MCD_Big_Data_Tarea_06_v4_Docker

November 7, 2024

1 Universidad Autónoma de Nuevo León

1.1 Facultad de Ciencias Fisico Matemáticas

1.1.1 Maestría en Ciencia de Datos

Alumno: Francisco D. Treviño B.

[ ]:

```

```

1.2 Tarea 6 (10 puntos): PySpark ML

[ ]:

```

```

2 Instalación de Spark

2.1 Se utilizó Docker para la instalación Spark usando una imagen, con el siguiente comando en Windows, según referencia en https://hub.docker.com/_/spark:

2.1.1 docker pull spark

2.2 Para correr Spark en Docker en modo interactivo se corre el siguiente comando:

2.2.1 docker pull jupyter/pyspark-notebook

2.2.2 docker run -it –name spark_container_big_data -p 8899:8888 -p 4040:4040 -v /c/MCD_Big_Data:/home/jovyan/work jupyter/pyspark-notebook

2.2.3 docker stop spark_container_big_data

2.2.4 docker start spark_container_big_data

[ ]:

```

```

[1]:

```
!pip install findspark
```

```
Requirement already satisfied: findspark in /opt/conda/lib/python3.11/site-
packages (2.0.1)
```

[2]:
```
!pip install pyspark
```

```
Requirement already satisfied: pyspark in /usr/local/spark/python (3.5.0)
Requirement already satisfied: py4j==0.10.9.7 in /opt/conda/lib/python3.11/site-
packages (from pyspark) (0.10.9.7)
```

[3]:
```
!pip install py4j
```

```
Requirement already satisfied: py4j in /opt/conda/lib/python3.11/site-packages
(0.10.9.7)
```

[4]:
```
# !pip install jupyterlab
```

[5]:
```
!pip install pandas
```

```
Requirement already satisfied: pandas in /opt/conda/lib/python3.11/site-packages
(2.0.3)
Requirement already satisfied: python-dateutil>=2.8.2 in
/opt/conda/lib/python3.11/site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.11/site-
packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in /opt/conda/lib/python3.11/site-
packages (from pandas) (2023.3)
Requirement already satisfied: numpy>=1.21.0 in /opt/conda/lib/python3.11/site-
packages (from pandas) (1.24.4)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.11/site-
packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
```

[6]:
```
!pip install mplfinance
```

```
Requirement already satisfied: mplfinance in /opt/conda/lib/python3.11/site-
packages (0.12.10b0)
Requirement already satisfied: matplotlib in /opt/conda/lib/python3.11/site-
packages (from mplfinance) (3.8.0)
Requirement already satisfied: pandas in /opt/conda/lib/python3.11/site-packages
(from mplfinance) (2.0.3)
Requirement already satisfied: contourpy>=1.0.1 in
/opt/conda/lib/python3.11/site-packages (from matplotlib->mplfinance) (1.1.1)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.11/site-
packages (from matplotlib->mplfinance) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/opt/conda/lib/python3.11/site-packages (from matplotlib->mplfinance) (4.43.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/opt/conda/lib/python3.11/site-packages (from matplotlib->mplfinance) (1.4.5)
Requirement already satisfied: numpy<2,>=1.21 in /opt/conda/lib/python3.11/site-
packages (from matplotlib->mplfinance) (1.24.4)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.11/site-packages (from matplotlib->mplfinance) (23.2)
```

```
Requirement already satisfied: pillow>=6.2.0 in /opt/conda/lib/python3.11/site-
packages (from matplotlib->mplfinance) (10.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/opt/conda/lib/python3.11/site-packages (from matplotlib->mplfinance) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/conda/lib/python3.11/site-packages (from matplotlib->mplfinance) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.11/site-
packages (from pandas->mplfinance) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in /opt/conda/lib/python3.11/site-
packages (from pandas->mplfinance) (2023.3)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.11/site-
packages (from python-dateutil>=2.7->matplotlib->mplfinance) (1.16.0)
```

[7]:
```
from pyspark import SparkConf
from pyspark import SparkContext as sc
```

[8]:
```
import os
import sys

import findspark
import pyspark

from pyspark.sql import DataFrame
from typing import List
import pyspark.sql.types as T
import pyspark.sql.functions as F
```

[9]:
```
from pyspark.sql import SparkSession
```

[10]:
```
findspark.init()
findspark.find()
```

[10]:
```
'/usr/local/spark'
```

[11]:
```
spark=SparkSession.builder.appName("Intraday").getOrCreate()
```

[12]:
```
spark
```

[12]:
```
<pyspark.sql.session.SparkSession at 0x7f0ac418ab50>
```

[13]:
```
# Paso 1: Lista de archivos CSV
# csv_directory = "c/MCD_Big_Data/"
csv_directory = "/home/jovyan/work/"

# Nombres de los archivos CSV
# csv_file_names = ["intraday_bmv_e.txt"]
csv_file_names = ["intraday_bmv_e_full.txt"]
```

```python
# Lista para almacenar los DataFrames cargados
df_list = []

# Cargar los archivos CSV, especificando que están delimitados por TAB
for csv_file_name in csv_file_names:
    # Leer el archivo CSV desde la ruta de Google Drive
    file_path = csv_directory + csv_file_name
    df = spark.read.csv(file_path, sep="\t", header=True, inferSchema=True)
    df_list.append(df)
```

[14]:
```python
# Unir todos los DataFrames en uno solo (si es necesario)
combined_df = df_list[0]
for df in df_list[1:]:
    combined_df = combined_df.union(df)

# Eliminar las columnas 'auction_indicator' y 'settlement'
combined_df = combined_df.drop('auction_indicator', 'settlement', 'chart')

# Mostrar una muestra del DataFrame unido
combined_df.show(5)
```

```
+-------------------+------------+-------------+--------------------+------+----
--+------+--------+----------+---------+-----------+--------------+-------------
----+------------+---+--------+
|         trade_time|match_number|instrument_id|           timestamp|volume| pri
ce|amount|buyer_id|buyer_name|seller_id|seller_name|operation_type|concertation_
type|price_setter|lot|  symbol|
+-------------------+------------+-------------+--------------------+------+----
--+------+--------+----------+---------+-----------+--------------+-------------
----+------------+---+--------+
|2024-06-24 07:30:00|           1|            3|2024-06-24 07:30:…|
1|193.64|193.64|     112|     SCTIA|       14|        GBM|             C|
0|           0|  0|FEMSAUBD|
|2024-06-24 07:30:00|           1|            5|2024-06-24 07:30:…|     1|
61.53| 61.53|     112|     SCTIA|      112|      SCTIA|             C|
C|           0|  0| WALMEX*|
|2024-06-24 07:30:00|           1|           80|2024-06-24 07:30:…|     1|
29.34| 29.34|      14|       GBM|       14|        GBM|             C|
C|           0|  0|SORIANAB|
|2024-06-24 07:30:00|           1|          124|2024-06-24 07:30:…|     2|
68.44|136.88|     112|     SCTIA|      112|      SCTIA|             C|
C|           0|  0|  BIMBOA|
|2024-06-24 07:30:00|           1|         1729|2024-06-24 07:30:…|     8|
62.41|499.28|     112|     SCTIA|      119|      ACTIN|             C|
0|           0|  0|  ALSEA*|
+-------------------+------------+-------------+--------------------+------+----
```

```
--+------+--------+----------+---------+-----------+-------------+-------------
----+------------+---+--------+
only showing top 5 rows
```

[15]:
```
combined_df.printSchema()
```

```
root
 |-- trade_time: timestamp (nullable = true)
 |-- match_number: integer (nullable = true)
 |-- instrument_id: integer (nullable = true)
 |-- timestamp: timestamp (nullable = true)
 |-- volume: integer (nullable = true)
 |-- price: double (nullable = true)
 |-- amount: double (nullable = true)
 |-- buyer_id: integer (nullable = true)
 |-- buyer_name: string (nullable = true)
 |-- seller_id: integer (nullable = true)
 |-- seller_name: string (nullable = true)
 |-- operation_type: string (nullable = true)
 |-- concertation_type: string (nullable = true)
 |-- price_setter: integer (nullable = true)
 |-- lot: integer (nullable = true)
 |-- symbol: string (nullable = true)
```

[16]:
```
combined_df = combined_df.repartition(10)
```

[17]:
```
# Convertir el DataFrame a un RDD
rdd = combined_df.rdd
```

[18]:
```
rdd.take(5)  # Muestra los primeros 5 registros del RDD
```

[18]:
```
[Row(trade_time=datetime.datetime(2024, 6, 24, 13, 1, 1), match_number=11488,
instrument_id=5, timestamp=datetime.datetime(2024, 6, 24, 13, 1, 1, 737000),
volume=426, price=60.73, amount=25870.98, buyer_id=54, buyer_name='MERL',
seller_id=0, seller_name='GS', operation_type='C', concertation_type='O',
price_setter=1, lot=1, symbol='WALMEX*'),
 Row(trade_time=datetime.datetime(2024, 6, 24, 13, 8, 20), match_number=7702,
instrument_id=3, timestamp=datetime.datetime(2024, 6, 24, 13, 8, 20, 485000),
volume=7, price=193.83, amount=1356.81, buyer_id=141, buyer_name='BTGP',
seller_id=0, seller_name='FMX', operation_type='C', concertation_type='O',
price_setter=0, lot=0, symbol='FEMSAUBD'),
 Row(trade_time=datetime.datetime(2024, 6, 24, 13, 43, 1), match_number=9407,
instrument_id=3, timestamp=datetime.datetime(2024, 6, 24, 13, 43, 1, 58000),
volume=164, price=194.39, amount=31879.96, buyer_id=123, buyer_name='JPM',
seller_id=51, seller_name='SANT', operation_type='C', concertation_type='O',
```

```
price_setter=1, lot=1, symbol='FEMSAUBD'),
 Row(trade_time=datetime.datetime(2024, 6, 24, 10, 29, 32), match_number=3737,
instrument_id=124, timestamp=datetime.datetime(2024, 6, 24, 10, 29, 32, 942000),
volume=144, price=66.93, amount=9637.92, buyer_id=137, buyer_name='BXMAS',
seller_id=0, seller_name='FMX', operation_type='C', concertation_type='O',
price_setter=1, lot=1, symbol='BIMBOA'),
 Row(trade_time=datetime.datetime(2024, 6, 24, 9, 50, 34), match_number=3767,
instrument_id=5, timestamp=datetime.datetime(2024, 6, 24, 9, 50, 34, 878000),
volume=700, price=61.47, amount=43029.0, buyer_id=123, buyer_name='JPM',
seller_id=51, seller_name='SANT', operation_type='C', concertation_type='O',
price_setter=1, lot=1, symbol='WALMEX*')]
```

[19]:

```
combined_df.describe().show()
```

```
+-------+------------------+------------------+-----------------+---------------
---+------------------+------------------+----------+----------------+---------
--+--------------+-----------------+------------------+------------------+------
+
|summary|      match_number|     instrument_id|           volume|
price|            amount|          buyer_id|buyer_name|
seller_id|seller_name|operation_type|concertation_type|      price_setter|
lot|symbol|
+-------+------------------+------------------+-----------------+---------------
---+------------------+------------------+----------+----------------+---------
--+--------------+-----------------+------------------+------------------+------
+
|  count|            491919|            491919|           491919|
491919|            491919|            491919|    491912|            491919|
491912|        491919|           491919|            491919|
491919|491919|
|   mean|6993.5315671889075|15558.248250220056|509.8265385154873|
95.79442137120172|38960.131471402085|  65.5031092517264|
NULL|69.22345142188043|       NULL|           NULL|
NULL|0.6892008643699471|0.6892008643699471|  NULL|
| stddev| 6116.054363745008| 70343.12703310873|19416.64929639428|288.50029695505
384|1297577.5228590586|60.693670565272505|      NULL|60.70343370001806|
NULL|         NULL|             NULL|0.4628212056142812|0.4628212056142812|
NULL|
|    min|                 1|                 3|                1|
28.69|              28.9|                 0|     ACTIN|                0|
ACTIN|             C|                C|                 0|
0|ALSEA*|
|    max|             32146|            351814|          6000000|
16000.0|            3.75E8|               141|     VECTO|              141|
VECTO|             X|                w|                 1|                 1|
WMT*|
+-------+------------------+------------------+-----------------+---------------
```

```
---+------------------+------------------+----------+----------------+---------
--+--------------+------------------+------------------+------------------+------
+
```

[20]:
```
combined_df.filter(combined_df.buyer_id.isin([149, 50])).orderBy(F.
↪asc("trade_time")).limit(100).show()
```

```
+----------+------------+-------------+---------+------+-----+------+--------+--
--------+---------+-----------+-------------+----------------+-----------+---
+-------+
|trade_time|match_number|instrument_id|timestamp|volume|price|amount|buyer_id|bu
yer_name|seller_id|seller_name|operation_type|concertation_type|price_setter|lot
|symbol|
+----------+------------+-------------+---------+------+-----+------+--------+--
--------+---------+-----------+-------------+----------------+-----------+---
+-------+
+----------+------------+-------------+---------+------+-----+------+--------+--
--------+---------+-----------+-------------+----------------+-----------+---
+-------+
```

[21]:
```
from pyspark.sql import functions as F

combined_df.groupBy(F.to_date("trade_time").alias("trade_date"), "symbol",␣
↪"buyer_id") \
           .sum("volume") \
           .orderBy(F.desc("trade_date")) \
           .show(truncate=False)
```

```
+----------+----------+--------+-----------+
|trade_date|symbol    |buyer_id|sum(volume)|
+----------+----------+--------+-----------+
|2024-07-05|CHDRAUIB  |123     |8912       |
|2024-07-05|BIMBOA    |14      |40136      |
|2024-07-05|CHDRAUIB  |54      |2300       |
|2024-07-05|ALSEA*    |51      |8462       |
|2024-07-05|FEMSAUBD  |112     |8          |
|2024-07-05|WALMEX*   |38      |38440      |
|2024-07-05|BIMBOA    |29      |38848      |
|2024-07-05|LACOMERUBC|141     |38311      |
|2024-07-05|BIMBOA    |112     |794        |
|2024-07-05|LACOMERUBC|0       |29964      |
|2024-07-05|SORIANAB  |14      |1845       |
|2024-07-05|CHDRAUIB  |137     |900        |
|2024-07-05|BIMBOA    |141     |17744      |
|2024-07-05|FEMSAUBD  |14      |571452     |
|2024-07-05|FEMSAUBD  |141     |17444      |
```

```
|2024-07-05|BIMBOA    |24      |5885       |
|2024-07-05|ALSEA*    |113     |13044      |
|2024-07-05|COST*     |0       |1448       |
|2024-07-05|WALMEX*   |112     |1240124    |
|2024-07-05|FEMSAUBD  |119     |32403      |
+----------+----------+--------+-----------+
only showing top 20 rows
```

[22]:

```python
# Realizar estadísticas descriptivas básicas
rdd_price = rdd.map(lambda row: row["price"])

# Filtrar valores nulos
rdd_price = rdd_price.filter(lambda x: x is not None)
```

[23]:

```python
filtered_rdd = rdd.filter(lambda row: row["symbol"] in ["WALMEX*"])

# Obtener solo la columna de precios y filtrar valores nulos
rdd_price = filtered_rdd.map(lambda row: row["price"]).filter(lambda x: x is␣
 ↪not None)

# Realizar estadísticas descriptivas básicas
count = rdd_price.count()
mean = rdd_price.mean()
min_value = rdd_price.min()
max_value = rdd_price.max()
stddev = rdd_price.stdev()

# Mostrar los resultados
print("Symbol: WALMEX*")
print(f"Count: {count}")
print(f"Mean: {mean}")
print(f"Min: {min_value}")
print(f"Max: {max_value}")
print(f"Standard Deviation: {stddev}")
```

```
Symbol: WALMEX*
Count: 170282
Mean: 61.833344011698216
Min: 60.11
Max: 63.09
Standard Deviation: 0.5602711227034469
```

[24]:

```python
from pyspark.sql import functions as F
from pyspark.sql import Window
```

```python
# Calcular estadísticas descriptivas para cada símbolo utilizando el DataFrame␣
↪de PySpark
statistics_by_symbol_df = (
    combined_df
    .groupBy("symbol")
    .agg(
        F.count("price").alias("Count"),
        F.mean("price").alias("Mean"),
        F.min("price").alias("Min"),
        F.max("price").alias("Max"),
        F.stddev("price").alias("Standard Deviation")
    )
)

# Mostrar los resultados
statistics_by_symbol_df.show(truncate=False)

# Convertir a un diccionario para una visualización similar al formato␣
↪anterior, si es necesario
statistics_by_symbol = {row["symbol"]: {
    "Count": row["Count"],
    "Mean": row["Mean"],
    "Min": row["Min"],
    "Max": row["Max"],
    "Standard Deviation": row["Standard Deviation"]
} for row in statistics_by_symbol_df.collect()}

# Mostrar el diccionario
for symbol, stats in statistics_by_symbol.items():
    print(f"Estadísticas para {symbol}:")
    print(f"  Count: {stats['Count']}")
    print(f"  Mean: {stats['Mean']}")
    print(f"  Min: {stats['Min']}")
    print(f"  Max: {stats['Max']}")
    print(f"  Standard Deviation: {stats['Standard Deviation']}")
    print("-" * 40)
```

```
+----------+------+------------------+-------+-------+------------------+
|symbol    |Count |Mean              |Min    |Max    |Standard Deviation|
+----------+------+------------------+-------+-------+------------------+
|SORIANAB  |594   |29.89638047138047 |28.69  |31.26  |0.5815359061491862|
|BIMBOA    |102314|65.07522455577939 |62.53  |68.7   |1.231335289117641 |
|ALSEA*    |70375 |63.08941042415578 |61.06  |65.21  |0.9659062397798771|
|COST*     |163   |15628.442822085894|15144.0|16000.0|206.3963691825507 |
|FEMSAUBD  |83760 |195.21430262296937|190.25 |200.27 |1.8748538340315664|
|WALMEX*   |170282|61.833344011696205|60.11  |63.09  |0.5602727678379705|
|LACOMERUBC|20613 |37.0466652112744  |35.45  |39.14  |0.8481255163315584|
```

```
|CHDRAUIB  |43591 |126.95723750315354|124.91 |129.89 |0.8210999781545977|
|WMT*      |227   |1241.3490308370046|1199.01|1286.99|13.495059664179085|
+----------+------+------------------+-------+-------+------------------+

Estadísticas para SORIANAB:
  Count: 594
  Mean: 29.89638047138047
  Min: 28.69
  Max: 31.26
  Standard Deviation: 0.5815359061491862
----------------------------------------
Estadísticas para BIMBOA:
  Count: 102314
  Mean: 65.07522455577939
  Min: 62.53
  Max: 68.7
  Standard Deviation: 1.231335289117641
----------------------------------------
Estadísticas para ALSEA*:
  Count: 70375
  Mean: 63.08941042415578
  Min: 61.06
  Max: 65.21
  Standard Deviation: 0.9659062397798771
----------------------------------------
Estadísticas para COST*:
  Count: 163
  Mean: 15628.442822085894
  Min: 15144.0
  Max: 16000.0
  Standard Deviation: 206.3963691825507
----------------------------------------
Estadísticas para FEMSAUBD:
  Count: 83760
  Mean: 195.21430262296937
  Min: 190.25
  Max: 200.27
  Standard Deviation: 1.8748538340315664
----------------------------------------
Estadísticas para WALMEX*:
  Count: 170282
  Mean: 61.833344011696205
  Min: 60.11
  Max: 63.09
  Standard Deviation: 0.5602727678379705
----------------------------------------
Estadísticas para LACOMERUBC:
  Count: 20613
```

```
  Mean: 37.0466652112744
  Min: 35.45
  Max: 39.14
  Standard Deviation: 0.8481255163315584
----------------------------------------
Estadísticas para CHDRAUIB:
  Count: 43591
  Mean: 126.95723750315354
  Min: 124.91
  Max: 129.89
  Standard Deviation: 0.8210999781545977
----------------------------------------
Estadísticas para WMT*:
  Count: 227
  Mean: 1241.3490308370046
  Min: 1199.01
  Max: 1286.99
  Standard Deviation: 13.495059664179085
----------------------------------------
```

[ ]:

```

```

[25]:

```python
df_a = combined_df
```

[26]:

```python
# Filtrar el DataFrame original para incluir solo las filas con el símbolo
 ↪"WALMEX*"
df_a = combined_df.filter(combined_df["symbol"] == "WALMEX*")

# Mostrar los primeros registros para verificar
df_a.show(2)
```

```
+-------------------+------------+-------------+--------------------+------+----
-+---------+--------+----------+---------+-----------+-------------+------------
-----+------------+---+-------+
|         trade_time|match_number|instrument_id|
timestamp|volume|price|  amount|buyer_id|buyer_name|seller_id|seller_name|operat
ion_type|concertation_type|price_setter|lot| symbol|
+-------------------+------------+-------------+--------------------+------+----
-+---------+--------+----------+---------+-----------+-------------+------------
-----+------------+---+-------+
|2024-06-24 08:30:46|        1653|            5|2024-06-24 08:30:…|
218|61.74|13459.32|       0|        GS|      113|       CITI|             C|
0|            1|  1|WALMEX*|
|2024-06-25 08:11:24|        2323|            5|2024-06-25 08:11:…|
2|61.22|  122.44|      14|       GBM|        0|         GS|             C|
0|            0|  0|WALMEX*|
+-------------------+------------+-------------+--------------------+------+----
-+---------+--------+----------+---------+-----------+-------------+------------
```

```
-----+------------+---+--------+
only showing top 2 rows
```

[ ]:

[27]:
```python
from pyspark.sql import functions as F

# Obtener una lista de símbolos únicos en el DataFrame
unique_symbols = [row["symbol"] for row in combined_df.select("symbol").
 ↪distinct().collect()]

# Crear un diccionario para almacenar DataFrames por símbolo
df_symbols = {}

# Iterar sobre cada símbolo y crear un DataFrame filtrado
for symbol in unique_symbols:
    df_symbols[symbol] = combined_df.filter(F.col("symbol") == symbol)

# Opcional: Mostrar los primeros registros de cada DataFrame para verificar
for symbol, df in df_symbols.items():
    print(f"Primeros registros para el símbolo {symbol}:")
    df.show(2)
```

```
Primeros registros para el símbolo SORIANAB:
+-------------------+------------+-------------+--------------------+------+----
-+------+--------+----------+---------+-----------+--------------+--------------
---+------------+---+--------+
|         trade_time|match_number|instrument_id|           timestamp|volume|pric
e|amount|buyer_id|buyer_name|seller_id|seller_name|operation_type|concertation_t
ype|price_setter|lot|  symbol|
+-------------------+------------+-------------+--------------------+------+----
-+------+--------+----------+---------+-----------+--------------+--------------
---+------------+---+--------+
|2024-06-24 12:14:15|          78|           80|2024-06-24 12:14:…|   161|
29.0|4669.0|      14|       GBM|      137|      BXMAS|             C|
0|           1|  1|SORIANAB|
|2024-06-25 07:30:00|           3|           80|2024-06-25 07:30:…|
1|29.45| 29.45|      14|       GBM|       14|        GBM|             C|
C|           0|  0|SORIANAB|
+-------------------+------------+-------------+--------------------+------+----
-+------+--------+----------+---------+-----------+--------------+--------------
---+------------+---+--------+
only showing top 2 rows

Primeros registros para el símbolo BIMBOA:
+-------------------+------------+-------------+--------------------+------+----
-+------+--------+----------+---------+-----------+--------------+--------------
```

```
---+------------+---+------+
|         trade_time|match_number|instrument_id|           timestamp|volume|pric
e|amount|buyer_id|buyer_name|seller_id|seller_name|operation_type|concertation_t
ype|price_setter|lot|symbol|
+-------------------+------------+-------------+--------------------+------+----
-+------+--------+----------+---------+----------+--------------+--------------
---+------------+---+------+
|2024-06-25 08:19:57|         644|          124|2024-06-25 08:19:…|
100|65.62|6562.0|       0|       FMX|      136|         MS|             C|
0|           1|  1|BIMBOA|
|2024-06-24 13:43:41|       14879|          124|2024-06-24 13:43:…|
100|65.72|6572.0|     113|      CITI|       54|       MERL|             C|
0|           1|  1|BIMBOA|
+-------------------+------------+-------------+--------------------+------+----
-+------+--------+----------+---------+----------+--------------+--------------
---+------------+---+------+
only showing top 2 rows

Primeros registros para el símbolo ALSEA*:
+-------------------+------------+-------------+--------------------+------+----
-+-------+--------+----------+---------+----------+--------------+-------------
----+------------+---+------+
|         trade_time|match_number|instrument_id|
timestamp|volume|price| amount|buyer_id|buyer_name|seller_id|seller_name|operati
on_type|concertation_type|price_setter|lot|symbol|
+-------------------+------------+-------------+--------------------+------+----
-+-------+--------+----------+---------+----------+--------------+-------------
----+------------+---+------+
|2024-06-24 07:53:25|         211|         1729|2024-06-24 07:53:…|
100|62.78| 6278.0|       0|        GS|      123|        JPM|             C|
0|           1|  1|ALSEA*|
|2024-06-24 11:17:49|        3193|         1729|2024-06-24 11:17:…|
107|61.61|6592.27|      14|       GBM|      141|       BTGP|             C|
0|           1|  1|ALSEA*|
+-------------------+------------+-------------+--------------------+------+----
-+-------+--------+----------+---------+----------+--------------+-------------
----+------------+---+------+
only showing top 2 rows

Primeros registros para el símbolo COST*:
+-------------------+------------+-------------+--------------------+------+----
----+--------+--------+----------+---------+----------+--------------+---------
--------+------------+---+------+
|         trade_time|match_number|instrument_id|           timestamp|volume|
price|  amount|buyer_id|buyer_name|seller_id|seller_name|operation_type|concerta
tion_type|price_setter|lot|symbol|
+-------------------+------------+-------------+--------------------+------+----
----+--------+--------+----------+---------+----------+--------------+---------
```

```
--------+------------+---+------+
|2024-06-24 08:41:39|           3|         2702|2024-06-24 08:41:…|
5|15220.01|76100.05|       0|     BMCAP|         0|      BMCAP|             C|
O|           1|  1| COST*|
|2024-06-24 10:16:28|           9|         2702|2024-06-24 10:16:…|
1|15288.51|15288.51|       0|     BMCAP|         0|      BMCAP|             C|
C|           0|  0| COST*|
+-------------------+------------+-------------+--------------------+------+----
----+--------+--------+----------+---------+----------+------------+---------
--------+------------+---+------+
only showing top 2 rows

Primeros registros para el símbolo FEMSAUBD:
+-------------------+------------+-------------+--------------------+------+----
--+-------+--------+----------+---------+----------+------------+------------
-----+------------+---+--------+
|         trade_time|match_number|instrument_id|           timestamp|volume|
price| amount|buyer_id|buyer_name|seller_id|seller_name|operation_type|concertat
ion_type|price_setter|lot|  symbol|
+-------------------+------------+-------------+--------------------+------+----
--+-------+--------+----------+---------+----------+------------+------------
-----+------------+---+--------+
|2024-06-24 13:08:20|        7702|            3|2024-06-24 13:08:…|
7|193.83|1356.81|     141|      BTGP|         0|        FMX|             C|
O|           0|  0|FEMSAUBD|
|2024-06-24 08:54:24|        1110|            3|2024-06-24 08:54:…|
100|196.93|19693.0|     141|      BTGP|       136|         MS|             C|
O|           1|  1|FEMSAUBD|
+-------------------+------------+-------------+--------------------+------+----
--+-------+--------+----------+---------+----------+------------+------------
-----+------------+---+--------+
only showing top 2 rows

Primeros registros para el símbolo WALMEX*:
+-------------------+------------+-------------+--------------------+------+----
-+--------+--------+----------+---------+----------+------------+------------
-----+------------+---+-------+
|         trade_time|match_number|instrument_id|
timestamp|volume|price|  amount|buyer_id|buyer_name|seller_id|seller_name|operat
ion_type|concertation_type|price_setter|lot| symbol|
+-------------------+------------+-------------+--------------------+------+----
-+--------+--------+----------+---------+----------+------------+------------
-----+------------+---+-------+
|2024-06-24 08:30:46|        1653|            5|2024-06-24 08:30:…|
218|61.74|13459.32|       0|        GS|       113|       CITI|             C|
O|           1|  1|WALMEX*|
|2024-06-25 08:11:24|        2323|            5|2024-06-25 08:11:…|
2|61.22|  122.44|      14|       GBM|         0|         GS|             C|
```

```
0|           0|  0|WALMEX*|
+-------------------+------------+-------------+--------------------+------+----
-+--------+--------+----------+---------+-----------+-------------+------------
-----+------------+---+-------+
only showing top 2 rows

Primeros registros para el símbolo LACOMERUBC:
+-------------------+------------+-------------+--------------------+------+----
-+------+--------+----------+---------+-----------+-------------+--------------
---+------------+---+----------+
|         trade_time|match_number|instrument_id|           timestamp|volume|pric
e|amount|buyer_id|buyer_name|seller_id|seller_name|operation_type|concertation_t
ype|price_setter|lot|    symbol|
+-------------------+------------+-------------+--------------------+------+----
-+------+--------+----------+---------+-----------+-------------+--------------
---+------------+---+----------+
|2024-06-24 13:47:53|        2731|       351814|2024-06-24 13:47:…|
90|37.57|3381.3|       0|       FMX|        0|        FMX|             C|
C|           0|  0|LACOMERUBC|
|2024-06-24 12:34:24|        1790|       351814|2024-06-24 12:34:…|
100|37.66|3766.0|     123|       JPM|      136|         MS|             C|
0|           1|  1|LACOMERUBC|
+-------------------+------------+-------------+--------------------+------+----
-+------+--------+----------+---------+-----------+-------------+--------------
---+------------+---+----------+
only showing top 2 rows

Primeros registros para el símbolo CHDRAUIB:
+-------------------+------------+-------------+--------------------+------+----
--+-------+--------+----------+---------+-----------+-------------+------------
-----+------------+---+--------+
|         trade_time|match_number|instrument_id|           timestamp|volume|
price| amount|buyer_id|buyer_name|seller_id|seller_name|operation_type|concertat
ion_type|price_setter|lot|  symbol|
+-------------------+------------+-------------+--------------------+------+----
--+-------+--------+----------+---------+-----------+-------------+------------
-----+------------+---+--------+
|2024-06-24 13:43:37|        5049|         6080|2024-06-24 13:43:…|
80|128.14|10251.2|     138|      CICB|       51|       SANT|             C|
0|           0|  0|CHDRAUIB|
|2024-06-24 10:31:05|        1999|         6080|2024-06-24 10:31:…|    31|
126.8| 3930.8|     123|       JPM|        0|        FMX|             C|
0|           0|  0|CHDRAUIB|
+-------------------+------------+-------------+--------------------+------+----
--+-------+--------+----------+---------+-----------+-------------+------------
-----+------------+---+--------+
only showing top 2 rows
```

```
Primeros registros para el símbolo WMT*:
+-------------------+------------+-------------+--------------------+------+----
---+--------+--------+----------+---------+----------+-------------+----------
-------+------------+---+------+
|         trade_time|match_number|instrument_id|           timestamp|volume|
price|  amount|buyer_id|buyer_name|seller_id|seller_name|operation_type|concerta
tion_type|price_setter|lot|symbol|
+-------------------+------------+-------------+--------------------+------+----
---+--------+--------+----------+---------+----------+-------------+----------
-------+------------+---+------+
|2024-06-24 10:53:10|          31|         2056|2024-06-24 10:53:…|     1|
1237.5|  1237.5|       0|      BMCAP|        0|       BMCAP|             C|
C|           0|  0|  WMT*|
|2024-06-24 08:51:22|          19|         2056|2024-06-24 08:51:…|
22|1237.52|27225.44|       0|      BMCAP|        0|       BMCAP|             C|
O|           1|  1|  WMT*|
+-------------------+------------+-------------+--------------------+------+----
---+--------+--------+----------+---------+----------+-------------+----------
-------+------------+---+------+
only showing top 2 rows
```

[ ]:

[28]:

```python
from pyspark.sql import functions as F
from pyspark.sql.window import Window

# Crear ventanas que particionen por símbolo y ordenen por fecha de operación␣
  ↪ascendente
day_window = Window.partitionBy('symbol').orderBy(F.to_date('trade_time'))
week_window = Window.partitionBy('symbol').orderBy(F.weekofyear('trade_time'))
month_window = Window.partitionBy('symbol').orderBy(F.month('trade_time'))

# -----------------------------------
# Variaciones respecto al día anterior
# -----------------------------------

# Obtener el último registro del día anterior
df_a = df_a.withColumn('last_day_close', F.lag(F.col('price')).over(day_window))
df_a = df_a.withColumn('last_day_close_date', F.lag(F.col('trade_time')).
  ↪over(day_window))

# Calcular la variación unitaria y porcentual diaria
df_a = df_a.withColumn('unitary_daily_variation', F.when(F.
  ↪col('last_day_close').isNotNull(),
                                                         F.col('price') - F.
  ↪col('last_day_close')).otherwise(None))
```

```
df_a = df_a.withColumn('percentage_daily_variation', F.when(F.
 ↪col('last_day_close').isNotNull() & (F.col('last_day_close') != 0),
                                                            (F.col('price') - F.
 ↪col('last_day_close')) / F.col('last_day_close') * 100).otherwise(None))

# -----------------------------------
# Variaciones respecto a la semana anterior
# -----------------------------------

# Obtener el último registro de la semana anterior
df_a = df_a.withColumn('last_week_close', F.lag(F.col('price')).
 ↪over(week_window))
df_a = df_a.withColumn('last_week_close_date', F.lag(F.col('trade_time')).
 ↪over(week_window))

# Calcular la variación unitaria y porcentual semanal
df_a = df_a.withColumn('unitary_weekly_variation', F.when(F.
 ↪col('last_week_close').isNotNull(),
                                                          F.col('price') - F.
 ↪col('last_week_close')).otherwise(None))
df_a = df_a.withColumn('percentage_weekly_variation', F.when(F.
 ↪col('last_week_close').isNotNull() & (F.col('last_week_close') != 0),
                                                             (F.col('price') -␣
 ↪F.col('last_week_close')) / F.col('last_week_close') * 100).otherwise(None))

# -----------------------------------
# Variaciones respecto al mes anterior
# -----------------------------------

# Obtener el último registro del mes anterior
df_a = df_a.withColumn('last_month_close', F.lag(F.col('price')).
 ↪over(month_window))
df_a = df_a.withColumn('last_month_close_date', F.lag(F.col('trade_time')).
 ↪over(month_window))

# Calcular la variación unitaria y porcentual mensual
df_a = df_a.withColumn('unitary_monthly_variation', F.when(F.
 ↪col('last_month_close').isNotNull(),
                                                           F.col('price') - F.
 ↪col('last_month_close')).otherwise(None))
df_a = df_a.withColumn('percentage_monthly_variation', F.when(F.
 ↪col('last_month_close').isNotNull() & (F.col('last_month_close') != 0),
                                                              (F.col('price') -␣
 ↪F.col('last_month_close')) / F.col('last_month_close') * 100).
 ↪otherwise(None))
```

```python
# Mostrar los resultados
df_a.select('trade_time', 'price', 'last_day_close', 'last_day_close_date',
 ↪'unitary_daily_variation', 'percentage_daily_variation',
            'last_week_close', 'last_week_close_date',
 ↪'unitary_weekly_variation', 'percentage_weekly_variation',
            'last_month_close', 'last_month_close_date',
 ↪'unitary_monthly_variation', 'percentage_monthly_variation').show()
```

```
+-------------------+-----+--------------+-------------------+------------------
-----+--------------------------+--------------+--------------------+----------
--------------+---------------------------+----------------+--------------------
-+-------------------------+---------------------------+
|         trade_time|price|last_day_close|last_day_close_date|unitary_daily_vari
ation|percentage_daily_variation|last_week_close|last_week_close_date|unitary_we
ekly_variation|percentage_weekly_variation|last_month_close|last_month_close_dat
e|unitary_monthly_variation|percentage_monthly_variation|
+-------------------+-----+--------------+-------------------+------------------
-----+--------------------------+--------------+--------------------+----------
--------------+---------------------------+----------------+--------------------
-+-------------------------+---------------------------+
|2024-06-24 08:57:39|61.67|          NULL|               NULL|
NULL|                      NULL|           NULL|                NULL|
NULL|                       NULL|            NULL|                NULL|
NULL|                       NULL|
|2024-06-24 08:20:41|61.58|         61.67|2024-06-24 08:57:39|
-0.09000000000000341|      -0.14593805740230809|          61.67| 2024-06-24
08:57:39|    -0.09000000000000341|        -0.14593805740230809|           61.67|
2024-06-24 08:57:39|     -0.09000000000000341|        -0.14593805740230809|
|2024-06-24 13:41:53|60.71|         61.58|2024-06-24 08:20:41|
-0.8699999999999974|       -1.4127963624553386|          61.58| 2024-06-24
08:20:41|     -0.8699999999999974|        -1.4127963624553386|           61.58|
2024-06-24 08:20:41|      -0.8699999999999974|        -1.4127963624553386|
|2024-06-24 13:42:48|60.65|         60.71|2024-06-24 13:41:53|
-0.0600000000000…|      -0.09883050568275781|          60.71| 2024-06-24
13:41:53|    -0.0600000000000…|        -0.09883050568275781|           60.71|
2024-06-24 13:41:53|      -0.0600000000000…|        -0.09883050568275781|
|2024-06-24 07:30:01|61.58|         60.65|2024-06-24 13:42:48|
0.9299999999999997|        1.5333882934872212|          60.65| 2024-06-24
13:42:48|      0.9299999999999997|         1.5333882934872212|           60.65|
2024-06-24 13:42:48|       0.9299999999999997|         1.5333882934872212|
|2024-06-24 10:03:50|61.41|         61.58|2024-06-24 07:30:01|
-0.1700000000000017|       -0.2760636570315065|          61.58| 2024-06-24
07:30:01|     -0.1700000000000017|        -0.2760636570315065|           61.58|
2024-06-24 07:30:01|      -0.1700000000000017|        -0.2760636570315065|
|2024-06-24 08:14:16| 61.8|         61.41|2024-06-24 10:03:50|
0.3900000000000057|        0.6350757205666839|          61.41| 2024-06-24
10:03:50|      0.3900000000000057|         0.6350757205666839|           61.41|
```

```
2024-06-24 10:03:50|      0.39000000000000057|          0.6350757205666839|
|2024-06-24 08:06:14|61.66|          61.8|2024-06-24 08:14:16|
-0.14000000000000057|       -0.2265372168284799|          61.8| 2024-06-24
08:14:16|    -0.14000000000000057|        -0.2265372168284799|            61.8|
2024-06-24 08:14:16|     -0.14000000000000057|        -0.2265372168284799|
|2024-06-24 13:34:05|60.57|         61.66|2024-06-24 08:06:14|
-1.0899999999999963|        -1.767758676613682|          61.66| 2024-06-24
08:06:14|     -1.0899999999999963|         -1.767758676613682|           61.66|
2024-06-24 08:06:14|      -1.0899999999999963|         -1.767758676613682|
|2024-06-24 12:36:48|60.78|         60.57|2024-06-24 13:34:05|
0.21000000000000085|       0.34670629024269584|          60.57| 2024-06-24
13:34:05|      0.21000000000000085|        0.34670629024269584|           60.57|
2024-06-24 13:34:05|      0.21000000000000085|        0.34670629024269584|
|2024-06-24 13:42:27|60.65|         60.78|2024-06-24 12:36:48|
-0.13000000000000256|      -0.21388614675880643|          60.78| 2024-06-24
12:36:48|    -0.13000000000000256|       -0.21388614675880643|           60.78|
2024-06-24 12:36:48|     -0.13000000000000256|       -0.21388614675880643|
|2024-06-24 09:24:07|61.46|         60.65|2024-06-24 13:42:27|
0.8100000000000023|        1.3355317394888744|          60.65| 2024-06-24
13:42:27|       0.8100000000000023|         1.3355317394888744|           60.65|
2024-06-24 13:42:27|       0.8100000000000023|         1.3355317394888744|
|2024-06-24 08:49:16|61.68|         61.46|2024-06-24 09:24:07|
0.21999999999999886|        0.3579563944028618|          61.46| 2024-06-24
09:24:07|      0.21999999999999886|         0.3579563944028618|           61.46|
2024-06-24 09:24:07|      0.21999999999999886|         0.3579563944028618|
|2024-06-24 09:01:26|61.57|         61.68|2024-06-24 08:49:16|
-0.10999999999999943|      -0.17833981841763852|          61.68| 2024-06-24
08:49:16|    -0.10999999999999943|       -0.17833981841763852|           61.68|
2024-06-24 08:49:16|     -0.10999999999999943|       -0.17833981841763852|
|2024-06-24 11:12:03|60.96|         61.57|2024-06-24 09:01:26|
-0.6099999999999994|        -0.9907422445996418|          61.57| 2024-06-24
09:01:26|     -0.6099999999999994|        -0.9907422445996418|           61.57|
2024-06-24 09:01:26|      -0.6099999999999994|        -0.9907422445996418|
|2024-06-24 11:02:45|60.94|         60.96|2024-06-24 11:12:03|
-0.0200000000000…|      -0.03280839895013636|          60.96| 2024-06-24
11:12:03|    -0.0200000000000…|        -0.03280839895013636|           60.96|
2024-06-24 11:12:03|     -0.0200000000000…|        -0.03280839895013636|
|2024-06-24 09:35:23|61.71|         60.94|2024-06-24 11:02:45|
0.7700000000000031|        1.2635379061371892|          60.94| 2024-06-24
11:02:45|       0.7700000000000031|         1.2635379061371892|           60.94|
2024-06-24 11:02:45|       0.7700000000000031|         1.2635379061371892|
|2024-06-24 12:52:53|60.77|         61.71|2024-06-24 09:35:23|
-0.9399999999999977|        -1.5232539296710383|          61.71| 2024-06-24
09:35:23|     -0.9399999999999977|        -1.5232539296710383|           61.71|
2024-06-24 09:35:23|      -0.9399999999999977|        -1.5232539296710383|
|2024-06-24 11:25:00|61.05|         60.77|2024-06-24 12:52:53|
0.27999999999999403|       0.46075366134604906|          60.77| 2024-06-24
12:52:53|      0.27999999999999403|        0.46075366134604906|           60.77|
```

```
2024-06-24 12:52:53|      0.27999999999999403|        0.46075366134604906|
|2024-06-24 13:55:27|60.67|         61.05|2024-06-24 11:25:00|
-0.37999999999999545|        -0.622440622440615|          61.05| 2024-06-24
11:25:00|    -0.37999999999999545|        -0.622440622440615|          61.05|
2024-06-24 11:25:00|     -0.37999999999999545|          -0.622440622440615|
+--------------------+-----+--------------+--------------------+------------------
-----+-------------------------+---------------+--------------------+----------
--------------+----------------------------+----------------+--------------------
-+-------------------------+---------------------------+
only showing top 20 rows
```

[29]:
```
df_a.head(3)
```

[29]:
```
[Row(trade_time=datetime.datetime(2024, 6, 24, 8, 30, 46), match_number=1653,
instrument_id=5, timestamp=datetime.datetime(2024, 6, 24, 8, 30, 46, 497000),
volume=218, price=61.74, amount=13459.32, buyer_id=0, buyer_name='GS',
seller_id=113, seller_name='CITI', operation_type='C', concertation_type='O',
price_setter=1, lot=1, symbol='WALMEX*', last_day_close=None,
last_day_close_date=None, unitary_daily_variation=None,
percentage_daily_variation=None, last_week_close=None,
last_week_close_date=None, unitary_weekly_variation=None,
percentage_weekly_variation=None, last_month_close=None,
last_month_close_date=None, unitary_monthly_variation=None,
percentage_monthly_variation=None),
 Row(trade_time=datetime.datetime(2024, 6, 24, 9, 38, 59), match_number=3372,
instrument_id=5, timestamp=datetime.datetime(2024, 6, 24, 9, 38, 59, 345000),
volume=100, price=61.58, amount=6158.0, buyer_id=0, buyer_name='GS',
seller_id=123, seller_name='JPM', operation_type='C', concertation_type='O',
price_setter=1, lot=1, symbol='WALMEX*', last_day_close=61.74,
last_day_close_date=datetime.datetime(2024, 6, 24, 8, 30, 46),
unitary_daily_variation=-0.1600000000000037,
percentage_daily_variation=-0.25915127955944883, last_week_close=61.74,
last_week_close_date=datetime.datetime(2024, 6, 24, 8, 30, 46),
unitary_weekly_variation=-0.1600000000000037,
percentage_weekly_variation=-0.25915127955944883, last_month_close=61.74,
last_month_close_date=datetime.datetime(2024, 6, 24, 8, 30, 46),
unitary_monthly_variation=-0.1600000000000037,
percentage_monthly_variation=-0.25915127955944883),
 Row(trade_time=datetime.datetime(2024, 6, 24, 12, 3, 9), match_number=9387,
instrument_id=5, timestamp=datetime.datetime(2024, 6, 24, 12, 3, 9, 335000),
volume=500, price=60.81, amount=30405.0, buyer_id=123, buyer_name='JPM',
seller_id=54, seller_name='MERL', operation_type='C', concertation_type='O',
price_setter=1, lot=1, symbol='WALMEX*', last_day_close=61.58,
last_day_close_date=datetime.datetime(2024, 6, 24, 9, 38, 59),
unitary_daily_variation=-0.769999999999996,
percentage_daily_variation=-1.2504059759662165, last_week_close=61.58,
```

```
last_week_close_date=datetime.datetime(2024, 6, 24, 9, 38, 59),
unitary_weekly_variation=-0.769999999999996,
percentage_weekly_variation=-1.2504059759662165, last_month_close=61.58,
last_month_close_date=datetime.datetime(2024, 6, 24, 9, 38, 59),
unitary_monthly_variation=-0.769999999999996,
percentage_monthly_variation=-1.2504059759662165)]
```

[ ]:

```
[30]: df_a.tail(3)
```

```
[30]: [Row(trade_time=datetime.datetime(2024, 7, 5, 10, 38), match_number=3387,
instrument_id=5, timestamp=datetime.datetime(2024, 7, 5, 10, 38, 0, 501000),
volume=14, price=61.59, amount=862.26, buyer_id=0, buyer_name='FMX',
seller_id=54, seller_name='MERL', operation_type='C', concertation_type='O',
price_setter=0, lot=0, symbol='WALMEX*', last_day_close=61.36,
last_day_close_date=datetime.datetime(2024, 7, 5, 13, 41, 38),
unitary_daily_variation=0.23000000000000398,
percentage_daily_variation=0.3748370273794067, last_week_close=61.36,
last_week_close_date=datetime.datetime(2024, 7, 5, 13, 41, 38),
unitary_weekly_variation=0.23000000000000398,
percentage_weekly_variation=0.3748370273794067, last_month_close=61.36,
last_month_close_date=datetime.datetime(2024, 7, 5, 13, 41, 38),
unitary_monthly_variation=0.23000000000000398,
percentage_monthly_variation=0.3748370273794067),
 Row(trade_time=datetime.datetime(2024, 7, 5, 10, 59), match_number=3756,
instrument_id=5, timestamp=datetime.datetime(2024, 7, 5, 10, 59, 0, 301000),
volume=118, price=61.6, amount=7268.8, buyer_id=0, buyer_name='FMX',
seller_id=136, seller_name='MS', operation_type='C', concertation_type='O',
price_setter=1, lot=1, symbol='WALMEX*', last_day_close=61.59,
last_day_close_date=datetime.datetime(2024, 7, 5, 10, 38),
unitary_daily_variation=0.00999999999999801,
percentage_daily_variation=0.01623640201331062, last_week_close=61.59,
last_week_close_date=datetime.datetime(2024, 7, 5, 10, 38),
unitary_weekly_variation=0.00999999999999801,
percentage_weekly_variation=0.01623640201331062, last_month_close=61.59,
last_month_close_date=datetime.datetime(2024, 7, 5, 10, 38),
unitary_monthly_variation=0.00999999999999801,
percentage_monthly_variation=0.01623640201331062),
 Row(trade_time=datetime.datetime(2024, 7, 5, 13, 16, 11), match_number=7462,
instrument_id=5, timestamp=datetime.datetime(2024, 7, 5, 13, 16, 11, 354000),
volume=600, price=61.32, amount=36792.0, buyer_id=112, buyer_name='SCTIA',
seller_id=0, seller_name='FMX', operation_type='C', concertation_type='O',
price_setter=1, lot=1, symbol='WALMEX*', last_day_close=61.6,
last_day_close_date=datetime.datetime(2024, 7, 5, 10, 59),
unitary_daily_variation=-0.28000000000000114,
percentage_daily_variation=-0.45454545454545636, last_week_close=61.6,
```

```
last_week_close_date=datetime.datetime(2024, 7, 5, 10, 59),
unitary_weekly_variation=-0.28000000000000114,
percentage_weekly_variation=-0.45454545454545636, last_month_close=61.6,
last_month_close_date=datetime.datetime(2024, 7, 5, 10, 59),
unitary_monthly_variation=-0.28000000000000114,
percentage_monthly_variation=-0.45454545454545636)]
```

[ ]:

```

```

3 1. Análisis Exploratorio de Datos (EDA)

4 a. Distribución de Volúmenes y Precios.

5 b. Análisis por Casos de Bolsa (Buyers/Sellers).

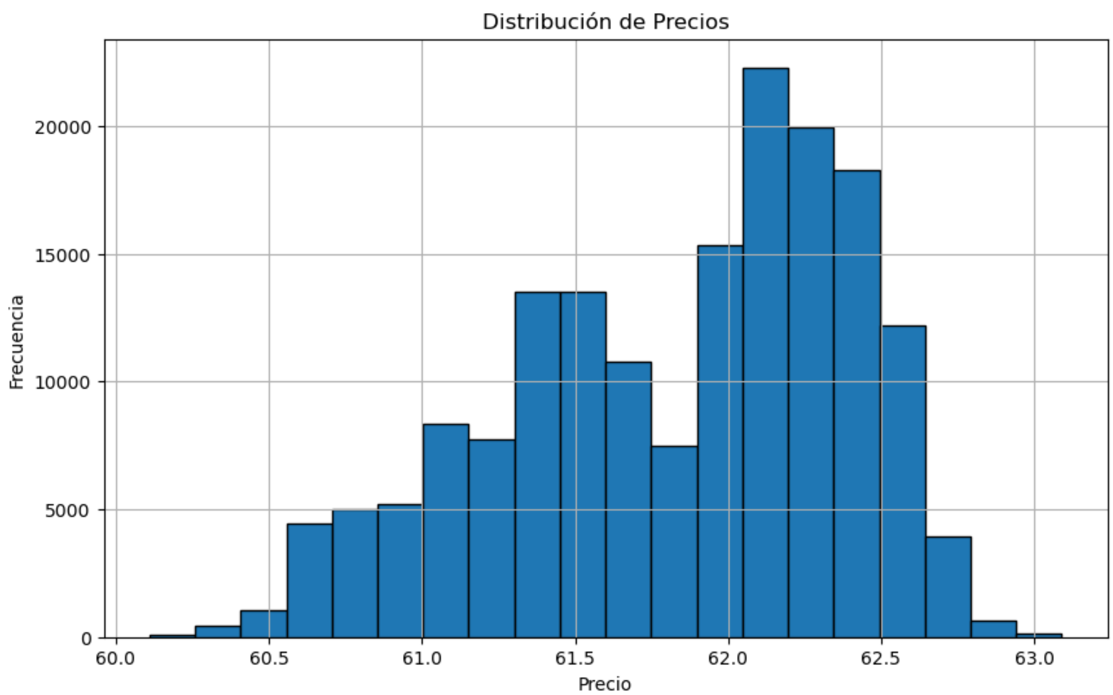
6 c. Análisis de Tiempos.

[31]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[32]:

```python
from pyspark.sql.functions import to_timestamp
from pyspark.sql.functions import col
from pyspark.sql.functions import to_date

# Convertir la columna trade_time a tipo timestamp en PySpark
df_a = df_a.withColumn('trade_time', to_timestamp('trade_time'))

# Convertir la columna 'price' a tipo double (si aún no lo es)
df_a = df_a.withColumn('price', col('price').cast('double'))

# Extraer la fecha de 'trade_time'
df_a = df_a.withColumn('date', to_date('trade_time'))

# Convertir el DataFrame de PySpark a pandas
df_pandas = df_a.select('price').toPandas()

# Graficamos la distribución de precios diarios
plt.figure(figsize=(10,6))
plt.hist(df_pandas['price'], bins=20, edgecolor='black')
plt.title('Distribución de Precios')
plt.xlabel('Precio')
plt.ylabel('Frecuencia')
plt.grid(True)
plt.show()
```



[33]:

```python
from pyspark.sql import functions as F
from pyspark.sql.functions import format_number

# Agrupamos los datos por 'buyer_name' y calculamos las agregaciones usando␣
↪PySpark, aplicamos format_number a las columnas numéricas

buyer_analysis = df_a.groupBy('buyer_name').agg(
    format_number(F.sum('volume'), 0).alias('total_volume'),
    format_number(F.avg('price'), 2).alias('average_price'),
    format_number(F.sum('amount'), 2).alias('total_amount'),
    F.count('match_number').alias('total_transactions')
)

# Mostrar el análisis de compradores
buyer_analysis.show()
```

```
+----------+------------+-------------+----------------+------------------+
|buyer_name|total_volume|average_price|    total_amount|total_transactions|
+----------+------------+-------------+----------------+------------------+
|       JPM|  31,712,592|        61.95|1,970,094,249.78|             33493|
|      CITI|   4,858,357|        62.17|  300,836,174.66|              4559|
|      SANT|   1,602,934|        61.89|   99,170,889.84|              1197|
|      NULL|     917,422|        61.70|   56,934,481.90|                 5|
|     BURSA|      61,726|        61.13|    3,770,799.97|                80|
|      BTGP|   1,809,221|        61.70|  111,712,806.20|              6954|
```

```
|     HSBCB|     238,259|        61.93|   14,796,770.07|                 349|
|      CICB|   2,520,950|        61.76|  155,738,621.44|                3260|
|       FMX|  44,257,504|        61.90|2,739,555,250.95|               37718|
|     BANOR|   1,810,217|        61.69|  111,422,807.29|                1534|
|        MS|  10,661,959|        61.53|  657,562,601.92|               24489|
|     MNXCB|      54,868|        61.78|    3,389,378.95|                 113|
|      ICAM|      20,390|        62.06|    1,259,821.33|                  25|
|     INVEX|  19,746,931|        62.14|1,221,800,236.36|                  63|
|       GBM|  39,463,375|        62.03|2,442,168,958.39|               22478|
|     MULVA|      12,959|        62.29|      808,344.13|                  42|
|     PUNTO|     956,364|        61.66|   59,126,860.91|                  78|
|      BARC|      32,534|        61.88|    2,017,303.50|                  48|
|        GS|   4,269,537|        61.81|  263,801,387.96|               15203|
|     VECTO|   1,018,680|        61.58|   62,678,206.71|                1828|
+----------+------------+-------------+----------------+------------------+
only showing top 20 rows
```

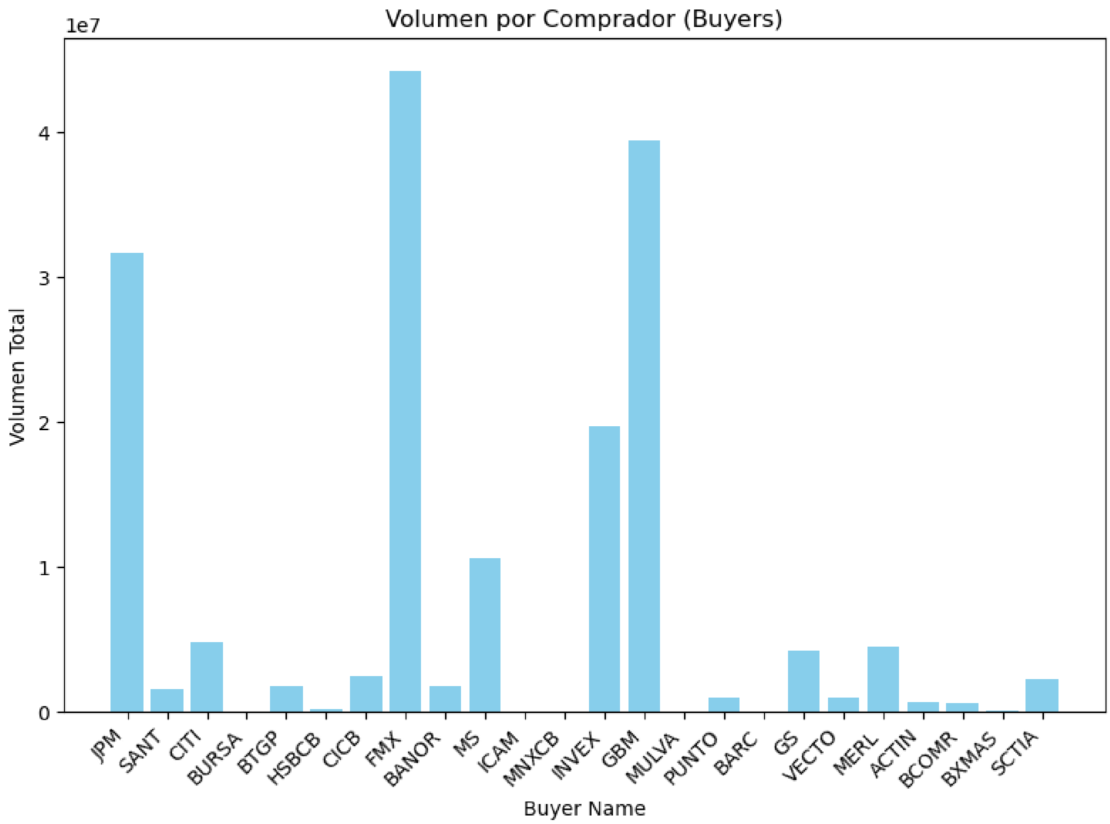
[34]:

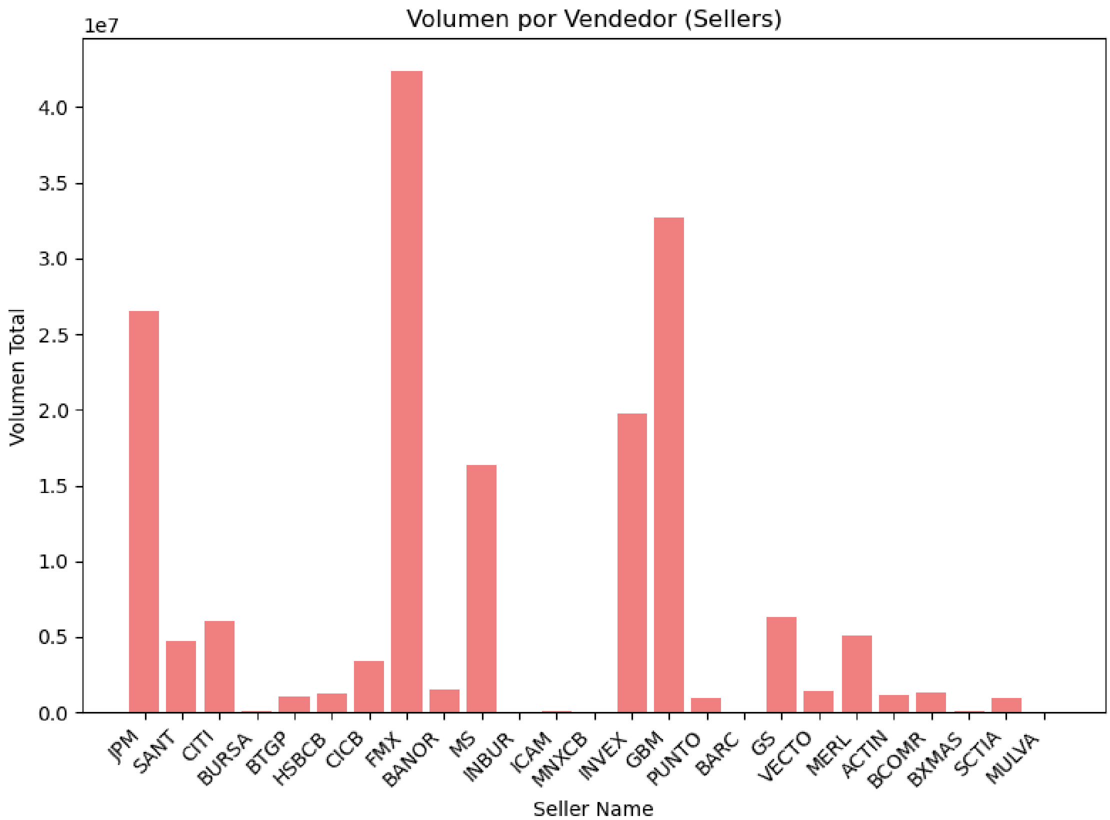
```python
# Agrupamos los datos por 'buyer_name' y calculamos las agregaciones usando
 ↪PySpark, aplicamos format_number a las columnas numéricas
seller_analysis = df_a.groupBy('seller_name').agg(
    format_number(F.sum('volume'), 0).alias('total_volume'),
    format_number(F.avg('price'), 2).alias('average_price'),
    format_number(F.sum('amount'), 2).alias('total_amount'),
    F.count('match_number').alias('total_transactions')
)

# Mostrar el análisis de vendedores
seller_analysis.show()
```

```
+-----------+------------+-------------+----------------+------------------+
|seller_name|total_volume|average_price|    total_amount|total_transactions|
+-----------+------------+-------------+----------------+------------------+
|        JPM|  26,546,600|        61.95|1,649,051,712.87|               28914|
|       SANT|   4,703,859|        61.50|  290,176,756.05|                4343|
|       CITI|   6,019,845|        61.74|  371,860,151.85|                9056|
|       NULL|     917,422|        61.70|   56,934,481.90|                   5|
|      BURSA|      81,997|        61.88|    5,073,615.90|                  99|
|       BTGP|   1,061,964|        61.85|   65,712,814.42|                2892|
|      HSBCB|   1,232,869|        61.78|   76,344,511.04|                4120|
|       CICB|   3,428,669|        61.77|  212,240,285.55|                4514|
|        FMX|  42,391,627|        61.83|2,622,571,502.23|               34000|
|      BANOR|   1,531,072|        61.86|   94,459,517.49|                1256|
|         MS|  16,395,587|        61.97|1,016,078,868.01|               37801|
|      INBUR|      16,748|        61.13|    1,022,290.40|                  26|
|       ICAM|     101,139|        61.77|    6,247,182.65|                 209|
|      MNXCB|      29,098|        62.51|    1,820,436.34|                  33|
|      INVEX|  19,724,062|        61.88|1,220,381,175.12|                 472|
```

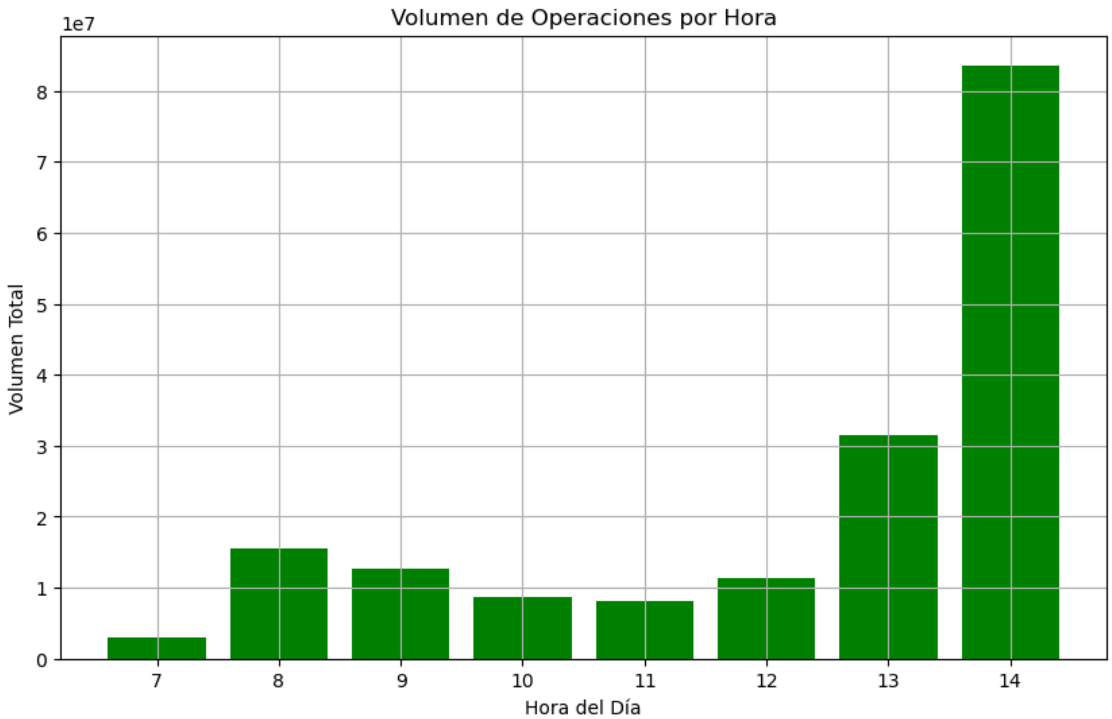
```
|         GBM|  32,698,808|        61.70|2,022,069,338.99|             8139|
|       MULVA|         269|        62.04|       16,695.43|                7|
|       PUNTO|     939,058|        61.71|   58,061,904.09|              293|
|        BARC|       9,844|        62.50|      615,251.98|               58|
|          GS|   6,297,305|        61.74|  389,987,988.83|            13476|
+------------+------------+-------------+----------------+-----------------+
only showing top 20 rows
```

[35]:
```python
from pyspark.sql import functions as F
import matplotlib.pyplot as plt

# Agrupar los datos por 'buyer_name' y 'seller_name' y sumar el volumen
buyer_volume = df_a.groupBy('buyer_name').agg(F.sum('volume').alias('volume')).
 ↪filter(F.col('buyer_name').isNotNull()).toPandas()
seller_volume = df_a.groupBy('seller_name').agg(F.sum('volume').
 ↪alias('volume')).filter(F.col('seller_name').isNotNull()).toPandas()

# Gráfica para el volumen por comprador (Buyers)
plt.figure(figsize=(8, 6))
plt.bar(buyer_volume['buyer_name'], buyer_volume['volume'], color='skyblue')
plt.title('Volumen por Comprador (Buyers)')
plt.xlabel('Buyer Name')
plt.ylabel('Volumen Total')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

# Gráfica para el volumen por vendedor (Sellers)
plt.figure(figsize=(8, 6))
plt.bar(seller_volume['seller_name'], seller_volume['volume'],␣
 ↪color='lightcoral')
plt.title('Volumen por Vendedor (Sellers)')
plt.xlabel('Seller Name')
plt.ylabel('Volumen Total')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

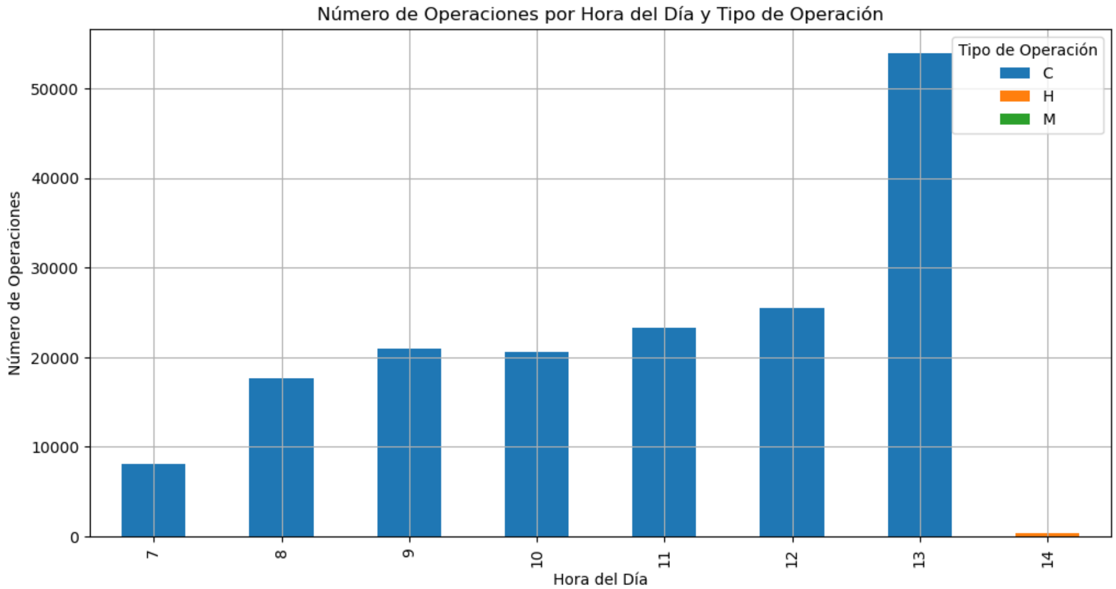

Volumen por Comprador (Buyers)
1e7
4
3
2
1
0
Volumen Total
JPM
SANT
CITI
BURSA
BTGP
HSBCB
CICB
FMX
BANOR
MS
ICAM
MNXCB
INVEX
GBM
MULVA
PUNTO
BARC
GS
VECTO
MERL
ACTIN
BCOMR
BXMAS
SCTIA
Buyer Name



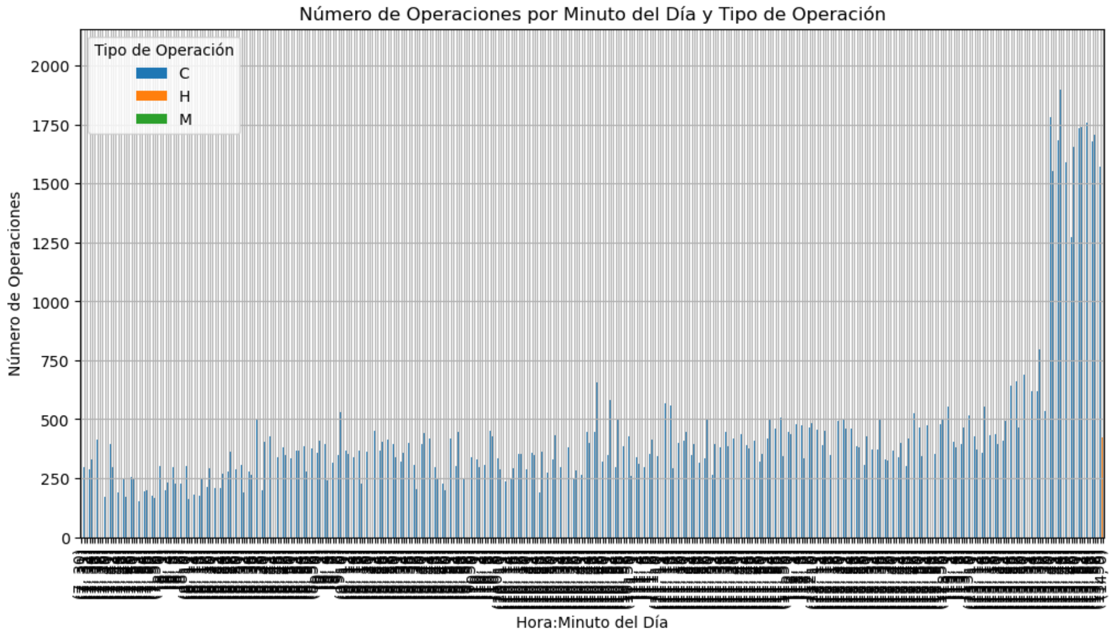
[36]:
```
from pyspark.sql.functions import hour

# Crear una nueva columna 'hour' extrayendo la hora de 'trade_time'
df_a = df_a.withColumn('hour', hour('trade_time'))

# Agrupar por la columna 'hour' y sumar los volúmenes
hourly_volume = df_a.groupBy('hour').agg(F.sum('volume').alias('total_volume')).
↪orderBy('hour').toPandas()

# Graficar el volumen por hora
plt.figure(figsize=(10,6))
plt.bar(hourly_volume['hour'], hourly_volume['total_volume'], color='green')
plt.title('Volumen de Operaciones por Hora')
plt.xlabel('Hora del Día')
plt.ylabel('Volumen Total')
plt.grid(True)
plt.show()
```



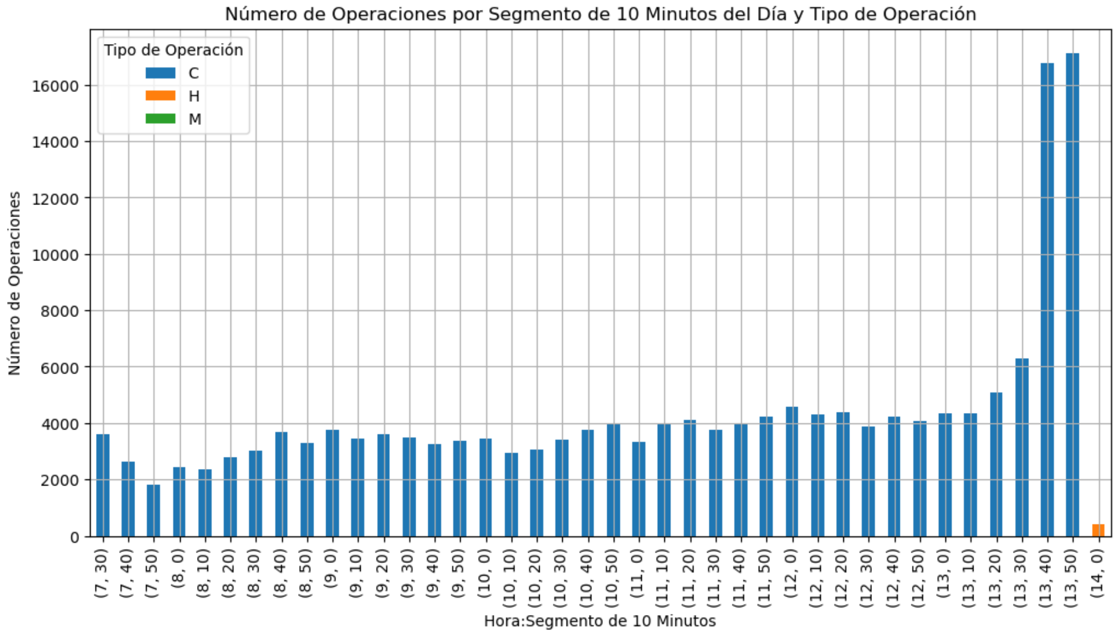
[37]:
```
# Agrupar los datos por 'operation_type' y 'hour' y contar las ocurrencias
operation_hour_analysis = df_a.groupBy('operation_type', 'hour').count().
 ↪orderBy('hour').toPandas()

# Crear una gráfica de barras apiladas para mostrar el número de operaciones␣
 ↪por tipo y hora del día
operation_hour_pivot = operation_hour_analysis.pivot(index='hour',␣
 ↪columns='operation_type', values='count').fillna(0)

# Graficar el análisis por hora del día y tipo de operación
operation_hour_pivot.plot(kind='bar', stacked=True, figsize=(12,6))
plt.title('Número de Operaciones por Hora del Día y Tipo de Operación')
plt.xlabel('Hora del Día')
plt.ylabel('Número de Operaciones')
plt.legend(title='Tipo de Operación')
plt.grid(True)
plt.show()
```



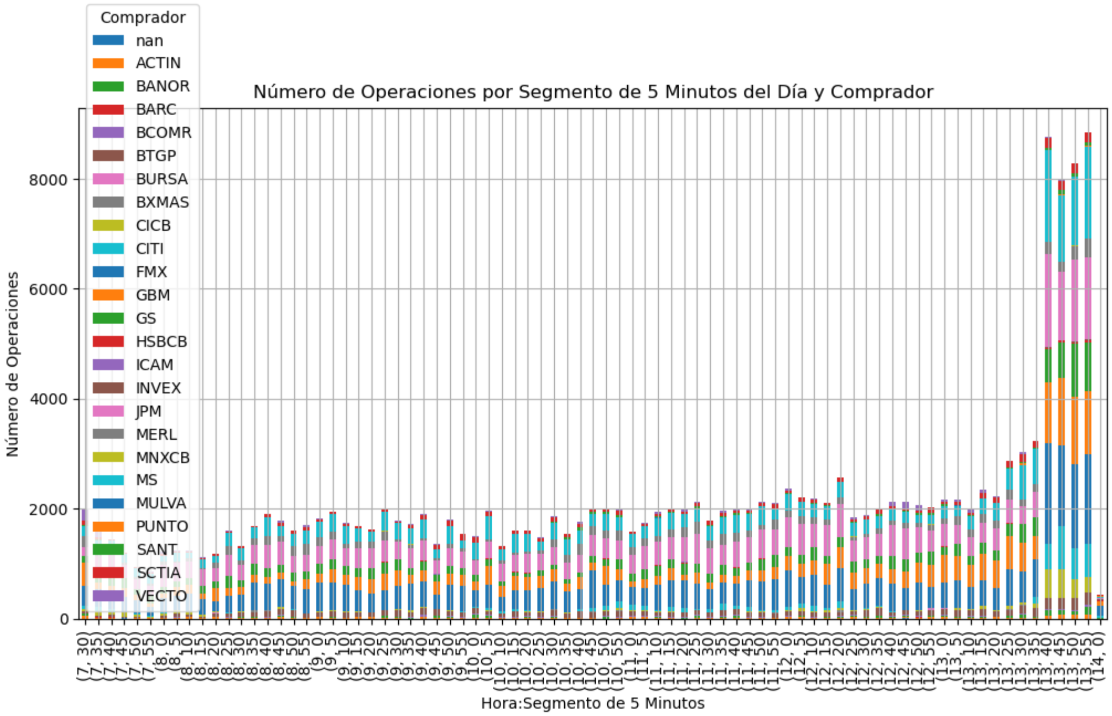
[38]:
```
from pyspark.sql.functions import minute

# Crear una nueva columna 'minute' extrayendo el minuto de 'trade_time'
df_a = df_a.withColumn('minute', minute('trade_time'))

# Agrupar los datos por 'operation_type' y 'minute' y contar las ocurrencias
operation_minute_analysis = df_a.groupBy('operation_type', 'hour', 'minute').
 ↪count().orderBy('hour', 'minute').toPandas()

# Crear una tabla pivote para organizar los datos para el gráfico
operation_minute_pivot = operation_minute_analysis.pivot(index=['hour',␣
 ↪'minute'], columns='operation_type', values='count').fillna(0)

# Graficar el análisis por minuto del día y tipo de operación
operation_minute_pivot.plot(kind='bar', stacked=True, figsize=(12,6))
plt.title('Número de Operaciones por Minuto del Día y Tipo de Operación')
plt.xlabel('Hora:Minuto del Día')
plt.ylabel('Número de Operaciones')
plt.legend(title='Tipo de Operación')
plt.grid(True)
plt.show()
```



[39]:
```
from pyspark.sql.functions import minute, floor

# Crear una nueva columna 'minute_segment' para agrupar por intervalos de 10␣
↪minutos
df_a = df_a.withColumn('minute_segment', (floor(minute('trade_time') / 10) *␣
↪10))

# Agrupar los datos por 'operation_type', 'hour' y el segmento de minutos
operation_minute_segment_analysis = df_a.groupBy('operation_type', 'hour',␣
↪'minute_segment').count().orderBy('hour', 'minute_segment').toPandas()

# Crear una tabla pivote para organizar los datos para el gráfico
operation_minute_segment_pivot = operation_minute_segment_analysis.
↪pivot(index=['hour', 'minute_segment'], columns='operation_type',␣
↪values='count').fillna(0)

# Graficar el análisis por segmento de 10 minutos del día y tipo de operación
operation_minute_segment_pivot.plot(kind='bar', stacked=True, figsize=(12,6))
plt.title('Número de Operaciones por Segmento de 10 Minutos del Día y Tipo de␣
↪Operación')
plt.xlabel('Hora:Segmento de 10 Minutos')
plt.ylabel('Número de Operaciones')
plt.legend(title='Tipo de Operación')
plt.grid(True)
plt.show()
```

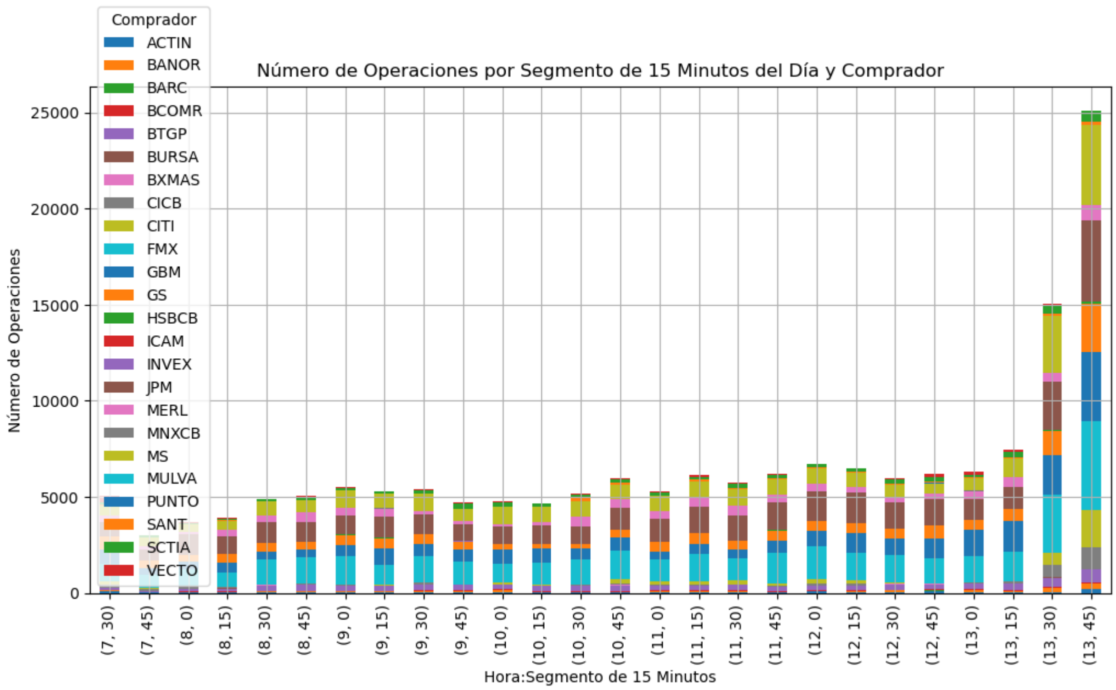


[40]:
```python
from pyspark.sql.functions import minute, floor

# Crear una nueva columna 'minute_segment' para agrupar en intervalos de 5␣
↪minutos
df_a = df_a.withColumn('minute_segment', (floor(minute('trade_time') / 5) * 5))

# Agrupar los datos por 'buyer_name', 'hour', y el segmento de 5 minutos
buyer_minute_segment_analysis = df_a.groupBy('buyer_name', 'hour',␣
↪'minute_segment').count().orderBy('hour', 'minute_segment').toPandas()

# Crear una tabla pivote para organizar los datos para el gráfico
buyer_minute_segment_pivot = buyer_minute_segment_analysis.pivot(index=['hour',␣
↪'minute_segment'], columns='buyer_name', values='count').fillna(0)

# Graficar el análisis por segmento de 5 minutos del día y comprador
buyer_minute_segment_pivot.plot(kind='bar', stacked=True, figsize=(12,6))
plt.title('Número de Operaciones por Segmento de 5 Minutos del Día y Comprador')
plt.xlabel('Hora:Segmento de 5 Minutos')
plt.ylabel('Número de Operaciones')
plt.legend(title='Comprador')
plt.grid(True)
plt.show()
```



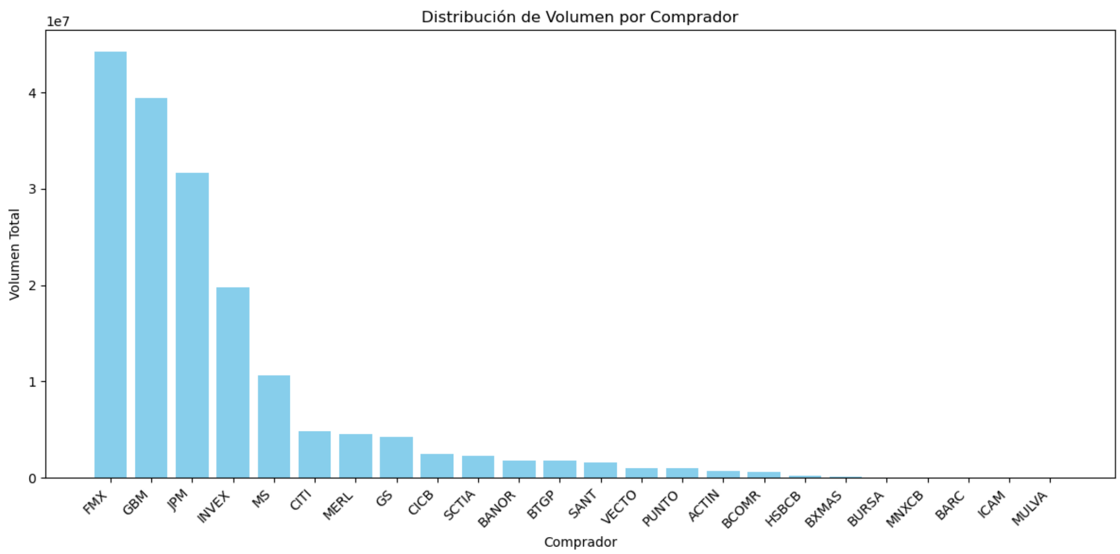
[41]:
```python
from pyspark.sql.functions import minute, floor, col

# Filtrar los datos para incluir solo registros donde 'operation_type' sea "C"
filtered_df = df_a.filter(col('operation_type') == 'C')

# Crear una nueva columna 'minute_segment' para agrupar en intervalos de 15 minutos
filtered_df = filtered_df.withColumn('minute_segment', (floor(minute('trade_time') / 15) * 15))

# Agrupar los datos por 'buyer_name', 'hour', y el segmento de 15 minutos y sumar las transacciones
buyer_minute_segment_analysis = filtered_df.groupBy('buyer_name', 'hour', 'minute_segment') \
    .agg(F.count('buyer_name').alias('transaction_count')).orderBy('hour', 'minute_segment').toPandas()

# Crear una tabla pivote para organizar los datos para el gráfico de compradores
buyer_minute_segment_pivot = buyer_minute_segment_analysis.pivot_table(index=['hour', 'minute_segment'], columns='buyer_name', values='transaction_count', aggfunc='sum').fillna(0)

# Graficar el análisis por segmento de 15 minutos del día y comprador
```

```
buyer_minute_segment_pivot.plot(kind='bar', stacked=True, figsize=(12,6))
plt.title('Número de Operaciones por Segmento de 15 Minutos del Día y␣
↪Comprador')
plt.xlabel('Hora:Segmento de 15 Minutos')
plt.ylabel('Número de Operaciones')
plt.legend(title='Comprador')
plt.grid(True)
plt.show()
```



[42]:
```
import matplotlib.pyplot as plt
import pandas as pd
from pyspark.sql import functions as F

# Agrupar por 'buyer_name' y sumar el volumen, filtrando valores nulos en␣
↪'buyer_name'
buyer_volume_df = df_a.filter(F.col('buyer_name').isNotNull()).
↪groupBy('buyer_name').agg(F.sum('volume').alias('total_volume'))

# Convertir a Pandas para facilitar la visualización
buyer_volume_pd = buyer_volume_df.orderBy(F.desc('total_volume')).toPandas()

# Crear el gráfico de barras
plt.figure(figsize=(12, 6))
```

```
plt.bar(buyer_volume_pd['buyer_name'], buyer_volume_pd['total_volume'],␣
 ↪color='skyblue')
plt.title('Distribución de Volumen por Comprador')
plt.xlabel('Comprador')
plt.ylabel('Volumen Total')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()

# Mostrar el gráfico
plt.show()
```



[43]:
```
from pyspark.sql import functions as F
from pyspark.sql.window import Window

# 1. Registro de transacciones (por casa de bolsa)
# Agrupar transacciones por 'buyer_name', 'seller_name', 'symbol', y fecha,␣
 ↪sumando el volumen
transactions_df = df_a.groupBy('buyer_name', 'seller_name', 'symbol', F.
 ↪to_date('trade_time').alias('trade_date')) \
    .agg(F.sum('volume').alias('total_volume'),
         F.avg('price').alias('avg_price'),
         F.sum(F.col('price') * F.col('volume')).alias('total_amount'))

# 2. Calcular precio de cierre por día
# Definir una ventana para obtener el precio de cierre (última transacción del␣
 ↪día)
day_window = Window.partitionBy('symbol', F.to_date('trade_time')).orderBy(F.
 ↪col('trade_time').desc())
```

```
# Obtener el precio de cierre del día para cada transacción
df_a = df_a.withColumn('closing_price', F.first('price').over(day_window))

# 3. Comparar con el precio de cierre y calcular ganancias/pérdidas
# Calcular ganancia/pérdida por transacción comparando el precio de compra/
↪venta con el precio de cierre
df_a = df_a.withColumn('gain_loss', F.when(F.col('buyer_name').isNotNull(),
                                           (F.col('closing_price') - F.
↪col('price')) * F.col('volume')) \
                                   .otherwise((F.col('price') - F.
↪col('closing_price')) * F.col('volume')))

# 4. Control diario por casa de bolsa
# Resumir ganancias/pérdidas y calcular la precisión respecto al precio de␣
↪cierre
daily_summary = df_a.groupBy('buyer_name', F.to_date('trade_time').
↪alias('trade_date')) \
    .agg(F.sum('gain_loss').alias('total_gain_loss'),
         F.avg(F.abs(F.col('price') - F.col('closing_price'))).
↪alias('avg_price_diff'))

# Mostrar el resumen diario de las casas de bolsa
# daily_summary.show()
```

[44]:

```
from pyspark.sql import functions as F
from pyspark.sql.window import Window

# Crear una ventana para obtener el precio de cierre (última transacción del␣
↪día)
day_window = Window.partitionBy('symbol', F.to_date('trade_time')).orderBy(F.
↪col('trade_time').desc())

# Obtener el precio de cierre del día
df_a = df_a.withColumn('closing_price', F.first('price').over(day_window))

# Calcular las ganancias/pérdidas por transacción
# Para compradores: (precio de cierre - precio de compra) * volumen
# Para vendedores: (precio de venta - precio de cierre) * volumen
df_a = df_a.withColumn('gain_loss',
    F.when(F.col('buyer_name').isNotNull(),
           (F.col('closing_price') - F.col('price')) * F.col('volume')) \
    .otherwise((F.col('price') - F.col('closing_price')) * F.col('volume'))
)
```

[45]:
```
# Ganancias netas diarias por casa de bolsa
daily_net_gain = df_a.groupBy('buyer_name', F.to_date('trade_time').
↪alias('trade_date')) \
    .agg(F.sum('gain_loss').alias('net_gain_loss'))

# Mostrar resultados diarios
# daily_net_gain.show()
```

[46]:
```
# Agregar semana de transacción
df_a = df_a.withColumn('week_of_year', F.weekofyear('trade_time'))

# Ganancias netas semanales por casa de bolsa
weekly_net_gain = df_a.groupBy('buyer_name', 'week_of_year') \
    .agg(F.sum('gain_loss').alias('net_gain_loss'))

# Mostrar resultados semanales
# weekly_net_gain.show()
```

[47]:
```
# Agregar mes de transacción
df_a = df_a.withColumn('month_of_year', F.month('trade_time'))

# Ganancias netas mensuales por casa de bolsa
monthly_net_gain = df_a.groupBy('buyer_name', 'month_of_year') \
    .agg(F.sum('gain_loss').alias('net_gain_loss'))

# Mostrar resultados mensuales
# monthly_net_gain.show()
```

[48]:
```
df_a.head(3)
```

[48]:
```
[Row(trade_time=datetime.datetime(2024, 6, 24, 14, 0, 5), match_number=17219,
instrument_id=5, timestamp=datetime.datetime(2024, 6, 24, 14, 0, 5, 223000),
volume=7000, price=60.64, amount=424480.0, buyer_id=54, buyer_name='MERL',
seller_id=28, seller_name='INVEX', operation_type='H', concertation_type='H',
price_setter=0, lot=0, symbol='WALMEX*', last_day_close=60.71,
last_day_close_date=datetime.datetime(2024, 6, 24, 12, 53, 28),
unitary_daily_variation=-0.07000000000000028,
percentage_daily_variation=-0.11530225662988022, last_week_close=60.71,
last_week_close_date=datetime.datetime(2024, 6, 24, 12, 53, 28),
unitary_weekly_variation=-0.07000000000000028,
percentage_weekly_variation=-0.11530225662988022, last_month_close=60.71,
last_month_close_date=datetime.datetime(2024, 6, 24, 12, 53, 28),
unitary_monthly_variation=-0.07000000000000028,
percentage_monthly_variation=-0.11530225662988022, date=datetime.date(2024, 6,
24), hour=14, minute=0, minute_segment=0, closing_price=60.64, gain_loss=0.0,
week_of_year=26, month_of_year=6),
 Row(trade_time=datetime.datetime(2024, 6, 24, 14, 0, 5), match_number=17230,
```

```
instrument_id=5, timestamp=datetime.datetime(2024, 6, 24, 14, 0, 5, 307000),
volume=2900, price=60.64, amount=175856.0, buyer_id=54, buyer_name='MERL',
seller_id=0, seller_name='FMX', operation_type='H', concertation_type='H',
price_setter=0, lot=0, symbol='WALMEX*', last_day_close=61.72,
last_day_close_date=datetime.datetime(2024, 6, 24, 7, 35, 14),
unitary_daily_variation=-1.0799999999999983,
percentage_daily_variation=-1.7498379779650006, last_week_close=61.72,
last_week_close_date=datetime.datetime(2024, 6, 24, 7, 35, 14),
unitary_weekly_variation=-1.0799999999999983,
percentage_weekly_variation=-1.7498379779650006, last_month_close=61.72,
last_month_close_date=datetime.datetime(2024, 6, 24, 7, 35, 14),
unitary_monthly_variation=-1.0799999999999983,
percentage_monthly_variation=-1.7498379779650006, date=datetime.date(2024, 6,
24), hour=14, minute=0, minute_segment=0, closing_price=60.64, gain_loss=0.0,
week_of_year=26, month_of_year=6),
 Row(trade_time=datetime.datetime(2024, 6, 24, 14, 0, 5), match_number=17235,
instrument_id=5, timestamp=datetime.datetime(2024, 6, 24, 14, 0, 5, 308000),
volume=412, price=60.64, amount=24983.68, buyer_id=113, buyer_name='CITI',
seller_id=136, seller_name='MS', operation_type='H', concertation_type='H',
price_setter=0, lot=0, symbol='WALMEX*', last_day_close=60.55,
last_day_close_date=datetime.datetime(2024, 6, 24, 13, 6, 39),
unitary_daily_variation=0.09000000000000341,
percentage_daily_variation=0.14863748967795773, last_week_close=60.55,
last_week_close_date=datetime.datetime(2024, 6, 24, 13, 6, 39),
unitary_weekly_variation=0.09000000000000341,
percentage_weekly_variation=0.14863748967795773, last_month_close=60.55,
last_month_close_date=datetime.datetime(2024, 6, 24, 13, 6, 39),
unitary_monthly_variation=0.09000000000000341,
percentage_monthly_variation=0.14863748967795773, date=datetime.date(2024, 6,
24), hour=14, minute=0, minute_segment=0, closing_price=60.64, gain_loss=0.0,
week_of_year=26, month_of_year=6)]
```

[49]:

```python
import matplotlib.pyplot as plt
import pandas as pd
from pyspark.sql import functions as F

# Añadir nuevas columnas basadas en la fecha, hora y día de la semana (en
 ↪formato numérico)
df_a = df_a.withColumn('trade_date', F.to_date('trade_time')) \
           .withColumn('hour', F.hour('trade_time')) \
           .withColumn('minute', F.minute('trade_time')) \
           .withColumn('day_of_week_num', (F.dayofweek('trade_time') - 1))  #
 ↪Ajustamos para que lunes sea 0

# Seleccionar las columnas numéricas relevantes para el análisis, incluyendo el
 ↪día de la semana numérico
```

```python
relevant_columns = ['price', 'volume', 'amount', 'hour', 'minute',␣
 ↪'day_of_week_num']

# Crear una matriz de covarianza vacía
covariance_matrix = {}

# Calcular la covarianza entre cada par de columnas numéricas
for col1 in relevant_columns:
    covariance_matrix[col1] = {}
    for col2 in relevant_columns:
        covariance = df_a.stat.cov(col1, col2)
        covariance_matrix[col1][col2] = covariance

# Convertir la matriz de covarianza a un DataFrame de pandas para mostrarla
covariance_df = pd.DataFrame(covariance_matrix)

# Mostrar la matriz de covarianza
print("Matriz de Covarianza:")
print(covariance_df)
```

```
Matriz de Covarianza:
                     price        volume        amount          hour  \
price             0.313906  6.359797e+01  4.241208e+03     -0.010626
volume           63.597972  1.056766e+09  6.553177e+10   1442.317562
amount         4241.208219  6.553177e+10  4.063971e+12  89167.551529
hour             -0.010626  1.442318e+03  8.916755e+04      3.731503
minute            0.531167 -1.648548e+04 -1.019430e+06      2.941392
day_of_week_num   0.228840 -5.115830e+01 -3.099533e+03     -0.117767

                     minute  day_of_week_num
price          5.311667e-01         0.228840
volume        -1.648548e+04       -51.158304
amount        -1.019430e+06     -3099.532806
hour           2.941392e+00        -0.117767
minute         2.992616e+02         0.357828
day_of_week_num  3.578283e-01         1.824068
```

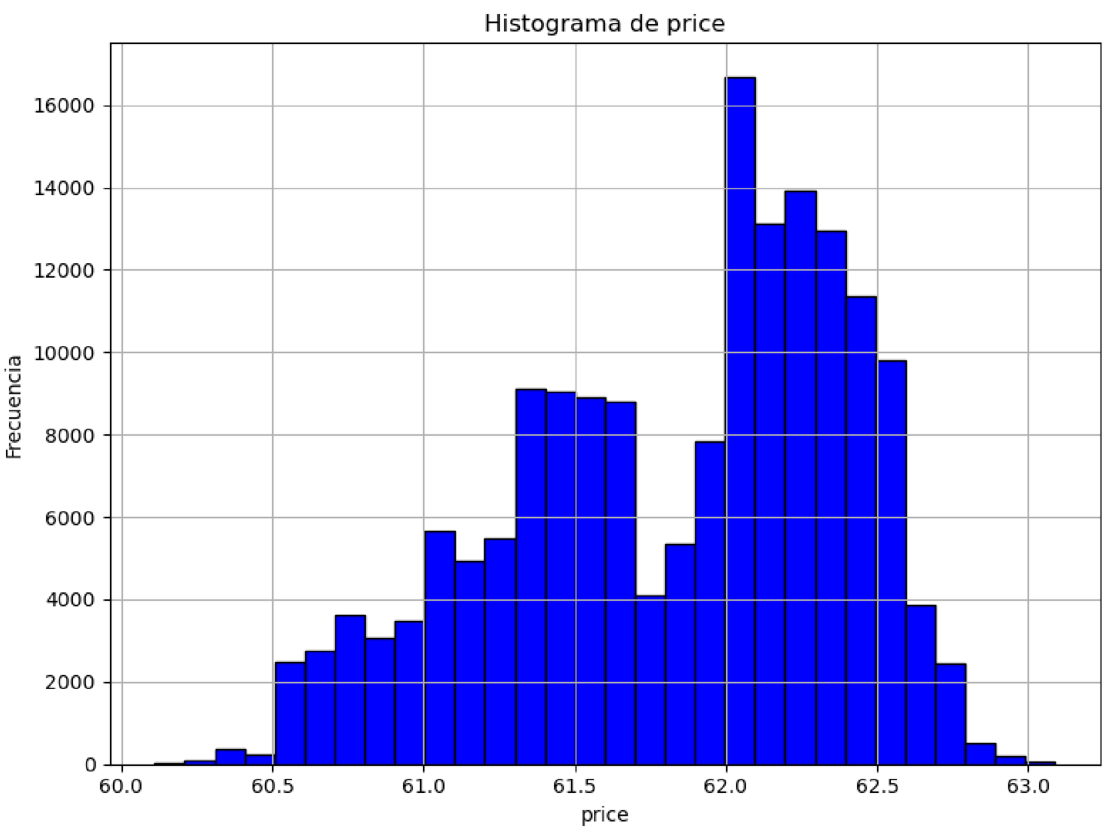
[ ]:

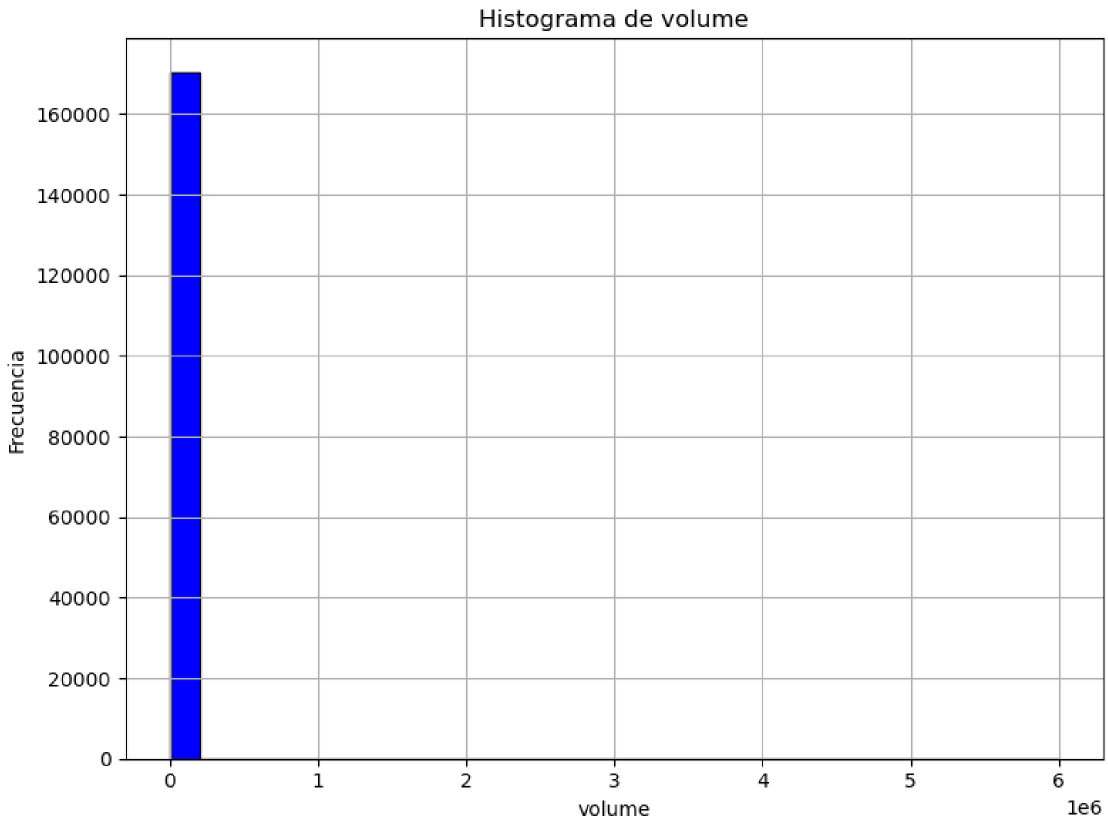
```python
```

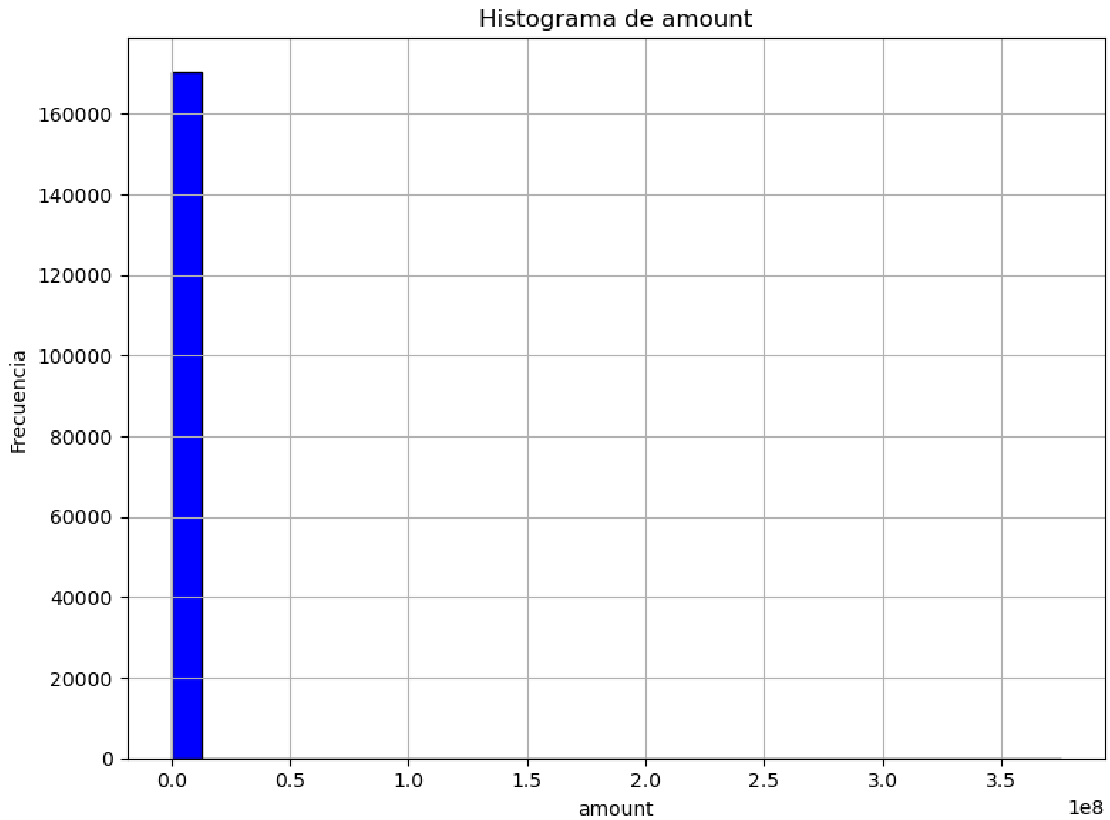
[50]:

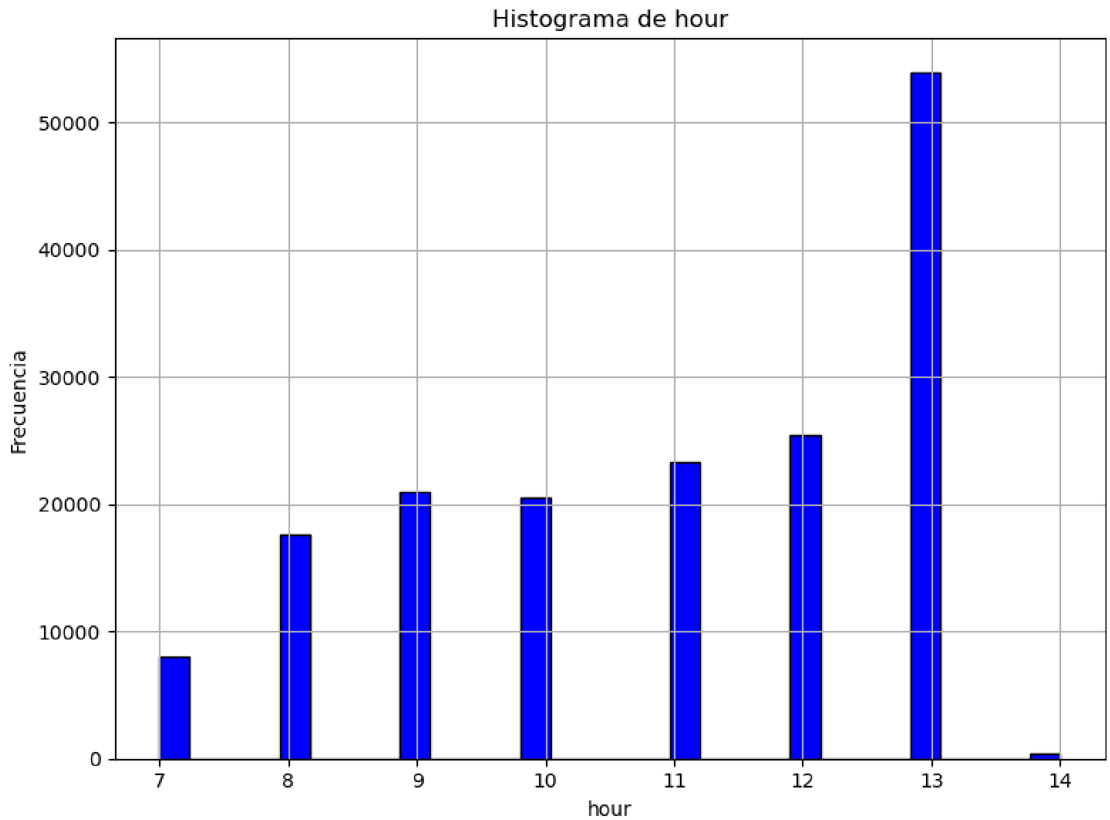
```python
import matplotlib.pyplot as plt
from pyspark.sql import functions as F

# Añadir nuevas columnas basadas en la fecha y hora
df_a = df_a.withColumn('trade_date', F.to_date('trade_time')) \
           .withColumn('day_of_week', F.date_format('trade_time', 'E')) \
           .withColumn('hour', F.hour('trade_time')) \
           .withColumn('minute', F.minute('trade_time'))
```

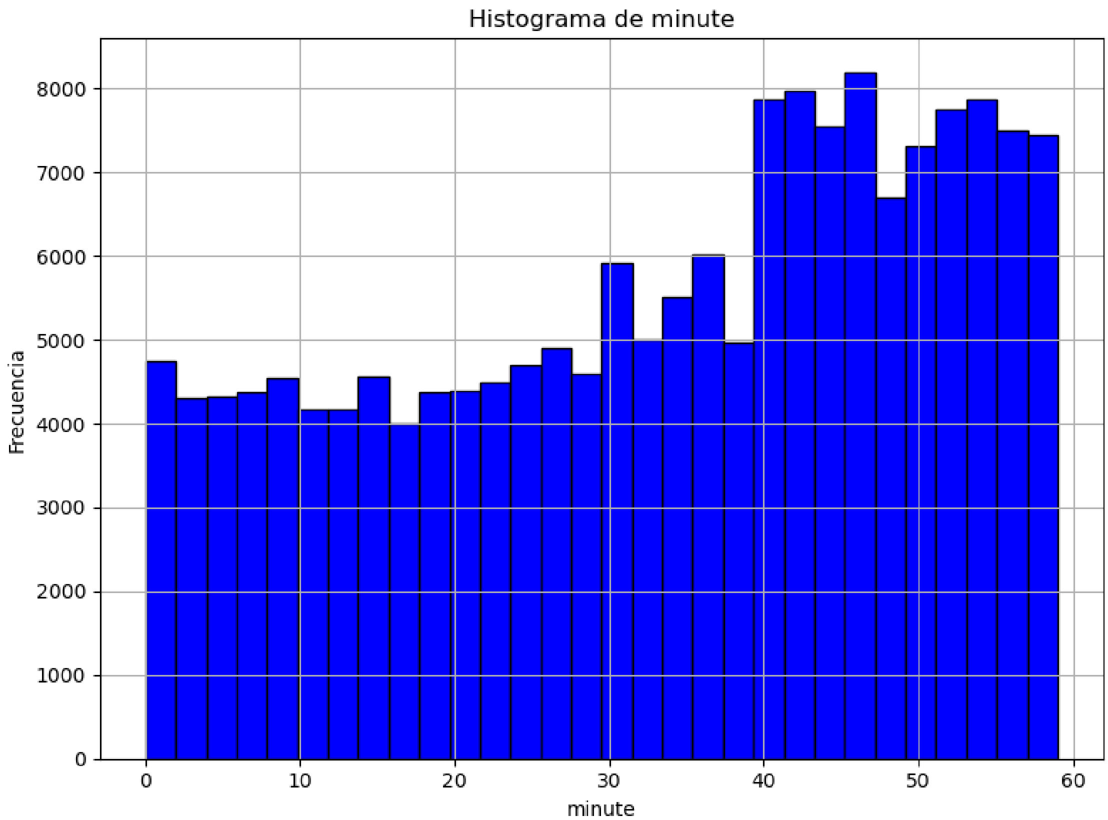
```python
# Seleccionar las columnas relevantes para el análisis
relevant_columns = ['price', 'volume', 'amount', 'trade_date', 'day_of_week',␣
↪'hour', 'minute']

# Convertir el DataFrame de PySpark a un DataFrame de pandas
df_pandas = df_a.select(relevant_columns).toPandas()

# Crear histogramas para las variables cuantitativas
quantitative_columns = ['price', 'volume', 'amount', 'hour', 'minute']
for column in quantitative_columns:
    plt.figure(figsize=(8, 6))
    plt.hist(df_pandas[column].dropna(), bins=30, color='blue',␣
↪edgecolor='black')
    plt.title(f'Histograma de {column}')
    plt.xlabel(column)
    plt.ylabel('Frecuencia')
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```




Histograma de volume
Frecuencia
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
2
3
4
5
6
1e6
volume

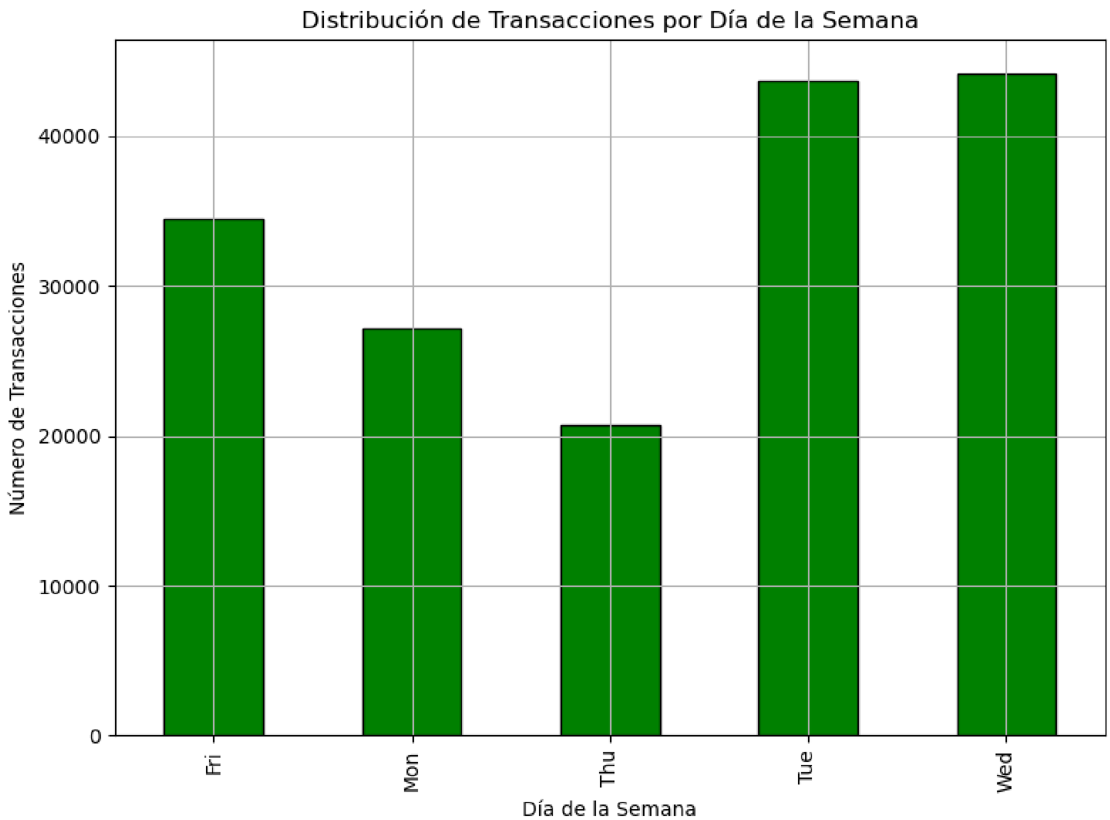

Histograma de amount
Frecuencia
amount
1e8
0
20000
40000
60000
80000
100000
120000
140000
160000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5


Histograma de hour
Frecuencia
0
10000
20000
30000
40000
50000
7
8
9
10
11
12
13
14
hour

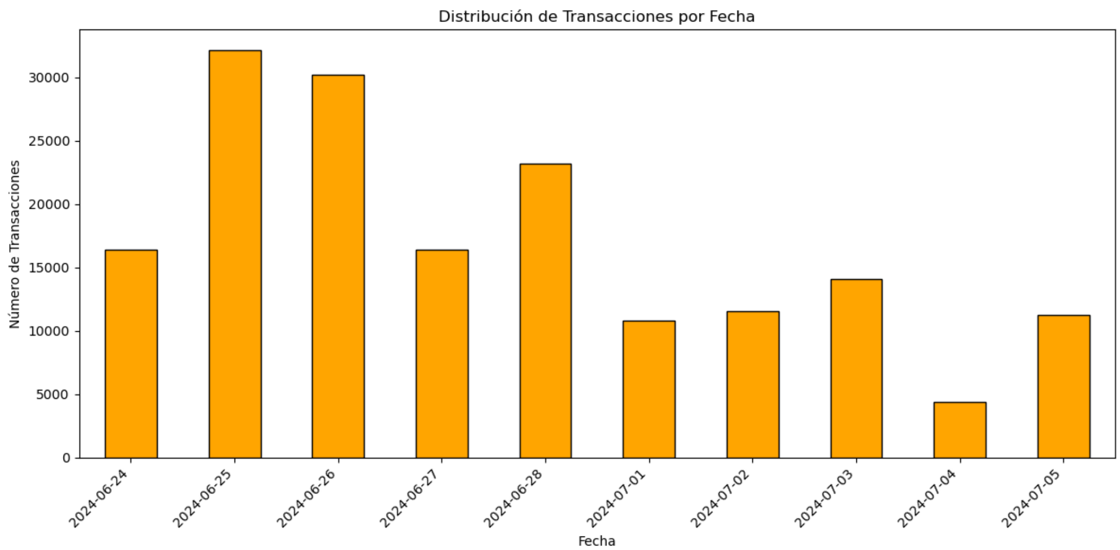


[51]:

```
# Crear gráfico de barras para la distribución por día de la semana
plt.figure(figsize=(8, 6))
df_pandas['day_of_week'].value_counts().sort_index().plot(kind='bar',␣
 ↪color='green', edgecolor='black')
plt.title('Distribución de Transacciones por Día de la Semana')
plt.xlabel('Día de la Semana')
plt.ylabel('Número de Transacciones')
plt.grid(True)
plt.tight_layout()
plt.show()
```



[52]:
```python
# Crear gráfico de barras para la distribución por fecha
plt.figure(figsize=(12, 6))
df_pandas['trade_date'].value_counts().sort_index().plot(kind='bar',␣
↪color='orange', edgecolor='black')
plt.title('Distribución de Transacciones por Fecha')
plt.xlabel('Fecha')
plt.ylabel('Número de Transacciones')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

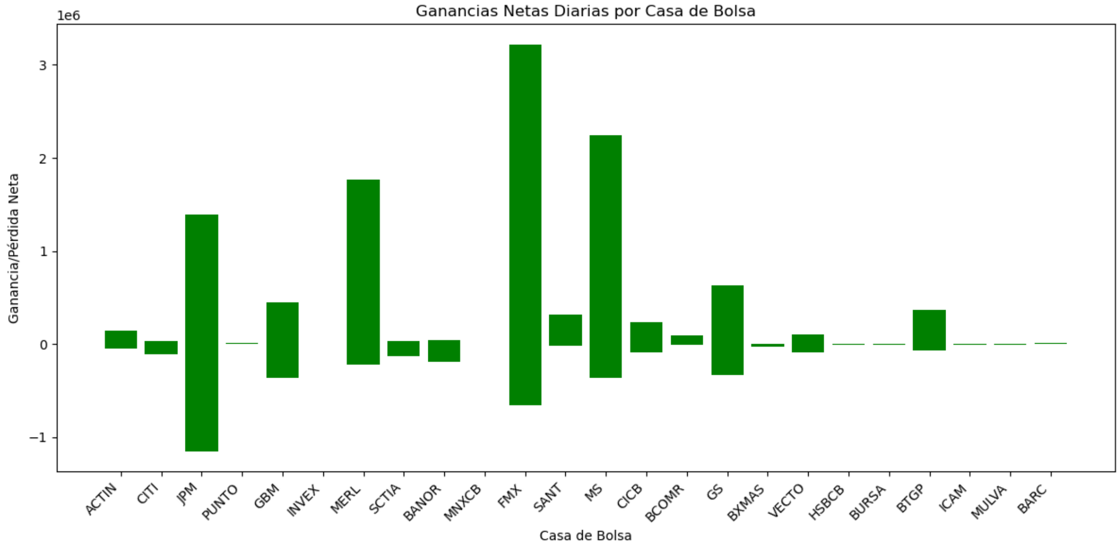


[ ]:

```

```

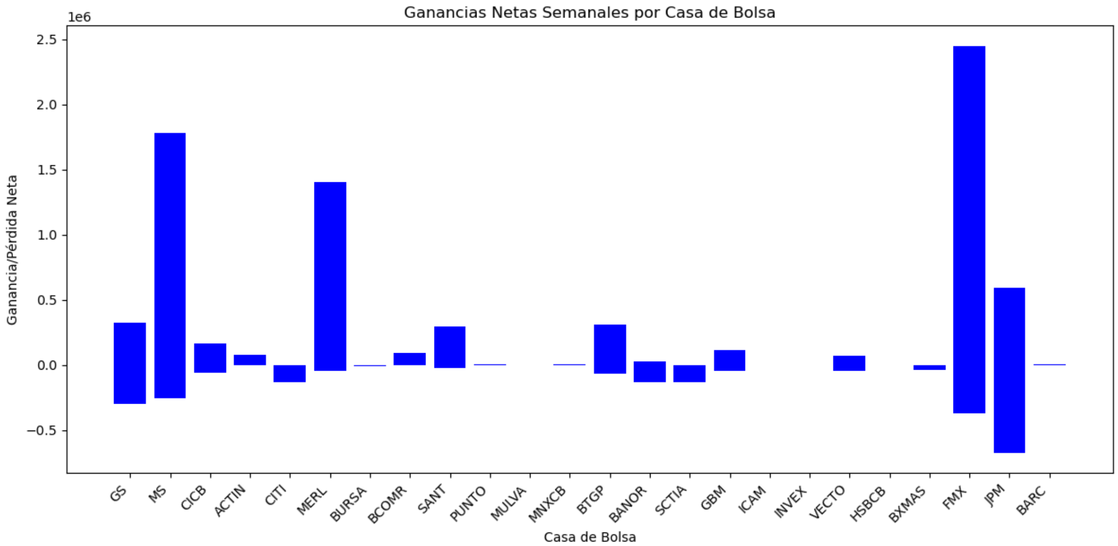
[53]:

```
import matplotlib.pyplot as plt
from pyspark.sql import functions as F

# Filtrar los valores nulos en 'buyer_name' y calcular las ganancias netas␣
↪diarias por casa de bolsa
daily_net_gain = df_a.filter(F.col('buyer_name').isNotNull()) \
    .groupBy('buyer_name', F.to_date('trade_time').alias('trade_date')) \
    .agg(F.sum('gain_loss').alias('net_gain_loss'))

# Convertir a Pandas para graficar
daily_net_gain_pd = daily_net_gain.toPandas()

# Verificar si hay datos para graficar
if not daily_net_gain_pd.empty:
    # Graficar las ganancias netas diarias por casa de bolsa
    plt.figure(figsize=(12, 6))
    plt.bar(daily_net_gain_pd['buyer_name'],␣
↪daily_net_gain_pd['net_gain_loss'], color='green')
    plt.title('Ganancias Netas Diarias por Casa de Bolsa')
    plt.xlabel('Casa de Bolsa')
    plt.ylabel('Ganancia/Pérdida Neta')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
else:
    print("No hay datos para graficar.")
```



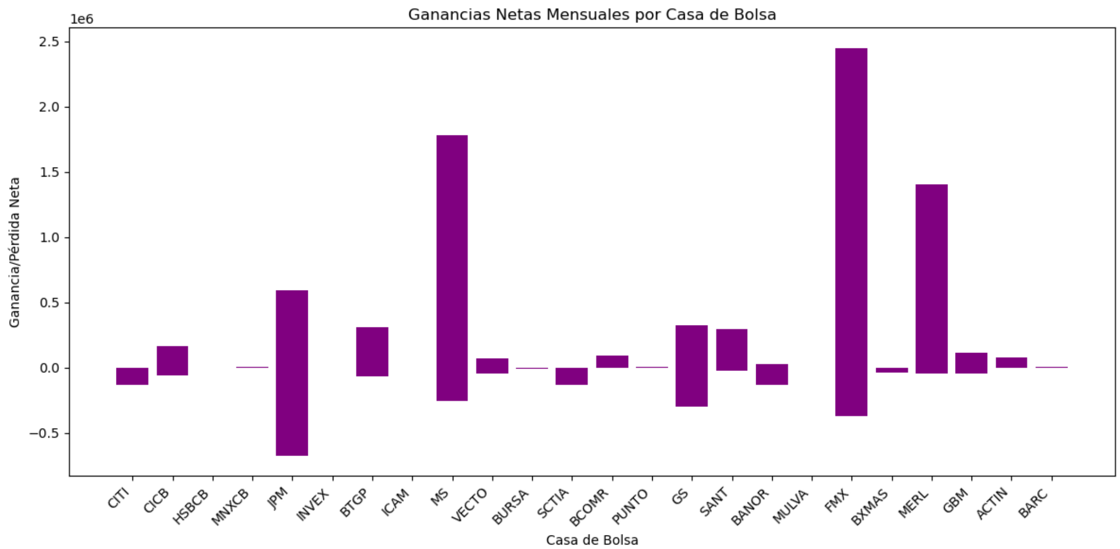
[54]:
```python
import matplotlib.pyplot as plt
from pyspark.sql import functions as F

# Agregar semana de transacción
df_a = df_a.withColumn('week_of_year', F.weekofyear('trade_time'))

# Filtrar valores nulos en 'buyer_name' y calcular las ganancias netas␣
↪semanales por casa de bolsa
weekly_net_gain = df_a.filter(F.col('buyer_name').isNotNull()) \
    .groupBy('buyer_name', 'week_of_year') \
    .agg(F.sum('gain_loss').alias('net_gain_loss'))

# Convertir a Pandas para graficar
weekly_net_gain_pd = weekly_net_gain.toPandas()

# Verificar si hay datos válidos para graficar
if not weekly_net_gain_pd.empty:
    # Graficar las ganancias netas semanales por casa de bolsa
    plt.figure(figsize=(12, 6))
    plt.bar(weekly_net_gain_pd['buyer_name'],␣
↪weekly_net_gain_pd['net_gain_loss'], color='blue')
    plt.title('Ganancias Netas Semanales por Casa de Bolsa')
    plt.xlabel('Casa de Bolsa')
    plt.ylabel('Ganancia/Pérdida Neta')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
else:
```

```
    print("No hay datos para graficar.")
```



[55]:
```
import matplotlib.pyplot as plt
from pyspark.sql import functions as F

# Agregar mes de transacción
df_a = df_a.withColumn('month_of_year', F.month('trade_time'))

# Filtrar valores nulos en 'buyer_name' y calcular las ganancias netas␣
↪mensuales por casa de bolsa
monthly_net_gain = df_a.filter(F.col('buyer_name').isNotNull()) \
    .groupBy('buyer_name', 'month_of_year') \
    .agg(F.sum('gain_loss').alias('net_gain_loss'))

# Convertir a Pandas para graficar
monthly_net_gain_pd = monthly_net_gain.toPandas()

# Verificar si hay datos válidos para graficar
if not monthly_net_gain_pd.empty:
    # Graficar las ganancias netas mensuales por casa de bolsa
    plt.figure(figsize=(12, 6))
    plt.bar(monthly_net_gain_pd['buyer_name'],␣
↪monthly_net_gain_pd['net_gain_loss'], color='purple')
    plt.title('Ganancias Netas Mensuales por Casa de Bolsa')
    plt.xlabel('Casa de Bolsa')
    plt.ylabel('Ganancia/Pérdida Neta')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
```

```
    plt.show()
else:
    print("No hay datos para graficar.")
```



[ ]:

```

```

[ ]:

```

```

[56]:

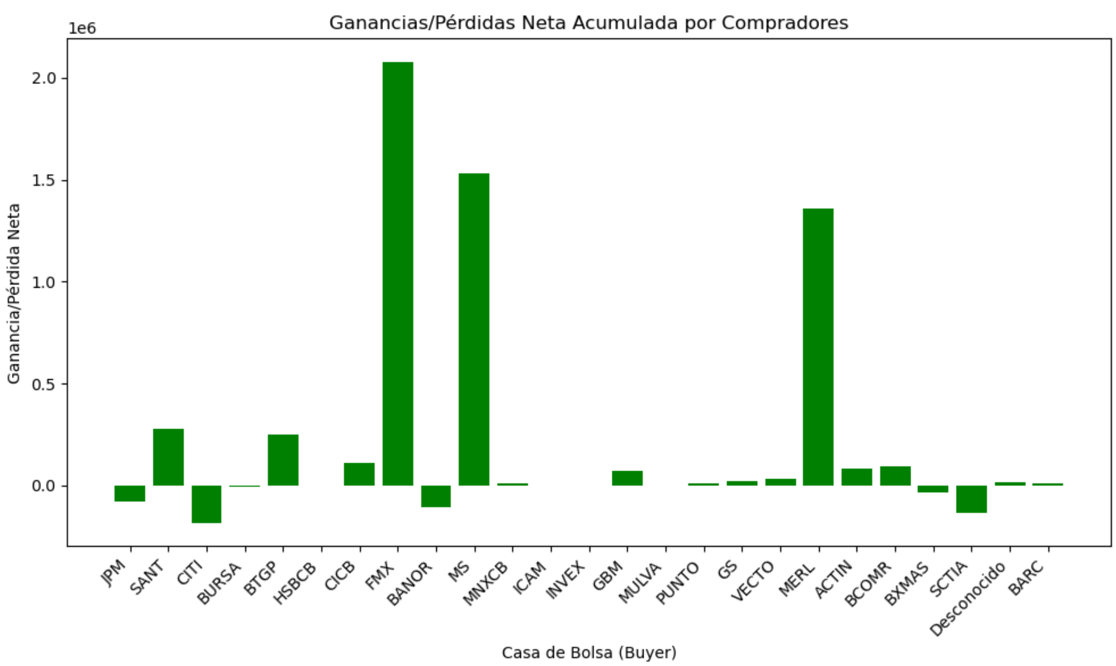
```
# 2. Medición de Rendimiento por Casas de Bolsa
#    a. Análisis de Performance
#    b. Volatilidad por Casa de Bolsa
#    c. Proporción de Ganancias
```

1. Análisis de Performance (Ganancia/Pérdida Neta Acumulada) Un gráfico de barras es útil para mostrar las ganancias/pérdidas netas acumuladas por cada casa de bolsa. Esto facilita la comparación entre compradores y vendedores.

[57]:

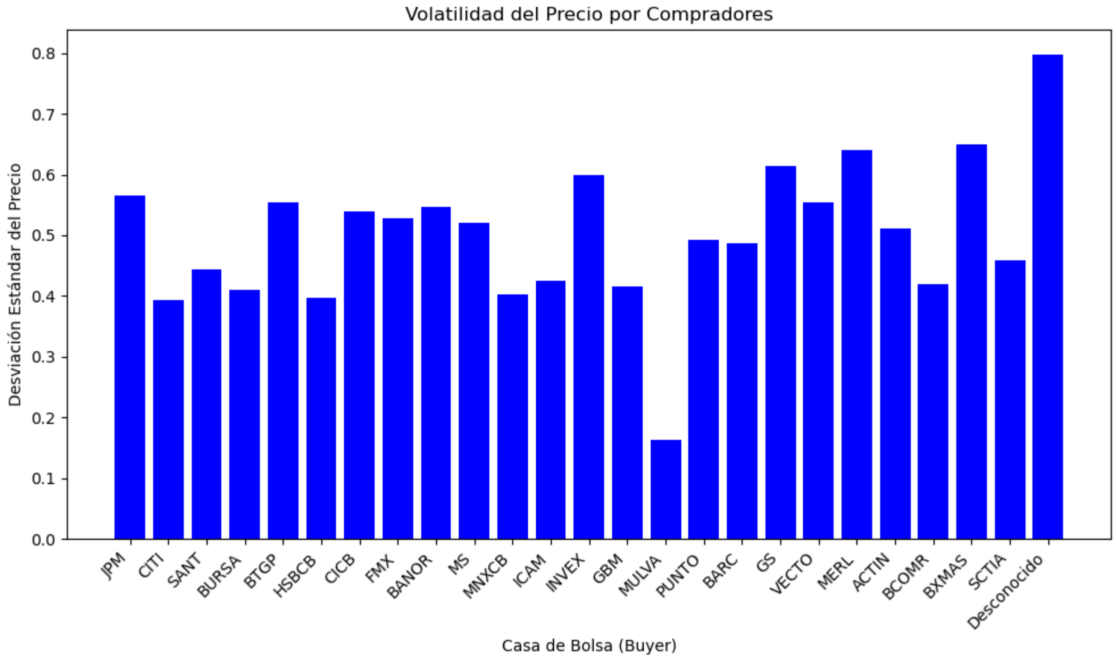
```
import matplotlib.pyplot as plt
from pyspark.sql import functions as F

# Agrupar por comprador y sumar las ganancias/pérdidas
performance_by_buyer = df_a.groupBy('buyer_name').agg(F.sum('gain_loss').
 ↪alias('total_gain_loss'))
performance_by_seller = df_a.groupBy('seller_name').agg(F.sum('gain_loss').
 ↪alias('total_gain_loss'))

# Convertir a Pandas para manipular y graficar
performance_by_buyer_pd = performance_by_buyer.toPandas()
performance_by_seller_pd = performance_by_seller.toPandas()
```

```
# Reemplazar valores nulos en buyer_name con 'Desconocido'
performance_by_buyer_pd['buyer_name'] = performance_by_buyer_pd['buyer_name'].
 ↪fillna('Desconocido')

# Filtrar filas donde 'buyer_name' o 'total_gain_loss' no sean nulos
performance_by_buyer_pd = performance_by_buyer_pd.dropna(subset=['buyer_name',␣
 ↪'total_gain_loss'])

# Graficar Ganancias/Pérdidas Acumuladas por Compradores
plt.figure(figsize=(10, 6))
plt.bar(performance_by_buyer_pd['buyer_name'],␣
 ↪performance_by_buyer_pd['total_gain_loss'], color='green')
plt.title('Ganancias/Pérdidas Neta Acumulada por Compradores')
plt.xlabel('Casa de Bolsa (Buyer)')
plt.ylabel('Ganancia/Pérdida Neta')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



2. Volatilidad por Casa de Bolsa Un gráfico de barras también es útil para visualizar la volatilidad de los precios de compra y venta por casa de bolsa. En este caso, la desviación estándar de los precios refleja la volatilidad.

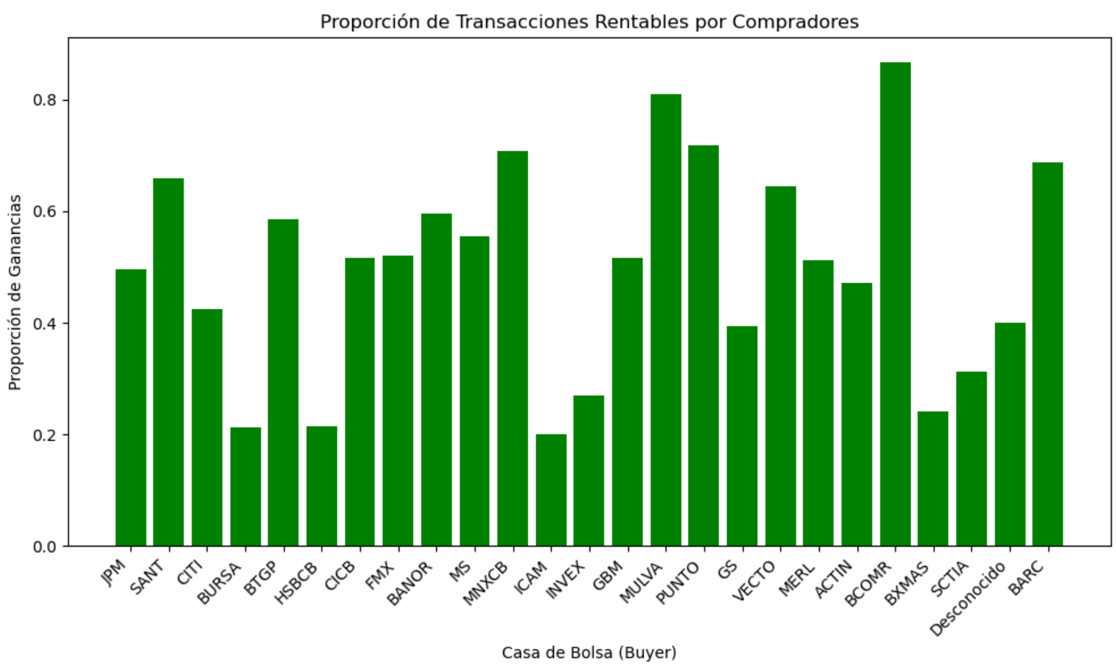
[58]:
```python
volatility_by_buyer = df_a.groupBy('buyer_name').agg(F.stddev('price').
    ↪alias('price_volatility'))
volatility_by_seller = df_a.groupBy('seller_name').agg(F.stddev('price').
    ↪alias('price_volatility'))



# Convertir los resultados de Spark a Pandas
volatility_by_buyer_pd = volatility_by_buyer.toPandas()
volatility_by_seller_pd = volatility_by_seller.toPandas()

import matplotlib.pyplot as plt

# Reemplazar valores nulos en 'buyer_name' con 'Desconocido'
volatility_by_buyer_pd['buyer_name'] = volatility_by_buyer_pd['buyer_name'].
    ↪fillna('Desconocido')

# Filtrar filas donde 'buyer_name' o 'price_volatility' no sean nulos
volatility_by_buyer_pd = volatility_by_buyer_pd.dropna(subset=['buyer_name',␣
    ↪'price_volatility'])

# Graficar Volatilidad por Compradores
plt.figure(figsize=(10, 6))
plt.bar(volatility_by_buyer_pd['buyer_name'],␣
    ↪volatility_by_buyer_pd['price_volatility'], color='blue')
plt.title('Volatilidad del Precio por Compradores')
plt.xlabel('Casa de Bolsa (Buyer)')
plt.ylabel('Desviación Estándar del Precio')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



3. Proporción de Transacciones Rentables Este gráfico puede ser representado como un gráfico de barras apiladas, donde se visualiza la proporción de transacciones rentables en comparación con el total de transacciones.

[59]:
```
profit_ratio_by_buyer = df_a.groupBy('buyer_name').agg(
    (F.sum(F.when(F.col('gain_loss') > 0, 1).otherwise(0)) / F.count('*')).
 ↪alias('profit_ratio')
)

profit_ratio_by_seller = df_a.groupBy('seller_name').agg(
    (F.sum(F.when(F.col('gain_loss') > 0, 1).otherwise(0)) / F.count('*')).
 ↪alias('profit_ratio')
)

# Convertir los resultados de Spark a Pandas
profit_ratio_by_buyer_pd = profit_ratio_by_buyer.toPandas()
profit_ratio_by_seller_pd = profit_ratio_by_seller.toPandas()

volume_by_buyer = df_a.groupBy('buyer_name').agg(F.sum('volume').
 ↪alias('total_volume'))
volume_by_seller = df_a.groupBy('seller_name').agg(F.sum('volume').
 ↪alias('total_volume'))

import matplotlib.pyplot as plt

# Reemplazar valores nulos en 'buyer_name' con 'Desconocido'
```

```
profit_ratio_by_buyer_pd['buyer_name'] = profit_ratio_by_buyer_pd['buyer_name'].
↪fillna('Desconocido')

# Filtrar filas donde 'buyer_name' o 'profit_ratio' no sean nulos
profit_ratio_by_buyer_pd = profit_ratio_by_buyer_pd.
↪dropna(subset=['buyer_name', 'profit_ratio'])

# Graficar Proporción de Transacciones Rentables por Compradores
plt.figure(figsize=(10, 6))
plt.bar(profit_ratio_by_buyer_pd['buyer_name'],␣
↪profit_ratio_by_buyer_pd['profit_ratio'], color='green')
plt.title('Proporción de Transacciones Rentables por Compradores')
plt.xlabel('Casa de Bolsa (Buyer)')
plt.ylabel('Proporción de Ganancias')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

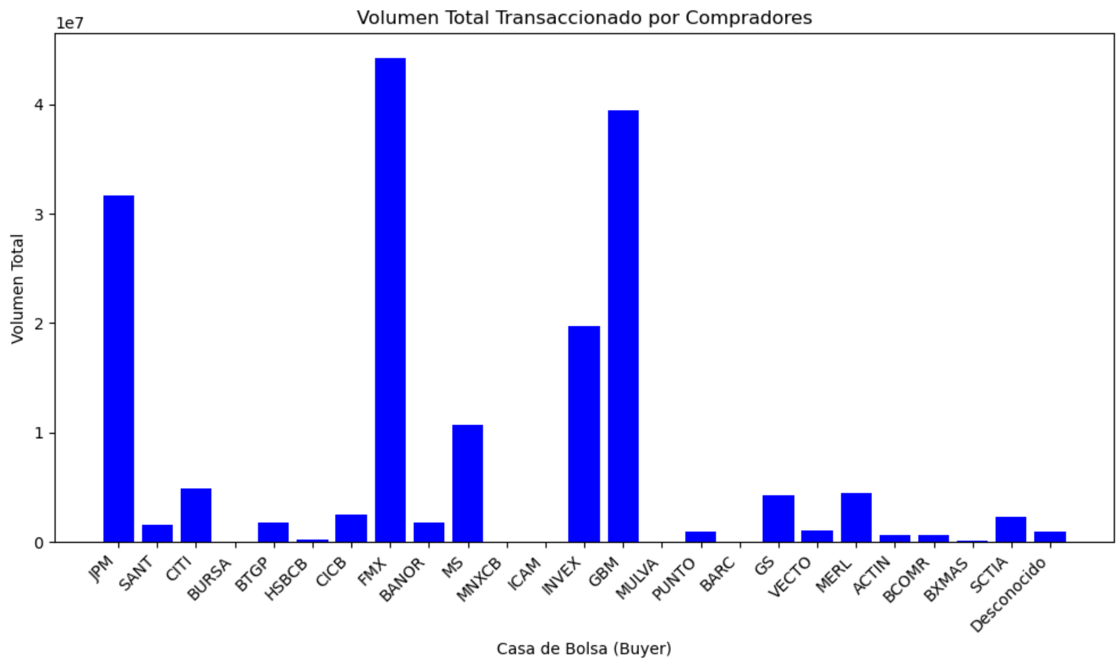


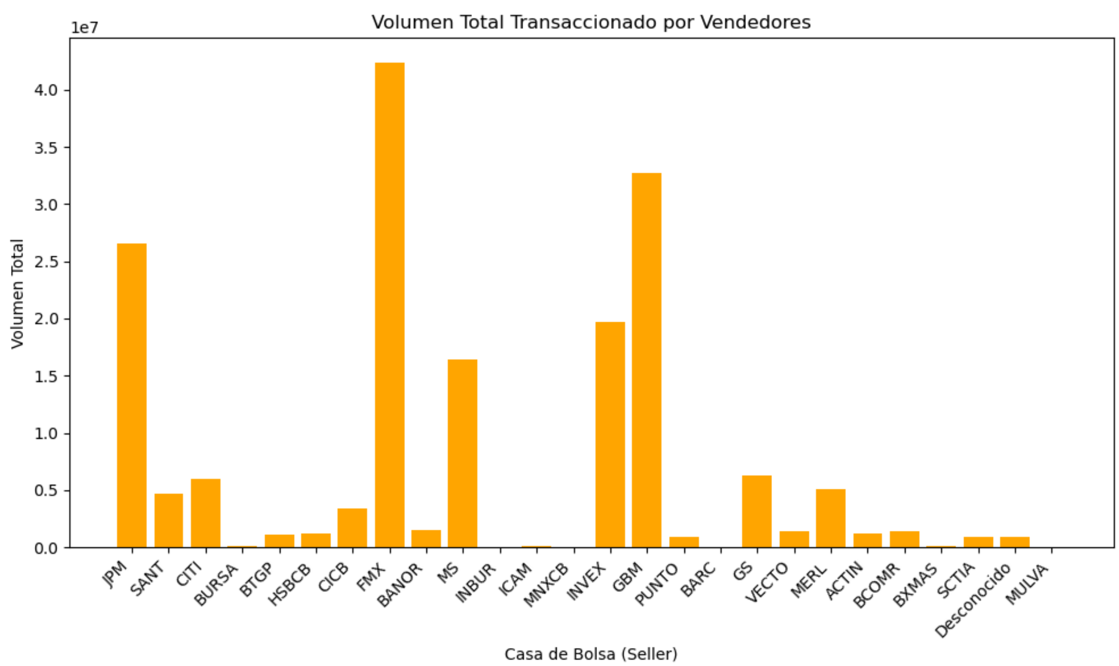
4. Volumen Total Transaccionado por Casa de Bolsa Este gráfico muestra el volumen total transaccionado por cada casa de bolsa. Es útil para identificar qué casas de bolsa dominan en términos de volumen.

[60]:
```
transactions_by_time_buyer = df_a.groupBy('buyer_name', 'hour').count()
transactions_by_time_seller = df_a.groupBy('seller_name', 'hour').count()
```

```
transactions_by_day_buyer = df_a.groupBy('buyer_name', 'day_of_week_num').
 ↪count()
transactions_by_day_seller = df_a.groupBy('seller_name', 'day_of_week_num').
 ↪count()

# Convertir los resultados de Spark a Pandas
volume_by_buyer_pd = volume_by_buyer.toPandas()
volume_by_seller_pd = volume_by_seller.toPandas()

import matplotlib.pyplot as plt
import pandas as pd
from pyspark.sql import functions as F

# Manejar valores nulos en 'buyer_name' y 'seller_name'
df_a = df_a.fillna({'buyer_name': 'Desconocido', 'seller_name': 'Desconocido'})

# Agrupar por comprador (buyer_name) y sumar los volúmenes
volume_by_buyer = df_a.groupBy('buyer_name').agg(F.sum('volume').
 ↪alias('total_volume'))

# Agrupar por vendedor (seller_name) y sumar los volúmenes
volume_by_seller = df_a.groupBy('seller_name').agg(F.sum('volume').
 ↪alias('total_volume'))

# Convertir a Pandas para graficar
volume_by_buyer_pd = volume_by_buyer.toPandas()
volume_by_seller_pd = volume_by_seller.toPandas()

# Graficar Volumen Total por Compradores
plt.figure(figsize=(10, 6))
plt.bar(volume_by_buyer_pd['buyer_name'], volume_by_buyer_pd['total_volume'],␣
 ↪color='blue')
plt.title('Volumen Total Transaccionado por Compradores')
plt.xlabel('Casa de Bolsa (Buyer)')
plt.ylabel('Volumen Total')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

# Graficar Volumen Total por Vendedores
plt.figure(figsize=(10, 6))
plt.bar(volume_by_seller_pd['seller_name'],␣
 ↪volume_by_seller_pd['total_volume'], color='orange')
plt.title('Volumen Total Transaccionado por Vendedores')
plt.xlabel('Casa de Bolsa (Seller)')
plt.ylabel('Volumen Total')
```

```python
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



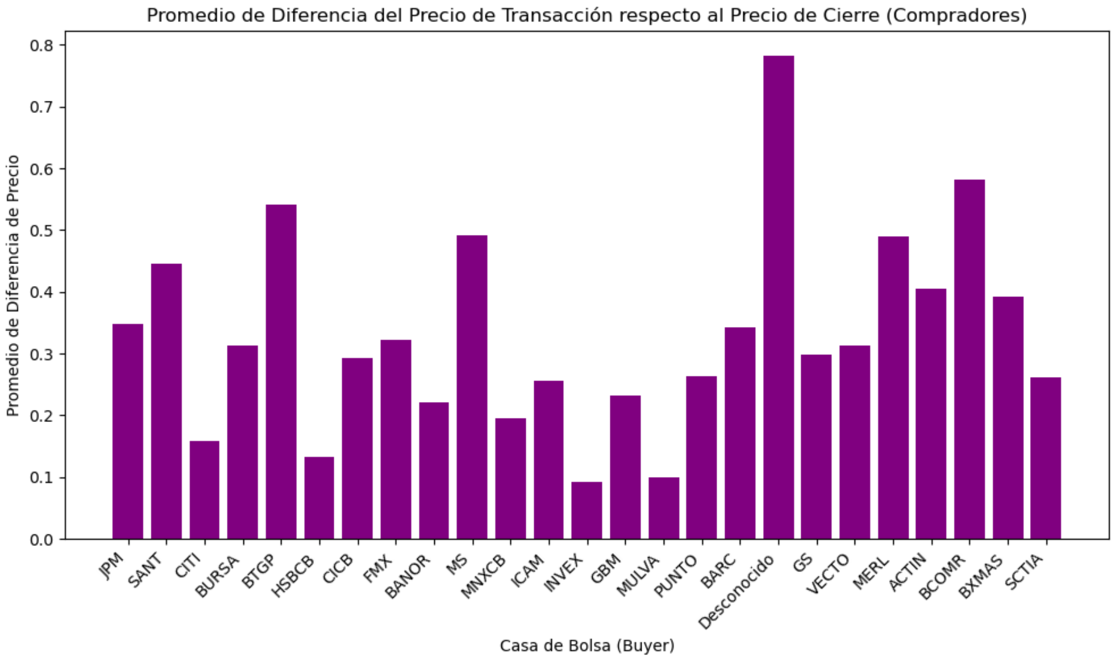


5. Diferencia Respecto al Precio de Cierre Este gráfico muestra la diferencia entre los precios de transacción y el precio de cierre para cada casa de bolsa, lo que permite visualizar qué tan cerca estuvieron las transacciones del precio final del día.

[61]:
```python
print(df_pandas.columns)
```

```
Index(['price', 'volume', 'amount', 'trade_date', 'day_of_week', 'hour',
       'minute'],
      dtype='object')
```

[62]:
```python
import matplotlib.pyplot as plt
import pandas as pd
from pyspark.sql import functions as F

# Calcular la diferencia de precio con el precio de cierre
df_a = df_a.withColumn('price_diff_from_closing', F.abs(F.col('price') - F.
 ↪col('closing_price')))

# Agrupar por comprador y calcular la diferencia promedio respecto al precio de␣
 ↪cierre
avg_price_diff_by_buyer = df_a.groupBy('buyer_name').agg(F.
 ↪mean('price_diff_from_closing').alias('avg_price_diff_from_closing'))

# Convertir a Pandas para graficar
avg_price_diff_by_buyer_pd = avg_price_diff_by_buyer.toPandas()

# Reemplazar valores nulos en buyer_name con 'Desconocido'
avg_price_diff_by_buyer_pd['buyer_name'] =␣
 ↪avg_price_diff_by_buyer_pd['buyer_name'].fillna('Desconocido')

# Graficar la diferencia promedio respecto al precio de cierre
plt.figure(figsize=(10, 6))
plt.bar(avg_price_diff_by_buyer_pd['buyer_name'],␣
 ↪avg_price_diff_by_buyer_pd['avg_price_diff_from_closing'], color='purple')
plt.title('Promedio de Diferencia del Precio de Transacción respecto al Precio␣
 ↪de Cierre (Compradores)')
plt.xlabel('Casa de Bolsa (Buyer)')
plt.ylabel('Promedio de Diferencia de Precio')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



[63]:
```
df_a.printSchema()
```

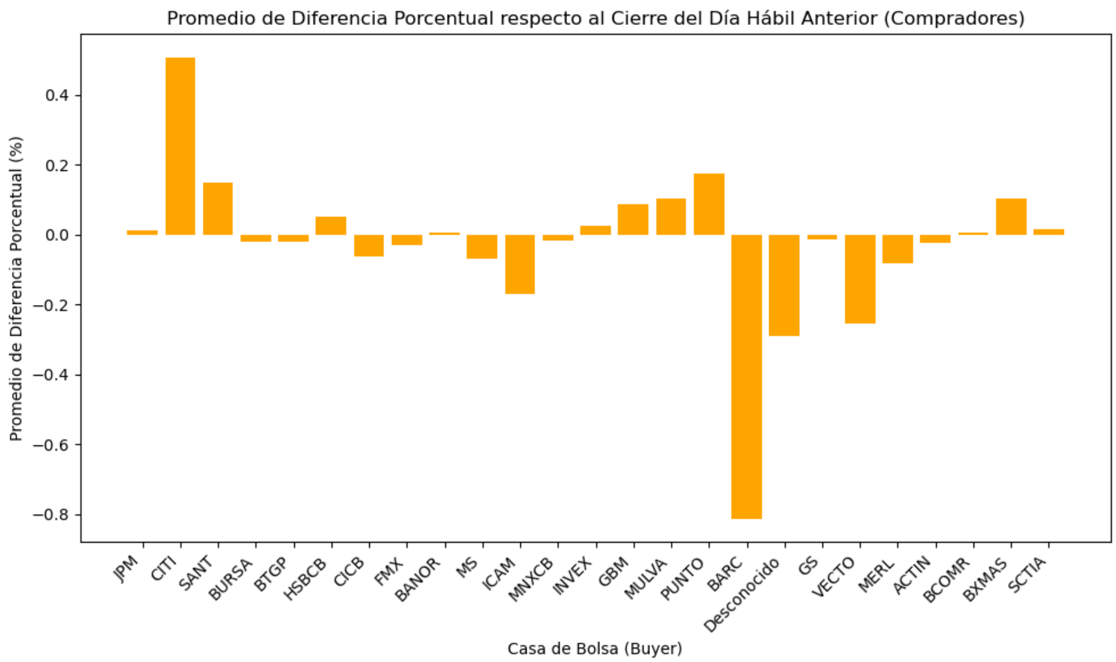
```
root
 |-- trade_time: timestamp (nullable = true)
 |-- match_number: integer (nullable = true)
 |-- instrument_id: integer (nullable = true)
 |-- timestamp: timestamp (nullable = true)
 |-- volume: integer (nullable = true)
 |-- price: double (nullable = true)
 |-- amount: double (nullable = true)
 |-- buyer_id: integer (nullable = true)
 |-- buyer_name: string (nullable = false)
 |-- seller_id: integer (nullable = true)
 |-- seller_name: string (nullable = false)
 |-- operation_type: string (nullable = true)
 |-- concertation_type: string (nullable = true)
 |-- price_setter: integer (nullable = true)
 |-- lot: integer (nullable = true)
 |-- symbol: string (nullable = true)
 |-- last_day_close: double (nullable = true)
 |-- last_day_close_date: timestamp (nullable = true)
 |-- unitary_daily_variation: double (nullable = true)
 |-- percentage_daily_variation: double (nullable = true)
 |-- last_week_close: double (nullable = true)
 |-- last_week_close_date: timestamp (nullable = true)
 |-- unitary_weekly_variation: double (nullable = true)
```

```
|-- percentage_weekly_variation: double (nullable = true)
|-- last_month_close: double (nullable = true)
|-- last_month_close_date: timestamp (nullable = true)
|-- unitary_monthly_variation: double (nullable = true)
|-- percentage_monthly_variation: double (nullable = true)
|-- date: date (nullable = true)
|-- hour: integer (nullable = true)
|-- minute: integer (nullable = true)
|-- minute_segment: long (nullable = true)
|-- closing_price: double (nullable = true)
|-- gain_loss: double (nullable = true)
|-- week_of_year: integer (nullable = true)
|-- month_of_year: integer (nullable = true)
|-- trade_date: date (nullable = true)
|-- day_of_week_num: integer (nullable = true)
|-- day_of_week: string (nullable = true)
|-- price_diff_from_closing: double (nullable = true)
```

[64]:

```python
import matplotlib.pyplot as plt
import pandas as pd
from pyspark.sql import functions as F

# Calcular la diferencia porcentual respecto al cierre del día hábil anterior
df_a = df_a.withColumn(
    'percentage_diff_from_prev_close',
    (F.col('price') - F.col('last_day_close')) / F.col('last_day_close') * 100
)

# Agrupar por comprador (buyer_name) y calcular la diferencia porcentual␣
↪promedio
avg_percentage_diff_by_buyer = df_a.groupBy('buyer_name').agg(F.
↪mean('percentage_diff_from_prev_close').
↪alias('avg_percentage_diff_from_prev_close'))

# Convertir a Pandas para graficar
avg_percentage_diff_by_buyer_pd = avg_percentage_diff_by_buyer.toPandas()

# Reemplazar valores nulos en buyer_name con 'Desconocido'
avg_percentage_diff_by_buyer_pd['buyer_name'] =␣
↪avg_percentage_diff_by_buyer_pd['buyer_name'].fillna('Desconocido')

# Graficar la diferencia porcentual promedio respecto al cierre del día hábil␣
↪anterior
plt.figure(figsize=(10, 6))
```

```python
plt.bar(avg_percentage_diff_by_buyer_pd['buyer_name'],
 ↪avg_percentage_diff_by_buyer_pd['avg_percentage_diff_from_prev_close'],
 ↪color='orange')
plt.title('Promedio de Diferencia Porcentual respecto al Cierre del Día Hábil
 ↪Anterior (Compradores)')
plt.xlabel('Casa de Bolsa (Buyer)')
plt.ylabel('Promedio de Diferencia Porcentual (%)')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



[65]:
```python
from pyspark.sql import Window
from pyspark.sql import functions as F

# Definir una ventana de partición por día, ordenada por el tiempo de
 ↪transacción
day_window = Window.partitionBy('date').orderBy(F.desc('trade_time'))

# Obtener el precio de cierre del día actual (último precio del día)
df_a = df_a.withColumn('current_closing_price', F.first('price').
 ↪over(day_window))

# Definir una ventana de partición para calcular el cierre del día anterior
 ↪(lag de un día)
previous_day_window = Window.partitionBy('symbol').orderBy('date')
```

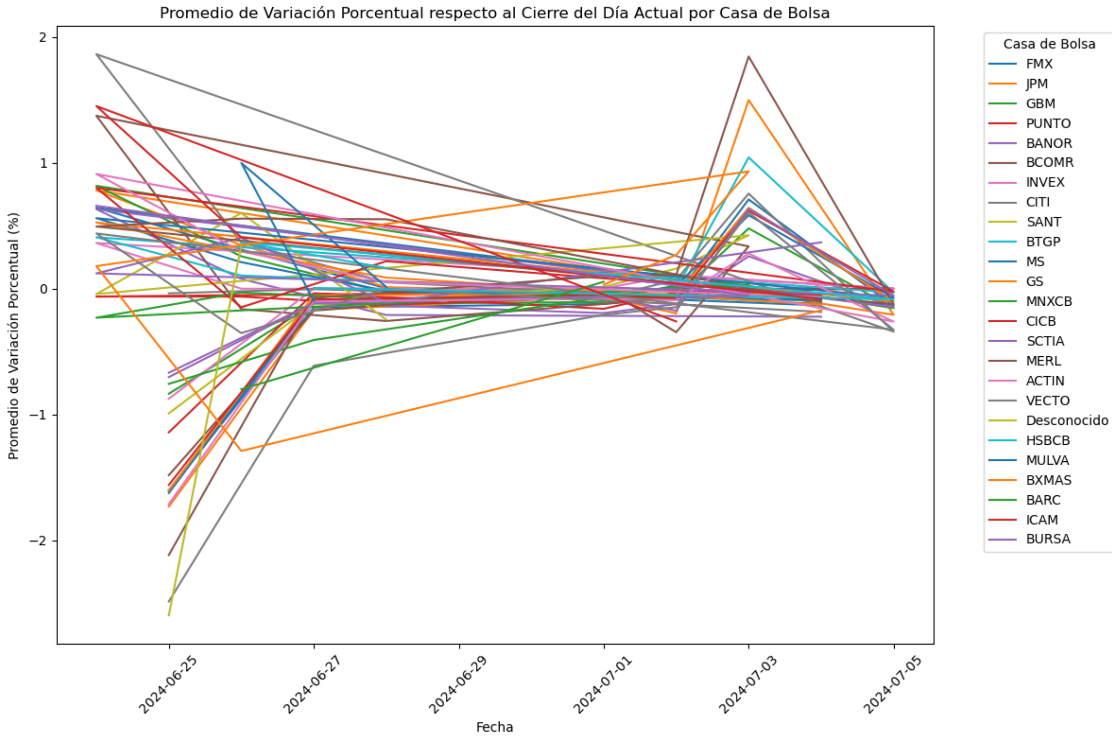
```
# Obtener el cierre del día anterior (cierre del día anterior)
df_a = df_a.withColumn('prev_closing_price', F.lag('current_closing_price', 1).
↪over(previous_day_window))

# Calcular la variación porcentual respecto al cierre del día anterior
df_a = df_a.withColumn(
    'percentage_variation_prev_close',
    F.when(F.col('prev_closing_price').isNotNull(),
           (F.col('current_closing_price') - F.col('prev_closing_price')) / F.
↪col('prev_closing_price') * 100
          ).otherwise(None)
)

# Calcular la variación porcentual respecto al precio de cierre del día en curso
df_a = df_a.withColumn(
    'percentage_variation_current_close',
    F.when(F.col('current_closing_price').isNotNull(),
           (F.col('price') - F.col('current_closing_price')) / F.
↪col('current_closing_price') * 100
          ).otherwise(None)
)

# Mostrar algunas columnas relevantes
df_a.select('date', 'trade_time', 'price', 'prev_closing_price',␣
↪'current_closing_price', 'percentage_variation_prev_close',␣
↪'percentage_variation_current_close').show(2)
```

```
+----------+-------------------+-----+------------------+---------------------+-
-----------------------------+----------------------------------+
|      date|         trade_time|price|prev_closing_price|current_closing_price|p
ercentage_variation_prev_close|percentage_variation_current_close|
+----------+-------------------+-----+------------------+---------------------+-
-----------------------------+----------------------------------+
|2024-06-24|2024-06-24 14:00:05|60.64|              NULL|                60.64|
NULL|                               0.0|
|2024-06-24|2024-06-24 14:00:05|60.64|             60.64|                60.64|
0.0|                               0.0|
+----------+-------------------+-----+------------------+---------------------+-
-----------------------------+----------------------------------+
only showing top 2 rows
```

[66]:
```
import matplotlib.pyplot as plt
import pandas as pd
from pyspark.sql import functions as F
from pyspark.sql import Window
```

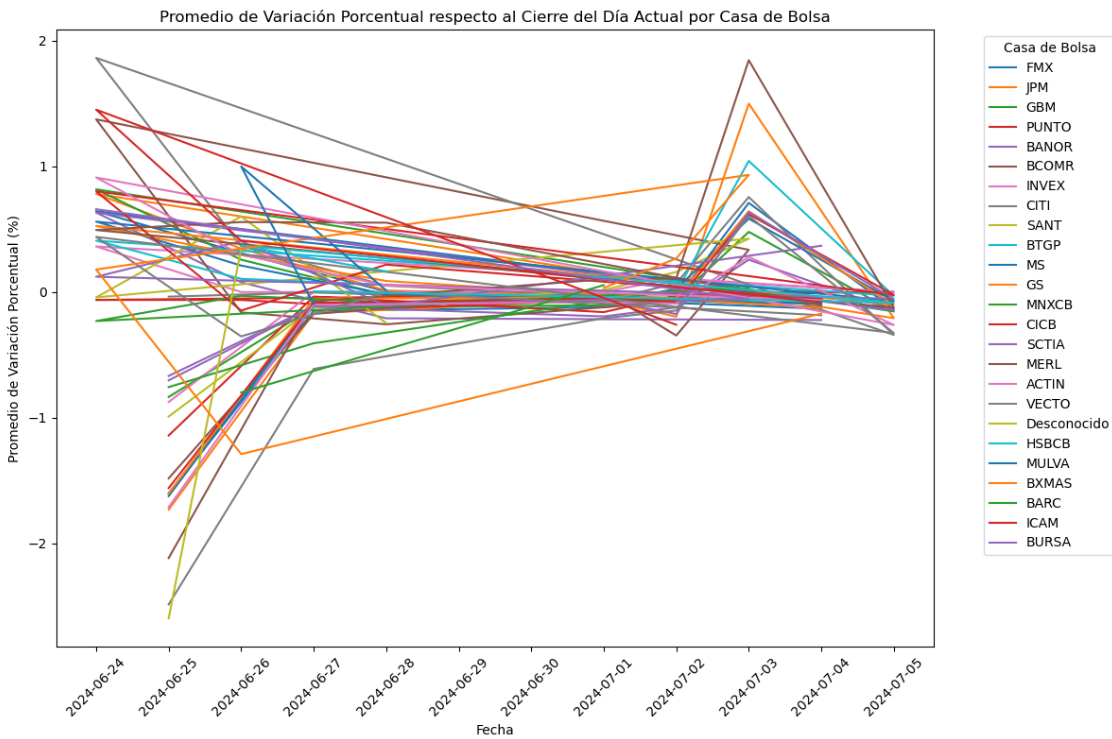
```python
# Definir una ventana de partición por día, ordenada por el tiempo de␣
↪transacción
day_window = Window.partitionBy('date').orderBy(F.desc('trade_time'))

# Obtener el precio de cierre del día actual (último precio del día)
df_a = df_a.withColumn('current_closing_price', F.first('price').
↪over(day_window))

# Calcular la variación porcentual respecto al precio de cierre del día en curso
df_a = df_a.withColumn(
    'percentage_variation_current_close',
    F.when(F.col('current_closing_price').isNotNull(),
           (F.col('price') - F.col('current_closing_price')) / F.
↪col('current_closing_price') * 100
          ).otherwise(None)
)

# Agrupar por casa de bolsa (buyer_name) y fecha, calcular el promedio de la␣
↪diferencia porcentual
avg_percentage_diff_by_buyer = df_a.groupBy('buyer_name', 'date').agg(
    F.mean('percentage_variation_current_close').alias('avg_percentage_diff')
)

# Convertir a Pandas para graficar
avg_percentage_diff_by_buyer_pd = avg_percentage_diff_by_buyer.toPandas()

# Reemplazar valores nulos en buyer_name con 'Desconocido'
avg_percentage_diff_by_buyer_pd['buyer_name'] =␣
↪avg_percentage_diff_by_buyer_pd['buyer_name'].fillna('Desconocido')

# Graficar la variación porcentual promedio para cada casa de bolsa en todas␣
↪las fechas
plt.figure(figsize=(12, 8))

# Crear un gráfico de líneas, uno por cada comprador
for buyer in avg_percentage_diff_by_buyer_pd['buyer_name'].unique():
    buyer_data =␣
↪avg_percentage_diff_by_buyer_pd[avg_percentage_diff_by_buyer_pd['buyer_name']␣
↪== buyer]
    plt.plot(buyer_data['date'], buyer_data['avg_percentage_diff'], label=buyer)

plt.title('Promedio de Variación Porcentual respecto al Cierre del Día Actual␣
↪por Casa de Bolsa')
plt.xlabel('Fecha')
plt.ylabel('Promedio de Variación Porcentual (%)')
```

```
plt.xticks(rotation=45)
plt.legend(title='Casa de Bolsa', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```



[67]:
```
import matplotlib.pyplot as plt
import pandas as pd
import matplotlib.dates as mdates
from pyspark.sql import functions as F
from pyspark.sql import Window

# Crear ventana de partición para obtener el precio de cierre diario
day_window = Window.partitionBy('date').orderBy(F.desc('trade_time'))
df_a = df_a.withColumn('current_closing_price', F.first('price').
 ↪over(day_window))

# Calcular la variación porcentual respecto al precio de cierre del día
df_a = df_a.withColumn(
    'percentage_variation_current_close',
    (F.col('price') - F.col('current_closing_price')) / F.
 ↪col('current_closing_price') * 100
)
```

```python
# Agrupar por comprador y fecha
avg_percentage_diff_by_buyer = df_a.groupBy('buyer_name', 'date').agg(
    F.mean('percentage_variation_current_close').alias('avg_percentage_diff')
)

# Convertir a Pandas y ajustar datos
avg_percentage_diff_by_buyer_pd = avg_percentage_diff_by_buyer.toPandas()
avg_percentage_diff_by_buyer_pd['buyer_name'] = avg_percentage_diff_by_buyer_pd['buyer_name'].fillna('Desconocido')

# Graficar
plt.figure(figsize=(12, 8))

# Gráfico de líneas para cada comprador
for buyer in avg_percentage_diff_by_buyer_pd['buyer_name'].unique():
    buyer_data = avg_percentage_diff_by_buyer_pd[avg_percentage_diff_by_buyer_pd['buyer_name'] == buyer]
    plt.plot(buyer_data['date'], buyer_data['avg_percentage_diff'], label=buyer)

plt.title('Promedio de Variación Porcentual respecto al Cierre del Día Actual por Casa de Bolsa')
plt.xlabel('Fecha')
plt.ylabel('Promedio de Variación Porcentual (%)')
plt.xticks(rotation=45)

# Formato de fecha en el eje X
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.gca().xaxis.set_major_locator(mdates.DayLocator(interval=1))  # Mostrar solo fechas con datos
plt.legend(title='Casa de Bolsa', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```



[ ]:

```
```

[ ]:

```
```

[68]:

```
df_a.show(2)
```

```
+-------------------+------------+-------------+--------------------+------+----
-+----------+--------+----------+---------+-----------+--------------+----------
-------+------------+---+-------+--------------+-------------------+------------
-----------+--------------------------+---------------+--------------------+----
--------------------+---------------------------+----------------+--------------
-------+-------------------------+----------------------------+----------+----+-
-----+--------------+-------------+---------+------------+-------------+--------
--+---------------+-----------+-----------------------+-------------------------
------+---------------------+------------------+-------------------------------
+----------------------------------+
|          trade_time|match_number|instrument_id|
timestamp|volume|price|     amount|buyer_id|buyer_name|seller_id|seller_name|oper
ation_type|concertation_type|price_setter|lot| symbol|last_day_close|last_day_cl
ose_date|unitary_daily_variation|percentage_daily_variation|last_week_close|last
_week_close_date|unitary_weekly_variation|percentage_weekly_variation|last_month
_close|last_month_close_date|unitary_monthly_variation|percentage_monthly_variat
ion|          date|hour|minute|minute_segment|closing_price|gain_loss|week_of_year|m
onth_of_year|trade_date|day_of_week_num|day_of_week|price_diff_from_closing|perc
```

```
entage_diff_from_prev_close|current_closing_price|prev_closing_price|percentage_
variation_prev_close|percentage_variation_current_close|
+-------------------+------------+-------------+--------------------+------+----
-+----------+--------+----------+---------+-----------+-------------+----------
-------+------------+---+-------+-------------+-------------------+------------
-----------+-------------------------+--------------+--------------------+----
-------------------+--------------------------+---------------+--------------
-------+------------------------+---------------------------+----------+----+-
-----+--------------+-------------+---------+-----------+------------+--------
--+---------------+-----------+----------------------+------------------------
------+--------------------+-----------------+-----------------------------
+---------------------------------+
|2024-06-27 14:00:05|       18033|            5|2024-06-27 14:00:…|
85446|62.09|5305342.14|     112|     SCTIA|        0|        FMX|             H|
H|           0|  0|WALMEX*|         62.12|2024-06-27 13:14:24|
-0.02999999999999403|      -0.04829362524145852|          62.12| 2024-06-27
13:14:24|    -0.02999999999999403|       -0.04829362524145852|           62.12|
2024-06-27 13:14:24|     -0.02999999999999403|
-0.04829362524145852|2024-06-27|  14|     0|             0|        62.09|
0.0|          26|            6|2024-06-27|              4|        Thu|
0.0|          -0.04829362524145852|               62.09|             62.04|
0.08059316569955555|                             0.0|
|2024-06-27 14:00:05|       18060|            5|2024-06-27 14:00:…|
22482|62.09|1395907.38|     118|     HSBCB|      118|      HSBCB|             H|
H|           0|  0|WALMEX*|         61.78|2024-06-27 08:50:50|
0.3100000000000023|         0.501780511492396|          61.78| 2024-06-27
08:50:50|       0.3100000000000023|          0.501780511492396|           61.78|
2024-06-27 08:50:50|       0.3100000000000023|
0.501780511492396|2024-06-27|  14|     0|             0|        62.09|      0.0|
26|            6|2024-06-27|              4|        Thu|                    0.0|
0.501780511492396|                62.09|             62.09|
0.0|                                0.0|
+-------------------+------------+-------------+--------------------+------+----
-+----------+--------+----------+---------+-----------+-------------+----------
-------+------------+---+-------+-------------+-------------------+------------
-----------+-------------------------+--------------+--------------------+----
-------------------+--------------------------+---------------+--------------
-------+------------------------+---------------------------+----------+----+-
-----+--------------+-------------+---------+-----------+------------+--------
--+---------------+-----------+----------------------+------------------------
------+--------------------+-----------------+-----------------------------
+---------------------------------+
only showing top 2 rows
```

[69]:
```python
combined_df.show(2)
```

```
+-------------------+------------+-------------+--------------------+------+----
```

```
--+--------+--------+----------+---------+-----------+-------------+-----------
------+-----------+---+--------+
|         trade_time|match_number|instrument_id|           timestamp|volume|
price|  amount|buyer_id|buyer_name|seller_id|seller_name|operation_type|concerta
tion_type|price_setter|lot|  symbol|
+-------------------+------------+-------------+--------------------+------+----
--+--------+--------+----------+---------+-----------+-------------+-----------
------+-----------+---+--------+
|2024-06-24 13:01:01|       11488|            5|2024-06-24 13:01:…|   426|
60.73|25870.98|      54|      MERL|        0|          GS|              C|
0|           1|  1| WALMEX*|
|2024-06-24 13:08:20|        7702|            3|2024-06-24 13:08:…|
7|193.83| 1356.81|     141|      BTGP|        0|         FMX|              C|
0|           0|  0|FEMSAUBD|
+-------------------+------------+-------------+--------------------+------+----
--+--------+--------+----------+---------+-----------+-------------+-----------
------+-----------+---+--------+
only showing top 2 rows
```

[ ]:

```

```

7 Tarea 6

7.1 Modelos de Machine Learning en PySpark usando el conjunto de datos de precios intradía de una emisora de BMV y otras emisoras

8 Cambio de granularidad a un minuto, para efectos prácticos y/o factibles.

1. Preparación de Datos Convertimos variables relevantes y llenamos valores faltantes.

[70]:

```python
from pyspark.sql import functions as F

# Obtener una lista de símbolos únicos en el DataFrame
unique_symbols = [row["symbol"] for row in combined_df.select("symbol").
↪distinct().collect()]

# Crear un diccionario para almacenar DataFrames por símbolo
df_symbols = {}
# df_symbols_excluded = {'BIMBOA'}
df_symbols_excluded = {''}

# Iterar sobre cada símbolo y crear un DataFrame filtrado
for symbol in unique_symbols:
    if symbol not in df_symbols_excluded:
        df_symbols[symbol] = combined_df.filter(F.col("symbol") == symbol)
```

```python
# Opcional: Mostrar los primeros registros de cada DataFrame para verificar
for symbol, df in df_symbols.items():
    print(f"Primeros registros para el símbolo {symbol}:")
    # Ordenar cronológicamente
    df.orderBy("trade_time").show(2)
```

```
Primeros registros para el símbolo SORIANAB:
+-------------------+------------+-------------+--------------------+------+----
-+------+--------+----------+---------+-----------+--------------+--------------
---+------------+---+--------+
|         trade_time|match_number|instrument_id|           timestamp|volume|pric
e|amount|buyer_id|buyer_name|seller_id|seller_name|operation_type|concertation_t
ype|price_setter|lot|  symbol|
+-------------------+------------+-------------+--------------------+------+----
-+------+--------+----------+---------+-----------+--------------+--------------
---+------------+---+--------+
|2024-06-24 07:30:00|          11|           80|2024-06-24 07:30:…|
1|29.34| 29.34|     119|     ACTIN|       14|        GBM|             C|
O|           0|  0|SORIANAB|
|2024-06-24 07:30:00|           1|           80|2024-06-24 07:30:…|
1|29.34| 29.34|      14|       GBM|       14|        GBM|             C|
C|           0|  0|SORIANAB|
+-------------------+------------+-------------+--------------------+------+----
-+------+--------+----------+---------+-----------+--------------+--------------
---+------------+---+--------+
only showing top 2 rows

Primeros registros para el símbolo BIMBOA:
+-------------------+------------+-------------+--------------------+------+----
-+------+--------+----------+---------+-----------+--------------+--------------
---+------------+---+------+
|         trade_time|match_number|instrument_id|           timestamp|volume|pric
e|amount|buyer_id|buyer_name|seller_id|seller_name|operation_type|concertation_t
ype|price_setter|lot|symbol|
+-------------------+------------+-------------+--------------------+------+----
-+------+--------+----------+---------+-----------+--------------+--------------
---+------------+---+------+
|2024-06-24 07:30:00|          14|          124|2024-06-24 07:30:…|
1|68.44| 68.44|     137|     BXMAS|       14|        GBM|             C|
O|           0|  0|BIMBOA|
|2024-06-24 07:30:00|          27|          124|2024-06-24 07:30:…|
1|68.44| 68.44|     137|     BXMAS|       14|        GBM|             C|
O|           0|  0|BIMBOA|
+-------------------+------------+-------------+--------------------+------+----
-+------+--------+----------+---------+-----------+--------------+--------------
---+------------+---+------+
```

```
only showing top 2 rows

Primeros registros para el símbolo ALSEA*:
+-------------------+------------+-------------+--------------------+------+----
-+------+--------+----------+---------+-----------+--------------+--------------
---+------------+---+------+
|         trade_time|match_number|instrument_id|           timestamp|volume|pric
e|amount|buyer_id|buyer_name|seller_id|seller_name|operation_type|concertation_t
ype|price_setter|lot|symbol|
+-------------------+------------+-------------+--------------------+------+----
-+------+--------+----------+---------+-----------+--------------+--------------
---+------------+---+------+
|2024-06-24 07:30:00|          32|         1729|2024-06-24 07:30:…|
3|62.41|187.23|      14|       GBM|       14|        GBM|             C|
C|           0|  0|ALSEA*|
|2024-06-24 07:30:00|          35|         1729|2024-06-24 07:30:…|
5|62.41|312.05|      14|       GBM|       14|        GBM|             C|
C|           0|  0|ALSEA*|
+-------------------+------------+-------------+--------------------+------+----
-+------+--------+----------+---------+-----------+--------------+--------------
---+------------+---+------+
only showing top 2 rows

Primeros registros para el símbolo COST*:
+-------------------+------------+-------------+--------------------+------+----
---+--------+--------+----------+---------+-----------+--------------+----------
-------+------------+---+------+
|         trade_time|match_number|instrument_id|           timestamp|volume|
price|  amount|buyer_id|buyer_name|seller_id|seller_name|operation_type|concerta
tion_type|price_setter|lot|symbol|
+-------------------+------------+-------------+--------------------+------+----
---+--------+--------+----------+---------+-----------+--------------+----------
-------+------------+---+------+
|2024-06-24 07:53:44|           1|         2702|2024-06-24 07:53:…|
5|15260.0| 76300.0|       0|     BMCAP|        0|      BMCAP|             C|
C|           1|  1| COST*|
|2024-06-24 07:53:44|           2|         2702|2024-06-24 07:53:…|
9|15260.0|137340.0|       0|     BMCAP|        0|      BMCAP|             C|
C|           1|  1| COST*|
+-------------------+------------+-------------+--------------------+------+----
---+--------+--------+----------+---------+-----------+--------------+----------
-------+------------+---+------+
only showing top 2 rows

Primeros registros para el símbolo FEMSAUBD:
+-------------------+------------+-------------+--------------------+------+----
--+------+--------+----------+---------+-----------+--------------+-------------
----+------------+---+--------+
```

```
|         trade_time|match_number|instrument_id|           timestamp|volume| pri
ce|amount|buyer_id|buyer_name|seller_id|seller_name|operation_type|concertation_
type|price_setter|lot|  symbol|
+-------------------+------------+-------------+--------------------+------+----
--+------+--------+----------+---------+----------+-------------+-------------
----+------------+---+--------+
|2024-06-24 07:30:00|           8|            3|2024-06-24 07:30:…|
1|193.64|193.64|      14|       GBM|       14|        GBM|             C|
C|           0|  0|FEMSAUBD|
|2024-06-24 07:30:00|           6|            3|2024-06-24 07:30:…|
1|193.64|193.64|      14|       GBM|       14|        GBM|             C|
C|           0|  0|FEMSAUBD|
+-------------------+------------+-------------+--------------------+------+----
--+------+--------+----------+---------+----------+-------------+-------------
----+------------+---+--------+
only showing top 2 rows

Primeros registros para el símbolo WALMEX*:
+-------------------+------------+-------------+--------------------+------+----
-+------+--------+----------+---------+----------+-------------+--------------
---+------------+---+-------+
|         trade_time|match_number|instrument_id|           timestamp|volume|pric
e|amount|buyer_id|buyer_name|seller_id|seller_name|operation_type|concertation_t
ype|price_setter|lot| symbol|
+-------------------+------------+-------------+--------------------+------+----
-+------+--------+----------+---------+----------+-------------+--------------
---+------------+---+-------+
|2024-06-24 07:30:00|          52|            5|2024-06-24 07:30:…|
1|61.58| 61.58|       0|       FMX|       14|        GBM|             C|
O|           0|  0|WALMEX*|
|2024-06-24 07:30:00|          39|            5|2024-06-24 07:30:…|
10|61.53| 615.3|      14|       GBM|        0|        FMX|             C|
O|           0|  0|WALMEX*|
+-------------------+------------+-------------+--------------------+------+----
-+------+--------+----------+---------+----------+-------------+--------------
---+------------+---+-------+
only showing top 2 rows

Primeros registros para el símbolo LACOMERUBC:
+-------------------+------------+-------------+--------------------+------+----
-+------+--------+----------+---------+----------+-------------+--------------
---+------------+---+----------+
|         trade_time|match_number|instrument_id|           timestamp|volume|pric
e|amount|buyer_id|buyer_name|seller_id|seller_name|operation_type|concertation_t
ype|price_setter|lot|    symbol|
+-------------------+------------+-------------+--------------------+------+----
-+------+--------+----------+---------+----------+-------------+--------------
---+------------+---+----------+
```

```
|2024-06-24 07:30:00|          11|       351814|2024-06-24 07:30:…|
1|38.55| 38.55|      14|       GBM|       14|        GBM|             C|
C|           0|  0|LACOMERUBC|
|2024-06-24 07:30:00|           9|       351814|2024-06-24 07:30:…|
1|38.55| 38.55|      14|       GBM|       14|        GBM|             C|
C|           0|  0|LACOMERUBC|
+-------------------+------------+-------------+--------------------+------+----
-+-------+--------+----------+---------+-----------+--------------+-------------
---+------------+---+----------+
only showing top 2 rows

Primeros registros para el símbolo CHDRAUIB:
+-------------------+------------+-------------+--------------------+------+----
--+------+--------+----------+---------+-----------+--------------+-------------
----+------------+---+--------+
|         trade_time|match_number|instrument_id|           timestamp|volume| pri
ce|amount|buyer_id|buyer_name|seller_id|seller_name|operation_type|concertation_
type|price_setter|lot|  symbol|
+-------------------+------------+-------------+--------------------+------+----
--+------+--------+----------+---------+-----------+--------------+-------------
----+------------+---+--------+
|2024-06-24 07:30:00|          12|         6080|2024-06-24 07:30:…|
2|127.01|254.02|      14|       GBM|       14|        GBM|             C|
C|           0|  0|CHDRAUIB|
|2024-06-24 07:30:00|          19|         6080|2024-06-24 07:30:…|
4|127.01|508.04|      14|       GBM|      119|      ACTIN|             C|
O|           0|  0|CHDRAUIB|
+-------------------+------------+-------------+--------------------+------+----
--+------+--------+----------+---------+-----------+--------------+-------------
----+------------+---+--------+
only showing top 2 rows

Primeros registros para el símbolo WMT*:
+-------------------+------------+-------------+--------------------+------+----
---+-------+--------+----------+---------+-----------+--------------+-----------
------+------------+---+------+
|         trade_time|match_number|instrument_id|           timestamp|volume|
price| amount|buyer_id|buyer_name|seller_id|seller_name|operation_type|concertat
ion_type|price_setter|lot|symbol|
+-------------------+------------+-------------+--------------------+------+----
---+-------+--------+----------+---------+-----------+--------------+-----------
------+------------+---+------+
|2024-06-24 07:30:00|           2|         2056|2024-06-24 07:30:…|
1|1230.01|1230.01|       0|     BMCAP|        0|      BMCAP|             C|
C|           0|  0|  WMT*|
|2024-06-24 07:30:00|           3|         2056|2024-06-24 07:30:…|
1|1230.01|1230.01|       0|     BMCAP|        0|      BMCAP|             C|
O|           0|  0|  WMT*|
```

```
+------------------+-----------+------------+------------------+------+----
---+-------+--------+----------+---------+----------+-------------+----------
------+-----------+---+------+
only showing top 2 rows
```

[ ]:

```

```

[71]:

```
from pyspark.sql import functions as F
from pyspark.sql import Window
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.evaluation import RegressionEvaluator

# Crear un nuevo diccionario para almacenar los DataFrames procesados por␣
↪símbolo
df_symbols_5min = {}

# Iterar sobre cada símbolo en el diccionario original `df_symbols`
for symbol, df_a in df_symbols.items():

    # Crear columna de tiempo a nivel de 5 minutos usando el inicio del␣
↪intervalo
    df_5min = df_a.withColumn("datetime_5min", F.window("trade_time", "5␣
↪minutes").start)

    # Agrupar por intervalos de 5 minutos, obteniendo el último precio y␣
↪volumen acumulado
    df_5min_summary = df_5min.groupBy("datetime_5min") \
        .agg(
            F.last("price").alias("close"),            # Precio de cierre␣
↪(último registro del intervalo de 5 minutos)
            F.sum("volume").alias("volume")            # Volumen acumulado en␣
↪el intervalo de 5 minutos
        )

    # Agregar columna de inicio y fin del intervalo de 5 minutos para mayor␣
↪claridad
    df_5min_summary = df_5min_summary.withColumn("minute_start", F.
↪col("datetime_5min")) \
                                     .withColumn("minute_end", F.
↪expr("datetime_5min + interval 5 minute"))

    # Definir una ventana para desplazar (lag) los datos un intervalo hacia␣
↪atrás
    window_spec = Window.orderBy("datetime_5min")
```

```python
    # Agregar las columnas de "close" y "volume" desplazadas en un intervalo de␣
↪5 minutos hacia atrás
    df_5min_lagged = df_5min_summary.withColumn("lagged_close", F.lag("close",␣
↪1).over(window_spec)) \
                                        .withColumn("lagged_volume", F.
↪lag("volume", 1).over(window_spec))

    # Filtrar las filas donde los valores desplazados (lagged) sean nulos
    df_5min_lagged = df_5min_lagged.filter(F.col("lagged_close").isNotNull())

    # Ordenar cronológicamente por el inicio del intervalo de 5 minutos
    df_5min_lagged = df_5min_lagged.orderBy("minute_start")

    # Almacenar el DataFrame procesado en el diccionario de resultados
    df_symbols_5min[symbol] = df_5min_lagged

# Ejemplo: mostrar los primeros registros para cada símbolo procesado
for symbol, df in df_symbols_5min.items():
    print(f"Primeros registros procesados para el símbolo {symbol}:")
    df.orderBy("minute_start").show(2)
```

```
Primeros registros procesados para el símbolo SORIANAB:
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
|      datetime_5min|close|volume|       minute_start|
minute_end|lagged_close|lagged_volume|
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
|2024-06-24 07:35:00| 29.7|    10|2024-06-24 07:35:00|2024-06-24 07:40:00|
29.34|           46|
|2024-06-24 08:05:00|30.09|     3|2024-06-24 08:05:00|2024-06-24 08:10:00|
29.7|           10|
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
only showing top 2 rows

Primeros registros procesados para el símbolo BIMBOA:
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
|      datetime_5min|close|volume|       minute_start|
minute_end|lagged_close|lagged_volume|
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
|2024-06-24 07:35:00|68.35|   720|2024-06-24 07:35:00|2024-06-24 07:40:00|
68.44|         1907|
|2024-06-24 07:40:00|68.34|  1909|2024-06-24 07:40:00|2024-06-24 07:45:00|
68.35|          720|
```

```
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
only showing top 2 rows

Primeros registros procesados para el símbolo ALSEA*:
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
|      datetime_5min|close|volume|       minute_start|
minute_end|lagged_close|lagged_volume|
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
|2024-06-24 07:35:00| 62.8|  1760|2024-06-24 07:35:00|2024-06-24 07:40:00|
62.55|         1907|
|2024-06-24 07:40:00|62.69|  4193|2024-06-24 07:40:00|2024-06-24 07:45:00|
62.8|         1760|
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
only showing top 2 rows

Primeros registros procesados para el símbolo COST*:
+-------------------+--------+------+-------------------+-------------------+---
---------+-------------+
|      datetime_5min|   close|volume|       minute_start|
minute_end|lagged_close|lagged_volume|
+-------------------+--------+------+-------------------+-------------------+---
---------+-------------+
|2024-06-24 08:40:00|15220.01|     8|2024-06-24 08:40:00|2024-06-24 08:45:00|
15260.0|           14|
|2024-06-24 09:15:00| 15286.5|     6|2024-06-24 09:15:00|2024-06-24 09:20:00|
15220.01|            8|
+-------------------+--------+------+-------------------+-------------------+---
---------+-------------+
only showing top 2 rows

Primeros registros procesados para el símbolo FEMSAUBD:
+-------------------+------+------+-------------------+-------------------+-----
-------+-------------+
|      datetime_5min| close|volume|       minute_start|
minute_end|lagged_close|lagged_volume|
+-------------------+------+------+-------------------+-------------------+-----
-------+-------------+
|2024-06-24 07:35:00|195.03|  3895|2024-06-24 07:35:00|2024-06-24 07:40:00|
194.05|         3835|
|2024-06-24 07:40:00|195.81|  5636|2024-06-24 07:40:00|2024-06-24 07:45:00|
195.03|         3895|
+-------------------+------+------+-------------------+-------------------+-----
-------+-------------+
only showing top 2 rows
```

```
Primeros registros procesados para el símbolo WALMEX*:
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
|      datetime_5min|close|volume|       minute_start|
minute_end|lagged_close|lagged_volume|
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
|2024-06-24 07:35:00|61.87|165811|2024-06-24 07:35:00|2024-06-24 07:40:00|
61.6|        80481|
|2024-06-24 07:40:00|61.88| 72170|2024-06-24 07:40:00|2024-06-24 07:45:00|
61.87|       165811|
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
only showing top 2 rows

Primeros registros procesados para el símbolo LACOMERUBC:
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
|      datetime_5min|close|volume|       minute_start|
minute_end|lagged_close|lagged_volume|
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
|2024-06-24 07:35:00| 38.0|    10|2024-06-24 07:35:00|2024-06-24 07:40:00|
38.55|          589|
|2024-06-24 07:45:00|38.53|  1398|2024-06-24 07:45:00|2024-06-24 07:50:00|
38.0|           10|
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
only showing top 2 rows

Primeros registros procesados para el símbolo CHDRAUIB:
+-------------------+------+------+-------------------+-------------------+-----
-------+-------------+
|      datetime_5min| close|volume|       minute_start|
minute_end|lagged_close|lagged_volume|
+-------------------+------+------+-------------------+-------------------+-----
-------+-------------+
|2024-06-24 07:35:00|127.01|   390|2024-06-24 07:35:00|2024-06-24 07:40:00|
127.01|          459|
|2024-06-24 07:40:00|127.43|  1037|2024-06-24 07:40:00|2024-06-24 07:45:00|
127.01|          390|
+-------------------+------+------+-------------------+-------------------+-----
-------+-------------+
only showing top 2 rows

Primeros registros procesados para el símbolo WMT*:
+-------------------+-------+------+-------------------+-------------------+----
```

```
--------+-------------+
|      datetime_5min|  close|volume|       minute_start|
minute_end|lagged_close|lagged_volume|
+-------------------+-------+------+-------------------+-------------------+----
--------+-------------+
|2024-06-24 08:30:00|1225.02|    27|2024-06-24 08:30:00|2024-06-24 08:35:00|
1230.01|           91|
|2024-06-24 08:35:00| 1239.0|     4|2024-06-24 08:35:00|2024-06-24 08:40:00|
1225.02|           27|
+-------------------+-------+------+-------------------+-------------------+----
--------+-------------+
only showing top 2 rows
```

[ ]:

[72]:

```python
from pyspark.sql import functions as F
from pyspark.sql import Window

# Crear un nuevo diccionario para almacenar los DataFrames agrupados a 5␣
↪minutos por símbolo
df_symbols_5min_ohlc = {}

# Iterar sobre cada símbolo en el diccionario original `df_symbols`
for symbol, df_a in df_symbols.items():

    # Crear columna de tiempo a nivel de 5 minutos usando el inicio del␣
↪intervalo
    df_5min = df_a.withColumn("datetime_5min", F.window("trade_time", "5␣
↪minutes").start)

    # Agrupar por intervalos de 5 minutos y calcular los valores de OHLC y␣
↪volumen
    df_5min_summary = df_5min.groupBy("datetime_5min") \
        .agg(
            F.first("price").alias("open"),          # Precio de apertura␣
↪(primer registro del intervalo de 5 minutos)
            F.max("price").alias("high"),            # Precio más alto
            F.min("price").alias("low"),             # Precio más bajo
            F.last("price").alias("close"),          # Precio de cierre
            F.sum("volume").alias("volume")          # Volumen acumulado en el␣
↪intervalo de 5 minutos
        )

    # Agregar columna de inicio y fin del intervalo de 5 minutos para mayor␣
↪claridad
```

```
    df_5min_summary = df_5min_summary.withColumn("minute_start", F.
 ↪col("datetime_5min")) \
                                     .withColumn("minute_end", F.
 ↪expr("datetime_5min + interval 5 minute"))


    # Almacenar el DataFrame procesado en el diccionario de resultados
    df_symbols_5min_ohlc[symbol] = df_5min_summary

# Ejemplo: mostrar los primeros registros para cada símbolo procesado en la␣
 ↪agregación OHLC
for symbol, df in df_symbols_5min.items():
    print(f"Primeros registros para el símbolo {symbol} (OHLC):")
    df.orderBy("datetime_5min").show(2)
```

```
Primeros registros para el símbolo SORIANAB (OHLC):
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
|      datetime_5min|close|volume|       minute_start|
minute_end|lagged_close|lagged_volume|
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
|2024-06-24 07:35:00| 29.7|    10|2024-06-24 07:35:00|2024-06-24 07:40:00|
29.34|           46|
|2024-06-24 08:05:00|30.09|     3|2024-06-24 08:05:00|2024-06-24 08:10:00|
29.7|           10|
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
only showing top 2 rows

Primeros registros para el símbolo BIMBOA (OHLC):
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
|      datetime_5min|close|volume|       minute_start|
minute_end|lagged_close|lagged_volume|
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
|2024-06-24 07:35:00|68.35|   720|2024-06-24 07:35:00|2024-06-24 07:40:00|
68.44|         1907|
|2024-06-24 07:40:00|68.34|  1909|2024-06-24 07:40:00|2024-06-24 07:45:00|
68.35|          720|
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
only showing top 2 rows

Primeros registros para el símbolo ALSEA* (OHLC):
+-------------------+-----+------+-------------------+-------------------+------
```

```
------+-------------+
|      datetime_5min|close|volume|       minute_start|
minute_end|lagged_close|lagged_volume|
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
|2024-06-24 07:35:00| 62.8|  1760|2024-06-24 07:35:00|2024-06-24 07:40:00|
62.55|         1907|
|2024-06-24 07:40:00|62.69|  4193|2024-06-24 07:40:00|2024-06-24 07:45:00|
62.8|         1760|
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
only showing top 2 rows

Primeros registros para el símbolo COST* (OHLC):
+-------------------+--------+------+-------------------+-------------------+---
---------+-------------+
|      datetime_5min|   close|volume|       minute_start|
minute_end|lagged_close|lagged_volume|
+-------------------+--------+------+-------------------+-------------------+---
---------+-------------+
|2024-06-24 08:40:00|15220.01|     8|2024-06-24 08:40:00|2024-06-24 08:45:00|
15260.0|           14|
|2024-06-24 09:15:00| 15286.5|     6|2024-06-24 09:15:00|2024-06-24 09:20:00|
15220.01|            8|
+-------------------+--------+------+-------------------+-------------------+---
---------+-------------+
only showing top 2 rows

Primeros registros para el símbolo FEMSAUBD (OHLC):
+-------------------+------+------+-------------------+-------------------+-----
-------+-------------+
|      datetime_5min| close|volume|       minute_start|
minute_end|lagged_close|lagged_volume|
+-------------------+------+------+-------------------+-------------------+-----
-------+-------------+
|2024-06-24 07:35:00|195.03|  3895|2024-06-24 07:35:00|2024-06-24 07:40:00|
194.05|         3835|
|2024-06-24 07:40:00|195.81|  5636|2024-06-24 07:40:00|2024-06-24 07:45:00|
195.03|         3895|
+-------------------+------+------+-------------------+-------------------+-----
-------+-------------+
only showing top 2 rows

Primeros registros para el símbolo WALMEX* (OHLC):
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
|      datetime_5min|close|volume|       minute_start|
minute_end|lagged_close|lagged_volume|
```

```
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
|2024-06-24 07:35:00|61.87|165811|2024-06-24 07:35:00|2024-06-24 07:40:00|
61.6|        80481|
|2024-06-24 07:40:00|61.88| 72170|2024-06-24 07:40:00|2024-06-24 07:45:00|
61.87|       165811|
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
only showing top 2 rows

Primeros registros para el símbolo LACOMERUBC (OHLC):
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
|      datetime_5min|close|volume|       minute_start|
minute_end|lagged_close|lagged_volume|
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
|2024-06-24 07:35:00| 38.0|    10|2024-06-24 07:35:00|2024-06-24 07:40:00|
38.55|          589|
|2024-06-24 07:45:00|38.53|  1398|2024-06-24 07:45:00|2024-06-24 07:50:00|
38.0|           10|
+-------------------+-----+------+-------------------+-------------------+------
------+-------------+
only showing top 2 rows

Primeros registros para el símbolo CHDRAUIB (OHLC):
+-------------------+------+------+-------------------+-------------------+-----
-------+-------------+
|      datetime_5min| close|volume|       minute_start|
minute_end|lagged_close|lagged_volume|
+-------------------+------+------+-------------------+-------------------+-----
-------+-------------+
|2024-06-24 07:35:00|127.01|   390|2024-06-24 07:35:00|2024-06-24 07:40:00|
127.01|          459|
|2024-06-24 07:40:00|127.43|  1037|2024-06-24 07:40:00|2024-06-24 07:45:00|
127.01|          390|
+-------------------+------+------+-------------------+-------------------+-----
-------+-------------+
only showing top 2 rows

Primeros registros para el símbolo WMT* (OHLC):
+-------------------+-------+------+-------------------+-------------------+----
--------+-------------+
|      datetime_5min|  close|volume|       minute_start|
minute_end|lagged_close|lagged_volume|
+-------------------+-------+------+-------------------+-------------------+----
--------+-------------+
|2024-06-24 08:30:00|1225.02|    27|2024-06-24 08:30:00|2024-06-24 08:35:00|
```

```
1230.01|           91|
|2024-06-24 08:35:00| 1239.0|     4|2024-06-24 08:35:00|2024-06-24 08:40:00|
1225.02|           27|
+-------------------+-------+------+-------------------+-------------------+----
--------+-------------+
only showing top 2 rows
```

[73]:

```python
# Crear un nuevo diccionario para almacenar los DataFrames con lagging por␣
↪símbolo
df_symbols_5min_ohlc_lagged = {}

# Iterar sobre cada símbolo en el diccionario de DataFrames `df_symbols_5min`
for symbol, df_5min_ohlc_summary in df_symbols_5min_ohlc.items():

    # Definir una ventana para desplazar (lag) los datos un intervalo hacia␣
↪atrás
    window_spec = Window.orderBy("datetime_5min")

    # Agregar las columnas de "close" y "volume" desplazadas en un intervalo de␣
↪5 minutos hacia atrás
    df_5min_ohlc_lagged = df_5min_ohlc_summary.withColumn("lagged_close", F.
↪lag("close", 1).over(window_spec)) \
                                              .withColumn("lagged_volume", F.
↪lag("volume", 1).over(window_spec))

    # Filtrar las filas donde los valores desplazados (lagged) sean nulos
    df_5min_ohlc_lagged = df_5min_ohlc_lagged.filter(F.col("lagged_close").
↪isNotNull())

    # Ordenar cronológicamente por el inicio del intervalo de 5 minutos y␣
↪almacenar el DataFrame
    df_symbols_5min_ohlc_lagged[symbol] = df_5min_ohlc_lagged.
↪orderBy("datetime_5min")

# Ejemplo: mostrar los primeros registros para cada símbolo procesado con␣
↪lagging
for symbol, df in df_symbols_5min_ohlc_lagged.items():
    print(f"Primeros registros procesados para el símbolo {symbol} con lag:")
    df.orderBy("datetime_5min").show(2)
```

```
Primeros registros procesados para el símbolo SORIANAB con lag:
+-------------------+-----+-----+-----+-----+------+-------------------+--------
-----------+------------+-------------+
|      datetime_5min| open| high|  low|close|volume|       minute_start|
minute_end|lagged_close|lagged_volume|
+-------------------+-----+-----+-----+-----+------+-------------------+--------
```

```
-----------+------------+-------------+
|2024-06-24 07:35:00|30.34|30.34| 29.7| 29.7|    10|2024-06-24
07:35:00|2024-06-24 07:40:00|       29.34|           46|
|2024-06-24 08:05:00|30.09|30.09|30.09|30.09|     3|2024-06-24
08:05:00|2024-06-24 08:10:00|        29.7|           10|
+-------------------+-----+-----+-----+-----+------+-------------------+--------
-----------+------------+-------------+
only showing top 2 rows

Primeros registros procesados para el símbolo BIMBOA con lag:
+-------------------+-----+-----+-----+-----+------+-------------------+--------
-----------+------------+-------------+
|      datetime_5min| open| high|  low|close|volume|       minute_start|
minute_end|lagged_close|lagged_volume|
+-------------------+-----+-----+-----+-----+------+-------------------+--------
-----------+------------+-------------+
|2024-06-24 07:35:00|68.35|68.35|68.24|68.35|   720|2024-06-24
07:35:00|2024-06-24 07:40:00|       68.44|         1907|
|2024-06-24 07:40:00|68.24|68.34|68.24|68.34|  1909|2024-06-24
07:40:00|2024-06-24 07:45:00|       68.35|          720|
+-------------------+-----+-----+-----+-----+------+-------------------+--------
-----------+------------+-------------+
only showing top 2 rows

Primeros registros procesados para el símbolo ALSEA* con lag:
+-------------------+-----+-----+-----+-----+------+-------------------+--------
-----------+------------+-------------+
|      datetime_5min| open| high|  low|close|volume|       minute_start|
minute_end|lagged_close|lagged_volume|
+-------------------+-----+-----+-----+-----+------+-------------------+--------
-----------+------------+-------------+
|2024-06-24 07:35:00|62.82|62.85|62.69| 62.8|  1760|2024-06-24
07:35:00|2024-06-24 07:40:00|       62.55|         1907|
|2024-06-24 07:40:00|62.69| 62.8|62.41|62.69|  4193|2024-06-24
07:40:00|2024-06-24 07:45:00|        62.8|         1760|
+-------------------+-----+-----+-----+-----+------+-------------------+--------
-----------+------------+-------------+
only showing top 2 rows

Primeros registros procesados para el símbolo COST* con lag:
+-------------------+--------+--------+-------+--------+------+-----------------
--+-------------------+------------+-------------+
|      datetime_5min|    open|    high|    low|   close|volume|
minute_start|         minute_end|lagged_close|lagged_volume|
+-------------------+--------+--------+-------+--------+------+-----------------
--+-------------------+------------+-------------+
|2024-06-24 08:40:00|15220.01|15220.01|15220.0|15220.01|     8|2024-06-24
08:40:00|2024-06-24 08:45:00|     15260.0|           14|
```

```
|2024-06-24 09:15:00| 15286.5| 15286.5|15286.5| 15286.5|     6|2024-06-24
09:15:00|2024-06-24 09:20:00|    15220.01|            8|
+-------------------+--------+--------+-------+--------+------+-----------------
--+-------------------+------------+-------------+
only showing top 2 rows

Primeros registros procesados para el símbolo FEMSAUBD con lag:
+-------------------+------+------+------+------+------+-------------------+----
---------------+------------+-------------+
|      datetime_5min|  open|  high|   low| close|volume|       minute_start|
minute_end|lagged_close|lagged_volume|
+-------------------+------+------+------+------+------+-------------------+----
---------------+------------+-------------+
|2024-06-24 07:35:00|195.02|195.04|193.76|195.03|  3895|2024-06-24
07:35:00|2024-06-24 07:40:00|      194.05|         3835|
|2024-06-24 07:40:00|195.79|195.85|195.04|195.81|  5636|2024-06-24
07:40:00|2024-06-24 07:45:00|      195.03|         3895|
+-------------------+------+------+------+------+------+-------------------+----
---------------+------------+-------------+
only showing top 2 rows

Primeros registros procesados para el símbolo WALMEX* con lag:
+-------------------+-----+-----+-----+-----+------+-------------------+--------
-----------+------------+-------------+
|      datetime_5min| open| high|  low|close|volume|       minute_start|
minute_end|lagged_close|lagged_volume|
+-------------------+-----+-----+-----+-----+------+-------------------+--------
-----------+------------+-------------+
|2024-06-24 07:35:00| 61.8|61.96|61.41|61.87|165811|2024-06-24
07:35:00|2024-06-24 07:40:00|        61.6|        80481|
|2024-06-24 07:40:00|61.89|62.24|61.83|61.88| 72170|2024-06-24
07:40:00|2024-06-24 07:45:00|       61.87|       165811|
+-------------------+-----+-----+-----+-----+------+-------------------+--------
-----------+------------+-------------+
only showing top 2 rows

Primeros registros procesados para el símbolo LACOMERUBC con lag:
+-------------------+----+-----+-----+-----+------+-------------------+---------
----------+------------+-------------+
|      datetime_5min|open| high|  low|close|volume|       minute_start|
minute_end|lagged_close|lagged_volume|
+-------------------+----+-----+-----+-----+------+-------------------+---------
----------+------------+-------------+
|2024-06-24 07:35:00|38.0| 38.0| 38.0| 38.0|    10|2024-06-24
07:35:00|2024-06-24 07:40:00|       38.55|          589|
|2024-06-24 07:45:00|38.6|38.96|38.53|38.53|  1398|2024-06-24
07:45:00|2024-06-24 07:50:00|        38.0|           10|
+-------------------+----+-----+-----+-----+------+-------------------+---------
```

```
----------+------------+-------------+
only showing top 2 rows

Primeros registros procesados para el símbolo CHDRAUIB con lag:
+-------------------+------+------+------+------+------+-------------------+----
---------------+------------+-------------+
|      datetime_5min|  open|  high|   low| close|volume|       minute_start|
minute_end|lagged_close|lagged_volume|
+-------------------+------+------+------+------+------+-------------------+----
---------------+------------+-------------+
|2024-06-24 07:35:00|126.99| 127.6|126.99|127.01|   390|2024-06-24
07:35:00|2024-06-24 07:40:00|      127.01|          459|
|2024-06-24 07:40:00|127.16|127.65|127.16|127.43|  1037|2024-06-24
07:40:00|2024-06-24 07:45:00|      127.01|          390|
+-------------------+------+------+------+------+------+-------------------+----
---------------+------------+-------------+
only showing top 2 rows

Primeros registros procesados para el símbolo WMT* con lag:
+-------------------+-------+-------+-------+-------+------+-------------------
+-------------------+------------+-------------+
|      datetime_5min|   open|   high|    low|  close|volume|       minute_start|
minute_end|lagged_close|lagged_volume|
+-------------------+-------+-------+-------+-------+------+-------------------
+-------------------+------------+-------------+
|2024-06-24 08:30:00|1225.02|1225.02|1225.01|1225.02|    27|2024-06-24
08:30:00|2024-06-24 08:35:00|     1230.01|           91|
|2024-06-24 08:35:00| 1239.0| 1239.0| 1239.0| 1239.0|     4|2024-06-24
08:35:00|2024-06-24 08:40:00|     1225.02|           27|
+-------------------+-------+-------+-------+-------+------+-------------------
+-------------------+------------+-------------+
only showing top 2 rows
```

[ ]:

```

```

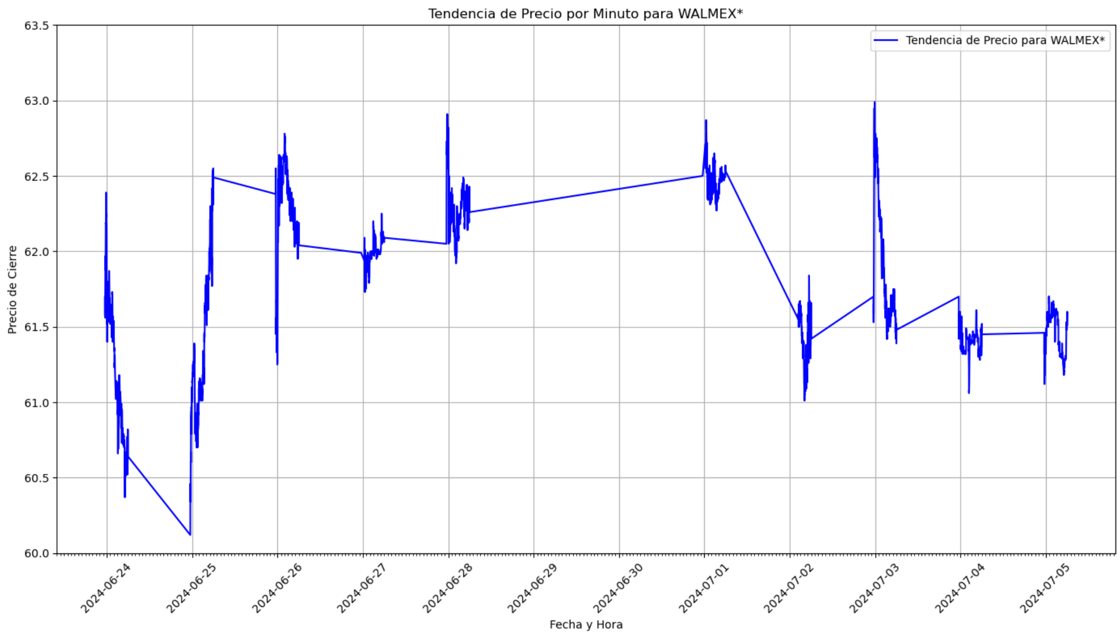
[ ]:

```

```

[74]:

```
import matplotlib.pyplot as plt
import pandas as pd
from pyspark.sql import functions as F

# Filtrar el DataFrame solo para el símbolo "WALMEX*"
df_walmex = df_symbols["WALMEX*"]

# Crear columna de tiempo a nivel de minuto usando el inicio del intervalo
```

```python
df_minute = df_walmex.withColumn("datetime_minute", F.window("trade_time", "1␣
↪minute").start)

# Agrupar por minuto y obtener el último precio (close) en cada minuto
df_minute_close = df_minute.groupBy("datetime_minute") \
    .agg(
        F.last("price").alias("close")         # Precio de cierre del minuto
    ).orderBy("datetime_minute")

# Convertir el DataFrame de Spark a Pandas
pandas_df = df_minute_close.toPandas()

# Asegurar que la columna de tiempo está en formato datetime y ordenada
pandas_df["datetime_minute"] = pd.to_datetime(pandas_df["datetime_minute"])
pandas_df = pandas_df.sort_values("datetime_minute")

# Graficar la tendencia del precio de cierre para "WALMEX*"
plt.figure(figsize=(14, 8))
plt.plot(pandas_df["datetime_minute"], pandas_df["close"], label="Tendencia de␣
↪Precio para WALMEX*", color="blue")

# Configurar el formato del eje X para mostrar solo el día y la hora en␣
↪intervalos de n horas
plt.gca().xaxis.set_major_formatter(plt.matplotlib.dates.
↪DateFormatter("%Y-%m-%d"))
plt.gca().xaxis.set_major_locator(plt.matplotlib.dates.
↪HourLocator(interval=24))  # Etiqueta cada n horas
plt.gca().xaxis.set_minor_locator(plt.matplotlib.dates.HourLocator(interval=1))␣
↪ # Divisiones menores cada hora
plt.xticks(rotation=45)

# Configuración del eje Y
plt.ylim(60, 63.5)  # Escala del eje Y

# Etiquetas y título
plt.xlabel("Fecha y Hora")
plt.ylabel("Precio de Cierre")
plt.title("Tendencia de Precio por Minuto para WALMEX*")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



[ ]:

```
```

[75]:

```
from pyspark.sql import functions as F
from pyspark.sql import Window
import pandas as pd
import matplotlib.pyplot as plt
import mplfinance as mpf

# Crear un nuevo diccionario para almacenar los DataFrames agrupados por día␣
↪por símbolo
df_symbols_daily_ohlc = {}

# Iterar sobre cada símbolo en el diccionario original `df_symbols`
for symbol, df_a in df_symbols.items():

    # Crear columna de tiempo a nivel de día usando el inicio del intervalo
    df_daily = df_a.withColumn("datetime_day", F.to_date("trade_time"))

    # Agrupar por día y calcular los valores de OHLC y volumen
    df_daily_summary = df_daily.groupBy("datetime_day") \
        .agg(
            F.first("price").alias("open"),          # Precio de apertura␣
↪(primer registro del día)
            F.max("price").alias("high"),            # Precio más alto del día
            F.min("price").alias("low"),             # Precio más bajo del día
            F.last("price").alias("close"),          # Precio de cierre del día
```

```python
            F.sum("volume").alias("volume")         # Volumen acumulado del día
        ).orderBy("datetime_day")

    # Almacenar el DataFrame procesado en el diccionario de resultados
    df_symbols_daily_ohlc[symbol] = df_daily_summary

# Generar gráficos de velas para cada símbolo
for symbol, df in df_symbols_daily_ohlc.items():
    # Convertir el DataFrame de Spark a Pandas
    pandas_df = df.toPandas()

    # Convertir la columna de fecha a formato datetime para mplfinance
    pandas_df["datetime_day"] = pd.to_datetime(pandas_df["datetime_day"])
    pandas_df.set_index("datetime_day", inplace=True)

    # Graficar con mplfinance
    mpf.plot(
        pandas_df,
        type='candle',
        style='charles',
        title=f"Gráfico de Velas Diarias para {symbol}",
        ylabel='Precio',
        volume=True,
        ylabel_lower='Volumen',
        datetime_format='%Y-%m-%d'
    )
```

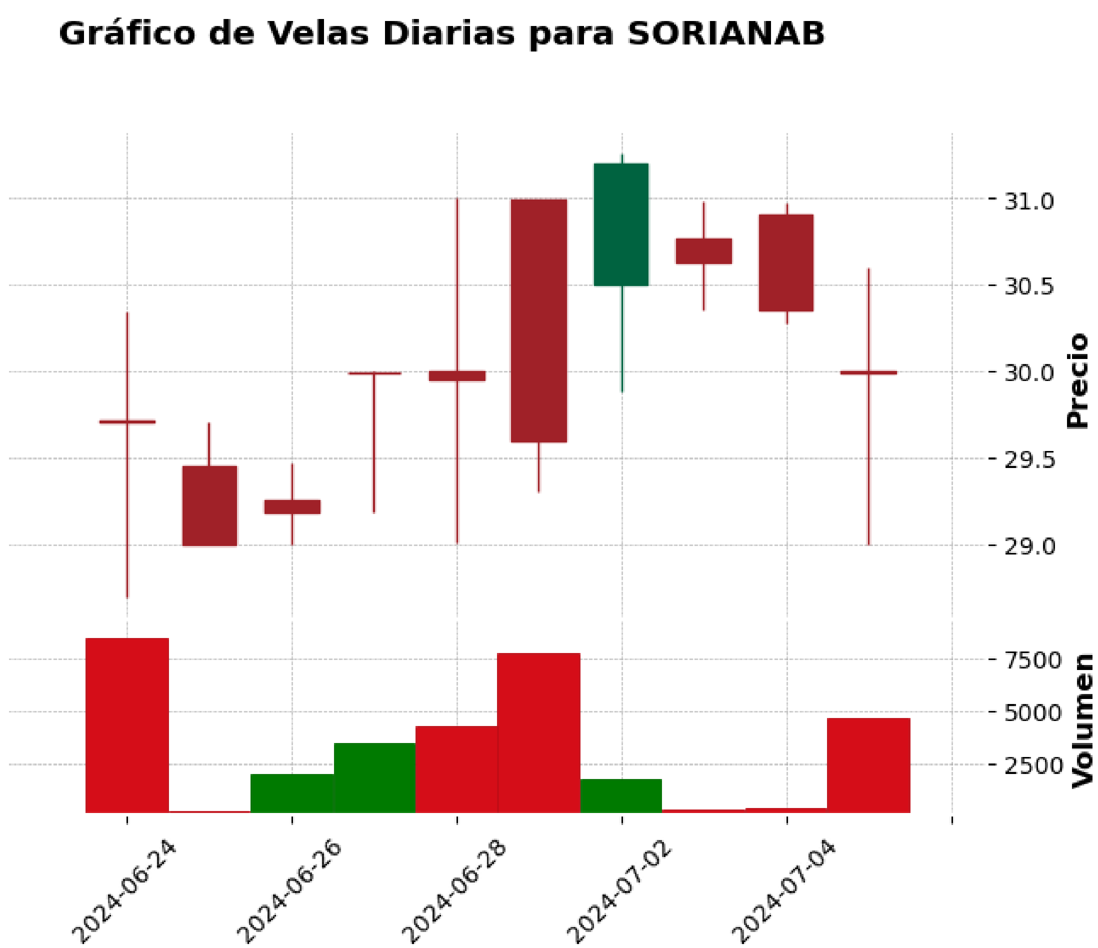


Gráfico de Velas Diarias para SORIANAB
31.0
30.5
30.0
29.5
29.0
Precio
7500
5000
2500
Volumen
2024-06-24
2024-06-26
2024-06-28
2024-07-02
2024-07-04

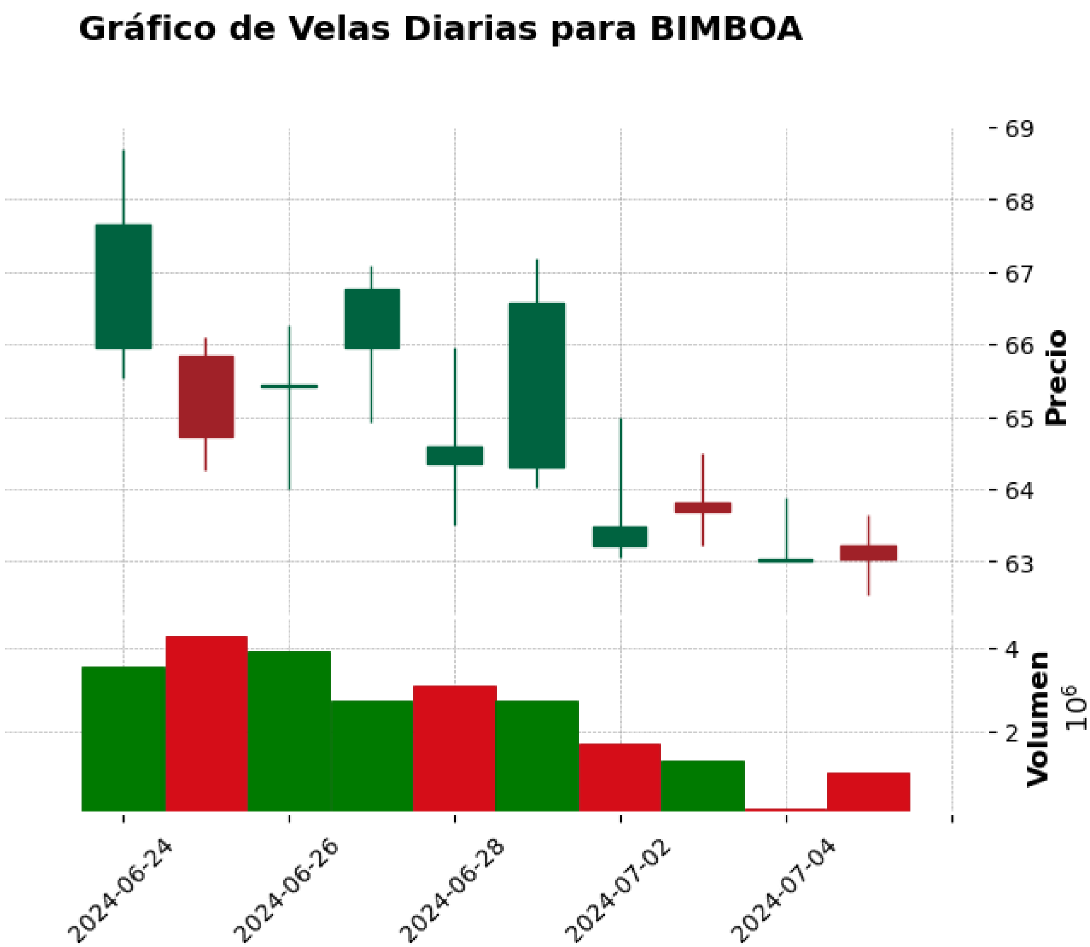


Gráfico de Velas Diarias para BIMBOA
69
68
67
66
65
64
63
Precio
4
2
Volumen
10^6
2024-06-24
2024-06-26
2024-06-28
2024-07-02
2024-07-04

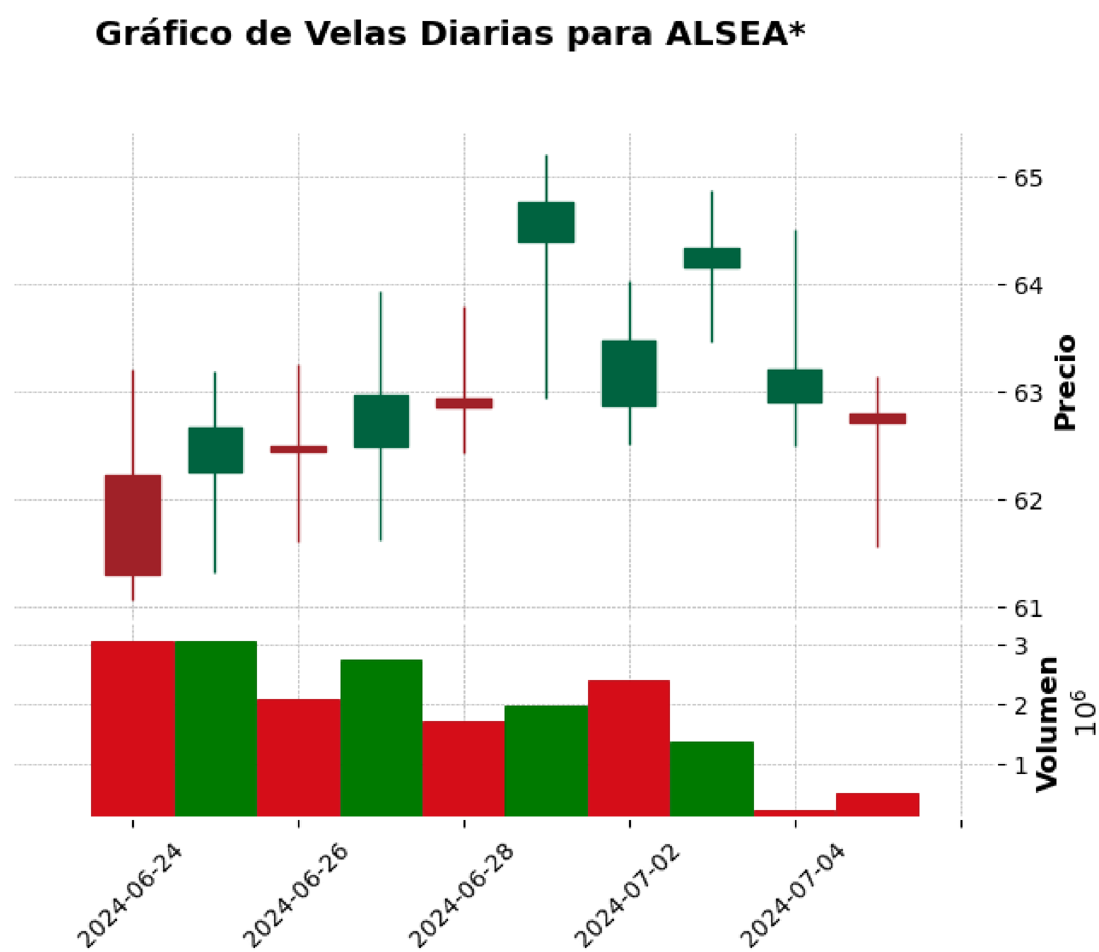


Gráfico de Velas Diarias para ALSEA*
65
64
63
62
61
Precio
3
2
1
Volumen
10^6
2024-06-24
2024-06-26
2024-06-28
2024-07-02
2024-07-04

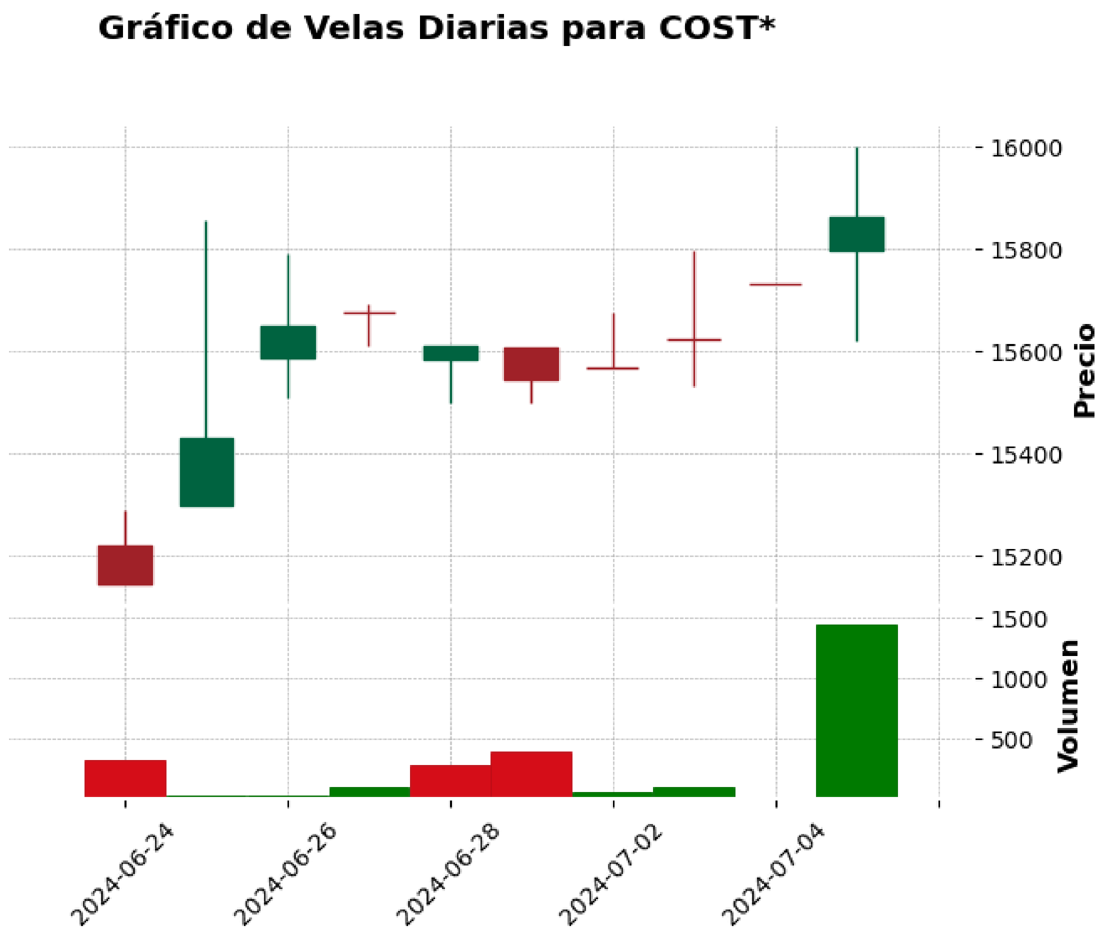


Gráfico de Velas Diarias para COST*
16000
15800
15600
15400
15200
Precio
1500
1000
500
Volumen
2024-06-24
2024-06-26
2024-06-28
2024-07-02
2024-07-04

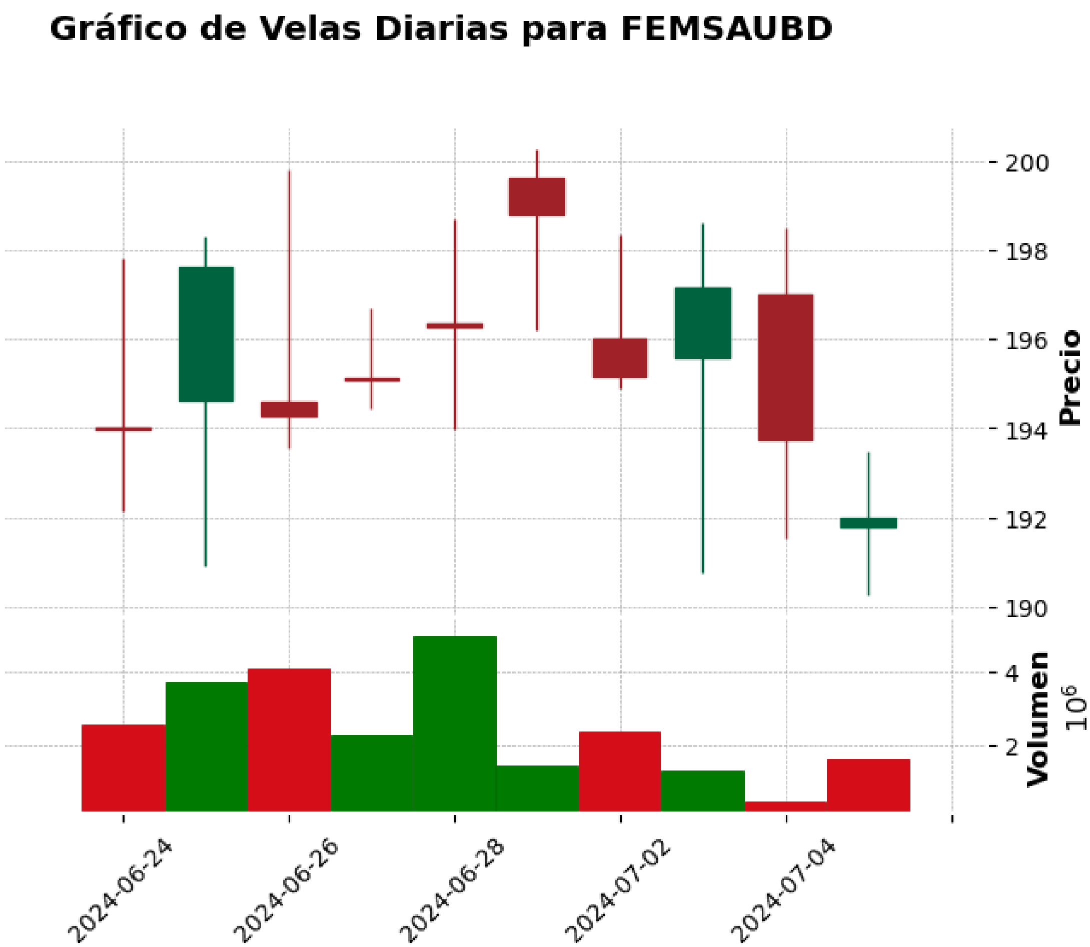


Gráfico de Velas Diarias para FEMSAUBD
Precio
200
198
196
194
192
190
Volumen
10^6
4
2
2024-06-24
2024-06-26
2024-06-28
2024-07-02
2024-07-04

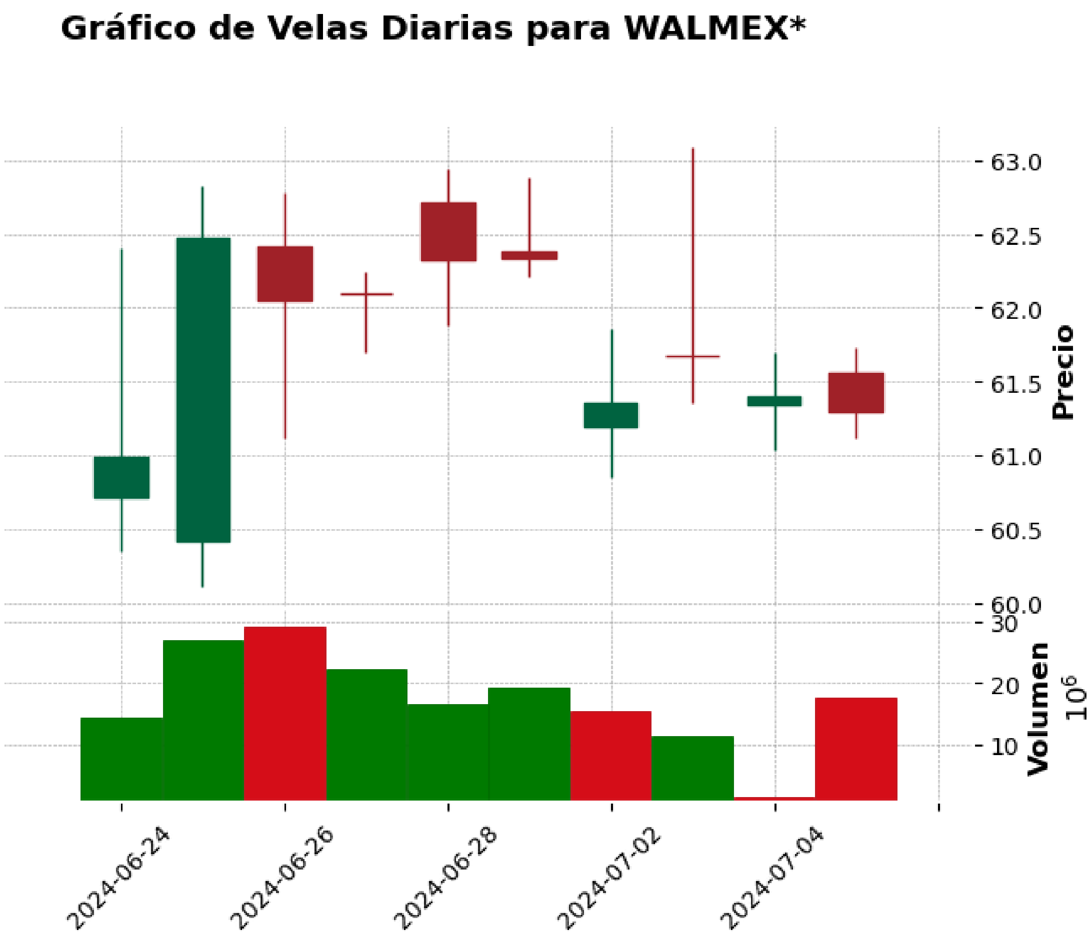


Gráfico de Velas Diarias para WALMEX*
Precio
Volumen
10^6
63.0
62.5
62.0
61.5
61.0
60.5
60.0
30
20
10
2024-06-24
2024-06-26
2024-06-28
2024-07-02
2024-07-04

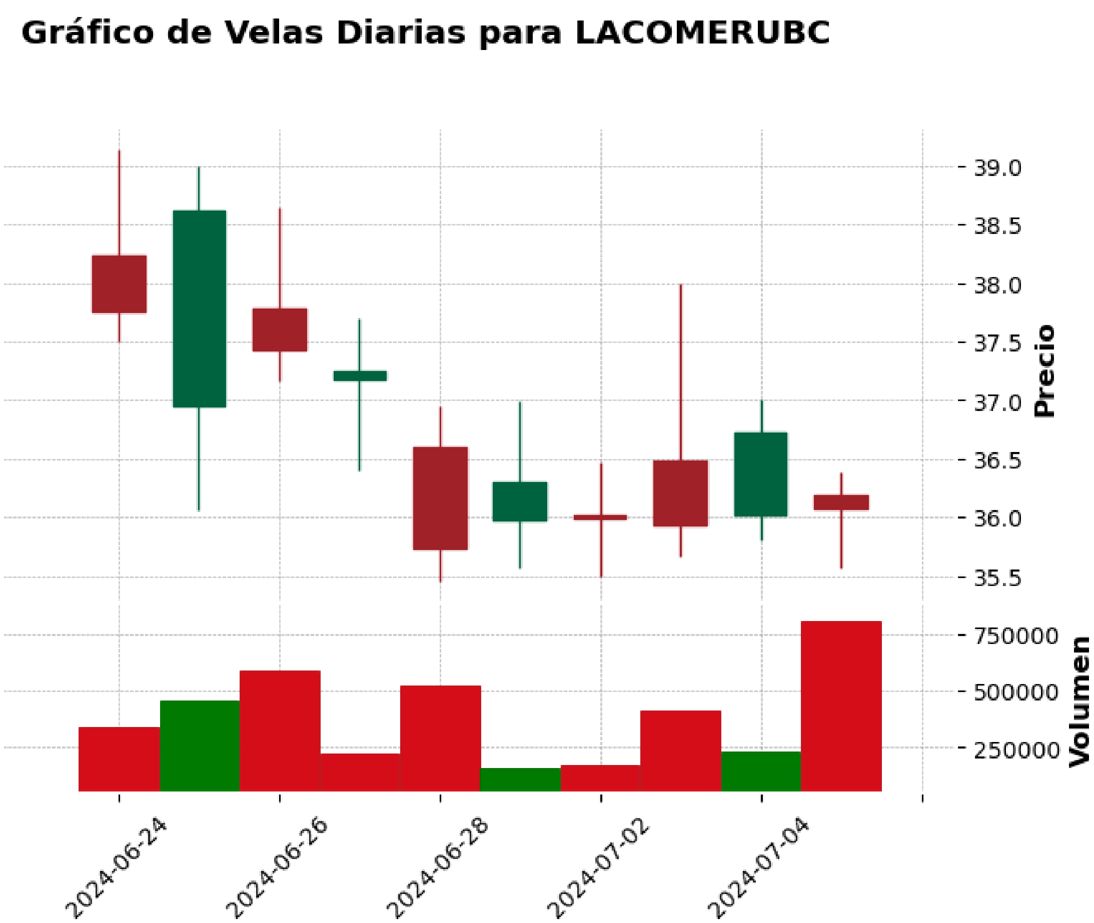


Gráfico de Velas Diarias para LACOMERUBC
39.0
38.5
38.0
37.5
37.0
36.5
36.0
35.5
Precio
750000
500000
250000
Volumen
2024-06-24
2024-06-26
2024-06-28
2024-07-02
2024-07-04

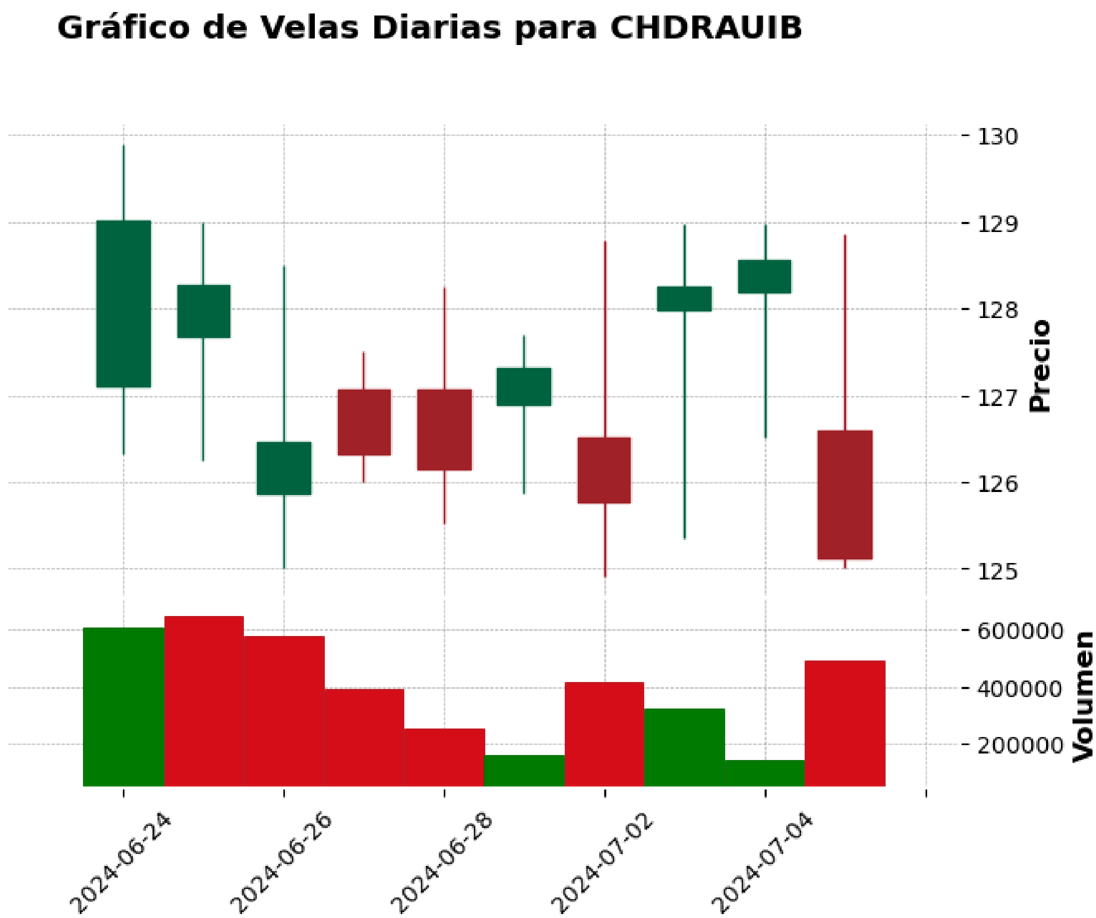
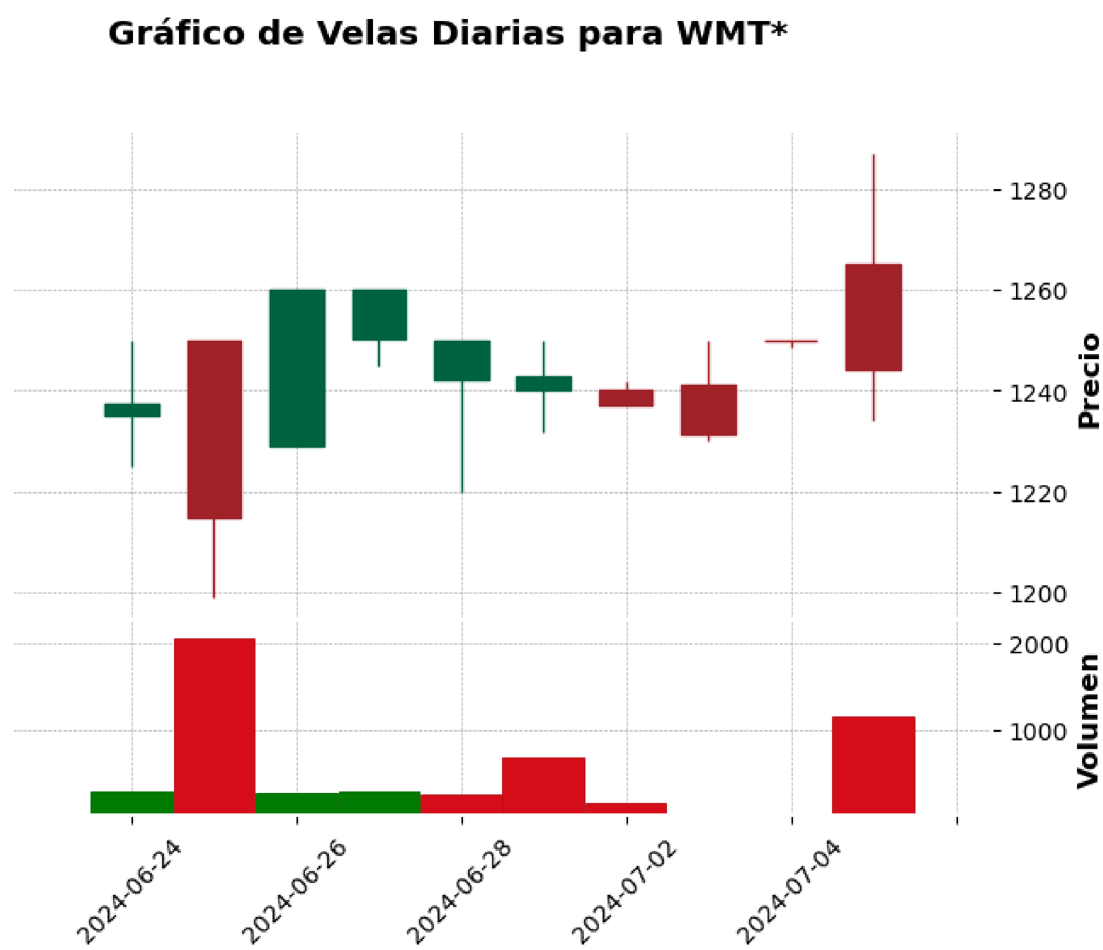


Gráfico de Velas Diarias para CHDRAUIB
130
129
128
127
126
125
Precio
600000
400000
200000
Volumen
2024-06-24
2024-06-26
2024-06-28
2024-07-02
2024-07-04



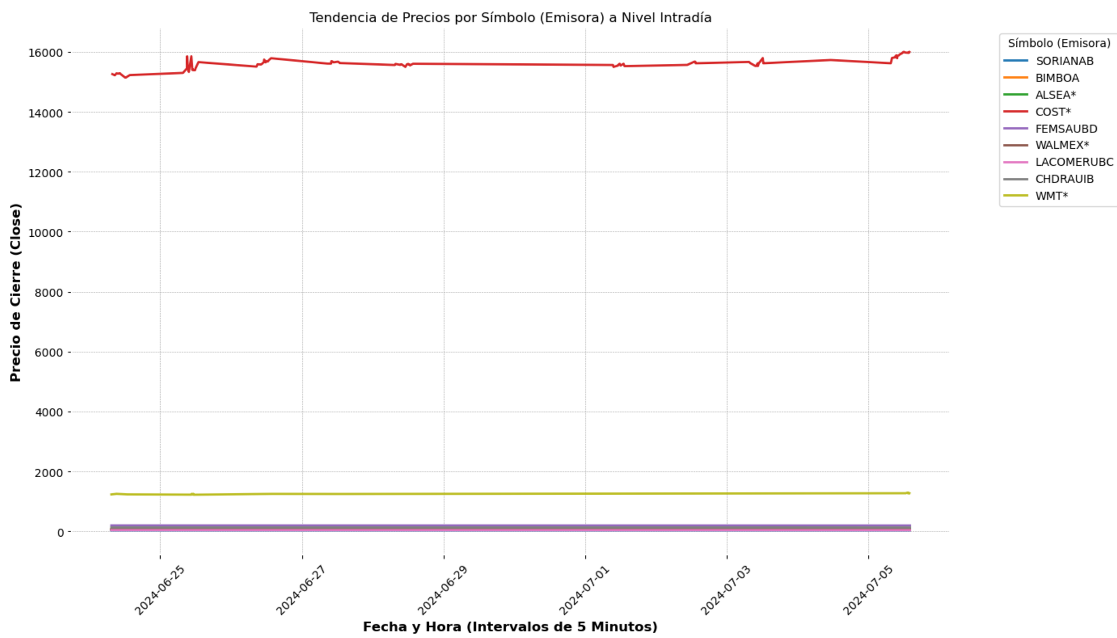
[ ]:

```

```

[76]:

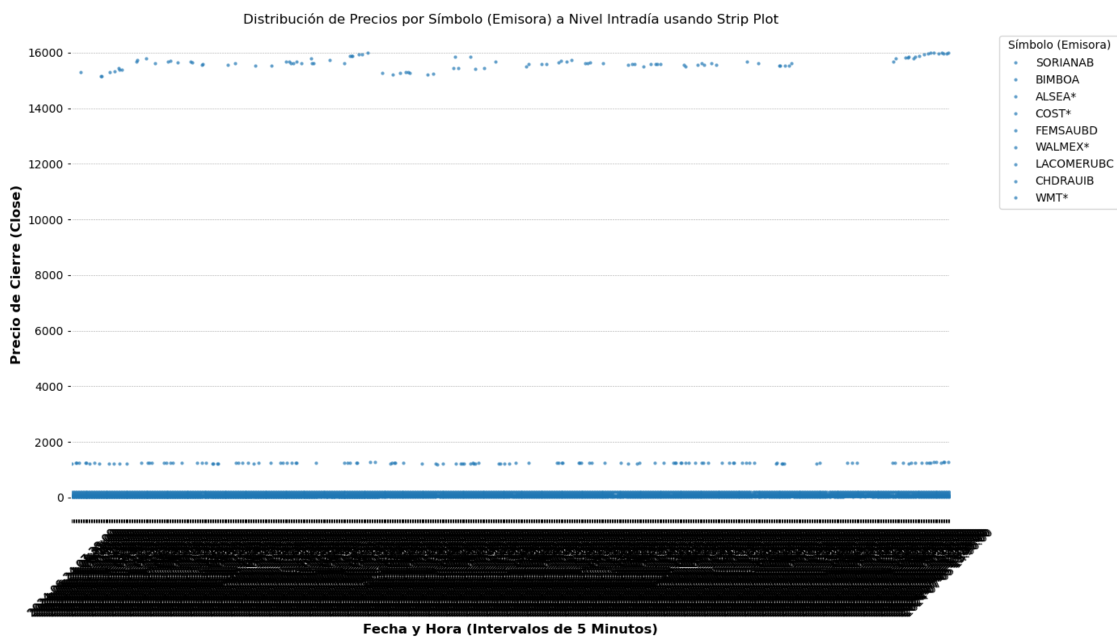
```python
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(14, 8))

# Convertir cada DataFrame de PySpark a Pandas y graficar la tendencia de␣
↪precios
for symbol, df in df_symbols_5min_ohlc.items():
    # Convertir el DataFrame de PySpark a Pandas
    pandas_df = df.select("datetime_5min", "close").orderBy("datetime_5min").
↪toPandas()

    # Graficar la tendencia de precios usando lineplot
    sns.lineplot(data=pandas_df, x="datetime_5min", y="close", label=symbol)
```

```
# Configuración de la gráfica
plt.title("Tendencia de Precios por Símbolo (Emisora) a Nivel Intradía")
plt.xlabel("Fecha y Hora (Intervalos de 5 Minutos)")
plt.ylabel("Precio de Cierre (Close)")
plt.legend(title="Símbolo (Emisora)", bbox_to_anchor=(1.05, 1), loc='upper␣
↪left')
plt.xticks(rotation=45)
plt.tight_layout()

# Mostrar la gráfica
plt.show()
```



[77]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Crear una figura grande para acomodar las gráficas de todos los símbolos
plt.figure(figsize=(14, 8))

# Convertir cada DataFrame de PySpark a Pandas y graficar la tendencia de␣
↪precios
for symbol, df in df_symbols_5min_ohlc.items():
    # Convertir el DataFrame de PySpark a Pandas
    pandas_df = df.select("datetime_5min", "close").orderBy("datetime_5min").
↪toPandas()
    pandas_df['symbol'] = symbol  # Agregar el símbolo como columna para␣
↪etiquetarlo en el gráfico
```

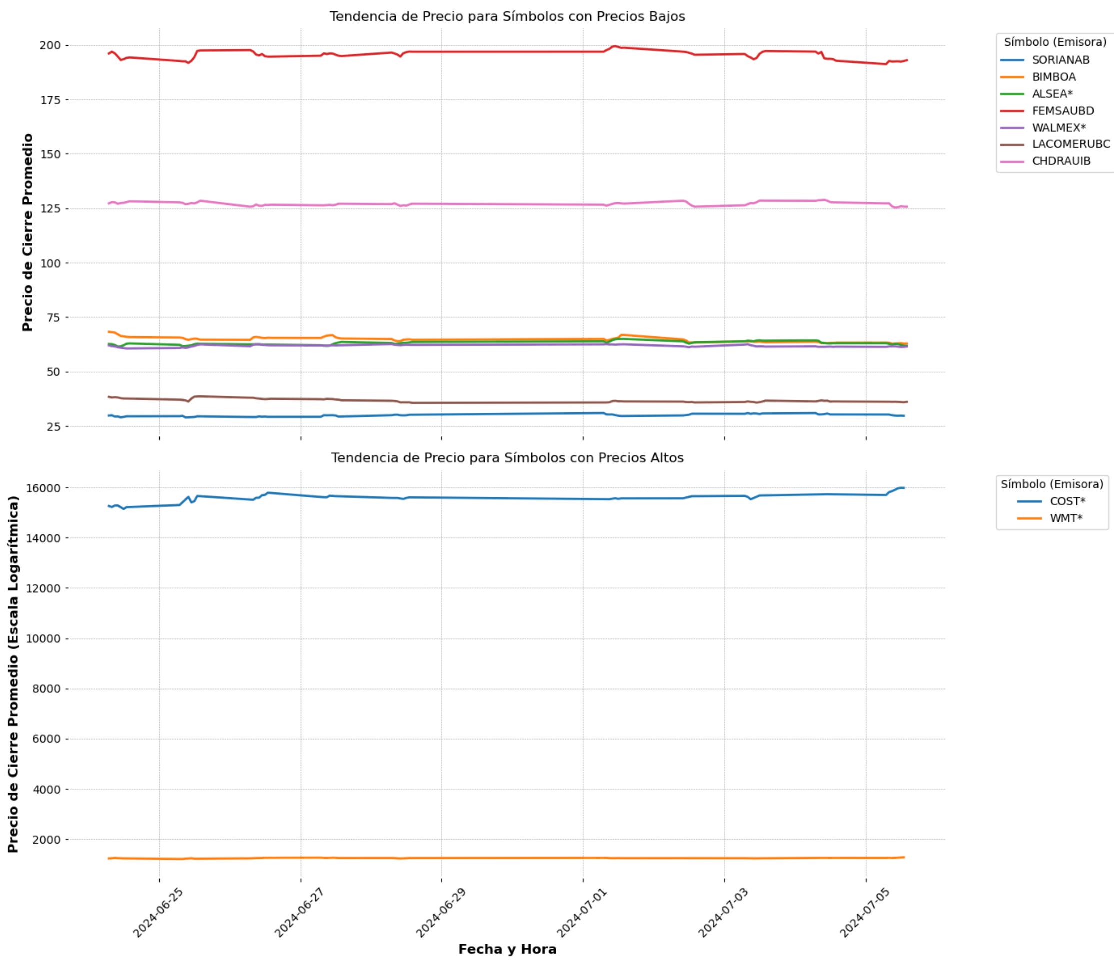
```
    # Usar sns.stripplot para visualizar la distribución de precios por símbolo␣
↪a lo largo del tiempo
    sns.stripplot(data=pandas_df, x="datetime_5min", y="close", hue="symbol",␣
↪jitter=0.3, dodge=True, size=3, alpha=0.7)

# Configuración de la gráfica
plt.title("Distribución de Precios por Símbolo (Emisora) a Nivel Intradía␣
↪usando Strip Plot")
plt.xlabel("Fecha y Hora (Intervalos de 5 Minutos)")
plt.ylabel("Precio de Cierre (Close)")
plt.legend(title="Símbolo (Emisora)", bbox_to_anchor=(1.05, 1), loc='upper␣
↪left')
plt.xticks(rotation=45)
plt.tight_layout()

# Mostrar la gráfica
plt.show()
```



[78]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
from pyspark.sql import functions as F

# Definir límites para separar símbolos de precios altos y bajos
high_price_threshold = 500  # Ajusta este umbral según sea necesario
```

```python
# Convertir cada DataFrame de PySpark a Pandas, reducir datos y agrupar por␣
↪precio
high_price_symbols = []
low_price_symbols = []

for symbol, df in df_symbols_5min_ohlc.items():
    # Agrupar los datos en intervalos de 1 hora para reducir puntos
    df_hourly = df.groupBy(F.window("datetime_5min", "1 hour").
↪alias("hour_window")) \
                  .agg(F.mean("close").alias("average_close")) \
                  .select(F.col("hour_window.start").alias("datetime"),␣
↪"average_close") \
                  .orderBy("datetime")

    # Convertir a Pandas para graficar
    pandas_df = df_hourly.toPandas()
    pandas_df['symbol'] = symbol  # Agregar el símbolo para referencia

    # Clasificar símbolos de acuerdo al precio promedio para graficar en␣
↪diferentes subplots
    if pandas_df['average_close'].mean() > high_price_threshold:
        high_price_symbols.append(pandas_df)
    else:
        low_price_symbols.append(pandas_df)

# Configurar las gráficas con subplots para símbolos de precios bajos y altos
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(14, 12), sharex=True)

# Gráfica para símbolos de precios bajos
for df in low_price_symbols:
    sns.lineplot(data=df, x="datetime", y="average_close", ax=ax1,␣
↪label=df['symbol'].iloc[0])

ax1.set_title("Tendencia de Precio para Símbolos con Precios Bajos")
ax1.set_ylabel("Precio de Cierre Promedio")
ax1.legend(title="Símbolo (Emisora)", bbox_to_anchor=(1.05, 1), loc='upper␣
↪left')

# Gráfica para símbolos de precios altos (escala logarítmica)
for df in high_price_symbols:
    sns.lineplot(data=df, x="datetime", y="average_close", ax=ax2,␣
↪label=df['symbol'].iloc[0])

ax2.set_title("Tendencia de Precio para Símbolos con Precios Altos")
# ax2.set_yscale("log")  # Escala logarítmica para precios altos
```

```
ax2.set_ylabel("Precio de Cierre Promedio (Escala Logarítmica)")
ax2.legend(title="Símbolo (Emisora)", bbox_to_anchor=(1.05, 1), loc='upper␣
↪left')

# Configuración final
plt.xlabel("Fecha y Hora")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



[ ]:

[79]:
```
import pandas as pd
import matplotlib.pyplot as plt
import mplfinance
from mplfinance.original_flavor import candlestick_ohlc
import matplotlib.dates as mdates
```

```python
# Iterar sobre cada símbolo en el diccionario df_symbols_5min
for symbol, df in df_symbols_5min_ohlc.items():
    # Convertir el DataFrame de Spark a Pandas
    pandas_df = df.select("datetime_5min", "open", "high", "low", "close", "volume").orderBy("datetime_5min").toPandas()

    # Convertir la columna de fecha y hora a formato numérico para la gráfica de velas
    pandas_df["datetime_5min"] = pd.to_datetime(pandas_df["datetime_5min"])
    pandas_df["datetime_num"] = pandas_df["datetime_5min"].map(mdates.date2num)

    # Crear el gráfico de velas
    fig, ax = plt.subplots(figsize=(12, 6))
    ax.set_title(f"Gráfico de Velas a 5 Minutos para {symbol}")
    ax.set_xlabel("Fecha y Hora")
    ax.set_ylabel("Precio de Cierre")

    # Formatear la fecha en el eje x
    ax.xaxis_date()
    ax.xaxis.set_major_formatter(mdates.DateFormatter("%Y-%m-%d %H:%M"))

    # Generar los datos OHLC para el gráfico de velas
    ohlc_data = pandas_df[["datetime_num", "open", "high", "low", "close"]].values
    candlestick_ohlc(ax, ohlc_data, width=0.0015, colorup="green", colordown="red")

    # Ajustar los ejes y mostrar el gráfico
    plt.xticks(rotation=45)
    plt.grid()
    plt.tight_layout()
    plt.show()
```

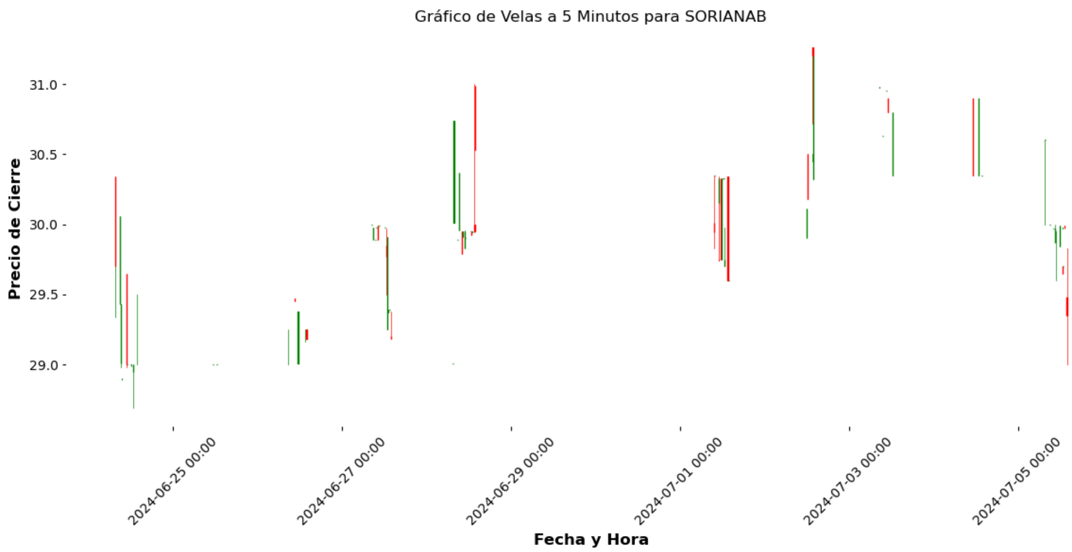


Gráfico de Velas a 5 Minutos para SORIANAB
Precio de Cierre
31.0
30.5
30.0
29.5
29.0
2024-06-25 00:00
2024-06-27 00:00
2024-06-29 00:00
2024-07-01 00:00
2024-07-03 00:00
2024-07-05 00:00
Fecha y Hora

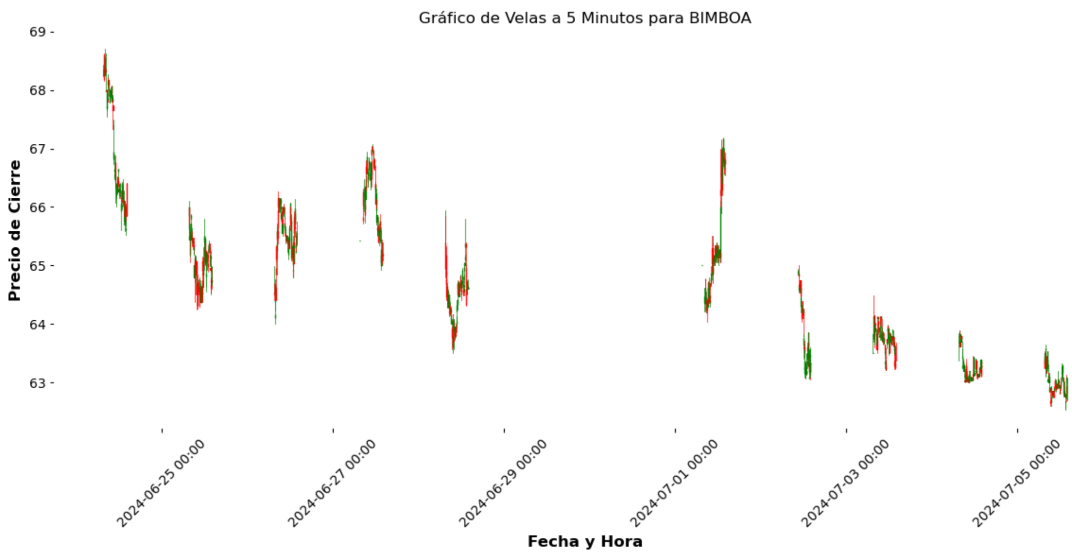


Gráfico de Velas a 5 Minutos para BIMBOA
Precio de Cierre
69
68
67
66
65
64
63
2024-06-25 00:00
2024-06-27 00:00
2024-06-29 00:00
2024-07-01 00:00
2024-07-03 00:00
2024-07-05 00:00
Fecha y Hora

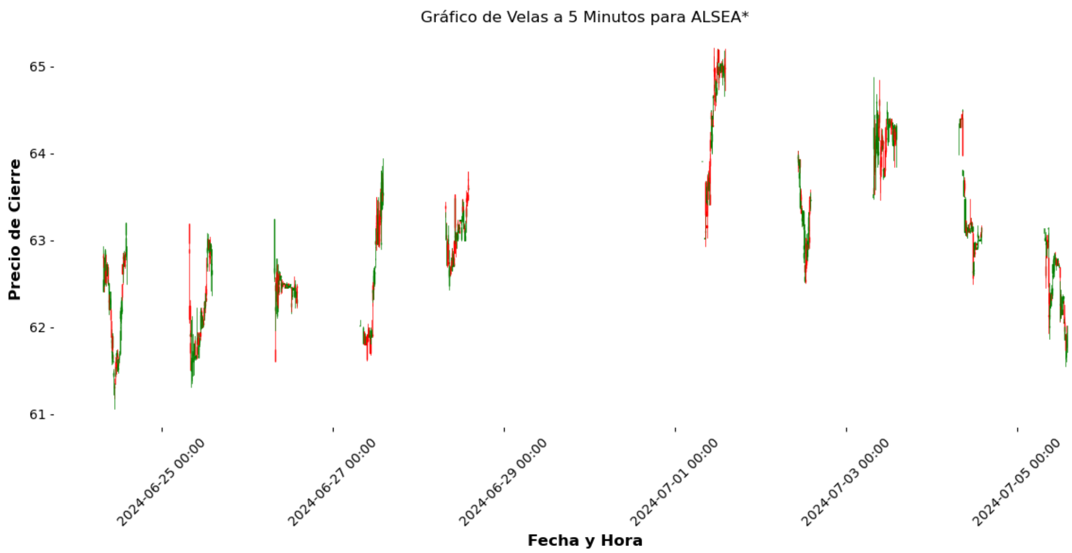


Gráfico de Velas a 5 Minutos para ALSEA*
65
64
63
62
61
Precio de Cierre
2024-06-25 00:00
2024-06-27 00:00
2024-06-29 00:00
2024-07-01 00:00
2024-07-03 00:00
2024-07-05 00:00
Fecha y Hora


Gráfico de Velas a 5 Minutos para COST*
16000
15800
15600
15400
15200
Precio de Cierre
2024-06-25 00:00
2024-06-27 00:00
2024-06-29 00:00
2024-07-01 00:00
2024-07-03 00:00
2024-07-05 00:00
Fecha y Hora

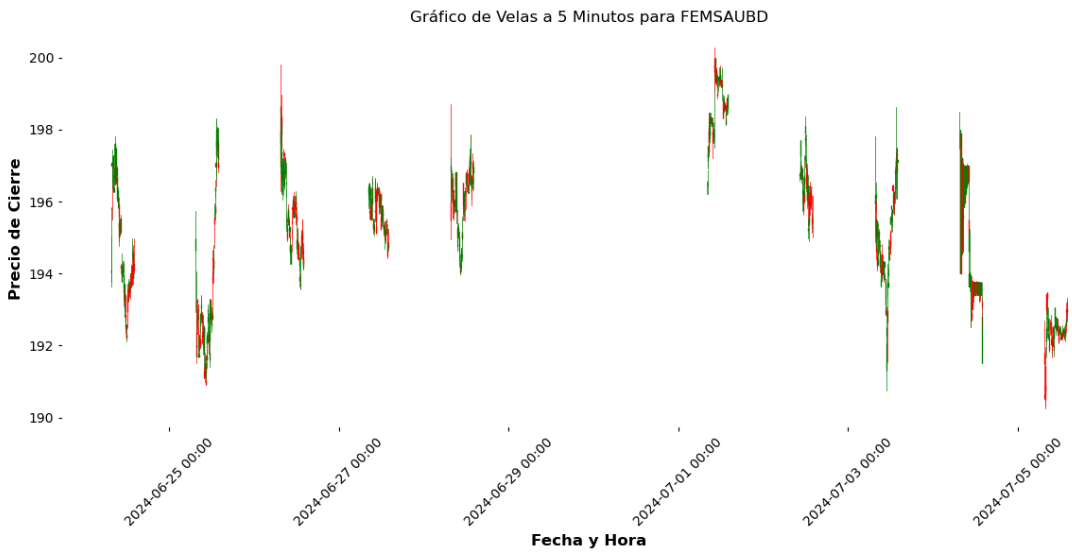


Gráfico de Velas a 5 Minutos para FEMSAUBD
Precio de Cierre
200
198
196
194
192
190
2024-06-25 00:00
2024-06-27 00:00
2024-06-29 00:00
2024-07-01 00:00
2024-07-03 00:00
2024-07-05 00:00
Fecha y Hora

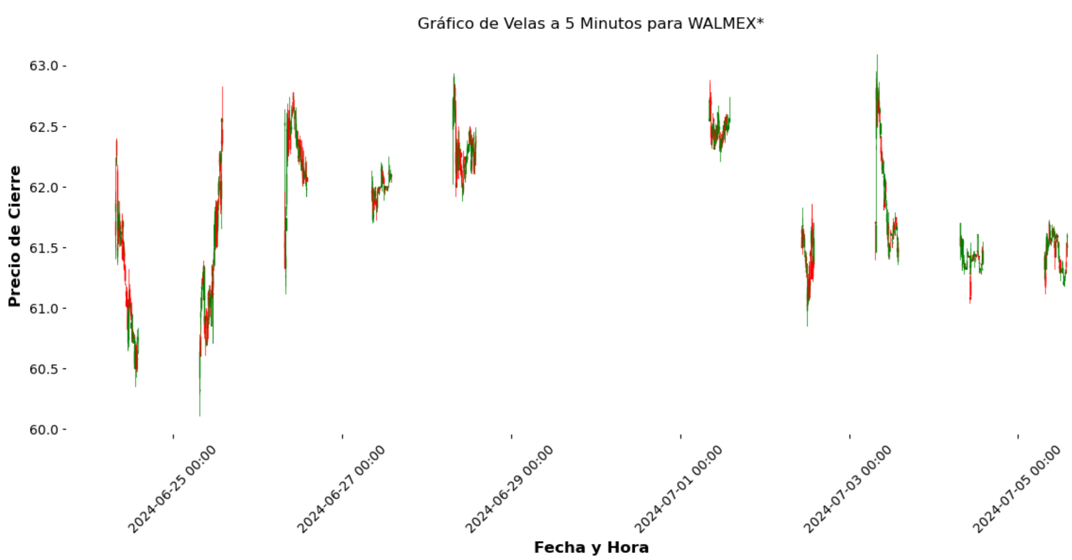


Gráfico de Velas a 5 Minutos para WALMEX*
Precio de Cierre
63.0
62.5
62.0
61.5
61.0
60.5
60.0
2024-06-25 00:00
2024-06-27 00:00
2024-06-29 00:00
2024-07-01 00:00
2024-07-03 00:00
2024-07-05 00:00
Fecha y Hora

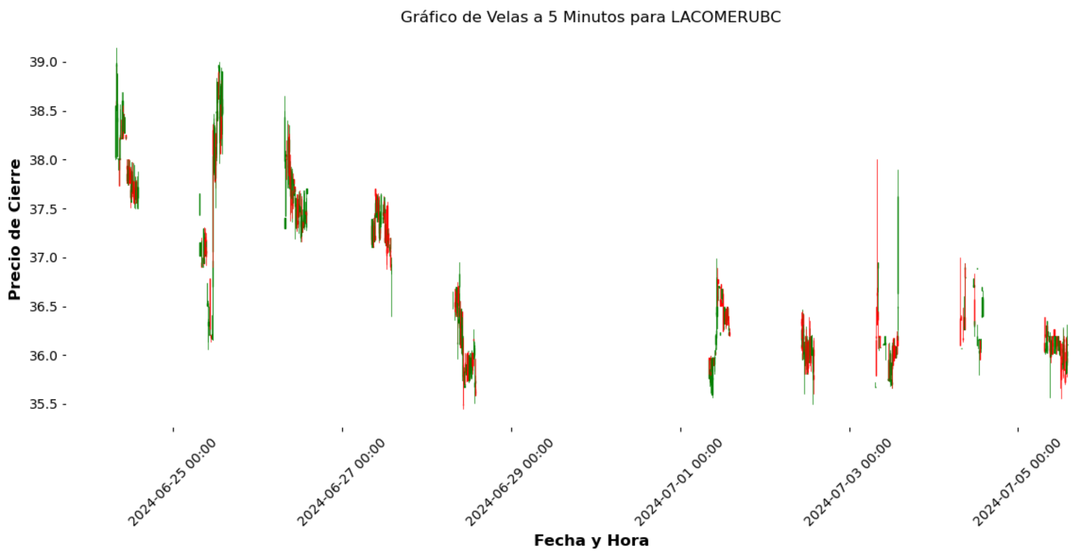


Gráfico de Velas a 5 Minutos para LACOMERUBC
Precio de Cierre
39.0
38.5
38.0
37.5
37.0
36.5
36.0
35.5
2024-06-25 00:00
2024-06-27 00:00
2024-06-29 00:00
2024-07-01 00:00
2024-07-03 00:00
2024-07-05 00:00
Fecha y Hora

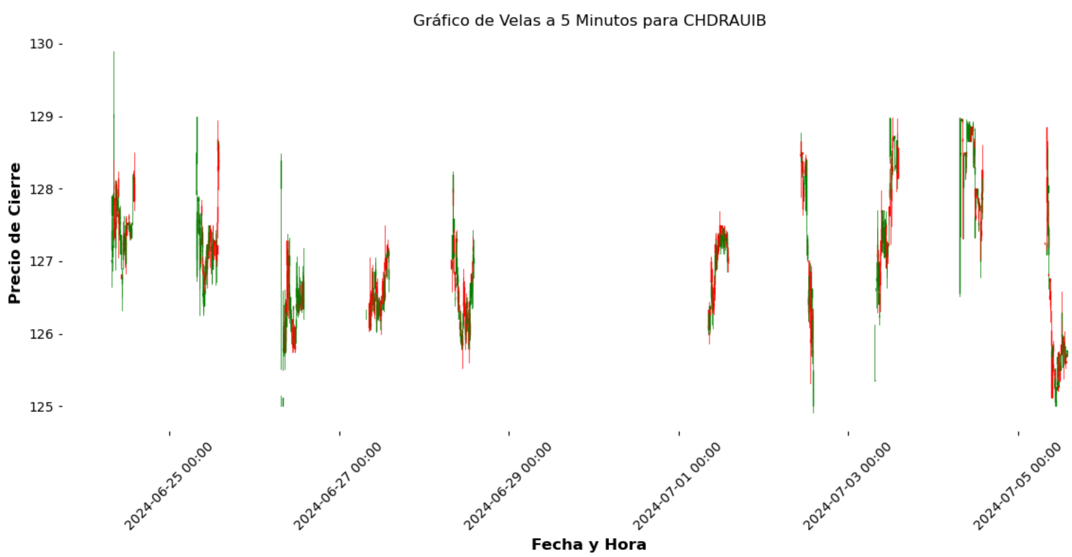
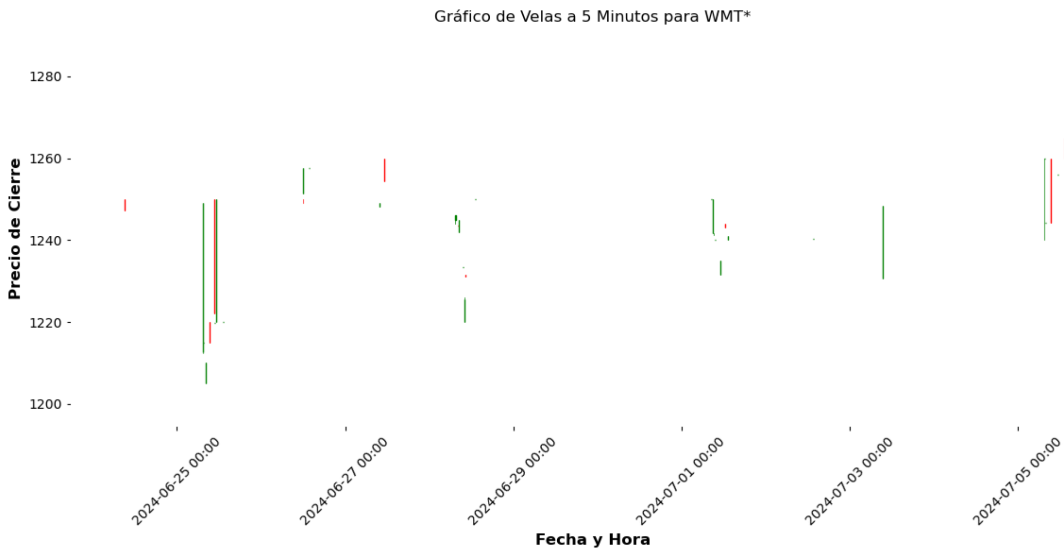
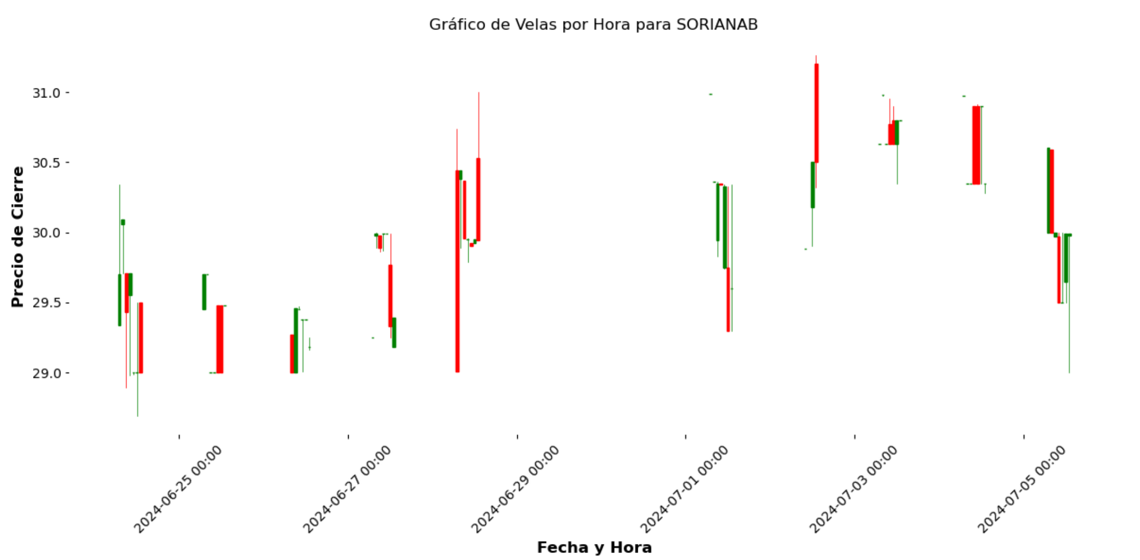


Gráfico de Velas a 5 Minutos para CHDRAUIB
Precio de Cierre
130
129
128
127
126
125
2024-06-25 00:00
2024-06-27 00:00
2024-06-29 00:00
2024-07-01 00:00
2024-07-03 00:00
2024-07-05 00:00
Fecha y Hora



[80]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import mplfinance
from mplfinance.original_flavor import candlestick_ohlc
import matplotlib.dates as mdates
from pyspark.sql import functions as F

# Iterar sobre cada símbolo en el diccionario df_symbols_5min
for symbol, df in df_symbols_5min_ohlc.items():
    # Agrupar por hora y calcular los valores OHLC y volumen
    df_hourly = df.groupBy(F.window("datetime_5min", "1 hour").
 ↪alias("hour_window")) \
                  .agg(
                      F.first("close").alias("open"),
                      F.max("high").alias("high"),
                      F.min("low").alias("low"),
                      F.last("close").alias("close"),
                      F.sum("volume").alias("volume")
                  ) \
                  .select(
                      F.col("hour_window.start").alias("datetime"),
                      "open", "high", "low", "close", "volume"
                  ) \
                  .orderBy("datetime")

    # Convertir a DataFrame de Pandas
    pandas_df = df_hourly.toPandas()
```

```
# Convertir la columna de fecha y hora a formato numérico para la gráfica␣
↪de velas
pandas_df["datetime"] = pd.to_datetime(pandas_df["datetime"])
pandas_df["datetime"] = pandas_df["datetime"].map(mdates.date2num)

# Crear el gráfico de velas
fig, ax = plt.subplots(figsize=(12, 6))
ax.set_title(f"Gráfico de Velas por Hora para {symbol}")
ax.set_xlabel("Fecha y Hora")
ax.set_ylabel("Precio de Cierre")

# Formatear la fecha en el eje x
ax.xaxis_date()
ax.xaxis.set_major_formatter(mdates.DateFormatter("%Y-%m-%d %H:%M"))

# Generar los datos OHLC para el gráfico de velas
ohlc_data = pandas_df[["datetime", "open", "high", "low", "close"]].values
candlestick_ohlc(ax, ohlc_data, width=0.03, colorup="green",␣
↪colordown="red")

# Ajustar los ejes y mostrar el gráfico
plt.xticks(rotation=45)
plt.grid()
plt.tight_layout()
plt.show()
```



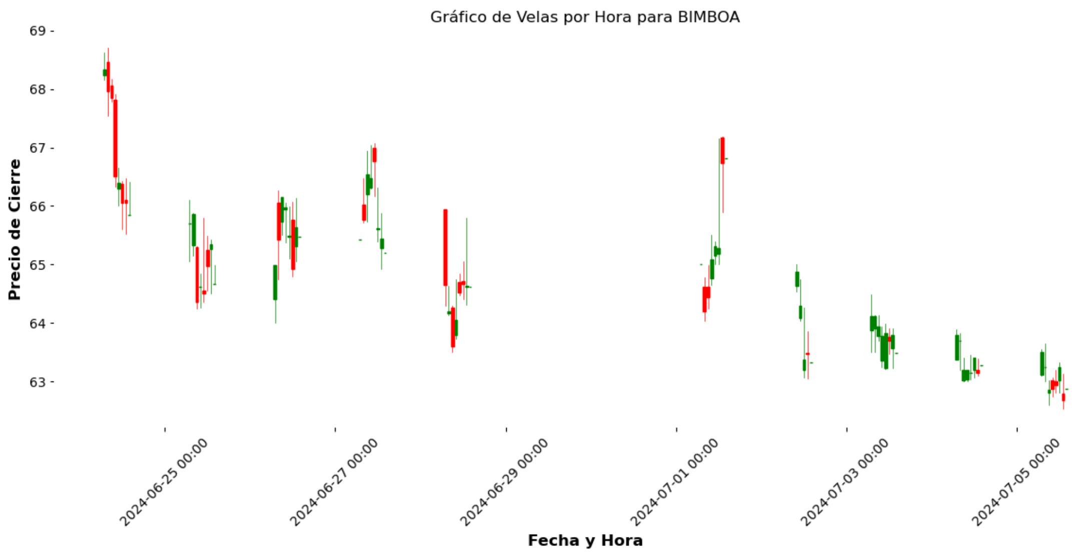


Gráfico de Velas por Hora para BIMBOA
69
68
67
66
65
64
63
Precio de Cierre
2024-06-25 00:00
2024-06-27 00:00
2024-06-29 00:00
2024-07-01 00:00
2024-07-03 00:00
2024-07-05 00:00
Fecha y Hora

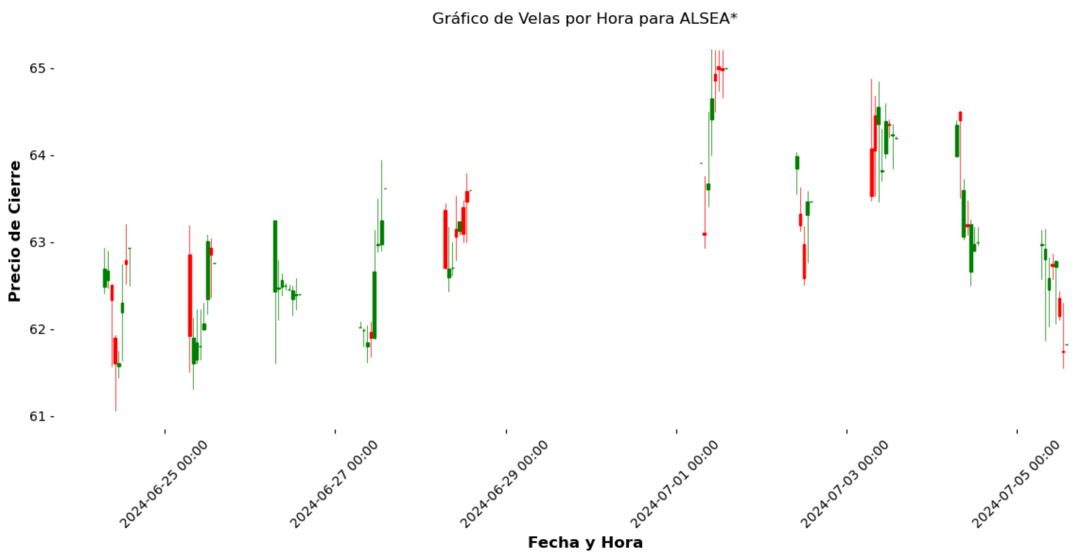


Gráfico de Velas por Hora para ALSEA*
65
64
63
62
61
Precio de Cierre
2024-06-25 00:00
2024-06-27 00:00
2024-06-29 00:00
2024-07-01 00:00
2024-07-03 00:00
2024-07-05 00:00
Fecha y Hora

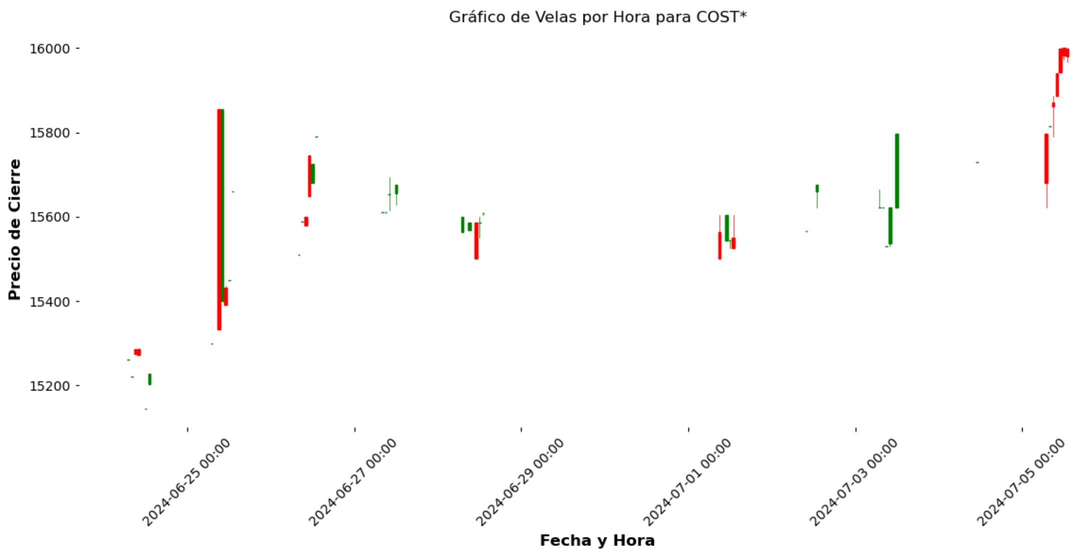


Gráfico de Velas por Hora para COST*
16000
15800
15600
15400
15200
Precio de Cierre
2024-06-25 00:00
2024-06-27 00:00
2024-06-29 00:00
2024-07-01 00:00
2024-07-03 00:00
2024-07-05 00:00
Fecha y Hora

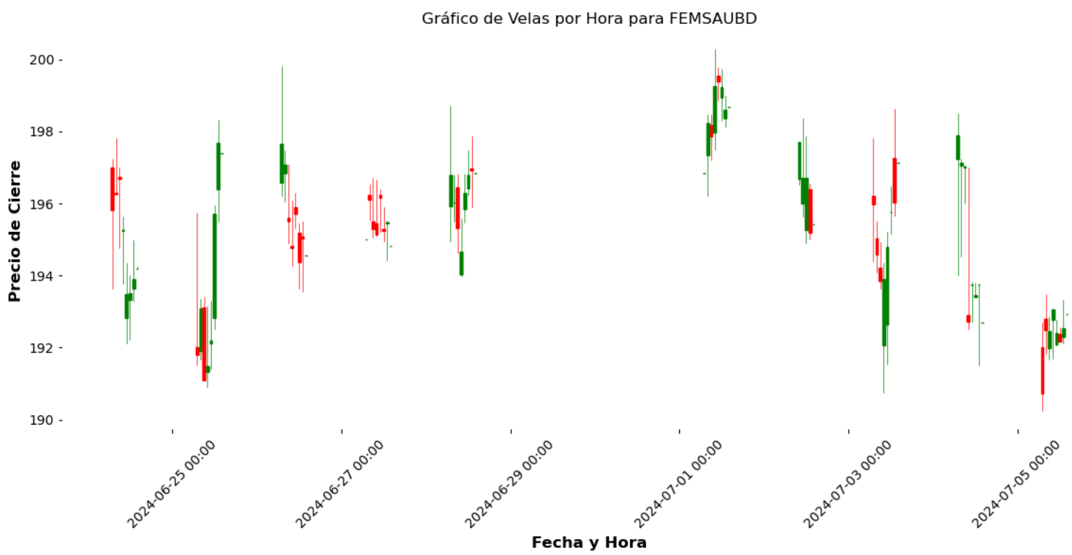


Gráfico de Velas por Hora para FEMSAUBD
200
198
196
194
192
190
Precio de Cierre
2024-06-25 00:00
2024-06-27 00:00
2024-06-29 00:00
2024-07-01 00:00
2024-07-03 00:00
2024-07-05 00:00
Fecha y Hora

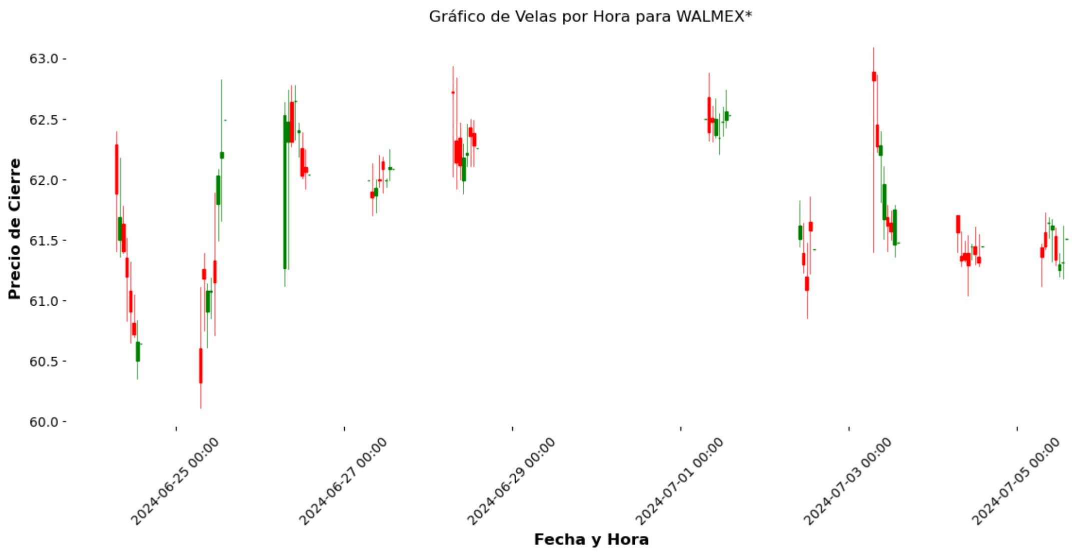


Gráfico de Velas por Hora para WALMEX*
Precio de Cierre
63.0
62.5
62.0
61.5
61.0
60.5
60.0
2024-06-25 00:00
2024-06-27 00:00
2024-06-29 00:00
2024-07-01 00:00
2024-07-03 00:00
2024-07-05 00:00
Fecha y Hora

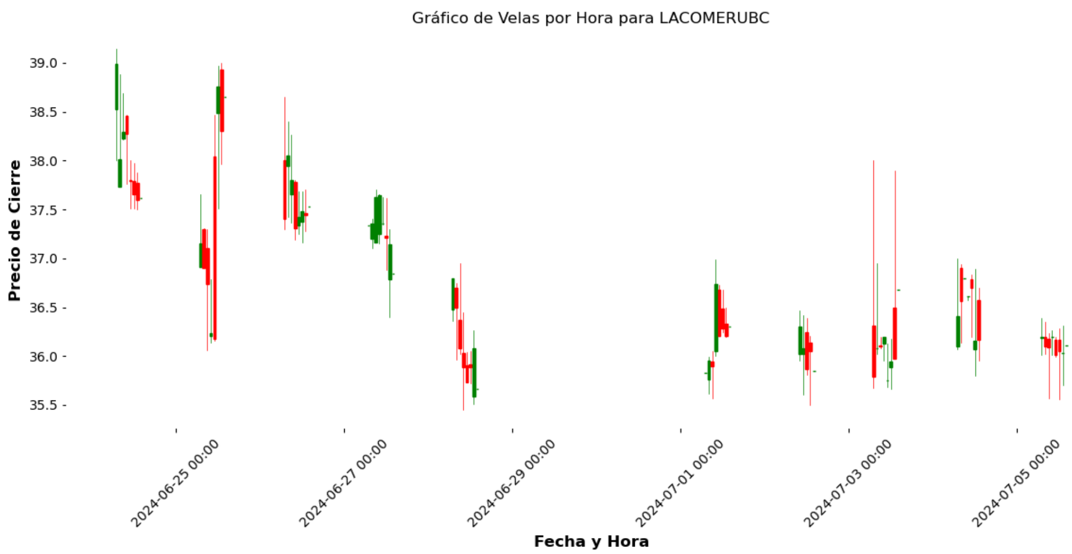
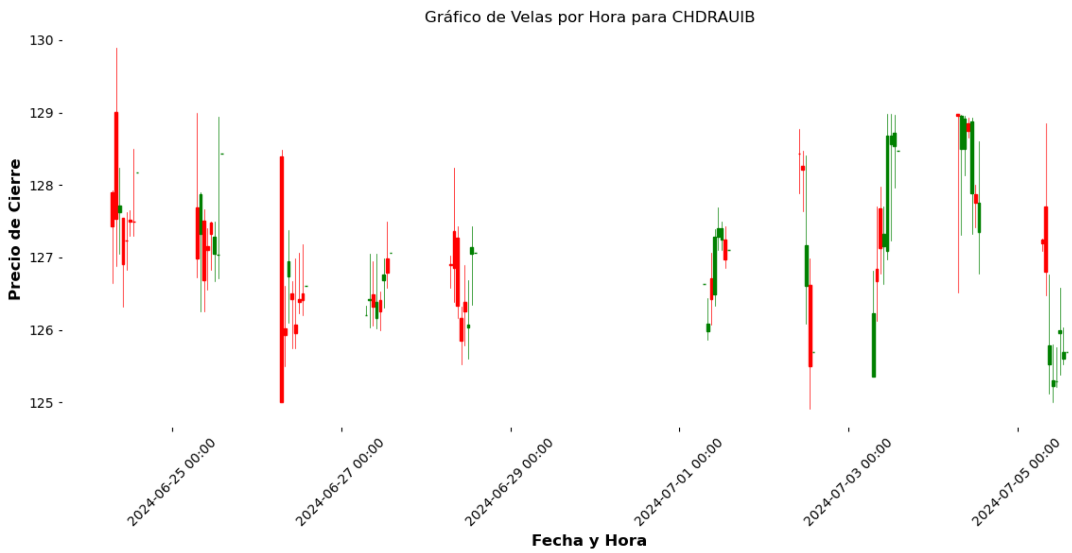
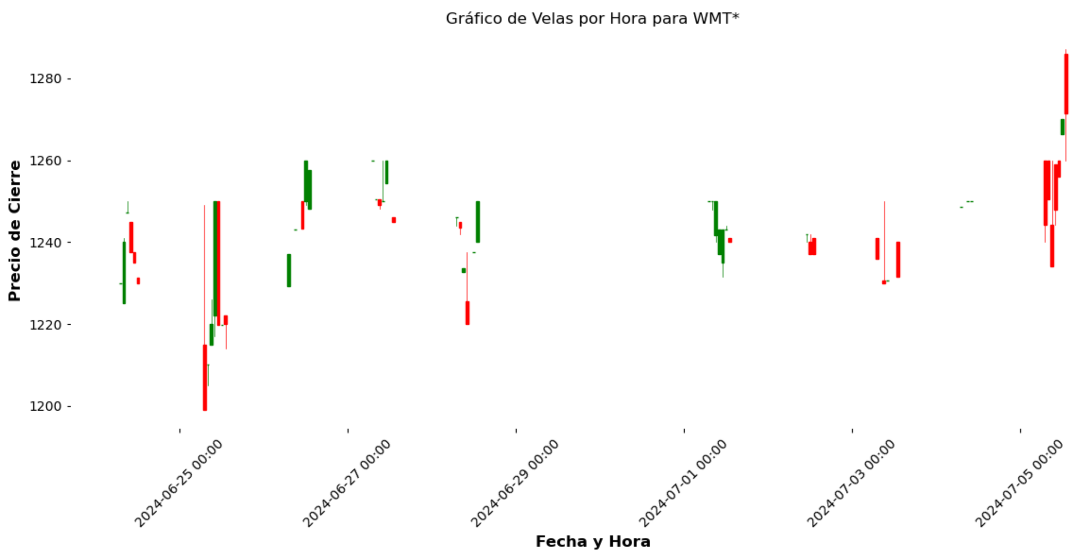


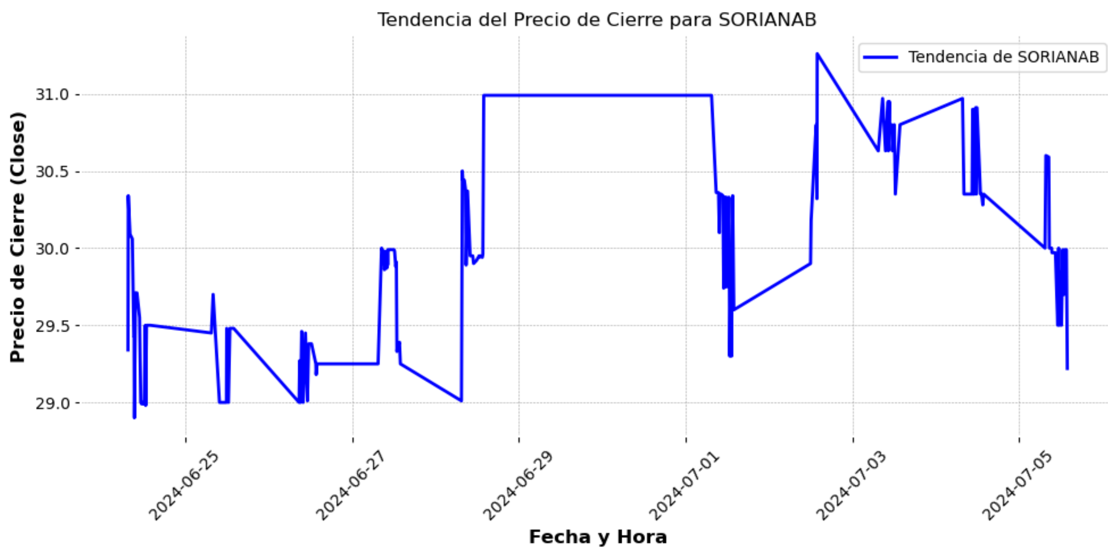
Gráfico de Velas por Hora para LACOMERUBC
Precio de Cierre
39.0
38.5
38.0
37.5
37.0
36.5
36.0
35.5
2024-06-25 00:00
2024-06-27 00:00
2024-06-29 00:00
2024-07-01 00:00
2024-07-03 00:00
2024-07-05 00:00
Fecha y Hora

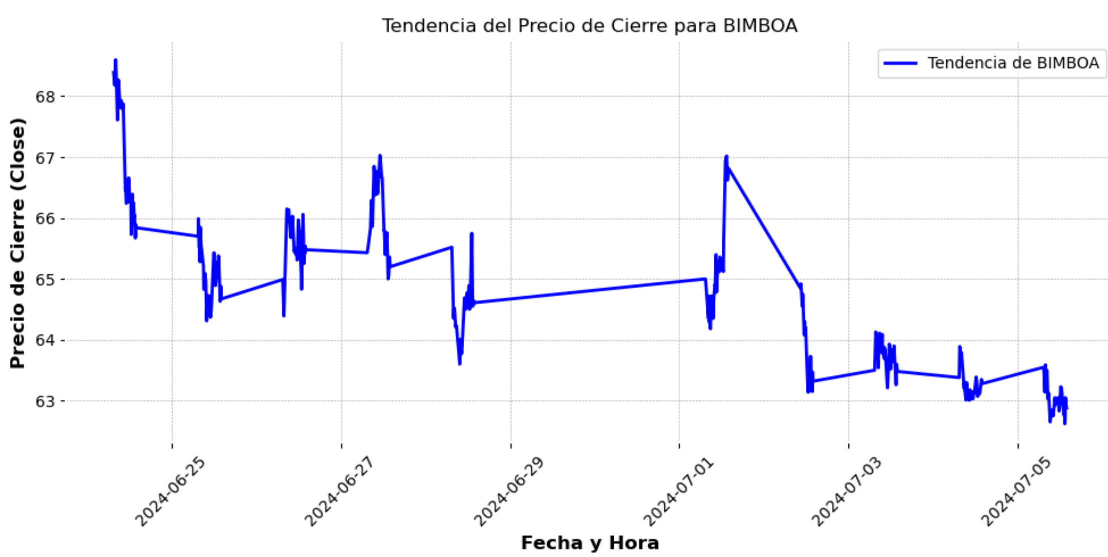


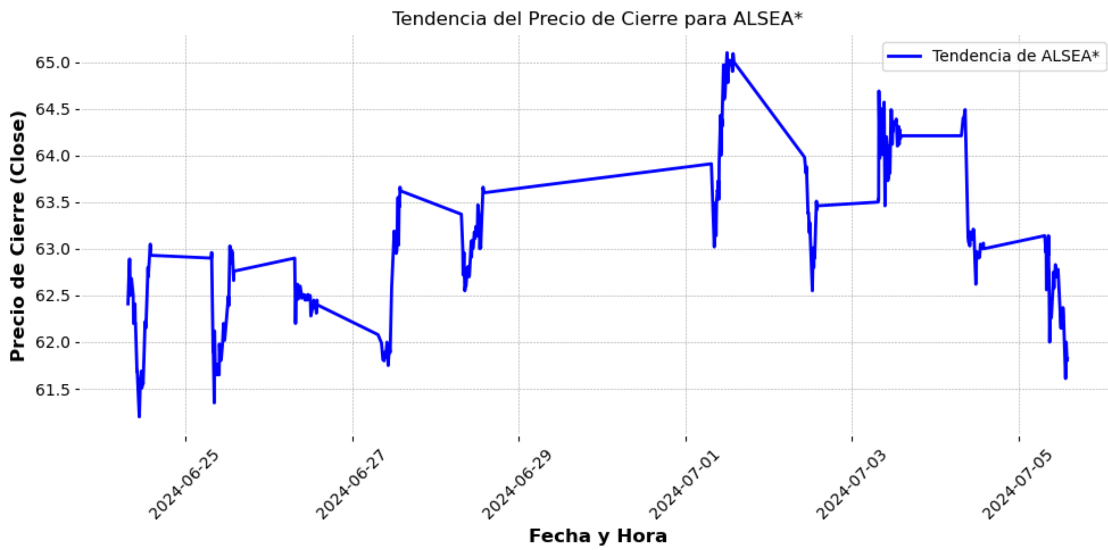


[81]:
```python
import matplotlib.pyplot as plt
from pyspark.sql import functions as F

# Convertir cada DataFrame de PySpark a Pandas y graficar la tendencia de␣
↪precios
for symbol, df in df_symbols_5min_ohlc.items():
    # Convertir el DataFrame de PySpark a Pandas
    pandas_df = df.select("datetime_5min", "close").orderBy("datetime_5min").
↪toPandas()
```

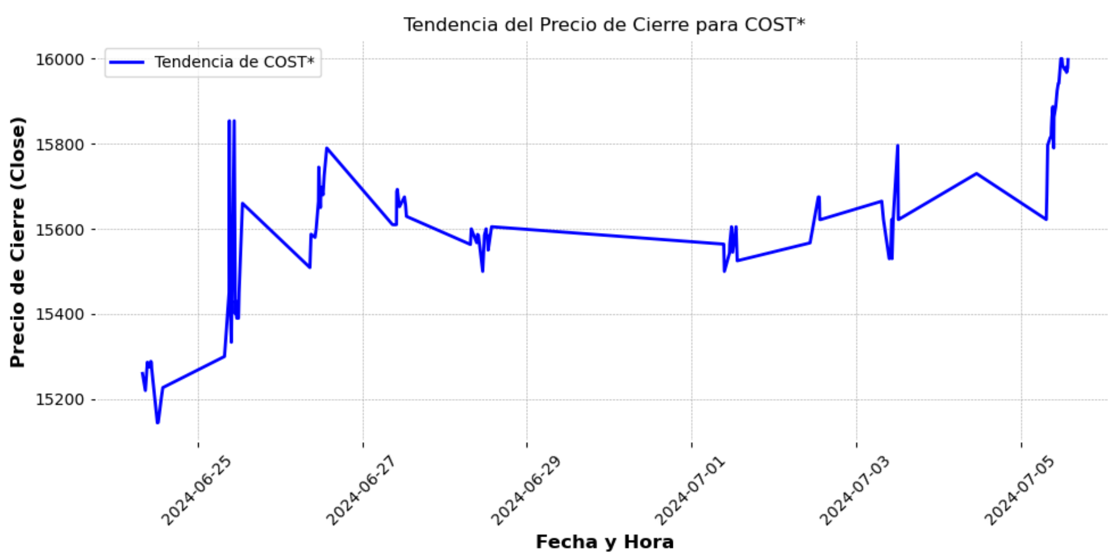
```
    # Crear una gráfica de línea para la tendencia del precio
    plt.figure(figsize=(10, 5))
    plt.plot(pandas_df["datetime_5min"], pandas_df["close"], label=f"Tendencia␣
↪de {symbol}", color="blue")
    plt.title(f"Tendencia del Precio de Cierre para {symbol}")
    plt.xlabel("Fecha y Hora")
    plt.ylabel("Precio de Cierre (Close)")
    plt.xticks(rotation=45)
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```



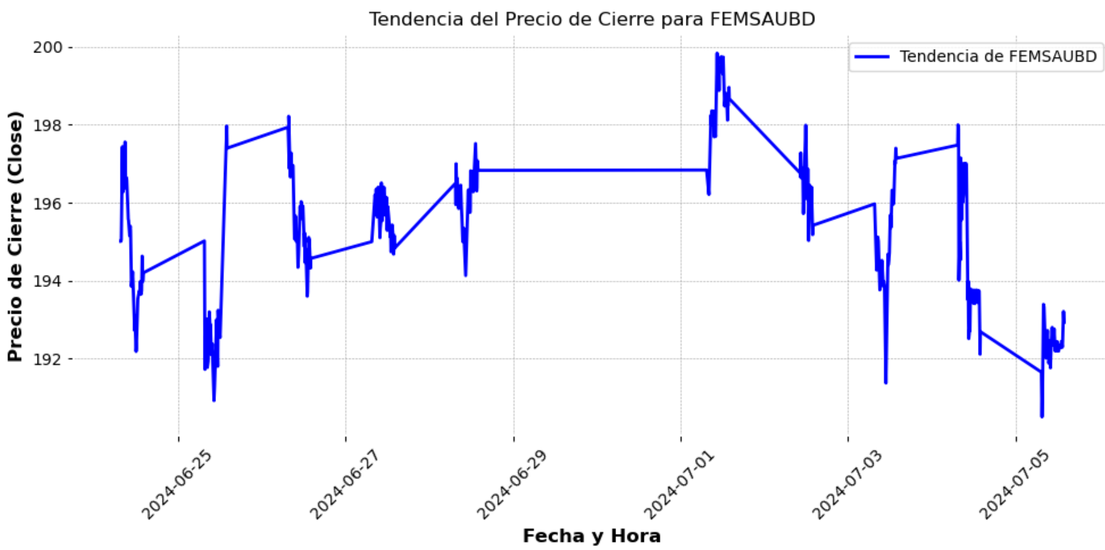


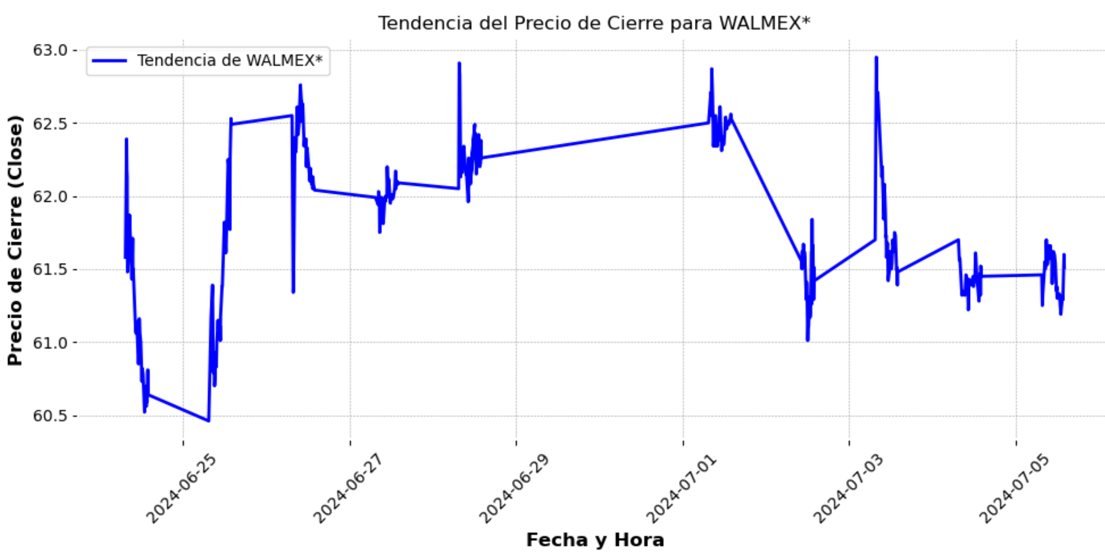


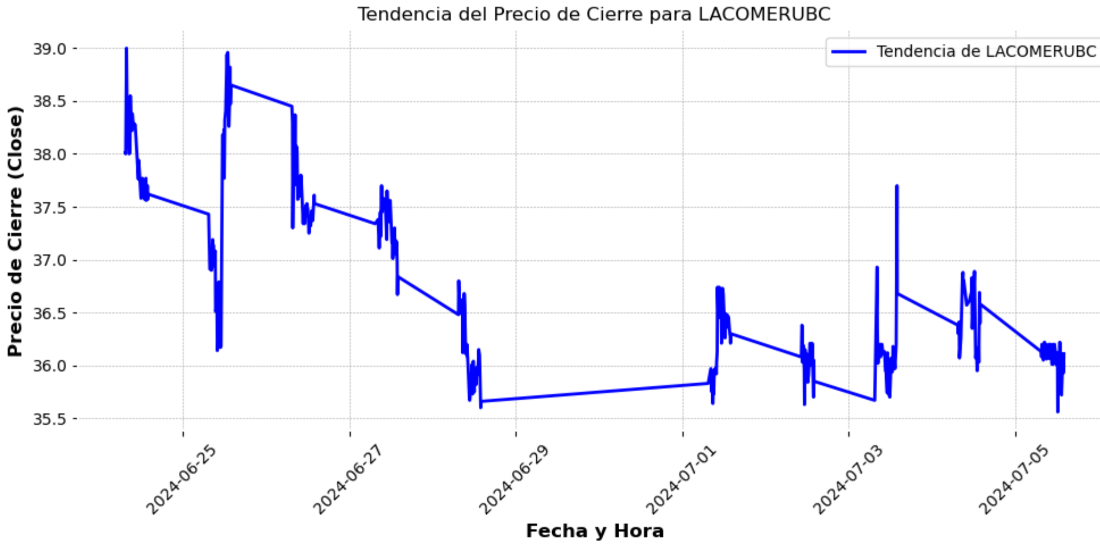
Tendencia del Precio de Cierre para ALSEA*
Tendencia de ALSEA*
Precio de Cierre (Close)
65.0
64.5
64.0
63.5
63.0
62.5
62.0
61.5
2024-06-25
2024-06-27
2024-06-29
2024-07-01
2024-07-03
2024-07-05
Fecha y Hora



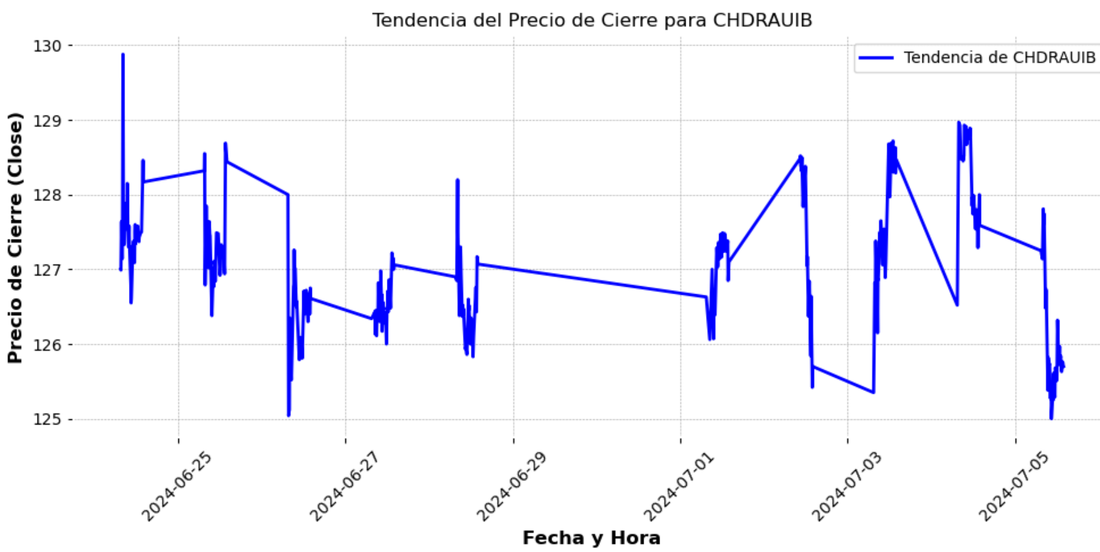
Tendencia del Precio de Cierre para COST*
Tendencia de COST*
Precio de Cierre (Close)
16000
15800
15600
15400
15200
2024-06-25
2024-06-27
2024-06-29
2024-07-01
2024-07-03
2024-07-05
Fecha y Hora


Tendencia del Precio de Cierre para FEMSAUBD
Tendencia de FEMSAUBD
Precio de Cierre (Close)
200
198
196
194
192
2024-06-25
2024-06-27
2024-06-29
2024-07-01
2024-07-03
2024-07-05
Fecha y Hora

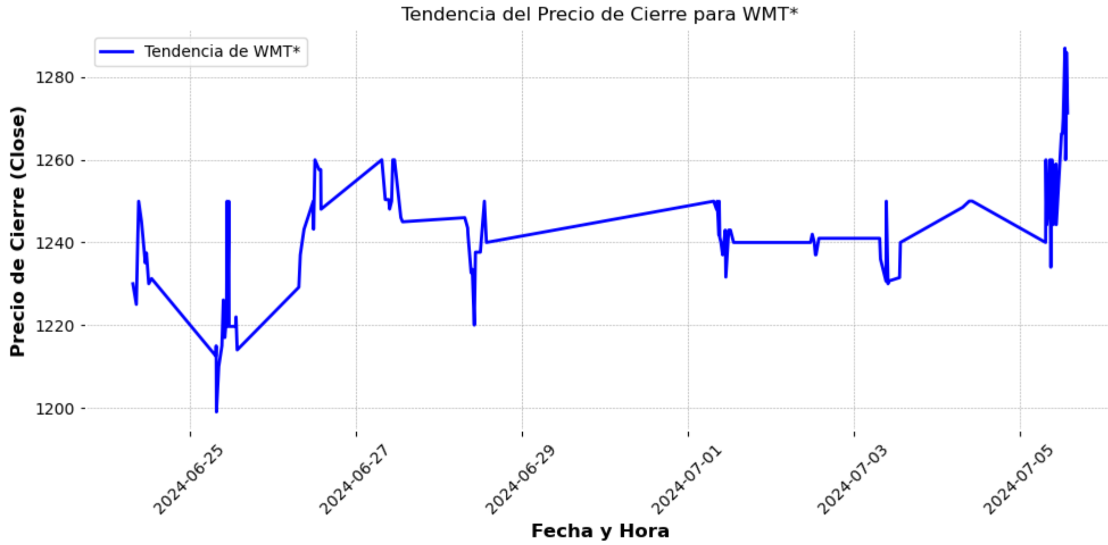

Tendencia del Precio de Cierre para WALMEX*
Tendencia de WALMEX*
Precio de Cierre (Close)
63.0
62.5
62.0
61.5
61.0
60.5
2024-06-25
2024-06-27
2024-06-29
2024-07-01
2024-07-03
2024-07-05
Fecha y Hora



Tendencia del Precio de Cierre para LACOMERUBC
Tendencia de LACOMERUBC
Precio de Cierre (Close)
Fecha y Hora
39.0
38.5
38.0
37.5
37.0
36.5
36.0
35.5
2024-06-25
2024-06-27
2024-06-29
2024-07-01
2024-07-03
2024-07-05



Tendencia del Precio de Cierre para CHDRAUIB
Tendencia de CHDRAUIB
Precio de Cierre (Close)
Fecha y Hora
130
129
128
127
126
125
2024-06-25
2024-06-27
2024-06-29
2024-07-01
2024-07-03
2024-07-05



[ ]:

```

```

[ ]:

```

```

[82]:

```
df_symbols_5min
```

[82]:

```
{'SORIANAB': DataFrame[datetime_5min: timestamp, close: double, volume: bigint,
minute_start: timestamp, minute_end: timestamp, lagged_close: double,
lagged_volume: bigint],
 'BIMBOA': DataFrame[datetime_5min: timestamp, close: double, volume: bigint,
minute_start: timestamp, minute_end: timestamp, lagged_close: double,
lagged_volume: bigint],
 'ALSEA*': DataFrame[datetime_5min: timestamp, close: double, volume: bigint,
minute_start: timestamp, minute_end: timestamp, lagged_close: double,
lagged_volume: bigint],
 'COST*': DataFrame[datetime_5min: timestamp, close: double, volume: bigint,
minute_start: timestamp, minute_end: timestamp, lagged_close: double,
lagged_volume: bigint],
 'FEMSAUBD': DataFrame[datetime_5min: timestamp, close: double, volume: bigint,
minute_start: timestamp, minute_end: timestamp, lagged_close: double,
lagged_volume: bigint],
 'WALMEX*': DataFrame[datetime_5min: timestamp, close: double, volume: bigint,
minute_start: timestamp, minute_end: timestamp, lagged_close: double,
lagged_volume: bigint],
 'LACOMERUBC': DataFrame[datetime_5min: timestamp, close: double, volume:
bigint, minute_start: timestamp, minute_end: timestamp, lagged_close: double,
lagged_volume: bigint],
 'CHDRAUIB': DataFrame[datetime_5min: timestamp, close: double, volume: bigint,
```

```
minute_start: timestamp, minute_end: timestamp, lagged_close: double,
lagged_volume: bigint],
 'WMT*': DataFrame[datetime_5min: timestamp, close: double, volume: bigint,
minute_start: timestamp, minute_end: timestamp, lagged_close: double,
lagged_volume: bigint]}
```

[ ]:

```
[83]: import pandas as pd
from datetime import datetime, timedelta
from pyspark.sql import functions as F
from pyspark.sql import Window

# Crear un DataFrame base con intervalos de 5 minutos, desde las 7:30 am hasta␣
↪las 2:30 pm
start_time = "07:30:00"
end_time = "14:30:00"
trade_dates = df_symbols_5min["WALMEX*"].select("minute_start").rdd.map(lambda␣
↪x: x[0].date()).distinct().collect()

# Generar el rango de tiempos usando pandas para cada fecha en `trade_dates`
base_times = []
for trade_date in trade_dates:
    start_dt = datetime.strptime(f"{trade_date} {start_time}", "%Y-%m-%d %H:%M:
↪%S")
    end_dt = datetime.strptime(f"{trade_date} {end_time}", "%Y-%m-%d %H:%M:%S")
    current_time = start_dt
    while current_time <= end_dt:
        base_times.append(current_time)
        current_time += timedelta(minutes=5)

# Crear un DataFrame de pandas y convertirlo a PySpark
base_df_pd = pd.DataFrame(base_times, columns=["minute_start"])
base_df = spark.createDataFrame(base_df_pd)

# Procesar el DataFrame de WALMEX (sin el carácter especial *)
df_walmex = df_symbols_5min["WALMEX*"].withColumnRenamed("close",␣
↪"close_WALMEX") \
                                      .withColumnRenamed("lagged_close",␣
↪"lagged_close_WALMEX") \
                                      .withColumnRenamed("minute_start",␣
↪"minute_start") \
                                      .withColumnRenamed("minute_end",␣
↪"minute_end")

# Unir el DataFrame de WALMEX con el DataFrame base
```

```
combined_df = base_df.join(df_walmex, on="minute_start", how="left")
```

[84]:
```
# Mostrar los primeros registros para verificar
combined_df.orderBy("minute_start").show(5)
```

```
+-------------------+-------------------+------------+------+-------------------
+-------------------+-------------+
|       minute_start|      datetime_5min|close_WALMEX|volume|
minute_end|lagged_close_WALMEX|lagged_volume|
+-------------------+-------------------+------------+------+-------------------
+-------------------+-------------+
|2024-06-24 07:30:00|               NULL|        NULL|  NULL|
NULL|               NULL|         NULL|
|2024-06-24 07:35:00|2024-06-24 07:35:00|       61.87|165811|2024-06-24
07:40:00|               61.6|        80481|
|2024-06-24 07:40:00|2024-06-24 07:40:00|       61.88| 72170|2024-06-24
07:45:00|              61.87|       165811|
|2024-06-24 07:45:00|2024-06-24 07:45:00|       62.29| 25755|2024-06-24
07:50:00|              61.88|        72170|
|2024-06-24 07:50:00|2024-06-24 07:50:00|       62.26| 31307|2024-06-24
07:55:00|              62.29|        25755|
+-------------------+-------------------+------------+------+-------------------
+-------------------+-------------+
only showing top 5 rows
```

[ ]:
```
```

[85]:
```
import pandas as pd
from datetime import datetime, timedelta
from pyspark.sql import functions as F

# Crear un DataFrame base con intervalos de 5 minutos, desde las 7:30 am hasta␣
↪las 2:30 pm
start_time = "07:30:00"
end_time = "14:30:00"
trade_dates = df_symbols_5min["WALMEX*"].select("minute_start").rdd.map(lambda␣
↪x: x[0].date()).distinct().collect()

# Generar el rango de tiempos usando pandas para cada fecha en `trade_dates`
base_times = []
for trade_date in trade_dates:
    start_dt = datetime.strptime(f"{trade_date} {start_time}", "%Y-%m-%d %H:%M:
↪%S")
    end_dt = datetime.strptime(f"{trade_date} {end_time}", "%Y-%m-%d %H:%M:%S")
    current_time = start_dt
    while current_time <= end_dt:
```

```
        base_times.append(current_time)
        current_time += timedelta(minutes=5)

# Crear un DataFrame de pandas y convertirlo a PySpark
base_df_pd = pd.DataFrame(base_times, columns=["minute_start"])
base_df = spark.createDataFrame(base_df_pd)

# Procesar el DataFrame de WALMEX y unirlo a la base
df_walmex = df_symbols_5min["WALMEX*"].withColumnRenamed("close",␣
↪"close_WALMEX") \
                                      .withColumnRenamed("lagged_close",␣
↪"lagged_close_WALMEX")
                                      # .withColumnRenamed("volume",␣
↪"volume_WALMEX") \
                                      # .withColumnRenamed("lagged_volume",␣
↪"lagged_volume_WALMEX")

# Unir el DataFrame de WALMEX con el DataFrame base
combined_df = base_df.join(df_walmex, on="minute_start", how="left")

# Iterar sobre los otros símbolos y unirlos al DataFrame combinado
for symbol, df_symbol in df_symbols_5min.items():
    if symbol == "WALMEX*":  # Saltar WALMEX ya que ya se ha agregado
        continue

    # Limpiar y renombrar las columnas del símbolo actual
    clean_symbol = symbol.replace("*", "")  # Remover caracteres especiales del␣
↪nombre de la columna
    # df_symbol = df_symbol.withColumnRenamed("close", f"close_{clean_symbol}")␣
↪\
    #                     .withColumnRenamed("volume",␣
↪f"volume_{clean_symbol}") \
    #                     .withColumnRenamed("lagged_close",␣
↪f"lagged_close_{clean_symbol}") \
    #                     .withColumnRenamed("lagged_volume",␣
↪f"lagged_volume_{clean_symbol}")

    df_symbol = df_symbol.withColumnRenamed("lagged_close",␣
↪f"lagged_close_{clean_symbol}")

    # Eliminar la columna `minute_end` antes de la unión, si no es el primer␣
↪símbolo
    df_symbol = df_symbol.drop("minute_end")
    df_symbol = df_symbol.drop("close")
    df_symbol = df_symbol.drop("volume")
    df_symbol = df_symbol.drop("lagged_volume")
```

```
    df_symbol = df_symbol.drop("datetime_5min")

    # Unir el DataFrame del símbolo actual con el DataFrame combinado
    combined_df = combined_df.join(df_symbol, on="minute_start", how="left")

# Filtrar filas donde "close_WALMEX" es NULL o vacío
combined_df_filtered = combined_df.filter(F.col("close_WALMEX").isNotNull())

# Mostrar los primeros registros para verificar
combined_df_filtered.orderBy("minute_start").show(5)
```

```
+-------------------+-------------------+------------+------+-------------------
+-------------------+-------------+---------------------+-------------------+---
---------------+-----------------+----------------------+-----------------------
+---------------------+----------------+
|       minute_start|      datetime_5min|close_WALMEX|volume|         minute_end
|lagged_close_WALMEX|lagged_volume|lagged_close_SORIANAB|lagged_close_BIMBOA|lag
ged_close_ALSEA|lagged_close_COST|lagged_close_FEMSAUBD|lagged_close_LACOMERUBC|
lagged_close_CHDRAUIB|lagged_close_WMT|
+-------------------+-------------------+------------+------+-------------------
+-------------------+-------------+---------------------+-------------------+---
---------------+-----------------+----------------------+-----------------------
+---------------------+----------------+
|2024-06-24 07:35:00|2024-06-24 07:35:00|       61.87|165811|2024-06-24
07:40:00|               61.6|        80481|                29.34|
68.39|             62.41|                 NULL|                 195.01|
38.01|                127.0|            NULL|
|2024-06-24 07:40:00|2024-06-24 07:40:00|       61.88| 72170|2024-06-24
07:45:00|              61.87|       165811|                 NULL|
68.34|             62.83|                 NULL|                 195.01|
NULL|                126.99|            NULL|
|2024-06-24 07:45:00|2024-06-24 07:45:00|       62.29| 25755|2024-06-24
07:50:00|              61.88|        72170|                 NULL|
68.29|             62.69|                 NULL|                 195.05|
38.0|                127.51|            NULL|
|2024-06-24 07:50:00|2024-06-24 07:50:00|       62.26| 31307|2024-06-24
07:55:00|              62.29|        25755|                 NULL|
68.24|              62.5|                 NULL|                  197.0|
38.59|                127.64|            NULL|
|2024-06-24 07:55:00|2024-06-24 07:55:00|       62.18|  6300|2024-06-24
08:00:00|              62.26|        31307|                 NULL|
68.18|              62.8|                 NULL|                 197.01|
39.0|                127.08|            NULL|
+-------------------+-------------------+------------+------+-------------------
+-------------------+-------------+---------------------+-------------------+---
---------------+-----------------+----------------------+-----------------------
+---------------------+----------------+
```

```
only showing top 5 rows
```

[ ]:

[86]:
```python
from pyspark.sql import functions as F

# Filtrar columnas que contienen "close" o "volume" con nombres completos para␣
 ↪evitar ambigüedad
relevant_columns = [col for col in combined_df_filtered.columns if "close_" in␣
 ↪col or "volume_" in col]
print(f"Relevant columns: {relevant_columns}")

# Seleccionar las columnas relevantes con nombres completos y convertir a un␣
 ↪DataFrame de Pandas
df_pandas = combined_df_filtered.select(*relevant_columns).toPandas()

# Generar la matriz de correlación
correlation_matrix = df_pandas.corr()

# Mostrar la matriz de correlación
print(correlation_matrix)
```

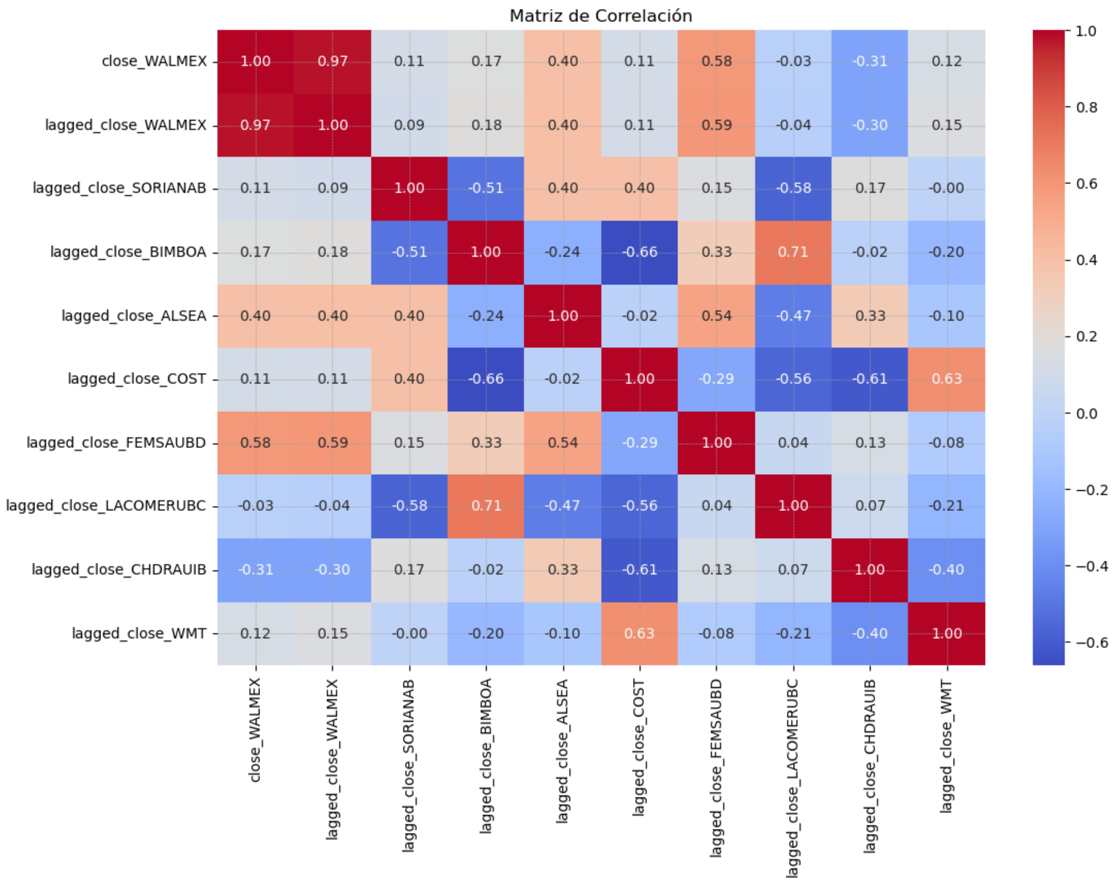
```
Relevant columns: ['close_WALMEX', 'lagged_close_WALMEX',
'lagged_close_SORIANAB', 'lagged_close_BIMBOA', 'lagged_close_ALSEA',
'lagged_close_COST', 'lagged_close_FEMSAUBD', 'lagged_close_LACOMERUBC',
'lagged_close_CHDRAUIB', 'lagged_close_WMT']
                         close_WALMEX  lagged_close_WALMEX  \
close_WALMEX                 1.000000             0.974324
lagged_close_WALMEX          0.974324             1.000000
lagged_close_SORIANAB        0.113386             0.094992
lagged_close_BIMBOA          0.168878             0.177320
lagged_close_ALSEA           0.397080             0.404669
lagged_close_COST            0.110172             0.112628
lagged_close_FEMSAUBD        0.584807             0.594909
lagged_close_LACOMERUBC     -0.034322            -0.039153
lagged_close_CHDRAUIB       -0.310118            -0.304773
lagged_close_WMT             0.124813             0.154884

                         lagged_close_SORIANAB  lagged_close_BIMBOA  \
close_WALMEX                          0.113386             0.168878
lagged_close_WALMEX                   0.094992             0.177320
lagged_close_SORIANAB                 1.000000            -0.505663
lagged_close_BIMBOA                  -0.505663             1.000000
lagged_close_ALSEA                    0.396951            -0.243041
lagged_close_COST                     0.398915            -0.661241
lagged_close_FEMSAUBD                 0.153841             0.325848
```

```
lagged_close_LACOMERUBC             -0.575410            0.712394
lagged_close_CHDRAUIB                0.170737           -0.019589
lagged_close_WMT                    -0.001988           -0.195611

                         lagged_close_ALSEA  lagged_close_COST  \
close_WALMEX                       0.397080           0.110172
lagged_close_WALMEX                0.404669           0.112628
lagged_close_SORIANAB              0.396951           0.398915
lagged_close_BIMBOA               -0.243041          -0.661241
lagged_close_ALSEA                 1.000000          -0.023556
lagged_close_COST                 -0.023556           1.000000
lagged_close_FEMSAUBD              0.539395          -0.285057
lagged_close_LACOMERUBC           -0.466752          -0.558046
lagged_close_CHDRAUIB              0.331714          -0.613384
lagged_close_WMT                  -0.103211           0.628363

                         lagged_close_FEMSAUBD  lagged_close_LACOMERUBC  \
close_WALMEX                          0.584807                -0.034322
lagged_close_WALMEX                   0.594909                -0.039153
lagged_close_SORIANAB                 0.153841                -0.575410
lagged_close_BIMBOA                   0.325848                 0.712394
lagged_close_ALSEA                    0.539395                -0.466752
lagged_close_COST                    -0.285057                -0.558046
lagged_close_FEMSAUBD                 1.000000                 0.044857
lagged_close_LACOMERUBC               0.044857                 1.000000
lagged_close_CHDRAUIB                 0.128213                 0.072405
lagged_close_WMT                     -0.075925                -0.208954

                         lagged_close_CHDRAUIB  lagged_close_WMT
close_WALMEX                         -0.310118          0.124813
lagged_close_WALMEX                  -0.304773          0.154884
lagged_close_SORIANAB                 0.170737         -0.001988
lagged_close_BIMBOA                  -0.019589         -0.195611
lagged_close_ALSEA                    0.331714         -0.103211
lagged_close_COST                    -0.613384          0.628363
lagged_close_FEMSAUBD                 0.128213         -0.075925
lagged_close_LACOMERUBC               0.072405         -0.208954
lagged_close_CHDRAUIB                 1.000000         -0.399015
lagged_close_WMT                     -0.399015          1.000000
```

[87]:

```python
import seaborn as sns
import matplotlib.pyplot as plt

# Generar el mapa de calor de la matriz de correlación
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Matriz de Correlación")
```

```python
plt.show()
```



[ ]:

[ ]:

[ ]:

[ ]:

2. Modelo de Regresión para Predicción de Precios

Este modelo de regresión lineal intenta predecir el precio en función de otras variables.

[88]:
```python
from functools import reduce
from pyspark.sql import functions as F

# Crear una condición que verifique si alguna columna tiene un valor nulo
null_condition = reduce(lambda a, b: a | b, [F.col(c).isNull() for c in␣
↪combined_df_filtered.columns])
```

```
# Filtrar filas con al menos un valor nulo en cualquier columna
df_with_any_nulls = combined_df_filtered.filter(null_condition)

df_with_any_nulls.orderBy("minute_start").show(5)
```

```
+-------------------+-------------------+------------+------+-------------------
+-------------------+-------------+---------------------+--------------------+---
---------------+-----------------+---------------------+----------------------
+---------------------+----------------+
|       minute_start|      datetime_5min|close_WALMEX|volume|         minute_end
|lagged_close_WALMEX|lagged_volume|lagged_close_SORIANAB|lagged_close_BIMBOA|lag
ged_close_ALSEA|lagged_close_COST|lagged_close_FEMSAUBD|lagged_close_LACOMERUBC|
lagged_close_CHDRAUIB|lagged_close_WMT|
+-------------------+-------------------+------------+------+-------------------
+-------------------+-------------+---------------------+--------------------+---
---------------+-----------------+---------------------+----------------------
+---------------------+----------------+
|2024-06-24 07:35:00|2024-06-24 07:35:00|       61.87|165811|2024-06-24
07:40:00|               61.6|        80481|                29.34|
68.39|             62.41|                 NULL|                195.01|
38.01|                127.0|               NULL|
|2024-06-24 07:40:00|2024-06-24 07:40:00|       61.88| 72170|2024-06-24
07:45:00|              61.87|       165811|                 NULL|
68.34|             62.83|                 NULL|                195.01|
NULL|               126.99|               NULL|
|2024-06-24 07:45:00|2024-06-24 07:45:00|       62.29| 25755|2024-06-24
07:50:00|              61.88|        72170|                 NULL|
68.29|             62.69|                 NULL|                195.05|
38.0|               127.51|               NULL|
|2024-06-24 07:50:00|2024-06-24 07:50:00|       62.26| 31307|2024-06-24
07:55:00|              62.29|        25755|                 NULL|
68.24|              62.5|                 NULL|                 197.0|
38.59|               127.64|               NULL|
|2024-06-24 07:55:00|2024-06-24 07:55:00|       62.18|  6300|2024-06-24
08:00:00|              62.26|        31307|                 NULL|
68.18|              62.8|                 NULL|                197.01|
39.0|               127.08|               NULL|
+-------------------+-------------------+------------+------+-------------------
+-------------------+-------------+---------------------+--------------------+---
---------------+-----------------+---------------------+----------------------
+---------------------+----------------+
only showing top 5 rows
```

[ ]:

[89]:
```python
from pyspark.sql import Window
from pyspark.sql import functions as F

# Define un esquema de ventana ordenado por "minute_start" para cada columna de␣
↪la emisora.
window_spec = Window.orderBy("minute_start").rowsBetween(Window.
↪unboundedPreceding, 0)

# Lista de columnas que contienen valores numéricos y requieren relleno hacia␣
↪adelante
columns_to_fill = [col for col in combined_df_filtered.columns if "close" in␣
↪col or "volume" in col]

# Aplica forward fill para cada columna en `columns_to_fill`
for col in columns_to_fill:
    combined_df_filtered = combined_df_filtered.withColumn(
        col, F.last(col, ignorenulls=True).over(window_spec)
    )

# Filtrar filas donde "close_WALMEX" es NULL o vacío
combined_df_filtered = combined_df_filtered.filter(F.col("close_WALMEX").
↪isNotNull())

# Mostrar algunos registros después de aplicar el relleno
combined_df_filtered.orderBy("minute_start").show(5)
```

```
+-------------------+-------------------+------------+------+-------------------
+-------------------+-------------+---------------------+-------------------+---
---------------+-----------------+---------------------+----------------------
+---------------------+----------------+
|       minute_start|      datetime_5min|close_WALMEX|volume|         minute_end
|lagged_close_WALMEX|lagged_volume|lagged_close_SORIANAB|lagged_close_BIMBOA|lag
ged_close_ALSEA|lagged_close_COST|lagged_close_FEMSAUBD|lagged_close_LACOMERUBC|
lagged_close_CHDRAUIB|lagged_close_WMT|
+-------------------+-------------------+------------+------+-------------------
+-------------------+-------------+---------------------+-------------------+---
---------------+-----------------+---------------------+----------------------
+---------------------+----------------+
|2024-06-24 07:35:00|2024-06-24 07:35:00|       61.87|165811|2024-06-24
07:40:00|               61.6|        80481|                29.34|
68.39|             62.41|              NULL|                195.01|
38.01|                127.0|               NULL|
|2024-06-24 07:40:00|2024-06-24 07:40:00|       61.88| 72170|2024-06-24
07:45:00|              61.87|       165811|                29.34|
68.34|             62.83|              NULL|                195.01|
38.01|               126.99|               NULL|
|2024-06-24 07:45:00|2024-06-24 07:45:00|       62.29| 25755|2024-06-24
```

```
07:50:00|                61.88|        72170|                29.34|
68.29|              62.69|             NULL|                195.05|
38.0|                127.51|             NULL|
|2024-06-24 07:50:00|2024-06-24 07:50:00|       62.26| 31307|2024-06-24
07:55:00|                62.29|        25755|                29.34|
68.24|               62.5|               NULL|                197.0|
38.59|                127.64|             NULL|
|2024-06-24 07:55:00|2024-06-24 07:55:00|       62.18|  6300|2024-06-24
08:00:00|                62.26|        31307|                29.34|
68.18|               62.8|               NULL|                197.01|
39.0|                127.08|             NULL|
+-------------------+-------------------+------------+------+-------------------
+-------------------+-------------+---------------------+-------------------+---
---------------+-----------------+---------------------+----------------------
+---------------------+----------------+
only showing top 5 rows
```

[90]:

```
from pyspark.sql import Window
from pyspark.sql import functions as F

# Lista de columnas que necesitan relleno hacia atrás
columns_to_fill = [col for col in combined_df_filtered.columns if "close" in␣
↪col or "volume" in col]

# Hacer una copia temporal del DataFrame ordenado en orden descendente para␣
↪simular un backward fill
df_desc = combined_df_filtered.orderBy(F.col("minute_start").desc())

# Aplicar el relleno hacia adelante en el DataFrame ordenado en forma␣
↪descendente
for col in columns_to_fill:
    window_spec = Window.orderBy(F.col("minute_start").desc()).
↪rowsBetween(Window.unboundedPreceding, 0)
    df_desc = df_desc.withColumn(col, F.last(col, ignorenulls=True).
↪over(window_spec))

# Restaurar el orden ascendente original
combined_df_filled = df_desc.orderBy("minute_start")

# Mostrar algunos registros después de aplicar el relleno hacia atrás para␣
↪verificar
combined_df_filled.show(5)
```

```
+-------------------+-------------------+------------+------+-------------------
+-------------------+-------------+---------------------+-------------------+---
---------------+-----------------+---------------------+----------------------
```

```
+--------------------+----------------+
|       minute_start|      datetime_5min|close_WALMEX|volume|         minute_end
|lagged_close_WALMEX|lagged_volume|lagged_close_SORIANAB|lagged_close_BIMBOA|lag
ged_close_ALSEA|lagged_close_COST|lagged_close_FEMSAUBD|lagged_close_LACOMERUBC|
lagged_close_CHDRAUIB|lagged_close_WMT|
+-------------------+-------------------+------------+------+-------------------
+-------------------+-------------+---------------------+-------------------+---
---------------+-----------------+---------------------+----------------------
+--------------------+----------------+
|2024-06-24 07:35:00|2024-06-24 07:35:00|       61.87|165811|2024-06-24
07:40:00|               61.6|        80481|                29.34|
68.39|             62.41|            15260.0|                195.01|
38.01|                 127.0|          1230.01|
|2024-06-24 07:40:00|2024-06-24 07:40:00|       61.88| 72170|2024-06-24
07:45:00|              61.87|       165811|                29.34|
68.34|             62.83|            15260.0|                195.01|
38.01|               126.99|          1230.01|
|2024-06-24 07:45:00|2024-06-24 07:45:00|       62.29| 25755|2024-06-24
07:50:00|              61.88|        72170|                29.34|
68.29|             62.69|            15260.0|                195.05|
38.0|               127.51|          1230.01|
|2024-06-24 07:50:00|2024-06-24 07:50:00|       62.26| 31307|2024-06-24
07:55:00|              62.29|        25755|                29.34|
68.24|              62.5|            15260.0|                 197.0|
38.59|               127.64|          1230.01|
|2024-06-24 07:55:00|2024-06-24 07:55:00|       62.18|  6300|2024-06-24
08:00:00|              62.26|        31307|                29.34|
68.18|              62.8|            15260.0|                197.01|
39.0|               127.08|          1230.01|
+-------------------+-------------------+------------+------+-------------------
+-------------------+-------------+---------------------+-------------------+---
---------------+-----------------+---------------------+----------------------
+--------------------+----------------+
only showing top 5 rows
```

[91]:
```python
# Mostrar algunos registros después de aplicar el relleno hacia atrás para
↪verificar
combined_df_filtered.orderBy("minute_start").show(5)
```

```
+-------------------+-------------------+------------+------+-------------------
+-------------------+-------------+---------------------+-------------------+---
---------------+-----------------+---------------------+----------------------
+--------------------+----------------+
|       minute_start|      datetime_5min|close_WALMEX|volume|         minute_end
|lagged_close_WALMEX|lagged_volume|lagged_close_SORIANAB|lagged_close_BIMBOA|lag
ged_close_ALSEA|lagged_close_COST|lagged_close_FEMSAUBD|lagged_close_LACOMERUBC|
lagged_close_CHDRAUIB|lagged_close_WMT|
```

```
+-------------------+-------------------+------------+------+-------------------
+-------------------+-------------+---------------------+-------------------+---
---------------+-----------------+---------------------+----------------------
+---------------------+----------------+
|2024-06-24 07:35:00|2024-06-24 07:35:00|       61.87|165811|2024-06-24
07:40:00|                61.6|        80481|                29.34|
68.39|            62.41|              NULL|                195.01|
38.01|               127.0|              NULL|
|2024-06-24 07:40:00|2024-06-24 07:40:00|       61.88| 72170|2024-06-24
07:45:00|               61.87|       165811|                29.34|
68.34|            62.83|              NULL|                195.01|
38.01|              126.99|              NULL|
|2024-06-24 07:45:00|2024-06-24 07:45:00|       62.29| 25755|2024-06-24
07:50:00|               61.88|        72170|                29.34|
68.29|            62.69|              NULL|                195.05|
38.0|              127.51|              NULL|
|2024-06-24 07:50:00|2024-06-24 07:50:00|       62.26| 31307|2024-06-24
07:55:00|               62.29|        25755|                29.34|
68.24|             62.5|              NULL|                 197.0|
38.59|              127.64|              NULL|
|2024-06-24 07:55:00|2024-06-24 07:55:00|       62.18|  6300|2024-06-24
08:00:00|               62.26|        31307|                29.34|
68.18|             62.8|              NULL|                197.01|
39.0|              127.08|              NULL|
+-------------------+-------------------+------------+------+-------------------
+-------------------+-------------+---------------------+-------------------+---
---------------+-----------------+---------------------+----------------------
+---------------------+----------------+
only showing top 5 rows
```

[ ]:

```

```

[92]:

```python
from pyspark.ml.regression import RandomForestRegressor, GBTRegressor,␣
↪LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.feature import VectorAssembler
import pandas as pd

# Preparar los datos para los modelos de regresión
if "features" in combined_df_filtered.columns:
    combined_df_filtered = combined_df_filtered.drop("features")

# Seleccionar características y etiqueta
```

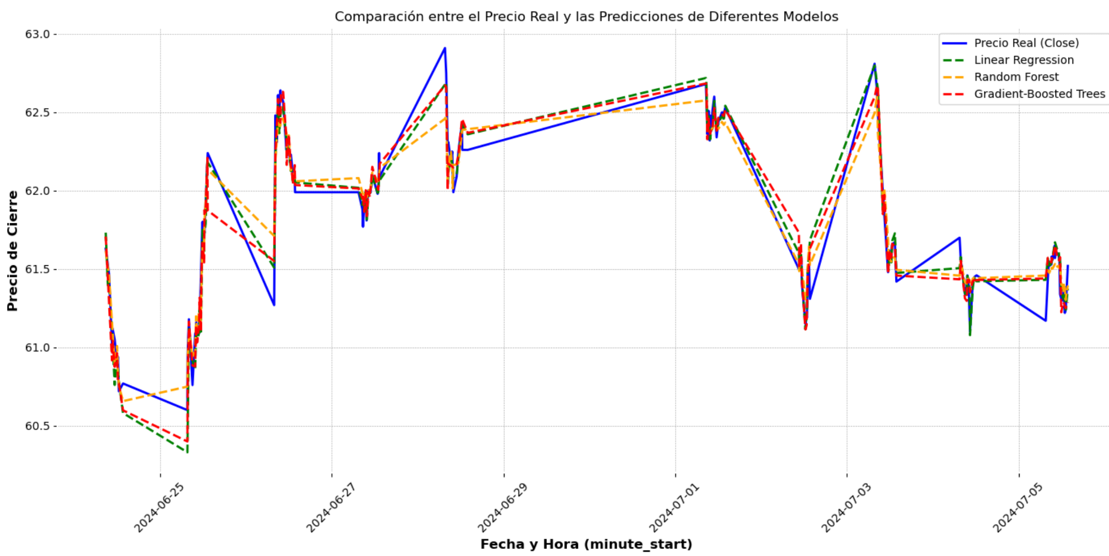
```python
feature_cols = ["lagged_close_WALMEX", "lagged_close_SORIANAB", "lagged_close_FEMSAUBD", "lagged_close_LACOMERUBC", "lagged_close_CHDRAUIB", "lagged_close_WMT", "lagged_close_BIMBOA", "lagged_close_COST", "lagged_close_ALSEA"]

# Eliminar filas con valores nulos en las columnas de características y etiqueta
combined_df_filtered = combined_df_filtered.dropna(subset=feature_cols + ["close_WALMEX"])

# Ensamblar características en una columna vectorial
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
combined_df_filtered = assembler.transform(combined_df_filtered)

# Dividir los datos en conjunto de entrenamiento y prueba
train, test = combined_df_filtered.randomSplit([0.8, 0.2], seed=747)

# Definir evaluadores para RMSE, MAE y R2
evaluator_rmse = RegressionEvaluator(labelCol="close_WALMEX", predictionCol="prediction", metricName="rmse")
evaluator_mae = RegressionEvaluator(labelCol="close_WALMEX", predictionCol="prediction", metricName="mae")
evaluator_r2 = RegressionEvaluator(labelCol="close_WALMEX", predictionCol="prediction", metricName="r2")

# Lista para almacenar los resultados de las métricas de cada modelo
results = []

# -----------------------------------------------------------
# Modelo 1: LinearRegression
# -----------------------------------------------------------
lr = LinearRegression(featuresCol="features", labelCol="close_WALMEX", predictionCol="prediction")
lr_model = lr.fit(train)
lr_predictions = lr_model.transform(test)

# Evaluación del modelo de Linear Regression
lr_rmse = evaluator_rmse.evaluate(lr_predictions)
lr_mae = evaluator_mae.evaluate(lr_predictions)
lr_r2 = evaluator_r2.evaluate(lr_predictions)

# Guardar resultados
results.append({"Model": "LinearRegression", "RMSE": lr_rmse, "MAE": lr_mae, "R2": lr_r2})

# -----------------------------------------------------------
# Modelo 2: RandomForestRegressor
```

```python
# ----------------------------------------------------------
rf = RandomForestRegressor(featuresCol="features", labelCol="close_WALMEX",␣
 ↪predictionCol="prediction")
rf_model = rf.fit(train)
rf_predictions = rf_model.transform(test)

# Evaluación del modelo de Random Forest
rf_rmse = evaluator_rmse.evaluate(rf_predictions)
rf_mae = evaluator_mae.evaluate(rf_predictions)
rf_r2 = evaluator_r2.evaluate(rf_predictions)

# Guardar resultados
results.append({"Model": "RandomForestRegressor", "RMSE": rf_rmse, "MAE":␣
 ↪rf_mae, "R2": rf_r2})

# ----------------------------------------------------------
# Modelo 3: Gradient-Boosted Tree Regressor
# ----------------------------------------------------------
gbt = GBTRegressor(featuresCol="features", labelCol="close_WALMEX",␣
 ↪predictionCol="prediction")
gbt_model = gbt.fit(train)
gbt_predictions = gbt_model.transform(test)

# Evaluación del modelo de GBT
gbt_rmse = evaluator_rmse.evaluate(gbt_predictions)
gbt_mae = evaluator_mae.evaluate(gbt_predictions)
gbt_r2 = evaluator_r2.evaluate(gbt_predictions)

# Guardar resultados
results.append({"Model": "GBTRegressor", "RMSE": gbt_rmse, "MAE": gbt_mae, "R2":
 ↪ gbt_r2})

# ----------------------------------------------------------
# Crear un DataFrame de pandas con los resultados y ordenar por RMSE
# ----------------------------------------------------------
results_df = pd.DataFrame(results)
results_df = results_df.sort_values(by="RMSE", ascending=True)

print("Resultados de las métricas para cada modelo:")
print(results_df)
```

```
Resultados de las métricas para cada modelo:
                   Model      RMSE       MAE        R2
0       LinearRegression  0.108654  0.074824  0.954414
2           GBTRegressor  0.117126  0.085092  0.947028
1  RandomForestRegressor  0.122030  0.094069  0.942500
```

[93]:
```python
import matplotlib.pyplot as plt

# Convertir el conjunto de prueba y las predicciones de cada modelo en␣
↪DataFrames de Pandas para la visualización
test_pd = test.select("minute_start", "close_WALMEX").toPandas()
lr_predictions_pd = lr_predictions.select("minute_start", "prediction").
↪toPandas()
rf_predictions_pd = rf_predictions.select("minute_start", "prediction").
↪toPandas()
gbt_predictions_pd = gbt_predictions.select("minute_start", "prediction").
↪toPandas()

# Renombrar las columnas para cada modelo para evitar conflictos en el␣
↪DataFrame combinado
lr_predictions_pd = lr_predictions_pd.rename(columns={"prediction":␣
↪"predicted_close_lr"})
rf_predictions_pd = rf_predictions_pd.rename(columns={"prediction":␣
↪"predicted_close_rf"})
gbt_predictions_pd = gbt_predictions_pd.rename(columns={"prediction":␣
↪"predicted_close_gbt"})

# Combinar los resultados en un único DataFrame de Pandas para facilitar la␣
↪comparación
comparison_df = test_pd.merge(lr_predictions_pd, on="minute_start", how="left")
comparison_df = comparison_df.merge(rf_predictions_pd, on="minute_start",␣
↪how="left")
comparison_df = comparison_df.merge(gbt_predictions_pd, on="minute_start",␣
↪how="left")

# Configuración de la gráfica
plt.figure(figsize=(14, 7))

# Graficar el precio real
plt.plot(comparison_df["minute_start"], comparison_df["close_WALMEX"],␣
↪label="Precio Real (Close)", color="blue")

# Graficar las predicciones de cada modelo
plt.plot(comparison_df["minute_start"], comparison_df["predicted_close_lr"],␣
↪label="Linear Regression", color="green", linestyle="--")
plt.plot(comparison_df["minute_start"], comparison_df["predicted_close_rf"],␣
↪label="Random Forest", color="orange", linestyle="--")
plt.plot(comparison_df["minute_start"], comparison_df["predicted_close_gbt"],␣
↪label="Gradient-Boosted Trees", color="red", linestyle="--")

# Etiquetas y título
plt.xlabel("Fecha y Hora (minute_start)")
```

```
plt.ylabel("Precio de Cierre")
plt.title("Comparación entre el Precio Real y las Predicciones de Diferentes␣
 ↪Modelos")
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()

# Mostrar la gráfica
plt.show()
```



[ ]:

```

```

[ ]:

```

```

3. Clasificación para Determinar si el Precio Subirá o Bajará

Este modelo de clasificación utiliza LogisticRegression para clasificar el comportamiento del precio (sube o baja).

[94]:

```
from pyspark.sql import functions as F
from pyspark.sql.types import DoubleType
from pyspark.sql import Window
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.evaluation import BinaryClassificationEvaluator

if "features" in combined_df_filtered.columns:
    combined_df_filtered = combined_df_filtered.drop("features")
```

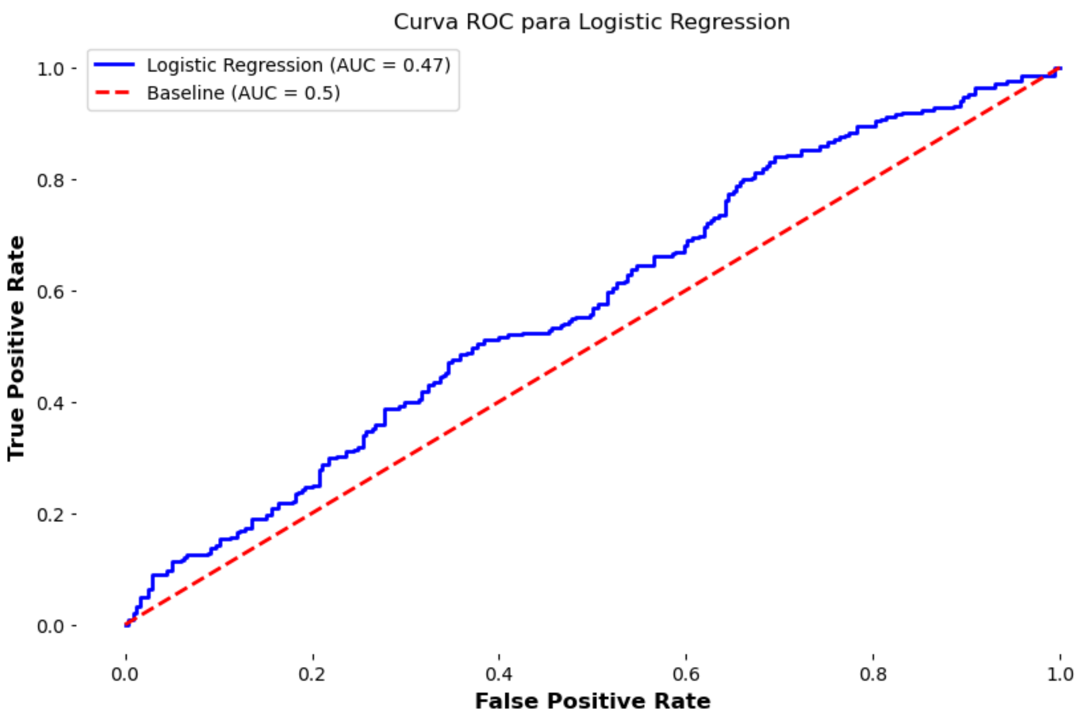
```python
# Crear columna que indique si el precio sube o baja en el siguiente intervalo
window_spec = Window.orderBy("minute_start")
combined_df_filtered = combined_df_filtered.withColumn("next_close_WALMEX", F.
 ↪lead("close_WALMEX").over(window_spec))

# Crear la columna de clasificación 'price_direction': 1 si el precio sube, 0␣
 ↪si baja
combined_df_filtered = combined_df_filtered.withColumn(
    "price_direction",
    F.when(F.col("next_close_WALMEX") > F.col("close_WALMEX"), 1).otherwise(0)
)

# Convertir las columnas de características a DoubleType
for col in feature_cols:
    combined_df_filtered = combined_df_filtered.withColumn(col, F.col(col).
 ↪cast(DoubleType()))

# Eliminar filas con valores nulos en las columnas de características y en la␣
 ↪etiqueta
combined_df_filtered = combined_df_filtered.dropna(subset=feature_cols +␣
 ↪["price_direction"])

# Ensamblar las características en una columna vectorial
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
combined_df_filtered = assembler.transform(combined_df_filtered)

# Dividir los datos en conjunto de entrenamiento y prueba
train, test = combined_df_filtered.randomSplit([0.8, 0.2], seed=747)

# Definir el modelo de clasificación: Logistic Regression
lr = LogisticRegression(featuresCol="features", labelCol="price_direction",␣
 ↪predictionCol="prediction")
lr_model = lr.fit(train)

# Realizar predicciones en el conjunto de prueba
lr_predictions = lr_model.transform(test)

# Evaluar el modelo de Logistic Regression usando el área bajo la curva ROC
evaluator = BinaryClassificationEvaluator(labelCol="price_direction",␣
 ↪metricName="areaUnderROC")
roc_auc = evaluator.evaluate(lr_predictions)

print(f"Área bajo la curva ROC para el modelo Logistic Regression = {roc_auc}")

# Mostrar algunas predicciones
```

```
lr_predictions.select("minute_start", "close_WALMEX", "price_direction",␣
↪"prediction", "probability").show(5)
```

```
Área bajo la curva ROC para el modelo Logistic Regression = 0.4740428293316028
+-------------------+------------+---------------+----------+--------------------
-+
|       minute_start|close_WALMEX|price_direction|prediction|
probability|
+-------------------+------------+---------------+----------+--------------------
-+
|2024-06-24 08:50:00|       61.63|              1|
0.0|[0.69810642454301…|
|2024-06-24 09:25:00|       61.54|              1|
0.0|[0.66270072875156…|
|2024-06-24 09:45:00|       61.47|              0|
0.0|[0.63638500208266…|
|2024-06-24 10:35:00|       61.02|              1|
0.0|[0.61763074262136…|
|2024-06-24 10:45:00|       61.12|              0|
0.0|[0.61506998920113…|
+-------------------+------------+---------------+----------+--------------------
-+
only showing top 5 rows
```

[95]:

```
# Obtener los puntos de la curva ROC
training_summary = lr_model.summary
roc = training_summary.roc.toPandas()

# Graficar la curva ROC
plt.figure(figsize=(10, 6))
plt.plot(roc['FPR'], roc['TPR'], label=f'Logistic Regression (AUC = {roc_auc:.
↪2f})', color="blue")
plt.plot([0, 1], [0, 1], 'r--', label="Baseline (AUC = 0.5)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Curva ROC para Logistic Regression")
plt.legend(loc="best")
plt.grid()
plt.show()
```



[ ]:

[ ]:

[96]:
```python
from pyspark.ml.classification import RandomForestClassifier, GBTClassifier

if "features" in combined_df_filtered.columns:
    combined_df_filtered = combined_df_filtered.drop("features")

# Ejemplo de Random Forest
rf = RandomForestClassifier(featuresCol="features", labelCol="price_direction",␣
↪predictionCol="prediction")
rf_model = rf.fit(train)
rf_predictions = rf_model.transform(test)

# Evaluación
rf_auc = evaluator.evaluate(rf_predictions)
print(f"Área bajo la curva ROC para el modelo Random Forest = {rf_auc}")

# Ejemplo de Gradient Boosted Trees
gbt = GBTClassifier(featuresCol="features", labelCol="price_direction",␣
↪predictionCol="prediction")
gbt_model = gbt.fit(train)
gbt_predictions = gbt_model.transform(test)
```

```
# Evaluación
gbt_auc = evaluator.evaluate(gbt_predictions)
print(f"Área bajo la curva ROC para el modelo Gradient Boosted Trees = {gbt_auc}")
```

```
Área bajo la curva ROC para el modelo Random Forest = 0.5187378325762493
Área bajo la curva ROC para el modelo Gradient Boosted Trees =
0.4484912394548994
```

[ ]:

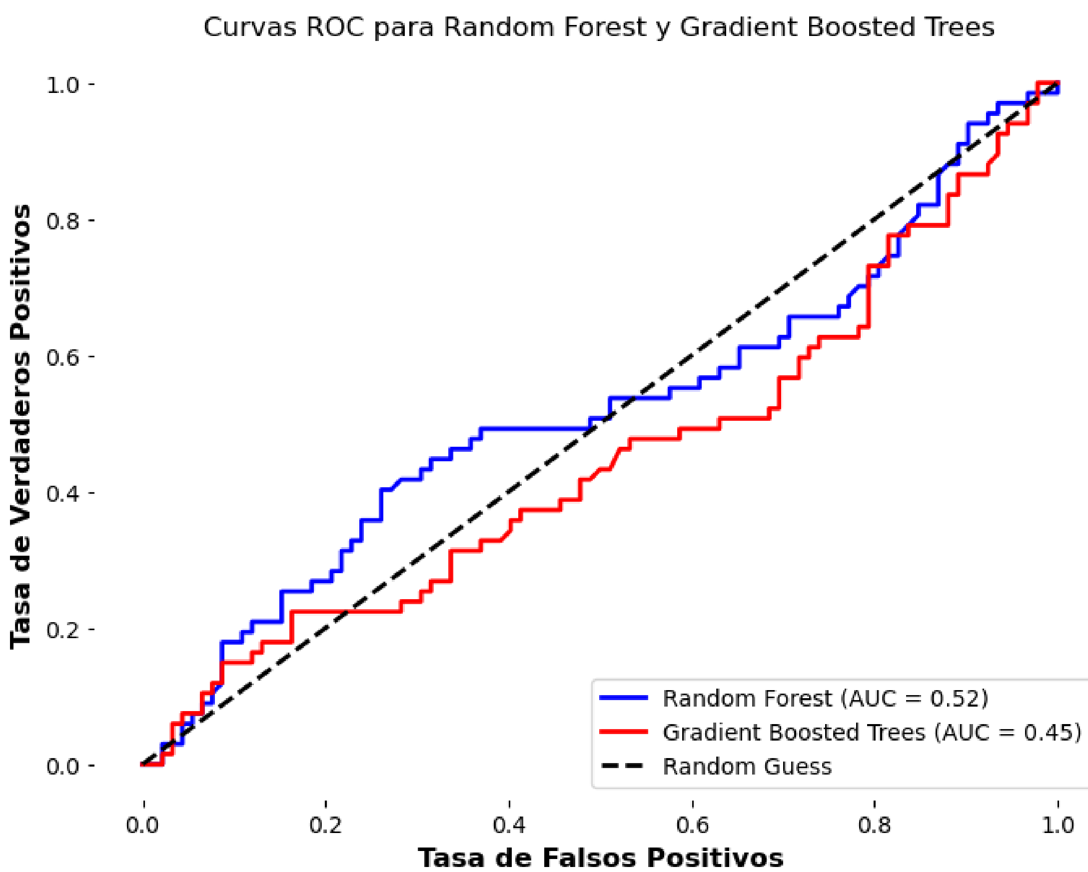
[97]:
```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
import matplotlib.pyplot as plt
import numpy as np
from pyspark.sql import functions as F
from pyspark.ml.functions import vector_to_array
from sklearn.metrics import roc_curve

# Evaluador AUC
evaluator = BinaryClassificationEvaluator(labelCol="price_direction", rawPredictionCol="rawPrediction", metricName="areaUnderROC")

# Entrenar el modelo Random Forest y evaluar AUC
rf = RandomForestClassifier(featuresCol="features", labelCol="price_direction", predictionCol="prediction", probabilityCol="probability")
rf_model = rf.fit(train)
rf_predictions = rf_model.transform(test)
rf_auc = evaluator.evaluate(rf_predictions)
print(f"Área bajo la curva ROC para el modelo Random Forest = {rf_auc}")

# Convertir el vector de probabilidad a un arreglo para obtener la probabilidad de la clase positiva
rf_roc = rf_predictions.withColumn("prob_positive", vector_to_array("probability")[1])

# Entrenar el modelo Gradient Boosted Trees (sin el argumento `rawPredictionCol` o `probabilityCol`)
gbt = GBTClassifier(featuresCol="features", labelCol="price_direction", predictionCol="prediction")
gbt_model = gbt.fit(train)
gbt_predictions = gbt_model.transform(test)
gbt_auc = evaluator.evaluate(gbt_predictions)
print(f"Área bajo la curva ROC para el modelo Gradient Boosted Trees = {gbt_auc}")
```

```
Área bajo la curva ROC para el modelo Random Forest = 0.5187378325762493
```

```
Área bajo la curva ROC para el modelo Gradient Boosted Trees =
0.4484912394548994
```

[98]:

```
# Extraer la probabilidad para la clase positiva desde `rawPrediction` en el␣
↪modelo GBT
gbt_roc = gbt_predictions.withColumn("prob_positive",␣
↪vector_to_array("rawPrediction")[1])

# Convertir a Pandas para graficar
rf_roc_pd = rf_roc.select("prob_positive", "price_direction").toPandas()
gbt_roc_pd = gbt_roc.select("prob_positive", "price_direction").toPandas()

# Generar FPR y TPR para diferentes umbrales
# Curva ROC para Random Forest
rf_fpr, rf_tpr, _ = roc_curve(rf_roc_pd["price_direction"],␣
↪rf_roc_pd["prob_positive"])

# Curva ROC para Gradient Boosted Trees
gbt_fpr, gbt_tpr, _ = roc_curve(gbt_roc_pd["price_direction"],␣
↪gbt_roc_pd["prob_positive"])

# Graficar las curvas ROC
plt.figure(figsize=(8, 6))
plt.plot(rf_fpr, rf_tpr, label=f"Random Forest (AUC = {rf_auc:.2f})",␣
↪color="blue")
plt.plot(gbt_fpr, gbt_tpr, label=f"Gradient Boosted Trees (AUC = {gbt_auc:.
↪2f})", color="red")
plt.plot([0, 1], [0, 1], 'k--', label="Random Guess")

plt.xlabel("Tasa de Falsos Positivos")
plt.ylabel("Tasa de Verdaderos Positivos")
plt.title("Curvas ROC para Random Forest y Gradient Boosted Trees")
plt.legend(loc="lower right")
plt.grid()
plt.show()
```

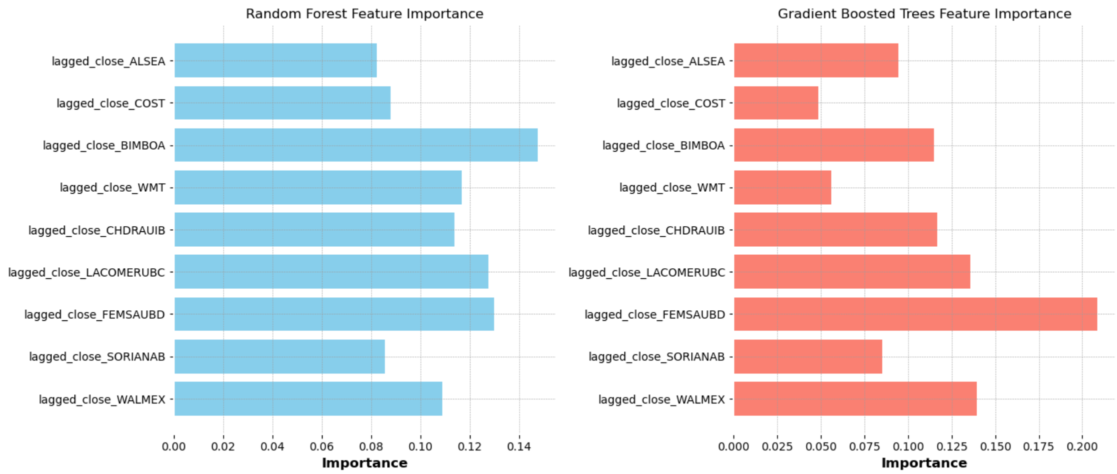


[99]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from pyspark.ml.classification import RandomForestClassifier, GBTClassifier
from pyspark.ml.feature import VectorAssembler

# Assuming `combined_df_filtered` has been prepared with feature columns and␣
↪label
if "features" in combined_df_filtered.columns:
    combined_df_filtered = combined_df_filtered.drop("features")

# Specify feature columns
feature_cols = ["lagged_close_WALMEX", "lagged_close_SORIANAB",␣
↪"lagged_close_FEMSAUBD",
                "lagged_close_LACOMERUBC", "lagged_close_CHDRAUIB",␣
↪"lagged_close_WMT",
                "lagged_close_BIMBOA", "lagged_close_COST",␣
↪"lagged_close_ALSEA"]
```

```python
# Assemble features
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
combined_df_filtered = assembler.transform(combined_df_filtered)

# Split data into training and testing
train, test = combined_df_filtered.randomSplit([0.8, 0.2], seed=747)

# Initialize models
rf = RandomForestClassifier(featuresCol="features", labelCol="price_direction",␣
↪predictionCol="prediction")
gbt = GBTClassifier(featuresCol="features", labelCol="price_direction",␣
↪predictionCol="prediction")

# Train the models
rf_model = rf.fit(train)
gbt_model = gbt.fit(train)

# Extract feature importances
rf_feature_importances = rf_model.featureImportances.toArray()
gbt_feature_importances = gbt_model.featureImportances.toArray()

# Convert to DataFrames for easier plotting
importance_df = pd.DataFrame({
    "Feature": feature_cols,
    "RandomForest Importance": rf_feature_importances,
    "GBT Importance": gbt_feature_importances
})
```

[100]:

```python
# Plot feature importances
fig, ax = plt.subplots(1, 2, figsize=(14, 6))

# Random Forest feature importance
ax[0].barh(importance_df["Feature"], importance_df["RandomForest Importance"],␣
↪color="skyblue")
ax[0].set_title("Random Forest Feature Importance")
ax[0].set_xlabel("Importance")

# Gradient Boosted Trees feature importance
ax[1].barh(importance_df["Feature"], importance_df["GBT Importance"],␣
↪color="salmon")
ax[1].set_title("Gradient Boosted Trees Feature Importance")
ax[1].set_xlabel("Importance")

plt.tight_layout()
plt.show()
```



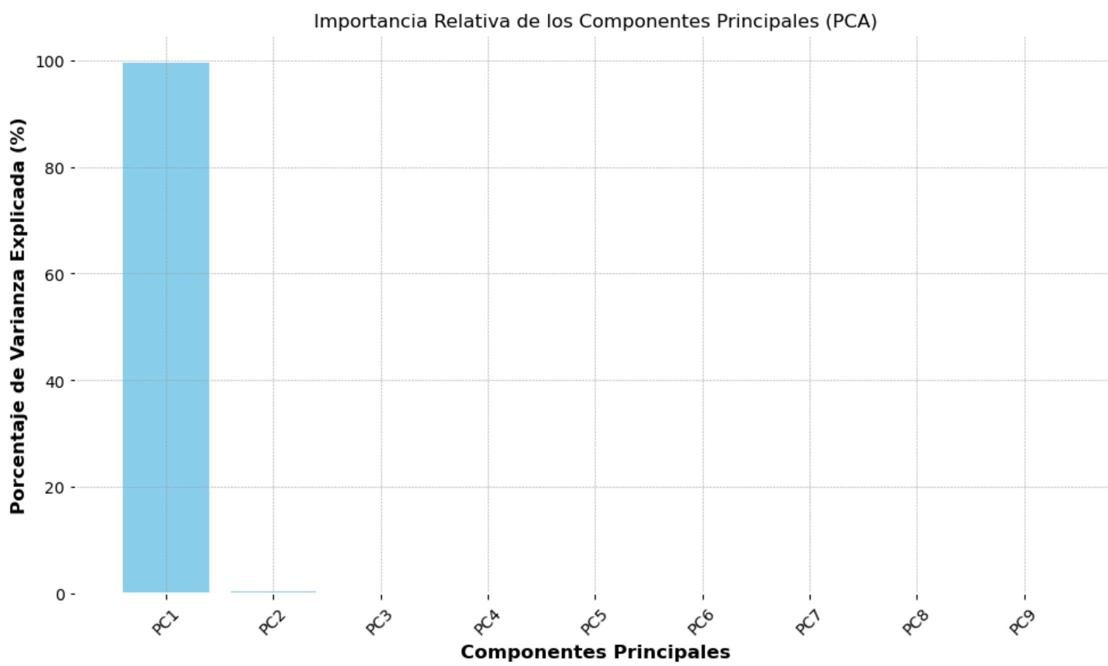
[ ]:

```

```

[101]:

```python
from pyspark.ml.feature import PCA, VectorAssembler
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

if "features" in combined_df_filtered.columns:
    combined_df_filtered = combined_df_filtered.drop("features")

# Seleccionar características para PCA
feature_cols = ["lagged_close_WALMEX", "lagged_close_SORIANAB",␣
 ↪"lagged_close_FEMSAUBD",
                "lagged_close_LACOMERUBC", "lagged_close_CHDRAUIB",␣
 ↪"lagged_close_WMT",
                "lagged_close_BIMBOA", "lagged_close_COST",␣
 ↪"lagged_close_ALSEA"]

# Ensamblar características en una columna vectorial
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
combined_df_filtered = assembler.transform(combined_df_filtered)

# Configurar y ajustar el modelo de PCA
pca = PCA(k=len(feature_cols), inputCol="features", outputCol="pca_features")
pca_model = pca.fit(combined_df_filtered)

# Extraer la importancia de cada componente
explained_variance = pca_model.explainedVariance.toArray()
```

```python
# Crear una gráfica de barras para mostrar la varianza explicada por cada␣
↪componente principal
plt.figure(figsize=(10, 6))
components = [f'PC{i+1}' for i in range(len(explained_variance))]
plt.bar(components, explained_variance * 100, color="skyblue")

# Etiquetas y título
plt.xlabel("Componentes Principales")
plt.ylabel("Porcentaje de Varianza Explicada (%)")
plt.title("Importancia Relativa de los Componentes Principales (PCA)")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Mostrar la varianza acumulada para cada componente principal en un DataFrame
cumulative_variance = np.cumsum(explained_variance)
variance_df = pd.DataFrame({"Componente Principal": components,
                            "Varianza Explicada (%)": explained_variance * 100,
                            "Varianza Acumulada (%)": cumulative_variance *␣
↪100})
print(variance_df)
```




```
  Componente Principal  Varianza Explicada (%)  Varianza Acumulada (%)
0                  PC1               99.594358               99.594358
1                  PC2                0.380991               99.975349
2                  PC3                0.014806               99.990155
```

```
3                  PC4              0.005489            99.995643
4                  PC5              0.001795            99.997439
5                  PC6              0.001123            99.998562
6                  PC7              0.000622            99.999184
7                  PC8              0.000530            99.999713
8                  PC9              0.000287           100.000000
```

[102]:
```python
# Obtener las cargas de los componentes principales
loadings = pca_model.pc.toArray()

# Crear un DataFrame para las cargas
loadings_df = pd.DataFrame(loadings, columns=feature_cols, index=[f"PC{i+1}"␣
↪for i in range(len(feature_cols))])
print("Cargas de los Componentes Principales:")
print(loadings_df)
```

```
Cargas de los Componentes Principales:
     lagged_close_WALMEX  lagged_close_SORIANAB  lagged_close_FEMSAUBD  \
PC1            -0.000742              -0.001303              -0.163739
PC2            -0.001136              -0.005580              -0.052518
PC3             0.001117               0.000874              -0.934943
PC4             0.002139               0.018049               0.006085
PC5             0.001971              -0.001470              -0.037756
PC6            -0.044746               0.998248               0.001426
PC7             0.003935               0.030096              -0.177006
PC8            -0.998985              -0.044540              -0.001527
PC9            -0.000633              -0.015795              -0.252023

     lagged_close_LACOMERUBC  lagged_close_CHDRAUIB  lagged_close_WMT  \
PC1                -0.087500               0.156158         -0.206606
PC2                 0.287338               0.107603         -0.182070
PC3                 0.001458               0.064478          0.319845
PC4                -0.367676              -0.411135          0.350006
PC5                 0.407865              -0.840728          0.022138
PC6                 0.035027               0.014629          0.000110
PC7                -0.672180              -0.285747         -0.580679
PC8                -0.004709              -0.004457         -0.000400
PC9                 0.393908              -0.046247         -0.601399

     lagged_close_BIMBOA  lagged_close_COST  lagged_close_ALSEA
PC1             0.450348           0.204546            0.808584
PC2            -0.050664           0.898276           -0.245871
PC3            -0.131709          -0.017005           -0.042235
PC4             0.670041           0.196013           -0.292462
PC5            -0.186193           0.043374            0.297242
PC6             0.004024          -0.005904            0.002102
PC7            -0.292533           0.088169           -0.061094
PC8            -0.000979           0.000128           -0.000536
```

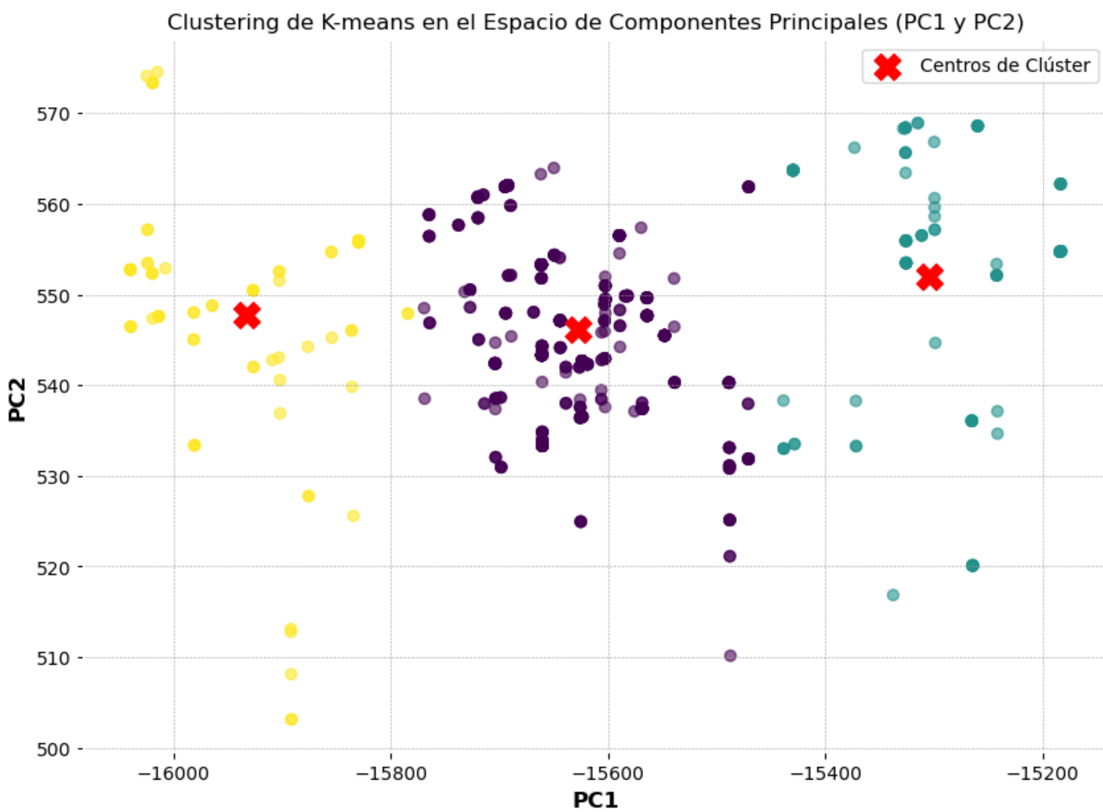
PC9   0.456133   -0.320720   -0.326086

[103]:
```
from pyspark.ml.feature import PCA, VectorAssembler
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
import matplotlib.pyplot as plt
import pandas as pd
from pyspark.sql import functions as F
from pyspark.sql.types import ArrayType, DoubleType
from pyspark.sql import SparkSession

if "features" in combined_df_filtered.columns:
    combined_df_filtered = combined_df_filtered.drop("features")

# Función UDF para convertir un vector en un array
def vector_to_array_udf(v):
    return v.toArray().tolist()

vector_to_array = F.udf(vector_to_array_udf, ArrayType(DoubleType()))

# Paso 1: Preparar los datos para PCA
feature_cols = ["lagged_close_WALMEX", "lagged_close_SORIANAB",␣
 ↪"lagged_close_FEMSAUBD",
                "lagged_close_LACOMERUBC", "lagged_close_CHDRAUIB",␣
 ↪"lagged_close_WMT",
                "lagged_close_BIMBOA", "lagged_close_COST",␣
 ↪"lagged_close_ALSEA"]

assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
assembled_df = assembler.transform(combined_df_filtered)

# Paso 2: Aplicar PCA para reducir a dos componentes principales
pca = PCA(k=2, inputCol="features", outputCol="pca_features")
pca_model = pca.fit(assembled_df)
pca_df = pca_model.transform(assembled_df)

# Convertir el vector de PCA a un array para facilitar la extracción de PC1 y␣
 ↪PC2
pca_df = pca_df.withColumn("pca_array", vector_to_array(F.col("pca_features")))␣
 ↪\
               .withColumn("PC1", F.col("pca_array")[0]) \
               .withColumn("PC2", F.col("pca_array")[1])

# Paso 3: Aplicar K-means en el espacio de los componentes principales
kmeans_assembler = VectorAssembler(inputCols=["PC1", "PC2"],␣
 ↪outputCol="pca_features_kmeans")
kmeans_input_df = kmeans_assembler.transform(pca_df)
```

```
kmeans = KMeans(featuresCol="pca_features_kmeans", k=3, seed=1)
kmeans_model = kmeans.fit(kmeans_input_df)
kmeans_predictions = kmeans_model.transform(kmeans_input_df)
```

[104]:
```
# Evaluar la calidad del modelo K-means
evaluator = ClusteringEvaluator(featuresCol="pca_features_kmeans",␣
 ↪predictionCol="prediction", metricName="silhouette")
silhouette_score = evaluator.evaluate(kmeans_predictions)
print(f"Silhouette Score para el modelo de K-means: {silhouette_score}")

# Extraer centros de los clústeres
centers = kmeans_model.clusterCenters()
print("Centros de los clústeres:")
for center in centers:
    print(center)

# Paso 4: Visualización de los clústeres en el espacio de los componentes␣
 ↪principales
kmeans_predictions_pd = kmeans_predictions.select("PC1", "PC2", "prediction").
 ↪toPandas()

plt.figure(figsize=(10, 7))
plt.scatter(kmeans_predictions_pd["PC1"], kmeans_predictions_pd["PC2"],␣
 ↪c=kmeans_predictions_pd["prediction"], cmap="viridis", alpha=0.6)
plt.scatter([center[0] for center in centers], [center[1] for center in␣
 ↪centers], c="red", marker="X", s=200, label="Centros de Clúster")
plt.title("Clustering de K-means en el Espacio de Componentes Principales (PC1␣
 ↪y PC2)")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.legend()
plt.show()
```

```
Silhouette Score para el modelo de K-means: 0.8306907611640852
Centros de los clústeres:
[-15627.48196488    546.15031262]
[-15304.14497514    551.95181767]
[-15933.17214686    547.71721083]
```



[ ]:

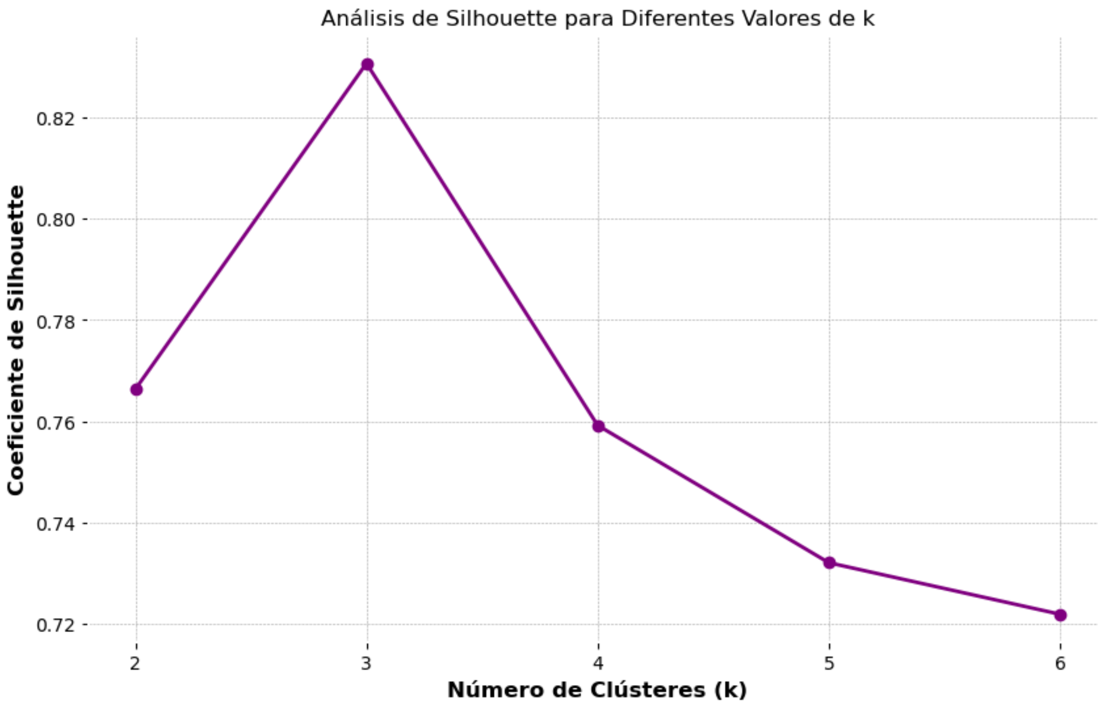
```

```

[105]:

```
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
from pyspark.ml.feature import VectorAssembler
import matplotlib.pyplot as plt

# Verificar y preparar las columnas PCA para clustering
if "features" in pca_df.columns:
    pca_df = pca_df.drop("features")

# Seleccionar las características para el clustering (PC1 y PC2)
feature_cols = ["PC1", "PC2"]

# VectorAssembler para agrupar PC1 y PC2 en una columna de características para␣
↪el modelo K-means
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
pca_df = assembler.transform(pca_df)

# Configuración de valores de k y cálculo del coeficiente de Silhouette
silhouette_scores = []
```

```python
ks = range(2, 7)  # Rango de valores de k
for k in ks:
    kmeans = KMeans(featuresCol="features", k=k, seed=1)

    # Ajuste del modelo y predicción
    model = kmeans.fit(pca_df)
    predictions = model.transform(pca_df)

    # Evaluación de la calidad del agrupamiento con el coeficiente de Silhouette
    evaluator = ClusteringEvaluator(featuresCol="features",␣
 ↪predictionCol="prediction", metricName="silhouette")
    silhouette_score = evaluator.evaluate(predictions)
    silhouette_scores.append(silhouette_score)
    print(f"Para k = {k}, Coeficiente de Silhouette = {silhouette_score}")

# Visualización del coeficiente de Silhouette para cada valor de k
plt.figure(figsize=(10, 6))
plt.plot(ks, silhouette_scores, marker='o', color='purple')
plt.xlabel("Número de Clústeres (k)")
plt.ylabel("Coeficiente de Silhouette")
plt.title("Análisis de Silhouette para Diferentes Valores de k")
plt.xticks(ks)
plt.grid(True)
plt.show()
```

```
Para k = 2, Coeficiente de Silhouette = 0.7663304812583079
Para k = 3, Coeficiente de Silhouette = 0.8306907611640852
Para k = 4, Coeficiente de Silhouette = 0.759221463665402
Para k = 5, Coeficiente de Silhouette = 0.7320900136797283
Para k = 6, Coeficiente de Silhouette = 0.7219122752135139
```

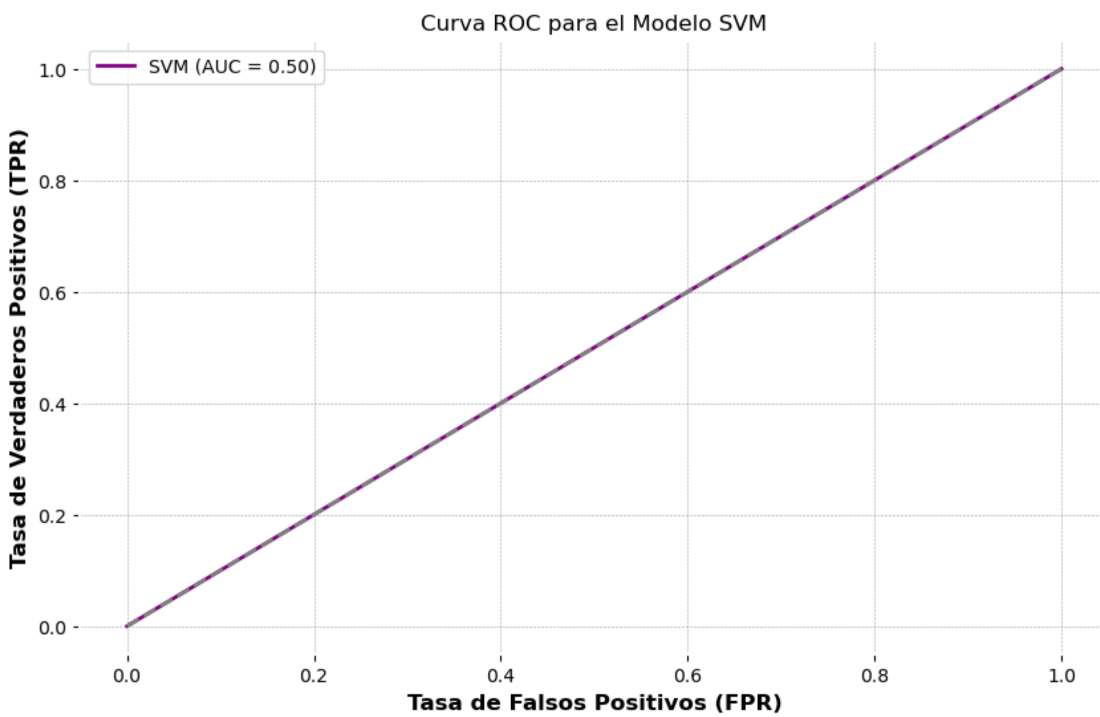


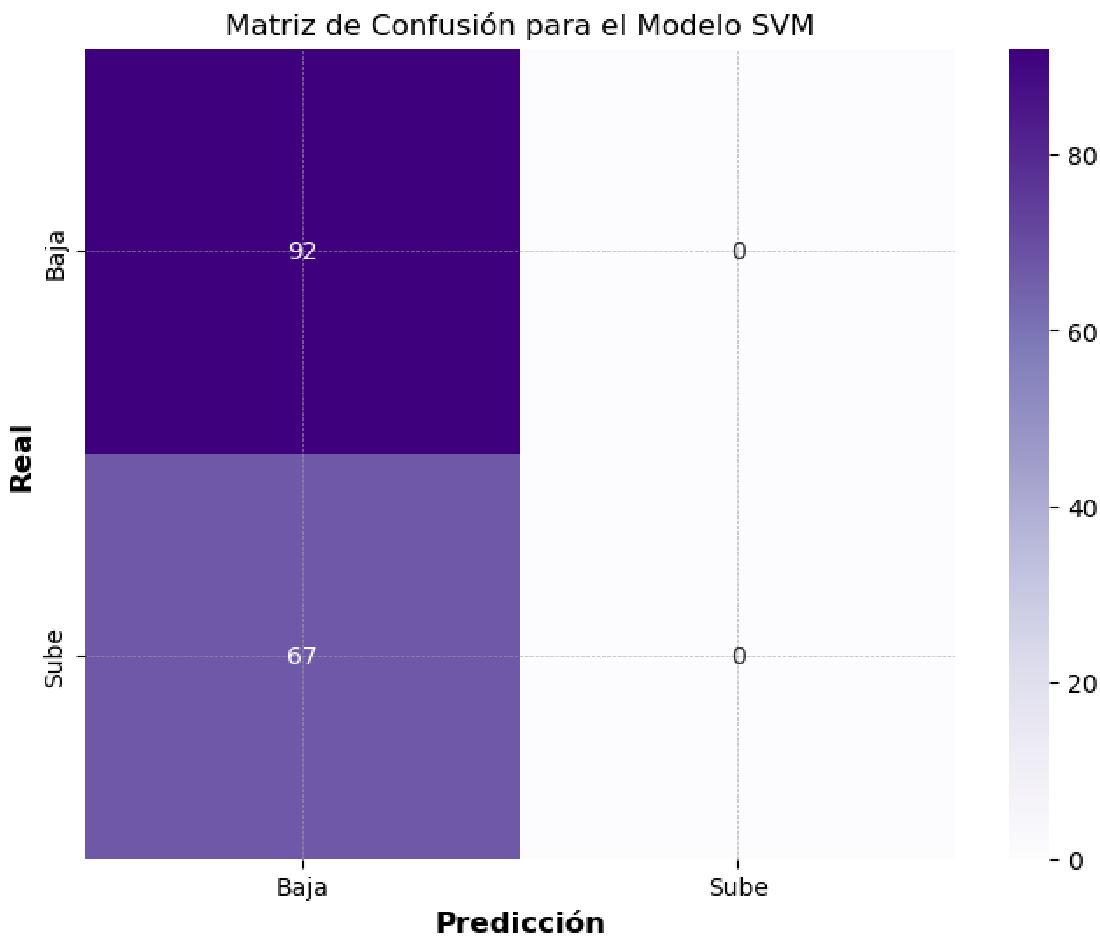
[106]:
```
from pyspark.ml.classification import LinearSVC
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.feature import VectorAssembler
from pyspark.sql import functions as F
from pyspark.sql.window import Window

# Preparar los datos
if "features" in combined_df_filtered.columns:
    combined_df_filtered = combined_df_filtered.drop("features")

# Seleccionar características y definir la etiqueta de clasificación binaria
feature_cols = ["lagged_close_WALMEX", "lagged_close_SORIANAB",␣
↪"lagged_close_FEMSAUBD", "lagged_close_LACOMERUBC", "lagged_close_CHDRAUIB",
                "lagged_close_WMT", "lagged_close_BIMBOA", "lagged_close_COST",␣
↪"lagged_close_ALSEA"]

# Crear una ventana para obtener el valor del próximo precio
window_spec = Window.orderBy("minute_start")

# Usar `lead` para obtener el próximo valor de `close_WALMEX` y determinar si␣
↪sube o baja
combined_df_filtered = combined_df_filtered.withColumn("next_close_WALMEX", F.
↪lead("close_WALMEX").over(window_spec))
combined_df_filtered = combined_df_filtered.withColumn(
```

```
    "price_direction", F.when(F.col("next_close_WALMEX") > F.
  ↪col("close_WALMEX"), 1).otherwise(0)
).drop("next_close_WALMEX")

# VectorAssembler para transformar las características en un vector para el␣
  ↪modelo SVM
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
combined_df_filtered = assembler.transform(combined_df_filtered)

# Dividir los datos en conjunto de entrenamiento y prueba
train, test = combined_df_filtered.randomSplit([0.8, 0.2], seed=747)

# Configuración y entrenamiento del modelo Linear SVC
svm = LinearSVC(featuresCol="features", labelCol="price_direction",␣
  ↪predictionCol="prediction", maxIter=10, regParam=0.1)
svm_model = svm.fit(train)
svm_predictions = svm_model.transform(test)

# Evaluación del modelo SVM
evaluator = BinaryClassificationEvaluator(labelCol="price_direction",␣
  ↪rawPredictionCol="prediction", metricName="areaUnderROC")
svm_auc = evaluator.evaluate(svm_predictions)
print(f"Área bajo la curva ROC para el modelo SVM: {svm_auc}")

# Obtener precisión adicional
accuracy = svm_predictions.filter(svm_predictions.price_direction ==␣
  ↪svm_predictions.prediction).count() / float(svm_predictions.count())
print(f"Precisión del modelo SVM: {accuracy}")

# Visualización de la curva ROC y matriz de confusión
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import roc_curve, confusion_matrix

# Curva ROC
svm_roc_df = svm_predictions.select("price_direction", "prediction").toPandas()
fpr, tpr, thresholds = roc_curve(svm_roc_df["price_direction"],␣
  ↪svm_roc_df["prediction"])
plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, label=f'SVM (AUC = {svm_auc:.2f})', color="purple")
plt.plot([0, 1], [0, 1], color="gray", linestyle="--")
plt.xlabel("Tasa de Falsos Positivos (FPR)")
plt.ylabel("Tasa de Verdaderos Positivos (TPR)")
plt.title("Curva ROC para el Modelo SVM")
plt.legend(loc="best")
plt.show()
```

```python
# Matriz de Confusión
conf_matrix = confusion_matrix(svm_roc_df["price_direction"],␣
↪svm_roc_df["prediction"])
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Purples",␣
↪xticklabels=["Baja", "Sube"], yticklabels=["Baja", "Sube"])
plt.xlabel("Predicción")
plt.ylabel("Real")
plt.title("Matriz de Confusión para el Modelo SVM")
plt.show()
```

```
Área bajo la curva ROC para el modelo SVM: 0.5
Precisión del modelo SVM: 0.5786163522012578
```





[107]:
```
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.sql import functions as F

if "features" in combined_df_filtered.columns:
    combined_df_filtered = combined_df_filtered.drop("features")

# Crear una columna binaria 'price_direction' como etiqueta de clasificación
# price_direction = 1 si el precio sube en la siguiente fila, de lo contrario 0
combined_df_filtered = combined_df_filtered.withColumn(
    "price_direction",
    F.when(F.lead("close_WALMEX").over(Window.orderBy("minute_start")) > F.
 ↪col("close_WALMEX"), 1).otherwise(0)
)
```

```python
# Eliminar filas con valores nulos en las columnas de características y en la␣
↪etiqueta
combined_df_filtered = combined_df_filtered.
↪dropna(subset=["lagged_close_WALMEX", "lagged_close_SORIANAB",
                                                                        ␣
↪"lagged_close_FEMSAUBD", "lagged_close_LACOMERUBC",
                                                                        ␣
↪"lagged_close_CHDRAUIB", "lagged_close_WMT",
                                                                        ␣
↪"lagged_close_BIMBOA", "lagged_close_COST", "lagged_close_ALSEA",
                                                            "price_direction"])

# Seleccionar las columnas de características
feature_cols = ["lagged_close_WALMEX", "lagged_close_SORIANAB",␣
↪"lagged_close_FEMSAUBD",
                "lagged_close_LACOMERUBC", "lagged_close_CHDRAUIB",␣
↪"lagged_close_WMT",
                "lagged_close_BIMBOA", "lagged_close_COST",␣
↪"lagged_close_ALSEA"]

# Ensamblar las características en una columna de vector
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
data = assembler.transform(combined_df_filtered)

# Dividir los datos en conjunto de entrenamiento y prueba
train, test = data.randomSplit([0.8, 0.2], seed=42)

# Crear el modelo de árbol de decisión
dt = DecisionTreeClassifier(featuresCol="features", labelCol="price_direction",␣
↪predictionCol="prediction")
dt_model = dt.fit(train)

# Hacer predicciones en el conjunto de prueba
predictions = dt_model.transform(test)

# Evaluar el modelo con el área bajo la curva ROC
evaluator = BinaryClassificationEvaluator(labelCol="price_direction",␣
↪rawPredictionCol="rawPrediction", metricName="areaUnderROC")
auc = evaluator.evaluate(predictions)
print(f"Área bajo la curva ROC para el modelo de Árbol de Decisión: {auc}")

# Ver las primeras predicciones
predictions.select("minute_start", "close_WALMEX", "price_direction",␣
↪"prediction", "probability").show(5)
```

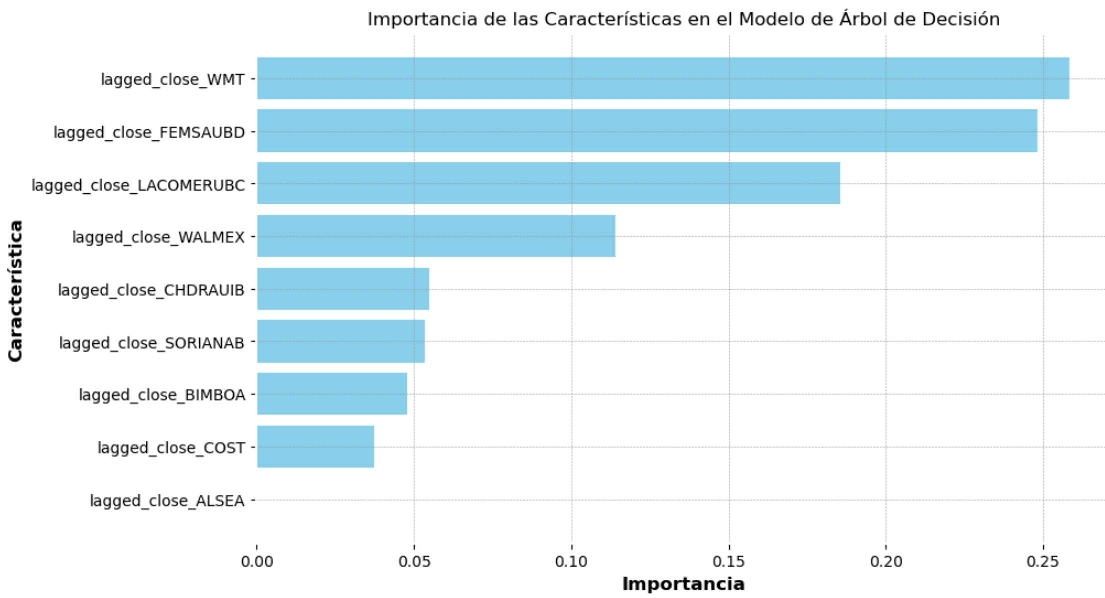
```
Área bajo la curva ROC para el modelo de Árbol de Decisión: 0.40429573348918757
+-------------------+------------+---------------+----------+------------------
```

```
-+
|       minute_start|close_WALMEX|price_direction|prediction|
probability|
+-------------------+------------+---------------+----------+-------------------
-+
|2024-06-24 08:50:00|       61.63|              1|
0.0|[0.67179487179487…|
|2024-06-24 09:10:00|       61.56|              0|
0.0|[0.59793814432989…|
|2024-06-24 09:20:00|       61.41|              1|
0.0|[0.59793814432989…|
|2024-06-24 09:45:00|       61.47|              0|
1.0|[0.33333333333333…|
|2024-06-24 10:15:00|       61.23|              0|
0.0|[0.59793814432989…|
+-------------------+------------+---------------+----------+-------------------
-+
only showing top 5 rows
```

[108]:

```python
import matplotlib.pyplot as plt
import pandas as pd

# Obtener la importancia de las características del modelo
feature_importances = dt_model.featureImportances.toArray()

# Crear un DataFrame para organizar la información
feature_importance_df = pd.DataFrame({
    "Feature": feature_cols,
    "Importance": feature_importances
}).sort_values(by="Importance", ascending=False)

# Mostrar la tabla de importancia
print(feature_importance_df)

# Visualizar la importancia de las características con una gráfica de barras
plt.figure(figsize=(10, 6))
plt.barh(feature_importance_df["Feature"], feature_importance_df["Importance"],␣
 ↪color="skyblue")
plt.xlabel("Importancia")
plt.ylabel("Característica")
plt.title("Importancia de las Características en el Modelo de Árbol de␣
 ↪Decisión")
plt.gca().invert_yaxis()  # Invertir el eje y para que las características más␣
 ↪importantes estén en la parte superior
plt.show()
```

	Feature	Importance
5	lagged_close_WMT	0.258562
2	lagged_close_FEMSAUBD	0.248375
3	lagged_close_LACOMERUBC	0.185542
0	lagged_close_WALMEX	0.114079
4	lagged_close_CHDRAUIB	0.054925
1	lagged_close_SORIANAB	0.053290
6	lagged_close_BIMBOA	0.048020
7	lagged_close_COST	0.037207
8	lagged_close_ALSEA	0.000000



[ ]:

[ ]:

[ ]: