[1]
```python
from google.colab import drive

drive.mount("/content/gdrive")
```

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

[2]
```python
%cd "/content/gdrive/MyDrive/Bloque 2 IA/Estadistica"
!ls
```

```
/content/gdrive/MyDrive/Bloque 2 IA/Estadistica
 Apuntes.gdoc   Entregable  'Entregable 2'   us2022q2a.csv   usfirms2022.csv
```

[3]
```python
import plotly.express as px
import pandas as pd
import numpy as np
```

Seleccion y limpieza de datos

[4]
```python
df1 = pd.read_csv('us2022q2a.csv')
df2 = pd.read_csv('usfirms2022.csv')
```

Dropeamos columnas que son repetidas en el dataset y no nos son utiles para lo que deseamos

[5]
```python
df1 = df1.drop(['fiscalmonth', 'year', 'cto'], axis=1)
df2 = df2.drop(['N', 'Country\nof Origin', 'Type of Asset'], axis=1)
```

Mergeamos los datos de los dos datasets

```python
df_merge = df1.merge(df2, left_on='firm', right_on='Ticker')
df_merge.head()
```

	firm	q	revenue	cogs	sgae	otheropexp	extraincome	finexp	incometax	totalassets	...	originalprice	sharesoutstanding	Ticker	Name	Class	Sector NAICS\nlevel 1
0	A	2000q1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	104.0000	452000.000	A	Agilent Technologies, Inc	Com	Manufacturing
1	A	2000q2	2485000.0	1261000.0	1010000.0	0.0	42000.0	0.0	90000.0	7321000.0	...	73.7500	452271.967	A	Agilent Technologies, Inc	Com	Manufacturing
2	A	2000q3	2670000.0	1369000.0	1091000.0	0.0	28000.0	0.0	83000.0	7827000.0	...	48.9375	453014.579	A	Agilent Technologies, Inc	Com	Manufacturing
3	A	2000q4	3372000.0	1732000.0	1182000.0	0.0	10000.0	0.0	163000.0	8425000.0	...	54.7500	456366.381	A	Agilent Technologies, Inc	Com	Manufacturing

Dropeamos los servicios que no son industriales

```python
df2_services = df_merge.copy()
df2_services = df2_services.drop(df2_services[(df2_services['Sector NAICS\nlevel 1'] == 'Manufacturing')].index)
df2_services = df2_services.drop(df2_services[(df2_services['Sector NAICS\nlevel 1'] == 'Finance and Insurance')].index)
df2_services = df2_services.drop(df2_services[(df2_services['Sector NAICS\nlevel 1'] == 'Information')].index)
df2_services = df2_services.drop(df2_services[(df2_services['Sector NAICS\nlevel 1'] == 'Retail Trade')].index)
df2_services = df2_services.drop(df2_services[(df2_services['Sector NAICS\nlevel 1'] == 'Mining, Quarrying, and Oil and Gas Extraction')].index)
df2_services = df2_services.drop(df2_services[(df2_services['Sector NAICS\nlevel 1'] == 'Wholesale Trade')].index)
df2_services = df2_services.drop(df2_services[(df2_services['Sector NAICS\nlevel 1'] == 'Utilities')].index)
df2_services = df2_services.drop(df2_services[(df2_services['Sector NAICS\nlevel 1'] == 'Transportation and Warehousing')].index)
df2_services = df2_services.drop(df2_services[(df2_services['Sector NAICS\nlevel 1'] == 'Real Estate and Rental and Leasing')].index)
df2_services = df2_services.drop(df2_services[(df2_services['Sector NAICS\nlevel 1'] == 'Health Care and Social Assistance')].index)
df2_services = df2_services.drop(df2_services[(df2_services['Sector NAICS\nlevel 1'] == 'Construction')].index)
df2_services = df2_services.drop(df2_services[(df2_services['Sector NAICS\nlevel 1'] == 'Arts, Entertainment, and Recreation')].index)
df2_services = df2_services.drop(df2_services[(df2_services['Sector NAICS\nlevel 1'] == '-')].index)
df2_services = df2_services.drop(df2_services[(df2_services['Sector NAICS\nlevel 1'] == 'Agriculture, Forestry, Fishing and Hunting')].index)
```

Obtenemos las R de cada registro

[8]
```python
df2_services['R'] = np.log(df2_services.groupby(['firm'])['adjprice'].shift(-1)) - np.log(df2_services.groupby(['firm'])['adjprice'].shift(3))
```

