Lab Introdução à Ciência de Dados e o Aprendizado de Máquina

O Python é uma linguagem bastante poderosa e com muitos recursos. Nessa introdução de Python para Ciência de Dados vamos rever alguns os recursos mínimos necessários para fazermos Análises de Dados e construírmos modelos de Aprendizado de Máquina ao longo do curso.

Imports básicos *

```
import numpy as np
import pandas as pd
import os
os.getcwd()
```

Variáveis e atribuições

```
inteiro = 123
real = np.pi
texto1 = '123'
texto2 = 'Led Zeppelin'
print(type(inteiro), type(real), type(texto1), type(texto2))
print(inteiro, real, texto1, texto2)
```

```
<class 'int'> <class 'float'> <class 'str'> <class 'str'>
123 3.141592653589793 123 Led Zeppelin
```

Variáveis no `ipynb` *

As células de código em um Python notebook não são programas isolados, mas um só programa. O conteúdo das variáveis do notebook é global. Assim as variáveis e imports de uma

célula são os mesmos ao longo de todo notebook.

```
x = 0; x
```

```
0
```

```
x = x + 1; x # rexecute várias vezes
```

```
3
```

```
del x
```

```
x # erro! volte execute a célula 'x = 0; x' e em seguida reexecute aqui
```

```
0
```

Caracteres e Valores Numéricos

Funciona

[ ] ↳ *1 cell hidden*

Não Funciona

```
texto1 = texto1 + 4
```

Integer, Float-point, Boolean

[ ] ↳ *1 cell hidden*

Math functions

```
import math
# also in math module: pi, exp, log, sin, cos, tan, ceil, floor, and more

print(math.pi)            # pi
print(math.exp(1))        # e

print(5 % 2)
print(np.mod(55,2))
```

```
3.141592653589793
2.718281828459045
1
1
```

Strings *

Strings são estruturas **imutáveis**. Além disso note a indexação dos *slices* de strings:

`str[ inicio : tamanho ]` ou `str[ inicio : fim+1 ]`

```
print(texto2[0:12])
print('0123456789012'[0:12])
```

```
Led Zeppelin
012345678901
```

String Operations *

Aproveite para entender como buscar ajudar aos comandos na internet.

Experimente:

> **Google**: 'How to change a string to lower case in Python'

> **Google**: 'How to find a string in a text in Python'

Ou experimente:

https://www.w3schools.com/python/default.asp

https://docs.python.org/3/ > Library Reference

```
str1 ='A arte existe porque a vida não basta. - Ferreira Gullar'

print(str1.find('arte'))
print(str1.lower())
str2 = str1.replace('arte','life')
print(str2)

print(str(4.2e-4))

# avançado, você pode saltar isso se quiser
str1 = 'love'
str2 = 'hate'
print(str1, str2)
print(str1 == str2)
print(str1 is str2)
str1 = str2
print(str1, str2)
print(str1 == str2)
print(str1 is str2) # e por isso imutável
```

```
print(str1 is str2) # é por isso imutável
```

```
2
a arte existe porque a vida não basta. – ferreira gullar
A life existe porque a vida não basta. – Ferreira Gullar
0.00042
love hate
False
False
hate hate
True
True
```

Não funciona

```
texto2[0:2] = 'x'
```

Print & Input

```
# Print & Input...
text = input('Entre com um texto: ')
print(type(text))
numero = float(input('Entre com um número: '))

print('Seu texto: ', text, '\n', 'Seu número: ', numero)
```

```
Entre com um texto: 12
<class 'str'>
Entre com um número: 12
Seu texto:  12
 Seu número:  12.0
```

Controlando o Fluxo do Programa

If-Then-Else *

Note que em Python as instruções **aninhadas** são **identadas** (*tab*), o que corresponde ao `{}` de linguagens como Java e C. Assim os comandos *aninhados* à condição ou laço do programa devem estar à direita do comando inicial da condição ou laço.

```
Condição 1...
 Comando A...
 Comando B...
 Condição 2...
   Comando C...
```

```
  Comando D...


 Comando E
```

Comando D executa dentro da condição 1, mas fora da condição 2.

```
a = 1964
b = 1984
if b > a:
  print("b is greater than a")
else:
  print("a is greater than b")

# # avançado, você pode saltar isso se quiser
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

```
b is greater than a
```

For & While *

Vamos dar preferência ao uso do `for`.

```
for i in range(5):
  print(i)
  if i == 3:
    break

for i in [12,18,84]:
    print(i)
```

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

Funções

Nossos programas não precisarão necessariamente de *criar* funções, mas é comum a chamada de funções e o seu uso pode permitir deixar o código mais modular e enxuto.

