

TEAM : 3  
SHUBHAM TALELE  
PRATHAMESH GAVALI  
MUSKAN CHAUHAN  
JAI JUNEJA  
MRUNALI GAIKWAD  
PANKAJ CHAVAN  
SHUBHAM APASHINGKAR

## **Project Title: Stock Market Data Analysis and Trend Forecasting using PySpark**

### **Introduction :**

### **Project Objectives :**

The primary objective of this project is to analyze the stock performance of Cipla Limited, a major pharmaceutical company, using historical stock market data. The aim is to identify trends, seasonal patterns, and correlations with external indicators to gain actionable insights. This analysis will support informed decision-making regarding potential investment strategies in Cipla's stock.

### **Data Overview**

The dataset comprises historical stock data for Cipla, including the following columns:

**Date:** The trading date of the stock.

**Open:** Opening price of the stock on that day.

**High:** Highest price reached during the trading day.

**Low:** Lowest price reached during the trading day.

**Close:** Closing price of the stock.

**Adj Close:** Adjusted closing price after accounting for splits and dividends.

**Volume:** Number of shares traded on that day.

## Step 1: Project Setup and Data Understanding

### Phase 1: Project Setup and Data Understanding

```
from pyspark.sql import SparkSession
```

```
>>> rdd = spark.sparkContext.textFile("/user/cdaccomcluster0116/CIPLA.csv")
>>> clean_rdd = rdd.zipWithIndex().filter(lambda row: row[1] not in (1, 2)).keys()
>> clean_rdd.saveAsTextFile("/user/cdaccomcluster0116/CIPLA_cleaned_temp")
>>> df = spark.read.csv("/user/cdaccomcluster0116/CIPLA_cleaned_temp", header=False,
inferSchema=True)
>>> columns = ['Price_Ticker_Date', 'Adj_Close', 'Close', 'High', 'Low', 'Open', 'Volume']
>>> df.show(5)
```

```
Some of Firefox's security features may offer less protection on your current operating system. How to fix this issue Don't show again
st(Adj_Close#1441 as float) AS Adj_Close#1490, cast(Close#1442 as float) AS Close#1491, cast(High#1443 as float) AS High#1492,
cast(Low#1444 as float) AS Low#1493, cast(Open#1445 as float) AS Open#1494, cast(Volume#1446 as int) AS Volume#1495
+-- Project [ c0#1426 AS Price_Ticker_Date#1440, _c1#1427 AS Adj_Close#1441, _c2#1428 AS Close#1442, _c3#1429 AS High#1443, _c4#1430 AS Low#1444, _c5#1431 AS Open#1445, _c6#1432 AS Volume#1446]
+- Relation[_c0#1426, _c1#1427, _c2#1428, _c3#1429, _c4#1430, _c5#1431, _c6#1432] csv
>>> df = df.select(*, col("Price_Ticker_Date").cast("timestamp"), col("Adj_Close").cast("float"), col("Close").cast("float"), col("High").cast("float"), col("Low").cast("float"), col("Open").cast("float"), col("Volume").cast("int"))
>>> df.createOrReplaceTempView("ciplastock")
>>> df.printSchema()
root
|-- Price_Ticker_Date: timestamp (nullable = true)
|-- Adj_Close: float (nullable = true)
|-- Close: float (nullable = true)
|-- High: float (nullable = true)
|-- Low: float (nullable = true)
|-- Open: float (nullable = true)
|-- Volume: integer (nullable = true)
>>> df.show(5)
+-----+-----+-----+-----+-----+
| Price_Ticker_Date|Adj_Close|Close|High|Low|Open|Volume|
+-----+-----+-----+-----+-----+
|2009-06-16 00:00:00|[236.80011|260.75|[263.4|253.0|254.9|1836688|
|2009-06-17 00:00:00|[233.75775|257.4|[262.3|252.2|262.3|1315892|
|2009-06-18 00:00:00|[233.21289|256.8|[263.2|253.2|259.7|2275192|
|2009-06-19 00:00:00|[241.06837|265.45|[267.1|258.0|258.0|1535067|
|2009-06-22 00:00:00|[237.57205|261.6|[269.5|255.0|267.0|1197637|
+-----+-----+-----+-----+
only showing top 5 rows
>>>
```

## 1.3 Basic Data Exploration

### a) Check the Schema

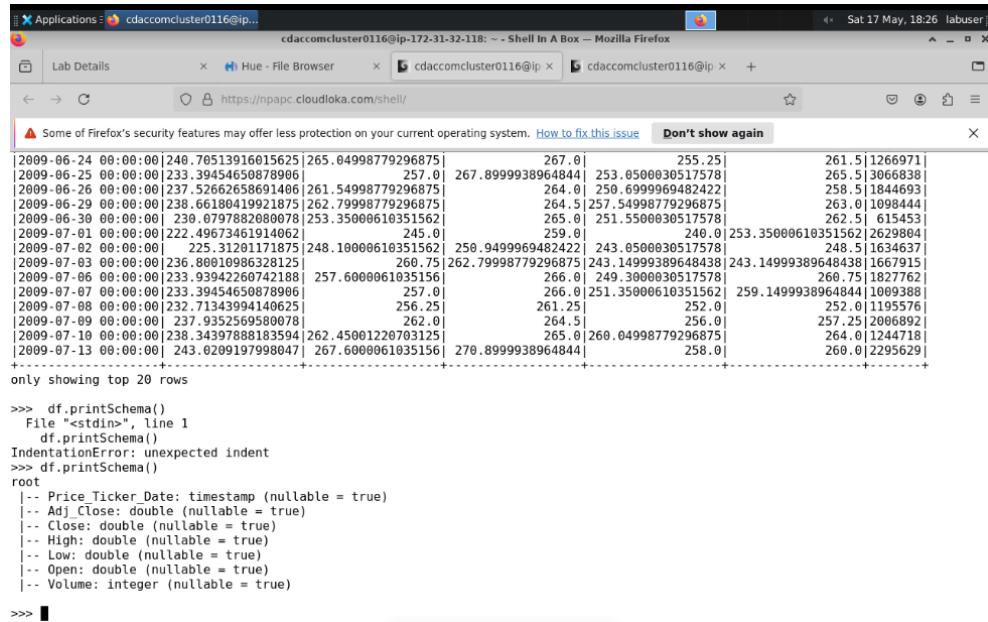
```
df.printSchema()
```

```
>>> df.printSchema()
```

```
root
```

```
-- Price_Ticker_Date: timestamp (nullable = true)
-- Adj_Close: double (nullable = true)
-- Close: double (nullable = true)
-- High: double (nullable = true)
-- Low: double (nullable = true)
```