Dropeamos todos los datos, a excepcion de el ultimo cuartil

[9]
```python
df_mask = df2_services['q'] == '2022q1'
df3 = df2_services[df_mask]
```

Dropeamos columnas que no se usaran en los calculos de nuestras variables independientes

[10]
```python
df3 = df3.drop(
    [
        'extraincome',
        'shortdebt',
        'longdebt',
        'stockholderequity',
        'Ticker',
        'Name',
        'Class',
        'Exchange / Src',
        'Sector\nEconomatica',
        'Sector NAICS\nlast available',
        'partind'
      ],
       axis=1)
```

```python
df3.head()
```

	firm	q	revenue	cogs	sgae	otheropexp	finexp	incometax	totalassets	totalliabilities	...	Sector NAICS\nlevel 1	R	Book	Market	EBIT	NetIncom
1525	ABM	2022q1	1936200.0	1659600.0	170600.0	0.0	6200.0	24300.0	4504900.0	2849600.0	...	Administrative and Support and Waste Managemen...	-0.008253	1655300.0	3.078651e+06	106000.0	75500.
1702	ABNB	2022q1	1508937.0	362623.0	918116.0	233329.0	1020.0	10706.0	17068442.0	12331026.0	...	Administrative and Support and Waste	-0.541818	4737416.0	1.088397e+08	-5131.0	-16857.

[12]
```python
df3['R'].isna().sum()
```

```
29
```

Revisamos las variables con las que obtendremos OPM, book_to_market y EPSP, eliminando los registros nulos y ceros para no entorpecer las operaciones

[13]
```python
df3['revenue'].isna().sum()
```

```
5
```

```python
df3['revenue'].describe()
```

```
count    3.720000e+02
mean     9.212180e+05
std      3.942143e+06
min      0.000000e+00
25%      6.649075e+04
50%      1.981825e+05
75%      7.438012e+05
max      6.801100e+07
Name: revenue, dtype: float64
```

[15]
```python
df_mask = df3['revenue'] > 0
df3 = df3[df_mask]
```

[16]
```python
df3['revenue'].isna().sum()
```

```
0
```

[17]
```python
df3['sharesoutstanding'].isna().sum()
```

```
1
```

[18]
```python
df3['sharesoutstanding'].describe()
```

```
count    3.670000e+02
mean     1.407183e+05
std      2.641917e+05
min      1.786296e+03
25%      2.958531e+04
50%      5.722365e+04
75%      1.400198e+05
max      2.721942e+06
Name: sharesoutstanding, dtype: float64
```

[19]
```python
df_mask = df3['sharesoutstanding'] >= 0
df3 = df3[df_mask]
```

[20]
```python
df3['sharesoutstanding'].isna().sum()
```

```
0
```

```python
df3['originalprice'].isna().sum()
```

```
1
```

[22]
```python
df3['originalprice'].describe()
```

```
count     366.000000
mean       75.408607
std       217.109625
min         0.260000
25%        10.522500
50%        29.585000
75%        77.442500
max      2781.350000
Name: originalprice, dtype: float64
```

```
df_mask = df3['originalprice'] >= 0
df3 = df3[df_mask]
```

+ Código + Texto

[24]
```
df3['originalprice'].isna().sum()
```

```
0
```

Calculamos Book to market ratio, epsp y opm a partir de los datos

[25]
```
df3['Book'] = df3['totalassets'] - df3['totalliabilities']
df3['Market'] = df3['originalprice'] * df3['sharesoutstanding']


df3['EBIT'] = df3['revenue'] - df3['cogs'] - df3['sgae'] - df3['otheropexp']
df3['NetIncome'] = df3['EBIT'] - df3['incometax'] - df3['finexp']
df3['EPS'] = df3['NetIncome'] / df3['sharesoutstanding']


df3['EPSP'] = df3['EPS'] / df3['originalprice']
df3['OPM'] = df3['EBIT'] / df3['revenue']
df3['Book_to_Market_ratio'] = df3['Book'] / df3['Market']
```

Dropeamos los nan en las variables de R, OPM, EPSP Y Book_to_Market_ratio

[26]
```
df3['R'].isna().sum()
```

```
23
```

[27]
```
df3['R'].describe()
```

```
count    343.000000
mean      -0.496691
std        0.638242
min       -3.482115
25%       -0.774205
50%       -0.320379
75%       -0.079993
max        1.591808
Name: R, dtype: float64
```

```
df_mask = df3['R'] >= -5
df3 = df3[df_mask]
```

