Airline Price Standard Deviation

df["airline_price_std"] = df.groupby("Airline")["Price"].transform(lambda x : x.std())

not understand this code

Let’s break it down **step-by-step** so you can fully understand what this line of code does:

python

```python
df["airline_price_std"] = df.groupby("Airline")["Price"].transform(lambda x : x.std())
```

🎯 Goal of the code

Add a new column `airline_price_std` to the DataFrame `df` that shows the **standard deviation of the price** for each airline, repeated for every row of that airline.

🧩 Component Breakdown

✅ `df.groupby("Airline")`

This groups the data by each unique value in the **"Airline"** column.

✅ `["Price"]`

After grouping, we only focus on the **"Price"** column within each group.

So now we have:

python

```python
# Pseudo view:
"AirAsia" → [120, 130, 125]
"Emirates" → [500, 520, 510]
```

✅ `.transform(lambda x: x.std())`

- `x` here is the group of prices (like [120, 130, 125] for AirAsia).
- `x.std()` calculates the **standard deviation** of those prices.

But `.transform()` is key here:

- It **returns a Series of the same length as the original DataFrame**, not just one value per group.
- So each row for "AirAsia" will get the same standard deviation value — repeated.

📦 Final Assignment

```python
df["airline_price_std"] = ...
```

This assigns the **computed standard deviation for each airline** into a **new column**, repeated for all rows within the same airline.

✅ Example

Suppose your DataFrame looks like this:

Airline	Price
AirAsia	120
AirAsia	130
AirAsia	125
Emirates	500
Emirates	520
Emirates	510

Then `df["airline_price_std"]` will look like:

Airline	Price	airline_price_std
AirAsia	120	5.0
AirAsia	130	5.0

Airline	Price	airline_price_std
AirAsia	125	5.0
Emirates	500	10.0
Emirates	520	10.0
Emirates	510	10.0

(Standard deviation calculated within each group.)

Would you like a visual example with a small DataFrame?

.transform method not understand

Great — let's focus only on the `.transform()` method to help you understand it clearly. This is a very important method in pandas and often confusing at first.

🔍 What is `.transform()`?

`.transform()` is used **after a** `groupby()` operation. It:

- Applies a function **to each group**.
- **Returns a Series** with the **same number of rows as the original DataFrame** (not collapsed like `.agg()` or `.apply()`).
- Fills **each row with the result corresponding to its group.**

🧠 Key Rule

> 📌 `.transform()` returns a value **for each row**, not just for each group.

✅ Quick Comparison Example

python

```
import pandas as pd

df = pd.DataFrame({
    "Airline": ["A", "A", "A", "B", "B"],
    "Price": [100, 110, 120, 200, 210]
})
```

🟢 Using `.groupby().mean()` (collapses the group)

```python
df.groupby("Airline")["Price"].mean()
```

Output:

```css
Airline
A    110.0
B    205.0
Name: Price, dtype: float64
```

Only **2 rows**, one for each group.

🔵 Using `.transform("mean")` (preserves original shape)

```python
df["mean_price"] = df.groupby("Airline")["Price"].transform("mean")
```

Output:

Airline	Price	mean_price
A	100	110.0
A	110	110.0
A	120	110.0
B	200	205.0
B	210	205.0

Now each row **knows its group's mean**. That's the power of `.transform()`.

✅ You can use custom functions too:

python

```python
df["std_price"] = df.groupby("Airline")["Price"].transform(lambda x: x.std())
```

This will give each row the **standard deviation** of its group.

🟠 In Short:

Feature	`.agg()` / `.mean()`	`.transform()`
Output shape	One row per group	Same shape as original DataFrame
Use case	Summary stats (like report)	Add group-level info to each row
Good for	Group total, count, etc.	Normalizing, z-score, filling values

---

Would you like a diagram or runnable notebook showing this step by step?

is it something like - suppose we have two group, we calculate calculation as group wise and distribute that group result to every group member

✅ **Exactly right!**

That's a perfect way to understand `.transform()`.

🔁 Let me rephrase your idea with an analogy:

> Imagine you're in a classroom. The teacher calculates the **average test score per group** (say, for Group A and Group B), and then writes that **same average score on the desk of each student in that group**.

That’s what `.transform()` does.

📊 Example:

Student	Group	Score
Alex	A	80
Ben	A	90
Chloe	A	100
David	B	70
Emma	B	60

Let’s use `.transform("mean")` on this:

```python
df["group_avg"] = df.groupby("Group")["Score"].transform("mean")
```

Output:

Student	Group	Score	group_avg
Alex	A	80	90.0
Ben	A	90	90.0
Chloe	A	100	90.0
David	B	70	65.0
Emma	B	60	65.0

Each student **inherits** the average score of their group. That’s `.transform()`.

So yes — your sentence:

> “We calculate values group-wise and assign the result to every group member”

is 💯 correct.

Would you like to see how `.transform()` compares visually to `.apply()` or `.agg()` next?