PREGUNTA 4

1.Comenzamos importando las librerías y cargando la data.

```
import numpy as np
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt
import sklearn.datasets as ds
import sklearn.model_selection as modelSel
import sklearn.linear_model as reglin
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.datasets import load_digits
import seaborn as sns

digits = load_digits()
print(digits.data.shape)
```

En el print se obtiene:(1797,64)

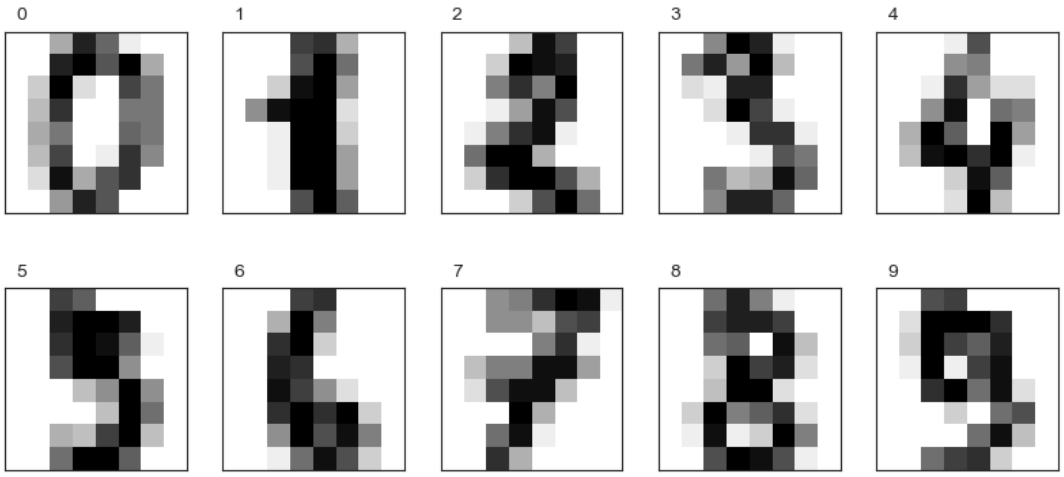
Lo cual indica que se tienen 1797 ocurrencias(imágenes) cada uno formada por un conjunto de 64 pixeles.

2. Definiendo la matriz de datos que tiene las instancias de 64 atributos cada una y el vector objetivo que tiene el número de dígito correspondiente.

```
X_digits, y_digits = digits.data, digits.target
```

3.Visualizamos las imágenes que tenemos en el dataset.

```
n_row, n_col = 2, 5


def print_digits(images, y, max_n=10):
    # set up the figure size in inches
    fig = plt.figure(figsize=(2. * n_col, 2.26 * n_row))
    i=0
    while i < max_n and i < images.shape[0]:
        p = fig.add_subplot(n_row, n_col, i + 1, xticks=[], yticks=[])
        p.imshow(images[i], cmap=plt.cm.binary, interpolation='nearest')
        # label the image with the target value
        p.text(0, -1, str(y[i]))
        i = i + 1


print_digits(digits.images, digits.target, max_n=10)
```