[29]
```
df3['R'].isna().sum()
```

```
0
```

[30]
```
df3['EPSP'].isna().sum()
```

```
0
```

[31]
```
df3['Book_to_Market_ratio'].isna().sum()
```

```
0
```

[32]
```
df3['OPM'].isna().sum()
```

```
0
```

[33]
```
df_clean_3 = df3.copy()
```

Sorteamos los datos por mercado

[34]
```
df_clean_3 = df_clean_3.sort_values(by = 'Market')
```

[35]
```
df_clean_3 = df_clean_3.reset_index()
df_clean_3 = df_clean_3.drop(['index'], axis = 1)
```

[36]
```
df_clean_3 = df_clean_3.reset_index()
```

Asignamos las varuables dummys para obtener las grandes, medianas y grandes empresas

```python
[37]
df_clean_3["isSmall"] = df_clean_3.index <= len(df_clean_3) / 3
df_clean_3["isSmall"] = df_clean_3["isSmall"].astype(int)

df_clean_3["isMedium"] = (df_clean_3.index <= (2 * len(df_clean_3) / 3)) & (df_clean_3.index > (len(df_clean_3) / 3))
df_clean_3["isMedium"] = df_clean_3["isMedium"].astype(int)
```

Guardamos las variables que usaremos para el calculo matricial

```python
[38]
Variables = df_clean_3[['EPSP', 'OPM', 'Book_to_Market_ratio', 'isSmall', 'isMedium', 'R']]
Variables_mat = Variables.cov().to_numpy()
```

Dividimos las variables en 'X' y 'y'

```python
[39]
X = df_clean_3[['EPSP', 'OPM', 'Book_to_Market_ratio', 'isSmall', 'isMedium']]
y = df_clean_3['R']
```

Matriz de varianza y covarianza

Seguimos las formula del documento https://docs.google.com/viewer?a=v&pid=sites&srcid=ZWdhZGVyemMubmV0fGZ6MzAzMHxneDozOTI1ODBmMGU2YTMzZGM1

Para obtener la matriz de varianza y covarianza

```python
[40] X_varcov_mat = X.to_numpy()
```

```python
[41] X_varcov_trans = np.transpose(X_varcov_mat)
[42] aux1 = np.dot(X_varcov_trans, X_varcov_mat)
[43] unos = np.zeros((X_varcov_trans.shape[1], 1))
[44] unos = 1 + unos
[45] aux2 = np.dot(X_varcov_trans, unos)
[46] aux3 = np.transpose(aux2)
     aux4 = np.dot(aux2, aux3)
```

\+ Código + Texto

```python
[48] aux5 = aux4 / X_varcov_mat.shape[0]
[49] aux6 = aux1 - aux5
[50] mat_varcov = aux6 / (X_varcov_mat.shape[0] - 1)
[51] mat_varcov
```

```
array([[ 8.10117893e-03,  9.44989090e-02, -5.59550077e-03,
        -1.22051289e-02,  5.27735695e-03],
       [ 9.44989090e-02,  2.66685817e+02,  3.75584007e-02,
         8.71823279e-02, -5.58580184e-01],
       [-5.59550077e-03,  3.75584007e-02,  2.60931238e-01,
         8.65885609e-02, -2.45076743e-02],
       [-1.22051289e-02,  8.71823279e-02,  8.65885609e-02,
         2.23517979e-01, -1.11758989e-01],
       [ 5.27735695e-03, -5.58580184e-01, -2.45076743e-02,
        -1.11758989e-01,  2.22546161e-01]])
```

```python
[52] mat = pd.DataFrame(mat_varcov)
[53] mat
```

	0	1	2	3	4
0	0.008101	0.094499	-0.005596	-0.012205	0.005277
1	0.094499	266.685817	0.037558	0.087182	-0.558580
2	-0.005596	0.037558	0.260931	0.086589	-0.024508
3	-0.012205	0.087182	0.086589	0.223518	-0.111759
4	0.005277	-0.558580	-0.024508	-0.111759	0.222546

Comprobacion de la matriz de varianza y covarianza

```python
[54] X.cov()
```

	EPSP	OPM	Book_to_Market_ratio	isSmall	isMedium
EPSP	0.008101	0.094499	-0.005596	-0.012205	0.005277
OPM	0.094499	266.685817	0.037558	0.087182	-0.558580
Book_to_Market_ratio	-0.005596	0.037558	0.260931	0.086589	-0.024508
isSmall	-0.012205	0.087182	0.086589	0.223518	-0.111759
isMedium	0.005277	-0.558580	-0.024508	-0.111759	0.222546

