

# Data Observability with Databand

Elena Lowery, WW Team, Data and AI





**Table of contents**

Contents

**Overview ..... 1**

**Required software, access, and files ..... 1**

**Part 1: Introduction to Databand SDK for data pipelines ..... 2**

**Part 2: Review a PySpark example ..... 26**

**Part 3: Create pipelines with lineage ..... Error! Bookmark not defined.**

**Part 4: Monitoring other pipelines ..... Error! Bookmark not defined.**

## Overview

---

In this lab you will learn how to use Databand to implement data observability for your data pipelines. You will complete the following tasks:

- Use Databand SDK in Python pipelines
- Use Databand SDK in PySpark pipelines

## Required software, access, and files

---

1. To complete exercises in this lab, you will need:

1. A *Python IDE*

- You can use any Python IDE with Python 3.8 or 3.9. We provide sample Python code/detailed instructions for
  - *Jupyter Notebooks* in [Cloud Pak for Data as a Service](#) (CPDaaS)
  - *JupyterLab* in [Anaconda Community Edition](#)
  - [PyCharm Community Edition](#) (can be installed with Anaconda Community Edition)

We recommend that you use at least 1 notebook environment and at least 1 IDE, such as *PyCharm* or *Visual Studio*.

2. A URL and a userid for the Databand demo environment (provided by the instructor).
3. Files from this [Box folder](#).

## Part 1: Introduction to Databand SDK for data pipelines

Databand provides the capability to monitor data pipeline status and create alerts for pipeline and data quality issues. In this section you will learn how to use the *Databand Tracking SDK* to monitor Python pipelines. Databand also supports data pipelines written in Java, Scala, and dbt. You can find more information about support for these pipelines in [documentation](#).

1. Log in to the Databand environment provided by your instructor.
2. Navigate to the **Pipelines** page and find the *prepare\_sales\_data* pipeline. You can filter by project – *Retail Analytics*.

This simple data pipeline reads data from a csv file, filters out several columns, and writes 2 datasets for different product categories.

The screenshot shows the Databand interface with the 'Pipelines' tab selected. The 'Project' filter is set to 'Retail Analytics'. The table below shows the status of three pipelines:

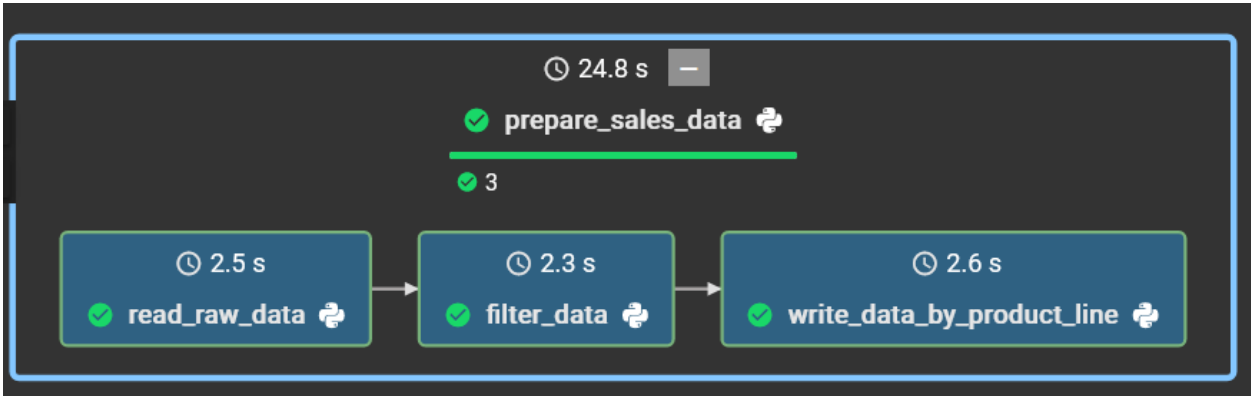
| Name                 | Owner     | Project          | Run States           |
|----------------------|-----------|------------------|----------------------|
| prepare_sales_data_2 | 1A3030897 | Retail Analytics | 3 Failed, 16 Success |
| prepare_sales_data_1 | 1A3030897 | Retail Analytics | 9 Failed, 8 Success  |
| prepare_sales_data   | wsuser    | Retail Analytics | 12 Failed, 9 Success |

3. Click on the pipeline, then on one of the successful runs.

The screenshot shows the 'prepare\_sales\_data' pipeline page. The 'Run list' tab is selected. The table below shows the details of the runs:

| Run Name | Status | Start Time               | End Time                 |
|----------|--------|--------------------------|--------------------------|
| weekly   | ✓      | Aug 18, 2022 02:54:31 PM | Aug 18, 2022 02:54:51 PM |
| weekly   | ✓      | Aug 18, 2022 02:53:59 PM | Aug 18, 2022 02:54:25 PM |

This view shows that the pipeline has three steps: *read\_raw\_data*, *filter\_data*, and *write\_data\_by\_product\_line*.



2. Click on the *read\_raw\_data* step in the pipeline, then switch to the **Metrics** tab.

The **Metrics** tab shows default and custom metrics that can be monitored for each step of the pipeline. The metrics on this page will be easier to understand after we review code for the pipeline.

The screenshot shows the 'Metrics' tab interface. On the left, there is a search bar labeled 'Search metric or task'. Below it is a table with two columns: 'Key' and 'Value'. The table contains several rows of metrics related to a CSV file. The fourth row is highlighted with a blue border. On the right side of the interface, a partial view of the pipeline step 'read\_raw\_data' (2.5 s) is visible, enclosed in a green box.

| Key   | Value                |
|---|----------------------|
| ☆ Retail_Products_and_Customers.csv.rows    | 88475                |
| ☆ Retail_Products_and_Customers.csv.columns | 24                   |
| ☆ Retail_Products_and_Customers.csv.shape1  | 24                   |
| ☆ Retail_Products_and_Customers.csv         | Show Schema 88475x24 |
| ☆ Retail_Products_and_Customers.csv.schema  | Show Schema          |
| ☆ Retail_Products_and_Customers.csv.shape0  | 88475                |

Click **Show Schema** next to *Retail\_Products\_And\_Customers.csv*. Here we see a preview of data that was read in the *read\_raw\_data* step of the pipeline.

