TC1002S Herramientas computacionales: el arte de la analítica

This is a notebook with all your work for the final evidence of this course

Niveles de dominio a demostrar con la evidencia

SING0202A

Interpreta interacciones entre variables relevantes en un problema, como base para la construcción de modelos bivariados basados en datos de un fenómeno investigado que le permita reproducir la respuesta del mismo. Es capaz de construir modelos bivariados que expliquen el comportamiento de un fenómeno.

Student information

- Name: Angel Eduardo Esquivel Vega
- ID: A01276114
- My carreer: ITC (Ingenieria en Tecnologías Computacionales)

Importing libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import drive
```

PART 1

Do clustering using your assigned dataset

a) Load data

```
drive.mount('/content/drive')
route = "/content/drive/My Drive/Data/A01276114_X.csv"
data = pd.read_csv(route)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mour
```

b) Data managment

Print the first 7 rows

```
data.head(7)
```

	Unnamed: 0	x1	x2	x3	x4	x5	x6	x7	
0	0	1.259952	0.469987	6.782171	6.289488	4.517730	-4.049429	-5.827139	2.
1	1	-7.288317	-2.758768	-7.621019	13.157248	11.145198	-4.507143	5.603168	-8.
2	2	2.299289	2.142595	5.201455	10.863968	4.599187	-4.893271	-7.192110	1.
3	3	-5.656540	-10.590100	3.423023	-0.323488	-5.020291	3.753454	4.335032	3.
4	4	2.765513	1.487666	7.243867	8.315266	5.344684	-6.519735	-3.879173	0.
5	5	-6.734926	1.611743	-1.444478	1.826075	-5.899354	0.714018	5.475416	-4.

Próximos pasos: Generar código con data | Ver gráficos recomendados | New interactive sheet

Print the last 4 rows

```
data.tail(4)
```

	Unnamed: 0	x1	x2	x3	x4	x5	x6	x7
839	839	8.130937	4.697175	7.886284	9.862232	-8.001242	-2.560356	-7.170618
840	840	2.530790	4.750932	7.912907	-10.463887	5.383948	-5.460728	5.666656
841	841	-7.166273	-6.081673	-9.466563	9.807595	9.250319	-4.479720	3.282340

How many rows and columns are in your data?

Use the `shape` method

```
print(f'There are {data.shape[0]} rows and {data.shape[1]} columns in the dataset')
```

```
There are 843 rows and 9 columns in the dataset
```

Print the name of all columns

Use the `columns` method

```
data.columns
```

```
Index(['Unnamed: 0', 'x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8'], dtype='object')
```

```
for i in range (0, data.shape[1]):
  print(f'{i+1}.- {data.columns[i]}')
```

```
1.- Unnamed: 0
2.- x1
3.- x2
4.- x3
5.- x4
6.- x5
7.- x6
8.- x7
9.- x8
```

What is the data type in each column

Use the `dtypes` method

```
data.dtypes
```

	0
Unnamed: 0	int64
x1	float64
x2	float64
x3	float64
x4	float64
x5	float64
x6	float64
x7	float64
x8	float64

dtype: object

What is the meaning of rows and columns?

Your responses here

1. id or counter
2. x1
3. x2
4. x3
5. x4
6. x5
7. x6
8. x7
9. x8

Print a statistical summary of your columns

```
data.describe()
```

	Unnamed: 0	x1	x2	x3	x4	x5	x6
count	843.000000	843.000000	843.000000	843.000000	843.000000	843.000000	843.000000
mean	421.000000	-1.605345	-0.160149	1.932459	3.410501	-0.090230	-1.959332
std	243.497433	6.251502	5.476995	5.832662	7.301196	7.331149	3.564240
min	0.000000	-13.906640	-13.912779	-14.838680	-15.006980	-13.610585	-11.302162
25%	210.500000	-7.405387	-4.967347	0.236241	-0.600704	-6.772878	-4.777551
50%	421.000000	-1.310489	1.824610	3.150535	5.220637	0.480621	-2.188991
75%	631.500000	3.889635	4.104535	5.955494	9.426208	6.454696	0.875965

1. What is the minumum and maximum values of each variable:
2. What is the mean and standar deviation of each variable:
3. What the 25%, 50% and 75% represent?:

1. Variable: Unnamed
    11. Minimum: 0
    12. Maximum: 842
    13. Mean: 421
    14. Standard deviation: 243.49
2. Variable: x1
    21. Minimum: -13.90
    22. Maximum: 11.14
    23. Mean: -1.60
    24. Standard deviation: 6.25
3. Variable: x2
    31. Minimum: -13.91
    32. Maximum: 9.59
    33. Mean: -0.16
    34. Standard deviation: 5.47
4. Variable: x3
    41. Minimum: -14.83
    42. Maximum: 13.30

    43. Mean: 1.93
    44. Standard deviation: 5.83

5. Variable: x4

    51. Minimum: -15.00
    52. Maximum: 16.04
    53. Mean: 3.41
    54. Standard deviation: 7.30

6. Variable: x5

    61. Minimum: -13.61
    62. Maximum: 14.92
    63. Mean: -0.9
    64. Standard deviation: 7.33

7. Variable: x6

    71. Minimum: -11.30
    72. Maximum: 7.05
    73. Mean: -1.95
    74. Standard deviation: 3.56

8. Variable: x7

    81. Minimum: -12.75
    82. Maximum: 12.56
    83. Mean: 0.58
    84. Standard deviation: 6.10

9. Variable: x8

    91. Minimum: -14.55
    92. Maximum: 8.94
    93. Mean: -2.46
    94. Standard deviation: 4.65

100. What 25%, 50%, 75% represent?

    101. They represent percentiles. These give insight into the spread and concentration of data.
    102. 25% is the determined value where a quarter of data values are smaller than this number.
    103. 50% means half of the data values are smaller than this number and the other half is bigger.
    104. 75% value indicates the point below which 75% of data falls.

Rename the columns using the same name with capital letters

```
data.columns = data.columns.str.upper()
data.columns
```

```
Index(['UNNAMED: 0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8'], dtype='object')
```

Rename the columns to their original names

```
data.columns = data.columns.str.lower()
data.columns
```

```
Index(['unnamed: 0', 'x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8'], dtype='object')
```

Use two different alternatives to get one of the columns

```
data.iloc[:,1] #Accesing by index
data.loc[:,'x1']  #Accesing by name
```

	x1
0	1.259952
1	-7.288317
2	2.299289
3	-5.656540
4	2.765513
...	...
838	-9.385715
839	8.130937
840	2.530790
841	-7.166273
842	-8.575859

843 rows × 1 columns

dtype: float64

Get a slice of your data set: second and thrid columns and rows from 62 to 72

```
tempDf = data.iloc[62:73,1:3]
tempDf
```

	x1	x2
62	-7.267612	-6.007407
63	-0.125624	4.852418
64	-9.843551	-9.726212
65	-5.603288	-3.815406
66	-6.094541	-2.729994
67	-8.085673	6.747062
68	-7.031435	-5.886675
69	-6.290652	3.126811
70	1.916145	-0.668688
71	1.051085	3.216116
72	4.390697	4.144576

Próximos pasos: Generar código con `tempDf` | Ver gráficos recomendados | New interactive sheet

For the second and thrid columns, calculate the number of null and not null values and verify that their sum equals the total number of rows

```
 #Iterating throught the second and third columns
for i in range (1,3):
  nulls = data.iloc[:,i].isnull().sum() #Calculating the sum of null values for i column
  notNulls = data.iloc[:,i].notnull().sum() #Calculating the sum of not null values for i cc
  if (nulls + notNulls) == data.shape[0]: #Verifying if the sum of null and not null values
    print(f"Verified for column {data.columns[i]}")
    print(f"Null values: {nulls} Not Null values: {notNulls}")
  else:
    print(f"Not verified for column {data.columns[i]}")
    print(f"Null values: {nulls} Not Null values: {notNulls}")
```

```
Verified for column x1
Null values: 0 Not Null values: 843
Verified for column x2
Null values: 0 Not Null values: 843
```

Discard the last column

```
data = data.iloc[:,:-1] #Updating the dataset without the last column
data.columns
```

```
Index(['unnamed: 0', 'x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7'], dtype='object')
```

```
totalNulls = data.isnull().sum().sum() #Calculating the total number of null values
totalNulls
```

