1. **DATA FILTERING**
   1. **595. Big Countries**

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| name | varchar |

| continent | varchar |

| area | int |

| population | int |

| gdp | bigint |

+-------------+---------+

name is the primary key (column with unique values) for this table.

Each row of this table gives information about the name of a country, the continent to which it belongs, its area, the population, and its GDP value.

A country is **big** if:

* it has an area of at least three million (i.e., 3000000 km2), or
* it has a population of at least twenty-five million (i.e., 25000000).

Write a solution to find the name, population, and area of the **big countries**.

Return the result table in **any order**.

The result format is in the following example.

**Input:**

World table:

+-------------+-----------+---------+------------+--------------+

| name | continent | area | population | gdp |

+-------------+-----------+---------+------------+--------------+

| Afghanistan | Asia | 652230 | 25500100 | 20343000000 |

| Albania | Europe | 28748 | 2831741 | 12960000000 |

| Algeria | Africa | 2381741 | 37100000 | 188681000000 |

| Andorra | Europe | 468 | 78115 | 3712000000 |

| Angola | Africa | 1246700 | 20609294 | 100990000000 |

+-------------+-----------+---------+------------+--------------+

**Output:**

+-------------+------------+---------+

| name | population | area |

+-------------+------------+---------+

| Afghanistan | 25500100 | 652230 |

| Algeria | 37100000 | 2381741 |

+-------------+------------+---------+

import pandas as pd

def big\_countries(world: pd.DataFrame) -> pd.DataFrame:

    new\_df = world[(world['area'] >= 3000000) | (world['population'] >= 25000000)]

    return new\_df[['name','population','area']]

* 1. **1757. Recyclable and Low Fat Products**

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| product\_id | int |

| low\_fats | enum |

| recyclable | enum |

+-------------+---------+

product\_id is the primary key (column with unique values) for this table.

low\_fats is an ENUM (category) of type ('Y', 'N') where 'Y' means this product is low fat and 'N' means it is not.

recyclable is an ENUM (category) of types ('Y', 'N') where 'Y' means this product is recyclable and 'N' means it is not.

Write a solution to find the ids of products that are both low fat and recyclable.

Return the result table in **any order**.

The result format is in the following example.

**Example 1:**

**Input:**

Products table:

+-------------+----------+------------+

| product\_id | low\_fats | recyclable |

+-------------+----------+------------+

| 0 | Y | N |

| 1 | Y | Y |

| 2 | N | Y |

| 3 | Y | Y |

| 4 | N | N |

+-------------+----------+------------+

**Output:**

+-------------+

| product\_id |

+-------------+

| 1 |

| 3 |

+-------------+

**Explanation:** Only products 1 and 3 are both low fat and recyclable.

import pandas as pd

def find\_products(products: pd.DataFrame) -> pd.DataFrame:

    new\_df = products[(products['low\_fats'] == 'Y') & (products['recyclable'] == 'Y')]

    return new\_df[['product\_id']]

* 1. **183. Customers Who Never Order**

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| id | int |

| name | varchar |

+-------------+---------+

id is the primary key (column with unique values) for this table.

Each row of this table indicates the ID and name of a customer.

Table: Orders

+-------------+------+

| Column Name | Type |

+-------------+------+

| id | int |

| customerId | int |

+-------------+------+

id is the primary key (column with unique values) for this table.

customerId is a foreign key (reference columns) of the ID from the Customers table.

Each row of this table indicates the ID of an order and the ID of the customer who ordered it.

Write a solution to find all customers who never order anything.

Return the result table in **any order**.

The result format is in the following example.

**Example 1:**

**Input:**

Customers table:

+----+-------+

| id | name |

+----+-------+

| 1 | Joe |

| 2 | Henry |

| 3 | Sam |

| 4 | Max |

+----+-------+

Orders table:

+----+------------+

| id | customerId |

+----+------------+

| 1 | 3 |

| 2 | 1 |

+----+------------+

**Output:**

+-----------+

| Customers |

+-----------+

| Henry |

| Max |

+-----------+

import pandas as pd

def find\_customers(customers: pd.DataFrame, orders: pd.DataFrame) -> pd.DataFrame:

    new\_df = customers[~customers['id'].isin(orders['customerId'])]

    return new\_df[['name']].rename(columns={'name':'Customers'})

* 1. **1148. Article Views I**

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| article\_id | int |

| author\_id | int |

| viewer\_id | int |

| view\_date | date |

+---------------+---------+

There is no primary key (column with unique values) for this table, the table may have duplicate rows.

Each row of this table indicates that some viewer viewed an article (written by some author) on some date.

Note that equal author\_id and viewer\_id indicate the same person.

Write a solution to find all the authors that viewed at least one of their own articles.

Return the result table sorted by id in ascending order.

The result format is in the following example.

**Example 1:**

**Input:**

Views table:

+------------+-----------+-----------+------------+

| article\_id | author\_id | viewer\_id | view\_date |

+------------+-----------+-----------+------------+

| 1 | 3 | 5 | 2019-08-01 |

| 1 | 3 | 6 | 2019-08-02 |

| 2 | 7 | 7 | 2019-08-01 |

| 2 | 7 | 6 | 2019-08-02 |

| 4 | 7 | 1 | 2019-07-22 |

| 3 | 4 | 4 | 2019-07-21 |

| 3 | 4 | 4 | 2019-07-21 |

+------------+-----------+-----------+------------+

**Output:**

+------+

| id |

+------+

| 4 |

| 7 |

+------+

import pandas as pd

def article\_views(views: pd.DataFrame) -> pd.DataFrame:

    new\_df = views[(views['author\_id']==views['viewer\_id'])]

    new\_df.drop\_duplicates(subset = 'author\_id', keep = 'first', inplace = True)

    new\_df.sort\_values(by='author\_id', ascending = True, inplace = True)

    return new\_df[['author\_id']].rename(columns={'author\_id':'id'})

1. **STRING METHODS**
   1. **1683. Invalid Tweets**

+----------------+---------+

| Column Name | Type |

+----------------+---------+

| tweet\_id | int |

| content | varchar |

+----------------+---------+

tweet\_id is the primary key (column with unique values) for this table.

This table contains all the tweets in a social media app.

Write a solution to find the IDs of the invalid tweets. The tweet is invalid if the number of characters used in the content of the tweet is **strictly greater** than 15.

Return the result table in **any order**.

The result format is in the following example.

**Example 1:**

**Input:**

Tweets table:

+----------+----------------------------------+

| tweet\_id | content |

+----------+----------------------------------+

| 1 | Vote for Biden |

| 2 | Let us make America great again! |

+----------+----------------------------------+

**Output:**

+----------+

| tweet\_id |

+----------+

| 2 |

+----------+

**Explanation:**

Tweet 1 has length = 14. It is a valid tweet.

