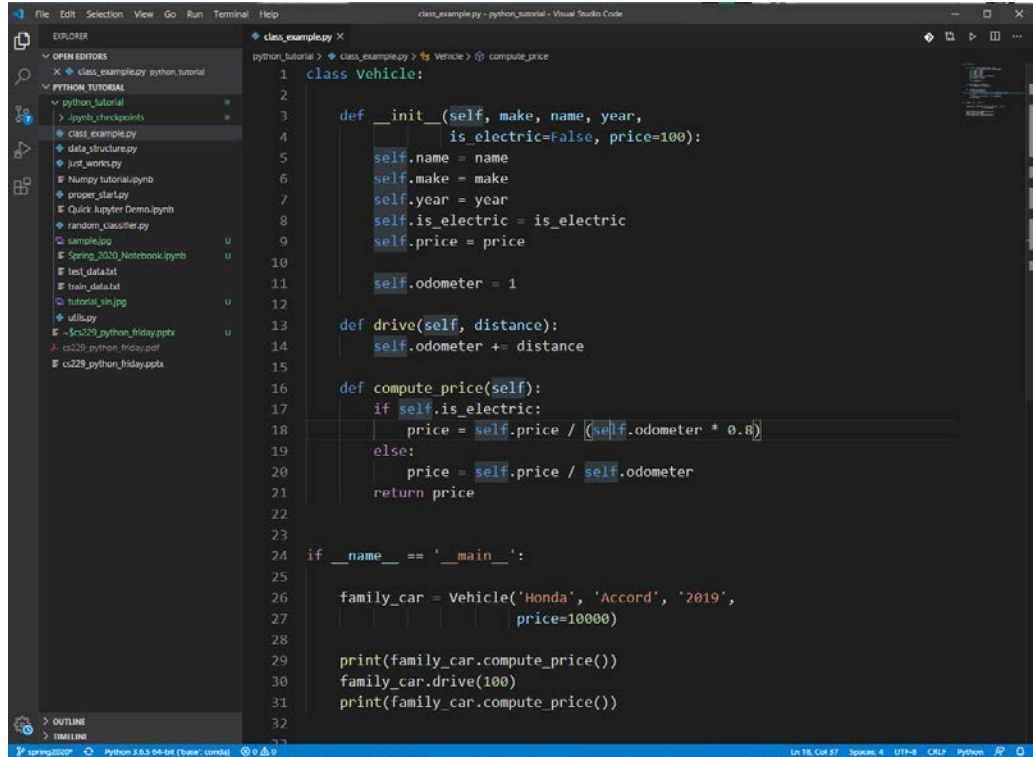


FYI, professional version free for students: <https://www.jetbrains.com/student/>

# To make you more prepared

## Visual Studio Code

- Light weight
- Wide variety of plugins to enable support for all languages
- Better UI



The screenshot shows the Visual Studio Code interface with a Python file named `class_example.py` open. The code defines a `Vehicle` class with the following methods and attributes:

```
1 class Vehicle:
2
3     def __init__(self, make, name, year,
4                 is_electric=False, price=100):
5         self.name = name
6         self.make = make
7         self.year = year
8         self.is_electric = is_electric
9         self.price = price
10
11         self.odometer = 1
12
13     def drive(self, distance):
14         self.odometer += distance
15
16     def compute_price(self):
17         if self.is_electric:
18             price = self.price / (self.odometer * 0.8)
19         else:
20             price = self.price / self.odometer
21         return price
22
23
24 if __name__ == '__main__':
25
26     family_car = Vehicle('Honda', 'Accord', '2019',
27                          price=10000)
28
29     print(family_car.compute_price())
30     family_car.drive(100)
31     print(family_car.compute_price())
32
```

# Basic Python

# String manipulation

## Formatting

```
print('I love CS229. (upper)'.upper())
print('I love CS229. (rjust 20)'.rjust(20))
print('we love CS229. (capitalize)'.capitalize())
print('      I love CS229. (strip)      '.strip())
```

## Concatenation

```
print('I like ' + str(cs_class_code) + ' a lot!')
print(f'{print} (print a function)')
print(f'{type(229)} (print a type)')
```

