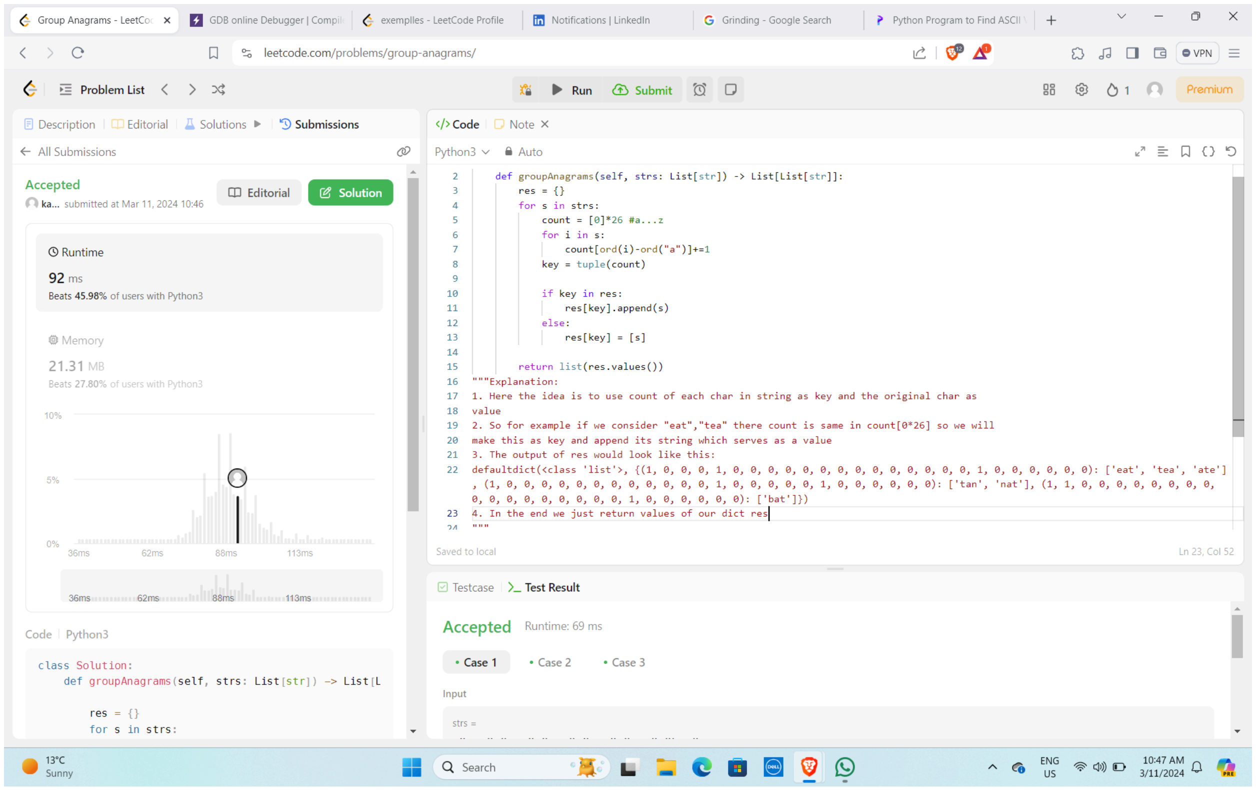
Day35 - March 11th 2024

1. Started my day as usual
2. Packed food and headed to library
3. Started marketing my profile for data engineer
4. Solved one medium leetcode problem


leetcode.com/problems/group-anagrams/

Accepted
ka... submitted at Mar 11, 2024 10:46

Runtime
92 ms
Beats 45.98% of users with Python3

Memory
21.31 MB
Beats 27.80% of users with Python3

Code | Python3

```python
class Solution:
    def groupAnagrams(self, strs: List[str]) -> List[L

        res = {}
        for s in strs:
```

```python
    def groupAnagrams(self, strs: List[str]) -> List[List[str]]:
        res = {}
        for s in strs:
            count = [0]*26 #a...z
            for i in s:
                count[ord(i)-ord("a")]+=1
            key = tuple(count)

            if key in res:
                res[key].append(s)
            else:
                res[key] = [s]

        return list(res.values())
"""Explanation:
1. Here the idea is to use count of each char in string as key and the original char as
value
2. So for example if we consider "eat","tea" there count is same in count[0*26] so we will
make this as key and append its string which serves as a value
3. The output of res would look like this:
defaultdict(<class 'list'>, {(1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0): ['eat', 'tea', 'ate']
, (1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0): ['tan', 'nat'], (1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0): ['bat']})
4. In the end we just return values of our dict res
"""
```