Tweet 2 has length = 32. It is an invalid tweet.

import pandas as pd

def invalid\_tweets(tweets: pd.DataFrame) -> pd.DataFrame:

    return tweets[tweets['content'].str.len() > 15][['tweet\_id']]

* 1. **1873. Calculate Special Bonus**

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| employee\_id | int |

| name | varchar |

| salary | int |

+-------------+---------+

employee\_id is the primary key (column with unique values) for this table.

Each row of this table indicates the employee ID, employee name, and salary.

Write a solution to calculate the bonus of each employee. The bonus of an employee is 100% of their salary if the ID of the employee is **an odd number** and **the employee's name does not start with the character**'M'. The bonus of an employee is 0 otherwise.

Return the result table ordered by employee\_id.

The result format is in the following example.

**Example 1:**

**Input:**

Employees table:

+-------------+---------+--------+

| employee\_id | name | salary |

+-------------+---------+--------+

| 2 | Meir | 3000 |

| 3 | Michael | 3800 |

| 7 | Addilyn | 7400 |

| 8 | Juan | 6100 |

| 9 | Kannon | 7700 |

+-------------+---------+--------+

**Output:**

+-------------+-------+

| employee\_id | bonus |

+-------------+-------+

| 2 | 0 |

| 3 | 0 |

| 7 | 7400 |

| 8 | 0 |

| 9 | 7700 |

+-------------+-------+

**Explanation:**

The employees with IDs 2 and 8 get 0 bonus because they have an even employee\_id.

The employee with ID 3 gets 0 bonus because their name starts with 'M'.

The rest of the employees get a 100% bonus.

import pandas as pd

def calculate\_special\_bonus(employees: pd.DataFrame) -> pd.DataFrame:

    employees['bonus'] = employees.apply(lambda row: row['salary'] if row['employee\_id'] % 2 ==1 and row['name'][:1] != 'M' else 0, axis = 1)

    return employees[['employee\_id', 'bonus']].sort\_values(by='employee\_id', ascending=True)

* 1. **1667. Fix Names in a Table**

+----------------+---------+

| Column Name | Type |

+----------------+---------+

| user\_id | int |

| name | varchar |

+----------------+---------+

user\_id is the primary key (column with unique values) for this table.

This table contains the ID and the name of the user. The name consists of only lowercase and uppercase characters.

Write a solution to fix the names so that only the first character is uppercase and the rest are lowercase.

Return the result table ordered by user\_id.

The result format is in the following example.

**Example 1:**

**Input:**

Users table:

+---------+-------+

| user\_id | name |

+---------+-------+

| 1 | aLice |

| 2 | bOB |

+---------+-------+

**Output:**

+---------+-------+

| user\_id | name |

+---------+-------+

| 1 | Alice |

| 2 | Bob |

+---------+-------+

import pandas as pd

def fix\_names(users: pd.DataFrame) -> pd.DataFrame:

    users['name'] = users['name'].str.capitalize()

    return users.sort\_values(by = 'user\_id', ascending = True)

* 1. **1517. Find Users With Valid E-Mails**

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| user\_id | int |

| name | varchar |

| mail | varchar |

+---------------+---------+

user\_id is the primary key (column with unique values) for this table.

This table contains information of the users signed up in a website. Some e-mails are invalid.

Write a solution to find the users who have **valid emails**.

A valid e-mail has a prefix name and a domain where:

* **The prefix name** is a string that may contain letters (upper or lower case), digits, underscore '\_', period '.', and/or dash '-'. The prefix name **must** start with a letter.
* **The domain** is '@leetcode.com'.

Return the result table in **any order**.

The result format is in the following example.

**Example 1:**

**Input:**

Users table:

+---------+-----------+-------------------------+

| user\_id | name | mail |

+---------+-----------+-------------------------+

| 1 | Winston | winston@leetcode.com |

| 2 | Jonathan | jonathanisgreat |

| 3 | Annabelle | bella-@leetcode.com |

| 4 | Sally | sally.come@leetcode.com |

| 5 | Marwan | quarz#2020@leetcode.com |

| 6 | David | david69@gmail.com |

| 7 | Shapiro | .shapo@leetcode.com |

+---------+-----------+-------------------------+

**Output:**

+---------+-----------+-------------------------+

| user\_id | name | mail |

+---------+-----------+-------------------------+

| 1 | Winston | winston@leetcode.com |

| 3 | Annabelle | bella-@leetcode.com |

| 4 | Sally | sally.come@leetcode.com |

+---------+-----------+-------------------------+

**Explanation:**

The mail of user 2 does not have a domain.

The mail of user 5 has the # sign which is not allowed.

The mail of user 6 does not have the leetcode domain.

The mail of user 7 starts with a period.

import pandas as pd

def valid\_emails(users: pd.DataFrame) -> pd.DataFrame:

    return users[users['mail'].str.match(r'^[A-Za-z][A-Za-z0-9-.\_][\*@leetcode\.com$](mailto:*@leetcode\.com$)')]

* 1. **1527. Patients With a Condition**

+--------------+---------+

| Column Name | Type |

+--------------+---------+

| patient\_id | int |

| patient\_name | varchar |

| conditions | varchar |

+--------------+---------+

patient\_id is the primary key (column with unique values) for this table.

'conditions' contains 0 or more code separated by spaces.

This table contains information of the patients in the hospital.

Write a solution to find the patient\_id, patient\_name, and conditions of the patients who have Type I Diabetes. Type I Diabetes always starts with DIAB1 prefix.

Return the result table in **any order**.

The result format is in the following example.

**Example 1:**

**Input:**

Patients table:

+------------+--------------+--------------+

| patient\_id | patient\_name | conditions |

+------------+--------------+--------------+

| 1 | Daniel | YFEV COUGH |

| 2 | Alice | |

| 3 | Bob | DIAB100 MYOP |

| 4 | George | ACNE DIAB100 |

| 5 | Alain | DIAB201 |

+------------+--------------+--------------+

**Output:**

+------------+--------------+--------------+

| patient\_id | patient\_name | conditions |

+------------+--------------+--------------+

| 3 | Bob | DIAB100 MYOP |

| 4 | George | ACNE DIAB100 |

+------------+--------------+--------------+

**Explanation:** Bob and George both have a condition that starts with DIAB1.

import pandas as pd

def find\_patients(patients: pd.DataFrame) -> pd.DataFrame:

    return patients[patients['conditions'].str.contains(r'(^DIAB1)|( DIAB1)')]

1. **DATA MANIPULATION**
   1. **177. Nth Highest Salary**

+-------------+------+

| Column Name | Type |

+-------------+------+

| id | int |

| salary | int |

+-------------+------+

id is the primary key (column with unique values) for this table.

