

Actividad Semanal 5



TC4029 Ciencia y analítica de datos (Gpo 10)

Alumno: Armando Bringas Corpus (A01200230)

Profesores: Dra. María de la Paz Rico Fernández, Mtra. Victoria Guerrero Orozco

Fecha: 25 de octubre de 2022

Bienvenido al notebook

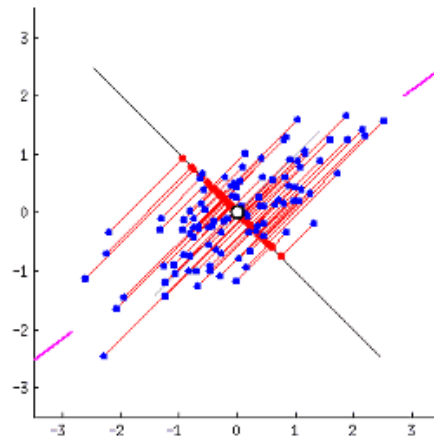
Repaso de Reducción de dimensiones

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 :)

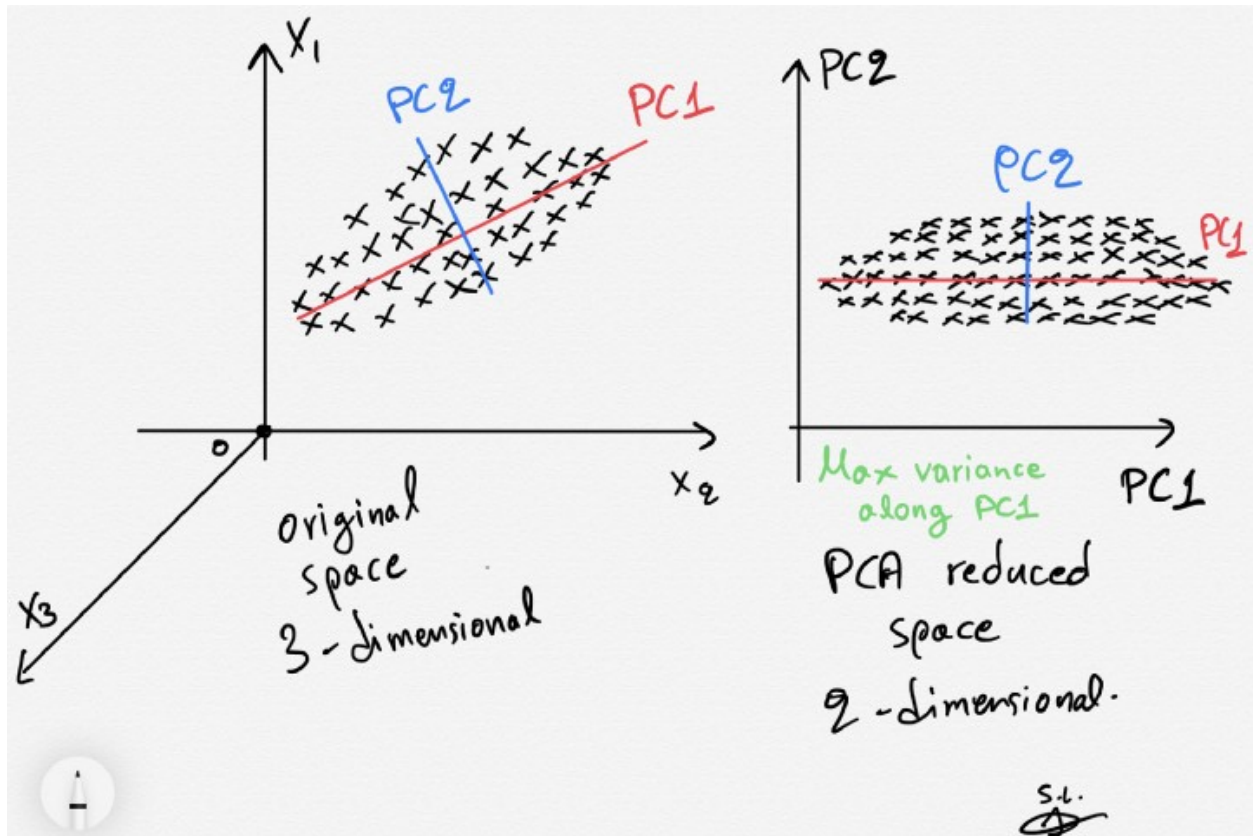
$$\mathbf{C} = \frac{\mathbf{X}^\top \mathbf{X}}{n - 1}$$

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

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

1. Para la reducci3n de dimensiones vamos a seleccionar k vectores de W y proyectaremos nuestros datos.

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



Ejercicio 1, Descomposici3n y composici3n

Descomposici3n

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 traves de las matrices $\mathbf{W} \mathbf{D} \mathbf{W}^{-1}$ (OJO. Esto es lo mismo de la ecuaci3n 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)

# Cálculo de la eigendecomposición
print("-----Eigen valores y Eigen vectores-----")
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=W.dot(D).dot(Winv)
print(B)
print("-----")

-----Matriz original-----
[[1 2 3]
 [4 5 6]
 [7 8 9]]
-----Eigen valores y Eigen vectores-----
[ 1.61168440e+01 -1.11684397e+00 -9.75918483e-16]
[[-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.]]
-----

Matriz 1
```

$$A = \begin{pmatrix} 3, 0, 2 \\ 3, 0, -2 \\ 0, 1, 1 \end{pmatrix}$$

```
#Matriz 1
```

```

# define la matriz
A = array([[3, 0, 2], [3, 0, -2], [0, 1, 1]])
print("-----Matriz original-----")
print(A)

# Cálculo de la eigendescomposición
print("-----Eigen valores y Eigen vectores-----")
values, vectors = eig(A)
print(values) #D
print(vectors) #W

