

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  
Кафедра технической кибернетики

**Отчет по лабораторной работе №3**

Дисциплина: «Инженерия данных»

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

Выполнил: Дубман Л.Б.

Группа: 6233-010402D

Самара 2024

## Содержание

Часть 1. Подготовка к выполнению лабораторной работы. ....	3
Часть 2. Построение пайплайна, который обучает любой классификатор из sklearn по заданному набору параметров.....	4
Шаг 1. Определение моделей и датасетов для работы. ....	4
Шаг 2. Разработка DAG. ....	6
Шаг 3. Разработка вспомогательных модулей. ....	7
Шаг 4. Обучение моделей. ....	9
Шаг 5. Запуск эксперимента.....	12
Часть 2. Построение пайплайна, который выбирает лучшую модель из обученных и производит её хостинг .....	15
Шаг 1. Разработка DAG. ....	15
Шаг 2. Разработка кода валидации моделей. ....	16
Шаг 3. Запуск DAG валидации моделей. ....	17
Шаг 4. Проверка отработки DAG.....	18
Заключение .....	19

## Часть 1. Подготовка к выполнению лабораторной работы.

В данной лабораторной работы необходимо реализовать обучение классификаторов из пакета `sklearn`. Для работы понадобятся docker-контейнеры с образами Airflow и Mlfow.

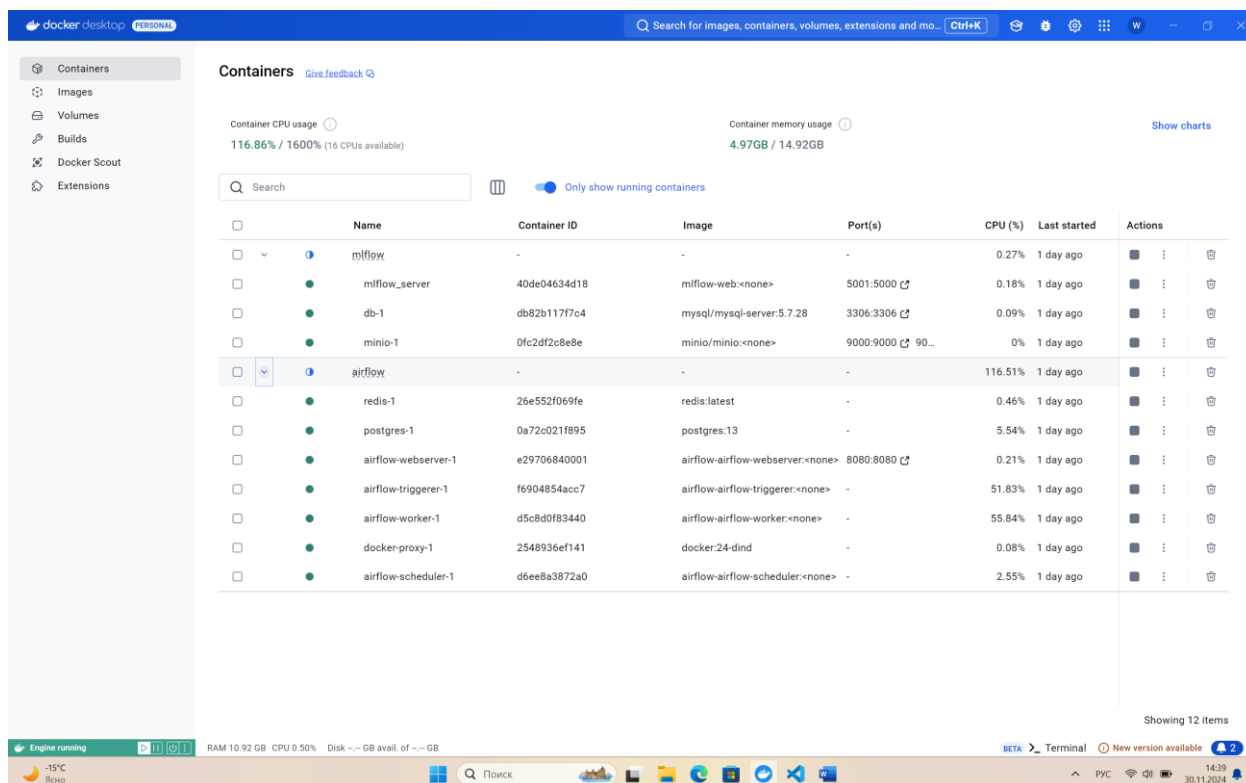


Рисунок 1 – Необходимые контейнеры для работы.

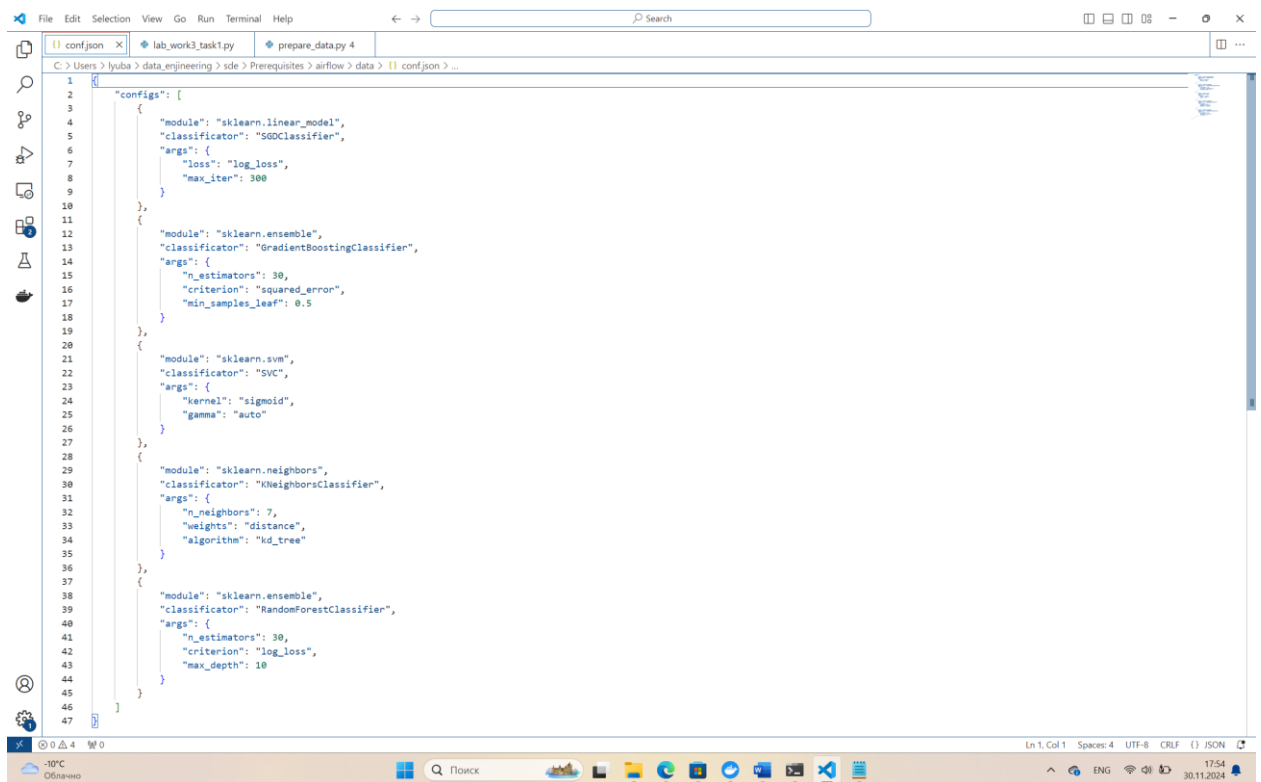
## Часть 2. Построение пайплайна, который обучает любой классификатор из sklearn по заданному набору параметров

