# Python for ML

How to be the most efficient

## Topics:

- 1. Running your code (3 options)
- 2. Numpy
- 3. Python Foundations
  - a. Data Structures
  - b. Functions
- 4. Matplotlib
- 5. Python Classes

## Running Python File on Command line

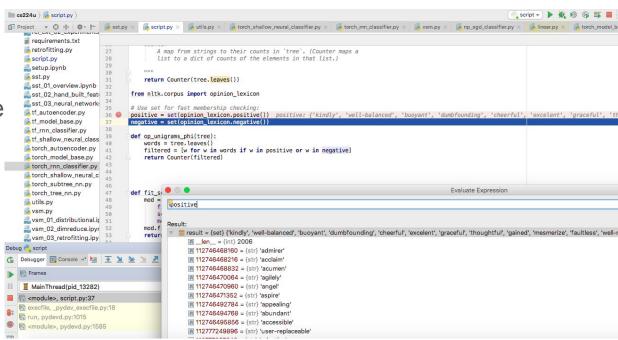
For example: 'python hellow\_world.py'

The file needs to be executing a function

```
def main(train_path, valid_path, save_path):
    """Problem 1(b): Logistic regression with Newton's Method.
    Args:
        train_path: Path to CSV file containing dataset for training.
        valid path: Path to CSV file containing dataset for validation.
        save path: Path to save predictions using np.savetxt().
    x train, y train = util.load dataset(train path, add intercept=True)
    # *** START CODE HERE ****
   # Train a logistic regression classifier
    # Plot decision boundary on top of validation set set
    # Use np.savetxt to save predictions on eval set to save path
    # *oko* END CODE HERE *oko*
```

## Running on Pycharm IDE

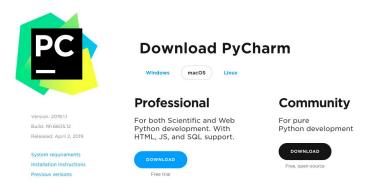
- 1. Debugger
- 2. Move around quickly
- 3. Profile code performance



## Pycharm Installation

#### Installation:

- 1. https://www.jetbrains.com/pycharm/download/#section=mac
- 2. Get Community (free)

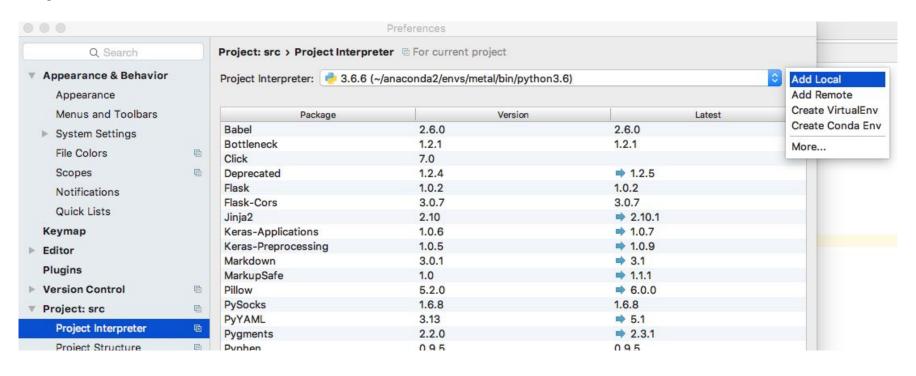


- 3. For Professional, you can get a free student license:
  - a. <a href="https://www.jetbrains.com/student/">https://www.jetbrains.com/student/</a>

## (Optional) Create conda python environment

- Create environment
  - o Install conda: <a href="https://www.anaconda.com/distribution/">https://www.anaconda.com/distribution/</a>
  - 'conda create --name python3'
- Activate environment:
  - 'source activate python3'

## Pycharm setup local environment



## Pycharm Main Hotkeys to Know

- Cmd + b: jump into function
- Cmd + [ : go forward (kind of like browsers)
- Cmd + ] : go backward

