# Uma introdução à Ciência de Dados Utilizando Python com Aplicações em Redes de Computadores e Segurança Digital

## Escola Regional de Informática do Espírito Santo - ERI-ES 2024

17 de outubro de 2024

Giovanni Comarela (UFES)

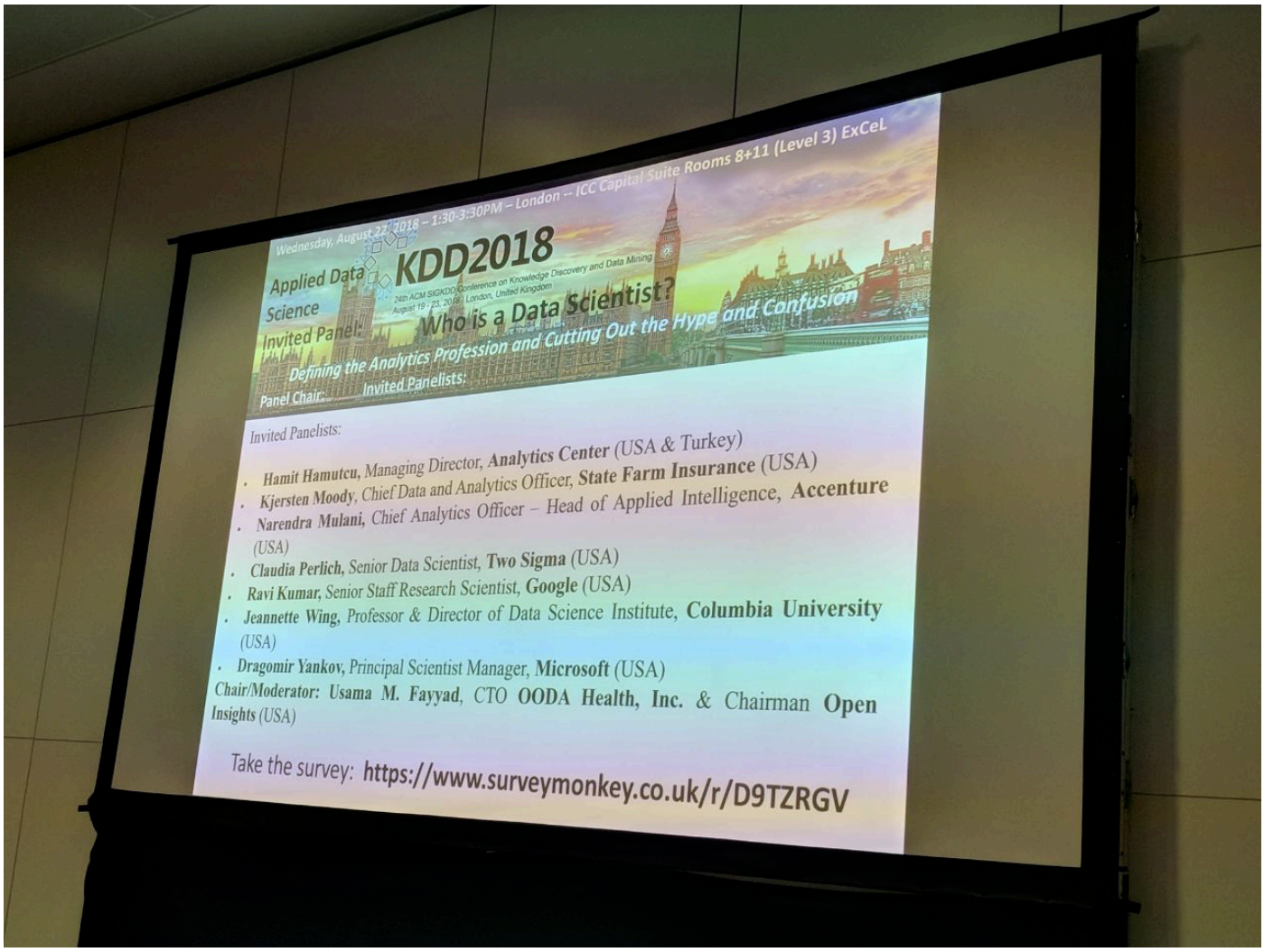
*gc@inf.ufes.br*

## URL

https://bit.ly/3A2I3Ms


## Ciência de Dados

Quem é um Cientista de Dados?




"Um Estatístico que sabe computação ou um Cientista da Computação que sabe estatística"

O que um Cientista de Dados deve estudar?

- Probabilidade
- Algoritmos
- Inferência Estatística
- Sistemas de Computação
- Aprendizado de Máquina
- Análise exploratória e visualização de dados

Qual a diferença entre Ciência de dados, estatística, aprendizado de máquina, mineração de dados e inteligência artificial?

Para os puristas, há muita! Na prática...

Preocupem-se com os problemas e em encontrar as ferramentas corretas para abordá-los!

Há vagas!

## Ciência de Dados em Redes e Segurança Digital

Para a comunidade de Redes e Segurança Digital, Ciência de Dados não é novidade! Nós simulamos e/ou analisamos dados provenientes de vários contextos. Exemplos:

- Estudo e aplicações de modelos baseados em Teoria de Filas
- Caracterização de Tráfego e Topologias
- Detecção de ataques, anomalias e *Spam*
- Caracterização e classificação de fluxos (detecção de ataques)
- Construção de *Caches* e CDNs
- Sem falar na camada de aplicação: recomendação, propaganda...

O Objetivo desse minicurso é mostrar como alguns desses tópicos podem ser feitos de forma **simples** utilizando Python

### Vou sair daqui sendo um especialista em Ciência de Dados?

**NÃO!** O objetivo desse minicurso é mostrar que há ferramentas simples para Ciência de Dados. O conhecimento dessas ferramentas sem o conhecimento do que elas significam é irrelevante.

## Por que Python?

1. Python é popular
2. Python é interpretado
3. Python é grátis
4. Python é portável
5. Python é **simples!**

## Dinâmica do nosso encontro

Ambiente participativo! Dúvidas são esperadas e bem vindas!

# Preliminares

Para esse minicurso, precisamos importar alguns dados e instalar algumas bibliotecas.

```
!rm -rf datasets-minicurso
!git clone https://github.com/gaabrielfranco/datasets-minicurso.git

!apt-get install libproj-dev proj-data proj-bin
!apt-get install libgeos-dev
!pip install cython
!pip install cartopy
```

```
Cloning into 'datasets-minicurso'...
remote: Enumerating objects: 14407, done.
remote: Total 14407 (delta 0), reused 0 (delta 0), pack-reused 14407 (from 1)
Receiving objects: 100% (14407/14407), 59.52 MiB | 18.05 MiB/s, done.
Resolving deltas: 100% (26/26), done.
Updating files: 100% (14398/14398), done.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
libproj-dev is already the newest version (9.1.1-1~jammy0).
proj-bin is already the newest version (9.1.1-1~jammy0).
proj-data is already the newest version (9.1.1-1~jammy0).
0 upgraded, 0 newly installed, 0 to remove and 49 not upgraded.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
libgeos-dev is already the newest version (3.11.1-1~jammy0).
0 upgraded, 0 newly installed, 0 to remove and 49 not upgraded.
Requirement already satisfied: cython in /usr/local/lib/python3.10/dist-packages (3.0.11)
Requirement already satisfied: cartopy in /usr/local/lib/python3.10/dist-packages (0.24.1)
Requirement already satisfied: numpy>=1.23 in /usr/local/lib/python3.10/dist-packages (from cartopy) (1.26.4)
Requirement already satisfied: matplotlib>=3.6 in /usr/local/lib/python3.10/dist-packages (from cartopy) (3.7.1)
Requirement already satisfied: shapely>=1.8 in /usr/local/lib/python3.10/dist-packages (from cartopy) (2.0.6)
Requirement already satisfied: packaging>=21 in /usr/local/lib/python3.10/dist-packages (from cartopy) (24.1)
Requirement already satisfied: pyshp>=2.3 in /usr/local/lib/python3.10/dist-packages (from cartopy) (2.3.1)
Requirement already satisfied: pyproj>=3.3.1 in /usr/local/lib/python3.10/dist-packages (from cartopy) (3.7.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.6->
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.6->cart
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.6-
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.6-
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.6->car
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.6->
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from pyproj>=3.3.1->cartopy) (2
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->mat
```

# Aquecimento

Exemplo extraído da página https://www.datagenetics.com/blog/september32012/

## Perguntas

- Como as pessoas escolhem senhas de cartões de crédito?
- Quais são as senhas mais populares?
- É possível explicar as senhas mais populares?

## Conjunto de dados

- Uma base de dados de senhas de cartões dos Estados Unidos
- Cartões de crédito nos Estados Unidos têm 4 dígitos
- Os dados não estão disponíveis. Tudo que eu apresentar aqui foi extraído/copiado da página citada
- "*Obviously, I don't have access to a credit card PIN number database. Instead I'm going to use a proxy. I'm going to use data condensed from released/exposed/discovered password tables and security breaches.*"

## Metodologia

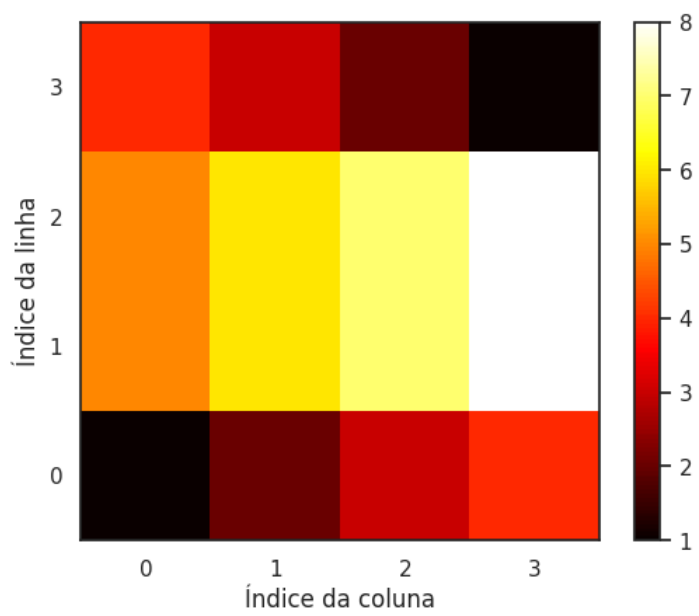
### Mapa de Calor

Uma matriz pode ser visualizada por meio de um mapa de calor

```
import numpy as np
from matplotlib import pyplot as plt

X = np.array([[1,2,3,4],[5,6,7,8],[5,6,7,8],[4,3,2,1]])
X
```

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

```
plt.imshow(X, cmap = 'hot', origin = 'lower')
plt.xticks([0, 1, 2, 3], ['0', '1', '2', '3'])
plt.yticks([0, 1, 2, 3], ['0', '1', '2', '3'])
plt.xlabel('Índice da coluna')
plt.ylabel('Índice da linha')
plt.colorbar();
```




## Representação visual dos dados

O que o autor do artigo fez foi o seguinte. Ele criou uma matriz $\mathbf{X}$ com a seguinte propriedade:

- Dada uma senha de quatro dígitos $abcd$,
  - $\mathbf{X}[cd, ab]$ conta a frequência da senha $abcd$.
- Exemplo: o elemento na coluna $12$ e linha $34$ conta quantas vezes a senha $1234$ aparece no conjunto de dados.

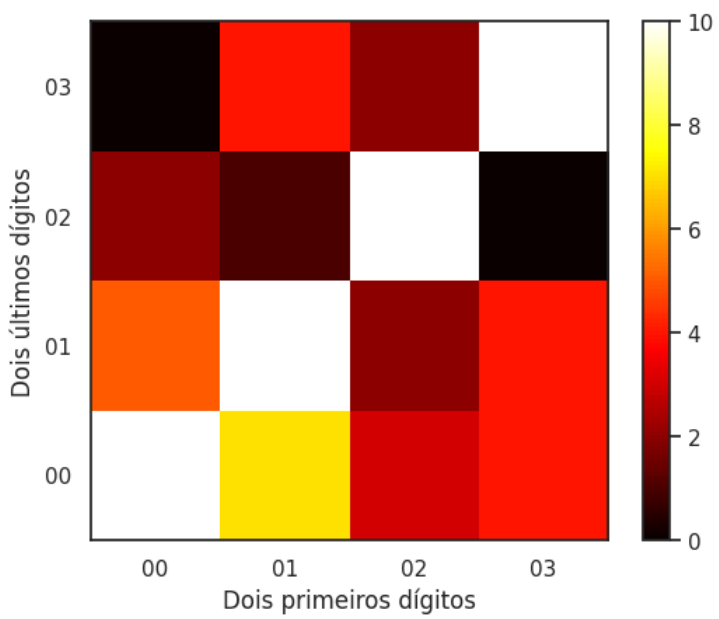
Assim, a matriz de interesse tem 100 linhas e 100 colunas.

## Exemplo

```
X = np.array([[10,7,3,4],[5,10,2,4],[2,1,10,0],[0,4,2,10]])
X
```

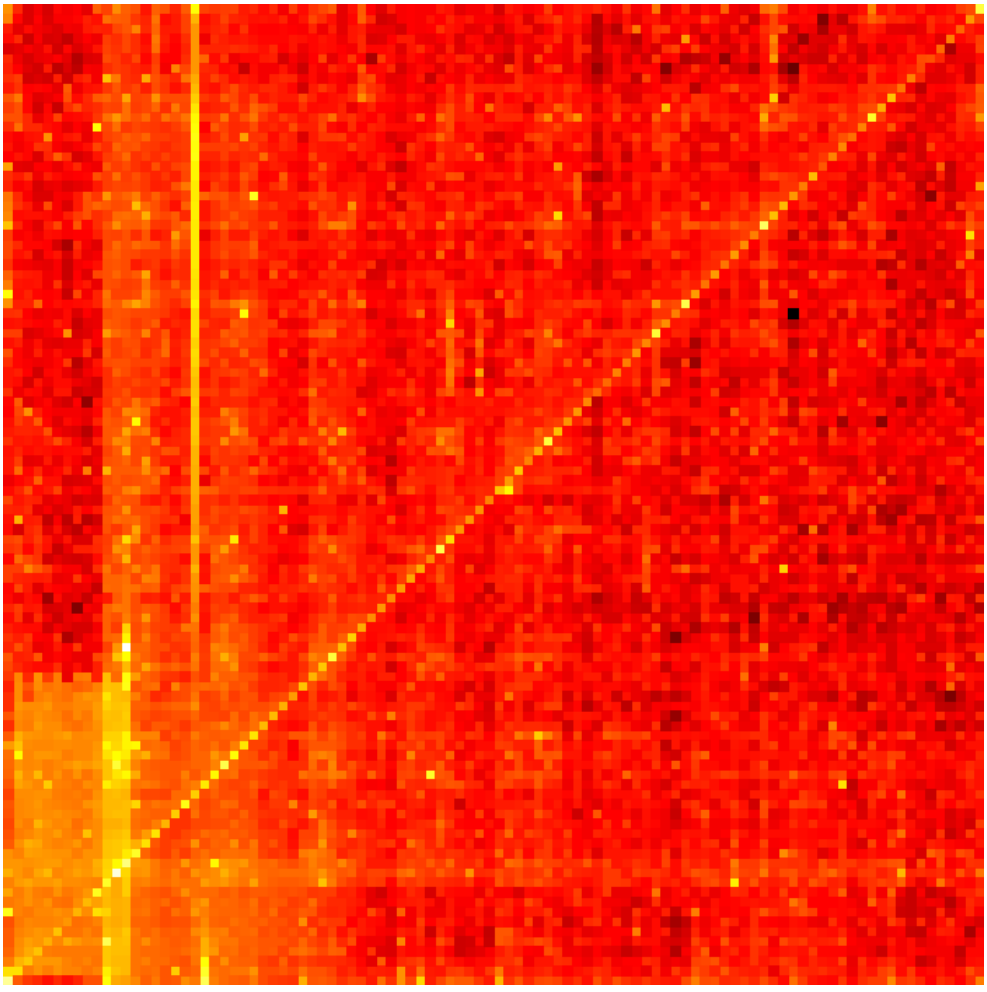
```
array([[10,  7,  3,  4],
       [ 5, 10,  2,  4],
       [ 2,  1, 10,  0],
       [ 0,  4,  2, 10]])
```

```
plt.imshow(X, cmap = 'hot', origin = 'lower')
plt.xticks([0, 1, 2, 3], ['00', '01', '02', '03'])
plt.yticks([0, 1, 2, 3], ['00', '01', '02', '03'])
plt.xlabel('Dois primeiros dígitos')
plt.ylabel('Dois últimos dígitos')
plt.colorbar();
```




## Resultados

O resultado principal do artigo consiste de um único gráfico



Primeiro, vamos ter certeza que entendemos o gráfico

- **Eixo** $x$ representa os dois **primeiros** dígitos da senha
- **Eixo** $y$ representa os dois **últimos** dígitos da senha
- Quanto mais clara for uma entrada da imagem, mais frequente é a senha relacionada

### Interpretação

#### Parte I - Diagonal principal

Senhas do tipo $xyxy$, ou seja, fáceis de lembrar

#### Parte II - Coluna vertical mais clara

Senhas do tipo $19xy$, ou seja, datas de nascimento

Veja que o início da coluna $20xy$ tem o mesmo comportamento

#### Parte III - Retângulo na parte inferior-esquerda

Senhas do tipo $abcd$ com $ab \leq 12$ e $cd \leq 31$.

