Python para Ciência de Dados

NumPy, Matplotlib, Pandas Introdutório

Luiz Alberto

Ciência da Computação

May 16, 2019

NumPy

Numeric Python:

Biblioteca para computação científica. Implementa arrays muldimensionais e permite a fácil execução de operações matemáticas (`www.numpy.org`).

- Utilizada em cálculos de alta performance de vetores e matrizes
- Fornece versões pré-compiladas de funções numéricas
- Alternativa à Lista em Python: NumPy Array
- Cálculos sobre matrizes inteiras (broacasting)
- Escrita em C e Fortran

NumPy

```
1  import numpy as np
2  alturas = [ 1.73, 1.68, 1.71, 1.89, 1.79 ]
3  pesos = [ 65.4, 59.2, 63.6, 88.4, 68.7 ]
4
5  np_alturas = np.array(alturas)
6  np_pesos = np.array(pesos)
7
8  imcs = np_pesos / np_alturas ** 2
9  print(imcs)
10 # output:
11 # [21.85171573 20.97505669 21.75028214 24.7473475 21.44127836]
```

Comparação com listas

```
1 import numpy as np
2 alturas = [ 1.73, 1.68, 1.71, 1.89, 1.79 ]
3 pesos = [ 65.4, 59.2, 63.6, 88.4, 68.7 ]
4
5 print(alturas/pesos)
6 # output:
7 # TypeError: unsupported operand type(s) for /: 'list' and '
    list'
```

Subsetting

```
1 import numpy as np
2 imcs
3 # array([21.85171573, 20.97505669, 21.75028214, 24.7473475,
      21.44127836])
4 imcs[1]
5 # 21.85171573
6 imcs > 23
7 # array([False, False, False, True, False])
8 imcs[imcs > 23]
9 # array([24.7473475])
```

Arrays multidimensionais

```
1
2 import numpy as np
3 medidas = np.array([[1.73, 1.68, 1.71, 1.89, 1.79],
4                    [65.4, 59.2, 63.6, 88.4, 68.7]])
5
6 print(medidas.shape) # retorna as dimensoes do array
7 # (2, 5)
```

Fatiamento (1)

```
1
2  import numpy as np
3  matriz = np.array([[1.73, 1.68, 1.71, 1.89, 1.79],
4                     [65.4, 59.2, 63.6, 88.4, 68.7]])
5
6  matriz[0]
7  # array([1.73, 1.68, 1.71, 1.89, 1.79])
8
9  matriz[0][2]
10 # 1.71
11
12 matriz[0, 2]
13 # 1.71
14
15 matriz[:, 1:3]
16 # array([[ 1.68,  1.71],
17 #        [59.2 , 63.6]])
18
19 matriz[1,: ]
```

Fatiamento (2)

```
20 # [65.4, 59.2, 63.6, 88.4, 68.7]
```

Estatística básica

```
1
2  import numpy as np
3  array = np.array([[0.173, 0.168, 0.171, 0.189, 0.179],
4                    [0.154, 0.259, 0.163, 0.388, 0.287]])
5
6  print(np.mean(array[1,: ]))
7  # 0.2502
8
9  print(np.median(array[1,: ]))
10 # 0.259
11
12 print(np.corrcoef(array[0, :] , array[1, :]))
13 # [[1.  0.79684]
14 #  [0.79684 1.]]
15
16 print(np.std(array[:, 0]))
17 # 0.0094
```

Geração de dados

```
1
2  import numpy as np
3  alturas = np.round(np.random.normal(1.75, 0.20, 5000), 2)
4  pesos = np.round(np.random.normal(60.32, 15, 5000), 2)
5
6  medidas = np.column_stack((alturas, pesos))
7  print(medidas)
8  # output:
9  #    [[1.73   65.9]
10 #     [2.03   69.91]
11 #     [1.48   63.18]
12 #  ...
13 #     [2.     52.73]
14 #     [1.97   27.79]
15 #     [1.89   44.29]]
```

Matplotlib

Definição:

Biblioteca python para plotagem de gráficos 2D (incluindo 3D) (`www.matplotlib.org`).

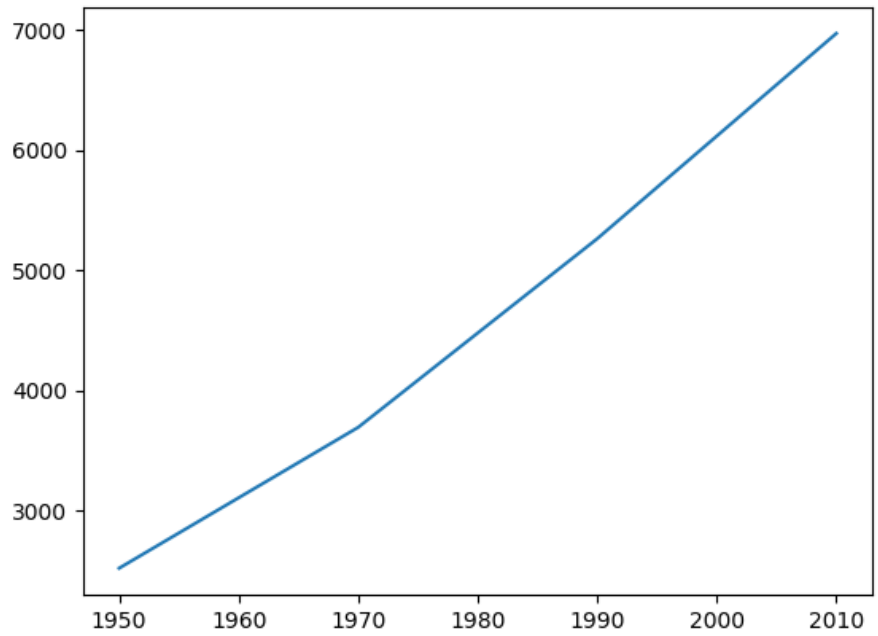
- Simplicidade de utilização
- Desenvolvimento gradual e interativo
- Grande controle sobre os elementos gráficos
- Exportação em formatos PNG, PDF, SVG e EPS

