taller_1

June 24, 2022

UNIVERSIDAD SIMÓN BOLÍVAR Departamento de Tecnología y Electrónica EC7817 - Tópico especial II: Inteligencia Artificial en Biomédica

Autor: William Chacón Profesor: Miguel Altuve

Junio, 2022

[942]:
```
import numpy as np
#import csv
#import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.cbook as cbook
#import statistics as stats

import scipy
from scipy import stats
#from scipy.stats import boxcox
#from scipy.special import inv_boxcox

import sklearn
import sklearn.model_selection as sklms
import sklearn.linear_model as skllm
import sklearn.metrics as sklmt
#from sklearn import cross_validation
#import sklearn.cross_validation as sklcv

#format of print for numpy float
np.set_printoptions(formatter={'float': lambda x: "{0:0.4f}".format(x)})
```

1 Taller 1

En el presente notebook se procede a realizar una estudio del dataset entregado.

Primero se inicia con la importación de los datos para su futuro pre-procesamiento.

[1467]:
```
file = np.loadtxt("abalone.data", dtype = str, delimiter =",")

print(file)
```

```
[['M' '0.455' '0.365' … '0.101' '0.15' '15']
 ['M' '0.35' '0.265' … '0.0485' '0.07' '7']
 ['F' '0.53' '0.42' … '0.1415' '0.21' '9']
 …
 ['M' '0.6' '0.475' … '0.2875' '0.308' '9']
 ['F' '0.625' '0.485' … '0.261' '0.296' '10']
 ['M' '0.71' '0.555' … '0.3765' '0.495' '12']]
```

[1468]:
```
print("Data file size is: (" + str(np.size(file[:][:],0)) + "," + str(np.
↪size(file[:][:],1)) + ")")
```

```
Data file size is: (4177,9)
```

[1469]:
```
varnames = np.array(['Sex', 'Length', 'Diameter', 'Height', 'Whole_weight',␣
↪'Shucked_weight', 'Viscera_weight', 'Shell_weight', 'Rings'])


data = np.vstack( (varnames, file) )

print(data)
```

```
[['Sex' 'Length' 'Diameter' … 'Viscera_weight' 'Shell_weight' 'Rings']
 ['M' '0.455' '0.365' … '0.101' '0.15' '15']
 ['M' '0.35' '0.265' … '0.0485' '0.07' '7']
 …
 ['M' '0.6' '0.475' … '0.2875' '0.308' '9']
 ['F' '0.625' '0.485' … '0.261' '0.296' '10']
 ['M' '0.71' '0.555' … '0.3765' '0.495' '12']]
```

[1470]:
```
print("Data array size is: (" + str(np.size(data[:][:],0)) + "," + str(np.
↪size(data[:][:],1)) + ")")
```

```
Data array size is: (4178,9)
```

A continuación, se procede a transformar los valores M, F e I del atributo 'sex' en 1, 0 y -1 respectivamente. De esta forma, se tendrán valores numéricos para todos los atributos de cada muestra del conjunto de datos. Con esto, se procede a convertir todos los valores del arreglo, que actualmente son strings, a números, floats.

[1471]:
```
file[ file[ : , 0 ] == 'M'  , 0 ] = 1
file[ file[ : , 0 ] == 'F'  , 0 ] = 0
file[ file[ : , 0 ] == 'I'  , 0 ] = -1
```

[1472]:
```
print(file)
```

```
[['1' '0.455' '0.365' … '0.101' '0.15' '15']
 ['1' '0.35' '0.265' … '0.0485' '0.07' '7']
 ['0' '0.53' '0.42' … '0.1415' '0.21' '9']
 …
```

```
['1' '0.6' '0.475' … '0.2875' '0.308' '9']
['0' '0.625' '0.485' … '0.261' '0.296' '10']
['1' '0.71' '0.555' … '0.3765' '0.495' '12']]
```

[1476]:
```
data2 = file.astype( float );
print(data2)
```

```
[[1.0000 0.4550 0.3650 … 0.1010 0.1500 15.0000]
 [1.0000 0.3500 0.2650 … 0.0485 0.0700 7.0000]
 [0.0000 0.5300 0.4200 … 0.1415 0.2100 9.0000]
 …
 [1.0000 0.6000 0.4750 … 0.2875 0.3080 9.0000]
 [0.0000 0.6250 0.4850 … 0.2610 0.2960 10.0000]
 [1.0000 0.7100 0.5550 … 0.3765 0.4950 12.0000]]
```

Teniendo el arreglo de números, se procede a determinar todas las variables estadísticas de cada conjunto de datos asociados a cada uno de los atributos.

[667]:
```
#histData2 = [ np.histogram(data2[:,0]), np.histogram(data2[:,1]), np.
↪histogram(data2[:,2]), np.histogram(data2[:,3]), np.histogram(data2[:,4]),␣
↪np.histogram(data2[:,5]), np.histogram(data2[:,6]), np.histogram(data2[:
↪,7]), np.histogram(data2[:,8])]
#print(histData2)
```

1.1 Análisis Estadístico

Inicializando variables de interés

[580]:
```
#Parámetros estadísticos de interés para los 9 atributos del arreglo

#Inicialización

DataMean = np.zeros((9,1))        #media
DataMedian = np.zeros((9,1))      #mediana
DataMode = np.zeros((9,1))        #moda
DataMax = np.zeros((9,1))         #máximo
DataMin = np.zeros((9,1))         #mínimo
DataRange = np.zeros((9,1))       #rango
DataDesv = np.zeros((9,1))        #desviación
DataSkew = np.zeros((9,1))        #asimetría
DataKurt = np.zeros((9,1))        #curtosis
```

[581]:
```
#cálculo de los parámetros estadísticos

DataMean = np.mean(data2, axis = 0)
DataMedian = np.median(data2, axis = 0)
DataMode = stats.mode(data2, axis = 0)
DataMax = np.max(data2, axis = 0)
```

```
DataMin = np.min(data2, axis = 0)
DataRange = DataMax - DataMin
DataDesv = np.std(data2, axis = 0)
DataSkew = stats.skew(data2, axis = 0, bias = 0)
DataKurt = stats.kurtosis(data2, axis = 0, bias = 0)
```

[582]:
```
#Presentación de los valores de los parámetros estadísticos estudiados

print('la media de los atributos es:');
print(DataMean);
print();
print('la mediana de los atributos es:');
print(DataMedian);
print();
print('la moda de los atributos es:');
print(DataMode);
print();
print('el valor máximo de los atributos es:');
print(DataMax);
print();
print('el valor mínimo de los atributos es:');
print(DataMin);
print();
print('lel rango de los atributos es:');
print(DataRange);
print();
print('la desviación estándar de los atributos es:');
print(DataDesv);
print();
print('la asimetría de los atributos es:');
print(DataSkew);
print();
print('el curtosis de los atributos es:');
print(DataKurt);
print();
```

```
la media de los atributos es:
[0.04452957 0.5239921  0.40788125 0.1395164  0.82874216 0.35936749
 0.18059361 0.23883086 9.93368446]

la mediana de los atributos es:
[0.     0.545  0.425  0.14   0.7995 0.336  0.171  0.234  9.    ]

la moda de los atributos es:
ModeResult(mode=array([[1.    , 0.55  , 0.45  , 0.15  , 0.2225, 0.175 , 0.1715,
0.275 ,
        9.    ]]), count=array([[1528,   94,  139,  267,    8,   11,   15,   43,
```

```
689]]))

el valor máximo de los atributos es:
[ 1.      0.815   0.65    1.13    2.8255  1.488   0.76    1.005  29.    ]

el valor mínimo de los atributos es:
[-1.0e+00  7.5e-02  5.5e-02  0.0e+00  2.0e-03  1.0e-03  5.0e-04  1.5e-03
  1.0e+00]

lel rango de los atributos es:
[ 2.      0.74    0.595   1.13    2.8235  1.487   0.7595  1.0035 28.    ]

la desviación estándar de los atributos es:
[0.8277156  0.12007854 0.09922799 0.04182205 0.49033031 0.22193638
 0.10960113 0.13918601 3.22378307]

la asimetría de los atributos es:
[-0.0830555  -0.63987327 -0.60919814  3.12881738  0.53095856  0.71909792
  0.59185215  0.62092683  1.1141019 ]

el curtosis de los atributos es:
[-1.53607188e+00  6.46209739e-02 -4.54755814e-02  7.60255092e+01
 -2.36435043e-02  5.95123678e-01  8.40117490e-02  5.31926126e-01
  2.33068743e+00]
```

[583]:
```
# Obtención de variables de IQR

DataQ1 = np.percentile(data2, 25, axis = 0)
DataQ2 = np.percentile(data2, 50, axis = 0)
DataQ3 = np.percentile(data2, 75, axis = 0)

DataIQR = stats.iqr(data2, axis = 0)
```

[584]:
```
#Presentación de los parámetros IQR

print('Q1 de los atributos es:');
print(DataQ1);
print();
print('Q2 de los atributos es:');
print(DataQ2);
print();
print('Q3 de los atributos es:');
print(DataQ3);
print();
print('IQR de los atributos es:');
print(DataIQR);
```

```
print();
```

```
Q1 de los atributos es:
[-1.      0.45    0.35    0.115   0.4415  0.186   0.0935  0.13    8.    ]

Q2 de los atributos es:
[0.     0.545  0.425  0.14   0.7995 0.336  0.171  0.234  9.    ]

Q3 de los atributos es:
[ 1.     0.615  0.48   0.165  1.153  0.502  0.253  0.329 11.   ]

IQR de los atributos es:
[2.     0.165  0.13   0.05   0.7115 0.316  0.1595 0.199  3.    ]
```

Es importante destacar que los atributos 'sex' (argumento 0) y 'number of rings' (argumento 8) son valores de naturaleza discreta, por lo que carece de sentido hablar de los parámetros estadísticos anteriormente calculados.

Siguiendo este orden de ideas, los valores correspondientes a los parámetros estadísticos calculados previamente para los atributos mencionados serán obviados.

Adicionalmente, tampoco se realizará un análisis o preprocesamiento de los mismos (inicialmente se consideró el normalizar el atributo de número de anillos, pero como el mismo representa el resultado que se pretende estimar con la regresión lineal, para las primeras pruebas no se realizará la normalización).

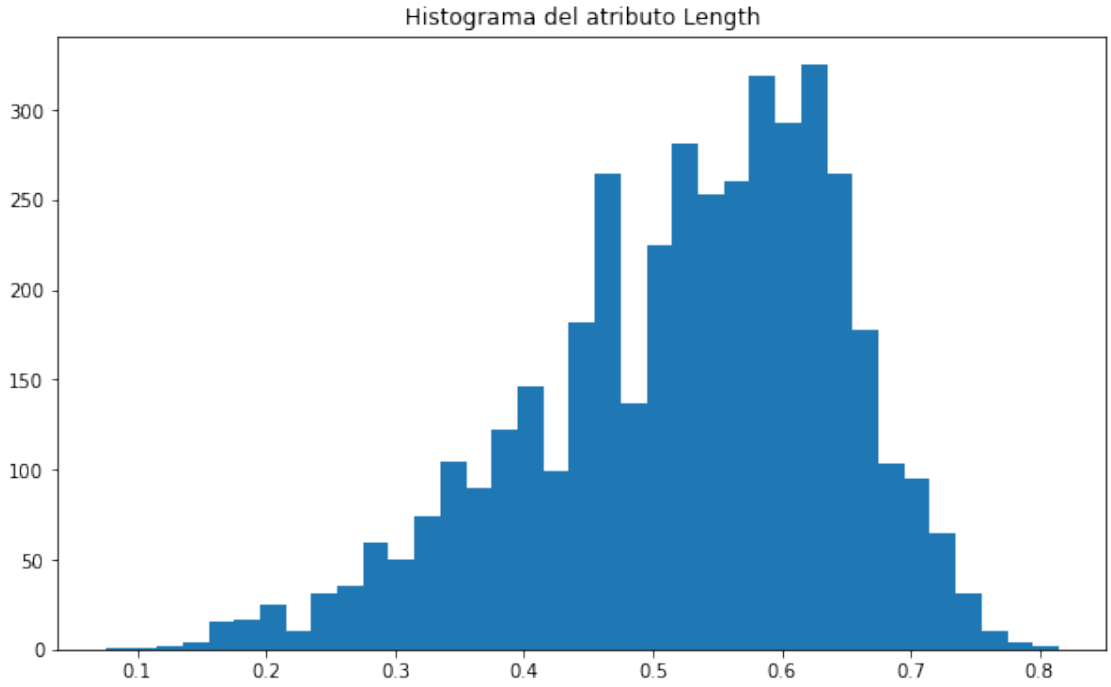
A continuación, se realizará un estudio del comportamiento de los atributos de variables continuas.

1.1.1 Análisis estadístico de los atributos

Primero se comenzará con los histogramas de cada una de las variables continuas.

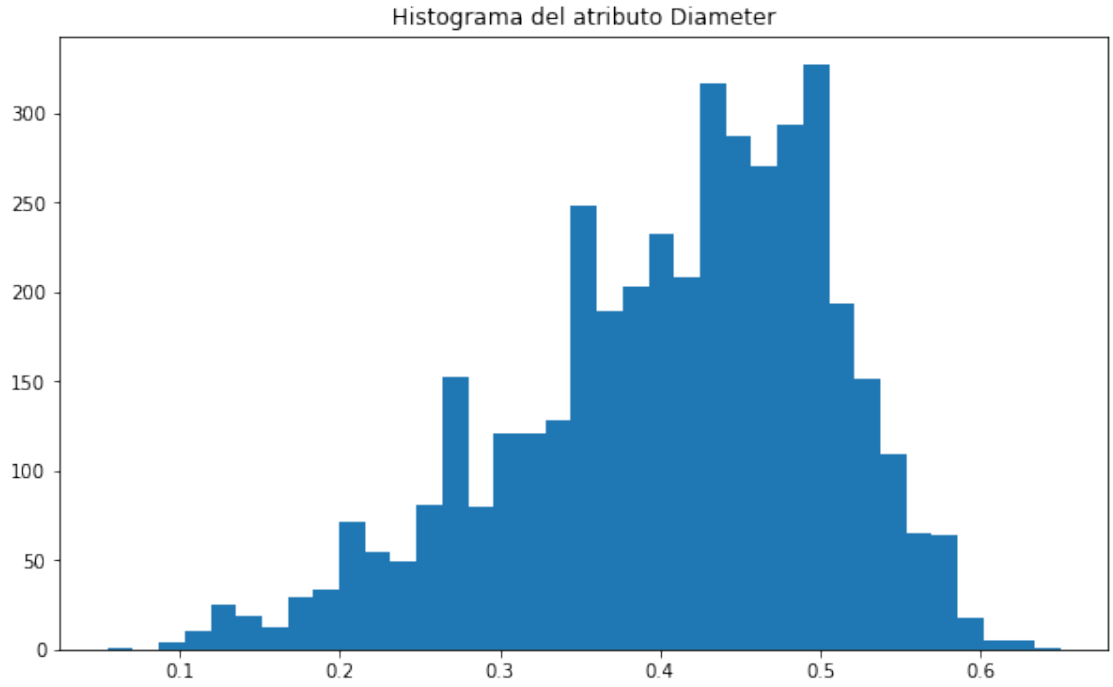
Atributo 'Length'

[585]:
```
#np.histogram( data2[:,1] )

fig = plt.figure(figsize =(10, 6))
plt.hist( data2[:,1] , bins = 'auto' );
plt.title('Histograma del atributo Length');
```



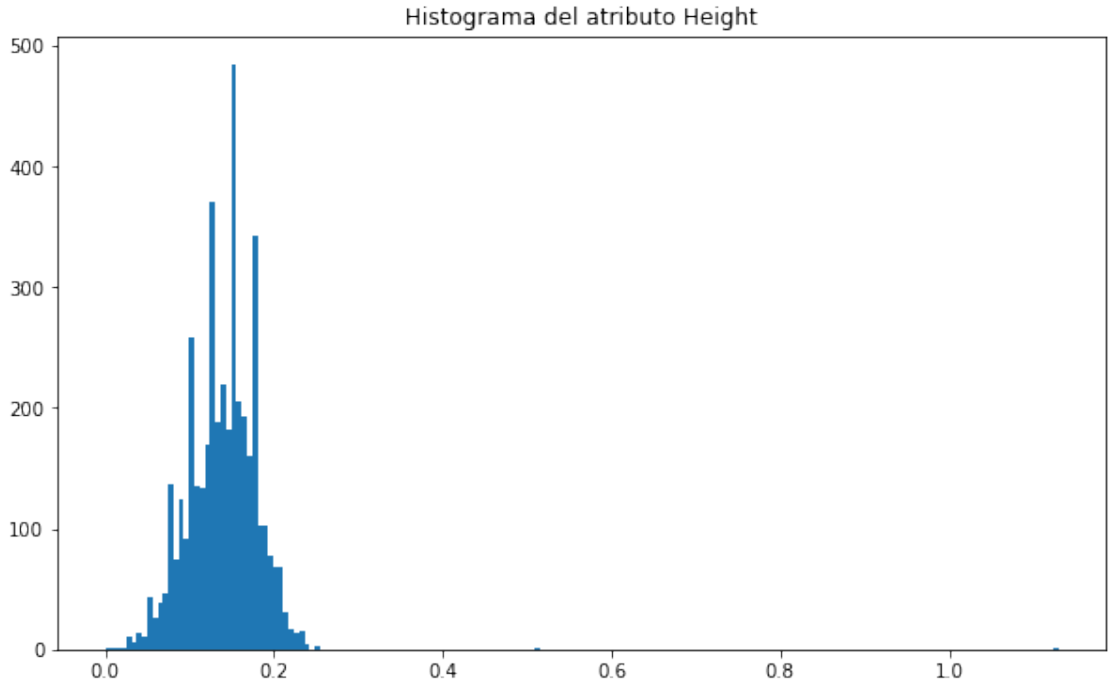
Atributo 'Diameter'.

[586]:
```
#np.histogram( data2[:,2] )

fig = plt.figure(figsize =(10, 6))
plt.hist( data2[:,2] , bins = 'auto' );
plt.title('Histograma del atributo Diameter');
```



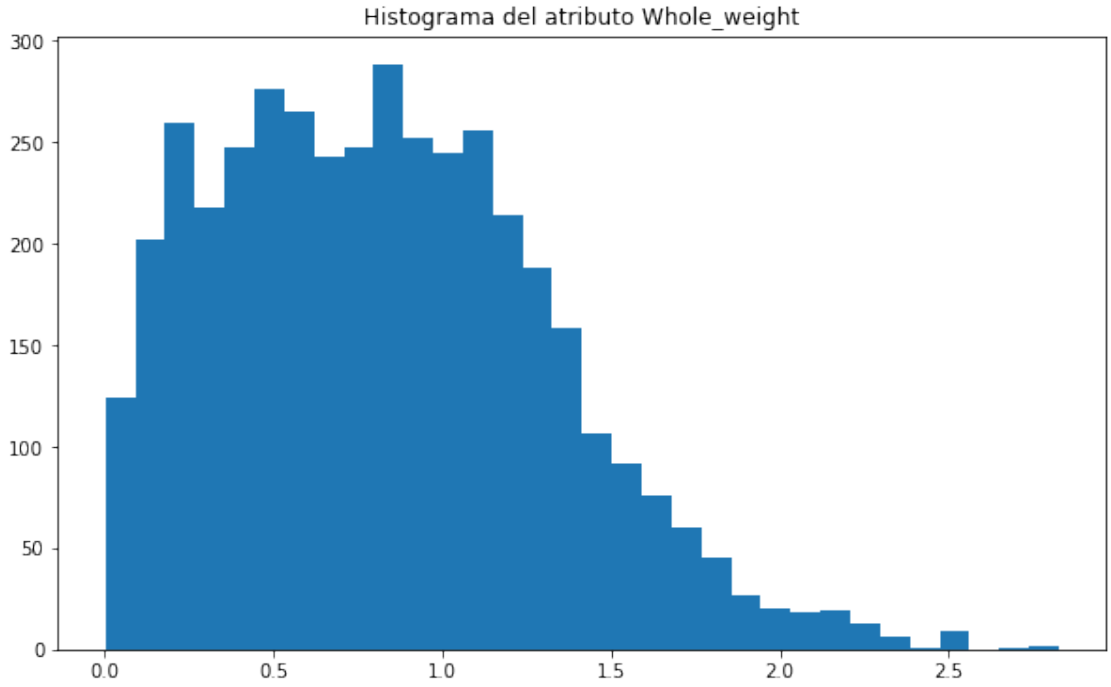
Atributo 'Height'

[587]:
```
#np.histogram( data2[:,3] )

fig = plt.figure(figsize =(10, 6))
plt.hist( data2[:,3] , bins = 'auto' );
plt.title('Histograma del atributo Height');
```



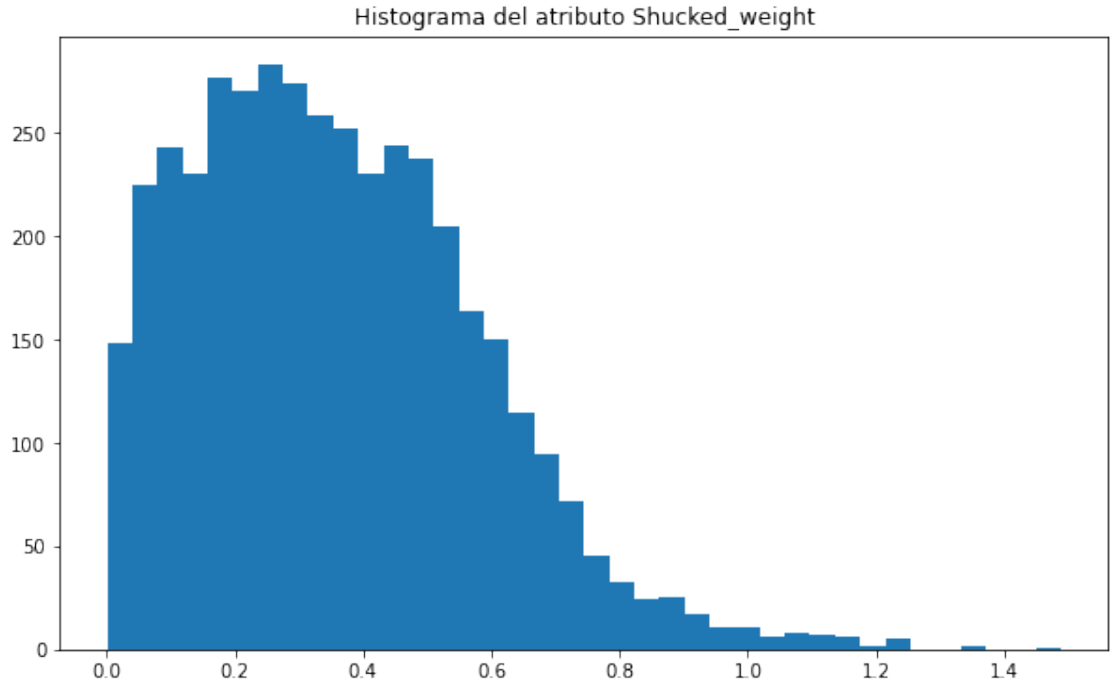
Atributo 'Whole_weight'

[588]:
```
#np.histogram( data2[:,4] )

fig = plt.figure(figsize =(10, 6))
plt.hist( data2[:,4] , bins = 'auto' );
plt.title('Histograma del atributo Whole_weight');
```



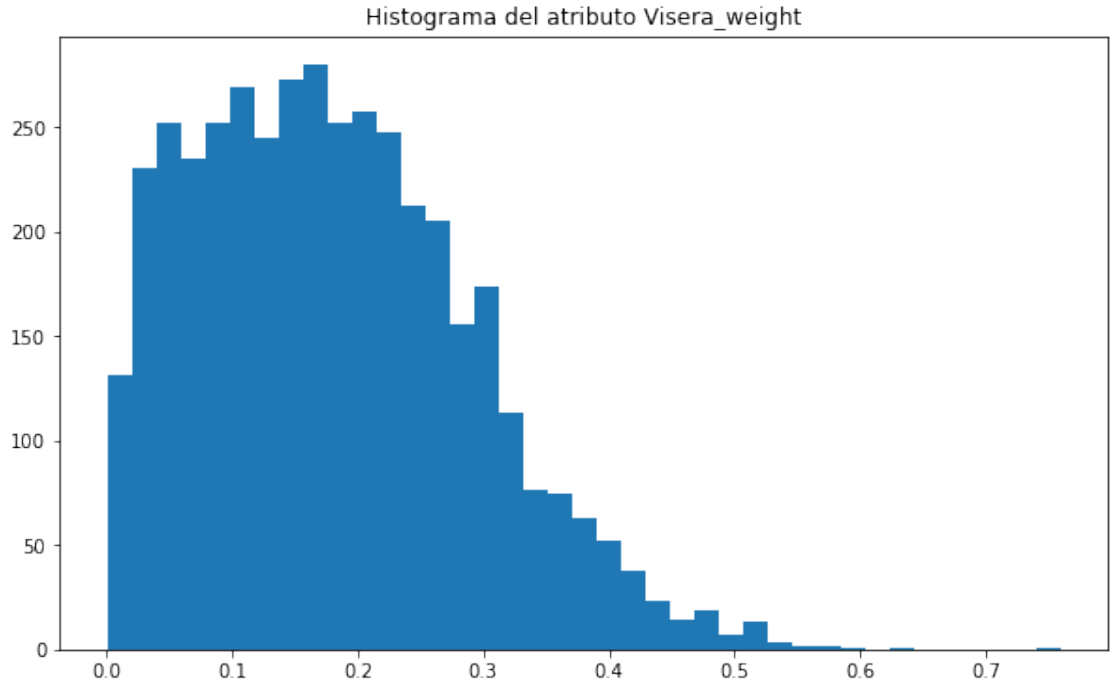
Atributo 'Shucked_weight'

[589]:
```
#np.histogram( data2[:,5] )

fig = plt.figure(figsize =(10, 6))
plt.hist( data2[:,5] , bins = 'auto' );
plt.title('Histograma del atributo Shucked_weight');
```



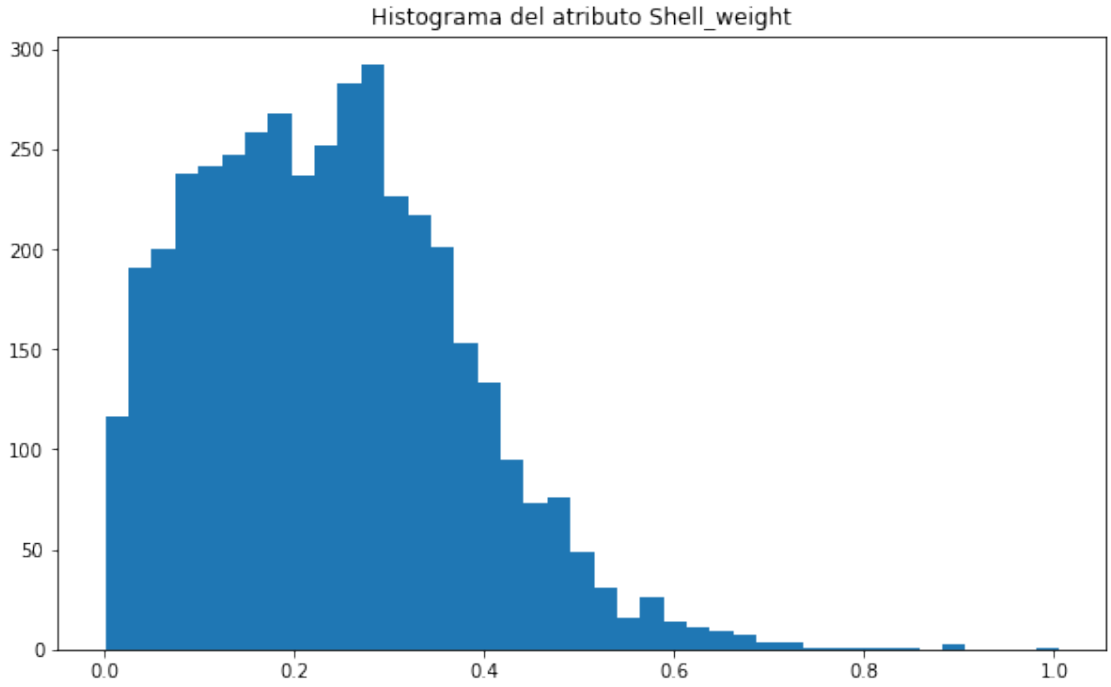
Atributo 'Visera_weight'

[590]:
```
#np.histogram( data2[:,6] )

fig = plt.figure(figsize =(10, 6))
plt.hist( data2[:,6] , bins = 'auto' );
plt.title('Histograma del atributo Visera_weight');
```



Atributo 'Shell_weight'

[591]:
```
#np.histogram( data2[:,7] )

fig = plt.figure(figsize =(10, 6))
plt.hist( data2[:,7] , bins = 'auto' );
plt.title('Histograma del atributo Shell_weight');
```



Ahora se procede a presentar los diagramas inter-quartil (IQR) de los atributos de variable continua.

