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: Juan Andrés Castellanos Huerta
- ID: A01644650
- My carreer: Ingeniería en Tecnologías Computacionales

Importing libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

PART 1

Do clustering using your assigned dataset

a) Load data

```
# Define where you are running the code: colab or local
RunInColab          = True     # (False: no  | True: yes)

# If running in colab:
if RunInColab:
    # Mount your google drive in google colab
    from google.colab import drive
    drive.mount('/content/drive')

    # Find location
    #!pwd
    #!ls
    #!ls "/content/drive/My Drive/Colab Notebooks/MachineLearningWithPython/"

    # Define path del proyecto
    Ruta            = "/content/drive/MyDrive/Semana tec marzo 2025/"

else:
    # Define path del proyecto
    Ruta            = ""
```

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

```
# Dataset url
url = Ruta + "A01644650_X.csv"


# Load the dataset
df = pd.read_csv(url)
```

b) Data managment

Print the first 7 rows

```
df.head(7)
```

	Unnamed: 0	x1	x2	x3	x4	x5	x6	x7
0	0	-8.443283	-6.711509	-7.563889	-6.705161	6.025630	-9.202648	-1.191524
1	1	-0.793602	-4.521070	-8.648917	0.189100	1.081764	6.815486	-1.910381
2	2	4.747184	-9.606812	-1.038137	-1.944936	-8.105299	0.091450	-12.404338
3	3	0.635537	-2.992165	-11.047625	-0.017570	-2.461859	6.969979	2.839043
4	4	8.627383	-11.050553	0.785710	-0.338527	-9.573409	-2.453437	-6.685373
5	5	-6.006378	-7.866500	-5.677027	-7.440886	5.780359	-9.531860	-4.883754

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

Print the last 4 rows

```
1 df.tail(4)
```

	Unnamed: 0	x1	x2	x3	x4	x5	x6	x7
536	536	-1.081587	-5.241337	-12.415608	-4.109154	3.411486	4.926264	0.952135
537	537	-5.512595	-5.302669	-9.181149	5.117335	-4.474352	1.600528	1.110067
538	538	-9.420539	-3.681109	-5.460886	-6.409490	7.802554	-10.279281	-3.652791

How many rows and columns are in your data?

Use the `shape` method

```
1 df.shape
```

```
(540, 12)
```

Print the name of all columns

Use the `columns` method

```
1 df.columns
```

```
Index(['Unnamed: 0', 'x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9',
       'x10', 'x11'],
      dtype='object')
```

What is the data type in each column

Use the `dtypes` method

```
df.dtypes
```

	0
Unnamed: 0	int64
x1	float64
x2	float64
x3	float64
x4	float64
x5	float64
x6	float64
x7	float64
x8	float64
x9	float64
x10	float64
x11	float64

dtype: object

What is the meaning of rows and columns?

Your responses here

1. The rows reseprent different data that has been aquired, (different observations)
2. Th columns have different numbers that go from x1 to x11 it has different data on each with floats data types
3. It has a total of 540 rows and 12 columns

...

Print a statistical summary of your columns

```
mi=df.min()
ma=df.max()
```

```
print(mi)
print(ma)
```

```
Unnamed: 0     0.000000
x1           -15.298948
x2           -14.076504
x3           -14.302118
x4           -12.040445
x5           -13.371542
x6           -15.050158
x7           -12.404338
x8           -13.971648
x9           -11.758037
x10           -7.222599
x11          -12.633755
dtype: float64
Unnamed: 0    539.000000
x1             11.361727
x2              0.647744
x3              7.825923
x4              5.117335
x5             12.382946
x6             10.623852
x7              6.558263
x8             13.740902
x9             14.595974
x10            14.434888
x11            11.260577
dtype: float64
```

```
mea=df.mean(numeric_only=True)
print(mea)
```

```
Unnamed: 0    269.500000
x1             -0.840239
x2             -7.662682
x3             -4.298405
x4             -3.364727
x5             -0.218336
x6             -2.285926
x7             -2.369891
x8              2.053382
x9              2.452473
x10             4.116719
x11             0.921038
dtype: float64
```

```
df.std(numeric_only=True)
```

	0
Unnamed: 0	156.028843
x1	5.910860
x2	2.753696
x3	4.734016
x4	3.251603
x5	6.494812
x6	6.036222
x7	2.771170
x8	6.534232
x9	6.356627
x10	5.133264
x11	5.940069

dtype: float64

```
q1=np.quantile(df,0.25)
q2=np.quantile(df,0.50)
q3=np.quantile(df,0.75)
print(q1)
print(q2)
print(q3)
```

```
-5.612289623042992
-0.7151528411608956
5.643352408678116
```

1. What is the minumum and maximum values of each variable: minimum: x1 -15.298948 x2 -14.076504 x3 -14.302118 x4 -12.040445 x5 -13.371542 x6 -15.050158 x7 -12.404338 x8 -13.971648 x9 -11.758037 x10 -7.222599 x11 -12.633755 Maximum: x1 11.361727 x2 0.647744 x3 7.825923 x4 5.117335 x5 12.382946 x6 10.623852 x7 6.558263 x8 13.740902 x9 14.595974 x10 14.434888 x11 11.260577
2. What is the mean and standar deviation of each variable: mean: x1 -0.840239 x2 -7.662682 x3 -4.298405 x4 -3.364727 x5 -0.218336 x6 -2.285926 x7 -2.369891 x8 2.053382 x9 2.452473 x10 4.116719 x11 0.921038 std: x1 5.910860 x2 2.753696 x3 4.734016 x4 3.251603 x5 6.494812 x6 6.036222 x7 2.771170 x8 6.534232 x9 6.356627 x10 5.133264 x11 5.940069

3. What the 25%, 50% and 75% represent?: Los quartiles de los datos que son respectivamente: -5.612289623042992

-0.7151528411608956 5.643352408678116

Rename the columns using the same name with capital letters

```
1 df=df.rename(columns={"x1": "X1","x2":"X2","x3": "X3","x4": "X4","x5": "X5","x6": "X6",'
2 df.head()
```

