Python Libraries for Data Mining

Ishwar Sethi

CSI5810 Fall 2020

Python Libraries for DM

- There are four highly useful libraries for data mining. These are:
  - **NumPy** for computation with multidimensional arrays/linear algebra
  - **Pandas** for high-performance data structure, DataFrame, for data manipulation, data importing and exporting, and plotting
  - **Matplotlib** is a comprehensive library for creating static, animated, and interactive visualizations in Python.
  - **Scikit-learn** is a machine learning library for building data mining models. Its built from NumPy, SciPy, and Matplotlib. SciPy is a python library for statistical computations.

- There are similar libraries in R for doing data mining work.

NumPy

- NumPy, an abbreviation for Numerical Python, is the core library for scientific computing with Python.
- It provides support for creating and processing N-dimensional array objects or ndarrays.
- It offers linear algebra operations that are much faster than those performed using traditional Python.
- It is a must for any data analysis task.
- The following slides show some basic functionality of NumPy.

NumPy Basics

```
import numpy as np
mat1 = np.array([[1, 2, 3],[4, 5, 6],[7, 8, 9]])# Creates a 2-dim array
mat2 = np.random.randint(12, size=(3,4))# Creates a 3x4 array of random integers
```

```
print(mat1,'\n\n',mat2)
```

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

```
# Array dimension, shape, size, and data type.
```

```
print(" mat2 ndim: ", mat1.ndim,'\t', "mat2 shape: ", mat2.shape,'\t',"mat2 size: ", mat2.size)
```

```
 mat2 ndim:  2   mat2 shape:  (3, 4)     mat2 size:  12
```

```
print("mat2 dtype: ", mat2.dtype)
```

```
mat2 dtype:  int64
```

```python
# Array Indexing & Slicing
```

```python
mat2[0,1] # Element at Ist row, second column intersection
```

```
7
```

```python
mat2[0,-1] # Element at Ist row, last column intersection
```

```
8
```

```python
mat2[:2,:3] # Sliced array with first 2 rows and 3 columns
```

```
array([[ 5,  7,  0],
       [11,  3,  8]])
```

```python
mat2[:2, ::2] # First 2 rows and every second column
```

```
array([[ 5,  0],
       [11,  8]])
```

Array reshaping and concatenation

```
mat2.reshape((2,6))# reshapes a 3x4 array to a 2x6 array
```

```
array([[ 5,  7,  0,  8, 11,  3],
       [ 8,  3,  0,  5, 10,  3]])
```

```
np.concatenate([mat1,mat2],axis=1)
```

```
array([[ 1,  2,  3,  5,  7,  0,  8],
       [ 4,  5,  6, 11,  3,  8,  3],
       [ 7,  8,  9,  0,  5, 10,  3]])
```

```
np.concatenate([mat1, mat2.reshape(4,3)])
```

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [ 5,  7,  0],
       [ 8, 11,  3],
       [ 8,  3,  0],
       [ 5, 10,  3]])
```

```
# You can also use hstack and vstak
np.hstack([mat1,mat2])
```

```
array([[ 1,  2,  3,  5,  7,  0,  8],
       [ 4,  5,  6, 11,  3,  8,  3],
       [ 7,  8,  9,  0,  5, 10,  3]])
```

```
np.vstack([mat1,mat2.reshape(4,3)])
```

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [ 5,  7,  0],
       [ 8, 11,  3],
       [ 8,  3,  0],
       [ 5, 10,  3]])
```

```python
#Splitting an array
np.split(mat2,[2])
```

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

```python
np.hsplit(mat2,[2])
```

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

```
# NumPy's universal functions utilize vectorized operations which are fast.
# So avoid using loops as much as possible.
# Some examples of the universal functions are given below.
```

```
np.max(mat2)
```

```
11
```

```
np.cos(mat2)
```

```
array([[ 0.28366219, -0.91113026, -0.65364362, -0.14550003],
       [ 0.0044257 , -0.65364362,  0.54030231, -0.14550003],
       [ 0.28366219, -0.65364362, -0.9899925 , -0.9899925 ]])
```

```
np.add.reduce(mat2)# Reduces the 3x4 mat2 array to 1x4 by adding elements of each column
```

```
array([21, 17,  8, 19])
```

```
np.multiply.reduce(mat2)# Same as above but elements are multiplied
```

```
array([275, 144,  12, 192])
```

```
np.sum(mat2)# Adds all elements of the array
```

```
65
```

```python
# Operations with vectors and matrices
```

```python
mat1*mat1#* operation does element by element multiplication
```

```
array([[ 1,  4,  9],
       [16, 25, 36],
       [49, 64, 81]])
```

```python
mat1.T# matrix transpose
```

```
array([[1, 4, 7],
       [2, 5, 8],
       [3, 6, 9]])
```

```python
mat1@mat2# matrix multiplication mat1 is a 3x3 matrix, mat2 is a 3x4. The result is 3x4
```

```
array([[ 42,  29,  15,  33],
       [105,  80,  39,  90],
       [168, 131,  63, 147]])
```

```python
np.dot(mat1[:,0],mat1[:,1])# dot product of two vectors
```

```
78
```

```python
# Inverse of a matrix. To perform this and other linear algebra operations
# we need to use linear algebra library linalg
np.linalg.inv(mat1)
```

```
array([[-4.50359963e+15,  9.00719925e+15, -4.50359963e+15],
       [ 9.00719925e+15, -1.80143985e+16,  9.00719925e+15],
       [-4.50359963e+15,  9.00719925e+15, -4.50359963e+15]])
```

```python
np.linalg.matrix_rank(mat1)
```

```
2
```

```python
s, V = np.linalg.eig(mat1)
print("Eigenvalues: ", s)
print("Eigenvectors: ",V)
```

```
Eigenvalues:  [ 1.61168440e+01 -1.11684397e+00 -1.30367773e-15]
Eigenvectors:  [[-0.23197069 -0.78583024  0.40824829]
 [-0.52532209 -0.08675134 -0.81649658]
 [-0.8186735   0.61232756  0.40824829]]
```