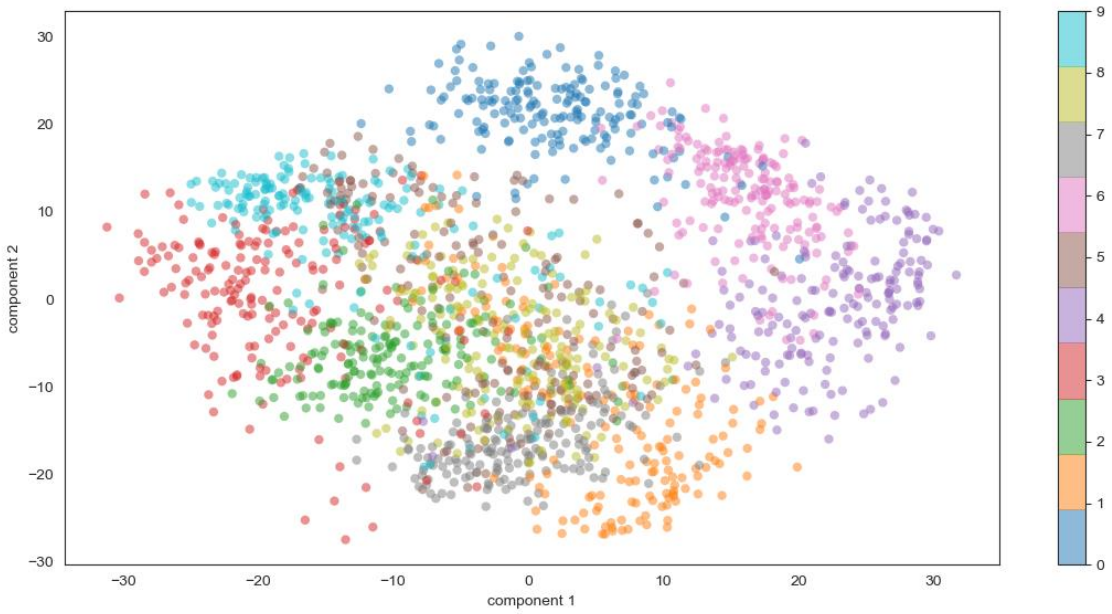
4.Como se sabe las imágenes están en un conjunto de 64 dimensiones (8x8 pixeles) por lo cual utilizaremos un PCA con 2 componentes para poder realizar un análisis exploratorio de la data.

```
pca = PCA(2)

projected = pca.fit_transform(X_digits)

print(projected.shape)
```

El objeto projected tiene un tamaño de : (1797,2)

4.Trazamos el conjunto de datos sobre el plano formado por los dos primeros componentes principales.

```
from matplotlib import cm

from colorspacious import cspace_converter

from collections import OrderedDict


cmap = OrderedDict()


#Trazamos los dos primeros componentes principales de cada punto

plt.scatter(projected[:, 0], projected[:, 1],

        c=digits.target, edgecolor='none', alpha=0.5,

        cmap =cm.get_cmap("tab10",10))

plt.xlabel('component 1')

plt.ylabel('component 2')

plt.colorbar();
```



- Podemos ver las 10 clases diferentes correspondientes a los 10 primeros dígitos, se nota que para cada clase sus ocurrencias se encuentran agrupadas en grupos de acuerdo a su target a la vez que los grupos son relativamente distintos.
- La excepción para esto es la clase 5 la cual tiene múltiples ocurrencias superpuestas con las demás ocurrencias de las demás clases. Mientras que el grupo 0 es el más diferenciado con respecto a los demás.

5. Definimos a cada imagen como una colección de 64 pixeles de la siguiente forma:

$$x = [x_1, x_2, x_3 \cdots x_{64}]$$

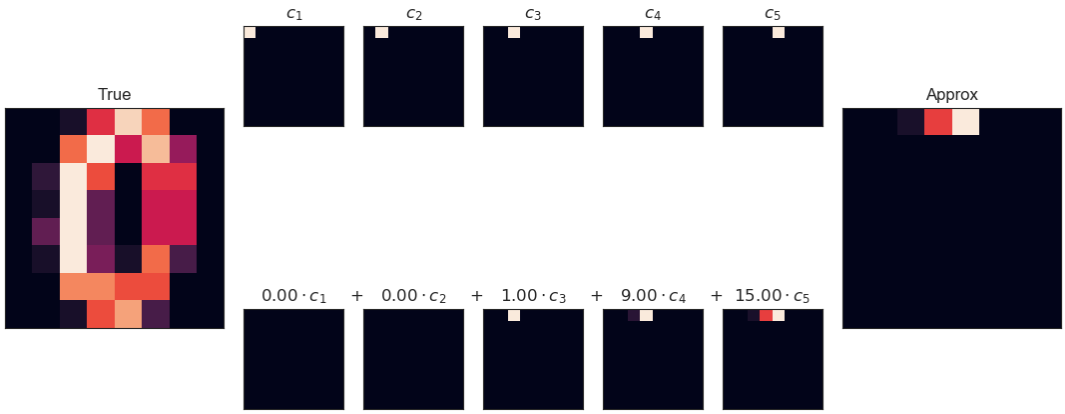
Para construir cada imagen, construiremos una base cuyos elementos serán los pixeles de esta y se manejará como coeficientes los pixeles que describen cada elemento de la base, para así mediante una combinación lineal obtener las diferentes imágenes.

