DÉPARTEMENT:INFORMATIQUE

FILIÈRE: Data & Intelligence for Smart Systems

Report on Stock Market Values Data Processing and Analytics (DPA) Assignment

Realized by:

Idelhaj Amine

Supervised by:

Academic year :
2023 | 2024

Introduction

The objective of this project was to utilize Spark Streaming to analyze the value of different stocks over time. In this report, we will outline the tasks performed and the key findings of this project. We worked on a dataset stored in a CSV file, focusing on two main columns: the stock name and its corresponding value.

Development Environment

For this project, we used the same development environment as the one used for the Spark and Spark Streaming practical work during the Data Processing and Analytics (DPA) course. Additionally, we made use of the provided notebooks available on Moodle to facilitate data processing and analysis.

Data Processing

The data used for this project was contained in a CSV file named "stock.csv." To ingest the data into Spark Streaming, we utilized a notebook named "Kafka_Producer_for_Project." This notebook handled the parsing of the CSV file and ingested the data into Kafka with a structured schema consisting of the stock name, price, and a timestamp generated at the time of ingestion.

Tasks 1. The N Most Valuable Stocks in Each Window

Introduction

In this section, we will explain the code that calculates the N most valuable stocks in each time window. This code is part of the project that utilizes Spark Streaming to analyze stock market values.

Code Purpose

The purpose of this code is to provide a mechanism for monitoring and analyzing the N most valuable stocks within specific time windows. The code enables real-time insights into which stocks are currently performing well based on their average prices. The code uses a sliding window approach to continuously update this information.

Code Details

Configuration The code begins by defining some configuration parameters:

- N: This parameter specifies the number of most valuable stocks to identify in each window.
- windowDuration: The duration of the time window within which to compute the most valuable stocks. In this code, it is set to "5 minutes."
- slideDuration: The duration for sliding the time window. In this code, it is set to "2 minutes."

Assigning Watermarks

To ensure that event-time processing is taken into account, a watermark is assigned to the dataset. Watermarks in Spark Streaming allow handling late data that might arrive out of order. In this code, the watermark is set to "5 minutes," meaning that data received with a timestamp within the last 5 minutes will be considered.

Grouping and Aggregation

The code groups the data by two key attributes: the window and the stock name. The window is defined using the window function from pyspark.sql.functions, and it groups the data based on the specified windowDuration and slideDuration. For each group, the code calculates the average price of the stocks in that group using the avg function. The result of this aggregation is assigned an alias "avgPrice."

Writing Aggregated Data

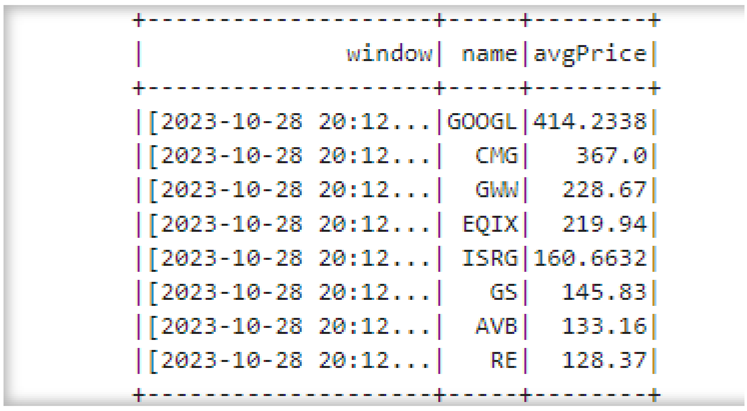
After aggregating the data, it is written to memory for querying. This in-memory table is given the query name "Aggr." The output mode is set to "update," which means it will continuously update as new data arrives. A streaming query is started to execute this operation.

Fetching the Top N Stocks

To identify the N most valuable stocks in each window, a SQL query is executed using the spark.sql function. The query selects the most valuable stocks from the "Aggr" in-memory table, ordering them by window (in descending order) and average price (in descending order). The LIMIT clause is used to restrict the result to the top N stocks.

