

▼ Carregar as bibliotecas

```
import numpy as np
from skimage import io, data, filters, util, color, transform, exposure
from scipy import ndimage, stats, fft
import matplotlib.pyplot as plt
from skimage.util import img_as_float, img_as_uint, img_as_ubyte
np.seterr(divide='ignore', invalid='ignore')

{'divide': 'warn', 'over': 'warn', 'under': 'ignore', 'invalid': 'warn'}
```

```
def show(img_list, caption_list):
    n = len(img_list)
    f, ax = plt.subplots(1,n, figsize=(10+5*(n-1),10))
    for i in range(n):
        ax[i].imshow(img_list[i], cmap='gray')
        ax[i].set_title(caption_list[i])
```

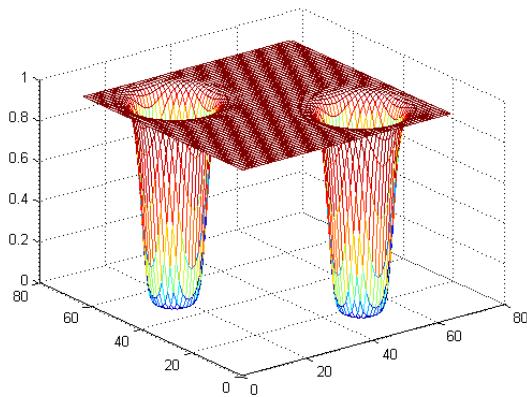
```
def gridFourier(M, N):
    u = np.arange(0, M)
    v = np.arange(0, N)

    u = u - np.floor(M/2)
    v = v - np.floor(N/2)
    U, V = np.meshgrid(u, v)
    return U, V
```

▼ Questão 1

Dada a seguinte imagem, eliminar o ruído produzido pelo ruído periódico. Teste com os seguinte filtros: média, mediana e a filtragem no domínio da frequência. Para o caso da filtragem no domínio da frequência, primeiro calcule o espectro de Fourier (Figura b) e eliminate a \textbf{região} ao redor dos ``spikes'' (assinalados com a seta vermelha). Os ``spikes'' estão localizados nas coordenadas (88,88) e (170,170). A terceira imagem mostra o processo depois de apagar esses valores. Para apagar os valores, basta atribuir zero para todos os elementos da região.

Filtro notch: São filtros capazes de rejeitar uma faixa bastante estreita de frequências. Sua utilização é recomendada quando o sinal a ser atenuado é bem definido. Pelo fato de atuar em faixas reduzidas de frequências, filtros notch interferem pouco na qualidade do sinal. A figura a continuação mostra apenas um par de regiões sendo retirado.



A área em torno da frequência de corte escolhida (D_0) que pode ser retirada é definida na construção do filtro. Seja D_0 a frequência de corte do filtro notch centrado em (u_0, v_0) e, por simetria $(-u_0, -v_0)$:

$$D_1(u, v) = \sqrt{(u - M/2 - u_0)^2 + (v - N/2 - v_0)^2}$$

$$D_2(u, v) = \sqrt{(u - M/2 + u_0)^2 + (v - N/2 + v_0)^2}$$

$$H(u, v) = \frac{1}{1 + \left(\frac{D_0^2}{D_1(u, v) D_2(u, v)} \right)^n}$$

Crie um filtro notch para remover o ruído periódico

```
def mask_butterworth_notch(width, height, d0, n, u, v):
    M=width
    N=height
    u0=M/2
    v0=N/2

    U, V = gridFourier(width, height)
    D = np.sqrt(U**2 + V**2)
    W = np.sqrt(u**2 + v**2)

    D1=np.sqrt( (U-u)**2 + (V-v)**2 )
    D2=np.sqrt( (U+u)**2 + (V+v)**2 )
    #H = 1 / (1 + ((D*W)/(D**2 - d0**2))**(2*n))
    H = 1 / (1 + ((D**2 / (D1 * D2)) ** n))
    #H = 1 / (1 + ((d0**2) / D1 * D2) ** n)
    return H

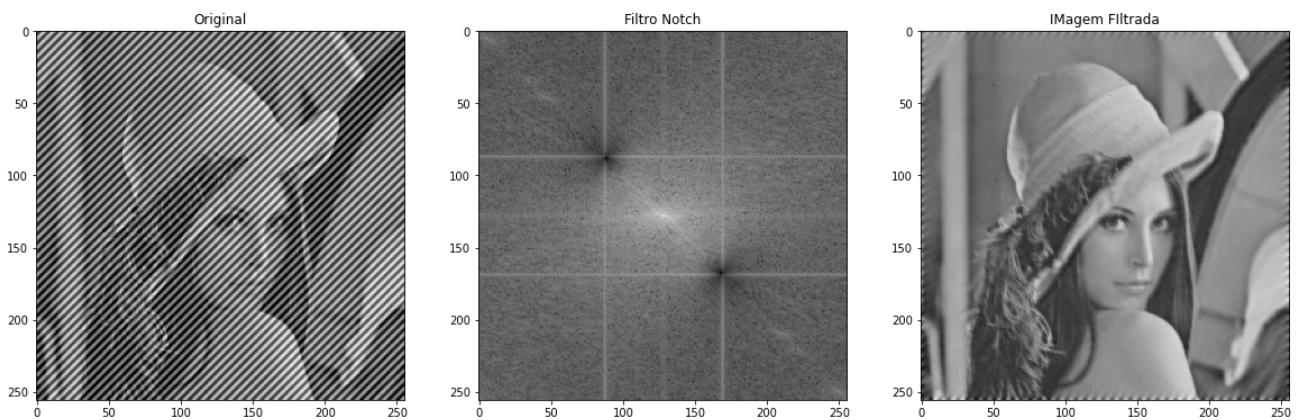
def passa_freq(img, mask):
    height, width = img.shape[:2]
    fimg = fft.fftshift( fft.fft2(img) )
    fimg = fimg * mask
    nimg = fft.ifft2(fimg)
    nimg = np.abs(nimg)
    return nimg, fimg
```

```

lenna = io.imread('https://drive.google.com/uc?id=1ExhUVoHaj2i4gN1UWC6U7RfP-boZhnRL', as_g
lin, col = lenna.shape[:2]
#HN = mask_butterworth_notch(col, lin, 50, 3, 40, 40)
HN = mask_butterworth_notch(col, lin, 58, 3, 40, 40)
nimg, fimg = passa_freq(lenna, HN)

show([lenna, np.log(np.abs(fimg)+1), nimg], ['Original','Filtro Notch', 'IMagem Filtrada'])

```



▼ Questão 2

Repita o processo de remoção do ruído periódico da questão anterior utilizando os seguintes filtro passa-bandas: ideal, Butterworth e Gaussiano.

Filtro Ideal:

$$H(u, v) = \begin{cases} 0 & \text{se } D_0 - \frac{W}{2} \leq D \leq D_0 + \frac{W}{2} \\ 1 & \text{otherwise} \end{cases}$$

Filtro Butterworth:

$$H(u, v) = \frac{1}{1 + \left(\frac{D \cdot W}{D^2 - D_0^2} \right)^{2n}}$$

Filtro Gaussiano:

$$H(u, v) = 1 - e^{\left(-\frac{D^2 - D_0^2}{D \cdot W}\right)^2}$$

```

def mask_ideal_band(width, height, d0, w):
    u=0
    v=0
    H=np.ones((width, height))
    U, V = gridFourier(width, height)
    D = np.sqrt(U**2 + V**2)
    D1=np.sqrt( (U-u)**2 + (V-v)**2 )
    D2=np.sqrt( (U+u)**2 + (V+v)**2 )
    for i in range(len(U)):
        for j in range(len(V)):
            if(d0-W/2<=D[i,j] and D[i,j]<=d0+W/2):
                H[i,j]=0
    return H

def mask_butterworth_band(width, height, d0, n, w):
    v=d0
    u=d0

    U, V = gridFourier(width, height)

    D = np.sqrt(U**2 + V**2)

    D1=np.sqrt( (U-u)**2 + (V-v)**2 )
    D2=np.sqrt( (U+u)**2 + (V+v)**2 )

    H = 1 / (1 + ((D*w)/(D**2 - d0**2))**(2*n))

    return H

def mask_gaussian_band(width, height, d0, w):
    U, V = gridFourier(width, height)
    D = np.sqrt(U**2 + V**2)

    H = 1 - np.exp( ( -(D**2-d0**2) / (D*w) ) **2)
    print(H)

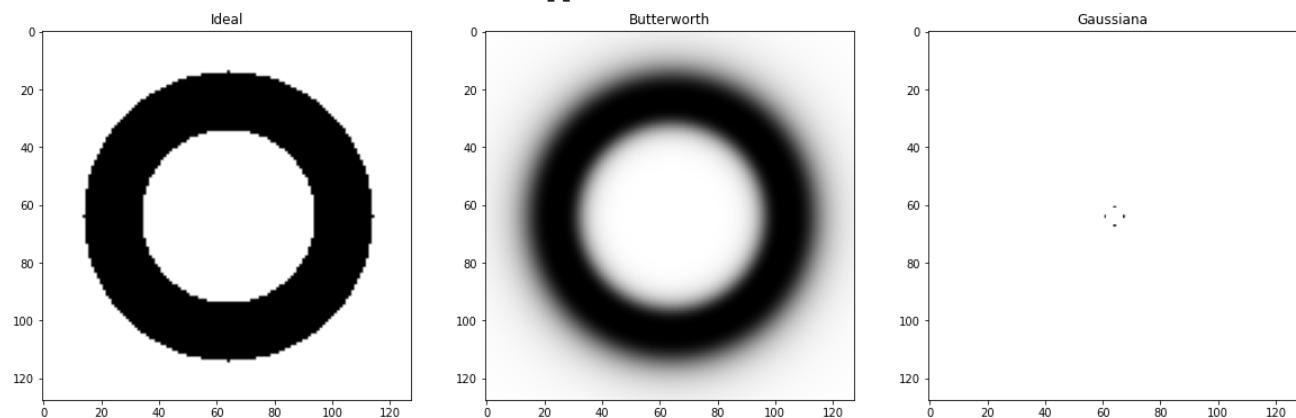
    return H

HI = mask_ideal_band(128, 128, 40, 20)
HB = mask_butterworth_band(128, 128, 40, 2, 20)
HG = mask_gaussian_band(128, 128, 40, 20)
show([HI, HB, HG], ['Ideal', 'Butterworth', 'Gaussiana'])

```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: RuntimeWarning: overfl
```