1.1.2 Extracción de las muestras con valores atípicos

Es importante mencionar que se selecciona el IQR como criterio para la detección de valores atípicos dentro de los datos ya que el mismo proporciona un rango más estrecho para los datos que el criterio de la $\mu \pm 3\sigma$.

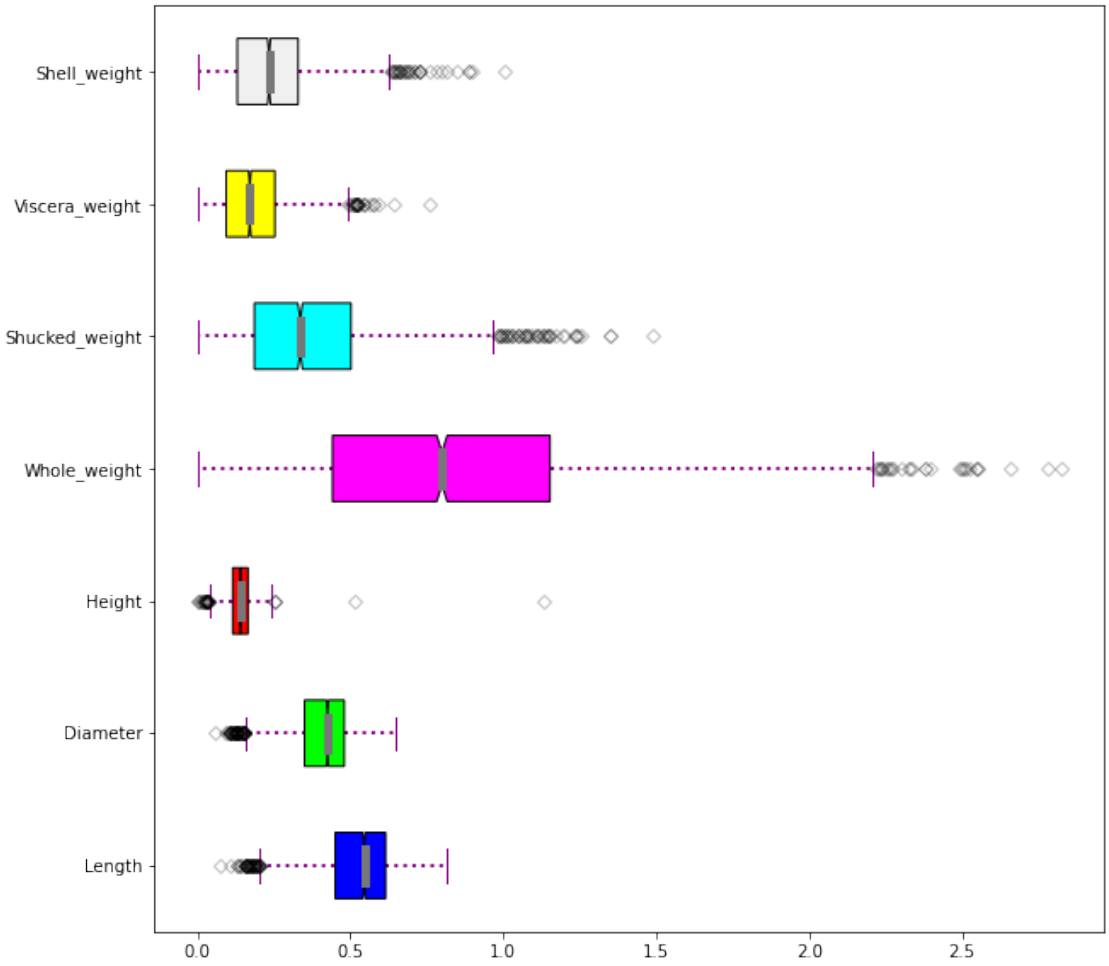
Figura 1: Esquema de la distribución e identificación de valores atípicos mediante IQR

fuente de la figura: https://towardsdatascience.com/create-and-customize-boxplots-with-pythons-matplotlib-to-get-lots-of-insights-from-your-data-d561c9883643?gi=c740821c406a

[592]:
```
stats2 = cbook.boxplot_stats(data2[:, 1:8 ], labels = varnames[1:8])
```

[593]:
```
#fig, ax = plt.subplots(1, 1)
fig = plt.figure(figsize =(10, 10))
ax = fig.add_subplot(111)

# Plot boxplots from our computed statistics
bp = ax.bxp([stats2[i] for i in range(7)], positions=range(7), patch_artist =␣
 ↪True, vert = 0, shownotches ='True');
```

```
colors = ['#0000FF', '#00FF00', '#FF0000', '#FF00FF', '#00FFFF', '#FFFF00',␣
 ↪'#F0F0F0']

for patch, color in zip(bp['boxes'], colors):
    patch.set_facecolor(color)

# changing color and linewidth of
# whiskers
for whisker in bp['whiskers']:
    whisker.set(color ='#8B008B',
                linewidth = 2,
                linestyle =":")

# changing color and linewidth of
# caps
for cap in bp['caps']:
    cap.set(color ='#8B008B',
            linewidth = 1)

# changing color and linewidth of
# medians
for median in bp['medians']:
    median.set(color ='#777777',
               linewidth = 5)

# changing style of fliers
for flier in bp['fliers']:
    flier.set(marker ='D',
              color ='#e7298a',
              alpha = 0.25)
```



Como se puede apreciar, existen una cantidad considerable de valores atípicos de acuerdo al estudio de IQR mostrado anteriormente.

Las muestras que presenten algún valor atípico en cualquiera de los atributos presentados anteriormente deben de ser removidos, de modo que no influencien negativamente el proceso de estimación de comportamiento.

A continuación, se procede a remover los valores atípicos según el criterio establecido.

[594]:
```
#Detector de valores atípicos por debajo del límite inferior
aux1 = data2 < ( DataQ1 - 1.5 * DataIQR ) * np.ones( (np.size(data2,0),np.
↪size(data2,1)) )
aux1 = np.sum(aux1, axis = 1)
```

[595]:
```
#Detector de valores atípicos por encima del límite supeior
aux2 = data2 > ( DataQ3 + 1.5 * DataIQR ) * np.ones( (np.size(data2,0),np.
↪size(data2,1)) )
aux2 = np.sum(aux2, axis = 1)
```

[596]:
```
#Cantidad de atributos cuyo valor es inferior al mínimo valor aceptable como␣
↪típico
sum(aux1 != 0)
```

[596]: 66

[597]:
```
#Cantidad de atributos cuyo valor es supeior al máximo valor aceptable como␣
↪típico
sum(aux2 != 0)
```

[597]: 330

[598]:
```
#Extracción de las muestras con valores atípicos
data3 = data2[ (aux1 == 0) & (aux2 == 0), :]
```

Cantidad de Muestras Restantes

[599]:
```
#Cantidad de muestras restantes
np.size(data3,0)
```

[599]: 3781

Cantidad de muestras extraidas de la data

[600]:
```
#Cantidad de muestras con valores removidas
np.size(data2,0) - np.size(data3,0)
```

[600]: 396

Si se realiza una comparación, se aprecia que la cantidad de muestras descartadas por poseer valores atípicos representa un 9.48% de la data original, dejando solo un 90.52% de la data como muestras apropiadas para efectuar la regresión.

La razón de haberse descartado tantas muestras se debe a la presencia de al menos un valor atípico en alguna de los 7 atributos continuas que, si se visualiza la gráfica del IQR, en algunos atributos presentaban una gran cantidad de valores atípicos.

No obstante, más del 90% de las muestras en una data de más de 4170 muestras es una cantidad aceptable para relaizar una primera predicción (iteración) del comportamiento de la Abalone.

[1477]:
```
print("Porcentaje de muestras removidas = " + str( ( np.size(data2,0) - np.
↪size(data3,0) ) / np.size(data2,0) *100 ))
print("Porcentaje de muestras restantes = " + str( np.size(data3,0) / np.
↪size(data2,0) *100 ))
```

```
Porcentaje de muestras removidas = 9.480488388795786
Porcentaje de muestras restantes = 90.51951161120422
```

Una vez removido los valores atípicos, se tienen 3781 muestras restantes con las cuales realizar la regresión lineal para predecir la edad del Abulón.

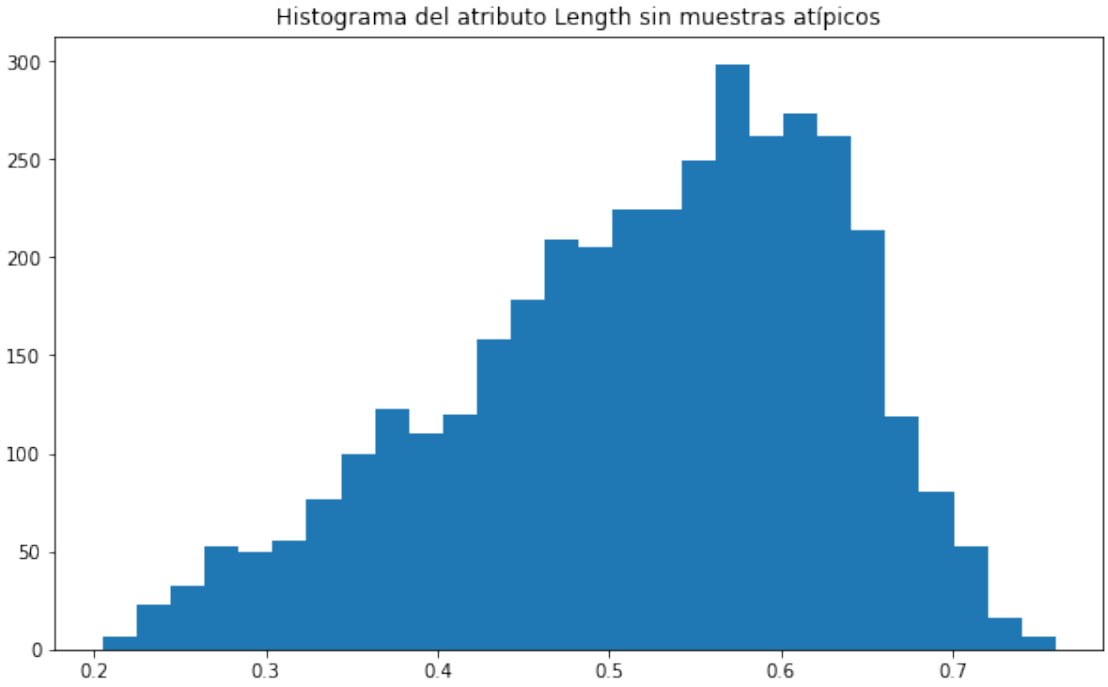
A continuación, se procede a realizar los preprocesamientos apropiados de los datos para asegurar un buen desempeño en durante el proceso de estimación.

1.1.3 Análisis estadístico de las muestras con valores típicos

Primero se debe visualizar el comportamiento estadístico de las variables de interés sin los valores atípicos.

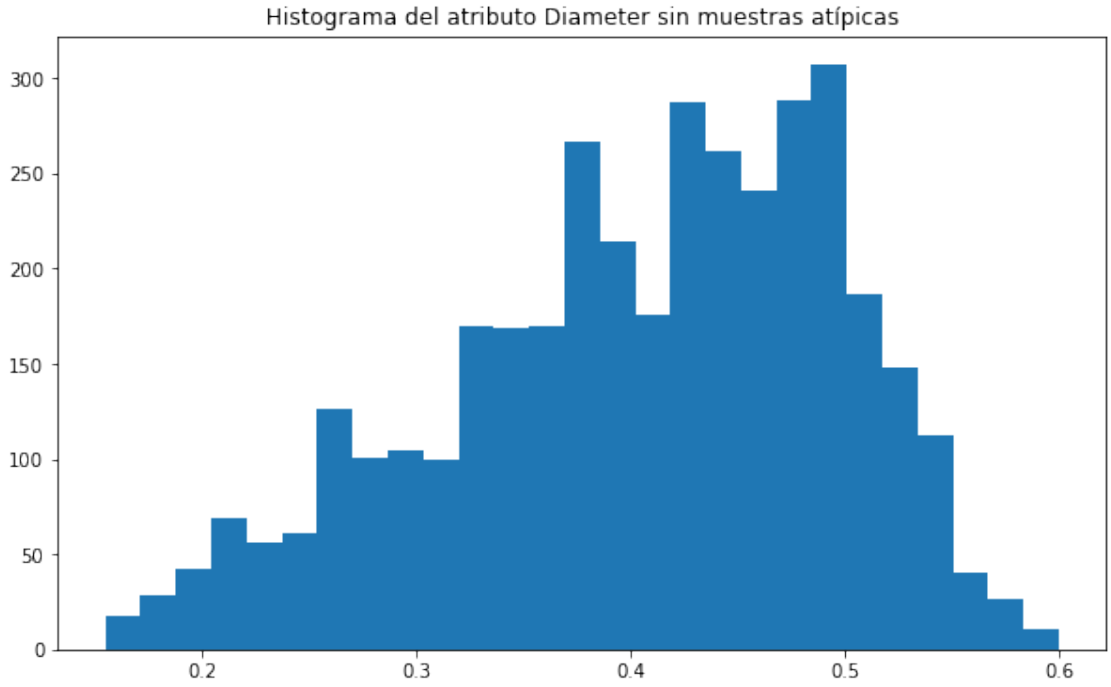
Atributo 'Lenght'.

[602]:
```
fig = plt.figure(figsize =(10, 6))
plt.hist( data3[:,1] , bins = 'auto' );
plt.title('Histograma del atributo Length sin muestras atípicos');
```



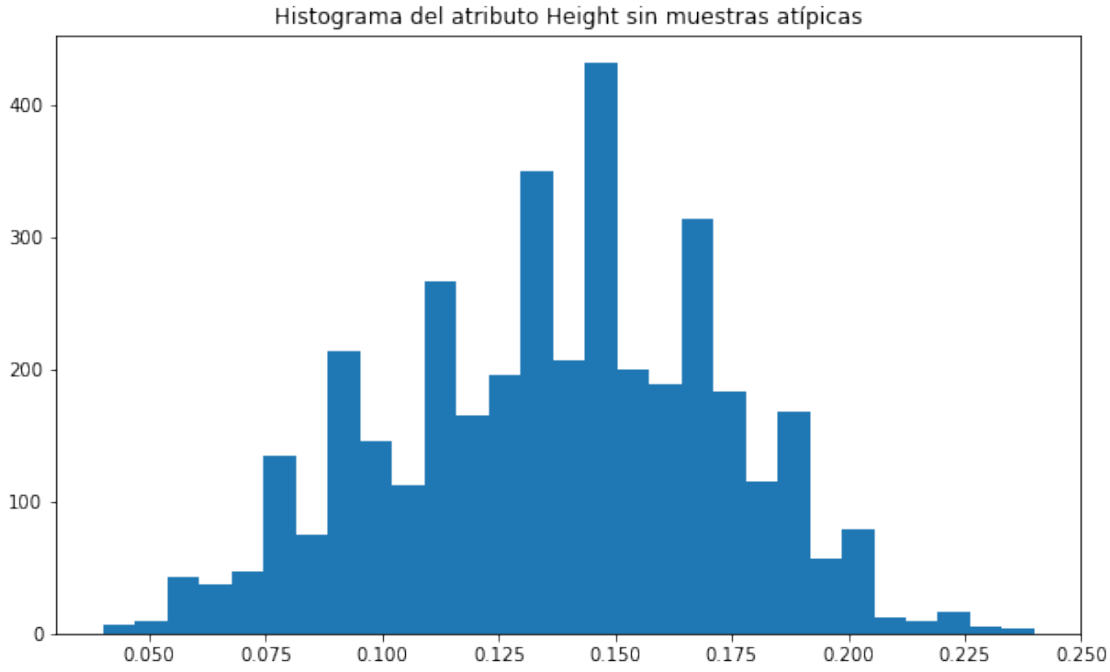
Atributo 'Diameter'.

[603]:
```
fig = plt.figure(figsize =(10, 6))
plt.hist( data3[:,2] , bins = 'auto' );
plt.title('Histograma del atributo Diameter sin muestras atípicas');
```



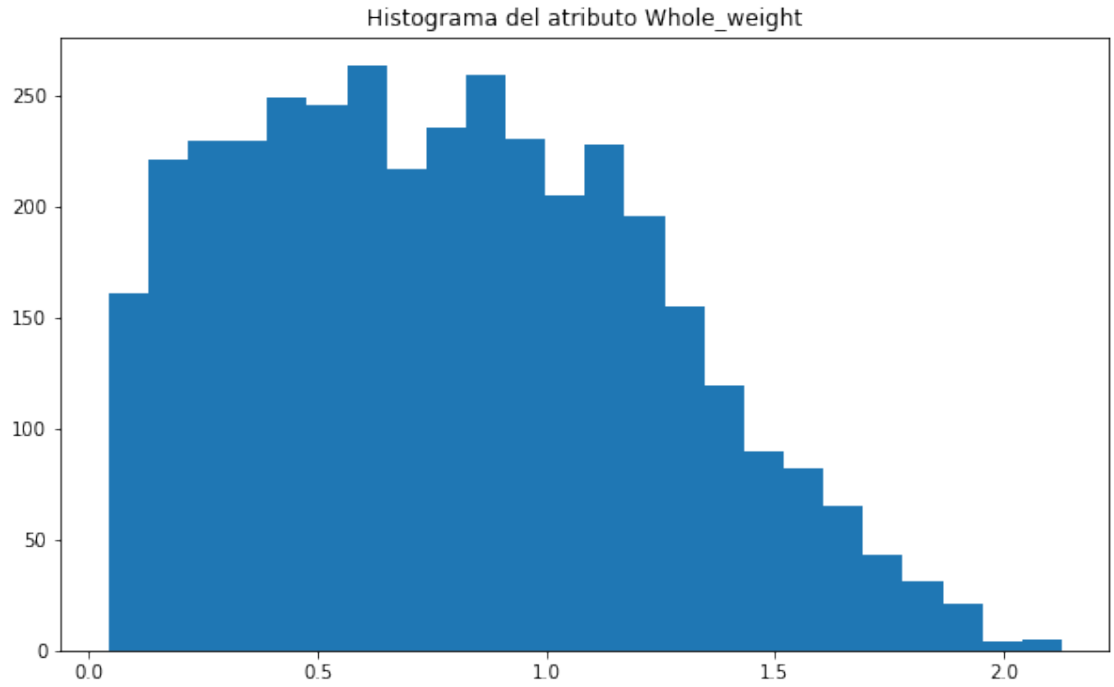
Atributo 'Height'.

[604]:
```
fig = plt.figure(figsize =(10, 6))
plt.hist( data3[:,3] , bins = 'auto' );
plt.title('Histograma del atributo Height sin muestras atípicas');
```



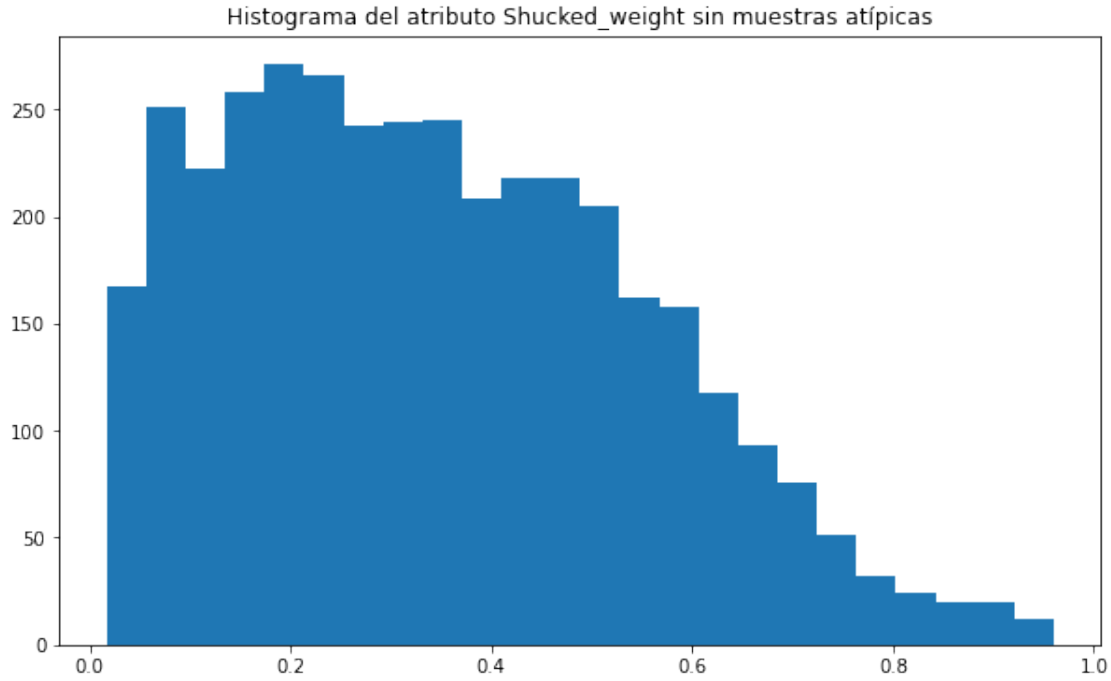
Atributo 'Whole_weight'.

[605]:
```
fig = plt.figure(figsize =(10, 6))
plt.hist( data3[:,4] , bins = 'auto' );
plt.title('Histograma del atributo Whole_weight');
```



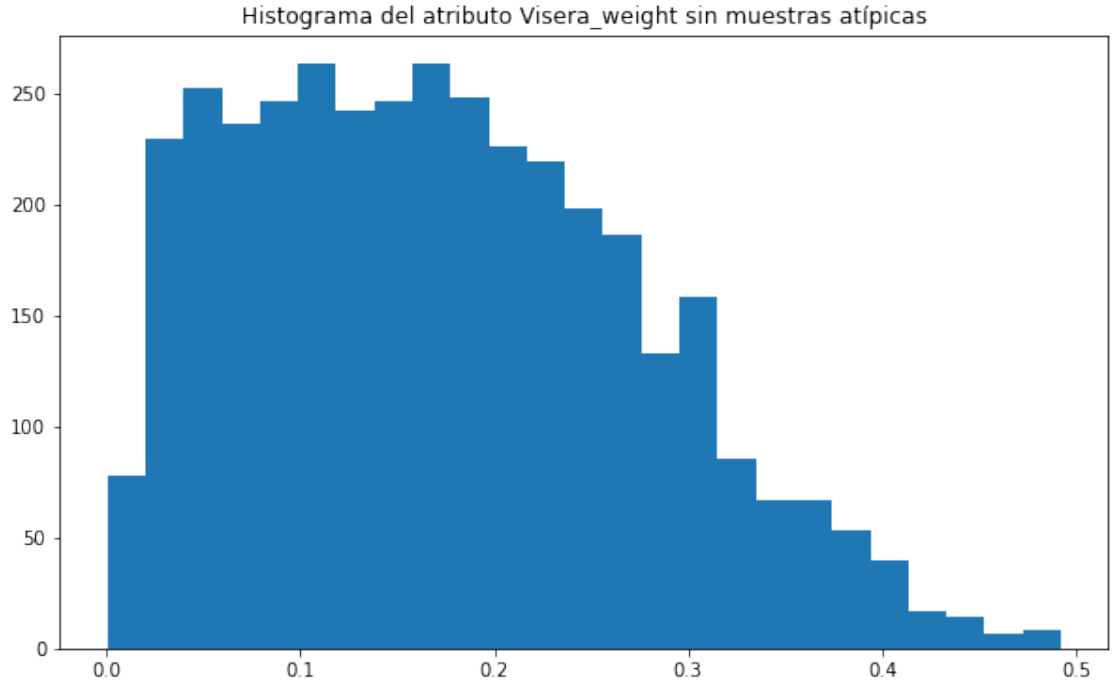
Atributo 'Shucked_weight'.

[606]:
```
fig = plt.figure(figsize =(10, 6))
plt.hist( data3[:,5] , bins = 'auto' );
plt.title('Histograma del atributo Shucked_weight sin muestras atípicas');
```



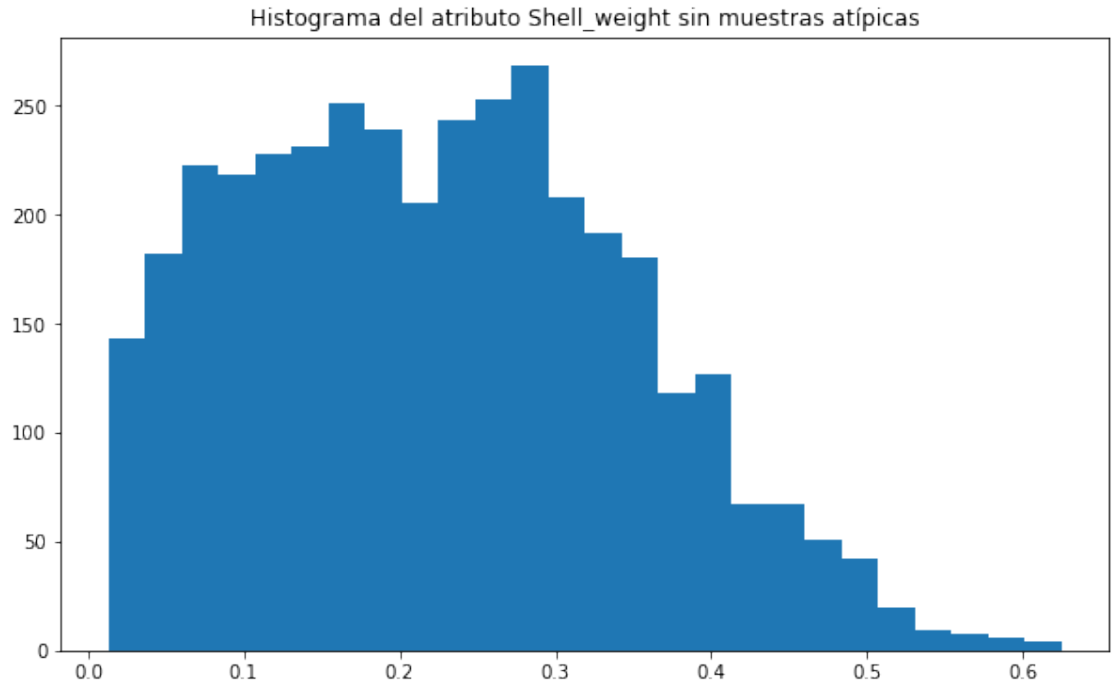
Atributo 'Visera_weight'.

