!!! Un poco más de Python en Análisis de datos.

Ejemplo2:

 # Cargando datos de archivo CSV

>>> import pandas as pd

#Recordar donde localizaron sus datos

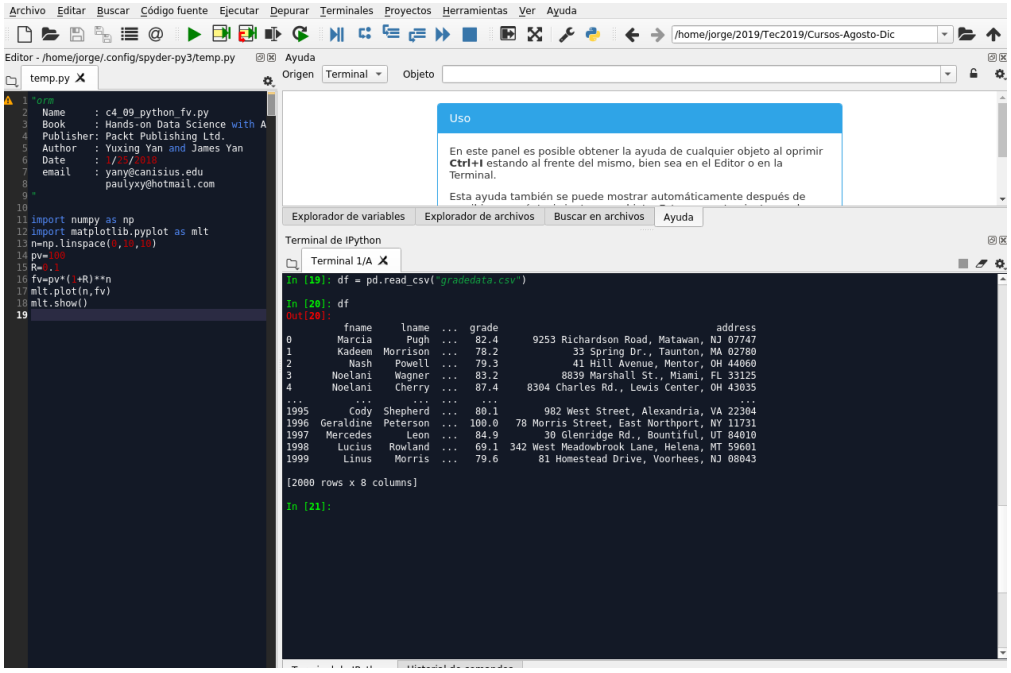
>>> Localizacion = "datos/gradedata.csv"

>>> df = pd.read_csv(Localizacion, header=None)}

>>> df.head()

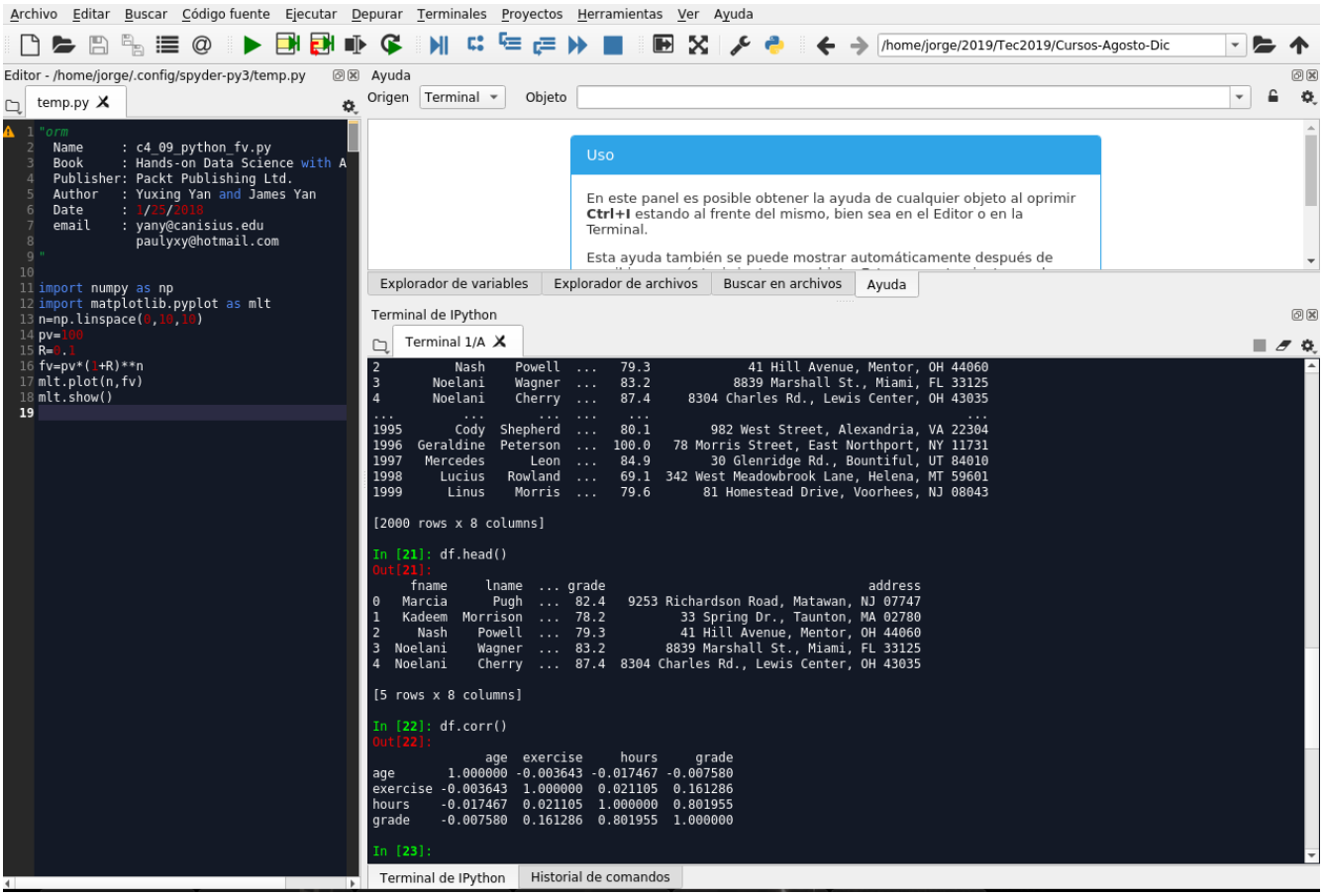
>>> ¿Cuántos renglones y columnas tiene gradedata.csv?

>>> df



Correlación entre variables (fácil!!!) librería pandas!!!

>>> df.corr()



Regresión lineal (más de una variable)

Usaremos además de pandas, la librería statsmodels.

>>> import statsmodels.formula.api as sm

>>> result = sm.ols(

formula='grade ~ age + exercise + hours',

data=df).fit()

>>> result.summary()

Terminal 1/A

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  grade   R-squared:                       0.664
Model:                            OLS   Adj. R-squared:                  0.664
Method:                 Least Squares   F-statistic:                     1315.
Date:                Wed, 25 Sep 2019   Prob (F-statistic):               0.00
Time:                        20:40:37   Log-Likelihood:                -6300.7
No. Observations:                2000   AIC:                         1.261e+04
Df Residuals:                    1996   BIC:                         1.263e+04
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     57.8704      1.321     43.804      0.000      55.279      60.461
age            0.0397      0.075      0.532      0.595      -0.107       0.186
exercise       0.9893      0.089     11.131      0.000       0.815       1.164
hours          1.9165      0.031     61.564      0.000       1.855       1.978
==============================================================================
Omnibus:                      321.187   Durbin-Watson:                   2.047
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2196.187
Skew:                          -0.567   Prob(JB):                         0.00
Kurtosis:                       8.007   Cond. No.                         213.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
"""
```

Resumen de los datos:

el R -cuadrado representa el porcentaje de la variación en los datos que pueden ser explicado por la regresión, 0.664, o 66.4% por ciento, es bueno, pero no excelente.
El valor p (representado aquí por el valor de P> | t |) representa la probabilidad de que la variable independiente **no tenga efecto en la dependiente variable.**

Notar que ejercicios y horas de estudio tienen un gran efecto en la obtención de grado!!!.

Veremos más detalles en el curso de esta tabla llena de información

Y eso es todo por hoy

¿Alguna duda?

Sigamos con el problema de la Multicolinealidad en Regresión lineal Multiple.

Usemos nuestra base de datos ya conocida “Problema3.csv”

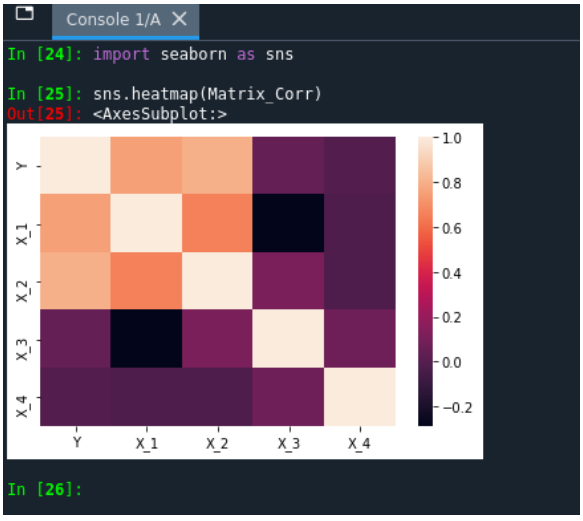
df= pd.read_csv(‘Problema3.csv’)

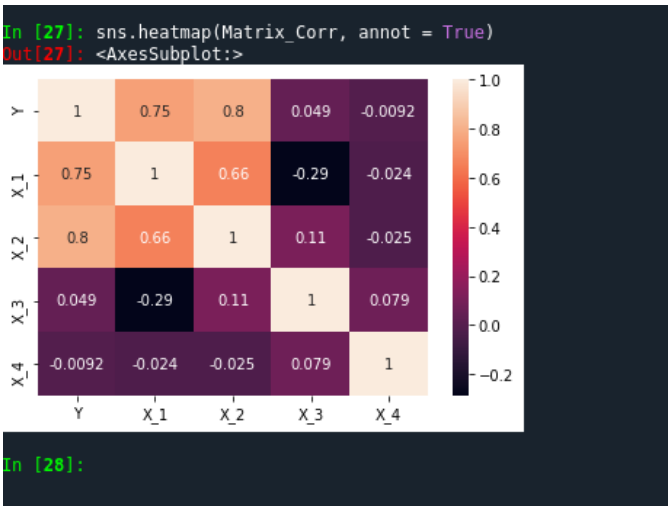
Matrix_Corr= df.corr()

```
In [18]: Matrix_Corr
Out[18]:
              Y       X_1       X_2       X_3       X_4
Y     1.000000  0.751479  0.802857  0.048848 -0.009197
X_1   0.751479  1.000000  0.660456 -0.287566 -0.023559
X_2   0.802857  0.660456  1.000000  0.112739 -0.025328
X_3   0.048848 -0.287566  0.112739  1.000000  0.078914
X_4  -0.009197 -0.023559 -0.025328  0.078914  1.000000