Comprobamos que las varianzas son iguales en nuestro calculo y lo obtenido por medio de pandas, por lo que la matriz es correcta

Matriz de correlaciones

Seguimos la formula de correlacion obtenida a base de los standar deviations y la matriz de varianza y covarianza

```
var = np.diagonal(mat_varcov)
```

[56]
```
stdr_dev = np.sqrt(var)
```

[57]
```
stdr_dev
```

```
array([ 0.09000655, 16.33051797,  0.51081429,  0.47277688,  0.47174798])
```

[58]
```
ya_no_se = [[ 0 for y in range( len(stdr_dev) ) ] for x in range( len(stdr_dev) )]
for i in range(len(stdr_dev)):
  for j in range(len(stdr_dev)):
    ya_no_se[i][j] = stdr_dev[i] * stdr_dev[j]
```

[59]
```
corr = [[ 0 for y in range( len(ya_no_se) ) ] for x in range( len(ya_no_se) )]
for i in range(len(ya_no_se)):
  for j in range(len(ya_no_se)):
    corr[i][j] = mat_varcov[i][j] / ya_no_se[i][j]
```

[108]
```
print(corr)
```

```
[[1.0, 0.06429137624010672, -0.12170314747329583, -0.28682171558280145, 0.1242888885114298], [0.06429137624010672, 1.0000000000000002, 0.004502400179255902, 0.01129203642553, -0.07250€
```

[61]
```
X.corr()
```

	EPSP	OPM	Book_to_Market_ratio	isSmall	isMedium
EPSP	1.000000	0.064291	-0.121703	-0.286822	0.124289
OPM	0.064291	1.000000	0.004502	0.011292	-0.072506
Book_to_Market_ratio	-0.121703	0.004502	1.000000	0.358543	-0.101702
isSmall	-0.286822	0.011292	0.358543	1.000000	-0.501091
isMedium	0.124289	-0.072506	-0.101702	-0.501091	1.000000

Comprobamos que las correlaciones son iguales en nuestro calculo y lo obtenido por medio de pandas, por lo que la matriz es correcta

Interpretacion de las correlaciones

Segun lo obtenido en la matriz de correlacion, podemos ver que las variables con mayor relacion entre ellas son la de isSmall con EPSP de manera negativa, isSmall con Book to Market de forma positiva e isMedium con isSmall de forma negativa.

Modelo 1

Betas

Seguimos la siguiente formula para obtener las betas

Primer valor de x son 1

x´ es x transpuesta

x^1 es x inversa

(x´x)^-1x´y

[62]
```
X['1'] = 1
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
[63] X = X[['1', 'isSmall', 'isMedium', 'EPSP', 'OPM', 'Book_to_Market_ratio']]
```

```
[64] X_mat = X.to_numpy()
```

```
[65] X_t = np.transpose(X_mat)
```

```
[66] X_sqr = np.dot(X_t, X_mat)
```

```
[67] X_inv = np.linalg.inv(X_sqr)
```

```
[68] aux = np.dot(X_inv, X_t)
```

Hat matrix

Para obtener la hat matrix, seguiremos la formula de:

H = X(X´X)^-1X´

```
hat_mat = np.dot(X_mat, aux)
```

```
print(hat_mat)
```

```
array([[ 7.99814542e-02,  3.69566482e-02,  3.64933116e-02, ...,
        -3.12175062e-03,  3.86339056e-04, -1.53510011e-03],
       [ 3.69566482e-02,  2.11714623e-02,  2.10072578e-02, ...,
        -1.23457873e-03,  1.55939187e-05, -6.59032828e-04],
       [ 3.64933116e-02,  2.10072578e-02,  2.08512576e-02, ...,
        -1.21840798e-03,  9.30848046e-06, -6.51292207e-04],
       ...,
       [-3.12175062e-03, -1.23457873e-03, -1.21840798e-03, ...,
         8.91219115e-03,  8.75610265e-03,  8.84028130e-03],
       [ 3.86339056e-04,  1.55939187e-05,  9.30848046e-06, ...,
         8.75610265e-03,  8.78947148e-03,  8.76956564e-03],
       [-1.53510011e-03, -6.59032828e-04, -6.51292207e-04, ...,
         8.84028130e-03,  8.76956564e-03,  8.80742646e-03]])
```

```
[71] hat_mat_pd = pd.DataFrame(hat_mat)
```

```
hat_mat_pd
```