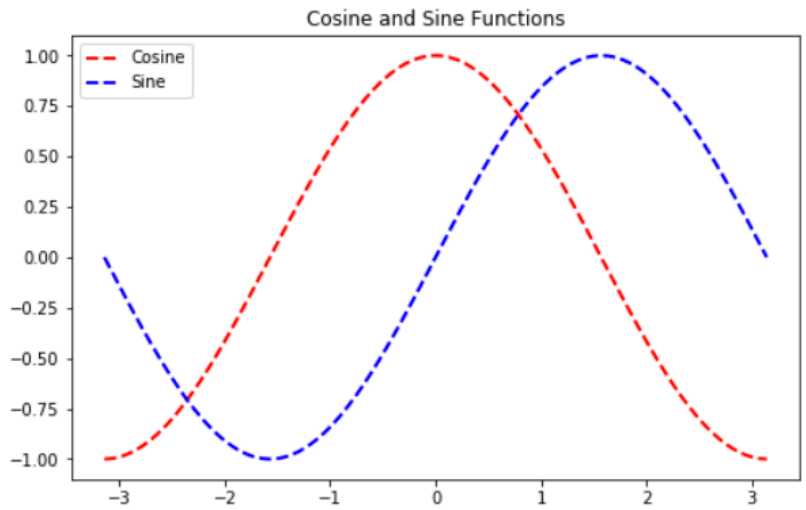
Matplotlib

- Matplotlib is a popular library for visualizing data represented by numpy arrays. Pyplot is a Matplotlib module; it provides a MATLAB-like interface to create and display a variety of plot types. An example of plotting is shown in the next slide. For more, see the tutorial on Matplotlib. https://matplotlib.org/tutorials/

```python
# Matplotlib is a Python package to do 2-D graphics
# Pyplot provides a convenient interface to matplotlib
# It is closely modeled after Matlab.
# Matplotlib works with NumPy objects.
# Ploting Example
```

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

```python
X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C, S = np.cos(X), np.sin(X)
plt.figure(figsize=(8,5))
plt.plot(X, C, c="red", linewidth= 2.0, linestyle="--", label="Cosine")
plt.plot(X, S, c="blue", linewidth= 2.0, linestyle="--", label="Sine")
plt.legend(loc='upper left')
plt.title('Cosine and Sine Functions')
plt.show()
```

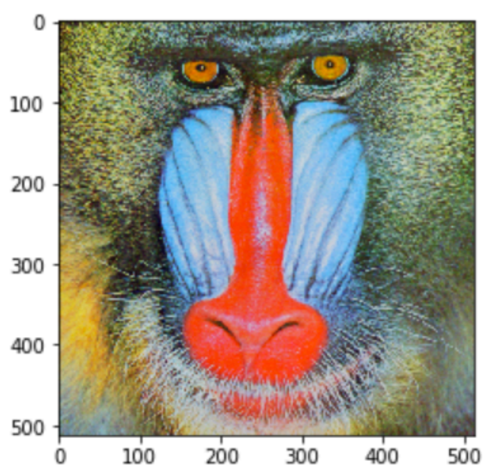


```python
from PIL import Image
img = Image.open('mendrill.jpg')
img.show()
```

```python
mat2 = np.asarray(img)# Converts image to a numpy array
```

```python
import matplotlib.pyplot as plt
plt.imshow(mat2)
```

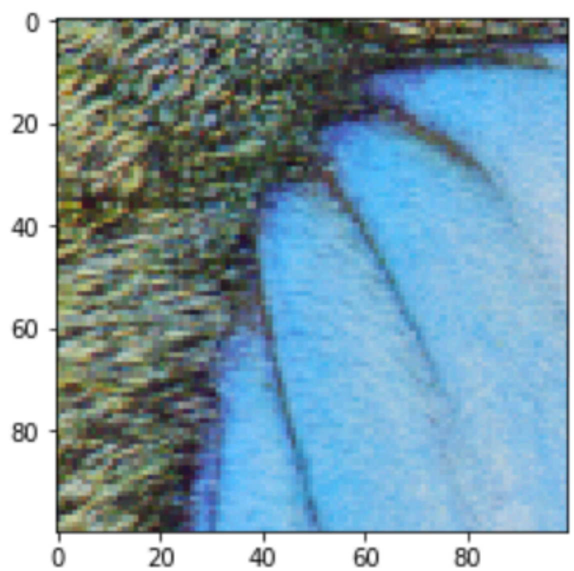
```
<matplotlib.image.AxesImage at 0x7f9d1882d6a0>
```



Reading and displaying images

```
mat3=mat2[100:200,100:200]# just look at an image patch
plt.imshow(mat3)
```

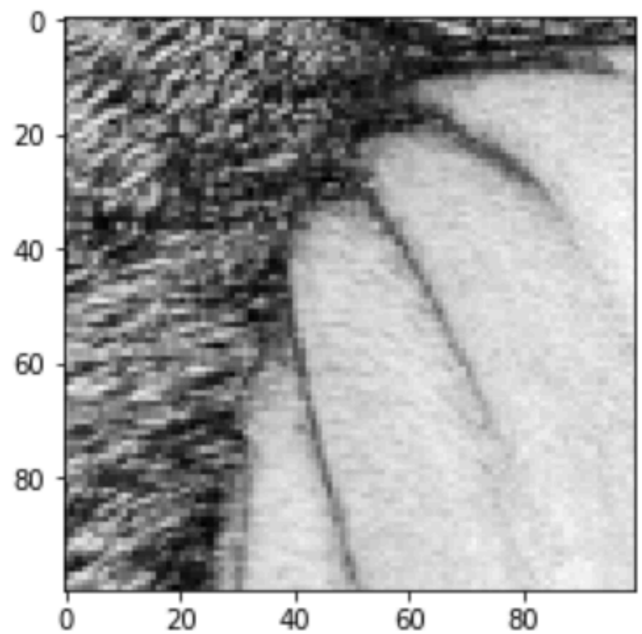
```
<matplotlib.image.AxesImage at 0x7f9cd8b5e940>
```




```
mat4 = mat3[:,:,0]# Just take the red plane
```

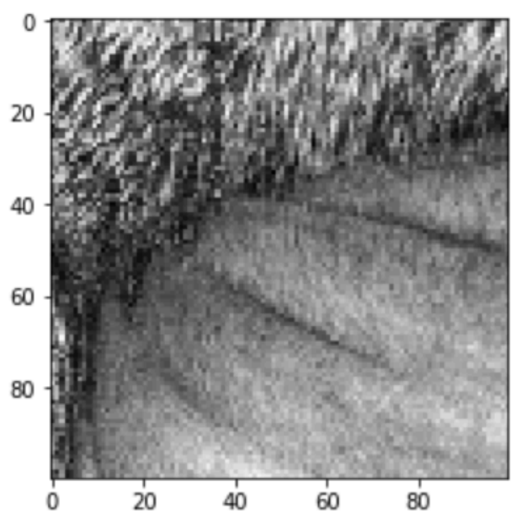
```
plt.gray()
plt.imshow(mat4)
```

```
<matplotlib.image.AxesImage at 0x7f9d08f1b2b0>
```



```
mat5 = mat4.T# Perform matrix transpose
plt.imshow(mat5)
```

<matplotlib.image.AxesImage at 0x7facf02d0ef0>

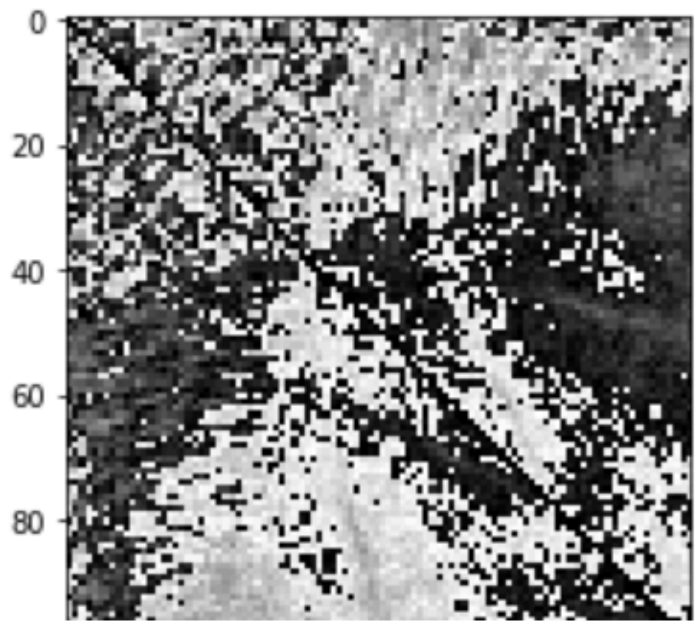



```
np.trace(mat4)
```

11467

```python
mat5 = mat4 - mat4.T
plt.imshow(mat5)
```

```
<matplotlib.image.AxesImage at 0x7facc09342e8>
```



