Nombre del estudiante: Diego Alonso Luna Ramirez

Matricula: A01793035

Nombre del trabajo: Actividad Semanal 5 Repaso Transformación y reducción de dimensiones

Fecha de entrega: 24 de octubre del 2022

Campus: Queretaro

Programa: Maestría en Inteligencia Artificial Aplicada (MNA-V)

Trimestre: Segundo

Materia: Ciencia y analítica de datos

Nombre del maestro: Maria de la Paz Rico Fernandez

Haz doble clic (o pulsa Intro) para editar

Bienvenido al notebook

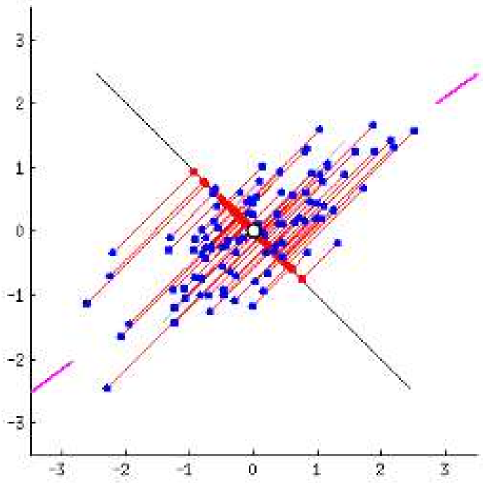
Repaso de Reducción de dimensiones

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 :)

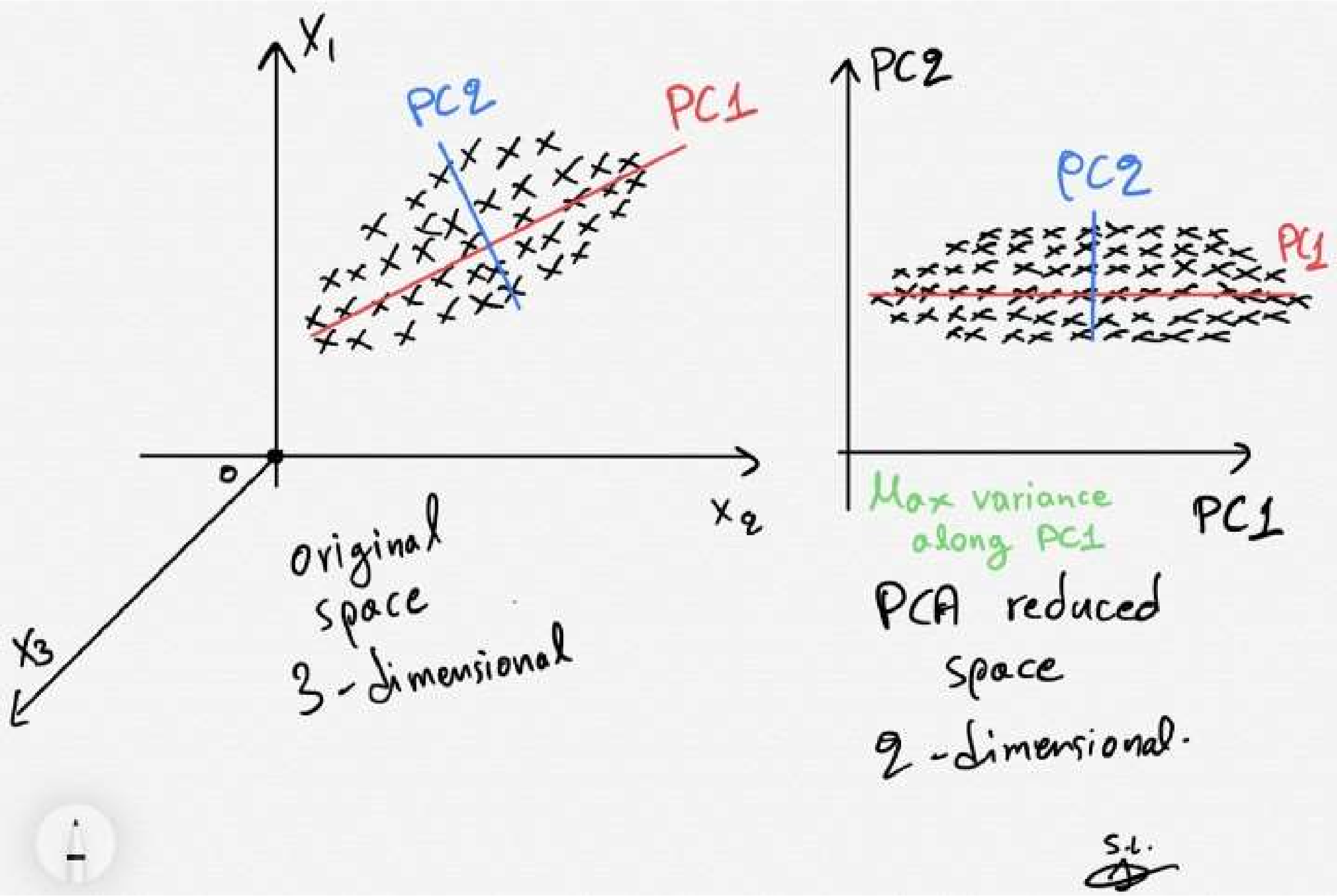
$$\mathbf{C} = \frac{\mathbf{X}^\top \mathbf{X}}{n-1}$$

