Task 1: Merging Datasets

1. (2 points) Load rideshare_data.csv and taxi_zone_lookup.csv.

As the first step in any Spark application, we begin by initialising a Spark session, which serves as the gateway for data operations.

```
# Initialize SparkSession
spark = SparkSession.builder \
    .appName("Rideshare_Taxi_Zone_Join") \
    .getOrCreate()
```

In order to, ensure Spark can collect the necessary datasets, this session is configured to access data stored in an S3 bucket. This includes configuring endpoint URLs and authentication credentials.

```
# Set S3 configurations
hadoopConf = spark.sparkContext._jsc.hadoopConfiguration()
hadoopConf.set("fs.s3a.endpoint", s3_endpoint_url)
hadoopConf.set("fs.s3a.access.key", s3_access_key_id)
hadoopConf.set("fs.s3a.secret.key", s3_secret_access_key)
hadoopConf.set("fs.s3a.path.style.access", "true")
hadoopConf.set("fs.s3a.connection.ssl.enabled", "false")
```

Next, we use the spark.read.csv function to load two datasets into Spark Data Frames, with the header=True this argument is set to indicate that the first row contains column names-
rideshare_data.csv: Data Frame rideshare_df records ridesharing journeys with details such pickup and dropoff locations, fares, and trip durations. These locations are represented as numeric IDs.
taxi_zone_lookup.csv: Data Frame taxi_zone_lookup_df maps LocationID to descriptive data like borough, zone, and service zone. As a result, the rideshare data is enhanced and becomes easier to understand.

```
# Task 1: Load rideshare_data.csv and taxi_zone_lookup.csv
rideshare_df = spark.read.csv("s3a://" + s3_data_repository_bucket + "/ECS765/rideshare_2023/rideshare_data.csv", header=True)
taxi_zone_lookup_df = spark.read.csv("s3a://" + s3_data_repository_bucket + "/ECS765/rideshare_2023/taxi_zone_lookup.csv", header=True)
```

These datasets are being loaded in order to get them ready for additional analysis and enrichment. By combining rideshare_df with taxi_zone_lookup_df, location IDs will be replaced with more detailed location data, making the ridesharing dataset easier to understand and analyse.

2. (5 points) Apply the 'join' function based on fields pickup_location and dropoff_location of rideshare_data table and the LocationID field of taxi_zone_lookup table, and rename those columns as Pickup_Borough, Pickup_Zone, Pickup_service_zone , Dropoff_Borough, Dropoff_Zone, Dropoff_service_zone. The join needs to be done in two steps. once using pickup_location and then output result is joined using dropoff_location. you will have a new dataframe (as shown below) with six new columns added to the original dataset.

```
|-- business: string (nullable = true)
|-- pickup_location: string (nullable = true)
|-- dropoff_location: string (nullable = true)
|-- trip_length: string (nullable = true)
|-- request_to_pickup: string (nullable = true)
|-- total_ride_time: string (nullable = true)
```

|-- on_scene_to_pickup: string (nullable = true)
|-- on_scene_to_dropoff: string (nullable = true)
|-- time_of_day: string (nullable = true)
|-- date: string (nullable = true)
|-- passenger_fare: string (nullable = true)
|-- driver_total_pay: string (nullable = true)
|-- rideshare_profit: string (nullable = true)
|-- hourly_rate: string (nullable = true)
|-- dollars_per_mile: string (nullable = true)
|-- Pickup_Borough: string (nullable = true)
|-- Pickup_Zone: string (nullable = true)
|-- Pickup_service_zone: string (nullable = true)
|-- Dropoff_Borough: string (nullable = true)
|-- Dropoff_Zone: string (nullable = true)
|-- Dropoff_service_zone: string (nullable = true)

Step 1: Joining on pickup_location

Two data frames rideshare_df and taxi_zone_lookup_df is joined by creating joined_df which performs a left join between the two data frame mentioned. The pickup_location from rideshare_df and the LocationID from taxi_zone_lookup_df must match for the join to be valid. In order to prevent duplicate columns after the join, the LocationID column from the lookup DataFrame is removed since it is no longer required. The columns Borough, Zone, and service_zone from taxi_zone_lookup_df are renamed to Pickup_Borough, Pickup_Zone, and Pickup_service_zone, respectively. This is to make it very evident that these columns relate to the ride pickup places.

```
# Join using pickup_location
joined_df = rideshare_df.join(taxi_zone_lookup_df, rideshare_df["pickup_location"] == taxi_zone_lookup_df["LocationID"], "left") \
    .drop("LocationID") \
    .withColumnRenamed("Borough", "Pickup_Borough") \
    .withColumnRenamed("Zone", "Pickup_Zone") \
    .withColumnRenamed("service_zone", "Pickup_service_zone")
```

Step 2: Joining on dropoff_location

A second join using taxi_zone_lookup_df with an alias dropoff_zone, is made using the first joined DataFrame (joined_df). By aliasing the LocationID used for pickup and dropoff, it becomes easier to distinguish between them.
The join condition now matches the dropoff_location from joined_df with the LocationID from the aliased taxi_zone_lookup_df (dropoff_zone).
The LocationID is removed following the join, just like in the first join, and the columns Borough, Zone, and service_zone are renamed to Dropoff_Borough, Dropoff_Zone, and Dropoff_service_zone, respectively, making it obvious that these columns match the dropoff locations.
The outcome of these activities is a new Data Frame that has been added to the original ridesharing dataset with six new columns (Pickup_Borough, Pickup_Zone, Pickup_service_zone, Dropoff_Borough, Dropoff_Zone, and Dropoff_service_zone). Other numeric LocationID has been replaced with these columns, which offer meaningful location data for both pickup and dropoff locations.

```
# Join using dropoff_location with column renaming
joined_df = joined_df.join(taxi_zone_lookup_df.alias("dropoff_zone"), joined_df["dropoff_location"] == col("dropoff_zone.LocationID"), "left") \
    .drop("LocationID") \
```

```
.withColumnRenamed("Borough", "Dropoff_Borough") \
.withColumnRenamed("Zone", "Dropoff_Zone") \
.withColumnRenamed("service_zone", "Dropoff_service_zone")
```

```
Number of rows: 69725864
root
 |-- business: string (nullable = true)
 |-- pickup_location: string (nullable = true)
 |-- dropoff_location: string (nullable = true)
 |-- trip_length: string (nullable = true)
 |-- request_to_pickup: string (nullable = true)
 |-- total_ride_time: string (nullable = true)
 |-- on_scene_to_pickup: string (nullable = true)
 |-- on_scene_to_dropoff: string (nullable = true)
 |-- time_of_day: string (nullable = true)
 |-- date: string (nullable = true)
 |-- passenger_fare: string (nullable = true)
 |-- driver_total_pay: string (nullable = true)
 |-- rideshare_profit: string (nullable = true)
 |-- hourly_rate: string (nullable = true)
 |-- dollars_per_mile: string (nullable = true)
 |-- Pickup_Borough: string (nullable = true)
 |-- Pickup_Zone: string (nullable = true)
 |-- Pickup_service_zone: string (nullable = true)
 |-- Dropoff_Borough: string (nullable = true)
 |-- Dropoff_Zone: string (nullable = true)
 |-- Dropoff_service_zone: string (nullable = true)

2024-04-04 22:23:04,835 INFO datasources.FileSourceStrategy: Pruning di
2024-04-04 22:23:04,835 INFO datasources.FileSourceStrategy: Pushed Fil
2024-04-04 22:23:04,835 INFO datasources.FileSourceStrategy: Post-Scan
2024-04-04 22:23:04,835 INFO datasources.FileSourceStrategy: Output Dat
```

3. (5 points) The date field uses a UNIX timestamp, you need to *convert the UNIX timestamp to the "yyyy-MM-dd" format*. For example, '1684713600' to '2023-05-22'. UNIX timestamp represents the number of seconds elapsed since January 1, 1970, UTC. However, in this dataframe, the UNIX timestamp is converted from (yyyy-MM-dd) without the specific time of day (hh-mm-ss). For example, the '1684713600' is converted from '2023-05-22'.

To convert a column called date from a UNIX timestamp to a human-readable date format ("yyyy-MM-dd"), we use the from_unixtime function. A long integer known as the UNIX timestamp is used to indicate how many seconds have passed since the epoch, or January 1, 1970. To make sure that the timestamp is handled as a numeric type before conversion, the cast("bigint") function is required.