Saved to local   Ln 23, Col 52

Test Result

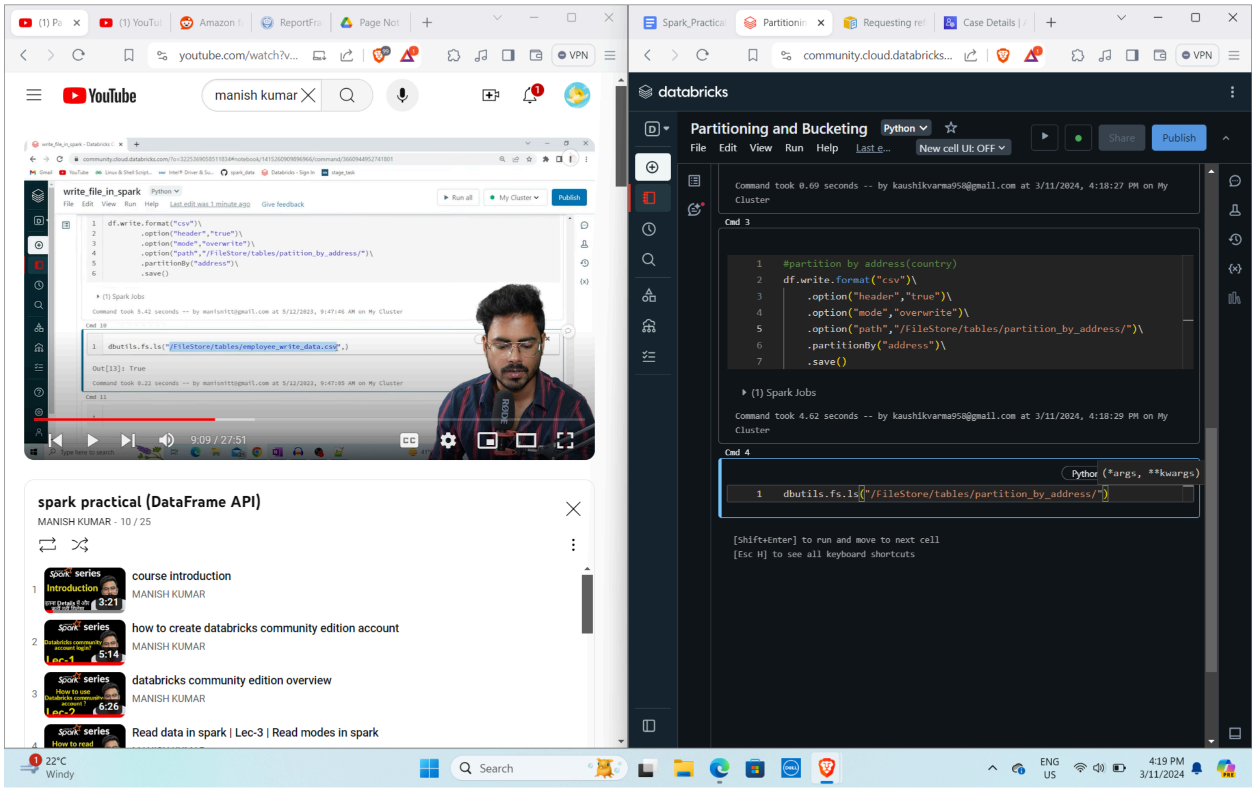
Accepted Runtime: 69 ms

Case 1 · Case 2 · Case 3

Input

strs =

5. Started learning spark practically



spark practical (DataFrame API)
MANISH KUMAR - 10 / 25

1 course introduction
MANISH KUMAR
2 how to create databricks community edition account
MANISH KUMAR
3 databricks community edition overview
MANISH KUMAR
4 Read data in spark | Lec-3 | Read modes in spark

write_file_in_spark

```
df.write.format("csv")\
        .option("header","true")\
        .option("mode","overwrite")\
        .option("path","/FileStore/tables/patition_by_address/")\
        .partitionBy("address")\
        .save()
```

(1) Spark Jobs
Command took 5.42 seconds -- by manisnitt@gmail.com at 5/12/2023, 9:47:46 AM on My Cluster

Cmd 10

```
dbutils.fs.ls("/FileStore/tables/employee_write_data.csv",)
```

```
Out[13]: True
```

Command took 0.22 seconds -- by manisnitt@gmail.com at 5/12/2023, 9:47:05 AM on My Cluster