```
0
```

Questions

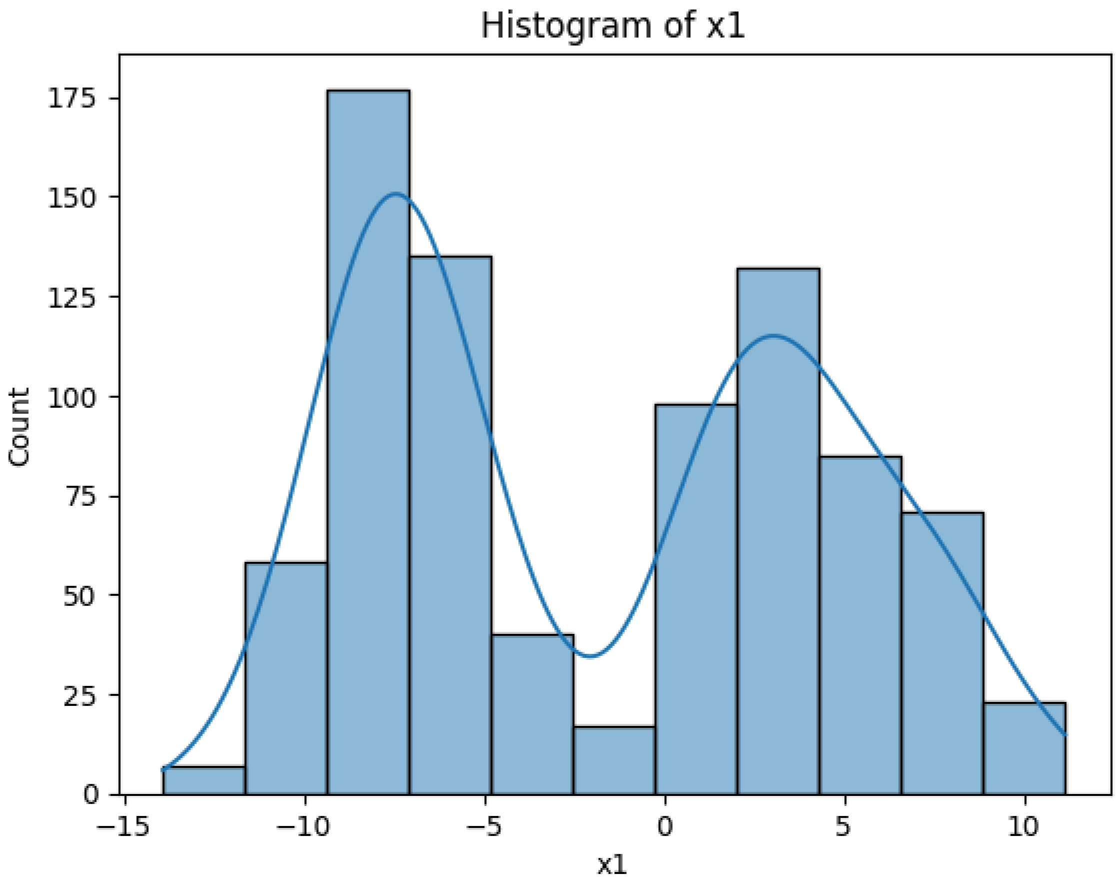
Based on the previos results, provide a full description of yout dataset

Your response: After modifying the dataset, we have 843 registries and 8 columns. First column is just an int id that starts on 0. Columns from 2-8 contain negative and positive floats. There are not null values in the dataset.

c) Data visualization

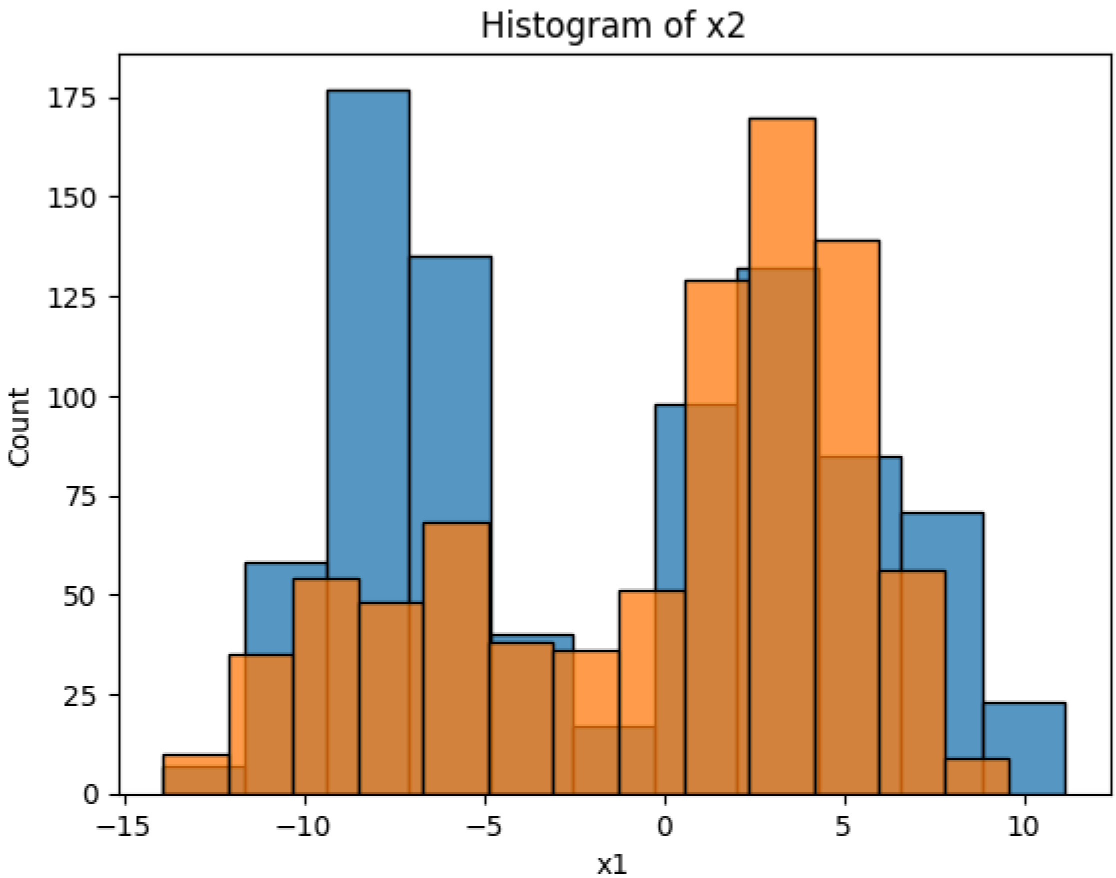
Plot in the histogram of one of the variables

```
#sns.histplot(data["x1"]).set_title("Histogram of x1")
sns.histplot(data["x1"], kde = True).set_title("Histogram of x1")
plt.show()
```

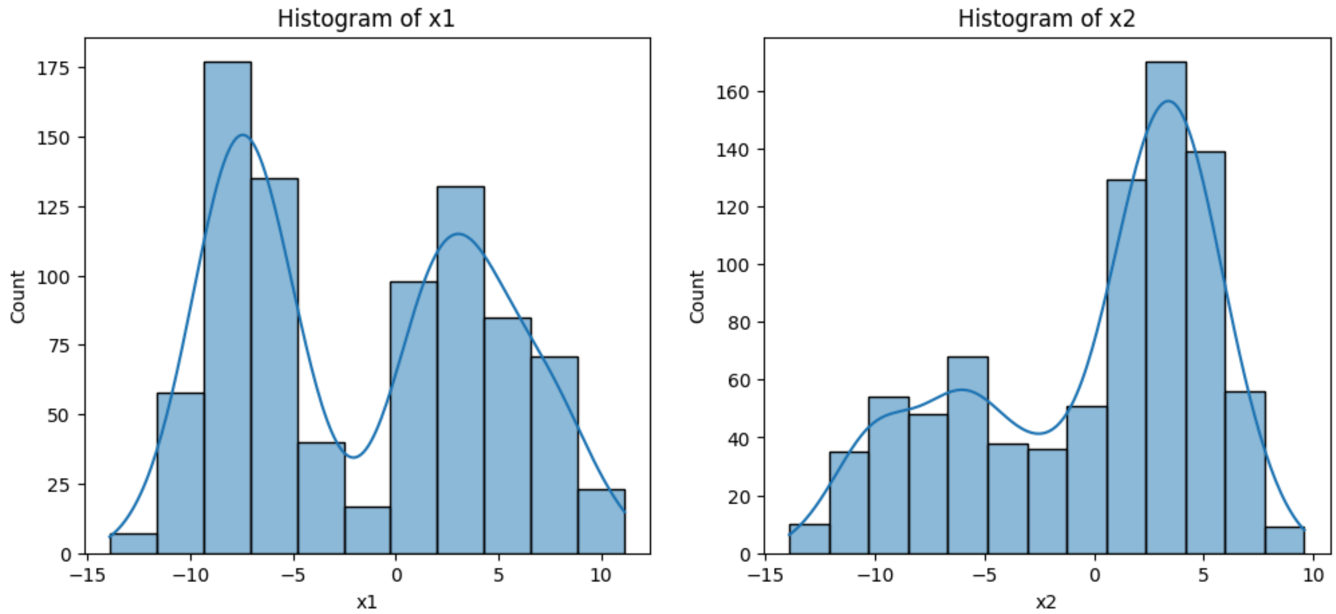


Plot in the same figure the histogram of two variables

