

---

# Actividad Semanal -- 5 Repaso Transformación y reducción de dimensiones

Ciencia y analítica de datos

Profesor: María de la Paz Rico Fernández

Juan Sebastián Ortega Briones A01794327

## Bienvenido al notebook

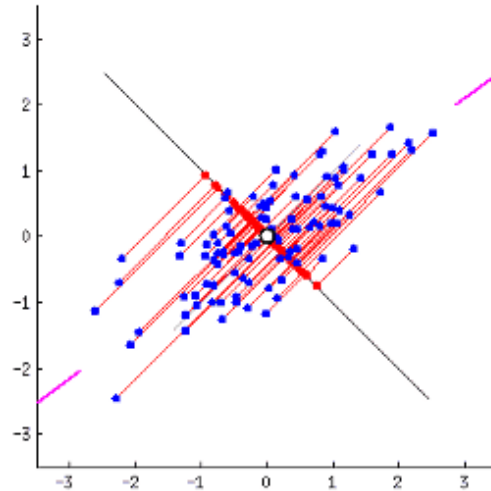
### Repaso de Reducción de dimensiones

El objetivo es que entendamos de una manera visual, que es lo que pasa cuando nosotros seleccionamos cierto número de componentes principales o % de variabilidad de una base de datos.

Primero entenderemos, que pasa adentro de PCA que se basa en lo siguiente a grandes razgos:

#### **Análisis de Componentes Principales**

El análisis de datos multivariados involucra determinar transformaciones lineales que ayuden a entender las relaciones entre las características importantes de los datos. La idea central del Análisis de Componentes Principales (PCA) es reducir las dimensiones de un conjunto de datos que presenta variaciones correlacionadas, reteniendo una buena proporción de la variación presente en dicho conjunto. Esto se logra obteniendo la transformación a un nuevo conjunto de variables: los componentes principales (PC). Cada PC es una combinación lineal con máxima varianza en dirección ortogonal a los demás PC.



Para entender un poco más de PCA y SVD, visita el siguiente link: *Truco: Prueba entrar con tu cuenta del tec :)*

<https://towardsdatascience.com/pca-and-svd-explained-with-numpy-5d13b0d2a4d8>

Basicamente, vamos a seguir los siguientes pasos:

1. Obtener la covarianza. OJO: X tiene sus datos centrados :)

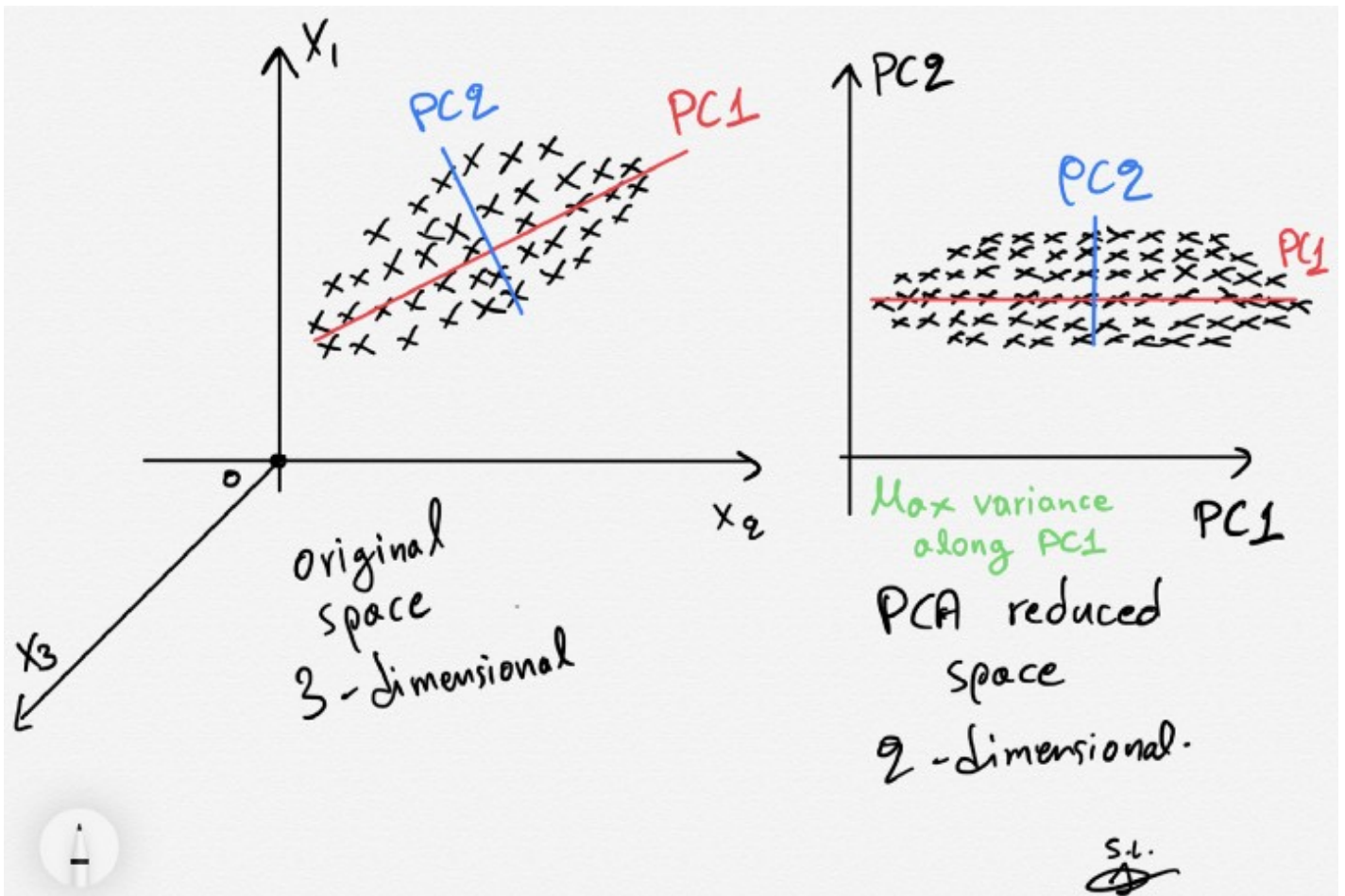
$$\mathbf{C} = \frac{\mathbf{X}^T \mathbf{X}}{n - 1}$$