Partitioning and Bucketing

Command took 0.69 seconds -- by kaushikvarma958@gmail.com at 3/11/2024, 4:18:27 PM on My Cluster

Cmd 3

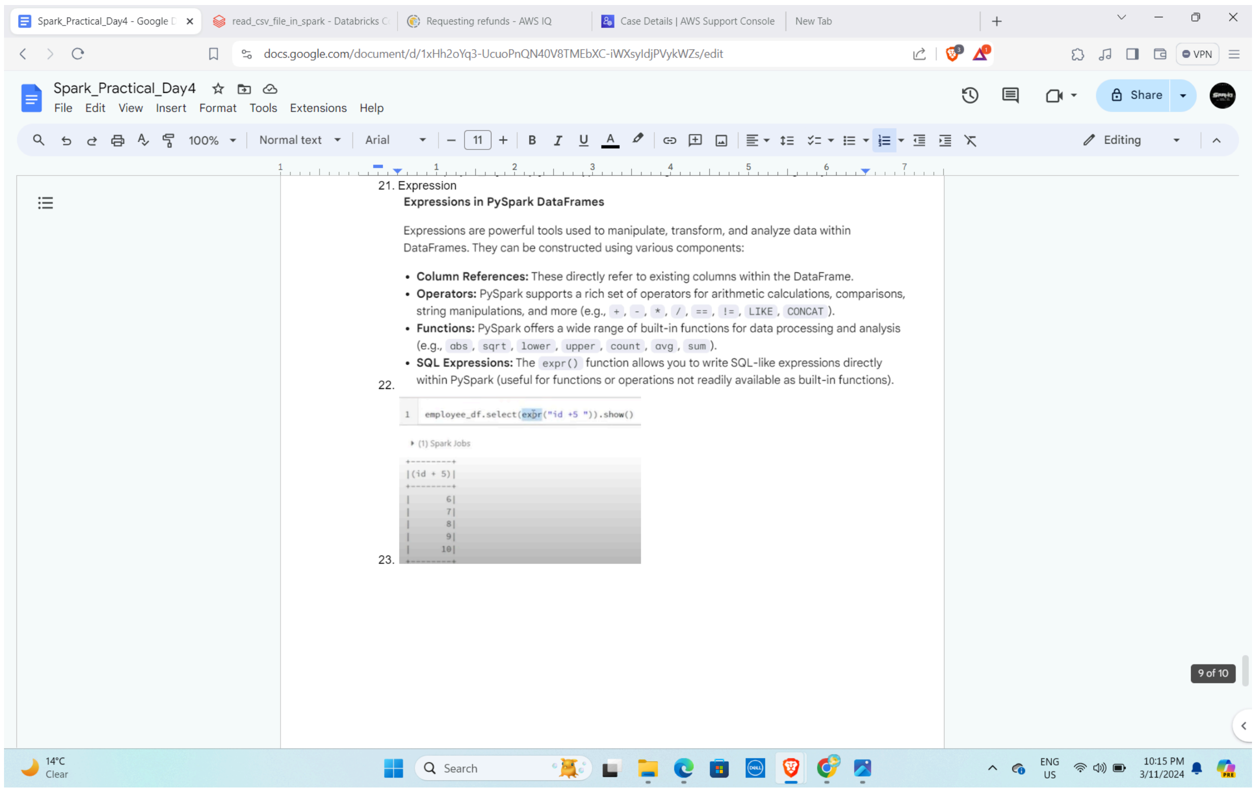
```
#partition by address(country)
df.write.format("csv")\
    .option("header","true")\
    .option("mode","overwrite")\
    .option("path","/FileStore/tables/partition_by_address/")\
    .partitionBy("address")\
    .save()
```

(1) Spark Jobs
Command took 4.62 seconds -- by kaushikvarma958@gmail.com at 3/11/2024, 4:18:29 PM on My Cluster

Cmd 4

```
dbutils.fs.ls("/FileStore/tables/partition_by_address/")
```

[Shift+Enter] to run and move to next cell
[Esc H] to see all keyboard shortcuts



Spark_Practical_Day4

21. Expression
Expressions in PySpark DataFrames

Expressions are powerful tools used to manipulate, transform, and analyze data within DataFrames. They can be constructed using various components:

- **Column References:** These directly refer to existing columns within the DataFrame.
- **Operators:** PySpark supports a rich set of operators for arithmetic calculations, comparisons, string manipulations, and more (e.g., `+`, `-`, `*`, `/`, `==`, `!=`, `LIKE`, `CONCAT`).
- **Functions:** PySpark offers a wide range of built-in functions for data processing and analysis (e.g., `abs`, `sqrt`, `lower`, `upper`, `count`, `avg`, `sum`).
- **SQL Expressions:** The `expr()` function allows you to write SQL-like expressions directly within PySpark (useful for functions or operations not readily available as built-in functions).

22.

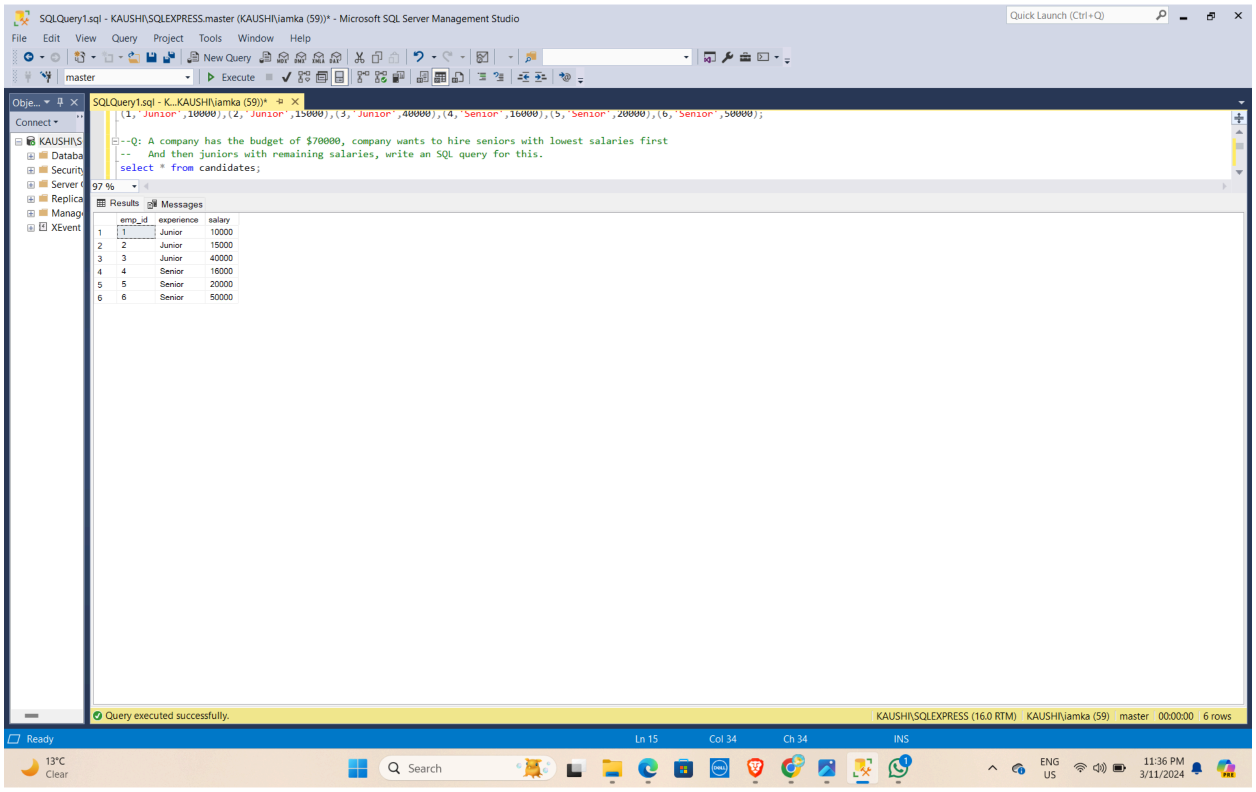
```
employee_df.select(expr("id +5 ")).show()
```

(1) Spark Jobs

```
+--------+
|(id + 5)|
+--------+
|       6|
|       7|
|       8|
|       9|
|      10|
+--------+
```

23.