Preview

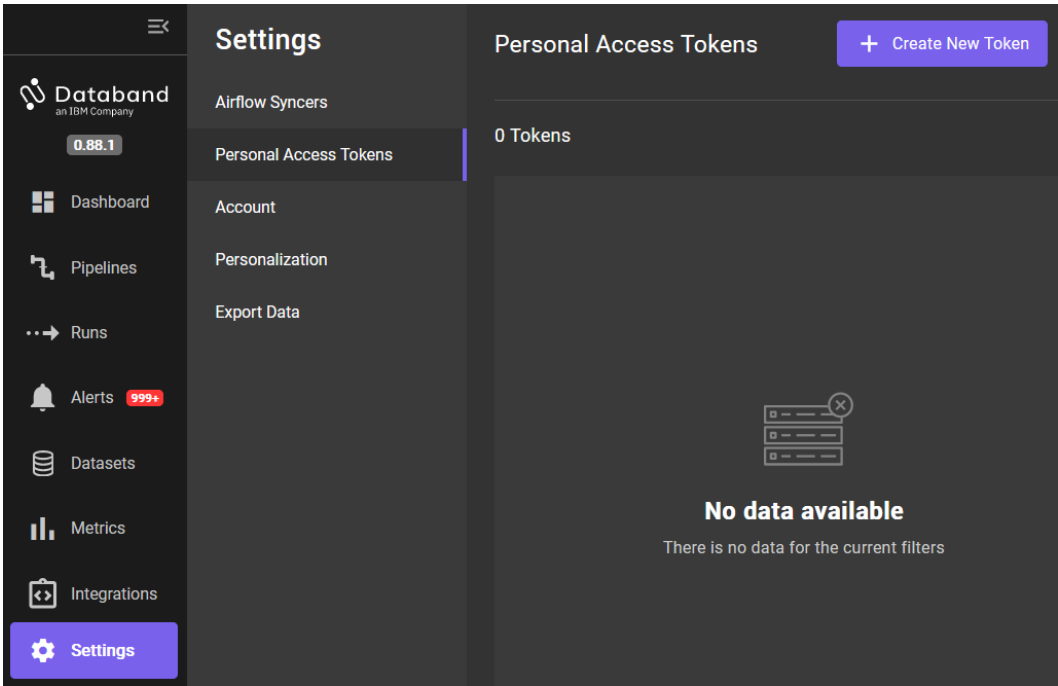
► DataFrame: 88475x24

|    | 1   | Buy | ID | Gender | Status | Children | Est_Income | Age | MARITAL | STATUS         | PROFESSION   | EDUCATION      | BUYER | CATEGORY |
|----|-----|-----|----|--------|--------|----------|------------|-----|---------|----------------|--------------|----------------|-------|----------|
| 2  | No  | 1   | F  | S      | 1      | 38000.00 | 24.393333  |     | S       | Missing        | Missing      |                | NaN   |          |
| 3  | Yes | 6   | M  | M      | 2      | 29616.00 | 49.426667  |     | S       | Missing        | Missing      |                | NaN   |          |
| 4  | Yes | 8   | M  | M      | 0      | 19732.80 | 50.673333  |     | S       | Private Sector | High School  |                | NaN   |          |
| 5  | No  | 11  | M  | S      | 2      | 96.33    | 56.473333  |     | S       | Private Sector | High School  | Product Viewer |       |          |
| 6  | No  | 14  | F  | M      | 2      | 52004.80 | 25.140000  |     | U       | Pharmacist     | Post College | Visitor        |       |          |
| 7  | No  | 17  | M  | M      | 2      | 53010.80 | 18.840000  |     | U       | Private Sector | High School  | Product Viewer |       |          |
| 8  | Yes | 18  | M  | M      | 1      | 75004.50 | 64.800000  |     | M       | Engineer       | College      | Product Viewer |       |          |
| 9  | Yes | 21  | M  | M      | 0      | 19749.30 | 60.366667  |     | U       | Engineer       | Post College | Product Viewer |       |          |
| 10 | No  | 22  | M  | S      | 1      | 57626.90 | 43.906667  |     | S       | Engineer       | Post College | Repeat Buyer   |       |          |
| 11 | Yes | 23  | M  | M      | 2      | 20078.00 | 32.846667  |     | S       | Pharmacist     | Post College | Product Viewer |       |          |

Next, you will create a copy of this pipeline and explore the details of monitoring in Databand.

4. Get an authorization token from Databand. The token is used in data pipelines when establishing a connection to Databand.

In Databand click on **Settings**, then select the **Personal Access Tokens** tab. Click the **Create New Token** button. Provide a unique name (for example, add your initials), and save the token in a notepad.



5. Load the sample data file and the Python script into your IDE. The samples are located in the *Workshop/Pipelines* folder (downloaded from Box).

- Script: *SimpleRetailDataPipeline\_with\_Databand.py*
- Csv file: *Retail\_Products\_and\_Customers.csv*
- Notebook: *SimpleRetailDataPipeline\_with\_Databand.ipynb*

In this lab we will review the *Python script*. If you're using the notebook example, see the instructions and comments in the notebook.

3. In the beginning of the script, we import pandas and Databand libraries. If your Python IDE does not automatically install libraries, then you can install them with pip:

- `pip install pandas`
- `pip install databand`

```
# Import pandas and databand libraries
import pandas as pd
from dbnd import dbnd_tracking, task, dataset_op_logger
```

4. Replace the *url* and *token* variables with the values for your Databand cluster.

```
# Import pandas and databand libraries
import pandas as pd
from dbnd import dbnd_tracking, task, dataset_op_logger

databand_url = 'insert_url'
databand_access_token = 'insert_token'
```

5. Replace the value of the *unique\_suffix* variable to your initials.

Since we may have many workshop participants using the same cluster, adding a unique suffix to assets that are tracked in Databand will make it easier for you to find your pipelines and datasets.

```
# Provide a unique suffix that will be added to various assets tracked in Databand. We use this approach because
# in a workshop many users are running the same sample pipelines
unique_suffix = '_el'
```

6. Scroll down to the bottom of the script – to the *prepare\_retail\_data()* function.

Review the code:

- This function starts tracking the pipeline in Databand and invokes 3 functions that represent pipeline steps: *read\_raw\_data()*, *filter\_data()*, *write\_data\_by\_product\_line()*. Notice that in Databand the names of the steps match the names of the Python functions in our pipeline.

```
def prepare_retail_data():
    with dbnd_tracking(
        conf={
            "core": {
                "databand_url": databand_url,
                "databand_access_token": databand_access_token,
            }
        },
        job_name="prepare_sales_data" + unique_suffix,
        run_name="weekly",
        project_name="Retail Analytics" + unique_suffix,
    ):
        # Call the step job - read data
        rawData = read_raw_data()

        # Filter data
        filteredData = filter_data(rawData)

        # Write data by product line
        write_data_by_product_line(filteredData)

    print("Finished running the pipeline")
```

- The main function, *prepare\_retail\_data*, starts tracking execution of the pipeline in Databand. Notice that the *job\_name* corresponds to the *pipeline name* in Databand and *project\_name* corresponds to the *project name*.

```
with dbnd_tracking(
    conf={
        "core": {
            "databand_url": databand_url,
            "databand_access_token": databand_access_token,
        }
    },
    job_name="prepare_sales_data" + unique_suffix,
    run_name="weekly",
    project_name="Retail Analytics" + unique_suffix,
):
```

| Name                    | Owner     | Project          | Run States             |
|-------------------------|-----------|------------------|------------------------|
| ^L prepare_sales_data_2 | 1A3030897 | Retail Analytics | 3 Failed<br>16 Success |
| ^L prepare_sales_data_1 | 1A3030897 | Retail Analytics | 9 Failed<br>8 Success  |
| ^L prepare_sales_data   | wsuser    | Retail Analytics | 12 Failed<br>9 Success |

- Review the implementation of each function.
  - First, we add the *@task* decorator before the declaration of each function in Python. This lets Databand know that we are starting execution of a pipeline step. The *@task* decorator uses the name of the function directly below it as the name of the pipeline step in Databand.
  - Next, we log datasets that are used in the pipeline with the *logger.set()* call. This call will log *metadata* and a small set of sample data. If the customer has concerns about logging data, they can turn off logging the data sample.
  - In our example we read a csv file into a pandas dataframe, which is what's logged in Databand. Logging dataset metadata in Databand is optional, and it should only be used if data engineers need dataset metadata for troubleshooting issues. When you log datasets, you will see data in the **Metrics** tab of a pipeline step. If you switch back to Databand **Metrics** view



of the `read_raw_data` step, you will notice that the schema and sample records match the few rows of the csv file.

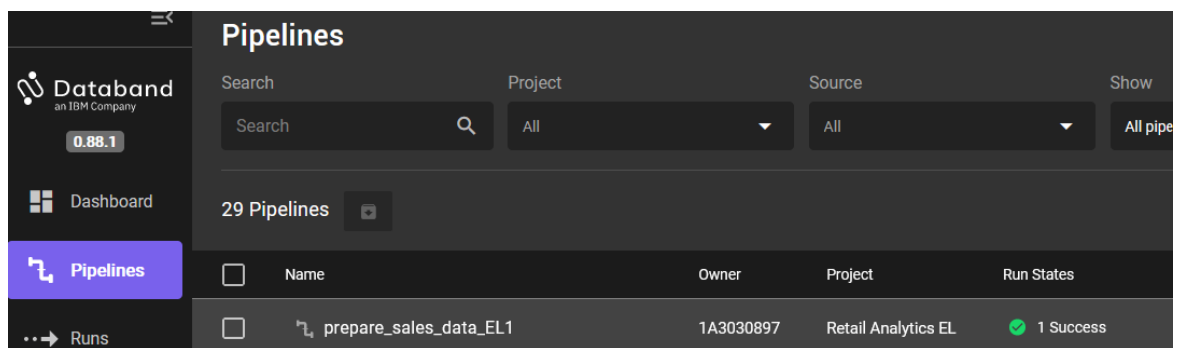
```
@task
def read_raw_data():
    retailData = pd.read_csv('Retail_Products_and_Customers.csv')

    # Log the data read
    with dataset_op_logger("local://WeeklySales/Retail_Products_and_Customers.csv", "read", with_schema=True,
                          with_preview=True) as logger:
        logger.set(data=retailData)

    return retailData
```

