TEFE 2017

Aula 3 - Geração de dados simulados - versão Python

suplemento elaborado por Danilo Lessa Bernardineli

Importando bibliotecas¶

No bloco seguinte, é feita a importação das principais bibliotecas utilizadas para a análise de dados no Python

---

```
import numpy as np # Biblioteca para manipulação númerica
import matplotlib.pyplot as plt # Biblioteca para visualização e gráficos
import scipy.stats as st # Biblioteca de funções estatísticas

# Configuração para mudar o tamanho das figuras ao usar a interface do Jupyter (opcional)
from pylab import rcParams
rcParams['figure.figsize'] = 10, 5
```

Definindo variáveis¶

O Python é uma linguagem orientada a objetos com tipagem fraca análoga com o MATLAB / Octave, e portanto variáveis podem ser atribuídas livremente sem especificar o tipo

---

```
a = 10
b = 20
c = np.arange(5) # sequencia com 5 elementos (de 0 até 4)
```

Exibindo variáveis¶

Os tipos e o conteúdo das variáveis no console do Python podem ser exibidas utilizando-se do comando whos

É possível inspecionar variáveis individuais digitando o nome da variável em um bloco e o executando

---

```
whos
```

```
Variable    Type       Data/Info
--------------------------------
N           int        6
a           int        10
b           int        20
c           ndarray    5: 5 elems, type `int64`, 40 bytes
dimensoes   list       n=2
np          module     <module 'numpy' from '/us<...>kages/numpy/__init__.py'>
plt         module     <module 'matplotlib.pyplo<...>es/matplotlib/pyplot.py'>
rcParams    RcParams   _internal.classic_mode: F<...>: 0.6\nytick.right: False
st          module     <module 'scipy.stats' fro<...>scipy/stats/__init__.py'>
```

```
c
```
```
array([0, 1, 2, 3, 4])
```

Ajuda¶

No IPython, que é a engine por trás da execução dos scripts no Jupyter e no Spyder, é possível utilizar o ponto de interrogação em funções ou variáveis para obter uma ajuda detalhando as mesmas.
Ao usar dois pontos de interrogação, é exibida uma ajuda mais extensa.

```
print?
```
```
print??
```
```
c?
```

Matrizes¶

Existe uma diferença fundamental entre o Numpy e MATLAB/Octave: neste último assume-se de que todos os números são matrizes, enquanto que no Numpy existe a diferenciação entre sequências e matrizes multidimensionais.

```
# Sequencia unidimensional de 3 zeros

dimensoes = 3

c = np.zeros(dimensoes) # Criar uma estrutura especificado pela váriavel dimensões

print(c) # Mostrar a váriavel c tal como é
print()
print(c.T) # A transposta de uma sequência em 1D é outra sequência em 1D
```
```
[ 0  0  0]

[ 0  0  0]
```

```
# matriz 3x1 de zeros

dimensoes = [3, 1]

c = np.zeros(dimensoes)

print(c)
print()
print(c.T)
```
```
[[ 0]
 [ 0]
 [ 0]]

[[ 0  0  0]]
```

```
# Matriz 3x3 de zeros

dimensoes = [3, 3]
c = np.zeros(dimensoes)
print(c)
```

```
[[ 0  0  0]
 [ 0  0  0]
 [ 0  0  0]]
```

```
# Matriz 3x5 de zeros

dimensoes = [3, 5]
c = np.zeros((3, 5))
print(c)
```

```
[[ 0  0  0  0  0]
 [ 0  0  0  0  0]
 [ 0  0  0  0  0]]
```