$$\mathrm{image}(x) = x_1 \cdot (\mathrm{pixel1}) + x_2 \cdot (\mathrm{pixel2}) + x_3 \cdot (\mathrm{pixel3}) \cdots x_{64} \cdot (\mathrm{pixel64})$$

6.Una forma de reducir la dimensionalidad es escoger algunos elementos del vector de pixeles y llevarlos a cero. Vamos a usar los 5 primeros pixeles de nuestro arreglo.

```
def plot_pca_components(x, coefficients=None, mean=0, components=None,
                        imshape=(8, 8), n_components=5, fontsize=12,
                        show_mean=True):
    if coefficients is None:
        coefficients = x

    if components is None:
        components = np.eye(len(coefficients), len(x))
    mean = np.zeros_like(x) + mean
    fig = plt.figure(figsize=(1.2 * (5 + n_components), 1.2 * 2))
    g = plt.GridSpec(2, 4 + bool(show_mean) + n_components, hspace=0.3)









def show(i, j, x, title=None):
        ax = fig.add_subplot(g[i, j], xticks=[], yticks=[])
        ax.imshow(x.reshape(imshape), interpolation='nearest')
        if title:
            ax.set_title(title, fontsize=fontsize)

    show(slice(2), slice(2), x, "True")

    approx = mean.copy()

    counter = 2
    if show_mean:
        show(0, 2, np.zeros_like(x) + mean, r'$\mu$')
```

```
        show(1, 2, approx, r'$1 \cdot \mu$')
        counter += 1

    for i in range(n_components):
        approx = approx + coefficients[i] * components[i]
        show(0, i + counter, components[i], r'$c_{0}$'.format(i + 1))
        show(1, i + counter, approx,
             r"${0:.2f} \cdot c_{1}$".format(coefficients[i], i + 1))
        if show_mean or i > 0:
            plt.gca().text(0, 1.05, '$+$', ha='right', va='bottom',
                           transform=plt.gca().transAxes, fontsize=fontsize)

    show(slice(2), slice(-2, None), approx, "Approx")
    return fig


sns.set_style('white')

fig = plot_pca_components(digits.data[10],
                          show_mean=False)
```



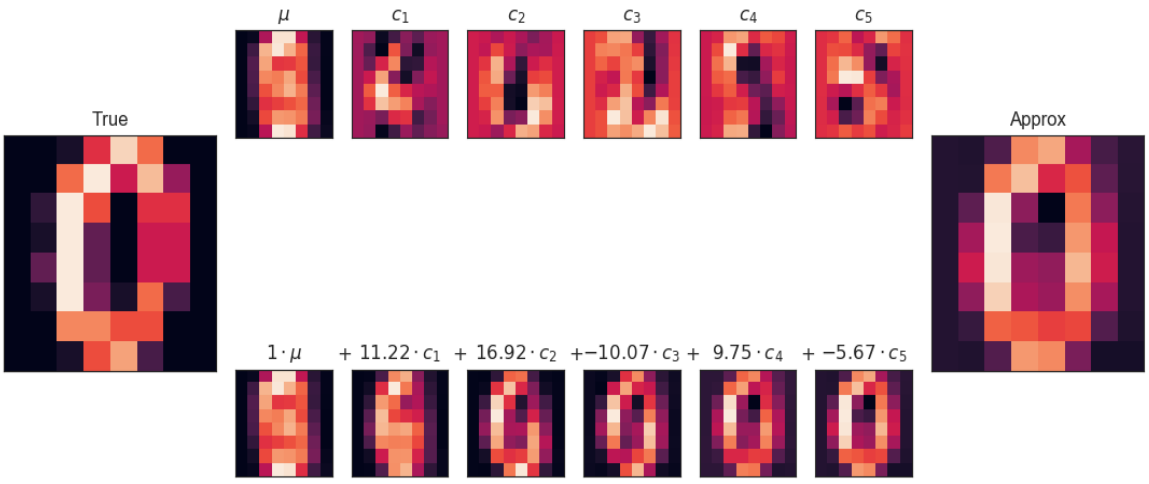
- En la fila superior se muestran los componentes del vector de pixeles que estamos usando , mientras que en la fila de abajo se muestra las combinación lineal de cada componente , la cual da como resultado una pequeña porción de la imagen original.

