U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

New York City Yellow Taxi Data Traffic Analysis Using Spark

Master in Data Science and Engineering
Big Data

ADRIAN CICAN, FRANCISCO PINTO, JOÃO MATOS
JOÃO SOARES, JOÃO VIEIRA, MANUEL SILVA

16 / 05 / 2025

TABLE OF CONTENTS

1. Overview – Our Goal

To Experiment Spark

Understand its advantages

Understand how it scales

Use a large data set (medium-scale for Spark)

Develop a data engineering/science project

1. Overview – Data Set Context

NEW YORK CITY YELLOW TAXI TRIP RECORDS

New York City Yellow Taxis are one of the city's most iconic forms of public transportation. Operated under the supervision of the **New York City Taxi and Limousine Commission (TLC)**, these taxis are licensed to pick up passengers anywhere in the five boroughs but are especially prevalent in Manhattan.

Records include fields capturing pickup and drop-off dates/times, pickup and drop-off locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts.

TOTAL SIZE

1,45 GB

(for 26 months of data)

DATA SOURCE:

HTTPS://WWW.NYC.GOV/SITE/TLC/ABOUT/TLC-TRIP-RECORD-DATA.PAGE

1. Overview – Task Addressed

Learning Tasks Executed

- L1 - Predict the Fare Amount Value
- L2 - Predict the Tip Value
- L3 – Driver Recommendations
- L4 - Trip Profitability Cluster

Query Tasks Executed

- Q1 - Time Series Analysis
- Q2 - Seasonal Patterns Analysis
- Q3 - Day of Week and Hourly Patterns
- Q4 - Means of Payment Distribution
- Q5 – Taxi Routes Analysis

2. Dataset Profiling – data description

Records from
01-2023
To
02-2025

Total size
1,45 GB

Found 26 parquet files:
1. yellow_tripdata_2023-01.parquet
2. yellow_tripdata_2023-02.parquet
3. yellow_tripdata_2023-03.parquet
4. yellow_tripdata_2023-04.parquet
5. yellow_tripdata_2023-05.parquet
6. yellow_tripdata_2023-06.parquet
7. yellow_tripdata_2023-07.parquet
8. yellow_tripdata_2023-08.parquet
9. yellow_tripdata_2023-09.parquet
10. yellow_tripdata_2023-10.parquet
11. yellow_tripdata_2023-11.parquet
12. yellow_tripdata_2023-12.parquet
13. yellow_tripdata_2024-01.parquet
14. yellow_tripdata_2024-02.parquet
15. yellow_tripdata_2024-03.parquet
16. yellow_tripdata_2024-04.parquet
17. yellow_tripdata_2024-05.parquet
18. yellow_tripdata_2024-06.parquet
19. yellow_tripdata_2024-07.parquet
20. yellow_tripdata_2024-08.parquet
21. yellow_tripdata_2024-09.parquet
22. yellow_tripdata_2024-10.parquet
23. yellow_tripdata_2024-11.parquet
24. yellow_tripdata_2024-12.parquet
25. yellow_tripdata_2025-01.parquet
26. yellow_tripdata_2025-02.parquet

LOAD

PySpark Data Frame
[all_data]

```python
# Helper function to extract year and month from filename
def extract_year_month(filename):
    # Extract the filename from the path
    basename = os.path.basename(filename)
    # Extract the year-month part (assuming format: yellow_tripdata_YYYY-MM.parquet)
    date_part = basename.split('_')[2].split('.')[0]  # This gives 'YYYY-MM'
    year, month = date_part.split('-')
    return int(year), int(month)

# Load each parquet file with year and month columns
all_data = None
file_count = 0

# Process 6 files at a time to avoid memory issues
chunk_size = 6
file_chunks = [parquet_files[i:i + chunk_size] for i in range(0, len(parquet_files), chunk_s

for chunk_index, chunk in enumerate(file_chunks):
    print(f"Processing chunk {chunk_index + 1} of {len(file_chunks)}...")

    chunk_data = None
    for file in chunk:
        year, month = extract_year_month(file)

        # Load the current file
        print(f"Loading {os.path.basename(file)}...")
        current_data = spark.read.parquet(file)

        # Add year and month columns
        current_data = current_data.withColumn("year", lit(year)) \
                                   .withColumn("month", lit(month))

        # Append to the chunk data
        if chunk_data is None:
            chunk_data = current_data
        else:
            chunk_data = chunk_data.unionByName(current_data, allowMissingColumns=True)

        file_count += 1

    # Append to the all_data
    if all_data is None:
        all_data = chunk_data
    else:
        all_data = all_data.unionByName(chunk_data, allowMissingColumns=True)

    # Clear the chunk data to free memory
    chunk_data = None

print(f"Successfully loaded {file_count} parquet files.")
```

2. Dataset Profiling – data description

Shape

86.532.715 instances

Shape

22 attributes

Data Types

11 numerical

Data Types

11 categorical

Data Types

timestamp[ns] **(2)**, long **(4)**, double **(13)**, integer **(2)**, string **(1)**

Duplicates

0 found

Missing Values

10 attributes with missing values

Missing Values

96,80% of instances with at least 1 missing value

L1

3. Learning Tasks Executed

FARE AMOUNT PREDICTION

The basic fare amount is calculated taking into account time and distance with a formula like this:

- **$3.00 – Initial charge when the ride begins.**
- **$0.70 per 1/5 mile when traveling over 12 mph.**
- **$0.70 per 60 seconds when traveling at 12 mph or less, or when stopped in traffic.**

Imagine you are in the position of a customer and want to know in advance how much the ride will cost, it will be complicated to try to emulate all the exact values. The idea is to provide an easy way to get a g.ood estimation. Also, it can be used to prevent abuse or device malfunction on the fare calculation.

Learning Task Goal

Help **Customers** to better anticipate the cost of a trip and prevent charge abuse by service provider.

L1

3. Learning Tasks Executed

FARE AMOUNT PREDICTION

We apply a **Random Forest Regressor** and create a model to predict the expected fare amount given the following variables:

- **Trip distance**
- **Trip duration**
- **Pick Up Location**
- **Drop Off Location**
- **Rate Code ID**
- **Pick Up Hour**
- **Passenger Count**
- **Day Of Week (pick up)**

This can easily be made available by an app.

Results:

RMSE: 2.69

MAE: 0.95

R2: 0.97

Feature Importances:

trip_distance	0.4500
trip_duration	0.2819
PULocationID_ohe	0.1041
DOLocationID_ohe	0.0404
RatecodeID_ohe	0.0027
VendorID_ohe	0.0011
pickup_hour	0.0006
passenger_count	0.0001
pickup_dayofweek	0.0001

3. Learning Tasks Executed

TIP AMOUNT PREDICTION

Usually, customers give tips to the drivers, and that information is being recorded. We wanted to see if we can construct a model to predict the tip amount given by customers, so an experiment on the human behavior.

Learning Task Goal

To predict the Tip Amount to be expected from a Trip.

L2

3. Learning Tasks Executed

TIP AMOUNT PREDICTION

We apply a **Random Forest Regressor** and create a model to predict the expected fare amount given the following variables:

- **Fare Amount**
- **Trip distance**
- **Trip duration**
- **Pick Up Location**
- **Drop Off Location**
- **Rate Code ID**
- **Pick Up Hour**
- **Vendor ID**
- **Passenger Count**
- **Day Off Week (pick up)**
- **is_weekend** [added features to base data set]
- **is_rush_hour** [added feature to base data set]

Results:

RMSE: 2.12

MAE: 1.02

R2: 0.68

Feature Importances:

fare_amount	0.3052
trip_distance	0.1404
trip_duration	0.1083
DOLocationID_ohe	0.0270
PULocationID_ohe	0.0158
pickup_hour	0.0029
VendorID_ohe	0.0026
RatecodeID_ohe	0.0025
passenger_count	0.0005
is_rush_hour	0.0005
pickup_dayofweek	0.0005
is_weekend	0.0003

L3

3. Learning Tasks Executed

DRIVER RECOMMENDATIONS

We used data to uncover patterns that can help taxi drivers make smarter decisions about when and where to work.

Learning Task Goal

Help **NYC taxi drivers maximize their earnings** by identifying

the best times and the best pickup zones to operate.

L3

3. Learning Tasks Executed

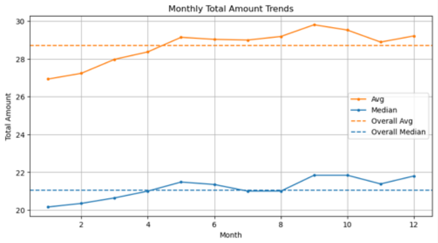
DRIVER RECOMMENDATIONS

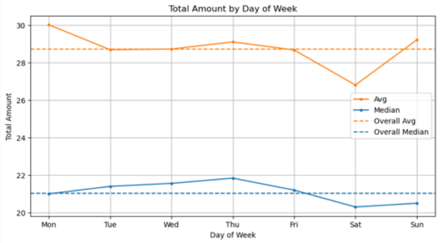
We used data to uncover patterns that can help taxi drivers make smarter decisions about when and where to work.

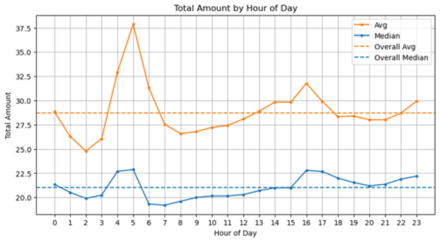
When is it best to work?

- Hours of the day
- Days of the week
- Months of the year

These patterns help drivers decide what time slots are most profitable; whether, for example, weekend nights are better than weekday afternoons.







