# UNA SERIE DE NOTAS

El objetivo del código es analizar las calificaciones de varios alumnos, sacando información sobre su avance particular y en grupo.

Para hacer los cálculos necesarios, vamos a necesitar los siguientes módulos:

* **collections**: del cual usaremos la clase **Counter**. Esta clase sirve para hacer cuentas sobre listas, palabras… Por ejemplo, puede contar el numero de letras que tiene una palabra, o el numero de palabras que tiene una frase.
* **math**: Este módulo proporciona acceso a las funciones matemáticas definidas en el estándar de C. En este caso usaremos ‘\*’.
* **matplotlib.pyplot**: este módulo se especializa en el diseño y representación de gráficas de dos dimensiones, perfecto para cálculos estadísticos.
* **pandas**: este módulo nos sirve pare leer y escribir en archivos con diferentes formatos (Excel, csv, etc.). Lo usaremos para obtener la información de nuestra tabla Excel.
* **numpy**: esta biblioteca permite representar vectores y matrices con el lenguaje de Python.

El código comienza creando la clase **JMPEstadisticas**. Esta clase va a contener todos los cálculos estadísticos que son útiles para sacar información. Los tipos de cálculos van a estar separados en funciones:

Media:

 def calculoMediaAritmetica(self):

        n = self.caracteristica.count()

        sumaValoresObservaciones = 0

        mediaAritmetica = 0

        for valorObservacion in self.caracteristica:

            sumaValoresObservaciones = sumaValoresObservaciones + valorObservacion

        mediaAritmetica = sumaValoresObservaciones / n

        return mediaAritmetica

la “n” la definimos como el número de elementos que hay en una característica del Excel. Esta variable puede tomar el valor de cualquiera de las características en nuestro archivo. En este caso, tomará el valor de las notas de los alumnos. Es decir, “n” es el numero de notas que hemos recopilado.

“sumaValoresObservaciones” de momento tiene valor nulo, pero su valor final es la suma de todas las notas de los alumnos. Para ir sumando estos valores hacemos el bucle “for”, que recorre toda la lista de notas (la característica) y suma cada valor.

Por último, “mediaAritmetica” es el valor final, que obtenemos de dividir “sumaValoresObservaciones” entre “n”. la función devolverá este valor.

Mediana:

def calculoMediana(self):

        mediana = 0

        caracteristica = self.caracteristica.sort\_values()

        caracteristica = caracteristica.reset\_index(drop=True)

        n = self.caracteristica.count()

        par = False

        if (n % 2 == 0):

            print("La cantidad de observaciones es par.")

            par = True

        if par:

            rango = (n / 2);

            print("RANGO = "+str(rango))

            rangoPython = rango-1

            valor1 = caracteristica[rangoPython]

            valor2 = caracteristica[rangoPython+1]

            mediana = valor1 +((valor2-valor1)/2)

        else:

            rango = ((n + 1) / 2)

            rangoPython = rango - 1

            mediana = caracteristica[rangoPython]

        return [mediana, rango]

En la línea 3 ordena la característica (las notas) de menor a mayor. En la 4, ponemos los índices por defecto acorde a la nueva lista ordenada.

“n” vuelve a tener el mismo valor que en la función anterior. Necesitamos saber si este numero es par o impar, ya que la mediana se calcula de maneras distintas dependiendo de ello. Para ello, hacemos el primer if, que devuelve true si “n” es par.

En el siguiente if/else hacemos el calculo de la mediana, haciendo un calculo distinto si el if anterior ha devuelto true o no. Empecemos por el caso fácil, que “n” sea impar: con “rango” dividimos el numero de notas en la mitad. Como ese numero es impar, le añadimos 1 antes de dividirlo, y en la siguiente línea encontramos el índice del número central, la mediana.

En el caso que “N” sea par, el calculo es parecido, solo que encontramos los dos valores centrales y hallamos su valor entremedio.

La función devuelve una lista con la mediana y el rango.

Moda:

def calculoModa(self):

        moda = Counter(self.caracteristica)

        return moda

En esta función usamos la clase Counter, que va a contar el numero de veces que se repite cada valor de nota, y nos va a devolver el que más aparezca, la moda.

Varianza y Desviación Tipica:

def calculoVarianzaDesviacionTipica(self):

        n = self.caracteristica.count()

        mediaAritmetica = self.caracteristica.mean()

        varianza = 0

        c3 = 0

        for valorObservacion in self.caracteristica:

            x = valorObservacion

            moy = mediaAritmetica

            c1 = valorObservacion - mediaAritmetica

            c2 = c1 \* c1

            c3 = c3 + c2

        varianza = c3 / (n - 1)

        desviacionTipica = sqrt(varianza)

        return ([varianza, desviacionTipica])

Para este cálculo vamos a necesitar la media, la cual hallamos en la tercera línea. Después, imitamos un sumatorio con un bucle “for”, que va a hacer los siguientes cáculos: c1, que es la resta del valor de la nota particular del alumno y la media de notas. C2, que es c1 elevado al cuadrado. C3, que va a ser el total de todos los c2 sumados.

Por último, calcula la varianza. “n” vuelve a ser el numero de notas.

Para hacer la desviación típica, hacemos la raíz cuadrada de la varianza.

La función devuelve una lista con estos dos valores.

Cuartiles:

 def calculoDelosCuartiles(self,mediana,rangoMediana):

        n = self.caracteristica.count()

        sort\_caracteristica = self.caracteristica.sort\_values()

        sort\_caracteristica = sort\_caracteristica.reset\_index(drop=True)

        q1 = 0

        q2 = mediana

        q3 = 0

        #Cálculo Q1

        restoDivision = rangoMediana%2

        if (restoDivision != 0):

            q1 = sort\_caracteristica[((rangoMediana/2)+1)-1]

        else:

            valorMin = sort\_caracteristica[((rangoMediana/2)-1)]

            valorMax = sort\_caracteristica[(rangoMediana/2)]

            q1 = (valorMin + ((valorMax - valorMin) / 2) + valorMax) / 2

        # Cálculo Q3

        nbdatos = len(sort\_caracteristica)+1

        nbDatosDesdeMediana = nbdatos - rangoMediana

        restoDivision = nbDatosDesdeMediana % 2

        if (restoDivision != 0):

            q3 = sort\_caracteristica[(rangoMediana+ceil(nbDatosDesdeMediana/2))-1]

        else:

            valorMinQ3 = sort\_caracteristica[(rangoMediana+(nbDatosDesdeMediana/2))-1]

            valorMaxQ3 = sort\_caracteristica[(rangoMediana+(nbDatosDesdeMediana/2))]

            q3 = (valorMin + ((valorMax - valorMin) / 2) + valorMax) / 2

        return ([q1, q2, q3])

Para estos cálculos volvemos a reordenar los valores, como para la mediana, valor que también necesitamos y denominamos como q2. Los valores que vamos a calcular son q1 y q3.

Comenzando con q1: primero dividimos el rango de la mediana entre dos con el porcentaje, que devuelve el resto. Si da un resto, es decir, el rango es impar, nos meteremos en el if. Si no, nos meteremos en el else.

En el if, tomamos como q1 el valor que se encuentra a la mitad del rango de valores. En el else, necesitamos un valor máximo y uno mínimo. Estos son el valor que se encuentran a la mitad del rango de valores (el max) y el anterior (el min). En este caso, q1, será el valor que se encuentre tras el cálculo de la línea 16.

Para q3:

Para este tenemos que hallar el numero de datos desde la mediana, que calcula en las primeras líneas de esta parte (**nbDatosDesdeMediana**). Despues, mira si este número de datos es par o impar. En el primer caso, necesitaremos, como antes, un valor máximo y uno mínimo, con los que, tras los cálculos de la línea 30, hayamos q3.

En el caso de que sea impar, simplemente tenemos que ver cual es el valor, en la lista de notas ordenadas (**sort\_caracteristica**) que está en el índice calculado en la línea 24.

La función devuelve una lista con los cuartiles y la mediana.

