

Deep Dive: Transforming your data with Code Repositories

What is Code Repositories in Foundry?

Code Repositories provides a web-based integrated development environment (IDE) for writing and collaborating on production-ready code in Foundry. With Code Repositories, data engineers can create efficient pipelines in bulk.

Example workflows that are a good fit for Code Repositories include:

- A daily pipeline at high data scale which requires incremental compute.
- A high-visibility pipeline with strict governance requirements to be able to revert to previous versions of historical code, or gate code changes on unit tests passing.

Learning objectives of the course

You will complete this course being able to create and manage code repositories, perform data transformations, and collaborate effectively.

By the end of this course, you will have

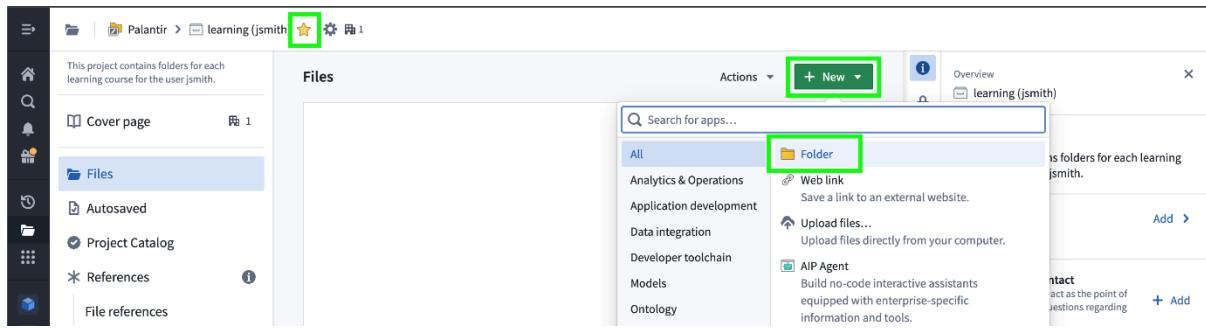
- created a PySpark Transform including casting and filtering,
- used joins and aggregations,
- worked collaboratively with other people using Branching,
- seen how Code Repositories interact with other Foundry tools like Data Lineage and Job Tracker.

Setting up your Project and Folder

Create a Course-Specific Training Folder

Step 1: Create your Folder

1. In the top left, click on the star to favorite your Project. This will allow you to find it quickly later on.
2. Click on **New**.
3. Click on **Folder**.



Step 2: Name your Folder

1. Name the new folder so that you can allocate it to the current learning course
 - o For example: Code Repo Training

This screenshot shows the same Palantir interface as the previous one, but now with a new folder named 'Code Repo Training' listed in the 'Files' table. The table has columns for 'NAME', 'LAST UPDATED', and 'TAGS'. The 'NAME' column shows a folder icon followed by the text 'Code Repo Training'. The 'LAST UPDATED' and 'TAGS' columns are currently empty.

Data Transformation

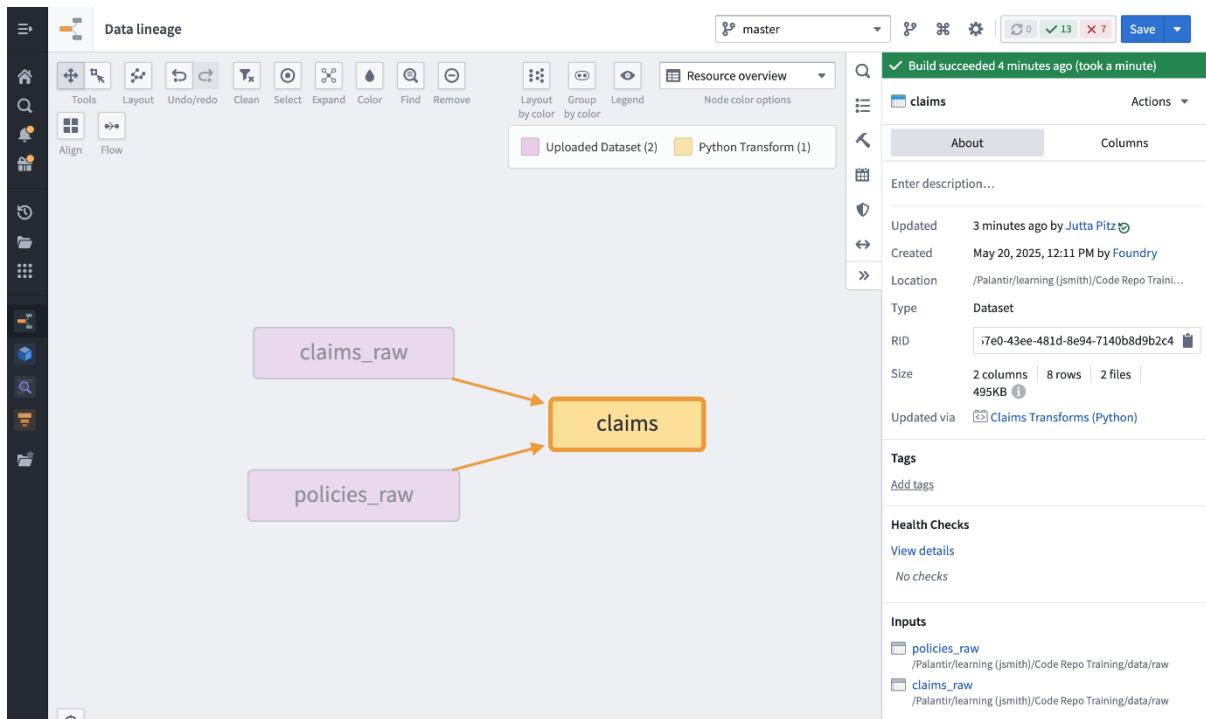
Introduction

In this section you will upload the data and create your Code Repository including the cleaning transformations to prepare your datasets. All the code you will be writing in this tutorial will be located in this repository.

Scenario

For this tutorial we assume the role of claims handler in a global insurance company. Your task is to present to the Chief Financial Officer the annual results of each line of business. Using code repository, you will process two main datasets, one containing each historic claim over the past years, and the other containing the corresponding policies, including the lines of business.

At the end of this section you will have implemented a cleaning pipeline using Python Transforms.



Download the Data Sources

Below are the different files you'll need to download and then upload into Foundry. Click the names of the files below to download them to your device.

- [claims_raw.csv](#): A raw csv file containing all claims submitted by insurance customers in the years 2023 and 2024.
- [policies_raw.csv](#): A raw csv file containing all policies the insurance has held in the years 2023 and 2024 together with the allocation to the respective line of business.

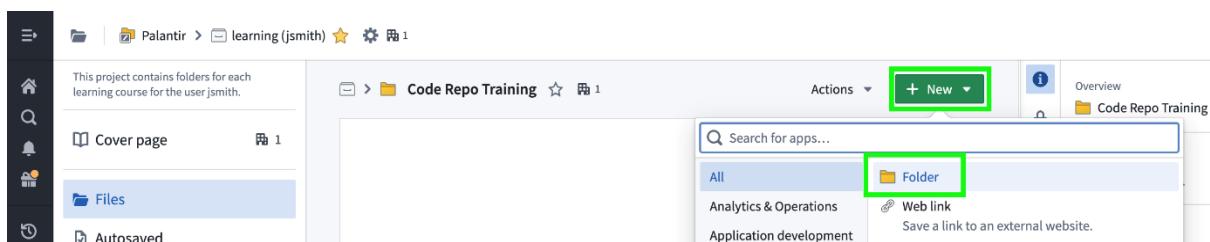
Important Downloading Tip: Your browser might open the files themselves instead of downloading them, use Ctrl + S or cmd + S to save the file.

Upload the Raw Datasets

In this section, you will upload your downloaded datasets.

Step 1: Create subfolders

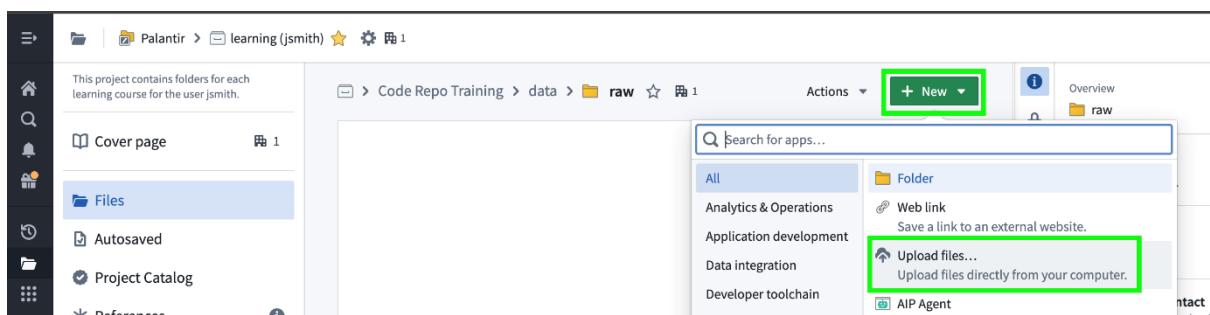
1. Open the training folder you created above: Code Repo Training
2. Create a new dedicated folder for your input datasets.
 1. Click **+New** in your project folder.
 2. Choose **Folder**.



3. Name the folder **data**.
3. Within the **data** folder, repeat the previous steps:
 1. Click **+New** in your **data** folder.
 2. Choose **Folder**.
 3. Name the folder "**raw**".

Step 2: Upload your datasets into the raw folder

1. Open the new **raw** folder.
 1. Click **+New**.
 2. Choose "**Upload files...**" in the dropdown.

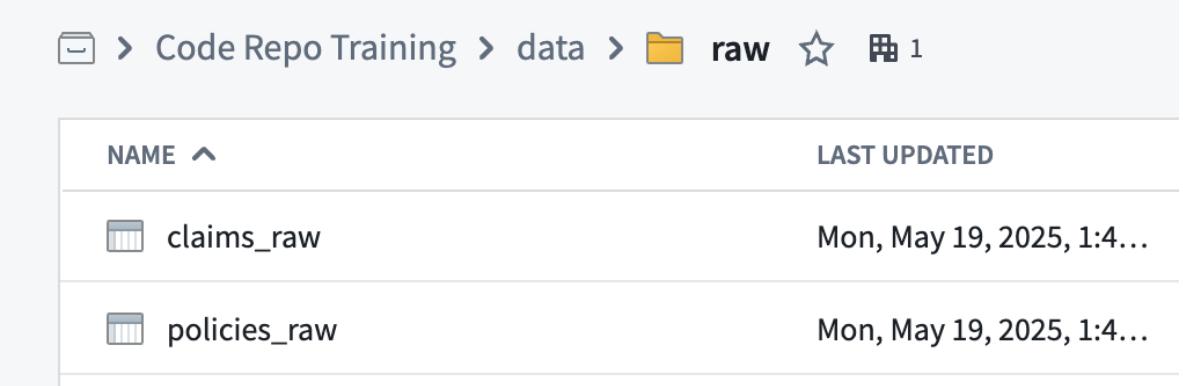


3. Choose the first file from your computer: *instructor_5lgholhrburismcba795c5iu_public_1747647595_claims_raw+(3)*
4. Rename this to **claims_raw.csv**
5. Select “**Upload as a structured dataset (recommended)**”.
6. Verify and tick confirmation checkboxes where required.
7. Click **Upload**.

The screenshot shows a 'Upload files' dialog box with the following details:

- Title:** Upload files
- Section:** ✓ Upload Restriction - Swirl
- Text:** Data uploaded must be notional or open source. No Sensitive, Restricted or Highly Restricted data is allowed on this stack; which includes customer data. Details around Palantir's data classification policy can be found [here](#). If you are not sure, please reach out to pcl@palantir.com.
- Note:** If you ingest unauthorized data by mistake, please immediately reach out to dataphone@palantir.com per the [Compliance Wiki SOP](#). Sensitive, Restricted, and Highly Restricted data categories include (but are not limited to): personally identifiable information (PII), Personal Health Information (PHI), customer data and classified data.
- Checkboxes:** I agree that the imported files conform to the policy.
- Drop Area:** A dashed box with a cloud icon containing an upward arrow, labeled "Drop files here or choose from your computer".
- File List:**
 - File:** claims_raw.csv **Size:** 138.93 KB **X**
 - Options:**
 - Selected:** **Upload as a structured dataset (recommended)**
Datasets are the most basic representation of tabular data. They can be used and transformed by many different applications. [Documentation ↗](#)
 - Upload to a new media set**
Media sets enable media-specific capabilities for media files (e.g. audio, imagery, video, and documents). [Documentation ↗](#)
 - Upload to a new unstructured dataset**
Unstructured datasets can store arbitrary files for processing and analysis. Structured data can be extracted from unstructured datasets using Pipeline Builder or Transforms. [Documentation ↗](#)
 - Upload as a raw file**
Raw files cannot be used in data pipelines, analyses, or models.
- Buttons:** A blue 'Upload' button at the bottom right.

2. Repeat this process with the second file: *instructor_5lgholhrburismcba795c5iu_public_1747647748_policies_raw+(1)*
 1. In the process, rename this file to **policies_raw.csv**
3. You should now see the following two datasets in your **raw** folder.
 1. If your datasets still have the long complex name, select them and rename them to **claims_raw** and **policies_raw**, respectively.

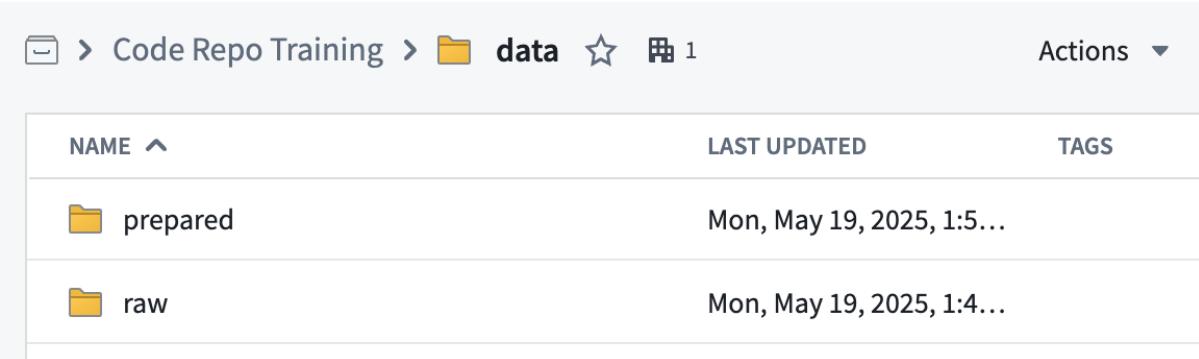


The screenshot shows a file browser interface with the following path: Code Repo Training > data > raw. The raw folder contains two files: claims_raw and policies_raw, both last updated on Mon, May 19, 2025, 1:4... The files are listed in descending order by name.

