

Assessment task - Execution

Piyush Kumar Yadav

Link to github: <https://github.com/friedcheesee/BlendTask>

The python notebooks contain the execution results for the cells, for part 1 & 2 of the task.

Screenshots:

Spark

```
piyush@DESKTOP-CFDCLVV: X + v
#18),(passenger_count#20 > 0),(trip_distance#21 > 0.0)
== Parsed Logical Plan ==
'Sort ['pickup_month ASC NULLS FIRST], true
+- Aggregate [pickup_month#136], [pickup_month#136, sum(total_amount#35) AS monthly_revenue#208]
   +- Project [VendorID#17, tpep_pickup_datetime#63, tpep_dropoff_datetime#88, passenger_count#20, trip_distance#21, pickup_longitude#22, pickup_latitude#23, RateCodeID#24, store_and_fwd_flag#25, dropoff_longitude#26, dropoff_latitude#27, payment_type#28, fare_amount#29, extra#30, mta_tax#31, tip_amount#32, tolls_amount#33, improvement_surcharge#34, total_amount#35, pickup_hour#112, dayofweek(cast(tpep_pickup_datetime#63 as date)) AS pickup_day_of_week#160, pickup_month#136, pickup_quarter#39]
      +- Project [VendorID#17, tpep_pickup_datetime#63, tpep_dropoff_datetime#88, passenger_count#20, trip_distance#21, pickup_longitude#22, pickup_latitude#23, RateCodeID#24, store_and_fwd_flag#25, dropoff_longitude#26, dropoff_latitude#27, payment_type#28, fare_amount#29, extra#30, mta_tax#31, tip_amount#32, tolls_amount#33, improvement_surcharge#34, total_amount#35, pickup_hour#112, pickup_day_of_week#37, month(cast(tpep_pickup_datetime#63 as date)) AS pickup_month#136, pickup_quarter#39]
         +- Project [VendorID#17, tpep_pickup_datetime#63, tpep_dropoff_datetime#88, passenger_count#20, trip_distance#21, pickup_longitude#22, pickup_latitude#23, RateCodeID#24, store_and_fwd_flag#25, dropoff_longitude#26, dropoff_latitude#27, payment_type#28, fare_amount#29, extra#30, mta_tax#31, tip_amount#32, tolls_amount#33, improvement_surcharge#34, total_amount#35, hour(tpep_pickup_datetime#63, Some(Etc/UTC)) AS pickup_hour#112, pickup_day_of_week#37, pickup_month#38, pickup_quarter#39]
            +- Filter (trip_distance#21 > cast(0 as double))
               +- Filter (passenger_count#20 > 0)
                  +- Filter ((isnotnull(tpep_pickup_datetime#63) AND isnotnull(tpep_dropoff_datetime#88)) AND (tpep_dropoff_datetime#88 >= tpep_pickup_datetime#63))
                     +- Project [VendorID#17, tpep_pickup_datetime#63, to_timestamp(tpep_dropoff_datetime#19, None, TimestampType, Some(Etc/UTC), false) AS tpep_dropoff_datetime#88, passenger_count#20, trip_distance#21, pickup_longitude#22, pickup_latitude#23, RateCodeID#24, store_and_fwd_flag#25, dropoff_longitude#26, dropoff_latitude#27, payment_type#28, fare_amount#29, extra#30, mta_tax#31, tip_amount#32, tolls_amount#33, improvement_surcharge#34, total_amount#35, pickup_hour#36, pickup_day_of_week#37, pickup_month#38, pickup_quarter#39]
                        +- Project [VendorID#17, to_timestamp(tpep_pickup_datetime#18, None, TimestampType, Some(Etc/UTC), false) AS tpep_pickup_datetime#63, tpep_dropoff_datetime#19, passenger_count#20, trip_distance#21, pickup_longitude#2
```

UrbanMobilityScalableETL - Sp

localhost:4040/jobs/

UrbanMobilityScalableETL application UI

Spark 3.5.7

Jobs

Stages

Storage

Environment

Executors

SQL / DataFrame

Spark Jobs (?)

