ECS765P- Big Data Processing

Coursework: NYC Rideshare Analysis

Name: Addalin Lariena Rose Devasahaya Rajan

Student ID: 230187746

Common APIs:

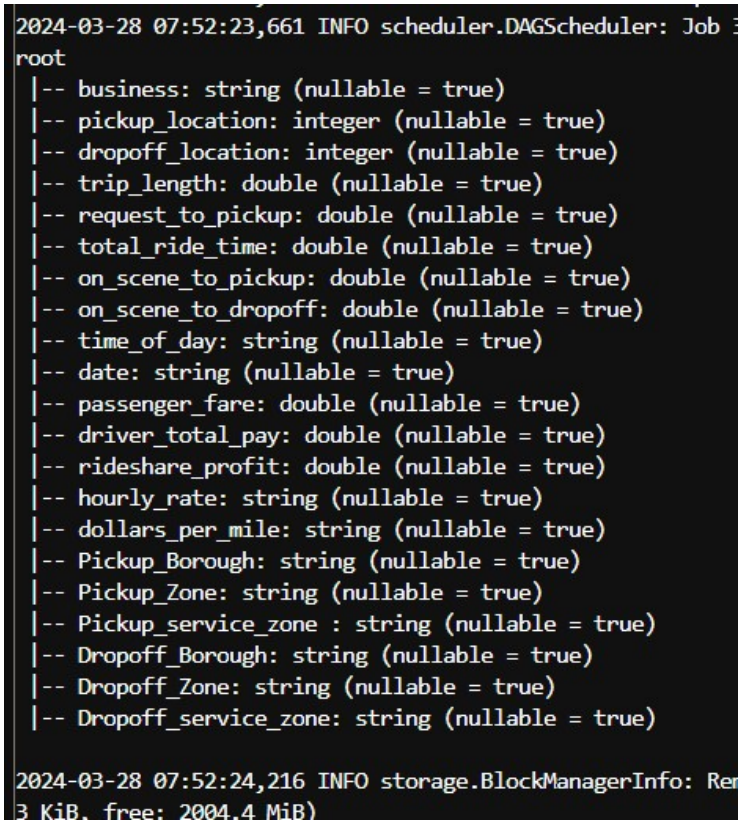
- **os** : This module provides to interact with the Operating System. In all of the codes, this module is used to set up configurations to access the files stored in the Amazon S3 bucket which prevents from hardcoding the credentials directly in the python files.
- **SparkSession**: This class is part of the Apache Spark's core API that allows to work with structured data and execute data processing tasks in a distributed manner. In all of the codes, it is used to perform spark operations such as reading, processing and writing data to storage.
- **from_unixtime**: This function is provided by Spark SQL. It is used to convert Unix timestamp values to date-time format that makes sense to us. In all of the codes, this is used to convert the Unix timestamp value of the column 'date' of the joined DataFrame into date value.
- **Col**: This is a utility function provided by Spark SQL. It is used to reference or create columns in the DataFrames.
- **Month**: This function is a part of pyspark.sql.functions module. Where the module itself gives various built-in functions for manipulation and transformation of the DataFrames. In all of the codes, this function is used for extracting month or day component from the converted 'date' column.
- **Concat**: Like the month function, concat is also a part of the pyspark.sql.functions module. It is used to concat the strings from multiple columns to one string column. In most of the codes, it is used to concat newly created column 'month' and 'business' column.
- **FloatType**: This datatype is part of the pyspark.sql.types module. It represents floating point numbers. In most of the code, it is used to convert the string datatype to float datatype to perform numerical operations.

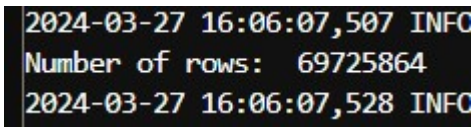
Task 1

Explanation:

1. Loaded the csv files from the shared buckets using SparkSession.
2. Applied join on rideshare and taxi zone lookup tables firstly on pickup location, new columns were renamed according to the task. Then the intermediate result was joined with taxi zone lookup based on dropoff location and again columns were renamed.
3. The 'date' column was converted to date format using 'from_unixtime' function and 'col' was also used to reference the 'date' column.
4. Printed schema using printSchema function and printed number of rows using count function

Visualization:





Challenges:

- Learning curve of using spark operations on the dataframe was challenging. But by researching on the spark operations and spark API, I was able to understand their operation and where to use them.

Knowledge Gained:

- Understanding Spark operations and Spark API
- Knowing how to merge datasets

Task 2

APIs Used Here:

- Lit: It is a function provided by pyspark.sql.functions. It is used to create new column with a constant value across.
- Count: It is an aggregate function provided by pyspark.sql.functions. It counts the non-null values in a dataframe or a within a group.
- StructType: It is a class provided by pyspark.sql.types. By specifying the names and datatypes of the columns, it is used to create a structure of the dataframe.
- StructField: It is a class provided by pyspark.sql.types. Used with StructType, it defines the column in the schema.
- StringType: It is a datatype provided by pyspark.sql.types. It is used to specify a column datatype as String.

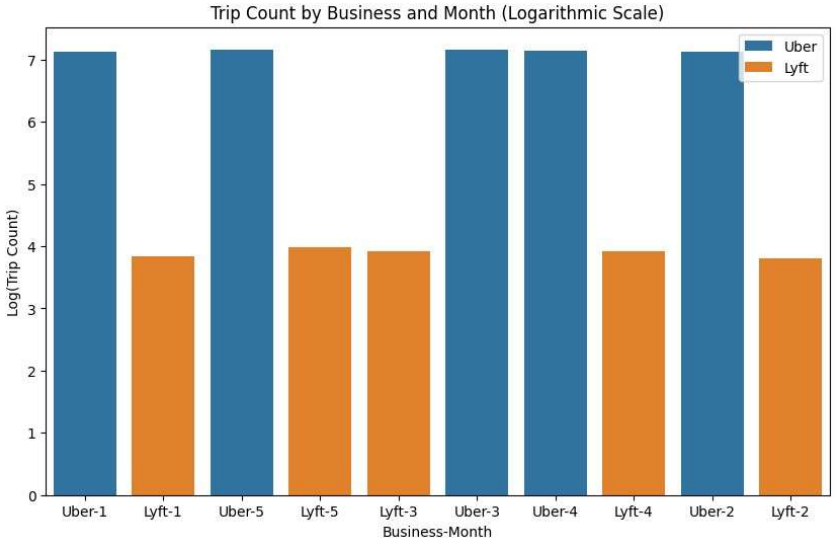
Explanation:

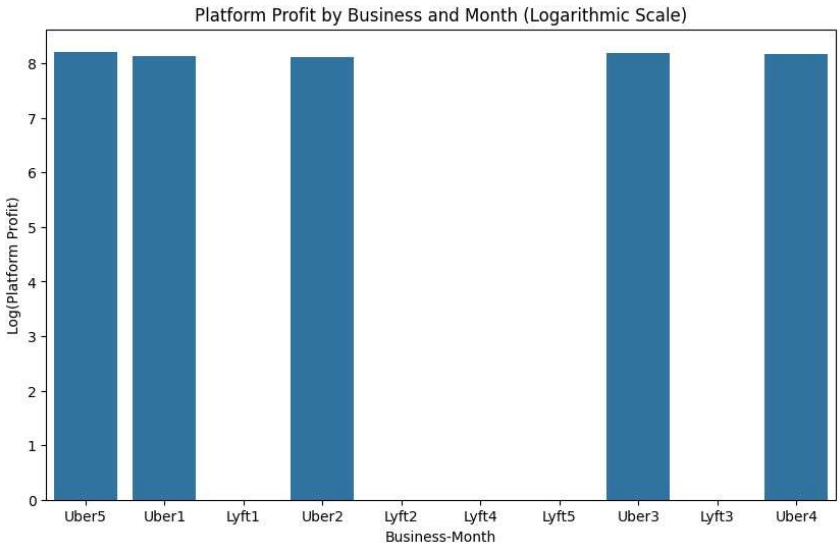
- Part 1:
    - Selected the columns only needed for this part to simplify processing and reduce memory usage
    - Created a new dataframe by building a new schema using StructType class
    - Processing trip count by looping into each month, grouping by business and counting by using count function, and finally storing them in the new dataframe
    - Ordering the results by month, repartitioning it to a single file and writing it as csv file to the shared bucket
    - Using Pandas, matplotlib, numpy and seaborn libraries the results were plotted into histogram
- Part 2:
    - Selected the columns only needed for this part to simplify processing and reduce memory usage
    - Extracting month values from ‘date’ column to a new column called ‘month’
    - 'rideshare_profit' field is casted to FloatType for further processing
    - Concating ‘business’ and ‘month’ column values to a new column called ‘business_month’
    - Adding the platform profits ('rideshare_profit' values) grouped by ‘business_month’ column using count function
    - Repartitioning the resulting dataframe to a single file and writing it as csv file to the shared bucket
    - Using Pandas, matplotlib, numpy and seaborn libraries the results were plotted into histogram
- Part 3:
    - Selected the columns only needed for this part to simplify processing and reduce memory usage
    - Extracting month values from ‘date’ column to a new column called ‘month’
    - 'driver_total_pay' field is casted to FloatType for further processing
    - Concating ‘business’ and ‘month’ column values to a new column called ‘business_month’
    - Adding the drivers earnings (‘driver_total_pay’ values) grouped by ‘business_month’ column

