IMPOPRTANT:

This report should only be read alongside the spark code, this PDF does not contain all the code, only the explanations and the outputs.

All Questions within a task in the python file should be ran separately, the single final task python file is mainly a guide (It does work but it is better to avoid trouble). (For some reason some questions cause conflict with each other).

All the work presented below is mine. As seen by the running time stamps in my screenshots, which (some) show my student's name: ec22954.

If you need to run a question yourself all the python files are named task(task_no.)-p(question no.) e.g. task2-p4.py will refer to the 4th question of task 2.

TASK 1:

Q1)

This question posed no challenge, I loaded the data frames with the data sets provided and printed the number of lines using the .count() method. However, utilising spark for the first time posed an exciting challenge. The code below reads: 22400728 entries.

```
2024-12-06 06:01:04,394 INFO scheduler.DAGScheduler: ResultStage 3 (count at NativeMethodAccessorImpl.java:0) finished in 0.270 s
2024-12-06 06:01:04,395 INFO scheduler.DAGScheduler: Job 2 is finished. Cancelling potential speculative or zombie tasks for this job
2024-12-06 06:01:04,395 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 3: Stage finished
2024-12-06 06:01:04,396 INFO scheduler.DAGScheduler: Job 2 finished: count at NativeMethodAccessorImpl.java:0, took 22.639288 s
The number of entries is: 22400728
2024-12-06 06:01:04,664 INFO datasources.FileSourceStrategy: Pruning directories with:
2024-12-06 06:01:04,672 INFO datasources.FileSourceStrategy: Pushed Filters: IsNotNull(tpep_pickup_datetime),IsNotNull(fare_amount),IsNotNull(trip_distance),GreaterThanOrEqual(tpep_
al(tpep_pickup_datetime,2023-02-07)
2024-12-06 06:01:04,673 INFO datasources.FileSourceStrategy: Post-Scan Filters: isnotnull(tpep_pickup_datetime#40),isnotnull(fare_amount#47),isnotnull(trip_distance#43),(cast(fare_
e#43 as int) < 1),(tpep_pickup_datetime#40 >= 2023-02-01),(tpep_pickup_datetime#40 <= 2023-02-07)
2024-12-06 06:01:04,674 INFO datasources.FileSourceStrategy: Output Data Schema: struct<tpep pickup datetime: string, trip distance: string, fare amount: string ... 1 more fields>
```

Q2)

Applied the .filter() method to filter the data set to meet certain criteria. The code is pretty clear, the fare amount < 550, the trip_distance less than 1 and the data is the first week of august.

I then formatted the date using the date_format() function to convert to yyyy-MM-dd and used the GroupBy() function to count the pick-up dates. I believe this was the fastest and easiest way to approach this problem.

```
2024-12-06 06:01:43,115 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 13.0, whose tasks have all completed, from pool
2024-12-06 06:01:43,115 INFO scheduler.DAGScheduler: ResultStage 13 (showString at NativeMethodAccessorImpl.java:0) finished in 0.480 s
2024-12-06 06:01:43,116 INFO scheduler.DAGScheduler: Job 7 is finished. Cancelling potential speculative or zombie tasks for this job
2024-12-06 06:01:43,116 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 13: Stage finished
2024-12-06 06:01:43,116 INFO scheduler.DAGScheduler: Job 7 finished: showString at NativeMethodAccessorImpl.java:0, took 0.487279 s
2024-12-06 06:01:43,140 INFO codegen.CodeGenerator: Code generated in 15.530032 ms
+----------+----------+
| trip_date|trip_count|
+----------+----------+
|2023-02-02|       311|
|2023-02-06|       344|
|2023-02-01|       275|
|2023-02-04|       315|
|2023-02-03|       318|
|2023-02-05|       377|
+----------+----------+

2024-12-06 06:01:43,160 INFO server.AbstractConnector: Stopped Spark@466c94fd{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2024-12-06 06:01:43,161 INFO ui.SparkUI: Stopped Spark web UI at http://zzzzzzzzzzzzfinaltask1-p2-5-efb5a7939a8dd3d4-driver-svc.data-science-ec22954.svc:4040
2024-12-06 06:01:43,166 INFO k8s.KubernetesClusterSchedulerBackend: Shutting down all executors
2024-12-06 06:01:43,166 INFO k8s.KubernetesClusterSchedulerBackend$KubernetesDriverEndpoint: Asking each executor to shut down
```

Q3)

I struggled quite a bit with this question at first. The join function was not working as intended at first, I believe it was because I was joining the wrong columns together so it was a human error which took me too long to fix.

One I have set the Location ID to their corresponding PULOcations and DOLOcations I dropped those and printed them.

However as seen by the code, I believe that I took the best approach, I joined the dataframes and manipulated the data efficiently.

The printSchematic() function was actually incredibly helpful throughout the project specially when it came to debugging and, understanding what exactly was in the data before playing around with it proved to be rather useful.

```
2024-12-06 05:07:44,962 INFO memory.MemoryStore: Block broadcast_5_piece0 stored as bytes in memory (estimated size 54.0 KiB, free 2003.5 MiB)
2024-12-06 05:07:44,963 INFO storage.BlockManagerInfo: Added broadcast_5_piece0 in memory on zzzzzzzzfinaltask1-p3-55db07939a5d6be6-driver-svc.data-science-ec22954.svc:7079 (size: 54.0 KiB, free: 2004.5 MiB)
2024-12-06 05:07:44,964 INFO spark.SparkContext: Created broadcast 5 from csv at NativeMethodAccessorImpl.java:0
2024-12-06 05:07:44,965 INFO execution.FileSourceScanExec: Planning scan with bin packing, max size: 134217728 bytes, open cost is considered as scanning 4194304 bytes.
root
 |-- tpep_pickup_datetime: string (nullable = true)
 |-- tpep_dropoff_datetime: string (nullable = true)
 |-- passenger_count: string (nullable = true)
 |-- trip_distance: string (nullable = true)
 |-- payment_type: string (nullable = true)
 |-- fare_amount: string (nullable = true)
 |-- extra: string (nullable = true)
 |-- mta_tax: string (nullable = true)
 |-- tip_amount: string (nullable = true)
 |-- tolls_amount: string (nullable = true)
 |-- total_amount: string (nullable = true)
 |-- congestion_surcharge: string (nullable = true)
 |-- airport_fee: string (nullable = true)
 |-- taxi_type: 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)

2024-12-06 05:07:45,224 INFO server.AbstractConnector: Stopped Spark@4e99a17{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2024-12-06 05:07:45,226 INFO ui.SparkUI: Stopped Spark web UI at http://zzzzzzzzzfinaltask1-p3-55db07939a5d6be6-driver-svc.data-science-ec22954.svc:4040
2024-12-06 05:07:45,230 INFO k8s.KubernetesClusterSchedulerBackend: Shutting down all executors
2024-12-06 05:07:45,231 INFO k8s.KubernetesClusterSchedulerBackend$KubernetesDriverEndpoint: Asking each executor to shut down
```

Q4)

This question was quite easy, it was adding 2 new columns and using the month() method to extract only the month from the datetime value. The new columns created where route and month.