|-- Open: double (nullable = true)  
|-- Volume: integer (nullable = true)



```

|-- Open: double (nullable = true)
|-- Volume: integer (nullable = true)

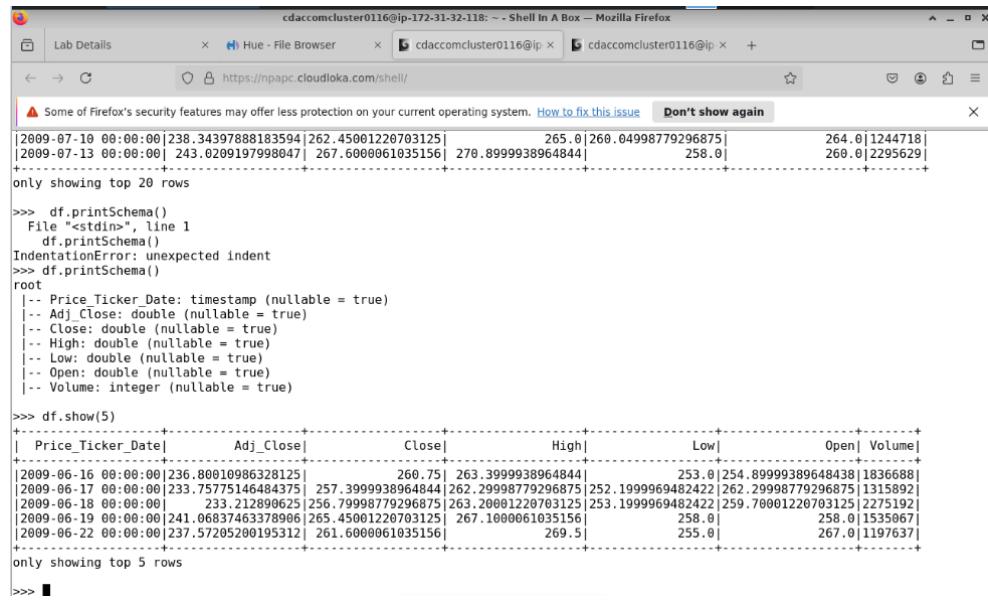
+-----+
| 2009-06-24 00:00:00 | 248.70513916015625 | [265.04998779296875] | 267.0 | 255.25 | 261.5 | 1266971 | |
| 2009-06-25 00:00:00 | 233.3945465087896 | 257.0 | 267.8999938964844 | 253.0500030517578 | 265.5 | 3066838 |
| 2009-06-26 00:00:00 | 237.52662658691466 | [261.54998779296875] | 264.0 | 250.699969482422 | 258.5 | 1844693 |
| 2009-06-29 00:00:00 | 238.66180419921875 | 262.79998779296875 | 264.5 | 257.54998779296875 | 263.0 | 1098444 |
| 2009-06-30 00:00:00 | 238.6797882080078 | 253.35000610351562 | 265.0 | 251.5500030517578 | 262.5 | 615453 |
| 2009-07-01 00:00:00 | 222.4967346194062 | 245.0 | 259.0 | 240.0 | 253.35000610351562 | 262.9884 |
| 2009-07-02 00:00:00 | 225.31201171875 | 248.10000610351562 | 250.9499969482422 | 243.0500030517578 | 248.5 | 1634637 |
| 2009-07-03 00:00:00 | 236.80010986328125 | 260.75 | 262.79998779296875 | 243.14999389648438 | 243.14999389648438 | 1667915 |
| 2009-07-07 00:00:00 | 233.93942260742188 | 257.6000061035156 | 266.0 | 249.3000030517578 | 266.5 | 1827762 |
| 2009-07-29 00:00:00 | 233.3945465087896 | 257.0 | 266.0 | 251.35000610351562 | 259.1499938964844 | 1093988 |
| 2009-07-08 00:00:00 | 232.71343994140625 | 256.25 | 261.25 | 252.0 | 252.0 | 1195576 |
| 2009-07-09 00:00:00 | 237.9352569580078 | 262.0 | 264.5 | 256.0 | 256.0 | 257.25 | 2066892 |
| 2009-07-10 00:00:00 | 238.3439788183594 | 262.45001220703125 | 265.0 | 260.04998779296875 | 264.0 | 1244718 |
| 2009-07-13 00:00:00 | 243.020917998047 | 267.6000061035156 | 270.8999938964844 | 258.0 | 260.0 | 2295629 |
+-----+
only showing top 20 rows

>>> df.printSchema()
File "<stdin>", line 1
df.printSchema()
IndentationError: unexpected indent
>>> df.printSchema()
root
|-- Price_Ticker_Date: timestamp (nullable = true)
|-- Adj_Close: double (nullable = true)
|-- Close: double (nullable = true)
|-- High: double (nullable = true)
|-- Low: double (nullable = true)
|-- Open: double (nullable = true)
|-- Volume: integer (nullable = true)

>>> 
```

## b) Show Initial Rows

df.show(5)



```

+-----+
| 2009-07-10 00:00:00 | 238.3439788183594 | [262.45001220703125] | 265.0 | 260.04998779296875 | 264.0 | 1244718 |
| 2009-07-13 00:00:00 | 243.020917998047 | 267.6000061035156 | 270.8999938964844 | 258.0 | 260.0 | 2295629 |
+-----+
only showing top 20 rows

>>> df.printSchema()
File "<stdin>", line 1
df.printSchema()
IndentationError: unexpected indent
>>> df.printSchema()
root
|-- Price_Ticker_Date: timestamp (nullable = true)
|-- Adj_Close: double (nullable = true)
|-- Close: double (nullable = true)
|-- High: double (nullable = true)
|-- Low: double (nullable = true)
|-- Open: double (nullable = true)
|-- Volume: integer (nullable = true)

>>> df.show(5)
+-----+
| Price_Ticker_Date | Adj_Close | Close | High | Low | Open | Volume |
+-----+
| 2009-06-16 00:00:00 | 236.80010986328125 | 260.75 | 263.3999938964844 | 253.0 | 254.89999389648438 | 1836688 |
| 2009-06-17 00:00:00 | 233.75775146484375 | 257.3999938964844 | 262.29998779296875 | 252.1999969482422 | 262.29998779296875 | 1315892 |
| 2009-06-18 00:00:00 | 233.212899625 | 256.79998779296875 | 263.20001220703125 | 253.1999969482422 | 259.70001220703125 | 2275192 |
| 2009-06-19 00:00:00 | 241.06837463378906 | 265.45001220703125 | 267.1000061035156 | 258.0 | 258.0 | 1535067 |
| 2009-06-22 00:00:00 | 237.57205200195312 | 261.6000061035156 | 269.5 | 255.0 | 267.0 | 1197637 |
+-----+
only showing top 5 rows

>>> 
```

### c) Summary Statistics

df.describe().show()

```

Sat 17 May, 18:29 labuser

+-----+-----+-----+-----+-----+-----+
| 2009-06-16 00:00:00 | [236.80010986328125] | 260.75 | 263.3999938964844 | [253.0 | 254.89999389648438 | 1836688 | |
| 2009-06-17 00:00:00 | [233.75775146484375] | 257.3999938964844 | [262.29998779296875 | 252.1999969482422 | 262.29998779296875 | 1315892 |
| 2009-06-18 00:00:00 | [233.212890625] | 238.79998779296875 | [256.79998779296875 | 263.20001220703125 | 253.1999969482422 | 259.70001220703125 | 2275192 |
| 2009-06-19 00:00:00 | [241.06837463378966] | 265.45001220703125 | 267.1000061035156 | 258.0 | 258.0 | 258.0 | 1535067 |
| 2009-06-22 00:00:00 | [237.57205200195312] | 261.6000061035156 | 269.5 | 255.0 | 267.0 | 267.0 | 1197637 |
+-----+-----+-----+-----+-----+-----+
only showing top 5 rows

+-----+-----+-----+-----+-----+-----+
| Price Ticker Date | [Adj Close] | Close | High | Low | Open | Volume |
+-----+-----+-----+-----+-----+-----+
| 2009-06-16 00:00:00 | [236.80010986328125] | 260.75 | 263.3999938964844 | [253.0 | 254.89999389648438 | 1836688 | |
| 2009-06-17 00:00:00 | [233.75775146484375] | 257.3999938964844 | [262.29998779296875 | 252.1999969482422 | 262.29998779296875 | 1315892 |
| 2009-06-18 00:00:00 | [233.212890625] | 238.79998779296875 | [256.79998779296875 | 263.20001220703125 | 253.1999969482422 | 259.70001220703125 | 2275192 |
| 2009-06-19 00:00:00 | [241.06837463378966] | 265.45001220703125 | 267.1000061035156 | 258.0 | 258.0 | 258.0 | 1535067 |
| 2009-06-22 00:00:00 | [237.57205200195312] | 261.6000061035156 | 269.5 | 255.0 | 267.0 | 267.0 | 1197637 |
+-----+-----+-----+-----+-----+-----+
only showing top 5 rows

+-----+-----+-----+-----+-----+-----+
| summary | Adj Close | Close | High | Low | Open | Volume |
+-----+-----+-----+-----+-----+-----+
| count | 7252 | 7252 | 7252 | 7252 | 7252 | 7252 |
| mean | 364.35484929355675 | 383.5360233171566 | 388.8218855439469 | 378.73519458503796 | 384.1531277442597 | 1661769.8798952814 |
| stddev | 353.5903804099754 | 357.576098235088 | 361.54596336324875 | 353.9001035863223 | 358.04488666959185 | 2141724.6277418886 |
| min | 4.025648593902588 | 4.716800212860107 | 4.716800212860107 | 4.716800212860107 | 4.716800212860107 | 0 |
| max | 1680.5 | 1688.5 | 1702.05048828125 | 1656.699951171875 | 1693.949951171875 | 56895213 |
+-----+-----+-----+-----+-----+-----+
>>> █

```

Step 1: Register DataFrame as SQL Temporary View

a) SQL Query: Check for Null Values in Each Column

null\_counts = spark.sql(""""

SELECT

    SUM(CASE WHEN Price\_Ticker\_Date IS NULL THEN 1 ELSE 0 END) AS Price\_Ticker\_Date\_nulls,

