

Curso: Ciencia y analítica de datos

Profesor titular: María de la Paz Rico Fernández

Profesor tutor: Victoria Guerrero Orozco

Nombre del estudiante: Ulises Guadalupe Ortega Mena

Matrícula: A01793983

Actividad Semanal -- 5

Repaso Transformación y reducción de dimensiones.

Bienvenido al notebook

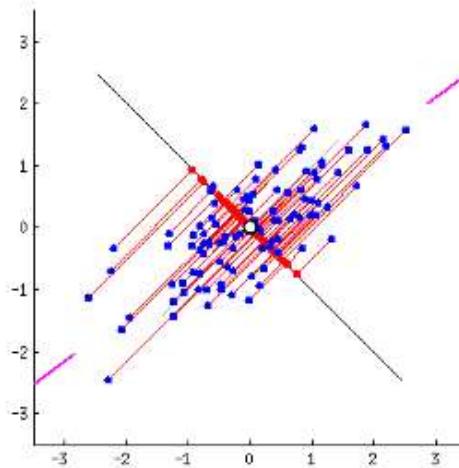
Repaso de Reducción de dimensiones

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 rasgos:

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 :)

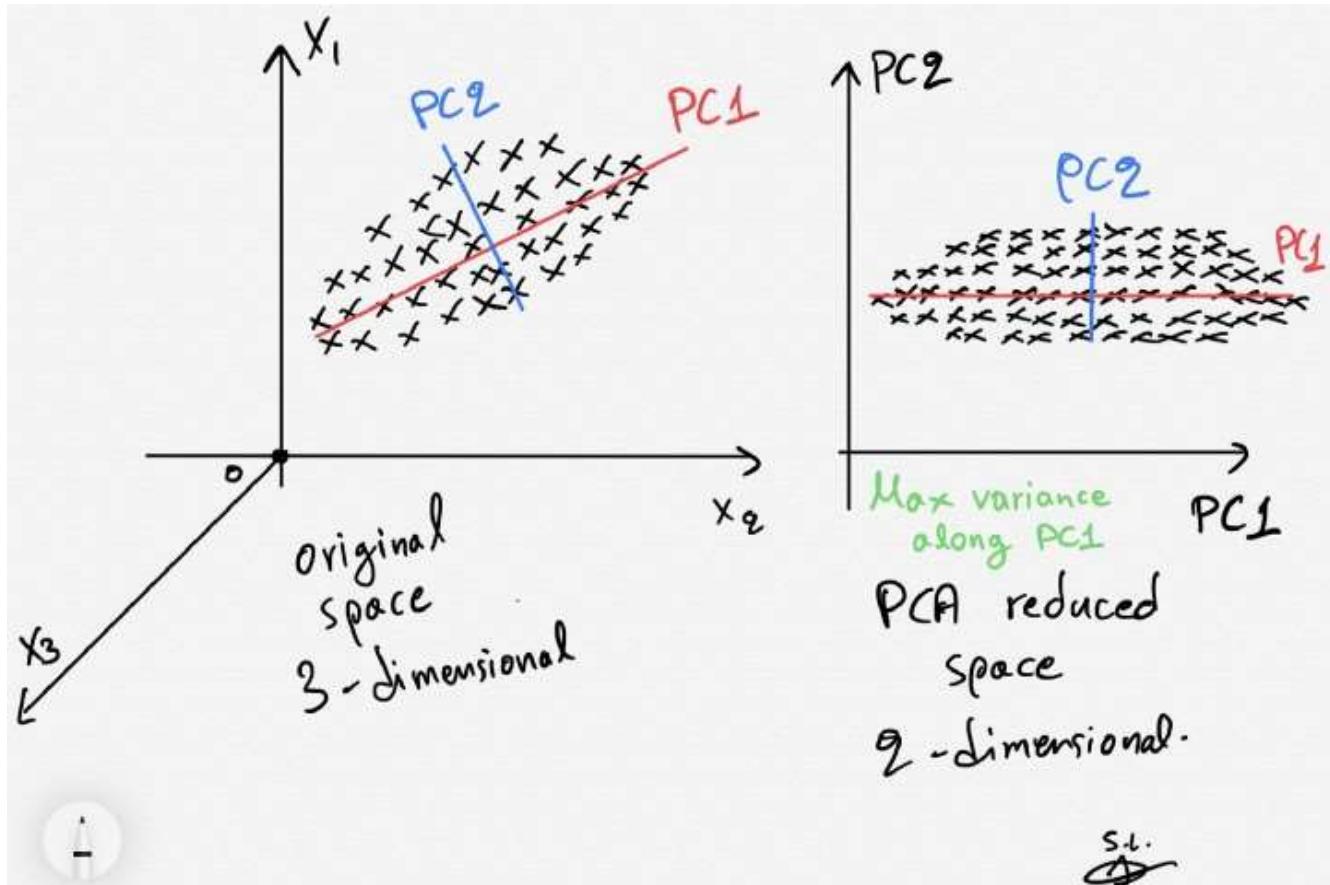
$$\mathbf{C} = \frac{\mathbf{X}^\top \mathbf{X}}{n - 1}$$