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

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

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

```
from numpy.linalg.linalg import inv
###-----------------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)
print("------------------------------")

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*D*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-------
[[ 1.80558242e+00  0.00000000e+00 -0.00000000e+00]
 [ 0.00000000e+00 -2.41260114e-02 -0.00000000e+00]
 [-0.00000000e+00 -0.00000000e+00 -2.17279621e-16]]
----------------------------
```

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

```
print(vectors) #W
print("-------Matriz reconstruida-------")
W = vectors
Winv = np.linalg.inv(W)
D = np.diag(values)
B= W*D*Winv
print(B)
print("----------------------------")
```

```
[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-------
[[ 2.72467795+4.91899789e-17j  0.        -0.00000000e+00j
  -0.        +0.00000000e+00j]
 [ 0.        +0.00000000e+00j -0.17129809+8.58873762e-01j
  -0.        +0.00000000e+00j]
 [ 0.        -0.00000000e+00j  0.        -0.00000000e+00j
   0.2613813 -7.69686255e-01j]]
----------------------------
```

```
A = array([[1, 3, 8], [2, 0, 0], [0, 0, 1]])
values, vectors = eig(A)
print(values) #D
print(vectors) #W
print("-------Matriz reconstruida-------")
W = vectors
Winv = np.linalg.inv(W)
D = np.diag(values)
B= W*D*Winv
print(B)
print("----------------------------")
```

```
[ 3. -2.  1.]
[[ 0.83205029 -0.70710678 -0.42399915]
 [ 0.5547002   0.70710678 -0.8479983 ]
 [ 0.          0.          0.31799936]]
-------Matriz reconstruida-------
[[ 1.8 -0.  -0. ]
 [-0.  -1.2 -0. ]
 [ 0.   0.   1. ]]
----------------------------
```

```
A = array([[5, 4, 0], [1, 0, 1], [10, 7, 1]])
values, vectors = eig(A)
print(values) #D
print(vectors) #W
print("-------Matriz reconstruida-------")
W = vectors
Winv = np.linalg.inv(W)
D = np.diag(values)
#TU CODIGO AQUI------------------------------------
B= W*D*Winv
```

```
print(B)
print("----------------------------")
```

```
[ 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-------
[[ 4.30570854  0.          0.        ]
 [ 0.          0.41184893  0.        ]
 [-0.          0.          0.0296579 ]]
----------------------------
```

¿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.

