Project Specification Document

This document is for the purpose of step by step implementation of the project.

Environment setup and basic structure files of the project created

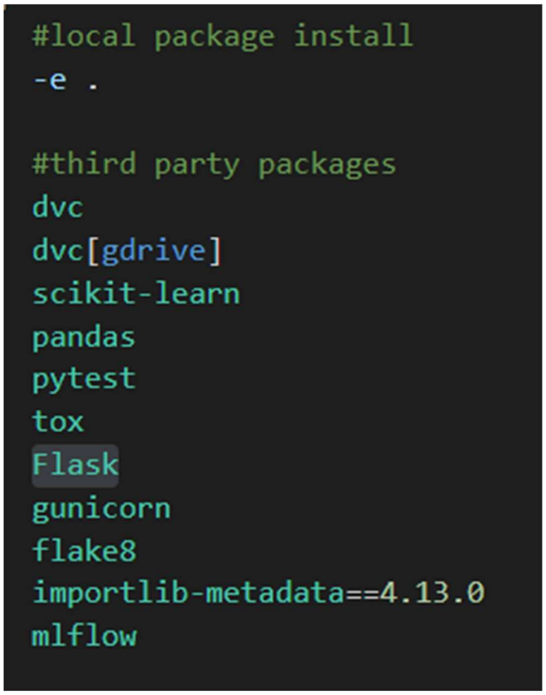
- Create a new environment.

Conda create -n wineq python=3.7 -y

- Activate the environment

Conda activate wineq

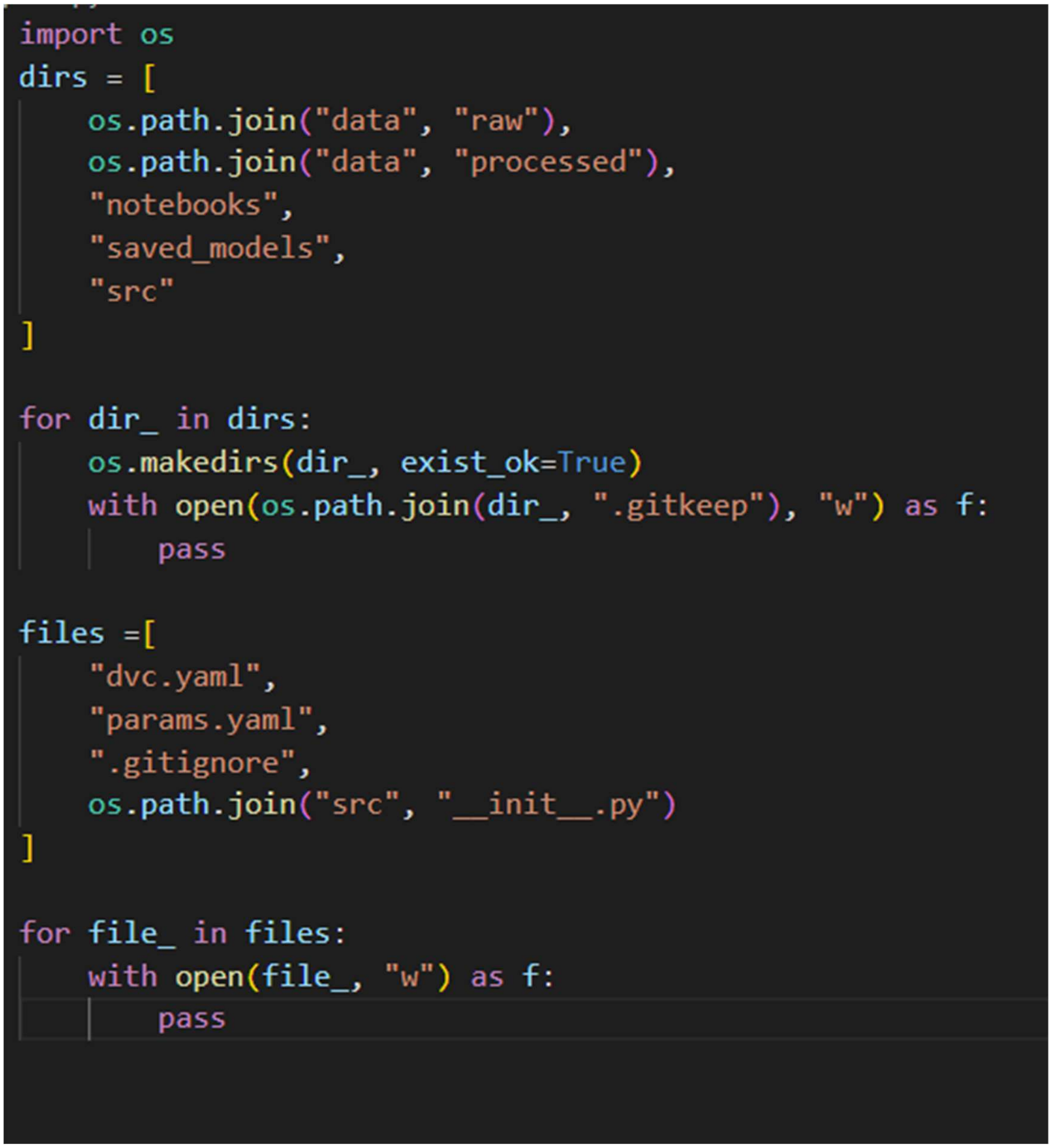
- Create requirements.txt for the packages needed to be installed



```
#local package install
-e .

#third party packages
dvc
dvc[gdrive]
scikit-learn
pandas
pytest
tox
Flask
gunicorn
flake8
importlib-metadata==4.13.0
mlflow
```

- Create README.md
- Download the data from /data_given folder

- Create template.py for creating more files for the project



```python
import os
dirs = [
    os.path.join("data", "raw"),
    os.path.join("data", "processed"),
    "notebooks",
    "saved_models",
    "src"
]

for dir_ in dirs:
    os.makedirs(dir_, exist_ok=True)
    with open(os.path.join(dir_, ".gitkeep"), "w") as f:
        pass

files =[
    "dvc.yaml",
    "params.yaml",
    ".gitignore",
    os.path.join("src", "__init__.py")
]

for file_ in files:
    with open(file_, "w") as f:
        pass
```

Input dataset, dvc & git initilization

- 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

- Initialized git repository

```
git init
```

- Initialize dvc

```
git dvc
```

after that, new folders / files are created:

- .dvc
- .dvcignore

- Add data to dvc tracking

```
dvc add data_given/winequality.csv
```

- 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
```

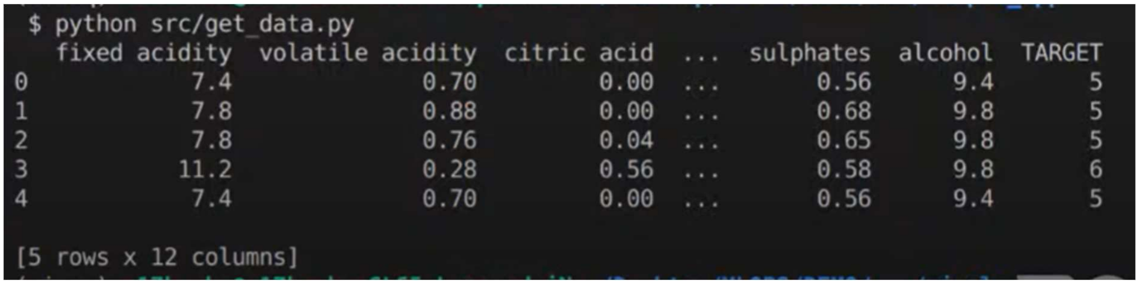
After the git commit, in the folder data/given, .gitignore is created automatically with the content “/winequality.csv” meaning that this file will not be uploaded to Git repository, we will keep it for dvc tracking locally.

Write source code files

- Write params.yaml and dvc.yaml
- 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:



```
 $ python src/get_data.py
   fixed acidity  volatile acidity  citric acid  ...  sulphates  alcohol  TARGET
0            7.4              0.70         0.00  ...       0.56      9.4       5
1            7.8              0.88         0.00  ...       0.68      9.8       5
2            7.8              0.76         0.04  ...       0.65      9.8       5
3           11.2              0.28         0.56  ...       0.58      9.8       6
4            7.4              0.70         0.00  ...       0.56      9.4       5