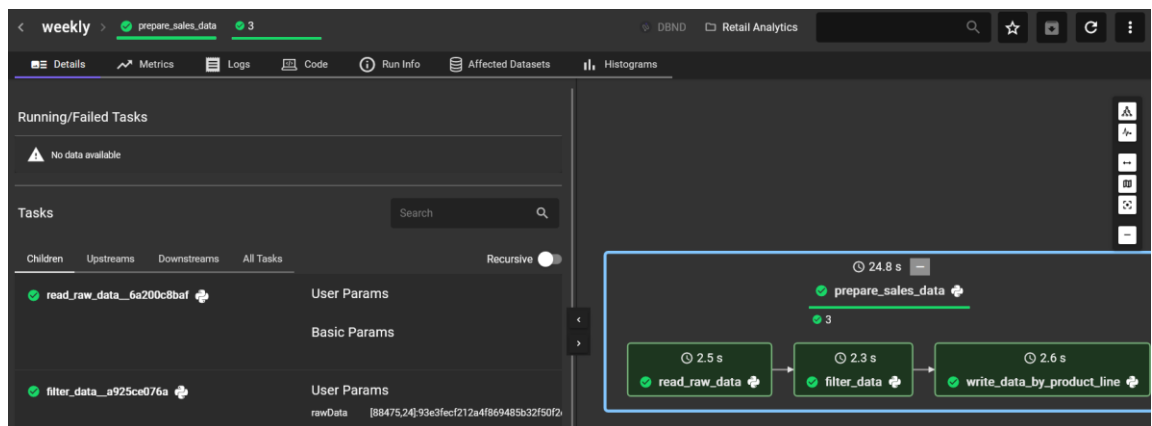
Next, we will run the code.

7. Save the changes you've made in the script and run it at least 5 times to generate some metrics data. You can run the script in debug mode if you would like to review the SDK in more detail.
  - As you are running the script, switch to the Databand environment, and monitor pipeline execution. You can find the pipeline in the **Pipelines** view.



| Pipelines                |                        |           |                     |            |
|--------------------------|------------------------|-----------|---------------------|------------|
| Search                   | Project                | Source    | Show                |            |
| Search                   | All                    | All       | All pipe            |            |
| 29 Pipelines             |                        |           |                     |            |
| <input type="checkbox"/> | Name                   | Owner     | Project             | Run States |
| <input type="checkbox"/> | prepare_sales_data_EL1 | 1A3030897 | Retail Analytics EL | 1 Success  |

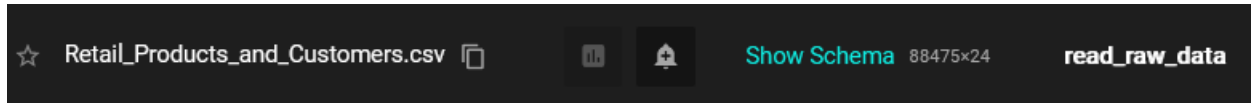
- Drill down to see the details of the run.



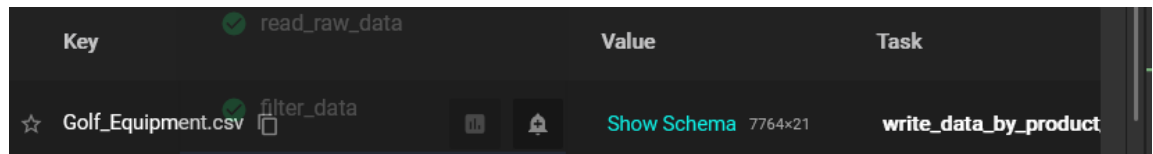
- Click on the **Metrics** tab, then select each step of the pipeline.

Explore the **Metrics**, notice that the schema corresponds to the dataset read or written in each step.

For example, the *Retail\_Products\_And\_Customers.csv* dataset (in Python the *retailData* pandas dataframe) has 88,475 rows and 24 columns.

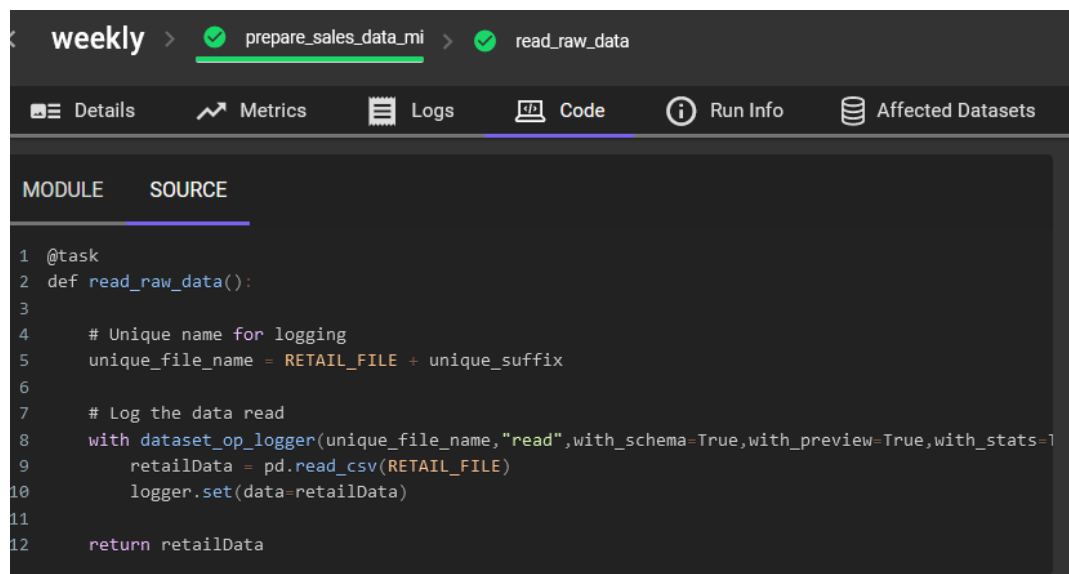


If you click on the *write\_data\_by\_product\_line* step, you will notice that one of the output datasets, *Golf Equipment* (in Python the *GolfEquipment* pandas dataframe), has 7764 rows and 21 columns.



- Click on the **Code** tab, then select each step of the pipeline.

Here we can review (but not edit) code that corresponds to the pipeline step. Databand is not used for editing code, code changes should be done in the IDE that's used to develop pipelines.

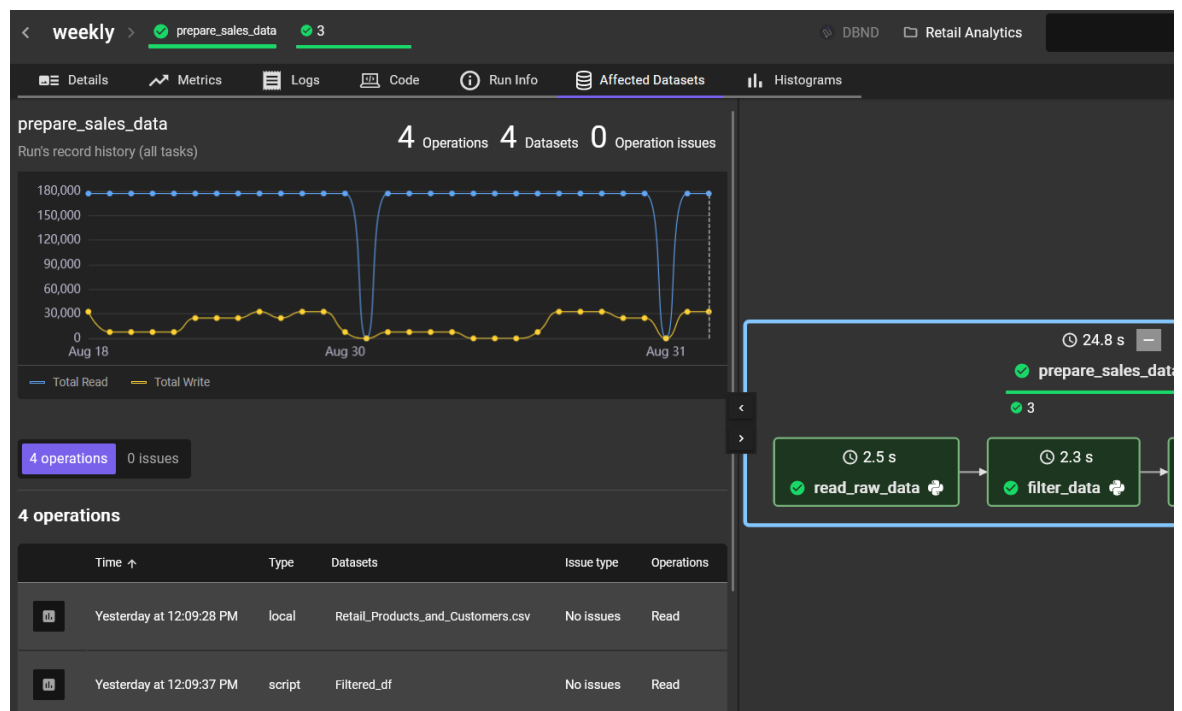


- Click on the **Run info** tab and notice that it captures the name of the Python script that implements the pipeline.