	Unnamed: 0	X1	X2	X3	X4	X5	X6	X7
0	0	-8.443283	-6.711509	-7.563889	-6.705161	6.025630	-9.202648	-1.191524
1	1	-0.793602	-4.521070	-8.648917	0.189100	1.081764	6.815486	-1.910381
2	2	4.747184	-9.606812	-1.038137	-1.944936	-8.105299	0.091450	-12.404338
3	3	0.635537	-2.992165	-11.047625	-0.017570	-2.461859	6.969979	2.839043

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

Rename the columns to their original names

```
1 df=df.rename(columns={"X1": "x1","X2":"x2","X3": "x3","X4": "x4","X5": "x5","X6": "x6",'
2 df.head()
```

	Unnamed: 0	x1	x2	x3	x4	x5	x6	x7
0	0	-8.443283	-6.711509	-7.563889	-6.705161	6.025630	-9.202648	-1.191524
1	1	-0.793602	-4.521070	-8.648917	0.189100	1.081764	6.815486	-1.910381
2	2	4.747184	-9.606812	-1.038137	-1.944936	-8.105299	0.091450	-12.404338
3	3	0.635537	-2.992165	-11.047625	-0.017570	-2.461859	6.969979	2.839043

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

Use two different alternatives to get one of the columns

```
1 fa=df.x1
2 sa=df["x1"]
```

```
print(fa)
print(sa)
```

```
0     -8.443283
1     -0.793602
2      4.747184
3      0.635537
4      8.627383
         ...
535   -7.118096
536   -1.081587
537   -5.512595
538   -9.420539
539   -2.323127
Name: x1, Length: 540, dtype: float64
0     -8.443283
1     -0.793602
2      4.747184
3      0.635537
4      8.627383
         ...
535   -7.118096
536   -1.081587
537   -5.512595
538   -9.420539
539   -2.323127
Name: x1, Length: 540, dtype: float64
```

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

```
df.iloc[62:73,2:4]
```

	x2	x3
62	-11.793956	-4.383925
63	-9.193375	-5.055669
64	-6.632175	-13.230222
65	-5.435085	-4.999426
66	-9.966464	0.581427
67	-4.309668	-8.505675
68	-2.419614	-10.946440
69	-8.365948	-3.259590
70	-7.979246	-8.247228
71	-4.894526	-8.588602
72	-7.375261	-3.282595

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

```
print("Null data")
print(df.iloc[:,2:4].isnull().sum())
print("Not null data")
print(df.iloc[:,2:4].notnull().sum())
```

```
Null data
x2    0
x3    0
dtype: int64
Not null data
x2    540
x3    540
dtype: int64
```

Discard the last column

```
df.drop(columns="x11")
```

	Unnamed: 0	x1	x2	x3	x4	x5	x6	x7
0	0	-8.443283	-6.711509	-7.563889	-6.705161	6.025630	-9.202648	-1.19152
1	1	-0.793602	-4.521070	-8.648917	0.189100	1.081764	6.815486	-1.91038
2	2	4.747184	-9.606812	-1.038137	-1.944936	-8.105299	0.091450	-12.40433
3	3	0.635537	-2.992165	-11.047625	-0.017570	-2.461859	6.969979	2.83904
4	4	8.627383	-11.050553	0.785710	-0.338527	-9.573409	-2.453437	-6.68537
...	...	...	...	...	...	...	...	..
535	535	-7.118096	-2.622957	-8.049187	-6.455791	6.761529	-9.211781	-3.77941
536	536	-1.081587	-5.241337	-12.415608	-4.109154	3.411486	4.926264	0.95213
537	537	-5.512595	-5.302669	-9.181149	5.117335	-4.474352	1.600528	1.11006
538	538	-9.420539	-3.681109	-5.460886	-6.409490	7.802554	-10.279281	-3.65279
539	539	-2.323127	-10.515889	-1.752999	-6.697097	2.575424	-2.623917	-0.47517

Questions

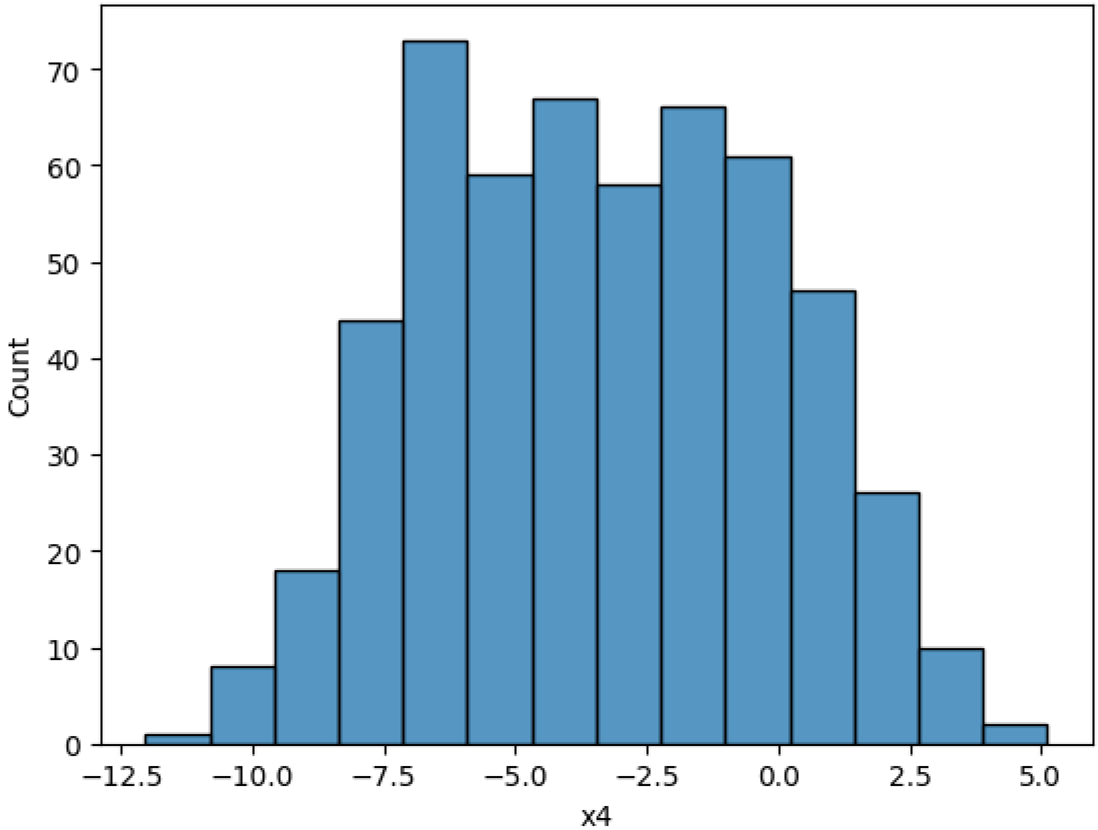
Based on the previos results, provide a full description of yout dataset

