



Concept | Window recipe

Watch the video



In this lesson, let's look at a powerful function for data enrichment, Windows. This lesson will cover:

- A conceptual overview of the [Window recipe](#).
- A practical demo of its advanced use in Dataiku on credit card transactions data.

Note

If you are already familiar with the concept of Windows, you can skip to the practical demo or move on to the tutorial.

A Window Cousin: The Group By Recipe

Before talking about the Window recipe, let's look at a related recipe, Group By.

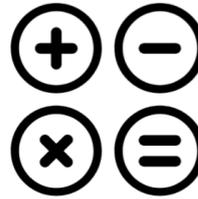
A **Group by** recipe has two important components:

- The group key
- The aggregations

GROUPING COMPONENTS



1. GROUP KEY



2. AGGREGATIONS

Let's first take a customer orders dataset. Then, we choose a group key, in this case, *Customer*. And then choose to compute some aggregations, in this case the average amount of each order.

Notice the dimensionality of our dataset has changed. We have just one row per grouping key, rather than one row per order.

The screenshot shows the Dataiku interface for a recipe named "compute_orders_prepared_by_tshirt_category". The recipe is currently in the "Group" step, which is highlighted with a green checkmark. The "Output column names" section is visible, listing the following columns:

Index	Column Name
1 (string)	tshirt_category
2 (date)	order_date_max
3 (double)	sale_value_sum
4 (bigint)	count

The "Output column names" table is highlighted with a red border. The interface also shows a "RUN" button at the bottom left and a "SAVED!" notification at the top right.

After applying a window

You may want to make these same grouped calculations on a dataset, while keeping the structure of the dataset the same. This is where we can use a window function.

A window function can perform this same grouped calculation, and append the values as a new column to the original dataset. This can help us make easy grouped comparisons or generate meaningful features for a machine learning model.

AFTER WINDOW



DATE	CUSTOMER	AMT	AVG_AMT
2020-07-12	Carol	2	3.33
2020-07-10	Carol	3	3.33
2020-01-25	Alice	1	3.5
2020-01-05	Bob	4	4
2020-07-21	Alice	6	3.5
2020-10-01	Carol	5	3.33



We can use a window to make calculations like:

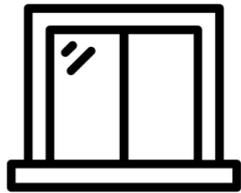
- Rank - is this a customer's first order? second? third?
- Lag - perhaps the quantity of each customer's previous order.
- A moving average can capture a customer's average order quantity over the previous three days.

Window components

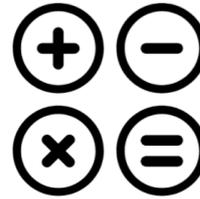
A window function has two important components:

- The window definition
- The aggregations

WINDOW COMPONENTS



1. WINDOW DEFINITION



2. AGGREGATIONS

These are similar to the components of a group by, with a few additions.

Window definition

Let's start with the window definition:

- First, we choose the partition, or column to group by — in this case, the *Customer*.
- Then, we order the rows within each partition by another column — in this case, by date, in descending order.
- We can optionally define a window frame based on the ordering column, in this case, the date. This can limit our aggregation calculation to just a subset of rows within each partition.

1. WINDOW DEFINITION

Order Column(s)

DATE	CUSTOMER	AMT
2020-07-10	Carol	3
2020-07-12	Carol	2
2020-10-01	Carol	5
2020-01-25	Alice	1
2020-07-21	Alice	6
2020-01-05	Bob	4

Partitioning Column(s)

Window Frame
*(Optional)

Window aggregations

Now let's look at the second component of a Window, the aggregations.

For each customer partition, ordered by date, and bounded by a window frame, what do you want to calculate?

Here, we chose to calculate the average amount — where the window definition was partition by customer, order by date, and no window frame. Remember, constricting the window frame is optional.

If we wanted to set a window frame which looks back at just the 3 previous months, and does not look at orders made in the future, our average amount calculation would look like this.