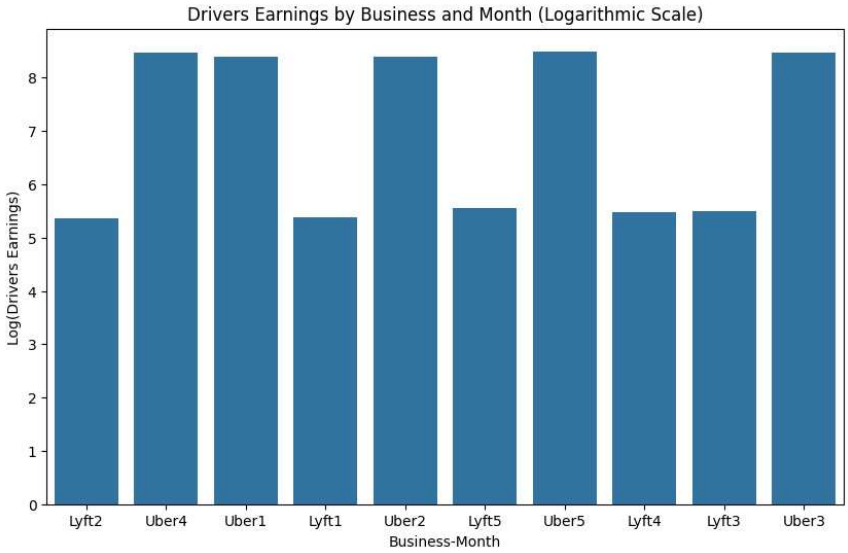
- Repartitioning the resulting dataframe to a single file and writing it as csv file to the shared bucket
- Using Pandas, matplotlib, numpy and seaborn libraries the results were plotted into histogram

- Part 4:
  - From the trip count:
    - It is noticeable that Uber has higher trip counts than Lyft in general
    - For both business the trend of trip count increases overall
    - The highest trip count value is for the month of May
    - From the results, as a driver I will try to maximise my earnings in month of May and as the CEO I can do more marketing or make informed decisions on the resource allocation problems.
  - From platform profits:
    - Uber has higher platform profits overall compared to Lyft
    - For both the businesses the platform profit increases towards May
    - Lyft shows negative profit in all months except May
    - From the results, as a stockbroker I can use this to guide my investment decisions by investing into Uber and as a CEO by looking into the company's finance I can make decisions for better cost management and better strategies in pricing
  - From drivers earnings:
    - Uber drivers earn higher compared to Lyft drivers overall
    - There is an increase of earnings towards May for both the businesses
    - The most profitable month for the drivers is in May
    - From the results, as a driver I can apply in Uber due to higher earnings compared to Lyft and as a CEO monitoring the drivers earnings can provide insights on the drivers satisfaction and retention rates

Visualization:







Challenges:

- Visualizing values with large variations between them in the histogram. Due to the large variation in the big and small values, the histogram did not showcase all the values properly. To rectify this I used logarithmic scaling to compress the large values for better visualization for better analysation and interpretation of the graph.

Knowledge Gained:

- Computing sum and total count by using Spark
- Writing results in a csv file
- Scaling extremely large values for better visualization
- Creating new dataframe from schema
- Knowing how to aggregate data

Task 3

APIs Used Here:

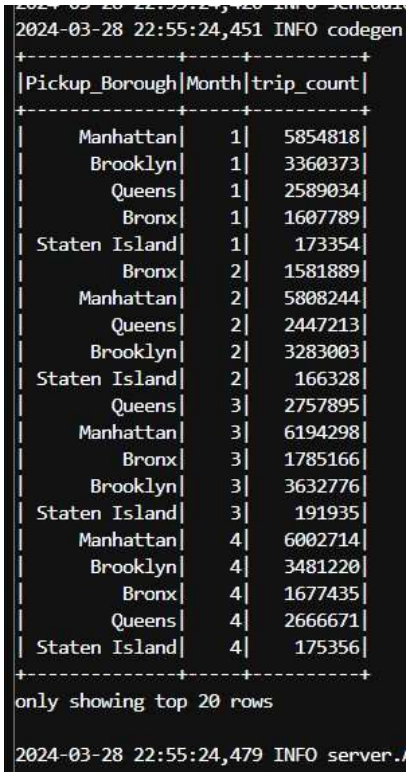
- Lit: It is a function provided by pyspark.sql.functions. It is used to create new column with a constant value across.
- Count: It is an aggregate function provided by pyspark.sql.functions. It counts the non-null values in a dataframe or a within a group.
- Sum: It is an aggregate function provided by pyspark.sql.functions. It aggregates values in a dataframe column or a within a group.
- StructType: It is a class provided by pyspark.sql.types. By specifying the names and datatypes of the columns, it is used to create a structure of the dataframe. StructField: It is a class provided by pyspark.sql.types. Used with StructType, it defines the column in the schema.
- StringType: It is a datatype provided by pyspark.sql.types. It is used to specify a column datatype as String.

