@author: Carlos Andrés Tejada Ramírez
Point 1 of Homework 1 from statistics III

Data exploration AND PCA (Point 1)

Use this data from the departments of Colombia to do the following tasks

Department	GDP Millions (x_1)	Population (x_2)	GDP per capita Millions (x_3)
Amazonas	1067855.672	76589	13.94267678
Antioquia	212514957.4	6407102	33.16865524
Arauca	8548114.653	262174	32.60473828
Atlántico	63764770.77	2535517	25.1486268
Bogotá D.C.	357258620.8	7412566	48.19634938
Bolívar	51404352.37	2070110	24.83170091
Boyacá	38858162.12	1217376	31.91960588
Caldas	23953112.45	998255	23.9949837
Caquetá	5461366.78	401849	13.59059443
Casanare	23660657.37	420504	56.26737766
Cauca	25758151.71	1464488	17.58850309
Cesar	37523918.98	1200574	31.25498218
Chocó	6001844.915	534826	11.2220515
Córdoba	24991953.76	1784783	14.00279685
Cundinamarca	91945942.28	2919060	31.49847632
Guainía	497704.0127	48114	10.34426597
Guaviare	1123857.696	82767	13.57857232
Huila	24011616.06	1100386	21.8210846
La Guajira	22262575.88	880560	25.28229295
Magdalena	19738417.36	1341746	14.710994
Meta	58439500.07	1039722	56.20685151
Nariño	21775426.15	1630592	13.35430699
Norte de Santander	23056874.23	1491689	15.45689097
Putumayo	5616558.269	348182	16.13109888
Quindío	11941644.16	539904	22.11808795
Risaralda	23786362.42	943401	25.21341659
San Andrés, Providencia y Santa Catalina (Archipiélago)	2125410.333	61280	34.68358898

Department	GDP Millions (x_1)	Population (x_2)	GDP per capita Millions (x_3)
Santander	92276678.16	2184837	42.23504003
Sucre	11516270.76	904863	12.7270877
Tolima	30438180.15	1330187	22.8826324
Valle del Cauca	139863153.5	4475886	31.2481492
Vaupés	381851.6785	40797	9.359797989
Vichada	956576.6785	107808	8.872965629

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

df = pd.read_csv("../Docs/Data/DEPARTMENTS.csv")

columnx_1 = df['GDP Millions (x_1)']
columnx_2 = df['Population (x_2)']
columnx_3 = df['GDP per capita Millions (x_3)']

df.describe()
```

```
       GDP Millions (x_1)  Population (x_2)  GDP per capita Millions
(x_3)
count        3.300000e+01      3.300000e+01
33.000000
mean         4.431886e+07      1.462379e+06
24.407856
std          7.167346e+07      1.705549e+06
12.766734
min          3.818517e+05      4.079700e+04
8.872966
25%          6.001845e+06      4.018490e+05
13.942677
50%          2.366066e+07      1.039722e+06
22.882632
75%          3.885816e+07      1.630592e+06
31.498476
max          3.572586e+08      7.412566e+06
56.267378
```

1. What is the mean, median, and standard deviation?, and the mode and mode repetition values for categorical data.

Mean

```
def calculateMean(column):
    return column.sum() / len(column)
```

```
# Calculate mean for each column
meanx_1 = calculateMean(columnx_1)
meanx_2 = calculateMean(columnx_2)
meanx_3 = calculateMean(columnx_3)

# Print results
print(f"Mean of GDP Millions (x_1): {meanx_1}")
print(f"Mean of Population (x_2): {meanx_2}")
print(f"Mean of GDP per capita Millions (x_3): {meanx_3}")
```

```
Mean of GDP Millions (x_1): 44318861.805990905
Mean of Population (x_2): 1462378.606060606
Mean of GDP per capita Millions (x_3): 24.40785586842424
```

Median

```
# Function to calculate the median
def calculateMedian(column):
    sortedColumn = sorted(column)
    n = len(sortedColumn)
    if n % 2 == 0:
        median = (sortedColumn[n//2 - 1] + sortedColumn[n//2]) / 2
    else:
        median = sortedColumn[n//2]
    return median

# Calculate median for each column
medianx_1 = calculateMedian(columnx_1)
medianx_2 = calculateMedian(columnx_2)
medianx_3 = calculateMedian(columnx_3)

# Print the results
print(f"Median of GDP Millions (x_1): {medianx_1}")
print(f"Median of Population (x_2): {medianx_2}")
print(f"Median of GDP per capita Millions (x_3): {medianx_3}")
```

```
Median of GDP Millions (x_1): 23660657.37
Median of Population (x_2): 1039722
Median of GDP per capita Millions (x_3): 22.8826324
```

Standard deviation

```
# Function to calculate the standard deviation
def calculateStdDev(column, mean):
    n = len(column)
    squaredDifferences = [(x - mean) ** 2 for x in column]
    variance = sum(squaredDifferences) / n
    stdDev = variance ** 0.5
    return stdDev
```

```
# Calculate standard deviation for each column
stdDevX1 = calculateStdDev(columnx_1, meanx_1)
stdDevX2 = calculateStdDev(columnx_2, meanx_2)
stdDevX3 = calculateStdDev(columnx_3, meanx_3)

# Print the results
print(f"Standard Deviation of GDP Millions (x_1): {stdDevX1}")
print(f"Standard Deviation of Population (x_2): {stdDevX2}")
print(f"Standard Deviation of GDP per capita Millions (x_3):
{stdDevX3}")
```

```
Standard Deviation of GDP Millions (x_1): 70579141.03250532
Standard Deviation of Population (x_2): 1679508.5872436275
Standard Deviation of GDP per capita Millions (x_3): 12.57181023560453
```

Mode and repetition values for categorical data

```
# Mode and value counts for categorical columns
modeValueCounts = {}
for column in df.select_dtypes(include=['object']).columns:
    modeValueCounts[column] = {
        'mode': df[column].mode().tolist(),
        'valueCounts': df[column].value_counts().to_dict()
    }

