Act 2. Multicolinealidad

Héctor Manuel Cárdenas Yáñez | A01634615

Siddhartha López Valenzuela | A00227694

Álvaro Morán Errejón | A01638034

Isaí Ambrocio | A01625101

```
!pip install ucimlrepo
```

```
Requirement already satisfied: ucimlrepo in
/usr/local/lib/python3.10/dist-packages (0.0.3)
```

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale, MinMaxScaler, StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
```

```
from google.colab import drive
drive.mount('/content/drive')
```

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

```
target_url = ("/content/abalone.data")
```

```
df = pd.read_csv(target_url, header = None, delimiter = ",")
```

```
from ucimlrepo import fetch_ucirepo

# Fetch Data
abalone = fetch_ucirepo(id = 1)

# Data (as pandas)
X = abalone.data.features
Y = abalone.data.targets

# Data Combined
df_combined = pd.concat([X, Y], axis = 1)

# Metadata
print(abalone.metadata)
```

```python
# Variable Information
print(abalone.variables)
```


```
{'uci_id': 1, 'name': 'Abalone', 'repository_url':
'https://archive.ics.uci.edu/dataset/1/abalone', 'data_url':
'https://archive.ics.uci.edu/static/public/1/data.csv', 'abstract':
'Predict the age of abalone from physical measurements', 'area': 'Life
Science', 'tasks': ['Classification', 'Regression'],
'characteristics': ['Tabular'], 'num_instances': 4177, 'num_features':
8, 'feature_types': ['Categorical', 'Integer', 'Real'],
'demographics': [], 'target_col': ['Rings'], 'index_col': None,
'has_missing_values': 'no', 'missing_values_symbol': None,
'year_of_dataset_creation': 1994, 'last_updated': 'Mon Aug 28 2023',
'dataset_doi': '10.24432/C55C7W', 'creators': ['Warwick Nash', 'Tracy
Sellers', 'Simon Talbot', 'Andrew Cawthorn', 'Wes Ford'],
'intro_paper': None, 'additional_info': {'summary': 'Predicting the
age of abalone from physical measurements.  The age of abalone is
determined by cutting the shell through the cone, staining it, and
counting the number of rings through a microscope -- a boring and
time-consuming task.  Other measurements, which are easier to obtain,
are used to predict the age.  Further information, such as weather
patterns and location (hence food availability) may be required to
solve the problem.\r\n\r\nFrom the original data examples with missing
values were removed (the majority having the predicted value missing),
and the ranges of the continuous values have been scaled for use with
an ANN (by dividing by 200).', 'purpose': None, 'funded_by': None,
'instances_represent': None, 'recommended_data_splits': None,
'sensitive_data': None, 'preprocessing_description': None,
'variable_info': 'Given is the attribute name, attribute type, the
measurement unit and a brief description.  The number of rings is the
value to predict: either as a continuous value or as a classification
problem.\r\n\r\nName / Data Type / Measurement Unit / Description\r\
n-----------------------------\r\nSex / nominal / -- / M, F, and I
(infant)\r\nLength / continuous / mm / Longest shell measurement\r\
nDiameter\t/ continuous / mm / perpendicular to length\r\nHeight /
continuous / mm / with meat in shell\r\nWhole weight / continuous /
grams / whole abalone\r\nShucked weight / continuous\t / grams /
weight of meat\r\nViscera weight / continuous / grams / gut weight
(after bleeding)\r\nShell weight / continuous / grams / after being
dried\r\nRings / integer / -- / +1.5 gives the age in years\r\n\r\nThe
readme file contains attribute statistics.', 'citation': None}}
```


```
             name     role         type demographic  \
0             Sex  Feature  Categorical        None
1          Length  Feature   Continuous        None
2        Diameter  Feature   Continuous        None
3          Height  Feature   Continuous        None
4    Whole_weight  Feature   Continuous        None
5  Shucked_weight  Feature   Continuous        None
6  Viscera_weight  Feature   Continuous        None
7    Shell_weight  Feature   Continuous        None
```

```
8           Rings   Target      Integer        None

                       description  units missing_values
0          M, F, and I (infant)   None             no
1    Longest shell measurement     mm             no
2      perpendicular to length     mm             no
3           with meat in shell     mm             no
4                whole abalone  grams             no
5               weight of meat  grams             no
6  gut weight (after bleeding)  grams             no
7            after being dried  grams             no
8  +1.5 gives the age in years   None             no
```

```
df_combined.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.rename(columns = {1: "Length", 2: "Diameter", 3: "Height", 4:
"Whole_weight",
                     5: "Shucked_weight", 6: "Viscera_weight", 7:
"Shell_weight",
                     8: "Rings"}, inplace=True)

df.head()
```

```
   0  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
```

```
x = df[["Length",     "Diameter",      "Height",   "Whole_weight",
     "Shucked_weight",
        "Viscera_weight",   "Shell_weight"]]

y = df["Rings"]
```

Etapa 1:

1) Calcula con los datos originales el valor de R^2 y los parámetros.

2) Busca una puntos "leverage", influyentes y outliers. Determina que puntos perjudican y cuales benefician si se ignoran.

3) Realiza una transformación para lidiar con los outliers.

4) Una vez hecho esto, calcula de nuevo el valor de R^2 y los parámetros obtenidos, comparar.

Calculo de R^2 y parametros.

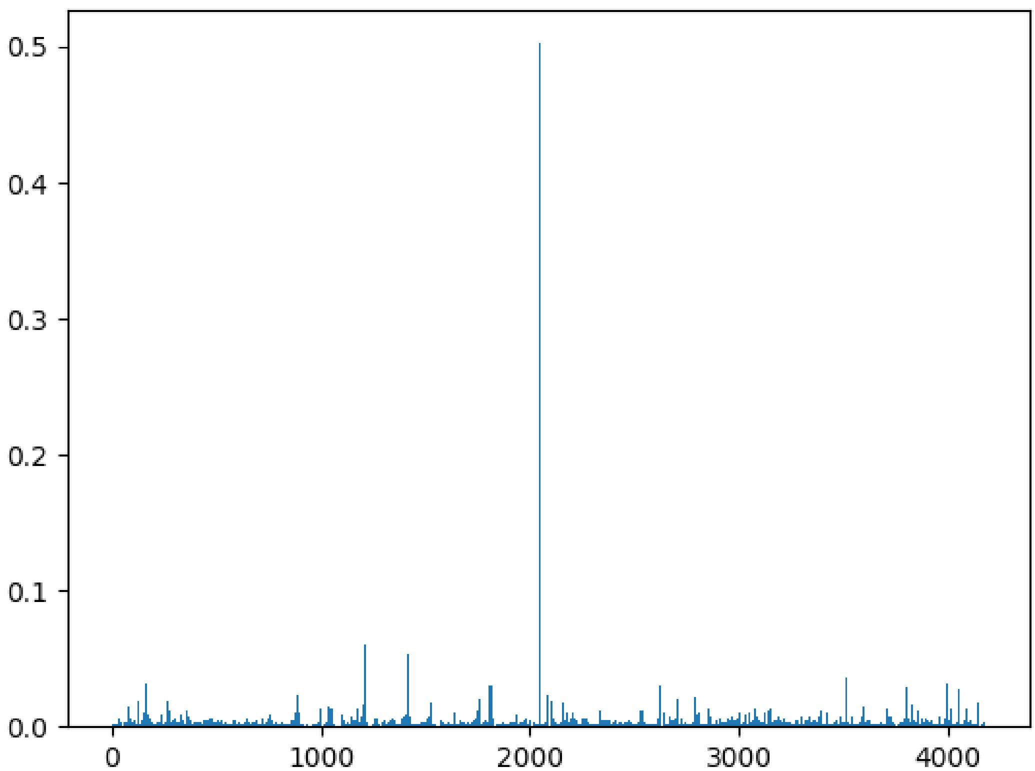
```
x_fit = sm.add_constant(x)

