ANÁLISIS DE ACCIONES

Martín Pérez Casais

Introducción

En este trabajo analizaré y prediceré el rendimiento futuro de NVIDIA, Apple y Meta Platforms.

En cuanto a la selección de variables, he decidido utilizar todas, ya que cualquiera de estas variables puede llegar a explicar la evolución de los rendimientos de nuestra empresa elegida. Exceptuando el total de activos, ROE, ROA, y apalancamiento de las otras empresas. Por ejemplo, si estoy prediciendo el rendimiento de NVIDIA, no voy a poner el apalancamiento financiero de Apple.

[4]:
```
# Importacion de librerias necesarias

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.svm import SVC
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, LSTM, Dense
```

[5]:
```
# Cargar e imprimir informacion esencial
df=pd.read_csv('final2.csv')
print(df.columns)
print(df.info())
print(df.describe())
```

```
Index(['Date', 'Rendimiento_NVIDIA', 'Rendimiento_Apple',
       'Rendimiento_Meta_Platforms', 'Rendimiento_VIX', 'Rendimiento_SP_500',
       'Rendimiento_DAX', 'Rendimiento_FTSE_100', 'Rendimiento_CAC_40',
       'Rendimiento_IBEX_35', 'Rendimineto_NIKKEI_225',
       'Rendimiento_NASDAQ_100', 'Redimiento_EUR_USD', 'Sentimiento',
       'Rendimiento_sentimiento', 'Año', 'Crecimiento_RN_NVIDIA',
       'Total_activos_NVIDIA', 'ROE_NVIDIA', 'ROA_NVIDIA',
       'Apalancamiento_NVIDIA', 'Crecimiento_RN_Apple', 'Total_activos_Apple',
```

```
       'ROE_Apple', 'ROA_Apple', 'Apalancamiento_Apple', 'Crecimiento_RN_Meta',
       'Total_activos_Meta', 'ROE_Meta', 'ROA_Meta', 'Apalancamiento_Meta'],
      dtype='object')
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1746 entries, 0 to 1745
Data columns (total 31 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Date                         1746 non-null   object
 1   Rendimiento_NVIDIA           1746 non-null   float64
 2   Rendimiento_Apple            1746 non-null   float64
 3   Rendimiento_Meta_Platforms   1746 non-null   float64
 4   Rendimiento_VIX              1746 non-null   float64
 5   Rendimiento_SP_500           1746 non-null   float64
 6   Rendimiento_DAX              1746 non-null   float64
 7   Rendimiento_FTSE_100         1746 non-null   float64
 8   Rendimiento_CAC_40           1746 non-null   float64
 9   Rendimiento_IBEX_35          1746 non-null   float64
 10  Rendimineto_NIKKEI_225       1746 non-null   float64
 11  Rendimiento_NASDAQ_100       1746 non-null   float64
 12  Redimiento_EUR_USD           1746 non-null   float64
 13  Sentimiento                  1746 non-null   float64
 14  Rendimiento_sentimiento      1745 non-null   float64
 15  Año                          1746 non-null   int64
 16  Crecimiento_RN_NVIDIA        1746 non-null   float64
 17  Total_activos_NVIDIA         1746 non-null   int64
 18  ROE_NVIDIA                   1746 non-null   float64
 19  ROA_NVIDIA                   1746 non-null   float64
 20  Apalancamiento_NVIDIA        1746 non-null   float64
 21  Crecimiento_RN_Apple         1746 non-null   float64
 22  Total_activos_Apple          1746 non-null   int64
 23  ROE_Apple                    1746 non-null   float64
 24  ROA_Apple                    1746 non-null   float64
 25  Apalancamiento_Apple         1746 non-null   float64
 26  Crecimiento_RN_Meta          1746 non-null   float64
 27  Total_activos_Meta           1746 non-null   int64
 28  ROE_Meta                     1746 non-null   float64
 29  ROA_Meta                     1746 non-null   float64
 30  Apalancamiento_Meta          1746 non-null   float64
dtypes: float64(26), int64(4), object(1)
memory usage: 423.0+ KB
None
       Rendimiento_NVIDIA  Rendimiento_Apple  Rendimiento_Meta_Platforms  \
count         1746.000000        1746.000000                 1746.000000
mean             0.001232           0.000377                    0.000322
std              0.014946           0.009096                    0.012042
min             -0.088515          -0.059808                   -0.133064
25%             -0.006642          -0.003846                   -0.005188
```

50%	0.001832	0.000497	0.000640
75%	0.010271	0.005457	0.006501
max	0.094686	0.049111	0.090901

	Rendimiento_VIX	Rendimiento_SP_500	Rendimiento_DAX	\
count	1746.000000	1746.000000	1746.000000	
mean	0.000451	0.000148	-0.000065	
std	0.033372	0.006096	0.006050	
min	-0.143613	-0.055439	-0.056697	
25%	-0.017786	-0.002061	-0.002162	
50%	-0.003038	0.000377	0.000342	
75%	0.015096	0.003136	0.002929	
max	0.217219	0.038949	0.045229	

	Rendimiento_FTSE_100	Rendimiento_CAC_40	Rendimiento_IBEX_35	\
count	1746.000000	1746.000000	1746.000000	
mean	-0.000023	0.000130	-0.000034	
std	0.004699	0.005648	0.005794	
min	-0.049998	-0.056885	-0.065801	
25%	-0.001930	-0.002251	-0.002796	
50%	0.000282	0.000330	0.000239	
75%	0.002284	0.002803	0.002936	
max	0.037639	0.034987	0.035722	

	Rendimineto_NIKKEI_225	…	Crecimiento_RN_Apple	Total_activos_Apple	\
count	1746.000000	…	1746.000000	1746.000000	
mean	0.000415	…	9.476334	348358.083620	
std	0.005895	…	12.760832	13411.943377	
min	-0.057475	…	-2.800000	323888.000000	
25%	-0.002744	…	2.020000	351002.000000	
50%	0.000458	…	5.510000	352583.000000	
75%	0.003810	…	7.790000	352755.000000	
max	0.042285	…	33.260000	364980.000000	

	ROE_Apple	ROA_Apple	Apalancamiento_Apple	Crecimiento_RN_Meta	\
count	1746.000000	1746.000000	1746.000000	1746.000000	
mean	144.666375	25.449387	5.890407	20.177715	
std	37.847279	4.230620	0.704907	12.934726	
min	73.690000	17.330000	4.960000	-1.120000	
25%	147.440000	26.130000	5.560000	15.690000	
50%	157.410000	27.500000	5.670000	21.600000	
75%	171.950000	28.060000	6.410000	26.610000	
max	175.460000	28.360000	6.960000	37.180000	

	Total_activos_Meta	ROE_Meta	ROA_Meta	Apalancamiento_Meta
count	1746.000000	1746.000000	1746.000000	1746.000000
mean	172442.237686	24.356804	18.417841	1.368511
std	30682.301154	4.906786	3.794559	0.098544

```
min          133376.000000    18.520000    13.190000              1.240000
25%          159316.000000    19.460000    16.020000              1.320000
50%          165987.000000    25.420000    18.830000              1.330000
75%          185727.000000    28.040000    19.920000              1.480000
max          229623.000000    31.100000    24.210000              1.500000

[8 rows x 30 columns]
```

[6]:
```
# Por si acaso tiene valores vacios le pongo el de la fila anterior
df.ffill(inplace=True)
```

Regresión lineal

Informe de NVIDIA

[7]:
```
# Defino las variables independientes en x y la variable a predecir en y
X_nvd=df[['Rendimiento_Apple',
       'Rendimiento_Meta_Platforms', 'Rendimiento_VIX', 'Rendimiento_SP_500',
       'Rendimiento_DAX', 'Rendimiento_FTSE_100', 'Rendimiento_CAC_40',
       'Rendimiento_IBEX_35', 'Rendimineto_NIKKEI_225',
       'Rendimiento_NASDAQ_100', 'Redimiento_EUR_USD', 'Sentimiento',
       'Rendimiento_sentimiento', 'Crecimiento_RN_NVIDIA',
       'Total_activos_NVIDIA', 'ROE_NVIDIA', 'ROA_NVIDIA',
       'Apalancamiento_NVIDIA']]
y_nvd=df["Rendimiento_NVIDIA"]
# Separo datos de entrenamiento y de prueba
X_train,X_test,y_train,y_test=train_test_split(X_nvd,y_nvd, test_size=0.3)
```

[8]:
```
# Normalizo las variables independientes, ya que pueden estar en escalas␣
 ↪diferentes, en el train utilizo fit para que los parametros se ajusten a␣
 ↪estos datos
scl=StandardScaler()
X_train_scl=scl.fit_transform(X_train)
X_test_scl=scl.transform(X_test)
#print(X_train.head())
#print(X_test.head())
```

[9]:
```
# Creo el modelo
lr=LinearRegression()

# Lo entreno con los datos de entrenamiento
lr.fit(X_train_scl, y_train)
```