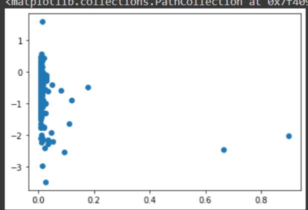
	0	1	2	3	4	5	6	7	8	9	...	333	334	335	336	337	338	339	
0	0.079981	0.036957	0.036493	0.005703	0.012052	0.029782	0.024680	-0.006519	0.015075	0.019901	...	-0.004172	0.006343	-0.008143	-0.004171	-0.000980	-0.006656	-0.005036	-0.0
1	0.036957	0.021171	0.021007	0.009232	0.008866	0.016961	0.018605	0.001757	0.009248	0.015641	...	-0.001498	0.002238	-0.003076	-0.001435	-0.000336	-0.002563	-0.001930	-0.0
2	0.036493	0.021007	0.020851	0.009279	0.008815	0.016819	0.018585	0.001829	0.009162	0.015634	...	-0.001467	0.002187	-0.003021	-0.001403	-0.000328	-0.002521	-0.001900	-0.0
3	0.005703	0.009232	0.009279	0.011155	0.006981	0.007682	0.012869	0.008103	0.005748	0.011620	...	0.000387	-0.000642	0.000548	0.000472	0.000113	0.000381	0.000301	0.0
4	0.012052	0.008866	0.008815	0.006981	0.009939	0.009782	0.006122	0.008836	0.010833	0.006876	...	-0.000341	0.000563	-0.000522	-0.000399	-0.000095	-0.000377	-0.000290	-0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
338	-0.006656	-0.002563	-0.002521	0.000381	-0.000377	-0.001973	-0.001294	0.001372	-0.000706	-0.000911	...	0.009170	0.008166	0.009541	0.009174	0.008866	0.009399	0.009248	0.0
339	-0.005036	-0.001930	-0.001900	0.000301	-0.000290	-0.001492	-0.000962	0.001036	-0.000544	-0.000680	...	0.009074	0.008313	0.009355	0.009077	0.008844	0.009248	0.009134	0.0

Para conocer los leverage points de los datos, obtenemos la diagonal de nuestra Hat matrix

```
[73] lp = np.diag(hat_mat_pd)
```

```
import matplotlib.pyplot as plt
plt.scatter(lp, y)
```

<matplotlib.collections.PathCollection at 0x7f409159a390>



\+ Código + Texto

```
[75] lp_df = pd.DataFrame(lp, columns = ['leverage_points'])
```

```
[76] lp_df.describe()
```

	leverage_points
count	343.000000
mean	0.017493
std	0.060969
min	0.008723
25%	0.009009
50%	0.009527
75%	0.011539
max	0.897466

Una vez obtenidos los leverage points, podemos ver en la grafica, que tenemos algunos con valores altos,(las que se encuentran por encima de 0.6), de igual forma, podemos ver que estos son leverage points altos, ya que la media de los datos es de 0.17, por lo que se encuentran muy separados.

De igual forma, podemos seguir la formula *(3k+3)/n*, con la cual obtenemos 3 veces el valor de la media de los leverage points, y los valores por encima de estos se consideran altos.

Datos a tomar:

- k = number of predictors (5)
- n = number of observations (343)

((3*5)+3)/343 = 0.0524781341107 => cualquier número por encima de este es un high leverage.

Dropearemos todos los datos por encima de esto.

[77]
```
mask = lp_df['leverage_points'] <= 0.0524781341107
lp_df = lp_df[mask]
```

```
lp_df.describe()
```

	leverage_points
count	336.000000
mean	0.011478
std	0.005558
min	0.008723
25%	0.009003
50%	0.009512
75%	0.011380
max	0.050592

Podemos ver que se eliminaron 7 registros, y una vez eliminados estos datos, vemos que ya no hay datos por encima del valor dicho

Finalmente obtenemos los valores de las Betas con outliers del primer modelo

[79]
```
B = np.dot(aux, y)
```

[80]
```
print(B)
```

```
[-0.32709095 -0.19304263 -0.10909288  3.15241522  0.00727031 -0.03304624]
```

MSE

```
y_pred = X * B
```

[82]
```
y_pred['sum'] = y_pred['1'] + y_pred['isSmall'] + y_pred['isMedium'] + y_pred['EPSP'] + y_pred['OPM'] + y_pred['Book_to_Market_ratio']
```

