Actividad 2: Multicolinealidad

Mario Alberto Castañeda Martínez - A01640152

Victor Hugo Arreola Elenes - A01635682

Luis Manuel Orozco Yáñez - A01707822

Fernando Ojeda Marín - A01639252

Se importan las librerías necesarias:

```
from google.colab import drive
drive.mount('/content/drive')
%cd "/content/drive/MyDrive/CONCENTRACION AI/data"
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
/content/drive/MyDrive/CONCENTRACION AI/data
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
import  scipy.stats as stats
import seaborn as sns
from scipy.stats import norm, uniform, skewnorm
import statsmodels.formula.api as smf
from sklearn.metrics import mean_squared_error
```

Se importa la base de datos "Abalone" para realizar la actividad, además se visualizan las variables del modelo sus métricas básicas:

```
df = pd.read_csv('abalone.csv')

df.head()
```

```
  Sex  Length  Diameter  Height  Whole weight  Shucked weight  Viscera
weight  \
0   M   0.455     0.365   0.095        0.5140          0.2245
0.1010
1   M   0.350     0.265   0.090        0.2255          0.0995
0.0485
2   F   0.530     0.420   0.135        0.6770          0.2565
0.1415
3   M   0.440     0.365   0.125        0.5160          0.2155
0.1140
```

```
4   I   0.330     0.255   0.080        0.2050          0.0895
0.0395

   Shell weight  Rings
0         0.150     15
1         0.070      7
2         0.210      9
3         0.155     10
4         0.055      7
```

```
df.describe()
```

```
            Length     Diameter       Height  Whole weight  Shucked
weight  \
count  4177.000000  4177.000000  4177.000000   4177.000000
4177.000000
mean      0.523992     0.407881     0.139516      0.828742
0.359367
std       0.120093     0.099240     0.041827      0.490389
0.221963
min       0.075000     0.055000     0.000000      0.002000
0.001000
25%       0.450000     0.350000     0.115000      0.441500
0.186000
50%       0.545000     0.425000     0.140000      0.799500
0.336000
75%       0.615000     0.480000     0.165000      1.153000
0.502000
max       0.815000     0.650000     1.130000      2.825500
1.488000

       Viscera weight  Shell weight        Rings
count     4177.000000   4177.000000  4177.000000
mean         0.180594      0.238831     9.933684
std          0.109614      0.139203     3.224169
min          0.000500      0.001500     1.000000
25%          0.093500      0.130000     8.000000
50%          0.171000      0.234000     9.000000
75%          0.253000      0.329000    11.000000
max          0.760000      1.005000    29.000000
```

Se define el dataframe quitando la varibles "Sex", además se establece un dataframe "x" que contiene todas las variables independientes y un dataframe "y" que contiene la variable de respuesta "Rings":

```
df = df.drop(['Sex'], axis=1)

x = df.drop(['Rings'], axis=1 )
y=df['Rings']
```

Con los datos originales, se obtiene un modelo de regresión, generando los parámetros del modelo:

```
x_fit=sm.add_constant(x)
model=sm.OLS(y,x_fit)
fitted_model=model.fit()
print(fitted_model.params)
```

```
const              2.985154
Length            -1.571897
Diameter          13.360916
Height            11.826072
Whole weight       9.247414
Shucked weight   -20.213913
Viscera weight    -9.829675
Shell weight       8.576242
dtype: float64
```

Teniendo los parámetros, se saca el R^2 del modelo generado:

```
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score, make_scorer
print('Coeficiente de determinación: ', fitted_model.rsquared)
```

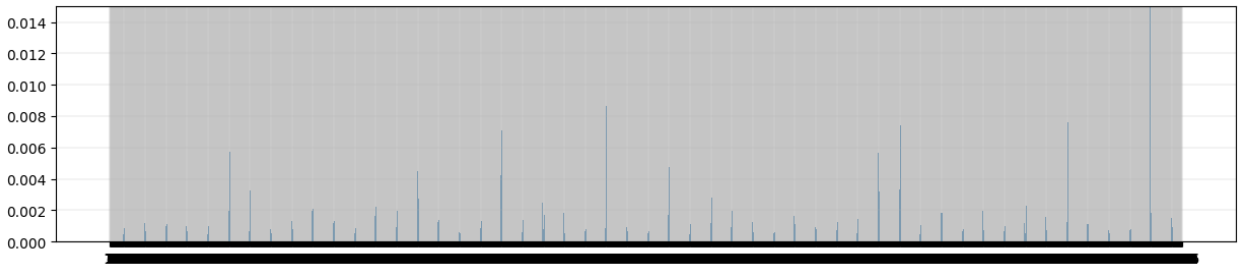
```
Coeficiente de determinación:  0.5276299399919839
```

El modelo de regresión inicial, aplicado a los datos originales, indica un coeficiente de determinación del 52% lo cual puede considerarse como un modelo de mediana/mediocre capacidad para predecir la variable de respuesta. A partir de esto, se estarán analizando los datos para observar si existen valores atípicos y cómo afecta al r^2 si se transforman los datos.

Se obtienen los scores de influencia y matrix hat de la base de datos:

```
influence = fitted_model.get_influence()
h_diag= influence.hat_matrix_diag
print(h_diag)
plt.figure(figsize=(15,3))
plt.bar(df.index,h_diag,width=0.1)
plt.ylim(0,0.015)
plt.xticks(df.index)
plt.grid(linewidth=0.2)
plt.show()
```

```
[0.00089205 0.00076875 0.00072514 ... 0.00160134 0.00103437
0.0033281 ]
```



Se obtienen los puntos leverage:

```
mapping = sorted(list(enumerate (h_diag)), key=lambda item: item[1],
reverse=True)
max_value_idxs = [item[0] for item in mapping]
print ("Top leverage values:")
print([item[1] for item in mapping][:5])
print("\nSample indexes with more leverage:")
print(df.iloc[max_value_idxs])

Top leverage values:
[0.5019723528421322, 0.05960859244317343, 0.05295671927323318,
0.03534619653809162, 0.03225467110113976]

Sample indexes with more leverage:
      Length  Diameter  Height  Whole weight  Shucked weight  Viscera
weight  \
2051   0.455     0.355   1.130        0.5940          0.3320
0.1160
1210   0.185     0.375   0.120        0.4645          0.1960
0.1045
1417   0.705     0.565   0.515        2.2100          1.1075
0.4865
3518   0.710     0.570   0.195        1.3480          0.8985
0.4435
163    0.725     0.560   0.210        2.1410          0.6500
0.3980
...      ...       ...     ...           ...             ...
...
837    0.475     0.365   0.125        0.5465          0.2290
0.1185
600    0.535     0.420   0.145        0.9260          0.3980
0.1965
3555   0.535     0.415   0.135        0.7800          0.3165
0.1690
2744   0.480     0.375   0.120        0.5895          0.2535
0.1280
488    0.540     0.420   0.135        0.8075          0.3485
0.1795

      Shell weight  Rings
```

```
2051         0.1335      8
1210         0.1500      6
1417         0.5120     10
3518         0.4535     11
163          1.0050     18
...             ...    ...
837          0.1720      9
600          0.2500     17
3555         0.2365      8
2744         0.1720     11
488          0.2350     11

[4177 rows x 8 columns]
```

Ahora se procede calcular y visualizar la distancia de Cook para encontrar puntos influyentes:

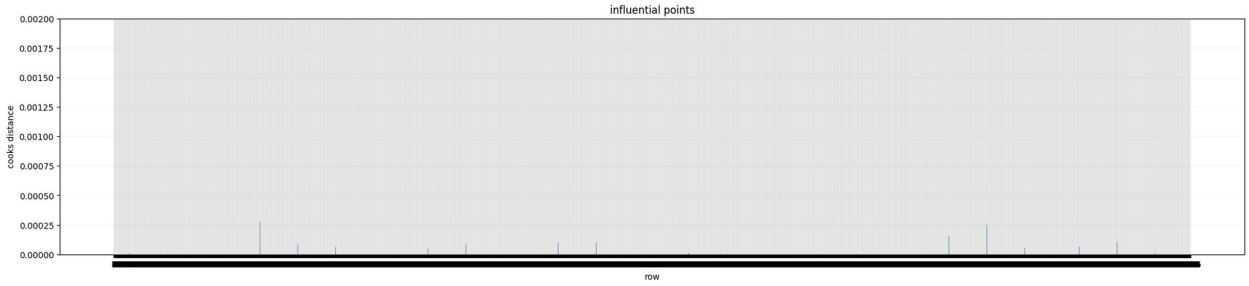
```
#distancia de cook y puntos influyentes:
np.set_printoptions(suppress=True)
cooks_dist = influence.cooks_distance[0]
print(cooks_dist)

[0.00087893 0.0000011  0.00006288 ... 0.00014442 0.00000386
0.00007827]

summary_cooks = influence.summary_frame()
print(summary_cooks)

      dfb_const  dfb_Length  dfb_Diameter  dfb_Height  dfb_Whole
weight  \
0      0.013647   -0.032927      0.045769   -0.048025
0.010240
1     -0.001956    0.000391      0.000189    0.000044
0.000010
2      0.009360    0.001597     -0.007949    0.001964          -
0.006124
3      0.002439   -0.008192      0.008150    0.001118          -
0.000097
4      0.000109   -0.000049      0.000021   -0.000022
0.000027
...         ...         ...           ...         ...                ..
.
4172  -0.001359   -0.001517      0.002022    0.001833          -
0.002014
4173  -0.001104    0.002819     -0.002169   -0.001326          -
0.000042
4174   0.002614    0.006257     -0.002471   -0.024792
0.004487
4175  -0.001575    0.001016     -0.000027   -0.001537          -
0.003201
4176   0.006267    0.000073     -0.003912   -0.000831
```

```
0.004942

      dfb_Shucked weight  dfb_Viscera weight  dfb_Shell weight
cooks_d  \
0              -0.006130           -0.016609         -0.009500
8.789307e-04
1              -0.000171           -0.000053         -0.000109
1.104468e-06
2               0.010278            0.005830          0.006062
6.288230e-05
3              -0.000220            0.000446         -0.001137
1.370315e-05
4              -0.000009           -0.000013         -0.000019
2.800798e-09
...                  ...                 ...               ...
...
4172           -0.000165            0.004425         -0.000339
5.445967e-06
4173            0.000044           -0.000011         -0.000184
1.825037e-06
4174           -0.002879           -0.012517          0.005699
1.444179e-04
4175            0.002913            0.002542          0.001413
3.855849e-06
4176            0.006734           -0.009569         -0.001904
7.827310e-05

      standard_resid  hat_diag  dffits_internal  student_resid
dffits
0           2.806306  0.000892         0.083854       2.808623
0.083923
1          -0.107167  0.000769        -0.002972      -0.107155 -
0.002972
2          -0.832610  0.000725        -0.022429      -0.832579 -
0.022428
3           0.328052  0.001018         0.010470       0.328017
0.010469
4           0.004717  0.001006         0.000150       0.004717
0.000150
...              ...       ...              ...            ...       .
..
4172        0.193886  0.001158         0.006601       0.193864
0.006600
4173        0.127301  0.000900         0.003821       0.127286
0.003821
4174       -0.848723  0.001601        -0.033990      -0.848695 -
0.033989
4175        0.172601  0.001034         0.005554       0.172581
0.005553
```

```
4176         0.433041  0.003328          0.025024        0.432999
0.025021

[4177 rows x 14 columns]
```

