

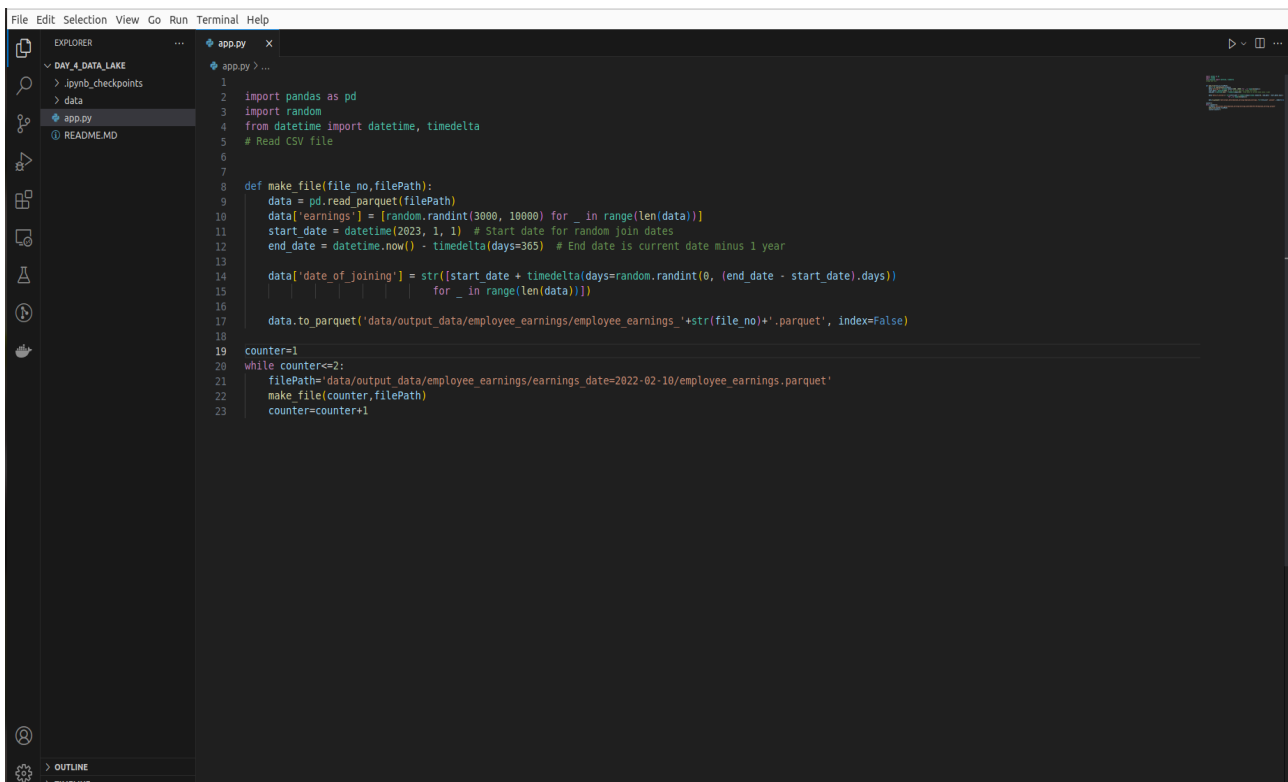
Group Member's Name:-

Muhammad Arsalan (2303.KHI.DEG.025)

Abdul Rehman (2303.KHI.DEG.035)

Arshad Shiwani (2303.KHI.DEG.026)