L3

3. Learning Tasks Executed

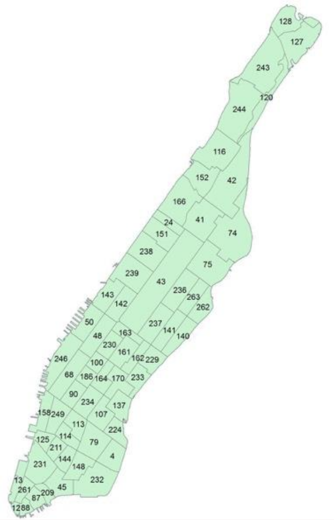
DRIVER RECOMMENDATIONS

We used data to uncover patterns that can help taxi drivers make smarter decisions about when and where to work.

Where should drivers position themselves?

- Pickup locations

This helps drivers prioritize zones where passengers pay more on average, or areas with high trip volume.



L3

3. Learning Tasks Executed

DRIVER RECOMMENDATIONS

We used data to uncover patterns that can help taxi drivers make smarter decisions about when and where to work.

Which features were not used for prediction?

- Drop-off location: drivers don't know where the passenger is headed when they accept the ride.
- Trip duration: drivers also can't predict how long a trip will take due to variables like traffic or destination choice.
- Passenger count: has no meaningful impact on total fare amount.

Results:

RMSE: $13.52

R2: 0.64

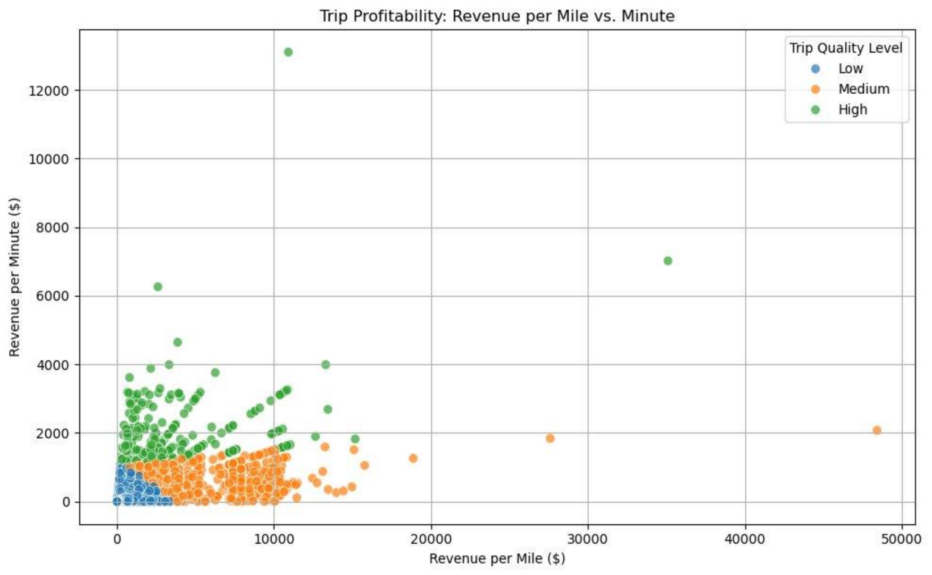
3. Learning Tasks Executed

L4

TRIP PROFITABILITY CLUSTER

- Classify taxi trips based on their profitability, grouping them into three categories (Low, Medium, High)
- These groups were defined using two key metrics:
    - **Revenue per Minute** – the amount of money generated per minute of the trip.
    - **Revenue per Mile** – the amount of money generated per mile traveled.

- The model provides a clear way to analyze and compare different types of trips based on how profitable they are.
- This analysis can help optimize **routes, pricing strategies, and resource allocation**.



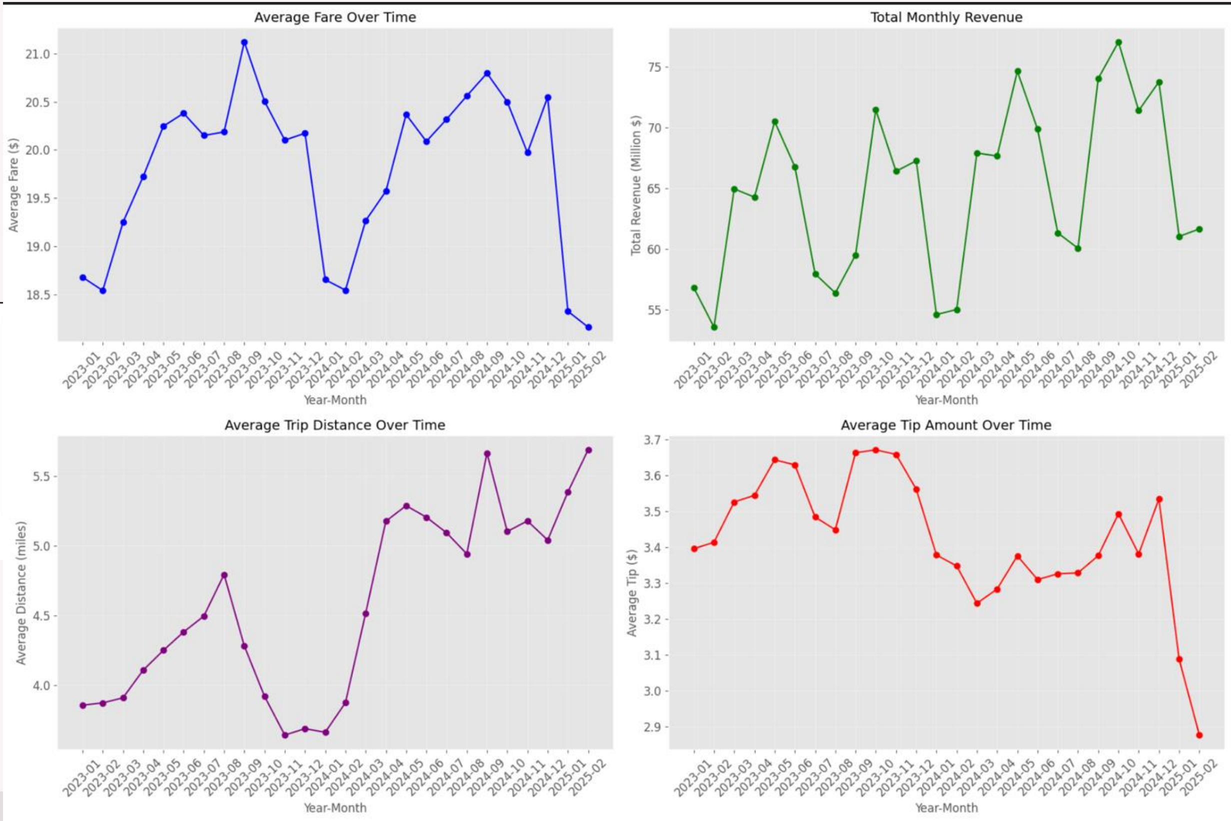
4. Query Tasks Executed

- PERFORMED DATA CLEANING: FILLED MISSING VALUES, REMOVED CORRUPTED OR INVALID RECORDS, AND ENGINEERED NEW FEATURES.
- FINAL DATASET SIZE: 85,086,541 ROWS × 25 COLUMNS.
- UTILIZED SPARK SQL AND DATA FRAME API FOR QUERYING, ACHIEVING SIMILAR PERFORMANCE WITH BOTH APPROACHES (APPROXIMATELY 10–20 SECONDS PER QUERY).
- EXPERIMENTED WITH RDDS, BUT PERFORMANCE AND EASE OF USE WERE INFERIOR COMPARED TO SPARK SQL AND THE DATA FRAME API.

```
temporal_data.createOrReplaceTempView("taxi_trips")
monthly_data = spark.sql("""
    SELECT
        year,
        month,
        year_month,
        COUNT(*) AS ride_count,
        SUM(fare_amount) AS total_fare,
        AVG(fare_amount) AS avg_fare,
        AVG(tip_amount) AS avg_tip,
        AVG(trip_distance) AS avg_distance,
        AVG(trip_duration_minutes) AS avg_duration
    FROM
        taxi_trips
    GROUP BY
        year, month, year_month
    ORDER BY
        year, month
""")
```