The use of concat_ws() method is certainly the best approach to tackle the use of route, whilst the method month() would pick-out the month from the data available. I believe there was no quicker way.

```
2024-12-06 06:07:03,800 INFO scheduler.DAGScheduler: Job 3 is finished. Cancelling potential speculative or zombie tasks for this job
2024-12-06 06:07:03,800 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 3: Stage finished
2024-12-06 06:07:03,800 INFO scheduler.DAGScheduler: Job 3 finished: showString at NativeMethodAccessorImpl.java:0, took 4.995097 s
2024-12-06 06:07:03,856 INFO codegen.CodeGenerator: Code generated in 34.786616 ms
+--------------------+---------------------+---------------+-------------+------------+-----+-------+----------+-----------+------------+---------------------+-----------+-----------+--------------+-----------------+----------
-----+---------------------+---------------------+------------------------+-----+
|tpep_pickup_datetime|tpep_dropoff_datetime|passenger_count|trip_distance|payment_type|fare_amount|extra|mta_tax|tip_amount|tolls_amount|total_amount|congestion_surcharge|airport_fee|taxi_type  |Pickup_Borough|Pickup_zone      |Pickup_ser
rough|Dropoff_zone         |Dropoff_service_zone|route                |Month|
+--------------------+---------------------+---------------+-------------+------------+-----+-------+----------+-----------+------------+---------------------+-----------+-----------+--------------+-----------------+----------
-----+---------------------+---------------------+------------------------+-----+
|2023-01-01 00:32:10 |2023-01-01 00:40:36  |1.0            |0.97         |2           |9.3        |1.0  |0.5    |0.0       |0.0         |14.3        |2.5                 |0.0        |yellow_taxi|Manhattan     |Midtown Center   |Yellow Zon
     |Lenox Hill West      |Yellow Zone          |Manhattan to Manhattan|1    |
|2023-01-01 00:55:08 |2023-01-01 01:01:27  |1.0            |1.1          |1           |7.9        |1.0  |0.5    |4.0       |0.0         |16.9        |2.5                 |0.0        |yellow_taxi|Manhattan     |Central Park     |Yellow Zon
     |Upper East Side South|Yellow Zone          |Manhattan to Manhattan|1    |
|2023-01-01 00:25:04 |2023-01-01 00:37:49  |1.0            |2.51         |1           |14.9       |1.0  |0.5    |15.0      |0.0         |34.9        |2.5                 |0.0        |yellow_taxi|Manhattan     |Clinton East     |Yellow Zon
     |Upper West Side North|Yellow Zone          |Manhattan to Manhattan|1    |
|2023-01-01 00:03:48 |2023-01-01 00:13:25  |0.0            |1.9          |1           |12.1       |7.25 |0.5    |0.0       |0.0         |20.85       |0.0                 |1.25       |yellow_taxi|Queens        |LaGuardia Airport|Airports
     |Astoria              |Boro Zone            |Queens to Queens      |1    |
|2023-01-01 00:10:29 |2023-01-01 00:21:19  |1.0            |1.43         |1           |11.4       |1.0  |0.5    |3.28      |0.0         |19.68       |2.5                 |0.0        |yellow_taxi|Manhattan     |Gramercy         |Yellow Zon
     |East Village         |Yellow Zone          |Manhattan to Manhattan|1    |
+--------------------+---------------------+---------------+-------------+------------+-----+-------+----------+-----------+------------+---------------------+-----------+-----------+--------------+-----------------+----------
-----+---------------------+---------------------+------------------------+-----+
only showing top 5 rows

2024-12-06 06:07:03,877 INFO server.AbstractConnector: Stopped Spark@56b94faa{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2024-12-06 06:07:03,879 INFO ui.SparkUI: Stopped Spark web UI at http://zzzzzzzzzzzzfinaltask1-p4-1-e7c083939a939f5b-driver-svc.data-science-ec22954.svc:4040
2024-12-06 06:07:03,885 INFO k8s.KubernetesClusterSchedulerBackend: Shutting down all executors
```

Q5)

Aggregated all the passengers with their associated tip_amount and calculated the average tip by dividing the total tip amount by the passenger count. A fairly easy question with only a few new functions such as agg().

```
2024-12-06 06:10:47,361 INFO scheduler.DAGScheduler: Job 4 is finished. Cancelling potential speculative or zombie tasks for this job
2024-12-06 06:10:47,361 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 6: Stage finished
2024-12-06 06:10:47,361 INFO scheduler.DAGScheduler: Job 4 finished: showString at NativeMethodAccessorImpl.java:0, took 0.212794 s
2024-12-06 06:10:47,387 INFO codegen.CodeGenerator: Code generated in 17.257272 ms
+-----+---------------------+---------------+------------------+------------------------+
|Month|route                |passenger_count|total_tip_amount  |average_tip_per_passenger|
+-----+---------------------+---------------+------------------+------------------------+
|3    |Manhattan to Queens  |104937         |869138.4999999586 |8.282479011215859       |
|7    |Manhattan to Queens  |89890          |700702.1999999543 |7.795107353431464       |
|7    |Manhattan to Unknown |9902           |53321.17000000006 |5.38488891133105        |
|1    |Queens to Bronx      |6053           |27825.890000000014|4.597041136626468       |
|1    |Bronx to Queens      |330            |258.56            |0.7835151515151515      |
|3    |Unknown to EWR       |48             |492.66999999999996|10.263958333333333      |
|5    |Staten Island to EWR |1              |17.85             |17.85                   |
|6    |Unknown to Manhattan |2665           |8531.239999999994 |3.2012157598499043      |
|4    |Bronx to Bronx       |2479           |1618.55           |0.6529043969342476      |
|12   |Manhattan to Manhattan|30            |78.55             |2.618333333333333       |
+-----+---------------------+---------------+------------------+------------------------+
only showing top 10 rows

2024-12-06 06:10:47,406 INFO server.AbstractConnector: Stopped Spark@389afff9{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2024-12-06 06:10:47,407 INFO ui.SparkUI: Stopped Spark web UI at http://zzzzzzzzzzzzfinaltask1-p5-1-c963df939a961e6b-driver-svc.data-science-ec22954.svc:4040
2024-12-06 06:10:47,411 INFO k8s.KubernetesClusterSchedulerBackend: Shutting down all executors
2024-12-06 06:10:47,411 INFO k8s.KubernetesClusterSchedulerBackend$KubernetesDriverEndpoint: Asking each executor to shut down
```

Q6 & 7)

I filtered the zero average tip by setting the average tip to 0. And top routes

Where ordered by ordering the average tip descending. Because it is OrderedBy() then the routes will also fall into order. This was my thinking process. OrderedBy() meant that by placing the average tip/p in descending order, it would automatically align the other rows with it. This meant that there is no need to order every column but that through ordering one, all others fall into place. This is also done later on and proves to be the most efficient way to organise data.

```
2024-12-06 06:15:18,884 INFO scheduler.DAGScheduler: ResultStage 24 (showString at NativeMethodAccessorImpl.java:0) finished in 1.295 s
2024-12-06 06:15:18,884 INFO scheduler.DAGScheduler: Job 14 is finished. Cancelling potential speculative or zombie tasks for this job
2024-12-06 06:15:18,885 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 24: Stage finished
2024-12-06 06:15:18,885 INFO scheduler.DAGScheduler: Job 14 finished: showString at NativeMethodAccessorImpl.java:0, took 54.722592 s
2024-12-06 06:15:18,910 INFO codegen.CodeGenerator: Code generated in 16.180877 ms
+-----+-----------------------+---------------+----------------+------------------------+
|Month|route                  |passenger_count|total_tip_amount|average_tip_per_passenger|
+-----+-----------------------+---------------+----------------+------------------------+
|2    |EWR to Brooklyn        |1              |35.76           |35.76                   |
|3    |Staten Island to Unknown|4              |122.66          |30.665                  |
|4    |Bronx to EWR           |1              |30.51           |30.51                   |
|6    |EWR to Brooklyn        |2              |56.93           |28.465                  |
|2    |Bronx to EWR           |2              |55.26           |27.63                   |
|1    |EWR to Queens          |3              |82.47           |27.49                   |
|1    |Bronx to EWR           |1              |26.75           |26.75                   |
|1    |EWR to Manhattan       |2              |46.3            |23.15                   |
|1    |EWR to Staten Island   |3              |67.72           |22.573333333333334      |
|5    |Staten Island to Unknown|4              |87.75           |21.9375                 |
+-----+-----------------------+---------------+----------------+------------------------+
only showing top 10 rows

2024-12-06 06:15:18,925 INFO server.AbstractConnector: Stopped Spark@1a78c89b{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2024-12-06 06:15:18,927 INFO ui.SparkUI: Stopped Spark web UI at http://zzzzzzzzzzzzzfinaltask1-p6-1-556a65939a989581-driver-svc.data-science-ec22954.svc:4040
2024-12-06 06:15:18,932 INFO k8s.KubernetesClusterSchedulerBackend: Shutting down all executors
2024-12-06 06:15:18,933 INFO k8s.KubernetesClusterSchedulerBackend$KubernetesDriverEndpoint: Asking each executor to shut down
2024-12-06 06:15:18,942 WARN k8s.ExecutorPodsWatchSnapshotSource: Kubernetes client has been closed (this is expected if the application is shutting down.)
2024-12-06 06:15:19,703 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
2024-12-06 06:15:19,722 INFO memory.MemoryStore: MemoryStore cleared
```

TASK 2:

Q1)