Each row of this table contains information about the salary of an employee.

Write a solution to find the nth highest salary from the Employee table. If there is no nth highest salary, return null.

The result format is in the following example.

**Example 1:**

**Input:**

Employee table:

+----+--------+

| id | salary |

+----+--------+

| 1 | 100 |

| 2 | 200 |

| 3 | 300 |

+----+--------+

n = 2

**Output:**

+------------------------+

| getNthHighestSalary(2) |

+------------------------+

| 200 |

+------------------------+

import pandas as pd

def nth\_highest\_salary(employee: pd.DataFrame, N: int) -> pd.DataFrame:

    unique\_values = employee.sort\_values(by='salary', ascending=False)['salary'].unique()

    unique\_df = pd.DataFrame({'salary': unique\_values})

    new\_column\_name = f"getNthHighestSalary({N})"

    if N <= len(unique\_df) :

        nth\_highest\_salary = unique\_df.iloc[N-1]['salary']

        result\_df = pd.DataFrame({new\_column\_name: [nth\_highest\_salary]})

    else:

        result\_df = pd.DataFrame({new\_column\_name: [None]})

    return result\_df

* 1. **176. Second Highest Salary**

+-------------+------+

| Column Name | Type |

+-------------+------+

| id | int |

| salary | int |

+-------------+------+

id is the primary key (column with unique values) for this table.

Each row of this table contains information about the salary of an employee.

Write a solution to find the second highest salary from the Employee table. If there is no second highest salary, return null (return None in Pandas).

The result format is in the following example.

**Example 1:**

**Input:**

Employee table:

+----+--------+

| id | salary |

+----+--------+

| 1 | 100 |

| 2 | 200 |

| 3 | 300 |

+----+--------+

**Output:**

+---------------------+

| SecondHighestSalary |

+---------------------+

| 200 |

+---------------------+

**Example 2:**

**Input:**

Employee table:

+----+--------+

| id | salary |

+----+--------+

| 1 | 100 |

+----+--------+

**Output:**

+---------------------+

| SecondHighestSalary |

+---------------------+

| null |

+---------------------+

import pandas as pd

def second\_highest\_salary(employee: pd.DataFrame) -> pd.DataFrame:

    sorted\_df = employee.sort\_values(by='salary', ascending= False)['salary'].unique()

    unique\_df = pd.DataFrame({'salary':sorted\_df})

    if len(sorted\_df) >= 2:

        top = unique\_df.iloc[1]['salary']

        result\_df = pd.DataFrame({'SecondHighestSalary': [top]})

    else:

        result\_df = pd.DataFrame({'SecondHighestSalary': [None]})

    return result\_df

* 1. **184. Department Highest Salary**

+--------------+---------+

| Column Name | Type |

+--------------+---------+

| id | int |

| name | varchar |

| salary | int |

| departmentId | int |

+--------------+---------+

id is the primary key (column with unique values) for this table.

departmentId is a foreign key (reference columns) of the ID from the Department table.

Each row of this table indicates the ID, name, and salary of an employee. It also contains the ID of their department.

Table: Department

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| id | int |

| name | varchar |

+-------------+---------+

id is the primary key (column with unique values) for this table. It is guaranteed that department name is not NULL.

Each row of this table indicates the ID of a department and its name.

Write a solution to find employees who have the highest salary in each of the departments.

Return the result table in **any order**.

The result format is in the following example.

**Example 1:**

**Input:**

Employee table:

+----+-------+--------+--------------+

| id | name | salary | departmentId |

+----+-------+--------+--------------+

| 1 | Joe | 70000 | 1 |

| 2 | Jim | 90000 | 1 |

| 3 | Henry | 80000 | 2 |

| 4 | Sam | 60000 | 2 |

| 5 | Max | 90000 | 1 |

+----+-------+--------+--------------+

Department table:

+----+-------+

| id | name |

+----+-------+

| 1 | IT |

| 2 | Sales |

+----+-------+

**Output:**

+------------+----------+--------+

| Department | Employee | Salary |

+------------+----------+--------+

| IT | Jim | 90000 |

| Sales | Henry | 80000 |

| IT | Max | 90000 |

+------------+----------+--------+

**Explanation:** Max and Jim both have the highest salary in the IT department and Henry has the highest salary in the Sales department.

import pandas as pd

def department\_highest\_salary(employee: pd.DataFrame, department: pd.DataFrame) -> pd.DataFrame:

    merged\_df = pd.merge(employee, department, left\_on = 'departmentId', right\_on = 'id')

    grouped\_df = merged\_df.groupby('name\_y')['salary'].max().reset\_index()

    return pd.merge(grouped\_df, merged\_df, on = ['name\_y', 'salary'], how = 'inner')[['name\_y', 'name\_x', 'salary']].rename(columns={'name\_y':'Department', 'name\_x':'Employee','salary':'Salary'})

* 1. **178. Rank Scores**

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| id | int |

| score | decimal |

+-------------+---------+

id is the primary key (column with unique values) for this table.

Each row of this table contains the score of a game. Score is a floating point value with two decimal places.

Write a solution to find the rank of the scores. The ranking should be calculated according to the following rules:

* The scores should be ranked from the highest to the lowest.
* If there is a tie between two scores, both should have the same ranking.
* After a tie, the next ranking number should be the next consecutive integer value. In other words, there should be no holes between ranks.

Return the result table ordered by score in descending order.

The result format is in the following example.

**Example 1:**

**Input:**

Scores table:

+----+-------+

| id | score |

+----+-------+

| 1 | 3.50 |

| 2 | 3.65 |

| 3 | 4.00 |

| 4 | 3.85 |

| 5 | 4.00 |

| 6 | 3.65 |

+----+-------+

**Output:**

+-------+------+

| score | rank |

+-------+------+

| 4.00 | 1 |

| 4.00 | 1 |

| 3.85 | 2 |

| 3.65 | 3 |

| 3.65 | 3 |

| 3.50 | 4 |

+-------+------+

import pandas as pd

def order\_scores(scores: pd.DataFrame) -> pd.DataFrame:

    scores['rank'] = scores['score'].rank(method = 'dense', ascending = False).astype(int)

    scores.sort\_values(by='score', ascending = False, inplace= True)

    return scores[['score', 'rank']]

* 1. **196. Delete Duplicate Emails**

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| id | int |

| email | varchar |

+-------------+---------+

id is the primary key (column with unique values) for this table.