[607]:
```
fig = plt.figure(figsize =(10, 6))
plt.hist( data3[:,6] , bins = 'auto' );
plt.title('Histograma del atributo Visera_weight sin muestras atípicas');
```



Atributo 'Shell_weight'.

[608]:
```
fig = plt.figure(figsize =(10, 6))
plt.hist( data3[:,7] , bins = 'auto' );
plt.title('Histograma del atributo Shell_weight sin muestras atípicas');
```



Claramente, los datos presentan comportamientos que discrepan de la tendencia de una distribución gaussiana. Por lo mismo, se plantea el transformar los datos.

1.1.4 Transformación de datos

Ahora se procede a transformar los datos para asegurar un comportamiento Gaussiano de la distribución de valores de todos los atributos continuos en aquellos donde no se presente un valor similar a una distribución Gaussiana.

En este caso las variables a transformar son todas asociadas a los pesos (weights): Whole_weight, Shucked_weight, Visera_weight, Shell_weight.

1.1.5 → Transformación raíz cuadrada

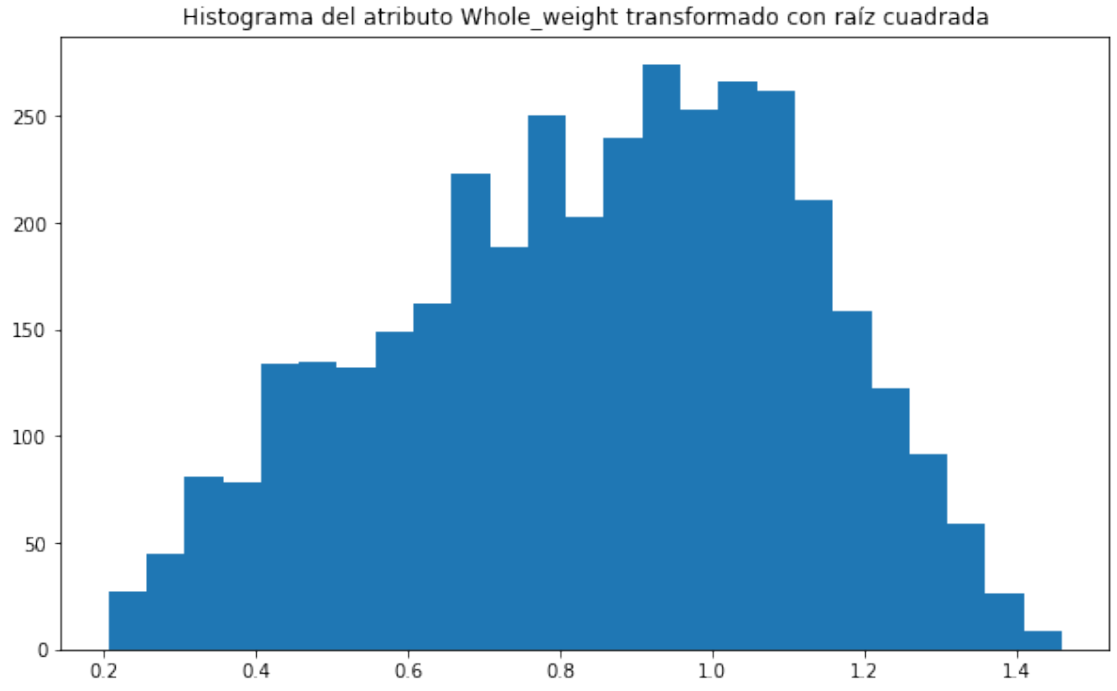
Primero se considera implementar una transformación de raíz cuadrada para todas estas variables.

$$u = \sqrt{y}$$

[609]:
```
#data4 = np.empty_like(data3)
#data4[:,:] = data3

data4 = np.copy(data3)

data4[:,4] = np.sqrt(data4[:,4])
data4[:,5] = np.sqrt(data4[:,5])
data4[:,6] = np.sqrt(data4[:,6])
data4[:,7] = np.sqrt(data4[:,7])
```

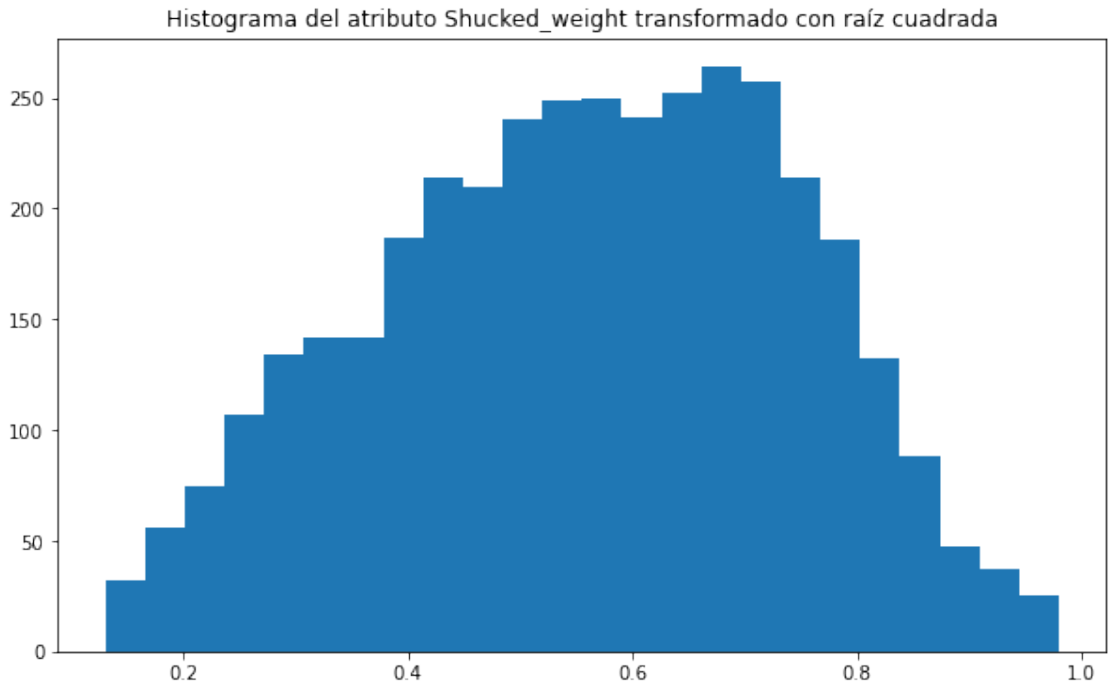
Atributo Whole_weight transformado con raíz cuadrada

[610]:
```
fig = plt.figure(figsize =(10, 6))
plt.hist( data4[:,4] , bins = 'auto' );
plt.title('Histograma del atributo Whole_weight transformado con raíz cuadrada')
plt.show()
```



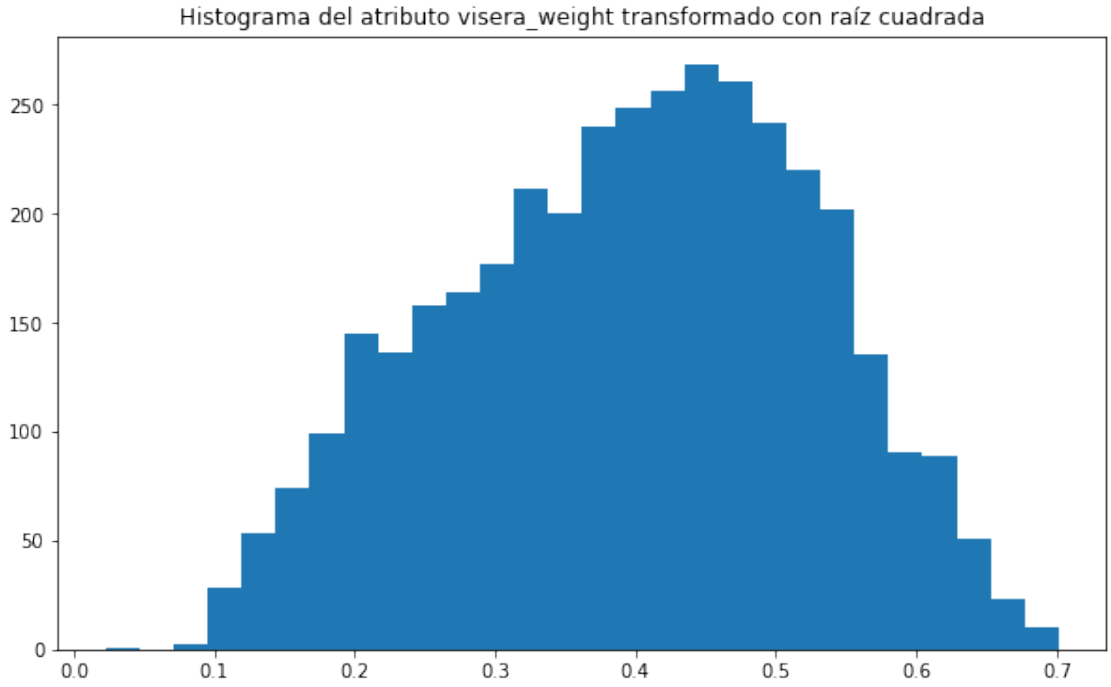
Atributo Shucked_weight transformado con raíz cuadrada

[611]:
```
fig = plt.figure(figsize =(10, 6))
plt.hist( data4[:,5] , bins = 'auto' );
plt.title('Histograma del atributo Shucked_weight transformado con raíz␣
 ↪cuadrada')
plt.show()
```



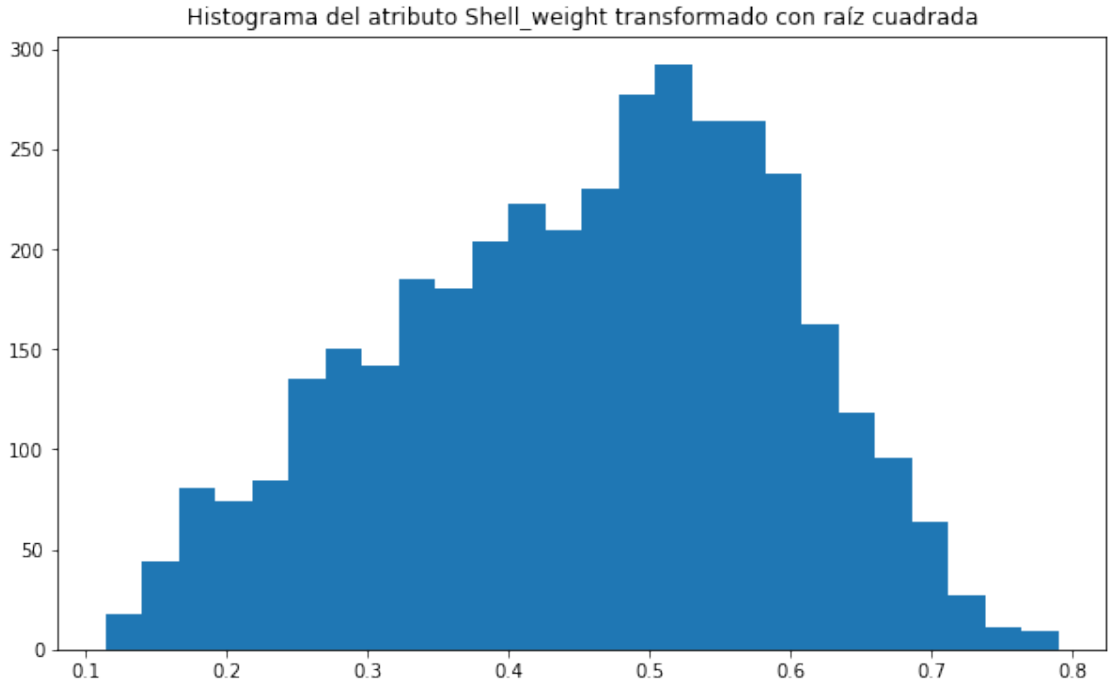
Atributo Visera_weight transformado con raíz cuadrada.

[612]:
```
fig = plt.figure(figsize =(10, 6))
plt.hist( data4[:,6] , bins = 'auto' );
plt.title('Histograma del atributo visera_weight transformado con raíz␣
 ↪cuadrada')
plt.show()
```



Atributo Shell_weight transformado con raíz cuadrada.

[613]:
```
fig = plt.figure(figsize =(10, 6))
plt.hist( data4[:,7] , bins = 'auto' );
plt.title('Histograma del atributo Shell_weight transformado con raíz cuadrada')
plt.show()
```



1.1.6 → Transformación logarítmica

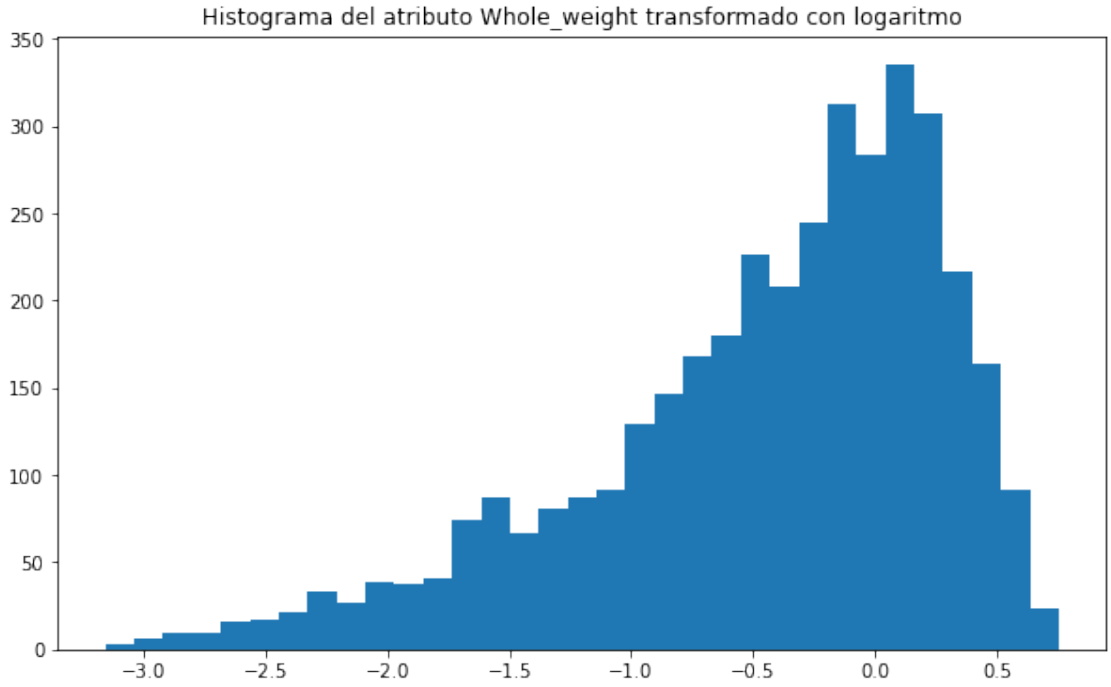
Ahora se considera implementar en los mismos atributos una transformación logaritmica.

$$u = \ln y$$

[614]:
```
#data5 = np.empty_like(data3)
#data5[:,:] = data3

data5 = np.copy(data3)

data5[:,4] = np.log(data5[:,4])
data5[:,5] = np.log(data5[:,5])
data5[:,6] = np.log(data5[:,6])
data5[:,7] = np.log(data5[:,7])
```

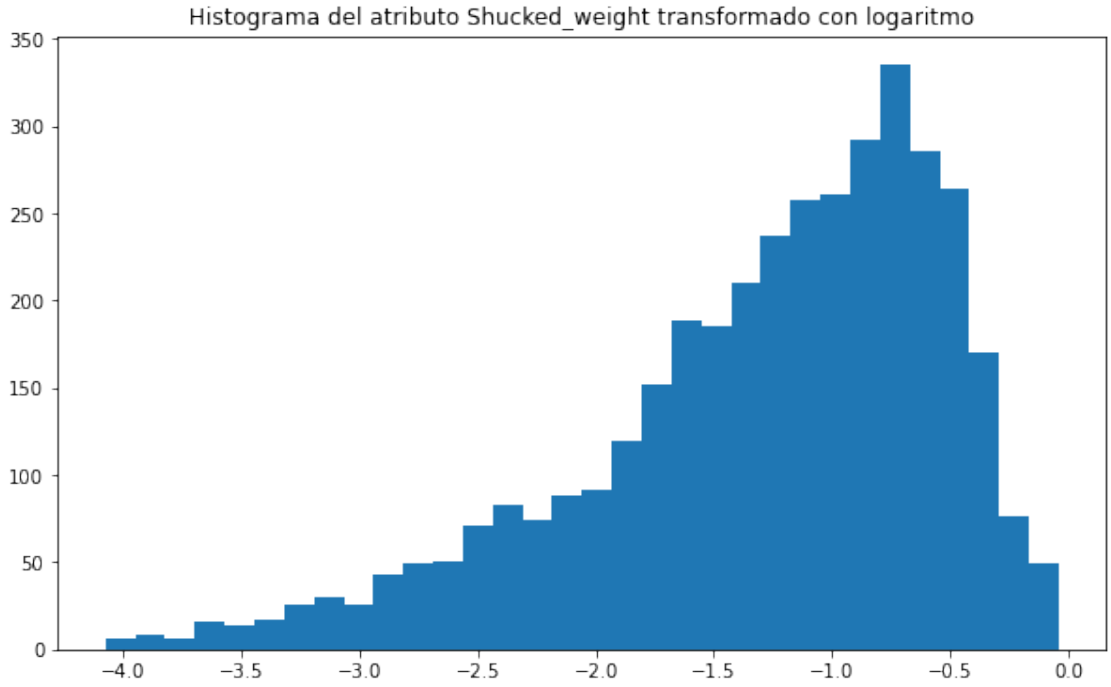
Atributo Whole_weight transformado con logaritmo neperiano.

[615]:
```
fig = plt.figure(figsize =(10, 6))
plt.hist( data5[:,4] , bins = 'auto' );
plt.title('Histograma del atributo Whole_weight transformado con logaritmo')
plt.show()
```



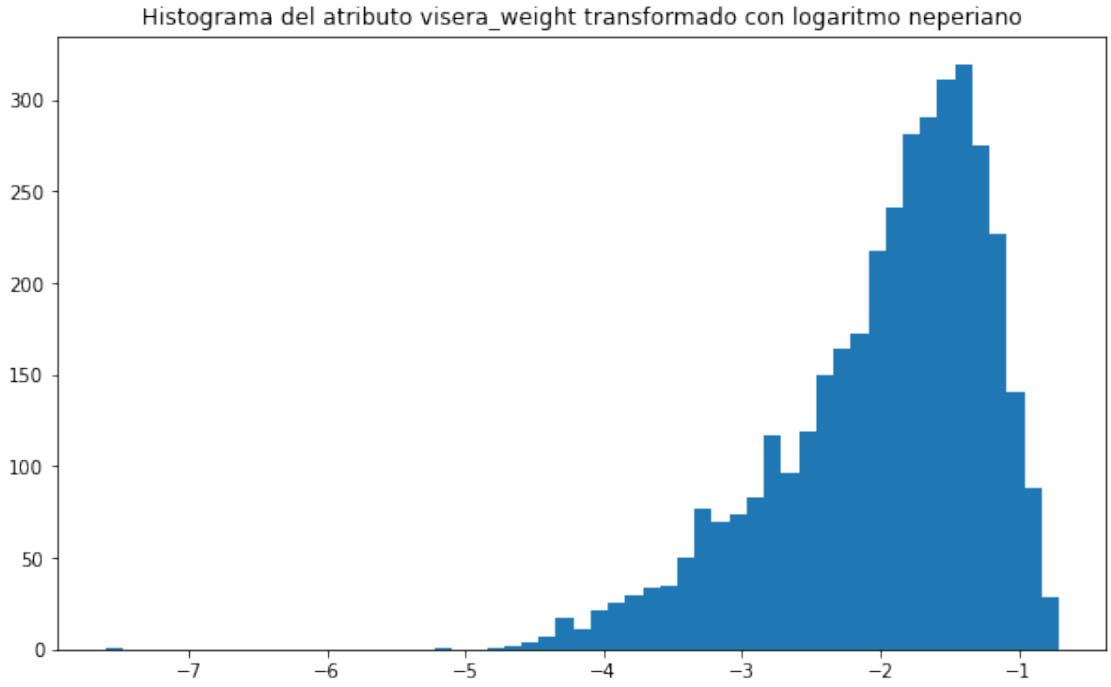
Atributo Shucked_weight transformado con logaritmo neperiano.

[616]:
```
fig = plt.figure(figsize =(10, 6))
plt.hist( data5[:,5] , bins = 'auto' );
plt.title('Histograma del atributo Shucked_weight transformado con logaritmo')
plt.show()
```



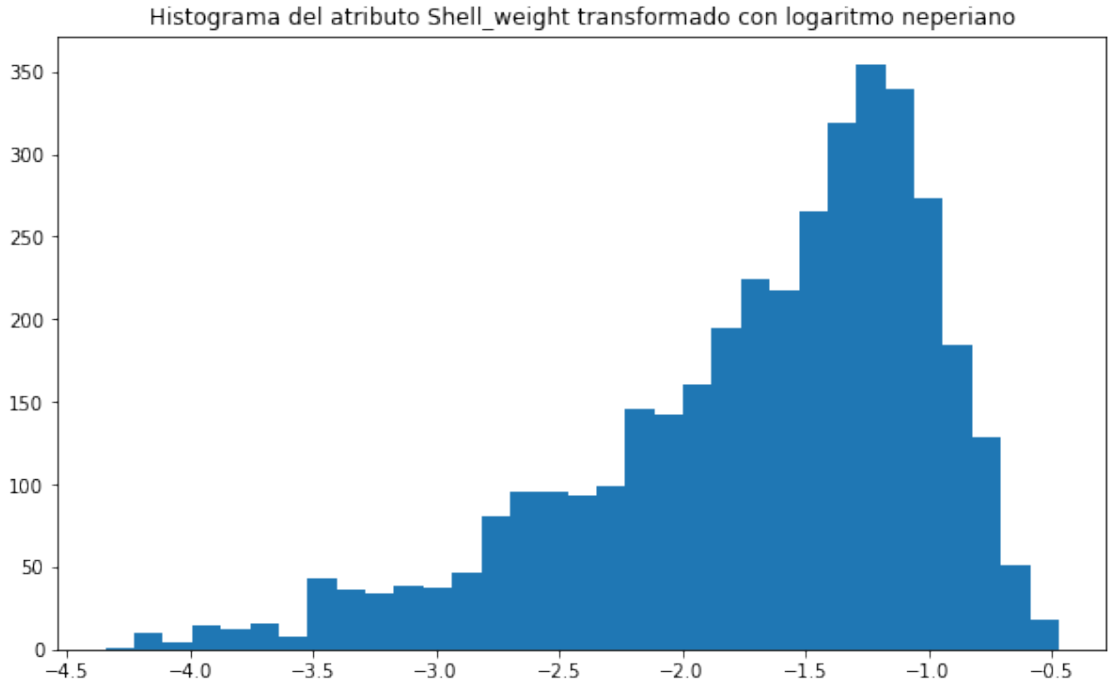
Atributo Visera_weight transformado con logaritmo neperiano.

[668]:
```
fig = plt.figure(figsize =(10, 6))
plt.hist( data5[:,6] , bins = 'auto' );
plt.title('Histograma del atributo visera_weight transformado con logaritmo␣
 ↪neperiano')
plt.show()
```



Atributo Shell_weight transformado con logaritmo neperiano.

[670]:
```
fig = plt.figure(figsize =(10, 6))
plt.hist( data5[:,7] , bins = 'auto' );
plt.title('Histograma del atributo Shell_weight transformado con logaritmo␣
 ↪neperiano')
plt.show()
```



1.1.7 → Transformación exponencial

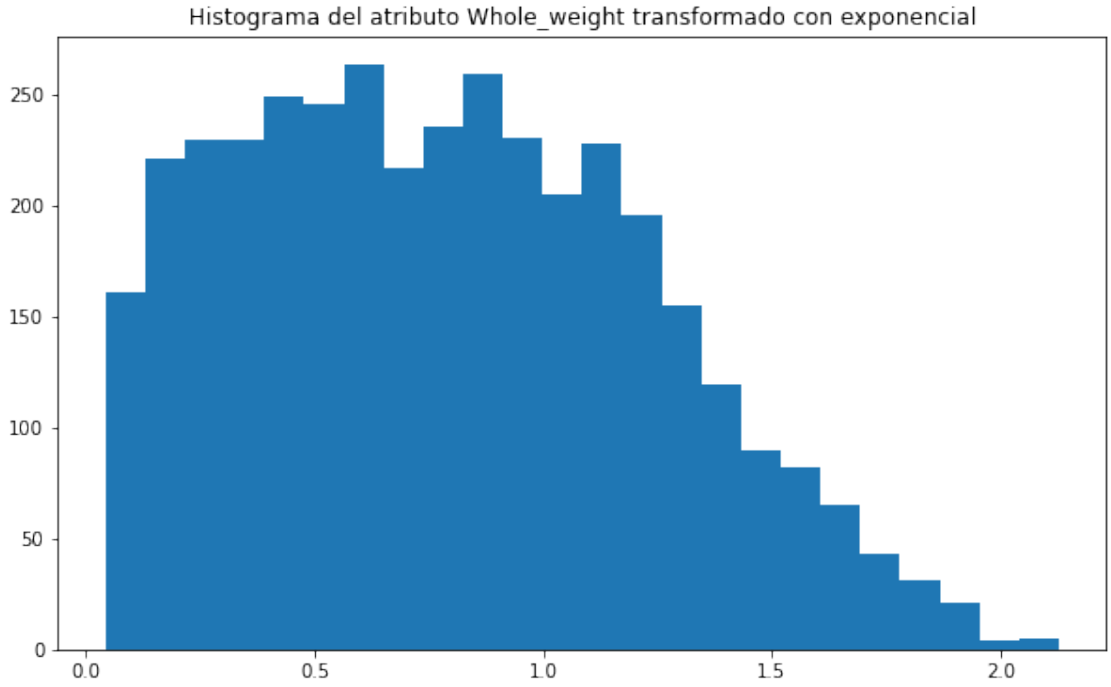
Luego, se procede a estimar el comportamiento de la distribución de los atributos Whole_weight, Shucked_weight, Visera_weight, Shell_weight al someterlos a una transformación con exponencial

$$u = e^y$$

[619]:
```
#data6 = np.empty_like(data3)
#data6[:,:] = data3

data6 = np.copy(data3)

data6[:,4] = np.exp(data5[:,4])
data6[:,5] = np.exp(data5[:,5])
data6[:,6] = np.exp(data5[:,6])
data6[:,7] = np.exp(data5[:,7])
```

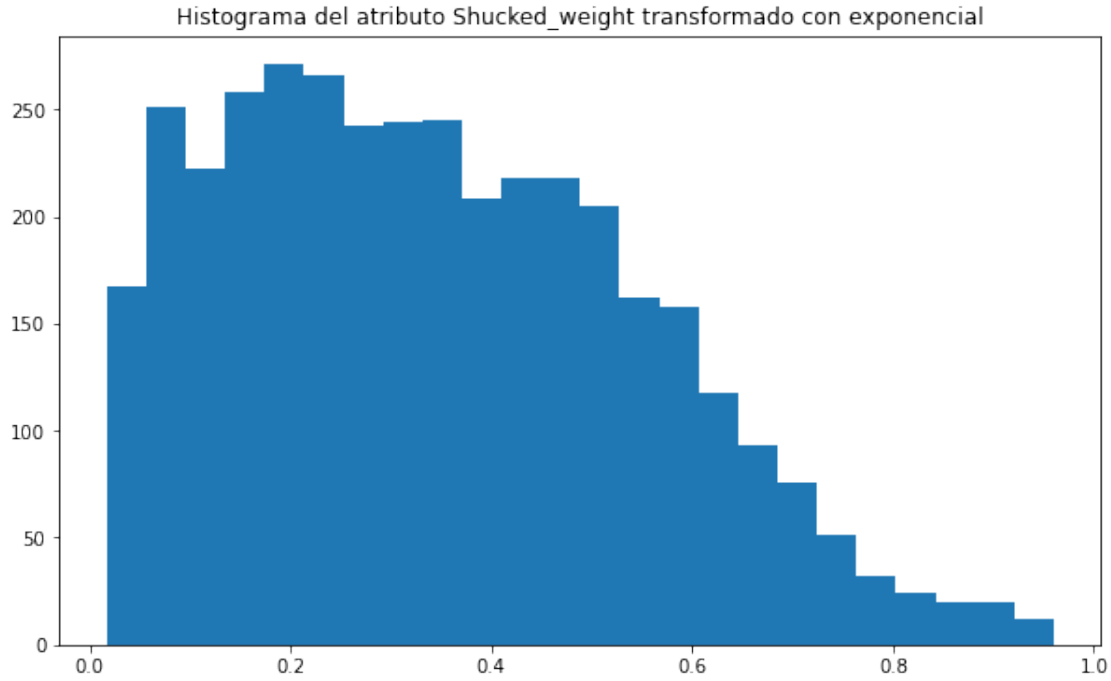
Atributo Whole_weight transformado con exponencial.