Vizualização de Dados

- Muito importante na visualização de dados
  - Explorar os dados
  - Apresentar "insights"

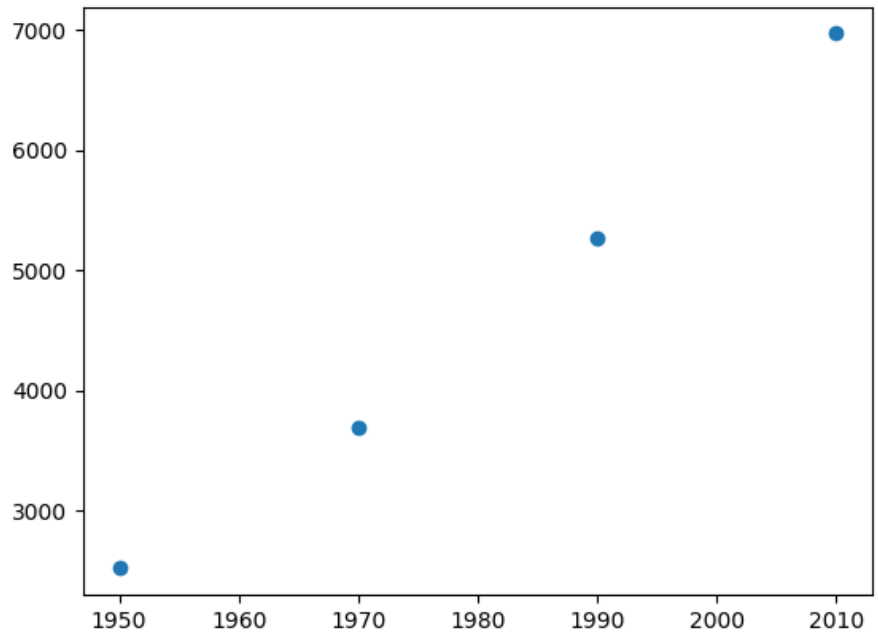
Line plot

```
1 import matplotlib.pyplot as plt
2 anos = [1950, 1970, 1990, 2010]
3 pops = [2519, 3692, 5263
4        , 6972]
5
6 plt.plot(anos, pops)
7 #          X         Y
8
9 plt.show()
```



Scatter plot

```
1 import matplotlib.pyplot as plt
2 anos = [1950, 1970, 1990, 2010]
3 pops = [2519, 3692, 5263
4       , 6972]
5
6 plt.scatter(anos, pops)
7 #           X         Y
8
9 plt.show()
```