Pandas

```
# pandas introduces two new data structures to Python - Series and DataFrame,
# both of which are built on top of NumPy.
Series: A series is a one-dimensional object similar to an array or a list.
By default, every item in the series has an index associated with it. The index need not be
integer
```

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

```
data = pd.Series(['apple', 5, '%Rate', 6, -3.33 ])
data
```

```
0    apple
1        5
2    %Rate
3        6
4    -3.33
dtype: object
```

```
# We can access values and index attributes of a series
data.values
```

```
array(['apple', 5, '%Rate', 6, -3.33], dtype=object)
```

```
data.index
```

```
RangeIndex(start=0, stop=5, step=1)
```

```
# We can explicitly assign index to values in the series
data =  pd.Series(['apple', 5, '%Rate', 6, -3.33 ],
                  index = ['a', 'b','c','d','e'])
data
```

```
a    apple
b        5
c    %Rate
d        6
e    -3.33
dtype: object
```

```
data['b']
```

```
5
```

```python
# A series object can be constructed directly from a Python dictionary
population_dict = {'California': 38332521,
                   'Texas': 26448193,
                   'New York': 19651127,
                   'Florida': 19552860,
                   'Illinois': 12882135}
population = pd.Series(population_dict)
population
```

```
California    38332521
Texas         26448193
New York      19651127
Florida       19552860
Illinois      12882135
dtype: int64
```

```python
# By default, the Series is created where the index is drawn from the sorted keys.
population['Florida']
```

```
19552860
```

```python
# We can use dictionary-like Python expressions and methods to look at key-value pairs
population.keys()
```

```
Index(['California', 'Texas', 'New York', 'Florida', 'Illinois'], dtype='object')
```

```python
list(population.items())
```

```
[('California', 38332521),
 ('Texas', 26448193),
 ('New York', 19651127),
 ('Florida', 19552860),
 ('Illinois', 12882135)]
```

DataFrame
A DataFrame is a tabular data structure comprised of rows and columns. It can also be considered a sequence of aligned Series objects.

```
area_dict = {'California': 423967, 'Texas': 695662, 'New York': 141297,
             'Florida': 170312, 'Illinois': 149995}
area = pd.Series(area_dict)
area
```

```
California    423967
Texas         695662
New York      141297
Florida       170312
Illinois      149995
dtype: int64
```

```
states = pd.DataFrame({'Population': population, 'Area': area})
states
```

	Population	Area
California	38332521	423967
Texas	26448193	695662
New York	19651127	141297
Florida	19552860	170312
Illinois	12882135	149995

```
# A single row of the DataFrame is extracted as a Series
states.loc["New York"]
```

```
Population    19651127
Area            141297
Name: New York, dtype: int64
```

```
# Extracting a subset of the DataFrame object
states.loc["New York":"Florida", "Area"]
```

```
New York    141297
Florida     170312
Name: Area, dtype: int64
```

```python
# Reading data to create DataFrame
# You read data in many formats
# CSV, Excel, sql, and JSON being most popular
```

```python
df = pd.read_csv("Cars2015.csv")
```

```python
df
```

	Make	Model	Type	LowPrice	HighPrice	Drive	CityMPG	HwyMPG	FuelCap	Length	Width	Wheelbase	Height	UTurn	Weight	Acc030	Acc060
0	Chevrolet	Spark	Hatchback	12.270	25.560	FWD	30	39	9.0	145	63	94	61	34	2345	4.4	12.8
1	Hyundai	Accent	Hatchback	14.745	17.495	FWD	28	37	11.4	172	67	101	57	37	2550	3.7	10.3
2	Kia	Rio	Sedan	13.990	18.290	FWD	28	36	11.3	172	68	101	57	37	2575	3.5	9.5
3	Mitsubishi	Mirage	Hatchback	12.995	15.395	FWD	37	44	9.2	149	66	97	59	32	2085	4.4	12.1
4	Nissan	Versa Note	Hatchback	14.180	17.960	FWD	31	40	10.9	164	67	102	61	37	2470	4.0	10.9
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
105	Toyoto	Sequioa	7Pass	44.395	64.320	RWD	12	18	26.4	205	80	122	75	42	6025	2.7	7.1
106	Nissan	Pathfinder	7Pass	29.510	43.100	FWD	19	25	19.5	192	73	112	72	40	4505	3.2	7.7
107	Acura	MDX	7Pass	42.865	57.080	FWD	18	27	19.5	194	77	111	68	40	4200	3.0	7.2
108	Hyundai	Santa Fe	7Pass	30.150	36.000	FWD	18	24	19.0	193	74	110	67	39	4210	3.0	7.6
109	GMC	Yukon	7Pass	47.740	67.520	RWD	16	22	26.0	204	81	116	74	41	5635	2.8	7.7