In [19]: type(Matrix_Corr)
Out[19]: pandas.core.frame.DataFrame

In [20]:
```

Notemos que los índices del dataframe Matrix_Corr son los nombres de las columnas.

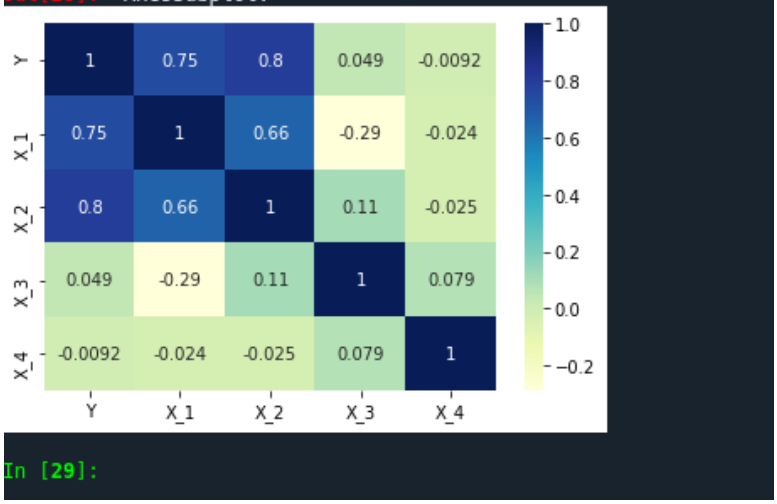
Usemos ahora la librería seaborn y su método heatmap para ver “mejor” la matriz de correlación





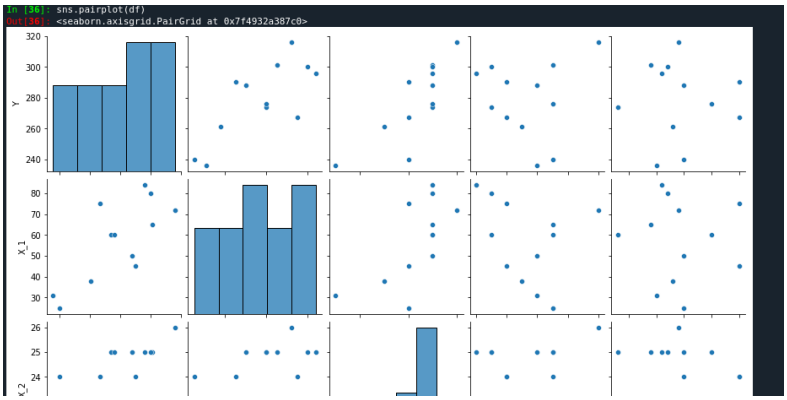
Cambiemos la paleta de Colores en heatmap.

```
In [28]: sns.heatmap(Matrix_Corr, annot = True, cmap="YlGnBu")
Out[28]: <AxesSubplot:>
```

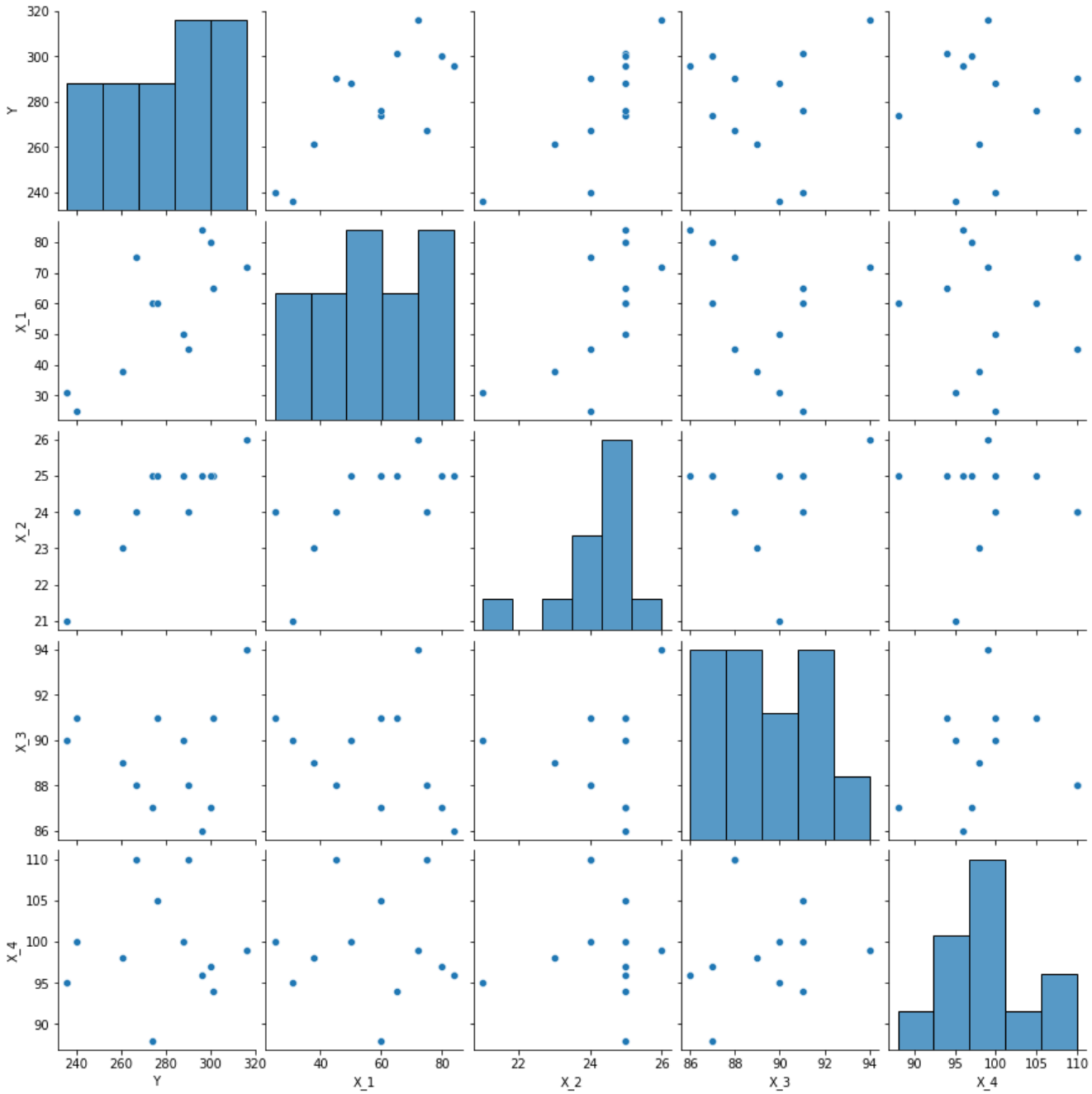



```
In [29]:
```

> > > sns.pairplot(df)



Resultado Final:



Resumen de Regresión lineal:

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

In [2]: import statsmodels.formula.api as sm

In [3]: df = pd.read_csv("Problema3.csv")

In [4]: df
Out[4]:
      Y  X_1  X_2  X_3  X_4
0   240   25   24   91  100
1   236   31   21   90   95
2   290   45   24   88  110
3   274   60   25   87   88
4   301   65   25   91   94
5   316   72   26   94   99
6   300   80   25   87   97
7   296   84   25   86   96
8   267   75   24   88  110
9   276   60   25   91  105
10  288   50   25   90  100
11  261   38   23   89   98

In [5]: l="Y ~ X_1 + X_2 + X_3 + X_4"

In [6]: l
Out[6]: 'Y ~ X_1 + X_2 + X_3 + X_4'

In [7]: resultado = sm.ols(formula=l, data=df).fit()
```

Console 1/A

```
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                      Y   R-squared:                       0.745
Model:                            OLS   Adj. R-squared:                  0.599
Method:                 Least Squares   F-statistic:                     5.106
Date:                Wed, 29 Sep 2021   Prob (F-statistic):             0.0303
Time:                        18:51:31   Log-Likelihood:                -46.745
No. Observations:                  12   AIC:                             103.5
Df Residuals:                       7   BIC:                             105.9
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept   -102.7132    207.859     -0.494      0.636    -594.221     388.795
X_1            0.6054      0.369      1.641      0.145      -0.267       1.478
X_2            8.9236      5.301      1.684      0.136      -3.610      21.457
X_3            1.4375      2.392      0.601      0.567      -4.218       7.093
X_4            0.0136      0.734      0.019      0.986      -1.722       1.749
==============================================================================
Omnibus:                        0.007   Durbin-Watson:                   1.772
Prob(Omnibus):                  0.996   Jarque-Bera (JB):                0.207
Skew:                          -0.039   Prob(JB):                        0.902
Kurtosis:                       2.362   Cond. No.                     6.82e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 6.82e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
"""

In [9]:
```

¿Que ocurre si quitamos X_1 y asumimos que la constante es cero?

```
In [9]: l="Y ~ X_1 + X_3 + X_4 -1"

In [10]: resultado = sm.ols(formula=l, data=df).fit()

In [11]: resultado.summary()
/home/jorge/.local/lib/python3.8/site-packages/scipy/stats/stats.py:1603: UserWarning: kurtosistest only valid for n>=20 ... c
n=12
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
Out[11]:
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                      Y   R-squared (uncentered):                   0.997
Model:                            OLS   Adj. R-squared (uncentered):              0.997
Method:                 Least Squares   F-statistic:                              1168.
Date:                Wed, 29 Sep 2021   Prob (F-statistic):                    5.63e-12
Time:                        20:37:29   Log-Likelihood:                         -48.832
No. Observations:                  12   AIC:                                      103.7
Df Residuals:                       9   BIC:                                      105.1
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
X_1            1.0339      0.249      4.144      0.003       0.470       1.598
X_3            2.5770      0.820      3.141      0.012       0.721       4.433
X_4           -0.1052      0.738     -0.143      0.890      -1.774       1.564
==============================================================================
Omnibus:                        0.140   Durbin-Watson:                   2.061
Prob(Omnibus):                  0.932   Jarque-Bera (JB):                0.087
Skew:                           0.092   Prob(JB):                        0.957
Kurtosis:                       2.625   Cond. No.                         33.8
```

IPython console | History

Últimas construcciones:

```
In [72]: df2
Out[72]:
   gestage
0       25
1       29
2       33

In [73]: R=modelo.predict(df2)

In [74]: R
Out[74]:
0    23.415593
1    26.535806
2    29.656018
dtype: float64

In [75]: H=pd.concat([df2,R], axis=1)

In [76]: H
Out[76]:
   gestage          0
0       25  23.415593
1       29  26.535806
2       33  29.656018

In [77]: H.columns=["gestage", "Pronostico"]

In [78]: H
Out[78]:
   gestage  Pronostico
0       25   23.415593
1       29   26.535806
2       33   29.656018

In [79]:
```

Sigamos con más análisis de datos

Ejemplo muy interesante: Los Datos corresponden a mediciones de 100 niños nacidos con bajo peso (es decir, menos de 1500gr.) en Boston Massachusetts. Para dichos bebés se miden varias variables. La variable que nos interesa es el perímetro cefálico al nacer (medido en cm.). Los datos están en el archivo **low_birth_weight_infants.txt**, la variable **headcirc** es la que contiene los datos del perímetro cefálico.

Asumamos que entra una mamá con su bebé recién nacido al consultorio de niños de bajo peso y quiero predecir su perímetro cefálico con la información de la muestra de los 100 niños.

Hagamos un poco de análisis de datos con lo que sabemos

import pandas as pd
df=pd.read_csv("**low_birth_weight_infants.txt**" , sep="\s+")
df.info()

```
In [7]: import pandas as pd

In [8]: df=pd.read_csv("low_birth_weight_infants.txt", sep="\s+")

In [9]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   headcirc  100 non-null    int64
 1   length    100 non-null    int64
 2   gestage   100 non-null    int64
 3   birthwt   100 non-null    int64
 4   momage    100 non-null    int64
 5   toxemia   100 non-null    int64
dtypes: int64(6)
memory usage: 4.8 KB

In [10]:
```

Definamos Y = perímetro cefálico (en cm.) de un bebé recién nacido con bajo peso. Estamos interesados en la media poblacional $\mathrm{E}(\mathrm{Y})=\mu.$ Sabemos que la media muestral $\bar{Y}$ es el mejor estimador que podamos dar para la media poblacional, veamos su valor:

```
In [10]: df.describe()
Out[10]:
         headcirc      length     gestage      birthwt      momage     toxemia
count  100.000000  100.000000  100.00000   100.000000  100.000000  100.00000
mean    26.450000   36.820000   28.89000  1098.850000   27.730000    0.21000
std      2.532117    3.571435    2.53419   269.993317    5.982896    0.40936
min     21.000000   20.000000   23.00000   560.000000   14.000000    0.00000
25%     25.000000   35.000000   27.00000   880.000000   23.000000    0.00000
50%     27.000000   38.000000   29.00000  1155.000000   28.000000    0.00000
75%     28.000000   39.000000   31.00000  1326.250000   32.000000    0.00000
max     35.000000   43.000000   35.00000  1490.000000   41.000000    1.00000

In [11]:
```

Tenemos así une media muestral igual a 26.45 cm con una desviación muestral de 2.53.