```
def mymax(a,b):
  if a > b:
    return a
  else:
    return b

mymax(1984,1964)
```

```
1984
```

Estruturas de Dados *

Estruturas de dados como Lists, Sets, Tuples, Sequencies e Dictionary desempenham um papel fundamental tanto para programação em geral como para aplicações mais avançadas como a análise de dados.

Coleções

Existem quatro tipos de dados de coleções em Python:

- Lista (**list**) é uma coleção que é ordenada e mutável. Permite membros duplicados.
- Tupla (**tuple**) é uma coleção que é ordenada e imutável. Permite membros duplicados.
- Conjunto (**set**) é uma coleção não ordenada e não indexada. Nenhum membro duplicado.
- Dicionário (**dictionary**) é uma coleção desordenada, mutável e indexada. Nenhum membro duplicado.

Vamos nos deter aqui apenas nas estruturas de listas e tuplas que terão mais utilidade para os nossos propósitos.

List *

Lista (**list**) é uma coleção que é **ordenada** e **mutável**. Permite membros **duplicados**. Associada a lista você encontra uma série métodos que implementam funções úteis para esses objetos.

```
# avançado, você pode saltar isso se quiser
mylist = ['A', 'B', 'C', 'A', 'B']
help(mylist)
# dir(mylist)
```

```
mylist = ['A', 'B', 'C', 'A', 'B']

len(mylist)

mylist.count('B')

mylist.index('B')
mylist.index('B', 2)

mylist.append('D')

mylist.sort()
mylist.insert(1,'X')

del mylist[0:1] # o mesmo que del fruits[0]
mylist.remove('B') # somente o primeiro elemento

mylist[1] = 'ZZ'

mylist = ['A', 'B', 'C', 'A', 'B']
print('A' in mylist)
if 'A' in mylist:
  print('Sim, A está na lista')

# avançado, você pode saltar isso se quiser
mylist = [[1,2,3], [11,22,33], [9,9,9]]
print(mylist[1])  # [11,22,33]
print(mylist[1][0])  # 11

# avançado, você pode saltar isso se quiser
mylist = ['A', 'B', 'C', 'A', 'B']
mylist1 = mylist            # NÃO USE
mylist2 = mylist.copy()     # USE
mylist.remove('B')
print(mylist1) # ['A', 'C', 'A', 'B']
print(mylist2) # ['A', 'B', 'C', 'A', 'B']
```

```
[11, 22, 33]
11
True
Sim, A está na lista
```

```
['A', 'C', 'A', 'B']
```

```
# avançado, você pode saltar isso se quiser
from IPython.display import IFrame
url = "https://pythontutor.com/iframe-embed.html#code=mylist%20%3D%20%5B'A',%20'B',%20'C',
IFrame(url, width='1000', height=350)
```

Percorrendo uma lista *

Existem basicamente duas forma de percorrer uma lista.

- **Por índice**, empregue quando precisar do índice (posição) ou alterar os elementos
- **Por iterador**, empregue apenas para recuperar (ler) os elementos

```
mylist = ['A', 'B', 'C', 'A', 'B']

for i in range(len(mylist)):
    print(i, mylist[i])

for item in mylist:
    print(item)
```

```
0 A
1 B
2 C
3 A
4 B
A
B
C
```

```
A
B
```

```
# avançado, você pode saltar isso se quiser
from IPython.display import IFrame
url = "https://pythontutor.com/iframe-embed.html#code=mylist%20%3D%20%5B'A',%20'B',%20'C',
IFrame(url, width='1000', height=350)
```

Populando uma lista *

```
mylist = []

for elem in ['Beatriz', 'Aline', 'Adriana', 'Daniel']:
    mylist.append(elem)

print(mylist)
```

```
['Beatriz', 'Aline', 'Adriana', 'Daniel']
```

```
# avançado, você pode saltar isso se quiser
from IPython.display import IFrame
url = "https://pythontutor.com/iframe-embed.html#code=mylist%20%3D%20%5B%5D%0A%20%0Afor%20
IFrame(url, width='1000', height=350)
```

Dictionary *

Dicionário (**dictionary**) é uma coleção desordenada, mutável e indexada. Nenhum membro duplicado. Você já teve algum contato com uma estrutura de dados `JSON` essa uma estrutura bastante semelhante em Python.