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

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:                        04:19:52   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.
```

```
influence = fitted_model.get_influence()

H_diag = influence.hat_matrix_diag
print(H_diag)

plt.bar(df.index, H_diag, width = 10)
plt.show()
```

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




```
mapping = sorted(list(enumerate(H_diag)), key = lambda item: item[1],
                 reverse = True)
```

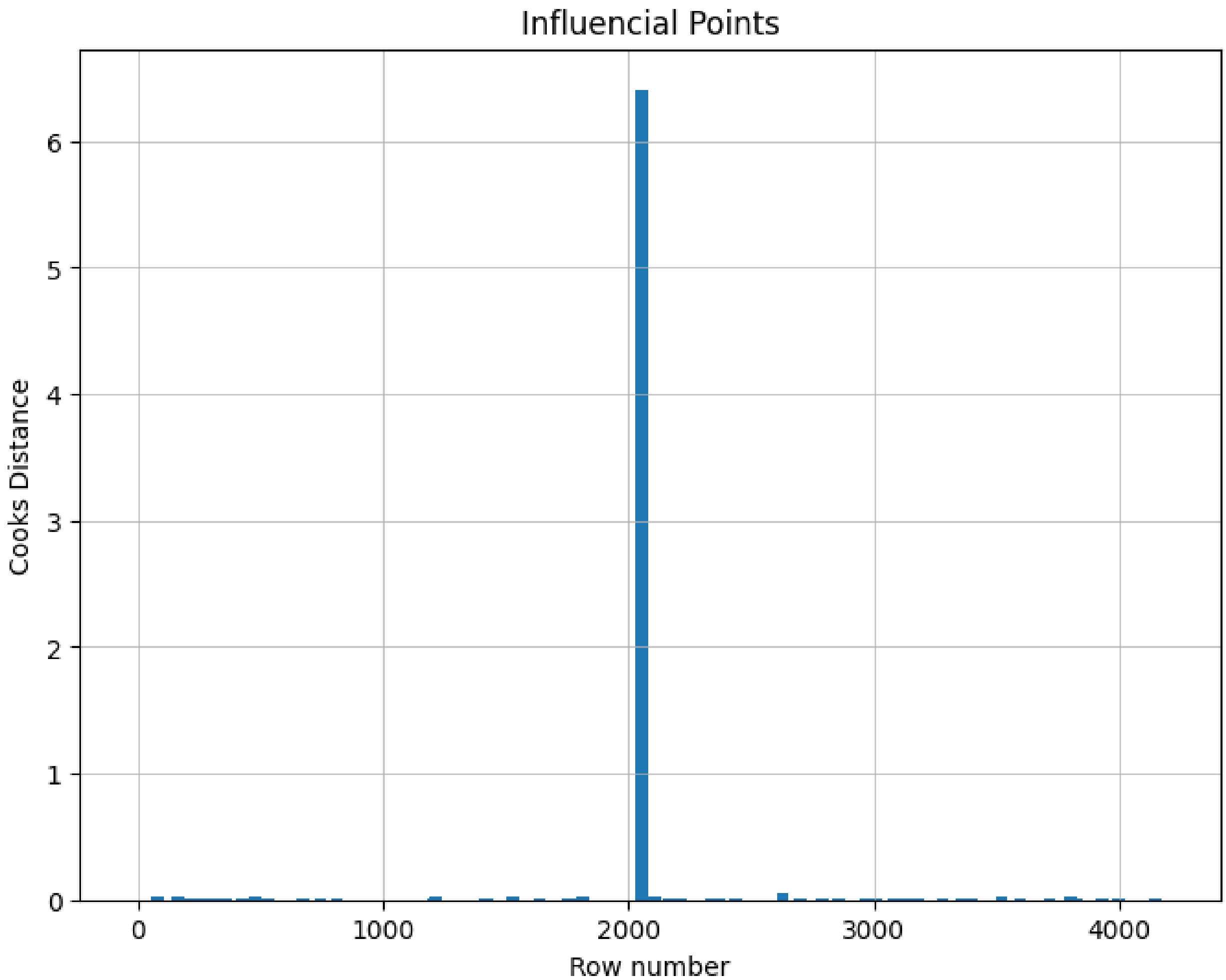
```
max_value_idxs = [item[0] for item in mapping]

print("Top leverage values:")
print([item[1] for item in mapping][:31])

print("n\Sample indexes with more leverage: ")
print(df.iloc[max_value_idxs])
```