# Reconstrucción de la matriz
values, vectors = np.linalg.eig(A)
W = vectors
Winv = np.linalg.inv(W)
D = np.diag(values)
print("-----Matriz reconstruida-----")
B=np.real(np rint(W.dot(D).dot(Winv)))
print(B)
print("-----")

-----Matriz original-----
[[ 3  0  2]
 [ 3  0 -2]
 [ 0  1  1]]
-----Eigen valores y Eigen vectores-----
[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.  2.]
 [ 3.  0. -2.]
 [ 0.  1.  1.]]
-----

Matriz 2

```

$$A2 = \begin{pmatrix} 1, 3, 8 \\ 2, 0, 0 \\ 0, 0, 1 \end{pmatrix}$$

```

#Matriz 2

# define la matriz
A2 = array([[1, 3, 8], [2, 0, 0], [0, 0, 1]])
print("-----Matriz original-----")
print(A2)
print("-----")

# Cálculo de la eigendescomposición
print("-----Eigen valores y Eigen vectores-----")
values, vectors = eig(A2)
print(values) #D
print(vectors) #W

```

```

# Reconstrucción de la matriz
values, vectors = np.linalg.eig(A2)
W = vectors
Winv = np.linalg.inv(W)
D = np.diag(values)
print("-----Matriz reconstruida-----")
B2=np.real(np rint(W.dot(D).dot(Winv)))
print(B2)
print("-----")

-----Matriz original-----
[[1 3 8]
 [2 0 0]
 [0 0 1]]
-----

-----Eigen valores y Eigen vectores-----
[ 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

$$A3 = \begin{pmatrix} 5, 4, 0 \\ 1, 0, 1 \\ 10, 7, 1 \end{pmatrix}$$

```

#Matriz 3

# define la matriz
A3 = array([[5, 4, 0], [1, 0, 1], [10, 7, 1]])
print("-----Matriz original-----")
print(A3)
print("-----")

# calcula la eigendescomposición
values, vectors = eig(A3)
print(values) #D
print(vectors) #W

# Reconstrucción de la matriz
values, vectors = np.linalg.eig(A3)
W = vectors
Winv = np.linalg.inv(W)
D = np.diag(values)
print("-----Matriz reconstruida-----")
B3=np.real(np rint(W.dot(D).dot(Winv)))
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 Decomposition (SVD).

Singular Value Decomposition(SVD)

Es otra descomposición que también nos ayudara a reducir dimensiones.

#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

```

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/2015/03/Ci.
#img = Image.open('lucy.jpg')
imggray = img.convert('LA')
imgmat = np.array(list(imggray.getdata(band=0)),float)

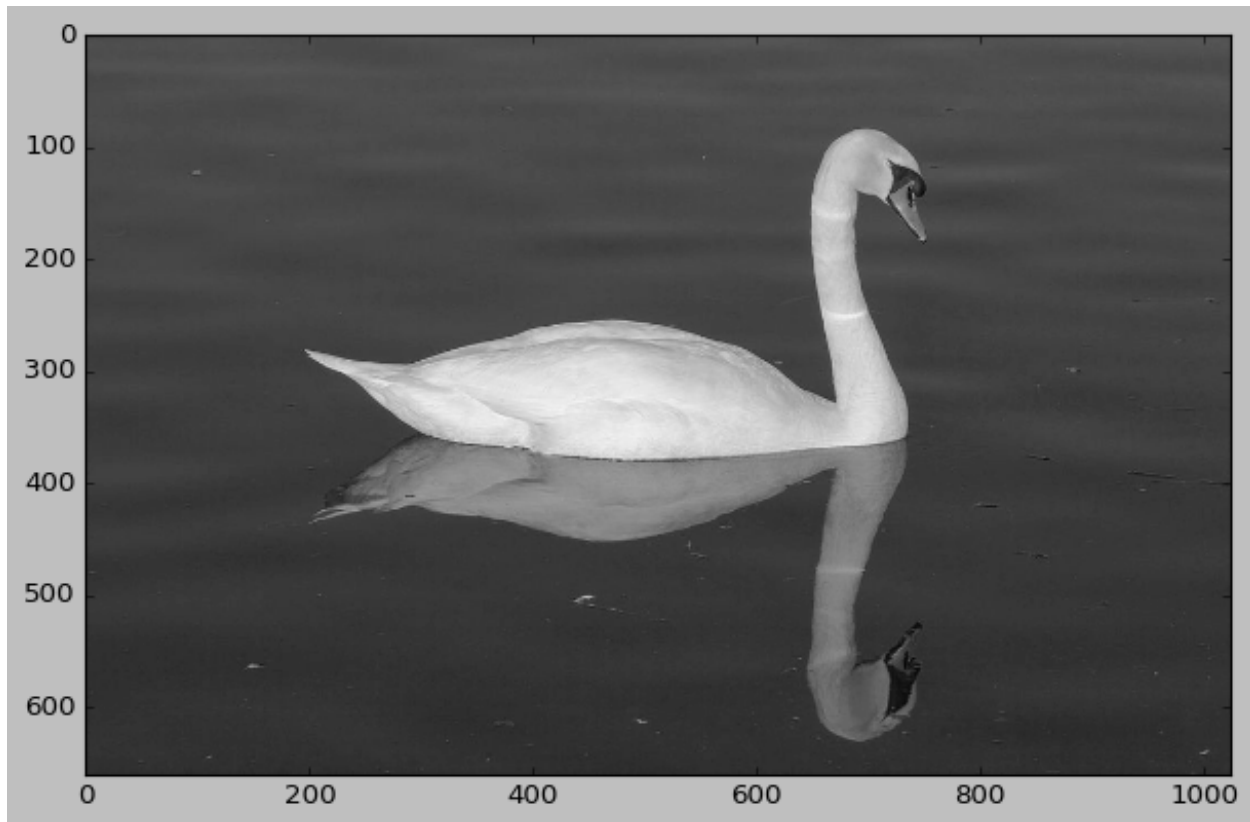
print(imgmat)