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

Перед тем обучать модели необходимо выбрать их и сформировать конфигурационный файл, с которым будет работать 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 я и буду работать. Для начала работы сформируем конфигурационный файл в формате json:



```
1  {
2    "configs": [
3      {
4        "module": "sklearn.linear_model",
5        "classifier": "SGDClassifier",
6        "args": {
7          "loss": "log_loss",
8          "max_iter": 300
9        }
10     },
11     {
12       "module": "sklearn.ensemble",
13       "classifier": "GradientBoostingClassifier",
14       "args": {
15         "n_estimators": 30,
16         "criterion": "squared_error",
17         "min_samples_leaf": 0.5
18       }
19     },
20     {
21       "module": "sklearn.svm",
22       "classifier": "SVC",
23       "args": {
24         "kernel": "sigmoid",
25         "gamma": "auto"
26       }
27     },
28     {
29       "module": "sklearn.neighbors",
30       "classifier": "KNeighborsClassifier",
31       "args": {
32         "n_neighbors": 7,
33         "weights": "distance",
34         "algorithm": "kd_tree"
35       }
36     },
37     {
38       "module": "sklearn.ensemble",
39       "classifier": "RandomForestClassifier",
40       "args": {
41         "n_estimators": 30,
42         "criterion": "log_loss",
43         "max_depth": 10
44       }
45     }
46   ]
47 }
```

Рисунок 2 – Формирование конфигурационного файла с моделями

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

## Шаг 2. Разработка DAG.

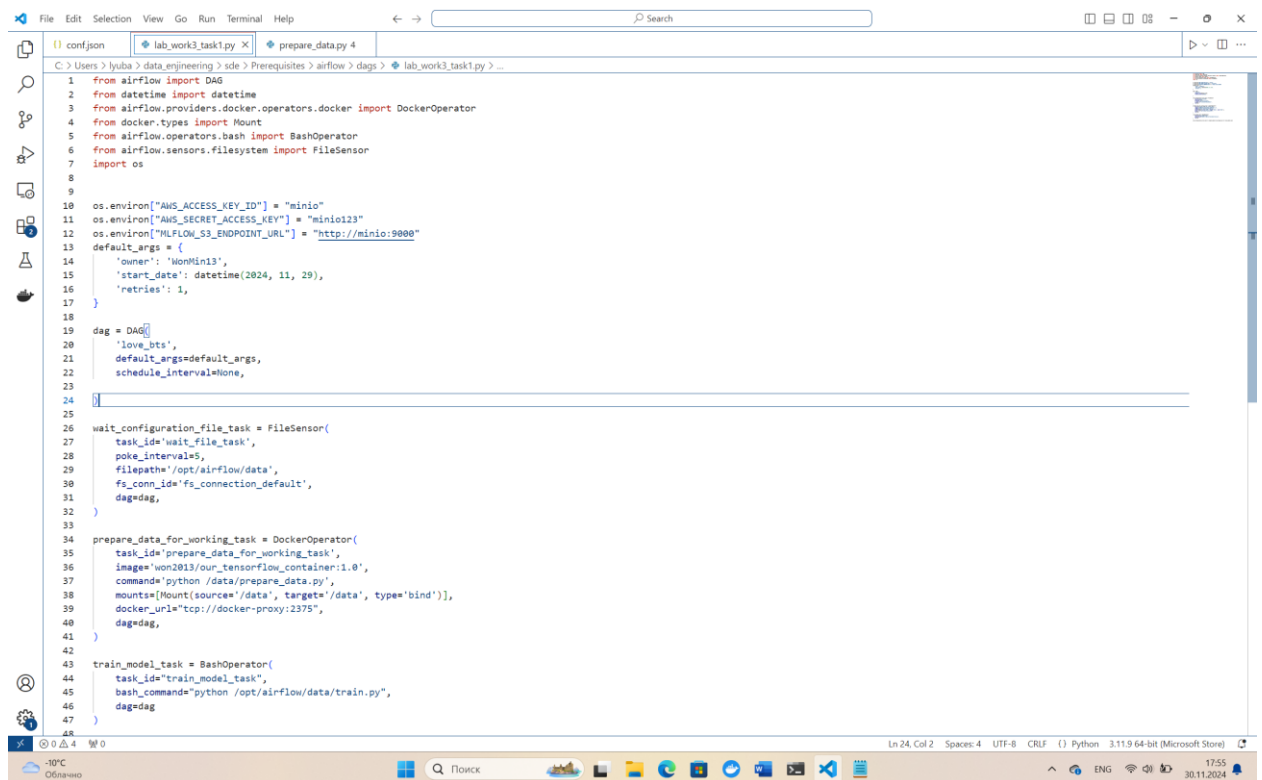
DAG реализующий пайплайн обучения классификаторов, состоит из 4 task:

*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 обучения и валидации моделей.

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

Код DAG первой части лабораторной работы приведен ниже.

The image shows a screenshot of a code editor window with a dark theme. The editor displays Python code for an Airflow DAG. The code includes imports for DAG, datetime, DockerOperator, Mount, BashOperator, FileSensor, and os. It sets environment variables for AWS credentials and the MLFLOW endpoint. A default\_args dictionary is defined with owner, start\_date, and retries. The DAG is created with a name 'love\_bts' and default\_args. Three tasks are defined: 'wait\_configuration\_file\_task' (FileSensor), 'prepare\_data\_for\_working\_task' (DockerOperator), and 'train\_model\_task' (BashOperator). The tasks are connected in a sequence. The editor's interface includes a sidebar with icons for Explorer, Search, and Run, and a status bar at the bottom showing file encoding and line numbers.