[620]:
```
fig = plt.figure(figsize = (10, 6))
plt.hist( data6[:,4] , bins = 'auto' );
plt.title('Histograma del atributo Whole_weight transformado con exponencial')
plt.show()
```



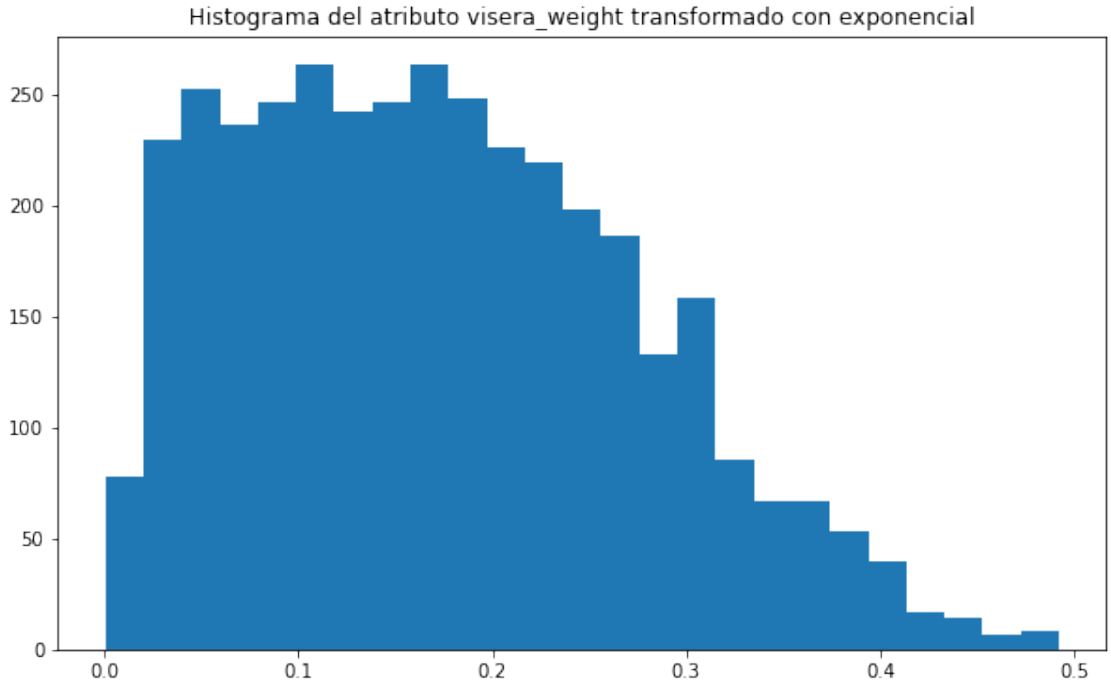
Atributo Shucked_weight transformado con logaritmo neperiano.

[621]:
```
fig = plt.figure(figsize = (10, 6))
plt.hist( data6[:,5] , bins = 'auto' );
plt.title('Histograma del atributo Shucked_weight transformado con exponencial')
plt.show()
```



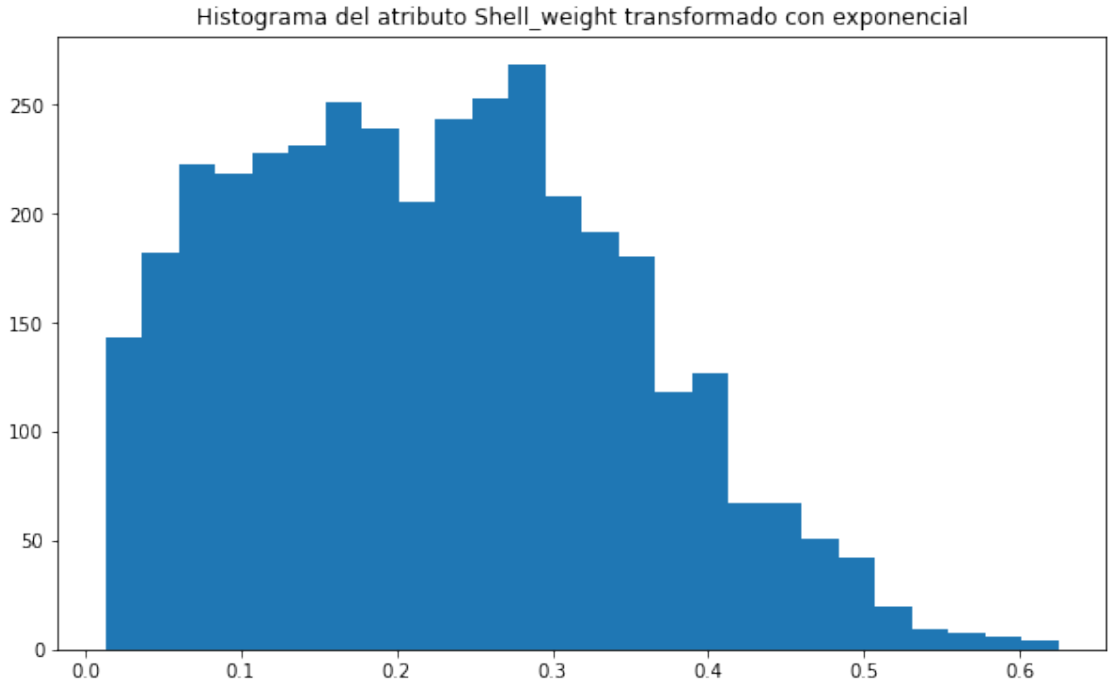
Atributo Visera_weight transformado con logaritmo neperiano.

[622]:
```
fig = plt.figure(figsize = (10, 6))
plt.hist( data6[:,6] , bins = 'auto' );
plt.title('Histograma del atributo visera_weight transformado con exponencial')
plt.show()
```



Atributo Shell_weight transformado con exponencial.

[623]:
```
fig = plt.figure(figsize = (10, 6))
plt.hist( data6[:,7] , bins = 'auto' );
plt.title('Histograma del atributo Shell_weight transformado con exponencial')
plt.show()
```



Como se puede apreciar, la transformación exponencial no muestra una mejora en la morfología de la distribución, a diferencia de la transformación raíz cuadrada y, en menor medida, de la transformación logarítmica.

En búsqueda de obtener un conjunto de datos con una distribución aún más similar a una distribución gaussiana, se procede a implementar otra transformada a los datos.

1.1.8 → Transformación Box-Cox

Por último, se consideró el implementar la transformación Box-Cox, cuya expresión se muestra a continuación

$$u = \begin{cases} \ln(y) & s \quad \lambda = 0 \\ \frac{(y^{\lambda}-1)}{\lambda} & s \quad \lambda \neq 0 \end{cases}$$

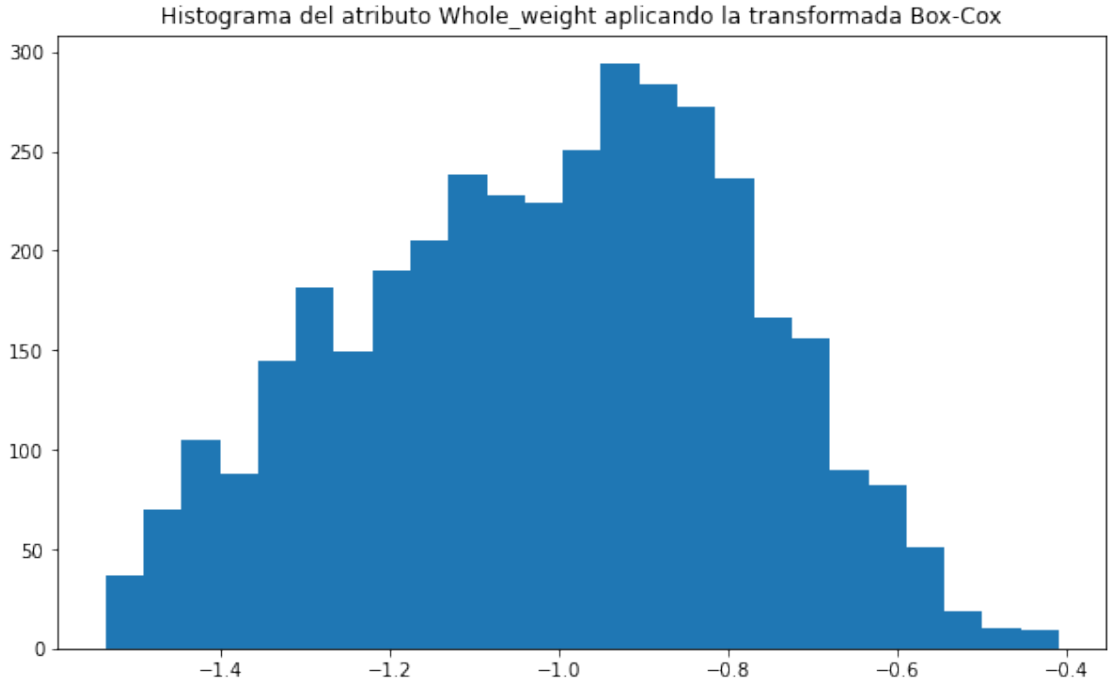
Aplicando dicha transformación a los atributos Whole_weight, Shucked_weight, Visera_weight, Shell_weight, se obtiene:

[624]:
```
#data7 = np.empty_like(data3)
#data7[:,:] = data3

data7 = np.copy(data3)

fitted_lambda = np.zeros((np.size(data3,1)))
```

```
data7[:,4], fitted_lambda[4] = stats.boxcox(data3[:,4])
data7[:,5], fitted_lambda[5] = stats.boxcox(data3[:,5])
data7[:,6], fitted_lambda[6] = stats.boxcox(data3[:,6])
data7[:,7], fitted_lambda[7] = stats.boxcox(data3[:,7])
```

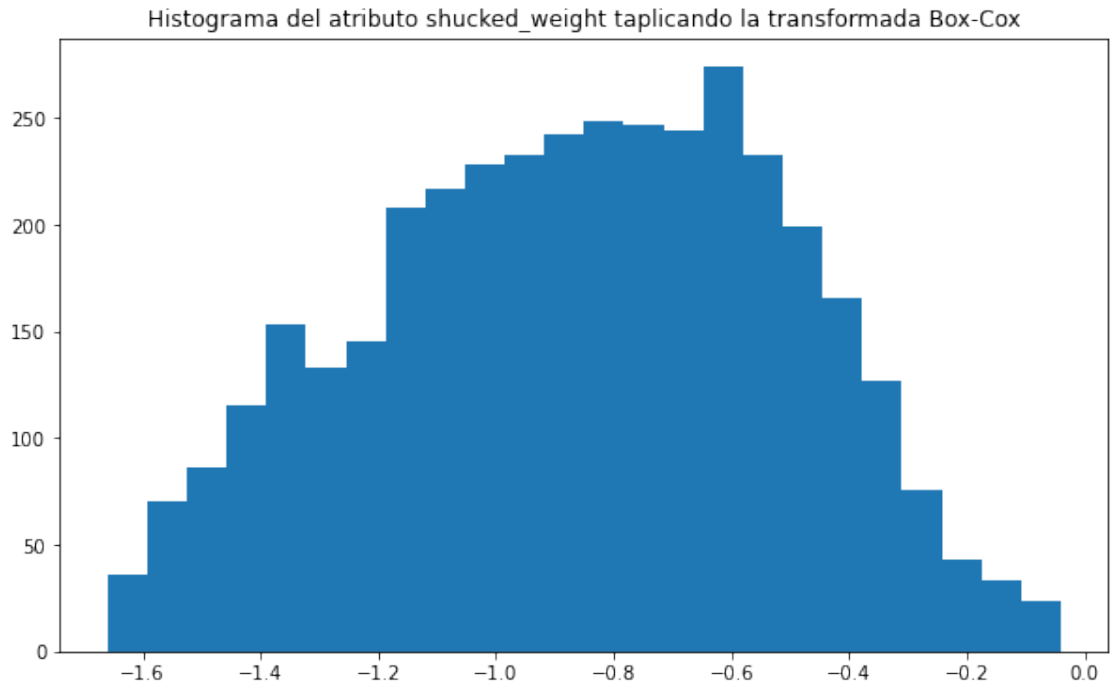
Atributo Whole_weight aplicando la transformada Box-Cox.

[625]:
```
fig = plt.figure(figsize = (10, 6))
plt.hist( data7[:,7] , bins = 'auto' );
plt.title('Histograma del atributo Whole_weight aplicando la transformada␣
 ↪Box-Cox')
plt.show()
```



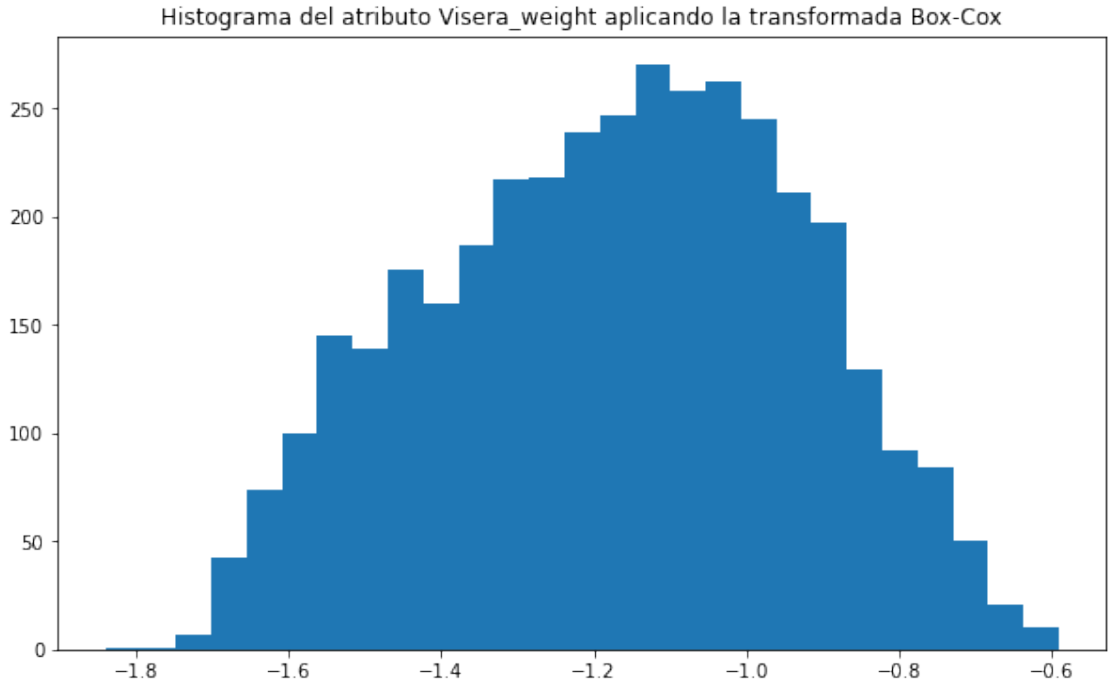
Atributo Shucked_weight aplicando la transformada Box-Cox.

[626]:
```
fig = plt.figure(figsize = (10, 6))
plt.hist( data7[:,5] , bins = 'auto' );
plt.title('Histograma del atributo shucked_weight taplicando la transformada␣
 ↪Box-Cox')
plt.show()
```



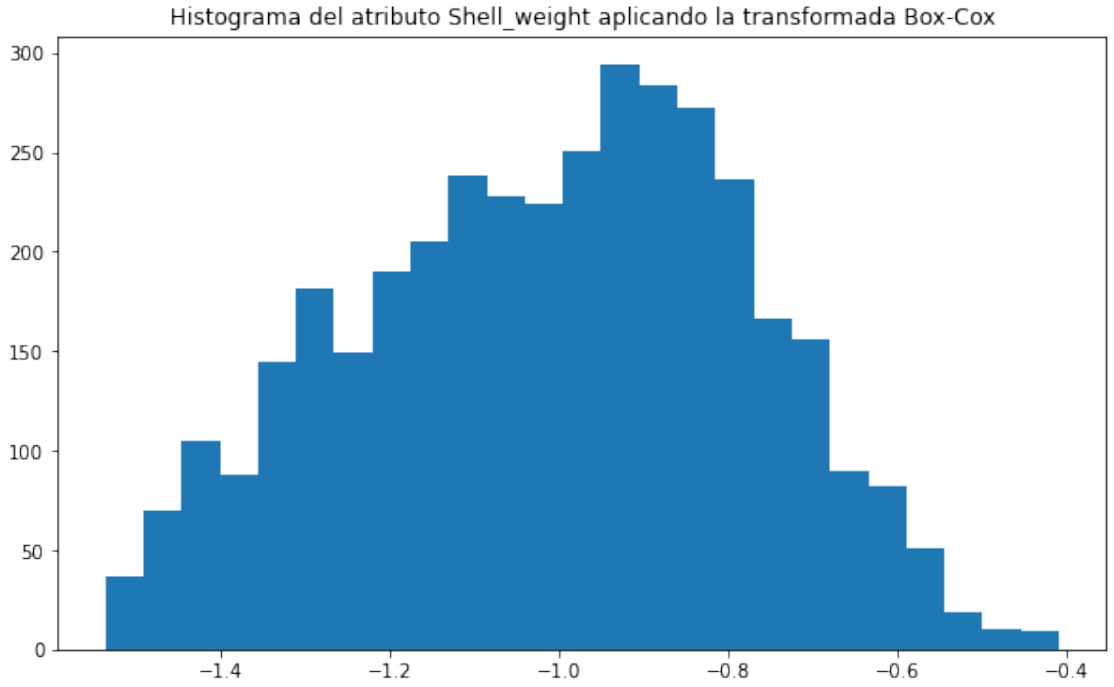
Atributo Visera_weight aplicando la transformada Box-Cox

[627]:
```
fig = plt.figure(figsize = (10, 6))
plt.hist( data7[:,6] , bins = 'auto' );
plt.title('Histograma del atributo Visera_weight aplicando la transformada␣
 ↪Box-Cox')
plt.show()
```



Atributo Shell_weight aplicando la transformada Box-Cox

[628]:
```
fig = plt.figure(figsize = (10, 6))
plt.hist( data7[:,7] , bins = 'auto' );
plt.title('Histograma del atributo Shell_weight aplicando la transformada␣
 ↪Box-Cox')
plt.show()
```



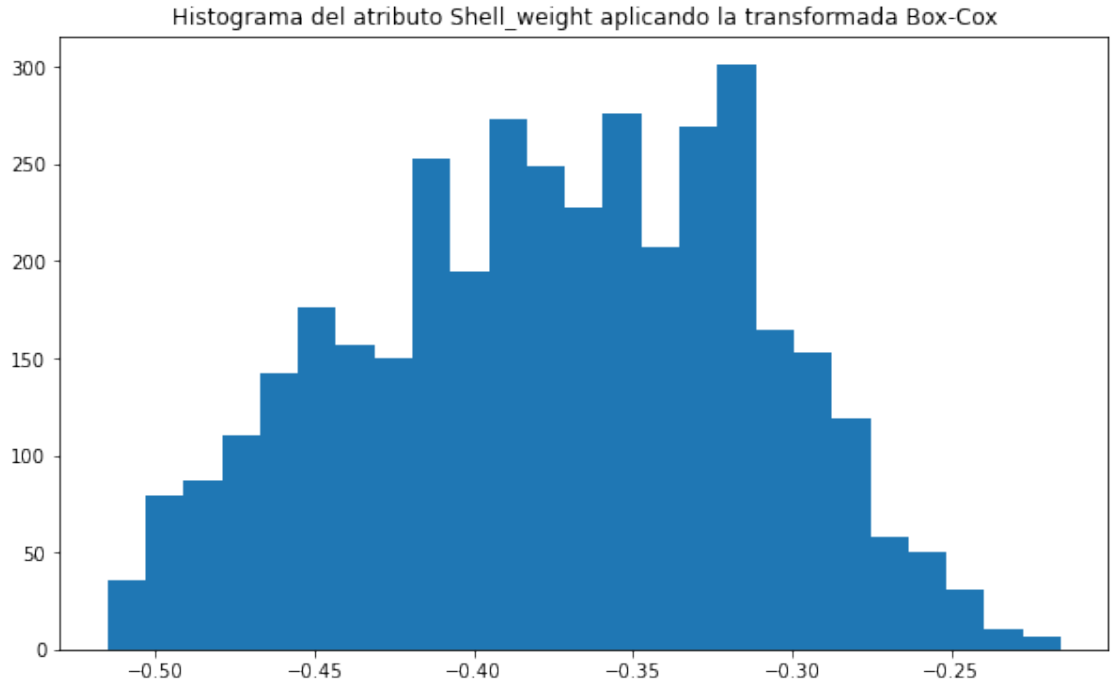
Claramente se puede apreciar que la transformación Box-Cox arroja distribuciones más cercanas a una distribución gaussiana.

Aplicando dicha transformación sobre los 3 atributos restantes (Length, Diameter y Height), se tiene:

[629]:
```
data7[:,1], fitted_lambda[1] = stats.boxcox(data3[:,1])
data7[:,2], fitted_lambda[2] = stats.boxcox(data3[:,2])
data7[:,3], fitted_lambda[3] = stats.boxcox(data3[:,3])
```

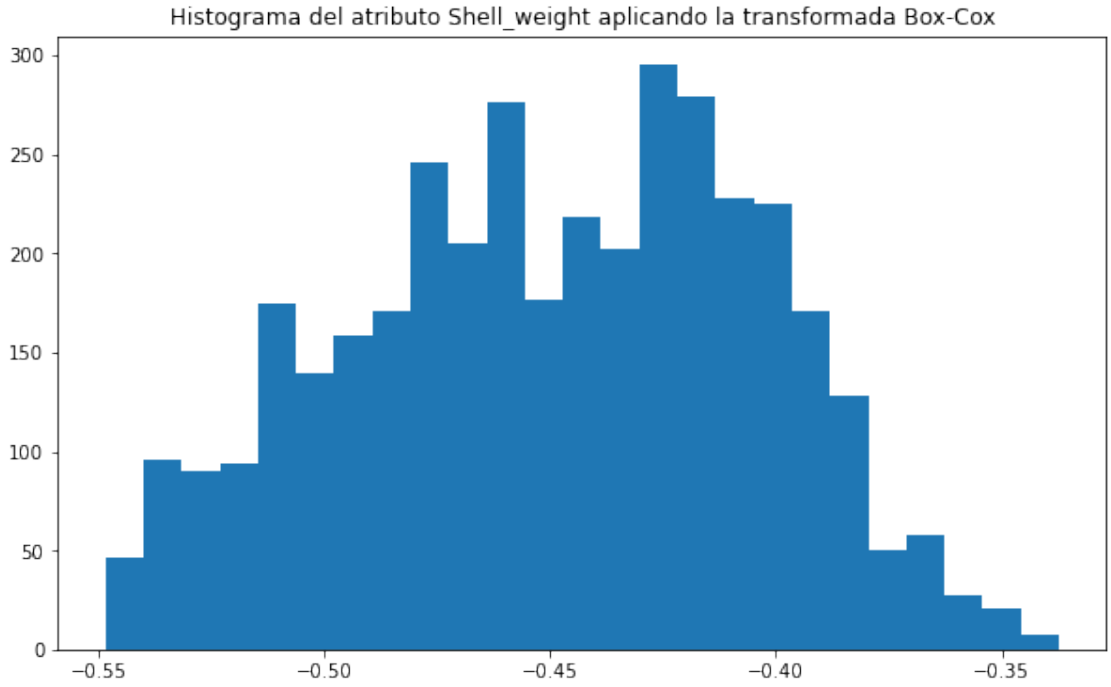
Atributo Length aplicando la transformada Box-Cox

[630]:
```
fig = plt.figure(figsize = (10, 6))
plt.hist( data7[:,1] , bins = 'auto' );
plt.title('Histograma del atributo Shell_weight aplicando la transformada␣
 ↪Box-Cox')
plt.show()
```



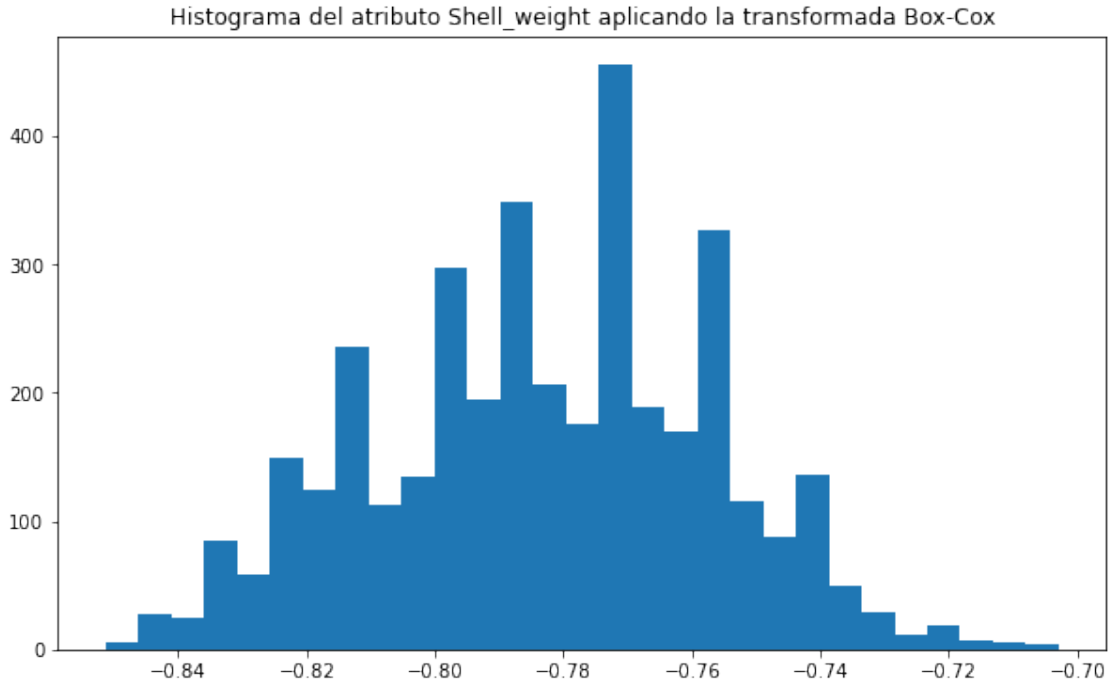
Atributo Diameter aplicando la transformada Box-Cox

[631]:
```
fig = plt.figure(figsize = (10, 6))
plt.hist( data7[:,2] , bins = 'auto' );
plt.title('Histograma del atributo Shell_weight aplicando la transformada␣
 ↪Box-Cox')
plt.show()
```



Atributo Height aplicando la transformada Box-Cox

[632]:
```
fig = plt.figure(figsize = (10, 6))
plt.hist( data7[:,3] , bins = 'auto' );
plt.title('Histograma del atributo Shell_weight aplicando la transformada␣
 ↪Box-Cox')
plt.show()
```



Como se puede apreciar, se obtiene una ligera mejoría en la distribución de datos de las variables 'Length', 'Diameter' y 'Height' aplicando la transformada Box-Cox sobre éstas.

