

Haz doble clic (o ingresa) para editar

---

Bienvenido al notebook

Repaso de Reducción de dimensiones

---

**Tecnológico de Monterrey**

---

**Ciencia y Analitica de datos**

Profesora Titular:

- María de la Paz Rico Fernández

Profesor Tutor:

- Julio César Galindo López

Estudiante:

- Jaik Yocks Sandoval

Matrícula:

- A01793725
- 

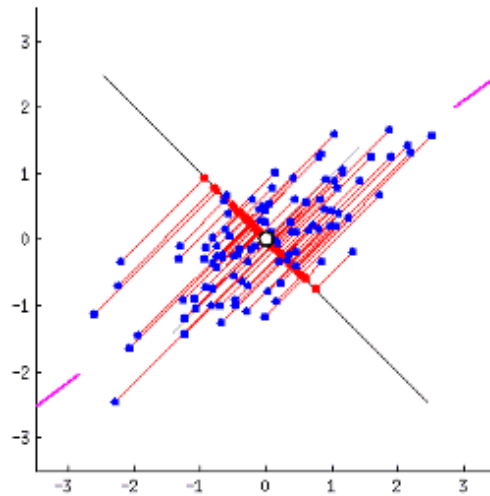
El objetivo es que entendamos de una manera visual, que es lo que pasa cuando nosotros seleccionamos cierto número de componentes principales o % de variabilidad de una base de datos.

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 :)

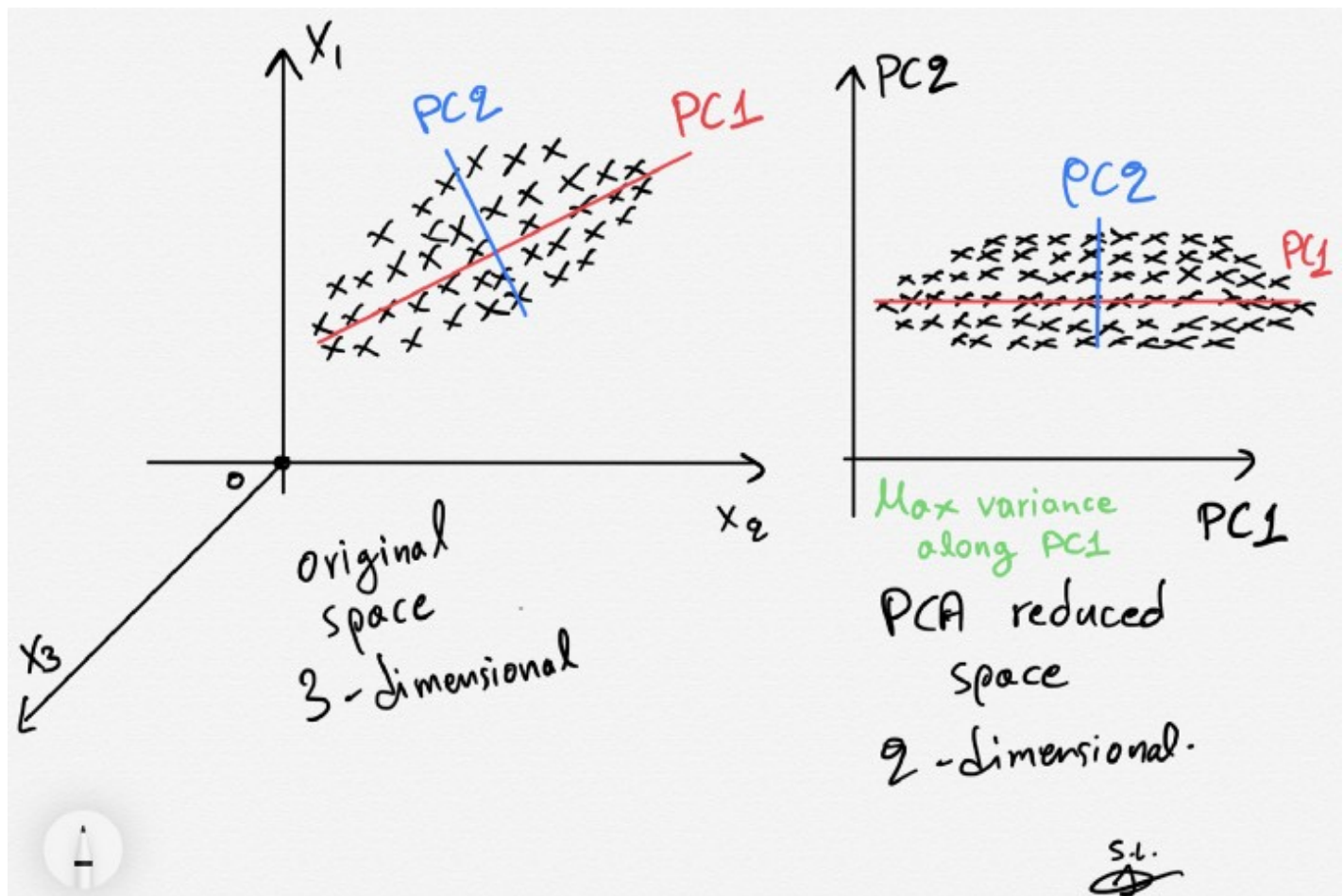
$$\mathbf{C} = \frac{\mathbf{X}^T \mathbf{X}}{n - 1}$$