NAME	LAST UPDATED
claims_raw	Mon, May 19, 2025, 1:4...
policies_raw	Mon, May 19, 2025, 1:4...

Step 3: Create the dedicated folder for your output datasets

1. Open the **data** folder, which is the parent folder to your raw folder.
2. Within the **data** folder:
 1. Click **+New** in your project folder.
 2. Choose **Folder**.
 3. Name the folder “**prepared**”.
 4. Your data folder should now contain two subfolders like shown below:



The screenshot shows a file browser interface with the following path: Code Repo Training > data. The data folder contains two subfolders: prepared and raw. The prepared folder was created in step 3. Both subfolders were last updated on Mon, May 19, 2025, 1:5... The raw folder is also present.

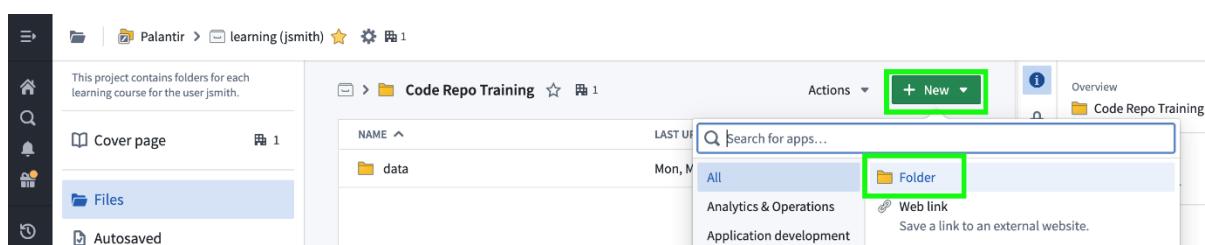
NAME	LAST UPDATED	TAGS
prepared	Mon, May 19, 2025, 1:5...	
raw	Mon, May 19, 2025, 1:4...	

Create a New Repository

In your training folder Code Repo Training, you will first create a dedicated folder called **logic** where your Code Repository will live, followed by the Code Repository itself.

Step 1: Create a subfolder for your repository

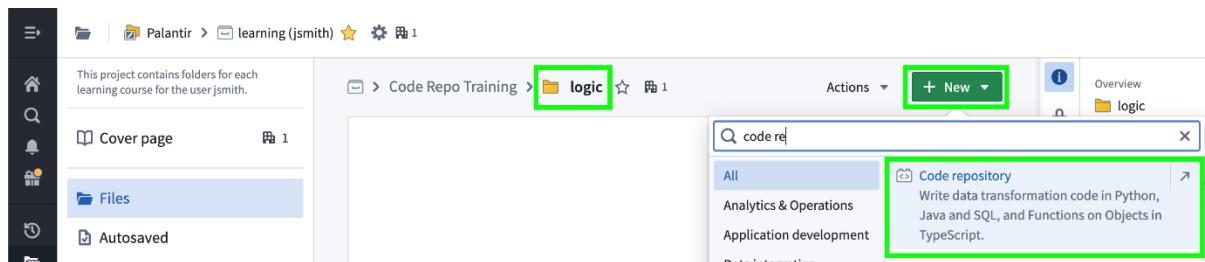
1. Navigate to your training folder. This folder should currently contain one folder called **data**.
2. Create a dedicated folder for your repo called "**logic**".
 1. click **+New** in your training folder.
 2. Choose **Folder**.



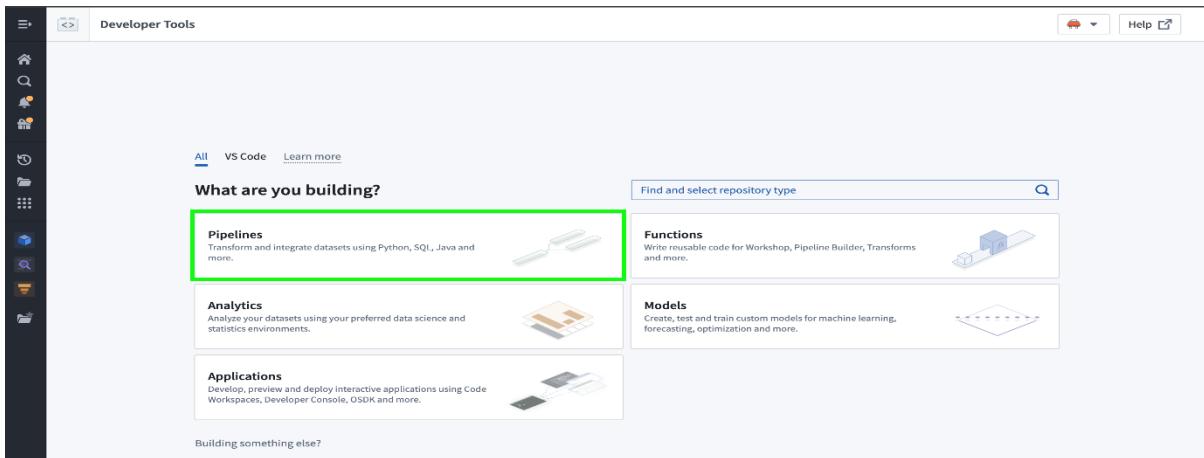
3. Name the folder "**logic**".
4. Open the new **logic** folder.

Step 2: Create your repository

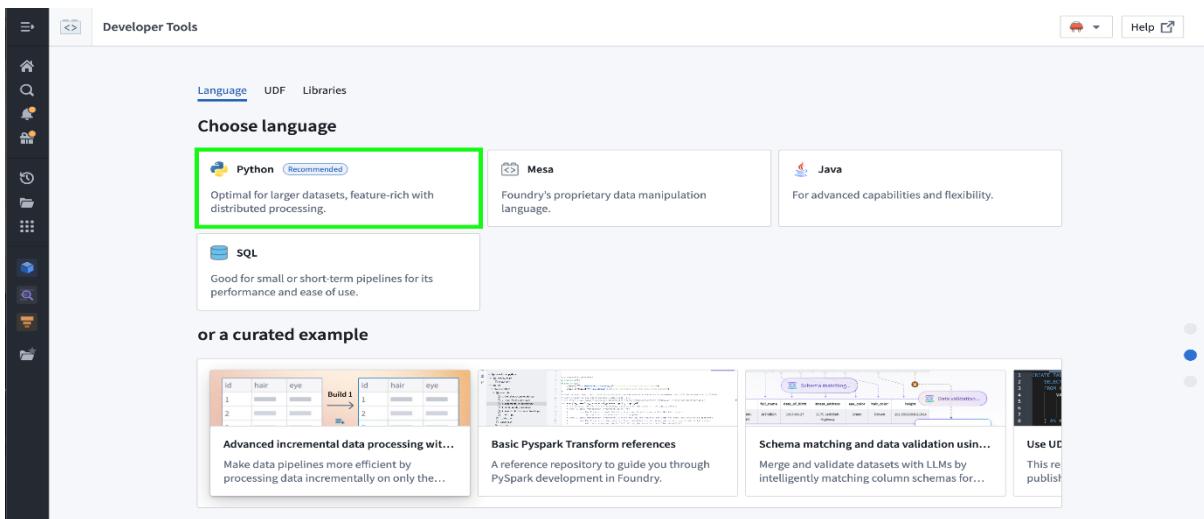
1. Click on **+ New** and choose **Code Repository** from the drop-down list.



2. Choose **Pipelines**.



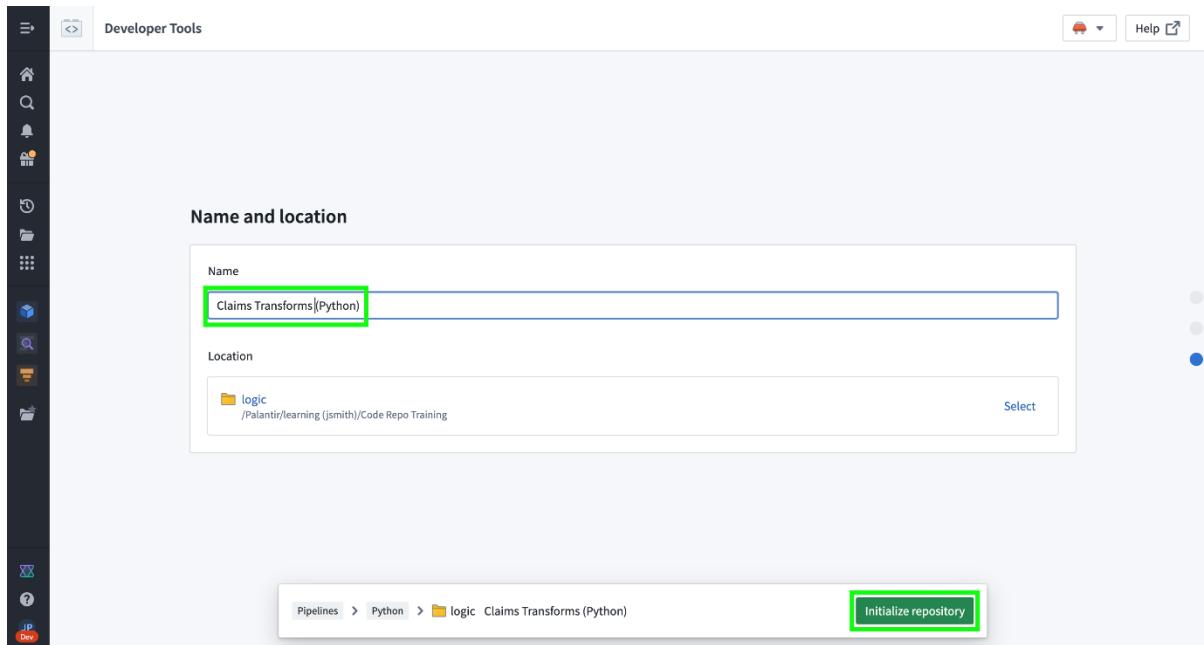
3. Choose Python.



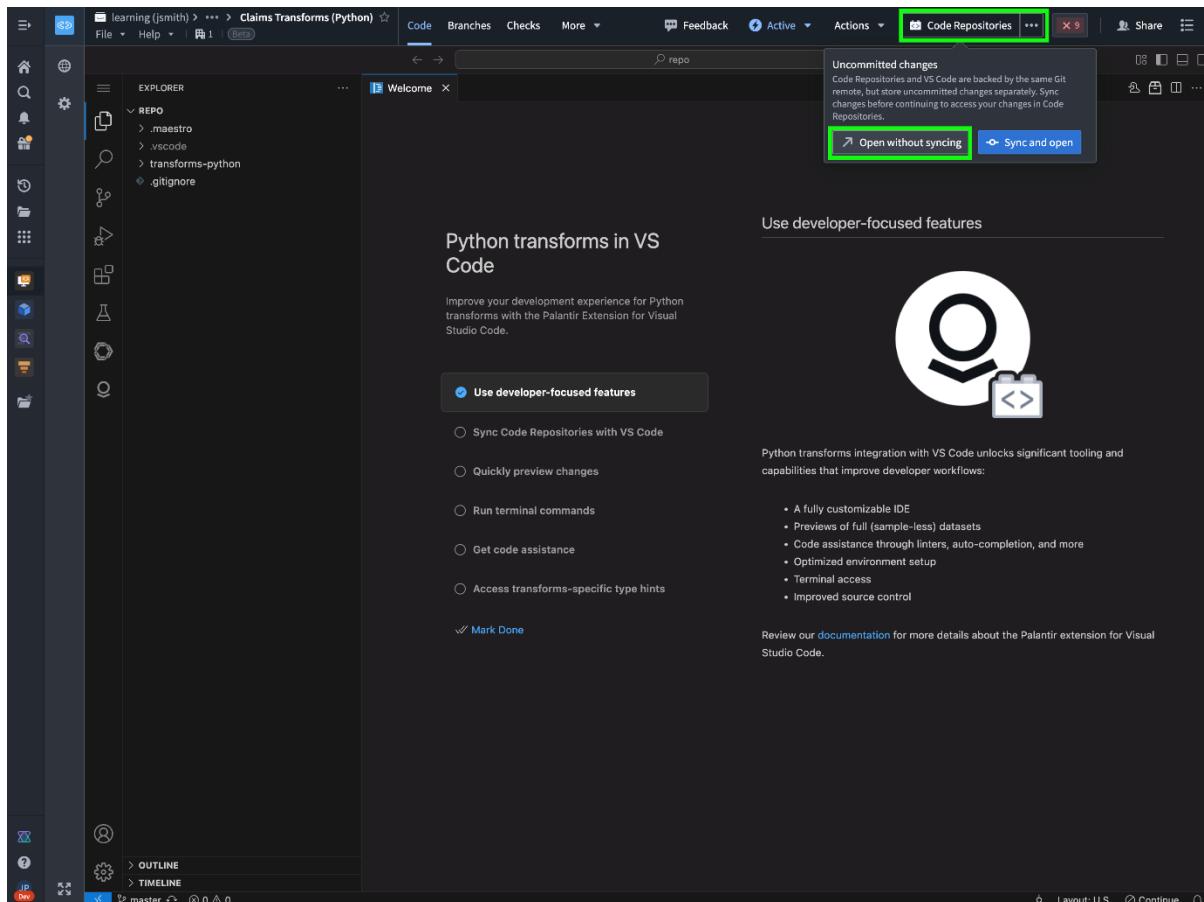
4. Name your repository Claims Transforms (Python).

