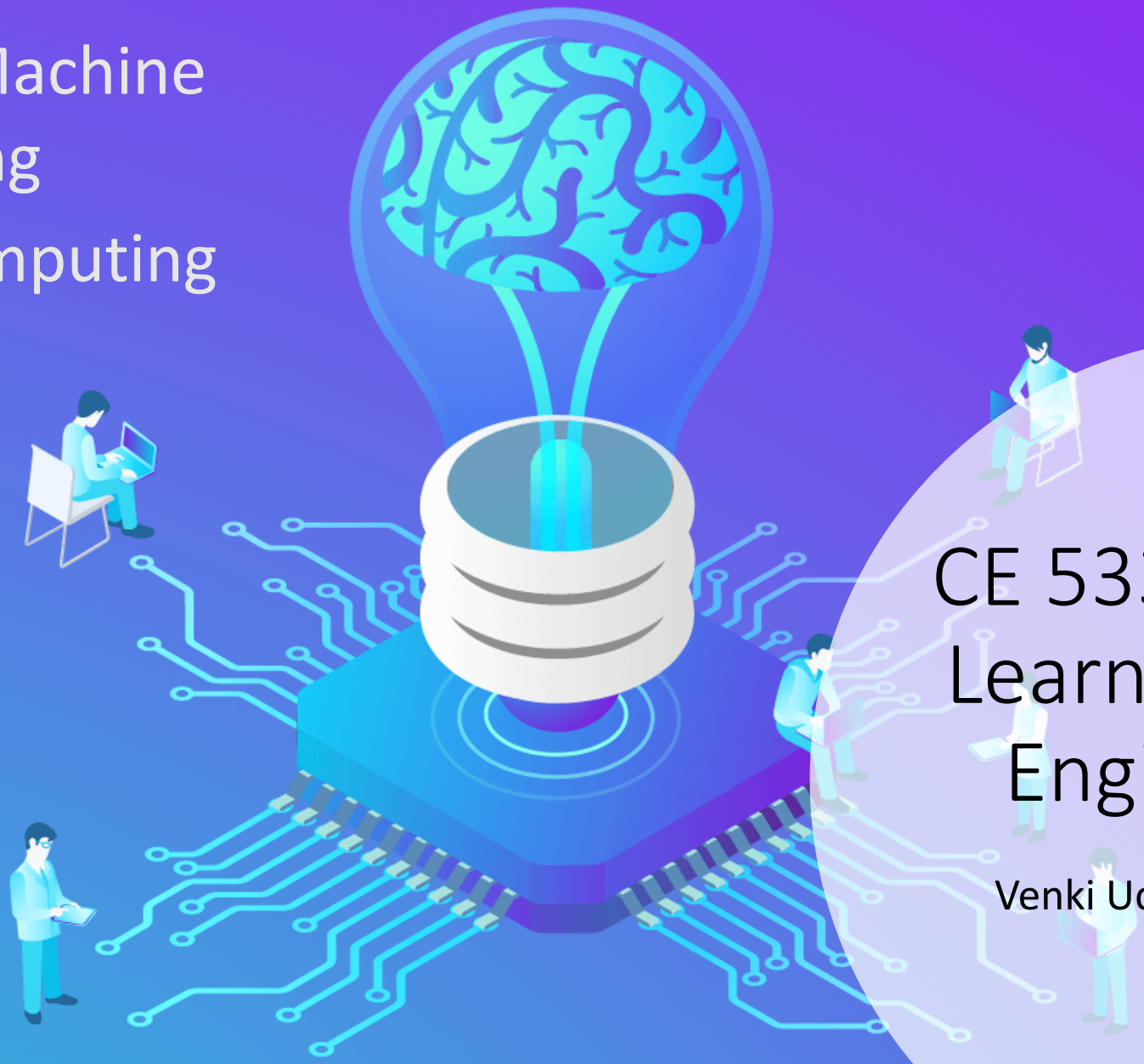
Python for Machine Learning

Scientific Computing



CE 5331 Machine Learning for Civil Engineers

Venki Uddameri, Ph.D. , P.E.

Recap and Goals

- Installed Python and Anaconda Environments
- Introduction to Python
  - Setting working directory
  - Adding comment lines
    - Docstrings
- Introduction to Pandas
  - Reading a csv
  - Extracting columns (attributes)
  - Extracting rows
  - Obtaining summary measures
- Basic graphing and charting in Python
  - Matplotlib package
- Control Statements
  - If, if-elif-else, if-else
  - For loop
  - While loop
  - Use of Boolean operators
- Functions
  - Passing inputs
  - Lambda functions
  - Pass by object reference
- Numpy
  - Matrix Calculations
  - Vectorization

Goal of this module is to explore Scientific Computing in Python

Scientific Computing

Machine learning algorithms utilize several scientific computing methods

- Root finding
- Integration

Python offers several options to perform these types of calculations

Scipy – Scientific and Technical Computing

- Scipy is a standard Python library for conducting scientific and technical computing
  - Builds on Numpy and is related other Numpy related packages such as matplotlib and pandas
    - Scipy is part of Numpy stack
    - Also referred to as Scipy stack
- Scipy library has several modules (sub libraries) that can be individually imported
  - Efficient memory management

Scipy Modules

- **constants**: physical constants and conversion factors
- **cluster**: hierarchical clustering, vector quantization, K-means
- **fftpack**: Discrete Fourier Transform algorithms
- **integrate**: numerical integration routines
- **interpolate**: interpolation tools
- **io**: data input and output
- **lib**: Python wrappers to external libraries
- **linalg**: linear algebra routines
- **misc**: miscellaneous utilities (e.g. image reading/writing)
- **ndimage**: various functions for multi-dimensional image processing
- **optimize**: optimization algorithms including linear programming
- **signal**: signal processing tools
- **sparse**: sparse matrix and related algorithms
- **spatial**: KD-trees, nearest neighbors, distance functions
- **special**: special functions
- **stats**: statistical functions
- **weave**: tool for writing C/C++ code as Python multiline strings

Scipy functionality is too vast to be explored in great detail – I will introduce some basic skills necessary for this course

Scipy – 1D Interpolation

- 1D interpolation comes handy in many situations
  - Fill missing data
  - Extract values at certain points
- ‘scipy.interpolate’ has several methods
  - ‘interp1d is a workhorse
  - Linear, quadratic and cubic interpolation
  - You can not only interpolate but fill-in with previous value, next value as well
  - Akima1d and cubic spline are also good options for nonlinear interpolation

<u>Illustrative Example</u>

Use the empirical Gringorten Plotting Position Function to Estimate the Frequency of wells not meeting the drinking water quality standards in the Ogallala Aquifer of Texas

$$F_{emp}(x) = \frac{rank(x) - \alpha}{N + 1 - 2\alpha}$$

For Gringorten α = 0.44

https://docs.scipy.org/doc/scipy/reference/interpolate.html

Code

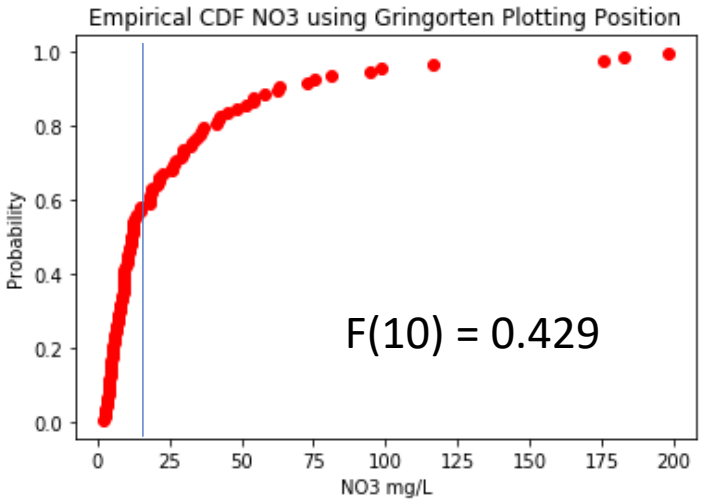
- Read NO3 data
- Write a function for Plotting positions
- Create Empirical CDF
- Interpolate

```
def plot_pos(X,alpha):
    """ plotting position using a pandas series X"""
    Xrank = X.rank()
    Xlen = X.size
    PP = (Xrank - alpha)/(Xlen+1-2*alpha)
    return PP
```

```
# Read Ogallala Data csv file
a = pd.read_csv('OgallalaData.csv')
NO3 = a.NO3Av # Pull out average NO3
NO3rank = NO3.rank()
NO3F = plot_pos(NO3,0.44) # Gringorten Plotting Position

# Make plot
plt.plot(NO3,NO3F,'ro')
plt.xlabel('NO3 mg/L')
plt.ylabel('Probability')
plt.title('Empirical CDF NO3 using Gringorten Plotting Position')

# Interpolate
NO3 = list(NO3) #convert pandas object to a list
NO3F= list(NO3F) # Convert pandas object to a list
f = intp.interp1d(NO3,NO3F) # Create an interpolation function
F10 = f(10.0) #Interpolate probability at a value of 10
Exceed = round(1 - F10,3)
```



Exceedance Risk = 0.571

Nearly 57% of the wells in the region are likely to show NO3 contamination

Bivariate Interpolation

- Bivariate interpolation can be carried out using many methods
  - Generally works for smooth functions
    - When data are sampled over a grid
- R does a much better job than Python for interpolating randomly sampled datasets
  - Recommend using packages in R for this rather than use scipy functionality for 2D interpolation
    - See akima package in R

Kernel Density Estimation (KDE)

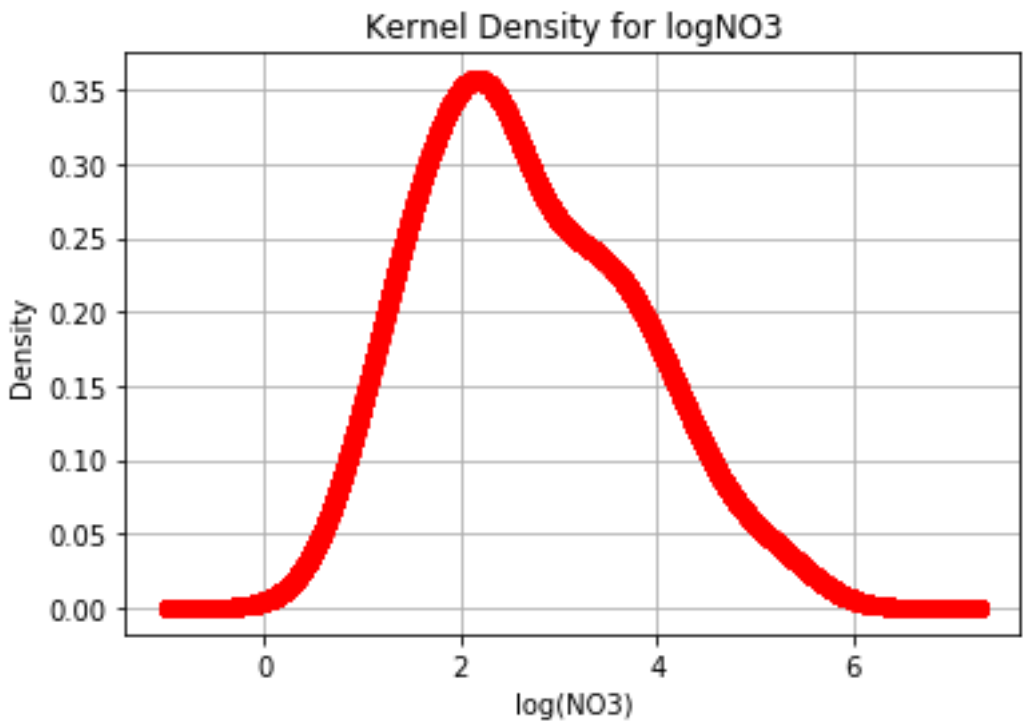
- KDE is a nonparametric approach to fitting probability density function to a data
  - A Histogram is a simple KDE
- Scipy offers a KDE function with Gaussian functions
- Both single and multidimensional KDEs can be used
- By default it uses Scott's rule for bandwidth
  - Other plug-in estimators can be used
- The KDE estimation in SciKit_learn package offers better functionality
  - We shall revisit this topic again

Several other packages do KDEs as well – see - https://jakevdp.github.io/blog/2013/12/01/kernel-density-estimation/