[5 rows x 12 columns]
```

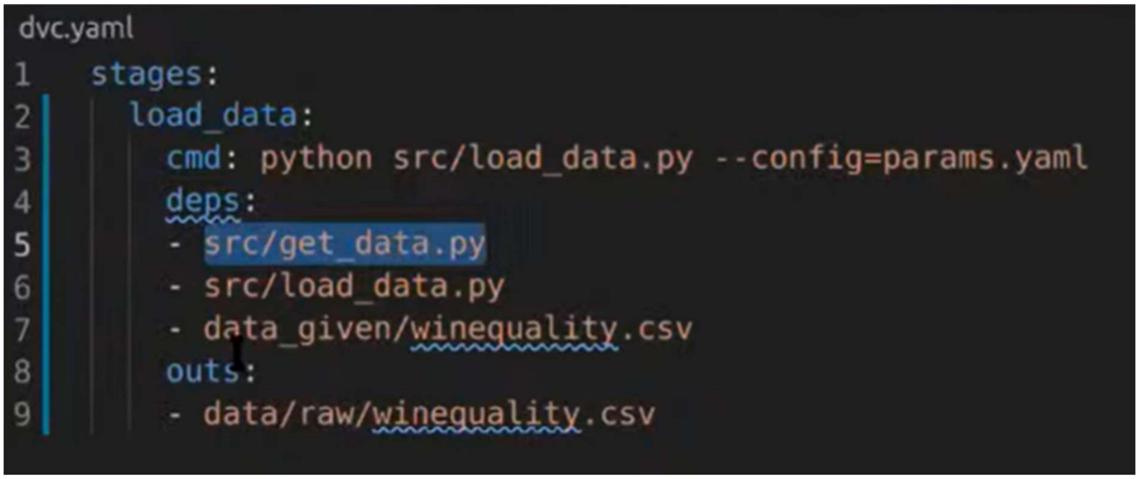
The data looks fine.

- 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.

- Run command dvc repro

The current dvc.yaml file is:

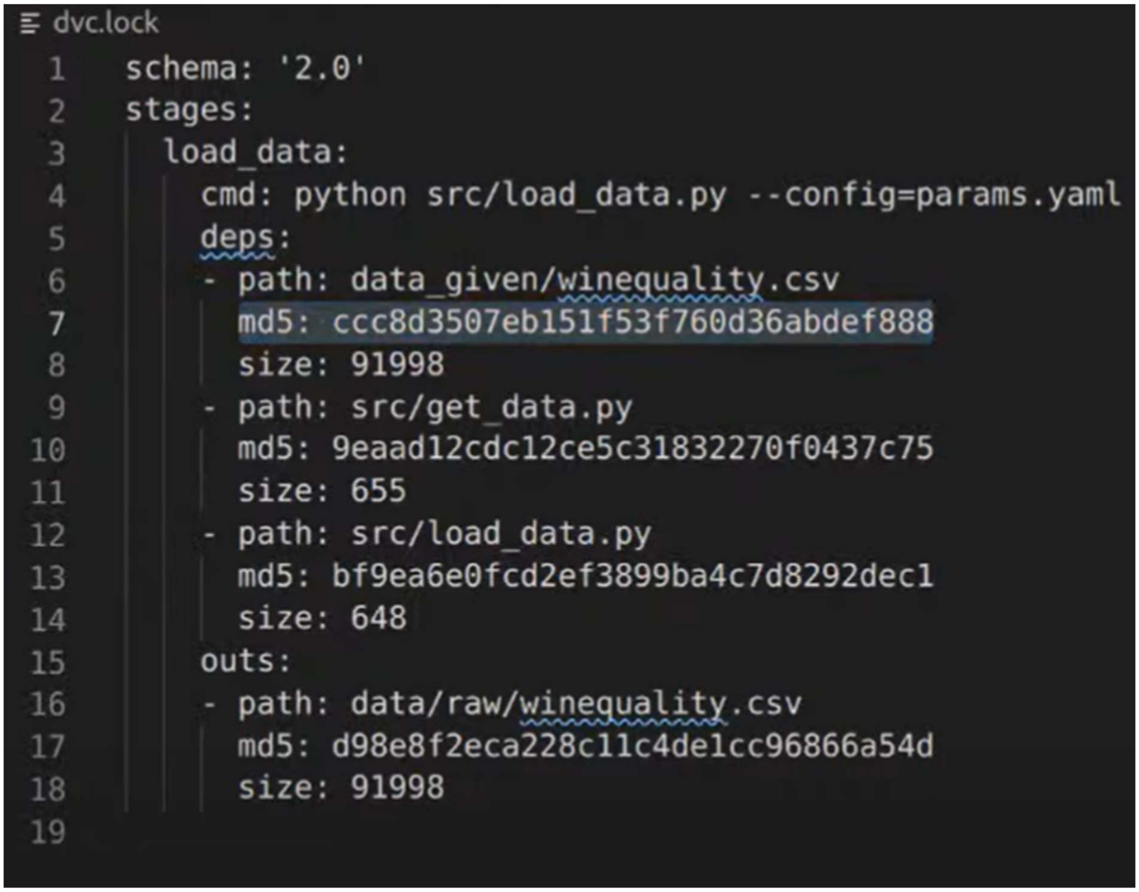


dvc.yaml

```
stages:
  load_data:
    cmd: python src/load_data.py --config=params.yaml
    deps:
    - src/get_data.py
    - src/load_data.py
    - data_given/winequality.csv
    outs:
    - data/raw/winequality.csv
```

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.

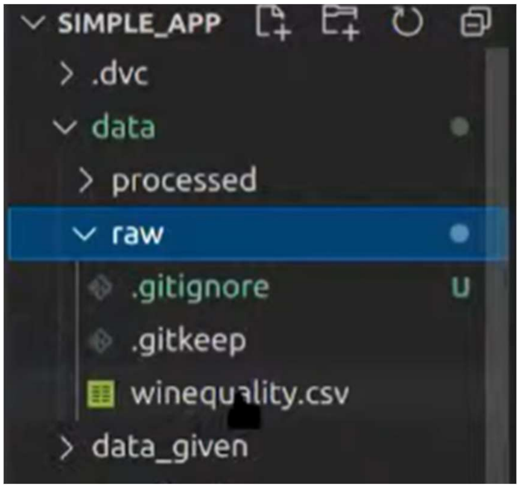
Dvc.lock file:



dvc.lock

```
schema: '2.0'
stages:
  load_data:
    cmd: python src/load_data.py --config=params.yaml
    deps:
    - path: data_given/winequality.csv
      md5: ccc8d3507eb151f53f760d36abdef888
      size: 91998
    - path: src/get_data.py
      md5: 9eaad12cdc12ce5c31832270f0437c75
      size: 655
    - path: src/load_data.py
      md5: bf9ea6e0fcd2ef3899ba4c7d8292dec1
      size: 648
    outs:
    - path: data/raw/winequality.csv
      md5: d98e8f2eca228c11c4de1cc96866a54d
      size: 91998
```

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

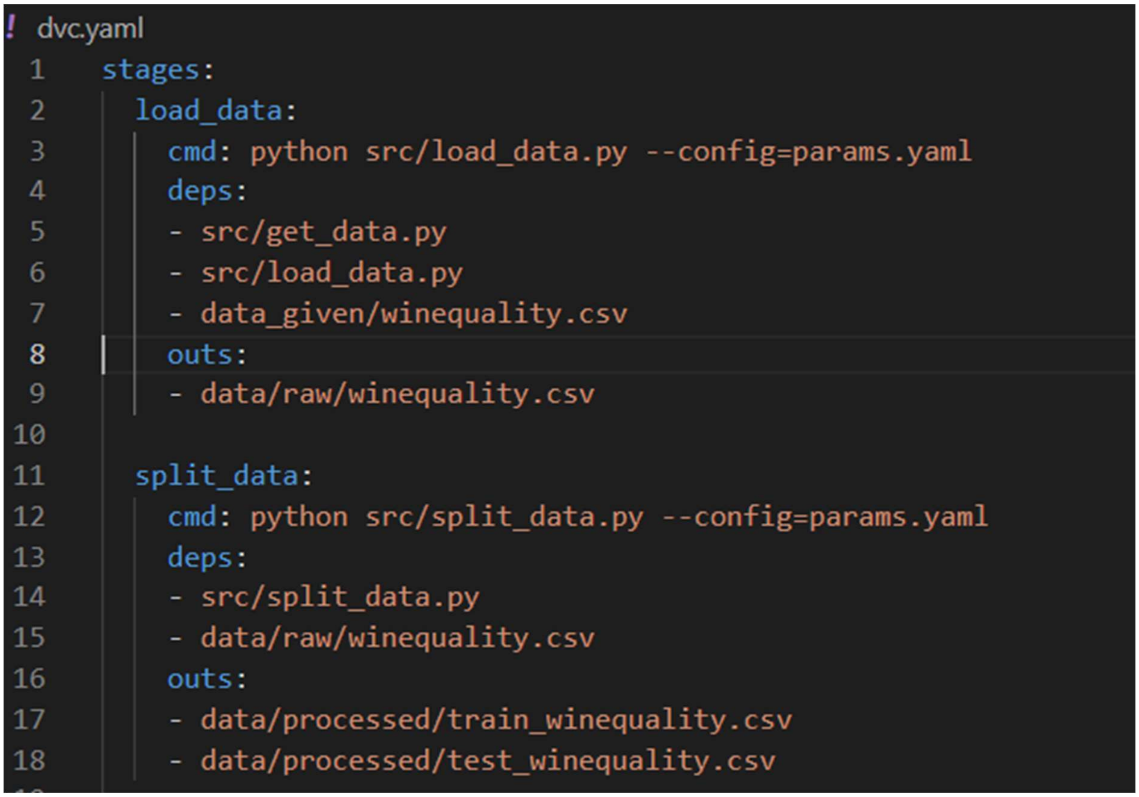


- Write split.py code

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

- Running dvc repro again for the stage split the data

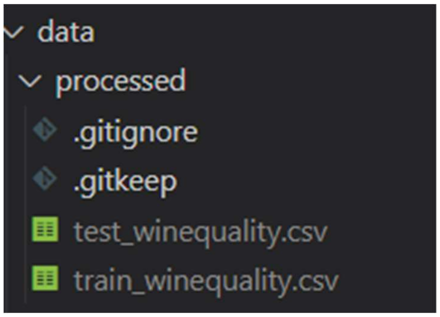
Dvc.yaml is updated the stage for splitting data:



dvc.yaml

```yaml
stages:
  load_data:
    cmd: python src/load_data.py --config=params.yaml
    deps:
    - src/get_data.py
    - src/load_data.py
    - data_given/winequality.csv
    outs:
    - data/raw/winequality.csv

  split_data:
    cmd: python src/split_data.py --config=params.yaml
    deps:
    - src/split_data.py
    - data/raw/winequality.csv
    outs:
    - data/processed/train_winequality.csv
    - data/processed/test_winequality.csv
```

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



```
data
  processed
    .gitignore
    .gitkeep
    test_winequality.csv
    train_winequality.csv
```

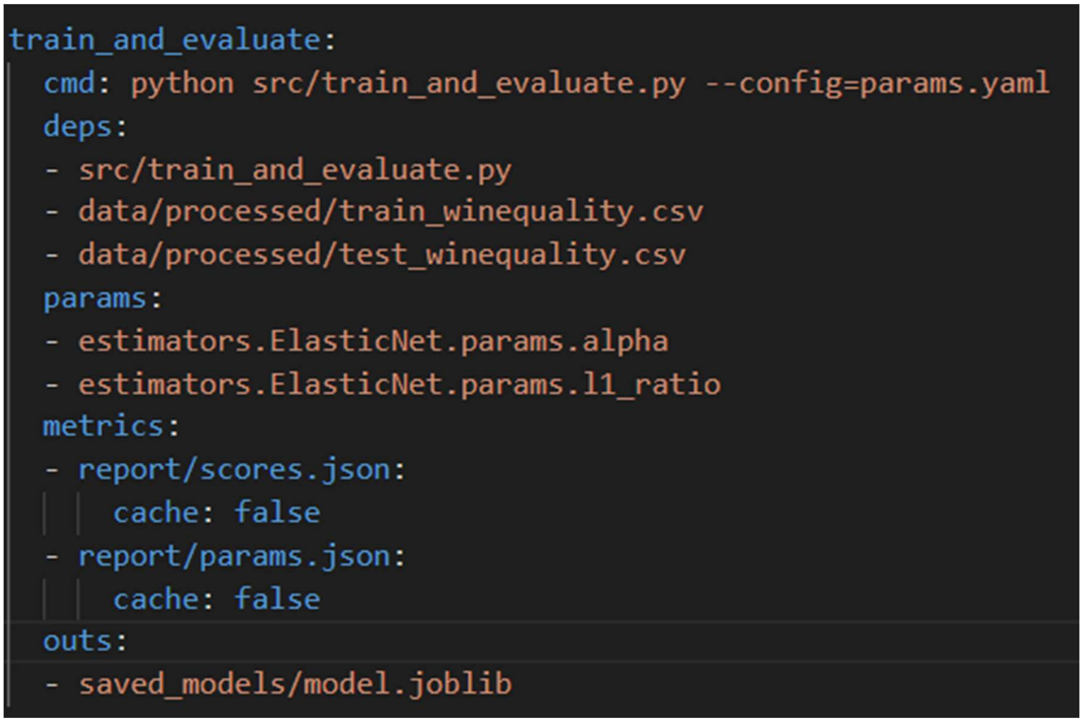
- Write train_and_evaluate.py

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.

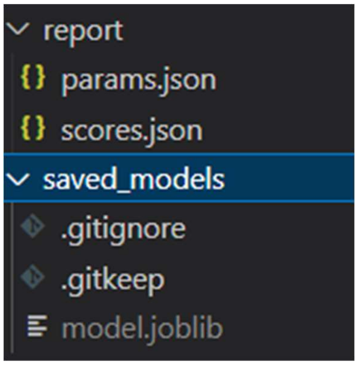
- Run dvc repro again with the updated dvc.yaml for the stage train_and_evaluate.

Updated dvc yaml part:



```
train_and_evaluate:
  cmd: python src/train_and_evaluate.py --config=params.yaml
  deps:
  - src/train_and_evaluate.py
  - data/processed/train_winequality.csv
  - data/processed/test_winequality.csv
  params:
  - estimators.ElasticNet.params.alpha
  - estimators.ElasticNet.params.l1_ratio
  metrics:
  - report/scores.json:
      cache: false
  - report/params.json:
      cache: false
  outs:
  - saved_models/model.joblib
```

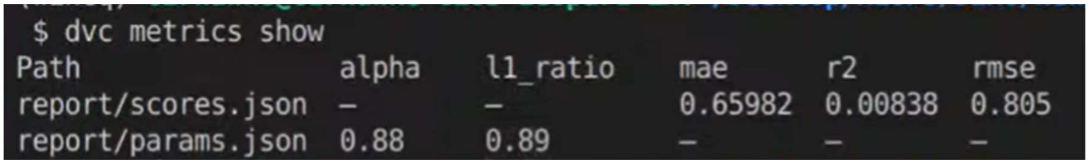
After running the command dvc repro, the model and the reports are saved correctly.



```
report
  params.json
  scores.json
saved_models
  .gitignore
  .gitkeep
  model.joblib
```

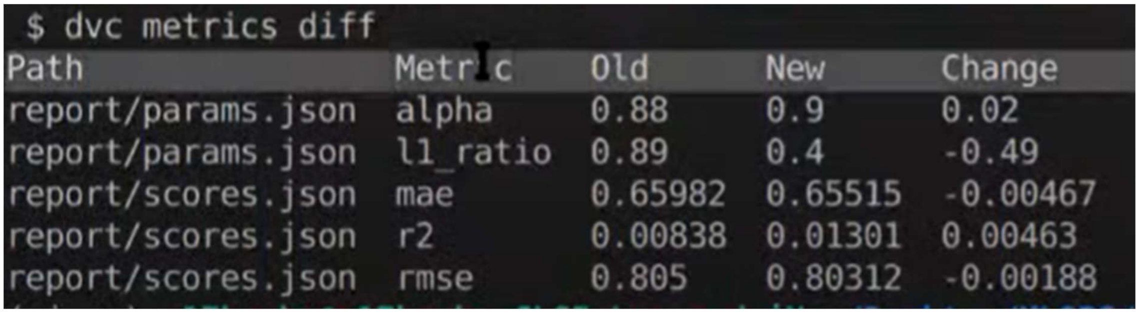
- Check and compare the metrics

Show metrics details with the command dvc metrics show



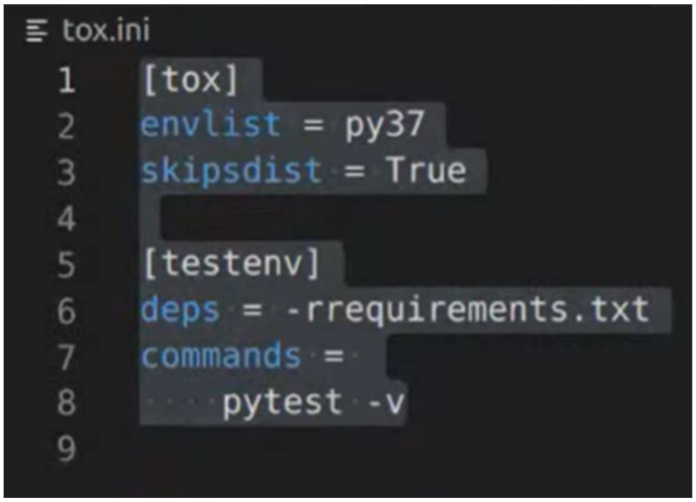
```
$ dvc metrics show
Path                alpha    l1_ratio    mae      r2       rmse
report/scores.json  –        –           0.65982  0.00838  0.805
report/params.json  0.88     0.89        –        –        –
```

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.



```
$ dvc metrics diff
Path                Metric    Old      New      Change
report/params.json  alpha     0.88     0.9      0.02
report/params.json  l1_ratio  0.89     0.4      -0.49
report/scores.json  mae       0.65982  0.65515  -0.00467
report/scores.json  r2        0.00838  0.01301  0.00463
report/scores.json  rmse      0.805    0.80312  -0.00188
```

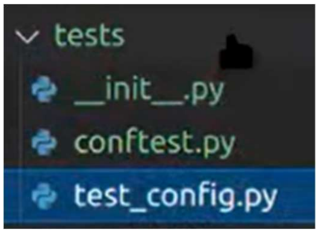
- Use tox and pytest to create virtual environment to standardize the testing of the project
- Create tox.ini file



tox.ini

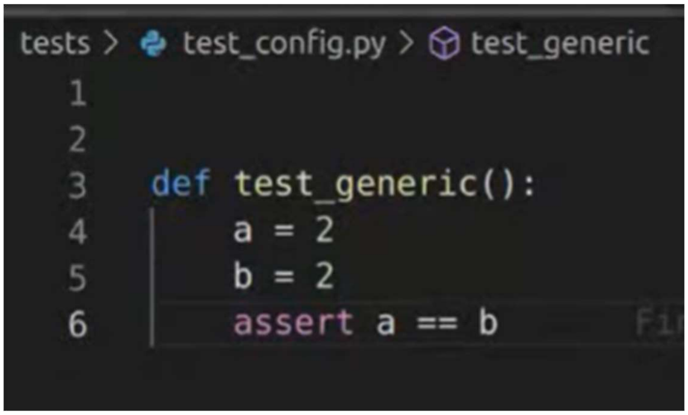
```
[tox]
envlist = py37
skipsdist = True

[testenv]
deps = -rrequirements.txt
commands =
    pytest -v

```

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



```
tests
  __init__.py
  conftest.py
  test_config.py
```

- In the test_config.py file, right some code to test the pytest function



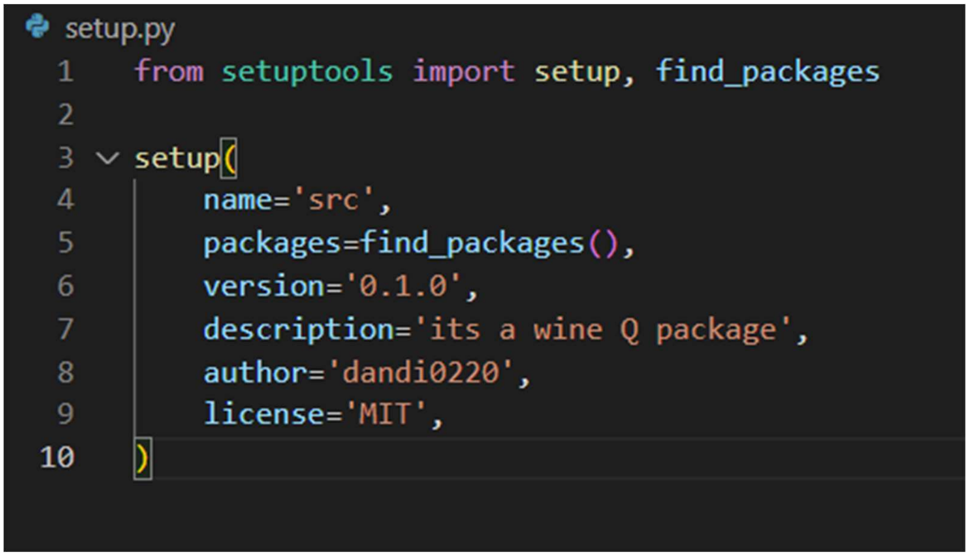
tests > test_config.py > test_generic

```python


def test_generic():
    a = 2
    b = 2
    assert a == b
```

- Command tox to test the test in the test_config.py

.tox folder will be created afterwards.

- Create setup.py:

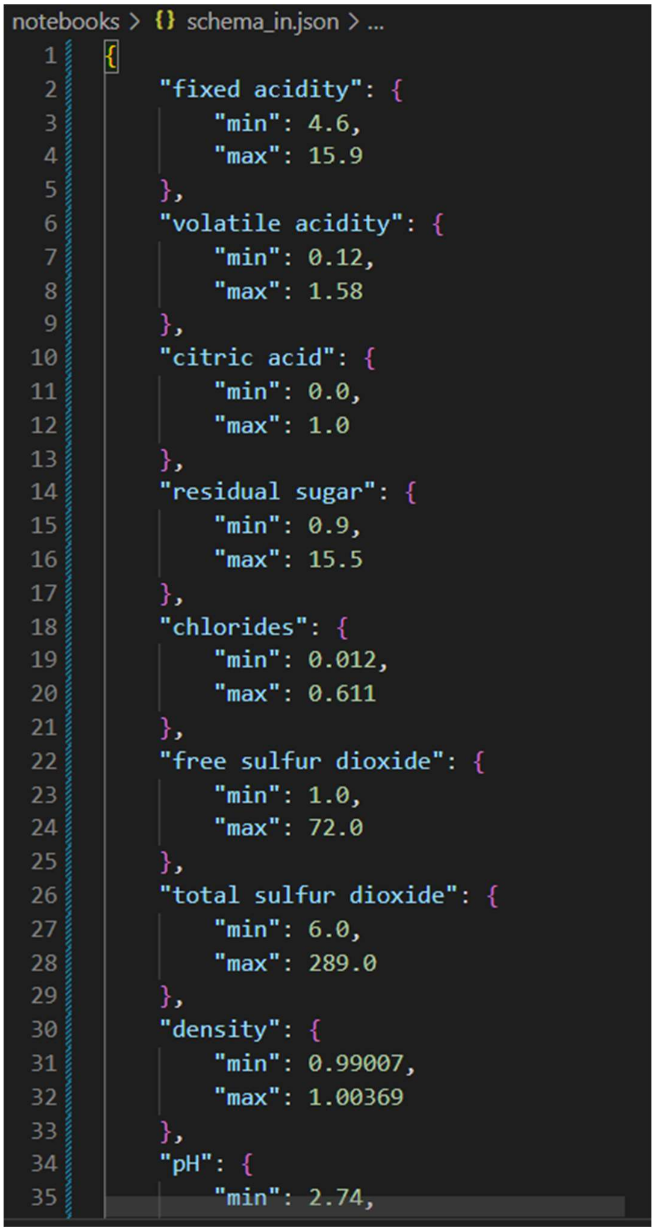


setup.py

```python
from setuptools import setup, find_packages

setup(
    name='src',
    packages=find_packages(),
    version='0.1.0',
    description='its a wine Q package',
    author='dandi0220',
    license='MIT',
)
```

- Run command pip install -e . for local package installation

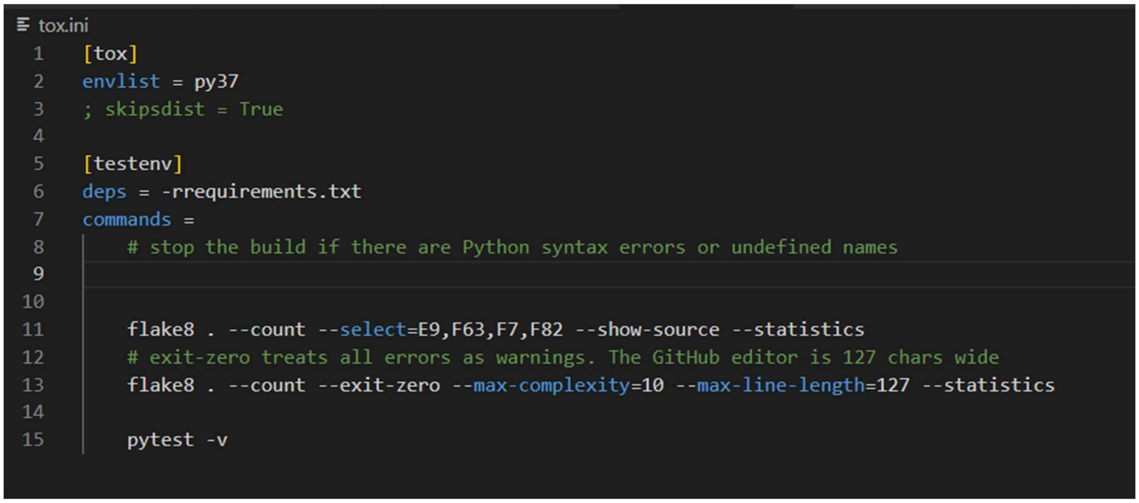
- (optional) Run command python setup.py sdist dbist_wheel to build your own package.
- Create jupyter notebook in the folder notebooks/ to find out the range of each feature and save it in a json file:



```
{
    "fixed acidity": {
        "min": 4.6,
        "max": 15.9
    },
    "volatile acidity": {
        "min": 0.12,
        "max": 1.58
    },
    "citric acid": {
        "min": 0.0,
        "max": 1.0
    },
    "residual sugar": {
        "min": 0.9,
        "max": 15.5
    },
    "chlorides": {
        "min": 0.012,
        "max": 0.611
    },
    "free sulfur dioxide": {
        "min": 1.0,
        "max": 72.0
    },
    "total sulfur dioxide": {
        "min": 6.0,
        "max": 289.0
    },
    "density": {
        "min": 0.99007,
        "max": 1.00369
    },
    "pH": {
        "min": 2.74,
```

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

Update flake8 part in tox.ini file:

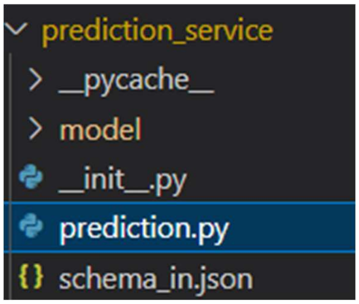


tox.ini

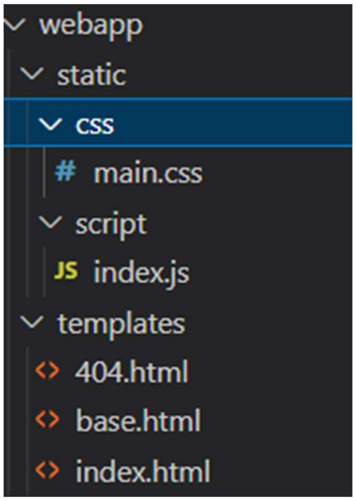
```
[tox]
envlist = py37
; skipsdist = True

[testenv]
deps = -rrequirements.txt
commands =
    # stop the build if there are Python syntax errors or undefined names


    flake8 . --count --select=E9,F63,F7,F82 --show-source --statistics
    # exit-zero treats all errors as warnings. The GitHub editor is 127 chars wide
    flake8 . --count --exit-zero --max-complexity=10 --max-line-length=127 --statistics

    pytest -v
```

Create web application for the model prediction

- The following files are created:

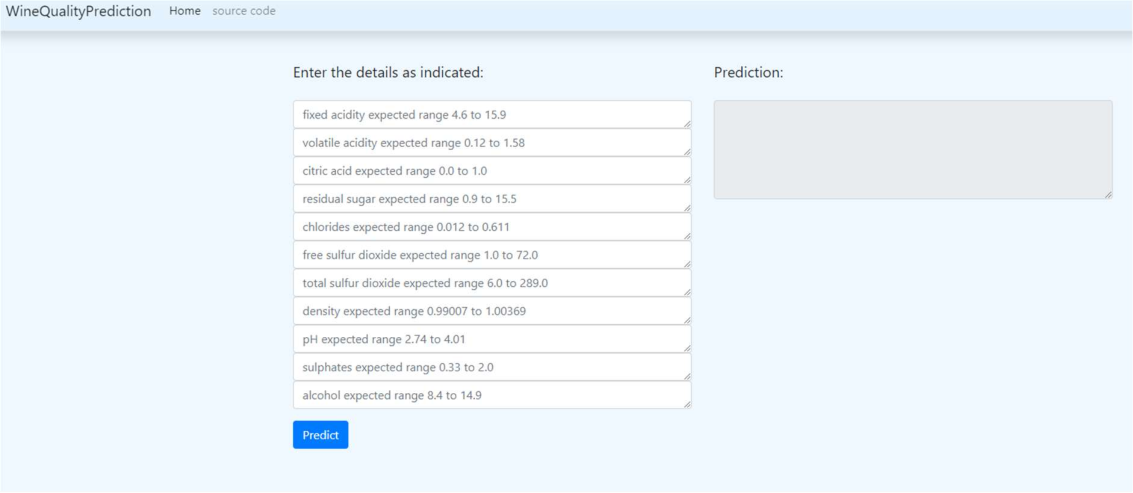


```
prediction_service
  __pycache__
  model
  __init__.py
  prediction.py
  schema_in.json
```



```
webapp
  static
    css
      main.css
    script
      index.js
  templates
    404.html
    base.html
    index.html
```

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



Enter the details as indicated:

Prediction:

fixed acidity expected range 4.6 to 15.9

volatile acidity expected range 0.12 to 1.58

citric acid expected range 0.0 to 1.0

residual sugar expected range 0.9 to 15.5

chlorides expected range 0.012 to 0.611

free sulfur dioxide expected range 1.0 to 72.0

total sulfur dioxide expected range 6.0 to 289.0

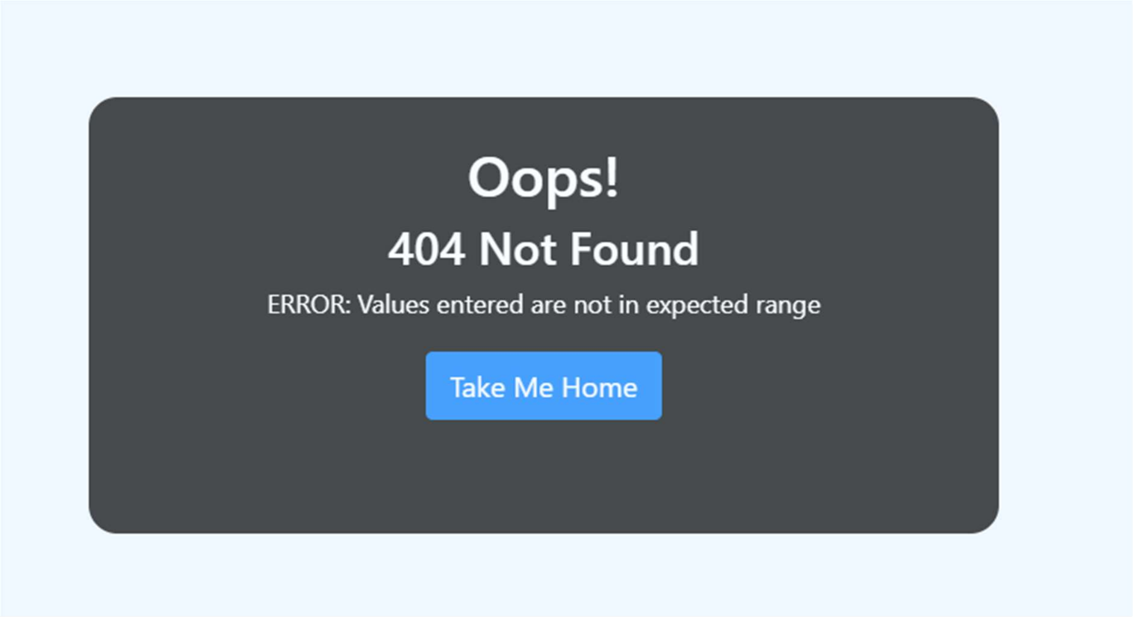
density expected range 0.99007 to 1.00369

pH expected range 2.74 to 4.01

sulphates expected range 0.33 to 2.0

alcohol expected range 8.4 to 14.9

- 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.



Oops!

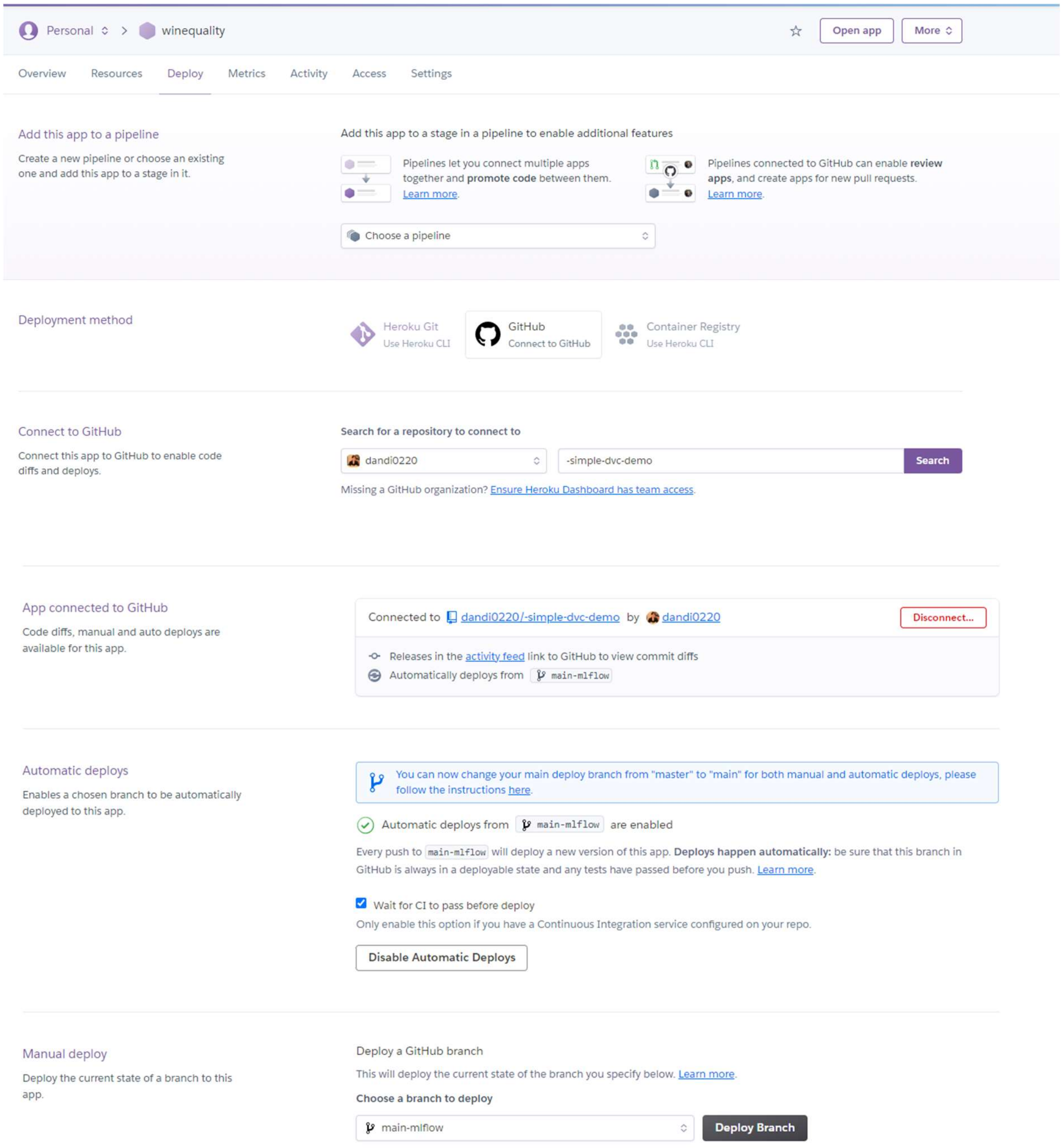
404 Not Found

ERROR: Values entered are not in expected range

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, choose the deployment method as GitHub and fill in the linked GitHub repository name.



Add this app to a pipeline

Create a new pipeline or choose an existing one and add this app to a stage in it.

Add this app to a stage in a pipeline to enable additional features

Pipelines let you connect multiple apps together and **promote code** between them. Learn more.

Pipelines connected to GitHub can enable **review apps**, and create apps for new pull requests. Learn more.

Choose a pipeline ⇕

Deployment method

Connect to GitHub

Connect this app to GitHub to enable code diffs and deploys.

Search for a repository to connect to

dandi0220 ⇕ | -simple-dvc-demo | Search

Missing a GitHub organization? Ensure Heroku Dashboard has team access.

App connected to GitHub

Code diffs, manual and auto deploys are available for this app.

Connected to dandi0220/-simple-dvc-demo by dandi0220 Disconnect...

Releases in the activity feed link to GitHub to view commit diffs

Automatically deploys from `main-mlflow`

Automatic deploys

Enables a chosen branch to be automatically deployed to this app.

You can now change your main deploy branch from "master" to "main" for both manual and automatic deploys, please follow the instructions here.

✓ Automatic deploys from `main-mlflow` are enabled

Every push to `main-mlflow` will deploy a new version of this app. **Deploys happen automatically:** be sure that this branch in GitHub is always in a deployable state and any tests have passed before you push. Learn more.

☑ Wait for CI to pass before deploy

Only enable this option if you have a Continuous Integration service configured on your repo.

Disable Automatic Deploys

Manual deploy

Deploy the current state of a branch to this app.

Deploy a GitHub branch

This will deploy the current state of the branch you specify below. Learn more.

Choose a branch to deploy

main-mlflow ⇕ | Deploy Branch

Update the HEROKU_APP_NAME and HEROKU_API_TOKEN in github Actions secrets setting.

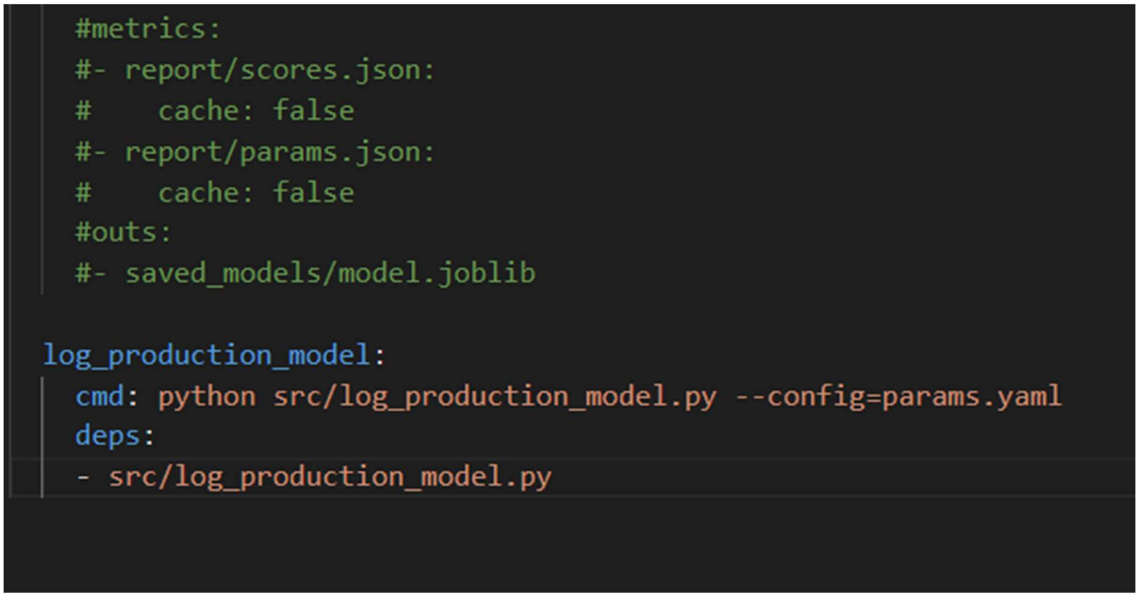
MLflow automation

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

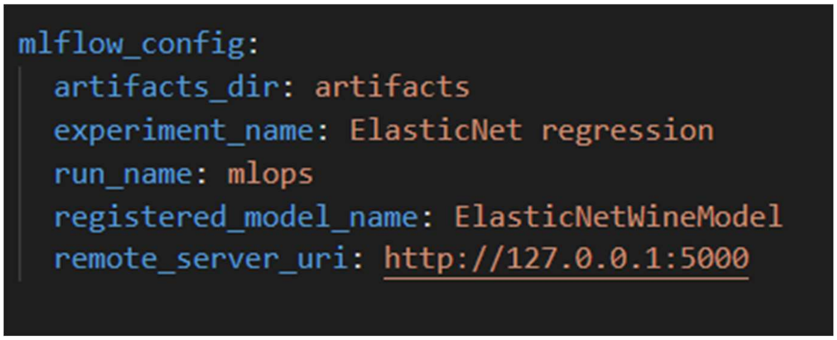
Git checkout -b main-mlflow

- Code changing in file dvc.yaml: remove the metrics and outs section, and add log_production_model section.



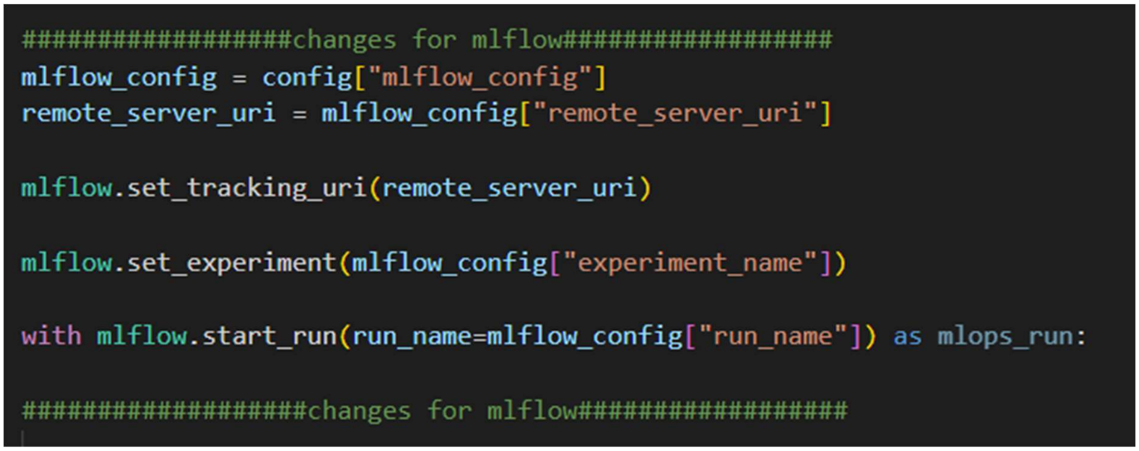
```
    #metrics:
    #- report/scores.json:
    #    cache: false
    #- report/params.json:
    #    cache: false
    #outs:
    #- saved_models/model.joblib

  log_production_model:
    cmd: python src/log_production_model.py --config=params.yaml
    deps:
    - src/log_production_model.py
```

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

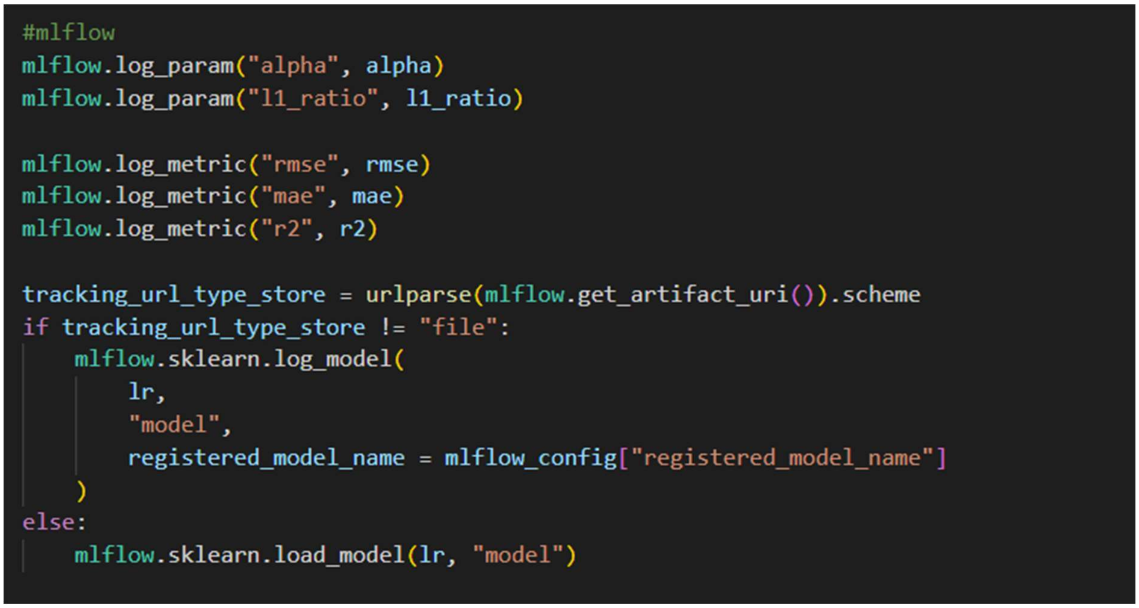


```
mlflow_config:
  artifacts_dir: artifacts
  experiment_name: ElasticNet regression
  run_name: mlops
  registered_model_name: ElasticNetWineModel
  remote_server_uri: http://127.0.0.1:5000
```

- Add mlflow in the requirements.txt
- Add mlflow code in train_and_evaluate.py

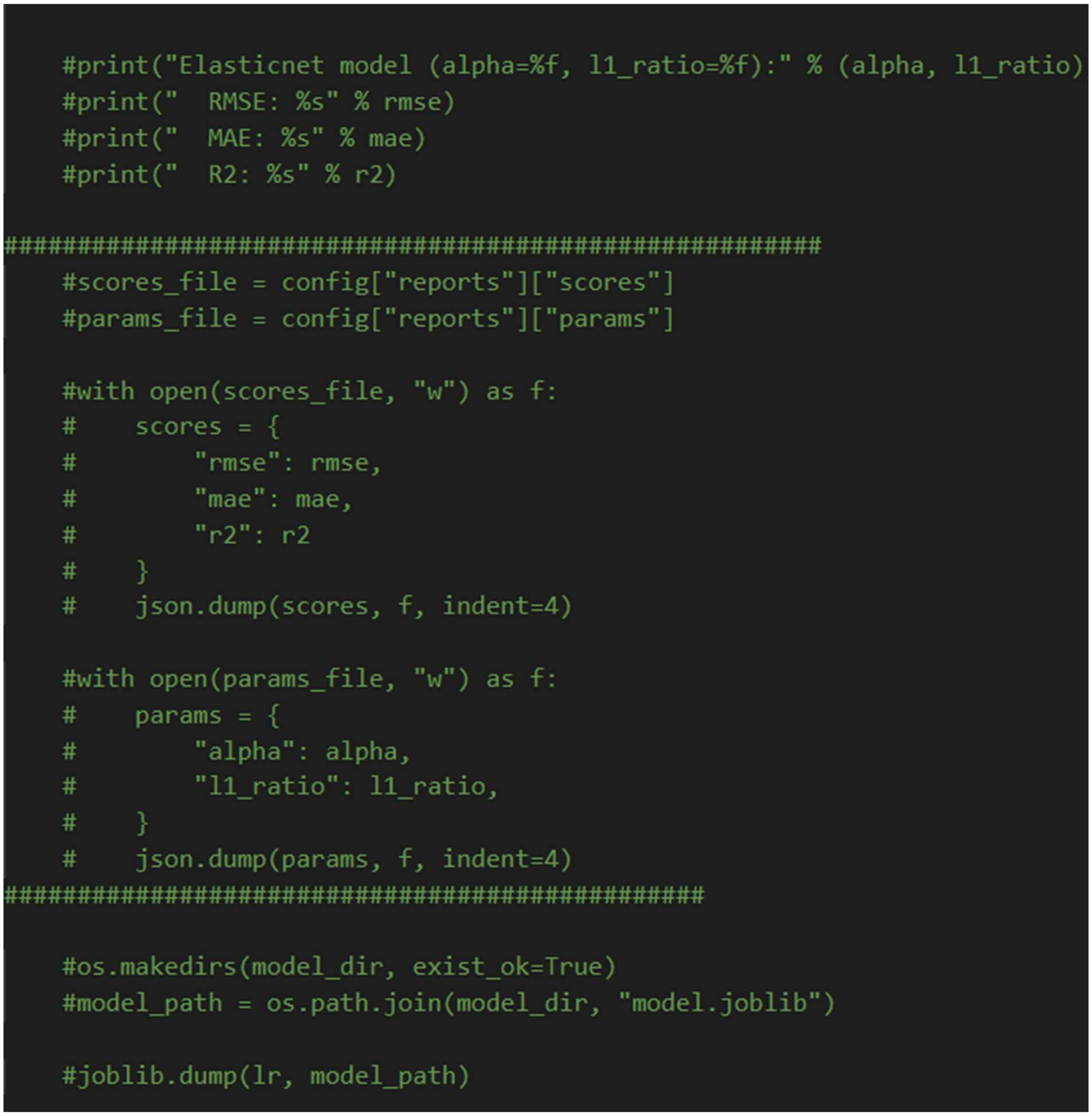


```
##################changes for mlflow##################
mlflow_config = config["mlflow_config"]
remote_server_uri = mlflow_config["remote_server_uri"]

mlflow.set_tracking_uri(remote_server_uri)

mlflow.set_experiment(mlflow_config["experiment_name"])

with mlflow.start_run(run_name=mlflow_config["run_name"]) as mlops_run:

###################changes for mlflow##################
```



```
#mlflow
mlflow.log_param("alpha", alpha)
mlflow.log_param("l1_ratio", l1_ratio)

mlflow.log_metric("rmse", rmse)
mlflow.log_metric("mae", mae)
mlflow.log_metric("r2", r2)

tracking_url_type_store = urlparse(mlflow.get_artifact_uri()).scheme
if tracking_url_type_store != "file":
    mlflow.sklearn.log_model(
        lr,
        "model",
        registered_model_name = mlflow_config["registered_model_name"]
    )
else:
    mlflow.sklearn.load_model(lr, "model")
```

Remove these codes:



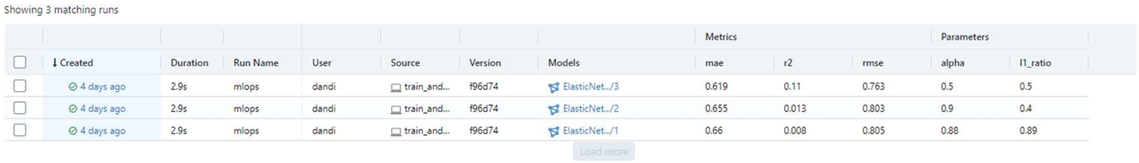
```
    #print("Elasticnet model (alpha=%f, l1_ratio=%f):" % (alpha, l1_ratio)
    #print("  RMSE: %s" % rmse)
    #print("  MAE: %s" % mae)
    #print("  R2: %s" % r2)

#########################################################
    #scores_file = config["reports"]["scores"]
    #params_file = config["reports"]["params"]

    #with open(scores_file, "w") as f:
    #    scores = {
    #        "rmse": rmse,
    #        "mae": mae,
    #        "r2": r2
    #    }
    #    json.dump(scores, f, indent=4)

    #with open(params_file, "w") as f:
    #    params = {
    #        "alpha": alpha,
    #        "l1_ratio": l1_ratio,
    #    }
    #    json.dump(params, f, indent=4)
#################################################

    #os.makedirs(model_dir, exist_ok=True)
    #model_path = os.path.join(model_dir, "model.joblib")

    #joblib.dump(lr, model_path)
```

- Create artifacts folder
- Mlflow server command

```
-   mlflow server \
-       --backend-store-uri sqlite:///mlflow.db \
-       --default-artifact-root ./artifacts
```

- Run command dvc repro

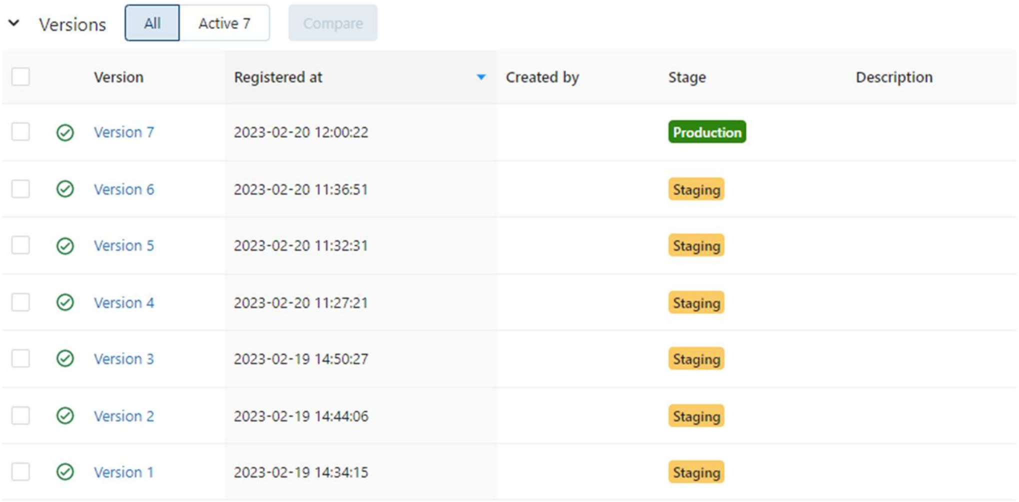
After this, the experiment is implemented and can be seen on mlflow UI.



Showing 3 matching runs

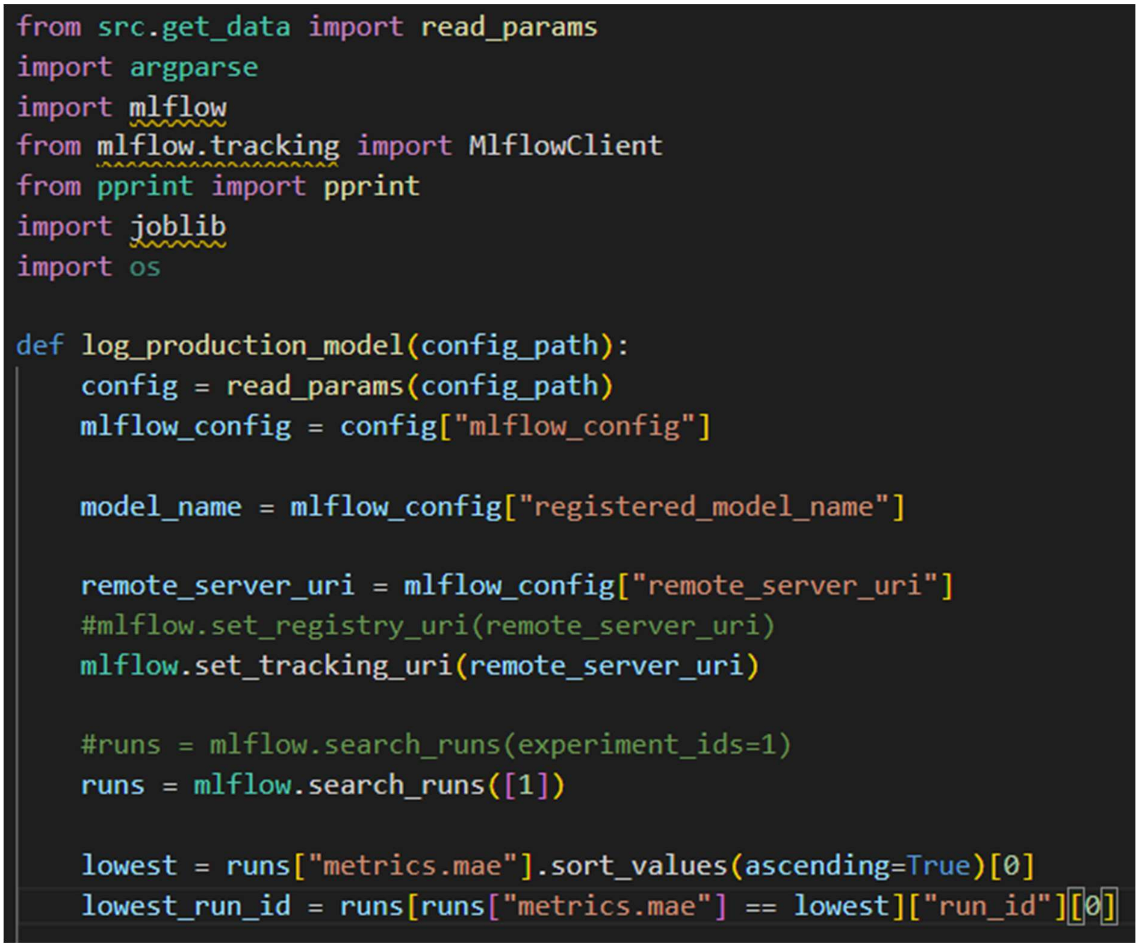
□								Metrics			Parameters	
□	↓ Created	Duration	Run Name	User	Source	Version	Models	mae	r2	rmse	alpha	l1_ratio
□	✓ 4 days ago	2.9s	mlops	dandi	train_and...	f96d74	ElasticNet.../3	0.619	0.11	0.763	0.5	0.5
□	✓ 4 days ago	2.9s	mlops	dandi	train_and...	f96d74	ElasticNet.../2	0.655	0.013	0.803	0.9	0.4
□	✓ 4 days ago	2.9s	mlops	dandi	train_and...	f96d74	ElasticNet.../1	0.66	0.008	0.805	0.88	0.89

- 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.

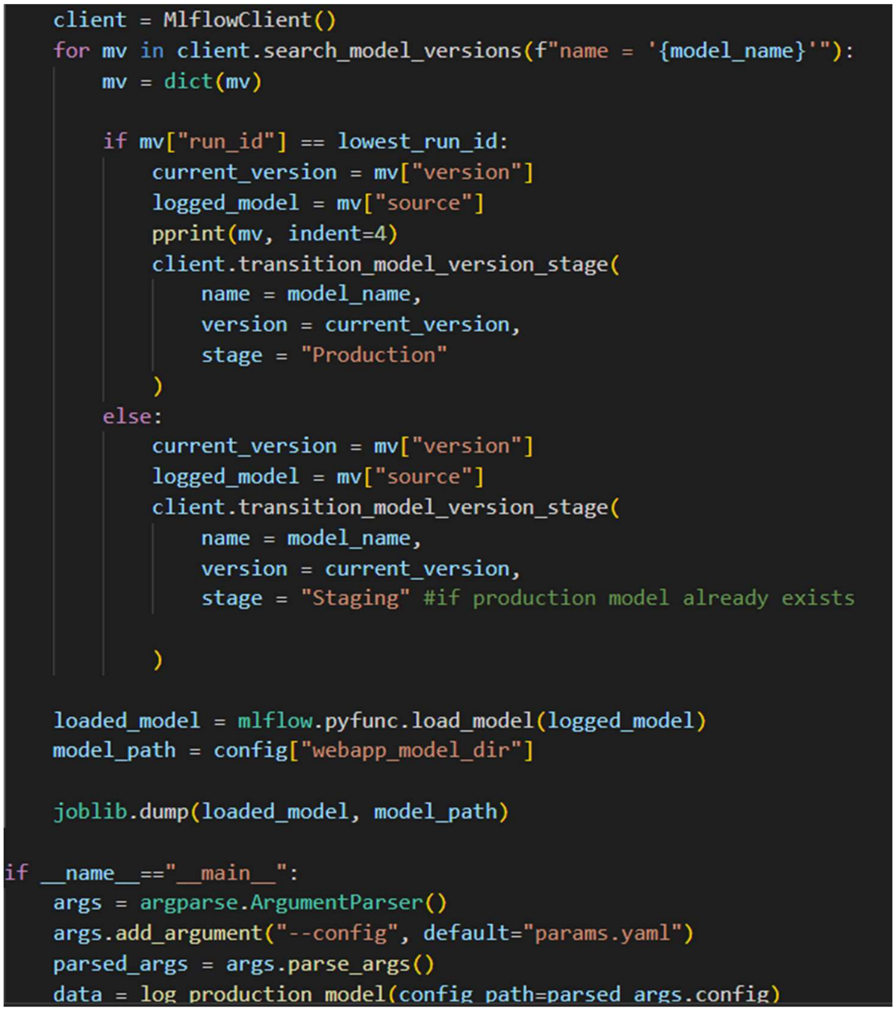


□		Version	Registered at	Created by	Stage	Description
□	✓	Version 7	2023-02-20 12:00:22		Production	
□	✓	Version 6	2023-02-20 11:36:51		Staging	
□	✓	Version 5	2023-02-20 11:32:31		Staging	
□	✓	Version 4	2023-02-20 11:27:21		Staging	
□	✓	Version 3	2023-02-19 14:50:27		Staging	
□	✓	Version 2	2023-02-19 14:44:06		Staging	
□	✓	Version 1	2023-02-19 14:34:15		Staging	

- 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



```
from src.get_data import read_params
import argparse
import mlflow
from mlflow.tracking import MlflowClient
from pprint import pprint
import joblib
import os

def log_production_model(config_path):
    config = read_params(config_path)
    mlflow_config = config["mlflow_config"]

    model_name = mlflow_config["registered_model_name"]

    remote_server_uri = mlflow_config["remote_server_uri"]
    #mlflow.set_registry_uri(remote_server_uri)
    mlflow.set_tracking_uri(remote_server_uri)

    #runs = mlflow.search_runs(experiment_ids=1)
    runs = mlflow.search_runs([1])

    lowest = runs["metrics.mae"].sort_values(ascending=True)[0]
    lowest_run_id = runs[runs["metrics.mae"] == lowest]["run_id"][0]
```



```
    client = MlflowClient()
    for mv in client.search_model_versions(f"name = '{model_name}'"):
        mv = dict(mv)

        if mv["run_id"] == lowest_run_id:
            current_version = mv["version"]
            logged_model = mv["source"]
            pprint(mv, indent=4)
            client.transition_model_version_stage(
                name = model_name,
                version = current_version,
                stage = "Production"
            )
        else:
            current_version = mv["version"]
            logged_model = mv["source"]
            client.transition_model_version_stage(
                name = model_name,
                version = current_version,
                stage = "Staging" #if production model already exists

            )

    loaded_model = mlflow.pyfunc.load_model(logged_model)
    model_path = config["webapp_model_dir"]

    joblib.dump(loaded_model, model_path)

if __name__=="__main__":
    args = argparse.ArgumentParser()
    args.add_argument("--config", default="params.yaml")
    parsed_args = args.parse_args()
    data = log_production_model(config_path=parsed_args.config)
```

After that, the new model will be automatically deployed to Heroku.