Double-click (or enter) to edit

Bienvenido al notebook

Repaso Transformación y reducción de dimensiones Jhon Edison Munoz Burgos A01793659 Ciencia Analitica de Datos Octubre -2023

> Indented block

> Indented block

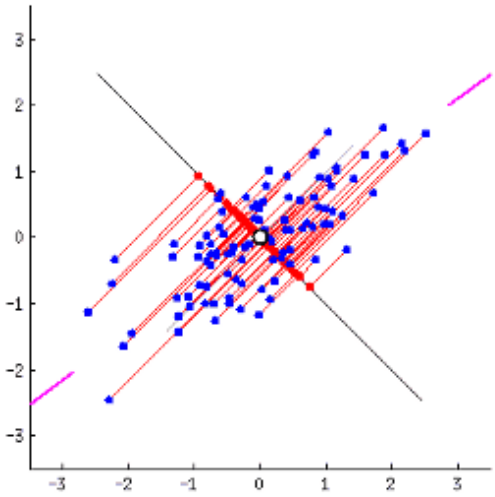
Repaso de Reducción de dimensiones

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 :)

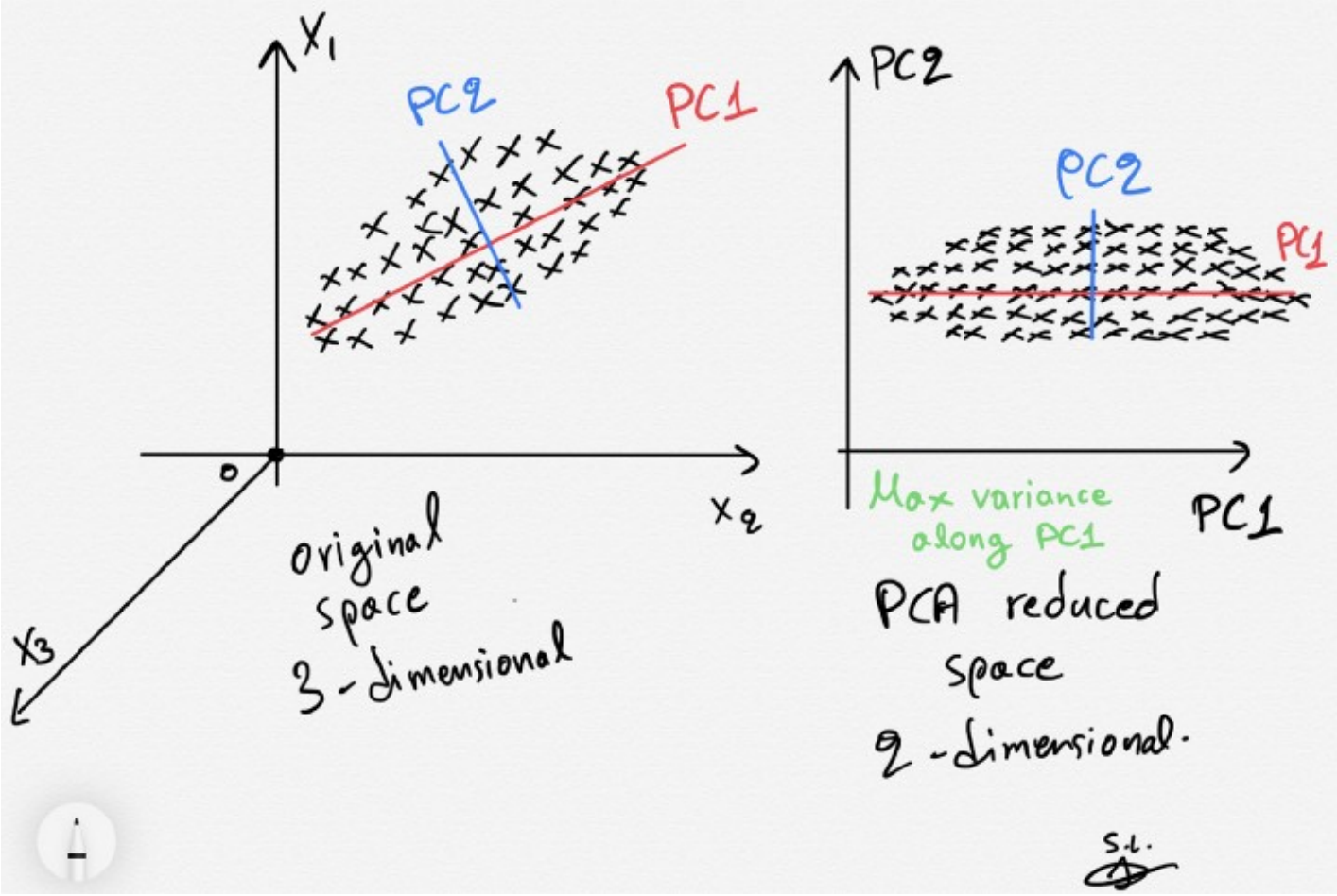
$$\mathbf{C} = \frac{\mathbf{X}^\top \mathbf{X}}{n-1}$$

