

01 DE DICIEMBRE DE 2022

# PRÁCTICA 9

Pronóstico - Regresión Lineal - Múltiple

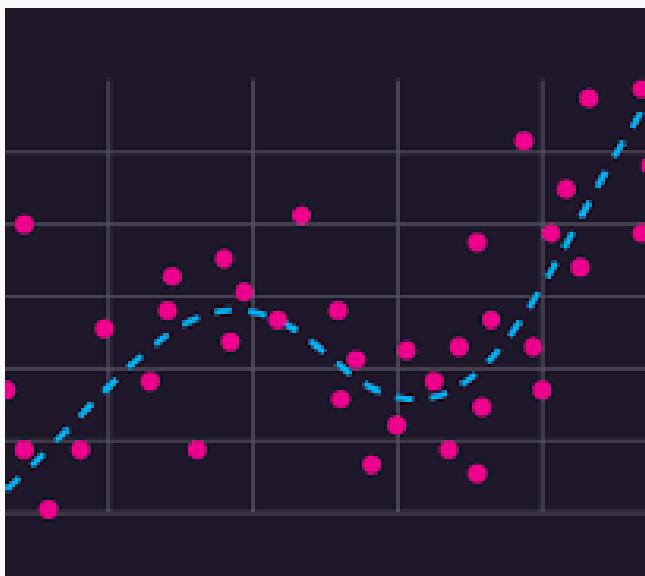


## Objetivo de la práctica

Por Angel Damian Monroy Mendoza

No. de Cuenta: 316040707

Generar un modelo de pronóstico de las acciones de una determinada empresa mexicana a través de un algoritmo de aprendizaje automático, en este caso, un modelo de regresión lineal.



## Características:

- Yahoo Finance ofrece una amplia variedad de datos de mercado sobre acciones, bonos, divisas y criptomonedas.
- También proporciona informes de noticias con varios puntos de vista sobre diferentes mercados de todo el mundo, todos accesibles a través de la biblioteca yfinance.
- En el comercio de acciones, 'alto' y 'bajo' se refieren a los precios máximos y mínimos en un período determinado.
- 'Apertura' y 'cierre' son los precios en los que una acción comenzó y terminó cotizando en el mismo periodo.
- El 'volumen' es la cantidad total de la actividad comercial.
- Los valores ajustados tienen en cuenta las acciones corporativas, como los 'dividendos', la 'división de acciones' y la emisión de nuevas acciones.

# Desarrollo

Para el desarrollo de esta práctica se obtuvieron dos modelos diferentes. En cada modelo se dividió el proceso en cinco secciones:

1. Importación de las bibliotecas necesarias y datos.
2. Descripción de la estructura de los datos.
3. Gráfica de los precios de las acciones.
4. Aplicación del algoritmo.
5. Nuevos pronósticos.

## 1. Bibliotecas y datos

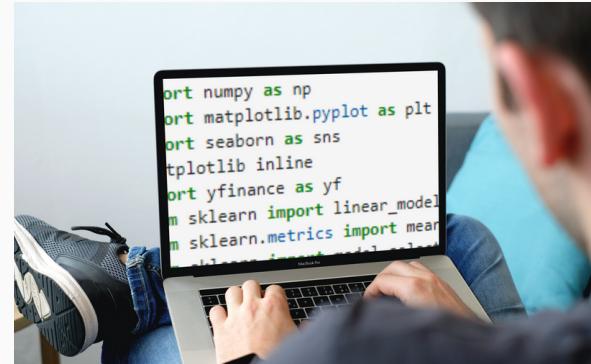
En este apartado se instalaron las bibliotecas necesarias para la manipulación de los datos, creación de vectores y matrices, generación de gráficas, la extracción de datos de yahoo finance y, posteriormente, en la sección de *Aplicación del Algoritmo*, se instalaron las bibliotecas para el *Modelo Lineal*, así como lo necesario para saber la eficiencia del modelo y su ajuste con *mean\_squared\_error*, *mean\_absolute\_error*, *r2\_score* y *model\_selection*

Vale la pena comentar un poco de la biblioteca de *yahoo finance* (*yfinance*), que nos permite utilizar datos financieros de empresas que cotizan en diversas bolsas de valores. Esta biblioteca nos permite obtener datos históricos de las acciones de una empresa, pues sólo basta con saber el 'Ticker' de la empresa que se desea analizar y colocarlo dentro del atributo que tiene *yfinance*.