Each row of this table contains an email. The emails will not contain uppercase letters.

Write a solution to**delete** all duplicate emails, keeping only one unique email with the smallest id.

For SQL users, please note that you are supposed to write a DELETE statement and not a SELECT one.

For Pandas users, please note that you are supposed to modify Person in place.

After running your script, the answer shown is the Person table. The driver will first compile and run your piece of code and then show the Person table. The final order of the Person table **does not matter**.

The result format is in the following example.

**Example 1:**

**Input:**

Person table:

+----+------------------+

| id | email |

+----+------------------+

| 1 | john@example.com |

| 2 | bob@example.com |

| 3 | john@example.com |

+----+------------------+

**Output:**

+----+------------------+

| id | email |

+----+------------------+

| 1 | john@example.com |

| 2 | bob@example.com |

+----+------------------+

**Explanation:** john@example.com is repeated two times. We keep the row with the smallest Id = 1.

import pandas as pd

# Modify Person in place

def delete\_duplicate\_emails(person: pd.DataFrame) -> None:

    person.sort\_values(by = 'id', ascending = True, inplace = True)

    person.drop\_duplicates(subset = ['email'], inplace = True, keep = "first")

* 1. **1795. Rearrange Products Table**

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| product\_id | int |

| store1 | int |

| store2 | int |

| store3 | int |

+-------------+---------+

product\_id is the primary key (column with unique values) for this table.

Each row in this table indicates the product's price in 3 different stores: store1, store2, and store3.

If the product is not available in a store, the price will be null in that store's column.

Write a solution to rearrange the Products table so that each row has (product\_id, store, price). If a product is not available in a store, do **not** include a row with that product\_id and store combination in the result table.

Return the result table in **any order**.

The result format is in the following example.

**Example 1:**

**Input:**

Products table:

+------------+--------+--------+--------+

| product\_id | store1 | store2 | store3 |

+------------+--------+--------+--------+

| 0 | 95 | 100 | 105 |

| 1 | 70 | null | 80 |

+------------+--------+--------+--------+

**Output:**

+------------+--------+-------+

| product\_id | store | price |

+------------+--------+-------+

| 0 | store1 | 95 |

| 0 | store2 | 100 |

| 0 | store3 | 105 |

| 1 | store1 | 70 |

| 1 | store3 | 80 |

+------------+--------+-------+

**Explanation:**

Product 0 is available in all three stores with prices 95, 100, and 105 respectively.

Product 1 is available in store1 with price 70 and store3 with price 80. The product is not available in store2.

import pandas as pd

def rearrange\_products\_table(products: pd.DataFrame) -> pd.DataFrame:

    return pd.melt(products, id\_vars=['product\_id'], value\_vars=['store1', 'store2', 'store3'], var\_name = 'store', value\_name = 'price').dropna(subset = ['price'])

1. **STATISTICS**
   1. **2082. The Number of Rich Customers**

+-------------+------+

| Column Name | Type |

+-------------+------+

| bill\_id | int |

| customer\_id | int |

| amount | int |

+-------------+------+

bill\_id is the primary key (column with unique values) for this table.

Each row contains information about the amount of one bill and the customer associated with it.

Write a solution to report the number of customers who had **at least one** bill with an amount **strictly greater** than 500.

The result format is in the following example.

**Example 1:**

**Input:**

Store table:

+---------+-------------+--------+

| bill\_id | customer\_id | amount |

+---------+-------------+--------+

| 6 | 1 | 549 |

| 8 | 1 | 834 |

| 4 | 2 | 394 |

| 11 | 3 | 657 |

| 13 | 3 | 257 |

+---------+-------------+--------+

**Output:**

+------------+

| rich\_count |

+------------+

| 2 |

+------------+

**Explanation:**

Customer 1 has two bills with amounts strictly greater than 500.

Customer 2 does not have any bills with an amount strictly greater than 500.

Customer 3 has one bill with an amount strictly greater than 500.

import pandas as pd

def count\_rich\_customers(store: pd.DataFrame) -> pd.DataFrame:

    rich\_df = store[store['amount'] > 500]

    return pd.DataFrame(data ={'rich\_count':[rich\_df['customer\_id'].nunique()]})

* 1. **1173. Immediate Food Delivery I**

+-----------------------------+---------+

| Column Name | Type |

+-----------------------------+---------+

| delivery\_id | int |

| customer\_id | int |

| order\_date | date |

| customer\_pref\_delivery\_date | date |

+-----------------------------+---------+

delivery\_id is the primary key (column with unique values) of this table.

The table holds information about food delivery to customers that make orders at some date and specify a preferred delivery date (on the same order date or after it).

If the customer's preferred delivery date is the same as the order date, then the order is called **immediate;** otherwise, it is called **scheduled**.

Write a solution to find the percentage of immediate orders in the table, **rounded to 2 decimal places**.

The result format is in the following example.

**Example 1:**

**Input:**

Delivery table:

+-------------+-------------+------------+-----------------------------+

| delivery\_id | customer\_id | order\_date | customer\_pref\_delivery\_date |

+-------------+-------------+------------+-----------------------------+

| 1 | 1 | 2019-08-01 | 2019-08-02 |

| 2 | 5 | 2019-08-02 | 2019-08-02 |

| 3 | 1 | 2019-08-11 | 2019-08-11 |

| 4 | 3 | 2019-08-24 | 2019-08-26 |

| 5 | 4 | 2019-08-21 | 2019-08-22 |

| 6 | 2 | 2019-08-11 | 2019-08-13 |

+-------------+-------------+------------+-----------------------------+

**Output:**

+----------------------+

| immediate\_percentage |

+----------------------+

| 33.33 |

+----------------------+

**Explanation:** The orders with delivery id 2 and 3 are immediate while the others are scheduled.

import pandas as pd

def food\_delivery(delivery: pd.DataFrame) -> pd.DataFrame:

    count = (delivery['order\_date'] == delivery['customer\_pref\_delivery\_date']).sum()

    return pd.DataFrame(data = {'immediate\_percentage': [round(((count\*100)/len(delivery)), 2)]})

* 1. **1907. Count Salary Categories**

+-------------+------+

| Column Name | Type |

+-------------+------+

| account\_id | int |

| income | int |

+-------------+------+

account\_id is the primary key (column with unique values) for this table.

Each row contains information about the monthly income for one bank account.

Write a solution to calculate the number of bank accounts for each salary category. The salary categories are:

* "Low Salary": All the salaries **strictly less** than $20000.
* "Average Salary": All the salaries in the **inclusive** range [$20000, $50000].
* "High Salary": All the salaries **strictly greater** than $50000.