[9]:
```
LinearRegression()
```

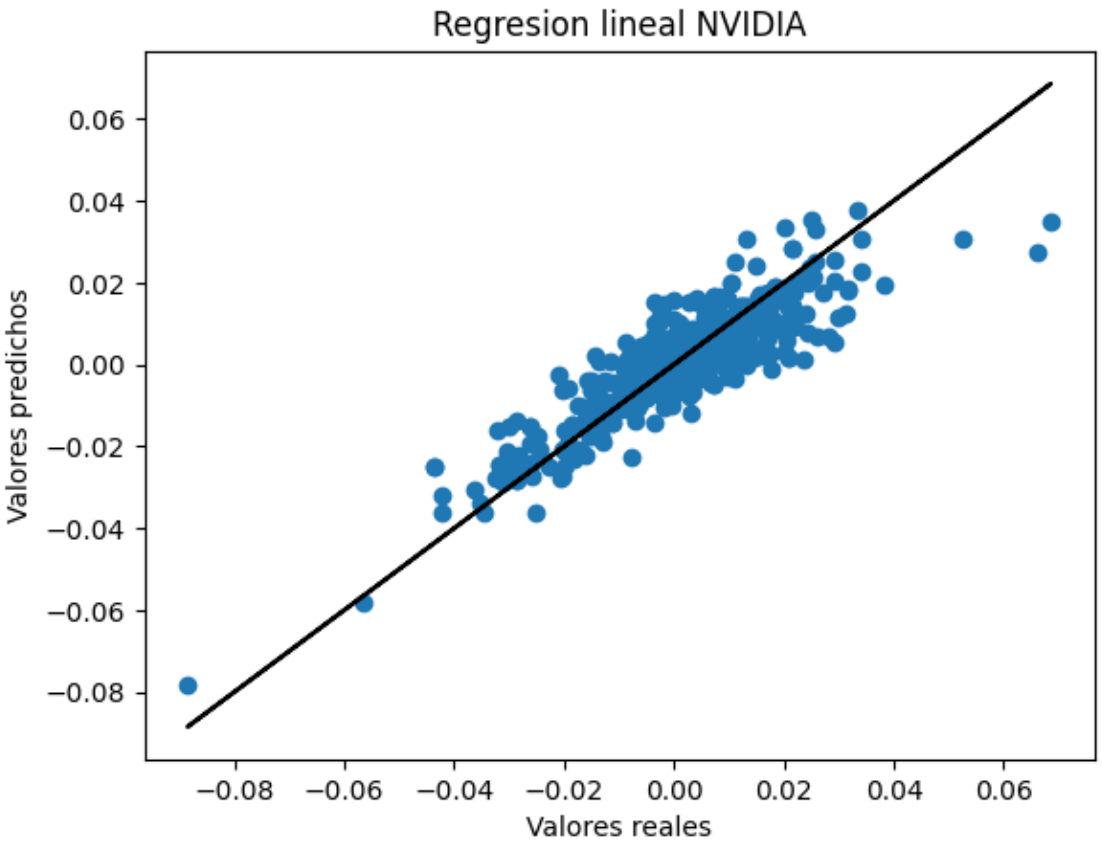
[10]:
```
# Hago la prediccion con los datos de prueba

y_pred=lr.predict(X_test_scl)
```

```
#print(y_pred)
```

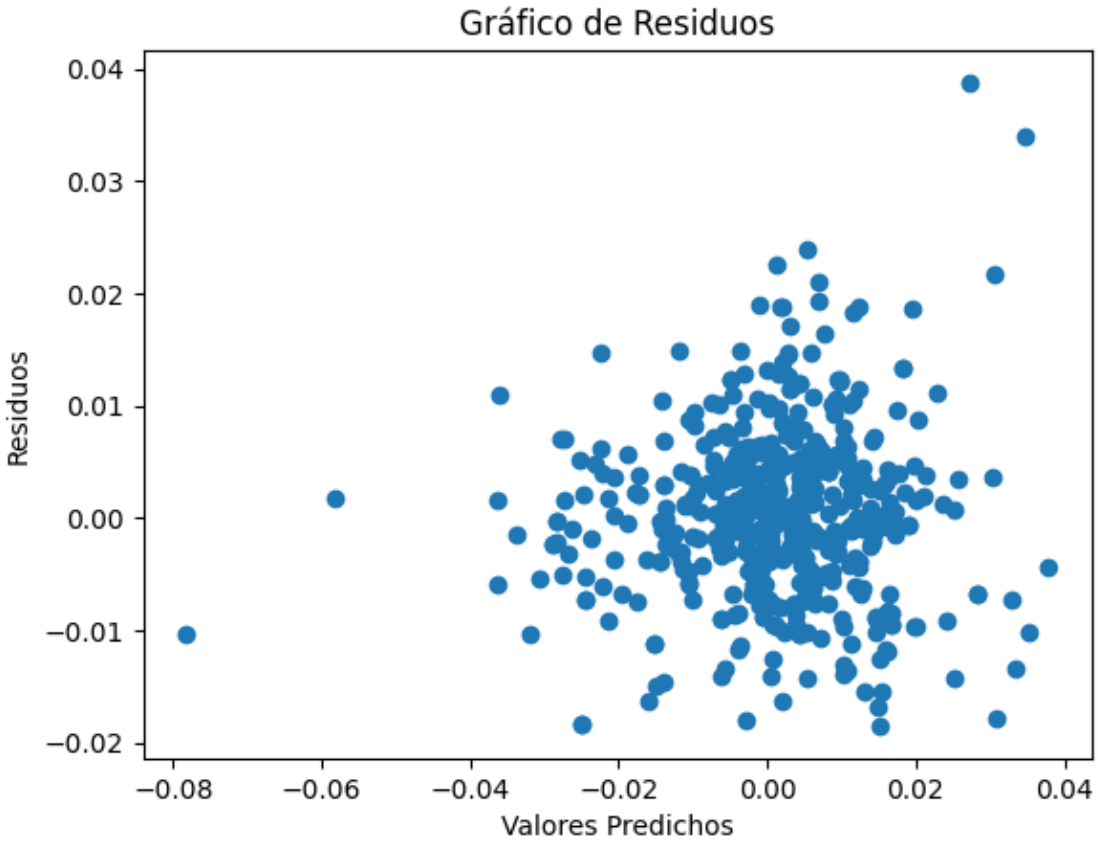
[11]:
```
# Calculo el RMSE y r cuarado
rmse=np.sqrt(mean_squared_error(y_test, y_pred))
r2=r2_score(y_test,y_pred)

print("RMSE: ",rmse)
print("R cuadrado: ",r2)
```

```
RMSE:  0.007637639293553116
R cuadrado:  0.7524940332857366
```

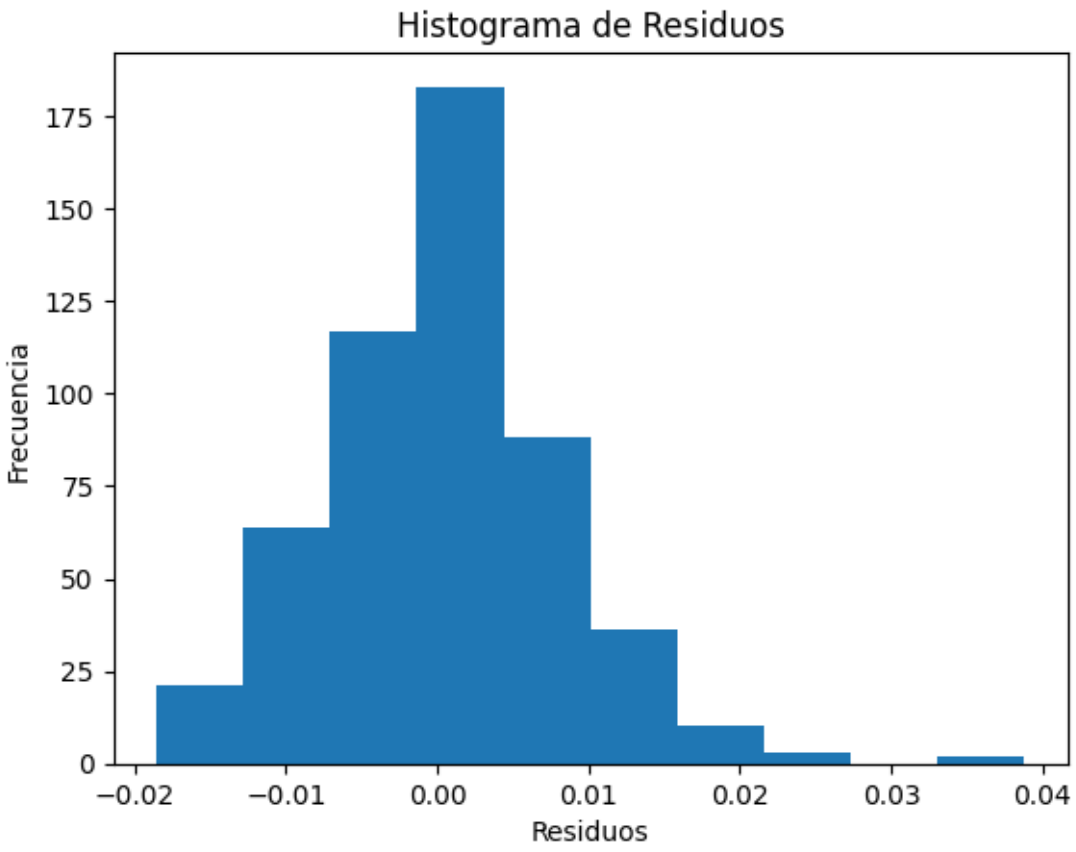
[12]:
```
# Un gráfico con una línea para indicar los valores reales y los puntos que␣
 ↪comparan los valores reales y predichos
plt.scatter(y_test, y_pred)
plt.plot(y_test,y_test, color='black')
plt.xlabel("Valores reales")
plt.ylabel("Valores predichos")
plt.title("Regresion lineal NVIDIA")
plt.show()
```



[13]:
```
# Calculo de residuos
res=y_test-y_pred
# Grafica de residuos
plt.scatter(y_pred, res)
plt.xlabel('Valores Predichos')
plt.ylabel('Residuos')
plt.title('Gráfico de Residuos')
plt.show()
```



[14]:
```
# Histograma de residuos
plt.hist(res)
plt.xlabel('Residuos')
plt.ylabel('Frecuencia')
plt.title('Histograma de Residuos')
plt.show()
```



En cuanto a los valores predichos contra los reales se puede observar que se tiendena a alinear bastante bien, ya que muchos muntos se aglomeran alrededor de la linea, aunque también hay cierta dispersión. El histograma de residuos, simétrica alrededor de 0 es buena señal, sin embargo, nos muestra cierta dispersion hacia el lado positivo.

El RMSE es bajo y r cuadrado nos indica que el 73% de la variabilidad del rendimiento lo explica nuestro modelo.

Informe de Apple

[15]:
```
# Defino las variables independientes en x y la variable a predecir en y
X_ap=df[['Rendimiento_NVIDIA',
       'Rendimiento_Meta_Platforms', 'Rendimiento_VIX', 'Rendimiento_SP_500',
       'Rendimiento_DAX', 'Rendimiento_FTSE_100', 'Rendimiento_CAC_40',
       'Rendimiento_IBEX_35', 'Rendimineto_NIKKEI_225',
       'Rendimiento_NASDAQ_100', 'Redimiento_EUR_USD', 'Sentimiento',
       'Rendimiento_sentimiento', 'Crecimiento_RN_Apple', 'Total_activos_Apple',
       'ROE_Apple', 'ROA_Apple', 'Apalancamiento_Apple']]
y_ap=df["Rendimiento_Apple"]
# Separo datos de entrenamiento y de prueba
```

```
X_train,X_test,y_train,y_test=train_test_split(X_ap,y_ap, test_size=0.3)
```

[16]:
```
# Normalizo las variables independientes, ya que pueden estar en escalas␣
↪diferentes, en el train utilizo fit para que los parametros se ajusten a␣
↪estos datos
scl=StandardScaler()
X_train_scl=scl.fit_transform(X_train)
X_test_scl=scl.transform(X_test)
#print(X_train.head())
#print(X_test.head())
```

[17]:
```
# Creo el modelo
lr=LinearRegression()

# Lo entreno con los datos de entrenamiento
lr.fit(X_train_scl, y_train)
```

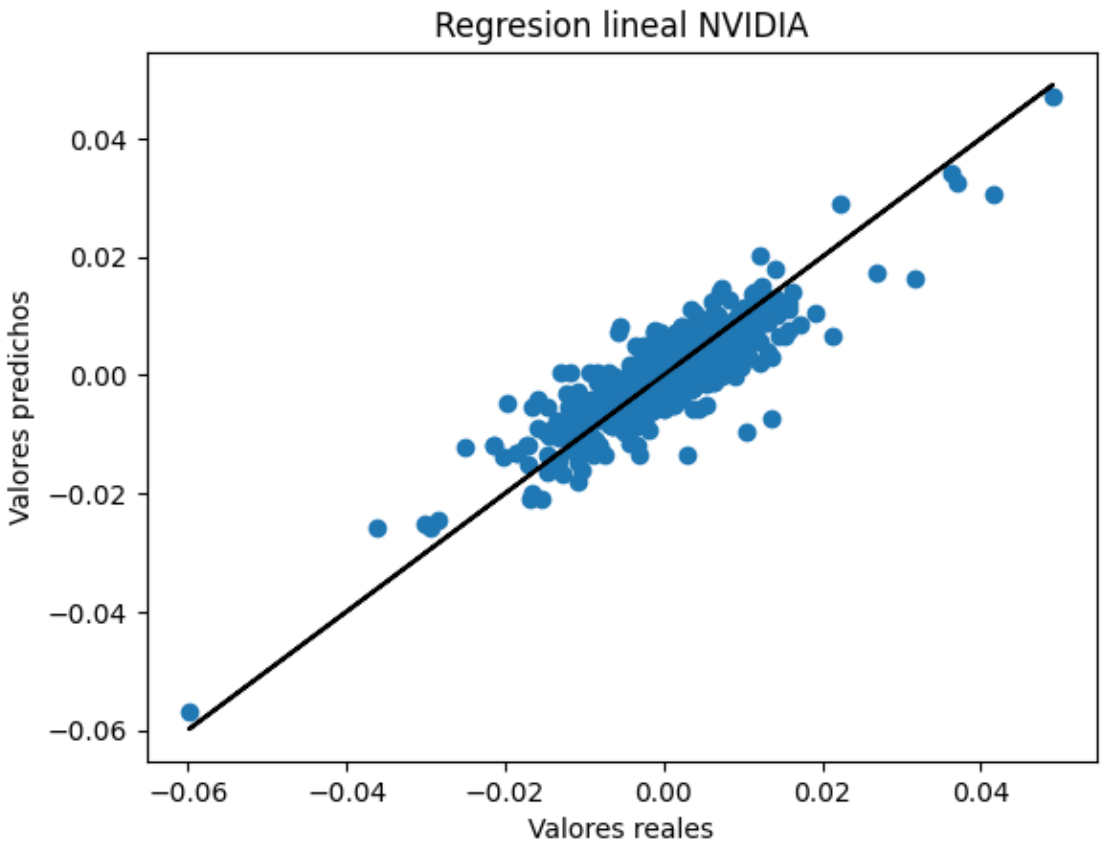
[17]:
```
LinearRegression()
```

[18]:
```
# Hago la prediccion con los datos de prueba

y_pred=lr.predict(X_test_scl)
#print(y_pred)
```

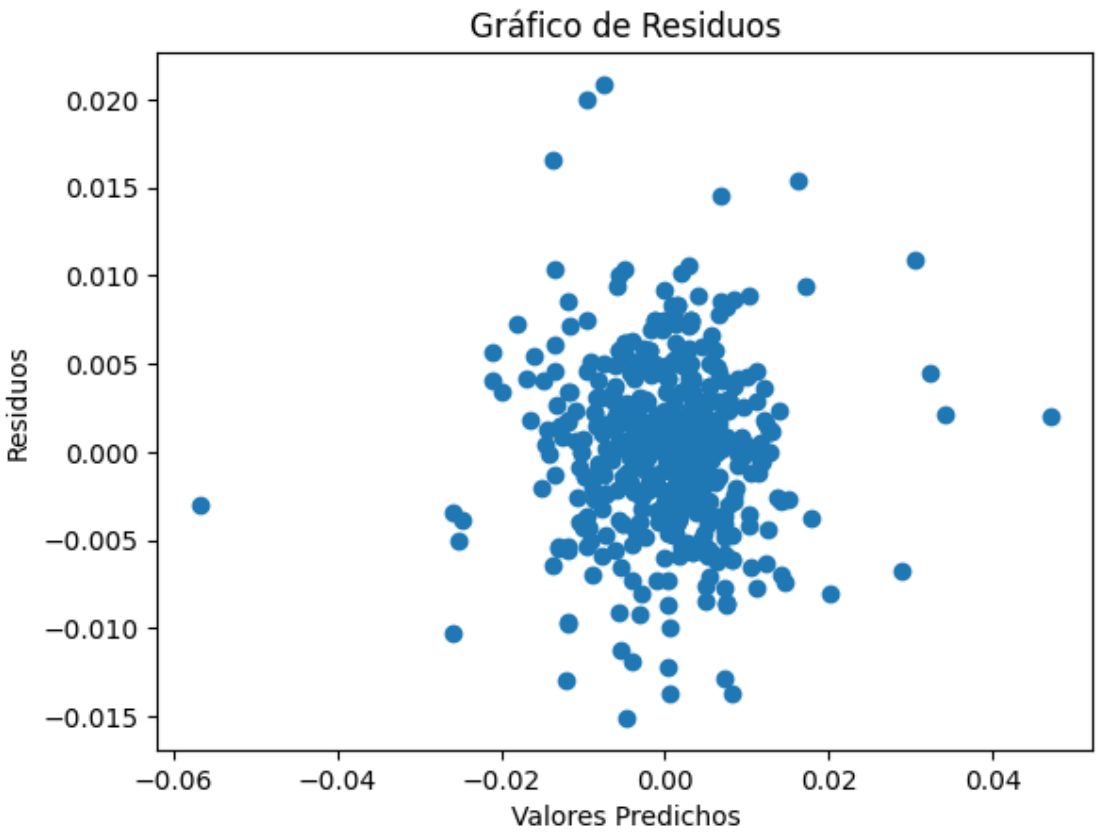
[19]:
```
# Calculo el RMSE y r cuarado
rmse=np.sqrt(mean_squared_error(y_test, y_pred))
r2=r2_score(y_test,y_pred)

print("RMSE: ",rmse)
print("R cuadrado: ",r2)
```

```
RMSE:  0.004661942118476681
R cuadrado:  0.7573862213157345
```

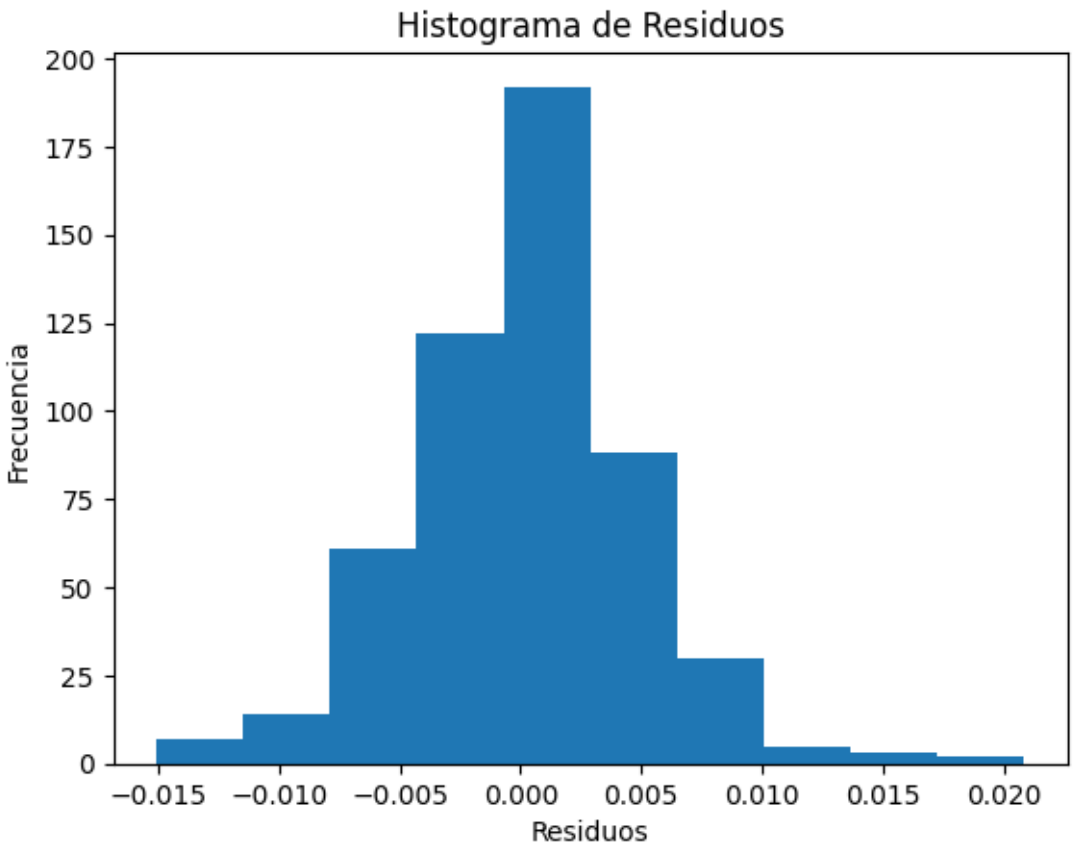
[20]:
```
# Un gráfico con una línea para indicar los valores reales y los puntos que␣
↪comparan los valores reales y predichos
plt.scatter(y_test, y_pred)
plt.plot(y_test,y_test, color='black')
plt.xlabel("Valores reales")
plt.ylabel("Valores predichos")
plt.title("Regresion lineal NVIDIA")
plt.show()
```



[21]:
```
# Calculo de residuos
res=y_test-y_pred
# Grafica de residuos
plt.scatter(y_pred, res)
plt.xlabel('Valores Predichos')
plt.ylabel('Residuos')
plt.title('Gráfico de Residuos')
plt.show()
```



[22]:
```
# Histograma de residuos
plt.hist(res)
plt.xlabel('Residuos')
plt.ylabel('Frecuencia')
plt.title('Histograma de Residuos')
plt.show()
```



Por un lado, el grafico de los valores predichos nos muestra que los puntos estan bien alineados con la linea ideal y la mayoría de puntos se concentra alrededor del origen. Los residuos estan en torno a 0, pero con cierta dispersion, tampoco parece haber un patron claro, lo que es un simbolo positivo.

El RMSE es bajo, por lo que la precision es buena. El r cuadrado indica que el 67% de la variabilidad de los rendimientos se explica por el modelo.

Informe de Meta

[23]:
```
# Defino las variables independientes en x y la variable a predecir en y
X_mt=df[[
        'Rendimiento_NVIDIA','Rendimiento_Apple', 'Rendimiento_VIX',␣
  ↪'Rendimiento_SP_500',
        'Rendimiento_DAX', 'Rendimiento_FTSE_100', 'Rendimiento_CAC_40',
        'Rendimiento_IBEX_35', 'Rendimineto_NIKKEI_225',
        'Rendimiento_NASDAQ_100', 'Redimiento_EUR_USD', 'Sentimiento',
        'Rendimiento_sentimiento',  'Crecimiento_RN_Meta',
        'Total_activos_Meta', 'ROE_Meta', 'ROA_Meta', 'Apalancamiento_Meta']]
y_mt=df["Rendimiento_Meta_Platforms"]
```

```
# Separo datos de entrenamiento y de prueba

X_train,X_test,y_train,y_test=train_test_split(X_mt,y_mt, test_size=0.3)
```

[24]:
```
# Normalizo las variables independientes, ya que pueden estar en escalas␣
↪diferentes, en el train utilizo fit para que los parametros se ajusten a␣
↪estos datos
scl=StandardScaler()
X_train_scl=scl.fit_transform(X_train)
X_test_scl=scl.transform(X_test)
#print(X_train.head())
#print(X_test.head())
```

[25]:
```
# Creo el modelo
lr=LinearRegression()

# Lo entreno con los datos de entrenamiento
lr.fit(X_train_scl, y_train)
```

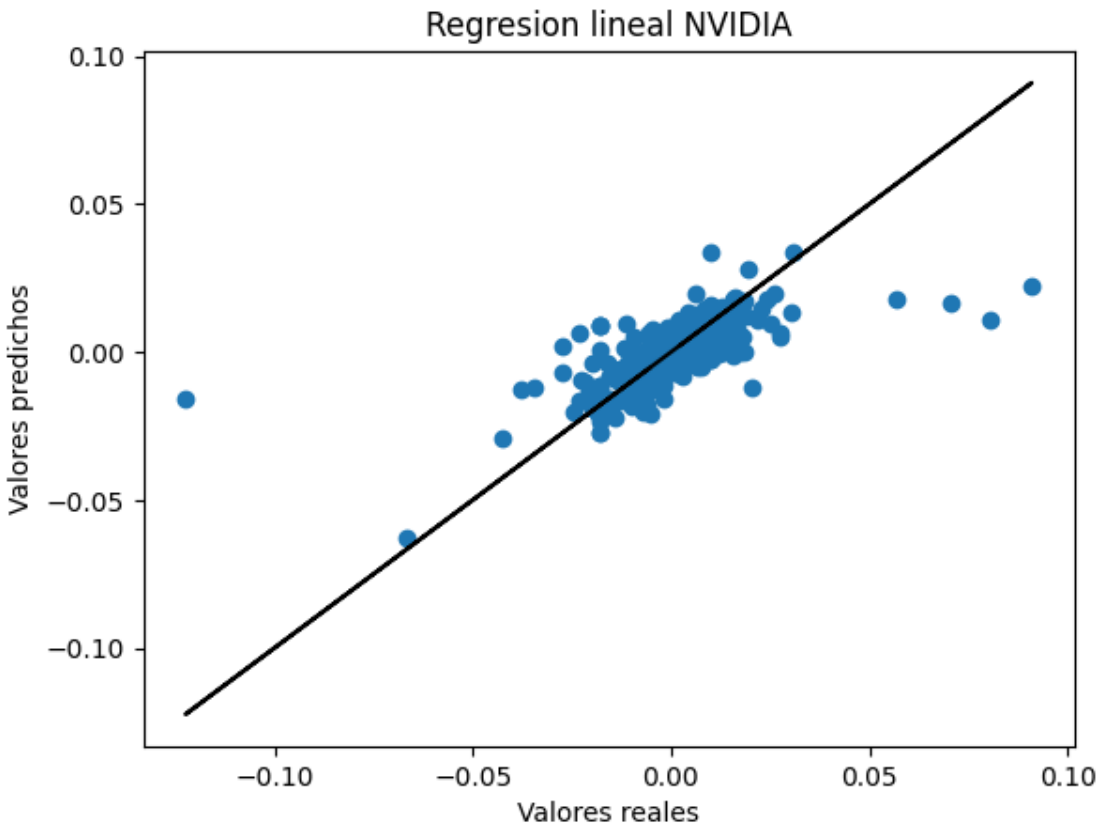
[25]:
```
LinearRegression()
```

[26]:
```
# Hago la prediccion con los datos de prueba

y_pred=lr.predict(X_test_scl)
#print(y_pred)
```

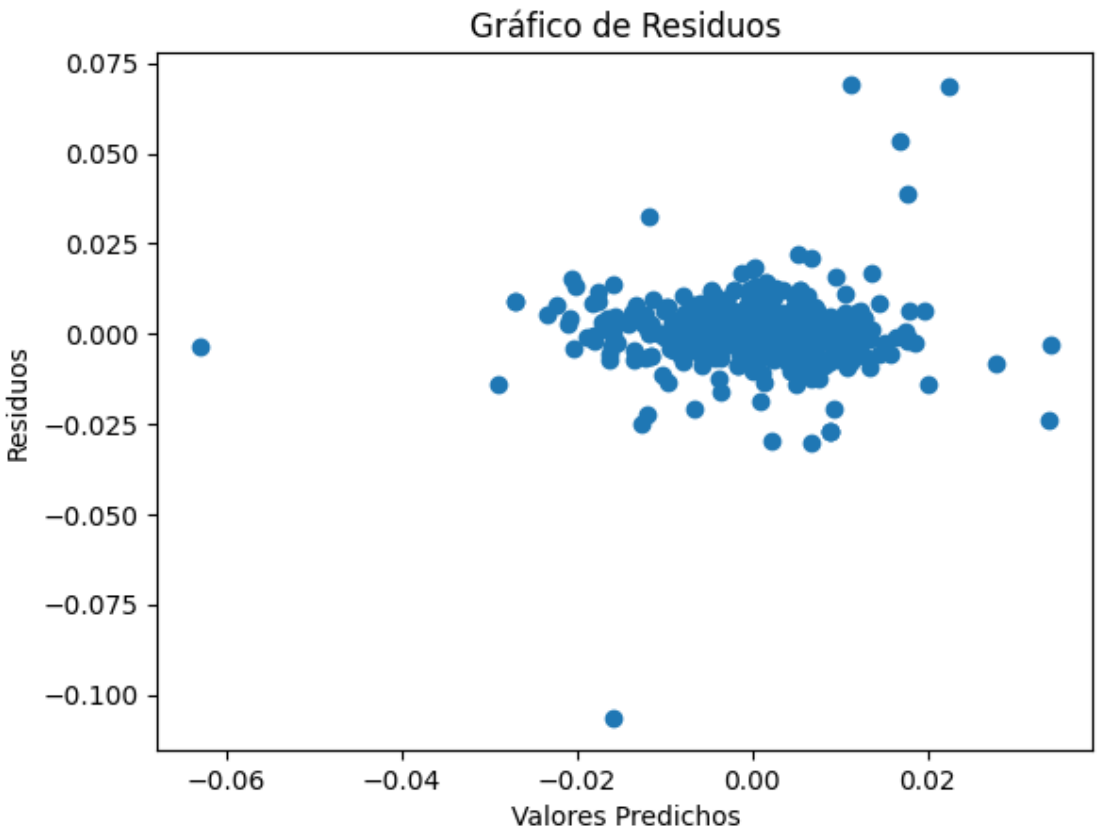
[27]:
```
# Calculo el RMSE y r cuarado
rmse=np.sqrt(mean_squared_error(y_test, y_pred))
r2=r2_score(y_test,y_pred)

print("RMSE: ",rmse)
print("R cuadrado: ",r2)
```

```
RMSE:  0.009844810448772497
R cuadrado:  0.4661707537339388
```

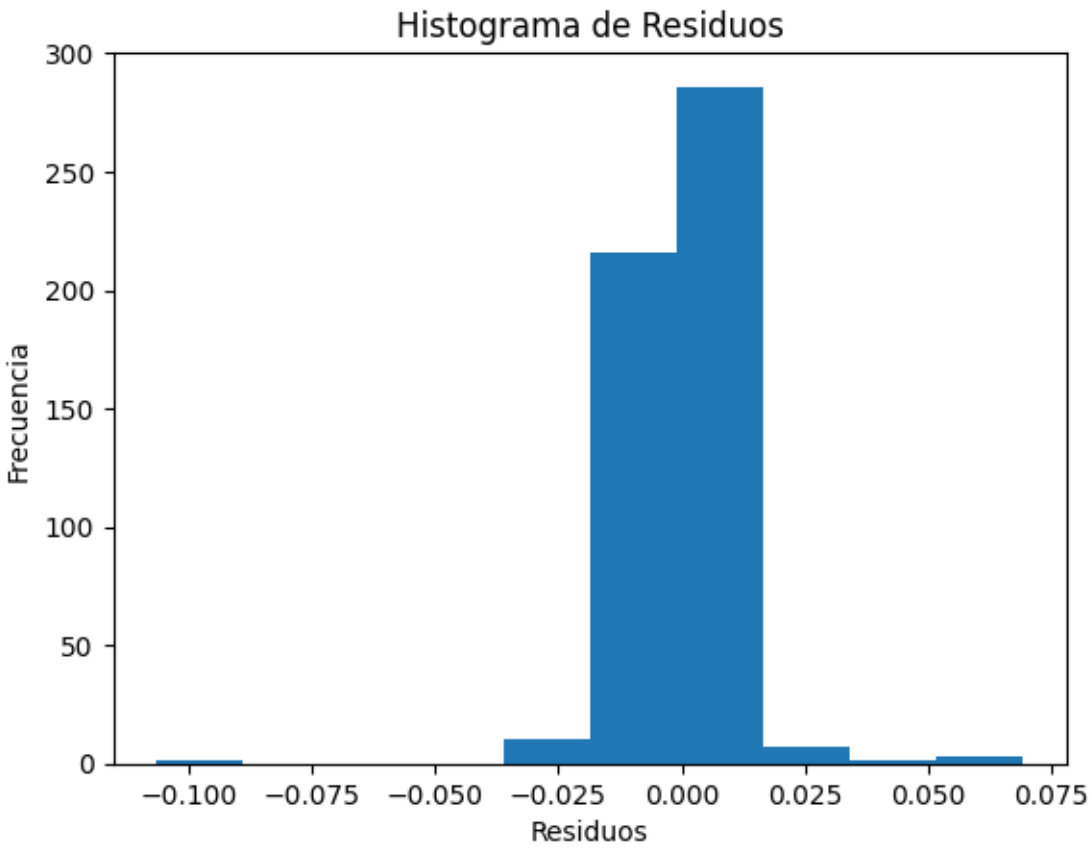
[28]:
```
# Un gráfico con una línea para indicar los valores reales y los puntos que␣
↪comparan los valores reales y predichos
plt.scatter(y_test, y_pred)
plt.plot(y_test,y_test, color='black')
plt.xlabel("Valores reales")
plt.ylabel("Valores predichos")
plt.title("Regresion lineal NVIDIA")
plt.show()
```



[29]:
```
# Calculo de residuos
res=y_test-y_pred
# Grafica de residuos
plt.scatter(y_pred, res)
plt.xlabel('Valores Predichos')
plt.ylabel('Residuos')
plt.title('Gráfico de Residuos')
plt.show()
```



[30]:
```
# Histograma de residuos
plt.hist(res)
plt.xlabel('Residuos')
plt.ylabel('Frecuencia')
plt.title('Histograma de Residuos')
plt.show()
```



En cuanto a los valores predichos contra los reales se puede ver una dispersion alrededor de la linea ideal, Los residuos estan dispersos en un rango mas amplio que en los modelos anterior. El RMSE es mas alto que en los anteriores y el r cuadrado indica que solo mas o menos la mitad de la variabilidad es explicada por el modelo.

Regresión logística

Para este analisis, transformaremos el rendimiento a predecir en una variable categorica binaria. El la precision se calculara como 1-(la media de los valores diferentes)

[32]:
```
# Transformar variables dependientes a binarias
df["Rendimiento_NVIDIA_bin"]=(df["Rendimiento_NVIDIA"]>0).astype(int)
df["Rendimiento_Apple_bin"]=(df["Rendimiento_Apple"]>0).astype(int)
df["Rendimiento_Meta_Platforms_bin"]=(df["Rendimiento_Meta_Platforms"]>0).
↪astype(int)
#print(df.head())
```

Informe de NVIDIA

[34]:
```
# Defino las variables independientes en x y la variable a predecir en y
X_nvd=df[['Rendimiento_Apple',
       'Rendimiento_Meta_Platforms', 'Rendimiento_VIX', 'Rendimiento_SP_500',
       'Rendimiento_DAX', 'Rendimiento_FTSE_100', 'Rendimiento_CAC_40',
       'Rendimiento_IBEX_35', 'Rendimineto_NIKKEI_225',
       'Rendimiento_NASDAQ_100', 'Redimiento_EUR_USD', 'Sentimiento',
       'Rendimiento_sentimiento', 'Crecimiento_RN_NVIDIA',
       'Total_activos_NVIDIA', 'ROE_NVIDIA', 'ROA_NVIDIA',
       'Apalancamiento_NVIDIA']]
y_nvd=df["Rendimiento_NVIDIA_bin"]
# Separo datos de entrenamiento y de prueba
X_train,X_test,y_train,y_test=train_test_split(X_nvd,y_nvd, test_size=0.3)
```

[35]:
```
# Normalizo las variables independientes, ya que pueden estar en escalas␣
↪diferentes, en el train utilizo fit para que los parametros se ajusten a␣
↪estos datos
scl=StandardScaler()
X_train_scl=scl.fit_transform(X_train)
X_test_scl=scl.transform(X_test)
#print(X_train.head())
#print(X_test.head())
```

[36]:
```
# Crear el modelo
lor=LogisticRegression()
# Entrenarlo
lor.fit(X_train_scl,y_train)
```

[36]:
```
LogisticRegression()
```

[37]:
```
# Realizo las predicciones
y_pred=lor.predict(X_test_scl)
```

[38]:
```
# Calculo de prediciones
# Como se calcula con sklearn
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
# Como se calculo en clase
accuracy_manual = 1-((y_pred != y_test).mean())
print(accuracy_manual)
# A partir de ahora lo hare con sklearn, ya que me parece mucho mas practico
```

```
0.8187022900763359
0.8187022900763359
```

[39]:
```
# Matriz de confusion
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

```
[[169  58]
 [ 37 260]]
```

Esto indica que el 82,6% de las predicciones del modelo son correctas aunque tiene bastantes falsos positivos

Informe de Apple

[40]:
```
# Defino las variables independientes en x y la variable a predecir en y
X_ap=df[['Rendimiento_NVIDIA',
       'Rendimiento_Meta_Platforms', 'Rendimiento_VIX', 'Rendimiento_SP_500',
       'Rendimiento_DAX', 'Rendimiento_FTSE_100', 'Rendimiento_CAC_40',
       'Rendimiento_IBEX_35', 'Rendimineto_NIKKEI_225',
       'Rendimiento_NASDAQ_100', 'Redimiento_EUR_USD', 'Sentimiento',
       'Rendimiento_sentimiento', 'Crecimiento_RN_Apple', 'Total_activos_Apple',
       'ROE_Apple', 'ROA_Apple', 'Apalancamiento_Apple']]
y_ap=df["Rendimiento_Apple_bin"]
# Separo datos de entrenamiento y de prueba

X_train,X_test,y_train,y_test=train_test_split(X_ap,y_ap, test_size=0.3)
```

[41]:
```
# Normalizo las variables independientes, ya que pueden estar en escalas␣
↪diferentes, en el train utilizo fit para que los parametros se ajusten a␣
↪estos datos
scl=StandardScaler()
X_train_scl=scl.fit_transform(X_train)
X_test_scl=scl.transform(X_test)
#print(X_train.head())
#print(X_test.head())
```

[42]:
```
# Crear el modelo
lor=LogisticRegression()
# Entrenarlo
lor.fit(X_train_scl,y_train)
```

[42]:
```
LogisticRegression()
```

[43]:
```
# Realizo las predicciones

y_pred=lor.predict(X_test_scl)
```

[44]:
```
# Calculo de prediciones
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
```

```
0.8244274809160306
```

[45]:
```
# Matriz de confusion
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

```
[[181  67]
 [ 25 251]]
```

La precision es buena pero no excelente. por otro lado, vuelve a haber bastantes falsos positivos.

Informe de Meta

[46]:
```
# Defino las variables independientes en x y la variable a predecir en y
X_mt=df[[
       'Rendimiento_NVIDIA','Rendimiento_Apple', 'Rendimiento_VIX',␣
 ↪'Rendimiento_SP_500',
       'Rendimiento_DAX', 'Rendimiento_FTSE_100', 'Rendimiento_CAC_40',
       'Rendimiento_IBEX_35', 'Rendimineto_NIKKEI_225',
       'Rendimiento_NASDAQ_100', 'Redimiento_EUR_USD', 'Sentimiento',
       'Rendimiento_sentimiento',  'Crecimiento_RN_Meta',
       'Total_activos_Meta', 'ROE_Meta', 'ROA_Meta', 'Apalancamiento_Meta']]
y_mt=df["Rendimiento_Meta_Platforms_bin"]
# Separo datos de entrenamiento y de prueba

X_train,X_test,y_train,y_test=train_test_split(X_mt,y_mt, test_size=0.3)
```

[47]:
```
# Normalizo las variables independientes, ya que pueden estar en escalas␣
 ↪diferentes, en el train utilizo fit para que los parametros se ajusten a␣
 ↪estos datos
scl=StandardScaler()
X_train_scl=scl.fit_transform(X_train)
X_test_scl=scl.transform(X_test)
#print(X_train.head())
#print(X_test.head())
```

[48]:
```
# Crear el modelo
lor=LogisticRegression()
# Entrenarlo
lor.fit(X_train_scl,y_train)
```

[48]:
```
LogisticRegression()
```

[49]:
```
# Realizo las predicciones

y_pred=lor.predict(X_test_scl)
```

[50]:
```
# Calculo de prediciones
accuracy = accuracy_score(y_test, y_pred)
```

```
print(accuracy)
```

```
0.7404580152671756
```

[51]:
```
# Matriz de confusion
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

```
[[163  72]
 [ 64 225]]
```

En este caso,a precision indica que el 77% de las predicciones son correctas. En cuanto a la matriz de confusion, hay una alta proporcion de falsos positivos de nuevo.

1.2 Arboles de decision

1.2.1 Informe de NVIDIA

[52]:
```
# Defino las variables independientes en x y la variable a predecir en y
X_nvd=df[['Rendimiento_Apple',
       'Rendimiento_Meta_Platforms', 'Rendimiento_VIX', 'Rendimiento_SP_500',
       'Rendimiento_DAX', 'Rendimiento_FTSE_100', 'Rendimiento_CAC_40',
       'Rendimiento_IBEX_35', 'Rendimineto_NIKKEI_225',
       'Rendimiento_NASDAQ_100', 'Redimiento_EUR_USD', 'Sentimiento',
       'Rendimiento_sentimiento', 'Crecimiento_RN_NVIDIA',
       'Total_activos_NVIDIA', 'ROE_NVIDIA', 'ROA_NVIDIA',
       'Apalancamiento_NVIDIA']]
y_nvd=df["Rendimiento_NVIDIA"]
# Separo datos de entrenamiento y de prueba
X_train,X_test,y_train,y_test=train_test_split(X_nvd,y_nvd, test_size=0.3)
```

[53]:
```
# Creo el modelo
rf = RandomForestRegressor()
# Lo entreno
rf.fit(X_train,y_train)
```

[53]:
```
RandomForestRegressor()
```

[54]:
```
# Hago la prediccion
y_pred=rf.predict(X_test)
```

[55]:
```
# Calcular RMSE
rmse=np.sqrt(mean_squared_error(y_test, y_pred))
print(rmse)
# Calcular r cuadrado
r2=r2_score(y_test,y_pred)
print(r2)
```

```
0.0071927454652336575
0.7942559933099408
```

1.2.2 El RMSE es bastante bajo, lo que indica una buena prediccion. En cuando a r cuadrado el resultado esta bastante bien, ya que el modelo explica casi el 80%de la variabilidad.

Informe de Apple

[56]:
```
# Defino las variables independientes en x y la variable a predecir en y
X_ap=df[['Rendimiento_NVIDIA',
       'Rendimiento_Meta_Platforms', 'Rendimiento_VIX', 'Rendimiento_SP_500',
       'Rendimiento_DAX', 'Rendimiento_FTSE_100', 'Rendimiento_CAC_40',
       'Rendimiento_IBEX_35', 'Rendimineto_NIKKEI_225',
       'Rendimiento_NASDAQ_100', 'Redimiento_EUR_USD', 'Sentimiento',
       'Rendimiento_sentimiento', 'Crecimiento_RN_Apple', 'Total_activos_Apple',
       'ROE_Apple', 'ROA_Apple', 'Apalancamiento_Apple']]
y_ap=df["Rendimiento_Apple"]
# Separo datos de entrenamiento y de prueba

X_train,X_test,y_train,y_test=train_test_split(X_ap,y_ap, test_size=0.3)
```

[57]:
```
# Creo el modelo
rf = RandomForestRegressor()
# Lo entreno
rf.fit(X_train,y_train)
```

[57]:
```
RandomForestRegressor()
```

[58]:
```
# Hago la prediccion
y_pred=rf.predict(X_test)
```

[59]:
```
# Calcular RMSE
rmse=np.sqrt(mean_squared_error(y_test, y_pred))
print(rmse)
# Calcular r cuadrado
r2=r2_score(y_test,y_pred)
print(r2)
```

```
0.003253477376107587
0.8614828530804757
```

EL RMSE es extremadamente bajo, lo que indica una altisima prediccion. En cuanto a r cuadrado, el modelo explica el 86% de la variabilidad.

Informe de Meta

[60]:
```
# Defino las variables independientes en x y la variable a predecir en y
X_mt=df[[
       'Rendimiento_NVIDIA','Rendimiento_Apple', 'Rendimiento_VIX',␣
 ↪'Rendimiento_SP_500',
       'Rendimiento_DAX', 'Rendimiento_FTSE_100', 'Rendimiento_CAC_40',
       'Rendimiento_IBEX_35', 'Rendimineto_NIKKEI_225',
       'Rendimiento_NASDAQ_100', 'Redimiento_EUR_USD', 'Sentimiento',
       'Rendimiento_sentimiento',  'Crecimiento_RN_Meta',
       'Total_activos_Meta', 'ROE_Meta', 'ROA_Meta', 'Apalancamiento_Meta']]
y_mt=df["Rendimiento_Meta_Platforms"]
# Separo datos de entrenamiento y de prueba

X_train,X_test,y_train,y_test=train_test_split(X_mt,y_mt, test_size=0.3)
```

[61]:
```
# Creo el modelo
rf = RandomForestRegressor()
# Lo entreno
rf.fit(X_train,y_train)
```

[61]:
```
RandomForestRegressor()
```

[62]:
```
# Hago la prediccion
y_pred=rf.predict(X_test)
```

[63]:
```
# Calcular RMSE
rmse=np.sqrt(mean_squared_error(y_test, y_pred))
print(rmse)
# Calcular r cuadrado
r2=r2_score(y_test,y_pred)
print(r2)
```

```
0.007121815810822666
0.6012839217802257
```

1.2.6 El RMSE, aunque no es muy alto, es mayor a las otras dos acciones. EL modelo explica el 50% de la variabilidad, mucho menos que en las anteriores.

SVM

Voy a utilizar el de clasifiacion, ya que considero que para este trabajo es suficiente saber si los rendimientos son positivos o negativos **Informe de NVIDIA**

[64]:
```
# Defino las variables independientes en x y la variable a predecir en y
X_nvd=df[['Rendimiento_Apple',
       'Rendimiento_Meta_Platforms', 'Rendimiento_VIX', 'Rendimiento_SP_500',
```

```
       'Rendimiento_DAX', 'Rendimiento_FTSE_100', 'Rendimiento_CAC_40',
       'Rendimiento_IBEX_35', 'Rendimineto_NIKKEI_225',
       'Rendimiento_NASDAQ_100', 'Redimiento_EUR_USD', 'Sentimiento',
       'Rendimiento_sentimiento', 'Crecimiento_RN_NVIDIA',
       'Total_activos_NVIDIA', 'ROE_NVIDIA', 'ROA_NVIDIA',
       'Apalancamiento_NVIDIA']]
y_nvd=df["Rendimiento_NVIDIA_bin"]
# Separo datos de entrenamiento y de prueba
X_train,X_test,y_train,y_test=train_test_split(X_nvd,y_nvd, test_size=0.3)
```

[65]:
```
# Normalizo las variables independientes, ya que pueden estar en escalas␣
 ↪diferentes, en el train utilizo fit para que los parametros se ajusten a␣
 ↪estos datos
scl=StandardScaler()
X_train_scl=scl.fit_transform(X_train)
X_test_scl=scl.transform(X_test)
#print(X_train.head(
#print(X_test.head(
```

[66]:
```
# Creo el modelo
svm=SVC()
# Lo entreno
svm.fit(X_train_scl,y_train)
```

[66]:
```
SVC()
```

[67]:
```
# Hago las predicciones
y_pred=svm.predict(X_test_scl)
```

[68]:
```
# Calculo precision
prec=accuracy_score(y_test,y_pred)
print(prec)
# Matriz de confusion
cm=confusion_matrix(y_test,y_pred)
print(cm)
```

```
0.8187022900763359
[[159  78]
 [ 17 270]]
```

La precision es bastante alta, aunque hay muchos falsos positivos.

Informe de Apple

[69]:
```
# Defino las variables independientes en x y la variable a predecir en y
X_ap=df[['Rendimiento_NVIDIA',
       'Rendimiento_Meta_Platforms', 'Rendimiento_VIX', 'Rendimiento_SP_500',
       'Rendimiento_DAX', 'Rendimiento_FTSE_100', 'Rendimiento_CAC_40',
```

```
       'Rendimiento_IBEX_35', 'Rendimineto_NIKKEI_225',
       'Rendimiento_NASDAQ_100', 'Redimiento_EUR_USD', 'Sentimiento',
       'Rendimiento_sentimiento', 'Crecimiento_RN_Apple', 'Total_activos_Apple',
       'ROE_Apple', 'ROA_Apple', 'Apalancamiento_Apple']]
y_ap=df["Rendimiento_Apple_bin"]
# Separo datos de entrenamiento y de prueba

X_train,X_test,y_train,y_test=train_test_split(X_ap,y_ap, test_size=0.3)
```

[70]:
```
# Normalizo las variables independientes, ya que pueden estar en escalas␣
↪diferentes, en el train utilizo fit para que los parametros se ajusten a␣
↪estos datos
scl=StandardScaler()
X_train_scl=scl.fit_transform(X_train)
X_test_scl=scl.transform(X_test)
#print(X_train.head(
#print(X_test.head(
```

[71]:
```
# Creo el modelo
svm=SVC()
# Lo entreno
svm.fit(X_train_scl,y_train)
```

[71]:
```
SVC()
```

[72]:
```
# Hago las predicciones
y_pred=svm.predict(X_test_scl)
```

[73]:
```
# Calculo precision
prec=accuracy_score(y_test,y_pred)
print(prec)
# Matriz de confusion
cm=confusion_matrix(y_test,y_pred)
print(cm)
```

```
0.8206106870229007
[[193  55]
 [ 39 237]]
```

Precision buena, aunque un poco inferior a la de NVIDIA, aunque hay bastatnes falsos positivos de nuevo.

Informe de Meta

[74]:
```
# Defino las variables independientes en x y la variable a predecir en y
X_mt=df[[
       'Rendimiento_NVIDIA','Rendimiento_Apple', 'Rendimiento_VIX',␣
↪'Rendimiento_SP_500',
```

```
       'Rendimiento_DAX', 'Rendimiento_FTSE_100', 'Rendimiento_CAC_40',
       'Rendimiento_IBEX_35', 'Rendimineto_NIKKEI_225',
       'Rendimiento_NASDAQ_100', 'Redimiento_EUR_USD', 'Sentimiento',
       'Rendimiento_sentimiento',  'Crecimiento_RN_Meta',
       'Total_activos_Meta', 'ROE_Meta', 'ROA_Meta', 'Apalancamiento_Meta']]
y_mt=df["Rendimiento_Meta_Platforms_bin"]
# Separo datos de entrenamiento y de prueba

X_train,X_test,y_train,y_test=train_test_split(X_mt,y_mt, test_size=0.3)
```

[75]:
```
# Normalizo las variables independientes, ya que pueden estar en escalas␣
↪diferentes, en el train utilizo fit para que los parametros se ajusten a␣
↪estos datos
scl=StandardScaler()
X_train_scl=scl.fit_transform(X_train)
X_test_scl=scl.transform(X_test)
#print(X_train.head(
#print(X_test.head(
```

[76]:
```
# Creo el modelo
svm=SVC()
# Lo entreno
svm.fit(X_train_scl,y_train)
```

[76]:
```
SVC()
```

[77]:
```
# Hago las predicciones
y_pred=svm.predict(X_test_scl)
```

[78]:
```
# Calculo precision
prec=accuracy_score(y_test,y_pred)
print(prec)
# Matriz de confusion
cm=confusion_matrix(y_test,y_pred)
print(cm)
```

```
0.7786259541984732
[[185  64]
 [ 52 223]]
```

La precision bastante buena. Por otro lado, vuelve a haber bastantes falsos positivos.

LSTM

Este es el que mas complicado, he tenido que revisar los apuntes de clase y varios foros.

Informe de NVIDIA

[79]:
```
# Defino las variables independientes en x y la variable a predecir en y
X_nvd=df[['Rendimiento_Apple',
       'Rendimiento_Meta_Platforms', 'Rendimiento_VIX', 'Rendimiento_SP_500',
       'Rendimiento_DAX', 'Rendimiento_FTSE_100', 'Rendimiento_CAC_40',
       'Rendimiento_IBEX_35', 'Rendimineto_NIKKEI_225',
       'Rendimiento_NASDAQ_100', 'Redimiento_EUR_USD', 'Sentimiento',
       'Rendimiento_sentimiento', 'Crecimiento_RN_NVIDIA',
       'Total_activos_NVIDIA', 'ROE_NVIDIA', 'ROA_NVIDIA',
       'Apalancamiento_NVIDIA']]
y_nvd=df["Rendimiento_NVIDIA"]
```

[80]:
```
# Normalizar los datos
scl=MinMaxScaler()
X_scl=scl.fit_transform(X_nvd)
```

[81]:
```
# Funcion para hacer secuencias de los datos, ya que segun he visto este modelo␣
 ↪lo requiere
def secu(X,y,pasos=10):
    # Listas vacias para cada variable
    X_s,y_s=[], []
    # Bucle para las secuencias
    for i in range(len(X)-pasos):
        # Para dividir las filas en secuencias
        X_s.append(X[i:i+pasos])
        # Asignarle un valor a cada secuencia
        y_s.append(y[i+pasos])
    return np.array(X_s), np.array(y_s)
```

[82]:
```
# Crear las secuencias con las variables
X_s,y_s=secu(X_scl,y_nvd)
#print(X_s)
#print(y_s)
```

[83]:
```
# Separo datos de entrenamiento y de prueba

X_train,X_test,y_train,y_test=train_test_split(X_s, y_s, test_size=0.3)
```

[84]:
```
m=Sequential([
    # Defino la entrada, 0 para numero de secuencias creadas, 1 para longitud␣
 ↪de secuencia y 2 para numero de caracteristicas
    Input(shape=(10, X_train.shape[2])),
    LSTM(10),
    Dense(1)

])
```

[85]:
```
# Compilo el modelo
m.compile(optimizer="adam", loss="mse")
```

[86]:
```
# Entreno al modelo
m.fit(X_train, y_train)
```

```
- loss:      38/38
             0.1721
```

[86]:
```
<keras.src.callbacks.history.History at 0x15f8c3caf60>
```

[87]:
```
# Miro la evaluacion del modelo, ya que MSE se pone en la compilacion del␣
 ↪modelo, por tanto el resultado sera el MSE
print(m.evaluate(X_test,y_test))
# Imprimir RMSE
print(np.sqrt(m.evaluate(X_test,y_test)))
```

```
- loss:      17/17
             0.0235
             0.024071764200925827
- loss:      17/17
             0.0235
             0.15515077892465068
```

[88]:
```
# Predecir los resultados
y_pred=m.predict(X_test)
#print(y_pred
```

```
17/17
```

[89]:
```
# Para ver el error en las predicciones
rmse=np.sqrt(mean_squared_error(y_test, y_pred))
print(rmse)
```

```
0.1551507800935155
```

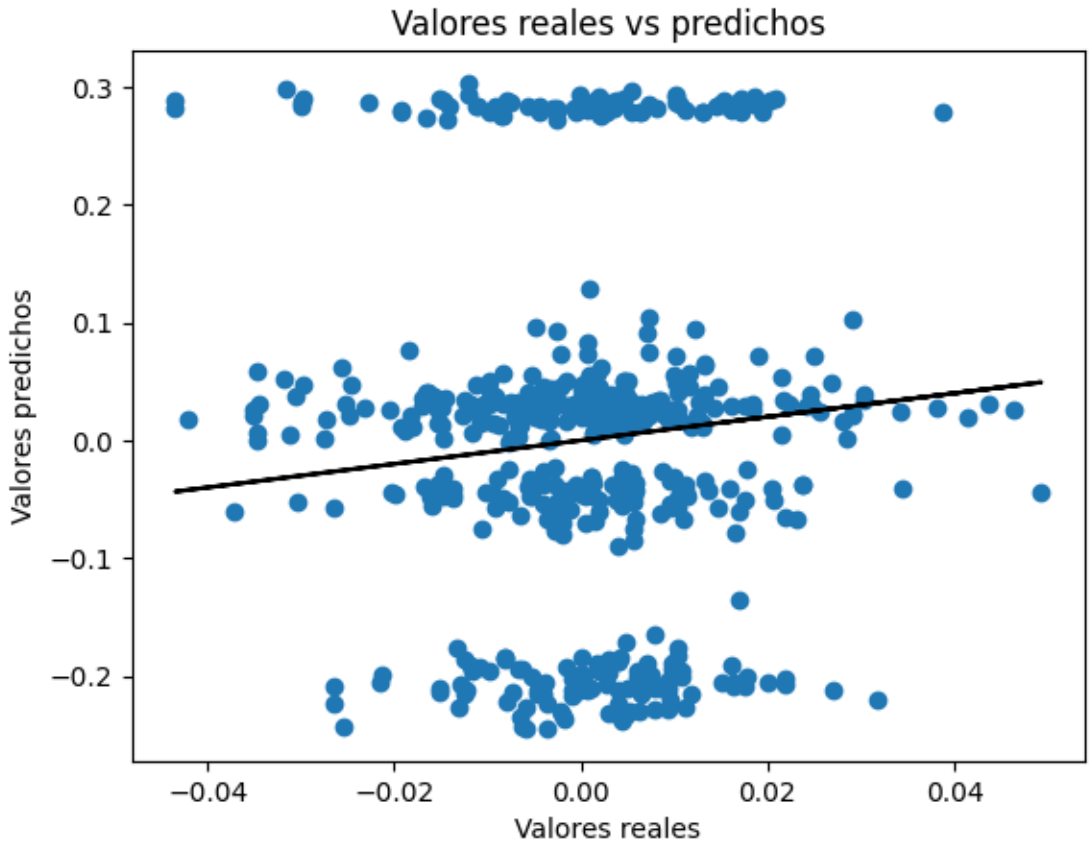
Como se puede observar, el RMSE es diferente que en evaluate ya que uno se hace durante el entrenamiento y otro con los datos de prueba.

[90]:
```
# Calcular r cuadrado
r2=r2_score(y_test,y_pred)
print(r2)
```

```
-120.93655559451061
```

[91]:
```
# Grafica de residuos
plt.scatter(y_test, y_pred)
plt.plot(y_test,y_test ,color='black')
plt.xlabel("Valores reales")
plt.ylabel("Valores predichos")
```

```
plt.title("Valores reales vs predichos")
plt.show()
```



El RMSE es bastante alto, el r2 negativo nos dice que el modelo no ha capturado ninguna relacion. En el gradico, los puntos estan demasiado dispersos.

Informe de Apple

[92]:
```
# Defino las variables independientes en x y la variable a predecir en y
X_ap=df[['Rendimiento_NVIDIA',
       'Rendimiento_Meta_Platforms', 'Rendimiento_VIX', 'Rendimiento_SP_500',
       'Rendimiento_DAX', 'Rendimiento_FTSE_100', 'Rendimiento_CAC_40',
       'Rendimiento_IBEX_35', 'Rendimineto_NIKKEI_225',
       'Rendimiento_NASDAQ_100', 'Redimiento_EUR_USD', 'Sentimiento',
       'Rendimiento_sentimiento', 'Crecimiento_RN_Apple', 'Total_activos_Apple',
       'ROE_Apple', 'ROA_Apple', 'Apalancamiento_Apple']]
y_ap=df["Rendimiento_Apple"]
```

[93]:
```
# Normalizar los datos
scl=MinMaxScaler()
```

```
X_scl=scl.fit_transform(X_ap)
```

[94]:
```
# Crear las secuencias con las variables
X_s,y_s=secu(X_scl,y_ap)
#print(X_s)
#print(y_s)
```

[95]:
```
# Separo datos de entrenamiento y de prueba

X_train,X_test,y_train,y_test=train_test_split(X_s, y_s, test_size=0.3)
```

[96]:
```
m=Sequential([
    # Defino la entrada, 0 para numero de secuencias creadas, 1 para longitud␣
 ↪de secuencia y 2 para numero de caracteristicas
    Input(shape=(10, X_train.shape[2])),
    LSTM(10),
    Dense(1)

])
```

[97]:
```
# Compilo el modelo
m.compile(optimizer="adam", loss="mse")
```

[98]:
```
# Entreno al modelo
m.fit(X_train, y_train)
```

```
38/38             4s 11ms/step -
loss: 0.0080
```

[98]:
```
<keras.src.callbacks.history.History at 0x15f8c9f14c0>
```

[99]:
```
# Miro la evaluacion del modelo, ya que MSE se pone en la compilacion del␣
 ↪modelo, por tanto el resultado sera el MSE
print(m.evaluate(X_test,y_test))
# Imprimir RMSE
print(np.sqrt(m.evaluate(X_test,y_test)))
```

```
17/17             1s 9ms/step - loss:
0.0014
0.0012503243051469326
17/17             0s 9ms/step - loss:
0.0014
0.035359925129260845
```

[100]:
```
# Predecir los resultados
y_pred=m.predict(X_test)
#print(y_pred)
```

```
17/17             1s 34ms/step
```

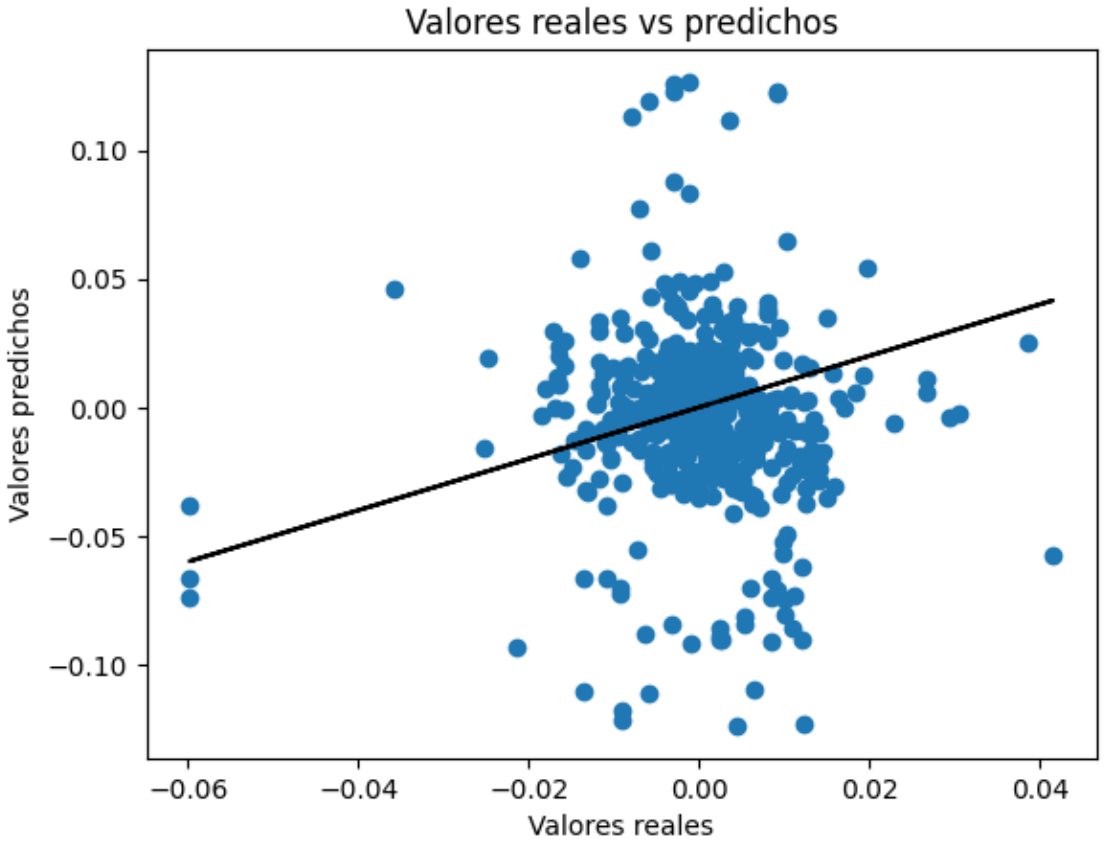
[101]:
```
# Para ver el error en las predicciones
rmse=np.sqrt(mean_squared_error(y_test, y_pred))
print(rmse)
```

```
0.03535992464635631
```

[102]:
```
# Calcular r cuadrado
r2=r2_score(y_test,y_pred)
print(r2)
```

```
-13.122236030299922
```

[103]:
```
# Grafica de residuos
plt.scatter(y_test, y_pred)
plt.plot(y_test,y_test ,color='black')
plt.xlabel("Valores reales")
plt.ylabel("Valores predichos")
plt.title("Valores reales vs predichos")
plt.show()
```



De nuevo el RMSE muy alto, r cuadrado extremadamente bajo y los puntos muy dispersos.

Informe de Meta

[104]:
```
# Defino las variables independientes en x y la variable a predecir en y
X_mt=df[[
       'Rendimiento_NVIDIA','Rendimiento_Apple', 'Rendimiento_VIX',␣
 ↪'Rendimiento_SP_500',
       'Rendimiento_DAX', 'Rendimiento_FTSE_100', 'Rendimiento_CAC_40',
       'Rendimiento_IBEX_35', 'Rendimineto_NIKKEI_225',
       'Rendimiento_NASDAQ_100', 'Redimiento_EUR_USD', 'Sentimiento',
       'Rendimiento_sentimiento',  'Crecimiento_RN_Meta',
       'Total_activos_Meta', 'ROE_Meta', 'ROA_Meta', 'Apalancamiento_Meta']]
y_mt=df["Rendimiento_Meta_Platforms"]
```

[105]:
```
# Normalizar los datos
scl=MinMaxScaler()
X_scl=scl.fit_transform(X_mt)
```

[106]:
```
# Crear las secuencias con las variables
X_s,y_s=secu(X_scl,y_mt)
#print(X_s)
#print(y_s)
```

[107]:
```
# Separo datos de entrenamiento y de prueba

X_train,X_test,y_train,y_test=train_test_split(X_s, y_s, test_size=0.3)
```

[108]:
```
m=Sequential([
    # Defino la entrada, 0 para numero de secuencias creadas, 1 para longitud␣
 ↪de secuencia y 2 para numero de caracteristicas
    Input(shape=(10, X_train.shape[2])),
    LSTM(10),
    Dense(1)

])
```

[109]:
```
# Compilo el modelo
m.compile(optimizer="adam", loss="mse")
```

[110]:
```
# Entreno al modelo
m.fit(X_train, y_train)
```

```
38/38            5s 13ms/step -
loss: 0.7648
```

[110]:
```
<keras.src.callbacks.history.History at 0x15f958984a0>
```

[111]:
```
# Miro la evaluacion del modelo, ya que MSE se pone en la compilacion del␣
 ↪modelo, por tanto el resultado sera el MSE
print(m.evaluate(X_test,y_test))
# Imprimir RMSE
print(np.sqrt(m.evaluate(X_test,y_test)))
```

```
17/17            1s 10ms/step -
loss: 0.0179
0.01767389290034771
17/17            0s 10ms/step -
loss: 0.0179
0.1329431942611118
```

[112]:
```
# Predecir los resultados
y_pred=m.predict(X_test)
#print(y_pred)
```

```
17/17            1s 20ms/step
```

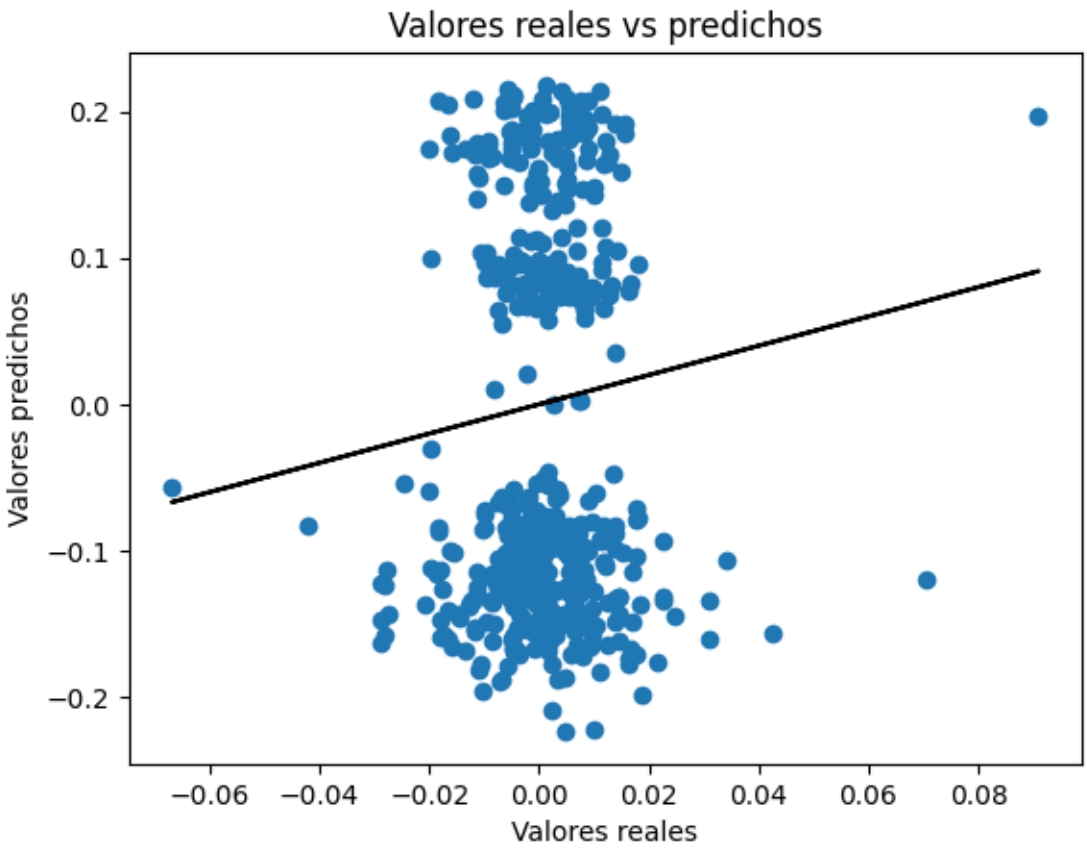
[113]:
```
# Para ver el error en las predicciones
rmse=np.sqrt(mean_squared_error(y_test, y_pred))
print(rmse)
```

```
0.13294319569034255
```

[114]:
```
# Calcular r cuadrado
r2=r2_score(y_test,y_pred)
print(r2)
```

```
-139.14685593499738
```

[115]:
```
# Grafica de residuos
plt.scatter(y_test, y_pred)
plt.plot(y_test,y_test ,color='black')
plt.xlabel("Valores reales")
plt.ylabel("Valores predichos")
plt.title("Valores reales vs predichos")
plt.show()
```



El RMSE extremandamente alto, el r cuadrado tan bajo nos dice que es hasta mejor predecir el promedio de los valores y los puntos estan muy dispersos.

Resultados obtenidos

Por lo tanto, los resultados obtenidos son los siguientes:

- Regresión lineal:
  - Nvidia:
    - RMSE: 0.007637639293553116
    - R2: 0.7524940332857366
  - Apple:
    - RMSE: 0.004661942118476681
    - R2: 0.7573862213157345
  - Meta:
    - RMSE: 0.009844810448772497
    - R2: 0.4661707537339388
- Regresión logística:
  - Nvidia:
    - Precisión: 0.8187022900763359
    - Matriz de confusión: [[169 58][37 260]]
  - Apple:
    - Precisión: 0.8244274809160306
    - Matriz de confusión: [[181 67][25 251]]
  - Meta:
    - Precisión: 0.7404580152671756
    - Matriz de confusión: [[163 72][64 225]]

- Árboles de decisión:
  - Nvidia:
    - RMSE: 0.0071927454652336575
    - R2: 0.7942559933099408
  - Apple:
    - RMSE: 0.003253477376107587
    - R2: 0.8614828530804757
  - Meta:
    - RMSE: 0.007121815810822666
    - R2: 0.6012839217802257
- SVM:
  - Nvidia:
    - Precisión: 0.8187022900763359
    - Matriz de confusión: [[159 78][17 270]]
  - Apple:
    - Precisión: 0.8206106870229007
    - Matriz de confusión: [[193 55][39 237]]
  - Meta:
    - Precisión: 0.7786259541984732
    - Matriz de confusión: [[185 64][52 223]]

- LSTM:
  - Nvidia:
    - RMSE: 0.1551507800935155
    - R2: -120.93655559451061
  - Apple:
    - RMSE: 0.03535992464635631
    - R2: -13.122236030299922
  - Meta:
    - RMSE: 0.13294319569034255
    - R2: -139.14685593499738

Conclusión final de inversión

Tras realizar todos estos análisis, se puede observar que los árboles de decisión es el modelo más adecuado para este trabajo, ya que es el modelo que tiene el RMSE más bajo, indicando que el error de las predicciones es muy bajo y, el R2 más alto, indicando que es el modelo que más explica la variabilidad de los rendimientos de las acciones.

En cuanto a la empresa que deberíamos elegir para invertir sería Apple, ya que dentro de los árboles de decisión, es la que tiene el RMSE más bajo y el R2 más alto. En referencia a porque no he escogido Meta o Nvidia para invertir, es porque el modelo muestra un desempeño muchísimo menor con estas acciones.

FIN