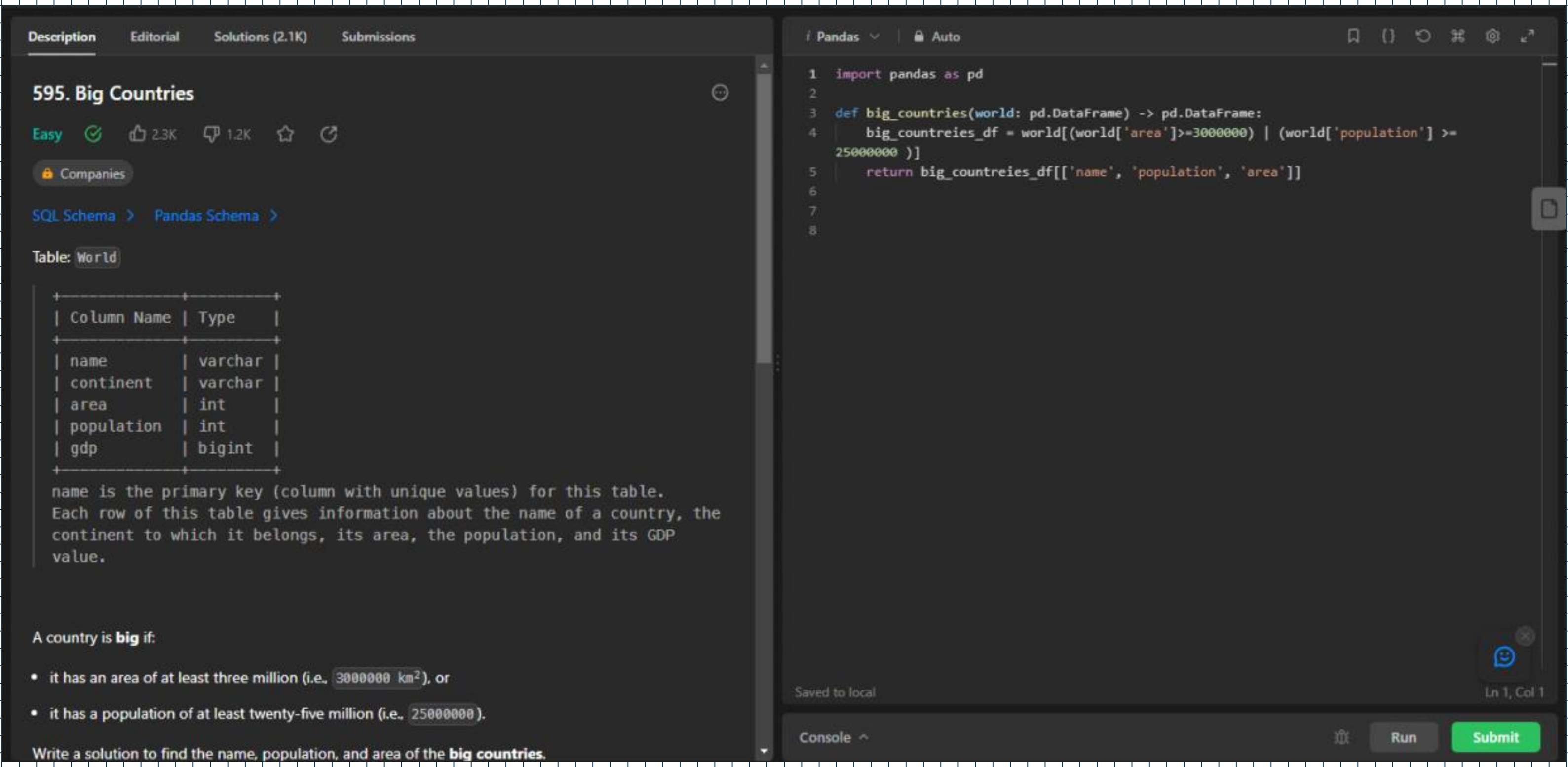
first

środa, 27 września 2023 16:36



595. Big Countries

Table: `World`

```
+-------------+---------+
| 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:

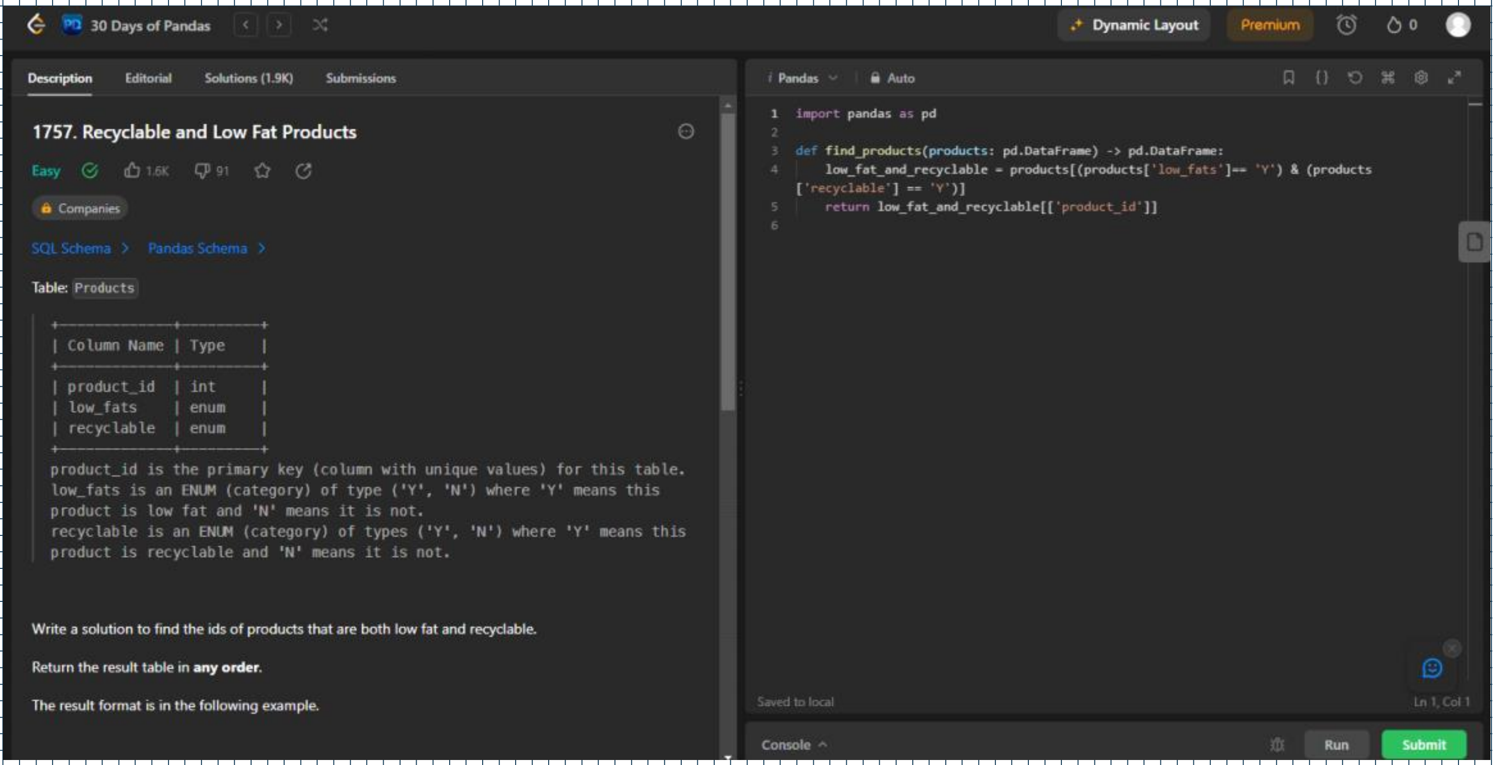
- it has an area of at least three million (i.e., `3000000 km²`), or
- it has a population of at least twenty-five million (i.e., `25000000`).

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

```python
import pandas as pd

def big_countries(world: pd.DataFrame) -> pd.DataFrame:
    big_countreies_df = world[(world['area']>=3000000) | (world['population'] >= 25000000 )]
    return big_countreies_df[['name', 'population', 'area']]
```

second

środa, 27 września 2023 16:36



1757. Recyclable and Low Fat Products

SQL Schema > Pandas Schema >

Table: `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.

```python
import pandas as pd

def find_products(products: pd.DataFrame) -> pd.DataFrame:
    low_fat_and_recyclable = products[(products['low_fats']== 'Y') & (products['recyclable'] == 'Y')]
    return low_fat_and_recyclable[['product_id']]
```

Merge isin

środa, 27 września 2023 16:36

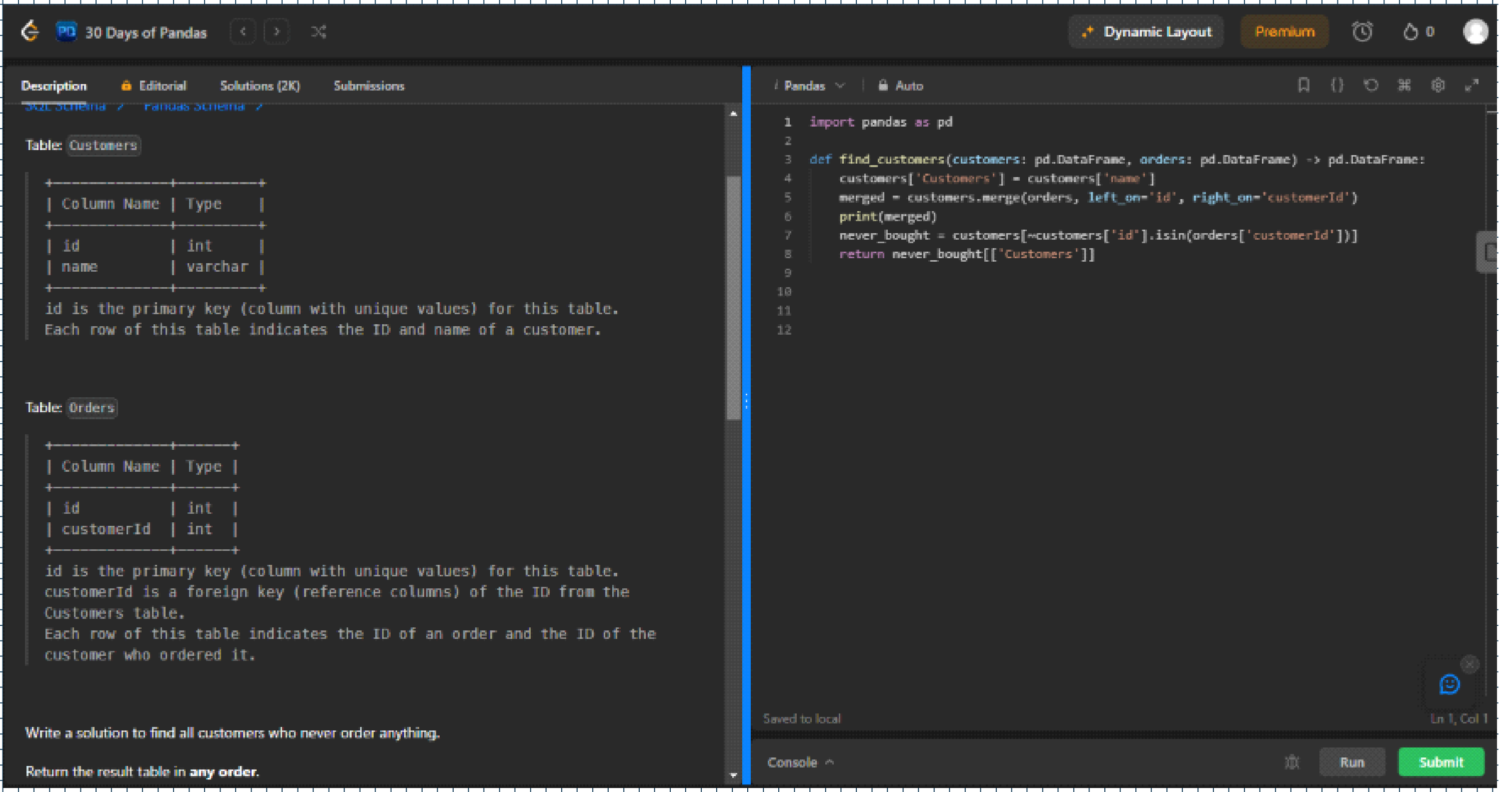


Table: Customers

```
+-------------+---------+
| 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.
```

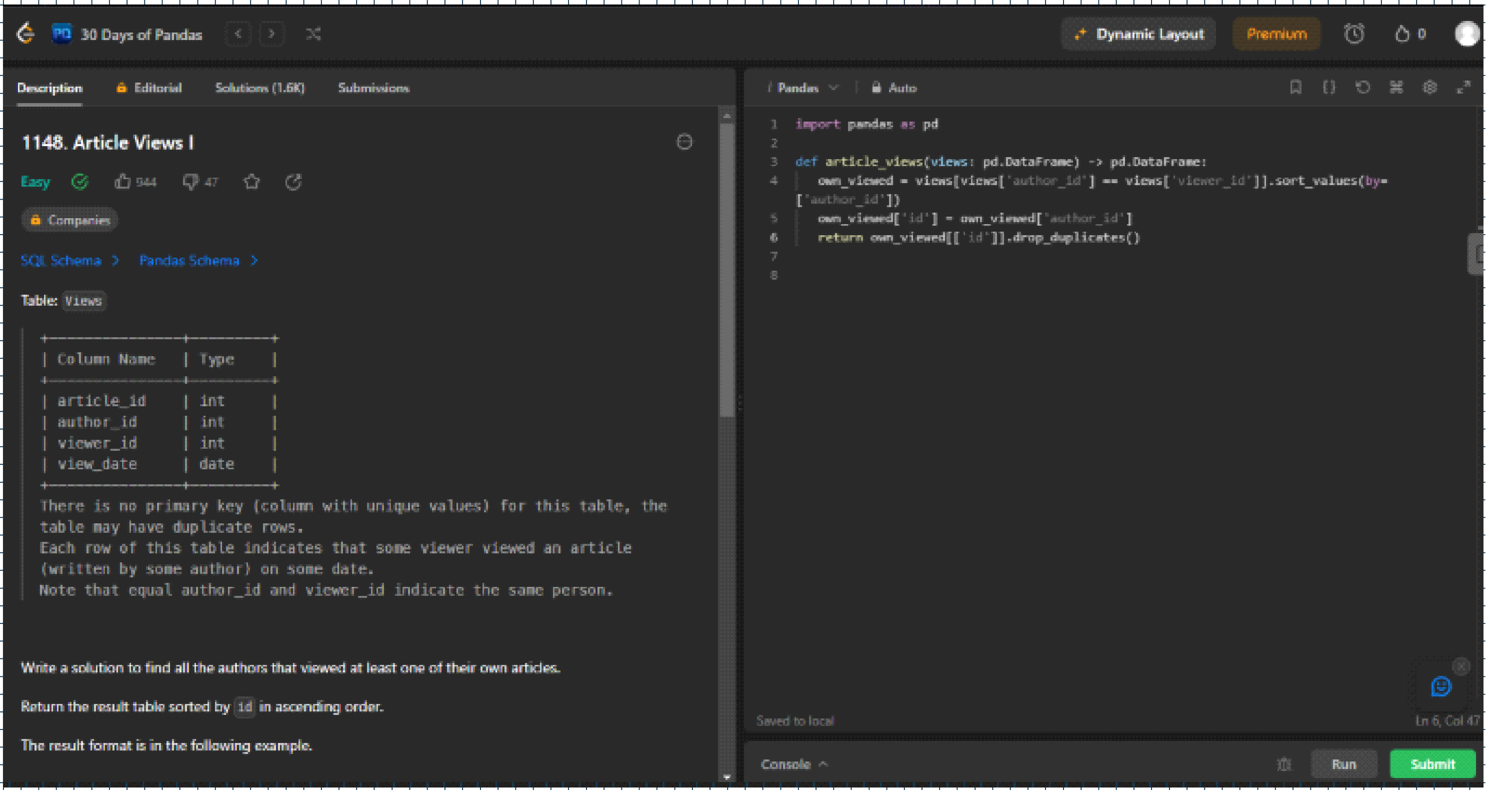
Write a solution to find all customers who never order anything.

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

```python
import pandas as pd

def find_customers(customers: pd.DataFrame, orders: pd.DataFrame) -> pd.DataFrame:
    customers['Customers'] = customers['name']
    merged = customers.merge(orders, left_on='id', right_on='customerId')
    print(merged)
    never_bought = customers[~customers['id'].isin(orders['customerId'])]
    return never_bought[['Customers']]
```

sort

środa, 27 września 2023 16:37



1148. Article Views I

Table: `Views`

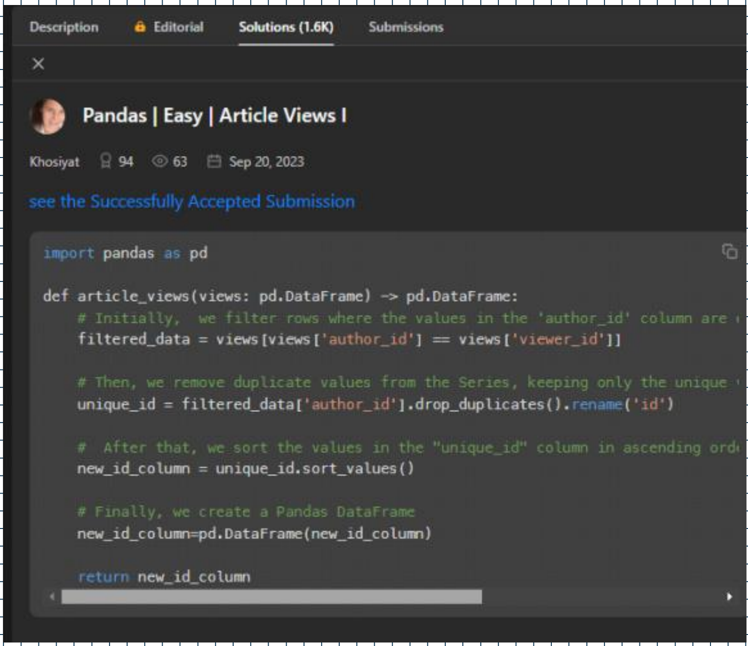
```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| article_id    | int     |
| author_id     | int     |
| viewer_id     | int     |
| view_date     | date    |
+---------------+---------+
There is no primary key (column with unique values) for this table, the
table 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.

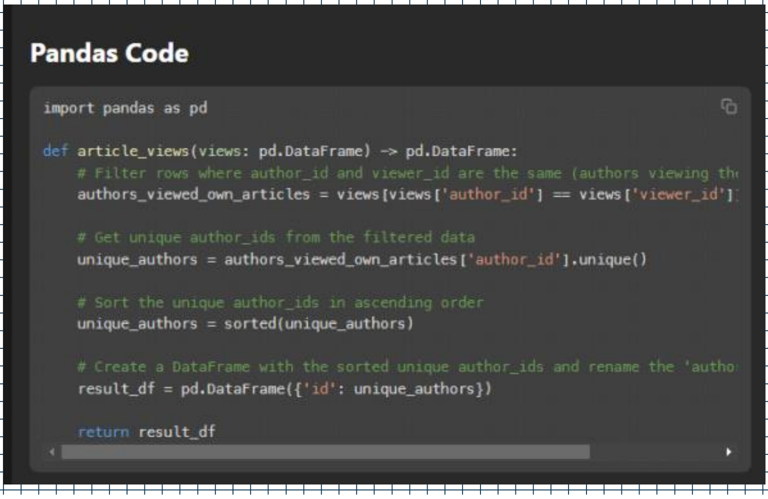
```python
import pandas as pd

def article_views(views: pd.DataFrame) -> pd.DataFrame:
    own_viewed = views[views['author_id'] == views['viewer_id']].sort_values(by=['author_id'])
    own_viewed['id'] = own_viewed['author_id']
    return own_viewed[['id']].drop_duplicates()
```



Pandas | Easy | Article Views I

see the Successfully Accepted Submission

```python
import pandas as pd

def article_views(views: pd.DataFrame) -> pd.DataFrame:
    # Initially,  we filter rows where the values in the 'author_id' column are 
    filtered_data = views[views['author_id'] == views['viewer_id']]

    # Then, we remove duplicate values from the Series, keeping only the unique 
    unique_id = filtered_data['author_id'].drop_duplicates().rename('id')

    #  After that, we sort the values in the "unique_id" column in ascending ord
    new_id_column = unique_id.sort_values()

    # Finally, we create a Pandas DataFrame
    new_id_column=pd.DataFrame(new_id_column)

    return new_id_column
```

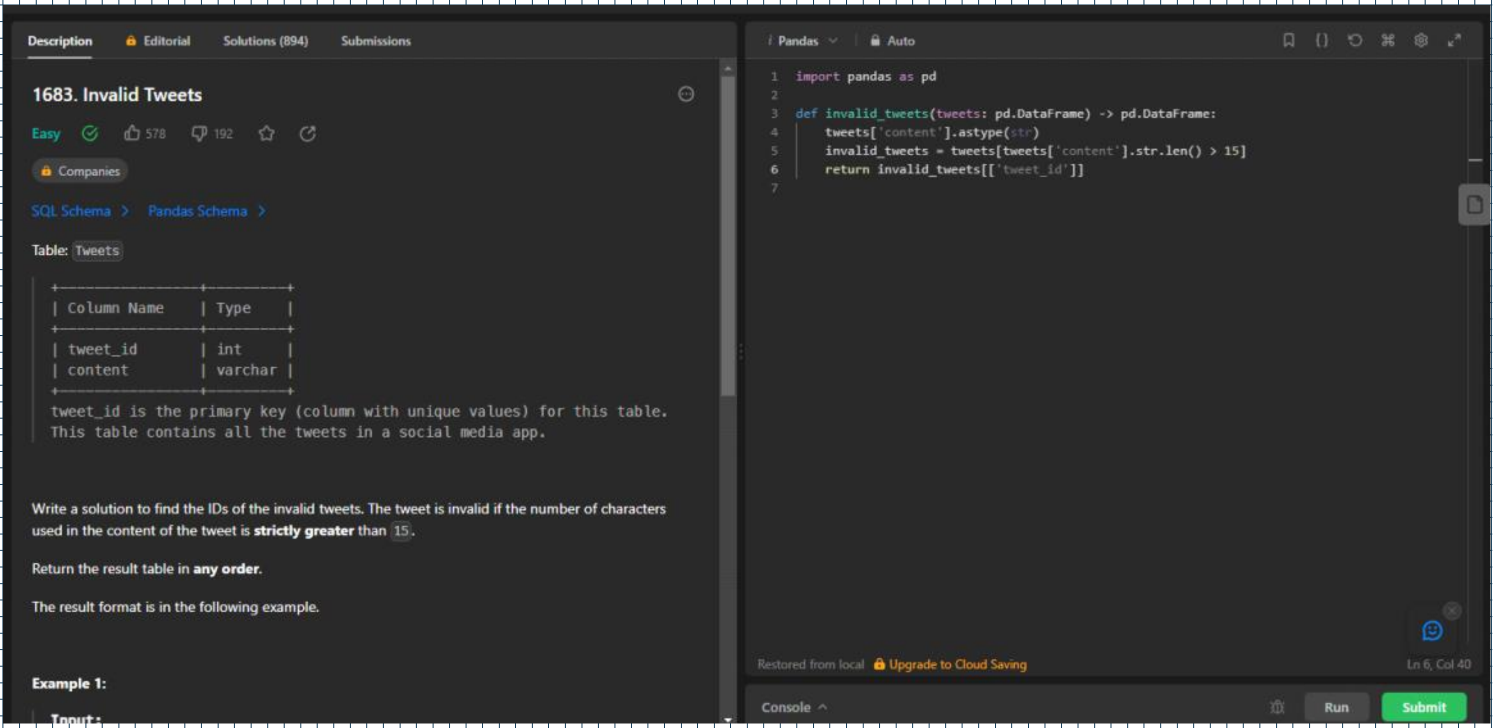


Pandas Code

```python
import pandas as pd

def article_views(views: pd.DataFrame) -> pd.DataFrame:
    # Filter rows where author_id and viewer_id are the same (authors viewing th
    authors_viewed_own_articles = views[views['author_id'] == views['viewer_id']]

    # Get unique author_ids from the filtered data
    unique_authors = authors_viewed_own_articles['author_id'].unique()

    # Sort the unique author_ids in ascending order
    unique_authors = sorted(unique_authors)

    # Create a DataFrame with the sorted unique author_ids and rename the 'autho
    result_df = pd.DataFrame({'id': unique_authors})

    return result_df
```

As type str len

środa, 27 września 2023 17:17



1683. Invalid Tweets

Table: 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:

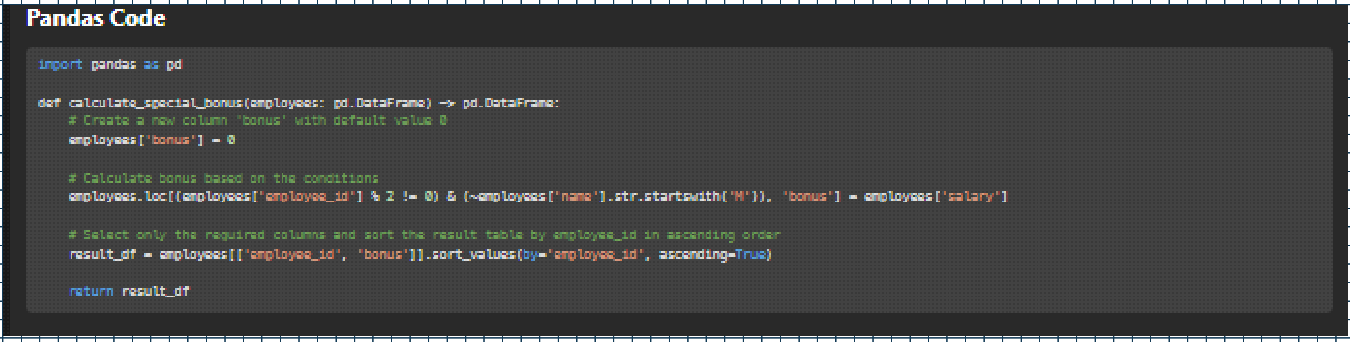
```python
import pandas as pd

def invalid_tweets(tweets: pd.DataFrame) -> pd.DataFrame:
    tweets['content'].astype(str)
    invalid_tweets = tweets[tweets['content'].str.len() > 15]
    return invalid_tweets[['tweet_id']]
```

Starts with

środa, 27 września 2023 18:57

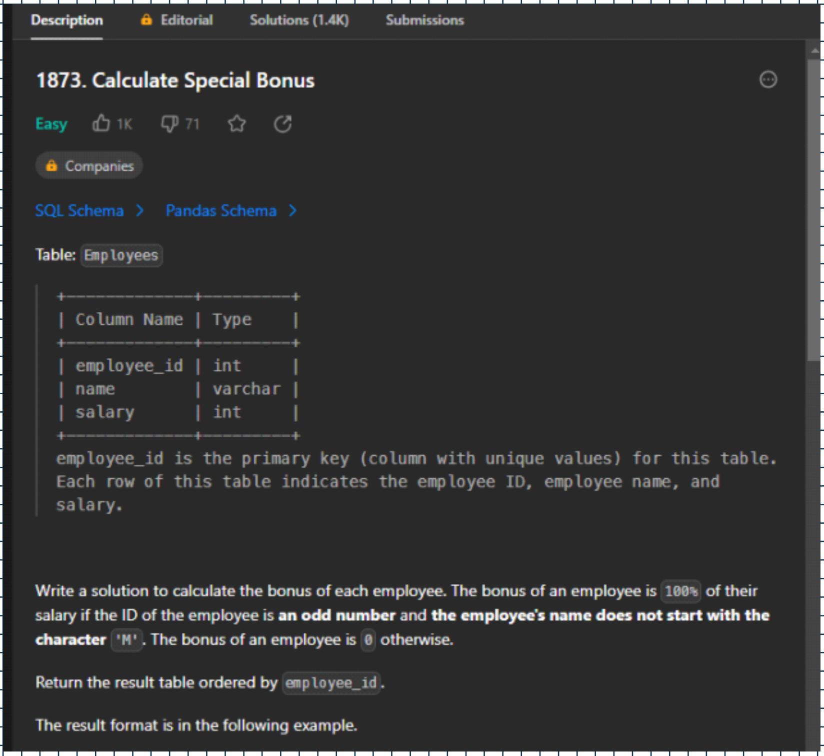
import pandas as pd def calculate_special_bonus(employees: pd.DataFrame) -> pd.DataFrame: # Create a new column 'bonus' with default value 0 employees['bonus'] = 0 # Calculate bonus based on the conditions employees.loc[(employees['employee_id'] % 2 != 0) & (~employees['name'].str.startswith('M')), 'bonus'] = employees['salary'] # Select only the required columns and sort the result table by employee_id in ascending order result_df = employees[['employee_id', 'bonus']].sort_values(by='employee_id', ascending=True) return result_df

Z <https://leetcode.com/problems/calculate-special-bonus/solutions/3853095/pandas-mysql-an-effortless-and-simple-approach-with-comments/?envType=study-plan-v2&envId=30-days-of-pandas&lang=pythondata>



Pandas Code

```python
import pandas as pd

def calculate_special_bonus(employees: pd.DataFrame) -> pd.DataFrame:
    # Create a new column 'bonus' with default value 0
    employees['bonus'] = 0

    # Calculate bonus based on the conditions
    employees.loc[(employees['employee_id'] % 2 != 0) & (~employees['name'].str.startswith('M')), 'bonus'] = employees['salary']

    # Select only the required columns and sort the result table by employee_id in ascending order
    result_df = employees[['employee_id', 'bonus']].sort_values(by='employee_id', ascending=True)

    return result_df
```



1873. Calculate Special Bonus

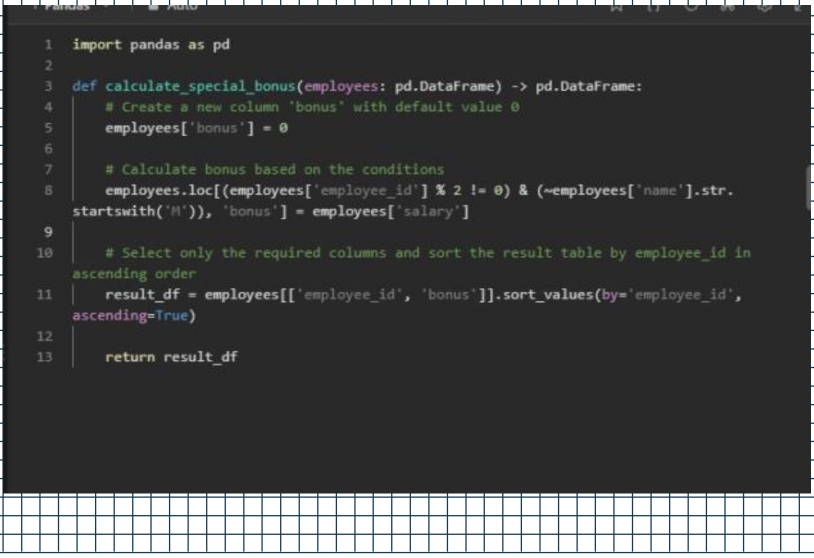
Table: `Employees`

```
+-------------+---------+
| 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`.

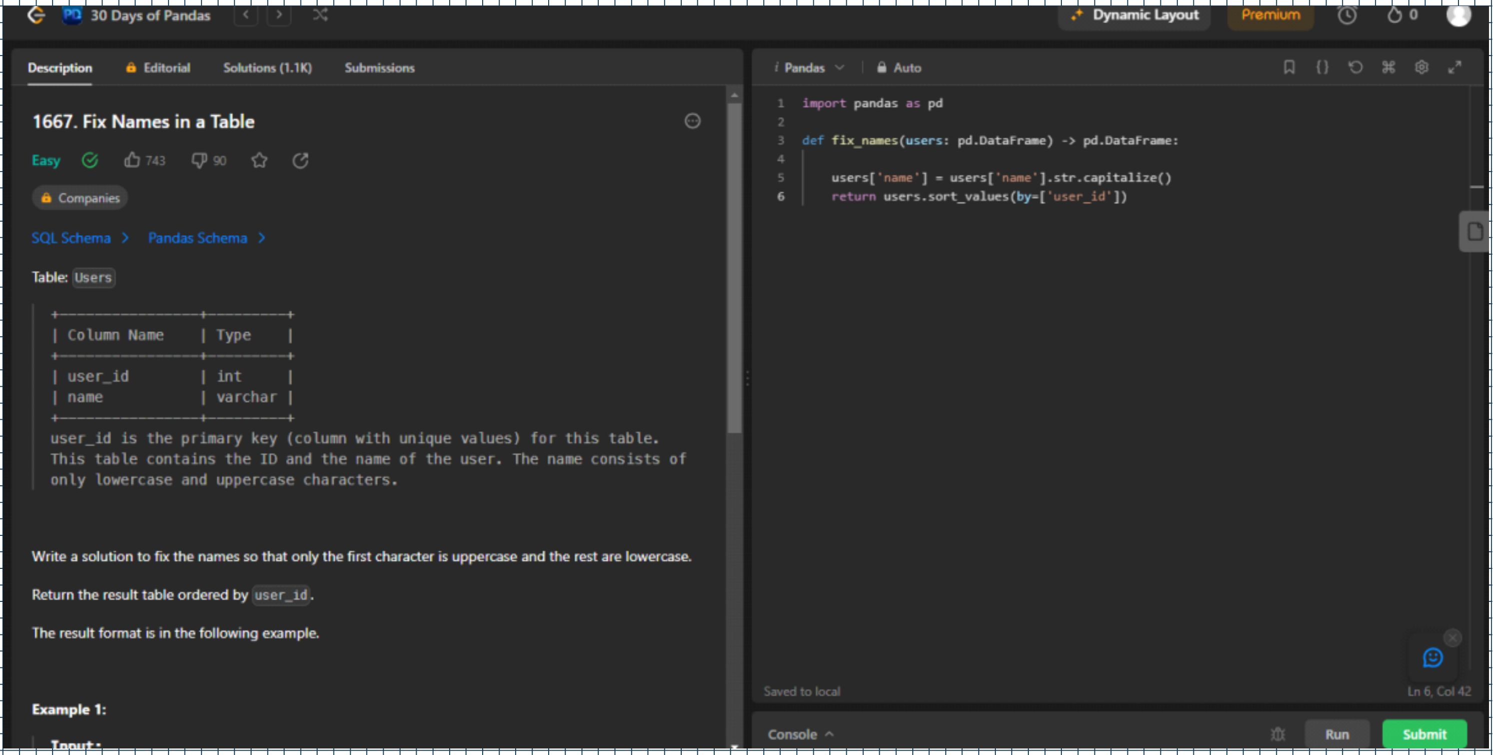
The result format is in the following example.