    SUM(CASE WHEN Adj\_Close IS NULL THEN 1 ELSE 0 END) AS Adj\_Close\_nulls,

    SUM(CASE WHEN Close IS NULL THEN 1 ELSE 0 END) AS Close\_nulls,

    SUM(CASE WHEN High IS NULL THEN 1 ELSE 0 END) AS High\_nulls,

    SUM(CASE WHEN Low IS NULL THEN 1 ELSE 0 END) AS Low\_nulls,

    SUM(CASE WHEN Open IS NULL THEN 1 ELSE 0 END) AS Open\_nulls,

    SUM(CASE WHEN Volume IS NULL THEN 1 ELSE 0 END) AS Volume\_nulls

FROM cipla\_data

""")

null\_counts.show()

```

cdacomcluster0116@ip-172-31-32-118: ~ - Shell In A Box — Mozilla Firefox
Lab Details x Hue - File Browser x cdacomcluster0116@ip x cdacomcluster0116@ip x +
https://npapc.cloudioka.com/shell/
⚠ Some of Firefox's security features may offer less protection on your current operating system. How to fix this issue Don't show again X

syntaxError: invalid syntax
>>> null_counts = spark.sql(""" SELECT SUM(CASE WHEN Price_Ticker Date IS NULL THEN 1 ELSE 0 END) AS Price_Ticker_Date_nulls,SU
! (CASE WHEN Adj_Close IS NULL THEN 1 ELSE 0 END) AS Adj_Close_nulls,SUM(CASE WHEN Close IS NULL THEN 1 ELSE 0 END) AS Close_nul
.s,SUM(CASE WHEN High IS NULL THEN 1 ELSE 0 END) AS High_nulls,SUM(CASE WHEN Low IS NULL THEN 1 ELSE 0 END) AS Low_nulls,SUM(CA
!E WHEN Open IS NULL THEN 1 ELSE 0 END) AS Open_nulls,SUM(CASE WHEN Volume IS NULL THEN 1 ELSE 0 END) AS Volume_nulls FROM cipl
!_data """) null_counts.show()
File "<stdin>", line 1
null_counts = spark.sql(""" SELECT SUM(CASE WHEN Price_Ticker Date IS NULL THEN 1 ELSE 0 END) AS Price_Ticker_Date_nulls,SU
! (CASE WHEN Adj_Close IS NULL THEN 1 ELSE 0 END) AS Adj_Close_nulls,SUM(CASE WHEN Close IS NULL THEN 1 ELSE 0 END) AS Close_nul
.s,SUM(CASE WHEN High IS NULL THEN 1 ELSE 0 END) AS High_nulls,SUM(CASE WHEN Low IS NULL THEN 1 ELSE 0 END) AS Low_nulls,SUM(CA
!E WHEN Open IS NULL THEN 1 ELSE 0 END) AS Open_nulls,SUM(CASE WHEN Volume IS NULL THEN 1 ELSE 0 END) AS Volume_nulls FROM cipl
!_data """) null_counts.show()

syntaxError: invalid syntax
>>> null_counts = spark.sql(""" SELECT SUM(CASE WHEN Price_Ticker Date IS NULL THEN 1 ELSE 0 END) AS Price_Ticker_Date_nulls,
    SUM(CASE WHEN Adj_Close IS NULL THEN 1 ELSE 0 END) AS Adj_Close_nulls, SUM(CASE WHEN Close IS NULL THEN 1 ELSE 0 END) AS Clo
    .se_nulls, SUM(CASE WHEN High IS NULL THEN 1 ELSE 0 END) AS High_nulls, SUM(CASE WHEN Low IS NULL THEN 1 ELSE 0 END) AS Lo
    .w_nulls, SUM(CASE WHEN Open IS NULL THEN 1 ELSE 0 END) AS Open_nulls, SUM(CASE WHEN Volume IS NULL THEN 1 ELSE 0 END) AS Vo
    lume_nulls FROM cipla_data """)
>>> null_counts.show()
+-----+-----+-----+-----+-----+-----+
|Price_Ticker_Date_nulls|Adj_Close_nulls|Close_nulls|High_nulls|Low_nulls|Open_nulls|Volume_nulls|
+-----+-----+-----+-----+-----+-----+
|          1|           1|          1|          1|          1|          1|          1|
+-----+-----+-----+-----+-----+-----+
>>>

```

## b) SQL Query: Summary Statistics Using SQL

```

summary_stats = spark.sql("""
SELECT
    COUNT(*) AS total_rows,
    AVG(Adj_Close) AS avg_adj_close,
    STDDEV(Adj_Close) AS stddev_adj_close,
    MIN(Adj_Close) AS min_adj_close,
    MAX(Adj_Close) AS max_adj_close,
    AVG(Volume) AS avg_volume,
    STDDEV(Volume) AS stddev_volume,
    MIN(Volume) AS min_volume,
    MAX(Volume) AS max_volume
FROM cipla_data
""")

summary_stats.show()

```

```

>>> summary_stats.show(truncate = False)
+-----+-----+-----+-----+-----+-----+
|total_rows|avg_adj_close |stddev_adj_close |min_adj_close |max_adj_close|avg_volume      |stddev_volume   |min_volu
me|max_volume|
+-----+-----+-----+-----+-----+-----+
|7253     |364.35484929355675|353.5903804099754|4.025648593902588|1680.5      |1661769.8798952014|2141724.6277418886|0
|56895213 |
+-----+-----+-----+-----+-----+-----+
>>> summary_stats.show(truncate = True)

```

### 1.5a) Check for Duplicates

```

total_count = df.count()

distinct_count = df.dropDuplicates().count()

```

```

print(f"Total rows: {total_count}")

print(f"Distinct rows: {distinct_count}")

print(f"Duplicate rows: {total_count - distinct_count}")

```

date	time	value1	value2	value3	value4	value5	value6
2009-06-16	00:00:00	236.80010986328125	260.75	263.3999938964844	253.0	254.89999389648438	1836688
2009-06-17	00:00:00	233.75775146484375	257.3999938964844	262.29998779296875	252.1999969482422	262.29998779296875	1315892
2009-06-18	00:00:00	233.212896025	256.79998779296875	263.20001220703125	253.1999969482422	259.70001220703125	2275192
2009-06-19	00:00:00	241.06837463378906	265.45001220703125	267.1000061035156	258.0	258.0	1535067
2009-06-22	00:00:00	237.57205200195312	261.6000061035156	269.5	255.0	267.0	1197637
2009-06-23	00:00:00	235.48324584960938	259.29998779296875	265.0	256.5	259.0	1149328
2009-06-24	00:00:00	240.76513916015625	265.04998779296875	267.0	255.25	261.5	1266971
2009-06-25	00:00:00	233.39454650878906	257.0	267.8999938964844	253.8500030517578	265.5	3066838
2009-06-26	00:00:00	237.52662658691466	261.54998779296875	264.0	250.6999969482422	258.5	1844693
2009-06-29	00:00:00	238.66180419921875	262.79998779296875	264.5	257.54998779296875	263.0	1098444
2009-06-30	00:00:00	238.6797882800078	253.35000610351562	265.0	251.5500030517578	262.5	615453
2009-07-01	00:00:00	222.49673461914062	245.0	259.0	240.0	253.35000610351562	2629884
2009-07-02	00:00:00	225.3120171875	248.10000610351562	250.9499969482422	243.0500030517578	248.5	1634637
2009-07-03	00:00:00	236.80010986328125	260.75	262.79998779296875	243.14999389648438	243.14999389648438	1667915
2009-07-06	00:00:00	233.9394260742188	257.6000061035156	266.0	249.3000030517578	260.75	1827762
2009-07-07	00:00:00	233.39454650878906	257.0	266.0	251.35000610351562	259.1499938964844	1099388
2009-07-08	00:00:00	232.71343094140625	256.25	261.25	252.0	252.0	1105576
2009-07-09	00:00:00	237.9352569580078	262.0	264.5	256.0	257.25	2006892
2009-07-10	00:00:00	238.34397888183594	262.45001220703125	265.0	260.04998779296875	264.0	1244718
2009-07-13	00:00:00	243.0209197998047	267.6000061035156	270.8999938964844	258.0	260.0	2295629

only showing top 20 rows

```

>>> total_count = df.count()
>>> distinct_count = df.dropDuplicates().count()
>>> print(f"Total rows: {total_count}")
Total rows: 7253
>>> print(f"Distinct rows: {distinct_count}")
Distinct rows: 7253
>>> print(f"Duplicate rows: {total_count - distinct_count}")
Duplicate rows: 0
>>>

