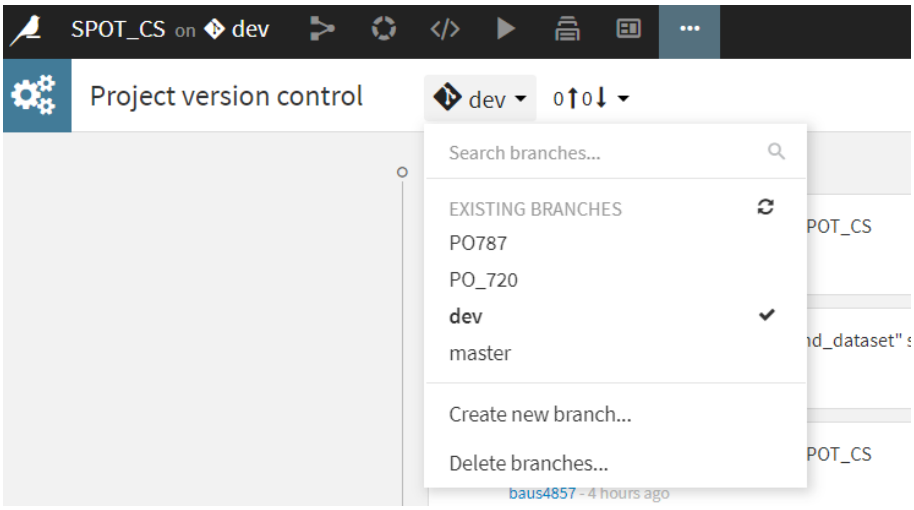
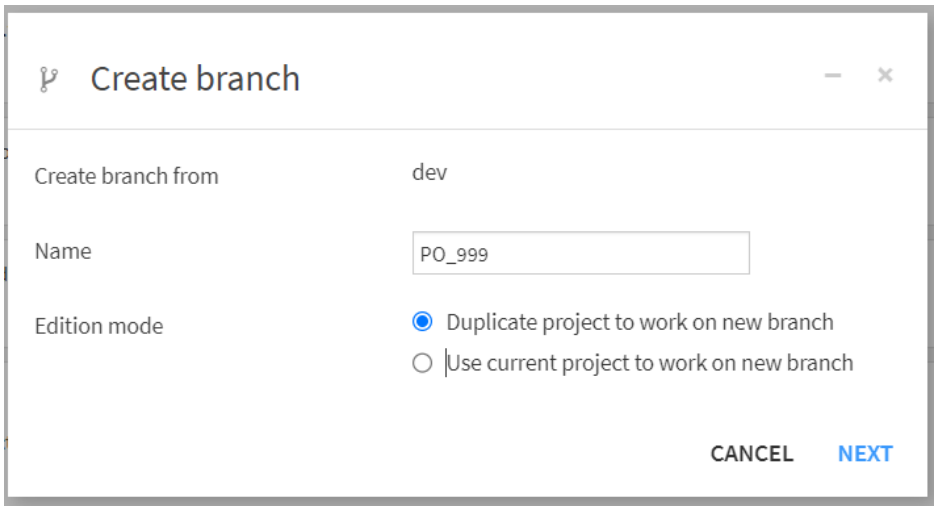
How to : start a new development

Create a dedicated feature branch

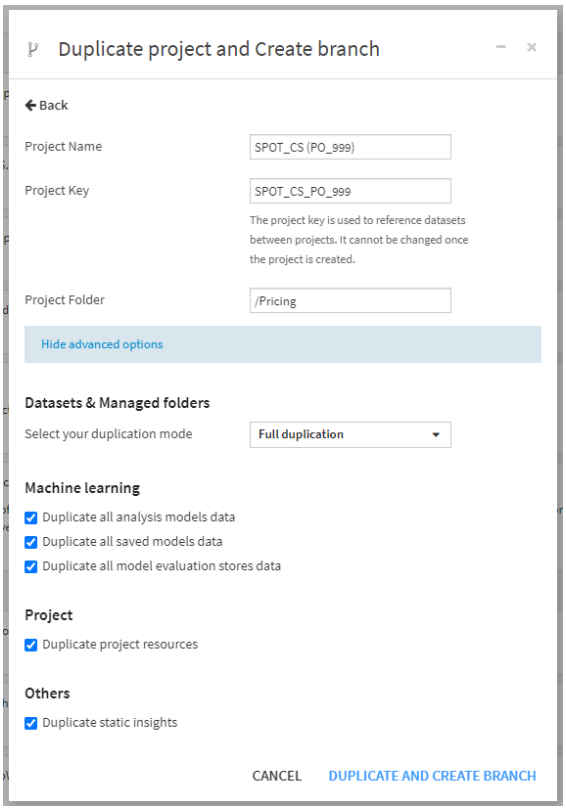
- From the dev project (SPOT_CS), fetch and make sure there is nothing to push/pull.



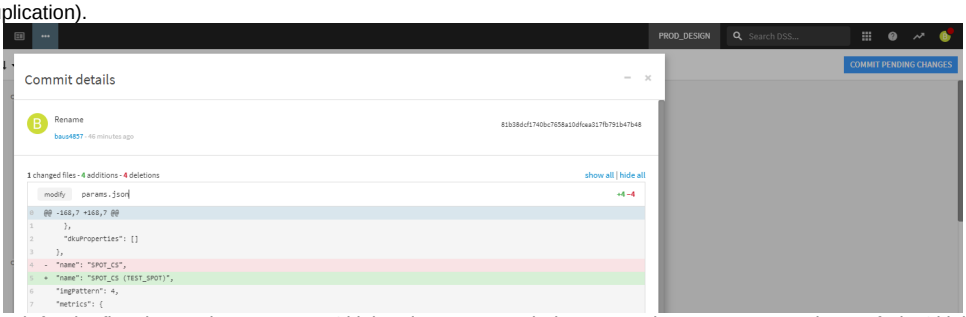
- Create a new branch, chose the "duplicate project" option and name it based on its Jira ticket.



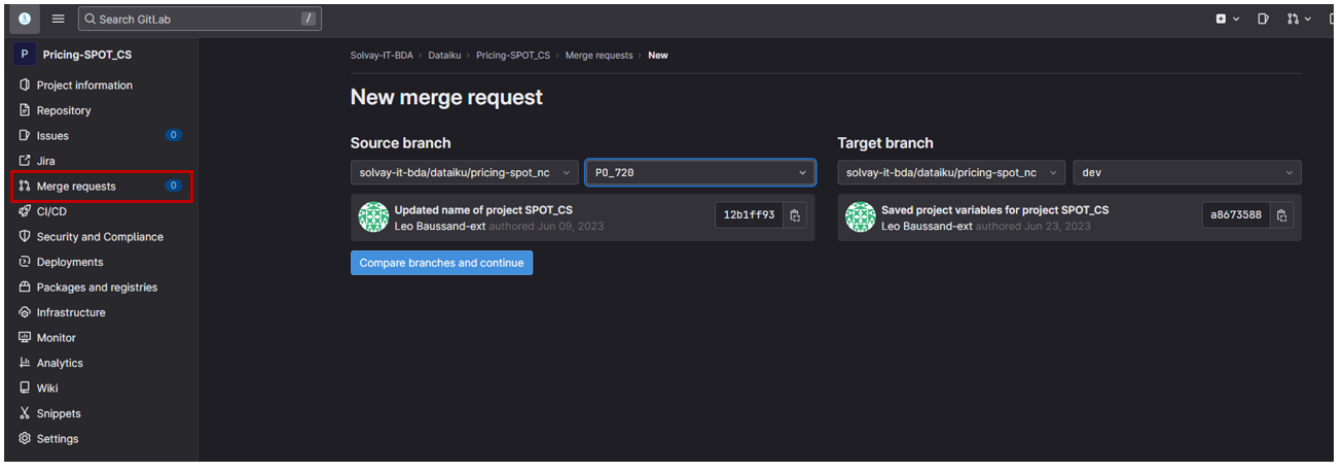
- Store it in the "Pricing" folder and select "full duplication" if you want to retrieve all the intermediate datasets.



- In the version control of the newly created project, do a first commit. It should include as only changes params.json (which is the renaming of the project after the duplication).



- Push your new branch for the first time so it appears on Gitlab. It is recommended to create the merge request in "Draft" in Gitlab directly.

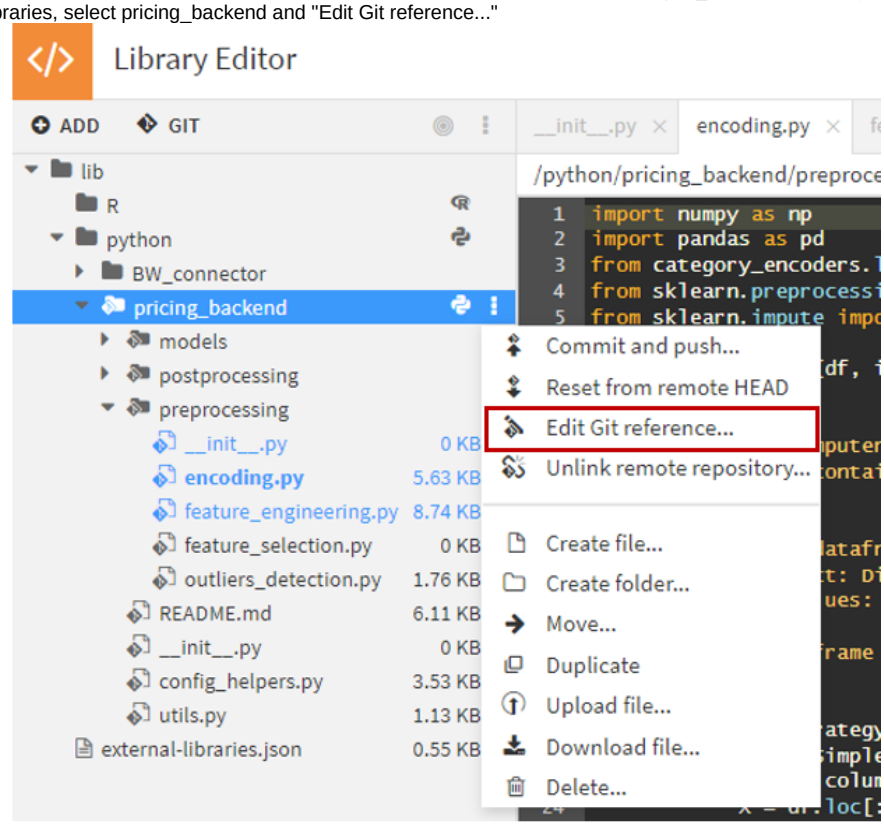


Manage the code-base reference

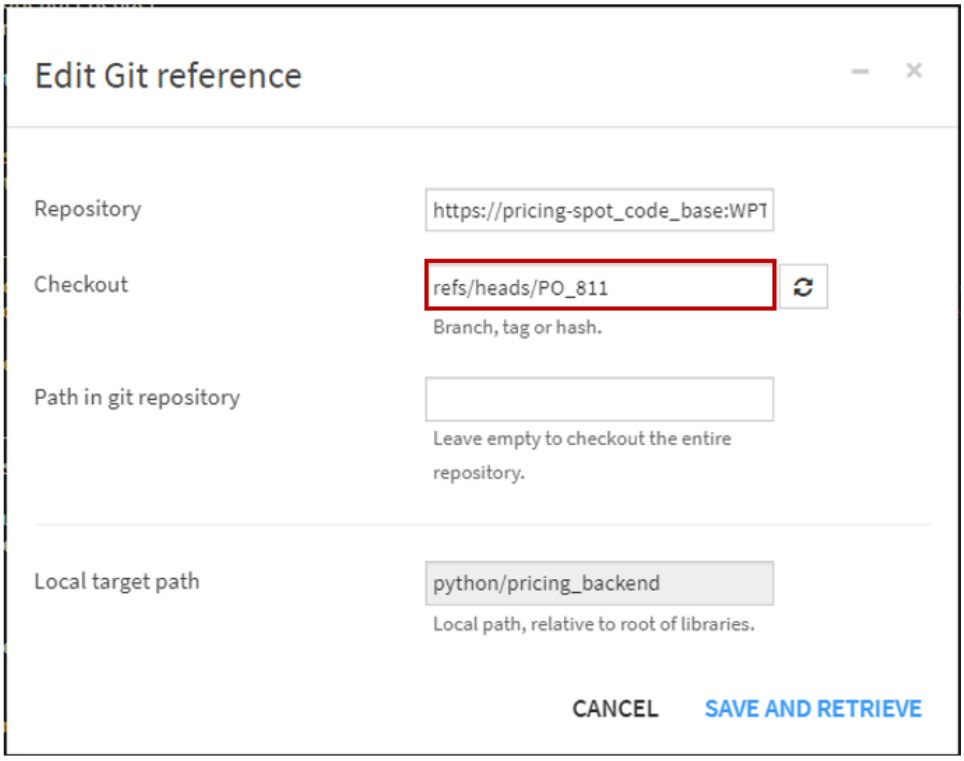
Note : this is only needed if you plan to make changes on the pricing_backend library.

The code base corresponds to all the code that is available in the libraries. As this code is useful for all the GBUs, it is versioned in its own Gitlab repository apart from the Dataiku one. For a new feature development including changes to the code base, you have to also create a branch on this repository to not impact the other working branches.

- In the code base Gitlab repository, create a new branch named the same as your Dataiku branch (PO_xxx where xxx is your Jira ticket number).
- In Dataiku, go to the libraries, select pricing_backend and "Edit Git reference..."



- Change the "Checkout" value from ref/heads/dev to ref/heads/your_branch_name



Develop on the new branch

You can then develop the functionality on the new branch, be careful that any csv stored in the folders of the flow will not be merged back on the dev branch at the end of the development.

Explicit commits are activated, which means that you will have to manually commit for every significant change you made on the Dataiku project.

During the development, it is advised to frequently rebase the dev to stay updated with the latest changes, especially if other branches have been merged on dev in the meantime.

Commit/push your code base changes

Note : this is only needed if you made changes on the pricing_backend library.

Remember to commit and push the changes you have made on the libraries before merging your branch back to dev.

It is also possible to do some rebase if the code base has been updated by another branch feature in the meantime.

Rebase your Dataiku branch

IMPORTANT: NEVER REBASE DEV OR MASTER !!! This only applies to feature branches or hotfixes that we want to merge to dev.

Because dev branch might have changed since you created your branch, you should update your branch as often as possible to include those changes and stay as close as possible to dev.

There are 2 main ways: merge and rebase, with pros and cons for each. In our workflow, we will favor rebasing.

One of the drawbacks of rebasing is to deal with conflicts multiple times with different commits. In order to avoid that we can first squash some commits by using an interactive rebase without changing the starting point.

Rebasing

Rebasing

```
# Clean state
git fetch
git checkout my_branch
git pull --rebase

# Rebase in-place with squash:
# first find forking point between your branch and dev
git merge-base HEAD origin/dev
# this will return a commit number. let's call it base_commit
git rebase -i base_commit
# this opens editor with the list of all your branch commits that are not in dev
# keep the first two as pick
# then squash the others as you want, for example squash all the following together by picking the 3rd commit
and squash for all following
# save and close, a new commit is made: rewrite properly its description with something meaningful, save and
close

# Your branch now has only 3 commits: first 2 automatically done by DSS, the 3rd has all your branch changes.
check status
git status

# Rebase to get dev updates
git fetch
git rebase origin/dev
# deal with potential conflicts up to 3 times only for each commit

# Verify your branch state and synchronize it to remote
git push --force-with-lease
```

Merge the code base

Note : this is only needed if you made changes on the pricing_backend library.

Local standard merge process can be used to bring back the changes from your branch to the dev code base.

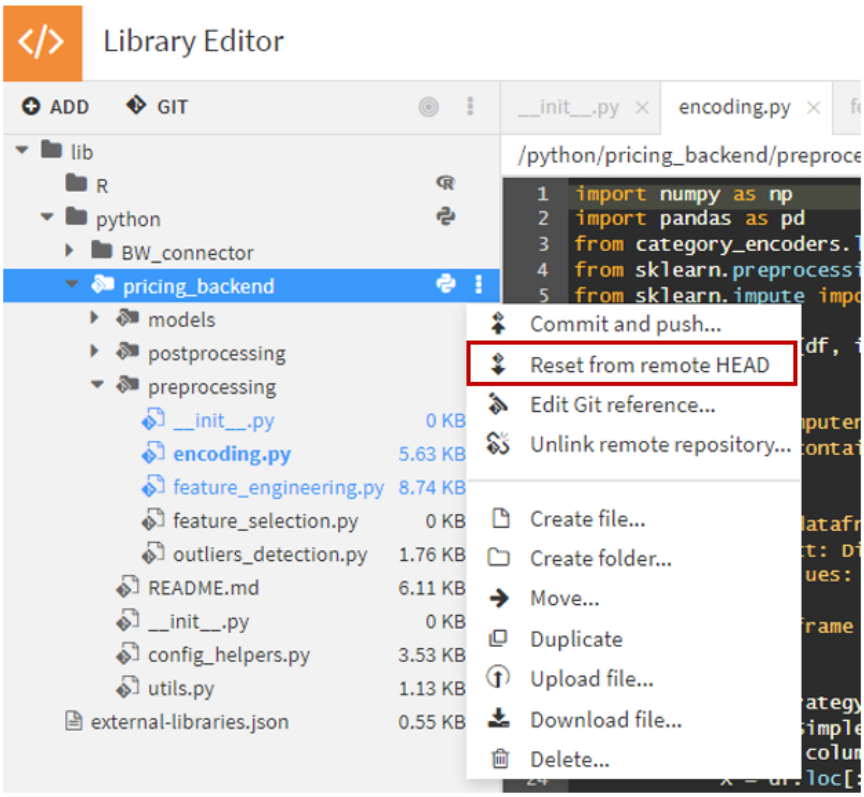
Create the merge request on Gitlab, then follow these instructions :

Make sure there are no pending changes on the libraries of the dev Dataiku project ; if there are, they should be commited and pushed before the following steps.

```
git fetch
git checkout dev
git pull
git merge --no-ff origin/my_branch
git push
```

Potential conflicts will have to be handled.

Afterwards, pull the changes on the Dataiku dev libraries by chosing the "Reset from remote HEAD" option



Merge the Dataiku project back to dev

merging dev to master

Merging dev to master should be done by direct merge for both the Dataiku project and code base :

git fetch
git checkout master
git pull
git merge --no-ff origin/dev
git push

After merging on master, make sure:

- To switch the reference of the library back to master instead of dev
- To re-enable the "Prod scenario" trigger
- To check that the local variables are still set with the "master" env

Merging feature branch to dev:

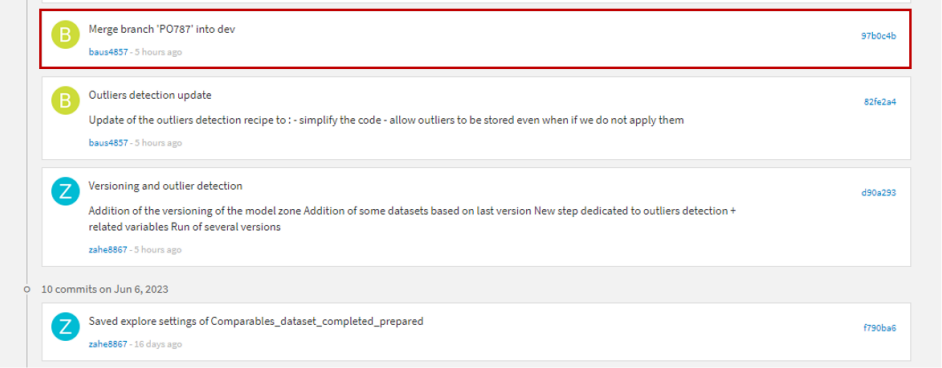
1. Make sure the full scenario has run successfully on your branch
2. If not already done, create a merge request on gitlab : assign to yourself, and add a reviewer
3. Reviewer does the review. Relaunch a scenario if there are any changes.
4. Interactive rebase to **remove the first commits (duplicated project / renaming) automatically generated by Dataiku** when creating the branch : we don't want to keep them because they would overwrite some parameters on dev (project title, key, etc.).

```
git fetch
git checkout my_branch
git pull --rebase
git rebase -i origin/dev
# editor is opened with the list of commits on the branch that are not in origin/master
# remove unwanted lines for commits you don't want to keep (at least the duplicated commit !) , save and
close
# (remark: interactive rebase can also be used to squash together multiple consecutive commits)
#if any conflicts rebase is paused, make sure to fix them: fix file, then
git add my_file_with_conflicts
# make sure currently rebased commit doesn't contain anymore conflicts
git status
# should be all green
# continue rebase
git rebase --continue
#repeat above if conflicts until end of rebase
#make sure branch is where you want it and push the changed branch
git status
git push --force-with-lease
```

5. Merge on target.

```
git checkout dev
git pull --rebase
git merge --no-ff my_branch
git push
```

6. Update the dev project (SPOT_CS) by doing a fetch and pull. Check that you have the latest changes including the merge commit.



7. Finally, **switch the Dataiku project library reference back to the dev** (refs/heads/dev) to be aligned for the project merge.

Clean the branches

1. Verify that the merge request is closed in gitlab.
2. Delete branches (on both repositories : the Dataiku project and code base)

```
git branch -d my_branch  # locally
git push origin --delete my_branch  # on remote
```

3. Delete the Dataiku branch project **(Important : check only the box to delete the managed datasets data, not the managed folders one)**
   As the managed folders used are common between the branches and the dev, deleting it would erase all the versions historized until now.