mp11v2

FINISHED

```
%flink.ssql(type=update)
DROP TABLE IF EXISTS stock_table;

-- create a table to glue data catalog table with columns for stock data,
-- sets a watermark to trigger late arrival events, and configures it to read
-- from a Kinesis stream, in JSON format with an ISO-8601 timestamp format.
CREATE TABLE stock_table(
    `date` STRING,
    ticker VARCHAR(6),
    open_price FLOAT,
    high FLOAT,
    low FLOAT,
    close_price FLOAT,
    adjclose FLOAT,
    volume BIGINT,
    event_time TIMESTAMP(3),
    WATERMARK FOR event_time as event_time - INTERVAL '5' SECOND
) WITH ( --connect to your kinesis data stream
    'connector' = 'kinesis',
    'stream' = 'mp11v2_ds',
    'aws.region' = 'us-east-1',
    'scan.stream.initpos' = 'TRIM_HORIZON',
    'format' = 'json',
    'json.timestamp-format.standard' = 'ISO-8601'
);
```

```
Table has been dropped.
Table has been created.
```

Took 3 sec. Last updated by anonymous at May 12 2023, 1:03:04 PM.

FINISHED

```
%flink.pyflink

from pyflink.datastream import StreamExecutionEnvironment
from pyflink.table import StreamTableEnvironment, DataTypes
from pyflink.table.udf import udf
from datetime import datetime

# Unregister the existing function if it exists
st_env.execute_sql("DROP TEMPORARY FUNCTION IF EXISTS calculate_cmgr")

# Define the custom function
@udf(result_type=DataTypes.FLOAT(), input_types=[DataTypes.STRING(), DataTypes.FLOAT()])
def calculate_cmgr(date_string: str, close_price: float) -> float:
    start_date = datetime.strptime("01/04/2021", "%m/%d/%Y").date()
    current_date = datetime.strptime(date_string, '%m/%d/%Y').date()

    # Check if the day is within the first three days of the month and not January 2021
    if current_date.day <= 3 and not (current_date.month == 1 and current_date.year == 2021):
        # Compute the number of months between the start date and the current date
        number_of_months = (current_date.year - start_date.year) * 12 + (current_date.month - start_date.month)

        # Calculate the CMGR
        cmgr = ((close_price / 92.3) ** (1 / number_of_months) - 1) * 100
        return cmgr
    else:
        return 0.0

# Register the custom function with a name
st_env.create_temporary_function("calculate_cmgr", calculate_cmgr)
```

Took 1 sec. Last updated by anonymous at May 12 2023, 1:03:09 PM.

FLINK JOB (/flinkdashboard/#/job/463ea03433fc8141a8954b29ad6fc2b4) ABORT

```
%flink.ssql(type=update)

SELECT event_time, calculate_cmgr(`date`, close_price) AS cmgr, `date`, close_price
FROM stock_table
WHERE calculate_cmgr(`date`, close_price) <> 0.0
```

settings

mp11v2

Available Fields

event_time | cmgr | date | close_price

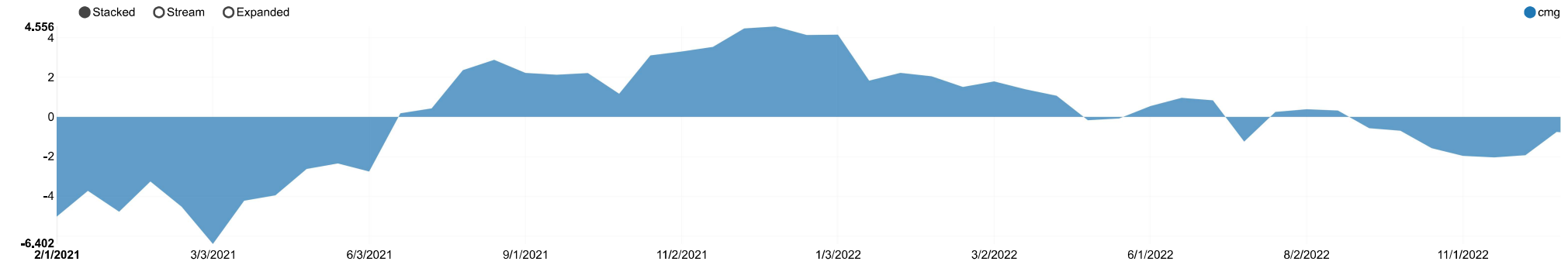
keys: date ✖

groups:

values: cmgr SUM ✖

xAxis :

Default | Rotate | Hide




```
Fail to run sql command: SELECT event_time, calculate_cmgr(`date`, close_price) AS cmgr, `date`, close_price
FROM stock_table
WHERE calculate_cmgr(`date`, close_price) <> 0.0
```

▸ **Job was cancelled.**

Took 48 sec. Last updated by anonymous at May 12 2023, 4:54:06 PM. (outdated)

FINISHED

```
%flink.pyflink

from pyflink.datastream import StreamExecutionEnvironment
from pyflink.table import StreamTableEnvironment, DataTypes
from pyflink.table.udf import udf

# Unregister the existing function if it exists
st_env.execute_sql("DROP TEMPORARY FUNCTION IF EXISTS calculate_ema")

# Define the EMA calculation function
@udf(result_type=DataTypes.FLOAT(), input_types=[DataTypes.FLOAT(), DataTypes.FLOAT()])
def calculate_ema(prev_ema: float, current_price: float) -> float:
    return

# Register the EMA calculation function
st_env.register_function("calculate_ema", calculate_ema)
```

Took 1 sec. Last updated by anonymous at May 12 2023, 4:21:33 PM. (outdated)

mp11v2

FLINK JOB (/flinkdashboard/#/job/6826597506541a72f08b26971d583f91) ABORT

```
%flink.ssql(type=update)
SELECT *
FROM (
    SELECT
    event_time,
    ticker,
    close_price,
    sma as avg_close_price,
    ((close_price - emaprev) * (2.0/11.0) + emaprev) AS ema_close_price,
    `date`
    FROM (
        SELECT
            event_time,
            ticker,
            close_price,
            sma,
            (close_price - LAG(sma, 1) OVER (ORDER BY event_time)) * (2.0/11.0) + LAG(sma, 1) OVER (ORDER BY event_time) as emaprev,
            `date`
        FROM (
            SELECT
                event_time,
                ticker,
                close_price,
                AVG(close_price) OVER (
                  PARTITION BY ticker
                  ORDER BY event_time
                  ROWS BETWEEN 9 PRECEDING AND CURRENT ROW
                ) AS sma,
                `date`
            FROM stock_table
        ) t
    ) t1
) t2
WHERE close_price < ema_close_price;
```

settings

Available Fields

event_time | ticker | close_price | avg_close_price | ema_close_price | date

keys: date ✖

groups:

values: ema_close_price SUM ✖ | close_price SUM ✖

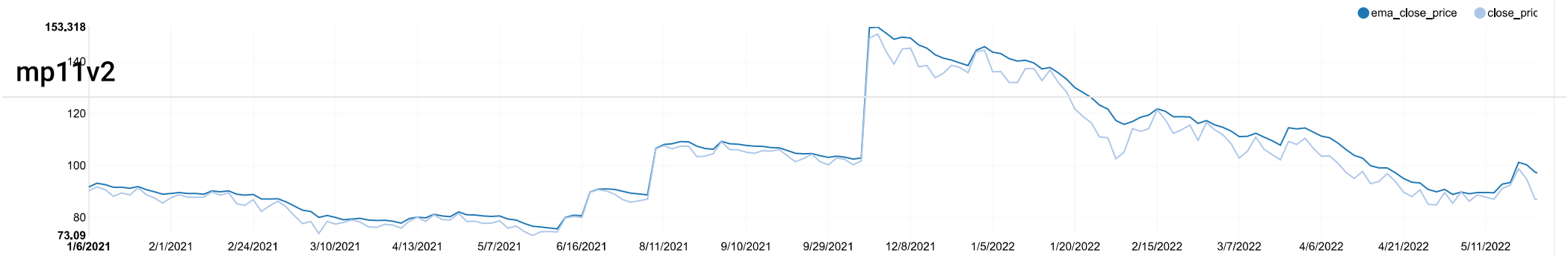
force Y to 0

zoom

Date format

xAxis :

Default | Rotate | Hide



mp11v2

```
Fail to run sql command: SELECT *
FROM (
    SELECT
    event_time,
    ticker,
    close_price,
    sma as avg_close_price,
    ((close_price - emaprev) * (2.0/11.0) + emaprev) AS ema_close_price,
    `date`
    FROM (
        SELECT
            event_time,
            ticker,
            close_price,
            sma,
            (close_price - LAG(sma, 1) OVER (ORDER BY event_time)) * (2.0/11.0) + LAG(sma, 1) OVER (ORDER BY event_time) as emaprev,
            `date`
        FROM (
```

▸ **Job was cancelled.**

Took 1 min 28 sec. Last updated by anonymous at May 12 2023, 5:01:03 PM. (outdated)

```
%flink.ssql(type=update)

SELECT *
FROM stock_table
    MATCH_RECOGNIZE(
        PARTITION BY ticker
        ORDER BY event_time
        MEASURES
            A.event_time as event_time,
            FIRST(A.`date`) AS initialPriceDate,
            LAST(B.`date`) AS dropDate,
            (-100*(C.close_price - A.close_price)/C.close_price) AS dropPercentage,
            FIRST(A.close_price) AS initialPrice,
            LAST(C.close_price) AS lastPrice
        ONE ROW PER MATCH
        AFTER MATCH SKIP PAST LAST ROW
        PATTERN (A B* C) WITHIN INTERVAL '1' MINUTE
        DEFINE
            B AS (B.close_price >= (A.close_price * 0.92)) AND (B.close_price <= (A.close_price * 1.08)),
            C AS (-100*(C.close_price - A.close_price)/C.close_price) >= 8.0
    )
```

FLINK JOB (/flinkdashboard/#/job/e09e68d801384a1e7fc53ae758b8dd59) ABORT

settings

ticker	event_time	initialPriceDate	dropDate	dropPercentage	initialPrice	lastPrice

mp11v2

AMD	2023-05-12 17:02:18.597	2/4/2022	2/10/2022	9.206572	123.6	113.18
AMD	2023-05-12 17:02:18.837	2/14/2022	3/4/2022	10.995629	114.27	102.95
AMD	2023-05-12 17:02:19.637	3/16/2022	4/5/2022	11.285815	115.37	103.67
AMD	2023-05-12 17:02:20.237	4/7/2022	4/11/2022	9.061146	103.72	95.1
AMD	2023-05-12 17:02:20.377	4/13/2022	4/20/2022	8.7813015	97.74	89.85
AMD	2023-05-12 17:02:20.917	5/4/2022	5/6/2022	15.122739	99.42	86.36
AMD	2023-05-12 17:02:21.237	5/17/2022	5/19/2022	9.593584	102.47	93.5
AMD	2023-05-12 17:02:21.537	5/27/2022	6/10/2022	17.553747	102.26	86.99

```
Fail to run sql command: SELECT *
FROM stock_table
    MATCH_RECOGNIZE(
        PARTITION BY ticker
        ORDER BY event_time
        MEASURES
            A.event_time as event_time,
            FIRST(A.`date`) AS initialPriceDate,
            LAST(B.`date`) AS dropDate,
            (-100*(C.close_price - A.close_price)/C.close_price) AS dropPercentage,
            FIRST(A.close_price) AS initialPrice,
            LAST(C.close_price) AS lastPrice
        ONE ROW PER MATCH
        AFTER MATCH SKIP PAST LAST ROW
        PATTERN (A B* C) WITHIN INTERVAL '1' MINUTE
        DEFINE
            B AS (B.close_price >= (A.close_price * 0.92)) AND (B.close_price <= (A.close_price * 1.08)),
```

▸ **Job was cancelled.**

Took 1 min 15 sec. Last updated by anonymous at May 12 2023, 6:04:49 PM.