Very simply I followed a straight forward approach similar to that of task 1. The printSchema() function was used to display the schema of the dataframe we are enquiring. It gives a clear view of the structure of the data, this includes column types, names, nulls…

This meant that I could understand any issues that could arise with the data that I want to manipulate before they came.

```
2024-12-06 06:21:24,208 INFO memory.MemoryStore: Block broadcast_5_piece0 stored as bytes in memory (estimated size 54.1 KiB, free 2003.5 MiB)
2024-12-06 06:21:24,210 INFO storage.BlockManagerInfo: Added broadcast_5_piece0 in memory on zzzzzzzzzzzzfinaltask2-p2-1-af2a96939aa0c7a7-driver-svc.data-science-ec22954.svc:7079 (size: 54.1 KiB, fre
2024-12-06 06:21:24,212 INFO spark.SparkContext: Created broadcast 5 from csv at NativeMethodAccessorImpl.java:0
2024-12-06 06:21:24,213 INFO execution.FileSourceScanExec: Planning scan with bin packing, max size: 93100933 bytes, open cost is considered as scanning 4194304 bytes.
root
 |-- hash: string (nullable = true)
 |-- nonce: string (nullable = true)
 |-- block_hash: string (nullable = true)
 |-- block_number: string (nullable = true)
 |-- transaction_index: string (nullable = true)
 |-- from_address: string (nullable = true)
 |-- to_address: string (nullable = true)
 |-- value: string (nullable = true)
 |-- gas: string (nullable = true)
 |-- gas_price: string (nullable = true)
 |-- input: string (nullable = true)
 |-- block_timestamp: string (nullable = true)
 |-- max_fee_per_gas: string (nullable = true)
 |-- max_priority_fee_per_gas: string (nullable = true)
 |-- transaction_type: string (nullable = true)

2024-12-06 06:21:24,424 INFO datasources.FileSourceStrategy: Pruning directories with:
2024-12-06 06:21:24,425 INFO datasources.FileSourceStrategy: Pushed Filters:
2024-12-06 06:21:24,425 INFO datasources.FileSourceStrategy: Post-Scan Filters:
2024-12-06 06:21:24,426 INFO datasources.FileSourceStrategy: Output Data Schema: struct<miner: string, size: string>
2024-12-06 06:21:24,470 INFO codegen.CodeGenerator: Code generated in 20.376268 ms
2024-12-06 06:21:24,504 INFO codegen.CodeGenerator: Code generated in 22.66602 ms
2024-12-06 06:21:24,510 INFO memory.MemoryStore: Block broadcast_6 stored as values in memory (estimated size 523.2 KiB, free 2002.9 MiB)
2024-12-06 06:21:24,533 INFO memory.MemoryStore: Block broadcast_6_piece0 stored as bytes in memory (estimated size 54.1 KiB, free 2002.9 MiB)
2024-12-06 06:21:24,535 INFO storage.BlockManagerInfo: Added broadcast_6_piece0 in memory on zzzzzzzzzzzzfinaltask2-p2-1-af2a96939aa0c7a7-driver-svc.data-science-ec22954.svc:7079 (size: 54.1 KiB, fre
2024-12-06 06:21:24,536 INFO spark.SparkContext: Created broadcast 6 from showString at NativeMethodAccessorImpl.java:0
2024-12-06 06:21:24,541 INFO execution.FileSourceScanExec: Planning scan with bin packing, max size: 134217728 bytes, open cost is considered as scanning 4194304 bytes.
```

```
2024-12-06 06:21:23,327 INFO memory.MemoryStore: Block broadcast_2_piece0 stored as bytes in memory (estimated size 54.1 KiB, free 2003.5 MiB)
2024-12-06 06:21:23,328 INFO storage.BlockManagerInfo: Added broadcast_2_piece0 in memory on zzzzzzzzzzzzfinaltask2-p2-1-af2a96939aa0c7a7-driver-svc.data-science-ec22954.svc:7079 (size: 54.1 KiB, free:
2024-12-06 06:21:23,329 INFO spark.SparkContext: Created broadcast 2 from csv at NativeMethodAccessorImpl.java:0
2024-12-06 06:21:23,331 INFO execution.FileSourceScanExec: Planning scan with bin packing, max size: 134217728 bytes, open cost is considered as scanning 4194304 bytes.
root
 |-- number: string (nullable = true)
 |-- hash: string (nullable = true)
 |-- parent_hash: string (nullable = true)
 |-- nonce: string (nullable = true)
 |-- sha3_uncles: string (nullable = true)
 |-- logs_bloom: string (nullable = true)
 |-- transactions_root: string (nullable = true)
 |-- state_root: string (nullable = true)
 |-- receipts_root: string (nullable = true)
 |-- miner: string (nullable = true)
 |-- difficulty: string (nullable = true)
 |-- total_difficulty: string (nullable = true)
 |-- size: string (nullable = true)
 |-- extra_data: string (nullable = true)
 |-- gas_limit: string (nullable = true)
 |-- gas_used: string (nullable = true)
 |-- timestamp: string (nullable = true)
 |-- transaction_count: string (nullable = true)
 |-- base_fee_per_gas: string (nullable = true)

2024-12-06 06:21:23,520 INFO datasources.InMemoryFileIndex: It took 14 ms to list leaf files for 1 paths.
2024-12-06 06:21:23,546 INFO datasources.InMemoryFileIndex: It took 8 ms to list leaf files for 1 paths.
2024-12-06 06:21:23,704 INFO datasources.FileSourceStrategy: Pruning directories with:
2024-12-06 06:21:23,706 INFO datasources.FileSourceStrategy: Pushed Filters:
2024-12-06 06:21:23,707 INFO datasources.FileSourceStrategy: Post-Scan Filters: (length(trim(value#54, None)) > 0)
```

Q2)

Here I used 2 important functions orderBy() which orders the columns of my dataframe with regards to a certain column. So in this case I orderedBy size so that I could get the top sizes mined and order the data frame by that standard. The other is desc() which simply orders them in descending order. (as asked). This is another example once again of being effective by re-arranging a single column and as that happens all other fall into place. Here:

```
2024-12-06 06:21:33,896 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 2: Stage finished
2024-12-06 06:21:33,897 INFO scheduler.DAGScheduler: Job 2 finished: showString at NativeMethodAccessorImpl.java:0, took 9.304849 s
2024-12-06 06:21:33,926 INFO codegen.CodeGenerator: Code generated in 19.13818 ms
2024-12-06 06:21:33,953 INFO codegen.CodeGenerator: Code generated in 18.43446 ms
+------------------+----------+
|             miner|total_size|
+------------------+----------+
|0x52bc44d5378309e...|       999|
|0x52bc44d5378309e...|       999|
|0x77120710f4bf338...|       999|
|0x1dcb8d1f0fcc8cb...|       999|
|0x52bc44d5378309e...|       999|
|0x52bc44d5378309e...|       999|
|0x52bc44d5378309e...|       999|
|0x52bc44d5378309e...|       999|
|0x52bc44d5378309e...|       999|
|0x52bc44d5378309e...|       999|
+------------------+----------+
only showing top 10 rows

2024-12-06 06:21:33,974 INFO server.AbstractConnector: Stopped Spark@69474243{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2024-12-06 06:21:33,976 INFO ui.SparkUI: Stopped Spark web UI at http://zzzzzzzzzzzzzfinaltask2-p2-1-af2a96939aa0c7a7-driver-svc.data-science-ec22954.svc:4040
2024-12-06 06:21:33,980 INFO k8s.KubernetesClusterSchedulerBackend: Shutting down all executors
2024-12-06 06:21:33,981 INFO k8s.KubernetesClusterSchedulerBackend$KubernetesDriverEndpoint: Asking each executor to shut down
2024-12-06 06:21:33,989 WARN k8s.ExecutorPodsWatchSnapshotSource: Kubernetes client has been closed (this is expected if the application is shutting down.)
2024-12-06 06:21:34,675 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
```

As you see in the picture above, I made the mistake a couple of times that the show() function sets truncate to True by default. I later learnt to change this and my results become clearer. I learnt to be aware of this issue, not being able to visualise the data means poor testing or outright wrong output. Fixing this was essential and so I did:

```
2024-12-06 12:09:59,093 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 2: Stage finished
2024-12-06 12:09:59,093 INFO scheduler.DAGScheduler: Job 2 finished: showString at NativeMethodAccessorImpl.java:0, took 5.226119 s
2024-12-06 12:09:59,117 INFO codegen.CodeGenerator: Code generated in 13.668842 ms
2024-12-06 12:09:59,136 INFO codegen.CodeGenerator: Code generated in 13.657908 ms
+----------------------------------------+----------+
|miner                                   |total_size|
+----------------------------------------+----------+
|0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5|999       |
|0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5|999       |
|0x77120710f4bf3383527dfb607a3b07b774276767|999       |
|0x1dcb8d1f0fcc8cbc8c2d76528e877f915e299fbe|999       |
|0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5|999       |
|0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5|999       |
|0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5|999       |
|0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5|999       |
|0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5|999       |
|0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5|999       |
+----------------------------------------+----------+
only showing top 10 rows

2024-12-06 12:09:59,200 INFO datasources.FileSourceStrategy: Pruning directories with:
2024-12-06 12:09:59,201 INFO datasources.FileSourceStrategy: Pushed Filters:
2024-12-06 12:09:59,201 INFO datasources.FileSourceStrategy: Post-Scan Filters:
```

Q3)

Here 2 very important function were used. date_format which successfully changed the format of the date to yyyy-MM-dd format. The other one which was imbedded within date_format, was from_unixtime which changed the format from a unix format to a regular date_format. Here is my output:

```
2024-12-06 06:23:20,832 INFO scheduler.DAGScheduler: ResultStage 2 (showString at NativeMethodAccessorImpl.java:0) finished in 0.413 s
2024-12-06 06:23:20,832 INFO scheduler.DAGScheduler: Job 2 is finished. Cancelling potential speculative or zombie tasks for this job
2024-12-06 06:23:20,832 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 2: Stage finished
2024-12-06 06:23:20,832 INFO scheduler.DAGScheduler: Job 2 finished: showString at NativeMethodAccessorImpl.java:0, took 0.420840 s
2024-12-06 06:23:20,871 INFO codegen.CodeGenerator: Code generated in 23.537984 ms
+----------+--------------+
| timestamp|formatted_date|
+----------+--------------+
|         0|    1970-01-01|
|1438269988|    2015-07-30|
|1438270017|    2015-07-30|
|1438270048|    2015-07-30|
|1438270077|    2015-07-30|
|1438270083|    2015-07-30|
|1438270107|    2015-07-30|
|1438270110|    2015-07-30|
|1438270112|    2015-07-30|
|1438270115|    2015-07-30|
+----------+--------------+
only showing top 10 rows

2024-12-06 06:23:20,899 INFO server.AbstractConnector: Stopped Spark@652c2443{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2024-12-06 06:23:20,902 INFO ui.SparkUI: Stopped Spark web UI at http://zzzzzzzzzzzzzfinaltask2-p3-1-3c9d8f939aa28738-driver-svc.data-science-ec22954.svc:4040
2024-12-06 06:23:20,908 INFO k8s.KubernetesClusterSchedulerBackend: Shutting down all executors
2024-12-06 06:23:20,909 INFO k8s.KubernetesClusterSchedulerBackend$KubernetesDriverEndpoint: Asking each executor to shut down
2024-12-06 06:23:20,921 WARN k8s.ExecutorPodsWatchSnapshotSource: Kubernetes client has been closed (this is expected if the application is shutting down.)
```

Q4)

This code aimed to perform an inner join between the 2 datasets provided by joining their similar hash columns. Here is the output for the number of entries: The join function has certainly been quite a challenge for me whilst working on this. I could not understand when an inner, outer, left or right join should take place. It was through trial and error that I understood the different outcomes each input gave.

```
2024-12-06 06:27:44,744 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 5.0, whose tasks have all completed, from pool
2024-12-06 06:27:44,746 INFO scheduler.DAGScheduler: ResultStage 5 (count at NativeMethodAccessorImpl.java:0) finished in 0.080 s
2024-12-06 06:27:44,746 INFO scheduler.DAGScheduler: Job 2 is finished. Cancelling potential speculative or zombie tasks for this job
2024-12-06 06:27:44,746 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 5: Stage finished
2024-12-06 06:27:44,747 INFO scheduler.DAGScheduler: Job 2 finished: count at NativeMethodAccessorImpl.java:0, took 16.611026 s
The number of lines is: 504708
2024-12-06 06:27:44,764 INFO server.AbstractConnector: Stopped Spark@2dd3bc7d{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2024-12-06 06:27:44,766 INFO ui.SparkUI: Stopped Spark web UI at http://zzzzzzzzzzzzzfinaltask2-p4-1-d9b7ba939aa664fc-driver-svc.data-science-ec22954.svc:4040
2024-12-06 06:27:44,770 INFO k8s.KubernetesClusterSchedulerBackend: Shutting down all executors
2024-12-06 06:27:44,771 INFO k8s.KubernetesClusterSchedulerBackend$KubernetesDriverEndpoint: Asking each executor to shut down
2024-12-06 06:27:44,778 WARN k8s.ExecutorPodsWatchSnapshotSource: Kubernetes client has been closed (this is expected if the application is shutting down.)
```

Q5)

This section extracts the entries from only September by utilising the filter() function. By doing this a range between the following dates: col("formatted_date") >= "2015-09-01") & (col("formatted_date") < "2015-10-01" wav be used to only select the month of September. On top of this I grouped by formatted date and aggregated block_count and unique senders count number. Here is my output of the Datagraph (Incomplete):

```
2024-12-06 06:32:00,532 INFO scheduler.DAGScheduler: Job 2 finished: showString at NativeMethodAccessorImpl.java:0, took 52.056151 s
2024-12-06 06:32:00,554 INFO codegen.CodeGenerator: Code generated in 13.684788 ms
2024-12-06 06:32:00,570 INFO codegen.CodeGenerator: Code generated in 11.009585 ms
+--------------+-----------+--------------------------+
|formatted_date|block_count|unique_senders_count_number|
+--------------+-----------+--------------------------+
|    2015-09-01|       1411|                       916|
|    2015-09-02|       1374|                       875|
|    2015-09-03|       1220|                       778|
|    2015-09-04|       1419|                       761|
|    2015-09-05|       1571|                       830|
|    2015-09-06|       1556|                       965|
|    2015-09-07|       1655|                       969|
|    2015-09-08|       1725|                      1003|
|    2015-09-09|       1732|                       980|
|    2015-09-10|       1685|                       989|
|    2015-09-11|       1870|                      1088|
|    2015-09-12|       1701|                       942|
|    2015-09-13|       1656|                       954|
|    2015-09-14|       1739|                      1013|
|    2015-09-15|       1673|                       906|
|    2015-09-16|       2050|                       920|
|    2015-09-17|       1883|                       982|
|    2015-09-18|       2035|                       891|
|    2015-09-19|       1786|                       840|
|    2015-09-20|       2206|                       904|
+--------------+-----------+--------------------------+
only showing top 20 rows

2024-12-06 06:32:00,589 INFO server.AbstractConnector: Stopped Spark@718b6a5{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2024-12-06 06:32:00,591 INFO ui.SparkUI: Stopped Spark web UI at http://zzzzzzzzzzzzzfinaltask2-p5-1-1febc2939aa9bd6a-driver-svc.data-science-ec22954
2024-12-06 06:32:00,596 INFO k8s.KubernetesClusterSchedulerBackend: Shutting down all executors
2024-12-06 06:32:00,596 INFO k8s.KubernetesClusterSchedulerBackend$KubernetesDriverEndpoint: Asking each executor to shut down
2024-12-06 06:32:00,605 WARN k8s.ExecutorPodsWatchSnapshotSource: Kubernetes client has been closed (this is expected if the application is shutting
2024-12-06 06:32:01,397 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
```

Q6)

I primarily began by applying a filter to the formatted data so that it would only select the days of October, This makes it much easier to work with and gives opportunity for better quality outcomes / insights. I then performed a calculation by multiplying the gas column with the gas price column. This provided essential information regarding the total transaction fee of gas.

