INTRODUCTION TO PANDAS

2877.

Write a solution to **create** a DataFrame from a 2D list called student_data. This 2D list contains the IDs and ages of some students.

The DataFrame should have two columns, student_id and age, and be in the same order as the original 2D list.

The result format is in the following example.

Example 1:	Output	::
Input:	+	+
student_data:	stude	ent_id age
]	+	+
[1, 15],	1	15
[2, 11],	2	11
[3, 11],	3	11
[4, 20]	4	20
]	+	+

Explanation:

A DataFrame was created on top of student_data, with two columns named student_id and age.

Solution -

import pandas as pd

def createDataframe(student_data: List[List[int]]) -> pd.DataFrame:
 return pd.DataFrame(student_data, columns=['student_id', 'age'])

2878.

DataFrame players:
+----+

| Column Name | Type |
+----+

| player_id | int |
| name | object |

| int |

age

```
| position | object |
|... | ... |
+----+
Write a solution to calculate and display the number of rows and columns of players.
Return the result as an array:
[number of rows, number of columns]
The result format is in the following example.
Example 1:
Input:
+-----+
| player_id | name | age | position | team
+-----+
      | Mason | 21 | Forward | RealMadrid
846
       | Riley | 30 | Winger | Barcelona |
| 749
       | Bob | 28 | Striker | ManchesterUnited |
| 155
       | Isabella | 32 | Goalkeeper | Liverpool |
| 583
| 388
       | Zachary | 24 | Midfielder | BayernMunich |
       | Ava | 23 | Defender | Chelsea |
| 883
       | Violet | 18 | Striker | Juventus
| 355
       | Thomas | 27 | Striker | ParisSaint-Germain |
| 247
       | Jack | 33 | Midfielder | ManchesterCity |
| 761
642
       | Charlie | 36 | Center-back | Arsenal |
Output:
[10, 5]
Explanation:
This DataFrame contains 10 rows and 5 columns.
Solution -
import pandas as pd
                    def getDataframeSize(players: pd.DataFrame) -> List[int]:
                      [num1, num2] = players.shape
```

return [num1, num2]

```
2879.
DataFrame: employees
+----+
| Column Name | Type |
+----+
| employee_id | int |
| name | object |
| department | object |
| salary | int |
+----+
Write a solution to display the first 3 rows of this DataFrame.
Example 1:
Input:
DataFrame employees
+-----+
| employee_id | name | department | salary |
+-----+
    | Bob | Operations | 48675 |
| 3
     | Alice | Sales | 11096 |
90
      | Tatiana | Engineering | 33805 |
| 9
      | Annabelle | InformationTechnology | 37678 |
| 60
      | Jonathan | HumanResources | 23793 |
| 49
      | Khaled | Administration | 40454 |
| 43
Output:
+----+
| employee_id | name | department | salary |
+----+
| 3 | Bob | Operations | 48675 |
| 9
     | Tatiana | Engineering | 33805 |
+----+
Explanation:
Only the first 3 rows are displayed.
```

Solution import pandas as pd def selectFirstRows(employees: pd.DataFrame) -> pd.DataFrame: return employees.head(3) 2880. DataFrame students | age | int | +----+ +----+ | Column Name | Type | +----+ Write a solution to select the name and age of the student with student_id = 101. | student_id | int | The result format is in the following example. | name | object | Example 1: +----+ Input: +----+ Output: | student_id | name | age | +----+ +----+ | name | age | | 101 | Ulysses | 13 | +----+ | 53 | William | 10 | | Ulysses | 13 | | 128 | Henry | 6 | +----+ | Henry | 11 | | 3 **Explanation:** Student Ulysses has student_id = 101, we select the name and age. Solution import pandas as pd def selectData(students: pd.DataFrame) -> pd.DataFrame: return students.loc[students['student_id'] == 101, 'name':]

2881.

DataFrame employees
+-----+
| Column Name | Type. |
+----+
| name | object |
| salary | int. |
+-----+

A company plans to provide its employees with a bonus.

Write a solution to create a new column name bonus that contains the **doubled values** of the salary column.

The result format is in the following example.

Example 1:

Input:

DataFrame employees	Output:
++	++
name salary	name salary bonus
++	++
Piper 4548	Piper 4548 9096
Grace 28150	Grace 28150 56300
Georgia 1103	Georgia 1103 2206
Willow 6593	Willow 6593 13186
Finn 74576	Finn 74576 149152
Thomas 24433	Thomas 24433 48866
++	++

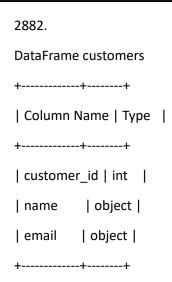
Explanation:

A new column bonus is created by doubling the value in the column salary.