```

```
1.5 b) Date Consistency Check (check for gaps in dates)
from pyspark.sql.functions import col, to_date, lag, datediff
from pyspark.sql.window import Window

# Step 1: Extract and sort unique dates
date_df = df.select(to_date(col("Price_Ticker_Date")).alias("date")).distinct().sort("date")

# Step 2: Create a lag column to compare each date with the previous one
window_spec = Window.orderBy("date")
date_diff_df = date_df.withColumn("prev_date", lag("date").over(window_spec)).withColumn("gap",
datediff(col("date"), col("prev_date")))

# Step 3: Filter for gaps greater than 1 day
date_gaps_df = date_diff_df.filter(col("gap") > 1)

print("Dates with gaps:")
date_gaps_df.show()

# Optional: just view first 10 dates
print("First 10 dates:")
date_df.show(10)
```

```

>>> from pyspark.sql.window import Window
>>> date_df = df.select(to_date(col("Price_Ticker_Date")).alias("date")).distinct().sort("date")
>>> window_spec = Window.orderBy("date")
>>> date_diff_df = date_df.withColumn("prev_date", lag("date").over(window_spec)).withColumn("gap", datediff(col("date"), col("prev_date")))
>>> date_gaps_df = date_diff_df.filter(col("gap") > 1)
>>> print("Dates with gaps:")
Dates with gaps:
>>> date_gaps_df.show()
25/05/17 19:02:05 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
+-----+-----+
| date| prev_date|gap|
+-----+-----+
|1996-01-08|1996-01-05| 3|
|1996-01-15|1996-01-12| 3|
|1996-01-22|1996-01-19| 3|
|1996-01-29|1996-01-26| 3|
|1996-02-05|1996-02-02| 3|
|1996-02-12|1996-02-09| 3|
|1996-02-19|1996-02-16| 3|
|1996-02-26|1996-02-23| 3|
|1996-03-04|1996-03-01| 3|
|1996-03-11|1996-03-08| 3|
|1996-03-18|1996-03-15| 3|
|1996-03-25|1996-03-22| 3|
|1996-04-01|1996-03-29| 3|
|1996-04-08|1996-04-05| 3|
|1996-04-15|1996-04-12| 3|
|1996-04-22|1996-04-19| 3|
|1996-04-29|1996-04-26| 3|
|1996-05-06|1996-05-03| 3|

```

## Phase 2: Data Cleaning and Preparation

### 1 Remove Rows with Missing Data

```
df_cleaned = df.dropna()
```

### 2 Ensure Date Column is Properly Formatted

```
>>> df_cleaned.printSchema()
```

```
root
```

```

|-- Price_Ticker_Date: timestamp (nullable = true)
|-- Adj_Close: double (nullable = true)
|-- Close: double (nullable = true)
|-- High: double (nullable = true)
|-- Low: double (nullable = true)
|-- Open: double (nullable = true)
|-- Volume: integer (nullable = true)
|-- Date: date (nullable = true)
```

Calculate Daily Returns :

```
from pyspark.sql.window import Window
```

```

from pyspark.sql.functions import col, lag, round

# Window to sort data by date
window_spec = Window.orderBy("Date")

# Add previous day's close price and compute daily return
df_returns = df_cleaned.withColumn("Prev_Close", lag("Close").over(window_spec)) \
    .withColumn("Daily_Return", round(((col("Close") - col("Prev_Close")) / col("Prev_Close")) * 100, 2))

```

```

+-----+-----+-----+-----+-----+-----+-----+
| Price | Ticker | Date | Adj Close | Close | High | Low | Open | Volume | D |
+-----+-----+-----+-----+-----+-----+-----+
| 1996-01-01 00:00:00 | 7.302499294281006 | 8.831999778747559 | 8.83333015441895 | 8.773332595825195 | 8.778665542602539 | 208125 | 1996-01-01 | null | null |
| 1996-01-02 00:00:00 | 7.209896087646484 | 8.720000267028809 | 9.013333320617676 | 8.39999618530273 | 8.831999778747559 | 1623750 | 1996-01-02 | 18.831999778747559 | -1.27 |
| 1996-01-03 00:00:00 | 7.159183025360107 | 8.658665657043457 | 8.880000114440918 | 8.626666069030762 | 8.720000267028809 | 721875 | 1996-01-03 | 18.720000267028809 | -0.71 |
| 1996-01-04 00:00:00 | 7.0842180252075195 | 8.567999839782715 | 8.653332710266113 | 8.39999618530273 | 8.658665657043457 | 620625 | 1996-01-04 | 18.658665657043457 | -1.05 |
| 1996-01-05 00:00:00 | 7.066579818725586 | 8.546666145324707 | 8.640000343322754 | 8.50666618347168 | 8.567999839782715 | 350625 | 1996-01-05 | 18.567999839782715 | -0.25 |
| 1996-01-06 00:00:00 | 6.945310592651367 | 8.39999618530273 | 8.520000457763672 | 8.37332977294922 | 8.546666145324707 | 645000 | 1996-01-06 | 18.546666145324707 | -1.72 |
| 1996-01-07 00:00:00 | 6.964052677154541 | 8.4226655960083 | 8.493332862854004 | 8.295999526977539 | 8.39999618530273 | 903750 | 1996-01-07 | 18.39999618530273 | 0.27 |
| 1996-01-08 00:00:00 | 7.0213799476623535 | 8.491999626159668 | 8.613332748413086 | 8.426666259765625 | 8.4226655960083 | 455625 | 1996-01-08 | 18.4226655960083 | 0.82 |
| 1996-01-09 00:00:00 | 7.09303617477417 | 8.578665733337402 | 8.613332748413086 | 8.486665725708008 | 8.491999626159668 | 110625 | 1996-01-09 | 18.491999626159668 | 1.02 |
| 1996-01-10 00:00:00 | 7.2165093421936035 | 8.727999687194824 | 8.800000190734863 | 8.65999984741211 | 8.800000190734863 | 543750 | 1996-01-10 | 18.578665733337402 | 1.74 |
+-----+-----+-----+-----+-----+-----+-----+
only showing top 10 rows
>>> █

```

### 3 Add 50-Day and 200-Day Moving Averages

```

from pyspark.sql.functions import avg

df_ma = df_returns.withColumn("MA_50", round(avg("Close").over(window_spec.rowsBetween(-49, 0)), 2)).withColumn("MA_200", round(avg("Close").over(window_spec.rowsBetween(-199, 0)), 2))

```

#### Final Output

```

df_ma.select("Date", "Close", "Daily_Return", "MA_50", "MA_200").show(10)

df_ma.write.csv("/user/cdaccomcluster0116/CIPLA_cleaned_final", header=True, mode="overwrite")

```

Sat 17 May, 19:25 labuser:

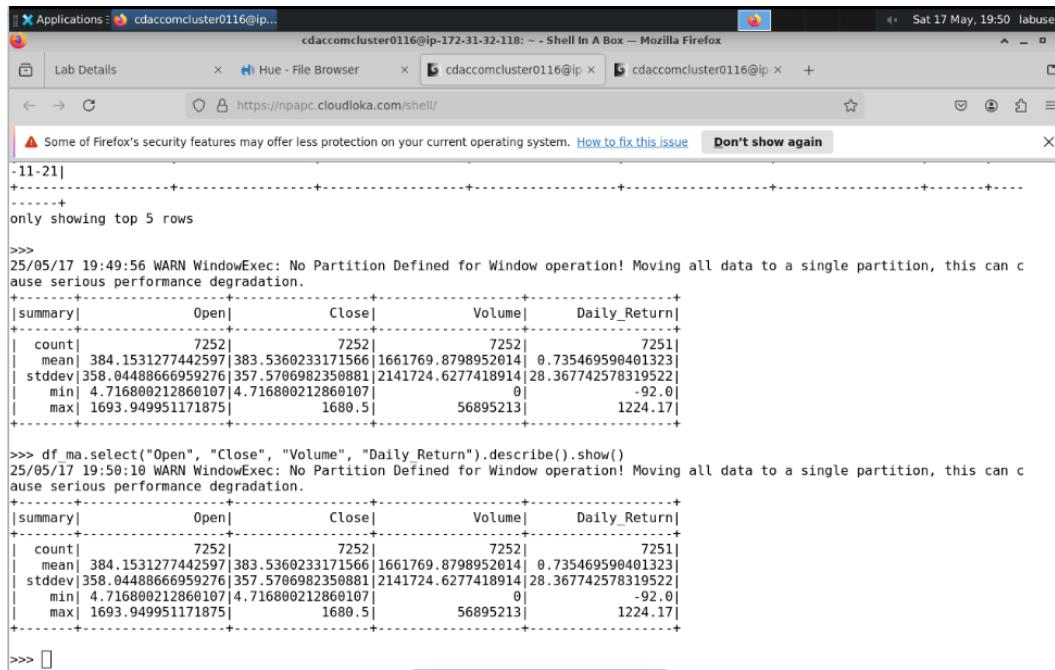
```
cdaccommcluster0116@ip... cdaccommcluster0116@ip... ~ - Shell In A Box - Mozilla Firefox
Lab Details File Browser cdaccommcluster0116@ip... cdaccommcluster0116@ip... + https://nppc.cloudloka.com/shell/ Sat 17 May, 19:25 labuser:
Some of Firefox's security features may offer less protection on your current operating system. How to fix this issue Don't show again

round(avgl("Close").over(window_spec.rowsBetween(-199, 0)), 2))
File "<stdin>", line 1
    df_ma = df.returns.withColumn("MA_50", round(avgl("Close").over(window_spec.rowsBetween(-49, 0)), 2)).withColumn("MA_200", round(avgl("Close").over(window_spec.rowsBetween(-199, 0)), 2)))
SyntaxError: unexpected character after line continuation character
>>> df_ma = df.returns.withColumn("MA_50", round(avgl("Close").over(window_spec.rowsBetween(-49, 0)), 2)).withColumn("MA_200", round(avgl("Close").over(window_spec.rowsBetween(-199, 0)), 2)))
>>> df_ma.select("Date", "Close", "Daily_Return", "MA_50", "MA_200").show(10)
25/05/17 19:24:12 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
25/05/17 19:24:12 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
25/05/17 19:24:12 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
+-----+-----+-----+-----+
| Date | Close|Daily_Return|MA_50|MA_200|
+-----+-----+-----+-----+
|1996-01-01|8.831999778747559| null| 8.83| 8.83|
|1996-01-02|8.720000267628809|-1.27| 8.78| 8.78|
|1996-01-03|8.658665657843457| -0.7| 8.74| 8.74|
|1996-01-04|8.54666614524707| 1.02| 8.71| 8.71|
|1996-01-05|8.54666614524707| -0.25| 8.67| 8.67|
|1996-01-08|8.39999618530273| -1.72| 8.62| 8.62|
|1996-01-09|8.4226655960083| 0.27| 8.59| 8.59|
|1996-01-10|8.491999626159668| 0.82| 8.58| 8.58|
|1996-01-11|8.57866573337492| 1.02| 8.58| 8.58|
|1996-01-12|8.727999687194824| 1.74| 8.59| 8.59|
+-----+
only showing top 10 rows
>>> 
```

## Phase 3: Exploratory Data Analysis (EDA)

### 3.1 Summary Statistics (Basic stats using DataFrame and SQL)

```
df_ma.select("Open", "Close", "Volume", "Daily_Return").describe().show()
```



```
-11-21|  
+-----+-----+-----+-----+  
only showing top 5 rows  
  
>>>  
25/05/17 19:49:56 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.  
+-----+-----+-----+-----+  
|summary| Open| Close| Volume| Daily_Return|  
+-----+-----+-----+-----+  
| count| 7252| 7252| 7252| 7251|  
| mean | 384.1531277442597|383.5360233171566|1661769.8798952014| 0.735469598401323|  
| stddev| 358.04488666959276|357.5706982350881|2141724.6277418914|28.367742578319522|  
| min  | 4.716800212860107|4.716800212860107| 0| -92.0|  
| max  | 1693.949951171875| 1680.5| 56895213| 1224.17|  
+-----+-----+-----+-----+  
  
>>> df_ma.select("Open", "Close", "Volume", "Daily_Return").describe().show()  
25/05/17 19:58:10 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.  
+-----+-----+-----+-----+  
|summary| Open| Close| Volume| Daily_Return|  
+-----+-----+-----+-----+  
| count| 7252| 7252| 7252| 7251|  
| mean | 384.1531277442597|383.5360233171566|1661769.8798952014| 0.735469598401323|  
| stddev| 358.04488666959276|357.5706982350881|2141724.6277418914|28.367742578319522|  
| min  | 4.716800212860107|4.716800212860107| 0| -92.0|  
| max  | 1693.949951171875| 1680.5| 56895213| 1224.17|  
+-----+-----+-----+-----+  
>>> □
```

Register DataFrame as Temp View for SQL queries:

```
df_ma.createOrReplaceTempView("cipla_data")
```

SQL query for summary stats of Close and Volume:

```
summary_stats = spark.sql("""  
SELECT  
    ROUND(AVG(Close), 2) AS avg_close,  
    ROUND(STDDEV(Close), 2) AS stddev_close,  
    MIN(Close) AS min_close,  
    MAX(Close) AS max_close,  
    ROUND(AVG(Volume), 2) AS avg_volume,  
    ROUND(STDDEV(Volume), 2) AS stddev_volume,  
    MIN(Volume) AS min_volume,  
    MAX(Volume) AS max_volume
```

```
FROM cipla_data
```

```
""")
```

```
summary_stats.show()
```

```
count| 7252| 7252| 7252| 7251|
+-----+-----+-----+-----+
| mean| 384.1531277442597|383.5360233171566|1661769.8798952014| 0.735469590401323|
| stddev| 358.04488666959276|357.5706982350881|2141724.6277418914|28.367742578319522|
| min| 4.716880212860107|4.716880212860107|0| -92.0|
| max| 1693.949951171875|1680.5| 56895213| 1224.17|
+-----+-----+-----+-----+  
  
>>> df_ma.select("Open", "Close", "Volume", "Daily_Return").describe().show()  
25/05/17 19:50:18 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.  
+-----+-----+-----+-----+  
|summary| Open| Close| Volume| Daily_Return|  
+-----+-----+-----+-----+  
| count| 7252| 7252| 7252| 7251|  
| mean| 384.1531277442597|383.5360233171566|1661769.8798952014| 0.735469590401323|  
| stddev| 358.04488666959276|357.5706982350881|2141724.6277418914|28.367742578319522|  
| min| 4.716880212860107|4.716880212860107|0| -92.0|  
| max| 1693.949951171875|1680.5| 56895213| 1224.17|  
+-----+-----+-----+-----+  
  
>>> df_ma.createOrReplaceTempView("cipla_data")  
>>> summary_stats = spark.sql(""" SELECT ROUND(AVG(Close), 2) AS avg_close, ROUND(STDDEV(Close), 2) AS stddev_close, MIN(Close) AS min_close, MAX(Close) AS max_close, ROUND(AVG(Volume), 2) AS avg_volume, ROUND(STDDEV(Volume), 2) AS stddev_volume, MIN(Volume) AS min_volume, MAX(Volume) AS max_volume FROM cipla_data """)  
>>> summary_stats.show()  
+-----+-----+-----+-----+-----+-----+  
|avg_close|stddev_close| min_close|max_close|avg_volume|stddev_volume|min_volume|max_volume|  
+-----+-----+-----+-----+-----+-----+-----+  
| 383.54| 357.57|4.716880212860107| 1680.5|1661769.88| 2141724.63| 0| 56895213|  
+-----+-----+-----+-----+-----+-----+-----+
```