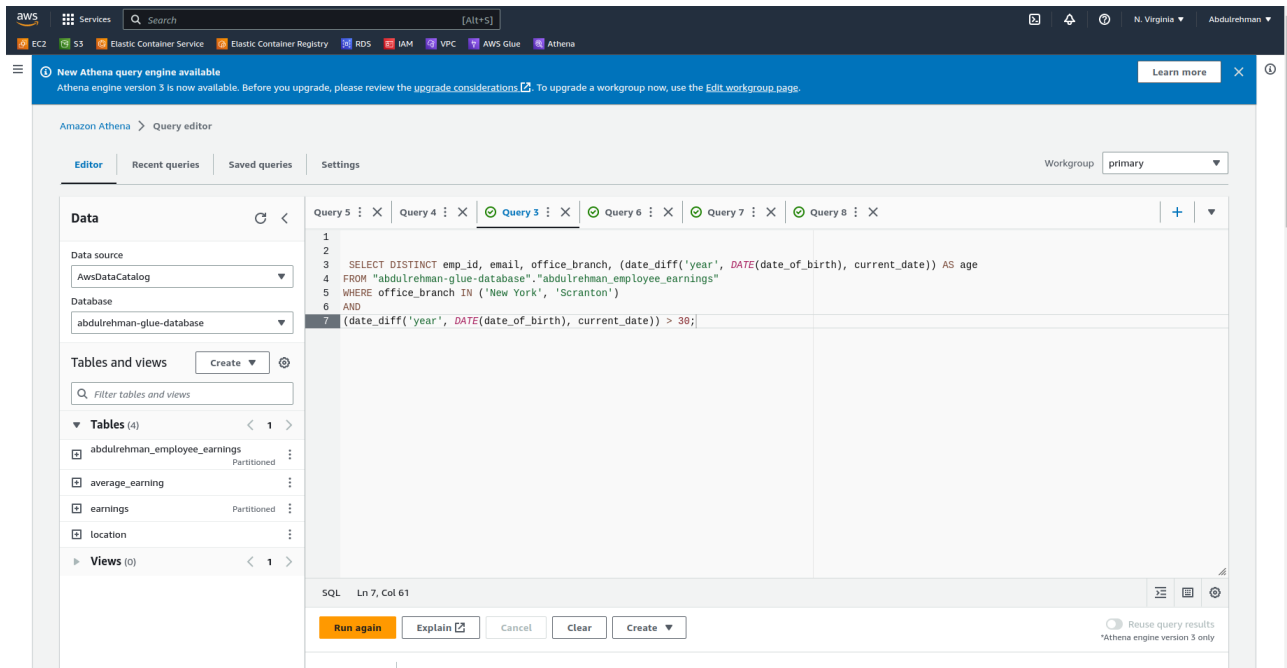
We made a function with name `make_file`, which will read the file and make random salary and date and then save the file, we here required two files, so we loop it twice.



```
File Edit Selection View Go Run Terminal Help
EXPLORER
  DAY_4_DATA_LAKE
    .ipynb_checkpoints
    data
    app.py
    README.MD
  OUTLINE
  TERMINAL

app.py
1
2 import pandas as pd
3 import random
4 from datetime import datetime, timedelta
5 # Read CSV file
6
7
8 def make_file(file_no, filePath):
9     data = pd.read_parquet(filePath)
10    data['earnings'] = [random.randint(3000, 10000) for _ in range(len(data))]
11    start_date = datetime(2023, 1, 1) # Start date for random join dates
12    end_date = datetime.now() - timedelta(days=365) # End date is current date minus 1 year
13
14    data['date_of_joining'] = str((start_date + timedelta(days=random.randint(0, (end_date - start_date).days))
15                                for _ in range(len(data))))
16
17    data.to_parquet('data/output_data/employee_earnings/employee_earnings_'+str(file_no)+'.parquet', index=False)
18
19 counter=1
20 while counter<=2:
21     filePath='data/output_data/employee_earnings/earnings_date=2022-02-10/employee_earnings.parquet'
22     make_file(counter,filePath)
23     counter=counter+1
```

This SQL query retrieves distinct employee IDs, email addresses, office branches, and ages from a table named "**abdulrehman_employee_earnings**" in a database called "**abdulrehman-glue-database.**" The query selects only those employees who work in either the "**New York**" or "**Scranton**" office branches and have an age greater than **30**.



The screenshot shows the 'Query results' page for the executed query. The status is 'Completed' with a time in queue of 129 ms, run time of 779 ms, and data scanned of 26.66 KB. The results are displayed as a table with 46 rows. The table has columns: #, emp_id, email, office_branch, and age. The first 15 rows are shown below.

#	emp_id	email	office_branch	age
1	654617	rogetio.woodall@gmail.com	New York	50
2	138911	claudio.heck@aol.com	Scranton	55
3	713294	sammy.dewitt@ibm.com	Scranton	35
4	312726	celine.lumpkin@gmail.com	New York	36
5	551149	michale.colson@comcast.net	Scranton	61
6	402180	allan.bernhardt@gmail.com	New York	61
7	823898	carlton.leclair@cox.net	Scranton	37
8	962291	whitney.shipman@gmail.com	Scranton	40
9	856379	terence.ferro@gmail.com	Scranton	41
10	391837	cory.hayden@gmail.com	New York	56
11	622405	harrison.hawk@hotmail.co.uk	Scranton	60
12	595558	denisha.ftch@msn.com	Scranton	32
13	314661	charles.quintero@gmail.com	New York	65
14	220965	almeta.brookins@gmail.com	Scranton	38
15	537591	samuel.wendt@bellsouth.net	New York	46

This SQL query retrieves the office branch, minimum earnings, maximum earnings, average earnings, total earnings, and earnings date from the "abdulrehman_employee_earnings" table in the "abdulrehman-glue-database" database. The results are grouped by office branch and earnings date. The records are then sorted in descending order based on the sum of earnings.