KDE

- Fit the KDE for Average NO3 concentration data
  - Log-transformation works better

```
from scipy import stats as st
NO3 = np.log(a.NO3Av) #extract data and log transform
NO3e = np.arange(-1,(max(NO3)+2),0.0005) #create eval pts
NO3kde = st.gaussian_kde(NO3) # Fit Gaussian KDE with
defaults
NO3d = st.gaussian_kde.evaluate(NO3kde,NO3e) # Get
densities at eval pts
# Make a KDE density plot
plt.plot(NO3e,NO3d,'ro')
plt.xlabel('log(NO3)')
plt.ylabel('Density')
plt.title('Kernel Density for logNO3')
plt.grid()
# Integrate to see if the AUC is one
NO3f = NO3kde.integrate_box_1d(-1,7) #0.999999944
```



Integration

- Integration, especially over a range of values is another important calculation that you will encounter in Machine Learning
  - Integrate a PDF to obtain CDF

- Scipy offers 3 options
  - Integration of a function with one variable
  - Integration of a function with more than one variable (Higher dimension integration)
  - 1D integration of data

Integration (scipy.integrate)

The scipy.integrate sub-package provides several integration techniques including an ordinary differential equation integrator. An overview of the module is provided by the help command:

```
>>> help(integrate)
 Methods for Integrating Functions given function object.

   quad          -- General purpose integration.
   dblquad       -- General purpose double integration.
   tplquad       -- General purpose triple integration.
   fixed_quad    -- Integrate func(x) using Gaussian quadrature of order n.
   quadrature    -- Integrate with given tolerance using Gaussian quadrature.
   romberg       -- Integrate func using Romberg integration.

 Methods for Integrating Functions given fixed samples.

   trapz         -- Use trapezoidal rule to compute integral from samples.
   cumtrapz      -- Use trapezoidal rule to cumulatively compute integral.
   simps         -- Use Simpson's rule to compute integral from samples.
   romb          -- Use Romberg Integration to compute integral from
                    (2**k + 1) evenly-spaced samples.

   See the special module's orthogonal polynomials (special) for Gaussian
      quadrature roots and weights for other weighting factors and regions.

 Interface to numerical integrators of ODE systems.

   odeint        -- General integration of ordinary differential equations.
   ode           -- Integrate ODE using VODE and ZVODE routines.
```

Numerical Integration

- Numerical integration using Newton-Coates formula is generic
  - Can be used with both functions and data so I will illustrate it here
  - If you have a function simply evaluate the function over a range of values to create X and Y dataset to integrate
- Newton-Coats formulas are only provided for 1D integration but you can repeatedly use the formulas to perform higher order integration

```
Methods for Integrating Functions given fixed samples.

  trapz         -- Use trapezoidal rule to compute integral from samples.
  cumtrapz      -- Use trapezoidal rule to cumulatively compute integral.
  simps         -- Use Simpson's rule to compute integral from samples.
  romb          -- Use Romberg Integration to compute integral from
                   (2**k + 1) evenly-spaced samples.
```

Example

- Calculate the expected value of NO3 using the nonparametric KDE function

$$E[x] = \int_{-\infty}^{+\infty} x.f(x)dx$$

Let us work with the log-transformed data to integrate and take the antilog to get the expected value

Expected NO3 concentration is 14.59 mg/L

```
# Code for computing KDE
from scipy import stats as st
NO3 = np.log(a.NO3Av) #extract data and log transform
NO3e = np.arange(-1,(max(NO3)+2),0.0005) #create eval pts
NO3kde = st.gaussian_kde(NO3) # Fit Gaussian KDE with
defaults
NO3d = st.gaussian_kde.evaluate(NO3kde,NO3e) # Get
densities at eval pts
# Make a KDE density plot
plt.plot(NO3e,NO3d,'ro')
plt.xlabel('log(NO3)')
plt.ylabel('Density')
plt.title('Kernel Density for logNO3')
plt.grid()
# Integrate to see if the AUC is one
NO3f = NO3kde.integrate_box_1d(-1,7) #0.999999944
# Code for Integration
from scipy import integrate as intg
num = NO3e*NO3d # Multiply x and f(x)
elx = intg.simps(num,NO3e) # Put f(x) and x
ENO3 = np.exp(elx) # 14.59 mg/L
```

Two-Dimensional Integration

- The joint distribution of two random variables is given as:

$$f(x,y) = \frac{1}{2\pi\sigma^2} exp\left(-\frac{(x^2+y^2)}{2\sigma^2}\right)$$

- Compute the cumulative distribution function (CDF) value F(x ≤ 10, y ≤ 20)
  - Assume $\sigma^2 = 10$

$$F(x \leq 10, y \leq 20) = \int_{-\infty}^{20}\int_{-\infty}^{10} f(x,y)dxdy$$

Fxa = 0.9992

One can perform double integrals directly using the dblquad function

```
# Get integration module from scipy
from scipy import integrate as intg
# Define the function
def funcpdf(y,x,ssq=10): #y should be first
    num1 = 1/(2*np.pi*ssq)
    num2 = (x**2+y**2)/(2*ssq)
    fx = num1 * np.exp(-num2)
    return fx
ssqa = 10
Fxa = intg.dblquad(funcpdf, -10000, 10,
lambda y: -10000, lambda y:
20,args=(ssqa,)
```

Inside integral (x)

Integration limits of y

Notice the argument ssqa needs to be passed as a tuple so (ssqa,)

Two-Dimensional Integration

- You can also do two-dimensional integration using repeated application of the Trapezoidal Rule

	x1	x2	---	xn
y1	f(y1,x1)	f(y1,x2)	---	f(y1,xn)
y2				
---				
yk	f(yk,x1)	f(yk,X2)	---	f(yk,xn)
Result of Int(y,f(y,xi))	Int(y,f(y,x1))	Int(y,f(y,x2))	---	Int(y,f(y,xn))

Step 1: Integrate Each column of f(x,y) wrt to y

Step 2: Integrate the result row wrt to x

Two-Dimensional Integration

- Note this process is time-intensive
- It is sensitive to how the x and y are discretized (sequence step length)

```
import numpy as np # Import numpy
Import integrate from scipy as intg  # integration functions from scipy
# Example of a double integral
def funcpdf(y,x,ssq=10):
   num1 = 1/(2*np.pi*ssq)
   num2 = (x**2+y**2)/(2*ssq)
   fx = num1 * np.exp(-num2)
   return fx

x = np.arange(-2500,10.1,0.1) # create sequence of X
y = np.arange(-2500,20.1,0.1) # Create a sequence of y
xx,yy = np.meshgrid(x,y) # use meshgrid to get combinations of x and y
fxy = funcpdf(yy,xx,ssqa) # comput fxy for all xx and yy

intx = np.zeros(len(x)) # create a vector of zeros to store integrals from 1st step
idx = 0 # Initialize the index
for i in x:  # Look through each column and integrate wrt y (1st stage)
   fxa = fxy[:,idx]
   intx[idx] = intg.trapz(y,fxa)
   idx = idx + 1

Fxy = intg.trapz(x,intx)  # Second stage integration
```

$\boxed{Fxy = 0.99072}$

Note Looping in Python is not optimal and the code takes some time to execute

What You Should Know

- Interpolation using scipy
- Integration using scipy
- Scipy has other important numerical methods such as
  - Linear Algebra (same as numpy)
  - Special Functions
  - Optimization
  - Fourier Transforms
  - Spatial algorithms

SciPy

Release: 1.4.1
Date: December 19, 2019

SciPy (pronounced "Sigh Pie") is open-source software for mathematics, science, and engineering.

- Installing and upgrading
- SciPy API
- Release Notes

Tutorial

Tutorials with worked examples and background information for most SciPy submodules.

- SciPy Tutorial
  - Introduction
  - Basic functions
  - Special functions (**scipy.special**)
  - Integration (**scipy.integrate**)
  - Optimization (**scipy.optimize**)
  - Interpolation (**scipy.interpolate**)
  - Fourier Transforms (**scipy.fft**)
  - Signal Processing (**scipy.signal**)
  - Linear Algebra (**scipy.linalg**)
  - Sparse eigenvalue problems with ARPACK
  - Compressed Sparse Graph Routines (**scipy.sparse.csgraph**)
  - Spatial data structures and algorithms (**scipy.spatial**)
  - Statistics (**scipy.stats**)
  - Multidimensional image processing (**scipy.ndimage**)
  - File IO (**scipy.io**)

We shall explore a few more of scipy functionalities in the coming lectures