```python
import pandas as pd

def calculate_special_bonus(employees: pd.DataFrame) -> pd.DataFrame:
    # Create a new column 'bonus' with default value 0
    employees['bonus'] = 0

    # Calculate bonus based on the conditions
    employees.loc[(employees['employee_id'] % 2 != 0) & (~employees['name'].str.startswith('M')), 'bonus'] = employees['salary']

    # Select only the required columns and sort the result table by employee_id in ascending order
    result_df = employees[['employee_id', 'bonus']].sort_values(by='employee_id', ascending=True)

    return result_df
```

capitalize

piątek, 29 września 2023 15:15



1667. Fix Names in a Table

Easy 743 90

Companies

SQL Schema > Pandas Schema >

Table: `Users`

```
+----------------+---------+
| 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.

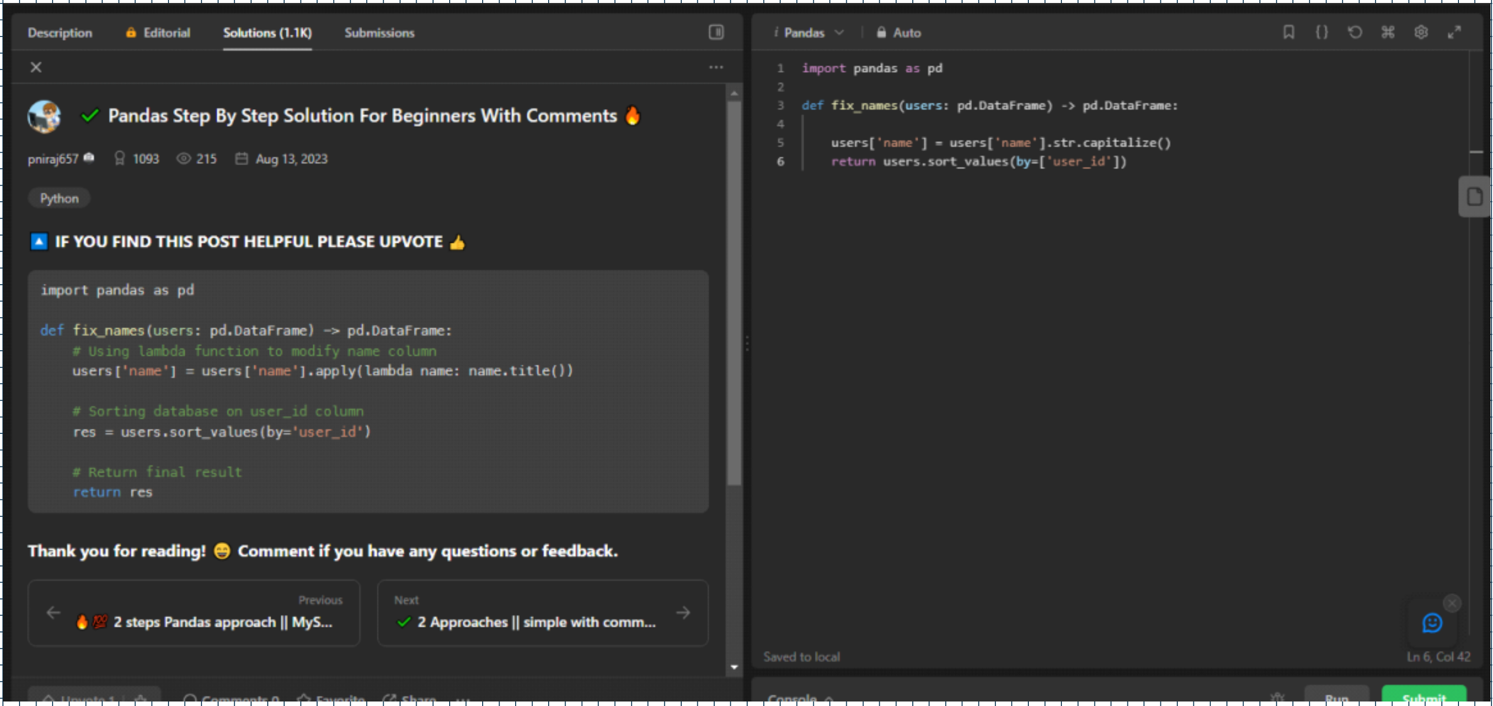
Return the result table ordered by `user_id`.

The result format is in the following example.

Example 1:

Input:

```python
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'])
```



✓ Pandas Step By Step Solution For Beginners With Comments 🔥

pniraj657 1093 215 Aug 13, 2023

🔼 **IF YOU FIND THIS POST HELPFUL PLEASE UPVOTE 👍**

```python
import pandas as pd

def fix_names(users: pd.DataFrame) -> pd.DataFrame:
    # Using lambda function to modify name column
    users['name'] = users['name'].apply(lambda name: name.title())

    # Sorting database on user_id column
    res = users.sort_values(by='user_id')

    # Return final result
    return res
```

Thank you for reading! 😄 Comment if you have any questions or feedback.

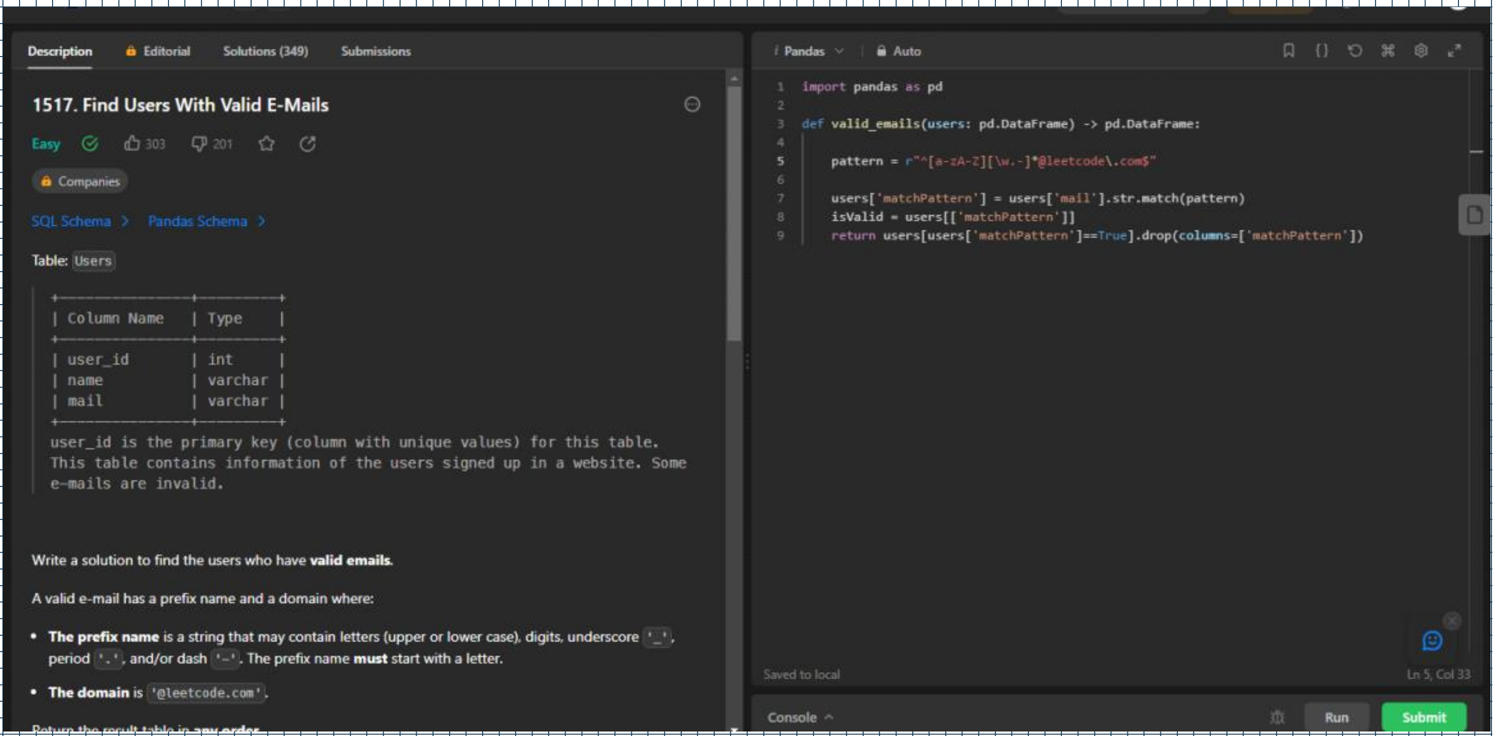
← Previous: 🔥💯 2 steps Pandas approach || MyS...

Next: ✓ 2 Approaches || simple with comm... →

```python
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'])
```

Regex exercise

piątek, 29 września 2023 18:17



1517. Find Users With Valid E-Mails

Easy 303 201

Companies

SQL Schema > Pandas Schema >

Table: `Users`

```
+---------------+---------+
| 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.
```

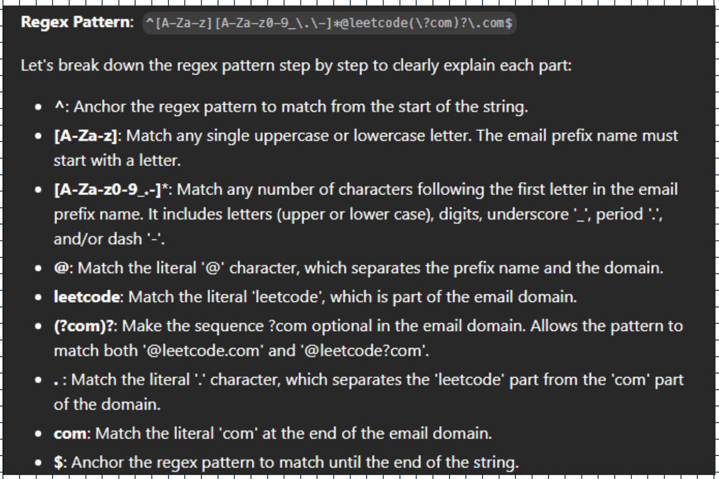
Write a solution to find the users who have **valid emails**.

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**.

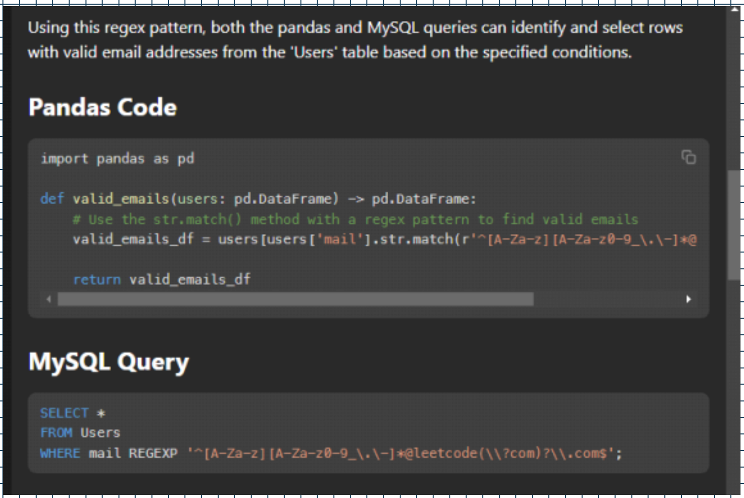
```python
import pandas as pd

def valid_emails(users: pd.DataFrame) -> pd.DataFrame:

    pattern = r"^[a-zA-Z][\w.-]*@leetcode\.com$"

    users['matchPattern'] = users['mail'].str.match(pattern)
    isValid = users[['matchPattern']]
    return users[users['matchPattern']==True].drop(columns=['matchPattern'])
```



Regex Pattern: `^[A-Za-z][A-Za-z0-9_\.\-]*@leetcode(\?com)?\.com$`

Let's break down the regex pattern step by step to clearly explain each part:

- **^**: Anchor the regex pattern to match from the start of the string.
- **[A-Za-z]**: Match any single uppercase or lowercase letter. The email prefix name must start with a letter.
- **[A-Za-z0-9_.-]***: Match any number of characters following the first letter in the email prefix name. It includes letters (upper or lower case), digits, underscore '_', period '.', and/or dash '-'.
- **@**: Match the literal '@' character, which separates the prefix name and the domain.
- **leetcode**: Match the literal 'leetcode', which is part of the email domain.
- **(?com)?**: Make the sequence ?com optional in the email domain. Allows the pattern to match both '@leetcode.com' and '@leetcode?com'.
- **.** : Match the literal '.' character, which separates the 'leetcode' part from the 'com' part of the domain.
- **com**: Match the literal 'com' at the end of the email domain.
- **$**: Anchor the regex pattern to match until the end of the string.



Using this regex pattern, both the pandas and MySQL queries can identify and select rows with valid email addresses from the 'Users' table based on the specified conditions.