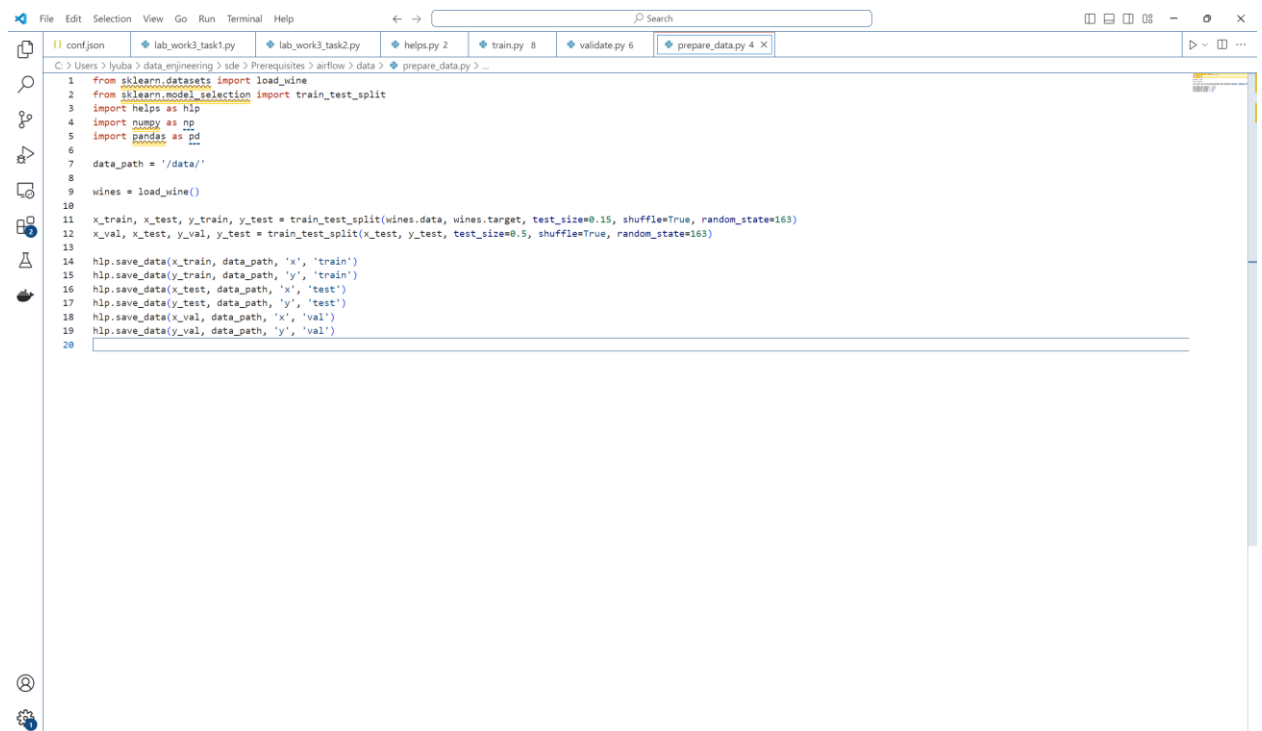
```
1 from airflow import DAG
2 from datetime import datetime
3 from airflow.providers.docker.operators.docker import DockerOperator
4 from docker.types import Mount
5 from airflow.operators.bash import BashOperator
6 from airflow.sensors.filesystem import FileSensor
7 import os
8
9
10 os.environ["AWS_ACCESS_KEY_ID"] = "minio"
11 os.environ["AWS_SECRET_ACCESS_KEY"] = "minio123"
12 os.environ["MLFLOW_S3_ENDPOINT_URL"] = "http://minio:9000"
13 default_args = {
14     'owner': 'NonMin13',
15     'start_date': datetime(2024, 11, 29),
16     'retries': 1,
17 }
18
19 dag = DAG(
20     'love_bts',
21     default_args=default_args,
22     schedule_interval=None,
23 )
24
25
26 wait_configuration_file_task = FileSensor(
27     task_id='wait_file_task',
28     poke_interval=5,
29     filepath='/opt/airflow/data',
30     fs_conn_id='fs_connection_default',
31     dag=dag,
32 )
33
34 prepare_data_for_working_task = DockerOperator(
35     task_id='prepare_data_for_working_task',
36     image='won2013/our_tensorflow_container:1.0',
37     command='python /data/prepare_data.py',
38     mounts=[Mount(source='/data', target='/data', type='bind')],
39     docker_url='tcp://docker-proxy:2375',
40     dag=dag,
41 )
42
43 train_model_task = BashOperator(
44     task_id='train_model_task',
45     bash_command='python /opt/airflow/data/train.py',
46     dag=dag,
47 )
```

Рисунок 3 – Код DAG.

### Шаг 3. Разработка вспомогательных модулей.

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

В данном скрипте определяем «корневую» папку, в которой будем работать. Далее импортируем 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 файлы, для последующего использования. Код описанного скрипта приведен на рисунке ниже.



```
1 from sklearn.datasets import load_wine
2 from sklearn.model_selection import train_test_split
3 import helpers as hlp
4 import numpy as np
5 import pandas as pd
6
7 data_path = '/data/'
8
9 wines = load_wine()
10
11 x_train, x_test, y_train, y_test = train_test_split(wines.data, wines.target, test_size=0.15, shuffle=True, random_state=163)
12 x_val, x_test, y_val, y_test = train_test_split(x_test, y_test, test_size=0.5, shuffle=True, random_state=163)
13
14 hlp.save_data(x_train, data_path, 'x', 'train')
15 hlp.save_data(y_train, data_path, 'y', 'train')
16 hlp.save_data(x_test, data_path, 'x', 'test')
17 hlp.save_data(y_test, data_path, 'y', 'test')
18 hlp.save_data(x_val, data_path, 'x', 'val')
19 hlp.save_data(y_val, data_path, 'y', 'val')
20
```

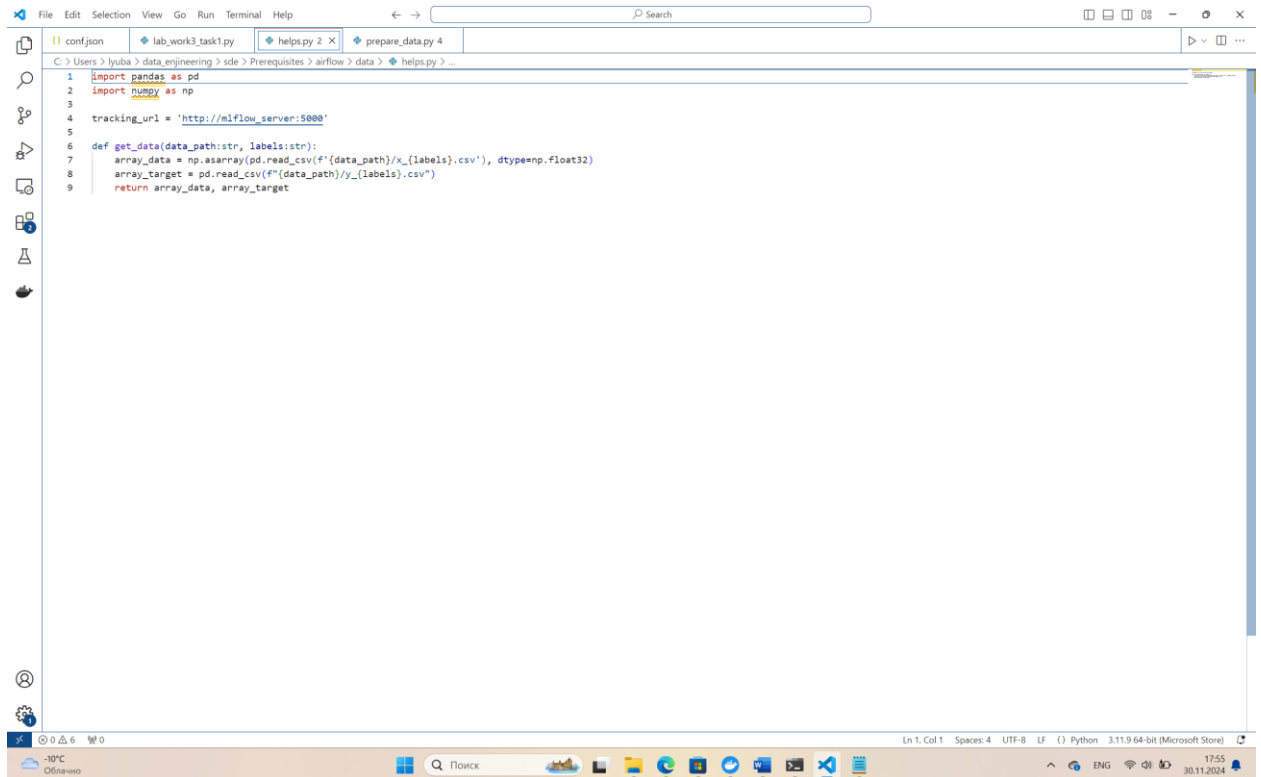
Рисунок 4 – Код, осуществляющий подготовку данных

В процессе подготовки данных был использован метод save\_data из модуля helpers. 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 будем сохранять;
- 2) data\_path – пусть куда будет сохранен файл;
- 3) types – тип датасета, который передаем (x или y);
- 4) labels – метка выборки (обучающая, тестовая, валидационная).