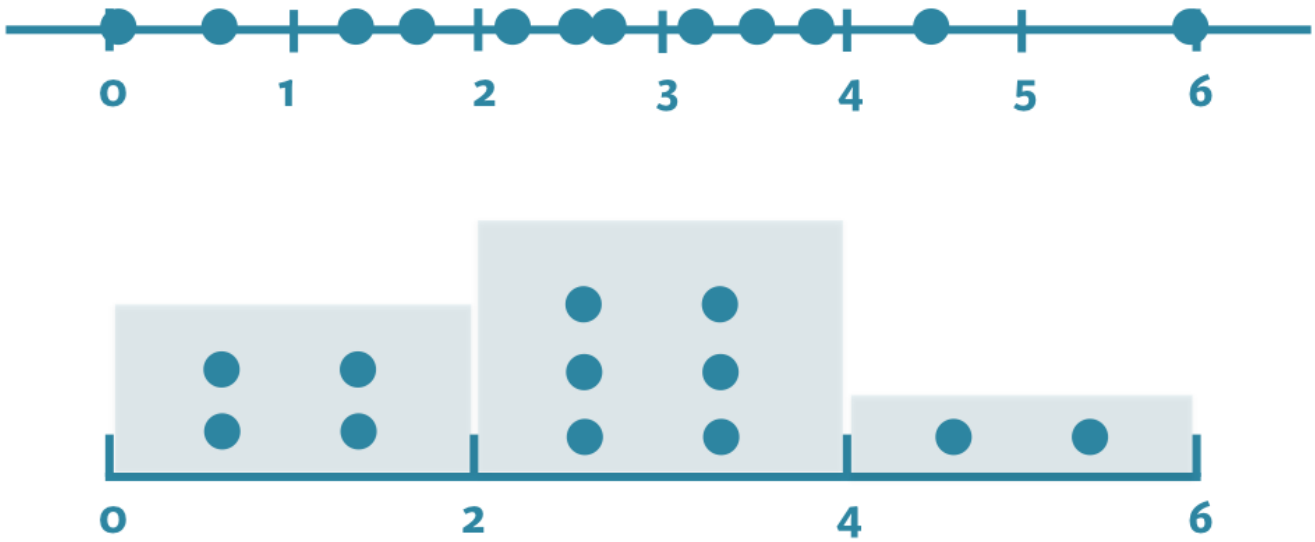


Hora de colocar a mão na massa (1)

Salvar cada um dos exercícios a seguir em um arquivo separado.

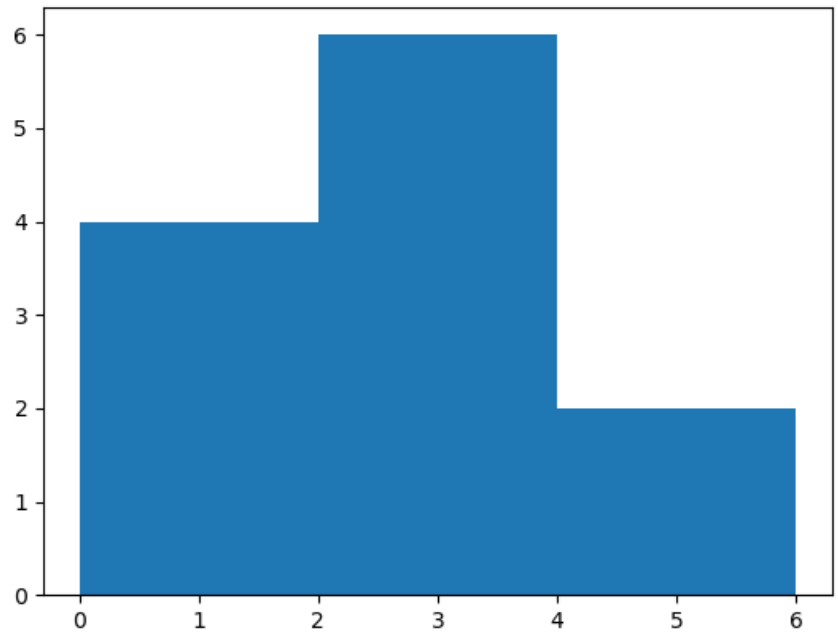
Histogram

- Utilizado para explorar dados
- Fornece uma idea da distribuição dos dados



Histogram

```
1 import matplotlib.pyplot as plt
2 valores = [
3       0,   0.6, 1.4, 1.6
4     , 2.2, 2.5, 2.6, 3.2
5     , 3.5, 3.9, 4.2, 6
6 ]
7
8 plt.hist(valores, bins=3)
9 plt.show()
```

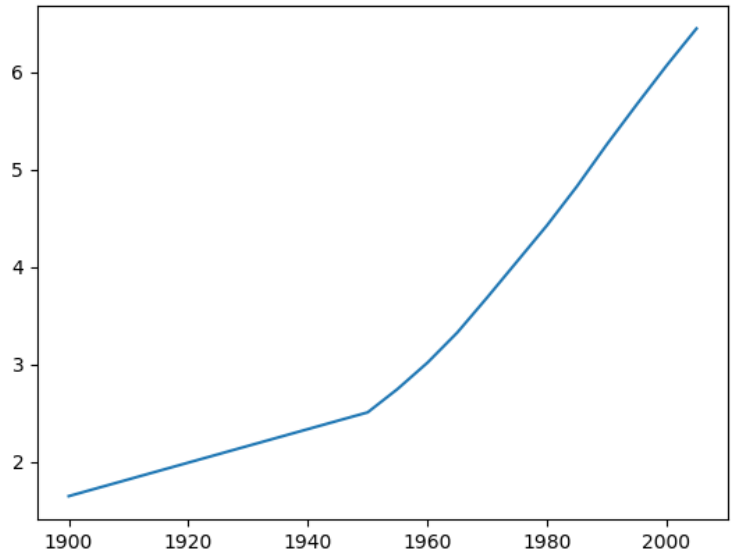


Customização

- Existem muitas opções
  - Diferentes tipos de gráficos
  - Diversas customizações
- A escolha depende
  - Dados
  - Estória a ser contada

