

# CURSO DE ESTADÍSTICA - PARTE

# 1

## ▼ 1 CONOCIENDO LOS DATOS

### ▼ 1.1 Dataset del proyecto

#### Muestra de domicilios Colombia - 2018

Las investigaciones por muestras de domicilios realizadas anualmente, buscan encontrar características generales de la población, de educación, trabajo, rendimiento y otras, de acuerdo con las necesidades de información del país, tales como las características de migración, fertilidad, casamientos, salud, nutrición, entre otros temas. Estas muestras al pasar de los años consistuyen una herramienta importante para la formulación, validación y evaluación de políticas dirigidas al desarrollo socioeconómico y la mejora de las condiciones de vida en Colombia.

#### Datos

Los datos fueron creados de manera didáctica para este curso.

#### Variables utilizadas

##### Ingreso

Ingresos mensuales (en miles de pesos) del trabajo principal para personas de 10 años o más.

##### Edad

Edad del entrevistado en la fecha de referencia en años.

## Altura

---

Altura del entrevistado em metros.

## Ciudad

---

Código de referência a 27 ciudades analizadas.

## Sexo

---

Código	Descripción
0	Masculino
1	Femenino

## Años de Estudio

---

Código	Descripción
1	Sin estudios y menos de 1 año
2	1 año
3	2 años
4	3 años
5	4 años
6	5 años
7	6 años
8	7 años
9	8 años
10	9 años
11	10 años
12	11 años
13	12 años
14	13 años
15	14 años
16	15 años o más
17	No se sabe
	No aplica

## Color

---

Código	Descripción
0	Indio
2	Blanco
4	Negro
6	Amarillo
8	Moreno
9	Sin declarar

## Tratamiento a los datos

Algunos de los tratamientos de datos más frecuentes son:

1. Eliminar las observaciones (líneas) con entradas de datos inválidos;
2. Eliminar observaciones donde hay datos perdidos (missing data);
3. Filtros propios de la investigación, por ejemplo: considerar solo las encuestas realizadas a la cabeza de familia (responsable por el domicilio).

### ▼ Importando pandas y leyendo el dataset del proyecto

<https://pandas.pydata.org/>

```
import pandas as pd
```

```
datos= pd.read_csv('datos.csv')
```

```
type(datos)
```

```
pandas.core.frame.DataFrame
```

```
datos.head()
```

	Ciudad	Sexo	Edad	Color	Años de Estudio	Ingreso	Altura
0	11	0	23	8	12	800	1.603808
1	11	1	23	2	12	1150	1.739790
2	11	1	35	8	15	880	1.760444
3	11	0	46	2	6	3500	1.783158
4	11	1	47	8	9	150	1.690631



## ▼ 1.2 Tipos de datos

---

### ▼ Variables cualitativas ordinales

- Variables que pueden ser ordenadas o que responden algún tipo de jerarquía

```
sorted(datos['Años de Estudio'].unique())
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]
```

### ▼ Variables cualitativas nominales

- Variables que no pueden ser ordenadas ni responden alguna jerarquía.

```
sorted(datos['Ciudad'].unique())
```

```
[11,  
12,  
13,  
14,  
15,  
16,  
17,  
21,  
22,  
23,  
24,  
25,  
26,  
27,  
28,  
29,  
31,  
32,  
33,  
35,  
41,  
42,  
43,  
50,  
51,  
52,  
53]
```

```
sorted(datos['Sexo'].unique())
```

```
[0, 1]
```

```
sorted(datos['Color'].unique())
```

```
[0, 2, 4, 6, 8]
```

## ▼ Variables cuantitativas discretas

- Variables que representan un conteo donde los valores posibles forman un conjunto finito o numerable.

Edad mínima

```
datos.Edad.min()
```

```
13
```

Edad máxima

```
datos.Edad.max()
```

```
99
```

Imprimiendo ambos

```
print('Desde %s hasta %s años' %(datos.Edad.min(), datos.Edad.max()))
```

```
Desde 13 hasta 99 años
```

## Observación

La variable edad puede ser clasificada de tres formas diferentes:

1. **CUANTITATIVA DISCRETA** - cuando representa años completos (números enteros);
2. **CUANTITATIVA CONTINUA** - cuando representa la edad exacta, siendo representada por fracciones de años;
3. **CUALITATIVA ORDINAL** - cuando representa intervalos de edad.

## ▼ Variables cuantitativas continuas

- Variables que representan un conteo o medición que asume valores en escalas continuas (números reales).

```
print('Desde %s hasta %s metros' %(datos.Altura.min(), datos.Altura.max()))
```

Desde 1.339244614 hasta 2.028496765 metros

## Clasificación de una variable



## ▼ 2 DISTRIBUCIÓN DE FRECUENCIAS

El primer paso en un análisis es conocer el comportamiento de las variables involucradas en el estudio. Utilizando técnicas estadísticas como el análisis de las **DISTRIBUCIONES DE FRECUENCIAS** e **HISTOGRAMAS** podemos evaluar mejor cómo se distribuyen los fenómenos del estudio.

### ▼ 2.1 Distribución de frecuencias para variables cualitativas

#### ▼ Método 1

[https://pandas.pydata.org/pandas-docs/version/0.22/generated/pandas.Series.value\\_counts.html](https://pandas.pydata.org/pandas-docs/version/0.22/generated/pandas.Series.value_counts.html)

Trabajando con la variable 'Sexo'

```
datos['Sexo'].value_counts()
```

```
0    53250
1    23590
Name: Sexo, dtype: int64
```

## Visualizando el porcentaje

```
datos['Sexo'].value_counts(normalize=True)*100
```

```
0    69.299844
1    30.700156
Name: Sexo, dtype: float64
```

## Creando una Tabla de frecuencia

```
frecuencia=datos['Sexo'].value_counts()
```

```
porcentaje= datos['Sexo'].value_counts(normalize=True)*100
```

```
dist_frec_cualitativa= pd.DataFrame({'Frecuencia': frecuencia, 'Porcentaje (%)': porcentaje})
```

```
dist_frec_cualitativa
```

	Frecuencia	Porcentaje (%)
0	53250	69.299844
1	23590	30.700156



## Parámetro para mejorar la tabla

```
dist_frec_cualitativa.rename(index={0: 'Masculino', 1: 'Femenino'}, inplace=True)
```

Imprimiendo para verificar si se salvó el cambio:

```
dist_frec_cualitativa
```

	Frecuencia	Porcentaje (%)
<b>Masculino</b>	53250	69.299844
<b>Femenino</b>	23590	30.700156



Agregando nuevo encabezado: Sexo

'rename\_axis()' es un método de DataFrame utilizado para renombrar los nombres de los ejes. En este caso, se utiliza para renombrar el eje de las columnas del DataFrame. 'Sexo': es el nuevo nombre asignado al eje de las columnas del DataFrame. En otras palabras, todas las columnas del DataFrame se relacionan con una variable categórica "Sexo". axis='columns': indica que el eje que se va a renombrar es el eje de las columnas del DataFrame.

```
dist_frec_cualitativa.rename_axis('Sexo', axis= 'columns', inplace=True)
```

```
dist_frec_cualitativa
```

	Sexo	Frecuencia	Porcentaje (%)
	<b>Masculino</b>	53250	69.299844
	<b>Femenino</b>	23590	30.700156

## ▼ Método 2

<https://pandas.pydata.org/pandas-docs/version/0.22/generated/pandas.crosstab.html>

```
sexo = {0: 'Masculino',
        1: 'Femenino'}

color = {0: 'Indio',
         2: 'Blanco',
         4: 'Negro',
         6: 'Amarillo',
         8: 'Pardo',
         9: 'Sin declarar'}
```

Construyendo las frecuencias para cada una de las categorías o casillas de referencia para cada proceso y para cada color.


```
frecuencia= pd.crosstab(datos.Sexo, datos.Color)
frecuencia
```



Color 0 2 4 6 8 


Renombrando el nombre de las filas respectivamente

```
frecuencia.rename(index=sexo, inplace=True)
frecuencia
```

Color	0	2	4	6	8	
Sexo						
<b>Masculino</b>	256	22194	5502	235	25063	
<b>Femenino</b>	101	9621	2889	117	10862	

Renombrando las columnas

```
frecuencia.rename(columns=color, inplace=True)
frecuencia
```

Color	Indio	Blanco	Negro	Amarillo	Pardo	
Sexo						
<b>Masculino</b>	256	22194	5502	235	25063	
<b>Femenino</b>	101	9621	2889	117	10862	

Denotando el resultado en porcentajes

```
porcentaje= pd.crosstab(datos.Sexo, datos.Color, normalize=True)*100
porcentaje.rename(index=sexo, inplace=True)
porcentaje.rename(columns=color, inplace=True)
porcentaje
```

Color	Indio	Blanco	Negro	Amarillo	Pardo	
Sexo						
<b>Masculino</b>	0.333160	28.883394	7.160333	0.305830	32.617126	
<b>Femenino</b>	0.131442	12.520822	3.759761	0.152264	14.135867	

Especificando las medias de ingreso de cada casilla

```

porcentaje= pd.crosstab(datos.Sexo, datos.Color,
                        aggfunc='mean',
                        values= datos.Ingreso)
porcentaje.rename(index=sexo, inplace=True)
porcentaje.rename(columns=color, inplace=True)
porcentaje

```

	Color	Indio	Blanco	Negro	Amarillo	Pardo
Sexo						
Masculino	1081.710938	2925.744435	1603.861687	4758.251064	1659.577425	
Femenino	2464.386139	2109.866750	1134.596400	3027.341880	1176.758516	

## 2.2 Distribución de frecuencia para variables cuantitativas (clases personalizadas)

### ▼ Paso 1 - Especifique los límites de cada clase

Utilice la siguiente clasificación:

**A ► Más de 20 SM**

**B ► De 10 a 20 SM**

**C ► De 4 a 10 SM**

**D ► De 2 a 4 SM**

**E ► Hasta 2 SM**

donde **SM** es el valor del salario mínimo en ese momento. En nuestro caso **788 mil pesos colombianos** (2018):

**A ► Más de 15.760**

**B ► De 7.880 a 15.760**

**C ► De 3.152 a 7.880**

**D ► De 1.576 a 3.152**

**E ► Hasta 1.576**

```
datos.Ingreso.min()
```

0

```
datos.Ingreso.max()
```

```
200000
```

```
clases= [0,1576, 3152,7880,15760,200000]
```

```
clases
```

```
[0, 1576, 3152, 7880, 15760, 200000]
```

```
labels=['E', 'D', 'C', 'B', 'A']
```

```
labels
```

```
['E', 'D', 'C', 'B', 'A']
```

## ▼ Paso 2 - Crear la tabla de frecuencias

<https://pandas.pydata.org/pandas-docs/version/0.22/generated/pandas.cut.html>

- bins: El número de bins o los bordes de los intervalos a los que se quiere discretizar los valores de x.
- labels: Las etiquetas a utilizar para cada bin. Si se omite, se utilizarán etiquetas numéricas.
- right: Indica si los bordes de los bins son cerrados a la derecha (por defecto True).
- include\_lowest: Indica si el primer bin debe incluir el valor mínimo de x (por defecto False).

```
pd.cut(
    x=datos.Ingreso,
    bins=clases,
    labels=labels,
    include_lowest=True
)
```

```
0      E
1      E
2      E
3      C
4      E
```

```
..
```

```
76835  E
76836  E
76837  E
76838  E
76839  E
```

```
Name: Ingreso, Length: 76840, dtype: category
```

```
Categories (5, object): ['E' < 'D' < 'C' < 'B' < 'A']
```

```
datos.head()
```