The screenshot shows the Amazon Athena Query Editor interface. On the left, the 'Data' panel shows the 'Data source' as 'AwsDataCatalog', the 'Database' as 'abdulrehman-glue-database', and a list of tables including 'abdulrehman_employee_earnings', 'average_earning', 'earnings', and 'location'. The 'Query editor' tab is active, displaying a SQL query:

```
1 SELECT office_branch, MIN(earnings) as min_earnings, MAX(earnings) as max_earnings, AVG(earnings) as avg_earnings, SUM(earnings) as total_earnings,
2 earnings_date
3 FROM "abdulrehman-glue-database"."abdulrehman_employee_earnings"
4 GROUP BY office_branch, earnings_date
5 ORDER BY SUM(earnings) desc;
```

The 'Run again' button is highlighted in orange. The 'Query stats' tab is also visible.

The screenshot shows the 'Query results' page for the executed query. The query is marked as 'Completed' with a status of 'Completed'. The results are displayed in a table with 8 columns: #, office_branch, min_earnings, max_earnings, avg_earnings, total_earnings, and earnings_date. The table contains 15 rows of data.

#	office_branch	min_earnings	max_earnings	avg_earnings	total_earnings	earnings_date
1	Nashua	3014	9970	6658.774193548387	206422	2022-02-16
2	Nashua	3088	9693	6628.419354838709	205481	2022-02-15
3	New York	3246	9756	7308.392857142857	204635	2022-02-15
4	New York	3151	9930	6796.214285714285	190294	2022-02-16
5	Nashua	2098	9728	6099.8387096774195	189095	2022-02-14
6	Nashua	2005	9786	6049.451612903225	187533	2022-02-13
7	Nashua	2006	9603	5997.967741935484	185937	2022-02-11
8	New York	2295	9889	6631.285714285715	185676	2022-02-12
9	Nashua	2124	9978	5764.5161290322585	178700	2022-02-12
10	Nashua	2066	9801	5619.903225806452	174217	2022-02-10
11	New York	2040	9954	6109.035714285715	171053	2022-02-14
12	Scranton	2788	9916	6830.6	170765	2022-02-13
13	New York	2141	9462	5998.178571428572	167949	2022-02-11
14	New York	2376	9972	5991.521428571428	167757	2022-02-10
15	New York	2195	9734	5615.535714285715	157235	2022-02-13

This SQL query calculates the earnings range for each office branch by finding the difference between the maximum and minimum average earnings per branch. The results are grouped by office branch.

The screenshot shows the Amazon Athena Query Editor interface. On the left, the 'Data' panel displays the 'abdurrehman-glue-database' with tables 'abdurrehman_employee_earnings', 'average_earning', 'earnings', and 'location'. The main editor shows a SQL query with 8 lines. The query calculates the earnings range for each office branch by finding the difference between the maximum and minimum average earnings per branch. The query is executed using the 'Run again' button.

