Software engineering best practices for notebooks

Article • 10/15/2024

This article provides a hands-on walkthrough that demonstrates how to apply software engineering best practices to your Azure Databricks notebooks, including version control, code sharing, testing, and optionally continuous integration and continuous delivery or deployment (CI/CD).

In this walkthrough, you will:

- Add notebooks to Azure Databricks Git folders for version control.
- Extract portions of code from one of the notebooks into a shareable module.
- Test the shared code.
- Run the notebooks from an Azure Databricks job.
- Optionally apply CI/CD to the shared code.

Requirements

To complete this walkthrough, you must provide the following resources:

- A remote repository with a Git provider that Databricks supports. This article's walkthrough uses GitHub. This walkthrough assumes that you have a GitHub repository named `best-notebooks` available. (You can give your repository a different name. If you do, replace `best-notebooks` with your repo's name throughout this walkthrough.) Create a GitHub repo if you do not already have one.

  > **Note**
  >
  > If you create a new repo, be sure to initialize the repository with at least one file, for example a `README` file.

- An Azure Databricks workspace. Create a workspace if you do not already have one.
- An Azure Databricks all-purpose cluster in the workspace. To run notebooks during the design phase, you attach the notebooks to a running all-purpose cluster. Later on, this walkthrough uses an Azure Databricks job to automate running the notebooks on

this cluster. (You can also run jobs on job clusters that exist only for the jobs' lifetimes.) Create an all-purpose cluster if you do not already have one.

Step 1: Set up Databricks Git folders

In this step, you connect your existing GitHub repo to Azure Databricks Git folders in your existing Azure Databricks workspace.

To enable your workspace to connect to your GitHub repo, you must first provide your workspace with your GitHub credentials, if you have not done so already.

Step 1.1: Provide your GitHub credentials

1. Click your username at the top right of the workspace, and then click **Settings** in the dropdown list.
2. In the **Settings** sidebar, under **User**, click **Linked accounts**.
3. Under **Git integration**, for **Git provider**, select **GitHub**.
4. Click **Personal access token**.
5. For **Git provider username or email**, enter your GitHub username.
6. For **Token**, enter your GitHub personal access token (classic) . This personal access token (classic) must have the **repo** and **workflow** permissions.
7. Click **Save**.

Step 1.2: Connect to your GitHub repo

1. On the workspace sidebar, click **Workspace**.
2. In the **Workspace** browser, expand **Workspace** > **Users**.
3. Right-click your username folder, and then click **Create** > **Git folder**.
4. In the **Create Git folder** dialog:
   a. For **Git repository URL**, enter the GitHub Clone with HTTPS URL for your GitHub repo. This article assumes that your URL ends with `best-notebooks.git`, for example `https://github.com/<your-GitHub-username>/best-notebooks.git`.
   b. For **Git provider**, select **GitHub**.
   c. Leave **Git folder name** set to the name of your repo, for example `best-notebooks`.
   d. Click **Create Git folder**.

Step 2: Import and run the notebook

In this step, you import an existing external notebook into your repo. You could create your own notebooks for this walkthrough, but to speed things up we provide them for you here.

Step 2.1: Create a working branch in the repo

In this substep, you create a branch named `eda` in your repo. This branch enables you to work on files and code independently from your repo's `main` branch, which is a software engineering best practice. (You can give your branch a different name.)

> **Note**
>
> In some repos, the `main` branch may be named `master` instead. If so, replace `main` with `master` throughout this walkthrough.

> **Tip**
>
> If you're not familiar with working in Git branches, see Git Branches - Branches in a Nutshell on the Git website.

1. The Git folder from Step 1.2 should be open. If not, then in the **Workspace** sidebar, expand **Workspace** > **Users**, then expand your username folder, and click your Git folder.
2. Next to the folder name under the workspace navigation breadcrumb, click the **main** Git branch button.
3. In the **best-notebooks** dialog, click the **Create branch** button.

   > **Note**
   >
   > If your repo has a name other than `best-notebooks`, this dialog's title will be different, here and throughout this walkthrough.

