```
# Santiago López
# A01643411
```

Santiago López

A01643411

1. Load Data set y otras cosas que necesites

> Agregar bloque entrecomillado

```
# prompt: Load the Digits data set from sklearn
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns



from sklearn.datasets import load_digits

digits = load_digits()
```

2. Entender los dataset

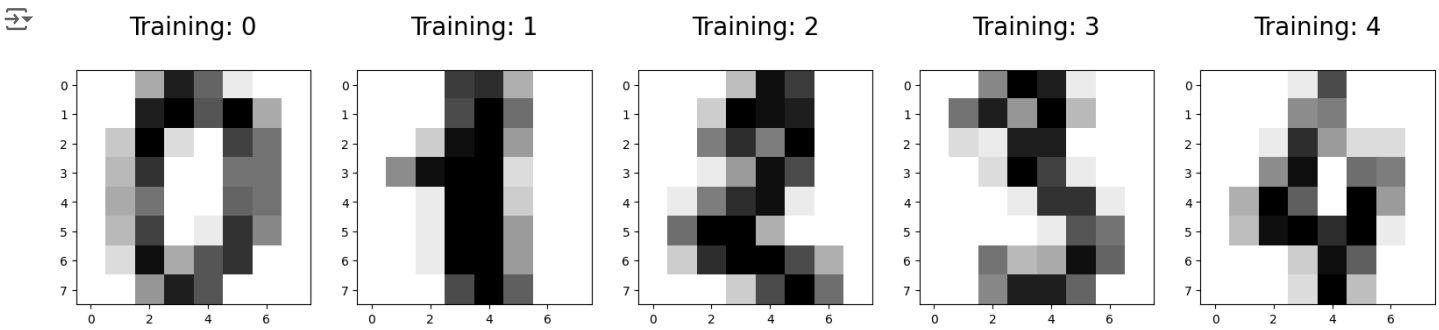
```
# prompt: get the head of the data

df = pd.DataFrame(data= np.c_[digits['data'], digits['target']],
                  columns= digits['feature_names'] + ['target'])
print(df.head())
```

```
   pixel_0_0  pixel_0_1  pixel_0_2  pixel_0_3  pixel_0_4  pixel_0_5  \
0        0.0        0.0        5.0       13.0        9.0        1.0
1        0.0        0.0        0.0       12.0       13.0        5.0
2        0.0        0.0        0.0        4.0       15.0       12.0
3        0.0        0.0        7.0       15.0       13.0        1.0
4        0.0        0.0        0.0        1.0       11.0        0.0

   pixel_0_6  pixel_0_7  pixel_1_0  pixel_1_1  ...  pixel_6_7  pixel_7_0  \
0        0.0        0.0        0.0        0.0  ...        0.0        0.0
1        0.0        0.0        0.0        0.0  ...        0.0        0.0
2        0.0        0.0        0.0        0.0  ...        0.0        0.0
3        0.0        0.0        0.0        8.0  ...        0.0        0.0
4        0.0        0.0        0.0        0.0  ...        0.0        0.0

   pixel_7_1  pixel_7_2  pixel_7_3  pixel_7_4  pixel_7_5  pixel_7_6  \
0        0.0        6.0       13.0       10.0        0.0        0.0
1        0.0        0.0       11.0       16.0       10.0        0.0
2        0.0        0.0        3.0       11.0       16.0        9.0
3        0.0        7.0       13.0       13.0        9.0        0.0
4        0.0        0.0        2.0       16.0        4.0        0.0

   pixel_7_7  target
0        0.0     0.0
1        0.0     1.0
2        0.0     2.0
3        0.0     3.0
4        0.0     4.0

[5 rows x 65 columns]
```

En este caso, los datos con los que estamos trabajando son pixeles.

