Minería de Datos **Ibañez Guzman Osvaldo**
Grupo: 1 **Semestre 2023-1**
Dr. Guillermo Gilberto Molero Castillo **Práctica #13: Árboles de decisión y Bosques Aleatorios**

Objetivo

Hacer un pronóstico del precio de las acciones a través de un algoritmo de aprendizaje automático. Tanto con bosques aleatorios como de árboles de decisión, además de observar que ocurre si agregamos mas variables a nuestro entrenamiento.

Fuente de Información

Datos de la bolsa de valores GRUMA, desde 29/11/2017 hasta el 28/11/2022

https://finance.yahoo.com/quote/GRUMAB.MX?p=GRUMAB.MX&.tsrc=fin-srch

Fuente de Datos

- Date – Día de registro
- Open – Precios de las acciones al inicio del periodo
- High – Precios de las acciones más alto del periodo
- Low – Precios de las acciones más bajo del periodo
- Close – Precios de las acciones al final del periodo
- Adj Close – Precio tomando en cuenta cualquier cosa que pueda afectar después del cierre
- Volume – Cantidad total de la actividad comercial

Desarrollo

Lo primero a realizar es la importación de las bibliotecas que nos ayudarán a la manipulación, creación de matrices, gráficas y formas de visualizar los datos que usaremos, en este caso serán los datos relacionados a las acciones de la empresa GRUMA.

Estos datos se encuentran en el archivo llamado GRUMA.MX.csv, la cual usando la biblioteca pandas y la función *read_csv* obtendremos los datos y los colocaremos en una tabla para su manejo, dicha tabla se muestra a continuación.

```
GRUMAHist = pd.read_csv('GRUMAB.MX.csv')
GRUMAHist
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2017-11-29	237.410004	238.990005	232.009995	234.289993	207.512543	1463597
1	2017-11-30	235.460007	237.470001	230.110001	231.039993	204.633942	1411562
2	2017-12-01	231.039993	237.919998	231.039993	236.320007	209.310577	1426324
3	2017-12-04	233.009995	237.029999	233.009995	235.729996	208.787964	1097002
4	2017-12-05	235.729996	238.490005	235.179993	236.860001	209.788849	1191916
...	...	...	...	...	...	...	...
1252	2022-11-22	234.029999	234.449997	228.100006	228.669998	228.669998	587542
1253	2022-11-23	229.759995	239.759995	219.369995	239.130005	239.130005	1580655
1254	2022-11-24	237.509995	241.610001	237.509995	241.589996	241.589996	73583
1255	2022-11-25	240.979996	240.979996	232.949997	234.000000	234.000000	209348
1256	2022-11-28	234.050003	235.470001	232.009995	234.000000	234.000000	420175

1257 rows × 7 columns

Podemos observar un total de siete columnas, con 1257 registros, pero debemos modificar dicha tabla ya que no cuenta con un índice definido, podríamos solucionarlo de nos formas, agregando una nueva columna o utilizando una columna ya existente, para este análisis tomaremos la columna “Date” como nuestro índice, ya que son valores únicos y no los necesitamos para nuestros modelos. Usando la función *set_index* y colocando el nombre de la columna obtenemos dicho resultado.

```
GRUMAHist=GRUMAHist.set_index('Date')
GRUMAHist
```

Date	Open	High	Low	Close	Adj Close	Volume
2017-11-29	237.410004	238.990005	232.009995	234.289993	207.512543	1463597
2017-11-30	235.460007	237.470001	230.110001	231.039993	204.633942	1411562
2017-12-01	231.039993	237.919998	231.039993	236.320007	209.310577	1426324
2017-12-04	233.009995	237.029999	233.009995	235.729996	208.787964	1097002
2017-12-05	235.729996	238.490005	235.179993	236.860001	209.788849	1191916
...	...	...	...	...	...	...
2022-11-22	234.029999	234.449997	228.100006	228.669998	228.669998	587542
2022-11-23	229.759995	239.759995	219.369995	239.130005	239.130005	1580655
2022-11-24	237.509995	241.610001	237.509995	241.589996	241.589996	73583
2022-11-25	240.979996	240.979996	232.949997	234.000000	234.000000	209348
2022-11-28	234.050003	235.470001	232.009995	234.000000	234.000000	420175

1257 rows × 6 columns

Demos obtener la información relacionada a la tabla, en dicha información observamos que contamos con seis columnas, donde todos tiene n valores numéricos y “Volume” es la única de tipo entero mientras todas son flotantes, al igual que en ningún registro contamos con datos nulos.

