UNAM

Ejercicio matrical MCO

Gómez Karla

__Minimos Cuadrados Ordinarios Matricial con Python__

$$y = X\beta + \upsilon$$

$$y_i = \hat{\alpha_0} + \hat{\beta}_1 X_i + \hat{\epsilon}_i$$
$$\hat{\beta}_i = (X'X)^{-1}X'y$$

Para obtener el vector *__beta__* de los MCO mediante algebra lineal se hara uso de las librerias:

- Pandas
- Numpy

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
militar = pd.read_csv('Datos/gmilitar.csv')
militar.head()
```

Out[2]:

	YEAR	Y	X2	X3	X4	X5
0	1962	51.1	560.3	0.6	16.0	0
1	1963	52.3	590.5	0.9	16.4	0
2	1964	53.6	632.4	1.1	16.7	0
3	1965	49.6	684.9	1.4	17.0	1
4	1966	56.8	749.9	1.6	20.2	1

In [3]:

```
militar.set_index('YEAR', inplace = True)
```

In [4]:

```
militar.head()
```

Out[4]:

	Y	X2	X3	X4	X5
YEAR					
1962	51.1	560.3	0.6	16.0	0
1963	52.3	590.5	0.9	16.4	0
1964	53.6	632.4	1.1	16.7	0
1965	49.6	684.9	1.4	17.0	1
1966	56.8	749.9	1.6	20.2	1

Y := Gasto en defensa anual en Billones de dolares

X2 := GNP = PNB anual en Billones de dolares

X3 := Ventas anuales del ejercito en Billones de dolares

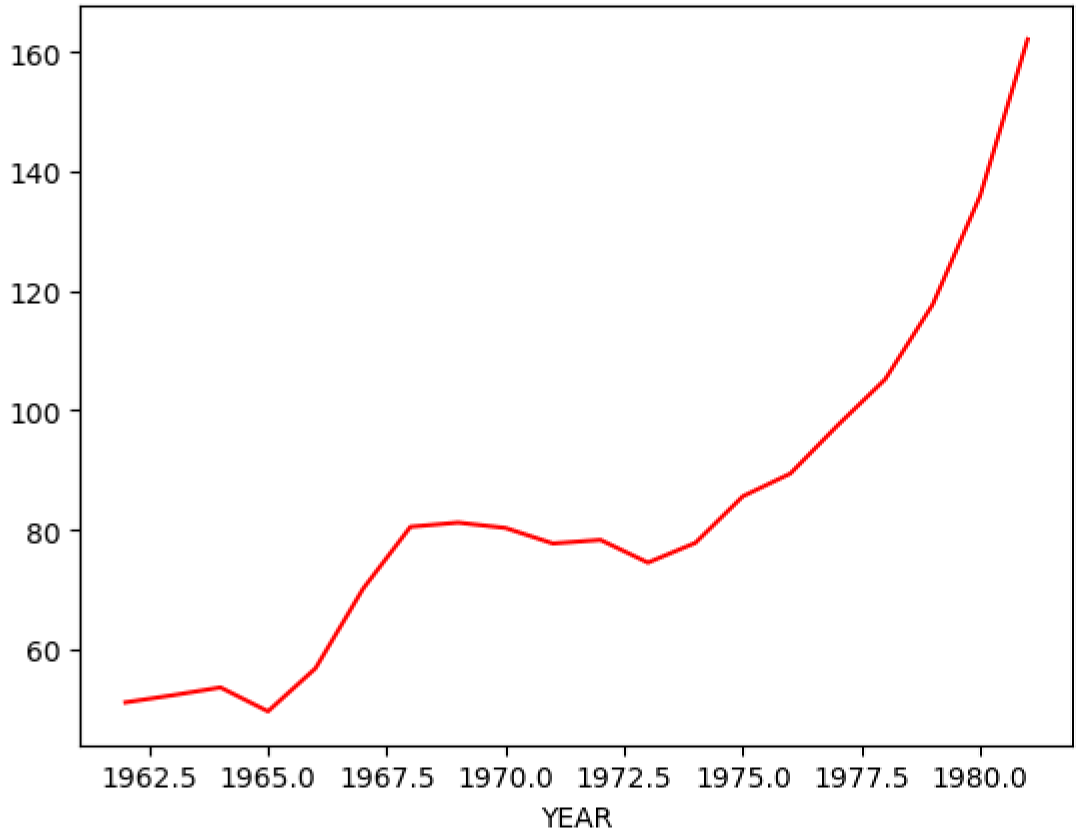
X4 := Sales de la industrai aeroespacial en Billones de dolares

X5 := Es una variable binaria (dummy) {1,0}. Donde 1 indica que hay un conflicto belico y 0 hay 'paz'

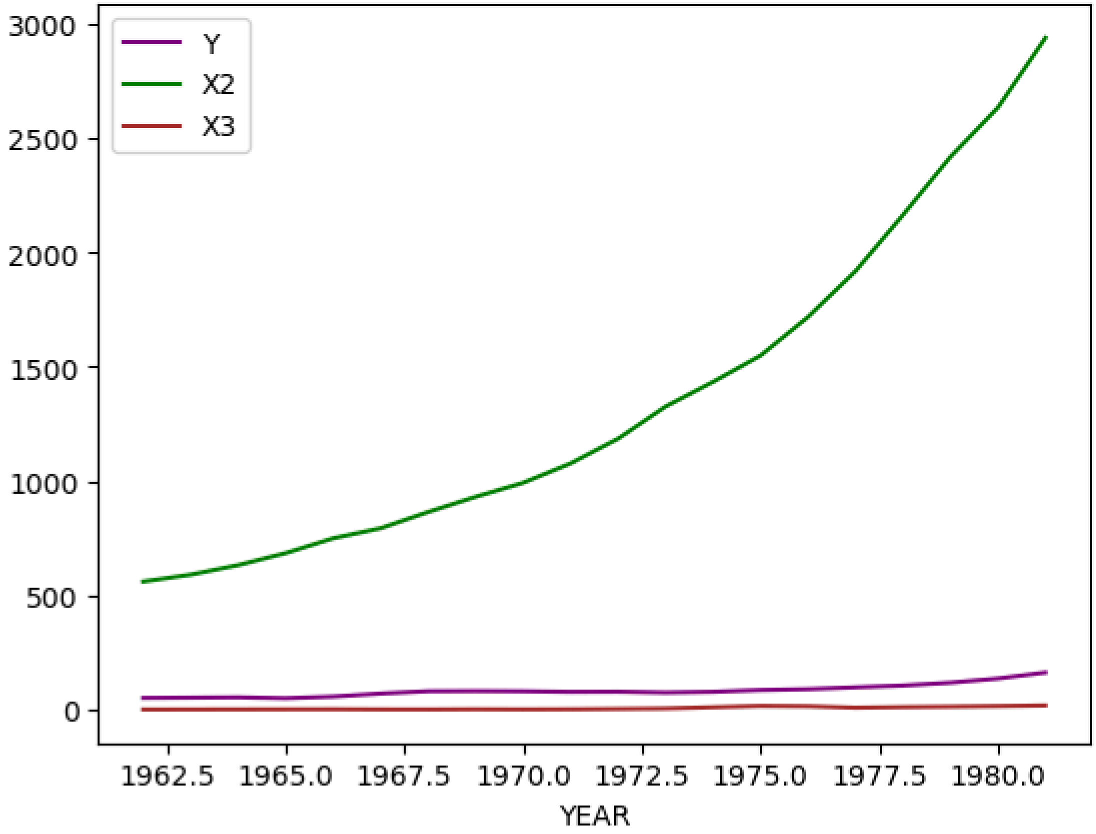
In [5]:

```
import matplotlib as mpl
import matplotlib.pyplot as plt
```

In [7]:
```
grafico1 = militar['Y'].plot(color = 'red')
```

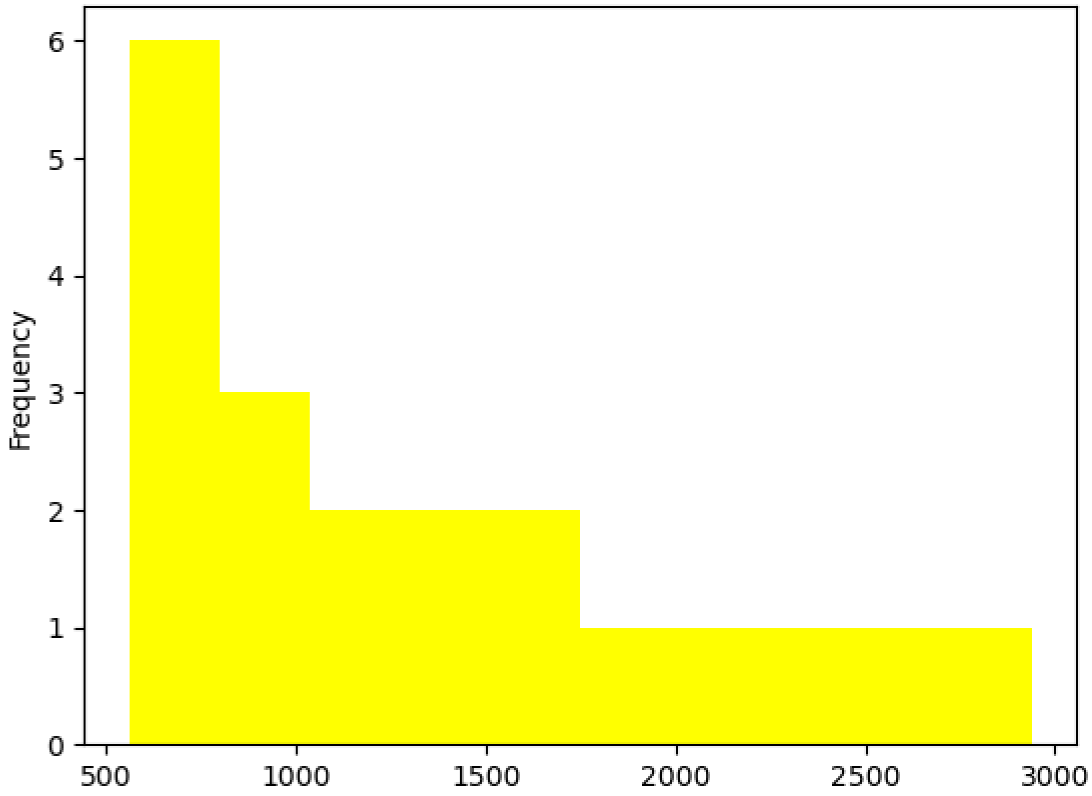


In [8]:
```
grafico2 = militar[['Y', 'X2', 'X3']].plot(color = ('purple', 'green', 'brown'))
```



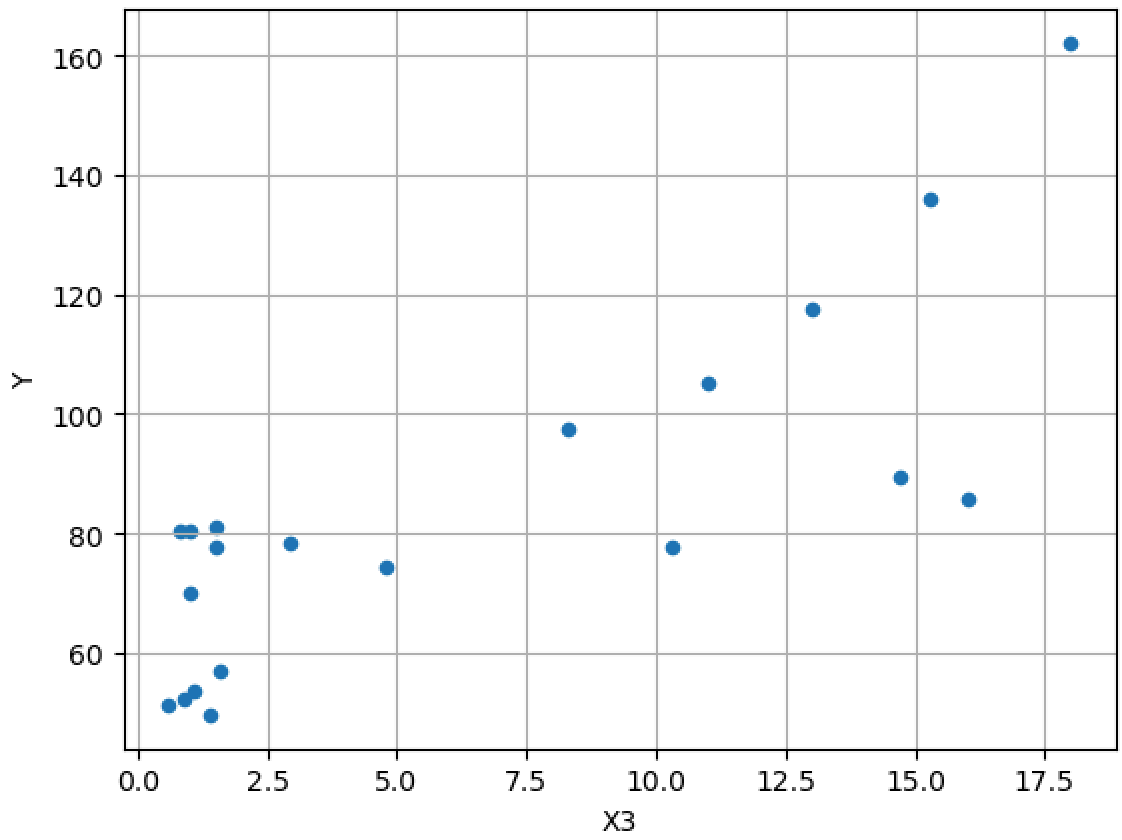
In [9]:
```
militar['X2'].plot.hist(color = 'yellow')
```

Out[9]:
```
<Axes: ylabel='Frequency'>
```

