

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

not understand this code

Sure! Let's break this line down step by step:

python



Copy



Edit

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

### 💡 What this line does:

It calculates the standard deviation of `Price` for each `Airline` and assigns it back to a new column `airline_price_std`, repeating the same standard deviation value for all rows of the same airline.

### 🔍 Step-by-step explanation:

1. `df.groupby("Airline")["Price"]`

This groups the DataFrame `df` by the column `Airline` and focuses only on the `Price` column.

For example, if `Airline` has values like `"Emirates"`, `"Qatar"`, etc., it will form groups like:

arduino



Copy



Edit

```
"Emirates" → [Price1, Price2, Price3, ...]
"Qatar"     → [Price4, Price5, ...]
```

2. `.transform(lambda x: x.std())`

- `x` is each group of `Price` values (one for each airline).
- `.std()` computes the standard deviation of that group.

`transform()` applies the function `(lambda x: x.std())` to each group but returns a Series of the same length as the original, repeating the result per row.

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

This creates a new column in the original DataFrame where each row gets the standard deviation of its airline's prices.

### ✅ Example:

python



Copy



Edit

```
import pandas as pd
df = pd.DataFrame({ "Airline": ["A", "A", "B", "B", "B"], "Price": [100, 200, 150, 160, 170] })
df["airline_price_std"] = df.groupby("Airline")["Price"].transform(lambda x: x.std())
print(df)
```

Output:

css



Copy



Edit

	Airline	Price	airline_price_std
0	A	100	70.710678
1	A	200	70.710678
2	B	150	10.000000
3	B	160	10.000000
4	B	170	10.000000

Every row within the same airline gets the same standard deviation value.

Would you like a similar explanation for `.mean()`, `.sum()`, or `.apply()`?







 Compare with Claude

not understand `.transform` method

No problem — let's focus purely on `.transform()` in pandas and make it easy to understand.

### 🔑 What is `.transform()` in pandas?

`.transform()` applies a function to each group, just like `.apply()`, but returns a result that's the same length as the original DataFrame.

In simpler words:

- `.apply()` gives one result per group.
- `.transform()` gives one result per row.

### 🧠 Think of `.transform()` as:

"Group by something, do a calculation for the group, and broadcast the result back to each member of that group."

### ✅ Example:

python

 Copy
  Edit

```
import pandas as pd
df = pd.DataFrame({ "Group": ["A", "A", "B", "B", "B"], "Value": [10, 20, 30, 40, 50] })
```

### 🎯 Goal:

Find the mean of each group and assign it to every row in that group.

python

 Copy
  Edit

```
df["group_mean"] = df.groupby("Group")["Value"].transform("mean")
```

### 🔍 What happens:

- Group A has values [10, 20] → mean = 15

- Group B has values [30, 40, 50] → mean = 40

So `df` becomes:

css

 Copy

 Edit

	Group	Value	group_mean
0	A	10	15
1	A	20	15
2	B	30	40
3	B	40	40
4	B	50	40

### Why not use `.apply()` here?

If you do this instead:

python

 Copy

 Edit

```
df.groupby("Group")["Value"].apply("mean")
```

You will just get:

css

 Copy

 Edit

```
Group
A    15.0
B    40.0
Name: Value, dtype: float64
```

This gives only one value per group — not broadcasted back to each row.

### In short:

- Use `.transform()` when you want **group-level stats repeated back** to each row in the group.
- Use `.apply()` when you want **one result per group**.

Would you like to try a small exercise with `.transform()` ?