```
GRUMAHist.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1257 entries, 2017-11-29 to 2022-11-28
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Open       1257 non-null   float64
 1   High       1257 non-null   float64
 2   Low        1257 non-null   float64
 3   Close      1257 non-null   float64
 4   Adj Close  1257 non-null   float64
 5   Volume     1257 non-null   int64  
dtypes: float64(5), int64(1)
memory usage: 68.7+ KB
```

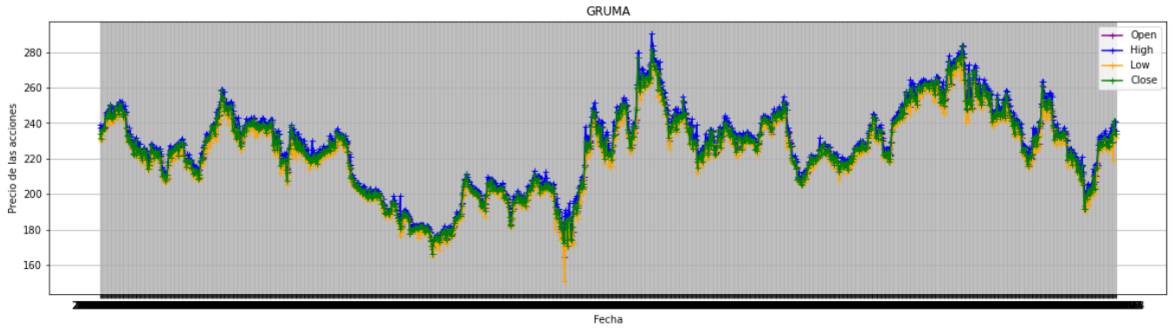
Además, podemos obtener sus valores estadísticos, como lo son el promedio, su desviación estándar, el mínimo y máximo de sus valores, así como sus cuartos percentiles, esto nos podría ayudar a observar si existen valores atípicos, por ejemplo, el valor mínimo de “Volume” es cero, esto podría ser un valor a típico, que podríamos considerar o indagar en el porqué de dicho valor.

```
GRUMAHist.describe()
```

	Open	High	Low	Close	Adj Close	Volume
count	1257.000000	1257.000000	1257.000000	1257.000000	1257.000000	1.257000e+03
mean	225.232864	228.207073	222.191965	225.223051	211.386217	8.022729e+05
std	22.767481	23.164466	22.355771	22.750871	24.833703	5.614090e+05
min	164.839996	170.080002	150.809998	166.070007	152.284058	0.000000e+00
25%	209.669998	212.660004	207.110001	208.940002	193.809875	4.467430e+05
50%	227.139999	230.149994	224.020004	226.979996	212.782990	6.509230e+05
75%	240.020004	243.000000	237.630005	240.119995	228.242767	1.013598e+06
max	283.859985	290.269989	276.399994	282.959991	274.924225	5.045875e+06

Graficaremos los datos de las acciones mas que nada, para entender el flujo de estos datos y ver como se relacionan entre ellos, podemos observar que en general todos se mueven de forma semejante, es de ir todos aumentan o disminuyen en los mismos momento, expecto en algunas ocasiones.

```
plt.figure(figsize=(20, 5))
plt.plot(GRUMAHist['Open'], color='purple', marker='+', label='Open')
plt.plot(GRUMAHist['High'], color='blue', marker='+', label='High')
plt.plot(GRUMAHist['Low'], color='orange', marker='+', label='Low')
plt.plot(GRUMAHist['Close'], color='green', marker='+', label='Close')
plt.xlabel('Fecha')
plt.ylabel('Precio de las acciones')
plt.title('GRUMA')
plt.grid(True)
plt.legend()
plt.show()
```



El siguiente paso es preparar los datos para la creación de nuestros modelos, para esto debemos tener dos conjuntos de datos, el primero eliminando las columnas de “Volume” y “Adj Close” el cual será nuestros modelos sin tomar en cuenta el valor del volumen de las acciones, y otro conjunto de datos únicamente eliminando la columna “Adj Close” donde si consideramos como variable predictora a “Close”