Podemos construir intervalos de confianza del 95% para la media poblacional de Y, usando la muestras.

¿Que necesitamos?, pues las siguientes librerías

```
import numpy as np
import scipy.stats as st
a=df["headcirc"]
st.t.interval(0.95, len(a)-1, loc=np.mean(a), scale=st.sem(a))
```

```
In [8]: import numpy as np

In [9]: import scipy.stats as st

In [10]: a=df["headcirc"]

In [11]: st.t.interval(0.95, len(a)-1, loc=np.mean(a), scale=st.sem(a))
Out[11]: (25.94757306586406, 26.95242693413594)

In [12]: st.t.interval(0.99, len(a)-1, loc=np.mean(a), scale=st.sem(a))
Out[12]: (25.784963427136237, 27.11503657286376)

In [13]:
```

Pero tenemos información adicional para hacer una mejor predicción sobre el perímetro cefálico.

```
In [13]: df.corr()
Out[13]:
          headcirc    length   gestage   birthwt    momage   toxemia
headcirc  1.000000  0.712733  0.780692  0.798837  0.132119  0.132043
length    0.712733  1.000000  0.675231  0.815552  0.217993  0.109024
gestage   0.780692  0.675231  1.000000  0.659938  0.265840  0.411968
birthwt   0.798837  0.815552  0.659938  1.000000  0.154572  0.011803
momage    0.132119  0.217993  0.265840  0.154572  1.000000  0.114119
toxemia   0.132043  0.109024  0.411968  0.011803  0.114119  1.000000

In [14]:
```

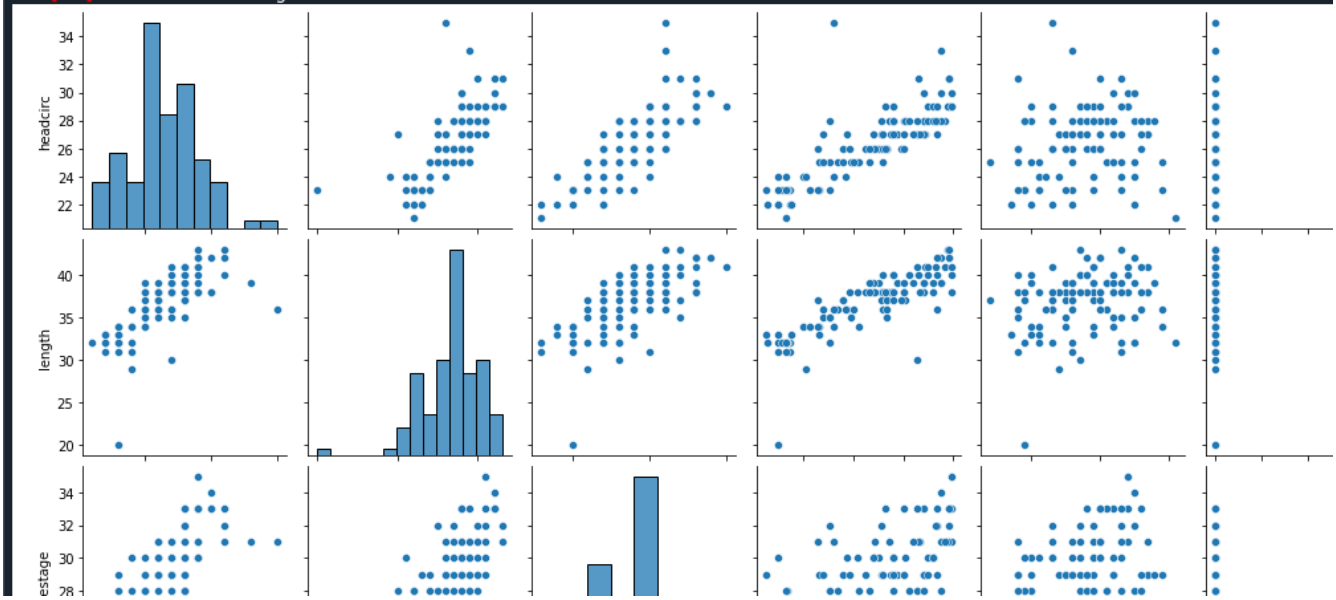
La variable **gestage** (**edad gestacional**, es decir, duración del embarazo medido en semanas).

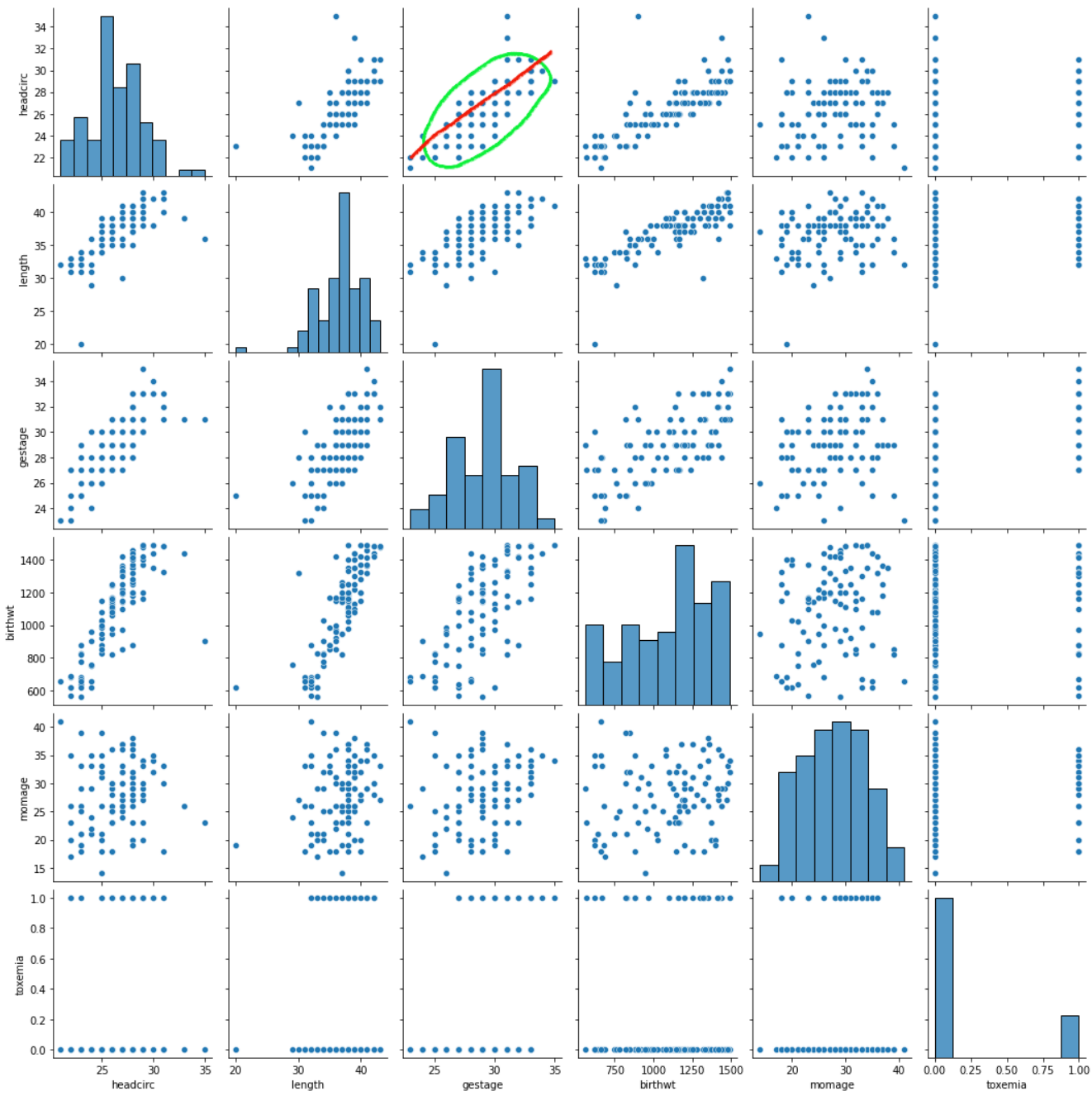
Objetivo: Ver si podemos predecir de una mejor manera el perímetro cefálico de un bebé al nacer si conocemos su edad gestacional.

Veamos como se ve todo esto usando la librería seaborn

import seaborn as sms

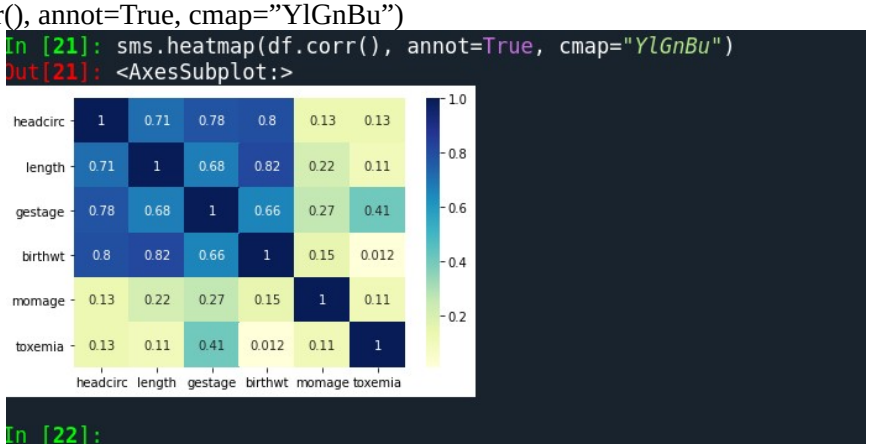
```
In [19]: import seaborn as sms

In [20]: sms.pairplot(df)
Out[20]: <seaborn.axisgrid.PairGrid at 0x7fbbae872f40>
```





Recordado la función heatmap de seaborn

sms.heamap(df.corr(), annot=True, cmap="YlGnBu")



Construyendo el modelo de regresión lineal:

```
In [31]: import statsmodels.formula.api as sm

In [32]: resultado=sm.ols(formula="headcirc ~ gestage", data=df).fit()

In [33]: resultado.summary()
Out[33]:
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                headcirc   R-squared:                       0.609
Model:                            OLS   Adj. R-squared:                  0.605
Method:                 Least Squares   F-statistic:                     152.9
Date:                Wed, 06 Oct 2021   Prob (F-statistic):           1.00e-21
Time:                        17:07:03   Log-Likelihood:                -187.28
No. Observations:                 100   AIC:                             378.6
Df Residuals:                      98   BIC:                             383.8
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      3.9143      1.829      2.140      0.035       0.284       7.544
gestage        0.7801      0.063     12.367      0.000       0.655       0.905
==============================================================================
Omnibus:                       23.475   Durbin-Watson:                   2.037
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               53.083
Skew:                           0.849   Prob(JB):                     2.97e-12
Kurtosis:                       6.140   Cond. No.                         334.
==============================================================================
```