```
[[-574495.91134065 -423390.42588704 -313652.87251162 ... -233566.97154747
 -313652.87251162 -423390.42588704]
 [-423390.42588704 -312151.90450875 -231339.88041366 ... -172343.33028798
 -231339.88041366 -312151.90450875]
 [-313652.87251162 -231339.88041366 -171520.89088066 ... -127834.72521398
 -171520.89088066 -231339.88041366]
 ...
 [-233566.97154747 -172343.33028798 -127834.72521398 ... -95317.94615645
 -127834.72521398 -172343.33028798]
 [-313652.87251162 -231339.88041366 -171520.89088066 ... -127834.72521398
 -171520.89088066 -231339.88041366]
 [-423390.42588704 -312151.90450875 -231339.88041366 ... -172343.33028798
 -231339.88041366 -312151.90450875]]
```



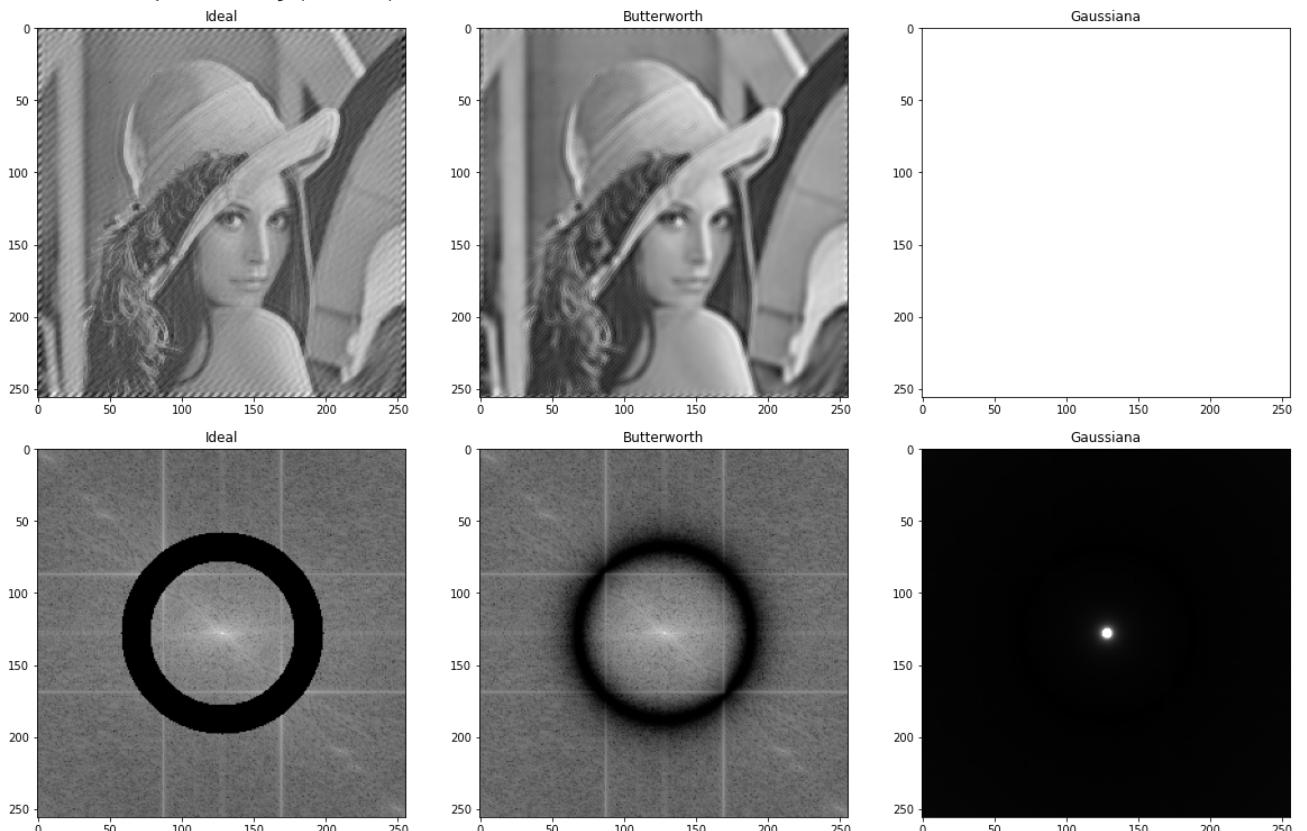
```
lenna = io.imread('https://drive.google.com/uc?id=1ExhUVoHaj2i4gN1UWC6U7RfP-boZhnRL', as_g
[]

lin, col = lenna.shape[:2]

HI = mask_ideal_band(col, lin, 60, 20)
HB = mask_butterworth_band(col, lin, 60, 2, 50)
HG = mask_gaussian_band(col, lin, 60, 50)
nimgI, fimgI = passa_freq(lenna, HI)
nimgB, fimgB = passa_freq(lenna, HB)
nimgG, fimgG = passa_freq(lenna, HG)