#Print results
print("\nMode and value counts for categorical columns:")
print(f"Mode:\n {df.mode().iloc[0]}")
for column, values in modeValueCounts.items():
    print(f"\nColumn: {column}")
    print(f"Mode:\n\t {values['mode']}")
    print(f"\nMode value counts:")
    for value, counts in values['valueCounts'].items():
        print(f"  {value}: {counts}")
```

```
Mode and value counts for categorical columns:
Mode:
 Department                          Amazonas
GDP Millions (x_1)                381851.6785
Population (x_2)                        40797
GDP per capita Millions (x_3)        8.872966
Name: 0, dtype: object

Column: Department
Mode:
      ['Amazonas', 'Antioquia', 'Arauca', 'Atlántico', 'Bogotá D.C.',
'Bolívar', 'Boyacá', 'Caldas', 'Caquetá', 'Casanare', 'Cauca',
'Cesar', 'Chocó', 'Cundinamarca', 'Córdoba', 'Guainía', 'Guaviare',
'Huila', 'La Guajira', 'Magdalena', 'Meta', 'Nariño', 'Norte de
```

```
Santander', 'Putumayo', 'Quindío', 'Risaralda', 'San Andrés
Providencia y Santa Catalina (Archipiélago)', 'Santander', 'Sucre',
'Tolima', 'Valle del Cauca', 'Vaupés', 'Vichada']

Mode value counts:
  Amazonas: 1
  Huila: 1
  Vaupés: 1
  Valle del Cauca: 1
  Tolima: 1
  Sucre: 1
  Santander: 1
  San Andrés Providencia y Santa Catalina (Archipiélago): 1
  Risaralda: 1
  Quindío: 1
  Putumayo: 1
  Norte de Santander: 1
  Nariño: 1
  Meta: 1
  Magdalena: 1
  La Guajira: 1
  Guaviare: 1
  Antioquia: 1
  Guainía: 1
  Cundinamarca: 1
  Córdoba: 1
  Chocó: 1
  Cesar: 1
  Cauca: 1
  Casanare: 1
  Caquetá: 1
  Caldas: 1
  Boyacá: 1
  Bolívar: 1
  Bogotá D.C.: 1
  Atlántico: 1
  Arauca: 1
  Vichada: 1
```

2.Draw a boxplot by hand. Using the data from table 1 and the following proportions.

Methods

```
import plotly.express as px
import plotly.graph_objects as go
import plotly.subplots as sp

def calculateQuartile(column, cuartile):
```

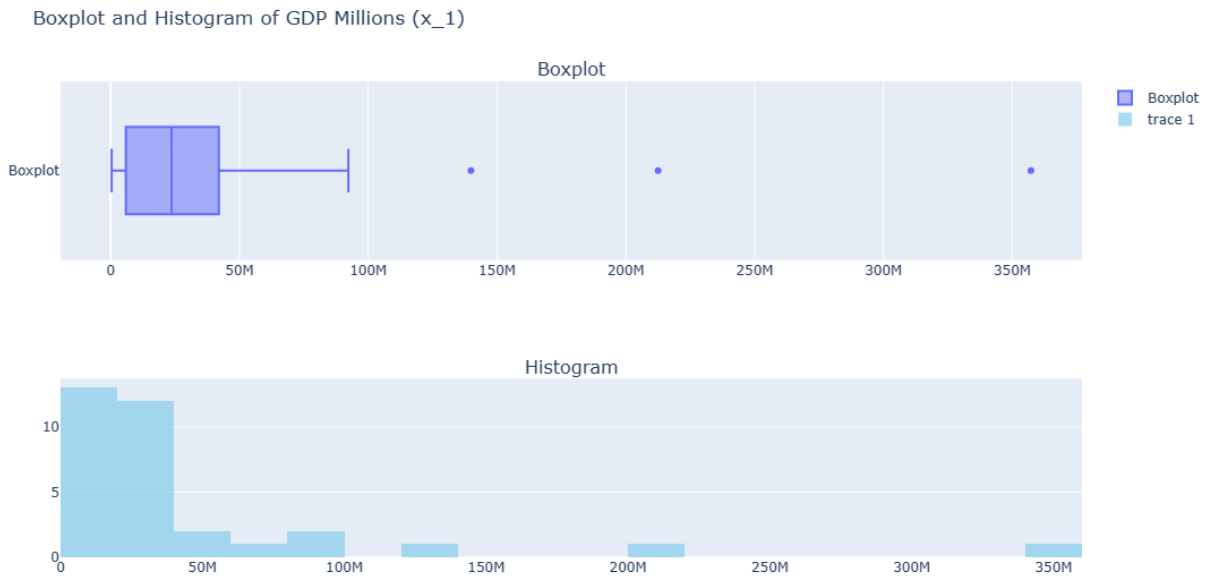
```
    column = sorted(column)
    columnLength = len(column)
    qIndex = cuartile * (columnLength // 4)

    return column[qIndex] if columnLength % 4 != 0 else
(column[qIndex] + column[qIndex + 1]) / 2

def calculateInterquartileRange(q1, q3):
    return q3 - q1

def getExtremeValues(column):
    column = sorted(column)
    # [Minimun, maximun]
    return [column[0], column[-1]]
```

Implementation

BOXPLOT GDP MILLIONS (x_1)

```
# calculate missing values from boxplot
q1x_1 = calculateQuartile(columnx_1, 1)
q3x_1 = calculateQuartile(columnx_1, 3)
iqrx_1 = calculateInterquartileRange(q1x_1, q3x_1)
upperx_1 = q3x_1 + 1.5 * iqrx_1
lowerx_1 = q1x_1 - 1.5 * iqrx_1
extremeValues = getExtremeValues(columnx_1)

# Print the calculated values
print(f"Maximum: {extremeValues[1]}")
print(f"Upper Bound: {upperx_1}")
print(f"Third Quartile (Q3): {q3x_1}")
print(f"Median (Q2): {medianx_1} AND Interquartile Range (IQR):
{iqrx_1}")
print(f"First Quartile (Q1): {q1x_1}")
print(f"Lower Bound: {lowerx_1}")
print(f"Minimum: {extremeValues[0]}")

# Create subcharts with two rows and one column
fig = sp.make_subplots(rows=2, cols=1, subplot_titles=['Boxplot',
'Histogram'])