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(values) #D
print(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=
print(B)
print("-----")

```

```

-----Matriz original-----
[[1 2 3]
 [4 5 6]
 [7 8 9]]
-----
[ 1.61168440e+01 -1.11684397e+00 -1.30367773e-15]
[[-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

# define la matriz
A1 = array([[3, 0, 2], [3, 0, -2], [0, 1, 1]])
print("-----Matriz original-----")
print(A1)
print("-----")
# calcula la eigendescomposición
values1, vectors1 = eig(A1)
print(values1) #D
print(vectors1) #W

#Reconstrucción

values1, vectors1 = np.linalg.eig(A1)

W1 = vectors1
Winv1 = np.linalg.inv(W1)
D1 = np.diag(values1)
#la matriz B tiene que dar igual a A

print("-----Matriz reconstruida-----")
# Realiza la reconstruccion de B=W*D*Winv, te da lo mismo de A? No, exactamente!!!!

#TU CODIGO AQUI-----

B11 = np.dot(W1,D1)

B21 = np.dot(B11,Winv1)

B1 = np rint(B21)

print(B1)
print("-----")

-----Matriz original-----
[[ 3  0  2]
 [ 3  0 -2]
 [ 0  1  1]]
-----
[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 reconstruida-----
[[ 3.+0.j  0.-0.j  2.-0.j]
 [ 3.-0.j  0.+0.j -2.+0.j]
 [ 0.+0.j  1.-0.j  1.+0.j]]
-----

```

```

#Matriz 2

# define la matriz
A2 = array([[1, 3, 8], [2, 0, 0], [0, 0, 1]])
print("-----Matriz original-----")
print(A2)
print("-----")
# calcula la eigendescomposición
values2, vectors2 = eig(A2)
print(values2) #D
print(vectors2) #W

#Reconstrucción

values2, vectors2 = np.linalg.eig(A2)

W2 = vectors2
Winv2 = np.linalg.inv(W2)
D2 = np.diag(values2)
#la matriz B tiene que dar igual a A

print("-----Matriz reconstruida-----")
# Realiza la reconstruccion de B=W*D*Winv, te da lo mismo de A? No, exactamente!!!!

#TU CODIGO AQUI-----

B12 = np.dot(W2,D2)

B22 = np.dot(B12,Winv2)

B2 = np rint(B22)

print(B2)
print("-----")

-----Matriz original-----
[[1 3 8]
 [2 0 0]
 [0 0 1]]
-----
[ 3. -2.  1.]
[[ 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
A3 = array([[5, 4, 0], [1, 0, 1], [10, 7, 1]])

```

```

print("-----Matriz original-----")
print(A3)
print("-----")
# calcula la eigendescomposición
values3, vectors3 = eig(A3)
print(values3) #D
print(vectors3) #W

#Reconstrucción

values3, vectors3 = np.linalg.eig(A3)

W3 = vectors3
Winv3 = np.linalg.inv(W3)
D3 = np.diag(values3)
#la matriz B tiene que dar igual a A

print("-----Matriz reconstruida-----")
# Realiza la reconstruccion de B=W*D*Winv, te da lo mismo de A? No, exactamente!!!!

