Lab 8

Enterprise guide to DataOps and MLOps – M.Sc Data Science and Analytics

Third Semester 2023

Lab Date/Time: 12/10/2023

Theory and Lab Faculty: Mr. Kunal Dey

Lab Description

In this lab you will learn about data version control.

Objectives

Lab Tool(s)

DVC
GIT
Any code editor

Lab Deliverables

At the end of the semester, you are required to submit the lab record, before the completion of final CA.

References:

https://github.com/Drakunal/DataOps-and-MLops-Lab

Data Version Control

1- DVC

Data scientists experiment with different versions of code, models, and data. Additionally, version control system like Git to manage the code, track versions, move forward and backward in time, and sharing of the code with various teams is also needed.

The versioning of code is important because it helps reproduce software on a much larger scale. The versioning of data is important is because it helps develop machine learning models with similar metrics at any given time by any developer in any team or organization.

Therefore, it is crucial to version the models as well as data. Git is not capable of storing large data files.

Not only is Git inefficient with larger files, but it is also not a standardized environment for storing large data files. Most data are stored in AWS S3 buckets, Google Cloud Storage, or any institutional remote storage server.

So one of the solutions is to use DVC.

1.1 Using DVC to Push

Installing the necessary package

```
pip install "dvc[all]"
```

This line of code installs the necessary package for doing data version control.
DVC is a system for Data Version Control that works hand in hand with Git to track the data files. It even has a similar syntax like Git so it's quite easy to learn.

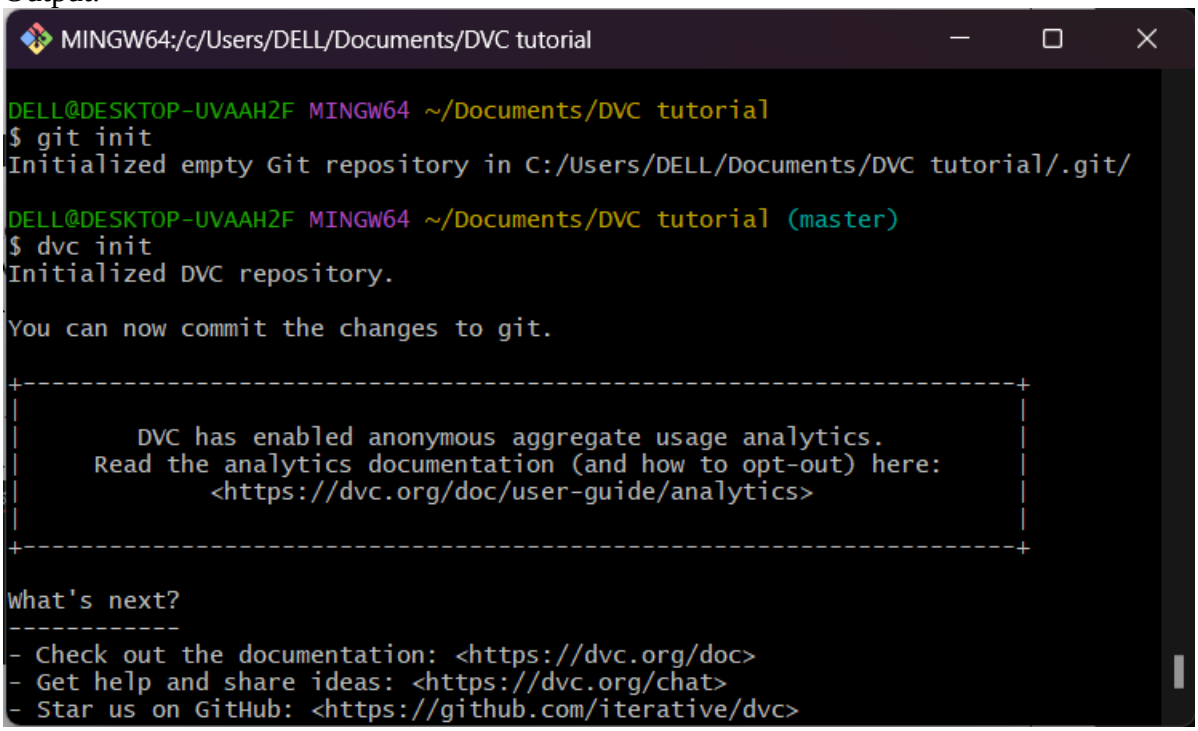
Project Setup

After creation of the project folder, navigate to it using your terminal/cmd.
Initialize a git repository in it and then initialize DVC in that folder.