# Add boxplot
fig.add_trace(go.Box(x = columnx_1, name='Boxplot'), row=1, col=1)

# Add histogram
fig.add_trace(go.Histogram(x = columnx_1, nbinsx=20,
marker_color='skyblue', opacity=0.7), row=2, col=1)

# show the graph
fig.update_layout(height=600, width=800, title_text='Boxplot and
```

```
Histogram of GDP Millions (x_1)')
fig.show()

Maximum: 357258620.8
Upper Bound: 88142637.9275
Third Quartile (Q3): 38858162.12
Median (Q2): 23660657.37 AND Interquartile Range (IQR): 32856317.205
First Quartile (Q1): 6001844.915
Lower Bound: -43282630.8925
Minimum: 381851.6785
```

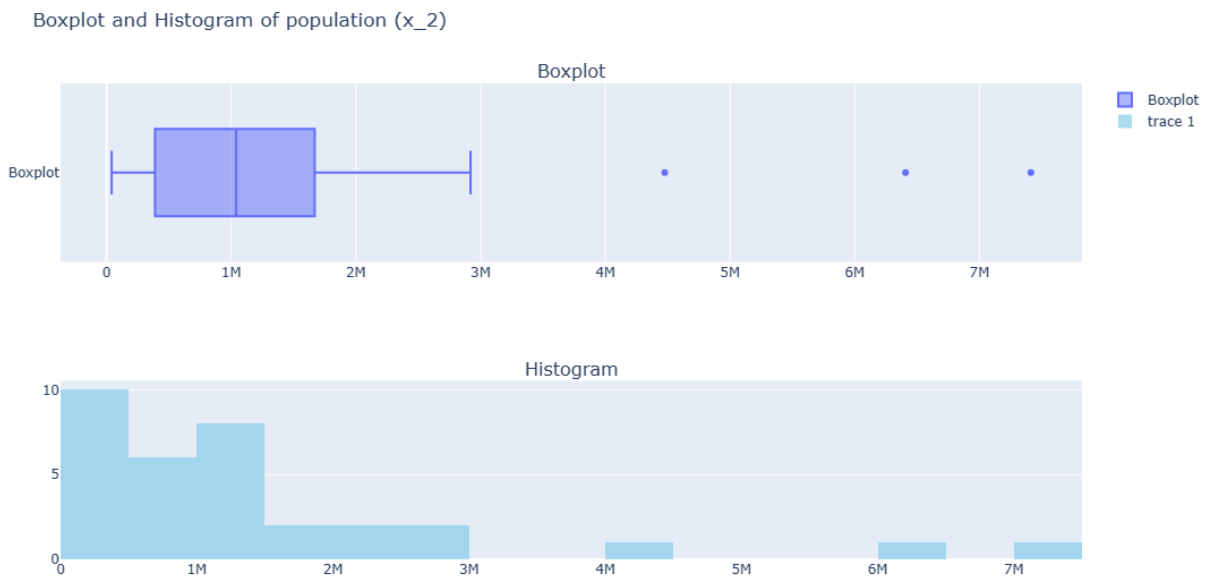


POPULATION (x_2)

```
# calculate missing values from boxplot
q1x_2 = calculateQuartile(columnx_2, 1)
q3x_2 = calculateQuartile(columnx_2, 3)
iqrx_2 = calculateInterquartileRange(q1x_2, q3x_2)
upperx_2 = q3x_2 + 1.5 * iqrx_2
lowerx_2 = q1x_2 - 1.5 * iqrx_2
extremeValues = getExtremeValues(columnx_2)

# Print the calculated values
print(f"Maximum: {extremeValues[1]}")
print(f"Upper Bound: {upperx_2}")
print(f"Third Quartile (Q3): {q3x_2}")
print(f"Median (Q2): {medianx_2} AND Interquartile Range (IQR):
{iqrx_2}")
print(f"First Quartile (Q1): {q1x_2}")
print(f"Lower Bound: {lowerx_2}")
```

```
print(f"Minimum: {extremeValues[0]}")

# Create subcharts with two rows and one column
fig = sp.make_subplots(rows=2, cols=1, subplot_titles=['Boxplot',
'Histogram'])

# Add boxplot
fig.add_trace(go.Box(x = columnx_2, name='Boxplot'), row=1, col=1)

# Add histogram
fig.add_trace(go.Histogram(x = columnx_2, nbinsx=20,
marker_color='skyblue', opacity=0.7), row=2, col=1)

# show the graph
fig.update_layout(height=600, width=800, title_text='Boxplot and
Histogram of population (x_2)')
fig.show()
```

```
Maximum: 7412566
Upper Bound: 3473706.5
Third Quartile (Q3): 1630592
Median (Q2): 1039722 AND Interquartile Range (IQR): 1228743
First Quartile (Q1): 401849
Lower Bound: -1441265.5
Minimum: 40797
```

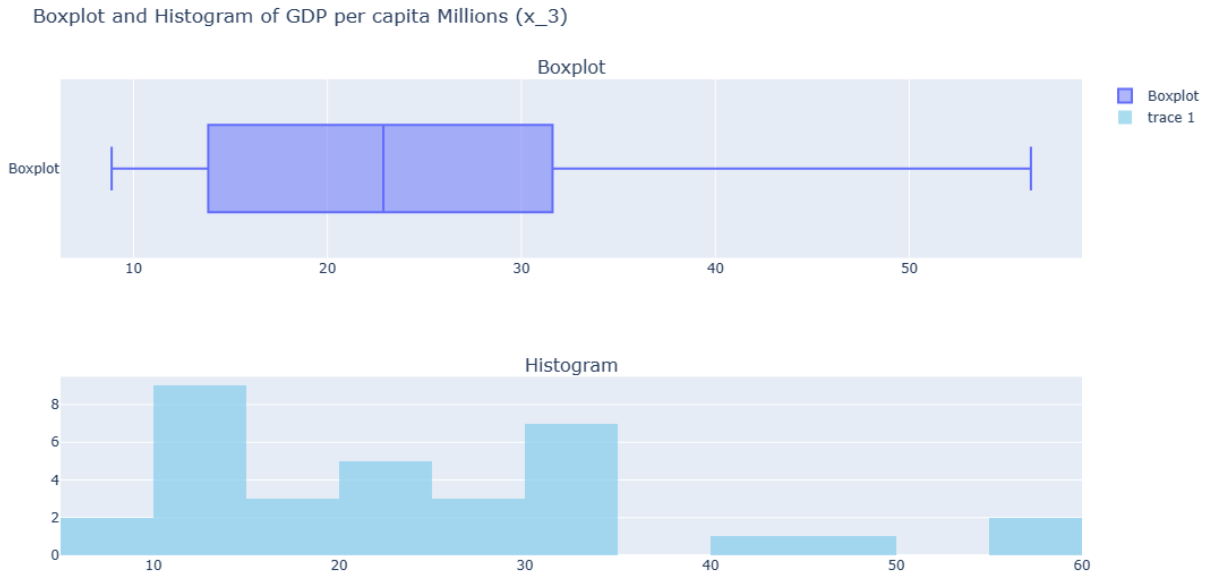


GDP per capita Millions (x_3)

