USING ONLY STOCK AND CRYPTO TABLE:

1. Understand Data Structure:
Objective: Establish time range and record count to scope the analysis.

Queries:
```
df_stocks.select(count("*").alias("total_rows"),
            min("Date").alias("start_date"),
            max("Date").alias("end_date")).show()

df_crypto.select(count("*").alias("total_rows"),
            min("timestamp").alias("start_date"),
            max("timestamp").alias("end_date")).show()
```

2. Exploratory Data Analysis (EDA)

Objective: Identify the most traded assets.

Queries:
```
df_stocks.groupBy("Symbol").agg(sum("Volume").alias("total_traded_volume"))\
      .orderBy(desc("total_traded_volume")).show(5)

df_crypto.groupBy("coin_name").agg(sum("volume").alias("total_traded_volume"))\
      .orderBy(desc("total_traded_volume")).show(5)
```

3. Performance Over Time

Objective: Track daily returns.

Query:

```
from pyspark.sql.window import Window
from pyspark.sql.functions import lag, col

window_stock = Window.partitionBy("Symbol").orderBy("Date")
df_stocks = df_stocks.withColumn("daily_return",
                      (col("Close") - lag("Close").over(window_stock)) /
lag("Close").over(window_stock))

window_crypto = Window.partitionBy("coin_name").orderBy("timestamp")
df_crypto = df_crypto.withColumn("daily_return",
                      (col("close") - lag("close").over(window_crypto)) /
lag("close").over(window_crypto))
```

4. Quarterly Performance

Objective: Detect trends and patterns over quarters.

Query:

```
from pyspark.sql.functions import date_trunc, avg

df_stocks = df_stocks.withColumn("quarter", date_trunc("quarter", col("Date")))
df_crypto = df_crypto.withColumn("quarter", date_trunc("quarter", col("timestamp")))

df_stocks.groupBy("quarter",
"Symbol").agg(avg("Close").alias("avg_quarterly_price")).show()
df_crypto.groupBy("quarter",
"coin_name").agg(avg("close").alias("avg_quarterly_price")).show()
```

5. Volatility Analysis

Objective: Measure the risk associated with assets.

Query:

```
from pyspark.sql.functions import stddev

df_stocks.groupBy("Symbol").agg(stddev("Close").alias("price_volatility")).show()
df_crypto.groupBy("coin_name").agg(stddev("close").alias("price_volatility")).show()
```

6. Correlation Between Stocks and Cryptos

Objective: Check if markets move together.

Query:

```
df_combined = df_stocks.join(df_crypto, df_stocks["Date"] == df_crypto["timestamp"],
"inner")
df_combined.select(corr("Close", "close").alias("price_correlation")).show()
```

7. Best & Worst Performing Assets

Objective: Identify top and bottom performers based on returns.

Query:

```
df_stocks = df_stocks.withColumn("total_return", (col("Close") - col("Open")) /
col("Open"))
df_crypto = df_crypto.withColumn("total_return", (col("close") - col("open")) /
col("open"))
```

```
df_stocks.orderBy(desc("total_return")).show(5)
df_stocks.orderBy("total_return").show(5)
df_crypto.orderBy(desc("total_return")).show(5)
df_crypto.orderBy("total_return").show(5)
```

8. Optional: Cumulative Portfolio Return

Objective: Simulate long-term growth with reinvested profits.

Query:

```
from pyspark.sql.functions import log, sum, exp

cumulative_window =
Window.partitionBy("Symbol").orderBy("Date").rowsBetween(Window.unboundedPrece
ding, 0)
df_stocks = df_stocks.withColumn("cumulative_return", exp(sum(log(1 +
col("daily_return"))).over(cumulative_window)))
```

9. Moving Average Crossovers

Objective: Detects bullish/bearish signals using moving average crossovers.

Query:

```
window_50 = Window.partitionBy("Symbol").orderBy("Date").rowsBetween(-49, 0)
window_200 = Window.partitionBy("Symbol").orderBy("Date").rowsBetween(-199, 0)
df_stocks = df_stocks.withColumn("SMA_50", avg("Close").over(window_50))
df_stocks = df_stocks.withColumn("SMA_200", avg("Close").over(window_200))
```

10. Volatility Comparison by Time Window

Objective**:** Measures how volatile each stock is over different months.