Your response: Los datos del data set son los siguientes, este tien un total de 12 columnas, la primera solo es una columna de enumeración, mientras que las otras 11 si contienen datos, estas ban del x1 al x11 y contienen datos tipo flotante, tiene un total de 540 filas donde la primera solo es de información, y las siguientes tienen datos.

c) Data visualization

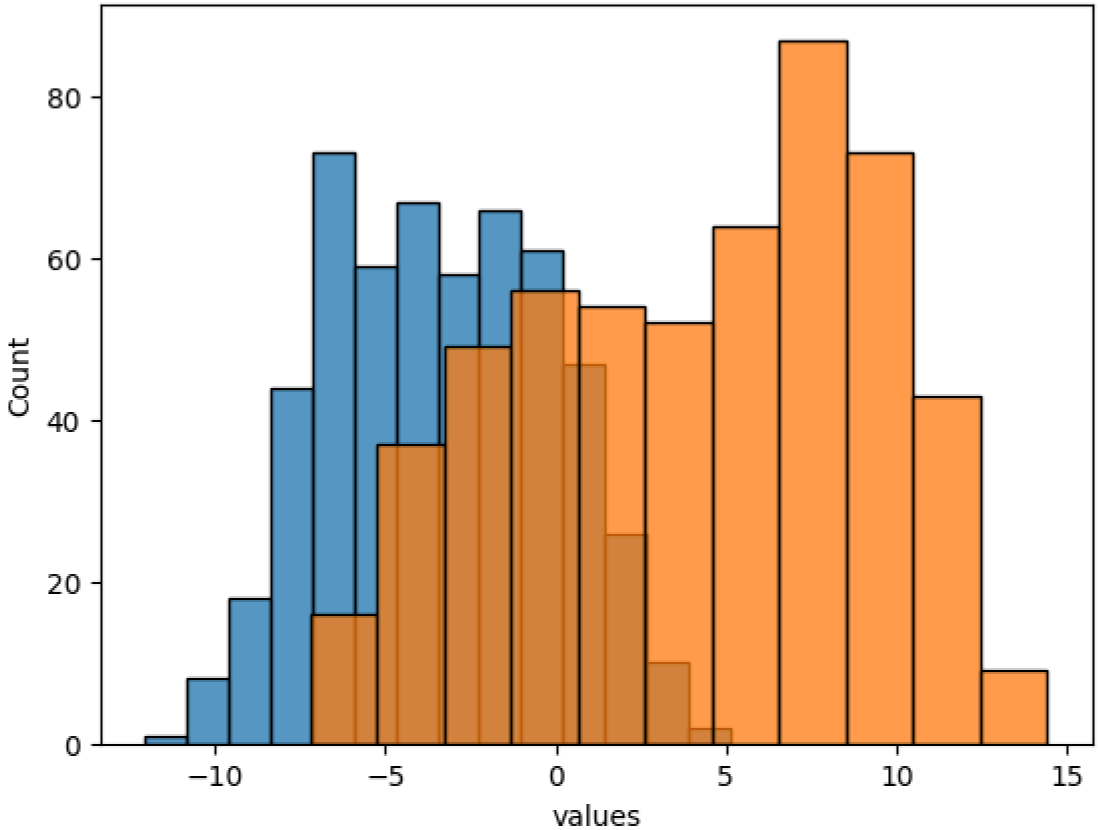
Plot in the histogram of one of the variables

```
1 sns.histplot(df.x4)
2 plt.show()
```



Plot in the same figure the histogram of two variables

```
sns.histplot(df.x4)
sns.histplot(df.x10)
plt.xlabel("values")
plt.show()
```

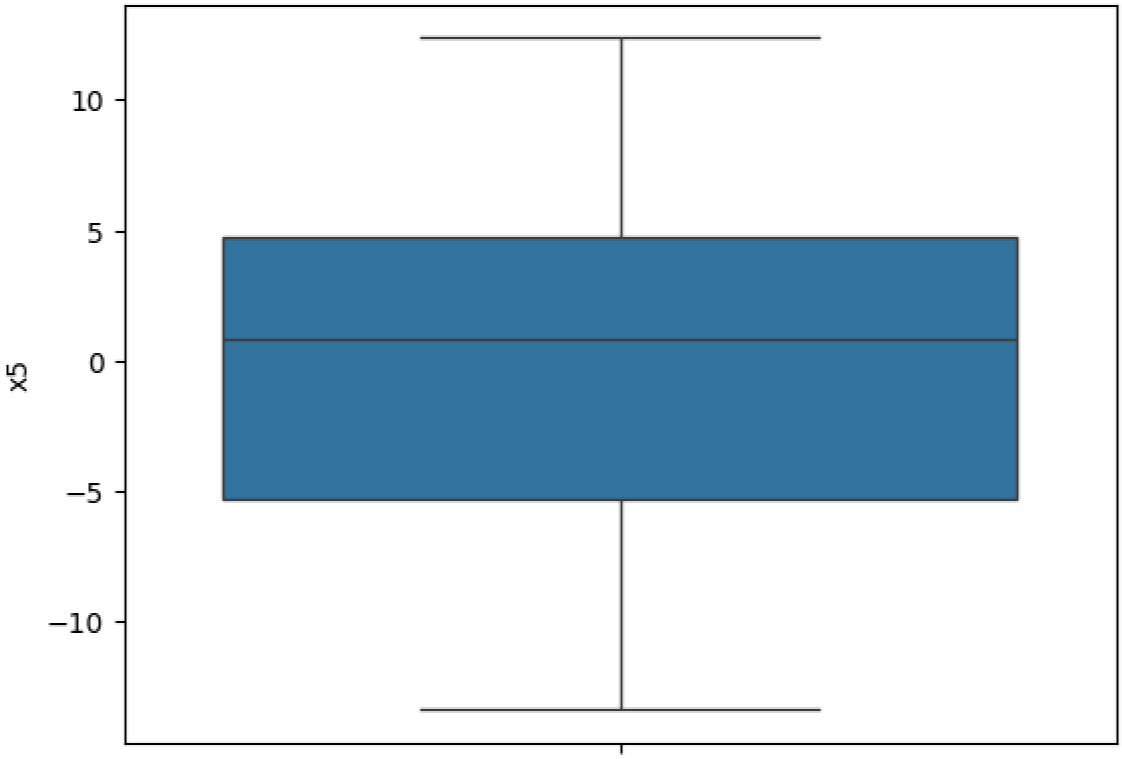


Based on these plots, provide a description of your data:

Your response here: Tienen un gran rango de valores que contienen tanto valores negativos como positivos, también hay varios valores que se encuentran mas en cierto rango, por ejemplo en x4 estos son del -7.5 al -5.0 y sus valores van del -12.5 aprox al 5 aprox, en e caso de x10 van del -7.5 aprox al 15 aprox y la mayoría de sus valores estan entre 5 y 10.

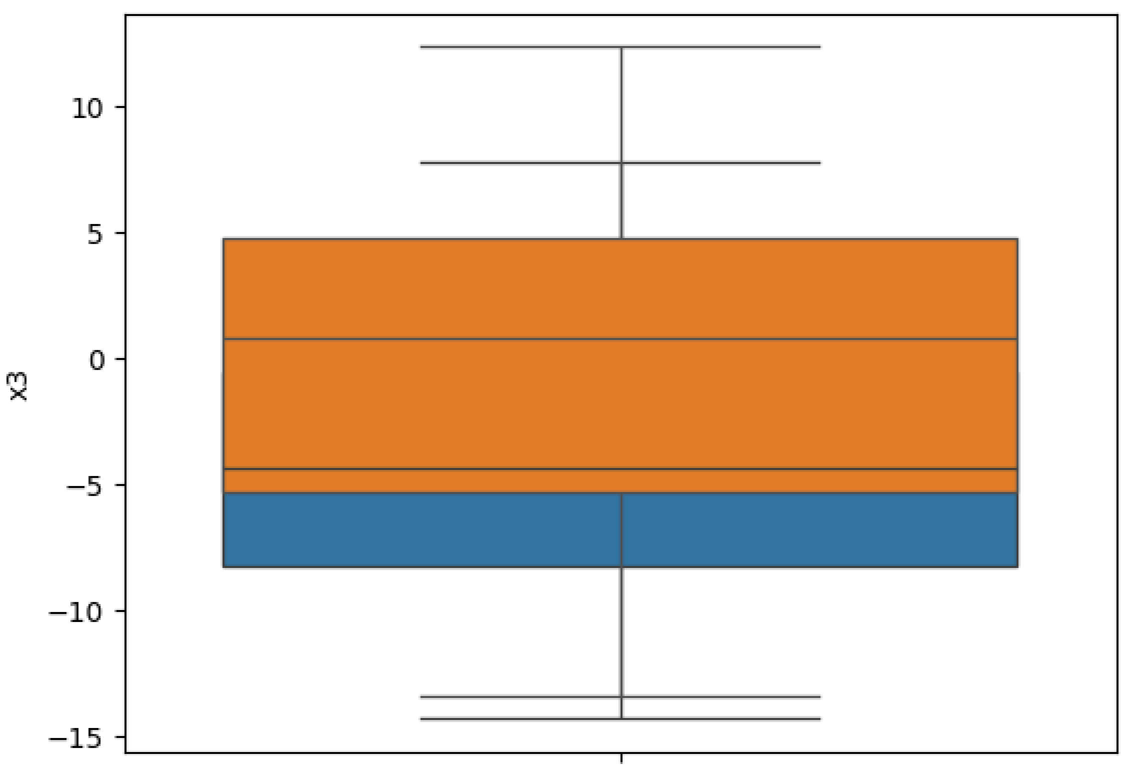
Plot the boxplot of one of the variables

```
sns.boxplot(df.x5)
plt.show()
```



Plot in the same figure the boxplot of two variables

```
sns.boxplot(df.x3)
sns.boxplot(df.x5)
plt.show()
```

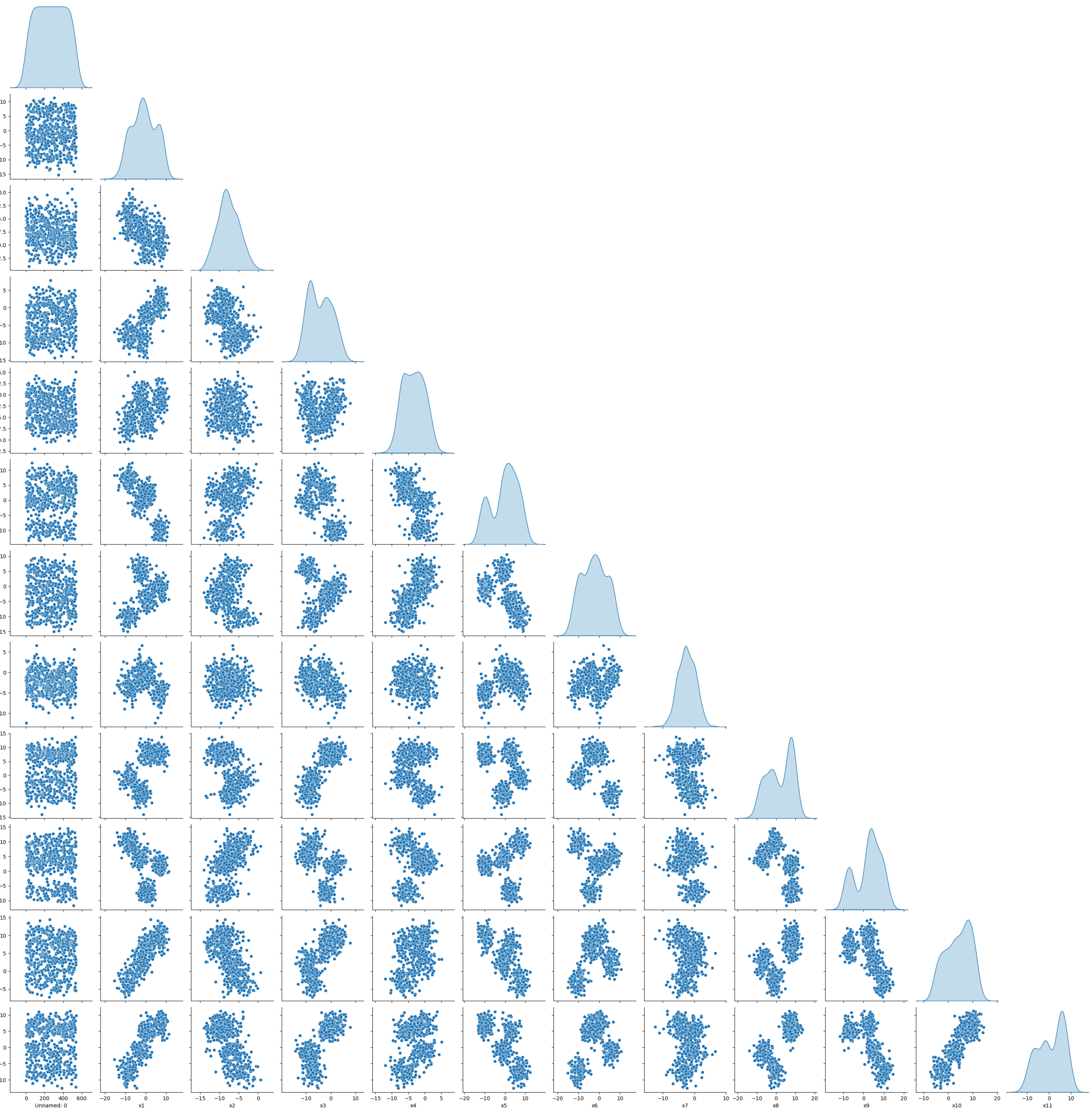


Based on these plots, provide a description of your data:

Your response here: Este muestra los cuartiles de las columnas x3 y x5 que se pueden ver representados con lineas y zonas coloreadas, también se muetra la media que sería el segundo cuartil y por ultimo también en el caso de haberlos, se mostrarían los valores atípicos

Plot the scatter plot between all pair of variables

```
sns.pairplot(df, corner=True,  diag_kind="kde")
plt.show()
```


Unnamed: 0
x1
x2
x3
x4
x5
x6
x7
x8
x9
x10
x11

Questions

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

Your response: Este plot que se hizo muestra la correlación de las diferentes variables utilizando un scatterplot y según que tanto esten correlacionadas es como aparece la grafica, por eso en los lugares que se compara la correlacion de los mismos valores no se ve tanto un scatterplot ya que su correlación es 1

d) Kmeans

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

```
from sklearn.cluster import KMeans

km = KMeans(n_clusters=3)

Cluster1 = km.fit_predict(df[["x7","x11"]])

Cluster1
```

```
array([0, 1, 2, 1, 2, 0, 2, 0, 1, 2, 0, 2, 1, 1, 0, 0, 2, 0, 1, 2, 1, 2,
       1, 2, 1, 2, 2, 1, 2, 1, 0, 2, 0, 2, 1, 0, 2, 0, 2, 1, 0, 2, 1, 0,
       1, 2, 1, 2, 0, 2, 2, 2, 2, 1, 0, 1, 1, 1, 1, 0, 2, 2, 2, 1, 0, 0,
       2, 0, 0, 2, 1, 0, 2, 1, 1, 2, 1, 2, 1, 0, 2, 2, 1, 2, 1, 2, 1, 1,
       0, 0, 1, 1, 1, 2, 0, 1, 2, 0, 2, 0, 0, 0, 0, 0, 1, 1, 2, 2, 0, 2,
       1, 2, 1, 2, 1, 2, 0, 2, 2, 0, 1, 0, 2, 2, 2, 2, 2, 1, 2, 0, 1, 1,
       0, 2, 2, 2, 1, 1, 2, 2, 2, 0, 2, 1, 2, 1, 1, 0, 2, 2, 2, 2, 2, 2,
       0, 2, 2, 0, 0, 0, 1, 1, 1, 1, 1, 2, 1, 0, 1, 2, 1, 2, 0, 1, 2, 1,
```

```
2, 2, 2, 2, 1, 2, 0, 1, 1, 2, 1, 2, 2, 2, 2, 1, 0, 1, 1, 0, 1, 1,
1, 2, 0, 1, 1, 0, 1, 0, 1, 2, 1, 0, 0, 1, 2, 0, 0, 2, 1, 2, 2, 1,
2, 0, 0, 2, 2, 2, 2, 0, 2, 1, 1, 2, 2, 0, 0, 2, 1, 2, 0, 1, 1, 0,
1, 2, 2, 2, 2, 2, 1, 1, 1, 2, 0, 0, 1, 2, 0, 2, 2, 0, 1, 0, 1, 2,
1, 2, 1, 0, 2, 2, 0, 0, 2, 2, 0, 0, 1, 1, 2, 2, 2, 1, 2, 1, 2, 1,
2, 2, 2, 2, 2, 0, 1, 0, 2, 0, 2, 1, 0, 2, 0, 1, 1, 2, 1, 2, 1, 1,
2, 2, 0, 0, 1, 2, 0, 1, 2, 2, 0, 2, 1, 2, 2, 0, 2, 0, 0, 1, 2, 0,
0, 0, 0, 1, 1, 2, 2, 0, 1, 2, 0, 1, 2, 2, 2, 2, 2, 1, 1, 2, 2, 0,
2, 0, 2, 2, 0, 0, 0, 2, 1, 0, 2, 1, 0, 0, 1, 0, 2, 0, 0, 2, 2, 0,
2, 1, 1, 2, 1, 0, 2, 1, 1, 2, 1, 2, 0, 1, 1, 1, 1, 2, 2, 2, 2, 2,
2, 1, 2, 2, 2, 1, 1, 1, 0, 2, 0, 1, 2, 1, 0, 2, 1, 1, 2, 0, 0, 1,
1, 1, 2, 1, 1, 2, 2, 1, 1, 0, 1, 2, 2, 2, 2, 0, 2, 1, 0, 2, 2, 2,
2, 2, 2, 2, 0, 0, 0, 0, 2, 0, 2, 2, 1, 2, 0, 2, 1, 0, 2, 1, 0, 2,
0, 0, 1, 2, 2, 0, 0, 0, 2, 1, 2, 2, 2, 1, 0, 2, 2, 0, 1, 1, 1, 2,
2, 0, 2, 0, 0, 2, 2, 2, 2, 2, 0, 2, 0, 2, 1, 2, 1, 2, 2, 1, 1, 2,
1, 2, 0, 1, 1, 2, 2, 1, 2, 2, 2, 1, 1, 2, 1, 1, 2, 2, 0, 0, 2, 2,
0, 2, 2, 2, 0, 2, 0, 0, 1, 1, 0, 2], dtype=int32)
```