Реализация `get_data()` представлена на рисунке ниже.



```
1 import pandas as pd
2 import numpy as np
3
4 tracking_url = 'http://mlflow_server:5000'
5
6 def get_data(data_path:str, labels:str):
7     array_data = np.asarray(pd.read_csv(f'{data_path}/x_{labels}.csv'), dtype=np.float32)
8     array_target = pd.read_csv(f'{data_path}/y_{labels}.csv')
9     return array_data, array_target
```

Рисунок 5 – Код вспомогательного модуля helps

Метод `get_data()` необходим для получения датасетов из файлов. На вход метод принимает 2 параметра:

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

На выходе возвращается два датасета: `array_data` и `array_target`



#### Шаг 4. Обучение моделей.

Обучение моделей сосредоточено в task - 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 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.

```
def generate_experiment_id(name_file:str):
    sources_string = '1234567890AaBbCcDdEeFfGgHhIiJjKkLlMm1234567890NnOoPpQqRrSsTtUuVvWwXxYyZz1234567890'
    list_str = []
    for i in range(19):
        list_str.append(sources_string[rnd.randint(0, len(sources_string)-1)])
    result_str = ''.join(list_str)
    f = open(f'{data_path}/{name_file}', 'w')
    f.write(result_str)
    f.close()
    return result_str
```

Рисунок 6 – Код генерации experiment\_id

Также перед началом обучения, была создана функция, которая будет осуществлять логирование метрик моделей в процессе обучения: logirovanie().

На вход данная функция принимает

- 1) current\_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.

```
def logirovanie(current_configs, y_test_dataset, current_prediction):  
    mlflow.log_params(current_configs)  
    mlflow.log_metrics({"f1": f1_score(y_test_dataset, current_prediction, average='weighted')})  
    mlflow.log_metrics({'acc': accuracy_score(y_test_dataset, current_prediction)})  
    mlflow.log_metrics({'precision': precision_score(y_test_dataset, current_prediction, average='weighted')})  
    mlflow.log_metrics({'recall': recall_score(y_test_dataset, current_prediction, average='weighted')})
```

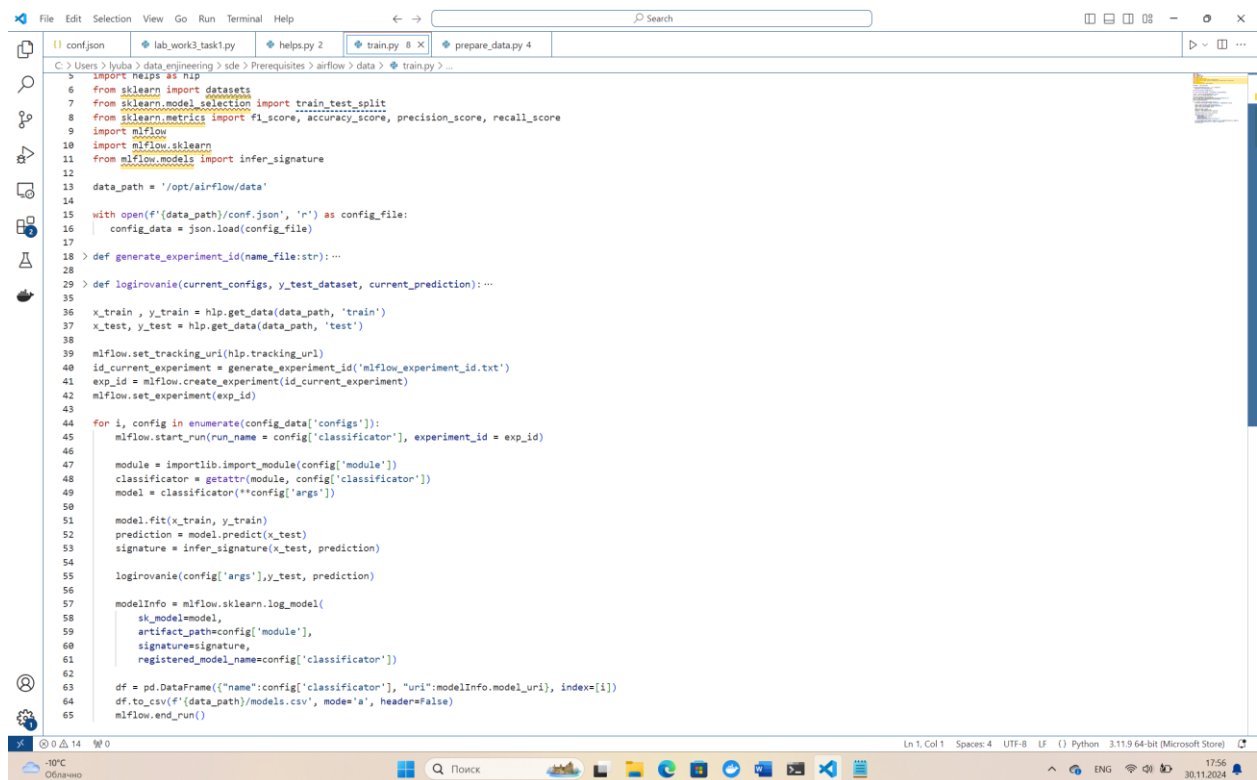
Рисунок 7 – Код логирования метрик

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.



```
File Edit Selection View Go Run Terminal Help
C:\Users\lyuba> data_engineering > sde > Prerequisites > airflow > data > trainpy > ...

6 from sklearn import datasets
7 from sklearn.model_selection import train_test_split
8 from sklearn.metrics import f1_score, accuracy_score, precision_score, recall_score
9 import mlflow
10 import mlflow.sklearn
11 from mlflow.models import infer_signature
12
13 data_path = '/opt/airflow/data'
14
15 with open(f'{data_path}/conf.json', 'r') as config_file:
16     config_data = json.load(config_file)
17
18 > def generate_experiment_id(name_file:str):...
19
20 > def logirovanie(current_configs, y_test_dataset, current_prediction):...
21
22 x_train, y_train = hlp.get_data(data_path, 'train')
23 x_test, y_test = hlp.get_data(data_path, 'test')
24
25 mlflow.set_tracking_uri(hlp.tracking_url)
26 id_current_experiment = generate_experiment_id('mlflow_experiment_id.txt')
27 exp_id = mlflow.create_experiment(id_current_experiment)
28 mlflow.set_experiment(exp_id)
29
30 for i, config in enumerate(config_data['configs']):
31     mlflow.start_run(run_name = config['classifier'], experiment_id = exp_id)
32
33     module = importlib.import_module(config['module'])
34     classifier = getattr(module, config['classifier'])
35     model = classifier(**config['args'])
36
37     model.fit(x_train, y_train)
38     prediction = model.predict(x_test)
39     signature = infer_signature(x_test, prediction)
40
41     logirovanie(config['args'], y_test, prediction)
42
43     modelInfo = mlflow.sklearn.log_model(
44         sk_model=model,
45         artifact_path=config['module'],
46         signature=signature,
47         registered_model_name=config['classifier'])
48
49 df = pd.DataFrame({'name':config['classifier'], 'uri':modelInfo.model_uri}, index=[i])
50 df.to_csv(f'{data_path}/models.csv', mode='a', header=False)
51 mlflow.end_run()
```

Рисунок 8 – Код обучения моделей

## Шаг 5. Запуск эксперимента

После подготовки кода переходим в Airflow и запускаем DAG по обучению моделей.