```
# calculate missing values from boxplot
q1x_3 = calculateQuartile(columnx_3, 1)
```

```
q3x_3 = calculateQuartile(columnx_3, 3)
iqrx_3 = calculateInterquartileRange(q1x_3, q3x_3)
upperx_3 = q3x_3 + 1.5 * iqrx_3
lowerx_3 = q1x_3 - 1.5 * iqrx_3
extremeValues = getExtremeValues(columnx_3)

# Print the calculated values
print(f"Maximum: {extremeValues[1]}")
print(f"Upper Bound: {upperx_3}")
print(f"Third Quartile (Q3): {q3x_3}")
print(f"Median (Q2): {medianx_3} AND Interquartile Range (IQR):
{iqrx_3}")
print(f"First Quartile (Q1): {q1x_3}")
print(f"Lower Bound: {lowerx_3}")
print(f"Minimum: {extremeValues[0]}")

# Create subcharts with two rows and one column
fig = sp.make_subplots(rows=2, cols=1, subplot_titles=['Boxplot',
'Histogram'])

# Add boxplot
fig.add_trace(go.Box(x = columnx_3, name='Boxplot'), row=1, col=1)

# Add histogram
fig.add_trace(go.Histogram(x = columnx_3, nbinsx=20,
marker_color='skyblue', opacity=0.7), row=2, col=1)

# show the graph
fig.update_layout(height=600, width=800, title_text='Boxplot and
Histogram of GDP per capita Millions (x_3)')
fig.show()
```

```
Maximum: 56.26737766
Upper Bound: 57.83217563000001
Third Quartile (Q3): 31.49847632
Median (Q2): 22.8826324 AND Interquartile Range (IQR):
17.555799540000002
First Quartile (Q1): 13.94267678
Lower Bound: -12.391022530000004
Minimum: 8.872965629
```



OBSERVATION

A difference can be observed between the values calculated in python for Q1, Q3, IQR, LOWER and UPPER compared to the values generated by the plotly.express graph.

This happens because the way Plotly Express handles outliers, calculates percentiles, and defines limits uses specific algorithms. However, these differences are normal and are due to the different approaches used both in the internal implementation of the libraries and in the "manual" calculation.

3. What is the covariance between the 2 variables x_1 and x_2?

Formula

$$\mathrm{Cov}(X,Y)=\frac{1}{n}\sum_{i=1}^{n}\left(X_i-\acute{X}\right)\cdot\left(Y_i-\acute{Y}\right)$$

Method

```
def calculateCovariance(columnX, columnY):
    columnLenght = len(columnX)

    if columnLenght != len(columnY):
        raise ValueError("Columns must have the same length")

    meanX = sum(columnX) / columnLenght
    meanY = sum(columnY) / columnLenght

    return sum((columnX[i] - meanX) * (columnY[i] - meanY) for i in
range(columnLenght)) / columnLenght
```

Immplementation

```
covPy = calculateCovariance(columnx_1, columnx_2)
covNp = np.cov(columnx_1, columnx_2, bias = True)[0, 1]

print(f"The covariance calculated by python is equal to: {covPy}")
print(f"The covariance calculated by NumPY is equal to: {covNp}")

if np.isclose(covPy, covNp):
    print("The two covariances are equal.")
else:
    print(f"Between the two covariances there is a difference of:
{abs(covariance_manual - covariance_numpy)}")
```

```
The covariance calculated by python is equal to: 113233718703629.03
The covariance calculated by NumPY is equal to: 113233718703629.06
The two covariances are equal.
```

4. What is the correlation between the variable x_1 and x_2 (Calculate by hand)?|

Formula

$$\mathrm{Cor}(X,Y)=\frac{\sum_{i=1}^{n}\left(X_i-\acute{X}\right)\left(Y_i-\acute{Y}\right)}{\sqrt{\sum_{i=1}^{n}\left(X_i-\acute{X}\right)^2}\sqrt{\sum_{i=1}^{n}\left(Y_i-\acute{Y}\right)^2}}$$

Method

```
def calculateCorrelation(columnX, columnY):
    columnLenght = len(columnX)

    if columnLenght != len(columnY):
        raise ValueError("Columns must have the same length")

    meanX = sum(columnX) / columnLenght
    meanY = sum(columnY) / columnLenght

    varX = sum((columnX[i] - meanX)**2 for i in
range(columnLenght))**0.5
    varY = sum((columnY[i] - meanY)**2 for i in
range(columnLenght))**0.5

    return calculate_covariance(columnX, columnY) / varX * varY
```

Immplementation

```
corPy = calculateCovariance(columnx_1, columnx_2)
corNp = np.corrcoef(columnx_1, columnx_2)[0, 1]
```

```
print(f"The correlation calculated by python is equal to: {covPy}")
print(f"The correlation calculated by NumPY is equal to: {covNp}")

if np.isclose(covPy, covNp):
    print("The two covariances are equal.")
else:
    print(f"Between the two covariances there is a difference of:
{abs(covariance_manual - covariance_numpy)}")
```