```
Top leverage values:
[0.5019723528421322, 0.05960859244317343, 0.05295671927323318,
0.03534619653809162, 0.03225467110113976, 0.03137035532146494,
0.030569900198627913, 0.029743133376019915, 0.029187480387489946,
0.027582991982753567, 0.022819457579137194, 0.0225973123537083,
0.022390983449918696, 0.021865898884646136, 0.020667010650559633,
0.01972418742547942, 0.019168900827115705, 0.01889425115695011,
0.018850250145169004, 0.018223665076092552, 0.01803260007364776,
0.01799005010101374, 0.01783043300560813, 0.017060855237230048,
0.017033402896857498, 0.016166436763450736, 0.016084235349256088,
0.015863561592938696, 0.015035592735435064, 0.0145964475291037,
0.01434820167022206]
n\Sample indexes with more leverage:
      0  Length  Diameter  Height  Whole_weight  Shucked_weight  \
2051  F   0.455     0.355   1.130        0.5940          0.3320
1210  I   0.185     0.375   0.120        0.4645          0.1960
1417  M   0.705     0.565   0.515        2.2100          1.1075
3518  M   0.710     0.570   0.195        1.3480          0.8985
163   F   0.725     0.560   0.210        2.1410          0.6500
...  ..     ...       ...     ...           ...             ...
837   I   0.475     0.365   0.125        0.5465          0.2290
600   I   0.535     0.420   0.145        0.9260          0.3980
3555  M   0.535     0.415   0.135        0.7800          0.3165
2744  M   0.480     0.375   0.120        0.5895          0.2535
488   M   0.540     0.420   0.135        0.8075          0.3485

      Viscera_weight  Shell_weight  Rings
2051          0.1160        0.1335      8
1210          0.1045        0.1500      6
1417          0.4865        0.5120     10
3518          0.4435        0.4535     11
163           0.3980        1.0050     18
...              ...           ...    ...
837           0.1185        0.1720      9
600           0.1965        0.2500     17
3555          0.1690        0.2365      8
2744          0.1280        0.1720     11
488           0.1795        0.2350     11

[4177 rows x 9 columns]
```

```
#suppress scientific notation
np.set_printoptions(suppress = True)

#obtain Cook's distance for each observation
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]
```

```
plt.figure(figsize = (8, 6))
plt.bar(df.index, cooks_dist, width=50)
plt.xlabel("Row number")
plt.ylabel("Cooks Distance")
plt.title("Influencial Points")
plt.grid(linewidth = 0.5)
```




```
mapping = sorted(list(enumerate(cooks_dist)), key = lambda item:
item[1],
                 reverse = True)

max_value_idxs = [item[0] for item in mapping]

print('Top cook´s distance values: ')
print([item[1] for item in mapping][:3])

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

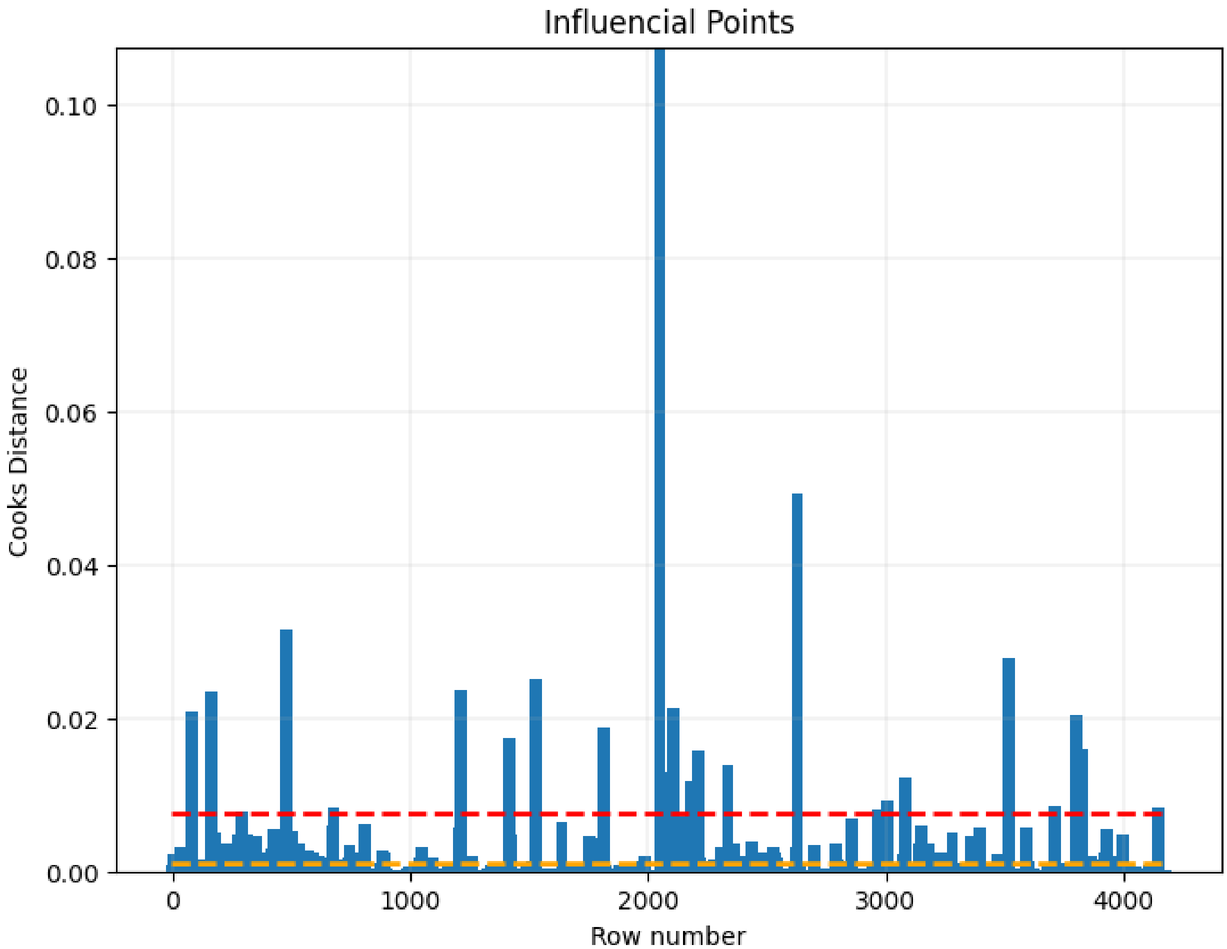
```
Top cook´s distance values:
[6.409299513058319, 0.04919146658760428, 0.031621158221939866]
Top sample indexes with more distance values:
      0  Length  Diameter  Height  Whole_weight  Shucked_weight  \
2051  F   0.455     0.355   1.130        0.5940          0.3320
2627  I   0.275     0.205   0.070        0.1055          0.4950
480   F   0.700     0.585   0.185        1.8075          0.7055
3518  M   0.710     0.570   0.195        1.3480          0.8985
```

```
1528  M   0.725     0.575   0.240        2.2100          1.3510
...  ..     ...       ...     ...           ...             ...
2522  M   0.545     0.450   0.150        0.8795          0.3870
2369  I   0.560     0.440   0.170        0.9445          0.3545
1272  I   0.475     0.355   0.100        0.5035          0.2535
1022  F   0.640     0.500   0.170        1.5175          0.6930
897   I   0.265     0.195   0.060        0.0920          0.0345

      Viscera_weight  Shell_weight  Rings
2051          0.1160        0.1335      8
2627          0.0190        0.0315      5
480           0.3215        0.4750     29
3518          0.4435        0.4535     11
1528          0.4130        0.5015     13
...              ...           ...    ...
2522          0.1500        0.2625     11
2369          0.2175        0.3000     12
1272          0.0910        0.1400      8
1022          0.3260        0.4090     11
897           0.0250        0.0245      6

[4177 rows x 9 columns]
```