```
MDatos = GRUMAHist.drop(columns = ['Volume', 'Adj Close'])
MDatos
```

Date	Open	High	Low	Close
2017-11-29	237.410004	238.990005	232.009995	234.289993
2017-11-30	235.460007	237.470001	230.110001	231.039993
2017-12-01	231.039993	237.919998	231.039993	236.320007
2017-12-04	233.009995	237.029999	233.009995	235.729996
2017-12-05	235.729996	238.490005	235.179993	236.860001
...	...	...	...	...
2022-11-22	234.029999	234.449997	228.100006	228.669998
2022-11-23	229.759995	239.759995	219.369995	239.130005
2022-11-24	237.509995	241.610001	237.509995	241.589996
2022-11-25	240.979996	240.979996	232.949997	234.000000

```
MDatosVol = GRUMAHist.drop(columns = ['Adj Close'])
MDatosVol
```

Date	Open	High	Low	Close	Volume
2017-11-29	237.410004	238.990005	232.009995	234.289993	1463597
2017-11-30	235.460007	237.470001	230.110001	231.039993	1411562
2017-12-01	231.039993	237.919998	231.039993	236.320007	1426324
2017-12-04	233.009995	237.029999	233.009995	235.729996	1097002
2017-12-05	235.729996	238.490005	235.179993	236.860001	1191916
...	...	...	...	...	...
2022-11-22	234.029999	234.449997	228.100006	228.669998	587542
2022-11-23	229.759995	239.759995	219.369995	239.130005	1580655
2022-11-24	237.509995	241.610001	237.509995	241.589996	73583
2022-11-25	240.979996	240.979996	232.949997	234.000000	209348
2022-11-28	234.050003	235.470001	232.009995	234.000000	420175

1257 rows × 5 columns

Deberemos importar varias bibliotecas, para la creación de los árboles de decisión y los bosques aleatorios, ambos serán de regresión ya que buscamos obtener un valor y no una clasificación, a continuación, se muestran las bibliotecas a utilizar.

```
from sklearn import model_selection
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, accuracy_score
from sklearn.tree import DecisionTreeRegressor
```

Tanto para arboles de decisión debemos separar nuestras variables predictoras (X) y nuestra variable a pronosticar (Y), para nuestros podemos tendremos dos arreglos de variables predictoras ya que para consideramos en uno tres variable y en otro agregamos el volumen de las acciones.

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

	0	1	2
0	237.410004	238.990005	232.009995
1	235.460007	237.470001	230.110001
2	231.039993	237.919998	231.039993
3	233.009995	237.029999	233.009995
4	235.729996	238.490005	235.179993
...	...	...	...
1252	234.029999	234.449997	228.100006
1253	229.759995	239.759995	219.369995
1254	237.509995	241.610001	237.509995
1255	240.979996	240.979996	232.949997
1256	234.050003	235.470001	232.009995

1257 rows × 3 columns

```
XVol = np.array(MDatosVol[['Open',
                           'High',
                           'Low',
                           'Volume']])
pd.DataFrame(XVol)
```

	0	1	2	3
0	237.410004	238.990005	232.009995	1463597.0
1	235.460007	237.470001	230.110001	1411562.0
2	231.039993	237.919998	231.039993	1426324.0
3	233.009995	237.029999	233.009995	1097002.0
4	235.729996	238.490005	235.179993	1191916.0
...	...	...	...	...
1252	234.029999	234.449997	228.100006	587542.0
1253	229.759995	239.759995	219.369995	1580655.0
1254	237.509995	241.610001	237.509995	73583.0
1255	240.979996	240.979996	232.949997	209348.0
1256	234.050003	235.470001	232.009995	420175.0

1257 rows × 4 columns

Y la variable a pronosticar será la misma para todos los modelos, es decir buscamos pronosticar la variable “Close”.

```
Y = np.array(MDatos[['Close']])
pd.DataFrame(Y)
```

	0
0	234.289993
1	231.039993
2	236.320007
3	235.729996
4	236.860001
...	...
1252	228.669998
1253	239.130005
1254	241.589996
1255	234.000000
1256	234.000000

1257 rows × 1 columns

Debemos hacer una división de los datos, para obtener nuestras variables de entrenamiento y las variables para comprar los resultados, para ambos modelos utilizaremos las misma configuraciones para poder observar mejor la diferencia en los resultados finales.