The result table **must** contain all three categories. If there are no accounts in a category, return 0.

Return the result table in **any order**.

The result format is in the following example.

**Example 1:**

**Input:**

Accounts table:

+------------+--------+

| account\_id | income |

+------------+--------+

| 3 | 108939 |

| 2 | 12747 |

| 8 | 87709 |

| 6 | 91796 |

+------------+--------+

**Output:**

+----------------+----------------+

| category | accounts\_count |

+----------------+----------------+

| Low Salary | 1 |

| Average Salary | 0 |

| High Salary | 3 |

+----------------+----------------+

**Explanation:**

Low Salary: Account 2.

Average Salary: No accounts.

High Salary: Accounts 3, 6, and 8.

import pandas as pd

def count\_salary\_categories(accounts: pd.DataFrame) -> pd.DataFrame:

    dict\_salary = {'category': ['Low Salary', 'Average Salary', 'High Salary']}

    low\_salary\_count = len(accounts[accounts['income'] < 20000])

    average\_salary\_count = len(accounts[(accounts['income'] >= 20000) & (accounts['income'] <= 50000)])

    high\_salary\_count = len(accounts[accounts['income'] > 50000])

    dict\_salary['accounts\_count'] = [low\_salary\_count, average\_salary\_count, high\_salary\_count]

    return pd.DataFrame(dict\_salary)

1. **Data Aggregation**

**5.1 1741. Find Total Time Spent by Each Employee**

+-------------+------+

| Column Name | Type |

+-------------+------+

| emp\_id | int |

| event\_day | date |

| in\_time | int |

| out\_time | int |

+-------------+------+

(emp\_id, event\_day, in\_time) is the primary key (combinations of columns with unique values) of this table.

The table shows the employees' entries and exits in an office.

event\_day is the day at which this event happened, in\_time is the minute at which the employee entered the office, and out\_time is the minute at which they left the office.

in\_time and out\_time are between 1 and 1440.

It is guaranteed that no two events on the same day intersect in time, and in\_time < out\_time.

Write a solution to calculate the total time **in minutes** spent by each employee on each day at the office. Note that within one day, an employee can enter and leave more than once. The time spent in the office for a single entry is out\_time - in\_time.

Return the result table in **any order**.

The result format is in the following example.

**Example 1:**

**Input:**

Employees table:

+--------+------------+---------+----------+

| emp\_id | event\_day | in\_time | out\_time |

+--------+------------+---------+----------+

| 1 | 2020-11-28 | 4 | 32 |

| 1 | 2020-11-28 | 55 | 200 |

| 1 | 2020-12-03 | 1 | 42 |

| 2 | 2020-11-28 | 3 | 33 |

| 2 | 2020-12-09 | 47 | 74 |

+--------+------------+---------+----------+

**Output:**

+------------+--------+------------+

| day | emp\_id | total\_time |

+------------+--------+------------+

| 2020-11-28 | 1 | 173 |

| 2020-11-28 | 2 | 30 |

| 2020-12-03 | 1 | 41 |

| 2020-12-09 | 2 | 27 |

+------------+--------+------------+

**Explanation:**

Employee 1 has three events: two on day 2020-11-28 with a total of (32 - 4) + (200 - 55) = 173, and one on day 2020-12-03 with a total of (42 - 1) = 41.

Employee 2 has two events: one on day 2020-11-28 with a total of (33 - 3) = 30, and one on day 2020-12-09 with a total of (74 - 47) = 27.

import pandas as pd

def total\_time(employees: pd.DataFrame) -> pd.DataFrame:

    employees['event\_day'] = pd.to\_datetime(employees['event\_day'])

    employees['total\_time'] = employees['out\_time'] - employees['in\_time']

    result\_df = employees.groupby(['emp\_id', 'event\_day'])['total\_time'].sum().reset\_index()

    return result\_df[['event\_day', 'emp\_id', 'total\_time']].rename(columns={'event\_day':'day'})

**5.2. 511. Game Play Analysis I**

+--------------+---------+

| Column Name | Type |

+--------------+---------+

| player\_id | int |

| device\_id | int |

| event\_date | date |

| games\_played | int |

+--------------+---------+

(player\_id, event\_date) is the primary key (combination of columns with unique values) of this table.

This table shows the activity of players of some games.

Each row is a record of a player who logged in and played a number of games (possibly 0) before logging out on someday using some device.

Write a solution to find the **first login date** for each player.

Return the result table in **any order**.

The result format is in the following example.

**Example 1:**

**Input:**

Activity table:

+-----------+-----------+------------+--------------+

| player\_id | device\_id | event\_date | games\_played |

+-----------+-----------+------------+--------------+

| 1 | 2 | 2016-03-01 | 5 |

| 1 | 2 | 2016-05-02 | 6 |

| 2 | 3 | 2017-06-25 | 1 |

| 3 | 1 | 2016-03-02 | 0 |

| 3 | 4 | 2018-07-03 | 5 |

+-----------+-----------+------------+--------------+

**Output:**

+-----------+-------------+

| player\_id | first\_login |

+-----------+-------------+

| 1 | 2016-03-01 |

| 2 | 2017-06-25 |

| 3 | 2016-03-02 |

+-----------+-------------+

import pandas as pd

def game\_analysis(activity: pd.DataFrame) -> pd.DataFrame:

    activity['event\_date'] = pd.to\_datetime(activity['event\_date'])

    grouped\_df = activity.groupby('player\_id')['event\_date'].min().reset\_index()

    return grouped\_df[['player\_id', 'event\_date']].rename(columns = {'event\_date':'first\_login'})

**5.3. 2356. Number of Unique Subjects Taught by Each Teacher**

+-------------+------+

| Column Name | Type |

+-------------+------+

| teacher\_id | int |

| subject\_id | int |

| dept\_id | int |

+-------------+------+

(subject\_id, dept\_id) is the primary key (combinations of columns with unique values) of this table.

Each row in this table indicates that the teacher with teacher\_id teaches the subject subject\_id in the department dept\_id.

Write a solution to calculate the number of unique subjects each teacher teaches in the university.

Return the result table in **any order**.

The result format is shown in the following example.

**Example 1:**

**Input:**

Teacher table:

+------------+------------+---------+

| teacher\_id | subject\_id | dept\_id |

+------------+------------+---------+

| 1 | 2 | 3 |

| 1 | 2 | 4 |

| 1 | 3 | 3 |

| 2 | 1 | 1 |

| 2 | 2 | 1 |

| 2 | 3 | 1 |

| 2 | 4 | 1 |

+------------+------------+---------+

**Output:**

+------------+-----+

| teacher\_id | cnt |

+------------+-----+

| 1 | 2 |

| 2 | 4 |

+------------+-----+

**Explanation:**

Teacher 1:

- They teach subject 2 in departments 3 and 4.