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 0x202654F6AA0>
```

```
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, :])
```

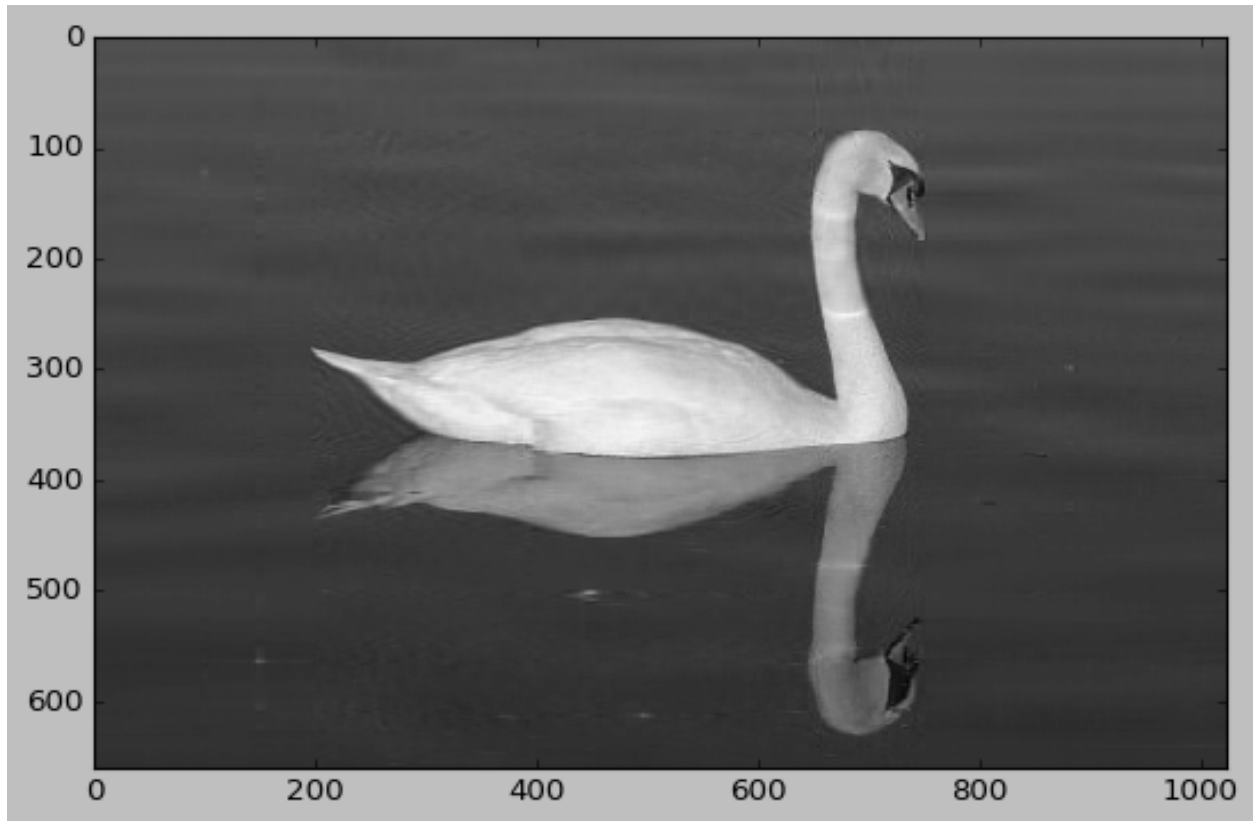
```
#ve las dimensiones de la imagen y su descomposicion
```

```
#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 imágenes

Imagen 1

Generada a través de DALL·E con la descripción: "Albert Camus and Hannah Arendt in 1977 having coffee and a depth discussion about existentialism, absurdism and the human condition".