## Formatting

```
print('Old school formatting: {:.2F}'.format(1.358))
```



# List

## List creation

```
list_1 = ['one', 'two', 'three']
```

```
list_1.append(4)  
list_1.insert(0, 'ZERO')
```

## Insertion/extension

```
list_2 = [1, 2, 3]  
list_1.extend(list_2)
```

## List comprehension

```
long_list = [i for i in range(9)]  
long_long_list = [(i, j) for i in range(3)  
                  for j in range(5)]  
long_list_list = [[i for i in range(3)]  
                  for _ in range(5)]
```

```
sorted(random_list)
```

## Sorting

```
random_list_2 = [(3, 'z'), (12, 'r'), (6, 'e'),  
                (8, 'c'), (2, 'g')]  
sorted(random_list_2, key=lambda x: x[1])
```

# Dictionary and Set

## Set

(unordered, unique)

```
my_set = {i ** 2 for i in range(10)}  
{0, 1, 64, 4, 36, 9, 16, 49, 81, 25}
```

## Dictionary

(mapping)

```
my_dict = {(5 - i): i ** 2 for i in range(10)}  
{5: 0, 4: 1, 3: 4, 2: 9, 1: 16, 0: 25, -1: 36,  
-2: 49, -3: 64, -4: 81}  
dict_keys([5, 4, 3, 2, 1, 0, -1, -2, -3, -4])
```

## Dictionary update

```
second_dict = {'a': 10, 'b': 11}  
my_dict.update(second_dict)
```

## Iterate through items

```
for k, it in my_dict.items():  
    print(k, it)
```

# Numpy & Scipy

# What is Numpy?

Numpy – package for vector and matrix manipulation

FYI, if you are interested in how/why vectorization is faster, checkout topics such as (completely optional)

- AVX instruction set (SIMD) and structure of x86 and RISC

- OpenMP and CUDA for multiprocessing

- Assembly-level optimization, memory stride, caching, etc.

- Or even about memory management, virtualization

- More bare metal → FPGA, TPU

# Convenient math functions, read before use!

| Python Command             | Description  |
|----------------------------|--|
| <code>np.linalg.inv</code> | Inverse of matrix (numpy as equivalent)  |
| <code>np.linalg.eig</code> | Get eigen value (Read documentation on <code>eigh</code> and numpy equivalent) |
| <code>np.matmul</code>     | Matrix multiply  |
| <code>np.zeros</code>      | Create a matrix filled with zeros (Read on <code>np.ones</code> )              |
| <code>np.arange</code>     | Start, stop, step size (Read on <code>np.linspace</code> )                     |
| <code>np.identity</code>   | Create an identity matrix  |
| <code>np.vstack</code>     | Vertically stack 2 arrays (Read on <code>np.hstack</code> )                    |

# Your friend for debugging

| Python Command                           | Description  |
|--|--|
| <code>array.shape</code>                 | Get shape of numpy array   |
| <code>array.dtype</code>                 | Check data type of array (for precision, for weird behavior)   |
| <code>type(stuff)</code>                 | Get type of a variable   |
| <code>import pdb; pdb.set_trace()</code> | Set a breakpoint ( <a href="https://docs.python.org/3/library/pdb.html">https://docs.python.org/3/library/pdb.html</a> ) |
| <code>print(f'My name is {name}')</code> | Easy way to construct a message  |

# Basic Numpy usage

Initialization from Python lists

```
array_1d = np.array([1, 2, 3, 4])  
array_1by4 = np.array([[1, 2, 3, 4]])
```

Lists with different types  
(Numpy auto-casts to higher  
precision, but it should be  
reasonably consistent)

```
large_array = np.array([i for i in range(400)])  
large_array = large_array.reshape((20, 20))  
  
from_list = np.array([1, 2, 3])  
from_list_2d = np.array([[1, 2, 3.0], [4, 5, 6]])  
from_list_bad_type = np.array([1, 2, 3, 'a'])
```

```
print(f'Data type of integer is {from_list.dtype}')  
print(f'Data type of float is {from_list_2d.dtype}')
```

Numpy supports many types  
of algebra on an entire array

```
array_1 + 5  
array_1 * 5  
np.sqrt(array_1)  
np.power(array_1, 2)  
np.exp(array_1)  
np.log(array_1)
```

# Dot product and matrix multiplication

A few ways to write dot product

```
array_1 @ array_2  
array_1.dot(array_2)  
np.dot(array_1, array_2)
```

Matrix multiplication like  $Ax$

```
weight_matrix = np.array([1, 2, 3, 4]).reshape(2, 2)  
sample = np.array([[50, 60]]).T  
np.matmul(weight_matrix, sample)
```