- They teach subject 3 in department 3.

Teacher 2:

- They teach subject 1 in department 1.

- They teach subject 2 in department 1.

- They teach subject 3 in department 1.

- They teach subject 4 in department 1.

import pandas as pd

def count\_unique\_subjects(teacher: pd.DataFrame) -> pd.DataFrame:

    new\_df = pd.DataFrame(teacher.groupby('teacher\_id')['subject\_id'].nunique()).reset\_index()

    return (new\_df.rename(columns = {'subject\_id':'cnt'}))

**5.4. 596. Classes More Than 5 Students**

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| student | varchar |

| class | varchar |

+-------------+---------+

(student, class) is the primary key (combination of columns with unique values) for this table.

Each row of this table indicates the name of a student and the class in which they are enrolled.

Write a solution to find all the classes that have **at least five students**.

Return the result table in **any order**.

The result format is in the following example.

**Example 1:**

**Input:**

Courses table:

+---------+----------+

| student | class |

+---------+----------+

| A | Math |

| B | English |

| C | Math |

| D | Biology |

| E | Math |

| F | Computer |

| G | Math |

| H | Math |

| I | Math |

+---------+----------+

**Output:**

+---------+

| class |

+---------+

| Math |

+---------+

**Explanation:**

- Math has 6 students, so we include it.

- English has 1 student, so we do not include it.

- Biology has 1 student, so we do not include it.

- Computer has 1 student, so we do not include it.

import pandas as pd

def find\_classes(courses: pd.DataFrame) -> pd.DataFrame:

    new\_df = courses.groupby('class')['student'].nunique().reset\_index()

    filtered\_df = new\_df[new\_df['student'] >= 5]

    return filtered\_df[['class']]

**5.5. 586. Customer Placing the Largest Number of Orders**

+-----------------+----------+

| Column Name | Type |

+-----------------+----------+

| order\_number | int |

| customer\_number | int |

+-----------------+----------+

order\_number is the primary key (column with unique values) for this table.

This table contains information about the order ID and the customer ID.

Write a solution to find the customer\_number for the customer who has placed **the largest number of orders**.

The test cases are generated so that **exactly one customer** will have placed more orders than any other customer.

The result format is in the following example.

**Example 1:**

**Input:**

Orders table:

+--------------+-----------------+

| order\_number | customer\_number |

+--------------+-----------------+

| 1 | 1 |

| 2 | 2 |

| 3 | 3 |

| 4 | 3 |

+--------------+-----------------+

**Output:**

+-----------------+

| customer\_number |

+-----------------+

| 3 |

+-----------------+

**Explanation:**

The customer with number 3 has two orders, which is greater than either customer 1 or 2 because each of them only has one order.

So the result is customer\_number 3.

**Follow up:** What if more than one customer has the largest number of orders, can you find all the customer\_number in this case? -🡪 the same code will work!!!

import pandas as pd

def largest\_orders(orders: pd.DataFrame) -> pd.DataFrame:

    new\_df = orders.groupby('customer\_number')['order\_number'].nunique().reset\_index()

    max\_order = new\_df[new\_df['order\_number'] == new\_df['order\_number'].max()]

   #if we do it like this: new\_df[new\_df['order\_number'].max()], we would get just a number, not the row of df wehre the max value is!

    return max\_order[['customer\_number']]

**5.6. 1484. Group Sold Products By The Date**

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| sell\_date | date |

| product | varchar |

+-------------+---------+

There is no primary key (column with unique values) for this table. It may contain duplicates.

Each row of this table contains the product name and the date it was sold in a market.

Write a solution to find for each date the number of different products sold and their names.

The sold products names for each date should be sorted lexicographically.

Return the result table ordered by sell\_date.

The result format is in the following example.

**Example 1:**

**Input:**

Activities table:

+------------+------------+

| sell\_date | product |

+------------+------------+

| 2020-05-30 | Headphone |

| 2020-06-01 | Pencil |

| 2020-06-02 | Mask |

| 2020-05-30 | Basketball |

| 2020-06-01 | Bible |

| 2020-06-02 | Mask |

| 2020-05-30 | T-Shirt |

+------------+------------+

**Output:**

+------------+----------+------------------------------+

| sell\_date | num\_sold | products |

+------------+----------+------------------------------+

| 2020-05-30 | 3 | Basketball,Headphone,T-shirt |

| 2020-06-01 | 2 | Bible,Pencil |

| 2020-06-02 | 1 | Mask |

+------------+----------+------------------------------+

**Explanation:**

For 2020-05-30, Sold items were (Headphone, Basketball, T-shirt), we sort them lexicographically and separate them by a comma.

For 2020-06-01, Sold items were (Pencil, Bible), we sort them lexicographically and separate them by a comma.

For 2020-06-02, the Sold item is (Mask), we just return it.

import pandas as pd

def categorize\_products(activities: pd.DataFrame) -> pd.DataFrame:

    activities = activities.drop\_duplicates(subset=['sell\_date', 'product'])

    result = (activities.groupby('sell\_date')['product'].**agg(num\_sold = 'count'**, products = lambda x: ','.join(sorted(x)))).reset\_index()

    return result

**5.7. 1693. Daily Leads and Partners**

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| date\_id | date |

| make\_name | varchar |

| lead\_id | int |

| partner\_id | int |

+-------------+---------+

There is no primary key (column with unique values) for this table. It may contain duplicates.

This table contains the date and the name of the product sold and the IDs of the lead and partner it was sold to.

The name consists of only lowercase English letters.

For each date\_id and make\_name, find the number of **distinct** lead\_id's and **distinct** partner\_id's.

Return the result table in **any order**.

The result format is in the following example.