2. Los componentes principales se van a obtener de la eigen descomposicion de la matriz de covarianza.

$$\mathbf{C} = \mathbf{W} \mathbf{\Lambda} \mathbf{W}^{-1}$$

3. Para la reducción de dimensiones vamos a seleccionar k vectores de W y proyectaremos nuestros datos.

$$\mathbf{X}_k = \mathbf{X} \mathbf{W}_k$$



## Ejercicio 1, Descomposición y composición

### Descomposición

Encuentra los eigenvalores y eigenvectores de las siguientes matrices

$$A = \begin{pmatrix} 3, 0, 2 \\ 3, 0, -2 \\ 0, 1, 1 \end{pmatrix} \quad A2 = \begin{pmatrix} 1, 3, 8 \\ 2, 0, 0 \\ 0, 0, 1 \end{pmatrix} \quad A3 = \begin{pmatrix} 5, 4, 0 \\ 1, 0, 1 \\ 10, 7, 1 \end{pmatrix}$$

y reconstruye la matriz original a través de las matrices  $WDW^{-1}$  (OJO. Esto es lo mismo de la ecuación del paso 2 solo le cambiamos la variable a la matriz diagonal)

### ▼ Eigenvalores y eigenvectores

```
###-----EJEMPLO DE EIGENVALORES
import numpy as np
from numpy import array
```

```

from numpy.linalg import eig
# define la matriz
A = array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("-----Matriz original-----")
print(A)
print("-----")
# calcula la eigendescomposición
values, vectors = eig(A)
print("valores eigen", values) #D
print(10*" ")
print("vectores",vectors) #W

#Ejemplo de reconstrucción

values, vectors = np.linalg.eig(A)

W = vectors
Winv = np.linalg.inv(W)
D = np.diag(values)
#la matriz B tiene que dar igual a A
#reconstruye la matriz
print("-----Matriz reconstruida-----")
# Realiza la reconstruccion de B=W*D*Winv, te da lo mismo de A?
#ojo, estas multiplicando matrices, no escalares ;)
#TU CODIGO AQUI-----
B= np.dot(W,np.dot(D,Winv)) # Reconstruccion
print(B)
print("-----")

-----Matriz original-----
[[1 2 3]
 [4 5 6]
 [7 8 9]]
-----
valores eigen [ 1.61168440e+01 -1.11684397e+00 -1.30367773e-15]
-----
vectores [[-0.23197069 -0.78583024  0.40824829]
 [-0.52532209 -0.08675134 -0.81649658]
 [-0.8186735   0.61232756  0.40824829]]
-----Matriz reconstruida-----
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
-----

A = array([[3, 0, 2], [3, 0, -2], [0, 1, 1]])
values, vectors = eig(A)
print(values) #D
print(vectors) #W

[3.54451153+0.j          0.22774424+1.82582815j  0.22774424-1.82582815j]
[[-0.80217543+0.j          -0.04746658+0.2575443j  -0.04746658-0.2575443j ] ]

```

```

[-0.55571339+0.j      0.86167879+0.j      0.86167879-0.j      ]
[-0.21839689+0.j     -0.16932106-0.40032224j -0.16932106+0.40032224j]]

```

```

#Matriz 1
import numpy as np
from numpy import array
from numpy.linalg import eig
# define la matriz
A = array([[3, 0, 2], [3 ,0, -2], [0, 1 ,1]])

print("-----Matriz original-----")
print(A)
print("-----")
# calcula la eigendescomposición
values, vectors = eig(A)
print("valores eigen", values) #D
print(10*" -")
print("vectores",vectors) #W

#Ejemplo de reconstrucción

values, vectors = np.linalg.eig(A)

W = vectors
Winv = np.linalg.inv(W)
D = np.diag(values)
#la matriz B tiene que dar igual a A
#reconstruye la matriz
print("-----Matriz reconstruida-----")
# Realiza la reconstruccion de B=W*D*Winv, te da lo mismo de A?
#ojo, estas multiplicando matrices, no escalares ;)
#TU CODIGO AQUI-----
B= np.dot(W,np.dot(D,Winv))      # Reconstruccion
print(B.round(decimals=0).real) # Redondeada y eliminando los numeros complejos
print("-----")

-----Matriz original-----
[[ 3  0  2]
 [ 3  0 -2]
 [ 0  1  1]]
-----
valores eigen [3.54451153+0.j      0.22774424+1.82582815j  0.22774424-1.825828
-----
vectores [[-0.80217543+0.j      -0.04746658+0.2575443j  -0.04746658-0.2575443
 [-0.55571339+0.j      0.86167879+0.j      0.86167879-0.j      ]
 [-0.21839689+0.j     -0.16932106-0.40032224j -0.16932106+0.40032224j]]
-----Matriz reconstruida-----
[[ 3.  0.  2.]
 [ 3.  0. -2.]

