Transformación y reducción de dimensiones

Nombre de la entrega: Actividad Semanal -- 5 Repaso Transformación y reducción de dimensiones

Alumno: Erick de Jesus Hernández Cerecedo

Matrícula: A01066428

Materia: Ciencia y analítica de datos

Profesor: María de la Paz Rico Fernández

Fecha: Jueves, 27 de octubre de 2022

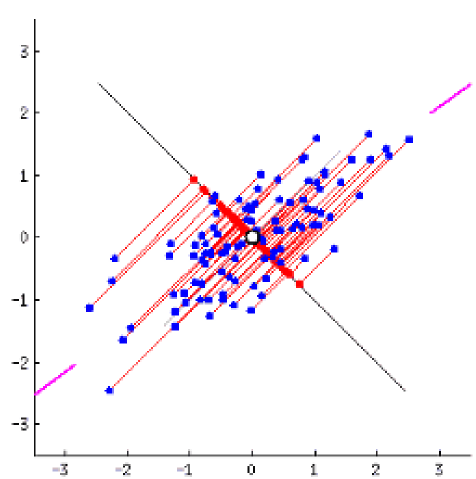
Repaso de Reducción de dimensiones

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 :)

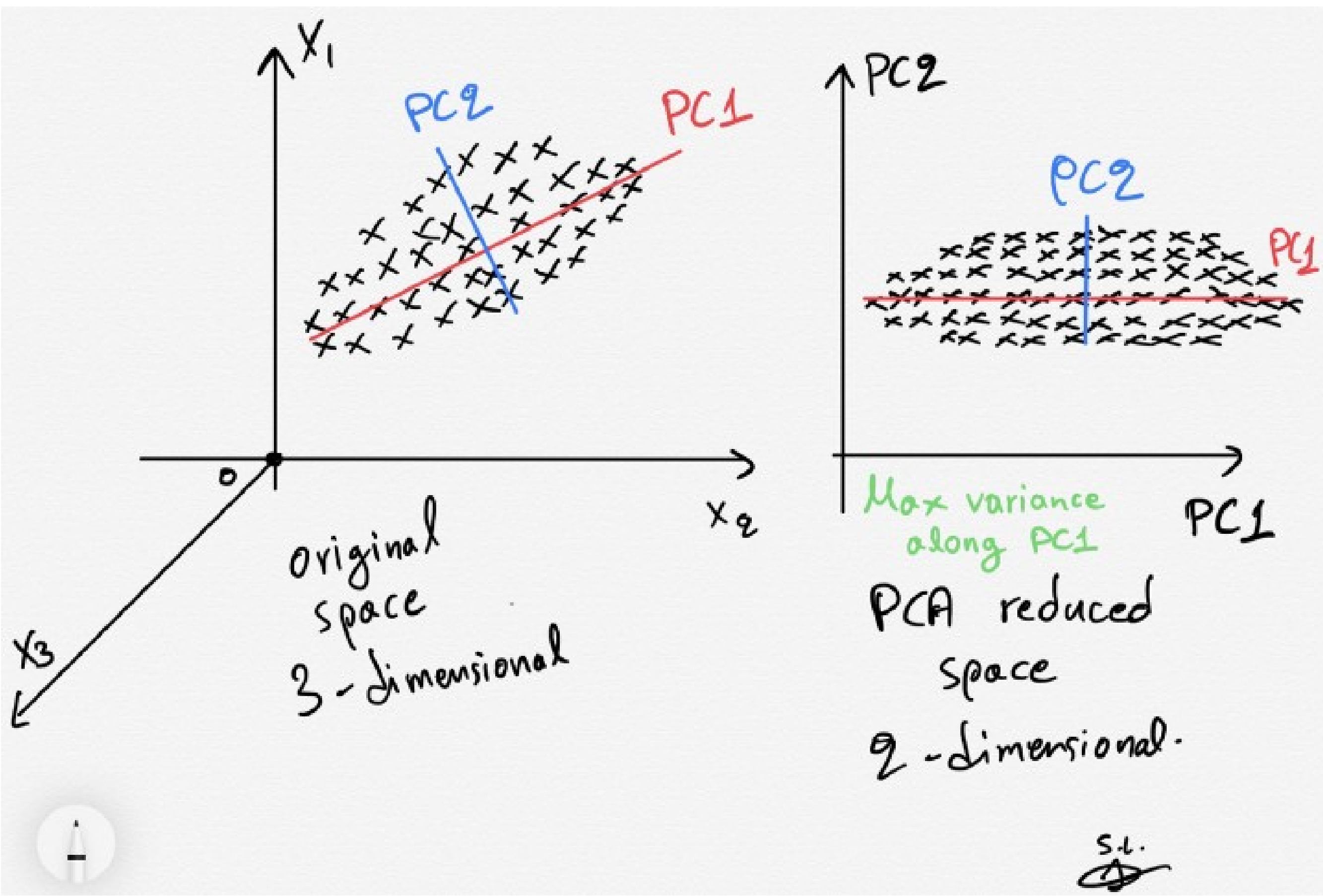
$$\mathbf{C} = \frac{\mathbf{X}^\top \mathbf{X}}{n-1}$$