```
mydict = {'Beatriz':1978, 'Aline':1987, 'Adriana':1984, 'Daniel':1996}

print(mydict.keys())

print(mydict['Adriana'])
```

```
dict_keys(['Beatriz', 'Aline', 'Adriana', 'Daniel'])
1984
```

Percorrendo um Dicionário *

```
for item in mydict:
    print(item, mydict[item])
```

```
Beatriz 1978
Aline 1987
Adriana 1984
Daniel 1996
```

Populando um Dicionário *

```
lista_nomes = [ 'Carolina', 'Henrique' ]
```

```
lista_niver = [ 2001, 2005 ]

mydict = {}

for i in range(len(lista_nomes)):
  if lista_nomes[i] not in mydict.keys(): # Se não está no dicionário acrescenta
    mydict[lista_nomes[i]] = lista_niver[i]

if 'Kate' not in mydict.keys():           # Se não está no dicionário acrescenta
  mydict['kate'] = 2002

print(mydict)
```

```
{'Carolina': 2001, 'Henrique': 2005, 'kate': 2002}
```

File Handling

Operações de arquivos como essa serão pouco comuns pois empregaremos na maior parte do tempo dados no formato de `dataframes` como visto na aula de Teoria. Assim, se quiser, pode deixar esse tópico para mais tarde.

```
note = ['This is a note']

f = open("note.txt", "w")
for line in note:
    f.write(line)
f.close()

f = open("note.txt", "r")
for line in f:
    print(line)
```

```
This is a note
```

Acesso dados Web

Isso pode ser útil, mas como no item anterior empregaremos na maior parte do tempo dados no formato de `dataframes` como visto na aula de Teoria. Assim, se quiser, pode deixar esse tópico para mais tarde.

```
import urllib.request

data = urllib.request.urlopen('http://meusite.mackenzie.br/rogerio/TIC/PlayBallcsv.csv')

for line in data:
    print(line.decode('utf-8'))
```

```
Day,Outlook,Temperature,Humidity,Wind,Play ball
```

```
D1,Sunny,Hot,High,Weak,No

D2,Sunny,Hot,High,Strong,No

D3,Overcast,Hot,High,Weak,Yes

D4,Rain,Mild,High,Weak,Yes

D5,Rain,Cool,Normal,Weak,Yes

D6,Rain,Cool,Normal,Strong,No

D7,Overcast,Cool,Normal,Strong,Yes

D8,Sunny,Mild,High,Weak,No

D9,Sunny,Cool,Normal,Weak,Yes

D10,Rain,Mild,Normal,Weak,Yes

D11,Sunny,Mild,Normal,Strong,Yes

D12,Overcast,Mild,High,Strong,Yes

D13,Overcast,Hot,Normal,Weak,Yes

D14,Rain,Mild,High,Strong,No
```

Numpy *

Talvez você não tenha notado. Mas o Python não vetores ou matrizes. A biblioteca `numpy` implementa então essas funções.

Para nossos propósitos será suficiente o uso de vetores e algumas poucas funções do `numpy`, mas aplicações mais sofisticadas podem requerem um domínio maior dessa biblioteca numérica.

Os vetores do `numpy` operam como listas indexadas e você pode empregar os mesmos códigos de listas para **percorrer** e **popular** vetores `numpy`.

```
import numpy as np

x = np.array([1,2,3])
x_list = x.tolist()

x = x * 2 + 1
# x_list = x_list * 2 + 1 # TypeError: can only concatenate list (not "int") to list

np.random.seed(1984) # semente de geração aleatória
x = np.array(np.random.randint(low=1, high=6, size=10))
print(x)

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

```
np.mean(x)
np.sum(x)
np.power(x,2)

x = np.insert(x,len(x),4) # Note: NÂO FAZ INPLACE
x = np.delete(x,len(x)-1) # Note: NÂO FAZ INPLACE
x = np.append(x,[4,5,6])
x[0] = 9
x
```

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

Set Up

Em alguns exercícios adiante você irá fazer programas para buscar valores em uma lista `x` ou `x_sorted`. O código abaixo produz essas lista, com 1000 valores aleatórios, e que será empregada nos exercícios seguintes.

```
import numpy as np
import random
random.seed(1984)
x = random.sample(range(1, 10000), 1000)
x_sorted = np.sort(x)
print(x[0:10])
print(x_sorted[0:10])
```

```
[5957, 8098, 5565, 5015, 3547, 6321, 1159, 3999, 4522, 824]
[ 12  24  27  48  55  58  81  99 103 109]
```

Exercício 1 (resolvido)