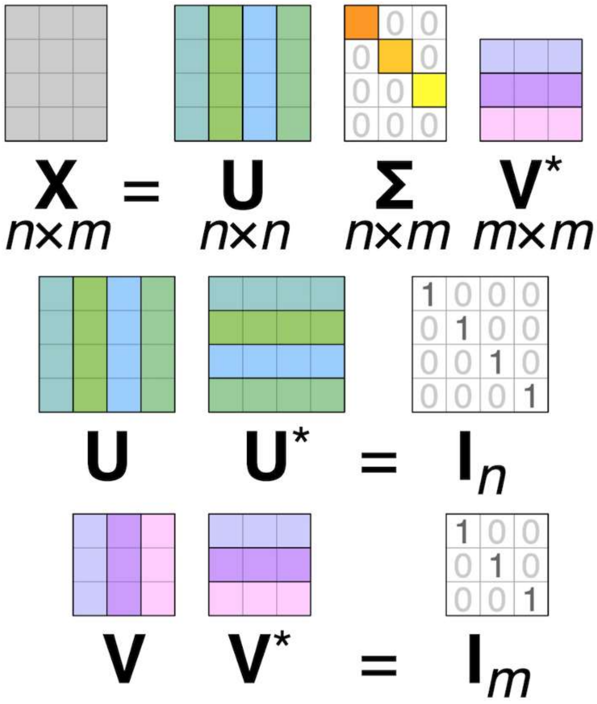
Es decir: Unidades-PC PC-Unidades

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

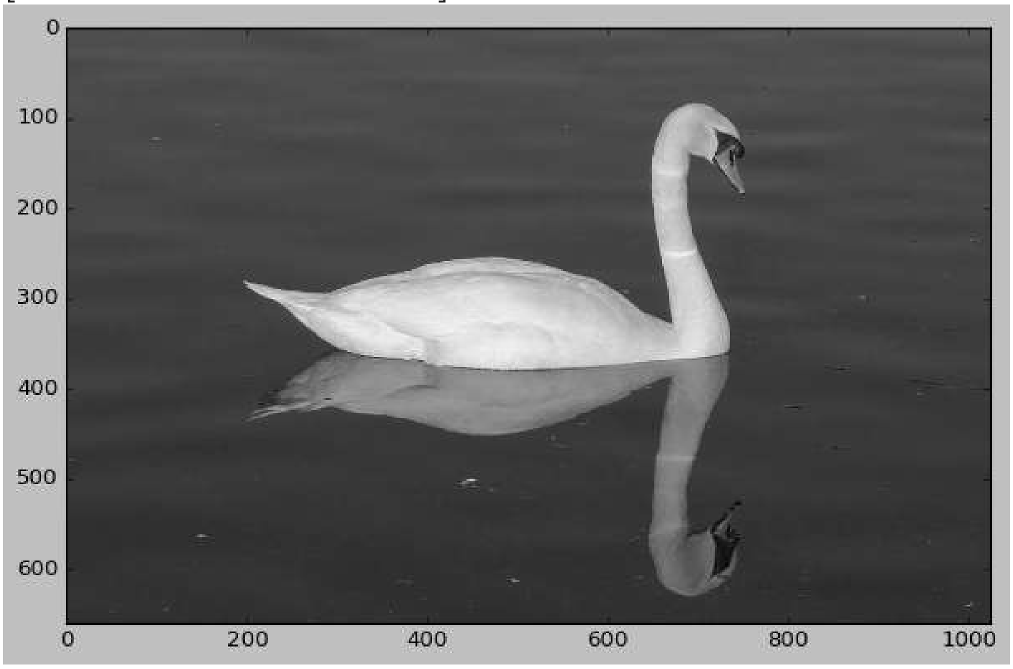
Juega con Lucy, una cisne, ayudala a encontrar cuantos valores singulares necesita para no perder calidad a través de SVD. Posteriormente usa 3 imágenes de tu preferencia y realiza la misma acción :D

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

Haz doble clic (o pulsa Intro) 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/uploa
#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 0x7F9688413ED0>
```

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