Task 3: Convert UNIX timestamp to "yyyy-MM-dd" format

joined_df = joined_df.withColumn("date", from_unixtime(col("date").cast("bigint"), "yyyy-MM-dd"))

4.(3 points) After performing the above operations, print the number of rows (69725864) and schema of the new dataframe in the terminal. Verify that your schema matches the above resulting schemas. You need to provide the screenshots of your scheme and the number of rows in your report.

As a quick check to make sure the DataFrame size is as intended, the count method is used to return the total number of rows in the DataFrame. The DataFrame's structure, including the names and data types of each column—including the newly added location data and converted date field—is generated by the printSchema method.

Task 4: Print number of rows and schema

print ("Number of rows:", joined_df.count())
joined_df.printSchema()
joined_df.show(5)

```
2024-04-04 22:23:24,373 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 6: Stage finished
2024-04-04 22:23:24,374 INFO scheduler.DAGScheduler: Job 5 finished: showString at NativeMethodAccessorImpl.java:0, took 0.200602 s
2024-04-04 22:23:24,417 INFO codegen.CodeGenerator: Code generated in 25.071544 ms
+--------+---------------+----------------+-----------+-----------------+---------------+------------------+-------------------+-----------+----------+--------------+-------------
---+----------------+-----------+----------------+-------------+--------------------+-------------------+--------------+--------------------+--------------------+
|business|pickup_location|dropoff_location|trip_length|request_to_pickup|total_ride_time|on_scene_to_pickup|on_scene_to_dropoff|time_of_day|      date|passenger_fare|driver_total_
pay|rideshare_profit|hourly_rate|dollars_per_mile|Pickup_Borough|         Pickup_Zone|Pickup_service_zone|Dropoff_Borough|        Dropoff_Zone|Dropoff_service_zone|
+--------+---------------+----------------+-----------+-----------------+---------------+------------------+-------------------+-----------+----------+--------------+-------------
---+----------------+-----------+----------------+-------------+--------------------+-------------------+--------------+--------------------+--------------------+
|    Uber|            151|             244|       4.98|            226.0|          761.0|              19.0|              780.0|    morning|2023-05-22|         22.82|           13
.69|            9.13|      63.18|            2.75|    Manhattan|    Manhattan Valley|        Yellow Zone|      Manhattan|Washington Height...|           Boro Zone|
|    Uber|            244|              78|       4.35|            197.0|         1423.0|             120.0|             1543.0|    morning|2023-05-22|         24.27|            1
9.1|            5.17|      44.56|            4.39|    Manhattan|Washington Height...|          Boro Zone|          Bronx|        East Tremont|           Boro Zone|
|    Uber|            151|             138|       8.82|            171.0|         1527.0|              12.0|             1539.0|    morning|2023-05-22|         47.67|           25
.94|           21.73|      60.68|            2.94|    Manhattan|    Manhattan Valley|        Yellow Zone|         Queens|   LaGuardia Airport|            Airports|
|    Uber|            138|             151|       8.72|            260.0|         1761.0|              44.0|             1805.0|    morning|2023-05-22|         45.67|           28
.01|           17.66|      55.86|            3.21|       Queens|   LaGuardia Airport|           Airports|      Manhattan|    Manhattan Valley|         Yellow Zone|
|    Uber|             36|             129|       5.05|            208.0|         1762.0|              37.0|             1799.0|    morning|2023-05-22|         33.49|           26
.47|            7.02|      52.97|            5.24|     Brooklyn|       Bushwick North|          Boro Zone|         Queens|     Jackson Heights|           Boro Zone|
+--------+---------------+----------------+-----------+-----------------+---------------+------------------+-------------------+-----------+----------+--------------+-------------
---+----------------+-----------+----------------+-------------+--------------------+-------------------+--------------+--------------------+--------------------+
only showing top 5 rows
```

Insight of Task 1:

Insight of the whole Task 1 is how Apache Spark offers a useful perspective on data processing, emphasising the significance of accurate data transformation and enrichment for analytical preparedness. Spark is an effective tool for joining various datasets together using join operations. Rideshare data may be enriched with a wealth of location details, adding context, and making the data more useful for in-depth analysis. Data readability and integrity are preserved by using column management operations following joins. Large-scale data processing can be accommodated by Spark's scalability, as illustrated by its ability to disperse computational burden. The traceability of modifications and the clarity of datasets undergoing various transformations are guaranteed by immutable data frames and aliasing. Furthermore, Spark's ability to handle a variety of data types and its smooth connection with cloud storage services like Amazon S3 demonstrate its critical position in contemporary data engineering and make it an effective tool for converting unprocessed data into meaningful insights. The dataset is made more useful for further analysis by completing the final two tasks. Dates are presented in a consistent manner, and the output of the schema guarantees that the data is correctly formatted and reflects the alterations that have been performed. This helps to improve the quality and dependability of the data, which is important for later uses such as trend analysis, predictive modelling, and decision-making based on the data.

Challenges Faced:

when juggling with the two Data Frames, errors such as a typo in the code, mismatched data types, pesky null values that I didn't anticipate popped, each one of these error were handled with care to keep the data flowing smoothly and made sure my analysis was on track.

Task 2: Aggregation of Data

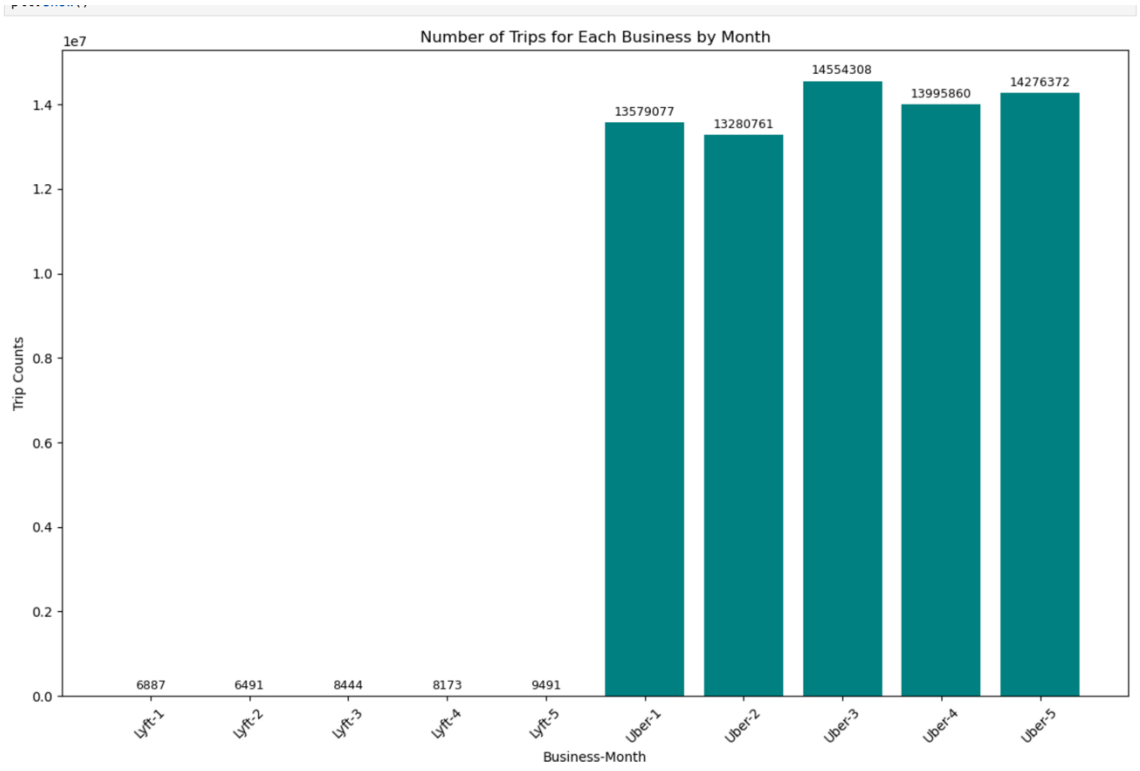
1. (6 points) Count the number of trips for each business in each month. Draw the histogram with 'business-month' on the x-axis and trip counts on the y-axis. For example, assume if the number of trips for Uber in January (i.e. 'Uber-1') is 222222222, 'Uber-1' will be on the x-axis, indicating 222222222 above the bar of 'Uber-1'.