Representam datas, lembrando que nos EUA, usam o padrão MM/DD.

#### Parte IV - Alguns exemplos específicos (pontos claros na figura)

- 1234
- 0987
- 4321
- 2345
- 2468
- 5678
- 7890
- 2580

### Conclusão

#### Parte I

Em geral, somos "preguiçosos" na hora de escolher senhas. Temos a tendência de escolher senhas fáceis de lembrar ou senhas que nos lembram de elementos importantes em nossas vidas.

#### Parte II

Essa conclusão foi possibilitada pela escolha da metodologia. Não pela escolha de uma LP ou mesmo por bibliotecas específicas. Nada muito complexo/elaborado foi empregado.

Esse é o tipo de conhecimento que eu espero que vocês ganhem. O difícil é saber o que programar, não como programar (já tem biblioteca pra quase tudo).

## Leitura e manipulação

A biblioteca **Pandas** oferece vários recursos para manipulação de dados de forma simples e eficiente.

O código abaixo é baseado na documentação da biblioteca.

```
import numpy as np
import pandas as pd
import seaborn as sns
sns.set()
```

## DataFrame

**DataFrame** é uma estrutura bi-dimensional com elementos rotulados. Os rótulos das linhas estão estão em **index** e os rótulos das colunas são chamados de **columns**.

Há várias maneiras de criar um objeto **DataFrame**.

```
dates = pd.date_range('20130101', periods = 6)
df = pd.DataFrame(np.random.randn(6, 4), index = dates, columns = list('ABCD'))
df
```

| | A | B | C | D |
|---|---|---|---|---|
| **2013-01-01** | 0.424792 | 1.504690 | -0.421368 | -0.293471 |
| **2013-01-02** | -1.234925 | -0.207308 | -0.470325 | 0.903739 |
| **2013-01-03** | 0.493421 | -0.761025 | 0.789710 | -1.096527 |
| **2013-01-04** | -1.626671 | -1.475435 | 0.854075 | 1.130309 |
| **2013-01-05** | 1.229504 | -2.335250 | 1.814233 | -0.669218 |
| **2013-01-06** | -0.968688 | 0.981308 | 0.067578 | 1.192624 |

```
df = pd.read_csv('datasets-minicurso/example.csv')
df
```

| | foo | bar |
|---|---|---|
| **0** | 1.274366 | 0.593544 |
| **1** | -0.409548 | 0.999609 |

```
df = pd.read_csv('datasets-minicurso/example.csv', index_col = 0, header = None)
df
```

| | 1 |
|---|---|
| **0** | |
| **foo** | bar |
| **1.274366** | 0.593544 |
| **-0.409548** | 0.999609 |

## Manipulação Básica

```
dates = pd.date_range('20130101', periods=8)
df = pd.DataFrame(np.random.randn(8,4), index=dates, columns=list('ABCD'))
df
```

| | A | B | C | D |
|---|---|---|---|---|
| **2013-01-01** | 1.607544 | -0.013965 | -2.234492 | 1.334927 |
| **2013-01-02** | 1.387772 | -1.804482 | 1.249637 | -0.500410 |
| **2013-01-03** | -0.671387 | -0.859255 | -1.864129 | 0.354350 |
| **2013-01-04** | 0.292506 | 1.832161 | 0.174687 | -0.280750 |
| **2013-01-05** | 1.622893 | -0.424435 | 0.499285 | 1.311844 |
| **2013-01-06** | -0.574322 | -0.418060 | -0.917478 | 0.835937 |
| **2013-01-07** | 0.868003 | -1.333691 | 0.017581 | 0.332499 |
| **2013-01-08** | -0.237203 | -0.081228 | 0.551844 | 0.996038 |

```
df.head()
```

| | A | B | C | D |
|---|---|---|---|---|
| **2013-01-01** | 1.607544 | -0.013965 | -2.234492 | 1.334927 |
| **2013-01-02** | 1.387772 | -1.804482 | 1.249637 | -0.500410 |
| **2013-01-03** | -0.671387 | -0.859255 | -1.864129 | 0.354350 |
| **2013-01-04** | 0.292506 | 1.832161 | 0.174687 | -0.280750 |
| **2013-01-05** | 1.622893 | -0.424435 | 0.499285 | 1.311844 |

```
df.tail()
```

| | A | B | C | D |
|---|---|---|---|---|
| **2013-01-04** | 0.292506 | 1.832161 | 0.174687 | -0.280750 |
| **2013-01-05** | 1.622893 | -0.424435 | 0.499285 | 1.311844 |
| **2013-01-06** | -0.574322 | -0.418060 | -0.917478 | 0.835937 |
| **2013-01-07** | 0.868003 | -1.333691 | 0.017581 | 0.332499 |
| **2013-01-08** | -0.237203 | -0.081228 | 0.551844 | 0.996038 |

```
df.index
```

```
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06', '2013-01-07', '2013-01-08'],
              dtype='datetime64[ns]', freq='D')
```

```
df.columns
```

```
Index(['A', 'B', 'C', 'D'], dtype='object')
```

```
df.values
```

```
array([[ 1.60754381, -0.01396486, -2.23449208,  1.33492729],
       [ 1.38777205, -1.80448225,  1.24963651, -0.5004103 ],
       [-0.67138663, -0.85925536, -1.86412909,  0.35434988],
       [ 0.29250613,  1.83216122,  0.17468688, -0.2807499 ],
       [ 1.62289312, -0.42443488,  0.49928539,  1.31184388],
       [-0.57432199, -0.41806011, -0.91747817,  0.83593669],
       [ 0.8680035 , -1.33369095,  0.01758089,  0.33249933],
       [-0.23720251, -0.08122765,  0.55184446,  0.99603816]])
```

```
df.T
```

| | 2013-01-01 | 2013-01-02 | 2013-01-03 | 2013-01-04 | 2013-01-05 | 2013-01-06 | 2013-01-07 | 2013-01-08 |
|---|---|---|---|---|---|---|---|---|
| **A** | 1.607544 | 1.387772 | -0.671387 | 0.292506 | 1.622893 | -0.574322 | 0.868003 | -0.237203 |
| **B** | -0.013965 | -1.804482 | -0.859255 | 1.832161 | -0.424435 | -0.418060 | -1.333691 | -0.081228 |
| **C** | -2.234492 | 1.249637 | -1.864129 | 0.174687 | 0.499285 | -0.917478 | 0.017581 | 0.551844 |
| **D** | 1.334927 | -0.500410 | 0.354350 | -0.280750 | 1.311844 | 0.835937 | 0.332499 | 0.996038 |

```
df.T.describe()
```

| | 2013-01-01 | 2013-01-02 | 2013-01-03 | 2013-01-04 | 2013-01-05 | 2013-01-06 | 2013-01-07 | 2013-01-08 |
|---|---|---|---|---|---|---|---|---|
| **count** | 4.000000 | 4.000000 | 4.000000 | 4.000000 | 4.000000 | 4.000000 | 4.000000 | 4.000000 |
| **mean** | 0.173504 | 0.083129 | -0.760105 | 0.504651 | 0.752397 | -0.268481 | -0.028902 | 0.307363 |
| **std** | 1.754894 | 1.523858 | 0.908952 | 0.918880 | 0.916468 | 0.765256 | 0.938027 | 0.572014 |
| **min** | -2.234492 | -1.804482 | -1.864129 | -0.280750 | -0.424435 | -0.917478 | -1.333691 | -0.237203 |
| **25%** | -0.569097 | -0.826428 | -1.110474 | 0.060828 | 0.268355 | -0.660111 | -0.320237 | -0.120221 |
| **50%** | 0.660481 | 0.374613 | -0.765321 | 0.233597 | 0.905565 | -0.496191 | 0.175040 | 0.235308 |
| **75%** | 1.403081 | 1.284170 | -0.414952 | 0.677420 | 1.389606 | -0.104561 | 0.466375 | 0.662893 |
| **max** | 1.607544 | 1.387772 | 0.354350 | 1.832161 | 1.622893 | 0.835937 | 0.868003 | 0.996038 |

```
df.sort_index(axis = 0, ascending = False)
```

| | A | B | C | D |
|---|---|---|---|---|
| **2013-01-08** | -0.237203 | -0.081228 | 0.551844 | 0.996038 |
| **2013-01-07** | 0.868003 | -1.333691 | 0.017581 | 0.332499 |
| **2013-01-06** | -0.574322 | -0.418060 | -0.917478 | 0.835937 |
| **2013-01-05** | 1.622893 | -0.424435 | 0.499285 | 1.311844 |
| **2013-01-04** | 0.292506 | 1.832161 | 0.174687 | -0.280750 |
| **2013-01-03** | -0.671387 | -0.859255 | -1.864129 | 0.354350 |
| **2013-01-02** | 1.387772 | -1.804482 | 1.249637 | -0.500410 |
| **2013-01-01** | 1.607544 | -0.013965 | -2.234492 | 1.334927 |

```
df = pd.DataFrame({'col1' : ['A', 'A', 'B', np.nan, 'D', 'C'],
                   'col2' : [2, 1, 9, 8, 7, 4],
                   'col3': [0, 1, 9, 4, 2, 3],})
df
```

| | col1 | col2 | col3 |
|---|---|---|---|
| **0** | A | 2 | 0 |
| **1** | A | 1 | 1 |
| **2** | B | 9 | 9 |
| **3** | NaN | 8 | 4 |
| **4** | D | 7 | 2 |
| **5** | C | 4 | 3 |

```
df.sort_values(by = ['col1'])
```

| | col1 | col2 | col3 |
|---|---|---|---|
| **0** | A | 2 | 0 |
| **1** | A | 1 | 1 |
| **2** | B | 9 | 9 |
| **5** | C | 4 | 3 |
| **4** | D | 7 | 2 |
| **3** | NaN | 8 | 4 |

```
df.sort_values(by = ['col1', 'col2'], ascending = [False, True])
```

| | col1 | col2 | col3 |
|---|---|---|---|
| **4** | D | 7 | 2 |
| **5** | C | 4 | 3 |
| **2** | B | 9 | 9 |
| **1** | A | 1 | 1 |
| **0** | A | 2 | 0 |
| **3** | NaN | 8 | 4 |

## Seleção em DataFrames

```
dates = pd.date_range('20130101', periods=6)
df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))
df
```

| | A | B | C | D |
|---|---|---|---|---|
| **2013-01-01** | -0.578175 | 0.378244 | -0.999294 | 0.281209 |
| **2013-01-02** | 0.282660 | -0.469350 | 0.219275 | 1.386948 |
| **2013-01-03** | 0.417278 | -0.601274 | 0.195273 | -0.914461 |
| **2013-01-04** | 0.910754 | 0.998019 | -0.041029 | 0.929553 |
| **2013-01-05** | -0.005196 | -1.140753 | 1.863788 | -0.028558 |
| **2013-01-06** | -0.801022 | -0.892576 | 1.092817 | -2.380730 |

```
df["A"] #Ou df.A
```

| | A |
|---|---|
| **2013-01-01** | -0.578175 |
| **2013-01-02** | 0.282660 |
| **2013-01-03** | 0.417278 |
| **2013-01-04** | 0.910754 |
| **2013-01-05** | -0.005196 |
| **2013-01-06** | -0.801022 |

**dtype:** float64

```
df[0:3]
```

| | A | B | C | D |
|---|---|---|---|---|
| **2013-01-01** | -0.578175 | 0.378244 | -0.999294 | 0.281209 |
| **2013-01-02** | 0.282660 | -0.469350 | 0.219275 | 1.386948 |
| **2013-01-03** | 0.417278 | -0.601274 | 0.195273 | -0.914461 |

```
df['20130102':'20130104']
```

| | A | B | C | D |
|---|---|---|---|---|
| **2013-01-02** | 0.282660 | -0.469350 | 0.219275 | 1.386948 |
| **2013-01-03** | 0.417278 | -0.601274 | 0.195273 | -0.914461 |
| **2013-01-04** | 0.910754 | 0.998019 | -0.041029 | 0.929553 |

```
df.loc[:,['A','B']]
```

| | A | B |
|---|---|---|
| **2013-01-01** | -0.578175 | 0.378244 |
| **2013-01-02** | 0.282660 | -0.469350 |
| **2013-01-03** | 0.417278 | -0.601274 |
| **2013-01-04** | 0.910754 | 0.998019 |
| **2013-01-05** | -0.005196 | -1.140753 |
| **2013-01-06** | -0.801022 | -0.892576 |

```
df.loc['20130102':'20130104',['A','B']]
```

| | A | B |
|---|---|---|
| **2013-01-02** | 0.282660 | -0.469350 |
| **2013-01-03** | 0.417278 | -0.601274 |
| **2013-01-04** | 0.910754 | 0.998019 |

```
df.iloc[3:5,0:2]
```

| | A | B |
|---|---|---|
| **2013-01-04** | 0.910754 | 0.998019 |
| **2013-01-05** | -0.005196 | -1.140753 |

```
df.at[df.index[0], 'A']
```

```
-0.5781748545678091
```

## Outras operações interessantes

### Junção

```
df1 = pd.DataFrame({'lkey': ['foo', 'bar', 'baz', 'foo'],
                    'value': [1, 2, 3, 5]})
df1
```

| | lkey | value |
|---|---|---|
| **0** | foo | 1 |
| **1** | bar | 2 |
| **2** | baz | 3 |
| **3** | foo | 5 |

```
df2 = pd.DataFrame({'rkey': ['foo', 'bar', 'baz', 'foo'],
                    'value': [5, 6, 7, 8]})
df2
```

| | rkey | value |
|---|---|---|
| **0** | foo | 5 |
| **1** | bar | 6 |
| **2** | baz | 7 |
| **3** | foo | 8 |

Clique duas vezes (ou pressione "Enter") para editar

```
df1.merge(df2, left_on='lkey', right_on='rkey')
```

| | lkey | value_x | rkey | value_y |
|---|---|---|---|---|
| **0** | foo | 1 | foo | 5 |
| **1** | foo | 1 | foo | 8 |
| **2** | bar | 2 | bar | 6 |
| **3** | baz | 3 | baz | 7 |
| **4** | foo | 5 | foo | 5 |
| **5** | foo | 5 | foo | 8 |

### Agregação por chave

```
df = pd.DataFrame({
    'A' : ['foo', 'bar', 'foo', 'bar', 'foo', 'bar', 'foo', 'foo'],
    'B' : ['one', 'one', 'two', 'three', 'two', 'two', 'one', 'three'],
    'C' : np.random.randn(8),
    'D' : np.random.randn(8)})
df
```

| | A | B | C | D |
|---|---|---|---|---|
| **0** | foo | one | 0.298437 | 0.813236 |
| **1** | bar | one | 0.581837 | -0.310868 |
| **2** | foo | two | 0.008352 | -0.526226 |
| **3** | bar | three | 0.043817 | -0.616195 |
| **4** | foo | two | -0.240834 | -1.224001 |
| **5** | bar | two | -0.310988 | 0.574355 |
| **6** | foo | one | 1.532417 | -0.534309 |
| **7** | foo | three | -0.393045 | -1.766432 |

```
df.groupby(['A']).sum()
```

| | B | C | D |
|---|---|---|---|
| **A** | | | |
| **bar** | onethreetwo | 0.314665 | -0.352708 |
| **foo** | onetwotwoonethree | 1.205327 | -3.237731 |

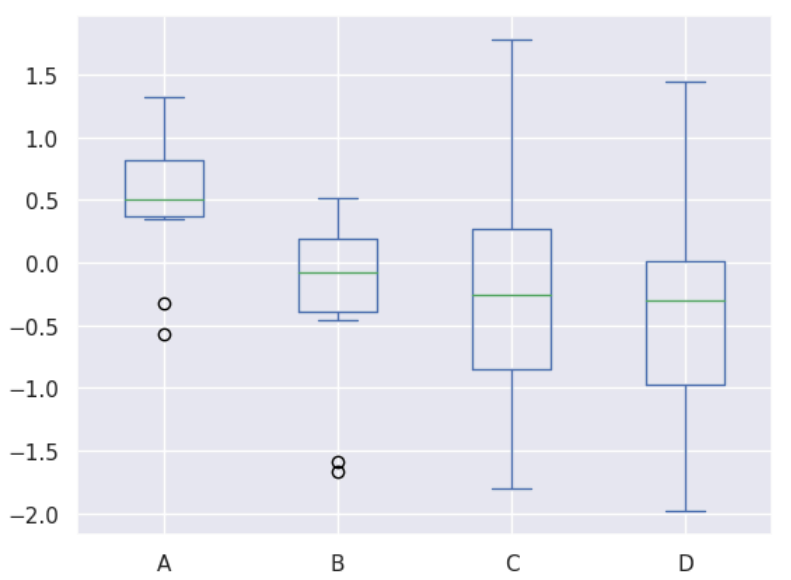
## Visualização de Dados em DataFrames

```
df = pd.DataFrame(np.random.randn(10, 4), index=
                  pd.date_range('1/1/2000', periods=10),
                  columns=['A', 'B', 'C', 'D'])
df
```

| | A | B | C | D |
|---|---|---|---|---|
| **2000-01-01** | 0.466990 | -0.026604 | 1.781048 | -0.031454 |
| **2000-01-02** | 0.435063 | -1.672394 | -0.894709 | 0.025484 |
| **2000-01-03** | 0.853884 | -0.138409 | -1.617975 | 1.445654 |
| **2000-01-04** | 0.538846 | 0.384904 | -0.727933 | 0.945962 |
| **2000-01-05** | 0.978346 | -0.463423 | 0.211011 | -1.524977 |
| **2000-01-06** | 1.322955 | 0.249357 | -0.374837 | -0.937160 |
| **2000-01-07** | -0.327140 | -0.003086 | 0.288361 | -0.168958 |
| **2000-01-08** | 0.350259 | -0.151506 | 1.200357 | -1.980959 |
| **2000-01-09** | 0.717062 | 0.517256 | -0.133404 | -0.425252 |
| **2000-01-10** | -0.566443 | -1.589430 | -1.798824 | -0.986695 |

```
df.plot();
```


```
df.plot(kind = 'box');
```




# Redução de Dimensionalidade

## Introdução

Em muitas situações, o conjunto de dados de interesse pode ser visto como uma matriz com $n$ linhas e $d$ colunas.