Criterio De Turkey:

def criterioDeTukey(self, primerCuartil, tercerCuartil):

        valoresAberrantesInferiores = []

        valoresAberrantesSuperiores = []

        caracteristica = self.caracteristica.sort\_values()

        intercuartil = tercerCuartil - primerCuartil

        print("Inter-cuartil = "+str(intercuartil))

        limiteInferior = primerCuartil - (1.5 \* intercuartil)

        limiteSuperior = tercerCuartil + (1.5 \* intercuartil)

        for valorObservacion in caracteristica:

            if valorObservacion < limiteInferior:

                valoresAberrantesInferiores.append(valorObservacion)

            if valorObservacion > limiteSuperior:

                valoresAberrantesSuperiores.append(valorObservacion)

        valoresAberrantes = valoresAberrantesInferiores + valoresAberrantesSuperiores

        return (valoresAberrantes)

Al principio de la función creamos dos listas: “valoresAberrantesInferiores” y “ValoresAberrantesSuperiores”, que van a almacenar las notas que estén muy alejadas del resto, ya sean muy pequeñas o grandes. También necesitaremos el intercuartil, que obtenemos al dividir el tercer cuartil por el primero. Este valor lo necesitamos para obtener los limites superior e inferior, y así poder clasificar notas como aberrantes.

Para añadir las notas aberrantes a sus respectivas hacemos un bucle “for”, que las va comparando con el límite. Si es menor que el limite inferior se añadirá a la lista “valoresAberrantesInferiores”, asimismo, si es mayor que el limite superior, se añadirá a “ValoresAberrantesSuperiores”.

Por último, unificamos estas dos listas en una, “ValoresAberrantes”, la cual devolvemos.

Visualización:

def visualizacion(self,media,mediana,cuartil\_1,cuartil\_2,cuartil\_3):

        plt.subplot(2, 2, 1)

        plt.hist(self.caracteristica)

        plt.title("Histograma y media")

        plt.axvline(media, color='red', linestyle='dashed', linewidth=1,label = str(media))

        plt.legend(loc='upper right')

        plt.subplot(2, 2, 2)

        plt.hist(self.caracteristica)

        plt.title("Histograma y mediana")

        plt.axvline(mediana, color='green', linestyle='dashed', linewidth=1,label = str(mediana))

        plt.legend(loc='upper right')

        plt.subplot(2, 2, 3)

        plt.hist(self.caracteristica)

        plt.title("Histograma y cuartiles")

        plt.axvline(cuartil\_1, color='orange', linestyle='dashed', linewidth=1,label = "Q1: "+str(cuartil\_1))

        plt.axvline(cuartil\_2, color='orange', linestyle='dashed', linewidth=1,label = "Q2: "+str(cuartil\_2))

        plt.axvline(cuartil\_3, color='orange', linestyle='dashed', linewidth=1,label = "Q3: "+str(cuartil\_3))

        plt.legend(loc='upper right')

        plt.subplot(2, 2, 4)

        plt.boxplot(self.caracteristica)

        plt.title("Diagrama de caja y bigotes")

        plt.show()

Con esta función vamos a crear y representar gráficas. Se puede observar que usamos completamente el módulo “**matplotlib.pyplot**” (escrito como **plt**) para ello.

La función “**subplots**” es para crear los ejes en los que se va a dibujar la gráfica.

La función **“hist”** es para dibujar un histograma con los datos del parámetro que se le introduzca.

**“title”** es el nombre de la gráfica, mientras que “**axvline”** es para elegir los colores, el estilo de línea y lo que quieres que represente esa línea. “**legend”** introduce la leyenda en la ubicación que le introduzcas entre paréntesis.

Por último, la función show se usa para mostrar todas las figuras.

Las graficas que hacemos en este caso son: ‘el histograma y la media’, ‘el histograma y los cuartiles’, ‘el histograma y la mediana’, ‘y el diagrama de cajas y bigotes’.

Análisis Carácteristica:

 def analisisCaracteristica(self):

        print("-----------------------------------------")

        print("      MEDIDA DE TENDENCIA CENTRAL        ")

        print("-----------------------------------------\n")

        print("-- CANTIDAD DE OBSERVACIONES --")

        # -Cantidad de observaciones

        n = self.caracteristica.count()

        print("Cantidad de observaciones = " + str(n))

        print ("\n-- MIN --")

        valoresOrdenados = self.caracteristica.sort\_values()

        valoresOrdenados = valoresOrdenados.reset\_index(drop=True)

        print("Valor mínimo: "+str(valoresOrdenados[0]))

        print ("\n-- MAX --")

        valoresOrdenados = self.caracteristica.sort\_values()

        valoresOrdenados = valoresOrdenados.reset\_index(drop=True)

        print("Valor máximo: " + str(valoresOrdenados[len(valoresOrdenados)-1]))