```
#Option 1 : Overlay both histograms in the same plot
sns.histplot(data["x1"]).set_title("Histogram of x1")
sns.histplot(data["x2"]).set_title("Histogram of x2")
plt.show()
```




```
#Option 2 : Two side by side histograms in same figure but different plots.
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
sns.histplot(data["x1"], kde = True).set_title("Histogram of x1")
plt.subplot(1,2,2)
sns.histplot(data["x2"], kde = True).set_title("Histogram of x2")
plt.show()
```

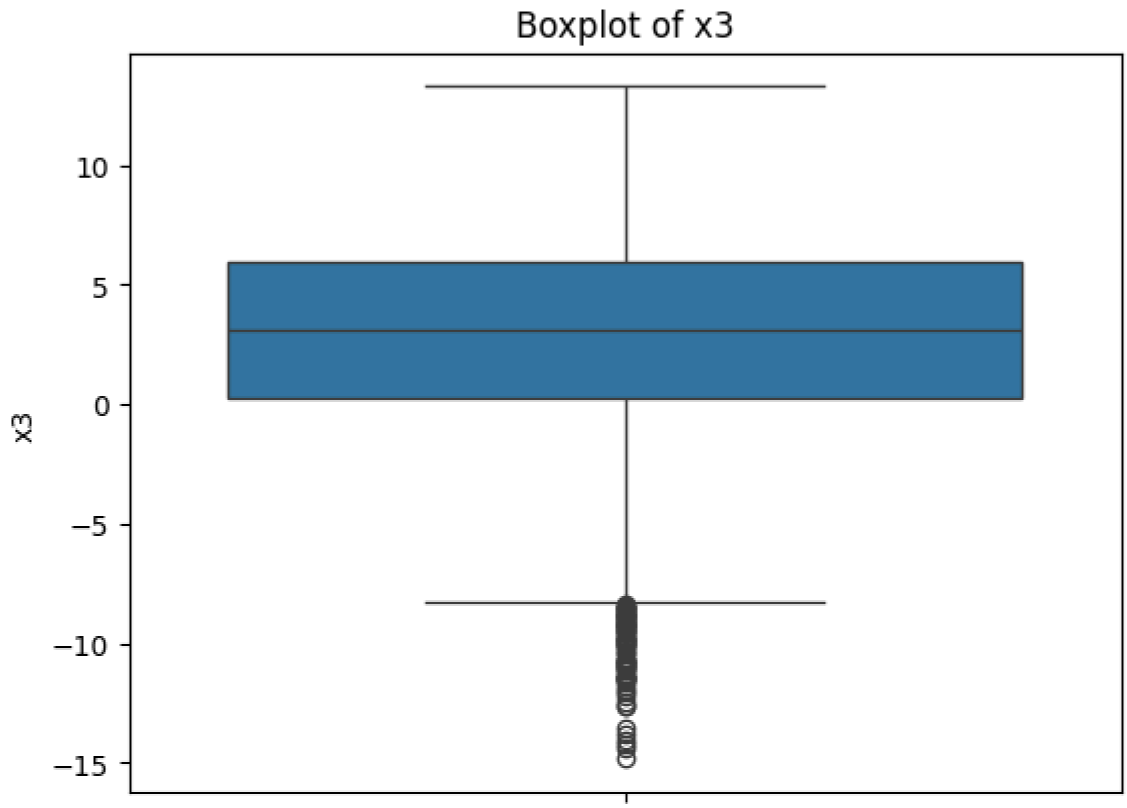


Based on these plots, provide a description of your data:

Your response here: Variables x1 and x2 seems to be not correlated since they dont follow similar KDEs. Data from both columns is in the range [-15, 10].

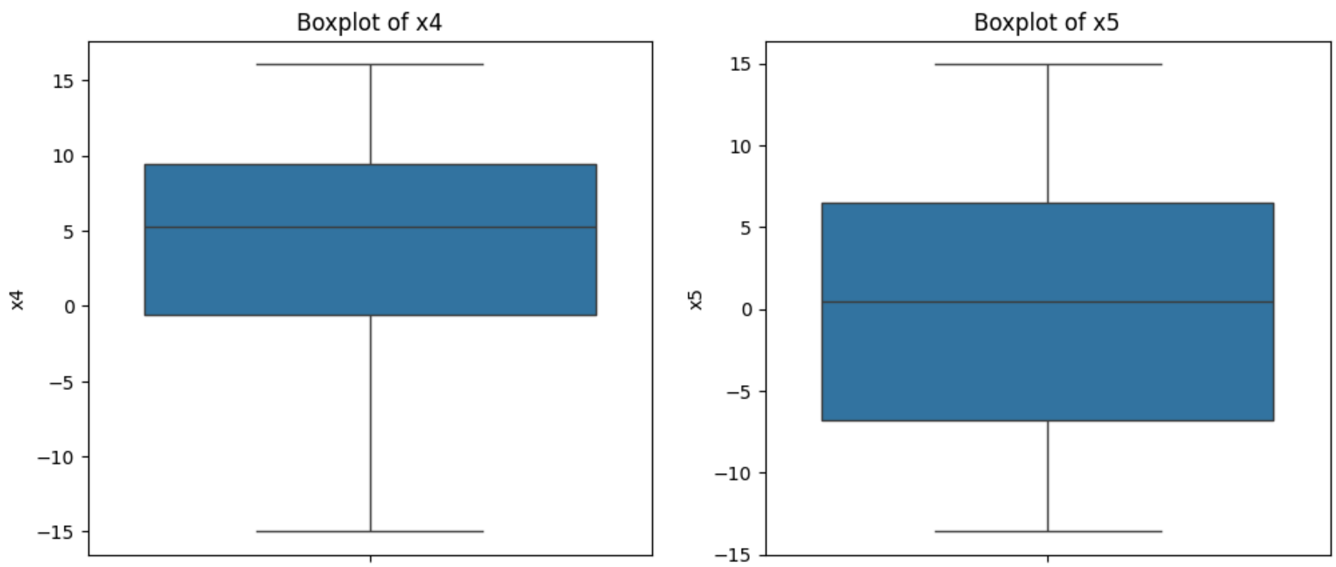
Plot the boxplot of one of the variables

```
sns.boxplot(data["x3"]).set_title("Boxplot of x3")
plt.show()
```



Plot in the same figure the boxplot of two variables

```
plt.figure(figsize = (12,5))
plt.subplot(1,2,1)
sns.boxplot(data["x4"]).set_title("Boxplot of x4")
plt.subplot(1,2,2)
sns.boxplot(data["x5"]).set_title("Boxplot of x5")
plt.show()
```

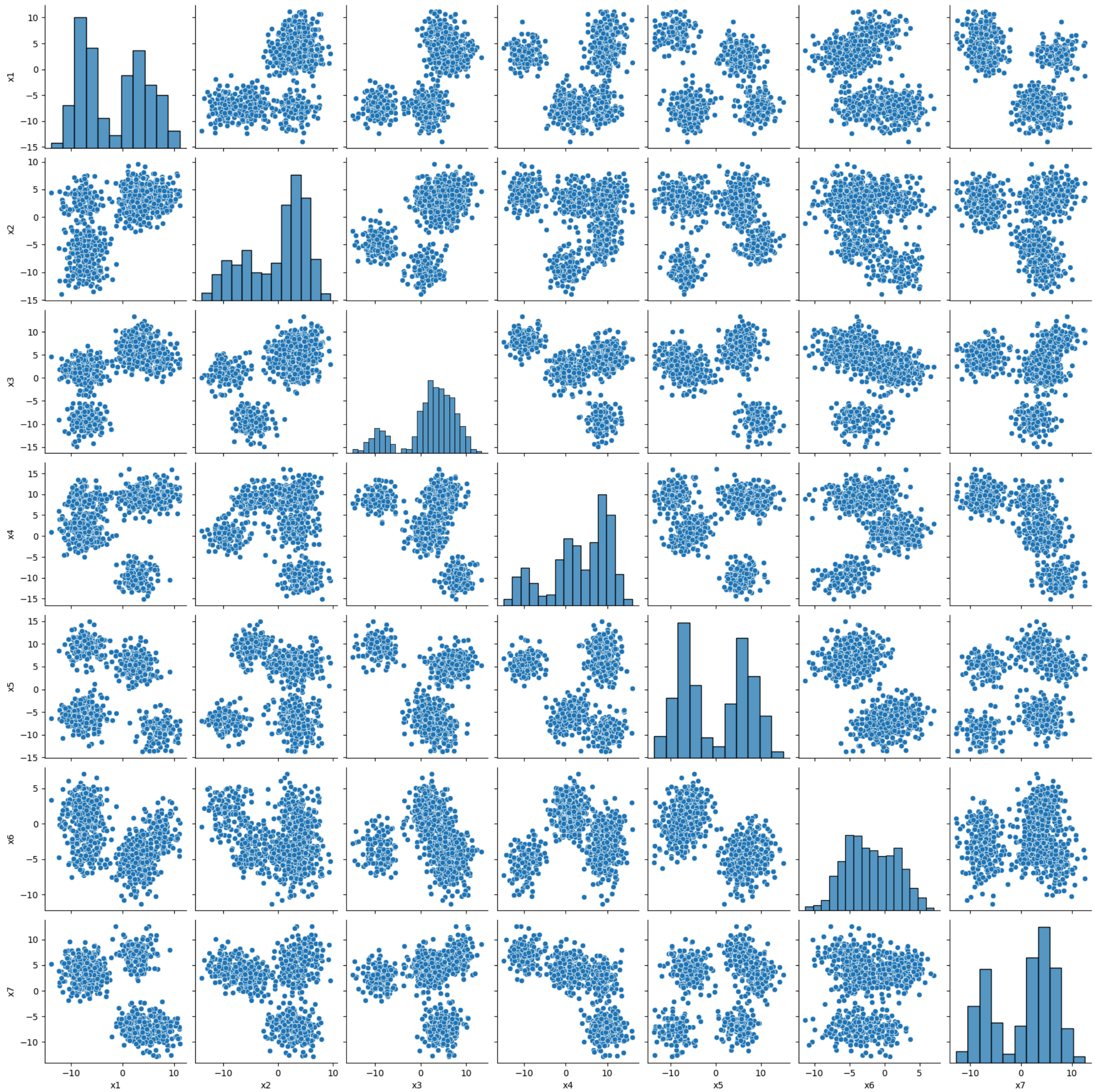


Based on these plots, provide a description of your data.

Your response here: x3 is more variable since it has several extreme low values. While, x4 and x5 are more satable and evenly spread around their medians.

Plot the scatter plot between all pair of variables