Dado o valor de entrada `a`. Encontre a posição, e o primeiro maior valor que `a`, na lista de valores ordenados `x_sorted`.

```
a = int(input('Entre com um valor entre 1 e 9999: '))

for i in range(len(x)):
  if x_sorted[i] >= a:
    print('O ', i, '-ésimo valor, igual a ', x_sorted[i], ' , é o primeiro valor maior que
    break
```

```
Entre com um valor entre 1 e 9999: 2020
O  204 -ésimo valor, igual a  2045  , é o primeiro valor maior que  2020
```

Exercício 2

Dado o valor de entrada `a`. Encontre a posição, e o primeiro maior valor que `a`, que seja **PAR**, na lista de valores ordenados `x_sorted`.

DICA: Para as instruções de `if` empregue as condições entre parênteses e os operadores & ou |

```
a = int(input('Entre com um valor entre 1 e 9999: '))

for i in range(len(x)):
  if (x_sorted[i] >= a) & (x_sorted[i] % 2 == 0):
    print('O ', i, '-ésimo valor, igual a ', x_sorted[i], ' , é o primeiro valor maior que
    break
```

```
Entre com um valor entre 1 e 9999: 2020
O  205 -ésimo valor, igual a  2052  , é o primeiro valor maior que  2020
```

Exercício 3

Percorra a lista `x` (note, é `x`, a lista de elementos **não ordenada**) e encontre o maior valor da lista.

```
maior = 0

for i in range(len(x)):
  if x[i] > maior:
    maior = x[i]

print('Maior valor da lista: ', maior)
```

```
Maior valor da lista:  9996
```

```
# Claro que você poderia empregar
max(x)

# Mas o exercício pede para percorrer a lista
```

```
9996
```

Exercício 4

A partir da lista inicial `x` crie uma lista com todos os valores pares e então empregue `sum(sualista)` para exibir o resultado da soma desses valores.

```
pares = []
```

```
for i in range(len(x)):
  if x[i] % 2 == 0:
    pares.append(x[i])

print(pares[0:10])
print('Soma dos valores pares: ', sum(pares))
```

```
[8098, 4522, 824, 1798, 8622, 6538, 5850, 4128, 4162, 7572]
Soma dos valores pares:  2496034
```

Exercício 5 (resolvido)

Crie um programa para criar uma lista de nomes a partir da entrada.

```
T = True
nomes = []

while T:
  nome = input('Entre com um nome ou \'END\' para terminar: ')
  if nome != 'END':
    nomes.append(nome)
  else:
    break

print(nomes)
```

```
Entre com um nome ou 'END' para terminar: Kate
Entre com um nome ou 'END' para terminar: Kim
Entre com um nome ou 'END' para terminar: Anna
Entre com um nome ou 'END' para terminar: END
['Kate', 'Kim', 'Anna']
```

Exercício 6

Crie um programa que dado um nome, busca a quantidade de vezes que esse nome aparece em uma lista. Empregue a lista de nomes do exercício anterior ou você pode criar uma nova.

```
nomes = ['Adriana','Daniel','Carolina','Adriana','Carol']
nome = input('Entre com um nome: ')

print(nomes.count(nome))
```

```
Entre com um nome: Adriana
2
```

Exercício 7, Dicionário de Têrmos

Crie um dicionário de têrmos (palavras) para a biografia da atriz **Kate Beckinsale**.

1. A primeira célula obtém o texto da biografia da atriz na variável `texto`.
2. Na segunda célula **adapte o código** para criar o dicionário de têrmos com a seguinte estrutura:

```
mydict = { 'word1': qty, 'word2': qty, 'word3': qty, ...}
```