```

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

```
-----  
#Matriz 2  
import numpy as np  
from numpy import array  
from numpy.linalg import eig  
# define la matriz  
A = array([[1, 3, 8], [2, 0, 0], [0, 0, 1]])  
print("-----Matriz original-----")  
print(A)  
print("-----")  
# calcula la eigendescomposición  
values, vectors = eig(A)  
print("valores eigen", values) #D  
print(10*" -")  
print("vectores", vectors) #W  
  
#Ejemplo de reconstrucción  
  
values, vectors = np.linalg.eig(A)  
  
W = vectors  
Winv = np.linalg.inv(W)  
D = np.diag(values)  
#la matriz B tiene que dar igual a A  
#reconstruye la matriz  
print("-----Matriz reconstruida-----")  
# Realiza la reconstruccion de B=W*D*Winv, te da lo mismo de A?  
#ojo, estas multiplicando matrices, no escalares ;)  
#TU CODIGO AQUI-----  
B= np.dot(W,np.dot(D,Winv))    # Reconstruccion  
print(B.round(decimals=0))      # Redondeada  
print("-----")  
  
-----Matriz original-----  
[[1 3 8]  
 [2 0 0]  
 [0 0 1]]  
-----  
valores eigen [ 3. -2.  1.]  
-----  
vectores [[ 0.83205029 -0.70710678 -0.42399915]  
 [ 0.5547002   0.70710678 -0.8479983 ]  
 [ 0.          0.          0.31799936]]  
-----Matriz reconstruida-----  
[[1. 3. 8.]  
 [2. 0. 0.]  
 [0. 0. 1.]]  
-----
```

```
#Matriz 3
```

```
#Matriz 2
```

```
import numpy as np
from numpy import array
from numpy.linalg import eig
# define la matriz
A = array([[5, 4, 0], [1, 0, 1], [10, 7, 1]])
print("-----Matriz original-----")
print(A)
print("-----")
# calcula la eigendescomposición
values, vectors = eig(A)
print("valores eigen", values) #D
print(10*" -")
print("vectores",vectors) #W
```

```
#Ejemplo de reconstrucción
```

```
values, vectors = np.linalg.eig(A)
```

```
W = vectors
Winv = np.linalg.inv(W)
D = np.diag(values)
#la matriz B tiene que dar igual a A
#reconstruye la matriz
print("-----Matriz reconstruida-----")
# Realiza la reconstruccion de B=W*D*Winv, te da lo mismo de A?
#ojo, estas multiplicando matrices, no escalares ;)
#TU CODIGO AQUI-----
B= np.dot(W,np.dot(D,Winv))    # Reconstruccion
print(B.round(decimals=0))      # Redondeada
print("-----")
```

```
-----Matriz original-----
[[ 5  4  0]
 [ 1  0  1]
 [10  7  1]]
-----
valores eigen [ 6.89167094 -0.214175  -0.67749594]
-----
vectores [[ 0.3975395  0.55738222  0.57580768]
 [ 0.18800348 -0.72657211 -0.81728644]
 [ 0.89811861 -0.40176864 -0.02209943]]
-----Matriz reconstruida-----
[[ 5.  4. -0.]
 [ 1. -0.  1.]
 [10.  7.  1.]]
-----
```

## ¿Qué significa reducir dimensiones?

Esto será cuando proyectemos a ese espacio de los componentes principales pero no los seleccionemos todos, solo los más importantes y viajemos de regreso a nuestras unidades a través de una proyección.

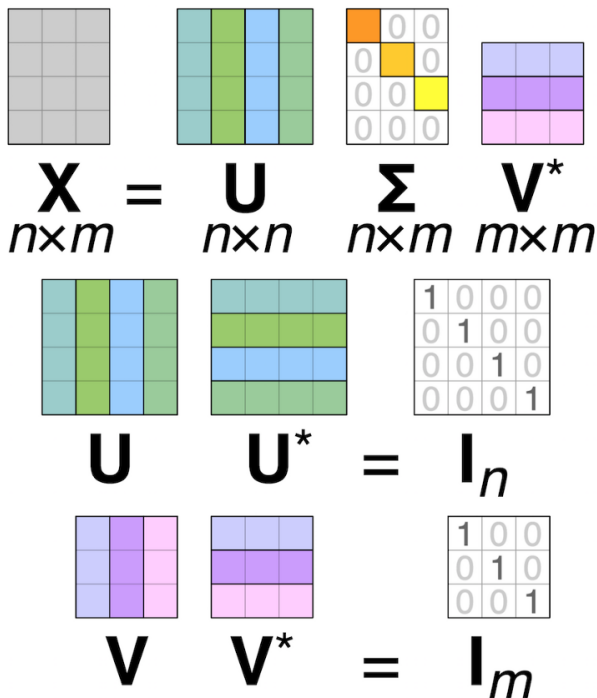
Es decir: Unidades-PC PC-Unidades

Veámoslo gráficamente, ¿qué pasa con esa selección de los PCs y su efecto?.

Para ello usaremos Singular Value Descomposition (SVD).

## Singular Value Descomposition(SVD)

Es otra descomposición que tambien nos ayudara a reducir dimensiones.



The diagram illustrates the SVD decomposition of a matrix  $X$  into matrices  $U$ ,  $\Sigma$ , and  $V^*$ . The decomposition is shown as  $X = U \Sigma V^*$  with dimensions  $n \times m = n \times n \times n \times m \times m \times m$ .

Matrix  $X$  is a 4x4 grid. Matrix  $U$  is a 4x4 grid. Matrix  $\Sigma$  is a 4x4 grid with diagonal elements 1, 0, 0, 0. Matrix  $V^*$  is a 4x4 grid.

Matrix  $U$  is a 4x4 grid. Matrix  $U^*$  is a 4x4 grid. Matrix  $I_n$  is a 4x4 grid with diagonal elements 1, 0, 0, 0.

Matrix  $V$  is a 4x4 grid. Matrix  $V^*$  is a 4x4 grid. Matrix  $I_m$  is a 4x4 grid with diagonal elements 1, 0, 0, 0.

## ▼ Ejercicio 2

Juega con Lucy, una cisne, ayudala a encontrar cuantos valores singulares necesita para no perder calidad a través de SVD. Posteriormente usa 3 imágenes de tu preferencia y realiza la misma acción :D