```
    (660, 1024)
```

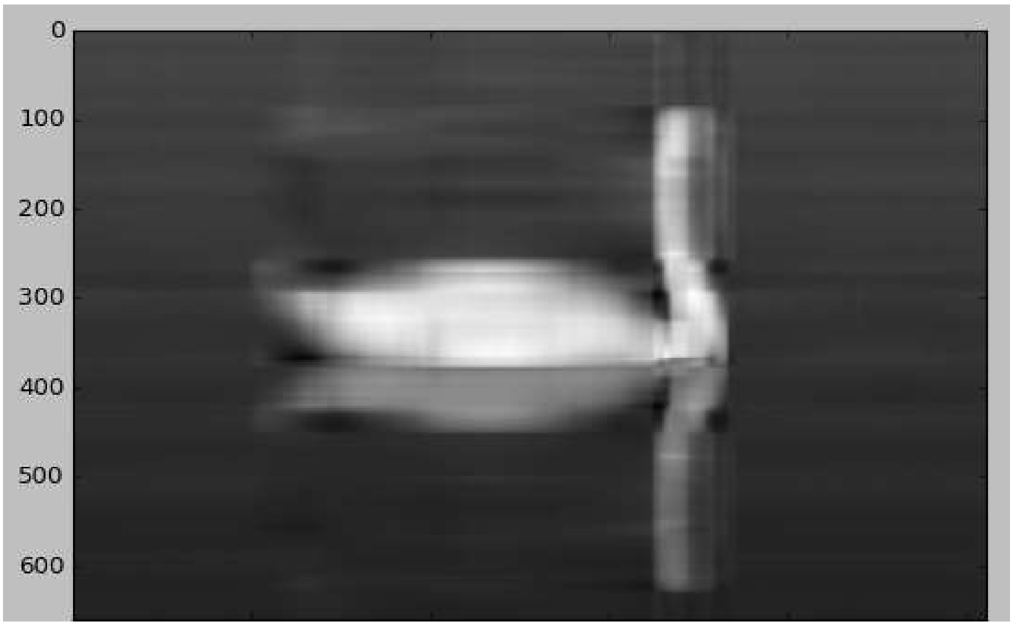
```
U.shape
```

```
    (660, 660)
```

```
V.shape
```

```
    (1024, 1024)
```

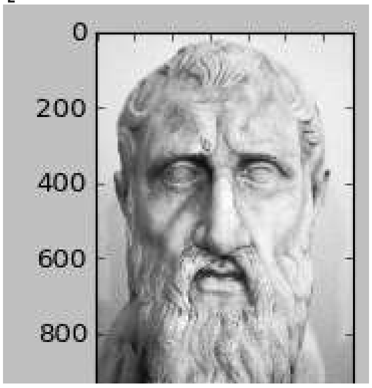
```
#Cuantos valores crees que son necesarios?
#A=U*D*V
#aqui los elegiremos---------------------------
# por las dimensiones de este caso en particular
#iremos de 0-660, siendo 660 como normalmente están los datos
#con 50 podemos observar que Lucy se ve casi igual, es decir conservamos aquello que en
# 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 = 5
#-----------------------------
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")
```



¡Ahora es tu turno!, comprime 3 imagenes

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

print(imgmot)

imgmot.shape = (imggray.size[1],imggray.size[0])
plt.figure(figsize=(6,3))
plt.imshow(imgmot,cmap='gray')
plt.show()
#-----------------------------------------------------
U,D,V = np.linalg.svd(imgmot)

nvalue = 20
#-------------------------------
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")
```

```
[212. 215. 217. ... 127. 120. 120.]
```

