ECE5984 – Applications of Machine Learning
Lecture 6 – Python

Creed Jones, PhD

Course Updates

- Quiz 2 on February 10
  - Covers lectures 4-7
- At the end of the semester, I will replace your lowest quiz grade with your next lowest grade
- HW1 is posted
  - Due on Feb 8
  - Submit via Canvas
- Remember to email me with project teams (3-4 members) if you wish

Today's Objectives

Python

- Concept
- numpy and scikit-learn

Intro to Python from UNC

A simple example

We will be using Python in this course, along with the numpy, scikit and pandas libraries (and maybe others)

- You can use any reasonable Python development environment
  - PyCharm from JetBrains (documentation is posted)
  - Anaconda and Spyder
  - Jupyter
  - Google CoLab
  - text editor and command line (*just kidding, don't do this!*)

- You will find that parts of Python will remind you of Java or C++
- and numpy has many similarities to Matlab

- This course does not have the goal to make you a great Python coder
  - I am not one
- but it's the best environment for us to explore the material of this course

numpy and scikit-learn are packages that add-on to Python, to allow additional capabilities

- NumPy enables many nice math capabilities
  - Linear algebra is the most significant
  - Mathematical data types, various algorithms…
  - Think of the capabilities of Matlab
  - https://numpy.org/

- scikit-learn supports machine learning and data analysis
  - we will be introducing this as the need arises
  - https://scikit-learn.org/stable/

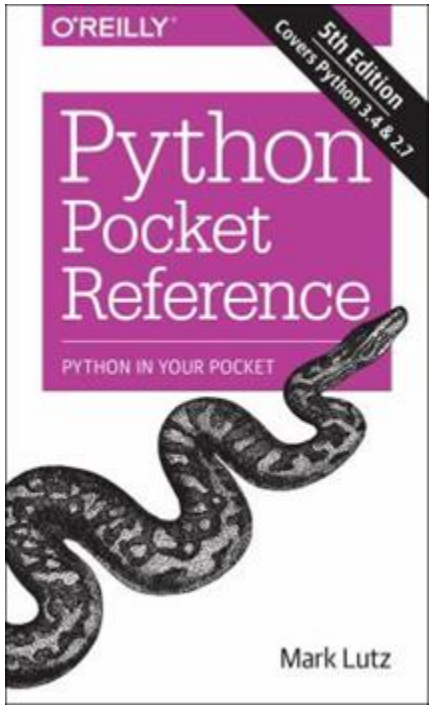
I like to use pandas for accessing data from Excel spreadsheets and other data formats (JSON, SQL, flat files, etc.)

- Typically, read data into a DataFrame object
- There are many functions for operating directly on a DataFrame
- I will usually also convert the data table into one or more numpy arrays
  - suitable for scikit-learn, OpenCV and other analytics packages

- https://pandas.pydata.org/

Here are some resources for you to get started in learning Python

- This is a useful reference to Python
- It's about $12



- Anaconda distribution
  - https://www.anaconda.com
- PyCharm Python IDE
  - https://www.jetbrains.com/pycharm/
- Official Python documentation
  - https://www.python.org/doc/
- A decent Python tutorial
  - https://www.w3schools.com/python/
- Numpy
  - https://numpy.org/

I am going to quickly walk through some slides from Professor Liu at the University of North Carolina

- A bit of Python origins
- Python syntax
- Brief intro to numpy

UNC
INFORMATION
TECHNOLOGY SERVICES

Python: An Introduction

Shubin Liu, Ph.D.
Research Computing Center
University of North Carolina at Chapel Hill

Agenda

- Introduction
- Running Python
- Python Programming
  - Data types
  - Control flows
  - Classes, functions, modules
- Hands-on Exercises

The PPT/WORD format of this presentation is available here:
http://its2.unc.edu/divisions/rc/training/scientific/
/afs/isis/depts/its/public_html/divisions/rc/training/scientific/short_courses/

What is python?

- Object oriented language
- Interpreted language
- Supports dynamic data type
- Independent from platforms
- Focused on development time
- Simple and easy grammar
- High-level internal object data types
- Automatic memory management
- It’s free (open source)!

Language properties

- Everything is an object
- Modules, classes, functions
- Exception handling
- Dynamic typing, polymorphism
- Static scoping
- Operator overloading
- Indentation for block structure

High-level data types

- Numbers: int, long, float, complex
- Strings: immutable
- Lists and dictionaries: containers
- Other types for e.g. binary data, regular expressions, introspection
- Extension modules can define new “built-in” data types

Comments

- Start with # and go to end of line
- What about C, C++ style comments?
  - NOT supported!

Python Syntax

- Much of it is similar to C syntax
- Exceptions:
  - missing operators: `++`, `--`
  - no curly brackets, `{}`, for blocks; uses whitespace
  - different keywords
  - lots of extra features
  - no type declarations!

