Python Intro

Introduction

Data Types

NumPy

Packages

Pandas

Licensed for personal use only for Vincent Chang <vincent.chang@macys.com> from Python ML @ Macy's 04/25/2108 @ 2018-04-25

Lesson Objectives

- Learn Python language basics

Python Intro

Introduction

Data Types

NumPy

Packages

Pandas

Licensed for personal use only for Vincent Chang <vincent.chang@macys.com> from Python ML @ Macy's 04/25/2108 @ 2018-04-25

About Python

- Python is a general-purpose, object-oriented, dynamic programming language.
- Python is also **language & environment** for data science computing and graphics
- Open source
- Rich ecosystem (lots of libraries)
- Great for modeling, machine learning, ad-hoc analytics
- Used by developers, but also popular among scientists and now data scientists.

Why Python

- Comprehensive
  - Pretty much any analysis can be done in Python
- State-of-the-art graphics capabilities
  - Because picture IS worth a thousands words
- Designed for interactive analysis
  - Most analysis is done this way
  - No time consuming edit / compile / run cycle
  - can support scripting too
- Open source
  - Commercial packages costs thousands

Why Python

- Python can import from variety of formats (csv, excel, db, …)
- Python is extensible
  - Thousands of libraries in PyPi (open source)
- Python usually gets ‘bleeding edge’ routines before other commercial packages !!
  - Power of open source
- Free IDEs (Spyder, Pycharm) are available
  - Easy to use / program
- Runs on multiple platforms (Mac, Windows, Linux)

Licensed for personal use only for Vincent Chang <vincent.chang@macys.com> from Python ML @ Macy's 04/25/2108 @ 2018-04-25

Python History

- Created By Guido Von Rossom in 1991
- Designed as an alternative to ”scripting languages” like PERL
  - Fully OOP (Object Oriented Programming)
- Dynamically Typed Language
  - “Duck Typing” – if it walks like a duck..
  - Automatic type conversion
- JIT (Just in Time)
  - Code compiles and runs in real time
  - No compile – package – deploy – cycle
- REPL Shell
  - Real time shell for analysis.

Python Versus R

Python	R
General Purpose Language	Specialized for Statistics and Analytics
Object-Oriented Approach	Mixed Paradigm: Procedural/Functional
Large Package Repository: PyPI	Huge Package Repository; CRAN
Dynamic Language	Dynamic Language
Uses various editors, IDEs	Rstudio is highly integrated IDE for R
Favored by Computer Scientists	Favored By Statisticians
Summary: General Purpose Language now used widely in analytics	**Summary**: Specialized Language for Statistical Programming now used widely in analytics

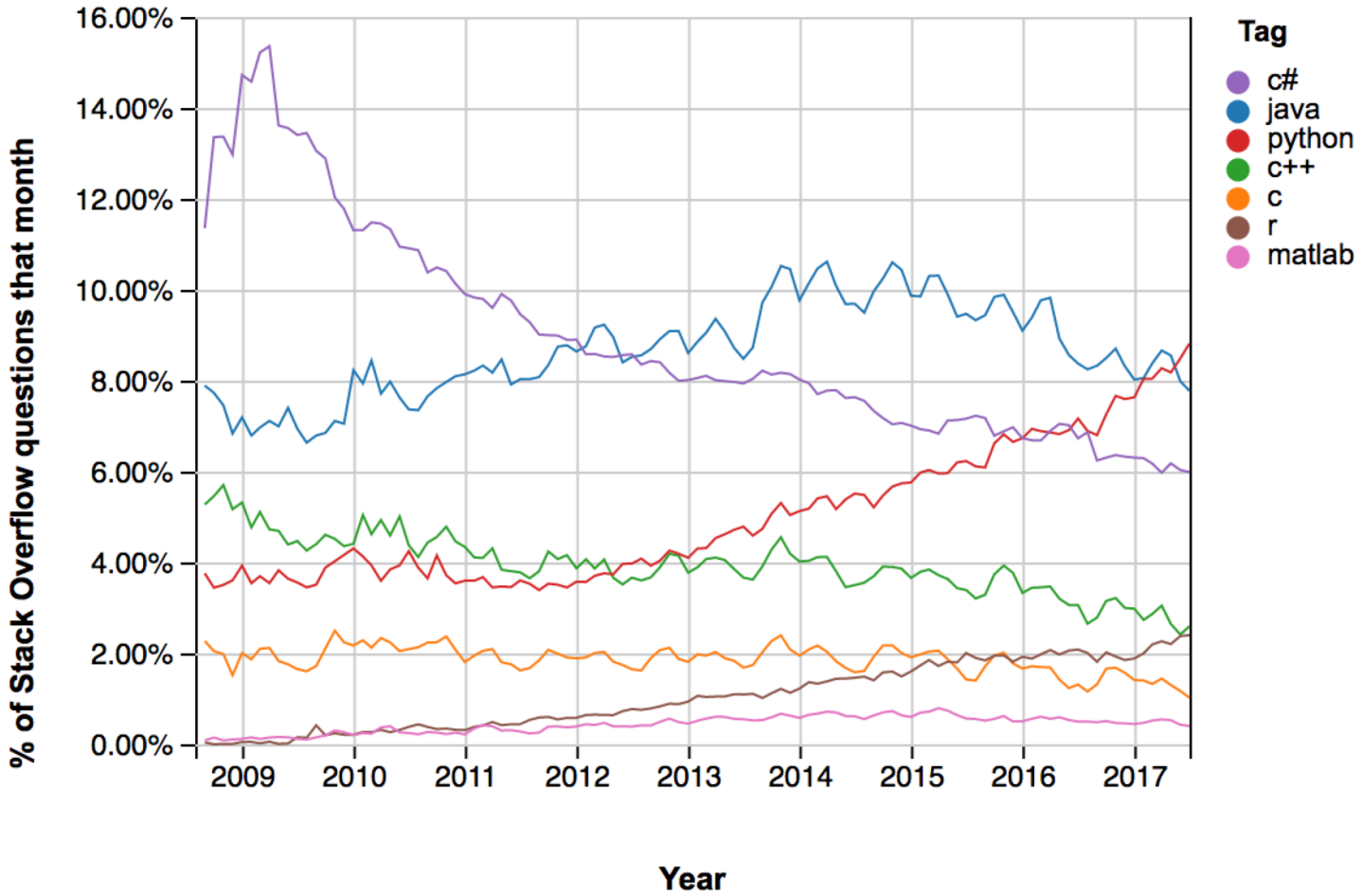
Licensed for personal use only for Vincent Chang <vincent.chang@macys.com> from Python ML @ Macy's 04/25/2108 @ 2018-04-25

Python Versus Java

Python	Java
Dynamically Typed Language	Statically Typed Language
Interactive REPL Shell	No REPL
Can’t build dependencies into object	Can build dependencies into a FAT JAR.
Good for interactive analytics	Good for “productionizing” analytics.
Relatively slow	Faster (but not as fast as native code).

Python Popularity

- Python Most popular language in 2018 on Stack Overflow
  - (Other than Javascript)



Python Use Cases

- Who uses Python?