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

```
df["Cluster"] = Cluster1
df.head()
```

	Unnamed: 0	x1	x2	x3	x4	x5	x6	x7	
0	0	-8.443283	-6.711509	-7.563889	-6.705161	6.025630	-9.202648	-1.191524	-:
1	1	-0.793602	-4.521070	-8.648917	0.189100	1.081764	6.815486	-1.910381	-!
2	2	4.747184	-9.606812	-1.038137	-1.944936	-8.105299	0.091450	-12.404338	!
3	3	0.635537	-2.992165	-11.047625	-0.017570	-2.461859	6.969979	2.839043	-:

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

Print the number associated to each cluster

```
print(df.Cluster.value_counts())
```

```
Cluster
2    234
1    166
0    140
Name: count, dtype: int64
```

Print the centroids

```
print(km.cluster_centers_)
```

```
[[-3.27474122 -7.16436737]
 [-0.38152905 -0.17246978]
 [-3.23907546  6.53419522]]
```

Print the intertia metric

```
print(km.inertia_)
```

```
5486.201641920694
```

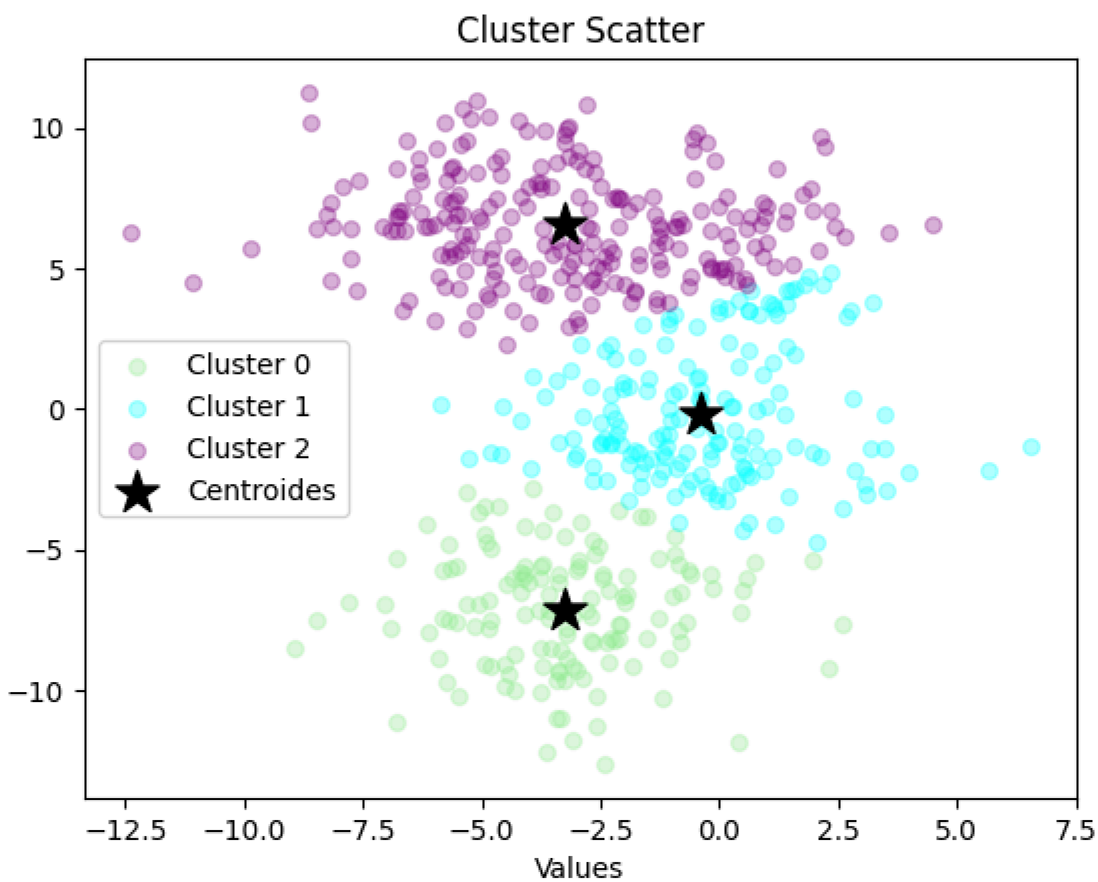
Plot a scatter plot of your data using different color for each cluster. Also plot the centroids

```
df1 = df[df.Cluster==0]
df2 = df[df.Cluster==1]
df3 = df[df.Cluster==2]

plt.scatter(df1.x7, df1.x11, label='Cluster 0', c='lightgreen', marker='o', s=32, alpha=
plt.scatter(df2.x7, df2.x11, label='Cluster 1', c='cyan', marker='o', s=32, alpha=0.3)
plt.scatter(df3.x7, df3.x11, label='Cluster 2', c='purple', marker='o', s=32, alpha=0.3)

plt.scatter(km.cluster_centers_[:,0], km.cluster_centers_[:,1], color='black', marker='*

plt.title('Cluster Scatter')
plt.xlabel('Values')
plt.legend()
plt.show()
```



Questions

Provides a detailed description of your results

Your response: En la tabla se puede ver un scatter plots de differentes datos de la información del csv, como se puede apreciar, tienen distintos colores y esto se debe a su posición respectiva al centroide es decir que dependiendo de a que distancia esten del centroide, es que color (grupo se les asigna)

d) Elbow plot

Compute the Elbow plot

```
# Intialize a list to hold sum of squared error (sse)
sse = []
# Define values of k
for k in range(1, 10):
# For each k
    km = KMeans(n_clusters=k, n_init="auto")
    km.fit(df[["x7","x11"]])
    sse.append(km.inertia_)
```

```
plt.style.use("fivethirtyeight")
plt.plot(range(1, 10), sse)
plt.xticks(range(1, 10))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.show()
```

