```
# 1) Imports
import numpy as np
import cv2
import matplotlib.pyplot as plt
from google.colab import files
from sklearn.decomposition import PCA
import math
```

```
# Função auxiliar para mostrar imagens lado a lado
def show_images(titles, images, cmap='gray', figsize=(15,5)):
    n = len(images)
    plt.figure(figsize=figsize)
    for i,img in enumerate(images):
        plt.subplot(1,n,i+1)
        if img.ndim==2:
            plt.imshow(img, cmap=cmap, vmin=0, vmax=255)
        else:
            plt.imshow(img)
        plt.title(titles[i])
        plt.axis('off')
    plt.tight_layout()
    plt.show()
```

```
# 2) Upload da imagem
uploaded = files.upload()

filename = next(iter(uploaded.keys()))
print("Arquivo enviado:", filename)
```

```
Escolher arquivos athletico.jpg
athletico.jpg(image/jpeg) - 77530 bytes, last modified: 27/10/2025 - 100% done
Saving athletico.jpg to athletico (1).jpg
Arquivo enviado: athletico (1).jpg
```

```
# 3) Leitura e conversão para grayscale
img_color = cv2.imread(filename)
if img_color is None:
    raise RuntimeError("Erro ao ler a imagem. Verifique o nome/arquivo.")
img_rgb = cv2.cvtColor(img_color, cv2.COLOR_BGR2RGB)
img_gray = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)
```

```
# 4) Mostrar imagem colorida e escala de cinza
show_images(['Original (RGB)', 'Cinza'], [img_rgb, img_gray], cmap='gray', figsize=(12,5))
```


```
# 5) Mostrar dimensões (pixels)
h, w = img_gray.shape
print(f"Dimensões da imagem em pixels: altura (rows) = {h}, largura (cols) = {w}")
print(f"Total de pixels = {h*w}")
```

```
Dimensões da imagem em pixels: altura (rows) = 531, largura (cols) = 850
Total de pixels = 451350
```

```
# 6) Converter para formato padrão para PCA (float, normalizado opcionalmente)
X = img_gray.astype(np.float64)
print(f"Formatação para PCA: matriz X com shape {X.shape} (observações x características)")
```

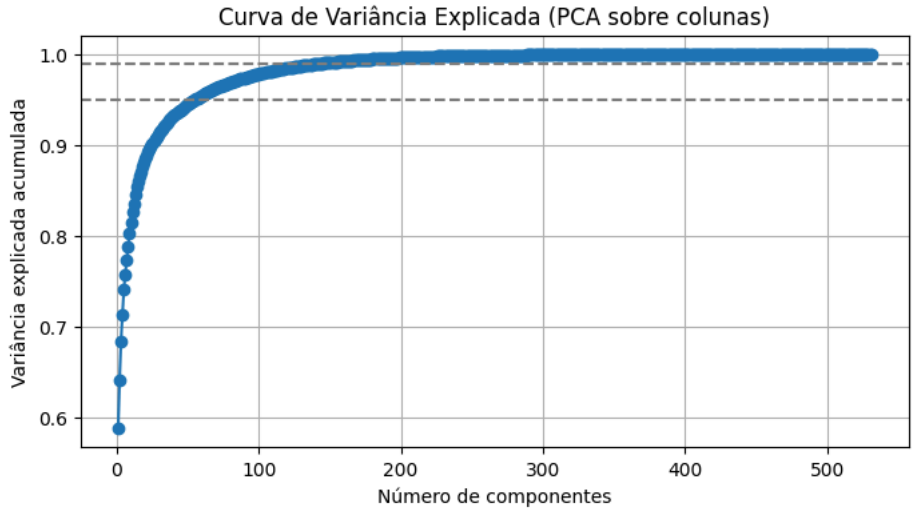
```
Formatação para PCA: matriz X com shape (531, 850) (observações x características)
```