[83]
```
E = y - y_pred['sum']
```

[84]
```
aux = E * E
```

[85]
```
print(aux)
```

```
0      0.151943
1      0.619076
2      0.088342
3      1.662498
4      0.089718
         ...
338    0.147653
339    0.025062
340    0.017362
```

[86]
```
MSE = aux.sum()/len(aux)
```

[87]
```
print(MSE)
```

```
0.28707952963553085
```

Multicolinealidad

Revisamos la multicolinearidad de las variables usando la funcion de VIF

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
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))]

print(vif_data)
```

```
                feature       VIF
0                     1  3.169416
1               isSmall  1.623790
2              isMedium  1.356360
3                  EPSP  1.095860
4                   OPM  1.011020
5  Book_to_Market_ratio  1.158820
```

Como podemos ver, al tener un vif de menos de 1.5 en todas las variables, no existe multicolienaridad en los datos, por lo que no se necesita generar cambios en estos

Modelo base

[89]
```
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

```
mod = sm.OLS(y, sm.add_constant(X)).fit()

print(mod.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      R   R-squared:                       0.293
Model:                            OLS   Adj. R-squared:                  0.283
Method:                 Least Squares   F-statistic:                     27.96
Date:                Sat, 26 Nov 2022   Prob (F-statistic):           1.09e-23
Time:                        00:39:10   Log-Likelihood:                -272.66
No. Observations:                 343   AIC:                             557.3
Df Residuals:                     337   BIC:                             580.4
Df Model:                           5
Covariance Type:            nonrobust
========================================================================================
                           coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------
1                       -0.3271      0.052     -6.295      0.000      -0.429      -0.225
isSmall                 -0.1930      0.079     -2.450      0.015      -0.348      -0.038
isMedium                -0.1091      0.072     -1.512      0.132      -0.251       0.033
EPSP                     3.1524      0.340      9.273      0.000       2.484       3.821
OPM                      0.0073      0.002      4.040      0.000       0.004       0.011
Book_to_Market_ratio    -0.0330      0.062     -0.536      0.592      -0.154       0.088
==============================================================================
Omnibus:                       35.305   Durbin-Watson:                   2.043
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               82.739
Skew:                          -0.514   Prob(JB):                     1.08e-18
Kurtosis:                       5.175   Cond. No.                         191.
==============================================================================
```

Leverage Points y Outliers

```
influence = mod.get_influence()
inf_sum = influence.summary_frame()

print(inf_sum.head())

student_resid = influence.resid_studentized_external
(cooks, p) = influence.cooks_distance
(dffits, p) = influence.dffits
leverage = influence.hat_matrix_diag
```

```
      dfb_1  dfb_isSmall  dfb_isMedium  dfb_EPSP   dfb_OPM  \
0 -0.044687    -0.031668     -0.020327 -0.027360  0.002654
1  0.029940    -0.019770      0.015261  0.073203 -0.008763
2  0.011037    -0.007701      0.005749  0.027817 -0.002096
3 -0.014111    -0.133038     -0.002716  0.112631 -0.013666
4 -0.002805     0.034266     -0.000482  0.017255  0.000608

   dfb_Book_to_Market_ratio   cooks_d  standard_resid  hat_diag  \
0                  0.206752  0.008190        0.751811  0.079981
1                 -0.147880  0.007803       -1.471246  0.021171
2                 -0.054685  0.001096       -0.555683  0.020851
3                  0.044585  0.010818       -2.398740  0.011155
4                  0.009680  0.000519        0.556899  0.009939

   dffits_internal  student_resid    dffits
0         0.221669       0.751325  0.221526
1        -0.216375      -1.473802 -0.216751
2        -0.081090      -0.555112 -0.081007
3        -0.254769      -2.415891 -0.256591
4         0.055797       0.556328  0.055740
```

[92]

```python
# Concatenación con la variable dependiente y (R)
from statsmodels.formula.api import ols

ym = pd.DataFrame(y)
y_dep = pd.concat([ym.R, inf_sum], axis = 1)
y_dep =y_dep.rename(columns={'hat_diag': 'leverage'})
y_dep.head()
```