A esto se le llama **compresión de imagenes** :o

Double-click (or enter) to edit



```

from six.moves import urllib
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np

plt.style.use('classic')
img = Image.open(urllib.request.urlopen('https://biblioteca.acropolis.org/wp-content/
#img = Image.open('lucy.jpg')
imggray = img.convert('LA')
imgmat = np.array(list(imggray.getdata(band=0)),float)

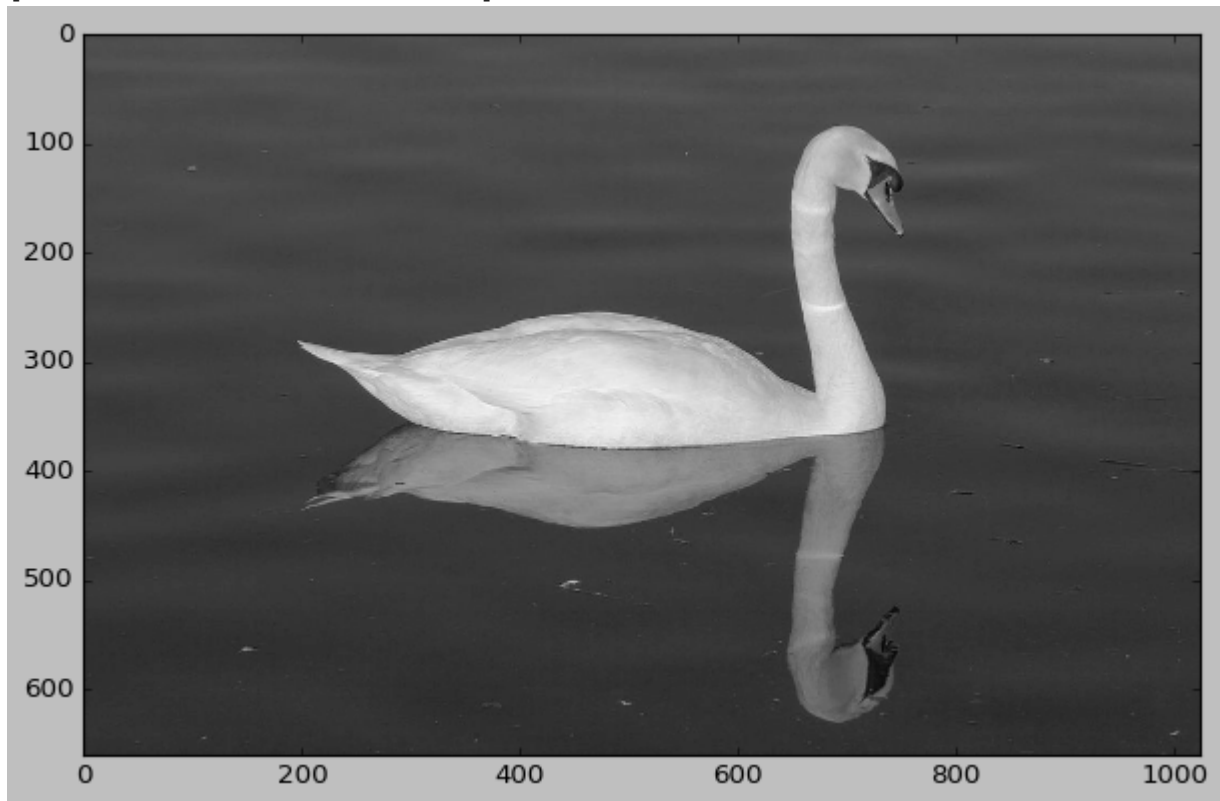
print(imgmat)

imgmat.shape = (imggray.size[1],imggray.size[0])

plt.figure(figsize=(9,6))
plt.imshow(imgmat,cmap='gray')
plt.show()
print(img)

```

```
[72. 73. 74. ... 48. 47. 47.]
```



```
<PIL.Image.Image image mode=LA size=1024x660 at 0x7F72AD31C650>
```

```

U,D,V = np.linalg.svd(imgmat)
imgmat.shape

```

```
(660, 1024)
```

U.shape

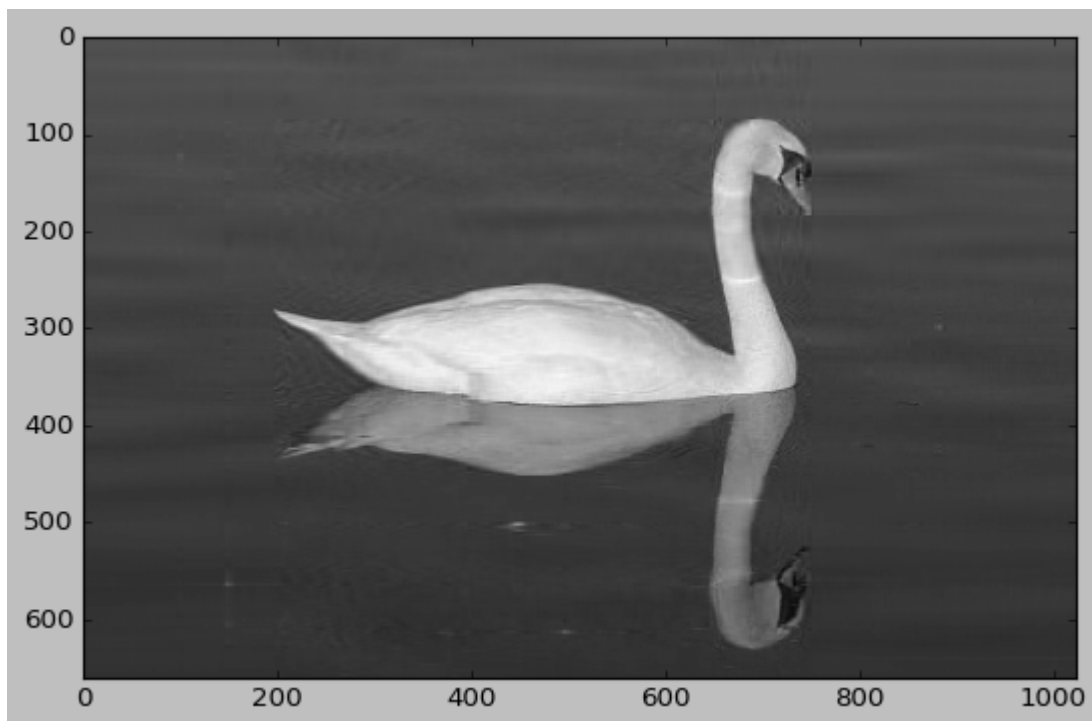
(660, 660)