6. Ended my day by solving complex SQL questions from Ankit'YT

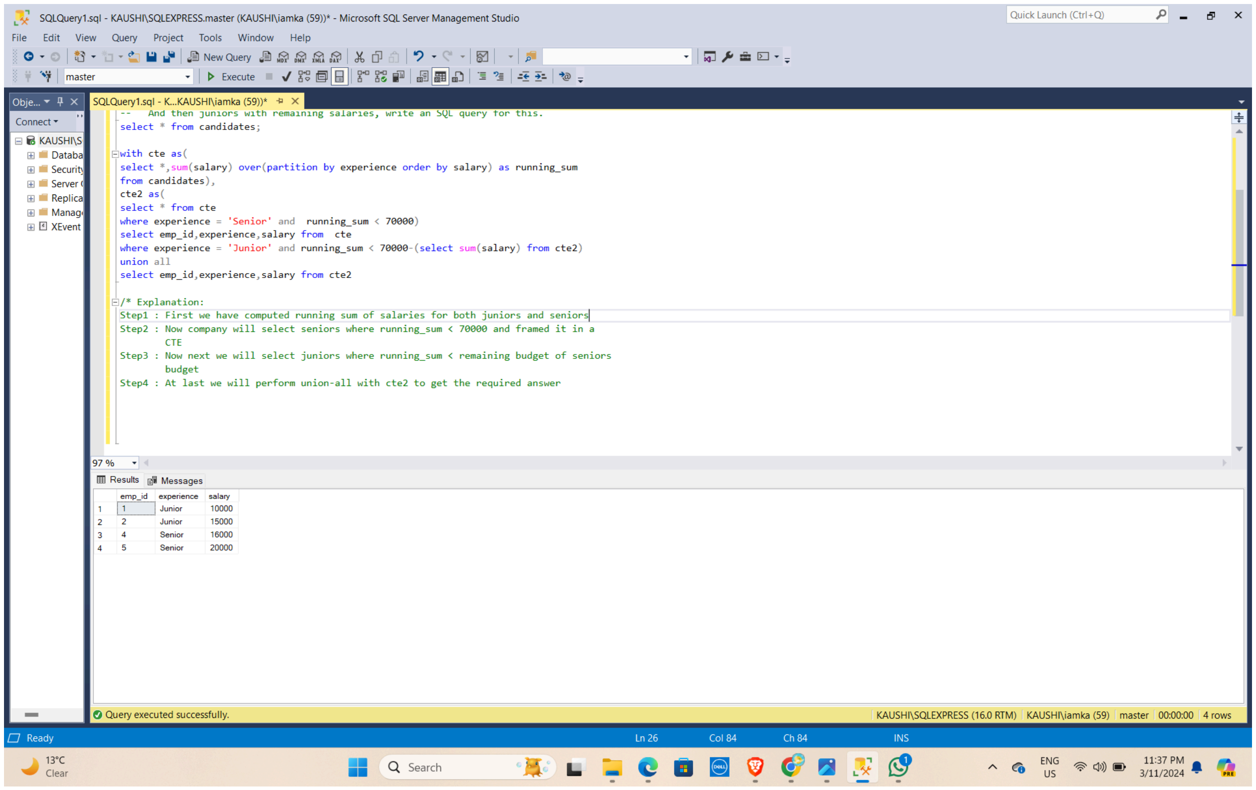


```
(1,'Junior',10000),(2,'Junior',15000),(3,'Junior',40000),(4,'Senior',16000),(5,'Senior',20000),(6,'Senior',50000);

--Q: A company has the budget of $70000, company wants to hire seniors with lowest salaries first
--   And then juniors with remaining salaries, write an SQL query for this.
select * from candidates;
```

```
   emp_id  experience  salary
1  1       Junior      10000
2  2       Junior      15000
3  3       Junior      40000
4  4       Senior      16000
5  5       Senior      20000
6  6       Senior      50000
```

Query executed successfully. KAUSHI\SQLEXPRESS (16.0 RTM) | KAUSHI\iamka (59) | master | 00:00:00 | 6 rows



```
--   And then juniors with remaining salaries, write an SQL query for this.
select * from candidates;

with cte as(
select *,sum(salary) over(partition by experience order by salary) as running_sum
from candidates),
cte2 as(
select * from cte
where experience = 'Senior' and  running_sum < 70000)
select emp_id,experience,salary from  cte
where experience = 'Junior' and running_sum < 70000-(select sum(salary) from cte2)
union all
select emp_id,experience,salary from cte2

/* Explanation:
Step1 : First we have computed running sum of salaries for both juniors and seniors
Step2 : Now company will select seniors where running_sum < 70000 and framed it in a
        CTE
Step3 : Now next we will select juniors where running_sum < remaining budget of seniors
        budget
Step4 : At last we will perform union-all with cte2 to get the required answer
```

```
   emp_id  experience  salary
1  1       Junior      10000
2  2       Junior      15000
3  4       Senior      16000
4  5       Senior      20000
```

Query executed successfully. KAUSHI\SQLEXPRESS (16.0 RTM) | KAUSHI\iamka (59) | master | 00:00:00 | 4 rows