show([nimgI, nimgB, nimgG], ['Ideal', 'Butterworth', 'Gaussiana'])
show([np.log(np.abs(fimgI)+1), np.log(np.abs(fimgB)+1), np.log(np.abs(fimgG)+1)], ['Ideal'
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: RuntimeWarning: overfl  
/usr/local/lib/python3.7/dist-packages/matplotlib/image.py:452: UserWarning: Warning  
    dv = np.float64(self.norm.vmax) - np.float64(self.norm.vmin)  
/usr/local/lib/python3.7/dist-packages/matplotlib/image.py:459: UserWarning: Warning  
    a_min = np.float64(newmin)  
/usr/local/lib/python3.7/dist-packages/matplotlib/image.py:464: UserWarning: Warning  
    a_max = np.float64(newmax)  
<string>:6: UserWarning: Warning: converting a masked element to nan.  
/usr/local/lib/python3.7/dist-packages/matplotlib/colors.py:993: UserWarning: Warning  
    data = np.asarray(value)
```

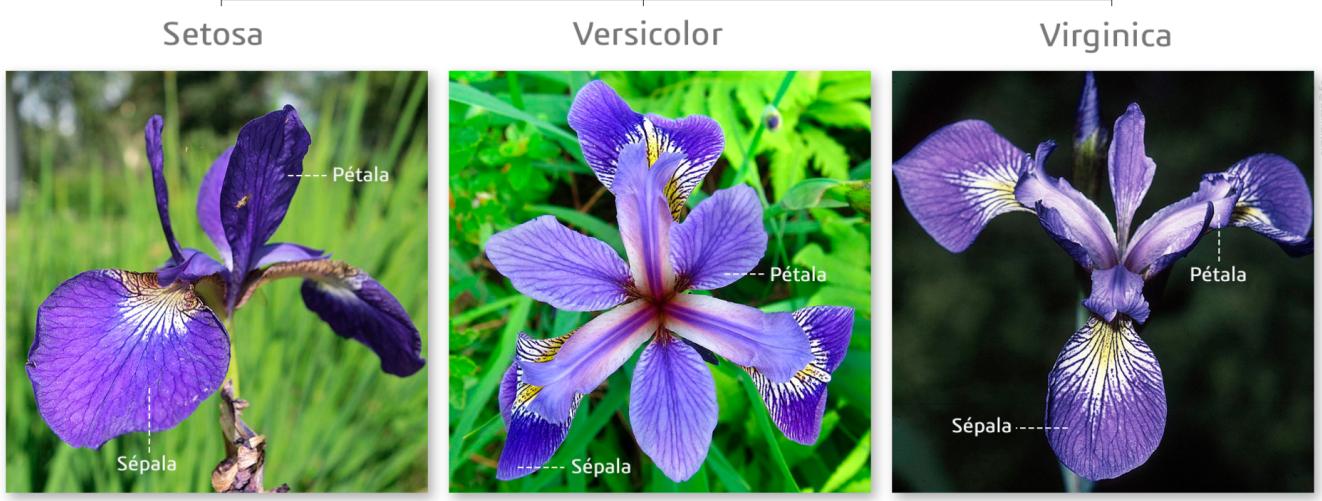


▼ Questão 3: Machine Learning

Base de datos Iris

Base de dados das Flores de Íris

Iris flower dataset



- 50 exemplos de 3 diferentes espécies de flores de iris
 - Medidas: o comprimento e a largura das sépalas e pétalas, em centímetros

Carregando a base Iris

```
# import load_iris function from datasets module
from sklearn.datasets import load_iris
```

```
iris = load_iris()
print(f'Informações presentes no dataset: {iris.keys()}')
print(f'Atributos do conjunto de dados: {iris.feature_names}')
print(f'Nomes das classes: {iris.target_names}')
print(f'Tamanho dos dados: {iris.data.shape}')
print(f'Tamanho das etiquetas: {iris.target.shape}')
```

```
Informações presentes no dataset: dict_keys(['data', 'target', 'frame', 'target_names'])
Atributos do conjunto de dados: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Nomes das classes: ['setosa' 'versicolor' 'virginica']
Tamanho dos dados: (150, 4)
Tamanho das etiquetas: (150,)
```

```
print(iris.data)
```

| L _{0.1} | D ₀ | 4.0 | 1.4 |
|------------------|----------------|-----|------|
| [5.8 | 2.6 | 4. | 1.2] |
| [5. | 2.3 | 3.3 | 1.] |
| [5.6 | 2.7 | 4.2 | 1.3] |
| [5.7 | 3. | 4.2 | 1.2] |
| [5.7 | 2.9 | 4.2 | 1.3] |
| [6.2 | 2.9 | 4.3 | 1.3] |
| [5.1 | 2.5 | 3. | 1.1] |
| [5.7 | 2.8 | 4.1 | 1.3] |
| [6.3 | 3 | 3.6 | 2.5] |