```
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
                                                                    test_size = 0.2,
                                                                    random_state = 0,
                                                                    shuffle = True)
```

```
XVol_train, XVol_test, YVol_train, YVol_test = model_selection.train_test_split(XVol, Y,
                                                                    test_size = 0.2,
                                                                    random_state = 0,
                                                                    shuffle = True)
```

Se entrena el modelo para el árbol de decisión de regresión, con una profundidad máxima de 14, con un mínimo de dos hojas y una separación de cuatro valores mínimos, este entrenamiento se hace para ambos modelos.

```
PronosticoAD = DecisionTreeRegressor(max_depth=14,
                                     min_samples_split=4,
                                     min_samples_leaf=2,
                                     random_state=0)
PronosticoAD.fit(X_train, Y_train)
```

```
DecisionTreeRegressor(max_depth=14, min_samples_leaf=2, min_samples_split=4,
                      random_state=0)
```

```
PronosticoADVol = DecisionTreeRegressor(max_depth=14,
                                     min_samples_split=4,
                                     min_samples_leaf=2,
                                     random_state=0)
PronosticoADVol.fit(XVol_train, YVol_train)
```

```
DecisionTreeRegressor(max_depth=14, min_samples_leaf=2, min_samples_split=4,
                      random_state=0)
```

Generamos el pronóstico para ambos modelos y obtenemos sus características entre ellas la importancia de las variables y el puntaje, que entre mas cercano sea este valor a 1 significa que es más exacto, para el caso donde no usamos volumen tiene un score de 0.9897 y donde usamos volumen es de 0.9889, podemos ver que prácticamente el valor es el mismo para ambos casos, usando o no usando la variable de volumen.

```
Y_Pronostico = PronosticoAD.predict(X_test)
YVol_Pronostico = PronosticoADVol.predict(XVol_test)
```

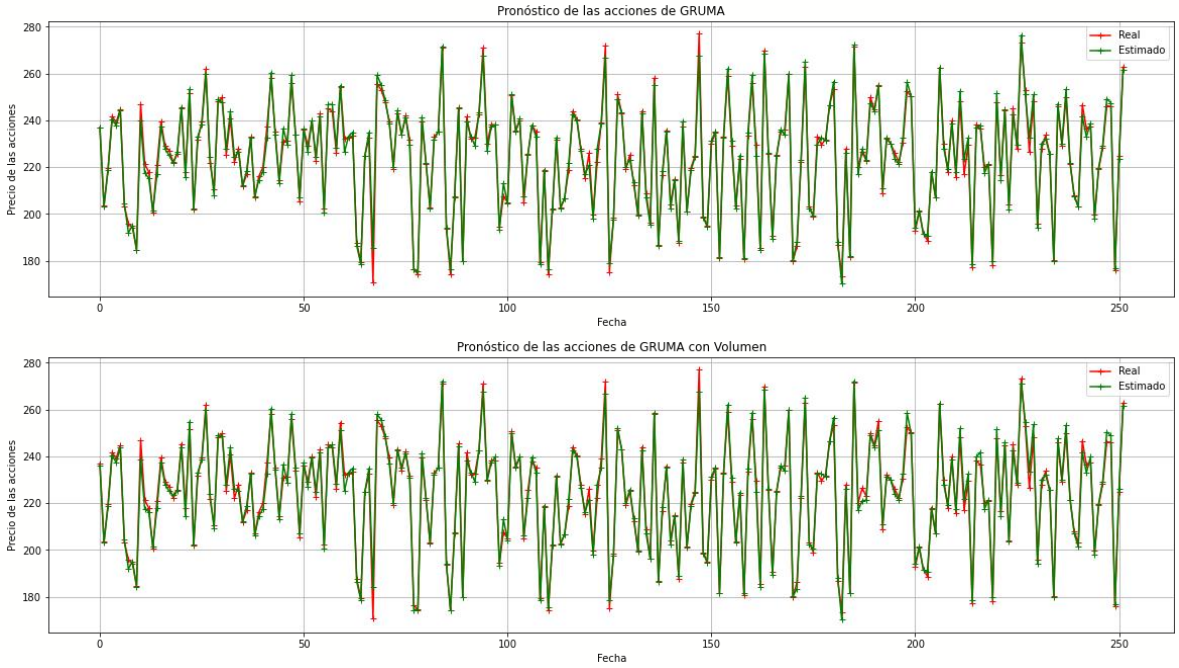
```
print('Sin Volumen')
print('Criterio: \n', PronosticoAD.criterion)
print('Importancia variables: \n', PronosticoAD.feature_importances_)
print("MAE: %.4f" % mean_absolute_error(Y_test, Y_Pronostico))
print("MSE: %.4f" % mean_squared_error(Y_test, Y_Pronostico))
print("RMSE: %.4f" % mean_squared_error(Y_test, Y_Pronostico, squared=False))
print('Score: %.4f' % r2_score(Y_test, Y_Pronostico))
```