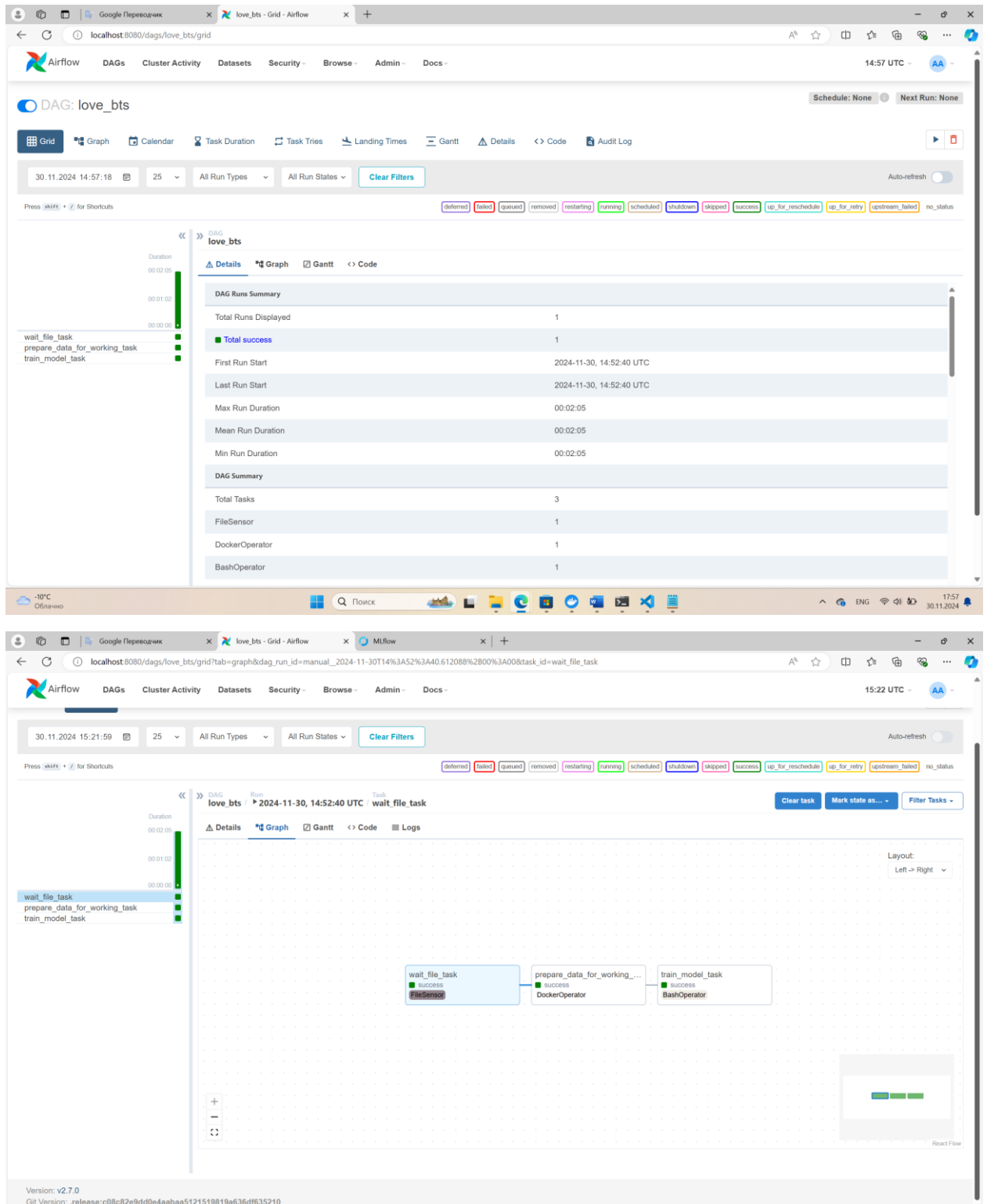
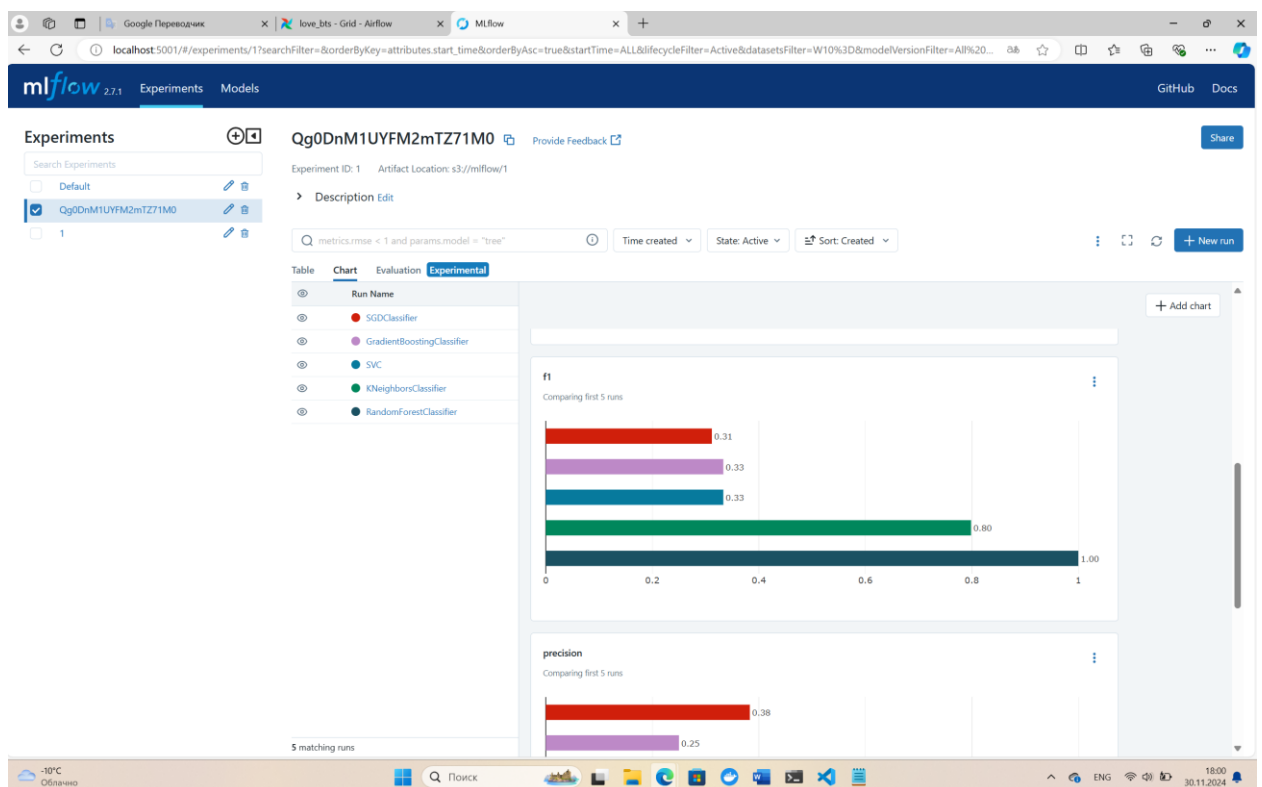
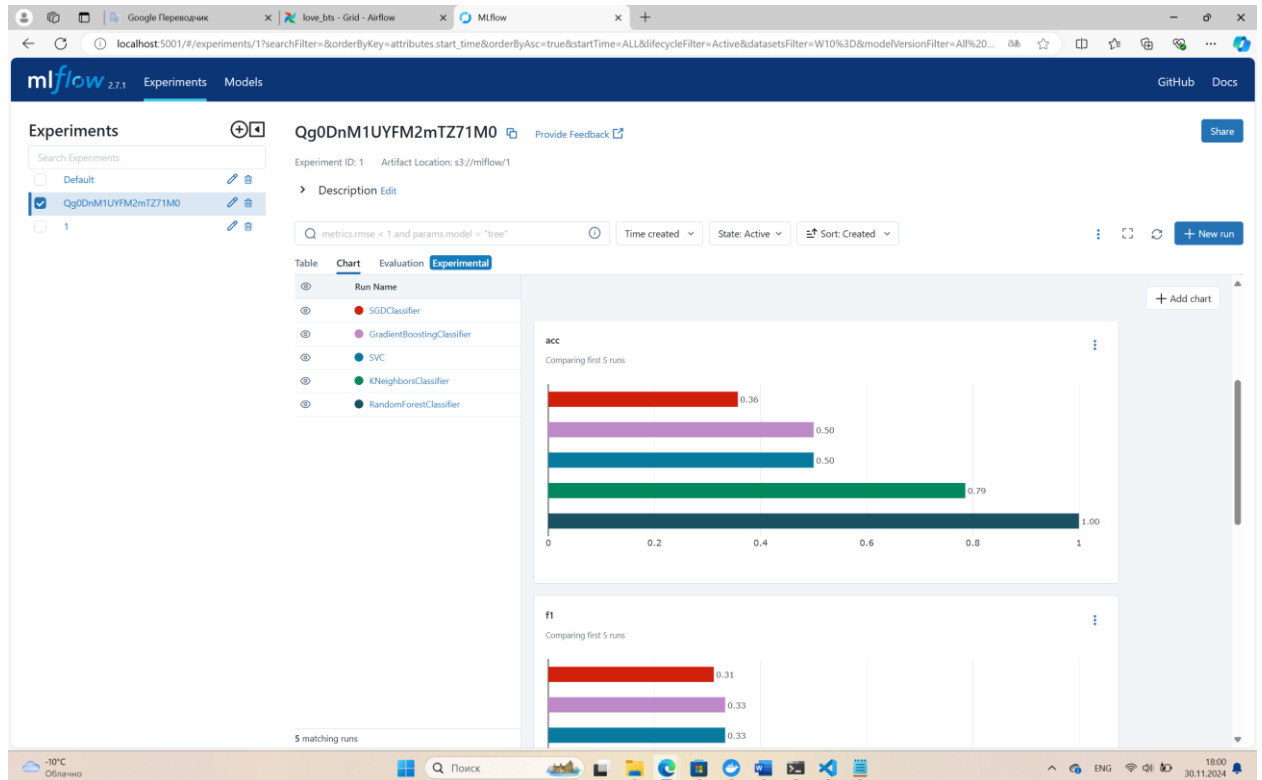


Рисунок 9 – Запуск DAG по обучению моделей.

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



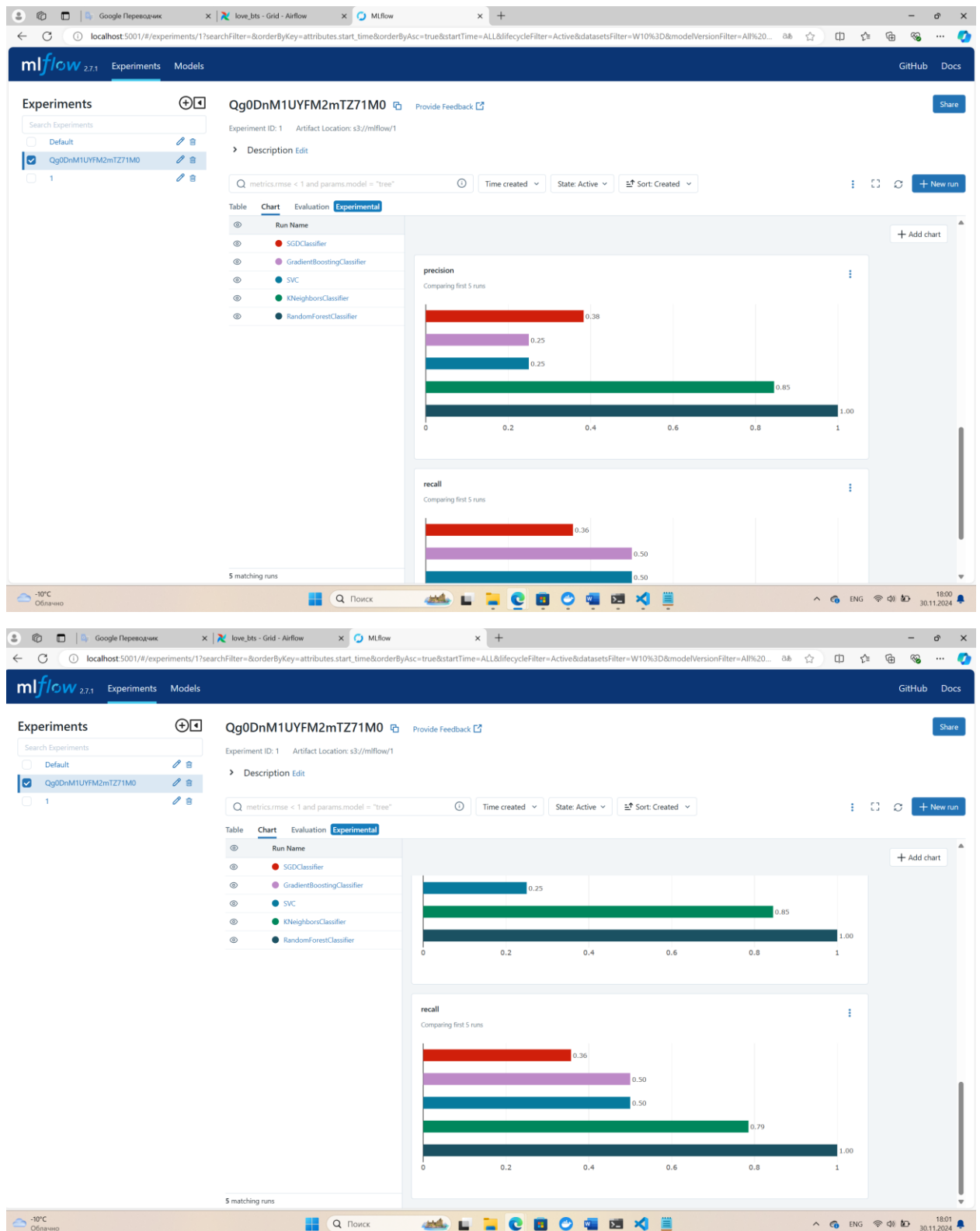


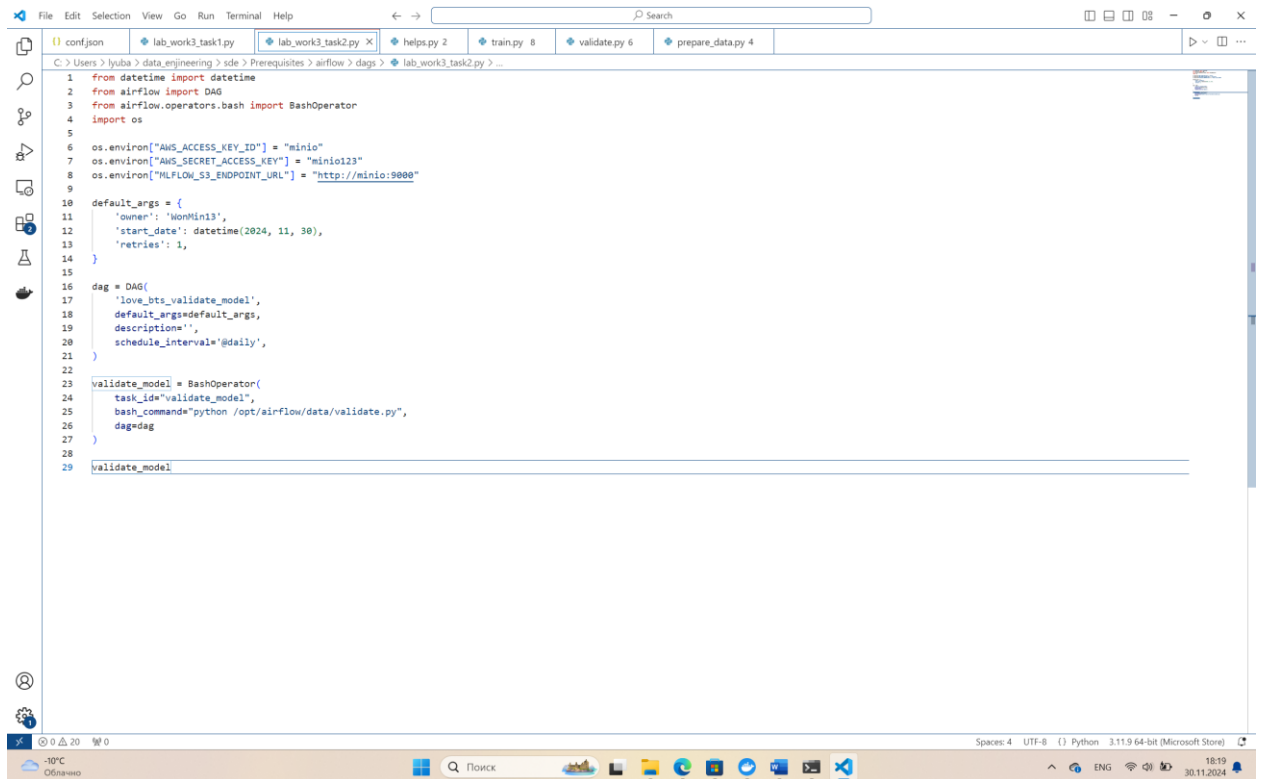
Рисунок 10 – Результат обучения моделей

В результате 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.

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



```
1 from datetime import datetime
2 from airflow import DAG
3 from airflow.operators.bash import BashOperator
4 import os
5
6 os.environ["AWS_ACCESS_KEY_ID"] = "minio"
7 os.environ["AWS_SECRET_ACCESS_KEY"] = "minio123"
8 os.environ["HMLFLOW_S3_ENDPOINT_URL"] = "http://minio:9000"
9
10 default_args = {
11     'owner': 'NonMin13',
12     'start_date': datetime(2024, 11, 30),
13     'retries': 1,
14 }
15
16 dag = DAG(
17     'love_bts_validate_model',
18     default_args=default_args,
19     description='',
20     schedule_interval='@daily',
21 )
22
23 validate_model = BashOperator(
24     task_id="validate_model",
25     bash_command="python /opt/airflow/data/validate.py",
26     dag=dag
27 )
28
29 validate_model
```

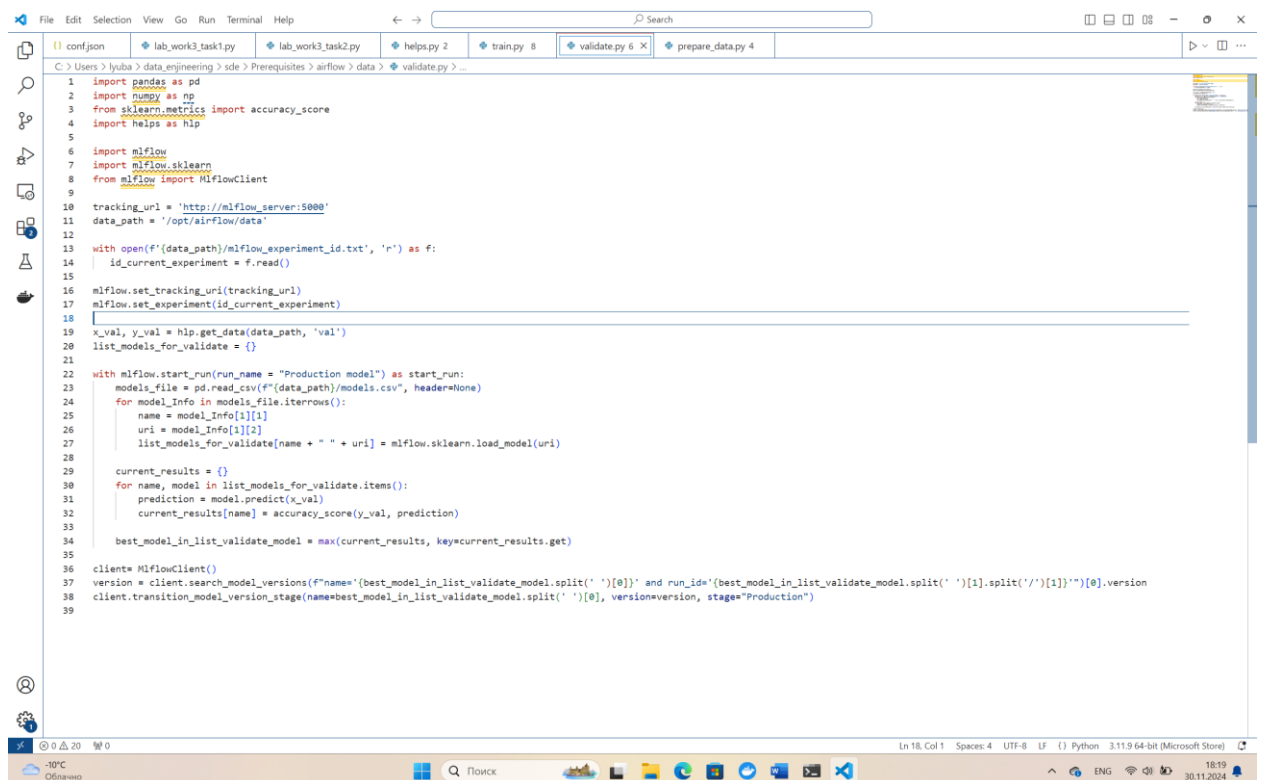
Рисунок 11 – DAG валидации моделей

## 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.



```
1 import pandas as pd
2 import numpy as np
3 from sklearn.metrics import accuracy_score
4 import heapq as hlp
5
6 import mlflow
7 import mlflow.sklearn
8 from mlflow import MlflowClient
9
10 tracking_url = 'http://mlflow_server:5000'
11 data_path = '/opt/airflow/data'