Los valores obtenidos de los parámetros de interés para los datos sometidos a la tranformación Box-Cox se muestran a continuación.

[633]:
```
#cálculo de los parámetros estadísticos

Data7Mean = np.mean(data7, axis = 0)
Data7Median = np.median(data7, axis = 0)
Data7Mode = stats.mode(data7, axis = 0)
Data7Max = np.max(data7, axis = 0)
Data7Min = np.min(data7, axis = 0)
Data7Range = Data7Max - Data7Min
Data7Desv = np.std(data7, axis = 0)
Data7Skew = stats.skew(data7, axis = 0, bias = 0)
Data7Kurt = stats.kurtosis(data7, axis = 0, bias = 0)
```

[634]:
```
#Presentación de los valores de los parámetros estadísticos estudiados

print('la media de los atributos transformandos con Box-Cox es:');
print(Data7Mean);
print();
print('la mediana de los atributos transformandos con Box-Cox es:');
print(Data7Median);
```

```
print();
print('la moda de los atributos transformandos con Box-Cox es:');
print(Data7Mode);
print();
print('el valor máximo de los atributos transformandos con Box-Cox es:');
print(Data7Max);
print();
print('el valor mínimo de los atributos transformandos con Box-Cox es:');
print(Data7Min);
print();
print('lel rango de los atributos transformandos con Box-Cox es:');
print(Data7Range);
print();
print('la desviación estándar de los atributos transformandos con Box-Cox es:');
print(Data7Desv);
print();
print('la asimetría de los atributos transformandos con Box-Cox es:');
print(Data7Skew);
print();
print('el curtosis de los atributos transformandos con Box-Cox es:');
print(Data7Kurt);
print();
```

```
la media de los atributos transformandos con Box-Cox es:
[ 0.02618355 -0.37429714 -0.44923381 -0.78283052 -0.28389804 -0.85986133
 -1.17246937 -1.00827998  9.43057392]

la mediana de los atributos transformandos con Box-Cox es:
[ 0.         -0.37194064 -0.44559486 -0.78133415 -0.24632203 -0.84175878
 -1.15950397 -0.99211595  9.        ]

la moda de los atributos transformandos con Box-Cox es:
ModeResult(mode=array([[ 1.        , -0.31486558, -0.4295832 , -0.77377901,
-1.00084226,
        -1.13439466, -1.14226961, -0.89641613,  9.        ]]),
count=array([[1360,   89,  129,  255,    8,   11,   15,   40,  685]]))

el valor máximo de los atributos transformandos con Box-Cox es:
[ 1.         -0.21558696 -0.33737971 -0.70287203  0.94852647 -0.04038032
 -0.59042323 -0.40916605 15.        ]

el valor mínimo de los atributos transformandos con Box-Cox es:
[-1.         -0.51501669 -0.54839027 -0.85134036 -1.44265423 -1.66021343
 -1.84032925 -1.53515665  4.        ]

lel rango de los atributos transformandos con Box-Cox es:
[ 2.          0.29942974  0.21101055  0.14846833  2.3911807   1.61983311
```

```
  1.24990601  1.1259906  11.        ]

la desviación estándar de los atributos transformandos con Box-Cox es:
[0.83217623 0.06197681 0.04475877 0.02629157 0.51460648 0.35070216
 0.2409268  0.23114727 2.33013364]

la asimetría de los atributos transformandos con Box-Cox es:
[-0.04900827 -0.12738934 -0.13477452 -0.06390032 -0.12853352 -0.11413007
 -0.11322562 -0.11972264  0.26920452]

el curtosis de los atributos transformandos con Box-Cox es:
[-1.55480736 -0.74652043 -0.76489945 -0.4929999  -0.78182313 -0.73667595
 -0.73430947 -0.71107438 -0.21464546]
```

1.1.9 Normalización de los datos transformados con Box-Cox

Posterior a la transformación de los datos, se procede a la normalización de los mismos. Esto se realiza en este orden para asegurar la estandarización (en términos de rango y orden de magnitud) de los datos transformados.

La metodología de normalización será la técnica Z-Score. La ecuación corresponde a:

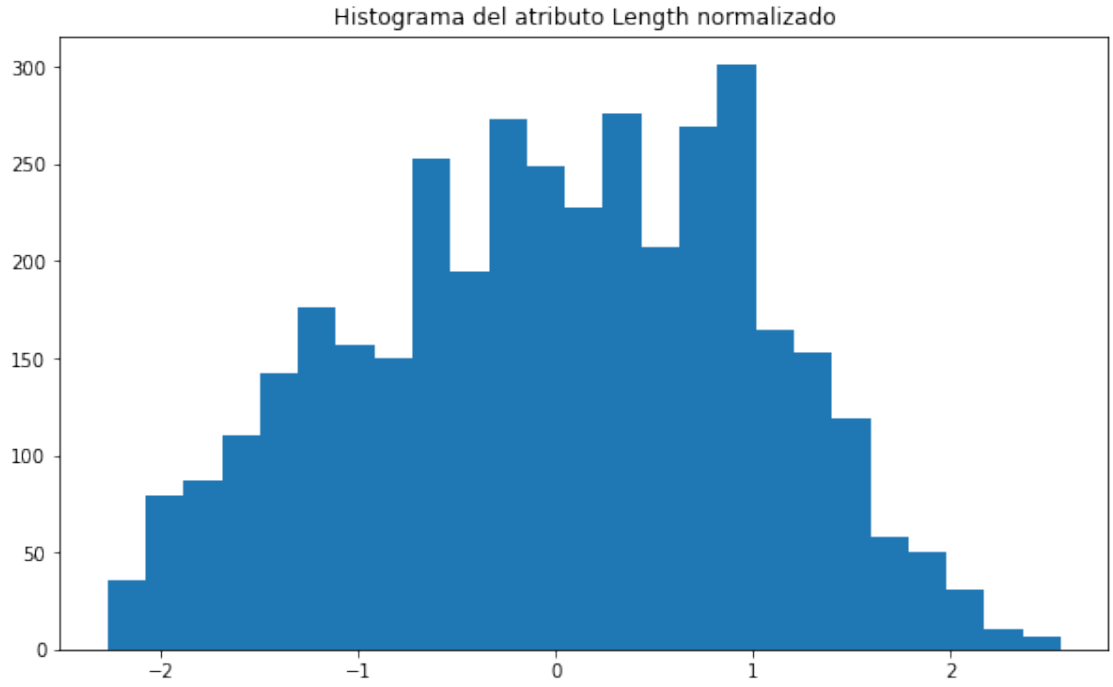
$$x_N = \frac{(x - \mu)}{\sigma}$$

La misma se implementará solo en los atributos de entrada de naturaleza continua.

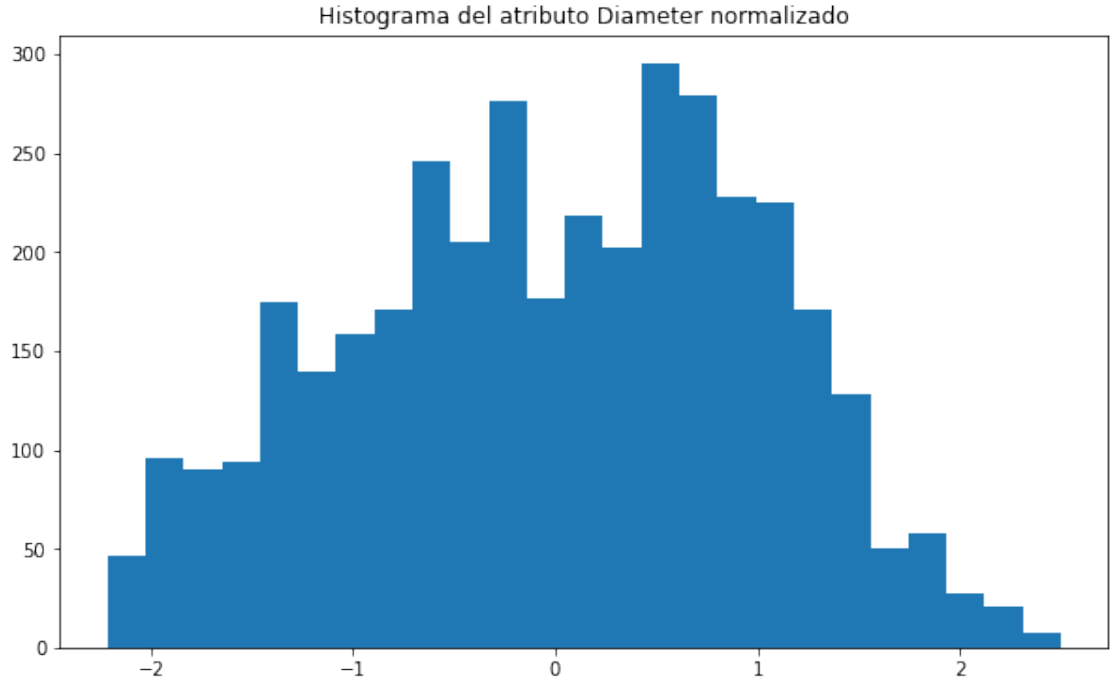
[639]:
```
#Aplicando la normalización Z-Score para todos los atributo excepto Sex y Rings.

data7N = np.empty_like(data3)

data7N = data7;

data7N[:,1:8] = (data7N[:,1:8] - ( Data7Mean[1:8].transpose() * np.ones( (np.
↪size(data7N[:,1:8],0),np.size(data7N[:,1:8],1)) ) )  ) / ( Data7Desv[1:8].
↪transpose()  * np.ones( (np.size(data7N[:,1:8],0), np.size(data7N[:,1:8],1))␣
↪) )
```

Se procede a mostrar los histogramas de las variables normalizadas.

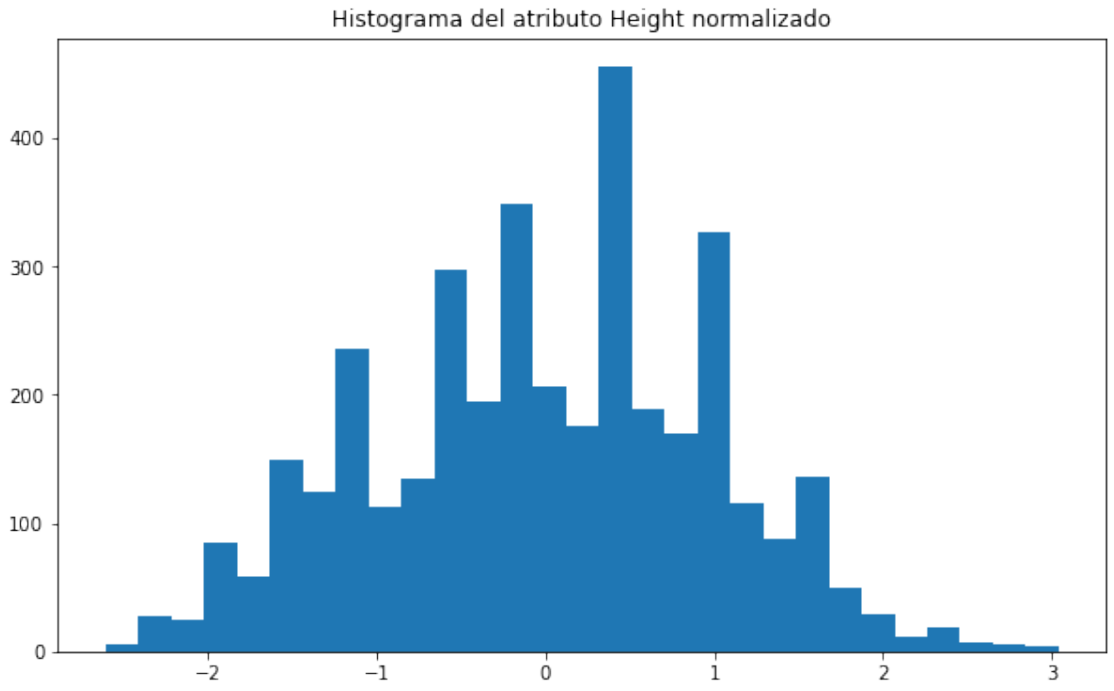
[640]:
```
fig = plt.figure(figsize = (10, 6))
plt.hist( data7N[:,1] , bins = 'auto' );
plt.title('Histograma del atributo Length normalizado')
plt.show()
```



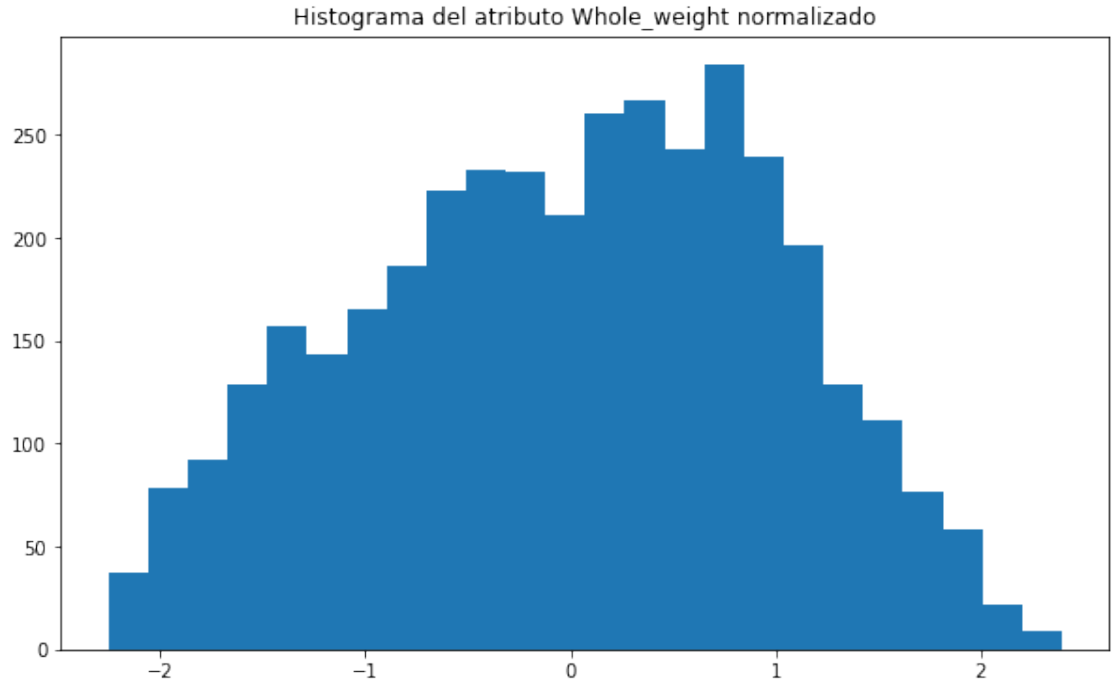
[641]:
```
fig = plt.figure(figsize = (10, 6))
plt.hist( data7N[:,2] , bins = 'auto' );
plt.title('Histograma del atributo Diameter normalizado')
plt.show()
```



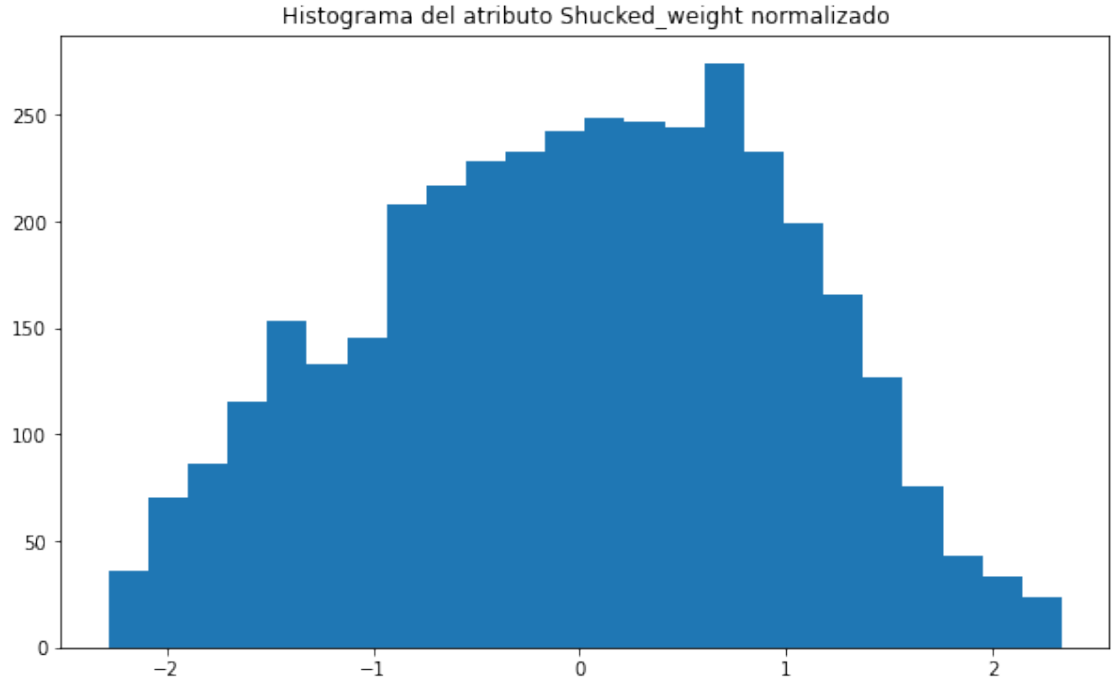
[642]:
```
fig = plt.figure(figsize = (10, 6))
plt.hist( data7N[:,3] , bins = 'auto' );
plt.title('Histograma del atributo Height normalizado')
plt.show()
```



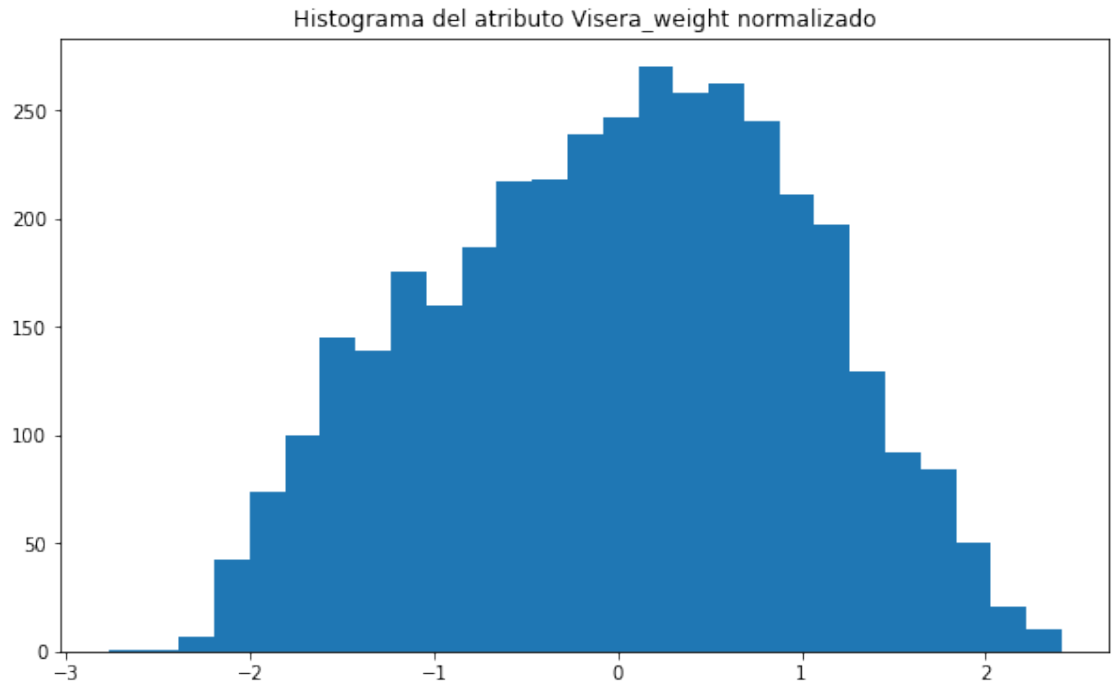
[643]:
```
fig = plt.figure(figsize = (10, 6))
plt.hist( data7N[:,4] , bins = 'auto' );
plt.title('Histograma del atributo Whole_weight normalizado')
plt.show()
```



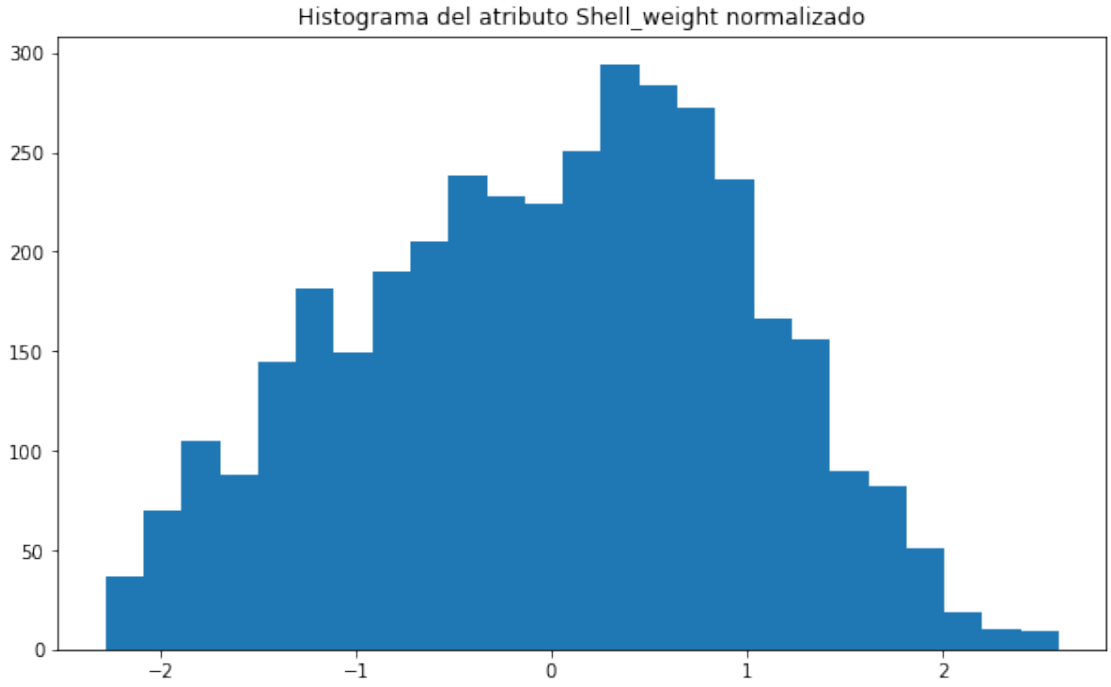
[644]:
```
fig = plt.figure(figsize = (10, 6))
plt.hist( data7N[:,5] , bins = 'auto' );
plt.title('Histograma del atributo Shucked_weight normalizado')
plt.show()
```



[645]:
```
fig = plt.figure(figsize = (10, 6))
plt.hist( data7N[:,6] , bins = 'auto' );
plt.title('Histograma del atributo Visera_weight normalizado')
plt.show()
```



[646]:
```
fig = plt.figure(figsize = (10, 6))
plt.hist( data7N[:,7] , bins = 'auto' );
plt.title('Histograma del atributo Shell_weight normalizado')
plt.show()
```



Como se puede apreciar para todas las distribución posterior al proceso de normalización Z-Score, los valores no están dentro del rango $[-1, 1]$, sino dentro del rango $[-2.698, 2.698]$. Esto resulta ser consecuencia del método usado para desechar los valores atípicos, IQR, que mantiene los valores típcios dentro del rango $[-2.698\sigma, 2.698\sigma]$.

Se pudiera modificar la expresión de Z-Score, haciendo el denominador igual a 2.698σ, de modo que se garantize que todos los valores se encontrarán dentro del intervalo $[-1, 1]$.

Lo relevante es que todos los valores de todos los atributos continuos tienen el mismo orden de magnitud dentro del mismo rango ($[-2.698\sigma, 2698\sigma]$).

1.1.10 → Cálculo de los parámetros estadísticos de los datos normalizados.