Se formula una gráfica para observar posibles puntos influyentes:

```
#graficar distancias de los puntos:
plt.figure(figsize=(25,5))
plt.bar(df.index, cooks_dist, width=0.01)
plt.xticks(df.index);
plt.ylim(0,0.002)
plt.xlabel('row')
plt.ylabel('cooks distance')
plt.title('influential points')
plt.grid(linewidth=0.1)
```




```
mapping_1 = sorted(list(enumerate(cooks_dist)), key=lambda item:
item[1], reverse=True)
max_values_idxs_1 = [item[0] for item in mapping_1]

print('Top cooks distance values: ')
print([item[1] for item in mapping][:5])

print('Top sample indexes with more distance values: ')
print(df.iloc[max_values_idxs_1])
```

```
Top cooks distance values:
[0.5019723528421322, 0.05960859244317343, 0.05295671927323318,
0.03534619653809162, 0.03225467110113976]
Top sample indexes with more distance values:
      Length  Diameter  Height  Whole weight  Shucked weight  Viscera
weight  \
2051   0.455     0.355   1.130        0.5940          0.3320
0.1160
2627   0.275     0.205   0.070        0.1055          0.4950
0.0190
480    0.700     0.585   0.185        1.8075          0.7055
0.3215
3518   0.710     0.570   0.195        1.3480          0.8985
```

```
0.4435
1528   0.725     0.575   0.240         2.2100           1.3510
0.4130
...      ...       ...     ...            ...              ...
...
2522   0.545     0.450   0.150         0.8795           0.3870
0.1500
2369   0.560     0.440   0.170         0.9445           0.3545
0.2175
1272   0.475     0.355   0.100         0.5035           0.2535
0.0910
1022   0.640     0.500   0.170         1.5175           0.6930
0.3260
897    0.265     0.195   0.060         0.0920           0.0345
0.0250

      Shell weight  Rings
2051         0.1335      8
2627         0.0315      5
480          0.4750     29
3518         0.4535     11
1528         0.5015     13
...             ...    ...
2522         0.2625     11
2369         0.3000     12
1272         0.1400      8
1022         0.4090     11
897          0.0245      6

[4177 rows x 8 columns]
```

```
mean_cooks = np.mean(cooks_dist)
mean_cooks
```

```
0.0018730877579285728
```

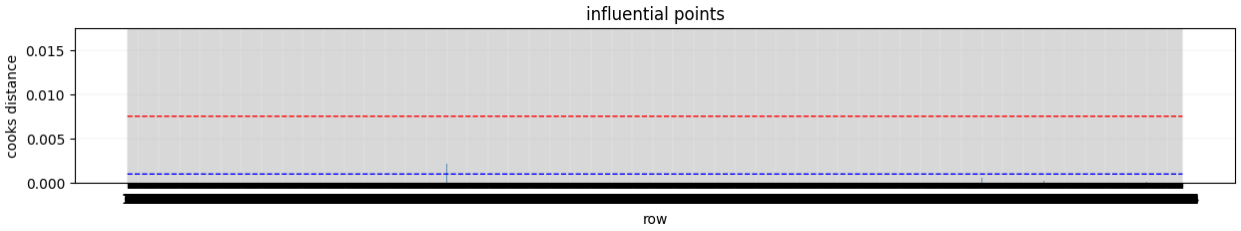
Se puede decir que un umbral para la distancia de Cook es 4/n lo que vendría siendo 4/4177 que da como resultado 0.000957, por lo que los datos con una distancia de Cook mayor, se pueden considerar como influyentes.

```
mean_cooks_list = [4*mean_cooks for _ in df.index]
cooks_threshold = [4/len(cooks_dist) for _ in df.index]
```

Ahora se visualiza una gráfica de los puntos influyentes del dataset:

```
plt.figure(figsize=(15,2))
plt.bar(df.index, cooks_dist, width=0.01)
plt.plot(df.index, mean_cooks_list, color='red', linestyle='--',
linewidth=1)
```

```
plt.plot(df.index, cooks_threshold, color='blue',
linestyle='--',linewidth=1)
plt.xticks(df.index)
plt.ylim(top=max(mean_cooks_list+ cooks_threshold) + 1e-2)
plt.xlabel('row')
plt.ylabel('cooks distance')
plt.title('influential points')
plt.grid(linewidth=0.1)
plt.show()
```



De una forma más clara y a partir de los umbrales establecidos para saber cuáles son los puntos influyentes del modelo, se obtiene el número específico de puntos influyentes y no influyentes del dataframe:

```
#puntos de influencia
influential_points = df.index[cooks_dist > 4/len(cooks_dist)]
print(influential_points)

df.iloc[influential_points,:].head(10)


Int64Index([   6,    9,   32,   33,   36,   67,   72,   81,   83,
85,
            ...
            3944, 3958, 3987, 3992, 3993, 3996, 4017, 4140, 4145,
4148],
           dtype='int64', length=253)

    Length  Diameter  Height  Whole weight  Shucked weight  Viscera
weight  \
6    0.530     0.415   0.150        0.7775          0.2370
0.1415
9    0.550     0.440   0.150        0.8945          0.3145
0.1510
32   0.665     0.525   0.165        1.3380          0.5515
0.3575
33   0.680     0.550   0.175        1.7980          0.8150
0.3925
36   0.540     0.475   0.155        1.2170          0.5305
0.3075
67   0.595     0.495   0.185        1.2850          0.4160
0.2240
```

```
72   0.595     0.475   0.170         1.2470           0.4800
0.2250
81   0.620     0.510   0.175         1.6150           0.5105
0.1920
83   0.595     0.475   0.160         1.3175           0.4080
0.2340
85   0.570     0.465   0.180         1.2950           0.3390
0.2225

    Shell weight  Rings
6           0.330     20
9           0.320     19
32          0.350     18
33          0.455     19
36          0.340     16
67          0.485     13
72          0.425     20
81          0.675     12
83          0.580     21
85          0.440     12
```

Se obtiene que hay 253 puntos influyentes en el modelo.

Ahora se obtienen los puntos no influyentes del modelo:

```
noninfluential_point = df.index[cooks_dist < 4/len(cooks_dist)]
print(noninfluential_point)
df.iloc[noninfluential_point,:].head(10)

Int64Index([   0,    1,    2,    3,    4,    5,    7,    8,   10,
11,
            ...
            4167, 4168, 4169, 4170, 4171, 4172, 4173, 4174, 4175,
4176],
           dtype='int64', length=3924)

    Length  Diameter  Height  Whole weight  Shucked weight  Viscera
weight  \
0    0.455     0.365   0.095        0.5140          0.2245
0.1010
1    0.350     0.265   0.090        0.2255          0.0995
0.0485
2    0.530     0.420   0.135        0.6770          0.2565
0.1415
3    0.440     0.365   0.125        0.5160          0.2155
0.1140
4    0.330     0.255   0.080        0.2050          0.0895
0.0395
5    0.425     0.300   0.095        0.3515          0.1410
0.0775
```

```
7    0.545     0.425   0.125         0.7680          0.2940
0.1495
8    0.475     0.370   0.125         0.5095          0.2165
0.1125
10   0.525     0.380   0.140         0.6065          0.1940
0.1475
11   0.430     0.350   0.110         0.4060          0.1675
0.0810
```

```
    Shell weight  Rings
0          0.150     15
1          0.070      7
2          0.210      9
3          0.155     10
4          0.055      7
5          0.120      8
7          0.260     16
8          0.165      9
10         0.210     14
11         0.135     10
```

Se obtiene que hay 3924 puntos no influyentes en el modelo,

Ahora que ya se sabe cuáles son los puntos influyentes en el modelo o dataset, se va a proceder para eliminarlos para ver cómo se comporta el modelo y ver si se tiene un R^2 mayor:

```
df_sin_influyentes = df.drop(influential_points)

x_sin = df_sin_influyentes.drop(['Rings'], axis=1 )
y_sin =df_sin_influyentes['Rings']
```

Se hace un modelo de regresión sin considerar los puntos influyentes:

```
x_fit_1=sm.add_constant(x_sin)
model=sm.OLS(y_sin,x_fit_1)
fitted_model=model.fit()
print(fitted_model.params)
print('Coeficiente de determinación: ', fitted_model.rsquared)
```

```
const              2.741742
Length            -2.132312
Diameter          12.486342
Height            19.318862
Whole weight      10.559372
Shucked weight   -21.182480
Viscera weight   -11.227879
Shell weight       5.158496
```

```
dtype: float64
Coeficiente de determinación:  0.5863022190662366
```

De acuerdo con el modelo sin los puntos influyentes, se obtiene un R^2 del 58%, lo que indica que el modelo mejora si se eliminan los puntos influyentes que son 253. Esto indica que el modelo mejora si se ignoran estos puntos.