Simple data types

- Numbers
  - Integer, floating-point, complex!
- Strings
  - characters are strings of length 1
- Booleans are `False` or `True`

Numbers

- The usual notations and operators
  - 12, 3.14, 0xFF, 0377, (-1+2)*3/4**5, abs(x), 0<x<=5
- C-style shifting & masking
  - 1<<16, x&0xff, x|1, ~x, x^y
- Integer division truncates :-(
  - 1/2 -> 0 # float(1)/2 -> 0.5
- Long (arbitrary precision), complex
  - 2L**100 -> 1267650600228229401496703205376L
  - 1j**2 -> (-1+0j)

Strings and formatting

```
i = 10

d = 3.1415926

s = "I am a string!"

print("%d\t%f\t%s" % (i, d, s))

print("newline\n")

print("no newline")
```

Variables

- No need to declare
- Need to assign (initialize)
  - use of uninitialized variable raises exception
- Not typed

```
if friendly: greeting = "hello world"
else: greeting = 12**2
print greeting
```

- ***Everything*** is a variable:
  - functions, modules, classes

Reference semantics

- Assignment manipulates references
  - x = y **does not make a copy** of y
  - x = y makes x **reference** the object y references
- Very useful; but beware!
- Example:

```
>>> a = [1, 2, 3]; b = a
>>> a.append(4); print b
[1, 2, 3, 4]
```

Simple data types: operators

- `+ - * / %` (like C)
- `+= -=` etc. (no ++ or --)
- Assignment using `=`
  - but semantics are different!

```
a = 1
a = "foo"   # OK
```

- Can also use + to concatenate strings

Strings

- "hello"+"world" "helloworld" # concatenation
- "hello"*3 "hellohellohello" # repetition
- "hello"[0] "h" # indexing
- "hello"[-1] "o" # (from end)
- "hello"[1:4] "ell" # slicing
- len("hello") 5 # size
- "hello" < "jello" 1 # comparison
- "e" in "hello" 1 # search
- New line: "escapes: \n "
- Line continuation: triple quotes ’’’
- Quotes: ‘single quotes’, "raw strings"

Simple Data Types

- Triple quotes useful for multi-line strings

```
>>> s = """ a long
... string with "quotes" or anything else"""
>>> s
' a long\012string with "quotes" or anything
else'
>>> len(s)
45
```

Methods in string

- upper()
- lower()
- capitalize()
- count(s)
- find(s)
- rfind(s)
- index(s)
- strip(), lstrip(), rstrip()
- replace(a, b)
- expandtabs()
- split()
- join()
- center(), ljust(), rjust()

Compound Data Type: List

- List:
  - A container that holds a number of other objects, in a given order
  - Defined in square brackets

```
a = [1, 2, 3, 4, 5]
print a[1]  # number 2
some_list = []
some_list.append("foo")
some_list.append(12)
print len(some_list)  # 2
```

List

- a = [99, "bottles of beer", ["on", "the", "wall"]]

- Flexible arrays, *not* Lisp-like linked lists
- Same operators as for strings
  - a+b, a*3, a[0], a[-1], a[1:], len(a)
- Item and slice assignment
  - a[0] = 98
  - a[1:2] = ["bottles", "of", "beer"]

    -> [98, "bottles", "of", "beer", ["on", "the", "wall"]]
  - del a[-1] # -> [98, "bottles", "of", "beer"]

More list operations

```
>>> a = range(5)            # [0,1,2,3,4]
>>> a.append(5)             # [0,1,2,3,4,5]
>>> a.pop()                 # [0,1,2,3,4]
5
>>> a.insert(0, 5.5)        # [5.5,0,1,2,3,4]
>>> a.pop(0)                # [0,1,2,3,4]
5.5
>>> a.reverse()                   # [4,3,2,1,0]
>>> a.sort()                # [0,1,2,3,4]
```

Operations in List

- append
- insert
- index
- count
- sort
- reverse
- remove
- pop
- extend

- Indexing e.g., L[i]
- Slicing e.g., L[1:5]
- Concatenation e.g., L + L
- Repetition e.g., L * 5
- Membership test e.g., ‘a’ in L
- Length e.g., len(L)

Nested List

- List in a list
- E.g.,
  - >>> s = [1,2,3]
  - >>> t = [‘begin’, s, ‘end’]
  - >>> t
  - [‘begin’, [1, 2, 3], ‘end’]
  - >>> t[1][1]
  - 2

Dictionaries

- Dictionaries: curly brackets
  - What is dictionary?
    - Refer value through key; “associative arrays”
  - Like an array indexed by a string
  - An unordered set of *key: value* pairs
  - *Values* of any type; keys of almost any type
    - {"name":"Guido", "age":43, ("hello","world"):1, 42:"yes", "flag": ["red","white","blue"]}

```
d = { "foo" : 1, "bar" : 2 }
print d["bar"]   # 2
some_dict = {}
some_dict["foo"] = "yow!"
print some_dict.keys() # ["foo"]
```

Methods in Dictionary

- keys()
- values()
- items()
- has_key(key)
- clear()
- copy()
- get(key[,x])
- setdefault(key[,x])
- update(D)
- popitem()

Dictionary details

- Keys must be **immutable**:
  - numbers, strings, tuples of immutables
    - these cannot be changed after creation
  - reason is *hashing* (fast lookup technique)
  - **not** lists or other dictionaries
    - these types of objects can be changed "in place"
  - no restrictions on values
- Keys will be listed in **arbitrary order**
  - again, because of hashing