## Jupyter Notebooks

- Notebooks are usually for experimentation
- https://jupyter.readthedocs.io/en/latest/install.html
- Run command 'jupyter notebook'

```
M In [12]:
            1 import re
            2 import pandas as pd
            3 pd.set option('max colwidth', 800)
            4 from multiprocessing.dummy import Pool as ThreadPool
               p = ThreadPool(5)
 In [2]:
           1 ALL SENTENCES PATH = "./finance sentences {}.csv"
 In [3]:
              all sentences = pd.concat([pd.read csv(ALL SENTENCES PATH.format(0)),
                              pd.read csv(ALL SENTENCES PATH.format(1)),
                              pd.read csv(ALL SENTENCES PATH.format(2)),
                              pd.read csv(ALL SENTENCES PATH.format(3)),
                              pd.read_csv(ALL_SENTENCES_PATH.format(4))], axis=0)
              all sentences = all sentences.dropna()
 In [4]:
              all sentences. shape
 Out[4]: (4064072, 3)
 In [5]: 1 texts = list(all sentences['processed sentence'].astype('str'))
```

## Debugging with Pdb

- 'import pdb; pdb.set\_trace()' to stop the program and break into the debugger
- https://realpython.com/python-debugging-pdb/

## Numpy

- General advice: do not just transpose matrices until dimensions match for a matrix multiplication, it will cause you more headaches than learning matrix multiplication:)
- Check matrix shapes using matrix.shape
- Using vectors of shape (n,) can cause errors. It's better to reshape them to (n,1) or (1,n)

## **Numpy Arrays**

```
import numpy as np
a = np.array([1, 2, 3]) # Create a rank 1 array
                # Prints "<class 'numpy.ndarray'>"
print(type(a))
               # Prints "(3,)"
print(a.shape)
print(a[0], a[1], a[2]) # Prints "1 2 3"
a[0] = 5
                        # Change an element of the array
                        # Prints "[5, 2, 31"
print(a)
                                 # Create a rank 2 array
b = np.array([[1,2,3],[4,5,6]])
print(b.shape)
                                 # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0]) # Prints "1 2 4"
```

## **Array Initialization**

```
import numpy as np
a = np.zeros((2,2)) # Create an array of all zeros
print(a)
        # Prints "[[ 0. 0.1
                  # [ 0. 0.11"
b = np.ones((1,2)) # Create an array of all ones
        # Prints "[[ 1. 1.]]"
print(b)
c = np.full((2,2), 7) \# Create a constant array
print(c) # Prints "[[ 7. 7.]
                   # [7. 7.11"
d = np.eye(2)  # Create a 2x2 identity matrix
print(d)
                  # Prints "[[ 1. 0.]
                   # [ 0. 1.11"
e = np.random.random((2,2)) # Create an array filled with random values
                        # Might print "[[ 0.91940167 0.08143941]
print(e)
                                [ 0.68744134  0.8723668711"
```

## **Array Indexing**

```
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"
```

- Replaces loops
- Makes your code much much faster
- Can cause hidden errors that are hard to debug (be careful at the beginning)

```
import numpy as np
# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
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) # Create an empty matrix with the same shape as x
# Add the vector v to each row of the matrix x with an explicit loop
for i in range(4):
   y[i, :] = x[i, :] + v
# Now y is the following
# [[ 2 2 4]
# [5 5 7]
# [8 8 10]
# [11 11 1311
print(y)
```

```
# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = x + v # Add v to each row of x using broadcasting
print(y) # Prints "[[ 2 2 4]
                    [ 5 5 7]
                    188101
                   [11 11 13]]"
```

```
X = np.array([1, 2, 3, 4])
X + 1
>> array([2, 3, 4, 5])
```

## Broadcasting Error (remember to check shapes!)

```
x = np.array([1,2,3]).reshape(3, 1)
   y = np.array([1,2,3])
    x.shape
(3, 1)
    y.shape
(3,)
 1 \times y
array([[2, 3, 4],
       [3, 4, 5],
```

## Other Numpy Commands

- np.reshape
- np.multiply
- np.random
- np.linalg (norms, etc)

## Python Foundations

## **Mathematical Operations**

```
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"
```

## **Logical Operations**

```
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"
```

#### **Functions**

```
def sign(x):
    if x > 0:
        return 'positive'
    elif x < 0:
        return 'negative'
    else:
        return 'zero'
```

#### Lists

```
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']"
```

## List Comprehensions

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

You can make this code simpler using a list comprehension:

```
nums = [0, 1, 2, 3, 4]
squares = [x ** 2 for x in nums]
print(squares) # Prints [0, 1, 4, 9, 16]
```

#### **Dictionaries**

```
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"
del d['fish'] # Remove an element from a dictionary
print(d.get('fish', 'N/A')) # "fish" is no longer a key; prints "N/A"
```

#### Sets

```
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) # Prints "True"
print(len(animals))
                          # Number of elements in a set; prints "3"
animals.add('cat')
                          # Adding an element that is already in the set does n
                         # Prints "3"
print(len(animals))
                         # Remove an element from a set
animals.remove('cat')
                          # Prints "2"
print(len(animals))
```

## **Tuples**

```
d = {(x, x + 1): x for x in range(10)} # Creat
t = (5, 6)  # Create a tuple
print(type(t)) # Prints "<class 'tuple'>"
print(d[t]) # Prints "5"
print(d[(1, 2)]) # Prints "1"
```

## Matplotlib

Full tutorial:

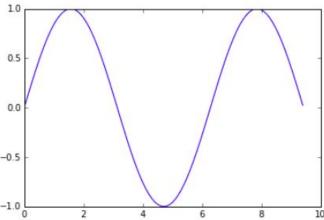
https://matplotlib.org/tutorials/introductory/pyplot.html#sphx-glr-tutorials-introductory-pyplot-py

## Matplotlib Function Plot

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

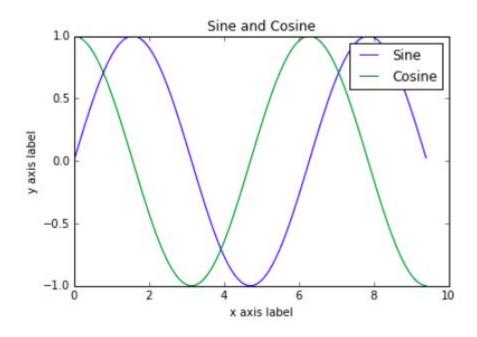
# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

# Plot the points using matplotlib
plt.plot(x, y)
plt.show() # You must call plt.show() to make graphics appear -1.0
2
```



## Matplotlib Function Plot

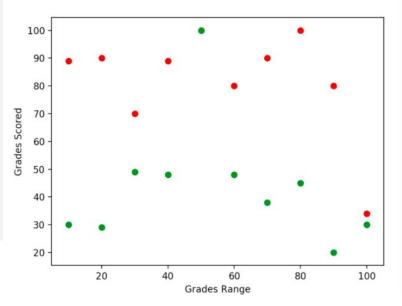
```
import numpy as np
import matplotlib.pyplot as plt
# Compute the x and y coordinates for po
x = np.arange(0, 3 * np.pi, 0.1)
y \sin = np.sin(x)
y cos = np.cos(x)
# Plot the points using matplotlib
plt.plot(x, y sin)
plt.plot(x, y cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
plt.show()
```



## Matplotlib Scatter Plot

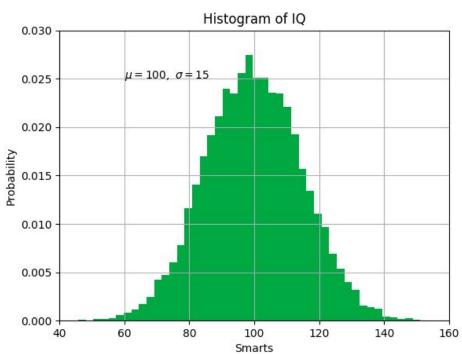
```
import matplotlib.pyplot as plt
import pandas as pd

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()
```



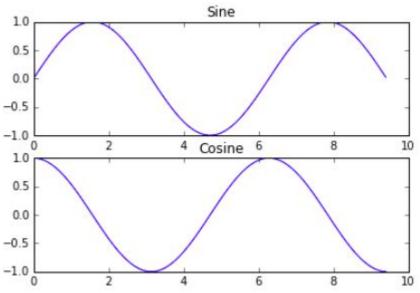
## Histograms

```
import numpy as np
import matplotlib.pyplot as plt
# Fixing random state for reproducibility
np.random.seed(19680801)
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)
# the histogram of the data
n, bins, patches = plt.hist(x, 50, density=True, facecolor='g
plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(60, .025, r'$\mu=100,\\sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()
```



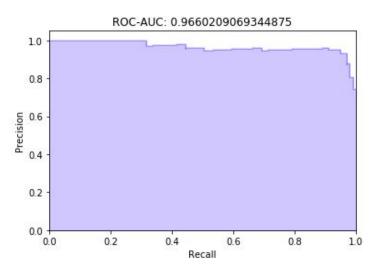
## Matplotlib Subplots

```
x = np.arange(0, 3 * np.pi, 0.1)
y \sin = np.sin(x)
y \cos = np.\cos(x)
# Set up a subplot 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 p
plt.subplot(2, 1, 2)
plt.plot(x, y cos)
plt.title('Cosine')
# Show the figure.
plt.show()
```



#### Precision-Recall Curves

```
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 kwarqs = ({'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()
```



## Other Libraries for Plotting (optional)

- Seaborn
- Plotly

## Python Classes

# Class example

```
class Customer(object):
    """A customer of ABC Bank with a checking account. Customers have the
    following properties:
    Attributes:
       name: A string representing the customer's name.
       balance: A float tracking the current balance of the customer's account.
    ....
    def __init__(self, name, balance=0.0):
        """Return a Customer object whose name is *name* and starting
       balance is *balance*."""
        self.name = name
        self.balance = balance
    def withdraw(self, amount):
        """Return the balance remaining after withdrawing *amount*
       dollars."""
       if amount > self.balance:
            raise RuntimeError('Amount greater than available balance.')
        self.balance -= amount
        return self.balance
```

"""Return the balance remaining after depositing \*amount\*

dollars.""" self.balance += amount return self.balance

def deposit(self, amount):

## Class Instantiation

- The class Customer(object) line does not "create" the class - this is defining the "blueprint"
- To instantiate the class we call the \_\_init\_\_ method with the proper number of arguments (minus self)
- \_\_init\_\_(self, name, balance=0.0)
- mario = Customer("Mario Srouji", 1000.0) instantiates an object mario of the class Customer

## What does self mean?

- self is the instance of the class we are using
- When defining a function (method) inside of a class need to include self as first argument so we can use it
- Syntactical way to define that this particular method should be applied to the given object instance
- mario.withdraw(100.0) = Customer.withdraw(mario, 100.0)

# Class example

```
class Customer(object):
    """A customer of ABC Bank with a checking account. Customers have the
    following properties:
    Attributes:
       name: A string representing the customer's name.
        balance: A float tracking the current balance of the customer's account.
```

```
def __init__(self, name, balance=0.0):
    """Return a Customer object whose name is *name* and starting
   balance is *balance*."""
    self.name = name
    self.balance = balance
def withdraw(self, amount):
    """Return the balance remaining after withdrawing *amount*
    dollars."""
    if amount > self.balance:
```

raise RuntimeError('Amount greater than available balance.') self.balance -= amount return self.balance def deposit(self, amount): """Return the balance remaining after depositing \*amount\* dollars.""" self.balance += amount return self.balance

## (Optional)

# Inheritance once again

```
class Car(Vehicle):
    """A car for sale by Jeffco Car Dealership."""
   base_sale_price = 8000
   wheels = 4
   def vehicle_type(self):
        """Return a string representing the type of vehicle this is."""
       return 'car'
class Truck(Vehicle):
    """A truck for sale by Jeffco Car Dealership."""
   base sale price = 10000
   wheels = 4
   def vehicle_type(self):
        """Return a string representing the type of vehicle this is."""
        return 'truck'
```