Questions

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

Your response: Se puede decir que el cluster k=3 es el mejor por el hecho de que es el mas balanceado según los datos mostrados

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

Your response: El cluster que yo elegí fue el donde k=5 por el hecho de que sería el punto de enmedio, sin emabrgo, no es el mejor cluster que se puede tomar en este caso

PART 2

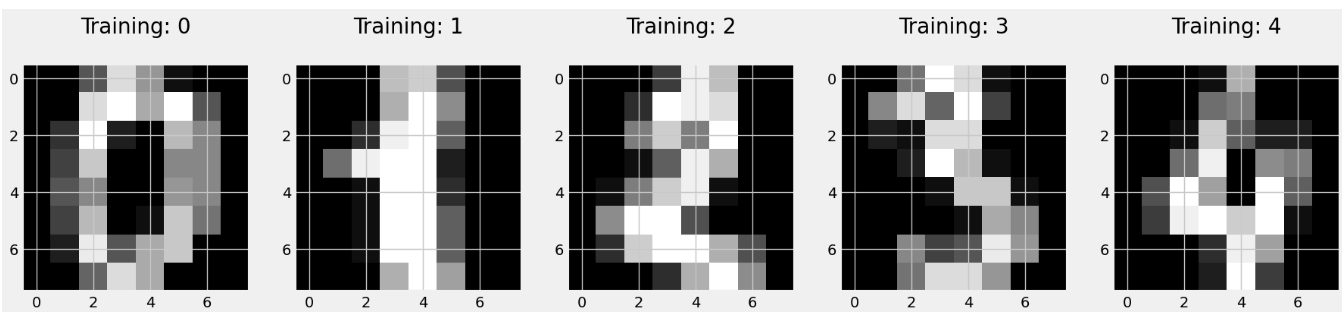
Do clustering using the "digits" dataset

Load the dataset from "sklearn.datasets"

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

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

```
plt.figure(figsize=(20, 4))
for index, (image, label) in enumerate(zip(digits.data[0:5], digits.target[0:5])):
    plt.subplot(1,5, index + 1)
    plt.imshow(np.reshape(image, (8,8)), cmap=plt.cm.gray)
    plt.title('Training: %i\n' % label, fontsize = 20)
```



3) Do K means clustering in the following cases:

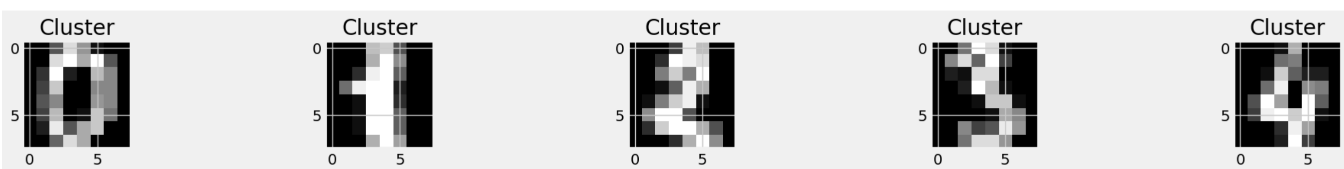
- KmeansAll: Using all 64 variables/pixels/features
- Kmeans1row: Using only the 8 variables/pixels/features from the firt row
- Kmeans4row: Using only the 8 variables/pixels/features from the fourth row
- Kmeans8row: Using only the 8 variables/pixels/ features from the eighth row

```
KmeansAll = KMeans(n_clusters=10, random_state=42)
KmeansAll.fit(digits.data)
Kmeans1row = KMeans(n_clusters=10,random_state=42)
Kmeans1row.fit(digits.data[:, :8])
Kmeans4row = KMeans(n_clusters=10, random_state=42)
Kmeans4row.fit(digits.data[:, 24:32])
Kmeans8row = KMeans(n_clusters=10, random_state=42)
Kmeans8row.fit(digits.data[:, 56:64])
```

```
KMeans
KMeans(n_clusters=10, random_state=42)
```

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

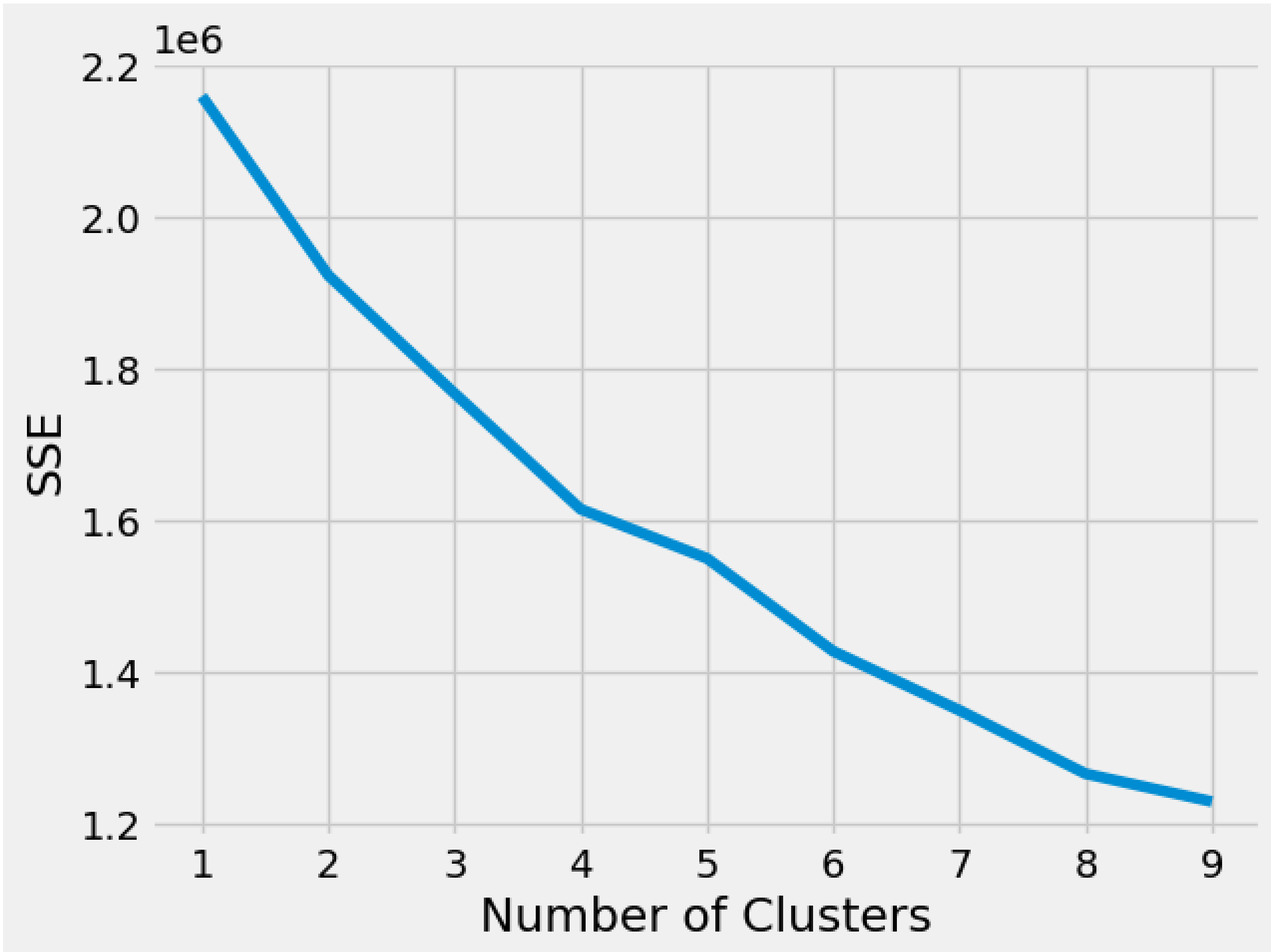
```
plt.figure(figsize=(20, 4))
for index, (image, real_label) in enumerate(zip(digits.data[0:5], digits.target[0:5])):
    estimated_label_all = KmeansAll.labels_[index]
    plt.subplot(2, 5, index + 6)
    plt.imshow(np.reshape(image, (8, 8)), cmap=plt.cm.gray)
    plt.title("Cluster")
plt.tight_layout()
plt.show()
```