As linhas representam objetos (e.g., pessoas) e as colunas os atributos (e.g., peso, altura, idade, etc.).

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1d} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & x_{n3} & \dots & x_{nd} \end{bmatrix}$$

O que acontece quando $n$ e $d$ são muito grandes?

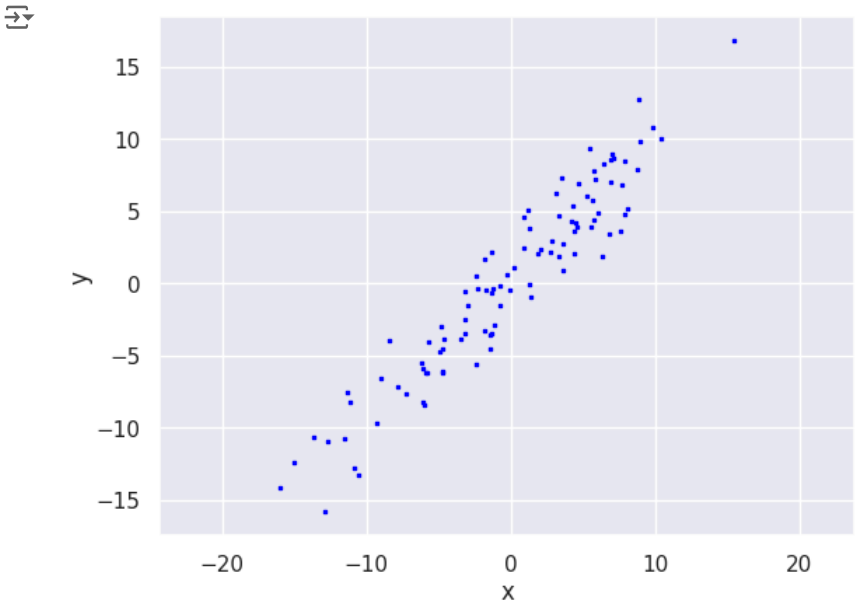
1. Há um problema relacionado ao armazenamento e à eficiência algorítmica.
2. **Maldição da dimensionalidade.**

## PCA - Análise de Componentes Principais

```
import numpy as np
import pandas as pd
from numpy import linalg
import scipy.linalg
from matplotlib import pyplot as plt
import seaborn as sns

sns.set()
```

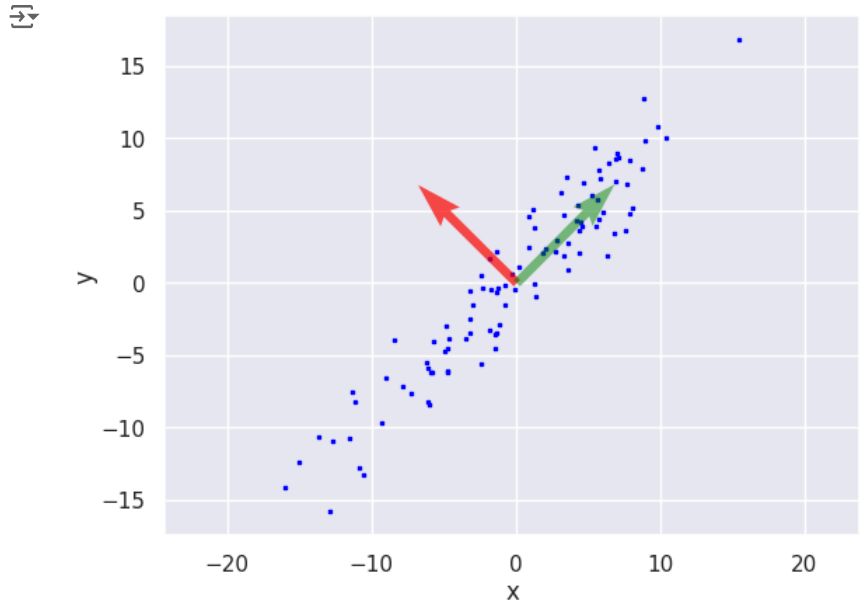
## Considere a seguinte figura

```
np.random.seed(0)
sigma = np.array([[41, 39], [39, 41]])
x = np.random.multivariate_normal([0, 0], sigma, 100)

plt.scatter(x[:, 0], x[:, 1], color = 'blue', marker = 's', s = 2)
plt.axis('equal')
plt.xlabel('x', labelpad = 2)
plt.ylabel('y', labelpad = 2);
```




## Qual a direção de maior variância dos dados?

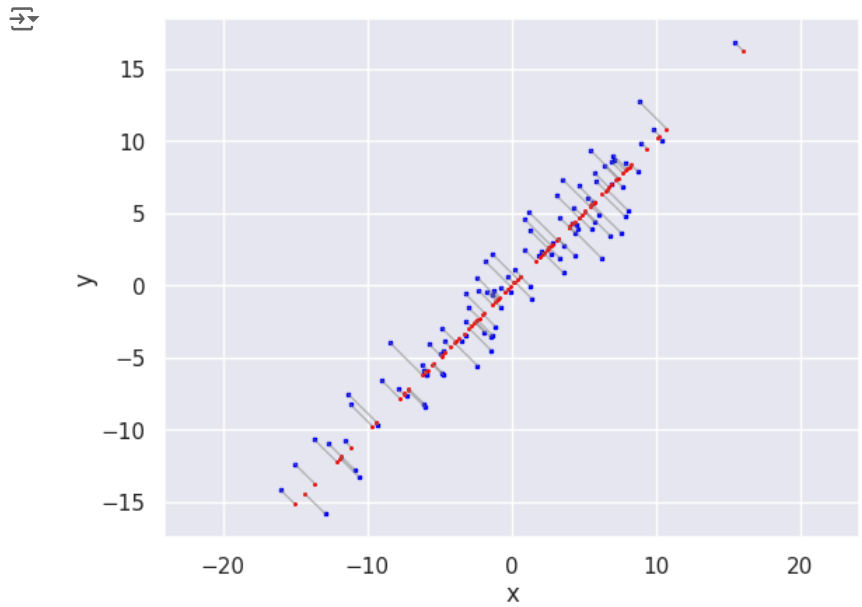
```
def getMaxVarianceDirections(x):
    n, d = x.shape
    sampleSigma = np.matmul(x.T, x) / float(n)
    values, vectors = linalg.eigh(sampleSigma)
    return values, vectors

# Plotando os Pontos
plt.scatter(x[:, 0], x[:, 1], color = 'blue', marker = 's', s = 2)
plt.axis('equal')
plt.xlabel('x', labelpad = 2)
plt.ylabel('y', labelpad = 2)

# Encontrando as direções ortogonais de variância máxima
values, vectors = getMaxVarianceDirections(x)

# Plotando os vetores das direções
plt.quiver([0], [0], vectors[0][0], vectors[0][1],
       color = "red", scale = 5,
       zorder = 10, alpha = 0.7, width = 0.013)

plt.quiver([0], [0], vectors[1][0], vectors[1][1],
       color = "green", scale = 5,
       zorder = 10, alpha = 0.5, width = 0.013);
```



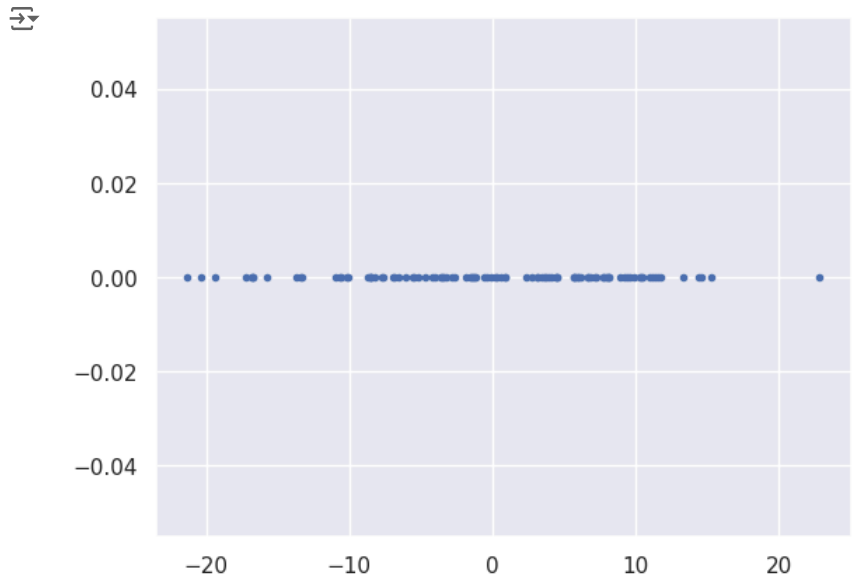

## Veja que se formos capazes de identificar essa primeira direção, podemos trabalhar com um conjunto de pontos no espaço real!

Ao fazer isso, perderemos informação, mas trabalharemos com menos dados e com dados mais simples (pontos no espaço real).

```
# Plotando os pontos
plt.scatter(x[:, 0], x[:, 1], color = 'blue', marker = 's', s = 2)
plt.axis('equal')
plt.xlabel('x', labelpad = 2)
plt.ylabel('y', labelpad = 2)

# Encontrando as coordenadas dos pontos projetados
U1 = vectors[:, 1].reshape([2, -1])
P1 = np.matmul(U1, U1.T)
xPrime = np.matmul(P1, x.T).T

# Plotando os pontos no subespaço (em vermelho) e o erro associado às projeções
plt.scatter(xPrime[:, 0], xPrime[:, 1], color = 'red', marker = '.', s = 5)
for original, projected in zip(x, xPrime):
    xx, yy = zip(original, projected)
    plt.plot(xx, yy, color = 'gray', lw = 1, alpha = 0.5);
```




```
newX = np.matmul(x, U1).ravel()
plt.scatter(newX, [0.0] * len(newX), marker = '.');
```




## A análise de componentes principais nos permite encontrar as direções responsáveis pela maior variabilidade de um conjunto de pontos

1. Como encontrar tais direções?
2. Dado um conjunto de dados com muitas dimensões, como encontrar um número adequado de dimensões para efetuar a redução?

```
from sklearn.decomposition import PCA
```

```
pca = PCA(n_components = 1)
pca.fit(x);
```

```
pca.components_
```

```
array([[0.70436626, 0.70983672]])
```

```
pca.explained_variance_
```

```
array([84.3970249])
```

```
pca.explained_variance_ratio_
```

```
array([0.97590927])
```

## SVD – Decomposição em Valores Singulares

### Definição

Seja $X$ uma matriz $n \times d$ de *rank* $l$. A decomposição em valores singulares de $X$ consiste em decompor $X$ da seguinte forma:

$X = USV^T$,

onde:

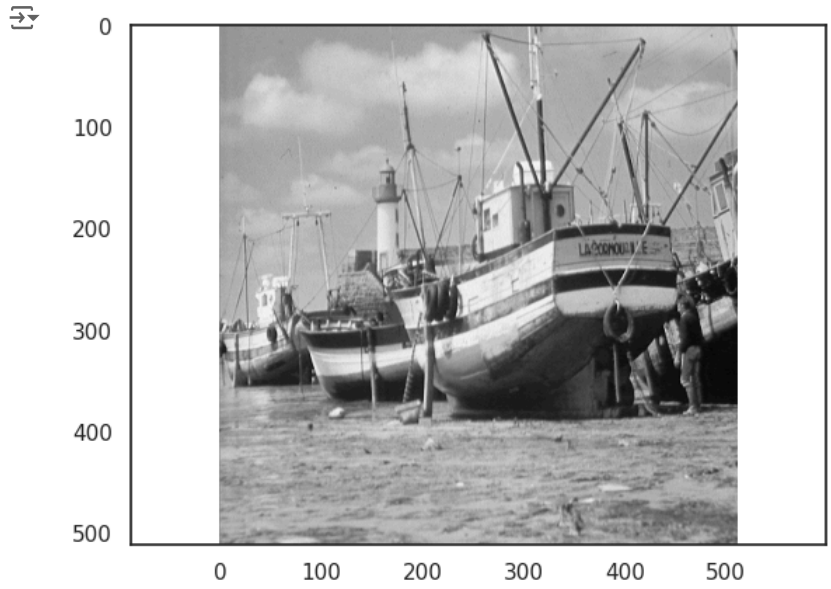
- $U$, $n \times l$, é uma matriz ortogonal;
- $S$, $l \times l$, é uma matriz diagonal, com elementos positivos e ordenados em ordem decrescente;
- $V$, $d \times l$, é uma matriz ortogonal;

## Para serve SVD?

1. Pode-se fazer PCA via SVD! (não veremos isso hoje)
2. Pode-se usar SVD para compressão!

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
boat = np.loadtxt('datasets-minicurso/boat.dat')
sns.set_style("white")
plt.imshow(boat, cmap = plt.cm.Greys_r)
plt.axis('equal');
```




## Obtendo a decomposição em valores singulares da matriz

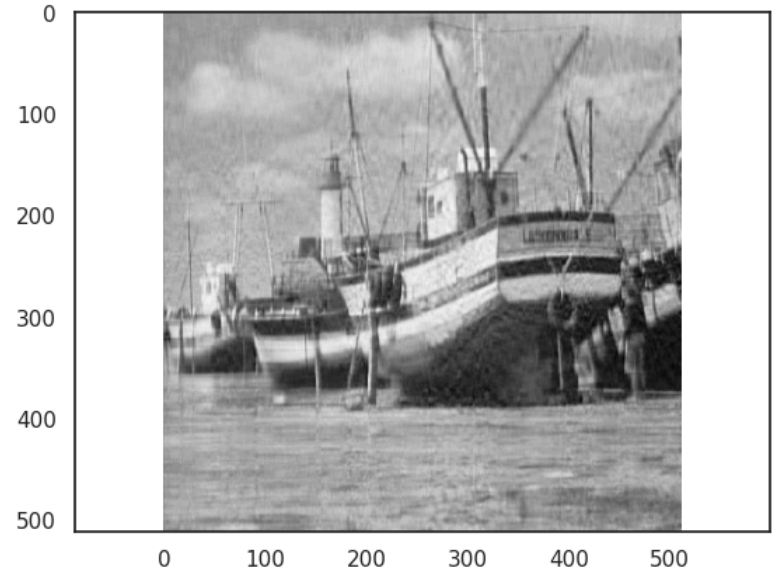
```
u, s, vt = np.linalg.svd(boat, full_matrices=False)
```

## Obtendo uma aproximação de *rank r*

```
r = 40

ur = u[:, :r]        # 512 x r
sr = np.diag(s[:r])  # r x r
vtr = vt[:r, :]      # r x 512

xr = np.dot(ur, np.dot(sr, vtr))
```

## Visualizando a imagem reconstruída

```
plt.imshow(xr, cmap = plt.cm.Greys_r)
plt.axis('equal');
```




```
sns.set()
```

## Exemplo: Minerando Anomalias de Tráfego

### Referência

**Lakhina, A., Crovella, M., and Diot, C.** *Mining anomalies using traffic feature distributions* (2005)

### Matriz de tráfego

Considere a matriz de tráfego $X$ abaixo. Linhas denotam intervalos de tempo e colunas denotam pares origem-destino.

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns

sns.set()
```

```
X = pd.read_csv('datasets-minicurso/abilene_traffic.csv', index_col = 0)
X.head()
```

| | ATLA-ATLA | ATLA-CHIN | ATLA-DNVR | ATLA-HSTN | ATLA-IPLS | ATLA-KSCY | ATLA-LOSA | ATLA-NYCM | ATLA-SNVA | ATLA-STTL |
|---|---|---|---|---|---|---|---|---|---|---|
| **2003-09-01 00:00:00** | 8466132.0 | 29346537.0 | 15792104.0 | 3646187.0 | 21756443.0 | 10792818.0 | 14220940.0 | 25014340.0 | 13677284.0 | 10591345.0 |
| **2003-09-01 00:10:00** | 20524567.0 | 28726106.0 | 8030109.0 | 4175817.0 | 24497174.0 | 8623734.0 | 15695839.0 | 36788680.0 | 5607086.0 | 10714795.0 |
| **2003-09-01 00:20:00** | 12864863.0 | 27630217.0 | 7417228.0 | 5337471.0 | 23254392.0 | 7882377.0 | 16176022.0 | 31682355.0 | 6354657.0 | 12205515.0 |
| **2003-09-01 00:30:00** | 10856263.0 | 32243146.0 | 7136130.0 | 3695059.0 | 28747761.0 | 9102603.0 | 16200072.0 | 27472465.0 | 9402609.0 | 10934084.0 |
| **2003-09-01** | 10068523.0 | 30164311.0 | 8061482.0 | 2922971.0 | 25642229.0 | 9104036.0 | 13270530.0 | 29171295.0 | 7624934.0 | 11227807.0 |