Solution -

import pandas as pd

def createBonusColumn(employees: pd.DataFrame) -> pd.DataFrame:
 employees['bonus'] = employees['salary'] * 2
 return employees



There are some duplicate rows in the DataFrame based on the email column.

Write a solution to remove these duplicate rows and keep only the **first** occurrence.

The result format is in the following example.

Example 1:

Input:		Output:		
++ customer_id name email		+	++	
		customer_id name email		
+	+	+	++	
1	Ella emily@example.com	1	Ella emily@example.com	
2	David michael@example.com	2	David michael@example.com	
3	Zachary sarah@example.com	3	Zachary sarah@example.com	
4	Alice john@example.com	4	Alice john@example.com	
5	Finn john@example.com	6	Violet alice@example.com	
6	Violet alice@example.com	+	++	
4				

Explanation:

Alic (customer_id = 4) and Finn (customer_id = 5) both use john@example.com, so only the first occurrence of this email is retained.

Solution -

import pandas as pd

def dropDuplicateEmails(customers: pd.DataFrame) -> pd.DataFrame:

return customers.drop_duplicates(subset = ['email'], keep = 'first')

2883. DataFrame students +----+ | Column Name | Type | +----+ | student_id | int |

| name | object |

| age | int |

+----+

There are some rows having missing values in the name column.

Write a solution to remove the rows with missing values.

The result format is in the following example.

Example 1:

Output:

+----+
| student_id | name | age |
+----+
32	Piper	5
779	Georgia	20
849	Willow	14
+-----+

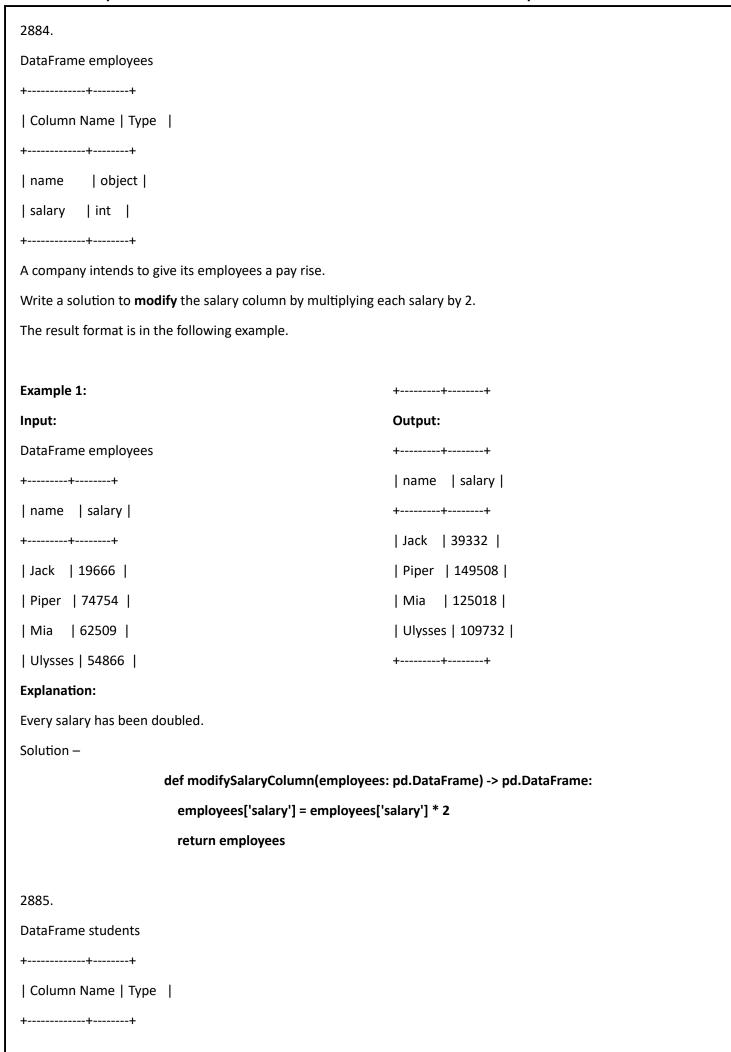
Explanation:

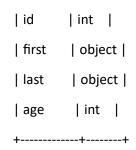
Student with id 217 havs empty value in the name column, so it will be removed.

Solution -

import pandas as pd

def dropMissingData(students: pd.DataFrame) -> pd.DataFrame:
 return students.dropna(subset='name')





Write a solution to rename the columns as follows:

- id to student_id
- first to first_name
- last to last_name
- age to age_in_years

The result format is in the following example.