Por último, se recalculan los valores de los parámetros estadísticos para los datos normalizados

[647]:
```
#cálculo de los parámetros estadísticos

Data7NMean = np.mean(data7N, axis = 0)
Data7NMedian = np.median(data7N, axis = 0)
Data7NMode = stats.mode(data7N, axis = 0)
Data7NMax = np.max(data7N, axis = 0)
Data7NMin = np.min(data7N, axis = 0)
Data7NRange = Data7NMax - Data7NMin
Data7NDesv = np.std(data7N, axis = 0)
Data7NSkew = stats.skew(data7N, axis = 0, bias = 0)
Data7NKurt = stats.kurtosis(data7N, axis = 0, bias = 0)
```

[671]:
```
#Presentación de los valores de los parámetros estadísticos estudiados

print('la media de los atributos normalizados con Box-Cox es:');
print(Data7NMean);
print();
print('la mediana de los atributos normalizandos con Box-Cox es:');
print(Data7NMedian);
print();
print('la moda de los atributos normalizandos con Box-Cox es:');
print(Data7NMode);
print();
print('el valor máximo de los atributos normalizandos con Box-Cox es:');
print(Data7NMax);
print();
print('el valor mínimo de los atributos normalizandos con Box-Cox es:');
print(Data7NMin);
print();
print('lel rango de los atributos normalizandos con Box-Cox es:');
print(Data7NRange);
print();
print('la desviación estándar de los atributos normalizandos con Box-Cox es:');
print(Data7NDesv);
print();
print('la asimetría de los atributos normalizandos con Box-Cox es:');
print(Data7NSkew);
print();
print('el curtosis de los atributos normalizandos con Box-Cox es:');
print(Data7NKurt);
print();
```

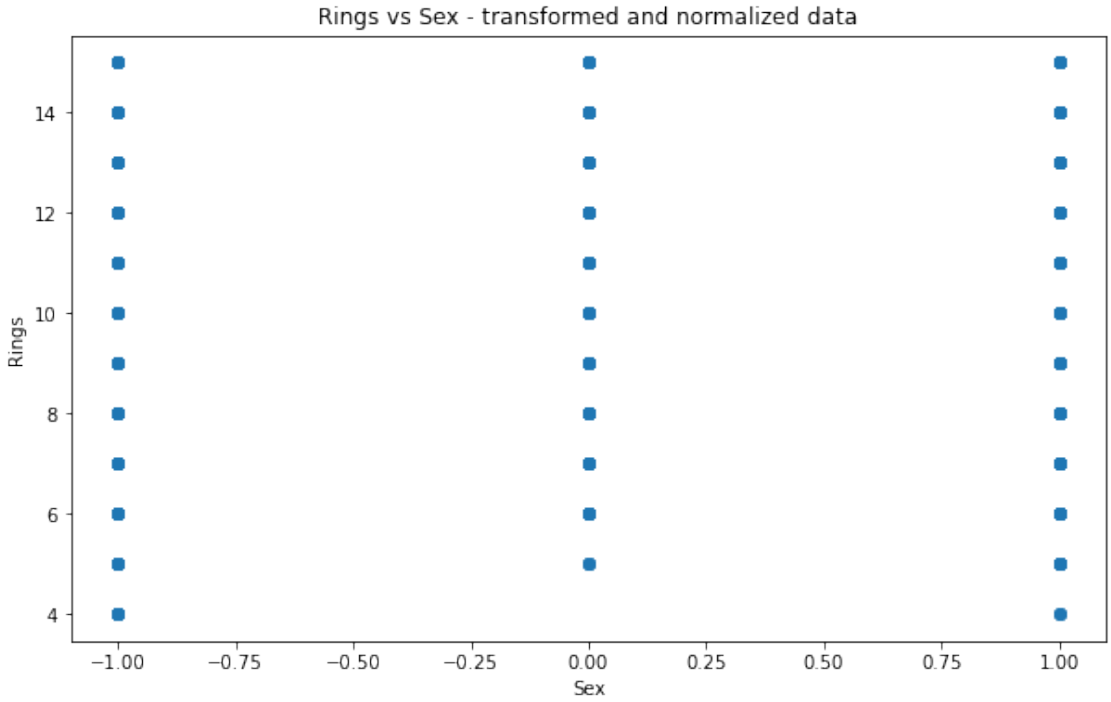
```
la media de los atributos normalizados con Box-Cox es:
[ 2.61835493e-02 -2.52917085e-15 -7.32637378e-14  1.77313158e-13
 -3.01853814e-17  3.53885424e-15  5.72570878e-15  8.74318984e-16
  9.43057392e+00]

la mediana de los atributos normalizandos con Box-Cox es:
[0.         0.03802239 0.0813012  0.0569143  0.07301892 0.05161802
 0.0538147  0.06992956 9.        ]

la moda de los atributos normalizandos con Box-Cox es:
ModeResult(mode=array([[ 1.        ,  0.95893217,  0.43903373,  0.34427402,
-1.39318925,
        -0.78281047,  0.12534829,  0.48395057,  9.        ]]),
count=array([[1360,   89,  129,  255,    8,   11,   15,   40,  685]]))

el valor máximo de los atributos normalizandos con Box-Cox es:
[ 1.          2.56079937  2.49904296  3.04122157  2.39488729  2.33668652
```

```
  2.41586295  2.59191436 15.        ]

el valor mínimo de los atributos normalizandos con Box-Cox es:
[-1.         -2.27051927 -2.2153526  -2.6057724  -2.2517326  -2.2821419
 -2.77204472 -2.27939825  4.        ]

lel rango de los atributos normalizandos con Box-Cox es:
[ 2.          4.83131864  4.71439556  5.64699397  4.6466199   4.61882842
  5.18790766  4.87131261 11.        ]

la desviación estándar de los atributos normalizandos con Box-Cox es:
[0.83217623 1.         1.         1.         1.         1.
 1.         1.         2.33013364]

la asimetría de los atributos normalizandos con Box-Cox es:
[-0.04900827 -0.12738934 -0.13477452 -0.06390032 -0.12853352 -0.11413007
 -0.11322562 -0.11972264  0.26920452]

el curtosis de los atributos normalizandos con Box-Cox es:
[-1.55480736 -0.74652043 -0.76489945 -0.4929999  -0.78182313 -0.73667595
 -0.73430947 -0.71107438 -0.21464546]
```

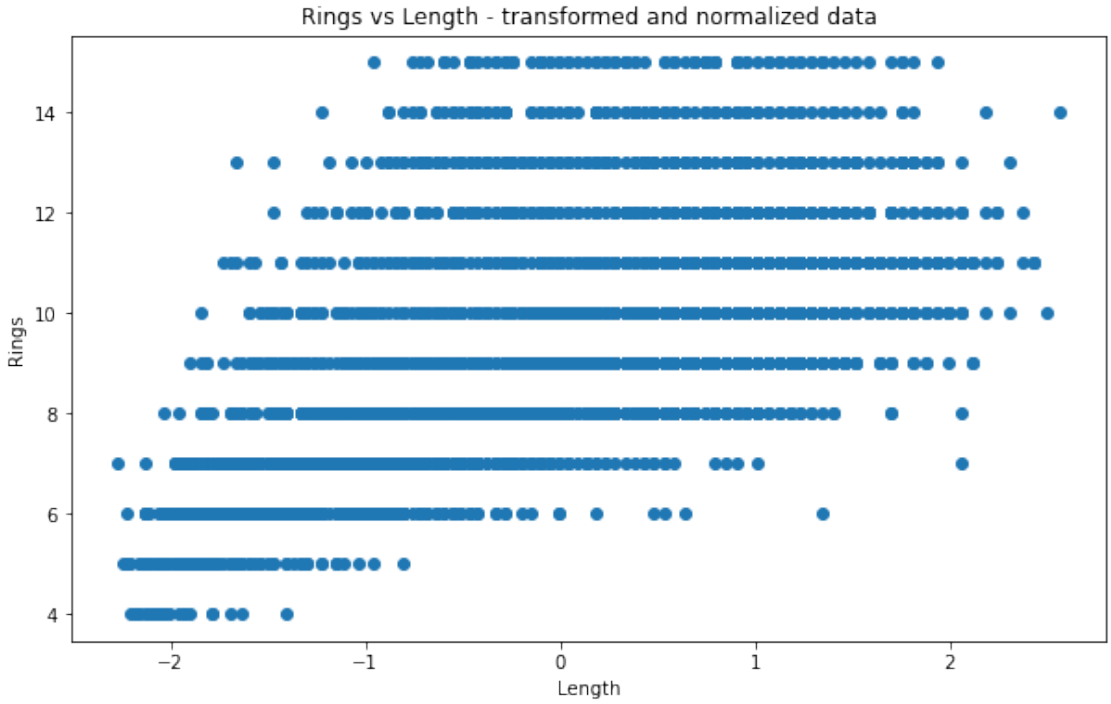
1.2 Estudio del comportamiento entre los datos

Con los datos transformados y normalizados, se procede a graficar los diagramas de dispersión (scatter plots) de cada una de los atributos a considerar como variables (Sex, Length, Diameter, Whole_weight, Shucked_weight, Visera_weight y Shell_weight) respecto al atributo a considerar como salida del sistema (Rigns).

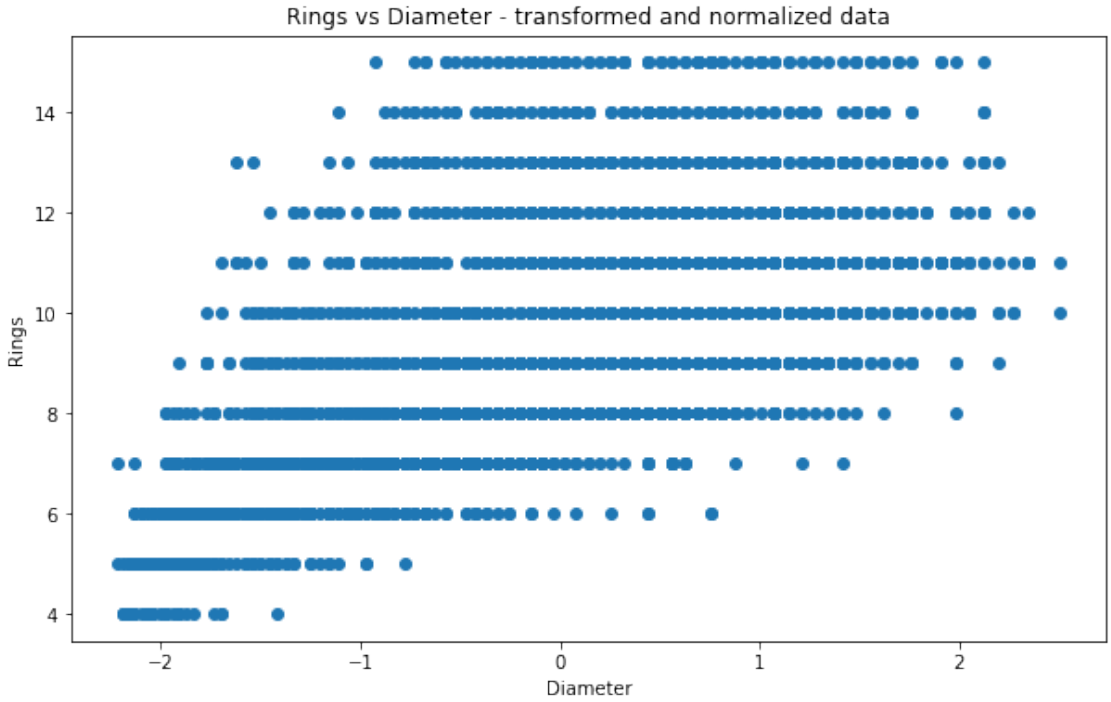
[649]:
```
#Scatter plot Rings vs Sex

fig = plt.figure(figsize = (10, 6))
plt.scatter(data7N[:,0], data7N[:,8])
plt.title('Rings vs Sex - transformed and normalized data')
plt.xlabel('Sex')
plt.ylabel('Rings')
plt.show()
```



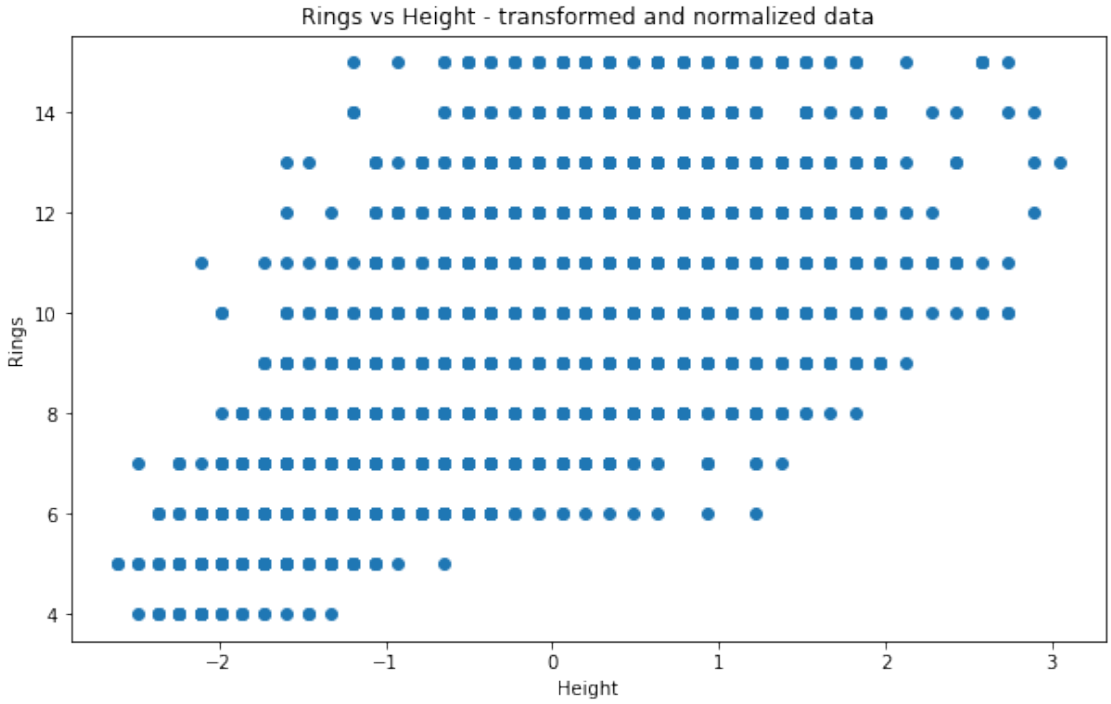
[650]:
```
#Scatter plot Rings vs Length

fig = plt.figure(figsize = (10, 6))
plt.scatter(data7N[:,1], data7N[:,8])
plt.title('Rings vs Length - transformed and normalized data')
plt.xlabel('Length')
plt.ylabel('Rings')
plt.show()
```



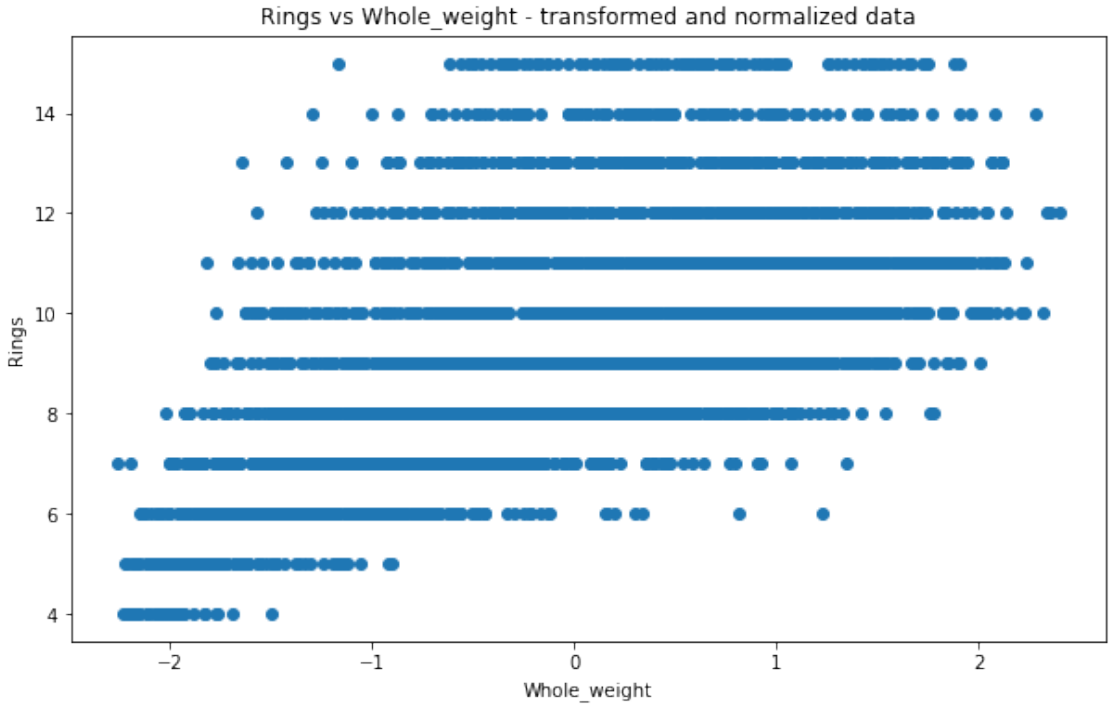
[651]:
```
#Scatter plot Rings vs Diameter

fig = plt.figure(figsize = (10, 6))
plt.scatter(data7N[:,2], data7N[:,8])
plt.title('Rings vs Diameter - transformed and normalized data')
plt.xlabel('Diameter')
plt.ylabel('Rings')
plt.show()
```



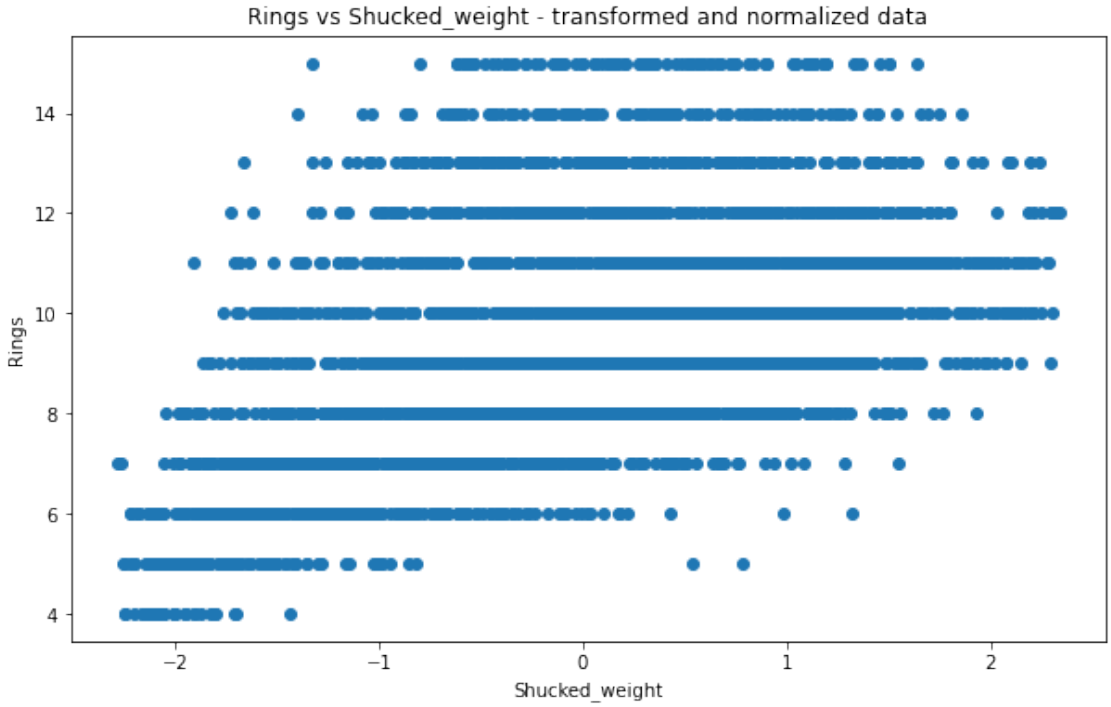
[652]:
```
#Scatter plot Rings vs Height

fig = plt.figure(figsize = (10, 6))
plt.scatter(data7N[:,3], data7N[:,8])
plt.title('Rings vs Height - transformed and normalized data')
plt.xlabel('Height')
plt.ylabel('Rings')
plt.show()
```



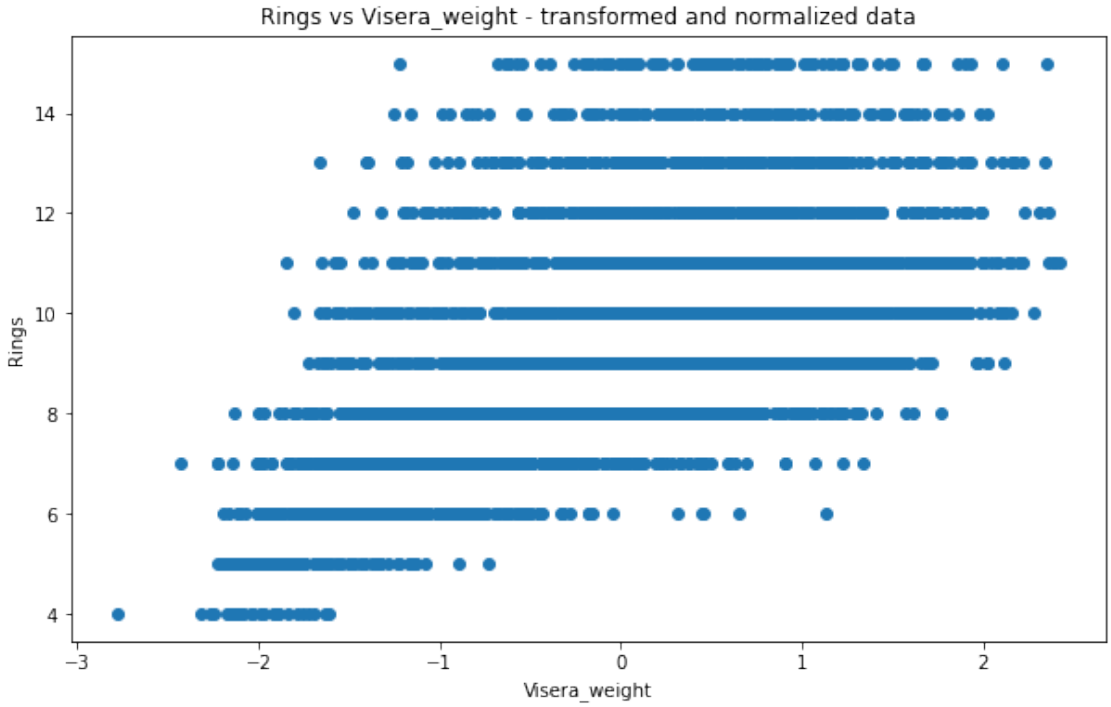
[653]:
```
#Scatter plot Rings vs Whole_weight

fig = plt.figure(figsize = (10, 6))
plt.scatter(data7N[:,4], data7N[:,8])
plt.title('Rings vs Whole_weight - transformed and normalized data')
plt.xlabel('Whole_weight')
plt.ylabel('Rings')
plt.show()
```



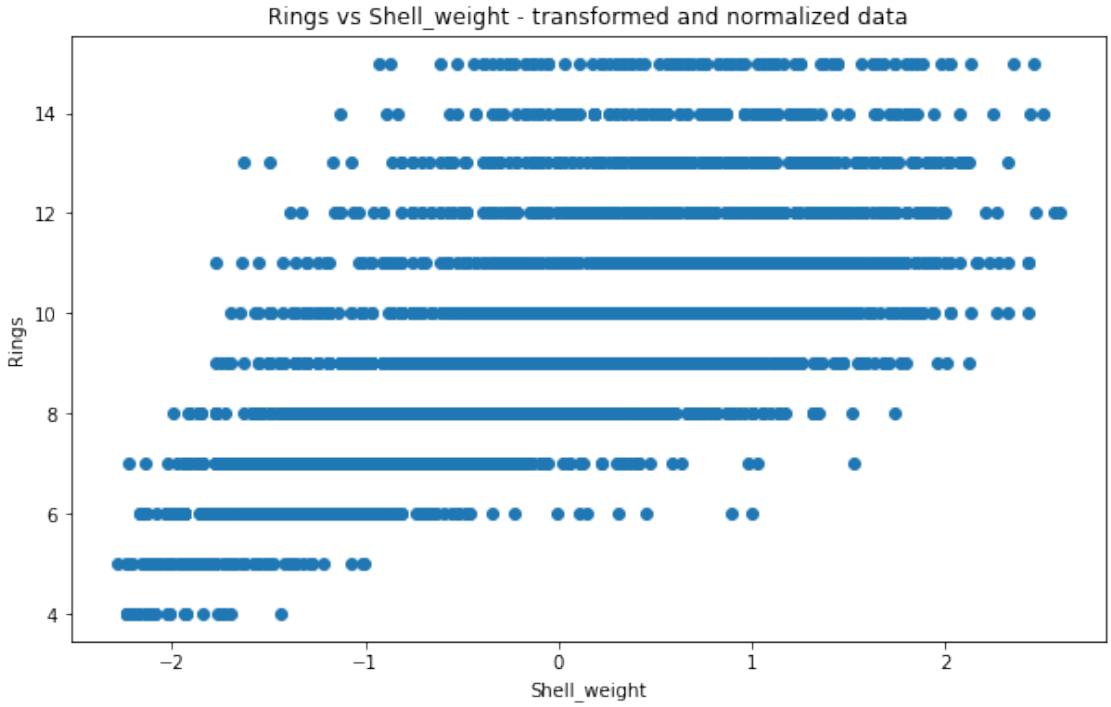
[654]:
```
#Scatter plot Rings vs Shucked_weight

fig = plt.figure(figsize = (10, 6))
plt.scatter(data7N[:,5], data7N[:,8])
plt.title('Rings vs Shucked_weight - transformed and normalized data')
plt.xlabel('Shucked_weight')
plt.ylabel('Rings')
plt.show()
```



[655]:
```
#Scatter plot Rings vs Visera_weight

fig = plt.figure(figsize = (10, 6))
plt.scatter(data7N[:,6], data7N[:,8])
plt.title('Rings vs Visera_weight - transformed and normalized data')
plt.xlabel('Visera_weight')
plt.ylabel('Rings')
plt.show()
```



[1463]:
```
#Scatter plot Rings vs Shell_weight

fig = plt.figure(figsize = (10, 6))
plt.scatter(data7N[:,7], data7N[:,8])
plt.title('Rings vs Shell_weight - transformed and normalized data')
plt.xlabel('Shell_weight')
plt.ylabel('Rings')
plt.show()
```



De los diagramas de dispersión presentados anteriormente, se puede observar que, para casi todos los valores de cada atributo normalizado, existe una correspondencia con varios valores de Rings cuando estos son mayores a 6, mientras que para valores de 4, solo unos pocos datos corresponden.

Esto afecta el desempeño del modelo, debido a que claramente no puede ser descrito como una recta y, adicionalmente, existen muchas combinaciones de diferentes valores de los atributos de entradas que corresponden a un mismo valor del atributo Rings, lo que sugiere un comportamiento no lineal.

Con base en lo anteriormente mencionado, resultará necesario el realizar combinaciones de inputs para mejorar el desempeño de la regresión lineal.

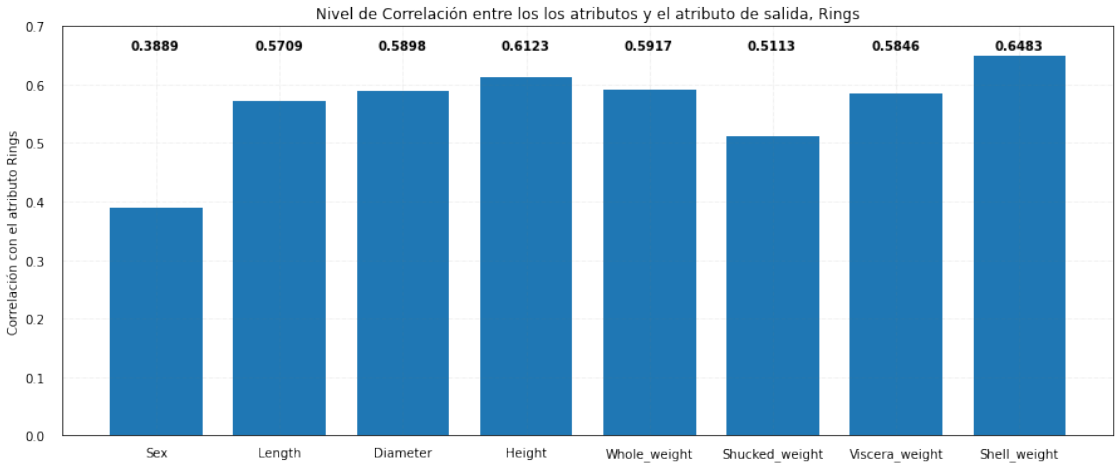
1.2.1 Correlación entre variables

Se debe evaluar, dentro del estudio estadístico de los atributos, la correlación que existe entre ellos.

Para medir la correlación entre los atributos de entrada y el atributo de salida, se procede a implementar la correlación de Pearson.