V.shape

(1024, 1024)

```
#Cuantos valores crees que son necesarios?
#A=U*D*V
#aqui los elegiremos-----
# por las dimensiones de este caso en particular
#iremos de 0-660, siendo 660 como normalmente están los datos
#con 50 podemos observar que Lucy se ve casi igual, es decir conservamos aquello que
# realidad estaba aportando a la imagen en este caso :D por medio de la variabilidad
#juega con el valor nvalue y ve que pasa con otros valores
nvalue = 45
#-----
reconstimg = np.matrix(U[:, :nvalue])*np.diag(D[:nvalue])*np.matrix(V[:nvalue, :])
#ve las dimensiones de la imagen y su descomposicion
#660x1024= U(660X660)D(660X1024)V(1024x1024)
      #=U(660Xnvalues)D(nvaluesXnvalue)V(nvaluesx1024)

      #=U(660X50)(50X50)(50X1024)
plt.imshow(reconstimg, cmap='gray')
plt.show()
print("Felicidades la imagen está comprimida")
```



Felicidades la imagen está comprimida

¡Ahora es tu turno!, comprime 3 imagenes

#imagen 1

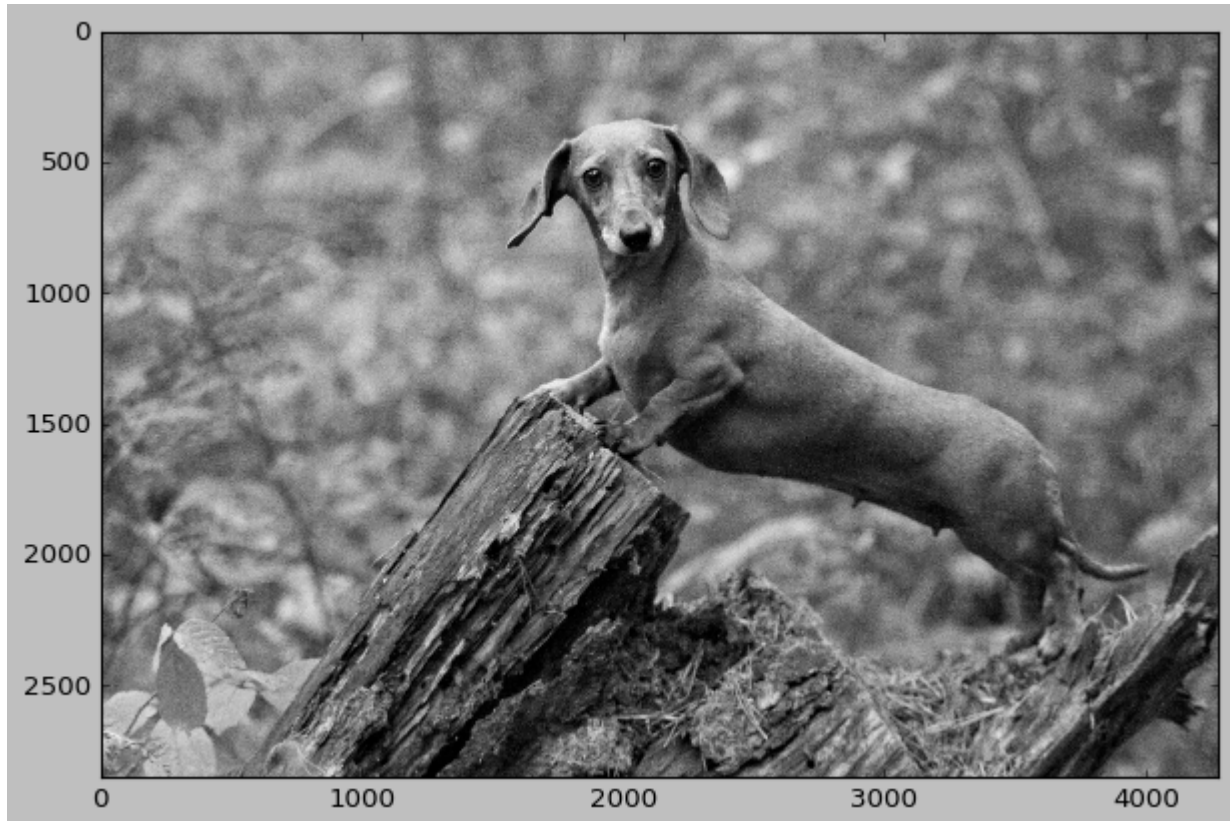
```
plt.style.use('classic')
url="https://www.lavanguardia.com/uploads/2022/05/13/627e3cce881f9.jpeg"
img = Image.open(urllib.request.urlopen(url)).convert('LA')
imggray = img.convert('LA')
imgmat = np.array(list(imggray.getdata(band=0)),float)

print(imgmat)

imgmat.shape = (imggray.size[1],imggray.size[0])

plt.figure(figsize=(9,6))
plt.imshow(imgmat,cmap='gray')
plt.show()
print(img)
U,D,V = np.linalg.svd(imgmat)
print("imgmat",imgmat.shape)
print("U",U.shape)
print("V",V.shape)
nvalue = 50
#-----
reconstimg = np.matrix(U[:, :nvalue])*np.diag(D[:nvalue])*np.matrix(V[:nvalue, :])
plt.imshow(reconstimg,cmap='gray')
print("Imagen comprimida")
plt.show()
```

```
[102. 101. 108. ... 57. 61. 59.]
```



```
<PIL.Image.Image image mode=LA size=4272x2848 at 0x7F72A96C4090>
```

```
imgmat (2848, 4272)
```

```
U (2848, 2848)
```

```
V (4272, 4272)
```

```
Traceback (most recent call last):
```

```
#imagen 2
```

```
plt.style.use('classic')
```

```
url="https://www.wasky.es/wp-content/uploads/adorable-animal-black-and-white-breed-22
```

```
img = Image.open(urllib.request.urlopen(url)).convert('LA')
```

```
imggray = img.convert('LA')
```

```
imgmat = np.array(list(imggray.getdata(band=0)),float)
```

```
print(imgmat)
```

```
imgmat.shape = (imggray.size[1],imggray.size[0])
```

```
plt.figure(figsize=(9,6))
```

```
plt.imshow(imgmat,cmap='gray')
```

```
plt.show()
```

```
print(img)
```

```
U,D,V = np.linalg.svd(imgmat)
```

```
print("imgmat",imgmat.shape)
```

```
print("U",U.shape)
```

```
print("V",V.shape)
```

```
nvalue = 50
```