**Example 1:**

**Input:**

DailySales table:

+-----------+-----------+---------+------------+

| date\_id | make\_name | lead\_id | partner\_id |

+-----------+-----------+---------+------------+

| 2020-12-8 | toyota | 0 | 1 |

| 2020-12-8 | toyota | 1 | 0 |

| 2020-12-8 | toyota | 1 | 2 |

| 2020-12-7 | toyota | 0 | 2 |

| 2020-12-7 | toyota | 0 | 1 |

| 2020-12-8 | honda | 1 | 2 |

| 2020-12-8 | honda | 2 | 1 |

| 2020-12-7 | honda | 0 | 1 |

| 2020-12-7 | honda | 1 | 2 |

| 2020-12-7 | honda | 2 | 1 |

+-----------+-----------+---------+------------+

**Output:**

+-----------+-----------+--------------+-----------------+

| date\_id | make\_name | unique\_leads | unique\_partners |

+-----------+-----------+--------------+-----------------+

| 2020-12-8 | toyota | 2 | 3 |

| 2020-12-7 | toyota | 1 | 2 |

| 2020-12-8 | honda | 2 | 2 |

| 2020-12-7 | honda | 3 | 2 |

+-----------+-----------+--------------+-----------------+

**Explanation:**

For 2020-12-8, toyota gets leads = [0, 1] and partners = [0, 1, 2] while honda gets leads = [1, 2] and partners = [1, 2].

For 2020-12-7, toyota gets leads = [0] and partners = [1, 2] while honda gets leads = [0, 1, 2] and partners = [1, 2].

import pandas as pd

def daily\_leads\_and\_partners(daily\_sales: pd.DataFrame) -> pd.DataFrame:

    daily\_sales = daily\_sales.drop\_duplicates()

    new\_df = daily\_sales.groupby(['date\_id', 'make\_name']).agg({"lead\_id": "nunique", "partner\_id":"nunique"}).reset\_index()

    return new\_df.rename(columns = {'lead\_id':'unique\_leads', 'partner\_id':'unique\_partners'})

1. **DATA INTEGRATION**
   1. **1050. Actors and Directors Who Cooperated At Least Three Times**

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| actor\_id | int |

| director\_id | int |

| timestamp | int |

+-------------+---------+

timestamp is the primary key (column with unique values) for this table.

Write a solution to find all the pairs (actor\_id, director\_id) where the actor has cooperated with the director at least three times.

Return the result table in **any order**.

The result format is in the following example.

**Example 1:**

**Input:**

ActorDirector table:

+-------------+-------------+-------------+

| actor\_id | director\_id | timestamp |

+-------------+-------------+-------------+

| 1 | 1 | 0 |

| 1 | 1 | 1 |

| 1 | 1 | 2 |

| 1 | 2 | 3 |

| 1 | 2 | 4 |

| 2 | 1 | 5 |

| 2 | 1 | 6 |

+-------------+-------------+-------------+

**Output:**

+-------------+-------------+

| actor\_id | director\_id |

+-------------+-------------+

| 1 | 1 |

+-------------+-------------+

**Explanation:** The only pair is (1, 1) where they cooperated exactly 3 times.

import pandas as pd

def actors\_and\_directors(actor\_director: pd.DataFrame) -> pd.DataFrame:

    new\_df = actor\_director.groupby(['actor\_id', 'director\_id'])['timestamp'].count().reset\_index()

    return new\_df[new\_df['timestamp'] >= 3][['actor\_id','director\_id']]

* 1. **1378. Replace Employee ID With The Unique Identifier**

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| id | int |

| name | varchar |

+---------------+---------+

id is the primary key (column with unique values) for this table.

Each row of this table contains the id and the name of an employee in a company.

Table: EmployeeUNI

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| id | int |

| unique\_id | int |

+---------------+---------+

(id, unique\_id) is the primary key (combination of columns with unique values) for this table.

Each row of this table contains the id and the corresponding unique id of an employee in the company.

Write a solution to show the **unique ID**of each user, If a user does not have a unique ID replace just show null.

Return the result table in **any** order.

The result format is in the following example.

**Example 1:**

**Input:**

Employees table:

+----+----------+

| id | name |

+----+----------+

| 1 | Alice |

| 7 | Bob |

| 11 | Meir |

| 90 | Winston |

| 3 | Jonathan |

+----+----------+

EmployeeUNI table:

+----+-----------+

| id | unique\_id |

+----+-----------+

| 3 | 1 |

| 11 | 2 |

| 90 | 3 |

+----+-----------+

**Output:**

+-----------+----------+

| unique\_id | name |

+-----------+----------+

| null | Alice |

| null | Bob |

| 2 | Meir |

| 3 | Winston |

| 1 | Jonathan |

+-----------+----------+

**Explanation:**

Alice and Bob do not have a unique ID, We will show null instead.

The unique ID of Meir is 2.

The unique ID of Winston is 3.

The unique ID of Jonathan is 1.

import pandas as pd

def replace\_employee\_id(employees: pd.DataFrame, employee\_uni: pd.DataFrame) -> pd.DataFrame:

    merged\_df = employees.merge(right = employee\_uni, how = 'outer', on = 'id').dropna(subset = ['name'])

    return (merged\_df[['unique\_id', 'name']])

* 1. **1280. Students and Examinations**

+---------------+---------+

| Column Name | Type |

+---------------+---------+

| student\_id | int |

| student\_name | varchar |

+---------------+---------+

student\_id is the primary key (column with unique values) for this table.

Each row of this table contains the ID and the name of one student in the school.

Table: Subjects

+--------------+---------+

| Column Name | Type |

+--------------+---------+

| subject\_name | varchar |

+--------------+---------+

subject\_name is the primary key (column with unique values) for this table.

Each row of this table contains the name of one subject in the school.

Table: Examinations

+--------------+---------+

| Column Name | Type |

+--------------+---------+

| student\_id | int |

| subject\_name | varchar |

+--------------+---------+

There is no primary key (column with unique values) for this table. It may contain duplicates.

Each student from the Students table takes every course from the Subjects table.

Each row of this table indicates that a student with ID student\_id attended the exam of subject\_name.

Write a solution to find the number of times each student attended each exam.

Return the result table ordered by student\_id and subject\_name.

The result format is in the following example.

**Example 1:**

**Input:**

Students table:

+------------+--------------+

| student\_id | student\_name |

+------------+--------------+

| 1 | Alice |

| 2 | Bob |

| 13 | John |

| 6 | Alex |

+------------+--------------+

Subjects table:

+--------------+

| subject\_name |

+--------------+

| Math |

| Physics |

| Programming |

+--------------+

Examinations table:

+------------+--------------+

| student\_id | subject\_name |

+------------+--------------+

| 1 | Math |

| 1 | Physics |

| 1 | Programming |

| 2 | Programming |

| 1 | Physics |

| 1 | Math |

| 13 | Math |

| 13 | Programming |

| 13 | Physics |

| 2 | Math |

| 1 | Math |

+------------+--------------+

**Output:**

+------------+--------------+--------------+----------------+

| student\_id | student\_name | subject\_name | attended\_exams |

+------------+--------------+--------------+----------------+

| 1 | Alice | Math | 3 |

| 1 | Alice | Physics | 2 |

| 1 | Alice | Programming | 1 |

| 2 | Bob | Math | 1 |

| 2 | Bob | Physics | 0 |

| 2 | Bob | Programming | 1 |

| 6 | Alex | Math | 0 |

| 6 | Alex | Physics | 0 |

| 6 | Alex | Programming | 0 |

| 13 | John | Math | 1 |

| 13 | John | Physics | 1 |

| 13 | John | Programming | 1 |

+------------+--------------+--------------+----------------+

**Explanation:**

The result table should contain all students and all subjects.