```
dataForPlot = data.iloc[:,1:]
sns.pairplot(dataForPlot)
plt.show()
```


x1
x2
x3
x4
x5
x6
x7

Questions

Based on the previos plots, provide a full description of yout dataset

Your response: Dataset appears to have multiple clusters within several variable pairs. This meaning there are distinct subgroups or patterns. Some show possible cluster associations and others seem to be uncorrelated. We can identify clusters by looking for groups of points that are closely packed together.

d) Kmeans

Do Kmeans clustering assuming a number of clusters according to your scatter plots

```
from sklearn.cluster import KMeans
nClusters = 4
kmeans = KMeans(n_clusters = nClusters, random_state = 0).fit(dataForPlot)
```

Add to your dataset a column with the estimated cluster to each data point

```
tempDF = pd.DataFrame(kmeans.labels_, columns = ["Cluster"])
dataForPlot = pd.concat([dataForPlot, tempDF], axis = 1)
dataForPlot
```

	x1	x2	x3	x4	x5	x6	x7	Cluster
0	1.259952	0.469987	6.782171	6.289488	4.517730	-4.049429	-5.827139	1
1	-7.288317	-2.758768	-7.621019	13.157248	11.145198	-4.507143	5.603168	2
2	2.299289	2.142595	5.201455	10.863968	4.599187	-4.893271	-7.192110	1
3	-5.656540	-10.590100	3.423023	-0.323488	-5.020291	3.753454	4.335032	0
4	2.765513	1.487666	7.243867	8.315266	5.344684	-6.519735	-3.879173	1
...	...	...	...	...	...	...	...	...
838	-9.385715	2.707788	-0.284174	3.218826	-5.888308	0.941448	1.355404	0
839	8.130937	4.697175	7.886284	9.862232	-8.001242	-2.560356	-7.170618	1
840	2.530790	4.750932	7.912907	-10.463887	5.383948	-5.460728	5.666656	3
841	-7.166273	-6.081673	-9.466563	9.807595	9.250319	-4.479720	3.282340	2
842	-8.575859	-11.971007	-0.527147	-2.053027	-5.990227	3.378329	6.574377	0

843 rows × 8 columns

Próximos pasos: Generar código con dataForPlot | Ver gráficos recomendados | New interactive sheet

Print the number associated to each cluster

```
print(np.unique(kmeans.labels_))

#Other way to solve this is:
#printedLabels = []
#for label in kmeans.labels_:
#  if(label in printedLabels):
#    continue
#  else:
#    printedLabels.append(label)
#    print(label)
```

```
[0 1 2 3]
```

Print the centroids

```
#Cluster centroides

print(kmeans.cluster_centers_)
```

```
[[-7.4859595  -3.07847822  1.53634008  0.93907053 -5.91394782  1.79948065
   4.26817995]
```

```
[ 4.78273558  2.80590521  4.79006952  9.69129469 -1.9172721  -3.17142186
 -7.31668349]
[-7.12188493 -4.80156964 -9.23083772  8.7931787   9.29389365 -3.56552912
  2.57179159]
[ 2.84440917  4.37660242  8.21168988 -9.71907569  5.78623643 -5.4177937
  7.12605264]]
```

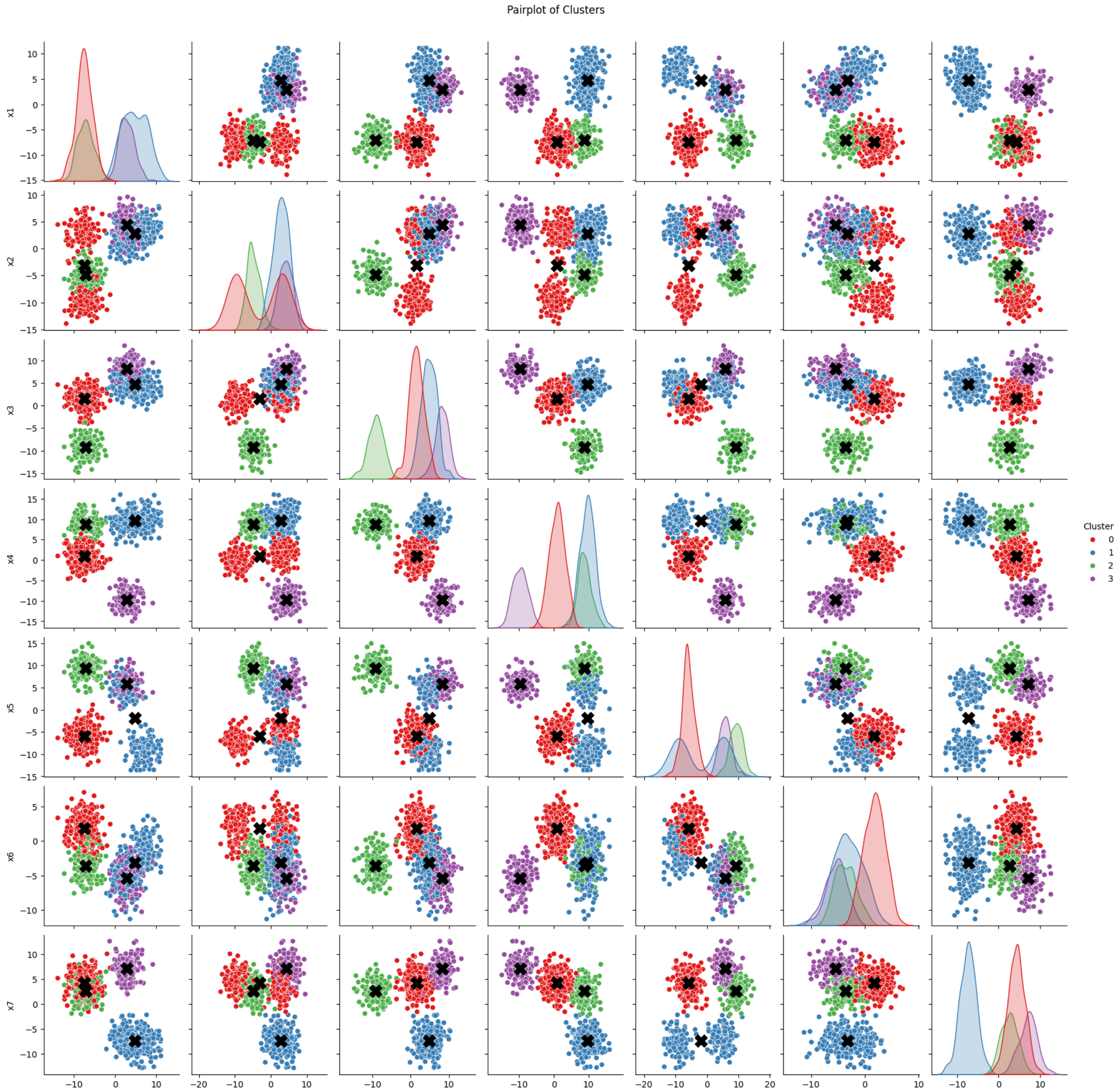
Print the intertia metric

```
kmeans.inertia_
```

```
51436.026458392946
```

Plot a scatter plot of your data using different color for each cluster. Also plot the centroids