- Python is very commonly used in the following areas:
  - Web Programming
  - Microservices
  - System Automation Tasks
  - Scientific Programming
  - Data Analysis / Data Science
  - Machine Learning / AI

Licensed for personal use only for Vincent Chang <vincent.chang@macys.com> from Python ML @ Macy's 04/25/2108 @ 2018-04-25

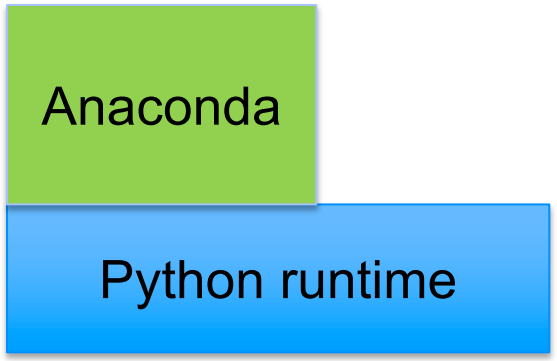
Is Python Fast?

- Pure python code is fairly slow
  - Faster than purely interpreted languages (LISP, Smalltalk, shell scripts)
  - But slower than managed languages like Java, .NET
  - And much slower than pure native code like C/C++/Fortran.
- Python does allow users to write native code
  - Python integrates with C, C++, Fortran, etc.
  - Typically performance sensitive code is written in native code (usually C/C++)
  - Cython compiler: allows you to mix C, C++ and python code.
- Native code is Fast!
- Write Native code (C/C++) for performance sensitive parts,
  - Everything else, pure python! Much easier.

Licensed for personal use only for Vincent Chang <vincent.chang@macys.com> from Python ML @ Macy's 04/25/2108 @ 2018-04-25

Installing Python

- Recommend downloading Anaconda from Continuum Analytics.
- Download Free version (Ananaconda Distribution)
- http://www.anaconda.com/donload



Why Anaconda?

- Anaconda has hundreds of the most commonly used DS packages already built!
  - No need for C/C++ compilers
  - Good for Windows Users! (Hard to build on Windows)
- Easy to Install Bundle
  - Platform Native bundling (MSI: Windows, DMG: Mac, etc)
- Commercially Available Support
  - Good for Enterprise Users
  - Easy for IT Services to “Certify” entire distribution including packages
- Separate from System Python
  - Anaconda is separate from your system python
  - So it won’t break anything else you may be doing on your machine in python.

Do I really need Anaconda?

- No, using your system python is fine (if you have one).
- Most Mac and Linux users already have Python
  - However, it may not be the latest version.
  - On linux, python 3.x is often called python3
- You may want to use ”virtualenv” to create a virtual environment for your data science work.
- You will have to download and install your own packages as-needed.

Python Console

- We can also use Python from console
- Though Spyder is a much better interface

```
$ python

# now we are in Python shell

>>>  help

# exit
>>> quit() # or control-D
```

Jupyter Notebooks

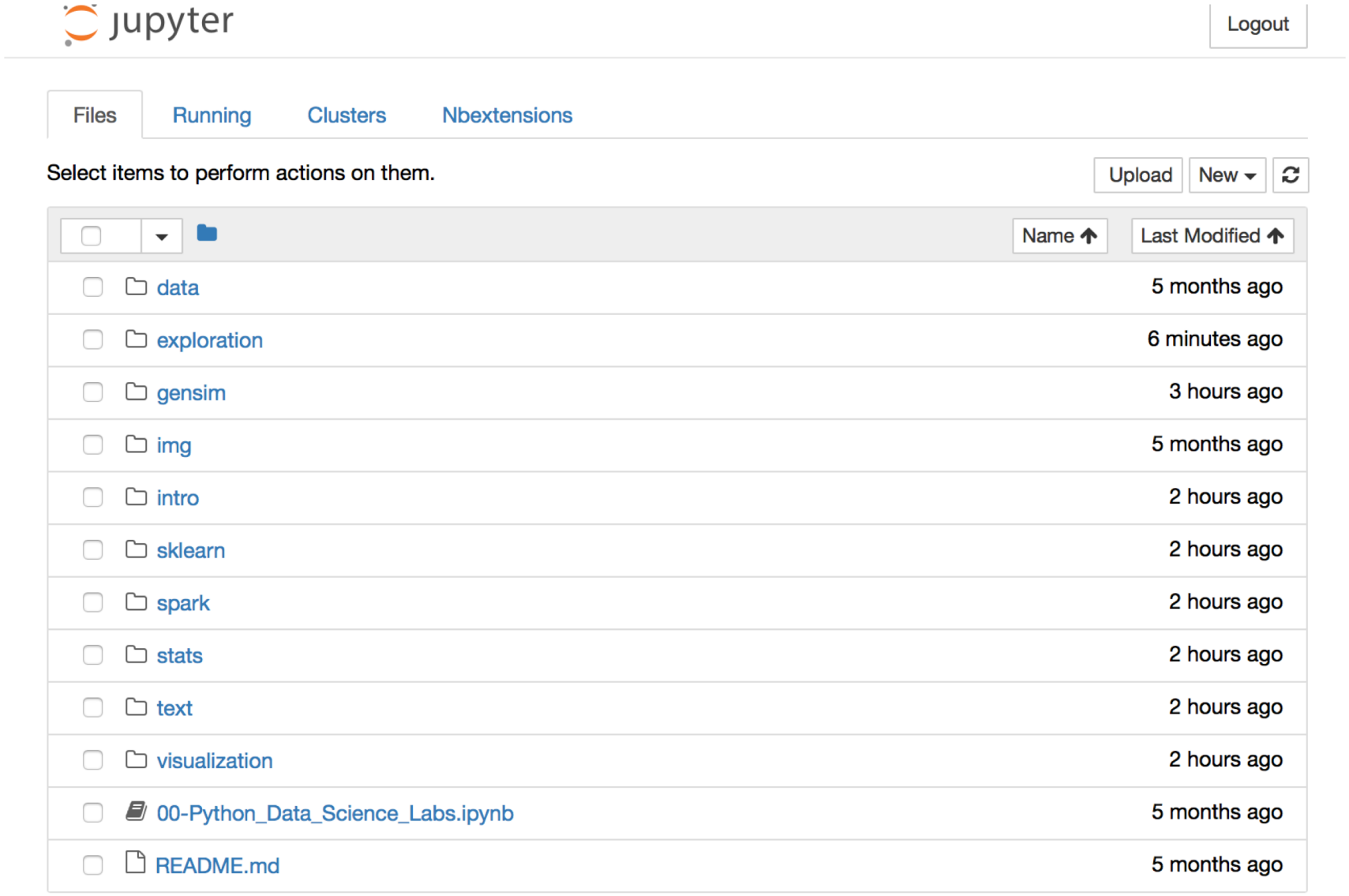
- Jupyter notebooks are a great way to showcase working python code.
- We do jupyter notebooks in this class!

Using Jupyter Notebooks

- Go to directory and type jupyter notebook

```
$ cd /path/to/my/notebooks

$ jupyter notebook

[I 22:28:59.637 NotebookApp] The Jupyter Notebook is running
at: http://localhost:8888/?token=YOURTOKEN

[I 22:28:59.637 NotebookApp] Use Control-C to stop this server
and shut down all kernels (twice to skip confirmation).

[C 22:28:59.639 NotebookApp]    Copy/paste this URL into
your browser when you connect for the first time,    to login
with a token:        http://localhost:8888/?token=YOURTOKEN
```

Browser for Jupyter Notebook



Lab: Introducing Notebooks

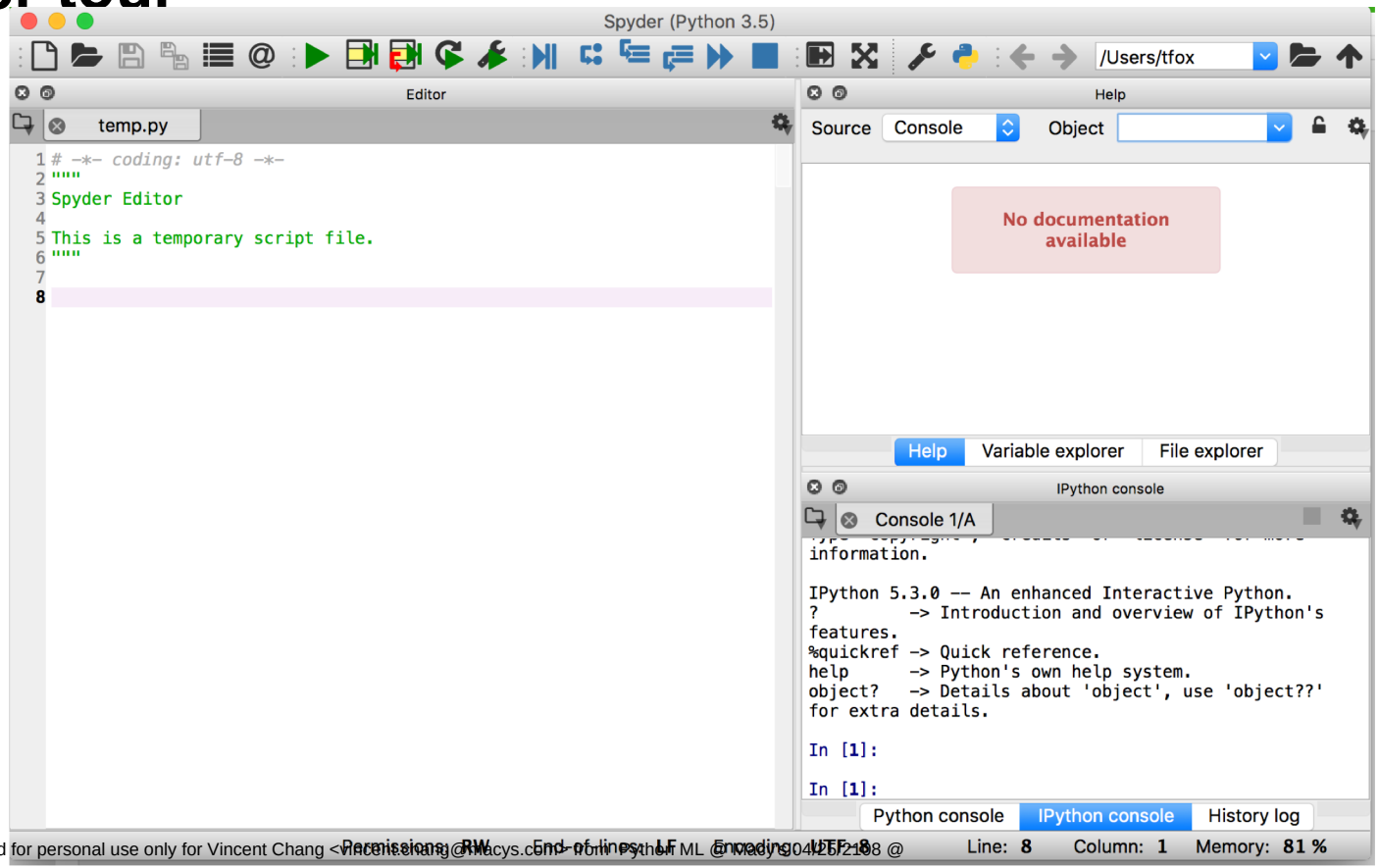
- **Overview**:
  Quick intro lab to Juptyer Notebook

- **Approximate time**:
  10 mins

- **Instructions:**
  intro / 01-LearningNotebooks.ipynb

Spyder

- A fantastic IDE for Python, available freely
- GUI
- Integrated documentation
- **A quick Spyder tour**

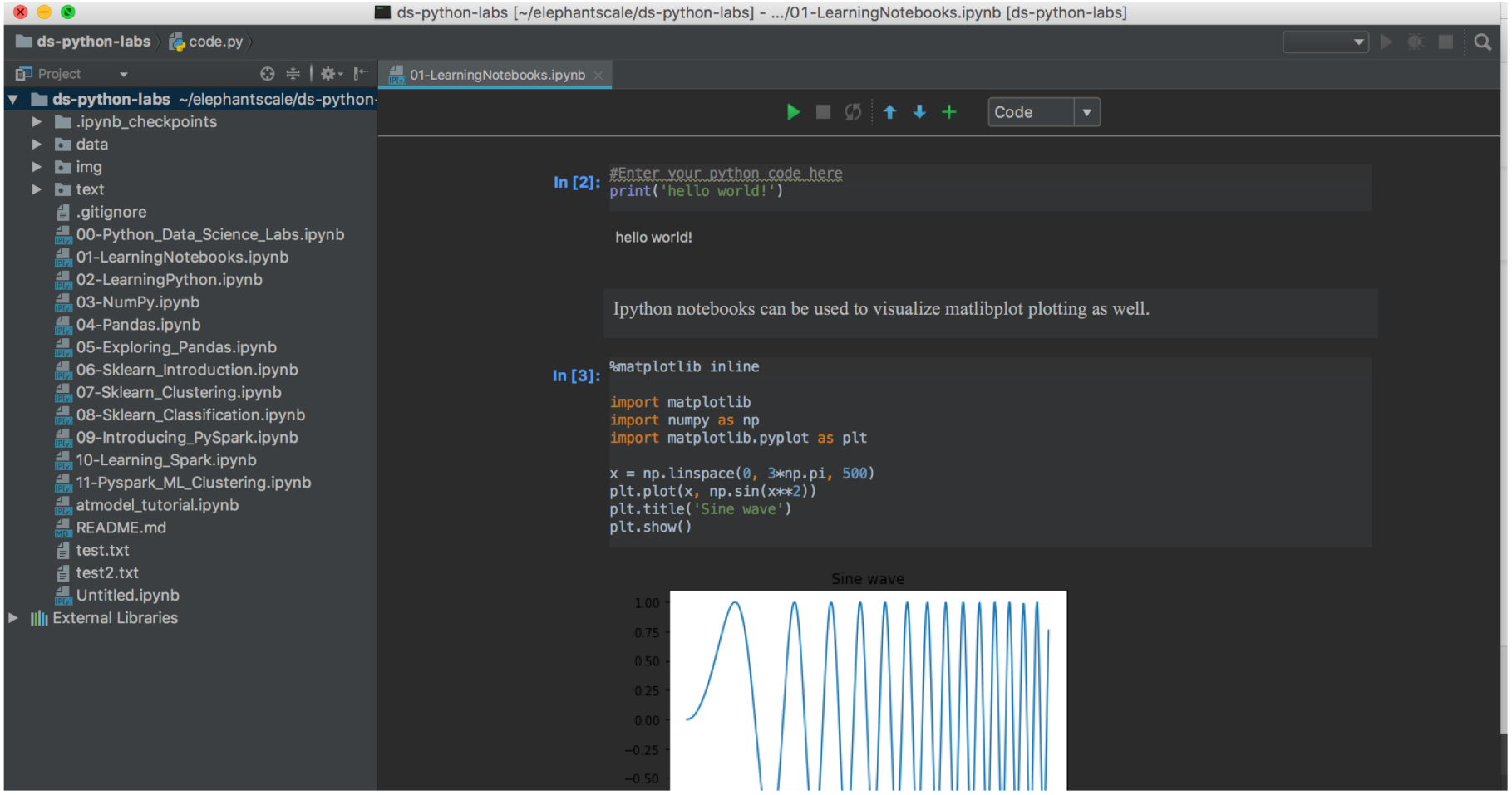


Getting Started With Python (Mini Lab)

```
# prints to screen
print("hello world")

# create variables (= and = are equivalent)
a = 10
b = 3.1
c = "hello world"
```

Other IDEs

- There are many other IDEs:
- Example: PyCharm from JetBrains



Python Data Types

Introduction

Data Types

NumPy

Packages

Pandas

Licensed for personal use only for Vincent Chang <vincent.chang@macys.com> from Python ML @ Macy's 04/25/2108 @ 2018-04-25

Python Operators (Assignments / Boolean)

operation	operator	Example
Assignment	=	a = 10
Logical		
And	and	a and b
Or	or	a or b
Not	not	not a

Python Operators: Arithmetic

Operation	operator	Example
Addition	+	a + b
Subtraction	-	a - b
Multiply	*	a * b
Division	/	a / b
Exponent	**	a ** 2
modulus	%	5 % 2 is 1
Integer division	//	5 // 2 is 2

Python Operators: Bitwise

Operation	operator	Example
Bitwise And	&	a & b
Bitwise Or	|	a | b
Bitwise Xor	^	a ^ b
Bitwise 1's complement	~	~a
Logical shift left	<< n	a << 2
Logical Shift Right	>> n	a >> 2

Licensed for personal use only for Vincent Chang <vincent.chang@macys.com> from Python ML @ Macy's 04/25/2108 @ 2018-04-25

Python Operators: Comparison

Comparison	operator	Example
less	<	a < b
Greater	>	a > b
Less than equal	<=	a <= b
Greater than equal	>=	a >= b
Equal	==	a == b
Not equal	!=	a != b

Python Data Types

- Scalar
  - float (3.1)
  - int (10)
  - long (10L)
  - complex (1+3j)
  - bool (True / False)
  - str (“hello world”)

- Complex
  - Lists [1, 2, 3]
  - Tuples (1,”hello”)
  - Dictionaries {“cat”: 3, “dog”, 4}

Python Scalar Types

- Numeric Types
  - int, long, float, complex
  - **x = 1.3 #float**
- int
  - **int()**
  - **y = int(x)**
- bool
  - **True / False**
- str
  - **x = “hello world”**

Lists

- List are an ordered collection of objects
- Can mix and match types (unlike NumPy Arrays)

```
a = [1, “abc”, 3.1]

a[1]
"abc”
```

Dictionaries

- Dictionaries are groups of key-value pairs
- Objects can be of ANY type (mix and match)

```
>>> a = {“one”: 1, “two”: 2, “three” : 3}

>>> a[“one”]
1
```

Named Items in Dictionaries

- Here we are creating a list with key=value (or name=value) pairs

```
>>> stock = {"symbol”:"goog",
             "ask”:100.1,
              "sell”:100.5}
>>> stock
{'sell': 100.6, 'symbol': 'GOOG', 'ask': 100.1}

# refer data by symbols / names
>>> stock[‘ask’]
100.1
```

Dictionaries : Adding New Attributes

```
> stock = {'symbol':'GOOG', 'ask':100.1, 'sell':100.6}
# Add a new attribute using []
> stock['company name'] = 'Google'
> stock['exchange'] = 'NASDAQ'

> stock
{'Company Name': 'Google', 'sell': 100.6, 'symbol':
'GOOG', 'ask': 100.1, 'Exchange': 'NASDAQ'}
```

Dictionaries

- You can think of a dictionary as a key-value pairs.
- Accessing list elements
  - By name (most common). e.g. **stock['price']**
- **Dictionaries are *IMPORTANT* in Python**
- Lots of Python functions return Dictionaries as a result.
  - Need to know how to navigate the dictionaries.

Tuples

- Tuples are groups of data
- How is a tuple different from a list?
  - Tuples are immutable (can't be changed in any way)
  - Tuples aren't able to be iterated over.
  - Tuples are fixed length

Arrays

- Python has no native array functionality
- List will serve in a pinch, but is very S-L-O-W
- The package numpy introduces a native array,
- We will discuss shortly.

Ranges

- range(5) creates a sequence from 0 to 4
- **range(start, stop step)** function is more flexible
  - **range**(1,10, 2) => 1,3,5,7,9
  - **range**(0, 0.1, 0.01) => 0.0, 0.01, …, 0.10

- More options: **?range**

Control Loops


```
# for loop
for (x in range(10)):
    print(x)


# while loop
x = 10
while (x >= 0):
    print(x)
    x = x - 1
```

Control Flow: IF-Else

```
if (condition or expression):
  statements
else:
  statements


# if .. else
grade = ‘pass' if (score > 50) else ‘fail

# similar to Java / C :
      x > 50 ? “pass” : “fail”
‘
```

User Functions

- Easy to write custom user functions and extend Python.
- Return type can by any type : scalar / object / NULL
- If end of function is reached without explicit return,
- the value of last evaluated expression is returned.

```
def function (arg1, arg2, …. ):
  statements
  return(result)  #don't need to say return
```

User Function Example

```
def my_max(x,y):
    if (x > y)
        return x
     else
        return y
```

Lambda Functions

- A Lambda is an anonymous function.
- Usually used as an argument to another function

```
lambda x : x * x #generate square
```

Map

- Performing an operation on a Array / matrix… Should I use a loop to go through?