#TU CODIGO AQUI-----

B13 = np.dot(W3,D3)

B23 = np.dot(B13,Winv3)

B3 = np rint(B23)

print(B3)
print("-----")

-----Matriz original-----
[[ 5  4  0]
 [ 1  0  1]
 [10  7  1]]
-----
[[ 6.89167094 -0.214175  -0.67749594]
 [ 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.

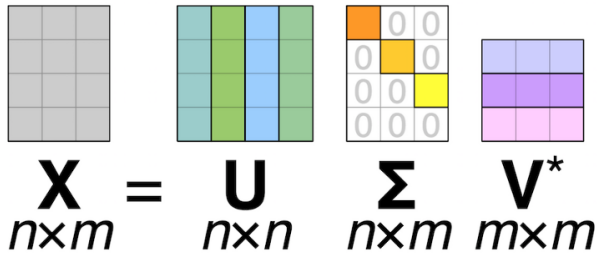
Es decir: Unidades-PC PC-Unidades

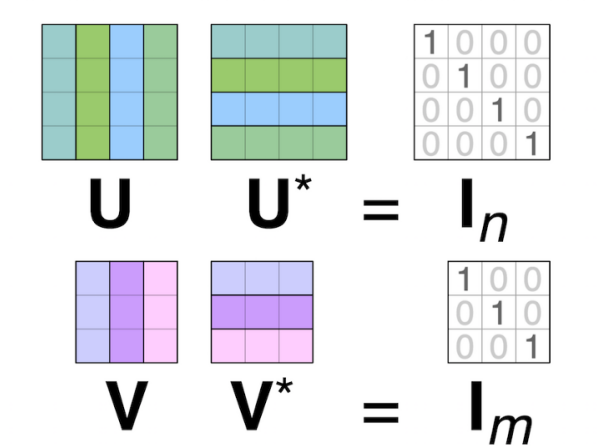
Veámoslo gráficamente, ¿qué pasa con esa selección de los PCs y su efecto?.

Para ello usaremos Singular Value Descomposition (SVD).

## Singular Value Descomposition(SVD)

Es otra descomposición que también nos ayudara a reducir dimensiones.



$$\begin{matrix}
 \begin{matrix} \text{4x4} \\ \text{X} \end{matrix} & = & \begin{matrix} \text{4x4} \\ \text{U} \end{matrix} & \begin{matrix} \text{4x4} \\ \Sigma \end{matrix} & \begin{matrix} \text{4x4} \\ \text{V}^* \end{matrix} \\
 n \times m & & n \times n & n \times m & m \times m
 \end{matrix}$$
  


$$\begin{matrix}
 \begin{matrix} \text{4x4} \\ \text{U} \end{matrix} & \begin{matrix} \text{4x4} \\ \text{U}^* \end{matrix} & = & \begin{matrix} \text{4x4} \\ \text{I}_n \end{matrix} \\
 \begin{matrix} \text{4x4} \\ \text{V} \end{matrix} & \begin{matrix} \text{4x4} \\ \text{V}^* \end{matrix} & = & \begin{matrix} \text{4x4} \\ \text{I}_m \end{matrix}
 \end{matrix}$$

### ▼ 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 imágenes** :o

Haz doble clic (o ingresa) para editar

```

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/uploads/
#img = Image.open('lucy.jpg')