Spark is first set up to process the dataset. Based on pickup and dropoff locations, the ridesharing and taxi zone lookup data are linked after being read from an S3 bucket. The data is then modified to

remove and rename columns that are not needed. After the 'date' column has been formatted correctly, the month is taken out of it. The code computes the trip counts by grouping by 'business' and'month'. Ultimately, the outcomes are shown and preserved in a CSV file for additional examination. Strategic decision-making based on monthly fluctuations in trip counts and business performance trends is made possible by this operation, which offers insights into the usage patterns of various rideshare firms over time.

Part 1 ==== Histogram with 'business-month' on the x-axis and trip counts on the y-axis ====

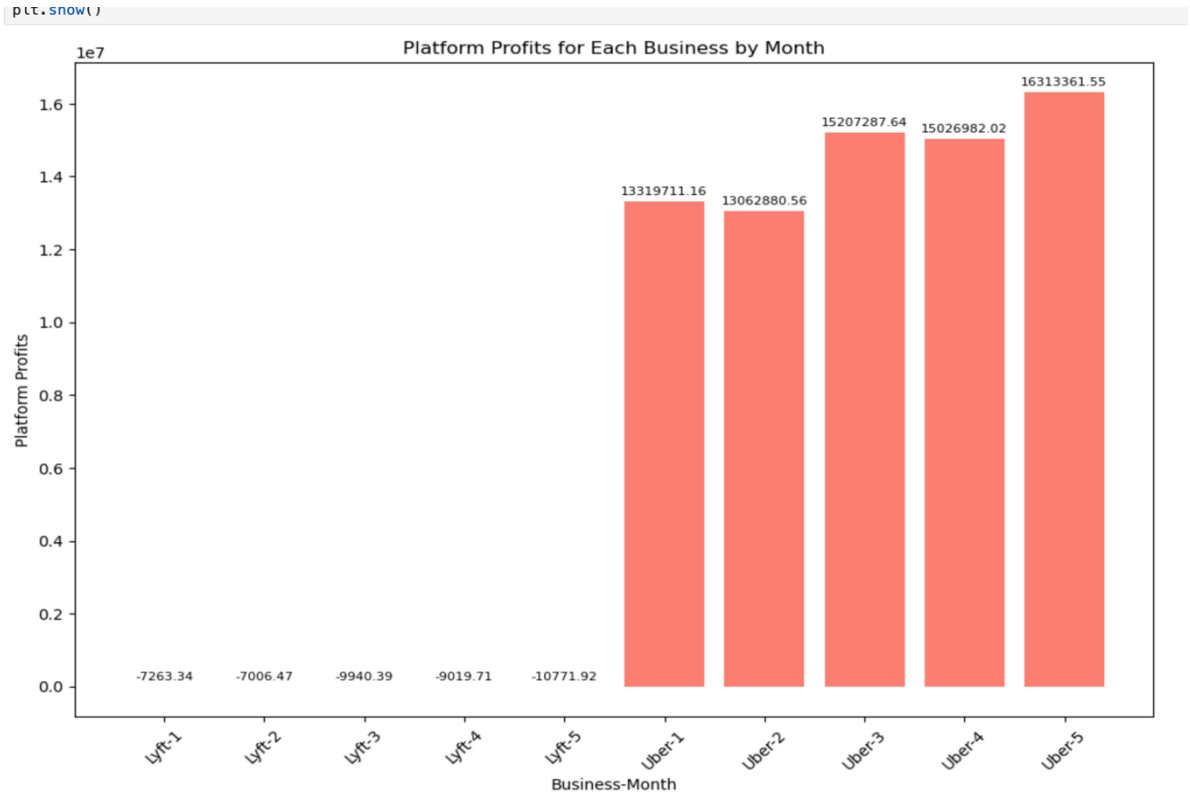
```
df_2 = df_2.withColumn("month", month("date"))

trip_count = df_2.groupBy('business', 'month').count()

trip_count.show(5)

trip_count.coalesce(1).write.mode('overwrite').csv("s3a://" + s3_bucket + "/T2_p1_hist")
```



2. (6 points) Calculate the platform's profits (rideshare_profit field) for each business in each month. Draw the histogram with 'business-month' on the x-axis and the platform's profits on the y-axis. For example, assume if the platform's profits for Uber in January is 33333333, 'Uber-1' will be on the x-axis, indicating 33333333 above the bar of 'Uber-1'.

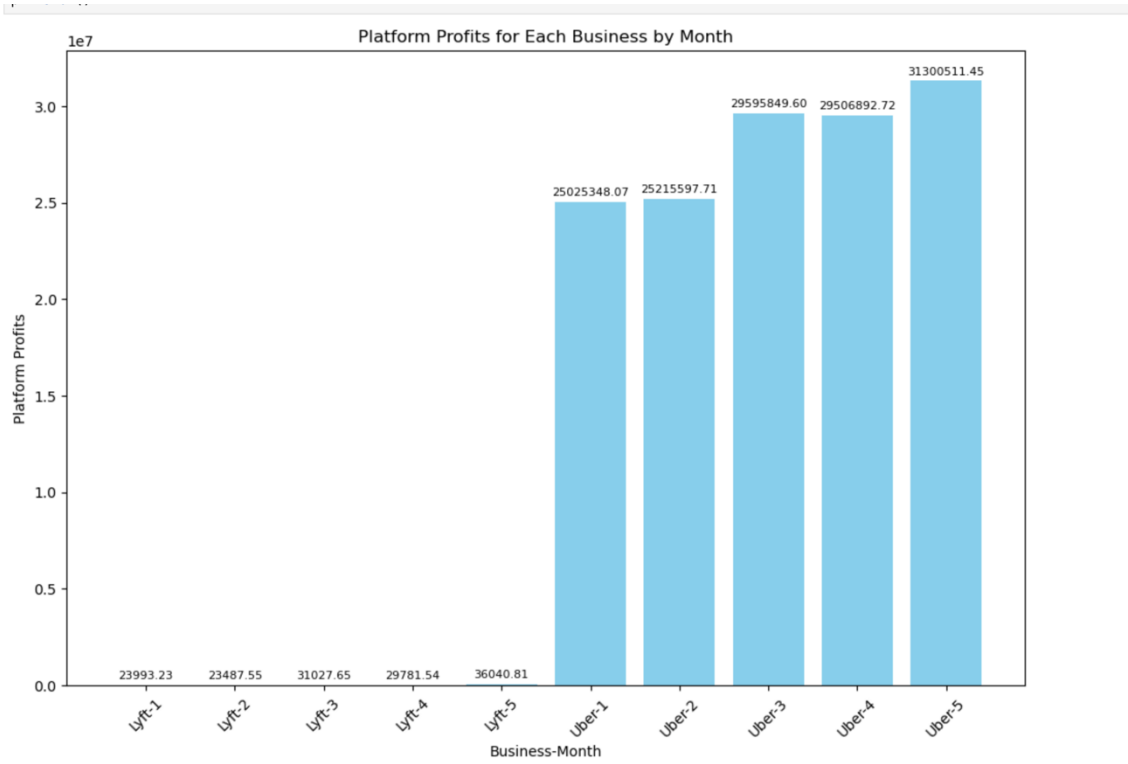
The provided code segment performs an analysis to calculate the platform's profits, specifically the 'rideshare_profit' field, for each business in each month. It begins by converting the 'rideshare_profit' column to a float type. Then, it groups the data by 'business' and 'month' and computes the sum of 'rideshare_profit'. The results are saved to a CSV file for further examination. This operation provides insights into the financial performance of each rideshare business on the platform over time, allowing stakeholders to understand revenue trends and identify opportunities for growth or optimization based on monthly variations in profits.

```
# Part 2 ==== Histogram with 'business-month' on the x-axis and the platform's profits on the y-axis ==== df_2 = df_2.withColumn("rideshare_profit", col("rideshare_profit").cast('float')) platform_prof = df_2.groupBy('business','month').sum('rideshare_profit')
platform_prof.coalesce(1).write.mode('overwrite').csv("s3a://" + s3_bucket + "/T2_p2_hist")
```



3. (5 points) Calculate the driver's earnings (driver_total_pay field) for each business in each month. Draw the histogram with 'business-month' on the x-axis and the driver's earnings on the y-axis. For example, assume if the driver's earnings for Uber in January is 4444444, 'Uber-1' will be on the x-axis, indicating 4444444 above the bar of 'Uber-1'.