```
[5.8 2.7 5.1 1.9]
[7.1 3.  5.9 2.1]
[6.3 2.9 5.6 1.8]
[6.5 3.  5.8 2.2]
[7.6 3.  6.6 2.1]
[4.9 2.5 4.5 1.7]
[7.3 2.9 6.3 1.8]
[6.7 2.5 5.8 1.8]
[7.2 3.6 6.1 2.5]
[6.5 3.2 5.1 2. ]
[6.4 2.7 5.3 1.9]
[6.8 3.  5.5 2.1]
[5.7 2.5 5.  2. ]
[5.8 2.8 5.1 2.4]
[6.4 3.2 5.3 2.3]
[6.5 3.  5.5 1.8]
[7.7 3.8 6.7 2.2]
[7.7 2.6 6.9 2.3]
[6.  2.2 5.  1.5]
[6.9 3.2 5.7 2.3]
[5.6 2.8 4.9 2. ]
[7.7 2.8 6.7 2. ]
[6.3 2.7 4.9 1.8]
[6.7 3.3 5.7 2.1]
[7.2 3.2 6.  1.8]
[6.2 2.8 4.8 1.8]
[6.1 3.  4.9 1.8]
[6.4 2.8 5.6 2.1]
[7.2 3.  5.8 1.6]
[7.4 2.8 6.1 1.9]
[7.9 3.8 6.4 2. ]
[6.4 2.8 5.6 2.2]
[6.3 2.8 5.1 1.5]
[6.1 2.6 5.6 1.4]
[7.7 3.  6.1 2.3]
[6.3 3.4 5.6 2.4]
[6.4 3.1 5.5 1.8]
[6.  3.  4.8 1.8]
[6.9 3.1 5.4 2.1]
[6.7 3.1 5.6 2.4]
[6.9 3.1 5.1 2.3]
[5.8 2.7 5.1 1.9]
[6.8 3.2 5.9 2.3]
[6.7 3.3 5.7 2.5]
[6.7 3.  5.2 2.3]
[6.3 2.5 5.  1.9]
[6.5 3.  5.2 2. ]
[6.2 3.4 5.4 2.3]
[5.9 3.  5.1 1.8]]
```

▼ Terminología

- Cada linha é uma observação (também conhecida como: amostra, exemplo, instância, registro - *sample, example, instance, record*)
- Cada coluna é um característica (também conhecido como: preditor, atributo, variável independente, entrada, regressor, covariável - *predictor, attribute, independent variable*,

input, regressor, covariate)

Requisitos para trabalhar com dados no scikit-learn

- Características (atributos) e etiquetas são objetos separados
- As características e etiquetas devem ser numéricos
- As características e etiquetas devem ser matrizes NumPy
- As características e etiquetas devem ter formas específicas

Padrão de modelagem de 4 etapas scikit-learn

Passo 1: importe a classe que você planeja usar

```
from sklearn.neighbors import KNeighborsClassifier
```

Passo 2: "Instanciar" o "estimador"

- "Estimator" é o termo do scikit-learn para modelo
- "Instanciar" significa "criar uma instância de"

```
knn = KNeighborsClassifier(n_neighbors=1)
```

- O nome do objeto não importa
- Pode especificar parâmetros de ajuste (também conhecidos como "hiperparâmetros") durante esta etapa
- Todos os parâmetros não especificados são definidos para seus padrões

```
# mostrando informações do modelo
print(knn)
```

```
KNeighborsClassifier(n_neighbors=1)
```

Passo 3: ajustar o modelo com os dados (também conhecido como "treinamento de modelo")

- O modelo está aprendendo a relação entre X e y
- O treino acontece no próprio objeto

```
# características
X = iris.data

# etiquetas
y = iris.target

knn.fit(X, y)

KNeighborsClassifier(n_neighbors=1)
```

Passo 4: predizer a resposta para uma nova observação

- Novas observações são chamadas de dados "fora da amostra"
- Usa as informações que aprendeu durante o processo de treinamento do modelo

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

```
knn.predict([[3, 5, 4, 2]])
array([2])
```

Segundo o classificador, o dado fornecido pertence à terceira classe

▼ Processamentos dos dados

Normalizar os dados

O objetivo da normalização é alterar os valores das colunas numéricas no conjunto de dados para uma escala comum, sem distorcer as diferenças nos intervalos de valores.

Normalização

$$X_{new} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Padronização

$$X_{changed} = \frac{X - \mu}{\sigma}$$

Importando o módulo de preprocessamento

```
from sklearn import preprocessing
```

Normalizando e padronizando os dados da base Iris