```



```

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

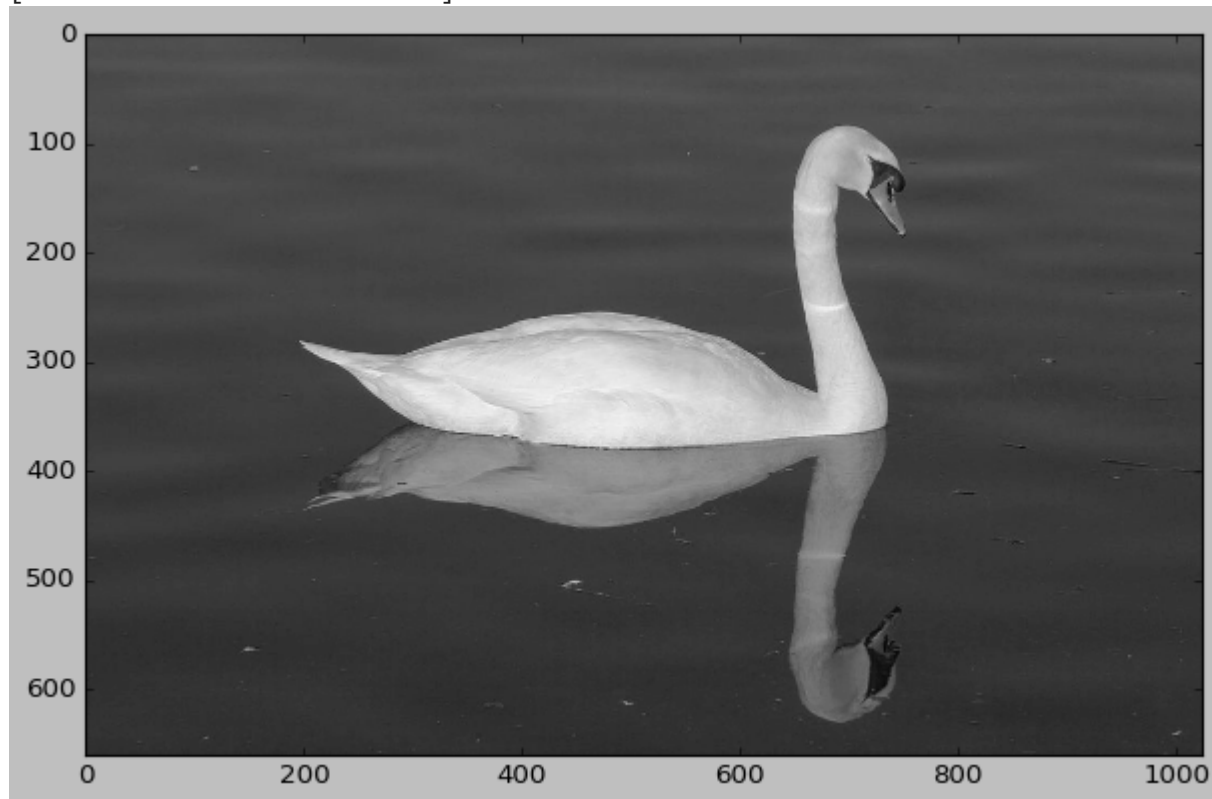
print(imgmat)

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 0x7FE9EAA47950>
```

```

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

```

```
(660, 1024)
```

```
U.shape
```

```
(660, 660)
```

```
V.shape
```

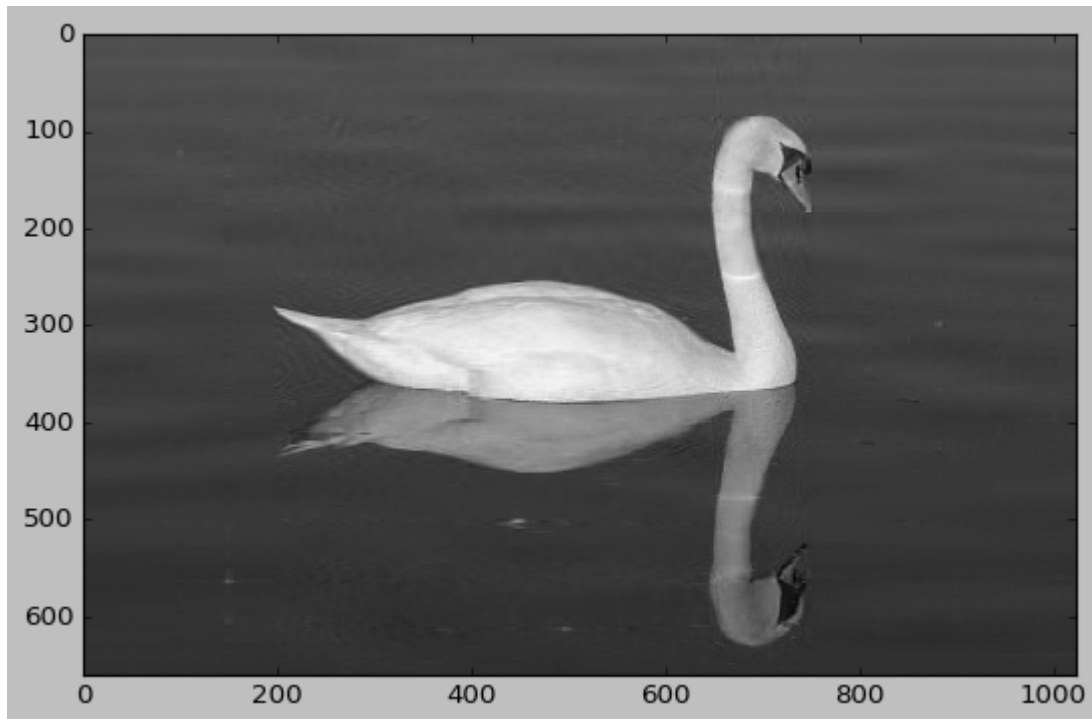
```
(1024, 1024)
```

```

#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 en
# 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 = 50
#-----
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')
img = Image.open(urllib.request.urlopen('https://st4.depositphotos.com/12985848/23162/i/1600/'))
#img = Image.open('imagen1.jpg')
imggray = img.convert('LA')
imgmat1 = np.array(list(imggray.getdata(band=0)), float)