Colocando valores em matrizes¶

Aqui colocaremos valores em matrizes do Numpy

```
# Jeito 1 - forma direta
x = [290, 310, 290, 320, 280, 330] # Fazer uma lista com as médias
x = np.array([x]) # Transformar a lista em um vetor-linha do Numpy, atenção no colchete

print("x: \n %s " % x)
print("x transposta: \n %s" % x.T)

print("x 2x3: \n %s " % x.reshape(2, 3)) # Exemplo de transformação
```

```
x:
 [[290 310 290 320 280 330]]

x transposta:
 [[290]
 [310]
 [290]
 [320]
 [280]
 [330]]

x 2x3:
 [[290 310 290]
 [320 280 330]]
```

```
# Jeito 2 - via transformação
x = [290, 310, 290, 320, 280, 330]
x = np.array(x) # Transformar a lista em uma sequência do Numpy
N = len(x) # Quantidade de elementos em x

x = x.reshape(N, 1) # Transformar a sequência em uma matriz N x 1 (vetor-coluna)
print("x: \n %s " % x)

x = x.reshape(1, N) # Transformar a sequência em uma matriz 1 x N (vetor-linha)
print("x transposta: \n %s" % x)

print("x 2x3: \n %s " % x.reshape(2, 3))
```

```
x:
 [[290]
 [310]
 [290]
 [320]
 [280]
 [330]]

x transposta:
 [[290 310 290 320 280 330]]

x 2x3:
 [[290 310 290]
 [320 280 330]]
```

```
print(np.mean(x)) # Média de x - todos os elementos
print(np.std(x)) # Desvio padrão de x - todos os elementos
```

```
303.333333333
17.9505493571
```

Formatação¶

Ao fazer print de matrizes do Numpy, a formatação pode ser feita através do np.set_printoptions. Maiores detalhes podem ser encontrados em https://docs.scipy.org/doc/numpy/reference/generated/numpy.set_printoptions.html

Numeros aleatórios¶

A geração de números aleatórios é feita através do submodulo numpy.random ou então do scipy.stats.

```
np.random.randn(5) # Gerar 5 números aleatórios gaussianos de média 0 e desvpad 1
```

```
array([-1, -2, -0, -0, -1])
```

```
np.random.rand(5) # Gerar 5 números aleatórios uniformes entre 0 e 1
```

```
array([ 1,  0,  0,  1,  1])
```

```
# Gerar 10 números gaussianos de média 300 e desv. pad 5
x = 300 + np.random.randn(10) * 5
print("Média: %.2f, Desvpad: %.2f" % (np.mean(x), np.std(x)))
```

```
Média: 298.66, Desvpad: 4.37
```

Limpar variáveis¶

As variáveis importadas na memória podem ser resetadas pelo comando %reset, porém atenção: você terá de reimportar as bibliotecas.

```
%reset
```

```
Once deleted, variables cannot be recovered. Proceed (y/[n])? n
Nothing done.
```

Loops¶

Loops no Python podem ser feitos pelo uso do while e do for. A indentação define o bloco do loop

```
# Gerar uma sequência de N números gaussianos com média x0 e desv. pad sigma

x0 = 300
sigma = 50
N = 6
x = np.zeros(N) # Gerar uma sequência de N zeros

for i in range(N):
    x[i] = x0 + sigma * np.random.randn()

print("%s" % np.round(x)) # Printar a sequência x arredondada
```

```
[ 194  290  395  259  332  385]
```

```
# Transformar a sequência acima em vetor coluna

x_col = x.reshape(N, 1)
print(x_col)
print(x_col.shape) # Dimensões do vetor x_col
```

```
[[ 354]
 [ 342]
 [ 290]
 [ 283]
 [ 293]
 [ 234]]
(6, 1)
```

```
# Transformar em vetor linha

x_row = x.reshape(1, N)
print(x_row)
print(x_row.shape) # Dimensões do vetor x_row
```

```
[[ 354  342  290  283  293  234]]
(1, 6)
```

```
# Gerar uma sequência de nREP desvio-padrões de sequências de N números gaussianos de média x0 e desv. pad sigma

x0 = 300
sigma = 50
N = 32
nREP = int(1e3)
s = np.zeros(nREP)

for qREP in range(nREP):
    x = np.zeros(N)
    for i in range(N):
        x[i] = x0 + sigma * np.random.randn()
    s[qREP] = np.std(x)
```

Visualização¶

Mostrar os desvio-padrões entre a segunda e a sexta iteração do loop em nREP:

```
print(s[1:6])
```

```
[ 45  48  55  45  49]
```

Contar a quantidade de valores na sequência s que possuem valor $s \leq 35.6$

```
np.sum(s <= 35.6)
```

```
12
```

```
# Fazer um gráfico dos desvio-padrões (eixo-y) x posição na sequência (eixo-x)
plt.plot(s, '*')
plt.show()
```

```
# Fazer um histograma dos desvio-padrões
plt.hist(s)
plt.show()
```