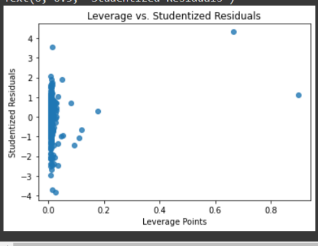
	R	dfb_1	dfb_isSmall	dfb_isMedium	dfb_EPSP	dfb_OPM	dfb_Book_to_Market_ratio	cooks_d	standard_resid	leverage	dffits_internal	student_resid	dffits
0	-0.589921	-0.044687	-0.031668	-0.020327	-0.027360	0.002654	0.206752	0.008190	0.751811	0.079981	0.221669	0.751325	0.221526
1	-1.773474	0.029940	-0.019770	0.015261	0.073203	-0.008763	-0.147880	0.007803	-1.471246	0.021171	-0.216375	-1.473802	-0.216751
2	-1.290984	0.011037	-0.007701	0.005749	0.027817	-0.002096	-0.054685	0.001096	-0.555683	0.020851	-0.081090	-0.555112	-0.081007
3	-2.218991	-0.014111	-0.133038	-0.002716	0.112631	-0.013666	0.044585	0.010818	-2.398740	0.011155	-0.254769	-2.415891	-0.256591
4	-0.253549	-0.002805	0.034266	-0.000482	0.017255	0.000608	0.009680	0.000519	0.556899	0.009939	0.055797	0.556328	0.055740

En esta tabla, podemos ver el student residual, los cooks distances, los drifts y los leverage points de nuestros datos

```python
import seaborn as sns

print ('Leverage vs. Studentized Residuals')
sns.regplot(leverage, mod.resid_pearson,  fit_reg=False)
plt.title('Leverage vs. Studentized Residuals')
plt.xlabel('Leverage Points')
plt.ylabel('Studentized Residuals')
```

```
Leverage vs. Studentized Residuals
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argume
  FutureWarning
Text(0, 0.5, 'Studentized Residuals')
```



Podemos comprobar que los leverage points obtenidos con algebra matricial son los mismos que los obtenidos con las funciones de python

Studentized residuals y High Leverage Points

Al igual que en los leverage points calculados con algebra matricial, se tomaran como high leverage points los que se sean mayores a tres veces el valor absoluto

[94]

```python
r = y_dep.student_resid
print (' student residual ')
print (r.describe())
```

```
 student residual 
count    343.000000
mean       0.016685
std        1.087878
min       -3.966665
25%       -0.510928
50%        0.163034
75%        0.621224
max        8.179974
Name: student_resid, dtype: float64
```

Podemos ver que existen valores mayores al valor absoluto de 3, por lo que consideraremos estos como outiers

[95]

```python
print(y_dep.student_resid[abs(r) > 3])
```

```
24     8.179974
41     3.650426
53    -3.807500
77    -3.966665
170   -3.027921
190    3.587238
Name: student_resid, dtype: float64
```

```
[96]
# High leverage
# point with leverage = (3k+3)/n
# k = number of predictors (5)
# n = number of observations (343)
#((3*3)+2)/343 = 0.0524781341107 => cualquier número por encima de este es un high leverage
l = y_dep.leverage

print (l.describe())
```

```
count    343.000000
mean       0.017493
std        0.060969
min        0.008723
25%        0.009009
50%        0.009527
75%        0.011539
max        0.897466
Name: leverage, dtype: float64
```

Podemos ver que existen valores por encima del marcado (0.0524), que son los high leverage point

```
[97]
# Leverage Point = ((3*3)+2)/343 = 0.0524781341107
print(y_dep.leverage[abs(l) > ((3*5)+3)/343])
```

```
0      0.079981
10     0.092529
23     0.177603
24     0.665677
64     0.110614
143    0.119640
190    0.897466
Name: leverage, dtype: float64
```

Con este filtro, podemos ver que hay 7 high leverage points en nuestros datos

Debido a esto, ahora sabemos que hay datos con gran influenci, ya que tenemos tanto outliers como high leverage, lo que puede afectar a los coeficientes

```
[98]
# Valores que son outliers y high leverage
outlier = pd.DataFrame((y_dep.R[abs(r) > 3]))
leverage= pd.DataFrame((y_dep.R[abs(l) > ((3*5)+3)/343]))

Influential=pd.merge(outlier,leverage, left_index=True, right_index=True)
print(Influential)
```

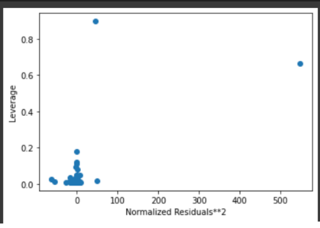
```
          R_x       R_y
24  -2.455893 -2.455893
190 -2.033554 -2.033554
```