Podemos asumir que la constate es cero en el modelo:

```
In [34]: resultado=sm.ols(formula="headcirc ~ gestage -1", data=df).fit()

In [35]: resultado.summary()
Out[35]:
<class 'statsmodels.iolib.summary.Summary'>
"""
                                 OLS Regression Results
=======================================================================================
Dep. Variable:                headcirc   R-squared (uncentered):                   0.996
Model:                            OLS   Adj. R-squared (uncentered):              0.996
Method:                 Least Squares   F-statistic:                          2.684e+04
Date:                Wed, 06 Oct 2021   Prob (F-statistic):                    2.43e-122
Time:                        17:11:53   Log-Likelihood:                         -189.57
No. Observations:                 100   AIC:                                      381.1
Df Residuals:                      99   BIC:                                      383.7
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
gestage        0.9145      0.006    163.819      0.000       0.903       0.926
==============================================================================
Omnibus:                       14.055   Durbin-Watson:                   2.044
Prob(Omnibus):                  0.001   Jarque-Bera (JB):               23.631
Skew:                           0.581   Prob(JB):                     7.39e-06
Kurtosis:                       5.079   Cond. No.                         1.00
==============================================================================
```

Nota: Si solo queremos ver los valor de los coeficientes:

resultado.params

```
In [38]: resultado.params
Out[38]:
gestage    0.914517
dtype: float64

In [39]: resultado=sm.ols(formula="headcirc ~ gestage", data=df).fit()

In [40]: resultado.params
Out[40]:
Intercept    3.914264
gestage      0.780053
dtype: float64
```

Para ver las predicciones:

resultado.predict()

```
In [46]: resultado.predict()
Out[46]:
array([26.53580585, 28.09591217, 29.6560185 , 28.09591217, 27.31585901,
       23.4155932 , 24.97569952, 26.53580585, 25.75575269, 26.53580585,
       24.19564636, 27.31585901, 26.53580585, 26.53580585, 26.53580585,
       26.53580585, 26.53580585, 29.6560185 , 29.6560185 , 26.53580585,
       25.75575269, 27.31585901, 24.97569952, 29.6560185 , 28.87596533,
       25.75575269, 26.53580585, 25.75575269, 26.53580585, 27.31585901,
       28.09591217, 27.31585901, 28.09591217, 26.53580585, 24.97569952,
       24.97569952, 24.97569952, 28.87596533, 28.09591217, 25.75575269,
       27.31585901, 26.53580585, 25.75575269, 28.09591217, 24.97569952,
       23.4155932 , 27.31585901, 25.75575269, 25.75575269, 23.4155932 ,
       21.85548687, 24.97569952, 25.75575269, 24.97569952, 24.97569952,
       24.19564636, 23.4155932 , 21.85548687, 24.19564636, 22.63554004,
       26.53580585, 26.53580585, 24.97569952, 27.31585901, 27.31585901,
       28.87596533, 29.6560185 , 24.97569952, 28.09591217, 24.19564636,
       24.97569952, 24.97569952, 31.21612482, 25.75575269, 27.31585901,
       28.09591217, 27.31585901, 24.97569952, 23.4155932 , 23.4155932 ,
       24.19564636, 26.53580585, 26.53580585, 30.43607166, 27.31585901,
       26.53580585, 29.6560185 , 27.31585901, 26.53580585, 22.63554004,
       29.6560185 , 23.4155932 , 28.87596533, 28.09591217, 28.09591217,
       28.09591217, 26.53580585, 28.87596533, 29.6560185 , 25.75575269])
```

¿Como calculamos los errores?

Queremos ahora hacer predicciones para 25, 29 y 33 semanas de gestación.

¿Como hacemos esto?, !Fácil!

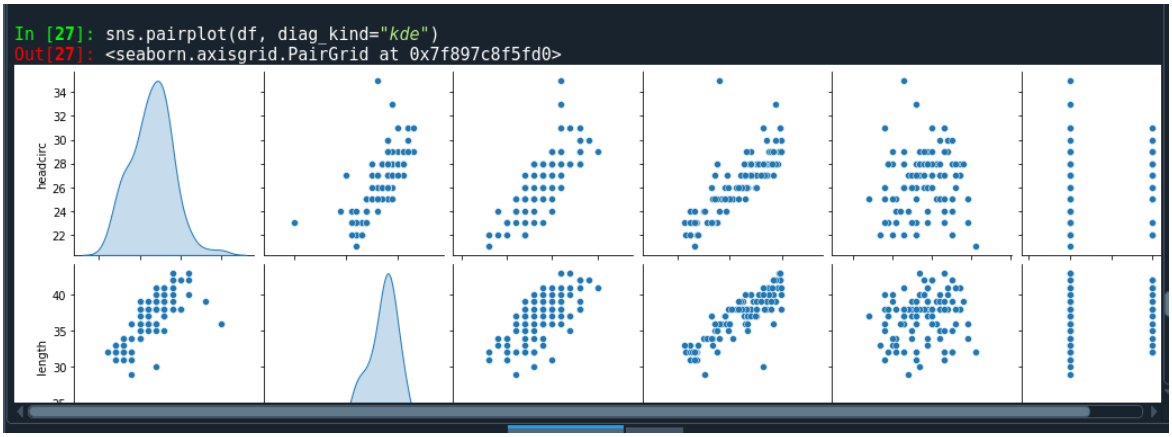
resultado.predict(pd.DataFrame({“gestage”:[25,29,33]}))

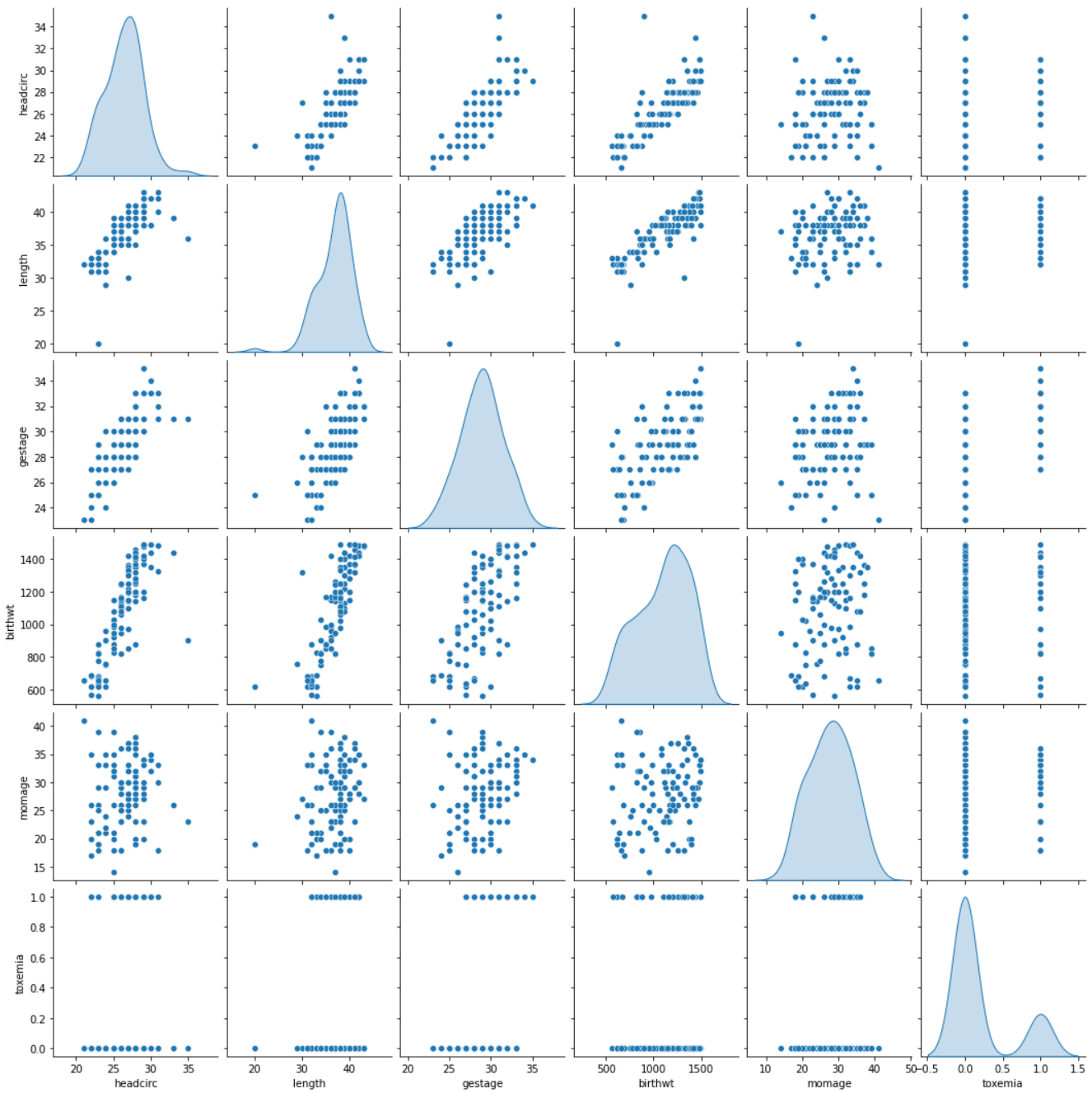
```
In [97]: resultado.predict(pd.DataFrame({"gestage":[25,29,33]}))
Out[97]:
0    23.415593
1    26.535806
2    29.656018
dtype: float64

In [98]:
```

Nota: Otra forma de hacerlo

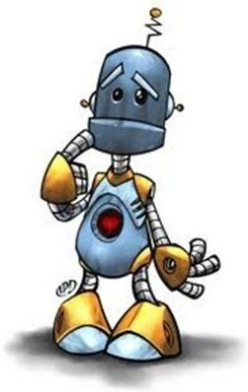
```
In [99]: Nuevos_Valores = dict(gestage=[25,29,33])

In [100]: type(Nuevos_Valores)
Out[100]: dict

In [101]: resultado.predict(pd.DataFrame(Nuevos_Valores))
Out[101]:
0    23.415593
1    26.535806
2    29.656018
dtype: float64

In [102]:
```

Un gráfico más muy interesante





¿Alguna duda?



Muliticolinealidad y Factor de Inflación de la Varianza (VIF)

- *La multicolinealidad se refiere a una alta correlación en dos o más variables independientes en el modelo de regresión.*

- *El factor de inflación de la varianza **(VIF)** mide el grado de multicolinealidad en el modelo de regresión.*

VIF está relacionado con el R-cuadrado, y su fórmula es

$$\mathrm{VIF}_i = \frac{1}{1 - R_i^2}$$

Donde R_i es el coeficiente de correlación multiple entre las X_i y el resto de las variables independientes.

Diagnóstico de multicolinealidad mediante el factor de inflación de la varianza (VIF)

- VIF, índices de tolerancia (TI) y coeficientes de correlación son métricas útiles para la detección de multicolinealidad**.**
- El rango de VIF para evaluar la multicolinealidad se da como

Valor VIF	Diagnostico
1	Ausencia total de multicolinealidad
(1 2]	Ausencia de una fuerte multicolinealidad
> 2	Presencia de multicolinealidad moderada a fuerte

- VIF puede detectar multicolinealidad, **pero no identifica variables independientes** que están causando multicolinealidad. Aquí, el análisis de correlación es útil para detectar variables independientes altamente correlacionadas

Ejemplo de diagnóstico y correción de la multicolinealidad, utilizando datos de presión arterial.

Datos:
https://reneshbedre.github.io/assets/posts/reg/bp.csv.

```
import pandas as pd
import statsmodels.formula.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
df = pd.read_csv("https://reneshbedre.github.io/assets/posts/reg/bp.csv")
```

df.info()

Veamos el calculo del VIF para algunas variables independientes, Age, Weight y BSA.