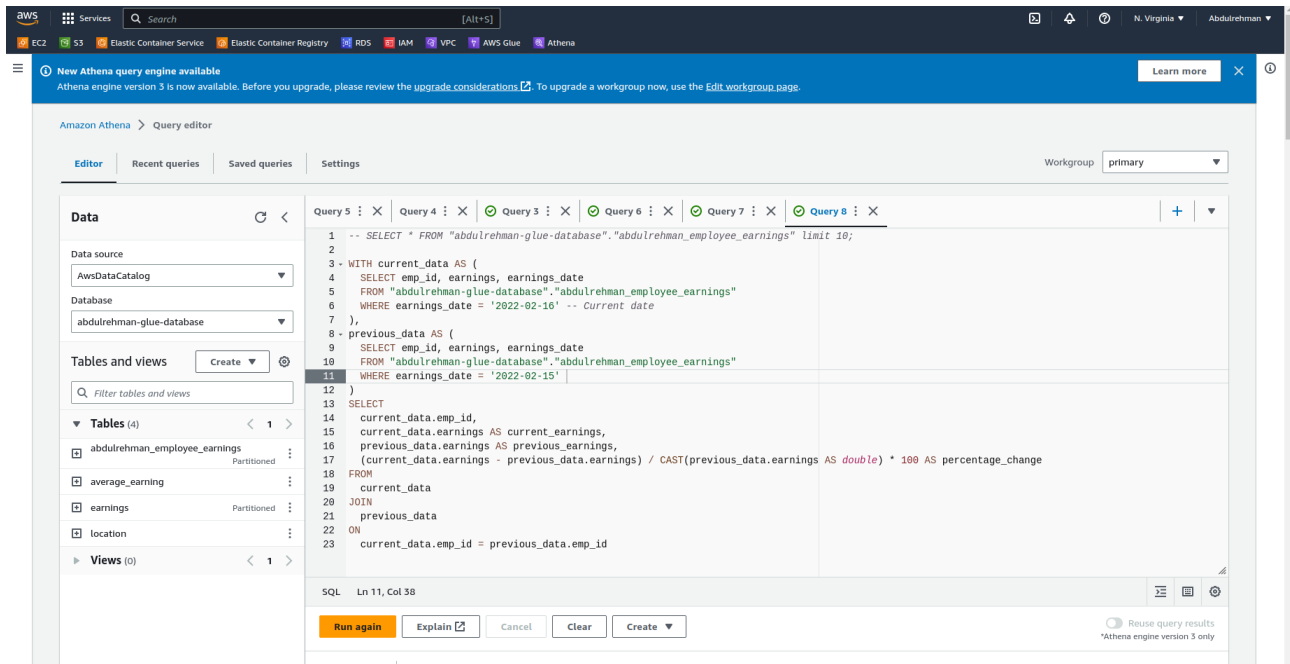
```
1 -- SELECT * FROM "abdurrehman-glue-database"."abdurrehman_employee_earnings" limit 10;
2
3 SELECT DISTINCT office_branch, (MAX(avg_earnings.value) - MIN(avg_earnings.value)) as earnings_range
4 FROM (
5 SELECT office_branch as ob, AVG(earnings) AS value FROM "abdurrehman-glue-database"."abdurrehman_employee_earnings" GROUP BY office_branch,
6 earnings_date
7 ) avg_earnings, "abdurrehman-glue-database"."abdurrehman_employee_earnings"
8 WHERE office_branch = avg_earnings.ob
9 GROUP BY office_branch;
```

S

The screenshot shows the Amazon Athena Query Editor interface with the query results displayed. The 'Query results' tab is active, showing a 'Completed' status with a green checkmark. The results are grouped by office branch, showing the earnings range for each branch. The results are displayed in a table with 4 rows and 2 columns: 'office_branch' and 'earnings_range'.

#	office_branch	earnings_range
1	Scranton	1779.2800000000007
2	Nashua	1038.8709677419356
3	New York	1692.8571428571422
4	Stanford	1053.375

This SQL query compares employee earnings from two different dates and calculates the percentage change.



The screenshot shows the AWS Athena Query Results page. The query is completed, with a time in queue of 184 ms, a run time of 951 ms, and data scanned of 2.61 KB. The results are displayed in a table with 15 rows and 5 columns: emp_id, current_earnings, previous_earnings, and percentage_change.

#	emp_id	current_earnings	previous_earnings	percentage_change
1	859327	3168	8536	-62.88659793814433
2	887387	4063	3893	4.366812227074235
3	896517	3151	7978	-60.503885685635495
4	721091	8143	4313	88.80129840018549
5	633636	4851	5388	-9.966592427616927
6	748190	4149	3675	12.897959183673468
7	936158	3487	9615	-63.733749349974
8	537591	7148	7386	-3.22231248307609
9	909018	8590	7053	21.792145186445484
10	878666	7609	3560	113.73595505617978
11	995710	8390	9720	-13.68312757201646
12	489275	5217	9693	-46.177653977096874
13	812053	6481	4442	45.90274651058082
14	728053	3014	3088	-2.3963730569948183
15	235295	9518	3924	142.5586136595311

In conclusion, S3 is not designed for sophisticated queries because its primary focus is on simple and scalable object storage. You can use tools like Apache Spark or Apache Hive to query data in S3, or you can use AWS Athena or AWS Glue to implement data lake architectures.