The provided code segment calculates the driver's earnings, specifically from the 'driver_total_pay' field, for each business in each month. It begins by converting the 'driver_total_pay' column to a float type. Then, it groups the data by 'business' and 'month' and computes the sum of 'driver_total_pay'. The results are saved to a CSV file for further analysis. This operation provides insights into the earnings of drivers affiliated with each rideshare business on the platform over time, allowing stakeholders to understand driver income trends and make informed decisions related to driver compensation and incentives



Part 3 ==== Histogram with 'business-month' on the x-axis and the driver's earnings on the y-axis ====

```
df_2 = df_2.withColumn('driver_total_pay', col('driver_total_pay').cast('float'))

driver_earn = df_2.groupBy('business','month').sum('driver_total_pay')

driver_earn.coalesce(1).write.mode('overwrite').csv("s3a://" + s3_bucket +"/T2_p3_hist")
```

4.(3 points) When we are analyzing data, it's not just about getting results, but also about extracting insights to make decisions or understand the market. Suppose you were one of the stakeholders, for example, the driver, CEO of the business, stockbroker, etc, What do you find from the three results? How do the findings help you make strategies or make decisions?

these insights derived from the analysis of platform profits, driver earnings, and trip counts empower stakeholders to make informed decisions and develop effective strategies tailored to their specific roles and objectives within the rideshare ecosystem. Whether it's optimizing business operations, enhancing driver satisfaction, or capitalizing on market opportunities, leveraging data-driven insights is key to achieving success in the dynamic rideshare industry.

Task 3: Top-K Processing

Spark is configured to deliver instant feedback on DataFrame operations before the task is started by initialising a SparkSession. Using environment variables, it establishes a secure connection to an AWS S3 bucket. The ridesharing and taxi zone data is then loaded into Spark DataFrames from S3 by the script. By merging these DataFrames via a sequence of left joins on location identifiers, rideshare records are enhanced with informative location information obtained from the taxi zone query. After a join, columns are renamed to make sense. Finally, a temporary Spark SQL view is constructed from the

joined DataFrame, establishing the foundation for future simple data manipulation and querying. It also converts UNIX timestamps into a human-readable date format.

1. (7 points) Identify the top 5 popular pickup boroughs each month. you need to provide a screenshot of your result in your report. The columns should include, Pickup_Borough, Month, and trip_count. you need to sort the output by trip_count by descending in each month. For example,

Pickup_Borough	Month	trip_count
Manhattan	1	5
Brooklyn	1	4
Queens	1	3
Bronx	1	2
Staten Island	1	1
...	...	...

According to task 3 we will generate a structured query that uses Apache Spark's SQL operations to identify the five pickup boroughs that users visit the most frequently each month out of a pooled rideshare dataset. It counts the travels for each distinct pairing after first organising the data by month and borough. Then, using Spark's window functions, we are assigning a row number to each record based on the trip count and sorts the results in descending order to build a ranking list using the grouped data. By applying a row number filter, only the top five records are retained for each month. This produces a carefully curated list that displays the busiest boroughs at the beginning of each month, sorted by decreasing trip count.

```
# Top 5 pickup boroughs for each month

top_pickup_boroughs_each_month = spark.sql("""
    SELECT Pickup_Borough, Month(date) AS Month, COUNT(*) AS trip_count
    FROM joined_df
    GROUP BY Pickup_Borough, Month
    ORDER BY Month, trip_count DESC
""")
# Creating a temporary view to rank the trip counts by month
top_pickup_boroughs_each_month.createOrReplaceTempView("top_pickup_boroughs_each_month")
# Selecting the top 5 popular pickup boroughs for each month
top_5_pickup_boroughs_each_month = spark.sql("""
    SELECT Pickup_Borough, Month, trip_count
    FROM (
        SELECT Pickup_Borough, Month, trip_count,
            ROW_NUMBER() OVER (PARTITION BY Month ORDER BY trip_count DESC) AS rn
        FROM top_pickup_boroughs_each_month
    )
    WHERE rn <= 5
    ORDER BY Month, trip_count DESC
""")
# Displaying the results
top_5_pickup_boroughs_each_month.show(100, truncate=False)
```

```
2024-04-04 23:08:03,487 INFO codegen.CodeGe
2024-04-04 23:08:03,509 INFO codegen.CodeGe
+---------------+-----+-----------+
|Pickup_Borough|Month|trip_count|
+---------------+-----+-----------+
|Manhattan      |1    |5854818    |
|Brooklyn       |1    |3360373    |
|Queens         |1    |2589034    |
|Bronx          |1    |1607789    |
|Staten Island  |1    |173354     |
|Manhattan      |2    |5808244    |
|Brooklyn       |2    |3283003    |
|Queens         |2    |2447213    |
|Bronx          |2    |1581889    |
|Staten Island  |2    |166328     |
|Manhattan      |3    |6194298    |
|Brooklyn       |3    |3632776    |
|Queens         |3    |2757895    |
|Bronx          |3    |1785166    |
|Staten Island  |3    |191935     |
|Manhattan      |4    |6002714    |
|Brooklyn       |4    |3481220    |
|Queens         |4    |2666671    |
|Bronx          |4    |1677435    |
|Staten Island  |4    |175356     |
|Manhattan      |5    |5965594    |
|Brooklyn       |5    |3586009    |
|Queens         |5    |2826599    |
|Bronx          |5    |1717137    |
|Staten Island  |5    |189924     |
+---------------+-----+-----------+

2024-04-04 23:08:03,802 INFO datasources.Fi
2024-04-04 23:08:03,803 INFO datasources.Fi
```

2. (7 points) Identify the top 5 popular dropoff boroughs each month. you need to provide a screenshot of your result in your report. The columns should include, Dropoff_Borough, Month, and trip_count. you need to sort the output by trip count by descending in each month. For example,

Dropoff_Borough	Month	trip_count
Manhattan	2	5
Brooklyn	2	4
Queens	2	3
Bronx	2	2
Staten Island	2	1
...	...	...

According to the requirement the dropoff locations from a dataset of ridesharing journeys are ranked and aggregated by using the Spark SQL script given below. To achieve it, the first step taken is to choose the borough in which each trip ended. Next, it to count the number of trips in each borough, group the trips by month, and finally arranges the count in descending order. To make the rating process easier, a temporary view is made. Then, using a window function, row numbers are assigned to every record within month-by-month partitions based on the quantity of trips. We are making sure that only the top five dropoff boroughs for each month are extracted by limiting the selection to rows with a rank of five or less. The end product is shown in tabular style, sorting the boroughs by month and the count of dropoffs, providing a clear view of the data.

```
# Top 5 drop off boroughs for each month
   top_dropoff_boroughs_each_month = spark.sql("""
      SELECT Dropoff_Borough, Month(date) AS Month, COUNT(*) AS trip_count
      FROM joined_df
      GROUP BY Dropoff_Borough, Month
      ORDER BY Month, trip_count DESC
   """)
   # Creating a temporary view to rank the trip counts by month
```

```
top_dropoff_boroughs_each_month.createOrReplaceTempView("top_dropoff_boroughs_each_month")
# Selecting the top 5 popular dropoff boroughs for each month
top_5_dropoff_boroughs_each_month = spark.sql("""
    SELECT Dropoff_Borough, Month, trip_count
    FROM (
        SELECT Dropoff_Borough, Month, trip_count,
            ROW_NUMBER() OVER (PARTITION BY Month ORDER BY trip_count DESC) AS rn
        FROM top_dropoff_boroughs_each_month
    )
    WHERE rn <= 5
    ORDER BY Month, trip_count DESC
""")
# Displaying the results
top_5_dropoff_boroughs_each_month.show(100, truncate=False)
```

```
+----------------+-----+-----------+
|Dropoff_Borough|Month|trip_count|
+----------------+-----+-----------+
|Manhattan       |1    |5444345    |
|Brooklyn        |1    |3337415    |
|Queens          |1    |2480080    |
|Bronx           |1    |1525137    |
|Unknown         |1    |535610     |
|Manhattan       |2    |5381696    |
|Brooklyn        |2    |3251795    |
|Queens          |2    |2390783    |
|Bronx           |2    |1511014    |
|Unknown         |2    |497525     |
|Manhattan       |3    |5671301    |
|Brooklyn        |3    |3608960    |
|Queens          |3    |2713748    |
|Bronx           |3    |1706802    |
|Unknown         |3    |566798     |
|Manhattan       |4    |5530417    |
|Brooklyn        |4    |3448225    |
|Queens          |4    |2605086    |
|Bronx           |4    |1596505    |
|Unknown         |4    |551857     |
|Manhattan       |5    |5428986    |
|Brooklyn        |5    |3560322    |
|Queens          |5    |2780011    |
|Bronx           |5    |1639180    |
|Unknown         |5    |578549     |
+----------------+-----+-----------+

2024-04-05 00:08:54,521 INFO datasources.F
2024-04-05 00:08:54,522 INFO datasources.F
2024-04-05 00:08:54,522 INFO datasources.F
2024-04-05 00:08:54,522 INFO datasources.F
```

3. (8 points) Identify the top 30 earnest routes. Use 'Pickup Borough' to 'Dropoff_Borough' as route, and use the sum of 'driver_total_pay' field as the profit, then you will have a route and total_profit relationship. The columns should include Route and total_profit. You need to provide a screenshot of your results in your report. Do not truncate the name of routes**. For example,**

Route	total_profit
Queens to Queens	222
Brooklyn to Queens	111
...	...

We have identified the top 30 earnest routes by using the Spark SQL script to determine the overall revenues for various routes that rideshare vehicles travel. By joining the names of the pickup and dropoff boroughs, it defines a route as a trip from one borough to another. The profitability of each

unique route is then calculated by adding together all the driver_total_pay for that route in the query. The Pickup_Borough and Dropoff_Borough pairings are used to aggregate these findings, and the computed total profit is used to arrange them in descending order. The final output will only show the top thirty most profitable routes thanks to the LIMIT 30 clause. A good view of the complete route names and associated earnings is provided by the show method with truncate=False, which displays the full route names without truncation.

```
# Top 30 earning routes
   top_30_earning_routes = spark.sql("""
      SELECT CONCAT(Pickup_Borough, ' to ', Dropoff_Borough) AS Route, SUM(driver_total_pay) AS
total_profit
      FROM joined_df
      GROUP BY Pickup_Borough, Dropoff_Borough
      ORDER BY total_profit DESC
      LIMIT 30
   """)
   # Displaying the results
   top_30_earning_routes.show(100, truncate=False)

   spark.stop()
```

```
2024-04-05 09:25:25,615 INFO codegen.CodeGenerator: Code gen
+------------------------------+-------------------+
|Route                         |total_profit       |
+------------------------------+-------------------+
|Manhattan to Manhattan        |3.3385772555002296E8|
|Brooklyn to Brooklyn          |1.739447214799921E8 |
|Queens to Queens              |1.1470684719998911E8|
|Manhattan to Queens           |1.0173842820999998E8|
|Queens to Manhattan           |8.603540026000004E7 |
|Manhattan to Unknown          |8.010710241999996E7 |
|Bronx to Bronx                |7.414622575999323E7 |
|Manhattan to Brooklyn         |6.799047559E7       |
|Brooklyn to Manhattan         |6.317161604999997E7 |
|Brooklyn to Queens            |5.0454162430000074E7|
|Queens to Brooklyn            |4.7292865360000156E7|
|Queens to Unknown             |4.629299990000004E7 |
|Bronx to Manhattan            |3.248632517000009E7 |
|Manhattan to Bronx            |3.197876345000006E7 |
|Manhattan to EWR              |2.375088861999999E7 |
|Brooklyn to Unknown           |1.0848827569999997E7|
|Bronx to Unknown              |1.0464800209999995E7|
|Bronx to Queens               |1.0292266499999996E7|
|Queens to Bronx               |1.018289873E7       |
|Staten Island to Staten Island|9686862.450000014   |
|Brooklyn to Bronx             |5848822.56          |
|Bronx to Brooklyn             |5629874.410000001   |
|Brooklyn to EWR               |3292761.7100000014  |
|Brooklyn to Staten Island     |2417853.8200000003  |
|Staten Island to Brooklyn     |2265856.4600000004  |
|Manhattan to Staten Island    |2223727.3700000006  |
|Staten Island to Manhattan    |1612227.7200000002  |
|Queens to EWR                 |1192758.6599999997  |
|Staten Island to Unknown      |891285.8100000004   |
|Queens to Staten Island       |865603.3799999999   |
+------------------------------+-------------------+

2024-04-05 09:25:25,624 INFO server.AbstractConnector: Stoppe
```

4. (3 points) Suppose you were one of the stakeholders, for example, either the driver, CEO of the business, or stockbroker, etc, what do you find (i.e., insights) from the previous three results? How do the findings help you make strategies or make decisions?

The information gained from these findings is priceless for everybody involved in the ridesharing business, including drivers, CEOs, and investors. The most popular pickup and dropoff boroughs each month's data reveals where rideshare services are most in demand, indicating where drivers are most needed and where more aggressive marketing may take place to increase market share. This is strategic data that can help a CEO decide how best to use resources to increase profitability and satisfy customers. Knowing these patterns as a driver will help me decide where and when to work to get the most fares.
Let's get started move on to the highest paying routes; this data is quite valuable. It provides a comprehensive image of financial hotspots by highlighting not only the locations of demand but also the locations of highest profit margins. A business leader may use this information to examine why some routes do better than others, which could result in changes to pricing tactics or the focus of partnerships or infrastructure improvements in those areas. A driver could use this information to concentrate on the more profitable routes. These insights help investors and stockbrokers make informed investment decisions by providing a comprehensive picture of the company's operating strengths and possible growth prospects. Essentially, these analytics inform more astute, data-driven strategies for competitive positioning in the ridesharing market, financial planning, and operational optimisation.

Insight of Task 3:

The monthly examination of the most popular pickup boroughs provides an insight into the rhythm of urban mobility, illustrating how local events and seasonal variations influence travel patterns. With this information, rideshare businesses can better allocate their fleet, adding more drivers to high-demand areas or providing incentives to balance demand. These observations will be very helpful to local leaders in designing improved infrastructure and public transit systems.
Examining the busiest drop-off locations provides a map of where people congregate, whether at work or at home, and hence signals for specific vehicle placement and infrastructure improvements. Additionally, having this knowledge makes it easier to foresee and adapt to changes in travel patterns brought on by seasonal or unique occurrences.
The examination of the most profitable routes highlights key areas of demand and revenue generation for rideshare businesses, indicating where to focus service and consider fare adjustments. Urban planners can use this information to guide improvements in municipal connections and services by identifying important transit routes and underserved areas.

Challenges Faced:

While trying to figure out which routes were the most profitable, I ran into problems with driver_total_pay entries' accuracy. To fix this, I implemented a validation mechanism that ensures data integrity and filters abnormalities, maintaining the veracity of the profit analysis. Additionally, because of the complexity and amount of data in the query, the first method of profit aggregation resulted in processing delays. I was able to remove these bottlenecks by fine-tuning Spark's parallel processing settings, optimising the aggregation logic, and selectively caching data. This allowed me to strike a compromise between query performance and the accuracy of the resultant insights.

Task4:

1. (4 points) Calculate the average 'driver_total_pay' during different 'time_of_day' periods to find out which 'time_of_day' has the highest average 'driver_total_pay'. You need to provide a screenshot of this question in your report. The columns should include time_of_day,

average_drive_total_pay. You need to sort the output by average_drive_total_pay by descending. For example,

time_of_day	average_drive_total_pay
afternoon	25
night	22
evening	20
morning	18

We are calculating the average total pay for drivers across different times of the day, providing a clear indicator of when drivers earn the most. By grouping the rideshare data by 'time_of_day' and calculating the average of 'driver_total_pay' for each group, we get a concise view of driver earnings segmented by the time. The results are then sorted in descending order of average pay, showing us immediately which time slots are the most profitable for drivers.
To get the average 'driver_total_pay' for each unique 'time_of_day' period in the ridesharing data, a Spark Data Frame operation is run. The 'time_of_day' column, which divides each journey into time slots like morning, afternoon, evening, and night, is used to first group the data. Subsequently, the average total pay for drivers within each time category is calculated via an aggregation function (F.avg). comprehension the pay dynamics during the day requires a comprehension of this aggregate. Lastly, to highlight the times when drivers are often most profitable, the results are sorted by average driver total pay (average_driver_total_pay) in descending order. The data is prepared for analysis by using orderBy(F.desc("average_driver_total_pay")) to guarantee that the first row of the output shows the time slot with the highest average pay for drivers.

```
avg_driverpay_df = rideshare_df.groupBy("time_of_day") \
                .agg(F.avg("driver_total_pay").alias("average_driver_total_pay")) \
                .orderBy(F.desc("average_driver_total_pay"))
```

```
2024-04-05 03:38:36,165 INFO codegen.CodeGenerator: Code g
+-----------+------------------------+
|time_of_day|average_driver_total_pay|
+-----------+------------------------+
|  afternoon|      21.212428756593535|
|      night|      20.087438003592705|
|    evening|      19.777427702398395|
|    morning|      19.633332793944845|
+-----------+------------------------+

2024-04-05 03:38:36,218 INFO datasources.FileSourceStrateg
```

2. (4 points) Calculate the average 'trip_length' during different time_of_day periods to find out which 'time_of_day' has the highest average 'trip_length'. You need to provide a screenshot of this question in your report. The columns should include, time_of_day, average_trip_length. You need to sort the output by average_trip_length by descending. For example,

time_of_day	average_trip_length
night	25
morning	22

```
| afternoon   |          20        |
| evening     |          18        |
```

The code groups the rideshare data by 'time_of_day' and calculates the average 'trip_length' for each group using Spark's aggregation methods. This indicates the times of day when rides are typically longer or shorter. Next, the results are arranged by 'average_trip_length' in descending order, which immediately indicates when journeys typically tend to be the longest.
After segmenting the dataset by "time_of_day," such as morning or night, the method determines the average "trip_length" for trips that take place during each of these time periods. The aggregation function that is built into Spark, which computes the average, is used to accomplish this. Following computation, these averages are arranged in an ordered list, with the longest averages corresponding to the earliest times of the day.

```
avg_trip_len_df = rideshare_df.groupBy("time_of_day") \
                .agg(F.avg("trip_length").alias("average_trip_length")) \
                .orderBy(F.desc("average_trip_length"))

   joined_df = avg_driverpay_df.join(avg_trip_len_df, "time_of_day")
```

```
2024-04-05 04:36:13,490 INFO scheduler.DAGScheduler: Jo
+-----------+-------------------+
|time_of_day|average_trip_length|
+-----------+-------------------+
|      night|   5.32398480196174|
|    morning|  4.927371866442788|
|  afternoon|  4.861410525661207|
|    evening|  4.484750367447519|
+-----------+-------------------+
```

3. (5 points) Use the above two results to calculate the average earned per mile for each time_of_day period. You need to use 'join' function first and use average_drive_total_pay divided by average_trip_length to get the average_earning_per_mile. You need to provide a screenshot of this question in your report. The columns should include, time_of_day, and average_earning_per_mile. For example,

```
| time_of_day | average_earning_per_mile |
|-------------|--------------------------|
| night       |          7               |
| morning     |          5               |
| afternoon   |          6               |
| evening     |          8             |
```

To satisfy the condition we are joining two datasets categorised by time of day to determine average driver pay and average trip length, respectively. It computes a new statistic, the average earning per mile, after joining. This is accomplished by dividing the average total compensation that a driver receives by the average distance travelled during each time period. The result is a figure that shows the average pay per mile that a driver receives at various times of the day. Rather than focusing only on total profits or trip duration, this computation is essential to determining earning efficiency. The output is a Data Frame with the time of day ranked from most to least profitable per mile, sorted by 'average_earning_per_mile' in descending order.

```
result_df = joined_df.withColumn("average_earning_per_mile", F.col("average_driver_total_pay") / F.col("average_trip_length")) \
            .select("time_of_day", "average_earning_per_mile") \
            .orderBy(F.desc("average_earning_per_mile"))
```

```
2024-04-05 05:19:01,263 INFO scheduler.DAGSchec
+-----------+-----------------------+
|time_of_day|average_earning_per_mile|
+-----------+-----------------------+
|    evening|        4.40992833089496|
|  afternoon|      4.3634308694200215|
|    morning|       3.984544565766397|
|      night|      3.7730081416068373|
+-----------+-----------------------+
```

4. (2 points) What do you find (i.e., insights) from the three results? How do the findings help you make strategies or make decisions?

After looking into the data, I discovered that drivers made the greatest money in the afternoons, maybe because of congested roads and high demand. Unexpectedly, the longest rides are throughout the night. The real deal, though, is that evening rides earn the highest revenue per mile even though they aren't the longest. If drivers want to be as efficient as possible and get more mileage out of every mile they drive, now is the greatest time to work. I could utilise these data as a decision-maker in a ridesharing company to encourage drivers to work late shifts possibly with incentives like higher pay to make sure there are enough cars available when and where people need them most. To maintain supply and demand in a healthy balance, it could also be a sign to modify the fare during specific periods. These trends may provide insight for municipal officials regarding the best locations and times to upgrade traffic or public transport, resulting in a faster and more comfortable journey for all.

Insight of Task 4:

Based on this observation we learn that a ridesharing company's strategic business decisions can be guided by an awareness of the ridership flow throughout the day. Trips during the night are lengthier, possibly more lucrative, and may reveal underutilised regions or periods for transportation. On a per-mile basis, evening hours are especially profitable, suggesting that this is the best time for drivers to earn money. Drivers can optimise their income during these hours and rideshare providers can match supply and demand by using surge pricing or bonuses in the evenings. These results serve as a warning to urban planners to improve public transport where it is most needed and to alleviate transit deserts. In my capacity as a city planner, I would consider this proof of the need to expand night-time transport choices. In general, these patterns teach us important lessons about matching customer behaviour with corporate strategy, providing effective customer service, and understanding how time of day dynamics affect urban transportation networks.

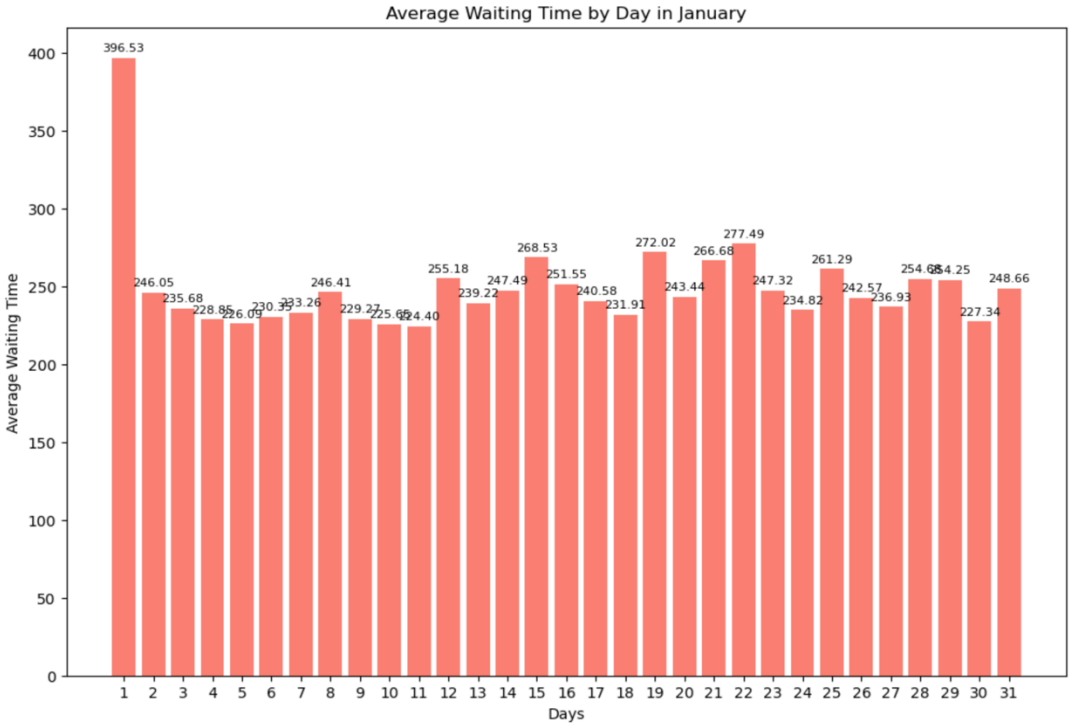
Challenges Faced:

Handling an 'unexpected EOF' problem revealed that it was a server glitch that frequently went away when the oc logs -f task4-spark-app-driver command was run again. Despite being a straightforward solution, the time-consuming nature of the repeating process made it clear that a more efficient solution or automation was required to avoid this step and save important time.

Task 5: Filtering Data

1. (10 points) Extract the data in January and calculate the average waiting time (use the "request_to_pickup" field) over time. You need to sort the output by day. Draw the histogram with 'days' on the x-axis and 'average waiting time' on the y-axis. For example, assume that the 'average waiting time on 'January 10' is '999', '10' will be on the x-axis, indicating the average waiting time value of 999 for day 10 in January.

For this a Spark session is generated to process January's rideshare data from an S3 bucket. A usable date format is created for the 'date' column, and the data is filtered so that rides from January are the only ones included. Next, by grouping the data by day and averaging the 'request_to_pickup' times, an aggregation is carried out to determine the average waiting time each day. The average waiting times are shown in chronological order by sorting the results by day.
The histogram will show variations in waiting times over the course of the month by plotting the days of January on the x-axis and the corresponding average waiting times on the y-axis.



2. (3 points) Which day(s) does the average waiting time exceed 300 seconds?

A filter is applied to the aggregated data in order to find days with average waiting times more than 300 seconds. The screenplay draws attention to these days, perhaps highlighting particular elements such as inclement weather, noteworthy occasions, or operational problems that contributed to longer wait times; however, pinpointing the precise causes would require more information or analysis.

The Spark session is then closed after the results and the days that were found to have larger wait times are both shown and saved back to the S3 bucket in CSV format.
The output, in particular the histogram and the days with larger wait times, offers insightful information about the effectiveness of services and riding experiences. It can assist in identifying operational bottlenecks and provide guidance for customer satisfaction methods, such as assigning extra drivers during peak wait times or resolving delays.

```
+---+------------------+
|day|  avg_waiting_time|
+---+------------------+
|  1| 396.5318744409635|
|  2|246.05148716456986|
|  3|235.68026834234155|
|  4|228.85434668408274|
|  5|226.08877381422872|
|  6|230.35306927438575|
|  7|233.25699185710533|
|  8|246.41358687741243|
|  9|  229.265944341545|
| 10|225.65276195086662|
| 11|224.40468798627612|
| 12|255.17599322195403|
| 13|239.22308233638282|
| 14|247.49345781069232|
| 15| 268.5346481777792|
| 16|251.55102299494047|
| 17| 240.5772885527869|
| 18|231.90770494488552|
| 19|272.02203820618143|
| 20|243.43761253646377|
+---+------------------+
only showing top 20 rows
```

3.(2 points) Why was the average waiting time longer on these day(s) compared to other days?

Determining the reason for the elevated average waiting times on certain days in January would necessitate a comprehensive investigation that goes beyond the current dataset. Such increases are typically caused by several circumstances, such as bad weather that slows down traffic, local events that increase demand, or operational issues with the ridesharing operator. For example, a significant snowstorm may cause fewer cars to be on the road, while an enormous concert may cause a dramatic spike in the number of ride requests compared to the supply. Furthermore, ineffective ridesharing platform or app matching between drivers and passengers may result from technical issues. Particularly in a busy city, rush hour traffic congestion can also greatly lengthen wait times. If there are delays in public transportation, more commuters may use ridesharing services, which could result in longer wait times. These factors highlight the intricate interactions between service capacity and outside events, each of which affects the precarious equilibrium between supply and demand in the ridesharing ecosystem. For rideshare firms to improve their predictive models and make proactive resource allocation adjustments, it is imperative that they comprehend these dynamics.

Insight of Task 5:
There are some days, like the first of the month, when the average wait time noticeably goes beyond the 300-second mark—possibly because of the aftermath of New Year's. These waiting time peaks may be caused by a number of things, such as fewer drivers being available, a rise in the demand for services, or slower traffic because of the holidays or the winter weather. These variations can be easily seen through the use of a histogram, which provides a daily overview of service effectiveness. The longer wait times highlight possible service bottlenecks where supply might not be able to keep up with demand. Ridesharing businesses may be prompted by these insights to implement more accurate capacity planning and demand forecasting, especially during recognised peak times. Drivers may be able to optimise their work schedules and increase their earnings by being aware of these tendencies.

All things considered, these results are vital for ridesharing services to improve user happiness and operational efficiency, particularly on days when service is prone to delays.

Challenges Faced:

When attempting to create the CSV file, I encountered syntax errors and runtime issues due to unexpected data types. To tackle these, I meticulously reviewed the code for syntax accuracy and introduced type checking . Additionally, I incorporated exception handling to capture and log errors for any unanticipated issues during runtime, ensuring smoother execution and successful CSV creation.

Task 6: Filtering Data

1. (5 points) Find trip counts greater than 0 and less than 1000 for different 'Pickup_Borough' at different 'time_of_day'. You need to provide a screenshot of this question in your report. The columns should include, Pickup_Borough, time_of_day, and trip_count. For example,

Pickup_Borough	time_of_day	trip_count
EWR	afternoon	8
EWR	morning	8
...	...	...

To highlight the frequency of trips by borough and time of day, this Spark SQL query sorts over ridesharing data, concentrating on trip counts ranging from one to a thousand. By grouping visits together and counting them within these constraints, the system separates moderate activity levels from excessive data points. Operational insights are revealed by this method, which could direct resource allocation. Examples of these insights include possible growth regions during particular periods that aren't too busy or too calm. When the results are arranged according to time and borough, it can be easier for ridesharing businesses to match supply and demand, drivers can choose the best times to work, and city planners can learn about moderate travel patterns that could use more transportation options. Overall, this analysis is about pinpointing the sweet spots in service utilization for strategic enhancements.

```
trip_counts_by_borough_and_time_of_day = spark.sql("""
        SELECT
        Pickup_Borough,
        time_of_day,
        COUNT(*) AS trip_count
    FROM joined_df
    GROUP BY Pickup_Borough, time_of_day
    HAVING trip_count > 0 AND trip_count < 1000
    ORDER BY Pickup_Borough, time_of_day
  """)
  trip_counts_by_borough_and_time_of_day.show(truncate=False)
```

```
2024-04-05 01:44:02,302 INFO codegen.CodeGene
+--------------+-----------+----------+
|Pickup_Borough|time_of_day|trip_count|
+--------------+-----------+----------+
|EWR           |afternoon  |2         |
|EWR           |morning    |5         |
|EWR           |night      |3         |
|Unknown       |afternoon  |908       |
|Unknown       |evening    |488       |
|Unknown       |morning    |892       |
|Unknown       |night      |792       |
+--------------+-----------+----------+
```

2. (5 points) Calculate the number of trips for each 'Pickup_Borough' in the evening time **(i.e., time_of_day field). You need to provide a screenshot of this question in your report. The columns should include Pickup_Borough, time_of_day, trip_count. For example,**

Pickup_Borough	time_of_day	trip_count
EWR	evening	23333
Unknown	evening	2222
...	...	...

We are intended to calculate the total number of ridesharing trips that took place during the evening in each 'Pickup_Borough'. It filters the dataset for values in the 'time_of_day' column that are designated as 'evening' using a SQL query. The COUNT (*) function is then used to count the number of trips in the filtered data for each borough; this count is aliased as "trip_count," and the results are grouped by "Pickup_Borough." An understandable summary of trip frequency by borough in the evening is produced by sorting this ordered data by borough name.

```
# Task2: Calculate the number of trips for each 'Pickup_Borough' in the evening time
    trip_counts_by_pickup_borough_evening = spark.sql("""
        SELECT Pickup_Borough, 'evening' AS time_of_day, COUNT(*) AS trip_count
        FROM joined_df
        WHERE time_of_day = 'evening'
        GROUP BY Pickup_Borough
        ORDER BY Pickup_Borough
    """)
    trip_counts_by_pickup_borough_evening.show(truncate=False)
```

```
+--------------+-----------+----------+
|Pickup_Borough|time_of_day|trip_count|
+--------------+-----------+----------+
|Bronx         |evening    |1380355   |
|Brooklyn      |evening    |3075616   |
|Manhattan     |evening    |5724796   |
|Queens        |evening    |2223003   |
|Staten Island |evening    |151276    |
|Unknown       |evening    |488       |
+--------------+-----------+----------+

2024-04-05 02:15:29,461 INFO datasources.F
```

3. (5 points) Calculate the number of trips that started in Brooklyn (Pickup_Borough field) and ended in Staten Island (Dropoff_Borough field). Show 10 samples in the terminal. You need to

provide a screenshot of this question (the 10 samples) and the number of trips in your report. The columns should include, Pickup_Borough, Dropoff_Borough, and Pickup_Zone. do not truncate the name of Pickup_Zone**. For example,**

Pickup_Borough	Pickup_Borough	Pickup_Zone
Brooklyn	Staten Island	Bay Ridge
...	...	...

We are using Apache Spark's power to run an exact query on a DataFrame. Finding and showcasing a sample of journeys that started in Brooklyn and concluded in Staten Island is the goal. Here the filter function is crucial since it reduces the dataset to only those entries in which "Brooklyn" appears in the 'Pickup_Borough' column and "Staten Island" appears in the 'Dropoff_Borough' column. We're left with just three columns of interest after applying this filter: "Pickup_Borough," "Dropoff_Borough," and "Pickup_Zone." We've made sure the zones are completely visible by setting truncate=False.

```
# Task3: Calculate the number of trips that started in Brooklyn and ended in Staten Island
   task3_result = joined_df.filter((col("Pickup_Borough") == "Brooklyn") & (col("Dropoff_Borough") == "Staten Island")) \
       .select("Pickup_Borough", "Dropoff_Borough", "Pickup_Zone") \
       .limit(10)
   task3_result.show(truncate=False)
```

```
2024-04-05 02:15:52,404 INFO codegen.CodeGenerator: Code generated in 10.439686 ms
+-------------+---------------+------------------------+
|Pickup_Borough|Dropoff_Borough|Pickup_Zone             |
+-------------+---------------+------------------------+
|Brooklyn      |Staten Island  |DUMBO/Vinegar Hill      |
|Brooklyn      |Staten Island  |Dyker Heights           |
|Brooklyn      |Staten Island  |Bensonhurst East        |
|Brooklyn      |Staten Island  |Williamsburg (South Side)|
|Brooklyn      |Staten Island  |Bay Ridge               |
|Brooklyn      |Staten Island  |Bay Ridge               |
|Brooklyn      |Staten Island  |Flatbush/Ditmas Park    |
|Brooklyn      |Staten Island  |Bay Ridge               |
|Brooklyn      |Staten Island  |Bath Beach              |
|Brooklyn      |Staten Island  |Bay Ridge               |
+-------------+---------------+------------------------+

2024-04-05 02:15:52,552 INFO datasources.FileSourceStrategy: Pruning directories wit
```

```
2024-04-05 02:40:52,602 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 16:
2024-04-05 02:40:52,602 INFO scheduler.DAGScheduler: Job 13 finished: count at NativeMethodAcces
Total number of trips from Brooklyn to Staten Island: 69437
2024-04-05 02:40:52,623 INFO server.AbstractConnector: Stopped Spark@f844493{HTTP/1.1,[http/1.1]
2024-04-05 02:40:52,625 INFO ui.SparkUI: Stopped Spark web UI at http://task6-spark-app-587a2c8e
```

Insight of Task 6:

By examining these results, we can see exactly how ridesharing demand fluctuates. It's clear that certain times of the day and specific routes see less traffic than others, indicating potential

for expansion and improved services. Demand for ridesharing peaks in the evenings, suggesting that drivers should be assigned strategically. The specifics of the Brooklyn to Staten Island route highlight the necessity of focused service enhancements as well as the significance of comprehending borough-specific travel requirements. All these insights together provide a strategic compass that rideshare businesses can use to streamline operations, drivers may use to increase earnings, and city planners can use to improve urban mobility.
The data's trends point to a more complex knowledge of riders' behaviours, one that may potentially grow in underutilised times and locations. These results are crucial for rideshare companies to adjust their schedules so that drivers are available when demand spikes, particularly at night. This serves as a guide for drivers to match their working hours with the highest earning periods. More broadly, this data can be used by local officials to better customise public transportation timetables and infrastructure upgrades to cover the areas where ridesharing services are most in demand. All together, these observations represent actionable intelligence that can be used as a roadmap to improve service delivery and satisfy the city's ever-changing transit requirements.

Challenges Faced:

While tackling the difficulties, I ran into coding issues that required painstaking debugging to fix logical or syntactic errors. These mistakes need to be corrected with extreme caution because they are frequently inherent in the development process. Furthermore, run-time errors were detected, which were probably caused by data or environmental problems. This meant that a careful examination of the data being processed and the code execution environment was required. In order to get beyond these obstacles, one needed to have a thorough understanding of the codebase in addition to the patience and perseverance to keep testing until everything worked smoothly.

Task7: Routes Analysis

1. (15 points) You need to analyse the 'Pickup_Zone' to 'Dropoff_Zone' routes to find the top 10 popular routes in terms of the trip count. Each total count of a route should include the trip counts for each unique route from Uber and Lyft (i.e., the sum of Uber and Lyft counts on the same route). Then you can get the top 10 in total count (e.g., sorting the result by total counts and displaying the top 10 routes or any other way). You may need to create a new column called Route which concats column Pickup_Zone with 'to' with column Dropoff_Zone. You need to give a screenshot of your result in your report, do not truncate the name of routes. The result table should include 'route', 'uber_count', 'lyft_count', and 'total_count'. For example,

Route	uber_count	lyft_count	total_count
JFK Airport to NA	253213	45	253258
Astoria to Jamaica	253212	12	253224
.....	.....	.....	.....

We created a Spark session specifically for analysing a dataset that is kept in an S3 bucket. Read in two CSV files for this session: one is ridesharing data, and the other is a query for taxi zones and made a new column in rideshare Data Frame named "Route" that details each trip's itinerary from the pickup to the dropoff locations. Then combined the data, breaking out the number of trips by rideshare service provider, for every distinct route.Going one step further, created a summary DataFrame that pivots on the service providers, providing me with distinct columns for the counts of Uber and Lyft. Then aggregated these counts into a single column called "total_count." Which helped us ensure an accurate count by using this clever manoeuvre to merge any null values to zero. With everything set up, we

concentrated on sorting the data so that we could see the most popular routes—the top 10 routes by total number of trips. By presenting this short, complete list of routes from Uber and Lyft along with their individual counts and the total, thus able to get an idea of the most frequently travelled routes.

```
    rideshare_d_p = "s3a://" + s3_data_repository_bucket + "/ECS765/rideshare_2023/rideshare_data.csv"
    rideshare_df = spark.read.option("header", "true").csv(rideshare_d_p)

    taxi_zone_lookup_pt = "s3a://" + s3_data_repository_bucket +
"/ECS765/rideshare_2023/taxi_zone_lookup.csv"
    taxi_zone_lookup_df = spark.read.option("header", "true").csv(taxi_zone_lookup_pt)

    rideshare_df = spark.read.option("header", "true").csv(rideshare_data_pt)
    rideshare_df = rideshare_df.withColumn("Route", F.concat_ws(" to ", "pickup_location",
"dropoff_location"))

    route_counts_df = rideshare_df.groupBy("Route", "business").count()

    route_summary_df = route_counts_df.groupBy("Route").pivot("business").agg(F.first("count"))
    route_summary_df = route_summary_df.withColumn("total_count", F.coalesce(F.col("Uber"), F.lit(0)) +
F.coalesce(F.col("Lyft"), F.lit(0)))

    top_routes_df = route_summary_df.orderBy(F.desc("total_count")).limit(10)
    top_routes_df.show(truncate=False)
```

```
2024-04-04 21:42:19,571 INFO scheduler.DAG
2024-04-04 21:42:19,597 INFO codegen.CodeG
+-----------+----+------+-----------+
|Route      |Lyft|Uber  |total_count|
+-----------+----+------+-----------+
|132 to 265|46  |253211|253257     |
|76 to 76  |184 |202719|202903     |
|26 to 26  |78  |155803|155881     |
|138 to 265|41  |151521|151562     |
|39 to 39  |26  |126253|126279     |
|216 to 132|1770|107392|109162     |
|61 to 61  |100 |98591 |98691      |
|14 to 14  |300 |98274 |98574      |
|7 to 7    |75  |90692 |90767      |
|129 to 129|19  |89652 |89671      |
+-----------+----+------+-----------+
```

Insight of Task 7:

Running this code, I'm discerning the travel trends and demands across various rideshare routes, which could be vital for optimizing service distribution. It reveals the busiest routes for Uber and Lyft, suggesting where to focus resources or where the market might be saturated. The comparative data between the two services can also signal where one may have a competitive edge. This understanding is key to strategic decisions like dynamic pricing or targeted marketing. Moreover, the exercise is sharpening my data analysis skills, particularly in data aggregation and interpretation, leveraging Spark's robust capabilities for real-world applications.

Challenges Faced:

When running the Spark job to analyse rideshare patterns, I encountered an unexpected server shutdown error. I promptly reached out to our management team, detailing the issue, and requesting their intervention to resolve it. While waiting for a fix, I tackled other challenges as well: ensuring data consistency across multiple sources proved complex, requiring careful normalization. Also, I had to optimize my Spark queries to manage memory usage more efficiently, preventing any further disruptions.