Code:

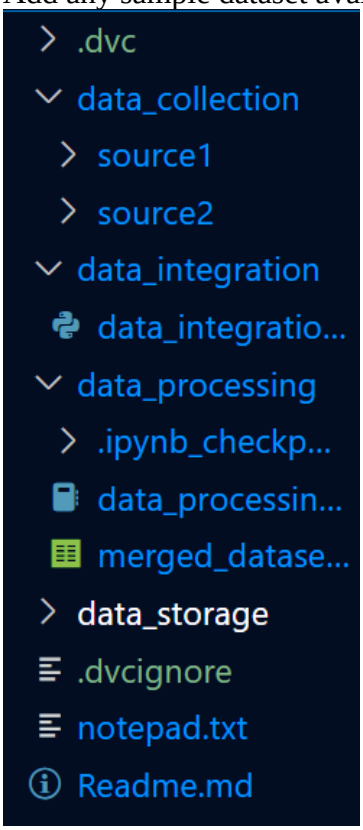
```
git init
dvc init
```

Output:



Dataset Collection

Add any sample dataset available, you can do the full thing in the Lab 1 folder itself.



A size check can be run on the data folders as shown:

```
DELL@DESKTOP-UVAAH2F MINGW64 ~/Documents/DVC tutorial (master)
$ ls -lh data_collection/source1
total 8.0K
-rw-r--r-- 1 DELL 197609 5.0K Jul 11 15:42 iris.csv
```

Here it can be seen that the file size is 5kb.

Adding of file

To add this data file to DVC, run:

```
dvc add data_collection/source1/iris.csv
```

Output:

```
DELL@DESKTOP-UVAAH2F MINGW64 ~/Documents/DVC tutorial (master)
$ dvc add data_collection/source1/iris.csv

To track the changes with git, run:

        git add 'data_collection\source1\iris.csv.dvc' 'data_collection\source1\
.gitignore'

To enable auto staging, run:

        dvc config core.autostage true
```

The output says to run the git command as well, so we need to run it.

Code:

```
git add 'data_collection\source1\iris.csv.dvc'
'data_collection\source1\.gitignore'
```

Now that we've added our new .dvc file to our git tracking, we can go ahead and commit it to our git:

```
$ git commit -m "data added"
[master (root-commit) feb89b7] data added
 5 files changed, 12 insertions(+)
 create mode 100644 .dvc/.gitignore
 create mode 100644 .dvc/config
 create mode 100644 .dvcignore
 create mode 100644 data_collection/source1/.gitignore
 create mode 100644 data_collection/source1/iris.csv.dvc
```

Setting up a remote storage for the data

We can simply utilize Google Drive for storing our versioned datasets and in this tutorial we're going to do exactly that.

Let's create a new folder in our Google drive and look at its URL:

https://drive.google.com/drive/u/0/folders/**cVtFRMoZKxe5iNMd-K_T50Ie**

Highlighted in bold is the ID of the folder that we want to copy to our terminal so that DVC can track our data in that newly created Drive folder.

Code:

```
DELL@DESKTOP-UVAAH2F MINGW64 ~/Documents/DVC tutorial (master)
$ dvc remote add -d storage gdrive://1pYLcz2G6rKoQmACv1OcCPRosW4yRKW3l
Setting 'storage' as a default remote.
```

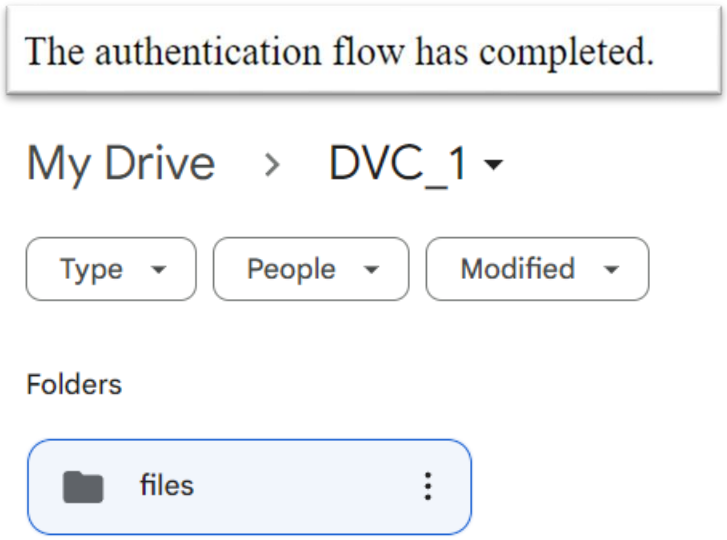
Committing the changes to git:

```
DELL@DESKTOP-UVAAH2F MINGW64 ~/Documents/DVC tutorial (master)
$ git commit .dvc/config -m "Configured remote storage."
[master f58b82e] Configured remote storage.
 1 file changed, 4 insertions(+)
```

Pushing the data to remote server:

```
DELL@DESKTOP-UVAAH2F MINGW64 ~/Documents/DVC tutorial (master)
$ dvc push
```

Output after you sign in with your account:



1.2 Using DVC to Pull

Deletion of files

If someone or the team members want to access the remotely stored data, it can be done with the pull command.

We are deleting the original data and its cache stored locally so that we can pull it from remote. (You need not do it when you are already collaborating, if any change happens in the data, it will be pulled automatically)

```
$ rm -f data_collection/source1/iris.csv
$ rm -rf .dvc/cache
```

Pull

Code:

```
dvc pull
```

```
DELL@DESKTOP-UVAAH2F MINGW64 ~/Documents/DVC tutorial (master)
$ dvc pull
A       data_collection\source1\iris.csv
1 file added and 1 file fetched
```

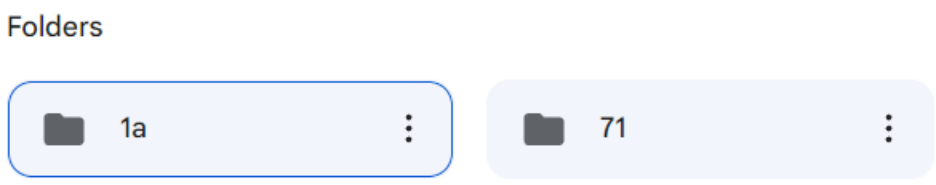
1.3 Tracking a different version of data

A new version of the same data file can also be tracked easily. It can be added to dvc and git again:

```
DELL@DESKTOP-UVAAH2F MINGW64 ~/Documents/DVC tutorial (master)
$ dvc add data_collection/source1/iris.csv

To track the changes with git, run:

        git add 'data_collection\source1\iris.csv.dvc'

To enable auto staging, run:

        dvc config core.autostage true

DELL@DESKTOP-UVAAH2F MINGW64 ~/Documents/DVC tutorial (master)
$ ^C

DELL@DESKTOP-UVAAH2F MINGW64 ~/Documents/DVC tutorial (master)
$ git add 'data_collection\source1\iris.csv.dvc'

DELL@DESKTOP-UVAAH2F MINGW64 ~/Documents/DVC tutorial (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   data_collection/source1/iris.csv.dvc
```

Commit and PUSH

```
DELL@DESKTOP-UVAAH2F MINGW64 ~/Documents/DVC tutorial (master)
$ git commit -m "dataset updates"
[master 9608356] dataset updates
 1 file changed, 1 insertion(+), 1 deletion(-)

DELL@DESKTOP-UVAAH2F MINGW64 ~/Documents/DVC tutorial (master)
$ dvc push
1 file pushed
```



Returning to a different dataset version

With dvc it is easy to return to the other dataset versions. With **git log –oneline**, we can see the commits that we did for our dataset.

```
DELL@DESKTOP-UVAAH2F MINGW64 ~/Documents/DVC tutorial (master)
$ git log --oneline
9608356 (HEAD -> master) dataset updates
f58b82e Configured remote storage.
feb89b7 data added
```

If we look at the git log of our project so far, we see that we have commited two .dvc file versions to git.
Therefore, we must go back to our previous version of the .dvc file, as that is the one git is tracking.

First, simply do a Git checkout to an older commit, like:

```
DELL@DESKTOP-UVAAH2F MINGW64 ~/Documents/DVC tutorial (master)
$ git checkout f58b82e data_collection/source1/iris.csv.dvc
Updated 1 path from 94037ba

DELL@DESKTOP-UVAAH2F MINGW64 ~/Documents/DVC tutorial (master)
$ dvc checkout
M       data_collection\source1\iris.csv
```

Additionally, if you want to keep these dataset changes, simply commit it to git again:

```
DELL@DESKTOP-UVAAH2F MINGW64 ~/Documents/DVC tutorial (master)
$ git commit data_collection/source1/iris.csv.dvc -m "reverted data changes."
[master 9579abc] reverted data changes.
 1 file changed, 1 insertion(+), 1 deletion(-)
DELL@DESKTOP-UVAAH2F MINGW64 ~/Documents/DVC tutorial (master)
$ git log --oneline
9579abc (HEAD -> master) reverted data changes.
9608356 dataset updates
f58b82e Configured remote storage.
feb89b7 data added
```

1.4 Conclusion

DVC, or Data Version Control, is a powerful tool designed to address the challenges associated with managing and versioning large and complex datasets in the field of data science and machine learning. It allows for efficient tracking, versioning, and collaboration on datasets, akin to how version control systems like Git manage code.

In conclusion, DVC offers several key benefits:

1. **Versioning and Reproducibility**: DVC enables precise versioning of datasets, ensuring that data changes can be tracked, rolled back, and reproduced accurately. This is crucial for maintaining data consistency and facilitating reproducible research.

2. **Efficient Storage and Scalability**: DVC employs a lightweight approach by storing metadata and references to data rather than duplicating entire datasets. This minimizes storage costs and allows for scalability as datasets grow.

3. **Collaboration and Teamwork**: DVC supports collaboration by providing a structured and organized framework for multiple team members to work on the same datasets simultaneously. It facilitates seamless integration with version control systems like Git.

4. **Facilitates Experiment Tracking**: DVC helps in associating specific datasets with machine learning models and experiments, enabling a clear understanding of which data was used to produce specific results. This promotes transparency and ease of experimentation.

5. **Integration with ML Tools**: DVC seamlessly integrates with popular machine learning tools and frameworks, allowing for a smooth workflow integration and enhancing the reproducibility of machine learning experiments.

In summary, DVC is a valuable tool in the data science toolkit, enabling effective management, versioning, and collaboration on datasets, which is essential for maintaining data integrity and achieving reproducibility in data-driven projects.