INTRODUCTION:

This project's main goal is to model and analyse taxi trip data from New York City, with an emphasis on both yellow and green cab services. From 2015 to 2022, the dataset includes detailed information about taxi trips, including fare amounts, trip distances, pickup and drop-off locations, and other important details.
Understanding the trends and insights from big datasets, like taxi trip data, can yield important business insights in the era of data-driven decision-making. Taxi drivers and city planners alike stand to gain from our ability to spot patterns, forecast consumer behaviour, and streamline operations through trip analysis. Even when working with big datasets, this project effectively delivers these insights by utilising Spark SQL, PySpark, and Machine Learning (ML) models.

PROJECT OVERVIEW:

The project is divided into three major sections:

Part A: Data Ingestion and Preparation:

This phase focuses on **data ingestion**, **cleaning**, and **preparation** for further analysis:

Data Ingestion:

These datasets contain critical taxi trip details that will be analysed throughout the project. Once the datasets are in Azure, we mount them to Databricks using Azure Blob Storage keys, allowing Databricks to read and process the files efficiently. After mounting, the data is stored in Databricks File System (DBFS), enabling easier management and faster querying throughout the analysis.
Yellow and green taxi datasets from 2015 to 2022 are loaded from Azure Blob Storage into Databricks File System (DBFS).

Table

	path	name	size	modificationTime
1	dbfs:/mnt/blobstorage/combined_taxi_data.parquet/	combined_taxi_data.parquet/	0	1728846996000
2	dbfs:/mnt/blobstorage/green_taxi_2015.csv	green_taxi_2015.csv	2179116866	1728809980000
3	dbfs:/mnt/blobstorage/green_taxi_2015.parquet	green_taxi_2015.parquet	404556105	1728010224000
4	dbfs:/mnt/blobstorage/green_taxi_2016.parquet	green_taxi_2016.parquet	346731598	1728010064000
5	dbfs:/mnt/blobstorage/green_taxi_2017.parquet	green_taxi_2017.parquet	251504885	1728009741000
6	dbfs:/mnt/blobstorage/green_taxi_2018.parquet	green_taxi_2018.parquet	193874396	1728009657000
7	dbfs:/mnt/blobstorage/green_taxi_2019.parquet	green_taxi_2019.parquet	142163406	1728009532000
8	dbfs:/mnt/blobstorage/green_taxi_2020.parquet	green_taxi_2020.parquet	37076741	1728009689000
9	dbfs:/mnt/blobstorage/green_taxi_2021.parquet	green_taxi_2021.parquet	23479865	1728009776000
10	dbfs:/mnt/blobstorage/green_taxi_2022.parquet	green_taxi_2022.parquet	20085692	1728009790000
11	dbfs:/mnt/blobstorage/location_data.parquet/	location_data.parquet/	0	1728834473000
12	dbfs:/mnt/blobstorage/yellow_taxi_2015.parquet	yellow_taxi_2015.parquet	2896588873	1728023518000
13	dbfs:/mnt/blobstorage/yellow_taxi_2016.parquet	yellow_taxi_2016.parquet	2616839324	1728026712000
14	dbfs:/mnt/blobstorage/yellow_taxi_2017.parquet	yellow_taxi_2017.parquet	2072273391	1728043331000
15	dbfs:/mnt/blobstorage/yellow_taxi_2018.parquet	yellow_taxi_2018.parquet	2098068908	1728043345000

19 rows | 8.39 seconds runtime

Refreshed 13 hours ago

To guarantee data completeness and integrity, row counts are computed for both taxi services.

07:08 PM (18s) 10 Python

```
# Load and count rows for yellow taxi data (2015-2022)
yellow_taxi_files = [
    "dbfs:/mnt/dbfs-storage/yellow_taxi_2015.parquet",
    "dbfs:/mnt/dbfs-storage/yellow_taxi_2016.parquet",
    "dbfs:/mnt/dbfs-storage/yellow_taxi_2017.parquet",
    "dbfs:/mnt/dbfs-storage/yellow_taxi_2018.parquet",
    "dbfs:/mnt/dbfs-storage/yellow_taxi_2019.parquet",
    "dbfs:/mnt/dbfs-storage/yellow_taxi_2020.parquet",
    "dbfs:/mnt/dbfs-storage/yellow_taxi_2021.parquet",
    "dbfs:/mnt/dbfs-storage/yellow_taxi_2022.parquet"
]

total_yellow_rows = 0

for file in yellow_taxi_files:
    df = spark.read.parquet(file)
    total_yellow_rows += df.count()

print(f"Total number of rows for Yellow Taxi data: {total_yellow_rows}")
```

```
(24) Spark Jobs
df:  pyspark.sql.dataframe.DataFrame = [VendorID: long, tpep_pickup_datetime: timestamp ... 17 more fields]
Total number of rows for Yellow Taxi data: 663055251
```

```
07:08 PM (10s)                                                    9

# Load and count rows for green taxi data (2015-2022)
green_taxi_files = [
    "dbfs:/mnt/dbfs-storage/green_taxi_2015.parquet",
    "dbfs:/mnt/dbfs-storage/green_taxi_2016.parquet",
    "dbfs:/mnt/dbfs-storage/green_taxi_2017.parquet",
    "dbfs:/mnt/dbfs-storage/green_taxi_2018.parquet",
    "dbfs:/mnt/dbfs-storage/green_taxi_2019.parquet",
    "dbfs:/mnt/dbfs-storage/green_taxi_2020.parquet",
    "dbfs:/mnt/dbfs-storage/green_taxi_2021.parquet",
    "dbfs:/mnt/dbfs-storage/green_taxi_2022.parquet"
]

total_green_rows = 0

for file in green_taxi_files:
    df = spark.read.parquet(file)
    total_green_rows += df.count()

print(f"Total number of rows for Green Taxi data: {total_green_rows}")
```

```
df:  pyspark.sql.dataframe.DataFrame = [VendorID: long, lpep_pickup_datetime: timestamp ... 18 more fields]
Total number of rows for Green Taxi data: 66200401
```