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 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

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

y reconstruye la matriz original a traves 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 [52]:

```
###-----------------EJEMPLO DE EIGENVALORES
from pprint import pprint
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-------------------------------------
WD = np.matmul(W, D)
B = np.matmul(WD, Winv)
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.]]
-----------------------------
```

In [53]:

```
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 [110…

```
#Matriz 1
A = array([[3, 0, 2], [3, 0, -2], [0, 1, 1]])
print("Matriz original: \n", A)
# calcula la eigendescomposición
values, vectors = eig(A)
print("Valores: \n", values) #D
print("Vectores: \n", vectors) #W

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

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

WD = np.matmul(W, D)
B = np.matmul(WD, Winv)
print("Matriz reconstruida: \n", B)
```

```
Matriz original:
 [[ 3  0  2]
 [ 3  0 -2]
 [ 0  1  1]]
Valores:
 [3.54451153+0.j         0.22774424+1.82582815j 0.22774424-1.82582815j]
Vectores:
 [[-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:
 [[ 3.00000000e+00-1.90344153e-18j  3.90312782e-16+2.44274141e-18j
   2.00000000e+00+1.20428621e-17j]
 [ 3.00000000e+00-1.25140509e-17j  7.77156117e-16-4.36522035e-17j
  -2.00000000e+00+7.21602177e-20j]
 [ 1.66533454e-16-4.34400411e-17j  1.00000000e+00-1.06407305e-17j
   1.00000000e+00+6.49738882e-17j]]
```

In [55]:

```
#Matriz 2
A = array([[1, 3, 8], [2, 0, 0], [0, 0, 1]])
print("Matriz original: \n", A)
# calcula la eigendescomposición
values, vectors = eig(A)
print("Valores: \n", values) #D
print("Vectores: \n", vectors) #W

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

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

WD = np.dot(W, D)
B = np.dot(WD, Winv)
print("Matriz reconstruida: \n", B)
```

```
Matriz original:
 [[1 3 8]
```

```
 [2 0 0]
 [0 0 1]]
Valores:
 [ 3. -2.  1.]
Vectores:
 [[ 0.83205029 -0.70710678 -0.42399915]
 [ 0.5547002   0.70710678 -0.8479983 ]
 [ 0.          0.          0.31799936]]
Matriz reconstruida:
 [[1.00000000e+00 3.00000000e+00 8.00000000e+00]
 [2.00000000e+00 7.41483138e-17 7.08397389e-16]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
```

In [56]:

```
#Matriz 3
A = array([[5, 4, 0], [1, 0, 1], [10, 7, 1]])
print("Matriz original: \n", A)
# calcula la eigendescomposición
values, vectors = eig(A)
print("Valores: \n", values) #D
print("Vectores: \n", vectors) #W

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

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

WD = np.dot(W, D)
B = np.dot(WD, Winv)
print("Matriz reconstruida: \n", B)
```

```
Matriz original:
 [[ 5  4  0]
 [ 1  0  1]
 [10  7  1]]
Valores:
 [ 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:
 [[ 5.00000000e+00  4.00000000e+00 -1.53912019e-15]
 [ 1.00000000e+00 -1.30389602e-15  1.00000000e+00]
 [ 1.00000000e+01  7.00000000e+00  1.00000000e+00]]
```

¿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.

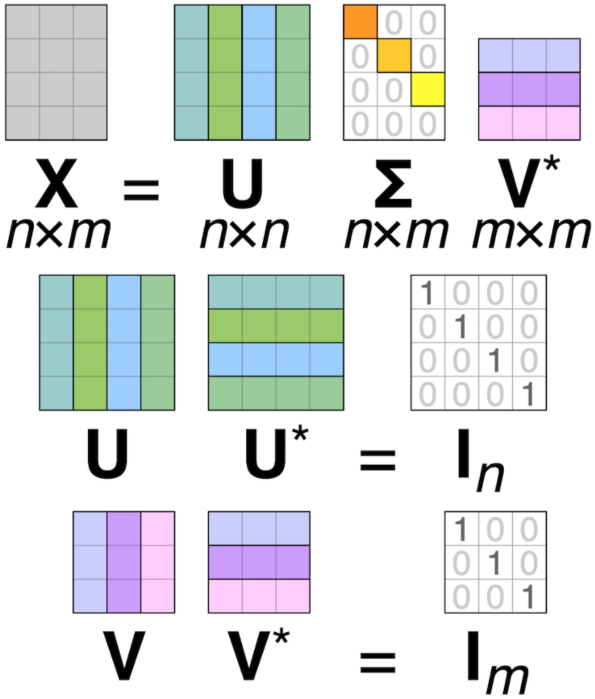
Es decir: Unidades-PC PC-Unidades

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.



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

In [67]:
```
import urllib
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np