2D matrix multiplication

```
mat1 = np.array([[1, 2], [3, 4]])  
mat2 = np.array([[5, 6], [7, 8]])  
np.matmul(mat1, mat2)
```

Element-wise multiplication

```
a = np.array([i for i in range(10)]).reshape  
(2, 5)
```

```
a * a  
np.multiply(a, a)  
np.multiply(a, 10)
```



# Broadcasting

Numpy compares dimensions of operands, then infers missing/mismatched dimensions so the operation is still valid. Be careful with *DIMENSIONS*

```
op1 = np.array([i for i in range(9)]).reshape(3, 3)
op2 = np.array([[1, 2, 3]])
op3 = np.array([1, 2, 3])
```

```
# Notice that the results here are DIFFERENT!
```

```
pp.pprint(op1 + op2)
pp.pprint(op1 + op2.T)
```

```
array([[ 1, 3, 5],
       [ 4, 6, 8],
       [ 7, 9, 11]])
```

```
array([[ 1, 2, 3],
       [ 5, 6, 7],
       [ 9, 10, 11]])
```

```
# Notice that the results here are THE SAME!
```

```
pp.pprint(op1 + op3)
pp.pprint(op1 + op3.T)
```

```
array([[ 1, 3, 5],
       [ 4, 6, 8],
       [ 7, 9, 11]])
```

```
array([[ 1, 3, 5],
       [ 4, 6, 8],
       [ 7, 9, 11]])
```



# Why should I vectorize my code?

Shorter code, faster execution

```
a = np.random.random(500000)
b = np.random.random(500000)
```

With loop

```
dot = 0.0
for i in range(len(a)):
    dot += a[i] * b[i]

print(dot)
```

Wall time: 345ms

Numpy dot product

```
print(np.array(a).dot(np.array(b)))
```

Wall time: 2.9ms

# An example with pairwise distance

Speed up depends on setup and nature of computation

```
samples = np.random.random((100, 5))
```

With loop

```
total_dist = []  
for s1 in samples:  
    for s2 in samples:  
        d = np.linalg.norm(s1 - s2)  
        total_dist.append(d)
```

```
avg_dist = np.mean(total_dist)
```

Wall time: 162ms

(imagine without Numpy norm)

Numpy with broadcasting

```
diff = samples[:, np.newaxis, :] -  
        samples[np.newaxis, :, :]  
distances = np.linalg.norm(diff, axis=-1)  
avg_dist = np.mean(distances)
```

Wall time: 3.5ms

# Plotting

# Other Python packages/tools

## Jupyter Notebook

- Interactive, re-execution, result storage

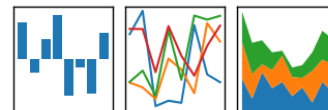


## Pandas (<https://pandas.pydata.org/>)

- Dataframe (database/Excel-like)
- Easy filtering, aggregation (also plotting, but few people uses Pandas for plotting)

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



## Matplotlib

- Visualization (line, scatter, bar, images and even interactive 3D)

matplotlib

# Example plots

<https://matplotlib.org/3.1.1/gallery/index.html>

## Import

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

## Create data

```
# Data for plotting
t = np.arange(0.0, 2.0, 0.01)
s = 1 + np.sin(2 * np.pi * t)
```

## Plotting

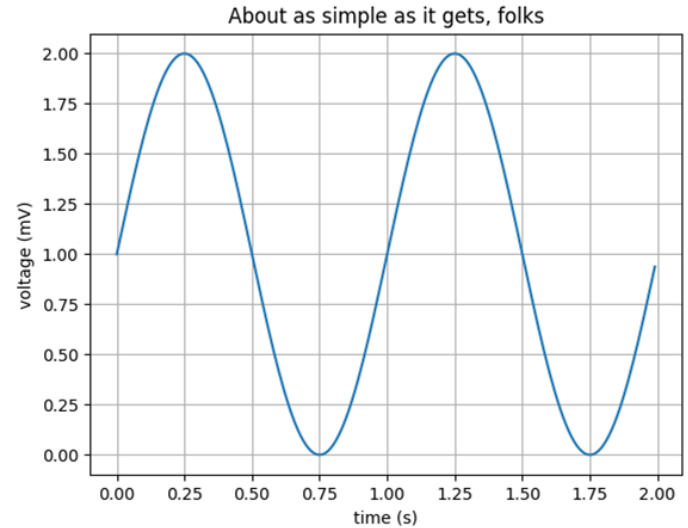
```
fig, ax = plt.subplots()
ax.plot(t, s)
```