Tuples

- What is a tuple?
  - A tuple is an ordered collection which cannot be modified once it has been created.
  - In other words, it's a special array, a read-only array.
- How to make a tuple? In round brackets
  - E.g.,

```
>>> t = ()
>>> t = (1, 2, 3)
>>> t = (1, )
>>> t = 1,
>>> a = (1, 2, 3, 4, 5)
>>> print a[1]  # 2
```

Operations in Tuple

- Indexing e.g., T[i]
- Slicing e.g., T[1:5]
- Concatenation e.g., T + T
- Repetition e.g., T * 5
- Membership test e.g., ‘a’ in T
- Length e.g., len(T)

List vs. Tuple

- What are common characteristics?
  - Both store arbitrary data objects
  - Both are of sequence data type
- What are differences?
  - Tuple **doesn't allow modification**
  - Tuple doesn't have methods
  - Tuple supports format strings
  - Tuple supports variable length parameter in function call.
  - Tuples **slightly faster**

Data Type Wrap Up

- Integers: 2323, 3234L
- Floating Point: 32.3, 3.1E2
- Complex: 3 + 2j, 1j
- Lists: l = [1,2,3]
- Tuples: t = (1,2,3)
- Dictionaries: d = {‘hello’ : ‘there’, 2 : 15}

Data Type Wrap Up

- Lists, Tuples, and Dictionaries can store any type (including other lists, tuples, and dictionaries!)
- Only lists and dictionaries are mutable
- All variables are references

Input

- The raw_input(string) method returns a line of user input as a string
- The parameter is used as a prompt
- The string can be converted by using the conversion methods int(string), float(string), etc.

File I/O

```
f = file("foo", "r")
line = f.readline()
print line,
f.close()
# Can use sys.stdin as input;
# Can use sys.stdout as output.
```

Files: Input

input = open('data', 'r')	Open the file for input
S = input.read()	Read whole file into one String
S = input.read(N)	Reads N bytes (N >= 1)
L = input.readlines()	Returns a list of line strings

Files: Output

output = open('data', 'w')	Open the file for writing
output.write(S)	Writes the string S to file
output.writelines(L)	Writes each of the strings in list L to file
output.close()	Manual close

`open()` and `file()`

- These are identical:

```
f = open(filename, "r")
f = file(filename, "r")
```

- The `open()` version is older
- The `file()` version is the recommended way to open a file now
  - uses object constructor syntax (next lecture)

OOP Terminology

- class -- a template for building objects
- instance -- an object created from the template (an instance of the class)
- method -- a function that is part of the object and acts on instances directly
- constructor -- special "method" that creates new instances

Objects

- Objects:
  - What is an object?
    - data structure, and
    - functions (methods) that operate on it

```
class thingy:
    # Definition of the class here, next slide
t = thingy()
t.method()
print t.field
```

- Built-in data structures (lists, dictionaries) are also objects
  - though internal representation is different

Defining a class

```
class Thingy:

    """This class stores an arbitrary object."""

    def __init__(self, value):

        """Initialize a Thingy."""

        self.value = value

    def showme(self):

        """Print this object to stdout."""

        print "value = %s" % self.value
```

constructor

method

Using a class (1)

```
t = Thingy(10)   # calls __init__ method

t.showme()       # prints "value = 10"
```

- `t` is an instance of class **`Thingy`**
- **`showme`** is a method of class **`Thingy`**
- **`__init__`** is the constructor method of class **`Thingy`**
  - when a **`Thingy`** is created, the **`__init__`** method is called
- Methods starting and ending with __ are "special" methods

Using a class (2)

```
print t.value  # prints "10"
```

- **value** is a *field* of class **Thingy**

```
t.value = 20   # change the field value

print t.value  # prints "20"
```

"Special" methods

- All start and end with __ (two underscores)
- Most are used to emulate functionality of built-in types in user-defined classes
- *e.g.* operator overloading
  - `__add__`, `__sub__`, `__mult__`, `...`
  - see python docs for more information

Control flow (1)

- `if, if/else, if/elif/else`

```
if a == 0:
    print "zero!"
elif a < 0:
    print "negative!"
else:
    print "positive!"
```

- Notes:
  - blocks delimited by indentation!
  - colon (:) used at end of lines containing control flow keywords

Control flow (3)

- `while` loops

```
a = 10
while a > 0:
    print a
    a -= 1
```

Control flow (4)

- `for` loops

```
for a in range(10):
    print a
```

- really a "foreach" loop

Control flow (5)

- Common `for` loop idiom:

```
a = [3, 1, 4, 1, 5, 9]
for i in range(len(a)):
    print a[i]
```

Control flow (6)

- Common `while` loop idiom:

```
f = open(filename, "r")
while True:
    line = f.readline()
    if not line:
        break
    # do something with line
```

Control flow (7): odds & ends

- `continue` statement like in C
- `pass` keyword:

```
if a == 0:
    pass  # do nothing
else:
    # whatever
```

Defining functions

```
def foo(x):
    y = 10 * x + 2
    return y
```

- All variables are local unless specified as `global`
- Arguments passed by value

Executing functions

```
def foo(x):
    y = 10 * x + 2
    return y

print foo(10)   # 102
```

Why use modules?