4. Time Series Analysis

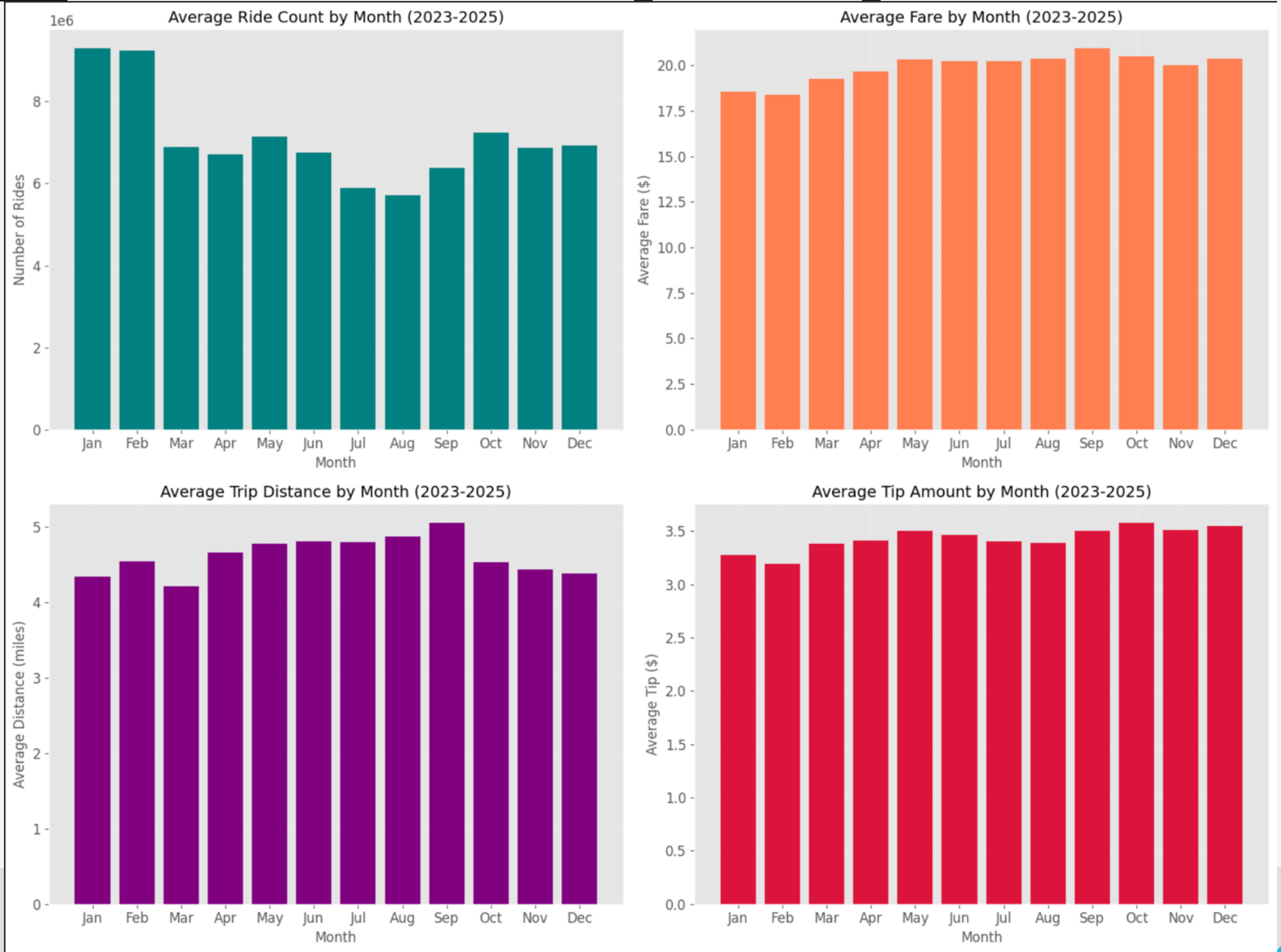
```
monthly_data2 = temporal_data.groupBy("year", "month", "year_month") \
    .agg(
        count("*").alias("ride_count"),
        sum("fare_amount").alias("total_fare"),
        avg("fare_amount").alias("avg_fare"),
        avg("tip_amount").alias("avg_tip"),
        avg("trip_distance").alias("avg_distance"),
        avg("trip_duration_minutes").alias("avg_duration")
    ) \
    .orderBy("year", "month")
```



01

```
month_patterns = temporal_data.groupBy('month') \
    .agg(count('*').alias('ride_count'),
         avg('fare_amount').alias('avg_fare'),
         avg('trip_distance').alias('avg_distance'),
         avg('tip_amount').alias('avg_tip')) \
    .orderBy('month')
```

4. Monthly Analysis

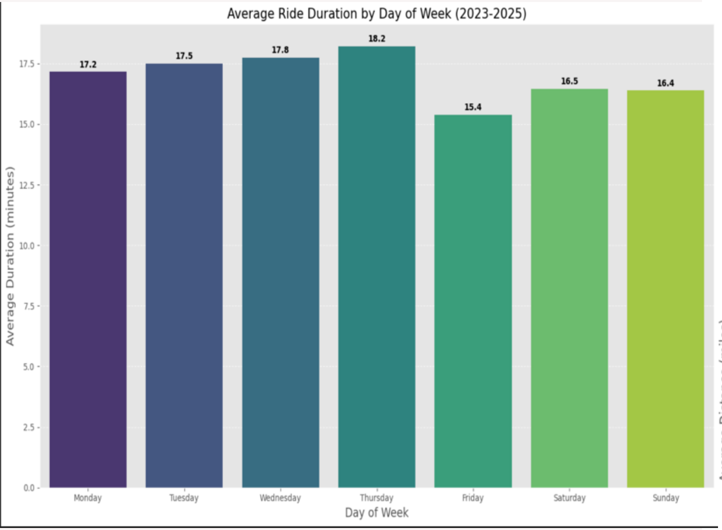


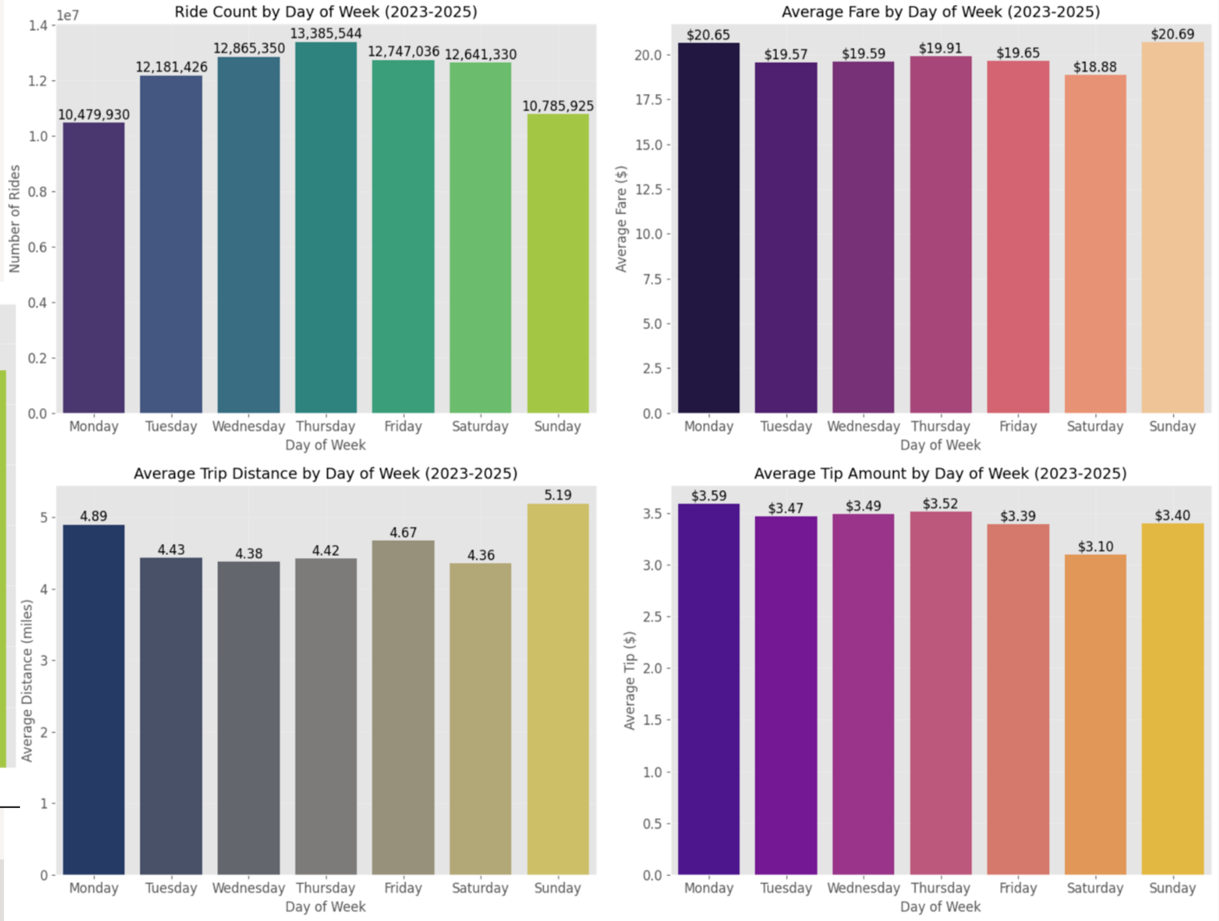
02

```
temporal_data.createOrReplaceTempView("temporal_data")
result = spark.sql("""
    SELECT
        month,
        COUNT(*) AS ride_count,
        AVG(fare_amount) AS avg_fare,
        AVG(trip_distance) AS avg_distance,
        AVG(tip_amount) AS avg_tip
    FROM temporal_data
    GROUP BY month
    ORDER BY month
""")
```