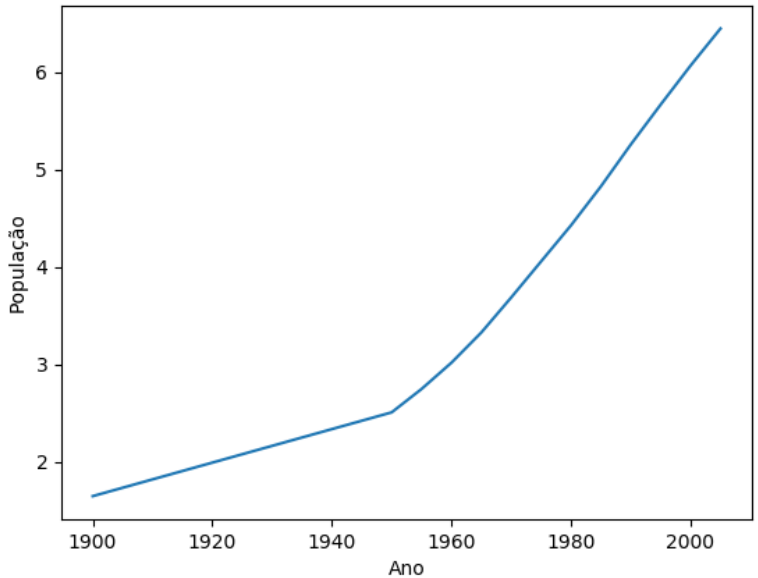
Customização

```
1  import matplotlib.pyplot as plt
2  anos = [
3    1900, 1950, 1955, 1960, 1965,
4    1970, 1975, 1980, 1985, 1990,
5    1995, 2000, 2005
6   ]
7  pops = [
8    1.65, 2.51, 2.75, 3.02, 3.33,
9    3.69, 4.06, 4.43, 4.83, 5.26,
10   5.67, 6.07, 6.45
11 ]
12
13 plt.plot(anos, pops)
14 #           X         Y
15
16 plt.show()
```



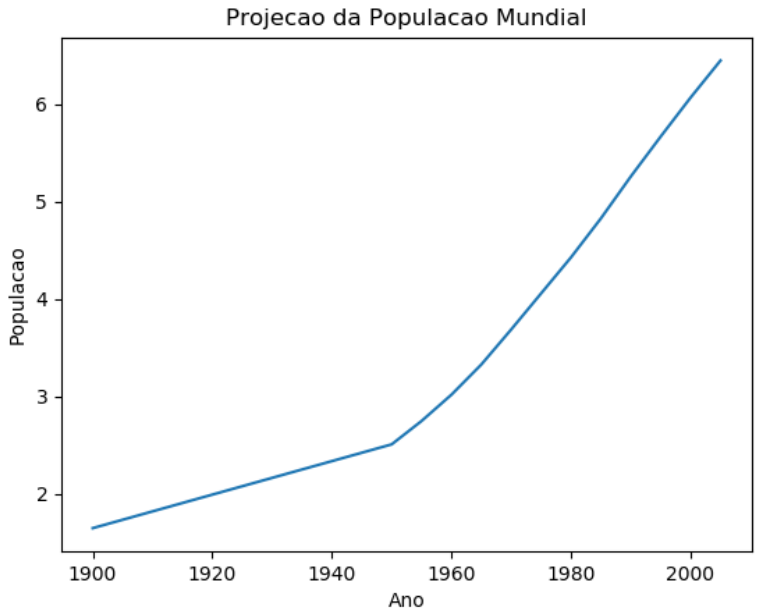
Títulos dos eixos X e Y

```
1  import matplotlib.pyplot as plt
2  anos = [
3    1900, 1950, 1955, 1960, 1965,
4    1970, 1975, 1980, 1985, 1990,
5    1995, 2000, 2005
6   ]
7  pops = [
8    1.65, 2.51, 2.75, 3.02, 3.33,
9    3.69, 4.06, 4.43, 4.83, 5.26,
10   5.67, 6.07, 6.45
11 ]
12
13 plt.plot(anos, pops)
14 plt.xlabel('Ano')
15 plt.ylabel('Populacao')
16 plt.show()
```



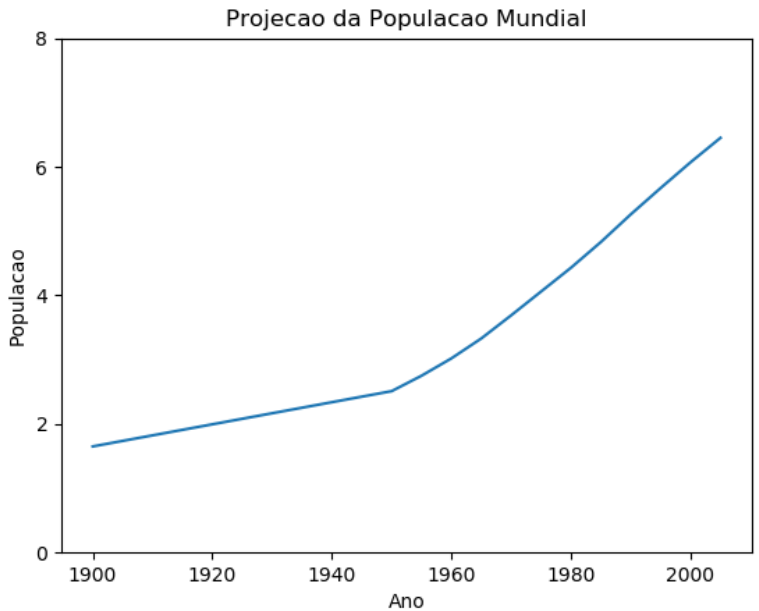
Título Principal

```
1 plt.plot(anos, pops)
2 plt.xlabel('Ano')
3 plt.ylabel('Populacao')
4 plt.title('Projecao da Populacao
      Mundial')
5 plt.show()
```



