PROYECTO INTELIGENCIA ARTIFICIAL GRUPO #6

INTEGRANTES:

1. MARIA CORDOVA
2. GIMGER FERNANDEZ
3. EDGAR PIN
4. NELSON RODRIGUEZ

IMPORTACION DE LIBRERIAS

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np

from mpl_toolkits.mplot3d import Axes3D

from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

import matplotlib.cm as cm
from sklearn.metrics import silhouette_samples, silhouette_score
```

CONEXION

```
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).
```

LECTURA DEL DATASET INDICADORES 2019

```
datos_2019 = pd.read_excel('/content/drive/My Drive/proyecto/indicadores2019_cia.xlsx')
```

ANALISIS EXPLORATORIO

```
datos_2019
```

	AÑO	EXPEDIENTE	NOMBRE	RAMA	DESCRIPCIÓN RAMA	RAMA 6 DÍGITOS	SUBRAMA 2 DÍGITOS
0	2019	1	ACEITES TROPICALES SOCIEDAD ANONIMA ATSA	A	AGRICULTURA, GANADERÃ A, SILVICULTURA Y PESCA.	A0126.01	A01
1	2019	2	ACERIA DEL ECUADOR CA ADELCA.	C	INDUSTRIAS MANUFACTURERAS.	C2410.25	C24
2	2019	3	ACERO COMERCIAL	G	COMERCIO AL POR MAYOR Y AL POR	G4659.99	G46
			AEROVIAS DEL CONTINENTE		TRANSPORTE Y		
					COMERCIO AL POR		

INDICAMOS LA RAMA A TRABAJAR EN ESTE CASO TOMAREMOS COMO REFERENCIA LA RAMA DE LA AGRICULTURA GANADERIA

```
du= datos_2019[(datos_2019.RAMA =="A")]
```

```
du.fillna(0)
du.dropna(inplace=True)
du
```

```
/usr/local/lib/python3.8/dist-packages/pandas/util/_decorators.py:311: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

LIMPIEZA DE DATOS

```
du.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1565 entries, 29 to 84841
Data columns (total 37 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   AÑO                                     1565 non-null   int64
 1   EXPEDIENTE                              1565 non-null   int64
 2   NOMBRE                                  1565 non-null   object
 3   RAMA                                    1565 non-null   object
 4   DESCRIPCIÓN RAMA                        1565 non-null   object
 5   RAMA 6 DÍGITOS                          1565 non-null   object
 6   SUBRAMA 2 DÍGITOS                       1565 non-null   object
 7   LIQUIDEZ CORRIENTE                      1565 non-null   float64
 8   PRUEBA ÁCIDA                            1565 non-null   float64
 9   ENDEUDAMIENTO DEL ACTIVO                1565 non-null   float64
 10  ENDEUDAMIENTO PATRIMONIAL               1565 non-null   float64
 11  ENDEUDAMIENTO A CORTO PLAZO             1565 non-null   float64
 12  ENDEUDAMIENTO A LARGO PLAZO             1565 non-null   float64
 13  COBERTURA DE INTERESES                  1565 non-null   float64
 14  ENDEUDAMIENTO DEL ACTIVO FIJO           1565 non-null   float64
 15  APALANCAMIENTO                          1565 non-null   float64
 16  APALANCAMIENTO FINANCIERO               1565 non-null   float64
 17  FORTALEZA PATRIMONIAL                   1565 non-null   float64
 18  ENDEUDAMIENTO PATRIMONIAL CORRIENTE     1565 non-null   float64
 19  ENDEUDAMIENTO PATRIMONIAL NO CORRIENTE  1565 non-null   float64
 20  APALANCAMIENTO A CORTO Y LARGO PLAZO    1565 non-null   float64
 21  ROTACIÓN DE CARTERA                     1565 non-null   float64
 22  ROTACIÓN DE ACTIVO FIJO                 1565 non-null   float64
 23  ROTACIÓN DE VENTAS                      1565 non-null   float64
 24  PERIODO MEDIO DE COBRANZA CORTO PLAZO   1565 non-null   float64
 25  PERIODO MEDIO DE PAGO CORTO PLAZO       1565 non-null   float64
 26  IMPACTO GASTOS ADMINISTRACIÓN Y VENTAS  1565 non-null   float64
 27  IMPACTO DE LA CARGA FINANCIERA          1565 non-null   float64
 28  RENTABILIDAD NETA DEL ACTIVO            1565 non-null   float64
 29  MARGEN BRUTO                            1565 non-null   float64
 30  MARGEN OPERACIONAL                      1565 non-null   float64
 31  RENTABILIDAD NETA DE VENTAS             1565 non-null   float64
 32  RENTABILIDAD OPERACIONAL DEL PATRIMONIO 1565 non-null   float64
 33  RENTABLIDAD FINANCIERA                  1565 non-null   float64
 34  UTILIDAD OPERACIONAL/TOTAL DE ACTIVOS   1565 non-null   float64
 35  ROE                                     1565 non-null   float64
 36  ROA                                     1565 non-null   float64
dtypes: float64(30), int64(2), object(5)
memory usage: 464.6+ KB
```

```
du.isnull().any()
```

```
AÑO                                        False
EXPEDIENTE                                 False
NOMBRE                                     False
RAMA                                       False
DESCRIPCIÓN RAMA                           False
RAMA 6 DÍGITOS                             False
SUBRAMA 2 DÍGITOS                          False
LIQUIDEZ CORRIENTE                         False
PRUEBA ÁCIDA                               False
ENDEUDAMIENTO DEL ACTIVO                   False
ENDEUDAMIENTO PATRIMONIAL                  False
ENDEUDAMIENTO A CORTO PLAZO                False
ENDEUDAMIENTO A LARGO PLAZO                False
COBERTURA DE INTERESES                     False
ENDEUDAMIENTO DEL ACTIVO FIJO              False
APALANCAMIENTO                             False
APALANCAMIENTO FINANCIERO                  False
FORTALEZA PATRIMONIAL                      False
ENDEUDAMIENTO PATRIMONIAL CORRIENTE        False
ENDEUDAMIENTO PATRIMONIAL NO CORRIENTE     False
APALANCAMIENTO A CORTO Y LARGO PLAZO       False
ROTACIÓN DE CARTERA                        False
ROTACIÓN DE ACTIVO FIJO                    False
ROTACIÓN DE VENTAS                         False
PERIODO MEDIO DE COBRANZA CORTO PLAZO      False
PERIODO MEDIO DE PAGO CORTO PLAZO          False
IMPACTO GASTOS ADMINISTRACIÓN Y VENTAS     False
IMPACTO DE LA CARGA FINANCIERA             False
RENTABILIDAD NETA DEL ACTIVO               False
MARGEN BRUTO                               False
MARGEN OPERACIONAL                         False
RENTABILIDAD NETA DE VENTAS                False
RENTABILIDAD OPERACIONAL DEL PATRIMONIO    False
```

```
RENTABILIDAD FINANCIERA                   False
UTILIDAD OPERACIONAL/TOTAL DE ACTIVOS     False
ROE                                       False
ROA                                       False
dtype: bool
```

```
data=du.drop(columns = ["AÑO"])
```

```
data
```

	EXPEDIENTE	NOMBRE	RAMA	DESCRIPCIÓN RAMA	RAMA 6 DÍGITOS	SUBRAMA 2 DÍGITOS	LIQUIDEZ CORRIENTE	PRUE ÁCI
29	258	COMPANIA ECUATORIANA DEL TE CA CETCA	A	AGRICULTURA, GANADERÃ A, SILVICULTURA Y PESCA.	A0127.09	A01	1.943430	0.9596
75	620	INCUBADORA NACIONAL CA INCA	A	AGRICULTURA, GANADERÃ A, SILVICULTURA Y PESCA.	A0146.01	A01	2.452399	2.3360
97	762	LA VINA CIA LTDA	A	AGRICULTURA, GANADERÃ A, SILVICULTURA Y PESCA.	A0119.03	A01	39.975285	39.9752
184	1324	TEXTILES TEXSA SA	A	AGRICULTURA, GANADERÃ A, SILVICULTURA Y PESCA.	A0116.05	A01	34.262527	26.1561
188	1342	PIRETRO LATINOAMERICANO CA PIRELA	A	AGRICULTURA, GANADERÃ A, SILVICULTURA Y PESCA.	A0128.03	A01	0.728577	0.6773
...	...	...	...	...	...	...	...	
82834	723649	LANFLOWERS S.A.	A	AGRICULTURA, GANADERÃ A, SILVICULTURA Y PESCA.	A0119.03	A01	0.955308	0.8940
82855	723681	LUA CACAO Y CHOCOLATE LUATE CIA.LTDA.	A	AGRICULTURA, GANADERÃ A, SILVICULTURA Y PESCA.	A0127.02	A01	2.285701	1.2710
83493	724476	CAMARONERA SAFANDO CAMSAFA S.A.	A	AGRICULTURA, GANADERÃ A, SILVICULTURA Y PESCA.	A0321.02	A03	1.011221	1.0112
83927	725017	AROMAS NATIVOS DEL ECUADOR ANESA NATIVAROMAS S.A.	A	AGRICULTURA, GANADERÃ A, SILVICULTURA Y PESCA.	A0150.00	A01	1.334564	1.3185
84841	726214	ESPINCORD S.A.	A	AGRICULTURA, GANADERÃ A, SILVICULTURA Y PESCA.	A0321.02	A03	0.776520	0.4945

1565 rows × 36 columns

MAPA DE CALOR DE LAS CORRELACIONES EN BASE A ROA

```
corr = data.set_index('ROA').corr()
corr.style.background_gradient(cmap ='coolwarm')
```

	EXPEDIENTE	LIQUIDEZ CORRIENTE	PRUEBA ÁCIDA	ENDEUDAMIENTO DEL ACTIVO	ENDEUDAMIENTO PATRIMONIAL	ENDEUDAMIEN A CORTO PLA
EXPEDIENTE	1.000000	-0.023753	-0.022061	0.263909	0.042671	0.0522
LIQUIDEZ CORRIENTE	-0.023753	1.000000	0.998908	-0.041646	0.004424	-0.0805
PRUEBA ÁCIDA	-0.022061	0.998908	1.000000	-0.039391	0.003615	-0.0728
ENDEUDAMIENTO DEL ACTIVO	0.263909	-0.041646	-0.039391	1.000000	0.105755	-0.0936
ENDEUDAMIENTO PATRIMONIAL	0.042671	0.004424	0.003615	0.105755	1.000000	-0.0897
ENDEUDAMIENTO A CORTO PLAZO	0.052286	-0.080536	-0.072802	-0.093629	-0.089750	1.0000
ENDEUDAMIENTO A LARGO PLAZO	-0.052286	0.080536	0.072802	0.093629	0.089750	-1.0000
COBERTURA DE INTERESES	0.015624	0.000412	0.000276	0.012688	0.002783	-0.0012
ENDEUDAMIENTO DEL ACTIVO FIJO	-0.015886	0.001116	0.001034	0.004011	-0.003592	0.0304
APALANCAMIENTO	0.042009	0.004461	0.003648	0.103946	0.999994	-0.0898
APALANCAMIENTO FINANCIERO	0.020176	-0.006620	-0.000606	-0.000482	0.021640	0.0307
FORTALEZA PATRIMONIAL	-0.021766	-0.000967	-0.002011	0.032855	0.634752	-0.0347
ENDEUDAMIENTO PATRIMONIAL CORRIENTE	0.109842	-0.010804	-0.009598	0.155289	0.643152	0.0932
ENDEUDAMIENTO PATRIMONIAL NO CORRIENTE	0.027961	0.011800	0.011918	0.068240	0.834794	-0.1414
APALANCAMIENTO A CORTO Y LARGO PLAZO	0.057781	0.007050	0.007519	0.106813	0.925606	-0.0954
ROTACIÓN DE CARTERA	-0.014610	-0.001503	-0.001699	-0.011809	0.003381	-0.0043
ROTACIÓN DE ACTIVO FIJO	-0.014046	-0.001226	-0.001271	0.007557	0.003228	0.0324
ROTACIÓN DE VENTAS	0.123411	-0.019228	-0.018289	0.126516	-0.019411	0.2132
PERIODO MEDIO DE COBRANZA CORTO PLAZO	-0.004155	0.000824	0.001052	0.020098	0.004322	-0.0245
PERIODO MEDIO DE PAGO CORTO PLAZO	-0.030784	-0.002184	-0.001842	0.010511	-0.004361	0.0389
IMPACTO GASTOS ADMINISTRACIÓN Y VENTAS	0.011484	-0.001012	-0.000799	0.070278	0.004917	-0.0348
IMPACTO DE LA CARGA FINANCIERA	0.055386	-0.003387	-0.002918	0.144522	0.045730	-0.2110
RENTABILIDAD NETA DEL ACTIVO	-0.061971	0.001172	0.000631	-0.531215	-0.035427	0.0759
MARGEN BRUTO	-0.003117	0.000943	0.000802	-0.026533	-0.002266	0.0098
MARGEN OPERACIONAL	-0.009136	0.000998	0.000804	-0.058056	-0.004180	0.0277
RENTABILIDAD NETA DE VENTAS	-0.036277	0.002013	0.001689	-0.143178	-0.005919	0.0427
RENTABILIDAD OPERACIONAL DEL PATRIMONIO	0.046006	-0.005072	-0.003066	0.075942	0.364625	0.0366
RENTABLIDAD	0.016647	0.001809	0.001496	0.026857	0.647977	0.0279

```
data.describe()
```

	EXPEDIENTE	LIQUIDEZ CORRIENTE	PRUEBA ÁCIDA	ENDEUDAMIENTO DEL ACTIVO	ENDEUDAMIENTO PATRIMONIAL	ENDEUDAMIENTO A CORTO PLAZO	ENDEI A LAI
count	1565.000000	1565.000000	1565.000000	1565.000000	1565.000000	1565.000000	15
mean	212651.573163	3.611597	3.158438	0.695185	19.091569	0.657878	
std	231486.688668	54.459652	54.359693	0.369955	135.514520	0.307271	
min	258.000000	0.003140	0.003140	0.001096	0.001097	0.000119	
25%	63075.000000	0.732460	0.532882	0.493342	0.973716	0.403596	
50%	129195.000000	1.121257	0.898475	0.712236	2.415759	0.714228	
75%	181689.000000	1.859574	1.557522	0.892908	6.722759	0.972112	
max	726214.000000	2145.126000	2145.126000	5.452787	3270.168000	1.000000	

8 rows × 31 columns

```
data.columns
```

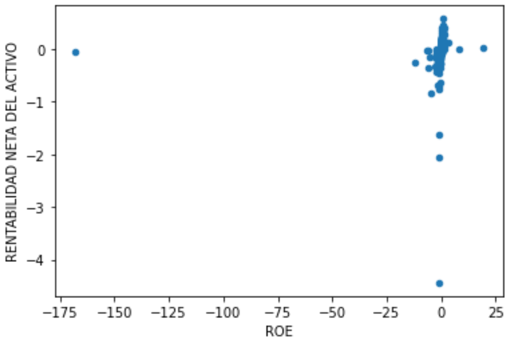
```
Index(['EXPEDIENTE', 'NOMBRE', 'RAMA', 'DESCRIPCIÓN RAMA', 'RAMA 6 DÍGITOS',
       'SUBRAMA 2 DÍGITOS', 'LIQUIDEZ CORRIENTE', 'PRUEBA ÁCIDA',
       'ENDEUDAMIENTO DEL ACTIVO', 'ENDEUDAMIENTO PATRIMONIAL',
       'ENDEUDAMIENTO A CORTO PLAZO', 'ENDEUDAMIENTO A LARGO PLAZO',
       'COBERTURA DE INTERESES', 'ENDEUDAMIENTO DEL ACTIVO FIJO',
       'APALANCAMIENTO', 'APALANCAMIENTO FINANCIERO', 'FORTALEZA PATRIMONIAL',
       'ENDEUDAMIENTO PATRIMONIAL CORRIENTE',
       'ENDEUDAMIENTO PATRIMONIAL NO CORRIENTE',
       'APALANCAMIENTO A CORTO Y LARGO PLAZO', 'ROTACIÓN DE CARTERA',
       'ROTACIÓN DE ACTIVO FIJO', 'ROTACIÓN DE VENTAS',
       'PERIODO MEDIO DE COBRANZA CORTO PLAZO',
       'PERIODO MEDIO DE PAGO CORTO PLAZO',
       'IMPACTO GASTOS ADMINISTRACIÓN Y VENTAS',
       'IMPACTO DE LA CARGA FINANCIERA', 'RENTABILIDAD NETA DEL ACTIVO',
       'MARGEN BRUTO', 'MARGEN OPERACIONAL', 'RENTABILIDAD NETA DE VENTAS',
       'RENTABILIDAD OPERACIONAL DEL PATRIMONIO', 'RENTABLIDAD FINANCIERA',
       'UTILIDAD OPERACIONAL/TOTAL DE ACTIVOS ', 'ROE', 'ROA'],
      dtype='object')
```

```
len(du)
```

```
1565
```

```
data[data['ROE']<500].sample(600).plot.scatter(x='ROE', y='RENTABILIDAD NETA DEL ACTIVO')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5e5b9f4c70>
```