Explanation:

- Part 1:
    - Selected the columns only needed for this part to simplify processing and reduce memory usage
    - Created a new dataframe by building a new schema using StructType class
    - Processing trip count by looping into each month, grouping by pickup borough, counting by using count function and taking only top 5 by descending order, and finally storing them in the new dataframe
    - Ordering the dataframe by month and printing it
- Part 2:
    - Selected the columns only needed for this part to simplify processing and reduce memory usage
    - Created a new dataframe by building a new schema using StructType class
    - Processing trip count by looping into each month, grouping by dropoff borough, counting by using count function and taking only top 5 by descending order, and finally storing them in the new dataframe
    - Ordering the dataframe by month and printing it
- Part 3:
    - Selected the columns only needed for this part to simplify processing and reduce memory usage
    - Adding the total_profit (‘driver_total_pay’ values) grouped by ‘Pickup_Borough’ and ‘Dropoff_Borough’ columns
    - Creating new column ‘Route’ by concatenating Pickup_Borough and Dropoff_Borough columns
    - Dropped Pickup_Borough and Dropoff_Borough columns
    - Ordering dataframe by total_profit in descending order and taking only top 30
    - Printing the resulting dataframe
- Part 4:
    - Manhattan has the highest trip count for pickup and dropoff borough overall
    - Manhattan to Manhattan route has the highest total profit
    - As a driver, since there is more demand and profit for the Manhattan route I will reschedule to align most of my trips this route.

- As CEO, by looking into this resources can be allocated to Manhattan route to meet the demands and also amp up the marketing efforts into this route

Visualization:

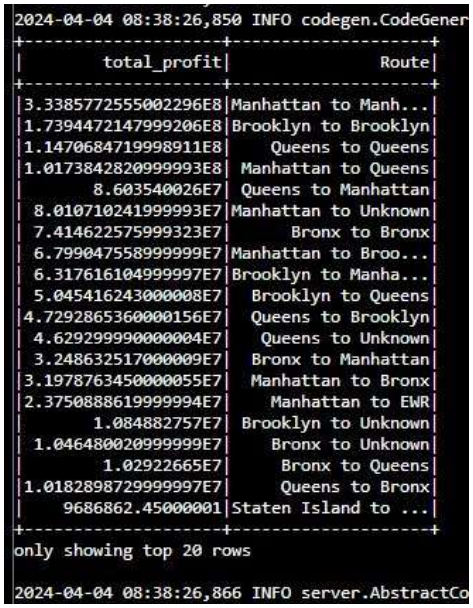


```
2024-03-28 22:55:24,451 INFO codegen
+--------------+-----+----------+
|Pickup_Borough|Month|trip_count|
+--------------+-----+----------+
|     Manhattan|    1|   5854818|
|      Brooklyn|    1|   3360373|
|        Queens|    1|   2589034|
|         Bronx|    1|   1607789|
| Staten Island|    1|    173354|
|         Bronx|    2|   1581889|
|     Manhattan|    2|   5808244|
|        Queens|    2|   2447213|
|      Brooklyn|    2|   3283003|
| Staten Island|    2|    166328|
|        Queens|    3|   2757895|
|     Manhattan|    3|   6194298|
|         Bronx|    3|   1785166|
|      Brooklyn|    3|   3632776|
| Staten Island|    3|    191935|
|     Manhattan|    4|   6002714|
|      Brooklyn|    4|   3481220|
|         Bronx|    4|   1677435|
|        Queens|    4|   2666671|
| Staten Island|    4|    175356|
+--------------+-----+----------+
only showing top 20 rows

2024-03-28 22:55:24,479 INFO server.
```

```
2024-03-28 23:10:51,708 INFO codegen.C
+---------------+-----+----------+
|Dropoff_Borough|Month|trip_count|
+---------------+-----+----------+
|        Queens|    1|   2480080|
|     Manhattan|    1|   5444345|
|         Bronx|    1|   1525137|
|      Brooklyn|    1|   3337415|
|       Unknown|    1|    535610|
|         Bronx|    2|   1511014|
|        Queens|    2|   2390783|
|     Manhattan|    2|   5381696|
|      Brooklyn|    2|   3251795|
|       Unknown|    2|    497525|
|     Manhattan|    3|   5671301|
|        Queens|    3|   2713748|
|       Unknown|    3|    566798|
|      Brooklyn|    3|   3608960|
|         Bronx|    3|   1706802|
|     Manhattan|    4|   5530417|
|      Brooklyn|    4|   3448225|
|         Bronx|    4|   1596505|
|        Queens|    4|   2605086|
|       Unknown|    4|    551857|
+---------------+-----+----------+
only showing top 20 rows

2024-03-28 23:10:51,726 INFO server.Ab
```



```
2024-04-04 08:38:26,850 INFO codegen.CodeGener
+--------------------+--------------------+
|        total_profit|               Route|
+--------------------+--------------------+
|3.3385772555002296E8|Manhattan to Manh...|
|1.7394472147999206E8|Brooklyn to Brooklyn|
|1.1470684719998911E8|    Queens to Queens|
|1.0173842820999993E8| Manhattan to Queens|
|       8.603540026E7| Queens to Manhattan|
| 8.010710241999993E7|Manhattan to Unknown|
| 7.414622575999323E7|      Bronx to Bronx|
| 6.799047558999999E7|Manhattan to Broo...|
| 6.317616104999997E7|Brooklyn to Manha...|
| 5.045416243000008E7|  Brooklyn to Queens|
|4.7292865360000156E7|  Queens to Brooklyn|
| 4.629299990000004E7|   Queens to Unknown|
| 3.248632517000009E7|  Bronx to Manhattan|
|3.1978763450000055E7|  Manhattan to Bronx|
|2.3750888619999994E7|    Manhattan to EWR|
|       1.084882757E7| Brooklyn to Unknown|
| 1.046480020999999E7|    Bronx to Unknown|
|        1.02922665E7|     Bronx to Queens|
|1.0182898729999997E7|     Queens to Bronx|
|    9686862.45000001|Staten Island to ...|
+--------------------+--------------------+
only showing top 20 rows

2024-04-04 08:38:26,866 INFO server.AbstractCo
```