Displaying the Result

Finally, the code displays the result, showing the time window, stock name, and average price of the N most valuable stocks in each window using the result.show() method.



```
+--------------------+-----+--------+
|              window| name|avgPrice|
+--------------------+-----+--------+
|[2023-10-28 20:12...|GOOGL|414.2338|
|[2023-10-28 20:12...|  CMG|   367.0|
|[2023-10-28 20:12...|  GWW|  228.67|
|[2023-10-28 20:12...| EQIX|  219.94|
|[2023-10-28 20:12...| ISRG|160.6632|
|[2023-10-28 20:12...|   GS|  145.83|
|[2023-10-28 20:12...|  AVB|  133.16|
|[2023-10-28 20:12...|   RE|  128.37|
+--------------------+-----+--------+
```

Conclusion

This code is essential for continuously monitoring stock performance within specified time windows, making it valuable for traders and investors looking to track real-time insights into the most valuable stocks and potentially optimize their investment strategies.

Tasks 2. Select the stocks that lost value between windows

Introduction

In this section, we will explain the code that identifies the stocks that lost value between time windows using Spark Streaming. This code is an integral part of the project's objective to analyze stock market values in real-time.

Code Purpose

The purpose of this code is to track and identify stocks that have experienced a decrease in their value when transitioning between different time windows. By analyzing this data, traders and investors can gain insights into stocks that might be facing downward trends.

Code Details

Class Definition:

StockState A class named StockState is defined to represent the state of each stock. It keeps track of the stock's name, its last recorded price, the end time of the last recorded time window, and a flag to indicate whether the stock lost value. The class includes an update method that updates the stock's state based on the received price and window end time. It determines if the stock lost value by comparing the last recorded price with the current price.

Watermark and Window Operations

The code defines a watermark and window operation on the streaming DataFrame (df). Watermarks are used to handle late-arriving data and are essential for event-time processing. The streaming DataFrame is grouped by the stock's name and the window it belongs to, based on a 5-minute window duration. This grouping allows for aggregation of data within these time windows. The last function is applied to obtain the last recorded price for each group within the specified window.

Batch Processing Function

The process_batch function is defined to process the DataFrame row by row for each batch of data. This function is executed for each batch of data received during the streaming process.

Processing Rows

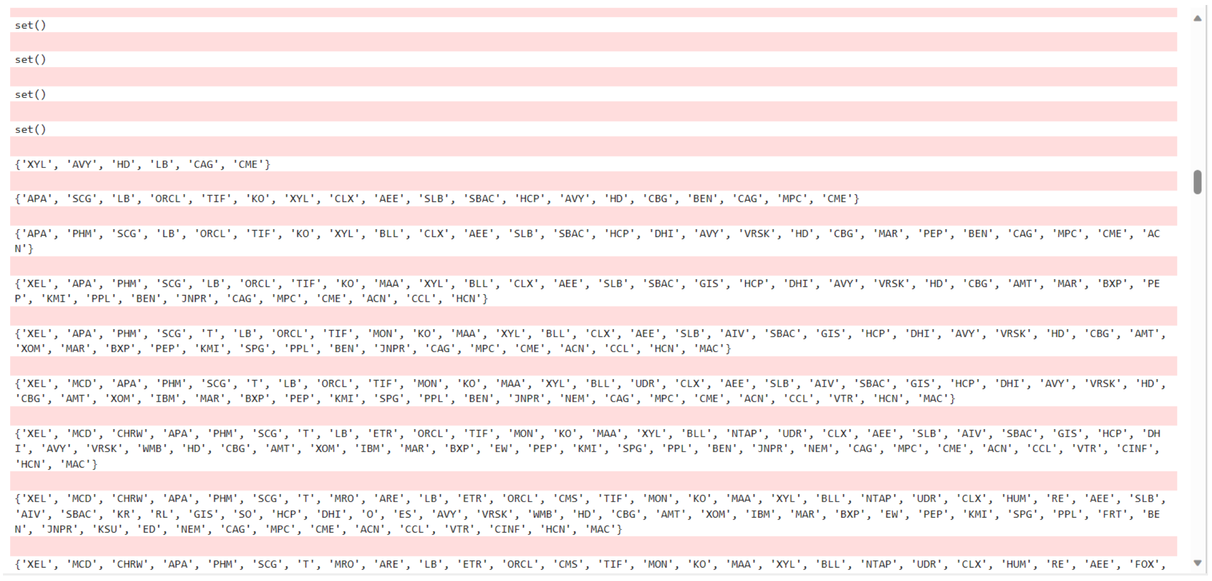
Inside the process_batch function, each row of data is processed individually. For each row, the stock's name, the last recorded price, and the end time of the window are extracted. The code retrieves the stock's state from the stock_states dictionary or creates a new one if it doesn't exist. The update method of the StockState class is called to update the stock's state based on the current price and window end time. If the stock lost value during this time window, it is added to the stocks_that_lost_value set.

Streaming Query

A streaming query is defined using the writeStream API. This query uses the process_batch function to handle each batch of data. The output mode is set to "update," indicating that the query will update as new data arrives. The streaming query is started using the start method.

Await Termination

The code includes query.awaitTermination(timeout=60) to allow the streaming query to run continuously for up to 60 seconds. This is a way to ensure the code does not run indefinitely.
The output is like this:



```
set()

set()

set()

set()

{'XYL', 'AVY', 'HD', 'LB', 'CAG', 'CME'}

{'APA', 'SCG', 'LB', 'ORCL', 'TIF', 'KO', 'XYL', 'CLX', 'AEE', 'SLB', 'SBAC', 'HCP', 'AVY', 'HD', 'CBG', 'BEN', 'CAG', 'MPC', 'CME'}

{'APA', 'PHM', 'SCG', 'LB', 'ORCL', 'TIF', 'KO', 'XYL', 'BLL', 'CLX', 'AEE', 'SLB', 'SBAC', 'HCP', 'DHI', 'AVY', 'VRSK', 'HD', 'CBG', 'MAR', 'PEP', 'BEN', 'CAG', 'MPC', 'CME', 'AC
N'}

{'XEL', 'APA', 'PHM', 'SCG', 'LB', 'ORCL', 'TIF', 'KO', 'MAA', 'XYL', 'BLL', 'CLX', 'AEE', 'SLB', 'SBAC', 'GIS', 'HCP', 'DHI', 'AVY', 'VRSK', 'HD', 'CBG', 'AMT', 'MAR', 'BXP', 'PE
P', 'KMI', 'PPL', 'BEN', 'JNPR', 'CAG', 'MPC', 'CME', 'ACN', 'CCL', 'HCN'}

{'XEL', 'APA', 'PHM', 'SCG', 'T', 'LB', 'ORCL', 'TIF', 'MON', 'KO', 'MAA', 'XYL', 'BLL', 'CLX', 'AEE', 'SLB', 'AIV', 'SBAC', 'GIS', 'HCP', 'DHI', 'AVY', 'VRSK', 'HD', 'CBG', 'AMT',
'XOM', 'MAR', 'BXP', 'PEP', 'KMI', 'SPG', 'PPL', 'BEN', 'JNPR', 'CAG', 'MPC', 'CME', 'ACN', 'CCL', 'HCN', 'MAC'}

{'XEL', 'MCD', 'APA', 'PHM', 'SCG', 'T', 'LB', 'ORCL', 'TIF', 'MON', 'KO', 'MAA', 'XYL', 'BLL', 'UDR', 'CLX', 'AEE', 'SLB', 'AIV', 'SBAC', 'GIS', 'HCP', 'DHI', 'AVY', 'VRSK', 'HD',
'CBG', 'AMT', 'XOM', 'IBM', 'MAR', 'BXP', 'PEP', 'KMI', 'SPG', 'PPL', 'BEN', 'JNPR', 'NEM', 'CAG', 'MPC', 'CME', 'ACN', 'CCL', 'VTR', 'HCN', 'MAC'}

{'XEL', 'MCD', 'CHRW', 'APA', 'PHM', 'SCG', 'T', 'LB', 'ETR', 'ORCL', 'TIF', 'MON', 'KO', 'MAA', 'XYL', 'BLL', 'NTAP', 'UDR', 'CLX', 'AEE', 'SLB', 'AIV', 'SBAC', 'GIS', 'HCP', 'DH
I', 'AVY', 'VRSK', 'WMB', 'HD', 'CBG', 'AMT', 'XOM', 'IBM', 'MAR', 'BXP', 'EW', 'PEP', 'KMI', 'SPG', 'PPL', 'BEN', 'JNPR', 'NEM', 'CAG', 'MPC', 'CME', 'ACN', 'CCL', 'VTR', 'CINF',
'HCN', 'MAC'}

{'XEL', 'MCD', 'CHRW', 'APA', 'PHM', 'SCG', 'T', 'MRO', 'ARE', 'LB', 'ETR', 'ORCL', 'CMS', 'TIF', 'MON', 'KO', 'MAA', 'XYL', 'BLL', 'NTAP', 'UDR', 'CLX', 'HUM', 'RE', 'AEE', 'SLB',
'AIV', 'SBAC', 'KR', 'RL', 'GIS', 'SO', 'HCP', 'DHI', 'O', 'ES', 'AVY', 'VRSK', 'WMB', 'HD', 'CBG', 'AMT', 'XOM', 'IBM', 'MAR', 'BXP', 'EW', 'PEP', 'KMI', 'SPG', 'PPL', 'FRT', 'BE
N', 'JNPR', 'KSU', 'ED', 'NEM', 'CAG', 'MPC', 'CME', 'ACN', 'CCL', 'VTR', 'CINF', 'HCN', 'MAC'}

{'XEL', 'MCD', 'CHRW', 'APA', 'PHM', 'SCG', 'T', 'MRO', 'ARE', 'LB', 'ETR', 'ORCL', 'CMS', 'TIF', 'MON', 'KO', 'MAA', 'XYL', 'BLL', 'NTAP', 'UDR', 'CLX', 'HUM', 'RE', 'AEE', 'FOX',
```

Stopping the Spark Session

After the streaming query has completed, the Spark session is stopped using spark.stop() to release any acquired resources.

Conclusion

This code is essential for identifying stocks that have lost value between time windows, providing real-time insights into which stocks may be experiencing negative trends or declines in value.

Tasks 3. Find the stocks that gained the most between windows

Introduction

This code is designed to identify the stock with the maximum gain between time windows using Spark Streaming. The code is part of the project's objective to analyze stock market values in real-time.

Code Purpose

The primary objective of this code is to determine which stock has gained the most in value when transitioning between different time windows. By analyzing this data, investors can gain insights into which stocks have experienced the most significant increase in value, making them potentially lucrative investment options.

Code Details

Class Definition:

StockState A class named StockState is defined to represent the state of each stock. It keeps track of the stock's name, its last recorded price, and the maximum gain achieved. The class includes an update method that updates the stock's state based on the received price. It calculates the gain by comparing the last recorded price with the current price and updates the maximum gain if a higher gain is achieved.

Watermark and Window Operations

The code defines a watermark and window operation on the streaming DataFrame (df). Watermarks are used to handle late-arriving data and are essential for event-time processing. The streaming DataFrame is grouped by the stock's name and the window it belongs to, based on a 5-minute window duration. This grouping allows for aggregation of data within these time windows. The last function is applied to obtain the last recorded price for each group within the specified window.

Batch Processing Function

The process_batch function is defined to process the DataFrame row by row for each batch of data. This function is executed for each batch of data received during the streaming process.

Processing Rows

Inside the process_batch function, each row of data is processed individually. For each row, the stock's name and the last recorded price are extracted. The code retrieves the stock's state from the stock_states dictionary or creates a new one if it doesn't exist. The update method of the StockState class is called to update the stock's state based on the current price.

Finding the Stock with Maximum Gain

After processing all the rows in the batch, the code identifies the stock with the maximum gain by using the max function and a lambda function. The key parameter of the max function specifies that the stock with the maximum max_gain should be selected. The code then retrieves and prints the stock with the maximum gain and the amount of gain it achieved.

Streaming Query

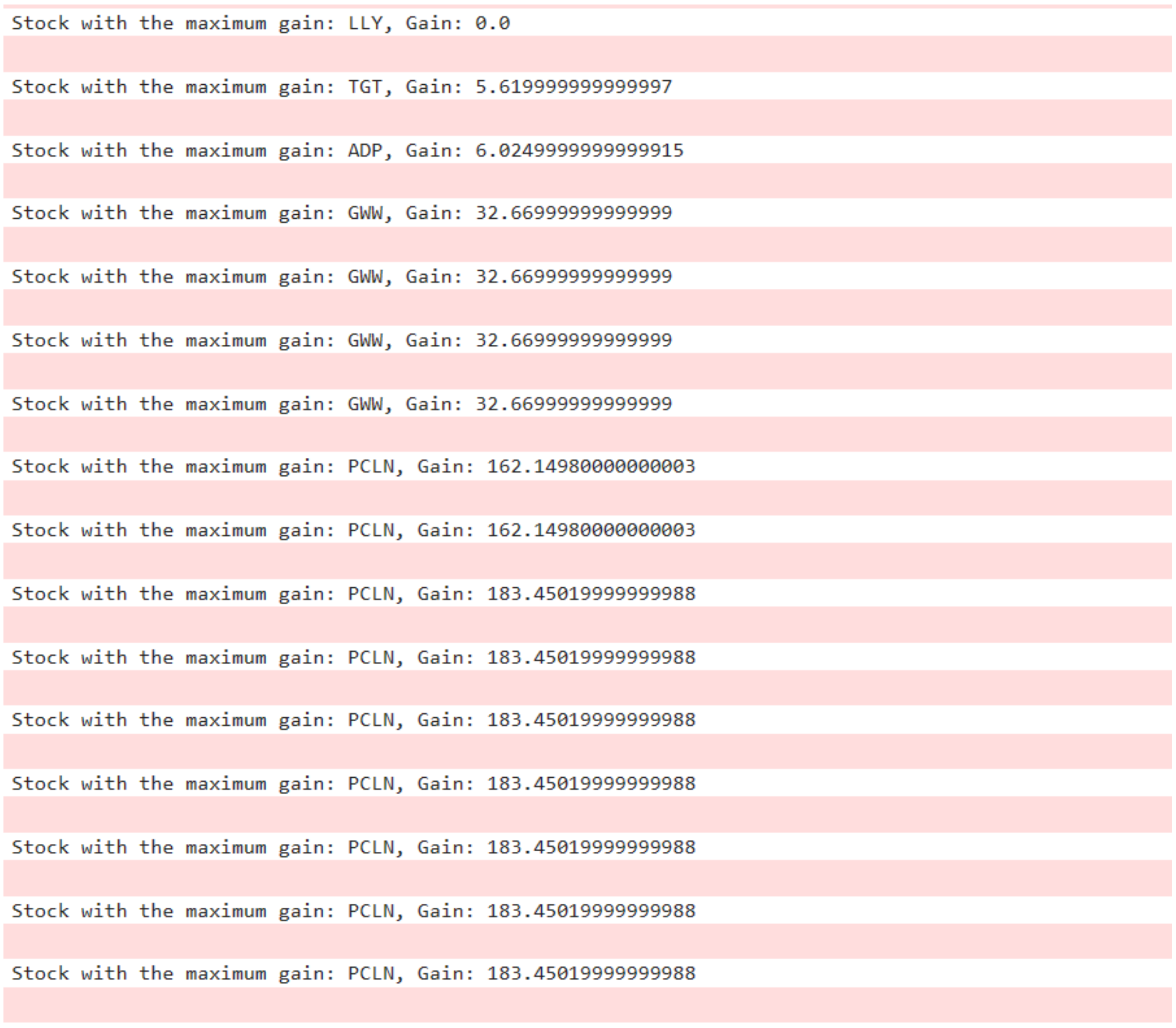
A streaming query is defined using the writeStream API. This query uses the process_batch function to handle each batch of data. The output mode is set to "update," indicating that the query will update as new data arrives. The streaming query is started using the start method.

Await Termination

The code includes query.awaitTermination(timeout=60) to allow the streaming query to run continuously for up to 60 seconds. This is a way to ensure the code does not run indefinitely.

Stopping the Spark Session

After the streaming query has completed, the Spark session is stopped using spark.stop() to release any acquired resources.



```
Stock with the maximum gain: LLY, Gain: 0.0

Stock with the maximum gain: TGT, Gain: 5.619999999999997

Stock with the maximum gain: ADP, Gain: 6.0249999999999915

Stock with the maximum gain: GWW, Gain: 32.66999999999999

Stock with the maximum gain: GWW, Gain: 32.66999999999999

Stock with the maximum gain: GWW, Gain: 32.66999999999999

Stock with the maximum gain: GWW, Gain: 32.66999999999999

Stock with the maximum gain: PCLN, Gain: 162.14980000000003

Stock with the maximum gain: PCLN, Gain: 162.14980000000003

Stock with the maximum gain: PCLN, Gain: 183.45019999999988

Stock with the maximum gain: PCLN, Gain: 183.45019999999988

Stock with the maximum gain: PCLN, Gain: 183.45019999999988

Stock with the maximum gain: PCLN, Gain: 183.45019999999988

Stock with the maximum gain: PCLN, Gain: 183.45019999999988

Stock with the maximum gain: PCLN, Gain: 183.45019999999988

Stock with the maximum gain: PCLN, Gain: 183.45019999999988
```

Conclusion

This code is essential for identifying the stock with the maximum gain, providing real-time insights into which stocks have experienced the most significant increase in value. Investors can use this information for making informed investment decisions.

Tasks 4. Implement a control that checks if a stock does not lose too much value in a period of time

Set to Hold Names of Stocks with Large Losses

The code initializes a set named stocks_with_large_losses to hold the names of stocks that have exceeded the maximum allowable percentage loss.

Maximum Allowable Percentage Loss

The variable max_allowable_percentage_loss is defined to set the threshold for the maximum allowable percentage loss. In this code, it's set to 5.0%, but you can modify this value as needed.

Class Definition

The StockState class remains consistent with the previous code. It keeps track of the stock's name, last price, and last window end.

Watermark and Window Operation

Similar to the previous code, the streaming DataFrame is processed with a watermark and window operation to group data by the stock name and time windows.

Updating Stock State

The update method within the StockState class calculates the percentage loss and checks if it exceeds the maximum allowable percentage loss. If the percentage loss for a stock exceeds the threshold, it is added to the stocks_with_large_losses set.

Process Batch Function

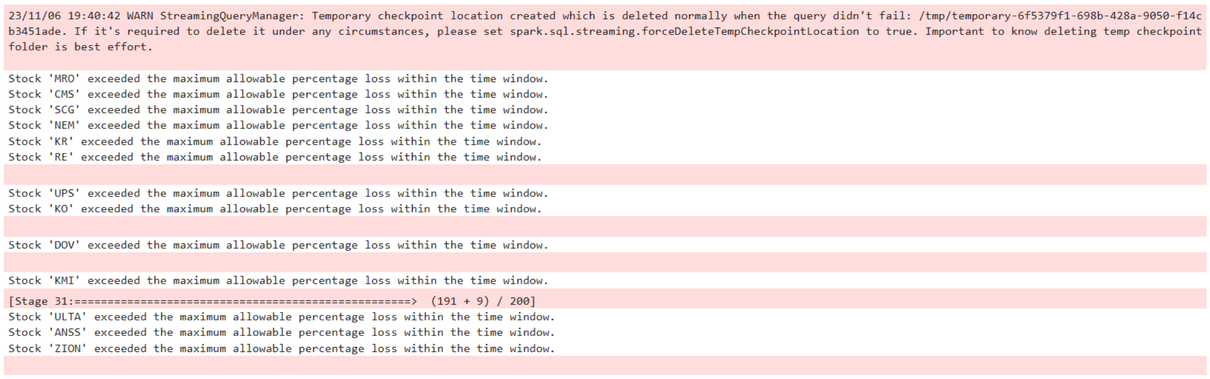
The process_batch function is executed for each batch of data received during streaming.

Logging or Taking Action

Within the process_batch function, for each stock that exceeded the maximum allowable percentage loss, a message is printed to the console.

Clearing the Set

After processing each batch, the stocks_with_large_losses set is cleared to ensure that only new instances of large losses are recorded.



```
23/11/06 19:40:42 WARN StreamingQueryManager: Temporary checkpoint location created which is deleted normally when the query didn't fail: /tmp/temporary-6f5379f1-698b-428a-9050-f14c
b3451ade. If it's required to delete it under any circumstances, please set spark.sql.streaming.forceDeleteTempCheckpointLocation to true. Important to know deleting temp checkpoint
folder is best effort.

Stock 'MRO' exceeded the maximum allowable percentage loss within the time window.
Stock 'CMS' exceeded the maximum allowable percentage loss within the time window.
Stock 'SCG' exceeded the maximum allowable percentage loss within the time window.
Stock 'NEM' exceeded the maximum allowable percentage loss within the time window.
Stock 'KR' exceeded the maximum allowable percentage loss within the time window.
Stock 'RE' exceeded the maximum allowable percentage loss within the time window.

Stock 'UPS' exceeded the maximum allowable percentage loss within the time window.
Stock 'KO' exceeded the maximum allowable percentage loss within the time window.

Stock 'DOV' exceeded the maximum allowable percentage loss within the time window.

Stock 'KMI' exceeded the maximum allowable percentage loss within the time window.
[Stage 31:=====================================================>  (191 + 9) / 200]
Stock 'ULTA' exceeded the maximum allowable percentage loss within the time window.
Stock 'ANSS' exceeded the maximum allowable percentage loss within the time window.
Stock 'ZION' exceeded the maximum allowable percentage loss within the time window.
```

Conclusion

This code allows you to monitor and identify stocks that have experienced significant percentage losses within the specified time windows, helping you identify potential issues in real-time.