```

```

print(imgmat1)

imgmat1.shape = (imggray.size[1],imggray.size[0])

plt.figure(figsize=(9,6))
plt.imshow(imgmat1,cmap='gray')
plt.show()
print(img)

U,D,V = np.linalg.svd(imgmat1)
imgmat1.shape

nvalue = 100
#-----
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")

```

```
[227. 243. 244. ... 50. 50. 50.]
```



```
U,D,V = np.linalg.svd(imgmat1)
imgmat1.shape
```

```
(1168, 1600)
```



```
U.shape
```

```
(1168, 1168)
```



```
V.shape
```

```
(1600, 1600)
```



```
#imagen 2
plt.style.use('classic')
img = Image.open(urllib.request.urlopen('https://st4.depositphotos.com/12982378/21946/i/1600/'))
#img = Image.open('imagen1.jpg')
imggray = img.convert('LA')
imgmat2 = np.array(list(imggray.getdata(band=0)),float)

print(imgmat2)

imgmat2.shape = (imggray.size[1],imggray.size[0])

plt.figure(figsize=(9,6))
plt.imshow(imgmat2,cmap='gray')
plt.show()
print(img)
```

```
U,D,V = np.linalg.svd(imgmat2)
imgmat2.shape

nvalue = 50
#-----
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")
```

```
[162. 163. 163. ... 50. 50. 50.]
```



```
U,D,V = np.linalg.svd(imgmat2)
imgmat2.shape
```



```
U.shape
```

```
(1168, 1168)
```



```
V.shape
```

```
(1600, 1600)
```

```
⌋DTI Image Image image mode=LA size=1600x1168 at 0x7EE9E8CE2210\
```

```
#imagen 3
plt.style.use('classic')
img = Image.open(urllib.request.urlopen('https://st4.depositphotos.com/13194036/21516/i/1600/'))
#img = Image.open('imagen1.jpg')
imggray = img.convert('LA')
imgmat3 = np.array(list(imggray.getdata(band=0)),float)

print(imgmat3)
```

```
imgmat3.shape = (imggray.size[1],imggray.size[0])
```

```
plt.figure(figsize=(9,6))
plt.imshow(imgmat3,cmap='gray')
plt.show()
print(img)
```

```
U,D,V = np.linalg.svd(imgmat3)
imgmat3.shape
```

```
nvalue = 10
```

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

```
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)
```