The screenshot shows the 'Run Info' tab in the Databand interface. At the top, there's a breadcrumb navigation: < weekly > prepare\_sales\_data 3. Below this, a toolbar contains icons for Details, Metrics, Logs, Code, Run Info (active), and Affected Datasets. The main content area is titled 'Run Info' and displays the following information:

- Command:** \$ SimpleRetailDataPipeline\_with\_DatabandEL.py
- Full command:** \$ C:/Users/ElenaLowery/PycharmProjects/Databand/modules/SimpleRetailDataPipeline\_...
- Start Time:** Yesterday at 12:09:26 PM
- End Time:** Yesterday at 12:09:51 PM

9. Click on the **Affected Datasets** tab and the “top” pipeline view (you should see a bold blue box around the entire pipeline, and not the individual steps). Here we have a single view of all datasets used by the pipeline. Since we don’t have any issues yet, we don’t need to review additional details at this time.



Next, we will introduce a few errors in the pipeline to understand how Databand can help us monitor and troubleshoot the errors.

We will start with introducing an error in the last step of the pipeline.

*Note: In this lab we are adding errors that are typically resolved at **build time** because it's the easiest way to create an error in code. In a production environment **runtime errors** will be handled the same way by Databand.*

- In the `write_data_by_product_Line()` function find the line of code that writes the equipment csv and change it to the name of the pandas dataframe that doesn't exist, for example, `golfEquipment1`.

*Note: While PyCharm shows the error, the code will still run because Python is not a compiled language.*

```
# Select any product line
golfEquipment = filteredData.loc[filteredData['Product line'] == 'Golf Equipment']

# Log the filtered data read
with dataset_op_logger(unique_file_name_2, "write", with_schema=True,
                      with_preview=True) as logger:
    # Write the csv file
    golfEquipment1.to_csv("Golf_Equipment.csv", index=False)

    loader.set(data=golfEquipment)
```

- Save the change and run the script. Switch to Databand and find your pipeline in the **Pipelines** tab.

Databand shows that the 3<sup>rd</sup> step of the pipeline failed and displays the error that caused the failure.

The screenshot shows the Databand interface for a pipeline named 'prepare\_sales\_data\_EL1'. The pipeline consists of three steps: 'readRawData' (2.2 s), 'filterData' (1.9 s), and 'writeDataByProductLine' (2 s). The 'writeDataByProductLine' step has failed, indicated by a red error icon. The error log for this step shows a NameError: name 'golfEquipment1' is not defined. The error log text is as follows:

```
File "C:\Users\elenalower\PycharmProjects\Databand\venv\lib\site-packages\dbnd\core\task.py", line 10, in <module>
    yield func_call.set_result
File "C:\Users\elenalower\PycharmProjects\Databand\venv\lib\site-packages\dbnd\core\task.py", line 10, in <module>
    fp_result = self.class_or_func(*call_args, **call_kwargs)
File "C:\Users\elenalower\PycharmProjects\Databand\modules\SimpleRetailDataPipeline.py", line 10, in writeDataByProductLine
    golfEquipment1.to_csv("Golf_Equipment.csv", index=False)
```

- Switch to the **Alerts** view and click **Add Alert**.

Create an alert for **Run State Failed**, and make it a **Critical** severity alert.

Run the pipeline again and refresh the **Alert** view. You should see a critical alert for your failed pipeline run.

### 13. Drill down to the alert and review the details.

As on the **Pipelines** page, we can see the error that caused the failure. We can also review the datasets that could be potentially affected by the error in the pipeline step.

During the first few failed runs of a new pipeline that's observed by Databand, you may see the message *"Unable to determine alert impact."* This happens because the impact analysis job did not run yet. If you see this message, return to the alert after 10-15 minutes. You can continue with the rest of the instructions in the lab.

< Alert: `prepare\_sales\_data EL` run state: `failed` CRITICAL

Type: Run state    Triggered Time: Today at 11:35:36 AM    Status: ● Triggered    Project: Retail Analytics EL


Overview    Lineage

**Origin**

Pipeline: [🔗 prepare\\_sales\\_data EL](#) >

Run: [↻ weekly](#) >

**Impact Analysis**



**Unable to determine alert impact**

To trace the impact of an alert, you must be logging dataset operations for the pipeline that triggered the alert.

[Learn more about tracking dataset operations](#)

**Error message**

```

1 - write_data_by_product_line
2   Traceback (most recent call last):
3     File "C:\Users\Elenal\Scripts\prepare_sales_data.py", line 10, in <module>
4       yield func_call.set
5     File "C:\Users\Elenal\Scripts\prepare_sales_data.py", line 10, in <module>
6       fp_result = self.cl
7     File "C:\Users\Elenal\Scripts\prepare_sales_data.py", line 10, in <module>
8       golfEquipment1.to_csv('Golf_Equipment.csv')

```

**Receivers**

[https://data-band.ibm.com/alerts/...](#)

Databand is able to determine (infer) the list of affected datasets because

- We used the `logger.set()` function in our data pipeline
- We had several successful runs of the pipeline, and these datasets were read or written in the successful runs.

Type: Run state    Triggered Time: Today at 11:51:50 AM    Status: ● Triggered    Project: Retail Analytics EL

Overview    Lineage

**Origin**

Pipeline: [🔗 prepare\\_sales\\_data EL](#) >

Run: [↻ weekly](#) >

**Impact Analysis**

Affected datasets 2    Missing operations 2

[📄 Golf\\_Equipment.csv](#) >

[📄 Camping\\_Equipment.csv](#) >

Since the error in a Python function can happen before or after the dataset is written, we need to investigate if the expected number of rows was written to the dataset.

14. Click on *Camping\_Equipment.csv*, then the **History** tab.

Based on this output, we have a consistent number of rows/columns that are being written to this dataset, including during the pipeline run that has failed (the last run shown on top of the table). That means that even though the pipeline run failed, data was written successfully.

This output aligns with the error we introduced in the code – it’s after we write the *CampingEquipment.csv*.

Overview

History

Operations

Seen during

Operation type

Pipeline

Issue type

Last 7 days

All

All

All

22 Operations

| Status                            | Type  | Issue type | Schema     | Records        | Origin  |
|-----------------------------------|-------|------------|------------|----------------|---|
| <div><div></div><div></div></div> | Write | No issues  | 21 columns | 24,866 records | <div><div></div>prepare_sales_data EL &gt;</div> <div>→ weekly &gt;</div> <div><div></div>write_data_by_product_line &gt;</div> |
| <div><div></div><div></div></div> | Write | No issues  | 21 columns | 24,866 records | <div><div></div>prepare_sales_data EL &gt;</div> <div>→ weekly &gt;</div> <div><div></div>write_data_by_product_line &gt;</div> |

Next, we will introduce the error earlier in the code.

15. In the `write_data_by_product_line()` function find the line of code that references the pandas dataframe that was passed into the function, and change it to a different name, for example, *filteredData1*.

Since *filteredData1* does not exist in this function, we will get an error.

Run the script and switch back to Databand.

```
@task
def write_data_by_product_line(filteredData):
    # Select any product line - we will write it to a separate file
    campingEquipment = filteredData1.loc[filteredData['Product line'] == 'Camping Equipment']
```

16. From the **Alerts** page, navigate to your failed pipeline run.

Notice that in addition to the **Affected datasets** tab, we now see **Missing operations** tab. A “missing operation” means that the code that writes datasets did not run. Databand knows that a successful execution should result in writing the *Camping Equipment* and *Golf Equipment* datasets because we had several successful runs of the pipeline.

In general, Databand *infers* affected datasets and pipelines by observing successful pipeline runs, which means that we should always ensure that a pipeline runs successfully several times after we configure it to be observed by Databand.

| Impact Analysis   |                       |                                  |
|-------------------|-----------------------|----------------------------------|
| Affected datasets | 2                     | Missing operations               |
| Operation type    | Dataset               | Description                      |
| Write             | Camping_Equipment.csv | Data wasn't written as expected. |
| Write             | Golf_Equipment.csv    | Data wasn't written as expected. |

Next, we will review what happens if we have errors earlier in the pipeline.

17. Add an error in the 1<sup>st</sup> step of the pipeline. For example, change the name of the csv file to the name of the file that does not exist (global variable). Save the script and run it.

```
# Data used in this pipeline
```

```
RETAIL_FILE = "https://raw.githubusercontent.com/elenalowery/data-samples/main/Retail_Products_and_Customers1.csv"
```

Switch to Databand and find your failed pipeline run (from the **Pipelines** view).

Since the pipeline failed on the first step, the other steps are not shown.

The screenshot shows the Databand interface with two panes. The left pane, titled 'Running/Failed Tasks', shows a task named 'read\_raw\_data\_a28b964968' with a red error icon. Below it, the 'Error logs' section displays a Python traceback for a `FileNotFoundError` with the message: `[Errno 2] No such file or directory: 'Retail_Products_and_Customers1.csv'`. The right pane shows a summary of the pipeline run. It lists two tasks: 'prepare\_sales\_data EL' with a yellow warning icon and 'read\_raw\_data' with a red error icon. The 'read\_raw\_data' task is highlighted with a red box and shows a duration of 0.2 s.

We can see that more datasets are shown in the **Missing Operations** tab of the pipeline alert.



The screenshot shows the Databand Alerts panel. The top header indicates a critical alert: "Alert: `prepare\_sales\_data EL` run state: `failed`" with a "CRITICAL" badge. The alert details show it was triggered at 12:00:57 PM and is in a "Triggered" state. The origin is identified as the "prepare\_sales\_data EL" pipeline, which runs weekly. The "Impact Analysis" section shows that 2 datasets were affected and 4 operations were missing. The affected operations are listed in the table below:

| Operation type | Dataset                           | Description                      |
|----------------|-----------------------------------|----------------------------------|
| Read           | Retail_Products_and_Customers.csv | Data wasn't read as expected.    |
| Read           | Filtered_df                       | Data wasn't read as expected.    |
| Write          | Camping_Equipment.csv             | Data wasn't written as expected. |
| Write          | Golf_Equipment.csv                | Data wasn't written as expected. |

Next, we will configure 2 other types of alerts that are not pipeline failures, but nevertheless could signal a problem with the pipeline or data.

One of the most frequent alerts that a data engineering team is interested in is the *run duration* alert because it can indicate a problem with the pipeline.

18. In Databand switch to the **Alerts** panel and click **Add Alert**.

The screenshot shows two buttons: "Add Receiver" and "Add Alert". The "Add Alert" button is highlighted with a purple background.

19. Select your pipeline.

The screenshot shows the "Create alert" dialog box. The "Define Alert" step is selected. The "Add alert for" section shows "Single pipeline" selected. The "Add Alert Settings For Pipeline" dropdown is set to "prepare\_sales\_data\_EL1".

20. Select **Run Duration (seconds)**, then **>**, and enter the value that's smaller than the shortest duration of your pipeline runs.

In our example the longest successful run is 17.3 seconds, that’s why we are specifying 15 seconds because we want to “force” the alert. Select any alert severity, then add a name and a description.

Notice that we have many options for creating alerts on this page. At this time we will focus on the overall pipeline duration. Click **Save**. Optionally, assign receivers for the alert.

| Run                  | Value  |
|----------------------|--------|
| Today at 12:00:34 PM | 2.3 s  |
| Today at 11:51:28 AM | 16.1 s |
| Today at 11:34:42 AM | 18.9 s |
| Today at 11:34:05 AM | 17.1 s |
| Today at 11:29:58 AM | 16.9 s |
| Today at 11:25:16 AM | 24.4 s |

Show active alerts

All alerts

19 Alerts defined Delete

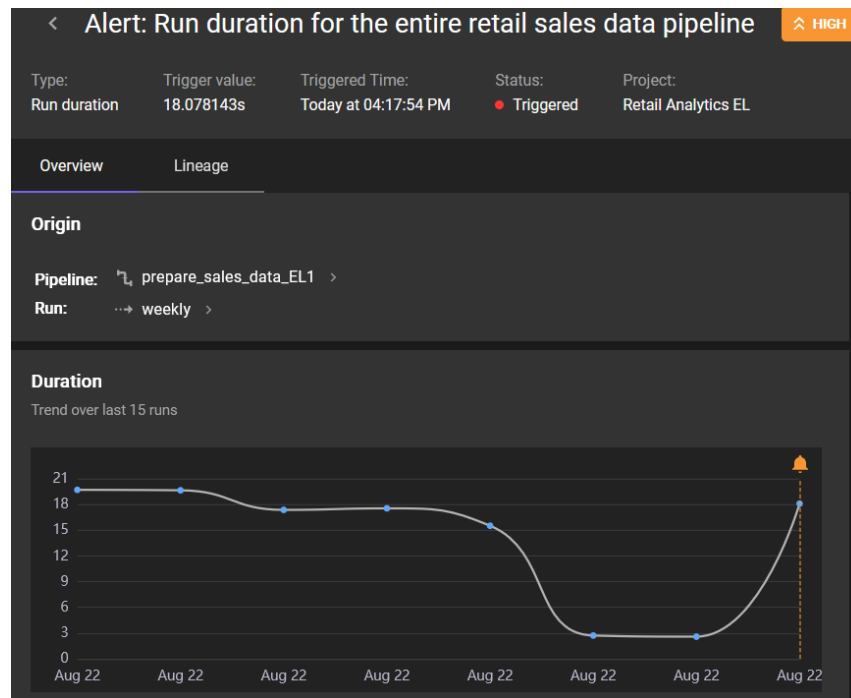
|                          | Active                              | Severity | Alert type   | Condition        | Origin                 |
|--------------------------|-------------------------------------|----------|--------------|------------------|------------------------|
| <input type="checkbox"/> | <input checked="" type="checkbox"/> | HIGH     | Run duration | greater than 15s | prepare_sales_data_EL1 |

21. Run your pipeline and check the **Alerts** page.

193 alerts Acknowledge Resolve Last seen

|   | Origin  |
|---|---|
| <input type="checkbox"/> <span>HIGH</span> Run duration <span>prepare_sales_data_EL1</span> | Run duration for the entire retail sales data pipeline ><br>Trigger value: 18.078143s<br>Today at 04:17:54 PM <span>Retail Analytics EL</span> <span>DBND</span> <span>Triggered</span> |

22. Drill down to the alert to view the details.

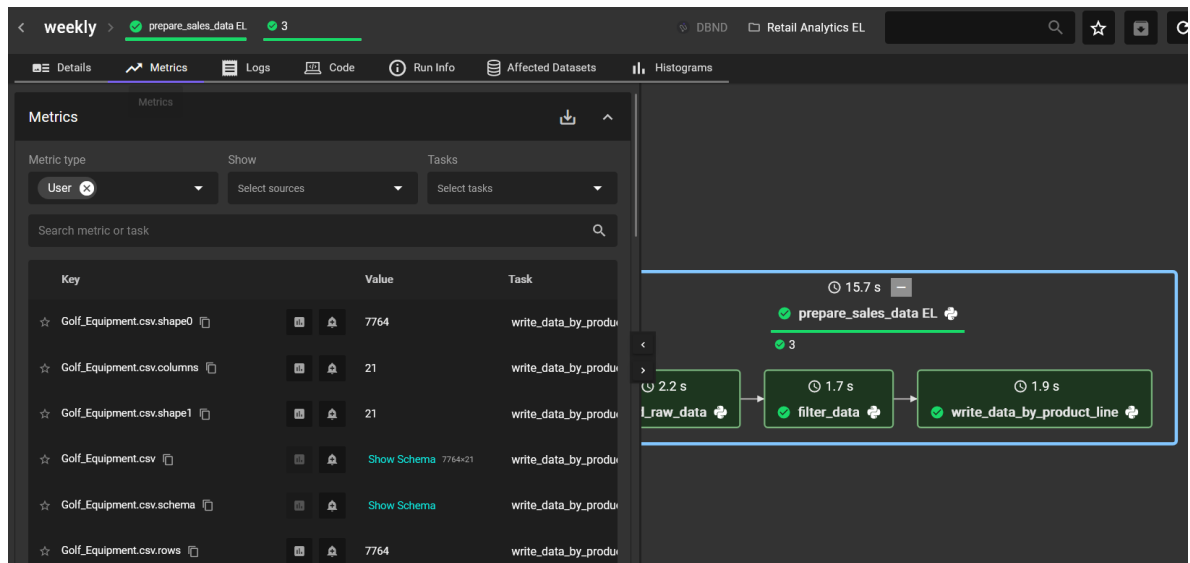


Now that you've tested the alert, either delete it or change the run duration to a higher value so that it does not generate too many unnecessary alerts in the workshop environment.

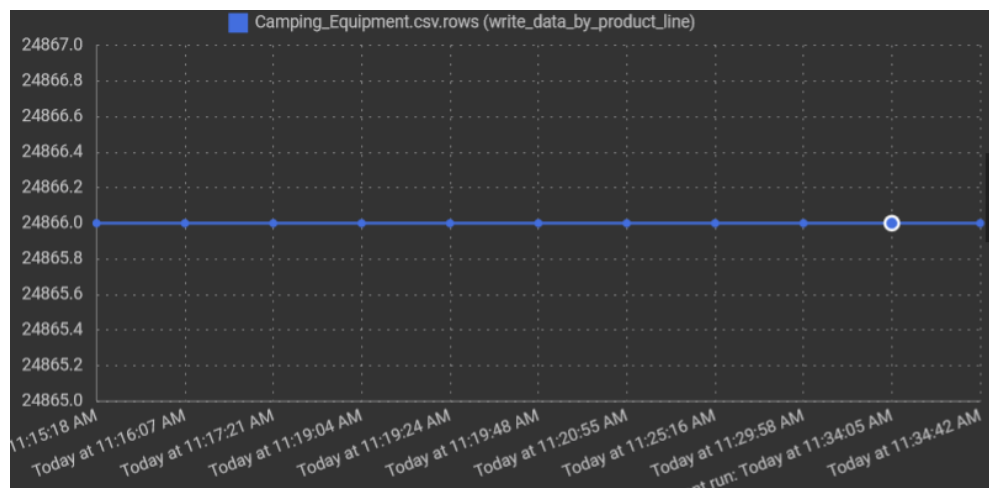
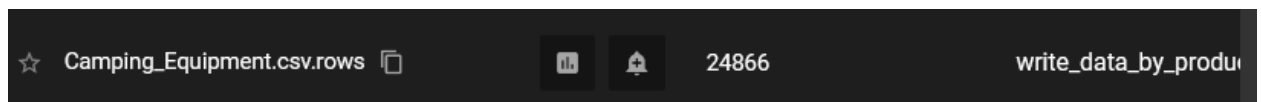
Next, we will create an alert for the *number of rows written* to the output dataset. We will use a different navigation approach to get to the alert definition page.

23. Navigate to your pipeline from the **Pipelines** panel, drill down to one of the successful runs, and make sure that the entire pipeline is selected (blue bold box around the entire pipeline).

Click on the **Metrics** tab.



Click on the **Show on chart** icon next to *Camping\_Equipment.csv.rows* and review the number of rows that has been written up to this point (scroll down to see the chart). Since we did not have runtime errors, the number of rows has consistently been 24,688. As a reminder, we created errors that resulted in a "missing operation".



24. Click the **Create Alert** button next to *Camping\_Equipment.csv.rows* (the bell icon). Notice that when the **Create alert** panel is displayed, the number of rows is pre-selected.

You can choose whether to create an alert based on a hardcoded value or an anomaly. We decided to create an alert for the number of rows less than 10,000.

If you don't want to change the pipeline code to test this alert, then simply specify the number less than 24,688 to create the alert. However, we recommend that you change the code to test the alert.

Select the *Camping\_Equipment.csv\_write\_rows* as the metric.

**Create alert**

Define Alert (1) | Assign Receiver (2)

Add alert for: **Single pipeline** | Multiple pipelines | Data

Add Alert Settings For Pipeline: prepare\_sales\_data EL | DBND

**Metric**

Search

Run Duration (seconds)  
Missing data operations  
Schema change

**Task**

filter\_data  
read\_raw\_data  
write\_data\_by\_product\_line

**Metrics**

Camping\_Equipment.csv\_write\_rows  
Camping\_Equipment.csv  
Golf\_Equipment.csv\_write\_rows  
Golf\_Equipment.csv.shape0  
Golf\_Equipment.csv.rows

**Value**

10000

**Recent Values**

| Run                  |
|----------------------|
| Today at 11:34:42 AM |
| Today at 11:34:05 AM |
| Today at 11:29:58 AM |
| Today at 11:25:16 AM |
| Today at 11:20:55 AM |
| Today at 11:19:48 AM |

In the *write\_data\_by\_product\_line()* change the line of code that selects records from the pandas dataframe to a different valid value, for example, *Outdoor Protection*. The number of records for this filter will be different. Save the script and run it.

```
@task
def write_data_by_product_line(filteredData):
    # Select any product line - we will write it to a separate file
    campingEquipment = filteredData.loc[filteredData['Product line'] == 'Outdoor Protection']
```

Next, change the value to an invalid value, for example, *Camping Equipment1*. The number of records should be 0. Save the script and run it.

```
@task
def write_data_by_product_line(filteredData):
    # Select any product line - we will write it to a separate file
    campingEquipment = filteredData.loc[filteredData['Product line'] == 'Camping Equipment1']
```

25. Switch to Databand and find the generated alerts.

| Origin                   |   |  |
|--------------------------|---|--|
| <input type="checkbox"/> | <b>HIGH</b> Task custom metric<br>Number of rows for camping equipment sales ><br>Trigger value: 0<br>Today at 04:57:14 PM <input type="checkbox"/> Retail Analytics EL DBND <span style="color: red;">●</span> Triggered | <a href="#">prepare_sales_data_EL1 &gt;</a><br><a href="#">writeDataByProductLine &gt;</a> |
| <input type="checkbox"/> | <b>HIGH</b> Task custom metric<br>Number of rows for camping equipment sales ><br>Trigger value: 0<br>Today at 04:57:14 PM <input type="checkbox"/> Retail Analytics EL DBND <span style="color: red;">●</span> Triggered | <a href="#">prepare_sales_data_EL1 &gt;</a><br><a href="#">writeDataByProductLine &gt;</a> |

26. Investigate the issue through **Datasets** view.

Databand

an IBM Company

0.89.8

Dashboard

Pipelines

→

Runs

Alerts

999+

Datasets

Search

Show

Search

All datasets

74 Datasets

| <input type="checkbox"/> | Name  | Total Records                 |
|--------------------------|---|-------------------------------|
| <input type="checkbox"/> | <div><div></div>Golf_Equipment.csv</div>                | <div><div></div>No Data</div> |
| <input type="checkbox"/> | <div><div></div>Camping_Equipment.csv</div>             | <div><div></div>No Data</div> |
| <input type="checkbox"/> | <div><div></div>Filtered_df</div>                       | <div><div></div>No Data</div> |
| <input type="checkbox"/> | <div><div></div>Retail_Products_and_Customers.csv</div> | <div><div></div>No Data</div> |

Find the dataset, drill down and click on the **History** tab. Notice that the number of records written in the last 2 pipeline runs is different.

Databand

an IBM Company

0.89.8

Dashboard

Pipelines

Runs

Alerts

Datasets

Metrics

Integrations

Settings

Docs

Camping\_Equipment.csv

Dataset Type: local

Reported From: DBND

Dataset Path: local://weekly\_... ping\_Equipment.csv

First Synced: Yesterday at 10:30:29 AM

Last Synced: Today at 12:22:54 PM

Overview

History

Operations

Seen during: Last 7 days

Operation type: All

Pipeline: All

Issue type: All

24 Operations

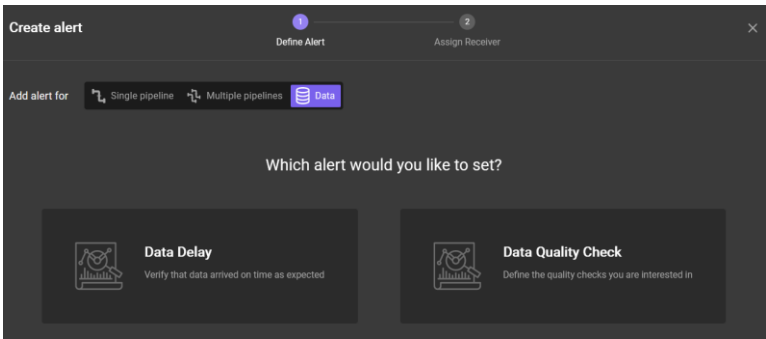
| Status                            | Type  | Issue type | Schema     | Records        | Origin   |
|-----------------------------------|-------|------------|------------|----------------|--|
| <div><div></div><div></div></div> | Write | No issues  | 21 columns | 0 records      | <div>🔗 prepare_sales_data EL</div> <div>→ weekly</div> <div>📄 write_data_by_product_line</div> |
| <div><div></div><div></div></div> | Write | No issues  | 21 columns | 8,620 records  | <div>🔗 prepare_sales_data EL</div> <div>→ weekly</div> <div>📄 write_data_by_product_line</div> |
| <div><div></div><div></div></div> | Write | No issues  | 21 columns | 24,866 records | <div>🔗 prepare_sales_data EL</div> <div>→ weekly</div> <div>📄 write_data_by_product_line</div> |

Next, we will add a *custom metric* to monitor data quality and set up an alert for it.

In Databand a *custom metric* is *business logic* that's implemented in a data pipeline and registered in Databand. Custom metrics can be used to monitor/alert for data quality or simply patterns in data that need to be investigated.

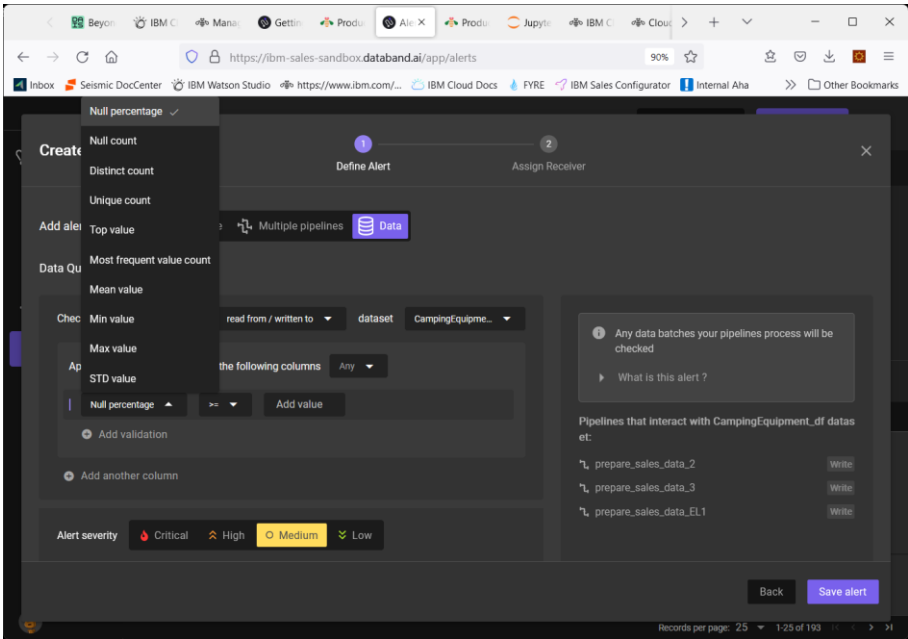
Let's start by reviewing the built-in metrics for data quality.

27. In Databand navigate to the **Alerts** page. Click **Add Alert -> Data** and select **Data Quality Check**.



28. Select the *Camping Equipment* dataset and explore quality alerts that you can set up with the options available in the UI.

Notice that we can select columns in the dropdown and set up alerts for *null* records, *min* and *max* values, *distinct counts*, etc.



While the built-in options cover the most important data quality checks, we may have use cases that require custom logic.

For example, in our retail use case we want to issue an alert if no sales have been reported from a particular state. In order to do this, we will need to create a custom metric in our data pipeline.

We added this function to our data pipeline:

```
def check_camping_equipment(rawData):

    metric_name = 'Sales from Alaska' + unique_suffix

    numberOfCampingEquipment_records = rawData['State'].tolist().count('Alaska')
    print(numberOfCampingEquipment_records)

    log_metric(metric_name, numberOfCampingEquipment_records)
```

In this example the `log_metric()` function sends the number of records for the specified state to Databand.

*Note: we hardcoded the state value for simplicity. In a production implementation, it's possible to make the values configurable so that states are changed without changing the code.*

29. Add the metric logging code to your `SimpleRetailDataPipeline_with_Databand` script or open `SimpleRetailDataPipeline_with_CustomMetric.py` script (from the `Workshop/Pipelines` folder)

If you're updating the original script, in addition to adding the function, add the function call to the `prepare_retail_data()` function (at the end).

```
def check_camping_equipment(rawData):

    metric_name = 'Sales from Alaska' + unique_suffix

    numberOfCampingEquipment_records = rawData['State'].tolist().count('Alaska')
    print(numberOfCampingEquipment_records)

    log_metric(metric_name, numberOfCampingEquipment_records)
```



```
def prepare_retail_data():
    with dbnd_tracking(
        conf={
            "core": {
                "databand_url": databand_url,
                "databand_access_token": databand_access_token,
            },
        },
        job_name="prepare_sales_data" + unique_suffix,
        run_name="weekly",
        project_name="Retail Analytics" + unique_suffix,
    ):
        # Call the step job - read data
        rawData = read_raw_data()

        # Filter data
        filteredData = filter_data(rawData)

        # Write data by product line
        write_data_by_product_line(filteredData)

        check_camping_equipment(rawData)

    print("Finished running the pipeline")
```

30. Run the updated script several times with different state values (keep the metric names the same).

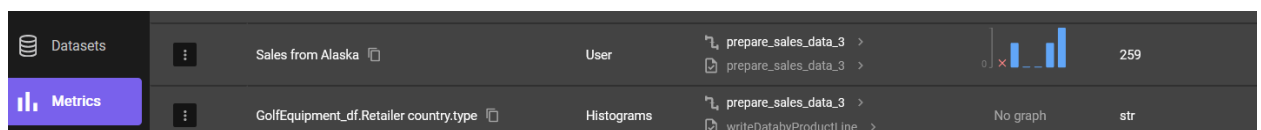
For example, filter for *Alaska* (no records), *Arizona*, and *Oregon*. We are providing different state names just to generate a different number of records in Databand.

```
def checkCampingEquipmentData(rawData):
    numberOfCampingEquipment_records = rawData['State'].tolist().count('Oregon')
    print(numberOfCampingEquipment_records)

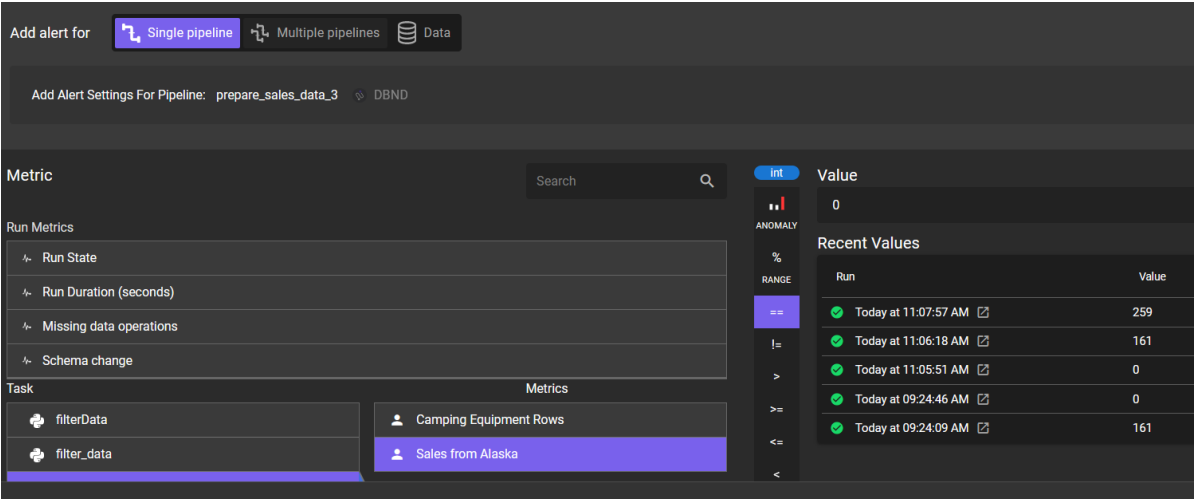
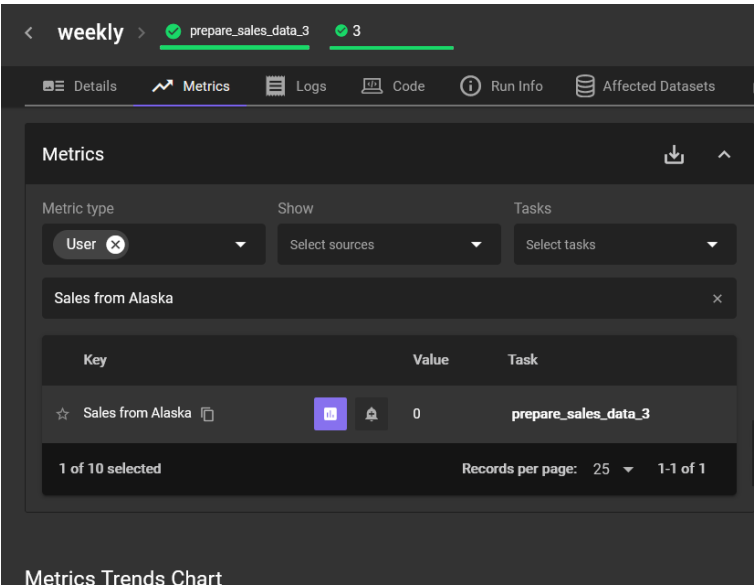
    log_metric('Sales from Alaska', numberOfCampingEquipment_records)
```

31. In Databand navigate to the **Metrics** page and find *Sales from Alaska* metric.

Notice the number of records recorded on each run.

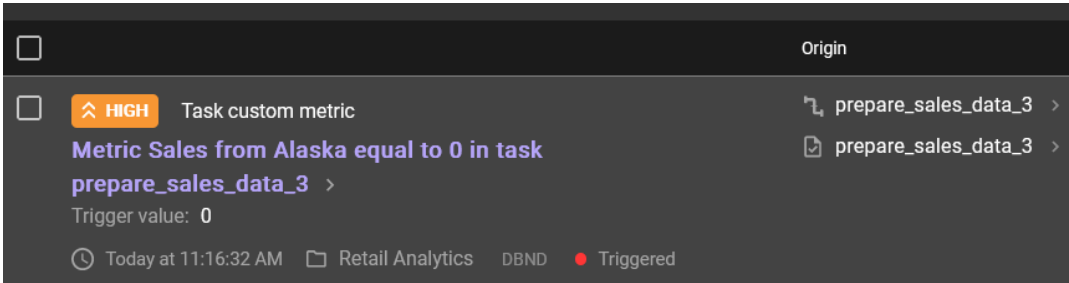


Drill down into the metric, then create an alert for number of records == 0.



32. Change the pipeline to count records from *Alaska* again and run it.

Switch to Databand and find the generated alert.



This concludes introduction to Databand Python SDK.

### Summary:

In this section you learned how to use Databand SDK to:

- Track Python pipelines in Databand
- Create alerts for data quality
- Create custom metrics.

These tasks are the most frequently used tasks in a production implementation of Databand.

## Part 2: Review a PySpark example

In this section you will use the Databand SDK to track execution of a Spark pipeline.

As you know, Cloud Pak for Data includes a Spark runtime. Many customers use other distributions of Spark, such as *Databricks*, *Amazon EMR*, and *Google Cloud DataProc*.











In this section we will review programmatic integration with Spark pipelines that's similar to Python integration.

**Important note:** If a customer is using an external (non-CPD) Spark, then they can enable integration on the Spark cluster level. If this integration is enabled, then Spark pipelines will be automatically monitored (no additional code is required). This applies to starting/stopping tracking and logging datasets, which means that integration with Spark is "no-code" integration. See [documentation](#) for configuration information.

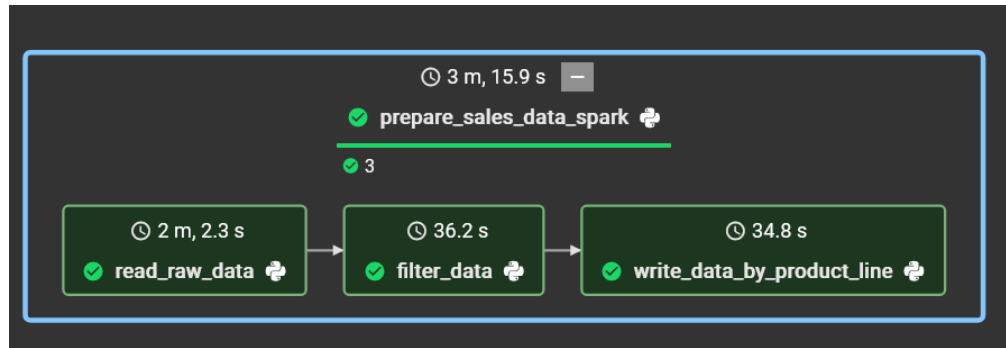
The PySpark example that we provided for this section is a PySpark notebook that performs ETL operations on the same retail dataset as Python example we reviewed in **Part 1**. We recommend that you use *Cloud Pak for Data as a Service (CPDaaS)* as the runtime environment.

1. Log in to CPDaaS and create a new project.
2. Import the *Retail\_Products\_and\_Customers.csv* from the */Pipelines* folder of the file downloaded from Box.
3. Create a notebook from file: *RetailDataPipeline\_Spark* (in the */Pipelines* folder). Make sure to set the Environment to *Default Spark 3.2 & Python 3.9 runtime*.

If you forget to select the Spark environment, you can do it later by stopping the notebook and changing the default environment from the Project view (select the vertical ellipses menu next to the notebook). To unlock the notebook, click on the lock icon.

| Notebooks   |                       |                                  |   |
|---|-----------------------|----------------------------------|---|
| Name  | Language              | Last modified                    |   |
|  <b>RetailDataPipeline_Spark</b><br>Notebook | Python 3.9 with Spark | 5 days ago<br>Elena Lowery (You) |  |
|  RetailDataPipeline_Spark_EL<br>Notebook     | Python 3.9 with Spark | 5 days ago                       |    |
|  Model_build_with_Databand<br>Notebook       | Python 3.9            | 5 days ago                       |    |
|  BankingTransactionsPrep<br>Notebook         | Python 3.9            | 5 days ago                       |    |
|  SimpleRetailDataPipeline<br>Notebook        | Python 3.9            | 5 days ago                       |    |

4. Follow the update instructions in the notebook:
  1. Generate the project token
  2. Replace code to read data from your object storage
  3. Provide Databand URL and token
  4. Rename the pipeline
5. Run the notebook and check Databand for pipeline status.



6. If you wish, introduce errors in the notebook.

You have finished **Part 2** of the lab.