110 rows × 17 columns

```python
df.describe() # Generates a summary statistics of the data
```

	LowPrice	HighPrice	CityMPG	HwyMPG	FuelCap	Length	Width	Wheelbase	Height	UTurn	Weight	Acc030	Acc060
count	110.000000	110.000000	110.000000	110.000000	110.000000	110.000000	110.000000	110.000000	110.000000	110.000000	110.00000	110.000000	110.000000
mean	32.808082	49.124309	20.781818	29.363636	18.004545	187.281818	73.281818	110.154545	61.427273	39.063636	3846.00000	3.069091	7.937273
std	15.926386	28.196937	4.546158	5.536745	4.374224	14.468017	3.629977	7.782816	6.602000	2.335515	867.49608	0.553843	1.645169
min	12.270000	15.395000	12.000000	18.000000	9.000000	145.000000	63.000000	92.000000	49.000000	32.000000	2085.00000	1.600000	4.100000
25%	21.720000	31.182500	17.000000	25.000000	14.650000	179.000000	71.000000	106.000000	57.000000	38.000000	3245.00000	2.700000	6.800000
50%	29.767500	42.010000	20.000000	28.000000	18.500000	190.000000	73.000000	110.000000	58.000000	39.000000	3772.50000	3.050000	7.900000
75%	42.075000	62.221250	24.000000	34.000000	19.725000	197.000000	76.000000	115.000000	66.000000	40.000000	4307.50000	3.400000	8.950000
max	84.300000	194.600000	37.000000	44.000000	33.500000	224.000000	81.000000	131.000000	79.000000	45.000000	6265.00000	4.400000	12.800000

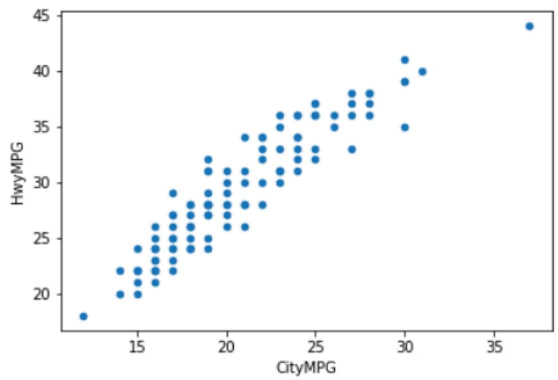
```python
# You can also aggregate statistics grouped by category
# For example, we may want to know average CityMPG for different car types
df[["Type", "CityMPG"]].groupby("Type").mean()
```

	CityMPG
Type	
7Pass	16.800000
Hatchback	28.363636
SUV	18.888889
Sedan	21.565217
Sporty	18.818182
Wagon	20.333333

```
# Pandas has a built in .plot() function as part of the DataFrame class.
# You can use this to visualize a df object
```

```
df.plot(x='CityMPG',y='HwyMPG',kind='scatter')
```

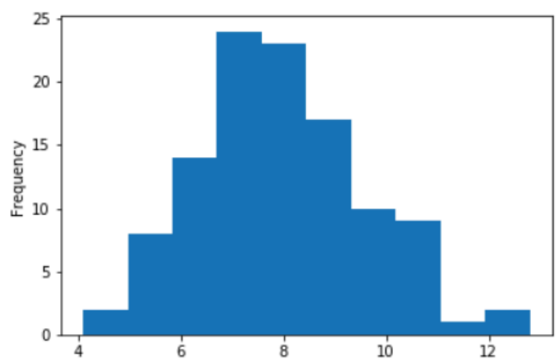
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9600456910>
```



The plot function uses the Matplotlib, discussed earlier.