```
# normalização
#X_new = preprocessing.MinMaxScaler().fit_transform(X)
minmax_scaler = preprocessing.MinMaxScaler().fit(X)
X_new = minmax_scaler.transform(X)

# padronização
std_scaler = preprocessing.StandardScaler().fit(X)
X_changed = std_scaler.transform(X)
```

▼ Separar o conjunto de dados em subconjuntos disjuntos

- Treino
- Validação
- Teste

Carregando o módulo para dividir a base de dados

```
# permite dividir o conjunto de dados em treino e teste
from sklearn.model_selection import train_test_split
```

A base é dividida em dois subconjuntos

- x_train: dados (atributos) de treino
- x_test: dados para teste
- y_train: etiquetas do conjunto de treino
- y_test: etiquetas do conjunto de teste

```
x_train, x_test, y_train, y_test = train_test_split(X_new, y, test_size=0.2)
```

▼ Seguir os 4 passos de modelagem de scikit-learn

Padrão de modelagem de 4 passos scikit-learn

- Passo 1: importe a classe que você planeja usar
- Passo 2: "Instanciar" o "estimador"
- Passo 3: ajustar o modelo com os dados (também conhecido como "treinamento de modelo")
- Passo 4: predizer a resposta para uma nova observação

```
# Passo 1
from sklearn.neighbors import KNeighborsClassifier
# Passo 2
knn = KNeighborsClassifier(n_neighbors=1)
# Passo 3
knn.fit(x_train, y_train)
# Passo 3
pred = knn.predict(x_test)
```

▼ Calcular a acurácia por classe usando uma matriz de confusão

Usar a função `metrics.confusion_matrix(y_test, pred)`, onde `y_test` são as verdadeiras etiquetas do conjunto de teste, e `y_pred` são as etiquetas preditas pelo classificador treinado.

```
from sklearn import metrics
```

```
m = metrics.confusion_matrix(y_test, pred)
tot = np.sum(m, axis=1, keepdims=True)
np.set_printoptions(precision=2)
m_porc = m/tot

print(f'Matriz de confusão:\n {m_porc}')

print(f'Acuracia média: {np.mean ( m_porc.diagonal() ): 0.2}\' )\n

Matriz de confusão:
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
Acuracia média: 1.0
```

Use o classificador SVM para classificar a base Wine

```
case=2
```

```
from sklearn.datasets import load_wine
wine = load_wine()

X = wine.data

y = wine.target

# Passo 1
from sklearn import svm

# Passo 2
clf_svm = svm.SVC(C=5, kernel='linear')
```

A função *train_test_split* pode retornar os índices das amostradas selecionadas para treinar e testar. Use esses índices para seleccionar o mesmo conjunto de dados:

- sem normalizar (X)
- normalizados (X_{new}), e
- padronizados ($X_{changed}$).

```
indices = np.arange(X.shape[0])
(
    x_train,
    x_test,
    y_train,
    y_test,
    indices_train,
```

```

    indices_test,
) = train_test_split(X, y, indices, test_size=0.2)

```

Verificar se os índices de treino correspondem com os dados atribuidos a *x_train*

```

amostra_treino = x_train[:5, :]
print(f'x_train: \n: {amostra_treino}')

amostra_indice_treino = X[ indices_train[:5] , :]
print(f'dados de treino usando os índices: \n: {amostra_indice_treino}')