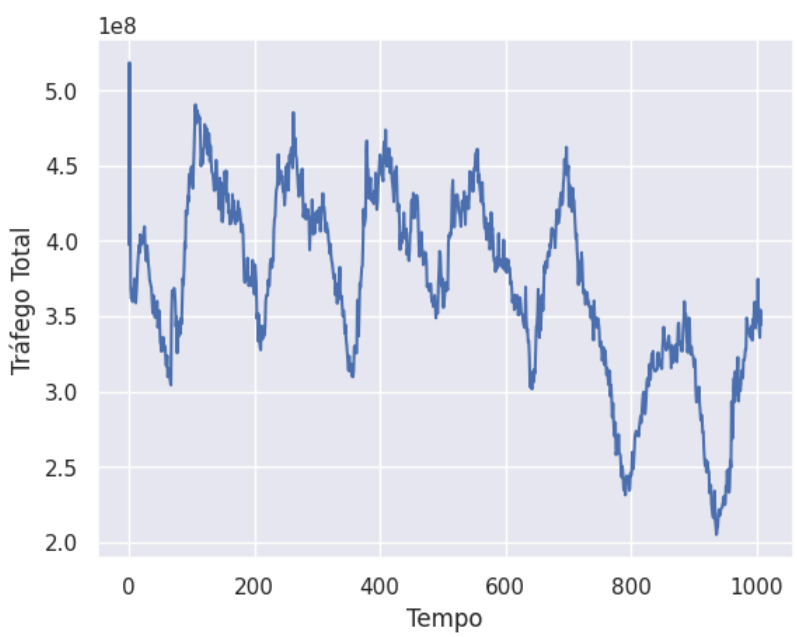
```
X.tail()
```

| | ATLA-ATLA | ATLA-CHIN | ATLA-DNVR | ATLA-HSTN | ATLA-IPLS | ATLA-KSCY | ATLA-LOSA | ATLA-NYCM | ATLA-SNVA | ATLA-STTL | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **2003-09-07 23:10:00** | 8849096.0 | 33461807.0 | 5866138.0 | 3786793.0 | 19097140.0 | 10561532.0 | 26092040.0 | 28640962.0 | 8343867.0 | 8820650.0 | ... |
| **2003-09-07 23:20:00** | 9776675.0 | 31474607.0 | 5874654.0 | 11277465.0 | 14314837.0 | 9106198.0 | 26412752.0 | 26168288.0 | 8638782.0 | 9193717.0 | ... |
| **2003-09-07 23:30:00** | 9144621.0 | 32117262.0 | 5762691.0 | 7154577.0 | 17771350.0 | 10149256.0 | 29501669.0 | 25998158.0 | 11343171.0 | 9423042.0 | ... |
| **2003-09-07 23:40:00** | 8802106.0 | 29932510.0 | 5279285.0 | 5950898.0 | 20222187.0 | 10636832.0 | 19613671.0 | 26124024.0 | 8732768.0 | 8217873.0 | ... |
| **2003-09-07** | 8716705.6 | 22660870.0 | 6240626.4 | 5657280.6 | 17406986.0 | 8808588.5 | 15962917.0 | 18267629.0 | 7767967.2 | 7470650.1 | |

## Tráfego total

O tráfego total ao longo do tempo é dado pela figura a seguir.

```
Xnorm = np.linalg.norm(X, axis = 1)

plt.plot(Xnorm)
plt.xlabel('Tempo')
_ = plt.ylabel('Tráfego Total')
```




## Como identificar anomalias?

Há intervalos de tempo em que o tráfego da matriz $X$ pode ser considerado anômalo?

Vamos assumir que o tráfego é não anômalo na maior parte do tempo. Daí, podemos tentar explorar o fenômeno de baixa dimensionalidade da seguinte forma:

1. Obtenha a decomposição em valores singulares de $X$
$$X = USV^T$$
2. Obtenha a aproximação de $X$ com *rank r*
$$X_r = U_r S_r V_r^T$$
3. Compute a porção de $X$ que não pode ser explicada por $X_r$
$$O = X - X_r$$
4. Identifique as linhas de $O$ com as maiores normas

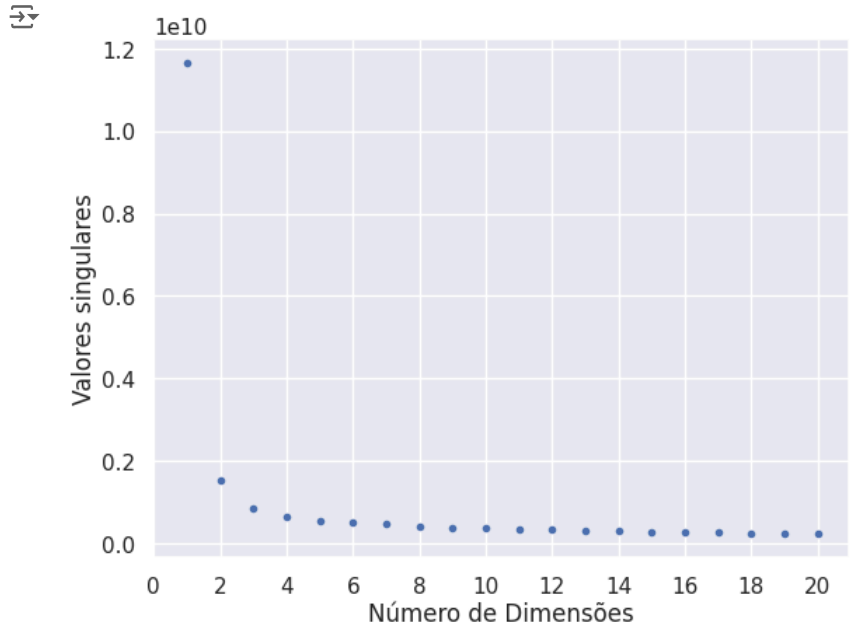
## Aplicando SVD

Primeiro, precisamos encontrar a decomposição em valores singulares de $X$ e determinar o valor de $r$

```
U, s, Vt = np.linalg.svd(X, full_matrices = False)
V = Vt.T
```

## Determinando o *rank* efetivo

```
plt.plot(range(1, 1 + 20), s[:20], 'b.')
plt.ylabel('Valores singulares', labelpad = 2)
plt.xlabel('Número de Dimensões', labelpad = 2)
plt.xticks(range(0, 21, 2));
```



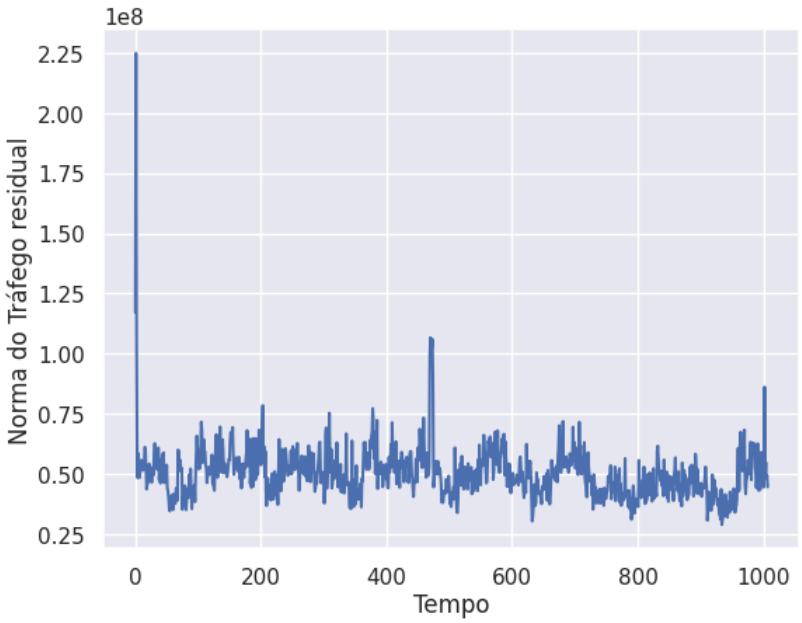

## Matrizes $X_r$ e $O$

```
r = 6

Ur = U[:, :r]
Sr = np.diag(s[:r])
Vr = V[:, :r]
Vrt = Vr.T

N = np.dot(Ur, np.dot(Sr, Vrt))
O = X - N
```

## Tráfego residual total

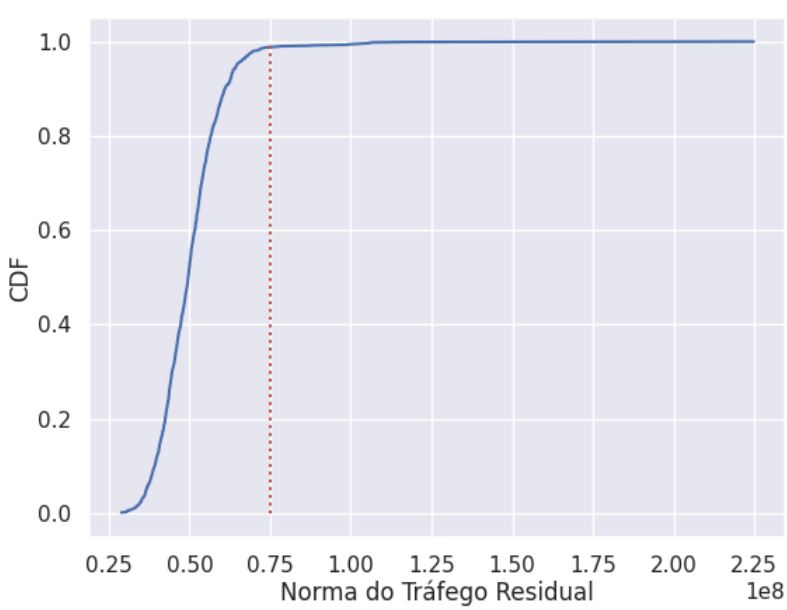
Veja que há um ruído + *outliers*

```
Onorm = np.linalg.norm(O, axis=1)
plt.plot(Onorm)
plt.xlabel('Tempo', labelpad = 2)
plt.ylabel('Norma do Tráfego residual', labelpad = 2);
```




## Determinando *outliers*

```
from statsmodels.distributions.empirical_distribution import ECDF

ecdf = ECDF(Onorm)

Xnorm = np.linalg.norm(X, axis=1)
plt.plot(ecdf.x, ecdf.y)
plt.plot([0.75 * 100000000, 0.75 * 100000000], [0, 1], 'r:')
plt.ylabel('CDF', labelpad = 2)
plt.xlabel('Norma do Tráfego Residual', labelpad = 2);
```



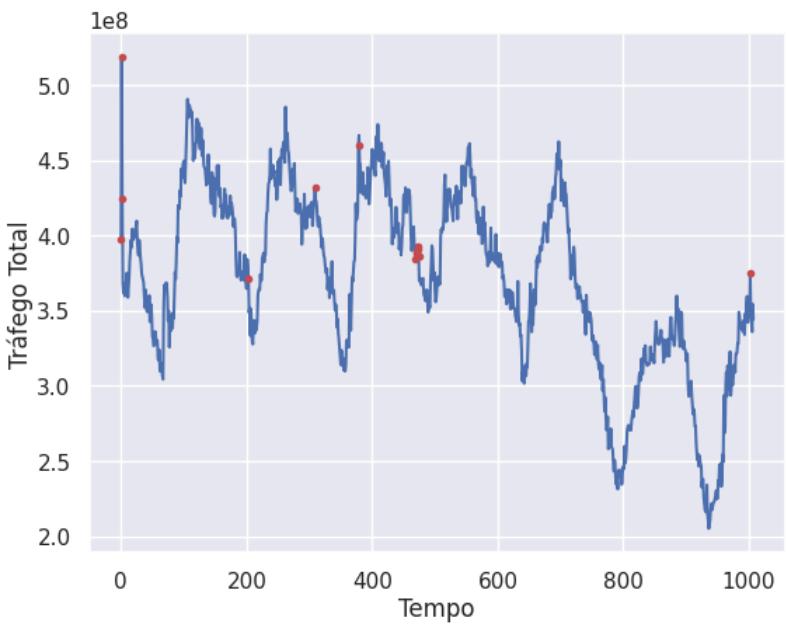

```
values = list(filter(lambda w: w > 0.75 * 10 ** 8, Onorm))
print(len(values))
```

```
13
```

## Visualizando períodos de tráfego anômalo

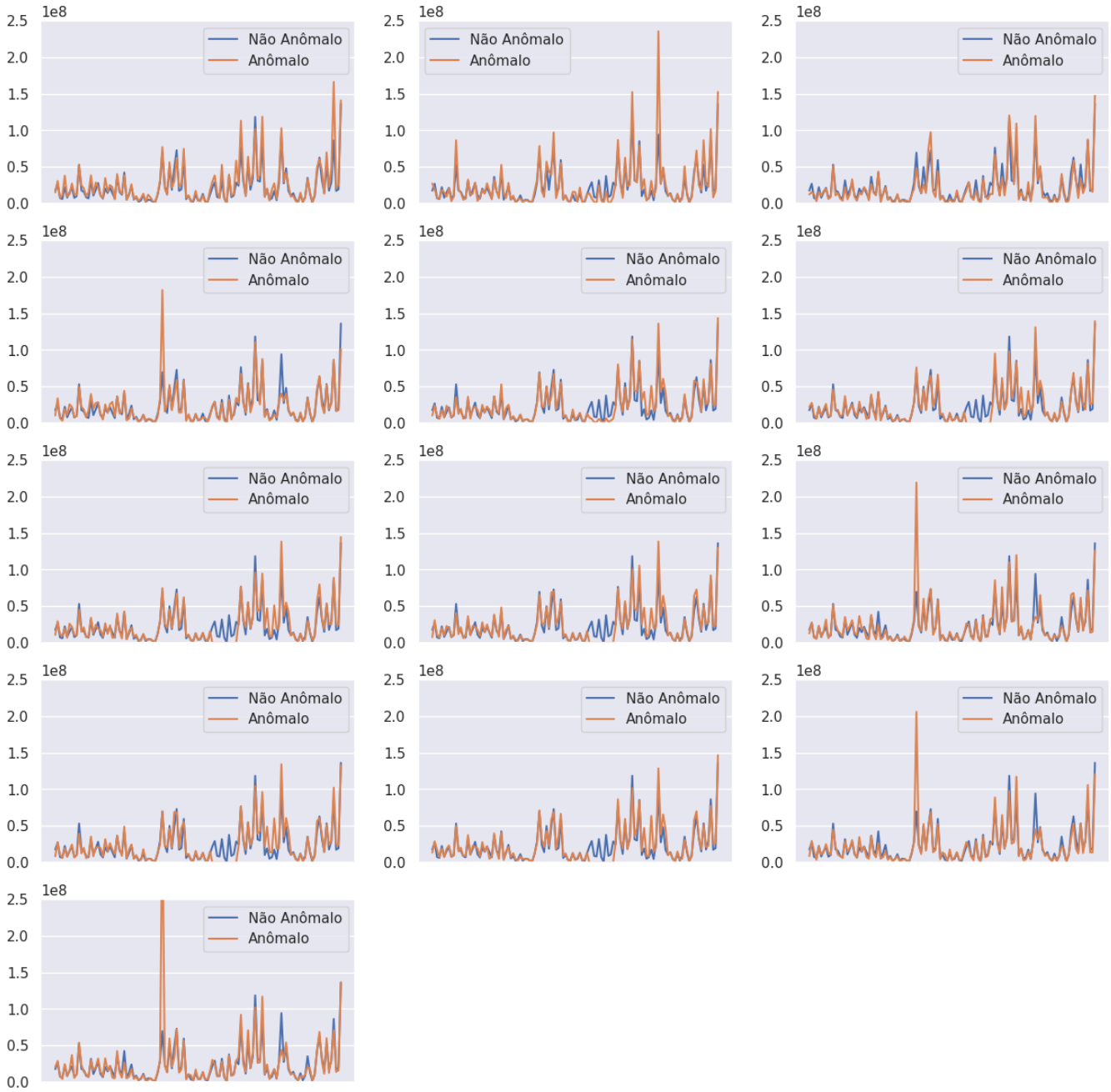
```
anomalies = np.argsort(Onorm)[-13:]
plt.plot(Xnorm)
plt.plot(anomalies, Xnorm[anomalies],'r.', lw = 0.5)
```

```
plt.ylabel('Tráfego Total', labelpad = 2)
plt.xlabel('Tempo', labelpad = 2);
```




## Do ponto de vista dos pares origem-destino

```
plt.figure(figsize=(15,15))
a = 13
anomalies = np.argsort(Onorm)[-a:]
normal = np.argsort(Onorm)[:-a]
for i in range(a):
  plt.subplot(5, 3, i + 1)
  X.iloc[normal, :].mean(axis = 0).plot(label = "Não Anômalo")
  X.iloc[anomalies[i], :].plot(label = 'Anômalo')
  plt.xticks([], [])
  plt.ylim(0, 2.5 * 10e7)
  plt.legend(loc = "best");
```




## Aprendizado Não Supervisionado

### Introdução

Aprendizado não supervisionado é uma área de aprendizado de máquina que tem o objetivo de **aprender** de dados não rotulados.