- Code reuse
  - Routines can be called multiple times within a program
  - Routines can be used from multiple programs
- Namespace partitioning
  - Group data together with functions used for that data
- Implementing shared services or data
  - Can provide global data structure that is accessed by multiple subprograms

Modules

- Modules are functions and variables defined in separate files
- Items are imported using from or import
  - from module import function
  - function()
  - import module
  - module.function()
- Modules are namespaces
  - Can be used to organize variable names, i.e.
    - atom.position = atom.position - molecule.position

Modules

- Access other code by importing modules

```
import math

print math.sqrt(2.0)
```

- or:

```
from math import sqrt

print sqrt(2.0)
```

Modules

- or:

```
from math import *
print sqrt(2.0)
```

- Can import multiple modules on one line:

```
import sys, string, math
```

- Only one "`from x import y`" per line

Example: NumPy Modules

- http://numpy.scipy.org/
- NumPy has many of the features of Matlab, in a free, multiplatform program. It also allows you to do intensive computing operations in a simple way
- Numeric Module: Array Constructors
  - ones, zeros, identity
  - arrayrange
- LinearAlgebra Module: Solvers
  - Singular Value Decomposition
  - Eigenvalue, Eigenvector
  - Inverse
  - Determinant
  - Linear System Solver

Arrays and Constructors

```
>>> a = ones((3,3),float)
>>> print a
[[1., 1., 1.],
 [1., 1., 1.],
 [1., 1., 1.]]
>>> b = zeros((3,3),float)
>>> b = b + 2.*identity(3) #"+" is overloaded
>>> c = a + b
>>> print c
[[3., 1., 1.],
 [1., 3., 1.],
 [1., 1., 3.]]
```

Overloaded operators

```
>>> b = 2.*ones((2,2),float) #overloaded
>>> print b
[[2.,2.],
 [2.,2.]]
>>> b = b+1    # Addition of a scalar is
>>> print b    #    element-by-element
[[3.,3.],
 [3.,3.]]
>>> c = 2.*b  # Multiplication by a scalar is
>>> print c    #    element-by-element
[[6.,6.],
 [6.,6.]]
```

Array functions

- >>> from LinearAlgebra import *
- >>> a = zeros((3,3),float) + 2.*identity(3)
- >>> print inverse(a)
- [[0.5, 0., 0.],
- [0., 0.5, 0.],
- [0., 0., 0.5]]
- >>> print determinant(inverse(a))
- 0.125
- >>> print diagonal(a)
- [0.5,0.5,0.5]
- >>> print diagonal(a,1)
- [0.,0.]

- transpose(a), argsort(), dot()

Catching Exceptions

#python code a.py

```
x = 0
try:
    print 1/x
except ZeroDivisionError, message:
    print "Can’t divide by zero:"
    print message
```

>>>python a.py

Can't divide by zero:

integer division or modulo by zero

Try-Finally: Cleanup

```
f = open(file)
try:
    process_file(f)
finally:
    f.close()     # always executed
print "OK"       # executed on success only
```

Python: Pros & Cons

- Pros
  - Free availability (like Perl, Python is open source).
  - Stability (Python is in release 2.6 at this point and, as I noted earlier, is older than Java).
  - Very easy to learn and use
  - Good support for objects, modules, and other reusability mechanisms.
  - Easy integration with and extensibility using C and Java.
- Cons
  - Smaller pool of Python developers compared to other languages, such as Java
  - Lack of true multiprocessor support
  - Absence of a commercial support point, even for an Open Source project (though this situation is changing)
  - Software performance slow, not suitable for high performance applications

References

- Python Homepage
  - http://www.python.org
- Python Tutorial
  - http://docs.python.org/tutorial/
- Python Documentation
  - http://www.python.org/doc
- Python Library References
  - http://docs.python.org/release/2.5.2/lib/lib.html
- Python Add-on Packages:
  - http://pypi.python.org/pypi

I like to use PyCharm, but you can use many different environments for Python development