#################################################################################
# Seccion de codigo implementada para solucionar problemas con el certificado
import ssl

try:
    _create_unverified_https_context = ssl._create_unverified_context
except AttributeError:
    # Legacy Python that doesn't verify HTTPS certificates by default
    pass
else:
    # Handle target environment that doesn't support HTTPS verification
    ssl._create_default_https_context = _create_unverified_https_context

# Fin de la seccion de codigo auxiliar
#################################################################################


plt.style.use('classic')
img = Image.open(urllib.request.urlopen('https://biblioteca.acropolis.org/wp-content/upl
#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 0x120A53880>
```

In [68]:
```
U,D,V = np.linalg.svd(imgmat)
imgmat.shape
```

Out[68]:
```
(660, 1024)
```

In [60]:
```
U.shape
```

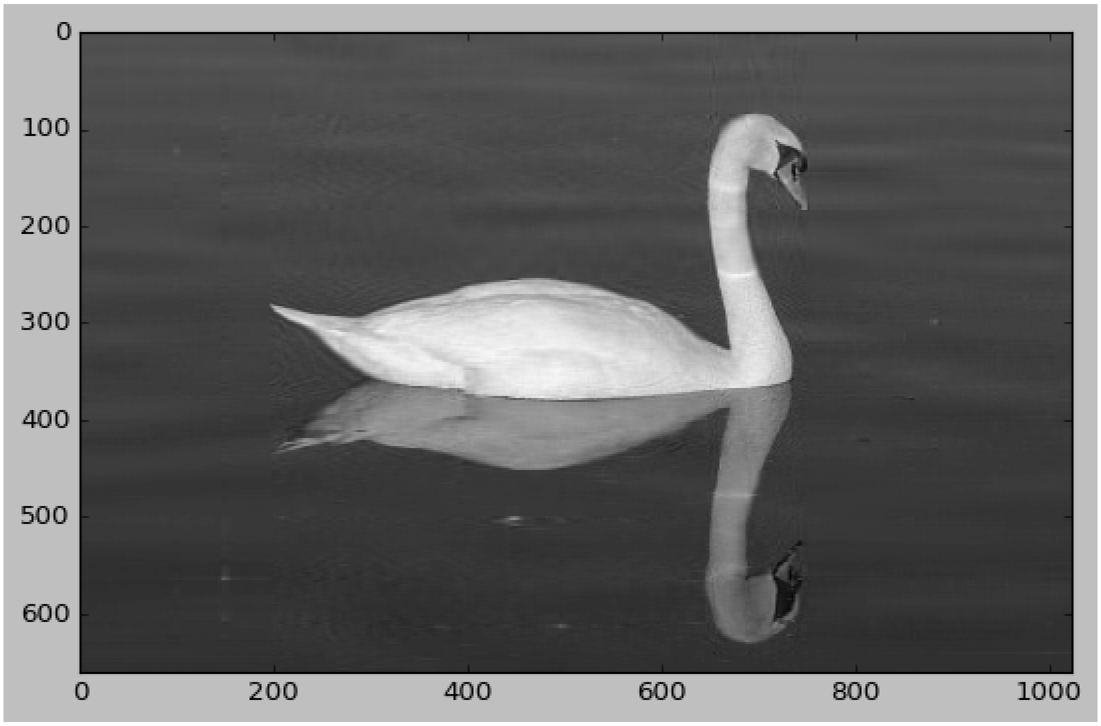
Out[60]:
```
(660, 660)
```

In [61]:
```
V.shape
```

Out[61]:
```
(1024, 1024)
```

In [76]:
```
#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")
```

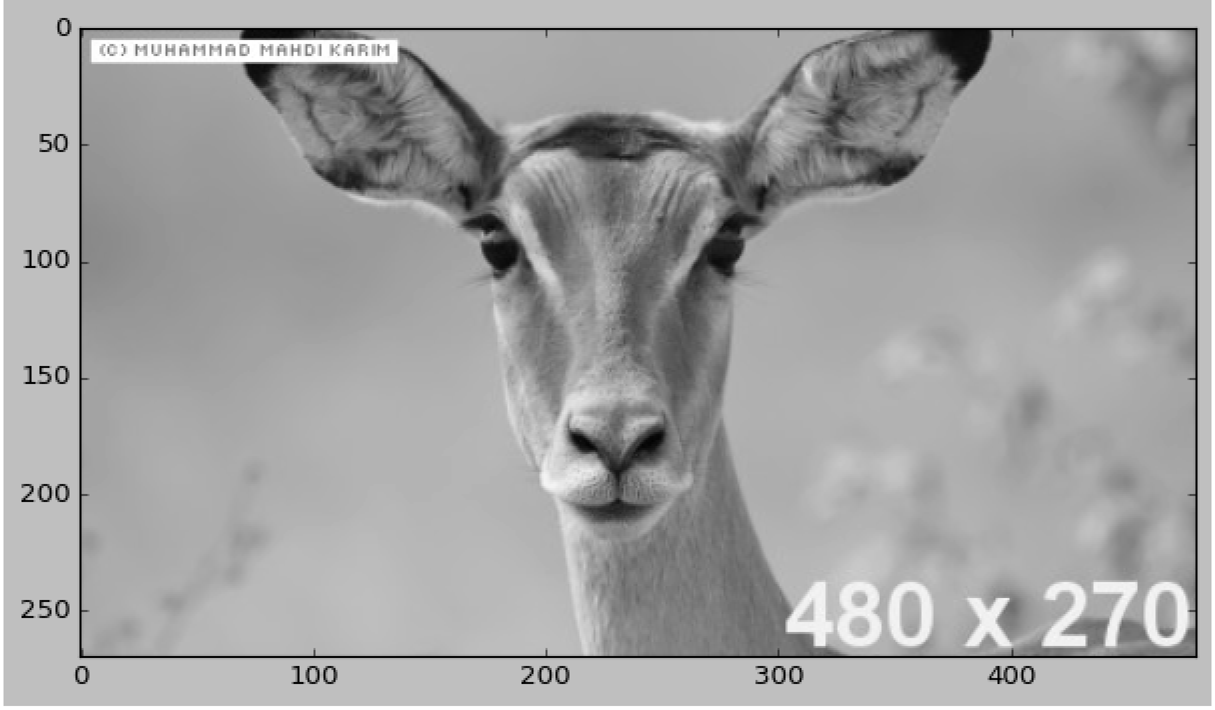