2. AGGREGATIONS



DATE	CUSTOMER	AMT
2020-07-10	Carol	3
2020-07-12	Carol	2
2020-10-01	Carol	5
2020-01-25	Alice	1
2020-07-21	Alice	6
2020-01-05	Bob	4



DATE	CUSTOMER	AMT	AVG_AMT
2020-07-10	Carol	3	3
2020-07-12	Carol	2	2.5
2020-10-01	Carol	5	3.33
2020-01-25	Alice	1	1
2020-07-21	Alice	6	6
2020-01-05	Bob	4	4

Let's take a close look at why setting this window frame changes our column values.

- We are partitioning our dataset by customer, ordering by date, setting a window frame of the previous 3 months till the current row's date, and then computing an average amount over this window definition.
- Same as before, we partition by customer, and order our rows within each partition by date.
- Then, we compute our average amount aggregation.

For Carol's first purchase, she has no purchase history, so the rolling three-month average is just the current purchase amount, 3. For her second purchase, we can average the amounts from the current purchase and the previous purchase, as that one happened within the last three months. For her third purchase, we can average the amounts from all three of her purchases which all happened within the last three months.

Then, with Alice's first purchase, we restart our average calculation, just considering the current purchase amount. For Alice's second purchase, we again restart our average

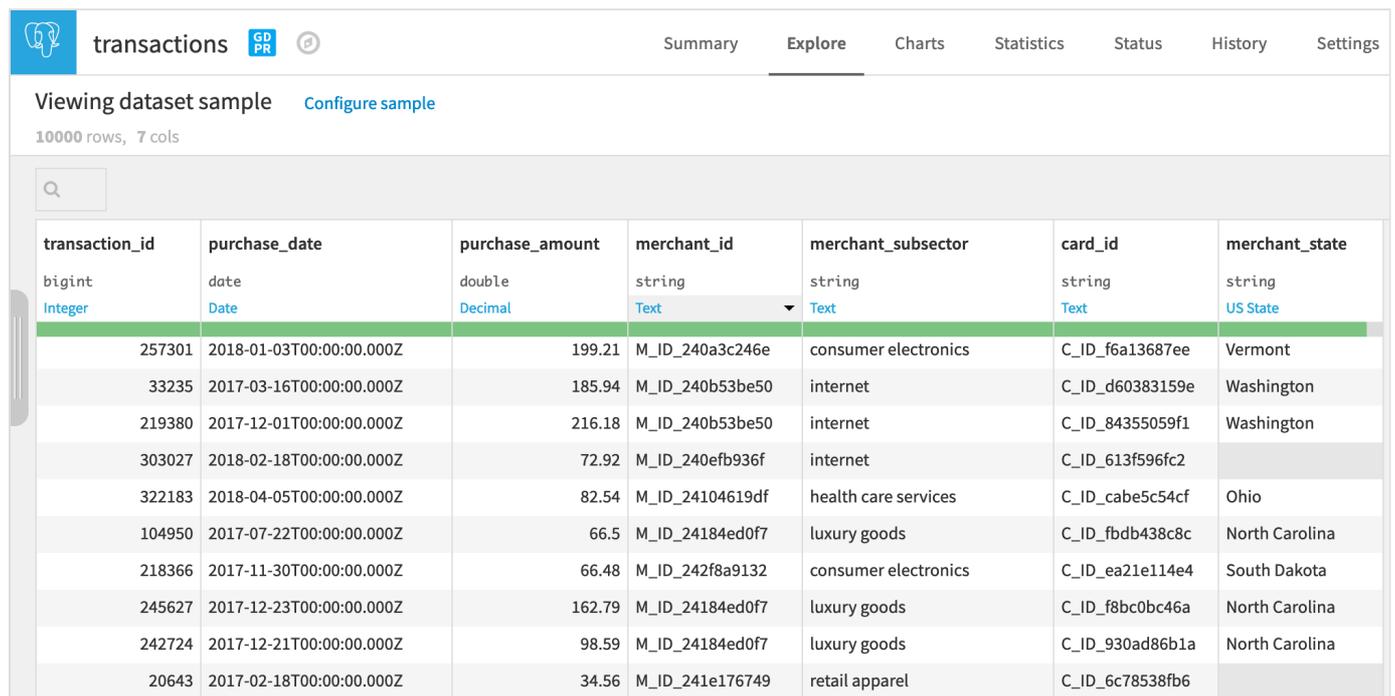
calculation. Her first purchase happened more than 3 months ago, so we exclude it from the aggregation calculation.

Bob only has one purchase, so his rolling 3 month average amount is just 4.

The Window recipe in Dataiku

Now, let's take a closer look at how Windows work in Dataiku with some more realistic data.

We start with a dataset of credit card transactions. Notice that we have a timestamp for each transaction, a transaction ID, a purchase amount, a merchant, and a merchant subsector, among other information.



The screenshot shows the Dataiku interface for a dataset named 'transactions'. The interface includes a search bar, a navigation menu with 'Summary', 'Explore', 'Charts', 'Statistics', 'Status', 'History', and 'Settings', and a table of data. The table has 7 columns: transaction_id, purchase_date, purchase_amount, merchant_id, merchant_subsector, card_id, and merchant_state. The data is displayed in a grid format with alternating row colors.

transaction_id	purchase_date	purchase_amount	merchant_id	merchant_subsector	card_id	merchant_state
257301	2018-01-03T00:00:00.000Z	199.21	M_ID_240a3c246e	consumer electronics	C_ID_f6a13687ee	Vermont
33235	2017-03-16T00:00:00.000Z	185.94	M_ID_240b53be50	internet	C_ID_d60383159e	Washington
219380	2017-12-01T00:00:00.000Z	216.18	M_ID_240b53be50	internet	C_ID_84355059f1	Washington
303027	2018-02-18T00:00:00.000Z	72.92	M_ID_240efb936f	internet	C_ID_613f596fc2	
322183	2018-04-05T00:00:00.000Z	82.54	M_ID_24104619df	health care services	C_ID_cabe5c54cf	Ohio
104950	2017-07-22T00:00:00.000Z	66.5	M_ID_24184ed0f7	luxury goods	C_ID_fbdb438c8c	North Carolina
218366	2017-11-30T00:00:00.000Z	66.48	M_ID_242f8a9132	consumer electronics	C_ID_ea21e114e4	South Dakota
245627	2017-12-23T00:00:00.000Z	162.79	M_ID_24184ed0f7	luxury goods	C_ID_f8bc0bc46a	North Carolina
242724	2017-12-21T00:00:00.000Z	98.59	M_ID_24184ed0f7	luxury goods	C_ID_930ad86b1a	North Carolina
20643	2017-02-18T00:00:00.000Z	34.56	M_ID_241e176749	retail apparel	C_ID_6c78538fb6	

Here, we want to look at the average and the sum of purchase amounts for a given merchant subsector, as well as for each individual merchant. But imagine we want these aggregations to include only today and the previous 3 days.

At the same time, we want to keep each transaction in a separate row so that we can compare the individual purchase amount to the average for this merchant and this subsector.

BUSINESS CASE

For each **merchant** in a given **subsector**,

what is the **average & sum** purchase amount

over the **previous 3 days** (and today)?

- **Partitioning Columns**
- **Aggregations**
- **Order By & Window Frame**

This is a type of problem that the Window recipe is great for: keeping the structure of the data the same while getting some additional information by looking at similar rows.

Defining the window

On the Window definition step, let's turn on Partitioning and choose *merchant_subsector*, because we want to find the sum and average amount of transactions made to each merchant. Additionally, we also want to look at the sum and average purchase amount for each merchant ID in each subsector, so we can add a second partitioning column — *merchant_id*.

Then let's order our rows within each partition by *purchase_date* in ascending order.

The third option — Window Frame — limits the number of rows your window can look at in each grouping. Let's turn it on:

- We have the option to limit the number of rows taken into account based on a value range from the order column, which in this case is the purchase date.
- Let's select this option and limit the rows to compute aggregations to only reflect the transactions made in the past 3 days, as well as the present day.

Interpret the output

After running the recipe, we can see our new columns in the output dataset:

- *purchase_amount_avg*
- *purchase_amount_sum*.