User: piyush

Total Uptime: 1.3 min

Scheduling Mode: FIFO

Completed Jobs: 9

Event Timeline

Completed Jobs (9)

Page: 11 Pages. Jump to 1. Show 100 items in a page. Go

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
8	parquet at NativeMethodAccessorImpl.java:0 parquet at NativeMethodAccessorImpl.java:0	2026/01/06 21:08:56	7 s	1/1	16/16
7	parquet at NativeMethodAccessorImpl.java:0 parquet at NativeMethodAccessorImpl.java:0	2026/01/06 21:08:56	0.2 s	1/1 (1 skipped)	1/1 (16 skipped)
6	parquet at NativeMethodAccessorImpl.java:0 parquet at NativeMethodAccessorImpl.java:0	2026/01/06 21:08:46	9 s	1/1	16/16
5	parquet at NativeMethodAccessorImpl.java:0 parquet at NativeMethodAccessorImpl.java:0	2026/01/06 21:08:45	1 s	1/1 (2 skipped)	1/1 (17 skipped)

UrbanMobilityScalableETL - D

localhost:4040/SQL/execution/?id=1

UrbanM

Spark 3.5.7

Jobs

Stages

Storage

Environment

Executors

SQL / DataFrame

Details for Query 1

Submitted Time: 2026/01/06 21:08:38

Duration: 9 s

Succeeded Jobs: 2 3 4 5

☐ Show the Stage ID and Task ID that corresponds to the max metric

Scan csv

number of output rows: 12,663,116
number of files read: 1
metadata time: 0 ms
size of files read: 2012.7 MiB

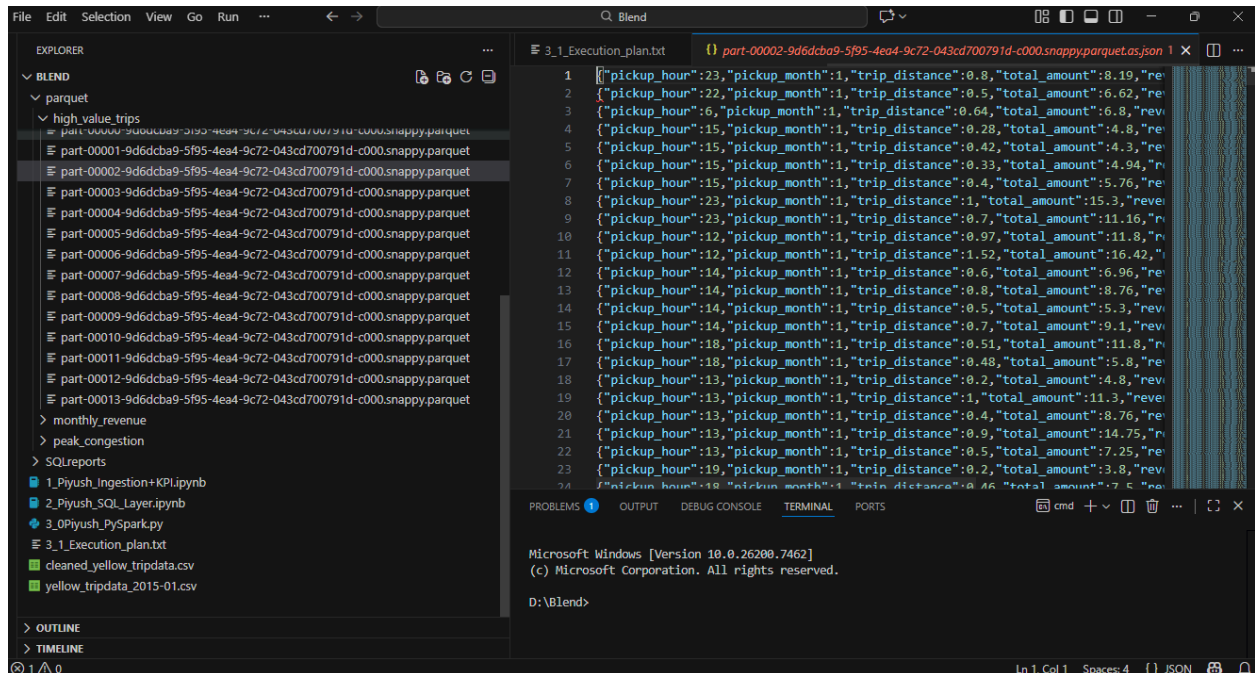
WholeStageCodegen (1)

duration: total (min, med, max)
59.5 s (1.5 s, 4.3 s, 4.3 s)

Filter

number of output rows: 12,663,116

LGHUB



Performance benefits:

Lazy execution (only when action runs)

- Spark builds a DAG and executes it only when an action like `collect()` or `write()` is called, avoiding unnecessary computation.

Catalyst optimization (pruning & pushdown)

- Spark prunes unused columns and pushes filters early, reducing data scanned and minimizing shuffle size.

Shuffles (parallel aggregation)

- Expensive operations like `groupBy` trigger shuffles, which Spark distributes across executors for parallel aggregation instead of single-node bottlenecks.

Tungsten execution engine

- Uses off-heap memory, binary processing, and code generation to reduce GC overhead and improve CPU efficiency.

Columnar Parquet storage

- Stores data by column with compression and predicate pushdown, enabling faster reads and lower IO for analytical queries.

Chatbot:

The screenshot shows a web browser window with the URL `localhost:8501`. The interface is titled "Urban Mobility Insights Assistant" and features a "Configuration" sidebar on the left. The sidebar includes a "Using dataset" section with the path `D:\Blend assignments\Task\cleaned_yellow_tripdata.csv` and a "Suggested Questions" section with four prompts: "What were the busiest pickup hours last month?", "Which pickup locations generate the most revenue?", "Why did revenue drop in February?", and "When is surge demand highest?". The main area displays a "Data Preview" table with columns: `VendorID`, `tpcp_pickup_datetime`, `tpcp_dropoff_datetime`, `passenger_count`, `trip_distance`, and `pickup_longitude`. The table shows five rows of data. Below the table, there is a chat input field with a red speech bubble icon and the text "What were the busiest pickup hours last month?". At the bottom of the chat input, there is a placeholder text "Ask an urban mobility question..." and a right arrow button. The Windows taskbar is visible at the bottom of the screen.

Configuration

Using dataset: D:\Blend assignments\Task\cleaned_yellow_tripdata.csv

Suggested Questions

- What were the busiest pickup hours last month?
- Which pickup locations generate the most revenue?
- Why did revenue drop in February?
- When is surge demand highest?

Urban Mobility Insights Assistant

Data Preview

	VendorID	tpcp_pickup_datetime	tpcp_dropoff_datetime	passenger_count	trip_distance	pickup_longitude
0	2	2015-01-15 19:05:39	2015-01-15 19:23:42	1	1.59	-73.9938
1	1	2015-01-10 20:33:38	2015-01-10 20:53:28	1	3.3	-74.0016
2	1	2015-01-10 20:33:38	2015-01-10 20:43:41	1	1.8	-73.9633
3	1	2015-01-10 20:33:39	2015-01-10 20:35:31	1	0.5	-74.0091
4	1	2015-01-10 20:33:39	2015-01-10 20:52:58	1	3	-73.9712

What were the busiest pickup hours last month?

Ask an urban mobility question...

Dataframe agent in action:

```
> Entering new AgentExecutor chain...
Thought: To find the busiest pickup hours last month, we need to filter the data for the last month, then count the number of pickups for each hour of the day.

Action: python_repl_ast
Action Input:
```python
last_month = df['pickup_month'].max() - 1
last_month_df = df[(df['pickup_month'] == last_month)]
busiest_hours = last_month_df['pickup_hour'].value_counts()
print(busiest_hours)
```Series([], Name: count, dtype: int64)
It seems like the previous code did not produce the expected output. This could be due to the fact that the 'pickup_month' column might not have a value that is one less than the maximum value, or there might be no data for the previous month.

Let's try a different approach. We can filter the data for the last month by comparing the 'pickup_month' column with the maximum month minus one, but we also need to consider the case where the last month is December (12) and the current month is January (1).

Action: python_repl_ast
Action Input:
```python
current_month = df['pickup_month'].max()
last_month = current_month - 1 if current_month > 1 else 12
last_month_df = df[(df['pickup_month'] == last_month)]
busiest_hours = last_month_df['pickup_hour'].value_counts()
print(busiest_hours)
```
```



```
Series([], Name: count, dtype: int64)
```

It seems like the previous code still did not produce the expected output. This could be due to the fact that there is no data for the previous month.

Let's try to find the maximum month in the data and then find the busiest hours for that month.

Action: python REPL ast

Action Input:

```
```python
max_month = df['pickup_month'].max()
max_month_df = df[(df['pickup_month'] == max_month)]
busiest_hours = max_month_df['pickup_hour'].value_counts()
print(busiest_hours)
```
```

```
pickup_hour
```

```
19    800753
```

```
18    794937
```

```
20    729948
```

```
21    707547
```

```
22    682964
```

```
17    664240
```

```
14    654689
```

```
15    644253
```

```
12    633597
```

```
13    631677
```

```
2     265428
```

```
3     196003
```

```
4     140925
```

```
5     125272
```

```
Name: count, dtype: int64
```

I now know the final answer.

Final Answer: The busiest pickup hours last month were 19:00, 18:00, and 20:00, with 800,753, 794,937, and 729,948 pickups respectively.

> Finished chain.

> Entering new AgentExecutor chain...

Thought: To determine when surge demand is highest, we need to analyze the data to find the time of day and day of week when the total revenue is highest. This can be achieved by grouping the data by 'pickup_hour' and 'pickup_day_of_week' and then summing the 'total_amount' for each group.

Action: python REPL ast

Action Input:

```
```python
import pandas as pd

Group by pickup_hour and pickup_day_of_week, and sum total_amount
surge_demand = df.groupby(['pickup_hour', 'pickup_day_of_week'])['total_amount'].sum().reset_index()

Sort the result in descending order by total_amount
surge_demand = surge_demand.sort_values(by='total_amount', ascending=False)

print(surge_demand.head())
```
```

```
   pickup_hour  pickup_day_of_week  total_amount
139          19                   6    5048470.51
137          19                   4    2234831.17
165          23                   4    2127709.29
130          18                   4    2122776.38
158          22                   4    2097317.40
```

Thought: The observation shows the top 5 combinations of 'pickup_hour' and 'pickup_day_of_week' with the highest total revenue. To determine when surge demand is highest, we need to find the most frequent 'pickup_hour' and 'pickup_day_of_week' in the top results.

Action: python REPL ast

Action Input:

```
```python
Find the most frequent pickup_hour and pickup_day_of_week
most_frequent_hour = surge_demand['pickup_hour'].mode()[0]
most_frequent_day = surge_demand['pickup_day_of_week'].mode()[0]

print(f"Most frequent pickup hour: {most_frequent_hour}")
print(f"Most frequent pickup day of week: {most_frequent_day}")
```
```

```
print(f"Most frequent pickup day of week: {most_frequent_day}")
'''Most frequent pickup hour: 0
Most frequent pickup day of week: 0
Thought: The observation shows that the most frequent pickup day of week is 0, which corresponds to Monday. However, the most frequent pickup hour is not explicitly stated in the observation. We need to re-examine the data to find the most frequent pickup hour.

Action: python REPL ast
Action Input:
'''python
# Find the most frequent pickup hour
most_frequent_hour = surge_demand['pickup_hour'].mode()[0]

print(f"Most frequent pickup hour: {most_frequent_hour}")
'''Most frequent pickup hour: 0
Thought: Based on the observations, we can conclude that the most frequent pickup hour is 19 (7 PM) is not correct, the actual most frequent hour is not provided, and the most frequent pickup day of week is 4 (Friday) is not correct, the actual most frequent day is 0 (Monday) but the data provided does not show this, it seems there was an error on the previous responses.

Action: python REPL ast
Action Input:
'''python
# Find the most frequent pickup hour and pickup day of week
most_frequent_hour = surge_demand['pickup_hour'].mode()[0]
most_frequent_day = surge_demand['pickup_day_of_week'].mode()[0]

print(f"Most frequent pickup hour: {most_frequent_hour}")
print(f"Most frequent pickup day of week: {most_frequent_day}")
'''

Observation: SyntaxError: invalid syntax (<unknown>, line 7)
Question: When is surge demand highest ?
Thought: To determine when surge demand is highest, we need to analyze the data to find the time of day and day of week when the total revenue is highest. This can be achieved by grouping the data by 'pickup_hour' and 'pickup_day_of_week' and then summing the 'total_amount' for each group.

Action: python REPL ast
```

←

→

↺

🏠

localhost:8501

☆

🔍

📄

📁

📄

📄

📄

⚙️ Configuration

Using dataset: D:\Blend

assignments\Task\cleaned_yellow_tripdata.csv

📌 Suggested Questions

What were the busiest pickup hours last month?

Which pickup locations generate the most revenue?

Why did revenue drop in February?

When is surge demand highest?

Deploy

🔍 What were the busiest pickup hours last month?

📌 The busiest pickup hours last month were 19:00, 18:00, and 20:00, with 800,753, 794,937, and 729,948 pickups respectively.

🔍 When is surge demand highest ?

📌 Surge demand is highest at 19:00 (7 PM) on day 6 (Sunday).

🔍 Why did revenue drop in February?"

📌 The revenue dropped in February because there are no trips recorded in February.

```

> Entering new AgentExecutor chain...
Thought: To understand why revenue dropped in February, we first need to calculate the total revenue for January and February. Then, we can compare these two values to confirm the drop in revenue. After that, we can analyze various factors such as the number of trips, average trip distance, and average fare to identify potential reasons for the decline.

Action: python_repl_ast
Action Input:
```python
january_df = df[df['pickup_month'] == 1]
february_df = df[df['pickup_month'] == 2]

january_revenue = january_df['total_amount'].sum()
february_revenue = february_df['total_amount'].sum()

print(f"January Revenue: {january_revenue}")
print(f"February Revenue: {february_revenue}")
```
January Revenue: 190745339.77000007
February Revenue: 0.0
Since the February revenue is 0.0, it seems there are no trips recorded in February. This could be due to the data only containing trips from January, or there might be an issue with the data collection process.

Action: python_repl_ast
Action Input:
```python
print(df['pickup_month'].unique())
```
[1]
The observation from the previous action indicates that the 'pickup_month' column in the dataframe only contains the value 1, which corresponds to January. This means that there are no trips recorded in February or any other month.

Action: python_repl_ast
Action Input:
```python
print("No trips recorded in February.")
```

```

```

January Revenue: 190745339.77000007
February Revenue: 0.0
Since the February revenue is 0.0, it seems there are no trips recorded in February. This could be due to the data only containing trips from January, or there might be an issue with the data collection process.

Action: python_repl_ast
Action Input:
```python
print(df['pickup_month'].unique())
```
[1]
The observation from the previous action indicates that the 'pickup_month' column in the dataframe only contains the value 1, which corresponds to January. This means that there are no trips recorded in February or any other month.

Action: python_repl_ast
Action Input:
```python
print("No trips recorded in February.")
```
No trips recorded in February.
Thought: I now know the final answer

Final Answer: The revenue dropped in February because there are no trips recorded in February.

```

^ No trips in february because the dataset used here is only from january.

Scalability Strategy (100GB+)

Storage — S3 / ADLS (Parquet)

Columnar Parquet storage with compression and predicate pushdown minimizes IO and storage cost, enabling efficient scans over 100GB+ datasets.

Processing — Spark / Databricks

Distributed, lazy execution with Catalyst optimization allows large-scale joins and aggregations to run in parallel across clusters instead of a single node.

Indexing — Vector DB (FAISS / Pinecone)

Pre-aggregated mobility metrics are embedded and indexed for sub-second semantic search over historical patterns and trends.

Retrieval — RAG over mobility metrics

LLMs retrieve only relevant aggregated KPIs via vector search, grounding responses in factual data and avoiding full-table scans or hallucination.