```
The correlation calculated by python is equal to: 113233718703629.03
The correlation calculated by NumPY is equal to: 113233718703629.06
The two covariances are equal.
```

5. What is the relationship between covariance and correlation?

Correlation and covariance are two statistical functions that measure the linear relationship between two forces. Both measure the strength and direction of the linear relationship. They can take negative values for inverse relationships. They can be zero for weak or non-existent relationships. Or, they can result in positive values for direct relationships.

Unlike the covariance, the Pearson correlation (used in this work) is normalized and its standard values are

$$[-1;1)$$

It is based on the covariance, which is divided by the product of the standard deviation of both variables.

6. Calculate the result of the K-means algorithm on this data set by hand (as we did in excel). We are going to create 4 groups, that is, k=4 (clusters)

```
import random
import math
from sklearn.cluster import KMeans
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

# Scaler
def scaleData(data):
    scaledData = data.copy()
    for column in data.columns:
        mean = calculateMean(data[column])
        standardDeviation = calculateStdDev(data[column], mean)
        scaledData[column] = (data[column] - mean) / standardDeviation
    return scaledData

# Plot design
def generateRandomColors(numberColors):
    return random.sample(list(plt.cm.colors.cnames.keys()),
```

```
numberColors)

def showCentroids(titles, clusters, finalCentroids):
    colors = generateRandomColors(len(clusters))
    for cluster_index, cluster_points in clusters.items():
        cluster_points = np.array(cluster_points)
        plt.scatter(cluster_points[:, 0], cluster_points[:, 1],
c=colors[cluster_index],
                    label=f'Cluster {cluster_index + 1}')

    finalCentroids = np.array(finalCentroids)

    plt.scatter(finalCentroids[:, 0], finalCentroids[:, 1], c='black',
marker='x', label='Centroids')
    plt.xlabel(titles[0])
    plt.ylabel(titles[1])
    plt.legend()
    plt.title(titles[2])
    plt.show()

# Calculate K-Means
def printCalculation(clusters, centroids, counter):
    #print(clusters)

print(f"({counter})------------------------------------------------")
    for idx, centroid in enumerate(centroids):
        print(f"\t* Centroid ({centroid[0]}, {centroid[1]}): {idx}")
        for point in clusters[idx]:
            print(f"\t\t- {point[0]}, {point[1]}")
    print("------------------------------------------------------")


def euclideanDistance(pointP, pointQ):
    return math.sqrt((pointP[0] - pointQ[0]) ** 2 + (pointP[1] -
pointQ[1]) ** 2)

def assignClusters(data, centroids):
    clusters = {}
    for index, row in data.iterrows():
        distances = [euclideanDistance(row, centroid) for centroid in
centroids]
        clusterAssignment = np.argmin(distances)
        if clusterAssignment not in clusters:
            clusters[clusterAssignment] = [row.values]
        else:
            clusters[clusterAssignment].append(row.values)
    return clusters

def updateCentroids(clusters):
    centroids = []
```

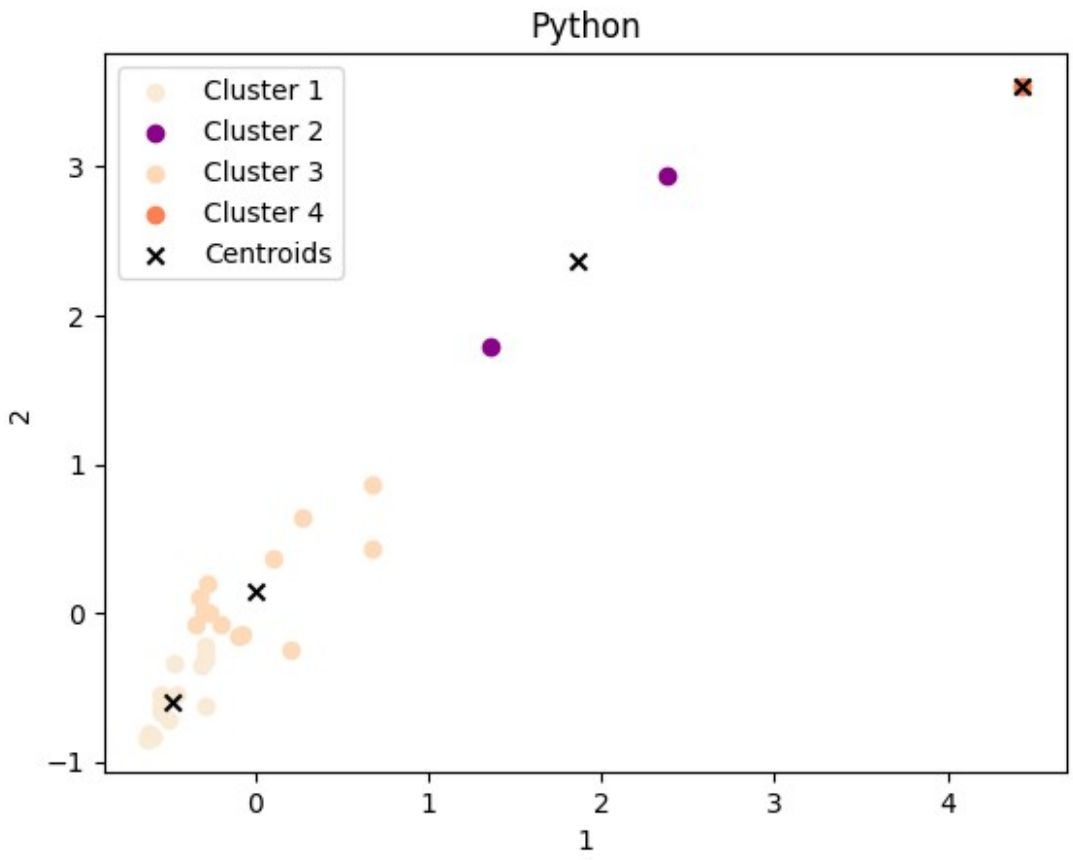
```
    for cluster in clusters.values():
        clusterMean = np.mean(cluster, axis=0)
        centroids.append(clusterMean)
    return centroids

def calculateK_meansPython(data, k, max_iterations=100):
    # Centroid initialization
    initial_centroids_indices = np.random.choice(len(data), k,
replace=False)
    centroids = data.iloc[initial_centroids_indices].values

    whileCounter = 1
    isCalculationFinished = False
    while not isCalculationFinished:
        clusters = assignClusters(data, centroids)

        new_centroids = updateCentroids(clusters)