### 3.2 Analyze Daily Returns (Mean and Stddev)

```
daily_returns_summary = spark.sql(""""
```

```
SELECT
```

```
ROUND(AVG(Daily_Return), 3) AS avg_daily_return,  
ROUND(STDDEV(Daily_Return), 3) AS stddev_daily_return,  
MIN(Daily_Return) AS min_daily_return,  
MAX(Daily_Return) AS max_daily_return
```

```
FROM cipla_data
```

```
""")
```

```
daily_returns_summary.show()
```

```

Applications : cdaccommcluster0116@ip... cdaccommcluster0116@ip-172-31-32-118: ~ - Shell in A Box - Mozilla Firefox
Lab Details   Hue - File Browser  cdaccommcluster0116@ip... cdaccommcluster0116@ip...
https://nppapc.cloudloka.com/shell/ Sat 17 May, 20:01 labuser
Some of Firefox's security features may offer less protection on your current operating system. How to fix this issue Don't show again X

+-----+-----+-----+-----+
| count| 7252| 7252| 7252| 7251|
| mean | 384.1531277442597| 383.5360233171566| 1661769.8798952014| 0.735469590401323|
| stddev| 358.04488666959276| 357.5706982350881| 2141724.6277418914| 28.367742578319522|
| min  | 4.716800212866107| 4.716800212866107| 0| 92.0|
| max  | 1693.949951171875| 1680.5| 56895213| 1224.17|
+-----+-----+-----+-----+
>>> df_ma.createOrReplaceTempView("cipla_data")
>>> summary_stats = spark.sql(""" SELECT ROUND(AVG(Close), 2) AS avg_close, ROUND(STDDEV(Close), 2) AS stddev_close, MIN(Close) AS min_close, MAX(Close) AS max_close, ROUND(AVG(Volume), 2) AS avg_volume, ROUND(STDDEV(Volume), 2) AS stddev_volume, MIN(Volume) AS min_volume, MAX(Volume) AS max_volume FROM cipla_data """)
>>> summary_stats.show()
+-----+-----+-----+-----+-----+
|avg_close|stddev_close|min_close|max_close|avg_volume|stddev_volume|min_volume|max_volume|
+-----+-----+-----+-----+-----+-----+-----+-----+
| 383.54| 357.57| 4.716800212866107| 1680.5| 1661769.88| 2141724.63| 0| 56895213|
+-----+-----+-----+-----+-----+-----+-----+-----+
>>>
25/05/17 20:00:32 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
+-----+-----+-----+-----+
|avg_daily_return|stddev_daily_return|min_daily_return|max_daily_return|
+-----+-----+-----+-----+
| 0.735| 28.368| 92.0| 1224.17|
+-----+-----+-----+-----+
>>>
```

### 3.3 Trend Analysis with Moving Averages :

**Calculate rolling averages with SQL window functions:**

moving\_avg = spark.sql("""

SELECT

Date,

Close,

AVG(Close) OVER (ORDER BY Date ROWS BETWEEN 49 PRECEDING AND CURRENT ROW)  
AS MA\_50,

AVG(Close) OVER (ORDER BY Date ROWS BETWEEN 199 PRECEDING AND CURRENT ROW) AS MA\_200

FROM cipla\_data

ORDER BY Date

""")

moving\_avg.show(10)

```

25/05/17 20:00:32 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
+-----+-----+-----+
|avg_daily_return|stddev_daily_return|min_daily_return|max_daily_return|
+-----+-----+-----+
|      0.735|       28.368|        -92.0|       1224.17|
+-----+-----+-----+

>>>
>>> moving_avg = spark.sql(""" SELECT Date, Close, AVG(Close) OVER (ORDER BY Date ROWS BETWEEN 49 PRECEDING AND CURRENT ROW) AS MA_50, AVG(Close) OVER (ORDER BY Date ROWS BETWEEN 199 PRECEDING AND CURRENT ROW) AS MA_200 FROM cipla_data ORDER BY Date """)
>>> moving_avg.show(10)
25/05/17 20:02:44 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
+-----+-----+-----+
|     Date|      Close|      MA_50|      MA_200|
+-----+-----+-----+
|1996-01-01|8.831999778747559|8.831999778747559|8.831999778747559|
|1996-01-02|8.728900267628809|8.776000022888184|8.776000022888184|
|1996-01-03|8.658665657643457|8.736888567606607|8.736888567606607|
|1996-01-04|8.567999839782715|8.694666385650635|8.694666385650635|
|1996-01-05|8.546666145324707|8.665066337585449|8.665066337585449|
|1996-01-08|8.399999618530273|8.620888551076254|8.620888551076254|
|1996-01-09|8.4226655960083|8.592570986066546|8.592570986066546|
|1996-01-10|8.491999626159668|8.579999566078186|8.579999566078186|
|1996-01-11|8.57866573337402|8.579851362440321|8.579851362440321|
|1996-01-12|8.727999687194824|8.59466619491577|8.59466619491577|
+-----+-----+-----+
only showing top 10 rows
>>>

```

### 3.4 Volume Analysis :

**Find days with highest volume and corresponding price changes:**

```

high_volume_days = spark.sql("""
SELECT
    Date,
    Volume,
    Close,
    LAG(Close, 1) OVER (ORDER BY Date) AS Prev_Close,
    ROUND(((Close - LAG(Close, 1) OVER (ORDER BY Date)) / LAG(Close, 1) OVER (ORDER BY Date)) * 100, 2) AS Price_Change_Percentage
FROM cipla_data
ORDER BY Volume DESC
LIMIT 10
""")

high_volume_days.show()

```

```

cdacomcluster0116@ip-172-31-32-118: ~ - Shell In A Box - Mozilla Firefox
Lab Details | Hue - File Browser | cdacomcluster0116@ip | cdacomcluster0116@ip | + | Sat 17 May, 20:04 labuser
https://npapc.cloudioka.com/shell/ | Don't show again

Some of Firefox's security features may offer less protection on your current operating system. How to fix this issue

1996-01-04|8.567999839782715|8.694666385650635|8.694666385650635|
1996-01-05|8.546666145324787|8.665066337585449|8.665066337585449|
1996-01-08|8.399999618530273|8.62088851076254|8.62088851076254|
1996-01-09|8.4226655960083|8.592570986066546|8.592570986066546|
1996-01-10|8.491999626159668|8.579999566078186|8.579999566078186|
1996-01-11|8.578665733337482|8.579851362440321|8.579851362440321|
1996-01-12|8.727999687194824|8.59466619491577|8.59466619491577|
only showing top 10 rows

>>> high_volume_days = spark.sql(""" SELECT Date, Volume, Close, LAG(Close, 1) OVER (ORDER BY Date) AS Prev_Close, ROLLING_AVG((Close - LAG(Close, 1) OVER (ORDER BY Date)) / LAG(Close, 1) OVER (ORDER BY Date)) * 100, 2) AS Price_Change_Percentage FROM cipla_data ORDER BY Volume DESC LIMIT 10 """)
>>> high_volume_days.show()
25/05/17 20:04:32 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.

+-----+-----+-----+-----+
| Date|Volume|Close|Prev_Close|Price_Change_Percentage|
+-----+-----+-----+-----+
|2020-08-10|56895213|795.5999755859375|728.6500244140625|9.19|
|2020-09-18|35527325|806.25|751.5|7.29|
|2021-04-12|39141861|902.4000244140625|883.0499877929688|2.19|
|2020-06-22|29312493|655.9500122070312|636.2000122070312|3.1|
|2020-04-09|27207708|579.5999755859375|512.75|13.04|
|2020-05-18|22898114|600.4500122070312|570.2999877929688|5.29|
|2020-11-10|22558440|717.2999877929688|763.2000122070312|-6.01|
|2020-11-09|21178707|763.2000122070312|789.9500122070312|-3.39|
|2020-04-07|20843608|492.25|449.20001220703125|9.58|
|2020-12-14|19785595|789.2999877929688|755.8499755859375|4.43|
+-----+-----+-----+-----+
>>> 

```

### 3.5 Seasonal Patterns Analysis :

**Extract month and calculate average close price and volume by month:**

```

monthly_avg = spark.sql("""
SELECT
MONTH(Date) AS Month,
ROUND(AVG(Close), 2) AS Avg_Close,
ROUND(AVG(Volume), 2) AS Avg_Volume
FROM cipla_data
GROUP BY MONTH(Date)
ORDER BY Month
""")
monthly_avg.show()

```

```

2021-04-12|30141861|902.4000244140625| 883.0499877929688| 2.19|
2020-06-22|29312493|655.9500122070312| 636.2000122070312| 3.1|
2020-04-09|27207708|579.5999755859375| 512.75| 13.04|
2020-05-18|22898114|600.4500122070312| 570.2999877929688| 5.29|
2020-11-10|22558440|717.2999877929688| 763.2000122070312| -6.01|
2020-11-09|21178707|763.2000122070312| 789.9500122070312| -3.39|
2020-04-07|20843608| 492.25|449.20001220703125| 9.58|
2020-12-14|19785595|789.2999877929688| 755.8499755859375| 4.43|
+-----+-----+-----+-----+
>>>
>>> monthly_avg = spark.sql(""" SELECT MONTH(Date) AS Month, ROUND(AVG(Close), 2) AS Avg_Close, ROUND(AVG(Volume), 2) AS Avg_Volume FROM cipla_data GROUP BY MONTH(Date) ORDER BY Month """)
>>> monthly_avg.show()
+----+----+----+
|Month|Avg_Close|Avg_Volume|
+----+----+----+
| 1| 368.69|1330922.15|
| 2| 371.15|1599518.51|
| 3| 357.13|1609869.96|
| 4| 364.66|2034940.75|
| 5| 371.01|1869967.48|
| 6| 377.66|1495427.04|
| 7| 388.37|1393323.37|
| 8| 403.14|1800281.79|
| 9| 417.12|1809566.21|
| 10| 411.15|1793428.71|
| 11| 391.52|1788810.06|
| 12| 380.16|1454816.34|
+----+----+----+
>>> ■