```
Felicidades la imagen está comprimida
```

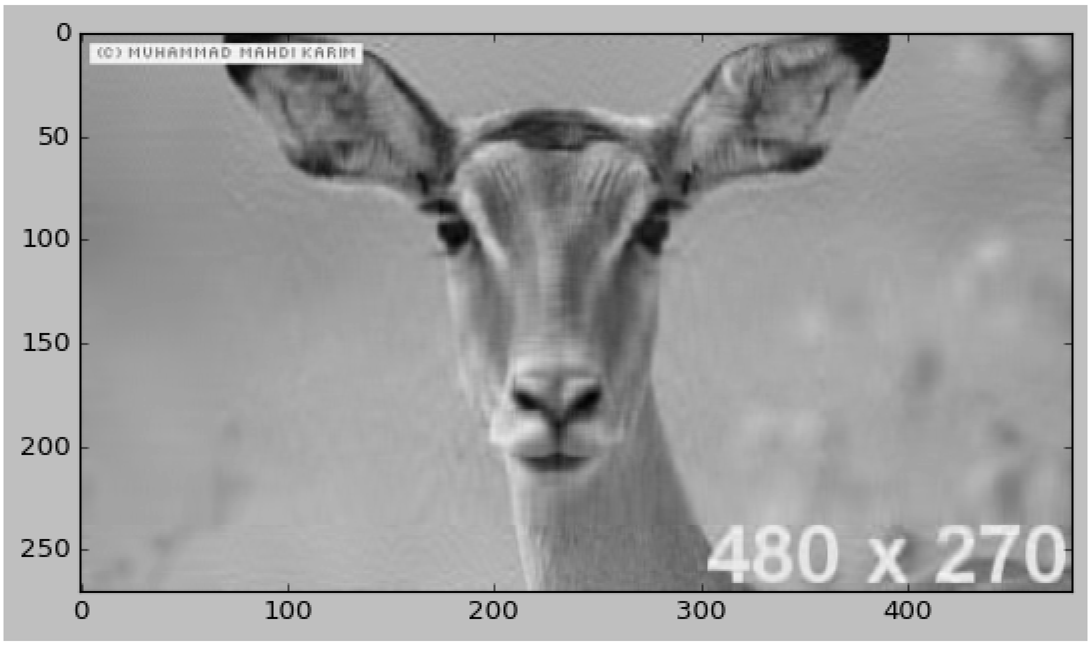
¡Ahora es tu turno!, comprime 3 imagenes

In [91]:
```
#imagen 1
plt.style.use('classic')
img = Image.open(urllib.request.urlopen('http://dummy-images.com/animals/dummy-480x270-I
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)

# Compresión de imagen
U,D,V = np.linalg.svd(imgmat)
nvalue = 40

reconstimg = np.matrix(U[:,:nvalue])*np.diag(D[:nvalue])*np.matrix(V[:nvalue,:])
plt.imshow(reconstimg,cmap='gray')
plt.show()
print("imagen está comprimida")
```