3. Plotear algunos ejemplos

```
# prompt: Can you plot some of the examples(digits)

#Importamos lo necesario
import matplotlib.pyplot as plt

#Hacemos que el tamaño de la imagen sea de 20 por 4
plt.figure(figsize=(20,4))

#Itera sobre los primeros 5 imagnes y sus 5 labels
for index, (image, label) in enumerate(zip(digits.images[0:5], digits.target[0:5])):
  #Esto es para saber que es one row y 5 columns
  plt.subplot(1, 5, index + 1)
  # Aqui el image es la imagen, lo otro que tiene que ser en gris y lo demas que lo haga en bloques
  plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
  #Ya con esto hacemos print y sus labels
  plt.title('Training: %i\n' % label, fontsize = 20)
```



4. Aplicar estadistica descriptiva y extraer conclusiones

```
# prompt: Can you do some statistical description like the mean and percentiles, im not sure on what but im sure you do

feature_means = df.drop('target', axis=1).mean()
print("Mean of each feature:\n", feature_means)


percentiles = df.drop('target', axis=1).quantile([0.25, 0.5, 0.75])
print("\nPercentiles of each feature:\n", percentiles)


target_mean = df['target'].mean()
print("\nMean of the target variable:", target_mean)

feature_std = df.drop('target', axis=1).std()
print("\nStandard deviation of each feature:\n", feature_std)


df.describe()
```

```
Mean of each feature:
 pixel_0_0     0.000000
pixel_0_1     0.303840
pixel_0_2     5.204786
pixel_0_3    11.835838
pixel_0_4    11.848080
               ...
pixel_7_3    12.089037
pixel_7_4    11.809126
pixel_7_5     6.764051
pixel_7_6     2.067891
pixel_7_7     0.364496
Length: 64, dtype: float64

Percentiles of each feature:
      pixel_0_0  pixel_0_1  pixel_0_2  pixel_0_3  pixel_0_4  pixel_0_5  \
0.25        0.0        0.0        1.0       10.0       10.0        0.0
0.50        0.0        0.0        4.0       13.0       13.0        4.0
0.75        0.0        0.0        9.0       15.0       15.0       11.0

      pixel_0_6  pixel_0_7  pixel_1_0  pixel_1_1  ...  pixel_6_6  pixel_6_7  \
0.25        0.0        0.0        0.0        0.0  ...        0.0        0.0
0.50        0.0        0.0        0.0        0.0  ...        1.0        0.0
0.75        0.0        0.0        0.0        3.0  ...        7.0        0.0

      pixel_7_0  pixel_7_1  pixel_7_2  pixel_7_3  pixel_7_4  pixel_7_5  \
0.25        0.0        0.0        1.0       11.0       10.0        0.0
0.50        0.0        0.0        4.0       13.0       14.0        6.0
0.75        0.0        0.0       10.0       16.0       16.0       12.0

      pixel_7_6  pixel_7_7
0.25        0.0        0.0
0.50        0.0        0.0
0.75        2.0        0.0

[3 rows x 64 columns]

Mean of the target variable: 4.490818030050083

Standard deviation of each feature:
 pixel_0_0    0.000000
pixel_0_1    0.907192
pixel_0_2    4.754826
pixel_0_3    4.248842
pixel_0_4    4.287388
               ...
pixel_7_3    4.374694
pixel_7_4    4.933947
pixel_7_5    5.900623
pixel_7_6    4.090548
pixel_7_7    1.860122
Length: 64, dtype: float64
```