5) Compute the Elbow plot

```
# Intialize a list to hold sum of squared error (sse)
sse = []
# Define values of k
for k in range(1, 10):
# For each k
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(digits.data)
    sse.append(kmeans.inertia_)
plt.style.use("fivethirtyeight")
plt.plot(range(1, 10), sse)
```

```
plt.xticks(range(1, 10))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.show()
```



Questions

Provides a detailed description of your results (e.g., in which case the clusterization is better, with KmeansAll, Kmeans1row, Kmeans4row, or Kmeans8row).

Your response (argue your response): Depende de de factores como la información a la que se le puede acceder para saber cual es el mejor metodo de clusterization, en este caso el mejor metodo de clusterization es el KmeansAll por que este tiene una mejor interpretación de los datos que se le brindan.

PART 3

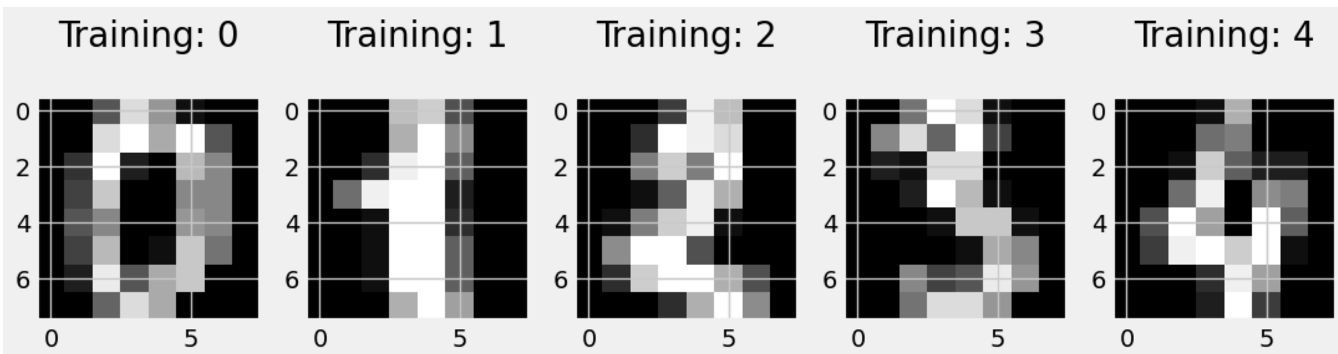
Do classification using the "digits" dataset

1) Load the dataset from "sklearn.datasets"

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

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

```
plt.figure(figsize=(12, 6))
for index, (image, label) in enumerate(zip(digits.data[0:5], digits.target[0:5])):
    plt.subplot(1,5, index + 1)
    plt.imshow(np.reshape(image, (8,8)), cmap=plt.cm.gray)
    plt.title('Training: %i\n' % label, fontsize = 20)
```



3) Split the dataset in train and test

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target, test_siz
```

4) Tune a classifier (Use the train set) in the following cases:

- ClassifierAll: Using all 64 variables/pixels/features
- Classifier1col: Using only the 8 variables/pixels/features from the firt column
- Classifier4col: Using only the 8 variables/pixels/features from the fourth column
- Classifier8col: Using only the 8 variables/pixels/ features from the eighth column

Note: in these four cases always use the same classification algorithm, e.g., a Suport Vector Machine

```
from sklearn.svm import SVC
classifierall = SVC()
classifierall.fit(X_train, y_train)
classifier1col = SVC()
classifier1col.fit(X_train[:, :1], y_train)
classifier4col = SVC()
classifier4col.fit(X_train[:, 3:4], y_train)
classifier8col = SVC()
classifier8col.fit(X_train[:, 7:8], y_train)
```

```
▾ SVC
SVC()
```

5) Make predictions (use the test set)

```
y_pred_all = classifierall.predict(X_test)
y_pred_1col = classifier1col.predict(X_test[:, :1])
y_pred_4col = classifier4col.predict(X_test[:, 3:4])
y_pred_8col = classifier8col.predict(X_test[:, 7:8])
```

6) Compute performance metrics

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred_all)
print(cm)
cm2 = confusion_matrix(y_test, y_pred_1col)
print(cm2)
cm3 = confusion_matrix(y_test, y_pred_4col)
print(cm3)
cm4 = confusion_matrix(y_test, y_pred_8col)
print(cm4)
```

```
[[33  0  0  0  0  0  0  0  0  0]
```