```
pairplot = sns.pairplot(dataForPlot, hue='Cluster', palette='Set1')
plt.suptitle('Pairplot of Clusters', y=1.02)
centroids = kmeans.cluster_centers_
for i in range(len(pairplot.axes)):
    for j in range(len(pairplot.axes)):
        ax = pairplot.axes[i, j]
        if i != j:  # Skip the diagonal
            # Ploting centroids
            ax.scatter(
                centroids[:, j],
                centroids[:, i],
                color='black',
                marker='X',
                s=200,
                label='Centroids'
            )
plt.show()
```


Pairplot of Clusters
Cluster
0
1
2
3
x1
x2
x3
x4
x5
x6
x7

Questions

Provides a detailed description of your results

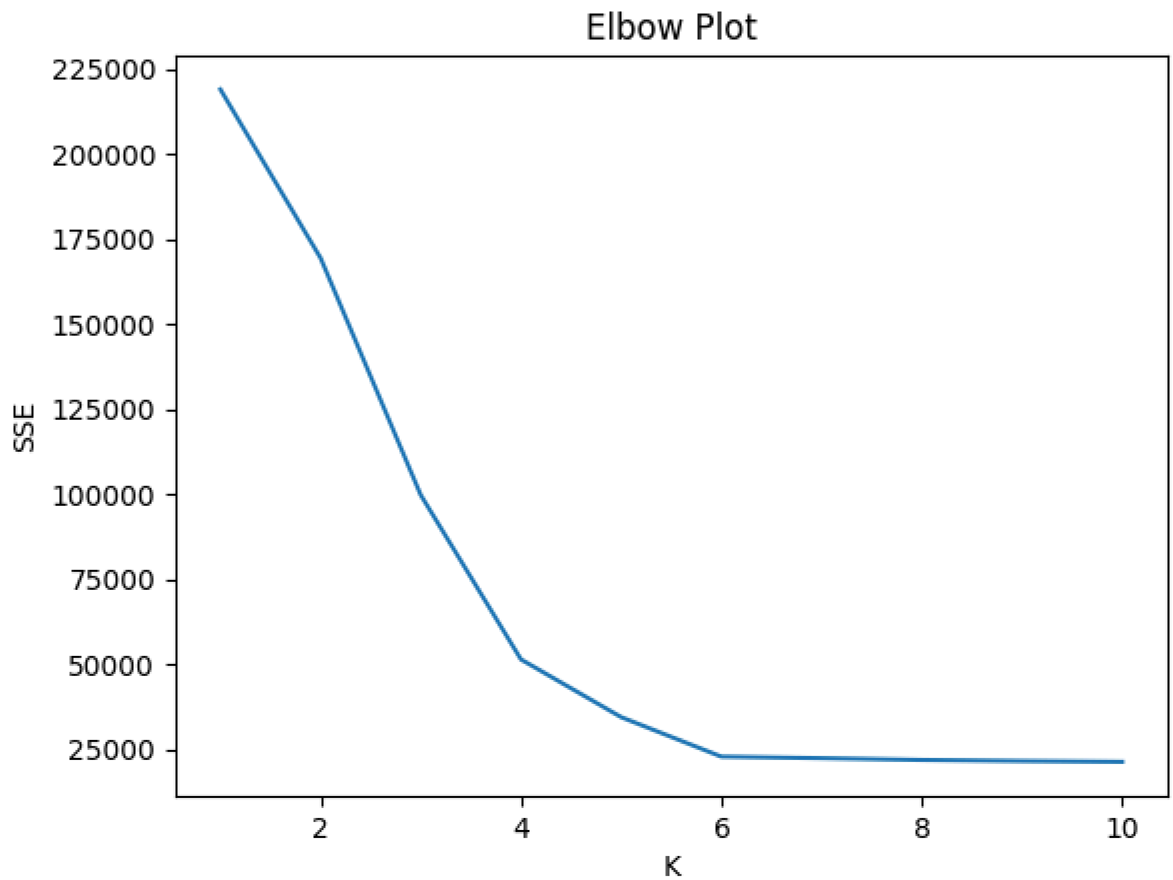
Your response: We can see clustering helped to properly create groups but there are some variable pairs where clusters seem to have not possible cluster association like x1 and x7.

d) Elbow plot

Compute the Elbow plot

```
# Intialize a list to hold sum of squared error (sse)
sse = []

# Define values of k
kR = range(1, 11)

# For each k
for k in kR:
  kmeans2 = KMeans(n_clusters=k, n_init="auto")
  kmeans2.fit_predict(dataForPlot[['x1','x2','x3','x4','x5','x6','x7']])
  sse.append(kmeans2.inertia_)


# Plot sse versus k
plt.plot(kR,sse)
plt.title('Elbow Plot')
```

```
plt.xlabel('K')
plt.ylabel('SSE')
plt.show()
```



Questions

What is the best number of clusters K? (argue your response)

Your response: It is 4 since the elbow plot shows it. It is the elbow point, meaning adding more clusters wont make it more accurate.

Does this number of clusters agree with your inital guess? (argue your response, no problem at all if they do not agree)

Your response: Yes since it is the same number.

PART 2

Do clustering using the "digits" dataset

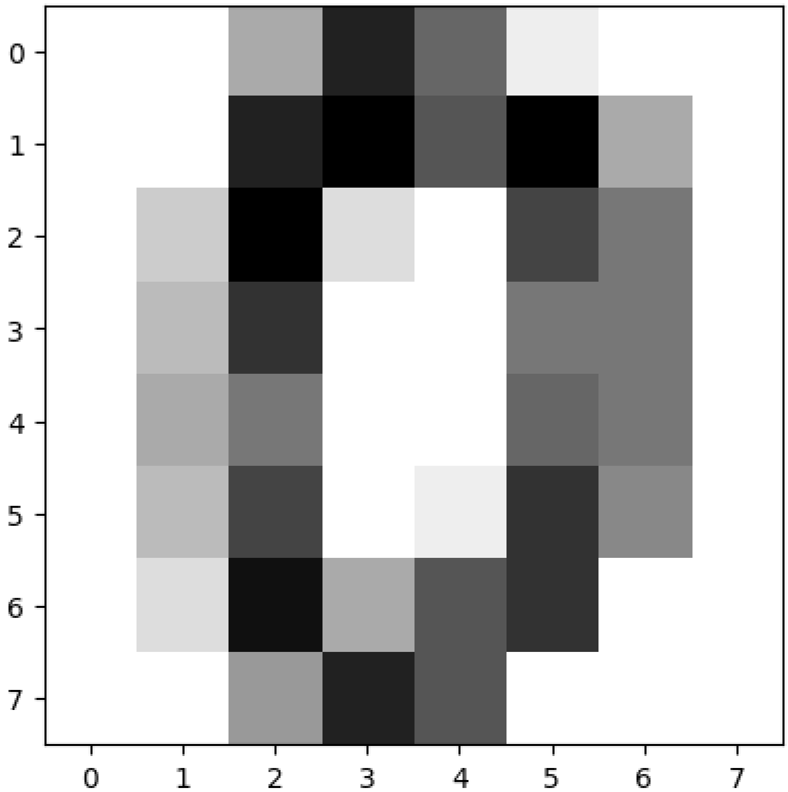
1) Load the dataset from "sklearn.datasets"

```
from sklearn import datasets
digits = datasets.load_digits()
```

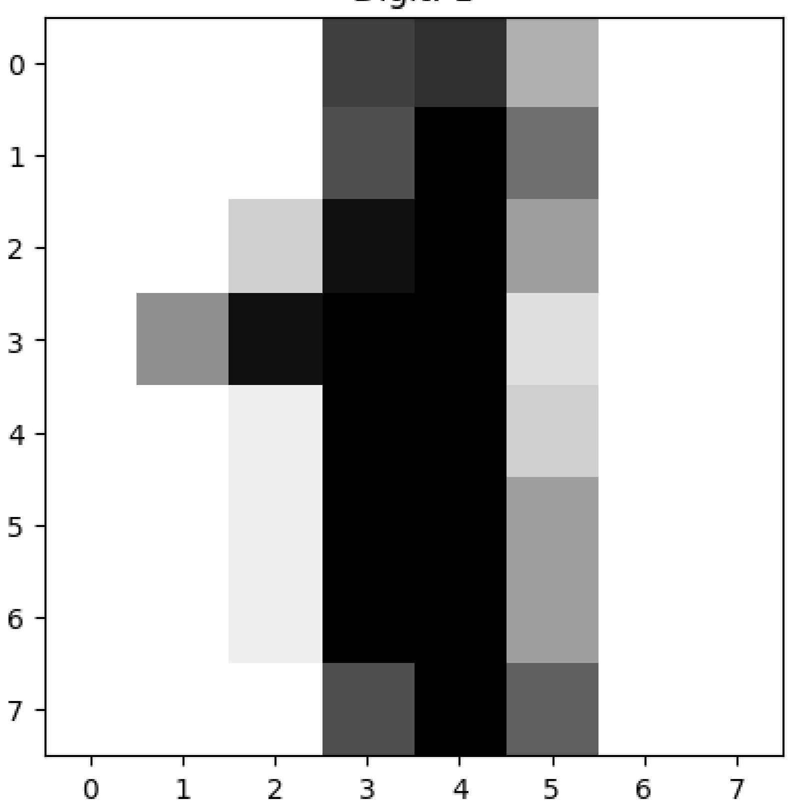
2) Plot some of the observations (add in the title the label/digit of that obserbation)