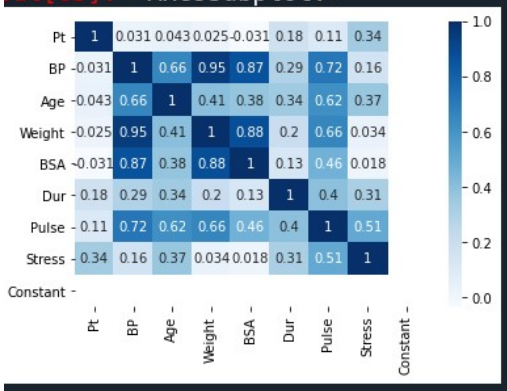
```
Type "copyright", "credits" or "license" for more information.

IPython 7.27.0 -- An enhanced Interactive Python.

In [1]: import pandas as pd

In [2]: import statsmodels.formula.api as sm

In [3]: from statsmodels.stats.outliers_influence import variance_inflation_factor

In [4]: df = pd.read_csv("https://reneshbedre.github.io/assets/posts/reg/bp.csv")

In [5]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 8 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Pt      20 non-null     int64
 1   BP      20 non-null     int64
 2   Age     20 non-null     int64
 3   Weight  20 non-null     float64
 4   BSA     20 non-null     float64
 5   Dur     20 non-null     float64
 6   Pulse   20 non-null     int64
 7   Stress  20 non-null     int64
dtypes: float64(3), int64(5)
memory usage: 1.4 KB

In [6]:
```

```
Console 1/A
Type "copyright", "credits" or "license" for more information.

IPython 7.27.0 -- An enhanced Interactive Python.

In [1]: import pandas as pd

In [2]: import statsmodels.formula.api as sm

In [3]: df = pd.read_csv("https://reneshbedre.github.io/assets/posts/reg/bp.csv")

In [4]: modelo=sm.ols(formula="Age ~ Weight + BSA + Dur + Pulse + Stress ", data=df).fit()

In [5]: modelo.rsquared
Out[5]: 0.43272283475439766

In [6]: modelo.summary()
Out[6]:
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                    Age   R-squared:                       0.433
Model:                            OLS   Adj. R-squared:                  0.230
Method:                 Least Squares   F-statistic:                     2.136
Date:                Wed, 13 Oct 2021   Prob (F-statistic):              0.121
Time:                        18:38:49   Log-Likelihood:                -40.527
No. Observations:                  20   AIC:                             93.05
Df Residuals:                      14   BIC:                             99.03
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     25.0441     12.039      2.080      0.056      -0.777      50.865
Weight        -0.2341      0.334     -0.700      0.495      -0.951       0.483
BSA            7.7062      8.260      0.933      0.367     -10.011      25.423
Dur            0.1266      0.259      0.489      0.632      -0.428       0.682
Pulse          0.4177      0.255      1.640      0.123      -0.129       0.964
Stress         0.0013      0.018      0.069      0.946      -0.038       0.041
==============================================================================
Omnibus:                        2.658   Durbin-Watson:                   2.198
Prob(Omnibus):                  0.265   Jarque-Bera (JB):                1.008
Skew:                           0.221   Prob(JB):                        0.604
Kurtosis:                       4.007   Cond. No.                     3.28e+03
==============================================================================
```

```
In [7]: VIF = 1/(1-0.433)

In [8]: VIF
Out[8]: 1.7636684303350971

In [9]: modelo=sm.ols(formula="Weight ~ Age + BSA + Dur + Pulse + Stress ", data=df).fit()

In [10]: 1/(1-modelo.rsquared)
Out[10]: 8.41703502963307

In [11]: modelo=sm.ols(formula="BSA ~ Weight + Age  + Dur + Pulse + Stress ", data=df).fit()

In [12]: 1/(1-modelo.rsquared)
Out[12]: 5.328751470118906

In [13]:
```

BP = blood Pressure
(Presión arterial)

```
In [64]: import seaborn as sns

In [65]: sns.heatmap(df.corr(), annot=True, cmap="Blues")
Out[65]: <AxesSubplot:>
```



Ahora, definamos el dataframe X que solo contenga a las variables independientes

X= `df[['Age', 'Weight', 'BSA', 'Dur', 'Pulse', 'Stress']]`

```
In [6]: X = df[['Age', 'Weight', 'BSA', 'Dur', 'Pulse', 'Stress']]

In [7]: X
Out[7]:
    Age  Weight   BSA   Dur  Pulse  Stress
0    47    85.4  1.75   5.1     63      33
1    49    94.2  2.10   3.8     70      14
2    49    95.3  1.98   8.2     72      10
3    50    94.7  2.01   5.8     73      99
4    51    89.4  1.89   7.0     72      95
5    48    99.5  2.25   9.3     71      10
6    49    99.8  2.25   2.5     69      42
7    47    90.9  1.90   6.2     66       8
8    49    89.2  1.83   7.1     69      62
9    48    92.7  2.07   5.6     64      35
10   47    94.4  2.07   5.3     74      90
11   49    94.1  1.98   5.6     71      21
12   50    91.6  2.05  10.2     68      47
13   45    87.1  1.92   5.6     67      80
14   52   101.3  2.19  10.0     76      98
15   46    94.5  1.98   7.4     69      95
16   46    87.0  1.87   3.6     62      18
17   46    94.5  1.90   4.3     70      12
18   48    90.5  1.88   9.0     71      99
19   56    95.7  2.09   7.0     75      99

In [8]: df4
```

Definimos una nueva columna en df

df[“Constant”]=1

```
In [8]: df["Constant"]=1

In [9]: df
Out[9]:
    Pt   BP  Age  Weight   BSA   Dur  Pulse  Stress  Constant
0    1  105   47    85.4  1.75   5.1     63      33         1
1    2  115   49    94.2  2.10   3.8     70      14         1
2    3  116   49    95.3  1.98   8.2     72      10         1
3    4  117   50    94.7  2.01   5.8     73      99         1
4    5  112   51    89.4  1.89   7.0     72      95         1
5    6  121   48    99.5  2.25   9.3     71      10         1
6    7  121   49    99.8  2.25   2.5     69      42         1
7    8  110   47    90.9  1.90   6.2     66       8         1
8    9  110   49    89.2  1.83   7.1     69      62         1
9   10  114   48    92.7  2.07   5.6     64      35         1
10  11  114   47    94.4  2.07   5.3     74      90         1
11  12  115   49    94.1  1.98   5.6     71      21         1
12  13  114   50    91.6  2.05  10.2     68      47         1
13  14  106   45    87.1  1.92   5.6     67      80         1
14  15  125   52   101.3  2.19  10.0     76      98         1
15  16  114   46    94.5  1.98   7.4     69      95         1
16  17  106   46    87.0  1.87   3.6     62      18         1
17  18  113   46    94.5  1.90   4.3     70      12         1
18  19  110   48    90.5  1.88   9.0     71      99         1
19  20  122   56    95.7  2.09   7.0     75      99         1

In [10]:
```

Constante = df[“Constant”]

```
In [19]: Constante=df["Constant"]

In [20]: dfX=pd.concat([Constante, X], axis=1)

In [21]: dfX
Out[21]:
    Constant  Age  Weight   BSA   Dur  Pulse  Stress
0          1   47    85.4  1.75   5.1     63      33
1          1   49    94.2  2.10   3.8     70      14
2          1   49    95.3  1.98   8.2     72      10
3          1   50    94.7  2.01   5.8     73      99
4          1   51    89.4  1.89   7.0     72      95
5          1   48    99.5  2.25   9.3     71      10
6          1   49    99.8  2.25   2.5     69      42
7          1   47    90.9  1.90   6.2     66       8
8          1   49    89.2  1.83   7.1     69      62
9          1   48    92.7  2.07   5.6     64      35
10         1   47    94.4  2.07   5.3     74      90
11         1   49    94.1  1.98   5.6     71      21
12         1   50    91.6  2.05  10.2     68      47
13         1   45    87.1  1.92   5.6     67      80
14         1   52   101.3  2.19  10.0     76      98
15         1   46    94.5  1.98   7.4     69      95
16         1   46    87.0  1.87   3.6     62      18
17         1   46    94.5  1.90   4.3     70      12
18         1   48    90.5  1.88   9.0     71      99
19         1   56    95.7  2.09   7.0     75      99

In [22]:
```

Estamos listos para determinar el VIF de las variables independientes Age, Weight, BSA Dur, Pulse y Stress, usando la función **variance_inflation_factor**

```
In [22]: variance_inflation_factor(dfX.values, 1)
Out[22]: 1.7628067217672165

In [23]: variance_inflation_factor(dfX.values, 2)
Out[23]: 8.417035029633047

In [24]: variance_inflation_factor(dfX.values, 3)
Out[24]: 5.328751470118887

In [25]: variance_inflation_factor(dfX.values, 4)
Out[25]: 1.2373094205198356

In [26]: variance_inflation_factor(dfX.values, 5)
Out[26]: 4.4135751655972895

In [27]: variance_inflation_factor(dfX.values, 6)
Out[27]: 1.8348453242645892

In [28]:
```

Para **Age:**

variance_inflation_factor(dfX.values, 1)
VIF = 1.7628067217672165

Weight:
variance_inflation_factor(dfX.values, 2)
VIF = 8.417035029633047

BSA:
variance_inflation_factor(dfX.values, 3)
VIF = 5.328751470118887

Dur:
variance_inflation_factor(dfX.values, 4)
VIF =1.2373094205198356

Pulse:
variance_inflation_factor(dfX.values, 5)
VIF =4.4135751655972895

Stress:
variance_inflation_factor(dfX.values, 6)
VIF =1.8348453242645892

variables	VIF
Age	1.762807
Weight	8.417035
BSA	5.328751
Dur	1.237309
Pulse	4.413575
Stress	1.834845

Lo anterior lo podemos hacer por medio de un **ciclo for**

for i in range(6):

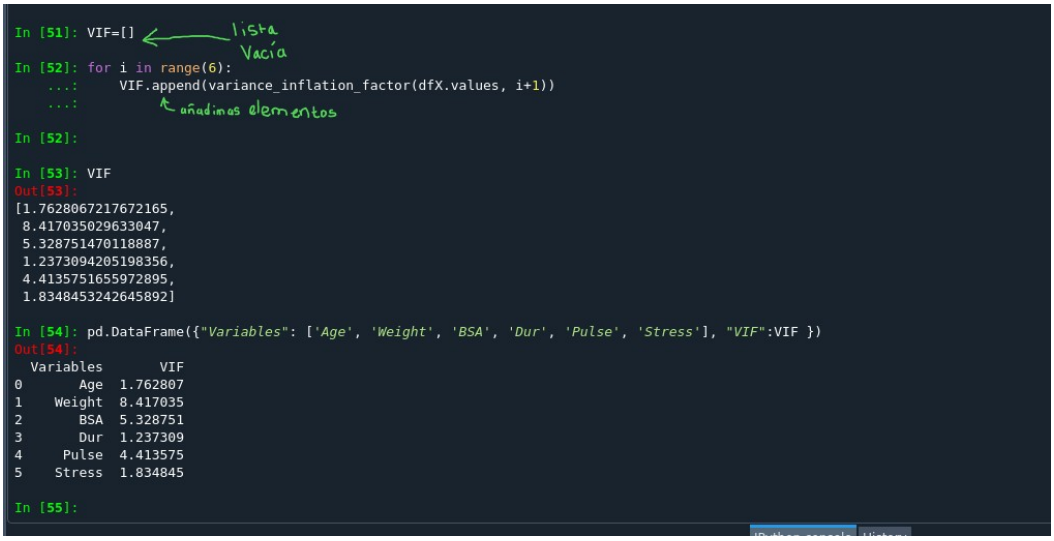
print(variance_inflation_factor(dfX.values, i+1))

```
In [34]: for i in range(6):
    ...:     print(variance_inflation_factor(dfX.values, i+1))
    ...:
1.7628067217672165
8.417035029633047
5.328751470118887
1.2373094205198356
4.4135751655972895
1.8348453242645892

In [35]:
```