Pandas Code

```python
import pandas as pd

def valid_emails(users: pd.DataFrame) -> pd.DataFrame:
    # Use the str.match() method with a regex pattern to find valid emails
    valid_emails_df = users[users['mail'].str.match(r'^[A-Za-z][A-Za-z0-9_\.\-]*@

    return valid_emails_df
```

MySQL Query

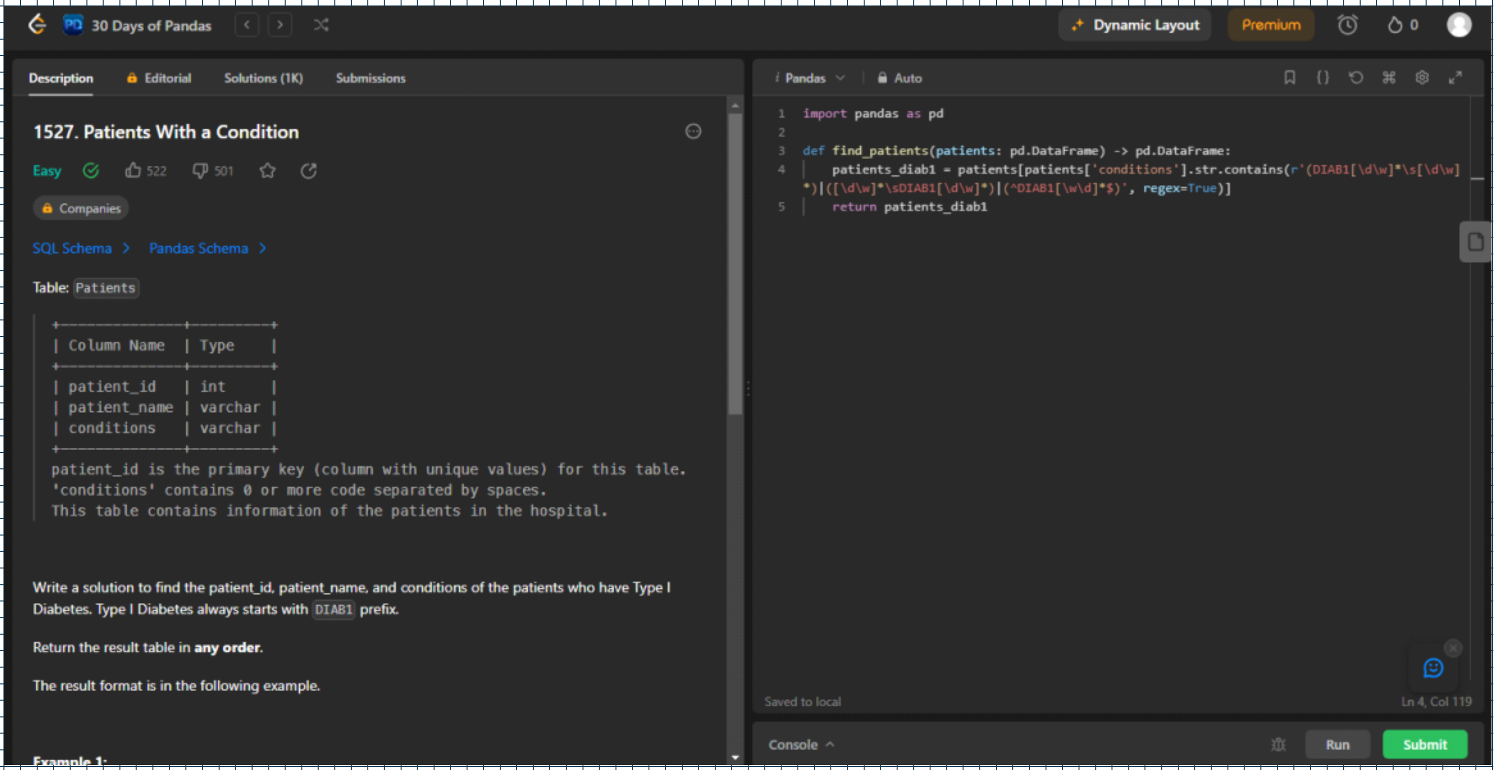
```sql
SELECT *
FROM Users
WHERE mail REGEXP '^[A-Za-z][A-Za-z0-9_\.\-]*@leetcode(\\?com)?\\.com$';
```

import pandas as pd def valid_emails(users: pd.DataFrame) -> pd.DataFrame: result_df = users[users['mail'].str.match(r'^[a-zA-Z][a-zA-Z0-9_.-]*@leetcode\.com')] return result_df

Z <https://leetcode.com/problems/find-users-with-valid-e-mails/solutions/3919566/easy-solution-pandas-and-mysql/?envType=study-plan-v2&envId=30-days-of-pandas&lang=pythondata>

regex

poniedziałek, 2 października 2023 12:36



1527. Patients With a Condition

Easy 522 501

Companies

SQL Schema > Pandas Schema >

Table: `Patients`

```
+--------------+---------+
| 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.

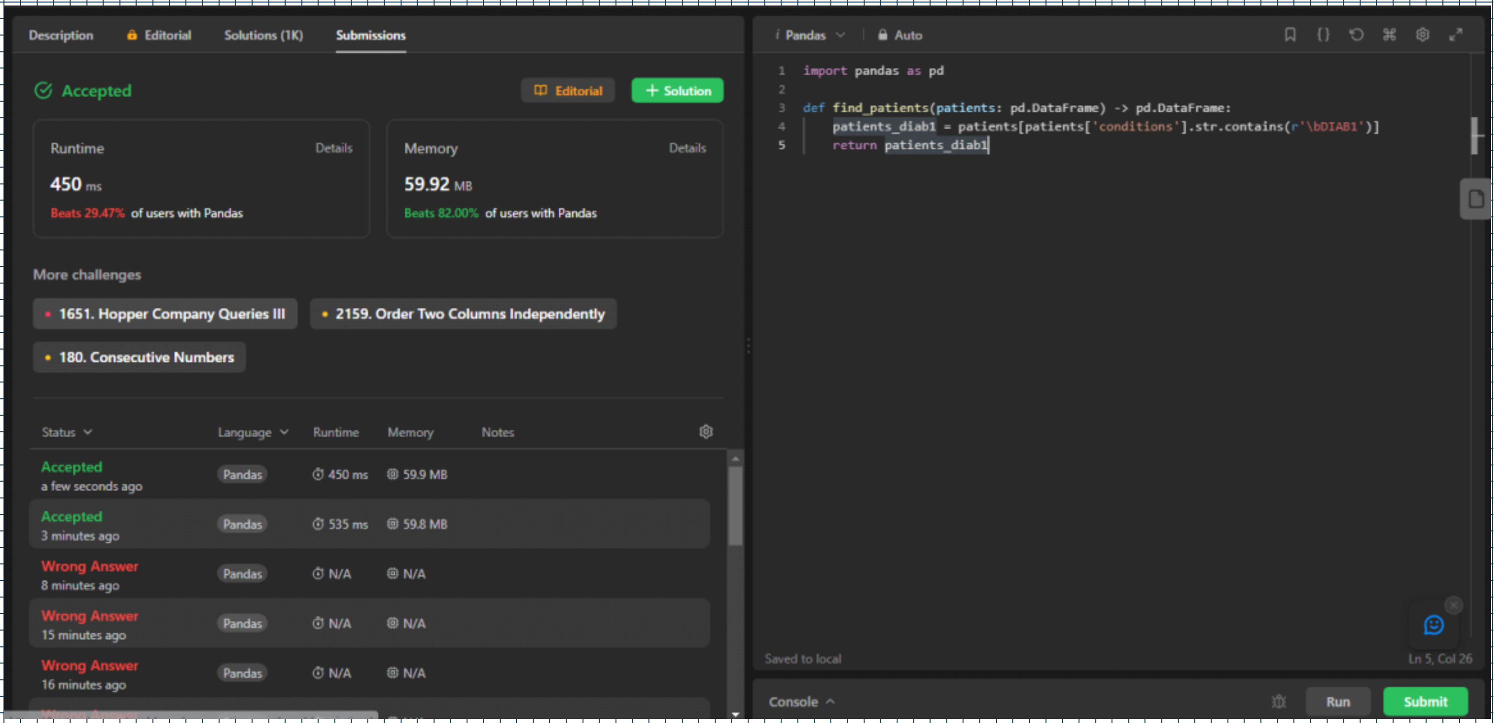
Return the result table in **any order**.

The result format is in the following example.

Example 1:

```python
import pandas as pd

def find_patients(patients: pd.DataFrame) -> pd.DataFrame:
    patients_diab1 = patients[patients['conditions'].str.contains(r'(DIAB1[\d\w]*\s[\d\w]*)|([\d\w]*\sDIAB1[\d\w]*)|(^DIAB1[\w\d]*$)', regex=True)]
    return patients_diab1
```

Saved to local Ln 4, Col 119



Accepted

Runtime Details

450 ms

Beats 29.47% of users with Pandas

Memory Details

59.92 MB

Beats 82.00% of users with Pandas

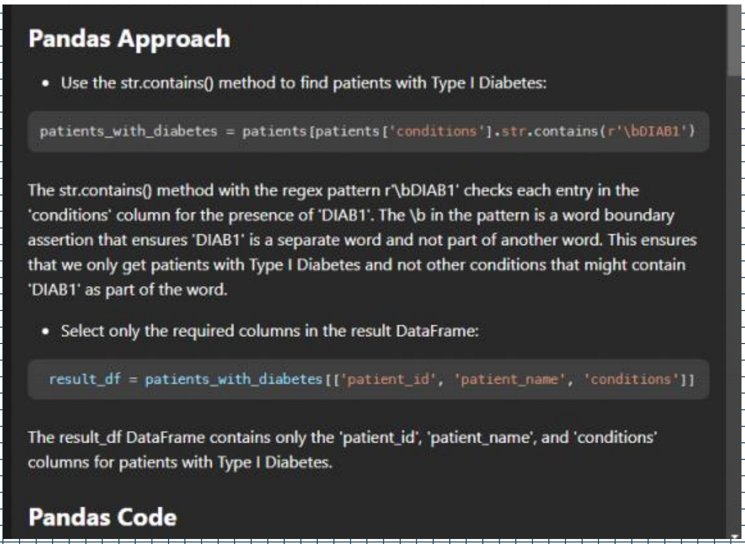
More challenges

- 1651. Hopper Company Queries III
- 2159. Order Two Columns Independently
- 180. Consecutive Numbers

Status	Language	Runtime	Memory	Notes
Accepted a few seconds ago	Pandas	450 ms	59.9 MB	
Accepted 3 minutes ago	Pandas	535 ms	59.8 MB	
Wrong Answer 8 minutes ago	Pandas	N/A	N/A	
Wrong Answer 15 minutes ago	Pandas	N/A	N/A	
Wrong Answer 16 minutes ago	Pandas	N/A	N/A	

```python
import pandas as pd

def find_patients(patients: pd.DataFrame) -> pd.DataFrame:
    patients_diab1 = patients[patients['conditions'].str.contains(r'\bDIAB1')]
    return patients_diab1
```

Saved to local Ln 5, Col 26



Pandas Approach

- Use the str.contains() method to find patients with Type I Diabetes:

```
patients_with_diabetes = patients[patients['conditions'].str.contains(r'\bDIAB1')
```

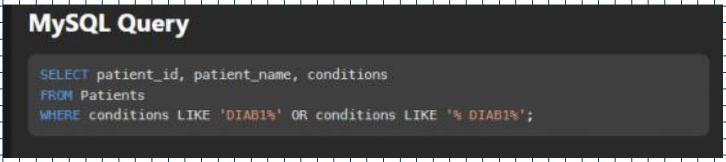
The str.contains() method with the regex pattern r'\bDIAB1' checks each entry in the 'conditions' column for the presence of 'DIAB1'. The \b in the pattern is a word boundary assertion that ensures 'DIAB1' is a separate word and not part of another word. This ensures that we only get patients with Type I Diabetes and not other conditions that might contain 'DIAB1' as part of the word.

- Select only the required columns in the result DataFrame:

```
 result_df = patients_with_diabetes[['patient_id', 'patient_name', 'conditions']]
```

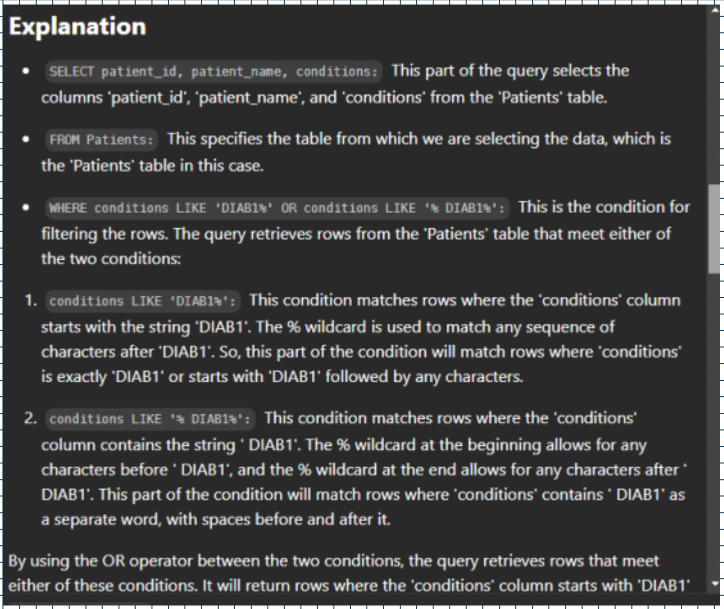
The result_df DataFrame contains only the 'patient_id', 'patient_name', and 'conditions' columns for patients with Type I Diabetes.

Pandas Code



MySQL Query

```
SELECT patient_id, patient_name, conditions
FROM Patients
WHERE conditions LIKE 'DIAB1%' OR conditions LIKE '% DIAB1%';
```

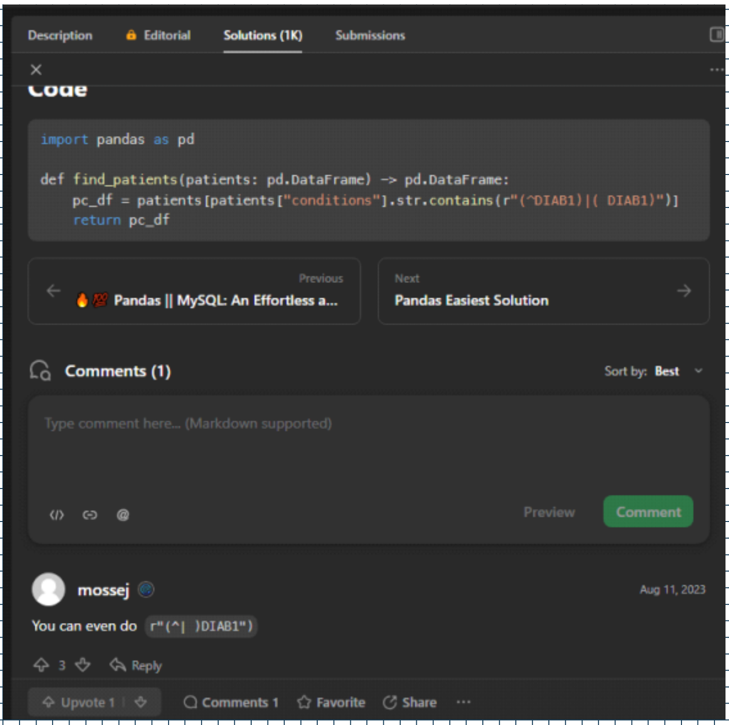


Explanation

- `SELECT patient_id, patient_name, conditions:` This part of the query selects the columns 'patient_id', 'patient_name', and 'conditions' from the 'Patients' table.
- `FROM Patients:` This specifies the table from which we are selecting the data, which is the 'Patients' table in this case.
- `WHERE conditions LIKE 'DIAB1%' OR conditions LIKE '% DIAB1%':` This is the condition for filtering the rows. The query retrieves rows from the 'Patients' table that meet either of the two conditions:

1. `conditions LIKE 'DIAB1%':` This condition matches rows where the 'conditions' column starts with the string 'DIAB1'. The % wildcard is used to match any sequence of characters after 'DIAB1'. So, this part of the condition will match rows where 'conditions' is exactly 'DIAB1' or starts with 'DIAB1' followed by any characters.
2. `conditions LIKE '% DIAB1%':` This condition matches rows where the 'conditions' column contains the string ' DIAB1'. The % wildcard at the beginning allows for any characters before ' DIAB1', and the % wildcard at the end allows for any characters after ' DIAB1'. This part of the condition will match rows where 'conditions' contains ' DIAB1' as a separate word, with spaces before and after it.

By using the OR operator between the two conditions, the query retrieves rows that meet either of these conditions. It will return rows where the 'conditions' column starts with 'DIAB1'



Code