```
for i in range(10):  # Plot the first 10 images
  plt.figure()
  plt.imshow(digits.images[i], cmap=plt.cm.gray_r, interpolation='nearest')
  plt.title(f'Digit: {digits.target[i]}')
  plt.show()
```

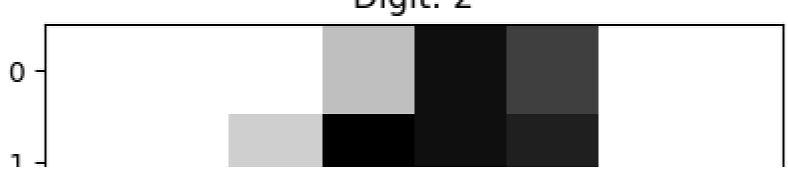
Digit: 0

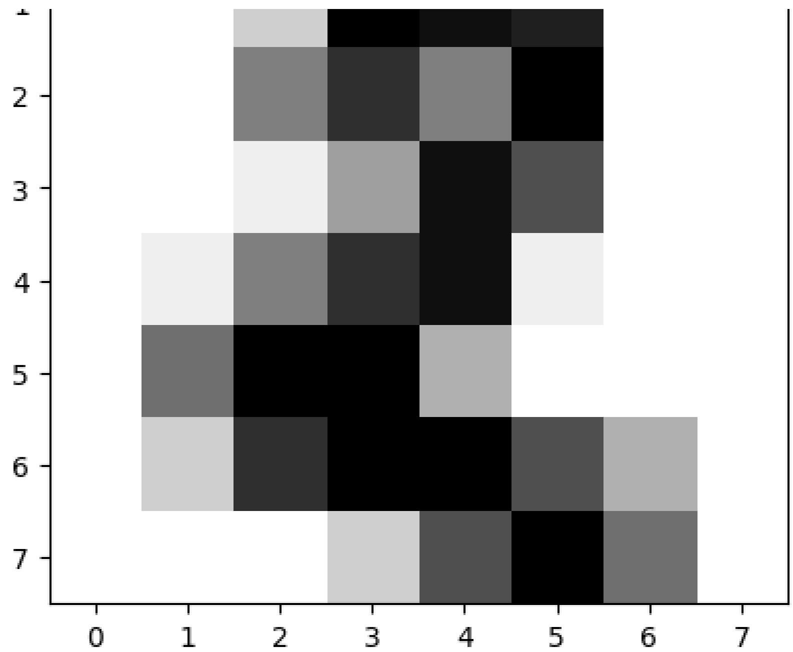


Digit: 1

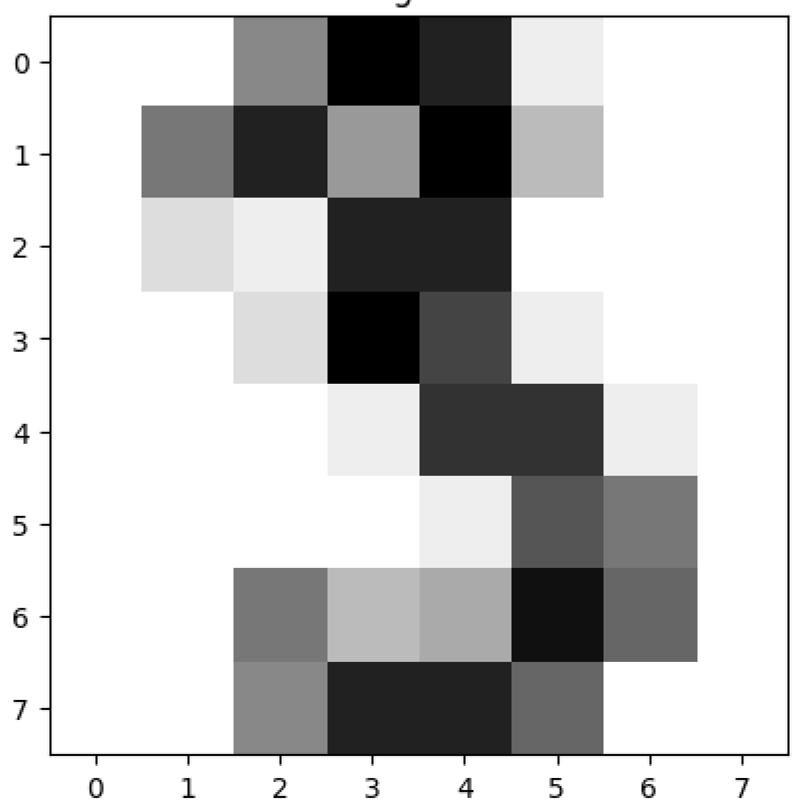


Digit: 2

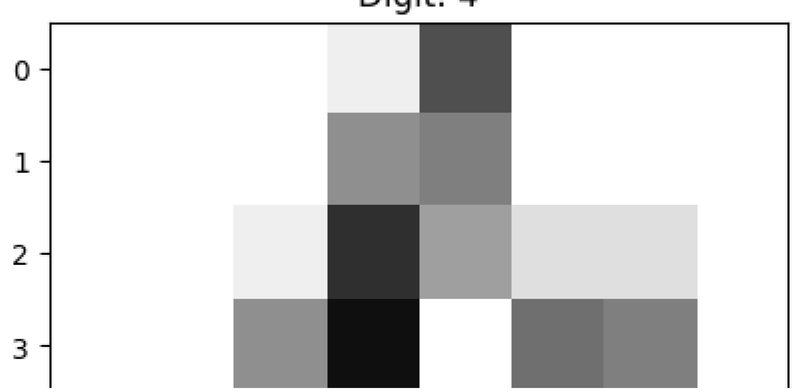


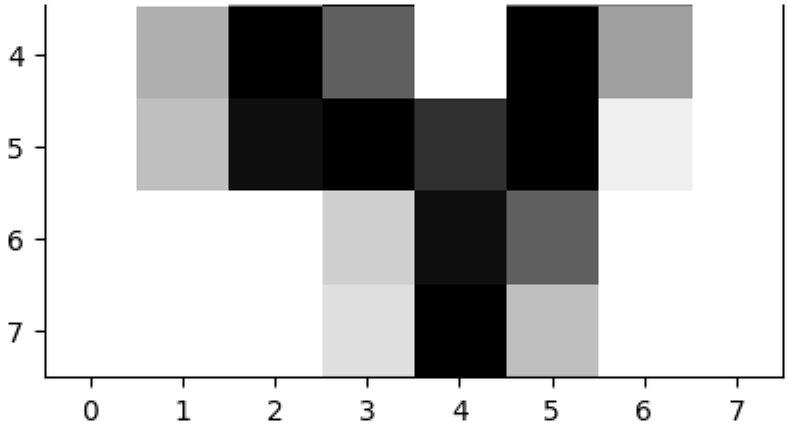


Digit: 3

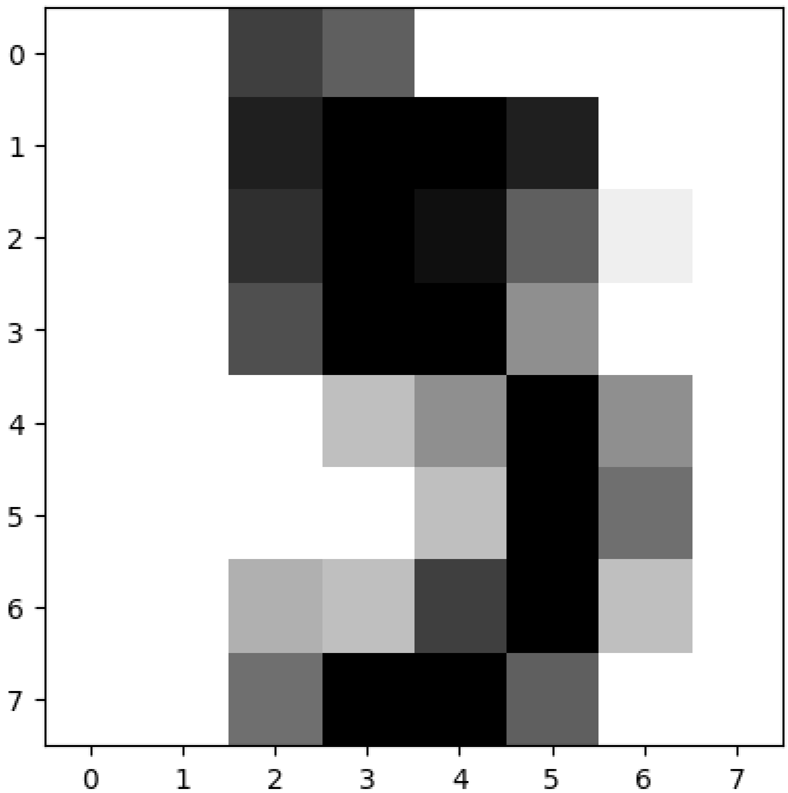


Digit: 4

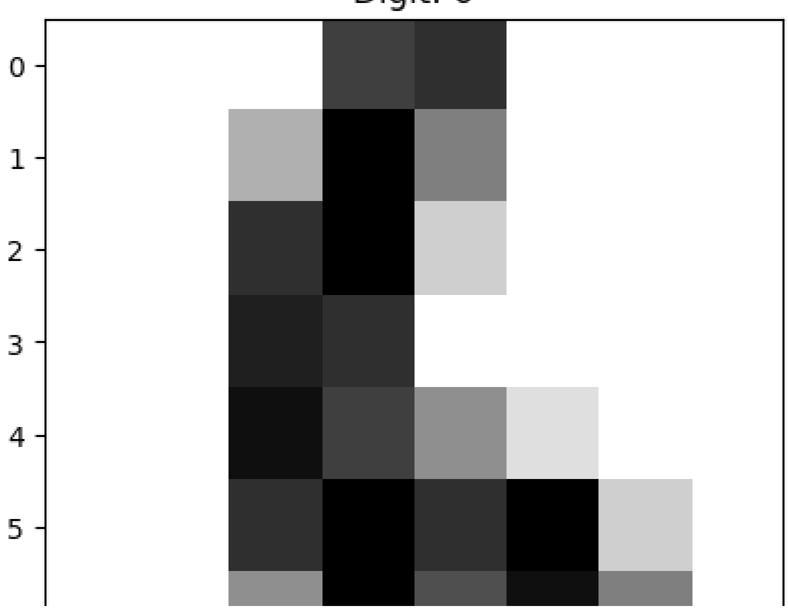


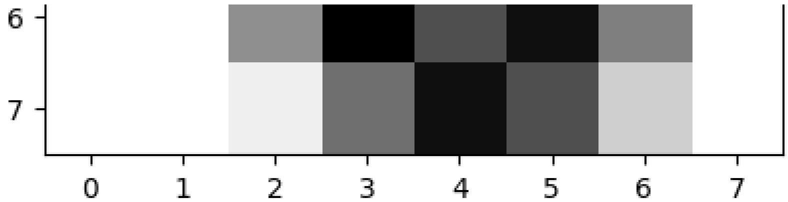


Digit: 5

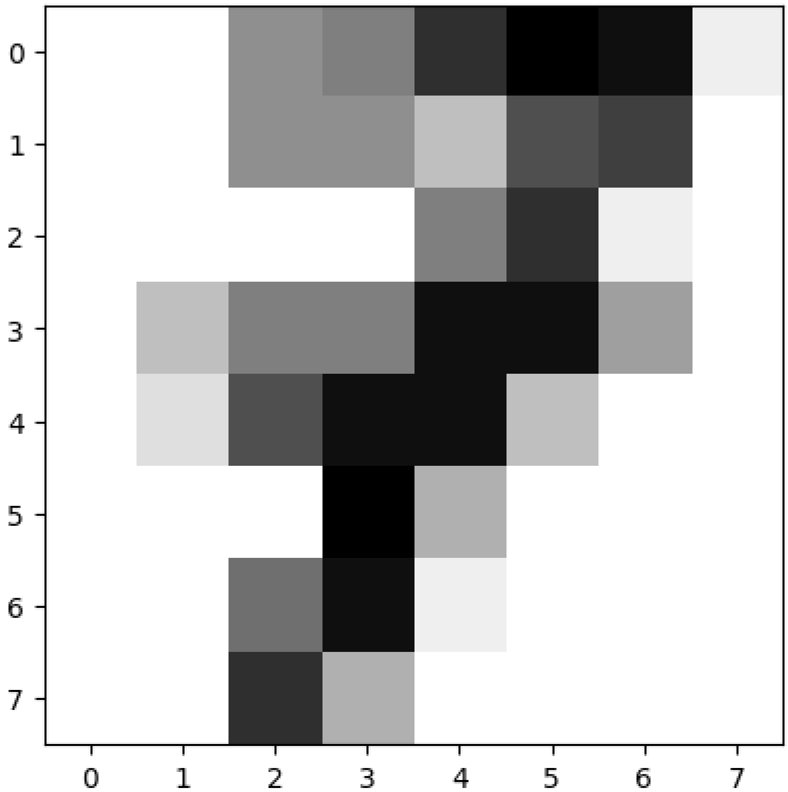


Digit: 6

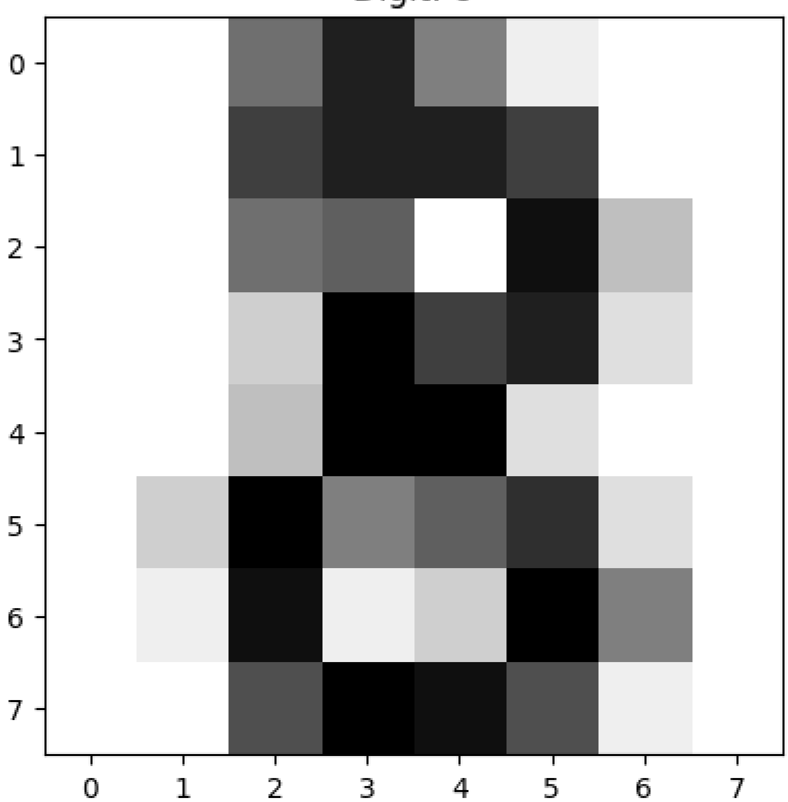


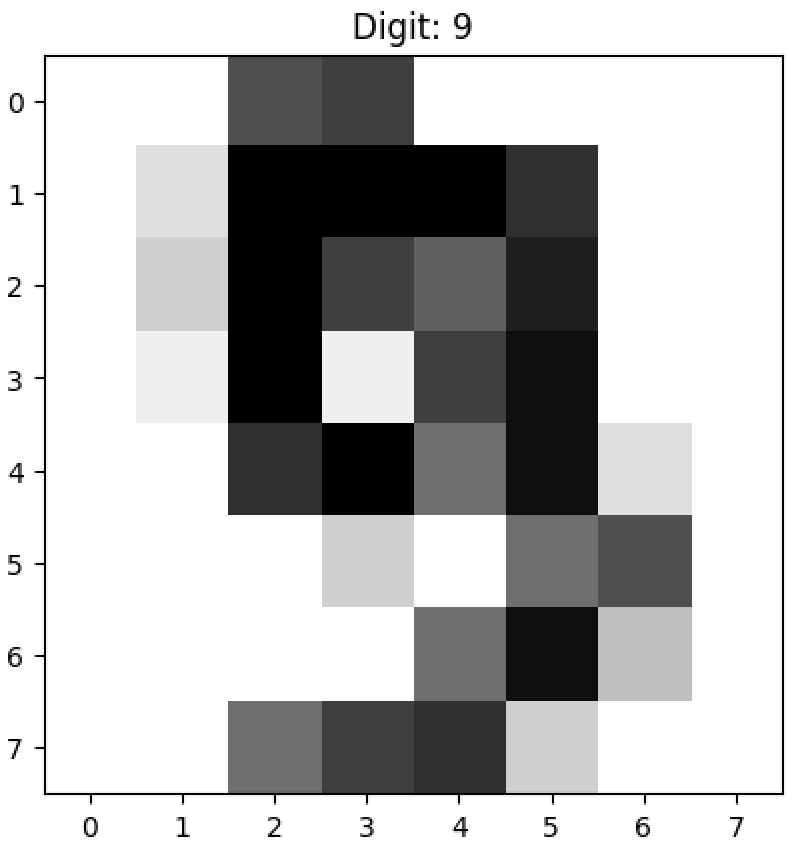


Digit: 7



Digit: 8




Digit: 9
0
1
2
3
4
5
6
7
0
1
2
3
4
5
6
7

3) Do K means clustering in the following cases:

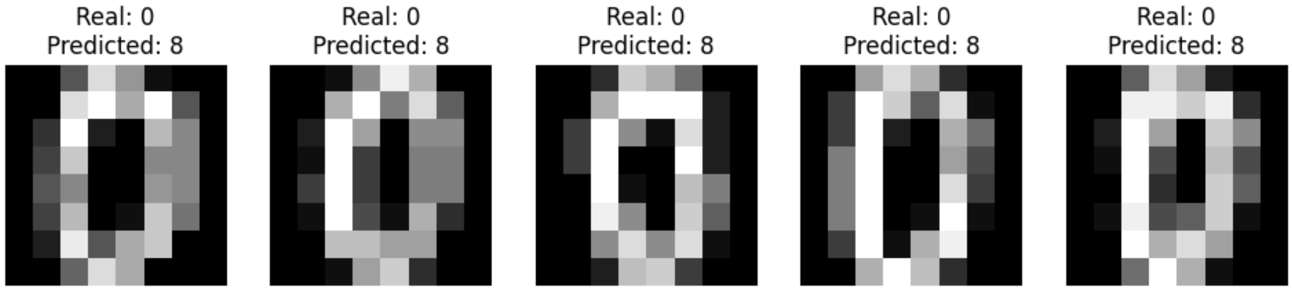
- KmeansAll: Using all 64 variables/pixels/features
- Kmeans1row: Using only the 8 variables/pixels/features from the firt row
- Kmeans4row: Using only the 8 variables/pixels/features from the fourth row
- Kmeans8row: Using only the 8 variables/pixels/ features from the eighth row