Más bonito (pero algo obscuro)

```
In [36]: pd.DataFrame({"Variables": ['Age', 'Weight', 'BSA', 'Dur', 'Pulse', 'Stress'], "VIF": [variance_inflation_factor(dfX.values, i+1)  for i in range(6)]})
Out[36]:
  Variables       VIF
0       Age  1.762807
1    Weight  8.417035
2       BSA  5.328751
3       Dur  1.237309
4     Pulse  4.413575
5    Stress  1.834845

In [37]:
```

Para mi, es mejor lo siguiente



Comprobando supuestos del error

Regresemos al caso perímetro cefálico:

```
import pandas as pd
df=pd.read_csv("low_birth_weight_infants.txt" , sep="\s+")
df.info()
```

```
Console 1/A

In [3]: df=pd.read_csv("low_birth_weight_infants.txt" , sep="\s+")

In [4]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   headcirc  100 non-null    int64
 1   length    100 non-null    int64
 2   gestage   100 non-null    int64
 3   birthwt   100 non-null    int64
 4   momage    100 non-null    int64
 5   toxemia   100 non-null    int64
dtypes: int64(6)
memory usage: 4.8 KB

In [5]: import statsmodels.formula.api as sm

In [6]: modelo=sm.ols(formula="headcirc ~ gestage + birthwt -1 ", data=df).fit()

In [7]: modelo.resid
Out[7]:
0    -0.389228
1    -0.403938
2    -0.966974
3    -0.326869
4     1.385161
        ...
95   -1.076079
96    0.340400
97   -0.066062
98   -2.376323
99    0.365127
Length: 100, dtype: float64

In [8]:
```

1. Los residuos tienes media cero. **OK!**

```
In [8]: residuos=modelo.resid

In [9]: residuos.describe()
Out[9]:
count    100.000000
mean       0.054083
std        1.428649
min       -3.530010
25%       -0.699804
50%        0.082480
75%        0.804369
max        7.645968
dtype: float64

In [10]: residuos.mean()
Out[10]: 0.0540830448910507

In [11]:
```

2. Normalidad de los errores

Veamos Histograma

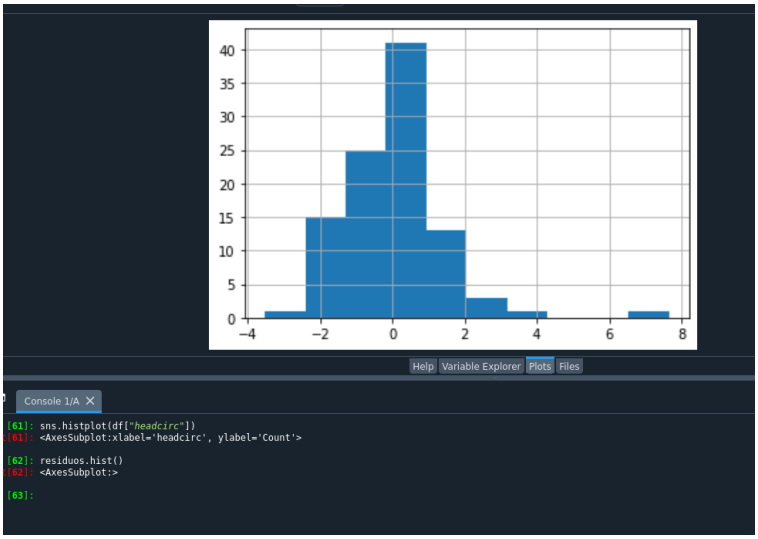
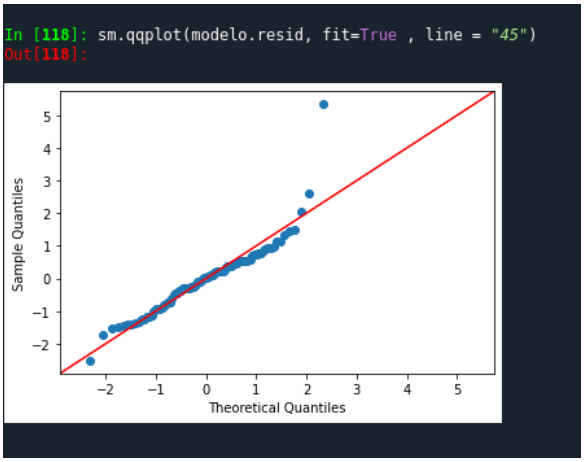


Gráfico QQ: Este gráfico es útil para determinar si los residuos siguen una distribución normal. Si los valores de los datos en el gráfico caen a lo largo de una línea aproximadamente recta en un ángulo de 45 grados, entonces los datos se distribuyen normalmente:

```
import statsmodels.api as sm  (Nota: Aquí se pierde la parte formula)

# Gráfico QQ
 sm.qqplot(modelo.resid, fit=True , line="45")
```



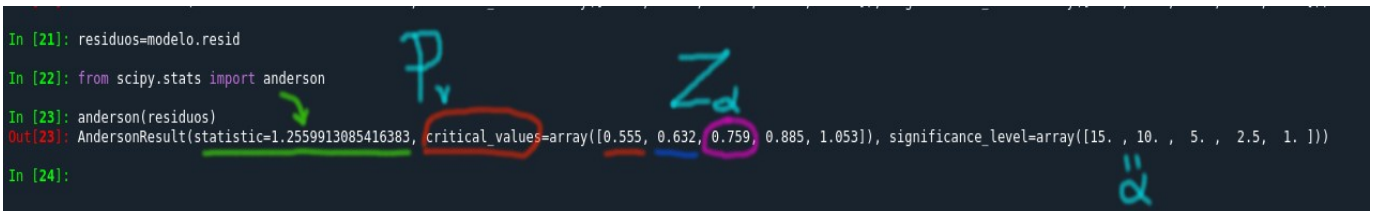
Método estadístico para detectar Normalidad:

Se utilizan las siguientes hipótesis nula H_0 y alternativa H_1.
H_0 : **Los datos se ajustan a una distribución normal**.

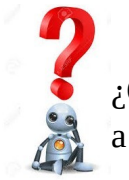
H_1 : **Los datos no se ajustan a una distribución normal**

Anderson-Darling Test:

```
#perform Anderson-Darling Test
from scipy.stats import anderson
residuos=modelo.resid
anderon(residuos)
```



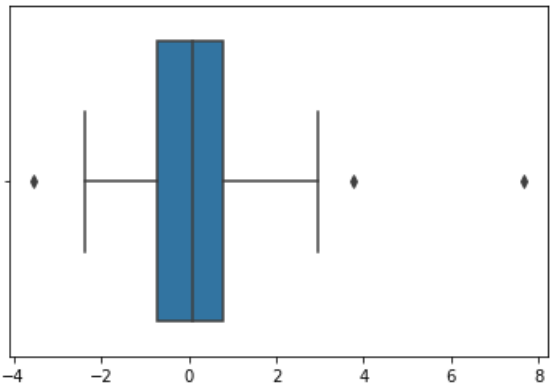
Sorpresa, la prueba nos dice que tenemos que rechazar hipótesis nula, es decir que los errores no se ajusta a una distribución normal, ya que el estadístico=1.255 es , mayor que cualquier valor critico.

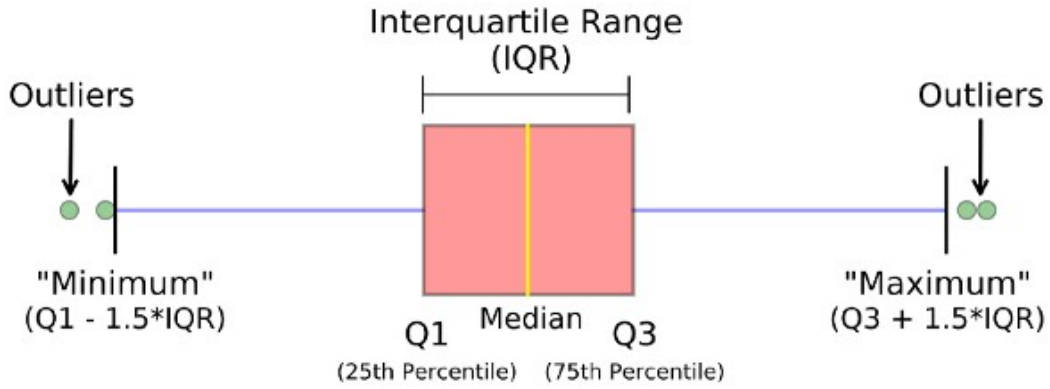


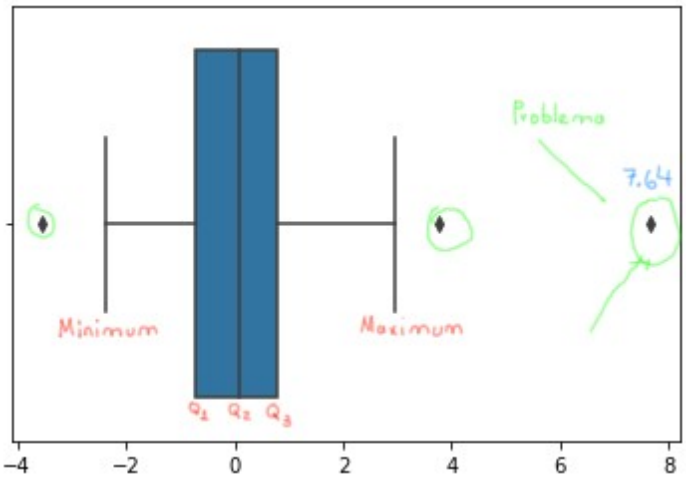
¿Que ocurre aquí?, los métodos gráficos nos dan evidencia de que los errores se ajustan a una distribución normal. ¡Veamos los datos atípicos!.

sns.boxplot(residuos)

sns.boxplot(modelo.resid)








```
In [124]: residuos.describe()
Out[124]:
count    100.000000
mean       0.054083
std        1.428649
min       -3.530010
25%       -0.699804
50%        0.082480
75%        0.804369
max        7.645968
dtype: float64

In [125]: residuos[30]
Out[125]: 7.645968091289099
```

```
In [23]: modelo.resid[30]
Out[23]: 7.645968091289099

In [24]: df.iloc[30]
Out[24]:
headcirc     35
length       36
gestage      31
birthwt     900
momage       23
toxemia       0
Name: 30, dtype: int64

In [25]: modelo.predict(df.iloc[30]) -df.iloc[30][0]
Out[25]:
30   -7.645968
dtype: float64
```

Quitemos solo ese valor crítico del residuo. **Renglón de datos en df que generan este error.**

e=residuos.drop([30],axis=0)

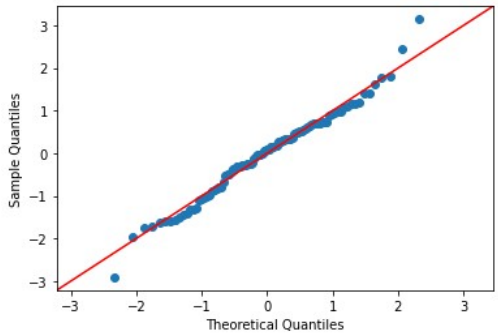
e

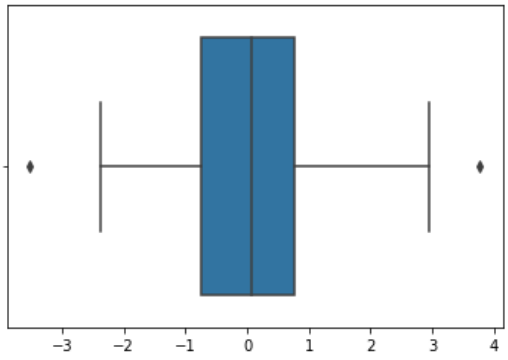
anderson(e)

anderson(e, dist="norm")

Vemos así que ya podemos quedarnos con hipótesis nula, es decir, los errores se ajustan a una distribución normal.