I lastly applied another filterwhich was used to refine the output data to only include rows where the transaction index was equal to 0. This made sure that the information being outputted stayed relevant to the desired criteria.

```
2024-12-06 06:36:11,031 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 7: Stage finished
2024-12-06 06:36:11,031 INFO scheduler.DAGScheduler: Job 2 finished: showString at NativeMethodAccessorImpl.java:0, took 34.556880 s
2024-12-06 06:36:11,065 INFO codegen.CodeGenerator: Code generated in 20.424756 ms
2024-12-06 06:36:11,093 INFO codegen.CodeGenerator: Code generated in 18.003883 ms
+--------------+---------------------+
|formatted_date|total_transaction_fee|
+--------------+---------------------+
|    2015-10-01| 1.465063073190187...|
|    2015-10-02| 2.464981746013086E19|
|    2015-10-03| 2.625289865558591E19|
|    2015-10-04| 2.438632750193587E19|
|    2015-10-05| 2.637662450505165...|
|    2015-10-06| 2.418495276446883E19|
|    2015-10-07| 3.778585599630305E19|
|    2015-10-08| 5.705912795259012E19|
|    2015-10-09| 2.902720072304586...|
|    2015-10-10| 2.201258740591100...|
|    2015-10-11| 1.619052176793014...|
|    2015-10-12| 2.168410861421186...|
|    2015-10-13| 5.992645513202179E19|
|    2015-10-14| 3.749759812719113E19|
|    2015-10-15| 4.601426445537218E19|
|    2015-10-16|  4.32299763950952E19|
|    2015-10-17| 2.423809054826823...|
|    2015-10-18| 2.876176820383356E19|
|    2015-10-19| 2.647615515603579E19|
|    2015-10-20| 3.402382771689090...|
+--------------+---------------------+
only showing top 20 rows

2024-12-06 06:36:11,119 INFO server.AbstractConnector: Stopped Spark@560c9579{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2024-12-06 06:36:11,121 INFO ui.SparkUI: Stopped Spark web UI at http://zzzzzzzzzzzzzfinaltask2-p6-1-d26f03939aadc05f-driver-svc.data-science-ec22954.svc:4040
2024-12-06 06:36:11,128 INFO k8s.KubernetesClusterSchedulerBackend: Shutting down all executors
2024-12-06 06:36:11,129 INFO k8s.KubernetesClusterSchedulerBackend$KubernetesDriverEndpoint: Asking each executor to shut down
2024-12-06 06:36:11,142 WARN k8s.ExecutorPodsWatchSnapshotSource: Kubernetes client has been closed (this is expected if the application is shutting down.)
2024-12-06 06:36:12,228 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
2024-12-06 06:36:12,253 INFO memory.MemoryStore: MemoryStore cleared
2024-12-06 06:36:12,254 INFO storage.BlockManager: BlockManager stopped
```

TASK 3:

Q1)

Once Again, I printed the number of entries in the given Dataframe, not too difficult:

```
2024-12-06 06:41:01,305 INFO scheduler.DAGScheduler: ResultStage 3 (count at NativeMethodAccessorImpl.java:0) finished in 0.233 s
2024-12-06 06:41:01,306 INFO scheduler.DAGScheduler: Job 2 is finished. Cancelling potential speculative or zombie tasks for this job
2024-12-06 06:41:01,306 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 3: Stage finished
2024-12-06 06:41:01,307 INFO scheduler.DAGScheduler: Job 2 finished: count at NativeMethodAccessorImpl.java:0, took 1.703361 s
The number of entries is: 466523
2024-12-06 06:41:01,537 INFO datasources.InMemoryFileIndex: It took 55 ms to list leaf files for 7 paths.
2024-12-06 06:41:01,601 INFO datasources.InMemoryFileIndex: It took 52 ms to list leaf files for 7 paths.
2024-12-06 06:41:01,712 INFO datasources.FileSourceStrategy: Pruning directories with:
2024-12-06 06:41:01,712 INFO datasources.FileSourceStrategy: Pushed Filters:
2024-12-06 06:41:01,712 INFO datasources.FileSourceStrategy: Post-Scan Filters: (length(trim(value#96, None)) > 0)
```

Q2)

Here I faced a challenge, since I was faced with issues when defining the StructTypes for the edges and vertecies. I did not quite understand what the purpose of this was. This meant that I was confused as to what I was doing. However. After going through the labs and enquiring online, the purpose for the use of these surprised me. They are higly applicable data processing techniques for modelling complex relationships between data point. It could prove to be useful for identifying interconnected nodes. Lab 7 however gave me the knowledge necessary for me to work on it and so I did. I firstly

defined the different StructFields within the StructType ‘method’ ensuring that my columns are the correct type. I then created 2 data frames and applied this logic to them. I used the show() function to display them. Here is the output:

```
2024-12-06 06:41:02,433 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 5: Stage finished
2024-12-06 06:41:02,434 INFO scheduler.DAGScheduler: Job 4 finished: showString at NativeMethodAccessorImpl.java:0, took 0.190642 s
2024-12-06 06:41:02,462 INFO codegen.CodeGenerator: Code generated in 20.847286 ms
+---+---+
|src|dst|
+---+---+
|82 |196|
|7  |7  |
|7  |7  |
|166|74 |
|236|229|
+---+---+
only showing top 5 rows

2024-12-06 06:41:02,521 INFO datasources.FileSourceStrategy: Pruning directories with:
2024-12-06 06:41:02,522 INFO datasources.FileSourceStrategy: Pushed Filters:
2024-12-06 06:41:02,522 INFO datasources.FileSourceStrategy: Post-Scan Filters:
2024-12-06 06:41:02,522 INFO datasources.FileSourceStrategy: Output Data Schema: struct<LocationID: int, Borough: string, Zone: string, service_zone: string ... 2 mor

2024-12-06 06:41:02,822 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 6: Stage finished
2024-12-06 06:41:02,823 INFO scheduler.DAGScheduler: Job 5 finished: showString at NativeMethodAccessorImpl.java:0, took 0.220655 s
2024-12-06 06:41:02,851 INFO codegen.CodeGenerator: Code generated in 18.560612 ms
+----------+-------------+-----------------------+------------+
|LocationID|Borough      |Zone                   |service_zone|
+----------+-------------+-----------------------+------------+
|1         |EWR          |Newark Airport         |EWR         |
|2         |Queens       |Jamaica Bay            |Boro Zone   |
|3         |Bronx        |Allerton/Pelham Gardens|Boro Zone   |
|4         |Manhattan    |Alphabet City          |Yellow Zone |
|5         |Staten Island|Arden Heights          |Boro Zone   |
+----------+-------------+-----------------------+------------+
only showing top 5 rows

2024-12-06 06:41:02,867 INFO server.AbstractConnector: Stopped Spark@3b717763{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2024-12-06 06:41:02,869 INFO ui.SparkUI: Stopped Spark web UI at http://zzzzzzzzzzzzzfinaltask3-p2-1-ba021c939ab29f8f-driver-svc.data-science-ec22954.svc:4040
2024-12-06 06:41:02,876 INFO k8s.KubernetesClusterSchedulerBackend: Shutting down all executors
2024-12-06 06:41:02,877 INFO k8s.KubernetesClusterSchedulerBackend$KubernetesDriverEndpoint: Asking each executor to shut down
2024-12-06 06:41:02,887 WARN k8s.ExecutorPodsWatchSnapshotSource: Kubernetes client has been closed (this is expected if the application is shutting down.)
2024-12-06 06:41:03,712 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
```

Q3)