Ahora, después de haber sacado los puntos influyentes y ver su impacto en el modelo, se procederá a sacar los outliers del modelo y ver qué sucede con el modelo si se hacen transformaciones para eliminarlos:

Detectar outliers usando z-score:

```
x = x.values

up_lim = x.mean() + 3*x.std()
dw_lim = x.mean() - 3*x.std()

print('upper limit: ', up_lim)
print('\nlower limit: ', dw_lim)

print('\n number of outlier samples: ', x[(x>up_lim)|
(x<dw_lim)].shape)
```

```
upper limit:  1.3231229276223053

lower limit:  -0.5577161079636301

 number of outlier samples:  (660,)
```

Detectar outliers usando los percentiles:

```
x = df.drop(['Rings'], axis=1 )

q1 = x.quantile(0.25)
q3 = x.quantile(0.75)
iqr = q3 - q1

print('q1: ', q1)
print('\q3: ', q3)
print('\niqr: ', iqr)
outliers_iqr = (x<q1 - 1.5*iqr) | (x > q3 + 1.5*iqr)
```

```
q1:  Length           0.4500
Diameter          0.3500
Height            0.1150
Whole weight      0.4415
Shucked weight    0.1860
Viscera weight    0.0935
Shell weight      0.1300
```

```
Name: 0.25, dtype: float64
\q3:  Length            0.615
Diameter          0.480
Height            0.165
Whole weight      1.153
Shucked weight    0.502
Viscera weight    0.253
Shell weight      0.329
Name: 0.75, dtype: float64

iqr:  Length            0.1650
Diameter          0.1300
Height            0.0500
Whole weight      0.7115
Shucked weight    0.3160
Viscera weight    0.1595
Shell weight      0.1990
dtype: float64
```

```
x = x.values

print('\nnumber of outliers sample: ', x[outliers_iqr].shape)
```

```
number of outliers sample:  (276,)
```

```
x = df.drop(['Rings'], axis=1 )
```

Después de conocer cuántos outliers hay en el dataset, se procede a realizar transformaciones para eliminarlos y así observar si el modelo mejora si se eliminan o controlan los outliers:

Escalado min-max:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(-2,2))
scaler.fit(x)
print('max_values: ', scaler.data_max_)
print('\nTransformation step: ')
print(scaler.transform(x), '\n')
#print(scaler.transform)
```

```
max_values:  [0.815  0.65   1.13   2.8255 1.488  0.76   1.005 ]

Transformation step:
[[ 0.05405405  0.08403361 -1.66371681 ... -1.39878951 -1.47070441
  -1.40807175]
 [-0.51351351 -0.58823529 -1.68141593 ... -1.73503699 -1.74720211
  -1.72695566]
 [ 0.45945946  0.45378151 -1.52212389 ... -1.31271015 -1.25740619
  -1.16890882]
 ...
```

```
 [ 0.83783784  0.82352941 -1.27433628 ... -0.58910558 -0.48847926
  -0.77827603]
 [ 0.97297297  0.8907563  -1.46902655 ... -0.57431069 -0.62804477
  -0.82610862]
 [ 1.43243243  1.36134454 -1.30973451 ...  0.54068594 -0.01974984
  -0.0328849 ]]
```

```
x_std = (x - x.min(axis=0)) / (x.max(axis=0) - x.min(axis=0))
max_val = 1  # Define el valor máximo  para la escala
min_val = -1  # Define el valor mínimo  para la escala
x_scaled = x_std * (max_val - min_val) + min_val

x_scaled = scaler.transform(x)
```

Después de hacer la transformación min_max se hace el modelo para ver si mejora el R^2:

```
x_fit=sm.add_constant(x_scaled)
model=sm.OLS(y,x_fit)
fitted_model=model.fit()
print(fitted_model.params)
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score, make_scorer
print('Coeficiente de determinación: ', fitted_model.rsquared)
```

```
const    12.279647
x1       -0.290801
x2        1.987436
x3        3.340865
x4        6.527519
x5       -7.514522
x6       -1.866410
x7        2.151565
dtype: float64
Coeficiente de determinación:  0.5276299399919837
```

Al hacer la transformación min-max, se observa que si bien, los parámetros cambian a comparación del modelo original, el R^2 es el mismo, del 52%, por lo que esta transformación no afecta al modelo.

Normalización z-score:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(x)
x_scaled = scaler.transform(x)

x_fit=sm.add_constant(x_scaled)
model=sm.OLS(y,x_fit)
```

```
fitted_model=model.fit()
print(fitted_model.params)
```

```
const    9.933684
x1      -0.188751
x2       1.325777
x3       0.494591
x4       4.534288
x5      -4.486203
x6      -1.077344
x7       1.193693
dtype: float64
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score, make_scorer
print('Coeficiente de determinación: ', fitted_model.rsquared)
```

```
Coeficiente de determinación:  0.5276299399919839
```

Esta transformación de normalización z-score para controlar los outliers, saca parámetros diferentes pero aún sigue sacando el mismo R^2, no tiene un impacto en el modelo de forma positiva.

Winsoring

```
x1 = x.values

q1 = np.quantile(x1,0.25)
q3= np.quantile(x1,0.75)
iqr = q3 - q1
print('q1:', q1)
print('q3:', q3)
```

```
q1: 0.16
q3: 0.505
```

```
from scipy.stats.mstats import winsorize
x_w = winsorize(x1)

x_fitt=sm.add_constant(x_w)
model=sm.OLS(y,x_fitt)
fitted_model=model.fit()
print(fitted_model.params)
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score, make_scorer
print('Coeficiente de determinación: ', fitted_model.rsquared)
```

```
const     2.985154
x1       -1.571897
x2       13.360916
x3       11.826072
```

```
x4        9.247414
x5      -20.213913
x6       -9.829675
x7        8.576242
dtype: float64
Coeficiente de determinación:  0.5276299399919839
```

Al hacer la transformación winsorize, se vuelve a tener un nulo efecto en el modelo, en general en esta etapa, solamente al eliminar los puntos influyentes, se pudo obtener un mejor modelo para los datos, en cambio con las transformaciones para outliers el modelo se quedó igual.