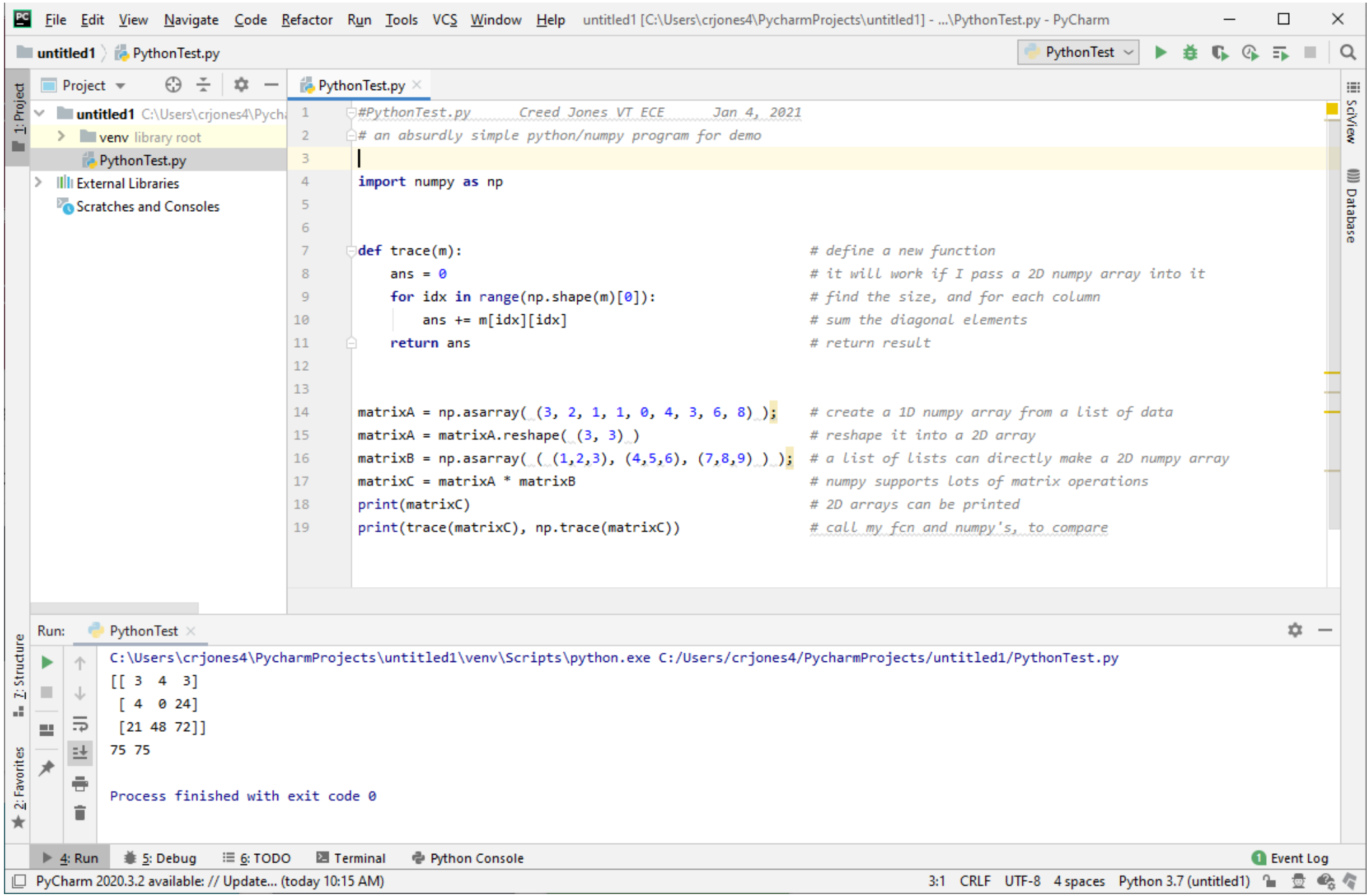
- JetBrains PyCharm
  - https://www.jetbrains.com/pycharm/
- Anaconda / Spyder
  - https://www.anaconda.com/
- Jupyter
  - https://jupyter.org/
- Google CoLab
  - https://colab.research.google.com/notebooks/intro.ipynb