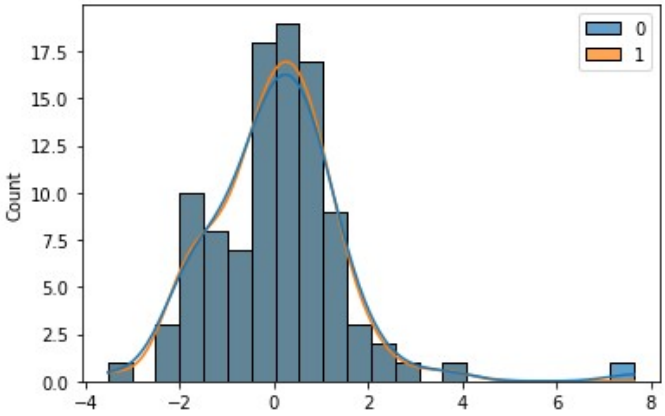
```
In [126]: e=residuos.drop([30],axis=0)

In [127]: e
Out[127]:
0    -0.389228
1    -0.403938
2    -0.966974
3    -0.326869
4     1.385161
        ...
95   -1.076079
96    0.340400
97   -0.066062
98   -2.376323
99    0.365127
Length: 99, dtype: float64

In [128]: anderson(e)
Out[128]: AndersonResult(statistic=0.4621465346816507, critical_values=array([0.555, 0.632, 0.758, 0.885, 1.052]), significance_level=array([15. , 10. ,  5. ,  2.5,  1. ]))

In [129]: anderson(e, dist="norm")
Out[129]: AndersonResult(statistic=0.4621465346816507, critical_values=array([0.555, 0.632, 0.758, 0.885, 1.052]), significance_level=array([15. , 10. ,  5. ,  2.5,  1. ]))

In [130]:
```

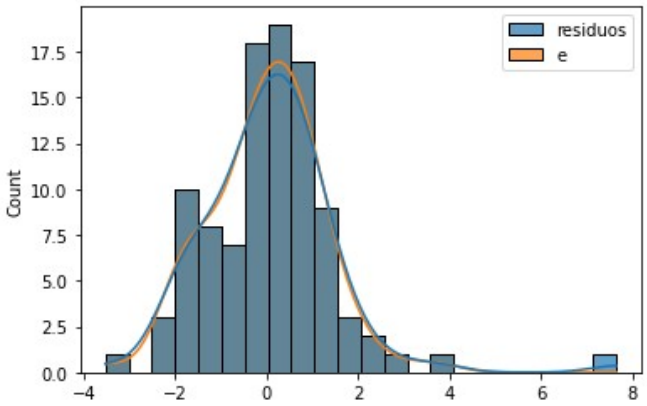




l=pd.concat([residuos, e], axis=1)

sns.histplot(data=l,alpha=0.7, kde=True)






```
In [164]: l
Out[164]:
           0         1
0  -0.389228 -0.389228
1  -0.403938 -0.403938
2  -0.966974 -0.966974
3  -0.326869 -0.326869
4   1.385161  1.385161
..       ...       ...
95 -1.076079 -1.076079
96  0.340400  0.340400
97 -0.066062 -0.066062
98 -2.376323 -2.376323
99  0.365127  0.365127
```

```
In [166]: l.columns=["residuos", "e"]

In [167]: l
Out[167]:
    residuos         e
0  -0.389228 -0.389228
1  -0.403938 -0.403938
2  -0.966974 -0.966974
3  -0.326869 -0.326869
4   1.385161  1.385161
..       ...       ...
95 -1.076079 -1.076079
96  0.340400  0.340400
97 -0.066062 -0.066062
98 -2.376323 -2.376323
99  0.365127  0.365127

[100 rows x 2 columns]

In [168]: sns.histplot(data=l,alpha=0.7, kde=True)
Out[168]: <AxesSubplot:ylabel='Count'>
```

Otra Prueba de Normalidad

Prueba de Shapiro-Wilk

La prueba de Shapiro-Wilk evalúa una muestra de datos y cuantifica la probabilidad de que los datos se extraigan de una distribución gaussiana, llamada así por Samuel Shapiro y Martin Wilk.

```
from scipy.stats import shapiro

# normality test
```

```
shapiro(residuos)

shapiro(e)
```

```
In [132]: from scipy.stats import shapiro

In [133]: shapiro(residuos)
Out[133]: ShapiroResult(statistic=0.9032313823699951, pvalue=2.012452569033485e-06)

In [134]: shapiro(e)
Out[134]: ShapiroResult(statistic=0.9870081543922424, pvalue=0.4455476403236389)
```

Una prueba más de Normalidad

Kolmogorov-Smirnov Test

from scipy.stats import kstest

kstest(residuos, "norm")
kstest(e, "norm")

```
In [138]: kstest(residuos, "norm")
Out[138]: KstestResult(statistic=0.08234536077916449, pvalue=0.4812841584485674)

In [139]: kstest(e, "norm")
Out[139]: KstestResult(statistic=0.08375950219330588, pvalue=0.4658928333292498)

In [140]:
```

Test de Normalidad, **¡OK!**

3. Igualdad de la varianza (**Heterocedasticidad**)

Una Prueba de Breusch-Pagan Utiliza las siguientes hipótesis nula H_0 y alternativa H_1.

H_0 : **No hay varianza constante (i,e Hay Homocedasticidad)**
H_1 : **Hay varianza constante (Existe Heterocedasticidad)**

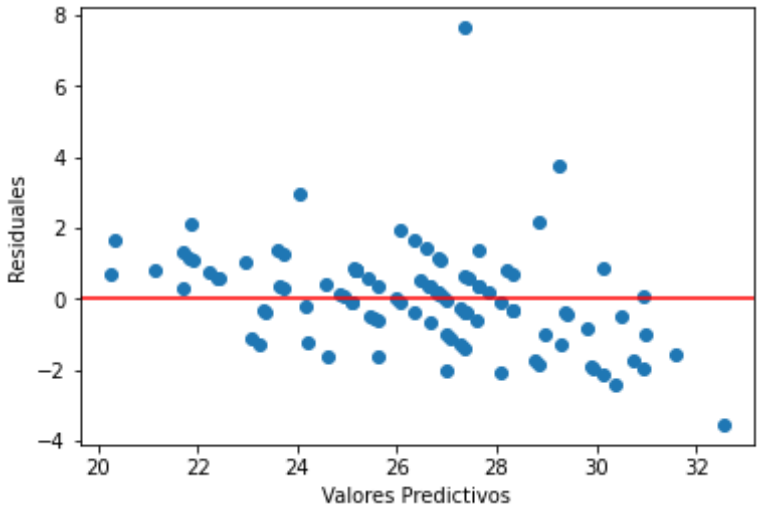
Ayuda: Poner teclas **Ctrl + Alt + Enter** , salto de instrucción)

import matplotlib.pyplot as plt

plt.scatter(modelo.predict(),**residuos**)

plt.axhline(0, color="red")
plt.xlabel("Valores Predictivos")
plt.ylabel("Residuales")

```
dtype: float64

In [11]: import matplotlib.pyplot as plt

In [12]: plt.scatter(modelo.predict(), residuales)
    ...: plt.axhline(0, color="red")
    ...: plt.xlabel("Valores Predictivos")
    ...: plt.ylabel("Residuales")
    ...:
Out[12]: Text(0, 0.5, 'Residuales')
```



Método estadístico para detectar Heteroscedasticidad (varianza constante):

import pandas as pd

df=pd.read_csv("low_birth_weight_infants.txt" , sep="\s+")
import statsmodels.formula.api as sm
modelo=sm.ols(formula="headcirc ~ gestage + birthwt -1 ", data=df).fit()
import statsmodels.stats.diagnostic as smd
breush_pagan_p = smd.het_breuschpagan(modelo.resid, modelo.model.exog)[1]

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

In [2]: df=pd.read_csv("low_birth_weight_infants.txt" , sep="\s+")

In [3]: import statsmodels.formula.api as sm

In [4]: modelo=sm.ols(formula="headcirc ~ gestage + birthwt -1 ", data=df).fit()

In [5]: import statsmodels.stats.diagnostic as smd

In [6]: breush_pagan_p = smd.het_breuschpagan(modelo.resid, modelo.model.exog)[1]

In [7]: breush_pagan_p
Out[7]: 0.0006722650506343095   <0.05

In [8]:
```

4. Independencia de los errores (Autocorelacción)

H_0 (hipótesis nula): No existe correlación entre los residuos.

H_A (hipótesis alternativa): Los residuos están autocorrelacionados.

Método estadístico para detectar autocorrelación

Test de de ***Durbin-Watson***
El estadístico de prueba siempre estará entre 0 y 4 con la siguiente interpretación:

- Una estadística de prueba de **2** indica que no hay correlación serial.
- Cuanto más cerca de **0** estén las estadísticas de la prueba, mayor será la evidencia de correlación serial positiva.
- Cuanto más cerca estén las estadísticas de la prueba de **4** , más evidencia de correlación serial negativa.

Regla: Los valores estadísticos de prueba entre el rango $[1.5, 2.5]$ se consideran que no hay problema de autocorrelación. Sin embargo, los valores fuera de este rango podrían indicar que la autocorrelación es un problema.

Veamos en nuestro ejemplo:

from statsmodels.stats.stattools import durbin_watson

durbin_watson (modelo.resid)

durbin_watson (e)

```
from statsmodels.stats.stattools import durbin_watson

durbin_watson (modelo.resid)
1.9129122348393195

durbin_watson (e)
1.7302364395630123
```

Independencia de los errores, Autocorelación.

Finalmente se cumplen los cuatro supuestos del error quitando los valores en la base que generaron el error.

```
In [224]: df.iloc[30]
Out[224]:
headcirc     35
length       36
gestage      31
birthwt     900
momage       23
toxemia       0
Name: 30, dtype: int64
```

df2=df.drop([30], axis=0)
import statsmodels.formula.api as sm
modelo2=sm.ols(formula="headcirc ~ gestage + birthwt -1", data=**df2**).fit()
modelo2.summary()

Nuestro modelo final sin este renglón es

OLS Regression Results

Dep. Variable:	headcirc	R-squared (uncentered):	0.998
Model:	OLS	Adj. R-squared (uncentered):	0.998
Method:	Least Squares	F-statistic:	2.362e+04
Date:	Wed, 12 Oct 2022	Prob (F-statistic):	4.10e-131
Time:	20:01:56	Log-Likelihood:	-158.39
No. Observations:	99	AIC:	320.8
Df Residuals:	97	BIC:	326.0
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
gestage	0.7604	0.022	34.624	0.000	0.717	0.804
birthwt	0.0040	0.001	7.037	0.000	0.003	0.005

Omnibus:	1.811	Durbin-Watson:	1.697
Prob(Omnibus):	0.404	Jarque-Bera (JB):	1.340
Skew:	-0.038	Prob(JB):	0.512
Kurtosis:	3.565	Cond. No.	205.

Antes de Limpiar era

modelo.summary()

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               headcirc   R-squared (uncentered):                   0.997
Model:                           OLS   Adj. R-squared (uncentered):              0.997
Method:                Least Squares   F-statistic:                          1.705e+04
Date:               Wed, 12 Oct 2022   Prob (F-statistic):                   2.57e-125
Time:                       20:14:11   Log-Likelihood:                         -177.14
No. Observations:                100   AIC:                                      358.3
Df Residuals:                     98   BIC:                                      363.5
Df Model:                          2
Covariance Type:           nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
gestage        0.7815      0.026     30.326      0.000       0.730       0.833
birthwt        0.0035      0.001      5.259      0.000       0.002       0.005
==============================================================================
Omnibus:                       47.080   Durbin-Watson:                   1.913
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              233.872
Skew:                           1.407   Prob(JB):                     1.64e-51
Kurtosis:                       9.943   Cond. No.                         203.
==============================================================================
```