```
kmeansAll = KMeans(n_clusters = 10, random_state = 0).fit(digits.data)
kmeans1row = KMeans(n_clusters = 10, random_state = 0).fit(digits.data[:,:8])
kmeans4row = KMeans(n_clusters = 10, random_state = 0).fit(digits.data[:,24:32])
kmeans8row = KMeans(n_clusters = 10, random_state = 0).fit(digits.data[:,56:64])
```

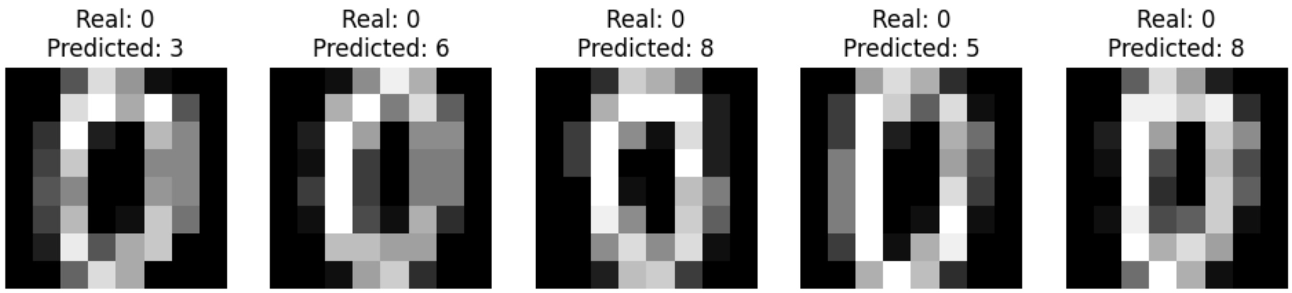
4) Verify your results. Plot several observations from the same digit and add in the title the real label and the estimated label to check in what observations the clusterization was correct or incorrect

```
def plot_digits_with_labels(real_labels, predicted_labels, digit, num_images=5):
    plt.figure(figsize=(12, 6))
    indices = np.where(real_labels == digit)[0][:num_images]

    for i, idx in enumerate(indices):
        plt.subplot(1, num_images, i + 1)
        plt.imshow(digits.images[idx], cmap='gray')
        plt.title(f'Real: {real_labels[idx]}\nPredicted: {predicted_labels[idx]}')
        plt.axis('off')
    plt.show()


for digit in range(10):
    print(f'Checking digit {digit} for KmeansAll')
    plot_digits_with_labels(digits.target, kmeansAll.labels_, digit)

    print(f'Checking digit {digit} for Kmeans1row')
    plot_digits_with_labels(digits.target, kmeans1row.labels_, digit)

    print(f'Checking digit {digit} for Kmeans4row')
    plot_digits_with_labels(digits.target, kmeans4row.labels_, digit)

    print(f'Checking digit {digit} for Kmeans8row')
    plot_digits_with_labels(digits.target, kmeans8row.labels_, digit)
```

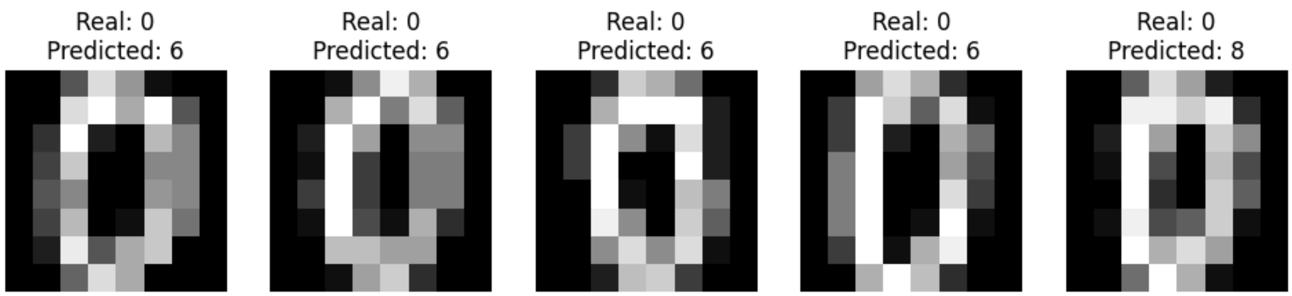
Checking digit 0 for KmeansAll



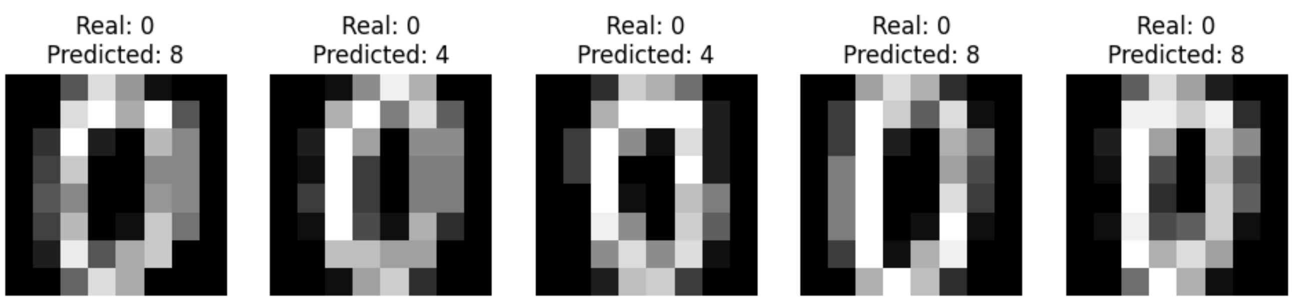
Checking digit 0 for Kmeans1row



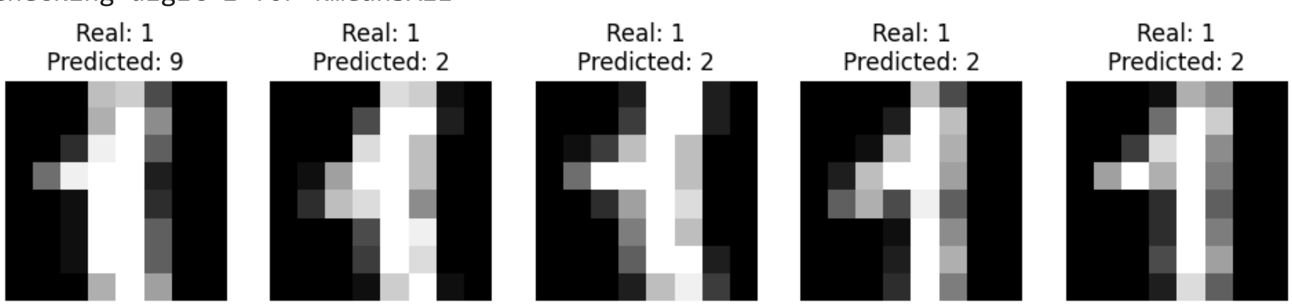
Checking digit 0 for Kmeans4row



Checking digit 0 for Kmeans8row



Checking digit 1 for KmeansAll



Checking digit 1 for Kmeans1row