Example 1:

Input:

++
id first last age
++
1 Mason King 6
2 Ava Wright 7
3 Taylor Hall 16
4 Georgia Thompson 18
5 Thomas Moore 10
++

Output:

Explanation:

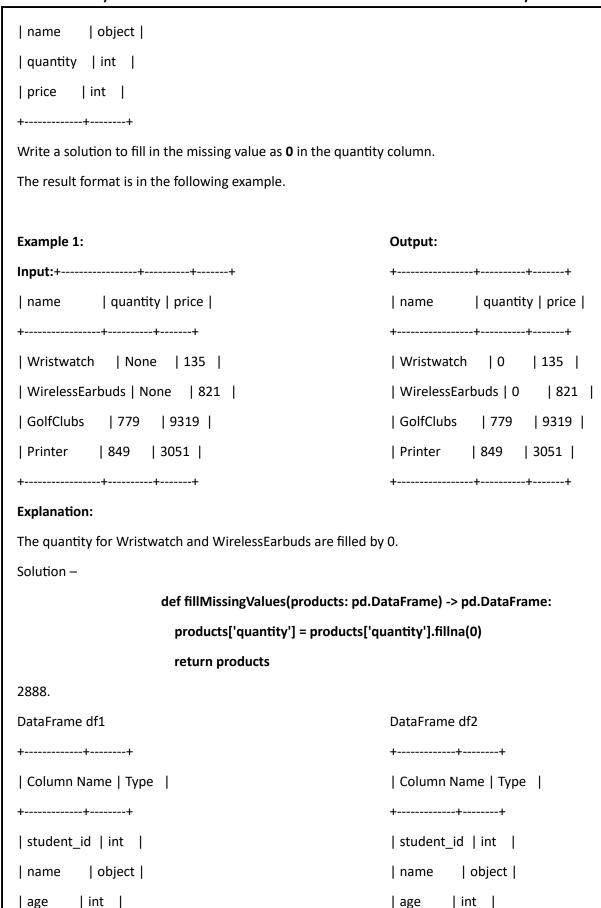
The column names are changed accordingly.

Solution -

def renameColumns(students: pd.DataFrame) -> pd.DataFrame:

return students.rename(columns={'id':'student_id', 'first':'first_name', 'last':'last_name', 'age':'age_in_years'})

2886. DataFrame students +----+ | Column Name | Type | +----+ | student_id | int | | name | object | | age | int | grade | float | +----+ Write a solution to correct the errors: The grade column is stored as floats, convert it to integers. The result format is in the following example. Example 1: Input: Output: DataFrame students: +----+ +----+ | student_id | name | age | grade | +----+ | student_id | name | age | grade | | 1 | Ava | 6 | 73 | +----+ | 1 | Ava | 6 | 73.0 | | 2 | Kate | 15 | 87 | | 2 | Kate | 15 | 87.0 | +----+ +----+ **Explanation:** The data types of the column grade is converted to int. Solution def changeDatatype(students: pd.DataFrame) -> pd.DataFrame: students['grade'] = students['grade'].astype(int) return students 2887. DataFrame products +----+ | Column Name | Type | +----+