We can now filter by merchant ID and subsector to see the average and sum of purchases for this grouping within the defined window frame of the past three days.

The screenshot shows a Dataiku interface with a dataset named 'transactions_windows'. The interface includes a search bar, navigation tabs (Summary, Explore, Charts, Statistics, Status, History, Settings), and a 'PARENT RECIPE' button. The dataset is displayed in a table view with 10000 rows and 9 columns. The columns are: purchase_date (date), purchase_amount (double), merchant_id (string), merchant_subsector (string), card_id (string), merchant_state (string), purchase_amount_avg (double), and purchase_amount_sum (double). The last two columns, purchase_amount_avg and purchase_amount_sum, are highlighted with a red box. The table shows 142 matching rows for the filters merchant_id and merchant_subsector.

	purchase_date	purchase_amount	merchant_id	merchant_subsector	card_id	merchant_state	purchase_amount_avg	purchase_amount_sum
1389	2017-01-02T00:00:00.000Z	126.4	M_ID_384ffe1bef	carpentry	C_ID_ebdd8351c5	Kansas	126.4	126.4
4248	2017-01-10T00:00:00.000Z	47.51	M_ID_384ffe1bef	carpentry	C_ID_c7c6196822	Kansas	45.915	91.83
4430	2017-01-10T00:00:00.000Z	44.32	M_ID_384ffe1bef	carpentry	C_ID_d05479a737	Kansas	45.915	91.83
5718	2017-01-14T00:00:00.000Z	34.62	M_ID_384ffe1bef	carpentry	C_ID_1e93353d7f	Kansas	34.62	34.62
6496	2017-01-16T00:00:00.000Z	135.33	M_ID_384ffe1bef	carpentry	C_ID_8878eac4a9	Kansas	84.97500000000001	169.95000000000002
8900	2017-01-21T00:00:00.000Z	42.42	M_ID_384ffe1bef	carpentry	C_ID_9390b4a8a0	Kansas	42.42	42.42
10011	2017-01-24T00:00:00.000Z	101.79	M_ID_384ffe1bef	carpentry	C_ID_47941c6849	Kansas	53.400000000000006	160.20000000000002
9752	2017-01-24T00:00:00.000Z	15.99	M_ID_384ffe1bef	carpentry	C_ID_7237295f06	Kansas	53.400000000000006	160.20000000000002
12536	2017-01-30T00:00:00.000Z	40.91	M_ID_384ffe1bef	carpentry	C_ID_acb4360974	Kansas	40.910000000000004	40.910000000000004
13379	2017-02-01T00:00:00.000Z	50.29	M_ID_384ffe1bef	carpentry	C_ID_00f50add7b	Kansas	45.6	91.2
13500	2017-02-02T00:00:00.000Z	81.85	M_ID_384ffe1bef	carpentry	C_ID_ebdd8351c5	Kansas	59.0675	236.27
13464	2017-02-02T00:00:00.000Z	63.22	M_ID_384ffe1bef	carpentry	C_ID_05230dd540	Kansas	59.0675	236.27
14157	2017-02-03T00:00:00.000Z	61.44	M_ID_384ffe1bef	carpentry	C_ID_4871cf1b81	Kansas	64.2	256.8
15722	2017-02-07T00:00:00.000Z	34.4	M_ID_384ffe1bef	carpentry	C_ID_3bb6a6012c	Kansas	34.400000000000003	34.400000000000003
18397	2017-02-13T00:00:00.000Z	24.72	M_ID_384ffe1bef	carpentry	C_ID_95259ba4d0	Kansas	24.720000000000027	24.720000000000027
22479	2017-02-22T00:00:00.000Z	34.4	M_ID_384ffe1bef	carpentry	C_ID_3bb6a6012c	Kansas	34.400000000000003	34.400000000000003

What's next?

Continue learning about this recipe by working through the [Tutorial | Window recipe](#) article.

Tip

You can find this content (and more) by registering for the Dataiku Academy course, [Visual Recipes](#). When ready, challenge yourself to earn a [certification](#)!