12
13 with open(f'{data_path}/mlflow_experiment_id.txt', 'r') as f:
14     id_current_experiment = f.read()
15
16 mlflow.set_tracking_uri(tracking_url)
17 mlflow.set_experiment(id_current_experiment)
18
19 x_val, y_val = hlp.get_data(data_path, 'val')
20 list_models_for_validate = {}
21
22 with mlflow.start_run(run_name = "Production model") as start_run:
23     models_file = pd.read_csv(f'{data_path}/models.csv', header=None)
24     for model_info in models_file.iterrows():
25         name = model_info[1][1]
26         uri = model_info[1][2]
27         list_models_for_validate[name + " " + uri] = mlflow.sklearn.load_model(uri)
28
29     current_results = {}
30     for name, model in list_models_for_validate.items():
31         prediction = model.predict(x_val)
32         current_results[name] = accuracy_score(y_val, prediction)
33
34     best_model_in_list_validate_model = max(current_results, key=current_results.get)
35
36 client = MlflowClient()
37 version = client.search_model_versions(f"name='{best_model_in_list_validate_model.split(' ')[0]}' and run_id='{best_model_in_list_validate_model.split(' ')[1].split('/')[1]}'")[0].version
38 client.transition_model_version_stage(name=best_model_in_list_validate_model.split(' ')[0], version=version, stage="Production")
39
```

Рисунок 12 – Код валидации моделей



### Шаг 3. Запуск DAG валидации моделей.

После окончания всех приготовлений перейдем в Airflow, для запуска DAG, который осуществит валидацию моделей.