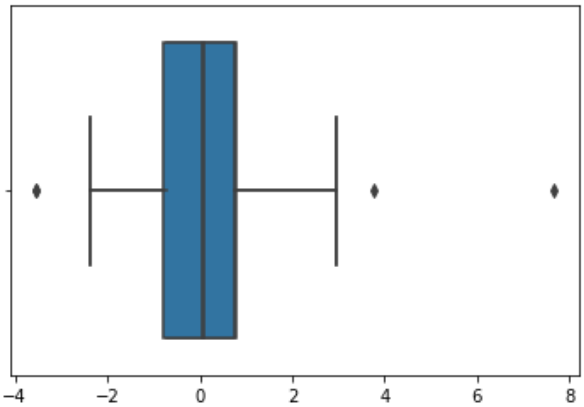
Date: Jueves 14 Octubre
Versión: 0.1c

Notas: Limpiando datos, ejemplo con los errores.

```
import pandas as pd
import statsmodels.formula.api as sm
df=pd.read_csv("https://raw.githubusercontent.com/crypto10pi/Datos/main/
low_birth_weight_infants.txt", sep="\s+")
df.info()
modelo=sm.ols(formula="headcirc ~ gestage + birthwt -1", data=df).fit()
residuos=modelo.resid
```

import seaborn as sns

sns.boxplot(residuos)



Q1 = residuos.quantile(0.25)
Q3 = residuos.quantile(0.75)
IQR= Q3-Q1
Maximum = Q3 + 1.5 * IQR
Minimum = Q1 - 1.5 * IQR

residuos > Maximum

l=residuos > Maximum
type(l)
l.info()

```
In [1]: import pandas as pd
   ...: import statsmodels.formula.api as sm
   ...: df=pd.read_csv("https://raw.githubusercontent.com/cryptol0pi/Datos/main/low_birth_weight_infants.txt", sep="\s+")

In [2]: df.info()
   ...: modelo=sm.ols(formula="headcirc ~ gestage + birthwt -1", data=df).fit()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   headcirc  100 non-null    int64
 1   length    100 non-null    int64
 2   gestage   100 non-null    int64
 3   birthwt   100 non-null    int64
 4   momage    100 non-null    int64
 5   toxemia   100 non-null    int64
dtypes: int64(6)
memory usage: 4.8 KB

In [3]: residuos=modelo.resid
   ...: Q1 = residuos.quantile(0.25)
   ...: Q3 = residuos.quantile(0.75)
   ...: IQR= Q3-Q1
   ...: Maximum = Q3 + 1.5 * IQR
   ...: Minimum = Q1 - 1.5 * IQR

In [4]: l=residuos > Maximum
   ...: type(l)
   ...: l.info()
<class 'pandas.core.series.Series'>
Int64Index: 100 entries, 0 to 99
Series name: None
Non-Null Count  Dtype
--------------  -----
100 non-null    bool
dtypes: bool(1)
memory usage: 900.0 bytes
```

s=residuos.index[l]

```
In [5]: s=residuos.index[l]

In [6]: s
Out[6]: Int64Index([30, 32], dtype='int64')

In [7]: list(s)
Out[7]: [30, 32]

In [8]:
```

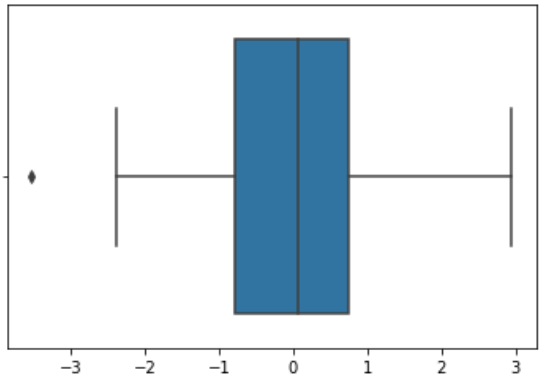
e=residuos.drop(s, axis=0)

e.describe()

e.info()

```
In [12]: e=residuos.drop(s, axis=0)

In [13]: e.describe()
Out[13]:
count    98.000000
mean     -0.061300
std       1.154619
min      -3.530010
25%      -0.780212
50%       0.063890
75%       0.746997
max       2.945761
dtype: float64

In [14]: e.info()
<class 'pandas.core.series.Series'>
Int64Index: 98 entries, 0 to 99
Series name: None
Non-Null Count  Dtype
--------------  -----
98 non-null     float64
dtypes: float64(1)
memory usage: 1.5 KB

In [15]:
```

sns.boxplot(e)



e.mean()
residuos.mean()

```
In [23]: e.mean()
Out[23]: -0.06130047227051146

In [24]: residuos.mean()
Out[24]: 0.0540830448910507
```

Quitemos ahora todos los errores outliers (atípicos).

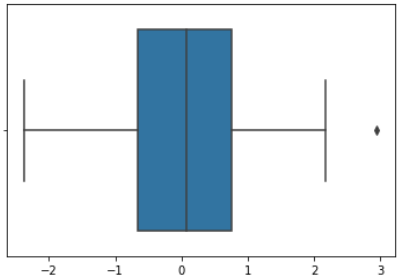
l = (residuos < Minimum) | (residuos > Maximum)

Esta linea contiene los índices de los renglones que dan problemas.

s=residuos.index[l]

e2=residuos.drop(**s**)

sns.boxplot(e2)



Veamos ahora los errores:

Valores=df.iloc[list(s)]
Valores
modelo.predict(Valores)

```
In [63]: Valores=df.iloc[list(s)]

In [64]: Valores
Out[64]:
    headcirc  length  gestage  birthwt  momage  toxemia
30        35      36       31      900      23        0
32        33      39       31     1440      26        0
72        29      41       35     1490      34        1

In [65]: modelo.predict(Valores)
Out[65]:
30    27.354032
32    29.230217
72    32.530010
dtype: float64
```

#Otra forma de hacer esto con solo los datos gestage y birthwt.

Valores=df[[“gestage”, “birthwt”]].iloc[list(s)]
modelo.predict(Valores)

```
In [67]: Valores=df[["gestage", "birthwt"]].iloc[list(s)]

In [68]: Valores
Out[68]:
    gestage  birthwt
30       31      900
32       31     1440
72       35     1490

In [69]: modelo.predict(Valores)
Out[69]:
30    27.354032
32    29.230217
72    32.530010
dtype: float64
```

#Los errores
modelo.predict(Valores) -df["headcirc"].iloc[s]

```
In [70]: modelo.predict(Valores) -df["headcirc"].iloc[s]
Out[70]:
30   -7.645968
32   -3.769783
72    3.530010
dtype: float64
```

Modelo final

df2=df.drop(s, axis=0)

df2.info()
modelo2=sm.ols(formula="headcirc ~ gestage + birthwt -1", data=df2).fit()
modelo2.summary()

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               headcirc   R-squared (uncentered):                   0.998
Model:                           OLS   Adj. R-squared (uncentered):              0.998
Method:                Least Squares   F-statistic:                          2.759e+04
Date:               Sat, 15 Oct 2022   Prob (F-statistic):                   4.67e-132
Time:                       11:45:10   Log-Likelihood:                         -146.27
No. Observations:                 97   AIC:                                      296.5
Df Residuals:                     95   BIC:                                      301.7
Df Model:                          2
Covariance Type:           nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
gestage        0.7640      0.020     37.922      0.000       0.724       0.804
birthwt        0.0039      0.001      7.463      0.000       0.003       0.005
==============================================================================
Omnibus:                        0.221   Durbin-Watson:                   1.561
Prob(Omnibus):                  0.895   Jarque-Bera (JB):                0.385
Skew:                          -0.081   Prob(JB):                        0.825
Kurtosis:                       2.738   Cond. No.                         202.
==============================================================================
```

Nota: Caso del residuo **r=7.645968091289099** y su índice **[30]**

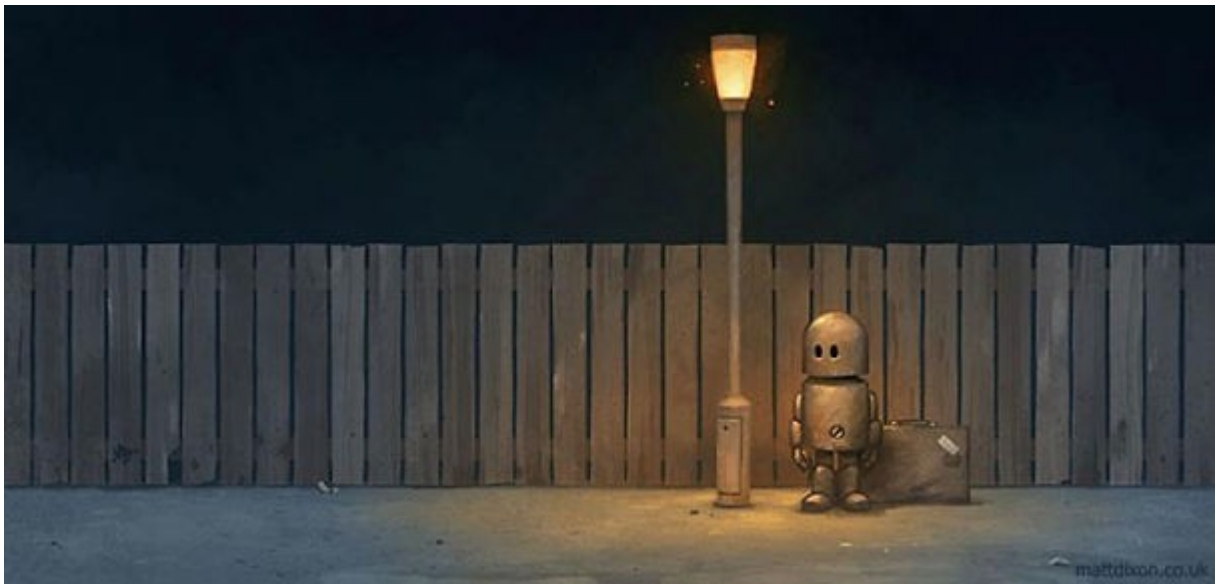
import pandas as pd

import statsmodels.formula.api as sm
df=pd.read_csv("https://raw.githubusercontent.com/crypto10pi/Datos/main/low_birth_weight_infants.txt", sep="\s+")
df.info()
modelo=sm.ols(formula="headcirc ~ gestage + birthwt -1", data=df).fit()
residuos=modelo.resid
residuos.max()

residuos == residuos.max()

l=residuos == residuos.max()
l
type(l)
l.info()
s=residuos.index[l]
s

```
In [15]: type(l)
Out[15]: pandas.core.series.Series

In [16]: l.info()
<class 'pandas.core.series.Series'>
Int64Index: 100 entries, 0 to 99
Series name: None
Non-Null Count  Dtype
--------------  -----
100 non-null    bool
dtypes: bool(1)
memory usage: 900.0 bytes

In [17]: s=residuos.index[l]

In [18]: s
Out[18]: Int64Index([30], dtype='int64')

In [19]: list(s)
Out[19]: [30]
```



Clase No, 19 Bloque 2

Seleccionar filas, columnas o registros en DataFrame

En los DataFrames de Pandas existen diferentes formas de seleccionar los registros de las filas y columnas. Siendo dos de las más importantes **iloc y loc**. La primera (iloc) permite seleccionar los elementos en base a la posición, mientras que la segunda (loc) permite seleccionar mediante etiquetas o declaraciones condicionales.

Veamos primero el caso iloc

import pandas as pd

df=pd.data_read(“https://reneshbedre.github.io/assets/posts/reg/bp.csv”)

df[0,0] = ? # Vemos el contenido del registro en filo 0, columna 0

df.iloc[0] = # Primera Renglón