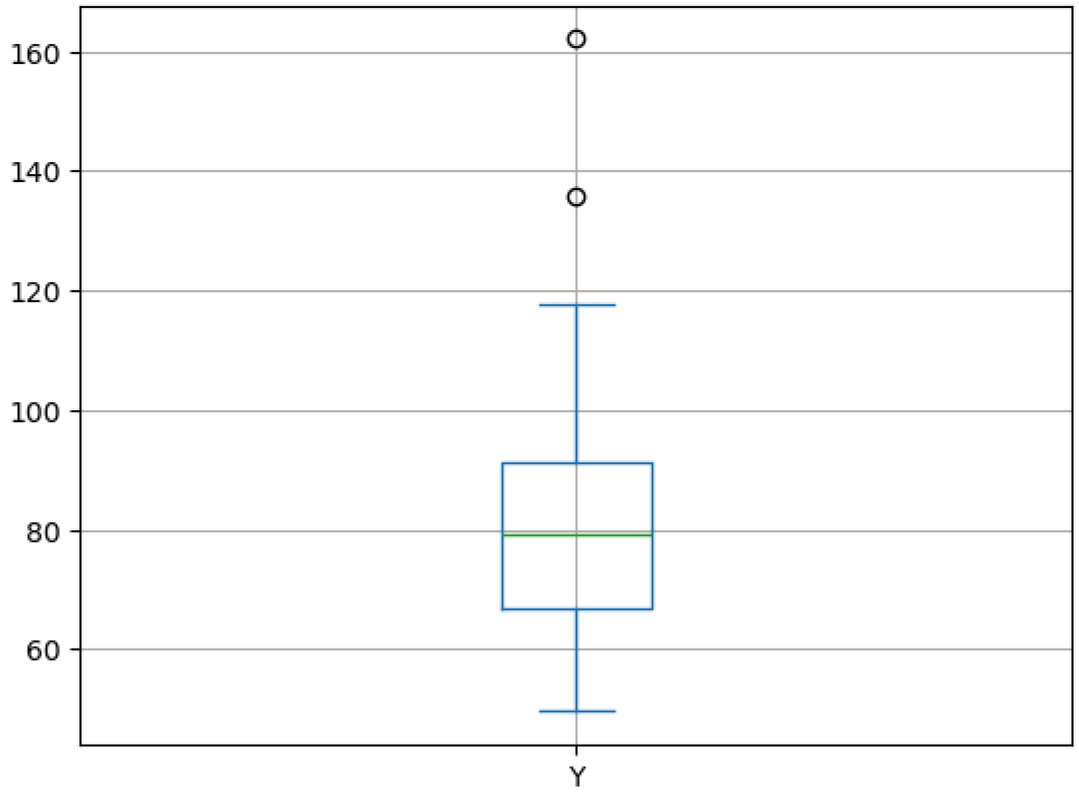


In [10]:
```
militar.plot.scatter(x = 'X3', y = 'Y',
                     grid = True)
```

Out[10]: <Axes: xlabel='X3', ylabel='Y'>

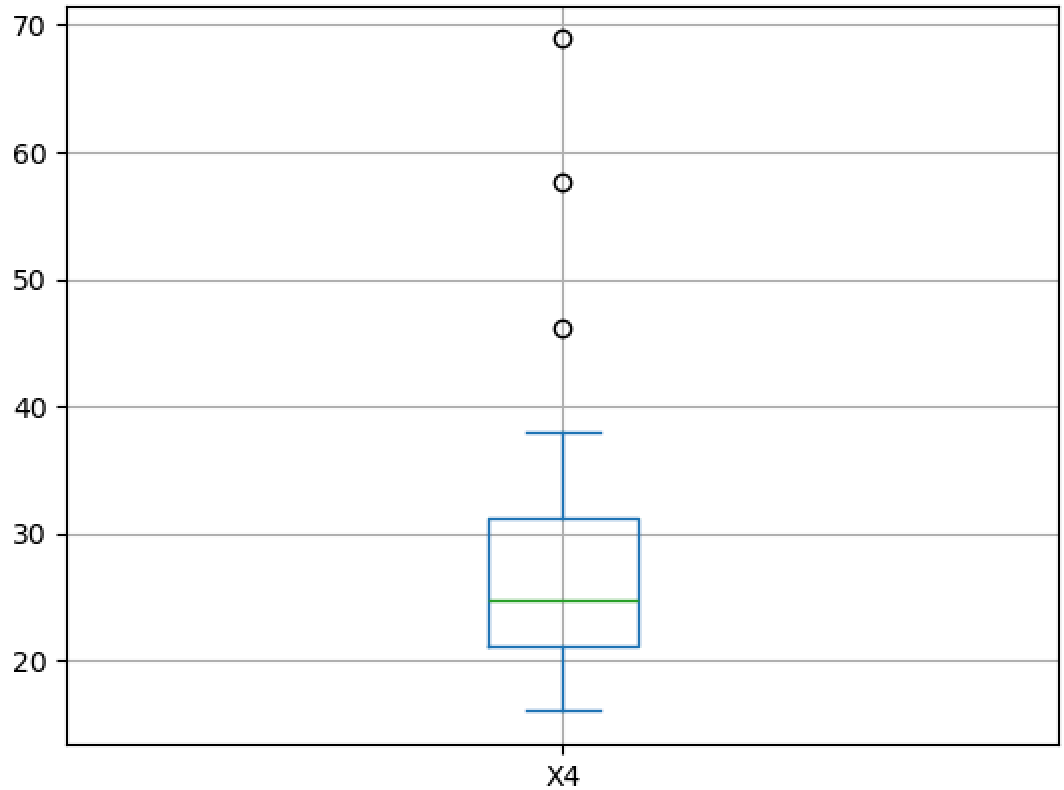


In [11]:
```
boxplot1 = militar.plot.box(y = ['Y'],
                            grid = True)
```

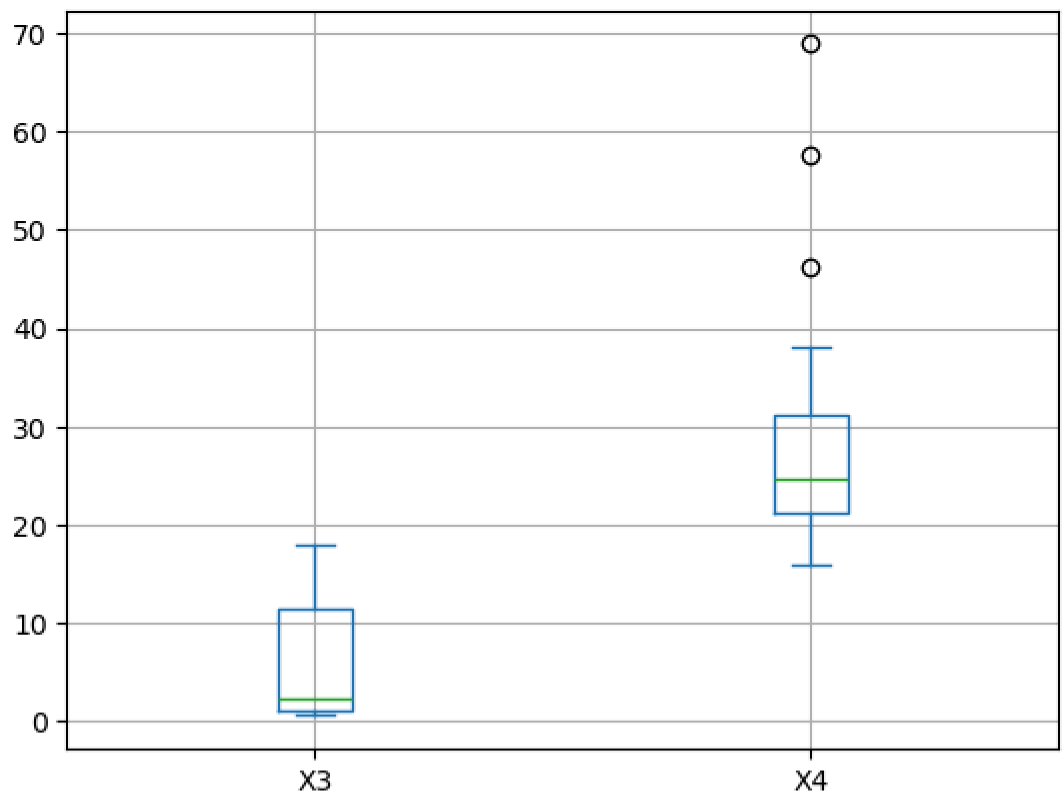


In [ ]:
```
# Ejercicio
# Construir el grafico de caja:
## 1. Para X4
## 2. Para X3 y X4 en solo grafico
```

In [13]:
```
boxplot_X4 = militar.plot.box(y = ['X4'],
                              grid = True)
```



In [12]:
```
boxplot_X3_X4 = militar.plot.box(y = ['X3','X4'],
                                 grid = True)
```



In [14]:
```
militar.describe()
```

Out[14]:

	Y	X2	X3	X4	X5
count	20.000000	20.000000	20.000000	20.000000	20.000000
mean	83.860000	1358.155000	6.287500	29.145000	0.400000
std	28.977712	723.316163	6.284021	13.999943	0.502625
min	49.600000	560.300000	0.600000	16.000000	0.000000
25%	66.775000	782.900000	1.075000	21.175000	0.000000
50%	79.300000	1131.750000	2.275000	24.700000	0.000000
75%	91.425000	1768.075000	11.500000	31.125000	1.000000
max	162.100000	2937.700000	18.000000	68.900000	1.000000

Construir el vector beta

$\hat{\beta}_i = (X'X)^{-1}X'y$

In [15]:

```
# generar la variable X1 = 1 y agregarla al dataframe
militar['X1'] =1
militar.head()
```

Out[15]:

	Y	X2	X3	X4	X5	X1
YEAR						
1962	51.1	560.3	0.6	16.0	0	1
1963	52.3	590.5	0.9	16.4	0	1
1964	53.6	632.4	1.1	16.7	0	1
1965	49.6	684.9	1.4	17.0	1	1
1966	56.8	749.9	1.6	20.2	1	1

Transformar el dataframe a un arreglo matrical con numpy

In [16]:
```
y = militar['Y'].to_numpy()
display(y)
```

```
array([ 51.1,  52.3,  53.6,  49.6,  56.8,  70.1,  80.5,  81.2,  80.3,
        77.7,  78.3,  74.5,  77.8,  85.6,  89.4,  97.5, 105.2, 117.7,
       135.9, 162.1])
```

In [18]:
```
X = militar[['X1', 'X2', 'X3', 'X4']].to_numpy()
display(X)
```

```
array([[1.0000e+00, 5.6030e+02, 6.0000e-01, 1.6000e+01],
       [1.0000e+00, 5.9050e+02, 9.0000e-01, 1.6400e+01],
       [1.0000e+00, 6.3240e+02, 1.1000e+00, 1.6700e+01],
       [1.0000e+00, 6.8490e+02, 1.4000e+00, 1.7000e+01],
       [1.0000e+00, 7.4990e+02, 1.6000e+00, 2.0200e+01],
       [1.0000e+00, 7.9390e+02, 1.0000e+00, 2.3400e+01],
       [1.0000e+00, 8.6500e+02, 8.0000e-01, 2.5600e+01],
       [1.0000e+00, 9.3140e+02, 1.5000e+00, 2.4600e+01],
       [1.0000e+00, 9.9270e+02, 1.0000e+00, 2.4800e+01],
       [1.0000e+00, 1.0776e+03, 1.5000e+00, 2.1700e+01],
       [1.0000e+00, 1.1859e+03, 2.9500e+00, 2.1500e+01],
       [1.0000e+00, 1.3264e+03, 4.8000e+00, 2.4300e+01],
       [1.0000e+00, 1.4342e+03, 1.0300e+01, 2.6800e+01],
       [1.0000e+00, 1.5492e+03, 1.6000e+01, 2.9500e+01],
       [1.0000e+00, 1.7180e+03, 1.4700e+01, 3.0400e+01],
       [1.0000e+00, 1.9183e+03, 8.3000e+00, 3.3300e+01],
       [1.0000e+00, 2.1639e+03, 1.1000e+01, 3.8000e+01],
       [1.0000e+00, 2.4178e+03, 1.3000e+01, 4.6200e+01],
       [1.0000e+00, 2.6331e+03, 1.5300e+01, 5.7600e+01],
       [1.0000e+00, 2.9377e+03, 1.8000e+01, 6.8900e+01]])
```

In [19]:
```
# Evitar la notacion cientifica
np.set_printoptions(suppress = True)
```

In [20]:
```
display(X)
```