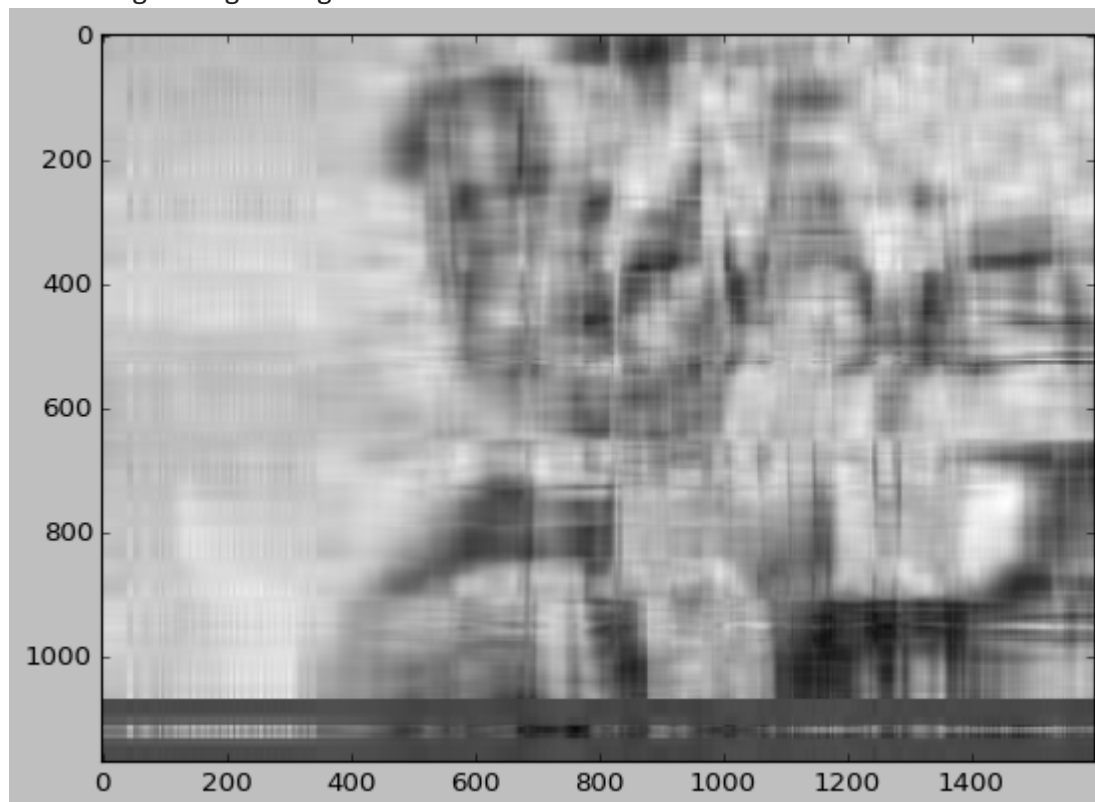
```
#=U(660X50)(50X50)(50X1024)
```

```
plt.imshow(reconstimg,cmap='gray')
plt.show()
print("Felicidades la imagen está comprimida")
```

```
[189. 189. 189. ... 50. 50. 50.]
```



```
<PIL.Image.Image image mode=LA size=1600x1168 at 0x7FE9E8DAE650>
```



```
Felicidades la imagen está comprimida
```

```
U,D,V = np.linalg.svd(imgmat3)
imgmat3.shape
```

```
(1168, 1600)
```

```
U.shape
```

```
(1168, 1168)
```

```
V.shape
```

```
(1600, 1600)
```

## ▼ 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>

```
import numpy as np
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

# Se grafica antes y después de aplicar la transformación PCA transform y se usa un color par

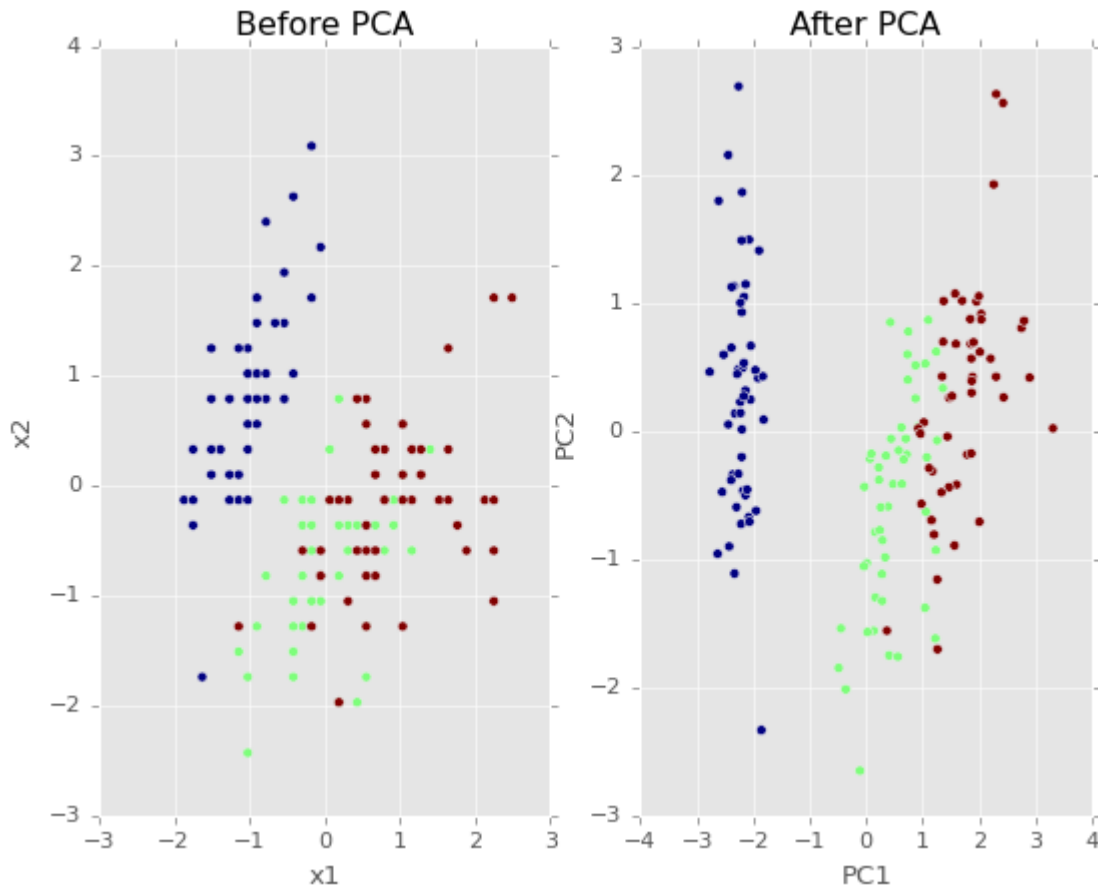
fig, axes = plt.subplots(1,2)
axes[0].scatter(X[:,0], X[:,1], c=y)
axes[0].set_xlabel('x1')
```



```

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_)
# array([0.72962445, 0.22850762])