```
# 7) Ajustar PCA. Primeiro vamos examinar quantos componentes explicam variância.
pca_full = PCA(svd_solver='full', whiten=False)
```

```
pca_full.fit(X)

explained_ratio = pca_full.explained_variance_ratio_
cumulative = np.cumsum(explained_ratio)

# Plot da variância explicada acumulada
plt.figure(figsize=(8,4))
plt.plot(np.arange(1, len(cumulative)+1), cumulative, marker='o')
plt.xlabel('Número de componentes')
plt.ylabel('Variância explicada acumulada')
plt.title('Curva de Variância Explicada (PCA sobre colunas)')
plt.grid(True)
plt.axhline(0.95, color='gray', linestyle='--')
plt.axhline(0.99, color='gray', linestyle='--')
plt.show()
```




```
# 8) Selecionar componentes que explicam 95% e 99% (usando n_components como float permite escolha automática)
pca_95 = PCA(n_components=0.95, svd_solver='full')
pca_95.fit(X)
n_comp_95 = pca_95.n_components_
print(f"Nº de componentes selecionados para 95% de variância explicada: {n_comp_95}")

pca_99 = PCA(n_components=0.99, svd_solver='full')
pca_99.fit(X)
n_comp_99 = pca_99.n_components_
print(f"Nº de componentes selecionados para 99% de variância explicada: {n_comp_99}")
```

```
Nº de componentes selecionados para 95% de variância explicada: 55
Nº de componentes selecionados para 99% de variância explicada: 142
```

```
# 9) Reconstruir a imagem usando inversa das projeções (transform -> inverse_transform)
X_trans_95 = pca_95.transform(X)       # shape (h, n_comp_95)
X_rec_95 = pca_95.inverse_transform(X_trans_95)  # shape (h, w)

X_trans_99 = pca_99.transform(X)       # shape (h, n_comp_99)
X_rec_99 = pca_99.inverse_transform(X_trans_99)  # shape (h, w)

# Garantir intervalo válido 0-255 e dtype uint8 para exibição
def to_uint8(im):
    im = np.clip(im, 0, 255)
    return im.astype(np.uint8)

img_rec_95 = to_uint8(X_rec_95)
img_rec_99 = to_uint8(X_rec_99)
```

```
# 10) Mostrar imagens: original e reconstruções
show_images(
    ['Original (cinza)', f'Reconstruída (95% -> {n_comp_95} comp.)', f'Reconstruída (99% -> {n_comp_99} comp.)'],
    [img_gray, img_rec_95, img_rec_99],
    cmap='gray',
    figsize=(15,6)
)
```

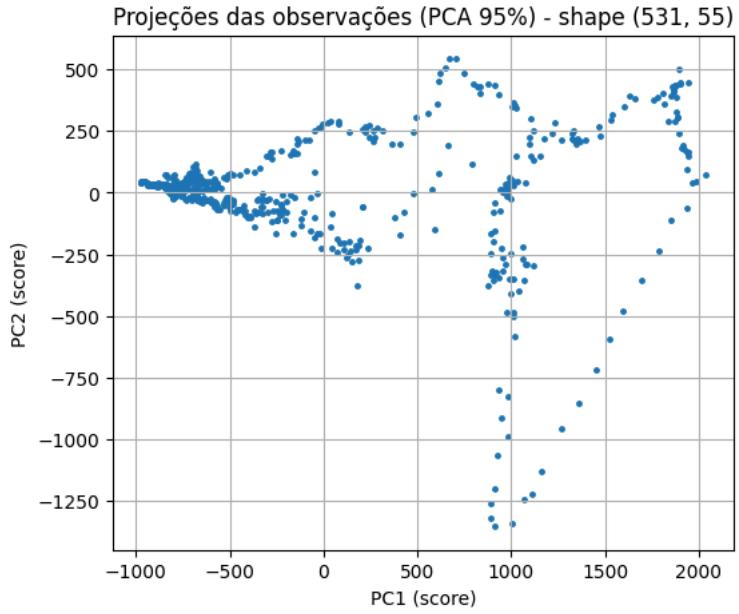

```
# 11) Mostrar quantidade de características:
print("Quantidade de características / componentes:")
print(f" - Características originais por observação (largura da imagem) = {w}")
print(f" - Componentes usados (95%) = {n_comp_95}")
print(f" - Componentes usados (99%) = {n_comp_99}")
print(f" - Total de observações (linhas) = {h}")
print(f" - Total de pixels (original) = {h*w}")
print()
print("Observação: a reconstrução tem a mesma forma (h x w) mas foi representada por menos componentes (redução dimensional).")