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

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

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

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



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

In []:

```
###-----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 eigendecomposición
values, vectors = eig(A)
#print(values) #D
#print(vectors) #W

#Ejemplo de reconstrucción
values, vectors = 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 reconstrucción de  $B = W \cdot D \cdot W^{-1}$ , te da lo mismo de A?
#ojo, estas multiplicando matrices, no escalares ;
#TU CODIGO AQUI-----
B = np.dot(np.dot(W,D),Winv)
print(B)
print("-----")
```

-----Matriz original-----

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

-----Matriz reconstruida-----

```
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
```

In []:

```
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]]
```

In []:

```
#Matriz 1
A1 = array([[3, 0, 2], [3, 0, -2], [0, 1, 1]])

print("-----Matriz original-----")
print(A1)
print("-----")

values, vectors = eig(A1)
W = vectors
Winv = np.linalg.inv(W)
D = np.diag(values)

B = np.dot(np.dot(W,D),Winv)

#reconstruye la matriz
print("-----Matriz reconstruida-----")
print(np.print(B))
print("-----")
```

-----Matriz original-----

```
[[ 3  0  2]
 [ 3  0 -2]
 [ 0  1  1]]
```

-----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]]
```

In []:

```
#Matriz 2
A2 = array([[1, 3, 8], [2, 0, 0], [0, 0, 1]])

print("-----Matriz original-----")
print(A2)
print("-----")

values, vectors = eig(A2)
W = vectors
Winv = np.linalg.inv(W)
D = np.diag(values)

B = np.dot(np.dot(W,D),Winv)

#reconstruye la matriz
print("-----Matriz reconstruida-----")
print(np.print(B))
print("-----")
```

-----Matriz original-----

```
[[1 3 8]
 [2 0 0]
 [0 0 1]]
```

-----Matriz reconstruida-----

```
[[1. 3. 8.]
 [2. 0. 0.]
 [0. 0. 1.]]
```

In []:

```
#Matriz 3
A1 = array([[5, 4, 0], [1, 0, 1], [10, 7, 1]])

print("-----Matriz original-----")
print(A1)
```

```

print("-----")

values, vectors = eig(A1)
W = vectors
Winv = np.linalg.inv(W)
D = np.diag(values)

B = np.dot(np.dot(W,D),Winv)

#reconstruye la matriz
print("-----Matriz reconstruida-----")
print(np.print(B))
print("-----")

```

-----Matriz original-----

```
[[ 5  4  0]
 [ 1  0  1]
 [10  7  1]]
```

-----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

Veamoslo 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.

$$X_{n \times m} = U_{n \times n} \Sigma_{n \times m} V^*_{m \times m}$$

$$U_{n \times n} U^*_{n \times n} = I_n$$

$$V^*_{m \times m} V_{m \times m} = I_m$$

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

```
In [ ]:
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/2015/0
#img = Image.open('lucy.jpg')
imggray = img.convert('LA')
```