```
[153. 152. 158. ... 120. 120. 123.]
```




```
<PIL.Image.Image image mode=LA size=480x270 at 0x120A504F0>
```



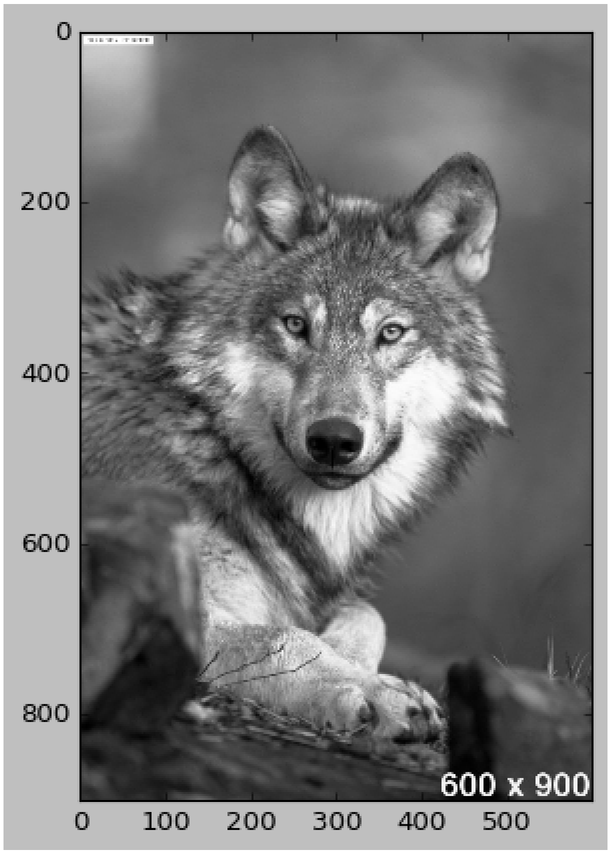

```
imagen está comprimida
```

In [86]:

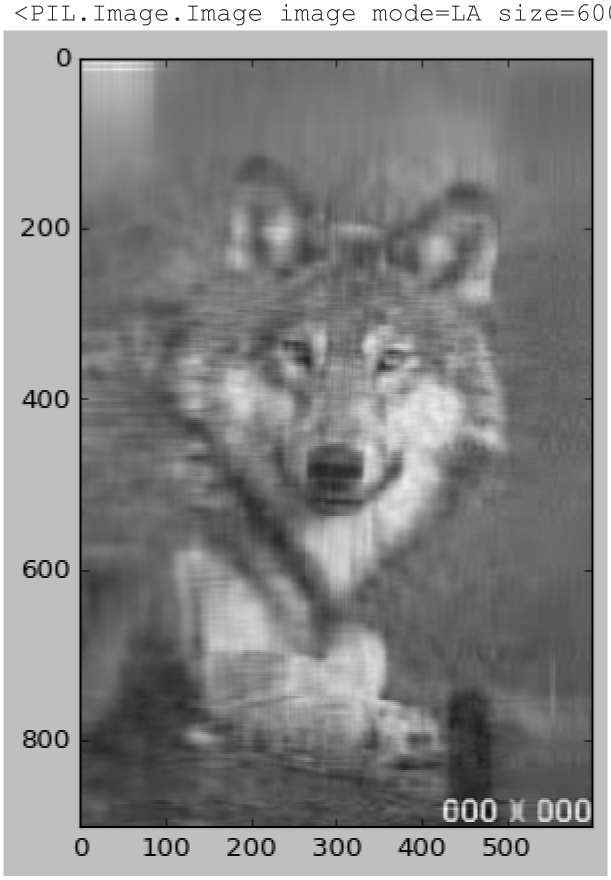
```
#imagen 2
plt.style.use('classic')
img = Image.open(urllib.request.urlopen('http://dummy-images.com/animals/dummy-600x900-C
imggray = img.convert('LA')
imgmat = np.array(list(imggray.getdata(band=0)),float)

print(imgmat)

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

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

# Compresión de imagen
U,D,V = np.linalg.svd(imgmat)
nvalue = 30

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

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

```
[192. 195. 191. ...  53.  54.  55.]
```




```
<PIL.Image.Image image mode=LA size=600x900 at 0x1210E5F00>
```



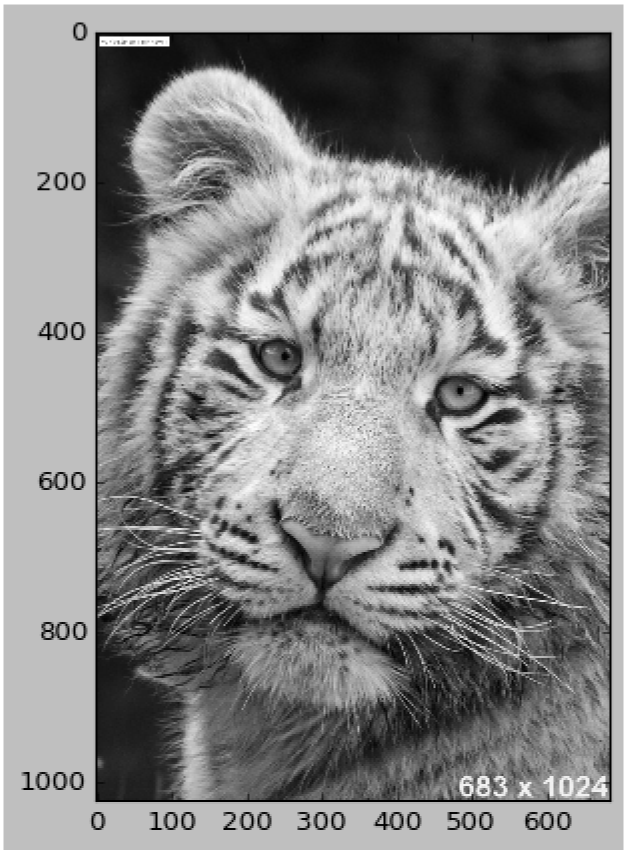
```
imagen está comprimida
```

In [93]:

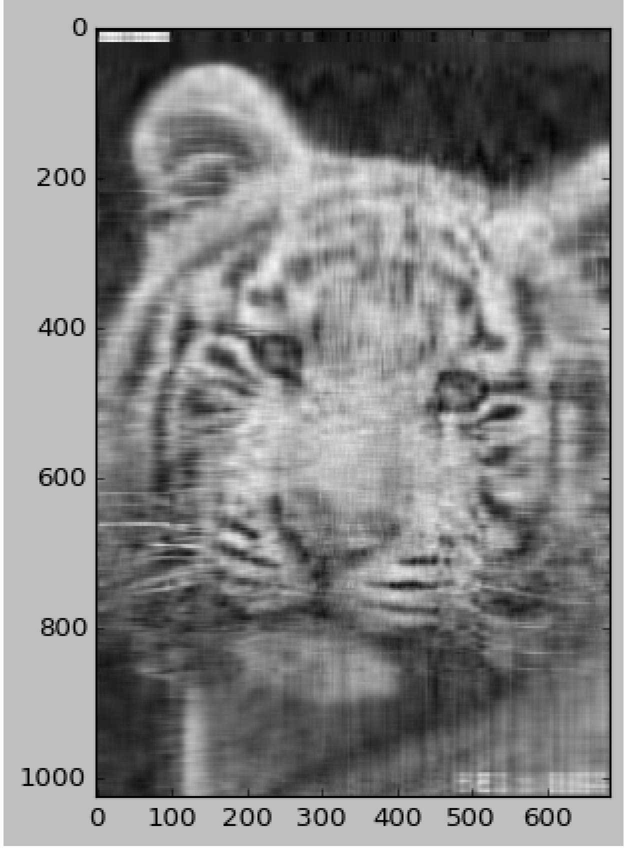
```
#imagen 3
plt.style.use('classic')
img = Image.open(urllib.request.urlopen('http://dummy-images.com/animals/dummy-683x1024-
imggray = img.convert('LA')
```

```
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)

# Compresión de imagen
U,D,V = np.linalg.svd(imgmat)
nvalue = 20

reconstimg = np.matrix(U[:,:nvalue])*np.diag(D[:nvalue])*np.matrix(V[:nvalue,:])
plt.imshow(reconstimg,cmap='gray')
plt.show()
print("imagen está comprimida")
```

```
[ 30.  25.  29. ... 147. 163. 165.]
```




```
<PIL.Image.Image image mode=LA size=683x1024 at 0x120C88460>
```




```
imagen está comprimida
```

Ejercicio 3

Feature importances

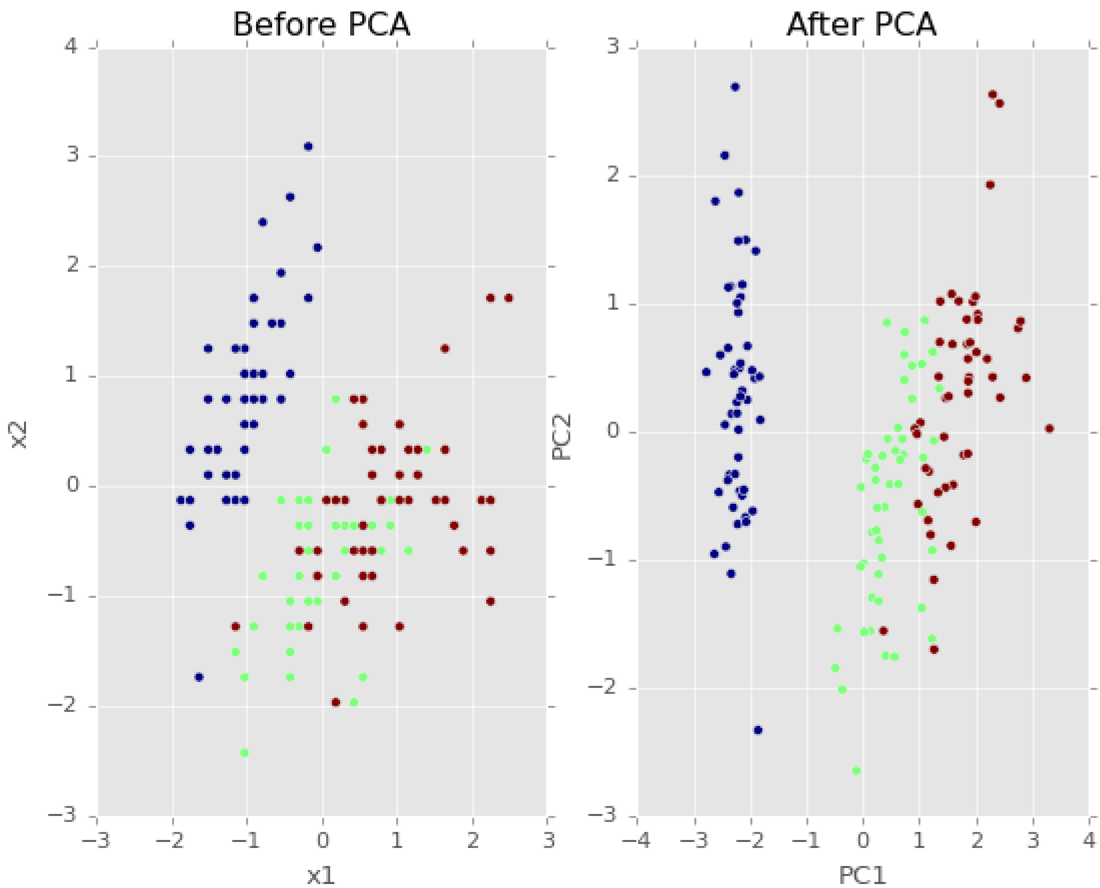
Para este ejercicio, te pediremos que sigas el tutorial de la siguiente pagina:

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

In [94]:
```
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
```

In [95]:
```
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 [96]:
```
print(pca.explained_variance_ratio_)
# array([0.72962445, 0.22850762])
```

```
[0.72962445 0.22850762]
```

6. Proof of eigenvalues of original covariance matrix being equal to the variances of the reduced space

In [99]:
```
np.cov(X_new.T)
#array([[2.93808505e+00, 4.83198016e-16],
#       [4.83198016e-16, 9.20164904e-01]])

# La diagonal de los datos en la matriz de covarianza es igual a los valores actuales de
```

Out[99]:
```
array([[2.93808505e+00, 5.33928780e-16],
       [5.33928780e-16, 9.20164904e-01]])
```

In [100…
```
# eigenvalues guardados en "pca.explained_variance_:"
pca.explained_variance_
```

Out[100]:
```
array([2.93808505, 0.9201649 ])
```

7. Feature importance

In [101…
```
# La importancia de cada característica se refleja en la magnitud de los valores corresp
# Estos son las caracteristicas mas importantes
print(abs( pca.components_ ))
#[[0.52106591 0.26934744 0.5804131 0.56485654]
```

```
# [0.37741762 0.92329566 0.02449161 0.06694199]]

# La primera fila en esta matriz es el PC1, el componente mas influyente
# podemos concluir que las características 1, 3 y 4 son las más importantes para PC1. De
```

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

8. The biplot

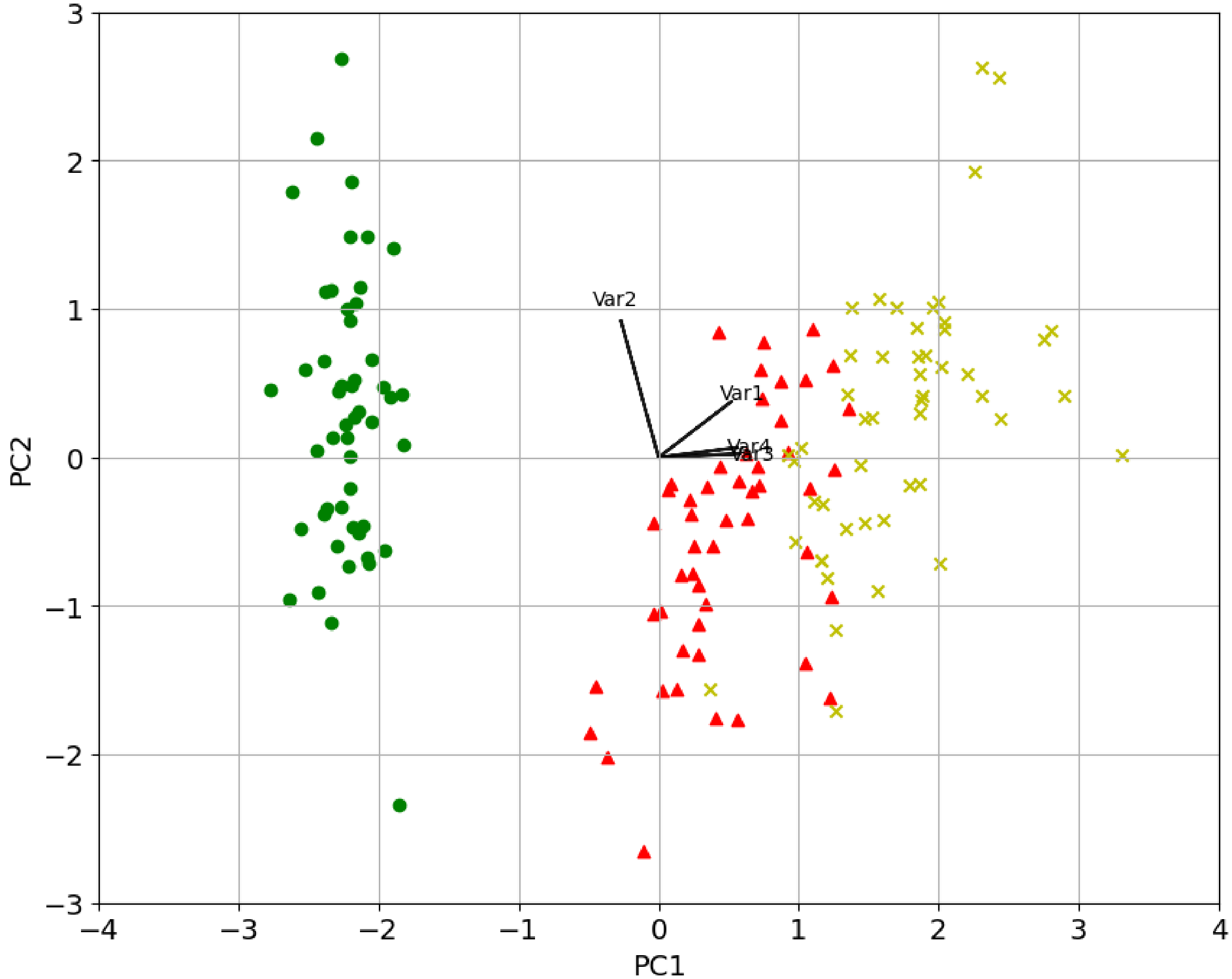
El biplot es la mejor manera de visualizar todo en uno después de un análisis PCA.
Hay una implementación en R pero no hay una implementación estándar en python, así que decidí escribir mi propia función para eso:

In [104…

```
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 o
    for i in range(n):
        #plot as arrows the variable scores (each variable has a score for PC1 and one f
        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)
```

In [105…

```
# Llame a la función (asegúrese de ejecutar primero los bloques iniciales de código dond
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()
```



In [107…

```
# Var 3 y Var 4 están extremadamente positivamente correlacionados
print(np.corrcoef(X[:,2], X[:,3])[1,0])
#0.9628654314027957
# Var 2and Var 3 are negatively correlated
print(np.corrcoef(X[:,1], X[:,2])[1,0])
#-0.42844010433054014
```

```
0.9628654314027957
-0.42844010433054014
```

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

De los mas relevante son los siguientes puntos:

1. *Principal Component Analsis* (PCA) es una tecnica de reducción dimensional, que identifica caracteristicas importantes entre las variables dadas a traves de combinaciones de las variables originales el resultado son componentes correlacionados cuyo peso tiende a repercutir en mayor medida a la desviacion estandar.

   Como resultado encontramos un compopnente con mayor relevancia a la hora de crear un modelo predictivo de datos.

2. Esta practica es util en casos:

   A. Donde existe multicolinealidad entre los datos.
   B. Cúando las dimensiones de los datos son altas.

C. Para la comprensión de los datos y eliminación del ruido.

3. En el caso de los datos z-scored por tener una desviación estandar de 1, la matriz de correlación será igual a la covarianza.

4. Los Eigenvalues son iguales a la matriz de varianza, ordenados en forma decreciente en la diagonal de la matriz.

5. La proyección (transformación) de los datos normalizados originales en el espacio PCA reducido se obtiene multiplicando (producto escalar) los datos normalizados originalmente por los vectores propios principales de la matriz de covarianza.

6. En la mayoria de los casos el PC1 tiene el mayor impacto en la varianza final del modelo, seguido por el PC2, PC3, ..., PC#, pero con mucha menor participacion en la varianza final, se debe elegir un numero estimado para continuar y no todos, ya que cierto porcentaje de la varianza puede ser desecho por ser muy pequeño, una regla no escrita es menor que el 5% puede ser despreciado.

7. Para cada componente es posible identificar las caracteristicas mas importantes, Cuanto mayores son estos valores absolutos, más contribuye una característica específica a ese componente principal.

Preguntas finales

¿Qué es feature importance y para que nos sirve?

Aportan el peso que tiene cada componente a la varianza, entre mas absoluto sea el valor mayor es la importancia del feature o caracteristica, no todos los features de un componente tienen la misma importancia.

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

Las relaciones entre variables y la comprencion de estas, debido que el ejercicio puede ser un poco complicado, pero separandolo en pasos e identificando las similitudes que existen entre la desvicion estandar, la varianza y covarianza entendí mejor la actvidad.

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

En conjuntos de datos con muchas caracteristicas o variables relacionadas, ya que nos permite averiguar relaciones entre ellas y los pesos o importancia de estas relaciones que son nombradas como componentes.