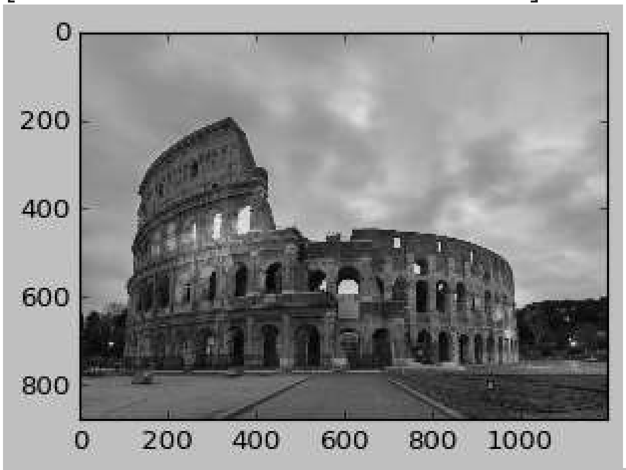


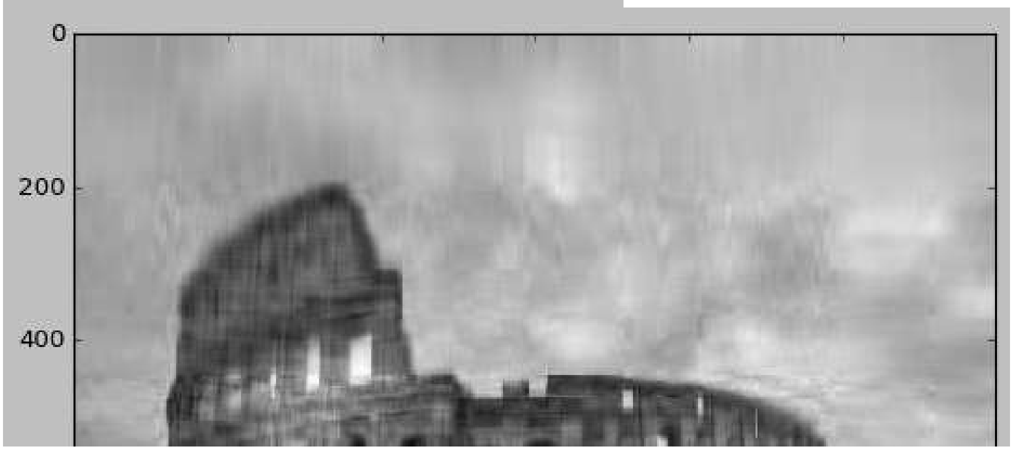

```
plt.style.use('classic')
img = Image.open(urllib.request.urlopen('https://biblioteca.acropolis.org/wp-content/uploa
imggray = img.convert('LA')
imgmot = np.array(list(imggray.getdata(band=0)),float)

print(imgmot)

imgmot.shape = (imggray.size[1],imggray.size[0])
plt.figure(figsize=(6,3))
plt.imshow(imgmot,cmap='gray')
plt.show()
#-----------------------------------------------------
U,D,V = np.linalg.svd(imgmot)

nvalue = 20
#-------------------------------
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")
```

```
[ 16.  17.  17. ... 151. 148. 147.]
```


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

print(imgmot)

imgmot.shape = (imggray.size[1],imggray.size[0])
plt.figure(figsize=(6,3))
plt.imshow(imgmot,cmap='gray')
plt.show()
#-----------------------------------------------------
U,D,V = np.linalg.svd(imgmot)

nvalue = 20
#-------------------------------
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")
```