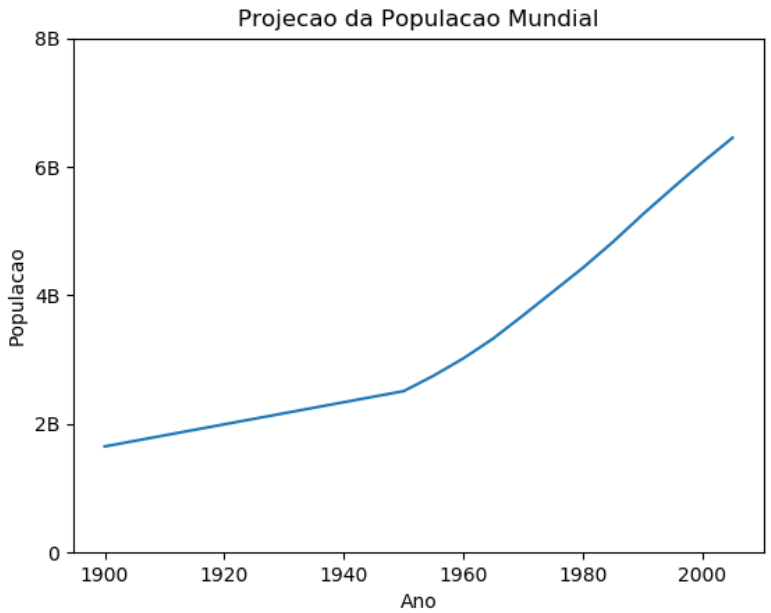
Ticks (1)

```
1 plt.plot(anos, pops)
2 plt.xlabel('Ano')
3 plt.ylabel('Populacao')
4 plt.title('Projecao da Populacao
      Mundial')
5 plt.yticks([0, 2, 4, 6, 8])
6 plt.show()
```



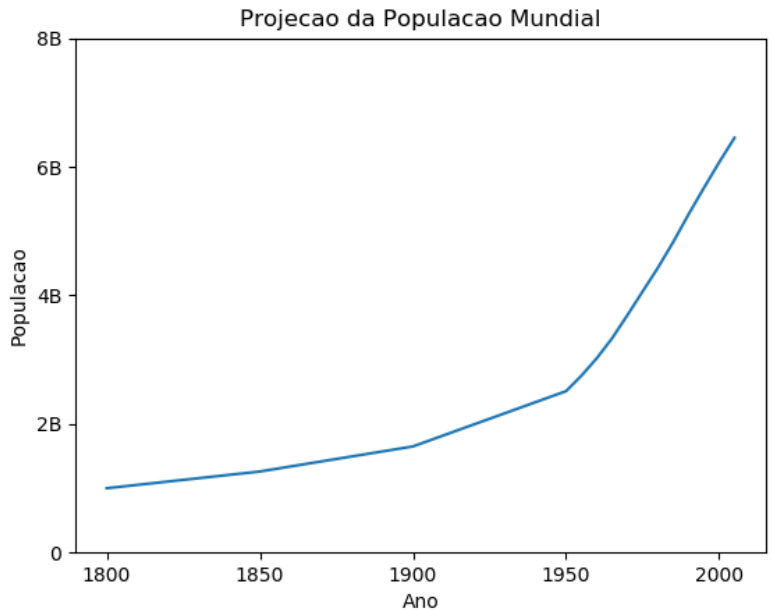
Ticks (2)

```
1 plt.plot(anos, pops)
2 plt.xlabel('Ano')
3 plt.ylabel('Populacao')
4 plt.title('Projecao da Populacao
      Mundial')
5 plt.yticks([0, 2, 4, 6, 8],
6      ['0', '2B', '4B', '6B', '8B'])
7 plt.show()
```

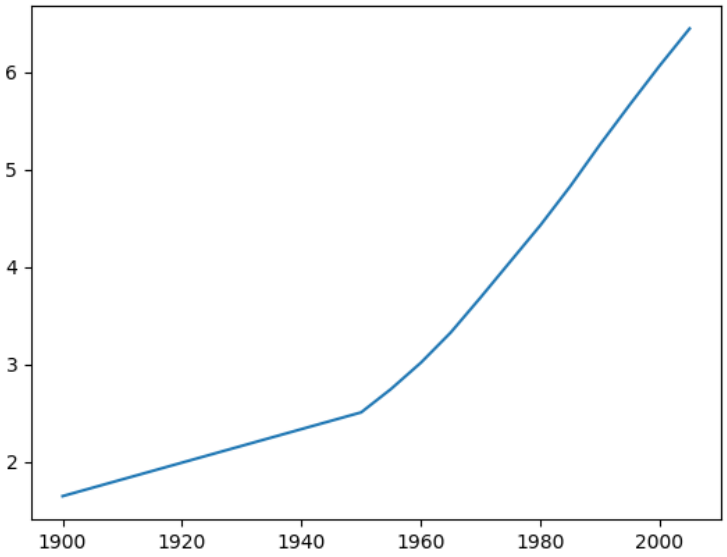


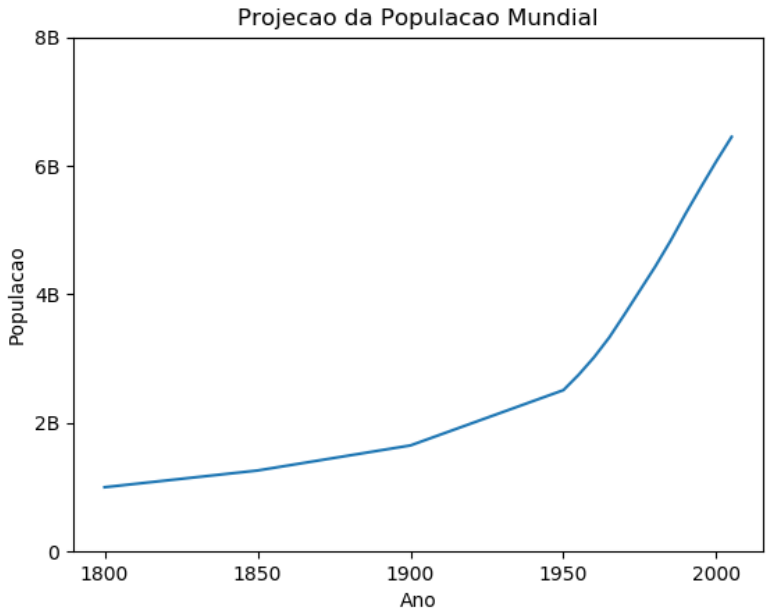
Adicionando Dados Históricos

```
1 anos = [1800, 1850] + anos
2 pops = [1.0, 1.26] + pops
3 plt.plot(anos, pops)
4 plt.xlabel('Ano')
5 plt.ylabel('Populacao')
6 plt.title('Projecao da Populacao
      Mundial')
7 plt.yticks([0, 2, 4, 6, 8],
8      ['0', '2B', '4B', '6B', '8B'])
9 plt.show()
```



Antes x Depois





Pandas

Definição:

Biblioteca para análise de dados. Fornece ferramentas para manipulação de estruturas de dados, como matrizes, vetores e dataframes(`www.pandas.pydata.org`).

- Pandas é uma biblioteca open-source com licença BSD
- Fornece ferramentas de alta performance e fáceis de utilizar para manipulação de estrutura de dados
- Escrita em Python

Pandas

Datasets:

Biblioteca para análise de dados. Fornece ferramentas para manipulação de estruturas de dados, como matrizes, vetores e dataframes(`www.pandas.pydata.org`).

- Pandas é uma biblioteca open-source com licença BSD
- Fornece ferramentas de alta performance e fáceis de utilizar para manipulação de estrutura de dados
- Escrita em Python

Pandas Series

Definição

Objeto que representa uma estrutura de dados unidimensional similar ao array mas com algumas características a mais.

- Composta por dois arrays:
- (i) dados
- (ii) índice

Series	
index	value
0	12
1	-4
2	7
3	9

Declarando uma Serie (1)

- Índice padrão

```
1 import pandas as pd
2 s = pd.Series([12, -4, 7, 9])
3 print(s)
4 # outuput:
5 # 0   12
6 # 1   -4
7 # 2   7
8 # 3   9
9 # dtype: int64
```

Declarando uma Serie (2)

- Índice personalizado

Livro: O Perfil das Empresas Brasilieiras em Gestão e Governança de Dados

```
import pandas as pd
s = pd.Series([12, -4, 7, 9], index=['a', 'b', 'c', 'd'])
print(s)
# outuput:
# a   12
# b   -4
# c   7
# d   9
# dtype: int64
```

Acessando índices e valores

```
1 import pandas as pd
2 s = pd.Series([12, -4, 7, 9], index=['a', 'b', 'c', 'd'])
3 print(s.values)
4 # outuput:
5 # [12 -4 7 9]
6 print(s.index)
7 # outuput:
8 # Index (['a', 'b', 'c', ''d], dtype='object')
```

Selecionando valores em uma Serie

```
1  import pandas as pd
2  s = pd.Series([12, -4, 7, 9], index=['a', 'b', 'c', 'd'])
3  print(s[2])
4  # outuput: 7
5  print(s['b'])
6  # outuput: -4
7  print(s[0:2])
8  # output:
9  # a   12
10 # b   -4
11 # dtype: int64
12 print(s[['a', 'b']])
13 # output:
14 # a   -4
15 # b   -7
16 # dtype: int64
```

Atribuindo valores

```
1  import pandas as pd
2  s = pd.Series([12, -4, 7, 9], index=['a', 'b', 'c', 'd'])
3  s[1] = 0
4  print(s)
5  # outuput:
6  # a   12
7  # b   0
8  # c   7
9  # d   9
10 # dtype: int64
11 s['b'] = 1
12 print(s)
13 # outuput:
14 # a   12
15 # b   1
16 # c   7
17 # d   9
18 # dtype: int64
```

Filtrando valores

```
1 import pandas as pd
2 s = pd.Series([12, -4, 7, 9], index=['a', 'b', 'c', 'd'])
3 print(s[s > 8])
4 # outuput:
5 # a   12
6 # d   9
7 # dtype: int64
```

Avaliando valores (1)

```
1  import pandas as pd
2  s = pd.Series([1, 0, 2, 1, 2, 3], index=['branco', 'branco', '
       azul'
3    , 'verde', 'verde', 'amarelo'])
4
5  # valores unicos
6  print(s.unique())
7  # outuput: [1 0 2 3]
8
9  # quantidade de ocorrencias
10 print(s.value_counts())
11 # outuput:
12 # 2    2
13 # 1    2
14 # 3    1
15 # 0    1
16
17
18
```

Avaliando valores (2)

```
19 # verificando a existencia
20 print(s.isin([0,3]))
21 # branco     False
22 # branco     True
23 # azul       False
24 # verde      False
25 # verde      False
26 # amarelo    True
```

Pandas Dataframes

Definição

Estrutura de dados tabular muito similar a uma planilha..

- Estende a **Serie** para multiplas dimensões
- Consiste de uma coleção ordenada de colunas
- Contém diversos valores com tipos diferentes

DataFrame

	columns		
index	color	object	price
0	blue	ball	1.2
1	green	pen	1.0
2	yellow	pencil	0.6
3	red	paper	0.9
4	white	mug	1.7

Dataframe baseado em dicionário (1)

```
1  import pandas as pd
2  dict = {
3  "pais":["Brasil","Russia","India","China","Africa do Sul"],
4  "capital":["Brasila","Moscou","Deli","Pequin","Pretoria"],
5  "populacao":[207.04, 143.5, 1252, 1357, 52.98]
6  }
7
8  brics = pd.DataFrame(dict)
9  print(brics)
10 #              pais     capital  populacao
11 # 0          Brasil     Brasila     207.04
12 # 1          Russia      Moscou     143.50
13 # 2           India  Nova Deli    1252.00
14 # 3           China      Pequin    1357.00
15 # 4  Africa do Sul    Pretoria      52.98
16
17
18
19
```

Dataframe baseado em dicionário (2)

```
20 """
21 Reindexando o dataframe.
22 """
23 brics.index = ["BR", "RU", "IN", "CH", "SA"]
24 print(brics)
25 #              pais     capital  populacao
26 # BR         Brasil     Brasila     207.04
27 # RU         Russia      Moscou     143.50
28 # I           India  Nova Deli    1252.00
29 # CH          China      Pequin    1357.00
30 # AS Africa do Sul    Pretoria      52.98
```

Dataframe baseado em arquivo CSV (1)

```
1  import os
2  import pandas as pd
3  brics = pd.read_csv("{}/pandas/brics.csv".format(os.getcwd()))
4  print(brics)
5  #   Unnamed: 0           pais    capital     area  populacao
6  # 0          BR         Brasil  Brasilia    8.516     207.40
7  # 1          RU         Russia    Moscou   17.100     143.50
8  # 2          IN          India      Deli    3.286    1252.00
9  # 3          CH          China    Pequin    9.597    1357.00
10 # 4          AS  Africa do Sul  Pretoria    1.221      52.98
11
12
13
14
15
16
17
18
19
```

Dataframe baseado em arquivo CSV (2)

```
20 """
21 Indexando o dataframe na leitura.
22 """
23 brics = pd.read_csv("{}/pandas/brics.csv".format(os.getcwd())
24   , index_col=0)
25 print(brics)
26 #                pais    capital     area  populacao
27 # BR          Brasil   Brasilia    8.516     207.40
28 # RU          Russia     Moscou   17.100     143.50
29 # IN           India       Deli    3.286    1252.00
30 # CH           China     Pequin    9.597    1357.00
31 # AS   Africa do Sul   Pretoria    1.221      52.98
```

Acesso a valores de colunas (1)

```
1  print(brics["pais"])
2  # BR            Brasil
3  # RU            Russia
4  # IN             India
5  # CH             China
6  # AS     Africa do Sul
7  # Name: pais, dtype: object
8  print(type(brics["pais"]))
9  # <class 'pandas.core.series.Series'>
10
11 print(brics[["pais"]])
12 #                pais
13 # BR            Brasil
14 # RU            Russia
15 # IN             India
16 # CH             China
17 # AS     Africa do Sul
18 # <class 'pandas.core.frame.DataFrame'>
19
```

Acesso a valores de colunas (2)

```
20
21 print(brics[["pais", "capital"]])
22 #                pais    capital
23 # BR           Brasil   Brasilia
24 # RU           Russia     Moscou
25 # IN            India       Deli
26 # CH            China     Pequin
27 # AS   Africa do Sul   Pretoria
```

Acesso a valores de linhas

```
print(brics[1:4])
#        pais capital     area  populacao
# RU   Russia  Moscou  17.100      143.5
# IN    India    Deli   3.286     1252.0
# CH    China  Pequin   9.597     1357.0
```

Operador []

- `[ ]`: tem funcionalidade limitada
- Idealmente:
  - para arrays bidimensionais
  - `um_array[linha, coluna]`
- Em Pandas:
  - `loc` (baseado em conteúdo)
  - `iloc` (baseado em posição)

Acesso a valores de linhas com **loc**

```
print(brics.loc["RU"])
# pais          Russia       <- Pandas Series
# capital       Moscou
# area            17.1
# populacao      143.5
# Name: RU, dtype: object

print(brics.loc[["RU"]])
#       pais capital  area  populacao  <- Pandas Dataframes
# RU  Russia  Moscou  17.1       143.5

print(brics.loc[["RU", "IN", "CH"]])
#       pais capital   area  populacao
# RU  Russia  Moscou 17.100       143.5
# IN   India    Deli  3.286      1252.0
# CH   China  Pequin  9.597      1357.0
```

Acesso a valores de linhas e colunas com **loc**

```
print(brics.loc[["RU", "IN", "CH"], ["pais", "capital"]])
#        pais capital
# RU  Russia  Moscou
# IN   India    Deli
# CH   China  Pequin

print(brics.loc[:, ["pais", "capital"]])
#                pais    capital
# BR          Brasil   Brasilia
# RU          Russia     Moscou
# IN           India       Deli
# CH           China     Pequin
# AS  Africa do Sul   Pretoria
```

Recapitulando

- Colchetes `[ ]`
  - acesso a colunas: `brics[["pais", "capital"]]`
  - acesso a linhas: somente via fatiamento `brics[1:4]`
- **loc**
  - acesso a linhas: `brics.loc[["RU", "IN", "CH"]]`
  - acesso a colunas: `brics.loc[:, ["pais", "capital"]]`
  - linhas e colunas:
    `brics.loc[["RU", "IN", "CH"], ["pais", "capital"]]`

Acesso a valores de linhas com **iloc**

```
print(brics.iloc[[1]])
#       pais capital  area  populacao
# RU  Russia  Moscou  17.1      143.5

print(brics.iloc[[1, 2, 3]])
#       pais capital    area  populacao
# RU  Russia  Moscou  17.100      143.5
# IN   India    Deli   3.286     1252.0
# CH   China  Pequin   9.597     1357.0
```

Acesso a valores de linhas e colunas com **iloc**

```
print(brics.iloc[:, [0, 1]])
#              pais    capital
# BR         Brasil   Brasilia
# RU         Russia     Moscou
# IN          India       Deli
# CH          China     Pequin
# AS   Africa do Sul   Pretoria

print(brics.iloc[[1, 2, 3], [0, 1]])
#       pais capital
# RU  Russia  Moscou
# IN   India    Deli
# CH   China  Pequin
```

Filtrando dados no dataframe (1)

- brics.csv

```
1  import os
2  import pandas as pd
3  brics = pd.read_csv("{}/aula-2/codigos/pandas/brics.csv".
       format(os.getcwd())
4    , index_col=0)
5
6  print(brics)
7  #                pais    capital     area  populacao
8  # BR          Brasil   Brasilia    8.516     207.40
9  # RU          Russia     Moscou   17.100     143.50
10 # IN           India       Deli    3.286    1252.00
11 # CH           China     Pequin    9.597    1357.00
12 # AS   Africa do Sul   Pretoria    1.221      52.98
```

Filtrando dados no dataframe (2)

- Meta: filtrar os países cuja área seja maior do que 8km
  1. selecionar a coluna área
  2. comparar a coluna selecionada
  3. Utilizar o resultado para selecionar os países

Filtrando dados no dataframe (3)

- **Passo 1**: selecionar a coluna desejada

```
print(brics["area"])
# BR      8.516
# RU     17.100
# IN      3.286
# CH      9.597
# AS      1.221
# Name: area, dtype: float64
```

Alternativas:

```
brics.loc[:, "area"]
brics.iloc[:, 2]
```

Filtrando dados no dataframe (4)

- **Passo 2**: comparar a coluna selecionada

```
print(brics["area"] > 8)
# BR     True
# RU     True
# IN    False
# CH     True
# AS    False
# Name: area, dtype: bool
```

Filtrando dados no dataframe (5)

- **Passo 3**: filtrando o dataset

```
print(brics[brics["area"] > 8])
#        pais    capital     area  populacao
# BR   Brasil   Brasilia    8.516       207.4
# RU   Russia     Moscou   17.100       143.5
# CH    China     Pequin    9.597      1357.0
```

Operadores booleanos (1)

```
1  import numpy as np
2  import os
3  import pandas as pd
4  brics = pd.read_csv("{}/aula-2/codigos/pandas/brics.csv".
       format(os.getcwd())
5    , index_col=0)
6
7  filtro_por_area = np.logical_and(brics["area"] > 8,
8                                   brics["area"] < 10)
9  print(brics[filtro_por_area])
10 #        pais   capital    area  populacao
11 # BR   Brasil  Brasilia   8.516      207.4
12 # CH    China    Pequin   9.597     1357.0
```