En este caso se analiza la empresa *Bachoco* con *Ticker = 'IBA'* y luego, se obtiene el historial del precio de las acciones, en donde se definió el comienzo del periodo con el 01 - 01 - 2018 y un final del periodo el 20 - 11 - 2022, con un intervalo diario.

```
# Para Bachoco
DataBC = yf.Ticker('IBA') #Ticker es un atributo de yfinance

BCHist = DataBC.history(start = '2018-1-1', end = '2022-11-20', interval='1d')
BCHist
```



Date	Open	High	Low	Close	Volume	Dividends
2018-01-02 00:00:00-05:00	52.177076	52.896823	52.177076	52.896823	52.177076	0.000000
2018-01-03 00:00:00-05:00	52.896830	53.525468	52.896830	53.525468	52.896830	0.000000
2018-01-04 00:00:00-05:00	53.625687	53.698571	53.625687	53.698571	53.625687	0.000000
2018-01-05 00:00:00-05:00	52.751060	53.479917	52.751060	53.479917	52.751060	0.000000
2018-01-08 00:00:00-05:00	53.106379	53.379699	53.106379	53.379699	53.106379	0.000000
...	...	...	...	...	...	...
2022-11-14 00:00:00-05:00	50.000000	50.849998	49.500000	50.849998	50.000000	0.000000
2022-11-15 00:00:00-05:00	51.310001	53.490002	51.310001	53.490002	51.310001	0.000000
2022-11-16 00:00:00-05:00	53.529999	54.240002	53.529999	54.240002	53.529999	0.000000
2022-11-17 00:00:00-05:00	53.090000	53.110001	53.090000	53.110001	53.090000	0.000000
2022-11-18 00:00:00-05:00	52.750000	52.750000	52.750000	52.750000	52.750000	0.000000
1231 rows × 7 columns						

# Desarrollo

## 2. Descripción de la estructura de los datos.

En este apartado, se realizó un llamado al método de `.info()` para observar de una forma más condensada los tipos de datos del conjunto de datos, así como verificar que no existieran valores nulos.

Asimismo, se realizó una llamada al método `.describe()` con el fin de observar de una forma condensada algunas características de nuestros datos como: media, desviación, valor mínimo, valor máximo, percentil inferior (25%), 50% y percentil superior (75%).

En este punto logramos descubrir que las variables de `Dividends` y `Stock Splits` contienen valores nulos o vacíos, pues propiedades como el promedio o la desviación estándar tenían un valor de 0.

## 3. Gráfica de los precios de las acciones.

Aquí se graficaron las características de `Open`, `High`, `Low` & `Close`, debido a que el objetivo era observar cuál fue el comportamiento de los precios de las acciones de la empresa, en el periodo establecido.

Aquí observamos que las cuatro características tienen un comportamiento bastante similar, excepto por determinados momentos puntuales en el '`High`' que subió más que las demás variables.

Otro punto notable es que, al rededor de marzo de 2020, Bachoco sufrió una caída en el precio de sus acciones, ya que antes de esa fecha sus acciones rondaban los \$45 y posterior a ella cayó al rededor de los \$30, lo cual se explica indudablemente al efecto que tuvo la pandemia de Covid-19 en la sociedad.

A partir de este punto vamos a integrar las explicaciones de ambos modelos diferenciandolos por subtítulos o expresando explicitamente que la explicación corresponde para los dos.

