

# Real-Time Point-of-Sale Analytics using Apache Kafka and PySpark

Gabriel McReynolds & Nathan Jarvis



# Introduction

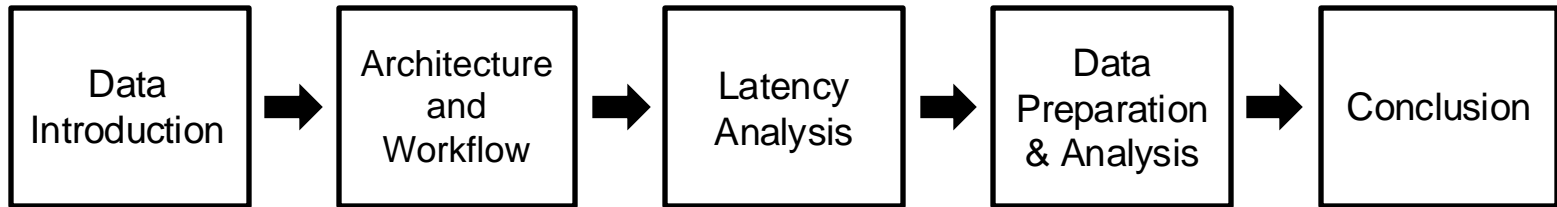
- Point-of-sale analytics is the process of collecting and analyzing data from the processing of transactions at a retail store
- With volatility in the market and narrowing margins in retail, POS analytics is critical for retailers to ensure they are running their inventory management program as effectively as possible
- If a POS system stores and reports data about inventory, retailers can have a better idea of what they're selling, what they're storing and what isn't moving.

# Introduction Cont.

- Derived from Databricks Solution Accelerator Real-Time Point-Of-Sale Analytics
- Instead of using Delta Live Tables, transaction data is captured via Kafka and streamed in real-time ensuring low-latency processing
- The project diverges from the original framework by replacing managed services with a custom Python server setup for cost efficiency and greater control over event handling.
- Provides a scalable framework adaptable to diverse retail environments, making it suitable for dynamic, fast-paced operations



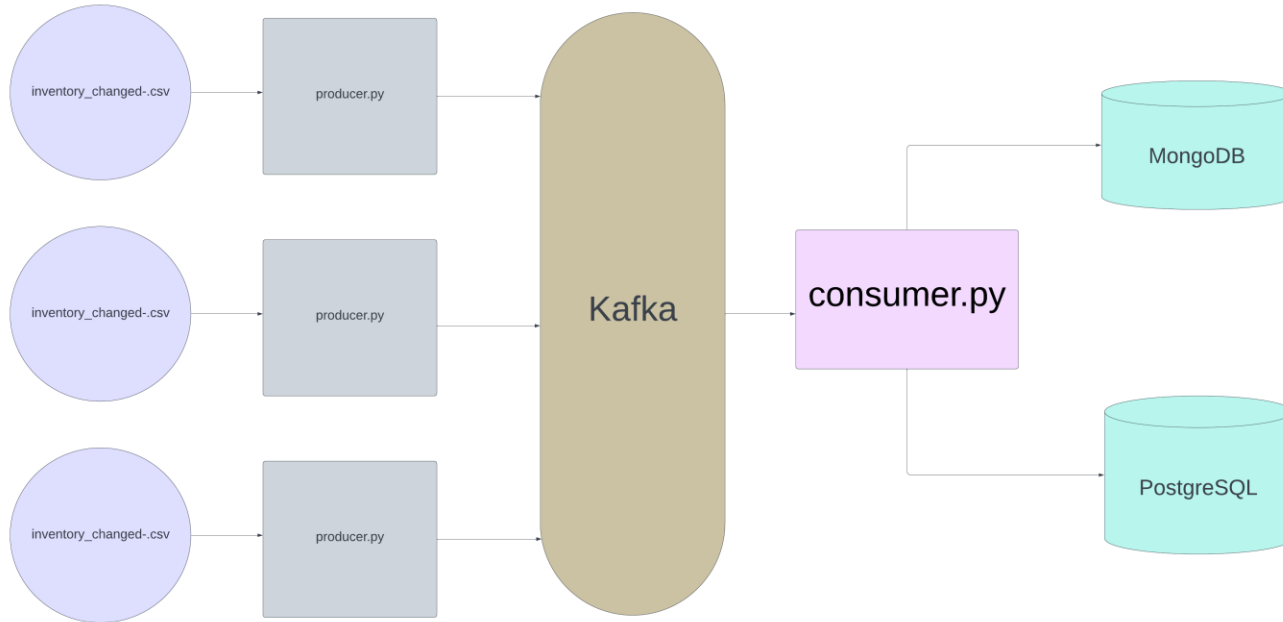
# Overview



# Understanding the Data

- Inventory Snapshot Files: Record inventory levels for items at specific times for online and in-store locations
- Inventory Change Files: Track changes to items in inventory (sold, shrink, restock, bopis) with timestamps, quantities, and transaction ID
- Metadata Files
  - Change Types: Maps reasons for inventory changes
  - Items: Details like item name, supplier, and safety stock
  - Store: Store names and IDs or online

# Architecture and Workflow



# Latency Analysis

## MongoDB

- Batch
- Single

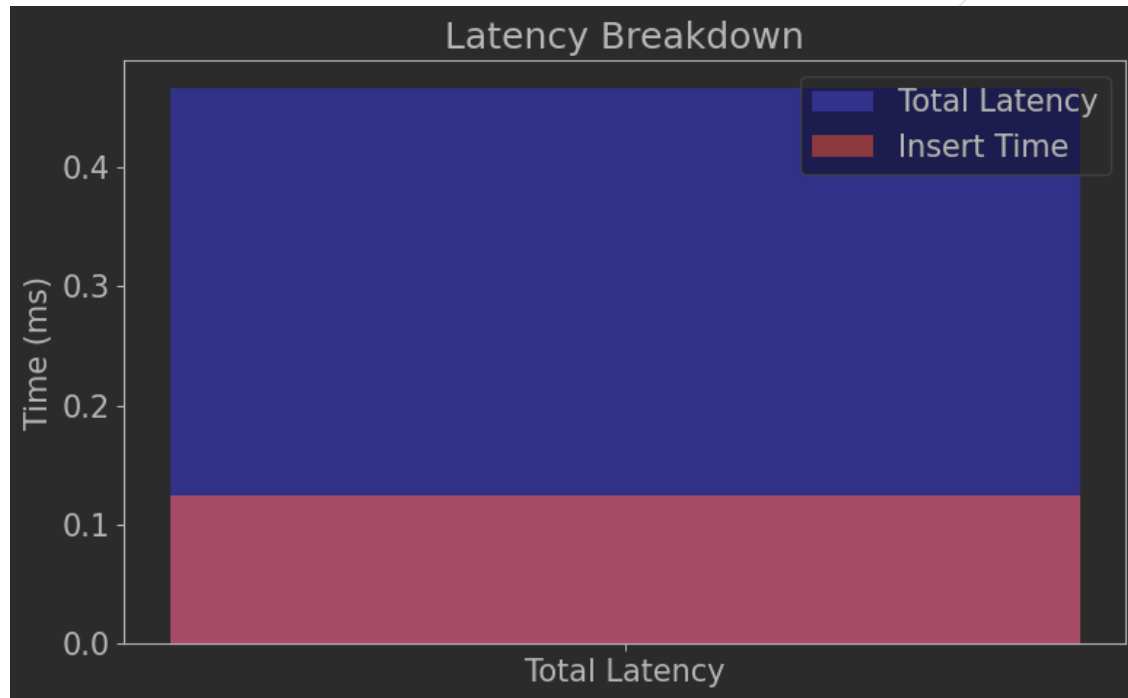
## SQL (Postgres)

- Batch
- Single

```
aggregated_df = df.groupBy("store_id","item_id").agg(pyspark_sum("quantity") \
.alias("delta_quantity")).orderBy("delta_quantity", ascending=True)
```

# Overall Latency Breakdown

- Latency Calculation is averaged over all rows in a batch (~3,000 rows)
- Insert time is not a huge portion of latency



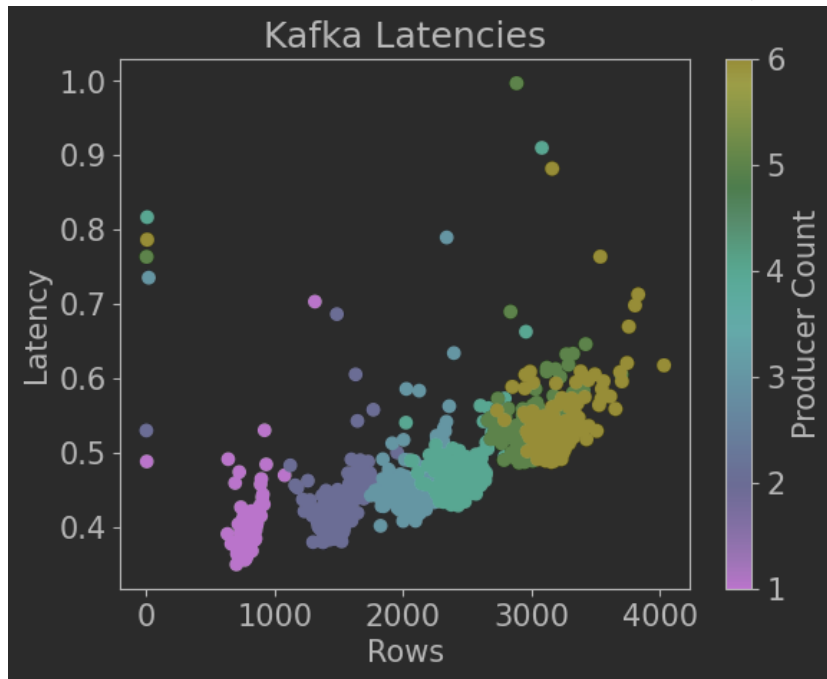


# Latency Collection

- producer.py adds a sent\_time to each row
- Time for batch = processing\_end – avg\_sent\_time
- Each dot represents one Kafka batch
- Increased strain by running multiple producers (color)

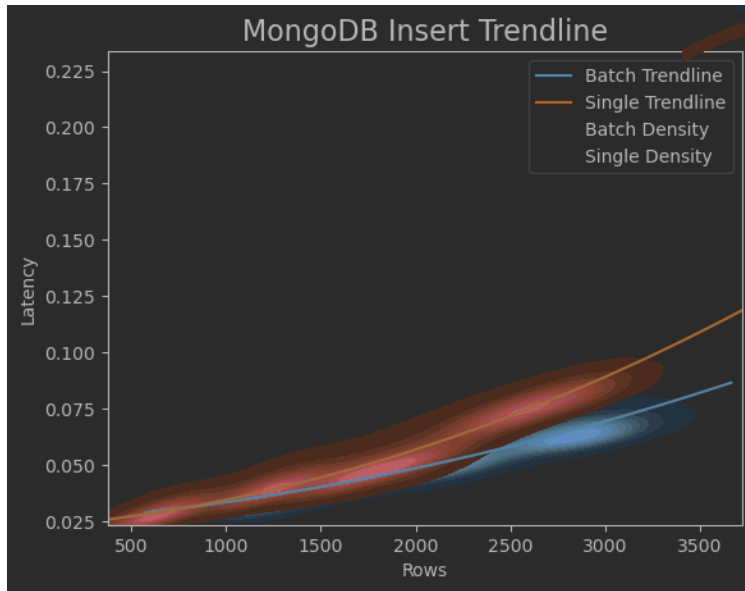


```
avg_sent_time = df.agg(avg("sent_time")).first()[0]
```

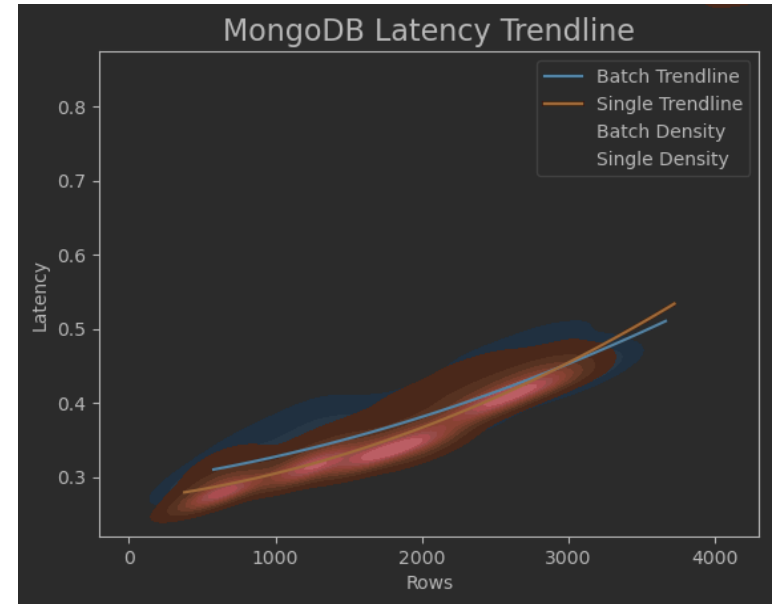


# MongoDB Single vs Batch

- Batch takes less time to insert (fewer inserted rows)



- Single takes less as batch size increases



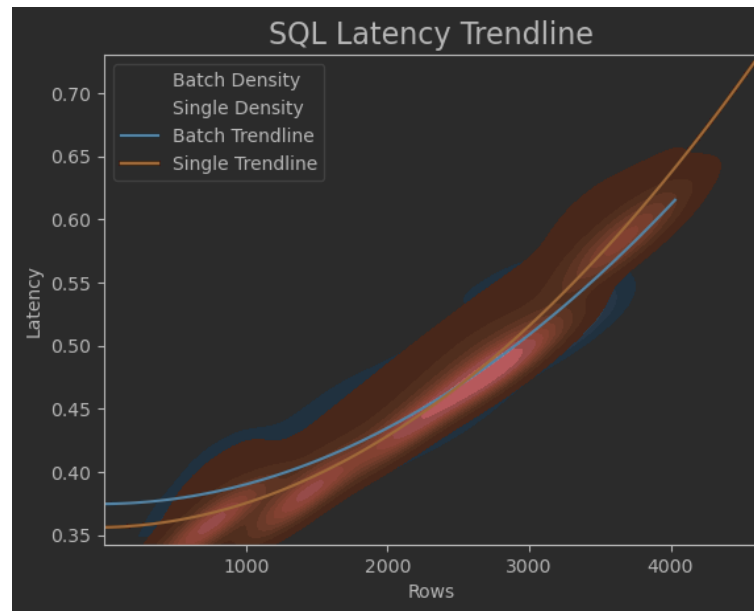
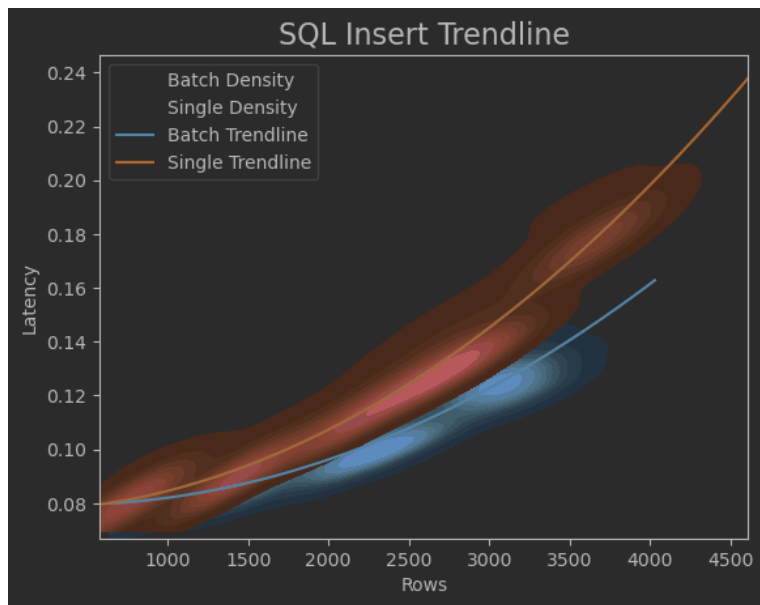
# MongoDB Aggregate Query

- Single takes ~2.4 seconds
- Batch takes ~1.6 seconds

```
db.single_latency.aggregate([
  {
    $group: {
      _id: { item_id: "$item_id", store_id: "$store_id" },
      delta_quantity: { $sum: "$quantity" }
    }
  }
])
```

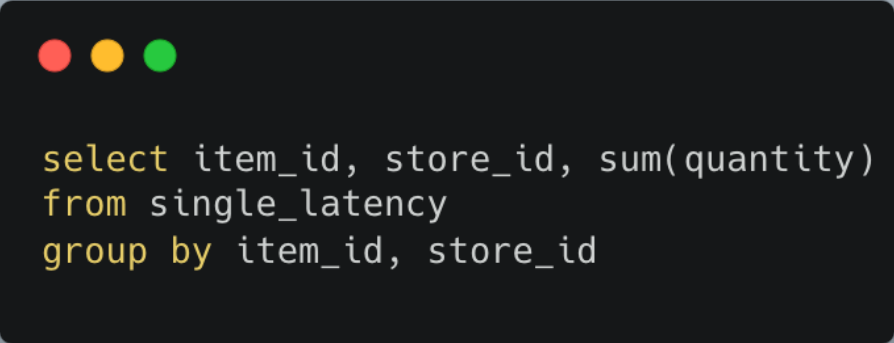
# PostgreSQL Single vs Batch

Similar story to MongoDB



# PostgreSQL Aggregate Query

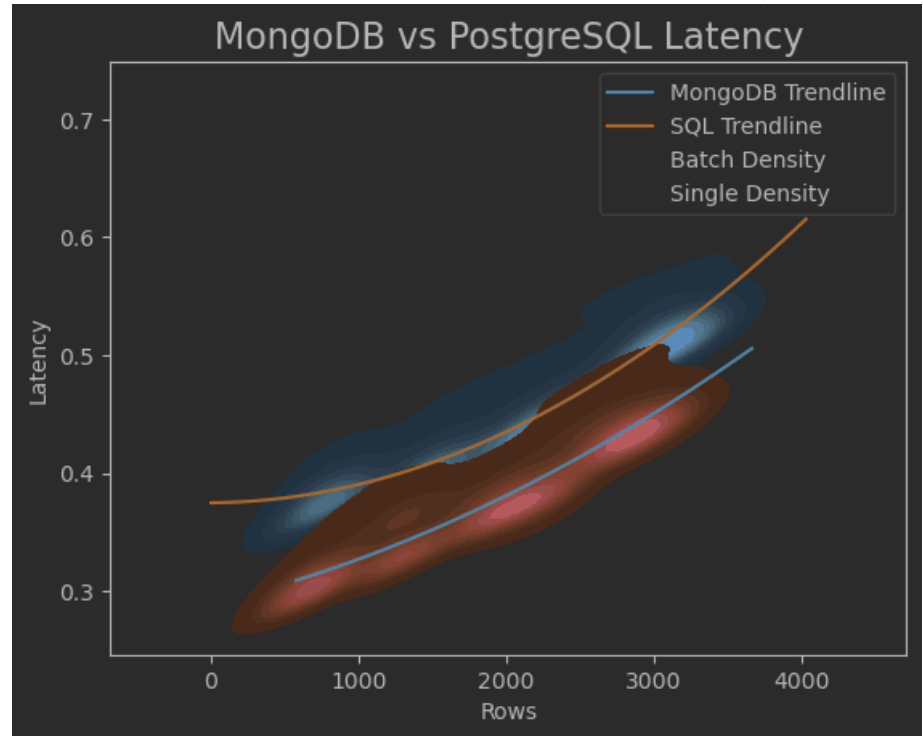
- Single takes ~1.5 seconds
- Batch takes ~.8 seconds
- Significantly faster than MongoDB



```
select item_id, store_id, sum(quantity)
from single_latency
group by item_id, store_id
```

# MongoDB vs PostgreSQL

- SQL is more influenced by batch size
- MongoDB has faster insert times
- SQL has faster aggregation times



# Updating the Data for Analysis

- A combined CSV was created by processing the results of snapshots and change files from Kafka topics, joined with metadata and the timestamp
- Important Fields
  - Item ID
  - Date Time
  - Quantity
  - Source Type (snapshot\_only or snapshot\_w\_change)
  - Store Name (names of stores or online)
  - Action (the corresponding change type, if no change type, then it is inventory report)
  - Final Quantity (quantity from snapshot + quantity from change)

# Analysis

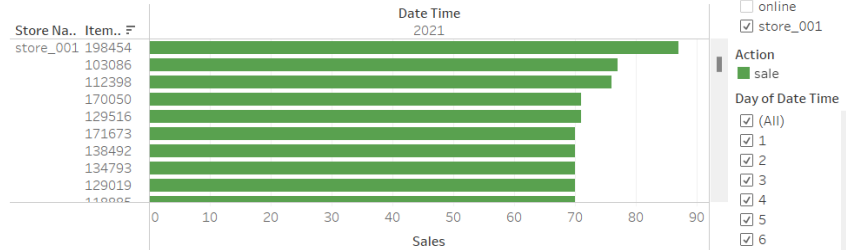
- Compare sales per store to see items that are performing well
- Compare sales per store by day and hour

Sales

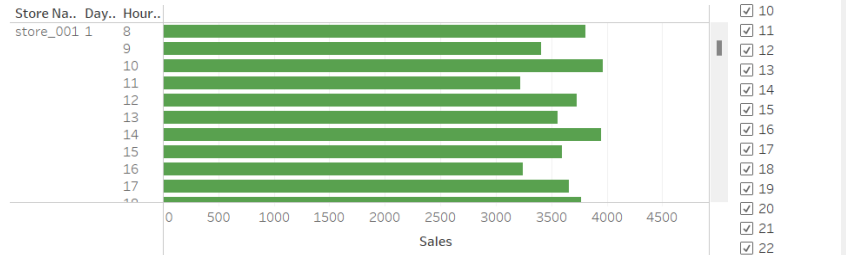
```
IF [Change Type] = "sale" OR [Change Type] = "bopis" THEN
    ABS([Quantity])
ELSE
    NULL
END
```

## Sales Dashboard

### Top Items by Sales



### Sales by Store



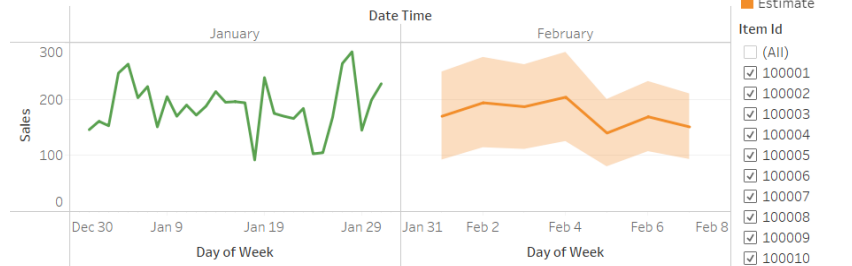


# Analysis

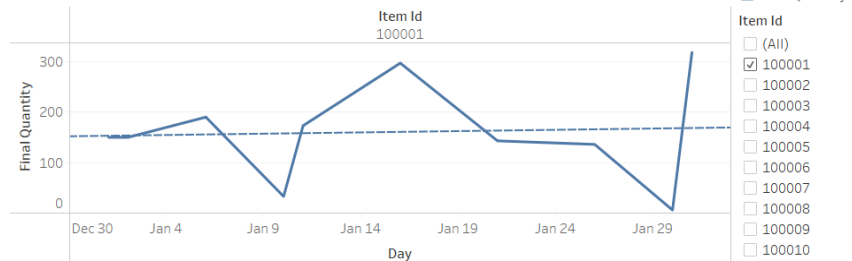
- Filter between items to get forecasted view of sales for each item for the next month
- Can compare with current final quantity

## Forecasting and Trends for Items

Forecast of Sales per Day



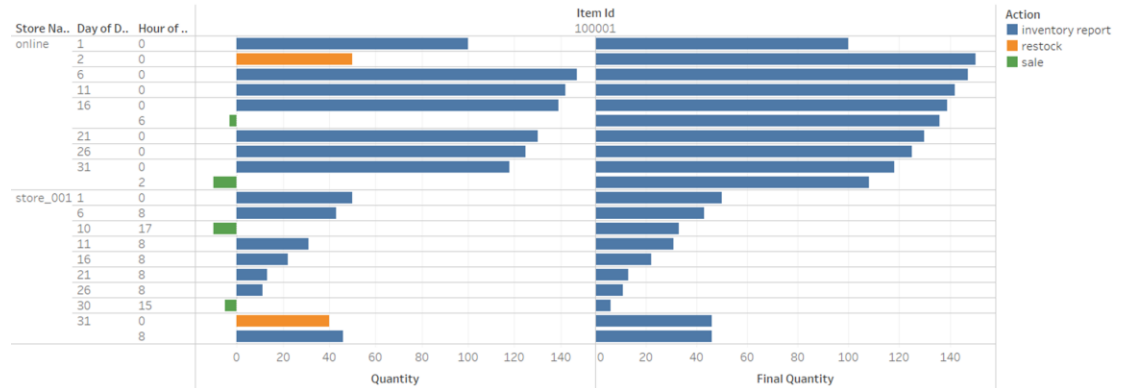
Final Quantity Trends by Item



# Analysis

- If extra insight about an item is needed, you can filter to that item and see specific quantity and the change type and compare to final quantity

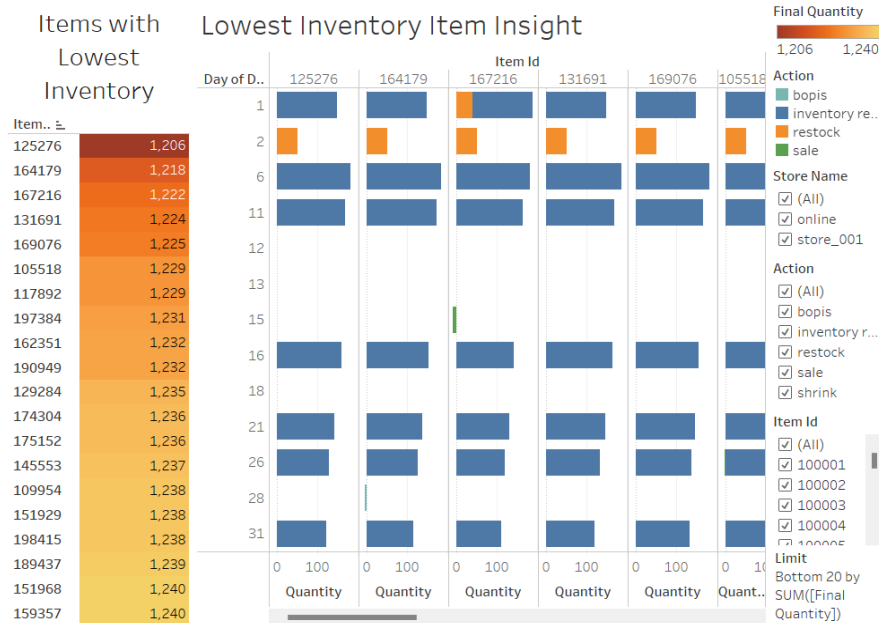
Total Item View By Hour



# Analysis

- See 20 items with the lowest current inventory
- Get info on those items to see their activity per day and hour

## Lowest Quantity Insight Dashboard



# Real-Time Streaming

- We can stream this all together by creating a workflow with a trigger that runs every hour
- Every hour it will rerun our producer to bring in the new data from the POS system and update our topics
- After running the producer, a trigger will activate to run the consumer. After creating the file to export there will be a pipeline to a tableau file with visuals that can be refreshed to get real-time analytics

# Future Improvements

- Investigate if SQL and MongoDB indexes can reduce query time
- Create better forecasting in Tableau after data grows to ample size and spans over months
- Put items into clusters to perform analysis to get better insights on similar items
- Track seasonality to better understand how to items and clusters are moving around different times of the year

# Conclusion

- Batch processing Kafka stream is most efficient for saving and querying data
- MongoDB inserts faster but SQL can query faster for aggregate query
- The data needs preparation and aggregating before analysis can be done
- There is a lot that can be forecasted currently, but more descriptive analytics needs more time and data

Questions?