Query:

```
df_stocks.groupBy("Symbol", date_trunc("month", col("Date")).alias("month"))\
    .agg(stddev("Close").alias("monthly_volatility"))
```

11. Daily Price Gap (High - Low)

Objective: Identifies intraday volatility and potential trading opportunities.

Query:

```
df_stocks = df_stocks.withColumn("price_gap", col("High") - col("Low"))
```

Identifies intraday volatility and potential trading opportunities.

12. Top Performing Assets Over a Custom Period

Objective: Highlights which stocks or cryptos gained the most.

Query:

```
from pyspark.sql.functions import first, last
window_period = Window.partitionBy("Symbol").orderBy("Date")
df_returns = df_stocks.withColumn("first_close", first("Close").over(window_period))\
                 .withColumn("last_close", last("Close").over(window_period))\
                 .withColumn("total_return", (col("last_close") - col("first_close")) /
col("first_close"))
```

13 . Daily Volume Spikes

Objective: Detects unusual trading activity.

Query:

```
window_vol = Window.partitionBy("Symbol").orderBy("Date").rowsBetween(-7, -1)
df_stocks = df_stocks.withColumn("avg_weekly_vol", avg("Volume").over(window_vol))\
              .withColumn("volume_spike", col("Volume") / col("avg_weekly_vol"))
```

14. Rolling Return Volatility

Objective: Measures short-term risk for each asset.

Query:

```
rolling_window = Window.partitionBy("Symbol").orderBy("Date").rowsBetween(-6, 0)
df_stocks = df_stocks.withColumn("7_day_volatility",
stddev("daily_return").over(rolling_window))
```

15 . Weekly Return Aggregation

Objective: Tracks how assets perform on a weekly basis.

Query:

```
from pyspark.sql.functions import weekofyear

df_stocks = df_stocks.withColumn("week", weekofyear("Date"))
```

```
df_weekly = df_stocks.groupBy("Symbol",
"week").agg(avg("daily_return").alias("weekly_return"))
```

16. Asset Ranking by Sharpe Ratio

Objective: Finds the best-performing assets when adjusted for risk.

Query:

```
risk_free_rate = 0.02
df_stats = df_stocks.groupBy("Symbol").agg(mean("daily_return").alias("avg_ret"),
stddev("daily_return").alias("vol"))
df_stats = df_stats.withColumn("sharpe", (col("avg_ret") - risk_free_rate) / col("vol"))
```

USING ONLY SENTIMENT TABLE:

1. Average Sentiment Per Day

Objective: Reveals how sentiment towards Bitcoin fluctuates daily.

Query:

```
df_sentiment.groupBy("timestamp").agg(avg("polarity").alias("avg_daily_polarity")).orde
rBy("timestamp").show()
```

2. Subjectivity vs Polarity Scatter Plot

Objective: Identifies if more subjective posts tend to be more positive or negative.

Query:

```
df_sentiment.select("polarity", "subjectivity").toPandas().plot.scatter(x="subjectivity",
y="polarity")
```

3. Sentiment Over Time by Subsection

Objective: Compares sentiment trends across different crypto topics (e.g., BTC, ETH).

Query:

```
from pyspark.sql.functions import avg
df_sentiment.groupBy("timestamp",
"subsection").agg(avg("polarity").alias("avg_polarity")).show()
```

4. Most Positive and Most Negative Posts

Objective: Highlights the most extreme opinions in the dataset.

Query:

```
df_sentiment.orderBy("polarity", ascending=False).select("timestamp", "cleaned_post",
"polarity").show(5)
df_sentiment.orderBy("polarity").select("timestamp", "cleaned_post",
"polarity").show(5)
```

5. Count of Posts by Sentiment Category

Objective: Categorizes the dataset into sentiment classes to understand overall opinion distribution.

Query:

```
from pyspark.sql.functions import when

df_sentiment = df_sentiment.withColumn("sentiment_label",
   when(col("polarity") > 0.1, "Positive")
   .when(col("polarity") < -0.1, "Negative")
   .otherwise("Neutral")
)

df_sentiment.groupBy("sentiment_label").count().show()
```

6. Sentiment Volume Analysis

Objective: Correlates the amount of user activity with average sentiment on each day.