Alice attended the Math exam 3 times, the Physics exam 2 times, and the Programming exam 1 time.

Bob attended the Math exam 1 time, the Programming exam 1 time, and did not attend the Physics exam.

Alex did not attend any exams.

John attended the Math exam 1 time, the Physics exam 1 time, and the Programming exam 1 time.

import pandas as pd

def students\_and\_examinations(students: pd.DataFrame, subjects: pd.DataFrame, examinations: pd.DataFrame) -> pd.DataFrame:

    examinations = examinations.groupby(['student\_id', 'subject\_name']).agg(attended\_exams=('subject\_name', 'count')).reset\_index()

    students = students.merge(subjects, how='cross')

    examinations = examinations.merge(students, on=['student\_id', 'subject\_name'],how='right')

    examinations = examinations.fillna(0)

    examinations = examinations.sort\_values(['student\_id', 'subject\_name'])

    return examinations[['student\_id', 'student\_name', 'subject\_name', 'attended\_exams']]

**6.4 570. Managers with at Least 5 Direct Reports**

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| id | int |

| name | varchar |

| department | varchar |

| managerId | int |

+-------------+---------+

id is the primary key (column with unique values) for this table.

Each row of this table indicates the name of an employee, their department, and the id of their manager.

If managerId is null, then the employee does not have a manager.

No employee will be the manager of themself.

Write a solution to find managers with at least **five direct reports**.

Return the result table in **any order**.

The result format is in the following example.

**Example 1:**

**Input:**

Employee table:

+-----+-------+------------+-----------+

| id | name | department | managerId |

+-----+-------+------------+-----------+

| 101 | John | A | None |

| 102 | Dan | A | 101 |

| 103 | James | A | 101 |

| 104 | Amy | A | 101 |

| 105 | Anne | A | 101 |

| 106 | Ron | B | 101 |

+-----+-------+------------+-----------+

**Output:**

+------+

| name |

+------+

| John |

+------+

import pandas as pd

def find\_managers(employee: pd.DataFrame) -> pd.DataFrame:

    grouped\_df = employee.groupby('managerId').agg(count = ('name', 'count')).reset\_index()

    grouped\_df = grouped\_df.rename(columns = {'managerId':'id'})

    merged\_df = employee[['id', 'name']].merge(grouped\_df, how = 'right', on = 'id').dropna()

    return merged\_df[merged\_df['count'] >= 5][['name']]

**my solution worked, but this is a cleaner solution in which they create a list of all managerIds with count >=5 and then print the names of ids from this list (.isin()):**

def find\_managers(employee: pd.DataFrame) -> pd.DataFrame:

    managers = employee.groupby('managerId', as\_index=False).agg(reporting=('id', 'count')).**query**('5 <= reporting')['managerId']

    return employee[employee['id'].**isin**(managers)][['name']]

* 1. **607. Sales Person**

+-----------------+---------+

| Column Name | Type |

+-----------------+---------+

| sales\_id | int |

| name | varchar |

| salary | int |

| commission\_rate | int |

| hire\_date | date |

+-----------------+---------+

sales\_id is the primary key (column with unique values) for this table.

Each row of this table indicates the name and the ID of a salesperson alongside their salary, commission rate, and hire date.

Table: Company

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| com\_id | int |

| name | varchar |

| city | varchar |

+-------------+---------+

com\_id is the primary key (column with unique values) for this table.

Each row of this table indicates the name and the ID of a company and the city in which the company is located.

Table: Orders

+-------------+------+

| Column Name | Type |

+-------------+------+

| order\_id | int |

| order\_date | date |

| com\_id | int |

| sales\_id | int |

| amount | int |

+-------------+------+

order\_id is the primary key (column with unique values) for this table.

com\_id is a foreign key (reference column) to com\_id from the Company table.

sales\_id is a foreign key (reference column) to sales\_id from the SalesPerson table.

Each row of this table contains information about one order. This includes the ID of the company, the ID of the salesperson, the date of the order, and the amount paid.

Write a solution to find the names of all the salespersons who did not have any orders related to the company with the name **"RED"**.

Return the result table in **any order**.

The result format is in the following example.

**Example 1:**

**Input:**

SalesPerson table:

+----------+------+--------+-----------------+------------+

| sales\_id | name | salary | commission\_rate | hire\_date |

+----------+------+--------+-----------------+------------+

| 1 | John | 100000 | 6 | 4/1/2006 |

| 2 | Amy | 12000 | 5 | 5/1/2010 |

| 3 | Mark | 65000 | 12 | 12/25/2008 |

| 4 | Pam | 25000 | 25 | 1/1/2005 |

| 5 | Alex | 5000 | 10 | 2/3/2007 |

+----------+------+--------+-----------------+------------+

Company table:

+--------+--------+----------+

| com\_id | name | city |

+--------+--------+----------+

| 1 | RED | Boston |

| 2 | ORANGE | New York |

| 3 | YELLOW | Boston |

| 4 | GREEN | Austin |

+--------+--------+----------+

Orders table:

+----------+------------+--------+----------+--------+

| order\_id | order\_date | com\_id | sales\_id | amount |

+----------+------------+--------+----------+--------+

| 1 | 1/1/2014 | 3 | 4 | 10000 |

| 2 | 2/1/2014 | 4 | 5 | 5000 |

| 3 | 3/1/2014 | 1 | 1 | 50000 |

| 4 | 4/1/2014 | 1 | 4 | 25000 |

+----------+------------+--------+----------+--------+

**Output:**

+------+

| name |

+------+

| Amy |

| Mark |

| Alex |

+------+

**Explanation:**

According to orders 3 and 4 in the Orders table, it is easy to tell that only salesperson John and Pam have sales to company RED, so we report all the other names in the table salesperson.

import pandas as pd

def sales\_person(sales\_person: pd.DataFrame, company: pd.DataFrame, orders: pd.DataFrame) -> pd.DataFrame:

    filtered\_sales\_person = sales\_person[~sales\_person['sales\_id'].isin(orders[orders['com\_id'].isin(company[company['name'] == 'RED']['com\_id'])]['sales\_id'])]

    return filtered\_sales\_person[['name']]