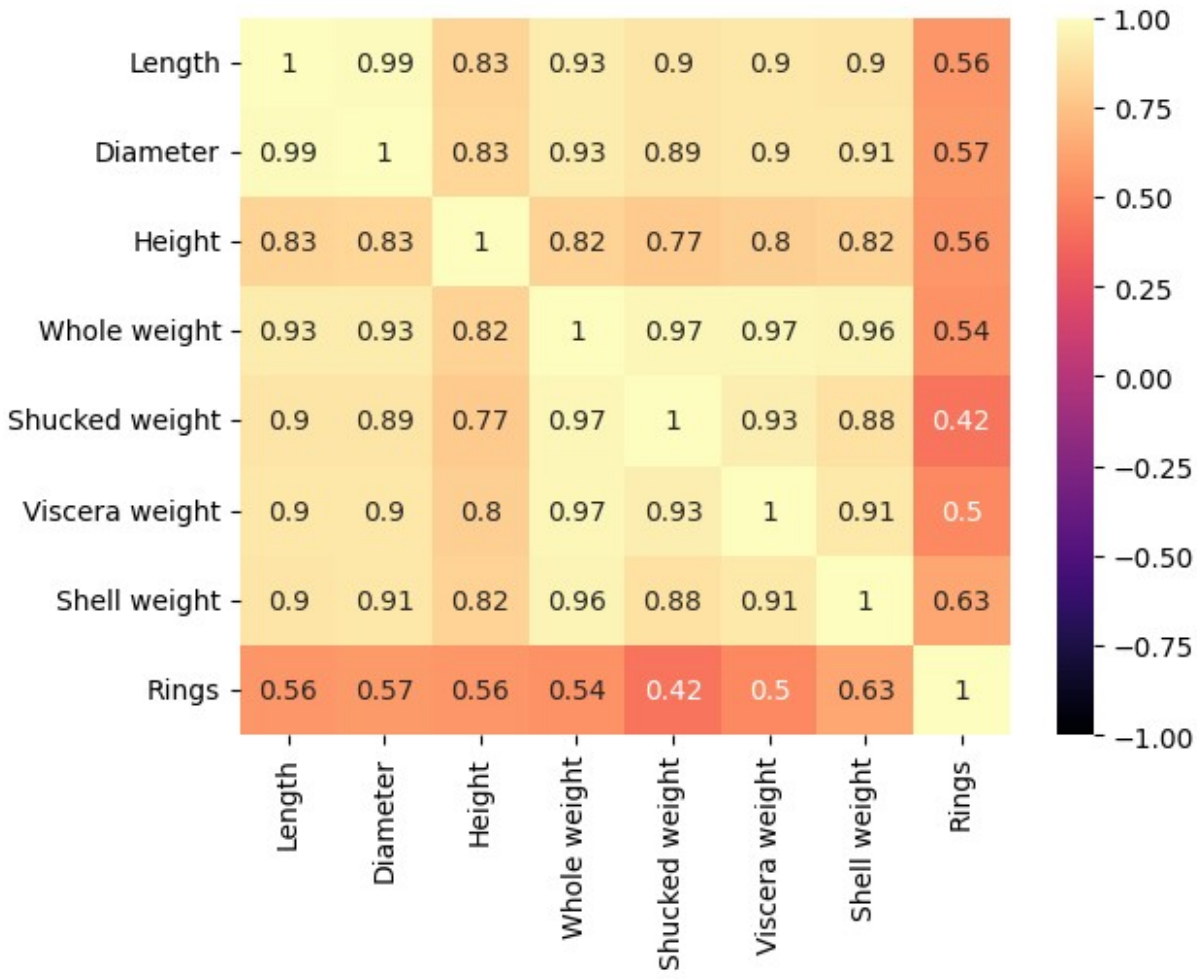
Buscar Multicolinealidad

En esta etapa, primero se mostrará el VIF y su reducción con los datos originales, luego se hará el proceso con los datos Min-Max y finalmente se hará el proceso con los datos estandarizados, tratando de disminuir el VIF de las variables (eliminando variables) y obtener diferentes modelos de regresión, interpretando el R^2 y los coeficientes obtenidos:

```
df = pd.read_csv('abalone.csv')

df = df.drop(['Sex'], axis=1)

x = df.drop(['Rings'], axis=1 )
y=df['Rings']
```

Para esta etapa, primero se observarán los coefientes de correlación entre las variables del dataset original:

```
# Se realiza un mapa de calor sobre la correlación entre las variables
sns.heatmap(df.corr(),vmin=-1, vmax=1, cmap='magma',annot = True)

<Axes: >
```

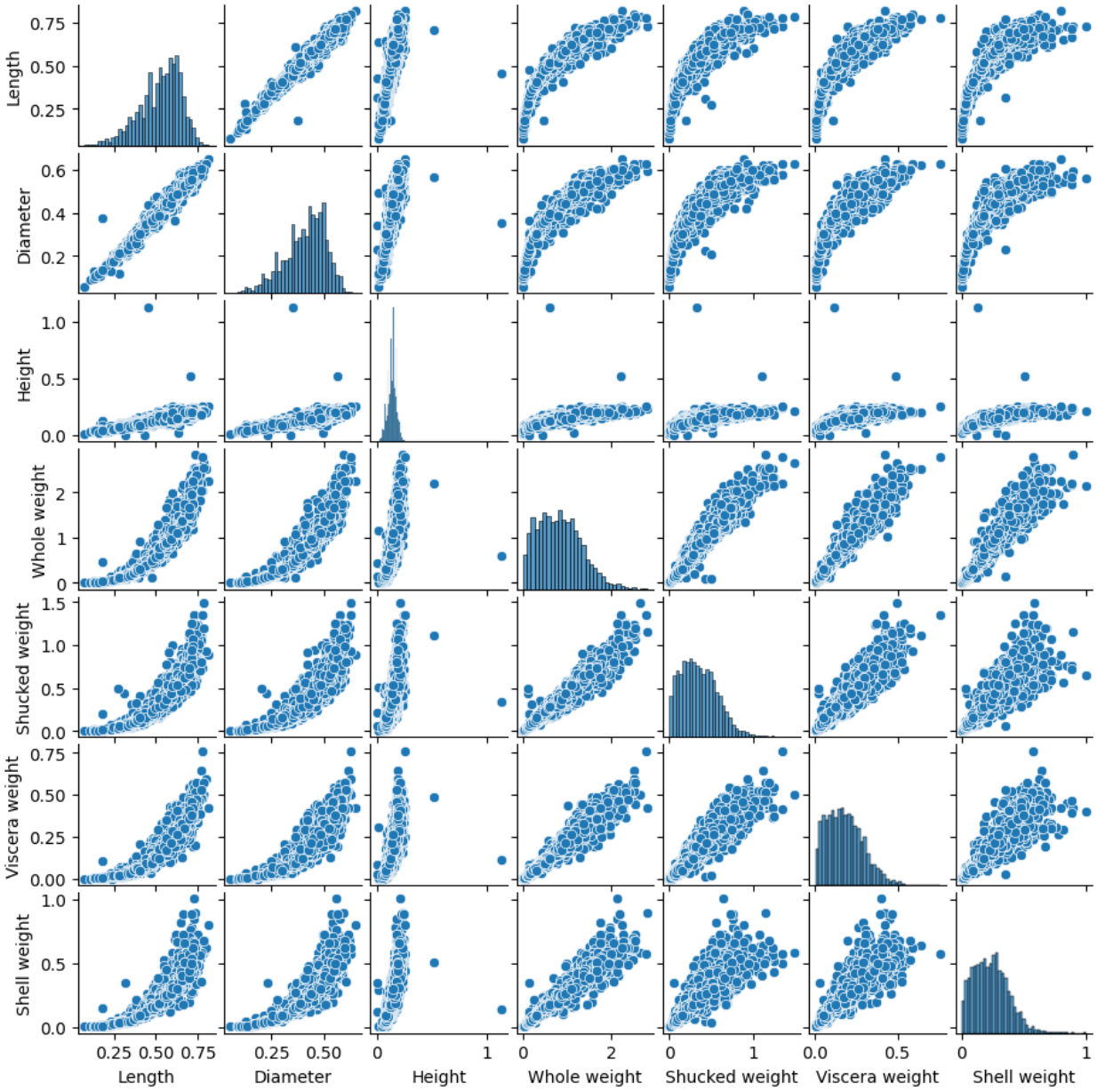


De acuerdo con la gráfica, el dataset presenta variables muy correlacionadas, lo que indica que existe colinealidad en el dataset original.

```
df.columns
```

```
Index(['Length', 'Diameter', 'Height', 'Whole weight', 'Shucked
weight',
       'Viscera weight', 'Shell weight', 'Rings'],
      dtype='object')
```

```
fig = sns.pairplot(x)
fig.fig.set_size_inches(9,9)
```



En el pairplot anterior se puede confirmar lo que se aprecia en los coeficientes de correlación, las variables presentan mucha correlación entre ellas, por lo que ahora se hará un modelo de regresión con los datos originales para ver el modelo de regresión original y tomarlo como referencia para compararlo con cambios posteriores:

```
model = sm.OLS(y, sm.add_constant(x))
fitted_model = model.fit()
print(fitted_model.summary())
```

```
                            OLS Regression Results
==============================================================================
========
```

```
Dep. Variable:                  Rings   R-squared:
0.528
Model:                            OLS   Adj. R-squared:
0.527
Method:                 Least Squares   F-statistic:
665.2
Date:                Sat, 21 Oct 2023   Prob (F-statistic):
0.00
Time:                        02:26:40   Log-Likelihood:
-9250.0
No. Observations:                4177   AIC:
1.852e+04
Df Residuals:                    4169   BIC:
1.857e+04
Df Model:                           7

Covariance Type:            nonrobust

==============================================================================
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
const          2.9852      0.269     11.092      0.000       2.458
3.513
Length        -1.5719      1.825     -0.861      0.389      -5.149
2.006
Diameter      13.3609      2.237      5.972      0.000       8.975
17.747
Height        11.8261      1.548      7.639      0.000       8.791
14.861
Whole weight   9.2474      0.733     12.622      0.000       7.811
10.684
Shucked weight -20.2139    0.823    -24.552      0.000     -21.828
-18.600
Viscera weight  -9.8297    1.304     -7.538      0.000     -12.386
-7.273
Shell weight   8.5762      1.137      7.545      0.000       6.348
10.805
==============================================================================
Omnibus:                      933.799   Durbin-Watson:
1.387
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
2602.745
Skew:                           1.174   Prob(JB):
0.00
Kurtosis:                       6.072   Cond. No.
```

```
131.
==============================================================================
========

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

```
residuos = fitted_model.resid
mse = np.mean(residuos**2)
print(f"MSE: {mse:.2f}")
```

```
MSE: 4.91
```

Ya teniendo el modelo, se buscará o confirmará la colinealidad del modelo mediante la visualización del VIF en las variables independientes:

Búsqueda VIF, primero con datos originales:

```
from statsmodels.stats.outliers_influence import
variance_inflation_factor
from statsmodels.tools.tools import add_constant

# Calcula el VIF para cada variable
vif_data = pd.DataFrame()
vif_data["Variable"] = x.columns
vif_data["VIF"] = [variance_inflation_factor(x.values, i) for i in
range(x.shape[1])]

# Mostrar los resultados
print(vif_data.sort_values('VIF', ascending=False))
```

```
         Variable         VIF
1        Diameter  748.879248
0          Length  695.083714
3    Whole weight  421.579746
4  Shucked weight  101.575906
6    Shell weight   81.772105
5  Viscera weight   63.348265
2          Height   42.117537
```

Por los VIF mostrados, se confirma que existe una profunda colinealidad en el modelo, por lo que ahora se tomarán medidas para ver si se puede solucionar este problema y se puede obtener un mejor modelo y obviamente, reducir el VIF de cada variable.

Se irán eliminando las variables de mayor VIF para ver los efectos:

```
# se elimina la variable Diameter
x1 = x.drop(['Diameter'], axis=1)
```

```python
# Calcula el VIF para cada variable
vif_data = pd.DataFrame()
vif_data["Variable"] = x1.columns
vif_data["VIF"] = [variance_inflation_factor(x1.values, i) for i in
range(x1.shape[1])]