I then used the triplets function for analysing the relationship between the Pick up locations and the Drop off destinations. It was useful for displaying the vertices and edges that connect them. Gere was my output:

```
2024-12-06 06:53:01,643 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 5.0, whose tasks have all completed, from pool
2024-12-06 06:53:01,644 INFO scheduler.DAGScheduler: ResultStage 5 (showString at NativeMethodAccessorImpl.java:0) finished in 0.268 s
2024-12-06 06:53:01,644 INFO scheduler.DAGScheduler: Job 5 is finished. Cancelling potential speculative or zombie tasks for this job
2024-12-06 06:53:01,644 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 5: Stage finished
2024-12-06 06:53:01,644 INFO scheduler.DAGScheduler: Job 5 finished: showString at NativeMethodAccessorImpl.java:0, took 0.275627 s
2024-12-06 06:53:01,671 INFO codegen.CodeGenerator: Code generated in 19.089784 ms
+--------------------------------------------------+---------+--------------------------------------------------------+
|src                                               |edge     |dst                                                     |
+--------------------------------------------------+---------+--------------------------------------------------------+
|[82, Queens, Elmhurst, Boro Zone]                 |[82, 196]|[196, Queens, Rego Park, Boro Zone]                     |
|[7, Queens, Astoria, Boro Zone]                   |[7, 7]   |[7, Queens, Astoria, Boro Zone]                         |
|[7, Queens, Astoria, Boro Zone]                   |[7, 7]   |[7, Queens, Astoria, Boro Zone]                         |
|[166, Manhattan, Morningside Heights, Boro Zone]  |[166, 74]|[74, Manhattan, East Harlem North, Boro Zone]           |
|[236, Manhattan, Upper East Side North, Yellow Zone]|[236, 229]|[229, Manhattan, Sutton Place/Turtle Bay North, Yellow Zone]|
|[75, Manhattan, East Harlem South, Boro Zone]     |[75, 235]|[235, Bronx, University Heights/Morris Heights, Boro Zone]|
|[260, Queens, Woodside, Boro Zone]                |[260, 160]|[160, Queens, Middle Village, Boro Zone]               |
|[95, Queens, Forest Hills, Boro Zone]             |[95, 264]|[264, Unknown, NV, N/A]                                 |
|[244, Manhattan, Washington Heights South, Boro Zone]|[244, 41]|[41, Manhattan, Central Harlem, Boro Zone]             |
|[83, Queens, Elmhurst/Maspeth, Boro Zone]         |[83, 7]  |[7, Queens, Astoria, Boro Zone]                         |
|[223, Queens, Steinway, Boro Zone]                |[223, 223]|[223, Queens, Steinway, Boro Zone]                     |
|[260, Queens, Woodside, Boro Zone]                |[260, 260]|[260, Queens, Woodside, Boro Zone]                     |
|[82, Queens, Elmhurst, Boro Zone]                 |[82, 193]|[193, Queens, Queensbridge/Ravenswood, Boro Zone]       |
|[146, Queens, Long Island City/Queens Plaza, Boro Zone]|[146, 145]|[145, Queens, Long Island City/Hunters Point, Boro Zone]|
|[146, Queens, Long Island City/Queens Plaza, Boro Zone]|[146, 145]|[145, Queens, Long Island City/Hunters Point, Boro Zone]|
|[130, Queens, Jamaica, Boro Zone]                 |[130, 191]|[191, Queens, Queens Village, Boro Zone]               |
|[95, Queens, Forest Hills, Boro Zone]             |[95, 134]|[134, Queens, Kew Gardens, Boro Zone]                   |
|[74, Manhattan, East Harlem North, Boro Zone]     |[74, 152]|[152, Manhattan, Manhattanville, Boro Zone]             |
|[95, Queens, Forest Hills, Boro Zone]             |[95, 173]|[173, Queens, North Corona, Boro Zone]                  |
|[210, Brooklyn, Sheepshead Bay, Boro Zone]        |[210, 155]|[155, Brooklyn, Marine Park/Mill Basin, Boro Zone]      |
+--------------------------------------------------+---------+--------------------------------------------------------+
only showing top 20 rows

2024-12-06 06:53:01,694 INFO server.AbstractConnector: Stopped Spark@3a04452e{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2024-12-06 06:53:01,698 INFO ui.SparkUI: Stopped Spark web UI at http://zzzzzzzzzzzzzfinaltask3-p3-4-3e4083939abda8aa-driver-svc.data-science-ec22954.svc:4040
2024-12-06 06:53:01,703 INFO k8s.KubernetesClusterSchedulerBackend: Shutting down all executors
2024-12-06 06:53:01,704 INFO k8s.KubernetesClusterSchedulerBackend$KubernetesDriverEndpoint: Asking each executor to shut down
2024-12-06 06:53:01,718 WARN k8s.ExecutorPodsWatchSnapshotSource: Kubernetes client has been closed (this is expected if the application is shutting down.)
2024-12-06 06:53:02,504 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
2024-12-06 06:53:02,524 INFO memory.MemoryStore: MemoryStore cleared
```

This method was certainly insightful. I can more clearly see now after working with it myself the usefulness of creating connected data. It makes everything so much easier to understand and uncover much more information when diving into it. They help to uncover relationships and patterns that otherwise would be too obscure. And I certainly see the use of it too in within different employment sectors such as cyber security, and how connected data helps uncover suspeicious activity with greater ease.

Q4)

I then proceeded to count the connected vertices by using the find() function together with the count() function. After successfully counting these connected vertices I set up a dataframe. This dataframe includes both source and destination renamed in 2 columns as “id” and together with this their corresponding Boroughs and service_zones.

This creates a comprehensive view of the relationships between Borough and service_zone. This was incredible to work with and insightful. It proves to be a very efficient way of bringing out patterns from data.

```
2024-12-06 06:49:03,332 INFO scheduler.DAGScheduler: Job 8 is finished. Cancelling potential speculative or zombie tasks for this job
2024-12-06 06:49:03,332 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 9: Stage finished
2024-12-06 06:49:03,333 INFO scheduler.DAGScheduler: Job 8 finished: showString at NativeMethodAccessorImpl.java:0, took 0.287017 s
2024-12-06 06:49:03,363 INFO codegen.CodeGenerator: Code generated in 21.924467 ms
+---+---+---------+------------+
| id| id|  Borough|service_zone|
+---+---+---------+------------+
| 82|196|   Queens|   Boro Zone|
|  7|  7|   Queens|   Boro Zone|
|  7|  7|   Queens|   Boro Zone|
|166| 74|Manhattan|   Boro Zone|
|236|229|Manhattan| Yellow Zone|
|260|160|   Queens|   Boro Zone|
|244| 41|Manhattan|   Boro Zone|
| 83|  7|   Queens|   Boro Zone|
|223|223|   Queens|   Boro Zone|
|260|260|   Queens|   Boro Zone|
+---+---+---------+------------+
only showing top 10 rows

2024-12-06 06:49:03,388 INFO server.AbstractConnector: Stopped Spark@3c96239e{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2024-12-06 06:49:03,391 INFO ui.SparkUI: Stopped Spark web UI at http://zzzzzzzzzzzzzfinaltask3-p4-1-0c46d9939ab9ec99-driver-svc.data-science-ec22954.svc:4040
2024-12-06 06:49:03,398 INFO k8s.KubernetesClusterSchedulerBackend: Shutting down all executors
2024-12-06 06:49:03,399 INFO k8s.KubernetesClusterSchedulerBackend$KubernetesDriverEndpoint: Asking each executor to shut down
2024-12-06 06:49:03,412 WARN k8s.ExecutorPodsWatchSnapshotSource: Kubernetes client has been closed (this is expected if the application is shutting down.)
2024-12-06 06:49:04,232 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
```

Q5)

Here I used another very useful method in order to calculate the shortest path (no. of vertices) to reach a target destination. These target destination or target vertices are called landmark locations. I then made a table to illustrate theses result:

```
2024-12-06 06:55:48,818 INFO scheduler.DAGScheduler: ResultStage 64 (showString at NativeMethodAccessorImpl.java:0) finished in 0.160 s
2024-12-06 06:55:48,818 INFO scheduler.DAGScheduler: Job 14 is finished. Cancelling potential speculative or zombie tasks for this job
2024-12-06 06:55:48,818 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 64: Stage finished
2024-12-06 06:55:48,819 INFO scheduler.DAGScheduler: Job 14 finished: showString at NativeMethodAccessorImpl.java:0, took 0.167870 s
2024-12-06 06:55:48,838 INFO codegen.CodeGenerator: Code generated in 15.292641 ms
+-------+-----------------+
|id_to_1|shortest_distance|
+-------+-----------------+
|147->1 |2                |
|19->1  |2                |
|39->1  |2                |
|71->1  |2                |
|180->1 |2                |
|130->1 |1                |
|66->1  |1                |
|138->1 |2                |
|171->1 |2                |
|170->1 |2                |
+-------+-----------------+
only showing top 10 rows

2024-12-06 06:55:48,856 INFO server.AbstractConnector: Stopped Spark@70fa8e23{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2024-12-06 06:55:48,859 INFO ui.SparkUI: Stopped Spark web UI at http://zzzzzzzzzzzzzfinaltask3-p5-1-9e454e939abff184-driver-svc.data-science-ec22954.svc:4040
2024-12-06 06:55:48,869 INFO k8s.KubernetesClusterSchedulerBackend: Shutting down all executors
2024-12-06 06:55:48,870 INFO k8s.KubernetesClusterSchedulerBackend$KubernetesDriverEndpoint: Asking each executor to shut down
2024-12-06 06:55:48,882 WARN k8s.ExecutorPodsWatchSnapshotSource: Kubernetes client has been closed (this is expected if the application is shutting down.)
```

Once again, it was quite instructive though challenging working with vertices and edges within data, to make things even as fundamental as finding the shortest path from a to b.