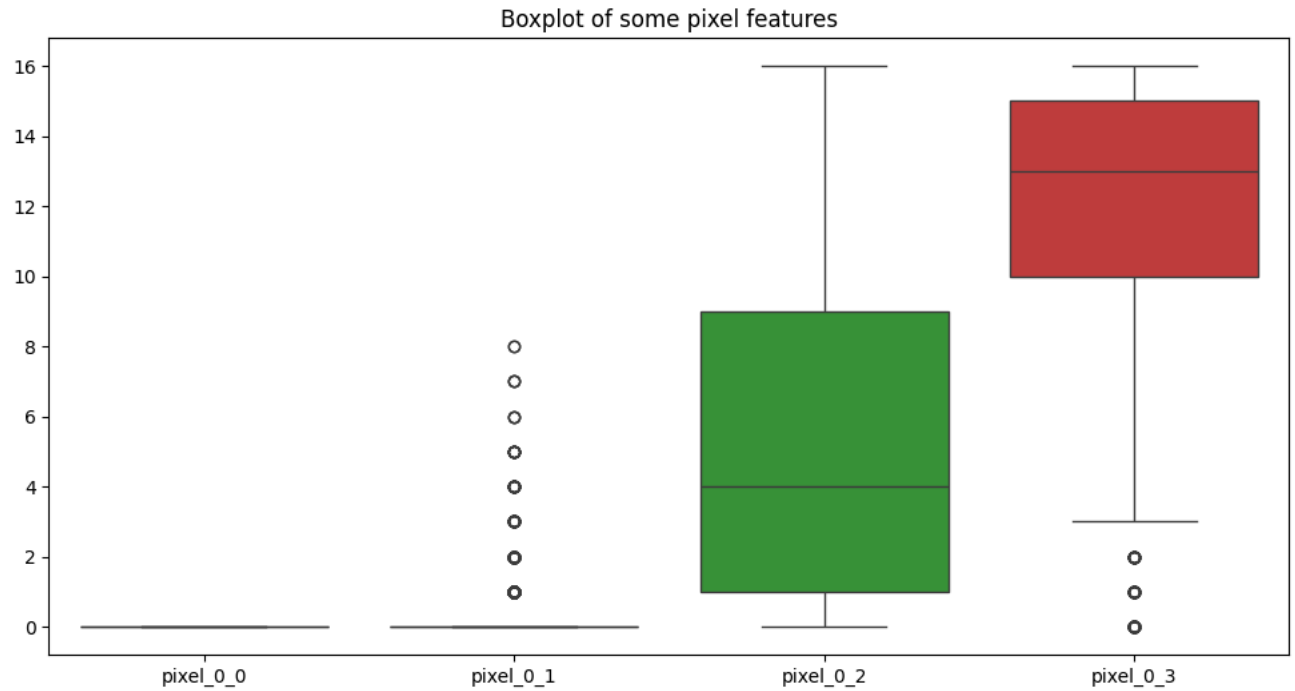
	pixel_0_0	pixel_0_1	pixel_0_2	pixel_0_3	pixel_0_4	pixel_0_5	pixel_0_6	pixel_0_7	pixel_1_0	pixel_1_1	...
count	1797.0	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	1797.000000	...
mean	0.0	0.303840	5.204786	11.835838	11.848080	5.781859	1.362270	0.129661	0.005565	1.993879	...
std	0.0	0.907192	4.754826	4.248842	4.287388	5.666418	3.325775	1.037383	0.094222	3.196160	...
min	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
25%	0.0	0.000000	1.000000	10.000000	10.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
50%	0.0	0.000000	4.000000	13.000000	13.000000	4.000000	0.000000	0.000000	0.000000	0.000000	...
75%	0.0	0.000000	9.000000	15.000000	15.000000	11.000000	0.000000	0.000000	0.000000	3.000000	...
max	0.0	8.000000	16.000000	16.000000	16.000000	16.000000	16.000000	15.000000	2.000000	16.000000	...

