SAMSUNG

Real-time Big Data Processing Pipeline with PySpark

Group 1

Do Kien Hung (darktheDE) - Team Leader, Data Engineer

Nguyen Van Quang Duy (QuangDuyReal) - Streaming & Analytics Engineer

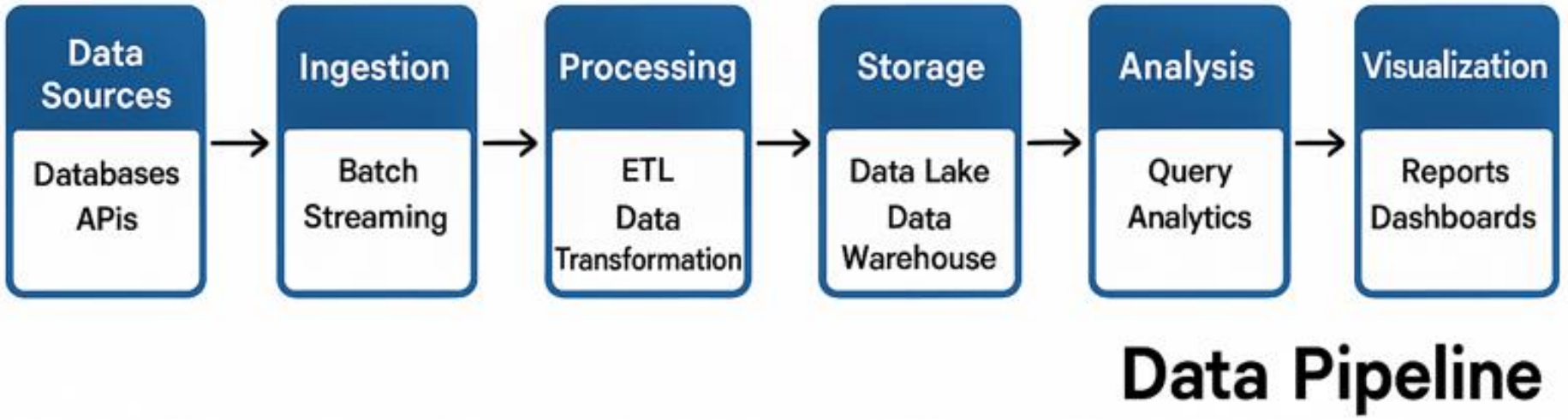
GitHub: QuangDuyReal/nyc-taxi-trip-analysis

Big Data Course

Together for Tomorrow!
Enabling People
Education for Future Generations

Project Objective: Building a Scalable, Fault-Tolerant Data Pipeline

- Core Goal: To design and implement an end-to-end Big Data pipeline using Apache PySpark to analyze NYC Taxi trip data.
  - **Scalability:** Capable of handling terabytes of data.
  - **Fault-Tolerance:** Resilient to failures during processing.
  - **Data Quality:** Ensuring trusted, analysis-ready data via the **Medallion Architecture**.



Data Source: NYC Taxi & Limousine Commission (TLC)

Dataset Characteristics

- **Source:** Official NYC TLC Trip Record Data (2023.1 and 2024.1)
- **Format:** Monthly partitioned Parquet files.
- **Scale:** ~10-15 million trips per month (~1-2 GB/file).

Field Name	Description	Role in Analysis
tpep_pickup_datetime	Trip start time	Time-series analysis
PULocationID	Pickup location ID	Geographic analysis
trip_distance	Trip distance in miles	Performance metrics
total_amount	Total fare amount	Financial analysis
payment_type	Method of payment	Passenger behavior analysis

Problem Statement & Project Objectives

Problem Statement:

- Raw taxi trip data is inconsistent, contains errors, and lacks a unified structure.
- How to transform this raw data into valuable business insights?
- How to support both historical analysis (strategic) and real-time operational monitoring (tactical)?

Project Objectives:

- Build a reliable, automated data pipeline.
- Implement the Medallion Architecture (Bronze, Silver, Gold) for progressive data refinement.
- Integrate both Batch and Streaming data processing.
- Deliver clean, analysis-ready data.

Technology Stack

- **Core Framework:** Apache PySpark 3.4.1
- **Programming Language:** Python 3.8+
- **Architecture:** Medallion (Bronze, Silver, Gold)
- **Processing Models:** Batch Processing & Structured Streaming
- **Data Format:** Parquet, Delta Lake
- **Supporting Libraries:** Pandas, Matplotlib, Seaborn, Plotly
- **Development Environment:** Jupyter Notebook, VS Code, Terminal

Project Prerequisites

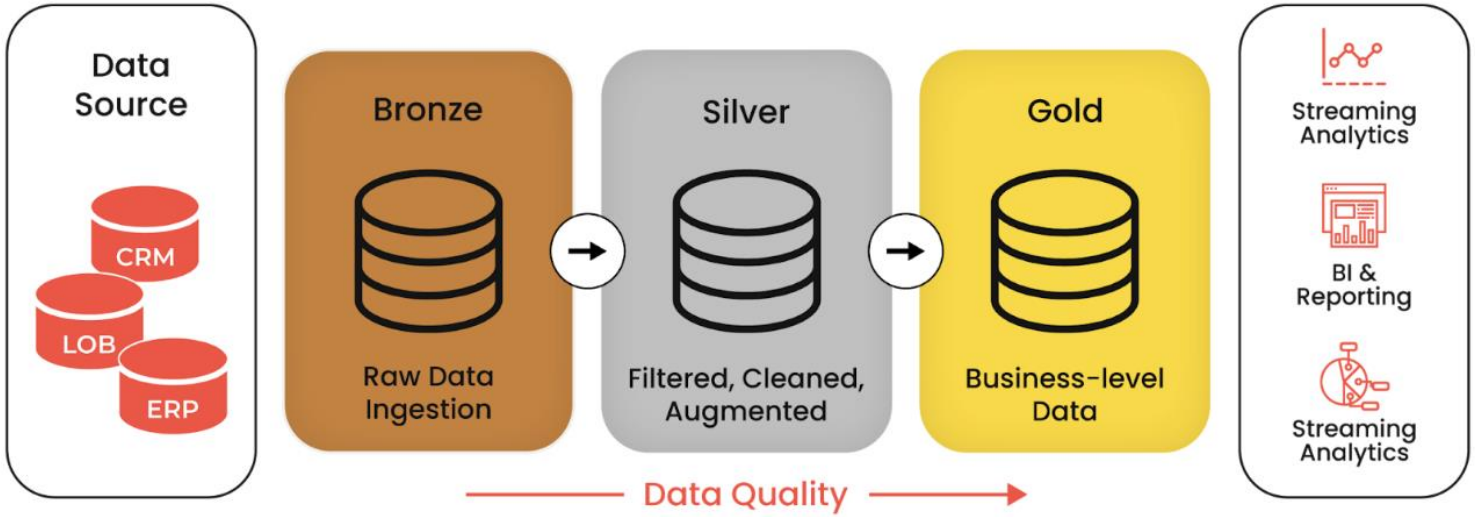
- **Technical Requirements to Run This Project**
- **Repository:** https://github.com/QuangDuyReal/nyc-taxi-trip-analysis
- **Core Components:**
  - Java 8+ (LTS recommended)
  - Apache Spark 3.4+
  - Python 3.8+ & pip
- **Environment Specific:**
  - **For Windows:** Hadoop binaries (winutils.exe, hadoop.dll) are required for Spark to access the local file system.
- **Dependencies:**
  - All Python libraries are listed in requirements.txt.
- **Setup:**
  - Environment variables (JAVA_HOME, SPARK_HOME, HADOOP_ HOME) must be configured correctly.

Project Structure

```
nyc_taxi_pipeline/
├── data/
│   ├── raw/ # Dữ liệu Parquet thô từ nguồn
│   ├── bronze/ # Dữ liệu lớp Bronze
│   ├── silver/ # Dữ liệu lớp Silver
│   └── gold/ # Dữ liệu lớp Gold
│   └── streaming_input/ # Dữ liệu để mô phỏng luồng
│   └── streaming_output/ # Kết quả từ job streaming
├── src/
│   ├── bronze_layer.py # Logic cho pipeline Bronze
│   ├── silver_layer.py # Logic cho pipeline Silver
│   ├── gold_layer.py # Logic cho pipeline Gold
│   ├── streaming_pipeline.py # Logic cho pipeline Streaming
│   ├── data_quality.py # Framework kiểm tra chất lượng dữ liệu
│   ├── utils.py # Các hàm tiện ích
│   └── main_pipeline.py # Script chính để thực thi toàn bộ pipeline
├── notebooks/
│   ├── 01_data_exploration.ipynb
│   └── 02_analytics_dashboard.ipynb # Notebook để visualize kết quả
├── config/
│   ├── spark_config.py
│   └── pipeline_config.yaml
├── checkpoint/ # Checkpoints cho streaming
├── requirements.txt # Các thư viện Python cần thiết
└── README.md # Tài liệu hướng dẫn này
```

System Architecture: The Medallion Model

- Source (Parquet) → [Bronze: Raw Data] → [Silver: Validated Data] → [Gold: Business Aggregates] → BI & Analytics
  - **Bronze:** Immutable, Partitioned, Metadata.
  - **Silver:** Cleaned, Validated, Feature Engineering.
  - **Gold:** Aggregated, Business-Ready, Performance-Optimized.



Pipeline Implementation Highlights

- Bronze Layer: Data Ingestion
  - Ingested raw Parquet files, preserved original data, and added ingestion metadata. Used Delta Lake for reliability.
- Silver Layer: Data Cleaning & Enrichment
  - Applied data quality rules (e.g., trip_ distance > 0).
  - Performed **Feature Engineering** to create valuable columns like trip_duration_minutes and speed-mph.
- Gold Layer: Business Aggregations
  - Created four key aggregated tables for analysis: Hourly Stats, Location Hotspots, Payment Analytics, and Vendor Performance.

Batch Processing: Bronze Layer

- **Key Tasks:**
  - Read all Parquet files from the raw data source.
  - Use mergeSchema to handle schema evolution automatically.
  - Add metadata columns: ingestion_timestamp and source_file for data lineage.
- **Evidence from Data:**
  - **Input:** Multiple Parquet files with varying schemas.
  - **Output:** A unified Delta table with 867,468 records.
  - Lower non-null counts in columns like passenger_count prove mergeSchema handled inconsistent files.
  - New columns ingestion_timestamp & source_file were successfully added.

Batch Processing: Bronze Layer

```
=== BRONZE LAYER DATA ===
            VendorID       tpep_pickup_datetime  ...    Airport_fee           ingestion_timestamp
count  867468.000000                     867468  ...  727306.000000                        867468
mean        1.741933  2024-01-26 10:03:06.784384  ...       0.121351  2025-07-29 11:44:54.398418432
min         1.000000        2024-01-01 00:00:58  ...      -1.750000     2025-07-29 11:44:54.398418
25%         1.000000  2024-01-25 16:13:23.750000  ...       0.000000  2025-07-29 11:44:54.398417920
50%         2.000000  2024-01-27 15:59:42.500000  ...       0.000000  2025-07-29 11:44:54.398417920
75%         2.000000        2024-01-29 20:07:28  ...       0.000000  2025-07-29 11:44:54.398417920
max         6.000000        2024-01-31 23:59:55  ...       1.750000     2025-07-29 11:44:54.398418
std         0.444368                        NaN  ...       0.455809                          NaN
```

Batch Processing: Bronze Layer

```
[8 rows x 19 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 867468 entries, 0 to 867467
Data columns (total 21 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   VendorID               867468 non-null  int32
 1   tpep_pickup_datetime   867468 non-null  datetime64[us]
 2   tpep_dropoff_datetime  867468 non-null  datetime64[us]
 3   passenger_count        727306 non-null  float64
 4   trip_distance          867468 non-null  float64
 5   RatecodeID             727306 non-null  float64
 6   store_and_fwd_flag     727306 non-null  object
 7   PULocationID           867468 non-null  int32
 8   DOLocationID           867468 non-null  int32
 9   payment_type           867468 non-null  int64
 10  fare_amount            867468 non-null  float64
 11  extra                  867468 non-null  float64
 12  mta_tax                867468 non-null  float64
 13  tip_amount             867468 non-null  float64
 14  tolls_amount           867468 non-null  float64
 15  improvement_surcharge  867468 non-null  float64
 16  total_amount           867468 non-null  float64
 17  congestion_surcharge   727306 non-null  float64
 18  Airport_fee            727306 non-null  float64
 19  ingestion_timestamp    867468 non-null  datetime64[ns]
 20  source_file            867468 non-null  object
dtypes: datetime64[ns](1), datetime64[us](2), float64(12), int32(3), int64(1), object(2)
memory usage: 129.1+ MB
None
```

Batch Processing: Silver Layer

- Key Tasks:
  - **Cleansing & Validation:** Remove records with invalid business logic (e.g., trip_distance <= 0, passenger_count out of range [1, 8]).
  - **Enrichment (Feature Engineering):**
    - tripdurationminutes: Calculate trip duration.
    - speedmph: Calculate average speed.
    - tippercentage: Calculate tip ratio.
    - qualityscore: Assess record quality.
- Evidence from Data:
  - Record count reduced from **867k** (Bronze) to **~691k** (Silver), removing ~20% of invalid data.
  - All columns now have consistent non-null counts, indicating clean data.
  - New features successfully created: trip_duration_minutes, speed_ mph, etc.

Batch Processing: Silver Layer

```
=== SILVER LAYER DATA ===
           VendorID        tpep_pickup_datetime  ... quality_score          processing_timestamp
count  691662.000000                      691662  ...      691662.0                        691662
mean        1.764586  2024-01-28 06:43:02.846191  ...           1.0  2025-07-29 11:45:14.598436864
min         1.000000         2024-01-24 14:01:32  ...           1.0     2025-07-29 11:45:14.598439
25%         2.000000  2024-01-26 11:11:12.250000  ...           1.0  2025-07-29 11:45:14.598438912
50%         2.000000  2024-01-28 00:00:19.500000  ...           1.0  2025-07-29 11:45:14.598438912
75%         2.000000  2024-01-30 09:25:32.750000  ...           1.0  2025-07-29 11:45:14.598438912
max         2.000000         2024-01-31 23:59:55  ...           1.0     2025-07-29 11:45:14.598439
std         0.424258                         NaN  ...           0.0                         NaN
```

Batch Processing: Silver Layer

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 691662 entries, 0 to 691661
Data columns (total 30 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   VendorID               691662 non-null  int32
 1   tpep_pickup_datetime   691662 non-null  datetime64[us]
 2   tpep_dropoff_datetime  691662 non-null  datetime64[us]
 3   passenger_count        691662 non-null  int64
 4   trip_distance          691662 non-null  float64
 5   RatecodeID             691662 non-null  int64
 6   store_and_fwd_flag     691662 non-null  object
 7   PULocationID           691662 non-null  int32
 8   DOLocationID           691662 non-null  int32
 9   payment_type           691662 non-null  int64
 10  fare_amount            691662 non-null  float64
 11  extra                  691662 non-null  float64
 12  mta_tax                691662 non-null  float64
 13  tip_amount             691662 non-null  float64
 14  tolls_amount           691662 non-null  float64
 15  improvement_surcharge  691662 non-null  float64
 16  total_amount           691662 non-null  float64
 17  congestion_surcharge   691662 non-null  float64
 18  Airport_fee            691662 non-null  float64
 19  year                   691662 non-null  int32
 20  month                  691662 non-null  int32
 21  ingestion_timestamp    691662 non-null  datetime64[ns]
 22  source_file            691662 non-null  object
 23  trip_duration_minutes  691662 non-null  float64
 24  pickup_hour            691662 non-null  int32
 25  pickup_day_of_week     691662 non-null  int32
 26  speed_mph              691662 non-null  float64
 27  tip_percentage         691662 non-null  float64
 28  quality_score          691662 non-null  float64
 29  processing_timestamp   691662 non-null  datetime64[ns]
dtypes: datetime64[ns](2), datetime64[us](2), float64(14), int32(7), int64(3), object(2)
```

Batch Processing: Gold Layer

- Key Tasks:
  - Aggregate data from the Silver Layer into business-centric KPI tables.
  - Example: hourly_stats table for performance analysis by hour of the day.
- Evidence from Data:
  - The hourly_stats table contains **24 rows**, one for each hour.
  - Actionable Insights:
    - Identify peak hours for revenue and trip volume.
    - Analyze trends in average fare and duration throughout the day.
    - Example: total_ trips peaks at hour 18 (191,880 trips), providing insights for driver allocation.

Batch Processing: Gold Layer

```
=== GOLD LAYER DATA ===
       pickup_hour    total_trips   avg_fare    avg_tip  ...  total_revenue    max_fare   min_fare          processing_timestamp
count    24.000000      24.000000  24.000000  24.000000  ...   2.400000e+01   24.000000  24.000000                            24
mean     11.500000  111723.000000  19.090810   3.415876  ...   3.068058e+06  570.440417   2.178333  2025-07-29 11:46:06.024773888
min       0.000000   12429.000000  16.442981   2.987693  ...   4.062820e+05  292.610000   1.010000     2025-07-29 11:46:06.024774
25%       5.750000   62079.750000  17.801164   3.195237  ...   1.741816e+06  453.272500   1.010000  2025-07-29 11:46:06.024773888
50%      11.500000  128594.500000  18.312622   3.429632  ...   3.454927e+06  520.890000   1.010000  2025-07-29 11:46:06.024773888
75%      17.250000  159628.000000  19.348529   3.585804  ...   4.293337e+06  699.302500   2.760000  2025-07-29 11:46:06.024773888
max      23.000000  191880.000000  27.862577   4.034232  ...   5.312603e+06  940.930000   6.100000     2025-07-29 11:46:06.024774
std       7.071068   59766.723654   2.406652   0.267465  ...   1.646398e+06  172.912298   1.656063                          NaN
```

Batch Processing: Gold Layer

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24 entries, 0 to 23
Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   pickup_hour          24 non-null     int32
 1   total_trips          24 non-null     int64
 2   avg_fare             24 non-null     float64
 3   avg_tip              24 non-null     float64
 4   avg_distance         24 non-null     float64
 5   avg_duration         24 non-null     float64
 6   avg_passengers       24 non-null     float64
 7   total_revenue        24 non-null     float64
 8   max_fare             24 non-null     float64
 9   min_fare             24 non-null     float64
 10  processing_timestamp  24 non-null     datetime64[ns]
 11  layer                24 non-null     object
dtypes: datetime64[ns](1), float64(8), int32(1), int64(1), object(1)
memory usage: 2.3+ KB
```

Batch Pipeline Results: Historical Data Ready

- Status: COMPLETED SUCCESSFULLY ✓
- Outcome: 4 analysis-ready Gold tables created.
  - hourlystats: 29 records
  - locationstats: 4,753 records
  - paymentstats: 10 records
  - vendorstats: 6 records
- Business Value:
  - Provides a solid foundation for historical analysis, strategic reporting, and BI dashboards.
  - Enables deep dives into performance by hour, location hotspots, payment trends, and vendor efficiency.

Real-time Processing: The Streaming Pipeline

- Key Tasks:
  - Simulate a continuous stream of new taxi trip data.
  - Apply a micro-batch Bronze -> Silver -> Gold process on the stream.
  - Use 'Sliding Windows' to calculate near real-time KPIs.
- Evidence from Data:
  - STREAMING INPUT: Sample of incoming raw events (148k records – 10 minutes of streaming).
  - STREAMING OUTPUT: Aggregated results per time window.
  - Real-time Insights:
    - Continuously updated metrics: tripcount, totalrevenue, avgspeed.
    - Enables detection of emerging hotspots or traffic congestion (via avgspeed drop).

Real-time Processing: Streaming Input

```
=== STREAMING INPUT DATA ===
           VendorID       tpep_pickup_datetime      tpep_dropoff_datetime  ...   total_amount  congestion_surcharge    Airport_fee
count  148231.000000                     148231                     148231  ...  148231.000000         141260.000000  141260.000000
mean        1.753904  2024-01-17 00:57:27.393864 2024-01-17 01:13:10.201293  ...      26.786814              2.255911       0.140448
min         1.000000        2023-12-31 23:56:45        2024-01-01 00:00:28  ...    -591.000000             -2.500000      -1.750000
25%         2.000000  2024-01-09 16:12:05.500000 2024-01-09 16:27:15.500000  ...      15.300000              2.500000       0.000000
50%         2.000000        2024-01-17 09:44:16        2024-01-17 10:00:41  ...      20.000000              2.500000       0.000000
75%         2.000000  2024-01-24 18:26:05.500000        2024-01-24 18:41:13  ...      28.560000              2.500000       0.000000
max         6.000000        2024-01-31 23:59:22        2024-02-02 13:56:52  ...     606.390000              2.500000       1.750000
std         0.432456                        NaN                        NaN  ...      23.030613              0.826723       0.486368
```

Real-time Processing: Streaming Input

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 148231 entries, 0 to 148230
Data columns (total 19 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   VendorID               148231 non-null  int32
 1   tpep_pickup_datetime   148231 non-null  datetime64[us]
 2   tpep_dropoff_datetime  148231 non-null  datetime64[us]
 3   passenger_count        141260 non-null  float64
 4   trip_distance          148231 non-null  float64
 5   RatecodeID             141260 non-null  float64
 6   store_and_fwd_flag     141260 non-null  object
 7   PULocationID           148231 non-null  int32
 8   DOLocationID           148231 non-null  int32
 9   payment_type           148231 non-null  int64
 10  fare_amount            148231 non-null  float64
 11  extra                  148231 non-null  float64
 12  mta_tax                148231 non-null  float64
 13  tip_amount             148231 non-null  float64
 14  tolls_amount           148231 non-null  float64
 15  improvement_surcharge  148231 non-null  float64
 16  total_amount           148231 non-null  float64
 17  congestion_surcharge   141260 non-null  float64
 18  Airport_fee            141260 non-null  float64
dtypes: datetime64[us](2), float64(12), int32(3), int64(1), object(1)
memory usage: 19.8+ MB
```

Real-time Processing: Streaming Output

```
=== STREAMING OUTPUT DATA ===
       trip_count  total_revenue  avg_distance  avg_duration  avg_speed
count   44.000000      44.000000     44.000000     44.000000  44.000000
mean    74.000000    1948.828636      3.341674     16.073304  12.249819
std     32.943927     953.127761      1.794698      6.957100   4.757270
min      3.000000     191.440000      1.798611      7.915152   7.569470
25%     48.250000    1245.760000      2.679433     12.894766   9.064826
50%     75.000000    1964.615000      2.985849     14.138044  11.085526
75%     96.750000    2467.057500      3.694000     16.613276  13.665905
max    146.000000    4381.960000     14.133333     51.341892  36.414689
```

Real-time Processing: Streaming Output

```
=== STREAMING OUTPUT DATA ===
       trip_count  total_revenue  avg_distance  avg_duration  avg_speed
count   44.000000      44.000000     44.000000     44.000000  44.000000
mean    74.000000    1948.828636      3.341674     16.073304  12.249819
std     32.943927     953.127761      1.794698      6.957100   4.757270
min      3.000000     191.440000      1.798611      7.915152   7.569470
25%     48.250000    1245.760000      2.679433     12.894766   9.064826
50%     75.000000    1964.615000      2.985849     14.138044  11.085526
75%     96.750000    2467.057500      3.694000     16.613276  13.665905
max    146.000000    4381.960000     14.133333     51.341892  36.414689
```

Real-time Processing: Streaming Output

```
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   window         44 non-null     object 
 1   trip_count     44 non-null     int64  
 2   total_revenue  44 non-null     float64
 3   avg_distance   44 non-null     float64
 4   avg_duration   44 non-null     float64
 5   avg_speed      44 non-null     float64
dtypes: float64(4), int64(1), object(1)
memory usage: 2.2+ KB
None
```

Real-time Results (10-min Stream): Live Business Pulse

Live Monitoring KPIs

Last Window Stats (at 06:50:00)

📊 **74 Trips**

💵 **$2,222.43 Revenue**

⏱️ **11.72 min** Avg. Duration

🚀 **30.30 mph** Avg. Speed

Key Trend Detected

🔥 **Significant Spike Detected!** (Compared to previous window)

📈 **Trips: +48.0%**

💰 **Revenue: +52.8%**

This is a high-value signal indicating a rapid increase in demand or operational activity.

Action: Dispatch more drivers, consider applying surge pricing.

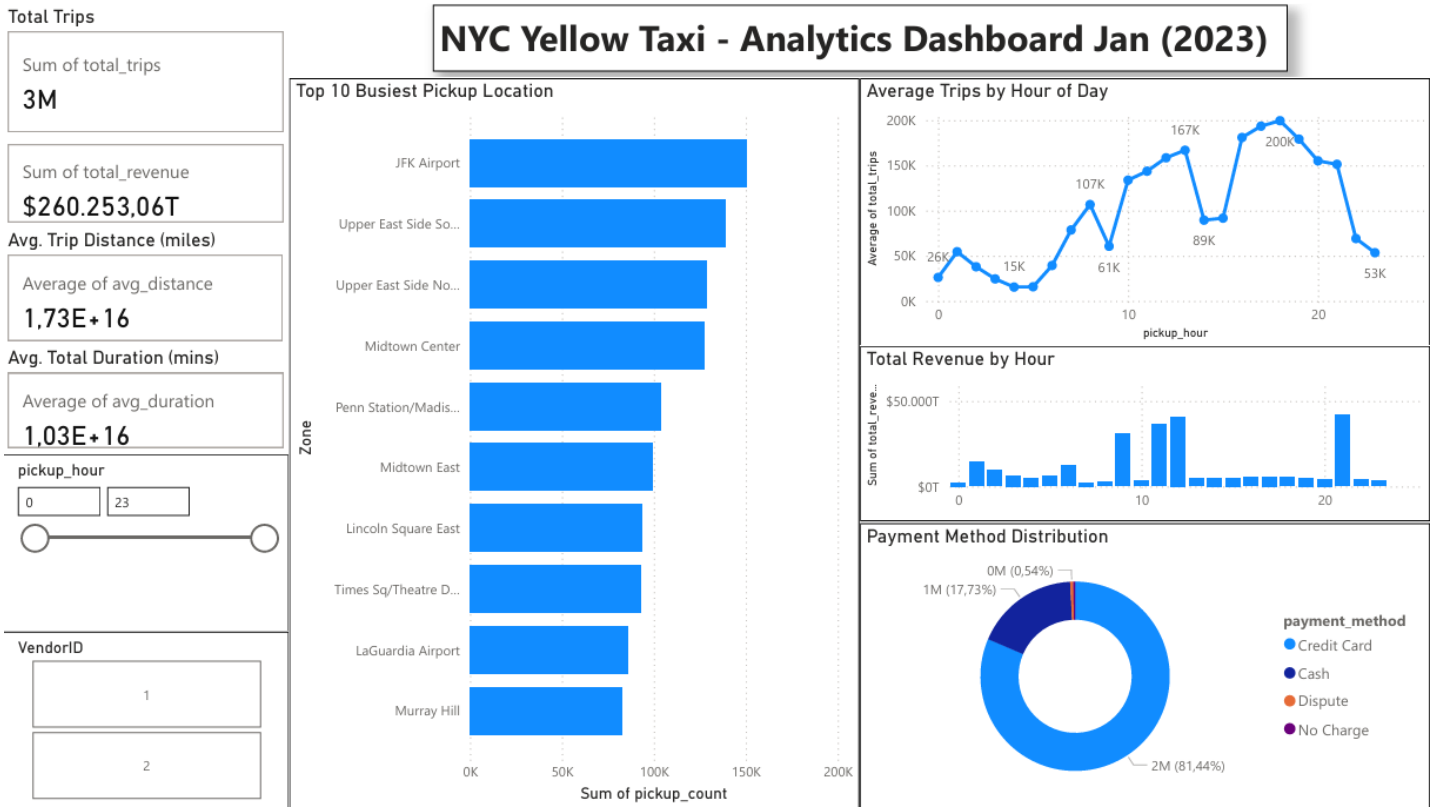
Real-time Results: Deeper Actionable Insights

- 🔑 **Hotspot Identification:**
  - The system identified top pickup locations in real-time (e.g., Location ID 161).
  - **Value:** Helps drivers reduce idle time and find customers faster.
- 💳 **Payment Method Analysis:**
  - Provided a live distribution of payment types (e.g., Credit Card dominance with ~230k trips & $6.4M revenue).
  - **Value:** Useful for real-time transaction monitoring and fraud detection.
- ✔✔ **Proven Stream Processing Capability:**
  - Processed a significant volume of records: location_metrics (97k), payment_metrics (10k), time_metrics (8.9k).
  - **Value:** Confirms the pipeline's ability to handle and accumulate high-volume data streams.

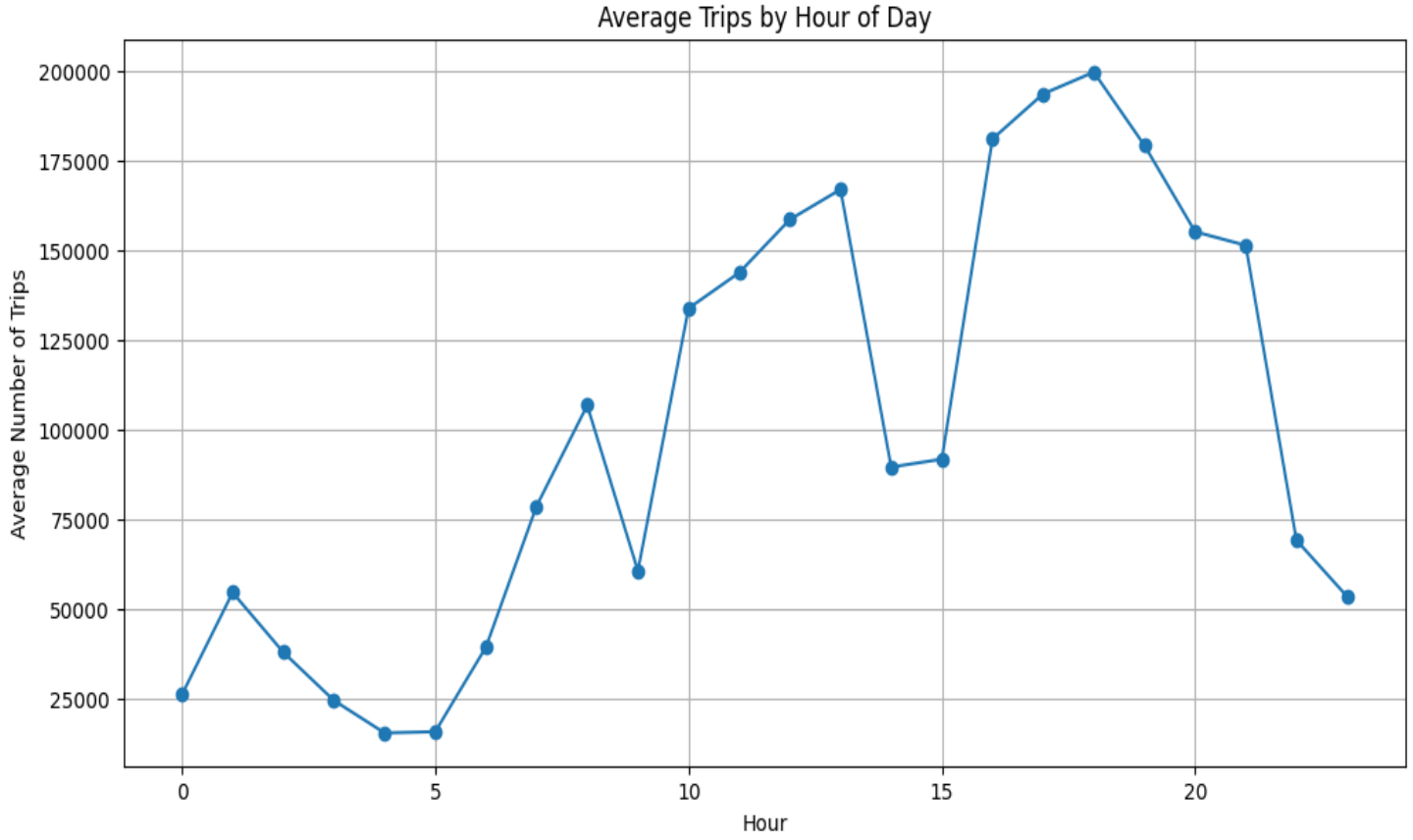
Performance & Infrastructure Analysis

- **Environment:**
  - Windows 10, 12 CPU Cores, 7.37GB RAM
  - Spark 3.5.6 running in local[*] mode.
- **⚠ Key Observation: High Memory Pressure**
  - Memory usage quickly reached **~92%** after the job started.
  - Available memory dropped to as low as **0.59GB**.
- **CPU Fluctuation:**
  - CPU usage peaked at **~41%** during heavy processing, which is normal behavior.
- **Implication:**
  - The system is resource-intensive on a local machine. For production, a **distributed cluster is necessary** for stability and optimal performance.

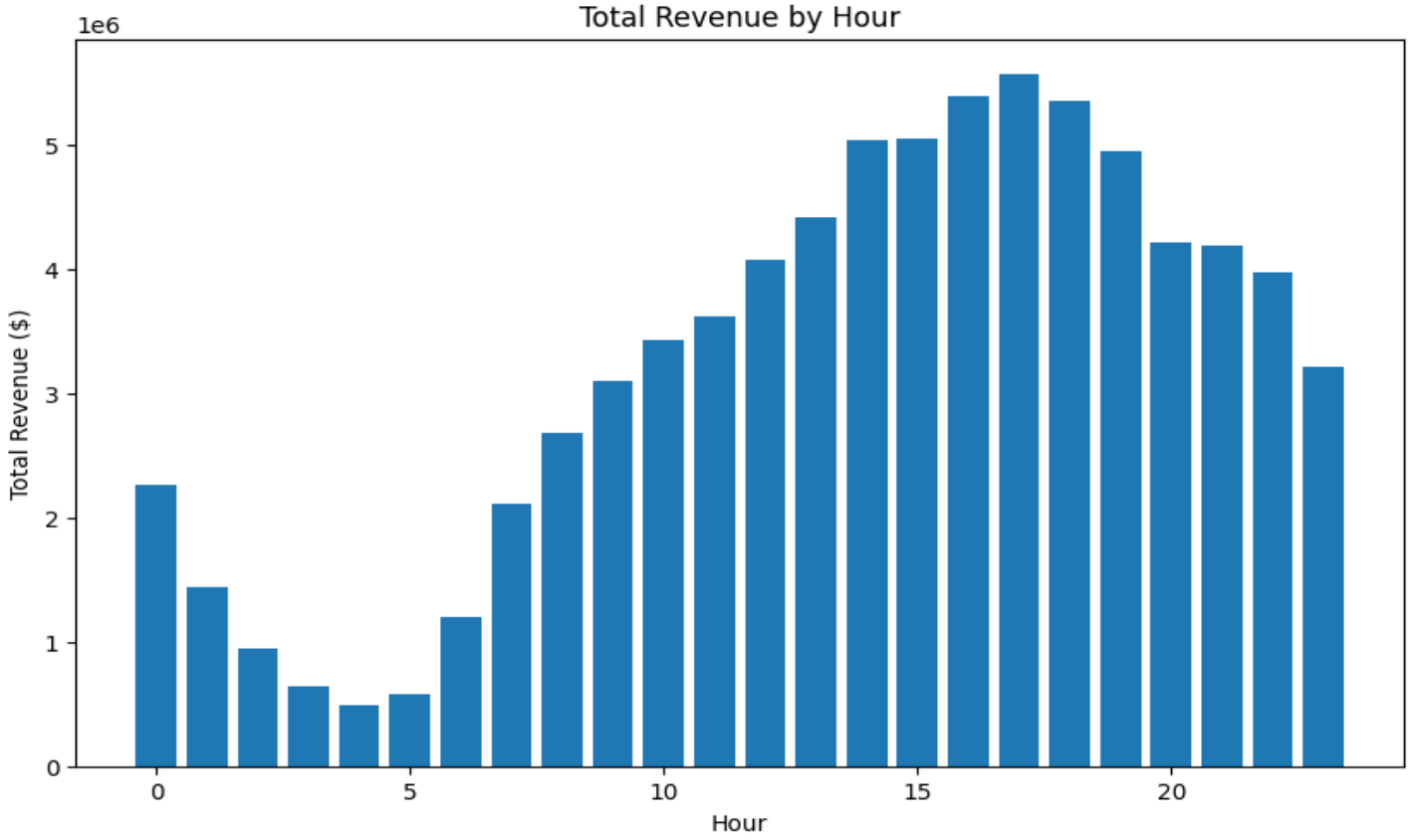
Dashboard Overview: A 360° View of NYC Taxi Operations



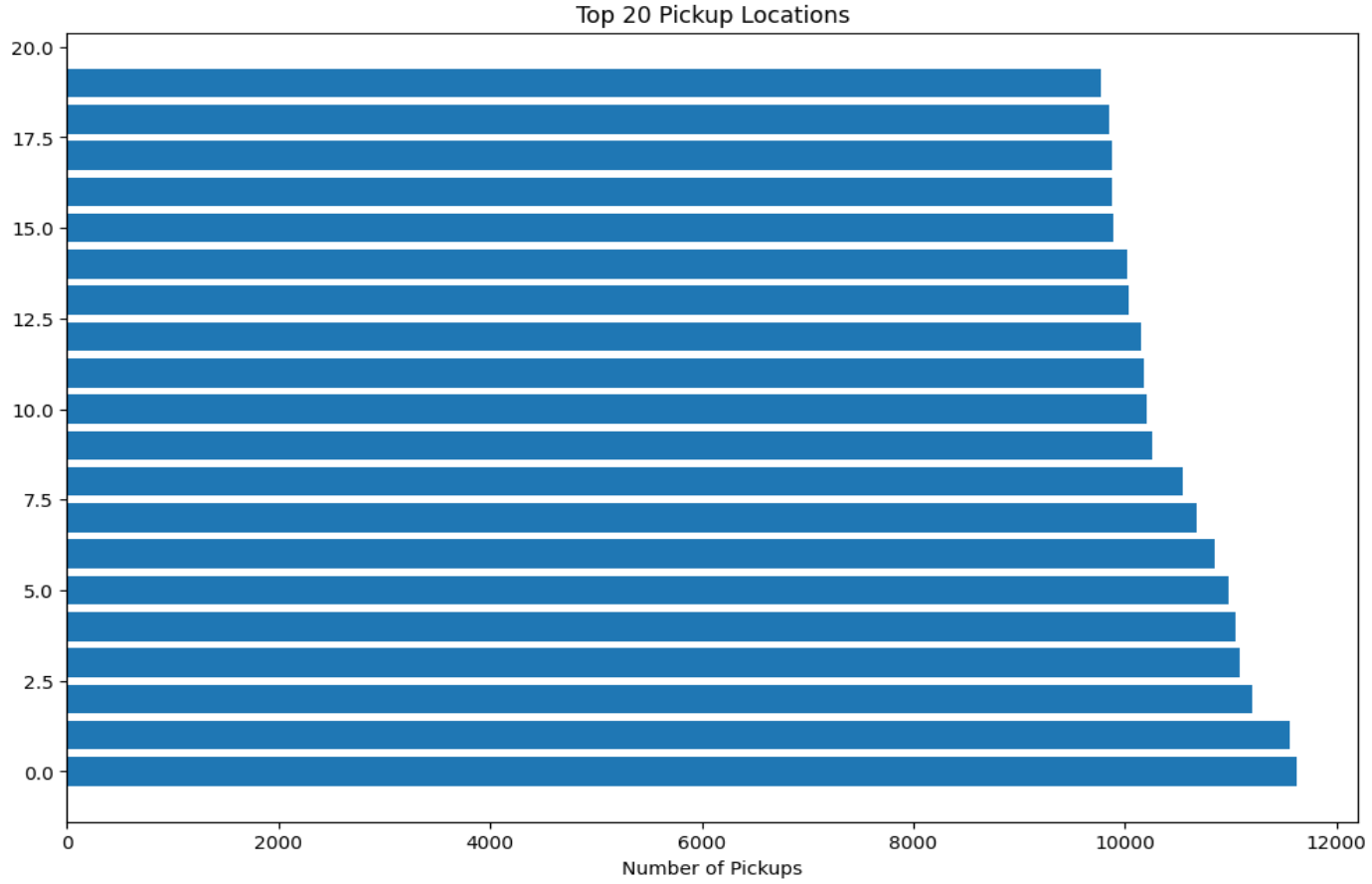
Analytics Deep Dive: Daily Demand Rhythm



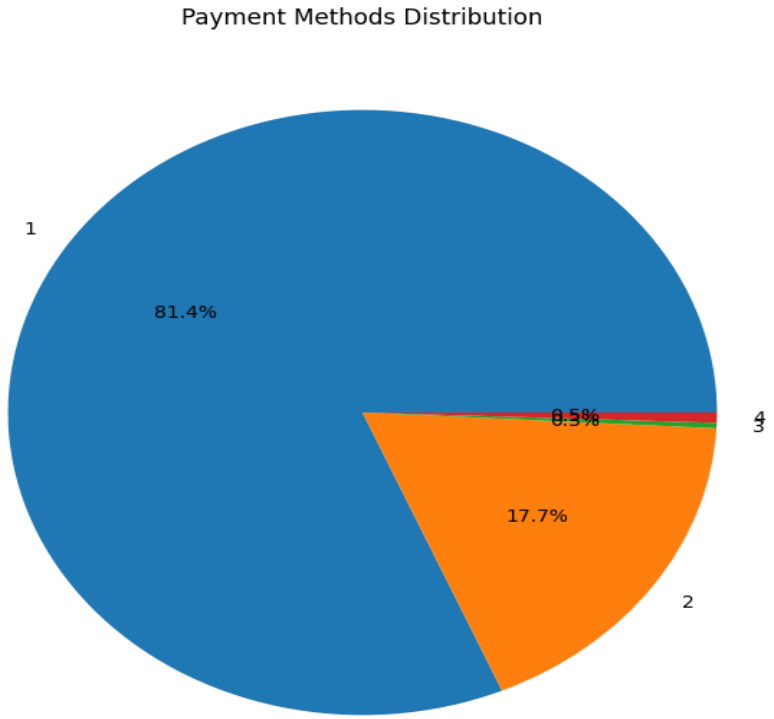
Analytics Deep Dive: Hourly Revenue Analysis



Analytics Deep Dive: Location Hotspots



Analytics Deep Dive: Payment Method Distribution



Summary of Key Insights

- Evening Demand Dominates: Peak hours are between 5-8 PM, driven by post-work activities.
- Cashless is King: Credit cards account for over 81% of payments, highlighting the need for digital payment systems.
- Location is Everything: Airports and commercial hubs in Manhattan are the most critical zones for drivers to maximize pickups.

Conclusion & Future Enhancements

- Conclusion
  - Successfully built an end-to-end, Medallion-based data pipeline.
  - Transformed raw data into reliable, analysis-ready datasets.
  - Delivered actionable business insights through an interactive dashboard.
- Future Enhancements
  - **Advanced Streaming:** Integrate **Apache Kafka** for true real-time processing.
  - **Machine Learning:** Develop a model to predict trip duration or fares.
  - **Cloud Deployment:** Deploy the pipeline on a cloud platform like AWS or Azure.

SAMSUNG