Location Referential Data and Parquet vs CSV Conversion:
we downloaded and loaded the location referential CSV file, which maps the taxi trip location IDs to actual geographical locations like boroughs and zones. This reference data is crucial for interpreting pickup and dropoff locations in the analysis.

```
2 days ago (3s)                                    25                    Python

# Load the CSV file from DBFS into a Spark DataFrame
location_df = spark.read.csv("/FileStore/taxi_zone_lookup.csv", header=True, inferSchema=True)

# Show the first few rows of the location data
location_df.show(5)

(3) Spark Jobs
location_df:  pyspark.sql.dataframe.DataFrame = [LocationID: integer, Borough: string ... 2 more fields]
+----------+-------------+--------------------+------------+
|LocationID|      Borough|                Zone|service_zone|
+----------+-------------+--------------------+------------+
|         1|          EWR|      Newark Airport|         EWR|
|         2|       Queens|         Jamaica Bay|   Boro Zone|
|         3|        Bronx|Allerton/Pelham G...|   Boro Zone|
|         4|    Manhattan|       Alphabet City| Yellow Zone|
|         5|Staten Island|       Arden Heights|   Boro Zone|
+----------+-------------+--------------------+------------+
only showing top 5 rows
```

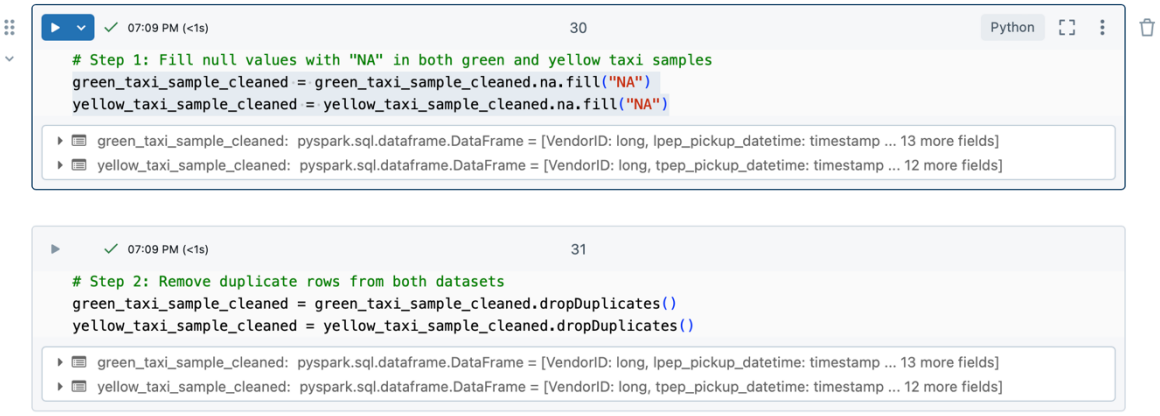
we converted the "Green Taxi 2015" dataset from Parquet format to CSV and compared the file sizes. Parquet, being a columnar storage format, is more optimized for analytical workloads, offering better compression and faster query performance than CSV. While CSV files are more portable and simpler, Parquet's efficiency makes it better suited for large-scale data processing, especially with structured datasets like taxi trips.

Parquet is a columnar storage format that allows for much more efficient compression than row-based formats like CSV. Since data is stored column-by-column, Parquet can apply more effective compression algorithms, resulting in smaller file sizes.

Since Parquet is columnar, it's optimized for reading specific columns without having to read the entire dataset. For example, if you need only a few columns for analysis, Parquet allows you to read just those columns, reducing the amount of I/O and speeding up processing.

DATA CLEANING:

Data cleaning is the process of identifying and correcting errors, inconsistencies, or missing values in a dataset to improve data quality. This step ensures that the dataset is accurate, complete, and ready for analysis. Common tasks in data cleaning include handling missing data, removing duplicates, and correcting data entry errors.



1. Trips Finishing Before the Starting Time: The first step is to remove any trips where the dropoff time is earlier than the pickup time. These entries are

clearly erroneous, likely due to system recording errors.

```
08:14 PM (<1s)                                                    40
from pyspark.sql.functions import col, unix_timestamp

# Define the valid datetime range
start_date = "2015-01-01"
end_date = "2022-12-31"
```

```
07:10 PM (<1s)                                                    41                    Python
# Step 1: Filter Green Taxi Sample
```

```
07:10 PM (<1s)                                                    42
# 1. Filter out trips where dropoff is before or equal to pickup time
green_taxi_cleaned = green_taxi_sample_reduced.filter(
    green_taxi_sample_reduced.lpep_dropoff_datetime > green_taxi_sample_reduced.lpep_pickup_datetime
)
green_taxi_cleaned:  pyspark.sql.dataframe.DataFrame = [VendorID: long, lpep_pickup_datetime: timestamp ... 13 more fields]
```

```
07:11 PM (<1s)                                                    59
# 1. Filter out trips where dropoff is before or equal to pickup time
yellow_taxi_cleaned = yellow_taxi_sample_reduced.filter(
    yellow_taxi_sample_reduced.tpep_dropoff_datetime > yellow_taxi_sample_reduced.tpep_pickup_datetime
)
yellow_taxi_cleaned:  pyspark.sql.dataframe.DataFrame = [VendorID: long, tpep_pickup_datetime: timestamp ... 12 more fields]
```

2. Trips Where Pickup/Dropoff Datetime is Outside the Valid Range: This step focuses on filtering out trips that have pickup or dropoff times outside a predefined valid range. For example, times outside the recorded period of interest (2015-2022) should be excluded.

```
07:10 PM (<1s)                                                    44                    Python
# 2. Filter trips with pickup/dropoff datetime outside the valid date range
green_taxi_cleaned = green_taxi_cleaned.filter(
    (col("lpep_pickup_datetime") >= start_date) &
    (col("lpep_dropoff_datetime") <= end_date)
)
green_taxi_cleaned:  pyspark.sql.dataframe.DataFrame = [VendorID: long, lpep_pickup_datetime: timestamp ... 13 more fields]
```

```
07:11 PM (<1s)                                                    60                    Python
# 2. Filter trips with pickup/dropoff datetime outside the valid date range
yellow_taxi_cleaned = yellow_taxi_cleaned.filter(
    (col("tpep_pickup_datetime") >= start_date) &
    (col("tpep_dropoff_datetime") <= end_date)
)
```

```
yellow_taxi_cleaned:  pyspark.sql.dataframe.DataFrame
    VendorID: long
    tpep_pickup_datetime: timestamp
    tpep_dropoff_datetime: timestamp
    passenger_count: double
    trip_distance: double
    PULocationID: long
    DOLocationID: long
    payment_type: long
    fare_amount: double
    extra: double
    tip_amount: double
    tolls_amount: double
    improvement_surcharge: double
    total_amount: double
```

3. Trips with Negative Speed: Trips where the calculated speed (distance/duration) is negative are invalid. These trips are removed, as negative speed is physically impossible and indicates a data entry error.

```
07:11 PM (<1s)                                                    61
# 3. Calculate trip duration in hours
yellow_taxi_cleaned = yellow_taxi_cleaned.withColumn("trip_duration_hours",
    (unix_timestamp("tpep_dropoff_datetime") - unix_timestamp("tpep_pickup_datetime")) / 3600
)
```

```
yellow_taxi_cleaned:  pyspark.sql.dataframe.DataFrame = [VendorID: long, tpep_pickup_datetime: timestamp ... 13 more fields]
```

```
07:11 PM (<1s)                                                    62                    Python
# 4. Calculate speed (miles per hour)
yellow_taxi_cleaned = yellow_taxi_cleaned.withColumn("speed_mph",
    col("trip_distance") / col("trip_duration_hours")
)
```

```
yellow_taxi_cleaned:  pyspark.sql.dataframe.DataFrame = [VendorID: long, tpep_pickup_datetime: timestamp ... 14 more fields]
```

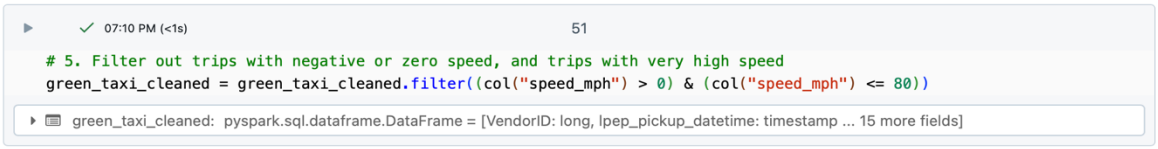
4. Trips with Very High Speed: Filtering out trips with excessively high speeds helps maintain the integrity of the dataset. For instance, NYC's maximum speed limit is around 55 mph, so trips exceeding this limit are flagged as unrealistic.

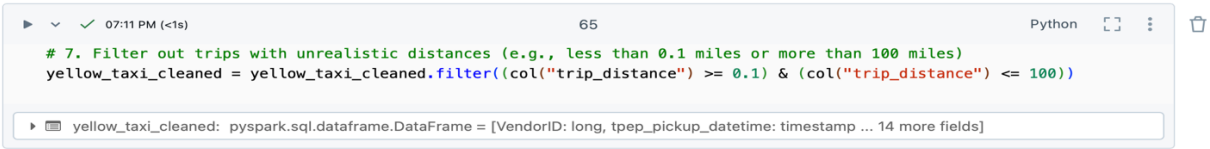
```
07:10 PM (<1s)                                                    50                    Python
# 4. Calculate speed (miles per hour)
# Filter out rows where trip_duration_hours is zero or negative to avoid division by zero
green_taxi_cleaned = green_taxi_cleaned.filter(col("trip_duration_hours") > 0)

# Recalculate speed in miles per hour
green_taxi_cleaned = green_taxi_cleaned.withColumn("speed_mph",
    col("trip_distance") / col("trip_duration_hours")
)
```

```
green_taxi_cleaned:  pyspark.sql.dataframe.DataFrame = [VendorID: long, lpep_pickup_datetime: timestamp ... 15 more fields]
```

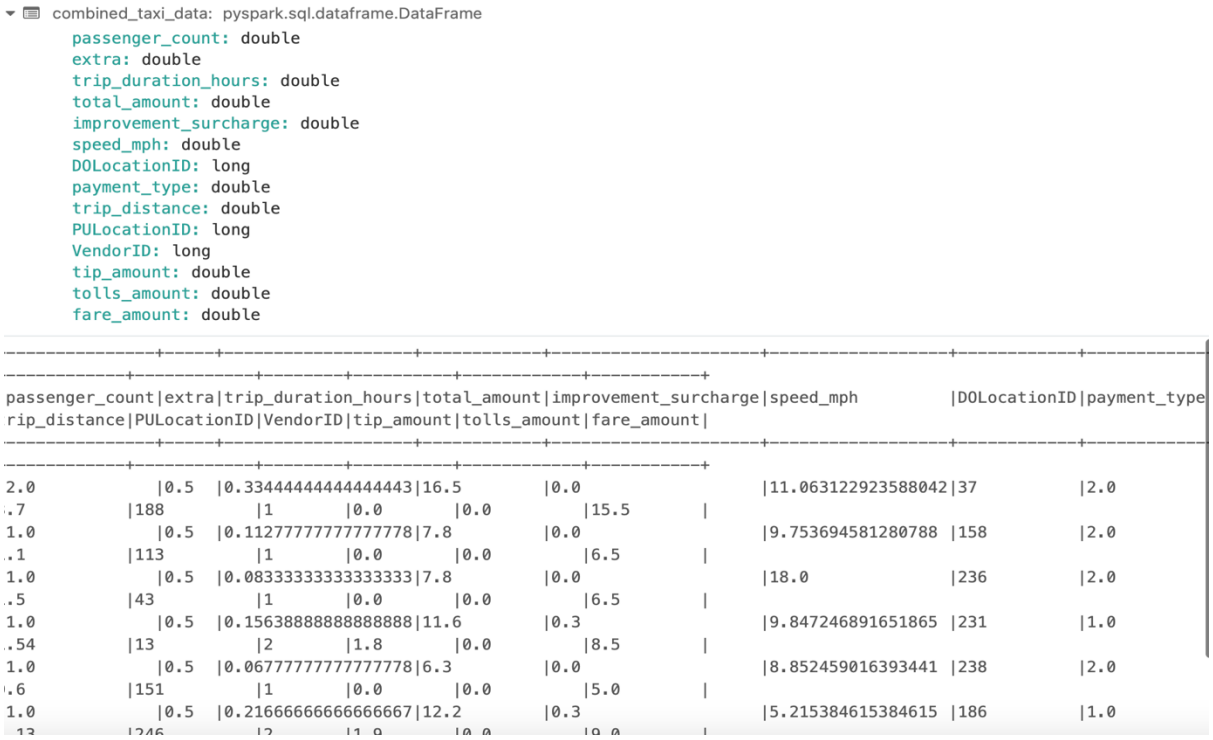
5. Trips That Are Too Short or Too Long (Duration-Wise): Trips with very short (e.g., less than 1 minute) or very long durations (e.g., more than 4 hours) are considered outliers and should be removed to avoid skewing the analysis.

6. Trips That Are Too Short or Too Long (Distance-Wise): Similarly, trips with extremely short or long distances (e.g., less than 0.1 miles or more than 100 miles) should be excluded, as they indicate recording errors or unrealistic travel patterns.





Combining Yellow and Green Taxi Datasets:
The datasets for Yellow and Green taxis are gathered from various services, and although they have comparable characteristics (such as total money, trip duration, etc.), their schemas varies slightly



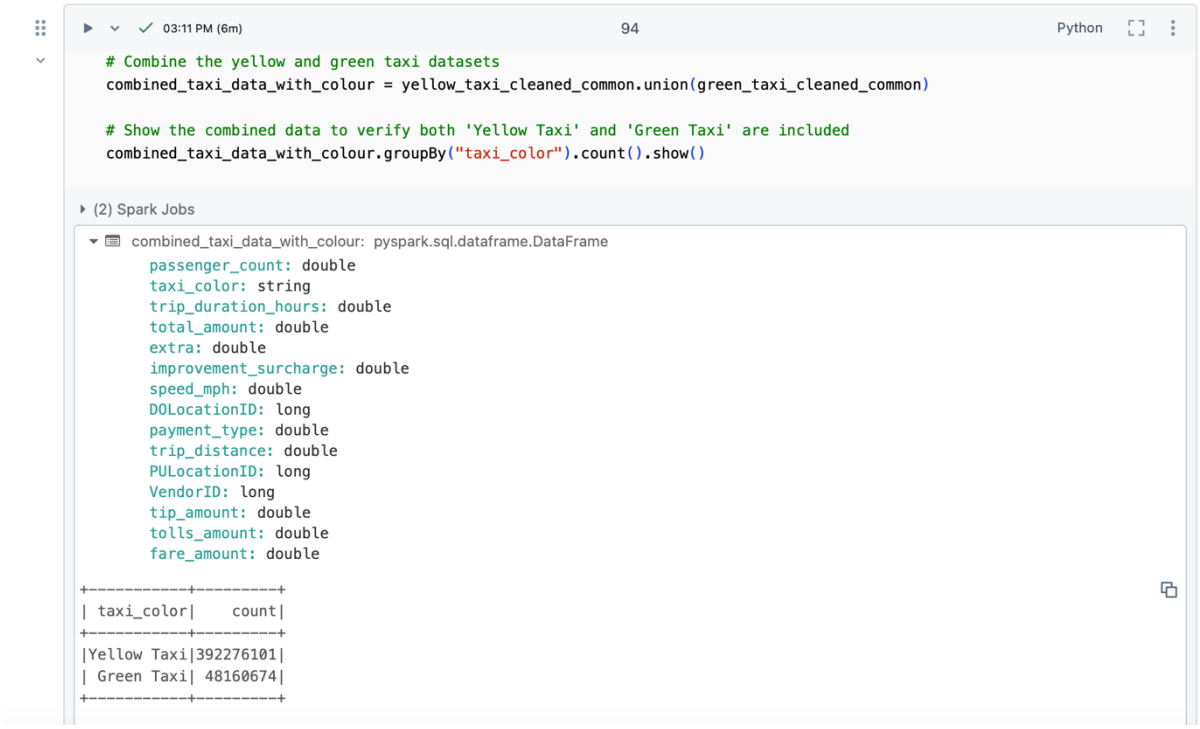
Schema Alignment: First, both datasets are aligned by ensuring they have the same columns. Any missing columns in one dataset are filled with null values to ensure consistency. By using union operations, we can concatenate the datasets vertically, treating them as one unified dataset of taxi trips.

Loading Location Data:

The union function is used to vertically join the rows from the yellow and green cab datasets into a single, cohesive DataFrame. Simplified analysis across both taxi services is made possible by combining the cleaned yellow and green taxi data into a single DataFrame. All trips regardless of taxi color are kept together for subsequent processing and querying in the analysis pipeline thanks to this consolidation. Future activities, such as joining with location data or running statistical analyses on the complete dataset, are made simpler by merging the

databases.

```
+---------------+-----+-------------------+------------+--------------------+------------------+------------+------------
------------+------------+--------+----------+------------+-----------+
|passenger_count|extra|trip_duration_hours|total_amount|improvement_surcharge|speed_mph         |DOLocationID|payment_type
trip_distance|PULocationID|VendorID|tip_amount|tolls_amount|fare_amount|
+---------------+-----+-------------------+------------+--------------------+------------------+------------+------------
------------+------------+--------+----------+------------+-----------+
|1.0            |0.0  |0.6577777777777778 |-101.8      |-0.3                |31.880067567567565|1           |2.0
20.97         |186          |2       |0.0       |-23.5       |-78.0      |
|1.0            |-1.0 |0.8988888888888888 |-124.55     |-0.3                |23.784919653893695|1           |2.0
21.38         |163          |2       |0.0       |-18.25      |-102.0     |
|2.0            |0.0  |0.5555555555555556 |99.05       |0.0                 |35.81999999999999 |1           |1.0
19.9          |158          |1       |10.0      |16.25       |72.5       |
|3.0            |0.0  |0.3913888888888889 |97.25       |0.0                 |36.53655074520937 |1           |1.0
14.3          |211          |1       |16.2      |23.75       |57.0       |
|1.0            |0.0  |0.4847222222222222 |76.55       |0.0                 |34.04011461318051 |1           |2.0
16.5          |186          |1       |0.0       |11.75       |64.5       |
|2.0            |0.0  |0.4955555555555556 |100.2       |0.0                 |41.165919282511204|1           |1.0
20.4          |161          |1       |16.65     |11.75       |71.5       |
|3.0            |0.0  |0.35444444444444445|86.1        |0.3                 |41.33228840125392 |1           |1.0
14.65         |114          |2       |14.3      |14.0        |57.5       |
|1.0            |1.0  |0.6013888888888889 |98.4        |0.3                 |29.23233256351039 |1           |1.0
```



Exporting Combined Data into Parquet Format in DBFS:

The next natural step after merging the yellow and green cab datasets is to save the combined data in a Parquet file inside the Databricks File System (DBFS). Because of its exceptional efficiency in terms of both query performance and storage, the Parquet format was chosen. When working with massive datasets, this columnar storage format is much faster and more storage-efficient than conventional formats like CSV since it improves compression and optimises

analytical queries. Data processing is made more efficient by using Parquet, which eventually enhances the effectiveness of any further analysis.

```
# Now save the combined dataset to Parquet
combined_taxi_cleaned.write.mode("overwrite").parquet("dbfs:/FileStore/combined_taxi_data.parquet")
```

▸ (2) Spark Jobs

▾ combined_taxi_cleaned: pyspark.sql.dataframe.DataFrame

```
passenger_count: double
extra: double
trip_duration_hours: double
total_amount: double
improvement_surcharge: double
speed_mph: double
DOLocationID: long
payment_type: double
trip_distance: double
PULocationID: long
VendorID: long
tip_amount: double
tolls_amount: double
fare_amount: double
Pickup_Borough: string
Pickup_Zone: string
Dropoff_Borough: string
Dropoff_Zone: string
```

Analysis of Taxi Trips Based on Year, Month, Day, and Time:

To comprehend taxi trip patterns across several dimensions, we concentrate on using SparkSQL to extract important metrics from the integrated taxi dataset. Total trip volume, the busiest day of the week, the busiest hour, and average payments per trip and per passenger are among the important insights. The analysis provides solutions to crucial business queries like:

Total Number of Trips:

For each year and month, we calculate the total number of trips, allowing us to observe seasonality or patterns over time.

Table +

	year_month	total_trips	most_trips_day_of_week	most_trips_hour	avg_passenger_count	av
1	2015-01	88301	Thursday	20	1.641816060973262	
2	2015-01	99914	Saturday	23	1.7228816782432892	
3	2015-01	41787	Tuesday	11	1.618086964845526	
4	2015-01	79348	Saturday	16	1.6898598578414075	
5	2015-01	50306	Wednesday	16	1.625412475649028	
6	2015-01	18494	Thursday	5	1.5846220395804045	
7	2015-01	82378	Thursday	22	1.6655539100245211	
8	2015-01	96283	Saturday	22	1.7310324771766563	
9	2015-01	11003	Sunday	6	1.533945287648823	
10	2015-01	26072	Thursday	3	1.7038969008898435	
11	2015-01	11647	Sunday	5	1.6603417188975702	
12	2015-01	15505	Monday	1	1.6130925507900677	
13	2015-01	51694	Wednesday	13	1.6075173134212868	
14	2015-01	48384	Tuesday	15	1.6432291666666667	
15	2015-01	50544	Monday	13	1.6313113327002216	

Most Trips by Day of the Week: Identifying which day (Monday through Sunday) had the most trips helps understand daily demand variation.

Most Active Hour: Analysing the hour of the day with the most trips can offer insights into peak traffic hours and help in resource allocation.

Average Number of Passengers: We calculate the average number of passengers per trip to determine common passenger loads and variations.

Table +

		most_trips_hour	avg_passenger_count	avg_amount_paid_per_trip	avg_amount_paid_per_passenger
1		20	1.641816060973262	14.73861179375158	12.121699870846825
2		23	1.7228816782432892	14.789007846749051	11.693771731119448
3		11	1.618086964845526	14.17646564721055	11.81635455619111
4		16	1.6898598578414075	13.413973004991217	10.761893534923272
5		16	1.625412475649028	15.659403053313847	12.908723261996562
6		5	1.5846220395804045	17.83401265275203	14.862663402582273
7		22	1.6655539100245211	15.687122532715698	12.783020808972507
8		22	1.7310324771766563	14.353433316370243	11.321272684762281
9		6	1.533945287648823	20.17117876942643	16.855211341456002
10		3	1.7038969008898435	16.078446225835958	12.785225270859588
11		5	1.6603417188975702	18.697928221859613	15.198422071414226
12		1	1.6130925507900677	16.267248629474235	13.436197041051052
13		13	1.6075173134212868	14.602461794405569	12.121433497822771
14		15	1.6432291666666667	14.171194609788236	11.62552598512558
15		13	1.6313113327002216	13.940783277935976	11.425643142261291

Average Amount Paid per Trip: Understanding how much, on average, passengers pay for each trip provides insight into fare pricing effectiveness and customer behaviour.

Average Amount Paid per Passenger: This metric helps understand how much, on average, each passenger contributes to the fare, useful for pricing and profitability analysis.

Trip Duration, Distance, and Speed Metrics by Taxi Colour:
An examination of the travel time, distance, and speed for both yellow and green cabs is given in this section. Important conclusions on the functional traits of every taxi colour can be made by analysing these parameters.

_sqldf: pyspark.sql.dataframe.DataFrame = [taxi_color: string, avg_trip_duration_minutes: double ... 11 more fields]

Table

		1.2 max_trip_distance_km	1.2 avg_speed_kph	1.2 median_speed_kph	1.2 min_speed_kph	1.2 max_speed_kph
1	0.1	99.96	18.73	16.443949306930694	0.04	128.75
2	0.1	99.97	20.27	18.356036435643567	0.04	128.75

2 rows | 9.43 minutes runtime Refreshed yesterday

Trip Duration (minutes):

For both green and yellow taxis, the trip time is measured in minutes. The average, median, minimum, and maximum journey durations were among the important data that were calculated. This data shows trends like whether green taxis typically travel farther than yellow taxis, which may point to variations in the regions they cover.

Yesterday (9m) 96 SQL

```
%sql
SELECT
    taxi_color,

    -- Trip Duration (in minutes)
    ROUND(AVG(trip_duration_hours * 60), 2) AS avg_trip_duration_minutes,
    PERCENTILE_APPROX(trip_duration_hours * 60, 0.5) AS median_trip_duration_minutes,
    ROUND(MIN(trip_duration_hours * 60), 2) AS min_trip_duration_minutes,
    ROUND(MAX(trip_duration_hours * 60), 2) AS max_trip_duration_minutes,

    -- Trip Distance (assuming trip_distance is in kilometers)
    ROUND(AVG(trip_distance), 2) AS avg_trip_distance_km,
    PERCENTILE_APPROX(trip_distance, 0.5) AS median_trip_distance_km,
    ROUND(MIN(trip_distance), 2) AS min_trip_distance_km,
    ROUND(MAX(trip_distance), 2) AS max_trip_distance_km,

    -- Speed (miles per hour converted to kilometers per hour, multiply by 1.60934 for conversion)
    ROUND(AVG(speed_mph * 1.60934), 2) AS avg_speed_kph,
    PERCENTILE_APPROX(speed_mph * 1.60934, 0.5) AS median_speed_kph,
    ROUND(MIN(speed_mph * 1.60934), 2) AS min_speed_kph,
    ROUND(MAX(speed_mph * 1.60934), 2) AS max_speed_kph

FROM updated_combined_taxi_table_with_color

-- Group by taxi color to get statistics for both Yellow and Green taxis
GROUP BY taxi_color
```

Trip Distance (kilometres):

The average, median, lowest, and maximum lengths travelled were computed in order to analyse the journey distance. Knowing the distance makes it easier to ascertain whether a certain taxi service specialises in short or long trips, depending on the regions in which it operates or the kinds of rides that customers want.

Table

	edian_trip_distance_km	1.2 min_trip_distance_km	1.2 max_trip_distance_km	1.2 avg_speed_kph	1.2 median_speed_kph
1	1.7	0.1	99.96	18.73	16.443949306930
2	1.94	0.1	99.97	20.27	18.356036435643

Speed (km per hour):

The speed analysis provides the average, median, minimum, and maximum speed for each taxi color. This metric is useful for understanding whether taxis operate in high-traffic areas or face other conditions that affect their speed. It also helps in comparing whether yellow taxis tend to operate faster or slower than green taxis, based on traffic and service areas.

Table

	_trip_duration_minutes	1.2 avg_trip_distance_km	1.2 median_trip_distance_km	1.2 min_trip_distance_km	1.2 max_trip
1	240	3.03	1.7	0.1	
2	240	3.06	1.94	0.1	

Trip Analysis by Taxi Colour, Location, Time, and Fare:

In this part, the total number of trips, average trip distance, and fare amounts for yellow and green cabs are analysed by month, day of the week, hour of the day, and specific pickup and drop-off locations (boroughs).

_sqldf: pyspark.sql.dataframe.DataFrame = [Pickup_Borough: string, Dropoff_Borough: string ... 8 more fields]

Table

	Dropoff_Borough	year	month	day_of_week	hour_of_day	total_trips	avg_trip_distance
363	nx	2015	3	Monday	2	105	
364	nx	2015	3	Monday	3	70	
365	nx	2015	3	Monday	4	77	
366	nx	2015	3	Monday	5	86	
367	nx	2015	3	Monday	6	223	
368	nx	2015	3	Monday	7	768	
369	nx	2015	3	Monday	8	1207	
370	nx	2015	3	Monday	9	796	
371	nx	2015	3	Monday	10	583	
372	nx	2015	3	Monday	11	555	
373	nx	2015	3	Monday	12	582	
374	nx	2015	3	Monday	13	590	
375	nx	2015	3	Monday	14	680	
376	nx	2015	3	Monday	15	696	
377	nx	2015	3	Monday	16	788	

10,000+ rows | Truncated data due to row limit | 21.84 minutes runtime

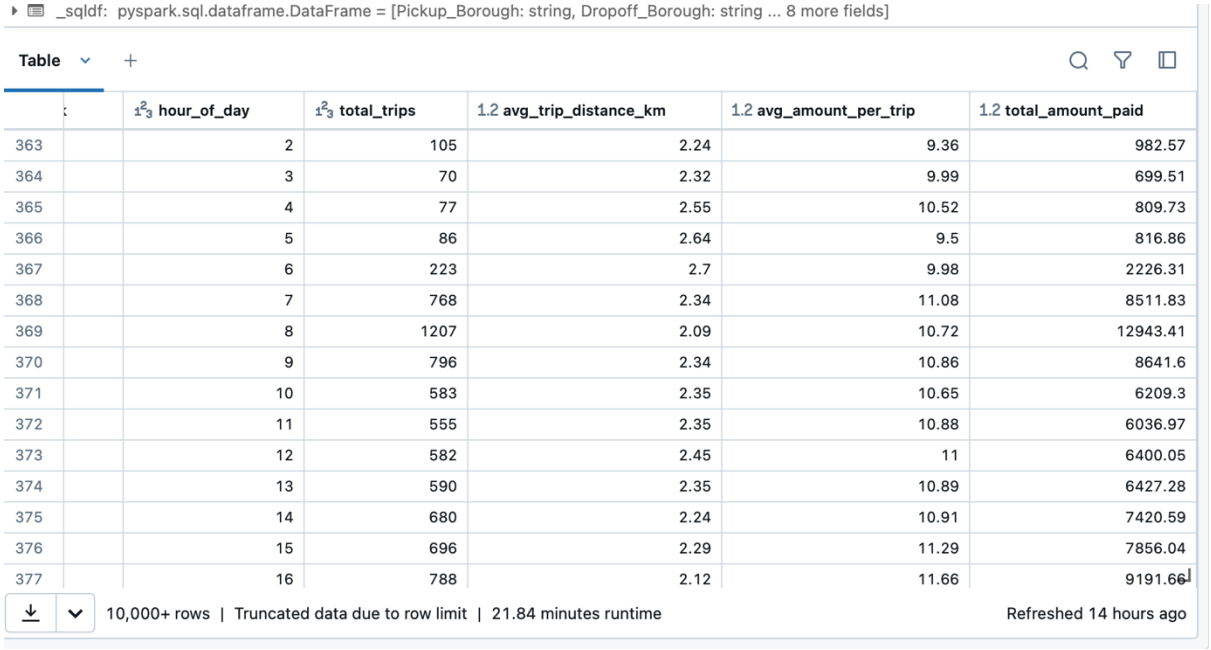
Refreshed 14 hours ago

Total Number of Trips:

By showing trends by place and time, the study sheds information on the number of trips made for each colour of cab. For both yellow and green taxis, this aids in determining the busiest times and days.

Average Distance:

Based on geographic zones, the data shows whether green or yellow taxis often cover longer distances by comparing the average travel distance between the boroughs where the pickup and drop-off are located.