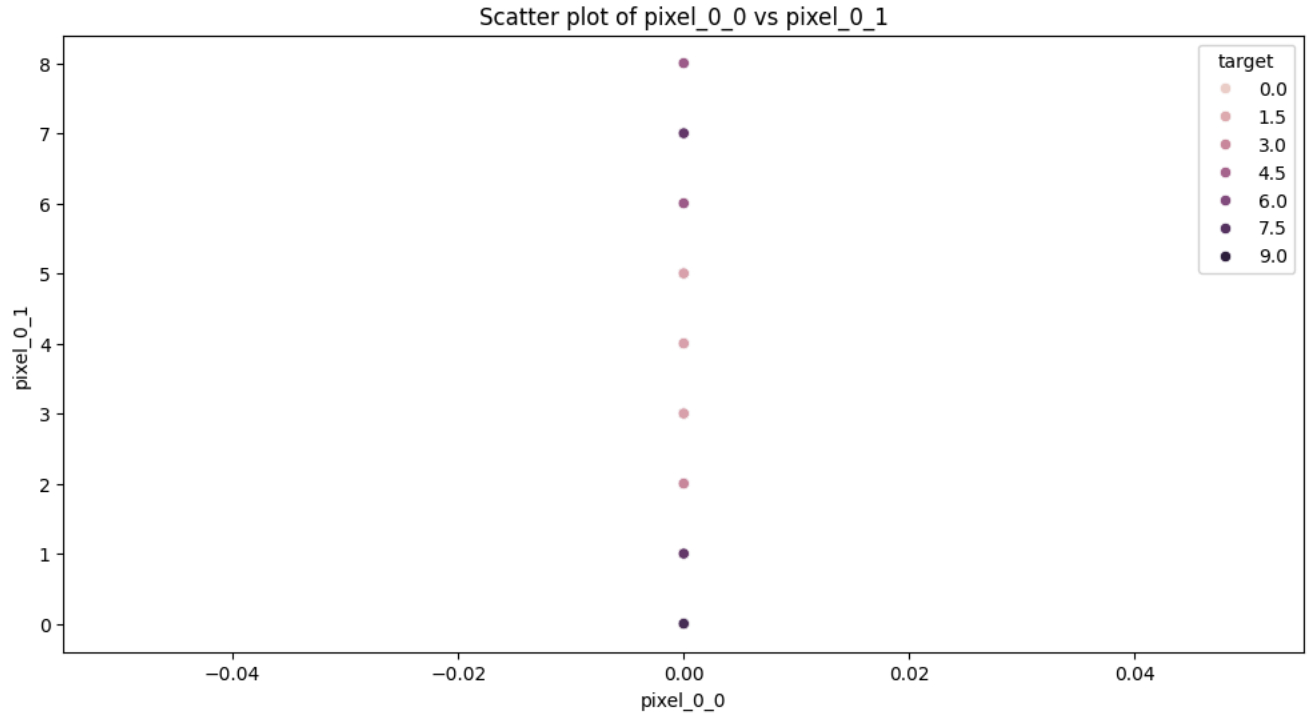
8 rows × 65 columns

Pues de las estadisticas que sacamos, podemos observar que hay algunos pixeles que se usan mas que otros. En este caso, contamos que pues entre todos los ejemplos, hay 1797 pixeles de cada pixel. Sacamos la desviacion estandar de algunos que nos dio desde 0 hasta valores como 5.9. Tambien, sacamos por ejemplo que para el pixel_0_2 en el percentil 50% es 4. Los promedios nos ayudan a ver el promedio de que tan oscuros estan en todas las imagenes. El mean del target value nos avisa pues de que tan numero hay tanto. Desviacion Standar para avisarnos que tanto cambia y que parte de la imagen suele cambiar mas.El median, el mode y la varianza. El median nos da pues el brigthness de en medio. Mode para saber cual es el mas popular. Igual que el estandar desviation.

5. Visualización

```
# prompt: Can you make a boxplot from some of the variables and scatter plots for some variables

# Boxplot for some features
plt.figure(figsize=(12, 6))
sns.boxplot(data=df[['pixel_0_0', 'pixel_0_1', 'pixel_0_2', 'pixel_0_3']])
plt.title('Boxplot of some pixel features')
plt.show()

# Scatter plot for some features
plt.figure(figsize=(12, 6))
sns.scatterplot(x='pixel_0_0', y='pixel_0_1', data=df, hue='target')
plt.title('Scatter plot of pixel_0_0 vs pixel_0_1')
plt.show()

# You can create more boxplots and scatter plots for different variables as needed.
# For example:
# sns.boxplot(data=df[['pixel_10_10', 'pixel_15_15']])
# sns.scatterplot(x='pixel_5_5', y='pixel_10_10', data=df, hue='target')

#El boxplot nos dice la distribucion de cada uno de los pixeles.
#Nos enseña la relacion entre dos puntos para ver que onda.
```