```
df['Acc060'].plot.hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f96005dd890>
```



More on pandas at https://pandas.pydata.org/pandas-docs/stable/index.html

SciPy

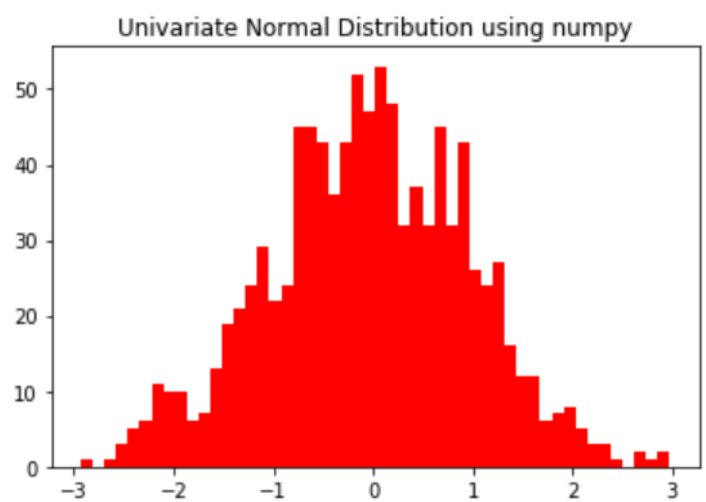
- The SciPy package contains various toolboxes to perform computations optimization, image processing, statistics, signal processing etc.
- It is designed to operate efficiently on numpy arrays.
- In the next few slides, some examples of scipy use are shown.

https://scipy-lectures.org/intro/scipy.html#linear-algebra-operations-scipy-linalg

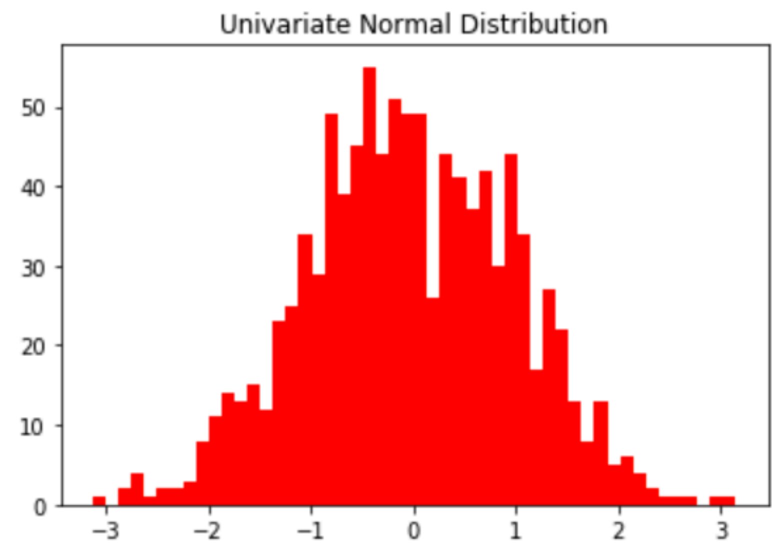
Generating Random Variables in Python

```
import scipy.stats
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(4321)# Fixed for reproducability
```

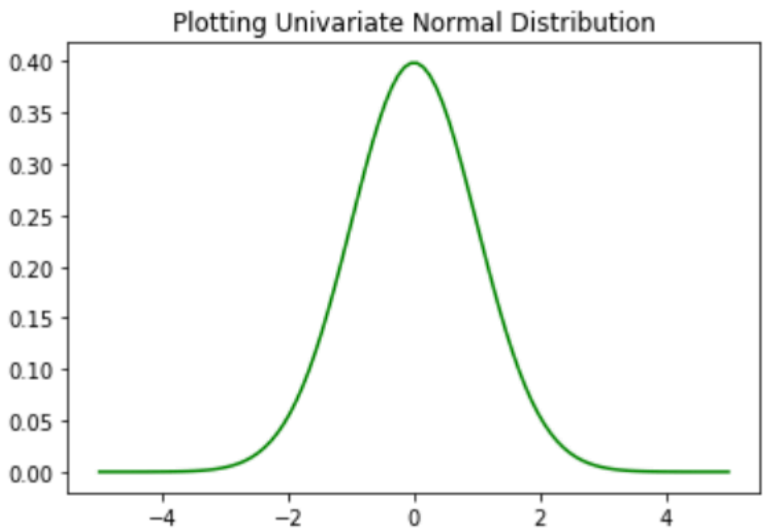
```
samples = np.random.normal(size=1000)
num_bins = 50
plt.hist(samples,50, color='red')
plt.title('Univariate Normal Distribution using numpy')
plt.show()
```




```
samples = scipy.stats.norm.rvs(size=1000)
num_bins = 50
plt.hist(samples,50, color='red')
plt.title('Univariate Normal Distribution using Scipy')
plt.show()
```



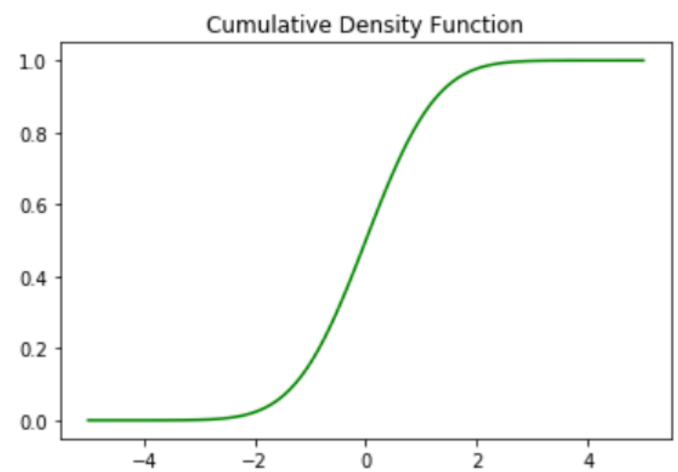
```python
x = np.linspace(-5,5,100)
y = scipy.stats.norm.pdf(x)
plt.plot(x,y, color='green')
plt.title('Plotting Univariate Normal Distribution')
plt.show()
```




```python
# Calculating the probability of x <2.0
prob = scipy.stats.norm.cdf(1.65, loc = 0, scale = 1)
print(prob)
```

```
0.9505285319663519
```

```python
z = scipy.stats.norm.cdf(x)
plt.plot(x,z, color='green')
plt.title('Cumulative Density Function')
plt.show()
```

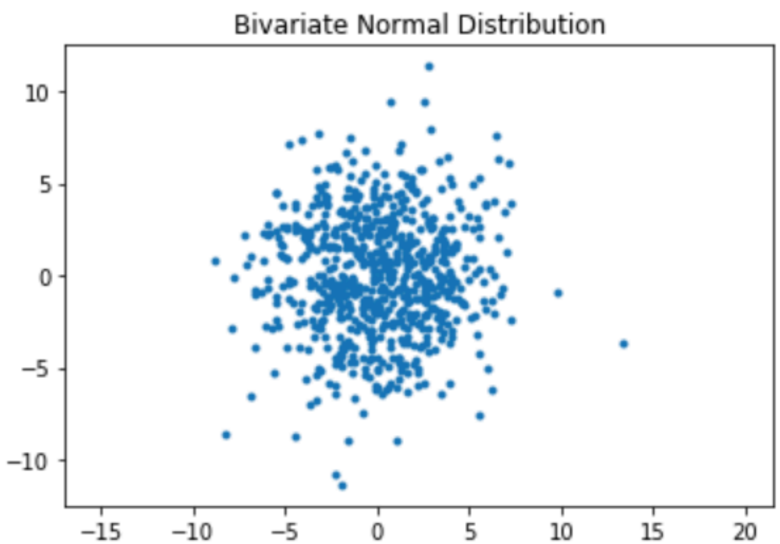