x_train:
: [[1.31e+01 1.50e+00 2.10e+00 1.55e+01 9.80e+01 2.40e+00 2.64e+00 2.80e-01
  1.37e+00 3.70e+00 1.18e+00 2.69e+00 1.02e+03]
 [1.18e+01 2.13e+00 2.78e+00 2.85e+01 9.20e+01 2.13e+00 2.24e+00 5.80e-01
  1.76e+00 3.00e+00 9.70e-01 2.44e+00 4.66e+02]
 [1.21e+01 2.16e+00 2.17e+00 2.10e+01 8.50e+01 2.60e+00 2.65e+00 3.70e-01
  1.35e+00 2.76e+00 8.60e-01 3.28e+00 3.78e+02]
 [1.18e+01 1.72e+00 1.88e+00 1.95e+01 8.60e+01 2.50e+00 1.64e+00 3.70e-01
  1.42e+00 2.06e+00 9.40e-01 2.44e+00 4.15e+02]
 [1.29e+01 2.99e+00 2.40e+00 2.00e+01 1.04e+02 1.30e+00 1.22e+00 2.40e-01
  8.30e-01 5.40e+00 7.40e-01 1.42e+00 5.30e+02]]
dados de treino usando os índices:
: [[1.31e+01 1.50e+00 2.10e+00 1.55e+01 9.80e+01 2.40e+00 2.64e+00 2.80e-01
  1.37e+00 3.70e+00 1.18e+00 2.69e+00 1.02e+03]
 [1.18e+01 2.13e+00 2.78e+00 2.85e+01 9.20e+01 2.13e+00 2.24e+00 5.80e-01
  1.76e+00 3.00e+00 9.70e-01 2.44e+00 4.66e+02]
 [1.21e+01 2.16e+00 2.17e+00 2.10e+01 8.50e+01 2.60e+00 2.65e+00 3.70e-01
  1.35e+00 2.76e+00 8.60e-01 3.28e+00 3.78e+02]
 [1.18e+01 1.72e+00 1.88e+00 1.95e+01 8.60e+01 2.50e+00 1.64e+00 3.70e-01
  1.42e+00 2.06e+00 9.40e-01 2.44e+00 4.15e+02]
 [1.29e+01 2.99e+00 2.40e+00 2.00e+01 1.04e+02 1.30e+00 1.22e+00 2.40e-01
  8.30e-01 5.40e+00 7.40e-01 1.42e+00 5.30e+02]]
```

```

# normalização
if(case>=1):
    minmax_scaler = preprocessing.MinMaxScaler().fit(X)
    X_new = minmax_scaler.transform(X)

# padronização
if(case>=2):
    std_scaler = preprocessing.StandardScaler().fit(X)
    X_changed = std_scaler.transform(X)

if(case==0):
    x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
else:
    x_train, x_test, y_train, y_test = train_test_split(X_new, y, test_size=0.2)

if(case==0):
    clf_svm.fit(x_train, y_train)
    pred0 = clf_svm.predict(x_test)

```

```

if(case==1):
    clf_svm.fit(x_train, y_train)
    pred1 = clf_svm.predict(x_test)

if(case==2):
    clf_svm.fit(x_train, y_train)
    pred2 = clf_svm.predict(x_test)

```

Calcule a taxa de acerto para cada tipo de conjunto de dados (sem normalizar, normalizado e padronizado). Para cada novo treino, instancie um novo modelo

```

if(case==0):
    m = metrics.confusion_matrix(y_test, pred0)
    tot = np.sum(m, axis=1, keepdims=True)
    np.set_printoptions(precision=2)
    m_porc = m/tot

    print(f'Matriz de confusão:\n {m_porc}')

    print(f'Acuracia média: {np.mean ( m_porc.diagonal() ): 0.2}\' )
```

Matriz de confusão:
[[0.91 0.09 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
Acuracia média: 0.97

```

if(case==1):
    m = metrics.confusion_matrix(y_test, pred1)
    tot = np.sum(m, axis=1, keepdims=True)
    np.set_printoptions(precision=2)
    m_porc = m/tot

    print(f'Matriz de confusão:\n {m_porc}')

    print(f'Acuracia média: {np.mean ( m_porc.diagonal() ): 0.2}\' )
```

Matriz de confusão:
[[1. 0. 0.]
 [0.06 0.94 0.]
 [0. 0. 1.]]
Acuracia média: 0.98

```

if(case==2):
    m = metrics.confusion_matrix(y_test, pred2)
    tot = np.sum(m, axis=1, keepdims=True)
    np.set_printoptions(precision=2)
    m_porc = m/tot

    print(f'Matriz de confusão:\n {m_porc}')
```

print(f'Acuracia média: {np.mean (m_porc.diagonal()): 0.2}\')

↳ Matriz de confusão:

```
[[1.  0.  0.]
 [0.  0.91 0.09]
 [0.  0.  1.]]
```

Acuracia média: 0.97

Produtos pagos do Colab - Cancelar contratos

✓ 0s conclusão: 18:07