```
[151. 151. 151. ...  67.  67.  70.]
```





Ejercicio 3

Feature importances

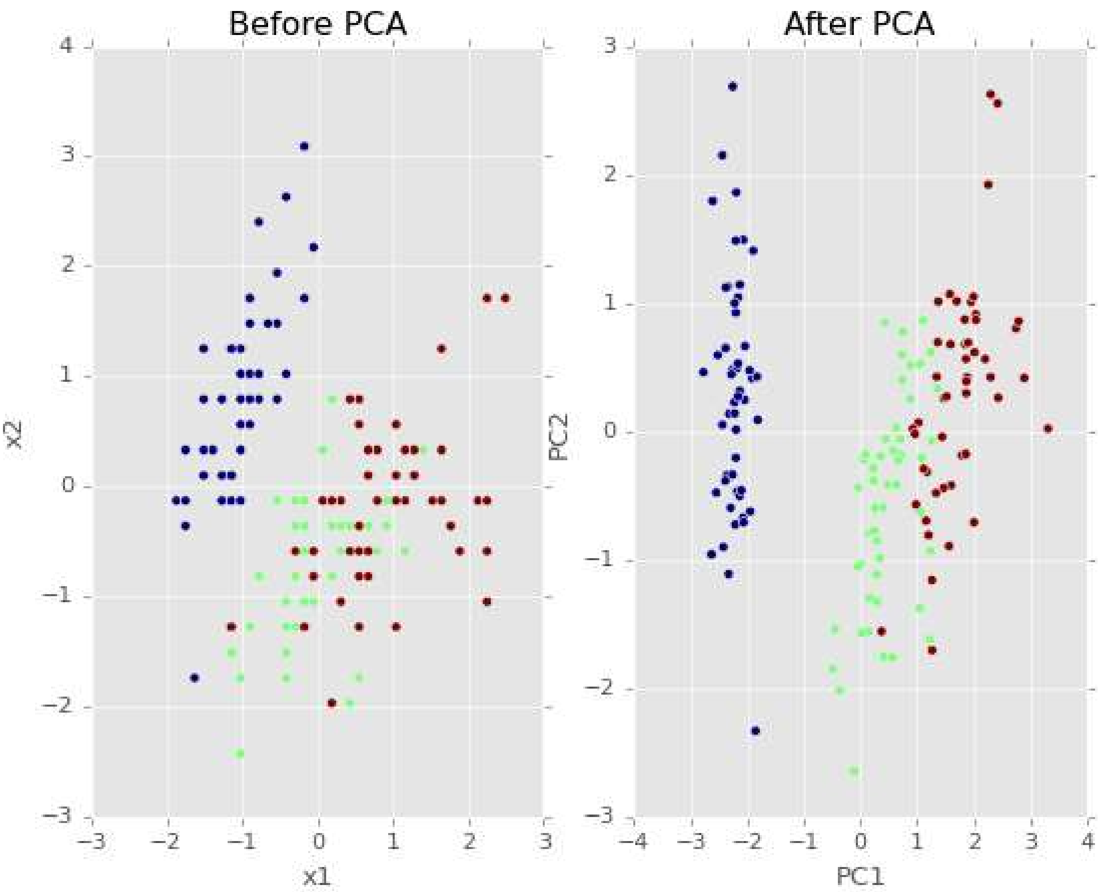
Para este ejercicio, te pediremos que sigas el tutorial de la siguiente pagina:

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')

#Cargar datos
iris = datasets.load_iris()
X = iris.data
y = iris.target

#Z-puntuación de las caracteristicas
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)
```

```
#The PCA model
pca = PCA(n_components=2) #Estimación de solo 2 PCs
X_new =  pca.fit_transform(X) #Proyectar los daots originales dentro del espacio 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('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]
```

```
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]])
```