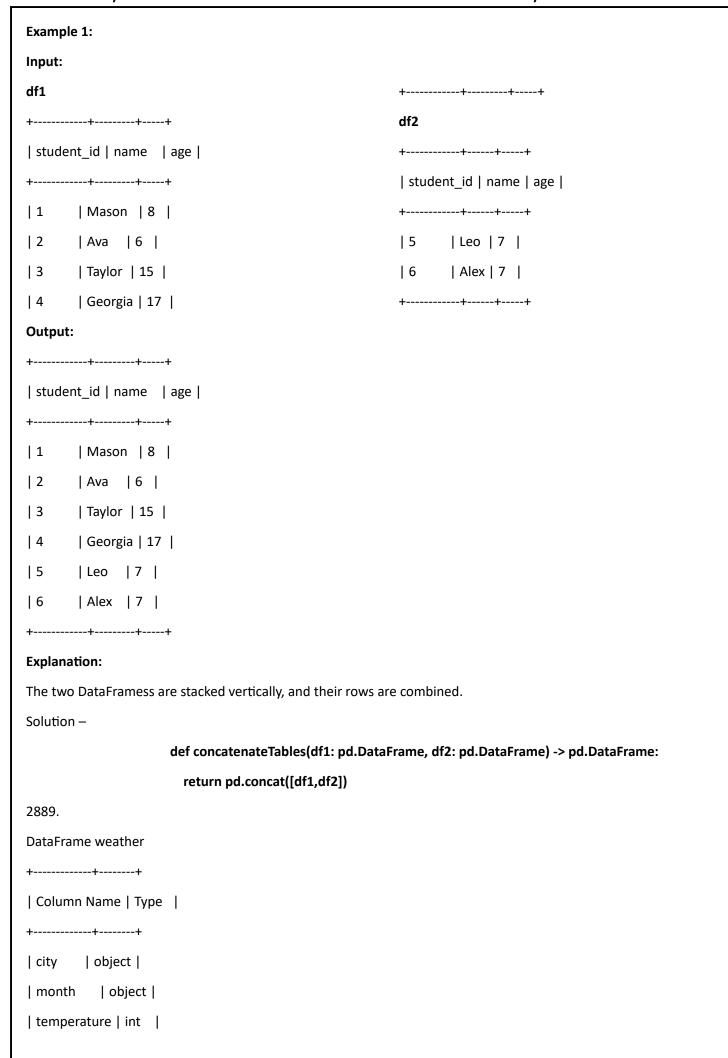


Write a solution to concatenate these two DataFrames vertically into one DataFrame.

+----+

The result format is in the following example.

+----+





Write a solution to **pivot** the data so that each row represents temperatures for a specific month, and each city is a separate column.

The result format is in the following example.

Example 1:

Input:	ElPaso May 43
++	++
city month temperature	Output:
++	++
Jacksonville January 13	month ElPaso Jacksonville
Jacksonville February 23	++
Jacksonville March 38	April 2 5
Jacksonville April 5	February 6 23
Jacksonville May 34	January 20 13
ElPaso January 20	March 26 38
ElPaso February 6	May 43 34
ElPaso March 26	++
ElPaso April 2	

Explanation:

The table is pivoted, each column represents a city, and each row represents a specific month.

Solution -

def pivotTable(weather: pd.DataFrame) -> pd.DataFrame:

return weather.pivot(index='month', columns='city', values='temperature')

2890.

DataFrame report

+----+

| Column Name | Type |

+----+

| product | object |

| quarter_1 | int |

| quarter_2 | int |

| quarter_3 | int |

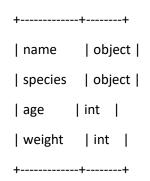
| quarter_4 | int |

+----+

+----+

| Column Name | Type |

Write a solution to **reshape** the data so that each row represents sales data for a product in a specific quarter. The result format is in the following example. Example 1: Input: +-----+ | product | quarter_1 | quarter_2 | quarter_3 | quarter_4 | +-----+ | Umbrella | 417 | 224 | 379 | 611 | | SleepingBag | 800 | 936 | 93 | 875 | +-----+ Output: +----+ | product | quarter | sales | +----+ | Umbrella | quarter_1 | 417 | | SleepingBag | quarter_1 | 800 | | Umbrella | quarter_2 | 224 | | SleepingBag | quarter_2 | 936 | | Umbrella | quarter_3 | 379 | | SleepingBag | quarter_3 | 93 | | Umbrella | quarter_4 | 611 | | SleepingBag | quarter_4 | 875 | +----+ **Explanation:** The DataFrame is reshaped from wide to long format. Each row represents the sales of a product in a quarter. Soltion def meltTable(report: pd.DataFrame) -> pd.DataFrame: return pd.melt(report, id_vars=['product'], var_name = 'quarter', value_name = 'sales') 2891. DataFrame animals



Write a solution to list the names of animals that weigh **strictly more than** 100 kilograms.

Return the animals sorted by weight in **descending order**.

The result format is in the following example.

Example 1:

Input:	++
DataFrame animals:	Output:
++	++
name species age weight	name
++	++
Tatiana Snake 98 464	Tatiana
Khaled Giraffe 50 41	Jonathan
Alex Leopard 6 328	Tommy
Jonathan Monkey 45 463	Alex
Stefan Bear 100 50	++
Tommy Panda 26 349	

Explanation:

All animals weighing more than 100 should be included in the results table.

Tatiana's weight is 464, Jonathan's weight is 463, Tommy's weight is 349, and Alex's weight is 328.

The results should be sorted in descending order of weight.

In Pandas, **method chaining** enables us to perform operations on a DataFrame without breaking up each operation into a separate line or creating multiple temporary variables.

Can you complete this task in just one line of code using method chaining?

Solution -

def findHeavyAnimals(animals: pd.DataFrame) -> pd.DataFrame:

return animals[animals['weight']>100].sort_values(['weight'], ascending=False)[['name']]