```

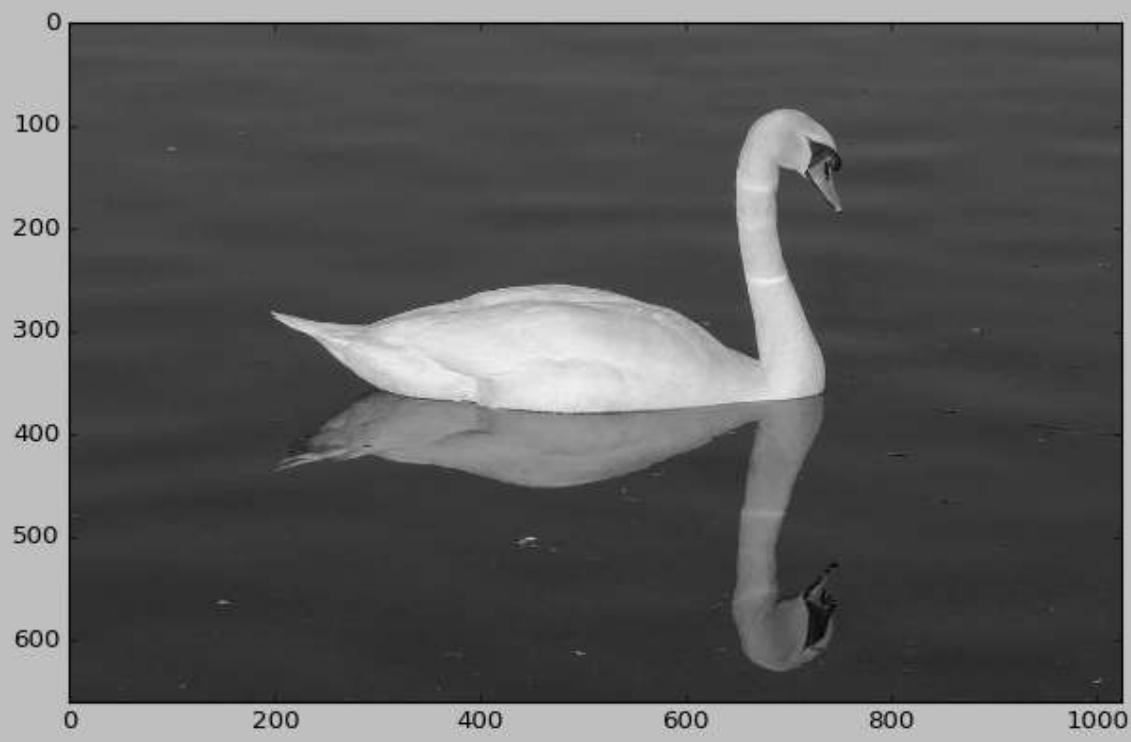
imgmat = np.array(list(imggray.getdata(band=0)),float)
print(imgmat)

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 0x2450A588A60>

In []:

```

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

```

Out[]:

```
(660, 1024)
```

In []:

```

U.shape

```

Out[]:

```
(660, 660)
```

In []:

```

V.shape

```

Out[]:

```
(1024, 1024)
```

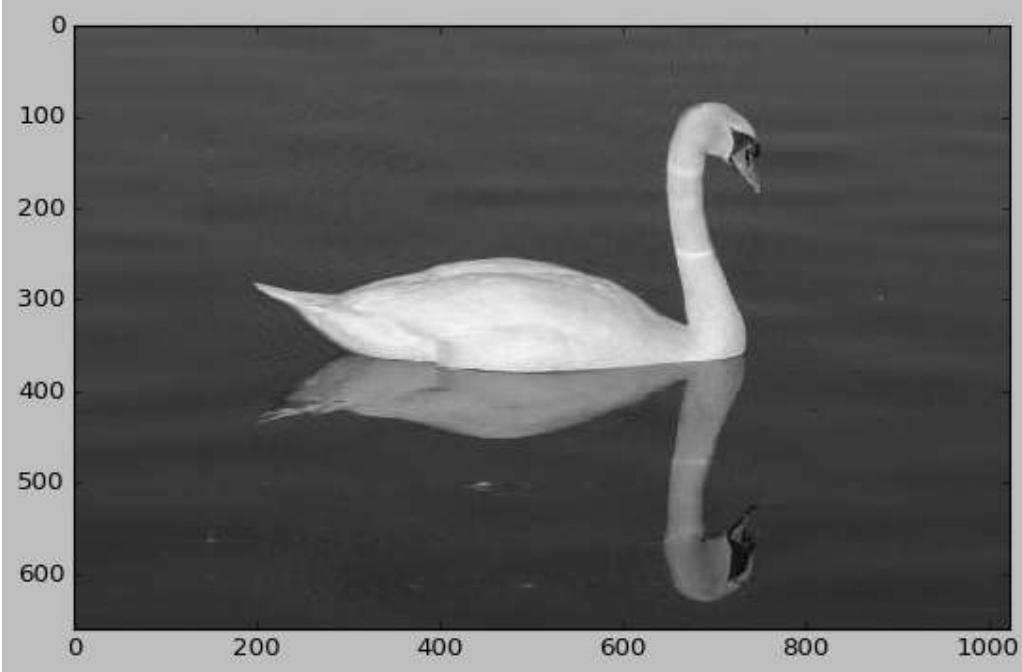
In []:

```

#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 estan 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[:,0:nvalue])*np.diag(D[0:nvalue])*np.matrix(V[0: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 està comprimida")

