

Coursework for ECS765P - Big Data Processing

Task 3: Analysis of Chicago Taxi Trips Data (25 points)

References:

1. Ramamonjison, R. (2025). Apache Spark Graph Processing https://search.library.qmul.ac.uk/iii/encore/record/C__Rb2519539__SApache%20Spark%20Graph%20Processing%20By%20Rindra%20Ramamonjison__Orightrightresult__U__X2?lang=eng&suite=def
2. Week 6 Lecture, Lab and related resources.

Dataset Description

The dataset contains information about taxi trips in Chicago starting from January 2024. It includes details such as trip start and end times, trip duration, distance, fare, tips, and pickup and dropoff locations. The data is anonymized to protect privacy. The dataset used in this coursework is `chicago_taxi_trips.csv` from `//data-repository-bkt/ECS765/Chicago_Taxitrips/` directory.

Purpose

The purpose of this task is to analyze taxi trip data to gain insights into trip patterns, fare distribution, and the connectivity between different community areas in Chicago.

Column Descriptions

Column Name	Description
Trip ID	A unique identifier for the trip.
Taxi ID	A unique identifier for the taxi.
Trip Start Timestamp	When the trip started, rounded to the nearest 15 minutes.
Trip End Timestamp	When the trip ended, rounded to the nearest 15 minutes.

Column Name	Description
Trip Seconds	Time of the trip in seconds.
Trip Miles	Distance of the trip in miles.
Pickup Census Tract	The Census Tract where the trip began.
Dropoff Census Tract	The Census Tract where the trip ended.
Pickup Community Area	The Community Area where the trip began.
Dropoff Community Area	The Community Area where the trip ended.
Fare	The fare for the trip.
Tips	The tip for the trip.
Tolls	The tolls for the trip.
Extras	Extra charges for the trip.
Trip Total	Total cost of the trip.
Payment Type	Type of payment for the trip.
Company	The taxi company.
Pickup Centroid Latitude	Latitude of the pickup location.
Pickup Centroid Longitude	Longitude of the pickup location.
Pickup Centroid Location	Location of the pickup centroid.
Dropoff Centroid Latitude	Latitude of the dropoff location.
Dropoff Centroid Longitude	Longitude of the dropoff location.
Dropoff Centroid Location	Location of the dropoff centroid.

Questions

1. Load Data (2 points)

- Load the taxi trips dataset into a DataFrame.
- Print the total number of entries in the DataFrame.
- Include a screenshot of your results in your report. For example:

```

2025-02-11 17:20:23,938 INFO scheduler.DAGScheduler: Job 2 is finished. Cancelling potential speculative or zombie tasks for this job
2025-02-11 17:20:23,938 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 3: Stage finished
2025-02-11 17:20:23,938 INFO scheduler.DAGScheduler: Job 2 finished: count at NativeMethodAccessorImpl.java:0, took 13.121950 s
Total number of entries: 1000000
2025-02-11 17:20:23,957 INFO server.AbstractConnector: Stopped Spark@19d4ad12{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2025-02-11 17:20:23,959 INFO ui.SparkUI: Stopped Spark web UI at http://task3-1-cce8ff94f605b39e-driver-svc.data-science-eex654.svc:4040

```

2. Define Schemas and Construct DataFrames (2 points)

- Define the `StructType` for the `vertexSchema` and `edgeSchema`.

- Construct the vertices DataFrame using the **Pickup Community Area** and **Dropoff Community Area** fields as the vertex information. Include the following fields in the vertices DataFrame:
 - **id**: Community area ID (either Pickup or Dropoff Community Area)
 - **Latitude**: Latitude of the centroid location (Pickup Centroid Latitude or Dropoff Centroid Latitude)
 - **Longitude**: Longitude of the centroid location (Pickup Centroid Longitude or Dropoff Centroid Longitude)
 - **Census Tract**: Census tract information (Pickup Census Tract or Dropoff Census Tract)
- Construct the edges DataFrame using the **Pickup Community Area** and **Dropoff Community Area** fields as the edge information. Include the following fields in the edges DataFrame:
 - **src**: Pickup Community Area
 - **dst**: Dropoff Community Area
 - **Trip Miles**: Distance of the trip
 - **Trip Seconds**: Duration of the trip
 - **Fare**: Fare amount
- Display 5 samples of both the edges and vertices DataFrames, ensuring that field names are not truncated.
- Include a screenshot of the results in your report. For example:

```

2025-02-12 12:23:47,767 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 10: Stage finished
2025-02-12 12:23:47,767 INFO scheduler.DAGScheduler: Job 4 finished: showString at NativeMethodAccessorImpl.java:0, took 0.127051 s
2025-02-12 12:23:47,786 INFO codegen.CodeGenerator: Code generated in 12.718855 ms
+-----+
|src|dst|Trip Miles|Trip Seconds|Fare|
+-----+
|src|dst|Trip Miles|Trip Seconds|Fare|
+-----+
only showing top 5 rows
+-----+
2025-02-12 12:23:47,797 INFO server.AbstractConnector: Stopped Spark@837931eda[HTTP/1.1,[http/1.1]]{0.0.0.0:4040}
2025-02-12 12:23:47,799 INFO ui.SparkUI: Stopped Spark web UI at http://task3-2-276f6994fa1bf917-driver-svc.data-science-eex654.svc:4040
2025-02-12 12:23:47,802 INFO k8s.KubernetesClusterSchedulerBackend: Shutting down all executors
2025-02-12 12:23:47,803 INFO k8s.KubernetesClusterSchedulerBackend$KubernetesDriverEndpoint: Asking each executor to shut down
2025-02-12 12:23:47,809 WARN k8s.ExecutorPodWatchSnapshotSource: Kubernetes client has been closed (this is expected if the application
is shutting down.)

```

```

2025-02-12 12:23:47,476 INFO scheduler.DAGScheduler: Job 3 is finished. Cancelling potential speculative or zombie tasks for this job
2025-02-12 12:23:47,476 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 9: Stage finished
2025-02-12 12:23:47,476 INFO scheduler.DAGScheduler: Job 3 finished: showString at NativeMethodAccessorImpl.java:0, took 0.083413 s
2025-02-12 12:23:47,505 INFO codegen.CodeGenerator: Code generated in 12.130045 ms

```

id	Latitude	Longitude	Census Tract

only showing top 5 rows

```

2025-02-12 12:23:47,575 INFO storage.BlockManagerInfo: Removed broadcast_6_piece0 on task3-2-276f6994fa1bf917-driver-svc.data-science-eex654.svc:7079 in memory (size: 12.7 KiB, free: 2004.4 MiB)
2025-02-12 12:23:47,577 INFO storage.BlockManagerInfo: Removed broadcast_6_piece0 on 10.132.110.58:45025 in memory (size: 12.7 KiB, free: 2.1 GiB)
2025-02-12 12:23:47,577 INFO storage.BlockManagerInfo: Removed broadcast_6_piece0 on 10.135.72.233:33577 in memory (size: 12.7 KiB, free: 2.1 GiB)

```

3. Create Graph and Display Samples (2 points)

- Create a graph using the vertices and edges DataFrames. In this context:
 - **Vertices** represent the nodes in the graph, which correspond to the community areas where taxi trips start or end.
 - **Edges** represent the connections between these nodes, which correspond to the taxi trips between different community areas.
- Show 10 samples of the graph DataFrame with columns `src`, `dst`, `Trip Miles`, `Trip Seconds`, `Fare`, `src_Latitude`, `src_Longitude`, `src_Census Tract`, `dst_Latitude`, `dst_Longitude`, and `dst_Census Tract`.
- Include a screenshot of your results. For Example:

```

2025-02-12 12:34:42,709 INFO scheduler.DAGScheduler: ResultStage 15 (showString at NativeMethodAccessorImpl.java:0) finished in 0.160 s
2025-02-12 12:34:42,709 INFO scheduler.DAGScheduler: Job 3 is finished. Cancelling potential speculative or zombie tasks for this job
2025-02-12 12:34:42,709 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 15: Stage finished
2025-02-12 12:34:42,709 INFO scheduler.DAGScheduler: Job 3 finished: showString at NativeMethodAccessorImpl.java:0, took 0.169854 s
2025-02-12 12:34:42,741 INFO codegen.CodeGenerator: Code generated in 14.617236 ms

```

src	dst	Trip Miles	Trip Seconds	Fare	src_Latitude	src_Longitude	src_Census Tract	dst_Latitude	dst_Longitude	dst_Census Tract

only showing top 10 rows

```

2025-02-12 12:34:42,755 INFO server.AbstractConnector: Stopped Spark@14addb70[HTTP/1.1,[http/1.1]]{0.0.0.0:4040}
2025-02-12 12:34:42,756 INFO ui.SparkUI: Stopped Spark web UI at http://task3-3-e2a6be94fa257586-driver-svc.data-science-eex654.svc:4040
2025-02-12 12:34:42,759 INFO k8s.KubernetesClusterSchedulerBackend: Shutting down all executors
2025-02-12 12:34:42,760 INFO k8s.KubernetesClusterSchedulerBackend$KubernetesDriverEndpoint: Asking each executor to shut down

```

4. Connected Community Areas Analysis (3 points)

- Count the number of edges where both the source (`src`) and destination (`dst`) vertices belong to the same community area.

- Display the total number of such connected vertices and 10 samples of the outcome.
- Include a screenshot of your results. For Example:

```

2025-02-12 13:07:12,457 INFO scheduler.DAGScheduler: ResultStage 4 (showString at NativeMethodAccessorImpl.java:0) finished in 0.231 s
2025-02-12 13:07:12,458 INFO scheduler.DAGScheduler: Job 3 is finished. Cancelling potential speculative or zombie tasks for this job
2025-02-12 13:07:12,458 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 4: Stage finished
2025-02-12 13:07:12,458 INFO scheduler.DAGScheduler: Job 3 finished: showString at NativeMethodAccessorImpl.java:0, took 0.235319 s
2025-02-12 13:07:12,486 INFO codegen.CodeGenerator: Code generated in 13.263047 ms

```

src	dst	Trip Miles	Trip Seconds	Fare
1	2	1.2	120	1.5
1	3	1.5	150	1.8
1	4	1.8	180	2.1
1	5	2.1	210	2.4
1	6	2.4	240	2.7
1	7	2.7	270	3.0
1	8	3.0	300	3.3
1	9	3.3	330	3.6
1	10	3.6	360	3.9
1	11	3.9	390	4.2
1	12	4.2	420	4.5
1	13	4.5	450	4.8
1	14	4.8	480	5.1
1	15	5.1	510	5.4
1	16	5.4	540	5.7
1	17	5.7	570	6.0
1	18	6.0	600	6.3
1	19	6.3	630	6.6
1	20	6.6	660	6.9
1	21	6.9	690	7.2
1	22	7.2	720	7.5
1	23	7.5	750	7.8
1	24	7.8	780	8.1
1	25	8.1	810	8.4
1	26	8.4	840	8.7
1	27	8.7	870	9.0
1	28	9.0	900	9.3
1	29	9.3	930	9.6
1	30	9.6	960	9.9
1	31	9.9	990	10.2
1	32	10.2	1020	10.5
1	33	10.5	1050	10.8
1	34	10.8	1080	11.1
1	35	11.1	1110	11.4
1	36	11.4	1140	11.7
1	37	11.7	1170	12.0
1	38	12.0	1200	12.3
1	39	12.3	1230	12.6
1	40	12.6	1260	12.9
1	41	12.9	1290	13.2
1	42	13.2	1320	13.5
1	43	13.5	1350	13.8
1	44	13.8	1380	14.1
1	45	14.1	1410	14.4
1	46	14.4	1440	14.7
1	47	14.7	1470	15.0
1	48	15.0	1500	15.3
1	49	15.3	1530	15.6
1	50	15.6	1560	15.9
1	51	15.9	1590	16.2
1	52	16.2	1620	16.5
1	53	16.5	1650	16.8
1	54	16.8	1680	17.1
1	55	17.1	1710	17.4
1	56	17.4	1740	17.7
1	57	17.7	1770	18.0
1	58	18.0	1800	18.3
1	59	18.3	1830	18.6
1	60	18.6	1860	18.9
1	61	18.9	1890	19.2
1	62	19.2	1920	19.5
1	63	19.5	1950	19.8
1	64	19.8	1980	20.1
1	65	20.1	2010	20.4
1	66	20.4	2040	20.7
1	67	20.7	2070	21.0
1	68	21.0	2100	21.3
1	69	21.3	2130	21.6
1	70	21.6	2160	21.9
1	71	21.9	2190	22.2
1	72	22.2	2220	22.5
1	73	22.5	2250	22.8
1	74	22.8	2280	23.1
1	75	23.1	2310	23.4
1	76	23.4	2340	23.7
1	77	23.7	2370	24.0
1	78	24.0	2400	24.3
1	79	24.3	2430	24.6
1	80	24.6	2460	24.9
1	81	24.9	2490	25.2
1	82	25.2	2520	25.5
1	83	25.5	2550	25.8
1	84	25.8	2580	26.1
1	85	26.1	2610	26.4
1	86	26.4	2640	26.7
1	87	26.7	2670	27.0
1	88	27.0	2700	27.3
1	89	27.3	2730	27.6
1	90	27.6	2760	27.9
1	91	27.9	2790	28.2
1	92	28.2	2820	28.5
1	93	28.5	2850	28.8
1	94	28.8	2880	29.1
1	95	29.1	2910	29.4
1	96	29.4	2940	29.7
1	97	29.7	2970	30.0
1	98	30.0	3000	30.3
1	99	30.3	3030	30.6
1	100	30.6	3060	30.9

only showing top 10 rows

```

2025-02-12 13:07:12,500 INFO server.AbstractConnector: Stopped Spark@75bf7e48[HTTP/1.1,[http/1.1]]{0.0.0.0:4040}
2025-02-12 13:07:12,501 INFO ui.SparkUI: Stopped Spark web UI at http://task3-4-fea15894fa44296c-driver-svc.data-science-eex654.svc:4040
2025-02-12 13:07:12,504 INFO k8s.KubernetesClusterSchedulerBackend: Shutting down all executors

```

5. Shortest Paths (5 points)

- Find the shortest paths from all other vertices (community areas) to a specific vertex (e.g., community area 49). Use BFS (breadth-first search) algorithm to find the shortest paths.
- Show 10 samples from your result.
- Include a screenshot of your results. For Example:

```

2025-02-24 14:49:42,974 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 95: Stage finished
2025-02-24 14:49:42,974 INFO scheduler.DAGScheduler: Job 8 finished: showString at NativeMethodAccessorImpl.java:0, took 0.093399 s
2025-02-24 14:49:42,996 INFO codegen.CodeGenerator: Code generated in 15.576272 ms

```

id	landmark	distance
1	1	0.0
1	2	0.1
1	3	0.2
1	4	0.3
1	5	0.4
1	6	0.5
1	7	0.6
1	8	0.7
1	9	0.8
1	10	0.9
1	11	1.0
1	12	1.1
1	13	1.2
1	14	1.3
1	15	1.4
1	16	1.5
1	17	1.6
1	18	1.7
1	19	1.8
1	20	1.9
1	21	2.0
1	22	2.1
1	23	2.2
1	24	2.3
1	25	2.4
1	26	2.5
1	27	2.6
1	28	2.7
1	29	2.8
1	30	2.9
1	31	3.0
1	32	3.1
1	33	3.2
1	34	3.3
1	35	3.4
1	36	3.5
1	37	3.6
1	38	3.7
1	39	3.8
1	40	3.9
1	41	4.0
1	42	4.1
1	43	4.2
1	44	4.3
1	45	4.4
1	46	4.5
1	47	4.6
1	48	4.7
1	49	4.8
1	50	4.9
1	51	5.0
1	52	5.1
1	53	5.2
1	54	5.3
1	55	5.4
1	56	5.5
1	57	5.6
1	58	5.7
1	59	5.8
1	60	5.9
1	61	6.0
1	62	6.1
1	63	6.2
1	64	6.3
1	65	6.4
1	66	6.5
1	67	6.6
1	68	6.7
1	69	6.8
1	70	6.9
1	71	7.0
1	72	7.1
1	73	7.2
1	74	7.3
1	75	7.4
1	76	7.5
1	77	7.6
1	78	7.7
1	79	7.8
1	80	7.9
1	81	8.0
1	82	8.1
1	83	8.2
1	84	8.3
1	85	8.4
1	86	8.5
1	87	8.6
1	88	8.7
1	89	8.8
1	90	8.9
1	91	9.0
1	92	9.1
1	93	9.2
1	94	9.3
1	95	9.4
1	96	9.5
1	97	9.6
1	98	9.7
1	99	9.8
1	100	9.9

only showing top 10 rows

```

2025-02-24 14:49:43,022 INFO server.AbstractConnector: Stopped Spark@50774e36[HTTP/1.1,[http/1.1]]{0.0.0.0:4040}
2025-02-24 14:49:43,024 INFO ui.SparkUI: Stopped Spark web UI at http://task3-5-575d2995386bd59d-driver-svc.data-science-eex654.svc:4040
2025-02-24 14:49:43,028 INFO k8s.KubernetesClusterSchedulerBackend: Shutting down all executors

```

6. PageRank Analysis (6 points)

- Perform PageRank on the graph DataFrame with `resetProbability` set to 0.15 and `tol` set to 0.01.
- Deduplicate the vertices DataFrame to ensure unique vertices.
- Create the GraphFrame using the deduplicated vertices and the edges.

- Sort vertices by descending PageRank value and show the top 5 results.
- Include a screenshot of your results. For Example:

```

2025-02-12 14:20:40,980 INFO scheduler.DAGScheduler: ResultStage 1096 (showString at NativeMethodAccessorImpl.java:0) finished in 0.881 s
2025-02-12 14:20:40,981 INFO scheduler.DAGScheduler: Job 25 is finished. Cancelling potential speculative or zombie tasks for this job
2025-02-12 14:20:40,981 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 1096: Stage finished
2025-02-12 14:20:40,981 INFO scheduler.DAGScheduler: Job 25 finished: showString at NativeMethodAccessorImpl.java:0, took 46.012517 s
2025-02-12 14:20:41,003 INFO codegen.CodeGenerator: Code generated in 11.839255 ms
2025-02-12 14:20:41,013 INFO codegen.CodeGenerator: Code generated in 5.845508 ms
+-----+
|id|pagerank|
+-----+
|8|0.11988828045910191|
|76|0.10755584200626768|
|32|0.0794174570327947|
|28|0.0665599503647182|
|33|0.03634550724544256|
+-----+
only showing top 5 rows

2025-02-12 14:20:41,030 INFO server.AbstractConnector: Stopped Spark@476c8cd6(HTTP/1.1,[http://1.1.1.1:4040])
2025-02-12 14:20:41,031 INFO ui.SparkUI: Stopped Spark web UI at http://task3-6-02761d94fa8402cc-driver-svc.data-science-ee654.svc:4040
2025-02-12 14:20:41,036 INFO k8s.KubernetesClusterSchedulerBackend: Shutting down all executors
2025-02-12 14:20:41,037 INFO k8s.KubernetesClusterSchedulerBackend$KubernetesDriverEndpoint: Asking each executor to shut down
2025-02-12 14:20:41,045 WARN k8s.ExecutorPodWatchSnapshotSource: Kubernetes client has been closed (this is expected if the application
is shutting down.)

```

- Modify the PageRank algorithm to consider the weights of the edges (e.g., trip distance) instead of treating all edges equally. This will give more importance to vertices connected by higher-weight edges.
- Define the vertices by selecting the unique community areas from both the pickup and dropoff locations, ensuring no duplicates.
- Construct the edges by selecting the pickup and dropoff community areas along with the trip distance, and rename the trip distance to represent the weight of the edge.
- Compute the total outgoing weight for each node by aggregating the weights of all outgoing edges from each source node.
- Normalize the weights of the edges by dividing each edge's weight by the total outgoing weight of its source node.
- Create the GraphFrame using the deduplicated vertices and the normalized edges.
- Sort vertices by descending PageRank value and show the top 5 results.
- Include a screenshot of your results. For Example:

```

2025-02-23 18:07:21,298 INFO codegen.CodeGenerator: Code generated in 10.467385 ms
2025-02-23 18:07:21,309 INFO codegen.CodeGenerator: Code generated in 6.746676 ms
+-----+
|id|pagerank|
+-----+
|8|0.11988828045910191|
|76|0.10755584200626768|
|32|0.0794174570327947|
|28|0.0665599503647182|
|33|0.03634550724544256|
+-----+
only showing top 5 rows

```

7. Fare Analysis with GraphFrames (5 points)

- Analyze the distribution of fares using a **GraphFrame** representation. Create a graph where the community areas (pickup and dropoff) are the vertices, and taxi trips (with attributes such as trip distance, duration, and fare) are the edges.
- Plot a histogram showing the fare distribution and analyze the factors influencing the fare amount, such as trip distance, duration, and time of day.
- Using the **GraphFrame**, identify and discuss how node and edge attributes (e.g., community areas, trip distance, duration) might correlate with the fare.
- Include a screenshot of your result displaying the fare distribution with respect to trip distance, trip duration and time of the day. For example:

```
2025-02-23 20:33:25,318 INFO scheduler.DAGScheduler: Job 10 is finished. Cancelling potential speculative or zombie tasks for this job
2025-02-23 20:33:25,318 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 18: Stage finished
2025-02-23 20:33:25,319 INFO scheduler.DAGScheduler: Job 10 finished: showString at NativeMethodAccessorImpl.java:0, took 19.29562 s
```

Trip Miles	Avg Fare
100	1.50
150	2.00
200	2.50
250	3.00
300	3.50
350	4.00
400	4.50
450	5.00
500	5.50
550	6.00
600	6.50
650	7.00
700	7.50
750	8.00
800	8.50
850	9.00
900	9.50
950	10.00
1000	10.50
1050	11.00
1100	11.50
1150	12.00
1200	12.50
1250	13.00
1300	13.50
1350	14.00
1400	14.50
1450	15.00
1500	15.50
1550	16.00
1600	16.50
1650	17.00
1700	17.50
1750	18.00
1800	18.50
1850	19.00
1900	19.50
1950	20.00
2000	20.50
2050	21.00
2100	21.50
2150	22.00
2200	22.50
2250	23.00
2300	23.50
2350	24.00
2400	24.50
2450	25.00
2500	25.50
2550	26.00
2600	26.50
2650	27.00
2700	27.50
2750	28.00
2800	28.50
2850	29.00
2900	29.50
2950	30.00
3000	30.50
3050	31.00
3100	31.50
3150	32.00
3200	32.50
3250	33.00
3300	33.50
3350	34.00
3400	34.50
3450	35.00
3500	35.50
3550	36.00
3600	36.50
3650	37.00
3700	37.50
3750	38.00
3800	38.50
3850	39.00
3900	39.50
3950	40.00
4000	40.50
4050	41.00
4100	41.50
4150	42.00
4200	42.50
4250	43.00
4300	43.50
4350	44.00
4400	44.50
4450	45.00
4500	45.50
4550	46.00
4600	46.50
4650	47.00
4700	47.50
4750	48.00
4800	48.50
4850	49.00
4900	49.50
4950	50.00
5000	50.50
5050	51.00
5100	51.50
5150	52.00
5200	52.50
5250	53.00
5300	53.50
5350	54.00
5400	54.50
5450	55.00
5500	55.50
5550	56.00
5600	56.50
5650	57.00
5700	57.50
5750	58.00
5800	58.50
5850	59.00
5900	59.50
5950	60.00
6000	60.50
6050	61.00
6100	61.50
6150	62.00
6200	62.50
6250	63.00
6300	63.50
6350	64.00
6400	64.50
6450	65.00
6500	65.50
6550	66.00
6600	66.50
6650	67.00
6700	67.50
6750	68.00
6800	68.50
6850	69.00
6900	69.50
6950	70.00
7000	70.50
7050	71.00
7100	71.50
7150	72.00
7200	72.50
7250	73.00
7300	73.50
7350	74.00
7400	74.50
7450	75.00
7500	75.50
7550	76.00
7600	76.50
7650	77.00
7700	77.50
7750	78.00
7800	78.50
7850	79.00
7900	79.50
7950	80.00
8000	80.50
8050	81.00
8100	81.50
8150	82.00
8200	82.50
8250	83.00
8300	83.50
8350	84.00
8400	84.50
8450	85.00
8500	85.50
8550	86.00
8600	86.50
8650	87.00
8700	87.50
8750	88.00
8800	88.50
8850	89.00
8900	89.50
8950	90.00
9000	90.50
9050	91.00
9100	91.50
9150	92.00
9200	92.50
9250	93.00
9300	93.50
9350	94.00
9400	94.50
9450	95.00
9500	95.50
9550	96.00
9600	96.50
9650	97.00
9700	97.50
9750	98.00
9800	98.50
9850	99.00
9900	99.50
9950	100.00

```
2025-02-23 20:33:44,064 INFO scheduler.DAGScheduler: Job 11 is finished. Cancelling potential speculative or zombie tasks for this job
2025-02-23 20:33:44,064 INFO scheduler.TaskSchedulerImpl: Killing all running tasks in stage 20: Stage finished
2025-02-23 20:33:44,065 INFO scheduler.DAGScheduler: Job 11 finished: showString at NativeMethodAccessorImpl.java:0, took 18.568875 s
```

Trip Seconds	Avg Fare
10	1.00
15	1.50
20	2.00
25	2.50
30	3.00
35	3.50
40	4.00
45	4.50
50	5.00
55	5.50
60	6.00
65	6.50
70	7.00
75	7.50
80	8.00
85	8.50
90	9.00
95	9.50
100	10.00
105	10.50
110	11.00
115	11.50
120	12.00
125	12.50
130	13.00
135	13.50
140	14.00
145	14.50
150	15.00
155	15.50
160	16.00
165	16.50
170	17.00
175	17.50
180	18.00
185	18.50
190	19.00
195	19.50
200	20.00
205	20.50
210	21.00
215	21.50
220	22.00
225	22.50
230	23.00
235	23.50
240	24.00
245	24.50
250	25.00
255	25.50
260	26.00
265	26.50
270	27.00
275	27.50
280	28.00
285	28.50
290	29.00
295	29.50
300	30.00
305	30.50
310	31.00
315	31.50
320	32.00
325	32.50
330	33.00
335	33.50
340	34.00
345	34.50
350	35.00
355	35.50
360	36.00
365	36.50
370	37.00
375	37.50
380	38.00
385	38.50
390	39.00
395	39.50
400	40.00
405	40.50
410	41.00
415	41.50
420	42.00
425	42.50
430	43.00
435	43.50
440	44.00
445	44.50
450	45.00
455	45.50
460	46.00
465	46.50
470	47.00
475	47.50
480	48.00
485	48.50
490	49.00
495	49.50
500	50.00
505	50.50
510	51.00
515	51.50
520	52.00
525	52.50
530	53.00
535	53.50
540	54.00
545	54.50
550	55.00
555	55.50
560	56.00
565	56.50
570	57.00
575	57.50
580	58.00
585	58.50
590	59.00
595	59.50
600	60.00
605	60.50
610	61.00
615	61.50
620	62.00
625	62.50
630	63.00
635	63.50
640	64.00
645	64.50
650	65.00
655	65.50
660	66.00
665	66.50
670	67.00
675	67.50
680	68.00
685	68.50
690	69.00
695	69.50
700	70.00
705	70.50
710	71.00
715	71.50
720	72.00
725	72.50
730	73.00
735	73.50
740	74.00
745	74.50
750	75.00
755	75.50
760	76.00
765	76.50
770	77.00
775	77.50
780	78.00
785	78.50
790	79.00
795	79.50
800	80.00
805	80.50
810	81.00
815	81.50
820	82.00
825	82.50
830	83.00
835	83.50
840	84.00
845	84.50
850	85.00
855	85.50
860	86.00
865	86.50
870	87.00
875	87.50
880	88.00
885	88.50
890	89.00
895	89.50

[illegible]