Python – Biblioteci pentru Data Science

În domeniul Data Science, utilizatorii au nevoie de următoarele operații:

- Obținerea datelor
- Manipularea și procesarea datelor
- Vizualizarea rezultatelor
- Comunicarea rezultatelor.

Există colecții bogate de metode numerice ce pot fi utilizate, fără a fi nevoie de rescrierea unor operații simple (crearea unui grafic).

În limbaje precum C/C++ există astfel de biblioteci (optimizate și foarte rapide), dar sunt dificil de utilizat datorită lipsei de interactivitate în timpul dezvoltării, managementului manual al memoriei etc.

Limbaje științifice precum Matlab prezintă avantaje datorită colecțiilor numeroase de algoritmi implementați pentru rezolvarea diferitelor tipuri de probleme, interfață prietenoasă, însă licența este costisitoare.

Alte limbaje open source, cum sunt R, Octave, Scilab pot înlocui o mare parte din bibliotecile din Matlab.

Python beneficiază de biblioteci pentru calcul științific, codul poate fi bine structurat și ușor de înțeles. În plus, oferă și biblioteci pentru alte taskuri, cum este managementul webserverelor. Interfața de lucru este mai puțin prietenoasă decât în cazul Matlab, dar este open source, cu o comunitate online puternică.

Bibliotecile științifice utilizate pentru Python care trebuie instalate:

Nota: Daca utilizati Google Colab bibliotecile sunt deja instalate, trebuie doar importate.

Numpy – obiecte numerice pentru calcul matricial

Instalare:

Comanda prin intermediul pip in cmd:

```
python -m pip install --user numpy
```

Va fi descarcata si instalata varianta potrivita a bibliotecii in functie de versiunea python si sistemul de operare instalate.

Scipy – Calcule științifice avansate

Instalare:

Comanda prin intermediul pip in cmd:

```
python -m pip install --user scipy
```

Matplotlib – Grafice 2-D

Instalare:

```
python -m pip install --user matplotlib
```

Documentație: http://docs.scipy.org/

Utilizarea Numpy

- Extinde mediul Python standard către array-uri multi-dimensionale
- Eficiență mai mare în calcule
- Orientat către calcul matricial

```
import numpy as np   # conventie pentru importul numpy
a = np.array([0, 1, 2, 3]) # crează un array
unidimensional
b = np.array([[0, 1, 2], [3, 4, 5]]) # 2 x 3 array
```

Exercițiu

Scrieți exemplele de mai sus și utilizați funcțiile len(), numpy.shape() și atributul ndim.

Moduri de a crea matrici cu ajutorul numpy:

```
a = np.arange(10) # 0 .. n-1
b = np.arange(1, 9, 2) # start, end (exclusive), step
c = np.linspace(0, 1, 6)   # start, end, num-points
d = np.linspace(0, 1, 5, endpoint=False)

e = np.ones((3, 3))
f = np.zeros((2, 2))
g = np.eye(3)
h = np.diag(np.array([1, 2, 3, 4]))

aa = np.random.rand(4)        # distributie uniforma in
[0, 1]
bb = np.random.randn(4)       # distributie Gaussiana
```

Exerciții

Utilizați funcțiile de mai sus si creati diverse structuri array. Verificați funcționalitatea *np.empty*.

Tipul default al datelor pentru numpy este float. Acesta poate fi explicit specificat în cadrul codului.

```
a = np.array([1, 2, 3])

b = np.array([1., 2., 3.])

c = np.array([1, 2, 3], dtype=float)

d = np.array([1+2j, 3+4j, 5+6*1j])

e = np.array([True, False, False, True])

f = np.array(['Bonjour', 'Hello', 'Hallo',])
```

```
g = np.ones((3, 3))
```

Folosiți atributul dtype pentru a verifica tipul variabilelor de mai sus și încercați să înțelegeți logica de lucru.

În calculele matriciale, prima dimensiune corespunde liniilor, iar a doua coloanelor.

```
a = np.diag(np.arange(3))
print(a)
a[2, 1] = 10
print(a)
```

Manipularea array-urilor:

```
a = np.arange(10)
print(a[2:9:3])  # [start:end:step]
print(a[::-1])
print(a[:4])
```

Exercitiu:

Utilizati combinatii ale parametrilor start, end si step pentru matricea rezultata din expresia:

```
x=np.arange(6) + np.arange(0, 51, 10)[:, np.newaxis]
```

Verificati functionarea functiei np.tile().

Copii și imagini

În Python operațiunea de atribuire nu funcționează la fel ca și în alte limbaje, ci creează o altă imagine a locației de memorie care a fost atribuită.

Testați următorul cod:

```
a = np.arange(10)
b = a[::2]
print(a, b)
b[0] = 12
```

```
print(b)
print(a)
```

Pentru a verifica dacă două variabile împart aceeași locație de memorie, putem utiliza funcția np.may_share_memory(a,b).

Testați următorul cod:

```
a = np.arange(10)
c = a[::2].copy()  # copiere fortata
c[0] = 12
print(c)
print(a)
print(np.may_share_memory(a, c))
```

Operații matematice de bază

```
a = np.array([1, 2, 3, 4])
print(a + 1) #aduna 1 element cu element
print(2**a) # inmulteste cu 2, element cu element
b= np.ones(4) + 1
print(a-b, a*b)
```

Înmulțirea nu este înmulțire de matrici. Pentru aceasta, se foloseste functia dot()

```
c = np.ones((3, 3))
print(c*c)
print(c.dot(c))
```

Comparații:

```
a = np.array([1, 2, 3, 4])
b = np.array([4, 2, 2, 4])
c = np.array([1, 2, 3, 4])
print(a==b)
print(a>b)
print(np.array_equal(a, b))
print(np.array_equal(a, c))
```

Transpunerea:

```
x=np.ones((4,4))
a = np.array([1, 2, 3, 4])
x[0,:]=a
print(x.T)
```

Exercitiu

Verificati functionalitatea functiilor `sum, min, max, argmin, argmax, any, all, mean, median, std, sort, argsort`

Utilizarea Matplotlib

Pyplot reprezintă interfața procedurală a matplotlib, bibliotecă orientată obiect.

```
import matplotlib.pyplot as plt
```

Grafice simple

Unidimensionale:

```
x = np.linspace(0, 3, 20)
y = np.linspace(0, 9, 20)
plt.plot(x, y)       # linie
plt.plot(x, y, 'o')  # puncte
plt.show()       # afisarea graficului
```

Bidimensionale:

```
image = np.random.rand(30, 30) # o matrice cu valori
random
plt.imshow(image, cmap=plt.cm.hot)    # un grafic in
care valorile precedente sunt ilustrate ca si culori
plt.colorbar() # scala de culori
```

```
plt.show()
```

Verificati functionalitatea functiei np.linspace()

```
X = np.linspace(-np.pi, np.pi, 256, endpoint=True) # un
array cu 256 valori de la –pi la +pi
C, S = np.cos(X), np.sin(X) # c este cosinus de X, iar
S este sinus de X.

plt.plot(X, C)
plt.plot(X, S)

plt.show()
```

Personalizarea graficului

```
plt.figure(figsize=(8, 6), dpi=80) # setarea
dimensiunii si a rezolutiei

plt.subplot(1, 1, 1)# crearea unui subgrafic

X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C, S = np.cos(X), np.sin(X)

plt.plot(X, C, color="blue", linewidth=1.0,
linestyle="-", label="cosinus") #setarea culorii
liniei, a grosimii si a stilului

plt.plot(X, S, color="green", linewidth=1.0,
linestyle="-", label="cosinus")
plt.xlim(-4.0, 4.0) # setarea limitelor axei X
plt.xticks(np.linspace(-4, 4, 9, endpoint=True)) #
setarea tickeri-lor pe axa X
plt.ylim(-1.0, 1.0)
plt.yticks(np.linspace(-1, 1, 5, endpoint=True))
plt.legend(loc='upper left')#pozitionarea legendei
plt.savefig("graphic_ex.png", dpi=72) # salveaza figura
```

```
plt.show()
```

Exercitii.

1. Studiati mai multe tipuri de grafice pe site-ul matplotlib.org.

2. **Temă:**
   In fisierul populatii.txt sunt descrise populațiile de iepuri, lincși și morcovi din Canada, pe o perioadă de 20 de ani.
   Plecând de la codul sursă de mai jos, calculați fără a folosi cicluri for/while:
   a) Media și variația standard pentru fiecare specie.
   b) Care specie are cea mai mare populație?
   c) In ce ani, orice populație este mai mare de 50 000? (np.any)
   d) Primii 2 ani pentru fiecare specie în care au avut cele mai mici populații (argsort)
   e) Realizati un grafic de comparație a populației de iepuri și lincși. Verificați corelația. (np.gradient, np.corrcoef)

```
import numpy as np
data = np.loadtxt('populatii.txt')
an, iepure, linx, morcov = data.T  # salveaza coloanele
ca si variabile

import matplotlib.pyplot as plt
plt.axes([0.2, 0.1, 0.5, 0.8])

plt.plot(an, iepure, an, linx, an, morcov)

plt.legend(('Iepure', 'Linx', 'Morcov'), loc=(1.05,
0.5))
plt.show()
```