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

```
0.0018730877579285728
```

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

plt.figure(figsize = (8 ,6))
plt.bar(df.index, cooks_dist, width=50)
plt.plot(df.index, mean_cooks_list, color="red", linestyle="--",
linewidth=2)
plt.plot (df.index, cooks_threshold, color="orange", linestyle="--",
          linewidth=2)

plt.xlabel ("Row number")
plt.ylabel("Cooks Distance")
plt.title("Influencial Points")
plt.ylim(top=max(mean_cooks_list + cooks_threshold) + 1e-1)
plt.grid(linewidth=0.2)
```




```
# Influencial points
influencial_points = df.index[cooks_dist > 4/len(cooks_dist)]
print(influencial_points)
df.iloc[influencial_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)
```

```
    0  Length  Diameter  Height  Whole_weight  Shucked_weight
Viscera_weight  \
6   F   0.530     0.415   0.150        0.7775          0.2370
0.1415
9   F   0.550     0.440   0.150        0.8945          0.3145
0.1510
32  M   0.665     0.525   0.165        1.3380          0.5515
0.3575
```

```
33  F   0.680     0.550   0.175         1.7980          0.8150
0.3925
36  F   0.540     0.475   0.155         1.2170          0.5305
0.3075
67  F   0.595     0.495   0.185         1.2850          0.4160
0.2240
72  F   0.595     0.475   0.170         1.2470          0.4800
0.2250
81  M   0.620     0.510   0.175         1.6150          0.5105
0.1920
83  M   0.595     0.475   0.160         1.3175          0.4080
0.2340
85  F   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
```

```
noninfluencial_point = df.index[cooks_dist < 4/len(cooks_dist)]
print(noninfluencial_point)
df.iloc[noninfluencial_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)

    0  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
```

```
5   I   0.425     0.300   0.095         0.3515          0.1410
0.0775
7   F   0.545     0.425   0.125         0.7680          0.2940
0.1495
8   M   0.475     0.370   0.125         0.5095          0.2165
0.1125
10  F   0.525     0.380   0.140         0.6065          0.1940
0.1475
11  M   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
```

Outliers usando Z-score

```
# Detect outliers using Z-score
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("\nNumber of outlier samples: ",
      x[(x > up_lim) |(x < dw_lim)].shape)
```

```
Upper limit:  Length            0.884271
Diameter          0.705601
Height            0.264998
Whole_weight      2.299909
Shucked_weight    1.025256
Viscera_weight    0.509436
Shell_weight      0.656439
dtype: float64

Lower limit:  Length            0.163713
Diameter          0.110162
Height            0.014035
Whole_weight     -0.642425
Shucked_weight   -0.306521
Viscera_weight   -0.148249
Shell_weight     -0.178777
dtype: float64
```

```
Number of outlier samples:  (4177, 7)
```

Outliers usando percentiles

```
# Detect Outliers using percentiles
Q1 = x.quantile(0.25)
Q3 = x.quantile (0.75)
iqr = Q3 - Q1

print("Q1: ", Q1)
print("\n Q3: ", Q3)
print("\n IQR:", iqr)

outliers_iqr = (x < Q1 - 1.5 * iqr) | (x > Q3 + 1.5 * iqr)
print("\n Number of Outlier Samples: ", x[outliers_iqr].shape)
```

```
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

 Number of Outlier Samples:  (4177, 7)
```

Manejo de valores atípicos

```
# Escalado MinMax
X = df[["Length",    "Diameter",     "Height",  "Whole_weight",
```

```
      "Shucked_weight",
        "Viscera_weight",   "Shell_weight"]]

y = df["Rings"]


scaler = MinMaxScaler(feature_range= (-1, 1))

# Compute the mean and std to be used for later scaling
scaler.fit(X, y)

# Per feature maximum
print('Max values: ', scaler.data_max_)

# Scale features of X according to feature range
print('\nTransformation Step: ')
x_transform = scaler.transform(X)
print(x_transform, '\n')
```

```
Max values:  [0.815  0.65   1.13   2.8255 1.488  0.76   1.005 ]

Transformation Step:
[[ 0.02702703  0.04201681 -0.83185841 ... -0.69939475 -0.73535221
  -0.70403587]
 [-0.25675676 -0.29411765 -0.84070796 ... -0.86751849 -0.87360105
  -0.86347783]
 [ 0.22972973  0.22689076 -0.76106195 ... -0.65635508 -0.62870309
  -0.58445441]
 ...
 [ 0.41891892  0.41176471 -0.63716814 ... -0.29455279 -0.24423963
  -0.38913802]
 [ 0.48648649  0.44537815 -0.73451327 ... -0.28715535 -0.31402238
  -0.41305431]
 [ 0.71621622  0.68067227 -0.65486726 ...  0.27034297 -0.00987492
  -0.01644245]]
```

Normalización Z-Score

```
scaler2 = StandardScaler()

# Compute the mean and std to be used for later scaling
scaler2.fit(X, y)

# The mean value for each feature in the training set
print('Means: ', scaler2.mean_)

# Perform standardization by centering and scaling
print('\nTransformation Step:')
x_transform2 = scaler2.transform(X)
print(x_transform2, '\n')
```

```
Means:  [ 1.          0.21349216  0.18615548 -0.75306832 -0.41438487 -
0.51799934
 -0.52575745 -0.5269938 ]

Transformation Step:
[[ 0.         -0.57455813 -0.43214879 ... -0.60768536 -0.72621157
  -0.63821689]
 [ 0.         -1.44898585 -1.439929   ... -1.17090984 -1.20522124
  -1.21298732]
 [ 0.          0.05003309  0.12213032 ... -0.4634999  -0.35668983
  -0.20713907]
 ...
 [ 0.          0.6329849   0.67640943 ...  0.74855917  0.97541324
   0.49695471]
 [ 0.          0.84118198  0.77718745 ...  0.77334105  0.73362741
   0.41073914]
 [ 0.          1.54905203  1.48263359 ...  2.64099341  1.78744868
   1.84048058]]


X = sm.add_constant(x_transform)

# Create the OLS model
model = sm.OLS(y, X)

# Fit the model
fitted_model = model.fit()

# Print the summary of the regression results
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:                        04:36:05   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         12.2796      0.717     17.117      0.000      10.873
13.686
x1            -0.5816      0.675     -0.861      0.389      -1.905
0.742
x2             3.9749      0.666      5.972      0.000       2.670
5.280
x3             6.6817      0.875      7.639      0.000       4.967
8.397
x4            13.0550      1.034     12.622      0.000      11.027
15.083
x5           -15.0290      0.612    -24.552      0.000     -16.229
-13.829
x6            -3.7328      0.495     -7.538      0.000      -4.704
-2.762
x7             4.3031      0.570      7.545      0.000       3.185
5.421
==============================================================================
========
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.
64.2
==============================================================================
========

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

X = sm.add_constant(x_transform2)

# Create the OLS model
model = sm.OLS(y, X)

# Fit the model
fitted_model = model.fit()
```

```python
# Print the summary of the regression results
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:                        04:38:08   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
x1         -2.421e-16   1.07e-16     -2.267      0.023   -4.51e-16    -
3.27e-17
x2            -0.1888      0.219     -0.861      0.389      -0.618
0.241
x3             1.3258      0.222      5.972      0.000       0.891
1.761
x4             0.4946      0.065      7.639      0.000       0.368
0.622
x5             4.5343      0.359     12.622      0.000       3.830
5.239
x6            -4.4862      0.183    -24.552      0.000      -4.844
-4.128
x7            -1.0773      0.143     -7.538      0.000      -1.358
-0.797
x8             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.
4.54e+16
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The smallest eigenvalue is 1.29e-29. This might indicate that
there are
strong multicollinearity problems or that the design matrix is
singular.
```

Una vez hecho esto, calcula de nuevo el valor de R^2 y los parámetros obtenidos, comparar.

Después de realizar el análisis y recalcular el valor de R^2 y los parámetros, observé que no hubo cambios sustanciales en los resultados. Los valores siguen siendo consistentes con los obtenidos anteriormente, lo que sugiere que las modificaciones realizadas no impactaron significativamente en el modelo.

Etapa 2:

1) Busca multicolinealidad en los datos usando VIF

2) Analiza y elimina variables independientes que indiquen que hay multicolinealidad

3) Calcular el valor de MSE.

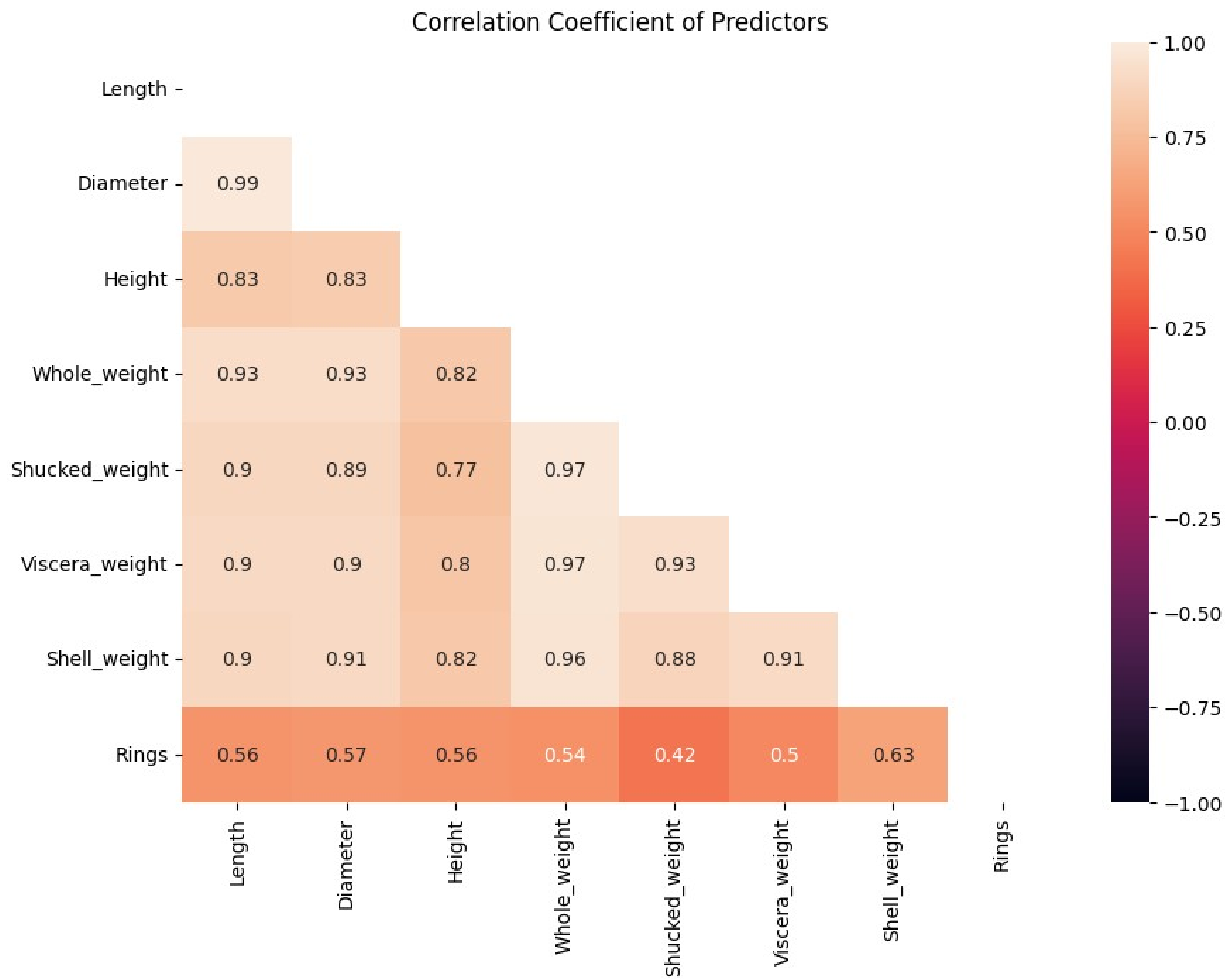
4) ¿Cómo cambio el valor de R^2 del modelo? ¿A que se lo adjudica?

5) ¿Como cambiaron los coeficientes? ¿Qué se interpretación se puede obtener con los nuevos valores de coeficientes?

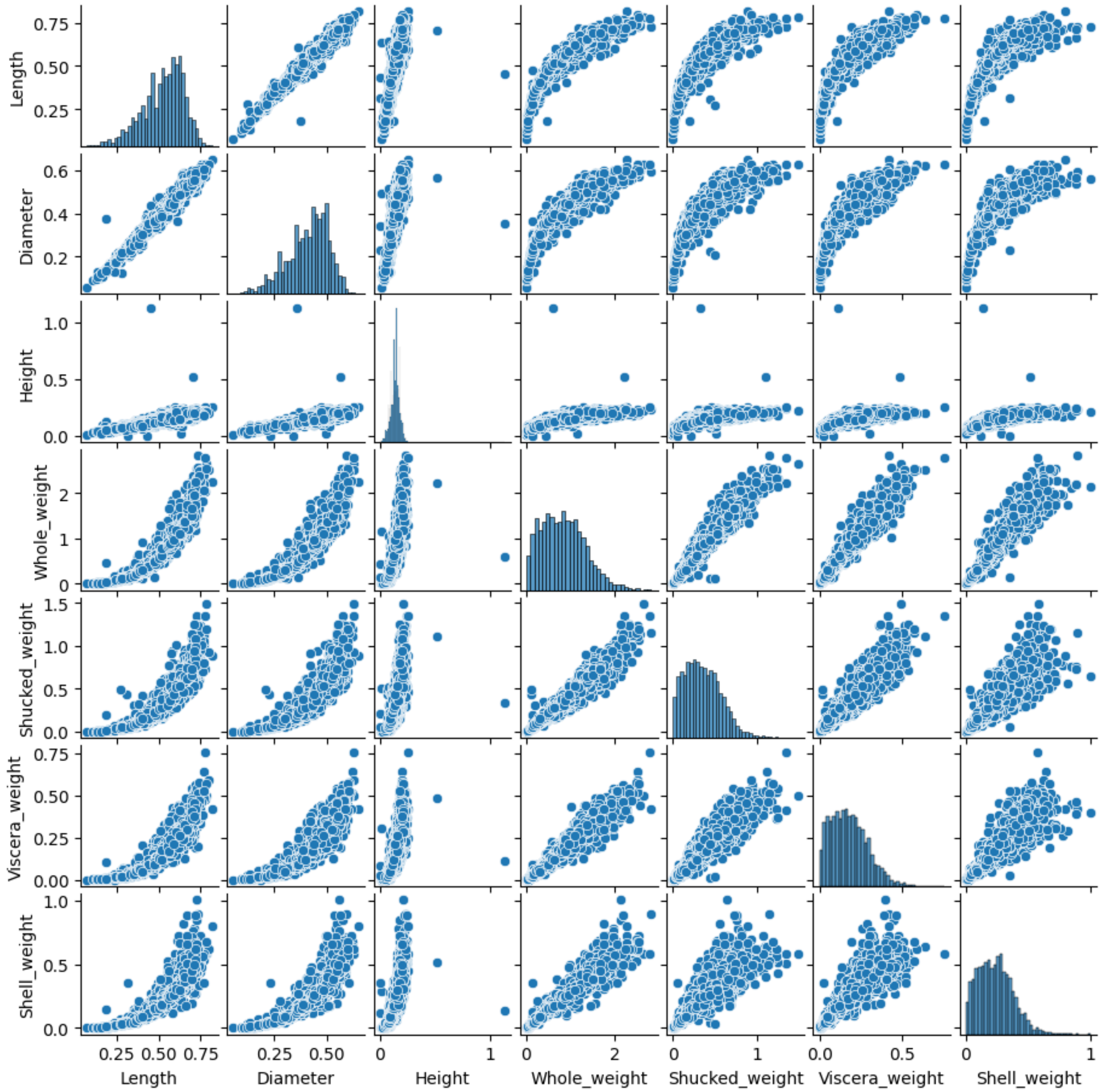
```python
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

plt.figure(figsize = (10, 7))

# Generate a mask to only show de bottom triangle
```

```
mask = np.triu(np.ones_like(df.corr(), dtype = bool))

# Generate Heatmap
sns.heatmap(df.corr(), annot = True, mask = mask, vmin = -1, vmax = 1)
plt.title('Correlation Coefficient of Predictors')
plt.show()
```

```
<ipython-input-59-337e745a60bb>:8: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it
will default to False. Select only valid columns or specify the value
of numeric_only to silence this warning.
  mask = np.triu(np.ones_like(df.corr(), dtype = bool))
<ipython-input-59-337e745a60bb>:11: FutureWarning: The default value
of numeric_only in DataFrame.corr is deprecated. In a future version,
it will default to False. Select only valid columns or specify the
value of numeric_only to silence this warning.
  sns.heatmap(df.corr(), annot = True, mask = mask, vmin = -1, vmax =
1)
```



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




```
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:                        04:59:22   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.
```

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

# Compute the VIF for all given features
def compute_vif(considered_features):
  # Add Independent Variable Set
  X = df[considered_features].copy()

  # The calculation of variance inflation requires a constant
  X['intercept'] = 1

  # VIF dataframe
  vif_data = pd.DataFrame()
  vif_data['feature'] = X.columns

  # Calculating VIF for each feature
  vif_data['VIF'] = [variance_inflation_factor(X.values, i)
                    for i in range(len(X.columns))]

  vif_data = vif_data[vif_data['feature'] != 'intercept']

  return vif_data

considered_features = ["Length", "Diameter",      "Height",
     "Whole_weight",  "Shucked_weight", "Viscera_weight",
     "Shell_weight"]

print(compute_vif(considered_features).sort_values('VIF', ascending =
False))
```

```
          feature         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
```

Tomando en cuenta que el valor "Whole_weight" es un valor atípica más nootorio dentro de Dataframe, debemos droppearlo del VIF para mejorar el análisis de datos

```
# Compute VIF values after removing a feature
considered_features.remove('Whole_weight')

print(compute_vif(considered_features).sort_values('VIF',
ascending=False))

          feature        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

print(considered_features)

['Length', 'Diameter', 'Height', 'Shucked_weight', 'Viscera_weight',
'Shell_weight']

X = df[considered_features]

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:                        04:59:57   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.
```

El valor de R_squared no mejoró lo suficiente como para se considerado algo relevante. Por lo que ahora vamos a analizar los datos sin las variables de "Shucked_wight" y "Viscera_weight", para posteriormente compararlos y determinar el mejor método.

```
# Compute VIF values after removing a feature
features_alt = [i for i in considered_features if i !=
'Shucked_weight']

print(compute_vif(features_alt).sort_values('VIF', ascending=False))

          feature        VIF
1        Diameter  41.818442
```

```
0          Length  40.140520
4    Shell_weight   7.736650
3  Viscera_weight   7.390557
2          Height   3.554536

X = df[features_alt]

model = sm.OLS(y, sm.add_constant(X))

fitted_model = model.fit()

print(fitted_model.summary())
```

```
                            OLS Regression Results

==============================================================================
========
Dep. Variable:                  Rings   R-squared:
0.438
Model:                            OLS   Adj. R-squared:
0.437
Method:                 Least Squares   F-statistic:
649.1
Date:                Sat, 21 Oct 2023   Prob (F-statistic):
0.00
Time:                        05:00:06   Log-Likelihood:
-9614.3
No. Observations:                4177   AIC:
1.924e+04
Df Residuals:                    4171   BIC:
1.928e+04
Df Model:                           5

Covariance Type:            nonrobust

==============================================================================
============
                     coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
------------
const              4.5894      0.283     16.220      0.000       4.035
5.144
Length            -7.5793      1.975     -3.837      0.000     -11.452
-3.707
Diameter          13.7447      2.440      5.634      0.000       8.962
18.528
Height            13.5193      1.688      8.011      0.000      10.211
16.828
Viscera_weight   -13.9921      0.929    -15.069      0.000     -15.812
```

```
-12.172
Shell_weight      18.2147      0.748     24.348      0.000      16.748
19.681
======================================================================
========
Omnibus:                     1107.451   Durbin-Watson:
1.139
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
3339.516
Skew:                           1.360   Prob(JB):
0.00
Kurtosis:                       6.434   Cond. No.
103.
======================================================================
========

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

```
# Compute VIF values after removing a feature
features_alt = [i for i in considered_features if i !=
'Viscera_weight']

print(compute_vif(features_alt).sort_values('VIF', ascending=False))
```

```
          feature        VIF
1        Diameter  41.791313
0          Length  40.320617
4    Shell_weight   6.930345
3  Shucked_weight   6.115478
2          Height   3.536331
```

```
X = df[features_alt]

model = sm.OLS(y, sm.add_constant(X))

fitted_model = model.fit()

print(fitted_model.summary())
```

```
                            OLS Regression Results

======================================================================
========
Dep. Variable:                  Rings   R-squared:
0.510
Model:                            OLS   Adj. R-squared:
0.509
Method:                 Least Squares   F-statistic:
866.7
```

```
Date:                    Sat, 21 Oct 2023   Prob (F-statistic):
0.00
Time:                            05:00:21   Log-Likelihood:
-9328.3
No. Observations:                    4177   AIC:
1.867e+04
Df Residuals:                        4171   BIC:
1.871e+04
Df Model:                               5

Covariance Type:                nonrobust

======================================================================
============
                     coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
------------
const              2.7936      0.268     10.427      0.000       2.268
3.319
Length            -1.8247      1.849     -0.987      0.324      -5.449
1.799
Diameter          14.0401      2.277      6.165      0.000       9.575
18.505
Height            12.2695      1.572      7.806      0.000       9.188
15.351
Shucked_weight   -11.5057      0.390    -29.539      0.000     -12.269
-10.742
Shell_weight      20.0665      0.661     30.350      0.000      18.770
21.363
======================================================================
========
Omnibus:                         1036.225   Durbin-Watson:
1.366
Prob(Omnibus):                      0.000   Jarque-Bera (JB):
3175.198
Skew:                               1.265   Prob(JB):
0.00
Kurtosis:                           6.442   Cond. No.
106.
======================================================================
========

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

# Calcular el valor de MSE
from sklearn.metrics import mean_squared_error
```

```
# Realizar predicciones con el modelo ajustado
predictions = fitted_model.predict()

mse = mean_squared_error(y, predictions)

print(f'MSE: {mse}')
```

```
MSE: 5.096983926334139
```

1) **¿Cómo cambio el valor de R^2 del modelo? ¿A que se lo adjudica?**

Los valores atípicos influyen en el valor de R cuadrada del modelos. Al analizar el modelo sin los valores atípicos de `Whole_weigh`, `Shucked_weight` y `Viscera_weight` respectivamente el valor de R cuadrada fue mejorando o empeorando. Esto se debe a que el peso que estos tienen sobre el modelo influyen significativamente, alterando el valor de R cuadrada.

2) **¿Como cambiaron los coeficientes? ¿Qué se interpretación se puede obtener con los nuevos valores de coeficientes?**

Los coeficientes de R cuadrada disminuyeron al ignorar los valores atípicos de `Shucked_weight` y `Viscera_weight`. El mejor valor de R cuadrada se sucedio al ignorar los datos de `Viscera_weight` con un valor de 5.10, por lo que podemos suponer que ese valor tiene un gran peso en el modelo. Sin embargo, cabe mencionar que para considera que el R cuadrada del modelo aún es muy bajo para ser ocnsiderado correcto o viable.

Etapa 3:

1) Analiza y determina el numero de componentes principales suficientes para mantener la cantidad de información justa necesaria.

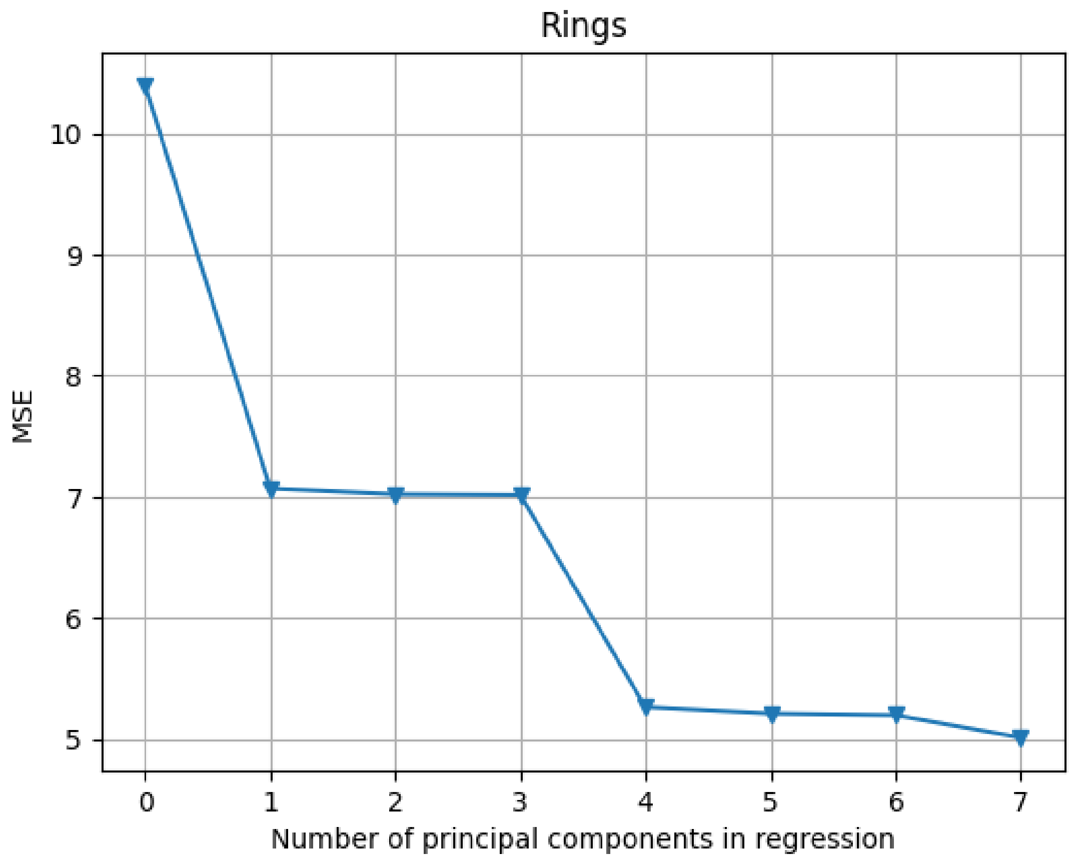
2) Obtenga de nuevo los valores de R^2 y MSE de esta aproximación.

3) ¿Mejoro el valor de R^2 y MSE del modelo PCR respecto al metodo de VIF?¿A que se lo adjudica?

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

pca = PCA()

x_reduced = pca.fit_transform(scale(x))

print("Returns 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())
```

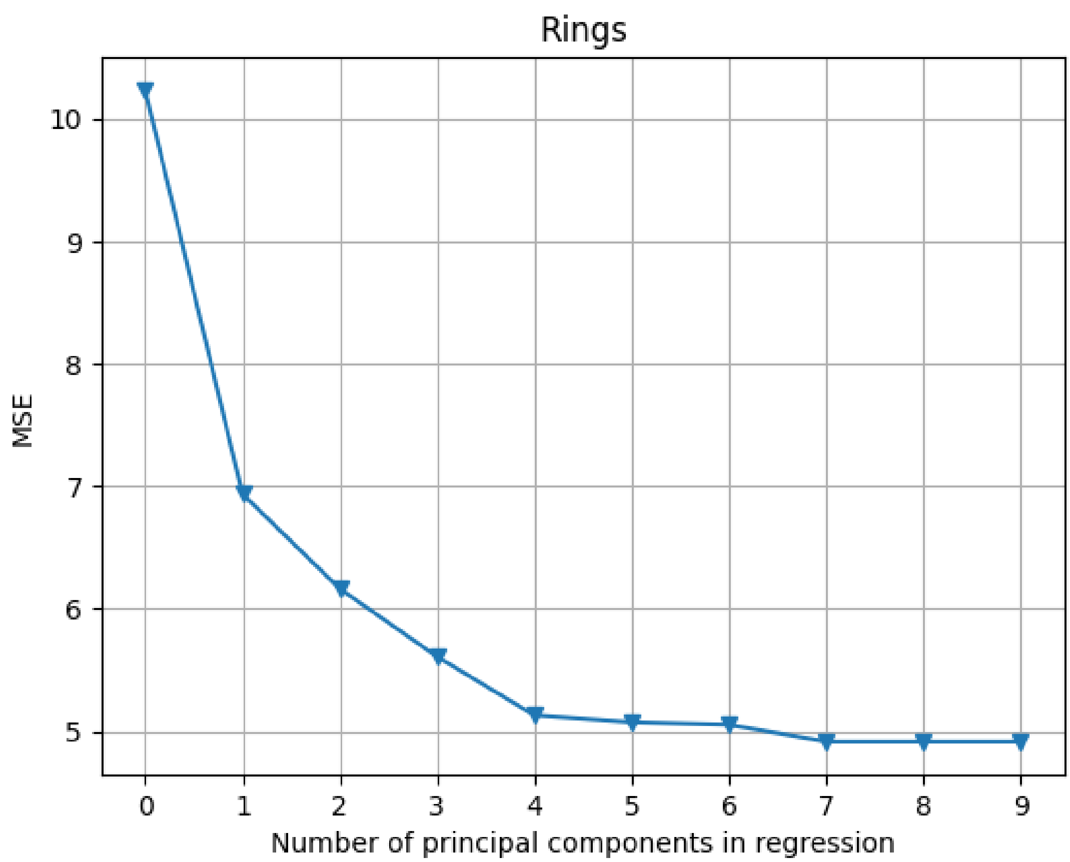
```
Returns a vector of the variance explained by each dimension.
[6.357 0.279 0.167 0.114 0.065 0.013 0.007]

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.   ]
```

```
n, pc = x_reduced.shape
kf_10 = model_selection.KFold(n_splits=10, shuffle=True,
random_state=1)
model = LinearRegression()
mse = []

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)

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))

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);
```




```
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.  ])

pca2 = PCA()

x_train, x_test, y_train, y_test = model_selection.train_test_split(x,
y,

test_size=0.5,

random_state=1)

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

kf_10 = model_selection.KFold(n_splits=10, shuffle=True,
random_state=1)
model = LinearRegression()
mse = []

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)

for i in np.arange(1, 10):
  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))

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);
```




```
x_reduced_test = pca2.transform(scale(x_test))[:,:5]
```

```python
model = sm.OLS(y_train, sm.add_constant(x_reduced_train[:,:5]))
fitted_model = model.fit()

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: 5.29
```

Mejoro el valor de R^2 y MSE del modelo PCR respecto al metodo de VIF?¿A que se lo adjudica?

Como podemos observar, el MSE no mostró una mejora significativa, destacando la necesidad de considerar otros factores. Por otro lado, el PCR demostró una mejora en R^2 al centrarse en componentes principales.