```
Sin Volumen
Criterio:
 squared_error
Importancia variables:
 [0.00195707 0.09396903 0.90407391]
MAE: 1.6602
MSE: 5.5205
RMSE: 2.3496
Score: 0.9897
```

```
print('Con Volumen')
print('Criterio: \n', PronosticoADVol.criterion)
print('Importancia variables: \n', PronosticoADVol.feature_importances_)
print("MAE: %.4f" % mean_absolute_error(YVol_test, YVol_Pronostico))
print("MSE: %.4f" % mean_squared_error(YVol_test, YVol_Pronostico))
print("RMSE: %.4f" % mean_squared_error(YVol_test, YVol_Pronostico, squared=False))
print('Score: %.4f' % r2_score(YVol_test, YVol_Pronostico))
```

```
Con Volumen
Criterio:
 squared_error
Importancia variables:
 [1.79824628e-03 9.39717972e-02 9.03398827e-01 8.31129130e-04]
MAE: 1.7523
MSE: 5.9454
RMSE: 2.4383
Score: 0.9889
```

A continuación se muestra una grafica comprando los valores reales con el valor pronosticado para cada uno de nuestros modelos, es posible observar que los valores son muy semejantes en ambos modelos.



Otro punto importante a considerar es la importancia que tiene cada una de las variables al momento de determinar el valor de cierre, en ambos casos se observa que la variable con mayor importancia es “Low” seguido de “High” y “Open” respectivamente, y en el caso del modelo que tiene volumen es la variable con menor importancia.

```
Importancia = pd.DataFrame({'Variable': list(MDatos[['Open', 'High', 'Low']]),
                            'Importancia': PronosticoAD.feature_importances_}).sort_values('Importancia', ascending=False)
Importancia
```

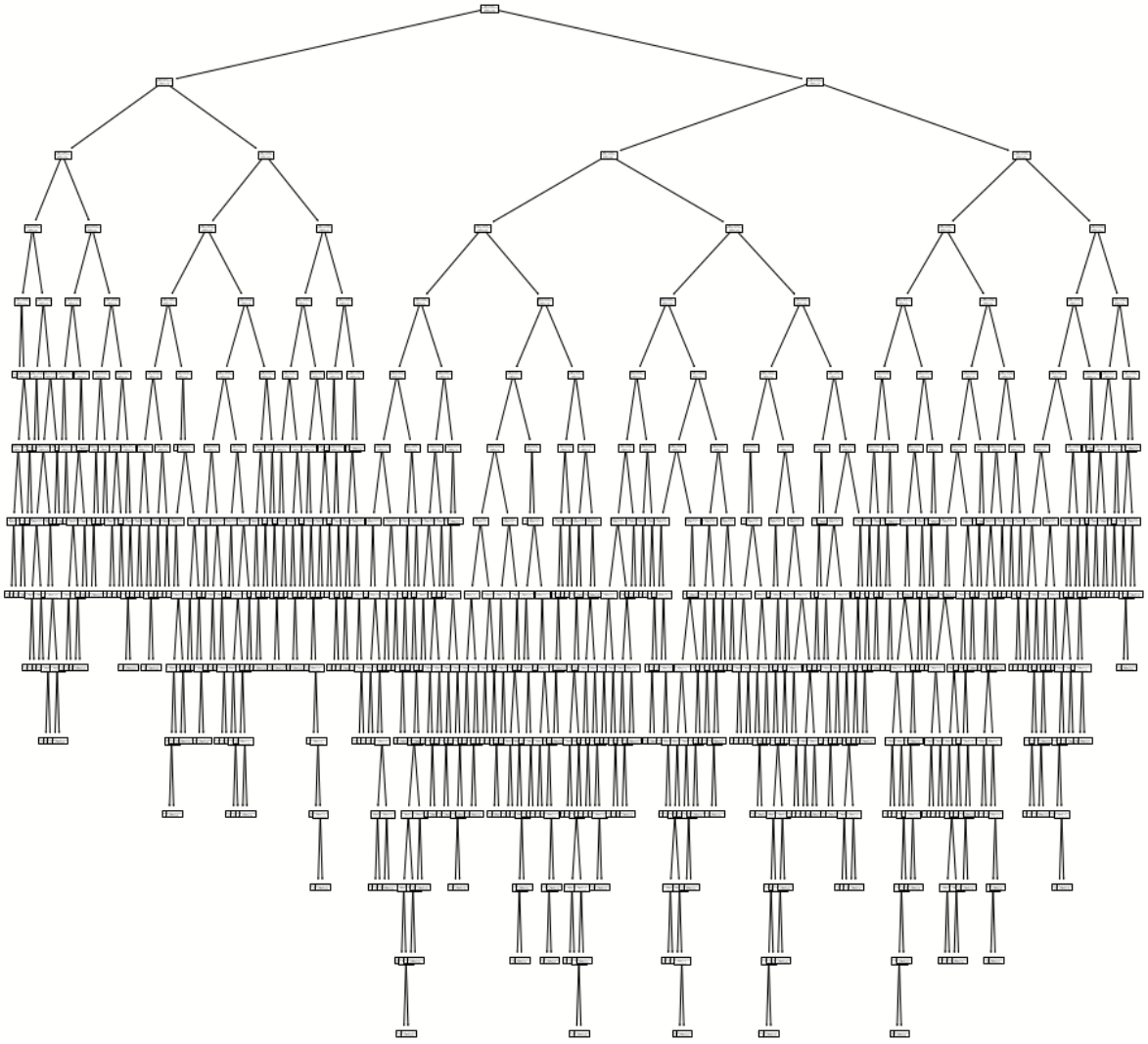
	Variable	Importancia
2	Low	0.904074
1	High	0.093969
0	Open	0.001957

```
ImportanciaVol = pd.DataFrame({'Variable': list(MDatosVol[['Open', 'High', 'Low','Volume']]),
                            'Importancia': PronosticoADVol.feature_importances_}).sort_values('Importancia', ascending=False)
ImportanciaVol
```

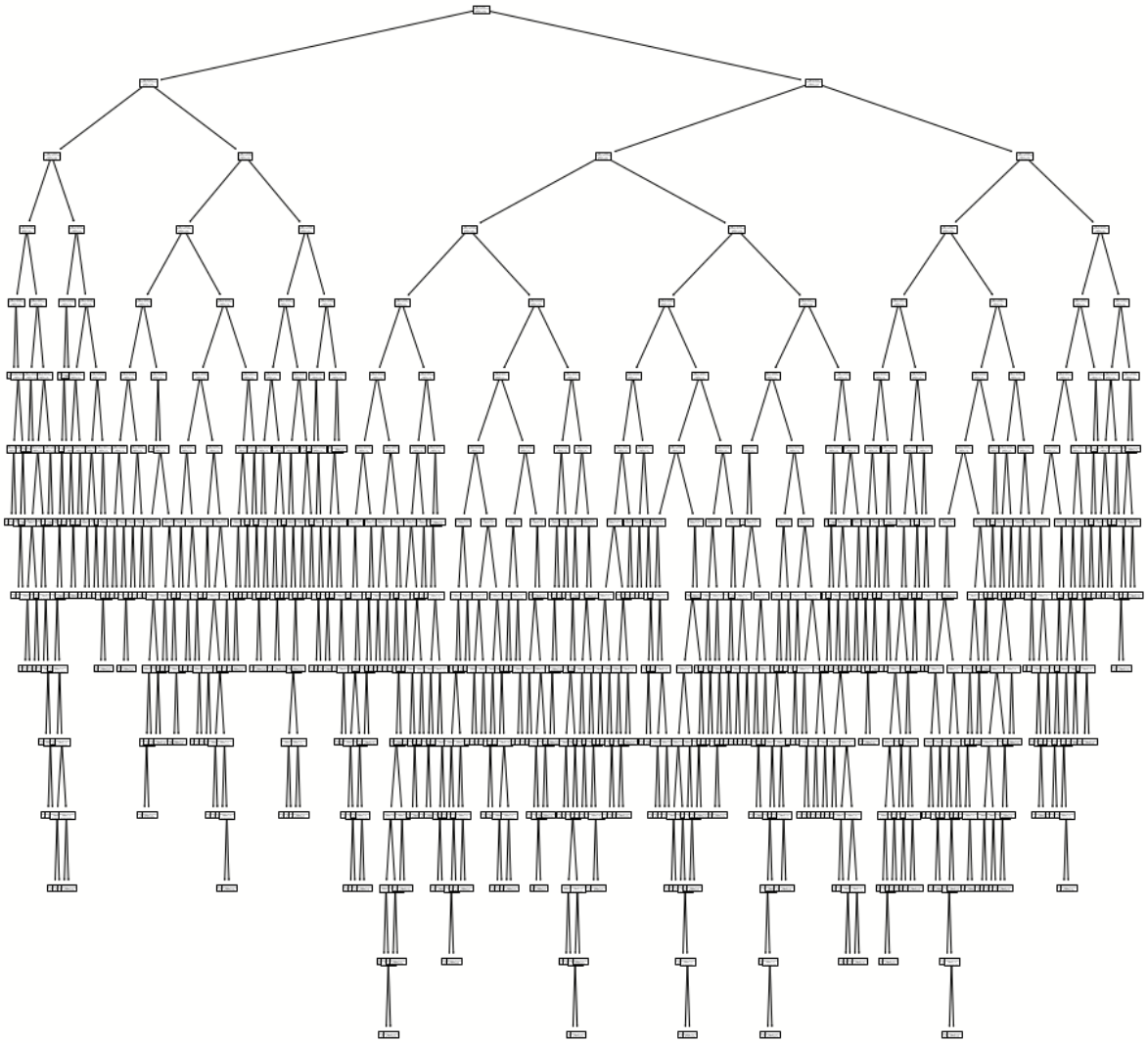
	Variable	Importancia
2	Low	0.903399
1	High	0.093972
0	Open	0.001798
3	Volume	0.000831

Procederemos a mostrar los árboles generados, primeramente el árbol si volumen, y posteriormente el árbol contemplando la variable volumen, a simple vista podemos observar que son bastante similares en cuanto a la estructura pero hay pequeños cambios entre uno y otro.

```
from sklearn.tree import plot_tree
plt.figure(figsize=(16,16))
plot_tree(PronosticoAD,
          feature_names = ['Open', 'High', 'Low'])
plt.show()
```



```
plt.figure(figsize=(16,16))
plot_tree(PronosticoADVol,
          feature_names = ['Open', 'High', 'Low','Volume'])
plt.show()
```



Ahora procederemos a realizar los modelos de pronostico de bosques aleatorios, para esto se usó la función *RandomForestRegressor* , y con las mismas configuraciones que se utilizaron los arboles de decisión, esto para que pudiéramos notar mejor pudiéramos notar la diferencia en los resultados finales.

```
PronosticoBA = RandomForestRegressor(n_estimators=100,
                                     max_depth=14,
                                     min_samples_split=4,
                                     min_samples_leaf=2,
                                     random_state=0)

PronosticoBA.fit(X_train, Y_train)
```

```
C:\Users\osva_\AppData\Local\Temp\ipykernel_19808\3002337042.py:7: DataConversionWarning: A column-vector y was passed when a 1
d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  PronosticoBA.fit(X_train, Y_train)
```

```
RandomForestRegressor(max_depth=14, min_samples_leaf=2, min_samples_split=4,
                      random_state=0)
```

```
PronosticoBAVol = RandomForestRegressor(n_estimators=100,
                                        max_depth=14,
                                        min_samples_split=4,
                                        min_samples_leaf=2,
                                        random_state=0)

PronosticoBAVol.fit(XVol_train, YVol_train)
```

```
C:\Users\osva_\AppData\Local\Temp\ipykernel_19808\851189389.py:7: DataConversionWarning: A column-vector y was passed when a 1d
array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  PronosticoBAVol.fit(XVol_train, YVol_train)
```

```
RandomForestRegressor(max_depth=14, min_samples_leaf=2, min_samples_split=4,
                      random_state=0)
```

De igual forma que al momento de usar árboles, procedemos a calcular los valores pronosticados de ambos bosques aleatorios, y posteriormente observar la importancia de cada una de las variables al momento de calcular, en la importancia de variables al momento de calcular el cierre, podemos observar algo similar que en los arboles de decisión, donde “Low” es la variable con mas importancia, seguida por “High”, “Open” y en el segundo modelo “Volume”, aunque los porcentajes son diferentes que en el caso de arboles.

```
Y_PronosticoBA = PronosticoBA.predict(X_test)
YVol_PronosticoBA = PronosticoBAVol.predict(XVol_test)
```

```
ImportanciaBosques = pd.DataFrame({'Variable': list(MDatos[['Open',
                                                            'High',
                                                            'Low']]),
                                   'Importancia': PronosticoBA.feature_importances_}).sort_values('Importancia', ascending=False)
ImportanciaBosques
```

	Variable	Importancia
2	Low	0.658944
1	High	0.338382
0	Open	0.002674

```
ImportanciaBosquesVol = pd.DataFrame({'Variable': list(MDatosVol[['Open',
                                                                  'High',
                                                                  'Low',
                                                                  'Volume']]),
                                      'Importancia': PronosticoBAVol.feature_importances_}).sort_values('Importancia', ascending=False)
ImportanciaBosquesVol
```

	Variable	Importancia
2	Low	0.707264
1	High	0.289073
0	Open	0.002695
3	Volume	0.000968

Ya que tenemos todos nuestros modelos hechos procederemos a obtener su Score, donde entre el valor este mas cercano a 1, significa que es mas exacto, podemos observar que en estos casos, el valor entre tener en cuenta o no el volumen no influye mucho en el resultado del score, es mas es ata un poco perjudicial para el modelo, ya que el puntaje baja algunas centésimas en comparación a donde no tomamos dicha columna.

```
print("SCORE : ")
print("Árbol de decisión:", r2_score(Y_test, Y_Pronostico))
print("Bosque aleatorio:", r2_score(Y_test, Y_PronosticoBA))
print("")
print("Árbol de decisión con Volumen:", r2_score(YVol_test, YVol_Pronostico))
print("Bosque aleatorio con Volumen:", r2_score(YVol_test, YVol_PronosticoBA))
```

```
SCORE :
Árbol de decisión: 0.9897144026133515
Bosque aleatorio: 0.9930740691152907

Árbol de decisión con Volumen: 0.9889228493211426
Bosque aleatorio con Volumen: 0.9925056143782158
```

Lo último a realizar es la comprobación generando nuevos pronósticos, para este caso utilizamos los datos de un registro que ya existe, sabiendo que el resultado de Close será 213.039993, así que probaremos los cuatro modelos hecho arboles de decisión y bosques aleatorios, tomando en cuenta tanto los que tiene volumen como los que no tienen volumen, en ambos caos se hizo un arreglo con los mismos valores, únicamente en uno agregando el volumen y los resultados podemos observar que ninguna es completamente exacto al valor original, pero no esta tan lejos del valor esperado, ya que recordemos que tenemos un buen Score pero no es completamente exacto.

```
# SIN VOLUMEN
# Close : 231.039993
PrecioAccion = pd.DataFrame({'Open': [235.460007],
                             'High': [237.470001],
                             'Low': [230.110001]})
print("Pronostico Arbol de Decisión sin Volumen",PronosticoAD.predict(PrecioAccion))
print("Pronostico Bosques Aleatorios sin Volumen",PronosticoBA.predict(PrecioAccion))
```

```
Pronostico Arbol de Decisión sin Volumen [236.5250015]
Pronostico Bosques Aleatorios sin Volumen [233.12326539]
```

```
# CON VOLUMEN
# Close: 231.039993
PrecioAccionVol = pd.DataFrame({'Open': [235.460007],
                             'High': [237.470001],
                             'Low': [230.110001],
                             'Volume': [1411562]})
print("Pronostico Arboles de Decisión con Volumen",PronosticoADVol.predict(PrecioAccionVol))
print("Pronostico Bosques Aleatorios con Volumen",PronosticoBAVol.predict(PrecioAccionVol))
```

```
Pronostico Arboles de Decisión con Volumen [236.5250015]
Pronostico Bosques Aleatorios con Volumen [233.38691389]
```

Conclusión

Es importante analizar aquellas variables que mas nos pueden ayudar en hacer un correcto análisis y no necesariamente utilizar todas, ya que como podemos observar puede resultar perjudicial, además de tener en claro que proceso vamos a hacer si un pronostico o una clasificación, en general en la práctica pude observar cómo los diferentes modelos con diferentes variables afectan el resultado final, pero en este caso no de manera exagerada, además de que al momento de obtener la importancia de cada una de las variables nos da la oportunidad de identificar de que variable podemos prescindir, y así tener un resultado mas exacto o un proceso menos complejo sin tatas variables.

Liga GitHub Practica 13

Cuaderno de Python de la practica 13

https://github.com/OsvaldoIG/MineriaDatos/tree/main/P13