	Ciudad	Sexo	Edad	Color	Años de Estudio	Ingreso	Altura
0	11	0	23	8	12	800	1.603808
1	11	1	23	2	12	1150	1.739790
2	11	1	35	8	15	880	1.760444
3	11	0	46	2	6	3500	1.783158
4	11	1	47	8	9	150	1.690631



## Análisis de frecuencia

```
frecuencia= pd.value_counts(
    pd.cut(
        x=datos.Ingreso,
        bins= clases,
        labels= labels,
        include_lowest=True
    )
)
frecuencia
```

```
E    49755
D    16700
C     7599
B     2178
A      608
Name: Ingreso, dtype: int64
```

## Obteniendo en porcentajes

```
porcentaje= pd.value_counts(
    pd.cut(
        x=datos.Ingreso,
        bins= clases,
        labels= labels,
        include_lowest=True
    ),
    normalize=True
)*100
porcentaje
```


```
E    64.751432
D    21.733472
```

```
C    9.889381
B    2.834461
A    0.791255
Name: Ingreso, dtype: float64
```

Tabla de distribución de frecuencias:


```
dist_frec_cuantitativa_personalizada= pd.DataFrame({'Frecuencia': frecuencia, 'Porcentaje (%)
```

```
dist_frec_cuantitativa_personalizada
```

	Frecuencia	Porcentaje (%)	
<b>E</b>	49755	64.751432	
<b>D</b>	16700	21.733472	
<b>C</b>	7599	9.889381	
<b>B</b>	2178	2.834461	
<b>A</b>	608	0.791255	

Ordenando de acuerdo al índice

```
dist_frec_cuantitativa_personalizada.sort_index(ascending=False)
```

	Frecuencia	Porcentaje (%)	
<b>A</b>	608	0.791255	
<b>B</b>	2178	2.834461	
<b>C</b>	7599	9.889381	
<b>D</b>	16700	21.733472	
<b>E</b>	49755	64.751432	

## 2.3 Distribución de frecuencia para variables cuantitativas (clases de amplitud fija)

### Importando bibliotecas

<http://www.numpy.org/>

```
import numpy as np
```

## ▼ Paso 1 - Definiendo el número de clases

### ▼ Regla de Sturges

$$k = 1 + \frac{10}{3} \log_{10} n$$

Imprimo cantidad de variables tiene:

```
n=datos.shape[0]  
n
```

```
76840
```

Calculando K:

```
k= 1+(10/3)* np.log10(n)  
k
```

```
17.285291187298853
```

Redondeando:

```
int(k)
```

```
17
```

## ▼ Paso 2 - Crear la tabla de frecuencias

```
frecuencia= pd.value_counts(  
    pd.cut(  
        datos.Ingreso,  
        bins= 17,  
        include_lowest=True  
    ),  
    sort=False
```

```
)
frecuencia


(-200.001, 11764.706]      75594
(11764.706, 23529.412]     1022
(23529.412, 35294.118]     169
(35294.118, 47058.824]      19
(47058.824, 58823.529]      16
(58823.529, 70588.235]       5
(70588.235, 82352.941]       4
(82352.941, 94117.647]       1
(94117.647, 105882.353]      6
(105882.353, 117647.059]     0
(117647.059, 129411.765]     1
(129411.765, 141176.471]     0
(141176.471, 152941.176]     0
(152941.176, 164705.882]     0
(164705.882, 176470.588]     0
(176470.588, 188235.294]     0
(188235.294, 200000.0]       3
Name: Ingreso, dtype: int64
```

```
porcentaje= pd.value_counts(
    pd.cut(
        datos.Ingreso,
        bins= 17,
        include_lowest=True
    ),
    sort=False,
    normalize=True
)*100
porcentaje

(-200.001, 11764.706]      98.378449
(11764.706, 23529.412]      1.330036
(23529.412, 35294.118]      0.219938
(35294.118, 47058.824]      0.024727
(47058.824, 58823.529]      0.020822
(58823.529, 70588.235]      0.006507
(70588.235, 82352.941]      0.005206
(82352.941, 94117.647]      0.001301
(94117.647, 105882.353]     0.007808
(105882.353, 117647.059]    0.000000
(117647.059, 129411.765]    0.001301
(129411.765, 141176.471]    0.000000
(141176.471, 152941.176]    0.000000
(152941.176, 164705.882]    0.000000
(164705.882, 176470.588]    0.000000
(176470.588, 188235.294]    0.000000
(188235.294, 200000.0]      0.003904
Name: Ingreso, dtype: float64
```

Obteniendo la tabla de frecuencias:

```
dist_frec_cuantitativa_amplitud_fija= pd.DataFrame({'Frecuencia': frecuencia, 'Porcentaje (%)': porcentaje})
dist_frec_cuantitativa_amplitud_fija
```

	Frecuencia	Porcentaje (%)	
(-200.001, 11764.706]	75594	98.378449	
(11764.706, 23529.412]	1022	1.330036	
(23529.412, 35294.118]	169	0.219938	
(35294.118, 47058.824]	19	0.024727	
(47058.824, 58823.529]	16	0.020822	
(58823.529, 70588.235]	5	0.006507	
(70588.235, 82352.941]	4	0.005206	
(82352.941, 94117.647]	1	0.001301	
(94117.647, 105882.353]	6	0.007808	
(105882.353, 117647.059]	0	0.000000	
(117647.059, 129411.765]	1	0.001301	
(129411.765, 141176.471]	0	0.000000	
(141176.471, 152941.176]	0	0.000000	
(152941.176, 164705.882]	0	0.000000	
(164705.882, 176470.588]	0	0.000000	
(176470.588, 188235.294]	0	0.000000	
(188235.294, 200000.0]	3	0.003904	

## ▼ 2.4 Histograma

El **HISTOGRAMA** es la representación gráfica de una distribución de frecuencia. Es un gráfico formado por un conjunto de rectángulos colocados uno al lado del otro, donde el área de cada rectángulo es proporcional a la frecuencia de la clase que representa.

### ▼ Importando la biblioteca

<https://seaborn.pydata.org/>

```
import seaborn as sns
```



```
ax= sns.distplot(datos.Altura, kde=False)
#tamaño de altura:12 ancho, 6 altura
ax.figure.set_size_inches(12, 6)
ax.set_title('Distribución de frecuencia - Altura', fontsize=18)
ax.set_xlabel('Altura (metros)', fontsize=14)
ax
```

<ipython-input-48-ed458bba9005>:1: UserWarning:

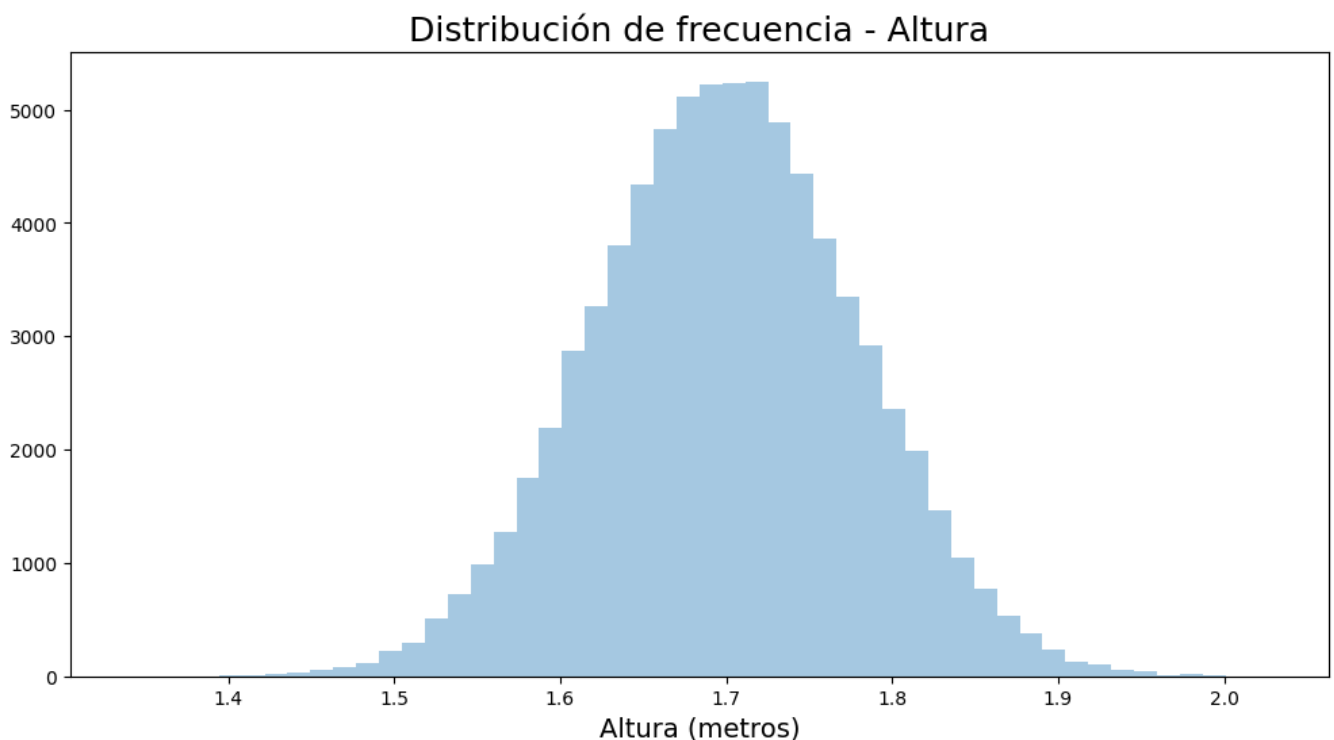
`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see

<https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751>

```
ax= sns.distplot(datos.Altura, kde=False)
<Axes: title={'center': 'Distribución de frecuencia - Altura'}, xlabel='Altura
(metros)'>
```



```
ax= sns.distplot(datos.Altura, kde=True)
```

```
ax.figure.set_size_inches(12, 6)
ax.set_title('Distribución de frecuencia - Altura - KDE', fontsize=18)
ax.set_xlabel('Altura (metros)', fontsize=14)
ax
```

<ipython-input-49-5fc447ed1765>:1: UserWarning:

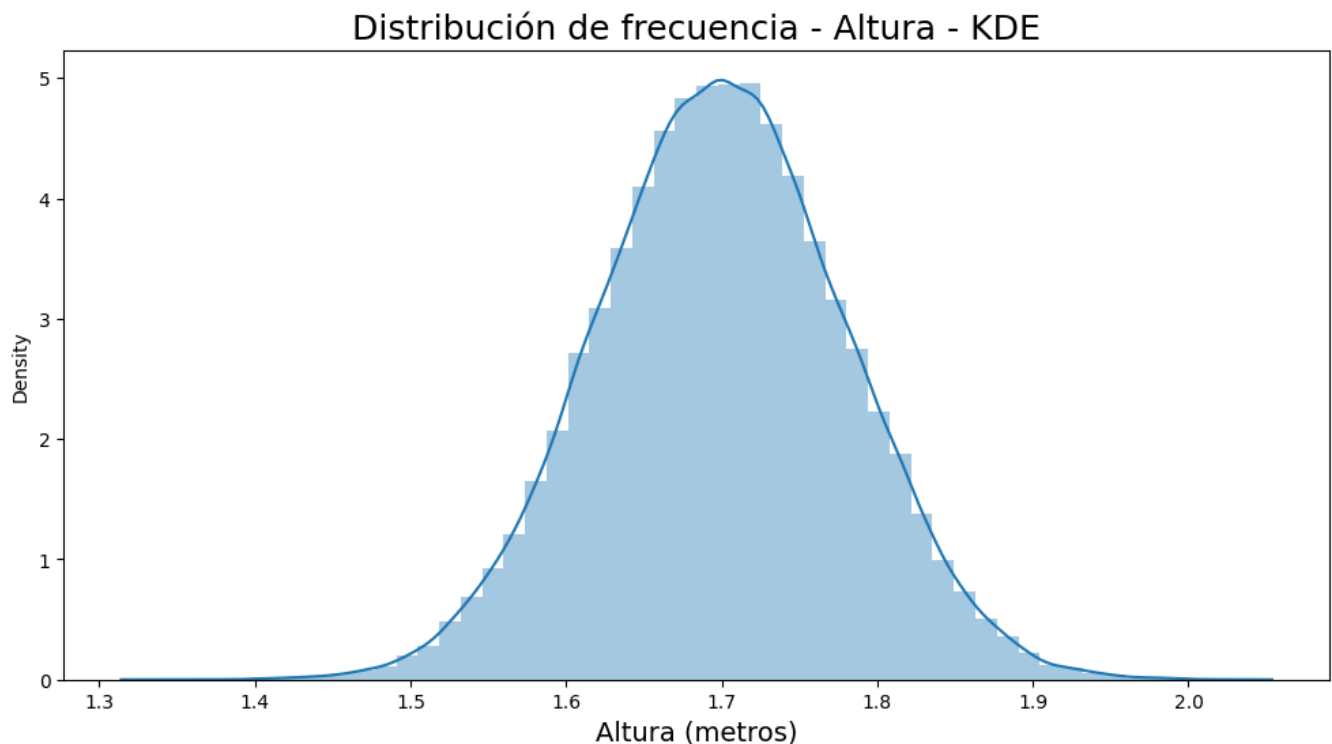
`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see

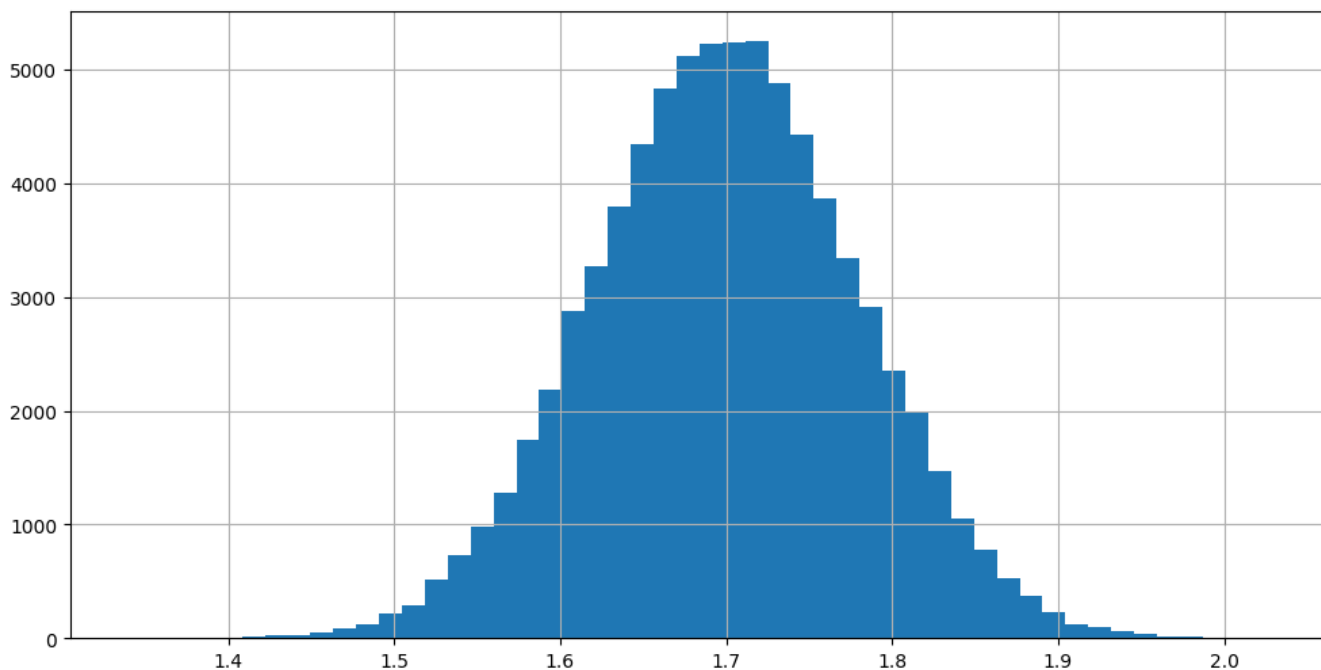
<https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751>

```
ax= sns.distplot(datos.Altura, kde=True)
<Axes: title={'center': 'Distribución de frecuencia - Altura - KDE'}, xlabel='Altura (metros)', ylabel='Density'>
```



```
datos.Altura.hist(bins=50, figsize=(12,6))
```

&lt;Axes: &gt;



dist\_frec\_cuantitativa\_personalizada

	Frecuencia	Porcentaje (%)	
<b>E</b>	49755	64.751432	
<b>D</b>	16700	21.733472	
<b>C</b>	7599	9.889381	
<b>B</b>	2178	2.834461	
<b>A</b>	608	0.791255	

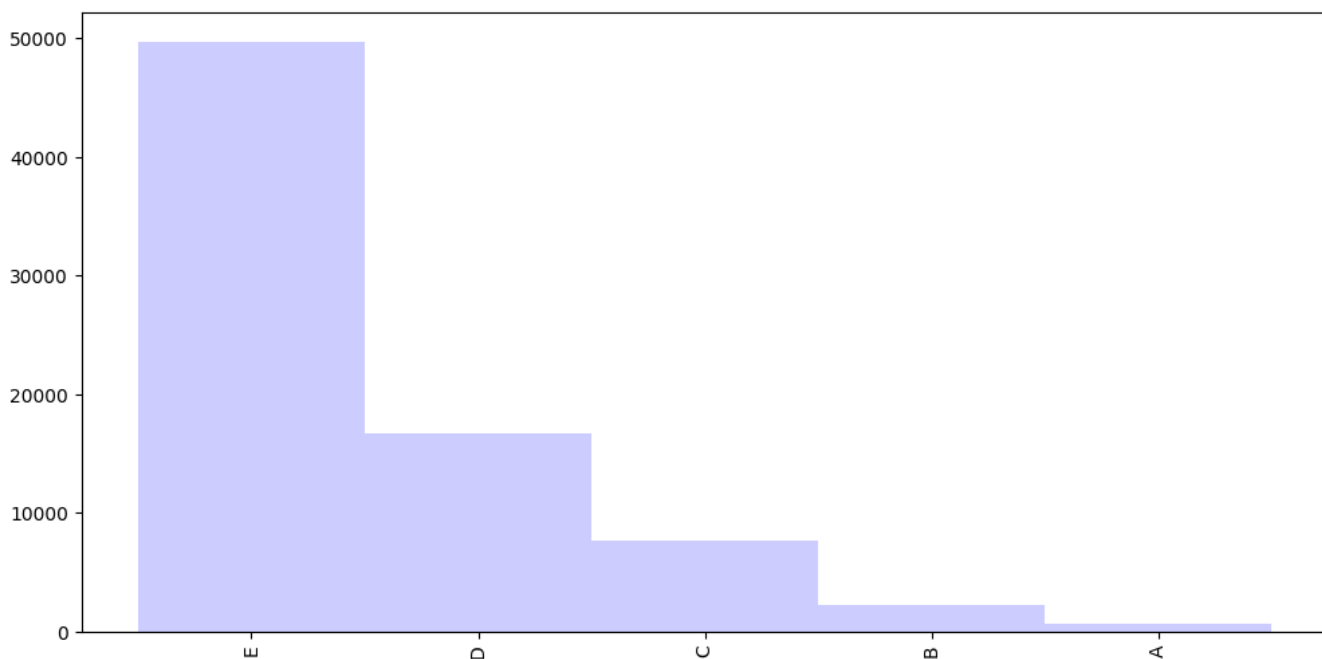
Graficando 'dist\_frec\_cuantitativa\_personalizada'

- width=1: Ancho de las barras en el gráfico. En este caso, se establece en 1.
- color='blue': Color de las barras. En este caso, se establece en azul.
- alpha=0.2: Transparencia de las barras. En este caso, se establece en 0.2.

- `figsize=(12,6)`: Tamaño de la figura que se generará. En este caso, se establece en 12 pulgadas de ancho y 6 pulgadas de alto.

```
dist_frec_cuantitativa_personalizada['Frecuencia'].plot.bar(width=1, color='blue', alpha=0.2,
dist_frec_cuantitativa_personalizada
```

	Frecuencia	Porcentaje (%)	✎
<b>E</b>	49755	64.751432	
<b>D</b>	16700	21.733472	
<b>C</b>	7599	9.889381	
<b>B</b>	2178	2.834461	
<b>A</b>	608	0.791255	



## ▼ 3 MEDIDAS DE TENDENCIA CENTRAL

## ▼ DataFrame de ejemplo

```
df = pd.DataFrame(data = {'María': [8, 10, 4, 8, 6, 10, 8],
                          'Pedro': [10, 2, 0.5, 1, 3, 9.5, 10],
                          'Pablo': [7.5, 8, 7, 8, 8, 8.5, 7]},
                  index = ['Matemática',
                          'Portugués',
                          'Inglés',
                          'Geografía',
                          'Historia',
                          'Física',
                          'Química'])
df.rename_axis('Asignaturas', axis = 'columns', inplace = True)
df
```

Asignaturas	María	Pedro	Pablo
Matemática	8	10.0	7.5
Portugués	10	2.0	8.0
Inglés	4	0.5	7.0
Geografía	8	1.0	8.0
Historia	6	3.0	8.0
Física	10	9.5	8.5
Química	8	10.0	7.0



## ▼ 3.1 Media aritmética

Es representada por  $\mu$  cuando se refiere a la población y por  $\bar{X}$  cuando se refiere a la muestra

$$\mu = \frac{1}{n} \sum_{i=1}^n X_i$$

donde

$n$  = número de observaciones (registros)

$X_i$  = valor de la  $i$ -ésima observación (registro)

La media de las notas de las asignaturas de María:

```
(8+10+4+8+6+10+8)/7
```

```
7.714285714285714
```

```
df["María"].mean()
```

```
7.714285714285714
```

Media de la variable 'Ingreso'

```
datos.Ingreso.mean()
```

```
2000.3831988547631
```

Existen muchos datos, se verifica si es la media real, entonces visualizamos:

La media del ingreso por sexo:

```
datos.groupby(['Sexo'])['Ingreso'].mean()
```

```
Sexo
```

```
0    2192.441596
```

```
1    1566.847393
```

```
Name: Ingreso, dtype: float64
```

Un ejemplo extra de la media para un grupo de ambos sexos:

```
dataset = pd.DataFrame({
    'Sexo': ['H', 'M', 'M', 'M', 'M', 'H', 'H', 'H', 'M', 'M'],
    'Edad': [53, 72, 54, 27, 30, 40, 58, 32, 44, 51]
})
dataset.groupby(['Sexo']).mean().loc['H']
```

```
Edad    45.75
```

```
Name: H, dtype: float64
```

## ▼ 3.2 Mediana

Para obtener la mediana de un conjunto de datos, debemos proceder de la siguiente manera:

1. Ordenar el conjunto de datos;
2. Identificar el número de observaciones (registros) del conjunto de datos ( $n$ );

## 3. Identificar el elemento del medio:

Cuando  $n$  sea impar, la posición del elemento del medio se obtendrá de la siguiente manera:

$$Elemento_{Md} = \frac{n + 1}{2}$$

Cuando  $n$  sea par, la posición del elemento mediano se obtendrá de la siguiente manera:

$$Elemento_{Md} = \frac{n}{2}$$

## 4. Obtener la mediana:

Cuando  $n$  sea impar:

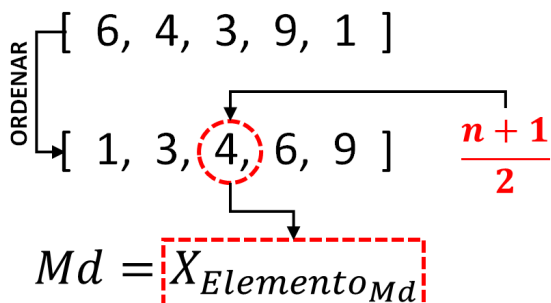
$$Md = X_{Elemento_{Md}}$$

Cuando  $n$  sea par:

$$Md = \frac{X_{Elemento_{Md}} + X_{Elemento_{Md}+1}}{2}$$

## ▼ Ejemplo 1 - n impar

**n impar**



```
notas_maria= df['María']
notas_maria
```

Matemática	8
Portugués	10
Inglés	4
Geografía	8
Historia	6
Física	10
Química	8

Name: María, dtype: int64

Ordenando:

```
notas_maria= notas_maria.sort_values()
notas_maria
```

Inglés	4
Historia	6
Matemática	8
Geografía	8
Química	8
Portugués	10
Física	10

Name: María, dtype: int64

Verificar cuántas notas tiene María

```
n= notas_maria.shape[0]
n
```

7

Visualizando el índice:

```
notas_maria=notas_maria.reset_index()
notas_maria
```



index María 

Verificando el índice anterior, se debe modificar el orden del índice dado que empieza por '0'

```
elemento_mediano= (n+1)/2
elemento_mediano
```

4.0

```
4    Química    8
```

```
notas_maria.loc[elemento_mediano -1]
```

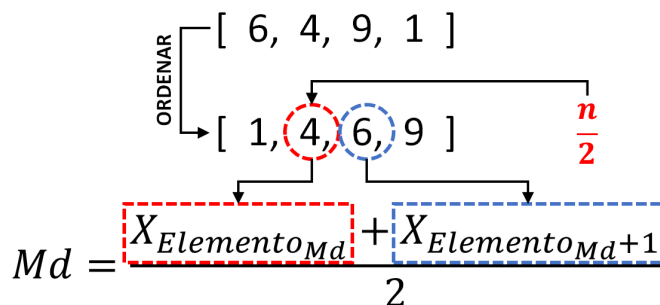
```
index    Geografía
María          8
Name: 3, dtype: object
```

```
notas_maria.median()
```

```
<ipython-input-65-d59c74e7d587>:1: FutureWarning: The default value of numeric_only in [
notas_maria.median()
María    8.0
dtype: float64
```

## ▼ Ejemplo 2 - n par

n par



```
notas_pedro= df.Pedro.sample(6,random_state= 101)
notas_pedro
```

```
Matemática    10.0
Inglés         0.5
Física         9.5
Historia       3.0
Química        10.0
Portugués      2.0
Name: Pedro, dtype: float64
```

```
n= notas_pedro.shape[0]
n
```

6

```
notas_pedro=notas_pedro.reset_index()
notas_pedro
```

	index	Pedro	
0	Matemática	10.0	
1	Inglés	0.5	
2	Física	9.5	
3	Historia	3.0	
4	Química	10.0	
5	Portugués	2.0	

```
elemento_mediano=n/2
elemento_mediano
```

3.0

```
notas_pedro.loc[elemento_mediano-1]
```

```
index    Física
Pedro      9.5
Name: 2, dtype: object
```

```
notas_pedro.median()
```

```
<ipython-input-71-b3e55fd8443a>:1: FutureWarning: The default value of numeric_only in [
    notas_pedro.median()
Pedro      6.25
dtype: float64
```

## ▼ Obtener la mediana en nuestro conjunto de datos

```
datos.Ingreso.median()
```

1200.0

Obteniendo la 'mitad' con quantile

```
datos.Ingreso.quantile()
```

```
1200.0
```

### ▼ 3.3 Moda

La moda se puede definir como el valor más frecuente de un conjunto de datos. La moda es ampliamente utilizada para datos cualitativos.

```
df
```

Asignaturas	María	Pedro	Pablo
Matemática	8	10.0	7.5
Portugués	10	2.0	8.0
Inglés	4	0.5	7.0
Geografía	8	1.0	8.0
Historia	6	3.0	8.0
Física	10	9.5	8.5
Química	8	10.0	7.0

Usando 'mode' para aplicar o buscar la moda

```
df.mode()
```

Asignaturas	María	Pedro	Pablo
0	8	10.0	8.0

ejemplificando una moda multimodal:

```
ejemplo= pd.Series([1,2,2,3,4,4,5,6,6])
ejemplo
```

```
0    1
1    2
```

```

2    2
3    3
4    4
5    4
6    5
7    6
8    6
dtype: int64

```

```
ejemplo.mode()
```

```

0    2
1    4
2    6
dtype: int64

```

## ▼ Obteniendo la moda de nuestro dataset

Moda de una variable cuantitativa

```
datos.Ingreso.mode()
```

```

0    788
Name: Ingreso, dtype: int64

```

Moda de la altura:

```
datos.Altura.mode()
```

```

0    1.568128
1    1.671225
2    1.681659
3    1.692977
4    1.708163
5    1.708370
6    1.753842
7    1.779073
8    1.796462
Name: Altura, dtype: float64

```

En general las variables continuas tendrás muchas modas, no es muy recomendable trabajar con ellas por éste caso

Para ver la moda de una lista de String:

```
import statistics as stat

datos_ejemplo= [" Hamburguesa Doble",
"Hamburguesa simple",
"Hamburguesa Doble",
"Hamburguesa Doble",
"Hamburguesa con queso",
"Hamburguesa Doble",
"Hamburguesa simple",
"Hamburguesa simple",
"Hamburguesa de pescado",
"Hamburguesa simple",
"Hamburguesa con queso"]
datos_ejemplo
```

```
[' Hamburguesa Doble',
'Hamburguesa simple',
'Hamburguesa Doble',
'Hamburguesa Doble',
'Hamburguesa con queso',
'Hamburguesa Doble',
'Hamburguesa simple',
'Hamburguesa simple',
'Hamburguesa de pescado',
'Hamburguesa simple',
'Hamburguesa con queso']
```

Verificando el valor de la moda:

```
stat.mode(datos_ejemplo)

'Hamburguesa simple'
```

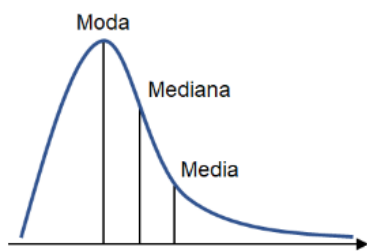
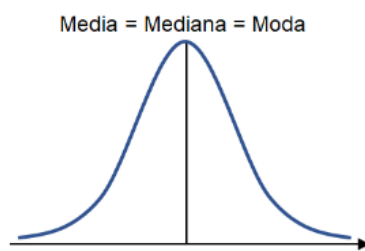
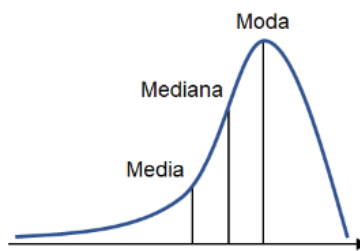
Se verifican cuantos datos hay

```
n= len(datos_ejemplo)
n
```

11

## ▼ 3.4 Relación entre media, mediana e moda

---

**Asimétrica a la derecha****Simétrica****Asimétrica a la izquierda**

## ▼ Evaluando la variable INGRESO

```
ax = sns.distplot(datos.query('Ingreso <20000').Ingreso)
ax.figure.set_size_inches(12,6)
ax
```

```
<ipython-input-84-6ec8fa2ffb45>:1: UserWarning:
```

```
`distplot` is a deprecated function and will be removed in seaborn v0.14.0.
```

```
Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
```

```
Moda = datos.Ingreso.mode()[0]
```

```
Moda
```

```
788
```

```
Mediana = datos.Ingreso.median()
```

```
Mediana
```

```
1200.0
```

```
Media = datos.Ingreso.mean()
```

```
Media
```

```
2000.3831988547631
```

```
Moda < Mediana < Media
```

```
True
```

```
0.0000
```

Tiene una simetría a la DERECHA

## ▼ Evaluando la variable ALTURA

```
ax = sns.distplot(datos.Altura)
```

```
ax.figure.set_size_inches(12,6)
```

```
ax
```

```
<ipython-input-89-ec2534fcef6f>:1: UserWarning:
```

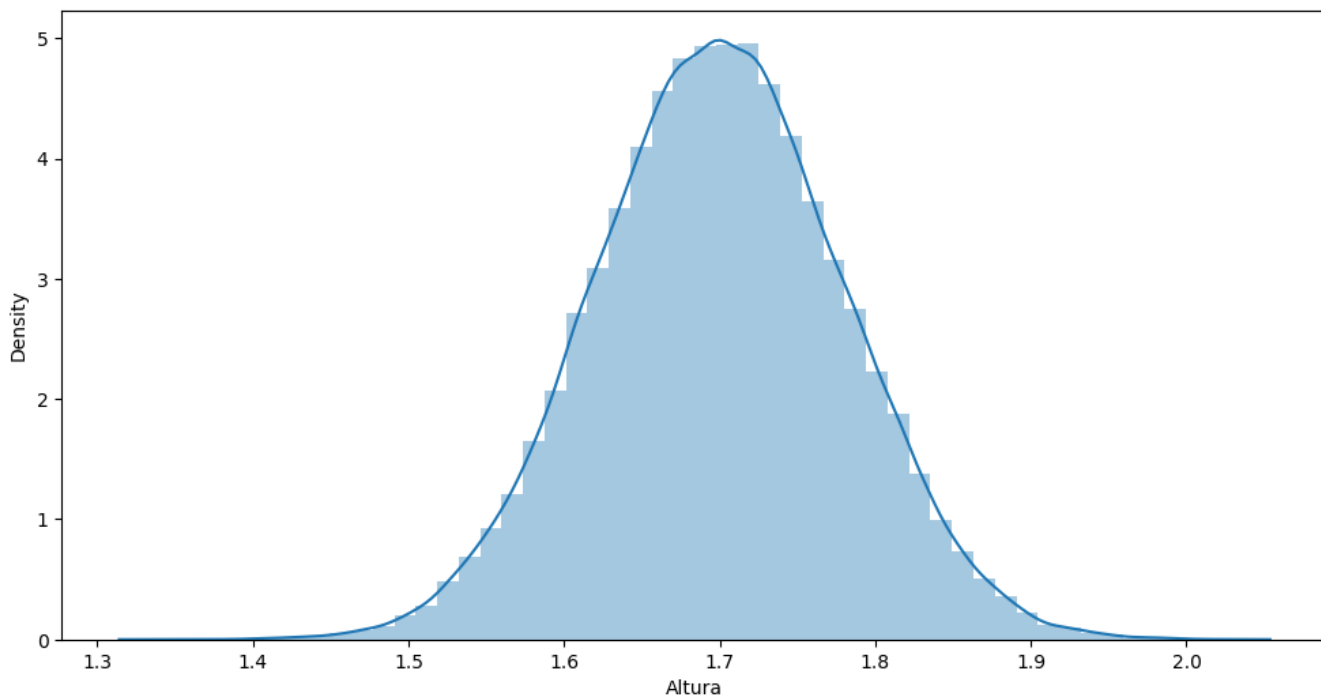
```
`distplot` is a deprecated function and will be removed in seaborn v0.14.0.
```

Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see

<https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751>

```
ax = sns.distplot(datos.Altura)
<Axes: xlabel='Altura', ylabel='Density'>
```



es MULTIMODAL:

```
Moda = datos.Altura.mode()
```

```
Moda
```

```
0    1.568128
1    1.671225
2    1.681659
3    1.692977
4    1.708163
5    1.708370
6    1.753842
7    1.779073
8    1.796462
Name: Altura, dtype: float64
```

```
Media = datos.Altura.mean()
```

```
Media
```

```
1.6995124540575741
```



```
Mediana = datos.Altura.median()  
Mediana
```

```
1.6993247325
```

Es SIMÉTRICA

---

## ▼ Evaluando la variable AÑOS DE ESTUDIO

Es discreta, niveles de enseñanza en un grupo de personas

```
ax = sns.distplot(datos["Años de Estudio"],bins = 17)  
ax.figure.set_size_inches(12,6)  
ax
```

```
<ipython-input-93-ffd656a109b5>:1: UserWarning:
```

```
`distplot` is a deprecated function and will be removed in seaborn v0.14.0.
```

Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see

<https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751>

```
ax = sns.distplot(datos["Años de Estudio"],bins = 17)
<Axes: xlabel='Años de Estudio', ylabel='Density'>
```



```
Moda = datos["Años de Estudio"].mode()[0]
Moda
```

```
12
```



```
Mediana = datos["Años de Estudio"].median()
Mediana
```

```
11.0
```



```
Media = datos["Años de Estudio"].mean()
Media
```

```
9.469664237376367
```

Años de Estudio

```
Moda > Mediana > Media
```

```
True
```

Es una variable SIMÉTRICA a la izquierda

## ▼ 4 MEDIDAS DE LOCALIZACIÓN

### ▼ 4.1 Cuartiles, deciles y percentiles

Hay una serie de medidas de posición similares en su diseño a la mediana, aunque no son medidas de tendencia central. Como se sabe, la mediana divide la distribución en dos partes iguales en términos del número de elementos en cada parte. Los cuartiles, por otro lado, permiten que la

distribución se divida en cuatro partes iguales en términos del número de elementos en cada uno; deciles en diez partes y centiles en cien partes iguales.

Por default con quantile obtenemos la mediana

```
datos.Ingreso.quantile()

1200.0
```

Ahora dividiendo en 3 partes:

25% de las observaciones ganan hasta un salario mínimo. Aquí por ejemplo como es 75, tiene 25% arriba, entonces tenemos que 25% ganan arriba de 2000.

```
datos.Ingreso.quantile([0.25, 0.5, 0.75])

0.25    788.0
0.50   1200.0
0.75   2000.0
Name: Ingreso, dtype: float64
```

Dividiendo en 10 partes iguales:

```
[i/10 for i in range(1,10)]

[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
```

Imprimiendo en 10 partes iguales:

```
datos.Ingreso.quantile([i/10 for i in range(1,10)])

0.1    350.0
0.2    788.0
0.3    800.0
0.4   1000.0
0.5   1200.0
0.6   1500.0
0.7   1900.0
0.8   2500.0
0.9   4000.0
Name: Ingreso, dtype: float64
```

Obteniendo percentiles:

4% de la muestra toda gana hasta 50 reales, hasta 50 pesos, entonces un 1% gana arriba de 15.000 o que un 2% gana arriba de 10.000.

```
datos.Ingreso.quantile([i/100 for i in range(1,100)] )
```

```
0.01      0.0
0.02      0.0
0.03      0.0
0.04      50.0
0.05     100.0
...
0.95     6000.0
0.96     7000.0
0.97     8000.0
0.98    10000.0
0.99    15000.0
Name: Ingreso, Length: 99, dtype: float64
```

Representación gráfica de X de la variable edad:

Para interpretar los cuartiles, los deciles, los percentiles, ya que la variable ingreso tiene sus valores extremos y queremos ver una representación más adecuada

- `sns.distplot()` es una función de Seaborn que traza la distribución de una variable.
- `datos.Edad` se refiere a la variable "Edad" dentro del conjunto de datos "datos".
- `'hist_kws={'cumulative': True}` es un argumento de palabras clave que se utiliza para configurar el histograma. Con `{'cumulative': True}`, se indica que se desea una distribución acumulada en el eje y.
- `kde_kws={'cumulative': True}` es otro argumento de palabras clave que se utiliza para configurar la estimación de densidad de kernel. Con `{'cumulative': True}`, se indica que se desea una distribución acumulada en el eje y.

```
ax= sns.distplot(datos.Edad,
                  hist_kws={'cumulative': True},
                  kde_kws={'cumulative': True},
                  bins=10)
ax.figure.set_size_inches(14, 6)
ax.set_title('Distribución de Frecuencia Acumulada', fontsize=18)
ax.set_ylabel('Acumulado', fontsize=14)
ax.set_xlabel('Años', fontsize=14)
ax
```

```
<ipython-input-103-e8f60731fce9>:1: UserWarning:
```

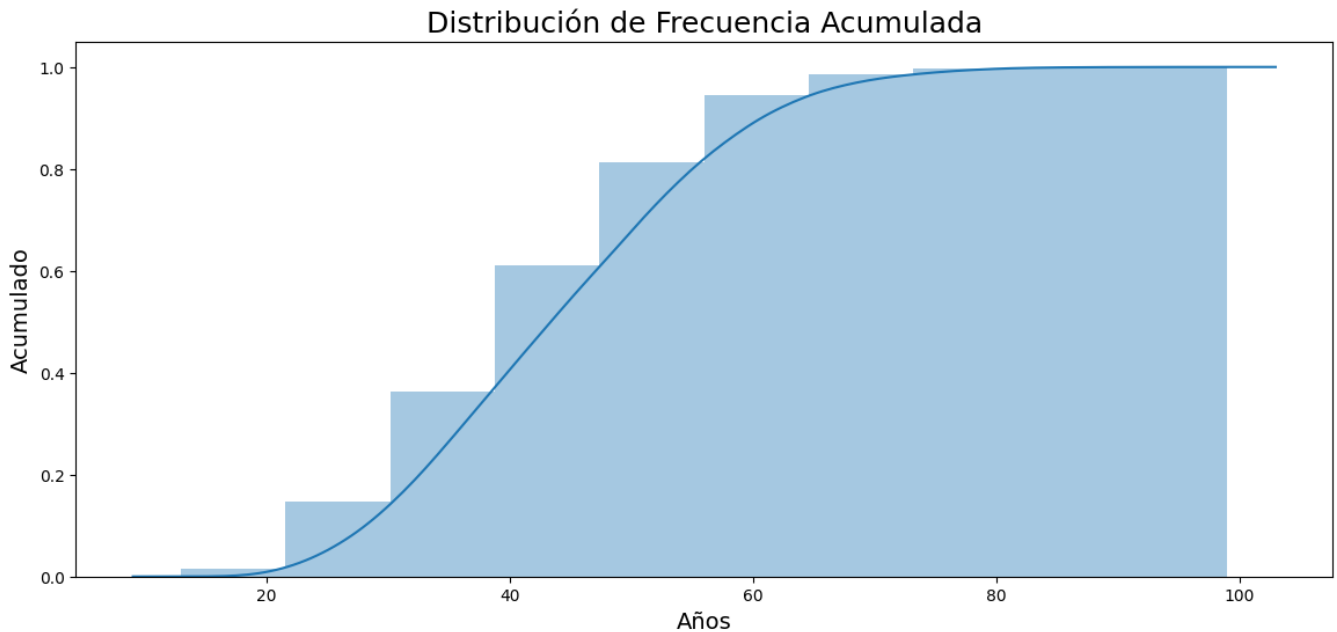
```
`distplot` is a deprecated function and will be removed in seaborn v0.14.0.
```

Please adapt your code to use either ``displot`` (a figure-level function with similar flexibility) or ``histplot`` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see

<https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751>

```
ax= sns.distplot(datos.Edad,  
<Axes: title={'center': 'Distribución de Frecuencia Acumulada'}, xlabel='Años',  
ylabel='Acumulado'>
```



Deciles de la variable edad

40% tiene valores debajo de 40. Tendríamos que aproximadamente un 80 y poco % de la muestra tiene valores de hasta 60 años.

```
datos.Edad.quantile([i/10 for i in range(1,10)] )
```

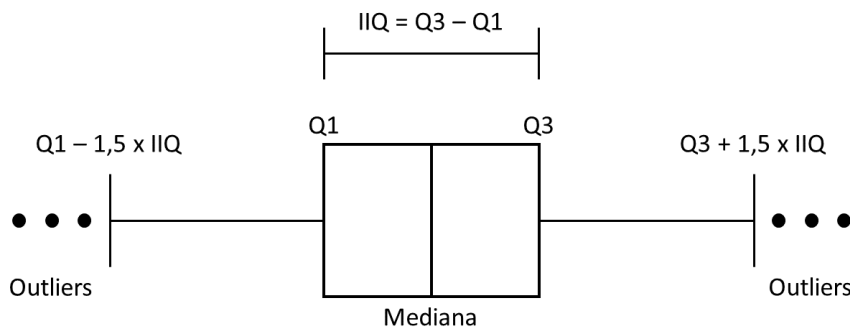
```

0.1    28.0
0.2    33.0
0.3    36.0
0.4    40.0
0.5    43.0
0.6    47.0
0.7    51.0
0.8    55.0
0.9    61.0
Name: Edad, dtype: float64

```

## ▼ 4.2 Box-plot

El *box plot* da una idea de la posición, dispersión, asimetría, colas y valores extremos. La posición central está dada por la mediana y la dispersión por *IIQ* (distancia inter cuartil). Las posiciones relativas de *Q1*, *Mediana* y *Q3* dan una idea de la simetría de la distribución. Las longitudes de las colas están dadas por las líneas que van desde el rectángulo a los valores remotos y por los valores extremos.



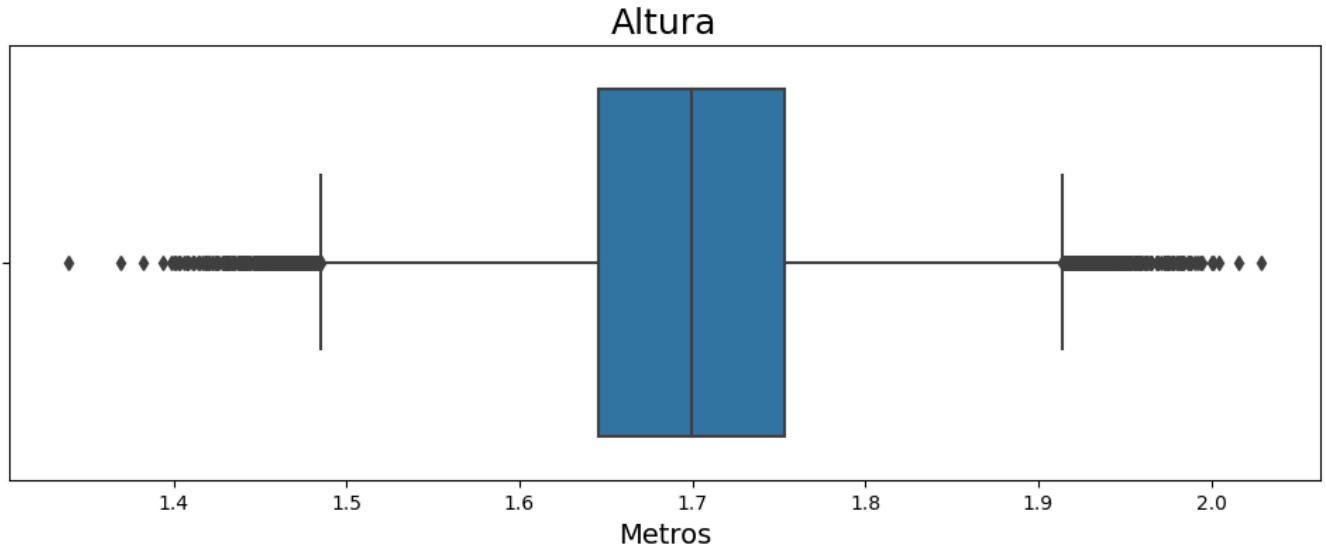
### Box-plot

Tenemos prácticamente la misma distancia entre el cuartil 1 y el cuartil 2, entre el cuartil 2 y cuartil 3, y básicamente tenemos la misma distancia entre el límite inferior y el cuartil 1 y entre el cuartil 3 y el límite superior. Y todos los puntos que se encuentran hacia los lados son posibles valores extremos o posibles outliers.

```
ax=sns.boxplot(x='Altura', data=datos, orient='h')
```

```
ax.figure.set_size_inches(12,4)
ax.set_title('Altura',fontsize=18)
ax.set_xlabel('Metros', fontsize=14)
ax
```

```
<Axes: title={'center': 'Altura'}, xlabel='Metros'>
```

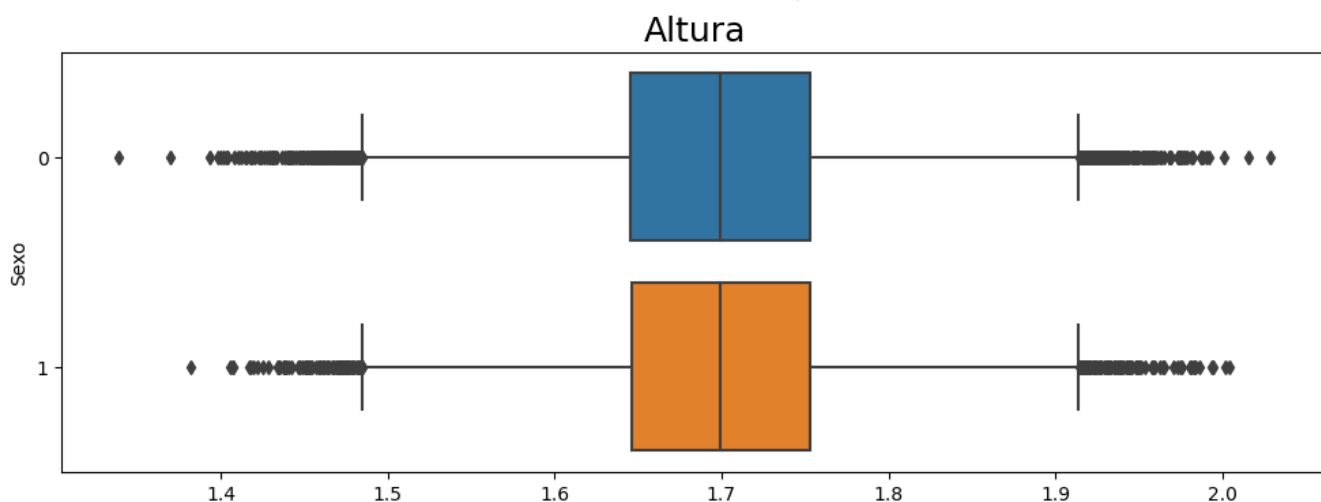


Dividiendo la variable y comparar boxplot de una misma variable de acuerdo a una a dos o varias categorías.

```
#comaparando altura y sexo
ax=sns.boxplot(x='Altura',y='Sexo', data=datos, orient='h')
```

```
ax.figure.set_size_inches(12,4)
ax.set_title('Altura',fontsize=18)
ax.set_xlabel('Metros', fontsize=14)
ax
```

```
<Axes: title={'center': 'Altura'}, xlabel='Metros', ylabel='Sexo'>
```



Analizando la variable ingreso

```
ax=sns.boxplot(x='Ingreso', data=datos, orient='h')
```

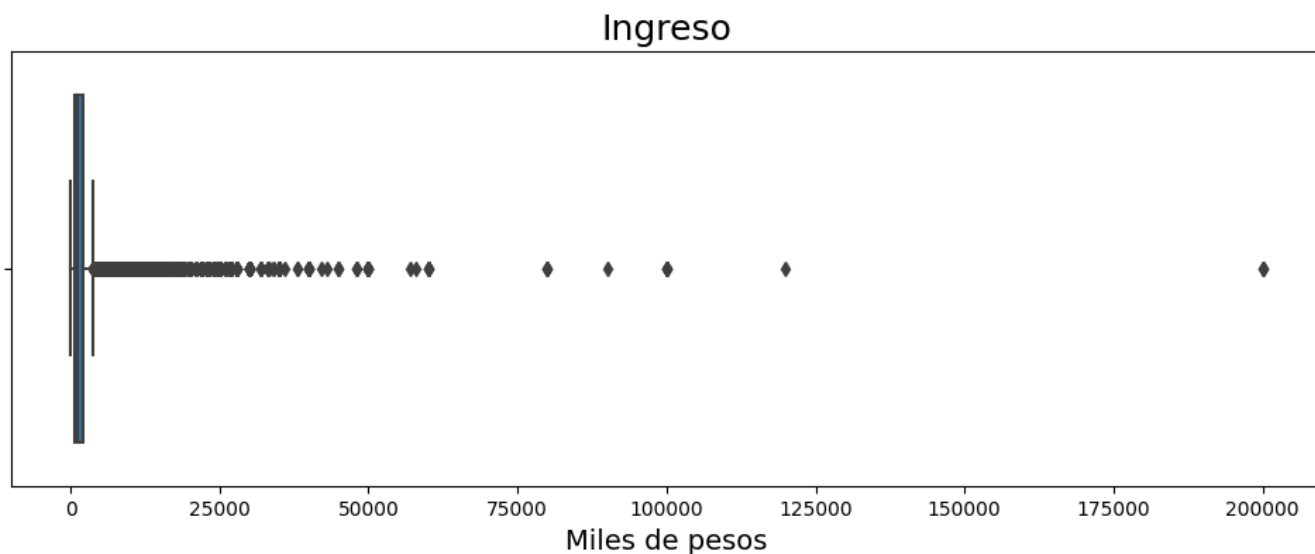
```
ax.figure.set_size_inches(12,4)
```

```
ax.set_title('Ingreso',fontsize=18)
```

```
ax.set_xlabel('Miles de pesos', fontsize=14)
```

```
ax
```

```
<Axes: title={'center': 'Ingreso'}, xlabel='Miles de pesos'>
```



Visualizando mejor lo anterior:

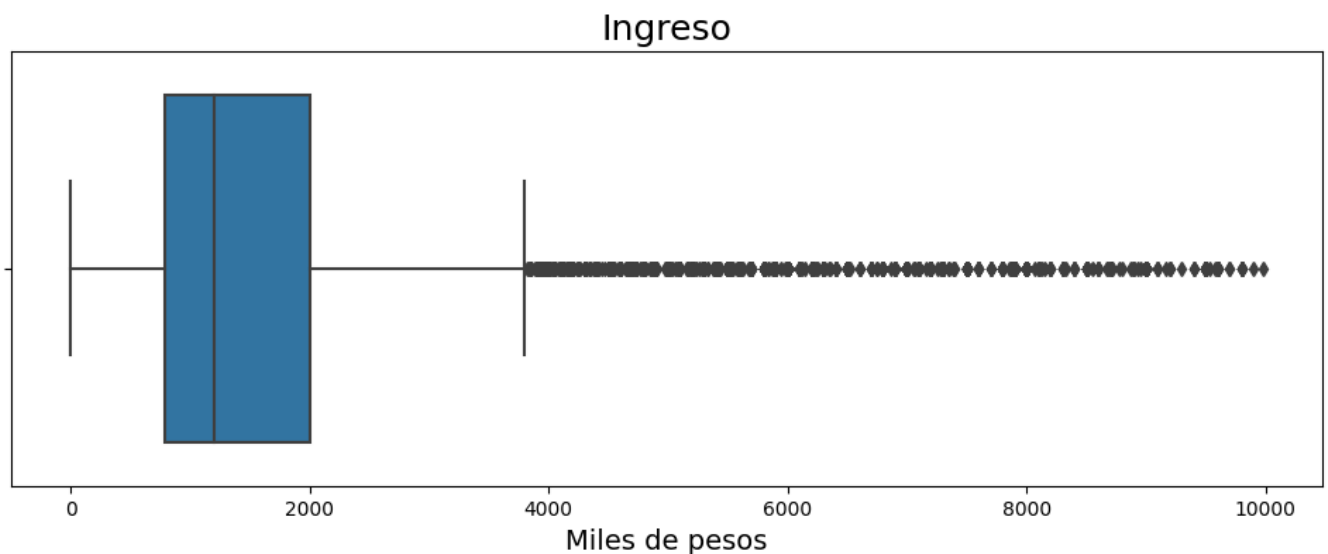


Entonces confirmamos que ella parece ser un poco asimétrica porque tiene aún muchos valores que jalen la variable hacia la derecha. Aquí la distancia entre la mediana y el cuartil 3 es mayor que entre el cuartil 1 y la mediana. Eso es una cosa que tiene una relación directa en el boxplot con la simetría de las variables.

```
ax=sns.boxplot(x='Ingreso', data=datos.query('Ingreso <10000'), orient='h')
```

```
ax.figure.set_size_inches(12,4)
ax.set_title('Ingreso',fontsize=18)
ax.set_xlabel('Miles de pesos', fontsize=14)
ax
```

```
<Axes: title={'center': 'Ingreso'}, xlabel='Miles de pesos'>
```