Link a la imagen

#imagen 1

```
# Procesamiento de la imagen, conversión a escala de grises y a una matriz de numpy
img = Image.open('img/DALL·E 2022-10-22 12.56.40.png')
imggray = img.convert('LA')
imgmat = np.array(list(imggray.getdata(band=0)),float)
```

```
# Matriz de la imagen
print(imgmat)
```

```
# Ploteo de la imagen
imgmat.shape = (imggray.size[1],imggray.size[0])
plt.style.use('classic')
plt.imshow(imgmat, cmap='gray')
plt.show()
```



```

print(img)

# Computación del SVD,
U,D,V = np.linalg.svd(imgmat)

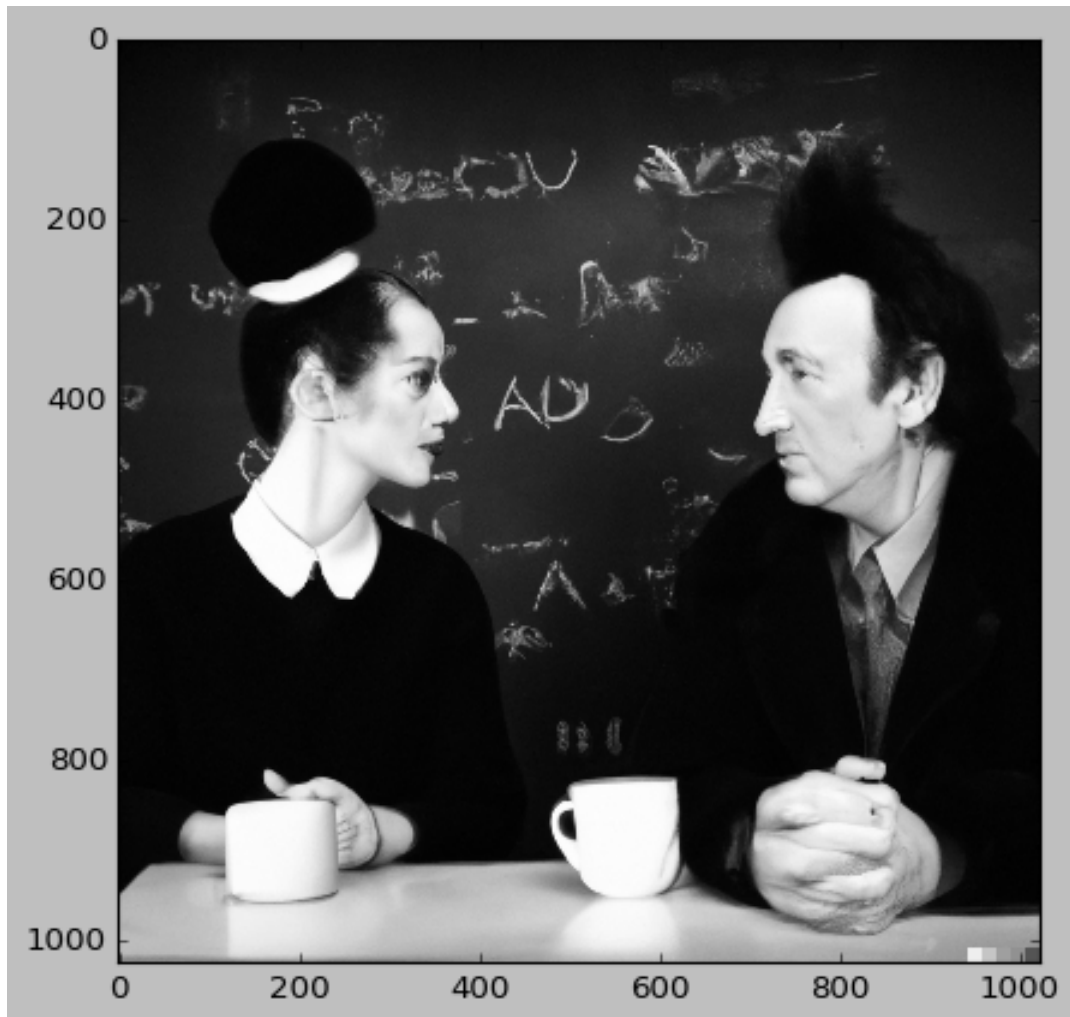
# Descripción de parámetros de la imagen
print(imgmat.shape)
print(U.shape)
print(V.shape)

# Factor de aproximación de 'n' a 'U, V'
nvalue = 70

# Reconstrucción de la imagen y ploteo de la imagen reconstruida
reconstimg = np.matrix(U[:, :nvalue])*np.diag(D[:nvalue])*np.matrix(V[:nvalue, :])
plt.title(f'n = {nvalue}')
plt.imshow(reconstimg, cmap='gray')
plt.show()

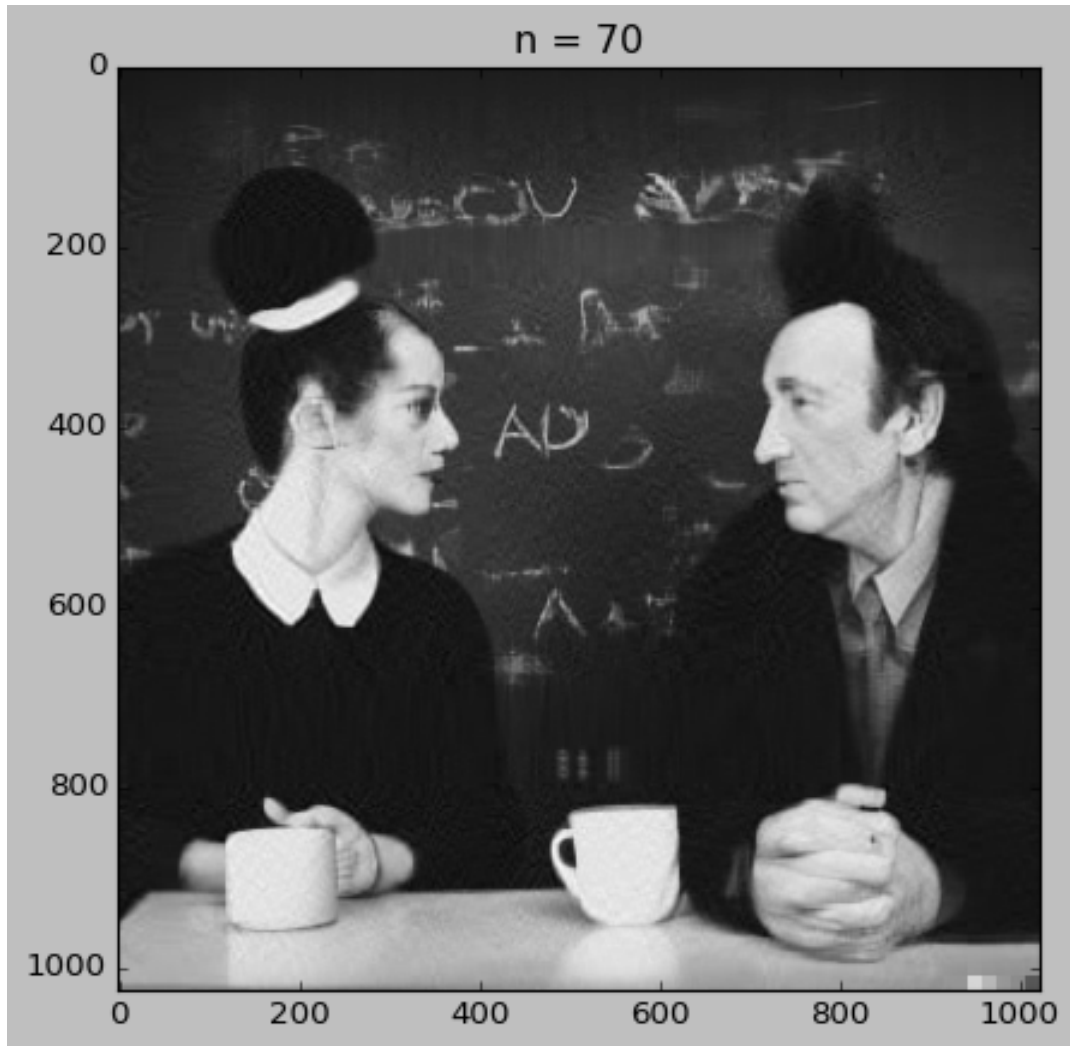
[12. 14. 14. ... 88. 88. 88.]