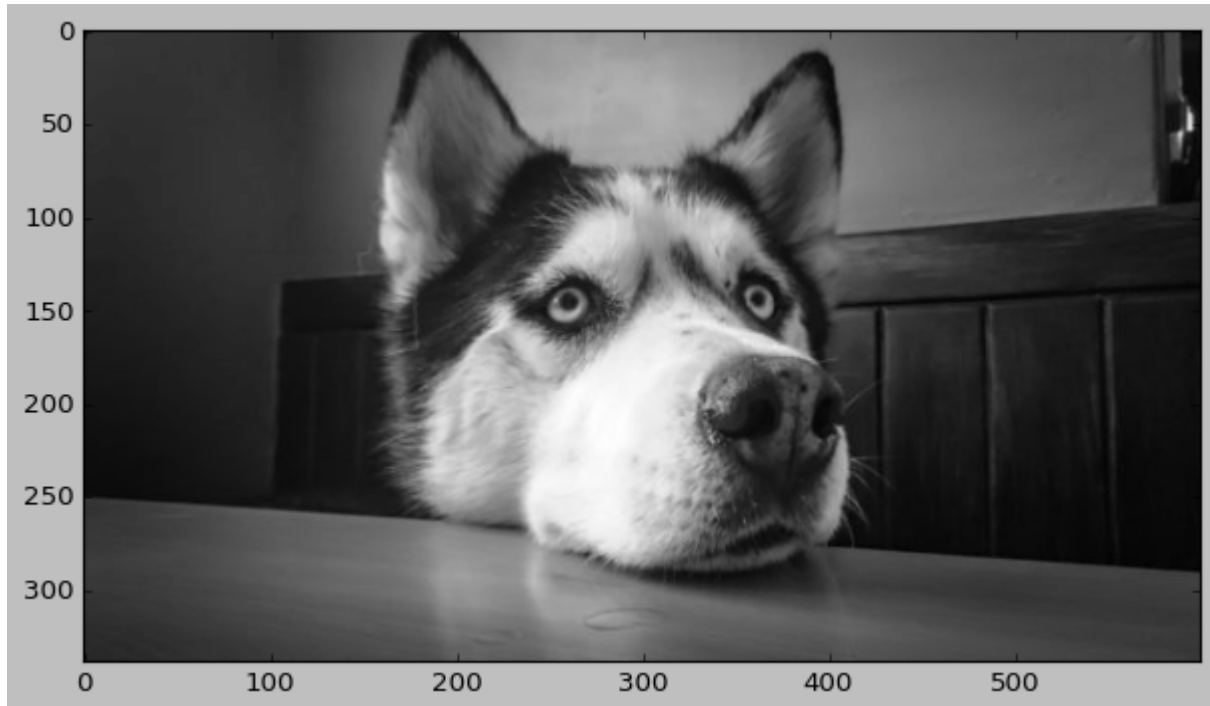
```
#-----
```

```
reconstimg = np.matrix(U[:, :nvalue])*np.diag(D[:nvalue])*np.matrix(V[:nvalue, :])
```

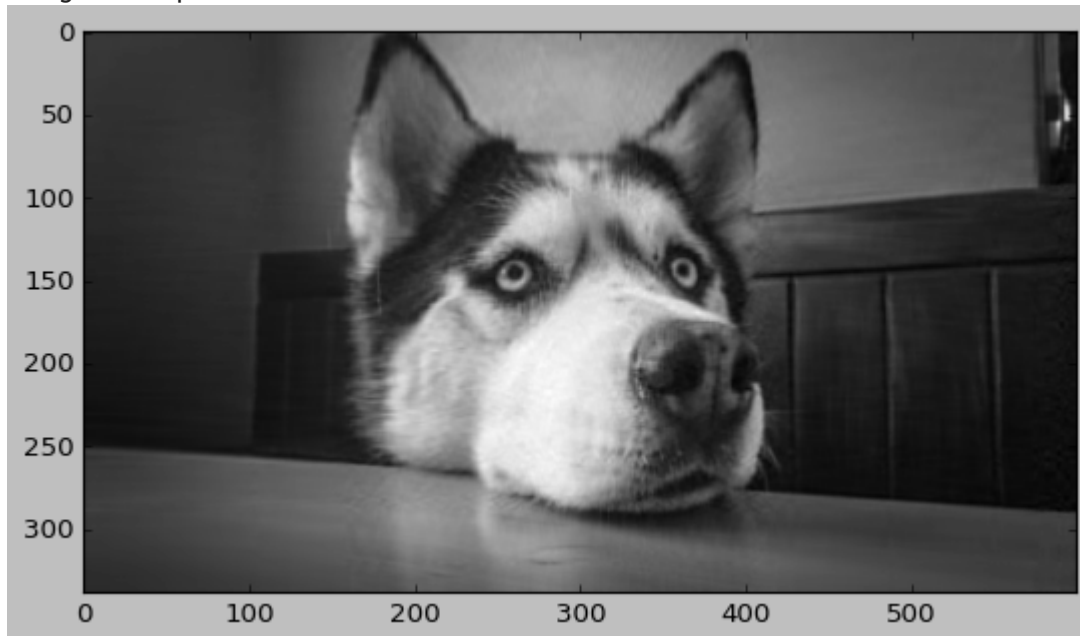
```
plt.imshow(reconstimg,cmap='gray')
```

```
print("Imagen comprimida")  
plt.show()
```

```
[60. 60. 60. ... 61. 62. 62.]
```



```
<PIL.Image.Image image mode=LA size=600x338 at 0x7F72A96157D0>  
imgmat (338, 600)  
U (338, 338)  
V (600, 600)  
Imagen comprimida
```



```
#imagen 3
```

```
plt.style.use('classic')  
url="https://t1.ea.ltmcdn.com/es/posts/1/6/2/10_curiosidades_del_golden_retriever_212  
img = Image.open(urllib.request.urlopen(url)).convert('LA')
```

```

imggray = img.convert('LA')
imgmat = np.array(list(imggray.getdata(band=0)),float)

print(imgmat)

imgmat.shape = (imggray.size[1],imggray.size[0])

plt.figure(figsize=(9,6))
plt.imshow(imgmat,cmap='gray')
plt.show()
print(img)
U,D,V = np.linalg.svd(imgmat)
print("imgmat",imgmat.shape)
print("U",U.shape)
print("V",V.shape)
nvalue = 50
#-----
reconstimg = np.matrix(U[:, :nvalue])*np.diag(D[:nvalue])*np.matrix(V[:nvalue, :])
plt.imshow(reconstimg,cmap='gray')
print("Imagen comprimida")
plt.show()

```



[124. 124. 124. ... 145. 139. 135.]



## ▼ Ejercicio 3

### Feature importances

Para este ejercicio, te pediremos que sigas el tutorial de la siguiente pagina:

<https://towardsdatascience.com/pca-clearly-explained-how-when-why-to-use-it-and-feature-importance-a-guide-in-python-7c274582c37e>



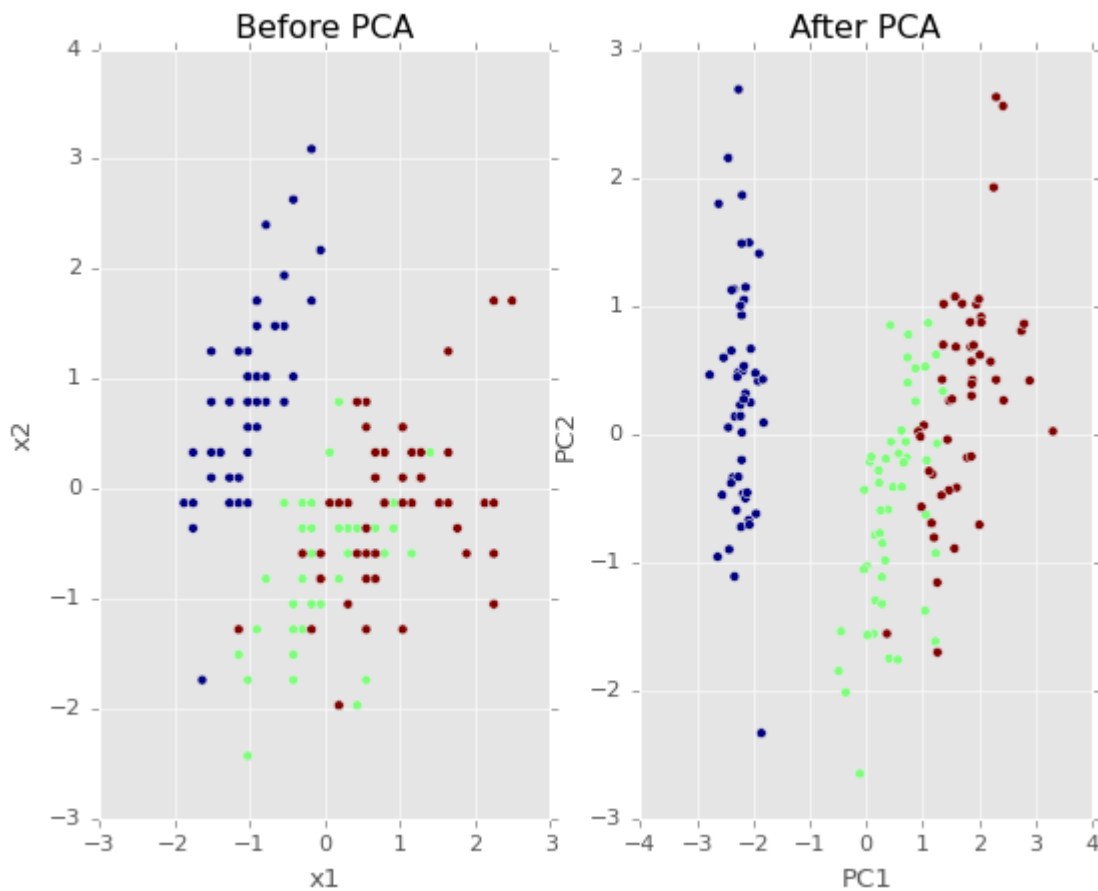
#tu codigo aqui

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA
import pandas as pd
from sklearn.preprocessing import StandardScaler
plt.style.use('ggplot')
# Load the data
iris = datasets.load_iris()
X = iris.data
y = iris.target
# Z-score the features
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)
# The PCA model
pca = PCA(n_components=2) # estimate only 2 PCs
X_new = pca.fit_transform(X) # project the original data into the PCA space
```

```

fig, axes = plt.subplots(1,2)
axes[0].scatter(X[:,0], X[:,1], c=y)
axes[0].set_xlabel('x1')
axes[0].set_ylabel('x2')
axes[0].set_title('Before PCA')
axes[1].scatter(X_new[:,0], X_new[:,1], c=y)
axes[1].set_xlabel('PC1')
axes[1].set_ylabel('PC2')
axes[1].set_title('After PCA')
plt.show()

```



```

print(pca.explained_variance_ratio_)

```

```

[0.72962445 0.22850762]

```

```

np.cov(X_new.T)

```

```

array([[2.93808505e+00, 5.33928780e-16],
       [5.33928780e-16, 9.20164904e-01]])

```

```

pca.explained_variance_

```



```

array([2.93808505, 0.9201649 ])

print(abs( pca.components_ ))

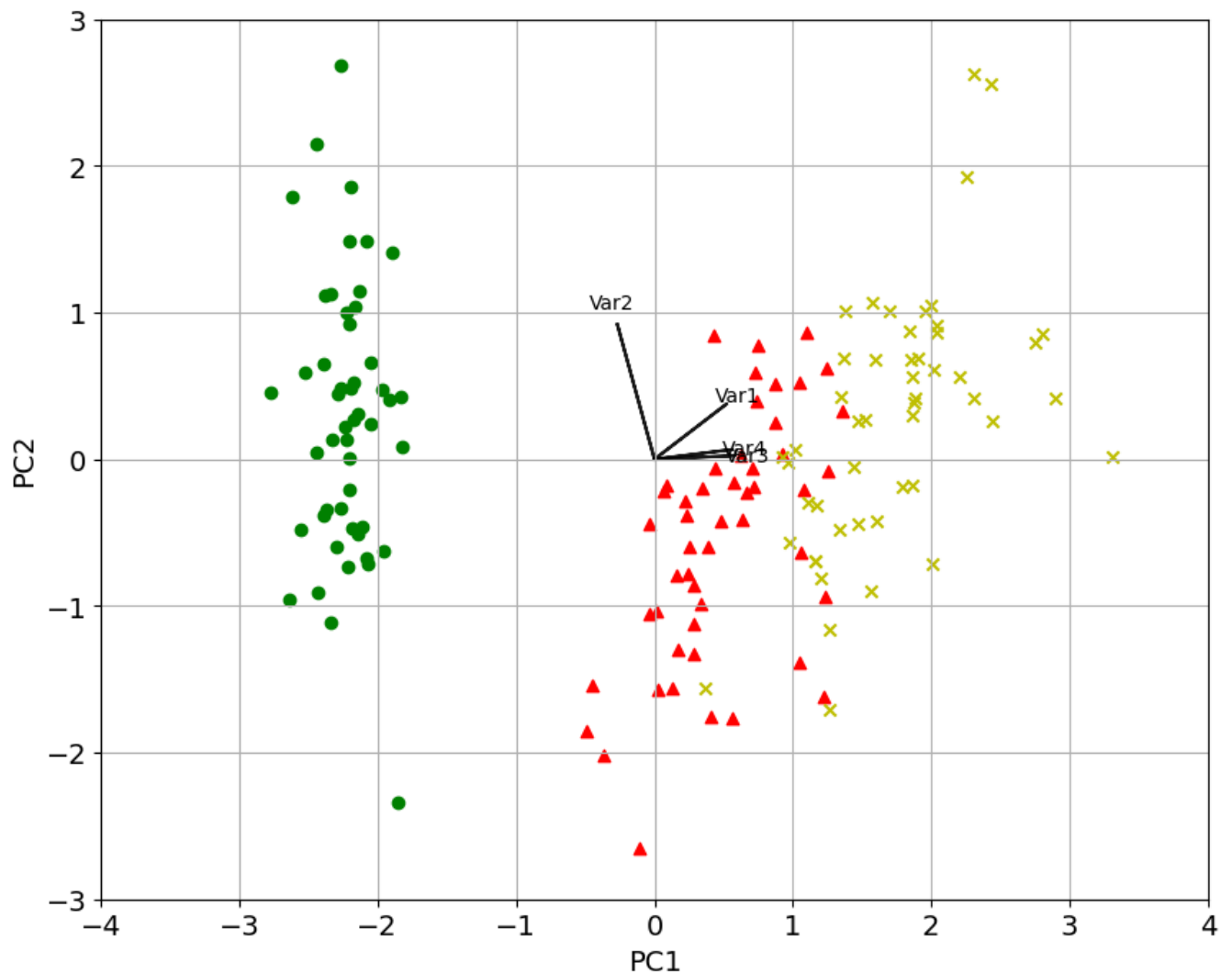
[[0.52106591 0.26934744 0.5804131  0.56485654]
 [0.37741762 0.92329566 0.02449161 0.06694199]]

def biplot(score, coeff , y):
    """
    Author: Serafeim Loukas, serafeim.loukas@epfl.ch
    Inputs:
        score: the projected data
        coeff: the eigenvectors (PCs)
        y: the class labels
    """
    xs = score[:,0] # projection on PC1
    ys = score[:,1] # projection on PC2
    n = coeff.shape[0] # number of variables
    plt.figure(figsize=(10,8), dpi=100)
    classes = np.unique(y)
    colors = ['g','r','y']
    markers=['o','^','x']
    for s,l in enumerate(classes):
        plt.scatter(xs[y==l],ys[y==l], c = colors[s], marker=markers[s]) # color base
    for i in range(n):
        #plot as arrows the variable scores (each variable has a score for PC1 and on
        plt.arrow(0, 0, coeff[i,0], coeff[i,1], color = 'k', alpha = 0.9,linestyle =
        plt.text(coeff[i,0]* 1.15, coeff[i,1] * 1.15, "Var"+str(i+1), color = 'k', ha

    plt.xlabel("PC{}".format(1), size=14)
    plt.ylabel("PC{}".format(2), size=14)
    limx= int(xs.max()) + 1
    limy= int(ys.max()) + 1
    plt.xlim([-limx,limx])
    plt.ylim([-limy,limy])
    plt.grid()
    plt.tick_params(axis='both', which='both', labelsize=14)

import matplotlib as mpl
mpl.rcParams.update(mpl.rcParamsDefault) # reset ggplot style
# Call the biplot function for only the first 2 PCs
biplot(X_new[:,0:2], np.transpose(pca.components_[0:2, :]), y)
plt.show()

```



```
# Var 3 and Var 4 are extremely positively correlated
np.corrcoef(X[:,2], X[:,3])[1,0]
```

```
0.9628654314027957
```

```
# Var 2and Var 3 are negatively correlated
np.corrcoef(X[:,1], X[:,2])[1,0]
```

```
-0.42844010433054014
```

Describe lo relevante del ejercicio y que descubriste de las variables analizadas.

### ¿Qué es feature importance y para que nos sirve?

Feature importance es una medida de que tan importante es una característica/feature dentro de un conjunto de datos para poder predecir su variable objetivo, cuanto más importante sea (o más

grande la magnitud del eigenvalor), más se usa en la predicción de la variable.

### **¿Qué hallazgos fueron los más relevantes durante el análisis del ejercicio?**

El hecho de verlo gráficamente lado a lado la entrada de datos y la salida del PCA me queda muy claro el resultado.

### **¿Dónde lo aplicarías o te sería de utilidad este conocimiento?**

Al tratar de generar modelos de predicción con conjuntos de datos que tienen muchas variables/features, para no hacer muy complejo el modelo y el costo/uso de CPU para procesarlo no sea tan alto, podemos usar esta técnica para discriminar las variables que no son tan importantes y que se reduzcan estos costos/tiempos de CPU.

[Colab paid products](#) - [Cancel contracts here](#)

✓ 0s completed at 8:04 PM