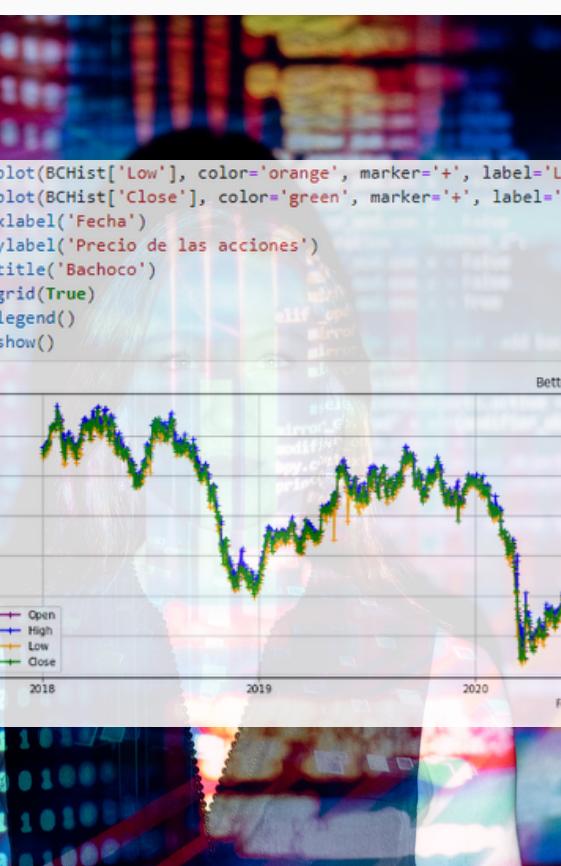
### 1er Modelo:

Se realizó una llamada al método `.drop` para eliminar las columnas que no formarían parte de nuestras variables independientes. En este caso se utilizaron las variables de: `Open`, `High`, `Low` & `Close`.

```
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype  
--- 
 0   Open            1231 non-null    float64
 1   High            1231 non-null    float64
 2   Low             1231 non-null    float64
 3   Close           1231 non-null    float64
 4   Volume          1231 non-null    int64  
 5   Dividends       1231 non-null    float64
 6   Stock Splits    1231 non-null    int64  
dtypes: float64(5), int64(2)
memory usage: 76.9 KB
```

```
[8]: BCHist.describe()
```

	Open	High	Low	Close	Volume	Dividends	Stock Splits
count	1231.000000	1231.000000	1231.000000	1231.000000	1231.000000	1231.0	1231.0
mean	44214651	44722657	43.648844	44.199783	10651.259139	0.03466	0.0
std	6.396712	6.400803	6.400375	6.425949	12007.035050	0.038567	0.0
min	27.072743	28.136618	26.404164	26.997410	600.000000	0.000000	0.0
25%	39.851068	40.389753	39.327840	39.686162	4400.000000	0.000000	0.0
50%	43.835125	44.258016	43.267871	43.812338	7200.000000	0.000000	0.0
75%	48.806268	49.201176	48.250345	48.787432	12450.000000	0.000000	0.0
max	58.208597	58.209471	57.519966	58.162327	168200.000000	0.489000	0.0



# Desarrollo

## 3. Gráfica de los precios de las acciones.

### 2do Modelo:

Se realizó una llamada al método `.drop` para eliminar las columnas que no formarían parte de nuestras variables independientes. En este caso se utilizaron las mismas variables del primer modelo más la variable de: **Volume**.

Por último, se realizó un `.dropna` con el fin de corroborar que no tuvieramos valores nulos en ambos modelos, pues cuando obtuvimos la información del `.describe()`, sólo pudimos afirmar que *Dividends* y *Stock Splits*, más no las demás variables.

```
X = np.array(MDatos[['Open',  
'High',  
'Low']])  
pd.DataFrame(X)
```

	0	1	2
0	52.177076	52.896823	52.177076
1	52.896830	53.525468	52.896830
2	53.625687	53.698571	52.486848

## 4. Aplicación del algoritmo.

Para este apartado se instalaron las bibliotecas necesarias para la aplicación del algoritmo de regresión lineal explicadas al principio de la práctica. Para ambos modelos se seleccionan las variables predictoras (X) y la variable a pronosticar (Y)

### 1er Modelo:

Variables predictoras: *Open*, *High* & *Low*

Variable a pronosticar: *Close*

### 2do Modelo:

Variables predictoras: *Open*, *High*, *Low* & *Volume*

Variable a pronosticar: *Close*

Después, para ambos modelos se hace la división de los datos utilizando la función de `model_selection` con el método `train_test_split`, en donde se le dan las variables predictoras y la variable a pronosticar; el tamaño del conjunto de testeo, el cual en este caso fue de 20% de prueba (p) y 80% de entrenamiento (e), recordando que es un parámetro que podemos modificar a nuestro criterio, pero que usualmente se deja en estas combinaciones:

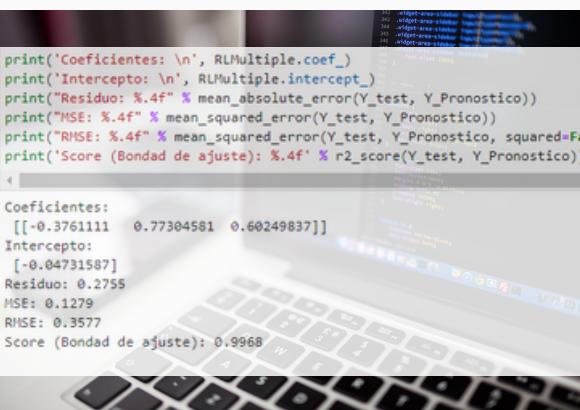
- 30% (p) y 70% (e).
- 25% (p) y 75% (e).
- 20% (p) y 80% (e).

```
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(  
X, Y, test_size=0.2)  
  
pd.DataFrame(X_test)
```

	0	1	2
0	57.707290	57.771065	56.869102
1	47.380722	50.862405	46.409961
2	49.051103	49.107604	47.318447
3	45.062460	46.021856	44.965553
4	39.160160	39.160160	38.250505

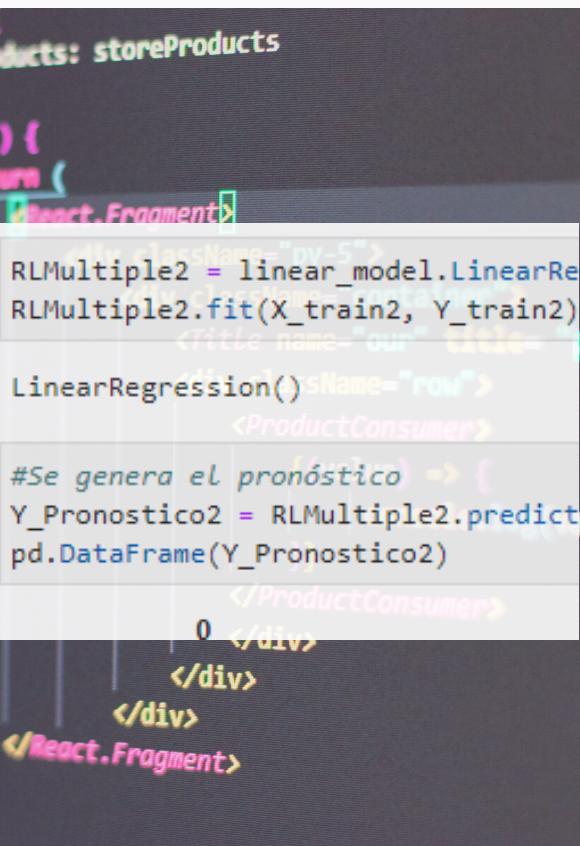
# Desarrollo

## 4. Aplicación del algoritmo



```
print('Coeficientes: \n', RLMultiple.coef_)
print('Intercepto: \n', RLMultiple.intercept_)
print("Residuo: %.4f" % mean_absolute_error(Y_test, Y_Pronostico))
print("MSE: %.4f" % mean_squared_error(Y_test, Y_Pronostico))
print("RMSE: %.4f" % mean_squared_error(Y_test, Y_Pronostico, squared=False))
print('Score (Bondad de ajuste): %.4f' % r2_score(Y_test, Y_Pronostico))
```

Coeficientes:  
[[-0.3761111 0.77304581 0.60249837]]  
Intercepto:  
[-0.04731587]  
Residuo: 0.2755  
MSE: 0.1279  
RMSE: 0.3577  
Score (Bondad de ajuste): 0.9968



```
RLMultiple2 = linear_model.LinearRegression()
RLMultiple2.fit(X_train2, Y_train2)

LinearRegression().fit(X_train2, Y_train2)
#Se genera el pronóstico >>>
Y_Pronostico2 = RLMultiple2.predict(X_train2)
pd.DataFrame(Y_Pronostico2)
```

RLMultiple2 = linear\_model.LinearRegression()  
RLMultiple2.fit(X\_train2, Y\_train2)  
  
LinearRegression().fit(X\_train2, Y\_train2)  
#Se genera el pronóstico >>>  
Y\_Pronostico2 = RLMultiple2.predict(X\_train2)  
pd.DataFrame(Y\_Pronostico2)

Asimismo, se agrego un estado random para volver reproducible los mismos datos obtenidos todas las veces que corramos nuestro código y un *shuffle* = 'True', que indica que los datos estén barajeados, es decir, que no agarre los primeros datos para el conjunto de entrenamiento y los últimos para el de prueba, sino que sea al azar la asignación de los datos de prueba y entrenamiento.

