In [1]:

```
%load_ext watermark
%watermark
```

```
2019-05-30T20:59:14+02:00

CPython 3.6.5
IPython 6.4.0

compiler   : GCC 7.2.0
system     : Linux
release    : 5.1.5-arch1-2-ARCH
machine    : x86_64
processor  :
CPU cores  : 4
interpreter: 64bit
```

ESTADÍSTICOS DESCRIPTIVOS

In [2]:

```
import numpy as np
import pandas as pd
import scipy.stats as stats
```

Aplicacion de la teoría

Para comprobar la teoría y aplicar los distintos estadísticos descriptivos se utilizará una lista con distintos valores numéricos. En el ejemplo se tratarán como una lista de pesos de productos.

In [3]:

```
pesos = [100,150,150,200,250,300,325,400,415,500,600,1000]
```

Estadísticos de tendencia Central

Las medidas de tendencia central, son medidas estadísticas que pretenden resumir en un solo valor a un conjunto de valores. Representan un centro en torno al cual se encuentra ubicado el conjunto de los datos. Las medidas de tendencia central más utilizadas son la media, la mediana y la moda.

Media: para calcular la media del peso, utilizamos la librería numpy y su método mean pasandole como parametro la lista de pesos

In [4]:

```
#Obtenemos la media de los
peso_media = np.mean(pesos)
print(peso_media)
```

```
365.8333333333333
```

Mediana: para calcular la mediana del peso, utilizamos la librería numpy y su método median pasandole como parametro la lista de pesos

In [5]:

```
peso_mediana = np.median(pesos)
print(peso_mediana)
```

```
312.5
```

Moda: para calcular la moda del peso, utilizamos la librería scipy (clase stats) y su método mode pasandole como parametro la lista de pesos

In [6]:

```
peso_moda = stats.mode(pesos)
```

```
print(peso_moda)
```

```
ModeResult(mode=array([150]), count=array([2]))
```

Estadísticos de dispersión

Las medidas de dispersión miden el grado de dispersión de los valores de una variable. Pretenden evaluar en qué medida los datos difieren entre sí, es decir, si las diferentes puntuaciones de una variable están muy alejadas de la media, por lo que cuanto mayor sea ese valor mayor será la variabilidad y cuanto menor sea, más homogénea será a la media.

Rango: para calcular el rango del peso, utilizamos la librería numpy, utilizamos los metodos maximo y minimo y los restamos

In [7]:

```
peso_rango = np.max(pesos) - np.min(pesos)
print(peso_rango)
```

```
900
```

Cuartiles y IQR: Dentro de la librería stats disponemos de una clase mstats y un método mquantiles

In [8]:

```
peso_cuantiles = stats.mstats.mquantiles(pesos)
print(peso_cuantiles)
```

```
[172.5  312.5  461.75]
```

La función pandas.qcut convierte una serie en una lista de N cuantiles. Si le pasasemos 4, tenemos los cuartiles. qcut devuelve una serie categórica nueva con los cuantiles como categorías

In [9]:

```
pesos_cuartiles = pd.qcut(pesos,4)
pesos_cuartiles.categories
```

Out[9]:

```
IntervalIndex([(99.999, 187.5], (187.5, 312.5], (312.5, 436.25], (436.25, 1000.0]],
              closed='right',
              dtype='interval[float64]')
```

Desviacion Standard: Con la librería numpy y el método std podemos obtener la desviación standard

In [10]:

```
peso_std = np.std(pesos)
print(peso_std)
```

```
239.6771555423856
```

Estadísticos de forma

Son aquellos que nos hablan de la forma de la distribución de datos en cuanto a su simetría y su apuntamiento.

Coeficiente de simetria: El coeficiente de simetría (skewness) se define como el tercer momento dividido por la desviación standard al cubo.

$$\frac{1}{N} * \frac{\sum_{n=1}^{n} (Xi-\bar{X})^{3}}{\sigma^3}$$

In [11]:

```
pow3 = lambda x: x*x*x
tercer_momento = lambda x: pow3(x-peso_media)
simetria_pesos = sum(map(tercer_momento,pesos))/(12*pow3(peso_std))
print(simetria_pesos)
```

```
1.3623858394083475
```

Coeficiente de curtosis: El coeficiente de curtosis se define como el cuarto momento dividido por la desviación estandard a la cuarta, tiene la siguiente formula:

$$\frac{1}{N} * \frac{\sum_{n=1}^{n} (Xi-\bar{X})^{4}}{\sigma^4}-3$$

Generalmente se le resta 3 al cuarto momento, ya que una distribución normal perfecta tiene un coeficiente de curtosis de 3. En la siguiente ejecución describimos la ecuación a mano

In [12]:

```
pow4 = lambda x: x*x*x*x
cuarto_momento = lambda x: pow4(x - peso_media)
curtosis_peso = sum(map(cuarto_momento,pesos))/(12*pow4(peso_std))-3
print(curtosis_peso)
```

```
1.4285722765161841
```

En este caso gracias a la libreria **scipy** y a su metodo **kurtosis** podemos obtenerlo directamente sin necesidad de desarrollar nosotros la ecuación

In [13]:

```
curtosis_peso2 = stats.kurtosis(pesos)
print(curtosis_peso2)
```

```
1.4285722765161841
```