1. Create a new environment.

Conda create -n wineq python=3.7 -y

1. Activate the environment

Conda activate wineq

1. Create requirements.txt for the packages needed to be installed

Text

Description automatically generated

1. Create README.md
2. Download the data from /data\_given folder
3. Create template.py for creating more files for the project

Text

Description automatically generated

1. Save the dataset in the /data\_given folder

Please find more details about the dataset:

<https://www.kaggle.com/datasets/uciml/red-wine-quality-cortez-et-al-2009>

1. Initialized git repository

git init

1. Initialize dvc

git dvc

after that, new folders / files are created:

* .dvc
* .dvcignore

1. Add data to dvc tracking

dvc add data\_given/winequality.csv

1. Git commit and push

git add . && git commit -m “first commit”

git remote add origin https://github.com/dandi0220/-simple-dvc-demo.git

git branch -M main

git push origin main

After the git commit, in the folder data/given, .gitignore is created automatically with the content “/winequality.csv” meaning that this file will not be uploaded to Git repository, we will keep it for dvc tracking locally.

1. Write params.yaml and dvc.yaml
2. Write get\_data.py

This python source code is for reading the parameters, process the data, and return in the form of dataframe.

In this python file, I have used print(df.head()) to see the output of the returned dataframe:

A screenshot of a computer

Description automatically generated with medium confidence

The data looks fine.

1. Write load\_data.py

This python code is for reading the data from data source and save it in the data/raw directory for further process.

1. Run command dvc repro

The current dvc.yaml file is:

Text

Description automatically generated

After running dvc repro, I see that a new file called dvc.lock is created. It keeps track of all the files in the ‘deps’ and ‘outs’ sections.

Dvc.lock file:

Text

Description automatically generated

And also the output file is saved in the data/raw directory successfully.

Graphical user interface, application

Description automatically generated

1. Write split.py code

The python file’s purpose is for splitting the raw data and saving it in data/processed folder

1. Running dvc repro again for the stage split the data

Dvc.yaml is updated the stage for splitting data:

Text

Description automatically generated

After that, the train and test data is created in the data/processed folder.

Text

Description automatically generated

1. Write train\_and\_evaluate.py

This python file’s purpose is for load the train and test data, train the model and save the model and its results.

The model will be saved in saved\_models folder, the results will be saved as json files in the folder reports.

1. Run dvc repro again with the updated dvc.yaml for the stage train\_and\_evaluate.

Updated dvc yaml part:

Text

Description automatically generated

After running the command dvc repro, the model and the reports are saved correctly.

Graphical user interface, text, application

Description automatically generated

1. Check and compare the metrics

Show metrics details with the command dvc metrics show

A screen shot of a computer

Description automatically generated with low confidence

Now, change the alpha and l1\_ratio parameter in the params.yaml and run dvc repro and followed by dvc params diff to see the difference in the model metrics.

A picture containing text, scoreboard, meter, close

Description automatically generated

1. Use tox and pytest to create virtual environment to standardize the testing of the project

* Create tox.ini file

A screenshot of a computer

Description automatically generated with medium confidence

* Create the folder tests and the following files to use later:

Graphical user interface, text

Description automatically generated

* In the test\_config.py file, right some code to test the pytest function

Text

Description automatically generated

* Command tox to test the test in the test\_config.py

.tox folder will be created afterwards.

* Create setup.py:

Text

Description automatically generated

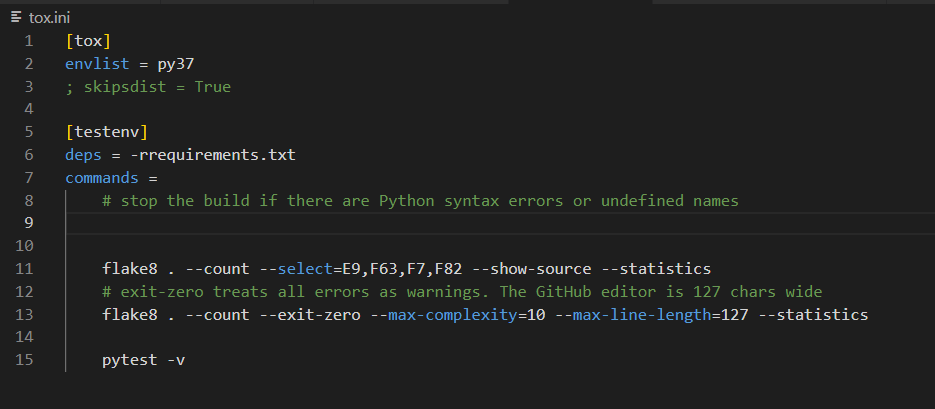
* Run command pip install -e . for local package installation
* (optional) Run command python setup.py sdist dbist\_wheel to build your own package.
* Create jupyter notebook in the folder notebooks/ to find out the range of each feature and save it in a json file:

Text

Description automatically generated

* Use flake8 to check python syntax errors and github line length not over 127

Update flake8 part in tox.ini file:



1. Create web application for the model prediction

* The following files are created:

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

* In addition, app.py is created
* The website looks like:

Graphical user interface, application

Description automatically generated

* Restricting the range of input data based on the existing data’s min and max value. If the input data is out of range, an error will be return.

Graphical user interface, application

Description automatically generated

1. Github Actions and Cloud deployment

* Create Procfile
* .Github/workflows/ci-cd.yaml is created for github actions and Heroku deployment
* Heroku app setup

In Heroku, create a new app and choose the deployment method as Github.

Update the HEROKU\_APP\_NAME and HEROKU\_API\_TOKEN in github Actions secrets setting.

1. MLflow

Based on certain parameters, we will experiment with the model result and find the model with the best perform as the production model. The model parameters and metrics results will be logged on mlflow UI.

* Create a new branch for mlflow

Git checkout -b main-mlflow

* Code changing in file dvc.yaml: remove the metrics and outs section, and add log\_production\_model section.

Text

Description automatically generated

* Code changing in the file params.yaml: add the following code

Text

Description automatically generated

* Add mlflow in the requirements.txt
* Add mlflow code in train\_and\_evaluate.py

Text

Description automatically generated

Text

Description automatically generated

Remove these codes:

Text

Description automatically generated

* Create artifacts folder
* Mlflow server command
* mlflow server \
* --backend-store-uri sqlite:///mlflow.db \
* --default-artifact-root ./artifacts
* Run command dvc repro

After this, the experiment is implemented and can be seen on mlflow UI.

Graphical user interface, text, application

Description automatically generated

* Changes the parameters alpha and l1\_ratio in params.yaml to run the experiment again
* Write code for src/log\_production\_model.py for production of the model. The model which has the lowest mae will be taken as the production model.
* Run the command dvc repro
* The correct model is changed to Production stage and rest is changed to Staging stage.

Graphical user interface

Description automatically generated

* Change the branch from amin to main-mlflow in the github workflows ci-cd.yaml so that the website is deployed on Heroku following the codes with the correct branch.
* git add . && git commit -m “updated codes” && git push origin main-mlflow

After that, git actions have been implemented.

Text

Description automatically generatedText

Description automatically generated