- Nope, use map.

```
a = [1, 2, 3, 4, 5]
a.map(lambda x: x * 2)
```

- Returns:

```
[2, 4, 6, 8, 10]
```

Lab: Intro

- **Overview**:
  Quick intro lab to Python

- **Approximate time**:
  10 mins

- **Instructions:**
  01-intro / 02-Introduction

NumPy

Introduction

Data Types

NumPy

Packages

Pandas

Licensed for personal use only for Vincent Chang <vincent.chang@macys.com> from Python ML @ Macy's 04/25/2108 @ 2018-04-25

About NumPy and Python

- Python is a rich language with ecosystem.
- Open source
- Rich package ecosystem (lots of libraries)
- Great for modeling, machine learning, ad-hoc analytics
- Used by scientists, now very popular among data scientists / analysts

Why NumPy

- Fast
  - Numpy does numeric computation in native code.
  - (Fast C++)
- Full Featured
  - Does many types of linear algebra
  - Matrix Manipulation
- Helps Support More Advanced Analytics
  - Pretty much all subsequent analytics built on top of numpy

Licensed for personal use only for Vincent Chang <vincent.chang@macys.com> from Python ML @ Macy's 04/25/2108 @ 2018-04-25

Numpy and SciPy

- SciPy is a companion package to NumPy
  - Designed for Scientific computing
  - Built on top of NumPy
- Has a number of useful things.

Lists in Python

- Python has a native type of sequence: the list.
  - Good for storing small data
  - Not good for multi-dimensional data.
  - Very slow ”at scale”

```
# A Simple List
>>> a = [1,2,3,4]
>>> print(a[2:3]
[2,3]

# Concatenating Lists
>>> b = [5,6,7,8]
>>> a + b
[1, 2, 3, 4, 5, 6, 7, 8
```

Enter the NdArray

- Numpy Arrays are fast:
  - Native Code (C++)
  - Fast vectorized operations.
- Homogenously typed (usually numeric types: int64, float64, etc)

```
# An ndarray
>>> a = np.array([1,2,3,4])
>>> a
array[1, 2, 3, 4]
>>> a.dtype # What is the type of a?
dtype('int64')
```

Array Types

- Arrays have types: (np.int64, np.float64, np.complex, etc)
- Types can be inferred

```
# An ndarray of integers (implicit)
>>> a = np.array([1,2,3,4])
>>> a
array[1, 2, 3, 4]
>>> a.dtype # What is the type of a?
dtype('int64')
```

- Types can also be specified

```
# An ndarray of floats (explicit)
>>> a = np.array([1,2,3,4], dtype=np.float64)
>>> a
array[1., 2., 3., 4,]
>>> a.dtype # What is the type of a?
dtype('float64')
```

Lists to ndarray

- We can convert plain Python lists to ndarray:
  - Result will be possibly nested (if multidimensional)
  - Single dimensional ndarrays (vectors) will be non-nested.

```
>>> import numpy as np
>>> a = [1,2,3,4] #List
>>> a_array = np.array(a) # Convert to ndarray
>>> a_list = a_array.tolist() # Back to list again
```

Multidimensional Arrays

- Arrays can be multidimensional

```
>>> import numpy as np

>>> a = np.arange(15).reshape(3,5)
>>> a
array[[ 0,  1,  2,  3,  4],
      [ 5,  6,  7,  8,  9],
      [10, 11, 12, 13, 14]]
>>> a.shape # Shows shape of array
(3,5)
>>> a.ndim # Number of Dimensions (Rank)
2
```

Creating Arrays

- There are many ways to create arrays:
- Convert from list: np.array(mylist)
- Pre-initialized with np.zeroes or np.ones – type float64

```
>>> np.zeroes((3,2)) #float64 by default
array([[0., 0., 0.]
       [0., 0., 0.]]
>>> np.ones((3,2))
array([[1., 1., 1.]
       [1., 1., 1.]]
```

- Range with np.arange

```
>>> np.arange(10, 30, 5) #use for int types only
array([10,15,20,25])
```

- Linspace (n numbers from a to b)

```
>>> np.linspace(0, 2, 9) #better for float types
array([0., 0.25, 0.5, 0.75, 1., 1.25, 1.5, 1.75, 2.])
```

Elementwise operation

- Basic arithmetic (+,-,* /) is performed elementwise (on arrays)

```
# Elementwise operation
>>> a = np.array([1,2,3,4])
>>> b = np.array([5,6,7,8])
array[1, 2, 3, 4]
>>> a + b
array[6, 8, 10, 12]
```

Upcasting

- When mixing types on arrays, results are "upcasted"
  - Example int -> float -> complex
- Examples

```
# An ndarray of integers (implicit)
>>> a = np.array([1,2,3,4])
>>> a
array[1, 2, 3, 4]
>>> a * 2.5  #result is floating point
array[2.5, 5., 7.5, 10.]
```

Broadcasting

- What is arrays have different shapes?
- **Broadcasting** allows arrays to be extended for elementwise operation **in some cases**
- Compatibility Scenarios:
  - 2 Arrays exactly same shape: perform elementwise
  - Array operated on scalar value (perform elementwise on scalar)
  - Arrays with same number of elements OR single element in matching dimensions

Licensed for personal use only for Vincent Chang <vincent.chang@macys.com> from Python ML @ Macy's 04/25/2108 @ 2018-04-25

Indexing

- Single Dimensions can be indexed as follows:

```
>>> a = np.array([1,2,3,4])
>>> a[1] # zero-based index
2
```

- Multiple Dimensions: use array[m,n] syntax

```
>>> a = np.arange(15).reshape(3,5)
>>> a
array[[ 0,  1,  2,  3,  4],
      [ 5,  6,  7,  8,  9],
      [10, 11, 12, 13, 14]]
>>> a[1][0] # DON'T DO THIS!! -- Slow
5
>>> a[1,0] #  This is faster
5
```

Slicing

- Slice by array[start:stop:step]

```
>>> a = np.array([1,2,3,4])
>>> a[2:3] # zero-based index
[3 4]
>>> a[0:3:2] #skipping by 2
[1 3]
```

- Multidimensional array: (separate slices by commas)

```
>>> a = np.arange(15).reshape(3,5)
>>> a
array[[ 0,  1,  2,  3,  4],
      [ 5,  6,  7,  8,  9],
      [10, 11, 12, 13, 14]]
>>> a[0:1,1:2]
[[1 2]
 [6 7]]
```

Arrays

- One-dimensional arrays
- Holds ordered collection of objects
  - Objects all have to be of the SAME TYPE (no mix-match)
- Arrays are created using np.array() function
  - **v1 = np.array([1,2,3,4,5])**
- Access elements from Array using indexes
  - Indexes start with 1
    (different from java/c where indexes start at 0)
  - **v1[3] => 3**
  - **v1[np.array([1,3])] => 1,3**
  - **v1[2:4] => 2,3,4**

Numpy Array Operations

- Arrays can be operated on just like first class variables

```
import numpy as np
v1 = np.array([1,2,3,4])
v2 = np.array([10,20,30,40])

v1 + v2
[11, 22, 33, 44]

v2 - v1
[9, 18, 27, 36]

v1 * 3
[3,  6,  9, 12]
```

Numpy Array Operations

```
x = np.array([1,2,3,4,5)

# Length: size of Array
length(x) => 5



# Statistics: min, max, avg, mean, median,
standard deviation, variance
np.mean(x) =>  3
np.median(x) => 3
np.std(x)  =>  1.58
np.var(x) => 2.5
```

Numpy Operations

```
# Sorting
sort(np.array([1,3,5,4,2))
 1, 2, 3, 4, 5

# Reverse
np.array([1,3,5,4,2])[::-1]
2, 4, 5, 3, 1

# Adding Elements
v1 = np.array([1,2,3,4])
v3 = v1.append([5,6,7])
v3
[1, 2, 3, 4, 5, 6, 7]

#Removing elements
V3.delete[2:4]
[1, 5, 6, 7]
```

Numpy Array Filtering (*Important*)

```
# Filtering:
x = np.array([1,-2, 3, -4, 5, -1])
x > 0
[TRUE, FALSE, TRUE, FALSE, TRUE, FALSE]

# Used as subscript, only elements matching
TRUE are retained
x [ x>0 ]
[ 1, 3, 5 ]

** VERY IMPORTANT **
```

Matrix

- A matrix is a 2-D array.
- Contains elements of the SAME type.

```
matrix( np.array(['a', 'b', 'c', 'd', 'e', 'f'),
                 [2,3])
```

- Which one the following is correct?