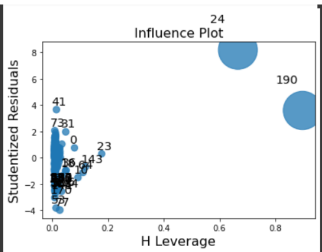
Estos serian los valores mas influentes, ya que son tanto high leevrage como outliers

```
from statsmodels.graphics.regressionplots import *

plt.scatter(y_dep.student_resid ** 3, y_dep.leverage)
plt.xlabel("Normalized Residuals**2")
plt.ylabel("Leverage")
plt.show()

influence_plot(mod)
plt.show()
```





Revisamos con los registros anteriores los cuales son:

- 0.665677 Y 0.897466 DE LEVERAGE RESPECTIVAMENTE
- 8.179974 Y 3.587238 DE STUDENTIZED RESIDUAL

Podemos decir aahora que estos dos son los datos con mayor influencia en el modelo.

Cooks distance

En este caso, igual tomaremos el triple de la media paa obtener los outlierspor medio del metodo de cooks

```
[100] limit = (y_dep.loc[:,"cooks_d"].mean())*3
outlier2 = pd.DataFrame((y_dep.R[abs(y_dep.cooks_d) > limit]))
print(outlier2)
```

```
            R
24  -2.455893
190 -2.033554
```

Podemos ver que al igual que en los otros metodos, nuestros outliers son los mismos

```
[100]
```

Modelo 2

Al ya conocer los outliers, high leverage y valores influenciables denro de nuestros datos, podemos mejorar el modelo descartando estos

```
X_m2 = X.copy()
Y_m2 = ym.copy()

X_m2 = X_m2.drop([24, 41, 53, 77, 170, 190])
Y_m2 = Y_m2.drop([24, 41, 53, 77, 170, 190])
```

Obtencion de las bbetas con algebra matricial

```
[102] X = X_m2[['1', 'isSmall', 'isMedium', 'EPSP', 'OPM', 'Book_to_Market_ratio']]
X_mat = X.to_numpy()
X_t = np.transpose(X_mat)
X_sqr = np.dot(X_t, X_mat)
X_inv = np.linalg.inv(X_sqr)
aux = np.dot(X_inv, X_t)
B = np.dot(aux, Y_m2)
print(B)
```

```
[[-0.33855748]
 [-0.06022982]
 [-0.07929252]
 [ 5.67505071]
 [ 0.01853454]
 [-0.05935155]]
```

```
model2 = sm.OLS(Y_m2, sm.add_constant(X_m2)).fit()
print(model2.summary())
```

```
model2 = sm.OLS(Y_m2, sm.add_constant(X_m2)).fit()
print(model2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      R   R-squared:                       0.363
Model:                            OLS   Adj. R-squared:                  0.353
Method:                 Least Squares   F-statistic:                     37.72
Date:                Sat, 26 Nov 2022   Prob (F-statistic):           1.40e-30
Time:                        00:39:11   Log-Likelihood:                -215.01
No. Observations:                 337   AIC:                             442.0
Df Residuals:                     331   BIC:                             464.9
Df Model:                           5
Covariance Type:            nonrobust
========================================================================================
                           coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------
1                       -0.3386      0.044     -7.615      0.000      -0.426      -0.251
isSmall                 -0.0602      0.070     -0.858      0.392      -0.198       0.078
isMedium                -0.0793      0.062     -1.281      0.201      -0.201       0.042
EPSP                     5.6751      0.527     10.760      0.000       4.638       6.713
OPM                      0.0185      0.005      3.797      0.000       0.009       0.028
Book_to_Market_ratio    -0.0594      0.053     -1.119      0.264      -0.164       0.045
==============================================================================
Omnibus:                       20.467   Durbin-Watson:                   1.984
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               23.928
Skew:                          -0.537   Prob(JB):                     6.37e-06
Kurtosis:                       3.743   Cond. No.                         111.
==============================================================================
```

Podemos ver que con este cambio, pasamos de tener una r^2 de 0.293 a 0.363.

Con esto podemos observar como los outliers y valores influenciables afectan a nuestro modelo, ya que al eliminar estos tuvimos una mejoria en nuestro modelo

Referencias

- and. (2019, October 21). DataSklr. DataSklr. https://www.datasklr.com/ols-least-squares-regression/diagnostics-for-leverage-and-influence