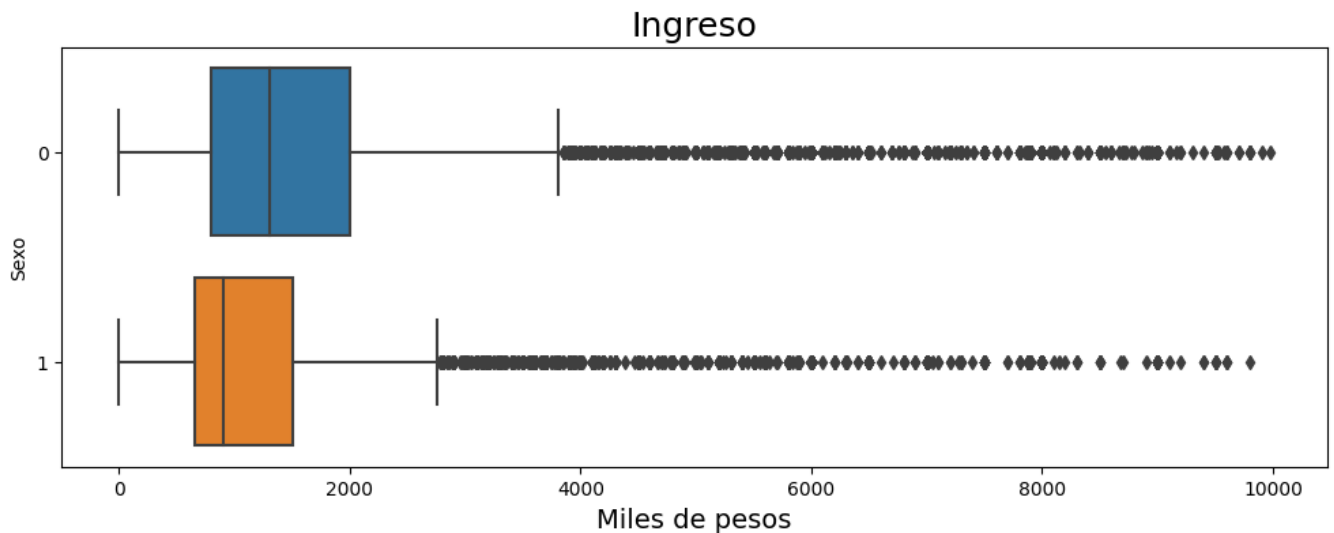


Visualizando con ña variable 'Sexo'

```
ax=sns.boxplot(x='Ingreso', y='Sexo', data=datos.query('Ingreso <10000'), orient='h')
```

```
ax.figure.set_size_inches(12,4)
ax.set_title('Ingreso',fontsize=18)
ax.set_xlabel('Miles de pesos', fontsize=14)
ax
```

```
<Axes: title={'center': 'Ingreso'}, xlabel='Miles de pesos', ylabel='Sexo'>
```



De lo anterior: el gráfico del boxplot muestra que claramente los hombres(caja azul), que eran el valor cero en sexo, parecen tener un ingreso mayor que las mujeres.El cuartil 1 está mayor, la mediana está mayor, el cuartil 3 está mayor.

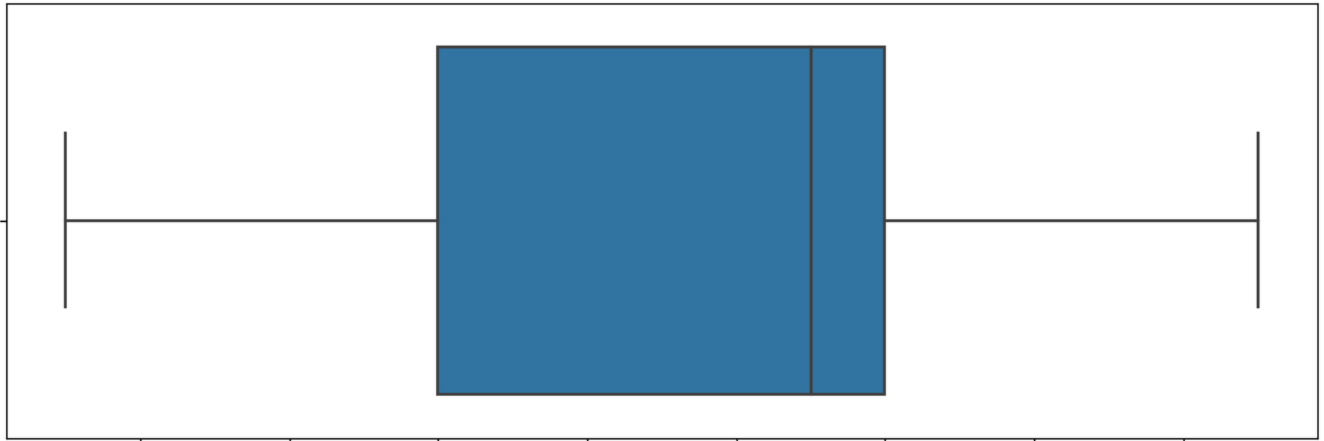
**Variable años de estudio:** No parece sugerir valores extremos esta variable, y además de eso parece que tiene una pequeña asimetría hacia la izquierda. Tiene un poco que la distancia allí entre el cuartil 1 y la mediana es bien mayor cuando la comparamos entre la mediana y el cuartil 3.

```
ax=sns.boxplot(x='Años de Estudio', data=datos, orient='h')
```

```
ax.figure.set_size_inches(12,4)
ax.set_title('Años de Estudio',fontsize=18)
ax.set_xlabel('Años', fontsize=14)
ax
```

```
<Axes: title={'center': 'Años de Estudio'}, xlabel='Años'>
```

### Años de Estudio



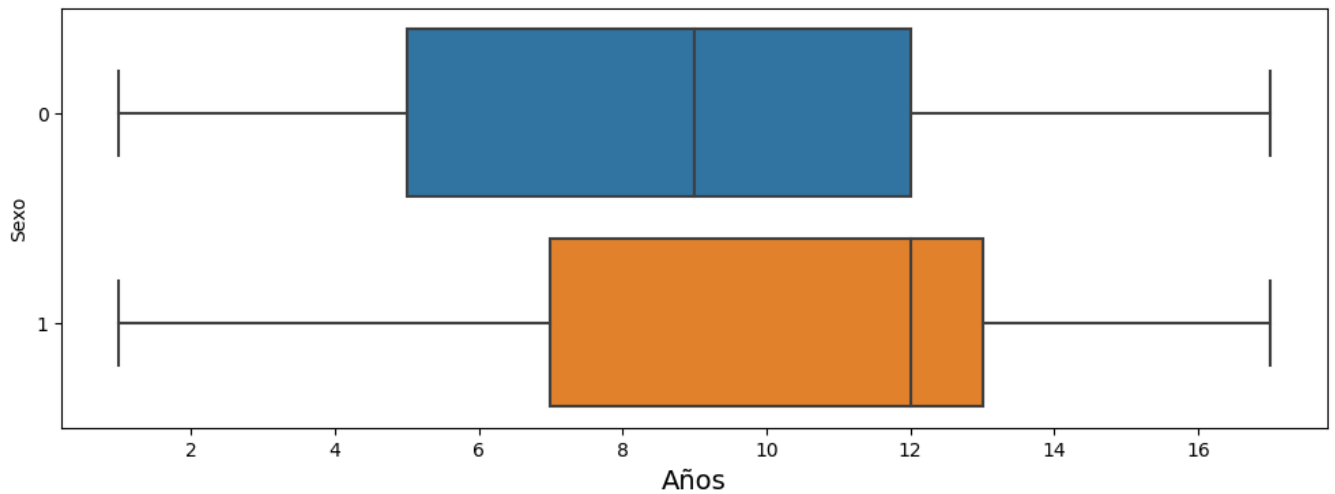
Las mujeres tienen más años de estudio que los hombres:

```
ax=sns.boxplot(x='Años de Estudio',y='Sexo', data=datos, orient='h')
```

```
ax.figure.set_size_inches(12,4)
ax.set_title('Años de Estudio',fontsize=18)
ax.set_xlabel('Años', fontsize=14)
ax
```

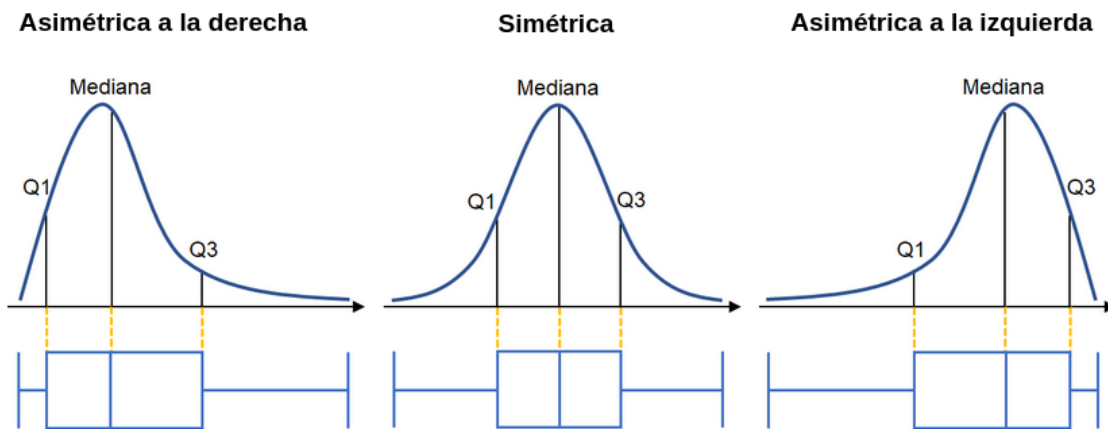
```
<Axes: title={'center': 'Años de Estudio'}, xlabel='Años', ylabel='Sexo'>
```

### Años de Estudio



Esto muestra un resultado que como confirmamos que analizando junto con los ingresos, teníamos que los hombres tienen más ingreso que las mujeres, parece que tienen más ingreso que las

mujeres, mientras tanto tienen menos años de estudio que las mujeres. O sea, esto parece ser un resultado que a veces sucede realmente y que el boxplot ya me está sugiriendo un análisis donde verifiquemos esa posible relación tanto entre la variable años de estudio como ingreso de acuerdo al sexo de las personas.



## ▼ 5 MEDIDAS DE DISPERSIÓN

Aunque las medidas de tendencia central y de localización proporcionan un resumen muy importante de los datos, pueden no ser suficientes para caracterizar diferentes conjuntos, especialmente cuando las observaciones de una distribución dada presentan datos muy dispersos.