Challenges:

No challenges were faced here

Knowledge Gained:

- Able to concatenate values of 2 columns into a new column
- Using spark to find the top-k data from the dataframe

Task 4

APIs Used Here:

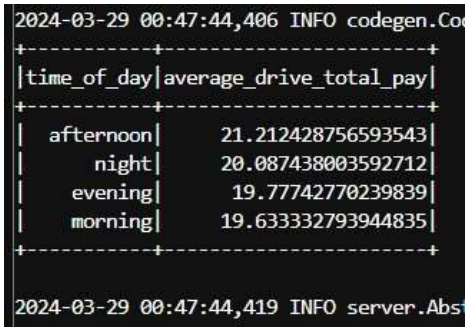
- avg: It is a function provided by pyspark.sql.functions. It calculates the mean of the values in a dataframe column.

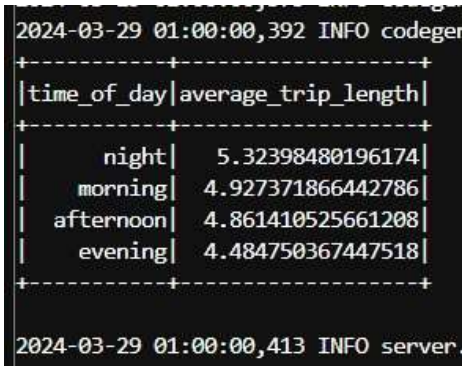
Explanation:

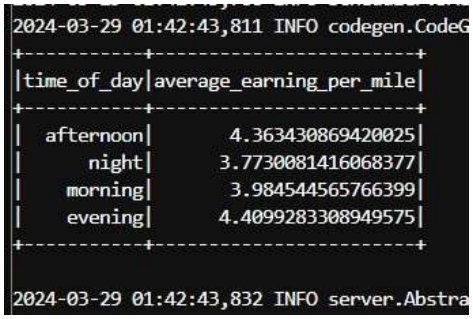
- Part 1:
  - Selected the columns only needed for this part to simplify processing and reduce memory usage
  - Filtering dataframe according to time of day into 4 new dataframes
  - List of tuples is made where each element has the time of day and the average of the total driver pay for that time of day from the filtered dataframes
  - A new dataframe is made with the list of tuples and is ordered by the average total driver pay
  - The resulting dataframe is printed
- Part 2:
  - Selected the columns only needed for this part to simplify processing and reduce memory usage
  - Filtering dataframe according to time of day into 4 new dataframes
  - List of tuples is made where each element has the time of day and the average of the trip length for that time of day from the filtered dataframes
  - A new dataframe is made with the list of tuples and is ordered by the average trip length
  - The resulting dataframe is printed
- Part 3:
  - Selected the columns only needed for this part to simplify processing and reduce memory usage
  - Filtering dataframe according to time of day into 4 new dataframes
  - List of tuples is made where each element has the time of day and the average of the trip length for that time of day from the filtered dataframes
  - A new dataframe is made with the list of tuples and is ordered by the average trip length
  - List of tuples is made where each element has the time of day and the average of the total driver pay for that time of day from the filtered dataframes
  - A new dataframe is made with the list of tuples and is ordered by the average total driver pay
  - The dataframes made for average of trip length and average of driver total pay is joined using inner join on the column time of day and a new column 'average_earning_per_mile' is created where the average driver total pay is divided by the average trip length
  - Unnecessary columns are dropped and the resulting dataframe is printed
- Part 4:
  - For average earnings and trip length:
    - Afternoon trips have the highest average earning per mile
    - Night trips have the longest trip length
  - For time of day and driver earnings

        - Drivers earn more per mile in afternoon trips
    - As a driver, I would align my schedule to more afternoon trips as it has more earnings and demand
    - As a CEO, I would ensure resources are allocated to afternoon trips and pricing strategies can be made to maximize night trips

Visualization:







Challenges:

No challenges were faced here

Knowledge Gained:

- Able to create dataframe from a list of tuples
- Using spark to join 2 dataframes and create a new column in the same line
- Able to provide average summaries on conditions