```
array([[   1.  ,  560.3 ,    0.6 ,   16.  ],
       [   1.  ,  590.5 ,    0.9 ,   16.4 ],
       [   1.  ,  632.4 ,    1.1 ,   16.7 ],
       [   1.  ,  684.9 ,    1.4 ,   17.  ],
       [   1.  ,  749.9 ,    1.6 ,   20.2 ],
       [   1.  ,  793.9 ,    1.  ,   23.4 ],
       [   1.  ,  865.  ,    0.8 ,   25.6 ],
       [   1.  ,  931.4 ,    1.5 ,   24.6 ],
       [   1.  ,  992.7 ,    1.  ,   24.8 ],
       [   1.  , 1077.6 ,    1.5 ,   21.7 ],
       [   1.  , 1185.9 ,    2.95,   21.5 ],
       [   1.  , 1326.4 ,    4.8 ,   24.3 ],
       [   1.  , 1434.2 ,   10.3 ,   26.8 ],
       [   1.  , 1549.2 ,   16.  ,   29.5 ],
       [   1.  , 1718.  ,   14.7 ,   30.4 ],
       [   1.  , 1918.3 ,    8.3 ,   33.3 ],
       [   1.  , 2163.9 ,   11.  ,   38.  ],
       [   1.  , 2417.8 ,   13.  ,   46.2 ],
       [   1.  , 2633.1 ,   15.3 ,   57.6 ],
       [   1.  , 2937.7 ,   18.  ,   68.9 ]])
```

In [21]:

```
# Transpuesta de X
trX = X.T
print(trX)
```

```
[[   1.      1.      1.      1.      1.      1.      1.      1.      1.
     1.      1.      1.      1.      1.      1.      1.      1.      1.
     1.      1.  ]
 [ 560.3   590.5   632.4   684.9   749.9   793.9   865.    931.4   992.7
  1077.6  1185.9  1326.4  1434.2  1549.2  1718.   1918.3  2163.9  2417.8
  2633.1  2937.7 ]
 [   0.6     0.9     1.1     1.4     1.6     1.      0.8     1.5     1.
     1.5     2.95    4.8    10.3    16.     14.7     8.3    11.     13.
    15.3    18.   ]
 [  16.     16.4    16.7    17.     20.2    23.4    25.6    24.6    24.8
    21.7    21.5    24.3    26.8    29.5    30.4    33.3    38.     46.2
    57.6    68.9 ]]
```

In [22]:

```
# Obtener la matriz (X'X)
X_X = np.dot(trX, X)
print(X_X)
```

```
[[      20.         27163.1          125.75         582.9     ]
 [   27163.1     46832239.23     248214.475     973615.51    ]
 [     125.75      248214.475       1540.9425     5028.435 ]
 [     582.9       973615.51        5028.435      20712.59   ]]
```

In [23]:
```
# determinante de X_X
print(np.linalg.det(X_X))
```

```
10949028165453.97
```

In [24]:
```
# Inversa de (X'X)
invX_X=np.linalg.inv(X_X)
print(invX_X)
```

```
[[ 0.38041615 -0.00020991  0.0264947  -0.00727079]
 [-0.00020991  0.00000171 -0.00007354 -0.00005653]
 [ 0.0264947  -0.00007354  0.0071487   0.00097556]
 [-0.00727079 -0.00005653  0.00097556  0.00267315]]
```

In [25]:
```
# Obtener Xy
Xy = np.dot(trX,y)
Xy
```

Out[25]:
```
array([   1677.2   , 2657079.68 ,   13317.485,   56437.61 ])
```

In [26]:
```
# obtener beta
beta = np.dot(invX_X, Xy)
print(beta)
```

```
[22.77514195  0.01670288 -0.6961735   1.46772866]
```

In [28]:
```
print( 'y = 22.775 + 0.016X2 - 0.696X3 + 1.467X4')
```

```
y = 22.775 + 0.016X2 - 0.696X3 + 1.467X4
```

In [ ]: