* DVC (Data Version Control)

- basics
- where applicable
- projects → basics
  - Sklear ML
  - Tensorflow (Deeplearing)
  - Pytorch

- This is helpful for Experiment tracking
- Data Versioning
- model Versioning artifacts.

- Automate ML pipelines with DVC
  - Stage 01 load & save.py
  - stage 02 split data.py.
  - Stage 03 train.py
  - stage 04 evaluate.py

- large data files, ML models, config files, scripts & notebooks → DVC → Version Controlled Copies, meta data & info about Changes, integrates with Git.
  - It build pipelines.

* DVC (Data Version Control)

- basics
- where applicable
- projects → basics
  - Sklear ML
  - Tensorflow (Deeplearing)
  - Pytorch

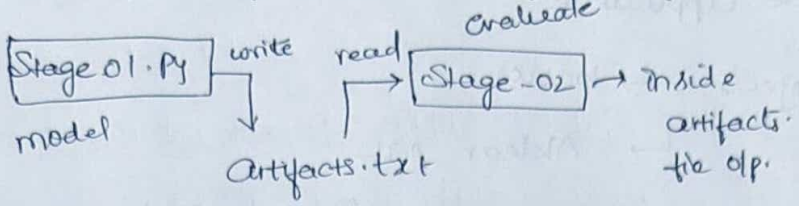
- This is helpful for Experiment tracking
- Data Versioning
- model Versioning artifacts.

- Automate ML pipelines with DVC
  - Stage 01 load & save.py
  - stage 02 split data.py.
  - Stage 03 train.py
  - stage 04 evaluate.py

- large data files, ML models, config files, scripts & notebooks → DVC → Version Controlled Copies, meta data & info about Changes, integrates with Git.
  - it build pipelines.

↳ ~~Install DVC~~

↳ Created Stage-01.py file & Stage 02 .py file.

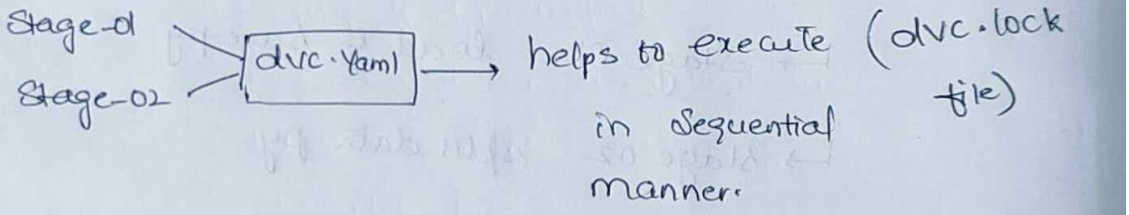


↳ in most of ml pipeline we write something & tries to create a file. & then read it.

↳ DAG Direct Acyclic graph should follow here we write Create & then ~~write~~ read. this flow should follow

↳ create .dvc , dvc ignore files (we are initializing them) dvc init, git init

↳ Create dvc.yaml file



↳ in dvc.yaml we give

Stages

↳ Stage 01

↳ cmd (running Stage 01 file)

↳ dependencies (files needed to run cmd

↳ o/p of file Stage 01

↳ Stage 02

↳ cmd

↳ dependency

↳ o/p's if any.

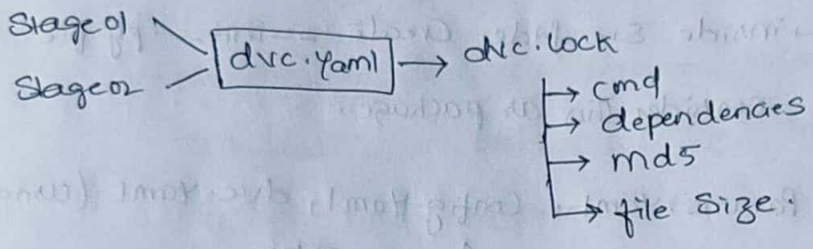
↳ ...

at the end this creates dvc.lock file.

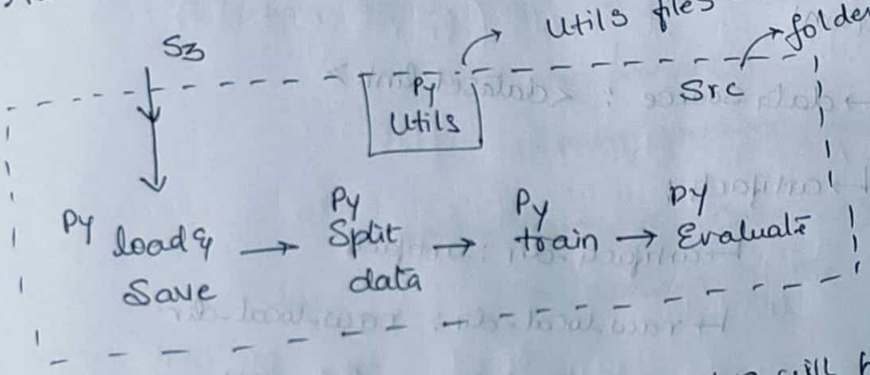
↳ dvc.lock file

↳ it consists of pipeline of stages.

↳ in this along with given variables in dvc.yaml file it has or generate md5 (which is a hash value/key) & also size of file it has.



↳ Automate ML pipelines with DVC



↳ Outside of src folder we will have (root folder

↳ Artifacts folder

↳ it consists raw data, processed data, model, stores metrics,

↳ params.yaml

↳ responsible for having all parameters responsible for training.

↳ dvc.yaml

↳ it consists of all stages steps that occurred, we mention those here

→ stage 01. load & save.py

→ stage 02 split data.py

→ stage 03 train.py

→ ~~evaluate~~ stage 04 evaluate.py

↳ Config.yaml

↳ After that we Create Requirements.txt file.

↳ working on DVC pipeline

↳ Create Src folder

↳ in thus Create Utils folders.

↳ in Utils Create __init__.py file

↳ inside Src folder Create __init__.py file So that it consider this as package.

↳ Create Params.yaml, Config.yaml, dvc.yaml. (using touch cmd)

inside config folder

↳ Config.yaml

↳ data source : <dataset link>

↳ artifacts :

↳ artifacts_dir : artifacts

↳ raw_local_dir : raw_local_dir

↳ raw_local_file : data.csv

↳ ~~Split_data~~ :

↳ In Src folder

↳ Create load_save.py file

↳ Inside Utils folder Create utils.py file.

↳ import yaml

↳ import os

↳ i/p path to yaml o/p : dictionary

↳ This takes config.yaml file as i/p & reads it & gives o/p as dictionary form of content in config.yaml.

↳ and then we need to create a setup.py file.

↳ i/p: readme.md file (it reads the file)
& setup(.....

↳ o/p: dependency_links.txt
PKG-INFO
requires.txt
SOURCES.txt
top_level.txt ...

↳ This setup.py file & src becomes local packages to
To install local packages, -e. in requirements.txt.
& then run requirements.txt.

↳ Now we need to work on load_save.py file.

↳ use read_~~config~~ yaml fn from all_utils folder from src.

↳ i/p: argparser()
config.yaml (or) params.yaml file

↳ o/p: name of given config file
(or) content of the config file.

↳ i/p: path to data source in config.yaml

o/p:- it reads the data in path & gives it.

↳ Now as we got the data from remote path, we need to save the data that we got from remote path into artifacts directory.

↳ Create fn inside this load_save.py file itself.

↳ Before create a directory, write a fn in all_utils.py.

↳ i/p: empty list

↳ o/p: directories.

↳ Now Considering config.yaml file
we need to save dataset in local dir
for that creating path to directory.

↳ i/p : artifacts_dir
raw_local_dir
raw_local_file
raw_local_dir_path
raw_local_file_path.

o/p: directory gets created
& data.csv file gets created.

Till now we have finished stage 01 (loading data)

↳ Now adding stage to dvc.yaml file

i/p: Stages:
load_data:
cmd: run load_save.py file with config.yaml as i/p

depencies: all_utils.py
config.yaml
load_save.py

o/p: data.csv.

↳ commit changes if needed.

↳ Now let us go further & add next stage (02) which is splitting of data.

↳ Create file in src folder src/Stage-02-Split_data.py

↳ i/p: Config path, params path
Split ratio, random_state

o/p:- ① df from data.csv from path

↳ fn:- Save-local-df → i/p : data, data_path
→ o/p :- Save file @ datapath

↳ train_data_path → i/p : artifacts_dir, Split_data dir, filename

↳ Create_directory → i/p : artifacts_dir, split_data_dir.

↳ once we do all this o/p:- train.csv
test.csv

↳ Now we can add this to dvc.yaml stage.

↳ mention deps, params, outs & all in dvc.yaml stage.

↳ Save_local_df → i/p : data, datapath
o/p:- datapath.

↳ defining directory path for storing split data from config.

↳ Now Create stage_03_train.py using touch cmd in src fold und Utils folder.

↳ in train we need to Specify directories & paths along with hyper parameters

↳ Create a trained model

↳ & Save the model to disk.

↳ So for Stage 03_train.py,

i/p:- fn to train model

paths & directories

CSV file

model params

o/p:- trained model in model_dir

(_ _ _ * . model file)

↳ Now Create another file ~~era~~ Stage-03-evaluate.py in src folder.

↳ i/p :- trained_model
text-data
testy-
predicted values
actual-values
reports-dir
scores
scores-dir
scores-filename

o/p :- predicted values
rmse, mae, r2 (metrics)
scores-dir
scores-filename
scores dir path
scores-filepath.

↳ if we want to add a stage where it gives best scores & best model among multiple models.