5. Leave the location as is (this should be your logic folder created above).

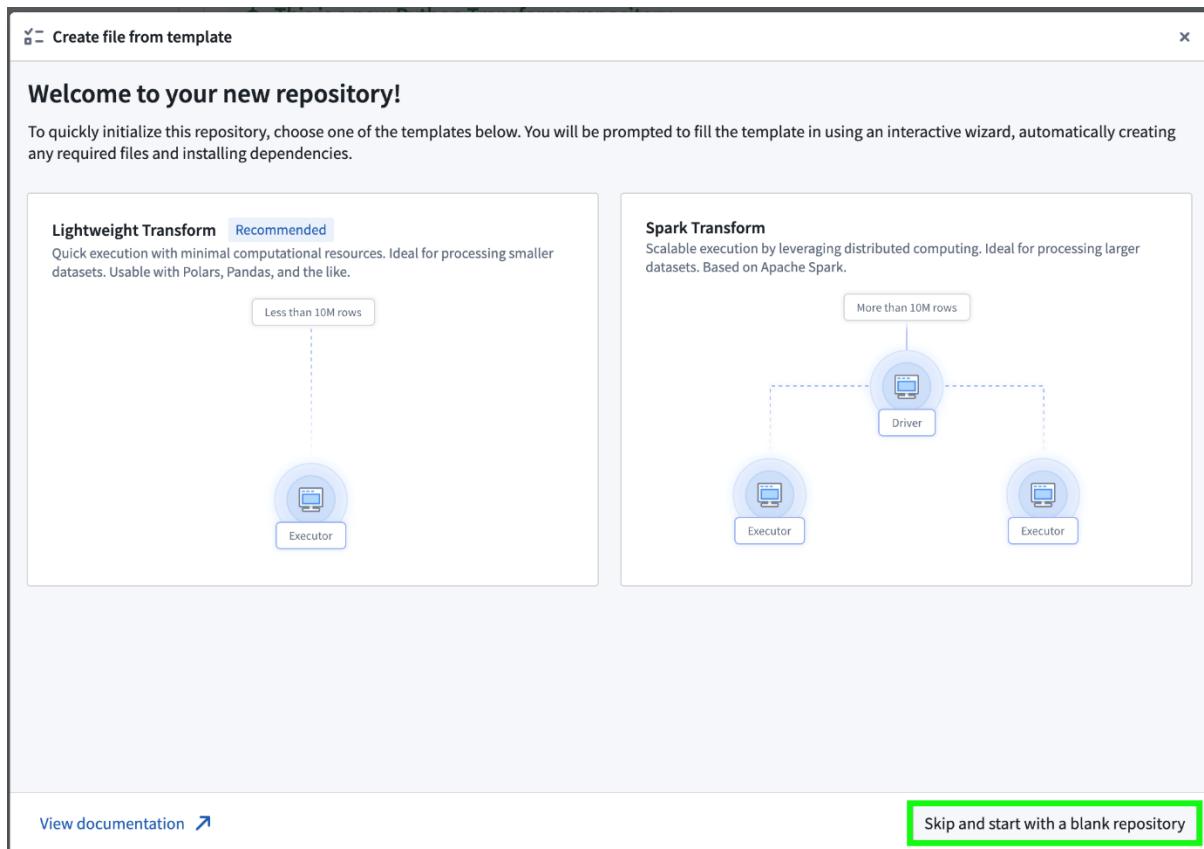
6. Click **Initialize repository**.



7. Sometimes, at this point, Foundry informs you about Code Repositories news. You can ignore this here and close a potential dialogue.
8. Depending on your Foundry instance, you might now land on the following Code Workspaces screen. If you do, click on **Code Repositories** in the top and then **Open without syncing**.



9. Click **Skip and start with a blank repository** in the next screen:



Your code repository is now set up and open.

learning (jsmith) > *** > Claims Transforms (Python) ★

Code Branches Checks More VS Code Preview Test Commit Build Propose changes

Branch master

Files Add

Search file names... transforms-python/README.md

This is a new Python Transforms repository This repository recommends visiting the examples.py file to get started Go to examples.py

Python Transforms in Foundry

Overview

Python is the most comprehensive language for authoring data transformations from within Foundry. Python Transforms include support for batch and incremental pipelines, creating and sharing reusable code libraries, and defining data expectations to ensure high quality data pipelines.

To get started, open [transforms-python/src/myproject/datasets/examples.py](#) and uncomment the example transform.

Local Development

It is possible to carry out high-speed, iterative development of Python Transforms locally. To get started, click the "Work locally" button in the top right.

Unit Testing

Unit testing is supported in Python transforms. Tests can be enabled by applying the `com.palantir.transforms.lang.pytest-defaults` Gradle plugin in the Python project [transforms-python/build.gradle](#).

Data Expectations

Data Expectations can be set up in a Python transforms repository. To get started, open the library search panel on the left side of your Code Repository, search for `transforms-expectations`, and click on "Add library" within the library tab.

Foundry Explorer Problems Debugger Preview Tests File Changes Build Docs Task Runner SQL Terminal

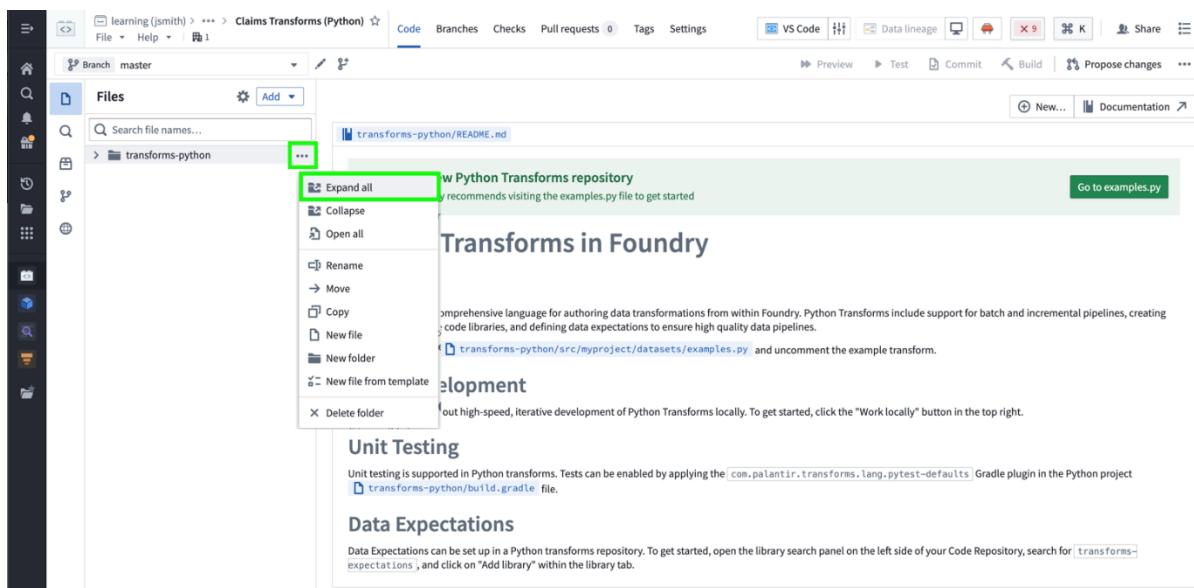
AIP autocomplete Project scoped Files saved Checks passed

Create Your First Transform

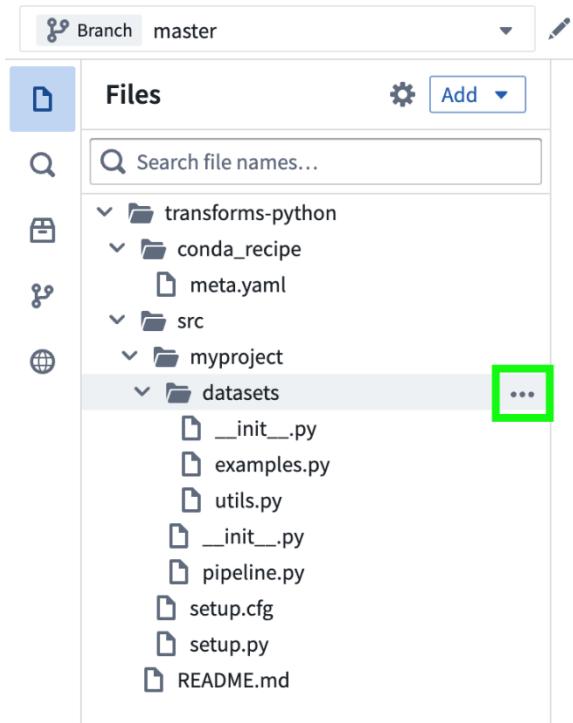
In this step, you will create your first data transformation using Code Repositories. You will add the claims_raw dataset as your input and create an identity transform. You will preview your dataset at the end of this lesson.

Step 1: Create a new Code Repository file

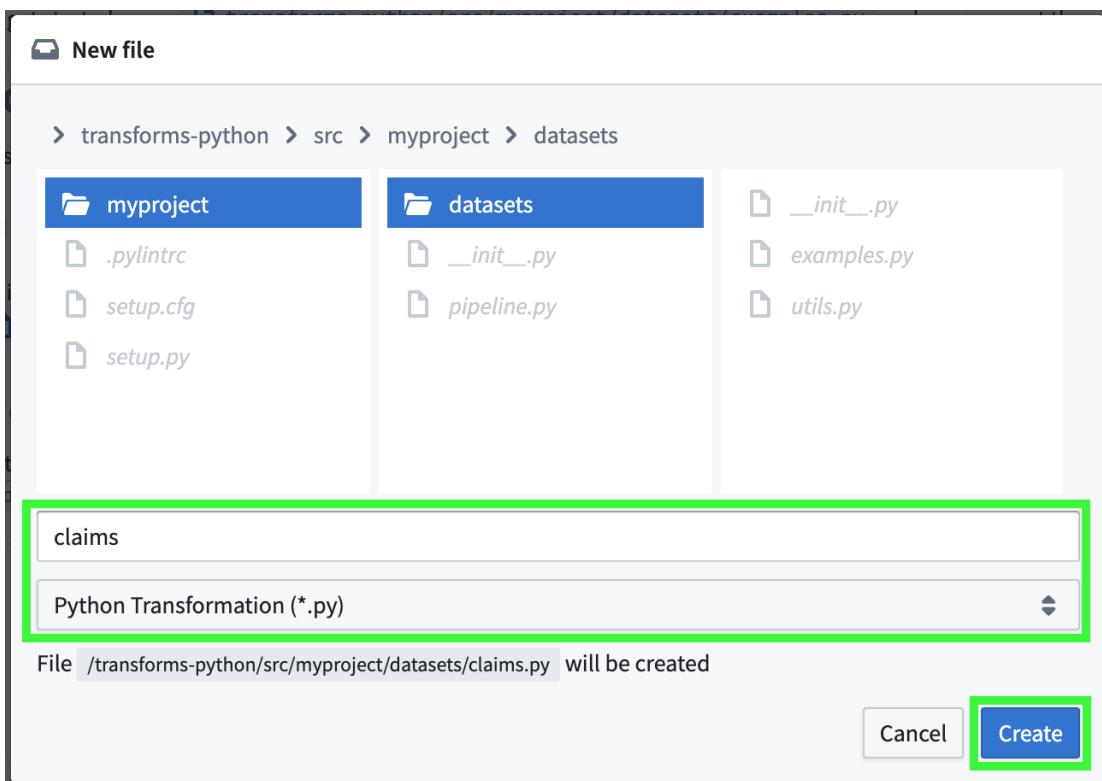
1. If the file structure on the left hand side is not yet expanded, hover over the **transforms-python** line, click the **three dots** and choose **Expand all**.



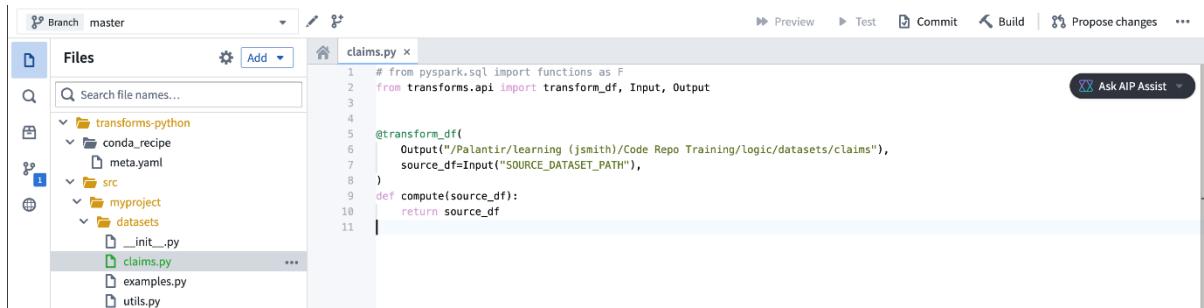
2. Hover over the **datasets** line in the file directory
3. Click the **three dots**.



4. Select **New File**.
5. Name the file “claims”.
6. Choose the **Python Transformation(*.py)** ending.
7. Click **Create**.



Code Repository will have created a simple Python file with an example input and output file.



The screenshot shows a code repository interface with a 'Branch master' dropdown. The main area displays a file named 'claims.py' with the following content:

```
# from pyspark.sql import functions as F
from transforms.api import transform_df, Input, Output

@transform_df(
    Output("/Palantir/learning (jsmith)/Code Repo Training/logic/datasets/claims"),
    source_df=Input("SOURCE_DATASET_PATH"),
)
def compute(source_df):
    return source_df
```

Step 2: Define your input and output datasets

Below is a comparison of your code before and after the instructions laid out in this step.

Before this step:



The screenshot shows a code editor with the 'claims.py' file open. The code is identical to the one shown in the previous screenshot:

```
# from pyspark.sql import functions as F
from transforms.api import transform_df, Input, Output

@transform_df(
    Output("/Palantir/learning (jsmith)/Code Repo Training/logic/datasets/claims"),
    source_df=Input("SOURCE_DATASET_PATH"),
)
def compute(source_df):
    return source_df
```

```
# from pyspark.sql import functions as F
from transforms.api import transform_df, Input, Output
```

```
@transform_df(
    Output("/Palantir/learning (jsmith)/Code Repo Training/logic/datasets/claims"),
    source_df=Input("SOURCE_DATASET_PATH"),
)

def compute(source_df):
    return source_df
```

After this step:

```

1 # from pyspark.sql import functions as F
2 from transforms.api import transform_df, Input, Output
3
4
5 @transform_df(
6     Output("/Palantir/learning (jsmith)/Code Repo Training/data/prepared/claims"),
7     source_df=Input("claims_raw")
8 )
9 def compute(source_df):
10     return source_df
11

```

```

# from pyspark.sql import functions as F

from transforms.api import transform_df, Input, Output

@transform_df(
    Output("/Palantir/learning (jsmith)/Code Repo Training/data/prepared/claims"),
    source_df=Input("/Palantir/learning (jsmith)/Code Repo Training/data/raw/claims_raw"),
)

def compute(source_df):
    return source_df

```

1. First, we need to define the **input dataset** in our file.

- In line 7, replace SOURCE_DATASET_PATH with the path of your claims input dataset, previously saved in your raw folder.

1. Open the **claims_raw** dataset in a new tab

1. Hold cmd/ctrl and click the **Code Repo Training** folder in your file path at the top



2. Click on **data**

3. Click on **raw**

4. Click on **claims_raw**

2. copy the **Location**

Preview History Details Health Compare

claims_raw

About Columns

Enter description...

Updated Yesterday at 1:47 PM by Jutta Pitz

Created Yesterday at 1:47 PM by Jutta Pitz

Location /Palantir/learning (jsmith)/Code .. 

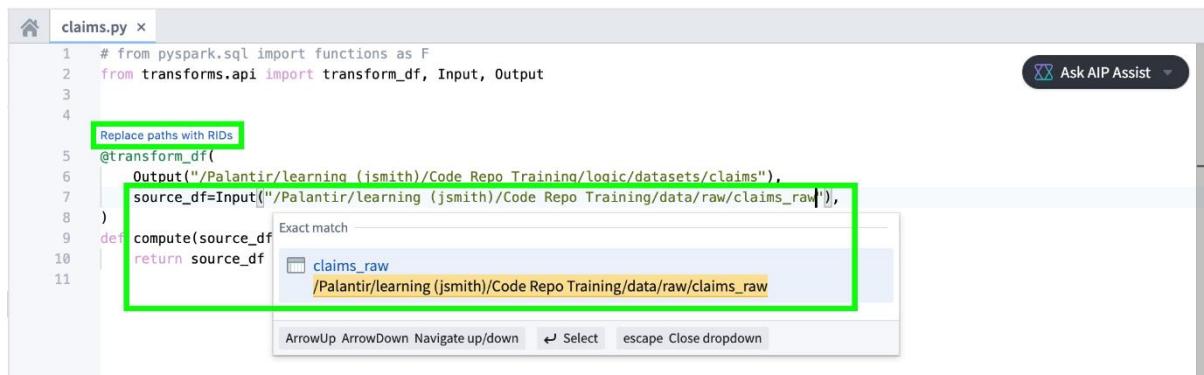
Type Raw dataset

RID 3-4ee2-b898-0d4084c57ee5 

Size 8 columns | 1 file | 139KB  [Calculate row count](#)

Updated via File imports • Import • Edit Schema

3. paste the path in your code as INPUT



```

1 # from pyspark.sql import functions as F
2 from transforms.api import transform_df, Input, Output
3
4 Replace paths with RIDs
5 @transform_df(
6     Output("/Palantir/learning (jsmith)/Code Repo Training/logic/datasets/claims"),
7     source_df=Input("/Palantir/learning (jsmith)/Code Repo Training/data/raw/claims_raw"),
8 )
9 def compute(source_df
10     return source_df
11

```

The screenshot shows a code editor window with a dropdown menu open at line 7. The menu is titled "Exact match" and contains a single item: "claims_raw /Palantir/learning (jsmith)/Code Repo Training/data/raw/claims_raw". The entire dropdown menu area is highlighted with a green box.

4. Foundry suggests to you the correct file to select

5. Click Replace paths with RIDs if this shows.

2. Second, we define our output dataset.

- Copy the path to your **prepared** folder which you created above.

The screenshot shows the Palantir Foundry interface. On the left is a sidebar with icons for Home, Search, Notifications, Project Catalog, References, Autosaved, and Trash. The main area has a breadcrumb navigation path: Palantir > learning (jsmith) > Code Repo Training > data. A file browser window is open, showing a folder named 'prepared' containing a subfolder 'raw'. The 'prepared' folder was created on Mon, May 19, 2025, 1:50:2... and modified on Mon, May 19, 2025, 1:49:3.... To the right is a detailed view of the 'prepared' dataset. It includes sections for Overview (with a yellow folder icon labeled 'prepared'), Description (with a placeholder 'Enter description...'), Documentation (with a note 'No documentation'), Project point of contact (with a note 'Add a user or group to act as the point of contact for issues or questions regarding this folder.'), and Metadata. The Metadata section shows the RID as b-b49b-4ed367eb400a, the Location as /Palantir/learning (jsmith)/Code Repo Training/data/prepared, and a button 'Copy full path to clipboard' which is highlighted with a green box. Other metadata fields include Space (Palantir), Portfolio (No portfolio), and Created (May 19, 2025, 1:50 PM by [user]).

- In line 6 of your claims.py file, paste the copied path in the Output("") brackets inbetween the quotes ""
- Extend the path by a /claims. This will be your newly created dataset originating from this transform.
- Your code file should currently look as follows:

The screenshot shows the PyCharm IDE with an open file named 'claims.py'. The code is as follows:

```

1 # from pyspark.sql import functions as F
2 from transforms.api import transform_df, Input, Output
3
4
5 @transform_df(
6     Output("/Palantir/learning (jsmith)/Code Repo Training/data/prepared/claims"),
7     source_df=Input("claims_raw").dataset.b5e0d3a7-edfb-4ee2-b898-0d4084c57ee5"),
8 )
9 def compute(source_df):
10     return source_df
11

```

The top bar of the IDE shows buttons for Preview, Test, Commit, Build, Propose changes, and more. There is also an 'Ask AIP Assist' button.

Step 3: Preview your dataset

1. Click the >> Preview button in the top bar to view your output dataset.
2. Preview produces a sample output without committing changes, running checks, or materializing any datasets in Foundry. Computing preview of output may take a few moments.

```

1 # from pyspark.sql import functions as F
2 from transforms.api import transform_df, Input, Output
3
4 @transform_df
5 def Output(*claims):
6     source_df=Input(*claims)
7     source_df=source_df.withColumn("date", F.col("date").cast("date"))
8     def compute(source_df):
9         return source_df
10

```

claim_id	policy_id	date	claim_value	is_accepted	state	zip_code
1 d0a02bf9-0344-4522-9 e6ec03ab-df55-4f1b-0	##2024-03-01###	9737.34	true	Maine	24031	
2 f36cdff8-ed7f-4389-8 e6ec03ab-df55-4f1b-0	##2023-08-01###	5305.21	true	Indiana	60869-7377	
3 3652fbf-659f-4dab-9 e5ef532-4419-45da-0	##2024-06-01###	3501.11	false	Kentucky	12312	
4 13de3470-6899-469f-8 e5ef532-4419-45da-0	##2024-04-01###	4685.67	false	South Dakota	00746-4018	
5 114949aa-f314-4f61-8 e5ef532-4419-45da-0	##2024-05-01###	2838.00	false	Mississippi	77202-7516	
6 a45b3ddd-365b-497f-9 e5ef532-4419-45da-0	##2023-09-01###	4759.99	true	South Dakota	46696	
7 3b817004-a7e8-463c-8 e5ef532-4419-45da-0	##2023-02-01###	7673.25	false	Florida		
8 6f07b00f-e9dc-4d50-a 3f43e4b5-40a7-4936-8	##2024-04-01###	3425.84	false	Rhode Island		
9 b48feb16-98a4-4de2-8 da3332d8-5fc4-43a9-0	##2023-01-01###	888.68	false	Arkansas	11116-3892	

3. Your output dataset is currently simply a copy of your input dataset, because - as you can see in the code - the compute function currently returns the source as a result. We will change this to a more meaningful transformation in the next steps
4. Please note that we currently have a data quality issue in our input dataset: the values in the **'date'** column are surrounded by '###' and hence cannot be stored as a date type. We'll clean this in the next lesson.

Clean Your Dataset

In this lesson, you will clean your input dataset by applying a simple cast and filter transformation and output a transformed dataset.

As a first transformation, we want to cast the date column from **String** to **Date**. In addition, we will filter by the Boolean column **is_accepted**.

Here is a comparison of what your code looks like right now and once you have run through these instructions.

Before applying these instructions:

The screenshot shows a code editor window with a tab labeled 'claims.py'. The code in the editor is:

```
1 # from pyspark.sql import functions as F
2 from transforms.api import transform_df, Input, Output
3
4
5 @transform_df(
6     Output("/Palantir/learning (jsmith)/Code Repo Training/data/prepared/claims"),
7     source_df=Input("claims_raw")
8 )
9 def compute(source_df):
10     return source_df
11
```

```
# from pyspark.sql import functions as F
from transforms.api import transform_df, Input, Output

@transform_df(
    Output("ri.foundry.main.dataset.610357e0-43ee-481d-8e94-7140b8d9b2c4"),
    source_df=Input("ri.foundry.main.dataset.b5e0d3a7-edfb-4ee2-b898-0d4084c57ee5"),
)
def compute(source_df):
    return source_df
```

After applying the following instructions:

The screenshot shows a code editor window with a tab labeled 'claims.py'. The code in the editor is:

```
1 from pyspark.sql import functions as F
2 from pyspark.sql import types as T
3 from transforms.api import transform_df, Input, Output
4
5
6 @transform_df(
7     Output("claims_main.dataset.610357e0-43ee-481d-8e94-7140b8d9b2c4"),
8     claims_raw=Input("claims_raw"),
9 )
10 def compute(claims_raw):
11     # Cast the "date" column from string to date
12     claims = claims_raw.withColumn("date", F.regexp_replace("date", "###", "")).cast(T.DateType())
13
14     # Filter out declined claims
15     claims = claims.filter(F.col("is_accepted") == True)
16
17     return claims
17
```

```
from pyspark.sql import functions as F
from pyspark.sql import types as T
from transforms.api import transform_df, Input, Output
```

```

@transform_df(
    Output("ri.foundry.main.dataset.610357e0-43ee-481d-8e94-7140b8d9b2c4"),
    claims_raw=Input("ri.foundry.main.dataset.b5e0d3a7-edfb-4ee2-b898-0d4084c57ee5"),
)

def compute(claims_raw):
    # Cast the "date" column from string to date

    claims = claims_raw.withColumn("date", F regexp_replace("date", "###",
    "")).cast(T.DateType()))

    # Filter out declined claims

    claims = claims.filter(F.col("is_accepted") == True)

    return claims

```

Step 1: Import functions and types

1. Uncomment the first line of your code repository, i.e. remove the “#” in line 1. This makes the Pyspark SQL library available for us to use in our transformation code.

```
# from pyspark.sql import functions as F
```

2. Include the following additional line of code right underneath the code above:

```
from pyspark.sql import types as T
```

3. Your first three lines of code should now look like this:



```

claims.py ×
1  from pyspark.sql import functions as F
2  from pyspark.sql import types as T
3  from transforms.api import transform_df, Input, Output

```

Step 2: Cast the previous string column to a date column

We want to clean up the date column by casting the string to a Date. We will use pyspark functions such as `withColumn` or `cast`. See the [official Spark documentation](#) for context and an overview of which functions are available.

Modify the following sections in the code file. You can copy and paste the code given in the steps below:

```
claims.py x
1 from pyspark.sql import functions as F
2 from pyspark.sql import types as T
3 from transforms.api import transform_df, Input, Output
4
5
6 @transform_df(
7     Output("claims", main.dataset.610357e0-43ee-481d-8e94-7140b8d9b2c4),
8     claims_raw=Input("claims_raw", dataset.b5e0d3a7-edfb-4ee2-b898-0d4084c57ee5),
9 )
10 def compute(claims_raw):
11     # Cast the "date" column from string to date
12     claims = claims_raw.withColumn("date", F regexp_replace("date", "###", "")).cast(T.DateType())
13
14 return claims
```

1. In line 8, replace source_df with claims_raw.
2. The previous lines 10 and 11 become:

```
def compute(claims_raw):
    # Cast the "date" column from string to date
    claims = claims_raw.withColumn("date", F regexp_replace("date", "###",
    "")).cast(T.DateType()))
    return claims
```

3. Preview your results to check whether this has worked as expected and that your date column is now of type **Date** as opposed to type **String**.

- Click **Preview** at the top of your Code Repository, as done before.
- Once the table shows, verify that the **date** column is indeed of type **Date** and that the trailing **##**s have been removed.

	claim_id	policy_id	date	claim_value
	String	String	Date	Double
1	d0a020f9-0344-4522-9 e6ec03ab-df55-4f1b-9	e6ec03ab-df55-4f1b-9	2024-03-01	
2	f36cdfd8-ed7f-4389-8 e6ec03ab-df55-4f1b-9	e6ec03ab-df55-4f1b-9	2023-08-01	
3	3652f9bf-659f-4da8-9 e5efa532-4419-45da-a	e5efa532-4419-45da-a	2024-06-01	
4	13de3470-6899-469f-8 e5efa532-4419-45da-a	e5efa532-4419-45da-a	2024-04-01	
5	114a940a-f314-4f61-8 e5efa532-4419-45da-a	e5efa532-4419-45da-a	2024-05-01	
6	a45b3ddd-365b-497f-9 e5efa532-4419-45da-a	e5efa532-4419-45da-a	2023-09-01	
7	3b817004-a7e8-463c-8 e5efa532-4419-45da-a	e5efa532-4419-45da-a	2023-02-01	
8	6f07b00f-e9dc-4d50-a 3f43e4b5-40a7-4936-8	3f43e4b5-40a7-4936-8	2024-04-01	
9	b48feb16-90a4-4de2-8 da3332d8-5fc4-43a9-a	da3332d8-5fc4-43a9-a	2023-01-01	

Step 3: Filter your dataset

Next, we want to filter the claims dataset to only those claims that have been accepted for payout.

1. Modify the Python function in the **claims.py** file as follows:

- o Extend the previous code in lines 10-13 to this code:

```
def compute(claims_raw):  
  
    # Cast the "date" column from string to date  
  
    claims = claims_raw.withColumn("date", F regexp_replace("date", "###",  
    "")).cast(T.DateType()))  
  
  
    # Filter out declined claims  
  
    claims = claims.filter(F.col("is_accepted") == True)  
  
    return claims
```



```
claims.py x  
1  from pyspark.sql import functions as F  
2  from pyspark.sql import types as T  
3  from transforms.api import transform_df, Input, Output  
4  
5  
6  @transform_df(  
7      Output("claims", main.dataset.610357e0-43ee-481d-8e94-7140b8d9b2c4"),  
8      claims_raw=Input("claims_raw", dataset.b5e0d3a7-edfb-4ee2-b898-0d4084c57ee5"),  
9  )  
10 def compute(claims_raw):  
11     # Cast the "date" column from string to date  
12     claims = claims_raw.withColumn("date", F regexp_replace("date", "###", "")).cast(T.DateType()))  
13  
14     # Filter out declined claims  
15     claims = claims.filter(F.col("is_accepted") == True)  
16  
17
```

2. Verify your results using Preview again

- o Click the **Preview** button in the top, just like previously
- o Confirm that only rows with *is_accepted* being *true* remain
 - Navigate to the *is_accepted* column in the Preview table at the bottom
 - Click the little triangle in the column header
 - Choose **View stats**

Showing 714 rows | 8 columns | Search columns...

	is_accepted	M	_coding
37.34	true		031
05.21	true		869-7;
59.99	true		696
69.26	true		076
08.52	true		984
65.06	true		757
42.33	true		-7:
27.91	true		...
60.91	true		613

SQL Terminal Autocomplete Project scoped Files saved Checks passed

- Confirm there is only **True** values appearing

Output claims x

is_accepted	Boolean	714 rows
True		714
False		0
Null		0

Step 4: Commit your work to save it

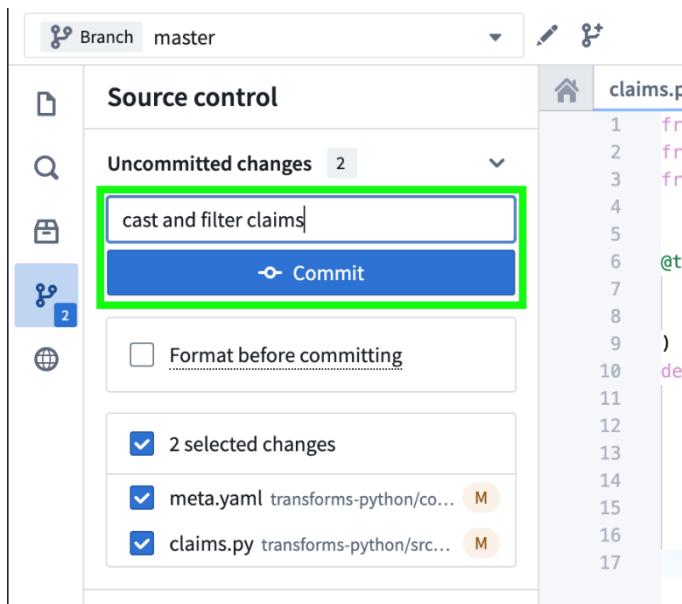
Commits will trigger checks which help us keep your code correct and performant.

1. Click **Commit** in the top right corner



```
claims.py x
1 from pyspark.sql import functions as F
2 from pyspark.sql import types as T
3 from transforms.api import transform_df, Input, Output
4
```

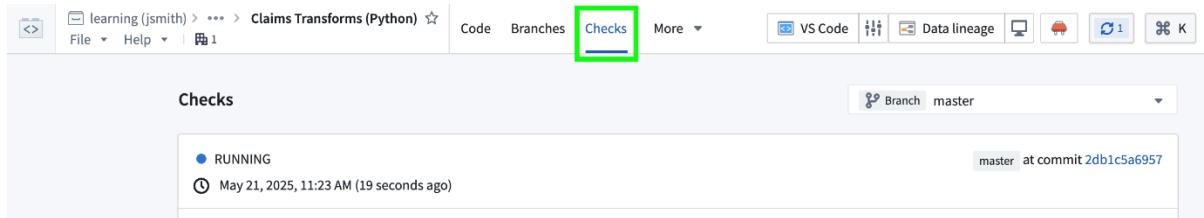
2. Input a commit message, e.g. "cast and filter claims"



3. click **Commit**

This will trigger checks, which you can see as follows:

1. Click the **Checks** tab in the top of your screen



2. Open the first Check to see its log messages

The screenshot shows the DataBrick interface with the 'Checks' tab selected. A green circle indicates a 'SUCCEEDED' status. The run was completed on May 21, 2025, at 11:23 AM, taking 1m 57.936s. The log messages section shows a detailed breakdown of the check steps, all of which succeeded.

```

Checks (took 1m 51.988s)
  - foundry-publish
    :transforms-python:versionPy | SUCCESS | 0.003 secs
    :transforms-python:symlinkMambaExecutable | SUCCESS | 0.003 secs
    :transforms-python:runRequires | SUCCESS | 0.002 secs
    :transforms-python:generatePomFileForUserCodePublication | SUCCESS | 0.002 secs
    :transforms-python:containerTransformsPublish | SUCCESS | 0.002 secs
    :transforms-python:checkSharedLibraries | SKIPPED | 0.001 secs
    :transforms-python:writeToVersionMetadata | SUCCESS | 0.001 secs
    :transforms-python:runPrintLockFile | SKIPPED | 0.0 secs
    :transforms-python:check | UNKNOWN | 0.0 secs
    :transforms-python:patch | UNKNOWN | 0.0 secs
    :transforms-python:checkScalaDependenciesInsight | SKIPPED | 0.0 secs
    :transforms-python:renderDependencyGraph | UNKNOWN | 0.0 secs
    :transforms-python:prepackage | UNKNOWN | 0.0 secs
    :transforms-python:publish | UNKNOWN | 0.0 secs
    :renderShrinkwrap | SUCCESS | 9.634 secs
    :checkShrinkwrap | SUCCESS | 0.755 secs
    :publishJobSpecs | SUCCESS | 0.031 secs
    :publishJobCustomMetadata | SUCCESS | 0.021 secs
    :publish | UNKNOWN | 0.0 secs
  
```

- Click the **Code** tab above and then the files icon in the left sidebar to return to your files.

The screenshot shows the DataBrick interface with the 'Code' tab selected. The left sidebar has a 'Files' icon highlighted with a green box. A blue bar at the top indicates a new commit has been made to the branch. The code editor shows a Python file named 'claims.py' with some code and a commit message input field.

```

claims.py x
1 from pyspark.sql import functions as F
2 from pyspark.sql import types as T
3 from transforms.api import transform_df, Input, Output
4
5
6 @transform_df(
7     Output("claims", main.dataset.b610357e0-43ee-481d-8e94-7140b8d92c4"),
8     claims_raw=Input("claims_raw", dataset.b5e0d3a7-edfb-4ee2-b898-0d4084c57ee1"),
9 )
10 def compute(claims_raw):
11     # Cast the "date" column from string to date
12     claims = claims_raw.withColumn("date", F regexp_replace("date", "###", "").cast(T.DateType()))
13
14     # Filter out declined claims
15     claims = claims.filter(F.col("is_accepted") == True)
16
17     return claims
  
```

Optional: if you see the blue bar with the message "A new commit has been made to this branch", click **Update to most recent version**.

Join in Another Dataset

In preparing the financial overview for the CFO, you need to add the lines of business for each claim to your dataset. This information is contained in the second dataset we uploaded previously, **policies_raw**.

We will use the JOIN operation to include the new column(s) to our existing claims dataset. We perform this transformation in the same code file which we used for our cast and filter operation.

Below you see the difference between the code file as is and post applying these changes.

Before applying these instructions:



```
claims.py x
1 from pyspark.sql import functions as F
2 from pyspark.sql import types as T
3 from transforms.api import transform_df, Input, Output
4
5
6 @transform_df(
7     Output("claims", main.dataset.610357e0-43ee-481d-8e94-7140b8d9b2c4"),
8     claims_raw=Input("claims_raw", dataset.b5e0d3a7-edfb-4ee2-b898-0d4084c57ee5"),
9 )
10 def compute(claims_raw):
11     # Cast the "date" column from string to date
12     claims = claims_raw.withColumn("date", F regexp_replace("date", "###", "") .cast(T.DateType()))
13
14     # Filter out declined claims
15     claims = claims.filter(F.col("is_accepted") == True)
16
17     return claims
```

```
from pyspark.sql import functions as F
```

```
from pyspark.sql import types as T
```

```
from transforms.api import transform_df, Input, Output
```

```
@transform_df(
```

```
    Output("ri.foundry.main.dataset.610357e0-43ee-481d-8e94-7140b8d9b2c4"),
```

```
    claims_raw=Input("ri.foundry.main.dataset.b5e0d3a7-edfb-4ee2-b898-0d4084c57ee5"),
```

```
)
```

```
def compute(claims_raw):
```

```
    # Cast the "date" column from string to date
```

```
    claims = claims_raw.withColumn("date", F regexp_replace("date", "###", "") .cast(T.DateType()))
```

```
    # Filter out declined claims
```

```
    claims = claims.filter(F.col("is_accepted") == True)
```

```
    return claims
```

After applying these instructions:



```
claims.py x
1  from pyspark.sql import functions as F
2  from pyspark.sql import types as T
3  from transforms.api import transform_df, Input, Output
4
5
6  @transform_df(
7      Output("claims", main.dataset.610357e0-43ee-481d-8e94-7140b8d9b2c4"),
8      claims_raw=Input("claims_raw", dataset.b5e0d3a7-edfb-4ee2-b898-0d4084c57ee5"),
9      policies_raw=Input("policies_raw", dataset.08535268-5766-41ef-b708-c092faf9f598"),
10  )
11 def compute(claims_raw, policies_raw):
12     # Cast the "date" column from string to date
13     claims = claims_raw.withColumn("date", F regexp_replace("date", "###", "") .cast(T.DateType()))
14
15     # Filter out declined claims
16     claims = claims.filter(F.col("is_accepted") == True)
17
18     # Join in policies to get LOB
19     claims = claims.join(
20         policies_raw,
21         on="policy_id",
22         how="left"
23     )
24
25     return claims
```

```
from pyspark.sql import functions as F
from pyspark.sql import types as T
from transforms.api import transform_df, Input, Output
```

```
@transform_df(
    Output("ri.foundry.main.dataset.610357e0-43ee-481d-8e94-7140b8d9b2c4"),
    claims_raw=Input("ri.foundry.main.dataset.b5e0d3a7-edfb-4ee2-b898-0d4084c57ee5"),
    policies_raw=Input("ri.foundry.main.dataset.08535268-5766-41ef-b708-c092faf9f598"),
)
def compute(claims_raw, policies_raw):
    # Cast the "date" column from string to date
    claims = claims_raw.withColumn("date", F regexp_replace("date", "###", "") .cast(T.DateType()))
# Filter out declined claims
claims = claims.filter(F.col("is_accepted") == True)
# Join in policies to get LOB
```

```

claims = claims.join(
    policies_raw,
    on="policy_id",
    how="left"
)
return claims

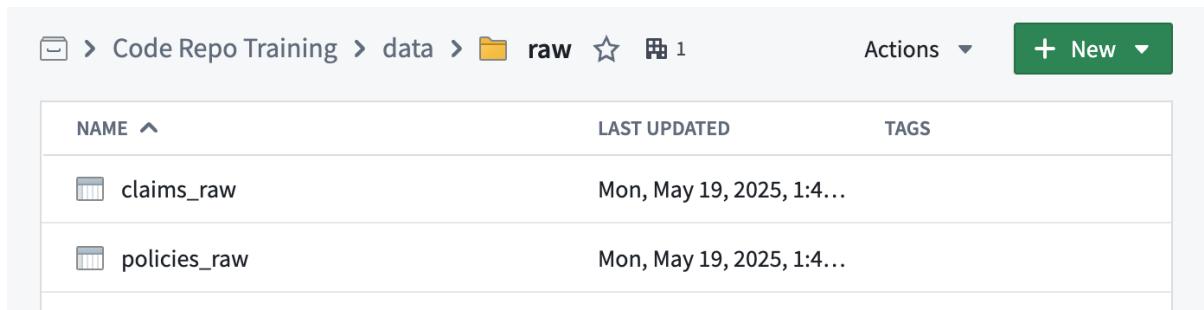
```

Step 1: Add your second input dataset

1. In between line 8 and 9, add the following second Input line following the previous one:

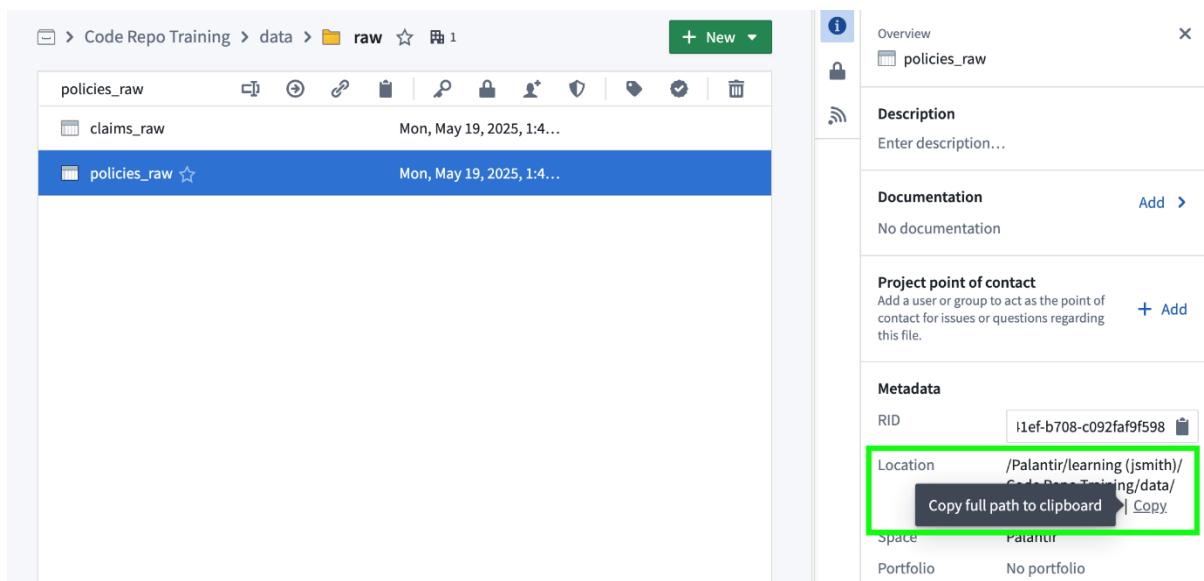
```
policies_raw=Input("<>path will go here></>"),
```

2. Navigate to your **policies_raw** dataset located in your **data > raw** folder.



NAME	LAST UPDATED	TAGS
claims_raw	Mon, May 19, 2025, 1:4...	
policies_raw	Mon, May 19, 2025, 1:4...	

3. Select **policies_raw** and copy its full location path (remember you can also copy the rid instead).



Overview
policies_raw

Description
Enter description...

Documentation
Add >
No documentation

Project point of contact
Add >
No user assigned

Metadata
RID: 1ef-b708-c092faf9f598
Location: /Palantir/learning (jsmith)/Code Repo Training/data/
Copy full path to clipboard | Copy

Space: Palantir
Portfolio: No portfolio

4. Go back to your code repository
5. Paste the path within the quotes of your new line

```
policies_raw=Input("path goes here"),
```

6. Your code file now looks like this:

```

claims.py x
1  from pyspark.sql import functions as F
2  from pyspark.sql import types as T
3  from transforms.api import transform_df, Input, Output
4
5
6  @transform_df(
7      Output("claims", main.dataset.610357e0-43ee-481d-8e94-7140b8d9b2c4"),
8      claims_raw=Input("claims_raw", dataset.b5e0d3a7-edfb-4ee2-b898-0d4084c57ee5"),
9      policies_raw=Input("policies_raw", dataset.08535268-5766-41ef-b708-c092faf9f598"),
10 )
11 def compute(claims_raw):
12     # Cast the "date" column from string to date
13     claims = claims_raw.withColumn("date", F.regexp_replace("date", "###", "")).cast(T.DateType())
14
15     # Filter out declined claims
16     claims = claims.filter(F.col("is_accepted") == True)
17
18     return claims

```

Step 2: Perform a basic join operation

The two datasets to join share a column, namely **policy_id**. We will use this as the foreign key when left joining the policies to claims. Our intent is to add the line of business to the original dataset.

1. In line 11, first add the policies_raw as a second input to your function:

```
def compute(claims_raw, policies_raw):
```

2. Below your filter for only accepted claims, i.e. below line 16 in the screenshot above, add the following code:

```
# Join in policies to get LOB
```

```
claims = claims.join(
    policies_raw,
    on="policy_id",
    how="left"
)
```

3. Your code file will now look as follows:

The screenshot shows a Jupyter Notebook environment with two tabs: 'claims.py' and 'Output claims'. The 'claims.py' tab displays a Python script for data processing, specifically using PySpark's SQL API to transform raw datasets into a final claims DataFrame. The 'Output' tab shows the resulting DataFrame, which includes columns for state, zip_code, street, line_of_business, policyholder_age, and policyholder_has... . A green box highlights the 'line_of_business' column, and a tooltip indicates there are 714 rows. The bottom navigation bar includes 'Preview', 'Tests', 'File Changes', 'Build', 'Docs', 'Task Runner', 'SQL', and 'Terminal' buttons.

```
1 from pyspark.sql import functions as F
2 from pyspark.sql import types as T
3 from transforms.api import transform_df, Input, Output
4
5
6 @transform_df(
7     Output("claims", main.dataset.610357e0-43ee-481d-8e94-7140b8d9b2c4"),
8     claims_raw=Input("claims_raw", dataset.b5e0d3a7-edfb-4ee2-b898-0d4084c57ee5"),
9     policies_raw=Input("policies_raw", dataset.08535268-5766-41ef-b708-c092faf9f598"),
10 )
11 def compute(claims_raw, policies_raw):
12     # Cast the "date" column from string to date
13     claims = claims_raw.withColumn("date", F.regexp_replace("date", "###", "")).cast(T.DateType())
14
15     # Filter out declined claims
16     claims = claims.filter(F.col("is_accepted") == True)
17
18     # Join in policies to get LOB
19     claims = claims.join(
20         policies_raw,
21         on="policy_id",
22         how="left"
23     )
24
25     return claims
```

	state	zip_code	street	line_of_business	policyholder_age	policyholder_has...
1	Maine	24031	044 Justina Manors	accident	74	null
2	Indiana	60869-7377	76161 Kirlin Causeway	accident	74	null
3	South Dakota	46696	830 Schuppe Road	liability	26	null
4	Montana	10076	4980 Lonie Union	life	57	null
5	Ohio	94984	066 Matilde Mission	life	57	null
6	Arizona	85757	96654 Adrien Highway	life	57	null
7	Wisconsin	64054-7333	09729 Ollie Extension	life	5	null
8	Oregon	79050	1865 Selina Ferry	life	5	null
9	Nevada	64613	7526 Metz Gardens	pet	67	null

4. Click **Preview** to confirm that the new dataset contains the `line_of_business` column

- Note: When using preview on larger datasets, it will typically sample a subset of the rows. If desired, the sampling strategy can be specified to ensure particular rows are included in the preview. This is of critical importance when using joins to ensure both samples have matching rows based on the join criteria.
 - Close the Preview by clicking the **Preview** button in the bottom navigation bar

5. **Commit** your work to save it, like you did previously.

The screenshot shows the Databricks IDE interface. On the left, there's a sidebar titled "Source control" with sections for "Uncommitted changes" (containing 1 item) and "joining claims with policies". Below these are buttons for "Format before committing" (unchecked), "1 selected changes" (checked), and "claims.py transforms-python/src..." (with a modified icon). The main area is a code editor with a file named "claims.py" open. The code imports functions and types from PySpark SQL and PySpark SQL transforms, and defines a function to compute claims by joining them with policies. A green box highlights the "Commit" button at the bottom of the code editor. At the top right, there are buttons for "Preview", "Test", "Commit" (which is highlighted with a red box), "Build", and "Propose changes". A "Ask AI P Assist" button is also visible.

```
1 from pyspark.sql import functions as F
2 from pyspark.sql import types as T
3 from transforms.api import transform_df, Input, Output
4
5
6 @transform_df(
7     Output("claims", main.dataset.610357e0-43ee-481d-8e94-7140b89b2c4a"),
8     claims_raw=Input("claims_raw", dataset.b5e0d3a7-edfb-4ec2-b898-d408c57ee5"),
9     policies_raw=Input("policies_raw", dataset.88535268-5766-41ef-b708-c092faf9f598"),
10 )
11 def compute(claims_raw, policies_raw):
12     # Cast the "date" column from string to date
13     claims = claims_raw.withColumn("date", F regexp_replace("date", "###", "").cast(T.DateType()))
14
15     # Filter out declined claims
```

6. Review your checks for any potential failures.

Build Your Dataset

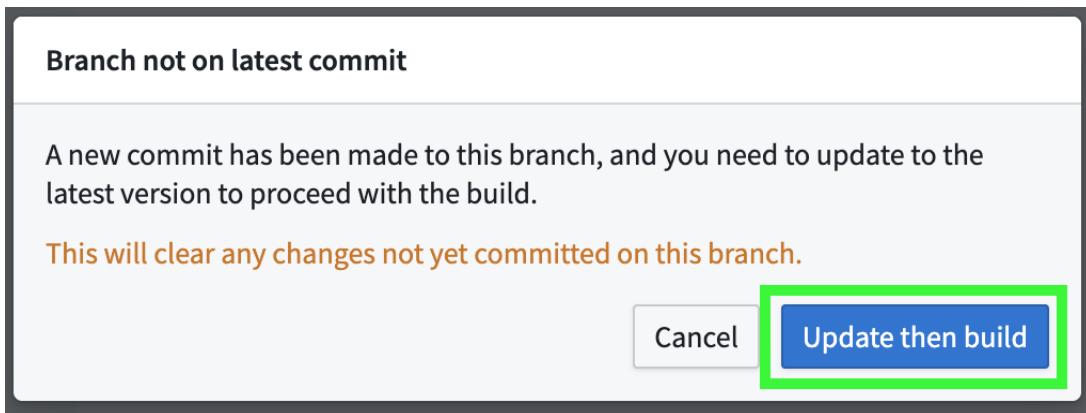
Previously, we only previewed our results and committed them to save our code.

- **Preview** produces a sample output without committing changes, running checks, or materializing any datasets in Foundry.
- **Committing** saves our work and triggers Checks, but it doesn't build the defined outputs as datasets.
- **Build** actually builds the dataset(s) which you define as your output in the code file(s). It also commits the latest code changes and runs checks as part of this.

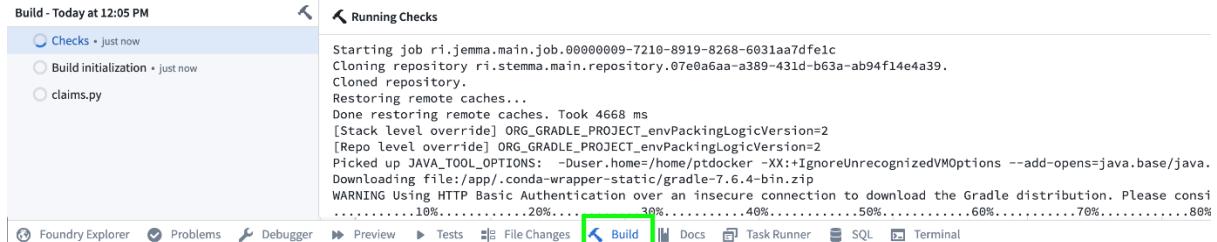
1. Click the **Build** button



2. If you get the following message, click **Update then build**.



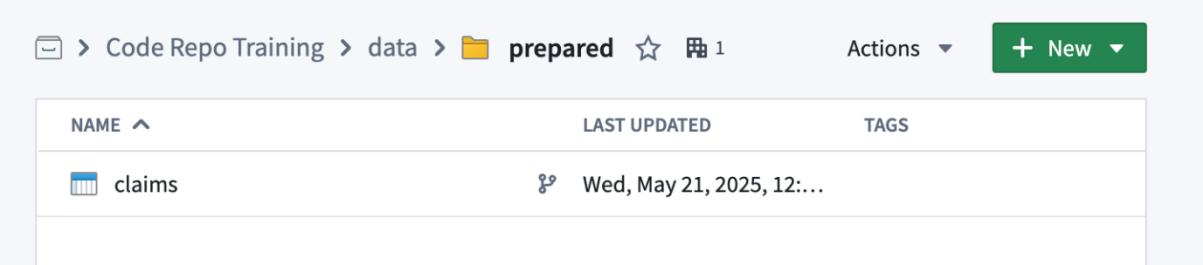
3. Building your dataset will kick off Checks automatically which you see in the bottom part of your screen.



- You can toggle the Build details on and off by clicking the **Build** tab in the bottom navigation bar.
 - The build might take a couple of minutes to complete.
4. Once the build is finished, it shows the preview of the dataset in the bottom of your code repository.
5. Open the dataset by clicking on “claims” just above the preview table, ideally holding ctrl/cmd to open it in a new tab.

policy_id	claim_id	date	claim_value	is_accepted	state	zip_code
e6ec03ab-df55-4f1b-9	d9a20f9-0344-4522-9	2024-03-01	9737.34	true	Maine	24031
e6ec03ab-df55-4f1b-9	f36cdfd8-ed7f-4389-8	2023-08-01	5305.21	true	Indiana	60869-7:
73d942d2-e0fe-4efa-9	aa3f996e-8aaa-4c54-8	2024-11-01	4918.76	true	Georgia	33811
75e4ff7d-c980-4583-8	3ebf380c-3f14-4672-9	2023-02-01	2677.39	true	North Carolina	51238
75e4ff7d-c980-4583-8	0249c080-d497-458e-9	2023-01-01	3230.99	true	New Hampshire	15249
75e4ff7d-c980-4583-8	285el098-cb39-4c17-b	2024-01-01	4992.63	true	Oregon	Calculate row count
75e4ff7d-c980-4583-8	436232d7-0448-4b15-b	2024-05-01	1578.20	true	New Yd	7-3:
75e4ff7d-c980-4583-8	66c364cb-6cc1-4cb4-9	2023-02-01	4528.45	true	Nevada	81717-3:

6. Verify that the dataset now exists in the **prepared** folder.



NAME	LAST UPDATED	TAGS
claims	Wed, May 21, 2025, 12:...	

Congratulations. You have built your first dataset using Code Repositories.

Collaboration via Branching

Introduction

Branching in Foundry Code Repositories is a powerful feature that allows you to create independent lines of development within your project, enabling multiple team members to work on different tasks simultaneously without interfering with the main codebase. By creating branches, you can experiment with new features, perform bug fixes, or implement enhancements in isolation, ensuring that the master branch remains stable and production-ready. This approach not only facilitates collaboration and code review processes but also supports efficient version control and project management within Foundry. Foundry branching implements an industry-standard Git-like version control paradigm. For the scope of this training, we'll be working with Code Repositories branches, as opposed to Foundry branches.

In this module, you will explore the following concepts:

- protecting your master branch,
- creating a new branch,
- perform a transform on your new branch,
- merge the changes to your master branch.

Protecting Master Branch

Protected branches can only be modified via pull requests. Make a branch protected if you have multiple users contributing to it and want to ensure that code is reviewed before it's merged into this branch.

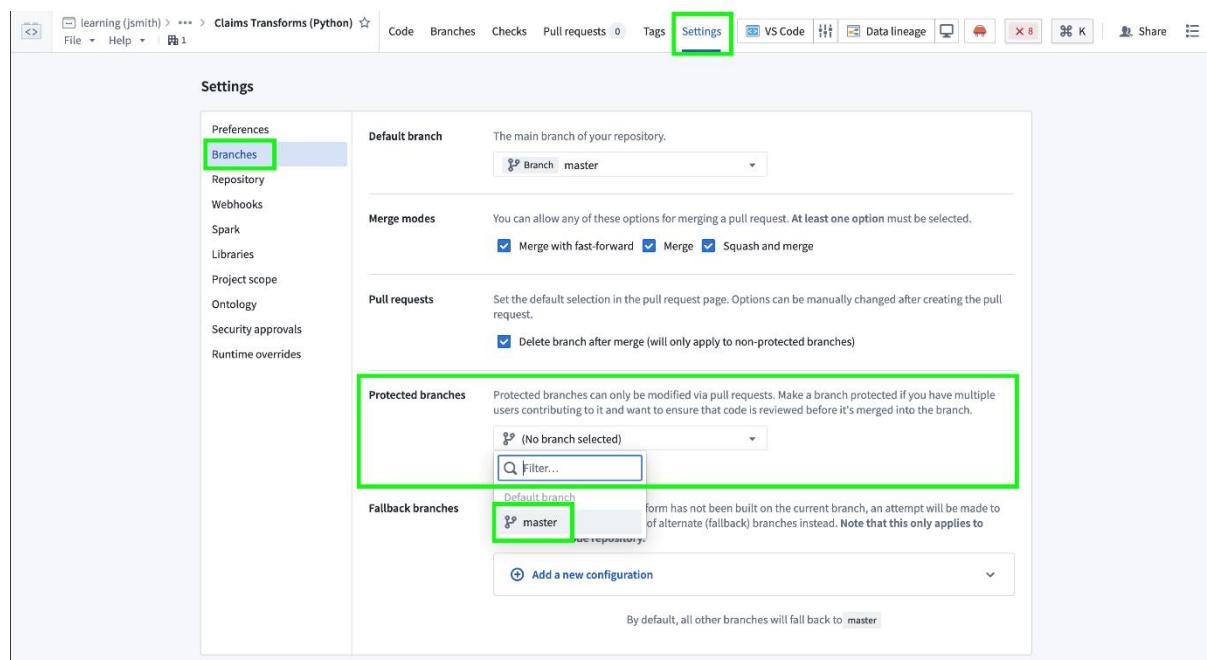
Step 1: Protect master branch

1. In your code repository, open **Settings** from the top navigation bar

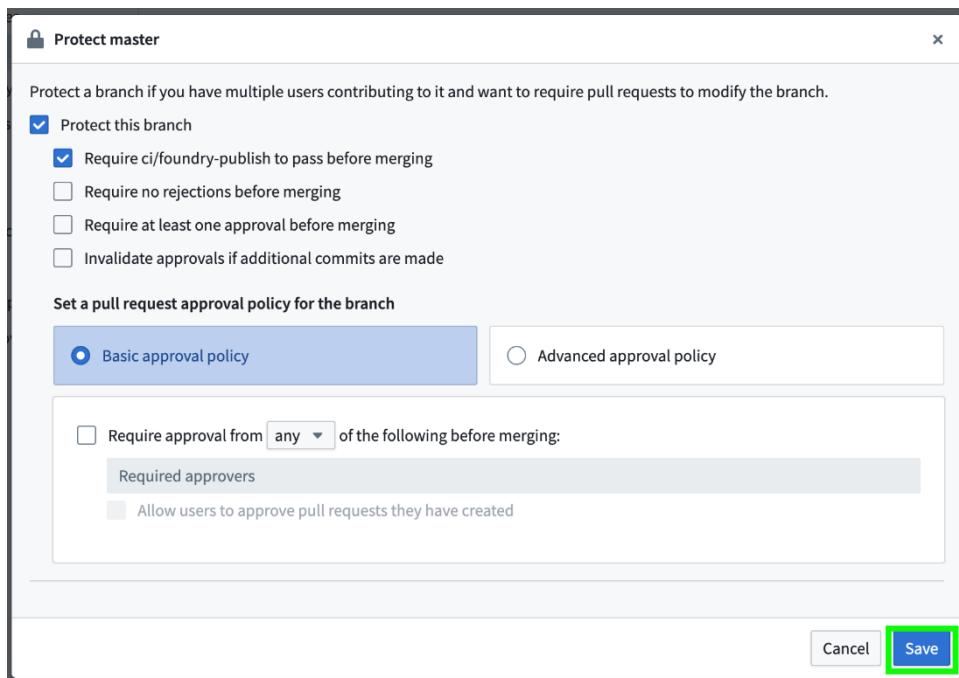
2. Select Branches

3. Choose the **master** branch as one of the **Protected branches**

- Important note: if you don't see the **Protected branches** option, it's likely due to missing permissions. To overcome this, you can either:
 1. try to change your Code Repositoryy permissions to Owner (this should be possible if it is located in your own project),
 2. ask an admin to give you privileged owner access to this repository, or
 3. just skip this part. You will still be able to proceed with the tutorial even if your master branch is not protected.



4. In the popup that shows, leave everything as is and click **Save**.



5. Your settings will update and show as selected:

The screenshot shows the Project Overview page with the 'Code' tab selected. On the left, there is a sidebar with icons for Home, Search, Notifications, and Project scope (ontology, security approvals, runtime overrides). The main area has tabs for 'Branches', 'Checks', 'Pull requests', and 'More'. Under 'Pull requests', there is a checkbox 'Delete branch after merge (will only apply to non-protected branches)'. The 'Protected branches' section contains a dropdown menu '(No branch selected)' and a list with 'master'. To the right of 'master' is a status message: 'Review Requirements ci/foundry-publish runs successfully.' A green box highlights the 'master' entry in the protected branches list.

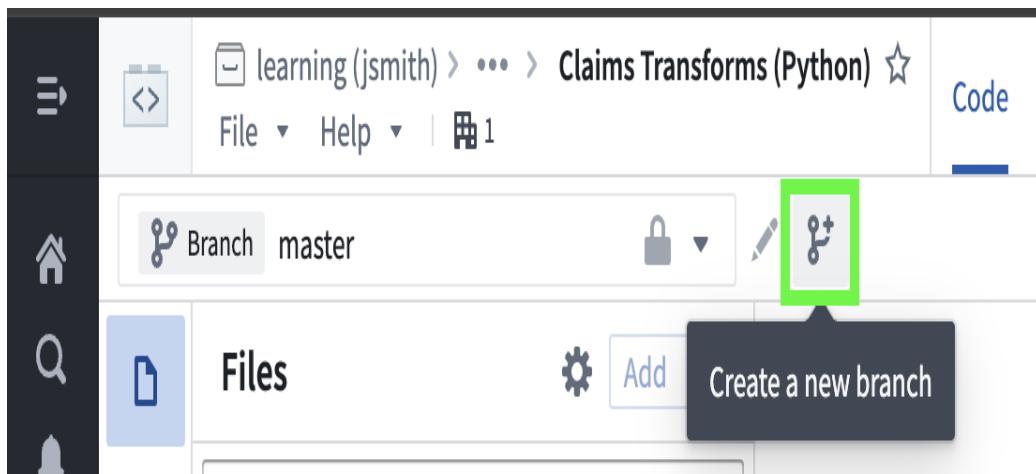
6. Return to your **Code** via the top navigation bar.

Performing a Transform on Your Branch

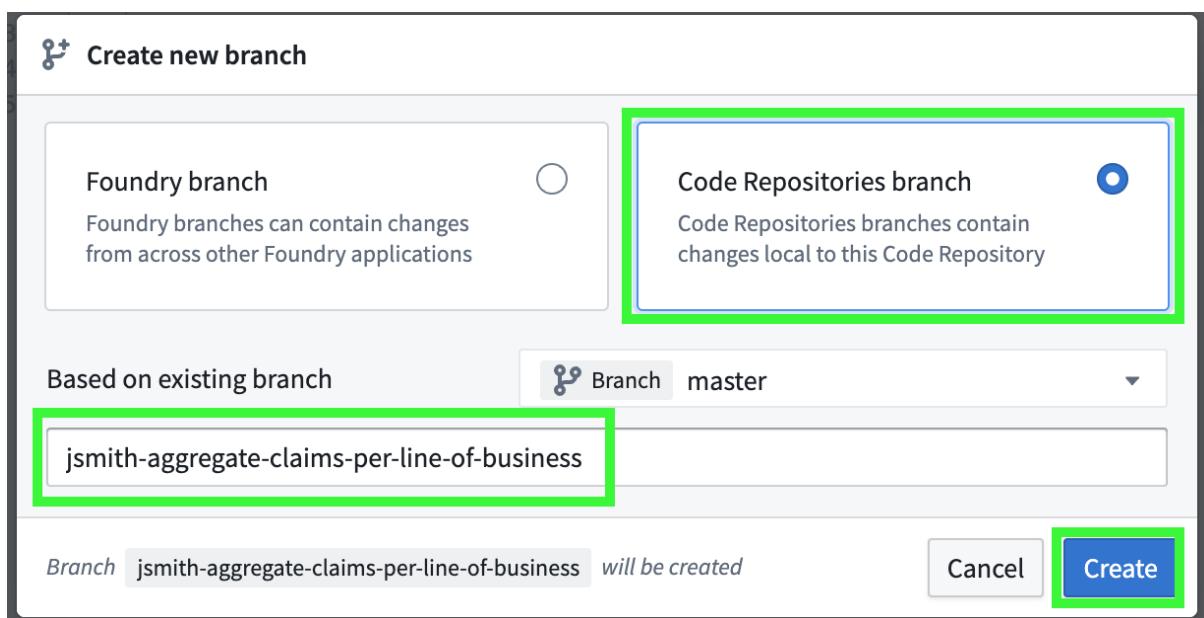
In this lesson, we will create another transform in your existing code file on a new branch.

Step 1: Create new branch

1. Click the little icon with the + to the right of your **Branch master** dropdown, by which you **Create a new branch**.



2. Select **Code Repositories branch**, if presented with the choice
3. **Name** your branch following a descriptive pattern, e.g. <your_username>-aggregate-claims-per-line-of-business
4. Click **Create**



5. You will now land on your code view showing the newly created branch.

Step 2: Create your transform

Remember that we wanted to find the average of all paid out claims by line of business. We will use the same code file as in the previous section, **claims.py**.

Your file should look as follows at this point in time:

```

1  from pyspark.sql import functions as F
2  from pyspark.sql import types as T
3  from transforms.api import transform_df, Input, Output
4
5
6  @transform_df(
7      Output("claims", main.dataset.610357e0-43ee-481d-8e94-7140b8d9b2c4"),
8      claims_raw=Input("claims_raw", dataset.b5e0d3a7-edfb-4ee2-b898-0d4084c57ee5"),
9      policies_raw=Input("policies_raw", dataset.08535268-5766-41ef-b708-c092faf9f598"),
10 )
11 def compute(claims_raw, policies_raw):
12     # Cast the "date" column from string to date
13     claims = claims_raw.withColumn("date", F regexp_replace("date", "###", "") .cast(T.DateType()))
14
15     # Filter out declined claims
16     claims = claims.filter(F.col("is_accepted") == True)
17
18     # Join in policies to get LOB
19     claims = claims.join(
20         policies_raw,
21         on="policy_id",
22         how="left"
23     )
24
25     return claims

```

1. Replace line 24 with the following code to aggregate claims by line of business.

```
# Compute average cost per LOB

claims_aggregated =
claims.groupBy("line_of_business").agg(F.avg("claim_value").alias("avg_claim_cost"))

return claims_aggregated
```

- o This will make your file look like this:

```

1  from pyspark.sql import functions as F
2  from pyspark.sql import types as T
3  from transforms.api import transform_df, Input, Output
4
5
6  @transform_df(
7      Output("claims", main.dataset.610357e0-43ee-481d-8e94-7140b8d9b2c4),
8      claims_raw=Input("claims_raw", dataset.b5e0d3a7-edfb-4ee2-b898-0d4084c57ee5),
9      policies_raw=Input("policies_raw", dataset.08535268-5766-41ef-b708-c092faf9f598),
10 )
11 def compute(claims_raw, policies_raw):
12     # Cast the "date" column from string to date
13     claims = claims_raw.withColumn("date", F regexp_replace("date", "###", "") .cast(T.DateType()))
14
15     # Filter out declined claims
16     claims = claims.filter(F.col("is_accepted") == True)
17
18     # Join in policies to get LOB
19     claims = claims.join(
20         policies_raw,
21         on="policy_id",
22         how="left"
23     )
24
25     # Compute average cost per LOB
26     claims_aggregated = claims.groupBy("line_of_business").agg(F.avg("claim_value").alias("avg_claim_cost"))
27
28
29     return claims_aggregated

```

2. Hit **Preview** in the top to run a preview of your results.
3. Confirm that you see a new table with two columns: **line_of_business** and **avg_claim_cost**

The screenshot shows the DataBrick interface. At the top, there's a navigation bar with icons for Home, Preview, Test, Commit, Build, and Propose changes. A green box highlights the 'Preview' button. To the right of the preview button is a 'Ask AIP Assist' dropdown.

The main area shows the Python code for 'claims.py'. The code imports necessary libraries and defines a function 'compute' that reads raw data from datasets, filters claims, joins them with policies, and then groups by line of business to calculate average claim cost.

Below the code is an 'Output' section titled 'claims'. It contains a table with 8 rows, each representing a line of business and its average claim cost. The table has two columns: 'line_of_business' and 'avg_claim_cost'. The data is as follows:

line_of_business	avg_claim_cost
accident	5084.3460839160890
liability	5471.4276136363640
life	4894.0605747126430
pet	4666.1075781249990
legal	5225.7619607843135
aviation	4991.3060465116300
marine	4716.8577142857140
auto	4850.4941346153840

At the bottom of the interface, there are various navigation links like Preview, Tests, File Changes, Build, Docs, Task Runner, SQL, Terminal, and a status bar indicating AIP autocomplete, Project scoped, Files saved, Checks passed, and a bell icon.

4. Commit your work to save it, on your feature branch

- Click **Commit** in the top right corner
- Name your commit, e.g. aggregating claims

The screenshot shows the GitHub interface for a repository named 'learning (jsmith)'. The current branch is 'jsmith-aggregate-claims-per-line-...'. The commit message is 'aggregating claims'.

The commit message is highlighted with a green box. Below the message, there are buttons for 'Commit' and 'Format before committing'. There are also checkboxes for '1 selected changes' and 'claims.py transforms-python/src...'.

The commit message is also shown in the code editor area on the right side of the interface.

5. Open Checks to see that your commit has run successfully

- Click **Checks** in the top navigation bar
- Confirm that the first check in line succeeds

The screenshot shows the 'Checks' tab selected in the top navigation bar of a GitHub repository named 'Claims Transforms (Python)'. There are two entries listed:

- SUCCEEDED: jsmith-aggregate-claims-per-line-of-business at commit ffb261e41a3 (May 21, 2025, 1:49 PM, took 1m 12.736s)
- SUCCEEDED: jsmith-aggregate-claims-per-line-of-business at commit 653bedc8128 (May 21, 2025, 1:34 PM, took 1m 8.148s)

Merge Your Branch into Master

Now that we have verified that our change creates the desired result and checks run successfully, we will merge the code change into the master branch via a Pull Request.

Step 1: Create a new pull request

1. Open **Branches** from the top navigation bar
2. Click on **Propose changes** in the panel belonging to your feature branch

The screenshot shows the 'Branches' tab selected in the top navigation bar of the same GitHub repository. It lists two branches:

Default branch	Checks	Pull request
master	Passed 1/1 32 minutes ago	-

Active branches	Checks	Pull request
jsmith-aggregate-claims-per-line-of-business	Passed 1/1 3 minutes ago	Propose changes

3. A new screen called **New pull request** will open
4. The title is autofilled. Modify if desired, e.g. "Computing average claim value by line of business"
5. Add a meaningful description which helps your colleagues understand the need and the specifics of your change, e.g. "Creating a new table with line of business and average claim value per such lob (as per CFO request)."
6. Leave the rest as is.
7. Click **Create pull request**

New pull request

Title (required)
Computing average claim value by line of business

Description
Creating a new table with line of business and average claim value per such line of business (as per CFO request)

Merge into (required)
Branch master from Branch jsmith-aggregate-claims-per-line-of-business Merge when ready

Showing 1 file change

```
MOD transforms-python/src/myproject/datasets/claims.py
...
21     on="policy_id",
22     how="left"
23 )
24     return claims
25
26     on="policy_id",
27     how="left"
28 )
29     # Compute average cost per LOB
30     claims_aggregated = claims.groupby("line_of_business").agg(F.avg("claim_value")
31     .alias("avg_claim_cost"))
32
33     return claims_aggregated
```

Step 2: Merge your changes

After clicking **Create pull request** in the previous step, you will land on the following screen. This is a summary of your pull request.

learning (jsmith) > *** > Claims Transforms (Python)

Code Branches Checks Pull requests 1 Tags Settings

VS Code Data lineage Pipeline Merge when ready

Computing average claim value by line of business

Creating a new table with line of business and average claim value per such line of business (as per CFO request)

Jutta Pitz wants to merge into master from jsmith-aggregate-claims-per-line-of-business 5 minutes ago

Review warnings before merging 1 warning

Some datasets directly affected by this pull request have not been built with the latest version of the code on jsmit

Configure and build...

Overview Files changed 1 Impact analysis Pipeline review Security changes Commits 1 Conversation

This pull request is open.

- ✓ jsmit
- ✓ No code approval required.
- ✓ Checks passed 10 minutes ago.

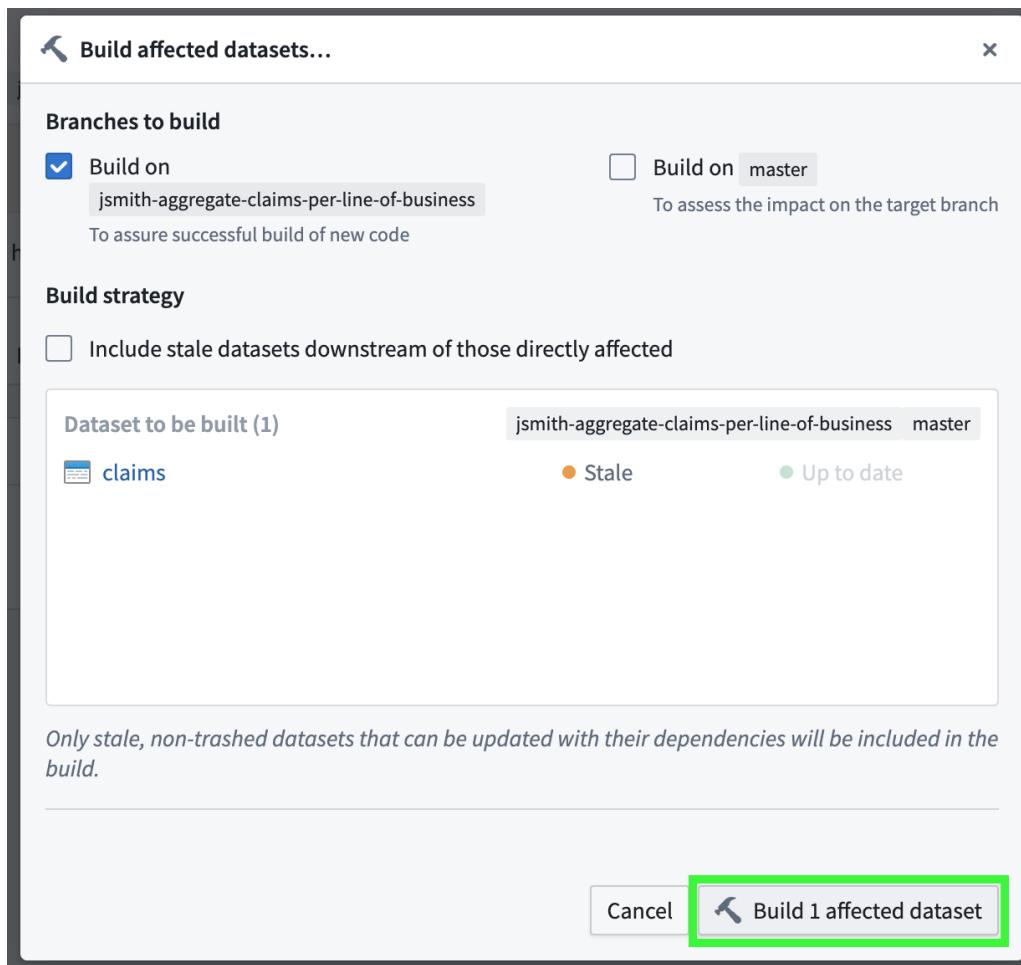
Add/remove reviewers View details

Review PR Delete branch when merged Merge when ready Close Squash and merge

In our example, a warnings panel shows which indicates that some of the datasets have not yet been built with the latest code changes. Normally, it's a good practice to build the dataset on a branch first (see optional steps below) but in our case we can also go ahead and merge without these.

[Optional steps]

- [optional] Click on **Configure and build**
- [optional] In the new panel that opens, click on **Build 1 affected dataset**



- [optional] Open the running build in a new tab by clicking on **View progress** (with **ctrl/cmd**) in the green notification that shows at the top.



outing average claim value by line of business

- [optional] Refresh your screen to see the screen without the warnings panel.

[End of optional steps]

1. Explore the bottom part of this screen further:
 - Review the updated files by clicking in the **Files changed 1** tab.
 - View a graph of your raw and prepared datasets under **Pipeline review**.
 - Review the Commits you made by clicking in the **Commits 1** tab.
 - Add reviewers where it currently says "**No code approval required.**" Find a teammate's name from the list to add them.

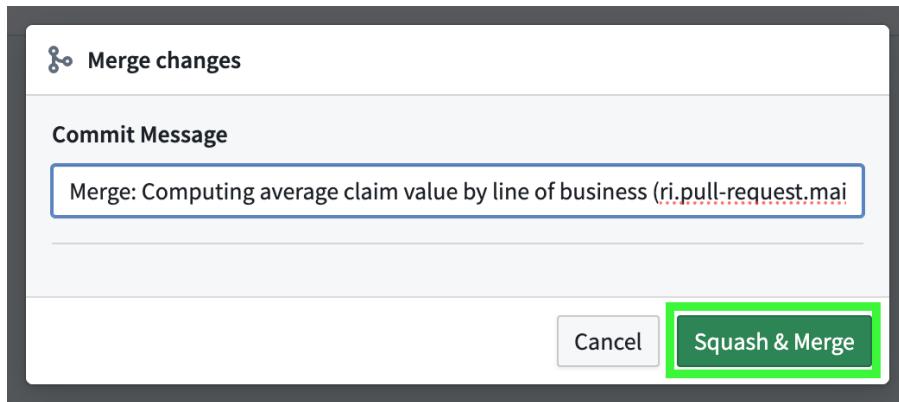
A screenshot of a GitHub pull request page. The title is "Claims Transforms (Python)" and the branch is "jsmith-aggregate-claims-per-line-of-business". The pull request has 1 file changed and 1 commit. The commit message is "Computing average claim value by line of business". The commit author is "Jutta Pitz (jpit)". The commit status shows "Checks passed 27 minutes ago". A green box highlights the commit author's name "Jutta Pitz (jpit)".

2. When you are done exploring, hit **Squash and merge**.

A screenshot of a GitHub pull request page. The title is "Claims Transforms (Python)" and the branch is "jsmith-aggregate-claims-per-line-of-business". The pull request has 1 file changed and 1 commit. The commit message is "Computing average claim value by line of business". The commit author is "Jutta Pitz (jpit)". The commit status shows "Checks passed 27 minutes ago". A green box highlights the commit author's name "Jutta Pitz". Below the commit, there is a section for "No code approval required" with an "Add/remove reviewers" link and a "Additional reviewers" section containing "Jutta Pitz". At the bottom, there are buttons for "Review PR", "Delete branch when merged" (unchecked), "Merge when ready" (unchecked), and a green box highlighting the "Squash and merge" button.

- General note: "Squash and merge" combines all the commits from a pull request into a single commit before merging it into the master, resulting in a cleaner, more concise commit history. On the other hand, "Merge" integrates all individual commits from the pull request as they are, preserving the detailed commit history but potentially making the commit log more cluttered.

3. Leave the Commit Message as is and hit **Squash & Merge**



4. Your Pull request will show as merged afterwards and the count next to Pull requests in the top navigation bar will show 0.

Creating a new table with line of business and average claim value per such line of business (as per CFO request)

[Edit description](#)

Jutta Pitz created pull request into master from jsmith-aggregate-claims-per-line-of-business 1 hour ago

Pull request was merged. 57 minutes ago

jsmith-aggregate-claims-per-line-of-business has been deleted.

Overview Files changed 1 Impact analysis Pipeline review Security changes Commits 1 Conversation

This pull request is closed.

- ✓ jsmith-aggregate-claims-per-line-of-business is up-to-date with master .
- ✓ No code approval required.

Additional reviewers:
Jutta Pitz

⚠ Some of the checks are not found: ci/foundry-publish

5. Click back on **Code**. Code Repositories will notify you that “A new commit has been made to this branch”, which in this case means the master branch.
6. Click “Update to most recent version” to see the changes made.

Branch master

Source control

A new commit has been made to this branch Update to most recent version

```
claims.py
1 from pyspark.sql import functions as F Jutta Pitz, 3 hours ago · cast and filter claims
2 from pyspark.sql import types as T
3 from transforms.api import transform_df, Input, Output
4
5
6 @transform_df(
7     Output("claims", main.dataset.610357e0-43ee-481d-8e94-7140b8d9b2c4),
8     claims_raw=Input("claims_raw", dataset.b5ed83a7-edfb-4ee2-b898-0d4084c57ee5"),
9     policies_raw=Input("policies_raw", dataset.08535268-5766-41ef-b708-c092fa9f598b"),
10 )
11 def compute(claims_raw, policies_raw):
12     # Cast the "date" column from string to date
13     claims = claims_raw.withColumn("date", F regexp_replace("date", "###", "").cast(T.DateType()))
14
15     # Filter out declined claims
16     claims = claims.filter(F.col("is_accepted") == True)
17
18     # Join in policies to get LOB
19     claims = claims.join(
20         policies_raw,
21         on="policy_id",
22         how="left"
23     )
24
25 return claims
```

Your code file will now entail the latest changes merged in from your Pull request.

Build Your Dataset

As a final step, you will build your desired dataset on master. You will also investigate the data lineage that we saw at the very beginning of this course.

Step 1: Build the final dataset on master

1. Click **Preview** to ensure your merged changes create the desired 2 column table.

The screenshot shows the DataBricks workspace interface. At the top, there's a navigation bar with 'File', 'Help', and a branch dropdown set to 'master'. Below it is a 'Source control' sidebar with 'Uncommitted changes' and a 'Commit' button. The main area displays the 'claims.py' code:

```
from pyspark.sql import functions as F
from pyspark.sql import types as T
from transforms.api import transform_df, Input, Output

@transform_df(
    Output("claims", main.dataset.610357e0-43ee-481d-8e94-7140b8d9b2c4"),
    claims_raw=Input("claims_raw", dataset.b5e0d3a7-edfb-4ee2-bd98-0d4084c57ee5"),
    policies_raw=Input("policies_raw", dataset.08535268-5766-41ef-b700-c092faf9f598"),
)
def compute(claims_raw, policies_raw):
    # Cast the "date" column from string to date
    claims = claims_raw.withColumn("date", F regexp_replace("date", "###", "") .cast(T.DateType()))

    # Filter out declined claims
    claims = claims.filter(F.col("is_accepted") == True)

    # Join in policies to get LOB
    claims = claims.join(
        policies_raw,
        on="policy_id",
        how="left"
    )
    # Compute average cost per LOB
    claims_aggregated = claims.groupBy("line_of_business").agg(F.avg("claim_value").alias("avg_claim_cost"))

    return claims_aggregated
```

Below the code editor is a preview panel titled 'Output claims' showing a table with two columns: 'line_of_business' (String) and 'avg_claim_cost' (Double). The table contains 8 rows of data:

line_of_business	avg_claim_cost
accident	5084.3460839160890
liability	5471.4276136363649
life	4894.0605747126430
pet	4666.1075781249990
legal	5225.7619607843135
aviation	4991.3060465116300
marine	4716.8577142857140
auto	4850.4941346153840

2. Click **Build**.

- o Just like we've seen before, Checks will run automatically.
- o The build then gets initialized.
- o Then the build runs and the build panel shows its progress with this view from job tracker.

```

1 from pyspark.sql import functions as F    Jutta Pitz, 4 hours ago · cast and filter claims
2 from pyspark.sql import types as T
3 from transforms.api import transform_df, Input, Output
4
5
6 @transform_df(
7     Output("claims", main.dataset.610357e0-43ee-481d-be94-7140bb8d9b2c"),
8     claims_raw=Input("claims_raw", dataset.b5e0d3a7-edfb-4ee2-b898-034084c57ee3),
9     policies_raw=Input("policies_raw", dataset.08535268-5766-41ef-b708-c092faf9f598"),
10 )
11 def compute(claims_raw, policies_raw):
12     # Cast the "date" column from string to date
13     claims = claims_raw.withColumn("date", F.regexp_replace("date", "###", "").cast(T.DateType()))
14
15     # Filter out declined claims
16     claims = claims.filter(F.col("is_accepted") == True)
17
18     # Join in policies to get LOB
19     claims = claims.join(
20         policies_raw,
21         on="policy_id",
22         how="left"
23     )
24
25     # Compute average cost per LOB
26     claims_aggregated = claims.groupby("line_of_business").agg(F.avg("claim_value").alias("avg_claim_cost"))
27
28
29 return claims_aggregated

```

Your dataset will show at the bottom of the page as soon as the build has completed.

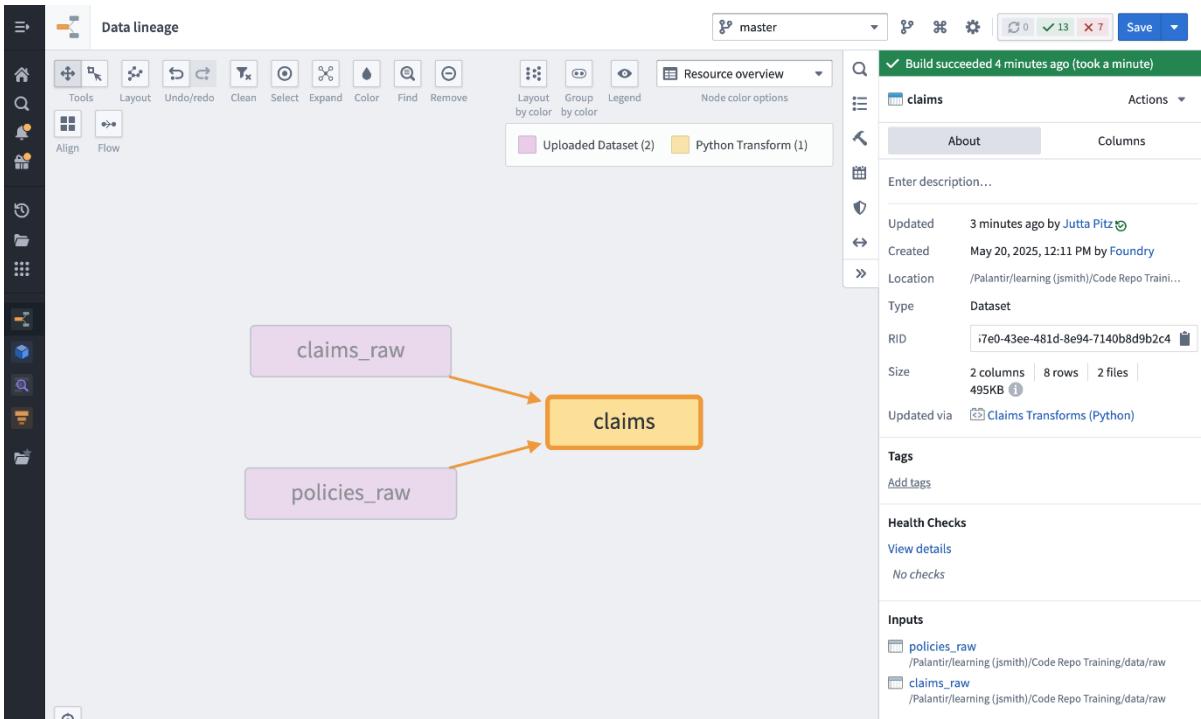
Step 2: View the data lineage

1. Click **claims** with cmd/ctrl to open the newly built dataset in a new tab.

line_of_business	avg_claim_cost
liability	5471.4276136363640
legal	5225.7619607843135
auto	4850.4941346153850
marine	4716.8577142857150
life	4894.0605747126460
pet	4666.1075781250000
aviation	4991.3060465116300
accident	5084.3460839160830

2. Click on **Explore Pipeline** and then **Explore data lineage**.

- The graph shows you your pipeline created with Code Repository where your two input datasets get joined to your output dataset.



Note if you only see the claims box and not its ancestors, you can add them to the graph by clicking the little arrow to the left on the claims box.



Summary

Congratulations, you've completed the Deep Dive: Transformations in Code Repositories! You have created a PySpark Transform including casting and filtering, used joins and aggregations, worked collaboratively with potential colleagues using Branching and seen how Code Repositories interact with other Foundry tools like Data Lineage.