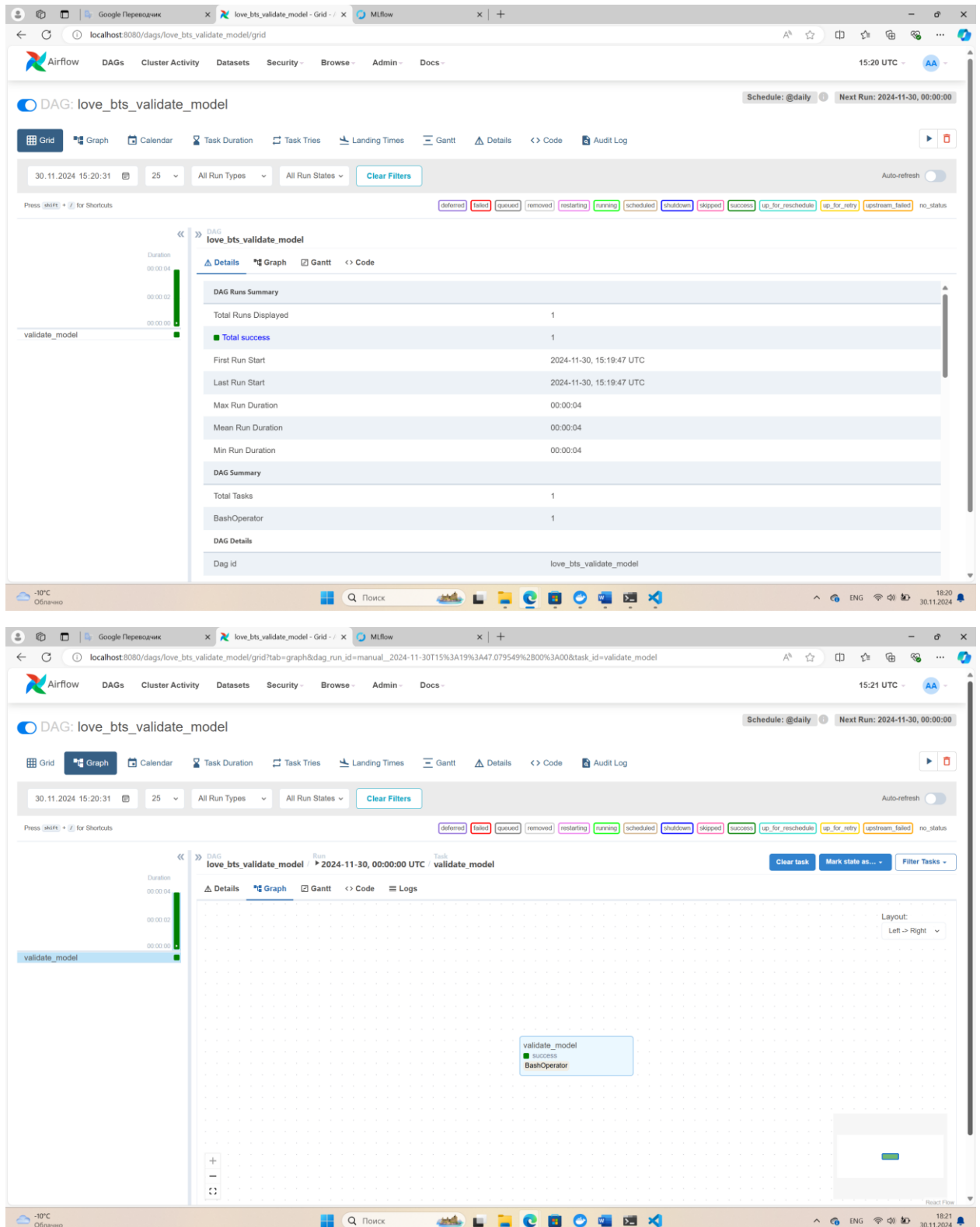
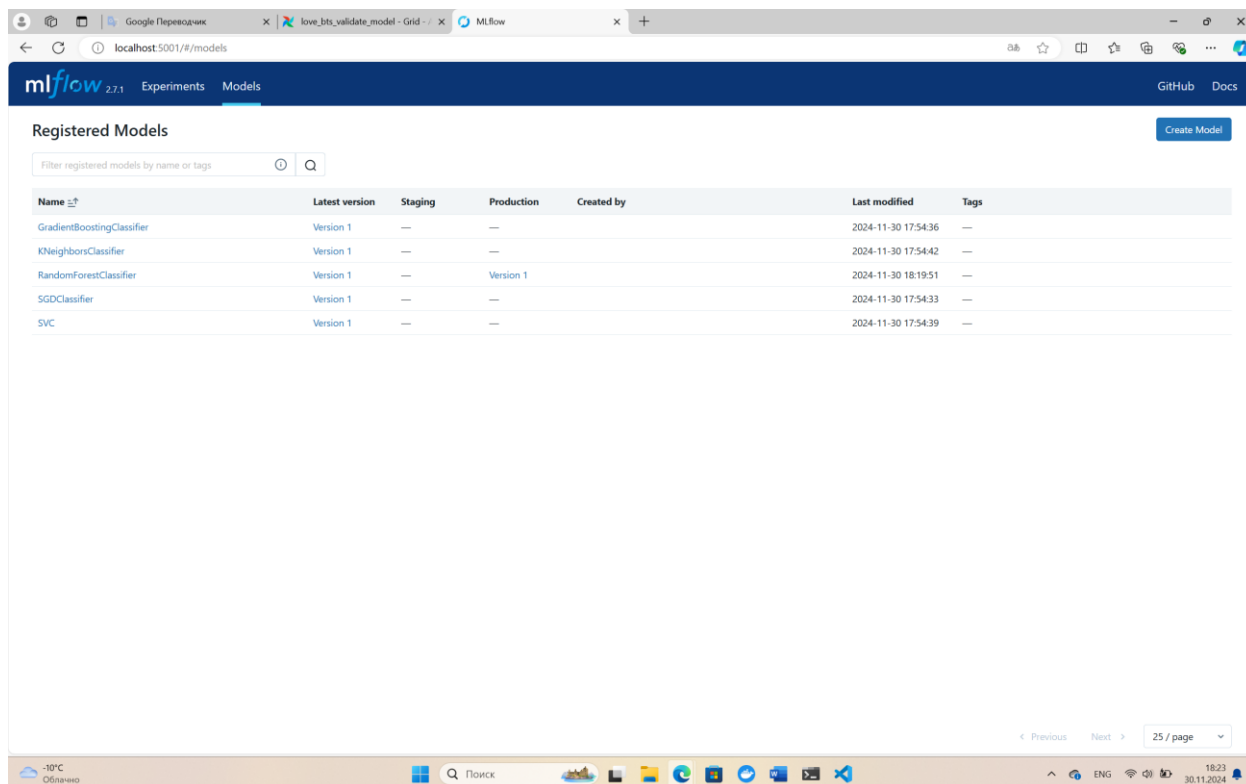


Рисунок 13 – Запуск DAG валидации моделей

#### Шаг 4. Проверка отработки DAG.

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



Name	Latest version	Staging	Production	Created by	Last modified	Tags
GradientBoostingClassifier	Version 1	—	—		2024-11-30 17:54:36	—
KNeighborsClassifier	Version 1	—	—		2024-11-30 17:54:42	—
RandomForestClassifier	Version 1	—	Version 1		2024-11-30 18:19:51	—
SGDClassifier	Version 1	—	—		2024-11-30 17:54:33	—
SVC	Version 1	—	—		2024-11-30 17:54:39	—

Рисунок 13 – Результат выполнения, DAG валидации моделей

## Заключение

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