## Format plot

```
ax.set(xlabel='time (s)', ylabel='voltage (mV)',
       title='About as simple as it gets, folks')
ax.grid()
```

## Save/show

```
fig.savefig("test.png")
plt.show()
```



# Plot with dash lines and legend

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

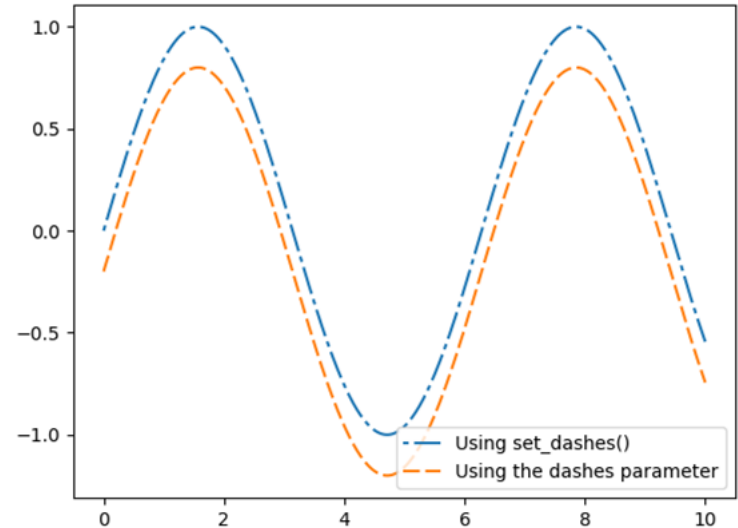
x = np.linspace(0, 10, 500)
y = np.sin(x)

fig, ax = plt.subplots()

line1, = ax.plot(x, y, label='Using set_dashes()')
# 2pt line, 2pt break, 10pt line, 2pt break
line1.set_dashes([2, 2, 10, 2])

line2, = ax.plot(x, y - 0.2, dashes=[6, 2],
                 label='Using the dashes parameter')

ax.legend()
plt.show()
```





# Using subplot

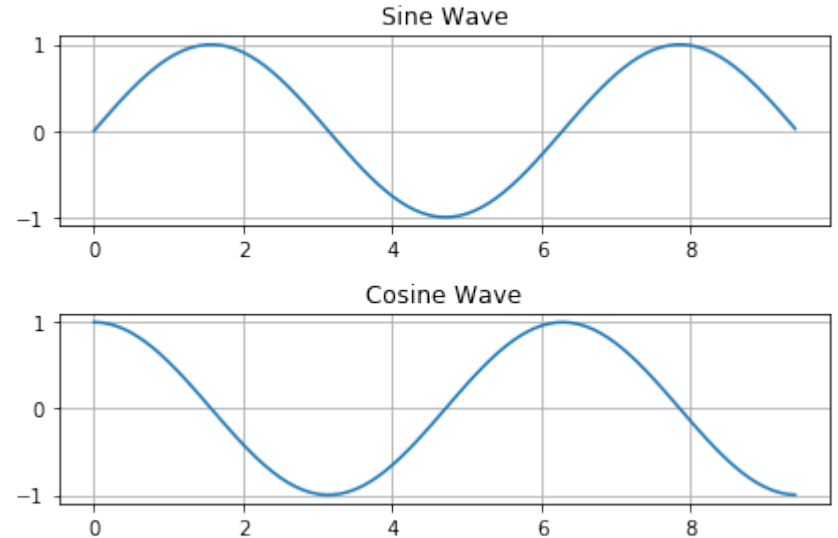
```
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Setup grid with height 2 and col 1.
# Plot the 1st subplot
plt.subplot(2, 1, 1)

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

# Now plot on the 2nd subplot
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine Wave')

plt.grid()
plt.tight_layout()
```

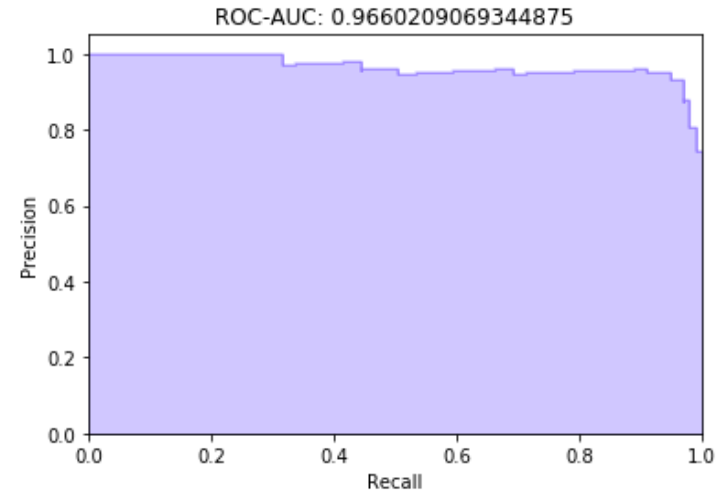


# Plot area under curve