# Mostrar los resultados
print(vif_data.sort_values('VIF', ascending=False))
```

```
         Variable         VIF
2    Whole weight  421.315969
3  Shucked weight  101.546201
5    Shell weight   80.738646
4  Viscera weight   63.267967
1          Height   41.541446
0          Length   40.828884
```

```python
model = sm.OLS(y, sm.add_constant(x1))
fitted_model = model.fit()
print(fitted_model.summary())
```

```
                            OLS Regression Results

==============================================================================
Dep. Variable:                  Rings   R-squared:
0.524
Model:                            OLS   Adj. R-squared:
0.523
Method:                 Least Squares   F-statistic:
763.8
Date:                Sat, 21 Oct 2023   Prob (F-statistic):
0.00
Time:                        02:27:00   Log-Likelihood:
-9267.8
No. Observations:                4177   AIC:
1.855e+04
Df Residuals:                    4170   BIC:
1.859e+04
Df Model:                           6

Covariance Type:            nonrobust

==============================================================================
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
const          2.9788      0.270     11.023      0.000       2.449
```

```
3.509
Length              8.1964      0.812     10.088      0.000        6.604
9.789
Height             12.8998      1.544      8.355      0.000        9.873
15.927
Whole weight        9.3558      0.735     12.721      0.000        7.914
10.798
Shucked weight    -20.2996      0.827    -24.558      0.000      -21.920
-18.679
Viscera weight    -10.1086      1.309     -7.725      0.000      -12.674
-7.543
Shell weight        9.3257      1.134      8.220      0.000        7.102
11.550
==============================================================================
========
Omnibus:                      937.096   Durbin-Watson:
1.379
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
2678.772
Skew:                           1.169   Prob(JB):
0.00
Kurtosis:                       6.150   Cond. No.
74.7
==============================================================================
========

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

```
residuos = fitted_model.resid
mse = np.mean(residuos**2)
print(f"MSE: {mse:.2f}")
```

```
MSE: 4.95
```

```
# se elimina la variable Whole weight
x1 = x1.drop(['Whole weight'], axis=1)

# Calcula el VIF para cada variable
vif_data = pd.DataFrame()
vif_data["Variable"] = x1.columns
vif_data["VIF"] = [variance_inflation_factor(x1.values, i) for i in
range(x1.shape[1])]

# Mostrar los resultados
print(vif_data.sort_values('VIF', ascending=False))
```

```
         Variable        VIF
1          Height  41.531111
0          Length  40.727775
```

```
3  Viscera weight  38.029878
2  Shucked weight  29.856146
4    Shell weight  26.107574
```

```
model = sm.OLS(y, sm.add_constant(x))
fitted_model = model.fit()
print(fitted_model.summary())
```

```
                            OLS Regression Results

==============================================================================
Dep. Variable:                  Rings   R-squared:
0.528
Model:                            OLS   Adj. R-squared:
0.527
Method:                 Least Squares   F-statistic:
665.2
Date:                Sat, 21 Oct 2023   Prob (F-statistic):
0.00
Time:                        02:27:09   Log-Likelihood:
-9250.0
No. Observations:                4177   AIC:
1.852e+04
Df Residuals:                    4169   BIC:
1.857e+04
Df Model:                           7

Covariance Type:            nonrobust

==============================================================================
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
const          2.9852      0.269     11.092      0.000       2.458
3.513
Length        -1.5719      1.825     -0.861      0.389      -5.149
2.006
Diameter      13.3609      2.237      5.972      0.000       8.975
17.747
Height        11.8261      1.548      7.639      0.000       8.791
14.861
Whole weight   9.2474      0.733     12.622      0.000       7.811
10.684
Shucked weight -20.2139      0.823    -24.552      0.000     -21.828
-18.600
Viscera weight  -9.8297      1.304     -7.538      0.000     -12.386
-7.273
```

```
Shell weight       8.5762      1.137      7.545      0.000       6.348
10.805
==============================================================================
========
Omnibus:                      933.799   Durbin-Watson:
1.387
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
2602.745
Skew:                           1.174   Prob(JB):
0.00
Kurtosis:                       6.072   Cond. No.
131.
==============================================================================
========

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

```
residuos = fitted_model.resid
mse = np.mean(residuos**2)
print(f"MSE: {mse:.2f}")
```

```
MSE: 4.91
```

```
# se elimina la variable Height
x1 = x1.drop(['Height'], axis=1)

# Calcula el VIF para cada variable
vif_data = pd.DataFrame()
vif_data["Variable"] = x1.columns
vif_data["VIF"] = [variance_inflation_factor(x1.values, i) for i in
range(x1.shape[1])]

# Mostrar los resultados
print(vif_data.sort_values('VIF', ascending=False))
```

```
          Variable        VIF
2  Viscera weight  37.950591
1  Shucked weight  29.677687
3    Shell weight  25.141156
0          Length   9.783280
```

```
model = sm.OLS(y, sm.add_constant(x1))
fitted_model = model.fit()
print(fitted_model.summary())
```

```
                            OLS Regression Results

==============================================================================
========
```

```
Dep. Variable:                    Rings   R-squared:
0.497
Model:                              OLS   Adj. R-squared:
0.496
Method:                   Least Squares   F-statistic:
1029.
Date:                  Sat, 21 Oct 2023   Prob (F-statistic):
0.00
Time:                          02:27:49   Log-Likelihood:
-9383.1
No. Observations:                 4177   AIC:
1.878e+04
Df Residuals:                     4172   BIC:
1.881e+04
Df Model:                            4

Covariance Type:             nonrobust

==============================================================================
=============
                     coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
------------
const               3.1694      0.274     11.568      0.000       2.632
3.707
Length             10.4897      0.797     13.164      0.000       8.927
12.052
Shucked weight    -11.7102      0.474    -24.679      0.000     -12.640
-10.780
Viscera weight      0.8714      1.053      0.827      0.408      -1.194
2.937
Shell weight       22.2694      0.676     32.944      0.000      20.944
23.595
==============================================================================
========
Omnibus:                      1069.660   Durbin-Watson:
1.337
Prob(Omnibus):                   0.000   Jarque-Bera (JB):
3225.402
Skew:                            1.312   Prob(JB):
0.00
Kurtosis:                        6.412   Cond. No.
39.3
==============================================================================
========

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

```
residuos = fitted_model.resid
mse = np.mean(residuos**2)
print(f"MSE: {mse:.2f}")
```

```
MSE: 5.23
```

En los bloques de código anteriores, se iban quitando las variables con mayor VIF en los datos originales, donde al final se eliminaron 3 variables, donde al final se siguen teniendo VIF muy grandes y en el modelo, se obtuvieron valores de R^2 de entre el 52% y el 49% finalmente. Ahora se hará el mismo procedimiento pero con los datos derivados de la transformación Min-Max:

```
# Calcula el VIF para cada variable
vif_data = pd.DataFrame()
vif_data["VIF"] = [variance_inflation_factor(x_scaled, i) for i in
range(x_scaled.shape[1])]

# Mostrar los resultados
print(vif_data.sort_values('VIF', ascending=False))
```

```
          VIF
3  234.115174
4   99.649079
6   91.488995
2   80.356328
5   71.349890
0   57.596879
1   53.810845
```

```
columna_a_eliminar = 3
mi_array_sin_columna = np.delete(x_scaled, columna_a_eliminar, axis=1)

# Calcula el VIF para cada variable
vif_data = pd.DataFrame()
vif_data["VIF"] = [variance_inflation_factor(mi_array_sin_columna, i)
for i in range(mi_array_sin_columna.shape[1])]
# Mostrar los resultados
print(vif_data.sort_values('VIF', ascending=False))
```

```
         VIF
0  57.548999
1  53.480526
2  46.235224
4  46.195925
5  36.002975
3  34.940052
```

```
columna_a_eliminar = 0
mi_array_sin_columna = np.delete(x_scaled, columna_a_eliminar, axis=1)
```

```
# Calcula el VIF para cada variable
vif_data = pd.DataFrame()
vif_data["VIF"] = [variance_inflation_factor(mi_array_sin_columna, i)
for i in range(mi_array_sin_columna.shape[1])]
# Mostrar los resultados
print(vif_data.sort_values('VIF', ascending=False))
```

```
          VIF
2  233.920557
3   99.474175
5   91.118686
1   78.077553
4   71.032092
0    7.430088
```

Se observa que al usar los datos transformados con Min-Max, los VIF siguen siendo altos aún al eliminar variables con VIF alto, por lo que se procederá a utilizar otra técnica para obtener mejores resultados y suprimir el VIF en las variables independientes.

