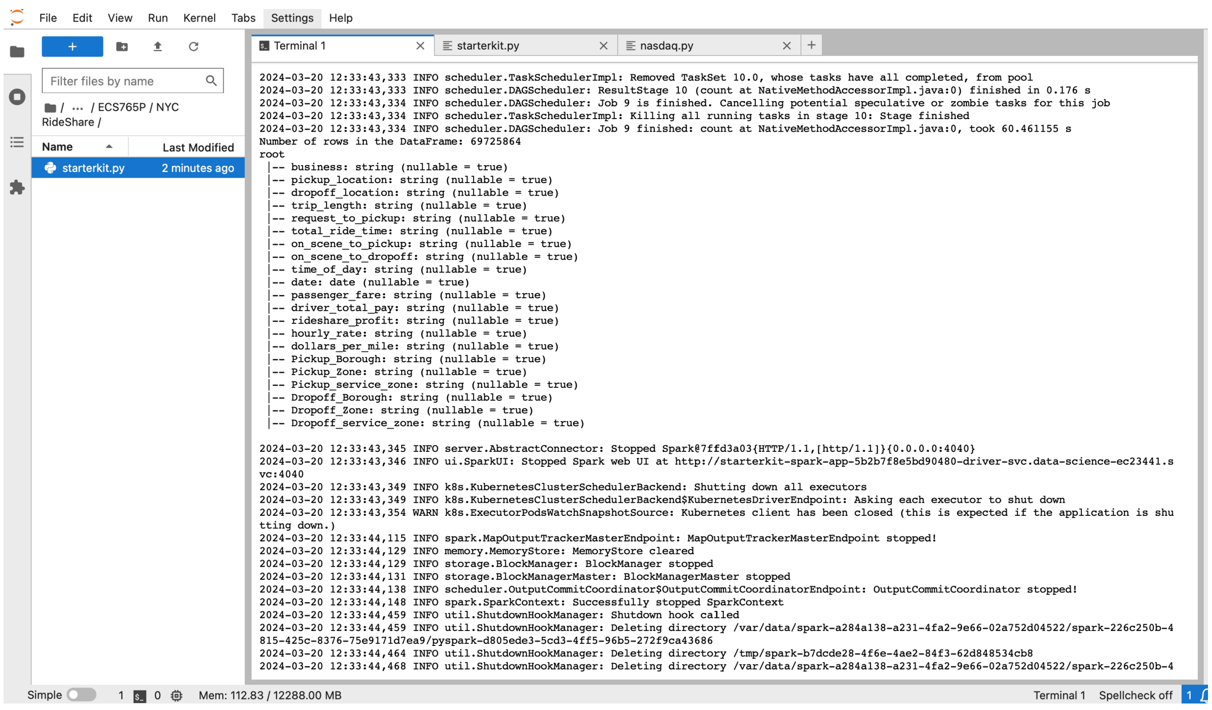
Name – Samrudhi Sunil Bhosale
Student ID – 230854907

TASK 1:

Explanation:

1. In this task, first we load the rideshare data and taxi zone data using **load()** function.
2. The rideshare data contains information about rides, like pickup and dropoff locations. The taxi zone data provides additional details about these locations such as borough and zone information.
3. The objective of this task is to add the rideshare data with additional location details.
4. I have merged rideshare data with taxi zone data using rideshare data's **pickup_location** and taxi_zone data's **LocationID**.
5. Similarly, for the dropoff location I have used rideshare data's **dropoff_location** with taxi zone data's **LocationID**.

Result:



```
2024-03-20 12:33:43,333 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 10.0, whose tasks have all completed, from pool
2024-03-20 12:33:43,333 INFO scheduler.DAGScheduler: ResultStage 10 (count at NativeMethodAccessorImpl.java:0) finished in 0.176 s
2024-03-20 12:33:43,334 INFO scheduler.DAGScheduler: Job 9 is finished. Cancelling potential speculative or zombie tasks for this job
2024-03-20 12:33:43,334 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 10: Stage finished
2024-03-20 12:33:43,334 INFO scheduler.DAGScheduler: Job 9 finished: count at NativeMethodAccessorImpl.java:0, took 60.461155 s
Number of rows in the DataFrame: 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: date (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)

2024-03-20 12:33:43,345 INFO server.AbstractConnector: Stopped Spark@7ffd3a03{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2024-03-20 12:33:43,346 INFO ui.SparkUI: Stopped Spark web UI at http://starterkit-spark-app-5b2b7f8e5bd90480-driver-svc.data-science-ec23441.s
vc:4040
2024-03-20 12:33:43,349 INFO k8s.KubernetesClusterSchedulerBackend: Shutting down all executors
2024-03-20 12:33:43,349 INFO k8s.KubernetesClusterSchedulerBackend$KubernetesDriverEndpoint: Asking each executor to shut down
2024-03-20 12:33:43,354 WARN k8s.ExecutorPodsWatchSnapshotSource: Kubernetes client has been closed (this is expected if the application is shu
tting down.)
2024-03-20 12:33:44,115 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
2024-03-20 12:33:44,129 INFO memory.MemoryStore: MemoryStore cleared
2024-03-20 12:33:44,129 INFO storage.BlockManager: BlockManager stopped
2024-03-20 12:33:44,131 INFO storage.BlockManagerMaster: BlockManagerMaster stopped
2024-03-20 12:33:44,138 INFO scheduler.OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
2024-03-20 12:33:44,148 INFO spark.SparkContext: Successfully stopped SparkContext
2024-03-20 12:33:44,459 INFO util.ShutdownHookManager: Shutdown hook called
2024-03-20 12:33:44,459 INFO util.ShutdownHookManager: Deleting directory /var/data/spark-a284a138-a231-4fa2-9e66-02a752d04522/spark-226c250b-4
815-425c-8376-75e9171d7ea9/pyspark-d805ede3-5cd3-4ff5-96b5-272f9ca43686
2024-03-20 12:33:44,464 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-b7dcde28-4f6e-4ae2-84f3-62d848534cb8
2024-03-20 12:33:44,468 INFO util.ShutdownHookManager: Deleting directory /var/data/spark-a284a138-a231-4fa2-9e66-02a752d04522/spark-226c250b-4
```

Challenges Faced:

1. One challenge was to correctly identify the type of join for the rideshare data with the taxi zone data based on location IDs. As we required all the data from rideshare and only the common data between rideshare and taxi zone.

2. Resolution- I have used Spark's join function with appropriate parameters to perform the **left join** between rideshare data and taxi zone data.
3. After joining the datasets, there was a need to rename columns to avoid conflicts.
4. Resolution- I have used spark's **alias()** function to rename the columns properly.
5. The rideshare data contained dates as timestamps, which were to be converted to a more readable format.
6. Resolution- I have used used Spark's **to_date()** and **from_unixtime()** functions to convert the timestamp column to a date format.

Knowledge Acquired:

1. Understanding of data joining techniques in Spark.
2. Data type conversion for improved data handling and analysis.

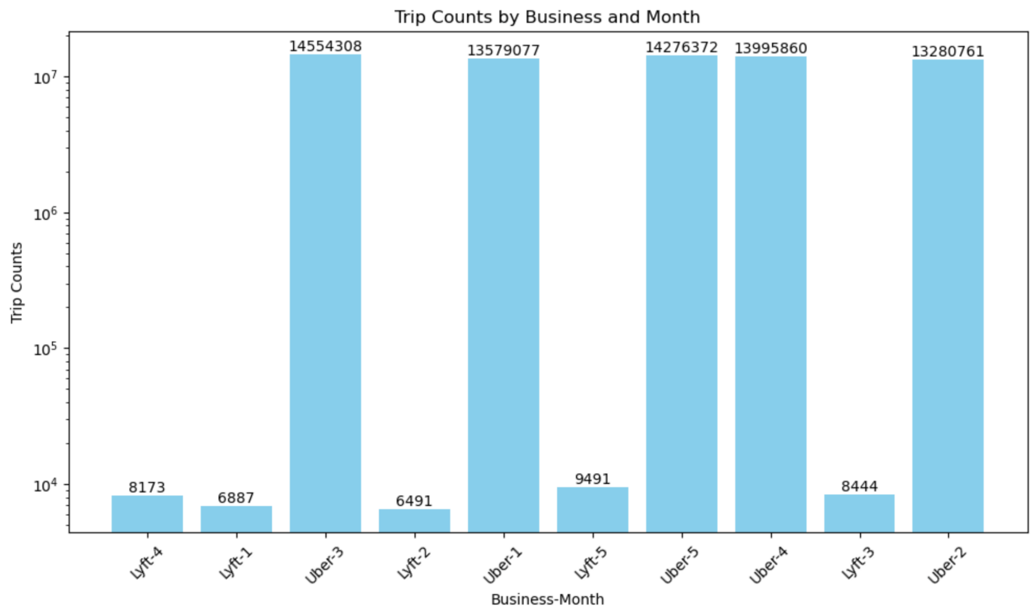
TASK 2:

Explanation:

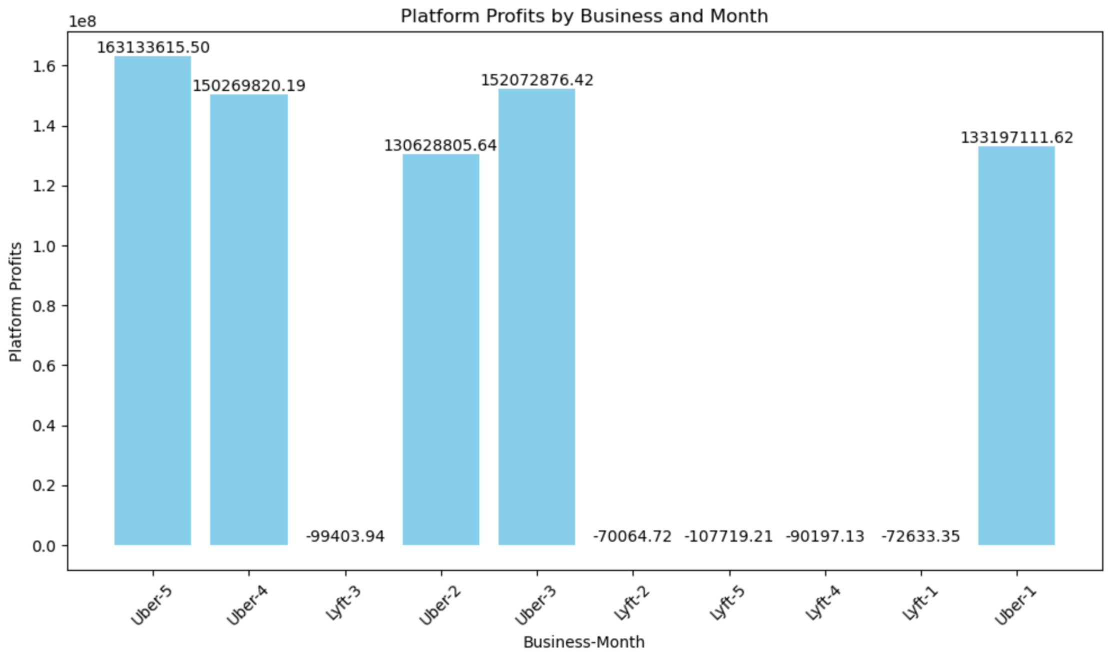
1. This task involves conducting monthly analysis on rideshare data to get insights into trip counts, platform profits, and drivers' earnings.
2. The trip counts were calculated by aggregating data based on business, month col and using **count()** function.
3. To calculate the platform profits, first I had to convert rideshare_profit into float and then aggregate business, month col and taking sum of rideshare_profit using **sum()** function.
4. Similarly, to get the drivers earnings I have converted driver_total_pay into float and then using **sum()** function calculated the total driver's earnings for each business in each month.

Result:

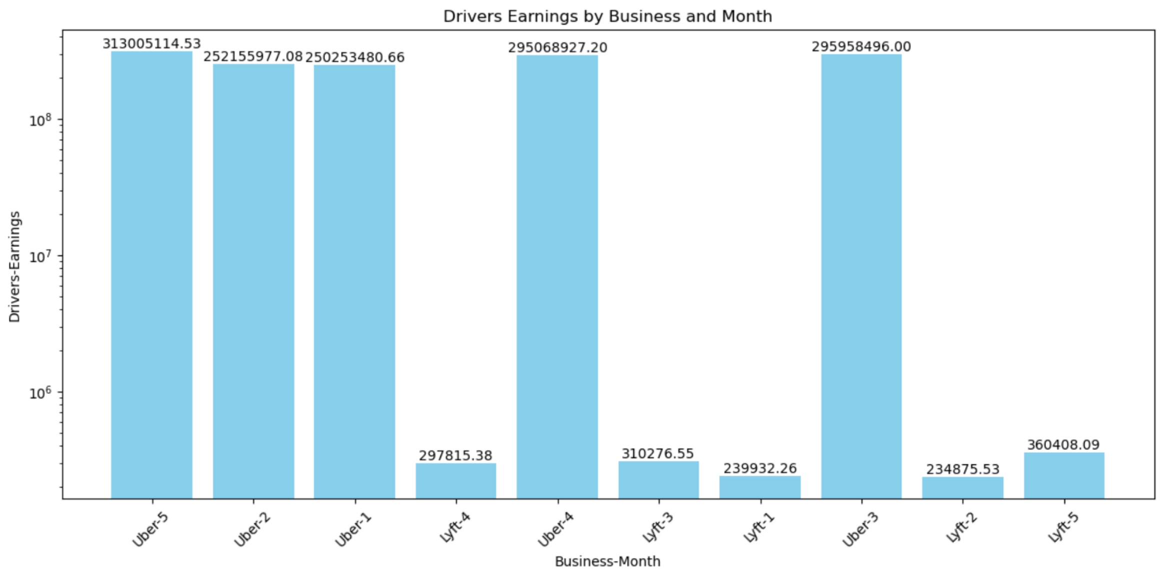
Part 1:



Part 2:



Part 3:



Challenges Faced:

1. Aggregating data by both business and month col to get insights of the data.
2. Resolution- I have used Spark's **groupBy()** function with appropriate columns **(groupBy("business", "month"))** to aggregate the data before performing calculations.
3. Converting columns to required data types for numerical calculations was one of the challenge.
4. Resolution- I have utilized Spark's **withColumn()** function along with **cast()** to convert columns like rideshare_profit and driver_total_pay to float to perform summation.

5. Writing the aggregated results to CSV files while ensuring proper formatting was a challenge.
6. Resolution- I have used Spark's **write()** function with suitable options (**option("header", "true"), mode("overwrite"))** to write the data to CSV files stored in the specified S3 bucket. Also, repartitioning the dataFrame to a single partition **(repartition(1))** helped in generating a single CSV file.

Knowledge Acquired:

1. Understanding of data aggregation techniques in Spark, to generate insights from a huge dataset.
2. Proficiency in handling data type conversions for numerical calculations and analysis.
3. Familiarity with writing aggregated data to external storage systems, such as S3.

Insights:

For Drivers-

1. Uber seems to have a higher number of trips, which indicate more opportunities for drivers to get passengers and therefore earn more income.
2. Higher earnings on Uber could attract more drivers to the platform.
3. The data could give Lyft drivers a stronger position to negotiate better terms with Lyft for its further development.

For CEO-

1. Investigate why customers prefer one service over the other. It could be due to service quality, price or availability.
2. For Lyft, strategies might include marketing adv, promotions, or discounts. For Uber, it might be about expansion into new markets.
3. Lyft may need to innovate in pricing, quality of service, or driver benefits to differentiate itself from Uber.

TASK 3:

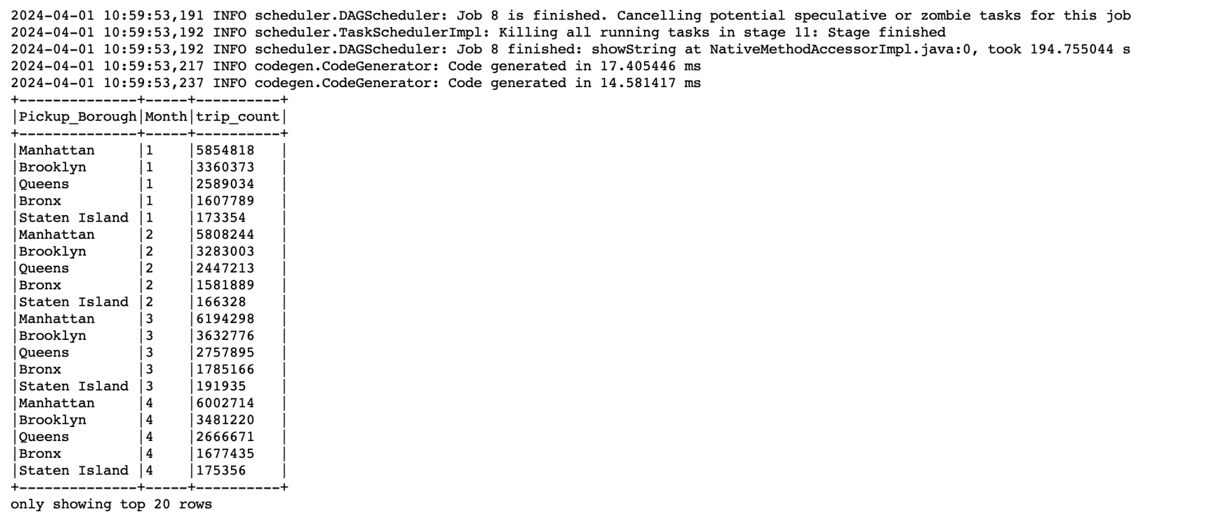
Explanation:

1. The task aims to analyze ride-sharing data to find out which boroughs are most popular for pickups and drop-offs for each month.
2. A Spark SQL query is executed to find the top 5 pickup boroughs for each month. The query selects the Pickup_Borough, month, and trip_count col from the pickup_data dataframe. It calculates the trip count for each pickup borough within each month using the **COUNT(*)** function and ranks them using the **ROW_NUMBER()** window function.
3. Similarly, spark SQL query is executed to find the top 5 dropoff boroughs for each month. Similar to the previous query, it selects the Dropoff_Borough, month, and trip_count columns from the dropoff_data dataframe.
4. Another objective of this task is to find top 30 most profitable routes.
5. For this a spark SQL query is executed to identify the top 30 routes based on the total profit earned. The query selects the Pickup_Borough and Dropoff_Borough

col and concatenated to form Route, and calculates the total profit earned by **sum(driver_total_pay)** for each route. The results are then ordered by total profit in descending order and limit of 30 is applied.

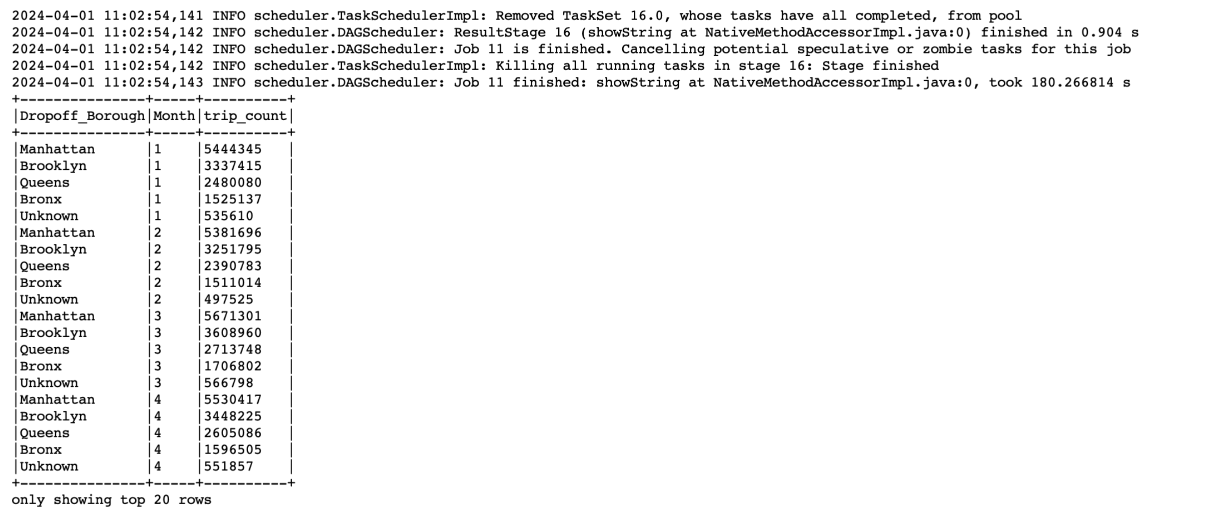
Result:

Part 1:



```
2024-04-01 10:59:53,191 INFO scheduler.DAGScheduler: Job 8 is finished. Cancelling potential speculative or zombie tasks for this job
2024-04-01 10:59:53,192 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 11: Stage finished
2024-04-01 10:59:53,192 INFO scheduler.DAGScheduler: Job 8 finished: showString at NativeMethodAccessorImpl.java:0, took 194.755044 s
2024-04-01 10:59:53,217 INFO codegen.CodeGenerator: Code generated in 17.405446 ms
2024-04-01 10:59:53,237 INFO codegen.CodeGenerator: Code generated in 14.581417 ms
+--------------+-----+----------+
|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    |
+--------------+-----+----------+
only showing top 20 rows
```

Part 2:



```
2024-04-01 11:02:54,141 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 16.0, whose tasks have all completed, from pool
2024-04-01 11:02:54,142 INFO scheduler.DAGScheduler: ResultStage 16 (showString at NativeMethodAccessorImpl.java:0) finished in 0.904 s
2024-04-01 11:02:54,142 INFO scheduler.DAGScheduler: Job 11 is finished. Cancelling potential speculative or zombie tasks for this job
2024-04-01 11:02:54,142 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 16: Stage finished
2024-04-01 11:02:54,143 INFO scheduler.DAGScheduler: Job 11 finished: showString at NativeMethodAccessorImpl.java:0, took 180.266814 s
+---------------+-----+----------+
|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    |
+---------------+-----+----------+
only showing top 20 rows
```

Part 3:

```
2024-04-01 11:11:50,471 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 20.0, whose tasks have all completed, from pool
2024-04-01 11:11:50,472 INFO scheduler.DAGScheduler: ResultStage 20 (showString at NativeMethodAccessorImpl.java:0) finished in 0.065 s
2024-04-01 11:11:50,473 INFO scheduler.DAGScheduler: Job 13 is finished. Cancelling potential speculative or zombie tasks for this job
2024-04-01 11:11:50,473 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 20: Stage finished
2024-04-01 11:11:50,473 INFO scheduler.DAGScheduler: Job 13 finished: showString at NativeMethodAccessorImpl.java:0, took 535.919033 s
2024-04-01 11:11:50,503 INFO codegen.CodeGenerator: Code generated in 22.948474 ms
+------------------------------+--------------------+
|Route                         |total_profit        |
+------------------------------+--------------------+
|Manhattan to Manhattan        |3.338577255500229E8 |
|Brooklyn to Brooklyn          |1.7394472147999212E8|
|Queens to Queens              |1.1470684719998907E8|
|Manhattan to Queens           |1.0173842820999995E8|
|Queens to Manhattan           |8.603540026000004E7 |
|Manhattan to Unknown          |8.010710241999993E7 |
|Bronx to Bronx                |7.414622575999324E7 |
|Manhattan to Brooklyn         |6.799047559E7       |
|Brooklyn to Manhattan         |6.317616104999998E7 |
|Brooklyn to Queens            |5.045416243000008E7 |
|Queens to Brooklyn            |4.729286536000015E7 |
|Queens to Unknown             |4.629299990000003E7 |
|Bronx to Manhattan            |3.2486325170000087E7|
|Manhattan to Bronx            |3.197876345000006E7 |
|Manhattan to EWR              |2.375088861999999E7 |
|Brooklyn to Unknown           |1.084882757E7       |
|Bronx to Unknown              |1.0464800209999997E7|
|Bronx to Queens               |1.0292266499999998E7|
|Queens to Bronx               |1.0182898729999999E7|
|Staten Island to Staten Island|9686862.450000014   |
+------------------------------+--------------------+
only showing top 20 rows
```

Challenges Faced:

1. The main challenge was understand the need to create temporary views for each dataframe and check whether they are correctly registered for Spark SQL queries.
2. Resolution: I have used the **createOrReplaceTempView()** function to create temporary views for each dataframe.
3. Another challenge was to correctly rank the top pickup and dropoff boroughs for each month.
4. Resolution- This is achieved by using the **ROW_NUMBER()** window function with appropriate partitioning and ordering within the Spark SQL queries.
5. Concatenating pickup and dropoff boroughs to form routes for the top 30 routes posed a challenge due to string manipulation.
6. Resolution- Used **CONCAT()** function within the Spark SQL query to concatenate the boroughs into routes.

Knowledge Acquired:

1. Understood the importance of creating temporary views for dataframes to enable Spark SQL operations.
2. Learned how to utilize window functions like ROW_NUMBER() for ranking data.

Insights:

For Driver -

1. Trip counts are highest in Manhattan, which indicates that a driver will find more customers and potentially make more money working in this borough.
2. Depending on demand and trip counts in each borough, a driver could manage its schedule to get higher trip counts.
3. Based on the most profitable routes, drivers can position themselves in certain areas to increase the chance of picking up higher-paying fares.

For CEO –

1. For Uber, add more cars in high-demand areas (like Manhattan) to increase the revenue and maintain market dominance.

2. Despite lower numbers, Lyft has the opportunity to capture market share from Uber by targeting less-served boroughs or improving services.

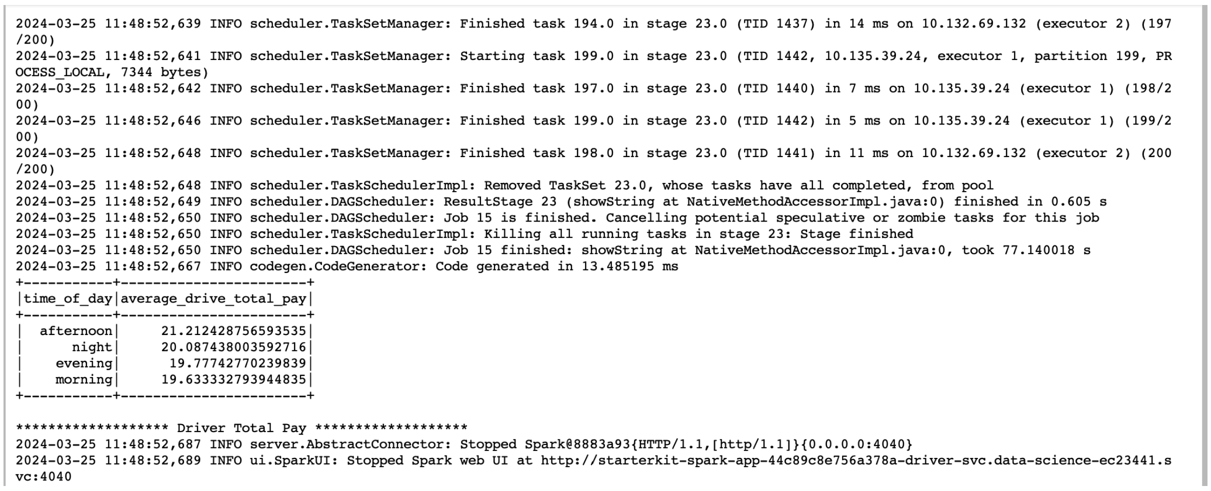
TASK 4:

Explanation:

1. In this task we calculate the average driver total pay for each time of day . The query selects the time_of_day column and calculates the average driver total pay using the **AVG()** aggregation function. The results are grouped by time_of_day and ordered by average_drive_total_pay in descending order.
2. In the next part of the task we find the average trip length for each time of day .
3. First the trip_length column in the rideshare_with_month dataframe was converted to float data type using the **withColumn**() and **cast()** function.
4. Spark SQL query was executed to calculate the average trip length for each time of day. Similar to the previous query, it selects the time_of_day column and calculates the average trip length using the **AVG()** aggregation function. The results were grouped by time_of_day and ordered by average_trip_length in descending order.
5. The avg_driver_total_pay and average_trip_length dataframes are joined on the time_of_day column to calculate the average earning per mile. The result is obtained by dividing the average driver total pay by average trip length. The orderBy() function is used to sort the results by average_earning_per_mile

Result:

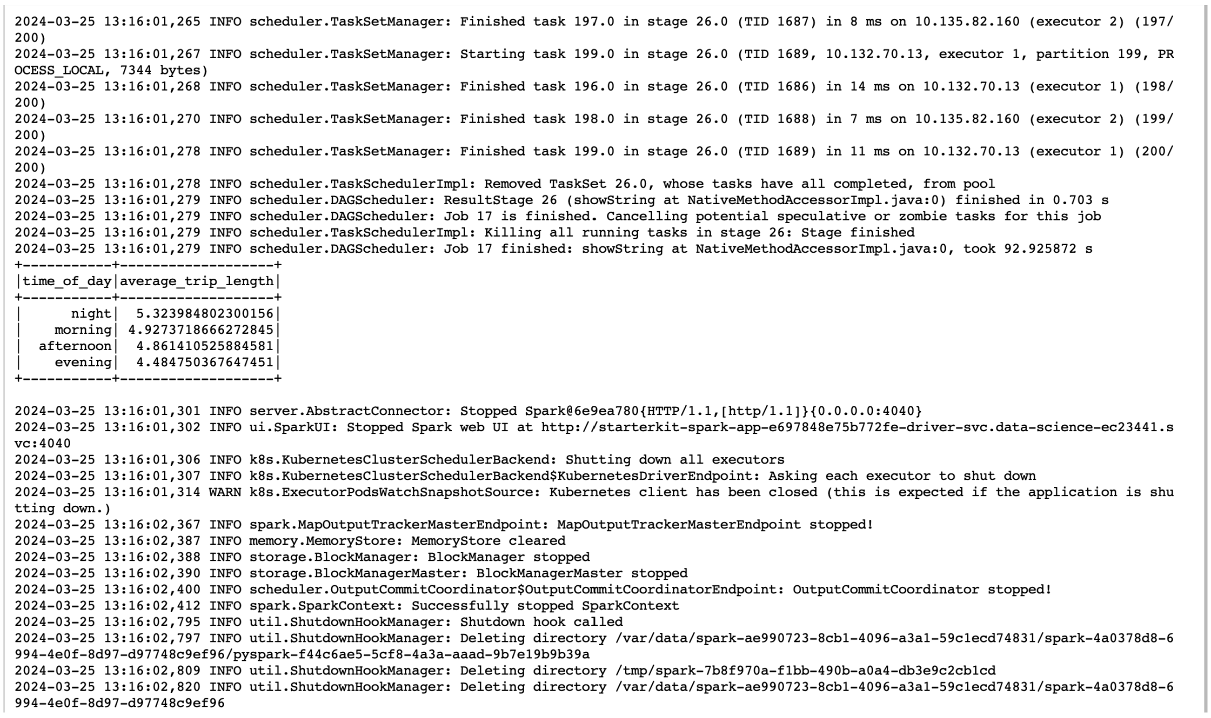
Part 1:



```
2024-03-25 11:48:52,639 INFO scheduler.TaskSetManager: Finished task 194.0 in stage 23.0 (TID 1437) in 14 ms on 10.132.69.132 (executor 2) (197
/200)
2024-03-25 11:48:52,641 INFO scheduler.TaskSetManager: Starting task 199.0 in stage 23.0 (TID 1442, 10.135.39.24, executor 1, partition 199, PR
OCESS_LOCAL, 7344 bytes)
2024-03-25 11:48:52,642 INFO scheduler.TaskSetManager: Finished task 197.0 in stage 23.0 (TID 1440) in 7 ms on 10.135.39.24 (executor 1) (198/2
00)
2024-03-25 11:48:52,646 INFO scheduler.TaskSetManager: Finished task 199.0 in stage 23.0 (TID 1442) in 5 ms on 10.135.39.24 (executor 1) (199/2
00)
2024-03-25 11:48:52,648 INFO scheduler.TaskSetManager: Finished task 198.0 in stage 23.0 (TID 1441) in 11 ms on 10.132.69.132 (executor 2) (200
/200)
2024-03-25 11:48:52,648 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 23.0, whose tasks have all completed, from pool
2024-03-25 11:48:52,649 INFO scheduler.DAGScheduler: ResultStage 23 (showString at NativeMethodAccessorImpl.java:0) finished in 0.605 s
2024-03-25 11:48:52,650 INFO scheduler.DAGScheduler: Job 15 is finished. Cancelling potential speculative or zombie tasks for this job
2024-03-25 11:48:52,650 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 23: Stage finished
2024-03-25 11:48:52,650 INFO scheduler.DAGScheduler: Job 15 finished: showString at NativeMethodAccessorImpl.java:0, took 77.140018 s
2024-03-25 11:48:52,667 INFO codegen.CodeGenerator: Code generated in 13.485195 ms
+-----------+----------------------+
|time_of_day|average_drive_total_pay|
+-----------+----------------------+
|  afternoon|     21.212428756593535|
|      night|     20.087438003592716|
|    evening|      19.77742770239839|
|    morning|     19.633332793944835|
+-----------+----------------------+

****************** Driver Total Pay ******************
2024-03-25 11:48:52,687 INFO server.AbstractConnector: Stopped Spark@8883a93{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2024-03-25 11:48:52,689 INFO ui.SparkUI: Stopped Spark web UI at http://starterkit-spark-app-44c89c8e756a378a-driver-svc.data-science-ec23441.s
vc:4040
```

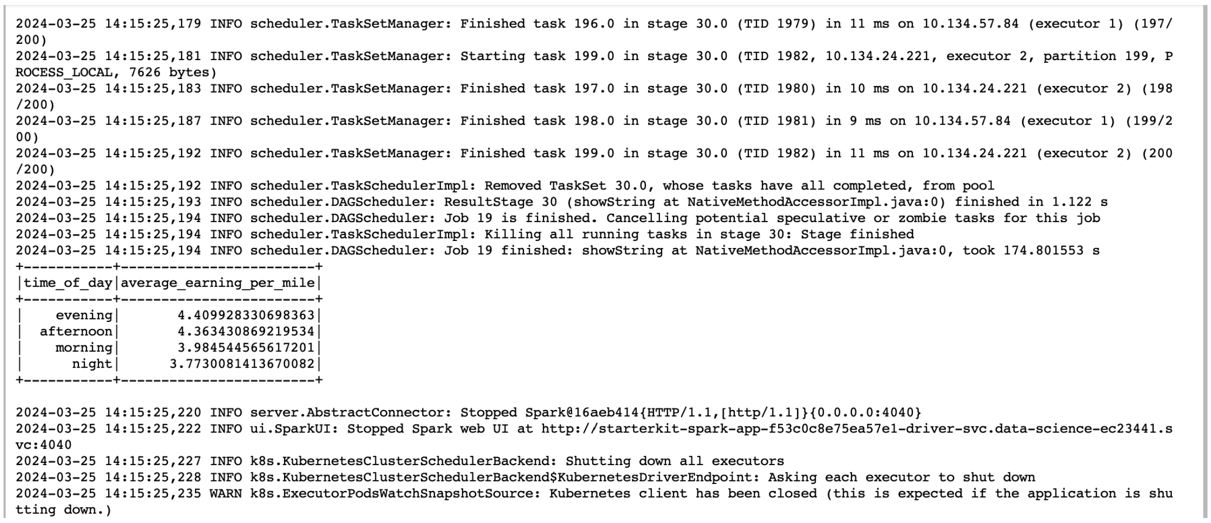
Part 2:



```
2024-03-25 13:16:01,265 INFO scheduler.TaskSetManager: Finished task 197.0 in stage 26.0 (TID 1687) in 8 ms on 10.135.82.160 (executor 2) (197/
200)
2024-03-25 13:16:01,267 INFO scheduler.TaskSetManager: Starting task 199.0 in stage 26.0 (TID 1689, 10.132.70.13, executor 1, partition 199, PR
OCESS_LOCAL, 7344 bytes)
2024-03-25 13:16:01,268 INFO scheduler.TaskSetManager: Finished task 196.0 in stage 26.0 (TID 1686) in 14 ms on 10.132.70.13 (executor 1) (198/
200)
2024-03-25 13:16:01,270 INFO scheduler.TaskSetManager: Finished task 198.0 in stage 26.0 (TID 1688) in 7 ms on 10.135.82.160 (executor 2) (199/
200)
2024-03-25 13:16:01,278 INFO scheduler.TaskSetManager: Finished task 199.0 in stage 26.0 (TID 1689) in 11 ms on 10.132.70.13 (executor 1) (200/
200)
2024-03-25 13:16:01,278 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 26.0, whose tasks have all completed, from pool
2024-03-25 13:16:01,279 INFO scheduler.DAGScheduler: ResultStage 26 (showString at NativeMethodAccessorImpl.java:0) finished in 0.703 s
2024-03-25 13:16:01,279 INFO scheduler.DAGScheduler: Job 17 is finished. Cancelling potential speculative or zombie tasks for this job
2024-03-25 13:16:01,279 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 26: Stage finished
2024-03-25 13:16:01,279 INFO scheduler.DAGScheduler: Job 17 finished: showString at NativeMethodAccessorImpl.java:0, took 92.925872 s
+-----------+-------------------+
|time_of_day|average_trip_length|
+-----------+-------------------+
|      night|  5.323984802300156|
|    morning| 4.9273718666272845|
|  afternoon|  4.861410525884581|
|    evening|  4.484750367647451|
+-----------+-------------------+

2024-03-25 13:16:01,301 INFO server.AbstractConnector: Stopped Spark@6e9ea780{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2024-03-25 13:16:01,302 INFO ui.SparkUI: Stopped Spark web UI at http://starterkit-spark-app-e697848e75b772fe-driver-svc.data-science-ec23441.s
vc:4040
2024-03-25 13:16:01,306 INFO k8s.KubernetesClusterSchedulerBackend: Shutting down all executors
2024-03-25 13:16:01,307 INFO k8s.KubernetesClusterSchedulerBackend$KubernetesDriverEndpoint: Asking each executor to shut down
2024-03-25 13:16:01,314 WARN k8s.ExecutorPodsWatchSnapshotSource: Kubernetes client has been closed (this is expected if the application is shu
tting down.)
2024-03-25 13:16:02,367 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
2024-03-25 13:16:02,387 INFO memory.MemoryStore: MemoryStore cleared
2024-03-25 13:16:02,388 INFO storage.BlockManager: BlockManager stopped
2024-03-25 13:16:02,390 INFO storage.BlockManagerMaster: BlockManagerMaster stopped
2024-03-25 13:16:02,400 INFO scheduler.OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
2024-03-25 13:16:02,412 INFO spark.SparkContext: Successfully stopped SparkContext
2024-03-25 13:16:02,795 INFO util.ShutdownHookManager: Shutdown hook called
2024-03-25 13:16:02,797 INFO util.ShutdownHookManager: Deleting directory /var/data/spark-ae990723-8cb1-4096-a3a1-59c1ecd74831/spark-4a0378d8-6
994-4e0f-8d97-d97748c9ef96/pyspark-f44c6ae5-5cf8-4a3a-aaad-9b7e19b9b39a
2024-03-25 13:16:02,809 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-7b8f970a-f1bb-490b-a0a4-db3e9c2cb1cd
2024-03-25 13:16:02,820 INFO util.ShutdownHookManager: Deleting directory /var/data/spark-ae990723-8cb1-4096-a3a1-59c1ecd74831/spark-4a0378d8-6
994-4e0f-8d97-d97748c9ef96
```

Part 3:



```
2024-03-25 14:15:25,179 INFO scheduler.TaskSetManager: Finished task 196.0 in stage 30.0 (TID 1979) in 11 ms on 10.134.57.84 (executor 1) (197/
200)
2024-03-25 14:15:25,181 INFO scheduler.TaskSetManager: Starting task 199.0 in stage 30.0 (TID 1982, 10.134.24.221, executor 2, partition 199, P
ROCESS_LOCAL, 7626 bytes)
2024-03-25 14:15:25,183 INFO scheduler.TaskSetManager: Finished task 197.0 in stage 30.0 (TID 1980) in 10 ms on 10.134.24.221 (executor 2) (198
/200)
2024-03-25 14:15:25,187 INFO scheduler.TaskSetManager: Finished task 198.0 in stage 30.0 (TID 1981) in 9 ms on 10.134.57.84 (executor 1) (199/2
00)
2024-03-25 14:15:25,192 INFO scheduler.TaskSetManager: Finished task 199.0 in stage 30.0 (TID 1982) in 11 ms on 10.134.24.221 (executor 2) (200
/200)
2024-03-25 14:15:25,192 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 30.0, whose tasks have all completed, from pool
2024-03-25 14:15:25,193 INFO scheduler.DAGScheduler: ResultStage 30 (showString at NativeMethodAccessorImpl.java:0) finished in 1.122 s
2024-03-25 14:15:25,194 INFO scheduler.DAGScheduler: Job 19 is finished. Cancelling potential speculative or zombie tasks for this job
2024-03-25 14:15:25,194 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 30: Stage finished
2024-03-25 14:15:25,194 INFO scheduler.DAGScheduler: Job 19 finished: showString at NativeMethodAccessorImpl.java:0, took 174.801553 s
+-----------+-----------------------+
|time_of_day|average_earning_per_mile|
+-----------+-----------------------+
|    evening|      4.409928330698363|
|  afternoon|      4.363430869219534|
|    morning|      3.984544565617201|
|      night|     3.7730081413670082|
+-----------+-----------------------+

2024-03-25 14:15:25,220 INFO server.AbstractConnector: Stopped Spark@16aeb414{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2024-03-25 14:15:25,222 INFO ui.SparkUI: Stopped Spark web UI at http://starterkit-spark-app-f53c0c8e75ea57e1-driver-svc.data-science-ec23441.s
vc:4040
2024-03-25 14:15:25,227 INFO k8s.KubernetesClusterSchedulerBackend: Shutting down all executors
2024-03-25 14:15:25,228 INFO k8s.KubernetesClusterSchedulerBackend$KubernetesDriverEndpoint: Asking each executor to shut down
2024-03-25 14:15:25,235 WARN k8s.ExecutorPodsWatchSnapshotSource: Kubernetes client has been closed (this is expected if the application is shu
tting down.)
```

Challenges Faced:

1. Converting the trip_length column to float data type required handling data type errors.
2. Resolution- The resolution involved using the **cast()** method along with the **withColumn()** function to ensure accurate data type conversion.
3. Joining the avg_driver_total_pay and average_trip_length dataframes using the **join()** function with the appropriate join condition.

Knowledge Acquired:

1. Need to maintain data type consistency for accurate calculations.

Insights:

For Drivers-

1. Drivers earn the most in the afternoon, which may suggest a higher volume of rides or may be higher rates during these hours.
2. The morning and evening times also show substantial earnings, likely due to commuter traffic.
3. The longest trips occur at night, which may be due to a higher number of long-distance fares, like travelling to airport or railway stations.

For CEO-

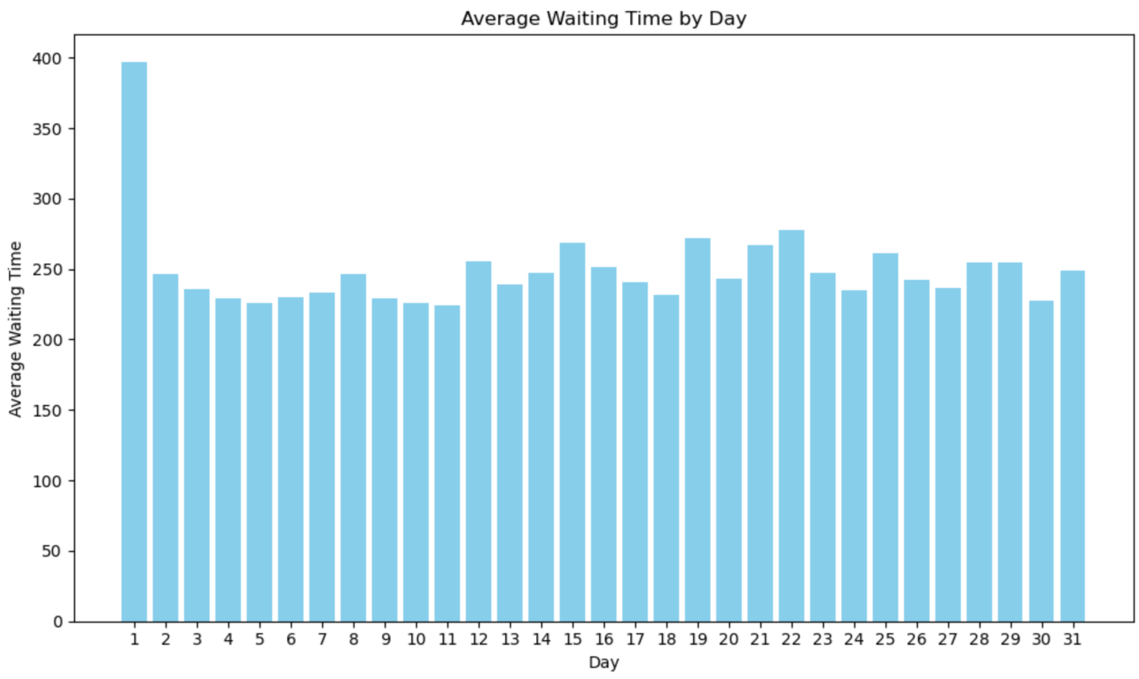
1. We can provide some discounts to boost number of rides during profitable times or encourage longer trips during off-peak hours. For example , we can provide discounts on rental cars during summer vacations.

TASK 5:

Explanation:

1. In this task we are calculating the average waiting time of each day for the month of January.
2. First I have converted the "request_to_pickup" column in the rideshare_with_month dataframe to integer data type using the **withColumn()** and **cast()** function.
3. A new dataframe named january_data is created by filtering the rideshare_with_month dataframe to include only the data for the month of January. This is achieved by using the **filter()** function with the condition **col("month") == 1.**
4. The **dayofmonth()** function from the pyspark.sql.functions module is applied to the "date" column of the january_data dataframe to extract the day component from the date.
5. The **groupBy()** function is applied to the "day" column of the january_data dataframe to group the records by day. Then, the **avg()** function is used to calculate the average value of the "request_to_pickup" column for each day.
6. The **repartition()** function is used to ensure that the data is written to a single partition before writing to a CSV file. The data is then written to a CSV file in the specified S3 bucket location using the **write()** function.

Result:



Challenges Faced:

1. Ensuring that the data is repartitioned to a single partition before writing to a CSV file can improve performance and avoid issues related to multiple small output files.
2. The resolution involved using the **repartition()** function.

Knowledge Acquired:

1. Understanding the options involved in writing dataframes to external storage like s3 bucket.

Insights:

1. We can observe from the visualization that **January 1st** has the most average waiting time.
2. Increased travel and celebrations around New Year's Day can lead to more ride requests, causing longer wait times for drivers.
3. We can see that every 7-8th day has a higher waiting time. Weekends tend to be busier for rideshare services in general as more people travel and go out. This naturally leads to higher wait times for riders and less availability for drivers.
4. There could be times when there simply aren't enough drivers on the road. This could be due to factors like bad weather or any festival.
5. Big concerts, sporting events, or conferences can create temporary spikes in demand for rides. This can lead to longer wait times for drivers.

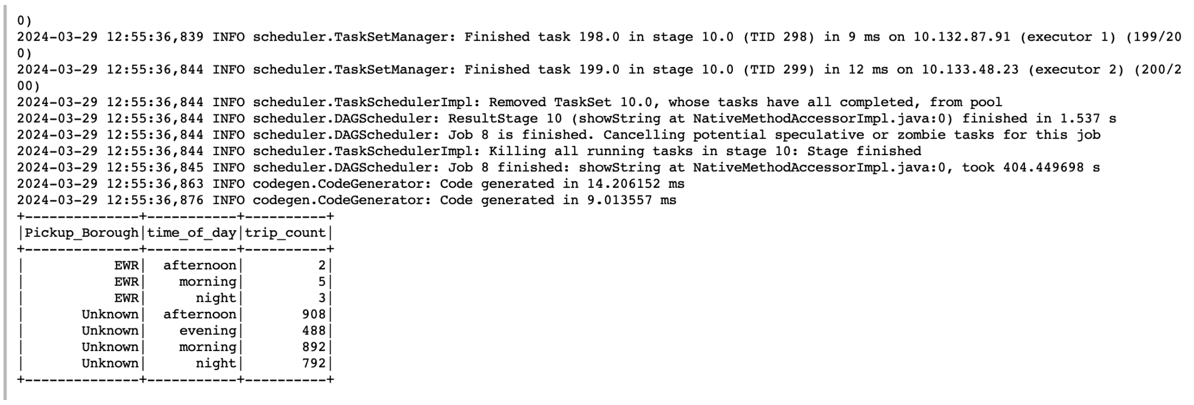
TASK 6:

Explanation:

1. In this task, we retrieve the trip count data for different pickup boroughs at different times of the day. It calculates the count of trips for each combination of pickup borough and time of day using the **COUNT(*)** function. The GROUP BY clause is used to group the data based on pickup borough and time of day. The HAVING clause filters the results to include only trip counts greater than 0 and less than 1000.
2. Next part focuses on trips occurring in the evening. We filter the data to include only trips where the time of day is 'evening' by using **time_of_day = 'evening'** condition**.**
3. In the third part of the task we retrieve data for trips that started in Brooklyn and ended in Staten Island. It selects the pickup borough, dropoff borough, and pickup zone (Pickup_Zone) columns from the rideshare_with_month DataFrame where the pickup borough is 'Brooklyn' and the dropoff borough is 'Staten Island'. It limits the output to the first 10 records using the LIMIT clause.
4. The total number of trips between Brookly and Staten Island is **69437**.

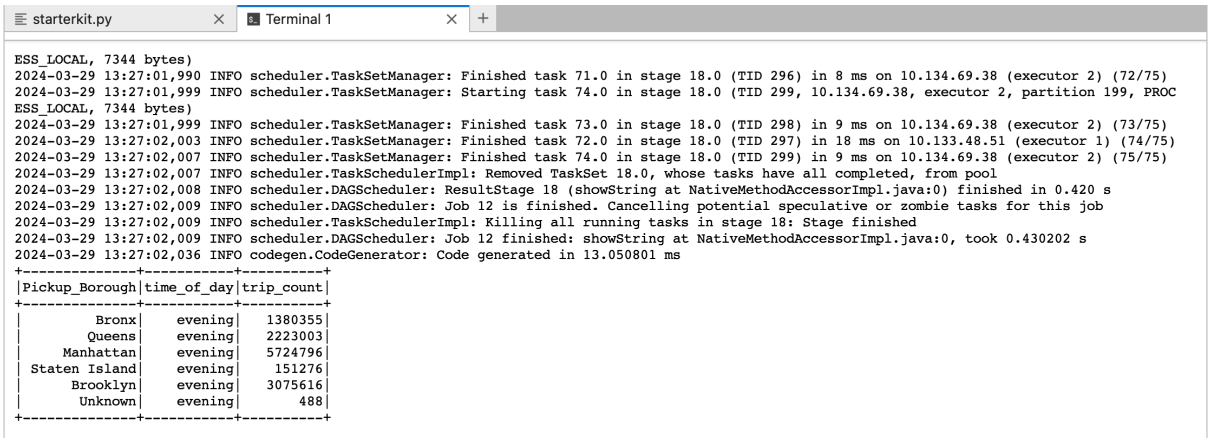
Result:

Part 1:



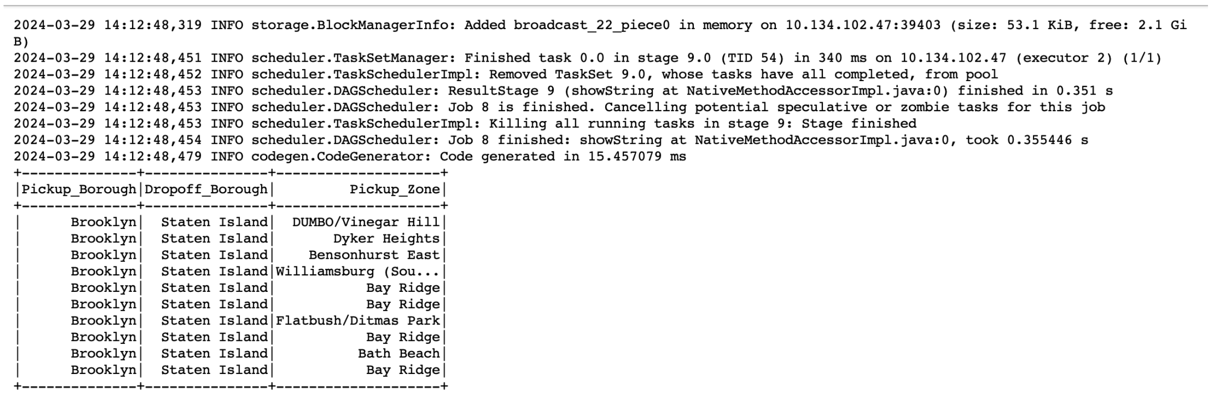
```
0)
2024-03-29 12:55:36,839 INFO scheduler.TaskSetManager: Finished task 198.0 in stage 10.0 (TID 298) in 9 ms on 10.132.87.91 (executor 1) (199/20
0)
2024-03-29 12:55:36,844 INFO scheduler.TaskSetManager: Finished task 199.0 in stage 10.0 (TID 299) in 12 ms on 10.133.48.23 (executor 2) (200/2
00)
2024-03-29 12:55:36,844 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 10.0, whose tasks have all completed, from pool
2024-03-29 12:55:36,844 INFO scheduler.DAGScheduler: ResultStage 10 (showString at NativeMethodAccessorImpl.java:0) finished in 1.537 s
2024-03-29 12:55:36,844 INFO scheduler.DAGScheduler: Job 8 is finished. Cancelling potential speculative or zombie tasks for this job
2024-03-29 12:55:36,844 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 10: Stage finished
2024-03-29 12:55:36,845 INFO scheduler.DAGScheduler: Job 8 finished: showString at NativeMethodAccessorImpl.java:0, took 404.449698 s
2024-03-29 12:55:36,863 INFO codegen.CodeGenerator: Code generated in 14.206152 ms
2024-03-29 12:55:36,876 INFO codegen.CodeGenerator: Code generated in 9.013557 ms
+--------------+-----------+----------+
|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|
+--------------+-----------+----------+
```

Part 2:



```
ESS_LOCAL, 7344 bytes)
2024-03-29 13:27:01,990 INFO scheduler.TaskSetManager: Finished task 71.0 in stage 18.0 (TID 296) in 8 ms on 10.134.69.38 (executor 2) (72/75)
2024-03-29 13:27:01,999 INFO scheduler.TaskSetManager: Starting task 74.0 in stage 18.0 (TID 299, 10.134.69.38, executor 2, partition 199, PROC
ESS_LOCAL, 7344 bytes)
2024-03-29 13:27:01,999 INFO scheduler.TaskSetManager: Finished task 73.0 in stage 18.0 (TID 298) in 9 ms on 10.134.69.38 (executor 2) (73/75)
2024-03-29 13:27:02,003 INFO scheduler.TaskSetManager: Finished task 72.0 in stage 18.0 (TID 297) in 18 ms on 10.133.48.51 (executor 1) (74/75)
2024-03-29 13:27:02,007 INFO scheduler.TaskSetManager: Finished task 74.0 in stage 18.0 (TID 299) in 9 ms on 10.134.69.38 (executor 2) (75/75)
2024-03-29 13:27:02,007 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 18.0, whose tasks have all completed, from pool
2024-03-29 13:27:02,008 INFO scheduler.DAGScheduler: ResultStage 18 (showString at NativeMethodAccessorImpl.java:0) finished in 0.420 s
2024-03-29 13:27:02,009 INFO scheduler.DAGScheduler: Job 12 is finished. Cancelling potential speculative or zombie tasks for this job
2024-03-29 13:27:02,009 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 18: Stage finished
2024-03-29 13:27:02,009 INFO scheduler.DAGScheduler: Job 12 finished: showString at NativeMethodAccessorImpl.java:0, took 0.430202 s
2024-03-29 13:27:02,036 INFO codegen.CodeGenerator: Code generated in 13.050801 ms
+--------------+-----------+----------+
|Pickup_Borough|time_of_day|trip_count|
+--------------+-----------+----------+
|         Bronx|    evening|   1380355|
|        Queens|    evening|   2223003|
|     Manhattan|    evening|   5724796|
| Staten Island|    evening|    151276|
|      Brooklyn|    evening|   3075616|
|       Unknown|    evening|       488|
+--------------+-----------+----------+
```

Part 3:



```
2024-03-29 14:12:48,319 INFO storage.BlockManagerInfo: Added broadcast_22_piece0 in memory on 10.134.102.47:39403 (size: 53.1 KiB, free: 2.1 Gi
B)
2024-03-29 14:12:48,451 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 9.0 (TID 54) in 340 ms on 10.134.102.47 (executor 2) (1/1)
2024-03-29 14:12:48,452 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 9.0, whose tasks have all completed, from pool
2024-03-29 14:12:48,453 INFO scheduler.DAGScheduler: ResultStage 9 (showString at NativeMethodAccessorImpl.java:0) finished in 0.351 s
2024-03-29 14:12:48,453 INFO scheduler.DAGScheduler: Job 8 is finished. Cancelling potential speculative or zombie tasks for this job
2024-03-29 14:12:48,453 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 9: Stage finished
2024-03-29 14:12:48,454 INFO scheduler.DAGScheduler: Job 8 finished: showString at NativeMethodAccessorImpl.java:0, took 0.355446 s
2024-03-29 14:12:48,479 INFO codegen.CodeGenerator: Code generated in 15.457079 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 (Sou...|
|      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|
+--------------+---------------+--------------------+
```

Challenges Faced:

1. Executing SQL queries on large datasets have impact on performance, especially if the queries involve complex aggregations or filters. It was essential to optimize queries and consider indexing columns for faster execution.

Knowledge Acquired:

1. Understanding how to apply filters in SQL queries using the WHERE clause to retrieve data that meets specific conditions.