```python
import pandas as pd

def find_patients(patients: pd.DataFrame) -> pd.DataFrame:
    pc_df = patients[patients["conditions"].str.contains(r"(^DIAB1)|( DIAB1)")]
    return pc_df
```

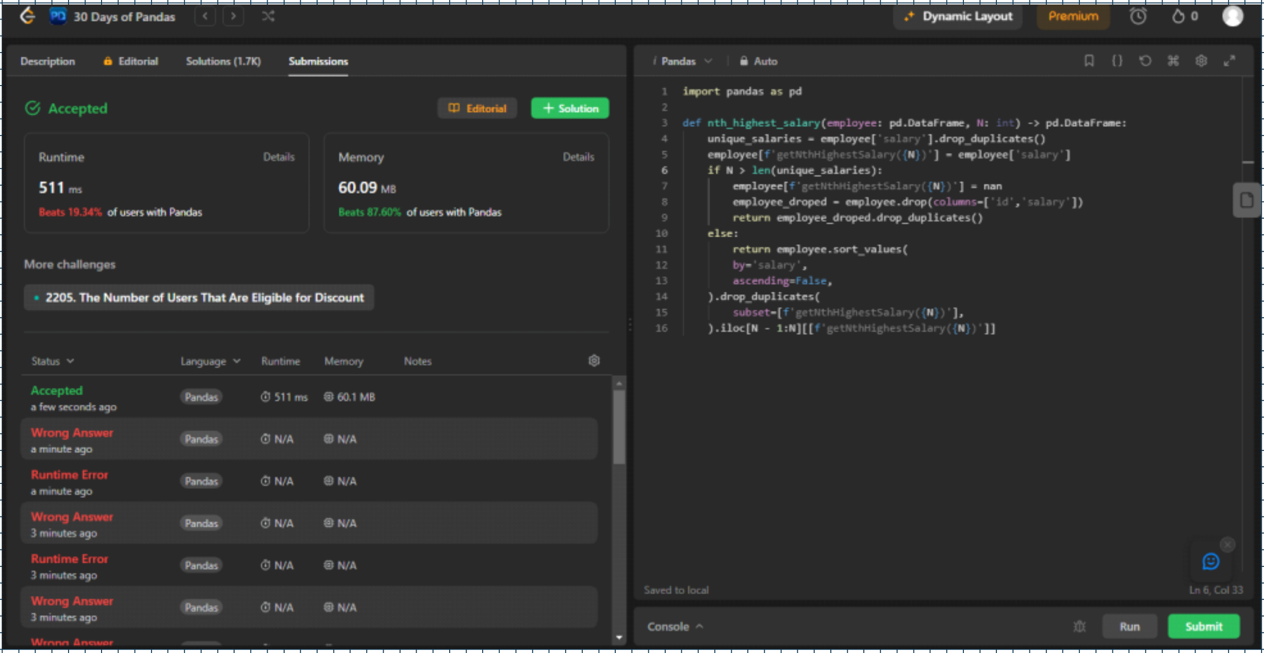
Previous
🔥💯 Pandas || MySQL: An Effortless a...
Next
Pandas Easiest Solution
Comments (1)
Sort by: Best
Type comment here... (Markdown supported)
Preview
Comment

mossej Aug 11, 2023

You can even do `r"(^| )DIAB1")`

Nth biggest value

poniedziałek, 2 października 2023 13:34



Accepted

Runtime Details

511 ms

Beats 19.34% of users with Pandas

Memory Details

60.09 MB

Beats 87.60% of users with Pandas

More challenges

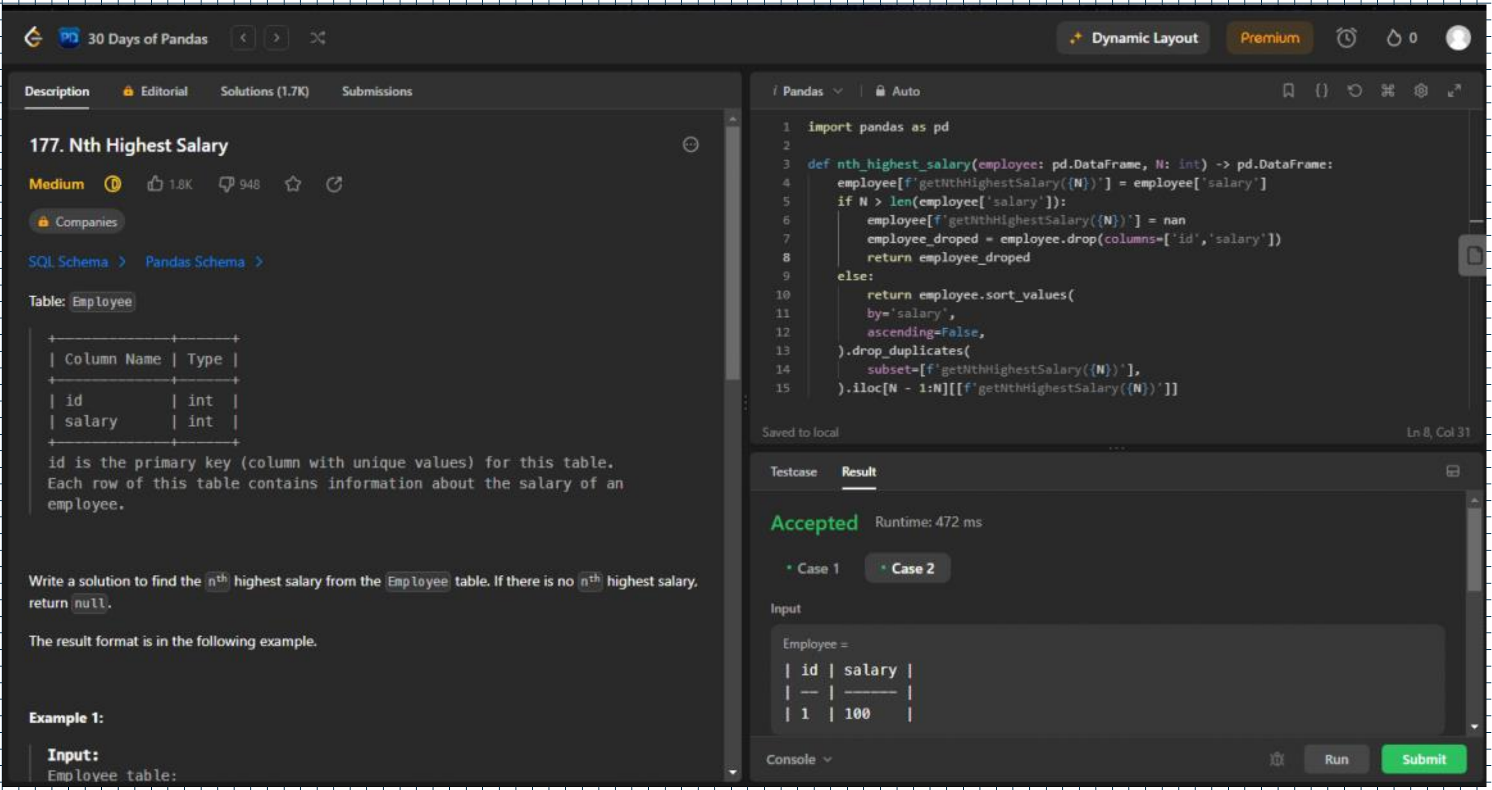
- 2205. The Number of Users That Are Eligible for Discount

Status	Language	Runtime	Memory	Notes
Accepted a few seconds ago	Pandas	511 ms	60.1 MB	
Wrong Answer a minute ago	Pandas	N/A	N/A	
Runtime Error a minute ago	Pandas	N/A	N/A	
Wrong Answer 3 minutes ago	Pandas	N/A	N/A	
Runtime Error 3 minutes ago	Pandas	N/A	N/A	
Wrong Answer 3 minutes ago	Pandas	N/A	N/A	
Wrong Answer				

```python
import pandas as pd

def nth_highest_salary(employee: pd.DataFrame, N: int) -> pd.DataFrame:
    unique_salaries = employee['salary'].drop_duplicates()
    employee[f'getNthHighestSalary({N})'] = employee['salary']
    if N > len(unique_salaries):
        employee[f'getNthHighestSalary({N})'] = nan
        employee_droped = employee.drop(columns=['id','salary'])
        return employee_droped.drop_duplicates()
    else:
        return employee.sort_values(
        by='salary',
        ascending=False,
    ).drop_duplicates(
        subset=[f'getNthHighestSalary({N})'],
    ).iloc[N - 1:N][[f'getNthHighestSalary({N})']]
```

Saved to local

Ln 6, Col 33



177. Nth Highest Salary

Medium 1.8K 948

Companies

SQL Schema > Pandas Schema >

Table: `Employee`

```
+-------------+------+
| 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 n^{th} highest salary from the `Employee` table. If there is no n^{th} highest salary, return `null`.

The result format is in the following example.

Example 1:

```
Input:
Employee table:
```

```python
import pandas as pd

def nth_highest_salary(employee: pd.DataFrame, N: int) -> pd.DataFrame:
    employee[f'getNthHighestSalary({N})'] = employee['salary']
    if N > len(employee['salary']):
        employee[f'getNthHighestSalary({N})'] = nan
        employee_droped = employee.drop(columns=['id','salary'])
        return employee_droped
    else:
        return employee.sort_values(
        by='salary',
        ascending=False,
    ).drop_duplicates(
        subset=[f'getNthHighestSalary({N})'],
    ).iloc[N - 1:N][[f'getNthHighestSalary({N})']]
```

Saved to local

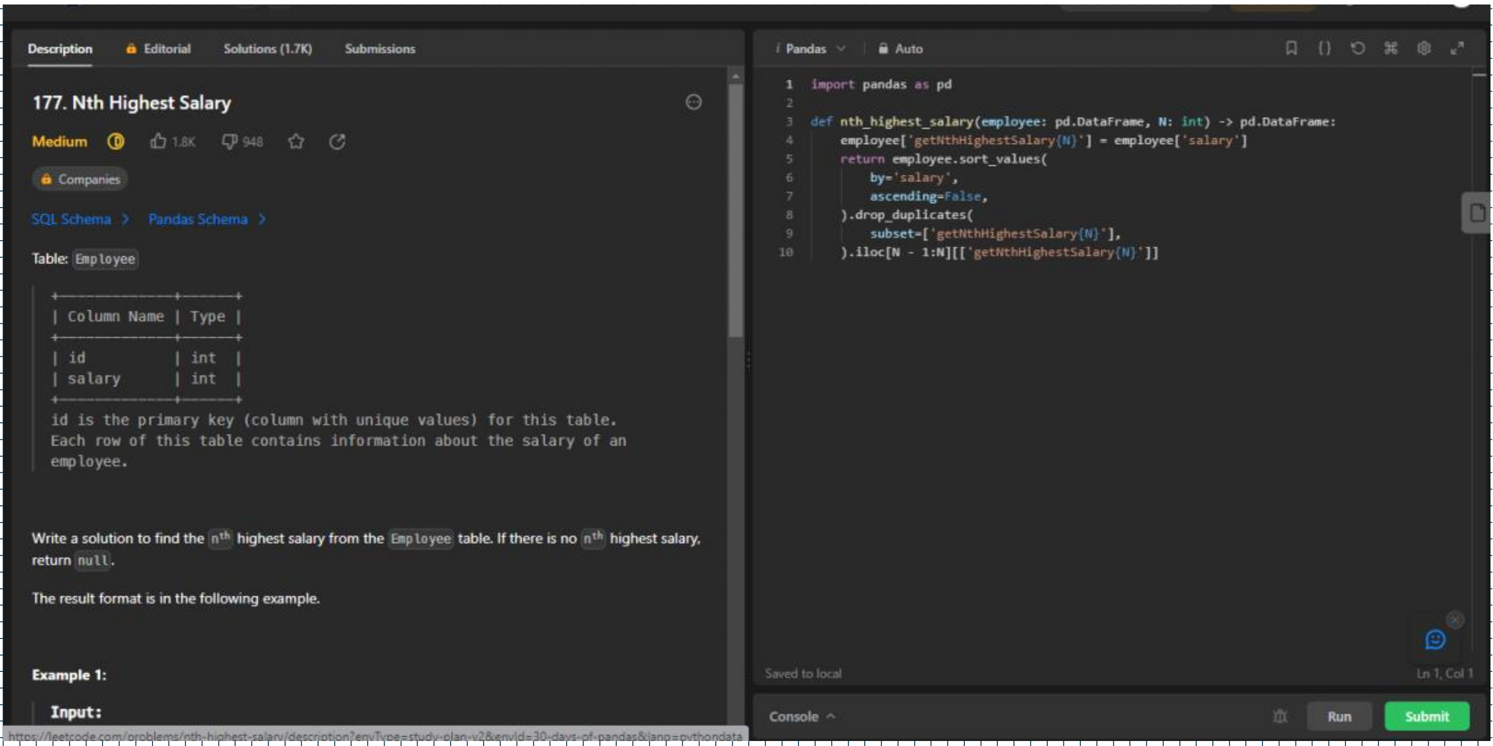
Ln 8, Col 31

Accepted Runtime: 472 ms

- Case 1
- Case 2

Input

```
Employee =
| id | salary |
| -- | ------ |
| 1  | 100    |
```



177. Nth Highest Salary

SQL Schema > Pandas Schema >

Table: `Employee`

```
+-------------+------+
| 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 `n`th highest salary from the `Employee` table. If there is no `n`th highest salary, return `null`.

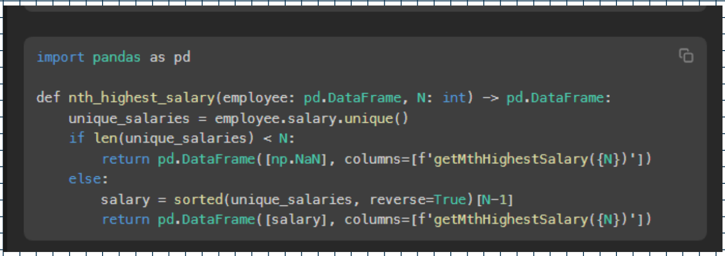
The result format is in the following example.

Example 1:

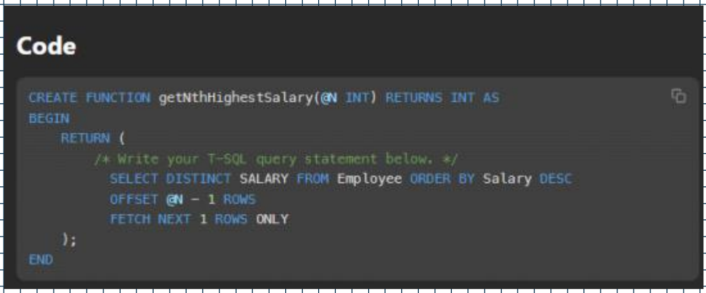
Input:

https://leetcode.com/problems/nth-highest-salary/description?envType=study-plan-v2&envId=30-days-of-pandas&lang=pythondata

```python
import pandas as pd

def nth_highest_salary(employee: pd.DataFrame, N: int) -> pd.DataFrame:
    employee['getNthHighestSalary(N)'] = employee['salary']
    return employee.sort_values(
        by='salary',
        ascending=False,
    ).drop_duplicates(
        subset=['getNthHighestSalary(N)'],
    ).iloc[N - 1:N][['getNthHighestSalary(N)']]
```

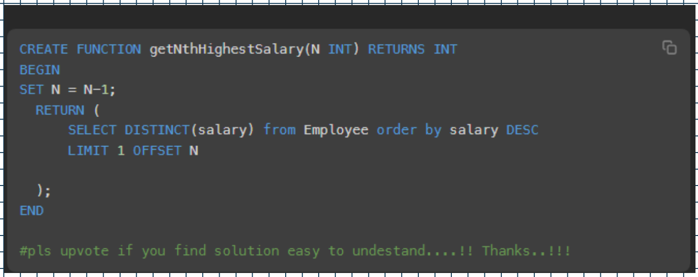


```python
import pandas as pd

def nth_highest_salary(employee: pd.DataFrame, N: int) -> pd.DataFrame:
    unique_salaries = employee.salary.unique()
    if len(unique_salaries) < N:
        return pd.DataFrame([np.NaN], columns=[f'getMthHighestSalary({N})'])
    else:
        salary = sorted(unique_salaries, reverse=True)[N-1]
        return pd.DataFrame([salary], columns=[f'getMthHighestSalary({N})'])
```

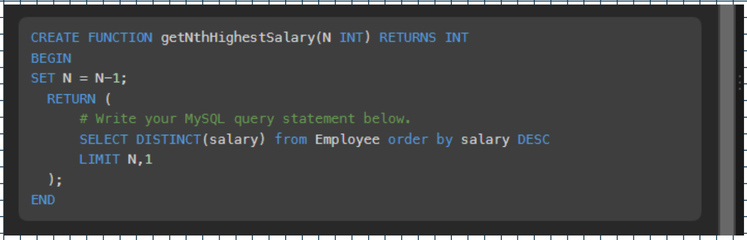


Code

```sql
CREATE FUNCTION getNthHighestSalary(@N INT) RETURNS INT AS
BEGIN
    RETURN (
        /* Write your T-SQL query statement below. */
          SELECT DISTINCT SALARY FROM Employee ORDER BY Salary DESC
          OFFSET @N - 1 ROWS
          FETCH NEXT 1 ROWS ONLY
    );
END
```



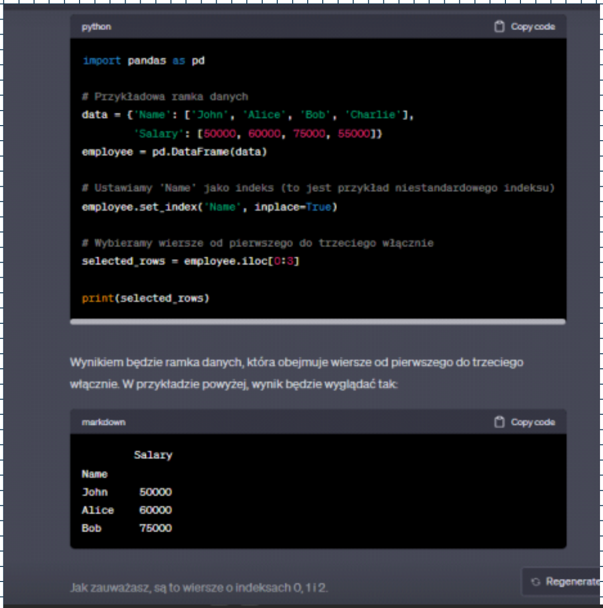
```sql
CREATE FUNCTION getNthHighestSalary(N INT) RETURNS INT
BEGIN
SET N = N-1;
  RETURN (
      SELECT DISTINCT(salary) from Employee order by salary DESC
      LIMIT 1 OFFSET N

  );
END

#pls upvote if you find solution easy to undestand....!! Thanks..!!!
```



```sql
CREATE FUNCTION getNthHighestSalary(N INT) RETURNS INT
BEGIN
SET N = N-1;
  RETURN (
      # Write your MySQL query statement below.
      SELECT DISTINCT(salary) from Employee order by salary DESC
      LIMIT N,1
  );
END
```

iloc

środa, 4 października 2023 13:28

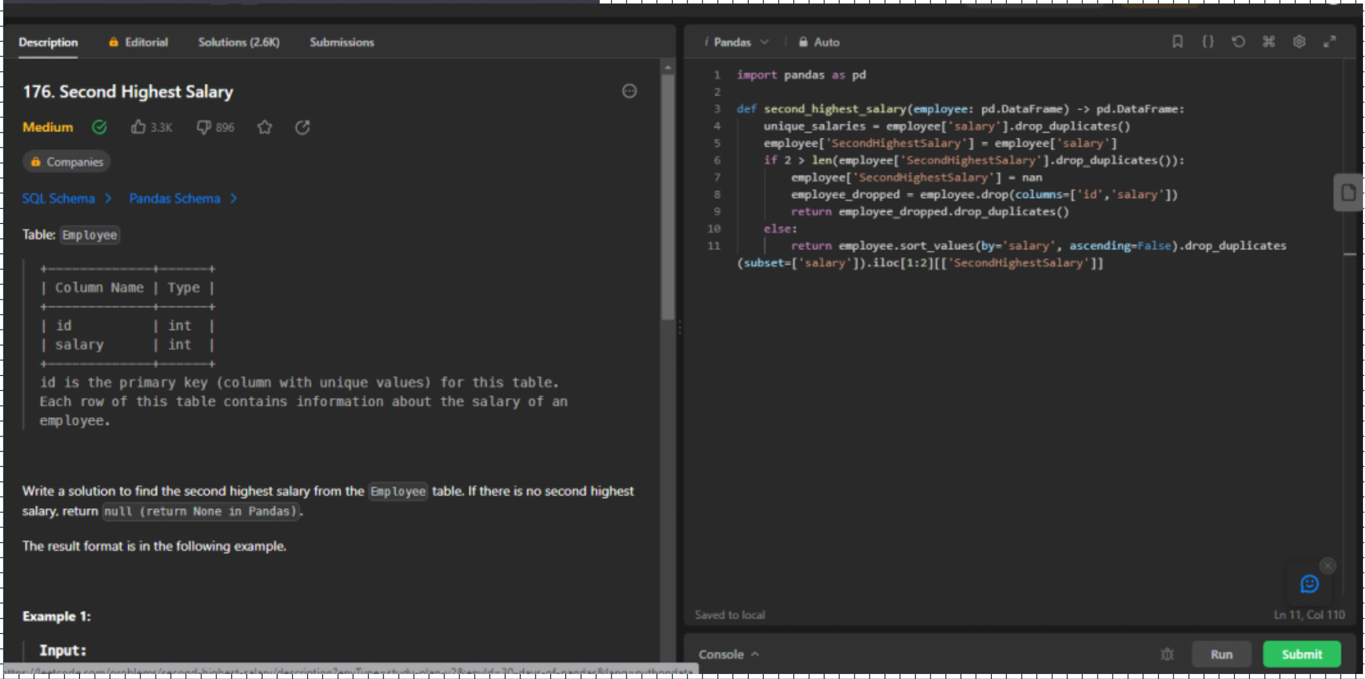


```python
import pandas as pd

# Przykładowa ramka danych
data = {'Name': ['John', 'Alice', 'Bob', 'Charlie'],
        'Salary': [50000, 60000, 75000, 55000]}
employee = pd.DataFrame(data)

# Ustawiamy 'Name' jako indeks (to jest przykład niestandardowego indeksu)
employee.set_index('Name', inplace=True)

# Wybieramy wiersze od pierwszego do trzeciego włącznie
selected_rows = employee.iloc[0:3]

print(selected_rows)
```

Wynikiem będzie ramka danych, która obejmuje wiersze od pierwszego do trzeciego włącznie. W przykładzie powyżej, wynik będzie wyglądać tak:

```
        Salary
Name
John     50000
Alice    60000
Bob      75000
```

Jak zauważasz, są to wiersze o indeksach 0, 1 i 2.



176. Second Highest Salary

Medium ✓ 3.3K 896

Companies

SQL Schema > Pandas Schema >

Table: `Employee`

```
+-------------+------+
| 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.
```

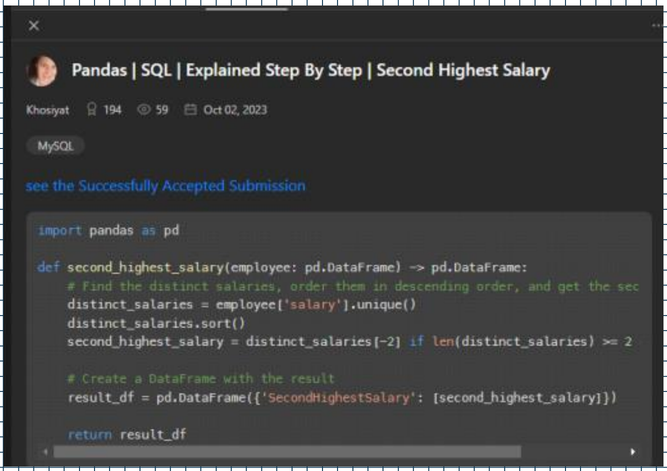
Write a solution to find the second highest salary from the `Employee` table. If there is no second highest salary, return `null (return None in Pandas)`.

The result format is in the following example.

Example 1:

Input:

```python
import pandas as pd

def second_highest_salary(employee: pd.DataFrame) -> pd.DataFrame:
    unique_salaries = employee['salary'].drop_duplicates()
    employee['SecondHighestSalary'] = employee['salary']
    if 2 > len(employee['SecondHighestSalary'].drop_duplicates()):
        employee['SecondHighestSalary'] = nan
        employee_dropped = employee.drop(columns=['id','salary'])
        return employee_dropped.drop_duplicates()
    else:
        return employee.sort_values(by='salary', ascending=False).drop_duplicates
(subset=['salary']).iloc[1:2][['SecondHighestSalary']]
```



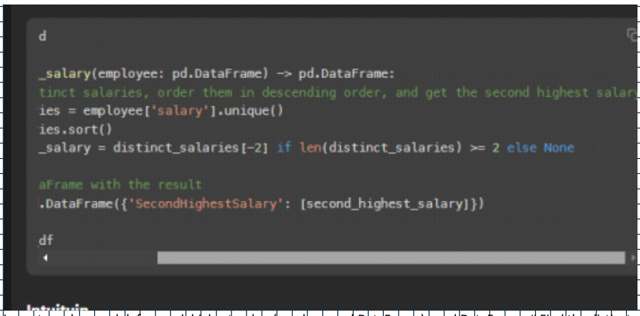
Pandas | SQL | Explained Step By Step | Second Highest Salary

Khosiyat 194 59 Oct 02, 2023

MySQL

see the Successfully Accepted Submission

```python
import pandas as pd

def second_highest_salary(employee: pd.DataFrame) -> pd.DataFrame:
    # Find the distinct salaries, order them in descending order, and get the sec
    distinct_salaries = employee['salary'].unique()
    distinct_salaries.sort()
    second_highest_salary = distinct_salaries[-2] if len(distinct_salaries) >= 2

    # Create a DataFrame with the result
    result_df = pd.DataFrame({'SecondHighestSalary': [second_highest_salary]})

    return result_df
```

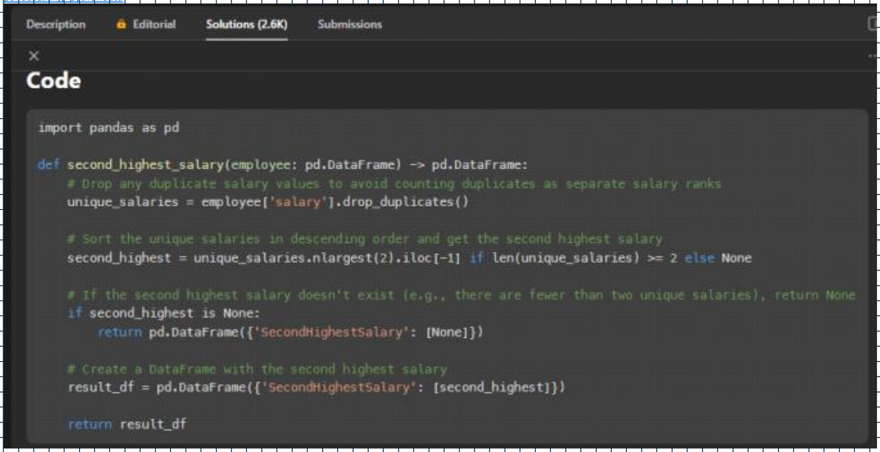


```
d

_salary(employee: pd.DataFrame) -> pd.DataFrame:
tinct salaries, order them in descending order, and get the second highest salar
ies = employee['salary'].unique()
ies.sort()
_salary = distinct_salaries[-2] if len(distinct_salaries) >= 2 else None

aFrame with the result
.DataFrame({'SecondHighestSalary': [second_highest_salary]})

df
```

Intuitoin

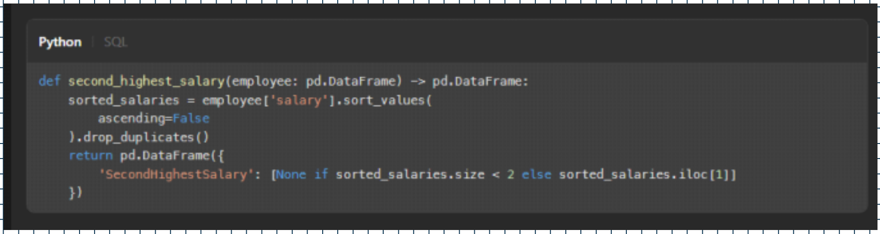
import pandas as pd def second_highest_salary(employee: pd.DataFrame) -> pd.DataFrame: # Find the distinct salaries, order them in descending order, and get the second highest salary distinct_salaries = employee['salary'].unique() distinct_salaries.sort() second_highest_salary = distinct_salaries[-2] if len(distinct_salaries) >= 2 else None # Create a DataFrame with the result result_df = pd.DataFrame({'SecondHighestSalary': [second_highest_salary]}) return result_df

Z <https://leetcode.com/problems/second-highest-salary/solutions/4120841/pandas-sql-explained-step-by-step-second-highest-salary/?envType=study-plan-v2&envId=30-days-of-pandas&lang=pythondata>



Code

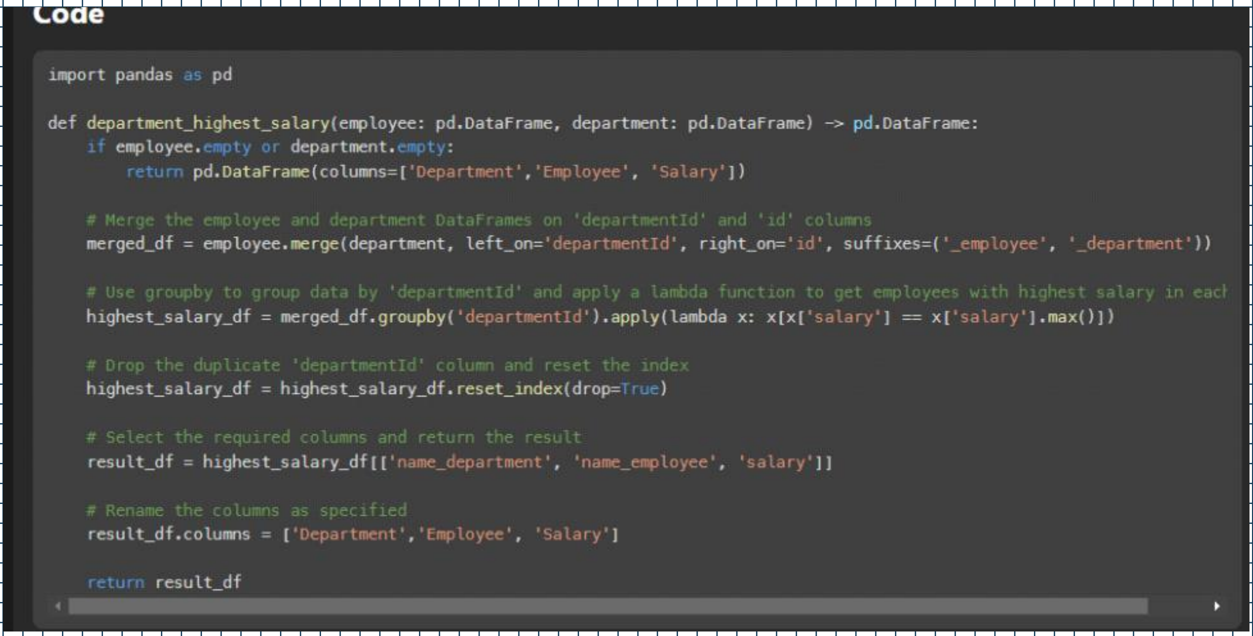
```
import pandas as pd

def second_highest_salary(employee: pd.DataFrame) -> pd.DataFrame:
    # Drop any duplicate salary values to avoid counting duplicates as separate salary ranks
    unique_salaries = employee['salary'].drop_duplicates()

    # Sort the unique salaries in descending order and get the second highest salary
    second_highest = unique_salaries.nlargest(2).iloc[-1] if len(unique_salaries) >= 2 else None

    # If the second highest salary doesn't exist (e.g., there are fewer than two unique salaries), return None
    if second_highest is None:
        return pd.DataFrame({'SecondHighestSalary': [None]})