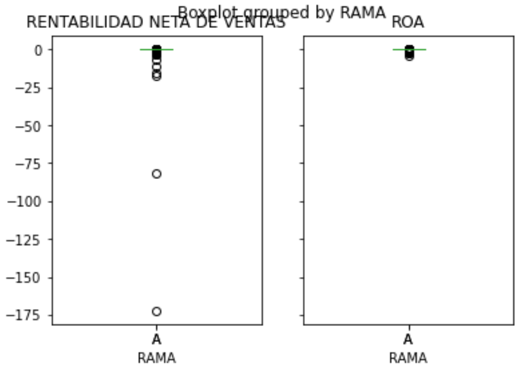


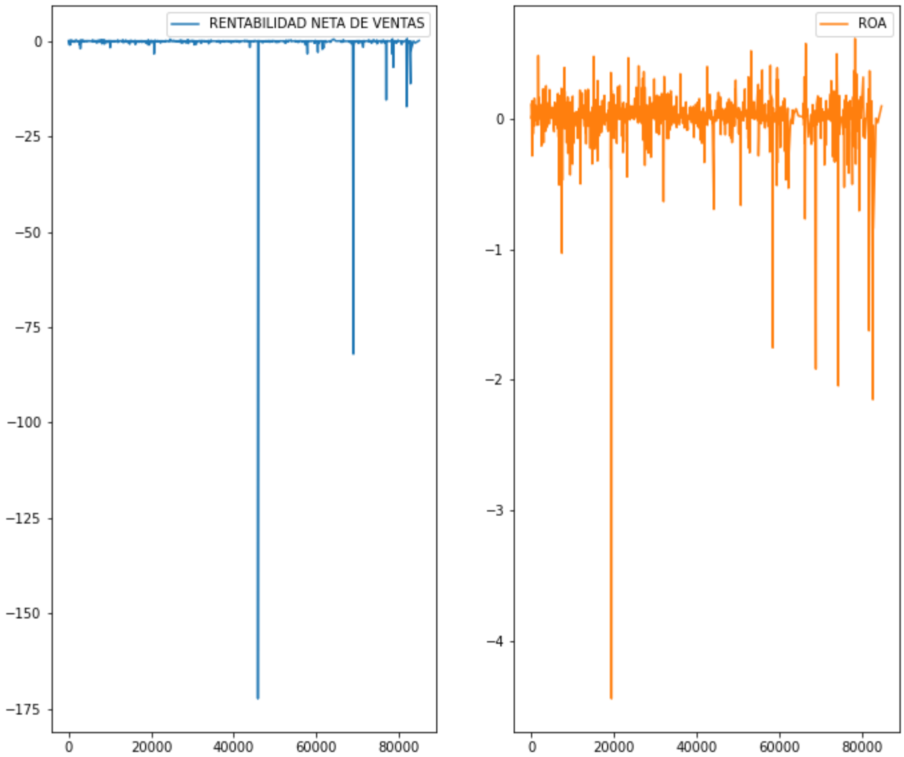

```
valor_por_salario = data.groupby("RAMA")["RENTABILIDAD NETA DE VENTAS","ROA"].max()
valor_por_salario.plot.bar()
plt.title("AGRICULTURA GANADERIA Y PESCA 2019")
plt.show()
```

```
<ipython-input-135-7169fe6bf98d>:1: FutureWarning: Indexing with multiple keys (implicitly conve
  valor_por_salario = data.groupby("RAMA")["RENTABILIDAD NETA DE VENTAS","ROA"].max()
```


```
du.boxplot(by ='RAMA', column=['RENTABILIDAD NETA DE VENTAS','ROA'], grid = False,)
du[['RENTABILIDAD NETA DE VENTAS','ROA']].plot(subplots=True, layout=(-1,5), figsize=(30,10) )
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f5edb1f4eb0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5edb201670>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5edaa49cd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5edaa4e2b0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5eda85ba00>]],
      dtype=object)
```





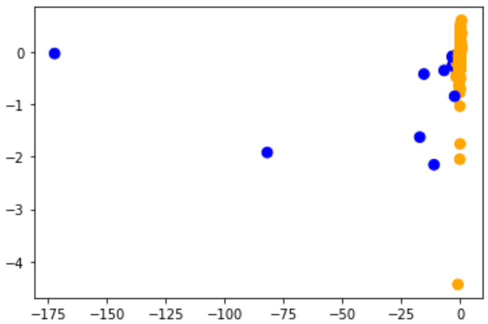
MODELO PREDICTIVO LINEAL

```
data.describe()
```

IENTO CTIVO	ENDEUDAMIENTO PATRIMONIAL	ENDEUDAMIENTO A CORTO PLAZO	ENDEUDAMIENTO A LARGO PLAZO	COBERTURA DE INTERESES	ENDEUDAMIENTO DEL ACTIVO FIJO	APALANCAMIENTO
00000	1565.000000	1565.000000	1565.000000	1.565000e+03	1565.000000	1565.000000
95185	19.091569	0.657878	0.342122	-3.073684e+02	42.312012	19.971441
01096	0.001097	0.000119	0.000000	-1.255988e+06	-0.202981	0.224578

```
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score
```

```
# Vamos a RECORTAR los datos en la zona donde se concentran más los puntos
# esto es en el eje X: entre 0 y 3.500
# y en el eje Y: entre 0 y 80.000
filtered_data = data[(data['RENTABILIDAD NETA DE VENTAS'] <=3) & (data['ROA'] < 3)]

colores=['orange','blue']
tamanios=[60,60]

f1 = filtered_data['RENTABILIDAD NETA DE VENTAS'].values
f2 = filtered_data['ROA'].values

# Vamos a pintar en colores los puntos por debajo y por encima de la media de Cantidad de Palabras
asignar=[]
for index, row in filtered_data.iterrows():
    if(row['RENTABILIDAD NETA DE VENTAS']>=-2):
        asignar.append(colores[0])
    else:
        asignar.append(colores[1])

plt.scatter(f1, f2, c=asignar, s=tamanios[0])
plt.show()
```

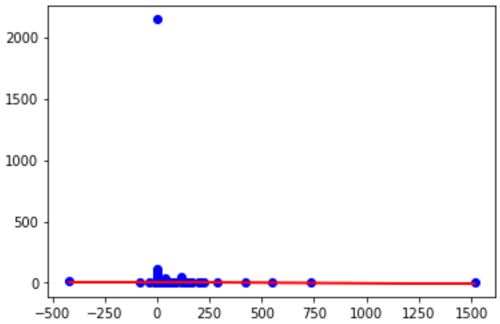



```
# Asignamos nuestra variable de entrada X para entrenamiento y las etiquetas Y.
dataX =filtered_data[["RENTABILIDAD OPERACIONAL DEL PATRIMONIO"]]
X_train = np.array(dataX)
y_train = filtered_data['LIQUIDEZ CORRIENTE'].values

# Creamos el objeto de Regresión Linear
regr = linear_model.LinearRegression()

# Entrenamos nuestro modelo
regr.fit(X_train, y_train)

# Hacemos las predicciones que en definitiva una línea (en este caso, al ser 2D)
y_pred = regr.predict(X_train)

# Veamos los coeficienetes obtenidos, En nuestro caso, serán la Tangente
print('Coefficients: \n', regr.coef_)
# Este es el valor donde corta el eje Y (en X=0)
print('Independent term: \n', regr.intercept_)
# Error Cuadrado Medio
print("Mean squared error: %.2f" % mean_squared_error(y_train, y_pred))
# Puntaje de Varianza. El mejor puntaje es un 1.0
print('Variance score: %.2f' % r2_score(y_train, y_pred))
```

```
Coefficients:
 [-0.00550695]
Independent term:
 3.643255931916952
Mean squared error: 2963.88
Variance score: 0.00
```

```
y_pred = regr.predict(X_train)
plt.scatter(X_train,y_train,color='b')
```

```
plt.plot(X_train, y_pred,color='r')
plt.show()
```



```
import torch
from torch import nn
from torch.nn import functional as F
import torch.utils.data as Data
from torch import optim
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split

import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
import tqdm
```

```
D1= data[['RENTABILIDAD NETA DE VENTAS','ROA']]
```

```
D2=data[['ROE']]
```

```
X = D1.values
y = D2.values

X_train, X_test, y_train, y_test =  train_test_split(X,y, random_state=1)
```

```
print(X.shape)
print(y.shape)
```

```
(1565, 2)
(1565, 1)
```

```
class DataMaker(Data.Dataset):
  def __init__(self, X, y):
    #scaler = StandardScaler()
    scaler = MinMaxScaler()
    self.targets = scaler.fit_transform(X.astype(np.float32))
    self.labels = y.astype(np.float32)

  def __getitem__(self, index):
    return self.targets[index, :], self.labels[index]

  def __len__(self):
    return len(self.targets)
```

```
class MLP(nn.Module):
  def __init__(self, n_features, hiddenA, hiddenB):
    super().__init__()

    self.linearA = nn.Linear(n_features, hiddenA)
    self.linearB = nn.Linear(hiddenA, hiddenB)
    self.linearC = nn.Linear(hiddenB, 1)

  def forward(self, x):
    yA = F.relu(self.linearA(x))
    yB = F.relu(self.linearB(yA))
    return self.linearC(yB)
```

```
torch.manual_seed(1)
```

```
<torch._C.Generator at 0x7f5ede49cb10>
```

```
train_set = DataMaker(X_train, y_train)
test_set = DataMaker(X_test, y_test)
```

```
bs = 25
n_samples , n_features = X.shape
train_loader = Data.DataLoader(train_set, batch_size=bs, shuffle=True)
test_loader = Data.DataLoader(test_set, batch_size=bs, shuffle=True)
```

```
model = MLP(n_features,100,50)
```

```
criterion = nn.MSELoss(size_average=False)
optimizer = optim.Adam(model.parameters(),lr=0.01)
```

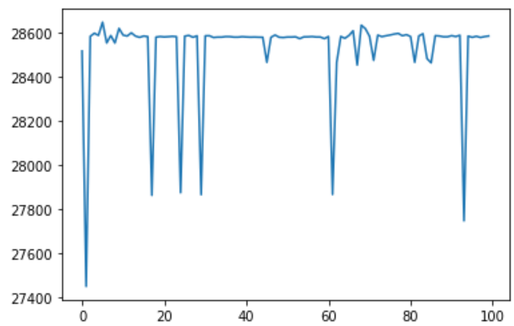
```
/usr/local/lib/python3.8/dist-packages/torch/nn/_reduction.py:42: UserWarning: size_average and reduce args will be deprecated, ple
  warnings.warn(warning.format(ret))
```

```
n_epochs = 100
all_losses = []
for epoch in range(n_epochs):
    progress_bar = tqdm.notebook.tqdm(train_loader, leave=False)
    losses = []
    total = 0
    for inputs, target in progress_bar:
        optimizer.zero_grad()
        y_pred = model(inputs)
        loss = criterion(y_pred, torch.unsqueeze(target,dim=1))

        loss.backward()

        optimizer.step()

        progress_bar.set_description(f'Loss: {loss.item():.3f}')

        losses.append(loss.item())
        total += 1

    epoch_loss = sum(losses) / total
    all_losses.append(epoch_loss)

    mess = f"Epoch #{epoch+1}\tLoss: {all_losses[-1]:.3f}"
    tqdm.tqdm.write(mess)
```

```
/usr/local/lib/python3.8/dist-packages/torch/nn/modules/loss.py:536: UserWarning: Using a target
  return F.mse_loss(input, target, reduction=self.reduction)
/usr/local/lib/python3.8/dist-packages/torch/nn/modules/loss.py:536: UserWarning: Using a target
  return F.mse_loss(input, target, reduction=self.reduction)
Epoch #1        Loss: 28516.329
Epoch #2        Loss: 27449.047
Epoch #3        Loss: 28582.948
Epoch #4        Loss: 28596.926
Epoch #5        Loss: 28587.307
Epoch #6        Loss: 28647.000
Epoch #7        Loss: 28553.039
Epoch #8        Loss: 28586.610
Epoch #9        Loss: 28552.749
Epoch #10       Loss: 28618.908
Epoch #11       Loss: 28588.503
Epoch #12       Loss: 28584.200
Epoch #13       Loss: 28598.852
Epoch #14       Loss: 28584.123
Epoch #15       Loss: 28578.428
Epoch #16       Loss: 28584.214
Epoch #17       Loss: 28581.757
Epoch #18       Loss: 27862.179
Epoch #19       Loss: 28579.420
Epoch #20       Loss: 28582.000
Epoch #21       Loss: 28580.298
Epoch #22       Loss: 28581.422
Epoch #23       Loss: 28582.912
Epoch #24       Loss: 28581.470
Epoch #25       Loss: 27873.903
Epoch #26       Loss: 28583.687
Epoch #27       Loss: 28587.136
Epoch #28       Loss: 28578.733
Epoch #29       Loss: 28585.375
Epoch #30       Loss: 27864.900
Epoch #31       Loss: 28585.124
Epoch #32       Loss: 28585.713
Epoch #33       Loss: 28577.692
Epoch #34       Loss: 28579.376
Epoch #35       Loss: 28579.458
Epoch #36       Loss: 28581.796
Epoch #37       Loss: 28581.475
Epoch #38       Loss: 28579.399
Epoch #39       Loss: 28579.390
Epoch #40       Loss: 28581.291
Epoch #41       Loss: 28580.117
Epoch #42       Loss: 28579.243
Epoch #43       Loss: 28579.753
Epoch #44       Loss: 28578.990
Epoch #45       Loss: 28578.792
Epoch #46       Loss: 28465.312
Epoch #47       Loss: 28578.606
Epoch #48       Loss: 28588.929
Epoch #49       Loss: 28578.826
Epoch #50       Loss: 28577.594
Epoch #51       Loss: 28579.735
Epoch #52       Loss: 28579.762
Epoch #53       Loss: 28580.724
Epoch #54       Loss: 28572.372
Epoch #55       Loss: 28580.495
Epoch #56       Loss: 28580.813
Epoch #57       Loss: 28581.828
Epoch #58       Loss: 28579.981
```

```
plt.plot(all_losses)
```

```
[<matplotlib.lines.Line2D at 0x7f5edfbcc070>]
```

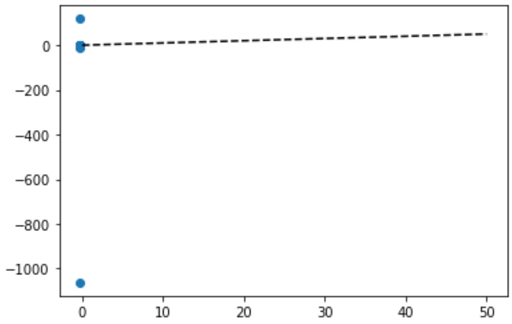



```
Epoch #77       Loss: 28593.259
```

```
y_pred = []
y_true = []
model.train(False)
for inputs, targets in test_loader:
  y_pred.extend(model(inputs).data.numpy())
  y_true.extend(targets.numpy())
```

```
print(len(y_pred),len(y_true))
plt.scatter(x=y_pred, y=y_true)
plt.plot([0,50], [0,50],'--k')
```

```
392 392
[<matplotlib.lines.Line2D at 0x7f5edf5e00a0>]
```




```
print("MAE:", mean_absolute_error(y_true, y_pred))
print("MSE:", mean_squared_error(y_true, y_pred))
print("R^2:", r2_score(y_true, y_pred))
```

```
MAE: 3.5467746
MSE: 2915.4019
R^2: -0.0015956673141215294
```

EVALUANDO MODELO 2019

```
dfs_rg_l = data[['ROE','RENTABILIDAD OPERACIONAL DEL PATRIMONIO', 'RENTABILIDAD NETA DE VENTAS']]
#Remover Outliers
```

```
dfs_rg_l
```

	ROE	RENTABILIDAD OPERACIONAL DEL PATRIMONIO	RENTABILIDAD NETA DE VENTAS
29	0.234019	1.272980	0.110454
75	0.090584	0.469808	0.031100
97	-0.001185	-0.001166	-0.059696
184	0.025209	0.202467	0.053528
188	0.010888	0.896037	0.009275
...	...	...	...
82834	-1.386262	-1.423590	-0.023413
82855	-4.738655	-4.718581	-2.395046
83493	0.075985	21.726015	0.000585
83927	-0.197612	0.099645	-0.458346
84841	0.909427	3.478133	0.133201

1565 rows × 3 columns

```
dfs_Y = dfs_rg_l[['ROE']]
dfs_Y
```

ROE

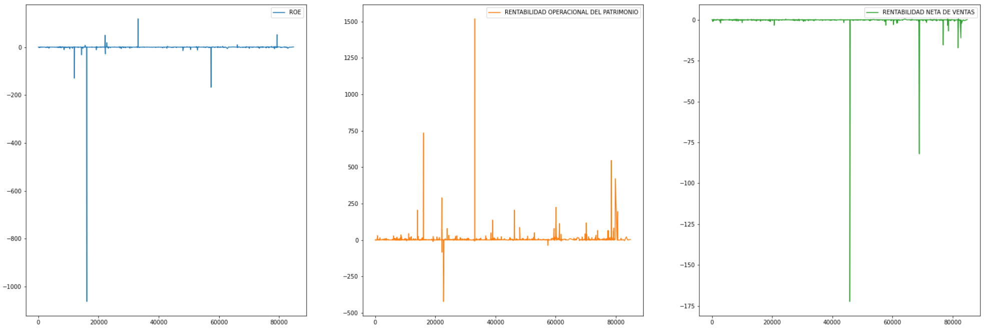
```
dfs_X = dfs_rg_l[['ROE','RENTABILIDAD OPERACIONAL DEL PATRIMONIO', 'RENTABILIDAD NETA DE VENTAS']]
dfs_X
```

	ROE	RENTABILIDAD OPERACIONAL DEL PATRIMONIO	RENTABILIDAD NETA DE VENTAS
29	0.234019	1.272980	0.110454
75	0.090584	0.469808	0.031100
97	-0.001185	-0.001166	-0.059696
184	0.025209	0.202467	0.053528
188	0.010888	0.896037	0.009275
...	...	...	...
82834	-1.386262	-1.423590	-0.023413
82855	-4.738655	-4.718581	-2.395046
83493	0.075985	21.726015	0.000585
83927	-0.197612	0.099645	-0.458346
84841	0.909427	3.478133	0.133201

1565 rows × 3 columns

```
dfs_X.plot(subplots=True, layout=(-1,3), figsize=(30,10) )
```

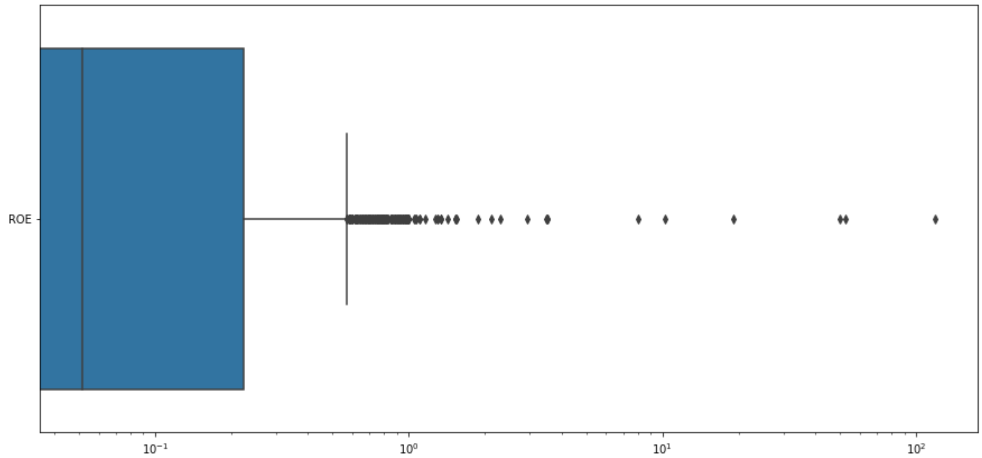
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f5edb0703a0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5edaf94910>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5edb3bdb80>]],
      dtype=object)
```




```
dfs_Y.plot(subplots=True, layout=(-1,3), figsize=(10,5) )
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f5e5b778460>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5e5be6a4f0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5e5bdd75b0>]],
      dtype=object)
```

```
plt.figure(figsize = (15,7))
ax =sns.boxplot(data = dfs_Y, orient="h")
ax.set(xscale="log")
plt.show()
```




```
scaler = StandardScaler()
inputs = scaler.fit_transform(dfs_X)
dX = np.array(inputs, dtype='float32')
dY = np.array(dfs_Y, dtype='float32')
```

```
dX
```

```
array([[ 0.03602017, -0.08927003,  0.06761088],
       [ 0.03083186, -0.10528919,  0.05132422],
       [ 0.02751244, -0.11468271,  0.03268897],
       ...,
       [ 0.03030379,  0.31866318,  0.0450612 ],
       [ 0.02040731, -0.11267205, -0.04913051],
       [ 0.06045087, -0.04528852,  0.07227939]], dtype=float32)
```

```
dY
```

```
array([[ 0.23401906],
       [ 0.09058373],
       [-0.00118463],
       ...,
       [ 0.07598484],
       [-0.19761197],
       [ 0.90942705]], dtype=float32)
```

```
print('tamaño de dX INDICADORES : ',dX.shape) #tamaño de dX INDICADORES
print('tamaño de dY ROE : ',dY.shape) #tamaño de dY ROE
```

```
tamaño de dX INDICADORES :  (1565, 3)
tamaño de dY ROE :  (1565, 1)
```

```
X = torch.from_numpy(dX)
Y = torch.from_numpy(dY)
```

```
from torch.utils.data import TensorDataset
dataset = TensorDataset(X,Y)
```

```
dataset[1:2]
```

```
(tensor([[ 0.0308, -0.1053,  0.0513]]), tensor([[0.0906]]))
```

```
from torch.utils.data import DataLoader
bs=64
train_loader = DataLoader(dataset,batch_size=bs,shuffle=True)
```

```
class ModeloRegresionLineal(torch.nn.Module):
 def __init__(self):
  super(ModeloRegresionLineal, self).__init__()
  self.linear = torch.nn.Linear(3,1) #X @ w.t() + b
 def forward(self, x):
  y_pred = self.linear(x)
  return y_pred
```

```
epochs = 30
ta = 1e-9 # Tasa Aprendizaje
modelo = ModeloRegresionLineal()
funcion_costo = torch.nn.MSELoss(reduction = 'mean')
optimizer = torch.optim.SGD(modelo.parameters(), lr = ta) #Actualiza los pesos w y el bias b

for i in range(epochs):
  for x,y in train_loader:
   preds = modelo(x)
   loss = funcion_costo(preds, y)
   optimizer.zero_grad()
   loss.backward()
   optimizer.step()

  print(f"Epoch {i}/{epochs}: Loss {loss}")
```

```
Epoch 0/30: Loss 3.7705414295196533
Epoch 1/30: Loss 1.0743095874786377
Epoch 2/30: Loss 0.627961277961731
Epoch 3/30: Loss 0.5228757858276367
Epoch 4/30: Loss 0.3778226375579834
Epoch 5/30: Loss 0.5754765272140503
Epoch 6/30: Loss 0.6021027565002441
Epoch 7/30: Loss 568.1649169921875
Epoch 8/30: Loss 0.6125538349151611
Epoch 9/30: Loss 0.6532401442527771
Epoch 10/30: Loss 37.30521774291992
Epoch 11/30: Loss 0.43684738874435425
Epoch 12/30: Loss 0.5169415473937988
Epoch 13/30: Loss 38331.1640625
Epoch 14/30: Loss 0.63414466381073
Epoch 15/30: Loss 0.9429479837417603
Epoch 16/30: Loss 0.6123935580253601
Epoch 17/30: Loss 4.168334007263184
Epoch 18/30: Loss 0.39348939061164856
Epoch 19/30: Loss 0.6212916374206543
Epoch 20/30: Loss 0.5643094182014465
Epoch 21/30: Loss 0.4787541627883911
Epoch 22/30: Loss 0.6774821877479553
Epoch 23/30: Loss 0.5652852654457092
Epoch 24/30: Loss 4.309688568115234
Epoch 25/30: Loss 3.741856098175049
Epoch 26/30: Loss 79.4842300415039
Epoch 27/30: Loss 1.4696449041366577
Epoch 28/30: Loss 17.733261108398438
Epoch 29/30: Loss 0.47814205288887024
```

```
modelo.linear.weight #w
```

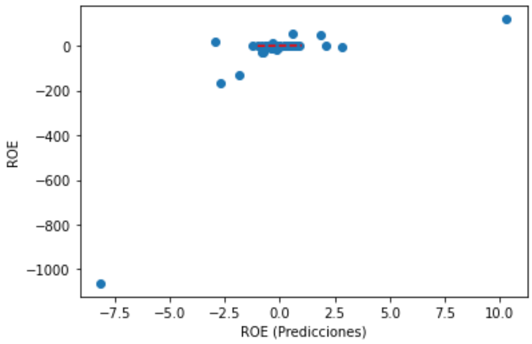
```
Parameter containing:
tensor([[0.3176, 0.3116, 0.0187]], requires_grad=True)
```

```
modelo.linear.bias #b
```

```
Parameter containing:
tensor([-0.4816], requires_grad=True)
```

```
# Evaluando el modelo
y_pred = []
y_true = []
modelo.train(False)
for inputs, targets in train_loader:
 y_pred.extend(modelo(inputs).data.numpy())
 y_true.extend(targets.numpy())
plt.scatter(y_pred, y_true)
plt.ylabel('ROE')
```

```
plt.xlabel('ROE (Predicciones)')
plt.plot([-1,1], [-1, 1], '--k', c='r')
plt.show()
# Calculando Errores
mae = mean_absolute_error(y_true=y_true, y_pred=y_pred)
mse = mean_squared_error(y_true=y_true, y_pred=y_pred, squared=True)
rmse = mean_squared_error(y_true=y_true, y_pred=y_pred, squared=False)
print(f"\nResultados Del año 2019 datos:")
print(f"MAE: {mae}")
print(f"MSE: {mse}")
print(f"RMSE: {rmse}")
```




```
Resultados Del año 2019 datos:
MAE: 1.7818398475646973
MSE: 751.5975952148438
RMSE: 27.415279388427734
```

**EVALUANDO MODELO 2020

LECTURAS DATASET

```
datos_2018 = pd.read_excel('/content/drive/My Drive/proyecto/indicadores2018_cia.xlsx')
```

```
datos_2017 = pd.read_excel('/content/drive/My Drive/proyecto/indicadores2017_cia.xlsx')
```

```
datos_2020 = pd.read_excel('/content/drive/My Drive/proyecto/indicadores2020_cia.xlsx')
```

```
datos_2020.fillna(0)
datos_2020.dropna(inplace=True)
datos_2020
```

	AÑO	EXPEDIENTE	NOMBRE	RAMA	DESCRIPCIÓN RAMA	RAMA 6 DÍGITOS	SUBRAMA 2 DÍGITOS	LIQU CORRI
0	2020	1	ACEITES TROPICALES SOCIEDAD ANONIMA ATSA	A	AGRICULTURA, GANADERÃ A, SILVICULTURA Y PESCA.	A0126.01	A01	6.35
2	2020	3	ACERO COMERCIAL ECUATORIANO S.A.	G	COMERCIO AL POR MAYOR Y AL POR MENOR REPARACIÃ...	G4659.99	G46	6.85
4	2020	22	AGENCIAS Y REPRESENTACIONES CORDOVEZ SA	G	COMERCIO AL POR MAYOR Y AL POR MENOR REPARACIÃ...	G4630.95	G46	1.74
7	2020	49	ALMACENES EL GLOBO DE QUITO SA	G	COMERCIO AL POR MAYOR Y AL POR MENOR REPARACIÃ...	G4719.00	G47	2.36

```
dfs_rg_l = datos_2020[['ROE','RENTABILIDAD OPERACIONAL DEL PATRIMONIO', 'RENTABILIDAD NETA DE VENTAS']]
#Remover Outliers

dfs_rg_l
```

	ROE	RENTABILIDAD OPERACIONAL DEL PATRIMONIO	RENTABILIDAD NETA DE VENTAS
0	0.017822	0.079047	0.145618
2	-0.080518	-0.120850	-0.034202
4	0.093772	0.167538	0.033429
7	-0.985009	-0.901599	-0.217195
8	-0.057209	0.319549	-0.044041
...	...	...	...
83769	0.090264	-6.274013	0.012410
83825	-0.341922	0.310856	-0.036434
83830	0.899413	0.901391	0.326974
83880	-1.014521	-0.762914	-51.375614
84433	0.533966	49.896259	0.008106

14020 rows × 3 columns

```
dfs_Y = dfs_rg_l[['ROE']]
dfs_Y
```

	ROE
0	0.017822
2	-0.080518
4	0.093772
7	-0.985009
8	-0.057209
...	...
83769	0.090264
83825	-0.341922
83830	0.899413
83880	-1.014521
84433	0.533966

14020 rows × 1 columns

```
dfs_X = dfs_rg_l[['ROE','RENTABILIDAD OPERACIONAL DEL PATRIMONIO', 'RENTABILIDAD NETA DE VENTAS']]
dfs_X
```

	ROE	RENTABILIDAD OPERACIONAL DEL PATRIMONIO	RENTABILIDAD NETA DE VENTAS
0	0.017822	0.079047	0.145618
2	-0.080518	-0.120850	-0.034202
4	0.093772	0.167538	0.033429
7	-0.985009	-0.901599	-0.217195
8	-0.057209	0.319549	-0.044041
...	...	...	...
83769	0.090264	-6.274013	0.012410
83825	-0.341922	0.310856	-0.036434
83830	0.899413	0.901391	0.326974
83880	-1.014521	-0.762914	-51.375614
84433	0.533966	49.896259	0.008106

```
scaler = StandardScaler()
inputs = scaler.fit_transform(dfs_X)
dX = np.array(inputs, dtype='float32')
dY = np.array(dfs_Y, dtype='float32')
```

```
dX
```

```
array([[ 0.04036301, -0.03196327,  0.0089452 ],
       [ 0.03204108, -0.03560445,  0.00878947],
       [ 0.04679025, -0.03035139,  0.00884804],
       ...,
       [ 0.11496707, -0.01698408,  0.00910226],
       [-0.04699835, -0.0472998 , -0.03567374],
       [ 0.08404137,  0.87546974,  0.00882611]], dtype=float32)
```

```
dY
```

```
array([[ 0.01782197],
       [-0.08051773],
       [ 0.09377224],
       ...,
       [ 0.8994128 ],
       [-1.014521  ],
       [ 0.53396606]], dtype=float32)
```

```
print('tamaño de dX INDICADORES : ',dX.shape) #tamaño de dX INDICADORES
print('tamaño de dY ROE : ',dY.shape) #tamaño de dY ROE
```

```
tamaño de dX INDICADORES :  (14020, 3)
tamaño de dY ROE :  (14020, 1)
```

```
X = torch.from_numpy(dX)
Y = torch.from_numpy(dY)
```

```
from torch.utils.data import TensorDataset
dataset = TensorDataset(X,Y)
```

```
dataset[1:2]
```

```
(tensor([[ 0.0320, -0.0356,  0.0088]]), tensor([[-0.0805]]))
```

```
from torch.utils.data import DataLoader
bs=64
train_loader = DataLoader(dataset,batch_size=bs,shuffle=True)
```

```
class ModeloRegresionLineal(torch.nn.Module):
 def __init__(self):
  super(ModeloRegresionLineal, self).__init__()
  self.linear = torch.nn.Linear(3,1) #X @ w.t() + b
 def forward(self, x):
  y_pred = self.linear(x)
  return y_pred
```

```
epochs = 30
ta = 1e-9 # Tasa Aprendizaje
modelo = ModeloRegresionLineal()
```

```
funcion_costo = torch.nn.MSELoss(reduction = 'mean')
optimizer = torch.optim.SGD(modelo.parameters(), lr = ta) #Actualiza los pesos w y el bias b

for i in range(epochs):
  for x,y in train_loader:
   preds = modelo(x)
   loss = funcion_costo(preds, y)
   optimizer.zero_grad()
   loss.backward()
   optimizer.step()

  print(f"Epoch {i}/{epochs}: Loss {loss}")
```

```
Epoch 0/30: Loss 0.40508660674095154
Epoch 1/30: Loss 0.07188256084918976
Epoch 2/30: Loss 0.05984983593225479
Epoch 3/30: Loss 0.39519166946411133
Epoch 4/30: Loss 0.08602160215377808
Epoch 5/30: Loss 0.050189536064863205
Epoch 6/30: Loss 0.14426694810390472
Epoch 7/30: Loss 0.056205134838819504
Epoch 8/30: Loss 0.3164328336715698
Epoch 9/30: Loss 0.7531360983848572
Epoch 10/30: Loss 0.10635189712047577
Epoch 11/30: Loss 0.07201410084962845
Epoch 12/30: Loss 0.0933629646897316
Epoch 13/30: Loss 0.08790195733308792
Epoch 14/30: Loss 0.032197535037994385
Epoch 15/30: Loss 0.03589369356632233
Epoch 16/30: Loss 0.07310549169778824
Epoch 17/30: Loss 0.2789304256439209
Epoch 18/30: Loss 0.244041308760643
Epoch 19/30: Loss 0.12054206430912018
Epoch 20/30: Loss 0.5896925926208496
Epoch 21/30: Loss 0.532899022102356
Epoch 22/30: Loss 0.5470948815345764
Epoch 23/30: Loss 0.17196761071681976
Epoch 24/30: Loss 0.09500324726104736
Epoch 25/30: Loss 0.5141614675521851
Epoch 26/30: Loss 1.830703854560852
Epoch 27/30: Loss 1.7079228162765503
Epoch 28/30: Loss 0.11422447860240936
Epoch 29/30: Loss 0.48125335574150085
```

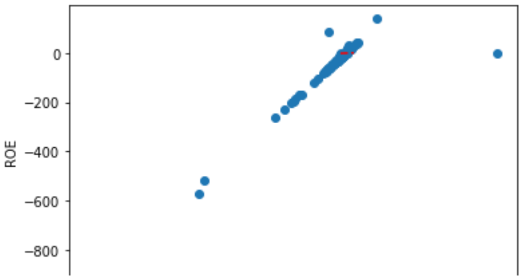
```
modelo.linear.weight #w
```

```
Parameter containing:
tensor([[-0.4006,  0.0599, -0.4219]], requires_grad=True)
```

```
modelo.linear.bias #b
```

```
Parameter containing:
tensor([0.3034], requires_grad=True)
```

```
# Evaluando el modelo
y_pred = []
y_true = []
modelo.train(False)
for inputs, targets in train_loader:
 y_pred.extend(modelo(inputs).data.numpy())
 y_true.extend(targets.numpy())
plt.scatter(y_pred, y_true)
plt.ylabel('ROE')
plt.xlabel('ROE (Predicciones)')
plt.plot([-1,1], [-1, 1], '--k', c='r')
plt.show()
# Calculando Errores
mae = mean_absolute_error(y_true=y_true, y_pred=y_pred)
mse = mean_squared_error(y_true=y_true, y_pred=y_pred, squared=True)
rmse = mean_squared_error(y_true=y_true, y_pred=y_pred, squared=False)
print(f"\n Prediccion año 2020 datos:")
print(f"MAE: {mae}")
print(f"MSE: {mse}")
print(f"RMSE: {rmse}")
```



EVALUANDO MODELO 2017

```
datos_2017.fillna(0)
datos_2017.dropna(inplace=True)
datos_2017
```

	AÑO	EXPEDIENTE	NOMBRE	RAMA	DESCRIPCIÓN RAMA	RAMA 6 DÍGITOS	SUBRAMA 2 DÍGITOS
1	2017	2	ACERIA DEL ECUADOR CA ADELCA.	C	INDUSTRIAS MANUFACTURERAS.	C2410.25	C24
2	2017	3	ACERO COMERCIAL ECUATORIANO S.A.	G	COMERCIO AL POR MAYOR Y AL POR MENOR REPARACIÃ...	G4659.99	G46
4	2017	22	AGENCIAS Y REPRESENTACIONES CORDOVEZ SA	G	COMERCIO AL POR MAYOR Y AL POR MENOR REPARACIÃ...	G4630.95	G46
7	2017	49	ALMACENES EL GLOBO DE QUITO SA	G	COMERCIO AL POR MAYOR Y AL POR MENOR REPARACIÃ...	G4719.00	G47
8	2017	63	CONFITECA C.A.	C	INDUSTRIAS MANUFACTURERAS.	C1073.21	C10
...	...	...	...	...	...	...	...
80431	2017	714930	CAMARONERA MARCRESCI MARCRESCISA S.A.	A	AGRICULTURA, GANADERÃ A, SILVICULTURA Y PESCA.	A0321.02	A03
80447	2017	714947	COMERCIALIZADORA Y CONSERVADORA DE PESCADO BUS...	A	AGRICULTURA, GANADERÃ A, SILVICULTURA Y PESCA.	A0311.01	A03
80460	2017	714960	METALAUSTROFERRETERIA CIA.LTDA.	G	COMERCIO AL POR MAYOR Y AL POR MENOR REPARACIÃ...	G4610.05	G46
80679	2017	715220	REYLACTEOS C.L.	C	INDUSTRIAS MANUFACTURERAS.	C1050.01	C10
80680	2017	715221	EXPOPLAST C.L.	C	INDUSTRIAS MANUFACTURERAS.	C2220.91	C22

13049 rows × 37 columns

```
dfs_rg_l = datos_2020[['ROE','RENTABILIDAD OPERACIONAL DEL PATRIMONIO', 'RENTABILIDAD NETA DE VENTAS']]
#Remover Outliers

dfs_rg_l
```

	ROE	RENTABILIDAD OPERACIONAL DEL PATRIMONIO	RENTABILIDAD NETA DE VENTAS
0	0.017822	0.079047	0.145618
2	-0.080518	-0.120850	-0.034202
4	0.093772	0.167538	0.033429
7	-0.985009	-0.901599	-0.217195
8	-0.057209	0.319549	-0.044041
...	...	...	...
83769	0.090264	-6.274013	0.012410

```
dfs_Y = dfs_rg_l[['ROE']]
dfs_Y
```

	ROE
0	0.017822
2	-0.080518
4	0.093772
7	-0.985009
8	-0.057209
...	...
83769	0.090264
83825	-0.341922
83830	0.899413
83880	-1.014521
84433	0.533966

14020 rows × 1 columns

```
dfs_X = dfs_rg_l[['ROE','RENTABILIDAD OPERACIONAL DEL PATRIMONIO', 'RENTABILIDAD NETA DE VENTAS']]
dfs_X
```

	ROE	RENTABILIDAD OPERACIONAL DEL PATRIMONIO	RENTABILIDAD NETA DE VENTAS
0	0.017822	0.079047	0.145618
2	-0.080518	-0.120850	-0.034202
4	0.093772	0.167538	0.033429
7	-0.985009	-0.901599	-0.217195
8	-0.057209	0.319549	-0.044041
...	...	...	...
83769	0.090264	-6.274013	0.012410
83825	-0.341922	0.310856	-0.036434
83830	0.899413	0.901391	0.326974
83880	-1.014521	-0.762914	-51.375614
84433	0.533966	49.896259	0.008106

14020 rows × 3 columns

```
scaler = StandardScaler()
inputs = scaler.fit_transform(dfs_X)
dX = np.array(inputs, dtype='float32')
dY = np.array(dfs_Y, dtype='float32')
```

```
dX
```

```
array([[ 0.04036301, -0.03196327,  0.0089452 ],
       [ 0.03204108, -0.03560445,  0.00878947],
       [ 0.04679025, -0.03035139,  0.00884804],
       ...,
       [ 0.11496707, -0.01698408,  0.00910226],
       [-0.04699835, -0.0472998 , -0.03567374],
       [ 0.08404137,  0.87546974,  0.00882611]], dtype=float32)
```

```
dY
```

```
array([[ 0.01782197],
       [-0.08051773],
       [ 0.09377224],
       ...,
       [ 0.8994128 ],
       [-1.014521  ],
       [ 0.53396606]], dtype=float32)
```

```
print('tamaño de dX INDICADORES : ',dX.shape) #tamaño de dX INDICADORES
print('tamaño de dY ROE : ',dY.shape) #tamaño de dY ROE
```

```
tamaño de dX INDICADORES :  (14020, 3)
tamaño de dY ROE :  (14020, 1)
```

```
X = torch.from_numpy(dX)
Y = torch.from_numpy(dY)
```

```
from torch.utils.data import TensorDataset
dataset = TensorDataset(X,Y)
```

```
dataset[1:2]
```

```
(tensor([[ 0.0320, -0.0356,  0.0088]]), tensor([[-0.0805]]))
```

```
from torch.utils.data import DataLoader
bs=64
train_loader = DataLoader(dataset,batch_size=bs,shuffle=True)
```

```
class ModeloRegresionLineal(torch.nn.Module):
 def __init__(self):
  super(ModeloRegresionLineal, self).__init__()
  self.linear = torch.nn.Linear(3,1) #X @ w.t() + b
 def forward(self, x):
  y_pred = self.linear(x)
  return y_pred
```

```
epochs = 30
ta = 1e-9 # Tasa Aprendizaje
modelo = ModeloRegresionLineal()
funcion_costo = torch.nn.MSELoss(reduction = 'mean')
optimizer = torch.optim.SGD(modelo.parameters(), lr = ta) #Actualiza los pesos w y el bias b

for i in range(epochs):
  for x,y in train_loader:
   preds = modelo(x)
   loss = funcion_costo(preds, y)
   optimizer.zero_grad()
   loss.backward()
   optimizer.step()

  print(f"Epoch {i}/{epochs}: Loss {loss}")
```

```
Epoch 0/30: Loss 0.2329263538122177
Epoch 1/30: Loss 0.34837332367897034
Epoch 2/30: Loss 0.2433815449476242
Epoch 3/30: Loss 0.05776943266391754
Epoch 4/30: Loss 0.16537369787693024
Epoch 5/30: Loss 0.7609979510307312
Epoch 6/30: Loss 0.10551682114601135
Epoch 7/30: Loss 0.12570995092391968
Epoch 8/30: Loss 0.08270823210477829
Epoch 9/30: Loss 0.4433460533618927
Epoch 10/30: Loss 19.59803009033203
Epoch 11/30: Loss 0.11380017548799515
Epoch 12/30: Loss 0.2834673523902893
Epoch 13/30: Loss 0.08755698800086975
Epoch 14/30: Loss 0.11523362994194031
Epoch 15/30: Loss 0.19123423099517822
Epoch 16/30: Loss 0.5826622247695923
Epoch 17/30: Loss 0.9881596565246582
Epoch 18/30: Loss 0.24328851699829102
Epoch 19/30: Loss 0.8955906629562378
Epoch 20/30: Loss 73.48155212402344
Epoch 21/30: Loss 0.03496107831597328
Epoch 22/30: Loss 0.09457211196422577
Epoch 23/30: Loss 0.36414679884910583
Epoch 24/30: Loss 0.06265504658222198
Epoch 25/30: Loss 1.380377173423767
Epoch 26/30: Loss 0.2700571119785309
```

```
Epoch 27/30: Loss 0.14082761108875275
Epoch 28/30: Loss 0.07182713598012924
Epoch 29/30: Loss 3.421013355255127
```

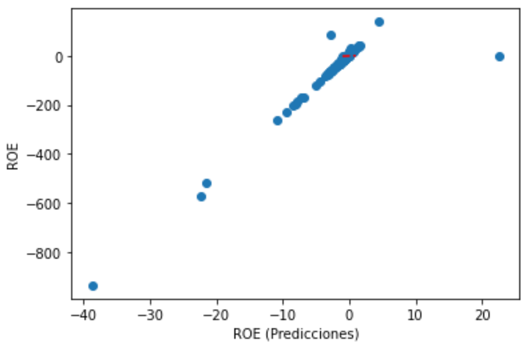
```
modelo.linear.weight #w
```

```
Parameter containing:
tensor([[ 0.4892, -0.0573, -0.1931]], requires_grad=True)
```

```
modelo.linear.bias #b
```

```
Parameter containing:
tensor([-0.2467], requires_grad=True)
```

```
# Evaluando el modelo
y_pred = []
y_true = []
modelo.train(False)
for inputs, targets in train_loader:
 y_pred.extend(modelo(inputs).data.numpy())
 y_true.extend(targets.numpy())
plt.scatter(y_pred, y_true)
plt.ylabel('ROE')
plt.xlabel('ROE (Predicciones)')
plt.plot([-1,1], [-1, 1], '--k', c='r')
plt.show()
# Calculando Errores
mae = mean_absolute_error(y_true=y_true, y_pred=y_pred)
mse = mean_squared_error(y_true=y_true, y_pred=y_pred, squared=True)
rmse = mean_squared_error(y_true=y_true, y_pred=y_pred, squared=False)
print(f"\nResultados Del año 2017 datos:")
print(f"MAE: {mae}")
print(f"MSE: {mse}")
print(f"RMSE: {rmse}")
```




```
Resultados Del año 2017 datos:
MAE: 1.0072892904281616
MSE: 128.7194061279297
RMSE: 11.345457077026367
```

EVALUANDO MODELO 2018