7.Utilizaremos PCA para reducir la dimensionalidad, tomando como base al conjunto de de componentes principales y como coeficientes al vector original de pixeles.

$$i\,estoy\,un\,ge(x) = \mathrm{mean} + x_1 \cdot (\mathrm{basis}1) + x_2 \cdot (\mathrm{basis}2) + x_3 \cdot (\mathrm{basis}3) \\ \cdots$$

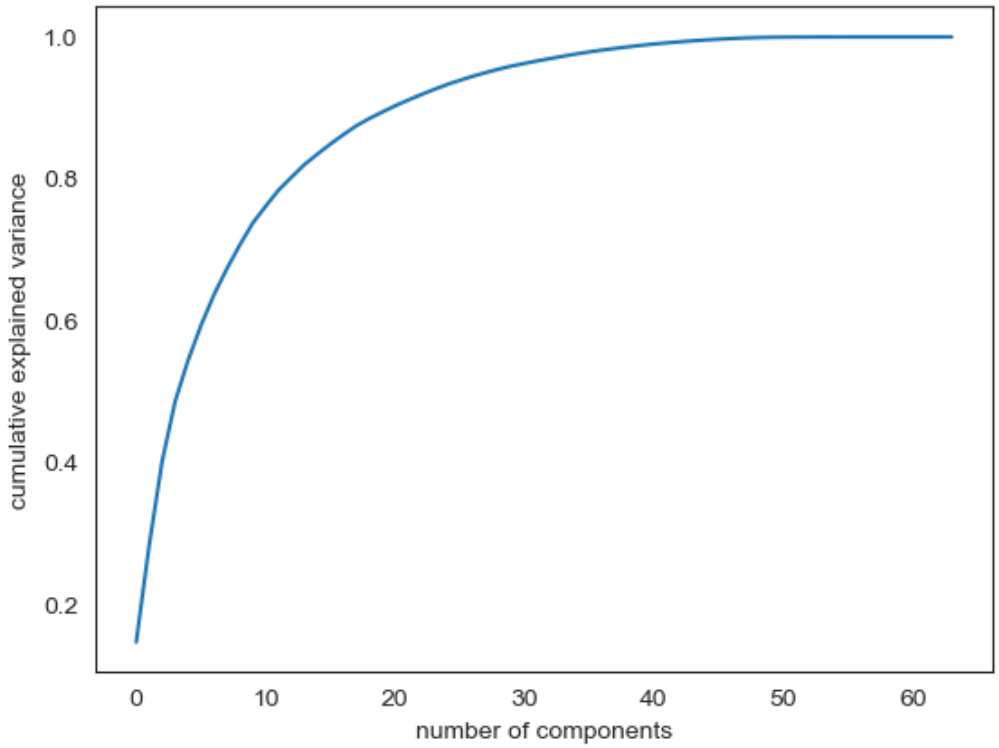
Usando a la media y a las 5 primeras componentes principales de PCA, se tiene:

```
pca = PCA(n_components=5)

Xproj = pca.fit_transform(digits.data)

sns.set_style('white')

fig = plot_pca_components(digits.data[10], Xproj[10],

                          pca.mean_, pca.components_)
```



Como se ve al usar como base a la media y a los 5 primeros componentes principales se obtiene mayor información (pixeles de la imagen original) que usando como base a los 5 primeros pixeles.

8.Debemos determinar cuantos componentes son necesarios para maximizar la varianza de la información, de tal modo que se pierda muy poco de las características originales de nuestros datos.



Como se ve aproximadamente con los 10 primeros componentes se tiene el 75% de la varianza explicada, mientras que con 50 componentes se tiene un valor cercano al 100% de la varianza explicada.

Para la proyección bidimensional que se trabajo al principio(n_components=2) se nota que se pierde demasiada información, y que para los primeros 20 componentes tenemos un total del 90% de la varianza explicada.