Query:

```
from pyspark.sql.functions import count

df_sentiment.groupBy("timestamp").agg(
   avg("polarity").alias("avg_polarity"),
   count("post").alias("post_count")
).orderBy("timestamp").show()
```

7. Sentiment Spikes Detection

Objective: Detects sudden shifts in sentiment that may indicate news events or market reactions.

Query:

```
from pyspark.sql.functions import lag
window = Window.orderBy("timestamp")
df_sentiment = df_sentiment.withColumn("previous_polarity",
lag("polarity").over(window))
df_sentiment = df_sentiment.withColumn("polarity_jump", col("polarity") -
col("previous_polarity"))
df_sentiment.filter(abs(col("polarity_jump")) > 0.5).select("timestamp", "post",
"polarity_jump").show()
```

9. Rolling Average Sentiment (7-day)

Objective: Smooths out daily noise to detect longer-term sentiment trends.

Query:

```
from pyspark.sql.window import Window
rolling = Window.orderBy("timestamp").rowsBetween(-6, 0)
df_sentiment = df_sentiment.withColumn("rolling_sentiment",
avg("polarity").over(rolling))
```

10. Sentiment Breakdown by Subject

Objective: Identifies which discussion topics attract more positive or negative sentiment.

Query:

```
df_sentiment.groupBy("subject").agg(avg("polarity").alias("avg_sentiment"),
count("post").alias("volume"))\
        .orderBy("avg_sentiment", ascending=False).show()
```

USING ALL THE THREE TABLES:

Creating a new one:

```
from pyspark.sql.functions import col

 Rename and prepare individual dataframes
stocks_prepped = df_stocks.select(
  col("Date").alias("DATE"),
  col("Symbol").alias("s_symbol"),
  col("Open").alias("s_open"),
  col("Close").alias("s_close")
)

crypto_prepped = df_crypto.select(
  col("timestamp").alias("DATE"),
  col("open").alias("c_open"),
  col("close").alias("c_close")
)

sentiment_prepped = df_sentiment.select(
  col("timestamp").alias("DATE"),
  col("polarity").alias("sentiment")
)

 Join all three on DATE
transformed_data = stocks_prepped.join(crypto_prepped, "DATE")\
                .join(sentiment_prepped, "DATE")

 Register for SQL queries (if using Spark SQL)
transformed_data.createOrReplaceTempView("transformed_data")
```

1. Impact of Sentiment on Market Movements

1.1 Does Sentiment Affect Market Open Prices?
This query examines whether yesterday's sentiment score impacts today's stock opening prices, potentially revealing if news sentiment predicts market movement.

```
sql
WITH sentiment_lagged AS (
  SELECT DATE, sentiment,
      LAG(sentiment) OVER (ORDER BY DATE) AS prev_day_sentiment
  FROM transformed_data
)
SELECT s_symbol AS Stock,
    ROUND(AVG((s_open - LAG(s_close) OVER (PARTITION BY s_symbol ORDER BY
DATE)) / LAG(s_close) OVER (PARTITION BY s_symbol ORDER BY DATE) * 100), 2) AS
avg_open_change,
    ROUND(AVG(prev_day_sentiment), 2) AS avg_previous_sentiment
FROM transformed_data
JOIN sentiment_lagged USING (DATE)
GROUP BY s_symbol
ORDER BY avg_previous_sentiment DESC;
```

1.2 Most News-Sensitive Stocks

This query identifies which stocks react the most to sentiment fluctuations, highlighting opportunities for sentiment-driven trading.

```sql
SELECT s_symbol AS Stock,
    ROUND(CORR(sentiment, (s_close - s_open) / s_open), 2) AS sentiment_correlation
FROM transformed_data
GROUP BY s_symbol
ORDER BY sentiment_correlation DESC;
```

1.3 Quarterly Sentiment vs. Market Performance

Compare quarterly sentiment averages against quarterly stock and crypto performance.

```sql
SELECT QUARTER(DATE) AS quarter,
    YEAR(DATE) AS year,
    ROUND(AVG(sentiment), 2) AS avg_sentiment,
    ROUND(AVG((s_close - s_open) / s_open) * 100, 2) AS avg_stock_return,
    ROUND(AVG((c_close - c_open) / c_open) * 100, 2) AS avg_crypto_return
FROM transformed_data
GROUP BY QUARTER(DATE), YEAR(DATE)
ORDER BY year DESC, quarter DESC;
```

🔷 Insight: See if quarterly market performance aligns with sentiment trends.

2. Market Trends and Volatility

2.1 Quarterly Performance Differences

Compare quarter-over-quarter (QoQ) stock and crypto returns.

```sql
WITH quarterly_returns AS (
  SELECT QUARTER(DATE) AS quarter,
      YEAR(DATE) AS year,
      s_symbol,
      ROUND(AVG((s_close - s_open) / s_open) * 100, 2) AS stock_return,
      ROUND(AVG((c_close - c_open) / c_open) * 100, 2) AS crypto_return
  FROM transformed_data
  GROUP BY QUARTER(DATE), YEAR(DATE), s_symbol
)
SELECT s_symbol, year, quarter,
    stock_return,
    LAG(stock_return) OVER (PARTITION BY s_symbol ORDER BY year, quarter) AS
prev_qtr_stock_return,
    crypto_return,
    LAG(crypto_return) OVER (PARTITION BY s_symbol ORDER BY year, quarter) AS
prev_qtr_crypto_return,
    ROUND(stock_return - LAG(stock_return) OVER (PARTITION BY s_symbol ORDER BY
year, quarter), 2) AS stock_qtr_change,
    ROUND(crypto_return - LAG(crypto_return) OVER (PARTITION BY s_symbol ORDER
BY year, quarter), 2) AS crypto_qtr_change
FROM quarterly_returns;
```

Insight: Measures how stocks and cryptos performed across different quarters and tracks momentum shifts.

2.2 Market Shocks & Anomalies by Quarter
Identify quarters with extreme volatility in stock or crypto markets.

sql
```
SELECT QUARTER(DATE) AS quarter,
    YEAR(DATE) AS year,
    ROUND(STDDEV((s_close - s_open) / s_open) * 100, 2) AS stock_volatility,
    ROUND(STDDEV((c_close - c_open) / c_open) * 100, 2) AS crypto_volatility
FROM transformed_data
GROUP BY QUARTER(DATE), YEAR(DATE)
ORDER BY year DESC, quarter DESC;
```

Insight: Helps in identifying quarters with abnormal price swings.

3. Trading Strategies & Market Patterns

3.1 Best & Worst Quarters for Each Asset
Find the best and worst quarters for each stock and crypto.

sql
```
SELECT s_symbol, QUARTER(DATE) AS quarter, YEAR(DATE) AS year,
    ROUND(AVG((s_close - s_open) / s_open) * 100, 2) AS avg_return
FROM transformed_data
GROUP BY s_symbol, QUARTER(DATE), YEAR(DATE)
ORDER BY avg_return DESC;
```

IInsight: Reveals which quarters are historically best and worst for each asset.

3.2 Which Quarters Have the Strongest Correlation Between Stocks & Crypto?
Find periods where stocks and crypto move together.

```sql
SELECT QUARTER(DATE) AS quarter, YEAR(DATE) AS year,
    ROUND(CORR(s_close, c_close), 2) AS stock_crypto_correlation
FROM transformed_data
GROUP BY QUARTER(DATE), YEAR(DATE)
ORDER BY year DESC, quarter DESC;
```

Insight: Helps in diversification strategies based on quarterly data.

4. Long-Term Market Behavior & Forecasting

4.1 Comparing Yearly Performance of Stocks vs. Crypto

Tracks year-over-year (YoY) market trends.

```sql
SELECT YEAR(DATE) AS year,
    ROUND(AVG((s_close - s_open) / s_open) * 100, 2) AS avg_stock_return,
    ROUND(AVG((c_close - c_open) / c_open) * 100, 2) AS avg_crypto_return
FROM transformed_data
GROUP BY YEAR(DATE)
ORDER BY year DESC;
```

Insight: Shows whether stocks or crypto performed better over the years.