        # Check convergence
        if np.all(np.isclose(centroids, new_centroids)):
            isCalculationFinished = True
        else:
            centroids = new_centroids
        printCalculation(clusters, centroids, whileCounter)
        whileCounter += 1
    return clusters, centroids

'''
columnx_1 = df['GDP Millions (x_1)']
columnx_2 = df['Population (x_2)']
columnx_3 = df['GDP per capita Millions (x_3)']
'''

k = 4
dfKMeans = pd.DataFrame({'columnx_1': columnx_1, 'columnx_2':
columnx_2})
dfKMeans = scaleData(dfKMeans)

clustersPy, centroidsPy = calculateK_meansPython(dfKMeans, k)
showCentroids(["1", "2", "Python"], clustersPy, centroidsPy)
```

```
(1)-------------------------------------------------
     * Centroid (-0.2989136232257357, -0.41768649826158977): 0
          - 0.2755192069432137, 0.6389597541151041
          - 0.100390716865566, 0.3618507214284563
          - -0.2629772738016568, 0.0012559590081381656
          - -0.2738331433799948, 0.19196352813445094
          - -0.3482678322008814, -0.07182613234421477
          - -0.3194064893140099, 0.10015631668514548
          - -0.3012503023548936, 0.017451767833767146
```

```
          - -0.46476325109827604, -0.3319516257881053
     * Centroid (1.8684017928650634, 2.3692140809293694): 1
          - 4.433884493576995, 3.5428145108234967
     * Centroid (-0.1993235460426166, 0.11348253613409279): 2
          - -0.6128015374127548, -0.825116118242023
          - -0.5068175473617147, -0.7146165343699464
          - -0.0773698801955662, -0.14587755485233858
          - -0.2885519582423287, -0.2763448842034892
          - -0.5505521101212472, -0.6314523272555121
          - -0.29269560572403025, -0.6203449115854225
          - -0.09627409354360784, -0.15588167160864233
          - -0.5428943499516939, -0.5522761914441204
          - 0.6748039119953922, 0.8673259577255674
          - -0.6208797266760304, -0.8420704822841459
          - -0.6120080731798277, -0.8214376613130595
          - -0.28772305030799933, -0.2155348348975698
          - -0.3125043122277849, -0.3464219298905008
          - 0.200068151261657, -0.25165492410745016
          - -0.5483532807400774, -0.6634063168972544
          - -0.4587363514537466, -0.549252687998786
          - -0.29091455471999345, -0.30900562819529287
          - -0.5978175825851814, -0.8342312844973648
          - 0.6794899406883126, 0.43016058353418535
          - -0.19666832796389713, -0.07870850263263965
          - -0.6225211795543907, -0.8464271137747943
          - -0.6143781929496754, -0.8065279429643747
     * Centroid (4.433884493576995, 3.5428145108234967): 3
          - 2.3830850465656157, 2.944148920402107
          - 1.353718539164511, 1.794279241456631
-------------------------------------------------
(2)-------------------------------------------------
     * Centroid (-0.4458246951691719, -0.5798929666505118): 0
          - -0.6128015374127548, -0.825116118242023
          - -0.5068175473617147, -0.7146165343699464
          - -0.2885519582423287, -0.2763448842034892
          - -0.5505521101212472, -0.6314523272555121
          - -0.29269560572403025, -0.6203449115854225
          - -0.5428943499516939, -0.5522761914441204
          - -0.6208797266760304, -0.8420704822841459
          - -0.6120080731798277, -0.8214376613130595
          - -0.28772305030799933, -0.2155348348975698
          - -0.3125043122277849, -0.3464219298905008
          - 0.200068151261657, -0.25165492410745016
          - -0.5483532807400774, -0.6634063168972544
          - -0.4587363514537466, -0.549252687998786
          - -0.29091455471999345, -0.30900562819529287
          - -0.5978175825851814, -0.8342312844973648
          - -0.46476325109827604, -0.3319516257881053
          - -0.6225211795543907, -0.8464271137747943
```

```
          - -0.6143781929496754, -0.8065279429643747
     * Centroid (1.8684017928650634, 2.3692140809293694): 1
          - 2.3830850465656157, 2.944148920402107
          - 1.353718539164511, 1.794279241456631
     * Centroid (-0.012153630521835261, 0.17973589391891498): 2
          - 0.2755192069432137, 0.6389597541151041
          - 0.100390716865566, 0.3618507214284563
          - -0.0773698801955662, -0.14587755485233858
          - -0.2629772738016568, 0.0012559590081381656
          - -0.09627409354360784, -0.15588167160864233
          - -0.2738331433799948, 0.19196352813445094
          - 0.6748039119953922, 0.8673259577255674
          - -0.3482678322008814, -0.07182613234421477
          - -0.3194064893140099, 0.10015631668514548
          - -0.3012503023548936, 0.017451767833767146
          - 0.6794899406883126, 0.43016058353418535
          - -0.19666832796389713, -0.07870850263263965
     * Centroid (4.433884493576995, 3.5428145108234967): 3
          - 4.433884493576995, 3.5428145108234967
-------------------------------------------------------
(3)-------------------------------------------------------
     * Centroid (-0.4838183920180442, -0.5992010868001036): 0
          - -0.6128015374127548, -0.825116118242023
          - -0.5068175473617147, -0.7146165343699464
          - -0.2885519582423287, -0.2763448842034892
          - -0.5505521101212472, -0.6314523272555121
          - -0.29269560572403025, -0.6203449115854225
          - -0.5428943499516939, -0.5522761914441204
          - -0.6208797266760304, -0.8420704822841459
          - -0.6120080731798277, -0.8214376613130595
          - -0.28772305030799933, -0.2155348348975698
          - -0.3125043122277849, -0.3464219298905008
          - -0.5483532807400774, -0.6634063168972544
          - -0.4587363514537466, -0.549252687998786
          - -0.29091455471999345, -0.30900562819529287
          - -0.5978175825851814, -0.8342312844973648
          - -0.46476325109827604, -0.3319516257881053
          - -0.6225211795543907, -0.8464271137747943
          - -0.6143781929496754, -0.8065279429643747
     * Centroid (1.8684017928650634, 2.3692140809293694): 1
          - 2.3830850465656157, 2.944148920402107
          - 1.353718539164511, 1.794279241456631
     * Centroid (0.004171121923048759, 0.1465519848399638): 2
          - 0.2755192069432137, 0.6389597541151041
          - 0.100390716865566, 0.3618507214284563
          - -0.0773698801955662, -0.14587755485233858
          - -0.2629772738016568, 0.0012559590081381656
          - -0.09627409354360784, -0.15588167160864233
          - -0.2738331433799948, 0.19196352813445094
```