```
def prec_rec_curve(model, X, Y_true, title="", verbose=False):
    probas_pred = model.predict_proba(X)[: , 1]
    pos_label = 1.0
    precision, recall, thresholds = precision_recall_curve(Y_true,
                                                         probas_pred,
                                                         pos_label=pos_label)

    step_kwargs = ({'step': 'post'}
                   if 'step' in signature(plt.fill_between).parameters
                   else {})
    plt.step(recall, precision, color='b', alpha=0.2,
             where='post')
    plt.fill_between(recall, precision, alpha=0.2, color='b', **step_kwargs)

    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.ylim([0.0, 1.05])
    plt.xlim([0.0, 1.0])
    plt.title(title+ "ROC-AUC: {}".format(auc(recall, precision)))
    plt.show()
```



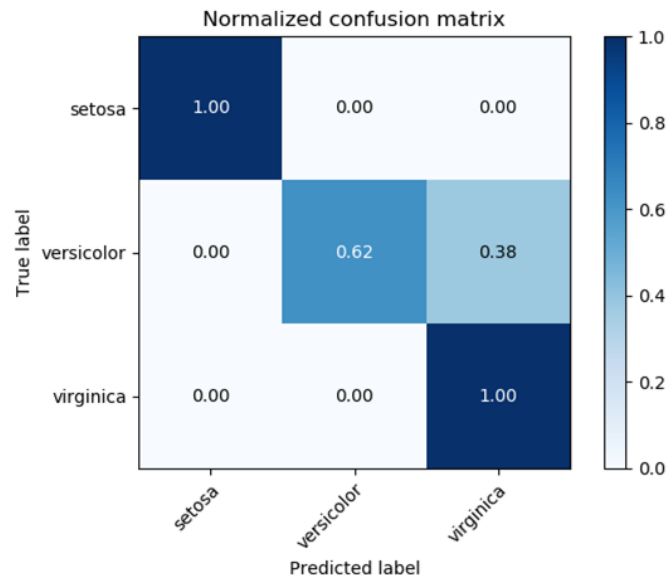
# Confusion matrix

[https://scikit-learn.org/stable/auto\\_examples/model\\_selection/plot\\_confusion\\_matrix.html](https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html)

```
fig, ax = plt.subplots()
im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
ax.figure.colorbar(im, ax=ax)
# We want to show all ticks...
ax.set(xticks=np.arange(cm.shape[1]),
       yticks=np.arange(cm.shape[0]),
       xticklabels=classes, yticklabels=classes,
       ylabel='True label', xlabel='Predicted label',
       title=title)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha='right',
         rotation_mode='anchor')

# Loop over data dimensions and create text annotations.
fmt = '.2f' if normalize else 'd'
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        ax.text(j, i, format(cm[i, j], fmt),
              ha='center', va='center',
              color="white" if cm[i, j] > thresh else "black")
fig.tight_layout()
```



Good luck on your  
HW/Project!

Questions?

# Links

[CS 231N Python Tutorial](#)

Additional slides in  
case of Q&A

# Where does my program start?

It just works

```
def do_something(number):  
    for i in number:  
        print(f'Hello {i}')  
  
do_something(5)
```

A function



Properly

```
def do_something(number):  
    for i in number:  
        print(f'Hello {i}')  
  
if __name__ == '__main__':  
    do_something(5)
```

# What is a class?

Initialize the class to  
get an **instance** using  
some parameters

Instance variable

Does something  
with the **instance**

```
class Vehicle:
    def __init__(self, make, name, year,
                 is_electric=False, price=100):
        self.name = name
        self.make = make
        self.year = year
        self.is_electric = is_electric
        self.price = price

        self.odometer = 0

    def drive(self, distance):
        self.odometer += distance

    def compute_price(self):
        if self.is_electric:
            price = self.price / (self.odometer * 0.8)
        else:
            price = self.price / self.odometer
        return price
```



# To use a class

Instantiate a class,  
get an instance

Call an instance method