Q6)

This for a while forced me to read through pyspark documentation. However I was able to understand what it was that was being asked of me in this question. pageRank() is an important function within graphframes bcause they help to calculate the importance of vertices in a graph. The restProbability represents the probability of randomly jumping to another vertex instead of following the link, tol (tolerence) serves as a threshold for convergence and puts an end to the page ranking one the difference between iterations are smaller than the specified tol given. Here is the output for my program:

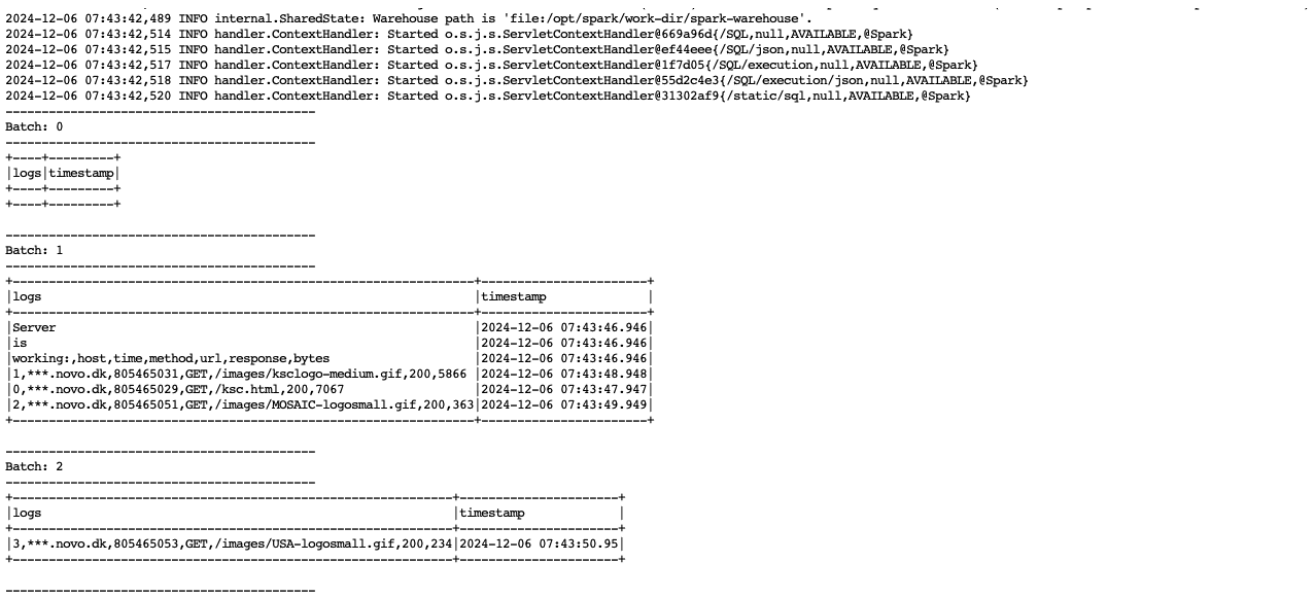
```
2024-12-06 06:58:19,129 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 2090: Stage finished
2024-12-06 06:58:19,130 INFO scheduler.DAGScheduler: Job 48 finished: showString at NativeMethodAccessorImpl.java:0, took 0.295073 s
2024-12-06 06:58:19,161 INFO codegen.CodeGenerator: Code generated in 24.417212 ms
2024-12-06 06:58:19,184 INFO codegen.CodeGenerator: Code generated in 16.855163 ms
+---+------------------+
|id |pagerank          |
+---+------------------+
|264|23.733400016313745|
|138|7.780849414226171 |
|132|7.195856951107596 |
|55 |4.471856828199626 |
|265|4.10645225541102  |
+---+------------------+
only showing top 5 rows

2024-12-06 06:58:19,204 INFO server.AbstractConnector: Stopped Spark@6773f68e{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2024-12-06 06:58:19,207 INFO ui.SparkUI: Stopped Spark web UI at http://zzzzzzzzzzzzzfinaltask3-p6-1-78562b939ac20a58-driver-svc.data-science-ec22954.svc:4040
2024-12-06 06:58:19,215 INFO k8s.KubernetesClusterSchedulerBackend: Shutting down all executors
2024-12-06 06:58:19,216 INFO k8s.KubernetesClusterSchedulerBackend$KubernetesDriverEndpoint: Asking each executor to shut down
2024-12-06 06:58:19,229 WARN k8s.ExecutorPodsWatchSnapshotSource: Kubernetes client has been closed (this is expected if the application is shutting down.)
```

TASK 4:

The screenshots taken here do not have running time stamps since they are out of frame. I will include some at the end.

Q1) Here we apply the host and the port specifier to the session. This way we are able to set up a stable connection to the streaming data source. It is an essential step for stream data processing .



Q2)

In this question I implemented the use of the function Watermark().

Watermark is a mechanism that handles data that arrives late. Using the code written as an example with the following code:

logsDF = logsDF.withWatermark("timestamp", "3 seconds")

in this case what is taking place is that the logsDF dataframe with basis on the time stamp will allow late arriving data to bee processed but with a cap of 3 seconds late. This means that any data that arrives later than 3 seconds will not be taken in. It is a good way to manage late data whilst also limiting the waiting period on spark. There are many different uses for it which I have been reading about including: financial transactions, LoT streams and other applications that rely on event time.

Q3)

Here some further components were added, this includes the watermark. This was explained in the section above, but to hammer the nail, the watermark is certainly an astonishing tool, by handling late incoming data, it maintains a consistency and accuracy that certainly make a difference in the results.

```
-------------------------------------------
Batch: 19
-------------------------------------------
+--------------------------------------------------------------+-----------------------+---+----------+--------+------+--------------------------+------------+-----+
|logs                                                          |timestamp              |idx|hostname  |time    |method|resource                  |responsecode|bytes|
+--------------------------------------------------------------+-----------------------+---+----------+--------+------+--------------------------+------------+-----+
|20,***.novo.dk,807951864,GET,/images/NASA-logosmall.gif,200,786|2024-12-06 07:14:58.451|20 |***.novo.dk|807951864|GET   |/images/NASA-logosmall.gif|200         |786  |
+--------------------------------------------------------------+-----------------------+---+----------+--------+------+--------------------------+------------+-----+

-------------------------------------------
Batch: 20
-------------------------------------------
+--------------------------------------------------------------+-----------------------+---+----------+--------+------+-----------------+------------+-----+
|logs                                                          |timestamp              |idx|hostname  |time    |method|resource         |responsecode|bytes|
+--------------------------------------------------------------+-----------------------+---+----------+--------+------+-----------------+------------+-----+
|21,***.novo.dk,807951907,GET,/htbin/cdt_main.pl,200,3714      |2024-12-06 07:14:59.452|21 |***.novo.dk|807951907|GET   |/htbin/cdt_main.pl|200         |3714 |
+--------------------------------------------------------------+-----------------------+---+----------+--------+------+-----------------+------------+-----+

-------------------------------------------
```

Q4)

I have set up a window time and a slide time near the top of the python page, these are essential for configuring the behaviour of the window() function. The window function allows the grouping of streaming data into time-based windows. This way we can view and analyse patterns over specified intervals

```
Batch: 1
-------------------------------------------
+------------------------------------------+---------+
|window                                    |gif_count|
+------------------------------------------+---------+
|[2024-12-06 07:15:30, 2024-12-06 07:16:30]|9        |
|[2024-12-06 07:15:00, 2024-12-06 07:16:00]|9        |
+------------------------------------------+---------+

-------------------------------------------
Batch: 2
-------------------------------------------
+------------------------------------------+---------+
|window                                    |gif_count|
+------------------------------------------+---------+
|[2024-12-06 07:15:30, 2024-12-06 07:16:30]|16       |
|[2024-12-06 07:15:00, 2024-12-06 07:16:00]|16       |
+------------------------------------------+---------+

-------------------------------------------
```

Q5)

Once again the window() function is used, in this case it is used to group the streaming data based on hostnames and time intervals together with the total_bytes. In this occasion the total_bytes are calculated by the addition of the bytes column for each unique hostname instance and within a specified times period which was set to 30 seconds.