Hoje, daremos foco a tarefa de agrupamento (ou *clustering*).

Nesse contexto, o objeto de estudo será uma matriz $\mathbf{X} \in \mathbb{R}^{n \times d}$, onde:

- $n$ é o número de objetos; e
- $d$ é a dimensionalidade de cada objeto.

**Problema:** Dada a matriz $\mathbf{X}$, tem-se interesse dividir os $n$ elementos em $k$ grupos de forma que:

- Elementos do mesmo grupo sejam "similares"; e
- Elementos de grupos diferentes não sejam "similares".

```
import numpy as np
from sklearn import cluster, datasets
```

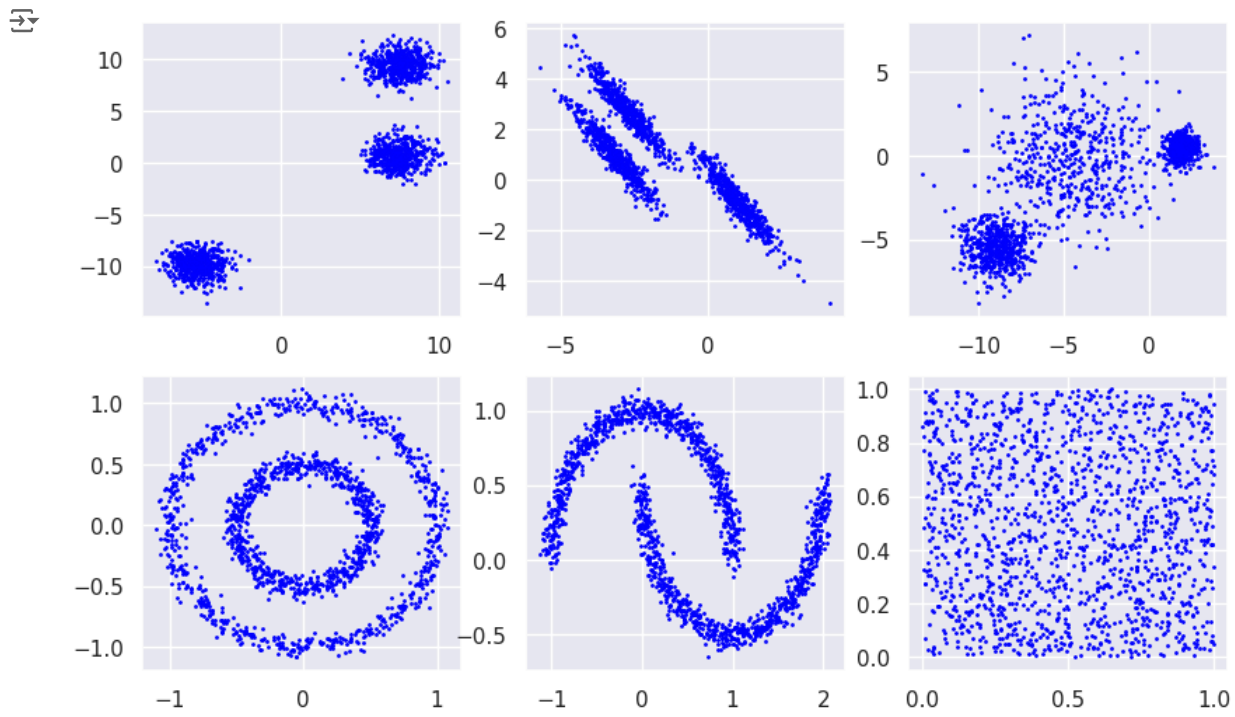
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

```
np.random.seed(0)
n_samples = 1500

# circles
noisy_circles, _ = datasets.make_circles(n_samples=n_samples, factor=.5,
                                         noise=.05)

# moons
noisy_moons, _ = datasets.make_moons(n_samples=n_samples, noise=.05)

# blobs
blobs, _ = datasets.make_blobs(n_samples=n_samples, random_state=8)

# no structure
no_structure = np.random.rand(n_samples, 2)

# elipses
random_state = 170
X, y = datasets.make_blobs(n_samples=n_samples, random_state=random_state)
transformation = [[0.6, -0.6], [-0.4, 0.8]]
X_aniso = np.dot(X, transformation)
aniso = X_aniso

# blobs with varied variances
varied, _ = datasets.make_blobs(n_samples=n_samples,
                                cluster_std=[1.0, 2.5, 0.5],
                                random_state=random_state)
```

```
def plot_figure_example(color):
  f, axarr = plt.subplots(2, 3, figsize=(10, 6))
  axarr[0, 0].scatter(blobs[:, 0], blobs[:, 1], s = 1, c = color[0])
  axarr[0, 1].scatter(aniso[:, 0], aniso[:, 1], s = 1, c = color[1])
  axarr[0, 2].scatter(varied[:, 0], varied[:, 1], s = 1, c = color[2])
  axarr[1, 0].scatter(noisy_circles[:, 0], noisy_circles[:, 1], s = 1,
                      c = color[3])
  axarr[1, 1].scatter(noisy_moons[:, 0], noisy_moons[:, 1], s = 1, c = color[4])
  axarr[1, 2].scatter(no_structure[:, 0], no_structure[:, 1], s = 1,
                      c = color[5])
```

## Quantos grupos?

```
plot_figure_example([["blue"] for _ in range(6)])
```



**Não** há um único algoritmo que seja capaz de produzir bons resultados em todas as situações. Por isso, precisamos estudar e entender bem soluções de diferentes paradigmas!

## K-means

Dada uma matriz $\mathbf{X} \in \mathbb{R}^{n \times d}$, cujas linhas são denotadas por $\mathbf{x}_1, \ldots, \mathbf{x}_n$, e um inteiro $k$, o objetivo é encontrar $k$ pontos, $\mathbf{c}_1, \ldots, \mathbf{c}_k$, de forma que a seguinte quantidade seja minimizada:

$$\sum_{i=1}^{n} \min_{1 \leq j \leq k} ||\mathbf{x}_i - \mathbf{c}_j||_2^2$$

O que isso significa no exemplo abaixo?

```
_ = plt.scatter(blobs[:, 0], blobs[:, 1], s = 2)
```


### Solução

Esse problema é difícil de ser resolvido de forma exata. Na verdade, não é conhecida (e não se sabe se existe) uma forma eficiente de resolvê-lo.

Na prática, uma heurística simples funciona muito bem. Tal heurística é sumarizada a seguir:

1. "Chute" de alguma forma os $k$ centroides
2. Repita os seguintes passos até que haja convergência
    - Associe cada ponto com o centroide mais próximo
    - Modifique o centroide de cada grupo como sendo a média de cada ponto

## Com a biblioteca *Scikit-learn*

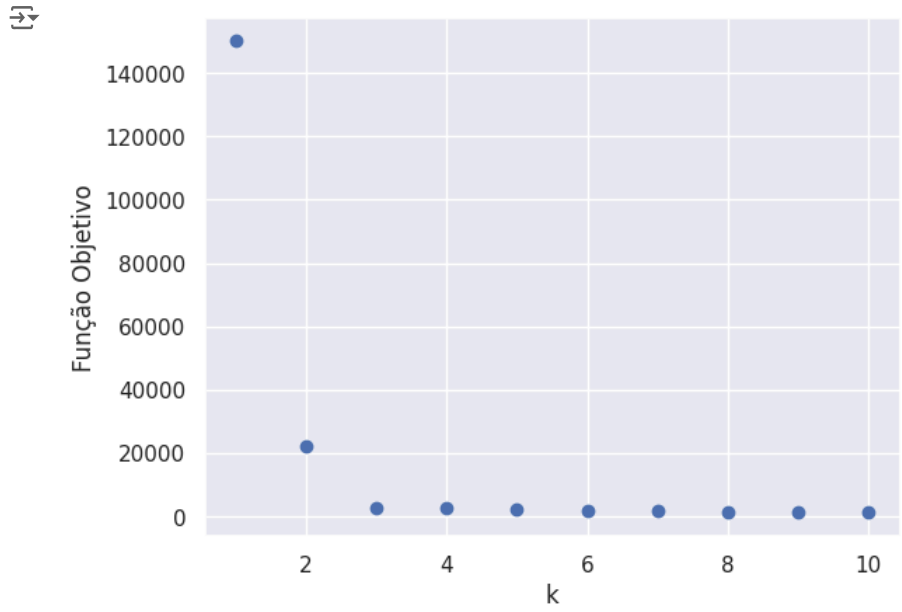
```
from sklearn.cluster import KMeans

# Fornecendo os parâmetros do modelo
kmeans = KMeans(n_clusters = 3)

# ajustando o modelo
kmeans.fit(blobs)

# obtendo o identificador de grupo de cada ponto
y_kmeans = kmeans.predict(blobs)

# cada ponto é mapeado para um número, correspondente ao cluster
# ao qual foi associado
y_kmeans
```

```
array([0, 0, 0, ..., 0, 1, 1], dtype=int32)
```

```
# Com esses números, podemos associar cada grupo a uma cor
centers = kmeans.cluster_centers_
plt.scatter(blobs[:, 0], blobs[:, 1], c=y_kmeans, s=10, cmap='viridis')
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=30);
```


## Como escolher $k$?

```
inertia = []
for i in range(1, 11):
  km = cluster.KMeans(n_clusters = i)
  km.fit(blobs)
  inertia.append(km.inertia_)

plt.scatter(range(1, 11), inertia)
plt.ylabel("Função Objetivo")
plt.xlabel("k");
```

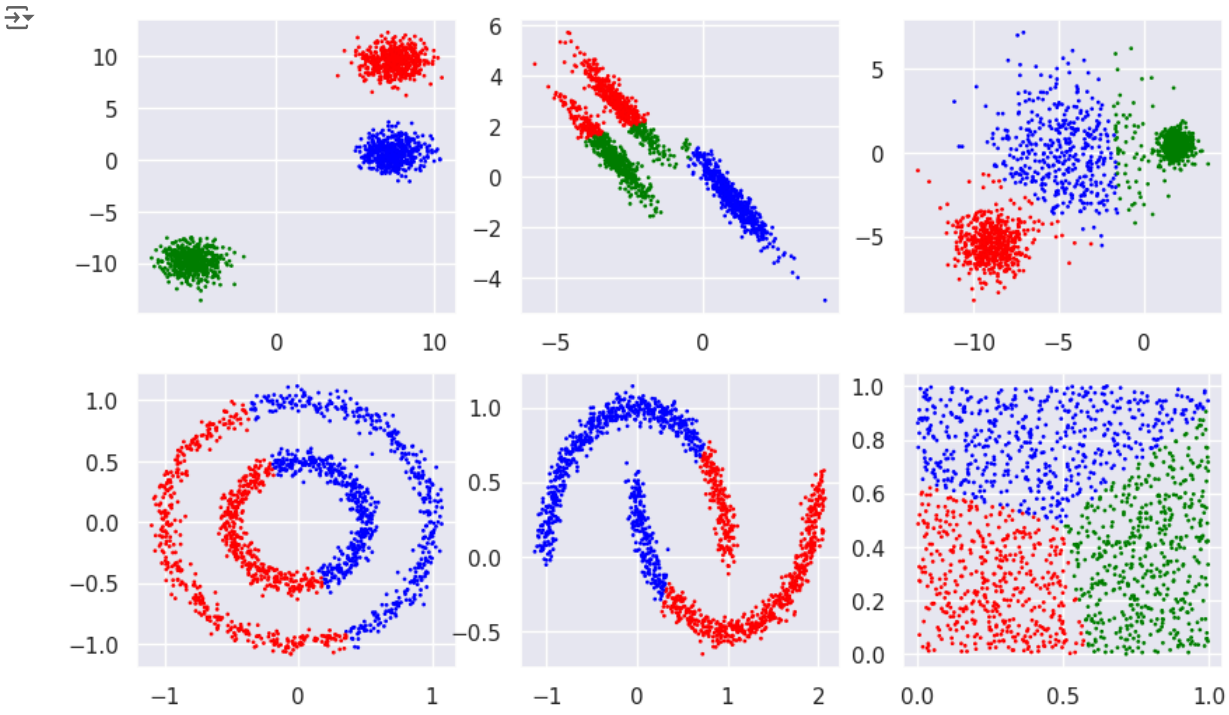



## Exemplos

```
def exec_model(dataset, model, params):
  labels = []
  if model == cluster.DBSCAN:
    for idx, param in enumerate(params):
      km = model(eps = param[0], min_samples = param[1])
```

```
      km.fit(dataset[idx])
      labels.append(km.labels_)
  elif model == cluster.SpectralClustering:
    for idx, param in enumerate(params):
      km = model(param, affinity='nearest_neighbors')
      km.fit(dataset[idx])
      labels.append(km.labels_)
  else:
    for idx, param in enumerate(params):
      km = model(param)
      km.fit(dataset[idx])
      labels.append(km.labels_)
  color = "blue red green yellow black purple".split()
  c = []
  for label in labels:
    c.append([color[w] for w in label])
  return c
```

```
dataset = [blobs, aniso, varied, noisy_circles, noisy_moons, no_structure]
n_groups = [3, 3, 3, 2, 2, 3]
colors = exec_model(dataset, cluster.KMeans, n_groups)
plot_figure_example(colors)
```



## Agrupamento Aglomerativo

### Introdução

O objetivo do agrupamento aglomerativo é criar uma representação hierárquica dos dados.

Dada uma matriz $\mathbf{X} \in \mathbb{R}^{n \times d}$, cujas linhas são denotadas por $\mathbf{x}_1, \ldots, \mathbf{x}_n$, e um inteiro $k$, o agrupamento aglomerativo funciona da seguinte forma:

1. Na inicialização, cada $\mathbf{x}_i$ se torna um grupo $\{\mathbf{x}_i\}$
2. O seguinte laço é repetido até que restem apenas $k$ grupos
    - Encontre os dois grupos mais próximos e os aglomere em apenas um

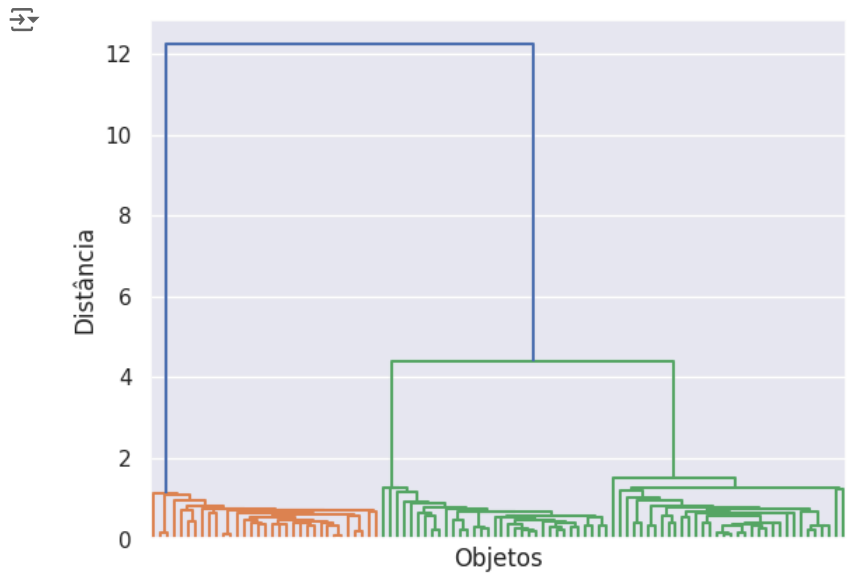
Há várias formas diferentes de calcular a distância entre dois grupos diferentes, sendo que cada forma resulta em uma variação distinta do algoritmo.

