МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное   
образовательное учреждение высшего образования  
«Самарский национальный исследовательский университет   
имени академика С.П. Королева»

(Самарский университет)

Институт информатики и кибернетики

Факультет информатики  
Кафедра технической кибернетики

**Отчет по лабораторной работе №2**

Дисциплина: «Инженерия данных»

Тема: **«Airflow и MLflow - логгирование экспериментов и версионирование моделей»**

Выполнили:

Мелешенко И.С.

Коршиков В.И.

Группа: 6233-010402D

Самара 2023

Содержание

[Часть 1. Подготовка к выполнению лабораторной работы. 3](#_Toc153270509)

[Часть 2. Построение пайплайна, который обучает любой классификатор из sklearn по заданному набору параметров 4](#_Toc153270510)

[Шаг 1. Определение моделей и датасетов для работы. 4](#_Toc153270511)

[Шаг 2. Разработка DAG-a. 5](#_Toc153270512)

[Шаг 3. Разработка вспомогательных модулей. 5](#_Toc153270513)

[Шаг 4. Обучение моделей. 7](#_Toc153270514)

[Шаг 5. Запуск эксперимента 9](#_Toc153270515)

[Часть 2. Построение пайплайна, который выбирает лучшую модель из обученных и производит её хостинг 11](#_Toc153270516)

[Шаг 1. Разработка DAG-a. 11](#_Toc153270517)

[Шаг 2. Разработка кода валидации моделей. 11](#_Toc153270518)

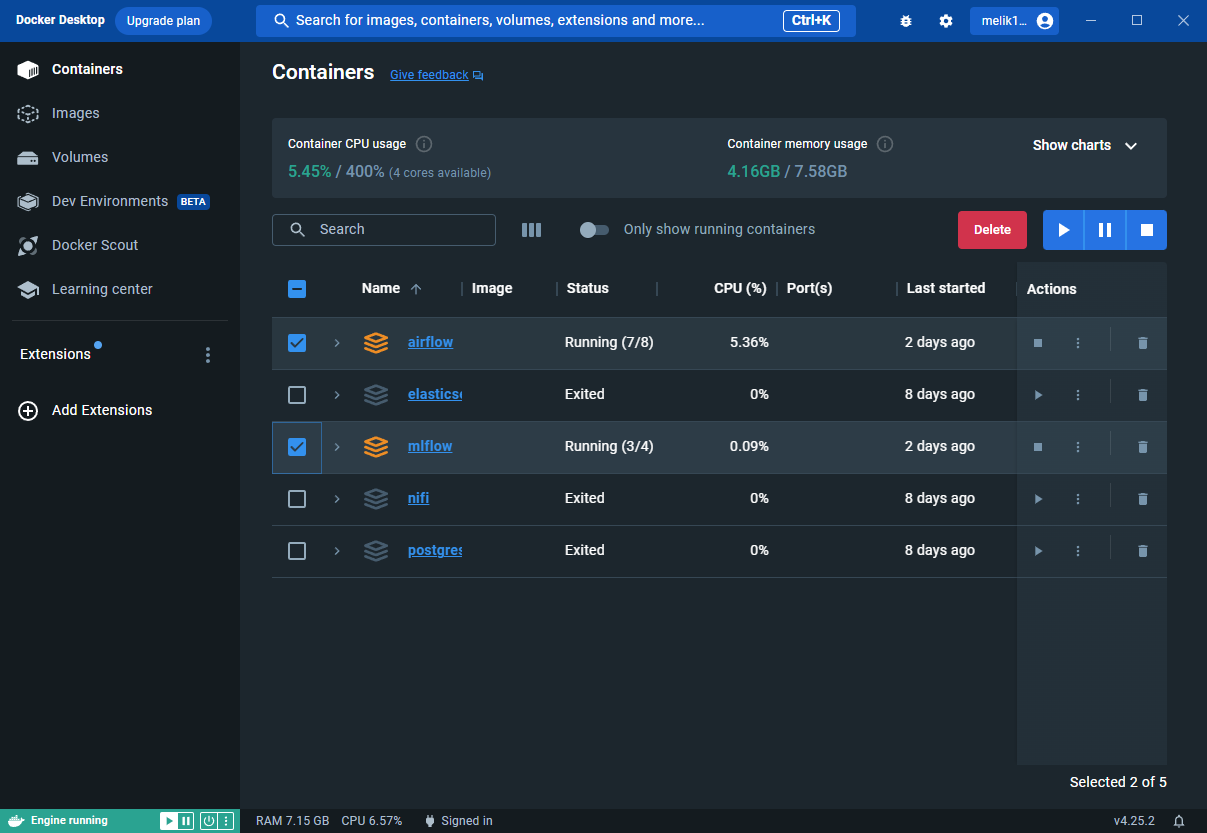
[Шаг 3. Запуск DAG-a валидации моделей. 12](#_Toc153270519)

[Шаг 4. Проверка отработки DAG-a. 12](#_Toc153270520)

[Заключение 13](#_Toc153270521)

# Часть 1. Подготовка к выполнению лабораторной работы.

В данной лабораторной работы нам необходимо реализовать обучение классификаторов из пакета sklearn. Для работы нам понадобятся docker-контейнеры с образами Airflow и Mlfow, а остальные отключить за ненадобностью.

  
Рисунок 1 – Необходимые контейнеры для работы.

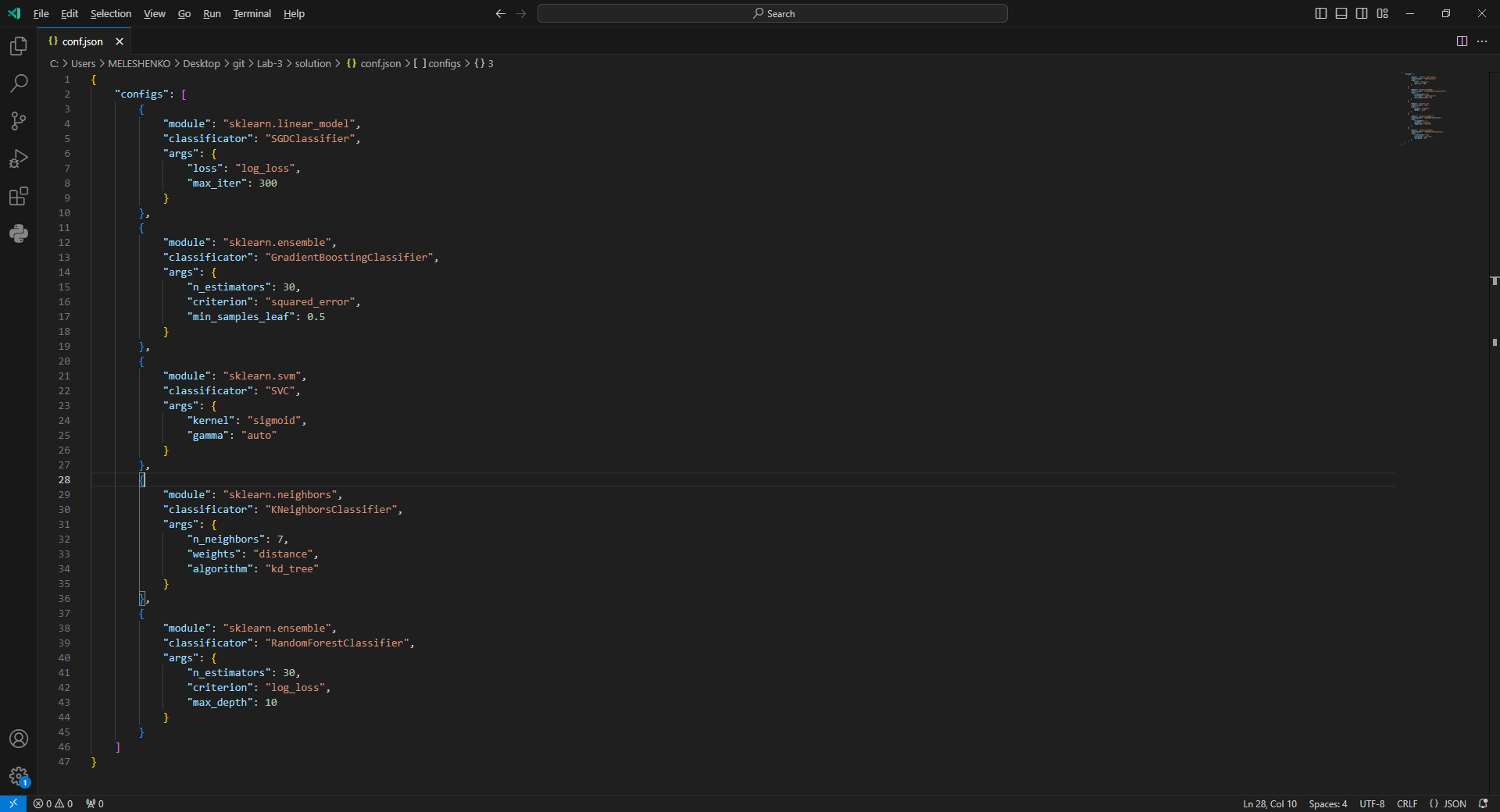
# Часть 2. Построение пайплайна, который обучает любой классификатор из sklearn по заданному набору параметров

## Шаг 1. Определение моделей и датасетов для работы.

Перед тем обучать модели необходимо выбрать их и сформировать конфигурационный файл, с которым будет работать наш DAG. Для выбора моделей воспользуемся официальной документацией по ссылке <https://scikit-learn.org/stable/modules/classes.html>. В процессе изучения данной ссылки мой выбор пал на:

* sklearn.linear\_model.SGDClassifier;
* sklearn.ensemble.GradientBoostingClassifier;
* sklearn.svm.SVC;
* sklearn.neighbors.KNeighborsClassifier;
* sklearn.ensemble.RandomForestClassifier;

С этими моделями мы будем работать. Для начала работы сформируем конфигурационный файл в формате json:

  
Рисунок 2 – Формирование конфигурационного фала с моделями

В качестве источника данных для обучения и валидации моделей был выбран стандартный датасет вин из библиотеки sklearn: load\_wines. Для использования его в процессе работы DAG-a он будет разделен на тестовую, тренировочную и валидационную выборки.

## Шаг 2. Разработка DAG-a.

DAG реализующий пайплайн обучения классификаторов, состоит из 4 task-ов:

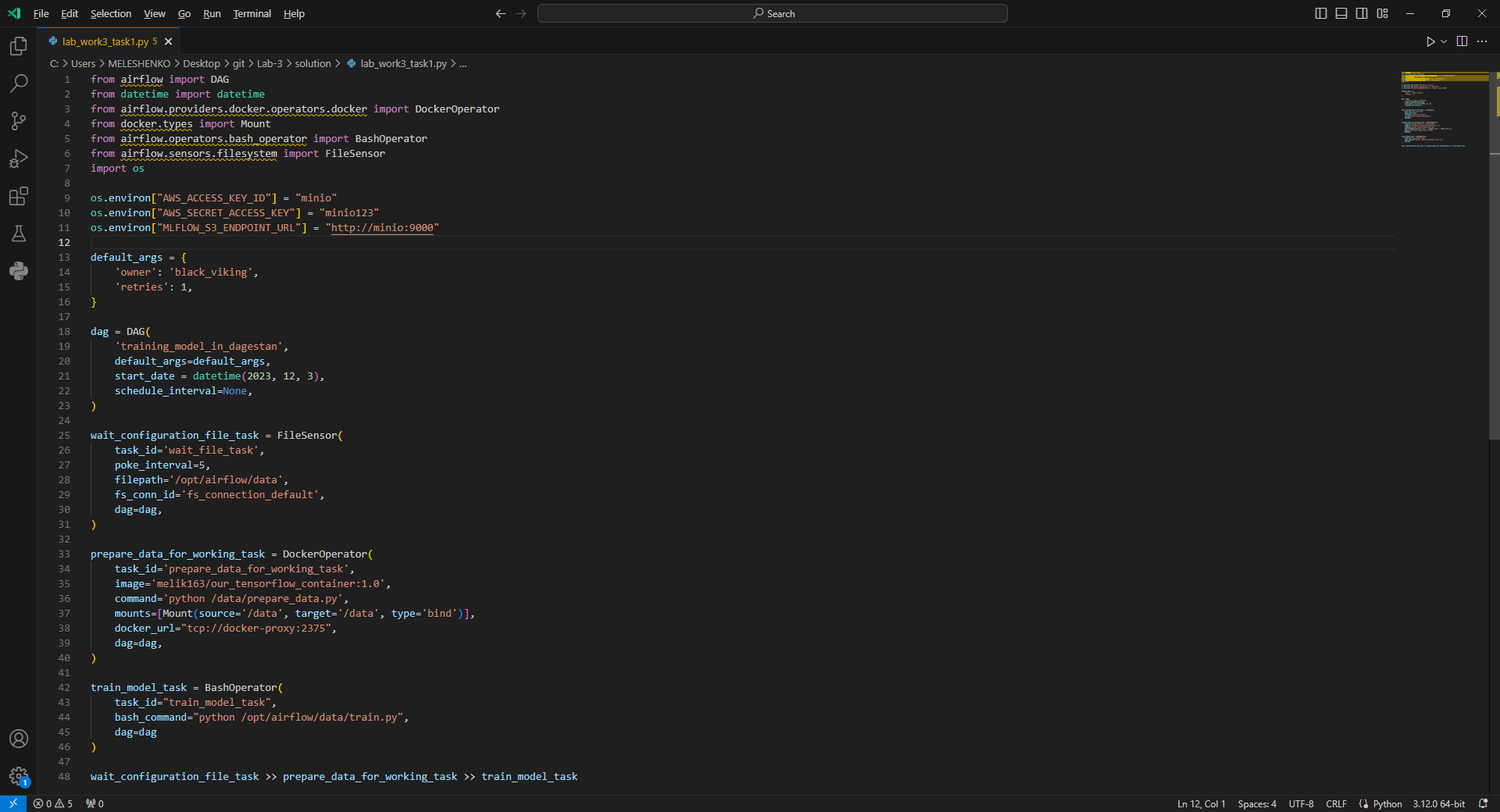
wait\_configuration\_file\_task – осуществляет мониторинг папки, в которой должен появиться конфигурационный файл, с описанием моделей.

prepare\_data\_for\_working\_task – осуществляет подготовку данных, которые будут использоваться в процессе обучение и валидации моделей.

train\_model\_task – осуществляет процесс обучения моделей и их логирование.

О двух последний поговорим подробнее чуть ниже.

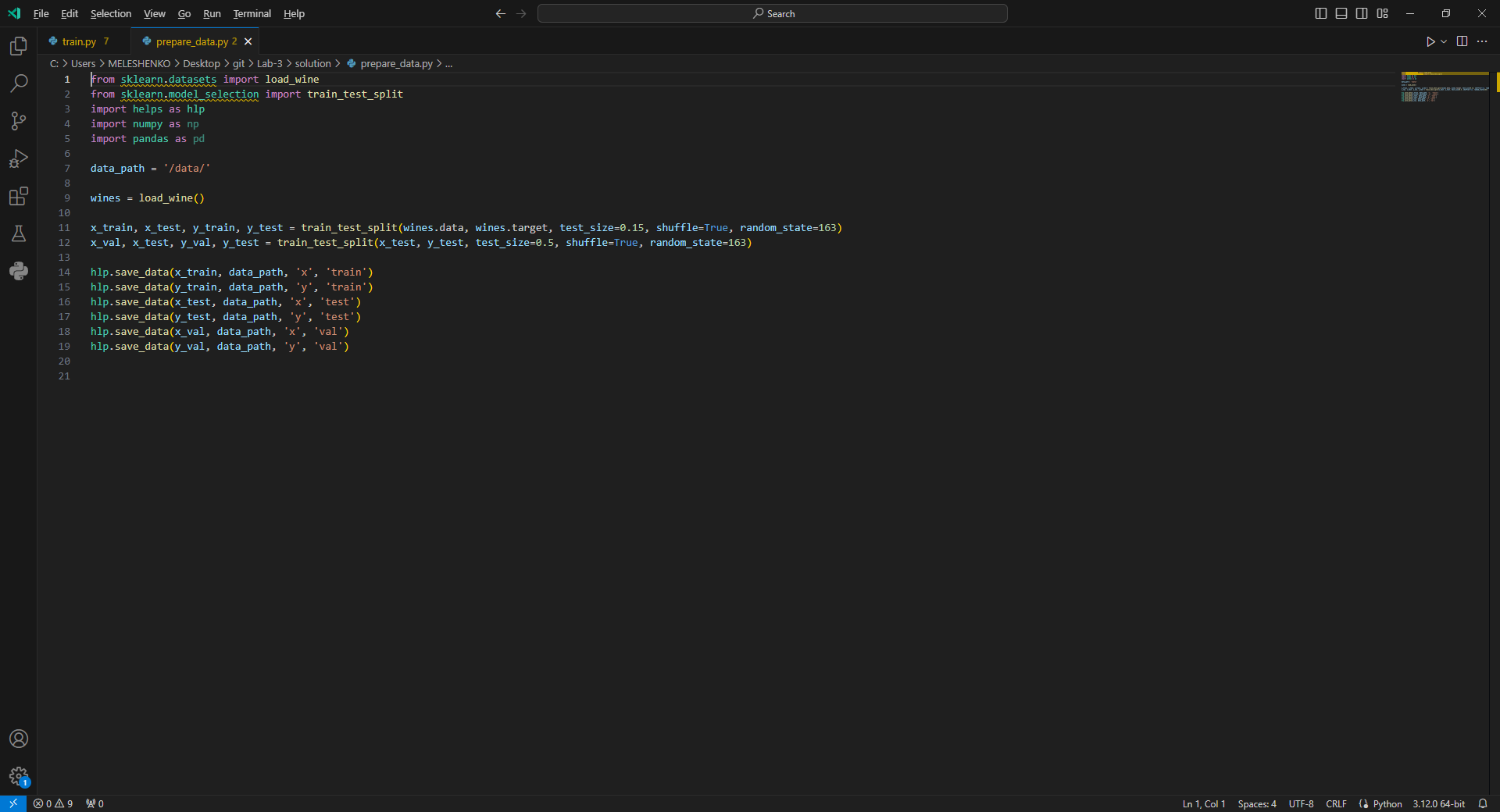
Код DAG-a первой части лабораторной работы приведен ниже, а также в репозитории с решением.

  
Рисунок 3 – Код DAG-a.

## Шаг 3. Разработка вспомогательных модулей.

Task-и prepare\_data\_for\_working\_task и train\_model\_task в своей работе используют подготовленные нами скрипты, рассмотрим их подробнее и начнем с task-a, который осуществляет подготовку данных prepare\_data\_for\_working\_task.

В данной скрипте мы определяем «корневую» папку в которой будем работать. Далее импортируем наш датасет. Это стандартный датасет вин из библиотеки sklearn. После того как подгрузили датасет, произведем его разделение на выборки: обучающую, тестовую и валидационную. Подготовленные данные сохраняем в файлики, для последующего использования. Код описанного скрипта приведен на рисунке ниже, а также в репозитории с решением.

  
Рисунок 4 – Код, осуществляющий подготовку данных

В процессе подготовки данных был использован метод save\_data из модуля helps. Это небольшой вспомогательный скрипт, в который вынесены операции сохранения и получения датасетов из файлов. Рассмотрим их подробнее.

Метод save\_data() необходим для сохранения датасетов в файл. На вход метод принимает 4 параметра:

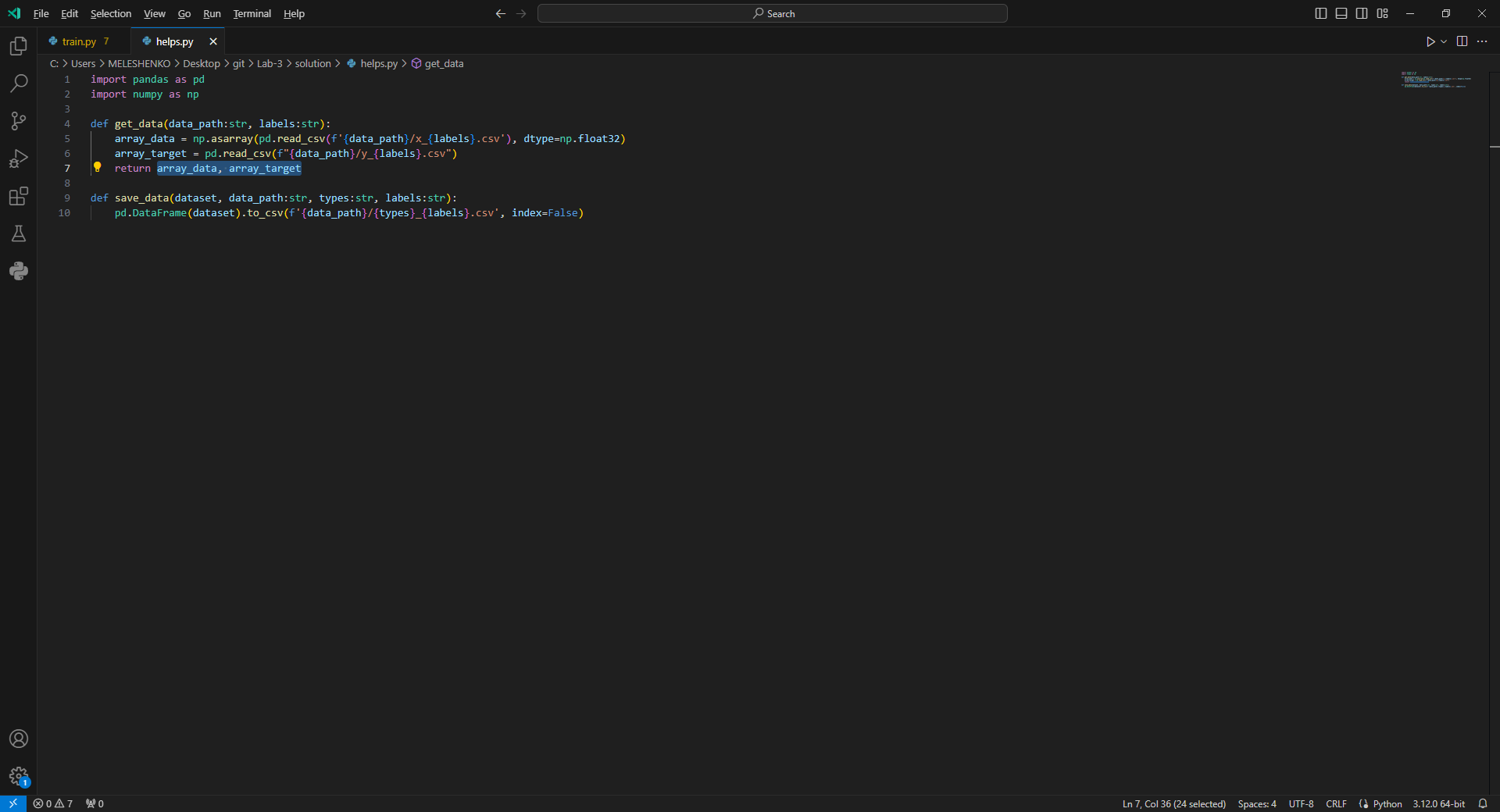
1. dataset - сам датасет, который мы будем сохранять;
2. data\_path – пусть куда будет сохранен файл;
3. types – тип датасета, который мы передаем (х или у);
4. labels – метка выборки (обучающая, тестовая, валидационная).

Метод get\_data() необходим для получения датасетов из файлов. На вход метод принимает 2 параметра:

1. data\_path – путь, откуда необходимо прочитать файл;
2. labels – метка выборки (обучающая, тестовая, валидационная).

На выходе возвращается два датасета: array\_data и array\_target

Реализация данного модуля представлена на рисунке ниже и в репозитории решения лабораторной работы.

  
Рисунок 5 – Код вспомогательного модуля helps

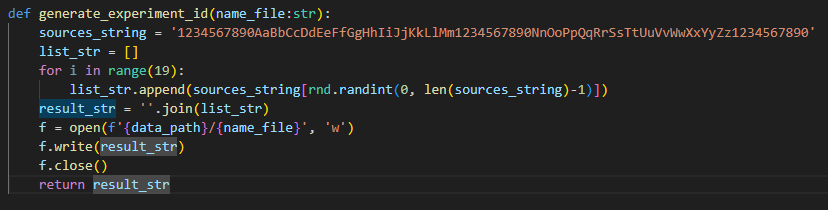
## Шаг 4. Обучение моделей.

Обучение моделей сосредоточено в task-e - train\_model\_task. Рассмотрим его подробнее.

В первую очередь получаем конфигурацию всех моделей, из полученного конфигурационного файла conf.json. Поскольку каждый запуск обучения моделей — это отдельный эксперимент, то для каждого необходим уникальный experiment\_id. Для этого была создана функция генерации этого самого experiment\_id: generate\_experiment\_id().

На вход она принимает название файла, в который будет сохранен наш experiment\_id.

В процессе работы, функция генерирует уникальный experiment\_id, после чего пишет в указанный файл, а также возвращает в качестве выходного значения для продолжения работы. Код функции приведен ниже.

  
Рисунок 6 – Код генерации experiment\_id

Также перед началом обучения, мы создали функцию, которая будет осуществлять логирование метрик моделей в процессе обучения: logirovanie().

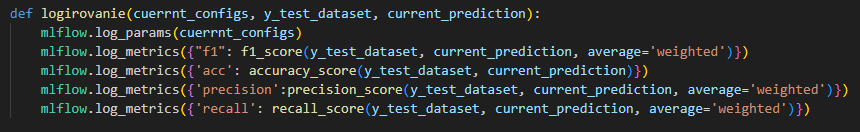
На вход данная функция принимает

1. cuerrnt\_configs – текущая конфигурация модели
2. y\_test\_dataset – датасет с истинными значениями.
3. current\_prediction – датасет с предсказанными значениями.

В процессе обучения производим логирование четырех метрик:

* F1
* Accuracy
* Precision
* Recall

Код приведен на рисунке ниже.

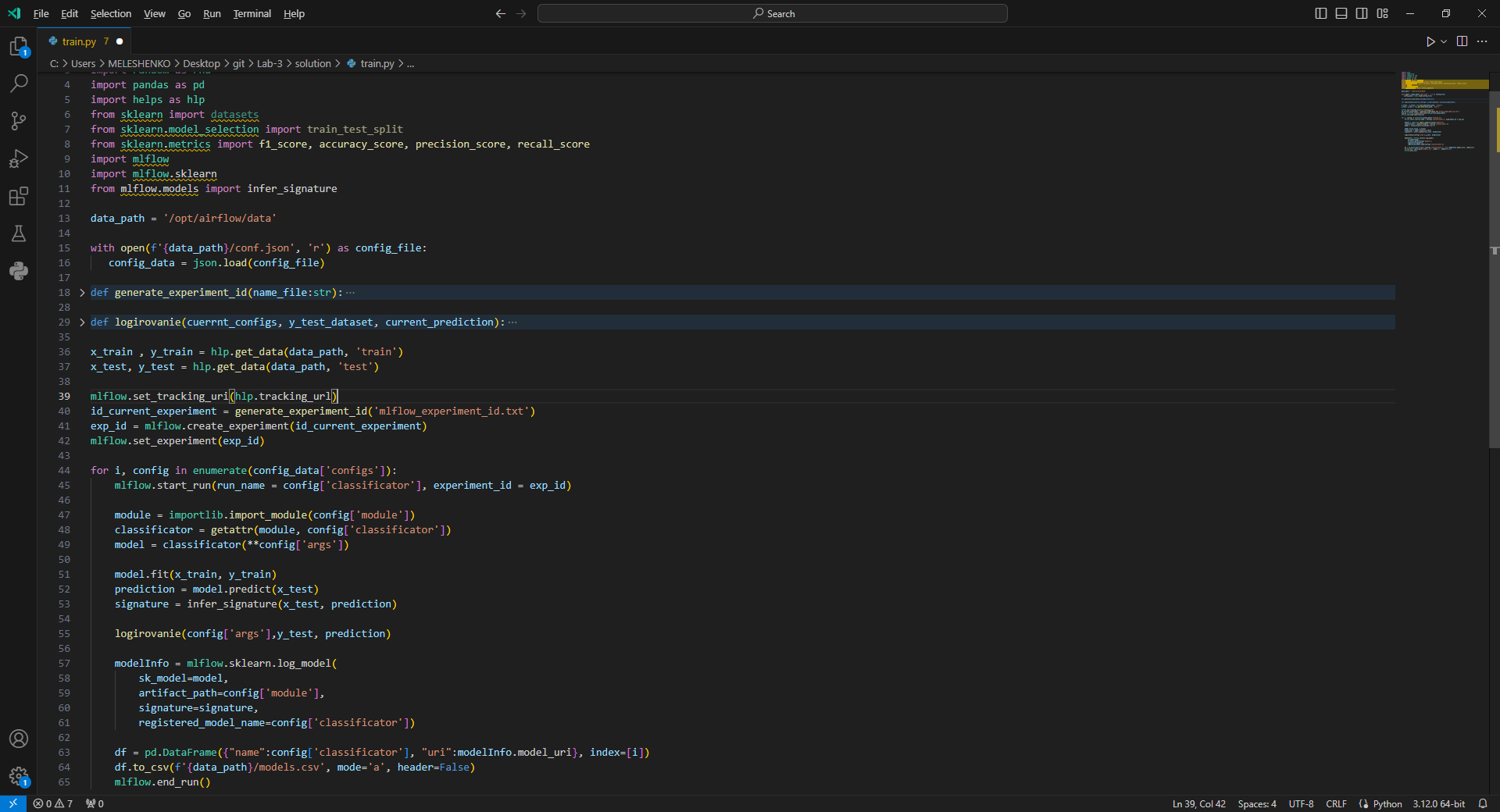
  
Рисунок 7 – Код логирования метрик

После всех приготовлений запускаем эксперимент по обучению моделей. При помощи функции get\_data() из самописного модуля helps получаем данные для обучения моделей. Устанавливаем подключение к mlflow. Генерируем experiment\_id при помощи функции generate\_experiment\_id() и подключаемся к эксперименту с только что созданным experiment\_id.

После чего в цикле для каждой модели проделываем следующие операции:

1. Получаем конфигурацию модели
2. Проводим процесс обучения
3. Логируем метрики
4. Логируем модель
5. Лог модели пишем в файл для дальнейшего использования во второй части лабораторной работы.

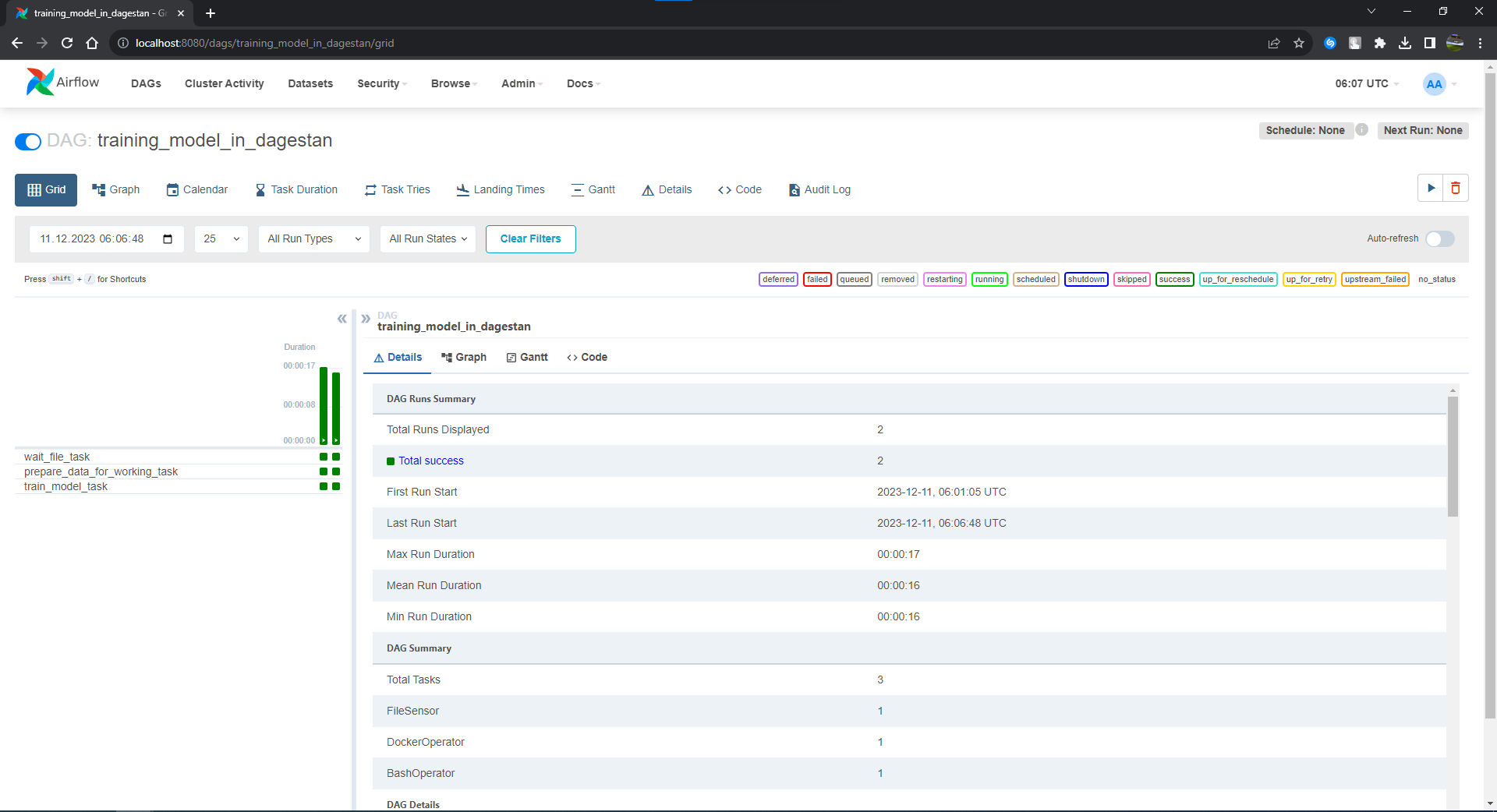
Код обучения моделей представлен на рисунке ниже, а также полная версия кода размещена в репозитории с решением лабораторной работы.