```
          - 0.6748039119953922, 0.8673259577255674
          - -0.3482678322008814, -0.07182613234421477
          - 0.200068151261657, -0.25165492410745016
          - -0.3194064893140099, 0.10015631668514548
          - -0.3012503023548936, 0.017451767833767146
          - 0.6794899406883126, 0.43016058353418535
          - -0.19666832796389713, -0.07870850263263965
     * Centroid (4.433884493576995, 3.5428145108234967): 3
          - 4.433884493576995, 3.5428145108234967
--------------------------------------------------
(4)--------------------------------------------------
     * Centroid (-0.4838183920180442, -0.5992010868001036): 0
          - -0.6128015374127548, -0.825116118242023
          - -0.5068175473617147, -0.7146165343699464
          - -0.2885519582423287, -0.2763448842034892
          - -0.5505521101212472, -0.6314523272555121
          - -0.29269560572403025, -0.6203449115854225
          - -0.5428943499516939, -0.5522761914441204
          - -0.6208797266760304, -0.8420704822841459
          - -0.6120080731798277, -0.8214376613130595
          - -0.28772305030799933, -0.2155348348975698
          - -0.3125043122277849, -0.3464219298905008
          - -0.5483532807400774, -0.6634063168972544
          - -0.4587363514537466, -0.549252687998786
          - -0.29091455471999345, -0.30900562819529287
          - -0.5978175825851814, -0.8342312844973648
          - -0.46476325109827604, -0.3319516257881053
          - -0.6225211795543907, -0.8464271137747943
          - -0.6143781929496754, -0.8065279429643747
     * Centroid (1.8684017928650634, 2.3692140809293694): 1
          - 2.3830850465656157, 2.944148920402107
          - 1.353718539164511, 1.794279241456631
     * Centroid (0.004171121923048759, 0.1465519848399638): 2
          - 0.2755192069432137, 0.6389597541151041
          - 0.100390716865566, 0.3618507214284563
          - -0.0773698801955662, -0.14587755485233858
          - -0.2629772738016568, 0.0012559590081381656
          - -0.09627409354360784, -0.15588167160864233
          - -0.2738331433799948, 0.19196352813445094
          - 0.6748039119953922, 0.8673259577255674
          - -0.3482678322008814, -0.07182613234421477
          - 0.200068151261657, -0.25165492410745016
          - -0.3194064893140099, 0.10015631668514548
          - -0.3012503023548936, 0.017451767833767146
          - 0.6794899406883126, 0.43016058353418535
          - -0.19666832796389713, -0.07870850263263965
     * Centroid (4.433884493576995, 3.5428145108234967): 3
          - 4.433884493576995, 3.5428145108234967
--------------------------------------------------
```



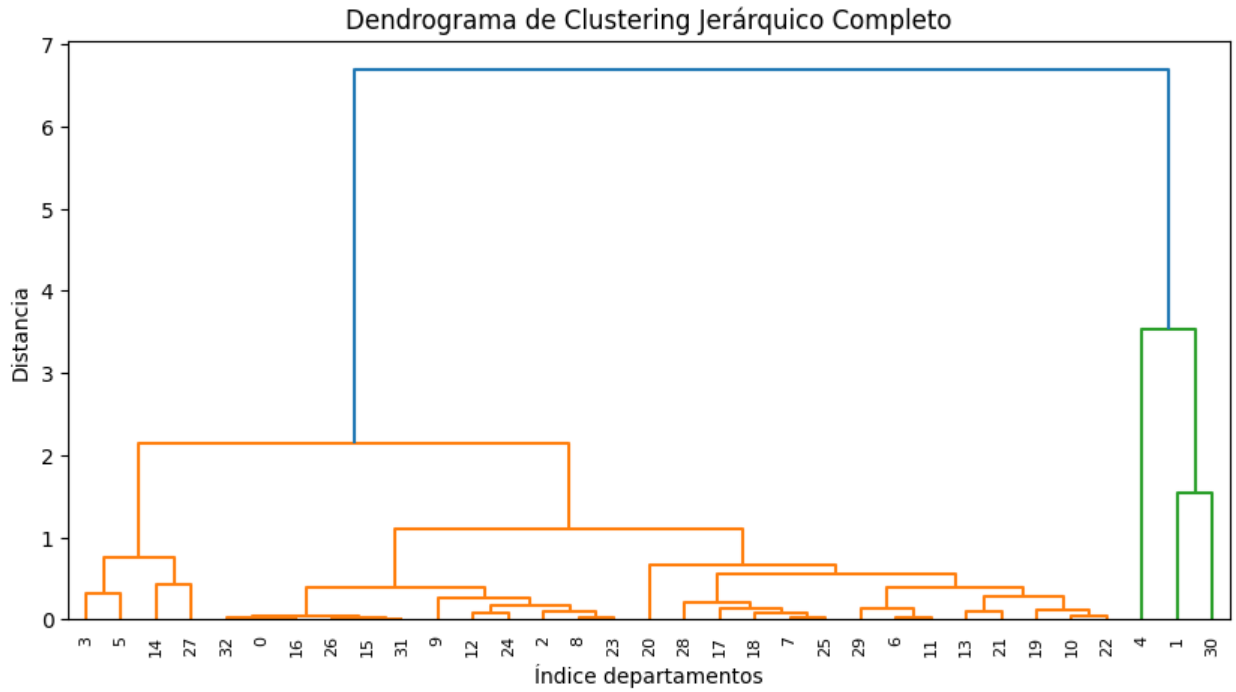
7. Calculate the result of a dedogram using the maximum distance in an excel