3. A terceira célula apresenta um gráfico de distribuição dos têrmos do seu dicionário

```
# Obtém o texto na lista 'texto'
# Não altere este código

import urllib.request  # the lib that handles the url stuff

texto = []

# Moby-Dick, Hermann Melville
# https://www.gutenberg.org/files/15/15-0.txt
# Ulysses, James Joyce
# https://www.gutenberg.org/files/4300/4300-0.txt

for line in urllib.request.urlopen('http://meusite.mackenzie.br/rogerio/kate_beckinsale.tx
#   print(line.decode('utf-8')) #utf-8 or iso8859-1 or whatever the page encoding scheme i
    texto.append(line.decode('utf-8'))

# f = open('/kate_beckinsale.txt','r')
# for line in f:
#    texto.append(line)

for i in range(len(texto)):
    texto[i] = texto[i].lower() # para unicidade
    texto[i] = texto[i].replace('\n','')
    texto[i] = texto[i].replace('.','')
    texto[i] = texto[i].replace(',','')
    texto[i] = texto[i].replace('(','')
```

```
    texto[i] = texto[i].replace('(','')
    texto[i] = texto[i].replace(')','')
    texto[i] = texto[i].replace('?','')
    texto[i] = texto[i].replace('\'','') # elimina ' e "

# print(texto)



mydict = {}                             # crie um dicionário vazio

for line in texto:

  line = line.lower()                   # converte para lower
  words = line.split()                  # separa cada palavra
#  print(words)

  for word in words:                      # para cada palavra em words
    if word not in mydict.keys():         # se palavra não está no dicionário
      mydict[word] = 1                    # acrescenta a word com o valor 1
    else:                                 # se a entrada já existe
      mydict[word] =  mydict[word] + 1    # apenas soma 1 ao valor já existente

print(mydict)
```

```
{'kathrin': 2, 'romary': 2, 'beckinsale': 70, 'chiswick': 2, '26': 2, 'de': 206, 'jul
```

```
print(mydict['atriz'])
```

```
5
```

```
print(mydict['britânica'])
```

```
2
```

```
# plot da frequência de termos
# não é necessário alterar esse código
import pandas as pd
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt

df = pd.DataFrame(mydict.items(),  columns=['word', 'count']).sort_values('count',ascendin
df = df[df['count'] > 4]
# df = df.iloc[ np.int(len(df)/2) - 10 : np.int(len(df)/2) + 10 ] # For books

plt.figure(figsize=(24,10))
mpl.style.use(['seaborn'])
sns.barplot(df.word,df['count'])
plt.xticks(rotation=90)
```

Exercício 8. Altere a entrada de dados do exercício anterior para empregar o texto Ulisses, de James Joyce. Qual dentre esses nomes é o personagem principal da obra: *bloom*, *ulysses* ou *stephen*?

DICA: Empregue a frequência dos têrmos do dicionário para identificar o personagem mais importante.

```
# Obtém o texto na lista 'texto'
# Não altere este código

import urllib.request  # the lib that handles the url stuff

texto = []

# Moby-Dick, Hermann Melville
# https://www.gutenberg.org/files/15/15-0.txt
```

```
# https://www.gutenberg.org/files/15/15-0.txt
# Ulysses, James Joyce
# https://www.gutenberg.org/files/4300/4300-0.txt

for line in urllib.request.urlopen('https://www.gutenberg.org/files/4300/4300-0.txt'):
#   print(line.decode('utf-8')) #utf-8 or iso8859-1 or whatever the page encoding scheme i
    texto.append(line.decode('utf-8'))

# f = open('/kate_beckinsale.txt','r')
# for line in f:
#    texto.append(line)

for i in range(len(texto)):
    texto[i] = texto[i].lower() # para unicidade
    texto[i] = texto[i].replace('\n','')
    texto[i] = texto[i].replace('.','')
    texto[i] = texto[i].replace(',','')
    texto[i] = texto[i].replace('(','')
    texto[i] = texto[i].replace(')','')
    texto[i] = texto[i].replace('?','')
    texto[i] = texto[i].replace('\'','') # elimina ' e "

# print(texto)


print(mydict['bloom'], mydict['ulysses'], mydict['stephen'])
```

```
 598 10 386
```

Luhn e Zipf Law e Ciência de Dados

A frequência de termos (quantidade que cada palavra aparece em um texto) tem grande importância em uma série de problemas envolvendo busca, classificação e sumarização automática de documentos.

As palavras que um texto contêm e a frequencia delas pode muito bem dar uma pista do conteúdo do texto

> Hans Peter Luhn (1957): **The weight of a term that occurs in a document is simply proportional to the term frequency**.

Entretanto parecem haver termos muito frequentes, como artigos, preposições etc. que parecem contribuir muito pouco para explicar o conteúdo de um texto.

> George Kingsley Zipf: **Zipf's law states that given a large sample of words used, the frequency of any word is inversely proportional to its rank in the frequency table. So word number n has a frequency proportional to 1/n.**

Isso ocorre em todo o texto de produção humana e, importante, em *qualquer língua*. Esse princípio é empregado desde para decifrar códigos e texto em linguas antigas, como em

modernos sistemas de mineração de texto (*search engines*, bibliotecas digitais, classificação automática de conteúdo etc.).

```
from IPython.display import IFrame
IFrame('https://demonstrations.wolfram.com/ZipfsLawAppliedToWordAndLetterFrequencies/', wi
```