```

```
[0.72962445 0.22850762]
```

# La prueba de la varianza máxima se puede también ver estimando la covarianza de la matriz d

```

np.cov(X_new.T)
array([[2.93808505e+00, 4.83198016e-16],
       [4.83198016e-16, 9.20164904e-01]])

array([[2.93808505e+00, 4.83198016e-16],
       [4.83198016e-16, 9.20164904e-01]])

```

# Se observa que estos valores (en la diagonal que tienen las varianzas) son iguales a los va

```
pca.explained_variance_
array([2.93808505, 0.9201649 ])
```

```
array([2.93808505, 0.9201649 ])
```

#Se buscan las features mas importantes:

```
print(abs( pca.components_ ))
```

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

# Biplot es la mejor manera de visualizar todo en uno siguiendo el analisis PCA.

```
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 based on gro
```

```
for i in range(n):
```

```
    #plot as arrows the variable scores (each variable has a score for PC1 and one for PC
```

```
    plt.arrow(0, 0, coeff[i,0], coeff[i,1], color = 'k', alpha = 0.9,linestyle = '-',line
```

```
    plt.text(coeff[i,0]* 1.15, coeff[i,1] * 1.15, "Var"+str(i+1), color = 'k', ha = 'cent
```

```
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])
```

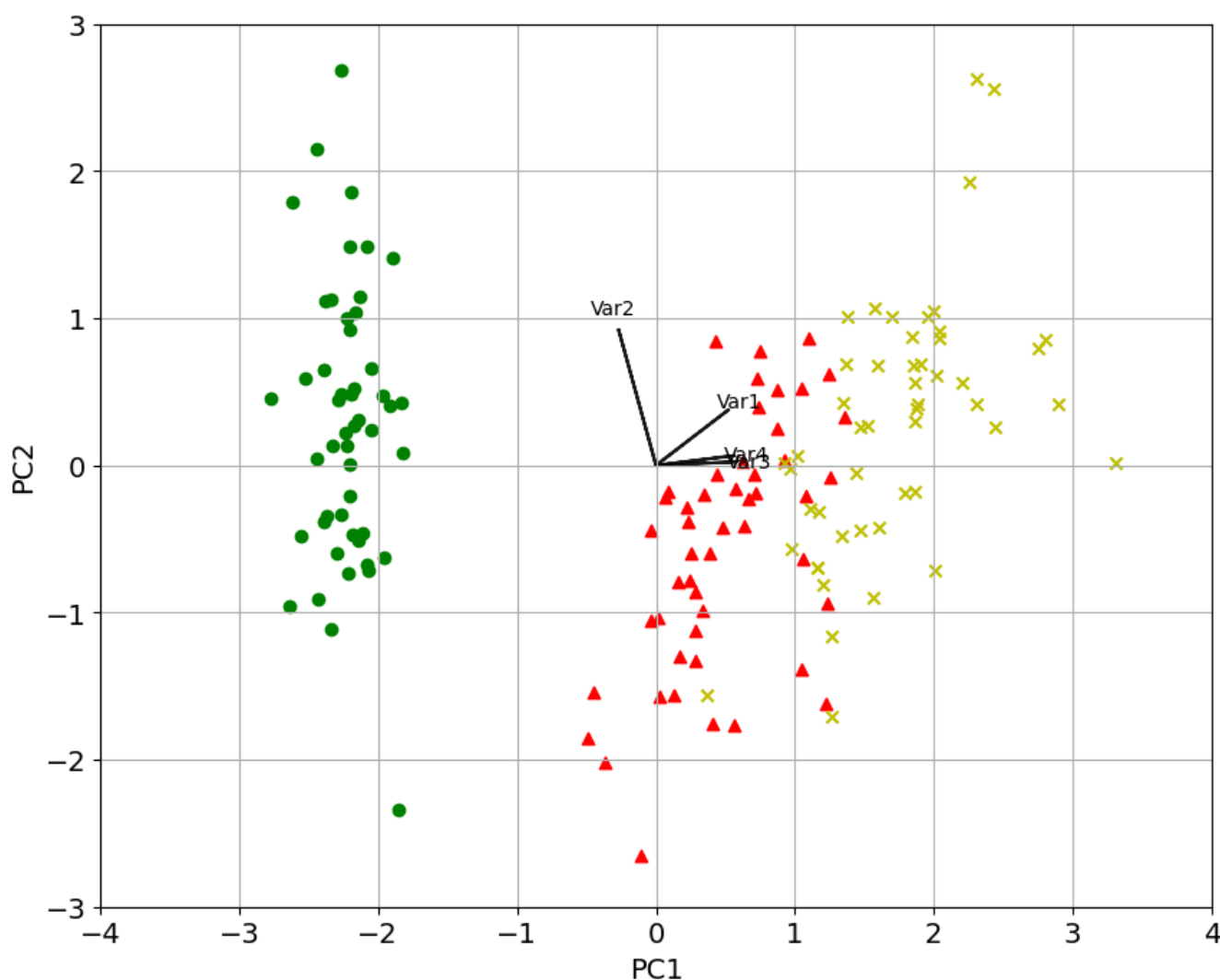
```
plt.grid()
```

```
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()
```



```
# Se verifica el código
```

```
# Var 3 and Var 4 are extremely positively correlated
np.corrcoef(X[:,2], X[:,3])[1,0]
0.9628654314027957
# Var 2 and Var 3 are negatively correlated
np.corrcoef(X[:,1], X[:,2])[1,0]
-0.42844010433054014
```

-0.42844010433054014

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

