

Un poco más de python



Loop (Ciclos o Bucles)

for ... in

Estructura básica

for "x" in "Valores":

accion1

accion2

etc.



Nota: range(n) da los números consecutivos 0, 1, 2, 3, ..., n-1.

```
>>> list(range(10))
```

```
>>> [0,1,2,3,4,5,6,7,8,9]
```

Ejemplo 0:

```
>>> for n in range(10):
```

```
>>>     print(n)
```

```
>>>     pass      # La instrucción "pass" señala el final del ciclo.
```

Multicolinealidad 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

$$VIF_i = \frac{1}{1 - R_i^2}$$

Donde R_i es el coeficiente de correlación múltiple 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 corrección 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 cálculo 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]:

```

```

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]:

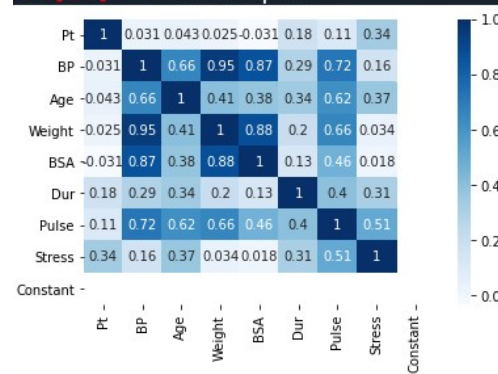
```

```

In [64]: import seaborn as sns

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

```



BP = blood Pressure
(Presión arterial)

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"]

Vamos ahora a pegar (concatenar, adjuntar) los dos dataframes, **Constante** y **X**

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

```
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**

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
```

```
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]:
```

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

```
In [51]: VIF=[]
In [52]: for i in range(6):
...:     VIF.append(variance_inflation_factor(dfX.values, i+1))
...:
In [52]:
In [53]: VIF
Out[53]:
[1.7628067217672165,
 8.417035029633047,
 5.328751470118887,
 1.2373094205198356,
 4.4135751655972895,
 1.8348453242645892]

In [54]: pd.DataFrame({"Variables": ['Age', 'Weight', 'BSA', 'Dur', 'Pulse', 'Stress'], "VIF":VIF })
Out[54]:
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 [55]:
```



Checando 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 X
[Terminal 2] No such file or directory: low_birth_weight_infants.txt

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     -1.076879
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

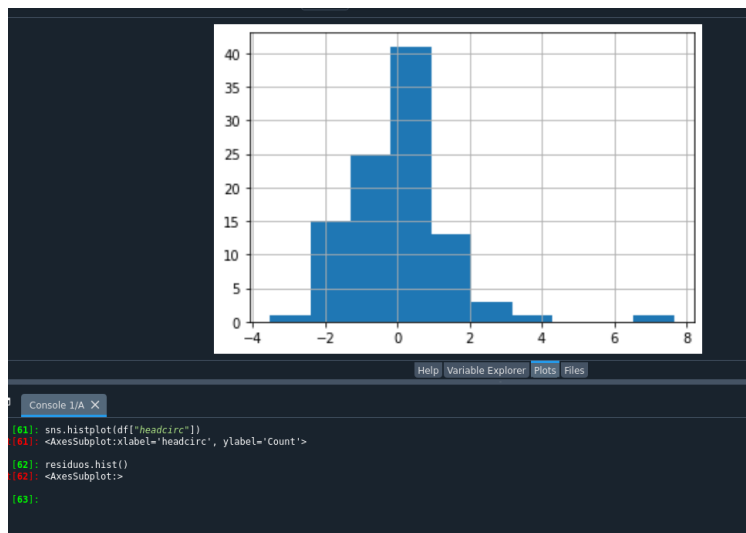
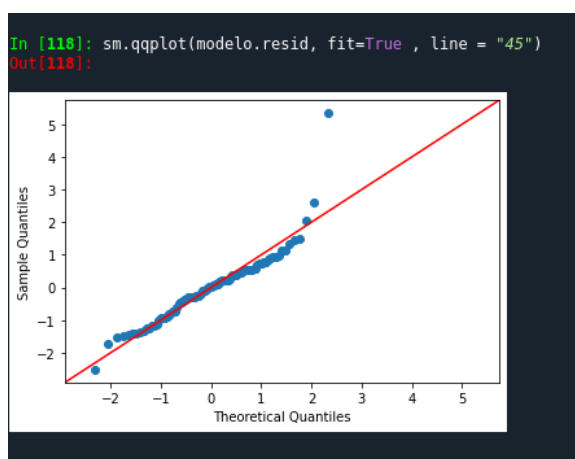


Gráfico QQ: Este gráfico es útil para determinar si los residuos siguen una distribución normal. Si los valores de los datos en el gráfico caen a lo largo de una línea aproximadamente recta en un ángulo de 45 grados, entonces los datos se distribuyen normalmente:

```
import statsmodels.api as sm
```

```
# Gráfico QQ
```

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



Método estadístico para detectar Normalidad:

Anderson-Darling Test:

```
#perform Anderson-Darling Test
```

```
from scipy.stats import anderson
residuos=modelo.resid
anderon(residuos)
```

```
In [119]: residuos=modelo.resid
In [120]: from scipy.stats import anderson
In [121]: residuos=modelo.resid
In [122]: anderson(residuos)
Out[122]: AndersonResult(statistic=1.2559913085416383, critical_values=array([0.555, 0.632, 0.759, 0.885, 1.053]), significance_level=array([15. , 10. , 5. , 2.5, 1. ]))
In [123]:
```

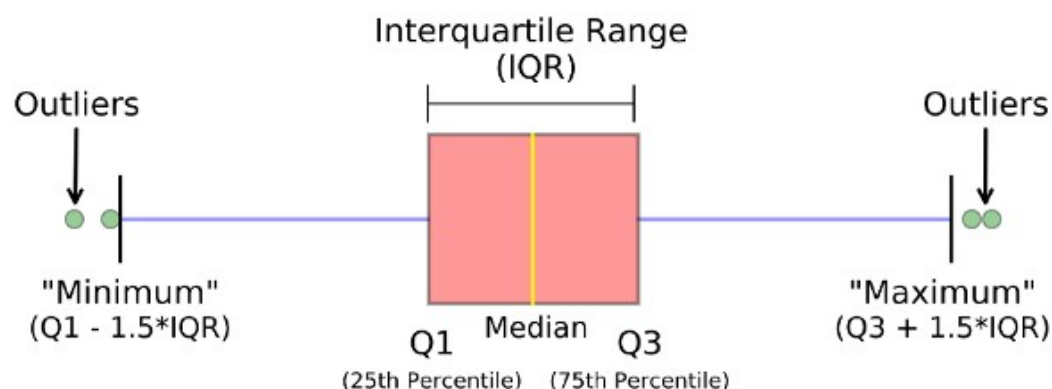
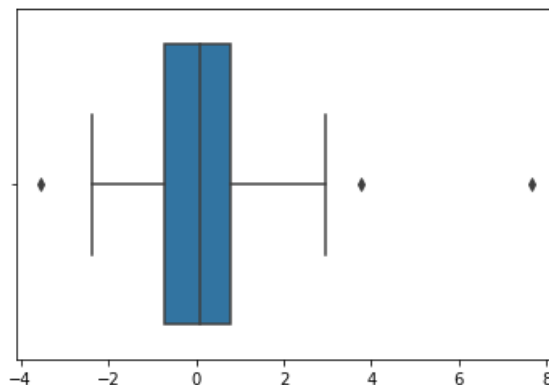
Sorprende, la prueba nos dice que tenemos que rechazar hipótesis nula, es decir que los errores no se ajustan a una distribución normal, ya que el estadístico=1.255 es , mayor que cualquier valor de significancia.

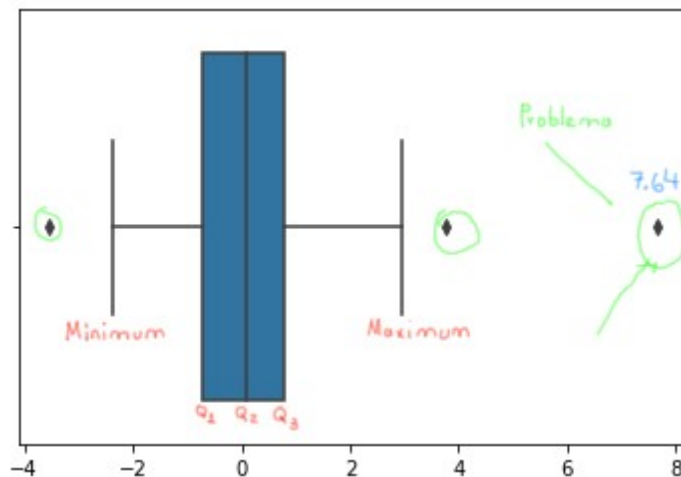


¿Que ocurre aquí?, los métodos gráficos nos dan evidencia de que los errores se ajustan a una distribución normal. ¡Veamos los datos atípicos!.

```
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
```

Quitamos solo ese valor crítico

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

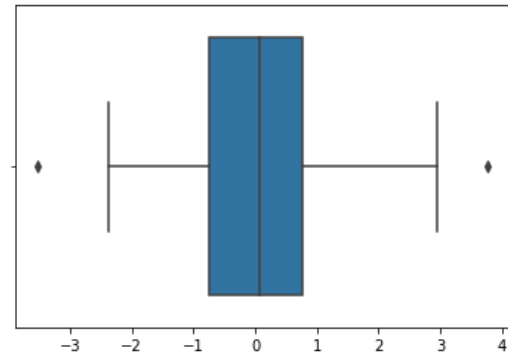
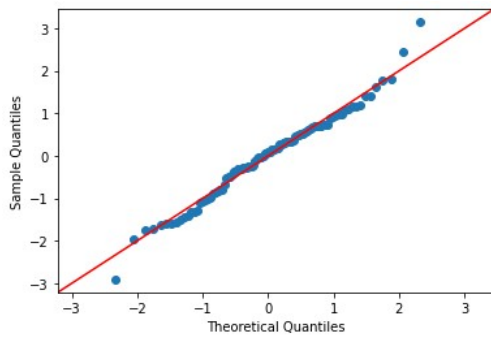
e

```
anderson(e)
```

```
anderson(e, dist="norm")
```

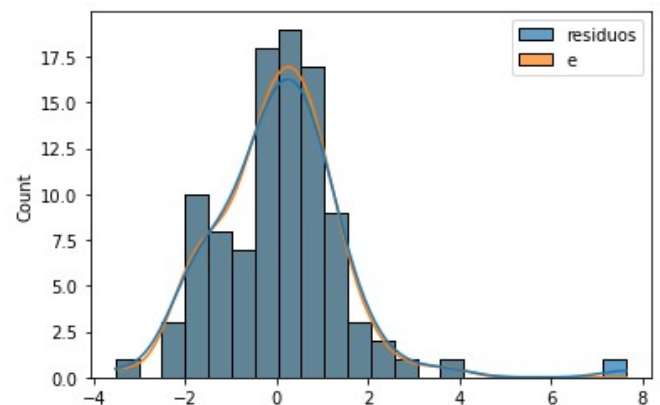
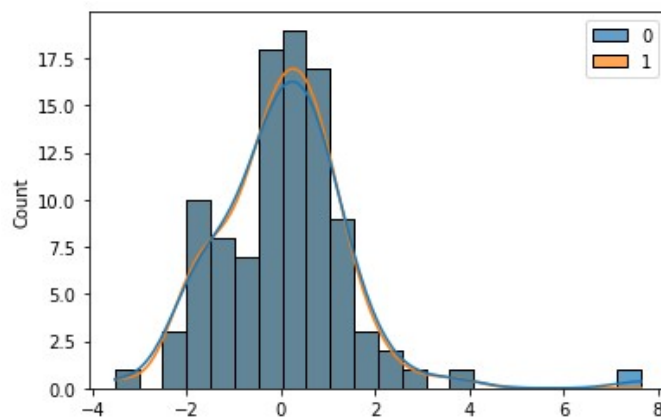
Vemos así que ya podemos quedarnos con hipótesis nula, es decir, los errores se ajustan a una distribución normal.

```
[126]: e=residuos.drop([30],axis=0)
[127]: e
[127]:
-0.389228
-0.403938
-0.906974
-0.326869
1.385161
...
-1.076079
0.348400
-0.066062
-2.376323
0.365127
length: 99, dtype: float64
[128]: anderson(e)
[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. ]))
[129]: anderson(e, dist="norm")
[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. ]))
[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

Una prueba de Breusch-Pagan utiliza las siguientes hipótesis nulas y alternativas:

La hipótesis nula (H_0): la homocedasticidad está presente.

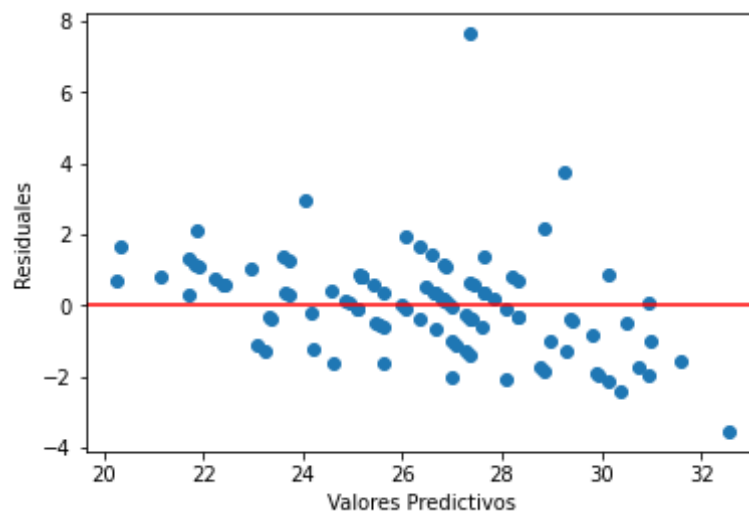
La hipótesis alternativa: (H_a): la homocedasticidad *no* está presente (es decir, existe heterocedasticidad)

Heteroscedasticidad: (Poner teclas Ctrl + Alt , salto de instrucción)

```
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:

```
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 (Autocorelación)

H₀ (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
```

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

```

=====
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: 0.1

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 fila 0, columna 0
```

```
df.iloc[0] = # Primera Renglón
```