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

```
###-----------------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-----------------------------------
x=np.dot(W,D)
B=np.dot(x,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.]]
-----------------------------
```

```
#Matriz 1
# define la matriz
A = array([[3, 0, 2], [3, 0, -2], [0, 1, 1]])
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-------")
x=np.dot(W,D)
B=np.dot(x,Winv)
print(B)
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-------
      [[ 3.00000000e+00+1.12272023e-16j  3.36536354e-16-9.67833078e-17j
         2.00000000e+00-3.95053829e-17j]
       [ 3.00000000e+00-1.66253281e-16j  9.99200722e-16+1.12958209e-16j
        -2.00000000e+00+2.77350775e-17j]
       [ 1.11022302e-16+1.03283117e-18j  1.00000000e+00-7.61630485e-17j
         1.00000000e+00+1.02438275e-16j]]
      ----------------------------
```

```
#Matriz 2
# define la matriz
A = array([[1, 3, 8], [2, 0, 0], [0, 0, 1]])
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-------")
x=np.dot(W,D)
B=np.dot(x,Winv)
print(B)
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-------
      [[1.00000000e+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]]
      ----------------------------
```

```
#Matriz 3
```

```
# define la matriz
A = array([[5, 4, 0], [1, 0, 1], [10, 7, 1]])
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-------")
x=np.dot(W,D)
B=np.dot(x,Winv)
print(B)
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-------
[[ 5.00000000e+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.

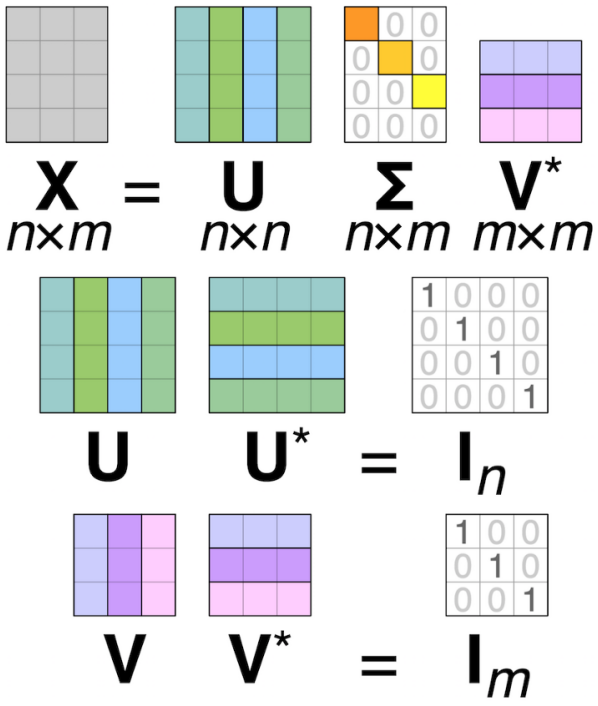
Es decir: Unidades-PC PC-Unidades

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

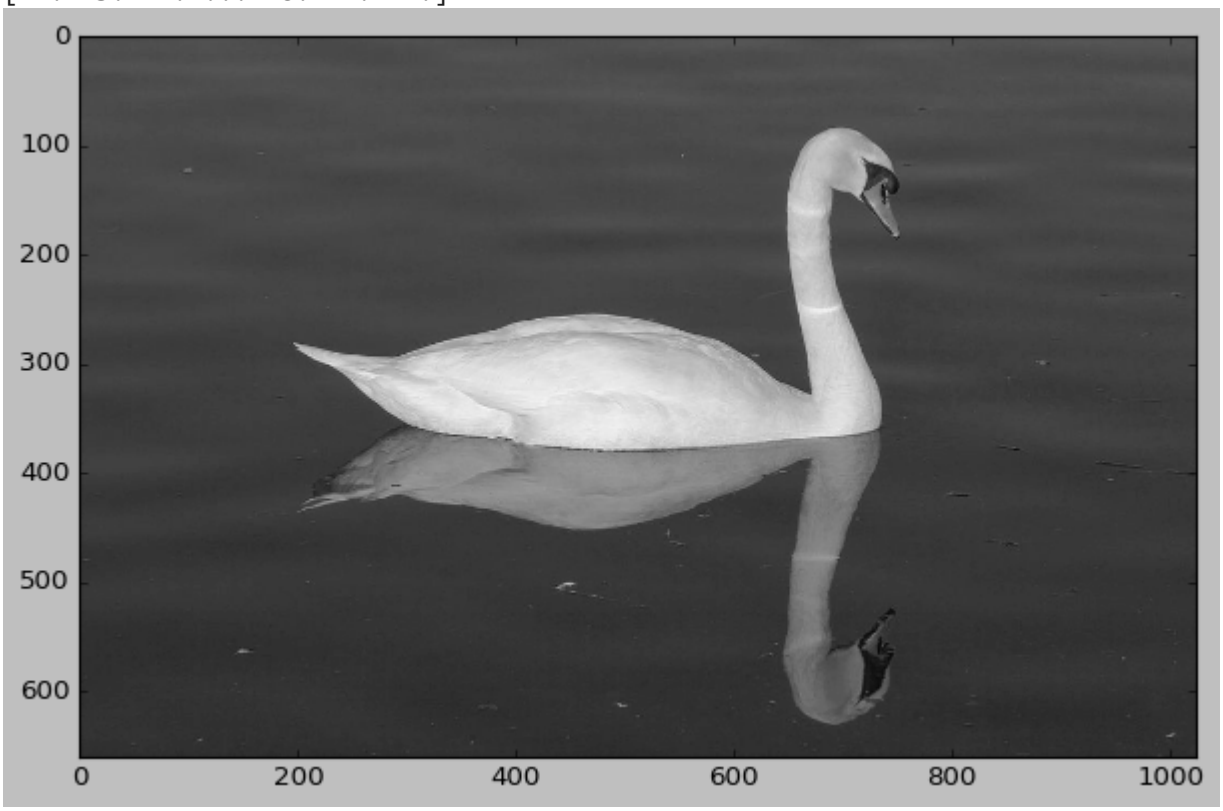
A esto se le llama **compresión de imagenes** :o

Double-click (or enter) to edit