```



```
<PIL.PngImagePlugin.PngImageFile image mode=RGB size=1024x1024 at 0x20265BAD390>
```

```
(1024, 1024)
(1024, 1024)
(1024, 1024)
```



Con $n = 70$ la imagen comprimida es se aproxima a la imagen original

Imagen 2

La propaganda de Rodchenko

#imagen 2

Procesamiento de la imagen, conversión a escala de grises y a una matriz de numpy

```
img = Image.open(urllib.request.urlopen('https://historia-arte.com/_/eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1Ni.
imggray = img.convert('LA')
imgmat = np.array(list(imggray.getdata(band=0)),float)
```

Matriz de la imagen

```
print(imgmat)
```

Ploteo de la imagen

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

```

plt.style.use('classic')
plt.imshow(imgmat, cmap='gray')
plt.show()
print(img)

# Computación del SVD
U,D,V = np.linalg.svd(imgmat)

# Descripción de parámetros de la imagen
print(imgmat.shape)
print(U.shape)
print(V.shape)

# factor de aproximación de 'n' a 'U, V'
nvalue = 100

# Descripción de parámetros de la imagen
print(imgmat.shape)
print(U.shape)
print(V.shape)

# Reconstrucción de la imagen y ploteo de la imagen reconstruida
reconstimg = np.matrix(U[:, :nvalue])*np.diag(D[:nvalue])*np.matrix(V[:nvalue, :])
plt.title(f'n = {nvalue}')
plt.imshow(reconstimg, cmap='gray')
plt.show()

[251. 251. 251. ... 255. 255. 255.]

```



<PIL.WebImagePlugin.WebImageFile image mode=RGB size=1500x1060 at 0x20265DD8790>

(1060, 1500)

(1060, 1060)

(1500, 1500)

(1060, 1500)

(1060, 1060)

(1500, 1500)



Con $n = 100$ la imagen comprimida es se aproxima a la imagen original

Imagen 3

Beso V de Roy Lichenstein

#imagen 3

Procesamiento de la imagen, conversión a escala de grises y a una matriz de numpy

```
img = Image.open(urllib.request.urlopen('https://historia-arte.com/_/eyJOeXAI0iJKV1QiLCJhbGciOiJIUzI1Ni.
imggray = img.convert('LA')
imgmat = np.array(list(imggray.getdata(band=0)),float)
```

Matriz de la imagen

```
print(imgmat)
```

Ploteo de la imagen

```
imgmat.shape = (imggray.size[1],imggray.size[0])
plt.style.use('classic')
plt.imshow(imgmat,cmap='gray')
plt.show()
print(img)
```

Computación del SVD

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

```

# Descripción de parámetros de la imagen
print(imgmat.shape)
print(U.shape)
print(V.shape)

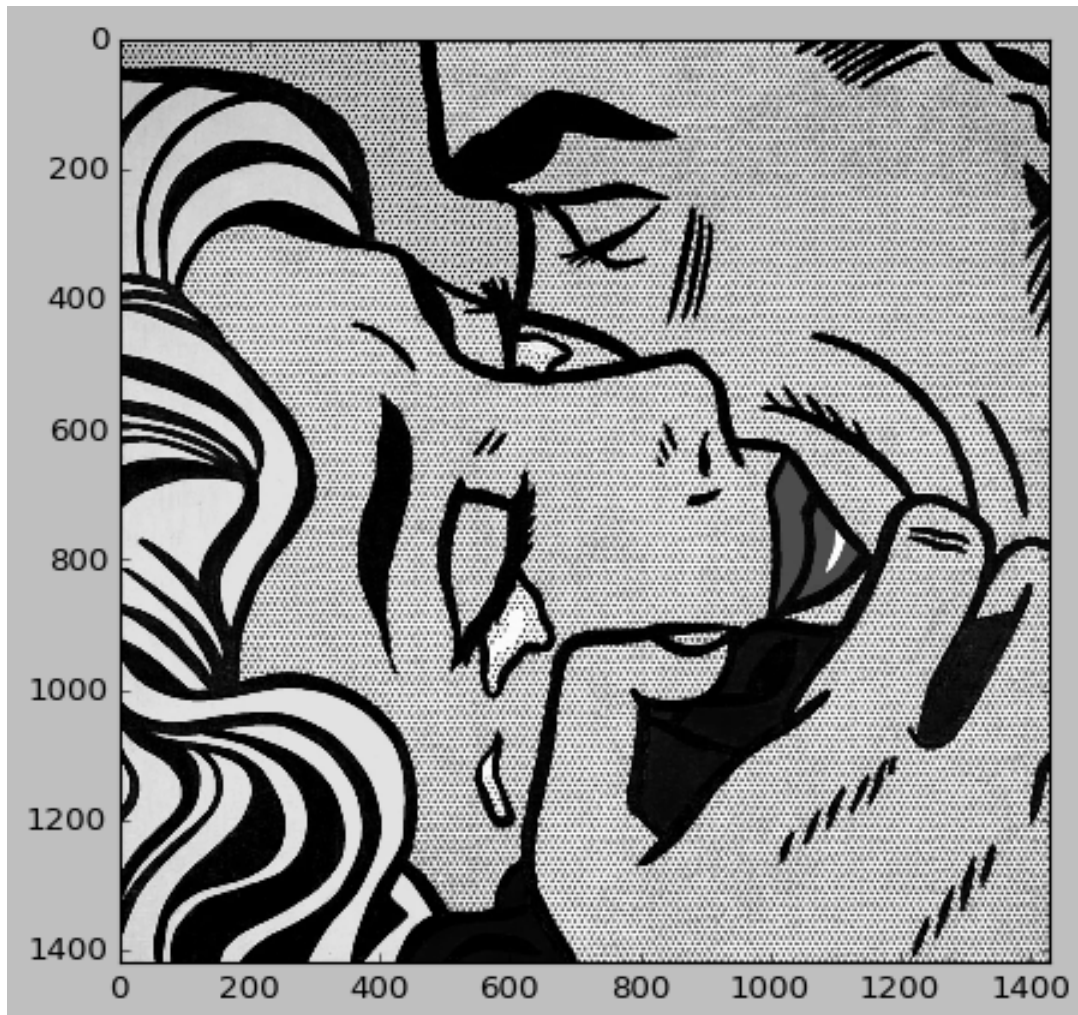
# factor de aproximación de 'n' a 'U, V'
nvalue = 100

# Descripción de parámetros de la imagen
print(imgmat.shape)
print(U.shape)
print(V.shape)

# Reconstrucción de la imagen y ploteo de la imagen reconstruida
reconsting = np.matrix(U[:, :nvalue])*np.diag(D[:, :nvalue])*np.matrix(V[:, :nvalue, :])
plt.title(f'n = {nvalue}')
plt.imshow(reconsting, cmap='gray')
plt.show()

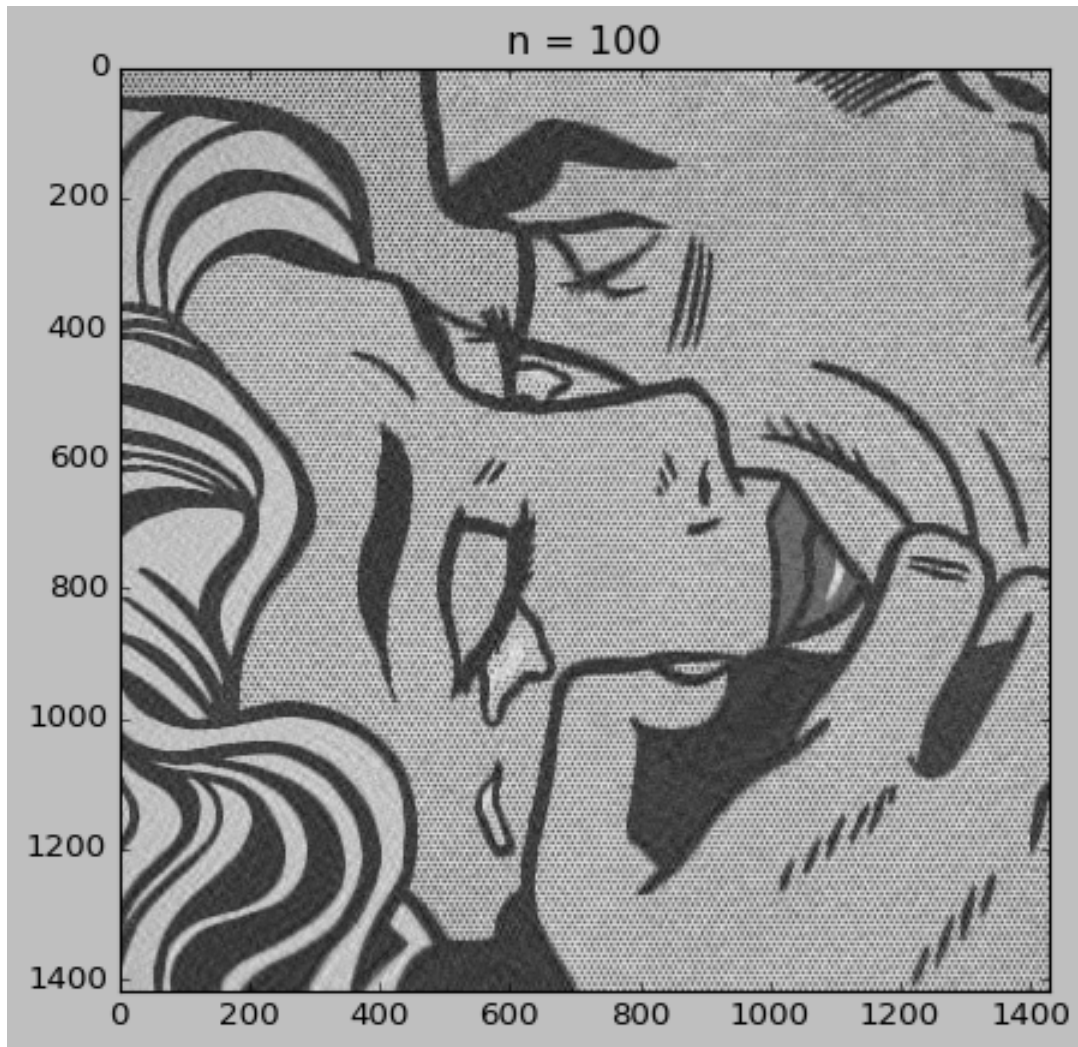
[ 2. 10. 159. ... 123. 165. 199.]

```



<PIL.WebPImagePlugin.WebPImageFile image mode=RGB size=1428x1417 at 0x20265B17CD0>


```
(1417, 1428)
(1417, 1417)
(1428, 1428)
(1417, 1428)
(1417, 1417)
(1428, 1428)
```



Con $n = 100$ la imagen comprimida es se aproxima a la imagen original

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>

Importación de Librerías

```
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')

```

Modelo PCA con el dataset de Iris

```

# Cargado de los datos
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Normalización de los datos por Z-score
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)

# Modelo de PCA
pca = PCA(n_components=2) # Estimado considerando 2 PCs
X_new = pca.fit_transform(X) # Proyección de los datos originales en el espacio del PCA

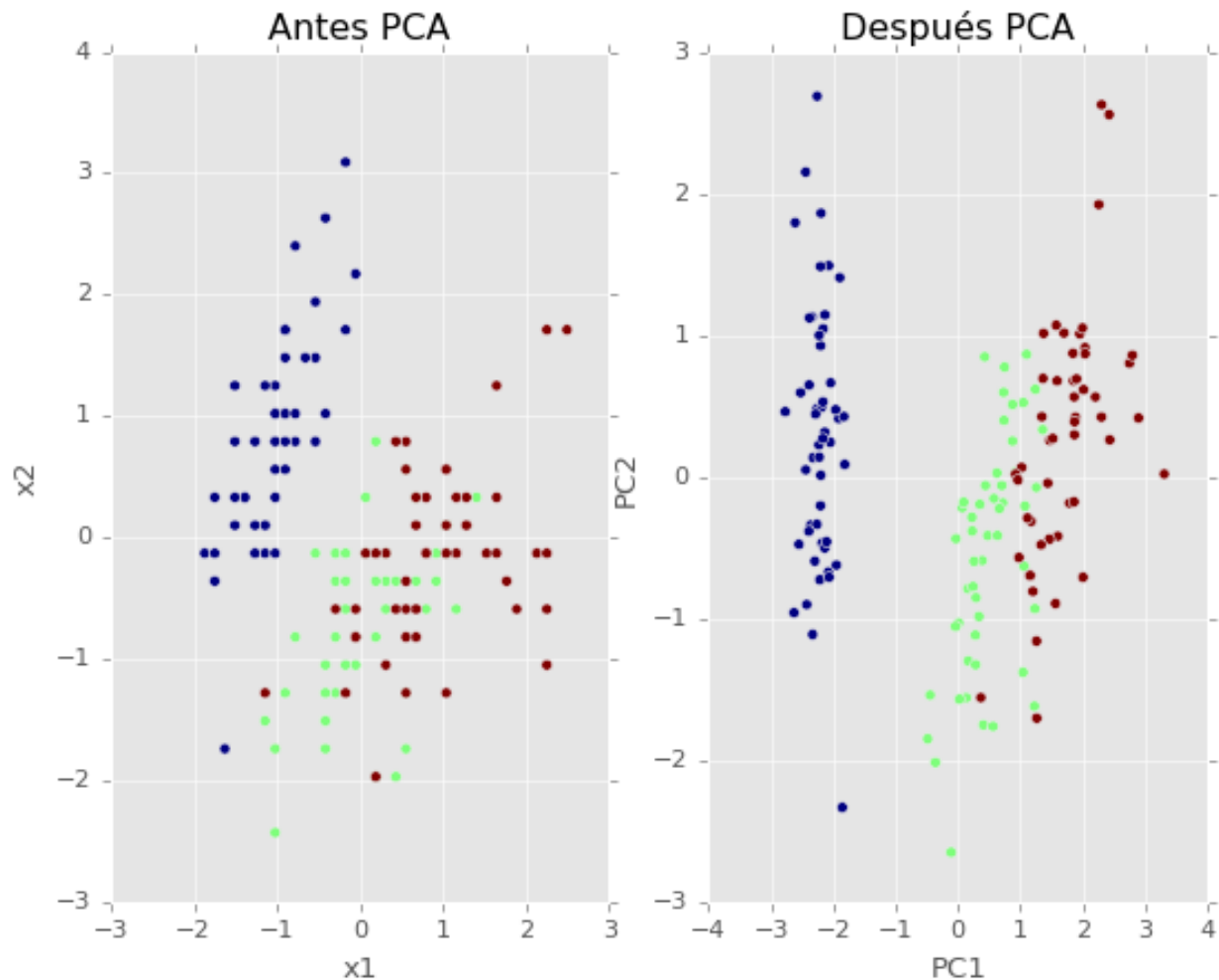
```

Plot de los datos antes y después de PCA

```

# Ploteo de los datos antes y después de transformarlos por el modelo de PCA
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('Antes 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('Después PCA')
plt.show()

```

Varianza de PC1 y PC2

```
print(pca.explained_variance_ratio_)
[0.72962445 0.22850762]
```

Podemos observar que el PC1 explica el 73% de la varianza y PC2 el 22%, juntos explican el 95% de la varianza total

Prueba de máxima varianza

Esta prueba demuestra que los eigenvalores de la matriz de covarianza original es igual que la varianza del espacio reducido

```
np.cov(X_new.T)
array([[2.93808505e+00, 1.53691966e-16],
       [1.53691966e-16, 9.20164904e-01]])

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

Por lo tanto podemos observar se los datos de la diagonal de la matriz de covarianza son los mismos que los eigenvalores del modelo de PCA

Features Importance

Ahora se procede a obtener los componentes más importantes

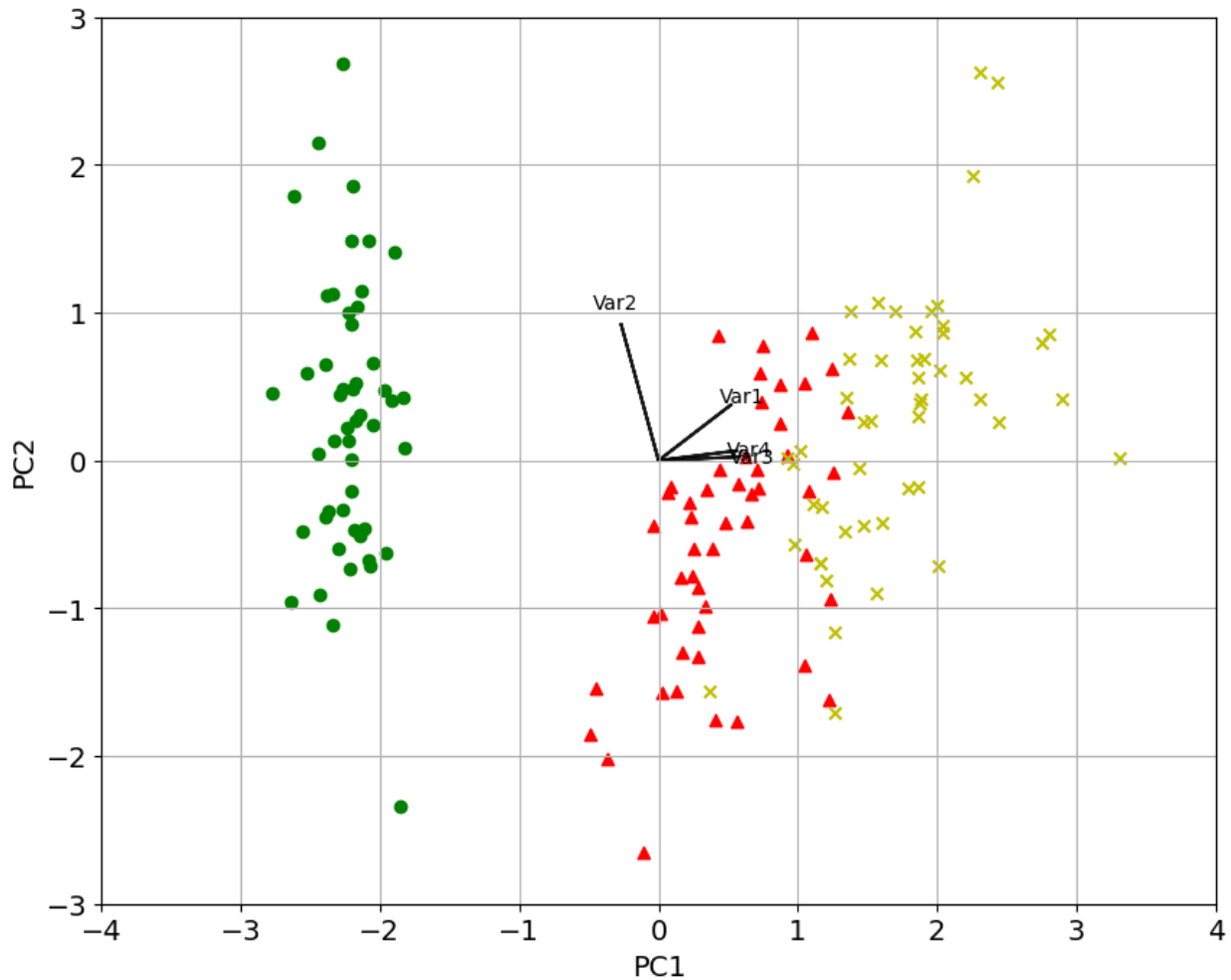
```
print(abs( pca.components_ ))  
[[0.52106591 0.26934744 0.5804131  0.56485654]  
 [0.37741762 0.92329566 0.02449161 0.06694199]]
```

Podemos observar que los componentes 1, 3 y 4 son los más importantes para el PC1. De igual manera los componentes 1 y 2 son los más importantes para el PC2.

Bi-plot

Nos ayuda a tener una mejor visualización del 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 group  
    for i in range(n):  
        #plot as arrows the variable scores (each variable has a score for PC1 and one for PC2)  
        plt.arrow(0, 0, coeff[i,0], coeff[i,1], color = 'k', alpha = 0.9,linestyle = '-',linewidth = 1.5)  
        plt.text(coeff[i,0]* 1.15, coeff[i,1] * 1.15, "Var"+str(i+1), color = 'k', ha = 'center', va = 'bottom')  
  
    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()
```



Como comentamos anteriormente los componentes 1, 3 y 4 son los más importantes para el PC1. De igual manera los componentes 1 y 2 son los más importantes para el PC2.

```
# Las variables 3 y 4 están positivamente correlacionadas
np.corrcoef(X[:,2], X[:,3])[1,0]
```

```
0.9628654314027957
```

```
# Las variables 2 y 3 están negativamente correlacionadas
np.corrcoef(X[:,1], X[:,2])[1,0]
```

```
-0.42844010433054014
```

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

Observamos que el análisis de PCA es una técnica de reducción de complejidad que nos puede ayudar a ver cuáles son los componentes y variables más relevantes. Para este proceso hicimos una transformación de los datos en donde transformamos las variables correlacionadas en un número pequeño de variables no correlacionadas.

Lo anterior se realizó haciendo una proyección (transformación ortogonal) de los datos originales a un espacio reducido del PCA utilizando los eigenvectores de la matriz de covarianza/correlación que nos ayudó a determinar los principales componentes 'PCs'. Estos datos proyectados nos ayudaron a observar donde se aporta la mayor varianza en los datos.

- En el análisis del PCA observamos como el PC1 explica el 73% de la varianza, el PC2 explica el 22% de la varianza, juntos explican el 95% total de la varianza

- Para observar la importancia de cada 'feature' en su magnitud de los valores de sus eigenvectores pudimos establecer que aquellos con mayor magnitud son los 'features' más importantes. En este caso, los features 1, 3 y 4 para el PC1, los features 1 y 2 para el PC2.
- Algo importante fue como la gráfica de biplot que no tiene una implementación directa en Python nos ayudó a visualizar el análisis del PCA y la correlación entre las variables. En este caso las variables 3 y 4 tienen una correlación positiva, mientras que las variables 2 y 3 tienen una correlación negativa.