

Project 3 Report:

Working with Big Data Infrastructure Spark

General Introduction:

In this report, we describe how the project delves into two key domains: Spark-RDDs for scalable analysis of COVID-19 related data and SparkSQL with MLlib for processing customer purchase transactions. For COVID-19 data, diverse RDD queries are designed to identify close contacts. In the purchase transaction realm, SparkSQL filters, groups, and analyses transactions, while MLlib predicts purchase prices. Together, these projects showcase the power of Apache Spark in handling diverse big data scenarios, from health crises to consumer behaviour analytics. We chose to do the project in Jupyter Notebook IDE due to our familiarity with it for both the tasks 1 & 2 individually. We'll see each one of them in detail one by one here.

Team Composition:

Our team members are namely,

- Sreeram Marimuthu
- Nitya Phani Santosh Oruganty
- Anushka Bangal

We cooperated to execute this project as planned. To ensure the project's success, each team member had a specific responsibility.

TASK 1- Spark-RDDs: Scalable Covid19-related Applications

Introduction:

In this part of the project, we explore the application of Spark Resilient Distributed Datasets (RDDs) to analyse and process data related to COVID-19 social distancing at a concert. The project involves two main steps as follows: data creation and queries.

Spark Configuration:

The script begins by configuring Spark to handle big data. It sets memory allocation for both executor and driver using ``SparkConf`` and creates a ``SparkContext`` named 'spark' to facilitate Spark operations.

Task 1.1 – Data Creation:

The data creation is the initial step of Task 1 in the Spark-RDDs project, focusing on scalable COVID-19-related applications. The code is designed to generate synthetic datasets simulating people attending an outdoor concert, with a subset of individuals marked as infected with COVID-19 with the help of faker library.

- **generate_concert_data Function:**

The function utilizes the Faker library to create synthetic data for individuals attending the concert. This includes attributes such as unique ID, X and Y coordinates, Name, and Age. The uniqueness of coordinates is ensured to mimic distinct seating positions for concertgoers.

- **Datasets:**

Three datasets, namely PEOPLE_large, INFECTED_small, and SOME_INFECTED_large, are created. These datasets represent people seated at an outdoor concert and contain 2D points (x, y) with additional attributes such as unique identifiers, names & ages. INFECTED_small is a subset of PEOPLE_large, containing individuals infected with COVID-19. SOME_INFECTED_large is either assumed to be based on records in INFECTED_small. It includes an additional attribute "INFECTED" indicating whether each person is infected or not.

PEOPLE_large Dataset:

The function is called to generate a dataset named 'PEOPLE_large.csv' containing 100,000 entries. This dataset represents all individuals present at the concert, with random attributes generated for each attendee. A preview of how it looks like is as below:

ID	X	Y	Name	Age
1	1142	8149	Patricia Decker	21
2	6788	193	Michelle Harmon	40
3	3837	1516	Andrea Meyer	42
4	5222	2005	Alexander Ewing	28
5	2315	207	Jacqueline Holmes	50
6	2409	1762	Donald Mccoy	48
7	2898	8213	Christopher White	76
8	2316	7964	Michael Lee	61
9	8932	1356	Dana Hebert	18
10	1985	6706	Kathy Snyder	94
11	5298	2974	Kimberly Ballard	100
12	4613	3471	Steven Vega	79
13	6419	8336	Brandon Lawrence	54
14	8360	4330	Whitney Smith	83
15	9157	2302	Savannah Hines	79
16	4359	5272	Heather Cooper	15
17	3411	7365	Barbara Norton	73
18	1386	5902	Sarah Shaw	22
19	7262	6290	Kathleen Morgan	53
20	6617	6190	Craig Thomas	28

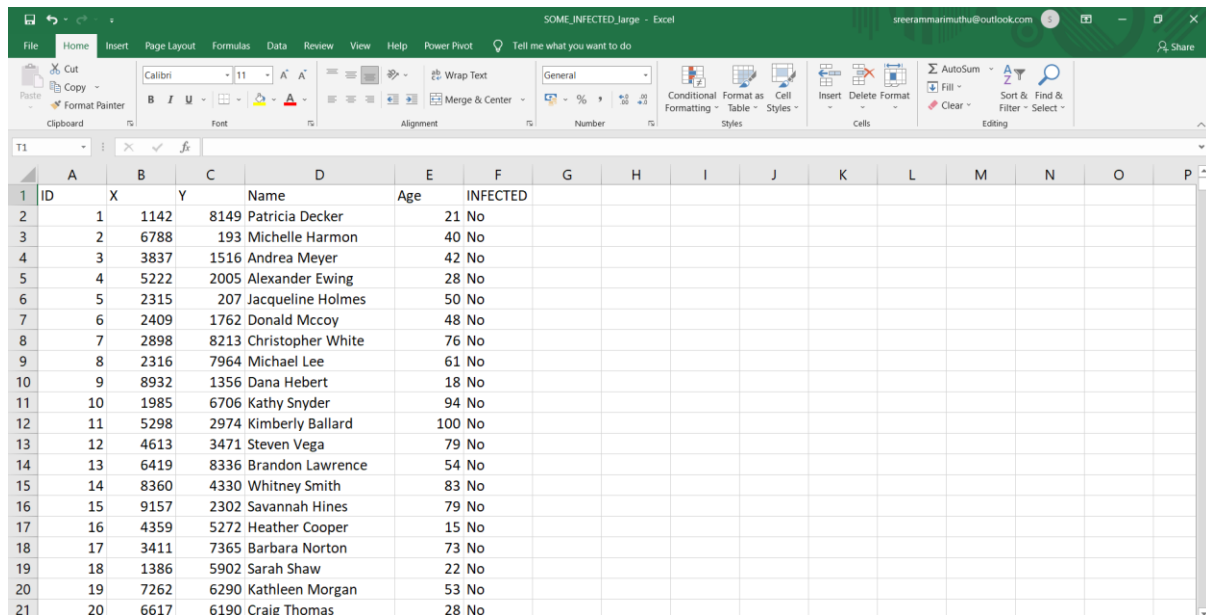
INFECTED_small Dataset:

A subset of 'PEOPLE_large' is sampled to create the 'INFECTED_small.csv' dataset, simulating individuals infected with COVID-19. This subset, containing 5000 entries, represents the known cases of infection among attendees. A preview of how it looks like is as below:

ID	X	Y	Name	Age
58601	4565	6521	Samantha Brown MD	84
20788	8020	2174	Michael Green	96
55866	3296	5421	Bruce Smith	89
46637	472	8965	Loretta Gomez	87
55751	2019	7510	Brenda Beltran	31
16989	5265	6786	Kimberly Kaufman	87
75355	3979	6618	Vanessa Webster	54
73846	3904	7230	Angela Howard	55
97621	4154	1553	Dr. Nicholas Smith	62
90721	1859	2560	Shannon Frederick	18
70473	3333	1755	Teresa Cook DDS	47
63401	6276	7589	Barbara Duncan	30
98857	8629	6708	Robert Duffy	24
20435	9686	2726	Timothy Clark	82
96147	9527	5164	Emily Hart	43
13804	338	4341	Christy Summers	57
33205	2712	5177	Rebecca Christian	71
59045	9363	6974	Jennifer Williams	17
82113	367	7813	Marc Dunn	56
50404	3554	8412	Rachael Kim	15

SOME_INFECTED_large Dataset:

Our final dataset, 'SOME_INFECTED_large.csv', is created based on the 'PEOPLE_large' dataset. An additional column, 'INFECTED', is introduced, marking individuals as 'Yes' if their ID matches those in 'INFECTED_small', and 'No' otherwise following the “Assumption_1”. A preview of how it looks like is as below:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	ID	X	Y	Name	Age	INFECTED										
2	1	1142	8149	Patricia Decker	21	No										
3	2	6788	193	Michelle Harmon	40	No										
4	3	3837	1516	Andrea Meyer	42	No										
5	4	5222	2005	Alexander Ewing	28	No										
6	5	2315	207	Jacqueline Holmes	50	No										
7	6	2409	1762	Donald Mccoy	48	No										
8	7	2898	8213	Christopher White	76	No										
9	8	2316	7964	Michael Lee	61	No										
10	9	8932	1356	Dana Hebert	18	No										
11	10	1985	6706	Kathy Snyder	94	No										
12	11	5298	2974	Kimberly Ballard	100	No										
13	12	4613	3471	Steven Vega	79	No										
14	13	6419	8336	Brandon Lawrence	54	No										
15	14	8360	4330	Whitney Smith	83	No										
16	15	9157	2302	Savannah Hines	79	No										
17	16	4359	5272	Heather Cooper	15	No										
18	17	3411	7365	Barbara Norton	73	No										
19	18	1386	5902	Sarah Shaw	22	No										
20	19	7262	6290	Kathleen Morgan	53	No										
21	20	6617	6190	Craig Thomas	28	No										

A subset of all the 3 above mentioned big datasets made up of first 1000 entries is attached with this submission for your reference.

RDDs:

The final section involves loading the generated datasets ('INFECTED_small.csv' and 'PEOPLE_large.csv') into Spark Resilient Distributed Datasets (RDDs) named 'infected_rdd' and 'person_rdd', respectively.

Conclusion: In summary, the code successfully generates synthetic datasets representing people attending a concert, some of whom are infected with COVID-19. These datasets will serve as the foundation for subsequent Spark RDD queries to analyse social distancing measures and potential infections in the simulated outdoor event.

Task 1.2 – Queries:

This part of the project involves designing scalable solutions in Spark using Pyspark and RDDs for three specific queries as expected.

Query 1: Identify people in PEOPLE-large who were in close contact with infected individuals from INFECTED-small, considering a 6-unit range. Return pairs (pi, infect-i) representing close contacts.

The objective of Query 1 is to identify individuals from the 'PEOPLE_large' dataset who had close contact with an infected person from the 'INFECTED_small' dataset, considering a 6-unit range. The resulting pairs are to be returned as (pi, infect-i).

Data Pre-processing:

- Two RDDs, 'infected_rdd' and 'person_rdd' created in the dataset creation stage, representing 'INFECTED_small' and 'PEOPLE_large' datasets, are loaded into the script.
- The CSV parsing function 'csv_' is defined to split each line into a list of values.

distance_between Function:

A function 'distance_between' is defined to calculate the Euclidean distance between two points in 2D space (here the concert layout), representing the X and Y coordinates.

RDD Transformation:

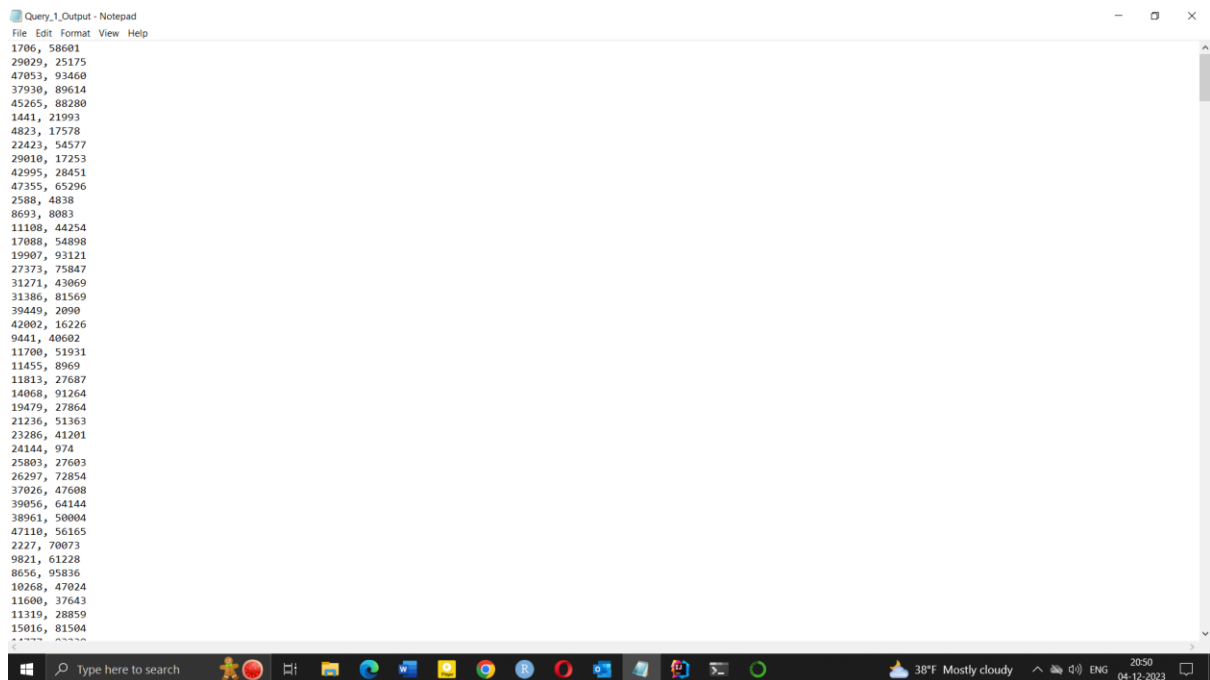
- 'infected_q1' and 'people_q1' RDDs are created by applying transformations to 'infected_rdd' and 'person_rdd', filtering out header lines.
- Coordinates are extracted from each RDD, and a new representation of data is created with the format (ID, X, Y, additional attributes).

Identifying Close Contacts:

- The IDs of infected individuals are collected, and the 'people_coordinates' RDD is filtered to exclude those in 'infected_id'.
- Cartesian product of 'infected_coordinates' and 'people_coordinates' is created to generate all possible pairs.
- A filter is applied to keep only those pairs with a distance less than or equal to 6 units, signifying close contacts.
- The result is mapped to a format (pi, infect-i).

Output:

- The close contact pairs are collected and stored in 'result_list'.
- The output is written to a text file named 'Query_1_Output.txt', where each line represents a close contact pair.



```
Query_1_Output - Notepad
File Edit Format View Help
1706, 58601
29029, 25175
47053, 93460
37930, 89614
45265, 88280
1441, 21993
4823, 17578
22423, 54577
29010, 17253
42995, 28451
47355, 65296
2588, 4838
8693, 8083
11108, 44254
17088, 54898
19907, 93121
27373, 75847
31271, 43069
31386, 81569
39449, 2090
42002, 16226
9441, 40602
11700, 51931
11455, 8969
11813, 27687
14068, 91264
19479, 27864
21236, 51363
23286, 41201
24144, 974
25803, 27603
26297, 72854
37026, 47608
39056, 64144
38961, 50004
47110, 56105
2227, 70073
9821, 61228
8656, 95836
10268, 47024
11600, 37643
11319, 28859
15016, 81504
```

The script successfully implements Query 1, identifying and recording individuals from 'PEOPLE_large' who were in close contact with infected persons from 'INFECTED_small' within a 6-unit range. The resulting pairs are saved for further analysis or reporting.

This implementation demonstrates the utilization of Spark RDDs to efficiently analyse and identify close contacts in a simulated scenario, providing valuable insights into potential COVID-19 transmissions at the outdoor concert.

Query 2: Similar to Query 1, but with duplicate contacts removed. Return unique pi.id values.

This section elucidates the Python code devised to execute Query 2 of Task 1 in the Spark-RDDs project. Query 2 seeks to identify distinct individuals from the 'PEOPLE_large' dataset who had close contact with an infected person from the 'INFECTED_small' dataset within a 6-unit range. The resulting output includes unique pi.id values with duplicate contacts removed.

Data Pre-processing:

Similar to Query 1, 'infected_q2' and 'people_q2' RDDs are generated by applying transformations and filtering out header lines.

RDD Transformation:

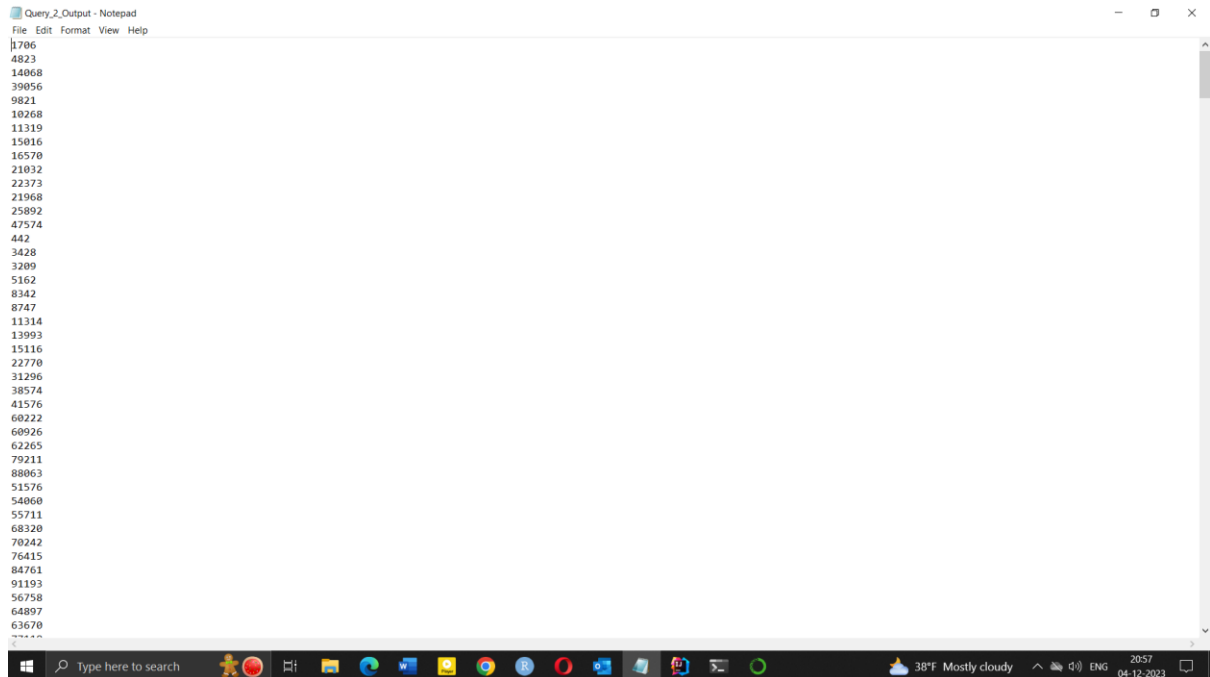
Coordinates are extracted from 'infected_q2' and 'people_q2', creating a new representation of data with the format (ID, X, Y, additional attributes).

Identifying Distinct Close Contacts:

- The IDs of infected individuals are collected, and 'people_coordinates' RDD is filtered to exclude those in 'infected_id'.
- Cartesian product of 'infected_coordinates' and 'people_coordinates' is created to generate all possible pairs.
- A filter is applied to keep only those pairs with a distance less than or equal to 6 units, signifying close contacts.
- Duplicate contacts are removed using Spark transformations:
- Mapping the pairs to (pi, contact pair) format.
- Reducing by key (pi) to retain only unique pairs.
- Mapping again to get the final distinct pairs.

Output:

- The distinct close contact pairs are collected and stored in 'distinct_result_list'.
- The output is written to a text file named 'Query_2_Output.txt', where each line represents a unique pi.id value.



```
Query_2_Output - Notepad
File Edit Format View Help
1706
4823
14068
39056
9821
10268
11319
15016
16570
21032
22373
21968
25892
47574
442
3428
3209
5162
8342
8747
11314
13993
15116
22770
31296
38574
41576
60222
60926
62265
79211
88063
51576
54060
55711
68320
70242
76415
84761
91193
56758
64897
63670
```

The script successfully implements Query 2, identifying and recording distinct individuals from 'PEOPLE_large' who had close contact with infected persons from 'INFECTED_small' within a 6-unit range. Duplicate contacts are removed to provide a clean list of unique pi.id values.

This implementation showcases the efficiency of Spark RDDs in handling and transforming data to achieve the desired analytical outcome, addressing the specific requirements of Query 2 in the context of the simulated outdoor concert scenario.

Query 3: Analyse SOME_INFECTED_large to count the number of close contacts within a 6-feet range for each infected person. Return pairs (infect-i, count-of-close-contacts-of-infect-i).

Query 3 aims to analyse the SOME_INFECTED_large dataset to count the number of close contacts within a 6-feet range for each infected person. The resulting output includes pairs (infect-i, count-of-close-contacts-of-infect-i).

Data Pre-processing:

The 'some_infected_rdd' RDD is loaded from the 'SOME_INFECTED_large.csv' file. Similar to previous queries, transformations and filtering are applied to exclude header lines and empty entries.

RDD Transformation:

Coordinates are extracted from 'some_infected_rdd', creating a new representation of data with the format (ID, X, Y, additional attributes).

Identifying Infected and Non-Infected Individuals:

'people_infected_yes' and 'people_infected_no' RDDs are generated by filtering 'some_infected_rdd' based on the 'INFECTED' attribute.

Calculating Close Contacts:

- Cartesian product of 'people_infected_yes' and 'people_infected_no' is created to generate all possible pairs.
- A filter is applied to keep only those pairs with a distance less than or equal to 6 units, signifying close contacts.
- The resulting pairs are mapped to the format (infect-i, non-infected person ID, non-infected person attributes).

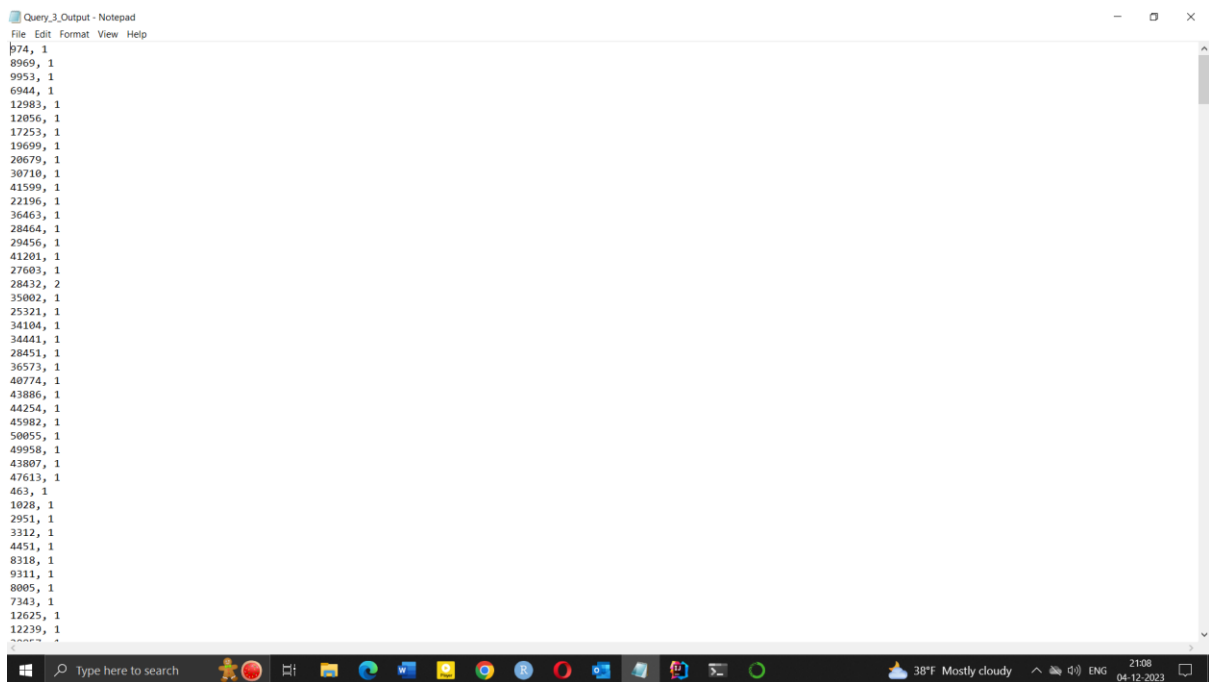
Counting Close Contacts for Each Infected Person:

- Close contact pairs are mapped to (infect-i, 1) format.

- A 'reduceByKey' operation is applied to calculate the count of close contacts for each infected person.

Output:

- The counts of close contacts for each infected person are collected.
- The output is written to a text file named 'Query_3_Output.txt', where each line represents an infected person and their count of close contacts.



```

File Edit Format View Help
974, 1
8969, 1
9953, 1
6944, 1
12983, 1
12056, 1
17253, 1
19699, 1
20679, 1
30710, 1
41599, 1
22196, 1
36463, 1
28464, 1
29456, 1
41201, 1
27603, 1
28432, 2
35002, 1
25321, 1
34104, 1
34441, 1
28451, 1
36573, 1
48774, 1
43886, 1
44254, 1
45982, 1
50055, 1
49958, 1
43807, 1
47613, 1
463, 1
1028, 1
2951, 1
3312, 1
4451, 1
8318, 1
9311, 1
8005, 1
7343, 1
12625, 1
12239, 1

```

The script effectively implements Query 3, analyzing the 'PEOPLE-SOME-INFECTED-large' dataset to count the number of close contacts within a 6-feet range for each infected person. The resulting pairs provide valuable information for assessing potential spread and identifying high-risk individuals.

Conclusion:

This implementation demonstrates the versatility of Spark RDDs in performing complex analytics on large-scale datasets, contributing crucial insights into the spread of infections in a simulated context. For each query, the solution is implemented in Spark using PySpark completely, focusing on RDDs. A brief description and discussion of each solution are provided.

TASK 2 - SparkSQL and MLlib for Processing Purchase Transactions

This project involves the creation and analysis of two datasets: Customers and Purchases. The datasets are processed using SparkSQL and MLlib for predicting purchase prices. The project consists of three steps: data creation & loading, SparkSQL workflows and MLlib workflows.

Step 1 – Data Creation:

This section of the report provides a comprehensive overview of the creation of two datasets, 'Customers' and 'Purchases,' as part of Task 2 in the SparkSQL and MLlib project. The datasets simulate customer information and purchase transactions, serving as the foundation for subsequent analysis using SparkSQL and MLlib.

Spark Session Configuration:

A Spark session is created with memory configurations to handle significant data. The session is named "Purchases," and memory specifications are set for both executor and driver.

Data Creation Functions:

- Two functions, ``generate_customers_data`` and ``generate_purchases_data``, are defined to generate random data for the 'Customers' and 'Purchases' datasets, respectively.
- The ``Faker`` library is employed to generate realistic customer names, and random numbers are utilized for age, country code, salary, transaction total, transaction number of items, and transaction description.

Data Generation:

- Random data for 'Customers' and 'Purchases' datasets is generated using the defined functions.
- The number of customers is set to 50,000, and the number of purchases is set to 5,000,000.

- The data is organized into lists of tuples, each representing a record in the dataset.

DataFrames Creation:

- Spark DataFrames, 'customers_df' and 'purchases_df,' are created from the generated data.
- Appropriate column names are assigned to each DataFrame.

Displaying Data:

The first five rows of each dataset are displayed using the `show` method to provide a glimpse of the generated data.

```
+-----+-----+-----+-----+
| ID|          Name|Age|CountryCode|    Salary|
+-----+-----+-----+-----+
|  1| Melissa Nelson| 98|          256|2192129.07|
|  2|  Dillon Lewis| 69|          358|5609320.09|
|  3|Christine Wolfe| 95|          332|7751332.78|
|  4|   Lisa Greene| 42|          172|4150480.34|
|  5|  Diana Stewart| 83|          146|6896931.95|
+-----+-----+-----+-----+
only showing top 5 rows
```

```
+-----+-----+-----+-----+-----+
|TransID|CustID|TransTotal|TransNumItems|          TransDesc|
+-----+-----+-----+-----+-----+
|      1| 38761|   1620.68|           15|Sound others inve...|
|      2| 49772|    331.5|            2|Health customer t...|
|      3| 13797|    796.66|            4| Direction who wide.|
|      4| 39414|    623.21|           15|Just memory to se...|
|      5| 26491|   1693.1|            9|Management situat...|
+-----+-----+-----+-----+-----+
only showing top 5 rows
```

Writing Datasets:

- The generated datasets are written to the local filesystem. In this case, they are saved in the Parquet format.
- 'customers_df' is written to "customers.parquet," and 'purchases_df' is written to "purchases.parquet."

DataFrames as SQL Tables:

- The DataFrames are registered as temporary SQL tables, allowing them to be queried using SparkSQL.
- 'customers_df' is registered as "Customers," and 'purchases_df' is registered as "Purchases."

Conclusion: The code successfully generates synthetic datasets mimicking customer information and purchase transactions. These datasets will serve as the foundation for subsequent SparkSQL and MLlib analyses, providing realistic data for evaluating and predicting purchase prices. The flexibility of SparkSQL allows for efficient querying and exploration of these datasets.

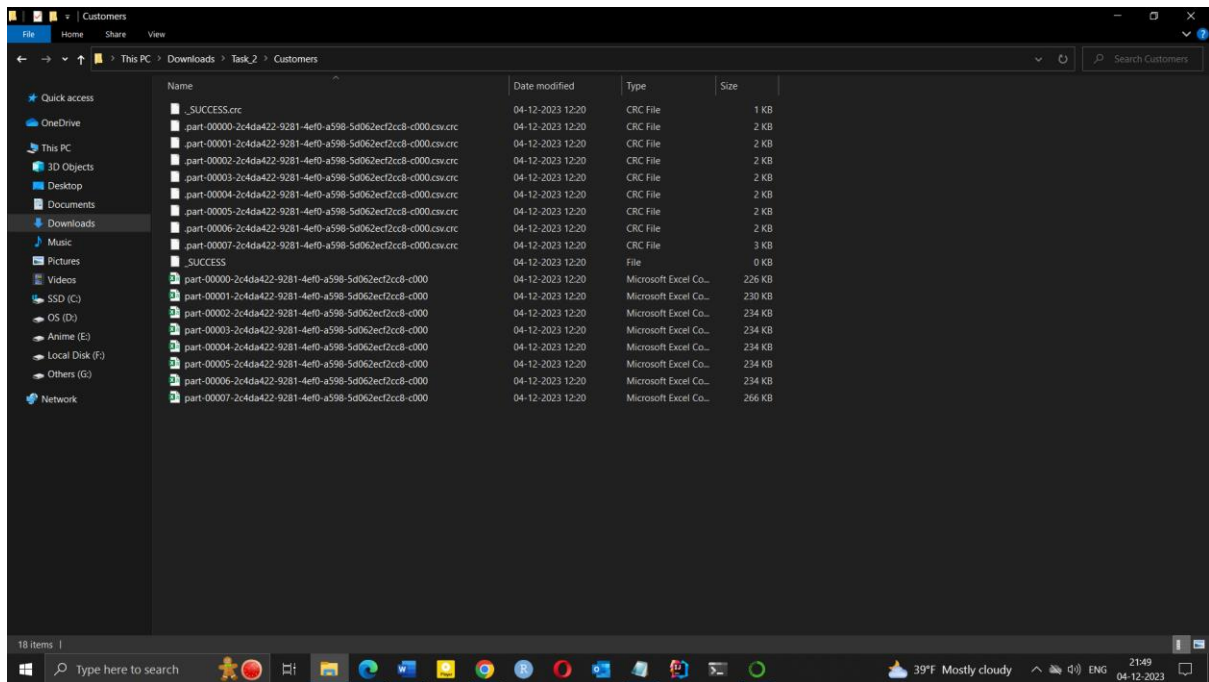
Task 2.0 – Load Data:

The code utilizes SparkSQL to write the datasets to both CSV files and Parquet format for efficient storage and future retrieval.

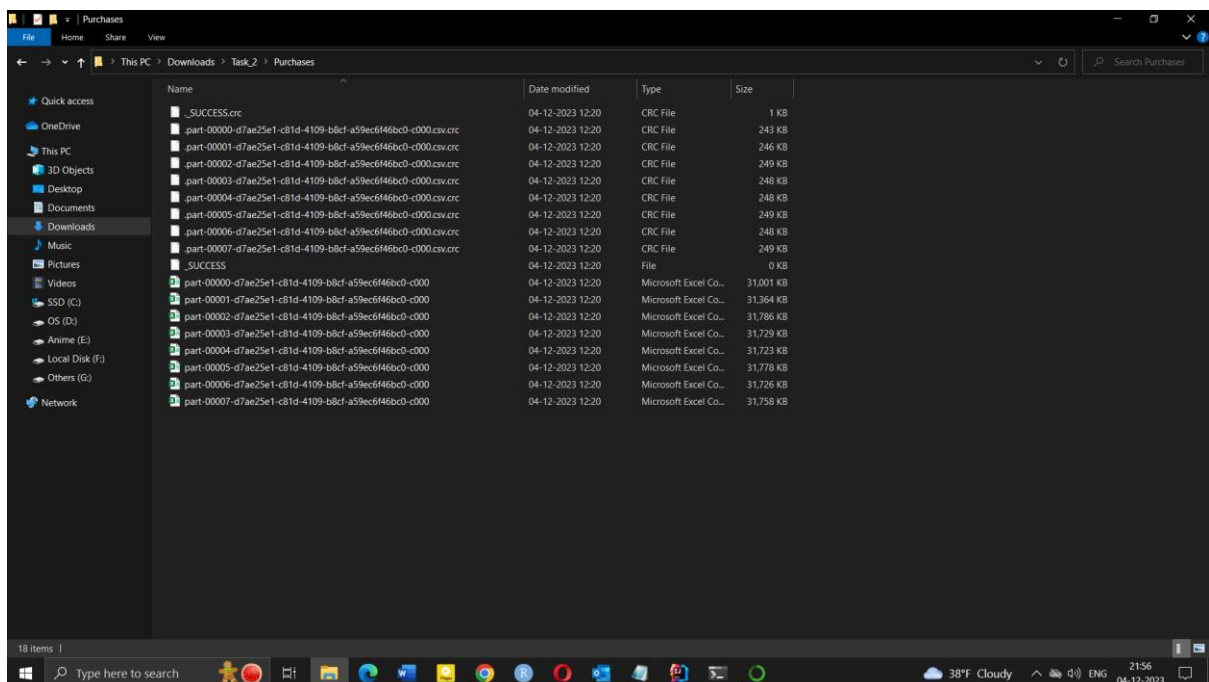
Writing Data to CSV Files:

- The `write` method is applied to both 'customers_df' and 'purchases_df' DataFrames to store the datasets in CSV format.
- The mode("overwrite") parameter ensures that any existing data in the storage location is replaced with the new datasets.
- The `header=True` parameter specifies that the CSV files should include headers.

Customers Folder: When data is stored to CSV files through Spark it is stored in multiple partitions across multiple csv files as follows:



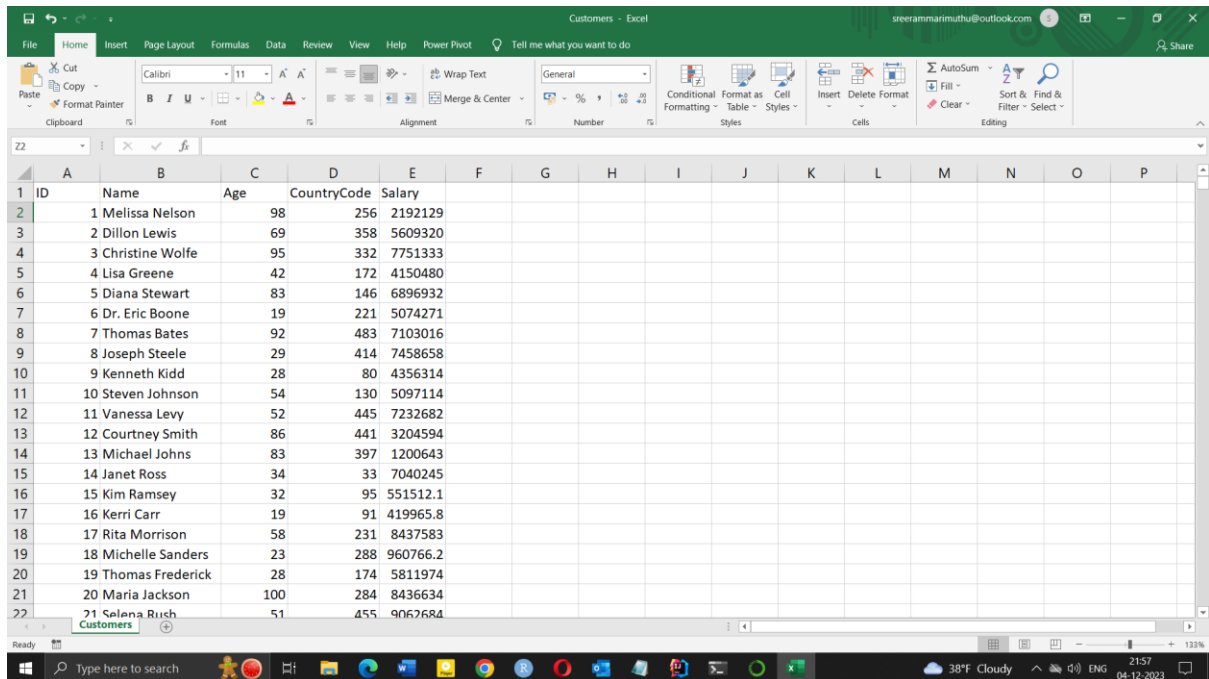
Purchases Folder: Similarly for this folder:



Pandas Conversion for Clear View:

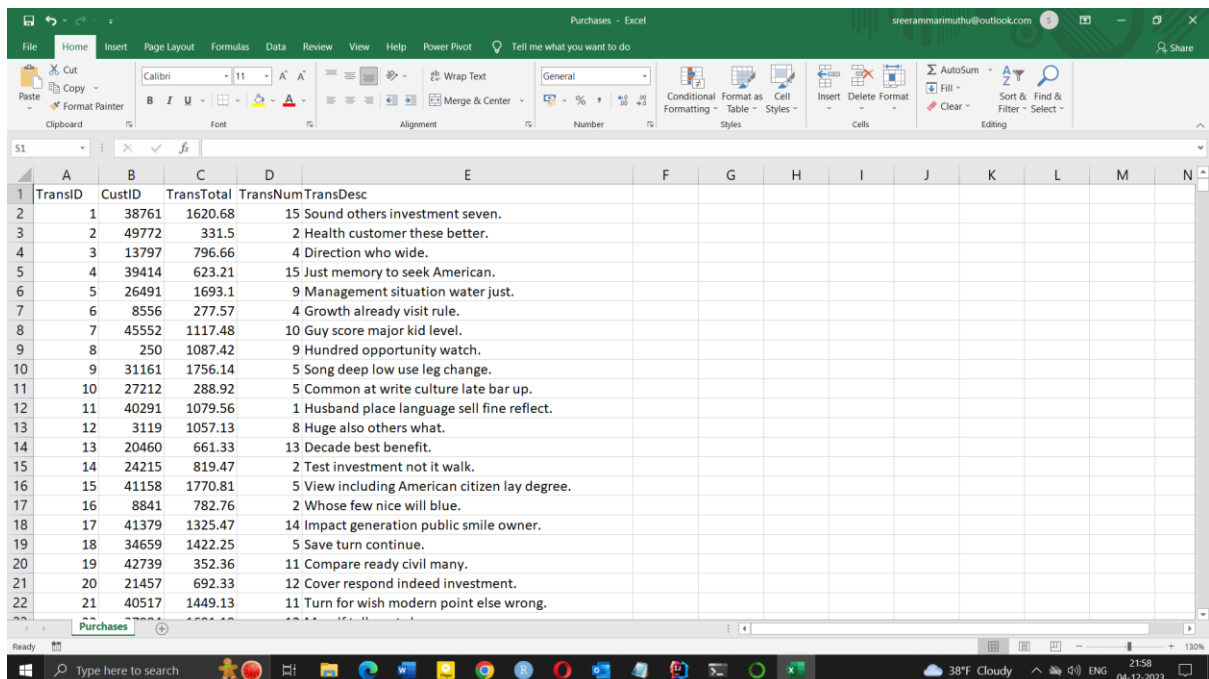
- The DataFrames are converted to Pandas DataFrames, 'customers_dff' and 'purchases_dff,' for clear and concise viewing.
- The Pandas DataFrames are then written to CSV files, 'Customers.csv' and 'Purchases.csv,' using the `to_csv` method.

Customers.csv



ID	Name	Age	CountryCode	Salary
1	Melissa Nelson	98	256	2192129
2	Dillon Lewis	69	358	5609320
3	Christine Wolfe	95	332	7751333
4	Lisa Greene	42	172	4150480
5	Diana Stewart	83	146	6896932
6	Dr. Eric Boone	19	221	5074271
7	Thomas Bates	92	483	7103016
8	Joseph Steele	29	414	7458658
9	Kenneth Kidd	28	80	4356314
10	Steven Johnson	54	130	5097114
11	Vanessa Levy	52	445	7232682
12	Courtney Smith	86	441	3204594
13	Michael Johns	83	397	1200643
14	Janet Ross	34	33	7040245
15	Kim Ramsey	32	95	551512.1
16	Kerri Carr	19	91	419965.8
17	Rita Morrison	58	231	8437583
18	Michelle Sanders	23	288	960766.2
19	Thomas Frederick	28	174	5811974
20	Maria Jackson	100	284	8436634
21	Salena Rich	51	455	9067684

Purchases.csv



TransID	CustID	TransTotal	TransNum	TransDesc
1	38761	1620.68	15	Sound others investment seven.
2	49772	331.5	2	Health customer these better.
3	13797	796.66	4	Direction who wide.
4	39414	623.21	15	Just memory to seek American.
5	26491	1693.1	9	Management situation water just.
6	8556	277.57	4	Growth already visit rule.
7	45552	1117.48	10	Guy score major kid level.
8	250	1087.42	9	Hundred opportunity watch.
9	31161	1756.14	5	Song deep low use leg change.
10	27212	288.92	5	Common at write culture late bar up.
11	40291	1079.56	1	Husband place language sell fine reflect.
12	3119	1057.13	8	Huge also others what.
13	20460	661.33	13	Decade best benefit.
14	24215	819.47	2	Test investment not it walk.
15	41158	1770.81	5	View including American citizen lay degree.
16	8841	782.76	2	Whose few nice will blue.
17	41379	1325.47	14	Impact generation public smile owner.
18	34659	1422.25	5	Save turn continue.
19	42739	352.36	11	Compare ready civil many.
20	21457	692.33	12	Cover respond indeed investment.
21	40517	1449.13	11	Turn for wish modern point else wrong.

Conclusion: The provided code successfully accomplishes the storage of the 'Customers' and 'Purchases' datasets in both CSV and Parquet formats. The use of Pandas DataFrames for clear viewing enhances the transparency and readability of the stored data, facilitating better understanding and exploration by users.

SparkSQL Workflows: [Tasks 2.1–2.4]

Task 2.1 - Filter out (drop) the Purchases from P with a total purchase amount above \$600. Store the result as T1.

Query:

The SparkSQL query aims to retrieve all columns from the 'Purchases' dataset where the 'TransTotal' (total purchase amount) is less than or equal to \$600.

Temporary View Creation:

- The result of the query is stored in a DataFrame named 't1_df'.
- The DataFrame is then converted into a temporary view named 'T1' to allow for subsequent SparkSQL operations.
- The first few rows of the T1 DataFrame are displayed using the ``show()`` method for quick visual inspection.

```
+-----+-----+-----+-----+-----+
|TransID|CustID|TransTotal|TransNumItems|TransDesc|
+-----+-----+-----+-----+-----+
| 2| 49772| 331.5| 2|Health customer t...|
| 6| 8556| 277.57| 4|Growth already vi...|
|10| 27212| 288.92| 5|Common at write c...|
|19| 42739| 352.36|11|Compare ready civ...|
|31| 3363| 526.67| 7| Sea pattern huge.|
|32| 26679| 170.93| 6|Cell career rathe...|
|36| 32405| 122.82| 9| Food surface fire.|
|37| 36329| 268.54| 5| Similar despite.|
|53| 48545| 596.66| 4|Everything coach ...|
|57| 3916| 440.4| 7|Wall financial ri...|
|59| 7551| 290.34| 3|Arrive term shoul...|
|61| 36052| 231.2| 2|Fund learn very n...|
|62| 1283| 102.6|14|Community include...|
|63| 48498| 327.02|12| Prove house food.|
|66| 1308| 543.88| 9|Region wrong econ...|
|67| 44753| 34.18| 3|However already n...|
|68| 16673| 587.24| 5|Realize campaign ...|
|72| 2920| 577.8|15|Claim us college ...|
|74| 14433| 78.29| 4|Both company worr...|
|82| 48740| 339.39|10|Medical fear stat...|
+-----+-----+-----+-----+-----+
only showing top 20 rows
```

Storing Results in CSV:

- The result DataFrame, 't1_df,' is converted to a Pandas DataFrame ('t1q_df') for easy handling and clear representation.
- The Pandas DataFrame is then written to a CSV file named 'T1_Output.csv' for external storage and further analysis.

	A	B	C	D	E	F	G	H	I	J	K
	TransID	CustID	TransTotal	TransNumItems	TransDesc						
1	2	49772	331.5	2	Health customer these better.						
2	6	8556	277.57	4	Growth already visit rule.						
3	10	27212	288.92	5	Common at write culture late bar up.						
4	19	42739	352.36	11	Compare ready civil many.						
5	31	3363	526.67	7	Sea pattern huge.						
6	32	26679	170.93	6	Cell career rather fine guy ok because film.						
7	36	32405	122.82	9	Food surface fire.						
8	37	36329	268.54	5	Similar despite.						
9	53	48545	596.66	4	Everything coach the share professor.						
10	57	3916	440.4	7	Wall financial right career avoid.						
11	59	7551	290.34	3	Arrive term should company.						
12	61	36052	231.2	2	Fund learn very never.						
13	62	1283	102.6	14	Community include right.						
14	63	48498	327.02	12	Prove house food.						
15	66	1308	543.88	9	Region wrong economy brother.						
16	67	44753	34.18	3	However already number force.						
17	68	16673	587.24	5	Realize campaign here service.						
18	72	2920	577.8	15	Claim us college provide.						
19	74	14433	78.29	4	Both company worry any.						
20											

Number of Records Check:

The count of records in the T1 DataFrame is obtained using the ``count()`` method to verify the number of filtered records. Out of 5 Million Purchases, 1.48 Million of them were below \$600.

```
In [6]: t1q_df.count() # Checking the number of records in this case

Out[6]: TransID          1481866
        CustID           1481866
        TransTotal       1481866
        TransNumItems    1481866
        TransDesc        1481866
        dtype: int64
```

Task 2.1 is successfully implemented. The SparkSQL query effectively filters out purchases with a total amount above \$600. The results are visually inspected, stored in a CSV file, and the count of records is verified for data integrity. The process ensures clarity and transparency in data manipulation and storage.

Task 2.2 - Group the Purchases in T1 by the Number of Items purchased. For each group calculate the median, min and max of total amount spent for purchases in that group.

Query:

The SparkSQL query aims to group the purchases in the T1 dataset by the 'TransNumItems' column (number of items) and calculate statistical measures such as the median, minimum, and maximum of the 'TransTotal' column (total amount spent).

Temporary View Creation:

- The result of the query is stored in a DataFrame named 't2_df.'
- A temporary view named 'T2' is created from the DataFrame for ease of reference in subsequent SparkSQL operations.
- The first few rows of the T2 DataFrame are displayed using the `show()` method for quick visual inspection.

TransNumItems	Median	Min	Max
7	303.97	10.01	600.0
6	304.58	10.0	600.0
9	305.1	10.01	600.0
5	305.31	10.0	600.0
1	305.17	10.0	600.0
10	305.49	10.0	600.0
3	304.56	10.0	600.0
12	305.74	10.01	600.0
8	304.18	10.01	600.0
11	303.72	10.0	600.0
2	305.51	10.0	599.99
4	305.63	10.01	600.0
13	302.87	10.01	600.0
14	305.44	10.0	599.99
15	303.07	10.01	600.0

Storing Results in CSV:

- The result DataFrame, 't2_df,' is converted to a Pandas DataFrame ('t2q_df') for easy handling and clear representation.
- The Pandas DataFrame is then written to a CSV file named 'T2_Output.csv' for external storage and further analysis.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	TransNum	Median	Min	Max										
2	7	303.97	10.01	600										
3	6	304.58	10	600										
4	9	305.1	10.01	600										
5	5	305.31	10	600										
6	1	305.17	10	600										
7	10	305.49	10	600										
8	3	304.56	10	600										
9	12	305.74	10.01	600										
10	8	304.18	10.01	600										
11	11	303.72	10	600										
12	2	305.51	10	599.99										
13	4	305.63	10.01	600										
14	13	302.87	10.01	600										
15	14	305.44	10	599.99										
16	15	303.07	10.01	600										
17														
18														

Number of Records Check:

The count of records in the T2 DataFrame is obtained using the `count()` method to verify the number of groups and statistical calculations.

```
In [10]: t2q_df.count() # Checking the number of records in this case
```

```
Out[10]: TransNumItems    15
         Median          15
         Min            15
         Max            15
         dtype: int64
```

Conclusion: Task 2.2 is successfully implemented. The SparkSQL query effectively groups T1 by the number of items, calculates statistics (median, min, max) for each group. The results are visually inspected, stored in a CSV file, and the count of records is verified for data integrity. This process ensures clear insights into the distribution of total amounts spent based on the number of items purchased.

Task 2.3 - Group the Purchases in T1 by customer ID only for young customers between 18 and 25 years of age. For each group report the customer ID, their age, and total number of items that this person has purchased, and total amount spent by the customer. Store the result as T3.

Query:

- The SparkSQL query aims to join the T1 and Customers datasets on 'CustID' and 'ID' respectively.
- It filters the results for customers aged between 18 and 25 and groups the purchases by 'CustID.'
- The query calculates the maximum age, total number of items, and total amount spent for each customer group.

Temporary View Creation:

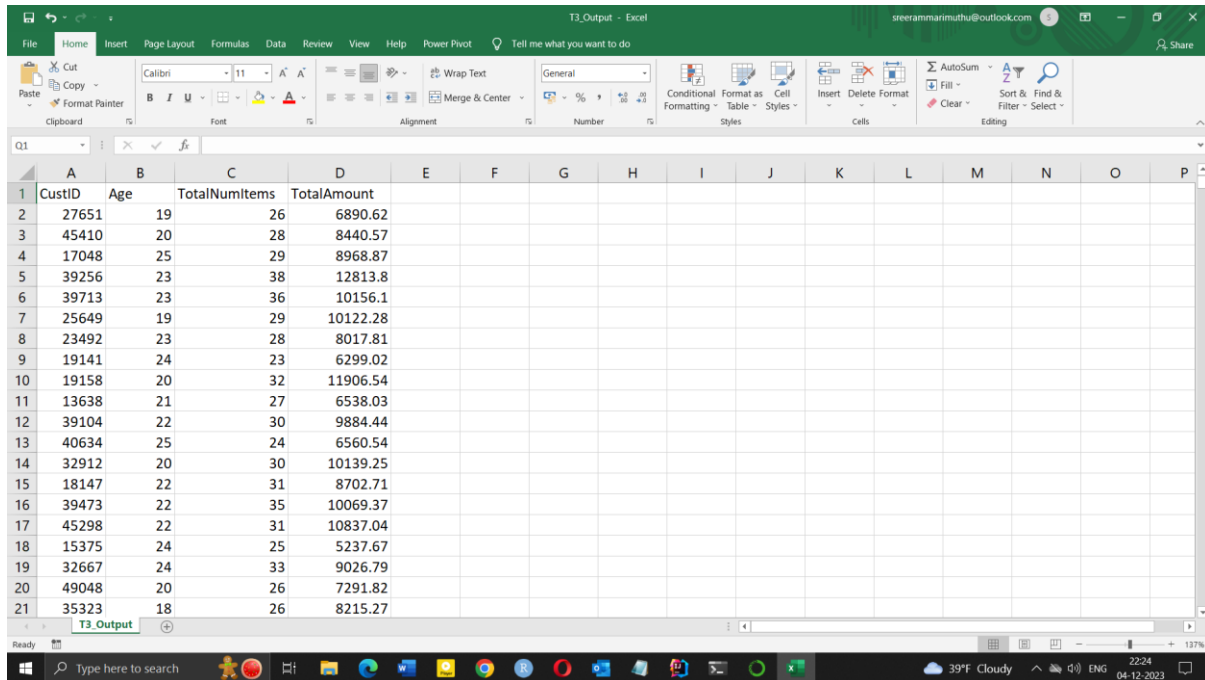
- The result of the query is stored in a DataFrame named 't3_df.'
- A temporary view named 'T3' is created from the DataFrame for subsequent SparkSQL operations.
- The first few rows of the T3 DataFrame are displayed using the `show()` method for quick visual inspection.

CustID	Age	TotalNumItems	TotalAmount
27651	19	26	6890.619999999999
45410	20	28	8440.57
17048	25	29	8968.87
39256	23	38	12813.800000000003
39713	23	36	10156.1
25649	19	29	10122.279999999999
23492	23	28	8017.809999999999
19141	24	23	6299.0199999999995
19158	20	32	11906.54
13638	21	27	6538.03
39104	22	30	9884.439999999999
40634	25	24	6560.539999999999
32912	20	30	10139.25
18147	22	31	8702.710000000001
39473	22	35	10069.37
45298	22	31	10837.039999999997
15375	24	25	5237.67
32667	24	33	9026.79
49048	20	26	7291.820000000001
35323	18	26	8215.269999999999

only showing top 20 rows

Storing Results in CSV:

- The result DataFrame, 't3_df,' is converted to a Pandas DataFrame ('t3q_df') for easy handling and clear representation.
- The Pandas DataFrame is then written to a CSV file named 'T3_Output.csv' for external storage and further analysis.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	CustID	Age	TotalNumItems	TotalAmount												
2	27651	19	26	6890.62												
3	45410	20	28	8440.57												
4	17048	25	29	8968.87												
5	39256	23	38	12813.8												
6	39713	23	36	10156.1												
7	25649	19	29	10122.28												
8	23492	23	28	8017.81												
9	19141	24	23	6299.02												
10	19158	20	32	11906.54												
11	13638	21	27	6538.03												
12	39104	22	30	9884.44												
13	40634	25	24	6560.54												
14	32912	20	30	10139.25												
15	18147	22	31	8702.71												
16	39473	22	35	10069.37												
17	45298	22	31	10837.04												
18	15375	24	25	5237.67												
19	32667	24	33	9026.79												
20	49048	20	26	7291.82												
21	35323	18	26	8215.27												

Number of Records Check:

The count of records in the T3 DataFrame is obtained using the `count()` method to verify the number of customer groups and calculated metrics.

```
In [14]: t3q_df.count() # Checking the number of records in this case
```

```
Out[14]: CustID      4813
         Age         4813
         TotalNumItems 4813
         TotalAmount   4813
         dtype: int64
```

Conclusion: Task 2.3 is successfully implemented. The SparkSQL query effectively joins T1 and Customers datasets, filters for young customers, and calculates metrics for each customer group. The results are visually inspected, stored in a CSV file, and the count of records is verified for data integrity. This process provides insights into the purchasing behavior of young customers based on total items and amount spent.

Task 2.4 - Return all customer pairs IDs (C1 and C2) from T3 such that

- a) C1 is younger in age than customer C2 and
- b) C1 spent in total more money than C2 but bought less items.

Store the result as T4 and report it back in the form (C1 ID, C2 ID, Age1, Age2, TotalAmount1, TotalAmount2, TotalItemCount1, TotalItemCount2) to the client side.

Query:

- The SparkSQL query (query_t4) performs a self-join on the T3 dataset (temporary view T3) to identify pairs of customers (C1 and C2).
- The conditions for selecting customer pairs are:
 - a. C1 is younger in age than customer C2.
 - b. C1 spent more money in total than C2 & bought fewer items than C2.
- The query extracts relevant attributes from both customer records to form the resulting DataFrame, 't4_df.'

Temporary View Creation:

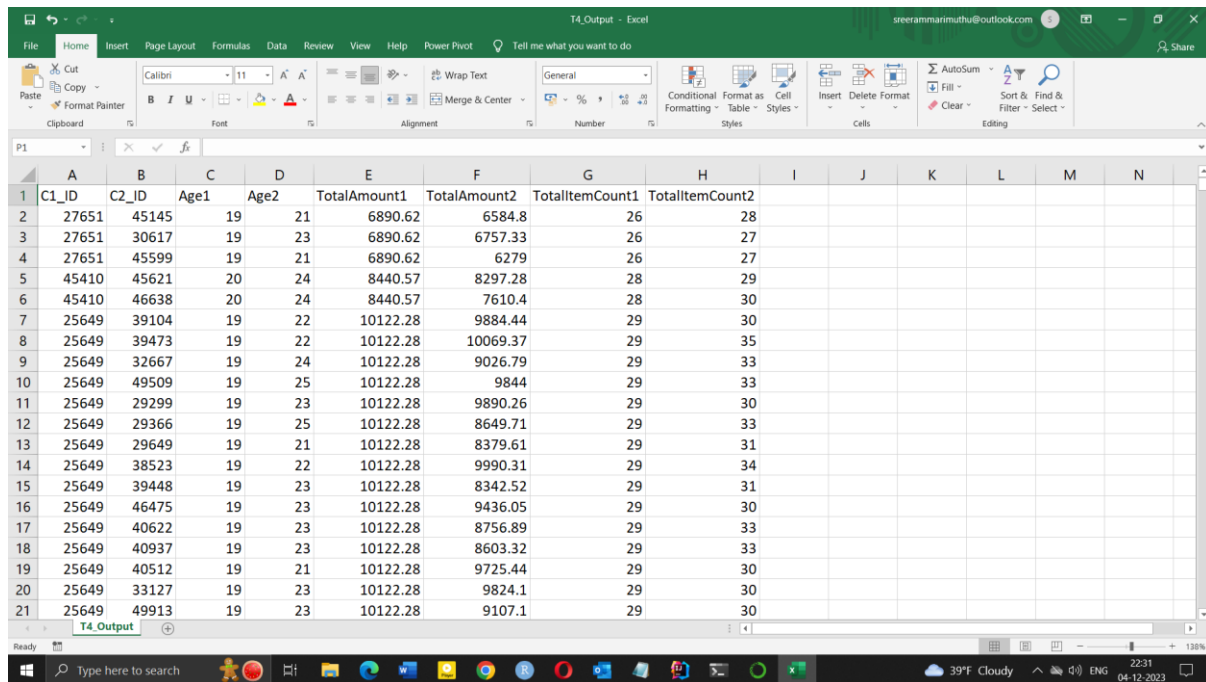
- The result of the query is stored in a DataFrame named 't4_df.'
- No additional temporary view is created for T4, as the results are directly utilized in the subsequent steps.
- The first few rows of the T4 DataFrame are displayed using the `show()` method for quick visual inspection.

C1_ID	C2_ID	Age1	Age2	TotalAmount1	TotalAmount2	TotalItemCount1	TotalItemCount2
27651	45145	19	21	6890.619999999999	6584.800000000001	26	28
27651	30617	19	23	6890.619999999999	6757.33	26	27
27651	45599	19	21	6890.619999999999	6279.000000000001	26	27
45410	45621	20	24	8440.57	8297.279999999999	28	29
45410	46638	20	24	8440.57	7610.399999999998	28	30
25649	39104	19	22	10122.279999999999	9884.439999999999	29	30
25649	39473	19	22	10122.279999999999	10069.37	29	35
25649	32667	19	24	10122.279999999999	9026.79	29	33
25649	49509	19	25	10122.279999999999	9844.0	29	33
25649	29299	19	23	10122.279999999999	9890.26	29	30
25649	29366	19	25	10122.279999999999	8649.710000000001	29	33
25649	29649	19	21	10122.279999999999	8379.609999999997	29	31
25649	38523	19	22	10122.279999999999	9990.309999999998	29	34
25649	39448	19	23	10122.279999999999	8342.52	29	31
25649	46475	19	23	10122.279999999999	9436.050000000001	29	30
25649	40622	19	23	10122.279999999999	8756.89	29	33
25649	40937	19	23	10122.279999999999	8603.32	29	33
25649	40512	19	21	10122.279999999999	9725.440000000002	29	30
25649	33127	19	23	10122.279999999999	9824.100000000002	29	30
25649	49913	19	23	10122.279999999999	9107.1	29	30

only showing top 20 rows

Storing Results in CSV:

- The result DataFrame, 't4_df,' is converted to a Pandas DataFrame ('t4q_df') for easy handling and clear representation.
- The Pandas DataFrame is then written to a CSV file named 'T4_Output.csv' for external storage and further analysis.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	C1_ID	C2_ID	Age1	Age2	TotalAmount1	TotalAmount2	TotalItemCount1	TotalItemCount2						
2	27651	45145	19	21	6890.62	6584.8	26	28						
3	27651	30617	19	23	6890.62	6757.33	26	27						
4	27651	45599	19	21	6890.62	6279	26	27						
5	45410	45621	20	24	8440.57	8297.28	28	29						
6	45410	46638	20	24	8440.57	7610.4	28	30						
7	25649	39104	19	22	10122.28	9884.44	29	30						
8	25649	39473	19	22	10122.28	10069.37	29	35						
9	25649	32667	19	24	10122.28	9026.79	29	33						
10	25649	49509	19	25	10122.28	9844	29	33						
11	25649	29299	19	23	10122.28	9890.26	29	30						
12	25649	29366	19	25	10122.28	8649.71	29	33						
13	25649	29649	19	21	10122.28	8379.61	29	31						
14	25649	38523	19	22	10122.28	9990.31	29	34						
15	25649	39448	19	23	10122.28	8342.52	29	31						
16	25649	46475	19	23	10122.28	9436.05	29	30						
17	25649	40622	19	23	10122.28	8756.89	29	33						
18	25649	40937	19	23	10122.28	8603.32	29	33						
19	25649	40512	19	21	10122.28	9725.44	29	30						
20	25649	33127	19	23	10122.28	9824.1	29	30						
21	25649	49913	19	23	10122.28	9107.1	29	30						

Number of Records Check:

- The count of records in the T4 DataFrame is obtained using the `count()` method to verify the number of customer pairs meeting the specified conditions.

```
In [18]: t4q_df.count() # Checking the number of records in this case

Out[18]: C1_ID      343226
          C2_ID      343226
          Age1       343226
          Age2       343226
          TotalAmount1 343226
          TotalAmount2 343226
          TotalItemCount1 343226
          TotalItemCount2 343226
          dtype: int64
```

Conclusion: Task 2.4 is successfully implemented. The SparkSQL query effectively identifies and selects customer pairs based on specified conditions. The results are visually inspected, stored in a CSV file, and the count of records is verified for data integrity. This process provides insights into customer pairs with differing spending and item purchase behaviour.

MLlib Workflows: [Task 2.5–2.9]

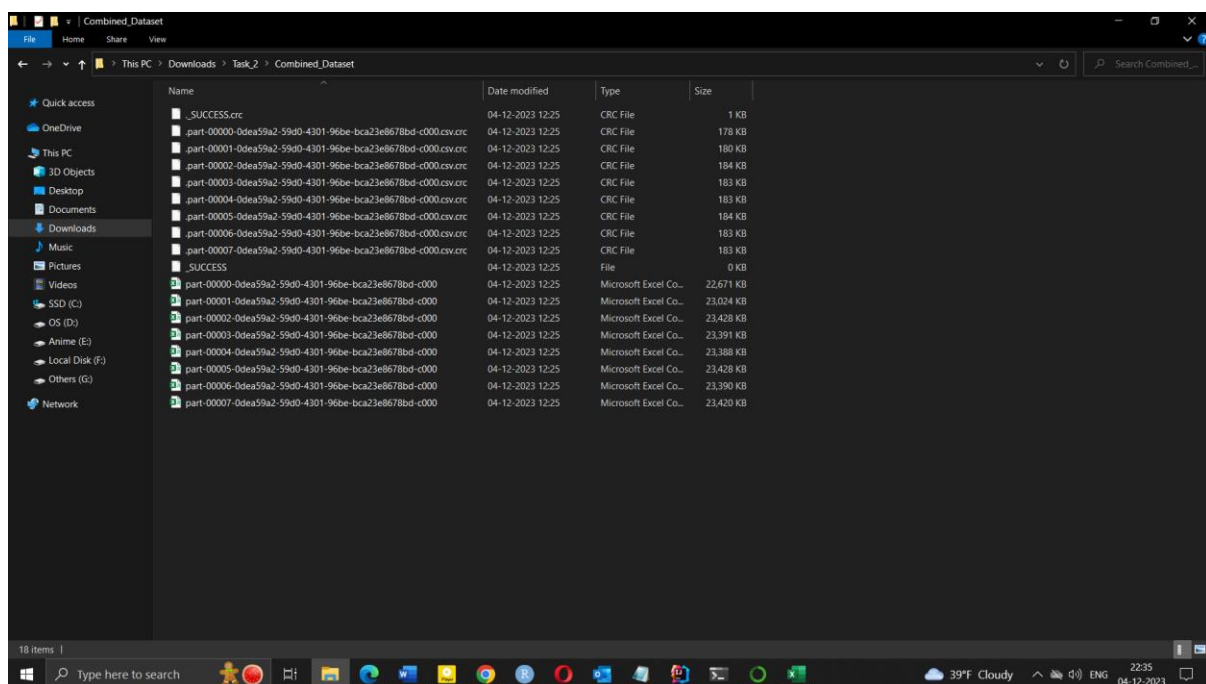
Task 2.5 - Data Preparation 1: Generate a dataset composed of customer ID, TransID, Age, Salary, TransNumItems and TransTotal. Store it as Dataset.

Data Generation:

- The workflow starts by creating a new DataFrame, `combined_dataset_df`, through a join operation between the `purchases_df` and `customers_df` DataFrames.
- The join is performed based on the common column, "CustID," and selects specific attributes, including customer ID, transaction ID, age, salary, number of items, and total transaction amount.

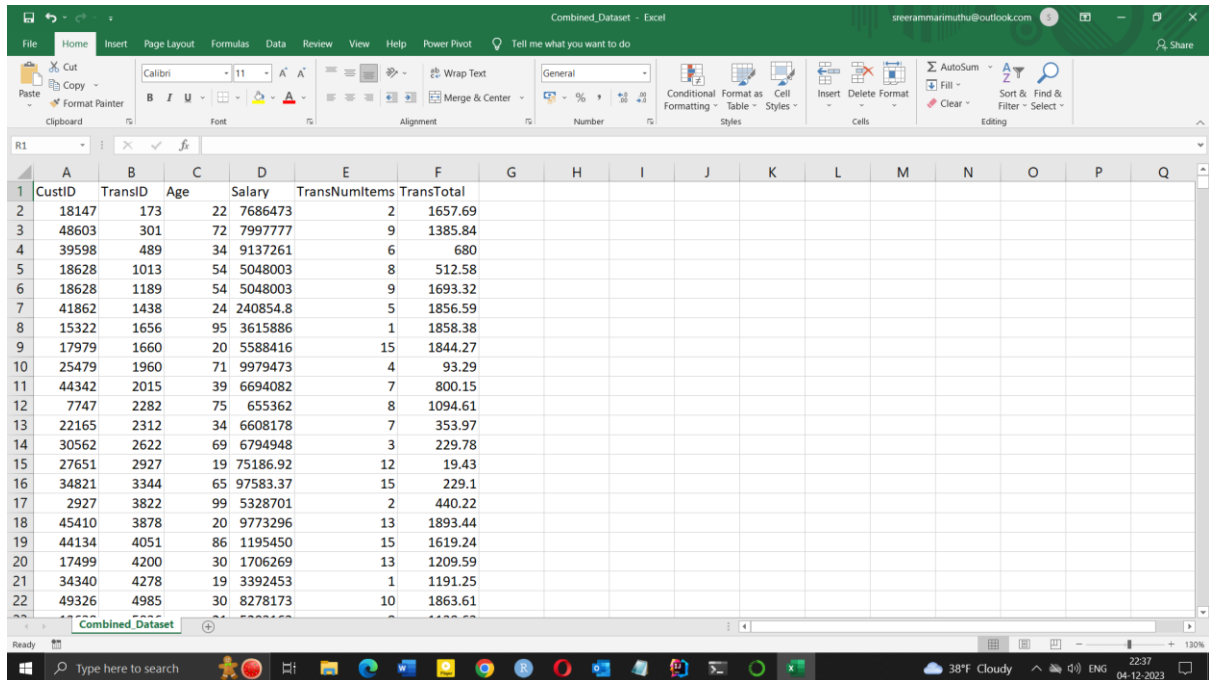
Storing the Dataset:

The resulting DataFrame, `combined_dataset_df`, is stored as a CSV file named "Combined_Dataset" using the `write` method with the "overwrite" mode. Similar to the Customers & Purchases here as you can when writing the file locally from Spark they are stored in multiple partitions.



Storing in CSV for Clear View:

Additionally, the DataFrame is converted to a Pandas DataFrame, `combined_ds`, for clearer visualization, and the Pandas DataFrame is stored as a CSV file named "Combined_Dataset.csv."



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	CustID	TransID	Age	Salary	TransNumItems	TransTotal											
2	18147	173	22	7686473	2	1657.69											
3	48603	301	72	7997777	9	1385.84											
4	39598	489	34	9137261	6	680											
5	18628	1013	54	5048003	8	512.58											
6	18628	1189	54	5048003	9	1693.32											
7	41862	1438	24	240854.8	5	1856.59											
8	15322	1656	95	3615886	1	1858.38											
9	17979	1660	20	5588416	15	1844.27											
10	25479	1960	71	9979473	4	93.29											
11	44342	2015	39	6694082	7	800.15											
12	7747	2282	75	655362	8	1094.61											
13	22165	2312	34	6608178	7	353.97											
14	30562	2622	69	6794948	3	229.78											
15	27651	2927	19	75186.92	12	19.43											
16	34821	3344	65	97583.37	15	229.1											
17	2927	3822	99	5328701	2	440.22											
18	45410	3878	20	9773296	13	1893.44											
19	44134	4051	86	1195450	15	1619.24											
20	17499	4200	30	1706269	13	1209.59											
21	34340	4278	19	3392453	1	1191.25											
22	49326	4985	30	8278173	10	1863.61											

Conclusion: Task 2.5 is successfully implemented, and the dataset composed of the specified attributes is generated and stored. The resulting CSV files, "Combined_Dataset" and "Combined_Dataset.csv," contain information necessary for subsequent machine learning tasks, providing a consolidated view of customer transactions and attributes.

Task 2.6 - Data Preparation 2: (Randomly) split Dataset into two subsets, namely, Trainset & Testset, such that Trainset contains 80% of Dataset and Testset the remaining 20%.

Random Splitting:

- The dataset, ``combined_dataset_df``, is randomly split into two subsets, namely ``train_df`` and ``test_df``.
- The split is performed using the ``randomSplit`` method, specifying the split ratios of 80% for the training set and 20% for the testing set.
- The seed parameter is set to 42 to ensure reproducibility.

Dataset Composition:

- The resulting DataFrames, ``train_df`` and ``test_df``, now represent the training and testing sets, respectively.
- The training set comprises 80% of the original dataset, and the testing set comprises the remaining 20%.

Conclusion:

Task 2.6 is successfully implemented, and the dataset is effectively split into training and testing sets based on the specified ratios. The split datasets, ``train_df`` and ``test_df``, are ready for further use in training and evaluating machine learning models.

Task 2.7 - Predict the price: Identify and train at least 3 machine learning algorithms to predict TransTotal. Specifically, treat this problem as a regression model in which the "features" (X) for this task are Age, Salary, TransNumItems and the "outcome" (Y) is TransTotal. The algorithms should be trained over Trainset (Task 2.6) and applied (inference) over Testset.

This section provides a comprehensive overview and analysis of the process of training and applying regression models to predict `TransTotal` in the context of Task 2.7. We decided to train four algorithms for this task as follow:

Data Preparation: Feature Vector

- Feature columns, including "Age," "Salary," and "TransNumItems," are selected for building the feature vector.
- A `VectorAssembler` is employed to assemble these features into a vector named "features."

Description for each of the four regression algorithms used in Spark: Linear Regression, Decision Tree Regression, Random Forest Regression, and Gradient-Boosted Tree Regression.

Linear Regression Model:

Linear Regression is a fundamental and widely used algorithm in statistical modeling and machine learning. It models the relationship between the dependent variable (target) and one or more independent variables (features) by fitting a linear equation to the observed data.

The Linear Regression model is initialized and fitted to the training set (`train_df`). Predictions are made on the test set (`test_df`).

Rationale for Selection:

- Linear Regression provides a straightforward interpretation of the relationship between input features and the target variable. Coefficients

assigned to each feature offer insights into the impact of each variable on the target.

- It is computationally efficient, making it suitable for large datasets. Its simplicity makes it a good baseline model for regression tasks.
- Linear Regression assumptions (linearity, independence, homoscedasticity, normality) can be examined, contributing to a better understanding of the data.

Decision Tree Model:

Decision Tree Regression is a non-linear model that recursively splits the dataset based on the features to predict the target variable. It forms a tree-like structure where each internal node represents a decision based on a feature, and each leaf node represents the predicted outcome.

The Decision Tree model is initialized, fitted, and evaluated similarly to the Linear Regression model.

Rationale for Selection:

- Decision Trees can capture complex relationships between features and the target variable that linear models might miss.
- It provides a measure of feature importance, helping identify the most influential variables in the prediction.
- The decision-making process of a tree is easily interpretable, making it useful for explaining the model to stakeholders.

Random Forest Model:

Random Forest Regression is an ensemble method that builds multiple decision trees and combines their predictions to improve accuracy and reduce overfitting. It introduces randomness both in the selection of data samples and the features considered for each split.

The Random Forest model is initialized, fitted, and evaluated similarly to the previous models.

Rationale for Selection:

- Random Forest leverages the power of multiple trees to enhance predictive performance and generalization.
- The randomness in feature selection and data sampling helps mitigate overfitting, making it more robust on diverse datasets.
- Random Forest can handle large-scale datasets and parallelize the training process, making it suitable for distributed computing environments like Spark.

GBT Model (Gradient-Boosted Tree):

Gradient-Boosted Tree Regression is another ensemble method that builds a sequence of decision trees, each correcting the errors of the previous one. It combines weak learners into a strong predictive model.

The GBT model is initialized, fitted, and evaluated similarly to the previous models.

Rationale for Selection:

- Gradient-Boosted Trees sequentially improve model accuracy by focusing on previously misclassified instances.
- Gradient-Boosted Trees often yield highly accurate predictions and are capable of capturing complex relationships.
- Like Random Forest, Gradient-Boosted Trees can model non-linear relationships effectively.

Conclusion:

- All four regression models are successfully trained and applied to predict `TransTotal`.
- Evaluation metrics provide insights into the accuracy and performance of each model on the test set.
- Further analysis can guide the selection of the most suitable model for predicting transaction totals based on customer attributes.

This project aims to showcase the scalability and efficiency of Spark for processing large-scale datasets and implementing machine learning workflows.

Task 2.8 - Predict the price: Identify and use at least 3 metrics to evaluate the algorithms from Task 2.7.

Analysing Metrics:

- The metrics (RMSE, MAE, and R-squared) for each model are printed for analysis and comparison.
- Metrics provide insights into the performance of each model in predicting `TransTotal` on the test set.

The metrics RMSE (Root Mean Squared Error), MAE (Mean Absolute Error), and R-squared are commonly used in regression analysis to evaluate the performance of predictive models. They provide different perspectives on the accuracy and goodness-of-fit of a regression model. When comparing regression models in Spark or any other environment, these metrics are often considered for the following reasons:

1. RMSE (Root Mean Squared Error):

- RMSE is the square root of the average of squared differences between the predicted and actual values.
- It measures the average magnitude of the errors between predicted and actual values. Smaller RMSE values indicate better model performance.
- RMSE penalizes larger errors more significantly, making it sensitive to outliers. This is important because large errors can have a disproportionate impact on the model's performance.

2. MAE (Mean Absolute Error):

- MAE is the average of the absolute differences between the predicted and actual values.
- It provides a measure of the average absolute error, ignoring the direction of errors. Like RMSE, smaller MAE values indicate better model performance.
- MAE is less sensitive to outliers compared to RMSE since it does not involve squaring the errors. This can be useful when the dataset contains extreme values that might unduly influence the RMSE.

3. R-squared (Coefficient of Determination):

- R-squared measures the proportion of the variance in the dependent variable that is predictable from the independent variables. It ranges from 0 to 1.
- A higher R-squared indicates a better fit of the model to the data. It represents the percentage of variation in the dependent variable explained by the independent variables.
- R-squared provides an overall measure of model fit and helps in understanding how well the model explains the variability in the target variable. However, it's important to note that R-squared should be used in conjunction with other metrics, as it may not reveal issues such as overfitting.

Linear Regression Model – Metrics:

Linear Regression Metrics:

RMSE: 574.3698283992732, MAE: 497.4818900678744, R^2 : -2.815956533286368e-06

Decision Tree Model – Metrics:

Decision Tree Metrics:

RMSE: 574.3800349482891, MAE: 497.4878180779392, R^2 : -3.835636251503516e-05

Random Forest Model – Metrics:

RandomForest Metrics:

RMSE: 574.3703855447804, MAE: 497.4820522243702, R^2 : -4.755986523941047e-06

GBT Model (Gradient-Boosted Tree) – Metrics:

GBT Metrics:

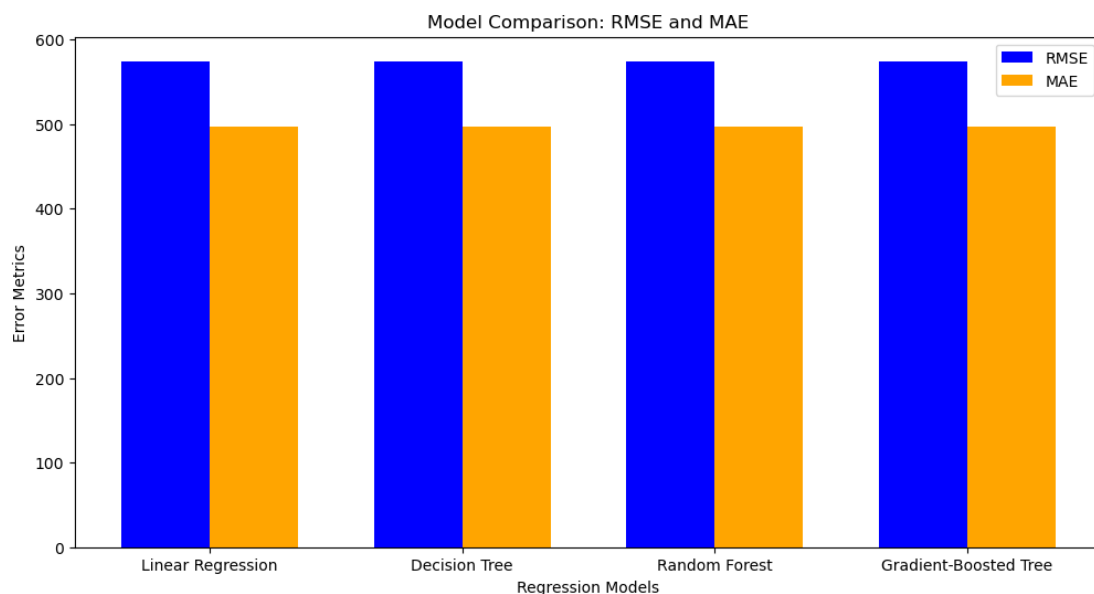
RMSE: 574.3878801488834, MAE: 497.4911068734467, R^2 : -6.567470441232182e-05

Task 2.9 - Report and discuss the results: Analyse the experimental results and report your conclusion.

When comparing regression models, using a combination of these metrics allows for a more comprehensive evaluation. No single metric is perfect for every scenario, and considering multiple metrics provides a more nuanced understanding of the strengths and weaknesses of each model. It's recommended to assess these metrics together to make informed decisions about the performance of regression models in Spark or any other environment.

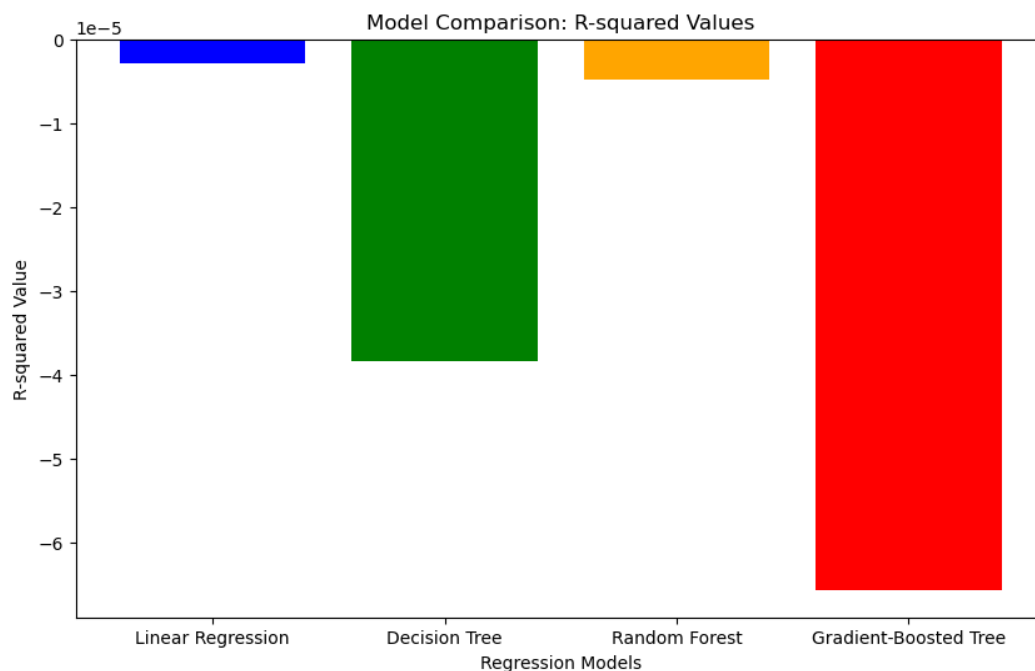
Comparison Table:

Model	RMSE	MAE	R-squared
Linear Regression	574.3698	497.4819	-2.82E-06
Decision Tree	574.38	497.4878	-3.84E-05
Random Forest	574.3704	497.4821	-4.76E-06
Gradient-Boosted Tree	574.3879	497.4911	-6.57E-05



Consistency across Models:

All four models (Linear Regression, Decision Tree, Random Forest, Gradient-Boosted Tree) exhibit very similar performance in terms of RMSE and MAE. The values are almost identical, suggesting that there is no significant difference in their predictive accuracy.



R-squared Values:

The R-squared values for all models are close to zero or negative, (Linear Regression being the closest to zero) indicating that the models are not explaining much of the variance in the target variable. This might suggest that the chosen features or the models themselves are not capturing the underlying patterns well.

Model Efficiency:

In terms of efficiency, the computational complexity of training and making predictions might differ among the models. However, based on the provided metrics alone, it's challenging to determine a clear winner in terms of efficiency.

Consideration for Further Investigation:

The low R-squared values and the similarity in RMSE and MAE across models suggest that there may be room for improvement. Further investigation could

involve feature engineering, hyperparameter tuning, or exploring more complex models to enhance predictive performance.

Model Selection: Since the performance metrics are quite close, the choice of the final model may also depend on factors such as interpretability, ease of implementation, and the specific requirements of the task.

Preferred Model: Given the provided metrics and the lack of a clear standout performer, we may initially lean towards a simpler model like Linear Regression for its interpretability and efficiency. However, it's essential to acknowledge that these models might not have reached their full potential, and further exploration and optimization could be worthwhile.

Conclusion:

In summary, the comparison of the regression models reveals that, based on the provided metrics alone, there is no substantial difference in their predictive performance. The choice of the final model may depend on factors beyond these metrics, and further exploration is recommended to potentially enhance the models' ability to capture the underlying patterns in the data. Additionally, considering the low R-squared values, it might be worthwhile to reassess the feature selection process and explore more advanced modeling techniques for improved results.

Team Member Contributions: Sreeram (myself) and Santosh collaborated and done the entire project work completely between us side by side.

Yours Truly,

Sreeram Marimuthu

Nitya Phani Santosh Oruganty

Anushka Bangal

December 4th 2023