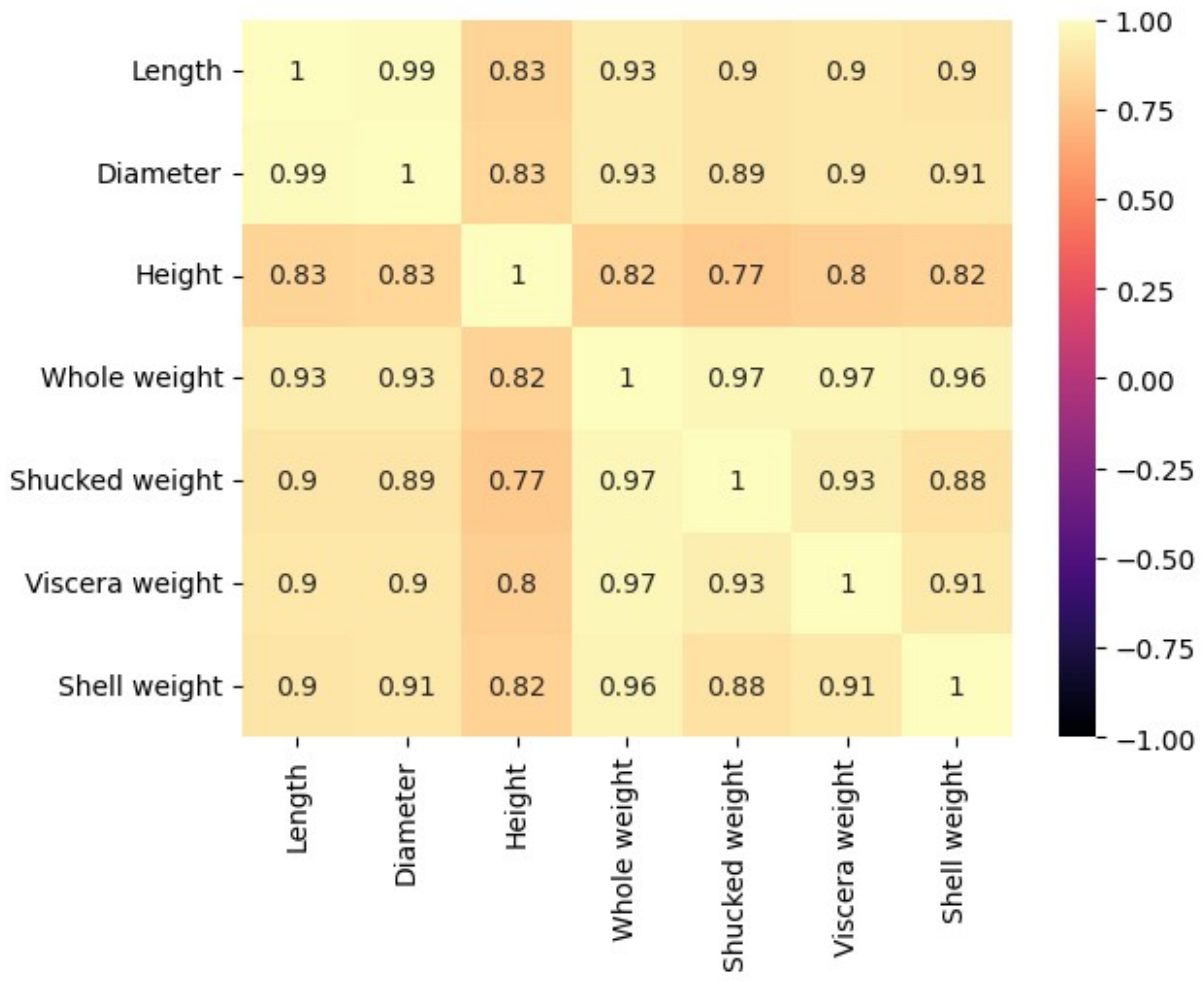
Esta nueva técnica para reducir el VIF consiste en tal cual, estandarizar los datos originales, en este caso las variables independientes, para reducir la multicolineadad y volver a eliminar variables y obtener R^2 y parámetros diferentes para ver cómo cambian:

```
df = pd.read_csv('abalone.csv')
df = df.drop(['Sex'], axis=1)
x = df.drop(['Rings'], axis=1 )
y=df['Rings']
```

Se estandarizan los datos originales:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df_estan = scaler.fit_transform(x)
df_estan = pd.DataFrame(df_estan,columns=x.columns)

# Se realiza un mapa de calor sobre la correlación entre las variables
sns.heatmap(df_estan.corr(),vmin=-1, vmax=1, cmap='magma',annot =
True)
```

```
<Axes: >
```



Al obtener los coeficientes de correlación con los datos estandarizados, se siguen observando variables muy correlacionadas, por lo que se tiene que seguir con la reducción de multicolinealidad.

```
# Calcula el VIF para cada variable
vif_data = pd.DataFrame()
vif_data["Variable"] = df_estan.columns
vif_data["VIF"] = [variance_inflation_factor(df_estan.values, i) for i
in range(df_estan.shape[1])]

# Mostrar los resultados
print(vif_data.sort_values('VIF', ascending=False))
```

```
          Variable         VIF
3    Whole weight  109.592750
1        Diameter   41.845452
0          Length   40.771813
4  Shucked weight   28.353191
6    Shell weight   21.258289
```

```
5  Viscera weight   17.346276
2          Height    3.559939
```

Los VIF obtenidos son considerablemente menores que en los datos originales y que en los datos con Min-Max. Se procede a obtener un modelo de regresión:

```
model = sm.OLS(y, sm.add_constant(df_estan))
fitted_model = model.fit()
print(fitted_model.summary())
```

```
                            OLS Regression Results

==============================================================================
Dep. Variable:                  Rings   R-squared:
0.528
Model:                            OLS   Adj. R-squared:
0.527
Method:                 Least Squares   F-statistic:
665.2
Date:                Sat, 21 Oct 2023   Prob (F-statistic):
0.00
Time:                        02:12:15   Log-Likelihood:
-9250.0
No. Observations:                4177   AIC:
1.852e+04
Df Residuals:                    4169   BIC:
1.857e+04
Df Model:                           7

Covariance Type:            nonrobust

==============================================================================
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
const              9.9337      0.034    289.481      0.000       9.866
10.001
Length            -0.1888      0.219     -0.861      0.389      -0.618
0.241
Diameter           1.3258      0.222      5.972      0.000       0.891
1.761
Height             0.4946      0.065      7.639      0.000       0.368
0.622
Whole weight       4.5343      0.359     12.622      0.000       3.830
5.239
Shucked weight    -4.4862      0.183    -24.552      0.000      -4.844
```

```
-4.128
Viscera weight    -1.0773      0.143     -7.538      0.000      -1.358
-0.797
Shell weight       1.1937      0.158      7.545      0.000       0.884
1.504
==============================================================================
========
Omnibus:                      933.799   Durbin-Watson:
1.387
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
2602.745
Skew:                           1.174   Prob(JB):
0.00
Kurtosis:                       6.072   Cond. No.
30.9
==============================================================================
========

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

```
residuos = fitted_model.resid
mse = np.mean(residuos**2)
print(f"MSE: {mse:.2f}")
```

```
MSE: 4.91
```

El modelo obtenido indica un r^2 de 52%, donde se observan que los parámetros son en su mayoría pequeños, esto porque seguramente al estandarizar datos, se disminuyen los valores de los datos. Pero aún existe mucha colinealidad, por lo que se va a proceder con eliminar variables:

```
x1 = df_estan.drop(['Whole weight'], axis=1)

# Calcula el VIF para cada variable
vif_data = pd.DataFrame()
vif_data["Variable"] = x1.columns
vif_data["VIF"] = [variance_inflation_factor(x1.values, i) for i in
range(x1.shape[1])]

# Mostrar los resultados
print(vif_data.sort_values('VIF', ascending=False))
```

```
          Variable        VIF
1         Diameter  41.819755
0           Length  40.763955
4  Viscera weight   10.697780
3  Shucked weight    8.852112
5    Shell weight    7.817781
2           Height    3.558443
```

```
model = sm.OLS(y, sm.add_constant(x1))
fitted_model = model.fit()
print(fitted_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Rings   R-squared:                       0.510
Model:                            OLS   Adj. R-squared:                  0.509
Method:                 Least Squares   F-statistic:                     722.1
Date:                Sat, 21 Oct 2023   Prob (F-statistic):               0.00
Time:                        02:12:33   Log-Likelihood:                -9328.3
No. Observations:                4177   AIC:                         1.867e+04
Df Residuals:                    4170   BIC:                         1.871e+04
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          9.9337      0.035    284.137      0.000       9.865      10.002
Length        -0.2271      0.223     -1.018      0.309      -0.665       0.210
Diameter       1.3952      0.226      6.171      0.000       0.952       1.838
Height         0.5113      0.066      7.753      0.000       0.382       0.641
Shucked weight -2.5735     0.104    -24.741      0.000      -2.777      -2.370
Viscera weight  0.0395     0.114      0.345      0.730      -0.185       0.264
Shell weight   2.7816      0.098     28.456      0.000       2.590       2.973
==============================================================================
Omnibus:                     1037.149   Durbin-Watson:                   1.367
```

```
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
3176.648
Skew:                           1.266   Prob(JB):
0.00
Kurtosis:                       6.441   Cond. No.
20.6
================================================================================
========

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

```
residuos = fitted_model.resid
mse = np.mean(residuos**2)
print(f"MSE: {mse:.2f}")
```

```
MSE: 5.10
```

Al eliminar la variable whole weight, se obtienen VIF menores lo que indica que se está reduciendo la multicolinealidad. En el caso del modelo de regresión, los parámetros siguen siendo pequeños y el R^2 cambia de tal forma que se reduce un poco, pasa a un 51%, esto puede deberse a que se eliminó una variable. A continuación eliminará otra variable con un VIF alto:

```
x1 = x1.drop(['Diameter'], axis=1)

# Calcula el VIF para cada variable
vif_data = pd.DataFrame()
vif_data["Variable"] = x1.columns
vif_data["VIF"] = [variance_inflation_factor(x1.values, i) for i in
range(x1.shape[1])]

# Mostrar los resultados
print(vif_data.sort_values('VIF', ascending=False))
```

```
          Variable        VIF
3  Viscera weight  10.690504
2  Shucked weight   8.851834
0          Length   8.013867
4    Shell weight   7.457755
1          Height   3.509983
```

```
model = sm.OLS(y, sm.add_constant(x1))
fitted_model = model.fit()
print(fitted_model.summary())
```

