DESCRIPTION

Table: Prices

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

| Column Name | Type |

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

| product_id | int |

| start_date | date |

| end_date | date |

| price | int |

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

(product_id, start_date, end_date) is the primary key (combination of columns with unique values) for this table.

Each row of this table indicates the price of the product_id in the period from start_date to end_date.

For each product_id there will be no two overlapping periods. That means there will be no two intersecting periods for the same product_id.

Table: UnitsSold

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

| Column Name | Type |

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

| product_id | int |

| purchase_date | date |

| units | int |

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

This table may contain duplicate rows.

Each row of this table indicates the date, units, and product_id of each product sold.

Write a solution to find the average selling price for each product. average_price should be **rounded to 2 decimal places**. If a product does not have any sold units, its average selling price is assumed to be 0.

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

The result format is in the following example.

Example 1:

Input:

Prices table:

```
+------------+------------+------------+--------+
| product_id | start_date | end_date   | price  |
+------------+------------+------------+--------+
| 1          | 2019-02-17 | 2019-02-28 | 5      |
| 1          | 2019-03-01 | 2019-03-22 | 20     |
| 2          | 2019-02-01 | 2019-02-20 | 15     |
| 2          | 2019-02-21 | 2019-03-31 | 30     |
+------------+------------+------------+--------+
```

UnitsSold table:

```
+------------+---------------+-------+
| product_id | purchase_date | units |
+------------+---------------+-------+
| 1          | 2019-02-25    | 100   |
| 1          | 2019-03-01    | 15    |
| 2          | 2019-02-10    | 200   |
| 2          | 2019-03-22    | 30    |
+------------+---------------+-------+
```

Output:

```
+------------+---------------+
| product_id | average_price |
+------------+---------------+
| 1          | 6.96          |
| 2          | 16.96         |
+------------+---------------+
```

Explanation:

Average selling price = Total Price of Product / Number of products sold.

Average selling price for product 1 = ((100 * 5) + (15 * 20)) / 115 = 6.96

Average selling price for product 2 = ((200 * 15) + (30 * 30)) / 230 = 16.96

SOLUTION

Option 1:

- Join dataframes
- Filter the dataframe for 'purchase_date'
- Compute price x units and total units
- Handle division by zero using np.where() and compute 'average_price'

```
import pandas as pd

def average_selling_price(prices: pd.DataFrame, units_sold: pd.DataFrame) -> pd.DataFrame:
    df = prices.merge(units_sold, how = 'left', on = 'product_id')
    df = df[((df['purchase_date']>= df['start_date']) & (df['purchase_date'] <= df['end_date'])) |
df['purchase_date'].isna()]
    dfp = df.groupby('product_id').apply(lambda x: (x['price'] *
x['units']).sum()).reset_index(name='price_x_units')
    dfu = df.groupby('product_id').apply(lambda x: x['units'].sum()).reset_index(name= 'units')
    result =dfp.merge(dfu, how = 'inner', on = 'product_id')
    result['average_price'] = np.where((result['units'] != 0) ,
result['price_x_units'].div(result['units'], fill_value=0).round(2), 0)
    return result[['product_id', 'average_price']]
```

- Snapshot of the same code above for readability purposes

```
import pandas as pd

def average_selling_price(prices: pd.DataFrame, units_sold: pd.DataFrame) -> pd.DataFrame:
    df = prices.merge(units_sold, how = 'left', on = 'product_id')
    df = df[((df['purchase_date']>= df['start_date']) & (df['purchase_date'] <= df['end_date'])) | df
['purchase_date'].isna()]
    dfp = df.groupby('product_id').apply(lambda x: (x['price'] * x['units']).sum()).reset_index
(name='price_x_units')
    dfu = df.groupby('product_id').apply(lambda x: x['units'].sum()).reset_index(name= 'units')
    result =dfp.merge(dfu, how = 'inner', on = 'product_id')
    result['average_price'] = np.where((result['units'] != 0) , result['price_x_units'].div(result
['units'], fill_value=0).round(2), 0)
    return result[['product_id', 'average_price']]
```

Option 2:

- Join dataframes
- Filter the dataframe for ‘purchase_date’
- Compute ‘average_price’ and handle division by zero using lambda

```
def average_selling_price(prices: pd.DataFrame, units_sold: pd.DataFrame) -> pd.DataFrame:
    df = prices.merge(units_sold, how = 'left', on = 'product_id')
    df = df[((df['purchase_date']>= df['start_date']) & (df['purchase_date'] <= df['end_date'])) |
df['purchase_date'].isna()]
    result = df.groupby('product_id').apply(lambda x: round((x['price'] * x['units']).sum() /
x['units'].sum(), 2) if x['units'].sum() != 0 else 0).reset_index(name='average_price')
    return result
```

- Snapshot of the same code above for readability purposes

```
import pandas as pd

def average_selling_price(prices: pd.DataFrame, units_sold: pd.DataFrame) -> pd.DataFrame:
    df = prices.merge(units_sold, how = 'left', on = 'product_id')
    df = df[((df['purchase_date']>= df['start_date']) & (df['purchase_date'] <= df['end_date'])) | df
['purchase_date'].isna()]
    result = df.groupby('product_id').apply(lambda x: round((x['price'] * x['units']).sum() / x
['units'].sum(), 2) if x['units'].sum() != 0 else 0).reset_index(name='average_price')
    return result
```