4. Day of Week Patterns

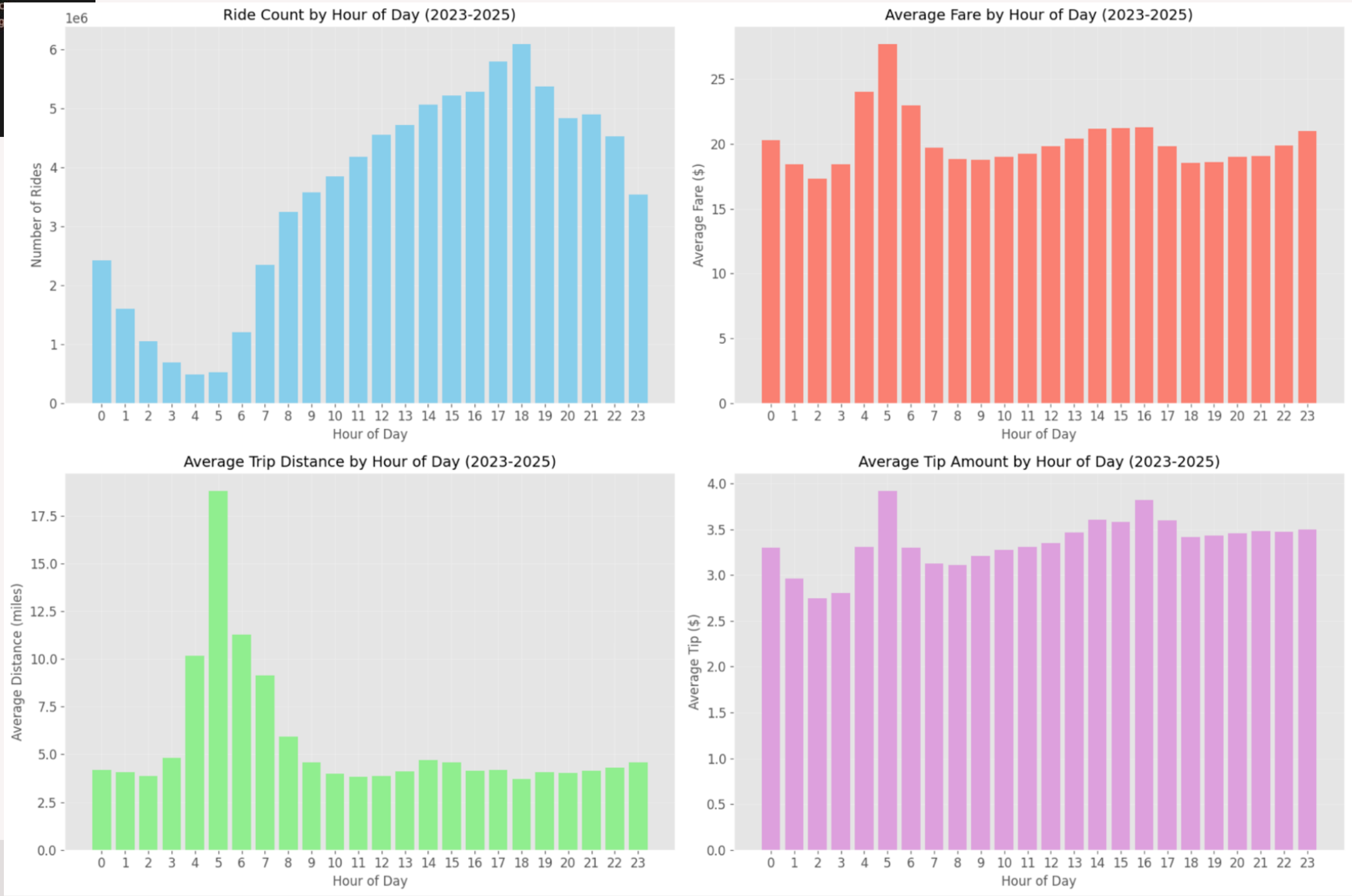
03





```
temporal_data.createOrReplaceTempView("taxi_data_view")
# Use SparkSQL for hourly patterns analysis
hourly_patterns = spark.sql("""
    SELECT
        pickup_hour,
        COUNT(*) AS ride_count,
        AVG(fare_amount) AS avg_fare,
        AVG(tip_amount) AS avg_tip,
        AVG(trip_distance) AS avg_distanc
        AVG(trip_duration_minutes) AS avg
    FROM
        taxi_data_view
    GROUP BY
        pickup_hour
    ORDER BY
        pickup_hour
""")
```

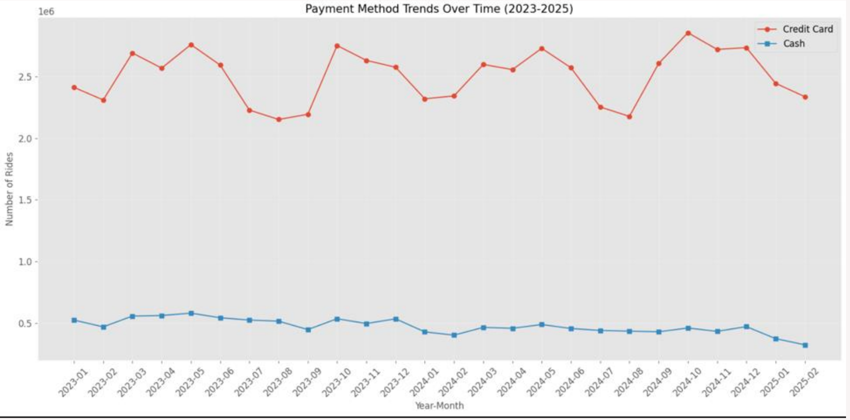
4. Hourly Patterns Analysis

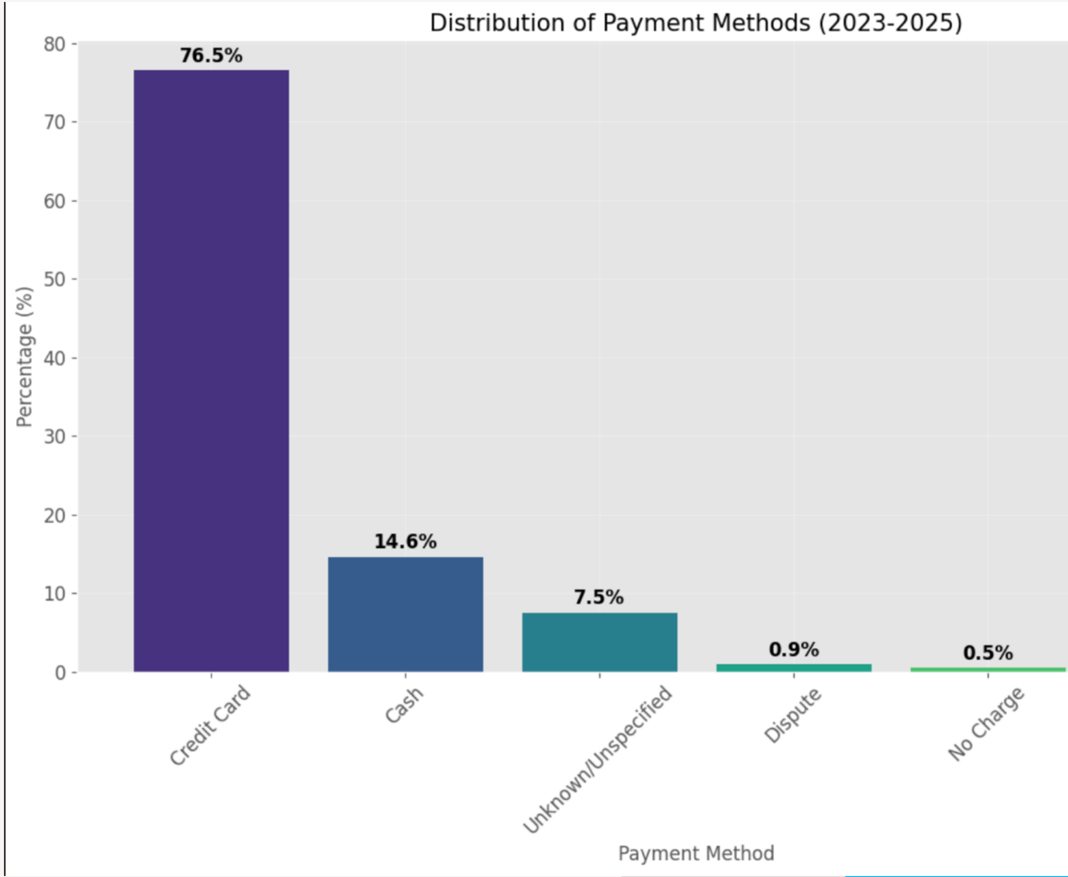


03

```
payment_distribution = temporal_data.groupBy('payment_type') \
    .agg(count('*').alias('ride_count'),
         avg('tip_amount').alias('avg_tip'),
         avg('fare_amount').alias('avg_fare')) \
    .orderBy(desc('ride_count'))
```

4. Payment Method Analysis



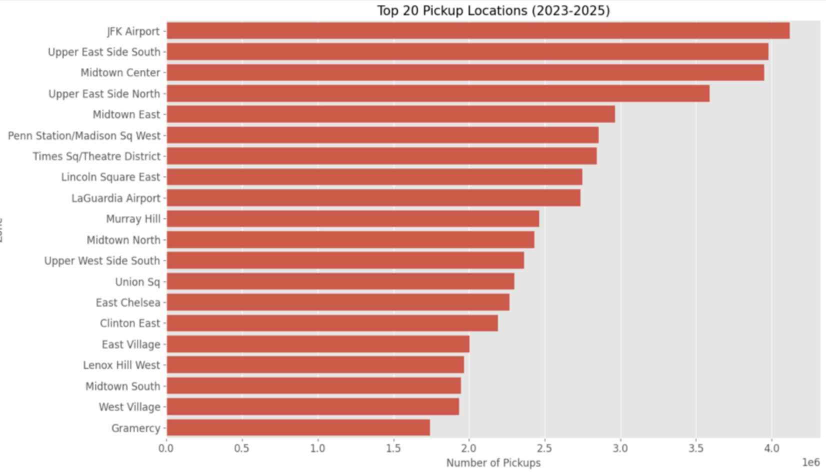


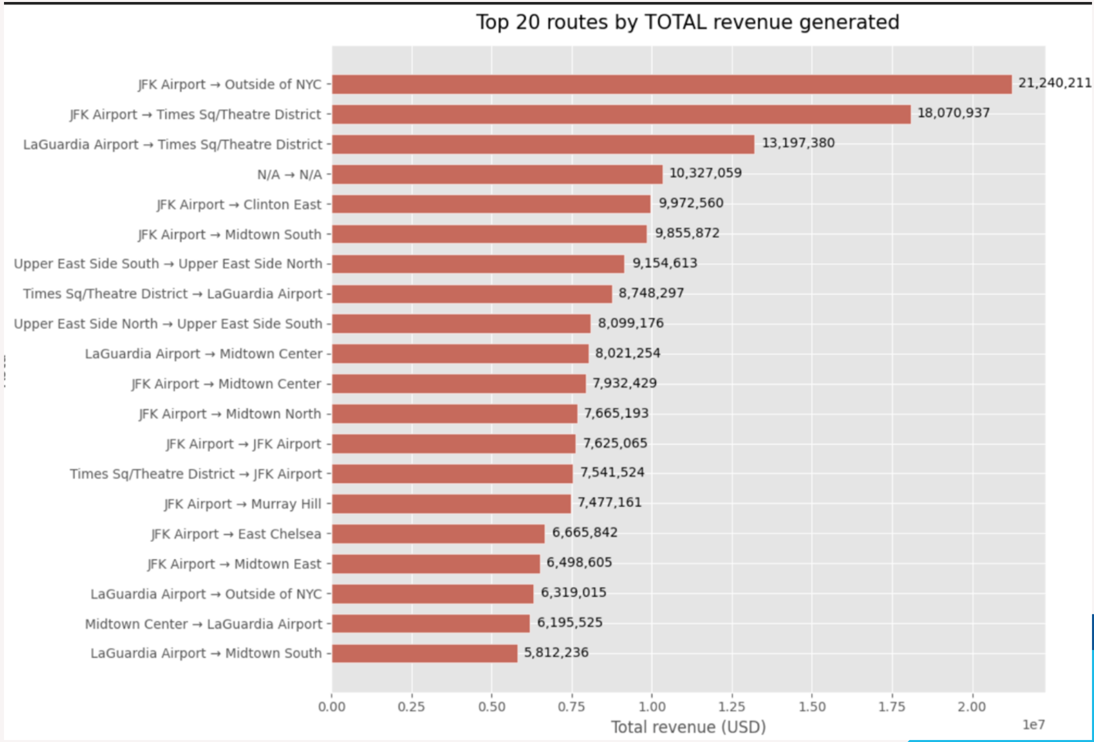
04

4. Top Taxi Routes Analysis

```
# Group by pickup location ID and sum the total_amount
pickup_total_revenue = temporal_data.groupBy('PULocationID') \
    .agg(
        spark_sum('total_amount').alias('total_revenue'),
        count('*').alias('trip_count')
    ) \
    .orderBy(desc('total_revenue'))

taxi_zones_spark_df = spark.read.csv('taxi_zone_lookup.csv', header=True, inferSchema=True)
# Join with zone information to get the names
pickup_revenue_with_names = pickup_total_revenue \
    .join(taxi_zones_spark_df, pickup_total_revenue.PULocationID == taxi_zones_spark_df.LocationID)
    .select(
        col('PULocationID'),
        col('Zone').alias('Pickup_Zone'),
        col('Borough').alias('Pickup_Borough'),
        spark_round(col('total_revenue'), 2).alias('total_revenue'),
        col('trip_count')
    )
```





05

5. Run Time and Scalability Analysis

DRIVER RECOMMENDATIONS MODEL

Predictors:

- PULocationI
- DOLocationID
- pickup_hour
- pickup_day_of_week
- pickup_month

Objective Variable: total_amount

Model: Linear Regression

Data: 01/02/2023 to 31/02/2025 (1.45 GB)

Train / Test Split: 80% / 20%

KPIs: RMSE, R2, Duration

5. Run Time and Scalability Analysis

BASE SPARK SESSION

- **Executor Instances** – 2

Sets the number of distributed workers (processes) to run tasks.

- **Executor Cores** – 1

Defines how many parallel tasks each executor can run.

- **Executor Memory** – 3 GB

Allocates 3 GB of RAM per executor for processing data.

- **Shuffle Partitions** – 50

Sets number of output partitions during shuffles like joins or aggregations.

DATAPROC

Region: europe-west2

№ Master Nodes: 1

№ Worker Nodes: 2

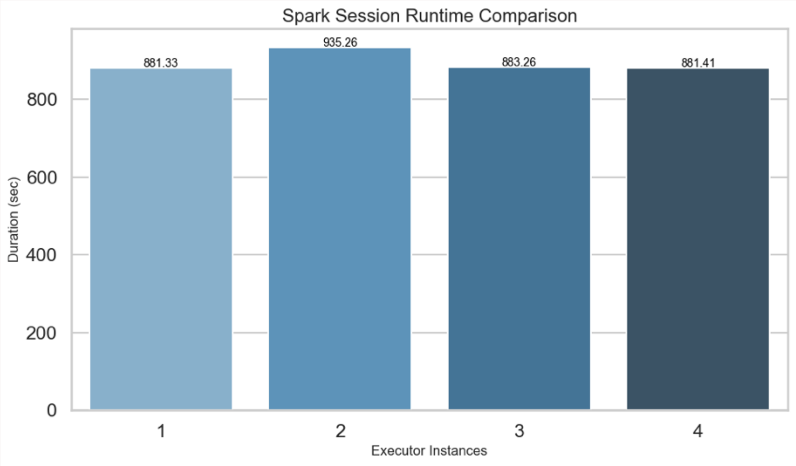
Type of Machine: n4-standard-2

Memory of Master Node: 100 GB

Memory of Worker Nodes: 200 GB

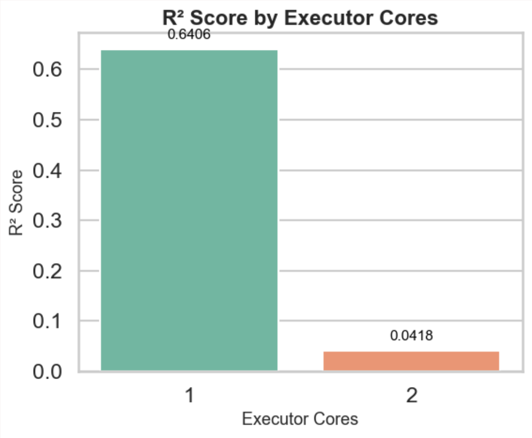
5. Run Time and Scalability Analysis

EXECUTOR INSTANCES



- Increasing the number of instances **did not lead** to a **reduction in training duration**

EXECUTOR CORES

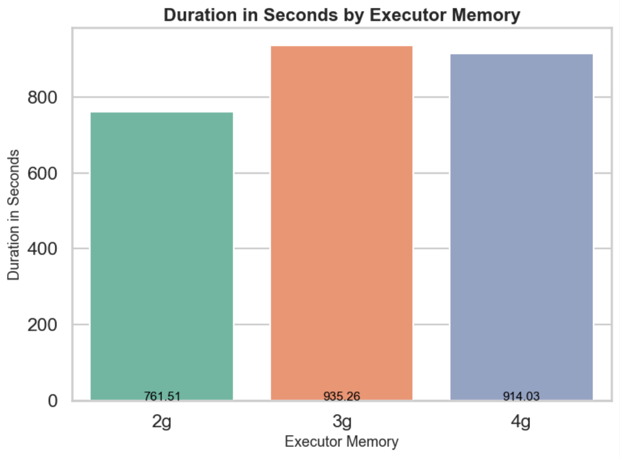


- Increasing the **number of cores** led to **huge fall in model performance**

5. Run Time and Scalability Analysis

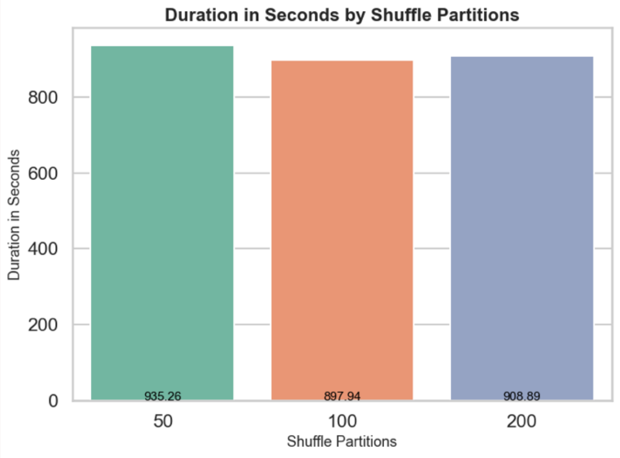
EXECUTOR MEMORY

- Executor Memory had the **most impact in performance**. 2 GB was **not enough** to get all the results

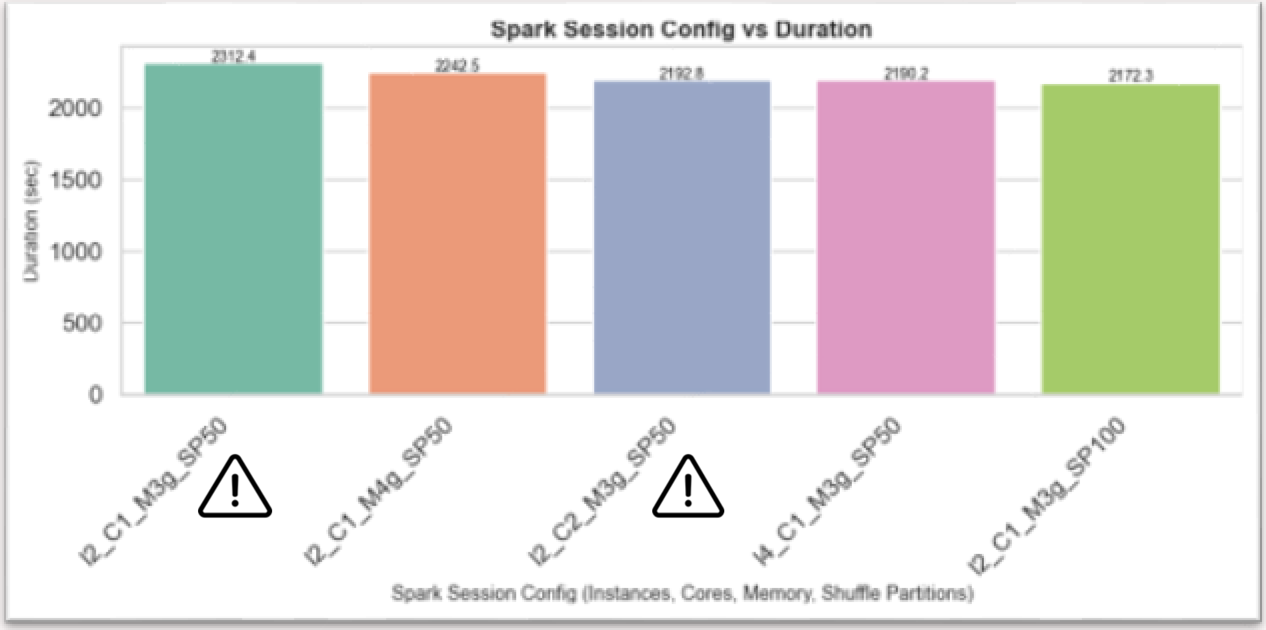


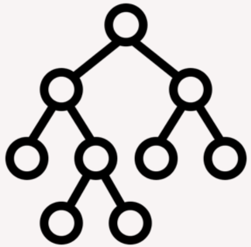
SHUFFLE PARTITIONS

- **100 shuffle partitions** seems to be the ideal number.



5. Run Time and Scalability Analysis





Model: Random Forest

Data: 10,000,000 lines (≈187 MB)

Nº Trees: 20

Max Depth: 10

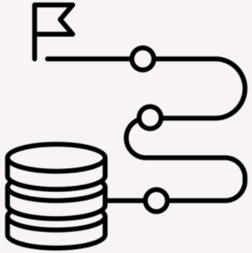
- ➢ The impact of the configurations **is notable** while applying Random Forest. However, their impact is **not significant.**

5. Run Time and Scalability Analysis

❖ Different ML models are affected **differently** by **Spark session configurations**

❖ **More resources ≠ Better performance**

❖ Model training performance depends more on **pipeline design** and **data size** than on **cluster tuning**



6. Looking Back and Ahead

Spark is a great tool for dealing with **larger datasets**

It allows **easy querying**, **machine learning** implementation and **many other functionalities**

Enables **advanced configuration** of parameters for **parallelization** and usage of clusters, **improving its performance**

Test **different functionalities** (Spark Streaming, etc...)

Work with other **cloud clusters** and **technologies** (AWS, Azure, ...)

Continue testing different **spark configurations**

THANKS!

Do you have any questions?

Extra (1)

DATA SET DICTIONARY

Field Name	Description
VendorID	A code indicating the TPEP provider that provided the record. 1 = Creative Mobile Technologies, LLC 2 = Curb Mobility, LLC 6 = Myle Technologies Inc 7 = Helix
tpep_pickup_datetime	The date and time when the meter was engaged.
tpep_dropoff_datetime	The date and time when the meter was disengaged.
passenger_count	The number of passengers in the vehicle.
trip_distance	The elapsed trip distance in miles reported by the taximeter.
RatecodeID	The final rate code in effect at the end of the trip. 1 = Standard rate 2 = JFK 3 = Newark 4 = Nassau or Westchester 5 = Negotiated fare 6 = Group ride 99 = Null/unknown
store_and_fwd_flag	This flag indicates whether the trip record was held in vehicle memory before sending to the vendor, aka "store and forward," because the vehicle did not have a connection to the server. Y = store and forward trip N = not a store and forward trip
PULocationID	TLC Taxi Zone in which the taximeter was engaged.
DOLocationID	TLC Taxi Zone in which the taximeter was disengaged.
payment_type	A numeric code signifying how the passenger paid for the trip. 0 = Flex Fare trip 1 = Credit card 2 = Cash 3 = No charge 4 = Dispute 5 = Unknown 6 = Voided trip
fare_amount	The time-and-distance fare calculated by the meter. For additional information on the following columns, see https://www.nyc.gov/site/tlc/passengers/taxi-fare.page
extra	Miscellaneous extras and surcharges.
mta_tax	Tax that is automatically triggered based on the metered rate in use.
tip_amount	Tip amount – This field is automatically populated for credit card tips. Cash tips are not included.
tolls_amount	Total amount of all tolls paid in trip.
improvement_surcharge	Improvement surcharge assessed trips at the flag drop. The improvement surcharge began being levied in 2015.
total_amount	The total amount charged to passengers. Does not include cash tips.
congestion_surcharge	Total amount collected in trip for NYS congestion surcharge.
airport_fee	For pick up only at LaGuardia and John F. Kennedy Airports.
cbd_congestion_fee	Per-trip charge for MTA's Congestion Relief Zone starting Jan. 5, 2025.

Extra (2)

Linear Regression Results

timestamp	executor_instances	executor_cores	executor_memory	shuffle_partitions	aqe	duration_sec	rmse	r2
2025-05-15 022217	2	1	3g	50	VERDADEIRO	935.26	13.5226	0.6406
2025-05-15 141026	2	1	2g	50	VERDADEIRO	761.51	86.3736	0.0418
2025-05-15 145543	2	1	4g	50	VERDADEIRO	914.03	13.5226	0.6406
2025-05-15 151055	1	1	3g	50	VERDADEIRO	881.33	13.5226	0.6406
2025-05-15 152611	3	1	3g	50	VERDADEIRO	883.26	13.5226	0.6406
2025-05-15 154139	4	1	3g	50	VERDADEIRO	881.41	13.5226	0.6406
2025-05-15 155415	2	2	3g	50	VERDADEIRO	726.3	86.3736	0.0418
2025-05-15 161529	2	1	3g	100	VERDADEIRO	897.94	13.5226	0.6406
2025-05-15 163210	2	1	3g	200	VERDADEIRO	908.89	13.5226	0.6406
2025-05-15 164818	2	1	3g	50	VERDADEIRO	903.05	13.5226	0.6406

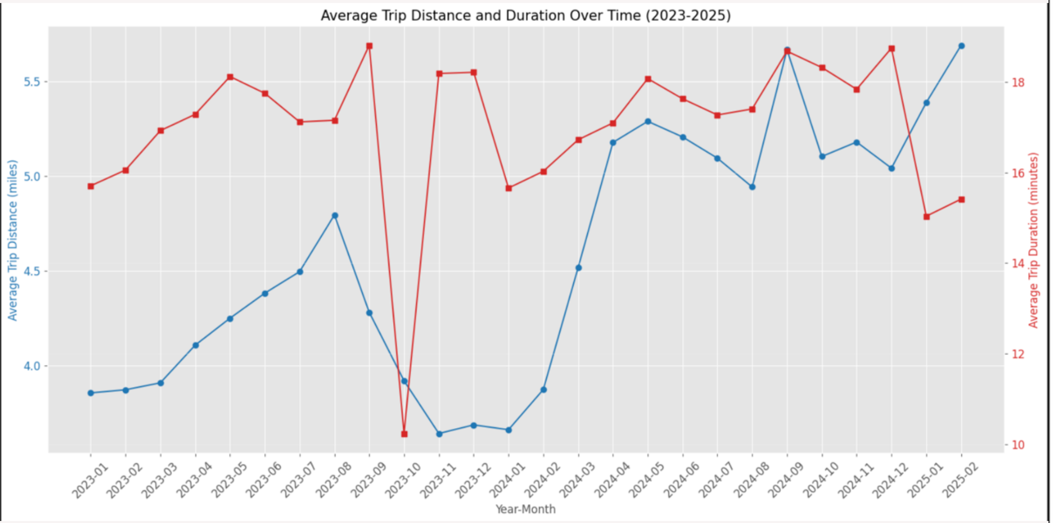
Extra (3)

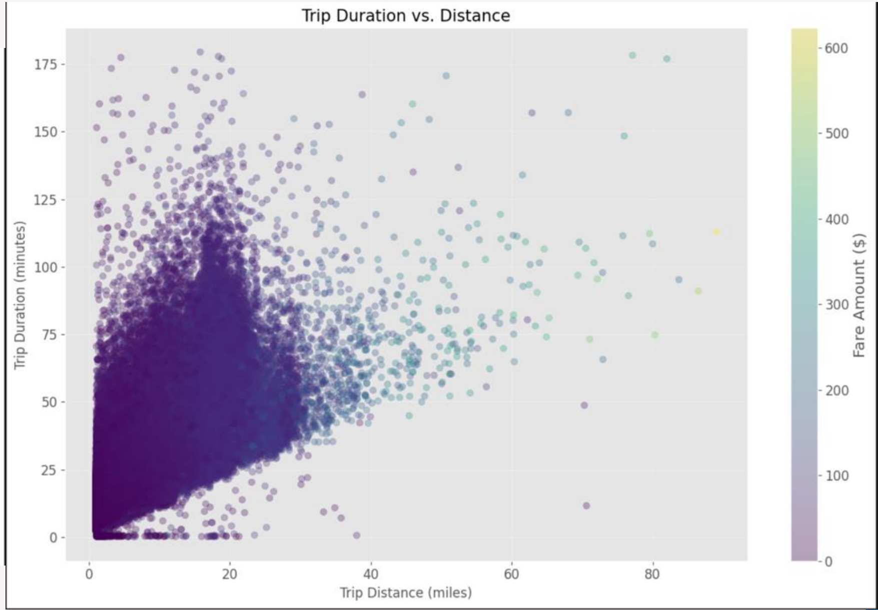
Random Forests Results

timestamp	executor_instances	executor_cores	executor_memory	shuffle_partitions	aqe	duration_sec	rmse	r2
15/05/2025 19:38	2	1	3g	50	VERDADEIRO	250.95	14.5757	0.7239
15/05/2025 20:25	2	1	3g	50	VERDADEIRO	2312.37	273.8961	0.6914
15/05/2025 21:08	2	1	4g	50	VERDADEIRO	2242.49	12.2813	0.6947
15/05/2025 21:46	2	2	3g	50	VERDADEIRO	2192.81	102.1273	0.6705
15/05/2025 22:24	4	1	3g	50	VERDADEIRO	2190.22	13.1926	0.6661
15/05/2025 23:05	2	1	3g	100	VERDADEIRO	2172.35	12.5697	0.6622

Extra (3)

4. Trip Distance and Duration Analysis






```python
# Monthly average trip distance and duration
monthly_trip_metrics = temporal_data.groupBy('year_month') \
    .agg(avg('trip_distance').alias('avg_distance'),
         avg('trip_duration_minutes').alias('avg_duration'),
         avg('fare_amount').alias('avg_fare')) \
    .orderBy('year_month')
```

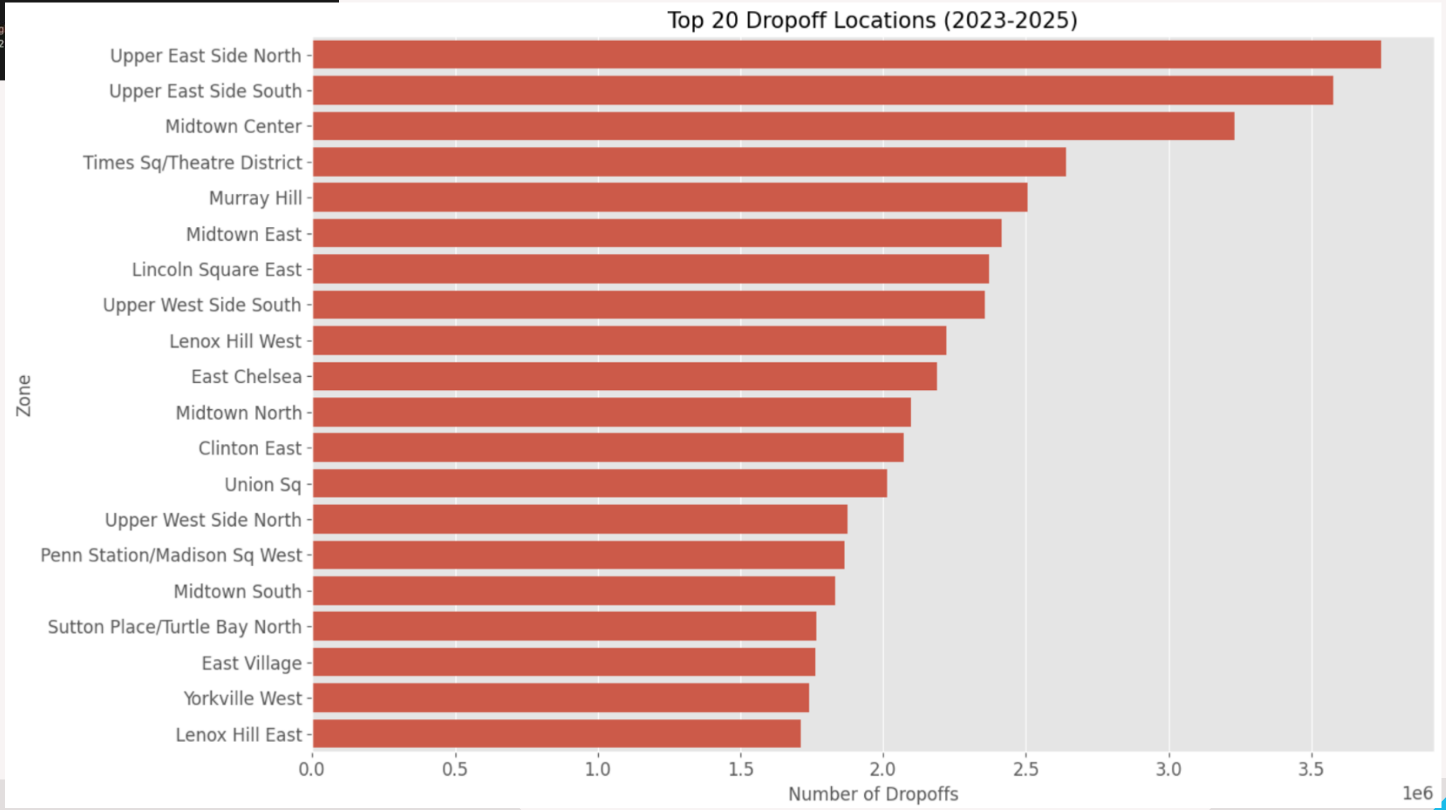
```python
filtered_trips = temporal_data.filter(
    (col('trip_distance') > 1) &
    (col('trip_distance') < 100) &
    (col('trip_duration_minutes') > 0.1)
    (col('trip_duration_minutes') < 180)
)
```

```
# Group by pickup location ID and sum the total_amount
pickup_total_revenue = temporal_data.groupBy('PULocationID') \
    .agg(
        spark_sum('total_amount').alias('total_revenue'),
        count('*').alias('trip_count')
    ) \
    .orderBy(desc('total_revenue'))

taxi_zones_spark_df = spark.read.csv('taxi_zone_lookup.csv', header=True, inferSchema=True)
# Join with zone information to get the names
pickup_revenue_with_names = pickup_total_revenue \
    .join(taxi_zones_spark_df, pickup_total_revenue.PULocationID == taxi_zones_spark_df.LocationID)
    .select(
        col('PULocationID'),
        col('Zone').alias('Pickup_Zone'),
        col('Borough').alias('Pickup_Boroug
        spark_round(col('total_revenue'), 2
        col('trip_count')
    )
```

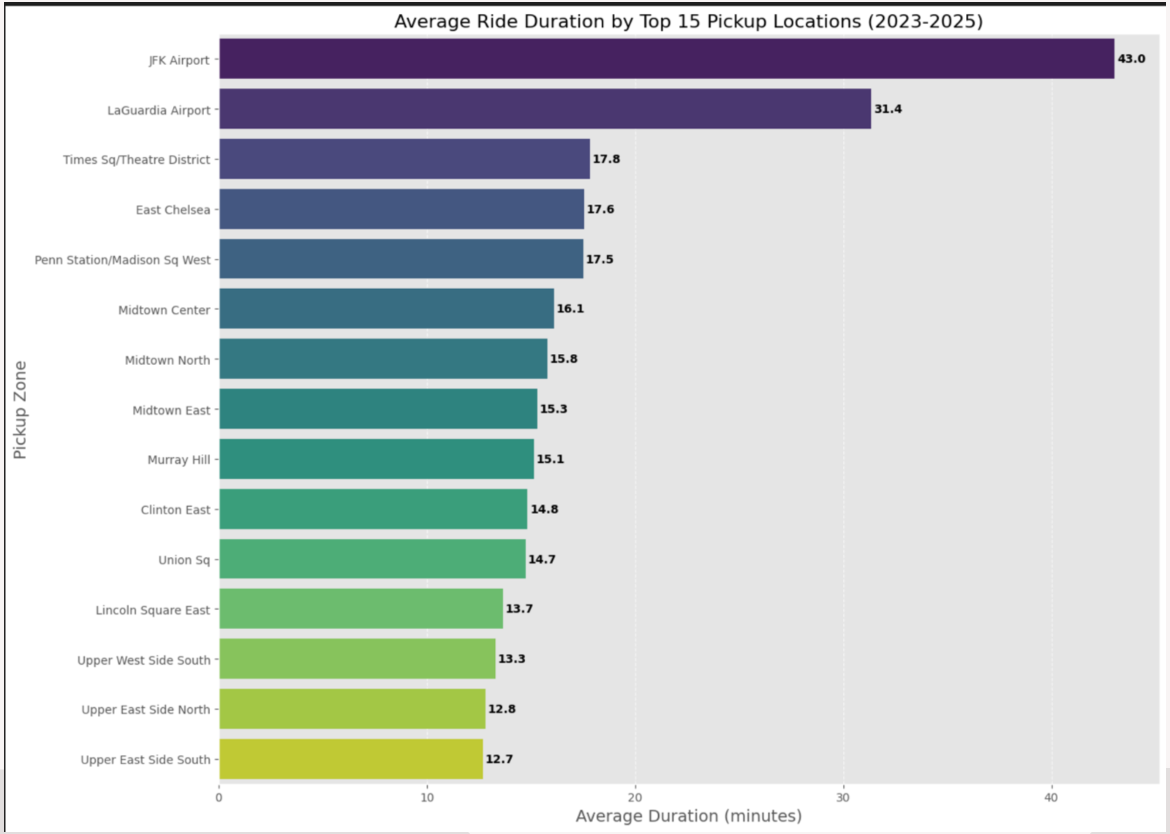
Extra (4)

4. Dropoff Locations Analysis



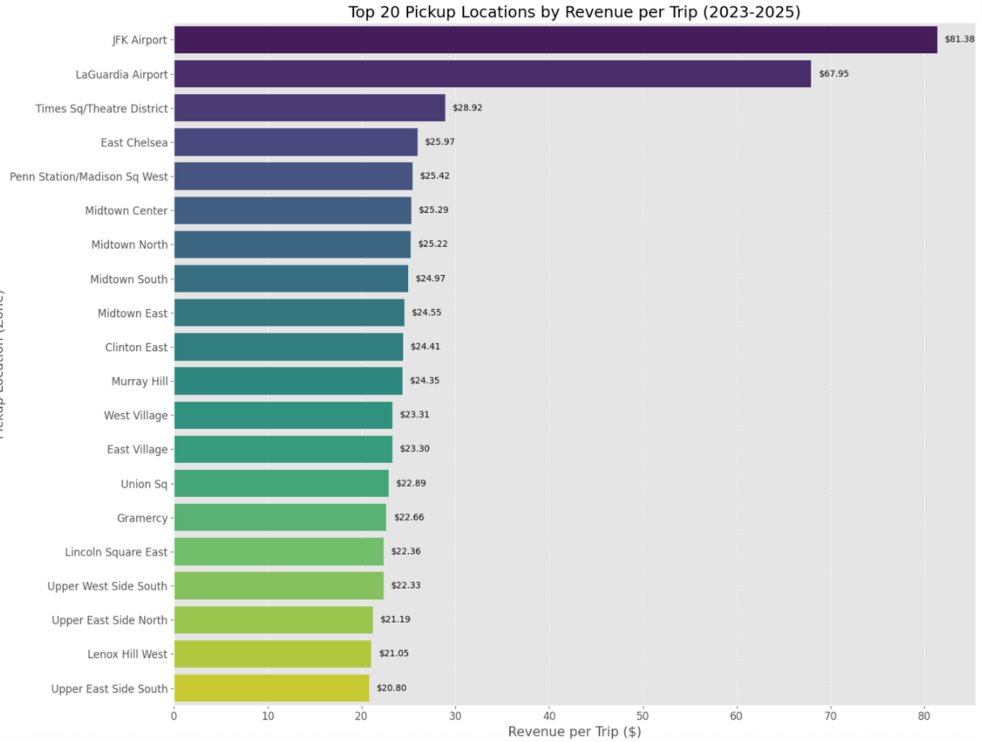
Extra (4)

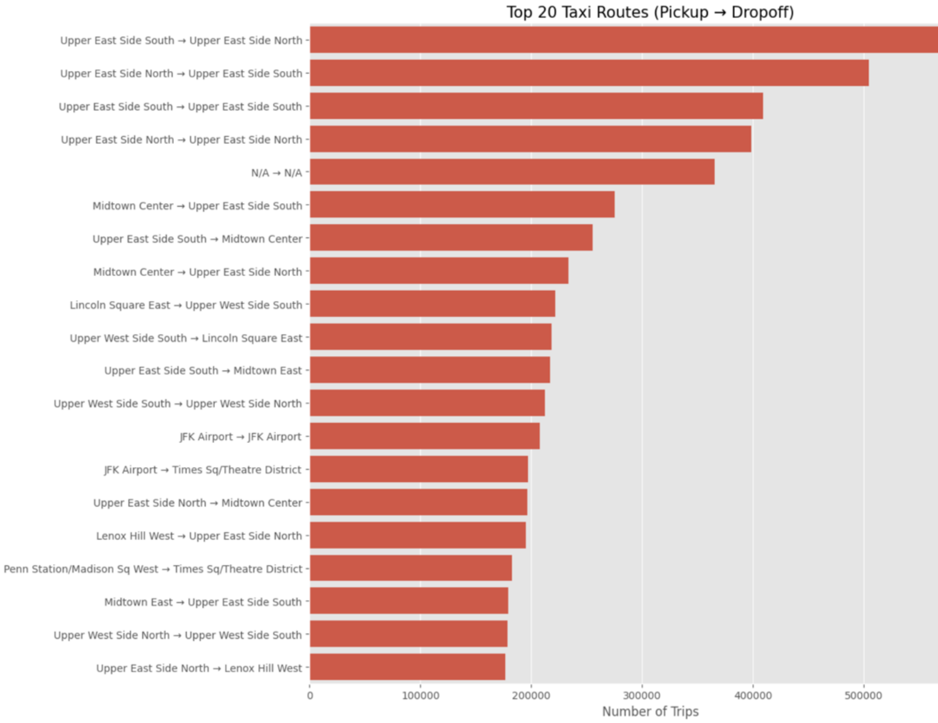
4. Ride Duration Analysis by Day of Week and Pickup Location



4. Top Taxi Routes Analysis

```python
## 1.1  Contagem e receita por rota (pickup+drop-off)
route_stats = (
    temporal_data
    .groupBy("PULocationID","DOLocationID")
    .agg(
        count("*").alias("trip_count"),
        spark_sum("total_amount").alias("total_revenue")
    )
    .withColumn("avg_revenue", col("total_revenue")/col("trip_count"))
)
## 1.2  Juntar nomes de zona (pickup e dropoff)
route_stats_named = (
    route_stats
    .join(taxi_zones_spark_df.withColumnRenamed("Zone","PU_Zone")
                             .withColumnRenamed("Borough","PU_Borough"),
          on=col("PULocationID")==col("LocationID"))
    .drop("LocationID")
    .join(taxi_zones_spark_df.withColumnRenamed("Zone","DO_Zone")
                             .withColumnRenamed("Borough","DO_Borough"),
          on=col("DOLocationID")==col("LocationID"))
    .drop("LocationID")
    .withColumn(
        "route",
        concat_ws(" → ", col("PU_Zone"), col("DO_Zone"))
    )
    # arredondar valores monetários a 2 casas
    .withColumn("total_revenue", spark_round(col("total_revenue"), 2))
```





```
month_patterns = temporal_data.groupBy('month') \
    .agg(count('*').alias('ride_count'),
         avg('fare_amount').alias('avg_fare'),
         avg('trip_distance').alias('avg_distance'),
         avg('tip_amount').alias('avg_tip')) \
    .orderBy('month')
```

Extra (5)

4. Seasonal Patterns Analysis

year	month	year_month	ride_count	total_fare	avg_fare	avg_tip	avg_distance	avg_duration
2023	1	2023-01	3041551	5.6809293499999136E7	18.67773826577267	3.395069153863128	3.8550892225709075	15.707911572090936
2023	2	2023-02	2889044	5.356293198000216E7	18.54001945972514	3.413395839593646	3.87167146294728 6	16.05701666941325
2023	3	2023-03	3373925	6.493803553001208E7	19.24702995176599	3.5254646650419166	3.9082126662562904	16.9267226548715
2023	4	2023-04	3258108	6.4262514370002165E7	19.72387482858216	3.5438779868579915	4.107034407699152	17.285953238300344
2023	5	2023-05	3481359	7.048344950999436E7	20.245958405896765	3.64294058728409	4.24929686653957	18.120959473011467
2023	6	2023-06	3275642	6.67575528800055E7	20.379990511785323	3.6288243800774294	4.381554327365438	17.751439565129143
2023	7	2023-07	2875819	5.794674761000166E7	20.14965045088083	3.4835162052979665	4.495323523489583	17.118752918734867
2023	8	2023-08	2792783	5.637390081999958E7	20.1855642991237	3.4481105048274827	4.794695051494969	17.15581952959787
2023	9	2023-09	2817019	5.9496846760002464E7	21.12049892457327	3.662685228606327	4.280863238763646	18.810690568055257
2023	10	2023-10	3485005	7.145478081995162E7	20.503494491385702	3.6707027651338255	3.9189727819609956	10.237202026969594
2023	11	2023-11	3302675	6.638858278999045E7	20.10145799692384	3.6580172224051184	3.6405988357918697	18.187871674728743
2023	12	2023-12	3333775	6.725025828998107E7	20.17240464337907	3.5611663714574195	3.686229091645144	18.21479107018497
2024	1	2024-01	2927046	5.459636679999159E7	18.65237744811376	3.377807923075358	3.6605391374097427	15.658283778253594
2024	2	2024-02	2966785	5.501609210999343E7	18.544010472613763	3.3472895642943916	3.8729094019281307	16.027887140456325
2024	3	2024-03	3524003	6.789177168996727E7	19.26552607644411	3.2430581926310915	4.515770897470589	16.72433379975503
2024	4	2024-04	3456538	6.764551532996994E7	19.570308594891753	3.282964934859093	5.177514481252299	17.09623012004201
2024	5	2024-05	3663676	7.462081419995418E7	20.367743817945197	3.375399833393333	5.289023802868546	18.074988213115365
2024	6	2024-06	3477261	6.985074654995042E7	20.08786414075631	3.309345108119753	5.2065194099604835	17.62986132859797
2024	7	2024-07	3018041	6.131815761996715E7	20.317204975004366	3.325468123197961	5.094495154306774	17.272310664434485
2024	8	2024-08	2920712	6.004947662997191E7	20.559876026794804	3.327926721294136	4.942407354781552	17.405283272253662

only showing top 20 rows

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

New York City Yellow Taxi Data Traffic Analysis Using Spark

Master in Data Science and Engineering
Big Data

Adrian Cican
Francisco Pinto
João Matos
João Soares
João Vieira
Manuel Silva