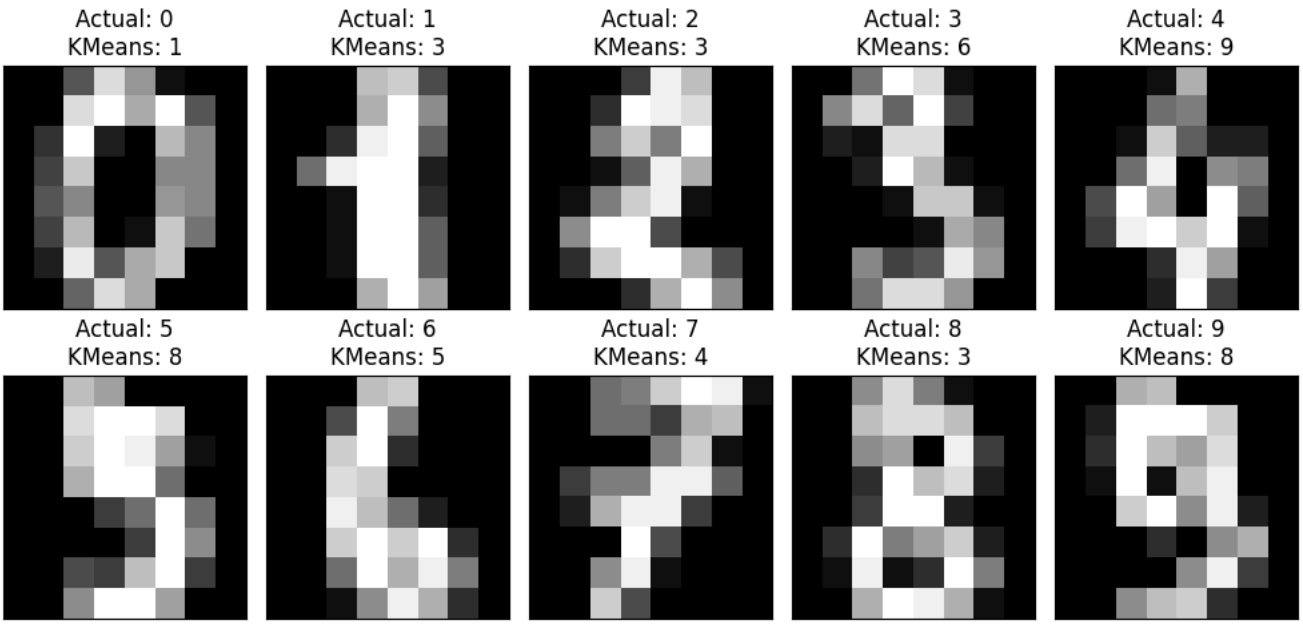


6. Do K-means with all variables

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_digits
import numpy as np

# Step 1: Load the digits dataset
digits = load_digits()
data = digits.data  # Flattened 8x8 pixel images
target = digits.target  # Actual digit labels
n_digits = len(digits.target_names)

# Step 2: Apply KMeans with the number of clusters equal to the number of digits (0-9)
kmeans = KMeans(n_clusters=n_digits, random_state=42)
kmeans.fit(data)
```

```
labels = kmeans.labels_  # KMeans predicted cluster labels

# Step 3: Filter for one image per digit (0-9)
unique_digits = []
for digit in range(10):
    digit_index = np.where(target == digit)[0][0]  # Get the index of the first occurrence of each digit
    unique_digits.append(digit_index)

# Step 4: Plot one image per digit, showing what KMeans thinks each digit is
fig, ax = plt.subplots(2, 5, figsize=(10, 5))  # 2 rows, 5 columns for displaying 10 images

for i, index in enumerate(unique_digits):
    # Get the actual digit and the predicted cluster (what KMeans thinks it is)
    actual_digit = target[index]  # The actual label (digit 0-9)
    kmeans_prediction = labels[index]  # KMeans cluster label

    # Plot the image
    ax[i // 5, i % 5].imshow(digits.images[index], cmap='gray')  # Show the image (reshaped 8x8)

    # Set the title to display both the actual digit and KMeans' prediction
    ax[i // 5, i % 5].set_title(f'Actual: {actual_digit}\nKMeans: {kmeans_prediction}')

    # Remove axis ticks for a cleaner look
    ax[i // 5, i % 5].set_xticks([])
    ax[i // 5, i % 5].set_yticks([])

# Adjust layout for better display
plt.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will c
  super()._check_params_vs_input(X, default_n_init=10)
```



Podemos observar la diferencia entre los diferentes centroides y los puntos o datos que conforman ese cluster.

7. Do K-means con solo variables de una de las Filas de c/Imagen, Con que filas se hace bien o mal

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_digits
import numpy as np
from collections import defaultdict
```

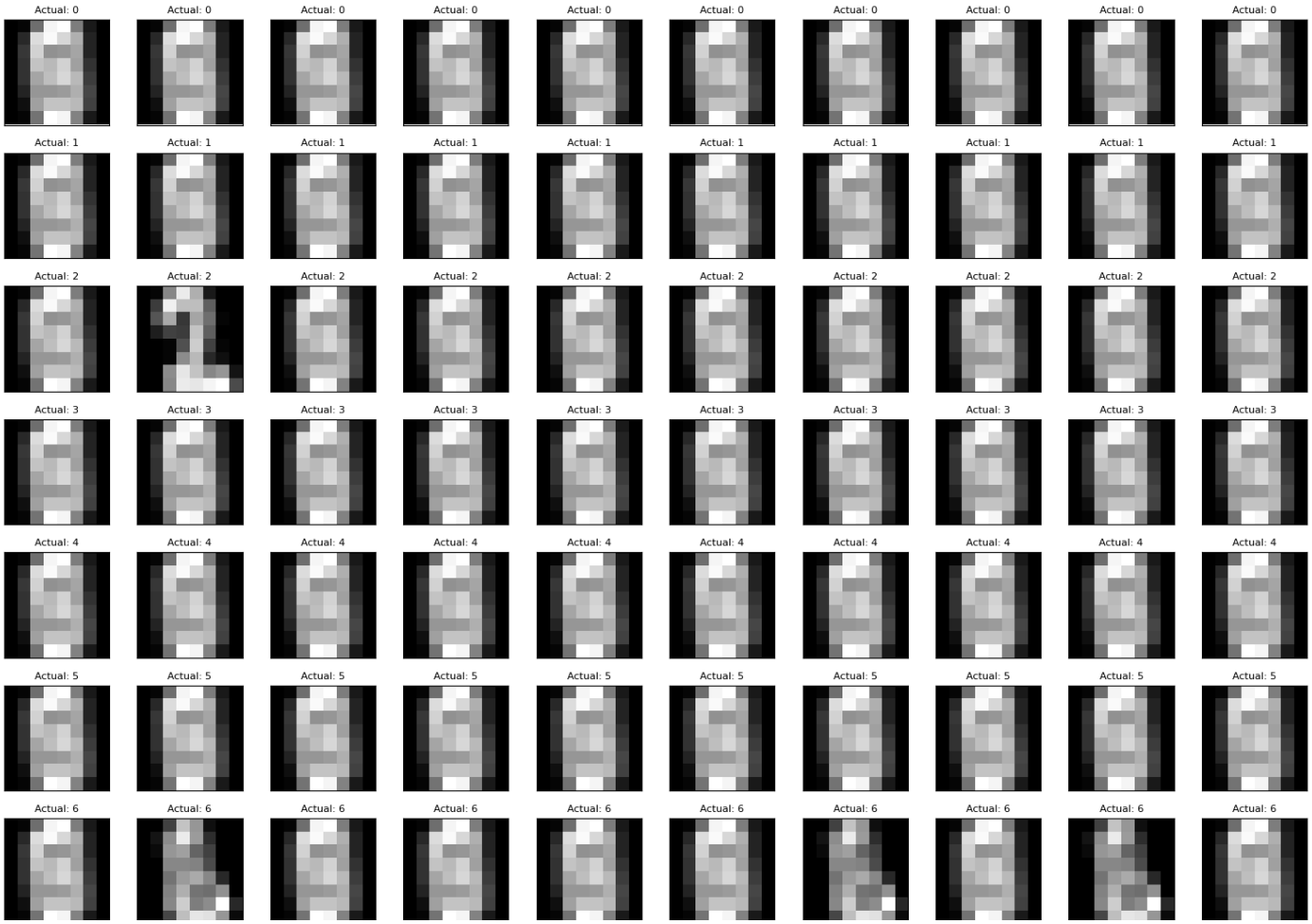
```
# Step 1: Load the digits dataset
digits = load_digits()
data = digits.images  # Shape: (n_samples, 8, 8)
target = digits.target  # Actual digit labels
n_digits = len(digits.target_names)

# Step 2: Specify the row to cluster on (e.g., row 1)
specified_row = 7  # This is the row you want to analyze (0-7)

# Step 3: Extract the specified row for each image
rows_data = data[:, specified_row, :]  # Extract only the specified row (shape: (n_samples, 8))

# Step 4: Apply KMeans on the rows, using 10 clusters (for digits 0-9)
kmeans = KMeans(n_clusters=10, random_state=42)
kmeans.fit(rows_data)
row_labels = kmeans.labels_  # KMeans predicted cluster labels for each row

# Step 5: Compute the average image for each cluster
cluster_images = defaultdict(list)
for cluster in range(10):
    # Find all the indices that belong to this cluster
    cluster_indices = np.where(row_labels == cluster)[0]

    # Compute the average image for this cluster
    if len(cluster_indices) > 0:
        avg_image = np.mean(data[cluster_indices], axis=0)  # Average across all images in the cluster
        cluster_images[cluster] = avg_image

# Step 6: Plot 10 examples for each digit (0-9), reconstructing the image from the cluster's average image
fig, ax = plt.subplots(10, 10, figsize=(15, 15))  # 10 rows for digits, 10 columns for each example

# Loop through each digit (0-9)
for digit in range(10):
    # Get the first 10 indices of the current digit
    digit_indices = np.where(target == digit)[0][:10]  # Get 10 examples of the digit

    # Loop through each of the 10 examples
    for example_num, digit_index in enumerate(digit_indices):
        # Get the KMeans cluster label for this row
        kmeans_prediction = row_labels[digit_index]  # KMeans cluster prediction for this row

        # Get the average image of the cluster it belongs to
        reconstructed_image = cluster_images[kmeans_prediction]

        # Plot the reconstructed image based on the cluster
        ax[digit, example_num].imshow(reconstructed_image, cmap='gray')

        # Set the title to display the actual digit
        ax[digit, example_num].set_title(f'Actual: {digit}', fontsize=8)

        # Remove axis ticks for a cleaner look
        ax[digit, example_num].set_xticks([])
        ax[digit, example_num].set_yticks([])

# Adjust layout for better display
plt.tight_layout()
plt.show()
```

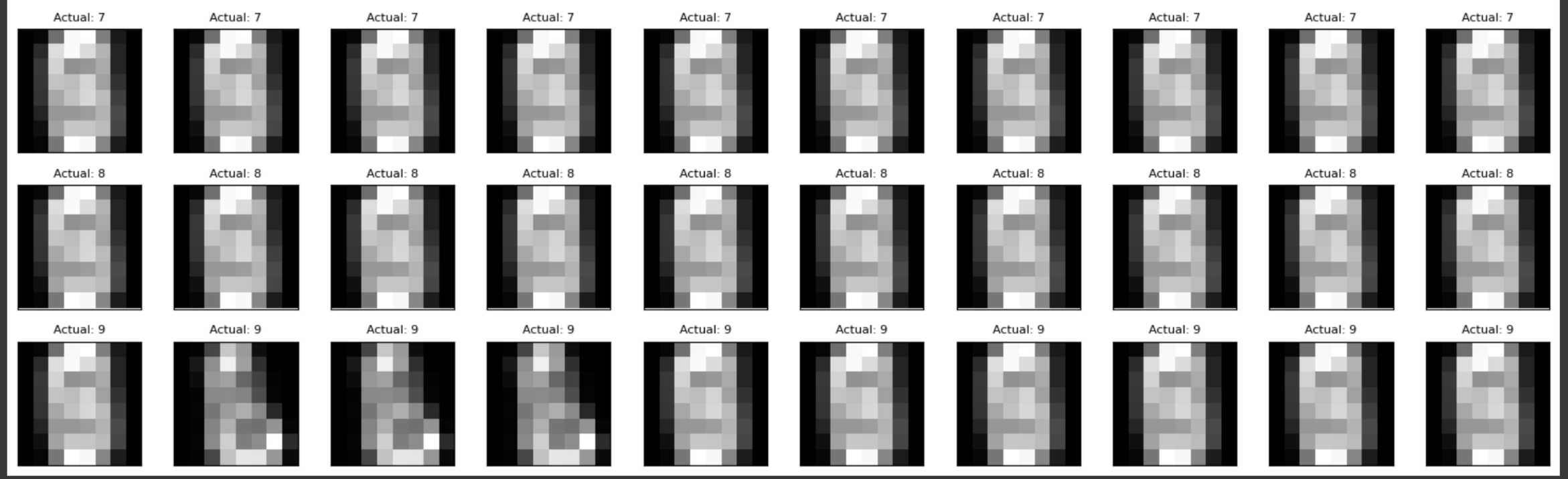
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will c
  super()._check_params_vs_input(X, default_n_init=10)
```

8. Do K-means con solo variables de una de las columnas de la imagen. Con que columnas se hace bien o mal

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_digits
import numpy as np
from collections import defaultdict

# Step 1: Load the digits dataset
digits = load_digits()
data = digits.images  # Shape: (n_samples, 8, 8)
target = digits.target  # Actual digit labels
n_digits = len(digits.target_names)

# Step 2: Specify the column to cluster on (e.g., column 1)
specified_column = 7  # This is the column you want to analyze (0-7)

# Step 3: Extract the specified column for each image
columns_data = data[:, :, specified_column]  # Extract only the specified column (shape: (n_samples, 8))

# Step 4: Apply KMeans on the columns, using 10 clusters (for digits 0-9)
kmeans = KMeans(n_clusters=10, random_state=42)
kmeans.fit(columns_data)
column_labels = kmeans.labels_  # KMeans predicted cluster labels for each column

# Step 5: Compute the average image for each cluster
cluster_images = defaultdict(list)
for cluster in range(10):
    # Find all the indices that belong to this cluster
    cluster_indices = np.where(column_labels == cluster)[0]

    # Compute the average image for this cluster
    if len(cluster_indices) > 0:
        avg_image = np.mean(data[cluster_indices], axis=0)  # Average across all images in the cluster
        cluster_images[cluster] = avg_image

# Step 6: Plot 10 examples for each digit (0-9), reconstructing the image from the cluster's average image
fig, ax = plt.subplots(10, 10, figsize=(15, 15))  # 10 rows for digits, 10 columns for each example

# Loop through each digit (0-9)
for digit in range(10):
    # Get the first 10 indices of the current digit
    digit_indices = np.where(target == digit)[0][:10]  # Get 10 examples of the digit

    # Loop through each of the 10 examples
    for example_num, digit_index in enumerate(digit_indices):
        # Get the KMeans cluster label for this column
        kmeans_prediction = column_labels[digit_index]  # KMeans cluster prediction for this column

        # Get the average image of the cluster it belongs to
        reconstructed_image = cluster_images[kmeans_prediction]

        # Plot the reconstructed image based on the cluster
        ax[digit, example_num].imshow(reconstructed_image, cmap='gray')

        # Set the title to display the actual digit
        ax[digit, example_num].set_title(f'Actual: {digit}', fontsize=8)

        # Remove axis ticks for a cleaner look
        ax[digit, example_num].set_xticks([])
        ax[digit, example_num].set_yticks([])

# Adjust layout for better display
plt.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will c
  super()._check_params_vs_input(X, default_n_init=10)
```





9. Conclusiones

Al analizar los diferentes outputs de los k mean por hileras vs columnas, puedo decir que he sacado datos interesantes. En general, los mejores numeros que si identifico bien fueron los del 0, 4 y 9. El mejor numero por si solo fue el del 0. El 8 fue un numero que no reconocia mucho. Yo creo que la razón de esto es que si se ve de cierta manera, se puede parecer a un 9 o 6. En algunos casos, no me daba ni un numero parecido, al analizar esto, pues seguramente es asi ya que no hay datos en esa hilera o columna por la cual no puede hacer un cluster ahi. En total, si hubiera hecho las 10 fotos por numero por hileras y columnas, hubiera hecho un total de 1,600 fotos. Para evitar esto, corri el programa varias veces y nomas le cambie el numero de hilera o columna de la cual tiene que hacer.

10. Referencias

OpenAI. (2024). ChatGPT (September 12, 2024 version) [Large language model]. https://openai.com/chatgpt