```
if __name__ == '__main__':  
    family_car = Vehicle('Honda', 'Accord', '2019',  
                        price=10000)  
    print(family_car.compute_price())  
    family_car.drive(100)  
    print(family_car.compute_price())
```

# String manipulation

## Formatting

```
stripped = '  I love CS229! '.strip()
```

```
upper_case = 'i love cs 229! '.upper()
```

```
capitalized = 'i love cs 229! '.capitalize()
```

## Concatenation

```
joined = 'string 1' + ' ' + 'string 2'
```

## Formatting

```
formatted = 'Formatted number {:.2F}'.format(1.2345)
```

# Basic data structures

## List

```
example_list = [1, 2, '3', 'four']
```

## Set (unordered, unique)

```
example_set = set([1, 2, '3', 'four'])
```

## Dictionary (mapping)

```
example_dictionary =
```

```
{
```

```
    '1': 'one',
```

```
    '2': 'two',
```

```
    '3': 'three'
```

```
}
```

# More on List

## 2D list

```
list_of_list = [[1,2,3], [4,5,6], [7,8,9]]
```

## List comprehension

```
initialize_a_list = [i for i in range(9)]
```

```
initialize_a_list = [i ** 2 for i in range(9)]
```

```
initialize_2d_list = [[i + j for i in range(5)] for j in range(9)]
```

## Insert/Pop

```
my_list.insert(0, 'stuff')
```

```
print(my_list.pop(0))
```

# More on List

Sort a list

```
random_list = [3,12,5,6]
```

```
sorted_list = sorted(random_list)
```

```
random_list = [(3, 'A'),(12, 'D'),(5, 'M'),(6, 'B')]
```

```
sorted_list = sorted(random_list, key=lambda x: x[1])
```

# More on Dict/Set

## Comprehension

```
my_dict = {i: i ** 2 for i in range(10)}
```

```
my_set = {i ** 2 for i in range(10)}
```

## Get dictionary keys

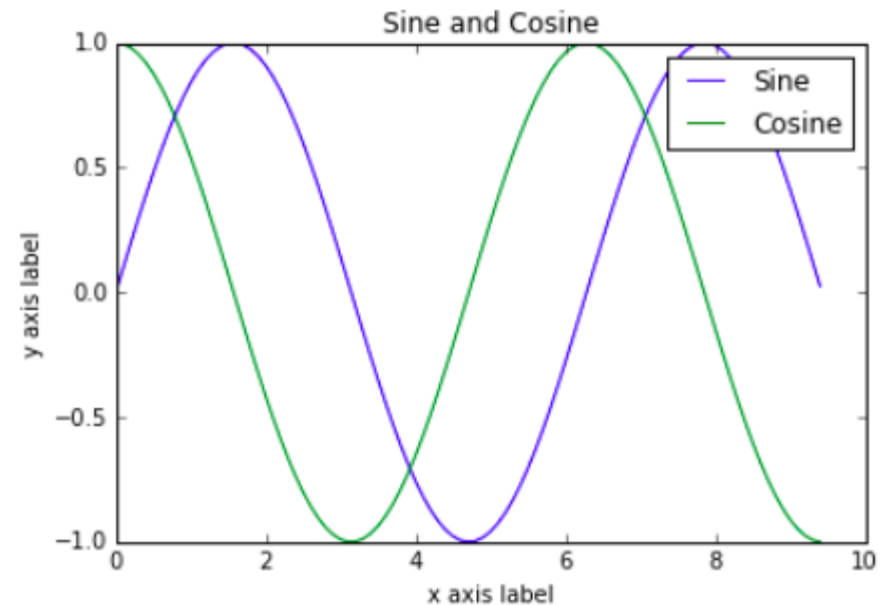
```
my_dict.keys()
```

# Another way for legend

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

# Compute the x and y coordinates for points
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()
```



# Scatter plot

```
import matplotlib.pyplot as plt
import pandas as pd

girls_grades = [89, 90, 70, 89, 100, 80, 90, 100, 80, 34]
boys_grades = [30, 29, 49, 48, 100, 48, 38, 45, 20, 30]
grades_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
plt.scatter(grades_range, girls_grades, color='r')
plt.scatter(grades_range, boys_grades, color='g')
plt.xlabel('Grades Range')
plt.ylabel('Grades Scored')
plt.show()
```