```

### 3.6 Identify High and Low Volatility Periods :

**Find days with largest absolute daily returns (high volatility days):**

```

volatility_days = spark.sql(""""
SELECT
    Date,
    Daily_Return,
    ABS(Daily_Return) AS Abs_Daily_Return
FROM cipla_data
ORDER BY Abs_Daily_Return DESC
LIMIT 10
""")  

volatility_days.show()

```

```

4| 364.66|2034940.75|
5| 371.01|1869967.48|
6| 377.66|1495427.04|
7| 388.37|1393323.37|
8| 403.14|1800281.79|
9| 417.12|1809566.21|
10| 411.15|1793428.71|
11| 391.52|1788810.06|
12| 380.16|1454816.34|
+---+-----+-----+
>>> volatility_days = spark.sql(""" SELECT Date, Daily_Return, ABS(Daily_Return) AS Abs_Daily_Return FROM cipla_data ORDER BY Abs_Daily_Return DESC LIMIT 10 """)
>>> volatility_days.show()
25/05/17 20:08:24 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
+-----+-----+-----+
| Date|Daily_Return|Abs_Daily_Return|
+-----+-----+-----+
|2003-11-27| 1224.17| 1224.17|
|2004-04-27| 1190.79| 1190.79|
|2003-12-26| 1170.27| 1170.27|
|2003-04-15| 1154.32| 1154.32|
|2004-05-11| 386.12| 386.12|
|2003-12-25| -92.0| 92.0|
|2004-04-26| -92.0| 92.0|
|2003-04-14| -92.0| 92.0|
|2003-11-26| -92.0| 92.0|
|2004-05-12| -79.53| 79.53|
+-----+-----+-----+
>>>

```

### Bonus: Visualizing (in PySpark shell, export or use external tools)

Example: export daily returns to CSV for plotting externally

```
df_ma.select("Date",
"Daily_Return").write.csv("/user/cdaccomcluster0116/CIPLA_daily_returns.csv", header=True)
```

## Phase 4: Correlation and Trend Analysis

### a) Correlation between Close Price and Volume

```
# Using Spark SQL corr() function
correlation_close_volume = spark.sql("""
SELECT corr(Close, Volume) AS corr_close_volume FROM cipla_data
""")
correlation_close_volume.show()
```

### b) Correlation Among Other Variables (Open, High, Low, Close)

```
# Example for Open vs High
spark.sql("SELECT corr(Open, High) AS corr_open_high FROM cipla_data").show()
spark.sql("SELECT corr(Open, Low) AS corr_open_low FROM cipla_data").show()
spark.sql("SELECT corr(High, Low) AS corr_high_low FROM cipla_data").show()
```



The screenshot shows a Firefox browser window with a terminal-like interface. The title bar reads "cdacomcluster0116@ip-172-31-32-118: ~ - Shell In A Box - Mozilla Firefox". The address bar shows "https://npapc.cloudloka.com/shell/". A warning message at the top says "⚠ Some of Firefox's security features may offer less protection on your current operating system. [How to fix this issue](#) **Don't show again**". The main content area displays Python code and its output:

```
...
>>> for var in variables:    corr = spark.sql(f"SELECT corr({var}, Close) AS corr_{var}_close FROM cipla_data")
...
>>> corr.show()
+-----+
|corr_Close_close|
+-----+
| 1.0|
+-----+
>>> spark.sql("SELECT corr(Open, High) AS corr_open_high FROM cipla_data").show()
+-----+
| corr_open_high|
+-----+
|0.9998678885345283|
+-----+
>>> spark.sql("SELECT corr(Open, Low) AS corr_open_low FROM cipla_data").show()
+-----+
| corr_open_low|
+-----+
|0.9998608157931267|
+-----+
>>> spark.sql("SELECT corr(High, Low) AS corr_high_low FROM cipla_data").show()
+-----+
| corr_high_low|
+-----+
|0.9998318809012967|
+-----+
>>>
```

## 4.2 Trend Analysis with Moving Averages (Crossovers)

To detect crossovers between the 50-day and 200-day moving averages (MA\_50 and MA\_200):

```
# Assuming df_ma has columns: Date, Close, MA_50, MA_200
from pyspark.sql.functions import lag, when, col
from pyspark.sql.window import Window
window_spec = Window.orderBy("Date")
df_ma = df_ma.withColumn("MA_50_prev", lag("MA_50").over(window_spec)) \
    .withColumn("MA_200_prev", lag("MA_200").over(window_spec))

# Define crossover signals
df_ma = df_ma.withColumn(
    "Signal",
    when(
        (col("MA_50") > col("MA_200")) & (col("MA_50_prev") <= col("MA_200_prev")),
        "Golden Cross (Bullish)"
    ).when(
        (col("MA_50") < col("MA_200")) & (col("MA_50_prev") >= col("MA_200_prev")),
        "Death Cross (Bearish)"
    ).otherwise("No Signal")
)
df_ma.select("Date", "MA_50", "MA_200", "Signal").filter(col("Signal") != "No Signal").show()
```

```

+-----+-----+
| Date| MA_50|MA_200| Signal|
+-----+-----+
|1996-03-12| 9.04| 9.03|Golden Cross (Bear...)|
|1996-11-12| 10.69| 10.72|Death Cross (Bear...)|
|1996-12-12| 10.95| 10.92|Golden Cross (Bear...)|
|1997-12-01| 18.45| 18.5|Death Cross (Bear...)|
|1998-04-24| 18.03| 17.93|Golden Cross (Bear...)|
|2000-03-29| 90.15| 90.44|Death Cross (Bear...)|
|2000-12-04| 70.46| 69.95|Golden Cross (Bear...)|
|2001-11-22| 87.11| 87.16|Death Cross (Bear...)|
|2001-12-17| 87.74| 87.67|Golden Cross (Bear...)|
|2002-02-28| 89.13| 89.27|Death Cross (Bear...)|
|2003-08-13| 63.02| 62.92|Golden Cross (Bear...)|
|2004-07-20| 92.02| 97.2|Death Cross (Bear...)|
|2004-09-23| 99.0| 98.78|Golden Cross (Bear...)|
|2005-04-13| 187.17| 107.21|Death Cross (Bear...)|
|2005-06-27| 111.45| 111.1|Golden Cross (Bear...)|
|2007-03-22| 243.29| 243.32|Death Cross (Bear...)|
|2008-01-21| 198.44| 198.25|Golden Cross (Bear...)|
|2008-11-11| 210.52| 210.65|Death Cross (Bear...)|
|2009-04-22| 205.2| 205.18|Golden Cross (Bear...)|
|2010-08-25| 327.26| 327.56|Death Cross (Bear...)|
+-----+-----+
only showing top 20 rows

```

>>> █

### 4.3 Price Volatility Analysis

#### a) Calculate 30-day Rolling Standard Deviation of Daily Returns (Volatility)

```
volatility = spark.sql("""
```

```
SELECT
```

```
Date,
```

```
Daily_Return,
```

```
STDDEV(Daily_Return) OVER (ORDER BY Date ROWS BETWEEN 29 PRECEDING AND
CURRENT ROW) AS rolling_volatility_30d
```

```
FROM cipla_data
```

```
ORDER BY Date
```

```
""")
```

```
volatility.show(10)
```

```

|2007-03-22|243.29|243.32|Death Cross (Bear...)
|2008-01-21|198.44|198.25|Golden Cross (Bull...)
|2008-11-11|210.52|210.65|Death Cross (Bear...)
|2009-04-22| 205.2|205.18|Golden Cross (Bull...)
|2010-08-25|327.26|327.56|Death Cross (Bear...)
+-----+
only showing top 20 rows

>>> volatility = spark.sql(""" SELECT Date, Daily_Return, STDEV(Daily_Return) OVER (ORDER BY Date ROWS BETWEEN 29 PRECEDING AND CURRENT ROW) AS rolling_volatility_30d FROM cipla_data ORDER BY Date """)
>>> volatility.show(10)
25/05/17 20:26:57 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
25/05/17 20:26:57 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
+-----+-----+-----+
| Date|Daily_Return|rolling_volatility_30d|
+-----+-----+-----+
|1996-01-01|      null|           null|
|1996-01-02|     -1.27|           null|
|1996-01-03|     -0.7|  0.40305086527633216|
|1996-01-04|    -1.05|  0.28746014216467186|
|1996-01-05|    -0.25|  0.4452246623896749|
|1996-01-08|    -1.72|  0.5581845572926575|
|1996-01-09|     0.27|  0.719184723604907|
|1996-01-10|     0.82|  0.8943100559596073|
|1996-01-11|     1.02|  0.9982269996634747|
|1996-01-12|     1.74|  1.1670047129296437|
+-----+
only showing top 10 rows
>>>

```

## 4.4 Detecting Seasonal Patterns

### a) Monthly Trends in Closing Prices (Average close price by month)

```

monthly_trends = spark.sql("""
SELECT
MONTH(Date) AS Month,
ROUND(AVG(Close), 2) AS Avg_Close
FROM cipla_data
GROUP BY MONTH(Date)
ORDER BY Month
""")
```

```
monthly_trends.show()
```

```

|1996-01-03|     -0.7|  0.40305086527633216|
|1996-01-04|    -1.05|  0.28746014216467186|
|1996-01-05|    -0.25|  0.4452246623896749|
|1996-01-08|    -1.72|  0.5581845572926575|
|1996-01-09|     0.27|  0.719184723604907|
|1996-01-10|     0.82|  0.8943100559596073|
|1996-01-11|     1.02|  0.9982269996634747|
|1996-01-12|     1.74|  1.1670047129296437|
+-----+
only showing top 10 rows

>>> monthly_trends = spark.sql(""" SELECT MONTH(Date) AS Month, ROUND(AVG(Close), 2) AS Avg_Close FROM cipla_data GROUP BY MONTH(Date) ORDER BY Month """)
>>> monthly_trends.show()
+-----+
|Month|Avg_Close|
+-----+
| 1| 368.69|
| 2| 371.15|
| 3| 357.13|
| 4| 364.66|
| 5| 371.01|
| 6| 377.66|
| 7| 388.37|
| 8| 403.14|
| 9| 417.12|
|10| 411.15|
|11| 391.52|
|12| 380.16|
+-----+

```

### 4.5 Detect Anomalous Trends (Outliers in Daily Returns)

Identify dates with extreme daily returns, e.g., beyond 3 standard deviations:

```
# Calculate mean and stddev of Daily_Return
stats = spark.sql("""
SELECT AVG(Daily_Return) AS mean_return, STDDEV(Daily_Return) AS stddev_return
FROM cipla_data
""").collect()[0]

mean_return = stats['mean_return']
stddev_return = stats['stddev_return']

threshold_upper = mean_return + 3 * stddev_return
threshold_lower = mean_return - 3 * stddev_return

anomalies = spark.sql(f"""
SELECT Date, Daily_Return
FROM cipla_data
WHERE Daily_Return > {threshold_upper} OR Daily_Return < {threshold_lower}
ORDER BY ABS(Daily_Return) DESC
""")
anomalies.show()
```

```
| 11| 391.52|
| 12| 380.16|
+-----+
>>> stats = spark.sql(""" SELECT AVG(Daily_Return) AS mean_return, STDDEV(Daily_Return) AS stddev_return FROM cipla_data """).collect()[0]
25/05/17 20:29:53 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
>>> mean_return = stats['mean_return']
>>> stddev_return = stats['stddev_return']
>>> threshold_upper = mean_return + 3 * stddev_return
>>> threshold_lower = mean_return - 3 * stddev_return
>>> anomalies = spark.sql(f""" SELECT Date, Daily_Return FROM cipla_data WHERE Daily_Return > {threshold_upper} OR Daily_Return < {threshold_lower} ORDER BY ABS(Daily_Return) DESC """)
>>> anomalies.show()
25/05/17 20:30:57 WARN WindowExec: No Partition Defined for Window operation! Moving all data to a single partition, this can cause serious performance degradation.
+-----+-----+
| Date|Daily_Return|
+-----+-----+
|2003-11-27| 1224.17|
|2004-04-27| 1190.79|
|2003-12-26| 1170.27|
|2003-04-15| 1154.32|
|2004-05-11| 386.12|
|2003-04-14| -92.0|
|2003-12-25| -92.0|
|2003-11-26| -92.0|
|2004-04-26| -92.0|
+-----+-----+
>>> 
```

## Phase 5: Time-Series Forecasting (Optional Advanced)

## Step 6: Strategic Insights and Recommendations

In this phase, we synthesize the findings from previous analyses to derive actionable insights and strategic recommendations. This step bridges the gap between raw data analysis and real-world decision-making, offering stakeholders actionable guidance for the CIPLA stock.

### 6.1 Key Findings

#### Price Trends (Moving Average Crossovers)

- *Insight:* Analysis of 50-day and 200-day moving averages reveals both bullish and bearish crossovers, signaling upward or downward momentum respectively.
- *Recommendation:* Use these crossovers as indicators for potential buy/sell points. Bullish crossovers suggest entry points, while bearish ones may prompt exits.

#### Volatility Analysis

- *Insight:* Volatility spikes (via rolling standard deviation of daily returns) are aligned with earnings reports or major sector developments.
- *Recommendation:* Avoid trades during these volatile windows unless well-informed. Risk-tolerant investors can explore quick entry/exit strategies based on high-volatility signals.

#### Trading Volume and Price Correlation

- *Insight:* Volume correlates moderately with price movements. Days with abnormally high volume often coincide with price surges or drops.
- *Recommendation:* Use volume trends to validate price movements. Spikes in volume can support the credibility of bullish/bearish signals.

#### Seasonal Patterns

- *Insight:* Historical monthly trends show consistent patterns—certain months (e.g., Q2 earnings season) exhibit higher average closing prices.
- *Recommendation:* Incorporate seasonal timing into investment strategy, planning entries and exits around historically stronger months.

#### Outliers and Event-Based Insights

- *Insight:* Extreme daily returns often reflect impactful news such as policy shifts, industry disruption, or key announcements.
- *Recommendation:* Employ real-time alert systems to track relevant news and events. This allows timely responses to market-moving information.

## **6.2 Strategic Recommendations for Stakeholders**

### **For Long-Term Investors:**

- Track long-term moving averages (200-day) as critical support/resistance indicators.
- Base buy/sell decisions on sustained crossovers and macroeconomic indicators.
- Ignore minor fluctuations to focus on long-term value appreciation.

### **For Short-Term Traders:**

- Use short-term MA (50-day) along with volume and volatility spikes to make trades.
- Monitor daily return outliers for breakout opportunities.
- Implement trailing stop-loss strategies to mitigate downside risks during volatile periods.

### **For Analysts and Portfolio Managers:**

- Integrate seasonality and historical performance into predictive modeling.
- Leverage inter-metric correlation (e.g., Volume vs. Daily\_Return) to rebalance portfolios.
- Track sectoral and global macro events for early signals affecting CIPLA stock behavior.

### **6.3 Summary of Recommendations**

#### **Buy Signals:**

- Bullish moving average crossovers.
- Volume spikes accompanying upward price movement.
- Historically strong seasonal windows.

#### **Sell Signals:**

- Bearish moving average crossovers.
- High volatility or unusually low trading volumes.
- Event-driven anomalies (e.g., policy change announcements).

#### **Risk Management:**

- Exercise caution during periods of extreme volatility.
- Set stop-loss levels during uncertain market conditions.
- Monitor price-volume alignment to validate technical indicators.

#### **Event-Based Strategy:**

- Pay close attention to quarterly earnings, major product rollouts, regulatory shifts, and macroeconomic changes.
- Develop a system to monitor and respond to key market-moving events in real time.

---

#### **Deliverable: Strategic Insights Report**

This report includes:

- **Findings and Insights:** Patterns, volatility phases, seasonal trends.
- **Investment Recommendations:** Strategies tailored to investor types.
- **Visual Aids:** Charts and graphs demonstrating MA crossovers, volume surges, and price trends from earlier analysis phases.

This strategic analysis positions investors and analysts to make data-driven, timely, and efficient decisions regarding CIPLA stock.