```python
location, scale = scipy.stats.norm.fit_loc_scale(samples)
# Fit normal distribution to 1000 data points
print(location,scale)
```

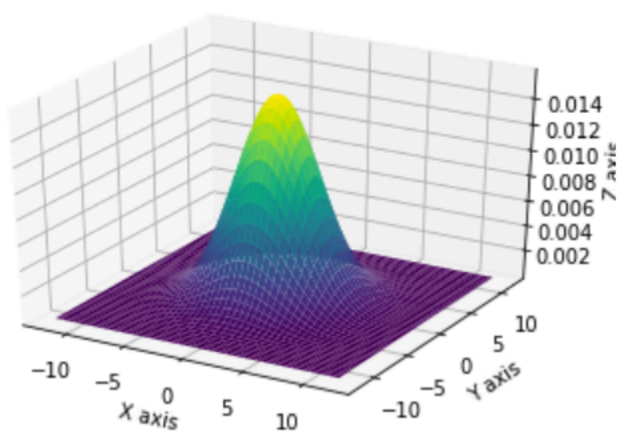
```
-0.04477094580940223 1.0037883177758524
```

Bivariate Normal Distribution

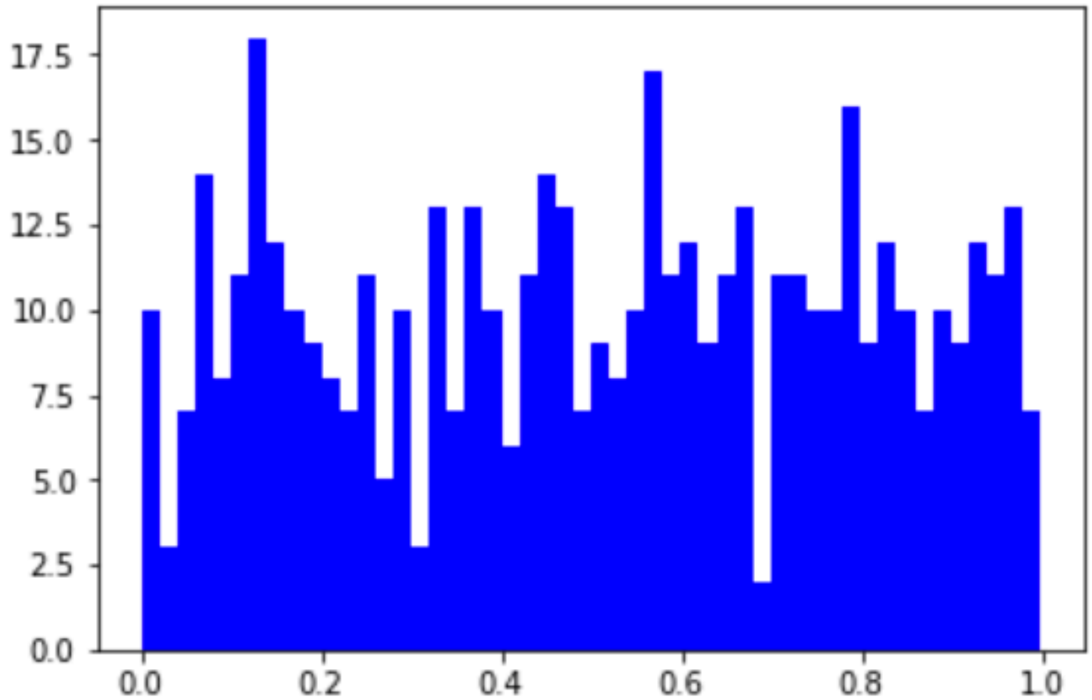
```
#Bivariate Normal Distribution
mean = [0, 0]
cov = [[10, 0], [0, 10]]  # diagonal covariance
x, y = np.random.multivariate_normal(mean, cov, 700).T
plt.plot(x, y, '.')
plt.axis('equal')
plt.title('Bivariate Normal Distribution')
plt.show()
```



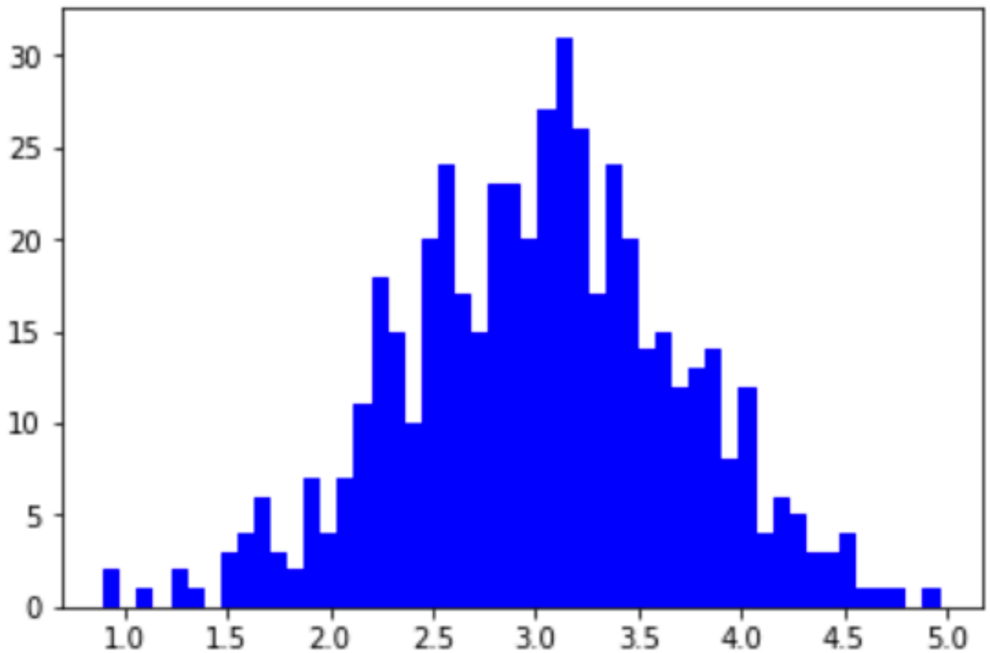
```
from mpl_toolkits.mplot3d import Axes3D
delta = 0.05
x1 = np.arange(-12.0,12.0,delta)
y1 = np.arange(-12.0,12.0,delta)
X, Y = np.meshgrid(x1,y1)
pos = np.empty(X.shape + (2,))
pos[:, :, 0] = X; pos[:, :, 1] = Y
rv = multivariate_normal(mean, cov)
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(X, Y, rv.pdf(pos),cmap='viridis',linewidth=0)
ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_zlabel('Z axis')
plt.show()
```



```python
# Central Limit Theorem Demo
from scipy.stats import uniform
r1 = uniform.rvs(size=500)
plt.hist(r1,50, color='blue')
plt.show()
```



```
rsum = 0
for k in range(6):
    rsum = rsum + uniform.rvs(size=500)
plt.hist(rsum,50, color='blue')
plt.show()
```



Scikit Learn

- Scikit-learn (Sklearn) is an extremely useful library for machine learning in Python. It comes with numerous built-in machine learning models that can be trained with user's data. It also comes with many well-known machine learning data sets. The library, largely written in Python, is built upon NumPy, SciPy and Matplotlib.
- You will be using this library in some of your work.
- Book mark the following link for your reference and use.

  https://scikit-learn.org/stable/tutorial/index.html