[821]:
```
#Correlaciones de Pearson de todos los atributos de entrada con el atributo de␣
↪salida.

#corrData = np.corrcoef([data7N[:,0],data7N[:,8]])

corrData = np.corrcoef(np.transpose(data7N))

#print(corrData)
```

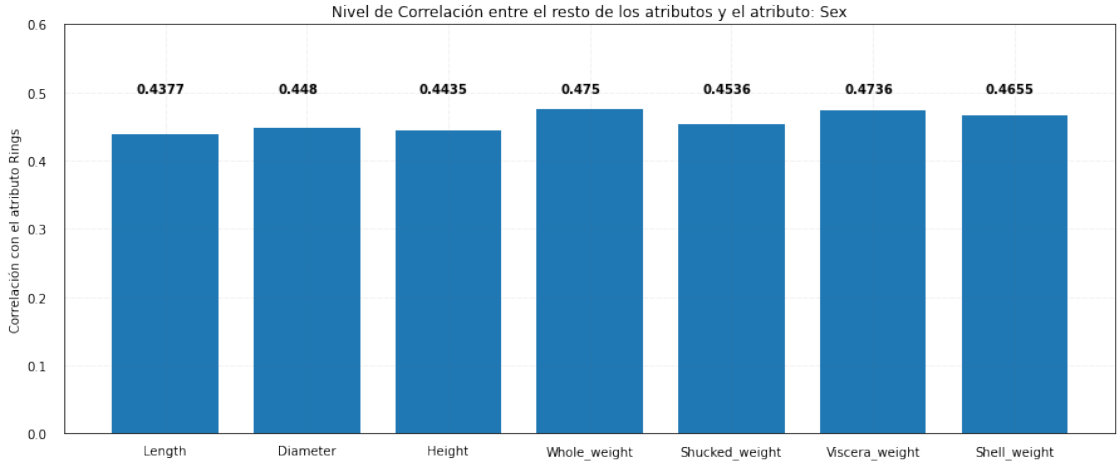
[1199]:
```
#print(list(varnames[0:8]))
#print(list(corrData[:,8]))

#Figure size
fig, ax = plt.subplots(figsize = (15, 6))

# Bar Plot
ax.bar(list(varnames[0:8]), list(corrData[0:8,8]), width = 0.75);

# Remove x, y Ticks
ax.xaxis.set_ticks_position('none')
ax.yaxis.set_ticks_position('none')

# Add padding between axes and labels
ax.xaxis.set_tick_params(pad = 5)
ax.yaxis.set_tick_params(pad = 10)

# Add x, y gridlines
ax.grid(b = True, color ='grey', linestyle ='-.', linewidth = 0.5, alpha = 0.2)

# Add annotation to bars
j = 0
for i in ax.patches:
    plt.text(i.get_x() + 0.17, i.get_y() + np.round( np.max( corrData[0:8,8] ),␣
 ↪1 ) + 0.06, str( np.round( corrData[j,8] , 4) ), fontsize = 10, fontweight␣
 ↪='bold', color ='black');
    j = j + 1

# Plot title
ax.set_title('Nivel de Correlación entre los los atributos y el atributo de␣
 ↪salida, Rings')

# y axis configuration
ax.set_ylim( [0, np.round( np.max( corrData[0:8,8] ) + 0.1, 1 )] )
ax.set_ylabel('Correlación con el atributo Rings')

#fig.text(0.9, 0.15, 'Jeeteshgavande30', fontsize = 12, color ='grey', ha␣
 ↪='right', va ='bottom', alpha = 0.7)

plt.show()
```



A continuación, se pretende calcular la correlación de los atributos considerados como variables de entrada entre ellos, de modo de detectar la influencia que arroja el uso de cada una al problema.

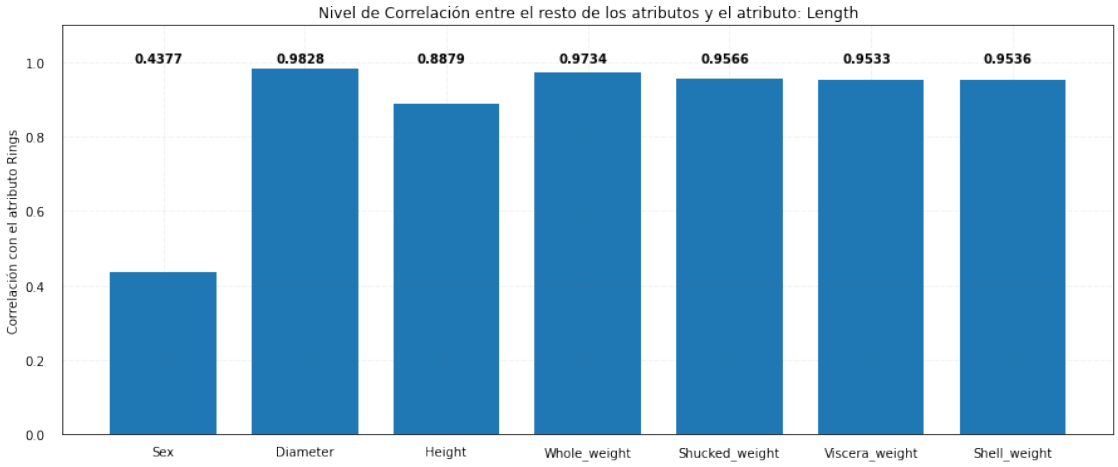
[830]:
```
#Se definen los vectores a graficar
aux = np.round(np.copy(corrData[0:8,0]), 15)
auxnames = np.copy(varnames[0:8])

auxAtr = auxnames[aux[:] == 1.];
auxnames = np.delete(auxnames, aux[:] == 1.);
aux = np.delete(aux, aux[:] == 1.);


# Figure size
fig, ax = plt.subplots(figsize = (15, 6))

# Bar Plot
ax.bar(list(auxnames), list(aux), width = 0.75);

# Remove x, y Ticks
ax.xaxis.set_ticks_position('none')
ax.yaxis.set_ticks_position('none')

# Add padding between axes and labels
ax.xaxis.set_tick_params(pad = 5)
ax.yaxis.set_tick_params(pad = 10)

# Add x, y gridlines
ax.grid(b = True, color ='grey', linestyle ='-.', linewidth = 0.5, alpha = 0.2)

# Add annotation to bars
j = 0
```

```
for i in ax.patches:
    plt.text(i.get_x()+0.17, i.get_y() + np.round( np.max( aux ), 1 ), str( np.
 ↪round( aux[j] , 4) ), fontsize = 10, fontweight ='bold', color ='black');
    j = j + 1

# Plot title
ax.set_title('Nivel de Correlación entre el resto de los atributos y el␣
 ↪atributo: ' + auxAtr[0])

# y axis configuration
ax.set_ylim( [0, np.round( np.max( aux ) + 0.1, 1 )] )
ax.set_ylabel('Correlación con el atributo Rings')

#fig.text(0.9, 0.15, 'Jeeteshgavande30', fontsize = 12, color ='grey', ha␣
 ↪='right', va ='bottom', alpha = 0.7)

plt.show()
```

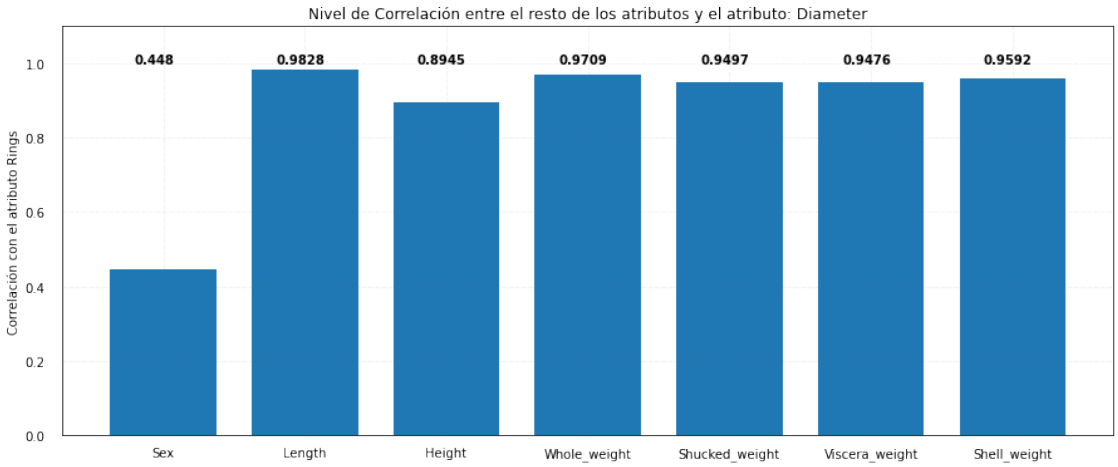


[827]:
```
#Se definen los vectores a graficar
aux = np.copy(corrData[0:8,1])
auxnames = np.copy(varnames[0:8])

auxAtr = auxnames[aux[:] == 1.0]
auxnames = np.delete(auxnames, aux[:] == 1.0);
aux = np.delete(aux, aux[:] == 1.0);


# Figure size
fig, ax = plt.subplots(figsize = (15, 6))

# Bar Plot
```

```
ax.bar(list(auxnames), list(aux), width = 0.75);

# Remove x, y Ticks
ax.xaxis.set_ticks_position('none')
ax.yaxis.set_ticks_position('none')

# Add padding between axes and labels
ax.xaxis.set_tick_params(pad = 5)
ax.yaxis.set_tick_params(pad = 10)

# Add x, y gridlines
ax.grid(b = True, color ='grey', linestyle ='-.', linewidth = 0.5, alpha = 0.2)

# Add annotation to bars
j = 0
for i in ax.patches:
    plt.text(i.get_x()+0.17, i.get_y() + np.round( np.max( aux ), 1 ), str( np.
 ↪round( aux[j] , 4) ), fontsize = 10, fontweight ='bold', color ='black');
    j = j + 1

# Plot title
ax.set_title('Nivel de Correlación entre el resto de los atributos y el␣
 ↪atributo: ' + auxAtr[0])

# y axis configuration
ax.set_ylim( [0, np.round( np.max( aux ) + 0.1, 1 )] )
ax.set_ylabel('Correlación con el atributo Rings')

#fig.text(0.9, 0.15, 'Jeeteshgavande30', fontsize = 12, color ='grey', ha␣
 ↪='right', va ='bottom', alpha = 0.7)

plt.show()
```

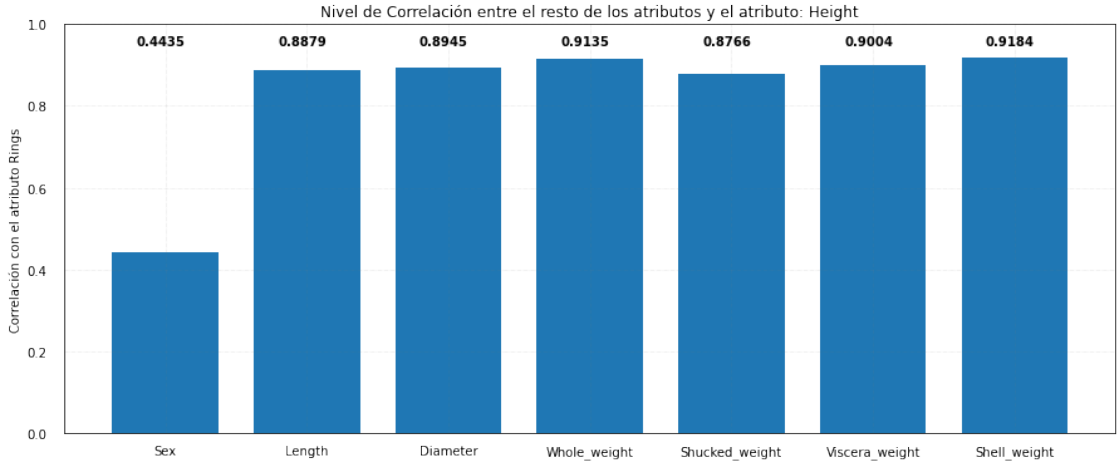


[819]:
```
#Se definen los vectores a graficar
aux = np.copy(corrData[0:8,2])
auxnames = np.copy(varnames[0:8])

auxAtr = auxnames[aux[:] == 1.0]
auxnames = np.delete(auxnames, aux[:] == 1.0);
aux = np.delete(aux, aux[:] == 1.0);


# Figure size
fig, ax = plt.subplots(figsize = (15, 6))

# Bar Plot
ax.bar(list(auxnames), list(aux), width = 0.75);

# Remove x, y Ticks
ax.xaxis.set_ticks_position('none')
ax.yaxis.set_ticks_position('none')

# Add padding between axes and labels
ax.xaxis.set_tick_params(pad = 5)
ax.yaxis.set_tick_params(pad = 10)

# Add x, y gridlines
ax.grid(b = True, color ='grey', linestyle ='-.', linewidth = 0.5, alpha = 0.2)

# Add annotation to bars
j = 0
for i in ax.patches:
    plt.text(i.get_x()+0.17, i.get_y() + np.round( np.max( aux ), 1 ), str( np.
 ↪round( aux[j] , 4) ), fontsize = 10, fontweight ='bold', color ='black');
    j = j + 1

# Plot title
ax.set_title('Nivel de Correlación entre el resto de los atributos y el␣
 ↪atributo: ' + auxAtr[0])

# y axis configuration
ax.set_ylim( [0, np.round( np.max( aux ) + 0.1, 1 )] )
ax.set_ylabel('Correlación con el atributo Rings')

#fig.text(0.9, 0.15, 'Jeeteshgavande30', fontsize = 12, color ='grey', ha␣
 ↪='right', va ='bottom', alpha = 0.7)

plt.show()
```



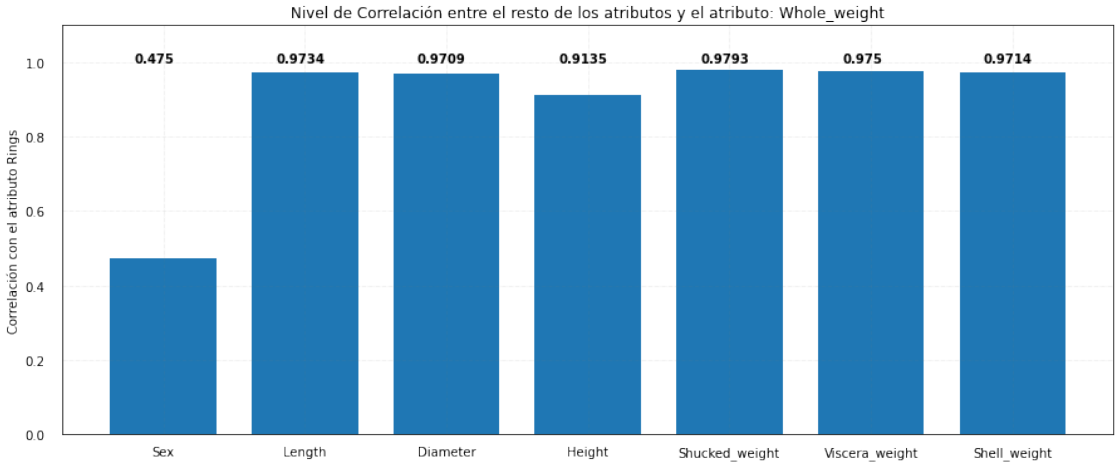
[820]:
```
#Se definen los vectores a graficar
aux = np.copy(corrData[0:8,3])
auxnames = np.copy(varnames[0:8])

auxAtr = auxnames[aux[:] == 1.0]
auxnames = np.delete(auxnames, aux[:] == 1.0);
aux = np.delete(aux, aux[:] == 1.0);


# Figure size
fig, ax = plt.subplots(figsize = (15, 6))

# Bar Plot
ax.bar(list(auxnames), list(aux), width = 0.75);

# Remove x, y Ticks
ax.xaxis.set_ticks_position('none')
ax.yaxis.set_ticks_position('none')

# Add padding between axes and labels
ax.xaxis.set_tick_params(pad = 5)
ax.yaxis.set_tick_params(pad = 10)

# Add x, y gridlines
ax.grid(b = True, color ='grey', linestyle ='-.', linewidth = 0.5, alpha = 0.2)

# Add annotation to bars
j = 0
for i in ax.patches:
```

```
    plt.text(i.get_x()+0.17, i.get_y() + np.round( np.max( aux ), 1 ) + 0.05,␣
 ↪str( np.round( aux[j] , 4) ), fontsize = 10, fontweight ='bold', color␣
 ↪='black');
    j = j + 1

# Plot title
ax.set_title('Nivel de Correlación entre el resto de los atributos y el␣
 ↪atributo: ' + auxAtr[0])

# y axis configuration
ax.set_ylim( [0, np.round( np.max( aux ) + 0.1, 1 )] )
ax.set_ylabel('Correlación con el atributo Rings')

#fig.text(0.9, 0.15, 'Jeeteshgavande30', fontsize = 12, color ='grey', ha␣
 ↪='right', va ='bottom', alpha = 0.7)

plt.show()
```

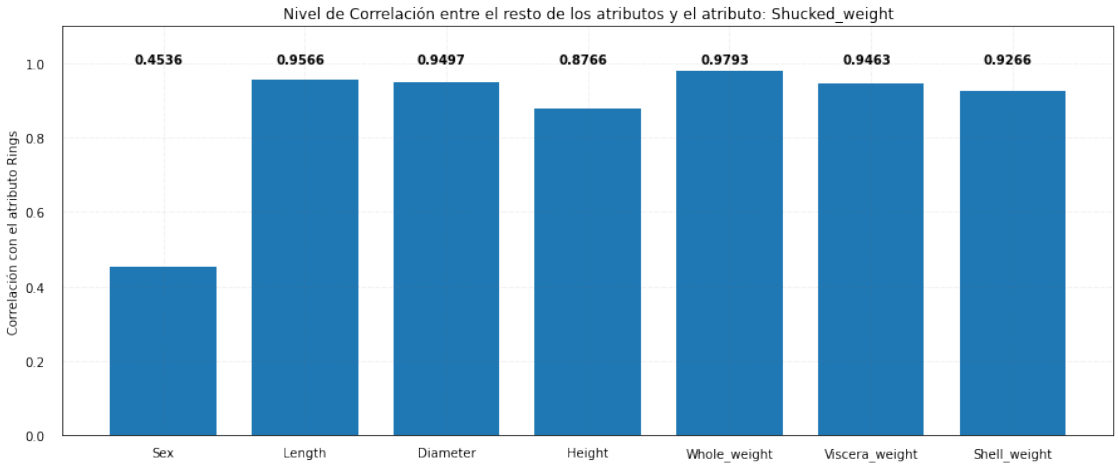


[663]:
```
#Se definen los vectores a graficar
aux = np.copy(corrData[0:8,4])
auxnames = np.copy(varnames[0:8])

auxAtr = auxnames[aux[:] == 1.0]
auxnames = np.delete(auxnames, aux[:] == 1.0);
aux = np.delete(aux, aux[:] == 1.0);


# Figure size
fig, ax = plt.subplots(figsize = (15, 6))

# Bar Plot
```

```
ax.bar(list(auxnames), list(aux), width = 0.75);

# Remove x, y Ticks
ax.xaxis.set_ticks_position('none')
ax.yaxis.set_ticks_position('none')

# Add padding between axes and labels
ax.xaxis.set_tick_params(pad = 5)
ax.yaxis.set_tick_params(pad = 10)

# Add x, y gridlines
ax.grid(b = True, color ='grey', linestyle ='-.', linewidth = 0.5, alpha = 0.2)

# Add annotation to bars
j = 0
for i in ax.patches:
    plt.text(i.get_x()+0.17, i.get_y() + np.round( np.max( aux ), 1 ), str( np.
 ↪round( aux[j] , 4) ), fontsize = 10, fontweight ='bold', color ='black');
    j = j + 1

# Plot title
ax.set_title('Nivel de Correlación entre el resto de los atributos y el␣
 ↪atributo: ' + auxAtr[0])

# y axis configuration
ax.set_ylim( [0, np.round( np.max( aux ) + 0.1, 1 )] )
ax.set_ylabel('Correlación con el atributo Rings')

#fig.text(0.9, 0.15, 'Jeeteshgavande30', fontsize = 12, color ='grey', ha␣
 ↪='right', va ='bottom', alpha = 0.7)

plt.show()
```

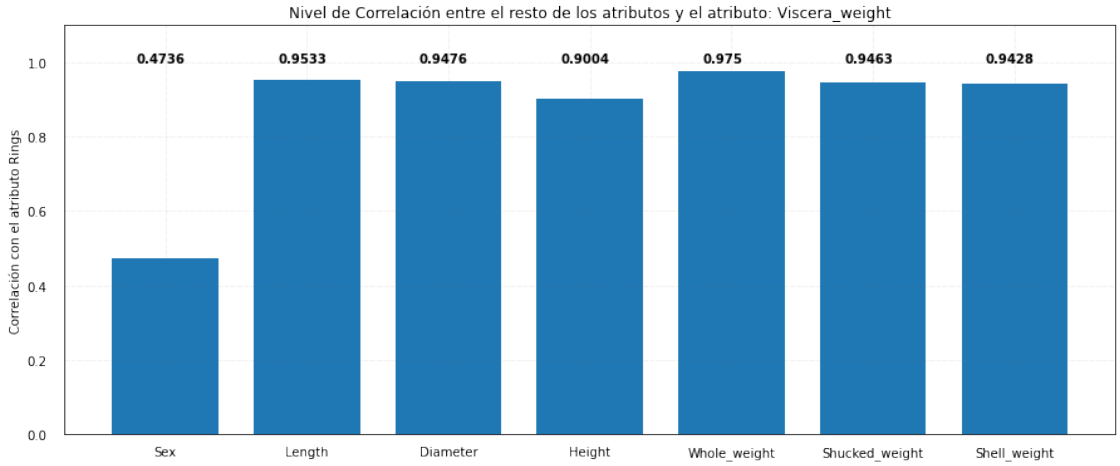


[664]:
```
#Se definen los vectores a graficar
aux = np.copy(corrData[0:8,5])
auxnames = np.copy(varnames[0:8])

auxAtr = auxnames[aux[:] == 1.0]
auxnames = np.delete(auxnames, aux[:] == 1.0);
aux = np.delete(aux, aux[:] == 1.0);


# Figure size
fig, ax = plt.subplots(figsize = (15, 6))

# Bar Plot
ax.bar(list(auxnames), list(aux), width = 0.75);

# Remove x, y Ticks
ax.xaxis.set_ticks_position('none')
ax.yaxis.set_ticks_position('none')

# Add padding between axes and labels
ax.xaxis.set_tick_params(pad = 5)
ax.yaxis.set_tick_params(pad = 10)

# Add x, y gridlines
ax.grid(b = True, color ='grey', linestyle ='-.', linewidth = 0.5, alpha = 0.2)

# Add annotation to bars
j = 0
for i in ax.patches:
    plt.text(i.get_x()+0.17, i.get_y() + np.round( np.max( aux ), 1 ), str( np.
 ↪round( aux[j] , 4) ), fontsize = 10, fontweight ='bold', color ='black');
    j = j + 1

# Plot title
ax.set_title('Nivel de Correlación entre el resto de los atributos y el␣
 ↪atributo: ' + auxAtr[0])

# y axis configuration
ax.set_ylim( [0, np.round( np.max( aux ) + 0.1, 1 )] )
ax.set_ylabel('Correlación con el atributo Rings')

#fig.text(0.9, 0.15, 'Jeeteshgavande30', fontsize = 12, color ='grey', ha␣
 ↪='right', va ='bottom', alpha = 0.7)

plt.show()
```



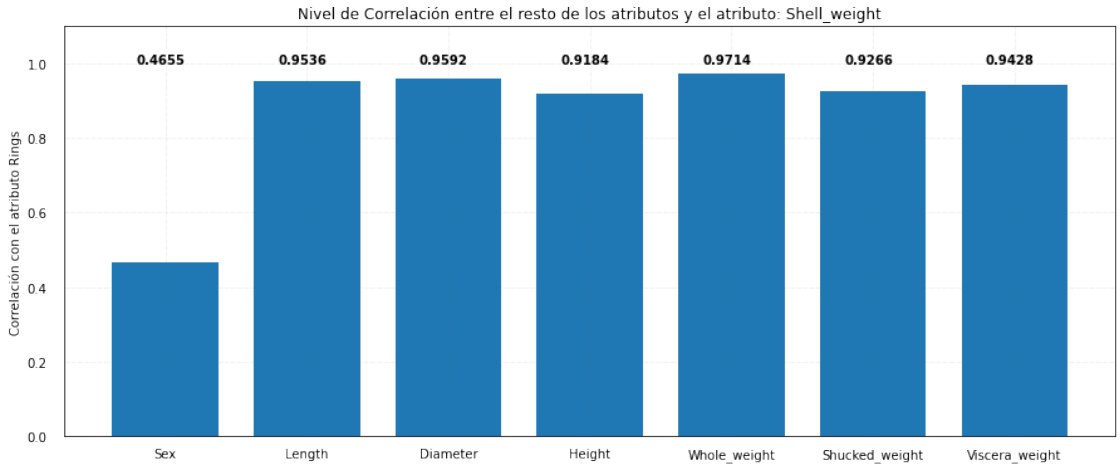
[665]:
```
#Se definen los vectores a graficar
aux = np.copy(corrData[0:8,6])
auxnames = np.copy(varnames[0:8])

auxAtr = auxnames[aux[:] == 1.0]
auxnames = np.delete(auxnames, aux[:] == 1.0);
aux = np.delete(aux, aux[:] == 1.0);


# Figure size
fig, ax = plt.subplots(figsize = (15, 6))

# Bar Plot
ax.bar(list(auxnames), list(aux), width = 0.75);

# Remove x, y Ticks
ax.xaxis.set_ticks_position('none')
ax.yaxis.set_ticks_position('none')

# Add padding between axes and labels
ax.xaxis.set_tick_params(pad = 5)
ax.yaxis.set_tick_params(pad = 10)

# Add x, y gridlines
ax.grid(b = True, color ='grey', linestyle ='-.', linewidth = 0.5, alpha = 0.2)

# Add annotation to bars
j = 0
for i in ax.patches:
    plt.text(i.get_x()+0.17, i.get_y() + np.round( np.max( aux ), 1 ), str( np.
 ↪round( aux[j] , 4) ), fontsize = 10, fontweight ='bold', color ='black');
```

```
    j = j + 1

# Plot title
ax.set_title('Nivel de Correlación entre el resto de los atributos y el␣
 ↪atributo: ' + auxAtr[0])

# y axis configuration
ax.set_ylim( [0, np.round( np.max( aux ) + 0.1, 1 )] )
ax.set_ylabel('Correlación con el atributo Rings')

#fig.text(0.9, 0.15, 'Jeeteshgavande30', fontsize = 12, color ='grey', ha␣
 ↪='right', va ='bottom', alpha = 0.7)

plt.show()
```



[708]:
```
#Se definen los vectores a graficar
aux = np.copy(corrData[0:8,7])
auxnames = np.copy(varnames[0:8])

auxAtr = auxnames[aux[:] == 1.0]
auxnames = np.delete(auxnames, aux[:] == 1.0);
aux = np.delete(aux, aux[:] == 1.0);


# Figure size
fig, ax = plt.subplots(figsize = (15, 6))

# Bar Plot
ax.bar(list(auxnames), list(aux), width = 0.75);

# Remove x, y Ticks
```

```
ax.xaxis.set_ticks_position('none')
ax.yaxis.set_ticks_position('none')

# Add padding between axes and labels
ax.xaxis.set_tick_params(pad = 5)
ax.yaxis.set_tick_params(pad = 10)

# Add x, y gridlines
ax.grid(b = True, color ='grey', linestyle ='-.', linewidth = 0.5, alpha = 0.2)

# Add annotation to bars
j = 0
for i in ax.patches:
    plt.text(i.get_x()+0.17, i.get_y() + np.round( np.max( aux ), 1 ), str( np.
 ↪round( aux[j] , 4) ), fontsize = 10, fontweight ='bold', color ='black');
    j = j + 1

# Plot title
ax.set_title('Nivel de Correlación entre el resto de los atributos y el␣
 ↪atributo: ' + auxAtr[0])

# y axis configuration
ax.set_ylim( [0, np.round( np.max( aux ) + 0.1, 1 )] )
ax.set_ylabel('Correlación con el atributo Rings')

#fig.text(0.9, 0.15, 'Jeeteshgavande30', fontsize = 12, color ='grey', ha␣
 ↪='right', va ='bottom', alpha = 0.7)

plt.show()
```



De estos resultados se puede apreciar que, salvo por el atributo Sex, el resto de atributos a imple-

mentar como variables de entrada en la regresión lineal presentan una alta correlación (superior a 0.85), lo que indica, en primera instancia, que el uso de muchas de estas variables no debería generar un gran aporte en cuanto a la precisión de la regresión.

No obstante, si se compara la correlación de cada uno de los atributos de entrada con la salida, se obtiene una correlación considerablemente más baja (menor a 0.65) para todas. Esto indica que el conjunto de atributos de entrada no necesariamente tiene una relación de asociación (parentezco o tendencia) definida con el atributo de salida.