Posteriormente, en ambos modelos se entrena el modelo a través de Regresión Lineal Múltiple; esto quiere decir que, creamos un objeto para instanciar de la biblioteca de modelo lineal (*linear\_model*), la función de *LinearRegression()* y luego, a ese objeto se le ajustan los datos de entrenamiento.

A continuación, con estos modelos ya entrenados se generan los pronósticos con el método *.predict* y las variables predictoras del testeо o prueba y se imprimen en un DataFrame de pandas.

Por último, se obtienen las características necesarias para poder armar nuestros modelos de regresión lineal, así como para poder evaluar qué tan buenos son. En este sentido, se imprimen los *Coeficientes*, el *Intercepto*, el *Residuo*, el *Error Cuadrático Medio* (MSE), la *Raíz del Error Cuadrático Medio* (RMSE) y el *Score o Bondad de Ajuste*, en donde se obtuvieron los siguientes valores:

### 1er Modelo:

Coeficientes: [[-0.3761111 0.77304581 0.60249837]]  
Intercepto: [-0.04731587]  
Residuo: 0.2755  
MSE: 0.1279  
RMSE: 0.3577  
Score (Bondad de ajuste): 0.9968

### 2do Modelo:

Coeficientes: [[-3.7626e-01 7.6891e-01 6.0702e-01 7.7180e-07]]  
Intercepto: [-0.06100351]  
Residuo: 0.2751  
MSE: 0.1275  
RMSE: 0.3571  
Score (Bondad de ajuste): 0.9968

# Desarrollo

## 5. Nuevos pronósticos.

Por último, para ambos modelos se realizó un pronóstico de una acción ficticia con ciertos valores de entrada (variables predictoras) y con los modelos que habíamos guardado en los objetos de *RLMultiple* y *RLMultiple2*, respectivamente, se pronosticó la variable objetivo.

# Conclusiones

De esta práctica podemos concluir que los modelos de regresión lineal suelen ser muy eficientes ante conjunto de datos que se logran ajustar bien y el comportamiento de sus variables son parecidos. Asimismo, tal y como comentamos en clase, es importante reconocer que, a pesar de ser uno de los algoritmos más sencillos, realiza muy bien su trabajo, pues contamos con una *Bondad de Ajuste o Score* de 0.9968, el cual hace referencia a un resultado bastante notable con el que podemos trabajar sin pedirle nada a otros algoritmos.

Por último, se apreció la importancia de los algoritmos de regresión en general, pues cuando los datos no se logran ajustar a una línea, podemos utilizar algún otro tipo de regresión para probar si encajan de una mejor manera y evitarse la complejidad que tiene algún otro algoritmo más avanzado, pues entre más complejo sea un algoritmo, su interpretación se vuelve más rebuscada.

```
PrecioAccion2 = pd.DataFrame({'Open': [52.2],  
                             'High': [52.7],  
                             'Low': [52.5],  
                             'Volume': [168200]})  
  
RLMultiple2.predict(PrecioAccion2)  
  
C:\Users\Principal\anaconda3\lib\site-packages\sklearn\linear_model\__init__.py:101: UserWarning: This Ridge regression was fitted without feature names  
  warnings.warn(  
array([[52.81784709]])  
  
PrecioAccion = pd.DataFrame({'Open': [118.2],  
                             'High': [122.2],  
                             'Low': [110.8]})  
  
RLMultiple.predict(PrecioAccion)  
  
C:\Users\Principal\anaconda3\lib\site-packages\sklearn\linear_model\__init__.py:101: UserWarning: This Ridge regression was fitted without feature names  
  warnings.warn(  
array([[116.71936969]])
```