### ▼ 5.1 Desviación media absoluta

$$DM = \frac{1}{n} \sum_{i=1}^n |X_i - \bar{X}|$$

df

Asignaturas	María	Pedro	Pablo
Matemática	8	10.0	7.5
Portugués	10	2.0	8.0

Visualizando las medias:

```
df.mean()
```

```
Asignaturas
María      7.714286
Pedro      5.142857
Pablo      7.714286
dtype: float64
```

Analizando medianas

```
df.median()
```

```
Asignaturas
María      8.0
Pedro      3.0
Pablo      8.0
dtype: float64
```

Calculando medidas de dispersión:

rodamos con los dos corchetes para convertirlo exactamente en un DataFrame.

```
notas_maria=df[['María']]
notas_maria
```

Asignaturas	María
Matemática	8
Portugués	10
Inglés	4
Geografía	8
Historia	6
Física	10
Química	8

## 1. Encontrando desviación respecto a la media

```
notas_media_maria= notas_maria.mean()[0]
notas_media_maria
```

7.714285714285714

Desvío en el conjunto de datos:

```
notas_maria['Desviación'] = notas_maria['María']-notas_media_maria
notas_maria
```

<ipython-input-117-8a7d17515ff4>:1: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: <https://pandas.pydata.org/pandas-docs/stable/user>  
notas\_maria['Desviación'] = notas\_maria['María']-notas\_media\_maria


Asignaturas	María	Desviación
Matemática	8	0.285714
Portugués	10	2.285714
Inglés	4	-3.714286
Geografía	8	0.285714
Historia	6	-1.714286
Física	10	2.285714
Química	8	0.285714

Creando una nueva variable para el desvío absoluto

```
notas_maria['Desviación']=notas_maria['Desviación'].abs()
notas_maria
```

```
<ipython-input-118-f9cd74c96380>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: <https://pandas.pydata.org/pandas-docs/stable/user>  
`notas_maria['Desviación']=notas_maria['Desviación'].abs()`

**Asignaturas** **María** **Desviación** 

**Matemática** 8 0.285714

**Portugués** 10 2.285714

Obteniendo la media de las desviaciones absolutas:

**Geografía** 8 0.285714

```
notas_maria['|Desviación|']=notas_maria['Desviación'].abs()
```

```
notas_maria
```

**Asignaturas** **María** **Desviación** **|Desviación|** 

**Matemática** 8 0.285714 0.285714

**Portugués** 10 2.285714 2.285714

**Inglés** 4 3.714286 3.714286

**Geografía** 8 0.285714 0.285714

**Historia** 6 1.714286 1.714286

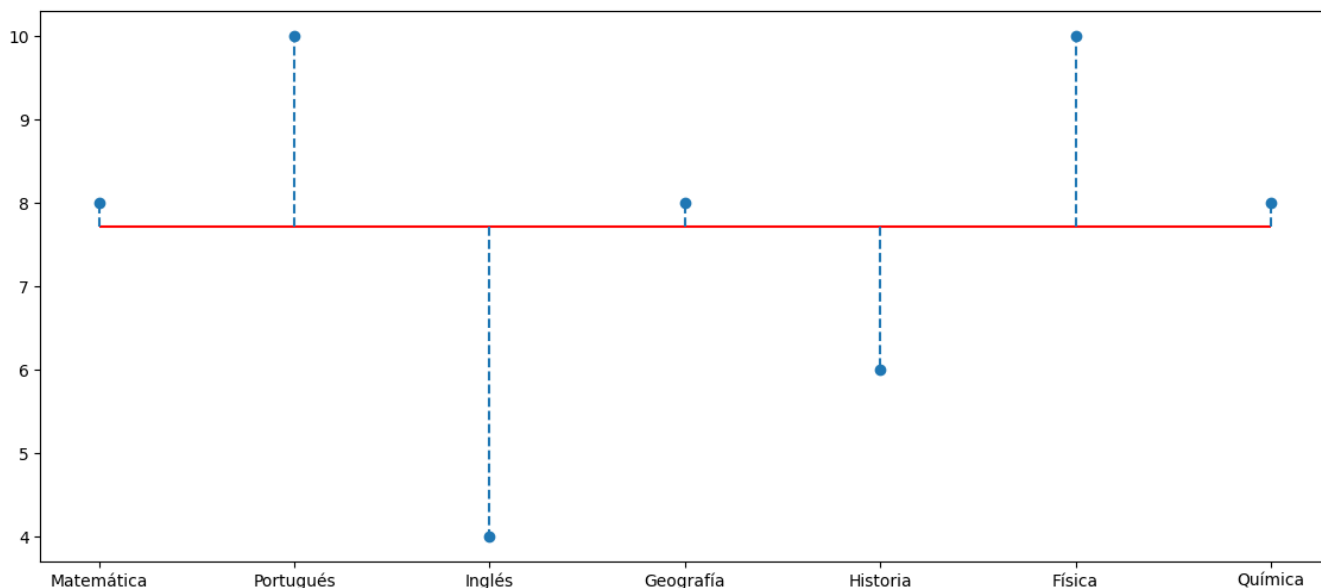
**Física** 10 2.285714 2.285714

**Química** 8 0.285714 0.285714

La línea roja es la media de las notas de María. Los puntos son los valores de las notas y la línea negra discontinua es el valor de la desviación

```
ax = notas_maria['María'].plot(style = 'o')
ax.figure.set_size_inches(14, 6)
ax.hlines(y = notas_media_maria, xmin = 0, xmax = notas_maria.shape[0] - 1, colors='red')
for i in range(notas_maria.shape[0]):
    ax.vlines(x = i, ymin = notas_media_maria, ymax = notas_maria['María'][i], linestyle='da
ax
```

&lt;Axes: &gt;



La media:

```
notas_maria['|Desviación|'].mean()
```

```
1.5510204081632648
```

Aplicando la función

```
notas_maria['María'].mad()
```

```
<ipython-input-122-17c50f3ea59d>:1: FutureWarning: The 'mad' method is deprecated and will be removed in a future version.
notas_maria['María'].mad()
1.5510204081632648
```

## ▼ 5.2 Varianza

### Varianza

La varianza se construye a partir de las diferencias entre cada observación y la media de los datos, es decir, la desviación alrededor de la media. Al calcular la varianza, las desviaciones alrededor de la media son elevadas al cuadrado.



## Varianza de la población

$$\sigma^2 = \frac{1}{n} \sum_{i=1}^n (X_i - \mu)^2$$


### ▼ Varianza de la muestra

$$S^2 = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2$$

Considerando las notas de María:

Agrega una nueva columna, elevando al cuadrado con ayuda de 'pow(2)'

```
notas_maria['(Desiación)^2'] = notas_maria['Desviación'].pow(2)
notas_maria
```

Asignaturas	María	Desviación	Desviación	(Desiación)^2	
<b>Matemática</b>	8	0.285714	0.285714	0.081633	
<b>Portugués</b>	10	2.285714	2.285714	5.224490	
<b>Inglés</b>	4	3.714286	3.714286	13.795918	
<b>Geografía</b>	8	0.285714	0.285714	0.081633	
<b>Historia</b>	6	1.714286	1.714286	2.938776	
<b>Física</b>	10	2.285714	2.285714	5.224490	
<b>Química</b>	8	0.285714	0.285714	0.081633	

Realizando manualmente las sumas de desviaciones al cuadrado:

- `notas_maria['(Desiación)^2'].sum()` devuelve la suma de los valores de la columna '(Desiación)^2'.
- `len(notas_maria)` devuelve el número total de filas en el DataFrame `notas_maria`.

- `len(notas_maria) - 1` se utiliza para calcular el denominador de la fórmula de la varianza muestral corregida, que se utiliza para estimar la varianza poblacional a partir de la varianza muestral.

```
#varianza de datos
notas_maria['(Desiación)^2'].sum()/(len(notas_maria) - 1)
```

```
4.57142857142857
```

Aplicando la función que tiene Python:

```
notas_maria['María'].var()
```

```
4.57142857142857
```

## ▼ 5.3 Desviación estándar

Una de las restricciones de la varianza es el hecho de que proporciona medidas cuadráticas de las unidades originales; la varianza de las medidas de longitud, por ejemplo, está en unidades de área. Por lo tanto, el hecho de que las unidades sean diferentes dificulta la comparación de la dispersión con las variables que la definen. Una forma de eliminar esta dificultad es considerar su raíz cuadrada.

Desviación estándar de la población

$$\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n (X_i - \mu)^2} \implies \sigma = \sqrt{\sigma^2}$$

### ▼ Desviación estándar de la muestra

$$S = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2} \implies S = \sqrt{S^2}$$

Raíz cuadrada de la varianza de las notas de María

```
varianza= notas_maria['María'].var()
varianza
```

4.57142857142857

Con una función de numpy calculamos la desviación estándar de las notas de María

```
np.sqrt(varianza)
```

2.1380899352993947

Otra forma de calcularlo:

```
desviacion_estandar = notas_maria['María'].std()
desviacion_estandar
```

2.1380899352993947

Comparando datos de los 3 conjuntos de datos, es decir de los 3 alumnos:

```
df
```

Asignaturas	María	Pedro	Pablo	
<b>Matemática</b>	8	10.0	7.5	
<b>Portugués</b>	10	2.0	8.0	
<b>Inglés</b>	4	0.5	7.0	
<b>Geografía</b>	8	1.0	8.0	
<b>Historia</b>	6	3.0	8.0	
<b>Física</b>	10	9.5	8.5	
<b>Química</b>	8	10.0	7.0	

Media:

```
df.mean()
```

```
Asignaturas
María      7.714286
```

```
Pedro    5.142857
Pablo    7.714286
dtype: float64
```

Mediana:

```
df.median()
```

```
Asignaturas
María    8.0
Pedro    3.0
Pablo    8.0
dtype: float64
```

Moda

```
df.mode()
```

Asignaturas	María	Pedro	Pablo
0	8	10.0	8.0



Las medidas de tendencia central no son suficiente ahora para describir todo el comportamiento que puede estar sucediendo con nuestros datos. Ahora la desviación estándar de María es mayor que la de Pablo. O sea ella puede tener el mismo valor central pero tiene una variabilidad mayor, tiene una desviación estándar mayor, lo que significa que ella no es tan constante en cuanto a sus notas.

Visualizando la desviación estándar:


```
df.std()
```

```
Asignaturas
María    2.138090
Pedro    4.460141
Pablo    0.566947
dtype: float64
```

Pablo tiene una desviación estándar pequeña, lo que significa que si su nota central es 8, él es constante en ese valor, o sea, él realmente obtiene siempre notas muy cerca de 8. Ya María tiene una variabilidad mayor y por eso las notas de ella tienen una desviación estándar mayor que las de Pablo.

## Un ejemplo extra:

```
dataset = pd.DataFrame({
    'Género': ['H', 'M', 'M', 'M', 'M', 'H', 'H', 'H', 'M', 'M'],
    'Edad': [53, 72, 54, 27, 30, 40, 58, 32, 44, 51]
})
dataset
```

	Género	Edad	
0	H	53	
1	M	72	
2	M	54	
3	M	27	
4	M	30	
5	H	40	
6	H	58	
7	H	32	
8	M	44	
9	M	51	

Desviación estándar de la edad:

```
edad_std= dataset['Edad'].std()
edad_std
```

```
14.184890239656813
```

```
dataset.std()
```

```
<ipython-input-142-d926424df4d9>:1: FutureWarning: The default value of numeric_only in
dataset.std()
Edad      14.18489
dtype: float64
```

Desviación estándar de H: hombres

```
hombres_std = dataset.loc[dataset['Género']=='H', 'Edad' ].std()
print('La desviación estándar de las edades para el género H es:', hombres_std)
```

La desviación estándar de las edades para el género H es: 11.89887949906769

Desviación estándar de M: Mujeres

✓ 0 s se ejecutó 17:32