4. Enter `eda`, and click **Create**.
5. Close this dialog.

Step 2.2: Import the notebook into the repo

In this substep, you import an existing notebook from another repo into your repo. This notebook does the following:

- Copies a CSV file from the owid/covid-19-data GitHub repository onto a cluster in your workspace. This CSV file contains public data about COVID-19 hospitalizations and intensive care metrics from around the world.
- Reads the CSV file's contents into a pandas DataFrame.
- Filters the data to contain metrics from only the United States.
- Displays a plot of the data.
- Saves the pandas DataFrame as a Pandas API on Spark DataFrame.
- Performs data cleansing on the Pandas API on Spark DataFrame.
- Writes the Pandas API on Spark DataFrame as a Delta table in your workspace.
- Displays the Delta table's contents.

While you could create your own notebook in your repo here, importing an existing notebook instead helps to speed up this walkthrough. To create a notebook in this branch or move an existing notebook into this branch instead of importing a notebook, see Workspace files basic usage.

1. From the **best-notebooks** Git folder, click **Create** > **Folder**.
2. In the **New folder** dialog, enter `notebooks`, and then click **Create**.
3. From the **notebooks** folder, click the kebab, then **Import**.
4. In the **Import** dialog:

   a. For **Import from**, select **URL**.

   b. Enter the URL to the raw contents of the `covid_eda_raw` notebook in the `databricks/notebook-best-practices` repo in GitHub. To get this URL: i. Go to https://github.com/databricks/notebook-best-practices. ii. Click the `notebooks` folder. iii. Click the `covid_eda_raw.py` file. iv. Click **Raw**. v. Copy the full URL from your web browser's address bar over into the **Import** dialog.

   > **Note**
   >
   > The **Import** dialog works with Git URLs for public repositories only.

   c. Click **Import**.

Step 2.3: Run the notebook

1. If the notebook is not already showing, open the **notebooks** folder, and then click the **covid_eda_raw** notebook inside of the folder.
2. Select the cluster to attach this notebook to. For instructions on creating a cluster, see Create a cluster.
3. Click **Run All**.
4. Wait while the notebook runs.

After the notebook finishes running, in the notebook you should see a plot of the data as well as over 600 rows of raw data in the Delta table. If the cluster was not already running when you started running this notebook, it could take several minutes for the cluster to start up before displaying the results.

Step 2.4: Check in and merge the notebook

In this substep, you save your work so far to your GitHub repo. You then merge the notebook from your working branch into your repo's `main` branch.

1. Next to the notebook's name, click the **eda** Git branch button.
2. In the **best-notebooks** dialog, on the **Changes** tab, make sure the **notebooks/covid_eda_raw.py** file is selected.
3. For **Commit message (required)**, enter `Added raw notebook.`
4. For **Description (optional)**, enter `This is the first version of the notebook.`
5. Click **Commit & Push**.
6. Click the pull request link in **Create a pull request on your git provider** in the banner.
7. In GitHub, create the pull request, and then merge the pull request into the `main` branch.
8. Back in your Azure Databricks workspace, close the **best-notebooks** dialog if it is still showing.

Step 3: Move code into a shared module

In this step, you move some of the code in your notebook into a set of shared functions outside of your notebook. This enables you to use these functions with other similar notebooks, which can speed up future coding and help ensure more predictable and consistent notebook results. Sharing this code also enables you to more easily test these

functions, which as a software engineering best practice can raise the overall quality of your code as you go.

Step 3.1: Create another working branch in the repo

1. Next to the notebook's name, click the **eda** Git branch button.
2. In the **best-notebooks** dialog, click the drop-down arrow next to the **eda** branch, and select **main**.
3. Click the **Pull** button. If prompted to proceed with pulling, click **Confirm**.
4. Click the **Create Branch** button.
5. Enter `first_modules`, and then click **Create**. (You can give your branch a different name.)
6. Close this dialog.

Step 3.2: Import the notebook into the repo

To speed up this walkthrough, in this substep you import another existing notebook into your repo. This notebook does the same things as the previous notebook, except this notebook will call shared code functions that are stored outside of the notebook. Again, you could create your own notebook in your repo here and do the actual code sharing yourself.

1. From the **Workspace** browser, right-click the **notebooks** folder, and then click **Import**.
2. In the **Import** dialog:

   a. For **Import from**, select **URL**.

   b. Enter the URL to the raw contents of the `covid_eda_modular` notebook in the `databricks/notebook-best-practices` repo in GitHub. To get this URL: i. Go to https://github.com/databricks/notebook-best-practices . ii. Click the `notebooks` folder. iii. Click the `covid_eda_modular.py` file. iv. Click **Raw**. v. Copy the full URL from your web browser's address bar over into the **Import Notebooks** dialog.

   > **Note**
   >
   > The **Import Notebooks** dialog works with Git URLs for public repositories only.

c. Click **Import**.

Step 3.3: Add the notebook's supporting shared code functions

1. From the **Workspace** browser, right-click the **best-notebooks** Git folder, and then click **Create** > **Folder**.
2. In the **New folder** dialog, enter `covid_analysis`, and then click **Create**.
3. From the **covid_analysis** folder click **Create** > **File**.
4. In the **New File Name** dialog, enter `transforms.py`, and then click **Create File**.
5. In the **transforms.py** editor window, enter the following code:

Python

```
import pandas as pd

# Filter by country code.
def filter_country(pdf, country="USA"):
  pdf = pdf[pdf.iso_code == country]
  return pdf

# Pivot by indicator, and fill missing values.
def pivot_and_clean(pdf, fillna):
  pdf["value"] = pd.to_numeric(pdf["value"])
  pdf = pdf.fillna(fillna).pivot_table(
    values="value", columns="indicator", index="date"
  )
  return pdf

# Create column names that are compatible with Delta tables.
def clean_spark_cols(pdf):
  pdf.columns = pdf.columns.str.replace(" ", "_")
  return pdf

# Convert index to column (works with pandas API on Spark, too).
def index_to_col(df, colname):
  df[colname] = df.index
  return df
```

Tip

For other code sharing techniques, see Share code between Databricks notebooks.

Step 3.4: Add the shared code's dependencies

The preceding code has several Python package dependencies to enable the code to run properly. In this substep, you declare these package dependencies. Declaring dependencies improves reproducibility by using precisely defined versions of libraries.

1. From the **Workspace** browser, right-click the **best-notebooks** Git folder, and then click **Create** > **File**.

   > **Note**
   >
   > You want the file that lists package dependencies to go into the Git folder's root, not into the **notebooks** or **covid_analysis** folders.

2. In the **New File Name** dialog, enter `requirements.txt`, and then click **Create File**.

3. In the **requirements.txt** editor window, enter the following code:

   > **Note**
   >
   > If the `requirements.txt` file is not visible, you may need to refresh your web browser.

```
-i https://pypi.org/simple
attrs==21.4.0
cycler==0.11.0
fonttools==4.33.3
iniconfig==1.1.1
kiwisolver==1.4.2
matplotlib==3.5.1
numpy==1.22.3
packaging==21.3
pandas==1.4.2
pillow==9.1.0
pluggy==1.0.0
py==1.11.0
py4j==0.10.9.3
```

```
pyarrow==7.0.0
pyparsing==3.0.8
pyspark==3.2.1
pytest==7.1.2
python-dateutil==2.8.2
pytz==2022.1
six==1.16.0
tomli==2.0.1
wget==3.2
```

> **Note**
>
> The preceding file lists specific package versions. For better compatibility, you can cross-reference these versions with the ones that are installed on your all-purpose cluster. See the "System environment" section for your cluster's Databricks Runtime version in Databricks Runtime release notes versions and compatibility.

Your repo structure should now look like this:

```
|-- covid_analysis
|   └── transforms.py
├── notebooks
|   ├── covid_eda_modular
|   └── covid_eda_raw (optional)
└── requirements.txt
```

Step 3.5: Run the refactored notebook

In this substep, you run the `covid_eda_modular` notebook, which calls the shared code in `covid_analysis/transforms.py`.

1. From the **Workspace** browser, click the **covid_eda_modular** notebook inside the **notebooks** folder.
2. Select the cluster to attach this notebook to.
3. Click **Run All**.
4. Wait while the notebook runs.

After the notebook finishes running, in the notebook you should see similar results as the `covid_eda_raw` notebook: a plot of the data as well as over 600 rows of raw data in the Delta table. The main difference with this notebook is that a different filter is used (an `iso_code` of `DZA` instead of `USA`). If the cluster was not already running when you started running this notebook, it could take several minutes for the cluster to start up before displaying the results.

Step 3.6: Check in the notebook and its related code

1. Next to the notebook's name, click the **first_modules** Git branch button.
2. In the **best-notebooks** dialog, on the **Changes** tab, make sure the following are selected:
   - **requirements.txt**
   - **covid_analysis/transforms.py**
   - **notebooks/covid_eda_modular.py**
3. For **Commit message (required)**, enter `Added refactored notebook.`
4. For **Description (optional)**, enter `This is the second version of the notebook.`
5. Click **Commit & Push**.
6. Click the pull request link in **Create a pull request on your git provider** in the banner.
7. In GitHub, create the pull request, and then merge the pull request into the `main` branch.
8. Back in your Azure Databricks workspace, close the **best-notebooks** dialog if it is still showing.

Step 4: Test the shared code

In this step, you test the shared code from the last step. However, you want to test this code without running the `covid_eda_modular` notebook itself. This is because if the shared code fails to run, the notebook itself would likely fail to run as well. You want to catch failures in your shared code first before having your main notebook eventually fail later. This testing technique is a software engineering best practice.

> **Tip**

For additional approaches to testing for notebooks, as well as testing for R and Scala notebooks, see Unit testing for notebooks.

Step 4.1: Create another working branch in the repo

1. Next to the notebook's name, click the **first_modules** Git branch button.
2. In the **best-notebooks** dialog, click the drop-down arrow next to the **first_modules** branch, and select **main**.
3. Click the **Pull** button. If prompted to proceed with pulling, click **Confirm**.
4. Click **Create Branch**.
5. Enter `first_tests`, and then click **Create**. (You can give your branch a different name.)
6. Close this dialog.

Step 4.2: Add the tests

In this substep, you use the pytest framework to test your shared code. In these tests, you assert whether particular test results are achieved. If any test produces an unexpected result, that particular test fails the assertion and thus the test itself fails.

1. From the **Workspace** browser, right-click your Git folder, and then click **Create** > **Folder**.
2. In the **New folder** dialog, enter `tests`, and then click **Create**.
3. From the **tests** folder, click **Create** > **File**.
4. In the **New File Name** dialog, enter `testdata.csv`, and then click **Create File**.
5. In **testdata.csv** editor window, enter the following test data:

```
entity,iso_code,date,indicator,value
United States,USA,2022-04-17,Daily ICU occupancy,
United States,USA,2022-04-17,Daily ICU occupancy per million,4.1
United States,USA,2022-04-17,Daily hospital occupancy,10000
United States,USA,2022-04-17,Daily hospital occupancy per million,30.3
United States,USA,2022-04-17,Weekly new hospital admissions,11000
United States,USA,2022-04-17,Weekly new hospital admissions per
million,32.8
Algeria,DZA,2022-04-18,Daily ICU occupancy,1010
Algeria,DZA,2022-04-18,Daily ICU occupancy per million,4.5
```

```
Algeria,DZA,2022-04-18,Daily hospital occupancy,11000
Algeria,DZA,2022-04-18,Daily hospital occupancy per million,30.9
Algeria,DZA,2022-04-18,Weekly new hospital admissions,10000
Algeria,DZA,2022-04-18,Weekly new hospital admissions per million,32.1
```

6. From the **tests** folder, click **Create** > **File**.

7. In the **New File Name** dialog, enter `transforms_test.py`, and then click **Create File**.

8. In **transforms_test.py** editor window, enter the following test code. These tests use standard `pytest` fixtures as well as a mocked in-memory pandas DataFrame:

Python

```python
# Test each of the transform functions.
import pytest
from textwrap import fill
import os
import pandas as pd
import numpy as np
from covid_analysis.transforms import *
from pyspark.sql import SparkSession

@pytest.fixture
def raw_input_df() -> pd.DataFrame:
  """
  Create a basic version of the input dataset for testing, including NaNs.
  """
  return pd.read_csv('tests/testdata.csv')

@pytest.fixture
def colnames_df() -> pd.DataFrame:
  df = pd.DataFrame(
    data=[[0,1,2,3,4,5]],
    columns=[
      "Daily ICU occupancy",
      "Daily ICU occupancy per million",
      "Daily hospital occupancy",
      "Daily hospital occupancy per million",
      "Weekly new hospital admissions",
      "Weekly new hospital admissions per million"
    ]
  )
  return df

# Make sure the filter works as expected.
def test_filter(raw_input_df):
  filtered = filter_country(raw_input_df)
  assert filtered.iso_code.drop_duplicates()[0] == "USA"
```

```
# The test data has NaNs for Daily ICU occupancy; this should get filled
to 0.
def test_pivot(raw_input_df):
  pivoted = pivot_and_clean(raw_input_df, 0)
  assert pivoted["Daily ICU occupancy"][0] == 0

# Test column cleaning.
def test_clean_cols(colnames_df):
  cleaned = clean_spark_cols(colnames_df)
  cols_w_spaces = cleaned.filter(regex=(" "))
  assert cols_w_spaces.empty == True

# Test column creation from index.
def test_index_to_col(raw_input_df):
  raw_input_df["col_from_index"] = raw_input_df.index
  assert (raw_input_df.index == raw_input_df.col_from_index).all()
```

Your repo structure should now look like this:

```
├── covid_analysis
│   └── transforms.py
├── notebooks
│   ├── covid_eda_modular
│   └── covid_eda_raw (optional)
├── requirements.txt
└── tests
    ├── testdata.csv
    └── transforms_test.py
```

Step 4.3: Run the tests

To speed up this walkthrough, in this substep you use an imported notebook to run the preceding tests. This notebook downloads and installs the tests' dependent Python packages into your workspace, runs the tests, and reports the tests' results. While you could run `pytest` from your cluster's web terminal, running `pytest` from a notebook can be more convenient.

> **Note**

Running `pytest` runs all files whose names follow the form `test_*.py` or `/*_test.py` in the current directory and its subdirectories.

1. From the **Workspace** browser, right-click the **notebooks** folder, and then click **Import**.
2. In the **Import Notebooks** dialog:

   a. For **Import from**, select **URL**.

   b. Enter the URL to the raw contents of the `run_unit_tests` notebook in the `databricks/notebook-best-practices` repo in GitHub. To get this URL: i. Go to https://github.com/databricks/notebook-best-practices . ii. Click the `notebooks` folder. iii. Click the `run_unit_tests.py` file. iv. Click **Raw**. v. Copy the full URL from your web browser's address bar over into the **Import Notebooks** dialog.

   > **Note**
   >
   > The **Import Notebooks** dialog works with Git URLs for public repositories only.

   c. Click **Import**.
3. Select the cluster to attach this notebook to.
4. Click **Run All**.
5. Wait while the notebook runs.

After the notebook finishes running, in the notebook you should see information about the number of passing and failed tests, along with other related details. If the cluster was not already running when you started running this notebook, it could take several minutes for the cluster to start up before displaying the results.

Your repo structure should now look like this:

```
├── covid_analysis
│   └── transforms.py
├── notebooks
│   ├── covid_eda_modular
│   ├── covid_eda_raw (optional)
│   └── run_unit_tests
├── requirements.txt
└── tests
```

```
    ├── testdata.csv
    └── transforms_test.py
```

Step 4.4: Check in the notebook and related tests

1. Next to the notebook's name, click the **first_tests** Git branch button.
2. In the **best-notebooks** dialog, on the **Changes** tab, make sure the following are selected:
   - **tests/transforms_test.py**
   - **notebooks/run_unit_tests.py**
   - **tests/testdata.csv**
3. For **Commit message (required)**, enter `Added tests.`
4. For **Description (optional)**, enter `These are the unit tests for the shared code..`
5. Click **Commit & Push**.
6. Click the pull request link in **Create a pull request on your git provider** in the banner.
7. In GitHub, create the pull request, and then merge the pull request into the `main` branch.
8. Back in your Azure Databricks workspace, close the **best-notebooks** dialog if it is still showing.

Step 5: Create a job to run the notebooks

In previous steps, you tested your shared code manually and ran your notebooks manually. In this step, you use an Azure Databricks job to test your shared code and run your notebooks automatically, either on-demand or on a regular schedule.

Step 5.1: Create a job task to run the testing notebook

1. On the workspace sidebar, click **Workflows**.
2. On the **Jobs** tab, click **Create Job**.
3. Edit the name of the job to be `covid_report`.
4. For **Task name**, enter `run_notebook_tests`.
5. For **Type**, select **Notebook**.
6. For **Source**, select **Git provider**.
7. Click **Add a git reference**.

8. In the **Git information** dialog:
   a. For **Git repository URL**, enter the GitHub Clone with HTTPS URL for your GitHub repo. This article assumes that your URL ends with `best-notebooks.git`, for example `https://github.com/<your-GitHub-username>/best-notebooks.git`.
   b. For **Git provider**, select **GitHub**.
   c. For **Git reference (branch / tag / commit)**, enter `main`.
   d. Next to **Git reference (branch / tag / commit)**, select **branch**.
   e. Click **Confirm**.
9. For **Path**, enter `notebooks/run_unit_tests`. Do not add the `.py` file extension.
10. For **Cluster**, select the cluster from the previous step.
11. Click **Create task**.

> **Note**
>
> In this scenario, Databricks does not recommend that you use the schedule button in the notebook as described in Create and manage scheduled notebook jobs to schedule a job to run this notebook periodically. This is because the schedule button creates a job by using the latest *working* copy of the notebook in the workspace repo. Instead, Databricks recommends that you follow the preceding instructions to create a job that uses the latest *committed* version of the notebook in the repo.

Step 5.2: Create a job task to run the main notebook

1. Click the + **Add task** icon.
2. A pop-up menu appears. Select **Notebook**.
3. For **Task name**, enter `run_main_notebook`.
4. For **Type**, select **Notebook**.
5. For **Path**, enter `notebooks/covid_eda_modular`. Do not add the `.py` file extension.
6. For **Cluster**, select the cluster from the previous step.
7. Verify **Depends on** value is `run_notebook-tests`.
8. Click **Create task**.

Step 5.3 Run the job

1. Click **Run now**.
2. In the pop-up, click **View run**.

> **Note**
>
> If the pop-up disappears too quickly, then do the following:
>
> a. On the sidebar in the **Data Science & Engineering** or **Databricks Mosaic AI** environment, click **Workflows**.
>
> b. On the **Job runs** tab, click the **Start time** value for the latest job with **covid_report** in the **Jobs** column.

3. To see the job results, click on the **run_notebook_tests** tile, the **run_main_notebook** tile, or both. The results on each tile are the same as if you ran the notebooks yourself, one by one.

> **Note**
>
> This job ran on-demand. To set up this job to run on a regular basis, see Trigger types for Databricks Jobs.

(Optional) Step 6: Set up the repo to test the code and run the notebook automatically whenever the code changes

In the previous step, you used a job to automatically test your shared code and run your notebooks at a point in time or on a recurring basis. However, you may prefer to trigger tests automatically when changes are merged into your GitHub repo, using a CI/CD tool such as GitHub Actions .

Step 6.1: Set up GitHub access to your workspace

In this substep, you set up a GitHub Actions workflow that runs jobs in the workspace whenever changes are merged into your repository. You do this by giving GitHub a unique Azure Databricks token for access.

For security reasons, Databricks discourages you from giving your Azure Databricks workspace user's personal access token to GitHub. Instead, Databricks recommends that you give GitHub a Microsoft Entra ID token that is associated with a Microsoft Entra ID

service principal. For instructions, see the Azure section of the Run Databricks Notebook GitHub Action page in the GitHub Actions Marketplace.

> **Important**
>
> Notebooks are run with all of the workspace permissions of the identity that is associated with the token, so Databricks recommends using a service principal. If you really want to give your Azure Databricks workspace user's personal access token to GitHub for personal exploration purposes only, and you understand that for security reasons Databricks discourages this practice, see the instructions to **create your workspace user's personal access token**.

Step 6.2: Add the GitHub Actions workflow

In this substep, you add a GitHub Actions workflow to run the `run_unit_tests` notebook whenever there is a pull request to the repo.

This substep stores the GitHub Actions workflow in a file that is stored within multiple folder levels in your GitHub repo. GitHub Actions requires a specific nested folder hierarchy to exist in your repo in order to work properly. To complete this step, you must use the website for your GitHub repo, because the Azure Databricks Git folder user interface does not support creating nested folder hierarchies.

1. In the website for your GitHub repo, click the **Code** tab.
2. Click the arrow next to **main** to expand the **Switch branches or tags** drop-down list.
3. In the **Find or create a branch** box, enter `adding_github_actions`.
4. Click **Create branch: adding_github_actions from 'main'**.
5. Click **Add file** > **Create new file**.
6. For **Name your file**, enter `.github/workflows/databricks_pull_request_tests.yml`.
7. In the editor window, enter the following code. This code uses the pull_request hook from the Run Databricks Notebook GitHub Action to run the `run_unit_tests` notebook.

   In the following code, replace:

- `<your-workspace-instance-URL>` with your Azure Databricks instance name.
- `<your-access-token>` with the token that you generated earlier.
- `<your-cluster-id>` with your target cluster ID.

YAML

```yaml
name: Run pre-merge Databricks tests

on:
  pull_request:

env:
  # Replace this value with your workspace instance name.
  DATABRICKS_HOST: https://<your-workspace-instance-name>

jobs:
  unit-test-notebook:
    runs-on: ubuntu-latest
    timeout-minutes: 15

    steps:
      - name: Checkout repo
        uses: actions/checkout@v2
      - name: Run test notebook
        uses: databricks/run-notebook@main
        with:
          databricks-token: <your-access-token>

          local-notebook-path: notebooks/run_unit_tests.py

          existing-cluster-id: <your-cluster-id>

          git-commit: "${{ github.event.pull_request.head.sha }}"

          # Grant all users view permission on the notebook's results, so
that they can
          # see the result of the notebook, if they have related access
permissions.
          access-control-list-json: >
            [
              {
                "group_name": "users",
                "permission_level": "CAN_VIEW"
              }
            ]
          run-name: "EDA transforms helper module unit tests"
```

8. Click **Commit changes**.

9. In the **Commit changes** dialog, enter `Create databricks_pull_request_tests.yml` into **Commit message**
10. Select **Commit directly to the adding_github_actions branch** and click **Commit changes**.
11. On the **Code** tab, click **Compare & pull request**, and then create the pull request.
12. On the pull request page, wait for the icon next to **Run pre-merge Databricks tests / unit-test-notebook (pull_request)** to display a green check mark. (It may take a few moments for the icon to appear.) If there is a red X instead of a green check mark, click **Details** to find out why. If the icon or **Details** are no longer showing, click **Show all checks**.
13. If the green check mark appears, merge the pull request into the `main` branch.

(Optional) Step 7: Update the shared code in GitHub to trigger tests

In this step, you make a change to the shared code and then push the change into your GitHub repo, which immediately triggers the tests automatically, based on the GitHub Action from the previous step.

Step 7.1: Create another working branch in the repo

1. From the **Workspace** browser, open the **best-notebooks** Git folder.
2. Next to the folder's name, click the **first_tests** Git branch button.
3. In the **best-notebooks** dialog, click the drop-down arrow next to the **first_tests** branch, and select **main**.
4. Click the **Pull** button. If prompted to proceed with pulling, click **Confirm**.
5. Click the + (**Create branch**) button.
6. Enter `trigger_tests`, and then click **Create**. (You can give your branch a different name.)
7. Close this dialog.

Step 7.2: Change the shared code

1. From the **Workspace** browser, in the **best-notebooks** Git folder, click the **covid_analysis/transforms.py** file.
2. Change the third line of this file:

```
Python

# Filter by country code.
```

To this:

```
Python

# Filter by country code. If not specified, use "USA."
```

Step 7.3: Check in the change to trigger the tests

1. Next to the file's name, click the **trigger_tests** Git branch button.
2. In the **best-notebooks** dialog, on the **Changes** tab, make sure **covid_analysis/transforms.py** is selected.
3. For **Commit message (required)**, enter `Updated comment`.
4. For **Description (optional)**, enter `This updates the comment for filter_country.`
5. Click **Commit & Push**.
6. Click the pull request link in **Create a pull request on your git provider** in the banner, and then create the pull request in GitHub.
7. On the pull request page, wait for the icon next to **Run pre-merge Databricks tests / unit-test-notebook (pull_request)** to display a green check mark. (It may take a few moments for the icon to appear.) If there is a red X instead of a green check mark, click **Details** to find out why. If the icon or **Details** are no longer showing, click **Show all checks**.
8. If the green check mark appears, merge the pull request into the `main` branch.

Feedback