  
Рисунок 8 – Код обучения моделей

## Шаг 5. Запуск эксперимента

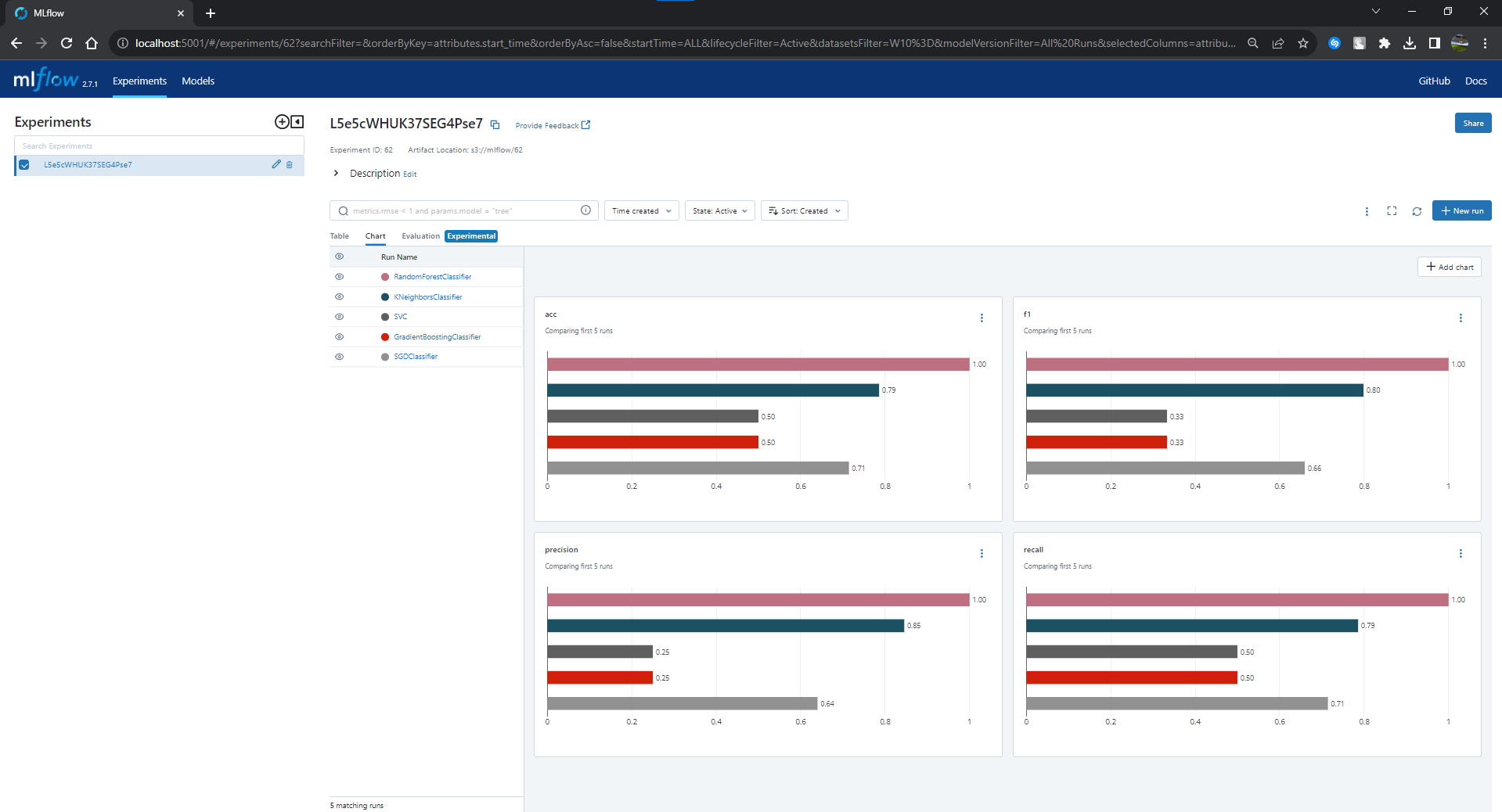
Теперь, когда весь код подготовлен, можно перейти в Airflow, и запустить наш DAG по обучению моделей.

Рисунок 9 – Запуск DAG-a по обучению моделей.



После отработки DAG-a перейдем в Mlflow по адресу <http://localhost:5001>, и посмотрим на результаты обучения, которые изображены на рисунке ниже.

Рисунок 10 – Результат обучения моделей

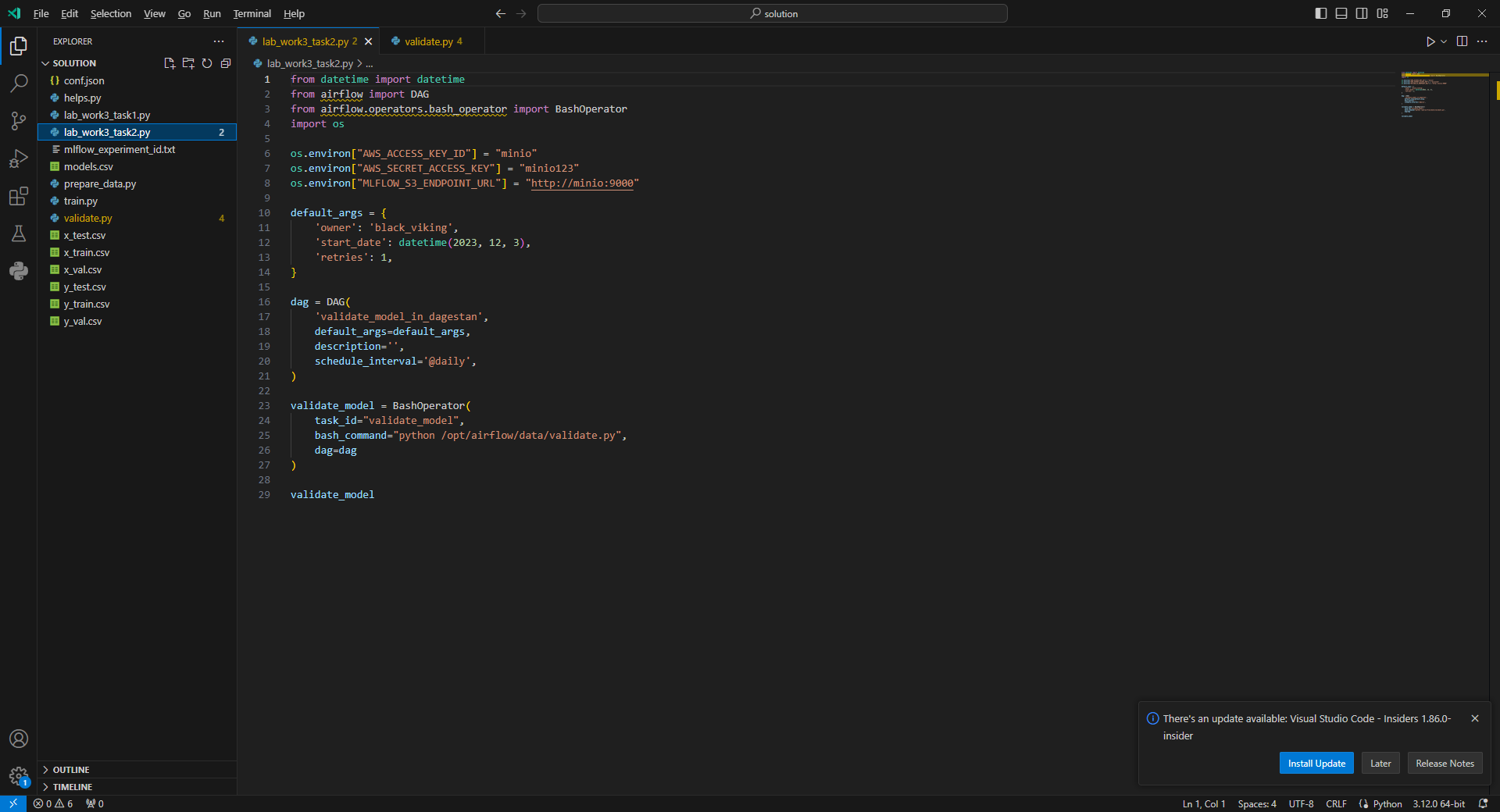


В результате мы получили набор обученных моделей, которые будут провалидированы на следующем этапе лабораторной работы.

# Часть 2. Построение пайплайна, который выбирает лучшую модель из обученных и производит её хостинг

## Шаг 1. Разработка DAG-a.

DAG валидации моделей состоит из одного task-a, который осуществляет процесс валидации моделей. Код DAG-a представлен на рисунке ниже, а также в репозитории с решением лабораторной работы.

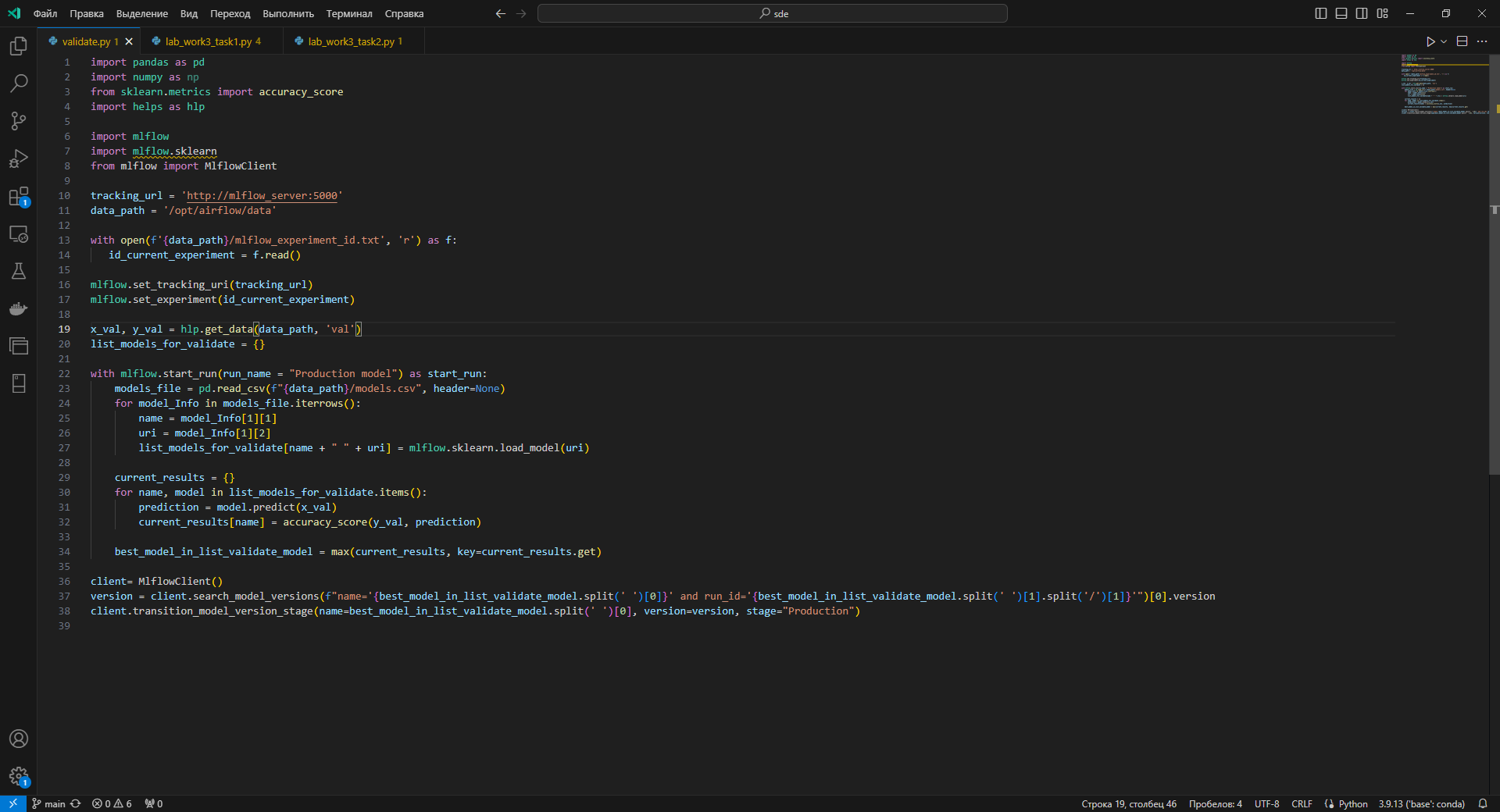
  
Рисунок 11 – DAG валидации моделей

## Шаг 2. Разработка кода валидации моделей.

После того как подготовили DAG, перейдем к разработке кода, осуществляющего процесс валидации модели. В начале получаем experiment\_id, который был сохранен в файл, на этапе обучения моделей, а после подключаемся к уже существующему эксперименту.

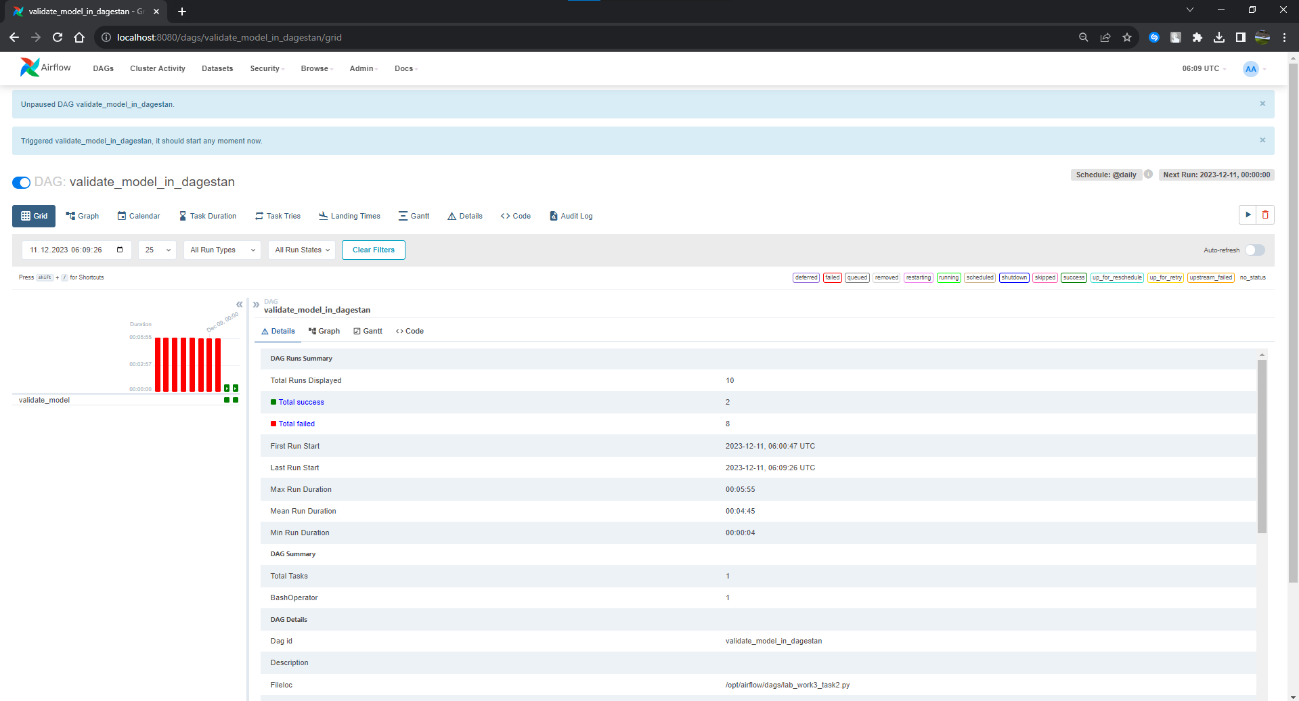
После подключения подгружаем данные для валидации моделей, которые были сохранены на этапе подготовки данных. Используя список сохраненных моделей на этапе обучения, определяем лучшую, на основании показателя “Accuracy” и выводим в ее в состояние “Production”. В итоге среди всех моделей хотя бы одна модель должна быть в состоянии “Production”.

Код валидации моделей представлен на рисунке ниже, а также в репозитории решения лабораторной работы.

  
Рисунок 12 – Код валидации моделей

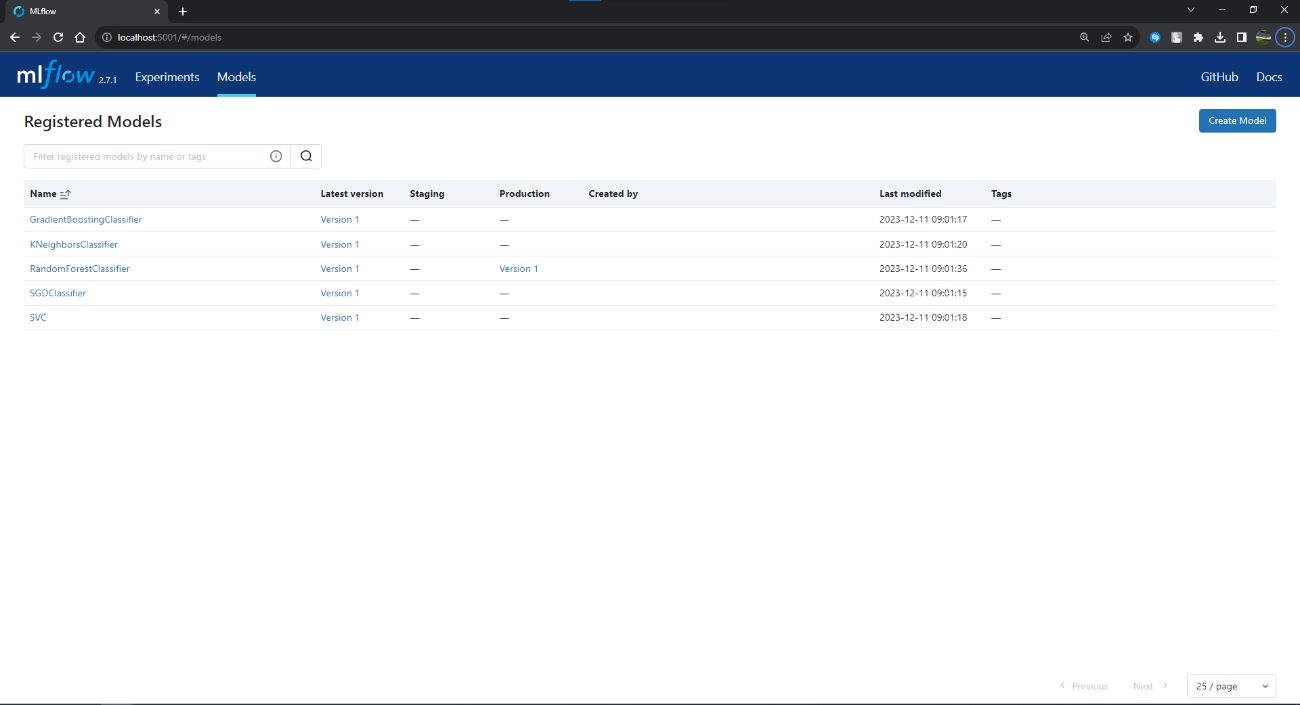
## Шаг 3. Запуск DAG-a валидации моделей.

После окончания всех приготовлений перейдем в Airflow, для запуска DAG-a, который осуществит валидацию моделей. На рисунке ниже мы можем наблюдать успешную отработку кода.

  
Рисунок 13 – Запуск DAG-а валидации моделей

## Шаг 4. Проверка отработки DAG-a.

Для того чтобы убеиться, в корректности отработки нашего DAG-a, перейдем в Mlflow, и проверим какие статусы имеют зарегистированные модели. На рисунке ниже наблюдаем, что модель RandomForestClassifier перешла в состояние “Production”, что говорит об успешной отработке DAG-a.

  
Рисунок 13 – Результат выполнения, DAG-а валидации моделей

# Заключение

В результате выполнения лабораторной работы получены навыки работы с логированием моделей и выводом моделей в “Production”.