        # -Media artimética:

        print("\n-- MEDIA --")

        media = self.calculoMediaAritmetica()

        print("Media aritmética calculada = " + str(media))

        print("> Observaciones: Si todas las observaciones tuvieran el mismo valor (reparto equitativo), este sería " + str(media))

        # -Media aritmética:

        print("\n-- MEDIANA --")

        mediana = self.calculoMediana()

        print("Mediana calculada = " + str(mediana[0]))

        print("> Observaciones: El valor que se encuentra en el punto medio de las observaciones es:" + str(mediana[0]))

        print("El reparto es: " + str(mediana[1]) + " valores en cada lado de la mediana")

        # -Moda

        print("\n-- MODA --")

        moda = self.calculoModa()

        print(moda)

        print("> Observacions: La moda permite determinar los valores observados con más frecuencia")

        print("\n\n-----------------------------------------")

        print("      MEDIDA DE DISPERSION        ")

        print("-----------------------------------------\n")

        print("-- RANGO --")

        print ("Rango de la serie = "+str(valoresOrdenados[len(valoresOrdenados)-1]-valoresOrdenados[0]))

        varianzaDesviacionTipica = self.calculoVarianzaDesviacionTipica()

        print("\n-- VARIANZA --")

        print("Varianza calculada = " + str(varianzaDesviacionTipica[0]))

        print("\n-- DESVIACION TIPICA --")

        print("Desviación típica calculada = " + str(varianzaDesviacionTipica[1]))

        desviacionTipica = varianzaDesviacionTipica[1]

        print("68 % de los valores de las observaciones se sitúan entre " + str(media - desviacionTipica) + " y " + str(

            media + desviacionTipica))

        print("95 % de los valores de las observaciones se sitúan entre " + str(media - (desviacionTipica \* 2)) + " y " + str(

            media + (desviacionTipica \* 2)))

        print("99 % de los valores de las observaciones se sitúan entre " + str(media - (desviacionTipica \* 3)) + " y " + str(

            media + (desviacionTipica \* 3)))

        print("\n\n-----------------------------------------")

        print("      CUARTILES        ")

        print("-----------------------------------------\n")

        cuartiles = self.calculoDelosCuartiles(mediana[0],mediana[1])

        print("25 % de las observaciones tienen un valor inferior a " + str(cuartiles[0]))

        print("50 % de las observaciones tienen un valor inferior a " + str(cuartiles[1]))

        print("75 % de las observaciones tienen un valor inferior a " + str(cuartiles[2]))

        print("\n\n-----------------------------------------")

        print("      DETECCION VALORES ABERRANTES        ")

        print("-----------------------------------------\n")

        print("> Criterios de Tukey")

        valoresAberrantes = self.criterioDeTukey(cuartiles[0], cuartiles[2])

        print("Cantidad de valores aberrantes: " + str(len(valoresAberrantes)))

        print("Valores:" + str(valoresAberrantes))

        print("\n\n-----------------------------------------")

        print("      VISUALIZACION        ")

        print("-----------------------------------------\n")

        print("Generación de las gráficas...")

        self.visualizacion(media,mediana[0],cuartiles[0],cuartiles[1],cuartiles[2])

Esta función es simplemente una formalización y organización del resto de funciones, de manera que las llamamos de manera ordenada y por apartados. Lo más remarcable es al principio, que haya datos como el valor mínimo, el máximo, y la cantidad de observaciones (lo que lleva siendo “n” en todo nuestro código)

Así hemos acabado con la clase **JMPEstadisticas.** Ahora, pasamos al archivo main.

#--- CREACION DE UN DATAFRAME ----

observaciones = pnd.DataFrame({'NOTAS':np.array([3,19,10,15,14,12,9,8,11,12,11,12,13,11,14,16])})

Primero crea un dataframe, llamado observaciones, que va a contener todas las notas que vamos a usar en nuestra analítica. Para esto, usa **pandas.**

#--- ANALISIS DE UNA CARACTERISTICA ---

stats = jmp.JMPEstadisticas(observaciones['NOTAS'])

stats.analisisCaracteristica()

Aquí inicializa el código, en la clase **JMPEstadisticas**, con los datos de “observaciones”. Directamente llama a la función final, que devuelve, como he dicho antes, todos los datos ordenados y con sus gráficas.