

Alumno: Luis Jose Navarrete Baduy

Matricula: A01793919

Profesora: María de la Paz Rico Fernández

## Bienvenido al notebook

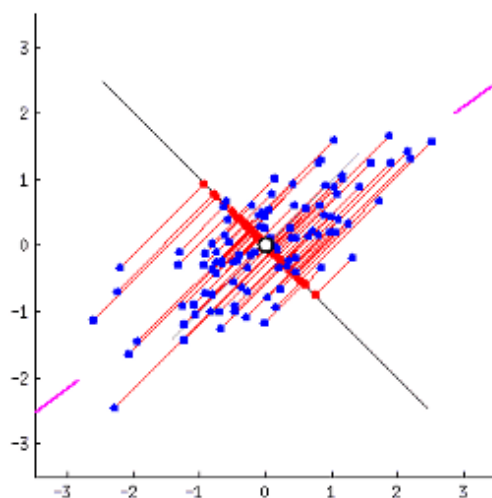
## Repaso de Reducción de dimensiones

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 :)

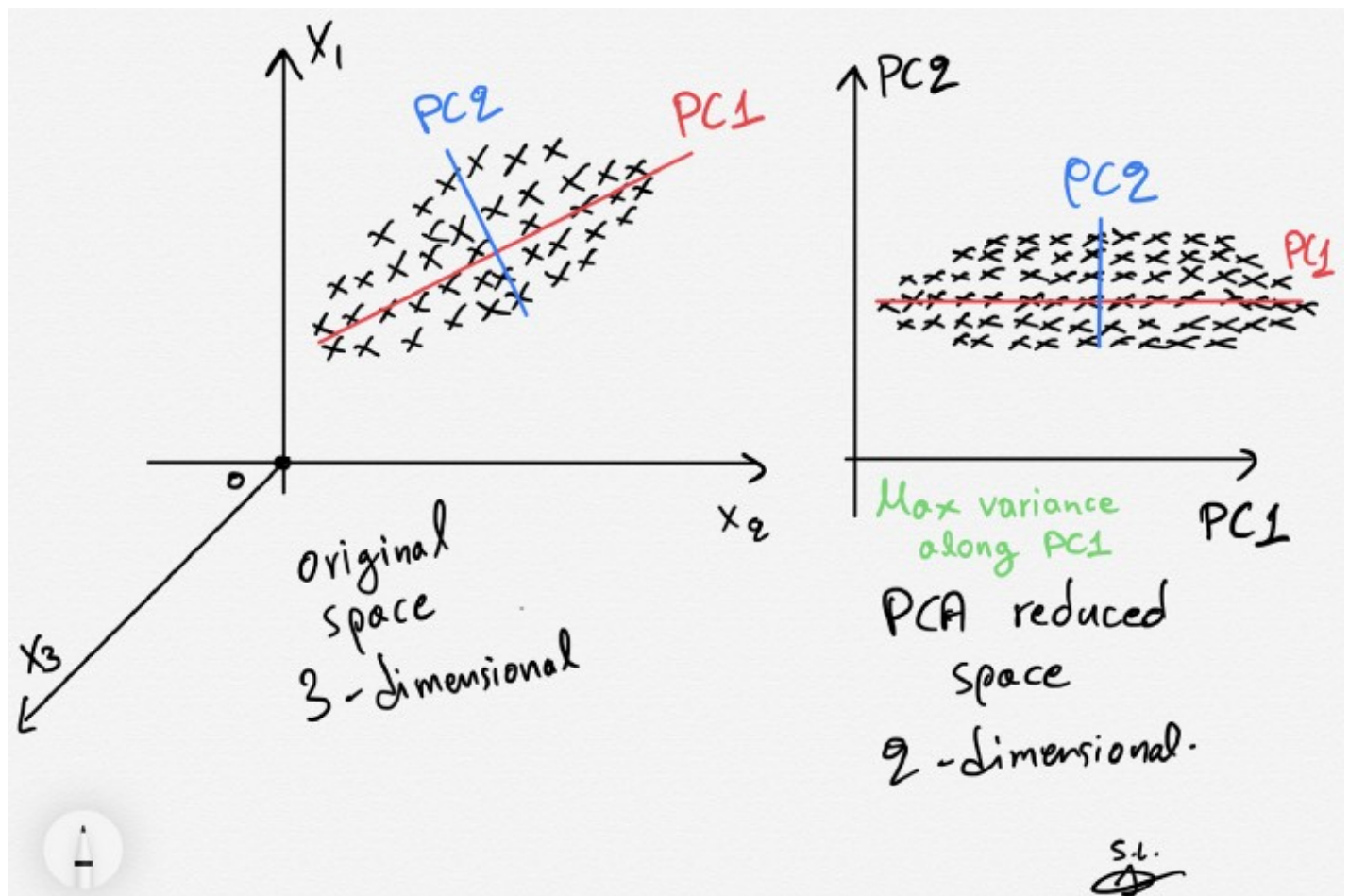
$$\mathbf{C} = \frac{\mathbf{X}^T \mathbf{X}}{n - 1}$$

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

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

y reconstruye la matriz original a través de las matrices  $W D W^{-1}$  (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-----
B=np.dot(np.dot(W,D),Winv).astype(int)
print(B)
print("-----")
```

```

<class 'numpy.ndarray'>
-----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]
 [3 4 6]
 [6 7 9]]
-----

```

```

def get_matrix_reconstruction(M):
    print("-----Matriz original-----")
    print(M)
    print("-----")
    values, vectors = np.linalg.eig(M)
    print(values) #D
    print(vectors) #W
    W = vectors
    Winv = np.linalg.inv(W)
    D = np.diag(values)
    print("-----Matriz reconstruida-----")
    B=np.dot(np.dot(W,D),Winv)
    print(B)
    print("-----")

```

```

#Matriz 1
A = array([[3, 0, 2], [3, 0, -2], [0, 1, 1]])
get_matrix_reconstruction(A)

```

```

-----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
A = array([[1, 3, 8], [2, 0, 0], [0, 0, 1]])
get_matrix_reconstruction(A)

-----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
A = array([[5, 4, 0], [1, 0, 1], [10, 7, 1]])
get_matrix_reconstruction(A)

-----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.

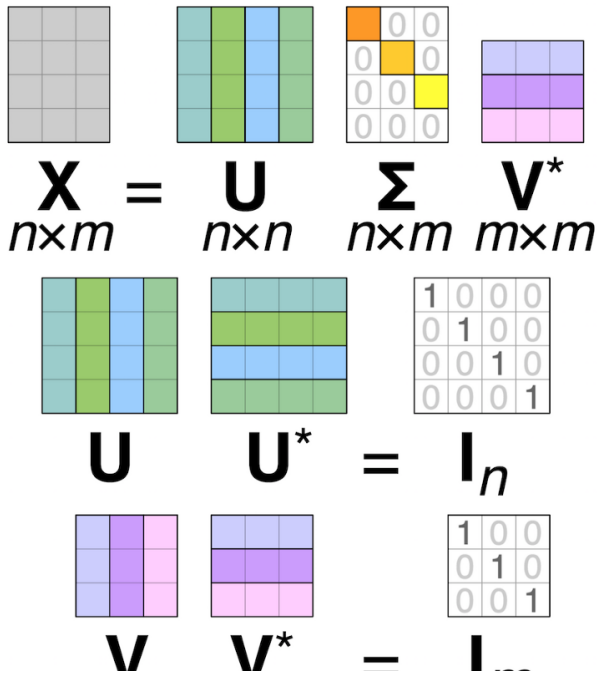
Es decir: Unidades-PC PC-Unidades

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.



The diagram illustrates the SVD decomposition of a matrix  $X$  into three components:  $U$ ,  $\Sigma$ , and  $V^*$ .

**Top Row:** A 4x4 matrix  $X$  is shown as a grid of gray squares. It is equal to the product of a 4x4 matrix  $U$  (green and blue squares), a 4x4 matrix  $\Sigma$  (diagonal matrix with orange, yellow, and gray squares), and a 4x4 matrix  $V^*$  (purple and pink squares).

**Bottom Row:** The matrices  $U$  and  $U^*$  are shown as grids of green and blue squares. They are equal to the identity matrix  $I_n$  (a 4x4 grid of 1s and 0s).

**Bottom Row:** The matrices  $V$  and  $V^*$  are shown as grids of purple and pink squares. They are equal to the identity matrix  $I_m$  (a 4x4 grid of 1s and 0s).

## ▼ 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 ingresa) 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/1
#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 0x7FD7FCA916D0>
```

```
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 e
```

```

# 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 = 660
#-----
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

```

def image_reduction(image_url, nvalue=50):
    plt.style.use('classic')
    img = Image.open(urllib.request.urlopen(image_url)).convert('LA')
    #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)
U,D,V = np.linalg.svd(imgmat)
#-----
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")

```

```

#imagen 1
url = 'https://external-content.duckduckgo.com/iu/?u=https%3A%2F%2Fcdn.vox-cdn.com%2F'
image_reduction(url, 100)

```



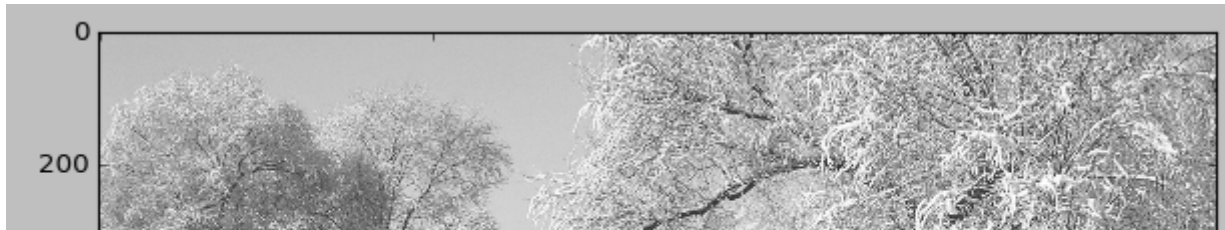
#imagen 2

```
url = 'https://external-content.duckduckgo.com/iu/?u=https%3A%2F%2Fi.pinimg.com%2Forig  
image_reduction(url, 10)
```



#imagen 3

```
url = 'https://external-content.duckduckgo.com/iu/?u=https%3A%2F%2Fcdn.wallpapersafar:  
image_reduction(url, 30)
```



## ▼ Ejercicio 3

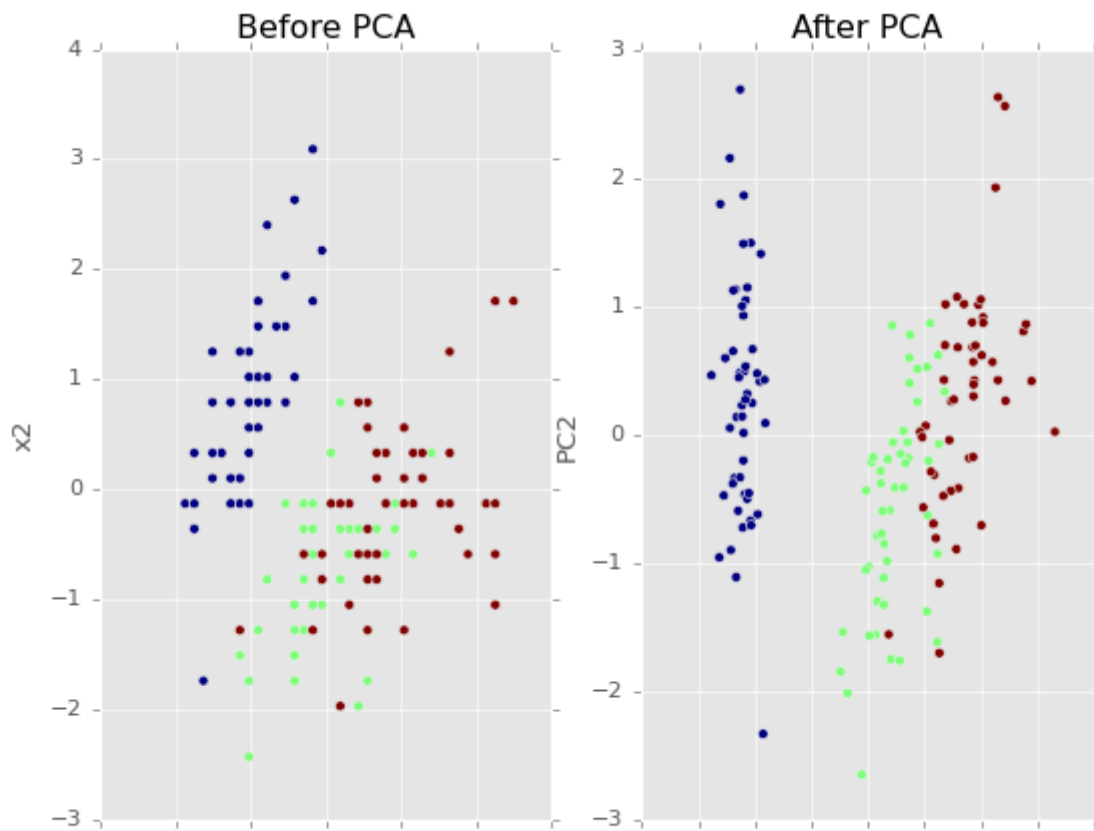
### Feature importances

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>

```
#tu codigo aqui
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

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(abs(pca.components_))
```

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

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

Considero que lo relevante del ejercicio es que muchas de las propiedades del dataset se pueden conservar en un espacio de dos dimensiones (componente) esto quiere decir que el comportamiento de las clases es muy marcado entre ellas, unas de las ventajas de pca es que no descarta del todo una variable en cada componente sino que mediante la combinación de ellas pueda maximizar la varianza. En el caso de que el dataset fuera muy complejo de encontrar patrones entre los grupos se tendría que considerar mas componentes. pca puede ser una herramienta poderosa para visualizar clústeres en datos multidimensionales. Además, también lo es al crear modelos de aprendizaje automático, ya que también se puede usar como una variable explicativa.

### ¿Qué es feature importance y para que nos sirve?

El feature importance indica cuánto contribuye cada característica a la predicción del modelo. Básicamente, determina el grado de utilidad de una variable específica para un modelo y predicción actual.

Hay muchos beneficios de tener un puntaje sobre la importancia de las características. Es posible determinar la relación entre variables independientes (características) y variables dependientes (objetivos). Al analizar las puntuaciones de importancia de las variables, podríamos encontrar características irrelevantes y excluirlas. Reducir el número de variables no significativas en el modelo puede acelerar el modelo o incluso mejorar su rendimiento.

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

Lo mas relevante fue como al utilizar las primeras dos características de los datos se puede ver que algunos grupos se comportan igual, sin embargo, las dos variables no pueden explicar al 100% el comportamiento de los datos. Con pca se logró determinar que con las primeras dos componentes podemos maximizar la varianza y lograr una mejor clasificación en caso que se quiera utilizar estas nuevas variables en algun modelo pasando de 4 a 2 dimensiones.

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

En proyectos donde se tenga que realizar un análisis exploratorio de datos para ver si existe agrupaciones entre las componentes de manera visual y también en proyectos donde se tenga que entrenar algun modelo con cientos o miles de variables, utilizar PCA nos ayudaría a simplificar el conjunto de datos conservando la mayoría de las propiedades de los datos y disminuyendo el tiempo de entrenamiento.