    # Create a DataFrame with the second highest salary
    result_df = pd.DataFrame({'SecondHighestSalary': [second_highest]})

    return result_df
```



```
def second_highest_salary(employee: pd.DataFrame) -> pd.DataFrame:
    sorted_salaries = employee['salary'].sort_values(
        ascending=False
    ).drop_duplicates()
    return pd.DataFrame({
        'SecondHighestSalary': [None if sorted_salaries.size < 2 else sorted_salaries.iloc[1]]
    })
```

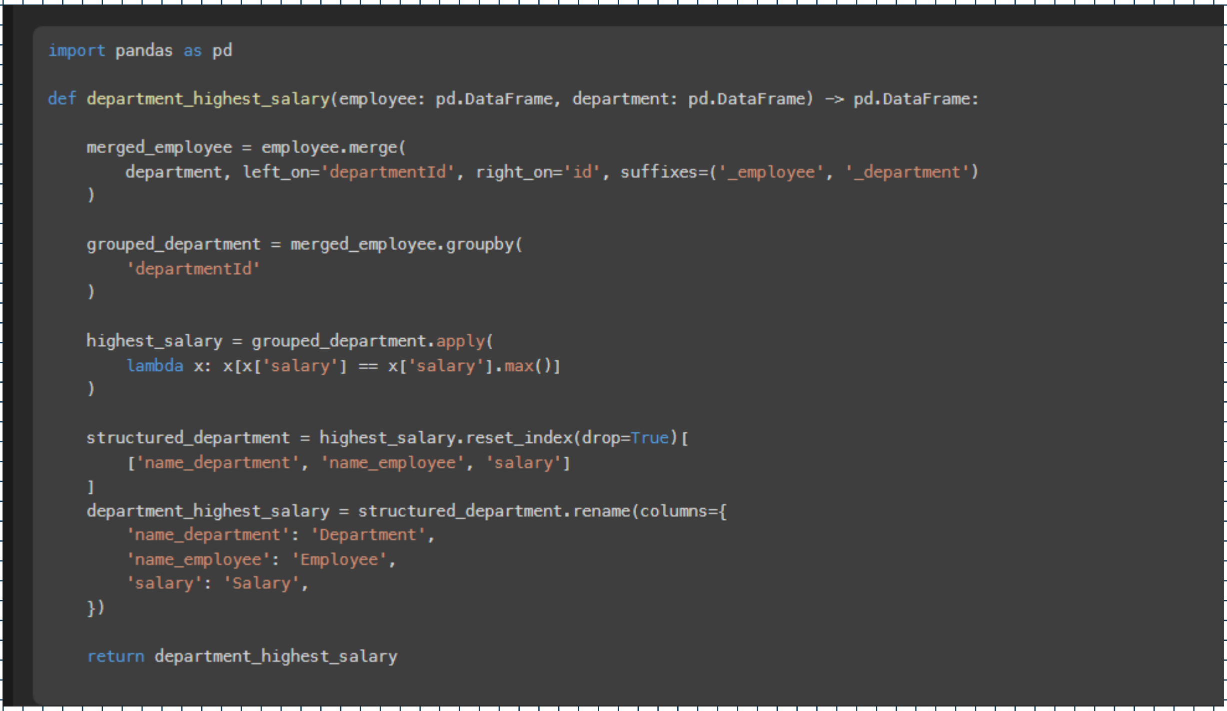
groupby transform(max) reset index lambda

środa, 4 października 2023 15:05

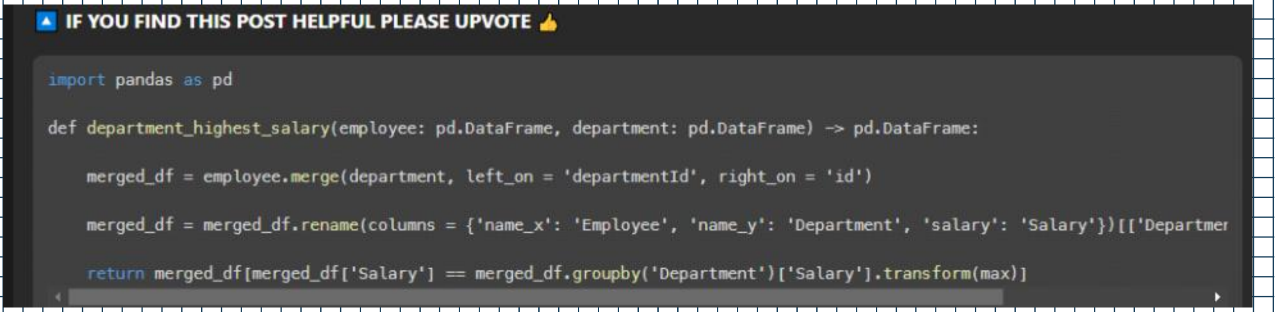


Code

```
import pandas as pd

def department_highest_salary(employee: pd.DataFrame, department: pd.DataFrame) -> pd.DataFrame:
    if employee.empty or department.empty:
        return pd.DataFrame(columns=['Department','Employee', 'Salary'])

    # Merge the employee and department DataFrames on 'departmentId' and 'id' columns
    merged_df = employee.merge(department, left_on='departmentId', right_on='id', suffixes=('_employee', '_department'))

    # Use groupby to group data by 'departmentId' and apply a lambda function to get employees with highest salary in each
    highest_salary_df = merged_df.groupby('departmentId').apply(lambda x: x[x['salary'] == x['salary'].max()])

    # Drop the duplicate 'departmentId' column and reset the index
    highest_salary_df = highest_salary_df.reset_index(drop=True)

    # Select the required columns and return the result
    result_df = highest_salary_df[['name_department', 'name_employee', 'salary']]

    # Rename the columns as specified
    result_df.columns = ['Department','Employee', 'Salary']

    return result_df
```

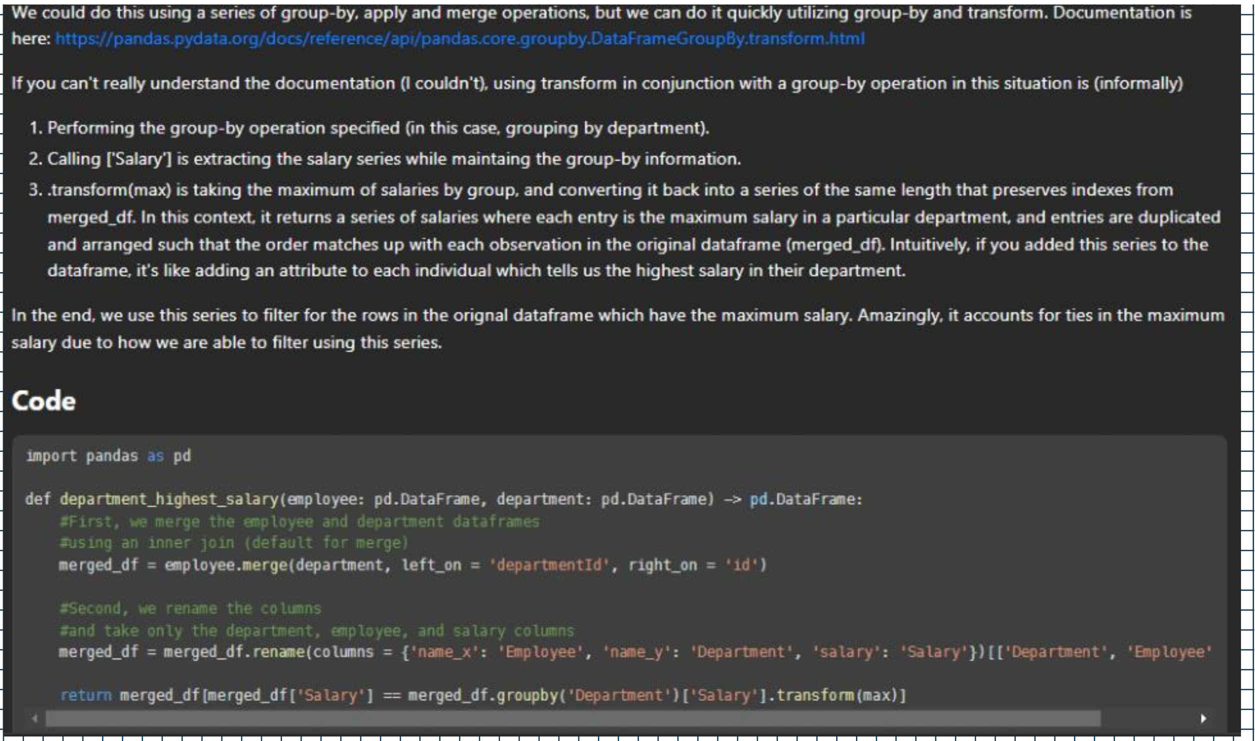


```
import pandas as pd

def department_highest_salary(employee: pd.DataFrame, department: pd.DataFrame) -> pd.DataFrame:

    merged_employee = employee.merge(
        department, left_on='departmentId', right_on='id', suffixes=('_employee', '_department')
    )

    grouped_department = merged_employee.groupby(
        'departmentId'
    )

    highest_salary = grouped_department.apply(
        lambda x: x[x['salary'] == x['salary'].max()]
    )

    structured_department = highest_salary.reset_index(drop=True)[
        ['name_department', 'name_employee', 'salary']
    ]
    department_highest_salary = structured_department.rename(columns={
        'name_department': 'Department',
        'name_employee': 'Employee',
        'salary': 'Salary',
    })

    return department_highest_salary
```



🔼 IF YOU FIND THIS POST HELPFUL PLEASE UPVOTE 👍

```
import pandas as pd

def department_highest_salary(employee: pd.DataFrame, department: pd.DataFrame) -> pd.DataFrame:

    merged_df = employee.merge(department, left_on = 'departmentId', right_on = 'id')

    merged_df = merged_df.rename(columns = {'name_x': 'Employee', 'name_y': 'Department', 'salary': 'Salary'})[['Departmen

    return merged_df[merged_df['Salary'] == merged_df.groupby('Department')['Salary'].transform(max)]
```



We could do this using a series of group-by, apply and merge operations, but we can do it quickly utilizing group-by and transform. Documentation is here: https://pandas.pydata.org/docs/reference/api/pandas.core.groupby.DataFrameGroupBy.transform.html

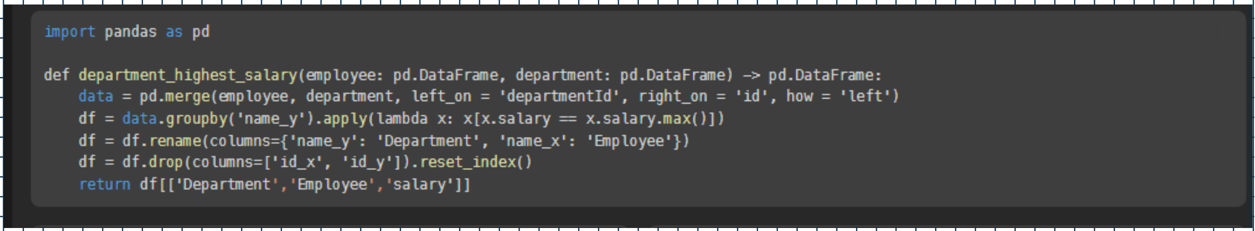
If you can't really understand the documentation (I couldn't), using transform in conjunction with a group-by operation in this situation is (informally)

1. Performing the group-by operation specified (in this case, grouping by department).
2. Calling ['Salary'] is extracting the salary series while maintaing the group-by information.
3. .transform(max) is taking the maximum of salaries by group, and converting it back into a series of the same length that preserves indexes from merged_df. In this context, it returns a series of salaries where each entry is the maximum salary in a particular department, and entries are duplicated and arranged such that the order matches up with each observation in the original dataframe (merged_df). Intuitively, if you added this series to the dataframe, it's like adding an attribute to each individual which tells us the highest salary in their department.

In the end, we use this series to filter for the rows in the orignal dataframe which have the maximum salary. Amazingly, it accounts for ties in the maximum salary due to how we are able to filter using this series.

Code

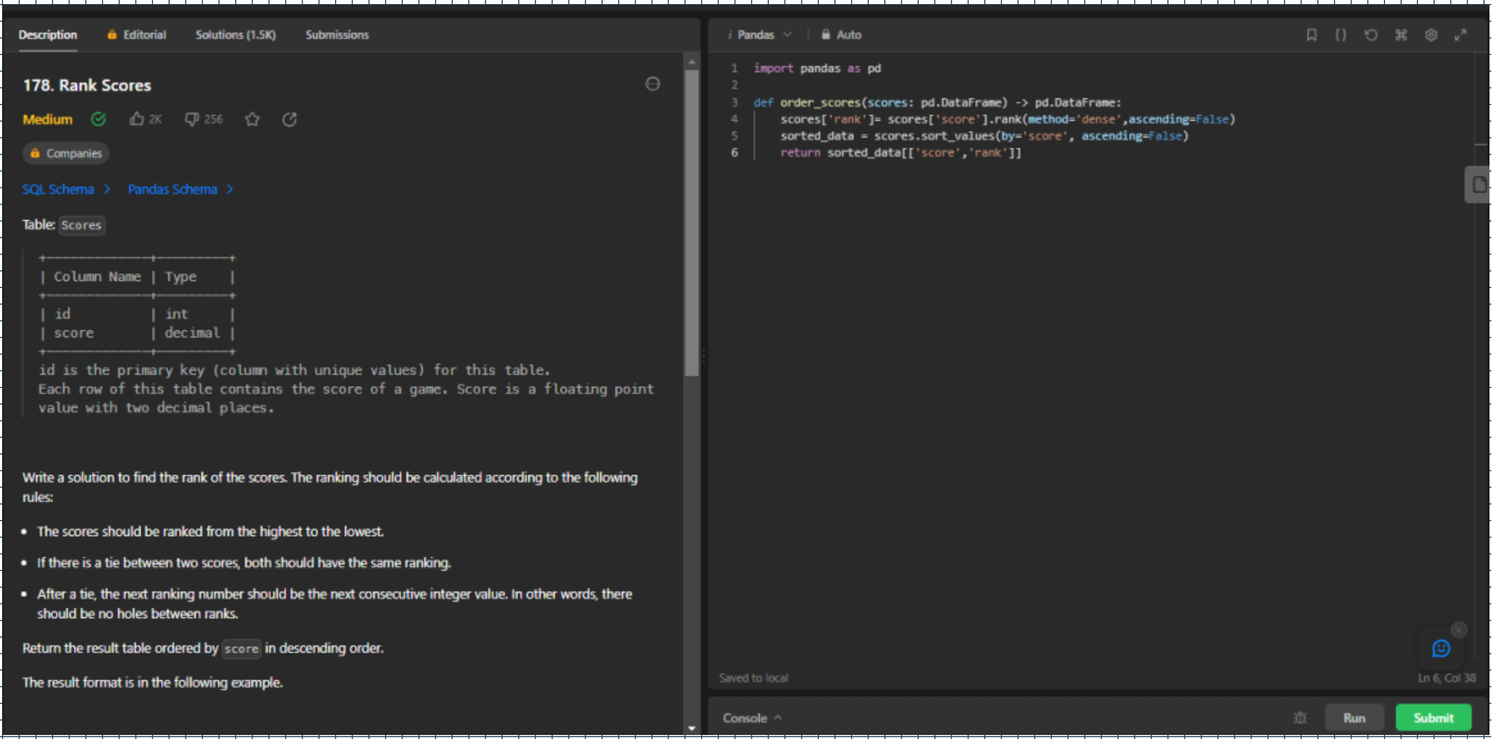
```
import pandas as pd

def department_highest_salary(employee: pd.DataFrame, department: pd.DataFrame) -> pd.DataFrame:
    #First, we merge the employee and department dataframes
    #using an inner join (default for merge)
    merged_df = employee.merge(department, left_on = 'departmentId', right_on = 'id')

    #Second, we rename the columns
    #and take only the department, employee, and salary columns
    merged_df = merged_df.rename(columns = {'name_x': 'Employee', 'name_y': 'Department', 'salary': 'Salary'})[['Department', 'Employee'

    return merged_df[merged_df['Salary'] == merged_df.groupby('Department')['Salary'].transform(max)]
```



```
import pandas as pd

def department_highest_salary(employee: pd.DataFrame, department: pd.DataFrame) -> pd.DataFrame:
    data = pd.merge(employee, department, left_on = 'departmentId', right_on = 'id', how = 'left')
    df = data.groupby('name_y').apply(lambda x: x[x.salary == x.salary.max()])
    df = df.rename(columns={'name_y': 'Department', 'name_x': 'Employee'})
    df = df.drop(columns=['id_x', 'id_y']).reset_index()
    return df[['Department','Employee','salary']]
```

Sort and rank

piątek, 6 października 2023 16:26



178. Rank Scores

Table: `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.

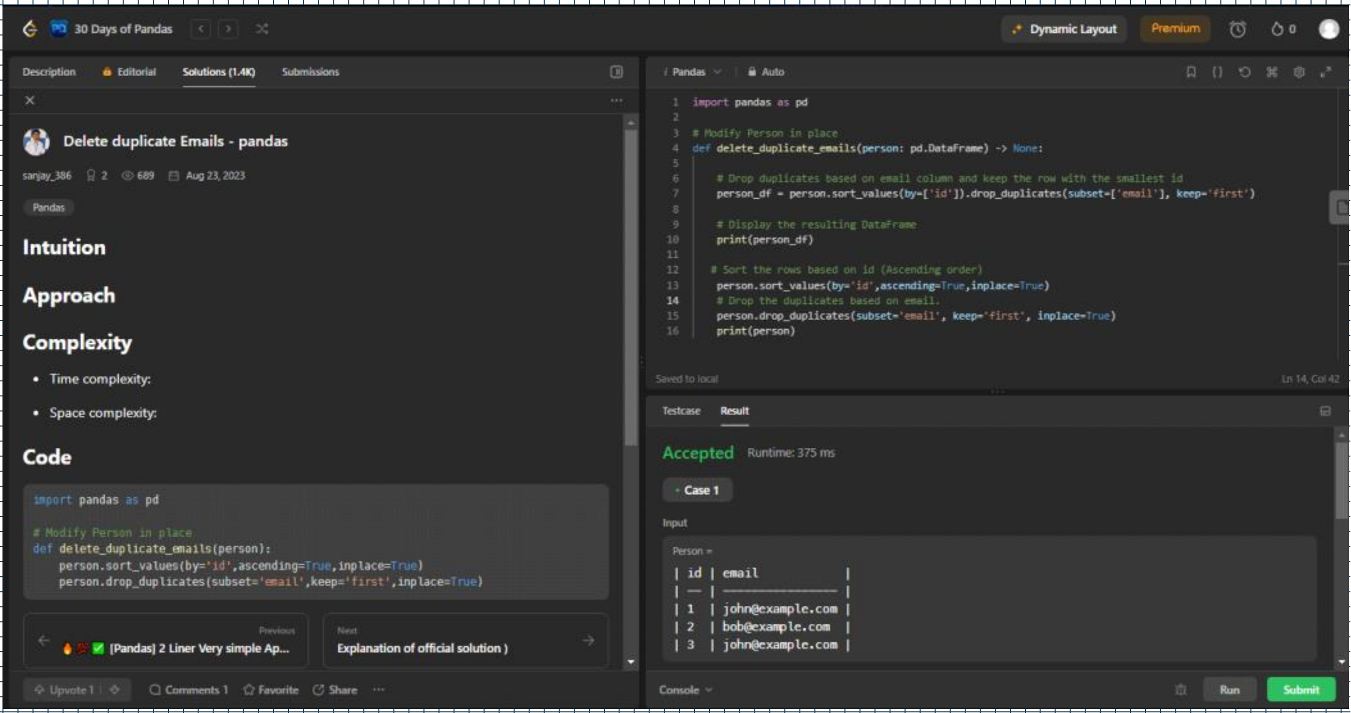
The result format is in the following example.