```



Felicidades la imagen está comprimida

¡Ahora es tu turno!, comprime 3 imágenes

In []:

```
#imagen 1
plt.style.use('classic')
img = Image.open(urllib.request.urlopen('https://biblioteca.acropolis.org/wp-content/uploads/2022/1
#img = Image.open('lucy.jpg')
imggray = img.convert('LA')
imgmat = np.array(list(imggray.getdata(band=0)),float)

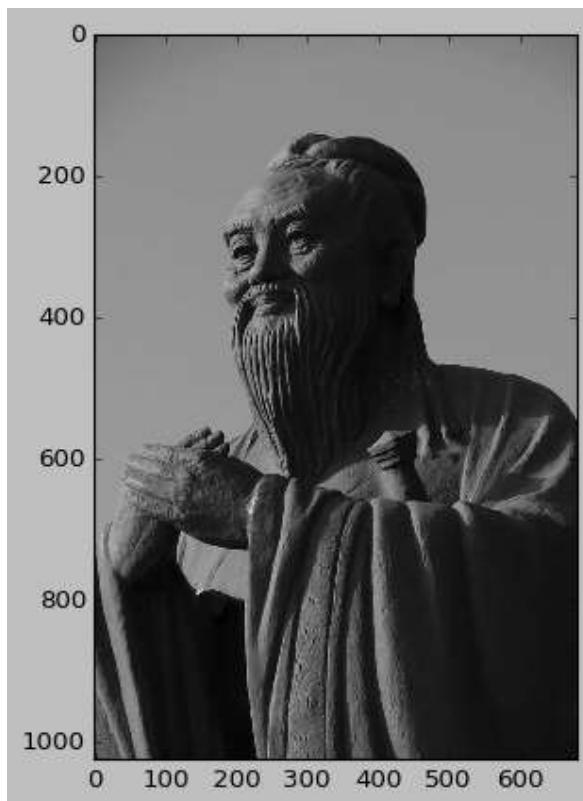
imgmat.shape = (imggray.size[1],imggray.size[0])

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

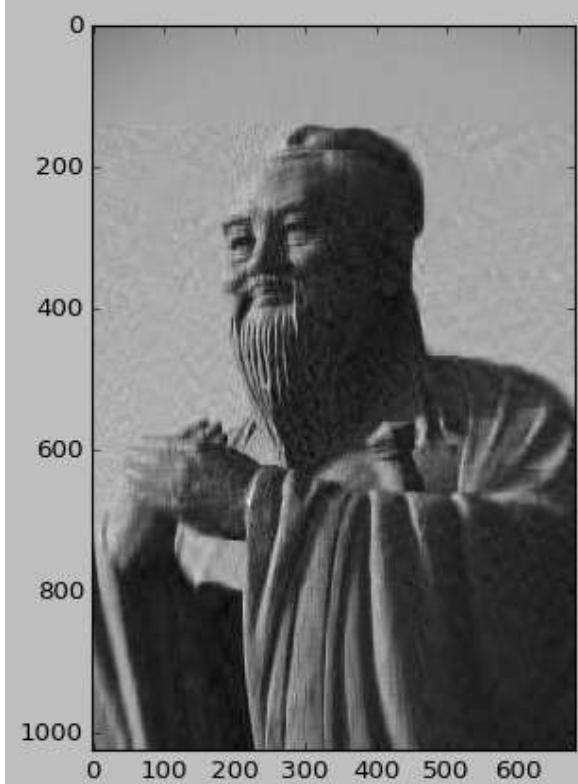
U,D,V = np.linalg.svd(imgmat)
print(imgmat.shape)

nvalue =40
#-----
reconstimg = np.matrix(U[:,0:nvalue])*np.diag(D[0:nvalue])*np.matrix(V[0:nvalue,:])

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



<PIL.Image.Image image mode=LA size=682x1024 at 0x2450AD53BB0>
(1024, 682)



Felicidades la imagen está comprimida

```
In [ ]: #imagen 2

plt.style.use('classic')
img = Image.open(urllib.request.urlopen('https://biblioteca.acropolis.org/wp-content/uploads/2014/1
#img = Image.open('lucy.jpg')
imggray = img.convert('LA')
imgmat = np.array(list(imggray.getdata(band=0)),float)

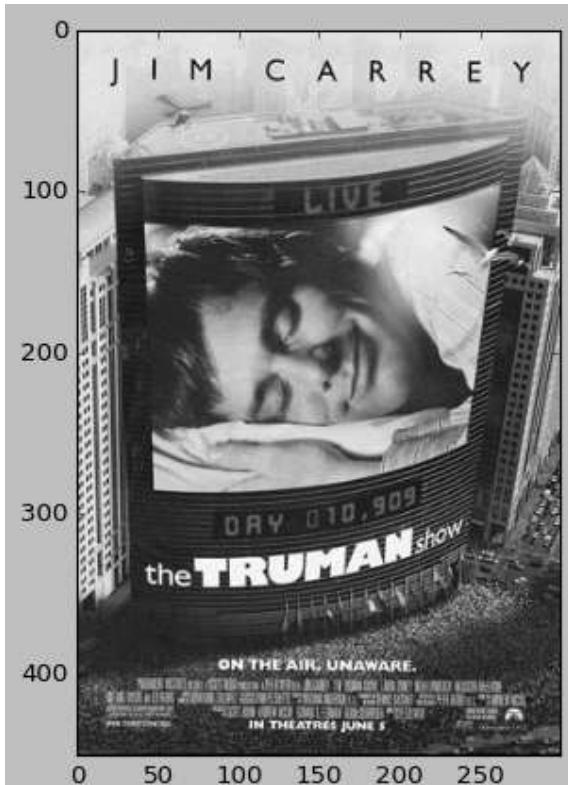
imgmat.shape = (imggray.size[1],imggray.size[0])