### Visualização Hierárquica

```
from scipy.cluster import hierarchy
import matplotlib.pyplot as plt
import itertools
```

```
n_samples = 100
np.random.seed(0)
X, _ = datasets.make_blobs(n_samples=n_samples, random_state=8)
plt.scatter(X[:, 0], X[:, 1], s=10);
```


```
Z = []
n, d = X.shape
for x, y in itertools.combinations(X, 2):
    Z.append(np.linalg.norm(x - y))
Z = hierarchy.linkage(Z)
hierarchy.dendrogram(Z)
plt.xticks([], [])
plt.ylabel('Distância')
plt.xlabel('Objetos');
```




## Com a biblioteca *Scikit-learn*

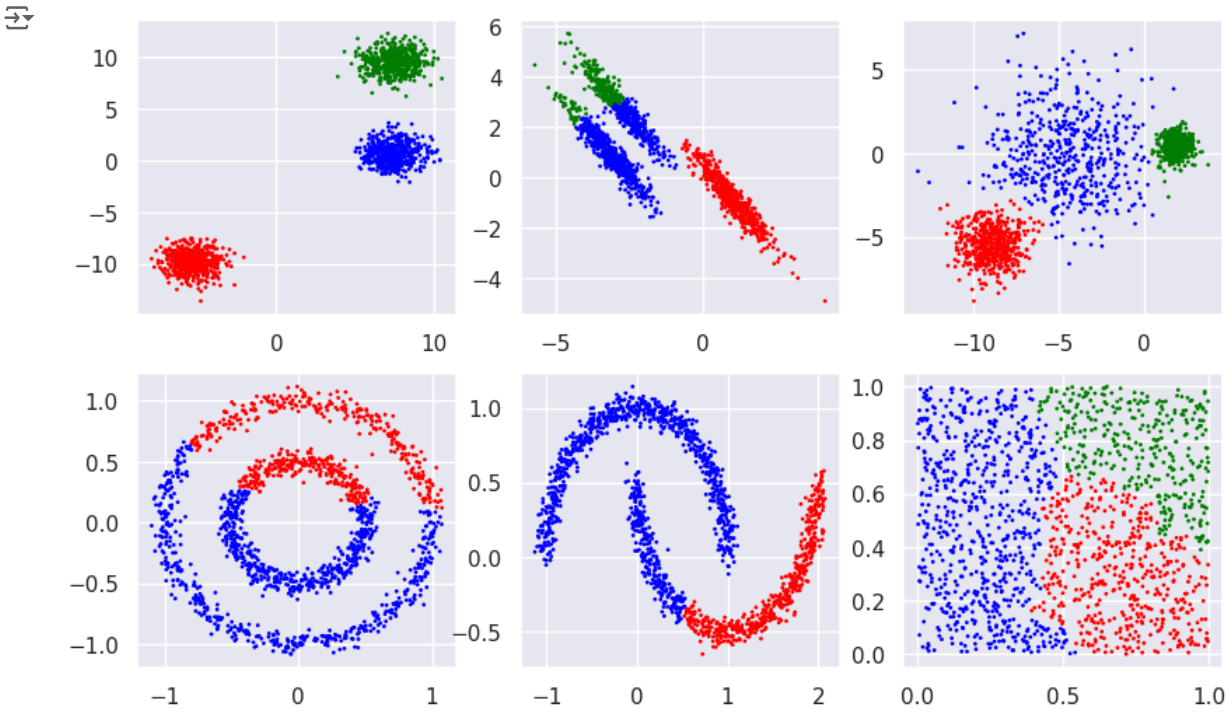
```
agg = cluster.AgglomerativeClustering(n_clusters = 3)
agg.fit(blobs);
```

```
agg.labels_
```

```
array([2, 2, 2, ..., 2, 1, 1])
```

## Exemplos

```
colors = exec_model(dataset, cluster.AgglomerativeClustering, n_groups)
plot_figure_example(colors)
```



## Agrupamento Baseado em Densidade

## Introdução

Tem o objetivo de encontrar grupos de alta densidade (pontos por região) que sejam isolados uns dos outros.

Dada uma matriz $\mathbf{X} \in \mathbb{R}^{n \times d}$, cujas linhas são denotadas por $\mathbf{x}_1, \ldots, \mathbf{x}_n$, um número real $\epsilon$ e um inteiro $\eta$, o agrupamento baseado em densidade (DBSCAN) funciona da seguinte forma:

1. Para cada ponto, encontre os pontos em uma $\epsilon$-vizinhança. Os pontos que tiverem mais de $\eta$ vizinhos são denominados *core points*;
2. Encontre conjuntos maximais de *core points* tais que cada *core point* esteja em uma $\epsilon$-vizinhaça de ao menos um outro ponto do conjunto;
3. Associe a cada ponto que não é um *core point* ao grupo mais próximo, se a distância for inferior a $\epsilon$;
4. Todos os outros pontos são rotulados como ruído.

**Não há definição do número de grupos! O resultado é baseado nos valores de $\epsilon$ e $\eta$.**

## Com a *Scikit-learn*

```
dbs = cluster.DBSCAN(eps = 0.8, min_samples = 5)
dbs.fit(blobs);
```

```
dbs.labels_
```

```
array([0, 0, 0, ..., 0, 2, 2])
```

```
dbs.core_sample_indices_
```

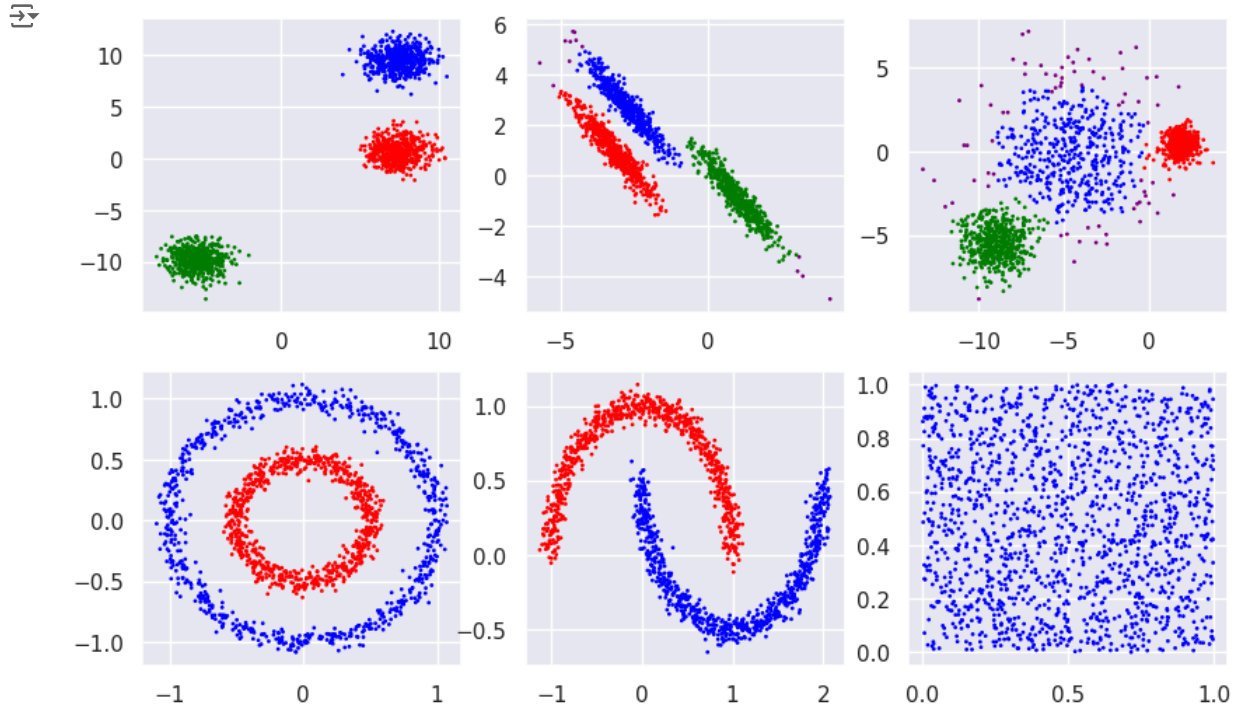
```
array([   0,    1,    2, ..., 1497, 1498, 1499])
```

```
noise = np.where(dbs.labels_ == -1)[0]
notNoise = np.where(dbs.labels_ != -1)[0]
```

```
plt.scatter(blobs[notNoise, 0], blobs[notNoise, 1], c=dbs.labels_[notNoise], s=10, cmap='viridis')
plt.scatter(blobs[noise, 0], blobs[noise, 1], color = 'red', marker = '+', s=10);
```

## Exemplos

```
params_dbs = [[1.5, 5], [0.35, 5], [0.8, 10], [0.23, 50], [0.2, 50], [0.3, 5]]
colors = exec_model(dataset, cluster.DBSCAN, params_dbs)
plot_figure_example(colors)
```



## Agrupamento Espectral

## Introdução

Uma abordagem diferente para realizar agrupamento!

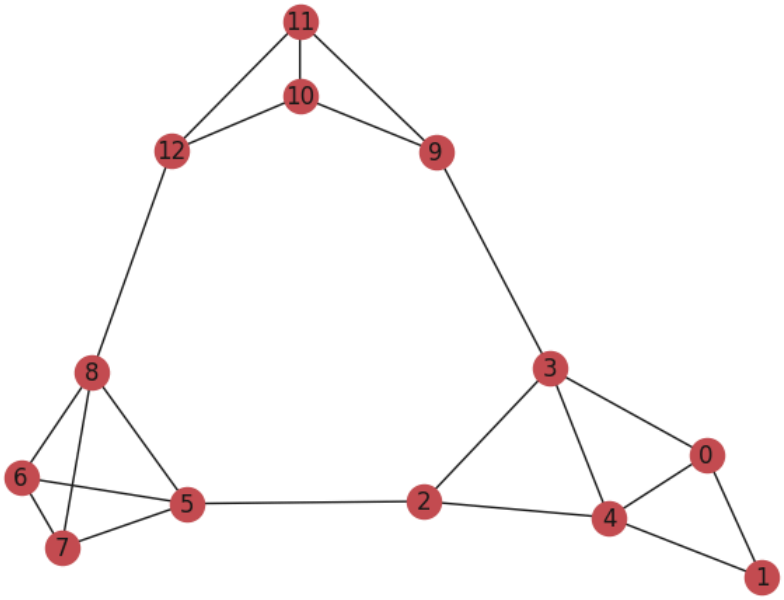
De onde vem o nome espectral?

Vamos motivar o método por meio de um exemplo simples.

Seja $G(V, E)$ um grafo simples e não direcionado, como o do exemplo abaixo:

```
import networkx as nx
```

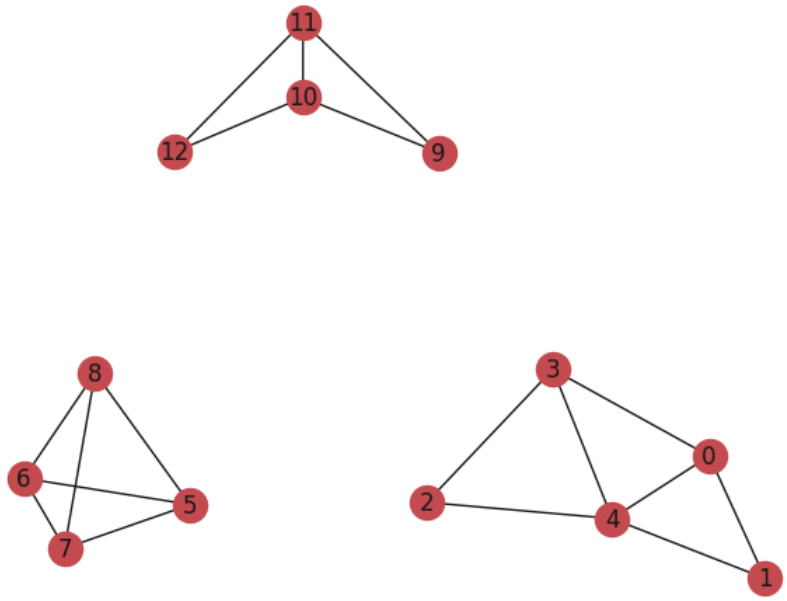
```
nodes = list(range(13))
edges = [(0, 1), (0, 3), (0, 4), (1, 4), (2, 3), (2, 4), (2, 5), (3, 4),
         (3, 9), (5, 6), (5, 7), (5, 8), (6, 7), (6, 8), (7, 8), (8, 12),
         (9, 10), (9, 11), (10, 11), (10, 12), (11, 12)]
labels = dict([(x, str(x)) for x in nodes])
G = nx.Graph()
G.add_nodes_from(nodes)
G.add_edges_from(edges)
pos = nx.spring_layout(G)
nx.draw(G, pos = pos, labels = labels, node_color='r')
plt.draw()
```




Repare que este grafo possui uma estrutura de comunidades. Veja no entanto que o grafo é conexo.

Considere agora a versão não conexa desse grafo.

```
nodes = list(range(13))
edges = [(0, 1), (0, 3), (0, 4), (1, 4), (2, 3), (2, 4), (3, 4),
         (5, 6), (5, 7), (5, 8), (6, 7), (6, 8), (7, 8), (9, 10), (9, 11),
         (10, 11), (10, 12), (11, 12)]
labels = dict([(x, str(x)) for x in nodes])
H = nx.Graph()
H.add_nodes_from(nodes)
H.add_edges_from(edges)
nx.draw(H, pos = pos, labels = labels, node_color = 'r')
plt.draw()
```




Matriz de adjacências

```
A = nx.to_numpy_array(H)
A
```

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

Matriz de graus

```
d = A.sum(axis = 1)
D = np.diag(np.squeeze(np.asarray(d)))
D
```

```
array([[3., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 2., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 2., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 3., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 4., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 3., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 3., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 3., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 3., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 2., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 3., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 3.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 2.]])
```

Matriz Laplaciana

```
L = D - A
L
#np.dot(L, np.ones(len(L)) / (len(L) ** (1/2)))
```

```
array([[ 3., -1.,  0., -1., -1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [-1.,  2.,  0.,  0., -1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  2., -1., -1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [-1.,  0., -1.,  3., -1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
```

```
[-1., -1., -1., -1.,  4.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
[ 0.,  0.,  0.,  0.,  0.,  3., -1., -1., -1.,  0.,  0.,  0.],
[ 0.,  0.,  0.,  0.,  0., -1.,  3., -1., -1.,  0.,  0.,  0.],
[ 0.,  0.,  0.,  0.,  0., -1., -1.,  3., -1.,  0.,  0.,  0.],
[ 0.,  0.,  0.,  0.,  0., -1., -1., -1.,  3.,  0.,  0.,  0.],
[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  2., -1., -1.],
[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., -1.,  3., -1.],
[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., -1., -1.,  3.],
[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., -1., -1.,  2.]])
```

Veja que a matriz Laplaciana do grafo possui uma estrutura bloco-diagonal. Além disso, considere os três vetores definidos a seguir.

```
u = np.concatenate([np.ones(5), np.zeros(8)])
v = np.concatenate([np.zeros(5), np.ones(4), np.zeros(4)])
w = np.concatenate([np.zeros(9), np.ones(4)])
print(u)
print(v)
print(w)
```

```
[1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1.]
```

```
np.dot(L, u)
```

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

```
np.dot(L, v)
```

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

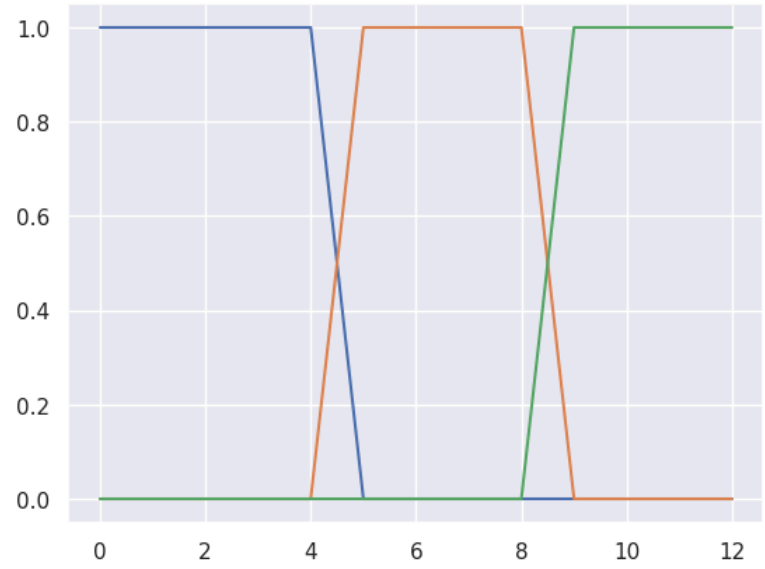
```
np.dot(L, w)
```

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

Esses três vetores são autovetores de $L$ associados ao autovalor zero. Veja também, que eles indicam a estrutura de cada comunidade do grafo.

**Há uma relação muito forte entre a estrutura de comunidades de um grafo e o espectro da matriz Laplaciana!**

```
_ = plt.plot(u)
_ = plt.plot(v)
_ = plt.plot(w)
```




## Com a biblioteca *Scikit-learn*

Na prática, queremos aplicar o método a um conjunto de dados, não necessariamente a um grafo.

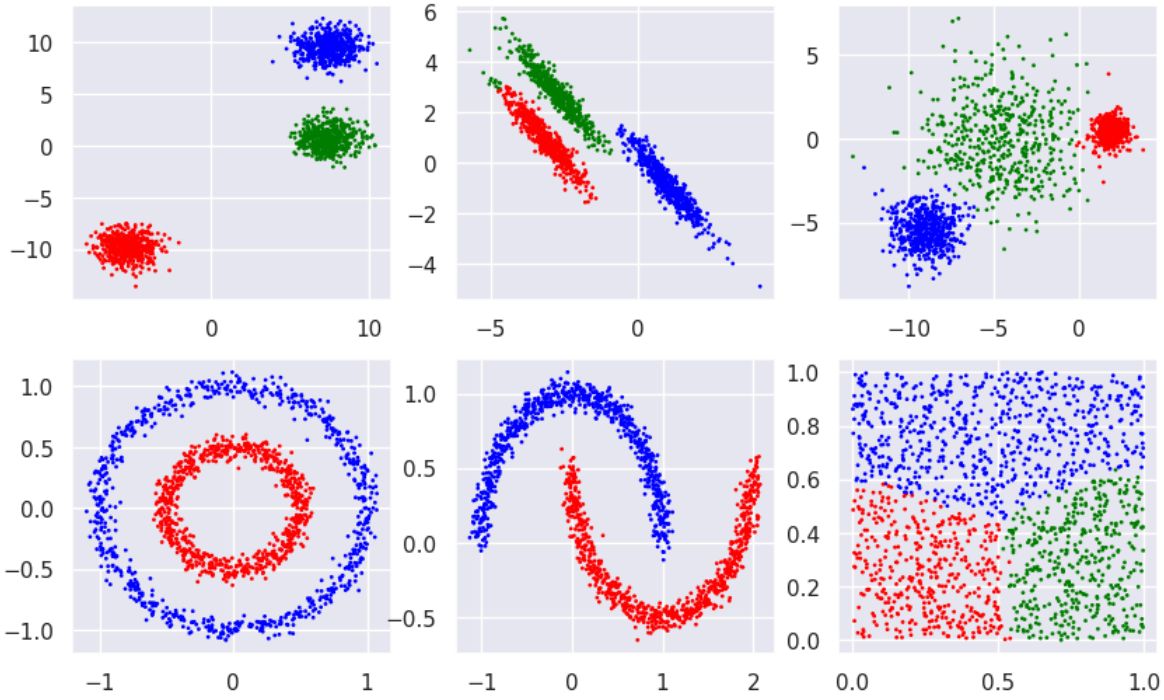
```
esp = cluster.SpectralClustering(n_clusters=3, affinity = 'nearest_neighbors')
_ = esp.fit(noisy_circles)
```

```
esp.labels_
```

```
array([0, 1, 2, ..., 2, 1, 1], dtype=int32)
```

```
colors = exec_model(dataset, cluster.SpectralClustering, n_groups)
plot_figure_example(colors)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/manifold/_spectral_embedding.py:329: UserWarning: Graph is not ful
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/manifold/_spectral_embedding.py:329: UserWarning: Graph is not ful
  warnings.warn(
```



## Exemplo: Detecção de Comunidades em Redes

Como exemplo de aplicação em Redes de Computadores, vamos considerar o exemplo simples de detectar comunidades em uma topologia.

```
import matplotlib.pyplot as plt
import seaborn as sns
import networkx as nx
import geopy.distance
import numpy as np

sns.set()
```

### Rede da empresa Cogent, 2010

O exemplo se refere à topologia de rede (ao nível de IP) que conecta vários pontos de presença na América do Norte e Europa.

Além da topologia, para cada ponto, sabe-se suas coordenadas geográficas.

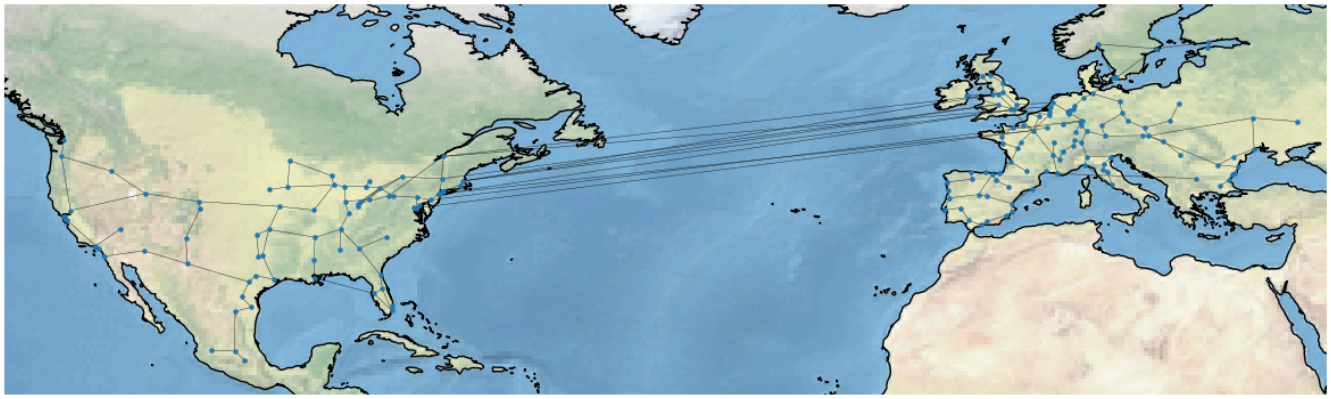
```
h = nx.read_graphml('datasets-minicurso/cogentco.graphml')
for node in list(h.nodes):
    pos[node] = (h.nodes[node]['Longitude'], h.nodes[node]['Latitude'])
```

O peso de cada aresta do grafo é inversamente proporcional à distância entre dois nós.

```
for x, y in h.edges:
  locx = (h.nodes[x]['Latitude'], h.nodes[x]['Longitude'])
```

```
  locy = (h.nodes[y]['Latitude'], h.nodes[y]['Longitude'])
  h.edges[(x, y)]['weight'] = np.exp(-geopy.distance.distance(locx, locy).km / 1000)
```

```
import cartopy.crs as ccrs

fig = plt.figure(figsize = (15, 5))
ax = plt.axes(projection=ccrs.PlateCarree())
ax.stock_img()
ax.set_extent([-130, 40, 15, 65], ccrs.PlateCarree())
ax.stock_img()
ax.coastlines()
nx.draw(h, pos, node_size = 5, width = 0.3)
```



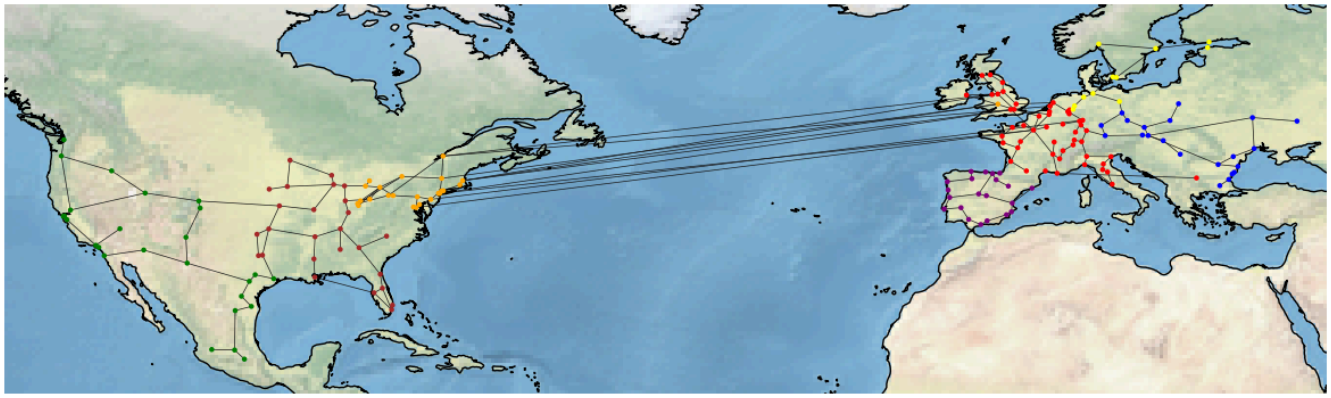
## Utilizando Agrupamento Espectral

```
from sklearn.cluster import SpectralClustering
A = nx.to_numpy_array(h)
sc = SpectralClustering(n_clusters=7, affinity='precomputed')
_ = sc.fit(A)
```

## Resultado do agrupamento

```
colors = ['red', 'green', 'blue', 'brown', 'purple', 'yellow', 'orange', 'cyan']
c = [colors[w] for w in sc.labels_]
```

```
plt.figure(figsize = (15, 5))
ax = plt.axes(projection=ccrs.PlateCarree())
ax.stock_img()
ax.set_extent([-130, 40, 15, 65], ccrs.PlateCarree())
ax.stock_img()
ax.coastlines()
nx.draw(h, pos, node_size = 5, width = 0.5, node_color = c)
```



# Aprendizado Supervisionado

## Introdução

Aprendizado supervisionado é uma área de aprendizado de máquina que tem o objetivo de **aprender** a partir de dados "rotulados".

Há duas tarefas principais:

- Regressão
- Classificação

Nesse contexto, o objeto de estudo serão uma matriz $\mathbf{X} \in \mathbb{R}^{n \times d}$ e um vetor $\mathbf{y}$, onde:

- $n$ é o número de objetos; e
- $d$ é a dimensionalidade de cada objeto.
- $y_i$ é o rótulo (ou valor) associado a $\mathbf{x}_i$

**O foco desse minicurso será em Classificação**

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn import datasets
import seaborn as sns

sns.set()
```

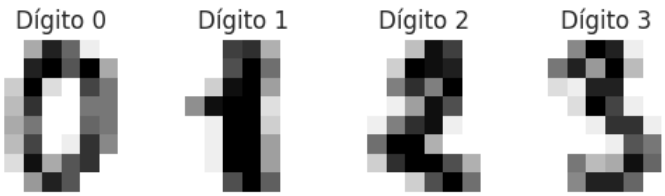
## Reconhecendo dígitos

```
digits = datasets.load_digits(return_X_y = False)
df = pd.DataFrame(digits.data)
df.head()
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.0 | 0.0 | 5.0 | 13.0 | 9.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 6.0 | 13.0 | 10.0 | 0.0 | 0.0 | 0.0 |
| **1** | 0.0 | 0.0 | 0.0 | 12.0 | 13.0 | 5.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 11.0 | 16.0 | 10.0 | 0.0 | 0.0 |
| **2** | 0.0 | 0.0 | 0.0 | 4.0 | 15.0 | 12.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 5.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.0 | 11.0 | 16.0 | 9.0 | 0.0 |
| **3** | 0.0 | 0.0 | 7.0 | 15.0 | 13.0 | 1.0 | 0.0 | 0.0 | 0.0 | 8.0 | ... | 9.0 | 0.0 | 0.0 | 0.0 | 7.0 | 13.0 | 13.0 | 9.0 | 0.0 | 0.0 |
| **4** | 0.0 | 0.0 | 0.0 | 1.0 | 11.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 16.0 | 4.0 | 0.0 | 0.0 |

5 rows × 64 columns

```
images_and_labels = list(zip(digits.images, digits.target))
for index, (image, label) in enumerate(images_and_labels[:4]):
    plt.subplot(2, 4, index + 1)
    plt.axis('off')
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title('Dígito %i' % label)
```




**Perguntas:**

- Dada uma imagem de um número, qual é o dígito equivalente?
- Qual o custo de uma predição errada?
- Há outros fatores relevantes?

## Classificação

Dados $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, a tarefa de classificação consiste de aprender um modelo $f$ que aprenda dos vetores de características os rótulos.

## Qualquer $f$ serve?

Há dois requisitos principais:

1. Deseja-se que $(\mathbf{x}, y) \in D$ se, e somente se, $f(\mathbf{x}) = y$ (*o tanto quanto possível)*
2. A função aprendida deve ser **generalizável**. Se um novo $(\mathbf{x}, y) \notin \mathbf{D}$ for observado, deseja-se que $f(\mathbf{x}) = y$.

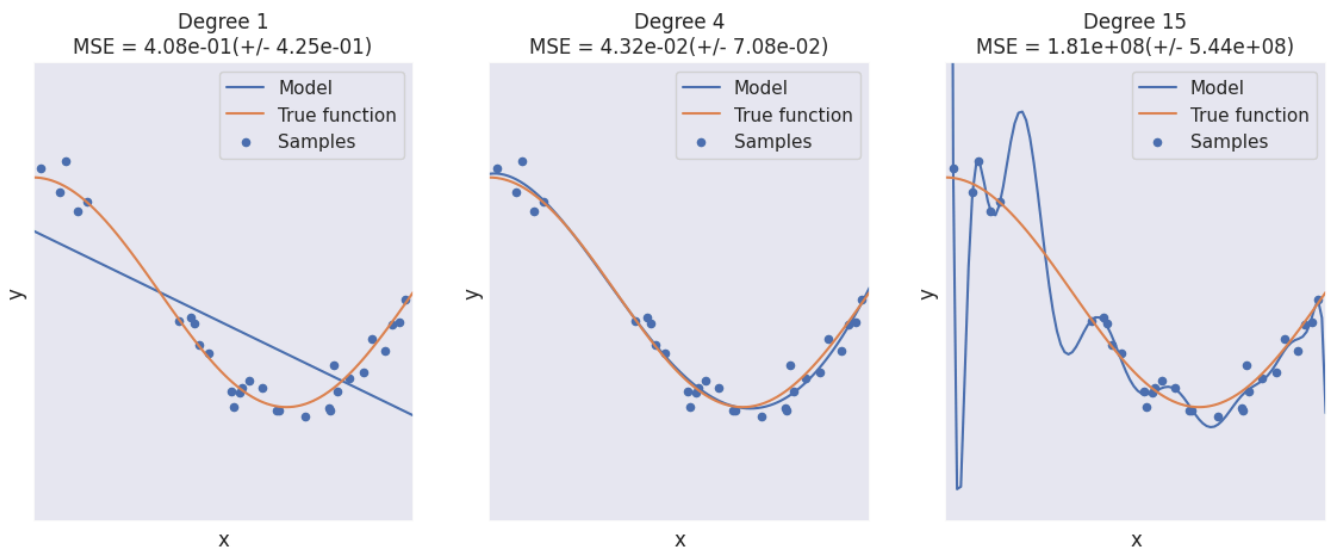
**Satisfazer apenas o requisito 1 é fácil. Satisfazer 1 e 2 simultaneamente é difícil!**

## Overfitting

Ocorre quando o modelo explica tão bem o conjunto de dados usado durante do aprendizado, que é incapaz de generalizar para dados não vistos.

Considere o seguinte exemplo para a tarefa de regressão (da biblioteca scikit-learn):

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score


def true_fun(X):
    return np.cos(1.5 * np.pi * X)

np.random.seed(0)

n_samples = 30
degrees = [1, 4, 15]

X = np.sort(np.random.rand(n_samples))
y = true_fun(X) + np.random.randn(n_samples) * 0.1

plt.figure(figsize=(14, 5))
for i in range(len(degrees)):
    ax = plt.subplot(1, len(degrees), i + 1)
    plt.setp(ax, xticks=(), yticks=())

    polynomial_features = PolynomialFeatures(degree=degrees[i],
                                             include_bias=False)
    linear_regression = LinearRegression()
    pipeline = Pipeline([("polynomial_features", polynomial_features),
                         ("linear_regression", linear_regression)])
    pipeline.fit(X[:, np.newaxis], y)

    # Evaluate the models using crossvalidation
    scores = cross_val_score(pipeline, X[:, np.newaxis], y,
                             scoring="neg_mean_squared_error", cv=10)

    X_test = np.linspace(0, 1, 100)
    plt.plot(X_test, pipeline.predict(X_test[:, np.newaxis]), label="Model")
    plt.plot(X_test, true_fun(X_test), label="True function")
    plt.scatter(X, y, edgecolor='b', s=20, label="Samples")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.xlim((0, 1))
    plt.ylim((-2, 2))
    plt.legend(loc="best")
    plt.title("Degree {}\nMSE = {:.2e}(+/- {:.2e})".format(
        degrees[i], -scores.mean(), scores.std()))
plt.show()
```

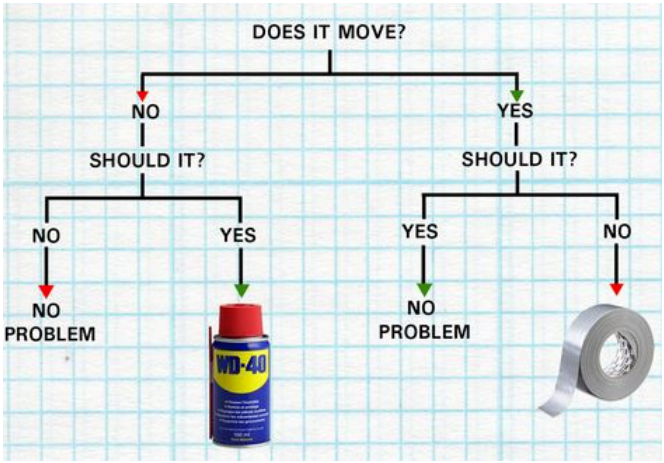



## Árvores de Decisão

### Introdução

O objetivo é formar uma "árvore" de regras a partir dos dados. Cada nó da árvore é um "teste" com base em algum atributo e cada aresta equivale à saída de um teste.

### Árvore de decisão para objetos defeituosos




**O processo de criação de uma árvore de decisão é complexo!**

### Com a biblioteca *Scikit-learn*

```
import graphviz
import pydotplus
from sklearn import tree
```

```
X, y = datasets.load_digits(return_X_y = True)
```

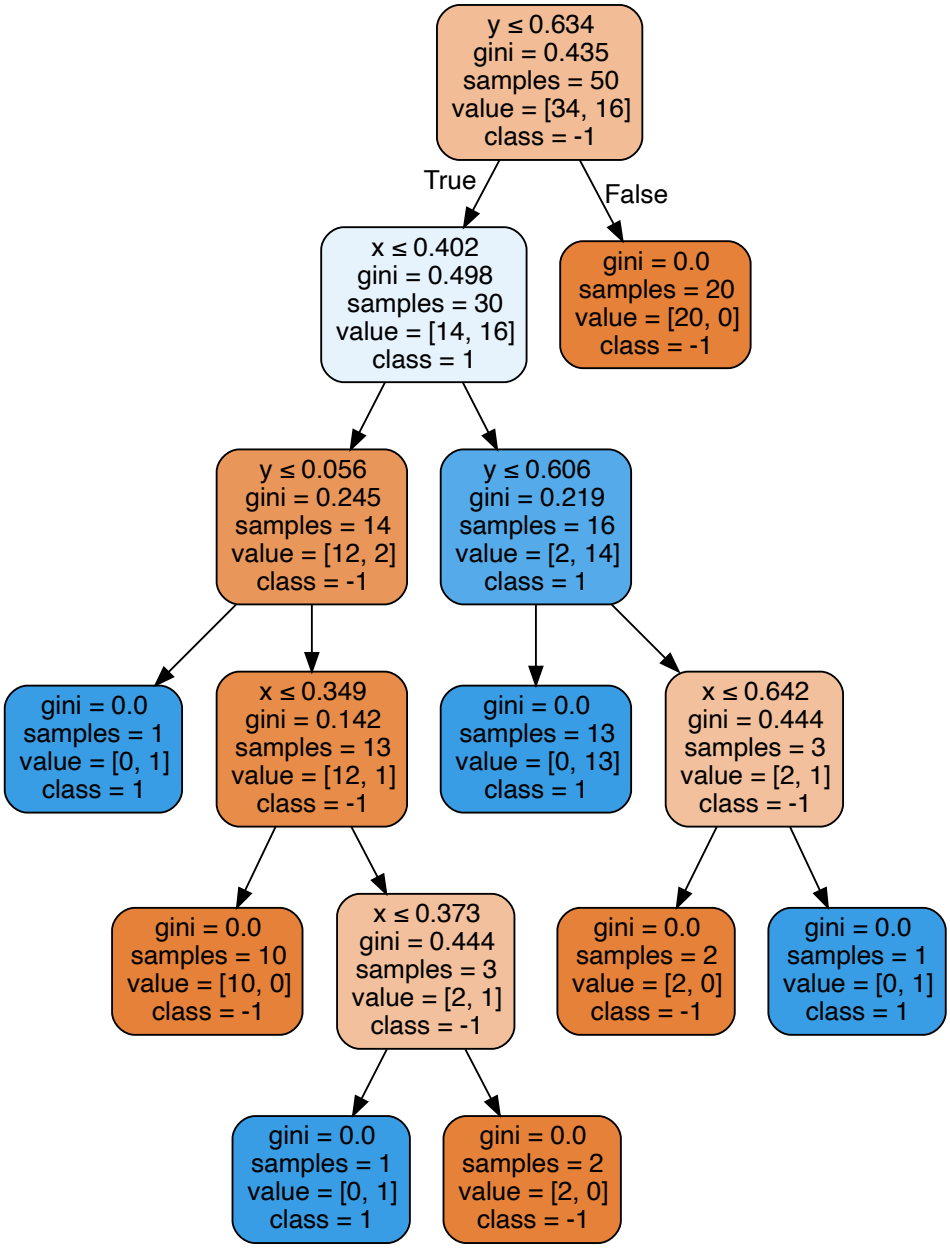
```
dt = tree.DecisionTreeClassifier()
dt.fit(X, y);
```

```
# apenas para ilustrar o funcionamento
dt.predict(X)
```

```
array([0, 1, 2, ..., 8, 9, 8])
```

## É possível visualizar a árvore

```
dt_example = pd.read_csv("datasets-minicurso/dt_example.csv")
X_dt = dt_example.iloc[:, :2]
y_dt = dt_example.iloc[:, 2:]

clf = tree.DecisionTreeClassifier(random_state = 0)
clf = clf.fit(X_dt, y_dt)

dot_data = tree.export_graphviz(clf, out_file=None,
                      feature_names=["x", "y"],
                      class_names=["-1", "1"],
                      filled=True, rounded=True,
                      special_characters=True)

graph = graphviz.Source(dot_data)
graph.render("dt-vis",view = True)
graph
```




## Florestas Aleatórias

## Introdução

- Suponha que você tenha $n$ classificadores independentes
- Suponha que cada um deles tenha uma chance $p > 0.5$ de estar correto.

**Podemos acertar com probabilidade maior que $p$?**

1. Dado um $\mathbf{x}$ a ser classificado, utilize os $n$ classificadores e para cada um obtenha uma previsão da classe de $\mathbf{x}$
2. Retorne a classe que receba a maioria dos votos!

**Qual a chance desse novo classificador estar correto?**

Pode-se mostrar que quando $n$ cresce, a chance de acertar a classe de $\mathbf{x}$ também cresce.

**A ideia de um classificador baseado em Floresta Aleatória é criar uma grande quantidade de árvores de decisão que sejam independentes (ou ao menos com baixa correlação)**

- Há várias maneiras de "tentar" criar árvores independentes umas das outras. Por exemplo, selecionando atributos para a árvore de decisão de forma aleatória.
- O ganho, no entanto, não é o mesmo que a teoria mostra! Por quê?

## Com a biblioteca *Scikit-learn*

```
from sklearn.ensemble import RandomForestClassifier

X, y = datasets.load_digits(return_X_y = True)

rf = RandomForestClassifier(n_estimators = 100)
rf.fit(X, y);
```

```
# apenas para ilustrar o funcionamento
rf.predict(X)
```

```
array([0, 1, 2, ..., 8, 9, 8])
```

# SVM – Support Vector Machine

## Introdução

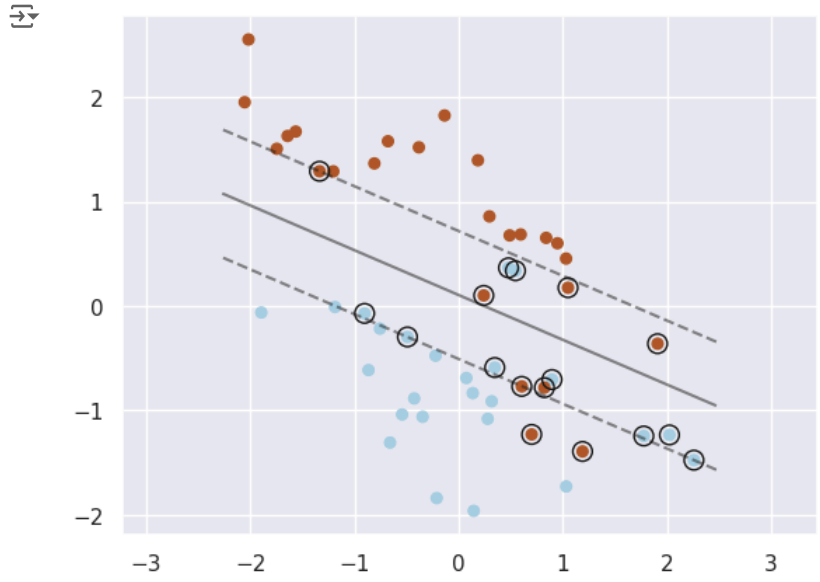
Tem o objetivo de encontrar um hiperplano que separe os dados da melhor forma possível.

```
from sklearn.datasets import make_classification
from sklearn import svm
import matplotlib.pyplot as plt
```

```
X, y = make_classification(n_samples = 50, random_state = 7, n_features = 2, n_redundant = 0, class_sep = 0.65)
plt.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)
plt.axis('equal');
```

Mas não queremos qualquer hiperplano. Queremos um de margem máxima.

```
clf = svm.SVC(kernel='linear', C=10)
clf.fit(X, y)

plt.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)
plt.axis('equal')

ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = clf.decision_function(xy).reshape(XX.shape)

ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
           linestyles=['--', '-', '--'])
_ = ax.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=100,
           linewidth=1, facecolors='none', edgecolors='k')
```



## Problema de otimização

Dados vetores de treinamento $\mathbf{x}_i \in \mathbb{R}^d$ em duas classes $y_i \in \{-1, 1\}$. Tem-se interesse em resolver:

$$\min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i$$

$$\text{Sujeito a } y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i,$$

$$\zeta_i \geq 0, i = 1, \ldots, n$$

O parâmetro $C$ define o tamanho da margem.

## Com a biblioteca *Scikit-learn*

```
clf = svm.SVC(kernel = 'linear', C = 1)
_ = clf.fit(X, y)
```

```
# apenas para ilustrar o funcionamento
clf.predict(X)
```

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

## Avaliação, Treino e Teste

Perguntas interessantes e relacionadas:

1. Como dizer se um classificador é adequado?
2. Como evitar o problema de *overfitting*?

Vamos respondê-las fazendo uso de um exemplo.

```
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn import datasets
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
```

## Conjunto de exemplo: *Breast Cancer*

```
X, y = datasets.load_breast_cancer(return_X_y = True)
```

## Dividindo o conjunto de dados em treinamento e teste

```
XTrain, XTest, yTrain, yTest = train_test_split(X, y, test_size = 0.2)
```

## Colocando os atributos na mesma ordem de grandeza

```
scaler = MinMaxScaler()
scaler.fit(XTrain)
XTrain = scaler.transform(XTrain)
```

## Determinando o valor de $C$ por meio de validação cruzada

```
model = SVC(kernel = 'linear')
cValues = np.logspace(-10, 10, 10)
hParameters = {'C': cValues}
```

```
clf = GridSearchCV(model, hParameters, cv = 3)
_ = clf.fit(XTrain, yTrain)
```

## Testando o modelo no conjunto de teste

```
XTest = scaler.transform(XTest)
yPredicted = clf.predict(XTest)
```

## Matriz de Confusão

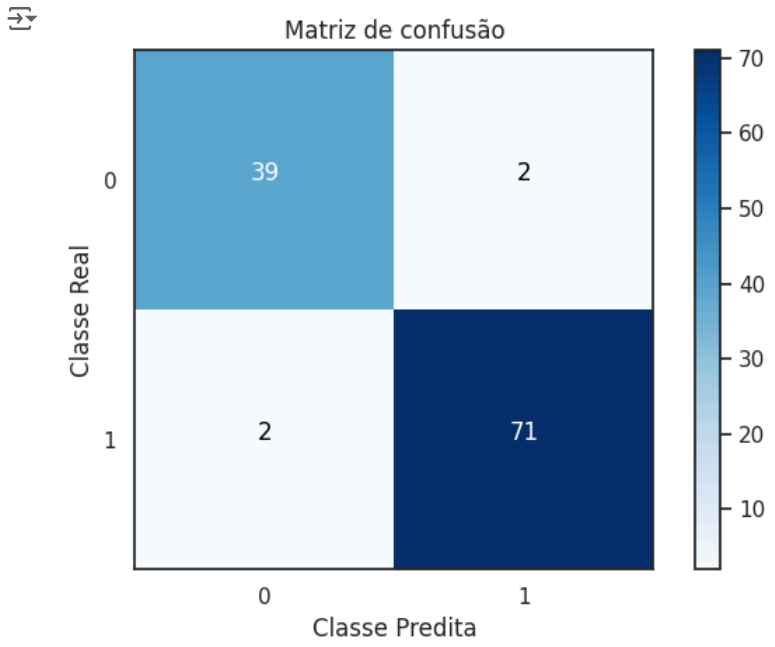
Vamos considerar que há duas classes: a positiva (1) e a negativa (0). Quando tentamos prever a classe de um $\mathbf{x}$ qualquer, quais são os possíveis resultados?

- Positivo Verdadeiro (**TP**): a classe é 1 e acertamos
- Positivo Falso (**FP**): a classe é 1 e erramos
- Negativo Falso (**FN**): a classe é 0 e erramos
- Negativo Verdadeiro (**TN**): a classe é 0 e acertamos

É comum organizar a quantidade de cada um desses eventos em uma matriz, a qual é chamada de **matriz de confusão**.

```
def plot_confusion_matrix(cm,
                          classes,
                          normalize=False,
                          title='Matriz de confusão',
                          cmap=plt.cm.Blues):
    import itertools
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('Classe Real')
    plt.xlabel('Classe Predita')
    plt.tight_layout()
```

```
cm = confusion_matrix(yTest, yPredicted)
sns.set_style('white')
plot_confusion_matrix(cm, [0, 1])
sns.set()
```



**A matriz de confusão é muito utilizada para derivar as seguinte medidas:**

1. **Precision**: dos que dizemos pertencer a classe positiva, qual a fração que realmente pertence?

$$P = \frac{TP}{TP + FP}$$

2. **Recall**: dos que pertencem a classe positiva, qual a fração que dizemos pertencer a classe positiva?

$$R = \frac{TP}{TP + FN}$$

3. **Acurácia**: fração de acertos

$$A = \frac{TP + TN}{TP + TN + FP + FN}$$

4. **F-score**: média harmônica entre **precision** e **recall**

$$F = \frac{2PR}{P + R}$$

```
accuracy_score(yTest, yPredicted)
```

```
0.9649122807017544
```

```
precision_score(yTest, yPredicted)
```

```
0.9726027397260274
```

```
recall_score(yTest, yPredicted)
```

```
0.9726027397260274
```

```
f1_score(yTest, yPredicted)
```

```
0.9726027397260274
```

## Exemplo: Classificando Aplicações de Data Centers

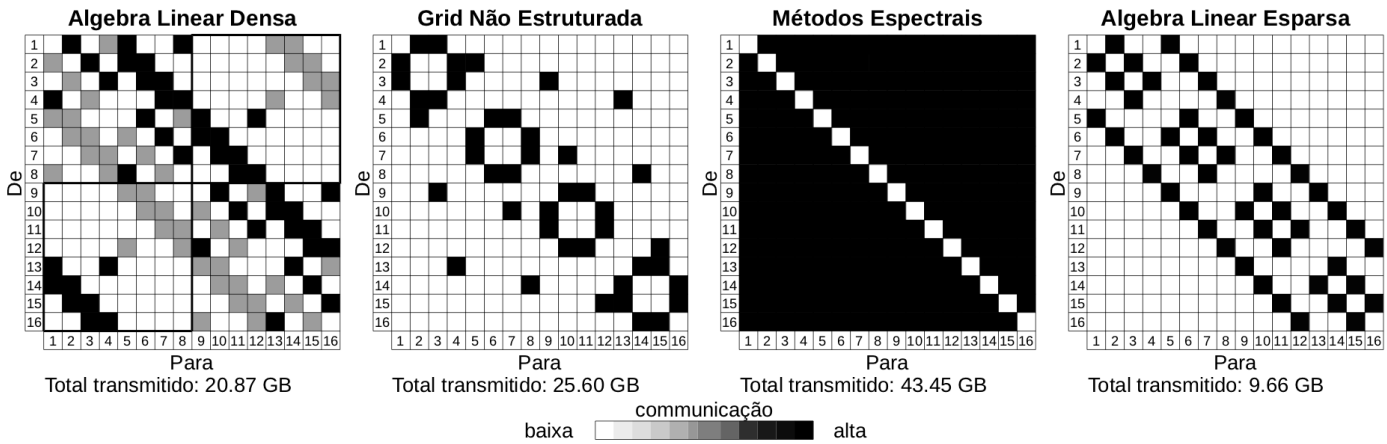
### Referência

**Trois, C., Bona, L. C., Oliveira, L. S., Martinello, M., Harewood- Gill, D., Fabro, M. D. D., Nejabati, R., Simeonidou, D., Lima, J. C. D., and Stein, B.** *Exploring textures in traffic matrices to classify data center communications* (2018)

Diferentes aplicações distribuídas podem ter demandas diferentes por recursos de rede.

Considere a matriz de tráfego de um data center e uma aplicação específica. Nesse caso, a entrada $(i, j)$ representando tráfego total enviado do nó $i$ para o nó $j$.

O exemplo abaixo ilustra diferentes perfis de aplicações científicas para uma estrutura com 16 nós computacionais.

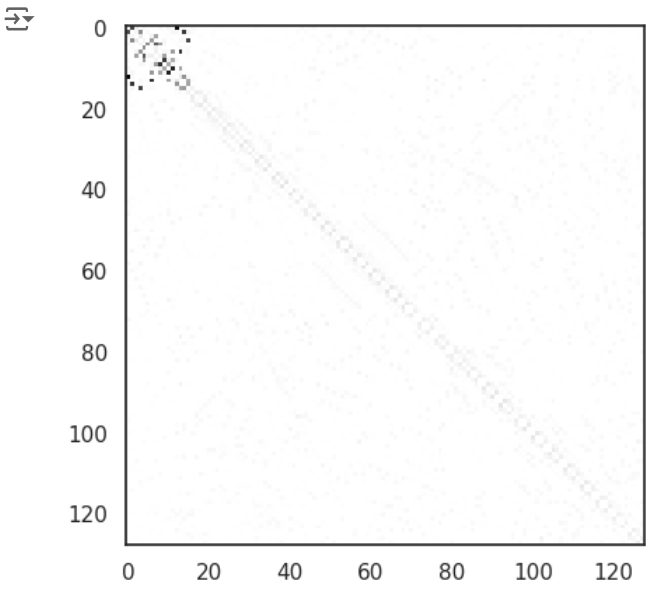



**É possível construir um modelo que aprenda a aplicação dada a imagem representando a matriz de tráfego?**

## 3 Aplicações em 128 nós

```
import glob
import cv2
import numpy as np
from matplotlib import pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics

sns.set_style('white')
```

```
plt.imshow(cv2.imread('datasets-minicurso/images/bstep3d/1.png', 0), cmap=plt.cm.gray, interpolation='nearest');
```




```
plt.imshow(cv2.imread('datasets-minicurso/images/cavity3d/1.png', 0), cmap=plt.cm.gray, interpolation='nearest');
```