On top of this dropna() was utilised as a way to clean up the data so that it may resemble the examples provided. The original data consisted of some nulls. This improves quality and reliability of the input.

```
-------------------------------------------
Batch: 4
-------------------------------------------
+-----------------------------------------+-----------------------+-----------+
|window                                   |hostname               |total_bytes|
+-----------------------------------------+-----------------------+-----------+
|[2024-12-06 07:28:00, 2024-12-06 07:29:00]|001.msy4.communique.net|44832.0    |
|[2024-12-06 07:27:30, 2024-12-06 07:28:30]|***.novo.dk            |247646.0   |
|[2024-12-06 07:27:00, 2024-12-06 07:28:00]|001.msy4.communique.net|9630.0     |
|[2024-12-06 07:27:00, 2024-12-06 07:28:00]|***.novo.dk            |247646.0   |
|[2024-12-06 07:27:30, 2024-12-06 07:28:30]|001.msy4.communique.net|54462.0    |
+-----------------------------------------+-----------------------+-----------+

-------------------------------------------
```

Q6)

The trigger functionality is utilised here, it plays a critical role when working with streaming data. It determines the frequency at which batch data is processed, and through doing this it massively increases it's efficiency by decreasing things like latency and throughput.

In the code we specify a trigger interval of 10 seconds. This means that the spark program will process batches of data at 10 second intervals. It means a consistent flow of incoming data which proves to be rather useful.

```
-------------------------------------------
Batch: 6
-------------------------------------------
+----------------------+-------------+
|window                |Correct_count|
+----------------------+-------------+
|007.thegap.com        |19           |
|001.msy4.communique.net|8           |
|***.novo.dk           |29           |
+----------------------+-------------+

-------------------------------------------
```

Some running times stamps for question 4:

```
, task resources: Map(cpus -> name: cpus, amount: 1.0)
2024-12-06 07:34:37,784 INFO k8s.KubernetesClusterSchedulerBackend$KubernetesDriverEndpoint: Registered executor NettyRpcEndpointRef(spark-client://Executor) (10.134.94.30:38856) with ID 1
2024-12-06 07:34:37,889 INFO storage.BlockManagerMasterEndpoint: Registering block manager 10.134.94.30:35473 with 2.1 GiB RAM, BlockManagerId(1, 10.134.94.30, 35473, None)
2024-12-06 07:34:38,050 INFO k8s.KubernetesClusterSchedulerBackend$KubernetesDriverEndpoint: Registered executor NettyRpcEndpointRef(spark-client://Executor) (10.132.68.187:42038) with ID 2
2024-12-06 07:34:38,122 INFO k8s.KubernetesClusterSchedulerBackend: SchedulerBackend is ready for scheduling beginning after reached minRegisteredResourcesRatio: 0.8
2024-12-06 07:34:38,164 INFO storage.BlockManagerMasterEndpoint: Registering block manager 10.132.68.187:39003 with 2.1 GiB RAM, BlockManagerId(2, 10.132.68.187, 39003, None)
2024-12-06 07:34:38,377 INFO internal.SharedState: Setting hive.metastore.warehouse.dir ('null') to the value of spark.sql.warehouse.dir ('file:/opt/spark/work-dir/spark-warehouse').
2024-12-06 07:34:38,377 INFO internal.SharedState: Warehouse path is 'file:/opt/spark/work-dir/spark-warehouse'.
2024-12-06 07:34:38,398 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@79f63aa8{/SQL,null,AVAILABLE,@Spark}
2024-12-06 07:34:38,399 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@5a66680d{/SQL/json,null,AVAILABLE,@Spark}
2024-12-06 07:34:38,399 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@714ae399{/SQL/execution,null,AVAILABLE,@Spark}
2024-12-06 07:34:38,400 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@17a3092{/SQL/execution/json,null,AVAILABLE,@Spark}
2024-12-06 07:34:38,402 INFO handler.ContextHandler: Started o.s.j.s.ServletContextHandler@2d178660{/static/sql,null,AVAILABLE,@Spark}
-------------------------------------------
Batch: 0
-------------------------------------------
+----+---------+---+--------+----+------+--------+------------+-----+
|logs|timestamp|idx|hostname|time|method|resource|responsecode|bytes|
+----+---------+---+--------+----+------+--------+------------+-----+
+----+---------+---+--------+----+------+--------+------------+-----+

-------------------------------------------
Batch: 1
-------------------------------------------
+--------------------------------------------------------------+--------------------+--------+-----------+---------+------+----------------------------+------------+-----+
|logs                                                          |timestamp           |idx     |hostname   |time     |method|resource                    |responsecode|bytes|
+--------------------------------------------------------------+--------------------+--------+-----------+---------+------+----------------------------+------------+-----+
|Server                                                        |2024-12-06 07:34:42.921|Server  |null       |null     |null  |null                        |null        |null |
|is                                                            |2024-12-06 07:34:42.921|is      |null       |null     |null  |null                        |null        |null |
|working:,host,time,method,url,response,bytes                  |2024-12-06 07:34:42.921|working:|host       |time     |method|url                         |response    |bytes|
|1,***.novo.dk,805465031,GET,/images/ksclogo-medium.gif,200,5866|2024-12-06 07:34:44.922|1       |***.novo.dk|805465031|GET   |/images/ksclogo-medium.gif  |200         |5866 |
|0,***.novo.dk,805465029,GET,/ksc.html,200,7067                |2024-12-06 07:34:43.921|0       |***.novo.dk|805465029|GET   |/ksc.html                   |200         |7067 |
|2,***.novo.dk,805465051,GET,/images/MOSAIC-logosmall.gif,200,363|2024-12-06 07:34:45.923|2       |***.novo.dk|805465051|GET   |/images/MOSAIC-logosmall.gif|200         |363  |
+--------------------------------------------------------------+--------------------+--------+-----------+---------+------+----------------------------+------------+-----+

-------------------------------------------
```

```
-------------------------------------------
Batch: 3
-------------------------------------------
+------------------------------------------+---------+
|window                                    |gif_count|
+------------------------------------------+---------+
|[2024-12-06 07:35:30, 2024-12-06 07:36:30]|21       |
|[2024-12-06 07:36:00, 2024-12-06 07:37:00]|9        |
+------------------------------------------+---------+

-------------------------------------------
Batch: 4
-------------------------------------------
+------------------------------------------+---------+
|window                                    |gif_count|
+------------------------------------------+---------+
|[2024-12-06 07:35:30, 2024-12-06 07:36:30]|30       |
|[2024-12-06 07:36:00, 2024-12-06 07:37:00]|18       |
+------------------------------------------+---------+

2024-12-06 07:36:43,040 ERROR v2.WriteToDataSourceV2Exec: Data source write support org.apache.spark.sql.execution.streaming.sources.MicroBatchWrite@24d53c07 is abort
2024-12-06 07:36:43,041 ERROR v2.WriteToDataSourceV2Exec: Data source write support org.apache.spark.sql.execution.streaming.sources.MicroBatchWrite@24d53c07 aborted.
-------------------------------------------



-------------------------------------------
Batch: 8
-------------------------------------------
+------------------------------+-------------+
|window                        |Correct_count|
+------------------------------+-------------+
|007.thegap.com                |34           |
|001.msy4.communique.net       |8            |
|***.novo.dk                   |29           |
|01-dynamic-c.wokingham.luna.net|5            |
+------------------------------+-------------+

-------------------------------------------
Batch: 9
-------------------------------------------
+------------------------------+-------------+
|window                        |Correct_count|
+------------------------------+-------------+
|007.thegap.com                |34           |
|001.msy4.communique.net       |8            |
|***.novo.dk                   |29           |
|01-dynamic-c.wokingham.luna.net|15           |
+------------------------------+-------------+

2024-12-06 07:39:23,258 ERROR v2.WriteToDataSourceV2Exec: Data source write support org.apache.spark.sql.execution.streaming.sources.MicroBatchWrite@16ba981a is aborti
2024-12-06 07:39:23,258 ERROR v2.WriteToDataSourceV2Exec: Data source write support org.apache.spark.sql.execution.streaming.sources.MicroBatchWrite@16ba981a aborted.
```