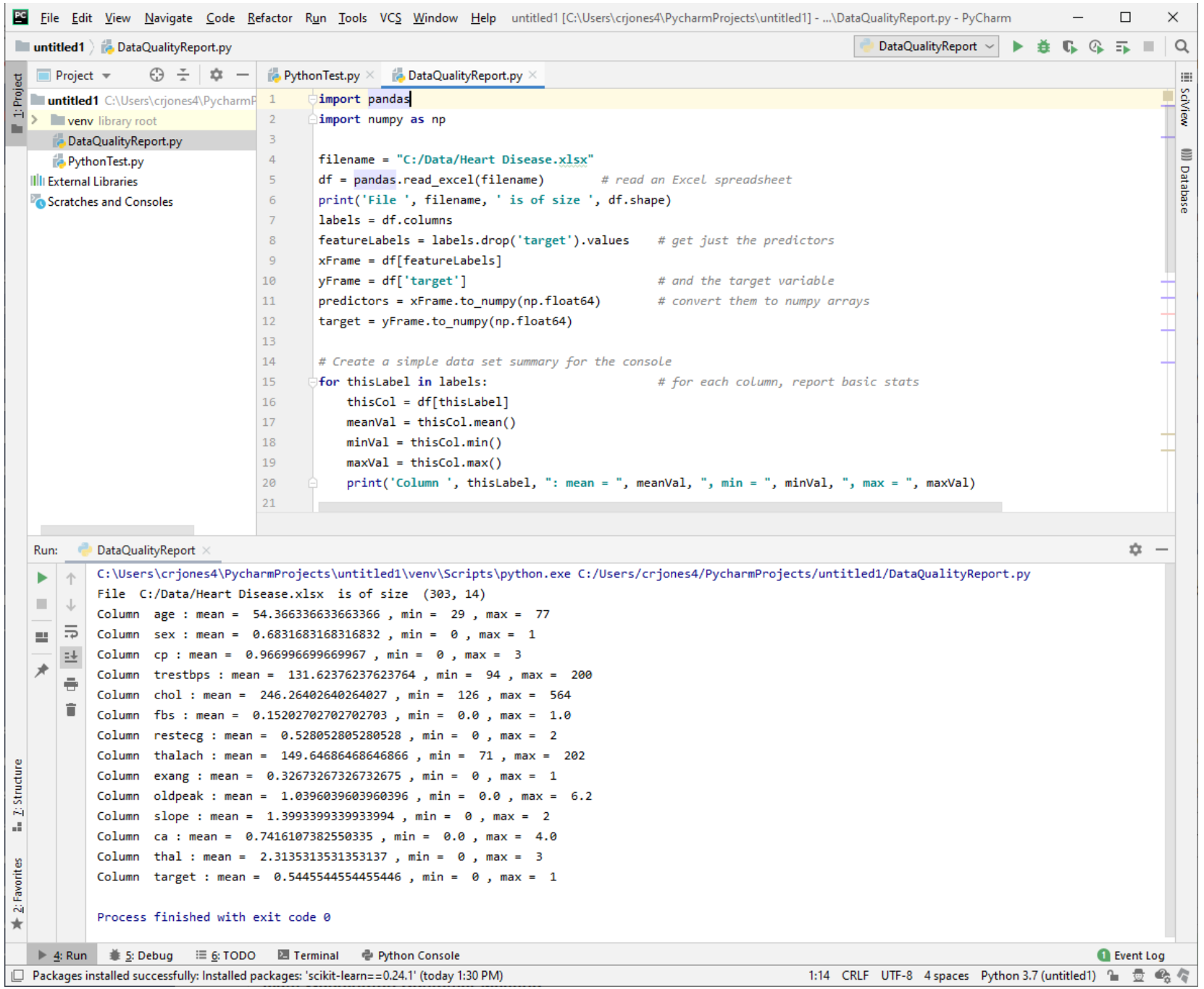
```
#PythonTest.py      Creed Jones VT ECE      Jan 4, 2021
# an absurdly simple python/numpy program for demo

import numpy as np


def trace(m):                                           # define a new function
    ans = 0                                             # it will work if I pass a 2D numpy array
into it
    for idx in range(np.shape(m)[0]):                   # find the size, and for each column
        ans += m[idx][idx]                              # sum the diagonal elements
    return ans                                          # return result


matrixA = np.asarray( (3, 2, 1, 1, 0, 4, 3, 6, 8) );    # create a 1D numpy array from a
                                                        # list of data
matrixA = matrixA.reshape( (3, 3) )                     # reshape it into a 2D array
matrixB = np.asarray( ( (1,2,3), (4,5,6), (7,8,9) ) );  # a list of lists can directly make
                                                        # a 2D numpy array
matrixC = matrixA * matrixB                             # numpy supports lots of matrix operations
print(matrixC)                                          # 2D arrays can be printed
print(trace(matrixC), np.trace(matrixC))                # call my fcn and numpy's, to compare
```

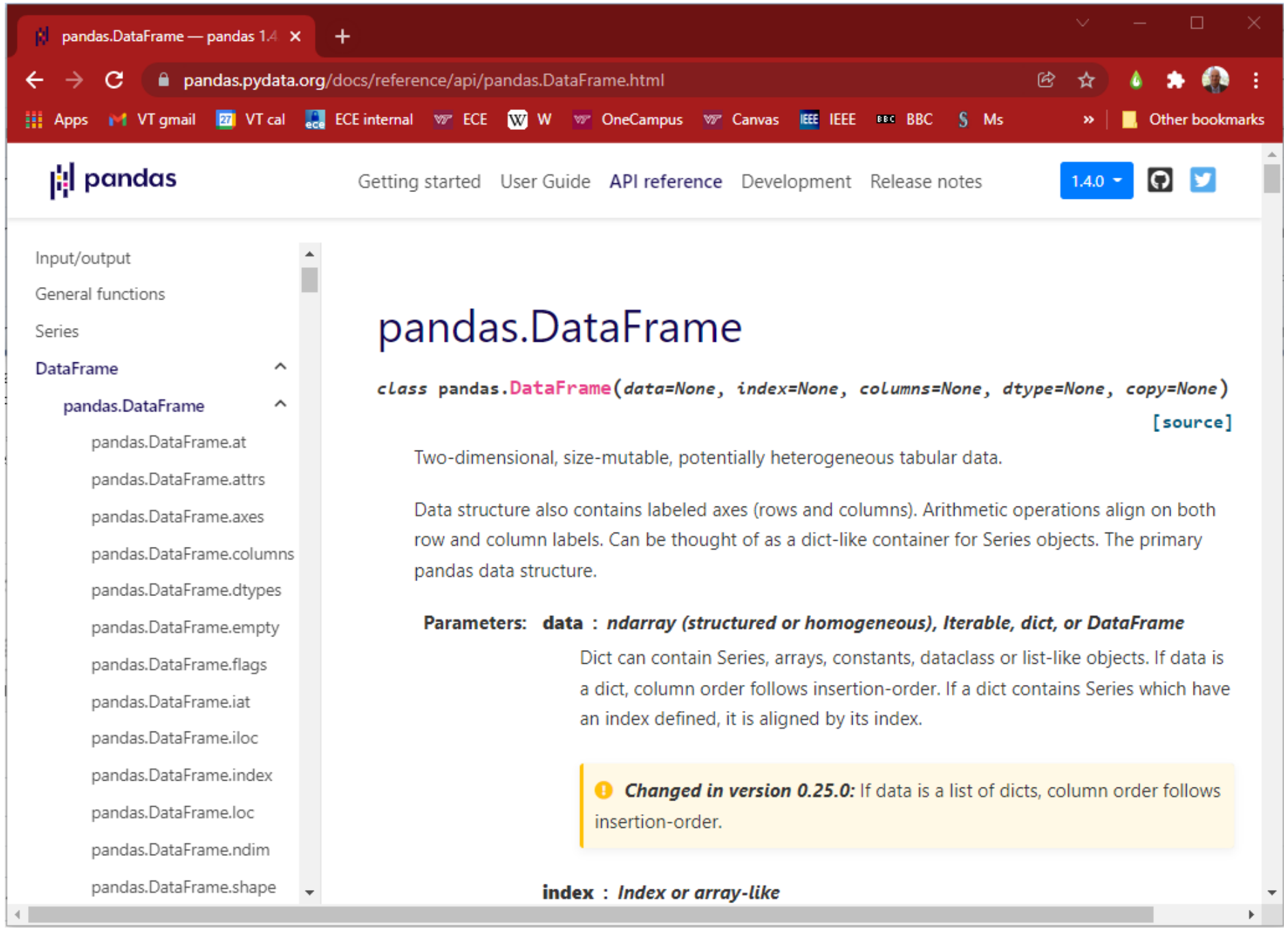

File Edit View Navigate Code Refactor Run Tools VCS Window Help untitled1 [C:\Users\crjones4\PycharmProjects\untitled1] - ...\PythonTest.py - PyCharm
untitled1 PythonTest.py
PythonTest
Project
untitled1 C:\Users\crjones4\Pych
venv library root
PythonTest.py
External Libraries
Scratches and Consoles
PythonTest.py
#PythonTest.py Creed Jones VT ECE Jan 4, 2021
# an absurdly simple python/numpy program for demo
import numpy as np
def trace(m): # define a new function
ans = 0 # it will work if I pass a 2D numpy array into it
for idx in range(np.shape(m)[0]): # find the size, and for each column
ans += m[idx][idx] # sum the diagonal elements
return ans # return result
matrixA = np.asarray((3, 2, 1, 1, 0, 4, 3, 6, 8)); # create a 1D numpy array from a list of data
matrixA = matrixA.reshape((3, 3)) # reshape it into a 2D array
matrixB = np.asarray(((1,2,3), (4,5,6), (7,8,9))); # a list of lists can directly make a 2D numpy array
matrixC = matrixA * matrixB # numpy supports lots of matrix operations
print(matrixC) # 2D arrays can be printed
print(trace(matrixC), np.trace(matrixC)) # call my fcn and numpy's, to compare
Run: PythonTest
C:\Users\crjones4\PycharmProjects\untitled1\venv\Scripts\python.exe C:/Users/crjones4/PycharmProjects/untitled1/PythonTest.py
[[3 4 3]
[4 0 24]
[21 48 72]]
75 75
Process finished with exit code 0
4: Run 5: Debug 6: TODO Terminal Python Console Event Log
PyCharm 2020.3.2 available: // Update... (today 10:15 AM)
3:1 CRLF UTF-8 4 spaces Python 3.7 (untitled1)


File Edit View Navigate Code Refactor Run Tools VCS Window Help untitled1 [C:\Users\crjones4\PycharmProjects\untitled1] - ...\DataQualityReport.py - PyCharm
untitled1 DataQualityReport.py
DataQualityReport
Project
untitled1 C:\Users\crjones4\PycharmP
venv library root
DataQualityReport.py
PythonTest.py
External Libraries
Scratches and Consoles
PythonTest.py
DataQualityReport.py
1 import pandas
2 import numpy as np
3
4 filename = "C:/Data/Heart Disease.xlsx"
5 df = pandas.read_excel(filename) # read an Excel spreadsheet
6 print('File ', filename, ' is of size ', df.shape)
7 labels = df.columns
8 featureLabels = labels.drop('target').values # get just the predictors
9 xFrame = df[featureLabels]
10 yFrame = df['target'] # and the target variable
11 predictors = xFrame.to_numpy(np.float64) # convert them to numpy arrays
12 target = yFrame.to_numpy(np.float64)
13
14 # Create a simple data set summary for the console
15 for thisLabel in labels: # for each column, report basic stats
16 thisCol = df[thisLabel]
17 meanVal = thisCol.mean()
18 minVal = thisCol.min()
19 maxVal = thisCol.max()
20 print('Column ', thisLabel, ": mean = ", meanVal, ", min = ", minVal, ", max = ", maxVal)
21
Run: DataQualityReport
C:\Users\crjones4\PycharmProjects\untitled1\venv\Scripts\python.exe C:/Users/crjones4/PycharmProjects/untitled1/DataQualityReport.py
File C:/Data/Heart Disease.xlsx is of size (303, 14)
Column age : mean = 54.366336633663366 , min = 29 , max = 77
Column sex : mean = 0.6831683168316832 , min = 0 , max = 1
Column cp : mean = 0.966996699669967 , min = 0 , max = 3
Column trestbps : mean = 131.62376237623764 , min = 94 , max = 200
Column chol : mean = 246.26402640264027 , min = 126 , max = 564
Column fbs : mean = 0.15202702702702703 , min = 0.0 , max = 1.0
Column restecg : mean = 0.528052805280528 , min = 0 , max = 2
Column thalach : mean = 149.64686468646866 , min = 71 , max = 202
Column exang : mean = 0.32673267326732675 , min = 0 , max = 1
Column oldpeak : mean = 1.0396039603960396 , min = 0.0 , max = 6.2
Column slope : mean = 1.3993399339933994 , min = 0 , max = 2
Column ca : mean = 0.7416107382550335 , min = 0.0 , max = 4.0
Column thal : mean = 2.3135313531353137 , min = 0 , max = 3
Column target : mean = 0.5445544554455446 , min = 0 , max = 1
Process finished with exit code 0
4: Run 5: Debug 6: TODO Terminal Python Console Event Log
Packages installed successfully: Installed packages: 'scikit-learn==0.24.1' (today 1:30 PM)
1:14 CRLF UTF-8 4 spaces Python 3.7 (untitled1)
SciView Database
1: Project 7: Structure 2: Favorites

```
import pandas
import numpy as np

filename = "C:/Data/Heart Disease.xlsx"
df = pandas.read_excel(filename)        # read an Excel spreadsheet
print('File {0} is of size {1}'.format(filename, df.shape))
labels = df.columns
featureLabels = labels.drop('target').values    # get just the predictors
xFrame = df[featureLabels]
yFrame = df['target']                           # and the target variable
predictors = xFrame.to_numpy(np.float64)        # convert them to numpy arrays
target = yFrame.to_numpy(np.float64)

# Create a simple data set summary for the console
for thisLabel in labels:                        # for each column, report basic stats
    thisCol = df[thisLabel]
    meanV = thisCol.mean()
    minV = thisCol.min()
    maxV = thisCol.max()
    print('Col {0}: mean = {1}, min = {2}, max = {3}'.format(thisLabel, meanV, minV, maxV))
```


pandas.DataFrame — pandas 1.4
pandas.pydata.org/docs/reference/api/pandas.DataFrame.html
Apps
VT gmail
VT cal
ECE internal
ECE
W
OneCampus
Canvas
IEEE
BBC
Ms
Other bookmarks
pandas
Getting started
User Guide
API reference
Development
Release notes
1.4.0
Input/output
General functions
Series
DataFrame
pandas.DataFrame
pandas.DataFrame.at
pandas.DataFrame.attrs
pandas.DataFrame.axes
pandas.DataFrame.columns
pandas.DataFrame.dtypes
pandas.DataFrame.empty
pandas.DataFrame.flags
pandas.DataFrame.iat
pandas.DataFrame.iloc
pandas.DataFrame.index
pandas.DataFrame.loc
pandas.DataFrame.ndim
pandas.DataFrame.shape
pandas.DataFrame
class pandas.DataFrame(data=None, index=None, columns=None, dtype=None, copy=None)
[source]
Two-dimensional, size-mutable, potentially heterogeneous tabular data.
Data structure also contains labeled axes (rows and columns). Arithmetic operations align on both row and column labels. Can be thought of as a dict-like container for Series objects. The primary pandas data structure.
Parameters: data : ndarray (structured or homogeneous), Iterable, dict, or DataFrame
Dict can contain Series, arrays, constants, dataclass or list-like objects. If data is a dict, column order follows insertion-order. If a dict contains Series which have an index defined, it is aligned by its index.
Changed in version 0.25.0: If data is a list of dicts, column order follows insertion-order.
index : Index or array-like

Python supports object-oriented programming

```
class ClassName:
    def __init__(self, defVal):      # kinda like a constructor in C++/Java
        self.var1 = 42
        self.var2 = defVal
        pass

    def f1(self, valIn):             # member function
        self.var1 += valIn
```

- I highly recommend using OOD/OOP from the very beginning…

SimpleStatsV2.py

```
import pandas
import StatsReport

filename = "C:/Data/Heart Disease.xlsx"
df = pandas.read_excel(filename) # read Excel spreadsheet
print('File {0} is of size {1}'.format(filename, df.shape))
labels = df.columns
report = StatsReport.StatsReport()

# Create a simple data set summary for the console
for thisLabel in labels:    # for each column, report stats
    thisCol = df[thisLabel]
    report.addCol(thisLabel, thisCol)
print(report.to_string())
```

StatsReport.py

```
import pandas

class StatsReport:
    def __init__(self):
        self.statsdf = pandas.DataFrame()
        self.statsdf['stat'] = ['mean', 'min', 'max']
        pass

    def addCol(self, label, d):
        self.statsdf[label] = [d.mean(), d.min(), d.max()]

    def to_string(self):
         return self.statsdf.to_string()
```

```
File C:/Data/Heart Disease.xlsx is of size (303, 14)
   stat        age       sex        cp    trestbps        chol       fbs   restecg     thalach     exang
0  mean  54.366337  0.683168  0.966997  131.623762  246.264026  0.152027  0.528053  149.646865  0.326733
1   min  29.000000  0.000000  0.000000   94.000000  126.000000  0.000000  0.000000   71.000000  0.000000
2   max  77.000000  1.000000  3.000000  200.000000  564.000000  1.000000  2.000000  202.000000  1.000000
```

Today's Objectives

Python

- Concept
- numpy and scikit-learn

Intro to Python from UNC

A simple example