¿Qué es feature importance y para que nos sirve? La importancia de cada feature se refleja a través de la magnitud de cada valor correspondiente en los vectores propios. Lo que se entiende que a mayor magnitud, mayor importancia.

Si se miran los valores absolutos de los vectores propios los componentes corresponden los k mas grandes de los valores propios. En este ejercicio los componentes fueron seleccionados para explicar la varianza por ejemplo.

Los valores mayores son los valores absolutos, que contribuyen con un feature específico en el componente principal.

¿Qué hallazgos fueron los más relevantes durante el análisis del ejercicio? Se aprecia que el espacio de PCA; la varianza es maximizada en lo largo PC1, lo que explica el 73% de la varianza y PC2 explica el 22% de la varianza. Así ambos componentes explican el 95% de la varianza.

Se puede verificar visualmente que la varianza es maximizada y que los features 1, 3 y 4 son los mas importantes para PC1. De manera similar, el feature 2 es el mas importante para el PC2.


Adicionalmente, la flechas de las variables y features dentro del punto en la misma dirección indican la correlación entre las variables que son representadas hacia cualquier lugar. Las puntas de las flechas que están en direcciones opuestas indican un contraste entre las variables que representan.

¿Dónde lo aplicarías o te sería de utilidad este conocimiento? La técnica PCA es particularmente útil en el procesamiento de datos donde existe la multicolinealidad entre los features y las variables.

Se puede utilizar PCA cuando las dimensiones de los features de entrada son altas, por ejemplo muchas variables.

PCA puede también ser utilizado para eliminar ruido de una señal o comprimir datos.

Productos pagados de Colab - [Cancela los contratos aquí](#)

 0 s se ejecutó 19:22