Task 5

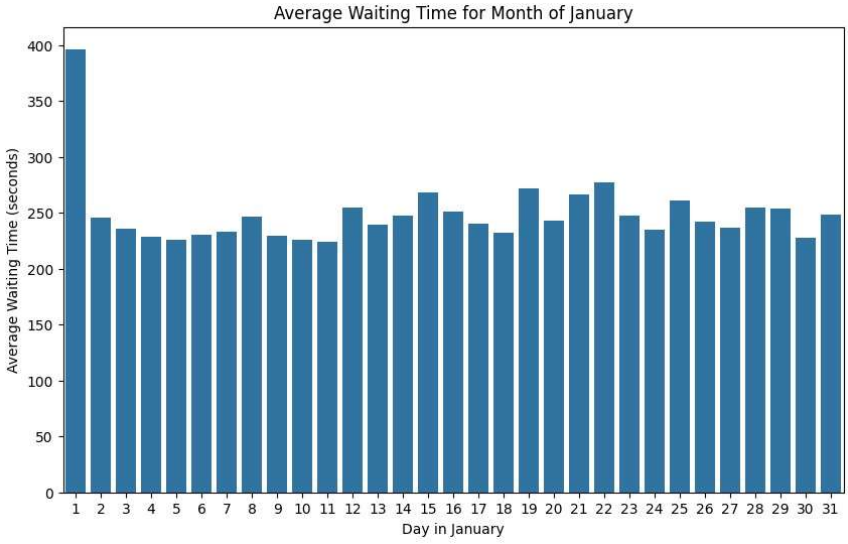
APIs Used Here:

- avg: It is a function provided by pyspark.sql.functions. It calculates the mean of the values in a dataframe column.
- Dayofmonth: It is a function provided by pyspark.sql.functions. It is used to extract the day of the month from columns that have DateType or TimestampType

Explanation:

- Part 1:
  - Selected the columns only needed for this part to simplify processing and reduce memory usage
  - Filtering dataframe according to month of January
  - Renamed 'date' column as 'day'
  - Average waiting time is calculated by averaging the 'request_to_pickup' column grouped by day into a new column 'average_waiting_time'
  - Dataframe is then ordered by day, repartitioned into a single file and written as csv file in the shared bucket
  - Using Pandas, matplotlib, numpy and seaborn libraries the results were plotted into histogram
- Part 2:
  - The average exceeds 300 seconds on the day 1
- Part 3:
  - It is noticeable that for day 1 it could be due to New Year's celebration

Visualization:



Challenges:

No challenges faced here

Knowledge Gained:

- Finding anomalies in a dataset using spark and visualization

Task 6

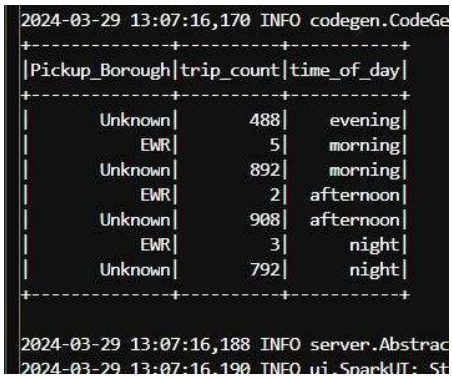
APIs Used Here:

- Lit: It is a function provided by pyspark.sql.functions. It is used to create new column with a constant value across.
- Count: It is an aggregate function provided by pyspark.sql.functions. It counts the non-null values in a dataframe or a within a group.

Explanation:

- Part 1:
  - Selected the columns only needed for this part to simplify processing and reduce memory usage
  - Filtering dataframe according to time of day into 4 new dataframes
  - counting trip count grouped by pickup borough for all 4 time of day in separate dataframes
  - joining all dataframes and filtering by trip count for more than 0 and less than 1000
  - printing the resulting dataframe
- Part 2:
  - Selected the columns only needed for this part to simplify processing and reduce memory usage
  - Filtering dataframe according to evening time of day
  - counting trip count grouped by pickup borough
  - printing the resulting dataframe
- Part 3:
  - Selected the columns only needed for this part to simplify processing and reduce memory usage
  - Filtering dataframe where Pickup_Borough is Brooklyn and Dropoff_Borough is Staten Island
  - printing the number of trips from Brooklyn to Staten Island and 10 rows of data from resulting dataframe

Visualization:



```
2024-03-29 13:07:16,170 INFO codegen.CodeGe
+-------------+----------+-----------+
|Pickup_Borough|trip_count|time_of_day|
+-------------+----------+-----------+
|      Unknown|       488|    evening|
|          EWR|         5|    morning|
|      Unknown|       892|    morning|
|          EWR|         2|  afternoon|
|      Unknown|       908|  afternoon|
|          EWR|         3|      night|
|      Unknown|       792|      night|
+-------------+----------+-----------+

2024-03-29 13:07:16,188 INFO server.Abstrac
2024-03-29 13:07:16,190 INFO ui.SparkUI: St
```

```
2024-03-29 12:38:18,266 INFO scheduler.DAGS
2024-03-29 12:38:18,295 INFO codegen.CodeGe
+--------------+----------+-----------+
|Pickup_Borough|trip_count|time_of_day|
+--------------+----------+-----------+
|        Queens|   2223003|    evening|
|       Unknown|       488|    evening|
|      Brooklyn|   3075616|    evening|
| Staten Island|    151276|    evening|
|     Manhattan|   5724796|    evening|
|         Bronx|   1380355|    evening|
+--------------+----------+-----------+

2024-03-29 12:38:18,312 INFO server.Abstract
2024-03-29 12:38:18,313 INFO ui.SparkUI: St
```

```
The number of trips from Brooklyn to Staten Island is  69437
```

```
2024-03-29 13:16:55,869 INFO codegen.CodeGenerator: Code
+--------------+--------------+--------------------+
|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|
+--------------+--------------+--------------------+
only showing top 10 rows

2024-03-29 13:16:55,892 INFO server.AbstractConnector: St
```

Challenges:

No challenges faced here

Knowledge Gained:

- To filter dataframe using one or more conditions

Task 7

APIs Used Here:

- Lit: It is a function provided by pyspark.sql.functions. It is used to create new column with a constant value across.
- Count: It is an aggregate function provided by pyspark.sql.functions. It counts the non-null values in a dataframe or a within a group.

Explanation:

- Selected the columns only needed for this part to simplify processing and reduce memory usage
- Aggregating the count of routes by ‘Pickup_Zone’, ‘Dropoff_Zone’, and business and Grouping the aggregated route counts by ‘Pickup_Zone’ and ‘Dropoff_Zone’
- Adding a new column 'total_count' that has the sum of counts for Uber and Lyft, ordering them in descending order and getting the top 10
- Adding new column ‘route’ by concatenating ‘Pickup_Zone’ and ‘Dropoff_Zone’ columns
- Unnecessary columns are dropped and resulting dataframe is printed

Visualization:

```
2024-04-04 14:05:27,158 INFO codegen.CodeGenerator: Code generated in 15.
+----+------+-----------+-----------------------------------------+
|Lyft|Uber  |total_count|Route                                    |
+----+------+-----------+-----------------------------------------+
|46  |253211|253257     |JFK Airport to NA                        |
|184 |202719|202903     |East New York to East New York           |
|78  |155803|155881     |Borough Park to Borough Park             |
|41  |151521|151562     |LaGuardia Airport to NA                  |
|26  |126253|126279     |Canarsie to Canarsie                     |
|1770|107392|109162     |South Ozone Park to JFK Airport          |
|100 |98591 |98691      |Crown Heights North to Crown Heights North|
|300 |98274 |98574      |Bay Ridge to Bay Ridge                   |
|75  |90692 |90767      |Astoria to Astoria                       |
|19  |89652 |89671      |Jackson Heights to Jackson Heights       |
+----+------+-----------+-----------------------------------------+

2024-04-04 14:05:27,174 INFO server.AbstractConnector: Stopped Spark@6f0
```

Challenges:

- Using Pivot to create columns for Uber and Lyft, I wanted to find a way to create columns for the businesses without extra step. So I had to learn and see what pivot does here.

Knowledge Gained:

- Usage of pivot for creating columns

Task 8

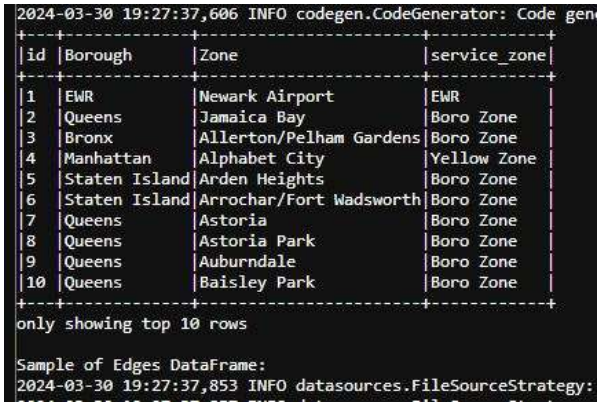
APIs Used Here:

- StructType: It is a class provided by pyspark.sql.types. By specifying the names and datatypes of the columns, it is used to create a structure of the dataframe.
- StructField: It is a class provided by pyspark.sql.types. Used with StructType, it defines the column in the schema.
- StringType: It is a datatype provided by pyspark.sql.types. It is used to specify a column datatype as String.
- Graph: It is provided by the graphframes. It allows for representing, manipulating and analysing graphs that are made from dataframes
- SQLContext: It is from pyspark.sql. It is used for working with structured data and executing SQL queries in Spark

Explanation:

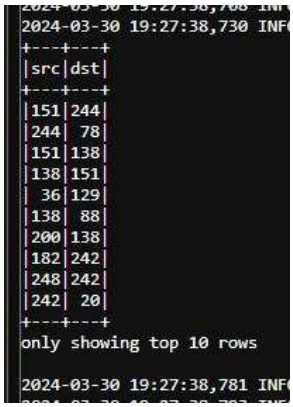
- Define the schema for vertices and edges in the graph
- Creating vertices and edges dataframes
- Creating graph using the vertices and edges dataframes
- Finding the vertices in the graph where the pickup and dropoff locations are in the same borough and service zone and printing the count of it
- Specific columns are selected from the vertices and 10 rows of data are printed

Visualization:



```
2024-03-30 19:27:37,606 INFO codegen.CodeGenerator: Code gen
+---+-------------+----------------------+------------+
|id |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   |
|6  |Staten Island|Arrochar/Fort Wadsworth|Boro Zone   |
|7  |Queens       |Astoria               |Boro Zone   |
|8  |Queens       |Astoria Park          |Boro Zone   |
|9  |Queens       |Auburndale            |Boro Zone   |
|10 |Queens       |Baisley Park          |Boro Zone   |
+---+-------------+----------------------+------------+
only showing top 10 rows

Sample of Edges DataFrame:
2024-03-30 19:27:37,853 INFO datasources.FileSourceStrategy:
```



```
2024-03-30 19:27:38,730 INF
+---+---+
|src|dst|
+---+---+
|151|244|
|244| 78|
|151|138|
|138|151|
| 36|129|
|138| 88|
|200|138|
|182|242|
|248|242|
|242| 20|
+---+---+
only showing top 10 rows

2024-03-30 19:27:38,781 INF
```

```
2024-04-01 11:00:45,214 INFO codegen
+---+---+---------+------------+
|id |id |Borough  |service_zone|
+---+---+---------+------------+
|182|242|Bronx    |Boro Zone   |
|248|242|Bronx    |Boro Zone   |
|242|20 |Bronx    |Boro Zone   |
|20 |20 |Bronx    |Boro Zone   |
|236|262|Manhattan|Yellow Zone |
|262|170|Manhattan|Yellow Zone |
|239|239|Manhattan|Yellow Zone |
|94 |69 |Bronx    |Boro Zone   |
|47 |247|Bronx    |Boro Zone   |
|76 |76 |Brooklyn |Boro Zone   |
+---+---+---------+------------+
only showing top 10 rows

2024-04-01 11:00:45,275 INFO spark.S
```

```
2024-03-30 19:29:39,527 INFO codegen.CodeGenerator: Code generated in 10.674354 ms
+--------------------------------------------------------+---------+-----------------------------------------------------+
|src                                                     |edge     |dst                                                  |
+--------------------------------------------------------+---------+-----------------------------------------------------+
|[151, Manhattan, Manhattan Valley, Yellow Zone]         |[151, 244]|[244, Manhattan, Washington Heights South, Boro Zone]|
|[244, Manhattan, Washington Heights South, Boro Zone]   |[244, 78] |[78, Bronx, East Tremont, Boro Zone]                 |
|[151, Manhattan, Manhattan Valley, Yellow Zone]         |[151, 138]|[138, Queens, LaGuardia Airport, Airports]           |
|[138, Queens, LaGuardia Airport, Airports]              |[138, 151]|[151, Manhattan, Manhattan Valley, Yellow Zone]      |
|[36, Brooklyn, Bushwick North, Boro Zone]               |[36, 129] |[129, Queens, Jackson Heights, Boro Zone]            |
|[138, Queens, LaGuardia Airport, Airports]              |[138, 88] |[88, Manhattan, Financial District South, Yellow Zone]|
|[200, Bronx, Riverdale/North Riverdale/Fieldston, Boro Zone]|[200, 138]|[138, Queens, LaGuardia Airport, Airports]       |
|[182, Bronx, Parkchester, Boro Zone]                    |[182, 242]|[242, Bronx, Van Nest/Morris Park, Boro Zone]        |
|[248, Bronx, West Farms/Bronx River, Boro Zone]         |[248, 242]|[242, Bronx, Van Nest/Morris Park, Boro Zone]        |
|[242, Bronx, Van Nest/Morris Park, Boro Zone]           |[242, 20] |[20, Bronx, Belmont, Boro Zone]                      |
+--------------------------------------------------------+---------+-----------------------------------------------------+
only showing top 10 rows

2024-03-30 19:29:39,582 INFO spark.SparkContext: Invoking stop() from shutdown hook
2024-03-30 19:29:39,601 INFO server.AbstractConnector: Stopped Spark@6660582a{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
```

```
2024-04-01 11:17:38,725 INFO
+---+------------------+
| id|          pagerank|
+---+------------------+
|265|11.105433344108016|
|  1| 5.471845424920981|
|132| 4.551132572067704|
|138| 3.568322341656072|
| 61|2.6763973653417987|
+---+------------------+
only showing top 5 rows
```

```
2024-04-01 11:00:43,943 INFO scheduler.DAGSchedule
countof same borough and service zone: 46886992
2024-04-01 11:00:44,188 INFO datasources.FileSourc
```

Challenges:

No challenges were faced here

Knowledge Gained:

- Creating graph and traversing it