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

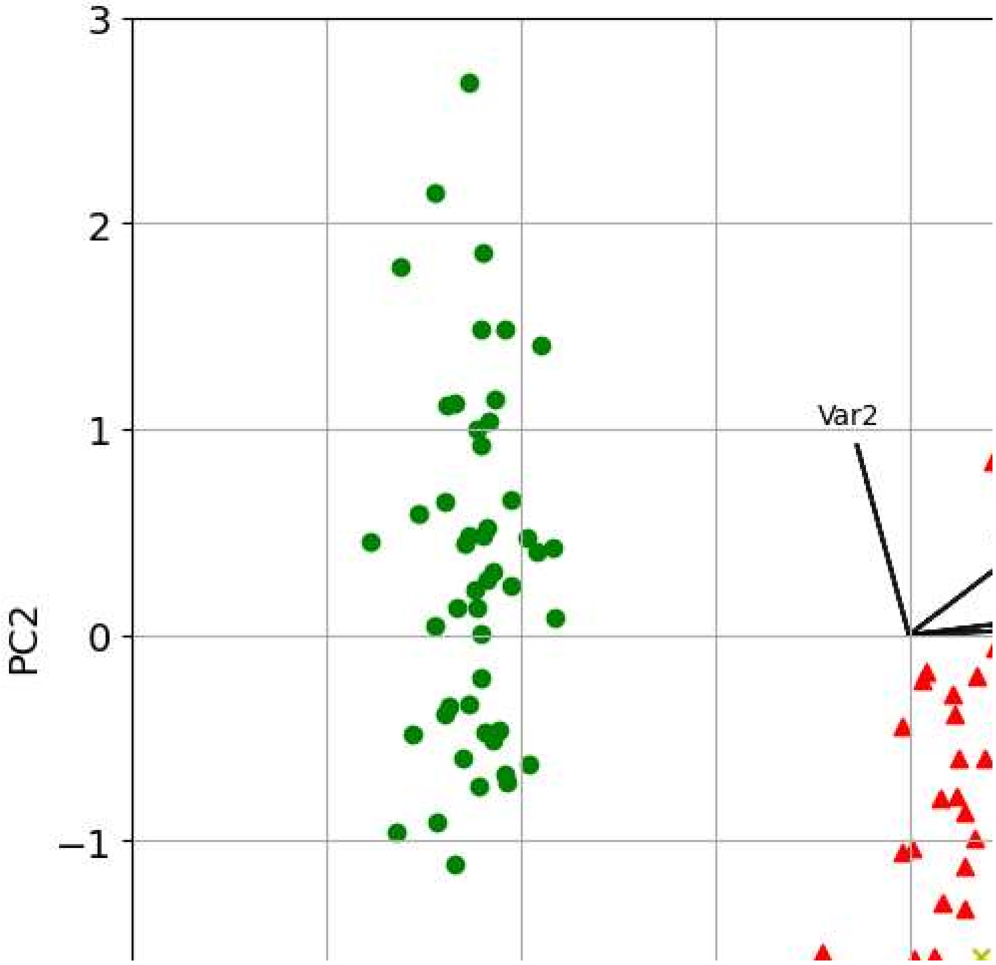
```
array([2.93808505, 0.9201649 ])
```

```
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):

    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
    for i in range(n):
        #plot as arrows the variable scores (each variable has a score for PC1 and one for
        plt.arrow(0, 0, coeff[i,0], coeff[i,1], color = 'k', alpha = 0.9,linestyle = '-',l
        plt.text(coeff[i,0]* 1.15, coeff[i,1] * 1.15, "Var"+str(i+1), color = 'k', ha = 'c

    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

     -0.42844010433054014
```

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

¿Qué es feature importance y para que nos sirve? Se refiere a las técnicas que asignan una puntuación a las características de entrada en función de su utilidad para predecir una variable de destino. Es importante para proyectos de modelado predictivo, lo que incluye proporcionar información sobre los datos, información sobre el modelo y la base para la reducción de la dimensionalidad y la selección de características que pueden mejorar la eficiencia y la eficacia de un modelo predictivo sobre el problema.

¿Qué hallazgos fueron los más relevantes durante el análisis del ejercicio? Las graficas son fundamentales, es la mejor manera de visualizar todo en uno después de un análisis PCA. Ahora sé, que algebra lineal, existen los eigenvectores de un operador lineal son los vectores no nulos que, cuando son transformados por el operador, dan lugar a un múltiplo escalar de sí mismos, con lo que no cambian su dirección.

Tambien que el PCA es una técnica de reducción que construye variables relevantes a través de combinaciones lineales, PCA lineal.

¿Dónde lo aplicarías o te sería de utilidad este conocimiento? PCA se puede utilizar cuando las dimensiones de las características de entrada son altas, también se puede utilizar para la eliminación de ruido y la compresión de datos, como podemos visualizarlo en la segunda actividad de las imagenes, las cuales, ayuda a comprender facilmente.