Lo anteriormente descrito resulta de interés para el criterio de diseño de las variables de entrada de los diferentes modelos de regresión a implementar.

1.3 Regresión Lineal

Teniendo el conjunto de datos previamente preprocesado, se procede a implementar una validación cruzada de 5 iteraciones (5-fold cross-validation).

1.3.1 Validación cruzada de 5 iteraciones (5-fold cross-validation)

Se procede a definir los 5 grupos de sets que se implementarán para la regresión lineal. Cada set contará con un grupo de datos de entrenamiento y otro grupo de datos de validación.

Las pruebas se realizarán con los modelos de k-fold cross validation y linear regretion provistos por la biblioteca SKLearn.

1.3.2 → modelo con todos los parámetros

Inicialmente se considerarán todos los atributos para la entrada del sistema. Este sería la primera iteración para la regresión y su fin es servir de referencia para comparar los resultados de las demás pruebas.

Se declaran las funciones que se implementarán a lo largo de toda la regresión lineal:

[1119]:
```
# función para calcular el Residual Sum of Squares (RSS)

def RSS(y_real, y_calc):
    return np.sum ( ( y_real - y_calc ) * ( y_real - y_calc ) )
```

[1128]:
```
# Función para efectuar la regresión lineal con validación cruzada

def KFoldsCVLinearRegresion(data, Nsplits, varShuffle = False, randomState =␣
↪None):

    # Se inicializa el k-folds cross validation
    kfcv = sklms.KFold(n_splits = Nsplits, shuffle = varShuffle, random_state =␣
↪randomState)

    # Se obtienen los índices de los splits de data.
    #kfcv.get_n_splits(data7N)
```

```
    # Se inicializa el modelo a entrenar
    model = skllm.LinearRegression()


    # Variables de interés a almacenar
    betas = np.zeros((np.size(data,1) -1, Nsplits))
    RSSent = np.zeros((5))
    RSSeva = np.zeros((5))
    R2ent = np.zeros((5))
    R2eva = np.zeros((5))
    R2CValScore = np.zeros((5,5))


    # Se generan los índices para separar los datos y se procede a predecir el␣
↪modelo para cada set de datos
    #kfcv.split(data7N)


    i=0;   #contador

    for train_index, test_index in kfcv.split(data7N):
        #print("TRAIN:", train_index, "TEST:", test_index)

        # Inputs de entrenamiento y prueba
        X_train, X_test = data[train_index, 0:-1], data[test_index, 0:-1]

        # Outputs de entrenamiento y prueba
        y_train, y_test = data[train_index, -1], data[test_index, -1]

        # Se entrena el modelo
        model.fit(X_train, y_train)

        # Se almacenan los coeficientes de la iteración
        betas[:,i] = model.coef_

        # Se prueba el modelo (datos de entrenamiento)
        modelOutTrain = model.predict(X_train)

        # Evaluación de la predicción (datos de entrenamiento)
        RSSent[i] = RSS( y_train , modelOutTrain )
        R2ent[i] = model.score(X_train, y_train)

        # Se prueba el modelo (datos de prueba)
        modelOutEval = model.predict(X_test)

        # Evaluación de la predicción (datos de prueba)
```

```
        RSSeva[i] = RSS( y_test , modelOutEval )
        R2eva[i] = model.score( X_test, y_test )

        R2CValScore[i,:] = sklms.cross_val_score(model, X_test, y_test, cv =␣
↪kfcv);

        # Se actualiza el contador
        i = i + 1;


    #for_end

    return betas , RSSent , RSSeva , R2ent , R2eva , R2CValScore;

#function_end
```

Se procede a implementar las funciones para estimar la regresión lineal.

[1129]:
```
# Se ejecuta la regresión lineal con cross validation
[ betas , RSSent , RSSeva , R2ent , R2eva , R2CValScore ] =␣
↪KFoldsCVLinearRegresion(data7N, 5);
```

Se presentan los resultados:

[1130]:
```
print('Coeficientes de cada iteración:')
print('la primera fila indica el nombre de la iteración durante el k-fold cross␣
↪validation.\nEl resto de valores de cada columna corresponden a los␣
↪coeficientes de dicha iteración. \n')
print( np.vstack((np.array([1, 2, 3, 4, 5]), betas)) )
```

```
Coeficientes de cada iteración:
la primera fila indica el nombre de la iteración durante el k-fold cross
validation.
El resto de valores de cada columna corresponden a los coeficientes de dicha
iteración.

[[1.0000 2.0000 3.0000 4.0000 5.0000]
 [0.2462 0.3241 0.3331 0.3273 0.3200]
 [-0.3719 -0.4928 -0.4686 -0.4084 -0.6199]
 [0.4509 0.4912 0.4129 0.3222 0.5823]
 [0.3306 0.4470 0.3802 0.3896 0.3829]
 [1.7286 2.0237 2.1085 2.2521 1.9849]
 [-2.1963 -2.7934 -2.5522 -2.8351 -2.6762]
 [-0.1052 0.0178 -0.1786 -0.0077 -0.0298]
 [1.4537 1.5930 1.5701 1.5633 1.6703]]
```

[1131]:
```
print('R^2 de cada iteración con los datos de entrenamiento:')
print(R2ent)
```

```
R^2 de cada iteración con los datos de entrenamiento:
[0.5113 0.5033 0.4982 0.5153 0.5053]
```

[1132]:
```
print('RSS de cada iteración con los datos de entrenamiento:')
print(RSSent)
```

```
RSS de cada iteración con los datos de entrenamiento:
[7025.4572 8701.7694 8077.5147 8101.3694 8480.4237]
```

[1133]:
```
print('R^2 de cada iteración con los datos de prueba:')
print(R2eva)
```

```
R^2 de cada iteración con los datos de prueba:
[0.4110 0.3493 0.5195 0.4596 0.4924]
```

[1134]:
```
print('RSS de cada iteración con los datos de prueba:')
print(RSSeva)
```

```
RSS de cada iteración con los datos de prueba:
[3326.7075 1520.8482 2084.4373 2060.4979 1695.2362]
```

[1135]:
```
print('error del modelo usando el comando cross_val_score:')
print(R2CValScore)
```

```
error del modelo usando el comando cross_val_score:
[[0.4935 0.5959 0.4683 0.2085 0.3467]
 [0.6253 0.5541 0.3397 0.7497 0.3118]
 [0.2146 0.4189 0.5694 0.3641 0.4803]
 [0.2186 0.5723 0.3850 0.4024 -0.4342]
 [0.5329 0.4750 0.5131 0.3616 0.5498]]
```

Como se puede apreciar, este modelo no presenta tan buena precisión, pues el promedio del RSS se encuentra alrededor de 0.5 para todos los casos

1.3.3 → Primer modelo con algunos parámetros modificados

Analizando la correlación entre los diferentes atributos de entrada respecto al atributo Rings, se puede apreciar que aquellos quienes presentan una menor correlación son los atributos Sex, Length y Shucked_weight. Esto indica que son los parámetros que menos información aportan para predecir el atributo Rings.

Adicionalmente, estos 3 atributos poseen la menor correlación con el atributo Height, el cual posee la mejor correlación respecto al atributo Rings.

Con base en lo anteriormente mencionado, se procede a crear una variable adicional a partir de los atributos de entrada para conseguir una mejor precisión del modelo.

Modificaciones a las variables de entrada → Se considera descartar la variable Sex dentro de la predicción debido a su baja correlación con la salida y su naturaleza (discreta, enfocada en clasificar y no cuantificar), lo que la hace muy difícil de manipular si se desea combinar con cualquiera de los otros atributos, que son de naturaleza continua.

→ Se relacionan los atributos Length y Shucked_weight con el atributo Height siguiendo la expresión:

$$atributoNuevo = \frac{height^2}{(length)(Shucked\_weight)}$$

Y eliminar estos 3 atributos del dataset.

[1137]:
```
# Se crea el nuevo conjunto de datos
data8 = np.copy(data7N)

data8 = np.insert(data8, 7, data8[:,3] * data8[:,3] / (data8[:,1] * data8[:
 ↪,5]), axis = 1)
data8 = np.delete(data8, 5,  axis = 1)
data8 = np.delete(data8, 3,  axis = 1)
data8 = np.delete(data8, 1,  axis = 1)
data8 = np.delete(data8, 0,  axis = 1)


# Se ejecuta la regresión lineal con cross validation
[ betas2 ,  RSSent2 , RSSeva2 , R2ent2 , R2eva2 , R2val2 ] =␣
 ↪KFoldsCVLinearRegresion(data8, 5);
```

Se presentan los resultados:

[1138]:
```
print('Coeficientes de cada iteración:')
print('la primera fila indica el nombre de la iteración durante el k-fold cross␣
 ↪validation.\nEl resto de valores de cada columna corresponden a los␣
 ↪coeficientes de dicha iteración. \n')
print( np.vstack((np.array([1, 2, 3, 4, 5]), betas2)) )
```

```
Coeficientes de cada iteración:
la primera fila indica el nombre de la iteración durante el k-fold cross
validation.
El resto de valores de cada columna corresponden a los coeficientes de dicha
iteración.

[[1.0000 2.0000 3.0000 4.0000 5.0000]
 [-0.0586 -0.2761 -0.3291 -0.4033 -0.2584]
 [-1.5708 -2.2612 -1.8461 -2.1805 -2.2356]
 [0.4676 0.6852 0.5599 0.7784 0.6733]
 [0.0004 0.0000 0.0000 -0.0000 0.0000]
 [2.6249 3.3321 3.0953 3.2946 3.3152]]
```

[1139]:
```
print('R^2 de cada iteración con los datos de entrenamiento:')
print(R2ent2)
```

```
R^2 de cada iteración con los datos de entrenamiento:
[0.4712 0.4462 0.4433 0.4587 0.4482]
```

[1140]:
```
print('RSS de cada iteración con los datos de entrenamiento:')
print(RSSent2)
```

```
RSS de cada iteración con los datos de entrenamiento:
[7602.4716 9702.9864 8961.4578 9048.7413 9459.0635]
```

[1141]:
```
print('R^2 de cada iteración con los datos de prueba:')
print(R2eva2)
```

```
R^2 de cada iteración con los datos de prueba:
[0.2499 0.2714 0.4619 0.4114 0.4445]
```

[1142]:
```
print('RSS de cada iteración con los datos de prueba:')
print(RSSeva2)
```

```
RSS de cada iteración con los datos de prueba:
[4236.9792 1702.8460 2334.0258 2244.4010 1855.1792]
```

[1143]:
```
print('error del modelo usando el comando cross_val_score:')
print(R2val2)
```

```
error del modelo usando el comando cross_val_score:
[[0.4302 0.5864 0.4924 0.1509 0.2503]
 [0.6292 0.5714 0.3281 0.7546 0.3059]
 [-0.0274 0.4673 0.5501 0.2904 0.3696]
 [0.1740 0.5546 0.4253 0.3567 -0.5547]
 [0.5157 0.5781 0.4867 0.3024 0.3914]]
```

Como se puede apreciar, en esta iteración se obtienen resultados peores al caso de considerar todos los atributos sin modificar.

El desempeño de este conjunto de datos de entradas no solo desmejoró el rendimiento de la regresión lineal, sino que también permitió observar efecto interesante: la variable Shell_weight, no modificada, tiene un aporte mucho menor (al menos 4 órdenes de magnitud menor) al resto de los atributos.

Este hecho se tomará encuentra en la siguiente iteración.

Se procede a plantear una nueva prueba para intentar mejorar el desempeño de la regresión.

1.3.4 → Segundo modelo con algunos parámetros modificados

Con el fin de mejorar aún más el modelo, se procede a probar modificar los atributos anteriormente modificados, junto con el atributo Shell_weight, y usar las nuevas variables como input de la regresión.

El sentido de dicha decisión se basa en mejorar la correlación de los atributos que tienen la correlación más baja con el atributo Rings.

Es importante destacar que los coeficientes del atributo Shell_weight es al menos 4 órdenes de magnitud menor que los coeficientes del resto de atributos, lo que destaca la poca influencia de esta variable al presentar variables resultantes de la combinación lineal de diversos atributos. Por ello, el mismo se modificará junto con los demás.

Con base en los resultados anteriores, se procede a realizar las siguientes modificaciones

Modificaciones a las variables de entrada → Se relacionan los atributos Length, Shucked_weight y Shell_weight con Height mediante la expresión:

$$atributoNuevo2 = \frac{Length + Height}{2}$$
$$atributoNuevo3 = \frac{Shucked\_weight + Height}{2}$$
$$atributoNuevo4 = \frac{Shell\_weight + Height}{2}$$

Y eliminar los atributos Length y Shucked_weight del dataset.

[1248]:
```
#Implementado el nuevo conjunto de datos
data9 = np.copy(data7N)

data9[:, 1] = (data9[:, 1] + data9[:,3]) *0.5;
data9[:, 5] = (data9[:, 5] + data9[:,3]) *0.5;
data9[:, 7] = (data9[:, 7] + data9[:,3]) *0.5;


# Se ejecuta la regresión lineal con cross validation
[ betas3 , RSSent3 , RSSeva3 , R2ent3 , R2eva3 , R2val3 ] =␣
↪KFoldsCVLinearRegresion(data9, 5);
```

[1249]:
```
print('Coeficientes de cada iteración:')
print('la primera fila indica el nombre de la iteración durante el k-fold cross␣
↪validation.\nEl resto de valores de cada columna corresponden a los␣
↪coeficientes de dicha iteración. \n')
print( np.vstack((np.array([1, 2, 3, 4, 5]), betas3)) )
```

```
Coeficientes de cada iteración:
la primera fila indica el nombre de la iteración durante el k-fold cross
validation.
El resto de valores de cada columna corresponden a los coeficientes de dicha
iteración.

[[1.0000 2.0000 3.0000 4.0000 5.0000]
 [0.2462 0.3241 0.3331 0.3273 0.3200]
```

```
 [-0.7439 -0.9856 -0.9371 -0.8169 -1.2399]
 [0.4509 0.4912 0.4129 0.3222 0.5823]
 [1.4451 2.1402 1.8310 2.0697 2.0087]
 [1.7286 2.0237 2.1085 2.2521 1.9849]
 [-4.3926 -5.5868 -5.1045 -5.6701 -5.3525]
 [-0.1052 0.0178 -0.1786 -0.0077 -0.0298]
 [2.9075 3.1860 3.1401 3.1267 3.3406]]
```

[1250]:
```
print('R^2 de cada iteración con los datos de entrenamiento:')
print(R2ent3)
```

```
R^2 de cada iteración con los datos de entrenamiento:
[0.5113 0.5033 0.4982 0.5153 0.5053]
```

[1251]:
```
print('residual sum of squares de cada iteración (RSS) con los datos de␣
 ↪entrenamiento:')
print(RSSent3)
```

```
residual sum of squares de cada iteración (RSS) con los datos de entrenamiento:
[7025.4572 8701.7694 8077.5147 8101.3694 8480.4237]
```

[1252]:
```
print('R^2 de cada iteración con los datos de prueba:')
print(R2eva3)
```

```
R^2 de cada iteración con los datos de prueba:
[0.4110 0.3493 0.5195 0.4596 0.4924]
```

[1253]:
```
print('residual sum of squares de cada iteración (RSS) con los datos de prueba:
 ↪')
print(RSSeva3)
```

```
residual sum of squares de cada iteración (RSS) con los datos de prueba:
[3326.7075 1520.8482 2084.4373 2060.4979 1695.2362]
```

[1299]:
```
print('error del modelo usando el comando cross_val_score:')
print(R2val3)
```

```
error del modelo usando el comando cross_val_score:
[[0.4935 0.5959 0.4683 0.2085 0.3467]
 [0.6253 0.5541 0.3397 0.7497 0.3118]
 [0.2146 0.4189 0.5694 0.3641 0.4803]
 [0.2186 0.5723 0.3850 0.4024 -0.4342]
 [0.5329 0.4750 0.5131 0.3616 0.5498]]
```

Como se puede apreciar, los resultados obtenidos en esta iteración son mejores a los arrojados en la segunda prueba, pero ligeramente por debajo de los arrojados en la primera, y la influencia de los diversos atributos y combinación de atributos se resulta ser similar (los coeficientes, β 's, tienen órdenes de magnitud similares y cercanos) sobre el modelo.

En vista de haber tenido una gran mejora al combinar los atributos con menor correlación respecto al atributo Rings con los atriutos con mayor correlación respecto a la misma, se procede a plantear otro escenario con datos resultantes de la combinación lineal de atributos para estimar el atributo Rings mediante la regresión lineal.

1.3.5 → Primer Modelo con todos los parámetros modificados

Si se observan los datos obtenidos en las correlaciones de cada atributo respecto al atributo Rings, los atributos con major correlación son Height y Shell_weight.

Si se pudiera crear variables con una correlación mayor, podría facilitar la estimación del atributo Rings mediante una regresión lineal.

En busca de tener una mejora sustancial en el modelo que permita garantizar una regresión lineal, y siguiendo lo planteado anteriormente, se procede a implementar las siguientes modificaciones sobre los datos.

Modificaciones a las variables de entrada → Se relacionan todos los demás atributos, exceptuando el atributo Sex, con el atributo Height y el atributo Shell_weight siguiendo la expresión:

$$atributoNuevo(1,\ 2,\ 4,\ 5\,y\,6) = \frac{atributoOriginal + Height + Shell\_weight}{3}$$

Y eliminar estos atributos del dataset.

→ Se relacionan los atributos Height y Shell_weights de forma que se creen dos variables independientes linealmente a las demás, siguiendo la expresión:

$$atributoNuevo3 = \sqrt[4]{(Height^2)(Shell\_weight^2)}$$
$$atributoNuevo7 = \frac{(Height)(Shell\_weight)}{Height + Shell\_weight}$$

Y eliminar los atributos Height y Shell_weight del dataset.

[1328]:
```
#Implementado el nuevo conjunto de datos
data10 = np.copy(data7N)

aux3 = data7N[:,3] + data7N[:,7];
data10[:,1:-1] = (data10[:,1:-1] + aux3[:,None])/3
data10[:,3] = np.sqrt( np.sqrt(data7N[:,3] * data7N[:,3] * data7N[:,7] *␣
 ↪data7N[:,7]) )
data10[:,7] = data7N[:,3] * data7N[:,7] / ( data7N[:,3] + data7N[:,7] )


# Se ejecuta la regresión lineal con cross validation
[ betas4 , RSSent4 , RSSeva4 , R2ent4 , R2eva4 , R2val4 ] =␣
 ↪KFoldsCVLinearRegresion(data10, 5);
```

[1329]:
```
print('Coeficientes de cada iteración:')
print('la primera fila indica el nombre de la iteración durante el k-fold cross␣
 ↪validation.\nEl resto de valores de cada columna corresponden a los␣
 ↪coeficientes de dicha iteración. \n')
print( np.vstack((np.array([1, 2, 3, 4, 5]), betas4)) )
```

```
Coeficientes de cada iteración:
la primera fila indica el nombre de la iteración durante el k-fold cross
validation.
El resto de valores de cada columna corresponden a los coeficientes de dicha
iteración.

[[1.0000 2.0000 3.0000 4.0000 5.0000]
 [0.1911 0.2557 0.2804 0.2606 0.2588]
 [-1.3345 -1.6550 -1.6884 -1.5193 -2.1410]
 [1.8791 2.1548 1.8984 1.6594 2.4700]
 [-0.5970 -0.8030 -0.7227 -0.7702 -0.7262]
 [11.1292 13.2321 13.1396 14.1641 13.3434]
 [-9.1474 -11.4362 -10.5974 -11.7337 -11.1519]
 [-1.2665 -1.0826 -1.5551 -1.3730 -1.3043]
 [0.0140 0.0007 -0.0019 0.0005 0.0030]]
```

[1330]:
```
print('R^2 de cada iteración con los datos de entrenamiento:')
print(R2ent4)
```

```
R^2 de cada iteración con los datos de entrenamiento:
[0.5239 0.5265 0.5160 0.5377 0.5215]
```

[1331]:
```
print('residual sum of squares de cada iteración (RSS) con los datos de␣
 ↪entrenamiento:')
print(RSSent4)
```

```
residual sum of squares de cada iteración (RSS) con los datos de entrenamiento:
[6843.6329 8295.7554 7790.9547 7727.4744 8203.3747]
```

[1332]:
```
print('R^2 de cada iteración con los datos de prueba:')
print(R2eva4)
```

```
R^2 de cada iteración con los datos de prueba:
[0.4430 0.3280 0.5391 0.4590 0.5226]
```

[1333]:
```
print('residual sum of squares de cada iteración (RSS) con los datos de prueba:
 ↪')
print(RSSeva4)
```

```
residual sum of squares de cada iteración (RSS) con los datos de prueba:
[3146.3873 1570.7162 1999.2948 2062.7681 1594.2314]
```

[1334]:
```
print('error del modelo usando el comando cross_val_score:')
print(R2val4)
```

```
error del modelo usando el comando cross_val_score:
[[0.5168 0.6091 0.5050 0.2956 0.4631]
 [0.6186 0.5902 0.3664 0.7364 0.3074]
 [0.3538 0.3443 0.5502 0.3954 0.4969]
 [0.0293 0.6104 0.4236 0.4834 -0.4166]
 [0.5460 0.4962 0.4907 0.4115 0.5674]]
```

Como se puede observar, los resultados anteriores son ligeramente mejores a los de la primera iteración, por lo que ya se ha encontrado una mejor solución al problema.

Sin embargo, la mejora es lo suficientemente baja como para no ser relevante.

En este sentido, se probará otra combinación que mejore considerablemente el resultado de la regresión lineal.

1.3.6 → Segundo Modelo con todos los parámetros modificados

Después de múltiples pruebas, se observó que los promedios con raíz o promedio de productos no aportan tanto al modelo.

La modificación que más aporta resulta ser la suma ponderada con igualdad de contribuciones.

En vista de esto, se procede a implementar las siguientes modificaciones al conjunto de datos.

→ Se relacionan todos los atributos, exceptuando el atributo Sex, con el atributo Height y el atributo Shell_weight siguiendo la expresión:

$$atributoNuevo = \frac{atributoOriginal + Height + Shell\_weight}{3}$$

Y eliminar estos atributos del dataset.

→ Se incluyen al inicio 2 nuevas columnas con los valores originales de los atributos Height y Shell_weight

[1455]:
```
#Implementado el nuevo conjunto de datos
data11 = np.copy(data7N)

aux2 = np.transpose( np.vstack( (data7N[:,3] , data7N[:,7]) ) )
aux3 = data7N[:,3] + data7N[:,7];
data11[:,1:-1] = ( data10[:,1:-1] + aux3[:,None] )/3
data11 = np.hstack( (aux2[:,:] , data11[:,:]) )

# Se ejecuta la regresión lineal con cross validation
[ betas5 , RSSent5 , RSSeva5 , R2ent5 , R2eva5 , R2val5 ] =␣
 ↪KFoldsCVLinearRegresion(data11, 5);
```

[1456]:
```
print('Coeficientes de cada iteración:')
print('la primera fila indica el nombre de la iteración durante el k-fold cross␣
 ↪validation.\nEl resto de valores de cada columna corresponden a los␣
 ↪coeficientes de dicha iteración. \n')
print( np.vstack((np.array([1, 2, 3, 4, 5]), betas5)) )
```

```
Coeficientes de cada iteración:
la primera fila indica el nombre de la iteración durante el k-fold cross
validation.
El resto de valores de cada columna corresponden a los coeficientes de dicha
iteración.

[[1.0000 2.0000 3.0000 4.0000 5.0000]
 [3.1747 4.2340 4.0120 3.8810 4.3845]
 [4.2774 5.2398 5.1450 4.9251 5.6443]
 [0.2061 0.2649 0.2907 0.2711 0.2684]
 [-4.3816 -5.1190 -5.5421 -4.7246 -6.6734]
 [3.3826 3.8797 2.9029 2.4273 4.4587]
 [-1.8130 -2.3875 -2.1675 -2.2854 -2.1901]
 [20.0984 23.8647 24.4320 26.9233 23.0983]
 [-22.0357 -27.7910 -25.4389 -28.6608 -26.4993]
 [-2.0422 -1.4757 -2.8233 -2.0087 -1.6920]
 [0.0159 -0.0131 -0.0326 -0.0261 -0.0314]]
```

[1457]:
```
print('R^2 de cada iteración con los datos de entrenamiento:')
print(R2ent5)
```

```
R^2 de cada iteración con los datos de entrenamiento:
[0.5322 0.5343 0.5246 0.5448 0.5314]
```

[1458]:
```
print('residual sum of squares de cada iteración (RSS) con los datos de␣
 ↪entrenamiento:')
print(RSSent5)
```

```
residual sum of squares de cada iteración (RSS) con los datos de entrenamiento:
[6724.5328 8159.4175 7653.1605 7609.0536 8033.7022]
```

[1459]:
```
print('R^2 de cada iteración con los datos de prueba:')
print(R2eva5)
```

```
R^2 de cada iteración con los datos de prueba:
[0.4559 0.3485 0.5481 0.4747 0.5245]
```

[1460]:
```
print('residual sum of squares de cada iteración (RSS) con los datos de prueba:
 ↪')
print(RSSeva5)
```

```
residual sum of squares de cada iteración (RSS) con los datos de prueba:
```

```
[3073.0294 1522.6278 1960.1615 2002.9239 1587.9521]
```

[1461]:
```
print('error del modelo usando el comando cross_val_score:')
print(R2val5)
```

```
error del modelo usando el comando cross_val_score:
[[0.4970 0.6032 0.5029 0.2979 0.4354]
 [0.6175 0.5776 0.3671 0.7337 0.3276]
 [0.2625 0.3615 0.5696 0.4208 0.4730]
 [0.0829 0.6066 0.3782 0.4148 -0.4272]
 [0.5497 0.4962 0.4844 0.4095 0.5527]]
```

Como se puede apreciar, los resultados son mejores que la vez anterior, pero siguen manteniendo una precisión bastante baja: el R^2 apenas supera el valor 0.53 para el entrenamiento en todos los subconjuntos de datos, pero en la evaluación solo 4 de 5 superan los 0.45, mientras que otro se encuentra por debajo de 0.35.

Esto indica que las variables usadas en las pruebas realizadas no garantizan una precisión suficiente.

1.4 Conclusiones

Como se pudo observar en las pruebas realizadas, el generar datos a partir de combinación de datos de diferentes variables (atributos) resulta beneficioso en ciertos casos. Si bien algunas incrementan la precisión, otras decrementan la misma.

Por otra parte, se debe de considerar la baja correlación que existen entre las variables de entrada (todos los atributos excepto el atributo Rings) con respecto a la variable de salida (el atributo Rings), y la alta correlación entre los atributos de entradas, exceptuando el atributo Sex, lo que indica la falta de tendencia del resto de atributos respecto a Rings.

Esto se corrobora con los diagramas de dispersión, donde se observa una gran nube de datos en todos los casos para diferentes valores del atributo Rings. Esto significa que diferentes valores de los demás atributos corresponden a un mismo valor de Ring y el rango de estos valores es amplio. En este sentido, al existir tantas combinaciones que permiten obtener el mismo valor de Rings, puede ser necesario implementar un modelo más complejo (no lineal) o combinaciones de inputs más complejas que permitan modelar el comportamiento de los datos para que la regresión lineal sea más efectiva.

En conclusión, se logró implementar una regresión lineal con validación cruzada a un conjunto de datos, posterior a su estudio estadístico y, con base a los resultados de su estudio estadístico, se propusieron diferentes combinaciones de atributos para mejorar el desempeño de la regresión lineal, logrando tener menores valores en el RSS y valores ligeramente más cercanos al máximo valor de R^2, pero aún se mantiene distante de lo deseado.

[ ]:
```

```