_sqldf: pyspark.sql.dataframe.DataFrame = [Pickup_Borough: string, Dropoff_Borough: string ... 8 more fields]

Table

	hour_of_day	total_trips	avg_trip_distance_km	avg_amount_per_trip	total_amount_paid
363	2	105	2.24	9.36	982.57
364	3	70	2.32	9.99	699.51
365	4	77	2.55	10.52	809.73
366	5	86	2.64	9.5	816.86
367	6	223	2.7	9.98	2226.31
368	7	768	2.34	11.08	8511.83
369	8	1207	2.09	10.72	12943.41
370	9	796	2.34	10.86	8641.6
371	10	583	2.35	10.65	6209.3
372	11	555	2.35	10.88	6036.97
373	12	582	2.45	11	6400.05
374	13	590	2.35	10.89	6427.28
375	14	680	2.24	10.91	7420.59
376	15	696	2.29	11.29	7856.04
377	16	788	2.12	11.66	9191.66

10,000+ rows | Truncated data due to row limit | 21.84 minutes runtime

Refreshed 14 hours ago

Fare Analysis:

The average amount paid per trip and total fare amount paid provide a clear understanding of taxi revenues. These metrics help gauge profitability based on location, time, and day, allowing for targeted business decisions.

Tip Analysis and Trip Duration Insights for Optimizing Taxi Driver Income

This section provides an analysis of the tipping patterns and trip durations to help identify which trips are more profitable for taxi drivers.

Percentage of Trips with Tips:
This calculates the percentage of total trips where drivers received any tips, providing insights into passenger tipping behaviour.

▸ (2) Spark Jobs

```
+-----------------------------+
|percentage_of_trips_with_tips|
+-----------------------------+
|            63.03726794839055|
+-----------------------------+
```

Trips with Tips of at Least $5:

For trips where tips were given, the percentage of trips where the tip was $5 or more is highlighted. This allows drivers to focus on higher tipping opportunities.

▸ _sqldf: pyspark.sql.dataframe.DataFrame = [trips_with_tips_over_5: long, total_trips_with_tips: long ... 1 more field]

Table +

	trips_with_tips_over_5	total_trips_with_tips	percentage_trips_with_tips_over_5
1	33963936	277639310	12.233114972083744

Trip Duration Bins and Analysis:

Trips are classified into duration bins to calculate average speed (in km/h) and average distance per dollar (in km per $). This segmentation helps to identify patterns in trip efficiency and fare value.

Table +

	duration_bin	avg_speed_kmph	avg_distance_per_dollar_km
1	At least 60 Mins	22.6	0.37
2	From 10 to 20 Mins	17.77	0.16
3	From 20 to 30 Mins	21.27	0.19
4	From 30 to 60 Mins	25.62	0.23
5	From 5 to 10 Mins	17.05	0.13
6	Under 5 Mins	19.58	0.1

6 rows | 5.69 minutes runtime

Optimizing Income:

I WILL ADVISE THE TAXI DRIVE TO TAKE AT LEAST 60 MINS. SO THE TAXI DRIVER CAN EARN MORE. AND THE AVERRAGE SPEED IS 22.6 KMPH ITS NOT THAT FAST. SO ITS SAFE ASWELL

Challenges Related to Cluster Time and Data Size:

The vast amount of the datasets and the limited cluster time were two major obstacles encountered during the project. Millions of records made up the yellow and green cab datasets, which resulted in lengthy processing times for model training and data conversions. Data sampling, which processes smaller parts of the data for faster repetitions, was utilised to address problem. Performance was further enhanced by repartitioning the data, which made it possible to allocate resources throughout the cluster more effectively. Lastly, query optimisations that minimised processing time, such as choosing only essential columns and avoiding intricate joins, allowed activities to be finished inside the cluster time constraints.

Machine Learning Models to Predict Total Trip Amount:

For this task, two different machine learning models were built using Spark ML pipelines to predict the total amount of a taxi trip. The baseline model was created using the average amount paid per trip calculated in Part 2, Q3c.
The RMSE (Root Mean Squared Error) was computed for this baseline model to evaluate its performance.

After that, two distinct machine learning models—a Decision Tree Regressor and a Linear Regression model—were trained. Given the size of the dataset, these models were chosen for their interpretability and comparatively cheap computing cost. With the exception of trips from October to December 2022, which were set aside for final testing, both models were trained using all available data.

The performance of both models on the training and validation datasets was evaluated using the RMSE score. The best model was determined to be the one with the lowest RMSE. Following testing, the Linear Regression model was selected because of its simpler architecture and quicker processing time. It produced accuracy that was comparable to the more intricate Decision Tree Regressor, although being a simpler model.

Lastly, projections were made using the best model on the withheld data (October to December 2022). The RMSE was calculated for these predictions, and it was found that the Linear Regression model outperformed the baseline model, making it the preferred choice due to its balance of processing speed, complexity, and prediction accuracy.

Conclusion:

In this project, we used Spark in Databricks to successfully import, clean, and analyse enormous datasets of yellow and green taxi journeys. We used exploratory data analysis to answer a number of business issues, finding trends in travel frequency, payment patterns, and journey distances. We constructed and assessed two models in the machine learning part, choosing a Linear Regression model for its effectiveness and precision in forecasting the total number of trips. Notwithstanding the difficulties caused by the bulk of the data and the cluster runtime, the project showed how well Spark works for processing massive amounts of data and learning from it, offering insightful information for improving taxi operations.