```python
import pandas as pd

def order_scores(scores: pd.DataFrame) -> pd.DataFrame:
    scores['rank']= scores['score'].rank(method='dense',ascending=False)
    sorted_data = scores.sort_values(by='score', ascending=False)
    return sorted_data[['score','rank']]
```

Inplace true drop mail duplicates

piątek, 6 października 2023 16:56

Inplace true replaces not creating new



Delete duplicate Emails - pandas

sanjay_386 2 689 Aug 23, 2023

Pandas

Intuition

Approach

Complexity

- Time complexity:
- Space complexity:

Code

```
import pandas as pd

# Modify Person in place
def delete_duplicate_emails(person):
    person.sort_values(by='id',ascending=True,inplace=True)
    person.drop_duplicates(subset='email',keep='first',inplace=True)
```

Previous: 🔥 💯 ✅ [Pandas] 2 Liner Very simple Ap...

Next: Explanation of official solution)

```
import pandas as pd

# Modify Person in place
def delete_duplicate_emails(person: pd.DataFrame) -> None:

    # Drop duplicates based on email column and keep the row with the smallest id
    person_df = person.sort_values(by=['id']).drop_duplicates(subset=['email'], keep='first')

    # Display the resulting DataFrame
    print(person_df)

   # Sort the rows based on id (Ascending order)
    person.sort_values(by='id',ascending=True,inplace=True)
    # Drop the duplicates based on email.
    person.drop_duplicates(subset='email', keep='first', inplace=True)
    print(person)
```

Saved to local

Ln 14, Col 42

Accepted Runtime: 375 ms

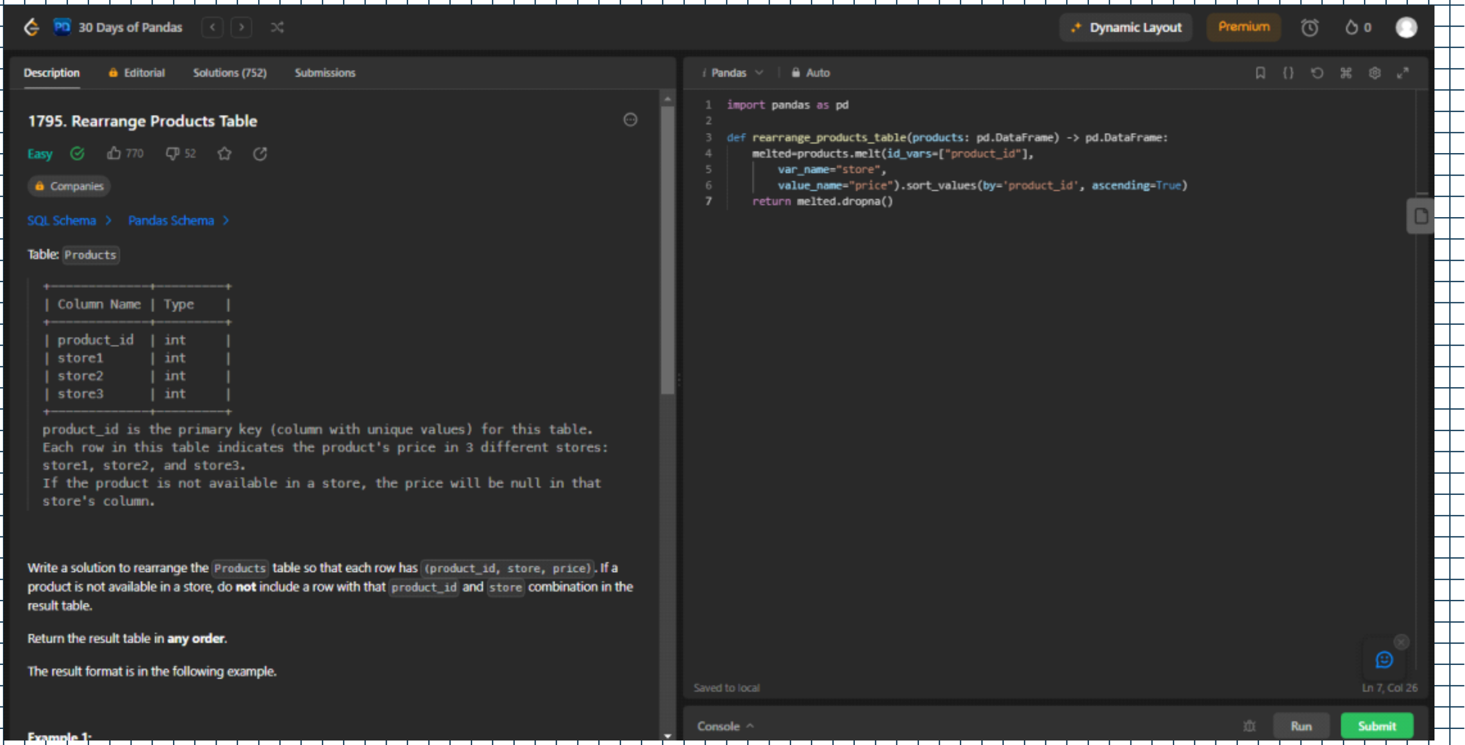
Case 1

Input

```
Person =
| id | email            |
| -- | ---------------- |
| 1  | john@example.com |
| 2  | bob@example.com  |
| 3  | john@example.com |
```

Columns to rows (melt)

piątek, 6 października 2023 17:57



1795. Rearrange Products Table

Table: `Products`

```
+-------------+---------+
| 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.
```

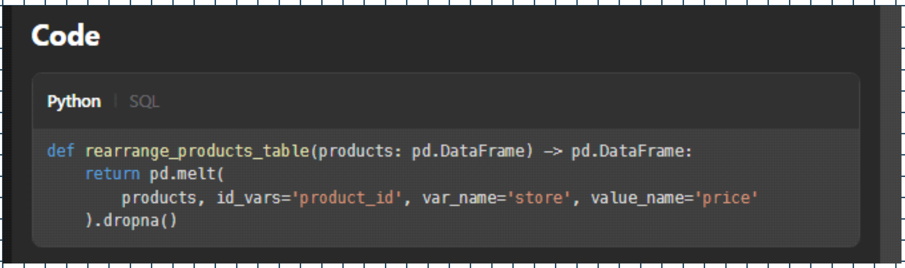
Write a solution to rearrange the `Products` table so that each row has `(product_id, store, price)`. If a product is not available in a store, do **not** include a row with that `product_id` and `store` combination in the result table.

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

The result format is in the following example.

Example 1:

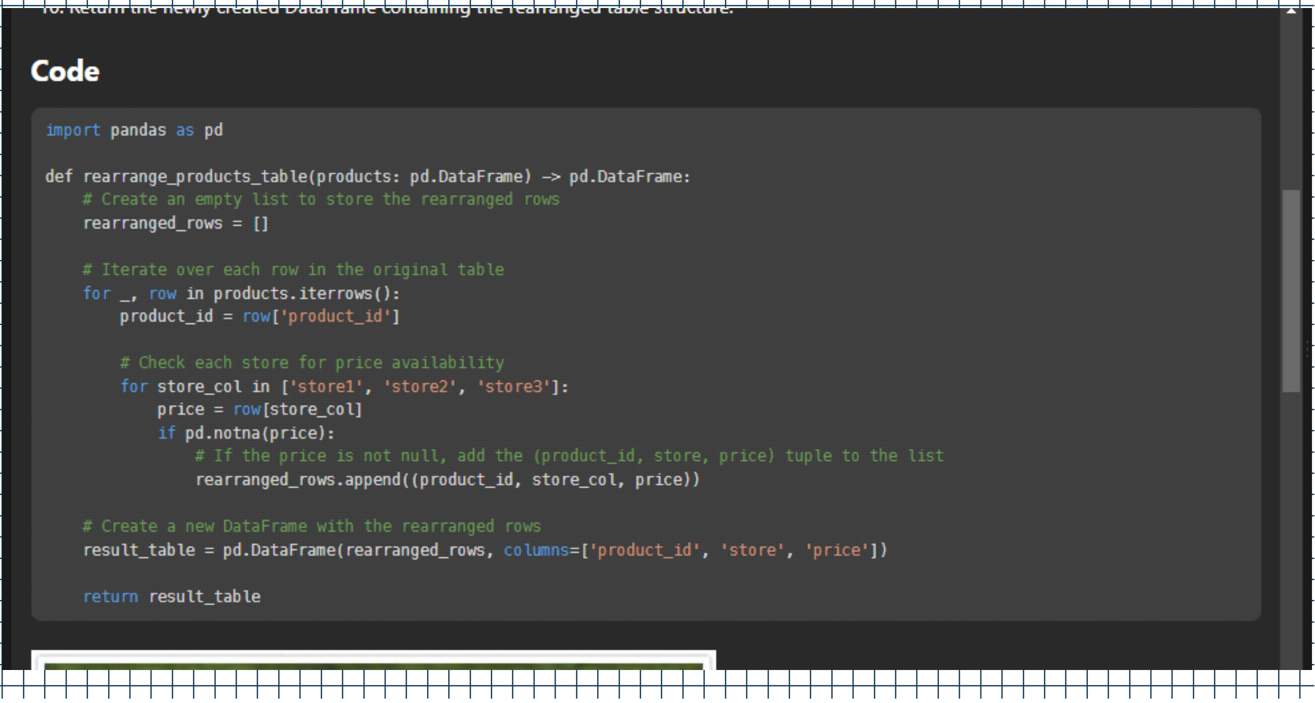
```python
import pandas as pd

def rearrange_products_table(products: pd.DataFrame) -> pd.DataFrame:
    melted=products.melt(id_vars=["product_id"],
        var_name="store",
        value_name="price").sort_values(by='product_id', ascending=True)
    return melted.dropna()
```



Code

Python | SQL

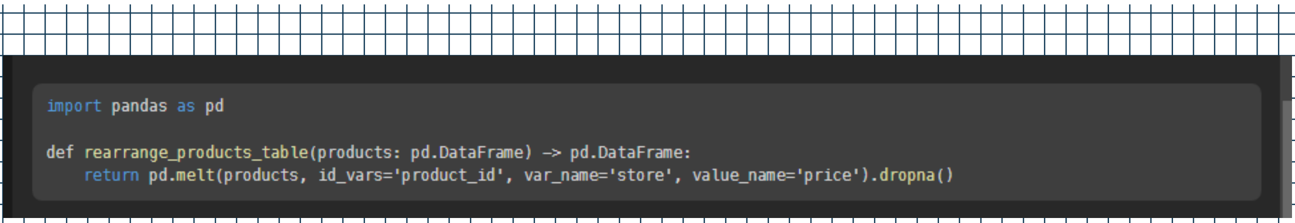
```python
def rearrange_products_table(products: pd.DataFrame) -> pd.DataFrame:
    return pd.melt(
        products, id_vars='product_id', var_name='store', value_name='price'
    ).dropna()
```



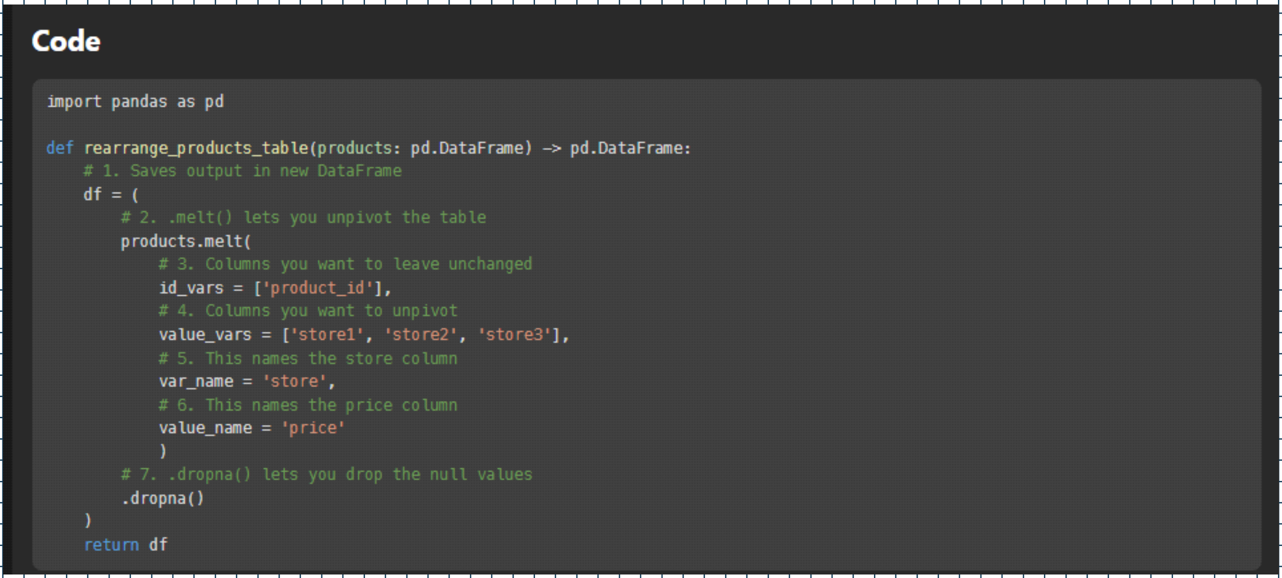
10. Return the newly created DataFrame containing the rearranged table structure.

Code

```python
import pandas as pd

def rearrange_products_table(products: pd.DataFrame) -> pd.DataFrame:
    # Create an empty list to store the rearranged rows
    rearranged_rows = []

    # Iterate over each row in the original table
    for _, row in products.iterrows():
        product_id = row['product_id']

        # Check each store for price availability
        for store_col in ['store1', 'store2', 'store3']:
            price = row[store_col]
            if pd.notna(price):
                # If the price is not null, add the (product_id, store, price) tuple to the list
                rearranged_rows.append((product_id, store_col, price))

    # Create a new DataFrame with the rearranged rows
    result_table = pd.DataFrame(rearranged_rows, columns=['product_id', 'store', 'price'])

    return result_table
```



```python
import pandas as pd

def rearrange_products_table(products: pd.DataFrame) -> pd.DataFrame:
    return pd.melt(products, id_vars='product_id', var_name='store', value_name='price').dropna()
```



Code

```python
import pandas as pd

def rearrange_products_table(products: pd.DataFrame) -> pd.DataFrame:
    # 1. Saves output in new DataFrame
    df = (
        # 2. .melt() lets you unpivot the table
        products.melt(
            # 3. Columns you want to leave unchanged
            id_vars = ['product_id'],
            # 4. Columns you want to unpivot
            value_vars = ['store1', 'store2', 'store3'],
            # 5. This names the store column
            var_name = 'store',
            # 6. This names the price column
            value_name = 'price'
            )
        # 7. .dropna() lets you drop the null values
        .dropna()
    )
    return df
```