	C 1	C 2	C 3
Python 1	a	b	c
Python 2	d	e	f

	C 1	C 2	C 3
Python 1	a	c	e
Python 2	b	d	f

Matrix Manipulation

- Numpy can represent matrices 2 ways:
  - As 2-D ndarrays (**preferred for Python > 3.5**)
  - np.mat class (subclassed from ndarray) (Python 2.x)
- Main difference is matrix multiply syntax

```
>>> # Arrays in Python > 3.5
>>> a = np.array([[4, 3], [2, 1]])
>>> b = np.array([[1, 2], [3,4]])
>>> print (a@b) # Matrix Mutiply is '@' in Python > 3.5
[[13 20]
 [ 5  8]]
```

```
>>> # Matrices
>>> a = np.mat('4 3; 2 1')
>>> b = np.mat('1 2; 3 4')
>>> print (a*b) # Matrix Mutiply
[[13 20]
 [ 5  8]]
```

Identity Matrix

- Create a basic identity matrix with np.identity(rank)

```
>>> np.identity(4)
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

- Create custom diagonals with np.eye(n, m, k)

```
>>> np.eye(5, 8, 1) #Shifted right by 1
array([[0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0.]])
```

Sparse Matrices

- The scipy.sparse package has a sparse matrix
- Great for times where we have a large matrix
  - With few items entered.

```
>>> mtx = sparse.csr_matrix((3, 4), dtype=np.int8)
>>> mtx.todense()  # Output is a matrix, not 2-D array
Matrix([0,0,0,0],
       [0,0,0,0]
       [0,0,0,0]])
```

- Sparse.csr_matrix: row oriented matrix
- Sparse.csc_matrix: column oriented matrix.

Entering values for CSR/CSC matrix

- Here is how we enter the values:
- We pass in a tuple of the following:
  - The raw data
  - The indices in the matrix that are to be nonzero
  - The corresponding pointers to the raw data.

```
>>> data = np.array([1,2,3])
>>> rows = np.array([2,4,6])
>>> ptrs = np.array([2,0,1])
>>> mtx = sparse.csr_matrix(data,rows, ptrs),
shape=(3,4))
>>> mtx.todense()
Matrix([0,0,3,0],
       [1,0,2,0]
       [0,0,0,0]])
```

Lab: Numpy

- **Overview**:
  Use NumPy

- **Approximate time**:
  5 mins

- **Instructions:**
  01-intro / 03-numpy.ipynb

Packages

Introduction

Data Types

NumPy

Packages

Pandas

Licensed for personal use only for Vincent Chang <vincent.chang@macys.com> from Python ML @ Macy's 04/25/2108 @ 2018-04-25

Python Libraries

- Libraries are Python are called ‘packages.’
- Published @ PyPI (Python Package Index)
- Thousands of packages (most open source)
  - 5000 + packages !!

- Installing a package:

```
$ pip install ”package name”
```

- In Spyder check ‘package’ section

- Using a package (importing)
  - `import package-name as package-alias`
  - `import numpy as np`

User Functions

- Easy to write custom user functions and extend Python.
- Return type can by any type : scalar / object / NULL
- If end of function is reached without explicit return,
  - the value of last evaluated expression is returned.

```
def function (arg1, arg2, …. ):
  statements
  return(result)
```

User Function Example

```
def my_max(x,y)
    if (x > y):
        return x
    else:
        return y
```

DocStrings

- Functions can (and should) define a docstring

```
def my_max(x,y)
    """ This re-implements the max function
    """
    if (x > y):
        return x
    else :
        return y
```

```
print(my_max.__doc__)
This re-implements the max function
```

Lambda Functions

- A Lambda is an anonymous function.
- Usually used as an argument to another function

```
lambda x : x * x #generate squre
```

Map

- Performing an operation on a Array / matrix… Should I use a loop to go through?

- Nope, use map.

```
a = [1, 2, 3, 4, 5]
a.map(lambda x: x * 2)
```

Pandas

Introduction

Data Types

NumPy

Packages

Pandas

Licensed for personal use only for Vincent Chang <vincent.chang@macys.com> from Python ML @ Macy's 04/25/2108 @ 2018-04-25

Series

- Pandas has a Series type
- Think of a single column in a database table
- Must be of same type, though can also be NA (np.nan)
- Defaults to float64 (notice that it is converted!)

```
s = pd.Series([1,3,5,np.nan, 6, 8])
0  1.0
1  3.0
2  5.0
3   NaN
4   6.0
5  8.0
dtype: float64
```

Data Frame

- Data frames hold ‘tabular’ data.
- Think ‘Excel spreadsheet’ or ‘database table.’
  - Rows & columns
- Each column can be a different type.
- Each row must have same length.

X	Y	Z
1	2	3
4	5	6

```
df = pd.DataFrame(
     {‘X' : [1, 4],
      ‘Y’ : [2, 5],
      ‘Z’ : [3,6 ]}
)
```

Creating A Data Frame

City	Pop	rain
San Francisco	10	2
Seattle	15	10
Los Angeles	20	1

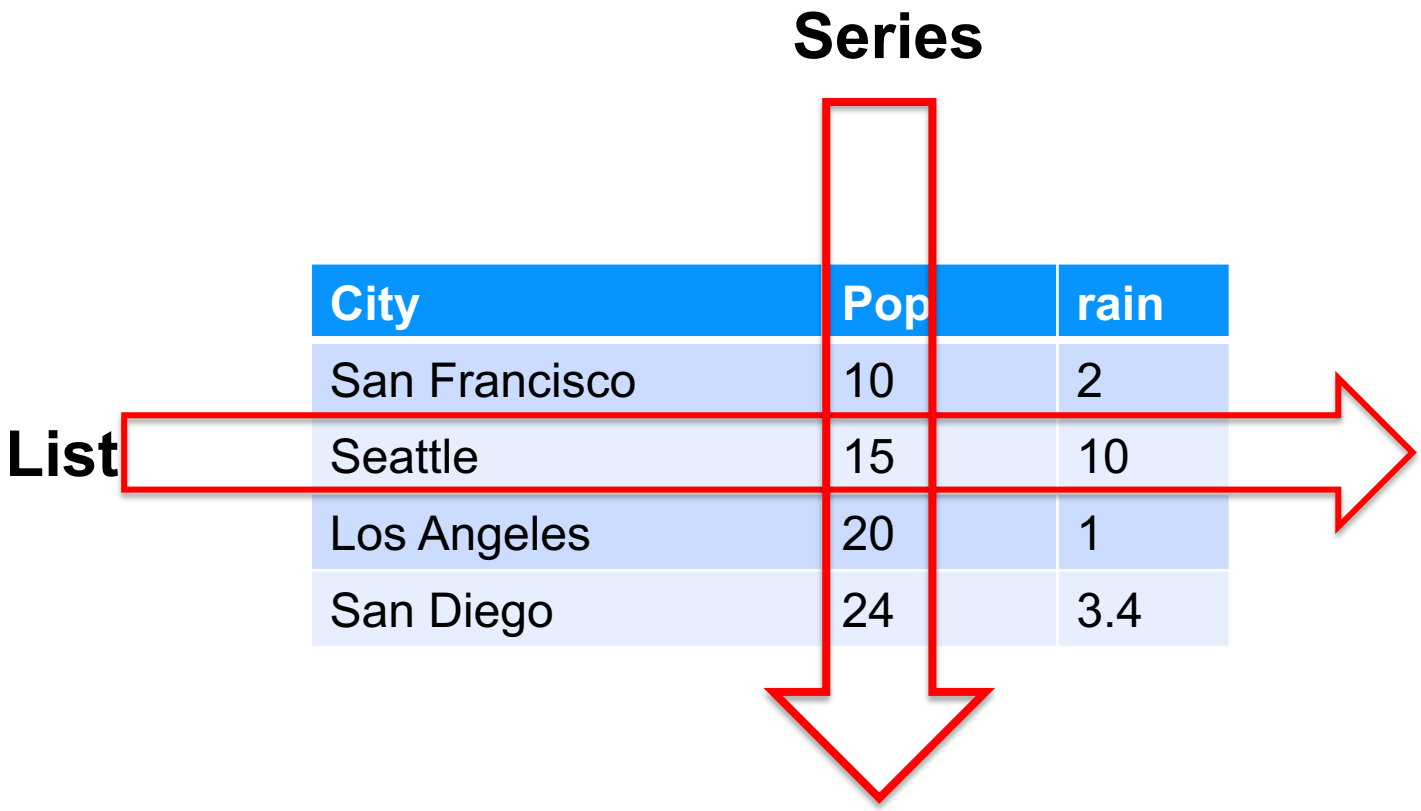
```
>>> cities = pd.DataFrame ({
          'city' : ['San Francisco', 'Seattle', 'Los Angeles'],
          'pop' : [10, 15, 20],
          'rain' = [2, 10,1]})

>>> cities
            city pop rain
0 San Francisco  10    2
1       Seattle  15   10
2  Los Angeles  20    1
```

Data Frame: Appending a New Row

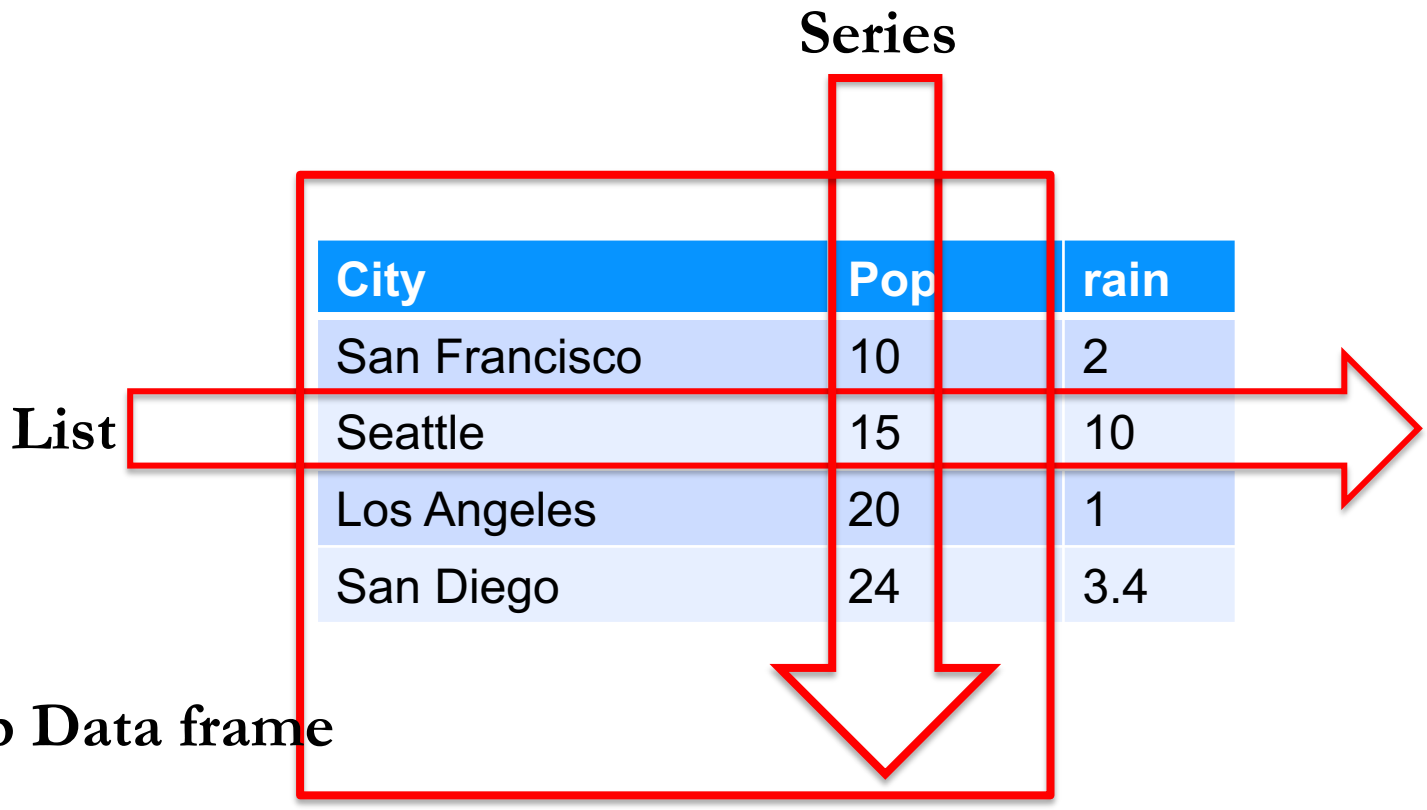
City	Pop	rain
San Francisco	10	2
Seattle	15	10
Los Angeles	20	1
San Diego	24	3.4

```
>>> cities = pd.DataFrame ({
          'city' : ['San Francisco', 'Seattle', 'Los Angeles'],
          'pop' : [10, 15, 20],
          'rain' = [2, 10,1]})


> sandiego = pd.DataFrame( {'city' :'San Diego','pop':24,
'rain':3.4)

> cities = cities.concat(sandiego)
```

Slicing Data Frames



Data Frame Slicing



City	Pop	rain
San Francisco	10	2
Seattle	15	10
Los Angeles	20	1
San Diego	24	3.4

Accessing a Data Frame by numeric index

City	Pop	rain
San Francisco	10	2
Seattle	15	10
Los Angeles	20	1
San Diego	24	3.4

```
# note we specify row index

> cities.iloc[[1]]

    city  pop  rain
0  San Francisco   10     2
```

Accessing a Data Frame

City	Pop	rain
San Francisco	10	2
Seattle	15	10
Los Angeles	20	1
San Diego	24	3.4

```
Columns are returned as series

# by name  dataframe['column_name'] or dataframe.
> cities['pop']  # can also say cities.pop
0  10.0
1  15.0
Name: cities, dtype: float64In [ ]:

# specify index: dataframe[:,idx ]
> cities.iloc[:,1]
10  15  20  24

> cities.iloc[:2].dtype
float64

# specify column index  :  dataframe.iloc[:, idx]
> cities.iloc[:,1]
10  15  20  24
```

Accessing a Data Frame using .iloc()

City	Pop		rain
San Francisco	10		2
Seattle	15		10
Los Angeles	20		1
San Diego	24		3.4

```
# Sub dataframes are obtained using iloc
# dataframe [n] : one column as a dataframe
# dataframe [, n]

> cities.iloc(,[1])
           city
1 San Francisco
2       Seattle
3   Los Angeles

type(cities.iloc(,[1]))
"data.frame"
```

Accessing Dataframes : Series vs. Sub Dataframe

Sub dataframe

```
> cities[[2]]

   pop
0   10
1   15
2   20
3   24

> type(cities[[2]])
<class
'pandas.core.frame.DataFra
```

Series

```
> cities[2]

10   15   20   24

> type(cities[2])
"<
'pandas.core.series.Series
'>"
```

Row
0
1
2
3

City	Pop	rain
San Francisco	10	2
Seattle	15	10
Los Angeles	20	1
San Diego	24	3.4

City	Pop	rain
San Francisco	10	2
Seattle	15	10
Los Angeles	20	1
San Diego	24	3.4

Apply function

- Similar to map()

```
def clean(x):
    x = x.replace("$", "").replace(",", "").replace(" ", "")
    return float(x)

data['revenue'] = data['Revenue'].apply(clean)

# Using lambda

data['revenue'] = data['revenue'].apply(
  lambda x : x.replace("," ""))
```

Lab: Dataframes in Pandas

- **Overview**:
  Learn Python Data frames

- **Approximate time**:
  15 minutes

- **Instructions** :
  01-intro/

Reading Data From Files

- Python can read from a variety of files.
- Text : CSV, TSV - **pd.read_csv()**
- Excel spreadsheets - **pd.read_excel()**
- Stata files – **pd.read_stata()**
- More handy functions in **read.***

```
# reading a file
Df = pd.read_csv(file)

# reading a URL
data.frame = pd.read_csv(http://somedomain.com/data.csv)
```

Reading From Databases

- Pandas has good connectivity with databases.
- pyodbc – uses ODBC protocol
- Various database packages
- Importing large datasets (> 1GB) from DB could be slow.
- For faster imports:
  - 1) export data out of DB as CSV etc.
  - 2) then import the file into Python

Example: Pandas and Databases

- Example: open connection to sqllite database
- Then call pandas.

```
import pandas as pd
import sqlite3

conn = sqlite3.connect("flights.db")

df = pd.read_sql_query("select * from airlines limit 5;",
conn)

df
```

Saving Data To Files

- write.* family supports exporting to multiple formats
- **pd.to_csv** writes csv (file name, sep = ,)
- **Encoding:** writes sv (encoding= ‘utf-8’)
- Index: Python will write the index unless you say (index=‘false’)
- Many other pd.to_* type functions

Working With Raw Data

- Datasets are seldom in a format to be used.
- Big part of data science is
  - Cleaning data up
  - Combining data sets etc.
- The 'not so sexy part of data science' ☺

Dealing With Missing Values

City	Month	Rainfall
San Franicco	Jan	10
San Francicso	Mar	5
Los Angeles	Mar	NA
Seattle	Apr	NA

- **Question**:
  Is the rainfall in Los Angeles in March zero?

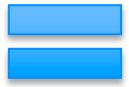
- Missing values can often ‘skew’ analysis.
- Python represents them with **NA** (Not Available).
- Use **df.dropna()** to remove **NA** values.

Combining Datasets : concat()

- **concat()** will merge two dataframes column-wise

A	B
X	10
Y	5

C	D
20	1.5
30	3.4



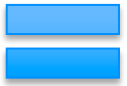
A	B	C	D
X	10	20	1.5
Y	5	30	3.4

Combining Dataframes : merge()

- **merge**() will try to take into account common column names and row names

A	B
X	10
Y	5

	C	D
X	20	1.5
Y	30	3.4



A	B	C	D
X	10	20	1.5
Y	5	30	3.4

Dealing With Duplicates

- Often data sets have duplicates
- It is a good practice to check for duplicates as part of the cleanup phase
- This is called '**de-duping**'
- Use **df.drop_duplicates()** function

Sorting Data Frames: sort_values()

- **Sort_Values()** function can help you sort a dataframe by columns

```
> citystats.sort_values('pop', ascending=True)


          city pop rain
1 San Francisco  10    2
2      Seattle  15   10
3  Los Angeles  20    1
4    San Diego  20    2
```

Sorting Data Frames: sort_values()

- Reverse sorting

```
> citystats.sort_values('pop', ascending=False)
```

```
          city pop rain
2      Seattle  15   10
1 San Francisco  10    2
4    San Diego  20    2
3  Los Angeles  20    1
```

describe() function

- **describe()** function gives you a quick glance of data
- Tells us: Min, max, median, …
- Gives a 'good feel' for the data
- Also tells quantiles (25%, 50%, 75%)

```
>>> df.describe()       numeric
count       3.0
mean        2.0
std         1.0
min         1.0
25%         1.5
50%         2.0
75%         2.5
max         3.0
```

Dealing With Categorical Variables

- Pandas can help us in dealing with categorical variables
  - Dtype="category"
- factorize()
  - Indexes and converts data into a index
- get_dummies()
  - Converts into a group of "one-hot" encoded variables.

Lab: Pandas

- **Overview**:
  Load data from files and cleanup

- **Approximate time**:
  15 minutes

- **Instructions** :
  01-intro / 04-Pandas

Exploring Data

- Pandas is a great tool for exploring datasets
- We are going to view the NYCFlights13 dataset
  - All flights to or from NYC airports in 2013
- Think of some interesting analytics that YOU can do with this data

Pandas Visualization

- Pandas allows us to do some quick visualizations
- Wraps content in matplotlib (covered later)
- We will do some basic Pandas visualizations in this lab.

Lab: Exploring Pandas

- **Overview**:
  Explore NYC flights dataset, and do some

- **Approximate time**:
  45 minutes

- **Instructions** :
  01-intro / 05-Exploring_Pandas

Lab: Data Preparation

- **Overview**:
  Load data from files and cleanup

- **Approximate time**:
  15 minutes

- **Instructions** :
  07-data-frame / 7.3-data-prep.md

Review Questions

Licensed for personal use only for Vincent Chang <vincent.chang@macys.com> from Python ML @ Macy's 04/25/2108 @ 2018-04-25