```
datos_2018.fillna(0)
datos_2018.dropna(inplace=True)
datos_2018
```

	AÑO	EXPEDIENTE	NOMBRE	RAMA	DESCRIPCIÓN RAMA	RAMA 6 DÍGITOS	SUBRAMA 2 DÍGITOS	LIQU CORRI
1	2018	2	ACERIA DEL ECUADOR CA ADELCA.	C	INDUSTRIAS MANUFACTURERAS.	C2410.25	C24	1.58
2	2018	3	ACERO COMERCIAL ECUATORIANO S.A.	G	COMERCIO AL POR MAYOR Y AL POR MENOR REPARACIÃ...	G4659.99	G46	3.08
4	2018	22	AGENCIAS Y REPRESENTACIONES CORDOVEZ SA	G	COMERCIO AL POR MAYOR Y AL POR MENOR REPARACIÃ...	G4630.95	G46	1.46
7	2018	49	ALMACENES EL GLOBO DE QUITO SA	G	COMERCIO AL POR MAYOR Y AL POR MENOR REPARACIÃ...	G4719.00	G47	1.44
8	2018	63	CONFITECA C.A.	C	INDUSTRIAS MANUFACTURERAS.	C1073.21	C10	1.76
...	...	...	...	...	...	...	...	
83339	2018	720498	OPENAUDIO S.A.	G	COMERCIO AL POR MAYOR Y AL POR MENOR REPARACIÃ...	G4759.06	G47	0.73
83409	2018	720576	BRADFORD GRILL BRADFORDGRILL	I	ACTIVIDADES DE ALOJAMIENTO Y DE	I5610.02	I56	0.29

```
dfs_rg_l = datos_2020[['ROE','RENTABILIDAD OPERACIONAL DEL PATRIMONIO', 'RENTABILIDAD NETA DE VENTAS']]
#Remover Outliers

dfs_rg_l
```

	ROE	RENTABILIDAD OPERACIONAL DEL PATRIMONIO	RENTABILIDAD NETA DE VENTAS
0	0.017822	0.079047	0.145618
2	-0.080518	-0.120850	-0.034202
4	0.093772	0.167538	0.033429
7	-0.985009	-0.901599	-0.217195
8	-0.057209	0.319549	-0.044041
...	...	...	...
83769	0.090264	-6.274013	0.012410
83825	-0.341922	0.310856	-0.036434
83830	0.899413	0.901391	0.326974
83880	-1.014521	-0.762914	-51.375614
84433	0.533966	49.896259	0.008106

14020 rows × 3 columns

```
dfs_Y = dfs_rg_l[['ROE']]
dfs_Y
```

	ROE
0	0.017822

```
dfs_X = dfs_rg_l[['ROE','RENTABILIDAD OPERACIONAL DEL PATRIMONIO', 'RENTABILIDAD NETA DE VENTAS']]
dfs_X
```

	ROE	RENTABILIDAD OPERACIONAL DEL PATRIMONIO	RENTABILIDAD NETA DE VENTAS
0	0.017822	0.079047	0.145618
2	-0.080518	-0.120850	-0.034202
4	0.093772	0.167538	0.033429
7	-0.985009	-0.901599	-0.217195
8	-0.057209	0.319549	-0.044041
...	...	...	...
83769	0.090264	-6.274013	0.012410
83825	-0.341922	0.310856	-0.036434
83830	0.899413	0.901391	0.326974
83880	-1.014521	-0.762914	-51.375614
84433	0.533966	49.896259	0.008106

14020 rows × 3 columns

```
scaler = StandardScaler()
inputs = scaler.fit_transform(dfs_X)
dX = np.array(inputs, dtype='float32')
dY = np.array(dfs_Y, dtype='float32')
```

```
dX
```

```
array([[ 0.04036301, -0.03196327,  0.0089452 ],
       [ 0.03204108, -0.03560445,  0.00878947],
       [ 0.04679025, -0.03035139,  0.00884804],
       ...,
       [ 0.11496707, -0.01698408,  0.00910226],
       [-0.04699835, -0.0472998 , -0.03567374],
       [ 0.08404137,  0.87546974,  0.00882611]], dtype=float32)
```

```
dY
```

```
array([[ 0.01782197],
       [-0.08051773],
       [ 0.09377224],
       ...,
       [ 0.8994128 ],
       [-1.014521  ],
       [ 0.53396606]], dtype=float32)
```

```
print('tamaño de dX INDICADORES : ',dX.shape) #tamaño de dX INDICADORES
print('tamaño de dY ROE : ',dY.shape) #tamaño de dY ROE
```

```
tamaño de dX INDICADORES :  (14020, 3)
tamaño de dY ROE :  (14020, 1)
```

```
X = torch.from_numpy(dX)
Y = torch.from_numpy(dY)
```

```
from torch.utils.data import TensorDataset
dataset = TensorDataset(X,Y)
```

```
dataset[1:2]
```

```
(tensor([[ 0.0320, -0.0356,  0.0088]]), tensor([[-0.0805]]))
```

```
from torch.utils.data import DataLoader
bs=64
train_loader = DataLoader(dataset,batch_size=bs,shuffle=True)
```

```
class ModeloRegresionLineal(torch.nn.Module):
 def __init__(self):
  super(ModeloRegresionLineal, self).__init__()
```

```
    self.linear = torch.nn.Linear(3,1) #X @ w.t() + b
   def forward(self, x):
    y_pred = self.linear(x)
    return y_pred

epochs = 30
ta = 1e-9 # Tasa Aprendizaje
modelo = ModeloRegresionLineal()
funcion_costo = torch.nn.MSELoss(reduction = 'mean')
optimizer = torch.optim.SGD(modelo.parameters(), lr = ta) #Actualiza los pesos w y el bias b

for i in range(epochs):
  for x,y in train_loader:
   preds = modelo(x)
   loss = funcion_costo(preds, y)
   optimizer.zero_grad()
   loss.backward()
   optimizer.step()

  print(f"Epoch {i}/{epochs}: Loss {loss}")
```

```
Epoch 0/30: Loss 0.03816503658890724
Epoch 1/30: Loss 0.028957800939679146
Epoch 2/30: Loss 15.449923515319824
Epoch 3/30: Loss 2.0858418941497803
Epoch 4/30: Loss 7.917113780975342
Epoch 5/30: Loss 0.012297725304961205
Epoch 6/30: Loss 1.0257222652435303
Epoch 7/30: Loss 0.009770125150680542
Epoch 8/30: Loss 7.122675895690918
Epoch 9/30: Loss 0.22992557287216187
Epoch 10/30: Loss 22.455156326293945
Epoch 11/30: Loss 2.2673659324645996
Epoch 12/30: Loss 0.7916465997695923
Epoch 13/30: Loss 3.1368720531463623
Epoch 14/30: Loss 0.010750258341431618
Epoch 15/30: Loss 0.2509259879589081
Epoch 16/30: Loss 0.052849724888801575
Epoch 17/30: Loss 0.13255004584789276
Epoch 18/30: Loss 0.004006273113191128
Epoch 19/30: Loss 0.12529690563678741
Epoch 20/30: Loss 0.2967700660228729
Epoch 21/30: Loss 0.1448555439710617
Epoch 22/30: Loss 0.23854151368141174
Epoch 23/30: Loss 0.03809897601604462
Epoch 24/30: Loss 0.08151651173830032
Epoch 25/30: Loss 0.31255555152893066
Epoch 26/30: Loss 0.034633819013834
Epoch 27/30: Loss 0.1523798406124115
Epoch 28/30: Loss 0.03502373397350311
Epoch 29/30: Loss 0.4350312054157257
```

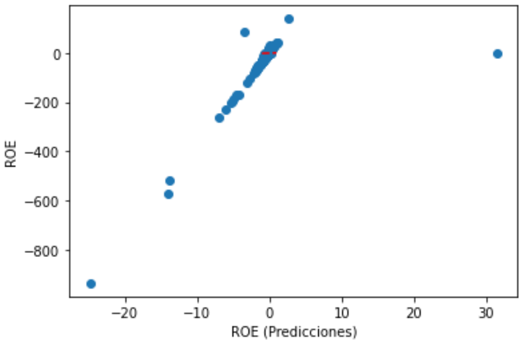
```
modelo.linear.weight #w
```

```
Parameter containing:
tensor([[ 0.3152, -0.0519, -0.2671]], requires_grad=True)
```

```
modelo.linear.bias #b
```

```
Parameter containing:
tensor([-0.0733], requires_grad=True)
```

```
# Evaluando el modelo
y_pred = []
y_true = []
modelo.train(False)
for inputs, targets in train_loader:
 y_pred.extend(modelo(inputs).data.numpy())
 y_true.extend(targets.numpy())
plt.scatter(y_pred, y_true)
plt.ylabel('ROE')
plt.xlabel('ROE (Predicciones)')
plt.plot([-1,1], [-1, 1], '--k', c='r')
plt.show()
# Calculando Errores
mae = mean_absolute_error(y_true=y_true, y_pred=y_pred)
mse = mean_squared_error(y_true=y_true, y_pred=y_pred, squared=True)
rmse = mean_squared_error(y_true=y_true, y_pred=y_pred, squared=False)
print(f"\nResultados Del año 2017 datos:")
print(f"MAE: {mae}")
print(f"MSE: {mse}")
print(f"RMSE: {rmse}")
```




```
Resultados Del año 2017 datos:
MAE: 0.916106104850769
MSE: 132.8055419921875
RMSE: 11.524128913879395
```

CONCLUSION

Segun los datos analizados podemos detectar que el año en el que mejor predice es el año del 2017 ya que nos da un error bien bajo por lo

0 s se ejecutó 01:30