```

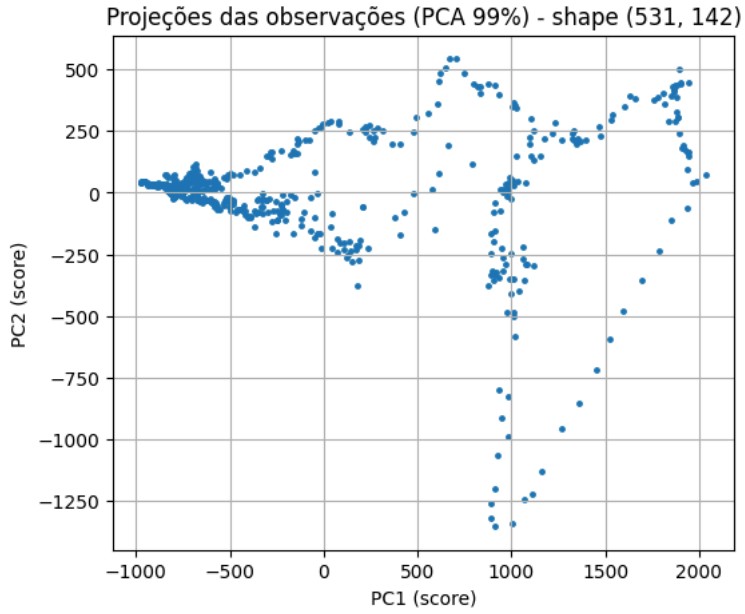
```
Quantidade de características / componentes:
 - Características originais por observação (largura da imagem) = 850
 - Componentes usados (95%) = 55
 - Componentes usados (99%) = 142
 - Total de observações (linhas) = 531
 - Total de pixels (original) = 451350

Observação: a reconstrução tem a mesma forma (h x w) mas foi representada por menos componentes (redução dimensional).
```

```
# 12) Gráficos das projeções das observações - plot primeira e segunda componente
def plot_scores(scores, title):
    plt.figure(figsize=(6,5))
    if scores.shape[1] >= 2:
        plt.scatter(scores[:,0], scores[:,1], s=6)
        plt.xlabel('PC1 (score)')
        plt.ylabel('PC2 (score)')
    else:
        plt.plot(scores[:,0], np.zeros_like(scores[:,0]), 'o')
        plt.xlabel('PC1 (score)')
        plt.yticks([])
    plt.title(title)
    plt.grid(True)
    plt.show()

plot_scores(X_trans_95, f'Projeções das observações (PCA 95%) - shape {X_trans_95.shape}')
plot_scores(X_trans_99, f'Projeções das observações (PCA 99%) - shape {X_trans_99.shape}')
```






```
# 13) Mostrar autovetores (componentes principais)
def plot_components(pca_obj, n_to_show=5):
    comps = pca_obj.components_
    n_show = min(n_to_show, comps.shape[0])
    plt.figure(figsize=(12, 2.5*n_show))
    for i in range(n_show):
        plt.subplot(n_show, 2, 2*i+1)
        plt.plot(comps[i])
        plt.title(f'Autovetor {i+1} (linha) - comprimento = {comps.shape[1]}')
        plt.grid(True)
        plt.subplot(n_show, 2, 2*i+2)

        vec = comps[i]
        vec_img = (vec - vec.min()) / (vec.max() - vec.min() + 1e-12) * 255
        vec_img = np.tile(vec_img.reshape(1, -1), (30,1))
        plt.imshow(vec_img, cmap='gray', aspect='auto', vmin=0, vmax=255)
        plt.title(f'Autovetor {i+1} (como imagem)')
        plt.axis('off')
    plt.tight_layout()
    plt.show()

print("Autovetores para PCA 95% (primeiros 5):")
plot_components(pca_95, n_to_show=5)

print("Autovetores para PCA 99% (primeiros 5):")
plot_components(pca_99, n_to_show=5)
```