```
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 0x7FA2E25D5110>
```

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

```
(660, 1024)
```

```
U.shape
```

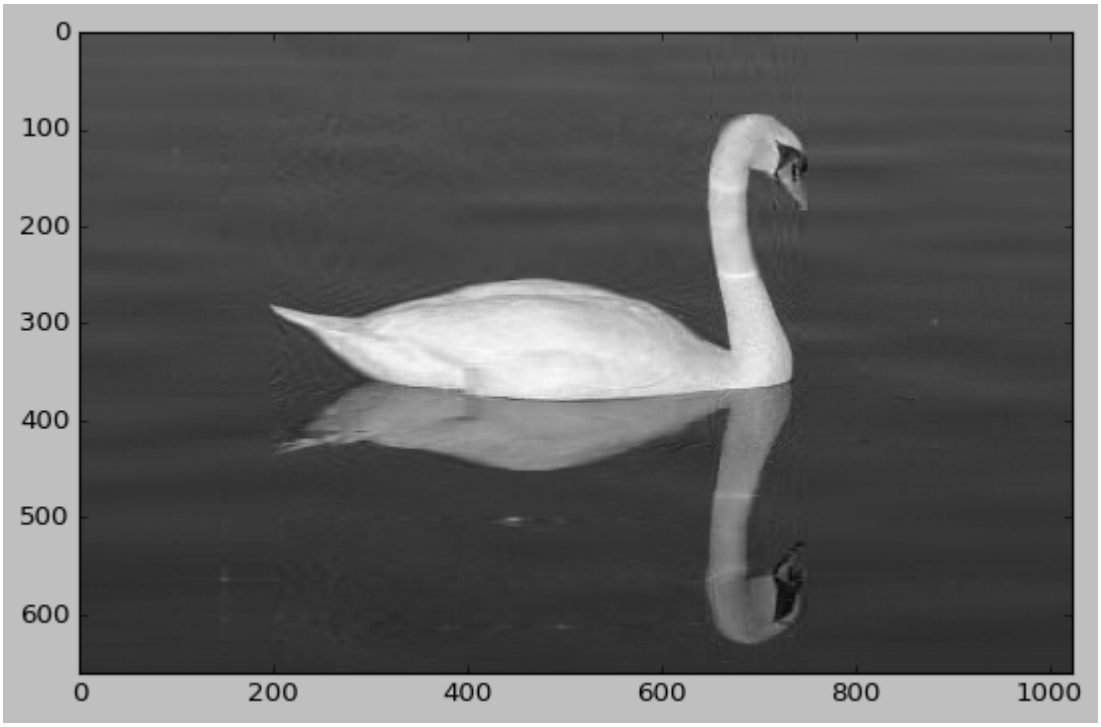
```
(660, 660)
```

```
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 = 40
#------------------------------
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
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/
imggray = img.convert('LA')
```

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

print(imgmat)
print("Imagen original")

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)
# Se imprime el tamaño de las matrices
print(imgmat.shape)
print (U.shape)
print(V.shape)
nvalue = 40
#------------------------------
reconstimg = np.matrix(U[:,:nvalue])*np.diag(D[:nvalue])*np.matrix(V[:nvalue,:])
plt.imshow(reconstimg,cmap='gray')
plt.show()
print("Felicidades la imagen está comprimida")
```

```
[ 3.  4.  5. ... 45. 45. 45.]
Imagen original
```


```
#imagen 2
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/
imggray = img.convert('LA')
imgmat = np.array(list(imggray.getdata(band=0)),float)

print(imgmat)
print("Imagen original")

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)
# Se imprime el tamaño de las matrices
print(imgmat.shape)
print (U.shape)
print(V.shape)
nvalue = 100
#------------------------------
```

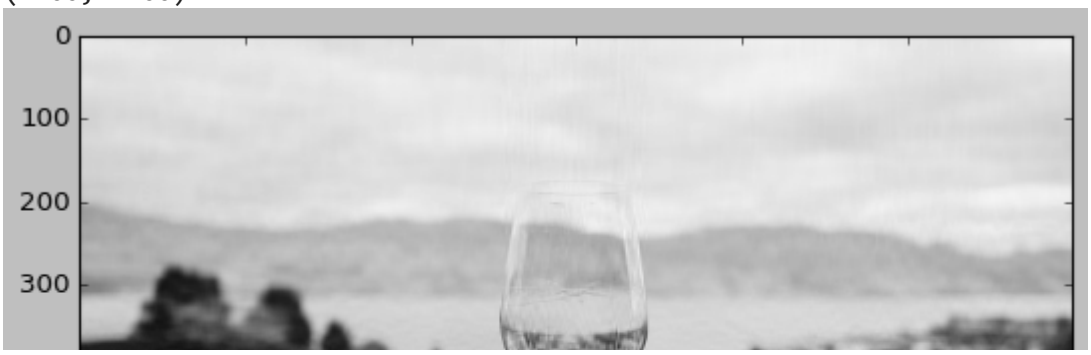
```
    reconstimg = np.matrix(U[:,:nvalue])*np.diag(D[:nvalue])*np.matrix(V[:nvalue,:])
    plt.imshow(reconstimg,cmap='gray')
    plt.show()
    print("Felicidades la imagen está comprimida")
```

```
[188. 188. 189. ... 187. 185. 188.]
Imagen original
```


```
#imagen 3
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/
imggray = img.convert('LA')
imgmat = np.array(list(imggray.getdata(band=0)),float)

print(imgmat)
print("Imagen original")

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)
# Se imprime el tamaño de las matrices
print(imgmat.shape)
print (U.shape)
print(V.shape)
nvalue = 60
#------------------------------
reconstimg = np.matrix(U[:,:nvalue])*np.diag(D[:nvalue])*np.matrix(V[:nvalue,:])
plt.imshow(reconstimg,cmap='gray')
plt.show()
print("Felicidades la imagen está comprimida")
```

```
[244. 244. 244. ...  28.  28.  28.]
Imagen original
```


```
<PIL.Image.Image image mode=LA size=1200x800 at 0x7FA2D8B54490>
(800, 1200)
(800, 800)
(1200, 1200)
```



Ejercicio 3

Feature importances

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

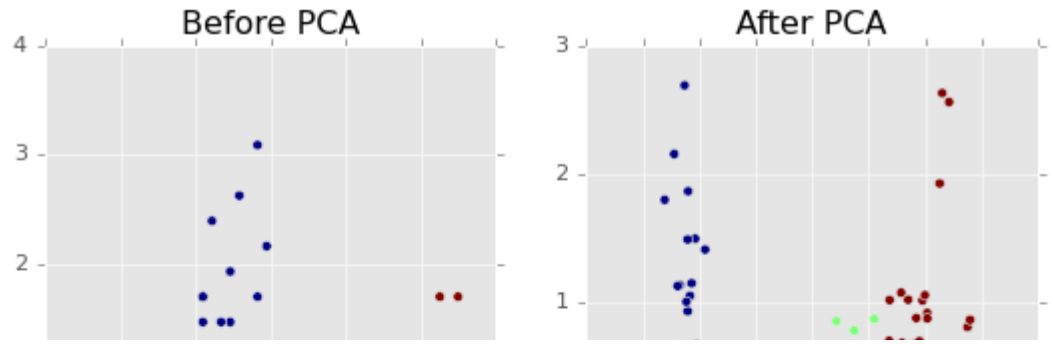
https://towardsdatascience.com/pca-clearly-explained-how-when-why-to-use-it-and-feature-importance-a-guide-in-python-7c274582c37e

```
Felicidades la imagen está comprimida
```

```
#tu codigo aqui
```

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

```
#tu codigo aqui
#Importamos Librerias
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   Cargamos los datos de la base de datos
iris = datasets.load_iris()
X = iris.data
y = iris.target
# Z-score the features  Puntuacion z de las caracteristicas
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)
# The PCA model  Modelo PCA
pca = PCA(n_components=2) # estimate only 2 PCs
X_new = pca.fit_transform(X) # project the original data into the PCA space
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])    #Se puede explicar que en el espacio de PCA, la varianza
```

```
[0.72962445 0.22850762]
```

```
np.cov(X_new.T)  #Prueba Maxima de Varianza  tambien se puede ver estimando la matriz de cova
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]])
```

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

```
pca.explained_variance_   # Observamos estos valores(en la diagonal tenemos las varianzas) so
array([2.93808505, 0.9201649])
```

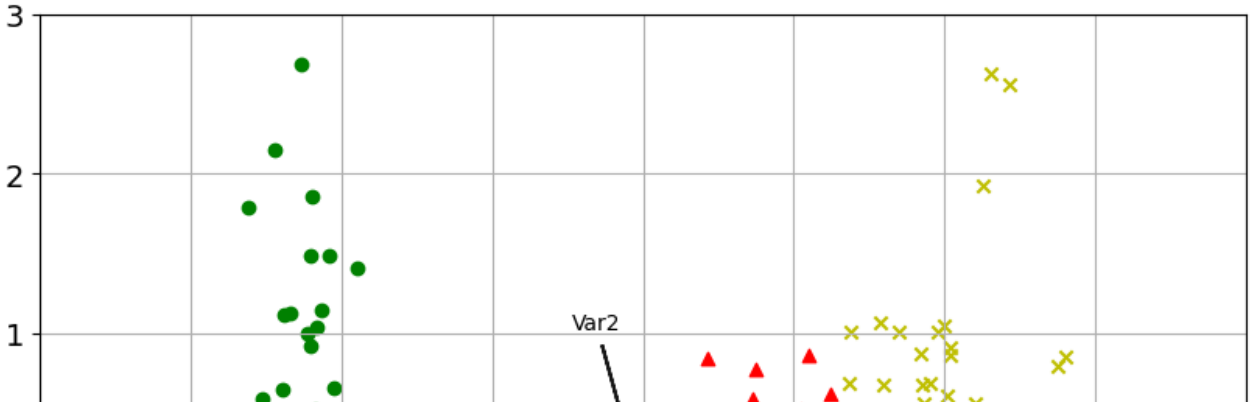
```
array([2.93808505, 0.9201649 ])
```

```
print(abs( pca.components_ ))    # Se puede concluir q las caracteristicas 1,3y 4 son mas imp
```

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

```
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 puede verificar que las variables 1,3 y4 son mas improtantes para PC1 , Y posteriomente l
```




```
# Var 3 and Var 4 are extremely positively correlated    #Se observa que la variable 3 y 4 so
np.corrcoef(X[:,2], X[:,3])[1,0]
0.9628654314027957
```

```
0.9628654314027957
```

```
# Var 2and Var 3 are negatively correlated    # Se observa que la variable 2y 3 No son correl
np.corrcoef(X[:,1], X[:,2])[1,0]
-0.42844010433054014
```

```
-0.42844010433054014
```

Justificación Ejercicio:

1.Realiza un comentario relacionado a los pasos que se llevaron a cabo en este proceso de features importances.

Ejercicio 1.

Por medio del ejercicio se puede comprender la función de las eigenvalores, donde los datos se transforman en otros componentes los cuales contienen la misma información la cual nos permite tomar decisiones. Cuando descomponemos la matrix y luego debemos recostruirla se observó que en algunos casos la matrix habia sido modificada minimamente pero la información continuaba siendo la misma.

Ejercicio 2.

Por medio de este ejercio se puede ver de una manera visual, como funcionan los componentes principales en la transformación por medio de comprimir la imagen, se observa que cuando N es cada vez menos la imagen pierde sus propiedades se observa distorcionada o pixelada, se observa que hay un N que es optimo para conservar todas las caracteristicas.

Ejercicio 3.

Este ejercicio nos permite conocer como es la transformación PCA. Se observa que hubo un cambio en 2 de sus componentes principales se puede explicar que en el espacio de PCA, la varianza maxima a lo largo del PC1 explica el 73% y PC2 el 22% apra tener una varianza del 95%, se puede observar que algunas variables tienen mayor influencia para el PC1 y otras para PC2, al igual podemos observar la correlación entre ellas . si son positivamete correlacionadas o negativamente correlaciones.

¿Qué es feature importance y para que nos sirve?

El feature importance nos permite identificar y validar cada atributo, con el objetivo de conocer sus pesos evaluar su correlación, nos permite reducir los datos sin afectar su importancia y lo que se quieren transmitir, tener predicciones claves y asertadas, al disminuir la dimensionalidad nuetro proyecto sera menos complejo lo cual permite entrenamientos mas rapidos y eficaces.

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

- Por medio de PCA, se realiza una reducción de dimensionalidad, y se basa en un aprendizaje no supervisado.
- Se puede obtener la misma información sin tener gran afectación en los datos realizando una comprensión de los mismos.
- El PCA proyecta los datos en un hiperplano y selecciona la poryeccion que conserva la maxima cantidad de varianza. Con menos datos.
- Por medio del PCA reduce la dimensionalidad del conjunto de entrenamiento antes de entrenarlo lo cual ayuda a que el entrenamiento sea mas rapido y eficiente.
- Se puede evaluar la correlación entre las variables.

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

-PCA se puede aplicar cuando hay muchas caracteristicas las cuales pueden ser miles o millones para cada instancia de entrenamiento, al tener estas caracteristicas hace que el entrenamiento sea lento y encontrar una buena solución puede ser dificil en estos casos PCA seria un metodo optimo.

- Este conocimiento es util al momento de visualizar datos , debido a que reduce la dimensionalidad de los mismos.
- Este conocimiento nos sirve para evaluar la correlacion entre las variables.