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

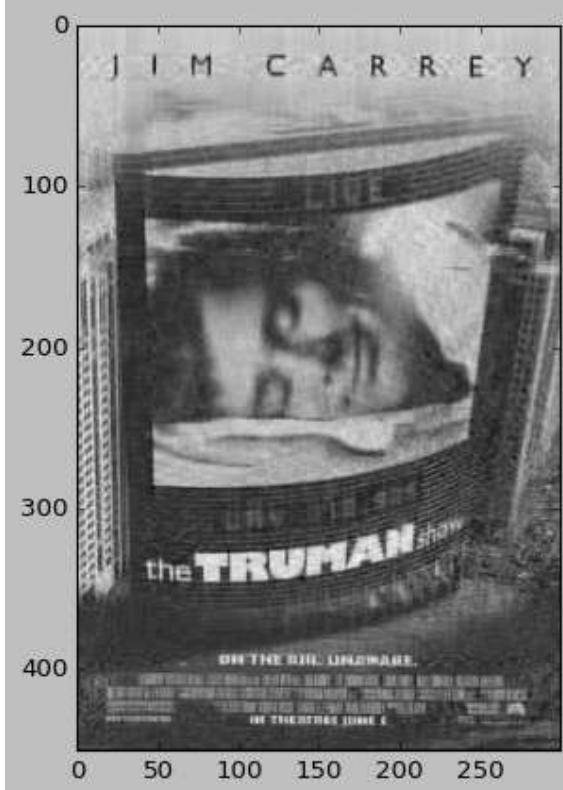
U,D,V = np.linalg.svd(imgmat)
print(imgmat.shape)

nvalue =40
#-----
reconstimg = np.matrix(U[:,0:nvalue])*np.diag(D[0:nvalue])*np.matrix(V[0:nvalue,:])

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



<PIL.Image.Image image mode=LA size=300x450 at 0x2450CDC3400>
(450, 300)