```
from scipy.cluster.hierarchy import dendrogram, linkage

def npEuclideanDistance(pointP, pointQ):
    return np.sqrt(np.sum((pointP - pointQ) ** 2))

def hierarchicalClustering(data):
    dataArray = data.to_numpy()

    clusters = [[point] for point in dataArray]
    distances = []
    labels = list(range(len(dataArray)))

    children = []
    parents = {}

    while len(clusters) > 1:
        maxDistance = np.inf
        mergeIndices = None

        for i in range(len(clusters)):
```

```
            for j in range(i + 1, len(clusters)):
                for pointP in clusters[i]:
                    for pointQ in clusters[j]:
                        distance = npEuclideanDistance(pointP, pointQ)
                        if distance < maxDistance:
                            maxDistance = distance
                            mergeIndices = (i, j)

        i, j = mergeIndices
        mergedCluster = clusters[i] + clusters[j]

        children.append([labels[i], labels[j]])
        parents[len(dataArray) + len(children) - 1] = [labels[i],
labels[j]]

        clusters[i] = mergedCluster
        del clusters[j]

        distances.append(maxDistance)

        labels[i] = len(dataArray) + len(children) - 1
        labels = np.delete(labels, j)

    index = list(range(len(dataArray)))
    labels = []
    for son in children:
        if son[0] in index:
            labels += [son[0]]
            index.remove(labels[-1])
        if son[1] in index:
            labels += [son[1]]
            index.remove(labels[-1])

    return children, distances, labels, parents

# Scipy
def hierarchicalClusteringScipy(data):
   model = linkage(data, method='complete', metric='euclidean')
   plotDendogram(model)

# Graph dendogram
def plotDendogram(Z):
    plt.figure(figsize=(10, 5))
    dendrogram(Z, leaf_rotation=90., leaf_font_size=8.,
labels=data.index)
    plt.title('Dendrograma de Clustering Jerárquico Completo')
    plt.xlabel('Índice departamentos')
    plt.ylabel('Distancia')
    plt.show()
```

```
data = scaleData(pd.DataFrame({'columnx_1': columnx_1, 'columnx_2':
columnx_2}))
column_department = df['Department']

# Ejecutar el algoritmo de clusterización jerárquica
children, distances, labels, children_info =
hierarchicalClustering(data)
print("Children:", children, "\n")
print("Parents:", children_info, "\n")
print("Distances between merged clusters:", distances, "\n")
print("Labels:", labels)

hierarchicalClusteringScipy(data)
print("Department:\n", column_department)
```

```
Children: [[0, 16], [15, 31], [33, 32], [35, 26], [36, 34], [6, 11],
[8, 23], [7, 25], [10, 22], [40, 18], [42, 17], [2, 39], [44, 12],
[45, 24], [41, 21], [47, 19], [48, 13], [49, 29], [38, 50], [37, 46],
[43, 28], [51, 53], [52, 9], [55, 54], [56, 20], [3, 5], [57, 58],
[14, 27], [59, 60], [61, 30], [62, 1], [63, 4]]

Parents: {33: [0, 16], 34: [15, 31], 35: [33, 32], 36: [35, 26], 37:
[36, 34], 38: [6, 11], 39: [8, 23], 40: [7, 25], 41: [10, 22], 42:
[40, 18], 43: [42, 17], 44: [2, 39], 45: [44, 12], 46: [45, 24], 47:
[41, 21], 48: [47, 19], 49: [48, 13], 50: [49, 29], 51: [38, 50], 52:
[37, 46], 53: [43, 28], 54: [51, 53], 55: [52, 9], 56: [55, 54], 57:
[56, 20], 58: [3, 5], 59: [57, 58], 60: [14, 27], 61: [59, 60], 62:
[61, 30], 63: [62, 1], 64: [63, 4]}

Distances between merged clusters: [0.0037630613690417232,
0.00465559937034815, 0.015096925811586002, 0.017538676065692156,
0.01878051122430197, 0.02138811900055807, 0.03202955361337503,
0.03274608465472497, 0.041558740815299006, 0.04319834790581762,
0.06081569850747371, 0.06593711792972792, 0.0795455955590147,
0.084212292952184, 0.08467401918891303, 0.10090189082969685,
0.10249631180544445, 0.10388065952627742, 0.12662819715464113,
0.14148144581220123, 0.15294500373266123, 0.15594185392006002,
0.18062013591228587, 0.21738462493289484, 0.2969186978709684,
0.32781001210061655, 0.41098072463608776, 0.43719048852508446,
0.45474107727126784, 1.1489854051309727, 1.5433067372098523,
2.1363938918225664]

Labels: [0, 16, 15, 31, 32, 26, 6, 11, 8, 23, 7, 25, 10, 22, 18, 17,
2, 12, 24, 21, 19, 13, 29, 28, 9, 20, 3, 5, 14, 27, 30, 1, 4]
```




```
Department:
 0                                                 Amazonas
1                                                 Antioquia
2                                                    Arauca
3                                                 Atlántico
4                                               Bogotá D.C.
5                                                    Bolívar
6                                                     Boyacá
7                                                     Caldas
8                                                   Caquetá
9                                                  Casanare
10                                                    Cauca
11                                                    Cesar
12                                                    Chocó
13                                                  Córdoba
14                                              Cundinamarca
15                                                   Guainía
16                                                 Guaviare
17                                                    Huila
18                                               La Guajira
19                                                Magdalena
20                                                     Meta
21                                                   Nariño
22                                       Norte de Santander
23                                                 Putumayo
24                                                  Quindío
25                                                Risaralda
26    San Andrés Providencia y Santa Catalina (Archi...
```

```
27                                                Santander
28                                                    Sucre
29                                                   Tolima
30                                          Valle del Cauca
31                                                   Vaupés
32                                                  Vichada
Name: Department, dtype: object
```

Observation

A great similarity can be observed between the dendrogram graph and the hierarchical clustering calculated without machine learning libraries. The design of the graph was not possible to develop, and given the short time given to deliver the assignment, the decision made was to use the library to show and compare the results.