```
                            OLS Regression Results

================================================================================
```

```
========
Dep. Variable:                   Rings   R-squared:
0.505
Model:                             OLS   Adj. R-squared:
0.505
Method:                  Least Squares   F-statistic:
851.4
Date:                 Sat, 21 Oct 2023   Prob (F-statistic):
0.00
Time:                         02:11:23   Log-Likelihood:
-9347.3
No. Observations:                4177   AIC:
1.871e+04
Df Residuals:                    4171   BIC:
1.874e+04
Df Model:                           5

Covariance Type:             nonrobust

================================================================================
============
                     coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
------------
const              9.9337      0.035    282.882      0.000       9.865
10.003
Length             1.0075      0.099     10.135      0.000       0.813
1.202
Height             0.5588      0.066      8.494      0.000       0.430
0.688
Shucked weight    -2.5699      0.104    -24.598      0.000      -2.775
-2.365
Viscera weight     0.0211      0.115      0.183      0.854      -0.204
0.246
Shell weight       2.9111      0.096     30.356      0.000       2.723
3.099
================================================================================
========
Omnibus:                     1040.962   Durbin-Watson:
1.358
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
3288.926
Skew:                           1.259   Prob(JB):
0.00
Kurtosis:                       6.544   Cond. No.
8.38
================================================================================
========
```

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

```
residuos = fitted_model.resid
mse = np.mean(residuos**2)
print(f"MSE: {mse:.2f}")
```

```
MSE: 5.10
```

Al eliminar la variable "Diameter", los VIF obtenidos vuelven a disminuir de forma importante, tanto que ahora la variable con más VIF tiene 10.69, esto indica que el dataframe está mejorando y seguramente solo necesite que se elimine otra variable para tener un dataset sin el problema de la multicolinealidad. En el caso del modelo de regresión obtenido, arroja un R^2 de 50%, lo cual indica que baja muy poco a comparación del anterior. Los coeficientes son muy parecidos a los anteriores y solo cambian por decimales.

```
x1 = x1.drop(['Viscera weight'], axis=1)

# Calcula el VIF para cada variable
vif_data = pd.DataFrame()
vif_data["Variable"] = x1.columns
vif_data["VIF"] = [variance_inflation_factor(x1.values, i) for i in
range(x1.shape[1])]

# Mostrar los resultados
print(vif_data.sort_values('VIF', ascending=False))
```

```
         Variable       VIF
0          Length  7.746817
3    Shell weight  6.598972
2  Shucked weight  6.114777
1          Height  3.489531
```

```
model = sm.OLS(y, sm.add_constant(x1))
fitted_model = model.fit()
print(fitted_model.summary())
```

```
                            OLS Regression Results

==============================================================================
========
Dep. Variable:                   Rings   R-squared:
0.505
Model:                             OLS   Adj. R-squared:
0.505
Method:                  Least Squares   F-statistic:
1064.
Date:                Sat, 21 Oct 2023   Prob (F-statistic):
0.00
```

```
Time:                         02:13:23   Log-Likelihood:
-9347.3
No. Observations:                4177   AIC:
1.870e+04
Df Residuals:                    4172   BIC:
1.874e+04
Df Model:                           4

Covariance Type:            nonrobust

==============================================================================
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
const          9.9337      0.035    282.915      0.000       9.865
10.003
Length         1.0109      0.098     10.344      0.000       0.819
1.202
Height         0.5598      0.066      8.534      0.000       0.431
0.688
Shucked weight    -2.5592      0.087    -29.476      0.000      -2.729
-2.389
Shell weight       2.9170      0.090     32.341      0.000       2.740
3.094
==============================================================================
Omnibus:                     1040.521   Durbin-Watson:
1.358
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
3288.766
Skew:                           1.258   Prob(JB):
0.00
Kurtosis:                       6.545   Cond. No.
6.16
==============================================================================

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

```
residuos = fitted_model.resid
mse = np.mean(residuos**2)
print(f"MSE: {mse:.2f}")
```

```
MSE: 5.14
```

Al eliminar la variable "Viscera weight", se obtienen VIFs aceptables para el dataset, de forma que las 4 variables tienen VIFs que no indican una alta colinealidad, por lo que se pudo conseguir el objetivo de reducir la multicolinealidad.

En el caso del modelo de regresión obtenido finalmente, se obtiene un R^2 de 50% lo cual es muy parecido al R^2 obtenido en el primer modelo con los datos originales, esto indica que a pesar de que la estandarización de variables no garantizó un aumento de R^2, sí mejoró el tema de multicolinealidad en las variables predictoras.Igualmente, se obtiene que el MSE iba aumentando a medida que se eliminaban variables lo que puede indicar que el eliminar variables afecta también al error del modelo.

Respecto a los coeficientes del modelo obtenido al final, se tiene la comparación con los modelos de regresión con datos originales, lo cuales tienen coeficientes grandes en el modelo anterior, se tienen coeficientes pequeños, salvo por el beta0 que es mayor que los modelos con datos originales.

Análisis de Componentes Principales:

```
df = pd.read_csv('abalone.csv')
df = df.drop(['Sex'], axis=1)
x = df.drop(['Rings'], axis=1 )
y=df['Rings']

from sklearn.preprocessing import scale

from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score, make_scorer
```

Primero, se van a obtener los porcentajes de varianza explicada por cada componente en los datos originales:

```
from sklearn.decomposition import PCA

pca = PCA(n_components=7).fit(x)
print('Componentes con máxima varianza: ')
for i,c in enumerate(pca.components_):
  print(f'Componente {i} = {c}')

print('\nPorcentaje de varianza explicada por cada componente
seleccionado: ')
for i,m in enumerate(pca.explained_variance_):
  print(f'Magnitud componente {i} = {m}')
```

```
Componentes con máxima varianza:
Componente 0 = [0.19315606 0.15955208 0.05928271 0.84261922 0.37195895
0.18225102
 0.22834926]
Componente 1 = [-0.35006929 -0.31882074 -0.13475175 -0.01882402
0.70343169 -0.01294771
 -0.51216078]
```

```
Componente 2 = [-0.65543596 -0.50547308 -0.08607958  0.31147028 -
0.3372725   0.02506135
  0.30999426]
Componente 3 = [-0.0387846   0.01806045  0.00468325 -0.12797716
0.35376715 -0.76297757
  0.52391176]
Componente 4 = [-0.15584501 -0.07483574  0.92444847 -0.16797945
0.16244383  0.20728245
  0.13392483]
Componente 5 = [-5.60615302e-04  3.02034552e-02  3.37704883e-01
3.84695312e-01
 -3.18402885e-01 -5.82880918e-01 -5.43986951e-01]
Componente 6 = [-0.62028519  0.78137995 -0.0473955  -0.00624787
0.0125725   0.03373286
 -0.03332151]

Porcentaje de varianza explicada por cada componente seleccionado:
Magnitud componente 0 = 0.3381707265140742
Magnitud componente 1 = 0.0039640302535688884
Magnitud componente 2 = 0.0029077141571337397
Magnitud componente 3 = 0.0010549043403742075
Magnitud componente 4 = 0.0004896638668630571
Magnitud componente 5 = 0.00042678748206040963
Magnitud componente 6 = 0.0001481417361125
```

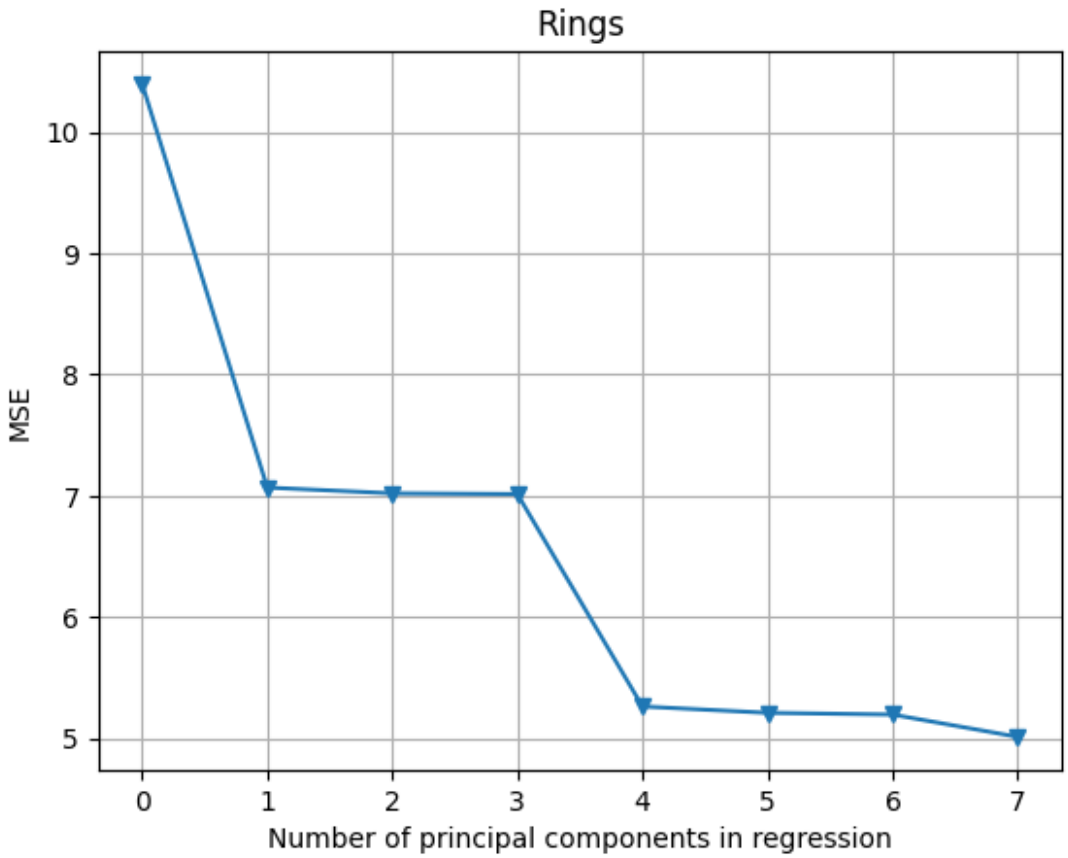
Ahora para empezar con el modelo PCR y el análisis PCA, se procederá a transformar las variables independientes, escalando los datos y observando la varianza explicada por cada componente:

```
np.set_printoptions(suppress=True, precision=3)
pca=PCA()

x_reduced = pca.fit_transform(scale(x))
print('Return a vector of the variance explained by each dimension')
print(pca.explained_variance_)
print('\nGives the variance explained solely by the i+1st dimension')
print(pca.explained_variance_ratio_)
print('\nReturn a vector x such that x[i] returns the cumulative
variance explained by the first i+1 dimensions')
print(pca.explained_variance_ratio_.cumsum())

Return a vector of the variance explained by each dimension
[6.357 0.279 0.167 0.114 0.065 0.013 0.007]

Gives the variance explained solely by the i+1st dimension
[0.908 0.04  0.024 0.016 0.009 0.002 0.001]

Return a vector x such that x[i] returns the cumulative variance
```

```
explained by the first i+1 dimensions
[0.908 0.948 0.972 0.988 0.997 0.999 1.   ]
```

Ahora se observará el número de componentes principales y cómo influyen en el modelo de regresión mediante el MSE obtenido por el modelo.

```
from sklearn import model_selection
from sklearn.linear_model import LinearRegression

#samples, principal components
n, pc = x_reduced.shape
# 10-fold CV, with shuffle
kf_10 = model_selection.KFold(n_splits=10, shuffle=True,
random_state=1)
model = LinearRegression()
mse = []
# Calculate ME with only the intercept (no principal components in
regression)
score = -1 * model_selection.cross_val_score(model, np.ones((n, 1)),
y.ravel(), cv=kf_10,scoring= 'neg_mean_squared_error').mean()

mse.append(score)

# Calculate MSE using CV for the 7 principle components,adding one
component at the time.
for i in np.arange (1, pc+1):
 score = -1 * model_selection.cross_val_score(model, x_reduced[:,:i],
y.ravel(),cv=kf_10,scoring= 'neg_mean_squared_error').mean()
 mse.append(score)

x_axis = np.arange(0, len(mse))
# Plot results
plt.plot(x_axis, mse,'-v')
plt.xlabel('Number of principal components in regression')
plt.ylabel('MSE')
plt.title('Rings')
plt.grid()
plt.xticks (x_axis); # you can comment this line
```



De primera instancia se observa que al utilizar los 7 componentes principales, el error con validación cruzada es el menor. Sin embargo se observa que el error disminuye de forma importante desde los 4 componentes principales, esto puede indicar que se puede obtener un modelo algo competente utilizando 4 variables.

Ahora se vuelve a visualizar el porcentaje de varianza de cada variable en el dataset ya transformado:

```
np.cumsum(np.round(pca.explained_variance_ratio_,decimals=4)*100)
```

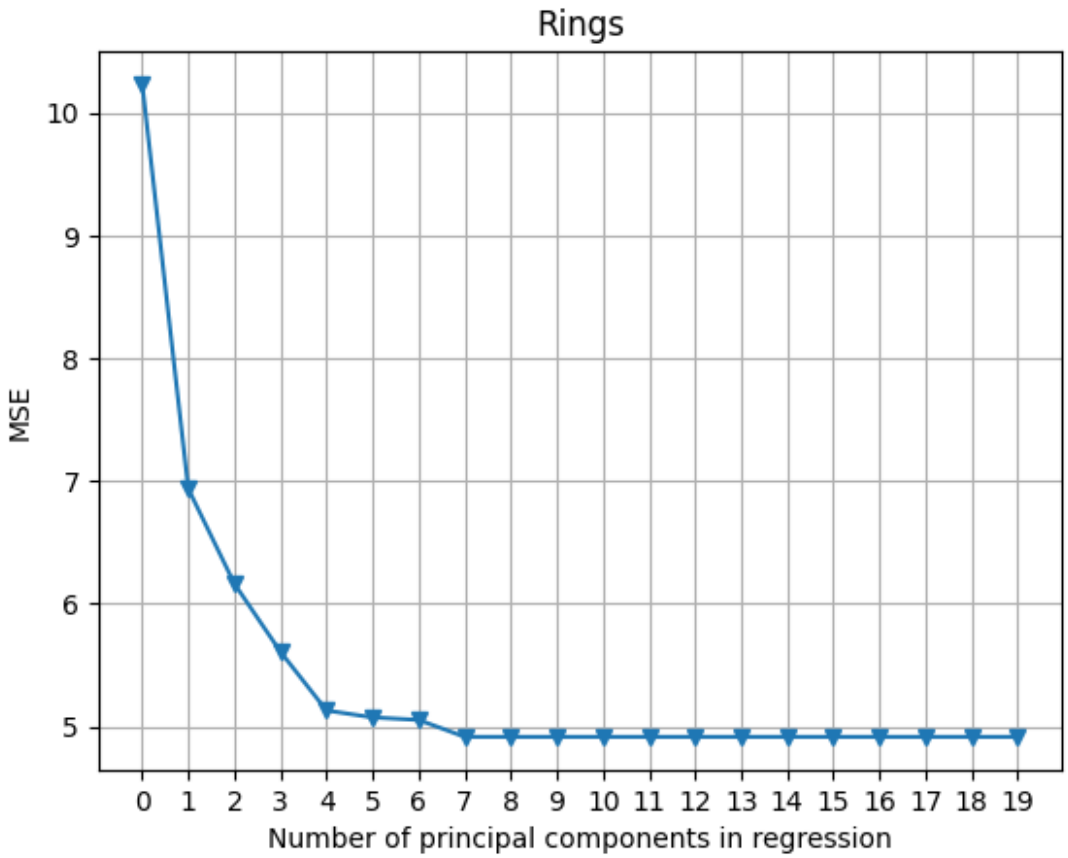
```
array([ 90.79,  94.78,  97.17,  98.8 ,  99.72,  99.9 , 100.  ])
```

Se observa que en la variable 4 y 5, ya se obtiene una importante explicación de los datos. Es importante notar que desde la primera variable, ya se explica un 90% de la varianza de los datos.

Se realiza el PCA con datos de entrenamiento, utilizando la validación cruzada:

```
pca2 = PCA()
# Split into training and test sets
x_train,x_test, y_train, y_test = model_selection. train_test_split(x,
y,test_size=0.5, random_state=1)

# Scale the data
```

```
x_reduced_train = pca2.fit_transform(scale(x_train))
n, pc = x_reduced_train.shape

# 10-fold CV, with shuffle
kf_10 = model_selection.KFold(n_splits=10, shuffle=True,
random_state=1)
model = LinearRegression ()
mse = []

# Calculate ME with only the intercept (no principal components in
regression)
score = -1*model_selection.cross_val_score(model, np.ones ((n, 1)),
y_train.ravel (),cv=kf_10, scoring='neg_mean_squared_error').mean()
mse.append (score)
# Calculate ME using CV for the 19 principle components, adding one
component at the time.
for i in np.arange(1, 20):
 score = -1*model_selection.cross_val_score (model,
x_reduced_train[:,:i],y_train.ravel(),cv=kf_10,scoring=
'neg_mean_squared_error').mean()
 mse.append(score)

x_axis = np.arange (0, len(mse))

# Plot results
plt.plot (x_axis, mse,'-v')
plt.xlabel('Number of principal components in regression')
plt.ylabel('MSE')
plt.title('Rings')
plt.grid()
plt.xticks(x_axis);  # you can comment this line
```



Con este modelo, se observa que al igual que el modelo anterior, el error del modelo más bajo, se produce cuando se utilizan los 7 componentes.

Finalmente, se observarán los valores MSE y R^2 con este modelo nuevo y los datos transformados:

```
x_reduced_test = pca2.transform(scale(x_test))[:,:7]

model = sm.OLS(y_train, sm.add_constant(x_reduced_train[:,:7]))
fitted_model = model.fit()

#prediction test data
pred = fitted_model.predict(sm.add_constant(x_reduced_test))
mse = mean_squared_error(y_test, pred)

print('mean squared error {}'.format(np.round(mse,2)))
```

```
mean squared error 5.06
```

```
print('Coeficiente de determinación: ', fitted_model.rsquared)
```

```
Coeficiente de determinación:  0.5237366134413659
```

Ya generado el PCR, se obtiene un R^2 del 52% y un MSE de 5.06, es necesario comparar este resultado con el modelo generado al estandarizar los datos y eliminar variables con alto VIF; la comparación parte que el R^2 del modelo PCR es mayor al del modelo final de VIF, el cual es igual al 50%.

La comparación con el MSE es que el error del PCR es menor al del último modelo generado en el método VIF, aunque es necesario recordar, que en el método VIF, al tener todas las variables, el error generado era menor que el PCR sin embargo al ir quitando variables independientes con VIF alto, el error iba aumentando hasta llegar a un error de 5.14, un poco mayor al del PCR.

Se puede considerar que tiene mucho que ver la transformación en el método PCR, ya que pudo utilizar todas las variables a diferencia del modelo VIF, que tuvo que eliminar variables para obtener un buen modelo.