Felicidades la imagen está comprimida

```
In [ ]: #imagen 3

plt.style.use('classic')
img = Image.open(urllib.request.urlopen('https://biblioteca.acropolis.org/wp-content/uploads/2019/0
#img = Image.open('lucy.jpg')
imggray = img.convert('LA')
imgmat = np.array(list(imggray.getdata(band=0)),float)

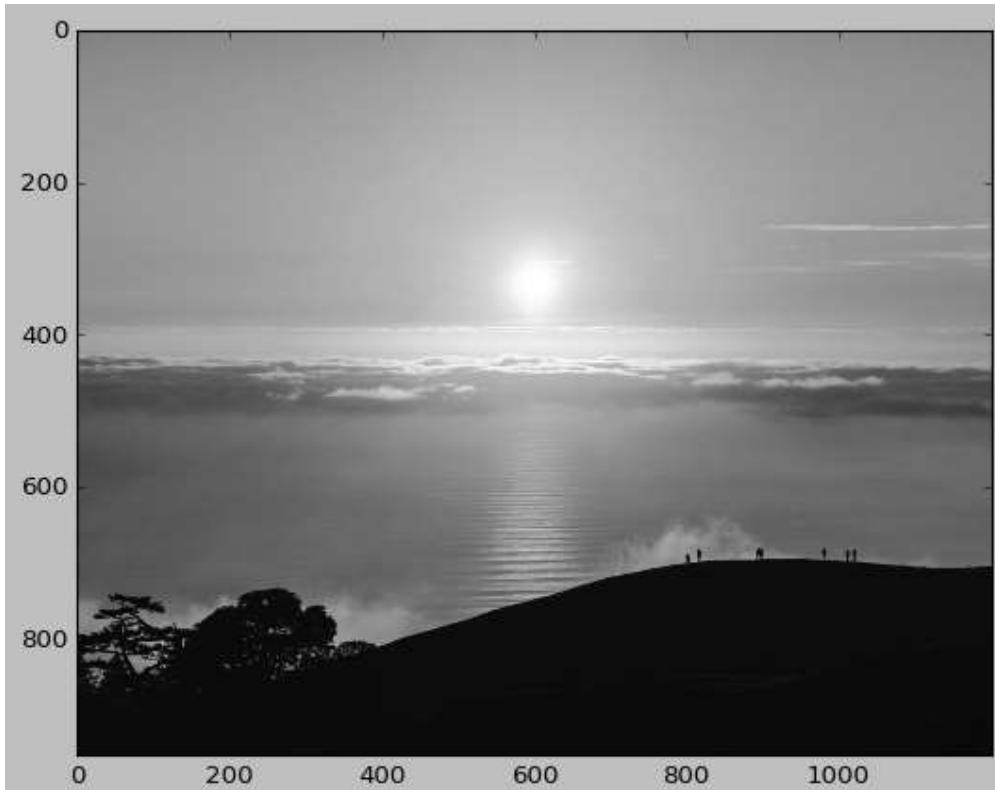
imgmat.shape = (imggray.size[1],imggray.size[0])

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

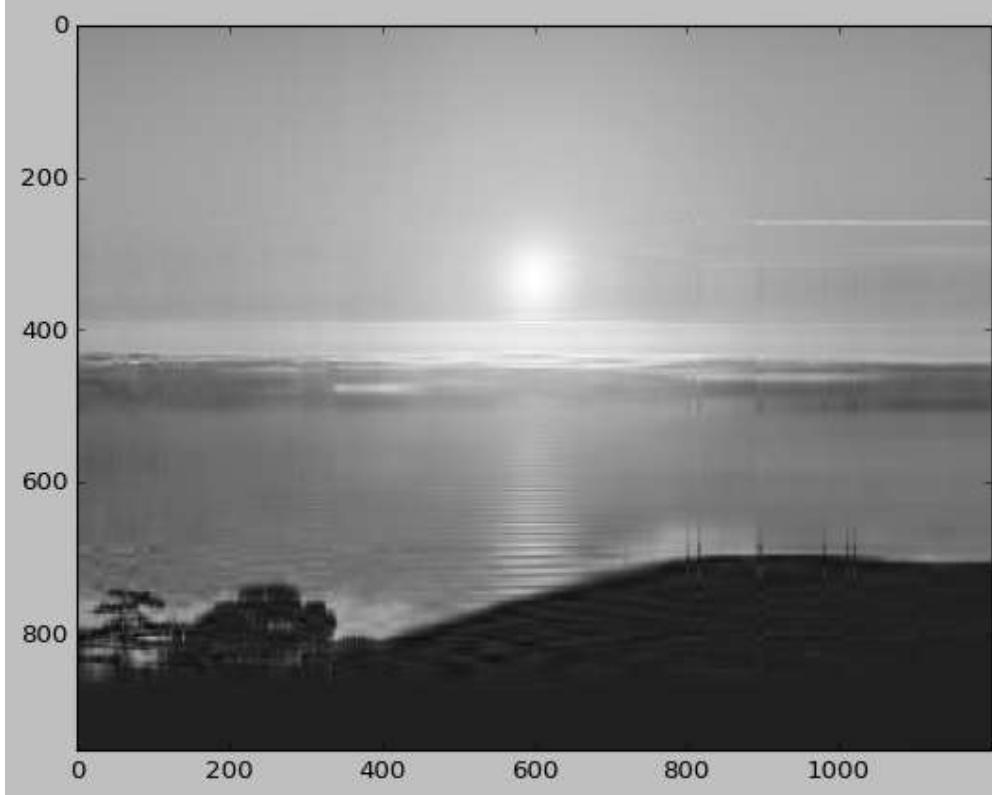
U,D,V = np.linalg.svd(imgmat)
print(imgmat.shape)

nvalue =15
#-----
reconstimg = np.matrix(U[:,0:nvalue])*np.diag(D[0:nvalue])*np.matrix(V[0:nvalue,:])

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



<PIL.Image.Image image mode=LA size=1200x951 at 0x2450B994D30>
(951, 1200)



Felicidades la imagen está comprimida

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>

In []:

```
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')
```

In []:

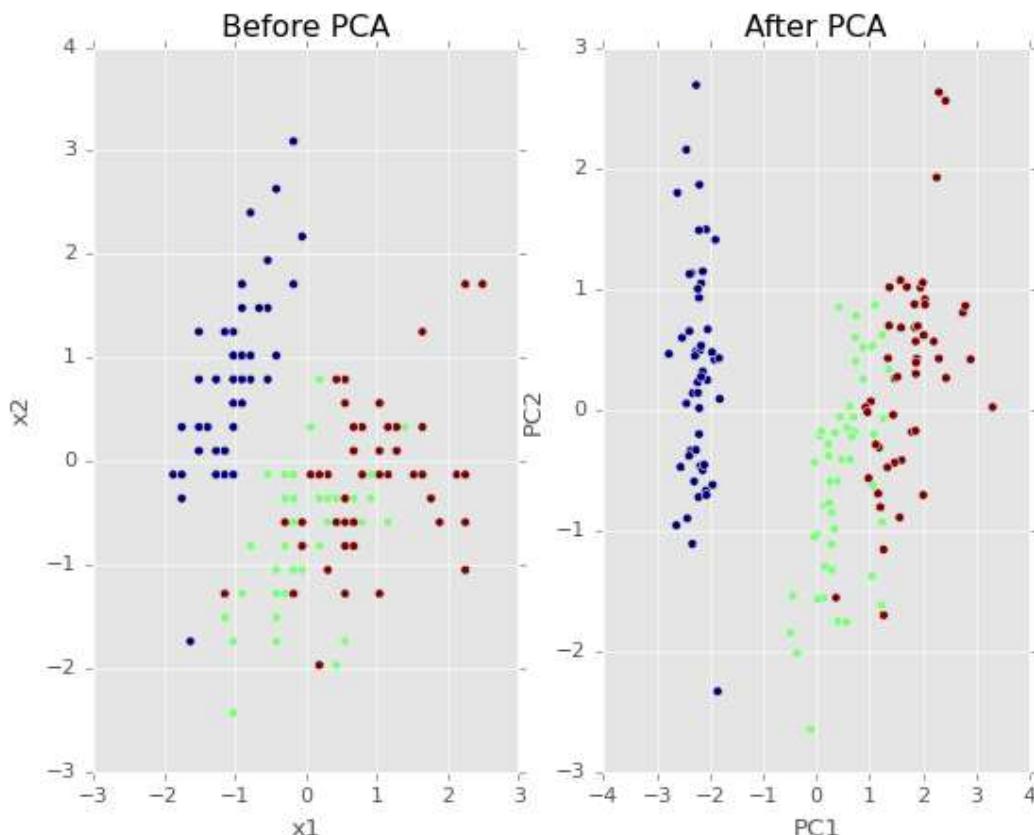
```
# Se cargan los datos
iris = datasets.load_iris()
X = iris.data
y = iris.target

scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)

# Modelo
pca = PCA(n_components=2) # Se estiman 2 PCs
X_new = pca.fit_transform(X) # Se proyecta la data en el dominio PCA
```

In []:

```
# Se grafican los datos del antes y después de PCA a modo de comparación
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()
```



```
In [ ]: # La varianza se maximiza a lo largo de PC1 (73%) y PC2 (22%).
print(pca.explained_variance_ratio_)
```

```
[0.72962445 0.22850762]
```

```
In [ ]: # La máxima varianza estimando la matriz de covarianza del espacio reducido
np.cov(X_new.T)
```

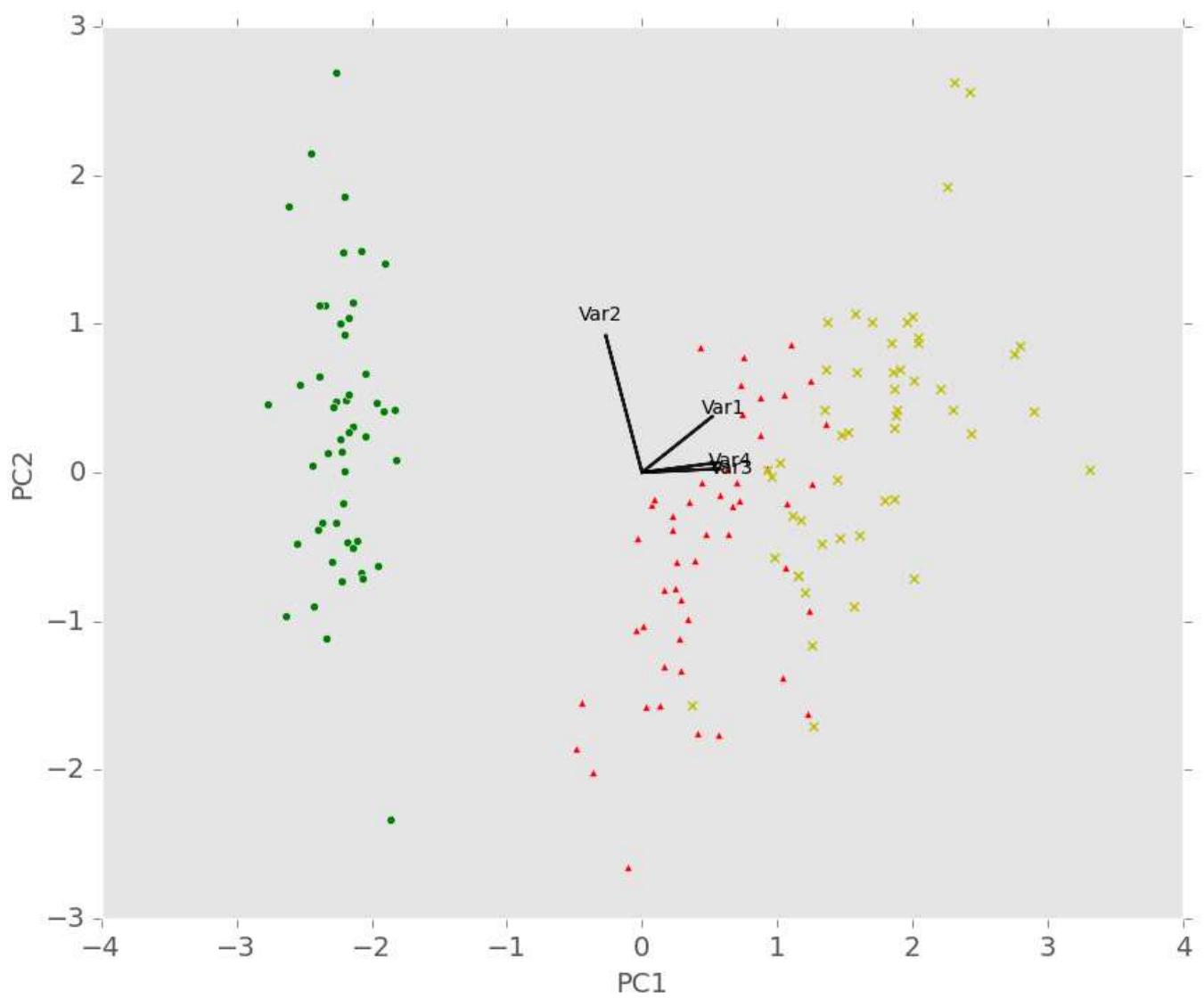
```
Out[ ]: array([[ 2.93808505e+00, -1.90749728e-16],
 [-1.90749728e-16,  9.20164904e-01]])
```

```
In [ ]: # Obtener componentes principales
print(abs( pca.components_ ))
```

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

```
In [ ]: # Función para visualizar los resultados de un análisis 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=2)
        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{1}", size=14)
    plt.ylabel("PC{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)
```

```
In [ ]: import matplotlib as mpl
# Llamar la función para los primeros dos componentes principales.
biplot(X_new[:,0:2], np.transpose(pca.components_[0:2, :]), y)
plt.show()
```



```
In [ ]: # Var 3 y Var 4 están extremadamente correlacionados positivamente
np.corrcoef(X[:,2], X[:,3])[1,0]
```

```
Out[ ]: 0.9628654314027961
```

```
In [ ]: # Var2 y Var3 están negativamente correlacionados
np.corrcoef(X[:,1], X[:,2])[1,0]
```

```
Out[ ]: -0.4284401043305398
```

- **Realiza un comentario relacionado a los pasos que se llevaron a cabo en este proceso de feature importances.**

Los pasos a seguir son muy sencillos y nada laboriosos, para el tipo de tarea que se realiza me parece que es muy práctico realizarlo y la utilidad es bastante amplia. Noté que son muy variados sus usos, no imaginaba que nos sirviera para comprimir data. Es una técnica muy cómo de usar y me gustaría profundizar más en el tema.

- **¿Qué es feature importance y para qué nos sirve?**

Es una técnica que se utiliza para identificar las características más importantes de un conjunto de datos. Al realizar el análisis se puede identificar la data que puede aportar mayor información al modelo y con ello realizar una implementación mucho más óptima.

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

Se encontró que para PC1 los componentes principales son las variables 1,3 y 4. Para PC2 los componentes principales son las variables 1 y 2. Bien se podría poner el enfoque en estos componentes para poder entrenar el modelo y que la implementación sea más óptima, ya que estamos alimentando al modelo con data realmente relevante.

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

En datasets donde la cantidad de variables de entrada sea muy grande y no esté seguro de cuáles descartar. Esto permitirá un uso óptimo de la data, para no alimentar al modelo con data "basura". Además, el no saturar al modelo durante el entrenamiento con datos que no aportan mucho, ayudará a que se consuman menos recursos de cómputo, lo que es muy viable cuando se llega a rentar servicios de nube, ya que ahí se cobra por el tiempo de uso y el performance que se adquiere con el servicio.